Andrzej Bąk, Grażyna Dehnel, Andrzej Dudek,
Eugeniusz Gatnar, Krzysztof Kania, Marek Walesiak,
Łukasz Wawrowski, Artur Zaborski

Analiza danych z Banku Danych Lokalnych z wykorzystaniem programu R

Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu
Wrocław 2024

Poszczególne rozdziały napisali:

Rozdział 1 – Andrzej Dudek
Rozdział 2 – Krzysztof Kania
Rozdział 3 – Andrzej Dudek, Marek Walesiak
Rozdział 4 – Andrzej Dudek, Marek Walesiak
Rozdział 5 – Grażyna Dehnel, Łukasz Wawrowski
Rozdział 6 – Andrzej Bąk
Rozdział 7 – Marek Walesiak
Rozdział 8 – Eugeniusz Gatnar
Rozdział 9 – Artur Zaborski
Rozdział 10 – Marek Walesiak
Rozdział 11 – Grażyna Dehnel, Łukasz Wawrowski
Rozdział 12 – Grażyna Dehnel, Łukasz Wawrowski
Aneks – Łukasz Wawrowski

Recenzja
Paweł Lula

Redakcja wydawnicza
Agnieszka Flasińska

Korekta
Barbara Łopusiewicz

Opracowanie graficzne, skład i łamanie
Małgorzata Myszkowska

Projekt okładki
Beata Dębska

Na okładce wykorzystano zdjęcie z zasobów Adobe Stock

ISBN 978-83-67899-45-1

DOI: 10.15611/2024.45.1

Cytuj jako: Bąk, A., Dehnel, G., Dudek, A., Gatnar, E., Kania, K., Walesiak, M., Wawrowski, Ł. i Zaborski, A. (2024). *Analiza danych z Banku Danych Lokalnych z wykorzystaniem programu R*. Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu.

Andrzej Bąk ORCID: **0000-0002-5980-9694**
Grażyna Dehnel ORCID: **0000-0002-0072-9681**
Andrzej Dudek ORCID: **0000-0002-4943-8703**
Eugeniusz Gatnar ORCID: **0000-0003-2909-439X**
Marek Walesiak ORCID: **0000-0003-0922-2323**
Łukasz Wawrowski ORCID: **0000-0002-1201-5344**
Artur Zaborski ORCID: **0000-0003-1374-2268**

Spis treści

Wstęp

Bank Danych Lokalnych (BDL) udostępniany przez GUS od 1995 r. jest największą w Polsce bazą danych oferującą potencjalnym użytkownikom ponad 97 tys. zmiennych statystycznych pogrupowanych tematycznie. BDL to znane i cenione źródło danych statystycznych dla badaczy stosujących metody wielowymiarowej analizy statystycznej. Portal **https://bdl.stat.gov.pl/** umożliwia pobieranie danych w formacie csv oraz xls. Tak przygotowane dane mogą być wykorzystywane w programach statystycznych, w tym w środowisku R. Tradycyjny sposób pozyskiwania danych z BDL może być uciążliwy. Wadą tego rozwiązania jest narzut czasowy związany z zapisaniem danych i ich późniejszym importem z poziomu środowiska obliczeń statystycznych oraz konieczność wykonania eksportu/importu nawet dla czynności cyklicznych, powtarzalnych np. co kwartał czy co rok. Monografia skierowana jest do użytkowników Banku Danych Lokalnych oraz osób, które we własnym zakresie chcą się z nim zapoznać. Opis systemu ewidencjonowania danych lokalnych można znaleźć w podręczniku dostępnym na stronie **https://bdl.stat.gov.pl/bdl/pomoc**.

Na potrzeby monografii opracowano pakiet `bdl` działający w środowisku R. Monografia jest pierwszą pozycją naukową prezentującą automatyczne wykorzystanie danych z Banku Danych Lokalnych w analizach statystycznych wykorzystujących metody wielowymiarowe. Wiedza zawarta w monografii pozwoli użytkownikom metod statystycznej analizy wielowymiarowej na szybkie i przyjazne przeprowadzanie analiz dla danych statystycznych pozyskanych z BDL. Ponadto czytelnik otrzyma aktualną wiedzę z zakresu statystycznej analizy wielowymiarowej oraz wszystkie skrypty programu R.

W monografii zaprezentowano nowy sposób automatycznego pozyskiwania danych z Banku Danych Lokalnych z wykorzystaniem pakietu `bdl` oraz interfejsu API (*Application Programming Interface*) oraz wykorzystanie tak pozyskanych danych w zastosowaniu dla wybranych metod statystycznej analizy wielowymiarowej.

Monografia składa się z 12 rozdziałów. W rozdziałach 3-12 oprócz części teoretyczno-metodycznej przedstawione zostaną zastosowania z wykorzystaniem programu R dla danych z BDL.

W rozdziale 1 scharakteryzowano środowisko R jako środowisko obliczeń statystycznych, a jednocześnie język programowania działający w tym środowisku. Zaprezentowano także najpopularniejsze rozszerzenie tego środowiska: RStudio.

Rozdział 2 prezentuje funkcje dostępne w pakiecie `bdl` oraz przedstawia, jak pracować z pakietem, żeby uzyskać dane z BDL w programie R.

W rozdziale 3 przedstawiono podstawy wizualizacji danych oraz formaty plików graficznych w R, a następnie scharakteryzowano wykresy podstawowe oraz zaawansowane z wykorzystaniem programu R dla danych z BDL.

Rozdział 4 poświęcony jest wizualizacji danych na mapach. Scharakteryzowano formaty w GIS, zagadnienie wczytywania map oraz rysowania mapy i wycinka mapy, zagadnienie wyznaczania współrzędnych geograficznych. Następnie dla danych z BDL zaprezentowano zagadnienie wprowadzania danych, rysowania map (warstwy kolorystyczne dla regionów, przedziały klasowe, nanoszenie etykiet i legendy) jednowarstwowych i wielowarstwowych.

W rozdziale 5 przedstawiono teoretyczne podstawy dotyczące weryfikacji hipotez statystycznych oraz dokonano przeglądu wybranych testów. W ich doborze kierowano się częstością stosowania w praktyce badań. Dla każdego testu przygotowano przykład z wykorzystaniem programu R oraz danych pochodzących z BDL.

Rozdział 6 jest poświęcony zagadnieniu brakujących danych. Problem brakujących (niekompletnych) danych bardzo często pojawia się w badaniach statystycznych i ekonometrycznych, niezależnie od tego, czy dane pochodzą ze źródeł pierwotnych, czy wtórnych. W przypadku zaobserwowania brakujących danych można pominąć obiekty lub zmienne z brakującymi danymi albo imputować (uzupełniać) brakujące dane. W pracy podjęto próbę zastosowania drugiego podejścia, a więc uzupełniania brakujących danych z wykorzystaniem metod zaproponowanych w literaturze przedmiotu oraz pakietów programu R. Przykłady imputacji bazują na danych rzeczywistych dostępnych w BDL.

W rozdziale 7 przedstawiono rozwiązania metodyczne pozwalające na przeprowadzanie porządkowania liniowego. Zaprezentowano istotę i założenia porządkowania liniowego, klasyczne i nieklasyczne procedury porządkowania liniowego oraz miary agregatowe w porządkowaniu liniowym. Na tej bazie przedstawiono zastosowania porządkowania liniowego z wykorzystaniem programu R dla danych z BDL.

Rozdział 8 zawiera prezentację nieparametrycznej metody regresji wykorzystującej rekurencyjny podział przestrzeni zmiennych. Modele regresyjne tego typu mają łatwą do interpretacji graficzną postać drzew, a ich zaletą jest wykorzystanie w charakterze zmiennych objaśniających zarówno metrycznych, jak i niemetrycznych. W rozdziale pokazano budowę tej klasy modeli w programie R dla danych z BDL oraz ich interpretację.

W rozdziale 9 scharakteryzowano wybrane zagadnienia skalowania wielowymiarowego. Zaprezentowano istotę oraz procedurę skalowania wielowymiarowego, podstawy metodologiczne modeli różnic indywidualnych umożliwiających porównywanie badanych zjawisk w różnych okresach oraz analizę *unfolding*. W drugiej części przedstawiono zastosowanie wybranych metod skalowania wielowymiarowego z wykorzystaniem programu R dla danych importowanych z BDL.

Rozdział 10 prezentuje analizę skupień. Przedstawiono istotę oraz podstawowe problemy zagadnienia klasyfikacji. Następnie wyodrębniono i scharakteryzowano siedem etapów typowej analizy skupień, którymi są: wybór obiektów i zmiennych do klasyfikacji, wybór formuły normalizacji wartości zmiennych, wybór miary odległości, wybór metody klasyfikacji, ustalenie liczby klas, ocena wyników klasyfikacji, opis (interpretacja) i profilowanie klas. W drugiej części tego rozdziału przedstawiono podstawowe pakiety i funkcje programu R oraz zastosowania z wykorzystaniem programu R dla danych z BDL.

W rozdziale 11 omówiono podstawy podejścia modelowego stosowanego w przypadku danych panelowych. W opisie modeli uwzględniono najczęściej wykorzystywane testy służące do ich weryfikacji. W rozdziale zaprezentowano przykłady zastosowań modeli panelowych z użyciem programu R. Dane wejściowe pochodzące z BDL dotyczyły m.in. stopy bezrobocia oraz wynagrodzeń.

Rozdział 12 poświęcono statystyce małych obszarów (SMO). W pierwszej części zawarto teoretyczne podstawy estymacji bezpośredniej, a następnie przedstawiono wybrane zagadnienia z zakresu estymacji pośredniej. W prezentacji ograniczono się do estymatorów należących do podejścia randomizacyjnego oraz modelowego. Druga część rozdziału zawiera przykłady zastosowań metod SMO w analizie rynku pracy, z uwzględnieniem programu R oraz wykorzystaniem danych z BDL.

W aneksie przedstawiono charakterystykę pakietu `ggplot2`, który wykorzystywany jest w rozdziałach 5, 11 i 12 do prezentacji graficznej.

Autorzy mają nadzieję, że monografia okaże się przydatna dla badaczy i praktyków, którzy zajmują się problematyką analizy danych, klasyfikacji, dyskryminacji, modelowania, wizualizacji itp. Zainteresuje więc z pewnością ekonomistów, psychologów, socjologów, biologów, botaników, archeologów, lekarzy i innych.

Wersję instalacyjną programu R oraz dodatkowe pakiety można pobrać ze strony: **https://cran.r-project.org/.** Wszystkie zastosowania zawarte w monografii przetestowano, stosując wersję 4.3.3 programu R. Do monografii dołączono skrypty.

DOI: 10.15611/2024.45.1.01

1 Charakterystyka programu i języka R oraz środowiska RStudio[1]

1.1. Uwagi wstępne

R (R Core Team, 2023) to środowisko obliczeń statystycznych, a jednocześnie język programowania działający w tym środowisku. Twórcami jego pierwszej wersji byli Ross Ihaka oraz Robert Gentleman z University of Auckland. Wybór nazwy, oprócz zbieżności z inicjałem imion obu autorów, uzasadniony był tym, że zaprojektowany został jako bezpłatny odpowiednik języka S wykorzystywanego w komercyjnym oprogramowaniu S-PLUS. Od sierpnia 1997 r. jego rozwojem zajmuje się międzynarodowy zespół programistów zwany R Core Team. Wspiera go od strony organizacyjnej i finansowej The R Foundation, której przewodniczą Ihaka i Gentleman. Pierwsza oficjalna wersja (1.0.0) języka R opublikowana została 29 lutego 2000 r. Od tego czasu regularnie, kilka razy w roku, pojawiają się nowe wersje pakietu, od końca 2004 r. publikowane były wersje pakietu z numerem głównym 2, od kwietnia 2013 r. wersje pakietu numerowane były w postaci 3.<numer_wersji>.<numer_podwersji>, a od kwietnia 2020 r. w postaci 4.<numer_wersji>.<numer_podwersji>.

R może działać w systemach Linux, Windows i MacOS. Dla każdego z nich dostępne są binarne wersje instalacyjne na stronie domowej projektu w części CRAN (*The Comprehensive R Archive Network*). Dostępne są również jego pliki źródłowe i przy pewnej wprawie z narzędziami GNU można je skompilować i w ten sposób zainstalować program.

Wśród wielu aplikacji statystycznych R wyróżnia się kilkoma zaletami.

Jest dystrybuowany na licencji GPL, co oznacza, że można go za darmo wykorzystywać do dowolnych (nawet komercyjnych) celów (Kopczewska i in., 2016, s. 18).

Dzięki wbudowanemu mechanizmowi pakietów ma bardzo szeroką funkcjonalność. Oprócz procedur dostępnych w „czystym" języku R (w wersji 4.3.2) istnieje ponad 19 tys. pakietów zawierających w sobie praktycznie wszystkie istotne algorytmy wielowymiarowej analizy statystycznej. Wielu badaczy, proponując nowe metody, tworzy równocześnie w języku R oprogramowanie je realizujące.

Umożliwia swobodne łączenie różnych metod i tworzenie własnych procedur badawczych.

Umożliwia łączenie kodu języka R w sposób dynamiczny z bibliotekami napisanymi w języku C, co może znacznie przyśpieszyć wykonywanie obliczeń w newralgicznych punktach skomplikowanych algorytmów.

[1] Rozdział 1 jest zaktualizowaną, rozbudowaną i zmienioną wersją rozdziału 1 z pracy Dudka (2009, s. 13-61).

Ma bardzo dobrą dokumentację zarówno programu, jak i pakietów dodatkowych. Pliki pdf dołączone do pakietów, w tym dodatkowe winiety, mogą być często używane jako pełne źródło wiedzy o algorytmach. Wszystkie podręczniki użytkownika zawierają ponadto odwołania do najważniejszych pozycji z literatury przedmiotu, co czyni je doskonałymi punktami startowymi do pogłębionej analizy literaturowej przedmiotu.

Zawiera w sobie pokaźnych rozmiarów kolekcję rzeczywistych zbiorów danych oraz umożliwia generowanie danych sztucznych według bardzo wielu różnych kryteriów.

Potrafi importować i eksportować dane i wykresy w wielu popularnych formatach.

Ma możliwość bezpośredniego pobierania danych z wielu statystycznych baz danych, a nawet portali społecznościowych, takich jak X (dawniej Twitter) bezpośrednio poprzez sieciowe API (lub usługi sieciowe – *webservices*).

To wszystko sprawia, że R jest prawdziwym *lingua franca* statystyki XXI w.

Jedynym minusem środowiska R jest brak interfejsu graficznego, takiego jaki oferują komercyjne programy statystyczne. Po nabraniu pewnej wprawy w obsłudze programu brak „okienek" przestaje jednak przeszkadzać, natomiast coraz bardziej doceniana jest elastyczność – możliwość swobodnego łączenia metod, poprzez traktowanie danych wyjściowych z jednej procedury jako danych wejściowych do innej. Takiej swobody nie oferują zazwyczaj pakiety komercyjne, a cena, którą należy za to zapłacić, polegająca na nauczeniu się kilku funkcji służących do wyświetlania, edycji i zapisywania danych, nie wydaje się zbyt wygórowana. Zwłaszcza, że od wydania wersji 2.0 programu R popularność zaczęła zyskiwać nakładka RStudio (dystrybuowana jako osobny program, stworzony w języku C++ i bibliotece Qt), oferująca wiele rozszerzeń interfejsu graficznego środowiska. RStudio będzie zaprezentowane w dalszej części tego rozdziału.

Ponadto użytkownicy, którzy nie chcą korzystać z poleceń tekstowych, mogą pracować wyłącznie w trybie „okienkowym" z wykorzystaniem pakietów `Rcmdr` tworzącego dodatkowy panel zarządzający środowiskiem lub `shiny` umożliwiającego tworzenie własnych formularzy poprzez interfejs www.

1.2. Podstawy pracy w środowisku R

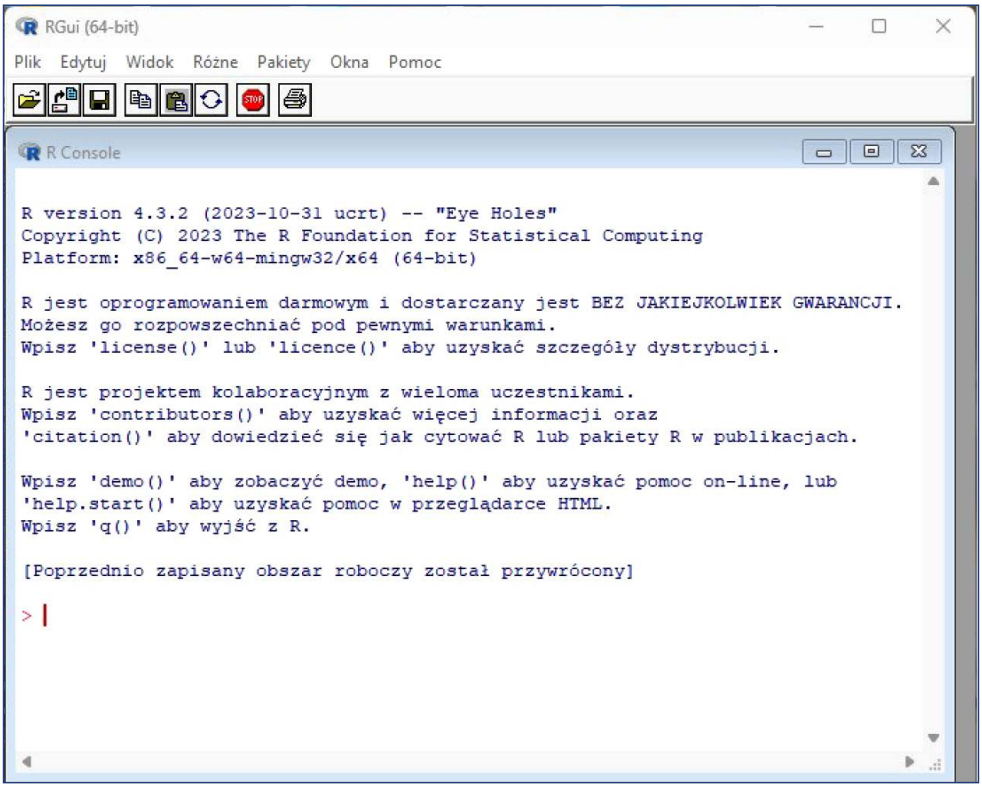
1.2.1. Tryby pracy w środowisku R

W środowisku R można pracować na trzy główne sposoby (i jeden „mieszany"). Wybór najbardziej efektywnego sposobu dla użytkownika zależy od jego doświadczenia oraz od stopnia skomplikowania zadania obliczeniowego.

a. Tryb interaktywny

Podstawowy i najczęściej wybierany przez początkujących użytkowników tryb pracy. Polega **na wprowadzaniu poleceń bezpośrednio w terminalu, w którym uruchomiony jest R (wydając polecenie R w wierszu poleceń systemu**) lub w oknie programu (tylko w systemie Windows – ta wersja

programu uruchamiana jest poprzez kliknięcie na ikonę na pulpicie lub w menu startowym systemu MS Windows). System R działający w osobnym oknie przedstawia rys. 1.1.



Rys. 1.1. Praca w programie R w osobnym oknie w systemie MS Windows

W tym trybie każde polecenie wydane przez użytkownika powoduje wykonanie odpowiednich wyliczeń i wyświetlenie ich wyniku, tak jak w poniższej przykładowej sesji (ostatnie polecenie oblicza silnię z liczby 10).

```
5+8
[1] 13
sin(pi/2)
[1] 1
log(2.718282)
[1] 1
sum(1, 4, 3, 6, 4, 3, 7, 2, 3, 4)
[1] 37
factorial(10)
[1] 3628800
```

Wyjątkiem jest sytuacja, kiedy używamy operatora podstawienia, w takim przypadku wartość wyrażenia zostaje zapamiętana w obiekcie, na który wskazuje ten operator. Wydanie polecenia:

```
wynik <- 2+2
```

lub alternatywnie, choć w wersji dużo rzadziej spotykanej,

```
2+2-> wynik
```

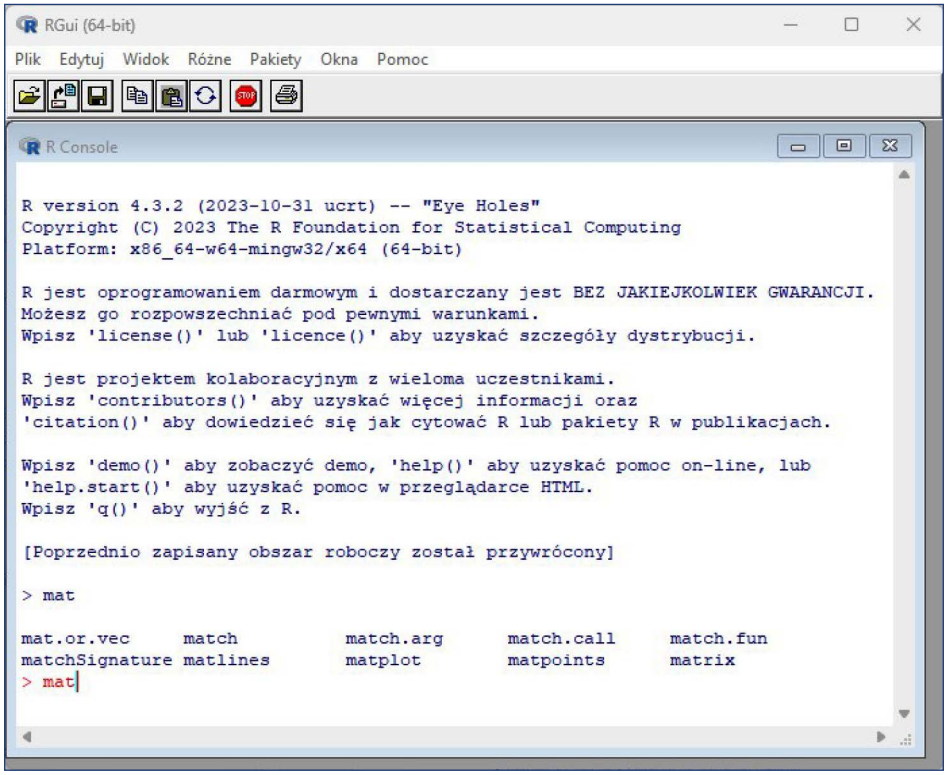
nie da żadnych widocznych rezultatów, poza zapisaniem rezultatu do zmiennej o nazwie `wynik`.

Również w przypadku używania w trybie interaktywnym złożonych konstrukcji programistycznych, takich jak pętle czy instrukcje warunkowe, wyniki instrukcji znajdujących się wewnątrz tych konstrukcji nie są wyświetlane.

Klawisze ⇑⇓ w trybie interaktywnym przywracają poprzednio wydane polecenie. Jeśli więc w trakcie ich pisania został popełniony błąd, to wystarczy wrócić do polecenia i poprawić błędnie wpisany fragment.

Jeśli wprowadzone zostanie polecenie niekompletne, bez nawiasu zamykającego lub cudzysłowu, czy niepełna instrukcja złożona `if`, `for`, `while`, to znak zachęty programu zmieni się na + i R będzie czekał na dokończenie instrukcji. Jeśli chcemy w takiej sytuacji przerwać wprowadzanie polecenia, można to zrobić, naciskając klawisz *Esc* (który służy również do przerwania aktualnie wykonywanych obliczeń, w przypadku dłużej trwających skryptów).

Kombinacja klawiszy *Ctrl+L* służy do wyczyszczenia zawartości aktualnego ekranu (równoważnie *Edytuj|Wyczyść konsolę* z menu). Klawisz tabulatora służy do uzupełniania nazw wydawanych poleceń, wystarczy wpisać ich początek, a program R uzupełni, o ile jest to możliwe, do prawidłowej nazwy funkcji. Naciśnięcie klawisza *Tab* podczas wpisywania polecenia wyświetla listę wszystkich funkcji języka rozpoczynających się od podanego fragmentu tekstu, a jeśli istnieje dokładnie jedna taka funkcja, to wprowadzany tekst jest automatycznie uzupełniany (por. rys. 1.2).



Rys. 1.2. Praca w programie R (RGui) w systemie MS Windows – działanie tabulatora

b. Tryb wsadowy

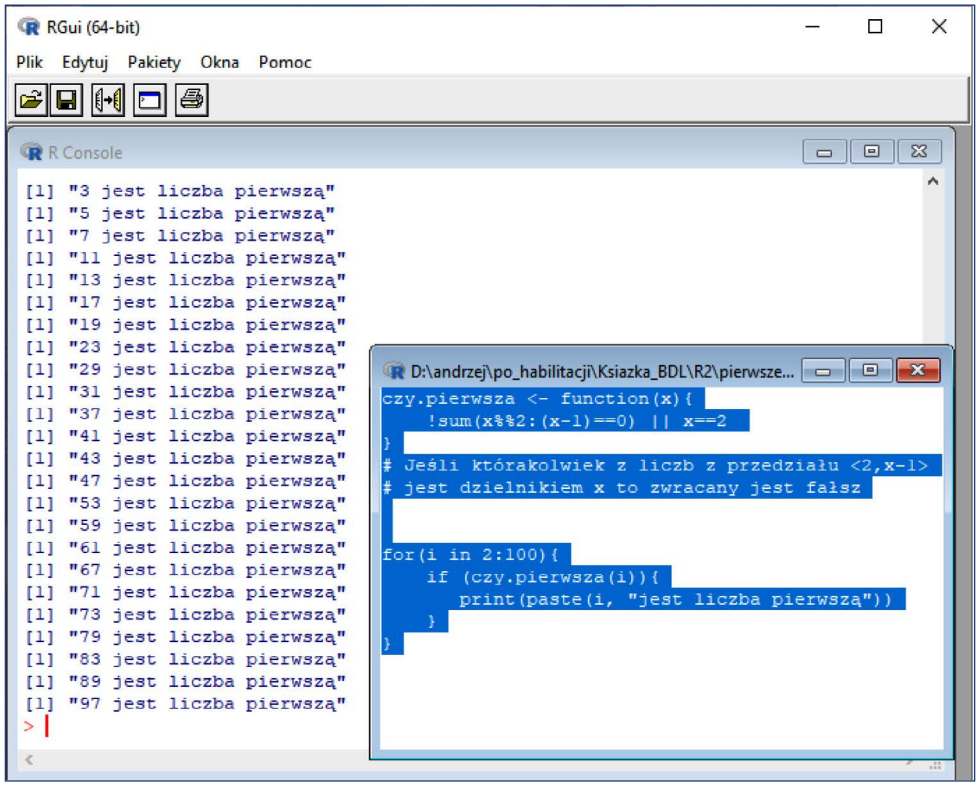
Jest to tryb, w którym skrypty języka R przygotowywane są w zewnętrznym edytorze i uruchamiane instrukcją `source ("nazwa_pliku.r")` lub przez polecenie menu *Plik|Źródłowy kod R …* . Jest to tryb „cichy", niewyświetlający wyników obliczeń cząstkowych. Funkcja `source` może być rów-

nież wywołana z argumentem `echo` ustawionym na `TRUE`, w takim przypadku, mimo uruchomienia skryptu w trybie wsadowym pokazywane są obliczenia cząstkowe, nawet te, które nie są bezpośrednio wyświetlane funkcją `print()`. Takiej sytuacji w programie RStudio odpowiada polecenie `Source with echo`.

Do wyświetlenia wyników skrypty najczęściej wykorzystują funkcję `print()`.

c. Tryb mieszany – korzystanie z edytora wewnętrznego środowiska R

Wewnętrzny edytor systemu R dostępny poprzez polecenia menu *Plik|Nowy skrypt* i *Plik|Otwórz skrypt* jest uboższy niż większość nawet najprostszych samodzielnych edytorów zewnętrznych. Niewielka ilość opcji edycyjnych jest w nim zrekompensowana przez silniejsze zintegrowanie z samym środowiskiem i możliwość uruchamiania skryptów z wnętrza edytora, bez konieczności używania polecenia *Plik|Źródłowy kod R …* . Służy do tego ikona *Uruchom linijkę lub zaznaczenie* (trzecia od lewej, zaznaczona na rys. 1.3), która powoduje przekopiowanie zaznaczonego w oknie edycji fragmentu kodu do okna konsoli R i uruchomienie go. Przy tym, podobnie jak w trybie interaktywnym, wyniki obliczeń pojawiają się natychmiast w oknie konsoli R (bez konieczności używania funkcji `print`). Ten tryb jest o tyle wygodny, że umożliwia uruchamianie tylko wybranych fragmentów skryptu oraz nie wymaga zapisywania zmian w skrypcie przed jego uruchomieniem. Jednocześnie nie oferuje praktycznie żadnych ułatwień edycyjnych (numeracji linii, automatycznego zamykania cudzysłowów i nawiasów czy sprawdzania składni). Wydaje się więc, że ten sposób pracy jest najwygodniejszy do edycji i uruchamiania skryptów o niezbyt dużych rozmiarach (kilkanaście lub kilkadziesiąt linii kodu).



Rys. 1.3. Uruchomienie skryptu w edytorze wewnętrznym środowiska R

d. Tryb „nadzorowany"

Tryb, w którym użytkownik nie wprowadza bezpośrednio poleceń R, ale korzysta z menu udostępnianego przez jeden z pakietów zarządzających, takich jak `Rcmdr`.

Funkcja `print`

Podczas pracy w trybie interaktywnym wyniki obliczeń pojawiają się natychmiast w oknie programu R. Inaczej jest przy uruchamianiu skryptów zewnętrznych lub po wyborze polecenia `source` w RStudio. W takim przypadku w konsoli środowiska R pojawiają się tylko te wartości wyrażeń, które zostaną wprost wskazane przez programistę za pomocą funkcji `print()`. W poniższym przykładzie funkcja `print` wyświetli wartość zmiennej `wynik`, czyli `sinus(pi/6)`

```
wynik <- sin(pi/6)
print(wynik)
[1] 0.5
```

Za pomocą funkcji `print` można zdefiniować sposób wyświetlania liczb, określić liczbę znaczących cyfr (argument `digits`), zdefiniować, czy tekst ma być wyświetlony w cudzysłowach (argument `quote`), oraz określić sposób wyświetlania wartości niedostępnych (`na.print`) czy zer (`zero.print`).

```
print(sin(60*pi/180),digits=11)
[1] 0.86602540378
print("Napis w cudzysłowie", quote=TRUE)
[1] "Napis w cudzysłowie"
print("Napis bez cudzysłowu", quote=FALSE)
[1] Napis bez cudzysłowu
print.default(c(1,3,NA,5,3,NA,NA,14), na.print="Brak wartości")
[1] 1 3 Brak wartości 5 3 Brak wartości Brak wartości 14
```

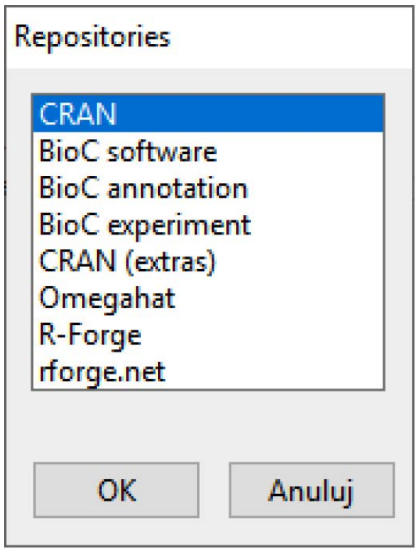
Dla niektórych typów danych, takich tak `tibble` w poleceniu `print`, można ponadto zdefiniować liczbę wyświetlanych wierszy tabeli danych. Przy wydrukach łączących stałe tekstowe z wartościami zmiennych funkcja `print` często jest używana razem z funkcją `paste` omówioną w dalszej części rozdziału.

1.2.2. Pakiety

„Nie pytaj się, czy coś można w R zrobić, pytaj się jak?". To zdanie, w oryginale dotyczące systemu Unix, dobrze oddaje wielość możliwości i zaimplementowanych algorytmów środowiska R. Siłą programu jest rozmaitość procedur i metod przez niego oferowanych. Zdecydowana większość znanych algorytmów wielowymiarowej analizy statystycznej jest w języku R oprogramowana, choć ze względu na ich różnorodność i ciągły rozwój nie wszystkie znajdują się w jądrze (*core*) systemu, a wiele umieszczonych jest w dodatkowych pakietach.

W wersji 4.3.x system R posiada 20243 pakietów opublikowanych na CRAN (stan na 15 stycznia 2024 r.).

Zdecydowana większość pakietów umieszczona jest na serwerach CRAN, choć niektóre umieszczone są w innych repozytoriach, a nawet serwerach systemu kontroli wersji GIT, takich jak github, gitlab czy bitbucket. Zmiana repozytoriów, z których R korzysta w trakcie sesji, odbywa się poprzez polecenie menu *Pakiety*|*Wybierz repozytoria…* (zob. rys. 1.4).

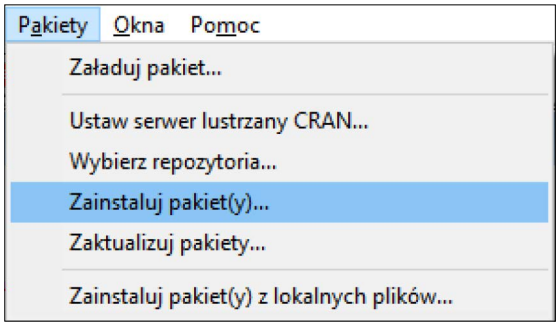


Rys. 1.4. Wybór repozytoriów pakietów w środowisku R

Do większości zastosowań wystarczy wybór dwóch pierwszych repozytoriów. Początkujący użytkownicy mogą jednak zaznaczyć wszystkie pozycje i mieć wybór z pakietów znajdujących się we wszystkich repozytoriach. Używanie funkcji z pakietów wymaga dwóch czynności: zainstalowania pakietu w systemie i załadowania go do pamięci operacyjnej.

1. Zainstalowanie pakietu w systemie jest czynnością jednorazową. Pakiet raz zainstalowany w systemie pozostaje w nim na zawsze. Instalacji pakietu/ów można wykonać za pomocą funkcji `install.packages()`

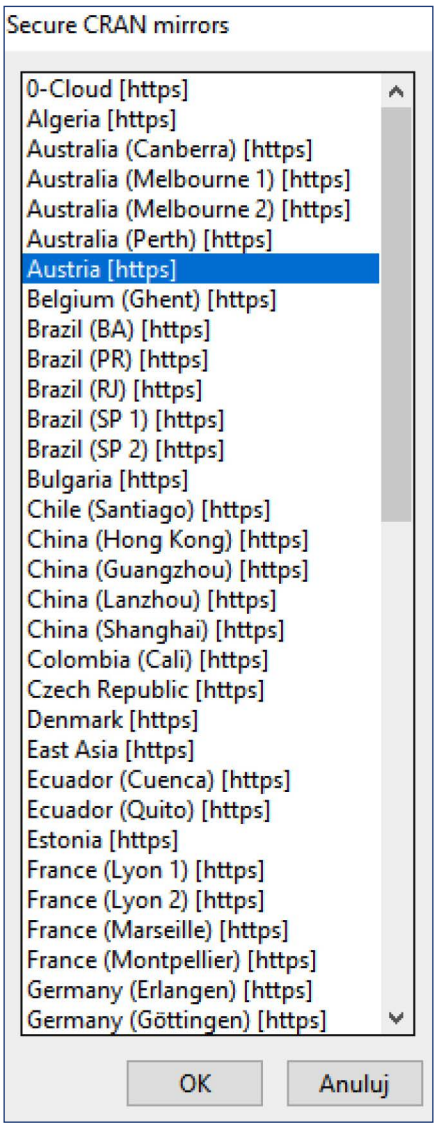
```
install.packages(c("bdl","keras"))
Instalowanie pakietów w ‘C:/Users/andrzej/Documents/R/win-library/4.3’
(ponieważ ‘lib’ nie jest określony)
--- Proszę wybrać serwer lustrzany CRAN do użycia w tej sesji ---
instalowanie dodatkowych zależności ‘config’, ‘reticulate’, ‘tensorflow’, ‘tfruns’
---
pakiet ‘bdl’ został pomyślnie rozpakowany oraz sumy MD5 zostały sprawdzone
pakiet ‘keras’ został pomyślnie rozpakowany oraz sumy MD5 zostały sprawdzone

Pobrane pakiety binarne są w C:\Users\andrzej\AppData\Local\Temp\RtmpiCANKl\
downloaded_packages
```



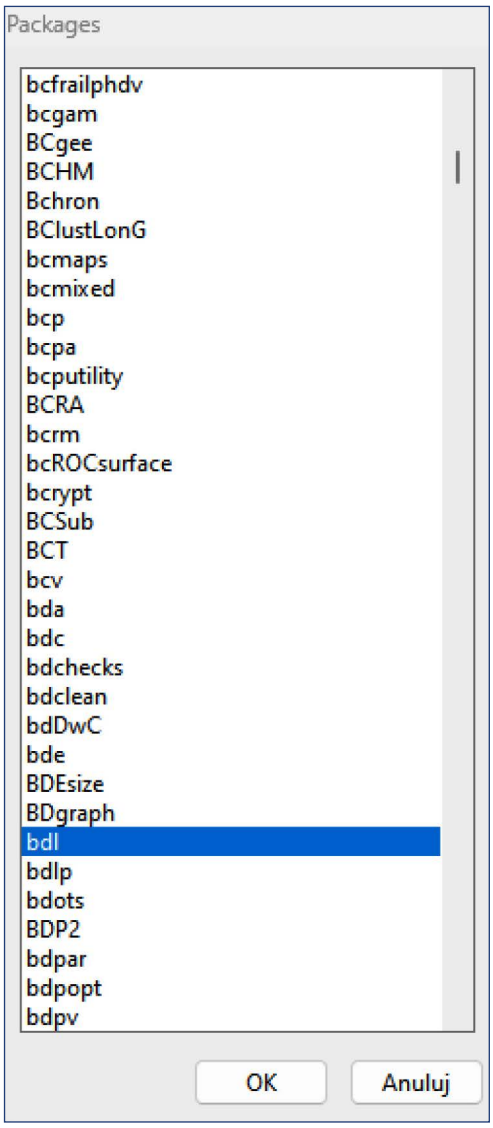
Rys. 1.5. Instalacja pakietów w środowisku R

Alternatywnym sposobem instalacji pakietu w systemie Windows jest wybranie z menu polecenia *Pakiety|Zainstaluj pakiet(y)* (rys. 1.5).

W kolejnym kroku należy dokonać wyboru serwera, z którego pakiet powinien zostać pobrany. Serwer ten będzie używany do końca sesji, chyba że zostanie zmieniony poleceniem menu *Pakiety|Ustaw serwer lustrzany CRAN…* (rys. 1.6). Kolejną czynnością jest wybór samego pakietu (rys. 1.7).



Rys. 1.6. Wybór serwera lustrzanego CRAN



Rys. 1.7. Wybór pakietu `bdl` do instalacji w środowisku R

Czasami, choć coraz rzadziej, zdarza się, że autorzy pakietów nie udostępniają ich poprzez CRAN czy inne repozytorium sieciowe. W takiej sytuacji pakiet należy pobrać z witryny internetowej autora(ów), a procedura instalacji pakietu przebiega trochę inaczej:

a) należy pobrać plik `<nazwa_pakietu_i_wersji>.zip` ze strony domowej autora/autorów i zapisać w dowolnym katalogu na dysku lokalnym,
b) wybrać z menu polecenie *Pakiety|Zainstaluj pakiet(y) z lokalnych plików…* i wskazać pliki zawierające pakiet.

2. W przeciwieństwie do instalacji pakietu jego załadowanie do pamięci operacyjnej powinno być wykonane na początku każdej sesji, w której wykorzystywane będą dane i funkcje znajdujące się w pakiecie. Służy do tego polecenie menu *Pakiety|Załaduj pakiet…* lub funkcja `library`. Funkcje z pakietu nie są dostępne przed jego załadowaniem, a próba skorzystania z nich kończy się komunikatem o błędzie, tak jak w poniższym przykładzie:

```
x <- data.Normalization(c(1,3,4,2,3), "n1")
Error: could not find function "data.Normalization"
library(clusterSim)
Loading  required package: ade4
Loading  required package: cluster
Loading  required package: R2HTML
Loading  required package: e1071
Loading  required package: class
x <- data.Normalization(c(1,3,4,2,3), "n1")
print(x)
[1] -2.4032928 0.3508232 2.2278812 -0.5262348 0.3508232
```

Listę pakietów **zainstalowanych** w systemie wraz z krótkim ich opisem zwraca polecenie `library()`.

Listę pakietów **załadowanych** do pamięci operacyjnej zwraca funkcja `search()`:

```
search()
 [1]  ".GlobalEnv"          "package:clusterSim"   "package:e1071"
 [4]  "package:class"       "package:R2HTML"       "package:cluster"
 [7]  "package:ade4"        "package:stats"        "package:graphics"
[10]  "package:grDevices"   "package:utils"        "package:datasets"
[13]  "package:methods"     "Autoloads"            "package:base"
```

Do trwałego usunięcia zainstalowanego pakietu służy polecenie `detach("package:nazwa_pakietu")`. Wyjątkiem jest pakiet `base`, którego nie można usunąć z instalacji programu R.

```
detach("package:base")
Error in detach(pos): detaching "package:base" is not allowed
```

Nie ma potrzeby „wyładowywania" pakietów z pamięci, gdyż jest to robione automatycznie po każdym zakończeniu sesji programu R lub po wydaniu polecenia *Różne|usuń wszystkie obiekty.*

Funkcja `help(package="<nazwa_pakietu>")` wyświetla dokładniejsze informacje o pakiecie, jego wersji, zależnościach z innymi pakietami, zawartości i autorach.

Jedną z informacji wyświetlanych przez funkcję `help(package=…)` jest wersja pakietu. Zmiana wersji pakietu nie jest w żaden sposób powiązana ze zmianą wersji samego środowiska R (którą możemy odczytać z menu *Pomoc|O programie* lub pisząc `version()`). Co więcej, z uwagi na dość długi czas pomiędzy kolejnymi wersjami programu R bardzo często zdarza się, że pomiędzy kolejnymi wersjami całego środowiska pojawia się zwykle kilka wersji pakietu. Polecenie menu *Pakiety|Zaktualizuj pakiety …* pozwala zaktualizować wszystkie będące w systemie pakiety.

1.2.3. System pomocy

Brak okienek i poleceń pogrupowanych w menu w środowisku R jest rekompensowany bardzo przyjaznym i elastycznym systemem pomocy. Jego elementy wymieniono poniżej.

1. Podręczniki użytkownika programu R zarówno dla początkujących, jak i dla doświadczonych programistów. Wszystkie podręczniki znajdują się w podkatalogu *\doc\manual* katalogu instalacyjnego środowiska R. Dokumentacja funkcji z pakietów wchodzących w skład jądra (*core*) systemu jest dostępna w pliku `refman.pdf`. Pliki `R-intro.pdf` i `R-lang.pdf` zawierają wstęp do programowania w R. `R-ext.pdf` to podręcznik tworzenia własnych pakietów.

`R-admin.pdf` zawiera opis instalacji i administracji programu R w systemach Linux, MacOS i MS Windows. `R-data.pdf` to instrukcja importu i eksportu danych z/do R. `R-ints.pdf` to zaawansowany podręcznik programistyczny dotyczący funkcji wewnętrznych systemu przeznaczony dla członków R Core Team oraz doświadczonych programistów języka R. Wreszcie podręcznik `fullrefman.pdf` zawiera pełną dokumentację pakietów wchodzących w skład jądra systemu. W systemie MS Windows wszystkie podręczniki są dostępne w menu *Pomoc|Podręczniki (w PDF)*.

2. Pomoc do poszczególnych funkcji dostępna poprzez polecenie `?<nazwa_funkcji>` lub `help("<nazwa_funkcji>")`. W wersjach 3.xx i 4.xx programu pomoc otwiera się w przeglądarce internetowej (w wersjach 2.xx było to odrębne okno programu).

Standardowa strona podręcznika dotycząca wybranej funkcji zawiera sekcje opisujące:

- nazwę (lub nazwy) i opis funkcji,
- sposób użycia funkcji,
- listę argumentów wraz z opisem ich znaczenia,
- opis zwracanych przez funkcję wartości,
- szczegóły dotyczące argumentów i zwracanych wartości funkcji oraz algorytmu w niej zastosowanego,
- dane adresowe autora(ów) funkcji,
- najważniejsze pozycje literatury związanej z zakresem tematycznym funkcji (ten fragment systemu pomocy wydaje się szczególnie cenny),
- odnośniki do stron pomocy pokrewnych funkcji,
- przykłady prawidłowych wywołań funkcji z różnymi argumentami,
- podpowiedź dotycząca argumentów poszczególnych funkcji.

3. Szybka podpowiedź dotycząca tylko argumentów wybranej funkcji, często wykorzystywana, w przypadku gdy użytkownik zapomni dokładnego sposobu przekazywania argumentów do funkcji, którą już zna, dostępna jest za pomocą polecenia `args`.

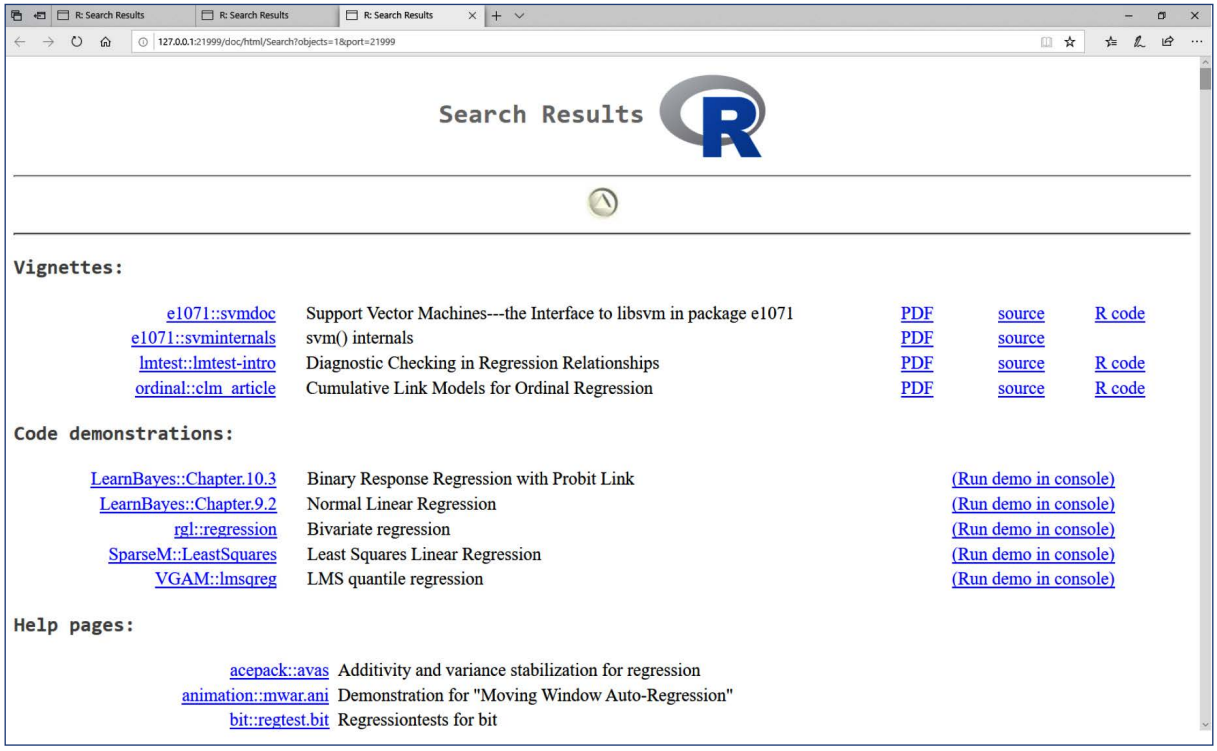
```
args(cluster.Sim)
function (x, p=1, minClusterNo, maxClusterNo, icq="S",
          outputHtml="", outputCsv="", outputCsv2="",
          normalizations=NULL, distances=NULL, methods=NULL)
```

4. Wyświetlenie i uruchomienie przykładów związanych z funkcją za pomocą polecenia `example`.
5. Wyszukiwanie w systemie pomocy (rys. 1.8). Polecenia `?` i `args` działają tylko dla pakietów załadowanych do pamięci operacyjnej. W celu uzyskania pomocy dla funkcji z pakietu, który jest zainstalowany w systemie, ale nie jest załadowany do pamięci, należy wydać polecenie

```
help("<nazwa_funkcji>", package="<nazwa_pakietu>") jak w przykładzie:
help("optSmacofSym_mMDS", package="mdsOpt")
```

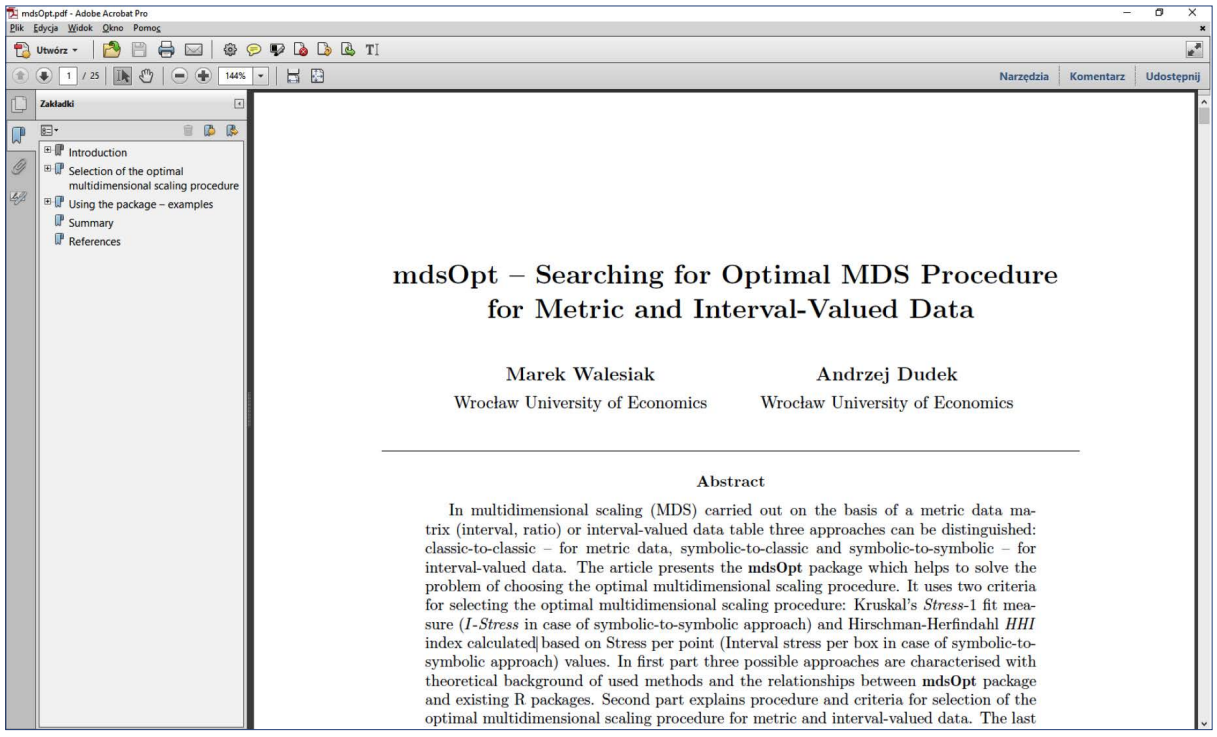
Do przeszukania systemu pomocy wszystkich zainstalowanych w środowisku R pakietów najwygodniej jest użyć funkcji `help.search(„fraza do wyszukania”)`, która wskaże pakiety i funkcje zawierające podaną frazę. Przykładowo, szukając funkcji związanych z regresją, należy wydać polecenie:

```
help.search("regression")
```



Rys. 1.8. Wyszukiwanie w systemie pomocy z pakietów zainstalowanych w środowisku R

6. Dodatkowe artykuły i opracowania objaśniające merytoryczne podstawy pakietu dostępne dla wielu pakietów poprzez polecenie `vignette(nazwa załadowanego pakietu)` lub jako odnośnik z okna pomocy pakietu `vignette("mdsOpt") (rys 1.9).`



Rys. 1.9. Winieta pakietu `mdsOpt`

7. Dla doświadczonych programistów cennym źródłem informacji może być zapis algorytmu w języku R. System umożliwi obejrzenie kodu źródłowego dowolnej funkcji poprzez podanie nazwy funkcji (bez argumentów i znaku ()).

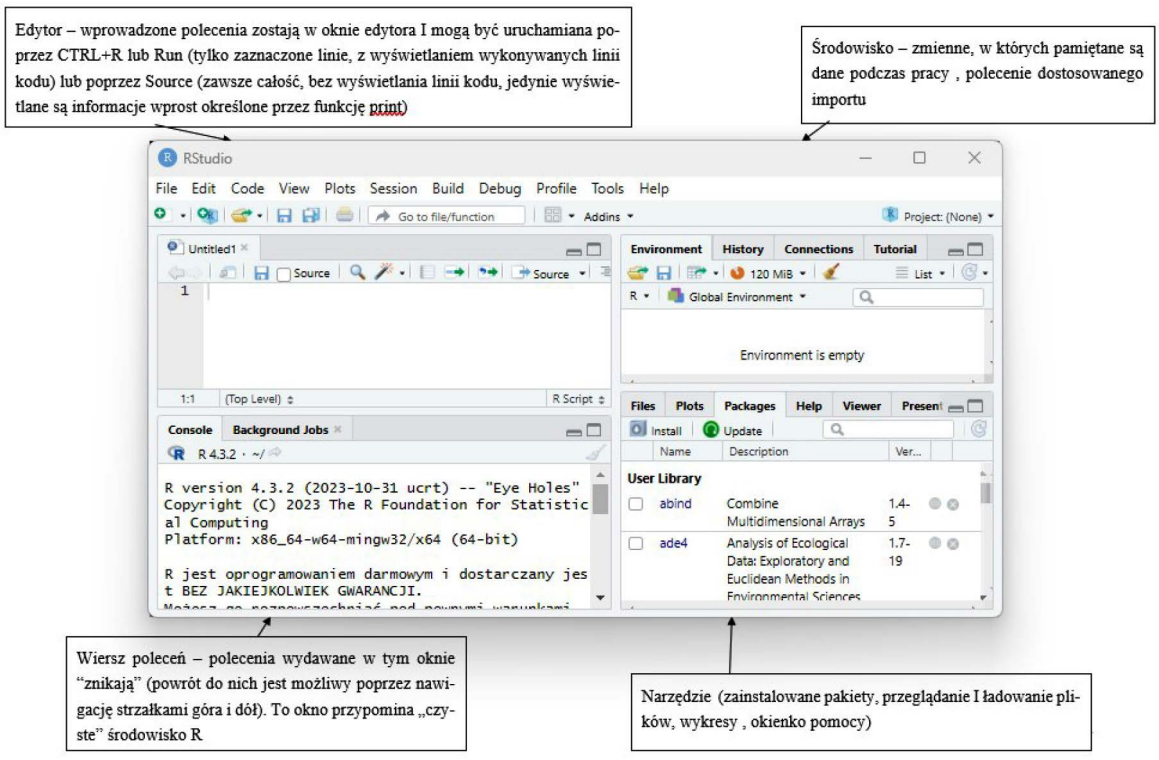
Zmiana języka w programie RGui

Domyślnym językiem edytora RGui jest język zdefiniowany w systemie operacyjnym lub angielski. Program ten ma dodatkowo możliwość wyboru języka. W systemie MS Windows ustalenie języka odbywa się poprzez edycję pliku *<katalog instalacyjny R>*, np. *c:\Program Files\R\R-3.6.0 > \etc\ Rconsole* (edycja tego pliku wymaga zazwyczaj uprawnień administratora systemu MS Windows). W linii *language* = należy podać kod języka. Przykładowo wpis *language* = *pt_BR* zmienia język na portugalski w odmianie brazylijskiej. Po zmianie języka wymagane jest zrestartowanie edytora RGui.

Listę aktualnych rozszerzeń językowych można znaleźć w folderze *<katalog instalacyjny R>\library\ translations*. Niektóre rozszerzenia dla języków (ko – koreański, ja – japoński, zn – chiński) bazujących na innym alfabecie niż łaciński wymagają dodatkowej konfiguracji systemu MS Windows lub działają tylko w odpowiedniej wersji językowej tego systemu.

1.3. Środowisko RStudio

RStudio jest zintegrowanym środowiskiem programistycznym (*IDE – Integrated Development Environment*) opartym na programie R, wprowadzającym wiele rozszerzeń i ułatwień dla użytkowników tego ekosystemu. W wersji bazowej (bez wsparcia technicznego) RStudio jest dystrybuowane na licencji freeware, choć istnieją też komercyjne wersje tego oprogramowania.

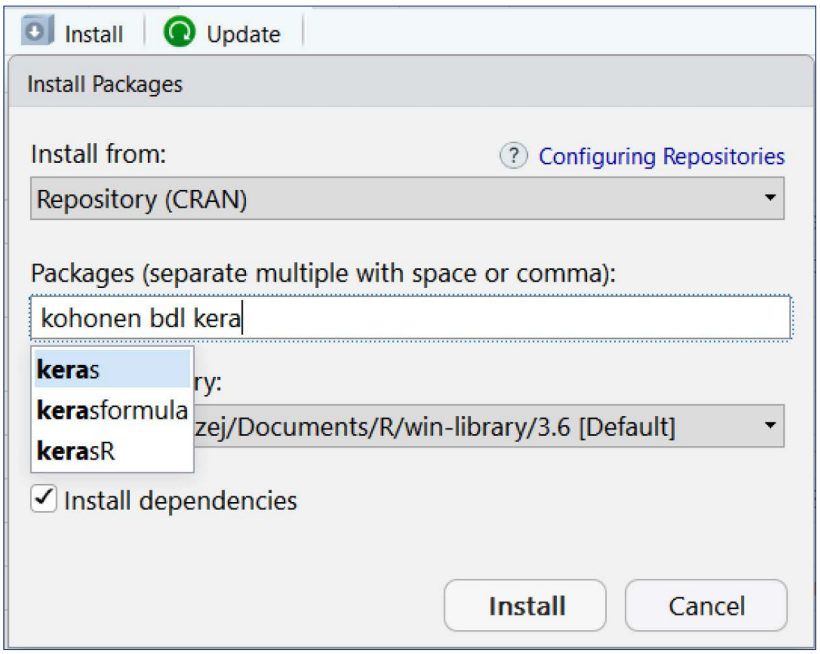


Rys. 1.10. Okno główne programu RStudio

Okno główne środowiska RStudio (rys. 1.10) podzielone jest na cztery panele: edytora skryptów języka ze skryptami otwartymi w osobnych zakładkach, konsoli R, panelu zmiennych środowiskowych z trzema zakładkami: środowiska, historii i połączenia z zewnętrznymi bazami danych oraz panelu narzędziowego z pięcioma zakładkami: plików, wykresów, pakietów, pomocy i dodatkowego podglądu lokalnych stron html.

Wśród licznych ułatwień programu RStudio wymienić można te najważniejsze (poniższy wybór ma charakter subiektywny, wynikający z doświadczeń autorów).

1. Autouzupełnianie treści poleceń i kodu języka R po naciśnięciu skrótu klawiaturowego *Ctrl+spacja*, a w przypadku listy argumentów funkcji poprzez *Ctrl+Shift+spacja*.
2. Możliwość pracy na wielu otwartych skryptach języka R – każdy z nich może być otwarty w innej zakładce panelu edycyjnego.
3. Podgląd zmiennych utworzonych podczas trwania sesji programu w panelu środowiska.
4. Możliwość uruchamiania kodu z okna edycyjnego fragmentami Run (*Ctrl+Enter*), w trybie wsadowym Source (*Ctrl+Shift+S*), w trybie wsadowym z pokazywaniem poleceń Source with Echo (*Ctrl+Shift+Enter*).
5. Instalacja jednego lub więcej pakietów z podpowiedzią przy wyborze po podaniu początku nazwy (rys. 1.11).



Rys. 1.11. Wybór pakietów do instalacji w programie RStudio

6. Możliwość instalacji pakietów, dla których nie ma jeszcze wersji przeznaczonych dla konkretnego systemu operacyjnego z ich plików źródłowych. Wymaga to zainstalowania w przypadku MS Windows narzędzi RTOOLS (**https://cran.r-project.org/bin/windows/Rtools/**). Jeżeli pakiet nie znajduje się na CRAN czy innym repozytorium sieciowym, przy instalacji należy wybrać opcję *Install From*|*Package Archive File*.
7. Możliwość ładowania do pamięci, wyładowania z pamięci i instalacji pakietów przyciskami clusterSim oraz 0.48-2.

8. Możliwość szybkiej, wizualnej zmiany katalogu roboczego poleceniami *Session|Set working directory|To …* oraz w panelu przeglądania plików poleceniami *More|Set as working directory/ Go to working directory*.
9. Możliwość bezpośredniego ładowania do środowiska skryptów, zbiorów z danymi, plików `rda` itp. w okienku przeglądania plików.
10. Możliwość zatrzymywania i analizowania skryptów R poprzez wbudowany `debugger` i `profiler`.
11. Automatyczne kolorowanie błędów rozpoznanych przez program.
12. Automatyczne kolorowanie nawiasów otwierających przy zaznaczeniu nawiasu zamykającego i odwrotnie.
13. Możliwość automatycznego formatowania kodu i ustawiania odpowiednich wcięć zwiększających jego czytelność – *Code|Reformat code* (*Ctrl+Shift+A*).

Z kolei za (niewielkie) minusy pracy z tym środowiskiem uznać można brak polskiej wersji językowej (co zwłaszcza w połączeniu z polską wersją samego programu R skutkuje komunikatami o błędach pisanymi w jednym zdaniu na przemian po polsku i po angielsku), a także małymi rozmiarami poszczególnych paneli, zwłaszcza podczas pokazów na projektorach o rozdzielczości mniejszej niż Full HD.

1.4. Podstawy języka R

1.4.1. Literały

W tabeli 1.1 przedstawiono literały (stałe dosłowne reprezentujące dane) występujące w środowisku R.

Tabela 1.1. Literały w języku R

Literał	Przykład	Znaczenie
`"<ciąg znaków>"`	`"Komentarz"`	Literał znakowy
`TRUE, T`		Prawda (stała logiczna)
`FALSE, F`		Fałsz (stała logiczna)
`NULL`		Literał oznaczający obiekt pusty (np. łańcuch znakowy niezawierający żadnego znaku)
`NA`		Literał oznaczający, iż dla danego obiektu/zmiennej nie jest określona związana z nim wartość
`NaN`		Literał oznaczający, że przy obliczaniu wartości obiektu/zmiennej wystąpił błąd numeryczny (np. dzielenie przez zero lub pierwiastkowanie liczby ujemnej)
`inf`		Literał oznaczający nieskończoność, otrzymaną np. przy dzieleniu liczby przez 0
`<Ciąg cyfr>`	`1`	Literał numeryczny bez podtypu
`<Ciąg cyfr>L`	`20L`	Literał numeryczny podtyp całkowity
`<ciąg cyfr>.<ciąg cyfr>`	`2.5`	Literał numeryczny podtyp zmiennopozycyjny (liczby rzeczywiste)
`0x<ciąg cyfr>`	`0x18`	Literał numeryczny, podtyp całkowity w systemie heksadecymalnym (0x18 odpowiada 24L)
`<ciąg cyfr>e<ciąg cyfr>`	`1e3`	Literał numeryczny, podtyp całkowity w zapisie naukowym (1e3 odpowiada 1*10^3, czyli 1000)
`pi`	`print(pi, digits=15)`	Stała π (w tym przypadku 3,141592653589790)

Źródło: opracowanie własne.

1.4.2. Komentarz

Znak # oznacza w języku R komentarz. Wszystkie instrukcje występujące od pierwszego wystąpienia symbolu # do końca wiersza nie są przez R interpretowane. Podobnie jak w innych językach programowania komentarzy, najczęściej wykorzystuje się do objaśniania bardziej złożonych fragmentów kodu, zwłaszcza w trybie wsadowym.

```
for(i in 1:liczbaNosnikow){# Dobór zmiennych do modelu metodą
# pojemności nośników informacji profesora Hellwiga
     t <- i
     kombinacja=c()
     for(j in 1:liczbaKandydatek){
         if((t %% 2)==1) kombinacja<-cbind(kombinacja, c(j))
           t <- t%/%2 # określenie zmiennych kandydatek dla metody
}
# Koniec obliczania pojemności integralnej nośników informacji
}
```

1.4.3. Nazwy obiektów

Nazwa obiektu w języku R może składać się z dowolnego ciągu liter, cyfr, znaków podkreślenia lub kropek. Nie może jednak zaczynać się od cyfry, znaku podkreślenia ani od pary *kropka cyfra*. Nazwy zaczynające się od kropek są zazwyczaj używane do oznaczenia wewnętrznych zmiennych programu i nie zaleca się ich stosowania do innych celów, choć ich użycie nie generuje błędu.

Nazwy w języku R są wrażliwe na wielkość liter, więc zmienne dane, DANE i Dane to trzy zupełnie różne zmienne, które mogą być równocześnie stosowane w pracy w środowisku R (choć stosowanie nazw zmiennych różniących się tylko wielkością liter znacznie zwiększa ryzyko pojawienia się błędu w programie).

Język R, w przeciwieństwie do wielu innych języków programowania, nie wymaga deklarowania obiektów i określania ich typu przed pierwszym użyciem, co z jednej strony jest ułatwieniem, ale z drugiej wymaga precyzji i wzmożonej uwagi, gdyż np. ciąg instrukcji:

```
> zmienna_1 <- 5
> if (zmienna1>4) print("Warunek spełniony")
```

powoduje wyświetlenie komunikatu o błędzie:

```
Error: object "zmienna1" not found
```

1.4.4. Operator podstawienia

Podstawienie wartości wyrażenia pod zmienną można w języku R zrealizować na trzy sposoby:

```
zmienna <- wyrażenie
zmienna = wyrażenie
wyrażenie -> zmienna
```

Operatory te można swobodnie łączyć, np. wydanie polecenia:

```
zmienna1=zmienna2 <- 2+2 -> zmienna3
```

spowoduje, że zarówno `zmienna1`, jak i `zmienna2` oraz `zmienna3` przybiorą wartość 4.

Operator <- może dotyczyć nie tylko liczb, ale również całych obiektów, np. wektorów czy macierzy:

```
kolumna.po.normalizacji <- (x - mean(x))/(sd(x))
# sd(standard deviation) – odchylenie standardowe, mean – średnia
Macierz.Suma <- Macierz.1+Macierz.2
```

Czwartym typem operatora podstawienia jest operator <<- wykorzystywany do tworzenia zmiennych globalnych wewnątrz funkcji. W języku R, podobnie jak w innych językach programowania, sformułowania zmienne lokalne i globalne odnoszą się do zakresu, w jakim są one dostępne i widoczne w programie. Zmienne lokalne to takie zmienne, które są definiowane wewnątrz funkcji i są dostępne tylko w ramach tej funkcji. Nie mają wpływu na resztę środowiska poza funkcją, w której zostały utworzone. Natomiast zmienne globalne to zmienne, które są definiowane na poziomie głównego środowiska i są dostępne z dowolnego miejsca w programie. Zmienne globalne definiuje się zazwyczaj przed wywołaniem funkcji, użycie operatora <<- jest wyjątkiem od tej zasady.

1.4.5. Atrybuty

Każdy obiekt złożony w języku R ma zdefiniowaną listę atrybutów z nim związanych. Atrybutami mogą być nazwy kolumn, nazwy kategorii zmiennych czy wymiary tablicy. Do wyświetlenia listy atrybutów związanych z obiektem służy polecenie `attributes()`. Przykładowe wywołanie tego polecenia:

```
library(clusterSim)
data(data_binary)
attributes(data_binary)
$names
[1] "v_1" "v_2" "v_3" "v_4" "v_5" "v_6" "v_7" "v_8" "v_9" "v_10"
$class
[1] "data.frame"
$row.names
[1] "1" "2" "3" "4" "5" "6" "7" "8"
```

wyświetla informacje o tym, że obiekt `data_binary` jest tabelą danych, zawiera 10 kolumn nazwanych `v_1`, `v_2`, ..., `v_10` i 8 wierszy o nazwach odpowiadających ich kolejnym numerom.

Zestawienie najważniejszych atrybutów mogących opisywać obiekty przedstawiono w tab. 1.2.

Tabela 1.2. Zestawienie najważniejszych atrybutów związanych z obiektami w języku R

Atrybut	Znaczenie
`class`	klasa obiektu
`col.names`	nazwy kolumn
`dim`	wymiary tablic/macierzy
`dimnames`	nazwy poszczególnych wymiarów (najczęściej odpowiada atrybutom `names`/`col.names` i `row.names`)
`levels`	kategorie (poziomy) zmiennej nominalnej
`names`	nazwy elementów listy / kolumn w tablicy danych
`row.names`	nazwy wierszy
`tsp`	dla szeregów czasowych: początek, koniec i częstotliwość

Źródło: opracowanie własne.

Atrybuty obiektów w języku R można wyświetlać i (prawie) dowolnie zmieniać przy pomocy funkcji `attr(nazwa_obiektu, nazwa_atrybutu)`. Dla obiektu `data_binary` z poprzedniego przykładu polecenie

```
print(attr(data_binary, "row.names"))
```

powoduje pojawienie się w oknie środowiska R:

```
[1] "1" "2" "3" "4" "5" "6" "7" "8"
```

Natomiast konstrukcja

```
attr(data_binary, "names") <- c("kolumna_pierwsza",
   "kolumna_druga", "kolumna_trzecia", "kolumna_czwarta",
   "kolumna_piąta", "kolumna_szósta", "kolumna_siódma",
   "kolumna_ósma", "kolumna_przedostatnia", "kolumna_ostatnia")
```

zmienia nazwy kolumn w tabeli danych.

Alternatywnie dla konstrukcji `attr(nazwa_obiektu, nazwa_atrybutu)` można używać instrukcji `nazwa_atrybutu(nazwa_obiektu)`. Poprzednie polecenie można w tej konwencji zapisać jako:

```
names(data_binary) <- c("kolumna_pierwsza", "kolumna_druga",
   "kolumna_trzecia", "kolumna_czwarta", "kolumna_piąta",
   "kolumna_szósta", "kolumna_siódma", "kolumna_ósma",
   "kolumna_przedostatnia", "kolumna_ostatnia")
```

Do wersji 2.4 w języku R do wyświetlenia i zmiany atrybutów można było używać selektora elementów listy (`$`), jednak w nowszych wersjach pakietu nie zaleca się tej techniki, czego konsekwencją jest pojawianie się odpowiednich ostrzeżeń przy próbach jej zastosowania.

1.4.6. Typy obiektów

W języku R wszystko jest obiektem. Obliczeń statystycznych dokonuje się zazwyczaj nie na pojedynczych liczbach, ale na złożonych strukturach, takich jak macierze czy tablice danych, więc obiekty języka R przechowują nie tylko pojedyncze wartości numeryczne, ale również wektory, listy, tabele (ramki) danych czy tablice wielowymiarowe (Biecek, 2017, s. 43-87; Komsta, 2004, s. 7-20). Wiele pakietów definiuje własne typy obiektów, jednak można wyróżnić kilka podstawowych typów obiektów (czyli, używając terminologii programowania obiektowego – klas) wystarczających do oprogramowania większości typowych zadań rozwiązywanych z wykorzystaniem języka R.

a. Wektor

Wektor to ciąg liczb, łańcuchów tekstowych lub wartości logicznych prawda/fałsz. Jeden z najważniejszych i najczęściej używanych typów obiektów.

Najpopularniejszym konstruktorem (funkcją tworzącą obiekt dane typu) wektora jest funkcja `c(wartości_ciągu_oddzielone_przecinkami)`. Na przykład polecenie

```
wektor1 <- c(3,4,2,4,5,7)
```

tworzy wektor o sześciu elementach.

Kolejne elementy wektora są indeksowane, a dostęp do elementów składowych umożliwia selektor `[]`. Na przykład dla utworzonej powyżej zmiennej `wektor1: wektor1[3]` wynosi 3, a `wektor1[5]` 5.

Inne konstruktory wektora:

- `<początek>:<koniec>`

Wektor zawierający wszystkie wartości całkowite z przedziału, np. polecenie

```
wektor2 <- 2:8
```

jest równoważne z poleceniem

```
wektor2 <- c(2,3,4,5,6,7,8)
```

a instrukcja

```
wektor3 <- 5:-1
```

tworzy wektor składający się z siedmiu elementów: `5, 4, 3, 2, 1, 0, -1.`

- `seq`

Funkcja `seq` również zwraca sekwencję liczb z przedziału (a bardziej precyzyjnie ciąg arytmetyczny). W najprostszej postaci jest odpowiednikiem operatora `:`. Dodatkowo można za jej pomocą ustalić różnicę i/lub długość ciągu oraz tak zdefiniować parametry, aby utworzony ciąg składał się z liczb mających część ułamkową. Polecenie

```
wektor4 <- seq(3, 6)
```

tworzy wektor składający się z elementów `3 4 5 6`.

Instrukcja ze zdefiniowaną różnicą ciągu (= 5)

```
wektor5 <- seq(2, 23, 5)
```

tworzy wektor składający się z elementów `2 7 12 17 22`.

Następne polecenie zwraca ciąg długości 4, argumenty przekazane funkcji są jednak tak zdefiniowane, że wyliczona na ich podstawie różnica ciągu nie jest liczbą całkowitą, więc utworzony wektor zawiera elementy niecałkowite:

```
wektor6 <- seq(12, 25, length.out=4)
print(wektor6)
[1] 12.00000 15.66667 20.33333 25.00000
```

Argumentem `along.with` funkcji `seq` może być również inny wektor. W takim przypadku funkcja zwraca ciąg takiej samej długości jak długość ciągu zdefiniowanego przez ten argument. Instrukcja

```
wektor7 <- seq(10,20, along.with=c(14,23,17,2,6,10))
```

tworzy wektor składający się z liczb `10 12 14 16 18 20.`

- `rep`

Funkcja `rep` zwraca wektor utworzony przez replikację elementów pierwszego argumentu. Przykładowe wywołania tej funkcji tworzą wektory o następujących elementach:

```
wektor8 <- rep(10, 20)
print(wektor8)
 [1] 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
wektor9 <- rep(c(10,13,17),5)
print(wektor9)
 [1] 10 13 17 10 13 17 10 13 17 10 13 17 10 13 17
wektor9a <- rep(c(10,13,17),each=5)
print(wektor9a)
 [1] 10 10 10 10 10 13 13 13 13 13 17 17 17 17 17
wektor10 <- rep(c(1,4,-3), c(2,3,5))
# lub alternatywnie
wektor10 <- c(rep(1,2),rep(4,3),rep(-3,5))
print(wektor10)
 [1] 1 1 4 4 4 -3 -3 -3 -3 -3
```

- `sample`

Wektor losowych liczb z określonego przedziału i o zadanej długości może być utworzony przez polecenie wykorzystujące funkcję `sample`:

```
wektor11 <- sample(1:10, 4)
print(wektor11)
[1] 7 6 8 5
wektor12 <- sample(1:20, 25, rep=TRUE) # ostatni argument oznacza, że
                                       # elementy mogą się powtarzać
print(wektor12)
[1] 14 12 5 7 8 4 9 18 3 17 3 18 13 9 7 8 19 20 10 5 10 15 8 1 18
```

- `wektory generowane zgodnie z zadanym rozkładem`

Grupa funkcji `r-<rozkład>` (`rnorm`, `runif`, `rt`, `rweibull` …) (pełna lista dostępna na **https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html**) generuje wektor *n* liczb zgodnie z wybranym rozkładem. Wektor 10 liczb zgodnie z rozkładem normalnym o średniej równej 0 i odchyleniu standardowym równym 1 (argumenty domyślne) tworzy polecenie:

```
wektor13<-runif(10)
print(wektor13)
  [1] 0.2875775 0.7883051 0.4089769 0.8830174 0.9404673 0.0455565 0.5281055
0.8924190 0.5514350
[10] 0.4566147
```

Wektor 15 liczb zgodnie z rozkładem *t*-Studenta o 6 stopniach swobody tworzy polecenie:

```
wektor14<-rt(15,df=6)
print(wektor14)
 [1] 1.6397575 -1.7703227 1.5970612 2.3598845 0.3904608 1.4423866 -1.1187452
-0.3534601 -0.0699085
[10] -0.2522411 -0.8927852 0.5152456 -0.1107163 -0.1501835 -4.6767790
```

Długość wektora

Funkcja `length()` zwraca liczbę elementów w wektorze. Jeśli obiekt `wektor15` jest utworzony instrukcją

```
wektor15 <- seq(1, 30, 7)
```

to funkcja

```
length(wektor15)
```

zwraca wartość 5.

Wektor a typy proste

Specyficzną dla języka R cechą jest brak rozróżnienia między wektorami a zmiennymi składającymi się z pojedynczych wartości. Pojedyncza zmienna traktowana jest po prostu jako wektor składający się z jednego elementu, nawet jeżeli utworzona została przez podstawienie `zmienna <- wartość`. Tak więc po wydaniu polecenia

```
zmienna <- 15
```

funkcja `is.vector(zmienna)` zwraca wartość `TRUE` (prawda) (funkcja jest opisana dokładniej w podrozdziale dotyczącym konwersji typów)

```
is.vector(zmienna)
[1] TRUE
```

Długością wektora `zmienna` jest 1

```
print(length(zmienna))
[1] 1
```

a pierwszy element wektora wynosi 15

```
print(zmienna[1])
[1] 15
```

b. Zmienna nominalna ze zdefiniowanymi kategoriami (typ wyliczeniowy, faktor)

Zmienną nominalną ze zdefiniowanymi kategoriami można traktować jako strukturę, która umożliwia nadawanie znaczących nazw kategoriom odpowiadającym poszczególnym wartościom przyjmowanym przez elementy. Przykładowo do utworzenia zmiennej nominalnej ze zdefiniowanymi kategoriami można użyć polecenia:

```
faktor <- factor(c(2,3,4), levels=1:5)
```

Ma ona w takim przypadku zdefiniowanych 5 kategorii zmiennych. Przy tak utworzonej zmiennej nominalnej polecenie

```
print(faktor)
```

wyświetla

```
[1] 2 3 4
Levels: 1 2 3 4 5
```

czyli zmienną nominalną z wartościami `2, 3, 4` przy zdefiniowanych kategoriach `1, 2, 3, 4, 5`.

Do zmiany nazw kategorii (poziomów) służy atrybut `levels`. Jeśli dla utworzonego wcześniej wektora zmienimy wartość tego atrybutu, zmieniona zostaje również interpretacja nazw poszczególnych kategorii. Po wydaniu polecenia

```
attr(faktor, "levels") <- c("zły","może_być","przeciętny", "dobry", "idealny")
```

lub zgodnie z uwagą z poprzedniego akapitu

```
levels(faktor) <- c("zły", "może_być", "przeciętny", "dobry", "idealny")
```

ponowne wyświetlenie zmiennej faktor poleceniem `print` da w rezultacie

```
[1] może_być przeciętny dobry
Levels: zły może_być przeciętny dobry idealny
```

Próba podstawienia pod zmienną nominalną, ze zdefiniowanymi kategoriami, elementu niezdefiniowanego w atrybucie `levels` kończy się komunikatem o błędzie, a wartość tego elementu będzie nieokreślona (=NA)

```
faktor[3] <- "nijaki"
Warning message:
invalid factor level, NAs generated in: `[<-.factor`(`*tmp*`, 3, value="nijaki")
```

Jeśli zaistnieje potrzeba poznania samych wartości liczbowych elementów zmiennej nominalnej, bez interpretacji nazw kategorii, można skorzystać z funkcji `as.numeric()`

```
as.numeric(faktor)
[1] 2 3 4
```

Jeśli przy tworzeniu zmiennej nominalnej argument `ordered` zostanie ustawiony na wartość `TRUE`, to powstanie zmienna nominalna uporządkowana (odpowiadająca skali porządkowej). Tego typu zmienne nie będą jednak wykorzystywane w dalszej części książki, nie zostaną więc bardziej szczegółowo omówione.

c. Lista

Przy zapisywaniu w języku R algorytmów wielowymiarowej analizy statystycznej dość często występuje sytuacja, gdy efektem działania procedury jest nie pojedyncza wartość, lecz obiekt złożony zawierający wiele informacji. Przykładowo, jeśli w jednym obiekcie mają być zawarte średnia, wartość maksymalna i minimalna wektora `x`, należy utworzyć listę funkcją `list`

```
x <- c(2,3,2,5,4,3,6,7,8)
opisStruktury <- list(srednia=mean(x), minimum=min(x),maksimum=max(x))
```

Wyświetlenie obiektu `opisStruktury` powoduje wyświetlenie wszystkich elementów składowych

```
print(opisStruktury)
$srednia
[1] 4.444444
$minimum
[1] 2
$maksimum
[1] 8
```

W celu odwołania się do poszczególnych elementów składowych obiektu złożonego należy użyć selektora `$`, np.

```
print(opisStruktury$maksimum)
```

lub selektora `[[ numer ]]` oznaczającego wybór elementu o podanym numerze. Poprzednie polecenie może być zapisane również jako

```
print(opisStruktury)[[3]]
```

Ten ostatni sposób bywa jednak rzadko wykorzystywany i dotyczy list, dla których nie zostały, z różnych powodów, zdefiniowane nazwy atrybutów składowych.

Jeśli obiekt złożony został utworzony w wyniku wykonania pewnej funkcji i nie znamy jego struktury, to nazwy elementów składowych są zawarte w atrybucie `names`. W poniższym przykładzie wyświetlone zostaną nazwy elementów składowych listy zwracanej przez funkcję `nnet` realizującą procedurę rozpoznawania niezamawianych e-maili (spamu) za pomocą sieci neuronowej:

```
library(kernlab)
library(nnet)
data(spam)
testSet<-c(1:100,1001:1100,2201:2400,3101:3200)
spam.nn2 <- nnet(type ~ ., data = spam, subset = -testSet, size = 2, rang = 0.1,
                 decay = 5e-4,Hess=FALSE, maxit = 200)
print(spam.nn2)
[1]    "n"        "nunits"        "nconn"          "conn"        "nsunits"
[6]    "decay"    "entropy"       "softmax"        "censored"    "value"
[11]   "wts"      "convergence"   "fitted.values"  "residuals"   "lev"
[16]   "call"     "terms"         "coefnames"      "xlevels"
```

Bardziej szczegółowe informacje o strukturze obiektu złożonego wyświetla funkcja `str`, zwracająca nie tylko nazwy jego składowych, ale także ich typy, długości tablic i aktualne wartości. Dla poprzedniego przykładu użycie `str` zamiast `names` wyświetli dużo więcej danych

```
str(spam.nn2)
List of 19
 $ n             : num [1:3] 57 2 1
 $ nunits        : int 61
 $ nconn         : num [1:62] 0 0 0 0 0 0 0 0 0 0 ...
 $ conn          : num [1:119] 0 1 2 3 4 5 6 7 8 9 ...
 $ nsunits       : int 61
 $ decay         : num 5e-04
 $ entropy       : logi TRUE
 $ softmax       : logi FALSE
 $ censored      : logi FALSE
 $ value         : num 513
 $ wts           : num [1:119] 1.9009 -0.0891 -0.14 0.1957 -2.6814 ...
 $ convergence : int 1
 $ fitted.values: num [1:4101, 1] 0.987 0.985 0.863 0.985 0.863 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$               : chr [1:4101] "101" "102" "103" "104" ...
  .. ..$               : NULL
 $ residuals     : num [1:4101, 1] 0.0131 0.0148 0.1373 0.0148 0.1373 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$               : chr [1:4101] "101" "102" "103" "104" ...
  .. ..$               : NULL
 $ lev           : chr [1:2] "nonspam" "spam"
 $ call          : language nnet.formula(formula = type ~ ., data = spam, size
= 2, rang = 0.1, decay = 5e-04, Hess = FALSE, maxit = 200| __truncated__
 $ terms         :Classes ‘terms’, ‘formula’ language type ~ make + address +
all + num3d + our + over + remove + internet + order + mail + receive + will +
peopl| __truncated__ ...
 .. ..- attr(*, "variables")= language list(type, make, address, all, num3d,
```

```
our, over, remove, internet, order, mail, receive, will, people, repor| __
truncated__ ...
 .. ..- attr(*, "factors")= int [1:58, 1:57] 0 1 0 0 0 0 0 0 0 0 ...
 .. .. ..- attr(*, "dimnames")=List of 2
 .. .. .. ..$ : chr [1:58] "type" "make" "address" "all" ...
 .. .. .. ..$ : chr [1:57] "make" "address" "all" "num3d" ...
 .. ..- attr(*, "term.labels")= chr [1:57] "make" "address" "all" "num3d" ...
 .. ..- attr(*, "order")= int [1:57] 1 1 1 1 1 1 1 1 1 1 ...
 .. ..- attr(*, "intercept")= int 1
 .. ..- attr(*, "response")= int 1
 .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
 .. ..- attr(*, "predvars")= language list(type, make, address, all, num3d,
our, over, remove, internet, order, mail, receive, will, people, repor|
__truncated__ ...
 .. ..- attr(*, "dataClasses")= Named chr [1:58] "factor" "numeric" "numeric"
"numeric" ...
 .. .. ..- attr(*, "names")= chr [1:58] "type" "make" "address" "all" ...
 $ coefnames : chr [1:57] "make" "address" "all" "num3d" ...
 $ xlevels : Named list()
 - attr(*, "class")= chr [1:2] "nnet.formula" "nnet"
```

Na podstawie tych informacji można wywnioskować, że składowa `fitted.values` to tablica z 4101 wierszami i jedną kolumną zawierającą obliczone wartości prawdopodobieństwa przydziału do klas (spam|nie-spam).

d. Tablica

Tablica to obiekt zawierający grupę elementów tego samego typu. Tablicę od wektorów odróżnia to, że elementy tablicy mogą być indeksowane w jednym lub więcej wymiarów. Do utworzenia tablicy służy funkcja `array(dane_początkowe, wymiary)`. Przykładowo polecenie:

```
tablica1 <- array(0, c(2,3))
```

tworzy tablicę o wymiarach 2×3 wypełnioną zerami. Polecenie

```
tablica2 <- array(, c(4,2))
```

tworzy tablicę o wymiarach 4×2 bez określonych wartości (`NA`).

Odwoływanie się do poszczególnych elementów tablicy odbywa się również jak w przypadku wektorów przez selektor `[]`, jednak wewnątrz selektora należy podać indeksy dla wszystkich wymiarów oddzielone przecinkami. Przykładowo, jeśli `tablica3` (utworzona poleceniem `tablica3<-rbind(c(1,8,10),c(-7,3,-5),c(1,-8,-5),c(-3,-2,-3))` zawiera wartości (działanie funkcji `rbind` jest omówione w dalszej części tego rozdziału)

```
     [,1] [,2] [,3]
[1,]    1    8   10
[2,]   -7    3   -5
[3,]    1   -8   -5
[4,]   -3   -2   -3
```

to odwołanie do `tablica3[2,3]` zwróci wartość `-5`, a do elementu `tablica3[3,1]` zwróci wartość `1`.

Pominięcie któregoś z wymiarów powoduje, że obiekt wynikowy zawiera wszystkie elementy indeksowane w tym wymiarze (dla tablic dwuwymiarowych, jeśli pominięty zostanie indeks w pierwszym wymiarze, to wartością wyrażenia jest cała kolumna, a jeśli w drugim, to wartością wyrażenia jest cały wiersz, czyli wyrażenie

```
tablica3[2,]
```

to wektor

```
[1] -7 3 -5
```

a wyrażenie:

```
tablica3[,1]
```

to wektor

```
[1] 1 -7 1 -3
```

R jest bardzo elastyczny, jeżeli chodzi o indeksowanie, selekcję czy sortowanie danych w tablicach. Dokładniejszy opis i przykłady tych operacji znajdują się w rozdziale *Operacje na tablicach*.

Tablice dwuwymiarowe (i macierze) mogą być tworzone również przez łączenie ze sobą wektorów i/lub innych tablic dwuwymiarowych, a nawet pojedynczych wartości liczbowych (te ostatnie są odpowiednio powielane, tworząc kolumnę/rząd). Wykorzystywane są do tego funkcje `cbind` i `rbind`, przy czym pierwsza z nich łączy kolumny, a druga wiersze tabeli. Przykładowo instrukcja

```
tablica4 <- cbind(1:5, rep(2,5), 5:1,-1)
```

tworzy tablicę

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    5   -1
[2,]    2    2    4   -1
[3,]    3    2    3   -1
[4,]    4    2    2   -1
[5,]    5    2    1   -1
```

a ciąg instrukcji

```
tablica5 <- NULL
tablica5 <- rbind(tablica5, c(0,30,45,60,90))
tablica5 <- rbind(tablica5, sin(tablica5*pi/180))
```

tworzy tablicę

```
[1,]    0       30.0    45.0000000      60.0000000      90
[2,]    0        0.5     0.7071068       0.8660254       1
```

Z kolei instrukcja

```
tablica6 <- cbind(rbind(array(1, c(2,2)), array(2, c(3,2))),
            array(3, c(5,2)))
```

tworzy tablicę

```
      [,1] [,2] [,3] [,4]
[1,]    1    1    3    3
[2,]    1    1    3    3
[3,]    2    2    3    3
[4,]    2    2    3    3
[5,]    2    2    3    3
```

Do określenia wymiarów tablicy służy atrybut `dim`. Dla tablicy utworzonej w poprzednim przykładzie funkcja `dim(tablica6)` zwróci `[1] 5 4`

Ponadto dla tablic dwuwymiarowych funkcja `nrow()`zwraca liczbę wierszy tablicy, a `ncol()` liczbę jego kolumn.

```
nrow(tablica6)
[1] 5
ncol(tablica6)
[1] 4
```

e. Macierz

Macierz to tablica posiadająca dwa wymiary. R jest językiem służącym do obliczeń statystycznych, więc w sposób naturalny macierze są w nim bardzo często wykorzystywane. Z tego powodu jest to wydzielony typ danych, jednak z punktu widzenia użytkownika/programisty nie ma większej różnicy między macierzami a tablicami dwuwymiarowymi, gdyż funkcje dotyczące macierzy działają również, jeśli ich argumentami są dwuwymiarowe tablice. Najwygodniejszym sposobem stworzenia macierzy jest użycie funkcji `matrix`, która domyślnie dane wejściowe traktuje jako kolejne kolumny macierzy. Polecenie:

```
macierz1 <- matrix(c(1,3,4,-1,2,4,3,2,1), ncol=3, nrow=3)
```

tworzy macierz o trzech kolumnach (argument `ncol`) i wierszach (argument `nrow`)

```
      [,1] [,2] [,3]
[1,]    1   -1    3
[2,]    3    2    2
[3,]    4    4    1
```

Poprzez użycie argumentu `byrow` możliwa jest również zmiana sposobu wpisywania danych na „wiersz po wierszu". Instrukcja

```
macierz2 <- matrix(c(2,3,1,-1,2,3,3,-2,2), ncol=3, nrow=3,
  byrow=TRUE)
```

tworzy macierz

```
      [,1] [,2] [,3]
[1,]    2    3    1
[2,]   -1    2    3
[3,]    3   -2    2
```

R zawiera bardzo wiele funkcji związanych z rachunkiem macierzowym.

`%*%` – operator mnożenia macierzy. Przykładowo mnożenie macierzy $\begin{bmatrix} 2 & 3 \\ 1 & 3 \end{bmatrix} \times \begin{bmatrix} -1 & 2 \\ -2 & 4 \end{bmatrix}$ jest realizowane przez ciąg instrukcji

```
macierz3  <- matrix(c(2,3,1,3),     ncol=2,    nrow=2,    byrow=TRUE)
macierz4  <- matrix(c(-1,2,-2,4),   ncol=2,    nrow=2,    byrow=TRUE)
print(macierz3%*%macierz4)
```

```
      [,1]  [,2]
[1,]   -8    16
[2,]   -7    14
```

`det()` – obliczenie wyznacznika macierzy kwadratowej. Do obliczenia wyznacznika macierzy $\begin{bmatrix} 2 & -1 \\ -4 & 2 \end{bmatrix}$ można użyć funkcji

```
det(matrix(c(2,-1,-4,2), ncol=2, nrow=2))
[1] 0
```

`solve(A)` – macierz odwrotna do nieosobliwej macierzy kwadratowej `A`. W następnym przykładzie obliczona zostanie macierz odwrotna do macierzy $\begin{bmatrix} 4 & 2 \\ 3 & -1 \end{bmatrix}$. Tym razem macierz zostanie utworzona za pomocą funkcji `array`, gdyż R traktuje dwuwymiarowe tablice tak samo jak macierze:

```
macierz5 <- array(c(4,3,2,-1), c(2,2))
solve(macierz5)
      [,1]   [,2]
[1,]   0.1    0.2
[2,]   0.3   -0.4
```

`solve(A,B)` – rozwiązanie układu `A×X=B`. Odwracając przykład dotyczący mnożenia macierzy, mamy:

```
solve(matrix(c(2,3,1,3),        ncol=2, nrow=2, byrow=TRUE),
      matrix(c(-8,16, -7,14),   ncol=2, nrow=2, byrow=TRUE))
      [,1]  [,2]
[1,]   -1     2
[2,]   -2     4
```

`t(A)` – macierz transponowana. Polecenie

```
t(matrix(c(1,3,4,5), ncol=2, nrow=2, byrow=TRUE))
```

zwróci macierz

```
      [,1]  [,2]
[1,]   1     4
[2,]   3     5
```

f. Tabela (ramka) danych (`data.frame`)

W wielu opracowaniach w języku polskim *data frame* jest tłumaczone jako „ramka danych", co być może jest lepszym tłumaczeniem dosłownym, jednak termin „tabela danych" lepiej oddaje prawdziwe znaczenie obiektów tej klasy, więc będzie on konsekwentnie stosowany w dalszej części rozdziału. Istotne ograniczenie tablic/macierzy dotyczy jednorodności ich elementów. Mogą one zawierać

dane liczbowe, dane w postaci łańcuchów tekstowych lub zmiennych logicznych TRUE/FALSE, jednak w jednej tablicy te dane nie mogą być pomieszane, a R niejawnie dokonuje konwersji danych według zasad:

- jeśli w tablicy są zmienne tekstowe, to liczby i wartości TRUE/FALSE są zamieniane na łańcuchy,
- jeśli w tablicy nie ma zmiennych tekstowych, a są liczbowe, to wartość TRUE jest zamieniana na 1, a wartość FALSE na 0.

O ile do obliczeń używane są dane numeryczne, konwersja ta nie stanowi większego problemu. Jeżeli jednak algorytm może dotyczyć danych niemetrycznych, to strukturą odpowiedniejszą do zapamiętania danych jest tabela danych, w której poszczególne kolumny mogą różnić się od siebie typami. Przykładowo poniższe instrukcje:

```
tabela.danych1 <- data.frame (LETTERS[1:10], 1:10, rep(c(F,T), 5))
names(tabela.danych1) <- c("Inicjał","Kolejność","Czy Parzysty")
```

tworzą tabelę danych

```
     Inicjał   Kolejność   Czy Parzysty
1       A           1           FALSE
2       B           2            TRUE
3       C           3           FALSE
4       D           4            TRUE
5       E           5           FALSE
6       F           6            TRUE
7       G           7           FALSE
8       H           8            TRUE
9       I           9           FALSE
10      J          10            TRUE
```

zawierającą kolumnę łańcuchów tekstowych, kolumnę liczb i kolumnę wartości logicznych TRUE/FALSE. W przykładzie została użyta standardowa dla środowiska R tablica LETTERS zawierająca 26 wielkich liter alfabetu łacińskiego (jak można się domyślić, tablica letters zawiera 26 małych liter tego alfabetu).

Do tworzenia tabel danych stosunkowo rzadko używa się funkcji data.frame, a o wiele częścią korzysta się w tym celu z funkcji read.csv omówionej w rozdziale dotyczącym importu/eksportu danych.

g. Obiekty typu tibble

Nowocześniejszą i zoptymalizowaną dla dużych zbiorów danych wersją tabel danych są obiekty typu tibble/tbl_df (ta nazwa nie ma w chwili pisania książki tłumaczenia na język polski). Korzystanie z niej wymaga zainstalowania pakietu tibble. Najważniejsze statystyczne bazy danych (EUROSTAT, BDL, DBNomics) zwracają najczęściej dane właśnie w tym formacie.

```
library(eurostat)
dlugosc_zycia_kobiet<-get_eurostat("sdg_03_10",
        filters=list(geo=eu_countries$code,time="2021",sex="F"))
class(dlugosc_zycia_kobiet)
[1]     "tbl_df"        "tbl"    "data.frame"
tibble::is_tibble(dlugosc_zycia_kobiet)
[1]     TRUE
```

1.4.7. Konwersja typów

W języku R (o ile jest to wykonalne) można zmieniać typ przy pomocy funkcji konwersji. Wszystkie tego typu funkcje mają format `as.nazwa_typu`. Konwersja może dotyczyć typów prostych, np.

```
t <- TRUE
print(as.integer(t))
```

daje w wyniku

```
[1] 1
```

a jeśli działanie

```
"5" + "4"
```

daje błąd

```
Error in "5" + "4": non-numeric argument to binary operator
```

bo `"5"` i `"4"` to w tym wypadku łańcuchy tekstowe, to operacja

```
as.integer("5") + as.integer("4")
```

daje w wyniku

```
[1] 9
```

Również klasy obiektów złożonych można konwertować na inne, jak w poniższym przykładzie, w którym wykorzystano funkcje konwersji najpierw na zmienną nominalną ze zdefiniowanymi kategoriami, a następnie z powrotem na wektor, aby dla danego wektora wyświetlić jego elementy rosnąco, bez powtórzeń

```
dane <- c(2,4,5,3,4,2,3,4,1,3,4,6,7,3,4,5,7)
bez.powtorzen <- as.vector(levels(as.factor(dane))
print(bez.powtorzen)
[1] "1" "2" "3" "4" "5" "6" "7"
```

Poprawne jest także rzutowanie tablic na typy proste, np. ciąg instrukcji

```
tablica7 <- array(c(0,1,1,0,1,0), c(2,3))
as.logical(tablica7)
```

zwraca wektor

```
[1] FALSE TRUE TRUE FALSE TRUE FALSE
```

W tabelach 1.3 i 1.4 przedstawiono wybrane funkcje konwersji. Tabela 1.3 zawiera funkcje dotyczące typów prostych, a tab. 1.4 funkcje konwersji klas obiektów złożonych.

Tabela 1.3. Najważniejsze funkcje konwersji typów prostych

as.<…>	Konwersja na …
`numeric`	typ liczbowy, bez określonego podtypu
`integer`	typ liczbowy, całkowity
`real`	typ liczbowy, podtyp rzeczywisty
`single`	typ liczbowy, podtyp rzeczywisty
`string`	typ tekstowy
`logical`	typ logiczny (wartości `TRUE` i `FALSE`)

Źródło: opracowanie własne.

Tabela 1.4. Najważniejsze funkcje konwersji klas obiektów złożonych

as.<…>	Konwersja na …
vector	wektor
factor	zmienną nominalną ze zdefiniowanymi kategoriami
matrix	macierz (tablicę dwuwymiarową)
array	tablicę
list	listę
data.frame	tabelę danych
tibble::as_tibble	obiekt typu *tibble*

Źródło: opracowanie własne.

Nazwy typów z tab. 1.3 i 1.4 mogą być też używane w funkcjach postaci `is.nazwa_typu`, które zamiast dokonywać konwersji, sprawdzają, czy argument jest danego typu zwracając odpowiednią wartość logiczną

```
> is.logical("łańcuch tekstowy")
[1] FALSE
> is.vector(1:100)
[1] TRUE
```

1.4.8. Operatory

a. Proste operatory arytmetyczne

Tabela 1.5 przedstawia znaczenie operatorów arytmetycznych w języku R.

Tabela 1.5. Operatory arytmetyczne w języku R

Operator	Znaczenie
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie zmiennopozycyjne (liczb rzeczywistych)
^	potęgowanie
**	potęgowanie (drugi sposób)
%/%	dzielenie całkowite
%%	reszta z dzielenia liczb całkowitych

Źródło: opracowanie własne.

Wszystkie operatory z tab. 1.5 mogą dotyczyć zarówno pojedynczych liczb (choć, jak wspomniano, są to tak naprawdę jednoelementowe wektory), jak i wektorów, tablic i macierzy. Obowiązuje przy tym kilka zasad:

- działania na dwóch liczbach dają w wyniku liczbę;
- działania na dwóch wektorach o tej samej długości dają w wyniku wektor o tej samej długości;

- działania w postaci <*wektor liczb*> operator <*liczba*> dają w wyniku wektor liczbowy, w którym każdy element powstał przez wykonanie działania na kolejnych elementach ciągu początkowego i danej liczby;
- jeśli argumentami działania są dwa wektory liczbowe różnych długości, to krótszy z tych wektorów jest replikowany tyle razy, ile potrzeba, aby był dłuższy lub równy dłuższemu, przy czym jeśli po replikacji długości obu argumentów się nie zgadzają, to działanie jest wykonywane, ale program generuje ostrzeżenie (*warning message*);
- gdy oba argumenty są obiektami wielowymiarowymi, warunkiem wykonalności działania jest zgodność wszystkich wymiarów obiektów.

Te zasady warto prześledzić na przykładach:

`7+8` daje w wyniku `15`
`9^.5` daje w wyniku `3`
`c(2,4,7)*4` daje w wyniku `8 16 28`
`1:10^2` zwraca kwadraty kolejnych liczb naturalnych od `1` do `10`
`c(2,3,-1,5)*1:2` daje w wyniku `2 6 -1 10`
`1:5*3:4` daje w wyniku `3 8 9 16 15` oraz wyświetla ostrzeżenie, że rozmiar dłuższego ciągu nie jest wielokrotnością krótszego ciągu

```
Warning message:
longer object length is not a multiple of shorter object length in: 1:5 * 3:4
```

`as.logical((1:10)%%2)` da w wyniku wektor

```
TRUE   FALSE   TRUE   FALSE   TRUE   FALSE   TRUE   FALSE   TRUE   FALSE
```

(warto zauważyć, że wektor ten zawiera dla liczb od 1 do 10 informację, czy jest to liczba nieparzysta – wartość `TRUE`, czy parzysta – wartość `FALSE`).

Natomiast mnożenie:

```
matrix(c(1,3,4,5,2,3),       ncol=2,   nrow=3,   byrow=TRUE)*
matrix(c(-3,2,-1,2,3,-2),    ncol=2,   nrow=3,   byrow=TRUE)
```

zwraca w wyniku macierz

```
      [,1]   [,2]
[1,]    -3      6
[2,]    -4     10
[3,]     6     -6
```

Warto zauważyć, że ostatni przykład nie jest mnożeniem macierzy, a jedynie mnożeniem odpowiadających sobie elementów tablicy. Z kolei instrukcja

```
matrix(c(1,3,4,5,2,3),       ncol=2,   nrow=3,   byrow=TRUE)*
matrix(c(3,4,2,-2,3,-4),     ncol=3,   nrow=2,   byrow=TRUE)
```

zakończy się komunikatem o błędzie niezgodności wymiarów:

```
Error in matrix(c(1, 3, 4, 5, 2, 3), ncol=2, nrow=3, byrow=T) * : non-conformable
arrays
```

b. Podstawianie dla wektorów/struktur złożonych

Podobnie jak możliwe jest wykonywanie operacji arytmetycznych na wektorach i strukturach dwu lub więcej wymiarowych, tak ich rezultaty mogą być podstawiane pod inne wektory/struktury wielowymiarowe. Zasady tego podstawiania wyznaczają reguły podobne do tych z poprzedniego podpunktu.

- Podstawienie pod wektor (lub kolumnę, lub wiersz tablicy) innego wektora (kolumny lub wiersza tablicy) tej samej długości skutkuje podstawieniem na każdej pozycji tego wektora odpowiadającego jej elementu wektora podstawianego.
- Podstawienie pod wektor (lub kolumnę, lub wiersz tablicy) pojedynczej liczby skutkuje podstawieniem na każdej pozycji tego wektora tej samej podstawianej liczby.
- Podstawienie pod wektor (lub kolumnę, lub wiersz tablicy) innego wektora (kolumny lub wiersza tablicy) różnej długości poprzedzane jest odpowiednim replikowaniem wektora podstawianego (o ile jest krótszy), tyle razy, ile potrzeba, aby był dłuższy od tego wektora lub mu równy, przy czym jeśli po replikacji długości nie są różne, to podstawienie jest „obcinane" i program generuje ostrzeżenie.
- Gdy obiekt, pod który następuje podstawienie, i obiekt podstawiany są obiektami wielowymiarowymi o tej samej liczbie wymiarów, podstawienie następuje dla wszystkich elementów, jeżeli łączna liczba elementów w obiekcie, pod który następuje podstawienie, i w obiekcie podstawianym jest taka sama. Przy tym należy pamiętać, że pierwszym wymiarem dla macierzy jest ten odpowiadający kolumnom, więc wyniki nie zawsze muszą być zgodne z naszą intuicją. Tę regułę ilustruje przykład podstawienia macierzy pod macierz (`m2[,]<-m3[,]`).
- Gdy obiekt, pod który następuje podstawienie, i obiekt podstawiany są obiektami wielowymiarowymi o różnej liczbie wymiarów, podstawienie jest albo ograniczane do odpowiedniej liczby wymiarów do odpowiedniej liczby wymiarów (jeżeli obiekt, pod który następuje podstawienie ma mniej wymiarów niż obiekt podstawiany), albo powielane (w przeciwnym przypadku).
- Gdy obiekt, pod który następuje podstawienie, oraz obiekt podstawiany są obiektami wielowymiarowymi o równej liczbie wymiarów i dodatkowo indeksy obiektu, pod który następuje podstawienie, są filtrowane, podstawienie jest możliwe tylko wtedy, gdy odpowiednie wymiary się zgadzają.

Te zasady ponownie warto prześledzić na przykładach.

Niech zmienne `x`, `m1`, `m2`, `m3` zawierają odpowiednio wektor i trzy macierze

```
x<-c(9,1,5,4,3,8,2,10,7,6)
m1<-matrix(c(66,31,50,88,98,93,94,29,9,57,
    72,14,56,19,21,74,35,51,79,12,
    63,89,10, 8,91,55,15,30,68,83,
    99,17, 2,73,38,32,33,46,60,64,
    78,53,97,45,69,52,85,71,82,13,
    25,92,81,49, 5, 6,96,62,37,43,
    40, 3,58,22,77,59,20,61,95,1,
    11,18,48, 7,34,70,87,41,67,76,
    23,75,47,16,28,39,44,26, 4,27,
    100,65,42,84,80,36,90,24,54,86),10,10,byrow=TRUE)
```

```
m2<-matrix(1:6,2,3)
m3<-matrix(7:12,3,2,byrow=TRUE)
print(x)
 [1]        9     1     5     4     3     8     2     10     7     6
print(m1)
       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
 [1,]    66    31    50    88    98    93    94    29     9     57
 [2,]    72    14    56    19    21    74    35    51    79     12
 [3,]    63    89    10     8    91    55    15    30    68     83
 [4,]    99    17     2    73    38    32    33    46    60     64
 [5,]    78    53    97    45    69    52    85    71    82     13
 [6,]    25    92    81    49     5     6    96    62    37     43
 [7,]    40     3    58    22    77    59    20    61    95      1
 [8,]    11    18    48     7    34    70    87    41    67     76
 [9,]    23    75    47    16    28    39    44    26     4     27
[10,]   100    65    42    84    80    36    90    24    54     86
print(m2)
       [,1]    [,2]    [,3]
[1,]      1       3       5
[2,]      2       4       6
print(m3)
       [,1]    [,2]
[1,]      7       8
[2,]      9      10
[3,]     11      12
```

Podstawienie wektora pod wektor:

```
x[]<-1:10
print(x)
 [1] 1 2 3 4 5 6 7 8 9 10
```

Podstawienie liczby pod wektor:

```
x[]<-7
print(x)
 [1] 7 7 7 7 7 7 7 7 7 7
```

Podstawienie wektora pod wektor innej długości:

```
x[]<-1:2
print(x)
 [1] 1 2 1 2 1 2 1 2 1 2
```

Podstawienie wektora będącego kolumną macierzy pod wektor:

```
x[]<-m1[,1]
print(x)
 [1] 66 72 63 99 78 25 40 11 23 100
```

Podstawienie wektora będącego wierszem macierzy pod wektor:

```
x[]<-m1[9,]
print(x)
 [1] 23 75 47 16 28 39 44 26 4 27
```

Podstawienie liczby pod wektor będący kolumną tablicy:

```
m1[,7]<-7
print(m1)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]   66   31   50   88   98   93    7   29    9    57
 [2,]   72   14   56   19   21   74    7   51   79    12
 [3,]   63   89   10    8   91   55    7   30   68    83
 [4,]   99   17    2   73   38   32    7   46   60    64
 [5,]   78   53   97   45   69   52    7   71   82    13
 [6,]   25   92   81   49    5    6    7   62   37    43
 [7,]   40    3   58   22   77   59    7   61   95     1
 [8,]   11   18   48    7   34   70    7   41   67    76
 [9,]   23   75   47   16   28   39    7   26    4    27
[10,]  100   65   42   84   80   36    7   24   54    86
```

Podstawienie liczby pod wszystkie elementy tablicy:

```
m1[,]<-11
print(m1)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]   11   11   11   11   11   11   11   11   11    11
 [2,]   11   11   11   11   11   11   11   11   11    11
 [3,]   11   11   11   11   11   11   11   11   11    11
 [4,]   11   11   11   11   11   11   11   11   11    11
 [5,]   11   11   11   11   11   11   11   11   11    11
 [6,]   11   11   11   11   11   11   11   11   11    11
 [7,]   11   11   11   11   11   11   11   11   11    11
 [8,]   11   11   11   11   11   11   11   11   11    11
 [9,]   11   11   11   11   11   11   11   11   11    11
[10,]   11   11   11   11   11   11   11   11   11    11
```

Podstawienie wektora pod wszystkie elementy tablicy (powielenie w drugim wymiarze):

```
m1[,]<-1:10
print(m1)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    1    1    1    1    1    1    1    1     1
 [2,]    2    2    2    2    2    2    2    2    2     2
 [3,]    3    3    3    3    3    3    3    3    3     3
 [4,]    4    4    4    4    4    4    4    4    4     4
 [5,]    5    5    5    5    5    5    5    5    5     5
 [6,]    6    6    6    6    6    6    6    6    6     6
 [7,]    7    7    7    7    7    7    7    7    7     7
 [8,]    8    8    8    8    8    8    8    8    8     8
 [9,]    9    9    9    9    9    9    9    9    9     9
[10,]   10   10   10   10   10   10   10   10   10    10
```

Podstawienie macierzy pod wektor (ograniczenie do pierwszego wymiaru):

```
x[]<-m2
Warning message:
In x[] <- m2 :
 number of items to replace is not a multiple of replacement length
print(x)
 [1] 1 2 3 4 5 6 1 2 3 4
```

Podstawienie macierzy pod macierz (kolejność wyznaczona przez pierwszy wymiar):

```
m2[,]<-m3[,]
print(m2)
      [,1]  [,2]  [,3]
[1,]    7    11    10
[2,]    9     8    12
```

Podstawienie macierzy pod macierz ograniczoną filtrami przy niedopasowanych wymiarach – operacja niewykonalna:

```
m1[1:2,]<-m3
Error in m1[1:2, ] <- m3 :
 number of items to replace is not a multiple of replacement length
```

Podstawienie macierzy pod macierz ograniczoną filtrami przy odpowiadających wymiarach:

```
m1[1:2,5:7]<-m3
print(m1)
       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
[1,]    1     1     1     1     7     11     10     1     1      1
[2,]    2     2     2     2     9     8      12     2     2      2
[3,]    3     3     3     3     3     3      3      3     3      3
[4,]    4     4     4     4     4     4      4      4     4      4
[5,]    5     5     5     5     5     5      5      5     5      5
[6,]    6     6     6     6     6     6      6      6     6      6
[7,]    7     7     7     7     7     7      7      7     7      7
[8,]    8     8     8     8     8     8      8      8     8      8
[9,]    9     9     9     9     9     9      9      9     9      9
[10,]  10    10    10    10    10    10     10     10    10     10
```

c. Operatory logiczne

`==, !=, >, >=, <, <=` to operatory porównania: równy, nierówny, większy, większy lub równy, mniejszy, mniejszy lub równy

```
2*2==4
[1] TRUE
7>3
[1] TRUE
-7>-3
[1] FALSE
-7>=-3
[1] FALSE
```

Operatory porównań według tych samych zasad co dla operatorów arytmetycznych mogą dotyczyć wektorów. Przykładowo porównanie

```
c(2,3,5,2,4)>5:1
```

zwróci wartości

```
[1] FALSE FALSE TRUE FALSE TRUE
```

a porównanie

```
c(2,3,5,2,4)==2
```

wartości

```
[1] TRUE FALSE FALSE TRUE FALSE
```

&& – logiczne „i”

Warunek x jest większy od 7 i x jest mniejszy od 15 można zapisać jako

```
(x>7 && x<15)
```

Warunek x nie równa się 4 i x nie równa się 6 jako

```
(x!=4 && x!=6)
```

|| – logiczne „lub”

Warunek x równa się 3 lub x równa się 5 zapisujemy jako:

```
(x==3 || x ==5)
```

! – logiczna negacja

Warunek „Obiekt nie jest macierzą” zapisujemy jako:

```
(!is.matrix(obiekt))
```

~ (tylda)

Tylda, używana zazwyczaj w konstrukcji modeli regresyjnych i dyskryminacyjnych, wskazuje, które ze zmiennych należy uznać za objaśniane, a które za objaśniające. Na przykład wywołanie funkcji

```
lm(popyt~cena+wydatki.na.reklame+cena.konkurencyjnego.produktu)
```

oznacza, że w budowanym modelu zmienną objaśnianą będzie popyt, a zmiennymi objaśniającymi jego cena, wydatki na reklamę i cena konkurencyjnego produktu.

Tylda często występuje w połączeniu z kropką, a `~.` to skrócony zapis oznaczający, że w modelu wszystkie zmienne są zmiennymi objaśniającymi, np.

```
rpart(klasa ~ .,data=x)
optFederov(~ ., pelny)
```

Warto pamiętać, że w języku R występują również pojedyncze operatory `& i |` – wykonujące również operacje logicznego „i” oraz „lub”, ale dotyczą wektorów, a nie pojedynczych warunków logicznych.

1.4.9. Funkcje w pakietach

Siłą języka i środowiska R jest różnorodność funkcji implementujących metody wielowymiarowej analizy statystycznej, obliczających wielkości statystyczne czy realizujących wiele innych złożonych zadań umieszczonych w wielu różnych pakietach.

Wywołanie funkcji języka R odbywa się przez podanie jej nazwy oraz argumentów wywołania. Nazwa funkcji może być poprzedzona nazwą pakietu, w której się znajduje, oraz symbolami `::`, choć tę konwencję stosuje się rzadko, zazwyczaj w przypadku funkcji z różnych pakietów o powtarzających się nazwach. W wywołaniach funkcji języka R argumenty funkcji mogą być podawane z nazwą lub bez nazwy, a także mogą mieć wartości domyślne (choć nie muszą ich mieć). Dokładna składnia wywołania dla każdej funkcji jest podana w jej pliku pomocy.

Przykładowo dla funkcji `cluster.Gen` z pakietu `clusterSim` zdefiniowanej jako

```
cluster.Gen<-function(numObjects=50,means=NULL,cov=NULL,fixedCov=TRUE,model=1,
  dataType="m",numCategories=NULL,numNoisyVar=0,numOutliers=0,
```

```
  rangeOutliers=c(1,10),inputType="csv2",inputHeader=TRUE,inputRowNames=TRUE,
  outputCsv="",outputCsv2="",outputColNames=TRUE,outputRowNames=TRUE)
```

poniższe pięć wywołań realizuje to samo:

```
dane1<-cluster.Gen(100,NULL,NULL,TRUE,5,"m",,12)
dane2<-cluster.Gen(100,,,,5,"m",,12)
dane3<-cluster.Gen(numObjects = 100, model = 5, dataType = "m",numNoisyVar = 12)
dane4<-cluster.Gen(100, model = 5, dataType = "m",numNoisyVar = 12)
clusterSim::dane5<-cluster.Gen(numNoisyVar=12, numObjects=100,
  dataType="m", model=5)
```

W pierwszym i drugim wywołaniu decyduje kolejność argumentów, w trzecim, czwartym i piątym przy wywołaniu podane są nazwy argumentów, w takim przypadku ich kolejność może być dowolna. Dodatkowo w piątym wywołaniu nazwa funkcji jest poprzedzona nazwą pakietu i symbolem `::`

W wywołaniach funkcji można też przekazywać pośrednio argumenty dla funkcji, do których funkcja główna się odwołuje. W deklaracji używany jest do tego symbol … , a każdorazowo jego znaczenie i listę funkcji zależnych można znaleźć w pliku pomocy.

Przykładowo dla funkcji

```
data.Normalization (x,type="n0",normalization="column",...)
```

`...` oznaczają argument przekazywany do funkcji `sum, mean, min, sd, mad`, a w tym argument `na.rm` (który jest argumentem tych funkcji, a nie jest argumentem dla `data.Normalization`) jak w przykładzie:

```
z2 <- data.Normalization(data_ratio,type="n10",normalization="row",na.rm=FALSE)
```

a. Funkcje matematyczne

Język R podobnie jak „zwykłe" arkusze kalkulacyjne oferuje zestaw funkcji umożliwiających obliczanie wartości podstawowych funkcji trygonometrycznych, wykładniczych czy logarytmicznych. Przykładowo, aby obliczyć sinus 30 stopni należy wydać polecenie

```
sin(30*pi/180)
[1] 0.5
```

Obliczenie logarytmu o podstawie 2 z 8192 odbywa się poprzez instrukcję

```
log2(8192)
[1] 13
```

W tabeli 1.6 zestawiono najważniejsze funkcje matematyczne zdefiniowane w języku R.

Tabela 1.6. Wybrane funkcje matematyczne języka R

Funkcja/argumenty	Znaczenie
`cos(x)`	cosinus kąta
`sin(x)`	sinus kąta
`tan(x)`	tangens kąta
`acos(x)`	arcus cosinus liczby
`asin(x)`	arcus sinus liczby

`atan(x)`	arcus tangens liczby w przedziale <–pi/2,pi/2>
`atan2(y, x)`	arcus tangens liczby w przedziale (–pi/pi>
`log`	logarytm naturalny
`log10`	logarytm dziesiętny
`log2`	logarytm o podstawie 2
`logb(x, base)`	logarytm o dowolnej podstawie
`log1p(x)`	`log(1+x)`
`exp`	funkcja wykładnicza o podstawie *e*
`expm1(x)`	`exp(x) - 1`
`factorial(x)`	silnia liczby
`sqrt(x)`	pierwiastek liczby
`abs(x)`	wartość bezwzględna liczby
`cosh(x)`	cosinus hiperboliczny kąta
`sinh(x)`	sinus hiperboliczny kąta
`tanh(x)`	tangens hiperboliczny kąta
`acosh(x)`	arcus cosinus hiperboliczny liczby
`asinh(x)`	arcus sinus hiperboliczny liczby
`atanh(x)`	arcus tangens hiperboliczny liczby
`round(x,digits)`	zaokrąglenie liczby do określonej liczby miejsc po przecinku
`ceiling(x)`	zaokrąglenie liczby do najbliższej większej od niej liczby całkowitej
`floor(x)`	zaokrąglenie liczby do najbliższej mniejszej od niej liczby całkowitej

Źródło: opracowanie własne na podstawie (Crawley, 2007).

b. Funkcje agregujące

Wiele funkcji języka R jako argument pobiera wektor, zwracając obliczoną na jego podstawie pojedynczą liczbę. Funkcje te realizują proste operacje statystyczne, a najważniejsze z nich przedstawia tab. 1.7, w której przyjęto, że zmienna `wektor` utworzona instrukcją

```
wektor <- 1:100
```

zawiera liczby naturalne od 1 do 100.

Tabela 1.7. Wybrane funkcje agregujące

Funkcja	Znaczenie	Wywołanie funkcji	Wynik
`mad`	medianowe odchylenie bezwzględne	`mad(wektor)`	`37.065`
`max`	wartość maksymalna	`max(wektor)`	`100`
`mean`	średnia	`mean(wektor)`	`50.5`
`median`	mediana	`median(wektor)`	`50.5`
`min`	wartość minimalna	`min(wektor)`	`1`
`prod`	iloczyn elementów	`prod(wektor)`	`9.332622e+157`

`quantile`	kwantyl
	50%
	30%
		
		`quantile(wektor,0.5)`
		`quantile(wektor,0.3)`
			
			`50.5`
			`30.7`

`sd`	odchylenie standardowe	`sd(wektor)`	`29.01149`
`sum`	suma elementów	`sum(wektor)`	`5050`

Źródło: opracowanie własne.

Funkcje z tab. 1.7 zawsze obliczają pojedynczą wartość liczbową. Dotyczy to nawet macierzy. Przykładowo dla macierzy utworzonej instrukcją

```
macierz6 <- matrix(c(2,4,3,2,5,4,3,2,4,5,1,2,4,5,3), nrow=5, ncol=3, byrow=TRUE)
print(macierz6)
      [,1]   [,2]   [,3]
[1,]    2      4      3
[2,]    2      5      4
[3,]    3      2      4
[4,]    5      1      2
[5,]    4      5      3
```

funkcja

```
mean(macierz6)
```

wyświetli

```
[1] 3.266667
```

Funkcje agregujące działają również, jeżeli dane wejściowe zawierają elementy typu logicznego. W takim przypadku wartości `FALSE` są traktowane jako zera, a wartość `TRUE` jako jedynki, np.

```
mean(c(T,T,T,F,T))
```

zwraca wartość `0.8`, a funkcja

```
sum(c(4,5,4,3,5,7,3,6,9)%%3==0)
```

wyświetli 4, czyli liczbę elementów wektora podzielnych przez 2.

Niekiedy jednak zamiast obliczać średnią czy inne funkcje agregujące z całej macierzy, zachodzi potrzeba obliczenia niezależnie średniej dla każdej kolumny/wiersza. Umożliwia to funkcja `apply(tablica, rodzaj_obliczeń, funkcja_agregująca)`. Argument `rodzaj_obliczeń` może przybierać wartości: `1` – obliczenia są dokonywane dla wierszy, `2` – obliczenia są dokonywane dla kolumn, `c(1,2)` – obliczenia są dokonywane i dla wierszy i dla kolumn, przy tablicach trójwymiarowych. Wywołania tej funkcji mogą mieć postać zbliżoną do poniższej

```
apply(macierz6, 1, "mean")
[1] 3.000000 3.666667 3.000000 2.666667 4.000000
apply(macierz6, 2, "sum")
[1] 16 17 16
```

1.4.10. Operacje na tablicach i tabelach danych dwu- lub więcej wymiarowych z wykorzystaniem indeksów

Jedną z istotnych funkcjonalności języka R jest zestaw operatorów i funkcji przeznaczonych do operacji na tablicach i tabelach danych. W nowoczesnych zastosowaniach możemy rozróżnić dwa sposoby. Sposobem, który wydaje się trudniejszy, za to bardziej elastyczny, jest odpowiednie indeksowanie struktur danych poprzez wyrażenia wewnątrz operatora `[ , ]`. Korzystanie z opcji indeksowania jest dość trudne dla użytkowników, którzy nie są równocześnie programistami i z myślą o nich przygotowany został pakiet `dplyr` realizujący podobne funkcje prostszymi w zapisie poleceniami. Ten drugi sposób zostanie omówiony w następnym podrozdziale, poniżej natomiast przed-

stawione zostaną zasady pracy na tablicach dwu- i więcej wymiarowych i na tabelach danych z użyciem odpowiedniego indeksowania.

a. Uzyskanie dostępu do określonego fragmentu tablicy

Załóżmy, że w tablicy x o wymiarach 10 na 10 znajdują się dane. Jeśli w indeksach tablicy podamy ciąg wartości, to odpowiednio:

`x[1:3,]` – zwraca tablicę składającą się z pierwszych trzech wierszy tablicy x,

`x[,c(2,4,7,9)]` – zwraca tablicę składającą się z kolumn 2, 4, 7 i 9 tablicy x,

`x[c(3,4,5),1:5]` – zwraca tablicę składającą się z wierszy 3, 4 i 5 i kolumn 1-5 tablicy x,

`x[,-c(3,6)]` – zwraca tablicę składającą się ze wszystkich wierszy i kolumn z wyjątkiem trzeciej i szóstej kolumny tablicy x,

`x[(1:5)*2,(1:5)*2]` – zwraca tablicę składającą się z parzystych wierszy i parzystych kolumn tablicy x,

`x[,c("prędkość","spalanie")]` – zwraca tablicę składającą się z kolumn o nazwach prędkość i spalanie tablicy x (przy założeniu, że w tablicy x istnieją kolumny o takich nazwach),

`x$spalanie` – (dla tabel danych) zwraca wektor składający się z kolumny „spalanie", przy założeniu, że w tabeli danych x istnieją kolumny o takich nazwach),

`cbind(x$prędkość,x$spalanie)` – (dla tabel danych) zwraca tablicę składającą się z kolumn o nazwach „prędkość" i „spalanie tabeli danych x" (przy założeniu, że w tabeli_danych x istnieją kolumny o takich nazwach),

`x[,1]<-1:10` – wpisanie do pierwszej kolumny tablicy x liczb od 1 do 10.

b. Wybór z kontrolą warunków

W celu wyświetlenia tylko tych elementów tablicy, które spełniają określony warunek, należy ten warunek podać w indeksie tablicy. Na przykład, aby z tablicy

```
x <- matrix(c(
  2.51609811,  -0.88439329,  2.25574143,  -0.09888813,  0.09858682,  2.07103444,
-0.50501079, -2.37929928, -0.08729011, 0.45091464,
  2.01637889,  0.90350289,  2.47826722,  -0.26250027,  -0.05320598,  -0.78758520,
-2.14470565, 0.89528323, 2.07769143, -0.38330014,
  -2.29660690,  2.29785191,  0.53269117,  -0.84799771,  -0.34644348,  -0.15558528,
0.01478956, 2.65159822, 0.92844337, 0.78053559,
  -2.38689849,  -2.40363831,  2.13913917,  0.60034724,  0.53650563,  0.08591188,
0.58878101, -0.41176659, -0.20592205, -2.08507603
), nrow = 10, byrow = TRUE)
print(x)
```

```
              [,1]         [,2]         [,3]         [,4]
 [1,]   2.51609811   2.01637889  -2.29660690  -2.38689849
 [2,]  -0.88439329   0.90350289   2.29785191  -2.40363831
```

```
 [3,]   2.25574143   2.47826722   0.53269117   2.13913917
 [4,]  -0.09888813  -0.26250027  -0.84799771   0.60034724
 [5,]   0.09858682  -0.05320598  -0.34644348   0.53650563
 [6,]   2.07103444  -0.78758520  -0.15558528   0.08591188
 [7,]  -0.50501079  -2.14470565   0.01478956   0.58878101
 [8,]  -2.37929928   0.89528323   2.65159822  -0.41176659
 [9,]  -0.08729011   2.07769143   0.92844337  -0.20592205
[10,]   0.45091464  -0.38330014   0.78053559  -2.08507603
```

wyświetlić tylko te wiersze, dla których w 3. kolumnie są wartości ujemne, należy wydać polecenie

```
x[x[,3]<0,]
              [,1]        [,2]        [,3]        [,4]
[1,]   2.51609811   2.01637889  -2.2966069  -2.38689849
[2,]  -0.09888813  -0.26250027  -0.8479977   0.60034724
[3,]   0.09858682  -0.05320598  -0.3464435   0.53650563
[4,]   2.07103444  -0.78758520  -0.1555853   0.08591188
```

Warto zauważyć, że wstawienie warunku logicznego w miejscu występowania indeksu tworzy wektor z wartościami logicznymi, w którym wartość `TRUE` określa te elementy, które mają zostać wybrane. Dla powyższych danych konstrukcja `x[,3]<0` tworzy wektor

```
FALSE TRUE FALSE TRUE FALSE FALSE FALSE TRUE FALSE TRUE,
```

a jej połączenie z `x[x[,3]<0,]` ostatecznie wybiera wiersze 2, 4, 8 i 10 z tablicy `x`.

c. Określenie liczby elementów wektora spełniających zadany warunek

W celu sprawdzenia, dla ilu elementów tablicy zachodzi warunek, należy użyć funkcji `sum` dla tablicy z odpowiednio wpisanym warunkiem. Aby np. policzyć, ile elementów w pierwszej kolumnie tablicy z poprzedniego przykładu jest większych od 0,4, należy wydać polecenie

```
sum(x[,1]>0.4)
[1] 4
```

d. Sortowanie

Do sortowania wektora służy instrukcja `sort`

```
sort(c(5,8,4,3,11,4,6,3,2))
```

zwraca ona w tym przypadku

```
2 3 3 4 4 5 6 8 11
```

Trochę inaczej wygląda sortowanie tablic według zawartości wybranej kolumny. Na przykład w celu posortowania tablicy

```
# Create the array
tablica <- matrix(c(
    69,  27,  78,
    18,  7,   28,
```

```
    39,  72,  93,
    42,  96,  50,
    92,  76,  49,
    11,  85,  90,
    14,  63,  17
), nrow = 7, byrow = TRUE)
colnames(tablica) <- c("zmienna1", "zmienna2", "zmienna3")
print(tablica)
```

```
     zmienna1   zmienna2   zmienna3
1          69         27         78
2          18          7         28
3          39         72         93
4          42         96         50
5          92         76         49
6          11         85         90
7          14         63         17
```

malejąco według `zmienna2`, należy wydać polecenie

```
posortowane <- tablica[order(tablica[,"zmienna2"],
 decreasing=TRUE),]
print(posortowane)
```

```
      zmienna1   zmienna2   zmienna3
4          42         96         50
6          11         85         90
5          92         76         49
3          39         72         93
7          14         63         17
1          69         27         78
2          18          7         28
```

Należy zauważyć, że w ostatnim przykładzie została użyta funkcja `order`, zwracająca indeksy wektora, które sortują go w malejącej (argument `decreasing=TRUE`) kolejności. Pozwala to na organizację danych bez bezpośredniego ich sortowania, a zamiast tego uzyskanie permutacji indeksów, które mogą być następnie użyte do uzyskania posortowanej wersji całej tablicy.

e. Zmiana wartości w kolumnie/wierszu

Pola z tablic wybrane mechanizmami filtrowania mogą być zmieniane poprzez instrukcję podstawienia (z ograniczeniami opisanymi w punkcie *Podstawianie dla wektorów / struktur złożonych*)

Dla tablicy `tab` zawierającej 7 wierszy i 5 kolumn

```
tab <- array(c(
 0.56911257, 0.09053295, 0.84315577, 0.11848042, 0.84234902, 0.74942732, 0.28903022,
 0.83966961, 0.31713652, 0.40104607, 0.03188484, 0.90281947, 0.17683654, 0.16214515,
 1.0000000,  0.1428571,  0.2000000,  0.5000000,  0.1666667,  0.2500000,  0.3333333,
 0.50636501, 0.09740855, 0.81447360, 0.62361654, 0.50534079, 0.62848473, 0.49019316,
 0.82012962, 0.88635066, 0.38771900, 0.77555019, 0.49061076, 0.09769058, 0.84718891
```

```
), dim = c(7, 5))
print(tab)
             [,1]        [,2]        [,3]        [,4]        [,5]
[1,]   0.56911257  0.83966961   1.0000000  0.50636501  0.82012962
[2,]   0.09053295  0.31713652   0.1428571  0.09740855  0.88635066
[3,]   0.84315577  0.40104607   0.2000000  0.81447360  0.38771900
[4,]   0.11848042  0.03188484   0.5000000  0.62361654  0.77555019
[5,]   0.84234902  0.90281947   0.1666667  0.50534079  0.49061076
[6,]   0.74942732  0.17683654   0.2500000  0.62848473  0.09769058
[7,]   0.28903022  0.16214515   0.3333333  0.49019316  0.84718891
```

wykonanie polecenia

```
tab[,3]<-1/tab[,3]
```

spowoduje, że wartości elementów w trzeciej kolumnie przyjmą swoje odwrotności

```
             [,1]        [,2]      [,3]        [,4]        [,5]
[1,]   0.56911257  0.83966961  1.000000  0.50636501  0.82012962
[2,]   0.09053295  0.31713652  7.000002  0.09740855  0.88635066
[3,]   0.84315577  0.40104607  5.000000  0.81447360  0.38771900
[4,]   0.11848042  0.03188484  2.000000  0.62361654  0.77555019
[5,]   0.84234902  0.90281947  5.999999  0.50534079  0.49061076
[6,]   0.74942732  0.17683654  4.000000  0.62848473  0.09769058
[7,]   0.28903022  0.16214515  3.000000  0.49019316  0.84718891
```

Wykonanie zaś poleceń

```
tab<-rbind(tab,NA)
tab[8,]<-apply(tab,2,sum,na.rm=TRUE)
```

spowoduje, że do tablicy zostanie dodany dodatkowy wiersz z sumą wierszy 1...7

```
             [,1]        [,2]       [,3]        [,4]        [,5]
[1,]   0.56911257  0.83966961   1.000000  0.50636501  0.82012962
[2,]   0.09053295  0.31713652   7.000002  0.09740855  0.88635066
[3,]   0.84315577  0.40104607   5.000000  0.81447360  0.38771900
[4,]   0.11848042  0.03188484   2.000000  0.62361654  0.77555019
[5,]   0.84234902  0.90281947   5.999999  0.50534079  0.49061076
[6,]   0.74942732  0.17683654   4.000000  0.62848473  0.09769058
[7,]   0.28903022  0.16214515   3.000000  0.49019316  0.84718891
[8,]   3.50208827  2.83153820  28.000001  3.66588238  4.30523972
```

f. Przykłady użycia tablic

W tablicy `klasyfikacje` znajdują się numery klas, do których należą obiekty. Do tablicy `liczebnoscKlas` należy zapisać, ile elementów należy do każdej z klas:

```
maxLiczbaKlas <- max(klasyfikacje)
liczebnoscKlas <- array(0,maxLiczbaKlas)
for (i in 1:maxLiczbaKlas)
liczebnoscKlas[i]<- sum(klasyfikacje==i)
```

W tablicy `klasyfikacje` znajdują się numery klas, do których należą obiekty. W tablicy `x` znajdują się dane właściwe (obiekty w kolejnych wierszach, zmienne w kolumnach). W celu utworzenia ta-

blicy, w której dla każdej klasy i każdej zmiennej znajdzie się odchylenie standardowe obliczone dla obiektów należących do tej klasy należy:

```
maxLiczbaKlas <- max(klasyfikacje)
odchylenia <- array(0, c(maxLiczbaKlas,dim(x)[2]))
for (i in 1:maxLiczbaKlas)
 odchylenia[i, ] <- apply(x[klasyfikacje == i, ], 2, sd)
```

Tablica `wspolczynniki` o wymiarach 10×2 zawiera w pierwszej kolumnie numer obiektu, a w drugiej pewien współczynnik obliczony dla tego obiektu. Należy podać numer pierwszego obiektu, który spełnia warunek współczynnik > 20.

```
print(wspolczynniki[wspolczynniki[,2]>20,][1,1])
```

1.4.11. Operacje na strukturach danych z wykorzystaniem pakietu dplyr

Pakiety `dplyr` i `tidyr`, część rodziny pakietów `tidyverse`, zawierają wiele funkcji ułatwiających operowanie na złożonych strukturach danych (tabelach danych i obiektach typu `tibble`). Wśród najważniejszych funkcji i operatorów tego pakietu wymienić można:

`inner_join()` – złączenie według kolumn z zachowaniem kolejności;
`union()` – złączenie według wierszy;
`filter()` – filtrowanie zawartości według pola;
`arrange()` – sortowanie według pola (pól);
`mutate()` – zmianę wartości pól;
`distinct()` – wybór tylko niepowtarzających się obiektów;
`select()` – wybór określonej kolumny (kolumn) z tabeli danych;
`group_by()` – utworzenie grupy przy agregowaniu danych (zastępującej poprzednio zdefiniowane grupy);
`group_by(add=TRUE)` – utworzenie dodatkowej grupy przy agregowaniu danych;
`summarise()` – zdefiniowanie funkcji agregujących dla danych pogrupowanych przez `group_by`;
`separate()` – podział pojedynczej kolumny na wiele kolumn za pomocą określonego separatora;
`unity()` – działająca odwrotnie niż funkcja `separate`, łączy wiele kolumn w jedną kolumnę, używając określonego separatora;
`pivot_longer()` – służąca do przekształcania danych z „szerokiego" formatu, w którym dane znajdują się w kolejnych kolumnach, do „dłuższego" formatu, w którym wiele kolumn jest zwijanych w pary klucz-wartość, co zwiększa (zwielokrotnia) liczbę wierszy zmniejszając liczbę kolumn;
`pivot_wider()` – działająca odwrotnie niż `pivot_longer` i służąca do przekształcania danych, tak aby kolejne wartości w parach klucz-wartość umieszczane były w osobnych kolumnach, jej efektem zwykle jest zmniejszenie (często wielokrotnie) liczbę wierszy, przy zwiększeniu liczby kolumn;
`%>%` – połączenie operacji filtrowania i sortowania w potok (do wersji 1.0.0 pakietu `dplyr` wymaga osobnej instalacji pakietu `magrittr`; wersje nowsze pakietu `dplyr` automatycznie importują operator `%>%`, nie wy-

magają więc od użytkownika oddzielnej instalacji `magrittr`); w RStudio operator przetwarzania potokowego może być wstawiony skrótem klawiaturowym *Ctrl+Shift+M*;

`%in%` – przy filtrowaniu – operator zawierania elementu w innym zbiorze;

`between()` – przy filtrowaniu – operator zawierania elementu w przedziale;

`head()` – wyświetlenie początku danych (opcjonalny parametr *n* określa liczbę wyświetlanych wierszy);

`tail()` – wyświetlenie końcowej porcji danych (opcjonalny parametr *n* określa liczbę wyświetlanych wierszy).

W zbiorze `Salaries` pakietu `cardata` znajdują się dane o wynagrodzeniach profesorów amerykańskiego uniwersytetu, zmienna `discipline` oznacza dyscyplinę nauki związaną z wykładowcą (`A` lub `B`), zmienna `sex` oznacza płeć wykładowcy, `yrs.service` liczbę lat pracy, a `salary` wartość wynagrodzenia.

Polecenie

```
library(carData)
library(dplyr)
library(tidyr)
a<-filter(Salaries, discipline=="A")
```

wybiera wykładowców z dyscypliny „A". Z kolei polecenie

```
b<-filter(a, sex=="Male")
```

wybiera mężczyzn, polecenie

```
c<-filter(b, yrs.service>15)
```

wybiera wykładowców o stażu pracy dłuższym niż 15 lat, a polecenie

```
d<-arrange(c,-salary)
```

sortuje dane od największego do najmniejszego wynagrodzenia.

```
print(d)
      rank          discipline    yrs.since.phd    yrs.service     sex    salary
  1   Prof              A              43              43         Male    205500
  2   Prof              A              42              18         Male    194800
  3   Prof              A              33              18         Male    186023
  4   Prof              A              27              23         Male    172505
  5   Prof              A              35              25         Male    168635
  6   Prof              A              32              28         Male    168500
  7   Prof              A              40              19         Male    166605
  8   Prof              A              31              27         Male    163200
  …
 96   AssocProf         A              30              23         Male     74000
 97   Prof              A              49              43         Male     72300
 98   AssocProf         A              45              39         Male     70700
 99   Prof              A              51              51         Male     57800
```

Wszystkie polecenia mogą być połączone w jedno z wykorzystaniem operatora przetwarzania potokowego

```
d<-Salaries %>% filter(discipline=="A") %>% filter(sex=="Male") %>%
 filter(yrs.service>15) %>% arrange(-salary)
```

Z kolei polecenie

```
medium<-filter(Salaries, between(salary,120000,180000))
```

wybiera wszystkich wykładowców o wynagrodzeniu rocznym z przedziału <120000,180000>.

W zbiorze `diamonds` pakietu `ggplot2` znajdują się dane opisujące cenę i inne atrybuty prawie 54 tys. diamentów. Aby wyświetlić tylko te o wadze (zmienna `carat`) większej niż 1 karat, o kolorze (zmienna `color`) `D` lub `E` i skali czystości (zmienna `clarity`) `VV1` lub `IF`, posortowane od najniższej do najwyższej ceny, należy wydać polecenie:

```
Library(ggplot2)
diamonds%>%filter(carat>1)%>% filter(color=="E" | color=="D") %>%
 filter(clarity %in% c("VVS1","IF"))%>%
 arrange(price) %>% print(n=123)
# A tibble: 123 x 10
      carat     cut     color clarity  depth  table  price    x      y      z
      <dbl>    <ord>    <ord>  <ord>   <dbl>  <dbl>  <int>  <dbl>  <dbl>  <dbl>
   1   1.01  Fair          E     VVS1    66.1    55    8529   6.19    6.1    4.06
   2   1.02  Very Good     E     VVS1    62.6    56    9498   6.35    6.5    4.02
   3   1.02  Very Good     E      IF     61.7    60   10029   6.38   6.52    2.98
   4   1.04  Premium       E     VVS1    62.5    59   10350   6.41   6.46    4.02
   5   1.01  Very Good     E     VVS1    62.2    54   10498   6.41   6.31    4.02
   6   1.04  Premium       E     VVS1    62.5    59   10539   6.46   6.41    4.02
   7   1.01  Good          E     VVS1    62.1    59   10567   6.31   6.34    2.99
   …
 121   1.07  Premium       D      IF     60.9    58   18279   6.67   6.57    4.03
 122   1.04  Very Good     D      IF     61.3    56   18542   6.53   6.55    4.01
 123   1.28  Ideal         E      IF     60.7    57   18700   7.09   6.99    4.27
```

Wybranie tylko kolumn `carat`, `color` i `price` ze zbioru `diamonds` odbywa się poprzez polecenie:

```
diamonds%>%select(c(carat,color,price))
# A tibble: 53,940 x 3
        carat    color    price
        <dbl>    <ord>    <int>
   1    0.23       E        326
   2    0.21       E        326
   3    0.23       E        327
   4    0.290      I        334
   5    0.31       J        335
   6    0.24       J        336
   7    0.24       I        336
   8    0.26       H        337
   9    0.22       E        337
  10    0.23       H        338
# ... with 53,930 more rows
```

Zmiana kolumny `price`, polegająca na obniżeniu wartości do 90% oryginalnej, może być zrealizowana przez polecenie

```
diamenty_z_obnizka<-diamonds%>%mutate(price=0.9*price)
```

W zbiorze `sales_data_125k.json` znajdują się wstępnie zagregowane dane o dziennej sprzedaży oddziałów pewnej firmy. Kolumna `category` zawiera informację o kategorii sprzedanego produktu, `date` – datę sprzedaży, `region` – region sprzedaży (kontynent), kolumna `sales_amount` – kwotę transakcji. Zbiór jest umieszczony w repozytorium github, a dostęp do niego może być zrealizowany następującymi instrukcjami:

```
library(jsonlite)
sales_data<-read_json("https://raw.githubusercontent.com/a-dudek-ue/bdl_book/
main/sales_data_125k.json",simplifyVector=TRUE)
head(sales_data,n=4)
          category           date          region      sales_amount
1   Clothing             2023-05-17    Asia                  9105
2   Sports & Outdoors    2023-04-03    North America         9874
3   Beauty & Health      2023-01-07    Europe                3946
4   Electronics          2023-06-22    South America         3855
```

Wybranie i wyświetlenie tylko transakcji z kategorii `Electronics` regionu `Europe` odbywa się poprzez polecenie:

```
sales_data %>% filter (category == "Electronics" & region=="Europe") -> f
f %>% head()
       category        date       region    sales_amount
 1   Electronics    2023-11-01    Europe          6016
 2   Electronics    2023-09-16    Europe          6042
 3   Electronics    2023-11-04    Europe           626
 4   Electronics    2023-06-14    Europe          7638
 5   Electronics    2023-01-25    Europe          9349
 6   Electronics    2023-12-29    Europe           565
```

Wybranie i wyświetlenie tylko transakcji z kategorii `Electronics` regionu `Europe` posortowanych według daty transakcji i malejąco według kwoty transakcji odbywa się poprzez polecenie:

```
sales_data %>% filter (category == "Electronics" & region=="Europe") %>% arrange
(date,-sales_amount) ->g
g %>% head(n=10)
   1  Electronics    2023-01-01    Europe    9919
   2  Electronics    2023-01-01    Europe    9404
   3  Electronics    2023-01-01    Europe    8834
   4  Electronics    2023-01-01    Europe    8262
   5  Electronics    2023-01-01    Europe    8079
   6  Electronics    2023-01-01    Europe    7595
   7  Electronics    2023-01-01    Europe    7210
   8  Electronics    2023-01-01    Europe    6369
   9  Electronics    2023-01-01    Europe    6333
  10  Electronics    2023-01-01    Europe    5977
```

Wybranie i wyświetlenie łącznych dziennych transakcji w kategorii `Clothing` w regionie `Asia` odbywa się przez wydanie polecenia:

```
sales_data%>%filter(region=="Asia")%>%filter(category=="Clothing") %>%
group_by(date) %>% summarise(daily=sum(sales_amount)) %>%
arrange(date)->h
h %>% head(n=8)
  1   2023-01-01    93275
  2   2023-01-02    103064
  3   2023-01-03    103216
  4   2023-01-04    63528
  5   2023-01-05    59812
  6   2023-01-06    73238
  7   2023-01-07    52262
  8   2023-01-08    56576
```

Wybranie i wyświetlenie średniej kwoty transakcji w każdym miesiącu w kategorii `Sports & Outdoors` w regionie `North America` (z wykorzystaniem funkcji `month` z pakietu `lubridate`) odbywa się przez polecenie:

```
library(lubridate)
sales_data %>% filter(region == "North America") %>%
 filter(category == "Sports & Outdoors") %>%
 arrange(date) %>% group_by(month(ymd(date))) %>%
 summarise(average_transaction= mean(sales_amount)) -> i
i %>% print(n=12)
  1       1         5094.
  2       2         4897.
  3       3         4908.
  4       4         4976.
  5       5         5091.
  6       6         4982.
  7       7         5030.
  8       8         5002.
  9       9         5043.
 10      10         4864.
 11      11         5236.
 12      12         4877.
```

Wybranie i wyświetlenie średnich dziennych obrotów (co zazwyczaj powinno się znacząco różnić od poprzedniego przykładu) w każdym miesiącu w kategorii `Sports & Outdoors` w regionie `North America` odbywa się poprzez polecenie:

```
sales_data %>% filter(region == "North America") %>%
 filter(category == "Sports & Outdoors") %>%
 arrange(date) %>% group_by(date) %>% summarize(daily = sum(sales_amount)) %>%
 group_by(month(ymd(date))) %>% summarise(average_daily_turnover = mean(daily)) -> j
j %>% print(n = 12)

   1         1        71145.
   2         2        67328.
   3         3        68707.
   4         4        70500.
   5         5        66184.
   6         6        65435.
   7         7        69125.
   8         8        66311.
   9         9        67912.
```

```
 10        10        64651.
 11        11        70861.
 12        12        61196.
```

Pakiet `dplyr` zawiera też funkcje służące do połączenia poziomego i pionowego zbiorów danych. Niech w zbiorach `join1.csv` i `join2.csv` znajdują się dane dotyczące pracowników pewnej spółki oraz ich stanowisk.

```
          Imię          Nazwisko            Pesel
1    Jan             Duda            91041957778
2    Adrian          Kowalski        96120556257
3    Stefan          Nowak           66110694195
4    Antoni          Maruszenko      44012723551
5    Michał          Wiśniewski      94020772898
6    Adam            Zębik           12242137715
          Pesel           Dział           Stanowisko
1    12242137715     Budowlany       Kasjer
2    44012723551     Remonty         Sprzedawca
3    66110694195     Remonty         Sprzedawca
4    91041957778     Budowlany       Sprzedawca
5    94020772898     Remonty         Starszy specjalista
6    96120556257     Ceramika        Kasjer
```

Aby otrzymać pełną tabelę zawierającą zarówno imiona i nazwisko, jak i stanowiska w kolejności takiej jak w pierwszej tablicy, należy zastosować konstrukcję:

```
library(dplyr)
j1<-data.frame(
  Imię = c("Jan", "Adrian", "Stefan", "Antoni", "Michał", "Adam"),
  Nazwisko = c("Duda", "Kowalski", "Nowak", "Maruszenko", "Wiśniewski", "Zębik"),
  Pesel = c("91041957778", "96120556257", "66110694195", "44012723551",
  "94020772898", "12242137715")
)
j2<-data.frame(
  Pesel = c("12242137715", "44012723551", "66110694195", "91041957778",
  "94020772898", "96120556257"),
  Dział = c("Budowlany", "Remonty", "Remonty", "Budowlany", "Remonty",
  "Ceramika"),
  Stanowisko = c("Kasjer", "Sprzedawca", "Sprzedawca", "Sprzedawca", "Starszy
  specjalista", "Kasjer")
)
j<- j1 %>% inner_join(j2,by="Pesel")j<- j1 %>% inner_join(j2,by="Pesel")
print(j)
```

```
     Imię     Nazwisko        Pesel          Dział          Stanowisko
1    Jan      Duda         91041957778     Budowlany      Sprzedawca
2    Adrian   Kowalski     96120556257     Ceramika       Kasjer
3    Stefan   Nowak        66110694195     Remonty        Sprzedawca
4    Antoni   Maruszenko   44012723551     Remonty        Sprzedawca
5    Michał   Wiśniewski   94020772898     Remonty        Starszy specjalista
6    Adam     Zębik        12242137715     Budowlany      Kasjer
```

W funkcji `inner_join` argument `by` wskazuje nazwę kolumny, według której następuje łączenie. Gdyby kolumna łączona w drugiej tabeli miała inną nazwę, może go zdefiniować jako `dane1%>%inner_join(dane2,by=c("nazwa_1"=" nazwa _2"))`.

W wyniku połączenia do zbioru wynikowego trafiają zawsze te wiersze z pierwszego zbioru i z drugiego zbioru, dla których wartości kolumny występują i w pierwszym, i w drugim zbiorze. Istnieją odmiany polecenia `inner_join` zachowujące się inaczej.

- `left_join()` do zbioru wynikowego wstawiane są wszystkie wiersze z pierwszego zbioru, jeśli w drugim zbiorze nie ma odpowiadających im danych, to są one uzupełniane przez NA,
- `right_join()` do zbioru wynikowego wstawiane są wszystkie wiersze z drugiego zbioru, jeśli w pierwszym zbiorze nie ma odpowiadających im danych, to są one uzupełniane przez NA,
- `full_join()` – połączenie `left_join()` oraz `right_join()`,
- `semi_join()` – do zbioru wynikowego wstawiane są wszystkie wiersze z pierwszego zbioru mające odpowiadające dane w zbiorze drugim, ale nieuzupełnione przez NA (wynik zawiera te same kolumny co pierwszy zbiór),
- `anti_join()` – do zbioru wynikowego wstawiane są wszystkie wiersze niepasujące do siebie z obu zbiorów, uzupełnione przez wartości NA.

Pakiety `dplyr` i `tidyr` umożliwiają również manipulowanie wierszami i kolumnami poprzez ich rozdzielanie i łączenie oraz „wydłużenie" lub „spłaszczenie" ich struktury.

Zbiór `authors` zawiera informacje o autorach niniejszej książki.

```
authors<-data.frame(
 id=1:8,
 signature=c("Andrzej Bąk       Jelenia Góra",
  "Grażyna Dehnel               Poznań",
  "Andrzej Dudek                Jelenia Góra",
  "Eugeniusz Gatnar             Katowice",
  "Krzysztof Kania              Jelenia Góra",
  "Marek Walesiak               Jelenia Góra",
  "Łukasz Wawrowski             Poznań",
  "Artur Zaborski               Jelenia Góra")
)
print(authors)
     id             signature
1    1           Andrzej Bąk    Jelenia Góra
2    2         Grażyna Dehnel          Poznań
3    3          Andrzej Dudek    Jelenia Góra
4    4       Eugeniusz Gatnar        Katowice
5    5        Krzysztof Kania    Jelenia Góra
6    6          Marek Walesiak    Jelenia Góra
7    7       Łukasz Wawrowski          Poznań
8    8          Artur Zaborski    Jelenia Góra
```

Rozdzielenie danych znajdujących się w kolumnie `signature` na cztery kolumny odbywa się poprzez polecenie:

```
authors %>% separate(signature,c("name","surname","address_1","address_2"),
  sep=" ") -> authors_separated
print(authors_separated)
```

```
  id          name     surname    address_1    address_2
1  1        Andrzej         Bąk      Jelenia         Góra
2  2        Grażyna      Dehnel       Poznań         <NA>
3  3        Andrzej       Dudek      Jelenia         Góra
4  4      Eugeniusz      Gatnar     Katowice         <NA>
5  5      Krzysztof       Kania      Jelenia         Góra
6  6          Marek    Walesiak      Jelenia         Góra
7  7         Łukasz   Wawrowski       Poznań         <NA>
8  8          Artur    Zaborski      Jelenia         Góra
```

Połączenie kolumn `address1` i `address2` w kolumnę `city` może być zrealizowane poprzez instrukcje:

```
authors_separated %>% unite("city",address_1,address_2,na.rm = TRUE)-> authors_ok
print(authors_ok)
```

```
  id          name     surname           city
1  1        Andrzej         Bąk   Jelenia_Góra
2  2        Grażyna      Dehnel         Poznań
3  3        Andrzej       Dudek   Jelenia_Góra
4  4      Eugeniusz      Gatnar       Katowice
5  5      Krzysztof       Kania   Jelenia_Góra
6  6          Marek    Walesiak   Jelenia_Góra
7  7         Łukasz   Wawrowski         Poznań
8  8          Artur    Zaborski   Jelenia_Góra
```

W zbiorze `storms` pakietu `ggplot2` znajdują się informacje o burzach tropikalnych.

Warto zauważyć, że ostatnie cztery kolumny zbioru `storms` zorganizowane są w formacie „szerokim" – w kolejnych kolumnach przechowywane są kolejne wartości.

```
data(storms)
tail(as.data.frame(storms),10)
```

```
        name year month day hour  lat  long              status category wind
pressure tropicalstorm_force_diameter hurricane_force_diameter
19528 Nicole 2022    11  10    0 26.7 -78.4           hurricane        1   65
980                          480                          20
19529 Nicole 2022    11  10    6 27.3 -79.8           hurricane        1   65
980                          570                          20
19530 Nicole 2022    11  10    7 27.6 -80.3           hurricane        1   65
980                          570                          20
19531 Nicole 2022    11  10   12 28.0 -81.6      tropical storm       NA   55
984                          480                           0
19532 Nicole 2022    11  10   18 29.0 -82.8      tropical storm       NA   40
989                          300                           0
19533 Nicole 2022    11  10   19 29.2 -83.0      tropical storm       NA   40
989                          300                           0
19534 Nicole 2022    11  11    0 30.1 -84.0      tropical storm       NA   35
992                          300                           0
```

```
19535 Nicole 2022    11  11    6 31.2 -84.6 tropical depression        NA   30
996                           0                          0
19536 Nicole 2022    11  11   12 33.2 -84.6 tropical depression        NA   25
999                           0                          0
19537 Nicole 2022    11  11   18 35.4 -83.8           other low        NA   25
1000                           0                          0
```

Przekształcanie danych z szerokiego formatu do dłuższego formatu, w którym wiele kolumn jest zwijanych w pary klucz-wartość, a wiersze multiplikowane, może zostać zrealizowane przez polecenie:

```
storms %>% mutate(row = row_number()) %>% pivot_
longer(col=c(wind,pressure,tropicalstorm_force_diameter,hurricane_force_
diameter),names_to="meas_type",values_to="value") -> storms_long
tail(as.data.frame(storms_long),40)
      name year month day hour  lat  long              status category   row
meas_type value
78109 Nicole 2022    11  10    0 26.7 -78.4           hurricane        1 19528
wind    65
78110 Nicole 2022    11  10    0 26.7 -78.4           hurricane        1 19528
pressure   980
78111 Nicole 2022    11  10    0 26.7 -78.4           hurricane        1 19528
tropicalstorm_force_diameter   480
78112 Nicole 2022    11  10    0 26.7 -78.4           hurricane        1 19528
hurricane_force_diameter    20
78113 Nicole 2022    11  10    6 27.3 -79.8           hurricane        1 19529
wind    65
78114 Nicole 2022    11  10    6 27.3 -79.8           hurricane        1 19529
pressure   980
78115 Nicole 2022    11  10    6 27.3 -79.8           hurricane        1 19529
tropicalstorm_force_diameter   570
78116 Nicole 2022    11  10    6 27.3 -79.8           hurricane        1 19529
hurricane_force_diameter    20
78117 Nicole 2022    11  10    7 27.6 -80.3           hurricane        1 19530
wind    65
78118 Nicole 2022    11  10    7 27.6 -80.3           hurricane        1 19530
pressure   980
78119 Nicole 2022    11  10    7 27.6 -80.3           hurricane        1 19530
tropicalstorm_force_diameter   570
78120 Nicole 2022    11  10    7 27.6 -80.3           hurricane        1 19530
hurricane_force_diameter    20
78121 Nicole 2022    11  10   12 28.0 -81.6      tropical storm       NA 19531
wind    55
78122 Nicole 2022    11  10   12 28.0 -81.6      tropical storm       NA 19531
pressure   984
78123 Nicole 2022    11  10   12 28.0 -81.6      tropical storm       NA 19531
tropicalstorm_force_diameter   480
78124 Nicole 2022    11  10   12 28.0 -81.6      tropical storm       NA 19531
hurricane_force_diameter     0
78125 Nicole 2022    11  10   18 29.0 -82.8      tropical storm       NA 19532
wind    40
78126 Nicole 2022    11  10   18 29.0 -82.8      tropical storm       NA 19532
pressure   989
```

```
78127 Nicole 2022    11  10   18 29.0 -82.8      tropical storm       NA 19532
tropicalstorm_force_diameter   300
78128 Nicole 2022    11  10   18 29.0 -82.8      tropical storm       NA 19532
hurricane_force_diameter     0
78129 Nicole 2022    11  10   19 29.2 -83.0      tropical storm       NA 19533
wind    40
78130 Nicole 2022    11  10   19 29.2 -83.0      tropical storm       NA 19533
pressure   989
78131 Nicole 2022    11  10   19 29.2 -83.0      tropical storm       NA 19533
tropicalstorm_force_diameter   300
78132 Nicole 2022    11  10   19 29.2 -83.0      tropical storm       NA 19533
hurricane_force_diameter     0
78133 Nicole 2022    11  11    0 30.1 -84.0      tropical storm       NA 19534
wind    35
78134 Nicole 2022    11  11    0 30.1 -84.0      tropical storm       NA 19534
pressure   992
78135 Nicole 2022    11  11    0 30.1 -84.0      tropical storm       NA 19534
tropicalstorm_force_diameter   300
78136 Nicole 2022    11  11    0 30.1 -84.0      tropical storm       NA 19534
hurricane_force_diameter     0
78137 Nicole 2022    11  11    6 31.2 -84.6 tropical depression       NA 19535
wind    30
78138 Nicole 2022    11  11    6 31.2 -84.6 tropical depression       NA 19535
pressure   996
78139 Nicole 2022    11  11    6 31.2 -84.6 tropical depression       NA 19535
tropicalstorm_force_diameter     0
78140 Nicole 2022    11  11    6 31.2 -84.6 tropical depression       NA 19535
hurricane_force_diameter     0
78141 Nicole 2022    11  11   12 33.2 -84.6 tropical depression       NA 19536
wind    25
78142 Nicole 2022    11  11   12 33.2 -84.6 tropical depression       NA 19536
pressure   999
78143 Nicole 2022    11  11   12 33.2 -84.6 tropical depression       NA 19536
tropicalstorm_force_diameter     0
78144 Nicole 2022    11  11   12 33.2 -84.6 tropical depression       NA 19536
hurricane_force_diameter     0
78145 Nicole 2022    11  11   18 35.4 -83.8           other low       NA 19537
wind    25
78146 Nicole 2022    11  11   18 35.4 -83.8           other low       NA 19537
pressure  1000
78147 Nicole 2022    11  11   18 35.4 -83.8           other low       NA 19537
tropicalstorm_force_diameter     0
78148 Nicole 2022    11  11   18 35.4 -83.8           other low       NA 19537
hurricane_force_diameter     0
```

Z kolei odwrotne przekształcanie danych, tak aby kolejne wartości w parach klucz-wartość umieszczane były w osobnych kolumnach, odbywa się poprzez instrukcje:

```
storms_long %>% pivot_wider(names_from="meas_type",values_from="value") ->
  storms_wide
```

```
tail(as.data.frame(storms_wide),10)
        name year month day hour  lat  long              status category   row
wind pressure tropicalstorm_force_diameter hurricane_force_diameter
19528 Nicole 2022    11  10    0 26.7 -78.4           hurricane        1 19528
65      980                         480                       20
19529 Nicole 2022    11  10    6 27.3 -79.8           hurricane        1 19529
65      980                         570                       20
19530 Nicole 2022    11  10    7 27.6 -80.3           hurricane        1 19530
65      980                         570                       20
19531 Nicole 2022    11  10   12 28.0 -81.6      tropical storm       NA 19531
55      984                         480                        0
19532 Nicole 2022    11  10   18 29.0 -82.8      tropical storm       NA 19532
40      989                         300                        0
19533 Nicole 2022    11  10   19 29.2 -83.0      tropical storm       NA 19533
40      989                         300                        0
19534 Nicole 2022    11  11    0 30.1 -84.0      tropical storm       NA 19534
35      992                         300                        0
19535 Nicole 2022    11  11    6 31.2 -84.6 tropical depression       NA 19535
30      996                           0                        0
19536 Nicole 2022    11  11   12 33.2 -84.6 tropical depression       NA 19536
25      999                           0                        0
19537 Nicole 2022    11  11   18 35.4 -83.8           other low       NA 19537
25     1000                           0                        0
```

1.5. Import/eksport danych

1.5.1. Bazy danych

R ma zestaw poleceń umożliwiających odczytywanie danych z plików pakietów statystycznych, takich jak SPSS i SAS i in. Jeśli dane do analizy znajdują się w zewnętrznej bazie danych, to można skorzystać z importu poprzez *Open Database Connectivity* (ODBC) poprzez wywołanie odpowiedniej kwerendy SQL. Przykładem takiego importu może być dokonanie na podstawie danych znajdujących się w tabeli `dane` w bazie danych `Baza` klasyfikacji hierarchicznej metodą Warda:

```
library("RODBC")
channel <- odbcConnect("Baza")
sqlTables(channel)
sh1 <- sqlQuery(channel, "select * from [Dane]")
d <- as.dist(sh1, diag=FALSE, upper=FALSE)
x <- hclust(d, method="ward")
odbcClose(channel)
```

Dla różnych serwerów baz danych istnieją też biblioteki umożliwiające połączenie poprzez inne protokoły niż ODBC (w tym protokoły natywne). Przykładowo połączenie z serwerem MYSQL może być realizowane przez pakiet `RMariaDB` skryptem

```
library(RMariaDB)
con <- dbConnect(RMariaDB::MariaDB(), host = "<server>",
 user = "<użytkownik>", password = "<hasło>",dbname="<baza>")
res <- dbSendQuery(con, "SELECT * FROM cruises WHERE id NOT IN (SELECT cruise_id
FROM cruise_cabin_occupancy)")
 data <- dbFetch(res,n=-1)
 dbHasCompleted(res)
 dbClearResult(res)

 for(i in 1:nrow(data)){
#... przetworzenia danych pobranych z tabeli MYSQL
```

1.5.2. MS Excel/csv

Bardzo popularnym formatem wymiany danych jest format `csv` umożliwiający m.in. import i export danych z MS Excel do środowiska R. Zapisanie danych w Excelu wymagania wybrania polecenia *Plik|Zapisz jako...* i wybrania odpowiedniego formatu.

Wczytanie tych danych w R natomiast to instrukcja `read.csv` (lub jeśli, tak jak w przypadku polskiego MS Office, jako symbol ułamka dziesiętnego używany jest przecinek, a nie kropka, funkcji `read.csv2`)

```
z <- read.csv2("lekcja_2.csv", header=TRUE,strip.white=TRUE, row.names=1)
```

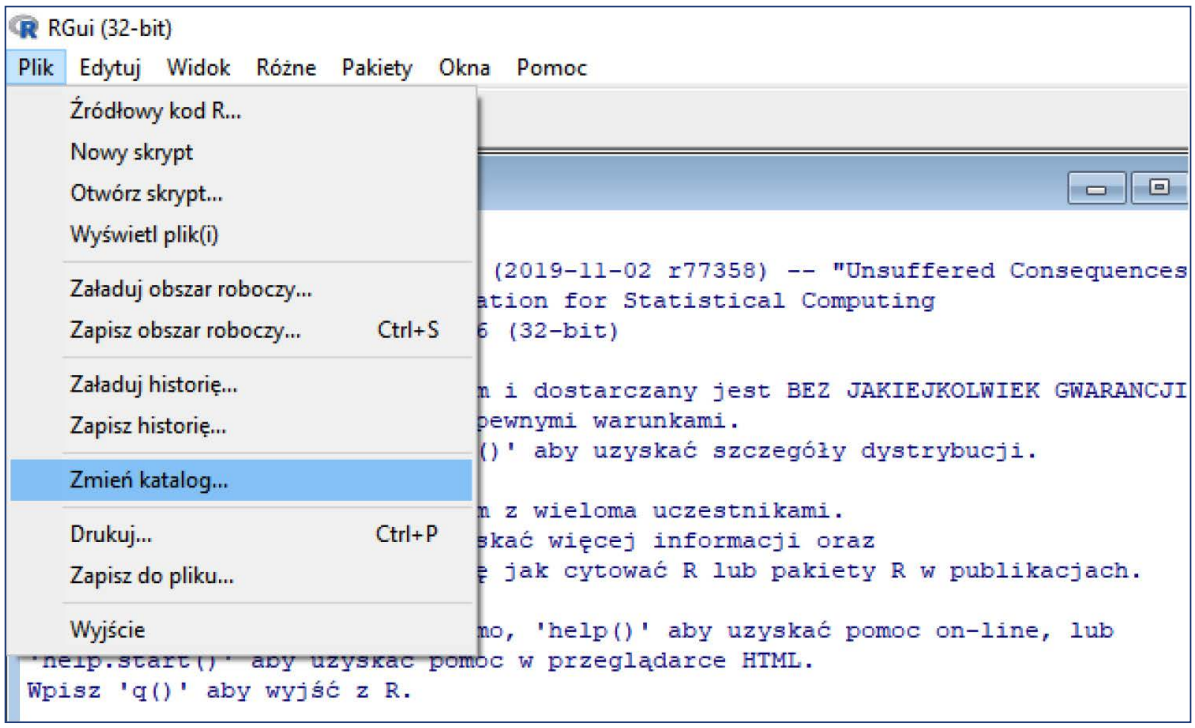
Podane argumenty funkcji oznaczają, że dane w pliku `csv` zawierają nagłówek (`header=TRUE`), w pierwszej kolumnie znajdują się nazwy obiektów (`row.names=1`) i że spacje występujące bezpośrednio po znaku separatora (";") będą przy odczycie danych pomijane (`strip.white=TRUE`). Należy zawsze podawać pełną ścieżkę dostępu do pliku z danymi z symbolem „/" jako separatorem nazwy, chyba że dane znajdują się w katalogu bieżącym programu (ustalanym poleceniem menu `File|change dir`), tak jak w powyższym przykładzie.

R ma również możliwość eksportowania danych w formacie `csv`. Można do tego celu użyć funkcji `write.table`

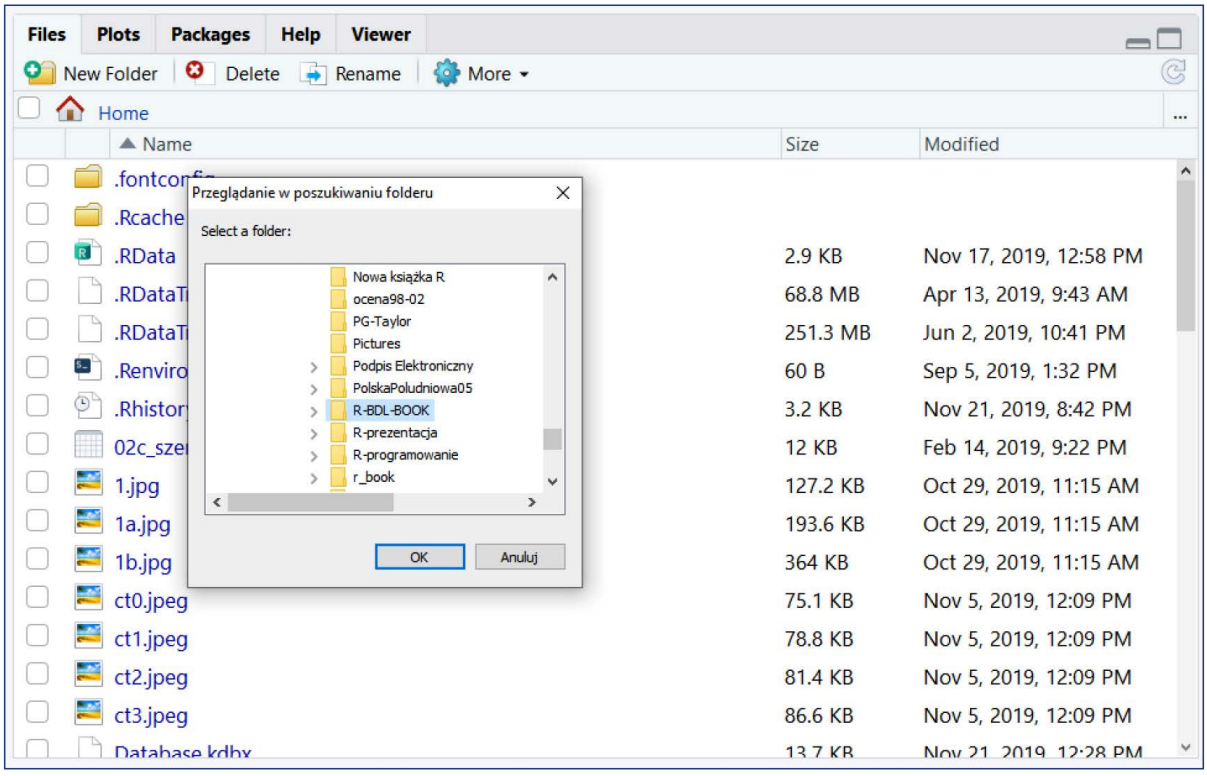
```
write.table(wynik, wynik.csv,sep=";",dec=",")
```

Argumenty `sep` i `dec` oznaczają symbole separatora danych i znaku ułamka dziesiętnego, więc dla polskiej wersji MS Office to `;` oraz `,` (gdyby pominąć te argumenty, to użyte zostałyby odpowiednio `,` oraz `.`). Nazwa pliku do zapisania może być podana w postaci pełnej ścieżki dostępu, jak w przykładzie powyżej, albo bez określania katalogu, wtedy plik `.csv` zostanie utworzony w katalogu bieżącym programu.

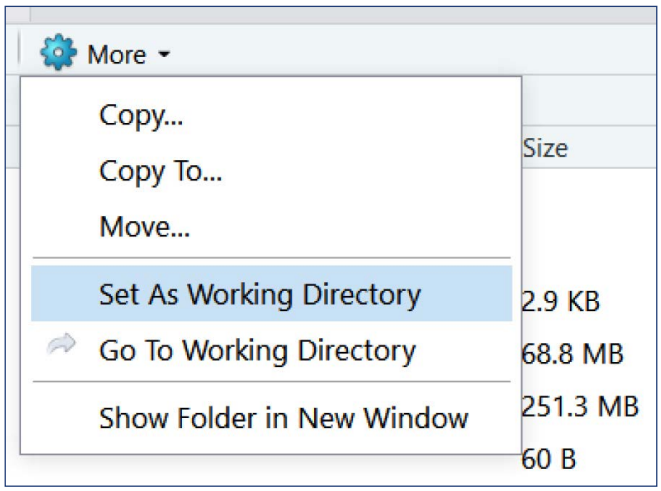
Przy imporcie danych należy zwrócić szczególną uwagę na to, aby pliki znajdowały się w aktualnym katalogu roboczym środowiska R. Aby zmienić katalog roboczy w programie R, należy wydać polecenie *Plik|Zmień katalog* (por. rys. 1.12), natomiast w środowisku RStudio zmiana katalogu roboczego odbywa się poprzez wskazanie katalogu w zakładce przeglądania plików panelu narzędziowego, czyli przez naciśnięcie symbolu `...` , właściwy wybór katalogu (por. rys. 1.13 i 1.14) oraz wybranie w tym samym panelu opcji *More|Set As Working Directory*).



Rys. 1.12. Wybór katalogu roboczego w RGui w systemie MS Windows



Rys. 1.13. Wybór katalogu roboczego w RStudio



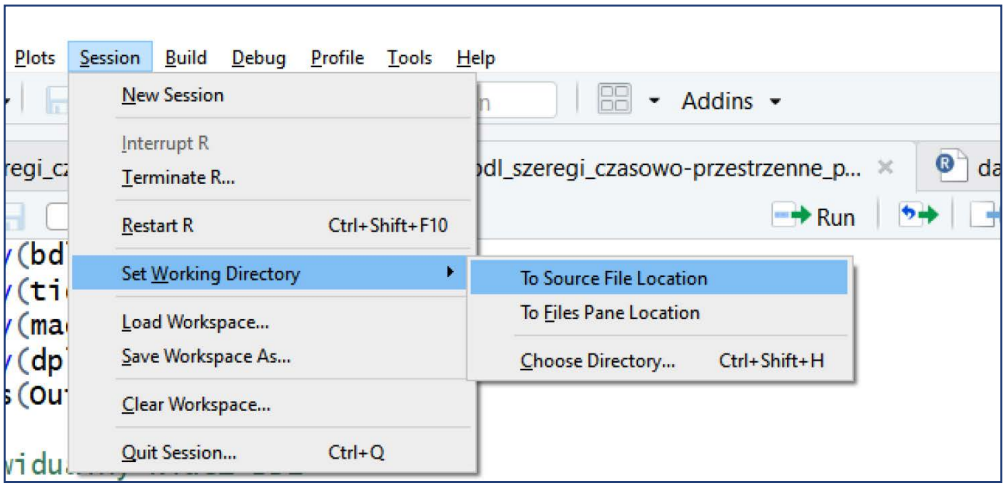
Rys. 1.14. Wybór katalogu roboczego w RStudio (cd.)

Jeżeli niewłaściwie wskażemy katalog roboczy, to próba załadowania pliku, który w nim się nie znajduje, kończy się błędem:

```
Error in file(file, "r"): cannot open the connection
In addition: Warning message:
In file(file, "r") :
  cannot open file ‘lekcja_2.csv’: No such file or directory
```

Oznacza to, że powinniśmy wrócić do wyboru katalogu roboczego lub podać przy wczytywaniu danych pełną ścieżkę dostępu do pliku.

Jeżeli właśnie wykonywany skrypt znajduje się już w katalogu, w którym są umieszczone pliki danych, to w RStudio można zmienić katalog roboczy poprzez „skrót", wybierając opcję *Session|Set As Working Directory|To Source File Location*.



Rys. 1.15. Wybór katalogu roboczego w RStudio, opcja alternatywna

Inną sytuacją błędną, która może wystąpić przy imporcie plików w formacie `csv` (spotykaną często, zwłaszcza u początkujących użytkowników) jest ta, w której niewłaściwie podamy argumenty `header`, `row.names`, `sep`, `dec` lub użyjemy funkcji `read.csv` zamiast `read.csv2` i odwrotnie).

Przykładowo, jeżeli dane się wczytały, ale zaczynają się od drugiego wiersza, pierwszy wiersz zawiera zamiast liczb nazwy zmiennych, pierwsza kolumna zawiera zamiast liczb nazwy obiektów lub brakuje wartości pierwszej zmiennej (kolumny).

Tego typu sytuacje błędne wynikają z nieprawidłowego stosowania argumentów `header` i `row.names`. Pierwszy z nich może przyjmować wartości `TRUE` (domyślnie) oraz `FALSE` i określa, czy dane zawierają wiersz nagłówkowy z nazwami zmiennych. Drugi wskazuje, czy dane zawierają kolumnę z nazwami obiektów, i może przyjmować wartości: `NULL` (domyślnie), gdy w danych nie ma kolumny z nazwami obiektów, lub wartość liczbową (najczęściej 1) z numerem kolumny zawierającej nazwę obiektu.

W przypadku danych z pliku `Dane_samochody.csv` prawidłowa forma wczytania pliku to

```
dane<-read.csv2("lekcja_2.csv",header=TRUE,row.names=1)
```

Gdyby natomiast w pliku nie było ani wiersza, ani kolumny nagłówkowej, prawidłową formą jego załadowania byłaby instrukcja

```
dane<-read.csv2("lekcja_2.csv",header=FALSE,row.names=NULL)
```

Format `csv` jest formatem uniwersalnym obsługiwanym praktycznie przez wszystkie arkusze kalkulacyjne, serwery bazodanowe i środowiska statystyczne. Natomiast do importu/eksportu danych bezpośrednio z/do programu *MS Excel* można wykorzystać pakiety `readxl` i `xlsx`.

Przykładowo, aby zsumować wszystkie liczby w pierwszym arkuszu skoroszytu `do_zsumowania.xls`, można wydać polecenia

```
library(readxl)
dane8<-as.matrix(read_excel("do_zsumowania.xls",1,col_names = F))
sum(dane8)
```

Z kolei zapisanie zbioru danych `diamonds` (zob. podrozdział 1.4) do skoroszytu MS Excel wymaga wydania poleceń:

```
library(xlsx)
librabr(ggplot2)
write.xlsx(wynik,file="eksport_z_programu_r.xlsx",sheetName="diamond", append=F)
```

W tym ostatnim przykładzie warto zwrócić uwagę na argument `append` funkcji `write.xlsx` informujący, czy ma zostać utworzony nowy skoroszyt, czy arkusz ma być dodany do skoroszytu już istniejącego.

1.5.3. Format JSON

JSON (*JavaScript Object Notation*) to lekki format wymiany danych, który zyskał w ostatnich latach bardzo duży obszar zastosowań i można go traktować jako standard wymiany danych, zwłaszcza dla danych innych niż tradycyjne tablice danych numerycznych przechowywanych w plikach `csv` i skoroszytach arkuszy kalkulacyjnych. Struktura danych w JSON jest zbliżona do obiektów w języku JavaScript, dzięki czemu jest bardzo popularna w kontekście aplikacji internetowych.

Najważniejsze konwencje formatu JSON:

- obiekty: zdefiniowane za pomocą par *klucz*: *wartość* ujętych w nawiasy klamrowe {}; klucze muszą być tekstem, wartości natomiast mogą być różnymi typami (liczbami, tekstem, tablicami, innymi obiektami);
- tablice: zbiór wartości ujętych w nawiasy kwadratowe []; mogą zawierać dowolny typ wartości, włączając inne obiekty lub tablice;
- liczby: liczby całkowite lub zmiennoprzecinkowe.
- tekst: ujęty w cudzysłów “ ”.
- wartości logiczne: true i false.
- null: reprezentuje brak wartości.

Przykładowa struktura pliku JSON:

```
[
        {
                "category": "Clothing",
                "date": "2023-05-17",
                "region": "Asia",
                "sales_amount": 9105
        },
        {
                "category": "Sports & Outdoors",
                "date": "2023-04-03",
                "region": "North America",
                "sales_amount": 9874
        }
]
```

W języku R JSON jest często używany do komunikacji między aplikacjami lub do wymiany danych z API. R oferuje kilka pakietów: `jsonlite`, `rjson`, `httr`, umożliwiających pracę z tym formatem. Pierwszy z nich jest chyba najbardziej popularny, a korzystanie z niego nie jest trudniejsze niż w przypadku plików `xls(x)` czy `csv`.

Utworzenie, wyświetlenie i zapisanie do pliku obiektu JSON może odbywać się poprzez polecenia zbliżone do następujących:

```
library(jsonlite)
data <- list(
name = "Stefan Lewandowski",
age = 30,
isEmployed = TRUE,
skills = c("R", "Python", "SQL"),
address = list(street = "Główna 123", city = "Warszawa")
)

json_data <- toJSON(data)
print(json_data)
{"name":["Stefan Lewandowski"],"age":[30],"isEmployed":[true],"skills":["R",
"Python","SQL"],"address":{"street":["Główna 123"],"city":["Warszawa"]}}
 write_json(data, "dane1.json")
write_json(ggplot2::diamonds, "kopia_diamentow.json")
```

Podczas wczytywania zapisanych w formacie JSON macierzy, tablic danych czy obiektów typu *tibble* należy zwrócić uwagę na ich właściwą interpretację. Domyślnie funkcja `read_json` traktuje wiersze tabeli jako złożone listy, co nie jest wygodne w interpretacji i w dalszym przetwarzaniu danych:

```
sales_data_not_simplified<-read_json("https://raw.githubusercontent.com/a-
dudek-ue/bdl_book/main/sales_data_125k.json")
head(sales_data_not_simplified,n=1)
[[1]]
[[1]]$category
[1] "Clothing"

[[1]]$date
[1] "2023-05-17"

[[1]]$region
[1] "Asia"

[[1]]$sales_amount
[1] 9105
```

Najprostszym sposobem zaimportowania takiego obiektu jako tablicy danych jest ustawienie parametru `simplifyVector` na `TRUE`

```
sales_data_data_frame<-read_json("https://raw.githubusercontent.com/a-dudek-
ue/bdl_book/main/sales_data_125k.json",simplifyVector=TRUE)
head(sales_data_data_frame,n=5)
                category        date          region       sales_amount
1               Clothing   2023-05-17            Asia               9105
2      Sports & Outdoors   2023-04-03   North America               9874
3        Beauty & Health   2023-01-07          Europe               3946
4            Electronics   2023-06-22   America South               3855
5        Beauty & Health   2023-02-25          Africa               3106
```

1.5.4. Format `Rdata/rda`

R ma również własny format zapisu danych `*.Rdata`, często stosowany w skróconej formie `.rda`, który jest o tyle lepszy od formatu `csv`, że można w nim zapisywać dowolne, nawet bardzo złożone obiekty. Do zapisywania obiektów w tym formacie służy polecenie `save`, a do ładowania danych do pamięci polecenie `load`. W poniższym przykładzie najpierw zostanie utworzony obiekt złożony `dane`, następnie zapisany do pliku `dane.rda`, po czym usunięty. Próba wyświetlenia jego zawartości skończy się niepowodzeniem, natomiast po użycia polecenia `load` obiekt `dane` będzie miał przywróconą pierwotną zawartość.

```
dane <- list(a1=1:5, a2=letters[15:10],a3=data.frame(rep(3,10),rep(c(F,T),5)))
print(dane)
$a1
[1] 1 2 3 4 5
$a2
[1] "o" "n" "m" "l" "k" "j"
$a3
```

```
   rep.2..10.  rep.c.F..T...5.
 1          3            FALSE
 2          3             TRUE
 3          3            FALSE
 4          3             TRUE
 5          3            FALSE
 6          3             TRUE
 7          3            FALSE
 8          3             TRUE
 9          3            FALSE
10          3             TRUE
save(dane, file="dane.rda")
rm(dane)
print(dane)
Error in print(dane): object "dane" not found
load(file="dane.rda")
print(dane)
$a1
[1] 1 2 3 4 5
$a2
[1] "o" "n" "m" "l" "k" "j"
$a3
     rep.2..10.    rep.c.F..T...5.
  1           3              FALSE
  2           3               TRUE
  3           3              FALSE
  4           3               TRUE
  5           3              FALSE
  6           3               TRUE
  7           3              FALSE
  8           3               TRUE
  9           3              FALSE
 10           3               TRUE
```

Wiele pakietów zawiera swoje własne zbiory danych, które można załadować do pamięci operacyjnej poleceniem `data` jak w następującym przykładzie:

```
library(clusterSim)
data(data_ordinal)
print(data_ordinal)
   v_1  v_2  v_3  v_4  v_5  v_6  v_7  v_8  v_9  v_10  v_11  v_12
1    3    2    3    2    2    3    1    3    2     1     2     2
2    1    2    3    1    1    4    1    4    1     3     1     1
3    4    3    2    4    2    4    3    3    2     1     3     2
4    1    2    1    4    1    2    2    4    1     2     2     1
5    2    3    3    1    2    4    2    3    1     2     3     3
6    3    2    2    2    1    2    1    3    2     1     2     2
7    2    4    2    2    1    4    1    4    1     2     1     2
8    1    3    3    1    2    4    2    4    1     2     2     1
9    2    2    2    1    3    4    3    3    1     2     3     3
```

```
 10    1    2    2    1    1    4    1    4    1    3    1    2
 11    2    4    3    4    2    4    3    4    1    2    2    2
 12    2    3    4    4    1    3    1    3    1    2    1    3
 13    2    2    1    1    1    2    1    3    2    3    3    3
 14    1    2    1    1    1    2    2    3    2    1    1    1
 15    3    3    2    1    2    4    2    3    1    1    1    2
 16    2    4    4    2    3    2    3    3    2    2    3    3
 17    2    4    3    1    4    2    2    4    1    2    2    2
 18    3    3    2    1    2    2    2    3    2    2    1    1
 19    3    4    3    3    3    4    3    3    1    1    3    3
 20    2    2    2    4    2    1    2    3    1    2    1    1
 21    1    3    3    1    2    4    2    4    1    2    2    3
 22    1    2    2    2    1    4    1    4    1    1    2    3
 23    2    4    3    1    3    4    2    3    1    1    2    3
 24    3    3    5    2    3    4    3    2    2    2    3    3
 25    3    2    2    1    2    4    3    1    2    2    3    2
 26    4    3    2    2    2    3    3    1    2    3    3    3
```

1.5.5. Repozytoria internetowe

Wiele zbiorów danych można pobrać bezpośrednio z Internetu poprzez coraz popularniejsze serwery usług sieciowych i współpracujące z nimi pakiety, takie jak `dbnomics`, `WDI`, `eurostat` czy `bdl`. Powstają także pakiety, takie jak `rtwitter`, umożliwiający pobieranie danych z portali społecznościowych, i `tidyquant`, pobierający szeregi czasowe dotyczące akcji i innych danych finansowych z serwisów typu Yahoo Finance.

Przykładowo wyświetlenie kursów akcji firmy KGHM w ostatnich 5 latach może odbywać się poprzez wydanie poleceń:

```
library(tidyquant)
library(lubridate)

getSymbols(
 "KGH.WA",
 from = Sys.Date() %m-% years(5),
 warnings = FALSE,
 auto.assign = TRUE,
)
kghm<-KGH.WA
print (kghm)
```

```
           KGH.WA.Open  KGH.WA.High  KGH.WA.Low  KGH.WA.Close  KGH.WA.Volume  KGH.WA.Adjusted
2019-05-08       96.14        97.72       95.60         95.90         446828         91.57574
2019-05-09       95.40        96.44       91.70         92.64         620722         88.46273
2019-05-10       92.88        95.84       92.88         95.64         516608         91.32746
2019-05-13       95.30        95.30       91.50         92.24         687962         88.08077
2019-05-14       92.94        93.40       91.68         92.32         569685         88.15717
2019-05-15       93.10        93.68       92.04         92.92         264158         88.73011
2019-05-16       94.64        96.98       94.18         96.82         519642         92.45425
2019-05-17       96.50        96.50       94.42         94.88         367624         90.60172
2019-05-20       95.00        95.28       93.32         94.10         363487         89.85690
2019-05-21       94.00        95.50       93.04         95.48         462614         91.17467
```

```
                ...
2024-04-22    143.00    143.50    139.25    139.65    580462    139.64999
2024-04-23    137.85    137.90    132.05    132.65   1400385    132.64999
2024-04-24    134.50    137.00    132.10    132.90    816131    132.89999
2024-04-25    134.20    139.20    134.00    137.95    933973    137.95000
2024-04-26    141.05    142.75    139.55    139.70    966763    139.70000
2024-04-29    140.30    143.80    139.50    143.80    792777    143.80000
2024-04-30    143.85    143.95    139.70    140.65   1520080    140.64999
2024-05-02    140.75    141.40    138.45    139.95    504042    139.95000
2024-05-06    141.75    144.40    141.70    144.20    603682    144.20000
2024-05-07    144.00    145.95    140.65    143.80    425369    143.80000
```

(uważny czytelnik na pewno zauważy, że powyższy skrypt ustawia dynamicznie daty początku i końca pobierania danych, więc każdorazowo wywołanie może dawać inne rezultaty).

Natomiast pobranie informacji o poziomie PKB *per capita* w poszczególnych krajach w 2022 r. może być wykonane poprzez instrukcje:

```
library("WDI")
gdp = WDI(indicator='NY.GDP.PCAP.KD', start=2022, end=2022)
> tail(gdp,10)
```

	country	iso2c	iso3c	year	NY.GDP.PCAP.KD
257	Uruguay	UY	URY	2022	18214.808
258	Uzbekistan	UZ	UZB	2022	3473.362
259	Vanuatu	VU	VUT	2022	2575.931
260	Venezuela, RB	VE	VEN	2022	NA
261	Viet Nam	VN	VNM	2022	3655.463
262	Virgin Islands (U.S.)	VI	VIR	2022	NA
263	West Bank and Gaza	PS	PSE	2022	3095.500
264	Yemen, Rep.	YE	YEM	2022	1017.873
265	Zambia	ZM	ZMB	2022	1308.102
266	Zimbabwe	ZW	ZWE	2022	1345.769

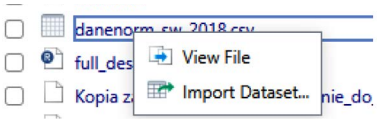
W kolejnych rozdziałach będą sukcesywnie przedstawiane przykłady takiej akwizycji danych, zwłaszcza dla pakietu `bdl`.

1.5.6. Graficzny import danych w RStudio

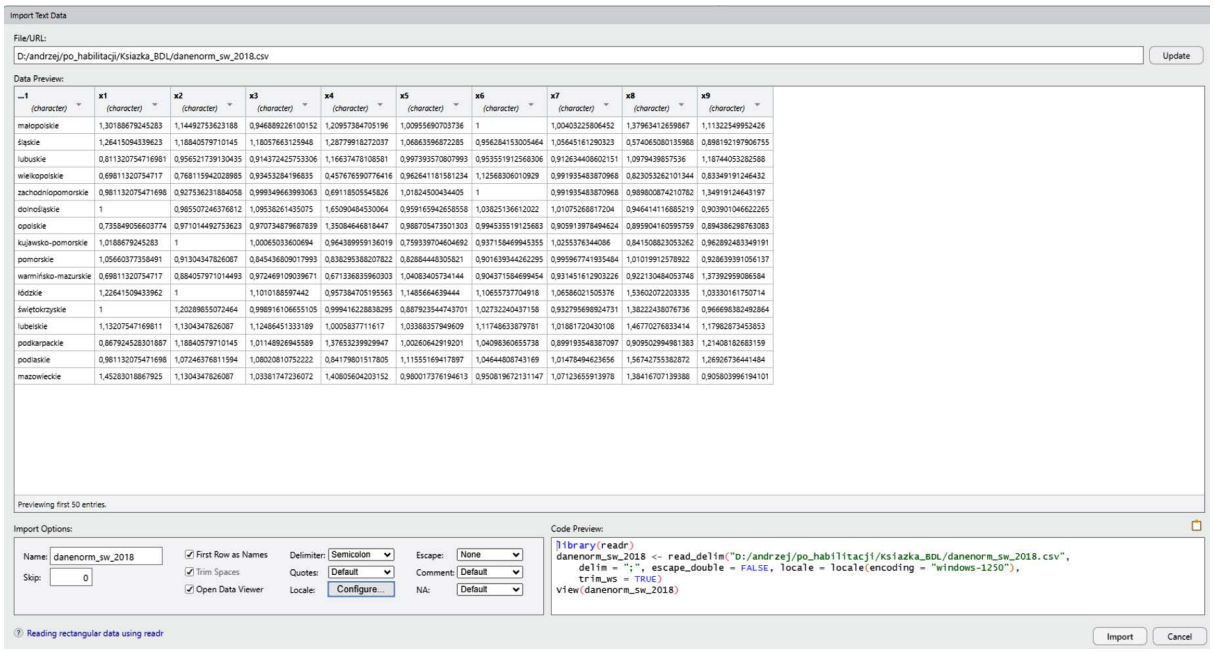
Środowisko RStudio oferuje funkcjonalność importu danych różnych formatów bez konieczności używania instrukcji programistycznych (choć w rzeczywistości należałoby napisać, że czynności związane z importem danych „w locie" tłumaczone są na funkcje odpowiednich pakietów).

Procedurę importu danych w formacie `csv` można opisać w następujących krokach:

1. W górnym menu RStudio wybierz *File*, a następnie *Import Dataset* lub skorzystaj ze skrótu klawiszowego *Ctrl+Shift+I*.

2. Wybierz opcję *From Text (base)* lub *From Text (readr)* w zależności od twoich potrzeb. Podobne działanie ma przycisk Import Dataset w panelu *Environment* lub pozycja *import dataset* z menu podręcznego po kliknięciu na plik `csv` w panelu plików 

3. Przeglądaj swoje pliki na komputerze i wybierz plik `csv`, który chcesz zaimportować. Kliknij *Open* lub *OK*.
4. Skonfiguruj opcje importu, takie jak separator kolumn, nagłówki w pierwszym wierszu czy wybór kolumn do zaimportowania. Kliknij *Import*.
5. Podaj nazwę dla nowego obiektu danych, który zostanie utworzony po imporcie. Kliknij *OK*.

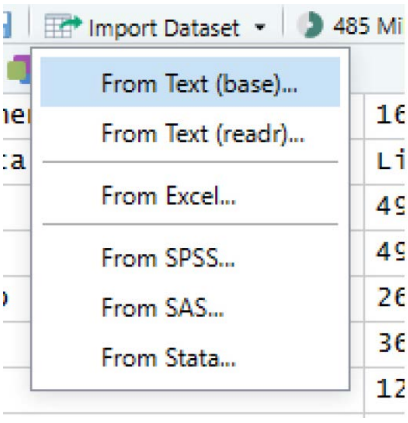


Rys. 1.16. „Graficzny" import pliku w formacie `csv` w RStudio

6. Po zakończeniu importu dane zostaną zaimportowane do środowiska R jako nowy obiekt. Można go znaleźć w panelu *Environment*.

Taki sposób importu jest niewątpliwe wygodny dla początkujących użytkowników R/RStudio, ma jednak dwie wady. Import musi być dokonywany zawsze na początku sesji, podczas gdy umieszczenie odpowiednich komend w kodzie skryptu jest czynnością jednorazową. Poza tym w obecnych wersjach R i pakietu `readr` nie ma możliwości zaimportowania pierwszej kolumny danych jako etykiet wierszy.

W podobny sposób RStudio automatyzuje import plików Excela i popularnych środowisk statystycznych SPSS, SAS czy Stata (por. rys 1.17).



Rys. 1.17. Menu importu plików w RStudio

1.6. Złożone konstrukcje programistyczne

1.6.1. Instrukcja warunkowa

```
if (warunek) instrukcja
if (warunek){
   instrukcja1
   instrukcja2
   ...
   instrukcjan
}
if (warunek){
   instrukcja1
   instrukcja2
   ...
   instrukcjan
}
else{
   instrukcja_alternatywna1
   instrukcja_alternatywna2
   ...
   instrukcja_alternatywnan
}
```

Instrukcja (pierwsza postać) lub ciąg instrukcji (druga postać) jest wykonywana tylko wtedy, gdy spełniony jest warunek. W przeciwnym wypadku, jeśli w instrukcji określony jest blok `else` (trzecia postać), wykonywany jest ciąg instrukcji alternatywnych. Poniższe instrukcje sprawdzają, czy liczba losowa z przedziału od 1 do 100 jest parzysta:

```
liczba <- sample(1:100, 1)
if (liczba%%2==0){
 print("Wylosowana liczba parzysta")
}
else{
 print("Wylosowana liczba nieparzysta")
}
```

1.6.2. Pętla for

Pętle wykorzystujemy, jeżeli podczas analizy zachodzi potrzeba wykonania tych samych czynności dla różnych wartości określonego parametru (najczęściej są to kolejne wartości z określonego przedziału, choć możliwe jest przechodzenie pętli poprzez wartości dowolnego wektora).

Przykładowo, wyświetlenie *i*-tych potęg dwójki dla *i* z przedziału od 1 do 10 może być zrealizowane poprzez skrypt:

```
for(i in 1:10){
 print(paste("2 do potęgi ",i,"wynosi",2^i))
}
```

Natomiast obliczenie sinusa dla 0,30,45,60 i 90 stopni wykonuje pętla

```
for(i in c(0,30,45,60,90)){
 print(paste("sinus ",i,"wynosi",sin(pi*i/180)))
}
```

Wewnątrz pętli może być wykonywany dowolny ciąg instrukcji. W następnym przykładzie pętla jest wykonana do przeprowadzania analizy skupień metodą *k*-średnich dla liczby klas zmieniającej się od 2 do 10 i wizualizacji wyników na jednym wykresie:

```
library(mlbench)
set.seed(123456)
dane<-mlbench.shapes(500)$x
windows()
par(mfrow=c(3,3))
for(i in 2:10){
   klasy<-kmeans(dane,i)$cluster
   kolory<-rainbow(i) [klasy]
   plot(dane,col=kolory,main=paste("Podział na",i,"klas(y)"))
}
```

A obliczenie indeksów Calińskiego-Harabasza dla tych klasyfikacji realizuje następujący fragment:

```
library(mlbench)
library(clusterSim)
set.seed(123456)
dane<-mlbench.shapes(500)$x
for(i in 2:10){
   klasy<-kmeans(dane,i)$cluster
   ch<-index.G1(dane,klasy)
print(paste("Indeks Calińskiego Harabasza przy podziale na ",i,
  "klas(y) wynosi ",ch))
}
```

1.6.3. Pętla z wyszukaniem danych spełniających określony warunek

Pętle są często łączone z instrukcjami warunkowymi, w celu znalezienia określenia, które elementy zbioru spełniają zadany warunek znalezienia minimalnej lub maksymalnej wartości w zadanym zbiorze, znalezienia pierwszej wartości spełniającej zadany warunek itp.

```
library(mlbench)
library(clusterSim)
set.seed(123456)
dane<-mlbench.shapes(500)$x
maksymalny_indeks_CH<-0
optymalna_liczba_klas<-2
for(i in 2:10){
    klasy<-kmeans(dane,i)$cluster
    ch<-index.G1(dane,klasy)
    if(ch>maksymalny_indeks_CH){
    optymalna_liczba_klas<-i
    maksymalny_indeks_CH<-ch
    }
```

```
}
print(paste("Optymalna liczba klas znaleziona na podstawie indeksu CH to ",
  optymalna_liczba_klas))
```

W języku R istnieje wiele gotowych funkcji ułatwiających wyszukiwanie danych spełniających określonych warunek, takich jak `which.min` i `which.max`, ułatwiających pisanie pętli takiego typu. Powyższy fragment może być uproszczony:

```
library(mlbench)
library(clusterSim)
set.seed(123456)
dane<-mlbench.shapes(500)$x
indeksy<-rep(0,10)
for(i in 2:10){
   klasy<-kmeans(dane,i)$cluster
   ch<-index.G1(dane,klasy)
   indeksy[i]<-ch
}
print(paste("Optymalna liczba klas znaleziona na podstawie indeksu CH to ",
  which.max(indeksy)))
```

1.6.4. Pętla z agregacją danych

Częstym zastosowaniem pętli jest agregacja danych (całościowa lub częściowa). Obliczenie sumy elementów ze zbioru może być zrealizowane przez pętlę:

```
wektor<-c(1,4,6,2,3,4,5,6,7,8,2,3,4,5,6,7,8,9,1,2,3)
suma<-0
for(w in wektor){
 suma=suma+w }
print(paste("Suma elementów wektora wynosi",suma))
```

W języku R pętla ta może być uproszczona do postaci:

```
wektor<-c(1,4,6,2,3,4,5,6,7,8,2,3,4,5,6,7,8,9,1,2,3)
print(paste("Suma elementów wektora wynosi",sum(wektor)))
```

A obliczenie, ile w wektorze razy występuje każda z cyfr, realizuje fragment

```
wektor<-c(1,4,6,2,3,4,5,6,7,8,2,3,4,5,6,7,8,9,1,2,3)
wystapienia<-array(0,c(max(wektor),2))
wystapienia[,1]<-1:max(wektor)
for(w in wektor){
 wystapienia[w,2]<-wystapienia[w,2]+1
}
print("Liczba wystąpień poszczególnych elementów wektora to:")
print(wystapienia)
```

który z kolei w języku R może być uproszczony jako:

```
wektor<-c(1,4,6,2,3,4,5,6,7,8,2,3,4,5,6,7,8,9,1,2,3)
print("Liczba wystąpień poszczególnych elementów wektora wynosi ")
wystapienia<-cbind(1:max(wektor),as.matrix(table(wektor)))
print(wystapienia)
```

1.6.5. Pętla `while`

```
while(warunek) instrukcja
while(warunek){
   instrukcja1
   instrukcja2
   ...
   instrukcjan
}
```

Instrukcje są wykonywane do momentu, gdy warunek przestaje być spełniony. Na przykład realizacja ciągu klasyfikacji z poprzedniego podpunktu za pomocą pętli `while` może wyglądać następująco:

```
library("clusterSim")
data(data_ratio)
x <- as.matrix(data_ratio)
liczba_klas <- 3
while(liczb_klas<7){
   cl <- kmeans(x, liczba_klas, 50)
   print(cl$cluster)
   liczba_klas <- liczba_klas+1
}
```

1.7. Inne przydatne funkcje

Tabela 1.8. Wybrane funkcje pomocnicze języka R

`formula`	Funkcja używana wtedy, gdy argument przekazywany do innej funkcji nie ma być traktowany jako łańcuch tekstowy, ale jako formuła (zazwyczaj zawierająca ~). Aby np. przeprowadzić analizę czynnikową dotyczącą wszystkich zmiennych tablicy danych `z`, nie trzeba w formule przekazywać do funkcji `factanal` wszystkich nazw zmiennych, a wystarczy użyć funkcji zgodnie ze składnią: `> factanal(formula(paste("~",paste(names(z),collapse="+"))),3)`.

`is.na`	Zwraca ciąg wartości, w którym `TRUE` oznacza, że na danej pozycji w wektorze lub tablicy wartość elementu jest nieokreślona.
	Przykładowo
	`> tablica8 <- array(,c(2,2))`
	`> tablica8[1,1] <- 3`
	`> tablica8[2,2] <- 4`
	`> print(tablica8)`
	`     [,1] [,2]`
	`[1,]    3   NA`
	`[2,]   NA    4`
	`> is.na(tablica8)`
	zwraca
	`      [,1]  [,2]`
	`[1,] FALSE  TRUE`
	`[2,] TRUE  FALSE`
	Funkcja może być też wykorzystana do zmiany wszystkich wartości nieokreślonych np. na 5
	`> tablica8[is.na(tablica8)] <- 5`
	`> print(tablica8)`
	`    [,1]  [,2]`
	`[1,]   3    5`
	`[2,]   5    4`

`is.nan`	Zwraca ciąg wartości, w którym `TRUE` oznacza, że na danej pozycji w wektorze lub tablicy, przy jej obliczaniu, wystąpił błąd. Przykładowo, aby podzielić całkowicie przez siebie odpowiadające sobie elementy dwóch wektorów, z założeniem, że 0 podzielone przez 0 nie ma generować błędu, a wynikiem takiej operacji ma być również 0, można użyć np. instrukcji:
	`wektor12 <- c(3,0,5,3,5,6,2,5,7,9,13,12,1,5)`
	`> wektor12 <- c(3,0,5,3,5,6,2,5,7,9,13,12,0,5,0)`
	`> wektor13 <- c(2,0,2,1,3,2,1,5,3,2,6,4,0,6,0)`
	`> wynik <- wektor12%/%wektor13`
	`> print(wynik)`
	`[1] 1 NaN 2 3 1 3 2 1 2 4 2 3 NaN 0 NaN`
	`> wynik[is.nan(wynik)] <- 0`
	`> print(wynik)`
	`[1] 1 0 2 3 1 3 2 1 2 4 2 3 0 0 0`

`is.null`	Funkcja sprawdzająca, czy dany obiekt ma wartość `NULL`. Do tego celu nie można użyć zwykłego operatora porównania, gdyż wygenerowałoby to błąd
	`argument is of length zero.`
	Żeby więc sprawdzić, czy obiekt nie jest pusty, należy użyć konstrukcji
	`> if(!is.null(obiekt))`
	`> {`
	`> …`
	`> }`

`list`	Wyświetlenie wszystkich obiektów środowiska R aktualnie znajdujących się w pamięci operacyjnej. W wersji okienkowej odpowiada poleceniu menu *Misc* | *List objects*.
`options`	Wyświetlenie (jeśli funkcja jest wywołana bez argumentów) lub ustalenie (w postaci `options(parametr=wartość)`) parametrów środowiska R, takich jak domyślny edytor (parametr `editor`), domyślne urządzenie graficzne (parametr `device`), liczba kolumn w wydruku (parametr `width`), sposobu wyświetlania ostrzeżeń (parametr `warn`) i in. Szczególnie istotne dla polskich użytkowników środowiska R jest wykorzystanie tej funkcji do ustalenia, że symbolem używanym do wyświetlania separatora dziesiętnego ma być przecinek, zamiast domyślnej kropki poprzez polecenie: `options(OutDec=",")`. Ta postać instrukcji będzie wykorzystywana wielokrotnie w przykładach z następnych rozdziałów.

`paste`	Konkatenacja (łączenie tekstu) zmiennych. Funkcja często używana w połączeniu z `print`, gdy wydruk ma być złożony ze stałego tekstu i wartości zmiennej.
	Przykładowo instrukcje
	`> zm1 <- zm2<-2`
	`> wynik <- zm1*zm2`
	`> print(paste(zm1,"*", zm2, "wynosi", wynik))`
	wyświetlą napis
	`"2 * 2 wynosi 4"`
	Polecenie
	`> napis<-paste(LETTERS,collapse="_")`
	utworzy zmienną zawierającą łańcuch tekstowy
	[1] "A_B_C_D_E_F_G_H_I_J_K_L_M_N_O_P_Q_R_S_T_U_V_W_X_Y_Z"
	Natomiast pętla
	`for(i in 1:5){`
	`print(paste(i, "do kwadratu rowna się ", i^2))`
	`}`
	wyświetli
	`[1] "1 do kwadratu rowna się 1"`
	`[1] "2 do kwadratu rowna się 4"`
	`[1] "3 do kwadratu rowna się 9"`
	`[1] "4 do kwadratu rowna się 16"`
	`[1] "5 do kwadratu rowna się 25"`

`rm`	Usunięcie wybranego obiektu lub wszystkich obiektów z pamięci operacyjnej `> rm(list=ls(all=TRUE))` W tej formie funkcja odpowiada poleceniu menu *Misc* | *Remove all objects* w wersji okienkowej środowiska R.

`sink`	Przekierunkowanie aktualnego strumienia wyjściowego do pliku. Użycie funkcji powoduje, że wszystkie instrukcje `print` w wykonywanym kodzie oraz instrukcje wyświetlające dane w trybie interaktywnym przesyłają tekst nie na ekran, ale do wskazanego w funkcji pliku (np. `sink("sesja.txt")`). Zakończenie przekierunkowania nastąpi po wydaniu polecenia `sink()`.

`summary`	Funkcja wyświetlająca najważniejsze zbiorcze informacje o obiektach. Co dokładnie wyświetla, zależy od klasy wyświetlanego obiektu. Na przykład dla zmiennej nominalnej ze zdefiniowanymi kategoriami `summary` wyświetla informację o tym, ile jest elementów danej kategorii
	`> faktor2 <- factor(c(1,2,3,4,2,3,4,2,3,4,1,2,3))`
	`> summary(faktor2)`
	`1 2 3 4`
	`2 4 4 3`

`switch`	Formuła wielowariantowa
	`> z <- switch(typnormalizacji,`
	`beznormalizacji=cbind(rezultat, x[, i]),`
	`standaryzacja=cbind(rezultat, (x[, i]- mean(x[, i]))/(sd(x[, i]))),`
	`unitaryzacja=(x[, i] - min(x[, i]))/(max(x[, i]) - min(x[, i])))`

`write`	Działa podobnie jak `print`, z tą różnicą, że dane są zapisywane do pliku, a nie wyświetlane na ekranie.

Źródło: opracowanie własne na podstawie (Dalgaard, 2002; Everitt, 2005; Everitt i Hothorn, 2006).

1.8. Tworzenie własnych funkcji

Podobnie jak w innych językach programowania funkcje to fragmenty kodu o określonej nazwie i liście argumentów zwracające wartości w postaci pojedynczej zmiennej lub obiektu złożonego. Przykładowa definicja funkcji sprawdzającej, czy dana liczba jest liczbą pierwszą, w języku R może wyglądać następująco:

```
czy.pierwsza <- function(x){
 !sum(x%%2:(x-1)==0) || x==2
}
# Jeśli którakolwiek liczba z przedziału <2,x-1> jest dzielnikiem
 x to zwracany jest fałsz
```

Żadne zmienne używane w funkcjach, z wyjątkiem wartości zwracanej, nie są widoczne na zewnątrz. Jedynym wyjątkiem od tej sytuacji jest wykorzystanie zmiennych globalnych (operator `<<-` , przypisanie `assign` do globalnego środowiska) (Gągolewski, 2014, s. 439-440).

Jeżeli funkcja ma zwracać więcej niż jedną wartość, to wygodnie w instrukcji return skorzystać z listy (instrukcja `list` połączona z `return`). Przykładowo procedura znalezienia największej wspólnej wielokrotności dwóch liczb może być zaimplementowana w funkcji

```
NWD<-function(x,y){
  NWD<-1
  n<-2
  liczba_iteracji<-0
  skladowe<-NULL
  while (n<min(x,y)){
    if(czy.pierwsza(n) && (x%%n)==0 && (y%%n)==0){
      NWD=NWD*n
      x=x%/%n
      y=y%/%n
      skladowe<-c(skladowe,n)
```

```
    }
    else{
      n=n+1
    }
    liczba_iteracji<-liczba_iteracji+1
         #zmienna globalna
    g_liczba_iteracji<<- liczba_iteracji
         #równoważnie
    #assign(g_liczba_iteracji,liczba_iteracji,.GlobalEnv)
  }
  return (list(liczba=NWD,skladowe=skladowe))
}
n<-1716
m<-2112
nwd<-NWD(m,n)
print(paste("NWD ",n,"i",m,"wynosi",nwd$liczba))
print(paste("otrzymany jako iloczyn liczb"))
print(nwd$skladowe)
print(paste("wynik osiągnięto po",g_liczba_iteracji,"krokach pętli"))
```

Literatura

Biecek, P. (2017). *Przewodnik po pakiecie R*. Oficyna Wydawnicza GiS.

Crawley, M. J. (2007). *The R Book*. John Wiley & Sons.

Dalgaard, P. (2002). *Introductory Statistics with R*. Springer.

Dudek, A. (2009). Wprowadzenie do programu R. W: M. Walesiak, E. Gatnar (red.), *Statystyczna analiza danych z wykorzystaniem programu R* (s. 13-61). Wydawnictwo Naukowe PWN.

Everitt, B. S. (2005). *An R and S-PLUS® Companion to Multivariate Analysis*. Springer.

Everitt, B. S. i Hothorn, T. (2006). *A Handbook of Statistical Analyses Using R*. Chapman & Hall/CRC.

Gągolewski, M. (2014). *Programowanie w języku R. Analiza danych, obliczenia, symulacje*. Wydawnictwo Naukowe PWN.

Komsta, Ł. (2004). *Wprowadzenie do języka R*. **http://cran.r-project.org/doc/contrib/Komsta-Wprowadzenie.pdf**

Kopczewska, K., Kopczewski, T. i Wójcik, P. (2016). *Metody ilościowe w R. Aplikacje ekonomiczne i finansowe*. CEDEWU.PL.

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

DOI: 10.15611/2024.45.1.02

2 API dla BDL i pakiet bdl w R

2.1. Dlaczego warto używać API BDL w R?

Interfejs programowania aplikacji API (*Application Programming Interface*) to zestaw reguł i definicji umożliwiających komunikację między różnymi programami. API pozwala programom wymieniać się danymi i funkcjami, co umożliwia tworzenie zaawansowanych aplikacji bez konieczności budowania wszystkiego od podstaw. W przypadku API BDL wymiana następuje pomiędzy instrukcjami wywołującymi API w dowolnym nowoczesnym języku programowania a serwerem, który ma dostęp do sieci wewnętrznej z infrastrukturą Banku Danych Lokalnych.

Aplikacja API BDL pozwala przeglądać i pobierać do dalszego przetwarzania pełny zakres danych znajdujących się w Banku Danych Lokalnych. Dane udostępniane są przez *webservice* typu REST w formatach XML, JSON oraz JSONAPI. Opis metod API BDL jest dostępny na stronie GUS **https://api.stat.gov.pl/Content/files/bdl/Opis_metod_API_BDL_v1.pdf.** Zakres danych udostępnianych w API i w aplikacji internetowej BDL jest taki sam. API BDL pozwala więc na pobieranie danych z banku bezpośrednio w środowisku aplikacji bez konieczności korzystania ze strony internetowej. Jest to niezwykle przydatne dla dewelopera aplikacji, ponieważ może pobierać dane w sposób całkowicie zautomatyzowany i powtarzalny. Te cechy otwierają kompletnie nowe możliwości wykorzystania danych BDL, jakich do tej pory nie było. Pobieranie danych z API BDL jest wprawdzie możliwe wprost, np. poprzez polecenie GET pakietu `httr`, jednak pakiet `bdl` daje bezpośredni dostęp do API BDL w R w postaci zestawu funkcji pobierających dane oraz funkcji pozwalających na ich wizualizację, co wydaje się znacznie wygodniejsze, zwłaszcza dla użytkownika początkującego lub średnio zaawansowanego.

2.2. Instalacja i ustawienia pakietu

Pakiet w wersji 1.0.5 (Szpadel i Kania, 2023) jest obecny w repozytorium CRAN. Można go zainstalować za pomocą komendy:

```
install.packages("bdl")
```

lub pobrać z repozytorium github:

```
remotes::install_github("statisticspoland/R_Package_to_API_BDL").
```

Chociaż nie jest to wymagane, zalecana jest rejestracja na stronie BDL API w celu pobrania klucza prywatnego, dzięki któremu będziemy w stanie pobrać więcej danych bez błędów dotyczących wykroczenia poza limit zapytań API. Klucz można uzyskać na stronie **https://api.stat.gov.pl/Home/BdlApi.** Następnie uzyskany klucz należy wprowadzić do środowiska R za pomocą komendy:

```
options(bdl.api_private_key = "klucz-użytkownika")
```

2.3. Szukanie identyfikatorów jednostek terytorialnych[1]

Pobieranie danych w pakiecie oraz API BDL opiera się na kodach – identyfikatorach danego zasobu, np. identyfikatorem (w skrócie `id`) dla miasta Wrocławia będzie `"030210564011"`, a dla zmiennej dotyczącej gęstości zaludnienia `"60559"`. Dlatego przed pobraniem musimy odszukać `id` interesujących nas zasobów. Do odnalezienia niezbędnych kodów jednostek można użyć komend:

`search_units()` – dla jednostek terytorialnych (poziomy od 1 do 6),

`search_unit_localities()` – dla jednostek terytorialnych (poziom 7) – miejscowości statystycznych.

Argumenty, jakie przybiera `search_units()`:

- `name` – szukana fraza;
- `level` – poziom jednostek terytorialnych od 0 do 6:
  0 – Polska,
  1 – makroregion,
  2 – województwo,
  3 – region,
  4 – podregion,
  5 – powiat,
  6 – gmina;
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`; opcjonalnie brak podania lat zwróci wszystkie dostępne lata;
- `kind` – rodzaj gminy: **https://bdl.stat.gov.pl/BDL/metadane/teryt/rodzaj**;
- `sort` – po której kolumnie ramka ma być sortowana; domyślnie sortuje rosnąco po `id`, ale przybiera wartości: `id`, `-id`, `name`, `-name`.

Przykład. Wyświetlenie powiatów (`level=5`) mających w nazwie „wro” (`name="wro"`) w latach 2000-2002, rodzaj gminy – gmina miejska (`kind="1"`).

```
search_units(name = "wro", level = "5", year = 2000:2002, kind = "1")
# A tibble: 2 x 6
  id           name                parentId     level kind  unitHasChanged
  <chr>        <chr>               <chr>        <int> <chr> <lgl>
1 030210423000 Powiat wrocławski   030210400000     5 1     FALSE
2 040416707000 Powiat inowrocławski 040416700000     5 1     FALSE
```

[1] Pełne nagłówki funkcji oraz opisy poszczególnych argumentów ujęte są w pakiecie `bdl` dostępnym w pliku `bdl.pdf` na stronie **https://cran.r-project.org/web/packages/bdl/index.html.**

Jest możliwość wywołania funkcji `search_units()` z pustym parametrem `name`, aby wyświetlić wszystkie jednostki terytorialne z danego poziomu:

- wszystkie województwa: `search_units(name="", level="2")`;
- wszystkie gminy: `search_units(name="", level="6")`.

Dla jednostek terytorialnych – miejscowości statystycznych (poziom 7)

Miejscowość statystyczna to wyodrębniony do celów statystycznych zespół kilku miejscowości obejmujący z reguły wieś oraz przyległe do niej przysiółki i inne mniejsze miejscowości, dla których łącznie zbierane są i opracowywane dane statystyczne. Jest on określony wspólną nazwą, przeważnie większej miejscowości. Dla miejscowości statystycznych pakiet przewiduje osobne funkcje z dodatkiem „locality". Dlatego skorzystamy z funkcji:

`search_unit_localities()`

Argumenty, które przyjmuje `search_unit_localities()`:

- `name` – szukana fraza,
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`. Opcjonalnie brak podania lat zwróci wszystkie dostępne lata.

Przykład. Wszystkie miejscowości statystyczne w latach 2000-2002 z `name="pol"` w nazwie, posortowane po nazwie alfabetycznie.

```
search_unit_localities(name = "pol", year = 2000:2002, sort = "name")
# A tibble: 425 × 6
   id                  name       parentId     level kind  unitHasChanged
   <chr>               <chr>      <chr>        <int> <chr> <lgl>
 1 060610919082-0110674 Adampol    060610919082     7 2     FALSE
 2 071412934062-0673124 Adampol    071412934062     7 2     FALSE
 3 071427133032-0675376 Adampol    071427133032     7 2     FALSE
 4 051011716032-0536829 Agnopol    051011716032     7 2     FALSE
 5 051011506022-0413050 Andrespol  051011506022     7 2     FALSE
 6 060611006062-0104515 Anielpol   060611006062     7 2     FALSE
 7 023015706025-0199303 Annapol    023015706025     7 5     FALSE
 8 060611108065-0383136 Annopol    060611108065     7 5     FALSE
 9 023015707062-0202347 Annopol    023015707062     7 2     FALSE
10 060611003142-0110993 Annopol    060611003142     7 2     FALSE
# ℹ 415 more rows
```

Drugim sposobem na znalezienie kodów jednostek jest listowanie jednostek podrzędnych `get_units()`. Zwraca listę wszystkich jednostek podrzędnych dla jednostki podanej w `parentId`. Wywołane bez żadnych parametrów zwraca listę wszystkich jednostek terytorialnych (4400 rekordów) i zajmuje to około minuty.

Argumenty dla `get_units()`:

- `parentId` – kod NUTS jednostki nadrzędnej;
- `level` – poziom jednostek od 1 do 6.

Przykład. Wyświetlenie jednostek podrzędnych dla `000000000000` (Polska) – wszystkie województwa `level="2"`.

```
get_units(parentId = "000000000000", level = "2")
# A tibble: 16 x 5
   id           name                 parentId      level unitHasChanged
   <chr>        <chr>                <chr>         <int> <lgl>
 1 011200000000 MAŁOPOLSKIE          010000000000      2 TRUE
 2 012400000000 ŚLĄSKIE              010000000000      2 TRUE
 3 020800000000 LUBUSKIE             020000000000      2 FALSE
 4 023000000000 WIELKOPOLSKIE        020000000000      2 FALSE
 5 023200000000 ZACHODNIOPOMORSKIE   020000000000      2 FALSE
 6 030200000000 DOLNOŚLĄSKIE         030000000000      2 FALSE
 7 031600000000 OPOLSKIE             030000000000      2 FALSE
 8 040400000000 KUJAWSKO-POMORSKIE   040000000000      2 FALSE
 9 042200000000 POMORSKIE            040000000000      2 FALSE
10 042800000000 WARMIŃSKO-MAZURSKIE  040000000000      2 FALSE
11 051000000000 ŁÓDZKIE              050000000000      2 FALSE
12 052600000000 ŚWIĘTOKRZYSKIE       050000000000      2 FALSE
13 060600000000 LUBELSKIE            060000000000      2 FALSE
14 061800000000 PODKARPACKIE         060000000000      2 TRUE
15 062000000000 PODLASKIE            060000000000      2 FALSE
16 071400000000 MAZOWIECKIE          070000000000      2 FALSE
```

Inne przykłady:

- wszystkie powiaty: `get_units(level="5")`;
- wszystkie gminy w województwie dolnośląskim: `get_units(parentId = "030200000000", level = "6")`.

Dla miejscowości statystycznych używamy funkcji `get_unit_localities()`, która wymaga podania argumentu `parentId:`

- `parentId` – kod NUTS jednostki nadrzędnej.

Przykład. Wyświetlenie miejscowości statystycznych dla jednostki `030210106062` (Jeżów Sudecki).

```
get_unit_localities(parentId = "030210106062")
# A tibble: 8 × 6
  id                    name          parentId     level kind  unitHasChanged
  <chr>                 <chr>         <chr>        <int> <chr> <lgl>
1 030210106062-0189747 Chrośnica     030210106062     7 2     FALSE
2 030210106062-0189753 Czernica      030210106062     7 2     FALSE
3 030210106062-0189760 Dziwiszów     030210106062     7 2     FALSE
4 030210106062-0189776 Janówek       030210106062     7 2     FALSE
5 030210106062-0189782 Jeżów Sudecki 030210106062     7 2     FALSE
6 030210106062-0189799 Płoszczyna    030210106062     7 2     FALSE
7 030210106062-0189813 Siedlęcin     030210106062     7 2     FALSE
8 030210106062-0189820 Wrzeszczyn    030210106062     7 2     FALSE
```

2.4. Szukanie identyfikatorów zmiennych (cech)

Kody zmiennych można odszukać na dwa sposoby. Pierwszym z nich jest wypisanie zmiennych dla danej podgrupy (najniższy poziom tematu) lub szukając bezpośrednio w wymiarach zmiennych. Tematy podzielone są na:

- K – kategorie, np. K3,
- G – grupy, np. G7,
- P – podgrupy, np. P2425.

W celu pobrania zmiennych należy najpierw znaleźć kod interesującej nas podgrupy. Możemy wyświetlać kolejne poziomy tematów za pomocą `get_subjects()` lub szukać bezpośrednio w nazwach tematów za pomocą `search_subjects()`. Komenda `get_subjects()` działa jedynie dla kategorii, grup i podgrup (tematów). Do wyświetlenia zmiennych dla danej podgrupy należy użyć `get_variables()`.

`get_subjects()` – wypisanie wszystkich kategorii

```
# A tibble: 33 × 5
 id    name                                    hasVariables children   levels
 <chr> <chr>                                   <lgl>        <list>     <list>
 1 K1   PODZIAŁ TERYTORIALNY                    FALSE         <chr [2]>  <int [3]>
 2 K2   SAMORZĄD TERYTORIALNY                   FALSE        <chr [11]> <int [3]>
 3 K3   LUDNOŚĆ                                 FALSE        <chr [8]>  <int [5]>
 4 K4   RYNEK PRACY                             FALSE        <chr [13]> <int [4]>
 5 K6   ROLNICTWO                               FALSE        <chr [8]>  <int [2]>
 6 K8   TRANSPORT I ŁĄCZNOŚĆ                    FALSE        <chr [15]> <int [3]>
 7 K9   STAN I OCHRONA ŚRODOWISKA               FALSE        <chr [19]> <int [4]>
 8 K10  NAUKA I TECHNIKA. SPOŁECZEŃSTWO INFORMACYJNE FALSE   <chr [6]>  <int [2]>
 9 K11  GOSPODARKA MIESZKANIOWA I KOMUNALNA     FALSE        <chr [11]> <int [4]>
10 K12  PRZEMYSŁ I BUDOWNICTWO                  FALSE        <chr [7]>  <int [4]>
# i 23 more rows
```

`get_subjects("K3")` – grupy podrzędne dla tematu K3 – LUDNOŚĆ

```
# A tibble: 8 × 6
 id    parentId name                           hasVariables   children      levels
 <chr> <chr>  <chr>                          <lgl>          <list>        <list>
1 G7    K3     STAN LUDNOŚCI                  FALSE          <chr [20]>    <int [4]>
2 G8    K3     MIGRACJE WEWNĘTRZNE I ZAGRANICZNE FALSE       <chr [14]>    <int [3]>
3 G10   K3     GOSPODARSTWA DOMOWE            FALSE          <chr [9]>     <int [1]>
4 G534  K3     URODZENIA I ZGONY              FALSE          <chr [18]>    <int [4]>
5 G535  K3     MAŁŻEŃSTWA, ROZWODY I SEPARACJE FALSE         <chr [14]>    <int [4]>
6 G557  K3     PROGNOZY                       FALSE          <chr [2]>     <int [1]>
7 G564  K3     DANE ARCHIWALNE                FALSE          <chr [10]>    <int [3]>
8 G655  K3     PROGNOZA LUDNOŚCI              FALSE          <chr [0]>     <int [0]>
```

`get_subjects("G7")` – podgrupy podrzędne dla G7 – STAN LUDNOŚCI

```
# A tibble: 20 × 6
   id    parentId name                                    hasVariables children levels
<chr>    <chr>   <chr>                                    <lgl>        <list>   <list>
 1 P1336  G7  Ludność wg miejsca zamieszkania i płci w p…  TRUE        <list>   <int>
 2 P1341  G7  Ludność wg pojedynczych roczników wieku i …  TRUE        <list>   <int>
 3 P1342  G7  Ludność w wieku przedprodukcyjnym (17 lat …  TRUE        <list>   <int>
 4 P2137  G7  Ludność wg grup wieku i płci                 TRUE        <list>   <int>
 5 P2425  G7  Gęstość zaludnienia oraz wskaźniki           TRUE        <list>   <int>
 6 P2426  G7  Wskaźniki obciążenia demograficznego         TRUE        <list>   <int>
 7 P2427  G7  Udział ludności wg ekonomicznych grup wiek…  TRUE        <list>   <int>
 8 P2462  G7  Ludność wg płci oraz w podziale na miasto …  TRUE        <list>   <int>
 9 P2463  G7  Ludność w miastach w % ogółu ludności (dan…  TRUE        <list>   <int>
10 P2577  G7  Ludnośćw wieku przedprodukcyjnym (14 lat …  TRUE        <list>   <int>
11 P2730  G7  Przeciętne dalsze trwanie życia             TRUE        <list>   <int>
12 P2914  G7  Ludność w gminach bez miast na prawach pow…  TRUE        <list>   <int>
13 P3361  G7  Ludność w wieku przedprodukcyjnym (17 lat …  TRUE        <list>   <int>
14 P3429  G7  Współczynnik feminizacji                    TRUE        <list>   <int>
15 P3447  G7  Ludność wg funkcjonalnych grup wieku i płc…  TRUE        <list>   <int>
16 P3472  G7  Ludność wg pojedynczych roczników wieku i …  TRUE        <list>   <int>
17 P3813  G7  Mediana wieku ludności według płci w podzi…  TRUE        <list>   <int>
18 P3814  G7  Mediana wieku ludności według płci           TRUE        <list>   <int>
19 P3895  G7  Oczekiwane trwanie życia w zdrowiu           TRUE        <list>   <int>
20 P3991  G7  Rezydenci w wieku przedprodukcyjnym (17 la…  TRUE        <list>   <int>
```

`search_subjects()` – wyświetla tematy z podaną frazą w nazwie

```
search_subjects("Gęstość zaludnienia")
# A tibble: 1 × 6
  id    parentId name                           hasVariables children   levels
  <chr> <chr>    <chr>                          <lgl>        <list>     <list>
1 P2425 G7   Gęstość zaludnienia oraz wskaźniki  TRUE        <list [0]> <int [2]>
```

Aby wyświetlić zmienne dla P2425 – Gęstość zaludnienia oraz wskaźniki

```
get_variables("P2425")
# A tibble: 6 × 6
      id subjectId n1                                level measureUnitId measureUnitName
    <int> <chr>     <chr>                             <int>         <int> <chr>
1   60559 P2425     ludność na 1 km2                      6            26 osoba
2  458238 P2425     gęstość zaludnienia powierzchni z…    6            26 osoba
3  458603 P2425     zmiana liczby ludności na 1000 mi…    6            26 osoba
4 1645341 P2425     ludność w tysiącach                   6            23 tys. osób
5 1645343 P2425     ludność w tysiącach kobiety           6            23 tys. osób
6 1645344 P2425     ludność w tysiącach mężczyźni         6            23 tys. osób
```

Można także skorzystać z wyszukiwania po nazwie `search_variables()`.

Przykład. Wyświetlenie zmiennych zawierających „samochody".

```
search_variables(name = "samochody")
# A tibble: 104 × 7
      id subjectId n1                    n2           level measureUnitId measureUnitName
   <int> <chr>     <chr>                 <chr>        <int>         <int> <chr>
 1  1688 P1883     samochody osobowe     ogółem           6             8 szt.
 2  1689 P1883     samochody osobowe     gospodarstw…     6             8 szt.
 3  1690 P1883     samochody ciężarowe   ogółem           6             8 szt.
 4  1691 P1883     samochody ciężarowe   gospodarstw…     6             8 szt.
 5  4670 P1733     samochody osobowe na … NA              3             8 szt.
 6  4671 P1733     samochody osobowe na … NA              3             8 szt.
 7  4672 P1733     samochody sanitarne   NA               3             8 szt.
 8  4681 P1733     samochody ciężarowo -… NA              5             8 szt.
 9  4683 P1733     samochody ciężarowe n… NA              3             8 szt.
10  4684 P1733     samochody ciężarowe n… NA              3             8 szt.
# ℹ 94 more rows
```

Argumenty, jakie przyjmuje `search_variables()`:

- `name` – szukana fraza;
- `subjectId` – kod nadrzędnego tematu;
- `level` – poziom zasilenia danych; filtruje zwracane zmienne do podanego poziomu i poziomów podrzędnych;
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`; opcjonalnie brak podania lat zwróci wszystkie dostępne lata.

Przykład. Wyszukanie zmiennych z frazą „samochody", z zasileniem na poziomie powiatów (`level = "5"`) w latach 2005-2007.

```
search_variables(name = "samochody", level = "5", year = 2005:2007)
# A tibble: 6 × 6
     id subjectId n1                                  level measureUnitId measureUnitName
  <int> <chr>     <chr>                               <int>         <int> <chr>
1  4681 P1733     samochody ciężarowo-osobowe             5             8 szt.
2 21768 P1733     samochody specjalne (łącznie z sani…    5             8 szt.
3 32556 P1733     samochody ciężarowe                     5             8 szt.
4 32561 P1733     samochody osobowe                       5             8 szt.
5 60531 P2420     samochody ciężarowe na 1000 ludności    5             8 szt.
6 60533 P2420     samochody osobowe na 1000 ludności      5             8 szt.
```

2.5. Pobieranie danych dla pojedynczej jednostki i wielu zmiennych

Tak samo jak w banku dzięki pakietowi możemy pobierać dane dla:

- pojedynczej jednostki, używając wielu zmiennych,
- pojedynczej zmiennej i wielu jednostek.

W celu ułatwienia pobierania danych funkcje pobierania, oprócz przyjmowania pojedynczych kodów jednostek lub zmiennych, przyjmują także ich wektory, a wynik komendy będzie złączeniem ramek danych dla pojedynczych ich wyników.

Chcąc pobrać dane dla pojedynczej jednostki lub kilku pojedynczych jednostek, użyjemy `get_data_by_unit()`. Argumenty dla `get_data_by_unit()`:

- `unitId` – 12 znakowy kod NUTS jednostki, np. `"000000000000"`, lub wektor kodów, np. `c("000000000000","030200000000")`;
- `varId` – pojedynczy kod zmiennej lub wektor zmiennych, np. `c("420","3643")`;
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`; opcjonalnie brak podania lat zwróci wszystkie dostępne lata;
- `type` – domyślnie `"code"`, opcjonalnie `"label"`; użycie `"label"` dodaje dodatkowe kolumny z opisem tekstowym kodów;
- `aggregateId` – poziom agregatu (zasilania danych); użyj funkcji `get_aggregates()` lub wejdź na stronę **https://bdl.stat.gov.pl/BDL/metadane/agregaty**, by uzyskać dodatkowe informacje.

Przykład. Wybranie danych dla województwa zachodniopomorskiego dla zmiennych `3643` (radni województwa w wieku 30-39 lat) i `1645343` (ludność kobiet w tysiącach) w latach 1999-2003, wraz z dodatkowymi etykietami (`type = "label"`).

```
get_data_by_unit(unitId = "023200000000", varId = c("3643", "1645343"),
  year = 1999:2003, type = "label")
# A tibble: 10 × 9
   id     year    val variableName  measureUnitId measureName attrId attributeDescription
   <chr>  <chr> <dbl> <chr>                <int> <chr>        <int> <chr>
 1 3643   1999      1  Radni wojewó…           26 osoba            1 ""
 2 3643   2000      2  Radni wojewó…           26 osoba            1 ""
 3 3643   2001      4  Radni wojewó…           26 osoba            1 ""
 4 3643   2002      3  Radni wojewó…           26 osoba            1 ""
 5 3643   2003      3  Radni wojewó…           26 osoba            1 ""
 6 1645343 1999   869. Gęstość zalu…           23 tys. osób        1 ""
 7 1645343 2000   870. Gęstość zalu…           23 tys. osób        1 ""
 8 1645343 2001   871. Gęstość zalu…           23 tys. osób        1 ""
 9 1645343 2002   871. Gęstość zalu…           23 tys. osób        1 ""
10 1645343 2003   870. Gęstość zalu…           23 tys. osób        1 ""
# i 1 more variable: lastUpdate <chr>
```

oraz bez argumentu `type`:

```
get_data_by_unit(unitId = "023200000000", varId = c("3643", "1645343"), year = 1999:2003)
# A tibble: 10 × 7
   id      year    val measureUnitId attrId attributeDescription lastUpdate
   <chr>   <chr> <dbl>         <int>  <int> <chr>                 <chr>
 1 3643    1999      1            26      1 ""                    2023-05-09T14:31:39.107
 2 3643    2000      2            26      1 ""                    2023-05-09T14:31:39.107
 3 3643    2001      4            26      1 ""                    2023-05-09T14:31:39.107
 4 3643    2002      3            26      1 ""                    2023-05-09T14:31:39.107
 5 3643    2003      3            26      1 ""                    2023-05-09T14:31:39.107
 6 1645343 1999   869.            23      1 ""                    2023-05-31T09:22:22.383
 7 1645343 2000   870.            23      1 ""                    2023-05-31T09:22:22.383
 8 1645343 2001   871.            23      1 ""                    2023-05-31T09:22:22.383
 9 1645343 2002   871.            23      1 ""                    2023-05-31T09:22:22.383
10 1645343 2003   870.            23      1 ""                    2023-05-31T09:22:22.383
```

Dla miejscowości statystycznych skorzystamy z tej samej funkcji z dodatkiem `locality`. Argumenty dla `get_data_by_unit_locality()`:

- `unitId` – 12-znakowy kod NUTS jednostki z 7-znakowym indywidualnym identyfikatorem oddzielonym myślnikiem, np. `"030210106062-0189782"`, lub ich wektor;
- `varId` – pojedynczy kod zmiennej lub wektor zmiennych, np. `c("420","3643")`;
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`; opcjonalnie brak podania lat zwróci wszystkie dostępne lata;
- `type` – typ zwróconych danych; domyślnie `"code"`, opcjonalnie `"label"`. Użycie `"label"` dodaje dodatkowe kolumny z opisem tekstowym kodów.

Przykład. Pobranie danych dotyczących mieszkań oddanych do użytkowania (`varId="415"`) dla Jeżowa Sudeckiego w latach 2008-2012 z dodatkowymi etykietami.

```
get_data_by_unit_locality(unitId = "030210106062-0189782",
  year = 2008:2012, varId = "415", type = "label")
# A tibble: 5 × 9
    id year   val variableName    measureUnitId measureName attrId attributeDescription
  <int> <chr> <int> <chr>                 <int> <chr>        <int> <chr>
1   415 2008     29 Mieszkania odda…          1 -                1 ""
2   415 2009     27 Mieszkania odda…          1 -                1 ""
3   415 2010     40 Mieszkania odda…          1 -                1 ""
4   415 2011     18 Mieszkania odda…          1 -                1 ""
5   415 2012     22 Mieszkania odda…          1 -                1 ""
# i 1 more variable: lastUpdate <chr>
```

Jeżeli chcemy porównać kilka jednostek jednocześnie, możemy w parametrze `unitId` użyć wektora jednostek. Kolumny wartości zostaną oznaczone odpowiednimi kodami jednostek, ale niektóre kolumny zostaną usunięte, by zachować spójność danych.

Przykład. Wybranie danych dla województwa zachodniopomorskiego oraz dolnośląskiego, zmiennych `3643` (radni województwa w wieku 30-39 lat) i `1645343` (liczba kobiet) w latach 1999--2003.

```
get_data_by_unit(unitId = c("023200000000", "030200000000"), varId=c("3643", "1645343"),
  year=c("1999","2000","2001","2002","2003"))
# A tibble: 10 × 11
   id      year  val_023200000000 val_030200000000 measureUnitId attrId_023200000000
   <chr>   <chr>            <dbl>            <dbl>         <int>                <int>
 1 3643    1999                 1                9            26                    1
 2 3643    2000                 2               10            26                    1
 3 3643    2001                 4                6            26                    1
 4 3643    2002                 3                6            26                    1
 5 3643    2003                 3                5            26                    1
 6 1645343 1999               869.            1513.           23                    1
 7 1645343 2000               870.            1511.           23                    1
 8 1645343 2001               871.            1511.           23                    1
 9 1645343 2002               871.            1509.           23                    1
10 1645343 2003               870.            1506.           23                    1
# i 5 more variables: attributeDescription_023200000000 <chr>, attrId_030200000000 <int>,
#   attributeDescription_030200000000 <chr>, lastUpdate_023200000000 <chr>,
#   lastUpdate_030200000000 <chr>
```

Przy pobieraniu danych dla większej liczby jednostek, np. wszystkich województw, powiatów lub gmin znajdujących się w danym makroregionie, użyjemy funkcji `get_data_by_variable()`.

2.6. Pobieranie danych dla pojedynczej zmiennej i wielu jednostek

Dla jednostek terytorialnych korzystamy z `get_data_by_variable()`. Jednostki dobierane są przez podanie jednostki nadrzędnej i poziomu jednostek.

Argumenty dla `get_data_by_variable()`:

- `varId` – pojedynczy kod zmiennej lub wektor zmiennych, np. `c("420","3643")`;
- `unitParentId` – pojedynczy kod jednostki nadrzędnej (opcjonalny);
- `unitLevel` – poziom jednostek terytorialnych od 0 do 6;
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`; opcjonalnie brak podania lat zwróci wszystkie dostępne lata;
- `aggregateId` – poziom agregatu (zasilania danych); użyj funkcji `get_aggregates()` lub wejdź na stronę **https://bdl.stat.gov.pl/bdl/metadane/agregaty**, aby uzyskać dodatkowe informacje.

Przykład. Pobranie danych dotyczących gęstości zaludnienia dla wszystkich regionów (`unitLevel=3`) podrzędnych dla `010000000000` (Makroregion Południowy) w latach 2008-2011.

```
get_data_by_variable(varId = "60559", unitParentId = "010000000000",
  year = 2008:2011, unitLevel = "3")
# A tibble: 8 × 8
  id           name               year    val measureUnitId measureName attrId
  <chr>        <chr>              <chr> <dbl> <chr>         <chr>        <int>
1 011210000000 REGION MAŁOPOLSKIE 2008   216. 26            osoba            1
2 011210000000 REGION MAŁOPOLSKIE 2009   217. 26            osoba            1
3 011210000000 REGION MAŁOPOLSKIE 2010   220. 26            osoba            1
4 011210000000 REGION MAŁOPOLSKIE 2011   220. 26            osoba            1
5 012410000000 REGION ŚLĄSKIE     2008   377. 26            osoba            1
6 012410000000 REGION ŚLĄSKIE     2009   376. 26            osoba            1
7 012410000000 REGION ŚLĄSKIE     2010   376. 26            osoba            1
8 012410000000 REGION ŚLĄSKIE     2011   375. 26            osoba            1
# ℹ 1 more variable: attributeDescription <chr>
```

Argument `unitParentId` jest opcjonalny. Możemy pobrać dane dla wszystkich jednostek na danym poziomie dzięki parametrowi `unitLevel`.

Przykład. Pobranie danych dotyczących gęstości zaludnienia dla wszystkich województw (`unitLevel=2`).

```
get_data_by_variable(varId = "60559", unitLevel = "2")
# A tibble: 336 × 8
```

```
   id           name        year    val measureUnitId measureName attrId
   <chr>        <chr>       <chr> <dbl> <chr>         <chr>        <int>
 1 011200000000 MAŁOPOLSKIE 2002   214. 26            osoba            1
 2 011200000000 MAŁOPOLSKIE 2003   214. 26            osoba            1
 3 011200000000 MAŁOPOLSKIE 2004   215. 26            osoba            1
 4 011200000000 MAŁOPOLSKIE 2005   215  26            osoba            1
 5 011200000000 MAŁOPOLSKIE 2006   216. 26            osoba            1
 6 011200000000 MAŁOPOLSKIE 2007   216  26            osoba            1
 7 011200000000 MAŁOPOLSKIE 2008   216. 26            osoba            1
 8 011200000000 MAŁOPOLSKIE 2009   217. 26            osoba            1
 9 011200000000 MAŁOPOLSKIE 2010   220. 26            osoba            1
10 011200000000 MAŁOPOLSKIE 2011   220. 26            osoba            1
# ℹ 326 more rows
# ℹ 1 more variable: attributeDescription <chr>
```

Jeżeli chcemy pobrać więcej niż jedną zmienną dla wszystkich województw, użyjemy wektora zmiennych w argumencie `varId`.

Przykład. Pobranie danych dotyczących gęstości zaludnienia (zmienna `"60559"`) oraz liczby kobiet w tysiącach (zmienna `"1645343"`) dla wszystkich województw (`unitLevel=2`).

```
get_data_by_variable(varId = c("60559", "1645343"), unitLevel = "2")
# A tibble: 448 × 13
   id           name        year  val_60559 val_1645343 measureUnitId_60559
   <chr>        <chr>       <chr>     <dbl>       <dbl> <chr>
 1 011200000000 MAŁOPOLSKIE 2002       214.       1666. 26
 2 011200000000 MAŁOPOLSKIE 2003       214.       1674. 26
 3 011200000000 MAŁOPOLSKIE 2004       215.       1678. 26
 4 011200000000 MAŁOPOLSKIE 2005       215        1682. 26
 5 011200000000 MAŁOPOLSKIE 2006       216.       1685. 26
 6 011200000000 MAŁOPOLSKIE 2007       216        1690. 26
 7 011200000000 MAŁOPOLSKIE 2008       216.       1694. 26
 8 011200000000 MAŁOPOLSKIE 2009       217.       1700. 26
 9 011200000000 MAŁOPOLSKIE 2010       220.       1717. 26
10 011200000000 MAŁOPOLSKIE 2011       220.       1723. 26
# ℹ 438 more rows
# ℹ 7 more variables: measureName_60559 <chr>, measureUnitId_1645343 <chr>,
#   measureName_1645343 <chr>, attrId_60559 <int>, attributeDescription_60559
<chr>,
#   attrId_1645343 <int>, attributeDescription_1645343 <chr>
```

Dla jednostek terytorialnych – miejscowości statystycznych korzystamy z `get_data_by_variable_locality()`. Jednostki dobierane są przez podanie jednostki nadrzędnej i poziomu jednostek.

Argumenty dla `get_data_by_variable_locality()`:

- `varId` – pojedynczy kod zmiennej lub wektor zmiennych, np. `c("420","3643")`;
- `unitParentId` – pojedynczy kod jednostki nadrzędnej;
- `year` – lata obowiązywania (w formie pojedynczej daty, np. `"2005"`, lub wektora dat `c("2004", "2005", "2006")`; opcjonalnie brak podania lat zwróci wszystkie dostępne lata.

Przykład. Pobranie danych dotyczących liczby bibliotek dla miejscowości statystycznych podrzędnych dla województwa małopolskiego (`011200000000`) w latach 2008-2011.

```
get_data_by_variable_locality(varId = "148190", year = 2008:2011,
  unitParentId = "011200000000")
# A tibble: 3,786 × 8
   id                   name         year    val measureUnitId measureName attrId
   <chr>                <chr>        <chr> <int> <chr>         <chr>        <int>
 1 011212001011-0981682 Bochnia      2009      4 35            ob.              1
 2 011212001011-0981682 Bochnia      2011      4 35            ob.              1
 3 011212001022-0321939 Stanisławice 2009      0 35            ob.              0
 4 011212001022-0321939 Stanisławice 2011      0 35            ob.              0
 5 011212001022-0811796 Baczków      2009      0 35            ob.              0
 6 011212001022-0811796 Baczków      2011      0 35            ob.              0
 7 011212001022-0811827 Bessów       2009      0 35            ob.              0
 8 011212001022-0811827 Bessów       2011      0 35            ob.              0
 9 011212001022-0811833 Bogucice     2009      0 35            ob.              0
10 011212001022-0811833 Bogucice     2011      0 35            ob.              0
# ℹ 3,776 more rows
# ℹ 1 more variable: attributeDescription <chr>
```

2.7. Dodatkowe narzędzia

Wraz z funkcjami pobierania danych w pakiecie znajdują się dodatkowe funkcje pozwalające na szybkie podsumowanie danych, wizualizację na wykresach oraz tworzenie map.

Podsumowanie danych. Aby dostać krótkie podsumowanie danych (średnią, odchylenie standardowe, minimalne i maksymalne wartości, wariancję), należy pobrać dane, zapisać do zmiennej i następnie wywołać je jako argument metody `summary()`.

Przykład

```
df <- get_data_by_variable(varId = "148190", unitParentId = "011200000000", unitLevel = 6)
summary(df)
# A tibble: 22 × 10
# Groups:   id [22]
   id           name         measur…¹ measure…² attrib…³  mean    std   min   max variance
   <chr>        <chr>        <chr>    <chr>     <chr>    <dbl>  <dbl> <dbl> <dbl>    <dbl>
 1 011212001000 Powiat boch… 23       tys. osób ""        51.8  1.86   48.8  54.1   3.47
 2 011212006000 Powiat krak… 23       tys. osób ""       131.  10.6   117.  153.  113.
 3 011212008000 Powiat miec… 23       tys. osób ""        25.8  0.942  24.0  27.6   0.887
```

```
 4 011212009000 Powiat myśl… 23       tys. osób ""        60.3  3.41   54.6  65.4  11.6
 5 011212014000 Powiat pros… 23       tys. osób ""        22.1  0.238  21.4  22.4   0.0566
 6 011212019000 Powiat wiel… 23       tys. osób ""        58.1  6.68   50.1  72.3  44.6
 7 011212161000 Powiat m. K… 23       tys. osób ""       406.   9.28  393.  428.   86.1
 8 011212205000 Powiat gorl… 23       tys. osób ""        54.5  0.558  53.4  55.2   0.311
 9 011212207000 Powiat lima… 23       tys. osób ""        62.1  2.99   57.0  65.8   8.96
10 011212210000 Powiat nowo… 23       tys. osób ""       102.   5.47   92.6 109.   30.0
# i 12 more rows
# i abbreviated names: ¹measureUnitId, ²measureName, ³attributeDescription
```

2.8. Pozostałe funkcje

W pakiecie znajduje się wiele innych funkcji pozwalających np. na wyświetlenie informacji o zmiennej `variable_info()` lub wyświetlenie informacji o jednostce `unit_info()`. Listę wszystkich funkcji wraz z opisem parametrów można znaleźć w pomocy R oraz na stronie: **https://statisticspoland.github.io/R_Package_to_API_BDL/reference/index.html**.

Literatura

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Szpadel, M. i Kania, K. (2023). *bdl: Interface and Tools for 'BDL' API*. R package version 1.0.5. **https://CRAN.R-project.org/package=bdl**

DOI: 10.15611/2024.45.1.03

3 Graficzna prezentacja danych[1]

3.1. Wprowadzenie

Do podstawowych zalet stosowania wykresów w publikacjach statystycznych zalicza się to, że (Pieniążek i in., 2014, s. 7):

- prezentują zjawisko bardziej przejrzyście niż wyłącznie liczby,
- pozwalają zorientować się w ogólnej charakterystyce zjawiska,
- przyciągają uwagę użytkownika, jeśli dane są starannie i w atrakcyjny sposób zaprezentowane,
- są efektywne, ponieważ percepcja wzrokowa jest szybsza niż odczytywanie wielu liczb z tabel,
- dają możliwość przekazania w prosty sposób złożonych informacji,
- są bardzo przydatne w porównaniach i pomocne w interpretacji zjawisk,
- pomagają w zapamiętaniu informacji o zjawiskach,
- stanowią narzędzie uniwersalne ze względu na język stosowany w różnych dziedzinach.

Z opracowywaniem wykresów związane są pewne zasady, do których należą następujące (Pieniążek i in., 2014, s. 21-22).

a. Zasady ogólne:

   - wykresy powinny charakteryzować się prostą budową, atrakcyjną formą i zwięzłym oraz czytelnym opisem,
   - wykresy powinny współgrać z pozostałą treścią publikacji, tekstem, tabelami i mapami, uzupełniać je, a nie powielać,
   - wykresy powinny przedstawiać informacje jasno i precyzyjnie, nie wprowadzając w błąd użytkownika danych.

b. Zasady dotyczące treści:

   - osie powinny zawierać opis jednostki prezentacji,
   - tytuł wykresu powinien w sposób jednoznaczny definiować przedmiot wykresu,
   - w przypadku wykresów złożonych tytuł może mieć charakter ogólny, a opis poszczególnych osi powinien uzupełniać właściwą treść,
   - naniesienie treści wykresu bezpośrednio na obszar wykresu (np. na wykresie liniowym) pozwala na minimalizację zawartości legendy.

[1] W rozdziale o prezentacji graficznej wykorzystano opracowania Biecka (2016, 2017) oraz Kopczewskiej i in. (2009).

c. Zasady dotyczące formy:

- należy odpowiednio dobrać formę wykresu do przedstawianego szeregu statystycznego (np. wykresy liniowe stosowane są do prezentacji przebiegu zjawiska w czasie),
- nie zaleca się stosowania wykresów trójwymiarowych (poza szczególnymi przypadkami),
- liczba elementów przy prezentowaniu struktury nie może być zbyt wielka; liczebność struktury powinna być ograniczona czytelnością przekazu; zaleca się wyodrębnienie 5-7 elementów struktury,
- obowiązuje zasada proporcjonalnego podziału osi czasu na wykresach dynamicznych, aby przebieg linii dynamiki odpowiadał rzeczywistym zmianom wartości,
- ze względu na czytelność wykresu nie zaleca się stosowania na jednym wykresie więcej niż 3 szeregów słupków dla wykresów dynamicznych,
- nie zaleca się używania zbyt wielu kolorów.

3.2. Podstawy wizualizacji danych – formatowanie wykresów

Wiele funkcji graficznych, oprócz własnych parametrów, może przekazywać wspólne dla całego środowiska parametry funkcji `par`. W tabeli 3.1 zestawiono najważniejsze parametry funkcji `par` oraz innych funkcji graficznych (np. `plot`).

Tabela 3.1. Najważniejsze parametry funkcji `par` oraz `plot` trybu graficznego środowiska R

Parametr	Znaczenie	Uwagi / wartości
`adj`	wyrównywanie tekstu	`0` – do lewej, `0.5` – centrowanie (domyślnie), `1` – do prawej
`ask`	`ask=TRUE` oznacza, że przed wykonaniem polecenia R poprosi o naciśnięcie dowolnego klawisza (domyślnie ustalony na `FALSE`)	
`bg`	kolor tła wykresu (domyślnie kolor przeźroczysty: `bg="transparent"`)	np. `bg="magenta"`
`cex`	wielkość przeskalowania tekstu i symboli na rysunku w stosunku do wielkości domyślnej `cex=1`	np. `cex=0.5` oznacza zmniejszenie o 50%, a `cex=2` – dwukrotne zwiększenie przykład zastosowania tego parametru zawiera rys. 3.1

`cex.axis`
`cex.lab`
`cex.main`
`cex.sub`
	dotyczy odpowiednio wyświetlenia wielkości znaczników na osi, etykiet osi, głównego tytułu (domyślnie `cex.main=1.2`), podtytułu	np. `cex.axis=1.4`

`col`	kolor punktów/figur na wykresie (domyślnie `"black"`)	

`col.axis`
`col.lab`
`col.main`
`col.sub`
	kolor odpowiednio znaczników na osi, etykiet osi, głównego tytułu, podtytułu	np. `col.axis="green"`

`crt`	kąt nachylenia wartości numerycznych	np. `crt=60`
`family`	rodzina (nazwa) używanej czcionki (domyślnie `"sans"`)	Do standardowych typów czcionek należą: `"serif"`, `"sans"`, `"mono"` i `"symbol"`
`fg`	kolor takich obiektów, jak osie i otoczenia rysunku (domyślnie `"black"`)	np. `fg="pink"`

`fig`	określenie obszaru na wykresie, w którym ma pojawić się diagram	używany zazwyczaj do łączenia kilku wykresów. Może być określony tylko poprzez funkcję `par` i w takim przypadku połączony najczęściej jest z parametrem `new=TRUE`
`font`	krój czcionki	`1` – krój normalny (domyślnie), `2` – pogrubienie, `3` – kursywa, `4` – kursywa i pogrubienie

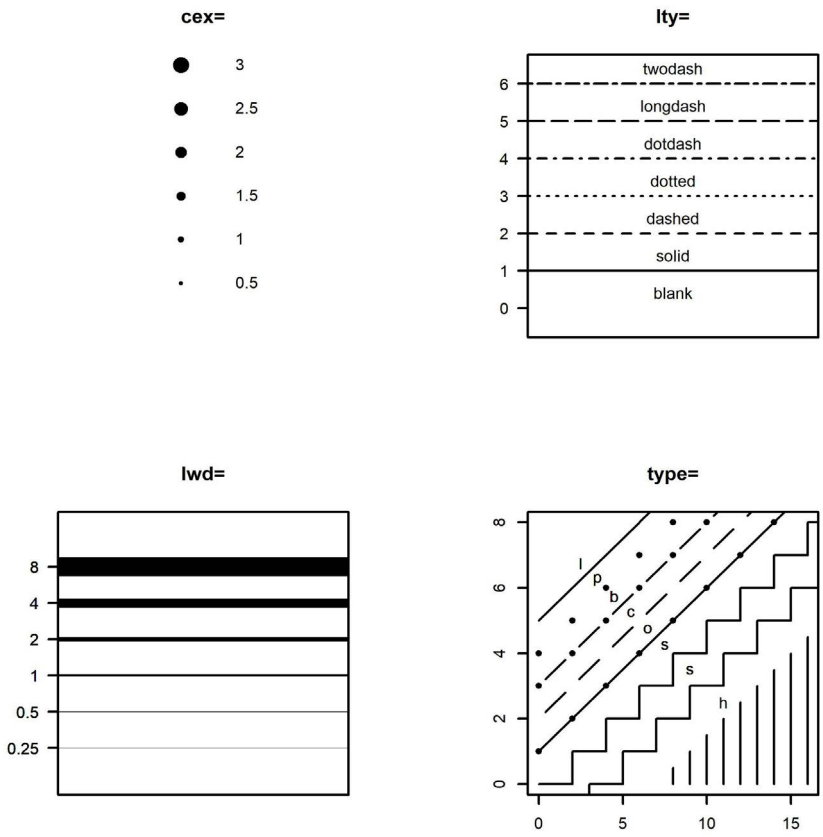
`font.axis`
`font.lab`
`font.main`
`font.sub`
	krój czcionki użytej do wyświetlenia odpowiednio znaczników na osi, etykiet osi, głównego tytułu, podtytułu	jw.

`lab`	liczba znaczników na osiach *x* oraz *y* i długość etykiet	domyślnie: `lab=c(5,5,7)`
`las`	położenie etykiet opisujących osie w stosunku do samych osi	`0` – (domyślne) – zawsze równolegle w stosunku do osi , `1` – zawsze poziome, `2` – zawsze prostopadle w stosunku do osi, `3` – zawsze pionowe
`lty`	rodzaj linii	`0` – bez linii, `1` – ciągła (domyślnie), `2` – kreskowana, `3` – kropkowana, `4` – kropka-kreska, `5` – długa kreska, `6` – podwójna kreska (przykład zastosowania tego parametru zawiera rys. 3.1)
`lwd`	grubość linii (`lwd=1` oznacza grubość domyślną)	wartości powyżej 1 (np. `lwd=1.5`) oznaczają zwiększenie grubości linii, a poniżej 1 (np. `lwd=0.5`) jej zmniejszenie (przykład zastosowania tego parametru zawiera rys. 3.1)
`main*`	tytuł wykresu	`main=""` oznacza, że nie będzie wyświetlany tytuł wykresu
`mfcol`, `mfrow`, `mfg`	parametr służący do tworzenia wykresów wieloczęściowych, gdzie `mfcol` (`mfrow`) tworzy tablicę obiektów graficznych, dzieląc ekran na odpowiednią liczbę kolumn i wierszy; kolejne podwykresy są wstawiane w kolejności według kolumn (wierszy), chyba że wskazane zostanie dokładne miejsce wstawienia podwykresu parametrem `mfg`	parametry `mfcol` i `mfrow` można ustalać tylko poprzez polecenie `par` np. `par(mfrow(2,2))`
`new`	`new=TRUE` – nowy wykres nie usuwa poprzedniej zawartości okna graficznego	
`pch`	oznaczenia symboli na rysunku (domyślnie `pch=1`)	prezentacja graficzna symboli znajduje się na rys. 3.2
`pty`	typ pola rysunku	`"s"` – kwadratowe, `"m"` – zmaksymalizowane (domyślnie)
`srt`	kąt nachylenia tekstu na osi *x*	np. `srt=60`
`sub*`	podtytuł umieszczony pod wykresem	`sub=""` oznacza, że nie będzie wyświetlany podtytuł wykresu
`type*`	typ tworzonego wykresu	`"p"` (wartość domyślna) – punkty, `"l"` – linie, `"b"` – linie i punkty, `"s"` – wykres schodkowy, `"h"` – linie pionowe, `"n"` – wykres pusty
`xlab*`	etykieta osi *x*	np. `xlab="zatrudnienie"`
`Xlog`	`xlog=TRUE` oznacza, że oś odciętych będzie osią logarytmiczną	
`xlim*`	zakres osi odciętych	np. `xlim=c(-1,1)`, `xlim=range(x)`
`ylab*`	etykieta osi *y*	np. `ylab="wydajność pracy"`
`ylog`	`ylog=TRUE` oznacza, że oś rzędnych będzie osią logarytmiczną	
`ylim*`	zakres osi rzędnych	np. `ylim=c(0,25)`, `ylim=range(y)`

* Nie dotyczy funkcji `par`.

Źródło: opracowanie własne na podstawie dokumentacji R oraz pracy (Walesiak i Gatnar, 2009).

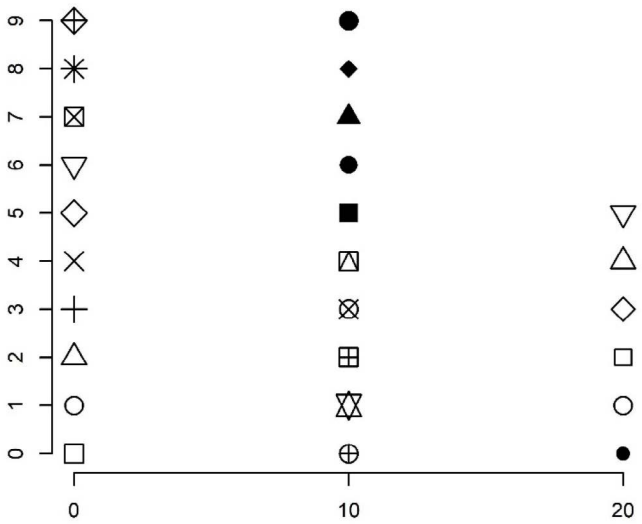
Graficzną prezentację parametrów `cex`, `lty`, `lwd` oraz `type` ukazano na rys. 3.1.



Rys. 3.1. Graficzna prezentacja parametrów `cex`, `lty`, `lwd` oraz `type`

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Kształt punktu określa się z wykorzystaniem argumentu `pch`. Wartości od 0 do 25 oznaczają znaki graficzne (zob. rys. 3.2). Wartości od 32 do 127 oznaczają znaki w kodzie ASCII.



Rys. 3.2. Graficzne oznaczenia przyjmowane przez parametr `pch`

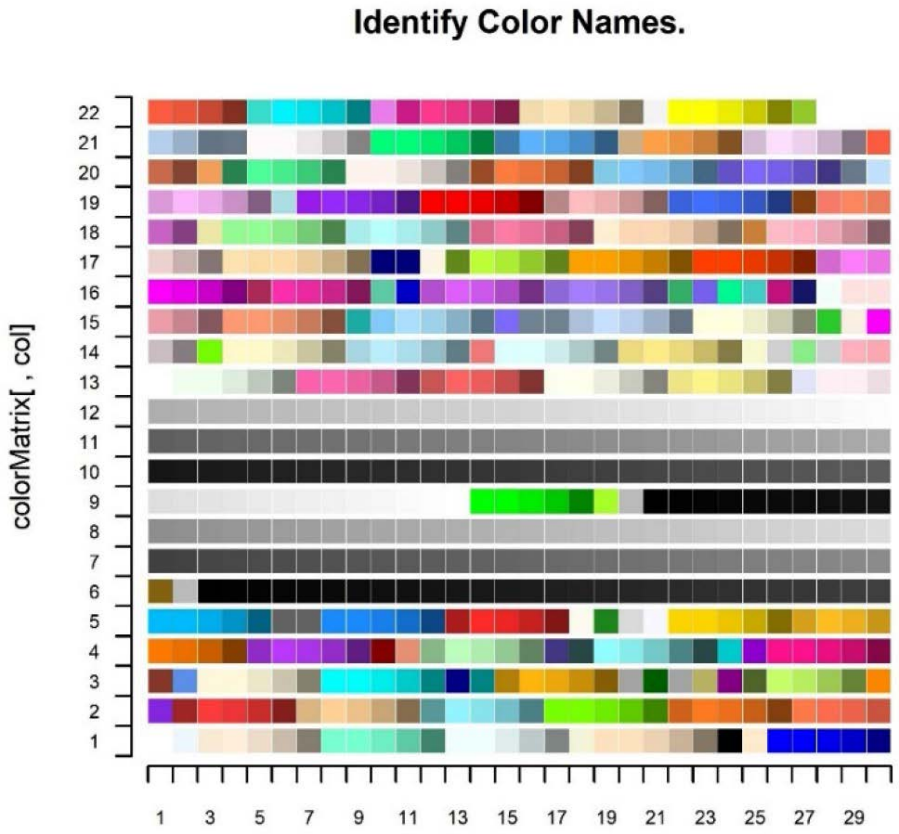
Źródło: opracowanie własne na podstawie (Crawley, 2007, s. 139-140).

3.2.1. Dobór kolorów

W programie R dostępnych jest 657 kolorów. Następujący skrypt pozwala wyświetlić wszystkie kolory:

```
library(fBasics)
colorLocator(locator=TRUE)
```

W macierzy kolorów należy wybrać interaktywnie żądane kolory, a następnie po naciśnięciu *Stop* w oknie pojawi się lista kolorów (rys. 3.3).

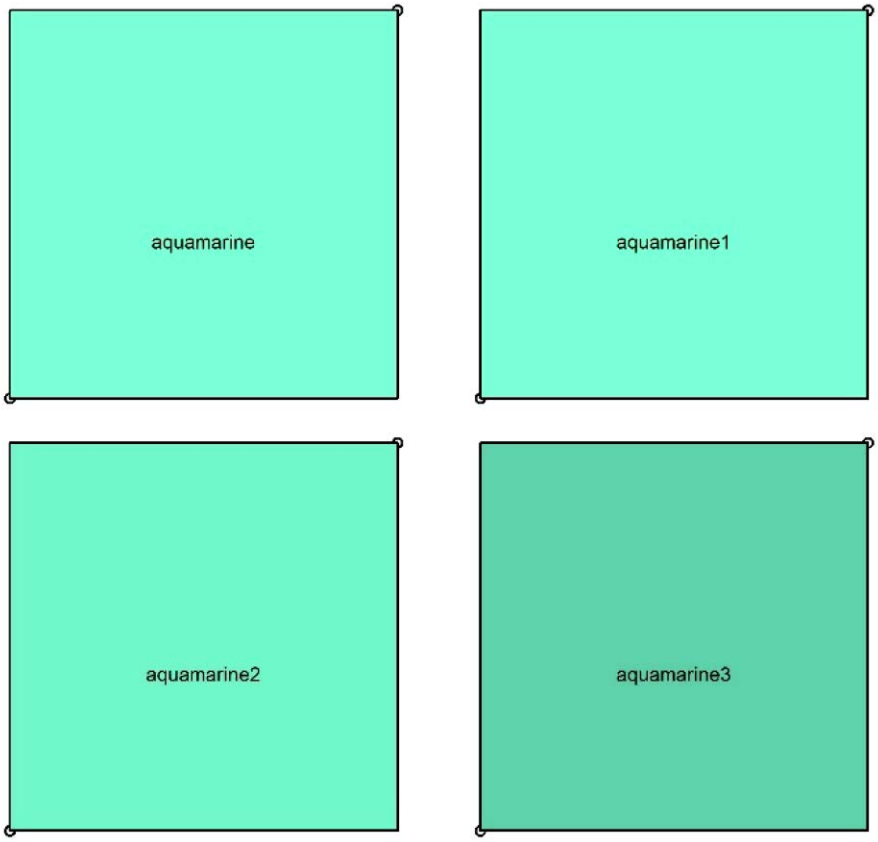


Rys. 3.3. Lista kolorów w programie R

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Wybrane kolory można wyświetlić na ekranie (rys. 3.4) za pomocą skryptu:

```
cl<-colors()
cl.m<-as.matrix(cl)
par(mfrow=c(2,2),mar=c(1,1,1,1))
for(i in 8:11){
  plot(c(1,1.5),c(1,1.5),xlab="",ylab="",bty="n",axes=FALSE)
  rect(1,1,1.5,1.5,col=cl.m[i],border="black")
  text(1.25,1.2,cl.m[i],cex=0.9)
}
```



Rys. 3.4. Wybrane kolory oraz ich nazwy

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Z kolei nazwy kolorów wyświetla się z wykorzystaniem poleceń:

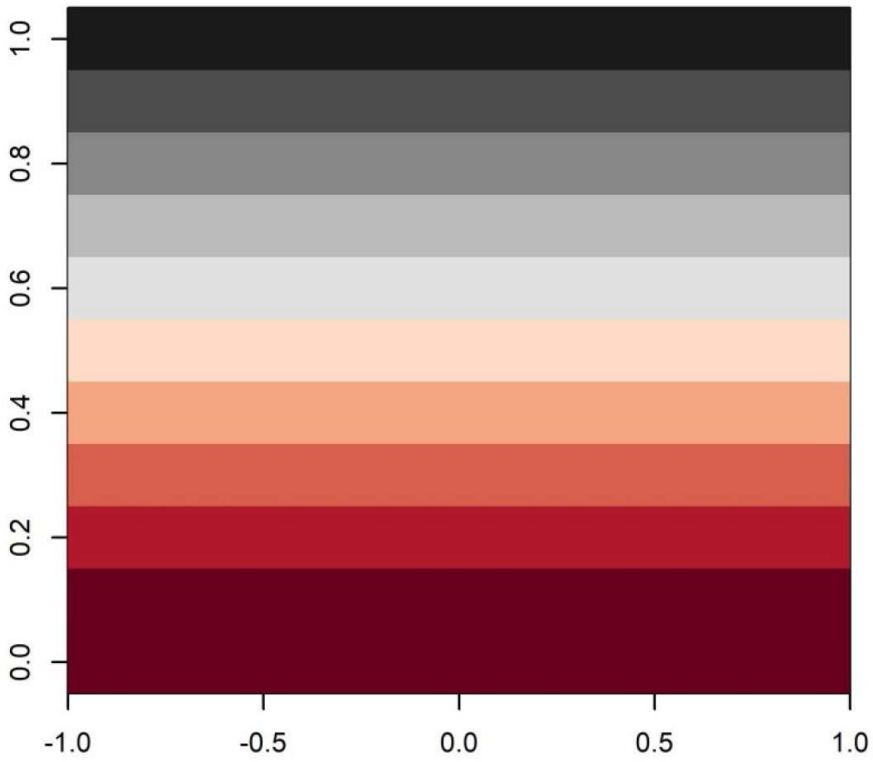
```
kolory<-colors()
kolory[1:19]
[1] "white"          "aliceblue"     "antiquewhite"  "antiquewhite1"
[5] "antiquewhite2" "antiquewhite3" "antiquewhite4" "aquamarine"
[9] "aquamarine1"   "aquamarine2"   "aquamarine3"   "aquamarine4"
[13] "azure"         "azure1"        "azure2"        "azure3"
[17] "azure4"        "beige"         "bisque"
```

W zastosowaniach graficznych programu R korzysta się najczęściej z wbudowanych palet kolorów:

- `rainbow()`, `heat.colors()`, `terrain.colors()`, `topo.colors()`, `cm.colors()` – podając w nawiasie liczbę wybranych kolorów,
- dodatkowe palety kolorystyczne można uzyskać z pakietu `RColorBrewer` (palety kolorów: `RdGy, Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd`).

Na przykład 10 kolorów z palety `"RdGy"` (rys. 3.5) otrzymuje się za pomocą skryptu:

```
library(RColorBrewer)
paleta1<-brewer.pal(10,"RdGy")
image(matrix(1:11,1),col=paleta1)
```



Rys. 3.5. Wybrane kolory z palety `"RdGy"` pakietu `RColorBrewer`

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

3.2.2. Tytuł, podtytuł, osie i opisy osi wykresu

Standardowe elementy wykresu (tytuł, podtytuł, opisy osi wykresu, legenda) modyfikuje się argumentami:

`main (main="")` oznacza, że nie będzie wyświetlany tytuł wykresu, określa tytuł wykresu (standardowo rysowany ponad wykresem pogrubioną czcionką).
`sub (sub="")` oznacza, że nie będzie wyświetlany podtytuł wykresu, określa podtytuł wykresu (rysowany poniżej wykresu).
`xlab` określa etykietę osi *X* wykresu (oś odciętych) – np. `xlab="Zatrudnienie"`.
`ylab` określa etykietę osi *Y* wykresu (oś rzędnych) – np. `ylab="Wydajność pracy"`.

Osie wykresów rysowane są zazwyczaj automatycznie w ramach rysunku głównego. Jeśli wymagane są specyficzne etykiety w funkcji `plot()`, zawieszamy rysowanie osi opcją `axes=FALSE`. Osie wprowadzamy osobno komendą `axis()`

`axis(side, at, labels, tick=TRUE, …)`
`side` (`1` – oś dolna, `2` – oś lewa, `3` – oś górna, `4` – oś prawa),
`at` określa, w których punktach osi mają być nałożone etykiety lub znaczniki,
`labels` pozwala zdefiniować wektor etykiet – np. `labels=c("2005","2010","2015")`,
`tick` określa, czy znaczniki mają być drukowane na wykresie.

3.3. Formaty plików graficznych w R

Program R umożliwia zapisanie grafiki w następujących formatach:

- rastrowych: png, bmp, tiff, jpeg,
- wektorowych: pdf, ps, svg.

Dla formatów rastrowych domyślne wymiary podawane są w pikselach (np. 640 × 480), a dla formatów wektorowych w calach.

Plik rastrowy (inaczej bitmapowy) jest obrazem lub grafiką składającymi się z prostokątnej siatki regularnie położonych pikseli o tej samej wielkości. Pliki te mogą być zmniejszane, jednak ich powiększanie spowoduje niewystarczającą ilość detali i postrzępienie obrazu. Pliki tego formatu są dużych rozmiarów, a ponadto nie można dokonać konwersji tego formatu do wektorów. Grafikę tego formatu charakteryzują bogactwo i głębia barw, kontrastu, jasności i nasycenia.

Plik wektorowy jest obrazem wewnętrznym zapisanym w postaci punktów łączących, zdefiniowanych za pomocą figur geometrycznych. Punkty, o których mowa, umiejscowione są w matematycznym układzie współrzędnych, dzięki temu grafiki zapisane w sposób wektorowy są w pełni skalowalne. Jest wykorzystywany do projektowania ilustracji, wykresów z danymi, ikon lub logotypów. Obrazy wymagają małej ilości informacji, co powoduje, że pliki tego formatu mają niewielkie rozmiary. Istnieje możliwość konwersji do grafiki rastrowej. Pliki tego formatu nie nadają się do pokazywania dużej liczby szczegółów.

Schemat zapisania w pliku wykresu w formacie np. `jpeg` o rozdzielczości 640 × 480 px jest następujący:

- otwieramy plik do zapisu grafiki:

  ```
  jpeg("rys.jpeg",width=640,height=480)
  ```

- wpisujemy do pliku przykładowy wykres przy użyciu np. funkcji `plot()`

  ```
  plot()
  ```

- do zakończenia zapisywania wykresu do pliku należy użyć funkcji:

  ```
  dev.off()
  ```

3.4. Wykresy podstawowe z wykorzystaniem programu R dla danych z BDL

Do wykresów podstawowych zalicza się: wykres słupkowy/kolumnowy, wykres kołowy 2D i 3D, wykres liniowy, wykres warstwowy, wykres radarowy, wykres pudełkowy (*boxplot*).

3.4.1. Wykres słupkowy/kolumnowy

Do cech charakterystycznych wykresów słupkowych/kolumnowych zalicza się to, że:

- dane przedstawiamy w postaci słupków,
- słupki można zaprezentować poziomo lub pionowo,

- słupki porównujemy pod względem ich długości,
- słupki mogą reprezentować liczebności, strukturę,
- do prezentacji wykorzystuje się dane w postaci z szeregów szczegółowych, szeregów rozdzielczych (punktowych, przedziałowych) oraz tablicy kontyngencji.

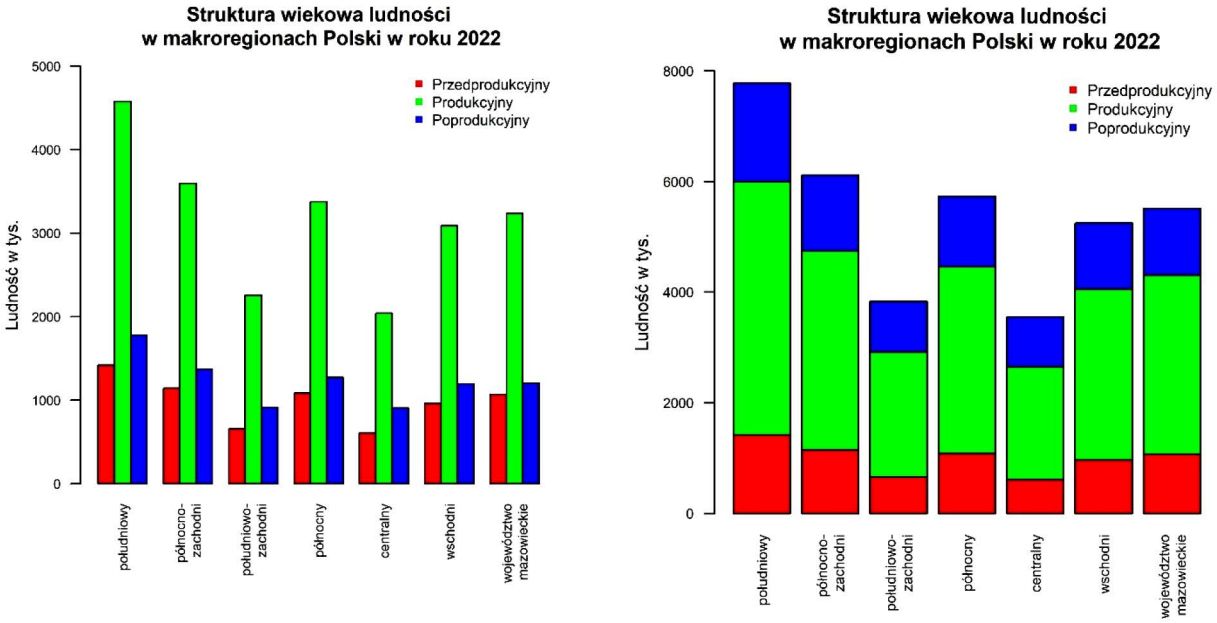
Rozróżnia się następujące typy wykresów słupkowych (kolumnowych) dla jednej lub kilku serii danych:

a) wykres słupkowy grupowany – w `barplot()` `beside=TRUE`;
b) wykres słupkowy skumulowany – w `barplot()` `beside=FALSE`;
c) wykres słupkowy skumulowany 100% – w `barplot()` `beside=FALSE`. Do utworzenia tego wykresu należy przedstawić dane jako udziały procentowe sumujące się do 100.

Wykres słupkowy grupowany (rys. 3.6 – lewy panel) dla zmiennej „Struktura wiekowa ludności w Polsce w roku 2022" (wiek przedprodukcyjny `"399979"`, produkcyjny `"399980"` i poprodukcyjny `"399980"`) realizuje skrypt 3.1a.

Skrypt 3.1a

```
library(bdl)
library(dplyr)
options(OutDec=",")
# Wczytanie informacji o zmiennej z BDL
no_uL=1 # (1 - makroregion)
d1<-get_data_by_variable("399979",unitLevel=no_uL,year=2022) %>%
  dplyr::select(id,name, x1=val)
d2<-get_data_by_variable("399980",unitLevel=no_uL,year=2022) %>%
  dplyr::select(id,name, x2=val)
d3<-get_data_by_variable("399981",unitLevel=no_uL,year=2022) %>%
  dplyr::select(id,name, x3=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
dane <- d1 %>%
  full_join (d2, by=c("id","name")) %>%
  full_join (d3, by=c("id","name"))
x<-as.data.frame(dane)
# Usunięcie z nazwy obiektu wyrazu Makroregion
x<-mutate(x,name=gsub("MAKROREGION ","",name))
x<-as.data.frame(x)
# Nadanie nazw wierszom i kolumnom
nazwy<-x[,2]
rownames(x)<-tolower(nazwy)
x<-x[,3:5]
colnames(x)<-c("Przedprodukcyjny","Produkcyjny","Poprodukcyjny")
rownames(x)<-c("południowy","północno-\n zachodni","południowo-\n zachodni",
  "północny","centralny","wschodni","województwo\n mazowieckie")
barplot(as.matrix(t(x/1000)),beside=TRUE,horiz=FALSE,
  col=c("red","green","blue"), ylim=c(0,5000),ylab="Ludność w tys.",
  main="Struktura wiekowa ludności\n w makroregionach Polski
  w roku 2022",cex.names=0.8,cex.axis=0.8,las=2)
legend("topright", legend=colnames(x), ncol=1, cex=0.9,bg="white",
  pch=c(15,15,15),col=c("red","green","blue"),bty="n")
```



Rys. 3.6. Wykres słupkowy grupowany (lewy panel) oraz wykres słupkowy skumulowany (prawy panel) dla zmiennej „Struktura wiekowa ludności w Polsce w roku 2022"

Źródło: opracowanie własne.

Wykres słupkowy skumulowany (rys. 3.6 – prawy panel) dla zmiennej „Struktura wiekowa ludności w Polsce w roku 2022" (wiek przedprodukcyjny, produkcyjny i poprodukcyjny) realizuje skrypt 3.1b (w skrypcie 3.1a w `barplot()` argument `beside=FALSE` oraz `ylim=c(0,8000)`).

3.4.2. Wykres kołowy 2D i 3D

Charakterystyczne dla wykresów kołowych jest to, że:

- ilustrują zjawisko w ujęciu strukturalnym,
- są atrakcyjne wizualnie,
- są traktowane jako mniej czytelne niż np. wykresy słupkowe (oko ludzkie z większą trudnością porównuje pola i kąty),
- na tego typu wykresach można przedstawić jedynie jedną serię danych.

W programie R wykres kołowy 2D tworzymy poleceniem `pie()`, a wykres kołowy 3D poleceniem `pie3D()` z pakietu `plotrix`.

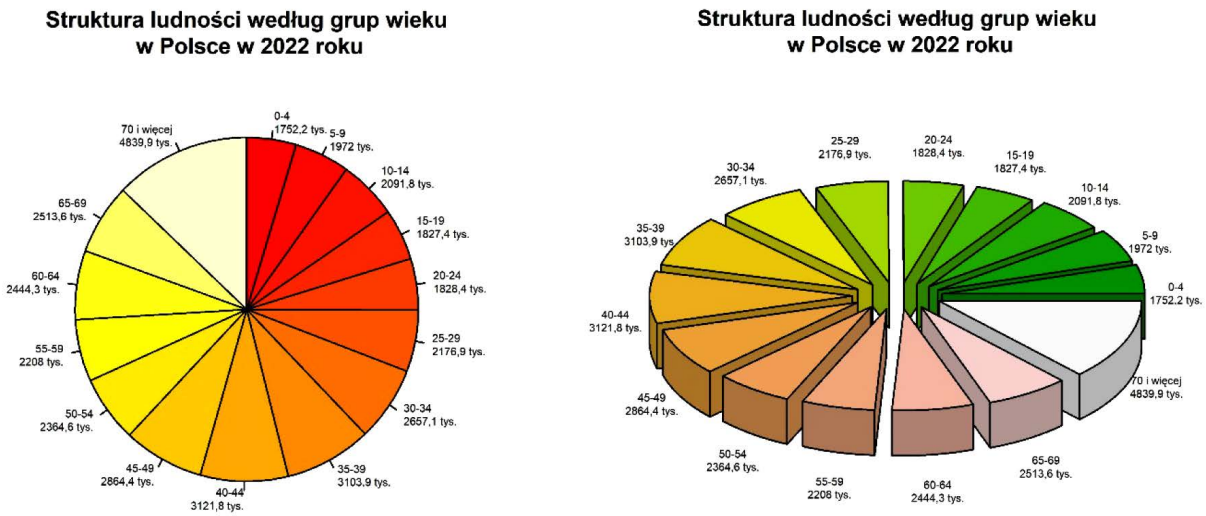
Wykres kołowy 2D (rys. 3.7 – lewy panel) dla zmiennej „Struktura ludności według grup wieku w Polsce w 2022 roku" realizuje skrypt 3.2a.

Skrypt 3.2a

```
library(bdl)
options(OutDec=",")
# Wczytanie informacji o zmiennej z BDL
no_uL=0 # (0 - Polska)
```

```
dane<-get_data_by_variable(c("72306","72307","72308","72309","47734","47694",
  "47722","47701","47707","47726","47717","47732","47739","72239","72240"),
  unitLevel=no_uL,year=2022)
dane<-t(dane[,4:18])
x<-as.data.frame(round(dane/1000,1))
# Nadanie nazw wierszom i kolumnom
rownames(x)<-c("0-4","5-9","10-14","15-19","20-24","25-29","30-34","35-39",
  "40-44","45-49","50-54","55-59","60-64","65-69","70 i więcej")
colnames(x)<-c("Ludność")
# wykres kołowy 2D
attach(x)
pie(Ludność,col=heat.colors(nrow(x)),labels=paste(rownames(x),paste(Ludność,
  "tys.",sep=""),sep="\n"),clockwise=TRUE,
  main="Struktura ludności według grup wieku\n w Polsce w 2022 roku",cex=0.6)
detach(x)
```

Wykres kołowy 3D (rys. 3.7 – prawy panel) dla zmiennej „Struktura ludności według grup wieku w Polsce w 2022 roku" realizuje skrypt 3.2b. Na wykresie kołowym 3D, aby zmienić wykres na rozsunięty, wystarczy dla parametru `explode` przypisać wartość większą niż 0, z kolei parametr `theta` określa kąt, pod którym widzimy dany wykres.



Rys. 3.7. Wykres kołowy 2D (lewy panel) oraz 3D (prawy panel) dla zmiennej „Struktura ludności według grup wieku w Polsce w 2022 roku"

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 3.2b

```
… (polecenia ze skryptu 3.2a do wykresu kołowego 2D)
# wykres kołowy 3D
attach(x)
pie3D(Ludność,radius=0.9,explode=0.2,theta=pi/4,labels=paste(rownames(x),
  paste(Ludność," tys.",sep=""),sep="\n"),
col=terrain.colors(nrow(x)),labelcex=0.6, edges=1000,
  main="Struktura ludności według grup wieku\n w Polsce w 2022 roku")
detach(x)
```

3.4.3. Wykresy liniowy i warstwowy

Do cech charakterystycznych wykresu liniowego należy to, że:

- jest jednym z najbardziej popularnych wykresów statystycznych,
- dane prezentowane są za pomocą linii, najczęściej łamanej,
- ta forma wykresu wykorzystywana jest najczęściej do przedstawienia danych w postaci szeregów czasowych,
- najczęściej na osi *x* występuje czas (miesiące, lata), a na osi *y* wybrana przez nas zmienna lub wybrane zmienne,
- wykresy liniowe pozwalają na szybkie rozeznanie się w tendencji dla zmiennej lub zmiennych,
- wykres liniowy jest najczęściej stosowany w analizach zjawisk giełdowych.

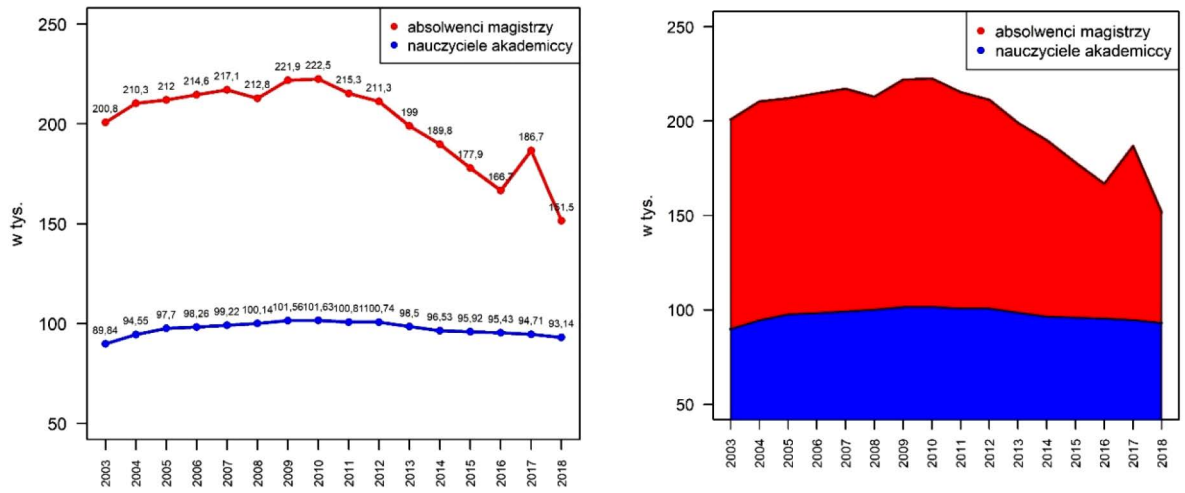
Wykres warstwowy poprzedzony jest sporządzeniem wykresu liniowego (lub kilku liniowych). Do wypełniania kolorem służy funkcja `polygon`. W funkcji `polygon` podajemy wektor współrzędnych *x* i współrzędnych *y* punktów, których wnętrze po połączeniu będzie wypełnione kolorem. Dlatego do każdej serii danych dodajemy dwa punkty o współrzędnych 0 w *y* oraz początek i koniec serii w *x*. W ten sposób zamalujemy obszar pod daną serią. Ze względu na to, że funkcja `polygon` dodaje zamalowane obszary do istniejącego już wykresu, należy zwrócić uwagę na kolejność zamalowywania obszarów pod seriami, aby nie ukryć serii o niskich wartościach pod spodem.

Wykresy liniowe (rys. 3.8 – lewy panel) dla zmiennych „Absolwenci magistrzy" oraz „Nauczyciele akademiccy" realizuje skrypt 3.3a.

Skrypt 3.3a

```
library(bdl)
library(dplyr)
options(OutDec=",")
# stałe
rok<-c(2003:2018)
no_uL=0 # (0 - Polska)
# Wczytanie informacji o zmiennych z BDL
d1<-get_data_by_variable("8273",unitLevel=no_uL,year=rok) %>%
  dplyr::select(year, x1=val)
d2<-get_data_by_variable("473986",unitLevel=no_uL,year=rok) %>%
  dplyr::select(year, x2=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
dane <- d1 %>% full_join (d2, by=c("year"))
x<-as.data.frame(dane)
# Nadanie nazw kolumnom
colnames(x)<-c("rok","v1","v2")
# Przeliczenia
x<-mutate(x,v1=round(v1/1000,2))
x<-mutate(x,v2=round(v2/1000,1))
# Wykres liniowy
attach(x)
plot(rok,v1, type="n",xlab="",ylim=c(50,250),ylab="w tys.",xaxt="n",las=2)
axis(1,at=x[,1],labels=x[,1],las=2,cex.axis=0.7)
lines(rok,v1,col="blue",lwd=2)
text(rok,v1,pos=3,labels=x[,2],cex=0.6)
```

```
points(rok,v1, pch=16, col="blue", cex=1)
lines(rok,v2, col="red",lwd=2)
points(rok,v2, pch=16, col="red", cex=1)
text(rok,v2,pos=3,labels=x[,3],cex=0.6)
legend("topright", c("absolwenci magistrzy","nauczyciele akademiccy"),
  pch=c(16,16), col=c("red","blue"),cex=0.9)
```



Rys. 3.8. Wykresy liniowe (lewy panel) oraz wykresy powierzchniowe (prawy panel) dla zmiennych „Absolwenci magistrzy" oraz „Nauczyciele akademiccy"

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Wykresy powierzchniowe (rys. 3.8 – prawy panel) dla zmiennych „Absolwenci magistrzy" oraz „Nauczyciele akademiccy" realizuje skrypt 3.3b.

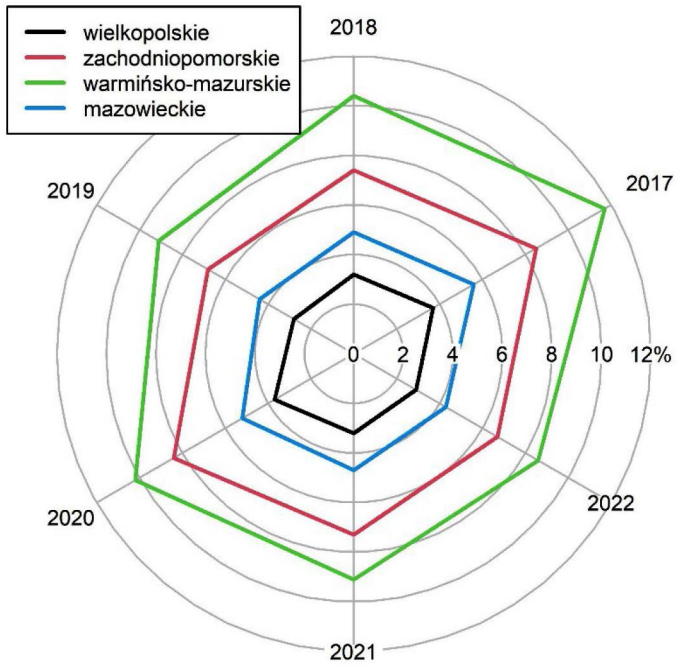
Skrypt 3.3b

```
… polecenia ze skryptu 3.3a do wykresu liniowego
# Wykres liniowy
attach(x)
plot(rok,v1, type="n",xlab="",ylim=c(50,250),ylab="w tys.",xaxt="n",las=2)
axis(1,at=x[,1],labels=x[,1],las=2,cex.axis=0.8)
lines(rok,v1,col="blue",lwd=2)
lines(rok,v2, col="red",lwd=2)
legend("topright", c("absolwenci magistrzy","nauczyciele akademiccy"),
  pch=c(16,16), col=c("red","blue"))
# Warstwy kolorów
x <- c(2003:2018, 2018, 2003)  # wektor współrzędnych x
polygon(x, c(v2,0,0), col="red")
polygon(x, c(v1,0,0), col="blue")
```

3.4.4. Wykres radarowy

Na wykresie radarowym dane przestawione są na bazie koła, w którym wartości liczbowe zaznaczone są na liniach odchodzących promieniście od centralnego punktu, formując w ten sposób pewne ramy. Wykres przypomina pajęczynę lub obraz na ekranie radaru. Wykres radarowy (*radar chart*,

spiker chart lub *web chart*) jest nadużywanym typem wykresu. Wykres ten nie jest efektywny, gdy powstałe ramy się przecinają oraz jest zbyt dużo obiektów na wykresie. Przeznaczony jest do prezentacji małych zbiorów danych (niewielka liczba wierszy i kolumn w macierzy danych). Zmienne opisujące obiekty (wiersze) muszą mieć zbliżony rząd wielkości.



Rys. 3.9. Wykres radarowy dla stopy bezrobocia w 4 wybranych województwach Polski w latach 2017-2022

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Wykres radarowy (rys. 3.9) dla stopy bezrobocia w 4 wybranych województwach Polski w latach 2017-2022 realizuje skrypt 3.4.

Skrypt 3.4

```
library(bdl)
library(dplyr)
library(plotrix)
options(OutDec=",")
# stałe
no_uL=2 # (2 - województwo)
# Wczytanie informacji o zmiennych z BDL
d1<-get_data_by_variable("60270",unitLevel=no_uL,year=2017) %>%
  dplyr::select(name,x1=val)
d2<-get_data_by_variable("60270",unitLevel=no_uL,year=2018) %>%
  dplyr::select(name, x2=val)
d3<-get_data_by_variable("60270",unitLevel=no_uL,year=2019) %>%
  dplyr::select(name,x3=val)
d4<-get_data_by_variable("60270",unitLevel=no_uL,year=2020) %>%
  dplyr::select(name,x4=val)
d5<-get_data_by_variable("60270",unitLevel=no_uL,year=2021) %>%
```

```
  dplyr::select(name,x5=val)
d6<-get_data_by_variable("60270",unitLevel=no_uL,year=2022) %>%
  dplyr::select(name,x6=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
dane<-d1 %>%
  full_join (d2, by=c("name")) %>%
  full_join (d3, by=c("name")) %>%
  full_join (d4, by=c("name")) %>%
  full_join (d5, by=c("name")) %>%
  full_join (d6, by=c("name"))
x<-as.data.frame(dane)
# Nadanie nazw wierszom i kolumnom
nazwy<-x[,1]
rownames(x)<-tolower(nazwy)
x<-x[,2:7]
colnames(x)<-c("2017","2018","2019","2020","2021","2022")
# Wykres radarowy
radial.plot(x[c(4,5,10,16),],rp.type="p",line.col="blue",grid.unit="%",
  labels=c("2017","2018","2019","2020","2021","2022"),lwd=2,radial.lim=c(0,12),
  clockwise=FALSE,label.prop=c(1.15,1.1,1.1,1.1,1.,1.))
legend(-14,14,c("wielkopolskie","zachodniopomorskie","warmińsko-mazurskie",
  "mazowieckie"),col=1:4,lty=1,lwd=2)
```

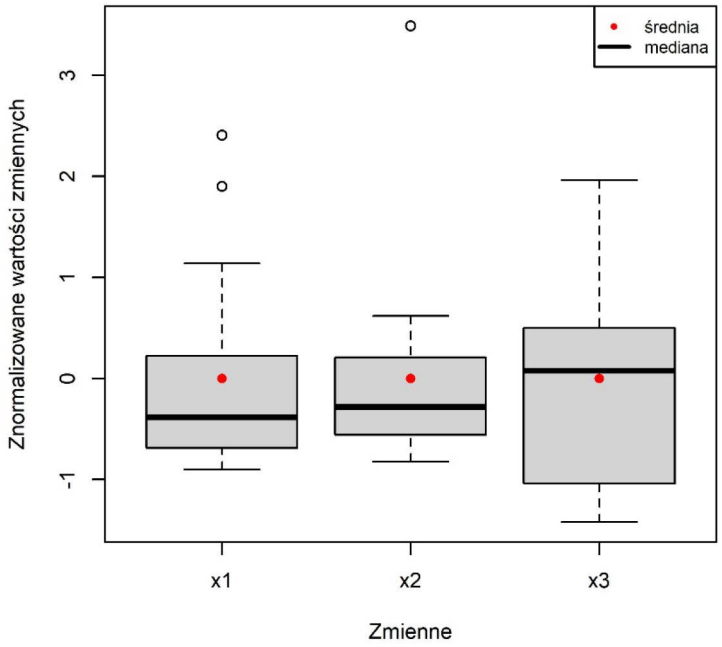
3.4.5. Wykres pudełkowy (*boxplot*)

Do przedstawienia zmienności jednej lub wielu zmiennych można użyć wykresu pudełkowego. Długość pudełka pokazuje zmienność danej zmiennej. Linia pozioma w pudełku przedstawia kwartyl drugi (wartość mediany) $Q_2 = Me$. Dolna linia pudełka to pierwszy kwartyl Q_1, a górna linia pudełka to trzeci kwartyl Q_3. Wartości na rysunku większe od sumy kwartyla trzeciego i trzech odchyleń ćwiartkowych $\left(Q_3 + 3\frac{Q_3 - Q_1}{2}\right)$ oraz mniejsze od różnicy pierwszego kwartyla i trzech odchyleń ćwiartkowych $\left(Q_1 - 3\frac{Q_3 - Q_1}{2}\right)$ są uznawane za nietypowe (*outliers*). Dolna i górna linie poziome połączone linią przerywaną z pudełkiem są odpowiednio wartością minimalną i maksymalną, gdy nie ma obserwacji nietypowych. Jeśli występują obserwacje nietypowe, to dolna i górna linie poziome oznaczają wartość najmniejszą i największą po wykluczeniu obserwacji nietypowych. Położenie linii mediany w pudełku pokazuje skośność rozkładu dla poszczególnych zmiennych. W rozkładach o asymetrii lewostronnej linia mediany jest bliżej wierzchołka pudełka, natomiast w rozkładach o asymetrii prawostronnej – bliżej podstawy pudełka.

W celu graficznej prezentacji wykresu pudełkowego zostaną wykorzystane dane dotyczące zanieczyszczenia powietrza i zgonów ogółem według regionów Polski (NUTS 2) w 2022 r. W analizie uwzględniono następujące zmienne:

x1 – emisję zanieczyszczeń pyłowych w tonach (`"1508"`) na 1 km^2,
x2 – emisję zanieczyszczeń gazowych w tonach (`"2090"`) na 100 km^2,
x3 – zgony ogółem (`"478004"`) na 10 tys. ludności.

Skrypt 3.5 przedstawia wykres pudełkowy dla zmiennych x1, x2 i x3 po standaryzacji (zob. rys. 3.10) oraz wyniki dla 5 parametrów. Dodatkowo kolorem czerwonym na rys. 3.10 oznaczono średnią arytmetyczną.



Rys. 3.10. Wykres pudełkowy dla zmiennych x1, x2 i x3 po standaryzacji

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 3.5

```
library(clusterSim)
library(bdl)
library(dplyr)
options(OutDec=",")
# stałe
rok<-2022
no_uL=3 # (3 - regiony)
# Wczytanie informacji o zmiennych z BDL
v1<-get_data_by_variable("2090",unitLevel=no_uL,year=rok) %>%
  dplyr::select(name, x1=val)
v2<-get_data_by_variable("1508",unitLevel=no_uL,year=rok) %>%
  dplyr::select(name, x2=val)
v3<-get_data_by_variable("478004",unitLevel=no_uL,year=rok) %>%
  dplyr::select(name, x3=val)
v4<-get_data_by_variable("72305",unitLevel=no_uL,year=rok) %>%
  dplyr::select(name, x4=val)
v5<-get_data_by_variable("2018",unitLevel=no_uL,year=rok) %>%
  dplyr::select(name, x5=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
dane <- v1 %>%
  full_join (v2, by=c("name")) %>%
  full_join (v3, by=c("name")) %>%
  full_join (v4, by=c("name")) %>%
```

```
  full_join (v5, by=c("name"))
dane<-as.data.frame(dane)
# Usunięcie z nazwy obiektu wyrazu region
dane<-mutate(dane,name=gsub("REGION ","",name))
x<-as.data.frame(dane)
# Nadanie nazw wierszom i kolumnom
nazwy<-x[,1]
rownames(x)<-tolower(nazwy)
x<-x[,2:6]
# Przeliczenia
attach(x)
x<-mutate(x,x1=round((x1/x5),2))
x<-mutate(x,x2=round((x2/x5)*100,2))
x<-mutate(x,x3=round((x3/x4)*10000,2))
# Usunięcie zbędnych kolumn
x<-x[,1:3]
# Wykres pudełkowy
z<-data.Normalization(x,type="n1")
mean<-apply(z,2,mean)
wyn<-boxplot(z,ylab="Znormalizowane wartości zmiennych",
  horizontal=FALSE,xlab="Zmienne")
points(x=1:3,y=as.vector(mean),col="red",pch=16)
legend("topright",legend=c("średnia","mediana"),
  pch=c(20,NA),lty=c(NA,1),lwd=c(NA,2),
  col=c("red","black"),ncol=1,cex=0.8)
print(wyn$stats)
```

```
           [,1]       [,2]        [,3]
[1,] -0,9001021 -0,8206044 -1,42066420
[2,] -0,6892898 -0,5573610 -1,04007290
[3,] -0,3856559 -0,2827429  0,07588201
[4,]  0,2246760  0,2079948  0,50046125
[5,]  1,1399808  0,6191094  1,96258212
```

Objaśnienia (w kolumnach podane są numery zmiennych):

`[1,]` wartość minimalna (lub najmniejsza bez uwzględniania obserwacji nietypowych),
`[2,]` pierwszy kwartyl,
`[3,]` mediana,
`[4,]` trzeci kwartyl,
`[5,]` wartość maksymalna (lub największa bez uwzględniania obserwacji nietypowych).

3.5. Wykresy zaawansowane z wykorzystaniem programu R dla danych z BDL

Wśród zaawansowanych wykresów można wymienić: graficzną prezentację rozkładów zmiennych (histogram, funkcja gęstości), wykresy rozrzutu danych metrycznych (*scatter plot*) oraz macierz wykresów rozrzutu, wykresy rozrzutu trzech zmiennych metrycznych (*bubble plot*), warunkowe wykresy rozrzutu (*trellis plot*), prezentację danych w postaci szeregów czasowych (*time series plot*), korelogram, wykres piramidowy, mapy cieplne (*heatmap*), wizualizację danych w 3D.

3.5.1. Histogram

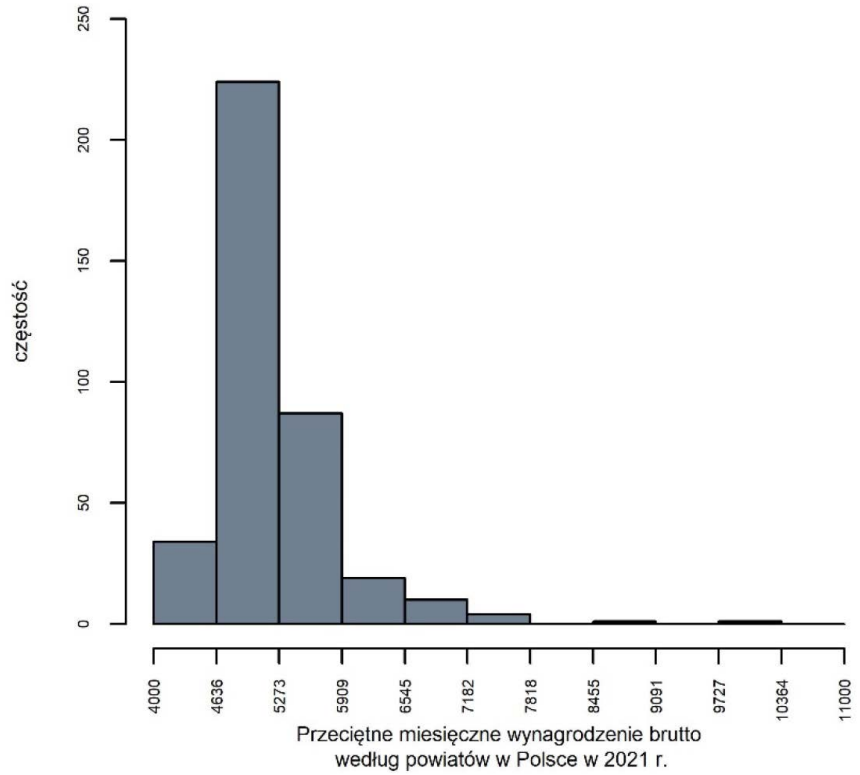
Funkcja `hist()` służy do sporządzenia histogramu zmiennej metrycznej. Liczbę słupków w histogramie można podać samemu (np. `breaks=12`) lub z wykorzystaniem np. formuły Freedmana i Diaconisa (Fox, 2002, s. 87):

$$\left[\frac{n^{1/3}[\max(X)-\min(X)]}{2[Q_3(X)-Q_1(X)]}\right], \tag{3.1}$$

gdzie: $\max(X) - \min(X)$ – rozstęp dla zmiennej X,
$Q_3(X) - Q_1(X)$ – rozstęp ćwiartkowy dla zmiennej X.

Za parametr `breaks` można przyjąć również inne wartości, np. `breaks="Sturges"` (zaokrąglona w górę liczba naturalna z wyrażenia $\log_2 n + 1$) lub `breaks="Scott"` (zaokrąglona w górę liczba naturalna z wyrażenia $\left[\frac{n^{1/3}[\max(X) - \min(X)]}{3{,}5 \cdot s(X)}\right]$). Powstaje wtedy jednak zwykle zbyt mało słupków w histogramie.

Skrypt 3.6 przedstawia histogram dla zmiennej „Przeciętne miesięczne wynagrodzenie brutto według powiatów w Polsce w 2021 r." (zob. rys. 3.11).



Rys. 3.11. Histogram dla zmiennej „Przeciętne miesięczne wynagrodzenie brutto według powiatów w Polsce w 2021 r."

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 3.6

```
library(bdl)
library(dplyr)
options(OutDec=",")
# Wczytanie informacji o zmiennej z BDL
no_uL=5 # (5 - powiat)
dane<-get_data_by_variable("64428",unitLevel=no_uL,year=2021) %>%
  dplyr::select(name,x=val)
x<-as.data.frame(dane)
# Nadanie nazw kolumnom
colnames(x)<-c("powiat","Wynagrodzenie")
# Histogram
attach(x)
breaks=seq(4000,11000,length.out=12)
hist(Wynagrodzenie,breaks=breaks,col="slategrey",ylim=c(0,250),
  xlab="Przeciętne miesięczne wynagrodzenie brutto\n według powiatów w Polsce
  w 2021 r.",ylab="częstość",main="",cex.lab=0.8,cex.axis=0.6,xaxt="n")
axis(1,at=breaks,label=round(breaks,0),cex.axis=0.6,las=2)
detach(x)
```

3.5.2. Funkcja gęstości

Istotnym zagadnieniem w analizie statystycznej jest graficzna prezentacja rozkładu zmiennych. Wymaga ona dla zaobserwowanych danych empirycznych oszacowania funkcji gęstości. Jedną z najbardziej popularnych i dających najlepsze efekty metod jest estymacja jądrowa. Ogólna postać estymatora jądrowego gęstości rozkładu jednej zmiennej dana jest wzorem (Everitt i Hothorn, 2006, s. 112):

$$\hat{f}(x) = \frac{1}{hn}\sum_{i=1}^{n} K\left(\frac{x-x_i}{h}\right), \tag{3.2}$$

gdzie: $K(.)$ – funkcja jądrowa,
x_i – i-ta obserwacja na zmiennej,
x – punkt, w którym estymowana jest funkcja gęstości,
h – szerokość pasma (stała),
n – liczebność próby.

Spośród funkcji jądrowych najczęściej wykorzystywanymi są odpowiednio jądro Gaussa (`kernel="gaussian"`) i jądro Epanechnikova (`kernel="epanechnikov"`) (Everitt i Hothorn, 2006, s. 113-115):

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \tag{3.3}$$

$$K(x) = \begin{cases} 1 - x^2 & x^2 < 1 \\ 0 & \text{w pozostałych przypadkach.} \end{cases} \tag{3.4}$$

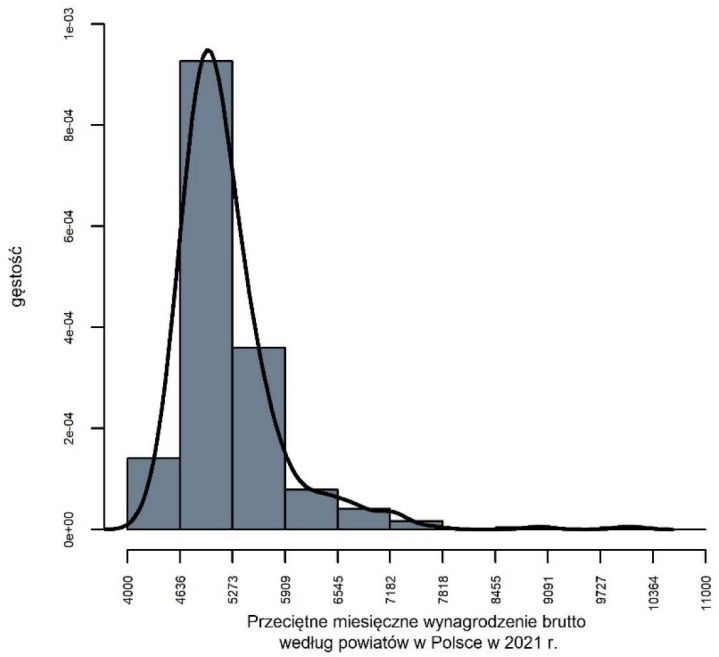
Oprócz tych funkcji jądrowych w funkcji `density` dostępne są jądra: trójkątne (`kernel="triangular"`), prostokątne (`kernel="rectangular"`), dwuwagowe (`kernel="biweight"`), kosinusoidalne (`kernel="cosine"`; `kernel="optocosine"`).

Istotnym zagadnieniem w procedurze estymacji funkcji gęstości jest dobór parametru szerokości pasma. Silverman (1986, s. 48) postuluje, aby domyślna wartość tej stałej obliczana była ze wzoru

$$h = \frac{0{,}9 \cdot \min[s(X); Q_3(X) - Q_1(X)]}{1{,}34 \cdot n^{-1/5}}, \tag{3.5}$$

gdzie: $s(X)$ – odchylenie standardowe dla zmiennej X,
$Q_3(X) - Q_1(X)$ – odchylenie ćwiartkowe dla zmiennej X.

Niekiedy jednak wartość szerokości pasma jest zbyt mała i w rzeczywistych problemach estymacyjnych dostajemy zbyt „postrzępioną" krzywą rozkładu. W takich przypadkach należy zwiększyć stałą, dobierając heurystycznie jej ostateczną wartość. W przypadku krańcowym, tzn. gdy $h \to \infty$, krzywa rozkładu przybiera postać prostej równoległej do osi odciętych. Parametr *n* funkcji `density` oznacza liczbę równo oddalonych punktów służących do estymacji funkcji gęstości, a parametr `width` – szerokość pasma.



Rys. 3.12. Histogram z oszacowaną funkcją gęstości dla zmiennej „Przeciętne miesięczne wynagrodzenie brutto według powiatów w Polsce w 2021 r."

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 3.7 przedstawia histogram z oszacowaną funkcją gęstości dla zmiennej „Przeciętne miesięczne wynagrodzenie brutto według powiatów w Polsce w 2021 r." (zob. rys. 3.12).

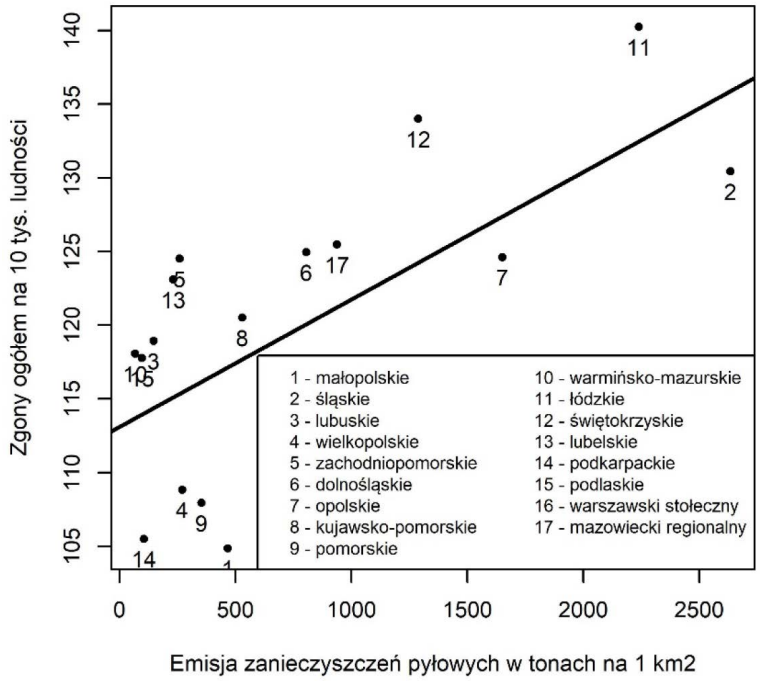
Skrypt 3.7

```
… początek taki sam jak w skrypcie 3.6
# Histogram z oszacowaną funkcją gęstości
attach(x)
breaks=seq(4000,11000,length.out=9)
hist(Wynagrodzenie,freq=FALSE,breaks=breaks,col="slategrey",ylim=c(0,0.001),
  xlab="Przeciętne miesięczne wynagrodzenie brutto\n według powiatów w Polsce
  w 2021 r.",ylab="gęstość",main="",cex.lab=0.8,cex.axis=0.6,xaxt="n")
```

```
axis(1,at=breaks,label=round(breaks,0),cex.axis=0.6,las=2)
# Funkcja gęstości
lines(density(Wynagrodzenie,kernel="gaussian",width=700,n=150),lwd=2)
detach(x)
```

3.5.3. Wykresy rozrzutu danych metrycznych (*scatter plot*)

Wykres rozrzutu danych metrycznych dla dwóch zmiennych (*scatter plot*) pozwala zaobserwować np., czy między zmiennymi zachodzą relacje (liniowe, nieliniowe), czy można wyodrębnić na wykresie klasy obiektów względnie jednorodnych.



Rys. 3.13. Wykres rozrzutu dla dwóch zmiennych x1 i x3 z linią regresji

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 3.8 przedstawia wykres rozrzutu dla zmiennych x1 (emisji zanieczyszczeń pyłowych w tonach na 1 km^2) i x3 (zgonów ogółem na 10 tys. ludności) z dopasowaną funkcją regresji (zob. rys. 3.13).

Skrypt 3.8

```
library(bdl)
library(dplyr)
options(OutDec=",")
# stałe
rok<-2022
no_uL=3 # (3 - regiony)
# Wczytanie informacji o zmiennych z BDL
v1<-get_data_by_variable("2090",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
```

```
v2<-get_data_by_variable("1508",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
v3<-get_data_by_variable("478004",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
v4<-get_data_by_variable("72305",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
v5<-get_data_by_variable("2018",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x5=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
dane <- v1 %>%
  full_join (v2, by=c("id","name")) %>%
  full_join (v3, by=c("id","name")) %>%
  full_join (v4, by=c("id","name")) %>%
  full_join (v5, by=c("id","name"))
dane<-as.data.frame(dane)
# Usunięcie z nazwy obiektu wyrazu region
dane<-mutate(dane,name=gsub("REGION ","",name))
x<-as.data.frame(dane)
# Nadanie nazw kolumnom
nazwy<-x[,2]
x<-x[,2:7]
colnames(x)<-c("region",paste("x",1:5,sep=""))
# Przeliczenia
attach(x)
x<-mutate(x,x1=round((x1/x5),2))
x<-mutate(x,x2=round((x2/x5)*100,2))
x<-mutate(x,x3=round((x3/x4)*10000,2))
# Usunięcie zbędnych kolumn
x<-x[,2:4]
# Nadanie nazw wierszom
rownames(x)<-tolower(region)
# Wykres rozrzutu (scatter plot)
attach(x)
plot(x1,x3,xlab="Emisja zanieczyszczeń pyłowych w tonach na 1 km2",
  ylab="Zgony ogółem na 10 tys. ludności",pch=20)
abline(lm(x3~x1),lwd=2)
text(x1,x3,pos=1)
legend("bottomright",legend=paste(1:17,rownames(x),sep=" - "),
  ncol=2, cex=0.8,bty="o")
detach(x)
```

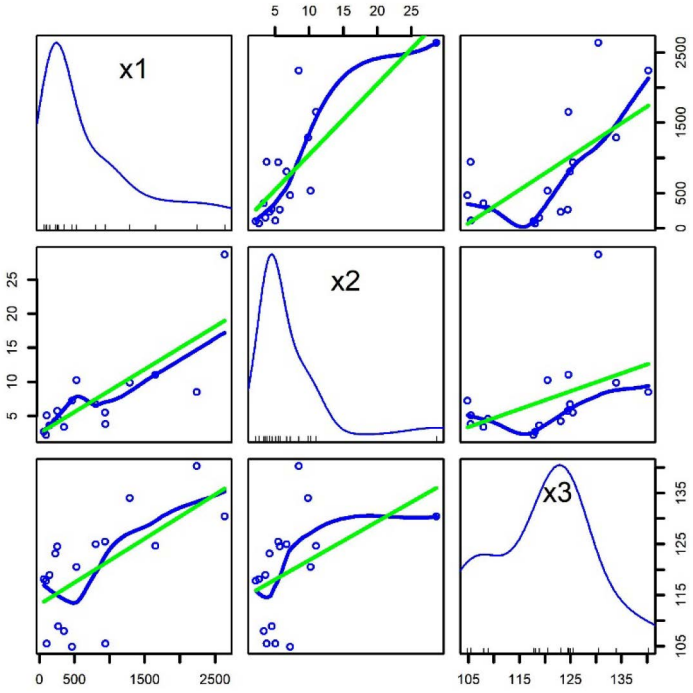
3.5.4. Macierz wykresów rozrzutu

Wykres rozrzutu dwóch zmiennych dla całej tabeli danych można sporządzić z wykorzystaniem funkcji `pairs()`. Lepsza jest jednak funkcja `scatterplotMatrix` pakietu `car`. Pozwala wzbogacić wykres rozrzutu dwóch zmiennych dla całej tabeli danych o informacje graficzne dla każdej zmiennej ujęte na głównej przekątnej (wykres funkcji gęstości – `diagonal=TRUE`), a poza główną przekątną wzbogacić o linie regresji (`regLine`) i regresje nieparametryczne (`smooth`).

Skrypt 3.9 przedstawia macierz wykresów rozrzutu dla zmiennych x1, x2 i x3 (dotyczących zanieczyszczenia powietrza i zgonów ogółem według regionów Polski w 2022 r.) z wymienionymi elementami (zob. rys. 3.14).

Skrypt 3.9

```
Fragment skryptu 3.8 do wykresu rozrzutu
# Macierz wykresów rozrzutu
library(car)
scatterplotMatrix(x,regLine=list(method=lm,lty=1,lwd=2,col="green"),
  smooth=list(spread=FALSE,lty.smooth=1),diagonal=TRUE,plot.points=TRUE,
  pch=1,lwd=1.5)
```



Rys. 3.14. Macierz wykresów rozrzutu dla zmiennych x1, x2 i x3

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

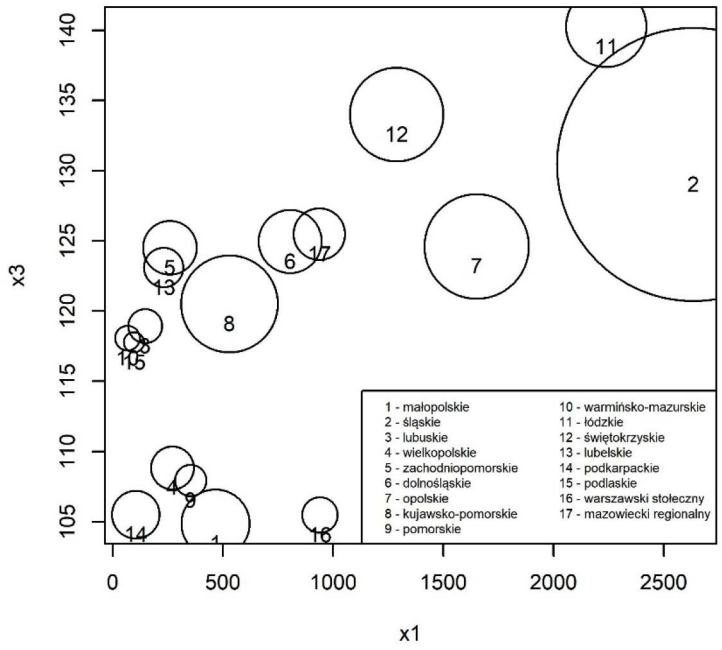
3.5.5. Wykresy rozrzutu trzech zmiennych metrycznych (*bubble plot*)

Specyficznym typem wykresu rozrzutu jest wykres przedstawiający jednocześnie trzy zmienne metryczne (*bubbleplot*). Trzecia zmienna jest reprezentowana przez promień okręgu, którego środek wyznaczają dwie pierwsze zmienne (można do tego celu użyć parametru `cex` funkcji `par`). Skala pomiaru trzeciej zmiennej musi być ilorazowa (promień nie może być ujemny), a rząd wielkości trzeciej zmiennej nie może być nieproporcjonalnie większy od rzędu wielkości pozostałych dwóch zmiennych. Ograniczenia te w pewnym sensie można zniwelować przez odpowiednią normalizację wartości zmiennych (Walesiak i Dudek, 2023).

Skrypt 3.10 przedstawia wykres dla trzech zmiennych metrycznych (dotyczących zanieczyszczenia powietrza i zgonów ogółem według regionów Polski w 2022 r.) x1, x2 i x3 (zob. rys. 3.15).

Skrypt 3.10

```
Fragment skryptu 3.9 do wykresu rozrzutu
# Wykres rozrzutu 3 zmiennych metrycznych
attach(x)
plot(x1,x3,cex=x2,pch=1)
text(x1,x3,pos=1)
legend("topleft",legend=paste(1:17,rownames(x),
  sep=" - "),ncol=2,cex=0.6,bty="o")
detach(x)
```



Rys. 3.15. Wykres rozrzutu trzech zmiennych metrycznych x1, x2 i x3 (*bubbleplot*)

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

3.5.6. Warunkowe wykresy rozrzutu (*trellis plot*)

Wykresy ze zmienną warunkującą (*conditioning variable*) dostępne są za pomocą funkcji `xyplot` w pakiecie `lattice`.

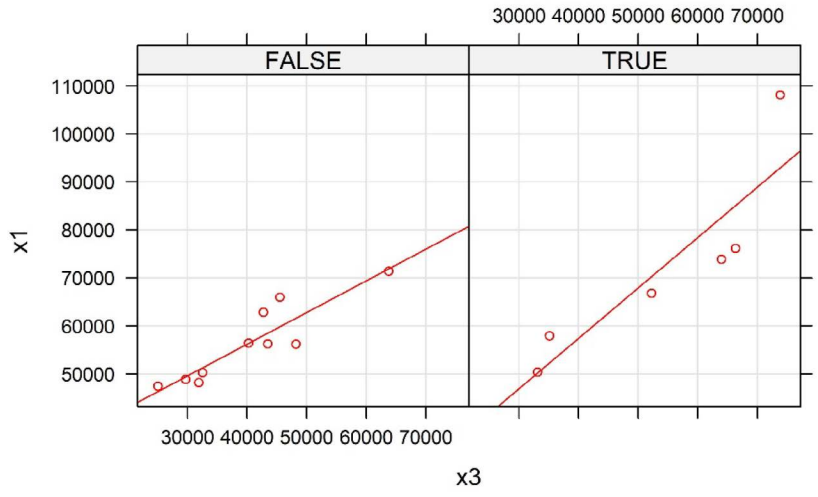
W celu graficznej prezentacji wykresów warunkowych zostaną wykorzystane następujące zmienne (dane dla 2021 r.):

x1 – produkt krajowy brutto Polski na 1 mieszkańca w zł (ceny bieżące) – `"458421"`,
x2 – nakłady inwestycyjne na 1 mieszkańca w zł (ceny bieżące) – `"60520"`,
x3 – produkcja sprzedana przemysłu na 1 mieszkańca w zł (ceny bieżące) – `"148612"`,
x4 – stopa bezrobocia rejestrowanego w % – `"60270"`.

Przykład zastosowania wykresu dla zmiennych x3 (oś odciętych) i x1 (oś rzędnych) ze zmienną warunkującą x2 przedstawia skrypt 3.11 (rys. 3.16).

Skrypt 3.11

```
library(bdl)
library(dplyr)
library(lattice)
options(OutDec=",")
# stałe
rok<-2021
no_uL=2 # (2 - województwo)
# Wczytanie informacji o zmiennych z BDL
d1<-get_data_by_variable("458421",unitParentId=NULL, unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
d2<-get_data_by_variable("60520",unitParentId=NULL,unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
d3<-get_data_by_variable("148612",unitParentId=NULL,unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
d4<-get_data_by_variable("60270",unitParentId=NULL,unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
dane <- d1 %>%
  full_join (d2, by=c("id","name")) %>%
  full_join (d3, by=c("id","name")) %>%
  full_join (d4, by=c("id","name"))
x<-as.data.frame(dane)
# Nadanie nazw wierszom i kolumnom
nazwy<-x[,2]
rownames(x)<-tolower(nazwy)
x<-x[,3:6]
# Wykres warunkowy
attach(x)
t<-xyplot(x1~x3|(x2>=mean(x2)),data=x,type=c("p","r","g"),pch=1,
  cex=0.7, col="red",asp=1)
print(t)
detach(x)
```



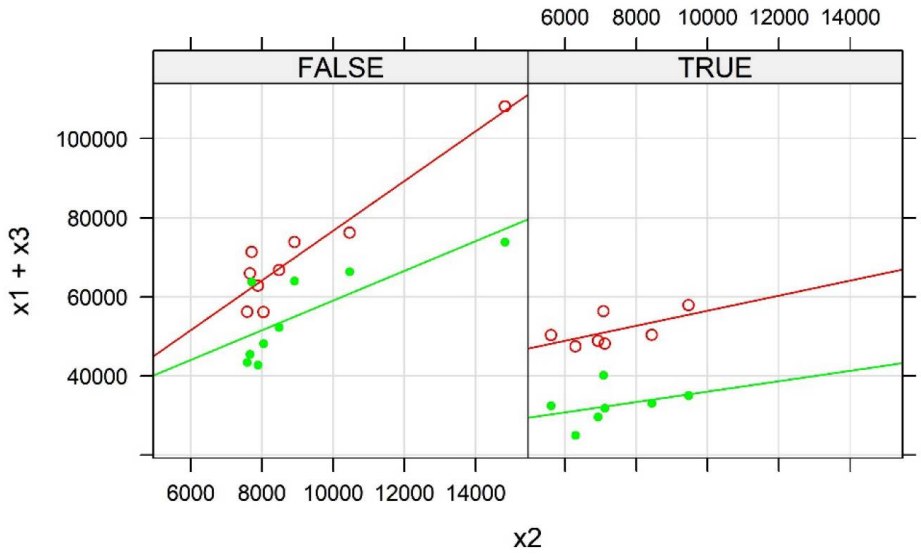
Rys. 3.16. Warunkowy wykres rozrzutu ze zmienną warunkującą x2
(FALSE – wartości zmiennej mniejsze od średniej, TRUE – wartości zmiennej większe równe średniej)

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Przykład zastosowania wykresu dla zmiennych x2 (oś odciętych) i x1 oraz x3 (oś rzędnych, x1 – kolor czerwony, x3 – kolor zielony) ze zmienną warunkującą x4 przedstawia skrypt 3.12 (rys. 3.17).

Skrypt 3.12

```
Fragment skryptu 3.11 do wykresu warunkowego
# Wykres warunkowy
attach(x)
t<-xyplot(x1+x3~x2|(x4>=mean(x4)),data=x,type=c("p","r","g"),
  pch=c(1,20),cex=0.8,col=c("red","green"),asp=1)
print(t)
detach(x)
```



Rys. 3.17. Warunkowy wykres rozrzutu dla zmiennych x2 (oś odciętych) oraz x1 i x3 (oś rzędnych) ze zmienną warunkującą x4 (FALSE – wartości zmiennej mniejsze od średniej, TRUE – wartości zmiennej większe równe średniej)

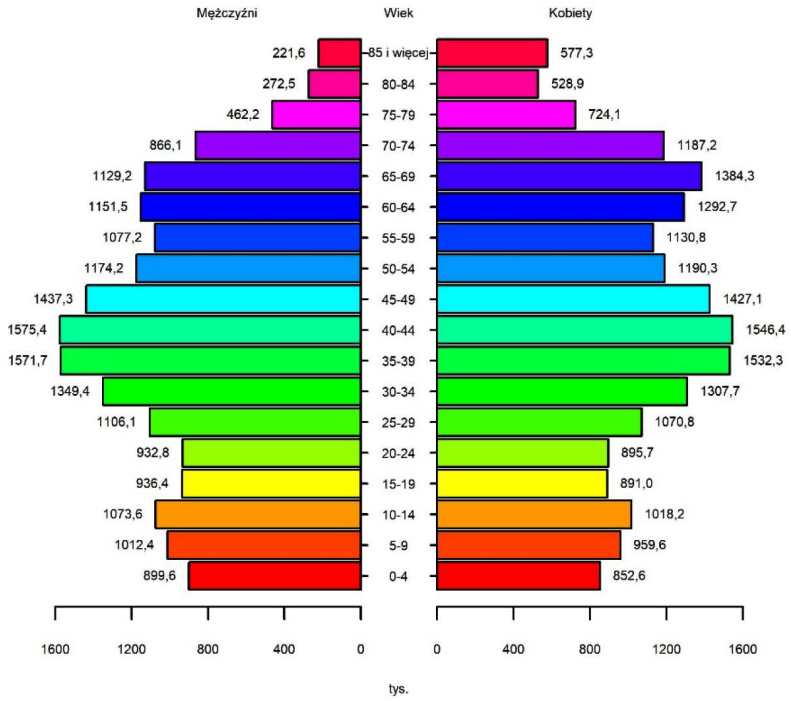
Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

3.5.7. Wykres piramidowy

Wykres piramidowy wykorzystywany jest w demografii do przedstawienia struktury płci i wieku ludności. Piramida wieku zazwyczaj dzielona jest na dwie części: lewą – odnoszącą się do mężczyzn i prawą – odnoszącą się do kobiet. Na podstawie piramidy można określić m.in.:

- ile jest (jaki jest odsetek) osób w danym przedziale wiekowym,
- w których grupach wiekowych pojawia się przewaga liczbowa danej płci,
- czy społeczeństwo danego państwa się starzeje, czy nie,
- czy następuje niż, czy wyż demograficzny,
- jak długo żyją mężczyźni i kobiety.

Wykres piramidowy (rys. 3.18) dla zmiennej „Struktura ludności Polski według płci i wieku w roku 2022” realizuje skrypt 3.13.



Rys. 3.18. Struktura ludności Polski według płci i wieku w roku 2022

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 3.13

```
library(bdl)
library(plotrix)
options(OutDec=",")
# Wczytanie informacji o zmiennych z BDL
no_uL<-0
d1<-get_data_by_variable(c("72301","72302","72303","72304","47711","47736",
  "47724","47712","47725","47728","47706","47715","47721","72243","76018",
  "76019","76020","76021"),unitLevel=no_uL,year=2022)
d1<-t(d1)
d1<-d1[4:21,]
d1<-as.numeric(d1)
d1<-as.data.frame(d1)
d2<-get_data_by_variable(c("72296","72297","72298","72299","47738","47696",
  "47695","47716","47698","47727","47723","47702","47693","72241","76014",
  "76015","76016","76017"),unitLevel=no_uL,year=2022)
d2<-t(d2)
d2<-d2[4:21,]
d2<-as.numeric(d2)
d2<-as.data.frame(d2)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
x<-cbind(d1,d2)
colnames(x)<-c("v1","v2")
rownames(x)<-c("0-4","5-9","10-14","15-19","20-24","25-29","30-34",
  "35-39","40-44","45-49","50-54","55-59","60-64","65-69","70-74",
  "75-79","80-84","85 i więcej")
x<-x/1000
```

```
# wykres piramidowy w demografii
attach(x)
par(mar=pyramid.plot(v1,v2,labels=row.names(x),
  top.labels=c("Mężczyźni","Wiek","Kobiety"),unit="tys.",
  labelcex=0.6,space=0.1,lxcol=rainbow(nrow(x)),
  rxcol=rainbow(nrow(x)),laxlab=c(0,400,800,1200,1600),
  raxlab=c(0,400,800,1200,1600),gap=200,show.values=TRUE))
detach(x)
```

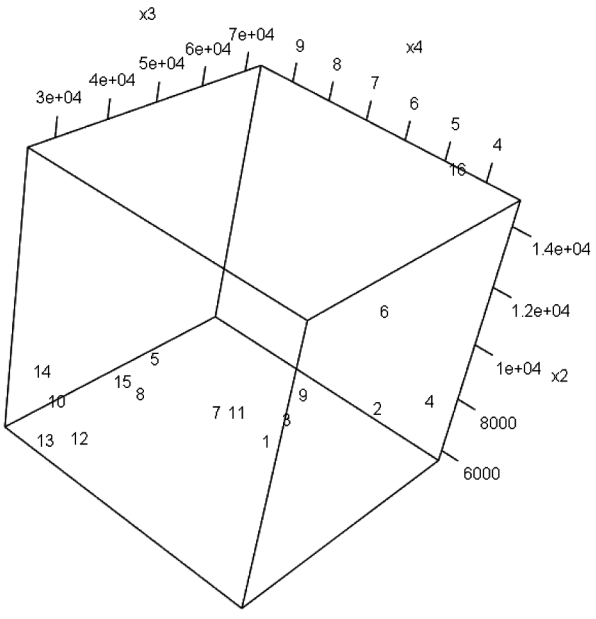
3.5.8. Wizualizacja danych w 3D

Funkcja `plot3d` z pakietu `rgl` jest funkcją przeznaczoną do przedstawiania danych metrycznych (przedziałowych lub ilorazowych) w przestrzeni trójwymiarowej. Funkcja `plot3d` wykorzystuje standard trójwymiarowej grafiki OpenGL. Otwiera ona nowe okienko graficzne z rysunkiem. Można zmieniać jego wielkość oraz go obracać za pomocą myszy. Zapisanie do pliku tego typu rysunków wymaga zastosowania funkcji `rgl.snapshot` pakietu `rgl` o przykładowej składni: `rgl.snapshot("D:/rys1.jpg")`.

Przykład wizualizacji danych w 3D dla zmiennych x1, x2 i x3 ze skryptu 3.11 przedstawiają skrypt 3.14 oraz rys. 3.19.

Skrypt 3.14

```
Fragment skryptu 3.11 do wykresu warunkowego
# Wykres 3D
library(clusterSim)
library(rgl)
attach(x)
no<-rep(1:nrow(x))
plot3d(x2,x3,x4,type="n",size=4)
text3d(x2,x3,x4,text=no)
detach(x)
# rgl.snapshot("Rys_4_17.jpg")
```



Rys. 3.19. Wizualizacja danych w 3D

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Literatura

Biecek, P. (2016). *Odkrywać! Ujawniać! Objaśniać! Zbiór esejów o sztuce prezentacji danych*. Fundacja Naukowa SmarterPoland.pl.

Biecek, P. (2017). *Przewodnik po pakiecie R*. Oficyna Wydawnicza GiS.

Crawley, M. J. (2007). *The R book*. John Wiley & Sons.

Everitt, B. S. i Hothorn, T. (2006). *A Handbook of Statistical Analyses Using R*. Chapman & Hall/CRC.

Fox, J. (2002). *An R and S-PLUS Companion to Applied Regression*. Sage.

Kopczewska, K., Kopczewski, T. i Wójcik, P. (2009). *Metody ilościowe w R. Aplikacje ekonomiczne i finansowe*. CEDEWU.PL.

Pieniążek, M., Szejgiec, B., Zych, M., Ajdyn, A. i Nowakowska G. (2014). *Graficzna prezentacja danych statystycznych. Wykresy, mapy, GIS*. GUS.

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Silverman, B. W. (1986). *Density Estimation*. Chapman and Hall.

Walesiak, M. i Gatnar, E. (red.). (2009). *Statystyczna analiza danych z wykorzystaniem programu R*. Wydawnictwo Naukowe PWN.

Walesiak, M. i Dudek, A. (2023). *clusterSim: Searching for Optimal Clustering Procedure for a Data Set*. R package version 0.51-3. **https://cran.r-project.org/web/packages/clusterSim/**

DOI: 10.15611/2024.45.1.04

4 Wizualizacja danych na mapach

4.1. Formaty w GIS

Dane odwzorowujące kontury krajów, kształty jednostek administracyjnych, linie rzek czy pojedyncze punkty na mapie z samej swojej natury muszą być złożone. Początkowo każdy program z rodziny GIS (*Geographic Information System*) próbował wprowadzać własne standardy zapisu takich danych, a z biegiem lat następował naturalny proces standaryzacji i coraz więcej formatów danych ma charakter otwarty, obsługiwany przez większość aplikacji GIS i pakietów statystycznych dostępnych na rynku. Najważniejsze z nich można podzielić na trzy główne grupy.

a. Dane programów z rodziny GIS. W tej grupie znajdują się m.in. MXD (*Map Exchange Document*) i QGS (używane przez program QGIS).
b. Dane w formatach wykorzystywanych w oprogramowaniu CAD (*Computer Aided Design*), wśród których najpopularniejszy jest format DWG używany przez oprogramowanie AutoCad.
c. Dane w postaci wektorowej:

- KML/KMZ (*Keyhole Markup Language*). Plików KML pierwotnie używano do przeglądania danych geograficznych w Google Earth i do zaznaczania własnych kształtów na Google Maps. Dane w tym formacie są zapisywane w dialekcie języka znaczników XML. Z kolei pliki KMZ to spakowane pliki zawierające główny plik KML i powiązane pliki pomocnicze.
- OpenStreetMap. Projekt OpenStreetMap powstał jako inicjatywa społeczności internetowej w reakcji na wprowadzenie ograniczeń licencyjnych i komercjalizację projektu Google Maps jako alternatywa dla tego narzędzia. Ten format jest również oparty na standardzie XML, a jego dwie najważniejsze odmiany to pliki z rozszerzeniem OSM zawierające dane tekstowe oraz dane binarne w plikach z rozszerzeniem PBF. Kompletną, aktualną mapę kuli ziemskiej w tych formatach można pobrać ze strony **https://planet.openstreetmap.org/** (w chwili pisania tego rozdziału mapa po kompresji miała rozmiar 49 GB).
- Shapefile – format używany w wielu aplikacjach z rodziny GIS oraz mający bardzo silne wsparcie w środowisku R. Format ten zostanie opisany w następnym podrozdziale i na nim będą opierać się przykładowe skrypty wizualizujące dane na mapach.

4.2. Wczytywanie map oraz rysowanie mapy i wycinka mapy

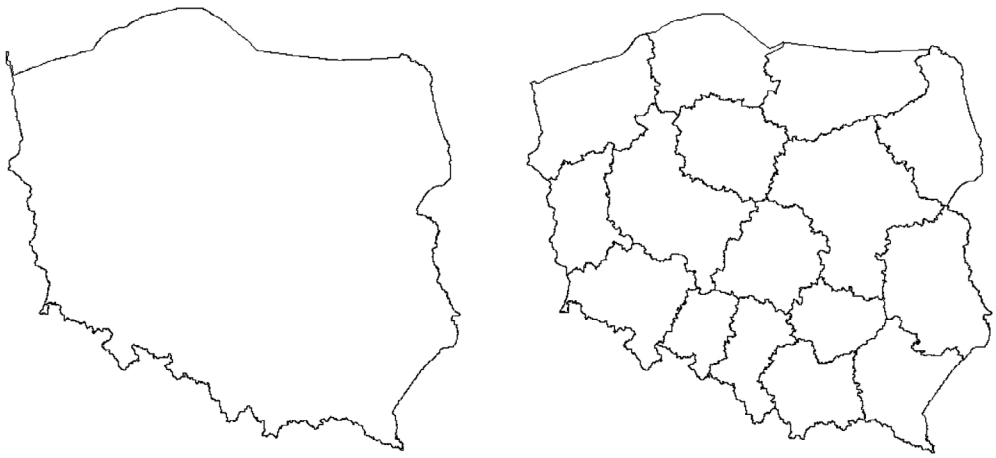
Zbiór danych musi być ściśle związany z mapą. Każda obserwacja ma konkretną lokalizację. Najpopularniejszym formatem jest shapefile z systemu GIS, który obejmuje następujące pliki:

`*.shp` – określa kontury obszarów,
`*.shx` – określa powiązania obszarów z bazą danych,
`*.dbf` – określa bazę zewnętrzną danych,
`*.prj` – przedstawia zapis układu współrzędnych geograficznych (zob. rozdz. 4.3).

Podstawową komendą wczytującą mapy jest `read_sf("*.shp")` z pakietu `sf`. Podstawową komendą do rysowania mapy jest `plot()`.

Do podstawowych stron www do pobrania plików `shapefile` zalicza się:

1. Pliki formatu `shapefile` dla Polski:
   https://dane.gov.pl/pl/dataset/726/resource/29515,panstwowy-rejestr-granic-i-powierzchni-jednostek-podziaow-terytorialnych-kraju-jednostki-administracyjne-format-shapefile-shp/table
2. Pliki formatu `shapefile` dla Polski:
   https://gis-support.pl/baza-wiedzy/dane-do-pobrania/
3. Pliki formatu `shapefile` dla krajów Unii Europejskiej:
   http://ec.europa.eu/eurostat/web/gisco/geodata/reference-data/administrative-units-statistical-units



Rys. 4.1. Mapa Polski oraz województw Polski

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 4.1 pozwala zobrazować graficznie mapę Polski oraz mapę Polski w podziale na województwa (rys. 4.1).

Skrypt 4.1

```
library(sf)
# wczytanie pliku shapefile dla Polski
```

```
pol<-read_sf(dsn=”A00_Granice_panstwa.shp”)
plot(pol$geometry)
windows()
# wczytanie pliku shapefile dla województw Polski
woj<-read_sf(dsn=”A01_Granice_Wojewodztw.shp”)
plot(woj$geometry)
```

Skrypt 4.2 pozwala zobrazować graficznie mapę Polski w podziale na powiaty (rys. 4.2).

Skrypt 4.2

```
library(sf)
# wczytanie pliku shapefile dla powiatów Polski
pow<-read_sf(dsn="A02_Granice_powiatow.shp")
plot(pow$geometry)
```

Rys. 4.2. Mapa Polski w podziale na powiaty

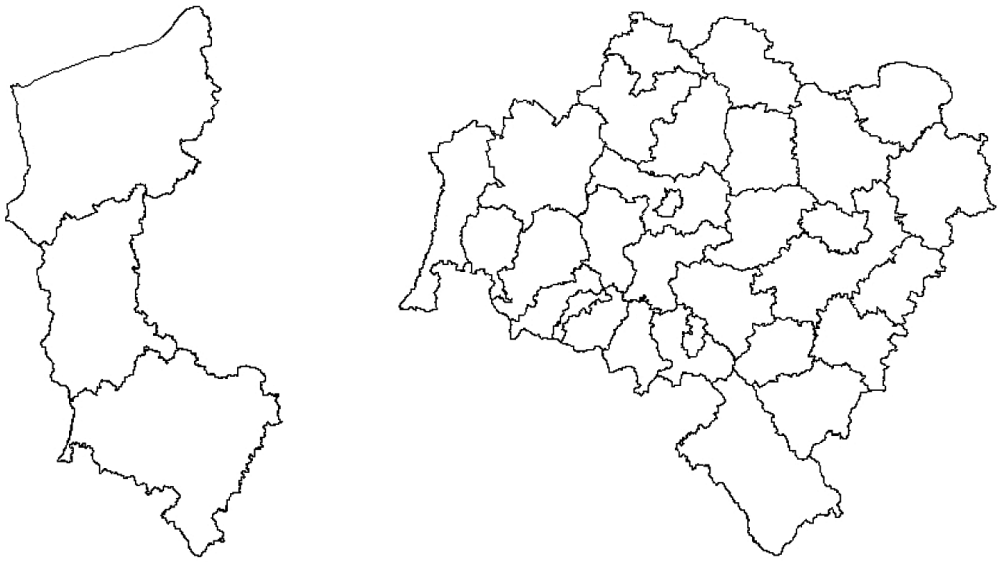
Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Skrypt 4.3 pozwala zobrazować graficznie województwa ściany zachodniej Polski oraz powiaty województwa dolnośląskiego (rys. 4.3).

Skrypt 4.3

```
library(sf)
library(dplyr)
# wczytanie pliku shapefile dla województw Polski
woj<-read_sf(dsn="A01_Granice_wojewodztw.shp")
# Mapa dla wybranych województw
woj<-cbind(woj,1:nrow(woj))
woj_wyb<-filter(as.data.frame(woj),woj$JPT_NAZWA_ == "dolnośląskie" |
  woj$JPT_NAZWA_ == "lubuskie" | woj$JPT_NAZWA_ == "zachodniopomorskie")
plot(woj_wyb$geometry)
windows()
# wczytanie pliku shapefile dla powiatów Polski
```

```
pow<-read_sf(dsn="A02_Granice_powiatow.shp")
# Mapa dla powiatów województwa dolnośląskiego
pow<-cbind(pow,1:nrow(pow))
pow_wyb<-filter(as.data.frame(pow),
  pow$JPT_JOR_ID == "13410")
plot(pow_wyb$geometry)
```



Rys. 4.3. Wycinek mapy Polski (województwa ściany zachodniej) oraz mapa powiatów województwa dolnośląskiego

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

4.3. Wyznaczanie współrzędnych geograficznych

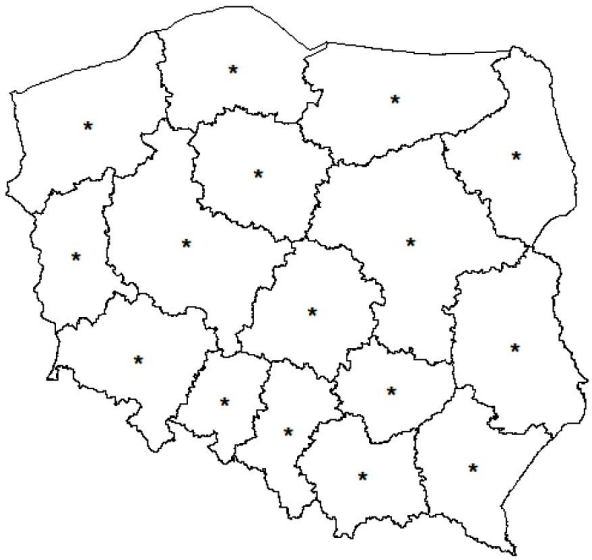
Istotnym elementem tworzenia map jest wyznaczanie współrzędnych środków regionów. Są one niezbędne w nakładaniu etykiet czy słupków na mapę. Podstawową komendą w R służącą wyznaczeniu współrzędnych (x, y) jest `st_centroid(x)`, gdzie `x` oznacza obiekt klasy `shp` wczytany komendą `read_sf()` oraz `gm_coordinates(x)` pakietu `geometries`. Skrypt 4.4 wyznacza współrzędne środków województw.

Skrypt 4.4

```
library(sf)
# wczytanie pliku shapefile dla województw Polski
woj<-read_sf(dsn="A01_Granice_Wojewodztw.shp")
# Obliczenie środków ciężkości województw
centroid<-st_centroid(woj$geometry)
library(geometries)
df<-gm_coordinates(centroid)
crds<-df[,3:4]
print(crds)
# Mapa ze współrzędnymi geograficznymi
plot(woj$geometry)
points(crds,pch="*",cex=1.5)points(crds,pch="*",cex=1.5)
```

Po uruchomieniu poleceń skryptu 4.4 otrzymuje się współrzędne geograficzne środków województw oraz mapę Polski w podziale na województwa (rys. 4.4).

```
         c1       c2
1  17.25014 52.32697
2  16.41150 51.08862
3  19.41631 51.60357
4  20.82249 53.85856
5  15.53608 53.58328
6  18.99270 50.32812
7  15.34248 52.19284
8  17.89895 50.64601
9  20.76926 50.76325
10 20.27056 49.85842
11 18.49008 53.07156
12 22.92958 53.26016
13 21.09985 52.34116
14 22.90293 51.21674
15 18.03833 54.17693
16 22.16977 49.95164
```



Rys. 4.4. Mapa ze współrzędnymi geograficznymi środków województw Polski

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

4.4. Wprowadzanie danych, rysowanie map – warstwy kolorystyczne dla regionów, przedziały klasowe, nanoszenie etykiet i legendy

Wizualizacja danych na mapie wymaga odpowiedniego przygotowania pobranego z BDL zbioru danych. Kolejność obiektów (np. województw, powiatów, gmin) w zbiorze danych pobranych z BDL musi być zgodna z ich kolejnością w pliku `*.dbf`. Zatem dane pobrane z BDL muszą być uporządkowane według kolumny JPT_NAZWA_ z pliku `*.dbf`. Na przykład w skrypcie 4.5 fragment kodu

z fazy 3 pt. „Uporządkowanie województw w zbiorze danych z BDL zgodnie z kolejnością w pliku `A01_Granice_wojewodztw.dbf`" rozwiązuje ten problem.

W nanoszeniu warstw kolorystycznych na mapę regionalną wykorzystuje się komendy: `brewer.pal()` z pakietu `RColorBrewer`; `classIntervals()` z pakietu `classInt`; `findColours()` z pakietu `classInt`. Komenda `brewer.pal(liczba_przedziałów, "typ palety")` wybiera kolory w oparciu o podaną liczbę przedziałów klasowych.

Komenda `classIntervals(zmienna, liczba_przedziałów, style="nazwa")` dzieli zmienną na przedziały według zadanego kryterium:

`style="fixed"` – podajemy własne przedziały klasowe,
`style="equal"` – przedziały o równej długości,
`style="quantile"` – stosuje podział kwantylowy (przedziały równoliczne).

Komenda `findColours(klasy, kolory)` przypisuje przedziałom wygenerowanym komendą `classIntervals()` kolory (np. uzyskane komendą `brewer.pal()`).

Ważnym elementem mapy są etykiety, które mogą być nazwami regionów, wartościami zmiennej. Mogą być nanoszone komendą `text()` o składni:

`text(współrzędna X, współrzędna Y, etykiety, opcje graficzne)`.

Aby podnieść funkcjonalność komendy `text()`, należy wykonać ją w pętli. Legendę nanosi się zwykle komendą `legend()`.

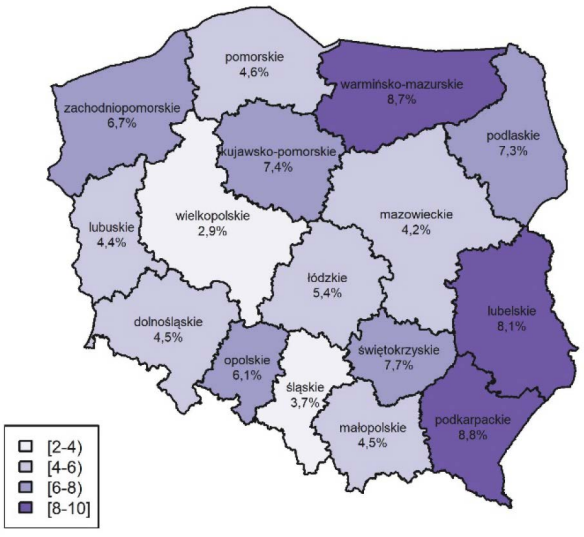
Do prezentacji na mapie wykorzystano dane o stopie bezrobocia rejestrowanego w Polsce według województw w 2022 r. W skrypcie 4.5 w formie komentarzy szczegółowo zaprezentowano poszczególne fazy związane z pobraniem danych z BDL, przygotowaniem danych do prezentacji na mapie oraz powiązaniem danych z jednostkami terytorialnymi występującymi na mapie.

Skrypt 4.5

```
library(sf)
library(classInt)
library(RColorBrewer)
library(bdl)
library(dplyr)
options(OutDec=",")
# Faza 1. Pobranie danych z BDL i zapisanie w odpowiednim formacie
dane<-get_data_by_variable("60270",unitLevel=2,year=2022) %>%
  dplyr::select(id,name, x=val)
x<-as.data.frame(dane)
# Nadanie nazw kolumnom
colnames(x)<-c("id","województwo","Bezrobocie_2022")
# Faza 2. Wczytanie pliku shapefile wraz z plikami pomocniczymi dla województw
regiony<-read_sf(dsn="A01_Granice_wojewodztw.shp")
# Faza 3. Uporządkowanie województw w zbiorze danych z BDL
# zgodnie z kolejnością w pliku A01_Granice_wojewodztw.dbf
if(!is.null(regiony$JPT_NAZWA_)){
  w<-factor(regiony$JPT_NAZWA_)
}
nazwy<-data.frame(województwo=w)
```

```
x<-mutate(x,województwo=factor(tolower(województwo),levels=levels(w)))
xx<-inner_join(nazwy,x)
row.names(xx)<-xx$województwo
# Faza 4. Przygotowanie zmiennej do prezentacji na mapie
zmienna<-xx$Bezrobocie_2022
# Faza 5. Przedziały klasowe i kolory
przedzialy<-4
kolory<-brewer.pal(przedzialy, name="Purples")
klasy<-classIntervals(zmienna,przedzialy,style="fixed",cutlabels=FALSE,
  fixedBreaks=c(2,4,6,8,10),dataPrecision=1)
tabela.kolorow<-findColours(klasy, kolory)
# Faza 6. Mapa + warstwy kolorystyczne
plot(regiony$geometry, col=tabela.kolorow)
title(main="Stopa bezrobocia według województw w 2022 r.")
# Faza 7. Legenda
leg=names(attr(tabela.kolorow, "table"))
leg=gsub(",", "-", leg)
leg=gsub("\\.", ",", leg)
legend("bottomleft", legend=leg, fill=attr(tabela.kolorow, "palette"),
  cex=0.8, bty="o")
# Faza 8. Współrzędne środków regionów (do prezentacji etykiet)
centroid<-st_centroid(regiony$geometry)
library(geometries)
df<-gm_coordinates(centroid)
crds<-df[,3:4]
nazwy<-row.names(xx)
# Faza 9. Etykiety (nazwy województw + stopa bezrobocia w %)
stopa<-xx$Bezrobocie_2022
stopa<-gsub("\\.", ",", stopa)
for(i in 1:16){
  text(crds[i,1], crds[i,2],
  labels=paste(nazwy[i],paste(stopa[i],"%",sep=""),sep="\n"), cex=0.6)
}
```

Po uruchomieniu poleceń skryptu 4.5 otrzymuje się wizualizację na mapie stopy bezrobocia według województw Polski w 2022 r. (rys. 4.5).



Rys. 4.5. Wizualizacja na mapie stopy bezrobocia według województw Polski w roku 2022

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

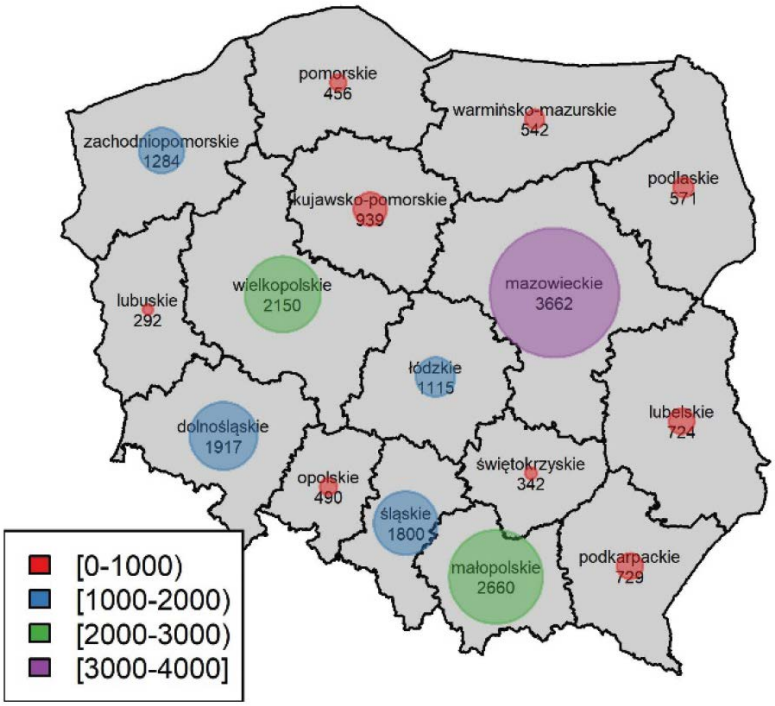
Inną formą prezentacji na mapie jest wykres bąbelkowy. Do prezentacji na mapie wykorzystano dane o liczbie osób ćwiczących tenis w sekcjach sportowych w 2018 r. według województw (skrypt 4.6).

Skrypt 4.6

```
library(sf)
library(classInt)
library(RColorBrewer)
library(bdl)
library(dplyr)
library(plotrix)
options(OutDec=",")
# Faza 1. Pobranie danych z BDL i zapisanie w odpowiednim formacie
dane<-get_data_by_variable("80651",unitLevel=2,year=2018) %>%
  dplyr::select(id,name, x=val)
x<-as.data.frame(dane)
# Nadanie nazw kolumnom
colnames(x)<-c("id","województwo","Liczba_graczy_2018")
# Faza 2. Wczytanie pliku shapefile wraz z plikami pomocniczymi dla województw
regiony<-read_sf(dsn="A01_Granice_wojewodztw.shp")
# Faza 3. Uporządkowanie województw w zbiorze danych z BDL zgodnie
# z kolejnością w pliku A01_Granice_wojewodztw.dbf
if(!is.null(regiony$JPT_NAZWA_)){
  w<-factor(regiony$JPT_NAZWA_)
}
nazwy<-data.frame(województwo=w)
x<-mutate(x,województwo=factor(tolower(województwo),levels=levels(w)))
xx<-inner_join(nazwy,x)
row.names(xx)<-xx$województwo
# Faza 4. Przygotowanie zmiennej do prezentacji na mapie
zmienna<-xx$Liczba_graczy_2018
# Faza 5. Przedziały klasowe i kolory
przedzialy<-4
kolory<-brewer.pal(przedzialy, name="Set1")
klasy<-classIntervals(zmienna, przedzialy, style="fixed", cutlabels=FALSE,
  fixedBreaks=(0:4)*1000,dataPrecision=1)
tabela.kolorow<-findColours(klasy, kolory)
# Faza 6. Mapa + warstwy kolorystyczne
plot(regiony$geometry, col="lightgrey")
title(main="Gracze ćwiczący tenis w sekcjach sportowych w 2018 r.")
# Faza 7. Legenda
leg=names(attr(tabela.kolorow, "table"))
leg=gsub(",", "-", leg)
leg=gsub("\\.", ",", leg)
legend("bottomleft", legend=leg, fill=attr(tabela.kolorow, "palette"),
  cex=1, bty="o")
# Faza 8. Współrzędne środków regionów (do prezentacji etykiet)
centroid<-st_centroid(regiony$geometry)
library(geometries)
df<-gm_coordinates(centroid)
crds<-df[,3:4]
nazwy<-row.names(xx)
```

```
# Faza 9. Etykiety (nazwy województw + stopa bezrobocia w %)
lg<-xx$Liczba_graczy_2018
par_skali=0.00025
for(i in 1:16){
  text(crds[i,1], crds[i,2], labels=paste(nazwy[i],lg[i],sep="\n"), cex=0.6)
  cl=col2rgb(tabela.kolorow[i])
  cl1=rgb(cl[1],cl[2],cl[3],max=255,alpha=120)
  draw.circle(crds[i,1], crds[i,2],lg[i]*par_skali,col=cl1,border=cl1)
}
```

Po uruchomieniu poleceń skryptu 4.6 otrzymuje się wizualizację na mapie liczby osób ćwiczących tenis w sekcjach sportowych w 2018 r. według województw (rys. 4.6).



Rys. 4.6. Wizualizacja na mapie liczby osób ćwiczących tenis w sekcjach sportowych według województw w roku 2018

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

4.5. Mapy wielowarstwowe

Na mapie można umieszczać w dowolnym punkcie inne typy wykresów (np. kołowy, słupkowy, bąbelkowy). Wykres kołowy umieszcza się z wykorzystaniem komendy `floating.pie()` z pakietu `plotrix` (Kopczewska i in., 2009).

Na mapie przedstawiona zostanie (z wykorzystaniem skryptu 4.7) stopa bezrobocia rejestrowanego według województw w 2022 r. (`d4`), a dodatkowo dla każdego województwa zaprezentowany zostanie wykres kołowy przedstawiający strukturę ludności w układzie struktury wieku ludności: przedprodukcyjny (`d1`), produkcyjny (`d2`) i poprodukcyjny (`d3`).

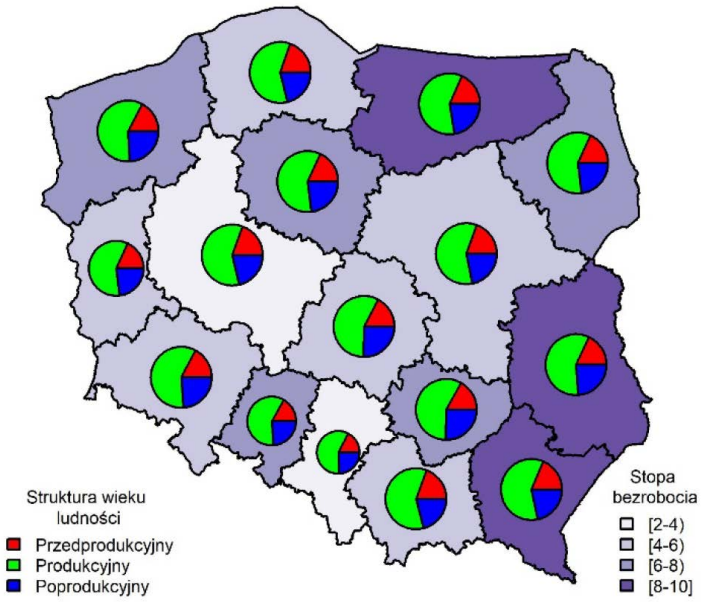
Skrypt 4.7

```
library(bdl)
library(sf)
library(RColorBrewer)
library(classInt)
library(dplyr)
library(plotrix)
# Faza 1. Pobranie danych z BDL i zapisanie w odpowiednim formacie
# Stałe
no_uL=2 # województwo
rok=2022
d1<-get_data_by_variable("149",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
d2<-get_data_by_variable("152",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
d3<-get_data_by_variable("155",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
d4<-get_data_by_variable("60270",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
# Faza 2. Połączenie zmiennych w jedną tabelę (ramkę) danych
data<-d1 %>%
  full_join (d2, by=c("id","name")) %>%
  full_join (d3, by=c("id","name")) %>%
  full_join (d4, by=c("id","name"))
x<-as.data.frame(data)
# Nadanie nazw wierszom
colnames(x)<-c("id","województwo","Przedprodukcyjny","Produkcyjny",
  "Poprodukcyjny","Bezrobocie_2022")
# Faza 3. Wczytanie pliku shapefile wraz z plikami pomocniczymi dla województw
regiony<-read_sf(dsn="A01_Granice_wojewodztw.shp")
# Faza 4. Uporządkowanie województw w zbiorze danych z BDL zgodnie
# z kolejnością w pliku A01_Granice_wojewodztw.dbf
if(!is.null(regiony$JPT_NAZWA_)){
  w<-factor(regiony$JPT_NAZWA_)
}
nazwy<-data.frame(województwo=w)
x<-mutate(x,województwo=factor(tolower(województwo),levels=levels(w)))
xx<-inner_join(nazwy,x)
row.names(xx)<-xx$województwo
# Faza 5. Przygotowanie zmiennej do prezentacji na mapie
zmienna<-xx$Bezrobocie_2022
# Faza 6. Przedziały klasowe i kolory
przedzialy<-4
kolory<-brewer.pal(przedzialy, name="Purples")
klasy<-classIntervals(zmienna, przedzialy, style="fixed", cutlabels=FALSE,
  fixedBreaks=c(2,4,6,8,10),dataPrecision=1)
tabela.kolorow<-findColours(klasy, kolory)
# Faza 7. Mapa + warstwy kolorystyczne
plot(regiony$geometry,col=tabela.kolorow)
title(main="Struktura wieku ludności i stopa bezrobocia\n według województw
  w Polsce w roku 2022")
# Faza 8. Legenda 1
```

```
leg=names(attr(tabela.kolorow, "table"))
leg=gsub(",", "-", leg)
legend("bottomright", title="Stopa\n bezrobocia", legend=leg,
  fill=attr(tabela.kolorow, "palette"), cex=0.7, bty="n")
# Faza 9. Współrzędne środków regionów (do prezentacji wykresów kołowych)
centroid<-st_centroid(regiony$geometry)
library(geometries)
df<-gm_coordinates(centroid)
crds<-df[,3:4]
# Faza 10. Wykresy kołowe
radius<-rep(0.5,16)
radius[6]<-0.35
radius[7]<-0.45
radius[8]<-0.4
for(i in 1:16){
  floating.pie(crds[i,1], crds[i,2], as.matrix(xx[i, 3:5]), radius=radius[i],
  col=rainbow(3))
}
# Faza 11. Legenda 2
leg<-colnames(xx[,3:5])
legend("bottomleft", title="Struktura wieku\n ludności", legend=leg,
  fill= rainbow(3), col=rainbow(3),cex=0.7,bty="n")
```

Po uruchomieniu poleceń skryptu 4.7 otrzymuje się wizualizację na mapie stopy bezrobocia według województw Polski w 2022 r. oraz wykres kołowy przedstawiający strukturę ludności w układzie struktury wieku ludności: przedprodukcyjnego, produkcyjnego i poprodukcyjnego (rys. 4.7).

W R nie ma automatycznej komendy do nakładania na mapę wykresów słupkowych (Kopczewska i in., 2009). Zastosowana zostanie komenda `rect()`, która wymaga podania współrzędnych dwóch punktów: dolnego lewego rogu prostokąta (x, y) oraz górnego prawego rogu prostokąta (x, y).



Rys. 4.7. Wizualizacja na mapie stopy bezrobocia według województw Polski oraz wykres kołowy przedstawiający strukturę ludności w układzie struktury wieku ludności w roku 2022

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

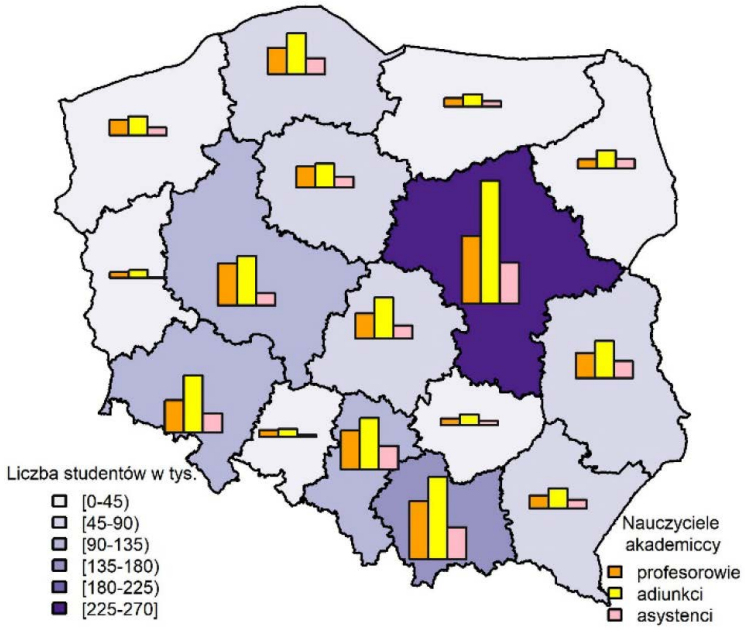
Na mapie przedstawiona zostanie (z wykorzystaniem skryptu 4.8) dla 2021 r. liczba studentów według województw w Polsce w tys. (`d1`) oraz wykres słupkowy dla każdego województwa prezentujący liczbę nauczycieli akademickich według stanowisk (profesorowie – `d2`, adiunkci – `d3`, asystenci – `d4`).

Skrypt 4.8

```
library(sf)
library(bdl)
library(dplyr)
library(classInt)
library(RColorBrewer)
library(plotrix)
# Faza 1. Pobranie danych z BDL i zapisanie w odpowiednim formacie
# Stałe
no_uL=2 # województwo
rok=2021
d1<-get_data_by_variable("1618607",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
d2<-get_data_by_variable("1618661",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
d3<-get_data_by_variable("1618688",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
d4<-get_data_by_variable("1618483",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
# Faza 2. Połączenie zmiennych w jedną tabelę (ramkę) danych
data<-d1 %>%
  full_join (d2, by=c("id","name")) %>%
  full_join (d3, by=c("id","name")) %>%
  full_join (d4, by=c("id","name"))
x<-as.data.frame(data)
# Nadanie nazw wierszom i kolumnom
colnames(x)<-c("id","województwo","x1","x2","x3","x4")
# Faza 3. Wczytanie pliku shapefile wraz z plikami pomocniczymi dla województw
regiony<-read_sf(dsn="A01_Granice_wojewodztw.shp")
# Faza 4. Uporządkowanie województw w zbiorze danych z BDL zgodnie
# z kolejnością w pliku A01_Granice_wojewodztw.dbf
if(!is.null(regiony$JPT_NAZWA_)){
  w<-factor(regiony$JPT_NAZWA_)
}
nazwy<-data.frame(województwo=w)
x<-mutate(x,województwo=factor(tolower(województwo),levels=levels(w)))
xx<-inner_join(nazwy,x)
row.names(xx)<-xx$województwo
# Faza 5. Przygotowanie zmiennej do prezentacji na mapie
zmienna<-xx$x4/1000
# Faza 6. Przedziały klasowe i kolory
przedzialy<-6
kolory<-brewer.pal(przedzialy, name="Purples")
klasy<-classIntervals(zmienna, przedzialy, style="fixed", cutlabels=FALSE,
  fixedBreaks=c(0,45,90,135,180,225,270),dataPrecision=1)
tabela.kolorow<-findColours(klasy, kolory)
```

```
# Faza 7. Mapa + warstwy kolorystyczne
plot(regiony$geometry,col=tabela.kolorow)
title(main="Liczba nauczycieli akademickich i studentów\n według województw
  w Polsce w roku 2021")
# Faza 8. Legenda 1
leg=names(attr(tabela.kolorow, "table"))
leg=gsub(",", "-", leg)
legend("bottomleft", title="Liczba studentów w tys.", legend=leg,
  fill=attr(tabela.kolorow, "palette"), cex=0.7, bty="n")
# Faza 9. współrzędne środków regionów
centroid<-st_centroid(regiony$geometry)
library(geometries)
df<-gm_coordinates(centroid)
crds<-df[,3:4]
# Faza 10. Parametry do rysowania słupków na mapie
aa<-0.4
bb<-0.15
a<-rep(0.4,16) # przesunięcie na osi Y
b<-rep(0.4,16) # przesunięcia na osi X
# przesunięcia słupków dla wybranych województw w pionie
a[c(3,4,5,6,7,8,9,11,12,14,15,16)]<-0
# przesunięcia słupków dla wybranych województw w poziomie
b[6]<-0.2
b[7]<-0.2
kolor1<-"orange" # kolor I słupka
kolor2<-"yellow" # kolor II słupka
kolor3<-"pink" # kolor III słupka
kolor4<-"grey10" # kolor ramki słupka
maks<-max(xx[,c(3:5)])
# Faza 11. Wykresy słupkowe
# I słupek
for(i in 1:16){
  rect(crds[i,1]-b[i]-bb, crds[i,2]-a[i], crds[i,1]-b[i]+bb,
  crds[i,2]-a[i]+3*aa*(xx[,3][i]/maks), angle=45, col=kolor1, border=kolor4)
}
# II słupek
for(i in 1:16){
  rect(crds[i,1]-b[i]+bb, crds[i,2]-a[i], crds[i,1]-b[i]+3*bb,
  crds[i,2]-a[i]+3*aa*(xx[,4][i]/maks), angle=45, col=kolor2, border=kolor4)
}
# III słupek
for(i in 1:16){
  rect(crds[i,1]-b[i]+3*bb, crds[i,2]-a[i], crds[i,1] -b[i]+5*bb,
  crds[i,2]-a[i]+3*aa*(xx[,5][i]/maks), angle=45, col=kolor3, border=kolor4)
}
# Faza 12. legenda 2
legend("bottomright", inset=-0.01,title="Nauczyciele\n akademiccy",
  legend=c("profesorowie","adiunkci","asystenci"),
  fill=c(kolor1,kolor2,kolor3), cex=0.7, bty="n")
```

Po uruchomieniu poleceń skryptu 4.8 otrzymuje się wizualizację na mapie liczby studentów w tysiącach oraz wykresy słupkowe według województw w Polsce prezentujące liczbę nauczycieli akademickich według stanowisk w 2021 r. (rys. 4.8).



Rys. 4.8. Wizualizacja na mapie liczby studentów (w tys.) oraz wykresy słupkowe dla każdego województwa prezentujące liczbę nauczycieli akademickich według stanowisk w roku 2021

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Literatura

Kopczewska, K., Kopczewski, T. i Wójcik, P. (2009). *Metody ilościowe w R. Aplikacje ekonomiczne i finansowe*. CEDEWU.PL.

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

DOI: 10.15611/2024.45.1.05

5 Testy statystyczne

5.1. Podstawy testowania hipotez

Większość prowadzonych obecnie badań opiera się na informacjach pochodzących z próby. Jeśli spełnia ona warunek reprezentatywności, pozwala na prowadzenie wnioskowania statystycznego dotyczącego całej populacji, z której została pobrana. W ramach wnioskowania statystycznego wyróżnia się (oprócz teorii estymacji) dział zwany testowaniem (weryfikacją) hipotez statystycznych. Przez hipotezę statystyczną rozumiemy dowolne przypuszczenie dotyczące populacji generalnej sformułowane bez pełnej informacji na jej temat. Hipotezy statystyczne weryfikowane są za pomocą testów statystycznych. W ramach każdego testu formułowane są dwie wzajemnie wykluczające się hipotezy: hipoteza zerowa H_0 sprawdzana danym testem oraz hipoteza alternatywna H_1 zwana badawczą. Mimo że to hipoteza H_0 jest sprawdzana testem (jej dotyczą podjęte w wyniku przeprowadzenia testu decyzje) z praktycznego punktu widzenia ważniejszą rolę odgrywa hipoteza H_1. W wielu testach bowiem może przyjąć różną postać, w zależności od rodzaju weryfikowanego przypuszczenia czy celu badania. Postać hipotezy H_1 przesądza o kształcie obszaru krytycznego, który może być dwustronny, prawostronny lub lewostronny (por. rys. 5.1). Postać hipotezy H_0 jest z góry ustalona. Z reguły wybiera się jedną hipotezę H_1 jako przeciwstawienie hipotezy H_0. Decyzja o postaci hipotezy H_1 powinna być podjęta przed badaniem (Sobczyk, 2010, s. 118).

Testowanie hipotez statystycznych opiera się na wynikach pochodzących z próby losowej, co oznacza, że przy podejmowaniu ostatecznej decyzji o wyniku testu możliwe jest popełnienie jednego z dwóch rodzajów błędów:

a) błąd I rodzaju – polegający na odrzuceniu hipotezy H_0, gdy jest ona prawdziwa,
b) błąd II rodzaju – który polega na przyjęciu hipotezy H_0, gdy jest ona fałszywa (tj. gdy prawdziwa jest jakaś inna hipoteza H_1) (Paradysz, 2005).

Prawdopodobieństwa wystąpienia błędów I i II rodzaju (oznaczane odpowiednio α i β) są wzajemnie ze sobą powiązane. Zmniejszenie jednego z nich powoduje zwiększenie drugiego. Stąd też w praktycznym zastosowaniu najczęściej wykorzystuje się tzw. *testy istotności*, które przy określonym przez badacza poziomie istotności α zapewniają jak najmniejszą wartość prawdopodobieństwa β.

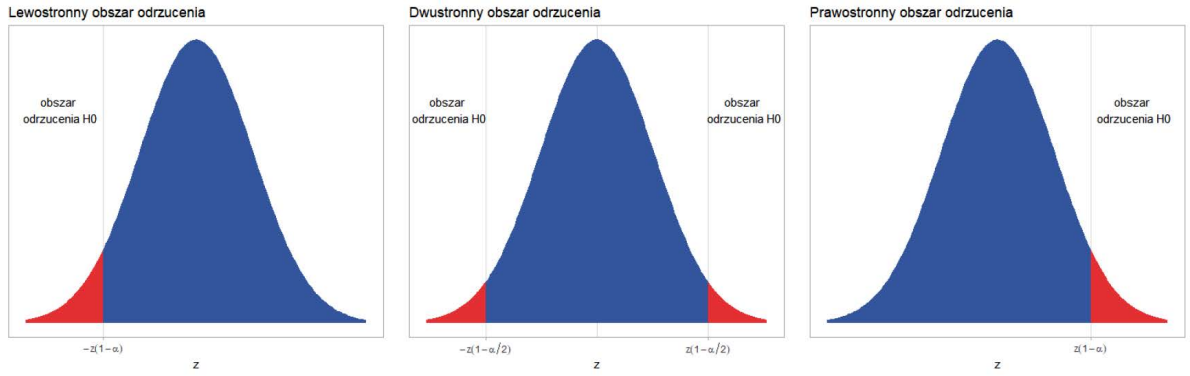
Testowanie hipotezy H_0 w klasycznym ujęciu przeprowadza się w następujących etapach.

1. Sformułowanie problemu badawczego tak, aby mógł on być zweryfikowany w matematyczny sposób.

2. Formalne określenie układu dwóch hipotez: zerowej (H_0) i alternatywnej (H_1).
3. Wybór odpowiedniego testu statystycznego (statystyki sprawdzającej, np. t lub z).
4. Obliczenie wartości empirycznej statystyki testowej z próby.
5. Wyznaczenie dopuszczalnego prawdopodobieństwa popełnienia błędu I rodzaju (czyli poziomu istotności α), a tym samym obszaru odrzucenia H_0, którego granicę określa *wartość krytyczna*.
6. Podjęcie decyzji na podstawie odniesienia wartości statystyki testowej do obszaru odrzucenia H_0.

Decyzja może być co najwyżej dwojakiego rodzaju:

a) odrzucenie hipotezy zerowej H_0 na korzyść hipotezy alternatywnej H_1,
b) brak podstaw do odrzucenia hipotezy zerowej H_0.



Rys. 5.1. Rodzaje obszarów krytycznych

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

W tabeli 5.1 przedstawiono możliwe decyzje oraz konsekwencje ich podjęcia z uwzględnieniem wystąpienia błędów I i II rodzaju.

Tabela 5.1. Możliwe decyzje występujące przy sprawdzaniu hipotez statystycznych

Decyzja statystyczna	Brak podstaw do odrzucenia H_0	Odrzucenie H_0
Nie odrzucaj hipotezy H_0	Właściwa decyzja $1 - \alpha$	Błąd II rodzaju β
Odrzuć H_0	Błąd I rodzaju α – poziom istotności	Właściwa decyzja $1 - \beta$ – moc testu

Źródło: opracowanie własne na podstawie (Walesiak, 1996).

W klasycznym podejściu hipoteza zerowa jest odrzucana, jeśli wartość statystyki testowej należy do obszaru krytycznego. W podejściu stosowanym obecnie zamiast wartości krytycznej brana pod uwagę jest wartość p, którą można zdefiniować jako najostrzejszy poziom istotności, przy którym możemy odrzucić hipotezę H_0. Porównanie wartości p z poziomem istotności α pozwala zatem na podjęcie identycznej decyzji dotyczącej wyniku testu, jak w przypadku podejścia klasycznego. Reguła odrzucenia H_0 dla $p \leq \alpha$ jest prawdziwa dla wszystkich testów statystycznych (Sokołowski, 2004; Szreder, 2019).

Testy statystyczne nazywane są parametrycznymi, jeśli hipotezy dotyczą wartości parametrów statystycznych populacji generalnej (np. średniej, wariancji, wskaźnika struktury, współczynnika korelacji). Jeśli natomiast odnoszą się do postaci rozkładu cechy statystycznej, losowości próby i współzależności cech, to mówimy o testach nieparametrycznych. Testy parametryczne zwykle mogą być stosowane, gdy wartości zmiennych mierzone są na skali przedziałowej lub ilorazowej. Ich implementacja wymaga jednak najczęściej spełnienia pewnych założeń, np. dotyczących liczebności próby lub typu rozkładu wymaganego przez dany test. W przypadku ich niespełnienia, odpowiedników niektórych testów parametrycznych można poszukać wśród testów nieparametrycznych. Idea konstrukcji testów nieparametrycznych z reguły sprowadza się do osłabienia skali pomiarowej z ilorazowej lub przedziałowej na porządkową lub nominalną. Często wykorzystywane są rangi zamiast oryginalnych wartości.

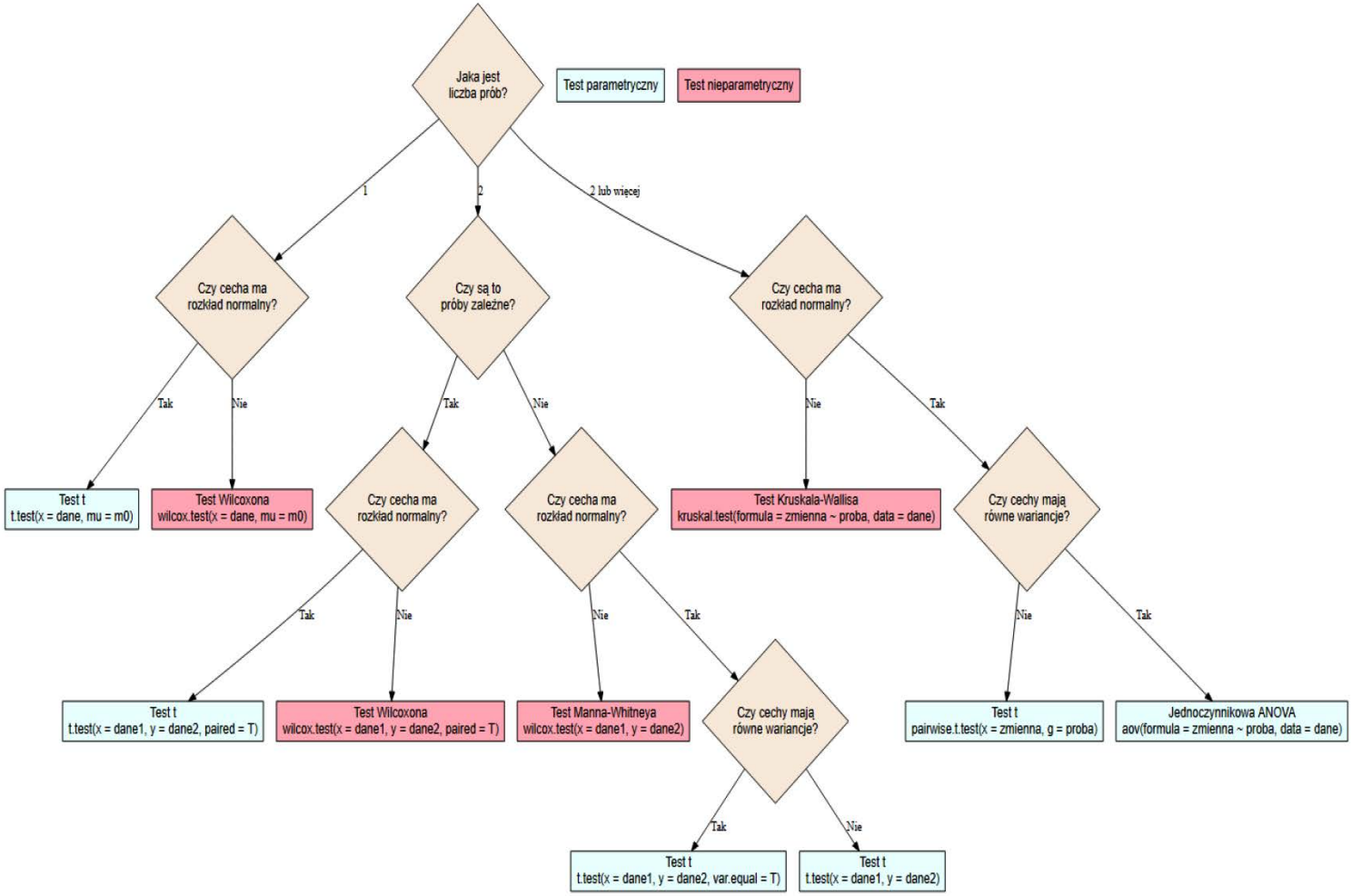
W rozdziale przedstawione zostały zarówno parametryczne, jak i nieparametryczne testy statystyczne najczęściej wykorzystywane w badaniach. Przede wszystkim omówiono testy odnoszące się do cech mierzonych na skali nominalnej, a następnie metody weryfikacji normalności rozkładu będącej warunkiem zastosowania wielu innych metod badawczych. Znaczną część rozdziału poświęcono testowaniu równości średnich w wielu grupach. Zestawianie wszystkich testów statystycznych zaprezentowanych w rozdziale wraz z przyporządkowanymi do nich funkcjami programu R (uwzględniające także rodzaj skali pomiarowej, liczbę badanych prób i relacje występujące między nimi) przedstawiono w tab. 5.2.

Tabela 5.2. Klasyfikacja wybranych testów statystycznych ze względu na skale pomiaru, liczbę badanych prób i relacje między nimi

Próby	Skala nominalna – nieparametryczne	Skala porządkowa – nieparametryczne	Skala ilorazowa – parametryczne
Jedna próba	Test zgodności χ^2 – `prop.test()`	Test zgodności Shapiro-Wilka – `shapiro.test()`, Test Wilcoxona – `wilcox.test()`	Test *t* – `t.test()`
Dwie próby niezależne	Test niezależności χ^2 – `chisq.test()`, Test zgodności χ^2 – `prop.test()`	Test Flignera-Killeena – `fligner.test()`, Test Manna-Whitneya – `wilcox.test()`	Test *F* – `var.test()`, Test *t* – `t.test()`
Dwie próby zależne	Test McNemara – `mcnemar.test()`	Test Wilcoxona – `wilcox.test()`	Test *t* – `t.test()`
K prób niezależnych	Test zgodności χ^2 – `chisq.test()`	Test Flignera-Killeena – `fligner.test()`, Test Kruskala-Wallisa – `kruskal.test()`	Test Bartletta – `bartlett.test()`, ANOVA – `aov()`

Źródło: opracowanie własne na podstawie (Boslaugh, 2012; Kopczewska i in., 2009; Shipunov, 2019).

Na rysunku 5.2 przedstawiono diagram, który prezentuje kolejne kroki podejmowane przy wyborze testu odpowiedniego z punktu widzenia badania.



Rys. 5.2. Etapy wyboru testu statystycznego

Źródło: opracowanie własne na podstawie (Boslaugh, 2012; Shipunov, 2019; Walesiak, 1996).

5.2. Testy dla zmiennych mierzonych na skali nominalnej

Test proporcji

Test proporcji pozwala odpowiedzieć na pytanie, czy odsetki w jednej, dwóch lub więcej grupach różnią się od siebie istotnie. Dla jednej próby układ hipotez został przedstawiony poniżej:

$H_0: p = p_0$,

$H_1: p \neq p_0$ lub $H_1: p > p_0$ lub $H_1: p < p_0$.

Układ hipotez w przypadku dwóch prób jest następujący:

$H_0: p_1 = p_2$,

$H_1: p_1 \neq p_2$ lub $H_1: p_1 > p_2$ lub $H_1: p_1 < p_2$.

Dla k badanych prób hipotezę zerową i alternatywną można zapisać następująco:

$H_0: p_1 = p_2 = p3 = \ldots = p_k$,

$H_1: \exists\ p_i \neq p_j$.

W takim przypadku hipoteza alternatywna oznacza, że co najmniej jeden odsetek różni się istotnie od pozostałych (Sobczyk, 2019).

Funkcja `prop.test` z pakietu `stats` umożliwia przeprowadzanie testu proporcji w programie R. Jako argumenty należy podać wektor, który zawiera licznik badanych odsetków – `x`, oraz wektor zawierający wartości mianownika – `n`. W przypadku jednej próby należy jeszcze dodać argument `p`, którego wartość oznacza weryfikowany odsetek. Dodatkowo można także określić hipotezę alternatywną oraz szerokość wyznaczanego przedziału ufności. W tej funkcji domyślnie używana jest poprawka Yatesa, która ma zastosowanie, jeśli wartości licznika badanych proporcji są mniejsze niż 5. W przypadku większych liczebności należy ustawić opcję `correct=FALSE` (R Core Team, 2023).

Test niezależności

Za pomocą testu niezależności χ^2 można sprawdzić, czy między dwiema cechami jakościowymi występuje zależność (Sobczyk, 2019). Układ hipotez jest następujący:

H_0: zmienne są niezależne,
H_1: zmienne nie są niezależne.

W programie R test niezależności można wywołać za pomocą funkcji `chisq.test()` z pakietu `stats`. Jako argument tej funkcji należy podać tablicę kontyngencji. W przypadku operowania na danych jednostkowych można ją utworzyć poprzez funkcję `table()`. Jeżeli wprowadzamy liczebności ręcznie, należy zadbać o to, żeby wprowadzony obiekt był typu `matrix` (Kopczewska i in., 2016).

Test McNemara

Ten test wykorzystywany jest do sprawdzania istotności różnic pomiędzy dwoma grupami zależnymi. Pozwala stwierdzić, czy w wyniku eksperymentu zmieniła się liczba odpowiadająca danej kategorii (Boslaugh, 2012). Hipotezy stanowią odpowiednio:

H_0: $p_1 = p_2$ – odsetek obserwacji z daną cechą nie zmienił się w wyniku eksperymentu,
H_1: $p_1 \neq p_2$ – odsetek obserwacji z daną cechą zmienił się w wyniku eksperymentu.

Dane wejściowe należy podać w postaci tablicy kontyngencji o wymiarach 2×2 i wstawić je do funkcji `mcnemar.test()`.

5.3. Testowanie normalności

Testy parametryczne z reguły wymagają spełnienia założenia o normalności rozkładu. W celu weryfikacji tego założenia należy wykorzystać jeden z testów normalności.

Test Shapiro-Wilka

W celu formalnego zweryfikowania rozkładu cechy można wykorzystać test Shapiro-Wilka. Układ hipotez w tym teście jest następujący:

H_0: $F(x) = F_0(x)$ – rozkład cechy ma rozkład normalny,
H_1: $F(x) \neq F_0(x)$ – rozkład cechy nie ma rozkładu normalnego.

W przeprowadzonych dotychczas symulacjach wykazano, że test Shapiro-Wilka ma największą moc spośród testów normalności, niemniej jednak jego ograniczeniem jest maksymalna liczba obserwacji[1], która wynosi 5000.

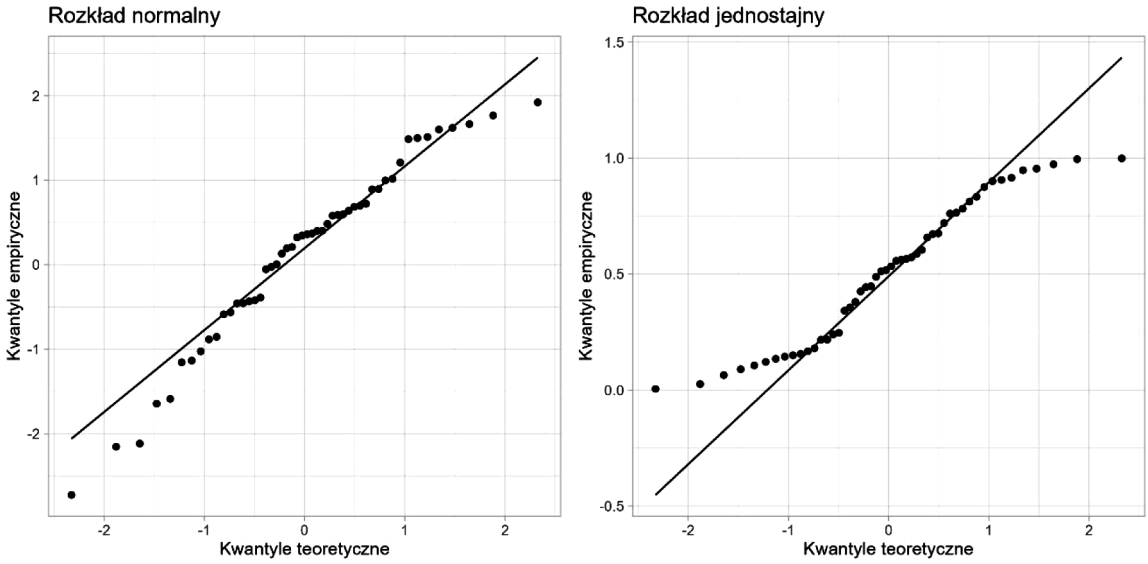
W programie R test Shapiro-Wilka można uruchomić za pomocą funkcji `shapiro.test()` jako argument, podając wektor wartości liczbowych, który chcemy zweryfikować (Kopczewska i in., 2016).

Wykres kwantyl-kwantyl

Normalność rozkładu może także zostać zweryfikowana przez utworzenie wykresu przedstawiającego porównanie wartości oryginalnych oraz odpowiadających im wartości pochodzących z rozkładu normalnego. Dodatkowo prowadzona jest linia regresji pomiędzy otrzymanymi wartościami. Punkty przebiegające w pobliżu tej linii oznaczają, że rozkład tej cechy jest normalny (Shipunov, 2019).

Na rysunku 5.3 przedstawiony jest wykres kwantyl-kwantyl dla 50 wartości wylosowanych z rozkładu normalnego i z rozkładu jednostajnego.

[1] W przypadku liczniejszych prób można wykorzystać test Kołmogorowa-Smirnova.



Rys. 5.3. Porównanie wartości wylosowanych z rozkładu normalnego oraz jednostajnego na wykresie typu kwantyl-kwantyl

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Jak można zauważyć, punkty na wykresie po lewej stronie nie odbiegają znacząco od linii prostej, zatem można przypuszczać, że rozkład tej cechy jest normalny. Z kolei na wykresie po prawej stronie obserwuje się odstępstwo od rozkładu normalnego – wartości na krańcach linii są od niej oddalone.

5.4. Testowanie wariancji

Oprócz założenia o normalności, niektóre metody statystyczne wymagają także równości wariancji (Sobczyk, 2019).

Test F równości wariancji

W tym teście zakładamy następujące hipotezy:

$H_0: s_1^2 = s_2^2,$

$H_1: s_1^2 \neq s_2^2.$

W programie R weryfikacja równości średnich jest możliwa z wykorzystaniem funkcji `var.test()`, która jako argumenty przyjmuje dwa wektory liczbowe.

Test Bartletta

Jeśli chcemy sprawdzić homogeniczność wariancji w więcej niż dwóch grupach, to należy skorzystać z testu Bartletta:

$H_0: s_1^2 = s_2^2 = s_3^2 = \ldots = s_k^2,$

$H_1: \exists_{i,j \in \{1,..,k\}} \; s_i^2 \neq s_j^2.$

Funkcja `bartlett.test()` w programie R umożliwia zastosowanie tego testu. Argumenty do tej funkcji można przekazać na dwa sposoby. Pierwszy polega na przypisaniu do argumentu `x` wektora zawierającego wartości cechy, a do argumentu `g` wektora zawierającego identyfikatory poszczególnych grup. Drugi sposób to zadeklarowanie formuły w postaci `zmienna_analizowa ~ zmienna_grupująca` oraz podanie zbioru danych przypisanego do argumentu `data` (Boslaugh, 2012).

Test Flignera-Killeena

W przypadku cech, które odbiegają od rozkładu normalnego, należy zastosować nieparametryczny test Flignera-Killeena. Za jego pomocą można sprawdzić równość wariancji dla dwóch lub więcej grup. Układ hipotez jest taki sam jak w przypadku testu Bartletta. W programie R ten test jest zaimplementowany w funkcji `fligner.test()` i jego użycie jest analogiczne do testu Bartletta (Biecek, 2019).

5.5. Testowanie wartości przeciętnej

Celem metod opisanych w tym podrozdziale jest wnioskowanie na temat istotności wartości średniej w jednej lub wielu grupach. Skorzystanie z danej metody związane jest zwykle ze spełnieniem określonych założeń dotyczących kształtu rozkładu oraz równości wariancji. Normalność rozkładu można zweryfikować, wykorzystując test Shapiro-Wilka (`shapiro.test()`), natomiast równość wariancji np. testem Bartletta (`bartlett.test()`). W przypadku testowania dwóch prób można wyróżnić sytuację, w której próby są zależne – badane są te same jednostki w dwóch okresach.

Test *t*-średnich

Weryfikacja równości średnich może odbywać się na zasadzie porównania wartości średniej w jednej grupie z arbitralnie przyjętym poziomem lub w dwóch różnych grupach. W pierwszym przypadku rozważamy układ hipotez:

$H_0: m = m_0,$

$H_1: m \neq m_0$ lub $H_1: m < m_0$ lub $H_1: m > m_0,$

natomiast w drugim przypadku hipotezy będą wyglądać następująco:

$H_0: m_1 = m_2,$

$H_1: m_1 \neq m_2$ lub $H_1: m_1 < m_2$ lub $H_1: m_1 > m_2.$

Alternatywnie hipotezę zerową można zapisać jako $m_1 - m_2 = 0$, czyli sprawdzamy, czy różnica pomiędzy grupami istotnie różni się od zera (Sobczyk, 2019).

W funkcji `t.test()` z pakietu `stats` w przypadku jednej próby należy podać argument `x`, czyli wektor z wartościami, które są analizowane oraz wartość, z którą tę średnią porównujemy (argument `mu`, który domyślnie jest równy 0). Dodatkowo w argumencie `alternative` wskazujemy, jaką hipotezę alternatywną bierzemy pod uwagę.

W celu weryfikacji równości średniej w dwóch próbach należy dodać argument `y` z wartościami w drugiej próbie. W tym przypadku mamy także możliwość określenia, czy próby są zależne (ar-

gument `paired`) lub czy wariancja w obu próbach jest taka sama (`var.equal`). Jeżeli wariancje są różne, to program R przeprowadzi test *t* Welcha i liczba stopni swobody nie będzie liczbą całkowitą.

ANOVA

W przypadku większej liczby grup stosuje się jednoczynnikową analizę wariancji (ANOVA). Ta analiza wymaga spełnienia założenia o normalności rozkładu i równości wariancji w badanych grupach (Aczel i Sounderpandian, 2017). Układ hipotez jest następujący:

$$H_0: m_1 = m_2 = m_3 = \ldots = m_k,$$

$$H_1: \exists_{i,j \in \{1,\ldots,k\}} \; m_i \neq m_j.$$

Za pomocą funkcji `aov()` można w R przeprowadzić jednoczynnikową analizę wariancji. Jako argument funkcji należy podać formułę przedstawiającą zależność zmiennej badanej do zmiennej grupującej, wykorzystując w tym celu symbol tyldy (~) w następującym kontekście: `zmienna_analizowana ~ zmienna_grupująca`. Przy takim zapisie należy także w argumencie `data` podać nazwę zbioru danych.

W porównaniu do wcześniej opisanych funkcji, `aov()` nie zwraca w bezpośrednim wyniku wartości *p*. Aby uzyskać tę wartość, należy wynik działania tej funkcji przypisać do obiektu, a następnie na nim wywołać funkcję `summary()`.

W przypadku odrzucenia hipotezy zerowej można przeprowadzić test Tukeya w celu identyfikacji różniących się par, wykorzystując funkcję `TukeyHSD()` i jako argument podając obiekt zawierający wynik ANOVA.

W sytuacji, w której założenia użycia testu parametrycznego nie są spełnione, należy skorzystać z testów nieparametrycznych. W przypadku testowania miar tendencji centralnej różnica pomiędzy testami parametrycznymi a nieparametrycznymi polega na zastąpieniu wartości średniej medianą. Z punktu widzenia obliczeń w miejsce oryginalnych wartości cechy wprowadza się rangi, czyli następuje osłabienie skali pomiarowej – z ilorazowej na porządkową.

Test Wilcoxona

Test Wilcoxona jest nieparametryczną wersją testu *t* (Sobczyk, 2019). Hipotezy w tym teście dotyczą równości rozkładów:

$$H_0: F_1 = F_2,$$

$$H_1: F_1 \neq F_2.$$

Wartość statystyki testowej będzie zależna od typu testu, natomiast w R funkcja, której należy użyć, to `wilcox.test()`. Argumenty tej funkcji są takie same jak w przypadku testu *t*.

Test Kruskala-Wallisa

Z kolei test Kruskala-Wallisa jest nieparametrycznym odpowiednikiem ANOVA (Aczel i Sounderpandian, 2017). Hipotezy są następujące:

$H_0: F_1 = F_2 = F_3 = \ldots = F_k,$

$H_1: \exists_{i,j \in \{1,..,k\}} \quad F_i \neq F_j.$

W programie R korzysta się z funkcji `kruskal.test()`, która przybiera takie same argumenty jak funkcja do metody ANOVA `aov()`. Główną różnicą jest sposób podawania wyniku testu, ponieważ w tym przypadku od razu otrzymujemy wartość *p*. W przypadku odrzucenia hipotezy zerowej należy sprawdzić, które grupy różnią się między sobą. Można to zrobić za pomocą funkcji `pairwise.wilcox.test()`.

5.6. Zastosowania z wykorzystaniem programu R dla danych z BDL

Wszystkie opisane w tym rozdziale testy znajdują się w pakiecie `stats`, który jest wczytywany do pamięci R w momencie uruchomienia programu. Dodatkowo wczytywany jest pakiet `tidyverse` umożliwiający łatwe wczytanie 8 innych pakietów, które stanowią kompletny zestaw narzędzi analityka (Wickham i in., 2019):

`ggplot2` – wizualizacja danych,
`dplyr` – przetwarzanie danych,
`readr` – wczytywanie plików tekstowych,
`purrr` – programowanie funkcyjne,
`tibble` – efektywne ramki danych,
`stringr` – operacje na tekstach,
`forcats` – operacje na czynnikach.

Wczytanie pakietu `tidyverse` uaktywnia także operator przetwarzania potokowego `%>%`. Ponadto podczas instalacji pakietu `tidyverse` pobieranych jest wiele innych pakietów, m.in. do pobierania danych z plików Excela. Pełny opis z przykładami użycia znajduje się na stronie **www.tidyverse.org**. W następnej kolejności wczytywany jest pakiet `bdl` (Szpadel i Kania, 2023).

Dane BDL wykorzystane w dalszych przykładach nie pochodzą z badania reprezentatywnego, dlatego nie spełniają warunku losowości próby wymaganego w stosowaniu testów statystycznych. Posłużono się nimi jedynie w celu zilustrowania sposobu praktycznego zastosowania omówionych metod.

Test proporcji – odsetek hoteli w różnych powiatach

Czy odsetki hoteli trzygwiazdkowych w dwóch turystycznych powiatach w Polsce: karkonoskim i tatrzańskim, różnią się od siebie w sposób istotny na poziomie istotności $\alpha = 0{,}05$? W tym celu wykorzystamy test proporcji. Na początku tego badania określamy układ hipotez:

$H_0: p_1 = p_2$ – odsetek hoteli trzygwiazdkowych w powiecie karkonoskim (p_1) nie różni się istotnie od odsetka w powiecie tatrzańskim (p_2),
$H_0: p_1 \neq p_2$ – odsetek hoteli trzygwiazdkowych w powiecie karkonoskim (p_1) różni się istotnie od odsetka w powiecie tatrzańskim (p_2).

Najpierw należy pobrać dane dotyczące wszystkich hoteli na poziomie powiatów. W BDL w kategorii *Turystyka*, w grupie *Turystyczne obiekty noclegowe i ich wykorzystanie* i podgrupie *Hotele, motele*

i pensjonaty wg kategorii są informacje o liczbie hoteli o liczbie gwiazdek od 1 do 5 oraz tych, które są w trakcie kategoryzacji. Korzystając z metadanych tej cechy, znajdujemy identyfikatory poszczególnych cech, które wykorzystujemy do pobrania danych z wykorzystaniem pakietu `bdl`.

```
hotele5g<-get_data_by_variable(varId=80823,unitLevel=5,year=2018)%>%
  mutate(hotel="h5g")
hotele4g<-get_data_by_variable(varId=80827,unitLevel=5,year=2018)%>%
  mutate(hotel="h4g")
hotele3g<-get_data_by_variable(varId=80831,unitLevel=5,year=2018)%>%
  mutate(hotel="h3g")
hotele2g<-get_data_by_variable(varId=80835,unitLevel=5,year=2018)%>%
  mutate(hotel="h2g")
hotele1g<-get_data_by_variable(varId=80839,unitLevel=5,year=2018)%>%
  mutate(hotel="h1g")
hotelenk<-get_data_by_variable(varId=80843,unitLevel=5,year=2018)%>%
  mutate(hotel="hnk")
```

Po pobraniu zbiorów możemy zauważyć, że mają taką samą liczbę obserwacji, ale nie w każdym powiecie znajdują się hotele o odpowiedniej kategorii – wówczas w danych pojawia się wartość 0. Dodatkowo do każdego z pobranego zbioru danych została dodana nowa kolumna z nazwą kategorii hotelu w celu połączenia wszystkich pobranych zbiorów.

W kolejnym kroku połączymy te zbiory dodając kolejne wiersze.

```
hotele_long<-bind_rows(hotele5g,bind_rows(hotele4g,bind_rows(hotele3g,
  bind_rows(hotele2g,bind_rows(hotele1g,hotelenk)))))
head(hotele_long)
```

W ten sposób uzyskamy nowy obiekt zawierający 2280 obserwacji o nazwie `hotele_long`. Pierwsze 6 obserwacji tego zbioru uzyskano komendą `head()`.

```
  id           name              year    val measureUnitId measureName attrId attributeDescription hotel
  <chr>        <chr>             <chr> <int> <chr>         <chr>       <int> <chr>                <chr>
1 011212001000 Powiat bocheński   2018      0 35            ob.             0 ""                   h5g
2 011212006000 Powiat krakowski   2018      0 35            ob.             0 ""                   h5g
3 011212008000 Powiat miechowski  2018      0 35            ob.             0 ""                   h5g
4 011212009000 Powiat myślenicki  2018      0 35            ob.             0 ""                   h5g
5 011212014000 Powiat proszowicki 2018      0 35            ob.             0 ""                   h5g
6 011212019000 Powiat wielicki    2018      0 35            ob.             0 ""                   h5g
```

Następnie za pomocą funkcji `pivot_wider` z pakietu `tidyr` możemy uzyskać zbiór, który będzie w wierszach zawierał nazwy powiatów, a w kolumnach kategorie hoteli.

```
hotele<-hotele_long %>%
  select(id, name, val, hotel) %>%
  pivot_wider(names_from = hotel, values_from = val)
head(hotele)
```

W wyniku realizacji poleceń w pamięci powstanie obiekt o nazwie `hotele`, a w konsoli zostanie wyświetlone pierwszych 6 obserwacji z tego zbioru.

```
  id           name                h5g   h4g   h3g   h2g   h1g   hnk
  <chr>        <chr>             <int> <int> <int> <int> <int> <int>
1 011212001000 Powiat bocheński      0     0     7     3     0     0
2 011212006000 Powiat krakowski      0     2    10     1     0     0
3 011212008000 Powiat miechowski     0     1     1     1     1     0
```

```
4 011212009000 Powiat myślenicki      0     2     6     0     0     0
5 011212014000 Powiat proszowicki     0     1     2     1     0     0
6 011212019000 Powiat wielicki        0     3    14     2     0     0
```

Następnie na obiekcie `hotele`, wykonujemy konieczne obliczenia – liczby hoteli ogółem w powiecie. Tę operację wykonujemy z wykorzystaniem funkcji `rowSums`, wskazując zakres kolumn z danymi liczbowymi. Potem wybieramy tylko dwa interesujące nas powiaty.

```
hotele_suma <- hotele %>%
  mutate(n=rowSums(.[3:8])) %>%
  filter(name %in% c("Powiat karkonoski", "Powiat tatrzański"))
hotele_suma
```

W wyniku działania tych komend otrzymujemy:

```
  id           name                h5g   h4g   h3g   h2g   h1g   hnk     n
  <chr>        <chr>             <int> <int> <int> <int> <int> <int> <dbl>
1 011216917000 Powiat tatrzański     2    10    21     4     1     0    38
2 030210106000 Powiat karkonoski     1     6    29    10     1     1    48
```

Wartości z tabeli umieszczamy jako argumenty funkcji `prop.test`. Pierwszy z nich `x` zawiera liczebności badanej cechy, czyli liczbę hoteli trzygwiazdkowych, natomiast `n` to liczba wszystkich obserwacji. Dodatkowo jeśli liczebności są większe niż 5, to należy wskazać na brak potrzeby obliczenia poprawki Yatesa, dodając argument `correct=FALSE`.

```
prop.test(x=hotele_suma$h3g, n=hotele_suma$n, correct=FALSE)
```

Rezultatem realizacji komendy jest następujący wynik:

```
2-sample test for equality of proportions without continuity correction
data:  hotele_suma$h3g out of hotele_suma$n
X-squared = 0.23145, df = 1, p-value = 0.6305
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.2616111  0.1585409
sample estimates:
   prop 1    prop 2
0.5526316 0.6041667
```

W wygenerowanym wyniku jest informacja o obliczonych odsetkach – w powiecie tatrzańskim 55% wszystkich hoteli stanowią hotele trzygwiazdkowe, natomiast w powiecie karkonoskim takich hoteli jest 60%. Na podstawie wartości p stwierdzamy, że proporcje w obu powiatach nie różnią się istotnie. Wynika to także z wyznaczonego przedziału ufności, który zawiera wartość 0.

Podany przykład można w bardzo prosty sposób rozszerzyć na większą liczbę prób, dodając kolejne powiaty. Porównajmy zatem analizowane powiaty ze stolicą Polski – Warszawą. Wówczas analizowany układ hipotez będzie następujący:

$H_0 : p_1 = p_2 = p_3$ – odsetek hoteli trzygwiazdkowych w powiecie karkonoskim (p_1) nie różni się istotnie od odsetka w powiecie tatrzańskim (p_2) i w mieście stołecznym Warszawa (p_3),
$H_0 : \exists_{i,j \in \{1,..,k\}} \; p_1 \neq p_2$ – w co najmniej dwóch powiatach odsetki hoteli trzygwiazdkowych istotnie się różnią.

Dodajmy wspominany powiat do zbioru danych i przeprowadźmy test proporcji.

```
hotele_suma_3<-hotele %>%
  mutate(n=rowSums(.[3:8], na.rm=T)) %>%
  filter(name %in% c("Powiat karkonoski","Powiat tatrzański",
  "Powiat m. st. Warszawa"))
prop.test(x=hotele_suma_3$h3g, n=hotele_suma_3$n, correct=FALSE)
```

W wyniku realizacji poleceń otrzymuje się:

```
   3-sample test for equality of proportions without continuity correction
data:  hotele_suma_3$h3g out of hotele_suma_3$n
X-squared = 6.567, df = 2, p-value = 0.0375
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3
0.5526316 0.6041667 0.3936170
```

W takim przypadku hipoteza zerowa zakłada równość wszystkich odsetków, natomiast hipoteza alternatywna występowanie co najmniej jednej pary, w której różnica pomiędzy proporcjami będzie istotna statystycznie. Przy poziomie istotności $\alpha = 0{,}05$ wartość p wskazuje, że należy odrzucić hipotezę zerową na korzyść hipotezy alternatywnej. Nie jest natomiast wskazane, w których powiatach te odsetki się różnią. Po odrzuceniu H_0 można przeprowadzić testy parami, w celu identyfikacji powiatów z istotnymi różnicami w proporcjach.

Test niezależności – zależność między miejscem zamieszkania a wykształceniem

Za pomocą testu niezależności χ^2 sprawdzimy, czy istnieje zależność między miejscem zamieszkania a wykształceniem. Na poziomie $\alpha = 0{,}05$ weryfikujemy następujący układ hipotez:

H_0: między miejscem zamieszkania a wykształceniem nie istnieje istotna zależność,
H_1: między miejscem zamieszkania a wykształceniem istnieje istotna zależność.

Niezbędne dane zebrane zostały podczas Narodowego Spisu Powszechnego 2011. Kategorii do pobrania jest całkiem dużo, dlatego skorzystamy z przeglądarki BDL w celu pozyskania identyfikatorów odpowiednich kategorii. W pierwszej kolejności pobieramy dane i dokonujemy następujących przekształceń:

a) wybór trzech kolumn: `id`, `name` i `val`,
b) zmiana nazwy kolumny `val` na właściwą dla pobranej kategorii,
c) połączenie zbiorów i wybór jednego powiatu.

```
# wykształcenie w miastach
miasta_wyzsze <- get_data_by_variable(varId=396107, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(m_wyzsze=val)
miasta_srednie <- get_data_by_variable(varId=396104, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(m_srednie=val)
miasta_gimnazjum <- get_data_by_variable(varId=396092, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(m_gimnazjum=val)
```

```
miasta_podstawowe <- get_data_by_variable(varId=396089, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(m_podstawowe=val)
# wykształcenie na wsi
wies_wyzsze <- get_data_by_variable(varId=396108, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(w_wyzsze=val)
wies_srednie <- get_data_by_variable(varId=396105, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(w_srednie=val)
wies_gimnazjum <- get_data_by_variable(varId=396093, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(w_gimnazjum=val)
wies_podstawowe <- get_data_by_variable(varId=396090, unitLevel=5) %>%
  select(id, name, val) %>%
  rename(w_podstawowe=val)
miasta_wyksztalcenie <- inner_join(miasta_wyzsze, inner_join(miasta_srednie,
  inner_join(miasta_gimnazjum, miasta_podstawowe))) %>%
  filter(name=="Powiat kaliski")
wies_wyksztalcenie <- inner_join(wies_wyzsze, inner_join(wies_srednie,
  inner_join(wies_gimnazjum, wies_podstawowe))) %>%
  filter(name=="Powiat kaliski")
```

W kolejnym kroku należy utworzyć obiekt, który będzie stanowił element wejściowy do funkcji `chisq.test()` w R. W naszym przypadku taki obiekt powinien zawierać dane o miejscu zamieszkania w wierszach i wykształceniu w kolumnach.

```
mw_wyksztalcenie<-rbind(as.numeric(miasta_wyksztalcenie[1,c(3:6)]),
  as.numeric(wies_wyksztalcenie[1,3:6]))
chisq.test(mw_wyksztalcenie)
```

Uruchomienie tych komend skutkuje uzyskaniem wyniku w konsoli R:

```
Pearson's Chi-squared test
data:  mw_wyksztalcenie
X-squared = 53.086, df = 3, p-value = 1.757e-11
```

Na podstawie wartości *p* możemy stwierdzić, że są podstawy do odrzucenia hipotezy zerowej. Zmienne miejsce zamieszkania i wykształcenie w powiecie kaliskim nie są niezależne.

Test normalności – rozkład stopy bezrobocia rejestrowanego

Za pomocą testu normalności zostanie sprawdzone, czy z istotnością $\alpha = 0{,}05$ rozkład stopy bezrobocia rejestrowanego na poziomie powiatów ma rozkład normalny.

H_0: rozkład stopy bezrobocia rejestrowanego na poziomie powiatów ma rozkład normalny,
H_1: rozkład stopy bezrobocia rejestrowanego na poziomie powiatów nie ma rozkładu normalnego.

Najpierw wyszukamy interesującą nas cechę za pomocą funkcji `search_variables()`.

```
search_variables("stopa bezrobocia rejestrowanego")
```

W wyniku realizacji komendy otrzymuje się:

```
       id subjectId n1       n2              level measureUnitId measureUnitName
    <int> <chr>     <chr>    <chr>           <int>         <int> <chr>
 1 461680 P3559     styczeń  stopa bezroboc~     5            50 %
 2 461681 P3559     luty     stopa bezroboc~     5            50 %
 3 461682 P3559     marzec   stopa bezroboc~     5            50 %
 4 461683 P3559     kwiecień stopa bezroboc~     5            50 %
 5 461684 P3559     maj      stopa bezroboc~     5            50 %
 6 461685 P3559     czerwiec stopa bezroboc~     5            50 %
 7 461686 P3559     lipiec   stopa bezroboc~     5            50 %
 8 461687 P3559     sierpień stopa bezroboc~     5            50 %
 9 461688 P3559     wrzesień stopa bezroboc~     5            50 %
10 461689 P3559     paździe~ stopa bezroboc~     5            50 %
11 461690 P3559     listopad stopa bezroboc~     5            50 %
12 461691 P3559     grudzień stopa bezroboc~     5            50 %
```

Na podstawie zwróconych danych pobieramy dane dla stopy bezrobocia w styczniu (`id=461680`) 2017 r. na poziomie powiatów (`level=5`) za pomocą funkcji `get_data_by_variable()`.

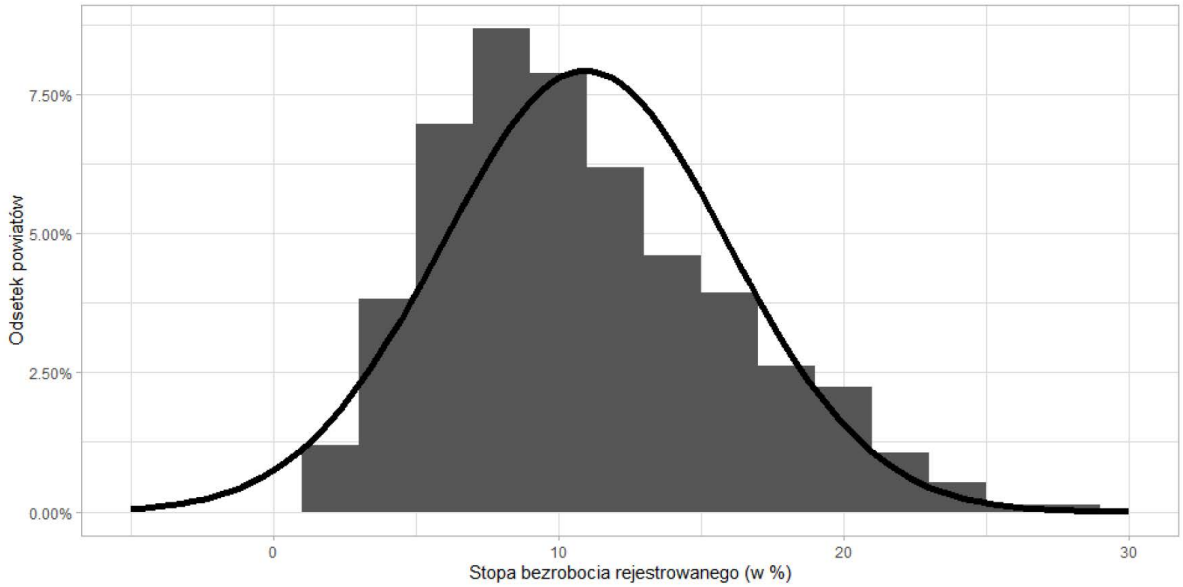
```
bezr_2017<-get_data_by_variable(varId=461680, year=2017, unitLevel=5)
head(bezr_2017)
```

Z wykorzystaniem funkcji `head` w konsoli wyświetlono 6 pierwszych obserwacji nowo utworzonego zbioru `bezr_2017`:

```
  id           name               year    val measureUnitId measureName attrId attributeDescription
  <chr>        <chr>              <chr> <dbl> <chr>         <chr>        <int> <chr>
1 011212001000 Powiat bocheński   2017    6.2 50            %                1 ""
2 011212006000 Powiat krakowski   2017    5.8 50            %                1 ""
3 011212008000 Powiat miechowski  2017    6.8 50            %                1 ""
4 011212009000 Powiat myślenicki  2017    5.4 50            %                1 ""
5 011212014000 Powiat proszowicki 2017    7.9 50            %                1 ""
6 011212019000 Powiat wielicki    2017    6.7 50            %                1 ""
```

Na rysunku 5.4 otrzymane dane przedstawiono na histogramie i dodano krzywą rozkładu normalnego.

```
ggplot(bezr_2017, aes(x = val)) +
  geom_histogram(aes(y =after_stat(density)), binwidth = 2) +
  stat_function(fun = dnorm, linewidth = 2,
                args = list(mean = mean(bezr_2017$val),
                            sd = sd(bezr_2017$val))) +
  scale_y_continuous(labels = scales::percent) +
  xlim(-5,35) +
  xlab("Stopa bezrobocia rejestrowanego (w %)") +
  ylab("Odsetek powiatów") +
  theme_light()
```



Rys. 5.4. Rozkład wartości stopy bezrobocia rejestrowanego na poziomie powiatów z dodaną krzywą rozkładu normalnego

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Już na podstawie powyższej grafiki można stwierdzić, że wartości stopy bezrobocia odbiegają od rozkładu normalnego. Formalnie zostanie to sprawdzone za pomocą testu Shapiro-Wilka i funkcji `shapiro.test()`.

```
shapiro.test(x=bezr_2017$val)
```

Wynikiem działania jest

```
Shapiro-Wilk normality test
data:  bezr_2017$val
W = 0.9653, p-value = 7.686e-08
```

W otrzymanym wyniku wartość p jest bardzo mała – wynosi $7{,}686 \cdot 10^{-8}$ i porównując tę wartość do przyjętego poziomu istotności $\alpha = 0{,}05$, należy podjąć decyzję o odrzuceniu hipotezy zerowej. Rozkład stopy bezrobocia na poziomie powiatów nie jest rozkładem normalnym.

Test wartości przeciętnej dla jednej próby – miesięczne wynagrodzenie brutto

Na podstawie danych o przeciętnym miesięcznym wynagrodzeniu brutto na poziomie powiatów w 2017 r. sprawdźmy, czy średnie wynagrodzenie istotnie różni się od 4000 zł ($\alpha = 0{,}01$). Formalny zapis hipotez jest następujący:

$H_0: m = 4000$,
$H_0: m \neq 4000$.

Rozpoczynamy od pobrania odpowiednich danych i sprawdzenia założenia dotyczącego rozkładu normalnego.

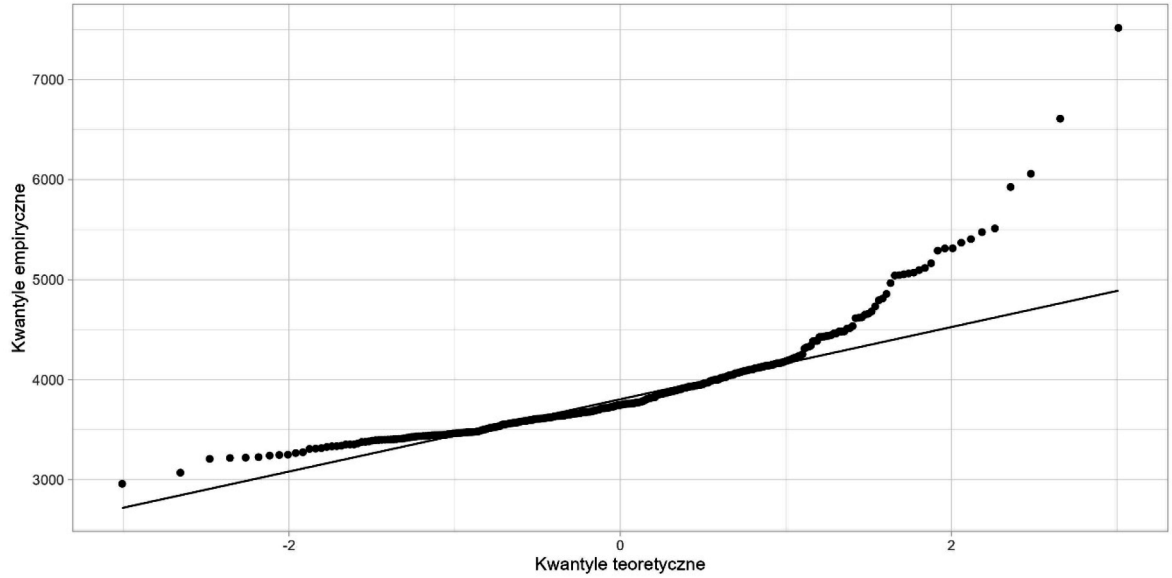
```
wynagrodzenie<-get_data_by_variable(varId=64428, year=2017, unitLevel=5)
shapiro.test(wynagrodzenie$val)
```

W wyniku uruchomienia otrzymuje się następujący wynik w konsoli:

```
Shapiro-Wilk normality test
data:  wynagrodzenie$val
W = 0.81062, p-value < 2.2e-16
```

Na podstawie testu Shapiro-Wilka podejmujemy decyzję o odrzuceniu hipotezy zerowej – wartości przeciętnego wynagrodzenia na poziomie powiatów nie mają rozkładu normalnego. Ponadto na rys. 5.5 został przedstawiony wykres kwantyl-kwantyl.

```
ggplot(wynagrodzenie, aes(sample = val)) +
  stat_qq() +
  stat_qq_line() +
  xlab("Kwantyle teoretyczne") +
  ylab("Kwantyle empiryczne") +
  theme_light()
```



Rys. 5.5. Wykres kwantyl-kwantyl dla przeciętnego miesięcznego wynagrodzeniu brutto na poziomie powiatów w 2017 roku

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Na wykresie można zaobserwować znaczne odstępstwa od linii regresji, zwłaszcza dla dużych wartości analizowanej cechy. Najwyższe wynagrodzenie zaobserwowano w powiecie lubińskim (siedziba KGHM), na drugim miejscu jest miasto na prawach powiatu Jastrzębie-Zdrój (kopalnie węgla kamiennego), a na trzecim miasto stołeczne Warszawa.

Z racji tego, że założenie o rozkładzie normalnym nie jest spełnione, to zamiast klasycznego testu *t* zostanie użyty test Wilcoxona.

```
wilcox.test(x=wynagrodzenie$val, mu=4000)
```

Wynik działania testu Wilcoxona jest następujący:

```
Wilcoxon signed rank test with continuity correction
data:  wynagrodzenie$val
```

```
V = 19329, p-value = 3.504e-15
alternative hypothesis: true location is not equal to 4000
```

Na podstawie wartości *p*, która jest bardzo mała, stwierdzamy, że przeciętny poziom wynagrodzeń na poziomie powiatów istotnie różni się od wartości 4000 zł.

Test wartości przeciętnej dla dwóch prób niezależnych – liczba ćwiczących koszykówkę na poziomie regionów

Sprawdźmy, czy średnia liczba ćwiczących w sekcjach sportowych koszykówkę w podregionach makroregionu północnego i południowego istotnie się różni na poziomie $\alpha = 0{,}01$. Analizowane są różne jednostki, ale ta sama cecha, zatem mamy do czynienia z próbami niezależnymi.

$H_0: m_1 = m_2$ – przeciętna liczba ćwiczących koszykówkę (m_1) w podregionach makroregionu północnego nie różni się istotnie od przeciętnej liczby ćwiczących koszykówkę (m_2) w podregionach makroregionu południowego,

$H_0: m_1 \neq m_2$ – przeciętna liczba ćwiczących koszykówkę (m_1) w podregionach makroregionu północnego różni się istotnie od przeciętnej liczby ćwiczących koszykówkę (m_2) w podregionach makroregionu południowego,

Wykorzystując metadane, znajdujemy identyfikatory interesujących nas zmiennych i dodajemy kolumnę zawierającą identyfikator makroregionu. Wartość `01` oznacza makroregion południowy, a `04` to makroregion północny. Na tej podstawie filtrujemy obserwacje, wybierając jedynie analizowane przekroje terytorialne.

```
koszykowka <- get_data_by_variable(varId=80597, year=2016, unitLevel=4) %>%
  mutate(region=substr(id,1,2)) %>%
  filter(region %in% c("01","04"))
```

Następnie obliczamy interesujące nas statystyki – średnią oraz odchylenie standardowe.

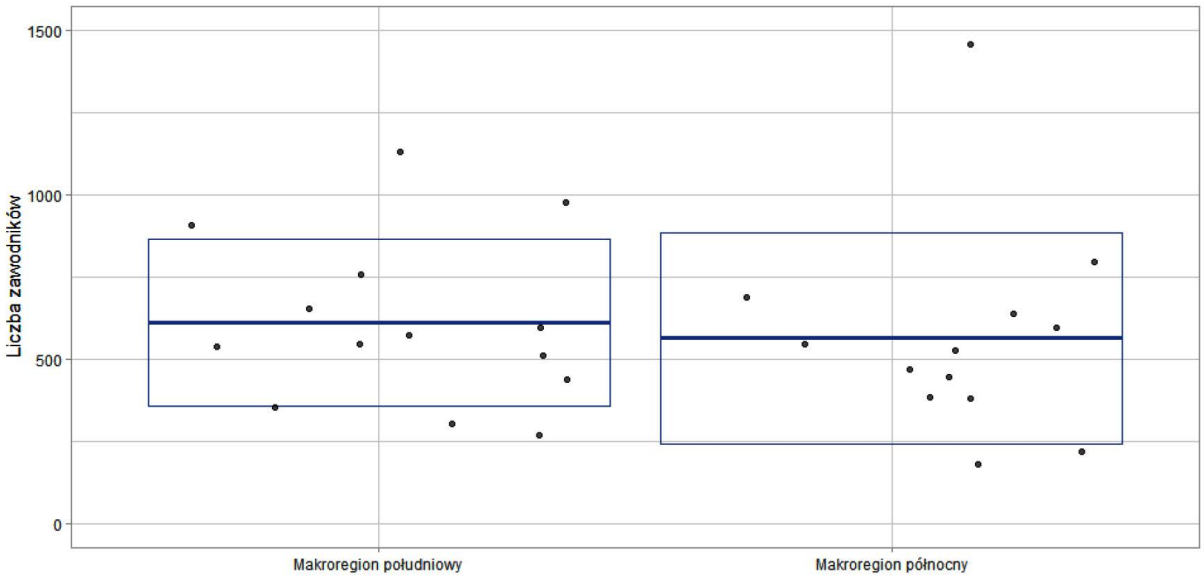
```
koszykowka %>%
  group_by(region) %>%
  summarise(liczba_podregionow=n(),
    srednia = round(mean(val)),
    odchylenie = round(sd(val)))
```

Rezultatem uruchomienia poleceń jest następująca tabela:

```
  region liczba_podregionow srednia odchylenie
  <chr>               <int>   <dbl>      <dbl>
1 01                     14     611        256
2 04                     13     563        321
```

Średnio w podregionach makroregionu południowego koszykówkę trenuje 611 osób, a w makroregionie północnym 563 osoby. Odchylenie standardowe wynosi odpowiednio 256 osób i 321 osób. Na rysunku 5.6 za pomocą punktów przedstawiono liczbę zawodników w poszczególnych podregionach, pogrubiona linia to średnia, a krawędzie prostokąta oznaczają wartości średniej +/– odchylenie standardowe.

```
koszykowka %>%
  group_by(region) %>%
  summarise(srednia = round(mean(val)),
    odchylenie = round(sd(val)))  %>%
  ggplot() +
geom_crossbar(aes(x=region, y=srednia,
  ymin=srednia-odchylenie,ymax=srednia+odchylenie),color="#466AAD") +
  geom_jitter(data = koszykowka, aes(x=region, y=val), alpha=0.5) +
 scale_x_discrete(labels = c("Makroregion południowy","Makroregion północny")) +
  ylab("Liczba zawodników") +
  xlab("") +
  ylim(0,1500) +
  theme_light()
```



Rys. 5.6. Liczba ćwiczących koszykówkę w podregionach oraz średnia i odchylenie standardowe w ramach makroregionów

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Sprawdzenie, czy średnie liczby osób trenujących koszykówkę w wybranych makroregionach różnią się istotnie, wymaga spełnienia założenia o normalności rozkładu obu cech oraz równości wariancji. W pierwszym kroku za pomocą testu Shapiro-Wilka weryfikujemy normalność rozkładu.

```
shapiro.test(koszykowka$val[koszykowka$region=="01"])
```

Wynik dla makroregionu południowego:

```
Shapiro-Wilk normality test
data:  koszykowka$val[koszykowka$region == "01"]
W = 0.94334, p-value = 0.4627
shapiro.test(koszykowka$val[koszykowka$region=="04"])
```

Wynik dla makroregionu północnego:

```
Shapiro-Wilk normality test
data:  koszykowka$val[koszykowka$region == "04"]
W = 0.83762, p-value = 0.01977
```

W obu przypadkach wartości p są większe od przyjętego poziomu istotności $\alpha = 0{,}01$, zatem nie mamy podstaw do odrzucenia hipotezy zerowej. Przyjmujemy, że liczba zawodników trenujących koszykówkę w obu makroregionach ma rozkład normalny.

Drugie założenie dotyczy równości wariancji. Można to zweryfikować, wykorzystując test równości wariancji.

```
var.test(val~region, data=koszykowka)
```

W wyniku uruchomienia uzyskuje się wynik:

```
F test to compare two variances
data:  val by region
F = 0.63474, num df = 13, denom df = 12, p-value = 0.4274
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.195953 2.001457
sample estimates:
ratio of variances
         0.6347435
```

Także w tym przypadku nie ma podstaw do odrzucenia hipotezy zerowej – wariancje obu cech są takie same. W tej sytuacji możemy skorzystać z klasycznego testu t.

```
t.test(val~region, data=koszykowka)
```

Wynikiem działania jest następujący raport z przeprowadzenia testu:

```
Welch Two Sample t-test
data:  val by region
t = 0.42545, df = 22.954, p-value = 0.6745
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -184.6092  280.1917
sample estimates:
mean in group 01 mean in group 04
        610.7143         562.9231
```

Otrzymane wyniki wskazują na to, że nie ma podstaw do odrzucenia hipotezy zerowej, zatem możemy przyjąć, że na poziomie makroregionów średnie liczby zawodników ćwiczących koszykówkę nie różnią się od siebie istotnie statystycznie.

Test wartości przeciętnej dla dwóch prób zależnych – przyrost naturalny w latach 2016 i 2017

Kolejny przykład będzie dotyczył prób zależnych – sprawdzimy średni przyrost naturalny na poziomie podregionów w latach 2016 i 2017 na poziomie istotności $\alpha = 0{,}01$. Analizowane są te same jed-

nostki, ale w dwóch okresach, zatem mamy do czynienia z próbami zależnymi. Pierwszym krokiem jest oczywiście sformułowanie hipotez:

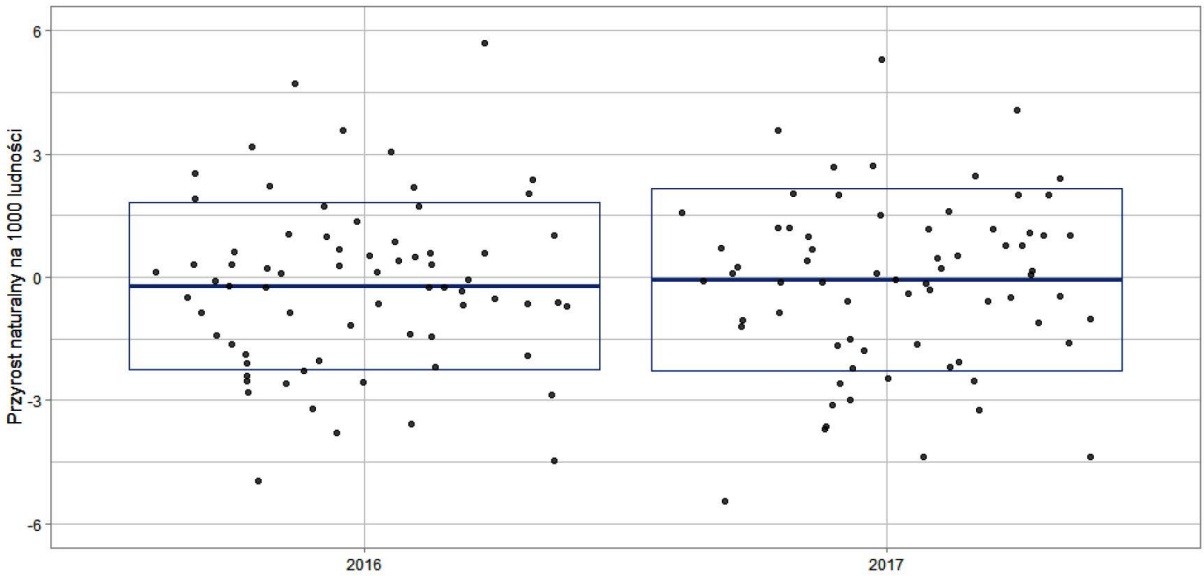
$H_0: m_1 = m_2$ – przeciętny przyrost naturalny w 2016 r. (m_1) nie różni się istotnie od przeciętnego przyrostu naturalnego w 2017 r. (m_2),
$H_0: m_1 \neq m_2$ – przeciętny przyrost naturalny w 2016 r. (m_1) różni się istotnie od przeciętnego przyrostu naturalnego w 2017 r. (m_2).

Następnie pobieramy odpowiednie dane.

```
przyrost<-get_data_by_variable(varId=450551, unitLevel=4, year=c(2016,2017))
```

Na potrzeby przeprowadzenia testu dokonujemy wizualizacji tych danych (por. rys. 5.7).

```
przyrost %>%
  group_by(year) %>%
  summarise(srednia = mean(val),
            odchylenie = sd(val)) %>%
  ggplot() +
  geom_crossbar(aes(x=year, y=srednia,
    ymin=srednia-odchylenie, ymax=srednia+odchylenie), color="#466AAD") +
  geom_jitter(data = przyrost, aes(x=year, y=val), alpha=0.5) +
  ylab("Przyrost naturalny na 1000 ludności") +
  xlab("") +
  ylim(-7,7) +
  theme_light()
```



Rys. 5.7. Przyrost naturalny w podregionach oraz średnia i odchylenie standardowe w latach 2016 i 2017

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

W 2016 r. średni przyrost naturalny wynosił –0,2, natomiast w 2017 r. –0,1. Oczywiście są to wartości przeciętne, ponieważ w rzeczywistości istnieje spore odchylenie od tych wartości. Najwyższy przyrost naturalny odnotowano w podregionie gdańskim – odpowiednio 5,7 i 6,8 w latach 2017 i 2018. Najniższe wartości obserwowane są w podregionie Miasto Łódź, odpowiednio –5,0 i –5,5.

W wyborze odpowiedniego testu pomoże przeprowadzenie testu normalności.

```
shapiro.test(przyrost$val[przyrost$year=="2016"])
```

Wynik testu dla 2016 r.:

```
  Shapiro-Wilk normality test
data:  przyrost$val[przyrost$year == "2016"]
W = 0.9874, p-value = 0.6843
```

```
shapiro.test(przyrost$val[przyrost$year=="2017"])
```

Wynik testu dla 2017 r.:

```
  Shapiro-Wilk normality test
data:  przyrost$val[przyrost$year == "2017"]
W = 0.98303, p-value = 0.4326
```

Dla obu lat nie ma podstaw do odrzucenia hipotezy zerowej – dane mają rozkład normalny. Przechodzimy zatem do testu *t* dla prób zależnych.

```
t.test(val~year, data=przyrost, paired=TRUE)
```

W wyniku realizacji funkcji otrzymuje się:

```
  Paired t-test
data:  val by year
t = -3.4277, df = 72, p-value = 0.00101
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -0.25066797 -0.06631834
sample estimates:
mean difference
     -0.1584932
```

Są podstawy do odrzucenia hipotezy zerowej – średnie przyrosty naturalne na poziomie podregionów różniły się istotnie między latami 2016 i 2017.

Test wartości przeciętnej dla trzech prób niezależnych – wskaźnik rentowności sprzedaży brutto według sekcji PKD

Czy średnie wskaźniki rentowności sprzedaży brutto w wybranych sekcjach PKD na poziomie województw istotnie różnią się między sobą przy $\alpha = 0{,}01$? Do analizy zostały wybrane trzy sekcje: przetwórstwo przemysłowe; budownictwo; handel hurtowy i detaliczny, naprawa pojazdów samochodowych, włączając motocykle. Zdefiniowano hipotezy badawcze:

$H_0: m_1 = m_2 = m_3$ – wartości przeciętne średnich wskaźników rentowności sprzedaży brutto nie różnią się istotnie od siebie w ramach sekcji PKD,

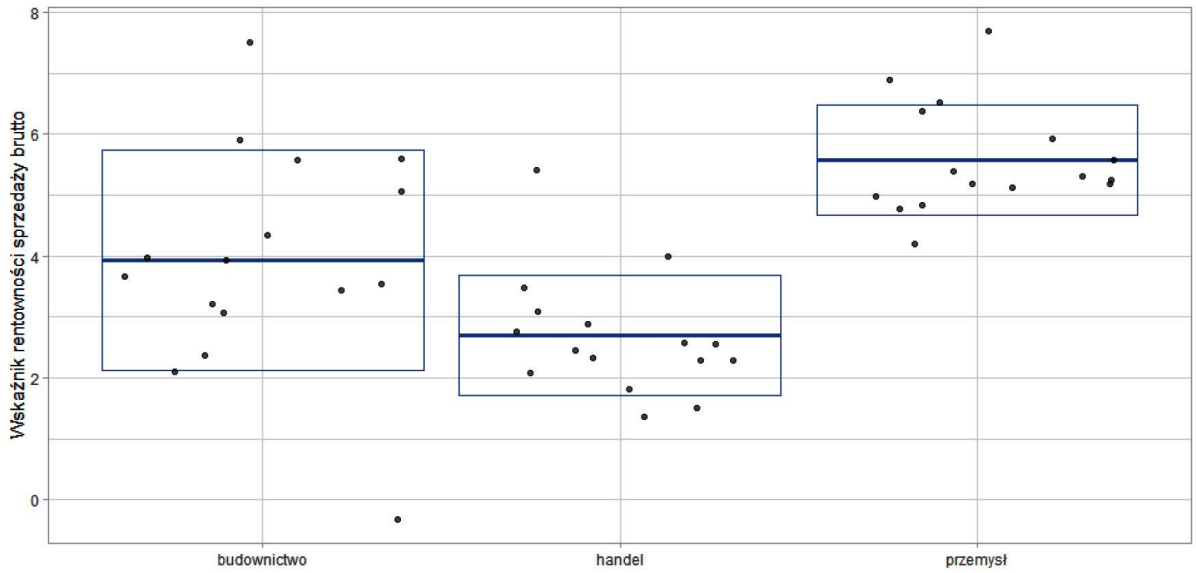
$H_1: \exists_{i,j\in\{1,2,3\}} \; m_i \neq m_j$ – istnieje co najmniej jedna para sekcji PKD, dla których wartości przeciętne średnich wskaźników rentowności sprzedaży brutto różnią się istotnie od siebie.

Dane w BDL znajdują się w kategorii *Finanse przedsiębiorstw*, grupie *Wyniki finansowe przedsiębiorstw* i podgrupie *Wyniki finansowe wg sekcji PKD 2007*. Korzystając z metadanych, znajdujemy odpowiednie identyfikatory cech i pobieramy dane. Do każdego zbioru dodajemy kolumnę z nazwą sekcji i łączymy wszystkie zbiory w jeden, co ułatwi późniejszą analizę.

```
przemysl<-get_data_by_variable(varId=214907, unitLevel=2, year=2017) %>%
  mutate(pkd = "przemysł")
budownictwo<-get_data_by_variable(varId=216234, unitLevel=2, year=2017)%>%
  mutate(pkd = "budownictwo")
handel<-get_data_by_variable(varId=216883, unitLevel=2, year=2017) %>%
  mutate(pkd = "handel")
rentownosc <- bind_rows(przemysl, bind_rows(budownictwo, handel)) %>%
  arrange(id)
head(rentownosc)
```

W wyniku realizacji otrzymuje się zbiór danych `rentownosc`, którego 6 pierwszych wierszy wyświetlonych jest poniżej:

```
      id           name        year    val measureUnitId measureName attrId attributeDescription pkd
  <chr>        <chr>       <chr> <dbl> <chr>         <chr>        <int> <chr>                <chr>
1 011200000000 MAŁOPOLSKIE 2017    6.5 50            %                1 ""                   przemysł
2 011200000000 MAŁOPOLSKIE 2017    5.1 50            %                1 ""                   budownictwo
3 011200000000 MAŁOPOLSKIE 2017    2.9 50            %                1 ""                   handel
4 012400000000 ŚLĄSKIE     2017    4.8 50            %                1 ""                   przemysł
5 012400000000 ŚLĄSKIE     2017    5.6 50            %                1 ""                   budownictwo
6 012400000000 ŚLĄSKIE     2017    2.8 50            %                1 ""                   handel
```



Rys. 5.8. Wskaźnik rentowności sprzedaży brutto w województwach oraz średnia i odchylenie standardowe w ramach sekcji PKD

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Rysunek 5.8 został utworzony z wykorzystaniem poniższych poleceń i przedstawia obserwacje ze zbioru (czarne punkty) oraz wartości średnie i odchylenie standardowe (niebieskie linie) w ramach wybranych sekcji PKD.

```
rentownosc %>%
     group_by(pkd) %>%
     summarise(srednia = mean(val),
       odchylenie=sd(val)) %>%
     ggplot() +
   geom_crossbar(aes(x=pkd, y=srednia,
     ymin=srednia-odchylenie,ymax=srednia+odchylenie),color="#466AAD") +
```

```
  geom_jitter(data = rentownosc, aes(x=pkd, y=val), alpha=0.5) +
    ylab("Wskaźnik rentowności sprzedaży brutto") +
    xlab("") +
    theme_light()
```

Średni wskaźnik rentowności sprzedaży brutto w sekcji budownictwo wynosi 3,9%, w sekcji handel 2,7%, a w przemyśle 5,6%. Powstaje pytanie, czy te wartości istotnie się między sobą różnią. W pierwszej kolejności sprawdzimy spełnienie założenia o normalności rozkładu badanych cech.

```
shapiro.test(rentownosc$val)
```

Efektem działania jest wydruk:

```
Shapiro-Wilk normality test
data:  rentownosc$val
W = 0.98011, p-value = 0.5834
```

Nie ma podstaw do odrzucenia hipotezy zerowej – wartości cechy mają rozkład normalny. W kolejnym kroku testujemy równość wariancji. Formalnie sprawdzamy to, wykorzystując test Bartletta.

```
bartlett.test(val~pkd, data=rentownosc)
```

Podsumowanie testu Bartletta jest następujące:

```
Bartlett test of homogeneity of variances
data:  val by pkd
Bartlett's K-squared = 8.9975, df = 2, p-value = 0.01112
```

Uzyskana wartość *p* jest większa od poziomu istotności 0,01, zatem stwierdzamy, że wariancje analizowanych cech nie różnią się istotnie między sobą. Spełnienie założeń o rozkładzie normalnym oraz równości wariancji umożliwia skorzystanie z metody ANOVA. W tym celu należy wywołać funkcję `aov()`. Funkcja ta w przeciwieństwie do dotychczas przedstawionych nie zwraca bezpośrednio wartości *p*. Wynik działania funkcji należy przypisać do obiektu, a następnie na tym obiekcie wywołać funkcję `summary()`.

```
rentownosc_anova<-aov(val~pkd, data=rentownosc)
summary(rentownosc_anova)
```

W rezultacie uruchomienia uzyskuje się:

```
             Df Sum Sq Mean Sq F value   Pr(>F)
pkd           2  66.83   33.41   19.84 6.65e-07 ***
Residuals    45  75.80    1.68
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Wartość *p* zaprezentowana w kolumnie *Pr* (>*F*) jest bardzo mała, zatem można stwierdzić, że średnie wskaźniki rentowności dla co najmniej jednej pary sekcji PKD różnią istotnie pomiędzy sobą. W celu sprawdzenia, które to pary, można wykorzystać jeden z testów *post-hoc*. W przypadku analizy ANOVA jest to test Tukeya, który można uruchomić funkcją `TukeyHSD()` podając jako argument wynik działania funkcji `aov()`.

```
TukeyHSD(rentownosc_anova)
```

Wynik testu Tukeya jest następujący:

```
  Tukey multiple comparisons of means
    95% family-wise confidence level
Fit: aov(formula = val ~ pkd, data = rentownosc)
$pkd
                          diff        lwr        upr     p adj
handel-budownictwo   -1.24375 -2.3558345 -0.1316655 0.0251867
przemysł-budownictwo  1.63750  0.5254155  2.7495845 0.0024511
przemysł-handel       2.88125  1.7691655  3.9933345 0.0000004
```

W wyniku działania funkcji otrzymujemy tabelę ze wszystkimi możliwymi parami zmiennych, obliczoną różnicę średnich oraz, najistotniejszą z naszego punktu widzenia, wartość p (kolumna `p adj`). W tym przypadku wartości p nie przekraczają poziomu 0,01 dla sekcji przemysł i budownictwo oraz przemysł i handel, co oznacza, że średnie wskaźniki rentowności różnią się między sobą dla tych sekcji.

Test wartości przeciętnej dla siedmiu prób niezależnych – wskaźnik wykrywalności przestępstw kryminalnych w regionach

Na podstawie danych o wskaźniku wykrywalności przestępstw kryminalnych na poziomie powiatów sprawdzimy, czy przeciętne wartości tego wskaźnika różnią się pomiędzy regionami. Przyjmujemy poziom istotności $\alpha = 0{,}01$, a hipotezy są następujące:

$H_0: m_1 = m_2 = \ldots = m_7$ – wartości przeciętne wskaźnika wykrywalności przestępstw kryminalnych nie różnią się istotnie pomiędzy regionami,
$H_1: \exists_{i,j \in \{1,2,\ldots,7\}} \; m_i \neq m_j$ – istnieje co najmniej jedna para regionów, dla których wartości przeciętne wskaźnika wykrywalności przestępstw kryminalnych różnią się istotnie.

Najpierw pobieramy dane o wskaźniku wykrywalności na poziomie powiatów oraz dołączamy dane o regionach. Następnie sprawdzamy normalność rozkładu.

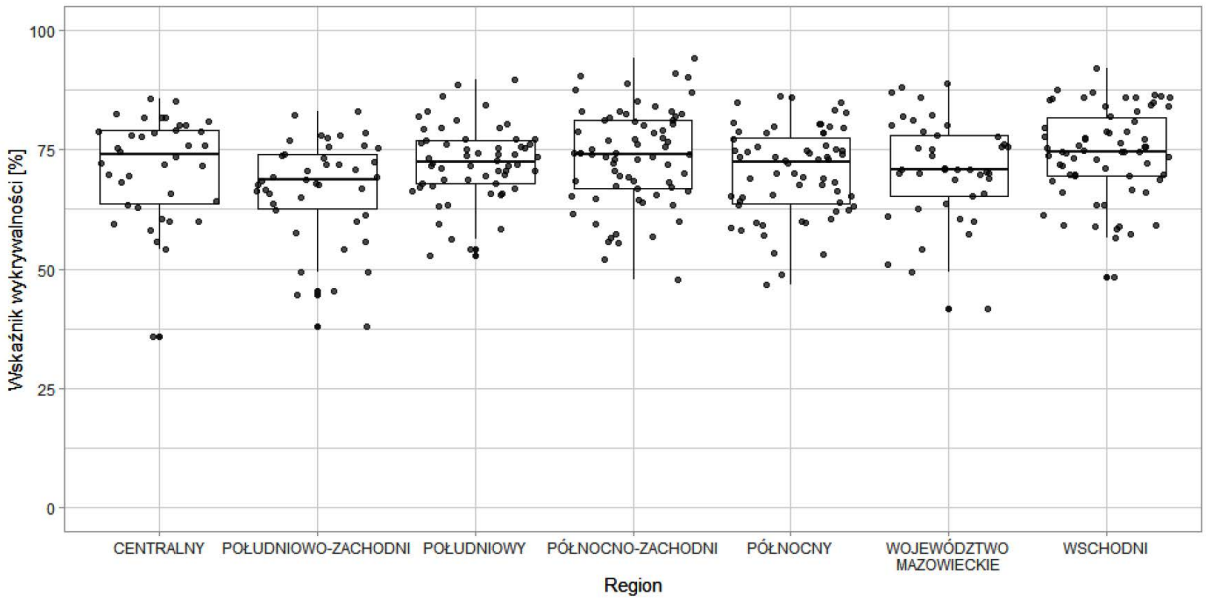
```
krym_powiat<-get_data_by_variable(varId=498634, unitLevel=5, year=2018)%>%
  mutate(region=substr(id,1,2), region=str_pad(region,12,"right",0))
krym_region<-get_data_by_variable(varId=498634, unitLevel=1, year=2018)%>%
  select(region=id, name_region=name) %>%
  mutate(name_region=gsub("MAKROREGION ", "", name_region),
    name_region=gsub(" ", "\n", name_region))
krym<-left_join(krym_powiat, krym_region)
shapiro.test(krym$val)
```

W efekcie wykonania poleceń został stworzony zbiór `krym`, który znajduje się w pamięci, natomiast w konsoli pojawił się wynik testu normalności.

```
Shapiro-Wilk normality test
data:  krym$val
W = 0.98079, p-value = 6.044e-05
```

Badanie normalności z wykorzystaniem testu Shapiro-Wilka wskazuje odstępstwo od rozkładu normalnego. W tej sytuacji nie jest możliwe zastosowanie metody ANOVA, tylko należy skorzystać z jej nieparametrycznej wersji, czyli testu Kruskala-Wallisa. Na rysunku 5.9 przedstawiono wykresy pudełkowe dla poszczególnych regionów.

```
ggplot(krym, aes(x=name_region, y=val)) +
  geom_boxplot() +
  geom_jitter(alpha=0.5) +
  ylab("Wskaźnik wykrywalności [%]") +
  xlab("Region") +
  ylim(0,100) +
  theme_light()
```



Rys. 5.9. Wskaźnik wykrywalności przestępstw kryminalnych w powiatach oraz miary pozycyjne w ramach regionów

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Na podstawie tego wykresu możemy zaobserwować, że mediany wskaźnika wykrywalności w różnych regionach są do siebie zbliżone i oscylują wokół wartości 75%. Za pomocą testu Kruskala-Wallisa sprawdzimy istotność różnic pomiędzy regionami.

```
kruskal.test(val~region, data=krym)
```

Wynikiem działania jest wydruk:

```
Kruskal-Wallis rank sum test
data:  val by region
Kruskal-Wallis chi-squared = 15.654, df = 6, p-value = 0.01574
```

Wartość p jest większa od przyjętego poziomu istotności $\alpha = 0{,}01$, zatem można stwierdzić, że wskaźniki wykrywalności przestępstw kryminalnych w analizowanych regionach nie różnią się istotnie statystycznie.

Literatura

Aczel, A. D. i Sounderpandian, J. (2017). *Statystyka w zarządzaniu*. Wydawnictwo Naukowe PWN.

Biecek, P. (2019). *Analiza danych z programem R. Modele liniowe z efektami stałymi, losowymi i mieszanymi*. Wydawnictwo Naukowe PWN.

Boslaugh, S. (2012). *Statistics in a Nutshell: A Desktop Quick Reference*. O'Reilly Media.

Kanji, G. K. (2006). *100 Statistical Tests*. Sage.

Kopczewska, K., Kopczewski, T. i Wójcik, P. (2016). *Metody ilościowe w R. Aplikacje ekonomiczne i finansowe*. CEDEWU.PL.

Paradysz, J. (red.). (2005). *Statystyka*. Wydawnictwo Akademii Ekonomicznej w Poznaniu.

R Core Team. (2023). R: *A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Shipunov, A. (2019). *Visual Statistics. Use R*. **https://mirror.dogado.de/cran/doc/contrib/Shipunov-visual_statistics.pdf**

Sobczyk, M. (2010). *Statystyka matematyczna*. Wydawnictwo C.H. Beck.

Sobczyk, M. (2019). *Statystyka*. Wydawnictwo Naukowe PWN.

Sokołowski, A. (2004). *O niewłaściwym stosowaniu metod statystycznych. Materiały szkoleniowe*. StatSoft Polska. **https://media.statsoft.pl/_old_dnn/downloads/naukowe1.pdf**

Szpadel M. i Kania K. (2023). *bdl: Interface and Tools for 'BDL' API*. R package version 1.0.5. **https://CRAN.R-project.org/package=bdl**

Szreder, M. (2019). Istotność statystyczna w czasach big data. *Wiadomości Statystyczne. The Polish Statistician*, *64*(11), 42-57. **https://doi.org/10.5604/01.3001.0013.7583**

Walesiak, M. (1996). *Metody analizy danych marketingowych*. Wydawnictwo Naukowe PWN.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu i Yutani, H. (2019). Welcome to the Tidyverse. *Journal of Open Source Software*, *4*(43), artykuł 1686. **https://doi.org/10.21105/joss.01686**

DOI: 10.15611/2024.45.1.06

6 Imputacja brakujących danych

6.1. Zagadnienie brakujących danych

W badaniach statystycznych bardzo ważne są jakość i dokładność danych, ponieważ te charakterystyki wpływają w znacznym stopniu na wyniki badań. Balicki (2004) zwraca uwagę, że jakość danych jest pojęciem szerszym niż dokładność danych. Na jakość danych, poza dokładnością, wpływają także elementy wynikające z celu badania statystycznego. Kordos (1988, s. 13) wyróżnia trzy charakterystyki określające jakość danych statystycznych (*quality of statistical data*):

- przydatność danych (*data relevance*),
- aktualność danych (*data timeliness*),
- dokładność danych (*data accuracy*).

Dokładność danych wyszczególniona wśród tych charakterystyk jest ściśle związana z zagadnieniem brakujących danych. Miarą dokładności danych jest ich podobieństwo do wartości (danych) prawdziwych. Wartości (dane) prawdziwe to te, które można by było zgromadzić bez wprowadzania żadnych błędów, ale w praktyce błędy w danych się pojawiają z różnych powodów. W statystyce rozróżnia się błędy losowe (*sampling errors*) i nielosowe (*nonsampling errors*).

Błędy losowe (statystyczne) są związane z badaniami reprezentacyjnymi, opartymi na próbach losowych. Ich źródłem są różnice między wynikami otrzymanymi na podstawie próby losowej a wynikami, które można by było otrzymać, badając całą zbiorowość. Wielkość błędu losowego podlega pomiarowi. Szreder uważa, że określenie „błąd losowy" (sformułowanie przyjęte w polskojęzycznej literaturze statystycznej) nie jest w pełni adekwatne do charakteru tego błędu i proponuje stosować określenie „błąd losowania", które jest dokładnym tłumaczeniem określenia anglojęzycznego *sampling error*. Przyczyny błędu losowania nie są subiektywne i nie wynikają z zaniedbań w procedurze badania. Błąd losowania jest skutkiem stosowania niedoskonałych mechanizmów generowania próby losowej pochodzącej z populacji (Szreder, 2004, s. 158-161).

Dokładność danych zależy przede wszystkim od błędów nielosowych, które mogą wystąpić w badaniach zarówno wyczerpujących, jak i reprezentacyjnych, a więc w każdym badaniu statystycznym. Błędy nielosowe nie są nieuniknione i są głównym źródłem niekompletności danych (*incomplete data*) oraz braków danych (Balicki, 2004; Kordos, 1988; Szreder, 2004).

Rubin (1976) po raz pierwszy w sposób formalny przedstawił przyczyny i procesy powstawania braków danych.

Wyróżnia się trzy mechanizmy (procesy) generujące braki danych o różnym stopniu losowości (Balicki, 2004; Graham, 2012; Kozłowski i Szreder, 2020; Misztal, 2012a, 2012b; Pokropek, 2018; Rubin, 1976):

- mechanizmy w pełni losowe (MCAR – *Missing Completely At Random*) – brakujące obserwacje zmiennej *X* pojawiają się w zbiorze danych całkowicie losowo i nie zależy to ani od wartości tej zmiennej, ani od innych obserwacji w tym zbiorze (np. błędy powstające podczas wprowadzania danych);
- mechanizmy losowe (MAR – *Missing At Random*) – brakujące obserwacje zmiennej *X* pojawiają się w zbiorze danych losowo i nie zależy to od wartości tej zmiennej, ale od innych obserwacji w tym zbiorze (np. odmowa udzielenia odpowiedzi na pytanie w badaniu sondażowym z określonych powodów);
- mechanizmy nielosowe (MNAR – *Missing Not At Random*) – brakujące obserwacje zmiennej *X* pojawiają się w zbiorze danych nielosowo i nie zależy to od wartości tej zmiennej ani od wartości innych zmiennych obserwowanych, lecz od zmiennych ukrytych (nieobserwowalnych), których nie można zmierzyć (np. niechęć respondentów do udzielenia odpowiedzi wynikająca z nieznanych powodów).

6.2. Podejścia stosowane w sytuacji brakujących danych

Problemu brakujących danych jeszcze nie rozwiązano jednoznacznie. Proponuje się różne podejścia, ale problem jest na tyle złożony, że optymalne rozwiązanie jest w dużym stopniu zależne od konkretnego przypadku. W literaturze przedmiotu wymienia się m.in. następujące podejścia do problemu radzenia sobie z brakami danych (Balicki, 2004; Pokropek, 2018):

- uwzględnienie w badaniu tylko danych kompletnych, a więc tych obiektów, dla których obserwacje wszystkich zmiennych są dostępne (pomija się obiekty, dla których nie ma obserwacji poszczególnych zmiennych);
- uwzględnienie w badaniu wszystkich obiektów w zakresie dostępnych wartości zmiennych (liczby obserwacji poszczególnych zmiennych dla kolejnych obiektów mogą być w tym podejściu różne, co ma konsekwencje w sensie statystycznym, np. niektóre statystyki, takie jak średnia, wariancja, kowariancja, mogą być obliczane na podstawie różnej liczby obserwacji),
- imputacja brakujących danych (uzupełnianie brakujących lub niepoprawnych danych na podstawie relacji występujących w zbiorze zaobserwowanych danych).

6.3. Metody imputacji brakujących danych

Imputacja jest obecnie najczęściej stosowanym podejściem w sytuacji brakujących danych. Imputacja polega na uzupełnieniu brakującej lub zastąpieniu niepoprawnej obserwacji zmiennej wartością oszacowaną na podstawie analizy zależności (relacji) występujących dla zaobserwowanych wartości zmiennych w zbiorze danych. Dotychczas w literaturze przedmiotu zaproponowano wiele metod imputacji brakujących danych. Najogólniej metody imputacji dzieli się na dwie grupy (Balicki, 2004; Laaksonen, 2018; Misztal, 2012b):

- metody imputacji jednokrotnej (pojedynczej) (*single imputation*) – uzupełnianie brakującej obserwacji jest przeprowadzane tylko raz,
- metody imputacji wielokrotnej (*multiple imputation*) – uzupełnianie brakującej obserwacji jest przeprowadzane wiele razy, a wyniki są agregowane.

Wśród szczegółowych technik imputacji brakujących danych (uzupełnienia brakującej wartości lub zastąpienia niepoprawnej wartości) najczęściej są stosowane następujące (Balicki, 2004; Kalton i Kasprzyk, 1986; Misztal 2012b; Piasecki, 2014):

- zastąpienie brakującej obserwacji średnią arytmetyczną zmiennej (*mean imputation, mean substitution*),
- zastąpienie brakującej obserwacji medianą zmiennej (*median imputation, median substitution*),
- zastąpienie brakującej obserwacji dominantą zmiennej (*mode imputation, mode substitution*),
- zastąpienie brakującej obserwacji wartością losową (*random imputation, random substitution*),
- zastąpienie brakującej obserwacji wartością stałą spoza próby (*cold-deck imputation*),
- zastąpienie brakującej obserwacji wartością podobną z próby (*hot-deck imputation*),
- zastąpienie brakującej obserwacji wartością oszacowaną na podstawie modelu regresji (*regression imputation*),
- techniki wykorzystujące miary odległości i metody klasyfikacji (*distance function matching*, *nearest neighbour imputation*), np. imputacja metodą *k* najbliższych sąsiadów (*kNN imputation*),
- techniki modelowe wykorzystujące metodę największej wiarygodności, algorytmy iteracyjne (np. EM).

6.4. Imputacja brakujących danych z wykorzystaniem wybranych pakietów programu R

W programie R dostępne są funkcje umożliwiające na pierwszym etapie analizy brakujących danych ich identyfikację (występowanie brakujących danych, liczba brakujących danych, udział procentowy brakujących danych w ogólnej liczbie obserwacji) oraz podstawowe metody rozwiązania tego problemu (usunięcie wierszy lub kolumn z brakującymi danymi). W tabeli 6.1 zestawiono podstawowe funkcje przydatne na tym etapie analizy brakujących danych.

Tabela 6.1. Wybrane pakiety i funkcje programu R stosowane na etapie analizy brakujących danych

Pakiet	Funkcja	Opis funkcji
`base`	`is.na()`	funkcja zwraca wartość logiczną informującą o występowaniu (lub nie) brakujących danych
`base`	`any()`	funkcja zwraca prawdę, jeżeli co najmniej jedna wartość wektora jest prawdziwa
`stats`	`complete.cases()`	funkcja zwraca wektor wartości logicznych informujących o kompletnych rekordach
`stats`	`na.omit()`	funkcja usuwa rekordy z brakującymi danymi

Źródło: opracowanie własne na podstawie (R Core Team, 2023).

W wielu pakietach programu R oferowane są implementacje metod imputacji brakujących danych, metod imputacji zarówno jednokrotnej, jak i wielokrotnej. Obecnie jest to ponad 130 pakietów zgodnie z informacjami zamieszczonymi na stronie CRAN Task View: Missing Data (R Core Team, 2023). Zestawienie wybranych pakietów i funkcji, w szczególności wykorzystanych w zamieszczonych przykładach, zawiera tab. 6.2.

Tabela 6.2. Wybrane pakiety i funkcje programu R stosowane do imputacji brakujących danych

Nazwa pakietu	Wybrane charakterystyki	Wybrane funkcje
`mice` – Multivariate Imputation by Chained Equations (Van Buuren S. i in., 2023)	imputacja wielokrotna z wykorzystaniem równań łańcuchowych	`mice()`, `md.pattern()`, `stripplot()`
`Hmisc` – Harrell Miscellaneous (Harrell, 2023)	imputacja jednokrotna za pomocą średniej arytmetycznej, mediany, wartości losowej	`impute()`, `na.delete()`, `transcan()`
`hydroGOF` – Goodness-of-Fit Functions for Comparison of Simulated and Observed Hydrological Time Series (Zambrano-Bigiarini, 2020)	miary dopasowania wartości imputowanych do wartości obserwowanych	`nrmse()`, `rmse()`
`Amelia` – A Program for Missing Data (Honaker i in., 2022)	imputacja wielokrotna brakujących danych z wykorzystaniem algorytmu EM (boostrap)	`amelia()`, `missmap()`

Źródło: opracowanie własne na podstawie (R Core Team, 2023).

6.5. Zastosowania z wykorzystaniem programu R dla danych z BDL

Zbiór danych empirycznych bez brakujących danych

W przykładzie wykorzystano wybrane zmienne opisujące ochronę środowiska w przekroju województw w Polsce w 2018 r. Dane zostały pobrane z Banku Danych Lokalnych za pomocą interfejsu API dla programu R – pakiet `bdl` (Szpadel i Kania, 2023) i zapisane w pliku `dane_os0.csv` z wykorzystaniem skryptu 6.1. Wybrane zmienne opisujące stan i ochronę środowiska w przekroju województw w 2018 r. to:

x1 – osady wytworzone w ciągu roku (10067),
x2 – ładunki zanieczyszczeń w ściekach odprowadzonych do wód lub do ziemi (10077),
x3 – zużycie wody na potrzeby gospodarki narodowej i ludności w ciągu roku (10086),
x4 – przemysłowe i komunalne oczyszczalnie ścieków (10087),

Skrypt 6.1

```
library(bdl)
library(tidyverse)
options(OutDec=",")# stałe
rok<-2018
no_uL=2 # (0-Polska,1-makroregion,2-województwo,3-region,4-podregion,
        # 5-powiat,6-gmina,7-miejscowość)
```

```
# wczytanie informacji o zmiennych z BDL - funkcja get_data_by_variable {bdl}
d1<-get_data_by_variable("10067",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x1=val)
d2<-get_data_by_variable("10077",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x2=val)
d3<-get_data_by_variable("10086",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x3=val)
d4<-get_data_by_variable("10087",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x4=val)
# połączenie zmiennych w jedną tabelę (ramkę) danych – funkcja
# full_join {dplyr}, join {dplyr}
# forward-pipe operator %>% {magrittr}
  data<-d1 %>%
  full_join (d2,by=c("id","name")) %>%
  full_join (d3,by=c("id","name")) %>%
  full_join (d4,by=c("id","name"))
# usunięcie z nazwy obiektu wyrazu Województwo - funkcja mutate {dplyr}
data<-mutate(data,name=gsub("Województwo","",name))
x<-as.data.frame(data)
# nazwy kolumn
colnames(x)<-c("id","Województwo","x1","x2","x3","x4")
cat("zbiór danych ","\n")
print(dim(x))
print(x)
x<-x[,2:ncol(x)]
write.csv2(x,file="dane_os0.csv",row.names=FALSE)
```

Skrypt 6.1. umożliwia pobranie z BDL wartości wskazanych zmiennych oraz zapisanie ich w pliku `dane_os0.csv`. Skrypt wyświetla także rozmiary pliku `dane_os0.csv` oraz jego zawartość.

```
> cat("zbiór danych ","\n")
zbiór danych
> print(dim(x))
[1] 16  6
> print(x)
             id         Województwo    x1         x2       x3      x4
1  011200000000         MAŁOPOLSKIE 52811  235850064  94088,8 1763385
2  012400000000             ŚLĄSKIE 65889 1477805257 137477,6 3375871
3  020800000000            LUBUSKIE 18072    1651751  31279,0  560011
4  023000000000       WIELKOPOLSKIE 67709    6795783 137057,0 2048008
5  023200000000  ZACHODNIOPOMORSKIE 27213   50134502  58140,5 1203086
6  030200000000        DOLNOŚLĄSKIE 43435  453880388  98013,7 1887160
7  031600000000             OPOLSKIE 15024   13711198  31031,8  618909
8  040400000000  KUJAWSKO-POMORSKIE 26477 1207227928  73449,0 1150601
9  042200000000           POMORSKIE 37737   26078725  82039,6 1596170
10 042800000000 WARMIŃSKO-MAZURSKIE 19229    1353247  45912,8  840153
11 051000000000             ŁÓDZKIE 44164    1987641  92404,3 1472417
12 052600000000      ŚWIĘTOKRZYSKIE 14870   11414630  34006,3  609400
13 060600000000           LUBELSKIE 22645    8963162  61019,7  884321
14 061800000000        PODKARPACKIE 24687    1730623  49673,1 1041984
15 062000000000           PODLASKIE 13801     827191  41733,5  649462
16 071400000000         MAZOWIECKIE 89307   26432547 213431,8 3345685
```

Plik danych `dane_os0.csv` nie zawiera braków, ale umożliwia przedstawienie zastosowań funkcji wprowadzających braki danych (symulacyjne), identyfikację brakujących danych, usuwanie wierszy i kolumn z brakującymi danymi oraz porównanie wybranych metod imputacji brakujących danych (zob. skrypt 6.2).

Skrypt 6.2

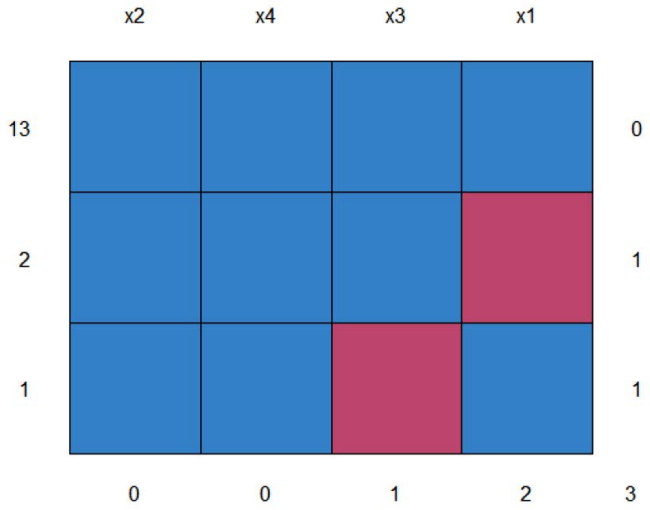
```
library(mice)
options("scipen"=100,"digits"=3)
options(OutDec=",")
dane<-read.csv2("dane_os0.csv",header=TRUE)
D<-dane[,2:ncol(dane)]
# D - macierz bez braków danych
cat("macierz bez brakujących danych ","\n")
print(dim(D))
print(D)
cat(" ","\n")
print(md.pattern(D,plot=FALSE))
# wprowadzenie brakujących danych (rekordów)
cat("wprowadzenie brakujących danych ","\n")
D1<-D
set.seed(123)
D1[sample(1:nrow(D), 2), "x1"] <- NA
D1[sample(1:nrow(D), 1), "x3"] <- NA
print(D1)
# identyfikacja brakujących danych (rekordów)
# funkcje - is.na {base}, complete.cases {stats}
# brakujące dane - liczba, udział procentowy, wzorzec
cat("czy są brakujące dane? ","\n")
print(any(is.na(D1)))
kr<-complete.cases(D1)
# liczba rekordów z kompletnymi danymi
cat("liczba rekordów z kompletnymi danymi ","\n")
print(sum(kr))
# liczba rekordów z brakującymi danymi
cat("liczba rekordów z brakującymi danymi ","\n")
print(sum(!kr))
# udział procentowy rekordów z brakującymi danymi
cat("udział procentowy rekordów z brakującymi danymi ","\n")
print(nrow(D1[!complete.cases(D1), ])/nrow(D1)*100)
# wzorzec braków danych - funkcja md.pattern {mice}
cat("wzorzec brakujących danych - ocena liczbowa ","\n")
print(md.pattern(D1,plot=FALSE))
cat("wzorzec brakujących danych - ocena graficzna i liczbowa ","\n")
print(md.pattern(D1,plot=TRUE))
write.csv2(D1,file="dane_os1.csv",row.names=FALSE)
```

W wyniku wykonania skryptu 6.2 wprowadzone zostały braki danych i zapisany został plik `dane_os1.csv` zawierający brakujące dane (5 rekordów, 3 zmienne, z brakami danych, tj. 16,7%). Funkcja `md.pattern()` z pakietu `mice` (Van Buuren i Groothuis-Oudshoorn, 2011) umożliwia ocenę liczbową i graficzną rozkładu brakujących danych (wzorzec braków danych). Jeżeli argument tej funkcji `plot=FALSE`, to prezentowana jest ocena liczbowa (częstości brakujących danych), a jeżeli

plot=TRUE, to prezentowana jest zarówno ocena liczbowa, jak i graficzna wzorca brakujących danych. Z rysunku 6.1 otrzymanego za pomocą skryptu 6.2 można odczytać, że 13 wierszy (rekordów) nie zawiera brakujących danych, zmienne x2 i x4 nie zawierają brakujących wartości, zmienna x1 zawiera dwie brakujące wartości, a zmienna x3 zawiera jedną brakującą wartość.

```
macierz bez brakujących danych
[1] 16  4
      x1         x2     x3      x4
1  52811  235850064  94089 1763385
2  65889 1477805257 137478 3375871
3  18072    1651751  31279  560011
4  67709    6795783 137057 2048008
5  27213   50134502  58141 1203086
6  43435  453880388  98014 1887160
7  15024   13711198  31032  618909
8  26477 1207227928  73449 1150601
9  37737   26078725  82040 1596170
10 19229    1353247  45913  840153
11 44164    1987641  92404 1472417
12 14870   11414630  34006  609400
13 22645    8963162  61020  884321
14 24687    1730623  49673 1041984
15 13801     827191  41734  649462
16 89307   26432547 213432 3345685

 /\     /\
{  `---'  }
{  O   O  }
==>  V <==  No need for mice. This data set is completely observed.
 \  \|/  /
  `-----'

   x1 x2 x3 x4
16  1  1  1  1 0
    0  0  0  0 0
wprowadzenie brakujących danych
      x1         x2     x3      x4
1  52811  235850064  94089 1763385
2  65889 1477805257 137478 3375871
3  18072    1651751     NA  560011
4  67709    6795783 137057 2048008
5  27213   50134502  58141 1203086
6  43435  453880388  98014 1887160
7  15024   13711198  31032  618909
8  26477 1207227928  73449 1150601
9  37737   26078725  82040 1596170
10 19229    1353247  45913  840153
11 44164    1987641  92404 1472417
12 14870   11414630  34006  609400
13 22645    8963162  61020  884321
14 24687    1730623  49673 1041984
15    NA     827191  41734  649462
16    NA   26432547 213432 3345685
czy są brakujące dane?
```

```
[1] TRUE
liczba rekordów z kompletnymi danymi
[1] 13
liczba rekordów z brakującymi danymi
[1] 3
udział procentowy rekordów z brakującymi danymi
[1] 18,8
wzorzec brakujących danych - ocena liczbowa
   x2 x4 x3 x1
13  1  1  1  1 0
2   1  1  1  0 1
1   1  1  0  1 1
    0  0  1  2 3
wzorzec brakujących danych - ocena graficzna i liczbowa
   x2 x4 x3 x1
13  1  1  1  1 0
2   1  1  1  0 1
1   1  1  0  1 1
    0  0  1  2 3
>
```



Rys. 6.1. Graficzna ocena brakujących danych (kolor niebieski – wartości obserwowane, kolor czerwony – wartości brakujące)

Źródło: opracowanie własne z wykorzystaniem funkcji `md.pattern()` z pakietu `mice` (R Core Team, 2023).

Podstawowe podejścia stosowane w przypadku brakujących danych to uwzględnienie tylko danych kompletnych, a więc usunięcie rekordów lub zmiennych zawierających braki danych (są to jednak podejścia powodujące znaczne straty informacji). Na przykładzie pliku zapisanego z wprowadzonymi brakami danych `dane_os1.csv` usunięto wiersze i kolumny zawierające braki danych (skrypt 6.3).

Skrypt 6.3

```
D<-read.csv2("dane_os1.csv",header=TRUE)
cat("zbiór danych z brakami ","\n")
print(D)
```

```
print(dim(D))
# usunięcie brakujących danych (wierszy, rekordów, obiektów)
cat("usunięcie brakujących danych - wierszy ","\n")
any(is.na(D))
dim(D)
D1<-na.omit(D)
any(is.na(D1))
print(dim(D1))
print(D1)
# usunięcie brakujących danych (kolumn, zmiennych)
cat("usunięcie brakujących danych - kolumn ","\n")
any(is.na(D))
dim(D)
D2<-D[,-which(colMeans(is.na(D))>0.01)]
any(is.na(D2))
print(dim(D2))
print(D2)
```

Wykonanie skryptu 6.3 powoduje usunięcie wierszy (rekordów) lub kolumn (zmiennych) z pliku `dane_os1.csv`.

```
zbiór danych z brakami
      x1         x2     x3      x4
1  52811  235850064  94089 1763385
2  65889 1477805257 137478 3375871
3  18072    1651751     NA  560011
4  67709    6795783 137057 2048008
5  27213   50134502  58141 1203086
6  43435  453880388  98014 1887160
7  15024   13711198  31032  618909
8  26477 1207227928  73449 1150601
9  37737   26078725  82040 1596170
10 19229    1353247  45913  840153
11 44164    1987641  92404 1472417
12 14870   11414630  34006  609400
13 22645    8963162  61020  884321
14 24687    1730623  49673 1041984
15    NA     827191  41734  649462
16    NA   26432547 213432 3345685
[1] 16  4
usunięcie brakujących danych - wierszy
[1] 13  4
      x1         x2     x3      x4
1  52811  235850064  94089 1763385
2  65889 1477805257 137478 3375871
4  67709    6795783 137057 2048008
5  27213   50134502  58141 1203086
6  43435  453880388  98014 1887160
7  15024   13711198  31032  618909
8  26477 1207227928  73449 1150601
9  37737   26078725  82040 1596170
10 19229    1353247  45913  840153
11 44164    1987641  92404 1472417
12 14870   11414630  34006  609400
```

```
13 22645    8963162  61020  884321
14 24687    1730623  49673 1041984
usunięcie brakujących danych - kolumn
[1] 16  2
           x2      x4
1   235850064 1763385
2  1477805257 3375871
3     1651751  560011
4     6795783 2048008
5    50134502 1203086
6   453880388 1887160
7    13711198  618909
8  1207227928 1150601
9    26078725 1596170
10    1353247  840153
11    1987641 1472417
12   11414630  609400
13    8963162  884321
14    1730623 1041984
15     827191  649462
16   26432547 3345685
>
```

W badaniach empirycznych stosowane są bardziej zaawansowane podejścia w przypadku brakujących danych, minimalizujące straty informacji. W tych podejściach wykorzystuje się metody imputacji danych, które nie usuwają obiektów lub zmiennych z brakami danych, ale uzupełniają te braki z wykorzystaniem prostych i złożonych metod statystycznych.

Skrypt 6.4 umożliwia imputację brakujących danych w pliku `dane_os1.csv` za pomocą średniej arytmetycznej, mediany, wartości losowej oraz metody równań łańcuchowych.

Imputacja wielowymiarowa za pomocą równań łańcuchowych (MICE – *Multivariate Imputation using Chained Equations*) jest algorytmem iteracyjnym zbudowanym w oparciu o łańcuch (sekwencję) równań regresji, w którym każde ogniwo dotyczy kolejnej zmiennej. Algorytm jest przedstawiony w pracy Van Buurena (2018). Algorytm MICE przewiduje wartości brakujące (szacuje ich prawdopodobieństwa) na podstawie wartości obserwowanych wielu zmiennych wykorzystując różne modele regresji. Metoda ta jest nazywana także regresją sekwencyjną lub próbkowaniem Gibbsa.

Skrypt 6.4

```
library(mice)
library(Hmisc)
library(hydroGOF)
# plik bez braków danych
dane<-read.csv2("dane_os0.csv",header=TRUE)
print(dane)
D0<-dane[,2:ncol(dane)]
# plik z brakami danych
D<-read.csv2("dane_os1.csv",header=TRUE)
cat("plik z brakami danych ","\n")
print(D)
# imputacja brakujących wartości - funkcja impute {Hmisc}
```

6. Imputacja brakujących danych

```
# imputacja za pomocą średniej arytmetycznej
any(is.na(D))
D1<-D
D1$x1<-impute(D$x1,mean)
D1$x3<-impute(D$x3,mean)
any(is.na(D1))
# imputacja za pomocą mediany
any(is.na(D))
D2<-D
D2$x1<-impute(D$x1,median)
D2$x3<-impute(D$x3,median)
any(is.na(D2))
# imputacja za pomocą wartości losowej
any(is.na(D))
D3<-D
D3$x1<-impute(D$x1,"random")
D3$x3<-impute(D$x3,"random")
any(is.na(D3))
# imputacja wielowymiarowa za pomocą równań łańcuchowych
# (multivariate imputation by chained equations) - funkcja mice {mice}
# metoda imputacji - cart (classification and regression trees)
D4<-D
D4<-mice(D,m=5,method='cart',seed=1010,printFlag=F)
# graficzna ocena jakości imputacji - funkcja stripplot {mice}
print(stripplot(D4,pch=c(21,20.5),cex=c(1,1.5)))
D4<-complete(D4)
# porównanie wyników imputacji - funkcja nrmse {hydroGOF}
# nrmse - normalized root mean squared error
cat("porównanie wyników imputacji (funkcja nrmse {hydroGOF}): ","\n")
cat("średnia arytmetyczna, mediana, wartości losowe,
  równania łańcuchowe","\n")
cat("mean ","\n")
print(nrmse(D1,D0))
cat("median ","\n")
print(nrmse(D2,D0))
cat("random ","\n")
print(nrmse(D3,D0))
cat("mice-cart ","\n")
print(nrmse(D4,D0))
```

W wyniku wykonania skryptu 6.4 otrzymano cztery warianty imputacji brakujących danych. Funkcja `nrmse()` z pakietu `hydroGOF` umożliwia porównanie jakości dopasowania wartości imputowanych do wartości obserwowanych (mniejsze wartości wskazują na lepsze dopasowanie brakujących danych za pomocą wartości imputowanych).

Funkcja `nrmse` z pakietu `hydroGOF` oblicza znormalizowany średni błąd kwadratowy (*nrmse*) między wartościami symulowanymi (sim, S_i) i obserwowanymi (obs, O_i). Wynik podany jest w procentach (%).

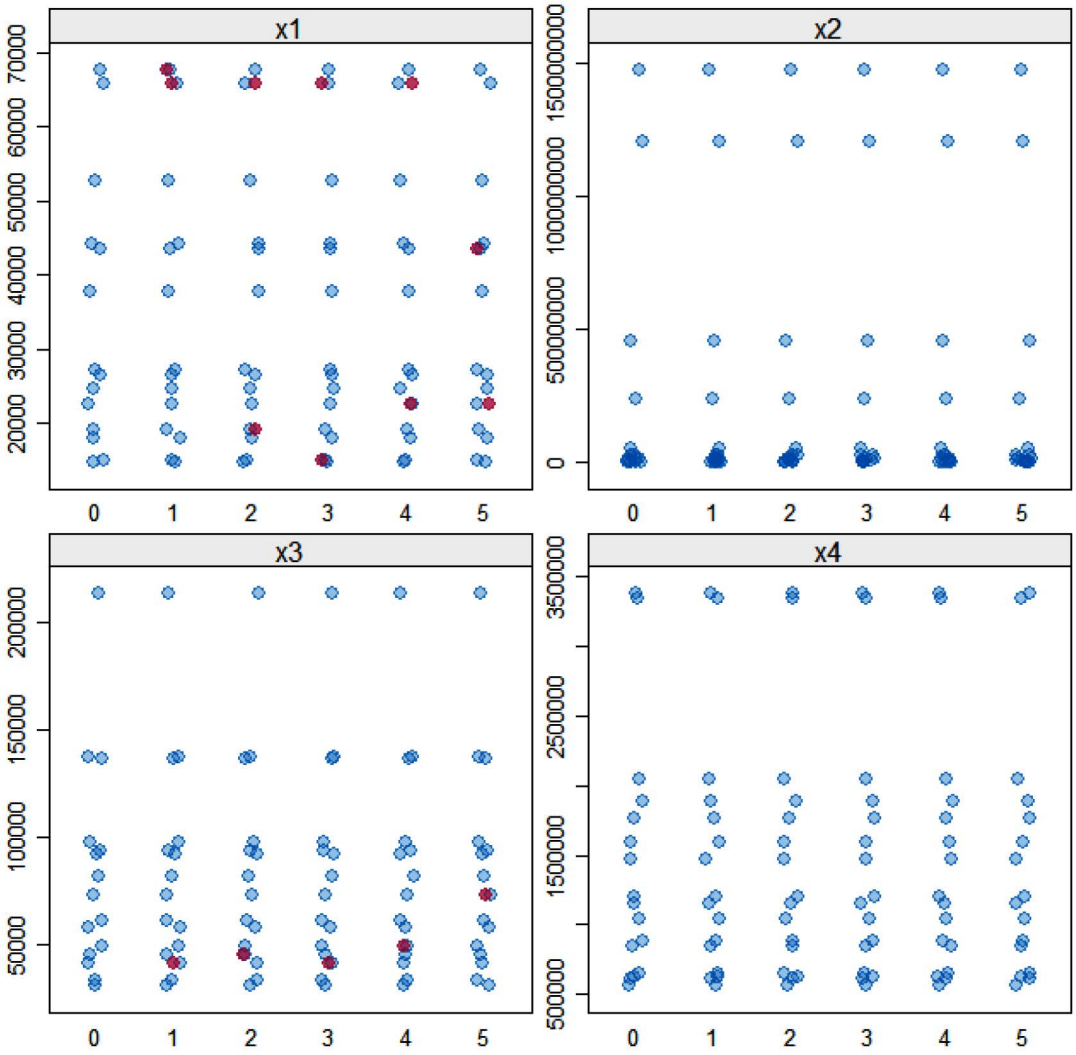
$$\text{nrmse} = 100 \cdot \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(S_i - O_i)^2}}{nval},$$

$$\text{nval} = \begin{cases} sd(O_i) & ,\text{norm} = \text{"sd"} \\ O_{\max} - O_{\min} & ,\text{norm} = \text{"maxmin"}, \end{cases}$$

gdzie norm wskazuje wartość (nval), która ma być stosowana do normalizacji błędu średniokwadratowego (RMSE). Może to być odchylenie standardowe obserwacji (wartość domyślna) lub różnica między zaobserwowanymi wartościami maksymalną a minimalną.

```
           Województwo    x1         x2     x3      x4
1          MAŁOPOLSKIE 52811  235850064  94089 1763385
2              ŚLĄSKIE 65889 1477805257 137478 3375871
3             LUBUSKIE 18072    1651751  31279  560011
4        WIELKOPOLSKIE 67709    6795783 137057 2048008
5   ZACHODNIOPOMORSKIE 27213   50134502  58141 1203086
6         DOLNOŚLĄSKIE 43435  453880388  98014 1887160
7             OPOLSKIE 15024   13711198  31032  618909
8   KUJAWSKO-POMORSKIE 26477 1207227928  73449 1150601
9            POMORSKIE 37737   26078725  82040 1596170
10 WARMIŃSKO-MAZURSKIE 19229    1353247  45913  840153
11             ŁÓDZKIE 44164    1987641  92404 1472417
12      ŚWIĘTOKRZYSKIE 14870   11414630  34006  609400
13           LUBELSKIE 22645    8963162  61020  884321
14        PODKARPACKIE 24687    1730623  49673 1041984
15           PODLASKIE 13801     827191  41734  649462
16         MAZOWIECKIE 89307   26432547 213432 3345685
plik z brakami danych
      x1         x2     x3      x4
1  52811  235850064  94089 1763385
2  65889 1477805257 137478 3375871
3  18072    1651751     NA  560011
4  67709    6795783 137057 2048008
5  27213   50134502  58141 1203086
6  43435  453880388  98014 1887160
7  15024   13711198  31032  618909
8  26477 1207227928  73449 1150601
9  37737   26078725  82040 1596170
10 19229    1353247  45913  840153
11 44164    1987641  92404 1472417
12 14870   11414630  34006  609400
13 22645    8963162  61020  884321
14 24687    1730623  49673 1041984
15    NA     827191  41734  649462
16    NA   26432547 213432 3345685
porównanie wyników imputacji (funkcja nrmse {hydroGOF}):
średnia arytmetyczna, mediana, wartości losowe, równania łańcuchowe
mean
  x1   x2   x3   x4
65,2  0,0 26,4  0,0
median
  x1   x2   x3   x4
70,9  0,0 21,4  0,0
random
  x1   x2   x3   x4
59,3  0,0 33,9  0,0
mice-cart
  x1   x2   x3   x4
65,3  0,0  5,3  0,0
```

Graficzną ocenę jakości imputacji metodą równań łańcuchowych otrzymaną za pomocą funkcji `stripplot()` z pakietu `mice` przedstawia rys. 6.2.



Rys. 6.2. Graficzna ocena jakości imputacji metodą równań łańcuchowych – plik `dane_os1.csv` (kolor niebieski – wartości obserwowane, kolor czerwony – wartości imputowane)

Źródło: opracowanie własne z wykorzystaniem funkcji `stripplot()` z pakietu `mice` (R Core Team, 2023).

Funkcja `stripplot()` służy do wizualizacji wartości obserwowanych i imputowanych za pomocą funkcji `mice()`. Wykres przedstawia osobno każdą zmienną i wyniki kolejnych imputacji wielokrotnych (domyślna liczba imputacji wielokrotnych wynosi 5). Kolor niebieski oznacza wartości obserwowane, a czerwony wartości imputowane.

Zbiór danych empirycznych z brakującymi danymi

W przykładzie wykorzystano wybrane zmienne opisujące ochronę społeczną w przekroju powiatów w Polsce w 2017 r. Dane zostały pobrane z Banku Danych Lokalnych za pomocą interfejsu API dla programu R – pakiet `bdl` (Szpadel i Kania, 2023) i zapisane w plikach `dane_oz0.csv` (braki danych wskazywane znacznikiem 0 – aktualne znaczniki w BDL) i `dane_oz1.csv` (braki danych wskazywane znacznikiem NA) z wykorzystaniem skryptu 6.5.

Skrypt 6.5

```
library(bdl)
library(tidyverse)
options(OutDec=",")
# stałe
rok<-2017
no_uL=5 #(0-Polska,1-makroregion,2-województwo,3-region,4-podregion,
        # 5-powiat,6–gmina,7-miejscowość)
# wczytanie informacji o zmiennych z BDL - funkcja get_data_by_variable {bdl}
d1<-get_data_by_variable("74066",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x1=val)
d2<-get_data_by_variable("410641",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x2=val)
d3<-get_data_by_variable("410728",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x3=val)
d4<-get_data_by_variable("10412",unitParentId=NULL,unitLevel=no_uL,year=rok)
%>% select(id,name, x4=val)
# połączenie zmiennych w jedną tabelę (ramkę) danych – funkcja
# full_join {dplyr}, join {dplyr}
# forward-pipe operator %>% {magrittr}
data<-d1 %>%
  full_join (d2,by=c("id","name")) %>%
  full_join (d3,by=c("id","name")) %>%
  full_join (d4,by=c("id","name"))
# usunięcie z nazwy obiektu wyrazu Powiat - funkcja mutate {dplyr}
data<-mutate(data,name=gsub("Powiat ","",name))
x<-as.data.frame(data)
# nazwy kolumn
colnames(x)<-c("id","powiat","x1","x2","x3","x4")
print(dim(x))
# indeksy obserwacji z "0"
i0<-which(x==0, arr.ind=TRUE)
i0<-as.vector(i0[,1])
cat("fragment zbioru danych z 0","\n")
print(x[i0[1:15],], row.names=FALSE)
write.csv2(x,file="dane_oz0.csv",row.names=FALSE)
# zmiana znacznika braku danych z "0" na "NA"
if(!any(is.na(x))) {x[x==0]<-NA}
cat("fragment zbioru danych z NA","\n")
print(x[i0[1:15],], row.names=FALSE)
write.csv2(x,file="dane_oz1.csv",row.names=FALSE)
```

Plik `dane_oz0.csv` zawiera brakujące dane oznaczone znacznikiem 0. Struktura pliku obejmuje 380 obiektów (powiaty w Polsce) i 4 zmienne:

x1 – lekarze dentyści (osoby) (74066),
x2 – odsetek dzieci objętych opieką w żłobkach (410641),
x3 – uczestnicy imprez kulturalnych (410728),
x4 – miejsca w placówkach stacjonarnej pomocy społecznej (10412).

```
> cat("fragment zbioru danych z 0","\n")
fragment zbioru danych z 0
```

```
> print(x[i0[1:15],], row.names=FALSE)
           id        powiat x1  x2    x3  x4
 023015904000     gostyński  0 3,6 12470 406
 023015913000   leszczyński  0 0,0 15400  30
 023015914000 międzychodzki  0 7,3 31300 349
 023015915000   nowotomyski  0 8,0  9119  52
 023016126000       śremski  0 8,6  6500 358
 023015913000   leszczyński  0 0,0 15400  30
 023216416000    świdwiński 19 0,0     0 174
 023216605000      gryficki 21 0,0  1000 504
 040410712000      rypiński 15 0,0     0  50
 040410717000     wąbrzeski 11 0,0     0  75
 040410811000  radziejowski 10 0,0  6410  70
 042815404000      elbląski 14 0,0  9900 229
 042815519000   węgorzewski  7 0,0  5600 272
 042815601000   bartoszycki 25 0,0  4200 306
 051011521000    brzeziński 13 0,0  1000 192
```

Plik `dane_oz1.csv` zawiera brakujące dane oznaczone znacznikiem NA (skrypt 6.5 przeprowadza zamianę 0 na NA). Obecnie w bazach BDL braki danych są kodowane za pomocą symbolu `0`. W programie R i w wielu bazach danych brakujące wartości (numeryczne i znakowe) są oznaczone symbolem `NA` (*not available*). Funkcje programu R przeznaczone do pracy z brakami danych odwołują się do tego symbolu (znacznika). W związku z tym w przykładach zamieszczonych w pracy znacznik `0` jest zamieniany na znacznik `NA`.

```
> cat("fragment zbioru danych z NA","\n")
fragment zbioru danych z NA
> print(x[i0[1:15],], row.names=FALSE)
           id        powiat x1  x2    x3  x4
 023015904000     gostyński NA 3,6 12470 406
 023015913000   leszczyński NA  NA 15400  30
 023015914000 międzychodzki NA 7,3 31300 349
 023015915000   nowotomyski NA 8,0  9119  52
 023016126000       śremski NA 8,6  6500 358
 023015913000   leszczyński NA  NA 15400  30
 023216416000    świdwiński 19  NA    NA 174
 023216605000      gryficki 21  NA  1000 504
 040410712000      rypiński 15  NA    NA  50
 040410717000     wąbrzeski 11  NA    NA  75
 040410811000  radziejowski 10  NA  6410  70
 042815404000      elbląski 14  NA  9900 229
 042815519000   węgorzewski  7  NA  5600 272
 042815601000   bartoszycki 25  NA  4200 306
 051011521000    brzeziński 13  NA  1000 192
```

Skrypt 6.6 umożliwia imputację brakujących danych w pliku `dane_oz1.csv` za pomocą średniej arytmetycznej, mediany oraz metody równań łańcuchowych.

Skrypt 6.6

```
source("readkeygraph.r")
library(mice)
library(Hmisc)
```

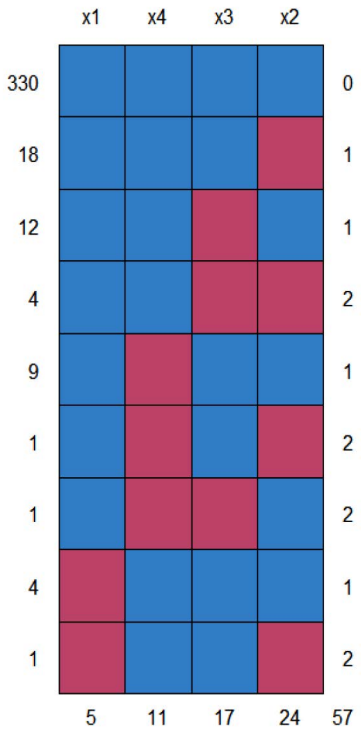
```
# dane z BDL - plik z brakami danych
dane<-read.csv2("dane_oz1.csv",header=TRUE)
cat("macierz danych ","\n")
print(dim(dane))
# indeksy obserwacji z brakami danych (NA)
ina<-which(is.na(dane), arr.ind=TRUE)
ina<-as.vector(ina[,1])
print(dane[ina[1:15],], row.names=FALSE)
D<-dane[,3:ncol(dane)]
# rekordy kompletne i z brakami danych
cat("czy są brakujące dane? ","\n")
print(any(is.na(D)))
kr<-complete.cases(D)
# liczba rekordów z kompletnymi danymi
cat("liczba rekordów z kompletnymi danymi ","\n")
print(sum(kr))
# liczba rekordów z brakującymi danymi
cat("liczba rekordów z brakującymi danymi ","\n")
print(sum(!kr))
# udział procentowy rekordów z brakującymi danymi
cat("udział procentowy rekordów z brakującymi danymi ","\n")
print(nrow(D[!complete.cases(D), ])/nrow(D)*100)
# wzorzec braków danych - funkcja md.pattern {mice}
cat("wzorzec brakujących danych - ocena liczbowa ","\n")
print(md.pattern(D))
keyPressed<-readkeygraph("[press any key to continue]")
cat("wzorzec brakujących danych - ocena graficzna ","\n")
# imputacja za pomocą średniej arytmetycznej - funkcja impute {Hmisc}
any(is.na(D))
D1<-impute(D,median)
any(is.na(D1))
# imputacja za pomocą mediany - funkcja impute {Hmisc}
any(is.na(D))
D1<-impute(D,median)
any(is.na(D))
# imputacja metodą równań łańcuchowych (Chained Equations) - mice {mice}
# metoda imputacji - pmm - predictive mean matching
D2<-mice(D,m=7,method='pmm',seed=1010,printFlag=F)
# graficzna ocena jakości imputacji - funkcja stripplot {mice}
cat("graficzna ocena jakości imputacji metodą równań łańcuchowych ","\n")
print(stripplot(D2,pch=c(21,20.5),cex=c(1,1.5)))
keyPressed<-readkeygraph("[press any key to continue]")
cat("zbiór danych z wartościami imputowanymi metodą
  równań łańcuchowych ","\n")
D3<-complete(D2)
print(D3[ina[1:15],], row.names=FALSE)
```

Z wykorzystaniem funkcji `md.pattern()` z pakietu `mice` otrzymano ocenę liczbową i graficzną (rys. 6.3) rozkładu brakujących danych (wzorzec braków danych).

```
> cat("macierz danych ","\n")
macierz danych
> print(dim(dane))
```

```
[1] 380   6
> # indeksy obserwacji z brakami danych (NA)
> ina<-which(is.na(dane), arr.ind=TRUE)
> ina<-as.vector(ina[,1])
> print(dane[ina[1:15],], row.names=FALSE)
          id        powiat x1  x2    x3  x4
 23015904000     gostyński NA 3,6 12470 406
 23015913000   leszczyński NA  NA 15400  30
 23015914000 międzychodzki NA 7,3 31300 349
 23015915000   nowotomyski NA 8,0  9119  52
 23016126000       śremski NA 8,6  6500 358
 23015913000   leszczyński NA  NA 15400  30
 23216416000    świdwiński 19  NA    NA 174
 23216605000      gryficki 21   NA  1000 504
 40410712000      rypiński 15  NA    NA  50
 40410717000     wąbrzeski 11  NA    NA  75
 40410811000  radziejowski 10  NA  6410  70
 42815404000      elbląski 14  NA  9900 229
 42815519000   węgorzewski  7  NA  5600 272
 42815601000   bartoszycki 25  NA  4200 306
 51011521000    brzeziński 13  NA  1000 192
> D<-dane[,3:ncol(dane)]
> # rekordy kompletne i z brakami danych
> cat("czy są brakujące dane? ","\n")
czy są brakujące dane?
> print(any(is.na(D)))
[1] TRUE
> kr<-complete.cases(D)
> # liczba rekordów z kompletnymi danymi
> cat("liczba rekordów z kompletnymi danymi ","\n")
liczba rekordów z kompletnymi danymi
> print(sum(kr))
[1] 330
> # liczba rekordów z brakującymi danymi
> cat("liczba rekordów z brakującymi danymi ","\n")
liczba rekordów z brakującymi danymi
> print(sum(!kr))
[1] 50
> # udział procentowy rekordów z brakującymi danymi
> cat("udział procentowy rekordów z brakującymi danymi ","\n")
udział procentowy rekordów z brakującymi danymi
> print(nrow(D[!complete.cases(D), ])/nrow(D)*100)
[1] 13,15789
> # wzorzec braków danych - funkcja md.pattern {mice}
> cat("wzorzec brakujących danych - ocena liczbowa ","\n")
wzorzec brakujących danych - ocena liczbowa
> print(md.pattern(D))
    x1 x4 x3 x2
330  1  1  1  1  0
18   1  1  1  0  1
12   1  1  0  1  1
4    1  1  0  0  2
9    1  0  1  1  1
```

```
1    1  0  1  0  2
1    1  0  0  1  2
4    0  1  1  1  1
1    0  1  1  0  2
     5 11 17 24 57
[press any key to continue]
wzorzec brakujących danych - ocena graficzna
```

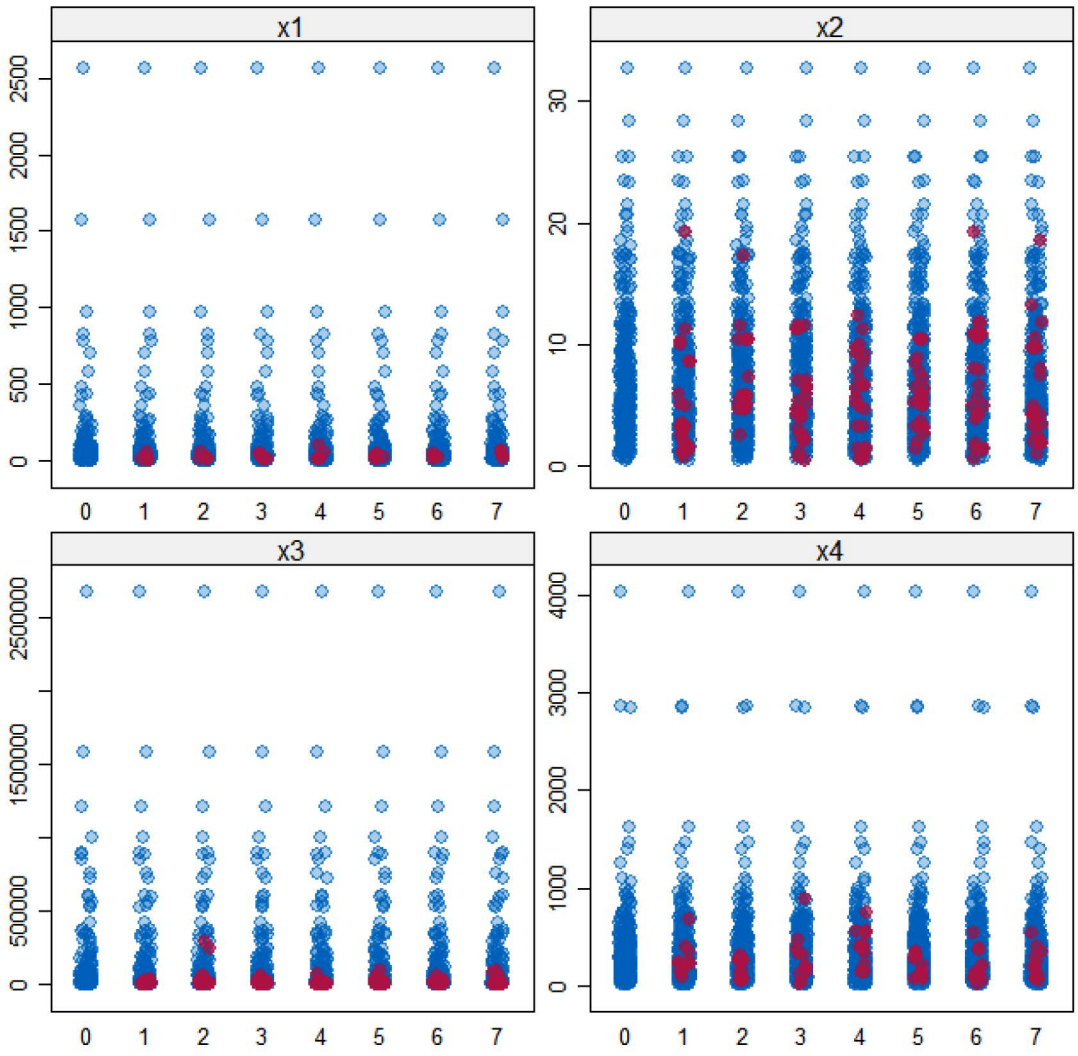


Rys. 6.3. Graficzna ocena brakujących danych (kolor niebieski – wartości obserwowane, kolor czerwony – wartości brakujące)

Źródło: opracowanie własne z wykorzystaniem funkcji `md.pattern()` z pakietu `mice` (R Core Team, 2023).

```
> cat("zbiór danych z wartościami imputowanymi metodą równań łańcuchowych
","\n")
zbiór danych z wartościami imputowanymi metodą równań łańcuchowych
> D3<-complete(D2)
> print(D3[ina[1:15],], row.names=FALSE)
 x1   x2    x3  x4
 51  3,6 12470 406
 22 10,0 15400  30
 14  7,3 31300 349
  7  8,0  9119  52
  3  8,6  6500 358
 22 10,0 15400  30
 19  1,8  7400 174
 21  5,1  1000 504
 15  4,9  1700  50
 11  1,2 11500  75
 10 10,3  6410  70
 14  3,5  9900 229
  7  2,6  5600 272
 25  8,6  4200 306
 13  3,0  1000 192
```

Graficzną ocenę jakości imputacji metodą równań łańcuchowych otrzymaną za pomocą funkcji `stripplot()` z pakietu `mice` przedstawia rys. 6.4.



Rys. 6.4. Graficzna ocena jakości imputacji metodą równań łańcuchowych – plik `dane_oz.csv` (kolor niebieski – wartości obserwowane, kolor czerwony – wartości imputowane)

Źródło: opracowanie własne z wykorzystaniem funkcji `stripplot()` z pakietu `mice` (R Core Team, 2023).

Problem brakujących danych dotyczy źródeł zarówno wtórnych, jak i pierwotnych. Usuwanie rekordów lub zmiennych wiąże się z niebezpieczeństwem utraty dużej liczby obserwacji. Metody imputacji jednokrotnej mogą wpływać negatywnie na dokładność danych. Metody imputacji wielokrotnej mogą wpływać pozytywnie na dokładność danych. W wyborze metody imputacji należy wykorzystywać miary i wykresy umożliwiające ocenę jakości imputacji. W programie R dostępnych jest wiele narzędzi programowych wspomagających imputację brakujących danych.

Wybór metody imputacji zależy od wielu czynników, takich jak: sposób gromadzenia danych (źródła pierwotne i wtórne), struktura zbioru danych (dane ustrukturyzowane i nieustrukturyzowane), skale pomiaru zmiennych (mocne i słabe), jakość danych (błędy losowe i nielosowe). W literaturze przedmiotu preferowane jest podejście stosowania różnych metod imputacji dla różnych typów zmiennych i różnego rozkładu brakujących wartości w rekordach oraz testowania, weryfikacji i oce-

ny jakości wyników imputacji. Praca (Hameed i Ali, 2023) zawiera przegląd najczęściej stosowanych metod imputacji brakujących danych ze wskazaniami ich zalet i ograniczeń.

Literatura

Balicki, A. (2004). Metody imputacji brakujących danych w badaniach statystycznych. *Wiadomości Statystyczne*, (9), 1-19.

Everitt, B. i Hothorn, T. (2011). *An Introduction to Applied Multivariate Analysis with R*. Springer.

Graham, J. W. (2012). *Missing Data. Analysis and Design*. Springer.

Hameed, W. M. i Ali, N. A. (2023). Missing Value Imputation Techniques: A Survey. *UHD Journal of Science and Technology*, *7*(1), 72-81. **https://doi.org/10.21928/uhdjst.v7n1y2023.pp72-81**

Harrell, F. E. (2023). *Hmisc: Harrell Miscellaneous. R Package Version 5.1-1*. **https://CRAN.R-project.org/package=Hmisc**

Honaker, J., King, G. i Blackwell, M. (2022). *Amelia: A Program for Missing Data*. R package version 1.8.1. **https://CRAN.R-project.org/package=Amelia**

Kalton, G. i Kasprzyk, D. (1986). The Treatment of Missing Survey Data. *Survey Methodology*, *12*(1), 1-16.

Kordos, J. (1988). *Jakość danych statystycznych*. Polskie Wydawnictwo Ekonomiczne.

Kozłowski, A. i Szreder, M. (2020). *Informacje spoza próby w badaniach statystycznych*. Wydawnictwo Uniwersytetu Gdańskiego.

Laaksonen, S. (2018). *Survey Methodology and Missing Data. Tools and Techniques for Practitioners*. Springer. **https://doi.org/10.1007/978-3-319-79011-4**

Misztal, M. (2012a). Imputation of Missing Data Using R Package. *Acta Universitatis Lodziensis. Folia Oeconomica*, (269), 131-144.

Misztal, M. (2012b). Wpływ wybranych metod uzupełniania brakujących danych na wyniki klasyfikacji obiektów z wykorzystaniem drzew klasyfikacyjnych w przypadku zbiorów danych o niewielkiej liczebności – ocena symulacyjna. *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, (242), 370-379.

Piasecki, T. (2014). Metody imputacji w badaniach gospodarstw domowych. *Wiadomości Statystyczne. The Polish Statistician*, *59*(9), 1-20. **https://doi.org/10.59139/ws.2014.09.1**

Pokropek, A. (2018). Wybrane statystyczne metody radzenia sobie z brakami danych. *Polskie Forum Psychologiczne*, *23*(2), 291-310. **https://doi.org/10.14656/PFP20180205**

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Rubin, D. B. (1976). Inference and Missing Data. *Biometrika*, *63*(3), 581-592.

Spector, P. (2008). *Data Manipulation with R*. New York: Springer.

Szpadel, M. i Kania K. (2023). *bdl: Interface and Tools for 'BDL' API*. R package version 1.0.5. **https://CRAN.R-project.org/package=bdl**

Szreder, M. (2004). *Metody i techniki sondażowych badań opinii*. Polskie Wydawnictwo Ekonomiczne.

Van Buuren, S. (2018). *Flexible Imputation of Missing Data* (2nd Edition). Chapman and Hall/CRC.

Van Buuren, S. i Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. *Journal of Statistical Software*, *45*(3), 1-67. **https://doi.org/10.18637/jss.v045.i03**

Van Buuren, S., Groothuis-Oudshoorn, K. i in. (2023). *mice: Multivariate Imputation by Chained Equations*. R package version 3.16.0. **https://cran.r-project.org/web/packages/mice**

Zambrano-Bigiarini, M. (2020). *hydroGOF: Goodness-of-Fit Functions for Comparison of Simulated and Observed Hydrological Time Series*. R package version 0.4-0. **https://CRAN.R-project.org/package=hydroGOF**

DOI: 10.15611/2024.45.1.07

7 Porządkowanie liniowe

7.1. Istota i założenia porządkowania liniowego

Zadaniem metod porządkowania liniowego zbioru obiektów jest uszeregowanie, czyli ustalenie kolejności obiektów lub ich zbiorów według określonego kryterium. Metody te mogą być zatem stosowane wtedy, gdy można przyjąć pewne nadrzędne kryterium, ze względu na które będzie można uporządkować obiekty od „najlepszego" do „najgorszego". Narzędziem metod porządkowania liniowego jest syntetyczny miernik rozwoju (miara agregatowa, będąca pewną funkcją agregującą informacje cząstkowe zawarte w poszczególnych zmiennych i wyznaczoną dla każdego obiektu ze zbioru obiektów *A*.

Przeprowadzenie porządkowania liniowego zbioru obiektów wymaga spełnienia następujących założeń (Abrahamowicz, 1985; Walesiak, 1993, s. 73).

a. Dany jest co najmniej dwuelementowy i skończony zbiór obiektów.
b. Istnieje pewne nadrzędne syntetyczne kryterium porządkowania (zjawisko złożone) elementów zbioru *A*, które nie podlega pomiarowi bezpośredniemu (np. poziom rozwoju gospodarczego państw świata, poziom spójności społecznej).
c. Dany jest skończony zbiór zmiennych merytorycznie związany z syntetycznym kryterium porządkowania. Zmienne mają charakter preferencyjny, tzn. wyróżnia się wśród nich stymulanty (większe wartości są bardziej preferowane), destymulanty (mniejsze wartości są bardziej preferowane) i nominanty (najbardziej preferowane są wartość lub przedział nominalny znajdujące się pomiędzy minimum i maksimum). Pojęcia stymulanty i destymulanty wprowadził Hellwig (1968), a nominanty – Borys (1978). Przeciwieństwem zmiennych preferencyjnych są zmienne neutralne (obojętne) (Borys, 1984, s. 111, 121). Formalne definicje stymulanty i destymulanty zawiera praca (Hellwig, 1981, s. 48), a nominanty praca (Borys, 1984, s. 118). Definicje dostępne są również w pracy (Walesiak, 2018).
d. Zmienne służące do opisu obiektów są mierzone przynajmniej na skali porządkowej (ze względu na to, że porządkowanie obiektów staje się możliwe, gdy dopuszczalne jest określenie na wartościach zmiennych przynajmniej relacji większości i mniejszości). Jeśli zmienne opisujące obiekty mierzone są na skali przedziałowej i (lub) ilorazowej należy sprowadzić je do porównywalności poprzez normalizację.
e. Relacją porządkującą elementy zbioru *A* jest relacja większości lub mniejszości dotycząca liczbowych wartości syntetycznego miernika rozwoju.

Ze względu na charakter prezentowanych danych w Banku Danych Lokalnych (BDL) w dalszej analizie, z wykorzystaniem porządkowania liniowego, stosowane będą dane metryczne (mierzone na skali przedziałowej lub ilorazowej) oraz symboliczne interwałowe.

7.2. Klasyczne i nieklasyczne procedury porządkowania liniowego

Ogólny schemat postępowania w porządkowaniu liniowym zbioru obiektów wygląda jak opisano dalej.

a. Dla metod bazujących na wzorcu (górny biegun) lub antywzorcu (dolny biegun) z wykorzystaniem danych metrycznych oraz symbolicznych interwałowych jest następujący:

$$P \rightarrow A \rightarrow X \rightarrow D \rightarrow \text{SDN} \rightarrow T_w \rightarrow N \rightarrow SM_w \rightarrow R, \tag{7.1}$$

gdzie: P – wybór zjawiska złożonego (nadrzędne syntetyczne kryterium porządkowania elementów zbioru A, które nie podlega pomiarowi bezpośredniemu), A – wybór obiektów, X – dobór zmiennych, D – zgromadzenie danych i konstrukcja macierzy danych $D = [x_{ij}]$ dla danych metrycznych (x_{ij} – wartość j-tej zmiennej dla i-tego obiektu) lub tablicy danych dla danych symbolicznych interwałowych $D = [x_{ij}^l, x_{ij}^u]$, gdzie x_{ij}^l oznacza dolną granicę interwału, a x_{ij}^u górną granicę interwału ($x_{ij}^l \leq x_{ij}^u$), SDN – identyfikacja zmiennych preferencyjnych (stymulanty, destymulanty, nominanty), T_w – transformacja nominant w stymulanty (wymagana tylko dla antywzorca). Metody transformacji zawiera m.in. praca (Walesiak, 2016a, s. 19), N – normalizacja wartości zmiennych. Dane symboliczne interwałowe wymagają specjalnego podejścia do problemu normalizacji wartości zmiennych. Dolną i górną granicę interwału j-tej zmiennej łączy się w jeden wektor zawierający $2n$ obserwacji. Takie podejście umożliwia zastosowanie metod normalizacji właściwych dla danych metrycznych. Normalizację zmiennych metrycznych przeprowadza się z wykorzystaniem funkcji `data.Normalization` pakietu `clusterSim`, a dla zmiennych symbolicznych interwałowych z wykorzystaniem funkcji `interval_normalization` pakietu `clusterSim` (Walesiak i Dudek, 2023), SM_w – obliczenie wartości miary syntetycznej (zmienna syntetyczna) przez agregację znormalizowanych wartości zmiennych – zastosowanie miar odległości od wzorca lub antywzorca z udziałem wag, R – uporządkowanie obiektów według wartości zmiennej syntetycznej (agregatowej).

b. Dla metod bezwzorcowych z wykorzystaniem danych metrycznych:

$$P \rightarrow A \rightarrow X \rightarrow D \rightarrow \text{SDN} \rightarrow T_b \rightarrow N \rightarrow SM_b \rightarrow R, \tag{7.2}$$

gdzie: $D = [x_{ij}]$ – zgromadzenie danych i konstrukcja macierzy danych $D = [x_{ij}]$, T_b – transformacja destymulant i nominant w stymulanty; metody transformacji zawiera m.in. praca (Walesiak, 2016a, s. 19), SM_b – obliczenie wartości miary syntetycznej (zmienna syntetyczna) przez agregację znormalizowanych wartości zmiennych – uśrednienie znormalizowanych wartości zmiennych z udziałem wag.

Pierwszy referat poświęcony koncepcji wzorca rozwoju i miary rozwoju w języku angielskim został zaprezentowany przez Hellwiga na konferencji UNESCO w Warszawie w 1967 r. (Hellwig, 1967). Praca ukazała się drukiem w języku angielskim w monografii pod redakcją Gostkowskiego (Hellwig, 1972). Pierwszy artykuł poświęcony koncepcji wzorca rozwoju i miary rozwoju w języku polskim ukazał się w czasopiśmie *Przegląd Statystyczny* w 1968 r. (Hellwig, 1968).

W pracach tych wprowadzono pojęcia:

- stymulant i destymulant (*stimulants and dis-stimulants*),
- wzorca rozwoju (*pattern of development*),
- miary rozwoju (*measure of development*) jako odległości od wzorca rozwoju (*distance from the pattern of development*).

Idea Hellwiga zapoczątkowała, można bez przesady powiedzieć, lawinę propozycji tworzenia metod porządkowania liniowego. Modyfikacje te zmierzały do (Borys i in., 1990; Pociecha i Zając, 1990):

- różnicowania sposobu normalizacji wartości zmiennych,
- wprowadzenia do zbioru zmiennych nominant,
- odmiennego ustalania wzorca rozwoju (bazy porównawczej),
- wykorzystania różnych konstrukcji miary agregatowej (tzw. miary rozwoju),
- wykorzystania zbiorów rozmytych w konstrukcji miary agregatowej.

W ostatnim okresie powstały koncepcje oraz zastosowania porządkowania liniowego (Walesiak i Dehnel, 2022; Walesiak i in., 2024):

- wykorzystujące liczby rozmyte (Jefmański i Dudek, 2016; Wysocki, 2010),
- wykorzystujące dane symboliczne interwałowe (Młodak, 2014),
- uwzględniające zależności przestrzenne (Antczak, 2013; Pietrzak, 2014),
- wykorzystujące połączenie skalowania wielowymiarowego z porządkowaniem liniowym dla danych klasycznych (Walesiak, 2016b) oraz symbolicznych interwałowych (Walesiak i Dehnel, 2018; Dehnel i Walesiak, 2019),
- bazujące na taksonomii relatywnej (Wydymus, 2013; Lira, 2015),
- wykorzystujące analizę głównych składowych (PCA) do porządkowania liniowego obiektów na podstawie wartości pierwszej głównej składowej (Bąk, 2018; Perkal, 1967),
- bazujące na miarach agregatowych z funkcją kary (Mazziotta i Pareto, 2016; 2018),
- bazujące na miarach agregatowych z poprawką na otoczenie danego obiektu (Łysoń i in., 2016),
- wykorzystujące elastyczne porządkowanie liniowe (Sokołowski i Markowska, 2019).

7.3. Miary agregatowe w porządkowaniu liniowym

Narzędziem metod porządkowania liniowego jest miara syntetyczna, będąca pewną funkcją agregującą informacje cząstkowe zawarte w poszczególnych zmiennych i wyznaczoną dla każdego obiektu ze zbioru obiektów. Formuły agregacji wartości zmiennych (konstrukcje syntetycznych mierników rozwoju – SMR) można ogólnie podzielić na (Grabiński, 1984, s. 38):

- wzorcowe,
- bezwzorcowe.

W formułach bezwzorcowych następuje uśrednienie znormalizowanych wartości zmiennych, z udziałem przyjętych wag. Formuły wzorcowe są różnego rodzaju odległościami poszczególnych obiektów od obiektu wzorcowego, którym w badaniach empirycznych jest na ogół tzw. dolny (antywzorzec) lub górny biegun rozwoju (wzorzec). Współrzędne wzorca obejmują najkorzystniejsze wartości zmiennych preferencyjnych (maksymalne dla stymulant i minimalne dla destymulant). Współrzędne antywzorca obejmują najmniej korzystne wartości zmiennych preferencyjnych (minimalne dla stymulant i maksymalne dla destymulant). Dla zmiennych symbolicznych interwałowych współrzędne wzorca i antywzorca wyznacza się osobno dla dolnej i górnej granicy interwału.

Tabela 7.1 zawiera typowe miary agregatowe (formuły SMR) oparte na wzorcu rozwoju stosowane dla danych metrycznych.

Tabela 7.1. Typowe miary agregatowe (formuły SMR) bazujące na wzorcu rozwoju dla danych metrycznych

Nazwa	Odległość d_i	Przedział zmienności
Miara rozwoju I (Hellwig, 1968)	$1-\frac{d_i^+}{\bar{d}^+ + 2s_d}$	$(-\infty; 1]$
Miara rozwoju II (Hellwig, 1981)	$1-\frac{d_i^+}{\sqrt{\sum_{j=1}^m \alpha_j (z_{+j}-z_{-j})^2}}$	$[0; 1]$
Miara TOPSIS (Hwang i Yoon, 1981)	$\frac{d_i^-}{d_i^- + d_i^+}$	$[0; 1]$
Odległość GDM1 (Walesiak, 2002)	$1-\mathrm{GDM1}_i^+$	$[0; 1]$
GDM1_TOPSIS (miara TOPSIS z odległością GDM1) (Walesiak, 2014)	$\frac{\mathrm{GDM1}_i^-}{\mathrm{GDM1}_i^- + \mathrm{GDM1}_i^+}$	$[0; 1]$

$i, l = 1, \ldots, n$ – numer obiektu, $j = 1, \ldots, m$ – numer zmiennej; $x_{+j}(x_{-j})$ – j-ta współrzędna obiektu wzorca (antywzorca), $z_{+j}(z_{-j})$ – znormalizowana j-ta współrzędna obiektu wzorca (antywzorca);

$\mathrm{GDM1}_i = \frac{1}{2} - \frac{\sum_{j=1}^m \alpha_j (z_{ij}-z_{wj})(z_{wj}-z_{ij}) + \sum_{j=1}^m \sum_{\substack{l=1 \\ l\neq i,w}}^n \alpha_j (z_{ij}-z_{lj})(z_{wj}-z_{lj})}{2\left[\sum_{j=1}^m \sum_{l=1}^n \alpha_j (z_{ij}-z_{lj})^2 \cdot \sum_{j=1}^m \sum_{l=1}^n \alpha_j (z_{wj}-z_{lj})^2\right]^{0,5}}$, gdzie $z_{wj} = z_{+j}$ $(z_{wj} = z_{-j})$ dla $\mathrm{GDM1}_i^+$ $(\mathrm{GDM1}_i^-)$; $\mathrm{GDM1}_i^-$ $(\mathrm{GDM1}_i^+)$ – odległość GDM1 obiektu i-tego od antywzorca (wzorca);

$d_i^+ = \sqrt{\sum_{j=1}^m \alpha_j^2 (z_{ij}-z_{+j})^2}$ – ważona odległość Euklidesa obiektu i-tego od wzorca;

$d_i^- = \sqrt{\sum_{j=1}^m \alpha_j^2 (z_{ij}-z_{-j})^2}$ – ważona odległość Euklidesa obiektu i-tego od antywzorca;

$\bar{d}^+ = \frac{1}{n}\sum_{i=1}^n d_i^+$, $s_d = \frac{1}{n}\sqrt{\sum_{i=1}^n (d_i^+ - \bar{d}^+)^2}$;

α_j – waga j-tej zmiennej ($\alpha_j \in [0; 1]$ i $\sum_{j=1}^m \alpha_j = 1$ lub $\alpha_j \in [0; m]$ i $\sum_{j=1}^m \alpha_j = m$).

Źródło: opracowanie własne.

Do typowych miar agregatowych (formuły SMR bezwzorcowe) dla danych metrycznych zalicza się:

$$p_i = \frac{1}{\sum_{j=1}^m \alpha_j} \sum_{j=1}^m \alpha_j z_{ij},\ i = 1, \ldots, n, \tag{7.3}$$

$$p_i = \frac{\sum_{j=1}^{m} \alpha_j}{\sum_{j=1}^{m} \frac{\alpha_j}{z_{ij}}}, i = 1, \ldots, n, \tag{7.4}$$

$$p_i = \left[\prod_{j=1}^{m} z_{ij}^{\alpha_j}\right]^{1/\sum_{j=1}^{m} \alpha_j}, i = 1, \ldots, n, \tag{7.5}$$

gdzie: p_i – wartość zmiennej syntetycznej (SMR) dla i-tego obiektu; z_{ij} – znormalizowana wartość j-tej zmiennej w i-tym obiekcie.

Formuły (7.3)-(7.5) to odpowiednio trzy postacie wartości przeciętnej – średnia arytmetyczna, średnia harmoniczna i średnia geometryczna. Można zastosować inne miary pozycyjne, np. medianę.

Dla danych symbolicznych interwałowych można zastosować miarę agregatową d_i bazującą na odległości Euklidesowej Ichino-Yaguchiego (Ichino i Yaguchi, 1994) od obiektu wzorca (Dehnel i Walesiak, 2019):

$$d_i = 1 - \sqrt{\sum_{j=1}^{m} \varphi\left(z_{ij}, z_{+j}\right)^2} \Big/ \sqrt{\sum_{j=1}^{m} \varphi\left(z_{+j}, z_{-j}\right)^2}, \tag{7.6}$$

gdzie: $z_{ij} = \left[z_{ij}^l, z_{ij}^u\right]$; $z_{+j} = \left[z_{+j}^l, z_{+j}^u\right]$; $z_{-j} = \left[z_{-j}^l, z_{-j}^u\right]$;
z_{ij}^l i z_{ij}^u – dolna i górna granica interwału dla j-tej zmiennej;
z_{+j}^l i z_{+j}^u (z_{-j}^l i z_{-j}^u) – dolna i górna granica interwału dla j-tej zmiennej dla obiektu wzorca (antywzorca);
$\varphi\left(z_{ij}, z_{+j}\right) = \left|z_{ij} \oplus z_{+j}\right| - \left|z_{ij} \otimes z_{+j}\right| + \gamma\left(2 \cdot \left|z_{ij} \otimes z_{+j}\right| - \left|z_{ij}\right| - \left|z_{+j}\right|\right)$;
$\varphi\left(z_{+j}, z_{-j}\right) = \left|z_{+j} \oplus z_{-j}\right| - \left|z_{+j} \otimes z_{-j}\right| + \gamma\left(2 \cdot \left|z_{+j} \otimes z_{-j}\right| - \left|z_{+j}\right| - \left|z_{-j}\right|\right)$;
$|\ \ |$ – długość interwału;
$z_{ij} \oplus z_{+j} = z_{ij} \cup z_{+j}$; $z_{ij} \otimes z_{+j} = z_{ij} \cap z_{+j}$;
$z_{+j} \oplus z_{-j} = z_{+j} \cup z_{-j}$; $z_{+j} \otimes z_{-j} = z_{+j} \cap z_{-j}$.

Miara agregatowa d_i o postaci (7.6) przybiera wartości z przedziału $[0; 1]$. Wyższe wartości miary agregatowej d_i oznaczają wyższy poziom rozwoju obiektu o numerze i. Obiekty porządkuje się według malejących wartości miary agregatowej d_i.

Podobnie można skonstruować miarę agregatową TOPSIS uwzględniającą odległość Euklidesową Ichino-Yaguchiego.

7.4. Zastosowania z wykorzystaniem programu R dla danych z BDL

Dane metryczne

Przeprowadzono porządkowanie liniowe województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r. W analizie uwzględniono następujące zmienne metryczne:

x1 – odsetek ogółu mieszkań w miastach wyposażonych w wodociąg,
x2 – odsetek ogółu mieszkań w miastach wyposażonych w łazienkę,
x3 – odsetek ogółu mieszkań w miastach wyposażonych w centralne ogrzewanie,

x4 – odsetek ogółu mieszkań w miastach wyposażonych w ustęp spłukiwany,
x5 – odsetek ogółu mieszkań w miastach wyposażonych w gaz sieciowy,
x6 – przeciętna liczba osób na 1 mieszkanie w miastach,
x7 – przeciętna liczba osób na 1 izbę w miastach,
x8 – przeciętna powierzchnia użytkowa w m^2 na 1 osobę w miastach,
x9 – przeciętna powierzchnia użytkowa 1 mieszkania w m^2 w miastach,
x10 – przeciętna liczba izb w mieszkaniu w miastach.

W tabeli 7.2 przedstawiono zmienne dostępne w Banku Danych Lokalnych oraz wymagane przeliczenia.

Tabela 7.2. Zmienne dostępne w BDL oraz wymagane przeliczenia

Zmienne dostępne w BDL	ID	Wymagane przeliczenia
v1 – odsetek ogółu mieszkań w miastach wyposażonych w wodociąg	60575	x1 = v1
v2 – odsetek ogółu mieszkań w miastach wyposażonych w łazienkę	60579	x2 = v2
v3 – odsetek ogółu mieszkań w miastach wyposażonych w centralne ogrzewanie	60576	x3 = v3
v4 – mieszkania w miastach wyposażone w ustęp spłukiwany	60837	x
v5 – mieszkania w miastach wyposażone w gaz sieciowy	60842	x
v6 – ludność ogółem w miastach (osoby)	453327	x
v7 – zasoby mieszkaniowe w miastach – powierzchnia użytkowa mieszkań w m^2	60810	x
v8 – zasoby mieszkaniowe w miastach – izby	60809	x
v9 – zasoby mieszkaniowe w miastach – mieszkania	60807	x
x4 – odsetek ogółu mieszkań w miastach wyposażonych w ustęp spłukiwany	x	x4 = (v4/v9) × 100
x5 – odsetek ogółu mieszkań w miastach wyposażonych w gaz sieciowy	x	x5 = (v5/v9) × 100
x6 – przeciętna liczba osób na 1 mieszkanie w miastach	x	x6 = v6/v9
x7 – przeciętna liczba osób na 1 izbę w miastach	x	x7 = v6/v8
x8 – przeciętna powierzchnia użytkowa w m^2 na 1 osobę w miastach	x	x8 = v7/v6
x9 – przeciętna powierzchnia użytkowa 1 mieszkania w m^2 w miastach	x	x9 = v7/v9
x10 – przeciętna liczba izb w mieszkaniu w miastach	x	x10 = v8/v9

Źródło: opracowanie własne.

Skrypt 7.1 pozwala na automatyczne pozyskanie danych z Banku Danych Lokalnych.

Skrypt 7.1

```
library(bdl)
library(dplyr)
options(OutDec=",")
# stałe
rok<-2021
no_uL=2 # (2 - województwo)
# Wczytanie informacji o zmiennych z BDL
v1<-get_data_by_variable("60575",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
v2<-get_data_by_variable("60579",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
v3<-get_data_by_variable("60576",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
```

```
v4<-get_data_by_variable("60837",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
v5<-get_data_by_variable("60842",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x5=val)
v6<-get_data_by_variable("453327",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x6=val)
v7<-get_data_by_variable("60810",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x7=val)
v8<-get_data_by_variable("60809",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x8=val)
v9<-get_data_by_variable("60807",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x9=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
data<-v1 %>%
  full_join (v2, by=c("id","name")) %>%
  full_join (v3, by=c("id","name")) %>%
  full_join (v4, by=c("id","name")) %>%
  full_join (v5, by=c("id","name")) %>%
  full_join (v6, by=c("id","name")) %>%
  full_join (v7, by=c("id","name")) %>%
  full_join (v8, by=c("id","name")) %>%
  full_join (v9, by=c("id","name"))
v<-as.data.frame(data)
# Nadanie nazw wierszom
nazwy<-v[,2]
rownames(v)<-tolower(nazwy)
# Wybranie 9 kolumn z danymi oraz dodanie 7 pustych na przeliczenia
v<-v[,3:11]
for(i in 1:7){
  v<-cbind(v,rep(0,nrow(v)))
}
# Nadanie nazw kolumnom
colnames(v)<-paste("v",1:16,sep="")
# Przeliczenia
v<-mutate(v,v10=round((v4/v9)*100,2))
v<-mutate(v,v11=round((v5/v9)*100,2))
v<-mutate(v,v12=round((v6/v9),2))
v<-mutate(v,v13=round((v6/v8),2))
v<-mutate(v,v14=round((v7/v6),2))
v<-mutate(v,v15=round((v7/v9),2))
v<-mutate(v,v16=round((v8/v9),2))
# Usunięcie zbędnych kolumn
x<-dplyr::select(v,-c(4:9))
# Nadanie nazw wierszom i kolumnom
colnames(x)<-paste("x",1:10,sep="")
rownames(x)<-tolower(nazwy)
print(x)
# Zapisanie danych do pliku
write.table(x, file="dane_wm_2021.csv", sep=";", dec=",",
  row.names=TRUE,col.names=NA)
```

Po wykonaniu skryptu 7.1 pobrany zostaje zbiór danych z BDL dla 10 zmiennych opisujących poziom warunków zamieszkiwania ludności w miastach w 2021 r. (dane zostają zapisane do pliku `dane_wm_2021.csv`). Po wykonaniu polecenia `print(x)` otrzymujemy:

```
                    x1   x2   x3    x4    x5   x6   x7    x8    x9  x10
małopolskie         98,3 97,5 90,9 97,81 71,24 2,13 0,62 30,48 64,82 3,44
śląskie             99,2 95,8 85,1 96,61 74,41 2,24 0,63 29,24 65,40 3,55
lubuskie            98,9 96,7 88,9 97,77 75,84 2,28 0,63 29,01 66,05 3,62
wielkopolskie       98,4 97,0 88,2 97,70 69,62 2,27 0,61 30,93 70,16 3,71
zachodniopomorskie  98,9 97,3 90,6 97,77 76,79 2,20 0,62 29,02 63,71 3,51
dolnośląskie        98,7 94,8 86,7 95,88 78,95 2,07 0,61 29,98 62,00 3,39
opolskie            98,7 96,9 88,3 97,64 78,82 2,31 0,63 29,56 68,25 3,68
kujawsko-pomorskie  98,7 96,0 87,7 97,67 72,23 2,25 0,64 27,13 60,96 3,50
pomorskie           97,8 96,7 90,6 97,44 68,76 2,17 0,63 29,00 63,05 3,43
warmińsko-mazurskie 99,3 98,0 91,4 98,83 63,69 2,35 0,66 26,22 61,56 3,56
łódzkie             97,7 93,2 85,3 95,02 63,19 2,02 0,60 30,22 61,15 3,38
świętokrzyskie      98,5 95,8 90,3 96,90 71,66 2,24 0,63 28,99 64,83 3,57
lubelskie           98,8 96,6 90,7 97,43 73,49 2,26 0,61 29,41 66,49 3,70
podkarpackie        98,9 97,6 91,5 98,00 88,41 2,48 0,64 29,00 71,95 3,88
podlaskie           99,3 97,8 92,5 98,38 47,28 2,33 0,61 28,94 67,40 3,79
mazowieckie         98,2 97,0 93,0 97,64 70,34 2,00 0,60 32,08 64,17 3,35
```

Zastosowano funkcję `pattern.GDM1` pakietu `clusterSim` do porządkowania liniowego województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r. (dane z pliku `dane_wm_2021.csv`). W składni poleceń dla skryptu 7.2 przyjęto następującą metodykę postępowania:

- dla każdej zmiennej określono charakter preferencji: stymulanty (zmienne od `x1` do `x5`, `x8`, `x9`, `x10`); destymulanty (`x6`, `x7`), zatem:
  `performanceVariable=c("s","s","s","s","s","d","d","s","s","s");`
- dla każdej zmiennej ustalono skalę jej pomiaru (skale ilorazowa i przedziałowa); wszystkie zmienne mierzone są na skali ilorazowej, a zatem `scaleType="r"`;
- przeprowadzono normalizację wartości zmiennych z zastosowaniem odpowiedniej metody normalizacji wartości zmiennych; w tym przypadku dopuszczalne są wszystkie metody normalizacyjne; zastosowano tutaj standaryzację pozycyjną: `normalization="n2"`;
- jako miarę agregatową zastosowano miarę `TOPSIS_GDM1` – TOPSIS z odległością GDM1;
- przyjęto następujące współrzędne obiektu wzorca:
  (`patternCoordinates="manual"`)
  `patternManual=c(100,100,100,100,100,"min","min","max","max","max");`
- współrzędne obiektu antywzorca wyznaczono automatycznie:
  (`patternCoordinates="dataBounds"`);
- w analizie zastosowano wagi jednakowe (`weightsType="equal"`).

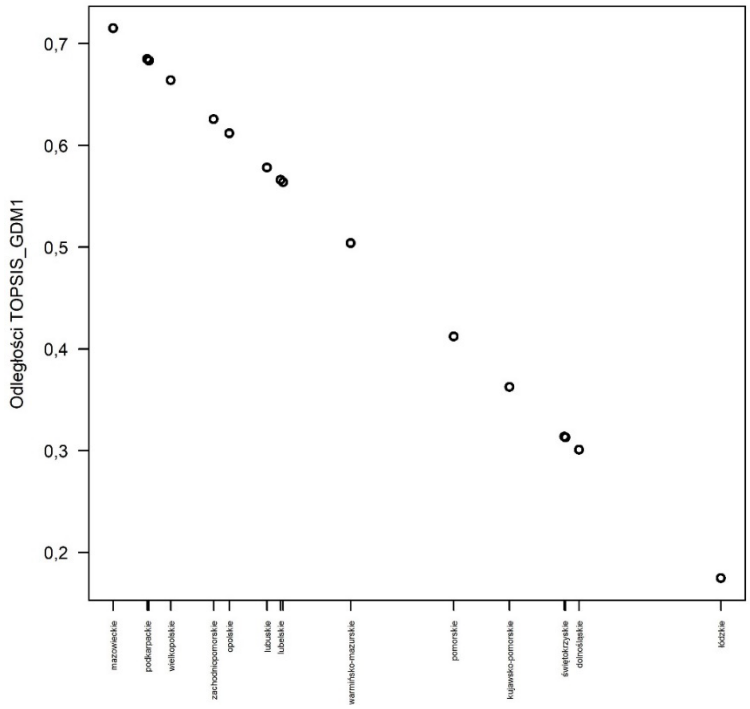
Skrypt 7.2

```
library(clusterSim)
x<-read.csv2("dane_wm_2021.csv",header=TRUE,row.names=1)
options(OutDec=",")
res_P<-pattern.GDM1(x,
  performanceVariable=c("s","s","s","s","s","d","d","s","s","s"),
  scaleType="r",normalization="n2",patternType="upper",
  patternCoordinates="manual",
  patternManual=c(100,100,100,100,100,"min","min","max","max","max"))
res_AP<-pattern.GDM1(x,
```

```
  performanceVariable=c("s","s","s","s","s","d","d","s","s","s"),
  scaleType="r",normalization="n2",patternType="lower",
  patternCoordinates="dataBounds")
TOPSIS_GDM<-res_AP$distances/(res_P$distances+res_AP$distances)
print("Uporządkowanie województw od najlepszego do najgorszego
  według miary TOPSIS_GDM1",quote=FALSE)
print(sort(TOPSIS_GDM,decreasing=TRUE))
plot(cbind(TOPSIS_GDM,TOPSIS_GDM),xlim=c(max(TOPSIS_GDM),min(TOPSIS_GDM)),
  ylim=c(min(TOPSIS_GDM),max(TOPSIS_GDM)),xaxt="n",xlab="",
  ylab="Odległości TOPSIS_GDM1",lwd=1.6,las=1)
axis(1,at=TOPSIS_GDM,labels=names(TOPSIS_GDM),las=2,cex.axis=0.5)
```

W wyniku zastosowania procedury ze skryptu 7.2 otrzymano uporządkowanie województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r. (graficzną formę prezentacji zamieszczono na rys. 7.1):

```
 [1] Uporządkowanie województw od najlepszego do najgorszego według miary
TOPSIS_GDM1
        mazowieckie        podkarpackie         małopolskie       wielkopolskie
          0,7150870           0,6847683           0,6832568           0,6639389
 zachodniopomorskie            opolskie            lubuskie            lubelskie
          0,6258529           0,6118797           0,5782781           0,5662592
          podlaskie warmińsko-mazurskie            pomorskie  kujawsko-pomorskie
          0,5639498           0,5038874           0,4122568           0,3627049
     świętokrzyskie             śląskie       dolnośląskie              łódzkie
          0,3138245           0,3129283           0,3008636           0,1746968
```



Rys. 7.1. Graficzna prezentacja uporządkowania województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r. od najlepszego do najgorszego według wartości miary `TOPSIS_GDM1`

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Najlepsze warunki zamieszkiwania w miastach w 2021 r. były w województwie mazowieckim, najgorsze zaś w województwie łódzkim.

Dane symboliczne interwałowe

Przeprowadzono porządkowanie liniowe województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r., wykorzystując zmienne z tab. 7.2. W tym przypadku porządkowanie liniowe przeprowadzono na podstawie danych symbolicznych interwałowych. W celu otrzymania danych symbolicznych interwałowych zastosowano dwustopniowe gromadzenie danych. Najpierw zgromadzono klasyczne dane metryczne opisujące poziom warunków zamieszkiwania ludności w miastach według powiatów Polski (380 powiatów opisanych 10 zmiennymi) z wykorzystaniem skryptu 7.3.

Skrypt 7.3

```
library(bdl)
library(dplyr)
options(OutDec=",")
# stałe
rok<-2021
no_uL=5 # (5 - powiat)
# Wczytanie informacji o zmiennych z BDL
v1<-get_data_by_variable("60575",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
v2<-get_data_by_variable("60579",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
v3<-get_data_by_variable("60576",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
v4<-get_data_by_variable("60837",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
v5<-get_data_by_variable("60842",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x5=val)
v6<-get_data_by_variable("453327",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x6=val)
v7<-get_data_by_variable("60810",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x7=val)
v8<-get_data_by_variable("60809",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x8=val)
v9<-get_data_by_variable("60807",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x9=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
data<-v1 %>%
  full_join (v2, by=c("id","name")) %>%
  full_join (v3, by=c("id","name")) %>%
  full_join (v4, by=c("id","name")) %>%
  full_join (v5, by=c("id","name")) %>%
  full_join (v6, by=c("id","name")) %>%
  full_join (v7, by=c("id","name")) %>%
  full_join (v8, by=c("id","name")) %>%
  full_join (v9, by=c("id","name"))
```

```
# Usunięcie z nazwy obiektu wyrazu Powiat
data<-mutate(data,name=gsub("Powiat ","",name))
v<-as.data.frame(data)
# Nadanie nazw wierszom
nazwy<-v[,1]
rownames(v)<-tolower(nazwy)
# Wybranie 9 kolumn z danymi oraz dodanie 7 pustych na przeliczenia
v<-v[,3:11]
for(i in 1:7){
  v<-cbind(v,rep(0,nrow(v)))
}
# Nadanie nazw kolumnom
colnames(v)<-paste("v",1:16,sep="")
# Przeliczenia
v<-mutate(v,v10=round((v4/v9)*100,2))
v<-mutate(v,v11=round((v5/v9)*100,2))
v<-mutate(v,v12=round((v6/v9),2))
v<-mutate(v,v13=round((v6/v8),2))
v<-mutate(v,v14=round((v7/v6),2))
v<-mutate(v,v15=round((v7/v9),2))
v<-mutate(v,v16=round((v8/v9),2))
# Usunięcie zbędnych kolumn
x<-dplyr::select(v,-c(4:9))
# Nadanie nazw wierszom i kolumnom
colnames(x)<-paste("x",1:10,sep="")
rownames(x)<-tolower(nazwy)
# Zapisanie danych do pliku
write.table(x,file="dane_wm_interval_2021.csv",sep=";",dec=",",
  row.names=TRUE,col.names=NA)
```

Po wykonaniu skryptu 7.3 pobrany zostaje zbiór danych metrycznych z BDL dla 10 zmiennych opisujących poziom warunków zamieszkiwania ludności w miastach według powiatów w 2021 r. Fragment zbioru danych (zapisanych do pliku `dane_wm_interval_2021.csv`) dla 5 powiatów jest następujący (polecenie `print(x[1:5,])`):

```
              x1   x2   x3    x4    x5   x6   x7    x8    x9  x10
011212001000 98,8 98,1 87,5 98,43 91,47 2,61 0,64 30,25 79,03 4,11
011212006000 96,1 94,6 86,3 95,27 78,13 2,47 0,65 29,63 73,24 3,79
011212008000 98,6 97,1 84,1 97,92 21,67 2,29 0,60 31,20 71,37 3,82
011212009000 98,3 97,6 85,4 98,03 91,32 2,90 0,67 30,63 88,88 4,32
011212014000 98,9 96,7 85,3 98,05 27,81 2,56 0,64 31,27 80,21 3,99
...
```

W pierwszej kolumnie znajdują się kody powiatów.

Następnie dane z pliku `dane_wm_interval_2021.csv` poddano agregacji do poziomu województw otrzymując dane symboliczne interwałowe. Dolną oraz górną granicę interwału dla każdej zmiennej w województwie uzyskano, obliczając kwartyl 1 i kwartyl 3 (zob. skrypt 7.4).

Skrypt 7.4

```
# Wczytanie danych
dane<-read.csv2("dane_wm_interval_2021.csv",row.names=1,header=TRUE)
# Obliczenie kwartyli dolnych
w1<-apply(dane[1:22,],2,quantile,0.25,na.rm=TRUE)
w2<-apply(dane[23:58,],2,quantile,0.25,na.rm=TRUE)
w3<-apply(dane[59:72,],2,quantile,0.25,na.rm=TRUE)
w4<-apply(dane[73:107,],2,quantile,0.25,na.rm=TRUE)
w5<-apply(dane[108:128,],2,quantile,0.25,na.rm=TRUE)
w6<-apply(dane[129:158,],2,quantile,0.25,na.rm=TRUE)
w7<-apply(dane[159:170,],2,quantile,0.25,na.rm=TRUE)
w8<-apply(dane[171:193,],2,quantile,0.25,na.rm=TRUE)
w9<-apply(dane[194:213,],2,quantile,0.25,na.rm=TRUE)
w10<-apply(dane[214:234,],2,quantile,0.25,na.rm=TRUE)
w11<-apply(dane[235:258,],2,quantile,0.25,na.rm=TRUE)
w12<-apply(dane[259:272,],2,quantile,0.25,na.rm=TRUE)
w13<-apply(dane[273:296,],2,quantile,0.25,na.rm=TRUE)
w14<-apply(dane[297:321,],2,quantile,0.25,na.rm=TRUE)
w15<-apply(dane[322:338,],2,quantile,0.25,na.rm=TRUE)
w16<-apply(dane[339:380,],2,quantile,0.25,na.rm=TRUE)
dane_wm_interval_min<-rbind(w1,w2,w3,w4,w5,w6,w7,w8,w9,
   w10,w11,w12,w13,w14,w15,w16)
write.table(dane_wm_interval_min,file="dane_wm_interval_k25.csv",
   sep=";",dec=",",col.names=NA)
# Obliczenie kwartyli górnych
w1a<-apply(dane[1:22,],2,quantile,0.75,na.rm=TRUE)
w2a<-apply(dane[23:58,],2,quantile,0.75,na.rm=TRUE)
w3a<-apply(dane[59:72,],2,quantile,0.75,na.rm=TRUE)
w4a<-apply(dane[73:107,],2,quantile,0.75,na.rm=TRUE)
w5a<-apply(dane[108:128,],2,quantile,0.75,na.rm=TRUE)
w6a<-apply(dane[129:158,],2,quantile,0.75,na.rm=TRUE)
w7a<-apply(dane[159:170,],2,quantile,0.75,na.rm=TRUE)
w8a<-apply(dane[171:193,],2,quantile,0.75,na.rm=TRUE)
w9a<-apply(dane[194:213,],2,quantile,0.75,na.rm=TRUE)
w10a<-apply(dane[214:234,],2,quantile,0.75,na.rm=TRUE)
w11a<-apply(dane[235:258,],2,quantile,0.75,na.rm=TRUE)
w12a<-apply(dane[259:272,],2,quantile,0.75,na.rm=TRUE)
w13a<-apply(dane[273:296,],2,quantile,0.75,na.rm=TRUE)
w14a<-apply(dane[297:321,],2,quantile,0.75,na.rm=TRUE)
w15a<-apply(dane[322:338,],2,quantile,0.75,na.rm=TRUE)
w16a<-apply(dane[339:380,],2,quantile,0.75,na.rm=TRUE)
dane_wm_interval_max<-rbind(w1a,w2a,w3a,w4a,w5a,w6a,w7a,w8a,
  w9a,w10a,w11a,w12a,w13a,w14a,w15a,w16a)
write.table(dane_wm_interval_max,file="dane_wm_interval_k75.csv",
  sep=";",dec=",",col.names=NA)
```

W wyniku wykonania skryptu 7.4 dolne i górne granice interwałów dla 10 zmiennych zapisano w plikach `dane_wm_interval_k25.csv` i `dane_wm_interval_k75.csv`.

W przypadku zmiennych symbolicznych interwałowych współrzędne wzorca i antywzorca wyznacza się osobno dla dolnej i górnej wartości interwału (zob. skrypt 7.5). Zmienne opisujące poziom warunków zamieszkiwania ludności w miastach mają następujące preferencje: stymulanty

(od x1 do x5, x8, x9, x10); destymulanty (x6, x7). Dodając wzorzec i antywzorzec, otrzymano tablicę danych $\mathbf{X} = [x_{ij}^l, x_{ij}^u]_{18\times 10}$ $(x_{ij}^l \leq x_{ij}^u; i,k = 1,\ldots,18)$. Dolną granicę interwału dla 18 obiektów (16 województw, wzorzec i antywzorzec) zapisano w pliku dane_wm_lower_P-AP.csv, a górną granicę interwału w pliku dane_wm_upper_P-AP.csv.

Skrypt 7.5

```
d<-read.csv2("dane_wm_2021.csv",header=TRUE,row.names=1)
# Wczytanie danych
dane_wm_lower<-read.csv2("dane_wm_interval_k25.csv",row.names=1,header=TRUE)
# współrzędne wzorca (kwantyl 25)
x1<-max(dane_wm_lower[,1])
x2<-max(dane_wm_lower[,2])
x3<-max(dane_wm_lower[,3])
x4<-max(dane_wm_lower[,4])
x5<-max(dane_wm_lower[,5])
x6<-min(dane_wm_lower[,6])
x7<-min(dane_wm_lower[,7])
x8<-max(dane_wm_lower[,8])
x9<-max(dane_wm_lower[,9])
x10<-max(dane_wm_lower[,10])
dane_lower_P<-cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10)
colnames(dane_lower_P)<-paste("x",1:10,sep="")
# współrzędne antywzorca (kwantyl 25)
v1<-min(dane_wm_lower[,1])
v2<-min(dane_wm_lower[,2])
v3<-min(dane_wm_lower[,3])
v4<-min(dane_wm_lower[,4])
v5<-min(dane_wm_lower[,5])
v6<-max(dane_wm_lower[,6])
v7<-max(dane_wm_lower[,7])
v8<-min(dane_wm_lower[,8])
v9<-min(dane_wm_lower[,9])
v10<-min(dane_wm_lower[,10])
dane_lower_AP<-cbind(v1,v2,v3,v4,v5,v6,v7,v8,v9,v10)
colnames(dane_lower_AP)<-paste("x",1:10,sep="")
dane_wm_lower_P_AP<-rbind(dane_wm_lower,dane_lower_P,dane_lower_AP)
rownames(dane_wm_lower_P_AP)<-c(rownames(d),"P","AP")
write.table(dane_wm_lower_P_AP,file="dane_wm_lower_P-AP.csv",sep=";",
   dec=",",col.names=NA)
# Wczytanie danych (kwantyl 75)
dane_wm_upper<-read.csv2("dane_wm_interval_k75.csv",row.names=1, header=TRUE)
# współrzędne wzorca (kwantyl 75)
x1<-max(dane_wm_upper[,1])
x2<-max(dane_wm_upper[,2])
x3<-max(dane_wm_upper[,3])
x4<-max(dane_wm_upper[,4])
x5<-max(dane_wm_upper[,5])
x6<-min(dane_wm_upper[,6])
x7<-min(dane_wm_upper[,7])
x8<-max(dane_wm_upper[,8])
x9<-max(dane_wm_upper[,9])
x10<-max(dane_wm_upper[,10])
```

```
dane_upper_P<-cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10)
colnames(dane_upper_P)<-paste("x",1:10,sep="")
# współrzędne antywzorca (kwantyl 75)
v1<-min(dane_wm_upper[,1])
v2<-min(dane_wm_upper[,2])
v3<-min(dane_wm_upper[,3])
v4<-min(dane_wm_upper[,4])
v5<-min(dane_wm_upper[,5])
v6<-max(dane_wm_upper[,6])
v7<-max(dane_wm_upper[,7])
v8<-min(dane_wm_upper[,8])
v9<-min(dane_wm_upper[,9])
v10<-min(dane_wm_upper[,10])
dane_upper_AP<-cbind(v1,v2,v3,v4,v5,v6,v7,v8,v9,v10)
colnames(dane_upper_AP)<-paste("x",1:10,sep="")
dane_wm_upper_P_AP<-rbind(dane_wm_upper,dane_upper_P,dane_upper_AP)
rownames(dane_wm_upper_P_AP)<-c(rownames(d),"P","AP")
write.table(dane_wm_upper_P_AP,file="dane_wm_upper_P-AP.csv",
  sep=";", dec=",",col.names=NA)
```

Wykonując polecenie `print(dane_wm_lower_P_AP)`, otrzymujemy macierz danych obejmującą dolne wartości interwałów dla 18 obiektów (16 województw, wzorzec i antywzorzec):

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
małopolskie	97,950	96,400	84,850	97,2375	68,1900	2,4175	0,6325	28,5200	69,0975	3,7975
śląskie	98,900	95,875	81,850	96,7100	63,3275	2,1900	0,6200	27,4475	62,5100	3,4425
lubuskie	98,800	95,975	84,700	97,1450	65,9425	2,3200	0,6325	27,6275	65,3200	3,5900
wielkopolskie	98,250	95,800	84,450	97,3100	44,6700	2,4050	0,6150	28,7250	71,2350	3,8400
zachodniopomorskie	98,900	96,700	86,700	98,0200	65,1100	2,2500	0,6100	26,8200	62,9700	3,5600
dolnośląskie	98,300	92,600	81,075	94,4850	73,1650	2,2250	0,6300	27,5625	62,6625	3,4625
opolskie	98,175	95,650	86,700	96,7700	71,1050	2,2725	0,6200	27,9950	67,4675	3,6275
kujawsko-pomorskie	98,450	94,900	82,500	97,1450	28,4000	2,3450	0,6450	25,5350	59,5150	3,4750
pomorskie	96,825	95,125	85,650	96,5275	53,5675	2,2025	0,6100	25,9725	62,1850	3,4925
warmińsko-mazurskie	99,000	97,200	88,600	98,5400	15,5100	2,3700	0,6500	25,0400	59,6800	3,5300
łódzkie	97,175	92,575	83,600	94,9150	10,5300	2,1500	0,6000	28,8100	64,5400	3,5500
świętokrzyskie	98,525	94,225	86,625	95,8425	11,2175	2,2900	0,6200	28,4450	64,5425	3,6475
lubelskie	98,075	93,950	86,925	95,4150	43,1550	2,3700	0,6075	28,0525	67,2250	3,7175
podkarpackie	98,200	96,000	87,500	96,9900	81,4500	2,5700	0,6500	27,2000	69,5200	3,8400
podlaskie	98,100	94,600	86,700	96,3650	0,3450	2,4250	0,5975	27,1325	68,2450	3,8575
mazowieckie	97,250	94,525	85,850	95,6150	21,6525	2,2825	0,6000	29,3450	67,7900	3,6350
P	99,000	97,200	88,600	98,5400	81,4500	2,1500	0,5975	29,3450	71,2350	3,8575
AP	96,825	92,575	81,075	94,4850	0,3450	2,5700	0,6500	25,0400	59,5150	3,4425

a wykonując polecenie `print(dane_wm_upper_P_AP)`, otrzymujemy macierz danych obejmującą górne wartości interwałów dla 18 obiektów (16 województw, wzorzec i antywzorzec):

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
małopolskie	99,200	98,325	89,625	98,7600	91,0175	2,7150	0,670	31,0575	84,4225	4,1375
śląskie	99,400	98,200	88,475	98,4450	80,9800	2,5600	0,660	31,1800	78,5350	4,0175
lubuskie	99,000	97,800	89,475	98,4650	85,0500	2,4850	0,660	29,3000	69,5175	3,7850
wielkopolskie	99,300	97,500	88,700	98,4750	86,2600	2,6900	0,660	31,0550	80,7600	4,1600
zachodniopomorskie	99,500	98,000	91,600	99,0500	86,1600	2,5200	0,670	29,7400	67,9400	3,7900
dolnośląskie	98,900	97,250	88,350	97,8725	88,8875	2,3600	0,660	29,8725	68,9450	3,6850
opolskie	99,100	97,625	90,250	98,2225	88,6225	2,5350	0,655	30,3475	71,0350	3,8275
kujawsko-pomorskie	99,000	96,200	87,800	98,0550	80,5750	2,6400	0,700	27,9050	70,8400	3,8200
pomorskie	98,800	97,800	90,050	98,4850	80,1275	2,6025	0,700	30,1825	71,0450	3,7800
warmińsko-mazurskie	99,500	98,600	91,800	99,1900	83,8500	2,5200	0,690	26,9800	67,0800	3,8100
łódzkie	98,400	95,725	88,225	97,0750	68,1050	2,4550	0,640	31,2400	72,9400	3,8400
świętokrzyskie	98,775	96,050	89,975	97,4400	81,4375	2,4100	0,640	30,2575	76,0250	3,9775

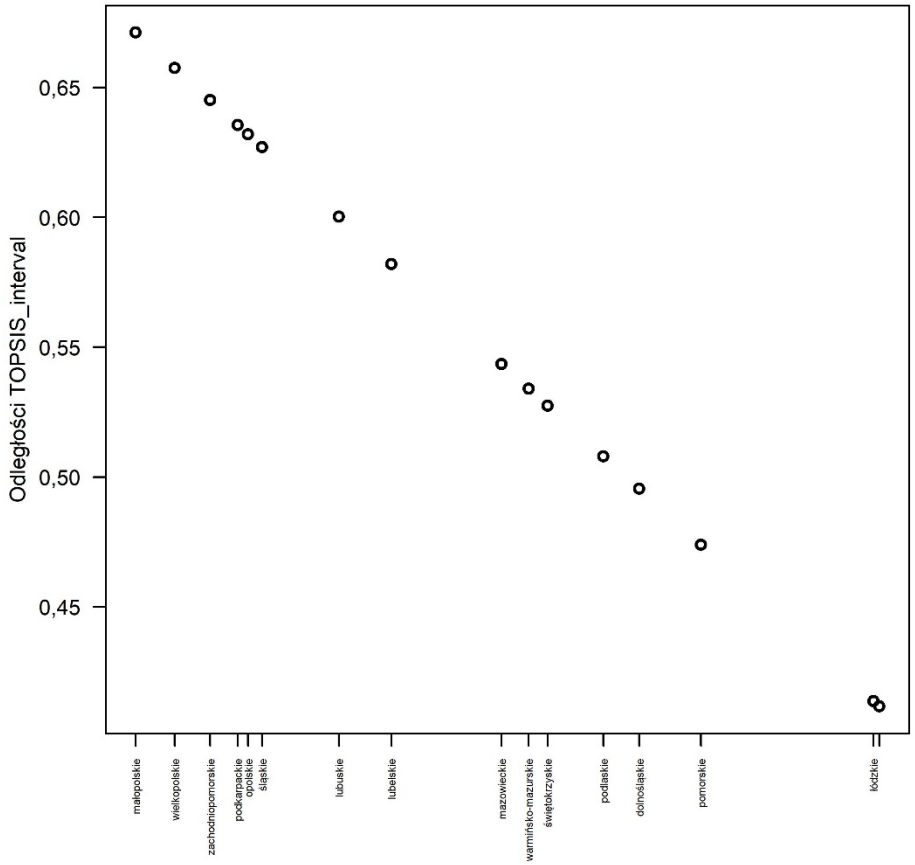
```
lubelskie          98,825 97,150 91,475 97,8750 85,2425 2,5300 0,650 31,0000 76,3100 4,0550
podkarpackie       99,000 98,000 91,800 98,6500 95,4800 2,8600 0,680 29,7800 84,8700 4,3400
podlaskie          99,200 97,700 90,700 98,3725 22,3150 2,6100 0,655 31,7525 80,2625 4,3375
mazowieckie        98,700 96,875 89,400 98,0175 80,8275 2,5000 0,630 32,2450 78,2100 4,0200
P                  99,500 98,600 91,800 99,1900 95,4800 2,3600 0,630 32,2450 84,8700 4,3400
AP                 98,400 95,725 87,800 97,0750 22,3150 2,8600 0,700 26,9800 67,0800 3,6850
```

Do uporządkowania województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r. na podstawie danych symbolicznych interwałowych (dane z plików: `dane_wm_lower_P-AP.csv` i `dane_wm_upper_P-AP.csv`) zastosowano skrypt 7.6. Jako miarę agregatową zastosowano TOPSIS z odległością Euklidesową Ichino-Yaguchiego. Przeprowadzono normalizację wartości zmiennych z zastosowaniem standaryzacji pozycyjnej (`type="n2"`). W analizie zastosowano wagi jednakowe.

Skrypt 7.6

```
library(clusterSim)
# Wczytanie danych interwałowych
x<-as.matrix(read.csv2("dane_wm_lower_P-AP.csv",row.names=1,header=T))
y<-as.matrix(read.csv2("dane_wm_upper_P-AP.csv",row.names=1,header=T))
# Normalizacja wartości zmiennych
n<-interval_normalization(x=x,y=y,dataType="separate_tables",type="n2")
# Zastosowanie miary TOPSIS_interval dla danych interwałowych
res_up<-dist.Symbolic(n$simple,type="U_2",power=2)
res_up1<-as.matrix(res_up)
licznik<-res_up1[nrow(res_up1),1:(ncol(res_up1)-2)]
TOPSIS_interval<- (licznik/(res_up1[nrow(res_up1),1:(ncol(res_up1)-2)]+
  res_up1[nrow(res_up1)-1,1:(ncol(res_up1)-2)]))
TOPSIS_interval1<-as.data.frame(TOPSIS_interval)
rownames(TOPSIS_interval1)<-rownames(x[1:16,])
print("Uporządkowanie obiektów od najlepszego do najgorszego
  wg TOPSIS_interval", quote=FALSE)
nr<-1:16
tab<-cbind(TOPSIS_interval1,nr)
wyn<-tab[order(tab,"TOPSIS_interval"],decreasing=TRUE),]
Pozycja<-1:16
wyn1<-cbind(wyn,Pozycja)
print(wyn1)
write.table(wyn1,"TOPSIS_interval.csv",dec=",",sep=";",col.names=NA,
  row.names=TRUE)
plot(cbind(TOPSIS_interval,TOPSIS_interval),xlim=c(max(TOPSIS_interval),
  min(TOPSIS_interval)),ylim=c(min(TOPSIS_interval),max(TOPSIS_interval)),
  xaxt="n",xlab="",ylab="Odległości TOPSIS_interval",lwd=1.6,las=1)
axis(1,at=TOPSIS_interval,labels=rownames(x[1:16,]),las=2,cex.axis=0.5)
```

W rezultacie zastosowania procedury ze skryptu 7.6 otrzymano następujące wyniki porządkowania liniowego województw ze względu na poziom warunków zamieszkiwania na podstawie danych symbolicznych interwałowych (graficzną formę prezentacji zamieszczono na rys. 7.2).



Rys. 7.2. Graficzna prezentacja uporządkowania województw Polski ze względu na poziom warunków zamieszkiwania ludności w miastach w 2021 r. od najlepszego do najgorszego według wartości miary `TOPSIS_interval`

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

```
[1] Uporządkowanie obiektów od najlepszego do najgorszego wg TOPSIS_interval
                  TOPSIS_interval nr Pozycja
małopolskie             0,6711615  1       1
wielkopolskie           0,6575018  4       2
zachodniopomorskie      0,6451316  5       3
podkarpackie            0,6355584 14       4
opolskie                0,6319258  7       5
śląskie                 0,6270112  2       6
lubuskie                0,6002981  3       7
lubelskie               0,5819524 13       8
mazowieckie             0,5435118 16       9
warmińsko-mazurskie     0,5340469 10      10
świętokrzyskie          0,5274383 12      11
podlaskie               0,5079906 15      12
dolnośląskie            0,4954743  6      13
pomorskie               0,4739138  9      14
łódzkie                 0,4137376 11      15
kujawsko-pomorskie      0,4116921  8      16
```

Najlepsze warunki zamieszkiwania w miastach w 2021 r. były w województwie małopolskim, najgorsze zaś w kujawsko-pomorskim. Wyniki otrzymane dla danych symbolicznych interwałowych różnią się od wyników dla danych metrycznych. Szczególnie widoczny jest spadek w rankingu województwa mazowieckiego. Dane interwałowe uwzględniają zróżnicowanie poziomu warunków zamieszkiwania w powiatach poszczególnych województw, a te są w przypadku województw zróżnicowane. Wyniki uśrednione reprezentowane przez dane metryczne uzależnione są od wartości nietypowych.

Literatura

Abrahamowicz, M. (1985). Konstrukcja syntetycznych mierników rozwoju w świetle twierdzenia Arrowa. *Prace Naukowe Akademii Ekonomicznej we Wrocławiu*, (311), 5-25.

Antczak, E. (2013). Przestrzenny taksonomiczny miernik rozwoju. *Wiadomości Statystyczne. The Polish Statistician*, *58*(7), 37-53. **https://doi.org/10.59139/ws.2013.07.3**

Bąk, A. (2018). Zastosowanie metod wielowymiarowej analizy porównawczej do oceny stanu środowiska w województwie dolnośląskim. *Wiadomości Statystyczne. The Polish Statistician*, *63*(1), 7-20. **https://doi.org/10.5604/01.3001.0014.0521**

Borys, T. (1978). Metody normowania cech statystycznych w badaniach porównawczych. *Przegląd Statystyczny*, *25*(2), 227-239.

Borys, T. (1984). Kategoria jakości w statystycznej analizie porównawczej. *Prace Naukowe Akademii Ekonomicznej we Wrocławiu*, (284), Seria: Monografie i Opracowania 23.

Borys, T., Strahl, D. i Walesiak, M. (1990). Wkład ośrodka wrocławskiego w rozwój teorii i zastosowań metod taksonomicznych. W: J. Pociecha (red.), *Taksonomia – teoria i zastosowania* (s. 12-23). Wydawnictwo Akademii Ekonomicznej w Krakowie.

Dehnel, G. i Walesiak, M. (2019). A Comparative Analysis of Economic Efficiency of Medium-sized Manufacturing Enterprises in Districts of Wielkopolska Province Using the Hybrid Approach with Metric and Interval-valued Data. *Statistics in Transition new series*, *20*(2), 49-67. **https://doi.org/10.21307/stattrans-2019-014**

Grabiński, T. (1984). *Wielowymiarowa analiza porównawcza w badaniach dynamiki zjawisk ekonomicznych*. Zeszyty Naukowe Akademii Ekonomicznej w Krakowie, Seria specjalna: Monografie nr 61.

Hellwig, Z. (1967). *Procedure of Evaluating High-Level Manpower Data and Typology of Countries by Means of the Taxonomic Method*, COM/WS/91, Warsaw, 9 December, 1967 (unpublished UNESCO working paper).

Hellwig, Z. (1968). Zastosowanie metody taksonomicznej do typologicznego podziału krajów ze względu na poziom ich rozwoju i strukturę wykwalifikowanych kadr. *Przegląd Statystyczny*, *15*(4), 307-327.

Hellwig, Z. (1972). Procedure of Evaluating High-Level Manpower Data and Typology of Countries by Means of the Taxonomic Method. W: Z. Gostkowski (red.), *Towards a System of Human Resources Indicators for Less Developed Countries* (s. 115-134). Ossolineum, The Polish Academy of Sciences Press.

Hellwig, Z. (1981). Wielowymiarowa analiza porównawcza i jej zastosowanie w badaniach wielocechowych obiektów gospodarczych. W: W. Welfe (red.), *Metody i modele ekonomiczno-matematyczne w doskonaleniu zarządzania gospodarką socjalistyczną* (s. 46-68). Polskie Wydawnictwo Ekonomiczne.

Hwang, C. L. i Yoon, K. (1981). *Multiple Attribute Decision Making – Methods and Applications. A State-of-the-Art Survey*. Springer.

Ichino, M. i Yaguchi, H. (1994). Generalized Minkowski Metrics for Mixed Feature-type Data Analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, *24*(4), 698-708. **https://doi.org/10.1109/21.286391**

Jefmański, B. i Dudek, A. (2016). Syntetyczna miara rozwoju Hellwiga dla trójkątnych liczb rozmytych. W: D. Appenzeller (red.), *Matematyka i informatyka na usługach ekonomii. Wybrane problemy modelowania i prognozowania zjawisk gospodarczych* (s. 29-40). Wydawnictwo Uniwersytetu Ekonomicznego w Poznaniu.

Lira, J. (2015). A Comparison of the Methods of Relative Taxonomy for the Assessment of Infrastructural Development of Counties in Wielkopolskie Voivodship. *Quantitative Methods in Economics*, *16*(2), 53-62.

Łysoń, P., Szymkowiak, M. i Wawrowski, Ł. (2016). Badania porównawcze atrakcyjności turystycznej powiatów z uwzględnieniem ich otoczenia. *Wiadomości Statystyczne. The Polish Statistician*, *12*(667), 45-57. **https://doi.org/10.5604/01.3001.0014.1117**

Mazziotta, M. i Pareto, A. (2016). On a Generalized Non-compensatory Composite Index for Measuring Socio-economic Phenomena. *Social Indicators Research*, *127*(3), 983-1003. **https://doi.org/10.1007/s11205-015-0998-2**

Mazziotta, M. i Pareto, A. (2018). Measuring Well-being Over Time: The Adjusted Mazziotta-Pareto Index Versus Other Non-compensatory Indices. *Social Indicators Research*, *136*(3), 967-976. **https://doi.org/10.1007/s11205-017-1577-5**

Młodak, A. (2014). On the Construction of an Aggregated Measure of the Development of Interval Data. *Computational Statistics*, *29*(5), 895-929. **https://doi.org/10.1007/s00180-013-0469-7**

Perkal, J. (1967). *Matematyka dla przyrodników i rolników. Część II*. PWN.

Pietrzak, M.B. (2014). Taksonomiczny miernik rozwoju (TMR) z uwzględnieniem zależności przestrzennych. *Przegląd Statystyczny*, *61*(2), 181-201.

Pociecha, J. i Zając, K. (1990). Wkład ośrodka krakowskiego w rozwój teorii i zastosowań metod taksonomicznych. W: J. Pociecha (red.), *Taksonomia – teoria i zastosowania* (s. 24-32). Wydawnictwo Akademii Ekonomicznej w Krakowie.

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Sokołowski, A. i Markowska, M. (2019). *Elastyczne porządkowanie liniowe obiektów*. XXXIII Konferencja Taksonomiczna nt. „Klasyfikacja i analiza danych – teoria i zastosowania" (Szczecin, 18-20 września 2019, Uniwersytet Szczeciński).

Walesiak, M. (1993). Statystyczna analiza wielowymiarowa w badaniach marketingowych. *Prace Naukowe Akademii Ekonomicznej we Wrocławiu*, (654), Seria: Monografie i Opracowania 101.

Walesiak, M. (2002). *Uogólniona miara odległości w statystycznej analizie wielowymiarowej*. Wydawnictwo Akademii Ekonomicznej we Wrocławiu.

Walesiak, M. (2014). Wzmacnianie skali pomiaru w statystycznej analizie wielowymiarowej. *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, (327). Taksonomia 22, 60-68.

Walesiak, M. (2016a). *Uogólniona miara odległości GDM w statystycznej analizie wielowymiarowej z wykorzystaniem programu R* (wyd. 2 popr. i rozsz.). Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu.

Walesiak, M. (2016b). Visualization of Linear Ordering Results for Metric Data with the Application of Multidimensional Scaling. *Ekonometria. Econometrics*, *2*(52), 9-21. **https://doi.org/10.15611/ekt.2016.2.01**

Walesiak, M. (2018). The Choice of Normalization Method and Rankings of the Set of Objects Based on Composite Indicator Values. *Statistics in Transition new series*, *19*(4), 693-710. **https://doi.org/10.21307/stattrans-2018-036**

Walesiak, M. i Dehnel, G. (2018). Evaluation of Economic Efficiency of Small Manufacturing Enterprises in Districts of Wielkopolska Province Using Interval-valued Symbolic Data and the Hybrid Approach. W: M. Papież i S. Śmiech (red.), *The 12th Professor Aleksander Zeliaś International Conference on Modelling and Forecasting of Socio-Economic Phenomena. Conference Proceedings* (s. 563-572). Foundation of the Cracow University of Economics. **https://doi.org/10.14659/SEMF.2018.01.57**

Walesiak, M. i Dehnel, G. (2022). A Dynamic Approach to Relative Taxonomy in the Assessment of Changes in the Social Cohesion of Polish Provinces in 2010-2018. *Argumenta Oeconomica*, *1*(48), 37-65. **https://doi.org/10.15611/aoe.2022.1.02**

Walesiak, M., Dehnel, G. i Dudek, A. (2024). A Dynamic Approach to Relative Taxonomy and Robust Measures of Central Tendency. *Communications in Statistics – Simulation and Computation*, *53*(6), 2645-2661. **https://doi.org/10.1080/03610918.2022.2083163**

Walesiak, M. i Dudek, A. (2023). *clusterSim: Searching for Optimal Clustering Procedure for a Data Set*. R package version 0.51-3. **https://cran.r-project.org/web/packages/clusterSim/**

Wydymus, S. (2013). Rozwój gospodarczy a poziom wynagrodzeń w krajach Unii Europejskiej – analiza taksonomiczna. *Zeszyty Naukowe Uniwersytetu Szczecińskiego*, (756). Finanse, Rynki Finansowe, Ubezpieczenia, (57), 631-645.

Wysocki, F. (2010). *Metody taksonomiczne w rozpoznawaniu typów ekonomicznych rolnictwa i obszarów wiejskich*. Wydawnictwo Uniwersytetu Przyrodniczego w Poznaniu.

DOI: 10.15611/2024.45.1.08

8 Drzewa klasyfikacyjne i regresyjne

8.1. Metoda rekurencyjnego podziału

Drzewa klasyfikacyjne i regresyjne (*Classification And Regression Trees* – CART) to nazwa jednej z najważniejszych metod budowy modeli matematycznych pozwalających wyjaśnić zachowanie złożonych zjawisk społecznych i ekonomicznych. Ma zastosowanie we wszystkich systemach uczenia maszynowego (*machine learning*), sztucznej inteligencji (*artificial intelligence*) czy w systemach BigData.

Jej podstawowymi zaletami są brak założeń co do rozkładów zmiennych objaśniających (jest przykładem metody nieparametrycznej) oraz sekwencyjny sposób budowy modelu

$$Y = f(X_1, \ldots, X_L) + \varepsilon \tag{8.1}$$

połączony z doborem zmiennych. Należy także dodać, że może być stosowana w przypadku dysponowania bardzo dużymi zbiorami danych, w których występują braki danych czy wartości oddalone.

Jej nazwa pochodzi od graficznej reprezentacji modelu, który ilustruje jednocześnie rekurencyjny podział (*recursive partitioning*) L-wymiarowej przestrzeni zmiennych na podprzestrzenie P_k, aż do chwili, gdy zmienna zależna Y osiągnie w każdej z nich minimalny poziom zróżnicowania.

Ten sposób budowy modelu globalnego (8.1) polegający na złożeniu modeli lokalnych, zbudowanych w poszczególnych podprzestrzeniach P_k, był stosowany w statystyce już w latach 60. przez Morgana i Sonquista (1963). Natomiast po raz pierwszy wnikliwą, formalną analizę jej własności oraz wykorzystanie w analizie dyskryminacyjnej i regresji przedstawili Breiman i in. (1984) w słynnej książce *Classification and Regression Trees*.

W ramach omawianej metody modele lokalne w każdej z K podprzestrzeni mają najprostszą postać, czyli wartość stałą:

$$f(\mathbf{x}_i) = \sum_{k=1}^{K} \alpha_k I(\mathbf{x}_i \in P_k), \tag{8.2}$$

gdzie: P_k $(k = 1, \ldots, K)$ to podprzestrzenie (segmenty) przestrzeni $\mathbf{X}^L$, α_k – parametry modelu, I – funkcja wskaźnikowa, mająca postać:

$$I(q) = \begin{cases} 1 & \text{gdy } q \text{ jest prawdziwe} \\ 0 & \text{w przeciwnym wypadku.} \end{cases} \tag{8.3}$$

Każdy z segmentów P_k jest definiowany poprzez jego granice w przestrzeni $\mathbf{X}^L$, które dla zmiennych metrycznych $X_1, \ldots, X_L$ można przedstawić jako:

$$I(\mathbf{x}_i \in P_k) = \prod_{l=1}^{L} I\left(v_{kl}^{(d)} \leq x_{il} \leq v_{kl}^{(g)}\right), \tag{8.4}$$

gdzie: wartości $v_{kl}^{(d)}$ oraz $v_{kl}^{(g)}$ – odpowiednio jego górna i dolna granica w l-tym wymiarze przestrzeni.

W przypadku gdy zmienne $X_1, \ldots, X_L$ mają charakter jakościowy, podprzestrzeń P_k można zdefiniować jako:

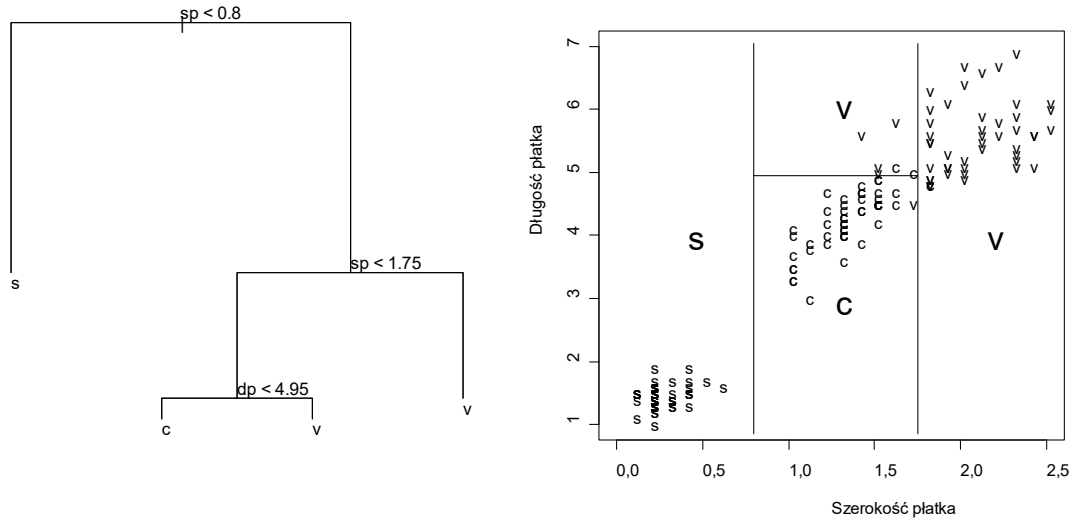
$$I(\mathbf{x}_i \in P_k) = \prod_{l=1}^{L} I(x_{il} \in B_{kl}), \tag{8.5}$$

gdzie: B_{kl} – podzbiór zbioru kategorii zmiennej X_l, tj. $B_{kl} \subseteq V_l$.

Jeżeli zmienna zależna Y w modelu (8.1) jest zmienną nominalną (jakościową), to model ten nazywany jest dyskryminacyjnym i reprezentuje go drzewo klasyfikacyjne. Wtedy parametry α_k modelu (8.2) są wyznaczane jako:

$$\alpha_k = \arg\max_j p(C_j | \mathbf{x}_i \in P_k). \tag{8.6}$$

Przykład modelu dyskryminacyjnego w postaci drzewa klasyfikacyjnego znajduje się na rys. 8.1, wraz z odpowiednim podziałem przestrzeni.



Rys. 8.1. Drzewo klasyfikacyjne oraz odpowiedni podział przestrzeni zmiennych

Źródło: (Gatnar, 2001).

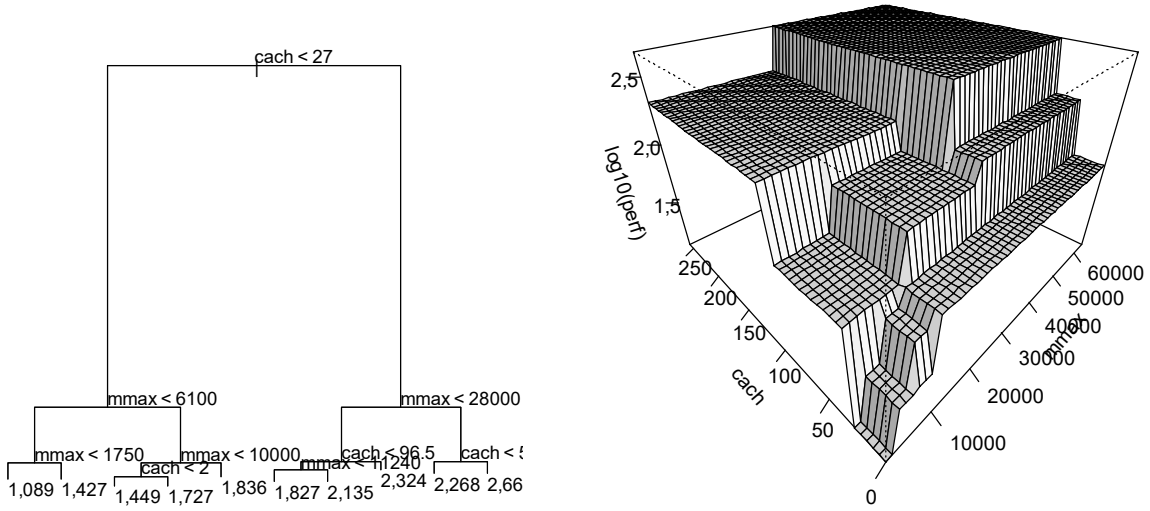
W przypadku gdy zmienna zależna Y jest zmienna metryczną (ilościową), model (8.1) jest modelem regresji, którego graficzną postacią jest drzewo regresyjne. Jego parametry są wyznaczane za pomocą formuły

$$\alpha_k = \frac{1}{N(k)} \sum_{x_i \in P_k} y_i, \tag{8.7}$$

gdzie: $N(k)$ – liczba obserwacji należących do segmentu P_k.

Innymi słowy, parametr ten jest wartością średnią zmiennej Y dla obserwacji znajdujących się w podprzestrzeni P_k.

Przykład modelu dla dwóch zmiennych objaśniających w postaci drzewa regresyjnego znajduje się na rys. 8.2, wraz z odpowiednim podziałem przestrzeni.



Rys. 8.2. Drzewo regresyjne i odpowiedni podział przestrzeni zmiennych

Źródło: (Gatnar, 2008).

Modele w postaci drzew klasyfikacyjnych i regresyjnych mają wiele zalet. Najważniejszą z nich jest łatwość interpretacji modelu w języku naturalnym. Sekwencja podziałów od korzenia drzewa do liścia generuje bowiem reguły koniunkcyjne w postaci:

$$\text{jeżeli } x > 0{,}8 \wedge z < 1{,}75 \wedge u < 4{,}95, \text{ to obiekt należy do klasy C.}$$

Mogą one także klasyfikować dane niepełne, tj. zawierające obserwacje, dla których nie można określić wartości pewnych zmiennych. Problem ten pojawia się np. wtedy, gdy te wartości są trudne do zmierzenia. Metoda ta jest również odporna na występowanie wartości nietypowych[1].

8.2. Funkcja kryterium podziału

Ogólna funkcja oceny jakości podziału przestrzeni P na podprzestrzenie $P_1, \ldots, P_K$ ma postać:

[1] W pracy Gatnara (2001) cały rozdział 8 poświęcono omówieniu metod wykorzystania omawianych tutaj modeli w przypadku zbiorów danych zawierających braki wartości zmiennych oraz obserwacje nietypowe.

$$\Delta Q(P) = Q(P) - \sum_{k=1}^{K} Q(P_k)p(k), \tag{8.8}$$

gdzie: $p(k)$ – frakcja obserwacji w P_k.

Kryterium (8.8) podlega maksymalizacji, tj. szukany jest taki podział przestrzeni P, który zapewni jak największą jednorodność uzyskanych podprzestrzeni P_k, dla $k = 1, \ldots, K$.

Ocena homogeniczności zarówno przestrzeni P, jak i podprzestrzeni P_k dokonywana jest za pomocą funkcji Q, której postać zależy od rodzaju budowanego modelu. W przypadku gdy model (8.1) jest modelem regresyjnym, funkcja Q jest funkcją kwadratową:

$$Q(P_k) = \frac{1}{N(k)} \sum_{\boldsymbol{x}_i \in P_k} (y_i - \alpha_k)^2, \tag{8.9}$$

gdzie wartość α_k jest wyznaczana za pomocą formuły (8.7).

Jeśli jednak zmienna Y jest zmienną nominalną, to zamiast miary (8.9) można wykorzystać błąd klasyfikacji (*misclassification error*):

$$Q(P_k) = 1 - \arg\max_j p(C_j|P_k) \tag{8.10}$$

albo wskaźnik Giniego (*Gini index*):

$$Q(P_k) = 1 - \sum_{j=1}^{J} \big(p(C_j|P_k)\big)^2, \tag{8.11}$$

albo miarę entropii:

$$Q(P_k) = -\sum_{j=1}^{J} p(C_j|P_k) \log_2 p\,(C_j|P_k). \tag{8.12}$$

Omówienie własności przedstawionych miar oraz charakterystyka innych, nieco mniej znanych, znajduje się w pracy Gatnara (2001).

Jeśli chodzi o wybór miary oceny homogeniczności podprzestrzeni, to Breiman i in. (1984) preferują stosowanie wskaźnika Giniego (8.11). Ma on jednak pewną wadę, ponieważ osiąga maksimum, gdy podprzestrzenie P_k zawierają jednakową liczbę obserwacji. Quinlan (1993) zaś w swoim algorytmie C4.5 stosuje entropię (8.12), która z kolei preferuje taki podział, który generuje maksymalną liczbę segmentów R_k.

Podział przestrzeni P na podprzestrzenie, gdy zmienne $X_1, \ldots, X_L$ są zmiennymi metrycznymi, w większości metod odbywa się za pomocą hiperpłaszczyzn równoległych do osi. Równanie takiej hiperpłaszczyzny ma wtedy postać $X_l = c$, gdzie zarówno wybór zmiennej X_l, jak i wartości c kontroluje miara (8.8).

Aby wyznaczyć stałą c, należy obliczyć wartość kryterium (8.8) dla wszystkich możliwych wariantów położenia punktu podziału zbioru wartości $V_l = \{v_{l1}, \ldots, v_{lT}\}$ zmiennej X_l:

$$c = \frac{v_{lt} + v_{lt+1}}{2}. \tag{8.13}$$

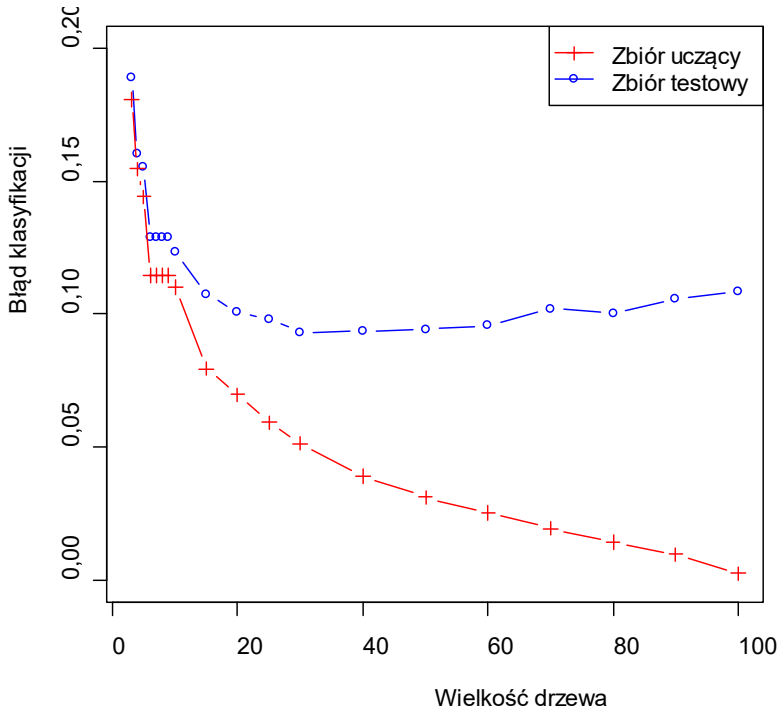
Uzyskuje się w ten sposób dwa zbiory obserwacji: $\{\mathbf{x}_i: x_{il} \leq c\}$ oraz $\{\mathbf{x}_i: x_{il} > c\}$. Inaczej mówiąc, dokonywana jest w ten sposób dyskretyzacja zmiennej X_l, czego rezultatem jest powstanie drzewa binarnego, w którym z każdego węzła wychodzą dwie krawędzie.

Należy wyjaśnić, że w procesie budowy modelu w postaci drzewa najpierw każda zmienna metryczna poddawana jest dyskretyzacji, a następnie wybierana jest ta spośród nich, dla której kryterium (8.8) osiąga maksimum.

Jeżeli zmienna X_l ma charakter niemetryczny, to zbiór jej kategorii $V_l = \{v_{l1}, \ldots, v_{lT}\}$ jest dzielony na dwa podzbiory (w przypadku drzewa binarnego), tak aby wartość kryterium (8.8) była jak największa (takich podziałów jest 2^T dla zmiennych porządkowych oraz $2^{T-1} - 1$ dla zmiennych nominalnych). Najczęściej punktem wyjścia jest podział V_l na T podzbiorów $\{v_{l1}\}, \ldots, \{v_{lT}\}$, a następnie te podzbiory są stopniowo łączone. W metodzie CHAID, którą zaproponował Kass (1980), tym procesem łączenia steruje statystyka χ^2.

8.3. Optymalna wielkość modelu

W przypadku modeli w postaci drzew klasyfikacyjnych i regresyjnych pojawia się problem wyboru takiej postaci modelu, by jego błąd predykcji był jak najmniejszy. Podobnie jak w przypadku innych metod, także tutaj wraz ze złożonością modelu (wielkością drzewa) błąd ten dla zbioru uczącego maleje do zera, dla zbioru testowego zaś – rośnie. Pokazuje to rys. 8.3, na którym widać wyraźnie, że najmniejszy błąd klasyfikacji dla zbioru testowego występuje dla drzewa liczącego 27 liści i wynosi 9,13%.



Rys. 8.3. Błąd klasyfikacji dla zbiorów uczącego i testowego

Źródło: (Gatnar, 2008).

Powstaje więc problem zatrzymania procesu podziału przestrzeni zmiennych P przed osiągnięciem pełnej homogeniczności podprzestrzeni P_k (w skrajnym przypadku w każdym liściu drzewa może znajdować się jedna obserwacja). Taki model bowiem, jak widać na rys. 8.3, jest idealnie dopasowany do zbioru uczącego (*overfits data*), ma jednak niewielką wartość predykcyjną.

Wśród metod wykorzystywanych w celu wyeliminowania tego zjawiska i zmniejszenia stopnia złożoności modelu, najczęściej[2] stosuje się tzw. przycinanie krawędzi (*pruning*). Zabieg ten powoduje redukcję wielkości drzewa przez usunięcie niektórych jego fragmentów, co może oznaczać, że z modelu zostaną wyeliminowane niektóre zmienne.

Breiman i in. (1984) zaproponowali pewną formę regularyzacji, która pozwala uzyskać kompromis między złożonością modelu i jego jakością (*cost-complexity pruning*) w postaci kryterium

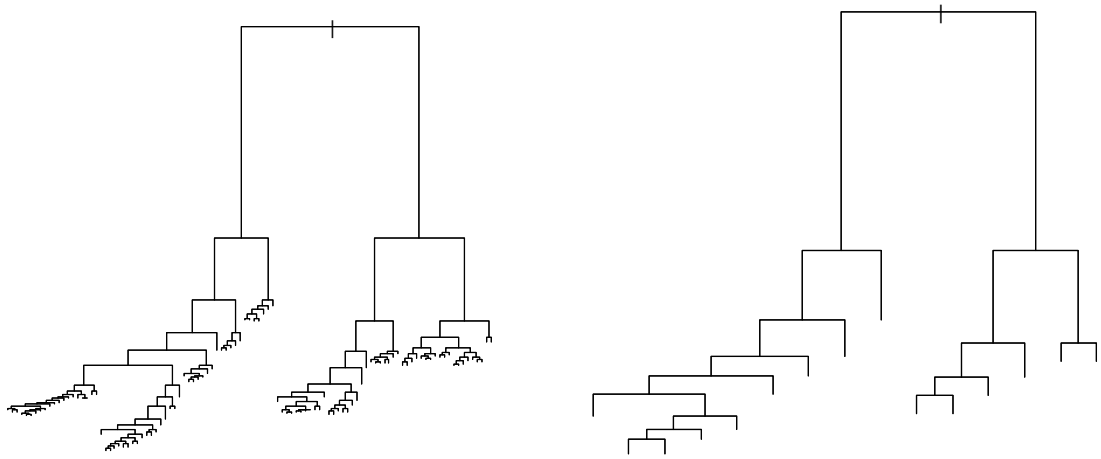
$$S_\lambda(D) = R(D) + \lambda \cdot K, \tag{8.14}$$

które podlega minimalizacji. W powyższej formule $R(D)$ to jakość modelu w postaci drzewa D:

$$R(D) = \sum_{k=1}^{K} Q(P_k)p(k), \tag{8.15}$$

gdzie $Q(P_k)$ zostało zdefiniowane za pomocą formuł (8.9)-(8.12), K oznacza liczbę liści i jest oceną złożoności modelu, λ zaś to tzw. parametr złożoności ($\lambda \geq 0$). Duże wartości parametru λ oznaczają podział na niewiele segmentów (małe drzewa), a małe wartości – drzewa bardziej rozbudowane, o dużej liczbie liści. W przypadku gdy $\lambda = 0$, powstaje drzewo maksymalne (pełne) D_0.

Z prawej strony rys. 8.4 znajduje się drzewo z przyciętymi krawędziami zgodnie z formułą (8.14) dla wartości parametru $\lambda = 25{,}0$. Ilustruje ono podział przestrzeni zmiennych na 12 segmentów i klasyfikuje błędnie 144 obserwacje ze zbioru uczącego, co oznacza błąd równy 9,4%.



Rys. 8.4. Drzewo pełne oraz drzewo przycięte ($\lambda = 25{,}0$)

Źródło: (Gatnar, 2008).

[2] Gatnar (2001) omawia także rzadziej stosowaną metodę skracania krawędzi drzewa (*shrinking*), które są proporcjonalne do stopnia homogeniczności w węzłach.

Proces przycinania drzewa przebiega w ten sposób, że najpierw budowane jest drzewo pełne D_0, a następnie, w kolejnych krokach, przycinane są te gałęzie, których brak nie powoduje zbyt dużego pogorszenia jakości modelu (w sensie miary (8.15)). Powstaje więc sekwencja drzew:

$$D_0, D_1, \ldots, D_t, D_{t+1} \ldots, D_T, \tag{8.16}$$

gdzie D_T oznacza drzewo składające się z korzenia (*root tree*). Każde z nich jest optymalne ze względu na wartość parametru $\lambda \in \{\lambda_t, \lambda_{t+1}\}$, zgodnie z formułą

$$\lambda_t = \frac{R(D_{t+1}) - R(D_t)}{K_t - K_{t+1}}, \tag{8.17}$$

gdzie K_t oznacza liczbę liści drzewa D_t. Optymalne drzewo wybrane z sekwencji (8.16) to drzewo, które daje najmniejszą wartość błędu klasyfikacji.

Breiman i in. (1984) proponują jednak wybierać drzewo optymalne jako najmniejsze z drzew, które leżą w odległości jednego odchylenia standardowego od tego drzewa, które generuje najmniejszą wartość miary R (*one SE rule*):

$$D^{opt} = \arg\min_{K_t}(R(D_t) + \sigma), \tag{8.18}$$

gdzie σ oznacza odchylenie standardowe miary $R(D_t)$.

W literaturze znane są także inne metody przycinania drzew klasyfikacyjnych i regresyjnych. Należy do nich metoda przycinania według najmniejszego błędu (*minimal error pruning*), która daje w efekcie model klasyfikujący obiekty z najmniejszym błędem klasyfikacji. Dla tej metody przycinania, w formule (8.18) zamiast $Q(D_t)$ stosuje się błąd klasyfikacji $e(D_t)$, który może być szacowany albo na podstawie zbioru testowego, albo za pomocą sprawdzania krzyżowego. Zamiast jednego sprawdzania krzyżowego można też dokonać kilku i uśrednić wyniki.

Kolejna metoda polega na przycinaniu według wielkości błędu klasyfikacji uzyskanego na podstawie zbioru testowego (*reduced error pruning*). Modyfikacja poprzedniej metody, która nie wymaga posiadania zbioru testowego, nazywana jest przycinaniem na podstawie pesymistycznej oceny błędu (*pessimistic error pruning*). Nazwa wynika stąd, że błąd dla zbioru uczącego ($e^U(D_t)$) daje zwykle zaniżoną wartość błędu klasyfikacji (optymistyczną), dlatego należy stosować poprawkę zaproponowaną przez Quinlana (1986):

$$e^{RC}(D_t) = \frac{1}{N}\left[\sum_{k=1}^{K} B(P_k) + \frac{K}{2}\right], \tag{8.19}$$

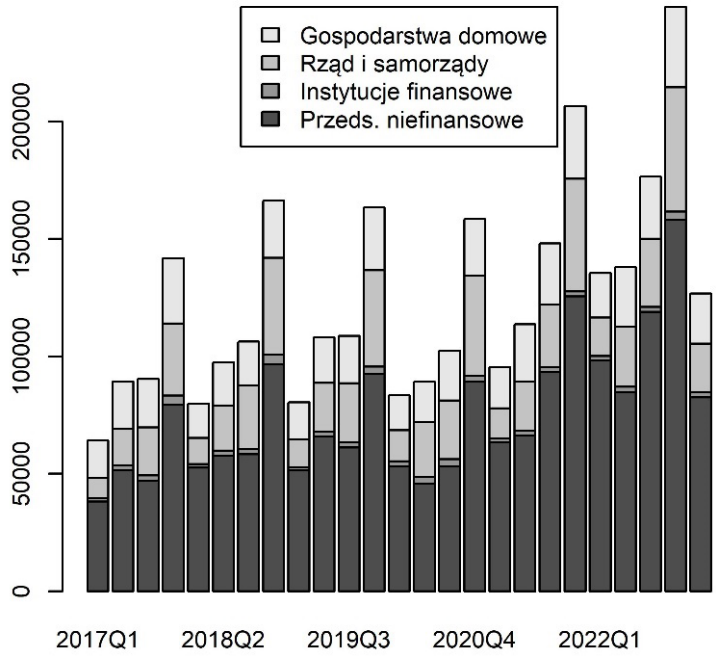
gdzie $B(P_k)$ oznacza liczbę obserwacji błędnie zaklasyfikowanych do P_k.

8.4. Zastosowania z wykorzystaniem programu R dla danych BDL

Modele drzew regresyjnych zostaną wykorzystane do analizy wpływu środków pochodzących z różnych źródeł na finansowanie inwestycji publicznych przez jednostki samorządu terytorialnego (JST) na poziomie województw (NUTS2) w latach 2005-2022.

Problem finansowania inwestycji w Polsce jest ważny. Oprócz konsumpcji, jest to najistotniejszy czynnik wzrostu PKB w naszym kraju. Ostatnie dane GUS pokazują, że inwestycje w Polsce wzrosły w 2022 r. o 4,6%, co jest bardzo dobrym wynikiem, ponieważ rok wcześniej wzrost wyniósł jedynie 2,1%. Stopa inwestycji (udział inwestycji w PKB) osiągnęła poziom 16,8%, tylko nieco poniżej wyniku z poprzedniego roku (17,0%).

Struktura nakładów inwestycyjnych w podziale na sektory instytucjonalne została pokazana na rys. 8.5. Mówiąc dokładnie, pokazano nakłady brutto na środki trwałe, zgodnie z metodologią ESA2010 w ujęciu kwartalnym w okresie od I kwartału 2017 do I kwartału 2023. Należy tutaj podkreślić, że pojęcie nakładów brutto na środki trwałe jest szersze niż pojęcie nakładów inwestycyjnych.



Rys. 8.5. Nakłady brutto na środki trwałe w Polsce w ujęciu kwartalnym (mln zł)

Źródło: obliczenia własne na podstawie danych GUS.

Jak widać na rys. 8.5, nakłady na inwestycje charakteryzują się silną sezonowością, związaną z faktem rozliczania się jednostek w IV kwartale każdego roku. Poza tym, duży wpływ na rozkład inwestycji jednostek samorządu terytorialnego ma cykl polityczny (czego ten wykres dobrze nie pokazuje), czyli wybory samorządowe przeprowadzone w latach 2014, 2018 i 2024, które zostały przesunięte ze względu na pandemię wirusa Sars-Cov-2.

Jak pokazują analizy zmian nakładów inwestycyjnych w poszczególnych sektorach w latach 2015--2019, związek przyczynowo-skutkowy jest taki, że jako pierwsze, po słabym 2016 r., ruszyły inwestycje publiczne i one pociągnęły następnie inwestycje prywatne polskich przedsiębiorstw. W rezultacie, to one odpowiadały w II kwartale 2019 r. za ok. połowę nakładów inwestycyjnych w Polsce. Rok 2020 to z kolei rok kryzysu pandemicznego, gdy, zwłaszcza w pierwszej jego połowie, aktywność inwestycyjna wszystkich sektorów była mocno stłumiona. Po kryzysie nastąpiło odbicie PKB, stymulowane głównie wzrostem nakładów inwestycyjnych polskich firm. Tak więc nakłady

brutto na środki trwałe po spadku w I kwartale 2021 r. o 1,2%, w kolejnych rosły odpowiednio o 5,8, 6,3 oraz 5,3% r/r.

W pierwszej połowie 2022 r. dynamika wydatków inwestycyjnych rosła zarówno w I, jak i w II kwartale odpowiednio o 13,0 oraz 16,9% r/r. Zdecydowane przyspieszenie widoczne było zwłaszcza w sektorze publicznym, gdzie w obu kwartałach wzrost inwestycji przekroczył poziom 20% r/r. Dużą aktywnością cechowały się również przedsiębiorstwa, których nakłady inwestycyjne wzrosły o 11,9% w I kwartale oraz o 21,5% w II kwartale. Natomiast w sektorze gospodarstw domowych aktywność inwestycyjna została stłumiona z powodu zacieśnienia polityki monetarnej i związanego z tym spadku popytu na kredyty hipoteczne wspierające zakup nieruchomości.

Jak widać na rys. 8.5, w rekordowym IV kwartale 2022 r. łączne nakłady brutto na środki trwałe w naszej gospodarce wyniosły prawie 249 mld zł, z czego największą część, bo aż 64% stanowiły nakłady przedsiębiorstw niefinansowych. Warto podkreślić, że polskie firmy, wspierane tarczami Polskiego Funduszu Rozwoju, w stosunkowo dobrej kondycji przetrwały kryzys pandemiczny, gromadząc środki na rozwój po ustaniu pandemii wirusa Sars-Cov-2.

Bardzo ważnym źródłem finansowania inwestycji publicznych, realizowanych głównie przez jednostki samorządu terytorialnego, są środki Unii Europejskiej. Polska właśnie obchodziła 19. rocznicę wstąpienia do Unii Europejskiej. Z członkostwem w UE wiążą się transfery środków pomocowych w ramach dwóch małych (SAPARD i PHARE) oraz dwóch dużych programów (nazywanych perspektywami). Do tej pory zakończyły się perspektywa 2007-2013 oraz perspektywa 2014-2020. Wykorzystanie środków europejskich jest zwykle opóźnione o dwa lata, a z powodu pandemii rozliczenie tej ostatniej perspektywy potrwało do końca 2023 r.

Jeśli chodzi o udział środków UE w finansowaniu inwestycji publicznych, to w zakończonej perspektywie 2007-2020 szacuje się go na 30%. Warto zatem zweryfikować hipotezę o tym, jakie znaczenie mają te środki w finansowaniu inwestycji przez województwa w Polsce w latach 2005-2022.

W bazie danych BDL znajduje się wartości nakładów na środki trwałe w ujęciu gmin, powiatów i województw. Jednak województwa w ostatnich latach miały największy lub równy gminom udział w podziale środków z UE. Rozkład wartości wydatków inwestycyjnych w 2022 r. był w układzie województw dosyć nierównomierny, z wyjątkową pozycją województwa mazowieckiego, co pokazano na rys. 8.6.

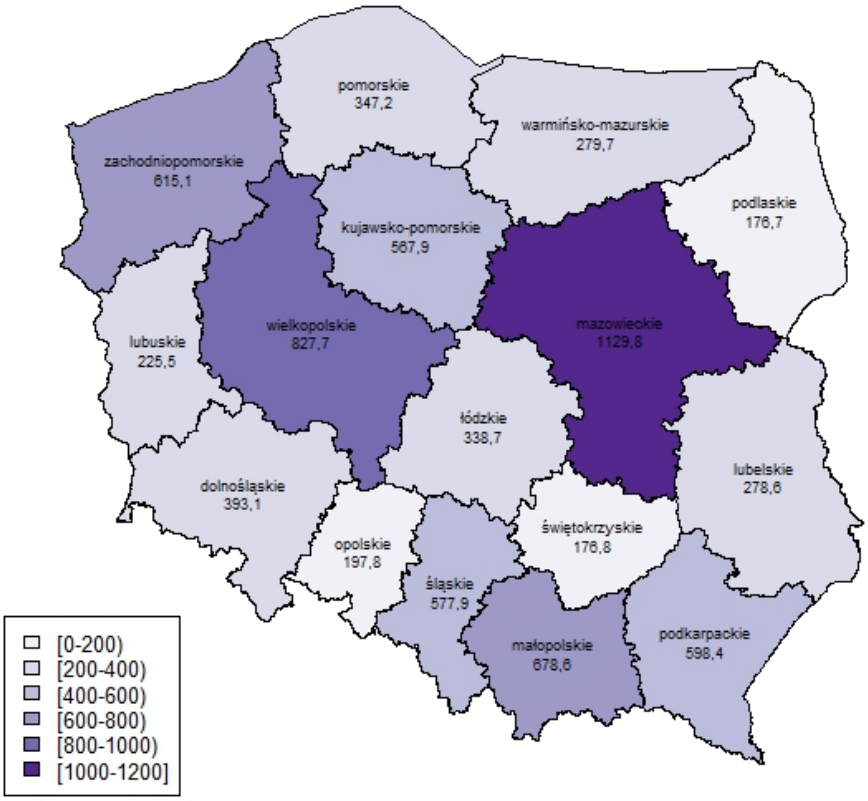
Aby taki wykres uzyskać, należy wykonać polecenia pokazane w skrypcie 8.1.

Skrypt 8.1

```
library(sf)
library(classInt)
library(RColorBrewer)
library(bdl)
library(dplyr)
options(OutDec=",")
# Wczytanie informacji o zmiennej z BDL
dane<-get_data_by_variable("6478",unitParentId=NULL,unitLevel=2,year=2022) %>%
  dplyr::select(name, x=val)
x<-as.data.frame(dane)
```

```
# Nadanie nazw kolumnom
colnames(x)<-c("województwo","Inwestycje")
# wczytanie pliku shapefile dla województw
regiony<-read_sf(dsn="A01_Granice_wojewodztw.shp")
# Uporządkowanie województw według kolejności pliku shp
if(!is.null(regiony$JPT_NAZWA_)){
  w<-factor(regiony$JPT_NAZWA_)
}
nazwy<-data.frame(województwo=w)
x<-mutate(x,województwo=factor(tolower(województwo),levels=levels(w)))
xx<-inner_join(nazwy,x)
row.names(xx)<-xx$województwo
# Zmienna do prezentacji na mapie
zmienna<-round(xx$Inwestycje/1000000,1)
# Przedziały klasowe i kolory
przedzialy<-6
kolory<-brewer.pal(przedzialy, name="Purples")
klasy<-classIntervals(zmienna, przedzialy, style="fixed", cutlabels=FALSE,
  fixedBreaks=c(0,200,400,600,800,1000,1200),dataPrecision=1)
tabela.kolorow<-findColours(klasy, kolory)
# Mapa
windows()
plot(regiony$geometry, col=tabela.kolorow)
title(main="Wydatki majątkowe inwestycyjne w mln zł\n według województw
 w 2022 r.")
# Legenda
leg=names(attr(tabela.kolorow, "table"))
leg=gsub(",", "-", leg)
#leg=gsub("\\.", ",", leg)
legend("bottomleft", legend=leg, fill=attr(tabela.kolorow, "palette"),
  cex=0.8, bty="o")
# Etykiety
centroid<-st_centroid(regiony$geometry)
library(geometries)
df<-gm_coordinates(centroid)
crds<-df[,3:4]
nazwy<-row.names(xx)
wydatki<-zmienna
zmienna<-gsub("\\.", ",", zmienna)
for(i in 1:16){
text(crds[i,1], crds[i,2], labels=paste(nazwy[i],paste(wydatki[i],"",sep=""),
  sep="\n"), cex=0.6)}
```

Zmienna `6478` w bazie BDL to właśnie wydatki majątkowe inwestycyjne, poziom województw zaś określa parametr `unitLevel=2`. Zmienna ta w modelu, który zostanie przygotowany w dalszej części rozdziału, będzie odgrywała rolę zmiennej zależnej `y`.



Rys. 8.6. Nakłady na inwestycje w 2022 r. w układzie województw (mln zł)

Źródło: obliczenia własne na podstawie danych BDL.

Finansowanie inwestycji samorządowych, w tym województw, odbywa się w oparciu o środki własne, dotacje z budżetu państwa oraz te, które pozyskano z Unii Europejskiej. Jako zmienne objaśniające do modelu będą służyły:

- x1 – zmienna 6455, czyli dochody własne razem,
- x2 – zmienna 6464, czyli dotacje z budżetu państwa,
- x3 – zmienna 199171, czyli środki z Unii Europejskiej,
- x4 – zmienna 6484, czyli wydatki na obsługę długu,
- x5 – zmienna 202522, czyli zadłużenie.

W przypadku wszystkich powyższych zmiennych należy zmienić jednostki, aby wyrazić je w milionach złotych.

Dodatkowo, w związku z obowiązywaniem w latach 2004-2017 indywidualnych wskaźników zadłużenia (Sekuła i Śmiechowicz, 2018), do modelu należy wprowadzić jeszcze dwie zmienne zero-jedynkowe:

- x6 – wskaźnik (x4/x1) < 0,6 oznacza, że zadłużenie nie może przekraczać 60% dochodów własnych,
- x7 – wskaźnik (x5/x1) < 0,15 oznacza, że wydatki na obsługę długu nie mogą przekraczać 15% dochodów województwa.

Należy rozpocząć od pozyskania danych charakteryzujących wymienione zmienne z Banku Danych Lokalnych, co pokazuje skrypt 8.2.

Skrypt 8.2

```
library(bdl)
library(tidyverse)
options(OutDec=",")
# stałe
rok<-c(2005:2022)
no_uL=2 # (2 - województwo)
# Wczytanie informacji o zmiennych z BDL
d1<-get_data_by_variable("6478",unitParentId=NULL, unitLevel=no_uL,year=rok) %>%
  select(id,name,year,x1=val)
d2<-get_data_by_variable("6455",unitParentId=NULL, unitLevel=no_uL,year=rok) %>%
  select(id,name,year,x2=val)
d3<-get_data_by_variable("6464",unitParentId=NULL, unitLevel=no_uL,year=rok) %>%
  select(id,name,year,x3=val)
d4<-get_data_by_variable("199171",unitParentId=NULL,
  unitLevel=no_uL,year=rok) %>%
  select(id,name,year,x4=val)
d5<-get_data_by_variable("6484",unitParentId=NULL, unitLevel=no_uL,year=rok) %>%
  select(id,name,year,x5=val)
d6<-get_data_by_variable("202522",unitParentId=NULL,
  unitLevel=no_uL,year=rok) %>%
  select(id,name,year,x6=val)
# Połączenie zmiennych w jedną tabelę (ramkę) danych
data <- d1 %>%
  full_join (d2, by=c("id","name","year")) %>%
  full_join (d3, by=c("id","name","year")) %>%
  full_join (d4, by=c("id","name","year")) %>%
  full_join (d5, by=c("id","name","year")) %>%
  full_join (d6, by=c("id","name","year"))
d<-as.data.frame(data)
# Dodanie 2 pustych na przeliczenia
for(i in 1:2) d<-cbind(d,rep(0,nrow(d)))
d<-d[,4:11]
# Nadanie nazw kolumnom
colnames(d)<-c("d1","d2","d3","d4","d5","d6","d7","d8")
# Przeliczenia
d<-mutate(d,d1=round((d1/1000000),2))
d<-mutate(d,d2=round((d2/1000000),2))
d<-mutate(d,d3=round((d3/1000000),2))
d<-mutate(d,d4=round((d4/1000000),2))
d<-mutate(d,d5=round((d5/1000000),2))
d<-mutate(d,d6=round((d6/1000000),2))
d<-mutate(d,d7=as.numeric((d5/d2)<0.6))
d<-mutate(d,d8=as.numeric((d6/d2)<0.15))
dane<-d
# ostateczne nazwy zmiennych
colnames(dane)<-c("y","x1","x2","x3","x4","x5","x6","x7")
# Zapisanie danych do pliku
write.table(dane,file="woj_dane.csv",sep=";",dec=",",row.names=TRUE,col.names=NA)
```

W rezultacie, w pliku `woj_dane.csv` znajduje się zbiór zawierający 288 obserwacji dla 8 zmiennych: zmiennej zależnej `y` oraz 7 zmiennych objaśniających `x1`-`x7`, w tym 2 zmiennych jakościowych (zero-jedynkowych).

Kolejnym krokiem jest zbudowanie modelu w postaci drzewa regresyjnego, w którym znajdą się wszystkie zmienne znajdujące się w ramce `dane`. Przyjęto założenie, że zbiór danych, zapisany w ostatnim poleceniu skryptu 8.2 do pliku, zostanie na nowo wczytany do programu R (R Core Team, 2023), ponieważ może być także wykorzystany w innym czasie lub do innych analiz.

Aby zbudować omówiony model regresji, należy uruchomić skrypt 8.3.

Skrypt 8.3

```
library(tree)
options(OutDec=",")
# wczytanie pliku z danymi
dane<-read.csv2("woj_dane.csv", header=TRUE, row.names=1)
# budowa modelu w postaci drzewa regresyjnego
m1<-tree(y~., data=dane)
# model w postaci listy węzłów drzewa regresyjnego
print(m1)
```

W rezultacie, na ekranie pojawi się struktura reprezentująca pełny model regresji, który zawiera 13 liści (węzłów końcowych), oznaczonych gwiazdką.

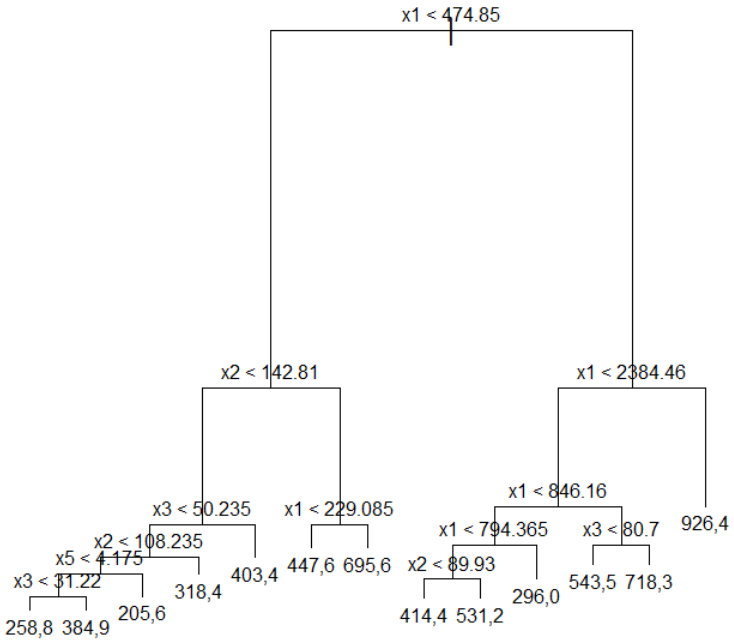
```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 208 9096000 383,6
   2) x1 < 474.85 129 3309000 296,6
     4) x2 < 142.81 118 1814000 269,9
       8) x3 < 50.235 106 1437000 254,8
        16) x2 < 108.235 83 1082000 237,2
          32) x5 < 4.175 28  522700 299,3
            64) x3 < 31.22 19  219600 258,8 *
            65) x3 > 31.22 9  206000 384,9 *
          33) x5 > 4.175 55  396300 205,6 *
        17) x2 > 108.235 23  236000 318,4 *
       9) x3 > 50.235 12  139500 403,4 *
     5) x2 > 142.81 11  509100 582,8
      10) x1 < 229.085 5  192800 447,6 *
      11) x1 > 229.085 6  148500 695,6 *
   3) x1 > 474.85 79 3215000 525,7
     6) x1 < 2384.46 74 2196000 498,7
      12) x1 < 846.16 50 1229000 456,2
        24) x1 < 794.365 42  955000 486,7
          48) x2 < 89.93 16  297400 414,4 *
          49) x2 > 89.93 26  522400 531,2 *
        25) x1 > 794.365 8   30080 296,0 *
      13) x1 > 846.16 24  688500 587,2
        26) x3 < 80.7 18  483300 543,5 *
        27) x3 > 80.7 6   67680 718,3 *
     7) x1 > 2384.46 5  161800 926,4 *
```

Model regresji zbudowany metodą drzew regresyjnych można także pokazać w postaci grafu, tak jak na rys. 8.7, po uruchomieniu skryptu 8.4, oczywiście pod warunkiem, że wcześniej powstał model `m1`, zgodnie z poleceniami w skrypcie 8.3.

Skrypt 8.4

```
# rysowanie modelu m1 ze skryptu 8.3 w postaci drzewa regresyjnego
# wraz z informacją o węzłach
plot(m1)
text(m1,cex=0.7)
```

Model pełny w postaci drzewa pokazano na rys. 8.7.



Rys. 8.7. Drzewo regresyjne jako ilustracja modelu `m1`

Źródło: obliczenia własne na podstawie danych BDL.

Interpretację tego modelu należy zacząć od kolejności występowania zmiennych w węzłach drzewa, ponieważ to oznacza wagę wpływu poszczególnych zmiennych objaśniających x1-x7 na zmienną zależną *Y*. Zastosowany algorytm rekurencyjnego podziału pokazuje, że zmienną o największej mocy predykcyjnej dla wielkości nakładów inwestycyjnych w województwach jest zmienna x1, czyli dochody własne. Można to wyjaśnić w ten sposób, że im większe dochody własne, tym większa skłonność do inwestowania. A w przypadku ich braku, rolę czynnika stymulującego odgrywa wielkość dotacji z budżetu państwa (x2). Dopiero później uruchamiane są środki z Unii Europejskiej (x3).

Województwa, w których kwartalne dochody własne przekraczały 474,85 mln zł, wydawały na inwestycje średnio 525,7 mln (węzeł nr 3). W przeciwnym wypadku średnia wartość wydatków wynosiła 296,6 mln złotych (węzeł nr 2).

Jak pokazuje model, większe wydatki inwestycyjne wynikają przede wszystkim z większych dochodów własnych. Skrajnie prawa gałąź drzewa na rys. 8.7 przedstawia regułę, że województwa, których dochody własne przekraczają w kwartale 2,384 mld zł (jest 5 takich województw), inwestują średnio 926,4 mln zł (węzeł nr 7).

Województwa, których dochody własne w kwartale są niższe niż 474,85 mln zł i które otrzymują dotacje z budżetu niższe niż 142,81 mln zł, wydają średnio 269,9 mln zł (węzeł nr 4). W przeciwnym wypadku kwartalne wydatki inwestycyjne rosną do poziomu 582,8 mln zł.

To potwierdza obserwację mechanizmu przyczynowego, który przedstawiono na początku tej części rozdziału, przy omawianiu zagadnienia finansowania inwestycji jednostek samorządu terytorialnego w Polsce. Jest on jeszcze lepiej widoczny na poziomie gmin.

Drzewo przedstawione na rys. 8.7 reprezentuje model maksymalnie złożony, w którym znajduje się 13 liści, co nieco komplikuje interpretację, i tak, jak wskazano w rozdz. 8.3, oznacza to, że jego zdolność predykcyjna będzie mała. Z tego powodu należy zastosować regularyzację, zgodnie z formułą (8.16). Realizuje to polecenie `prune.tree` znajdujące się także w pakiecie `tree`. Przyjęto tutaj wartość parametru `best=3`, aby wskazać, że przycięcie powinno ograniczyć jego wielkość do 3 węzłów (liści) i znacznie uprościć interpretację modelu. Wynik użycia tego polecenia przedstawiono w skrypcie 8.5.

Skrypt 8.5

```
# przycięcie drzewa do minimalnego rozmiaru (3 liście)
m1.pr<-prune.tree(m1,best=3)
print(m1.pr)
# rysunek drzewa przyciętego razem z informacjami o węzłach
plot(m1.pr)
text(m1.pr,cex=0.7)
```

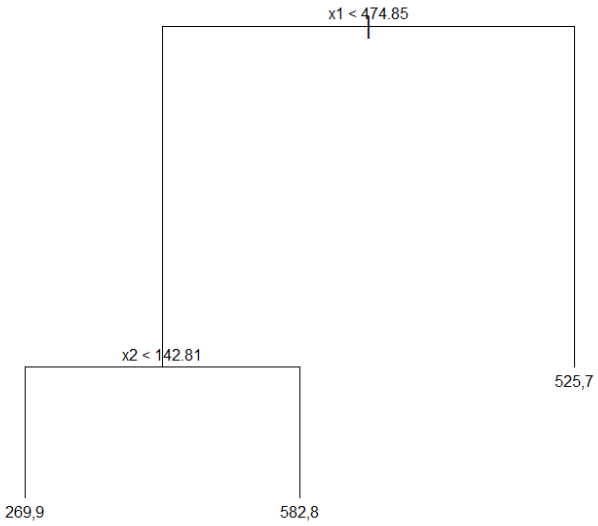
Na ekranie pojawi się wynik:

```
node), split, n, deviance, yval
      * denotes terminal node

1) root 208 9096000 383,6
  2) x1 < 474.85 129 3309000 296,6
    4) x2 < 142.81 118 1814000 269,9 *
    5) x2 > 142.81 11  509100 582,8 *
  3) x1 > 474.85 79 3215000 525,7 *
```

Trzy węzły oznaczone gwiazdką to liście, w których znajdują się średnie wartości nakładów inwestycyjnych w milionach złotych. Ułatwia to interpretację modelu oraz umożliwia ewentualną predykcję. Podobnie jak w modelu wyjściowym (rys. 8.6) w korzeniu drzewa znajduje się zmienna x1, o największej wartości predykcyjnej, czyli środki własne, kolejną zaś jest x2, czyli dotacje inwestycyjne z budżetu państwa.

Drzewo po przycięciu znajduje się na rys. 8.8. W przypadku tego przyciętego drzewa można przykładowo podać interpretację skrajnej ścieżki z jego lewej strony (idąc od korzenia do liścia), która w zapisie koniunkcyjnym ma postać: `(x1<474,85)∩(x2<142,81)`. To oznacza, że jeżeli

środki własne (x1) nie przekraczają 474,85 mln złotych, i jednocześnie dotacje (x2) są mniejsze niż 142,81 mln złotych, to średnie wydatki inwestycyjne w województwie wynoszą 269,9 mln złotych.



Rys. 8.8. Drzewo przycięte dla modelu `m1`

Źródło: obliczenia własne na podstawie danych BDL.

Aby ułatwić analizę zjawiska, można dobrać inną, bardziej odpowiednią liczbę liści drzewa poprzez wartość parametru `best` w poleceniu `prune.tree` w skrypcie 8.5.

Literatura

Breiman, L., Friedman, J., Olshen, R. i Stone, C. (1984). *Classification and Regression Trees*. CRC Press.

Gatnar, E. (2001). *Nieparametryczna metoda dyskryminacji i regresji*. Wydawnictwo Naukowe PWN.

Gatnar, E. (2008). *Podejście wielomodelowe w zagadnieniach dyskryminacji i regresji*. Wydawnictwo Naukowe PWN.

Kass, G. V. (1980). An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Applied Statistics*, *29*(2), 119-127. **https://doi.org/10.2307/2986296**

Morgan, J. N. i Sonquist, J. A. (1963). Problems in the Analysis of Survey Data: A Proposal. *Journal of the American Statistical Association*, *58*(302), 417-434. **https://doi.org/10.2307/2283276**

Quinlan, J. R. (1983). Learning Efficient Classification Procedures and Their Application to Chess and Games. W: R. S. Michalski, J. G. Carbonell, T. M. Mitchell (red.), *Machine Learning. An Artificial Intelligence Approach* (s. 126-142).

Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, *1*, 81-106. **https://doi.org/10.1007/BF00116251**

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Sekuła, A. i Śmiechowicz, J. (2018). Wydatki inwestycyjne województw samorządowych po wprowadzeniu indywidualnego wskaźnika zadłużenia. *Optimum Economic Studies*, *3*(93), 183-194. **https://doi.org/10.15290/oes.2018.03.93.15**

DOI: 10.15611/2024.45.1.09

9 Skalowanie wielowymiarowe

9.1. Idea skalowania wielowymiarowego

Termin „skalowanie wielowymiarowe" został zdefiniowany w 1952 r. przez Torgersona, który przedstawił jednocześnie pierwsze metryczne metody rozwiązania problemu. Największy rozwój metodologii skalowania wielowymiarowego przypada na lata 60. i 70. ubiegłego wieku. Najważniejsze z tego okresu są prace przedstawiające podstawy niemetrycznego skalowania (Kruskal, 1964a, 1964b; Shepard, 1962) oraz dotyczące wielowymiarowego skalowania różnic indywidualnych (Carroll i Chang, 1970, Takane i in., 1977). Lata 80. i 90. to przede wszystkim okres poszukiwań nowych metod prezentacji wyników skalowania (Young i Hamer, 1987, s. 15-40).

Skalowanie wielowymiarowe jest zbiorem metod, za pomocą których dokonuje się reprezentacji macierzy odległości (niepodobieństw) między obiektami w przestrzeni m-wymiarowej w geometryczną strukturę symbolizujących te obiekty punktów w przestrzeni r-wymiarowej ($r < m$), w celu wykrycia ilościowych lub jakościowych relacji występujących między obiektami. Ze względu na możliwości interpretacji, przestrzeń, w której prezentowane są wyniki skalowania wielowymiarowego, jest zazwyczaj przestrzenią dwuwymiarową.

Dla danego zbioru obiektów $A = \{A_1, \dots, A_n\}$ oraz określonych dla tych obiektów niepodobieństw (odległości) δ_{ik} tworzących macierz $\mathbf{\Delta} = [\delta_{ik}]_{nxn}$, poszukuje się takiego odwzorowania φ zbioru obiektów w zbiór punktów $\mathbf{X}$ w przestrzeni r-wymiarowej, aby

$$d_{ik} \approx \hat{d}_{ik} = f(\delta_{ik}), \tag{9.1}$$

gdzie: d_{ik} – odległość między punktami $\mathbf{x}_i$ oraz $\mathbf{x}_k$ w przestrzeni r-wymiarowej, $\hat{d}_{ik}$ – wartość funkcji regresji przekształcającej δ_{ik} w $\hat{d}_{ik}$, która jest miarą niepodobieństwa w przestrzeni r-wymiarowej.

W zależności od tego, jaki charakter mają zmienne opisujące badane obiekty, funkcja φ musi spełniać odpowiednie wymagania (Takane i in., 1977; Young i in., 1976):

- dla zmiennych mierzonych na skali porządkowej,

$$\delta_{ik} < \delta_{i'k'} => f(\delta_{ik}) \leq f(\delta_{i'k'}), \tag{9.2}$$

- dla zmiennych mierzonych na skali przedziałowej lub ilorazowej $f(\delta_{ik})$ jest liniowo zależne od δ_{ik}, tak że

$$f(\delta_{ik}) = a_0 + a_1\delta_{ik}, \tag{9.3}$$

przy $a_0 = 0$ dla zmiennych mierzonych na skali ilorazowej.

W skalowaniu wielowymiarowym przeprowadzanym w oparciu o macierz danych metrycznych w przestrzeni m-wymiarowej, macierz niepodobieństw $\mathbf{\Delta} = [\delta_{ik}]_{nxn}$ jest to macierz odległości wyznaczona na podstawie macierzy $\mathbf{Z} = [z_{ij}]_{nxm}$ (z_{ij} – znormalizowana wartość j-tej zmiennej dla i-tego obiektu). Metody normalizacji zmiennych w skalowaniu wielowymiarowym opisuje Walesiak (2016).

Wielkości $f(\delta_{ik})$ wyznaczane są w taki sposób, aby minimalizowały wartość funkcji dopasowania STRESS (*Standardized Residual Sum of Squares* – standaryzowana suma kwadratów reszt). W zależności od stosowanej procedury skalowania wielowymiarowego, funkcja dopasowania może przyjmować różne postacie analityczne (tab. 9.1).

Tabela 9.1. Wybrane postacie analityczne funkcji dopasowania w skalowaniu wielowymiarowym

Funkcja dopasowania	Postać analityczna
STRESS-1	$S = \sqrt{\frac{\sum_{i<k}(d_{ik}-f(\delta_{ik}))^2}{\sum_{i,k} d_{ik}^2}}$
STRESS-2	$S = \sqrt{\frac{\sum_{i<k}(d_{ik}-f(\delta_{ik}))^2}{\sum_{i,k}(d_{ik}-\bar{d}_{ik})^2}}$
S-STRESS	$SS = \sum_{i<k}\left(d_{ik}^2 - f(\delta_{ik})^2\right)^2$
Współczynnik Younga	$S = \sqrt{\frac{\sum_{i<k}\left(d_{ik}^2-f(\delta_{ik})^2\right)^2}{\sum_{i<k}\left(d_{ik}^2\right)^2}}$
Współczynnik alienacji Guttmana	$K = \sqrt{(1-r_c)^2}$, gdzie: $r_c = \frac{\sum_{i<k} d_{ik}\cdot f(\delta_{ik})}{\sqrt{\sum_{i<k} d_{ik}^2}\cdot\sqrt{\sum_{i<k} f(\delta_{ik})^2}}$
Miara największej wiarygodności	$ML = \sum_{i<k}(\log d_{ik} - \log f(\delta_{ik}))^2$

Źródło: opracowanie własne na podstawie (Borg i Groenen, 2005, s. 202-204; T. F. Cox i M. A. A. Cox, 2001, s. 74; Davison, 1983, s. 89; Kruskal, 1964b; Takane i in., 1977).

Na wartość funkcji dopasowania wpływa wiele czynników. Najważniejsze z nich to: liczba obiektów n (im większa wartość n, tym większa wartość funkcji dopasowania), liczba wymiarów r (im większa wartość r, tym mniejsza wartość funkcji dopasowania), błędy występujące w danych oraz liczba brakujących danych (Borg i Groenen, 2005, s. 54).

W interpretacji wyników skalowania wielowymiarowego należy brać pod uwagę nie tylko wartość funkcji dopasowania, ale również procentową dekompozycję udziału obiektów w wartości funkcji dopasowania (*stress per point).* Procentowy udział i-tego obiektu w wartości funkcji dopasowania dla STRESS-1 (SPP_i) oblicza się za pomocą formuły:

$$SPP_i = \frac{\sqrt{\frac{\sum_{k=1,k\neq i}^{n}(d_{ik}-f(\delta_{ik}))^2}{\sum_{k=1,k\neq i}^{n} d_{ik}^2}}}{\text{STRESS-1}} \cdot 100\%. \tag{9.4}$$

W skalowaniu wielowymiarowym wskazane jest, aby procentowe zróżnicowanie udziału poszczególnych obiektów w wartości funkcji dopasowania było jak najmniejsze (Borg i in., 2013, s. 68).

Ważnym zagadnieniem jest interpretacja wymiarów skalowania wielowymiarowego, mających charakter zmiennych ukrytych, pozwalających na wyjaśnienie podobieństw i różnic między obiektami. Najbardziej popularne jest podejście subiektywne, mimo że nie występują w tym przypadku ilościowe podstawy przyporządkowania wymiarów do cech badanych obiektów. W podejściu subiektywnym badacz do interpretacji osi na mapie percepcyjnej wykorzystuje swoją wiedzę na temat obiektów oraz zmiennych wpływających na określone rozmieszczenie punktów.

W metodach obiektywnych bada się (najczęściej stosując współczynniki korelacji) relacje między wymiarami skalowania wielowymiarowego a ocenami zmiennych dla każdego obiektu. Zmienne, które są wysoko skorelowane z wymiarami skalowania wielowymiarowego, służą do interpretacji osi.

9.2. Procedury skalowania wielowymiarowego

Procedury skalowania obejmują klasyczne skalowanie wielowymiarowe (KSW) oraz skalowanie najmniejszych kwadratów (*least squares scaling*), w którym przy wykorzystaniu rachunku różniczkowego poszukuje się konfiguracji punktów minimalizującej wartość funkcji dopasowania przy warunkach (9.2)-(9.3) zależnych od skal pomiaru zmiennych.

Klasyczne skalowanie wielowymiarowe jest najprostszą odmianą ze wszystkich metod skalowania wielowymiarowego. W przeciwieństwie do innych metod, KSW nie jest procedurą o charakterze iteracyjnym. Jego idea, którą zaprezentowali Young i Householder (1938), opiera się na wyznaczaniu na podstawie macierzy odległości w przestrzeni euklidesowej współrzędnych punktów reprezentujących obiekty. Na tej podstawie Torgerson (1952) zaproponował pierwszy algorytm skalowania wielowymiarowego w oparciu o dekompozycję macierzy kwadratów odległości. Na podstawie znanych kwadratów odległości d_{ik}^2 wyznacza się macierz produktów skalarnych $\mathbf{B} = -\frac{1}{2}\mathbf{H} \circ \boldsymbol{\Delta}^{(2)} \circ \mathbf{H}$, gdzie $\boldsymbol{\Delta}^{(2)}$ jest macierzą kwadratów odległości, $\mathbf{H} = \mathbf{I} - \frac{\mathbf{1}}{\boldsymbol{n}}\mathbf{1} \circ \mathbf{1}^{\boldsymbol{T}}$, przy czym $\mathbf{I}$ oznacza macierz jednostkową, natomiast $\mathbf{1} = (1,\ 1,\ \ldots,\ 1)^T$ jest wektorem n jedynek. Macierz $\mathbf{B}$ poddawana jest dekompozycji $\mathbf{B} = \mathbf{V}_1 \circ \boldsymbol{\Lambda}_1 \circ \mathbf{V}_1^T$, gdzie $\boldsymbol{\Lambda}_1 = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_r)$ jest diagonalną macierzą niezerowych wartości własnych, a $\mathbf{V}_1 = [v_1, v_2, \ldots, v_r]$ macierzą wektorów własnych odpowiadających wartościom własnym. W oparciu o wartości własne i wektory własne wyznaczana jest macierz współrzędnych punktów $\mathbf{X} = \mathbf{V}_1 \circ \boldsymbol{\Lambda}_1^{\frac{1}{2}}$, gdzie: $\boldsymbol{\Lambda}_1^{\frac{1}{2}} = \text{diag}\left(\lambda_1^{\frac{1}{2}},\ \lambda_2^{\frac{1}{2}},\ \ldots,\ \lambda_r^{\frac{1}{2}}\right)$. Szczegółowy algorytm klasycznego skalowania wielowymiarowego przedstawiony jest m.in. w pracach: Torgersona (1952), Borga i Groenena (2005, s. 261-263) oraz Zaborskiego (2001, s. 57-58).

Klasyczne skalowanie wielowymiarowe realizowane jest w programie R za pomocą funkcji `cmdscale` pakietu `stats`, a w przypadku modelu z wagami za pomocą funkcji `wcmdscale` pakietu `vegan`.

Procedury skalowania najmniejszych kwadratów mają charakter iteracyjny i przebiegają przez następujące etapy.

1. Wyznaczenie konfiguracji początkowej na podstawie macierzy niepodobieństw (zazwyczaj z wykorzystaniem klasycznego skalowania wielowymiarowego).
2. Znormalizowanie konfiguracji punktów, tak aby ich środek ciężkości znajdował się w początku układu współrzędnych.
3. Obliczenie odległości między punktami konfiguracji.
4. Szacowanie regresji, spełniającej odpowiedni dla skal pomiaru zmiennych warunek (9.2)-(9.3).
5. Obliczenie wartości funkcji dopasowania. Jeżeli różnica wartości funkcji dopasowania w danym i w poprzednim cyklu iteracyjnym jest mniejsza od ustalonej wcześniej stałej lub wykonano maksymalną, ustaloną liczbę iteracji, następuje prezentacja wyniku. W przeciwnym wypadku wyznaczana jest nowa konfiguracja punktów i przechodzi się do etapu 2.

Nowa konfiguracja punktów może być wyznaczana z wykorzystaniem metody gradientowej lub metody majoryzacji funkcji dopasowania. W metodzie gradientowej, dla funkcji dopasowania STRESS obliczane są wszystkie pochodne cząstkowe, a następnie wyznaczany jest gradient funkcji $\frac{\delta S}{\delta \mathbf{x}}$. Współrzędne wektora $\mathbf{x} = (x_{11}, \dots, x_{1r}, \dots, x_{nr})$ po K-tej iteracji wynoszą:

$$\mathbf{x}_K = \mathbf{x}_{K-1} - \frac{\frac{\delta S}{\delta x}\alpha_K}{\left|\frac{\delta S}{\delta x}\right|}, \tag{9.5}$$

przy czym α_K zmienia się w każdym cyklu iteracyjnym (Bąk, 2004, s. 101-102; Cox i Cox, 2001, s. 70; Kruskal, 1964b; Zaborski, 2001, s. 66). Skalowanie wielowymiarowe z wykorzystaniem metody gradientowej może być realizowane za pomocą jednej z funkcji: `isoMDS` pakietu `MASS`, `metaMDS` pakietu `vegan` lub `nmds` pakietu `ecodist`.

Idea metody majoryzacji opiera się na zastępowaniu w kolejnych cyklach iteracyjnych funkcji S-TRESS przez pewną funkcję „pomocniczą", której minimalizacja jest znacznie łatwiejsza niż minimalizacja S-STRESS (Borg i Groenen, 2005, s. 179-181; De Leeuw i Heiser, 1980; Guttman, 1968). W nieważonym modelu współrzędne punktów w K-tym cyklu iteracyjnym wyznaczane są zgodnie z równaniem (9.6), nazywanym potocznie transformacją Guttmana:

$$\mathbf{X}_K = n^{-1}\mathbf{B}(\mathbf{X}_{K-1})\mathbf{X}_{K-1}, \tag{9.6}$$

gdzie elementy macierzy $\mathbf{B}(\mathbf{X}_{K-1})$ dla współrzędnych punktów wyznaczonych w poprzednim cyklu iteracyjnym wynoszą:

$$b_{ik} = \begin{cases} \frac{-\delta_{ik}}{d_{ik}(\mathbf{X}_{K-1})} & \text{dla } i \neq k \text{ i } d_{ik}(\mathbf{X}_{K-1}) \neq 0 \\ 0 & \text{dla } i \neq k \text{ i } d_{ik}(\mathbf{X}_{K-1}) = 0 \\ -\sum_{\substack{k=1 \\ i \neq k}}^{n} b_{ik} & \text{dla } i = k \end{cases}. \tag{9.7}$$

Więcej na temat metody majoryzacji i jej zastosowania w różnych odmianach skalowania wielowymiarowego piszą m.in. De Leeuw i Mair (2009). Metoda majoryzacji funkcji dopasowania w skalowaniu wielowymiarowym jest wykorzystywana w funkcji `smacofSym` pakietu `smacof`.

9.3. Modele różnic indywidualnych

Modele różnic indywidualnych mogą mieć zastosowanie m.in. wtedy, gdy badanie relacji zachodzących między obiektami przeprowadzane jest w różnych okresach. Prekursorami badań różnic indywidualnych byli Bloxom (1968) i Horan (1969). Kontynuując prace w tym temacie, Carroll i Chang (1970) przedstawili model INDSCAL (*INdividual Differences SCALing*), Carroll i Wish (1974) model INDIOSCAL (*INdividual DIfferences in Orientation SCALing*), a Lingoes i Borg (1978) model PINDIS (*Procrustes INdividual DIfferences Scaling*).

Skalowania wielowymiarowego dokonuje się w oparciu o macierz $\mathbf{\Delta} = [\delta_{ik}]_{nxn}$, w której elementy δ_{ik} przedstawiają niepodobieństwa między obiektami A_i oraz A_k, więc mamy do czynienia z sytuacją, kiedy danych jest p macierzy $\mathbf{\Delta}_t = [\delta_{ik,t}]$ $(t = 1, 2, \dots, p)$, przy czym każda macierz $\mathbf{\Delta}_t$ przedstawia niepodobieństwa obiektów dla różnych okresów. Na podstawie p macierzy $\mathbf{\Delta}_t$ określa się wspólną konfigurację punktów $\mathbf{X} = [x_{ia}]$, gdzie $a = 1, 2, \dots, r$, reprezentującą badane obiekty w r-wymiarowej przestrzeni, która nazywana jest grupową przestrzenią zmiennych (*group stimulus space*).

W związku z tym, że w poszczególnych okresach różne zmienne mogą mieć decydujący wpływ na (nie)podobieństwa między obiektami, w modelach różnic indywidualnych wprowadza się wagi w_{ta} zmiennej a dla okresu t, tworzące przestrzeń wag $\mathbf{W} = [w_{ta}]$. Geometrycznie punkty wag prezentują okresy, w których przeprowadzano badanie, a współrzędne tych punktów to wagi przypisywane wymiarom w poszczególnych okresach. Dla ustalonej konfiguracji $\mathbf{X}$ indywidualne konfiguracje punktów $\mathbf{X}_t$ dla okresu t otrzymuje się mnożąc współrzędne punktów z grupowej przestrzeni przez odpowiednie wagi:

$$\mathbf{X}_t = \mathbf{X}\mathbf{W}_t, \tag{9.8}$$

gdzie $\mathbf{W}_t$ jest diagonalną macierzą wag o wymiarach $r \times r$ dla okresu t.

Modele różnic indywidualnych pozwalają na badanie interesujących nas zjawisk w różnych aspektach. W wyniku ich stosowania otrzymujemy (Zaborski, 2001, s. 67):

- wspólną dla wszystkich okresów konfigurację punktów przedstawiającą w przestrzeni r-wymiarowej relacje zachodzące między badanymi obiektami, przy uwzględnieniu różnic z różnych okresów;
- przestrzeń wag, której punktami są okresy badania; w przestrzeni tej można porównywać okresy pomiędzy sobą pod względem wag przypisywanych wymiarom;
- konfiguracje indywidualne umożliwiające badanie relacji między obiektami w różnych okresach.

W zależności od tego, co jest punktem wyjścia przy wyznaczeniu konfiguracji wspólnej oraz przestrzeni wag, można wymienić dwie grupy modeli realizowanych przez wymienione funkcje programu R:

- modele oparte na macierzach niepodobieństw – funkcje `smacofIndDiff`, `indscal` oraz `idioscal` pakietu `smacof`,
- modele oparte na konfiguracjach punktów z różnych okresów – funkcja `indscal` pakietu `SensoMiner` i funkcja `procGPA` pakietu `shapes`.

9.4. Analiza *unfolding*

Analiza *unfolding* (tłumaczona często jako metoda rozwijania) jest metodą, w której na podstawie macierzy danych dokonuje się rozmieszczenia na mapie percepcyjnej punktów reprezentujących obiekty oraz zmienne. Dzięki temu możliwa jest ocena zależności występujących między obiektami, między zmiennymi, jak również między obiektami a zmiennymi. Pozycje na mapie percepcyjnej, jakie zajmują punkty oznaczające zmienne, to tzw. punkty idealne, czyli punkty posiadające najbardziej preferowany poziom realizacji tej zmiennej. Pozycja punktów reprezentujących zmienne w stosunku do punktu obiektu określa względne znaczenie danej zmiennej dla oceny obiektu. Im bliżej punktu idealnego znajduje się punkt reprezentujący obiekt, tym lepiej dany obiekt jest oceniany ze względu na tę zmienną. Przed przeprowadzeniem skalowania wielowymiarowego zmienne w macierzy danych powinny zostać znormalizowane, a stymulanty i nominanty należy zamienić na destymulanty, ponieważ interpretacja wyników w analizie *unfolding* ma charakter preferencyjny.

Analiza *unfolding* często prowadzi do tzw. rozwiązań zdegenerowanych (De Leeuw, 2006). Z rozwiązaniem zdegenerowanym mamy do czynienia wtedy, gdy wartość funkcji dopasowania STRESS jest równa lub bliska 0 (co by wskazywało na bardzo dobre lub wręcz idealne dopasowanie), a struktura danych wejściowych jest nadal ukryta i nieinterpretowalna. W takim przypadku punkty należące do dwóch zbiorów, tj. zbioru obiektów i zbioru zmiennych, są oddalone od siebie o taką samą odległość. Sposobem na uniknięcie rozwiązań zdegenerowanych może być modyfikacja funkcji dopasowania STRESS, poprzez zastosowanie w jej konstrukcji współczynnika zmienności Pearsona, definiowanego jako (Busing i in., 2005):

$$v(\hat{\mathbf{d}}) = \frac{\text{odchylenie standardowe } (\hat{\mathbf{d}})}{\text{średnia } (\hat{\mathbf{d}})} = \frac{\sqrt{K^{-1}\sum_{j=1}^{K}\left(\hat{d}_j - \bar{\hat{d}}_j\right)^2}}{\bar{\hat{d}}}, \tag{9.9}$$

gdzie: $\bar{\hat{d}} = K^{-1}\sum_{j=1}^{K}\hat{d}_j$ (K – liczba wszystkich $\hat{d}_{ik}$).

Współczynnik zmienności traktowany jest jako diagnostyka służąca identyfikacji rozwiązań o równych wartościach $\hat{d}_{ik}$. W modelu wykorzystującym współczynnik zmienności do „karania" funkcji dopasowania w przypadku równych wartości $\hat{d}_{ik}$ funkcja dopasowania przyjmuje postać:

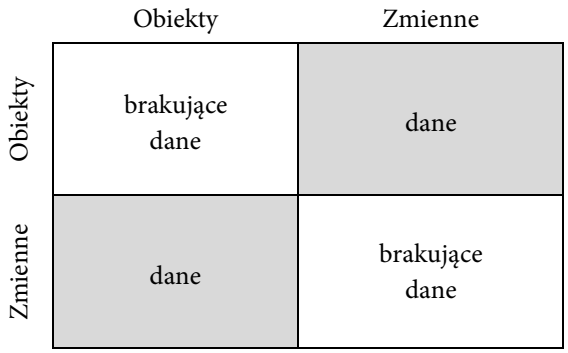
$$S_p = S^{2\lambda}\left(1 + \frac{\omega}{v^2(\hat{\mathbf{d}})}\right), \tag{9.10}$$

gdzie: S – znormalizowana funkcja STRESS-1, $0 < \lambda < 1,\ \omega > 0$ – parametry karania.

Parametr λ wpływa na równowagę pomiędzy znormalizowaną wartością funkcji STRESS-1 a czynnikiem kary $\left(1 + \frac{\omega}{v^2(\hat{\mathbf{d}})}\right)$. Im mniejsza wartość parametru λ, tym większy wpływ czynnika kary na wartość funkcji dopasowania. Parametr ω określa wielkość kary nałożonej na znormalizowaną wartość funkcji STRESS-1 przy określonym poziomie $v^2(\hat{\mathbf{d}})$.

W przeciwieństwie do metod skalowania wielowymiarowego, w których konfigurację punktów reprezentujących obiekty wyznacza się na podstawie symetrycznej macierzy niepodobieństw, w analizie *unfolding* dane wejściowe prezentowane są w postaci prostokątnej macierzy danych. Wiersze tej macierzy różnią się od jej kolumn. Pierwsze reprezentują obiekty, drugie zaś – zmienne. Taką macierz można traktować jako podmacierz macierzy niepodobieństw, w której dane są tylko

niepodobieństwa między obiektami a zmiennymi. Na jej podstawie przeprowadza się skalowanie wielowymiarowe, w którym niepodobieństwa między obiektami i między zmiennymi traktuje się jako brakujące dane.



Rys. 9.1. Macierz danych jako podmacierz macierzy niepodobieństw

Źródło: opracowanie własne na podstawie (Borg i Groenen, 2005, s. 295).

W analizie *unfolding* możliwe są dwa podejścia. W podejściu niezależnym (*unconditional unfolding*) porównywane są między sobą wszystkie elementy macierzy danych, a w podejściu zależnym (*conditional unfolding*) dokonuje się porównań niepodobieństw między wartościami zmiennych oddzielnie dla każdego obiektu (Borg i Groenen 2005, s. 299-300). Analizę *unfolding* w programie R (R Core Team, 2023) przeprowadza się za pomocą funkcji `smacofRect` pakietu `smacof` lub funkcji `unfold` pakietu `munfold`.

9.5. Zastosowania z wykorzystaniem programu R dla danych z BDL

Skalowanie wielowymiarowe

Za pomocą skalowania wielowymiarowego zbadano podobieństwa województw Polski pod względem dostępności do opieki zdrowotnej w 2021 r. W analizie uwzględniono następujące zmienne metryczne:

x1 – lekarze (personel pracujący) na 10 tys. ludności,
x2 – pielęgniarki i położne (personel pracujący) na 10 tys. ludności,
x3 – łóżka w hospicjach, zakładach opiekuńczo-leczniczych na 100 tys. ludności,
x4 – łóżka w szpitalach ogólnych na 10 tys. ludności,
x5 – przychodnie na 10 tys. ludności,
x6 – liczba aptek i punktów aptecznych na 10 tys. ludności,
x7 – średnia liczba porad lekarskich na osobę,
x8 – liczba szpitalnych oddziałów ratunkowych na 100 tys. ludności,
x9 – kadra medyczna pracująca w zespołach ratownictwa medycznego na 10 tys. ludności.

W tabeli 9.2 przedstawiono wykorzystane do badania zmienne dostępne w Banku Danych Lokalnych oraz odpowiednie przeliczenia.

Tabela 9.2. Zmienne dostępne w BDL oraz dokonane przeliczenia

Zmienne dostępne w BDL	ID	Wymagane przeliczenia
v1 – lekarze (personel pracujący) na 10 tys. ludności	454185	x1=v1
v2 – pielęgniarki i położne (personel pracujący) na 10 tys. ludności	472267	x2=v2
v3 – łóżka w hospicjach, zakładach opiekuńczo-leczniczych na 100 tys. ludności	454130	x3=v3
v4 – łóżka w szpitalach ogólnych	152354	x
v5 – przychodnie ogółem	53019	x
v6 – apteki	194884	x
v7 – punkty apteczne	1213	x
v8 – liczba osób przypadająca na szpitalny oddział ratunkowy	455427	x
v9 – porady lekarskie ogółem	1616687	x
v10 – ludność ogółem	72305	x
v11 – kadra medyczna pracująca w zespołach ratownictwa medycznego	196364	x
x4 – łóżka w szpitalach ogólnych na 10 tys. ludności	x	x4=(v4/v10) x 10.000
x5 – liczba przychodni na 10 tys. ludności	x	x5=(v5/v10) x 10.000
x6 – liczba aptek i punktów aptecznych na 10 tys. ludności	x	x6=(v6+v7)/v10 x 10.000
x7 – średnia liczba porad lekarskich na osobę	x	x7=v9/v10
x8 – liczba szpitalnych oddziałów ratunkowych na 100 tys. ludności	x	x8=(1/v8) x 100.000
x9 – kadra medyczna pracująca w zespołach ratownictwa medycznego na 10 tys. ludności	x	x9=(v11/v10) x 10.000

Źródło: opracowanie własne.

Pozyskanie danych z Banku Danych Lokalnych jest realizowane za pomocą skryptu 9.1.

Skrypt 9.1

```
library(bdl)
library(tidyverse)
options(OutDec=",")
# stałe
rok<-2021
no_uL=2 # (2 - województwo)
# Wczytanie informacji o zmiennych z BDL
v1<-get_data_by_variable("454185",unitLevel=no_uL,year=rok) %>%
  select(id,name, x1=val)
v2<-get_data_by_variable("472267",unitLevel=no_uL,year=rok) %>%
  select(id,name, x2=val)
v3<-get_data_by_variable("454130",unitLevel=no_uL,year=rok) %>%
  select(id,name, x3=val)
v4<-get_data_by_variable("152354",unitLevel=no_uL,year=rok) %>%
  select(id,name, x4=val)
v5<-get_data_by_variable("53019",unitLevel=no_uL,year=rok) %>%
  select(id,name, x5=val)
```

```
v6<-get_data_by_variable("194884",unitLevel=no_uL,year=rok) %>%
  select(id,name, x6=val)
v7<-get_data_by_variable("1213",unitLevel=no_uL,year=rok) %>%
  select(id,name, x7=val)
v8<-get_data_by_variable("455427",unitLevel=no_uL,year=rok) %>%
  select(id,name, x8=val)
v9<-get_data_by_variable("1616687",unitLevel=no_uL,year=rok) %>%
  select(id,name, x9=val)
v10<-get_data_by_variable("72305",unitLevel=no_uL,year=rok) %>%
  select(id,name, x10=val)
v11<-get_data_by_variable("196364",unitLevel=no_uL,year=rok) %>%
  select(id,name, x11=val)
# Połączenie zmiennnych w jedną tabelę danych
data <- v1 %>%
  full_join (v2, by=c("id","name")) %>%
  full_join (v3, by=c("id","name")) %>%
  full_join (v4, by=c("id","name")) %>%
  full_join (v5, by=c("id","name")) %>%
  full_join (v6, by=c("id","name")) %>%
  full_join (v7, by=c("id","name")) %>%
  full_join (v8, by=c("id","name")) %>%
  full_join (v9, by=c("id","name")) %>%
  full_join (v10, by=c("id","name")) %>%
  full_join (v11, by=c("id","name"))
v<-as.data.frame(data)
# Nadanie nazw wierszom
nazwy<-v[,2]
rownames(v)<-tolower(nazwy)
# Wybranie 11 kolumn z danymi oraz dodanie 6 pustych na przeliczenia
v<-v[,3:13]
for(i in 1:6){
  v<-cbind(v,rep(0,nrow(v)))
}
# Nadanie nazw kolumnom
colnames(v)<-paste("v",1:17,sep="")
# Przeliczenia
v<-mutate(v,v12=round((v4/v10)*10000,2))
v<-mutate(v,v13=round((v5/v10)*10000,2))
v<-mutate(v,v14=round((v6+v7)/v10*10000,2))
v<-mutate(v,v15=round((v9/v10),2))
v<-mutate(v,v16=round((1/v8)*100000,2))
v<-mutate(v,v17=round((v11/v10)*10000,2))
# Usunięcie zbędnych kolumn
x<-dplyr::select(v,-c(4:11))
# Nadanie nazw wierszom i kolumnom
colnames(x)<-paste("x",1:9,sep="")
rownames(x)<-tolower(nazwy)
print(x)
# Zapisanie danych do pliku
write.table(x,file="dane_sw_2021.csv",sep=";",dec=",",row.names=TRUE,col.names
=NA)
```

Po realizacji skryptu 9.1 został pobrany zbiór danych z BDL dla 11 zmiennych opisujących dostępność do opieki zdrowotnej mieszkańców w poszczególnych województwach w 2021 r. Dane zostały zapisane do pliku `dane_sw_2021.csv`. Po wykonaniu polecenia `print(x)` otrzymujemy:

```
                    x1    x2    x3    x4   x5   x6   x7   x8   x9
małopolskie         41,5  90,6 106,1 40,28 5,93 3,33 7,31 0,61 4,80
śląskie             42,9  98,4 116,8 50,81 6,06 3,35 7,89 0,32 4,92
lubuskie            25,1  67,8 115,6 41,09 5,82 3,39 6,58 0,81 5,56
wielkopolskie       32,6  77,8  49,5 37,63 5,49 3,83 7,24 0,71 4,62
zachodniopomorskie  35,3  71,0  71,5 42,74 5,70 3,53 7,16 0,61 6,10
dolnośląskie        41,1  86,5 145,9 46,26 5,44 3,39 7,55 0,55 4,78
opolskie            28,5  80,9 113,2 46,13 5,48 3,44 6,49 0,74 5,72
kujawsko-pomorskie  33,8  82,1 102,5 42,72 4,56 3,35 7,50 0,50 5,05
pomorskie           39,6  66,8  71,6 33,54 4,85 2,97 7,55 0,55 4,57
warmińsko-mazurskie 27,5  74,9  61,1 46,74 6,44 3,12 7,05 0,80 7,37
łódzkie             48,7  90,9  86,2 48,50 6,80 3,75 8,12 0,84 5,01
świętokrzyskie      36,6 100,0 104,2 46,34 5,32 3,60 6,93 0,84 5,11
lubelskie           44,5 100,7 101,8 51,86 6,10 3,85 7,91 0,83 6,36
podkarpackie        30,3  93,9 130,3 43,47 6,17 3,61 6,70 0,67 6,45
podlaskie           45,2  95,8  73,2 48,42 6,86 3,65 7,65 1,13 6,96
mazowieckie         55,2 109,7 114,3 45,48 5,84 3,18 8,25 0,58 4,77
```

Na podstawie przedstawionych danych przeprowadzono skalowanie wielowymiarowe (skrypt 9.2) za pomocą funkcji `smacofSym` pakietu `smacof`. Wcześniej dokonano normalizacji zmiennych z wykorzystaniem standaryzacji ilorazowej `normalization="9a"`, a w celu interpretacji osi obliczono współczynniki korelacji zmiennych z osiami konfiguracji punktów.

Skrypt 9.2

```
library(clusterSim)
library(smacof)
library(MASS)
options(OutDec=",")
# Wczytanie zbioru danych
x<-read.csv2("dane_sw_2021.csv",header=TRUE,row.names=1)
# Normalizacja zmiennych
z<-data.Normalization(x,type="n9a",normalization="column",na.rm=FALSE)
write.table(z,file="danenorm_sw_2021.csv",sep=";",dec=",",
  row.names=TRUE,col.names=NA)
znm<-as.matrix(z)
# Wyznaczenie odległości między obiektami
d<-dist.GDM(znm,method="GDM1")
#Przeprowadzenie skalowania wielowymiarowego
med<-smacofSym(delta=d,ndim=2,type="ratio",eps=1e-06)
# Współrzędne konfiguracji obiektów
print("Współrzędne konfiguracji obiektów",quote=FALSE)
print(med$conf)
print("Wartość funkcji dopasowania STRESS",quote=FALSE)
print(med$stress)
# Wyznaczanie korelacji między zmiennymi a wymiarami
print("Współczynniki korelacji", quote=FALSE)
r<-cbind(x,med$conf)
wyn<-round(cor(r),4)
print(wyn[1:ncol(x),(ncol(x)+1):(ncol(x)+ncol(med$conf))])
```

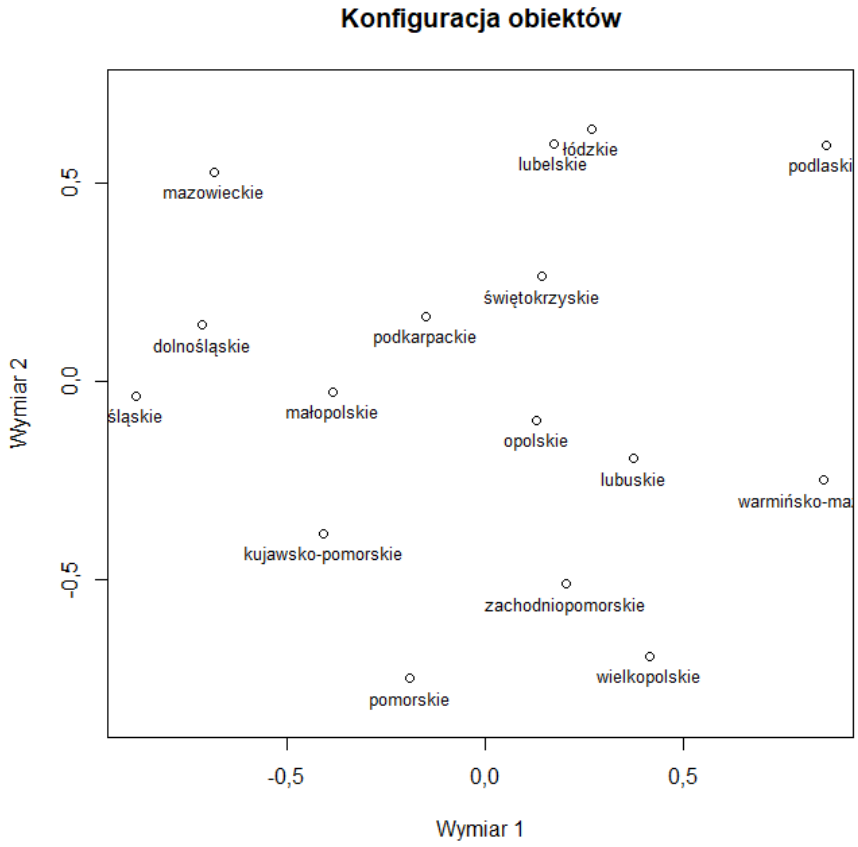
```
# Wyznaczanie konfiguracji obiektów
print ("Wyznaczanie konfiguracji obiektów", quote=FALSE)
plot(med$conf, xlab="Wymiar 1",ylab="Wymiar 2",main="Konfiguracja
obiektów",asp=1)
text(med$conf,labels=rownames(znm),pos=1,cex=0.8)
windows()
plot(med,"stressplot",main="Dekompozycja udziału obiektów w %\n w wartości
  miary dopasowania STRESS-1",ylim=c(0,30),xlab="Obiekty",ylab="%")
windows()
plot(med,"resplot",xlab="d-hats",ylab="d",main="Wykres dopasowania
odległości")
print("Współczynnik korelacji Pearsona r", guote=FALSE)
r<-cor(med$delta,med$confdist)
print(r)
```

W wyniku zastosowanej procedury otrzymano współrzędne oraz konfigurację obiektów, wartość funkcji dopasowania STRESS, współczynniki korelacji zmiennych z osiami mapy percepcyjnej, wykres przedstawiający procentową dekompozycję udziału obiektów w wartości funkcji dopasowania STRESS-1 i wykres dopasowania odległości d_{ik} oraz $\hat{d}_{ik}$.

```
[1] Współrzędne konfiguracji obiektów
                            D1          D2
małopolskie         -0,3857022 -0,02537057
śląskie             -0,8817318 -0,03552808
lubuskie             0,3737132 -0,19352553
wielkopolskie        0,4149665 -0,69281391
zachodniopomorskie   0,2031146 -0,50974802
dolnośląskie        -0,7153505  0,14266669
opolskie             0,1282089 -0,09776831
kujawsko-pomorskie  -0,4083972 -0,38302541
pomorskie           -0,1904265 -0,74777175
warmińsko-mazurskie  0,8537521 -0,24815961
łódzkie              0,2671876  0,63738350
świętokrzyskie       0,1439673  0,26456415
lubelskie            0,1734035  0,60025120
podkarpackie        -0,1510304  0,16509091
podlaskie            0,8586363  0,59602093
mazowieckie         -0,6843115  0,52773381
```

```
[1] Wartość funkcji dopasowania STRESS
[1] 0,211633
```

```
[1] Współczynniki korelacji
        D1     D2
x1 -0,3829 0,6061
x2 -0,3408 0,8107
x3 -0,6668 0,3946
x4  0,0191 0,7605
x5  0,4304 0,6270
x6  0,3473 0,3878
x7 -0,3493 0,4269
x8  0,8468 0,4443
x9  0,6671 0,2246
```



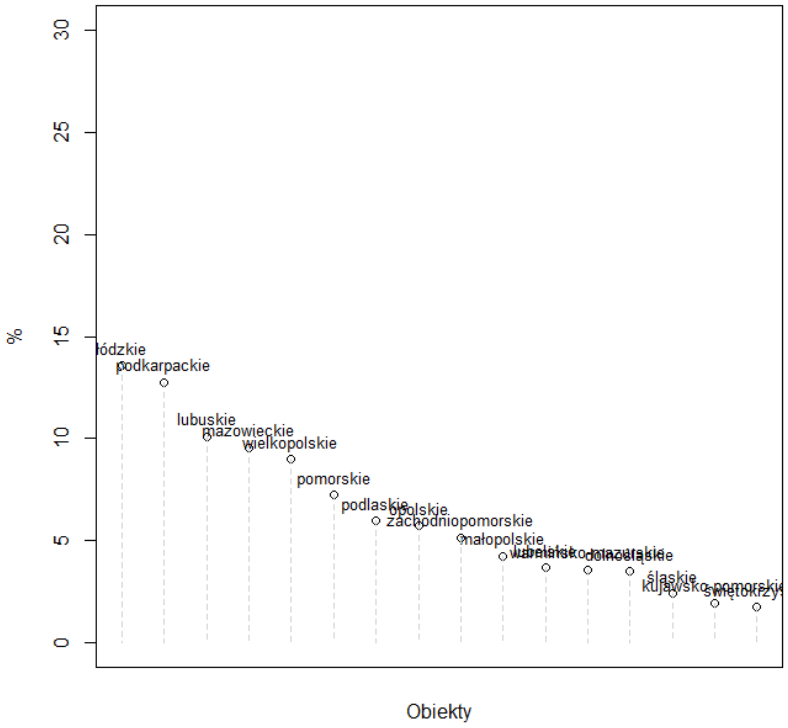
Rys. 9.2. Konfiguracja punktów reprezentujących województwa

Źródło: opracowanie własne na podstawie danych BDL.

Z wymiarem pierwszym najbardziej skorelowane są zmienne x3 (łóżka w hospicjach, zakładach opiekuńczo-leczniczych na 100 tys. ludności), x8 (liczba szpitalnych oddziałów ratunkowych na 100 tys. ludności) oraz x9 (kadra medyczna pracująca w zespołach ratownictwa medycznego na 10 tys. ludności), a z wymiarem drugim zmienna x1 (lekarze (personel pracujący) na 10 tys. ludności), x2 (pielęgniarki i położne (personel pracujący) na 10 tys. ludności), x4 (łóżka w szpitalach ogólnych na 10 tys. ludności) oraz x5 (liczba przychodni na 10 tys. ludności), i to właśnie te zmienne mogą posłużyć do interpretacji osi otrzymanej mapy percepcyjnej.

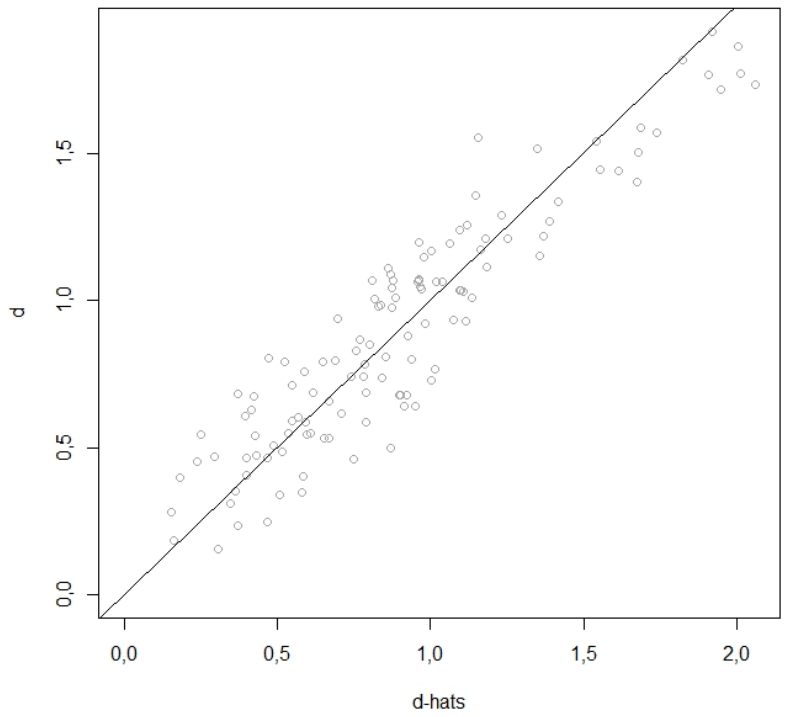
Rysunek 9.3. przedstawia procentową dekompozycję udziału obiektów (województw) w wartości funkcji dopasowania STRESS-1 (*stress per point*). W prezentowanym przykładzie udziały obiektów w wartości funkcji dopasowania mieszczą się w przedziale od 2 do 14%, co można przyjąć za wartości akceptowalne.

Również wykres dopasowania odległości d_{ik} oraz $\hat{d}_{ik}$ wskazuje na poprawność analizy (rys. 9.4). Współczynnik korelacji liniowej Pearsona otrzymanych w wyniku skalowania wielowymiarowego odległości między obiektami oraz ich niepodobieństw $r = 0{,}8658$.



Rys. 9.3. Procentowa dekompozycja udziału obiektów w wartości funkcji dopasowania STRESS-1

Źródło: opracowanie własne na podstawie danych BDL.



Rys. 9.4. Wykres dopasowania odległości d_{ik} (d) oraz $\hat{d}_{ik}$ (d-hats)

Źródło: opracowanie własne na podstawie danych BDL.

Analiza *unfolding*

Na podstawie danych pozyskanych z Banku Danych Lokalnych za pomocą skryptu 9.3 (dane z pliku `dane_sw_2021.csv`) przeprowadzono analizę *unfolding* za pomocą funkcji `smacofRect` pakietu `smacof` (skrypt 9.3). Wszystkie zmienne w badaniu są stymulantami, dlatego po przekształceniu ilorazowym (`normalization="n11"`) zostały zamienione na destymulany za pomocą formuły `max(znm)-znm`.

Skrypt 9.3

```
# Przeprowadzenie analizy unfolding
library(clusterSim)
library(smacof)
library(MASS)
options(OutDec=",")
# Wczytanie zbioru danych
x<-read.csv2("dane_sw_2021.csv",header=TRUE,row.names=1)
# Normalizacja zmiennych
znu<-data.Normalization(x,type="n11",normalization="column",na.rm=FALSE)
write.table(znu,file="danenorm_sw_2021.csv",sep=";",dec=",",
  row.names=TRUE,col.names=NA)
znmu<-as.matrix(znu)
#Zamiana stymulant na destymulanty
znmudes<-max(znmu)-znmu
med.unf<-smacofRect(znmudes,ndim=2,type="ratio",conditionality="row",
  ties="primary",eps=1e-6)
print(med.unf)
print("Współrzędne konfiguracji obiektów",quote=FALSE)
print(med.unf$conf.row, main="Współrzędne konfiguracji obiektów")
print("Współrzędne konfiguracji zmiennych",quote=FALSE)
print(med.unf$conf.col)
plot(med.unf,asp=1,xlab="Wymiar 1",ylab="Wymiar 2",main="Konfiguracja wspólna")
windows()
# Tworzenie wykresów dekompozycji funkcji dopasowania dla obiektów i zmiennych
op<-par(mfrow=c(1,2))
main1<-paste("Dekompozycja STRESS-1 dla obiektów")
main2<-paste("Dekompozycja STRESS-1 dla zmiennych")
xlab1<-"Wiersze"
xlab2<-"Kolumny"
ylab<-"SPP (%)"
spp.perc.row<-sort((med.unf$spp.row/sum(med.unf$spp.row)*100),decreasing=TRUE)
xaxlab<-names(spp.perc.row)
xaxlab<-(1:length(spp.perc.row))[order((med.unf$spp.row/summed.unf$spp.row)*100),
  decreasing=TRUE)]
xlim1<-c(1,length(spp.perc.row))
ylim1<-range(spp.perc.row)
plot(1:length(spp.perc.row),spp.perc.row,xaxt="n",type="p",xlab=xlab1,ylab=ylab,
  main=main1,xlim=xlim1,ylim=ylim1)
text(1:length(spp.perc.row), spp.perc.row,labels=xaxlab,pos=3,cex=0.8)
for(i in 1:length(spp.perc.row))lines(c(i,i),
  c(spp.perc.row[i],0),col="lightgray",lty=2)
spp.perc.col<-sort((med.unf$spp.col/sum(med.unf$spp.col)*100),decreasing=TRUE)
xaxlab<-names(spp.perc.col)
xlim1<-c(1,length(spp.perc.col))
ylim1<-range(spp.perc.col)
plot(1:length(spp.perc.col),spp.perc.col,xaxt="n",type="p",
```

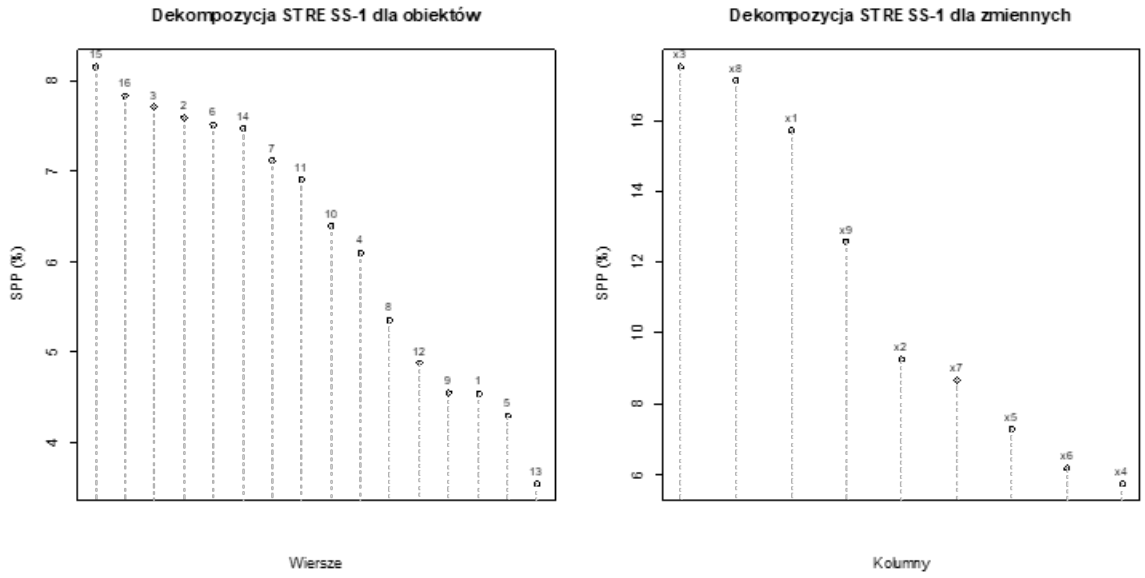
```
  xlab=xlab2,ylab=ylab,main=main2,xlim=xlim1,ylim=ylim1)
text(1:length(spp.perc.col),spp.perc.col,labels=xaxlab,pos=3,cex=0.8)
for(i in 1:length(spp.perc.col))
lines(c(i,i),c(spp.perc.col[i],0),col="lightgray",lty=2)
```

Po wykonaniu polecenia `print(med.unf)` otrzymujemy podstawowe informacje o zastosowanej procedurze, w tym o wartości funkcji dopasowania STRESS-1 oraz wartości czynnika kary funkcji dopasowania:

```
Model:               Rectangular smacof
Number of subjects:  16
Number of objects:   9
Transformation:      none
Conditionality:      row

Stress-1 value:    0,317446
Penalized Stress:  41,23482
Number of iterations: 1
```

W dalszej kolejności prezentowane są współrzędne punktów reprezentujących obiekty i zmienne, dekompozycja udziału obiektów i zmiennych (w %) w wartości miary dopasowania STRESS-1 (rys. 9.5) oraz konfiguracja łączna przedstawiająca obiekty i zmienne (rys. 9.6).

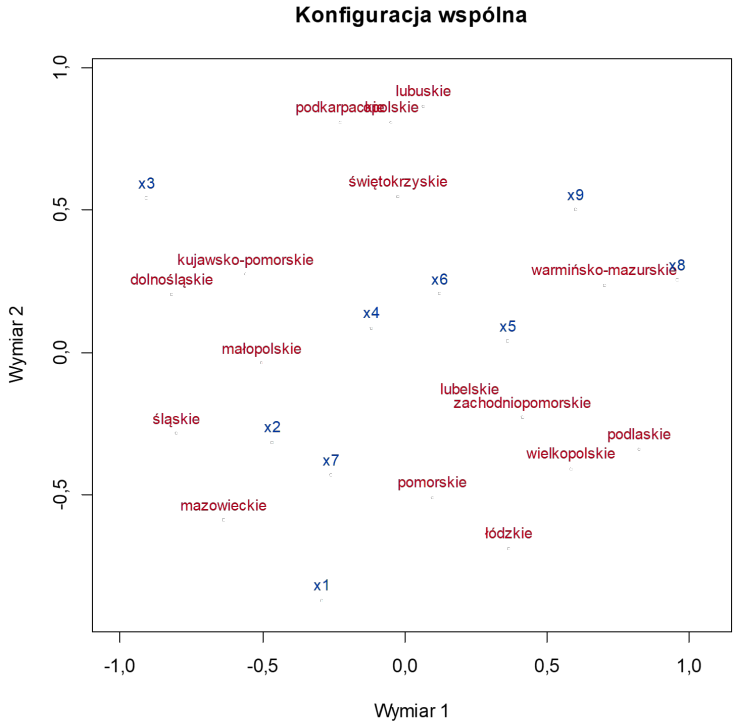
```
[1] Współrzędne konfiguracji obiektów
                            D1          D2
małopolskie         -0,50151973 -0,03939873
śląskie             -0,80344481 -0,28563414
lubuskie             0,06637584  0,86325309
wielkopolskie        0,58535166 -0,41098812
zachodniopomorskie   0,41288181 -0,22985443
dolnośląskie        -0,81956556  0,20171059
opolskie            -0,04946126  0,80776935
kujawsko-pomorskie  -0,56066738  0,27319904
pomorskie            0,09636613 -0,51104351
warmińsko-mazurskie  0,70146627  0,23417353
łódzkie              0,36547197 -0,68953504
świętokrzyskie      -0,02222743  0,54743237
lubelskie            0,22560610 -0,18360576
podkarpackie        -0,22833944  0,80692454
podlaskie            0,82390538 -0,34172237
mazowieckie         -0,63733514 -0,58980177

[1] Współrzędne konfiguracji zmiennych
           D1          D2
x1 -0,2934103 -0,87300708
x2 -0,4675252 -0,31775094
x3 -0,9065722  0,54077295
x4 -0,1190208  0,08449646
x5  0,3610672  0,03846308
x6  0,1241308  0,20495993
x7 -0,2583522 -0,43167475
x8  0,9577402  0,25385962
x9  0,6019426  0,49988074
```



Rys. 9.5. Procentowa dekompozycja udziału obiektów i zmiennych w wartości funkcji dopasowania STRESS-1

Źródło: opracowanie własne na podstawie danych BDL.

Rysunek 9.6 ukazuje względny wpływ poszczególnych zmiennych na ocenę województw pod kątem dostępności do opieki zdrowotnej. Im bliżej punktu województwa jest punkt reprezentujący zmienną, tym lepiej jest oceniane województwo ze względu na daną zmienną.



Rys. 9.6. Konfiguracja łączna prezentująca województwa i zmienne

Źródło: opracowanie własne na podstawie danych BDL.

Na przykład dla województwa warmińsko-mazurskiego, spośród wszystkich zmiennych, zmienna x8 (liczba szpitalnych oddziałów ratunkowych na 100 tys. ludności) ma największe znaczenie w ocenie dostępności do opieki zdrowotnej (podobnie jak zmienna x5 (przychodnie ogółem) dla województw lubelskiego i zachodniopomorskiego). Dla województwa śląskiego największe znaczenie ma zmienna x2 (pielęgniarki i położne (personel pracujący) na 10 tys. ludności), a drugą w kolejności jest zmienna x7 (średnia liczba porad lekarskich na osobę).

Literatura

Bąk, A. (2004). *Dekompozycyjne metody pomiaru preferencji w badaniach marketingowych*. Wydawnictwo Akademii Ekonomicznej we Wrocławiu.

Bloxom, B. (1968). Individual Differences in Multidimensional Scaling. *ETS Research Bulletin Series*, *1968*(2), 68-45 **https://doi.org/10.1002/j.2333-8504.1968.tb00567.x**

Borg, I. i Groenen, P. J. F. (2005). *Modern Multidimensional Scaling. Theory and Applications* (2nd Edition). Springer.

Borg, I., Groenen, P. J. F. i Mair, P. (2013). *Applied Multidimensional Scaling*. Springer. **https://doi.org/10.1007/978-3-642-31848-1**

Busing, F. M. T. A., Groenen, P. J. F. i Heiser, W. J. (2005). Avoiding Degeneracy in Multidimensional Unfolding by Penalizing on the Coefficient of Variation. *Psychometrika*, *70*(1), 71-79. **https://doi.org/10.1007/s11336-001-0908-1**

Carroll, J. D. i Chang, J. J. (1970). Analysis of Individual Differences in Multidimensional Scaling via an N-way Generalization of „Eckart-Young" Decomposition. *Psychometrika*, *35*(3), 283-319. **https://doi.org/10.1007/BF02310791**

Carroll, J. D. i Wish, M. (1974). Models and Methods for Three-way Multidimensional Scaling. W: D. H., Krantz R. C. Atkinson, R. D. Luce, P. Suppes (red.), *Contemporary Developments in Mathematical Psychology* (t. II, s. 57-105). Freeman.

Cox, T. F. i Cox, M. A. A. (2001). *Multidimensional Scaling* (2nd Edition). Chapman and Hall.

Davison, M. L. (1983). *Multidimensional Scaling*. John Wiley & Sons.

De Leeuw, J. (2006). *On Degenerate Nonmetric Unfolding Solutions*. Department of Statistics, UCLA. **https://escholarship.org/uc/item/3fp207gc**

De Leeuw, J. i Heiser, W. (1980). Multidimensional Scaling with Restrictions on the Configuration. W: P. R. Krishnaiah (red.), *Multivariate Analysis* (t. V, s. 501-522). North-Holand.

De Leeuw, J. i Mair, P. (2009). Multidimensional Scaling Using Majorization: SMACOF in R. *Journal of Statistical Software*, *31*(3), 1-30. **http://hdl.handle.net/10.18637/jss.v031.i03**

Guttman, L. (1968). A General Nonmetric Technique for Finding the Smallest Coordinate Space for a Configuration of Points. *Psychometrika*, *33*(4), 469-506. **https://doi.org/10.1007/BF02290164**

Horan, C. B. (1969). Multidimensional Scaling: Combining Observations When Individuals Have Different Perceptual Structures. *Psychometrika*, *34*(2), 139-165. **https://doi.org/10.1007/BF02289341**

Kruskal, J. B. (1964a). Multidimensional Scaling by Optimising Goodness of Fit to a Nonmetric Hypothesis. *Psychometrika*, *29*(1), 1-27. **https://doi.org/10.1007/BF02289565**

Kruskal, J. B. (1964b). Nonmetric Multidimensional Scaling: A Numerical Method. *Psychometrika*, *29*(2), 115-129. **https://doi.org/10.1007/BF02289694**

Lingoes, J. C. i Borg, I. (1978). A Direct Approach to Individual Differences Scaling Using Increasingly Complex Transformations. *Psychometrika*, *43*(4), 491-519. **https://doi.org/10.1007/BF02293810**

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Shepard, R. N. (1962). Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function. I. *Psychometrika*, *27*(2), 125-140. **https://doi.org/10.1007/BF02289630**

Takane, Y., Young, F. W. i de Leeuw, J. (1977). Nonmetric Individual Differences in Multidimensional Scaling: An Alternating Least Squares Method with Optimal Scaling Features. *Psychometrika*, *42*(1), 7-67. **https://doi.org/10.1007/BF02293745**

Torgerson, W. S. (1952). Multidimensional Scaling: I. Theory and Method. *Psychometrika*, *17*(4), 401--419. **https://doi.org/10.1007/BF02288916**

Walesiak, M. (2016). Wybór grup metod normalizacji wartości zmiennych w skalowaniu wielowymiarowym. *Przegląd Statystyczny*, *63*(1), 7-18. **https://doi.org/10.5604/01.3001.0014.1145**

Young, F. W., de Leeuw, J. i Takane, Y. (1976). Regression with Qualitative and Quantitative Variables: An Alternating Least Squares Method with Optimal Scaling Features. *Psychometrika*, *41*(4), 505-529. **https://doi.org/10.1007/BF02296972**

Young, F. W. i Hamer, R. M. (1987). *Multidimensional Scaling. History*, *Theory and Applications*. Lawrence Erlbaum Associates.

Young, G. i Householder, A. S. (1938). Discussion of a Set of Points in Terms of Their Mutual Distances. *Psychometrika*, *3*(1), 19-22. **https://doi.org/10.1007/BF02287916**

Zaborski, A. (2001). *Skalowanie wielowymiarowe w badaniach marketingowych*. Wydawnictwo Akademii Ekonomicznej we Wrocławiu.

DOI: 10.15611/2024.45.1.10

10 Analiza skupień[1]

10.1. Podstawowe problemy zagadnienia klasyfikacji

Według najogólniejszej koncepcji klasyfikacja jest zbiorem klas odpowiednio wyróżnionym z klasyfikowanego zbioru obiektów. Zawężone sformułowanie zagadnienia klasyfikacji zbioru A o elementach A_i $(i = 1, \dots, n)$ na klasy $P_1, \dots, P_u$ spełnia warunki: zupełności, rozłączności i niepustości.

Można wyróżnić cztery podstawowe problemy decydujące o skali trudności problemu klasyfikacji (Gordon, 1999, s. 3-4; Milligan, 1996, s. 343-344).

Pierwszy problem dotyczy liczby klasyfikowanych obiektów. Liczbę wszystkich podziałów zbioru n obiektów na u niepustych klas wyznacza się ze wzoru (Everitt i in., 2001, s. 99; Gordon, 1999, s. 40)

$$L(n,u) = \frac{1}{u!}\sum_{s=1}^{u}(-1)^{u-s}\binom{u}{s}s^n, \tag{10.1}$$

gdzie: $s = 1, \dots, u$ – numer klasy.

Przykładowe liczby wszystkich możliwych podziałów zbioru n obiektów na u niepustych klas są następujące: $L(5,3) = 25$, $L(10,4) = 34.105$, $L(50,5) > 7{,}4 \cdot 10^{32}$, $L(100,6) > 9 \cdot 10^{74}$.

Liczbę wszystkich podziałów zbioru obiektów wyznacza się ze wzoru

$$L(n) = L(n,1) + \cdots + L(n,n). \tag{10.2}$$

Przykładowe liczby wszystkich możliwych podziałów zbioru n obiektów są następujące: $L(5) = 52$, $L(10) = 115.975$, $L(50) > 1{,}8 \cdot 10^{47}$; $L(100) > 4{,}7 \cdot 10^{115}$.

Wraz ze wzrostem liczby klasyfikowanych obiektów liczba możliwych podziałów zbioru n obiektów staje się ogromna. Zatem rozpatrzenie wszystkich możliwych podziałów zbioru n obiektów, z punktu widzenia pewnego kryterium, i wybór na tej podstawie podziału najlepszego nie jest wykonalne dla większych liczebności zbioru obiektów. W tej sytuacji niezbędne są efektywne metody i algorytmy klasyfikacji.

[1] Rozdziały 10.1-10.3 opracowano na podstawie (Walesiak, 2009, s. 407-421; 2016, s. 64-75).

Drugim elementem decydującym o skali trudności problemu klasyfikacji jest liczba zmiennych opisujących badane obiekty. Dla jednej zmiennej uzyskujemy uporządkowanie obiektów na prostej, a dla dwóch zmiennych rozmieszczenie obiektów na płaszczyźnie. W zasadzie dla tych dwóch przypadków istnieje możliwość wizualizacji rozmieszczenia obiektów. Uwzględnienie więcej niż trzech zmiennych powoduje, że do rozwiązania problemu klasyfikacji niezbędne stają się odpowiednie metody i algorytmy klasyfikacji.

Trzeci problem dotyczy rozmieszczenia obiektów w przestrzeni klasyfikacji i braku powszechnie akceptowanej definicji klasy. W literaturze przedmiotu wypracowano wiele definicji klas, por. m.in. (Dąbrowski i Laus-Mączyńska, 1978, s. 62-66; Everitt, 1974, s. 43-48), które mają zastosowanie w specyficznych przypadkach. Głównym celem klasyfikacji jest badanie podobieństwa lub odrębności obiektów i ich zbiorów. Celem tym jest więc podział zbioru obiektów na klasy zawierające obiekty podobne ze względu na obserwacje na zmiennych (tzw. klasy względnie jednorodne). Ponadto obiekty znajdujące się w różnych klasach powinny być jak najmniej podobne. Postuluje się, aby wyodrębnione klasy spełniały dwa kryteria: wewnętrznej spójności i zewnętrznej izolacji (Gordon, 1999, s. 3).

Czwarty problem decydujący o skali trudności problemu klasyfikacji dotyczy braku szeroko akceptowanej i ujednoliconej teorii klasyfikacji.

10.2. Etapy występujące w typowej analizie skupień

W literaturze przedmiotu wyodrębnia się siedem etapów typowej analizy skupień, por. np. (Milligan, 1996, s. 342-343; Walesiak, 2004, 2005a):

1. Wybór obiektów i zmiennych.
2. Wybór metody normalizacji wartości zmiennych.
3. Wybór miary odległości (etap ten nie występuje w analizie skupień bazującej bezpośrednio na macierzy danych).
4. Wybór metody klasyfikacji.
5. Ustalenie liczby klas.
6. Ocena wyników klasyfikacji.
7. Opis (interpretacja) i profilowanie klas.

Etap 1. Wybór obiektów i zmiennych

Należy odpowiedzieć na pytanie, czy badaniem objąć całą populację, czy tylko jej próbkę? Jeśli zdecydowano się na badanie próbkowe, to należy określić elementarną jednostkę badania, wybrać metodę doboru próby i określić jej liczebność. W każdym badaniu statystycznym, w tym również w niewyczerpującym badaniu wielowymiarowym, można przyjąć jedno z dwóch podejść: stochastyczne lub opisowe. W podejściu stochastycznym zakłada się, że zbiór obserwacji (obiektów) stanowi próbę losową pochodzącą z populacji (o nieskończonej lub skończonej liczebności). Podejście stochastyczne, w którym rozpatrywane zmienne są losowe, wolno przyjąć przede wszystkim w przypadku badań eksperymentalnych, tzn. gdy istnieje możliwość powtórzenia badania w takich samych warunkach. Wtedy zbiór obserwacji może być traktowany jako próba

losowa. W podejściu opisowym zmienne nie są losowe, lecz są zmiennymi w zwykłym sensie. Badaniu nie podlegają wtedy właściwości stochastyczne zbioru obserwacji. Podejście opisowe przyjmuje się z reguły wtedy, gdy dane pochodzą ze sprawozdawczości statystycznej. Dobór próby powinno się tak przeprowadzić, aby klasy wyodrębnione na jej podstawie odpowiadały strukturze klas populacji.

Wybór zmiennych jest jednym z najważniejszych, a zarazem najtrudniejszych zagadnień. Od jakości zestawu zmiennych zależy bowiem wiarygodność ostatecznych wyników klasyfikacji i trafność podejmowanych na ich podstawie decyzji. W procedurze klasyfikacji należy uwzględnić tylko te zmienne, które mają zdolność dyskryminacji zbioru obiektów. Podejście polegające na uwzględnieniu jak największej liczby zmiennych jest nieuzasadnione. Dodanie do zbioru jednej lub kilku nieistotnych zmiennych nie pozwala na odkrycie w zbiorze obiektów właściwej struktury klas (Milligan, 1994; 1996, s. 348).

Do rozwiązania zagadnienia doboru zmiennych służą zasadniczo dwa ujęcia: dobór merytoryczny w ścisłym tego słowa znaczeniu i dobór merytoryczno-formalny. Oba ujęcia obejmują dwie fazy. Faza I jest taka sama w obu ujęciach, różnice zaś występują w fazie II. Punktem wyjścia obu ujęć (faza I) jest skonstruowanie wstępnej listy zmiennych na podstawie własnej hipotezy roboczej badacza (wynikającej z jego znajomości przedmiotu badania oraz wiedzy płynącej z szeroko pojętej teorii ekonomii) oraz współpracy z przedstawicielami odpowiednich dyscyplin naukowych (ekspertami).

Redukcja wstępnej listy zmiennych z wykorzystaniem analizy merytorycznej (faza II) jest działaniem w głównej mierze subiektywnym. Dokonuje się jej na podstawie własnej znajomości przedmiotu badania, wykorzystując współpracę przedstawicieli odpowiednich dyscyplin naukowych (ekspertów) oraz opierając się na szeroko pojętej teorii ekonomii. Redukcja wstępnej listy zmiennych z wykorzystaniem metod doboru zmiennych (faza II) polega na zastosowaniu formalnych algorytmów wyboru zmiennych.

W zagadnieniu wyboru zmiennych na potrzeby klasyfikacji zbioru obiektów na względnie jednorodne klasy wyróżnia się trzy podejścia (Gnanadesikan i in., 1995; Grabiński, 1992, s. 42):

1) selekcję zmiennych – dobór mniejszej liczby zmiennych przez eliminację tych, które nie mają zdolności dyskryminacji zbioru obiektów;
2) wprowadzenie zróżnicowanych wag dla poszczególnych zmiennych wyrażających ich relatywną ważność;
3) zastąpienie oryginalnych zmiennych nowymi, „sztucznymi" zmiennymi o pożądanych właściwościach.

Zagadnienie selekcji zmiennych jest szczególnym przypadkiem ważenia zmiennych, ponieważ zmienne usunięte otrzymują wagę 0, a zmienne wybrane wagę 1.

W metodach opartych na niepowielaniu informacji (miernikiem niepowielania informacji jest współczynnik korelacji liniowej Pearsona lub odległości oparte na nim) wyróżnia się:

- dualne procedury taksonometryczne służące selekcji zmiennych (w pierwszym kroku wyodrębnia się klasy zmiennych podobnych, a w kroku drugim dokonuje się wyboru reprezentantek poszczególnych klas); celem zastosowania tych procedur jest wybór takiego

zestawu zmiennych, w którym są one wzajemnie nieskorelowane oraz są silnie skorelowane ze zmiennymi wyeliminowanymi z badań, por. np. (Grabiński, 1992, s. 44-47);
- propozycję wprowadzenia ważenia zmiennych proporcjonalnie do stopnia ich skorelowania z pozostałymi zmiennymi; system ten nadaje wyższe wagi zmiennym silniej skorelowanym z pozostałymi zmiennymi (Grabiński, 1992, s. 34-35);
- zastosowanie analizy czynnikowej lub analizy głównych składowych w celu transformacji wielowymiarowej przestrzeni zmiennych w jakościowo nowy układ (przestrzeń) czynników (ortogonalnych składowych głównych) o mniejszej liczbie wymiarów.

Dla metod wyboru zmiennych opartych na niepowielaniu informacji zgłaszane są następujące uwagi krytyczne:

- dla większości metod klasyfikacji nie przyjmuje się założenia o normalności rozkładu zmiennych ani o ich nieskorelowaniu (Milligan, 1996, s. 347-348),
- metody wyboru zmiennych oparte na ich nieskorelowaniu niekoniecznie prowadzą do właściwych rezultatów (Guyon i Elisseeff, 2003; Walesiak, 2005b),
- Sneath (cyt. za: Milligan, 1996, s. 348) pokazał, że transformacja pierwotnej wielowymiarowej przestrzeni klasyfikacji w przestrzeń o mniejszej liczbie ortogonalnych składowych, w wyniku zastosowania analizy głównych składowych, może spowodować utratę struktury klas z pierwotnej przestrzeni.

Z tego względu w literaturze proponowane są następujące metody wyboru zmiennych pozwalające na określenie zdolności zmiennych do dyskryminacji zbioru obiektów: metody selekcji zmiennych i metody ważenia zmiennych.

Metody selekcji zmiennych. Ze względu na to, że nie jest możliwe rozpatrzenie wszystkich możliwych kombinacji zmiennych z powodu ich ogromnej liczby (np. dla 20 zmiennych otrzymuje się $2^{20} - 1 = 1.048.575$ kombinacji, dla 30 zmiennych 1.073.741.823) stosuje się w praktyce uproszczone procedury.

- Sokołowski (1992, s. 12-13, 50-51) zaproponował miarę zdolności grupowania dla indywidualnych zmiennych i zestawu zmiennych.
- Donoghue (1995) zastosował do selekcji zmiennych jednowymiarowe miary oparte na współczynnikach skośności $g_1 = \frac{M_3}{M_2^{3/2}}$ i kurtozy (ekscesu) $g_2 = \frac{M_4}{M_2^2} - 3$:, $m = g_2 - g_1^2$, $b = (g_1^2 + 1)/(g_2 + 3)$. Jeśli b $(0 < b \leq 1)$ i m odchylają się od progów wartości (np. $b > 0{,}333$ i $m < 0$ dla rozkładu normalnego), oznacza to występowanie wielomodalności dla poszczególnych zmiennych. Graniczne wartości dla innych rozkładów zmiennych podaje Donoghue (1995, s. 395).
- Fowlkes i in. (1987; 1988) zaproponowali procedurę doboru zmiennych, znaną w analizie regresji pod nazwą procedury selekcji „w przód", powiązaną z metodami hierarchicznymi.
- Carmone i in. (1999) zaproponowali heurystyczną procedurę doboru zmiennych (HINoV) powiązaną z metodą *k*-średnich i skorygowanym indeksem Randa. W pakiecie `clusterSim` (Walesiak i Dudek, 2023) znajduje się funkcja prezentująca rozszerzoną wersję tej metody dla danych niemetrycznych i innych metod klasyfikacji.

Metody ważenia zmiennych

- DeSarbo i in. (1984) zaproponowali ważoną metodę *k*-średnich, w której poszukuje się wag dla zmiennych, optymalizujących wyniki klasyfikacji w sensie funkcji *stress* Kruskala;
- De Soete (1986; 1988), na podstawie ważonej odległości euklidesowej, zaproponował metodę, w której poszukuje się wag dla zmiennych, optymalizujących spełnianie w macierzy odległości własności ultrametryki ($d_{ik} \leq \max\{d_{il}, d_{kl}\}$, dla wszystkich trójek obiektów o numerach i, k, l) lub addytywności ($d_{ik} + d_{ip} \leq max\{d_{il}, d_{kp}, d_{ip}, d_{kl}\}$, dla wszystkich czwórek obiektów o numerach i, k, l, p);
- Makarenkov i Legendre (2001), na podstawie ważonej odległości euklidesowej, zaproponowali metodę, w której poszukuje się wag dla zmiennych, optymalizujących spełnianie kryterium podziału w metodzie *k*-średnich na ustaloną liczbę klas;
- Friedman i Meulman (2004) opracowali algorytm ważenia zmiennych w ramach wyodrębnionych klas, a nie całej zbiorowości badanych obiektów. W algorytmie tym poszczególne klasy mogą być opisane innym zestawem zmiennych lub tymi samymi zmiennymi jednak o innej strukturze wag.

Analizę porównawczą wybranych metod ważenia i selekcji zmiennych zawierają prace: Gnanadesikana i in. (1995); Makarenkova i Legendre'a (2001) oraz Milligana (1989). Obszerniej o problemach selekcji i ważenia zmiennych w zagadnieniu klasyfikacji traktują m.in. prace Walesiaka (2005b) oraz Korzeniewskiego (2012).

Etap 2. Wybór formuły normalizacji wartości zmiennych

Normalizację przeprowadza się, gdy zmienne opisujące obiekty badania mierzone są na skali przedziałowej lub ilorazowej. W odniesieniu do słabych skal pomiaru (nominalnej i porządkowej) nie zachodzi potrzeba normalizacji, na ich wartościach bowiem nie wyznacza się ani relacji równości różnic i przedziałów, ani stosunków.

Celem normalizacji wartości zmiennych jest doprowadzenie zmiennych do porównywalności. Uzyskuje się to przez pozbawienie mian wyników pomiaru oraz ujednolicenie ich rzędów wielkości. Pierwszy cel normalizacji jest jednoznaczny. Stanowi on warunek *sine qua non* normalizacji. Cel drugi nie jest jednoznaczny, a zatem dopuszcza w tym zakresie różne rozwiązania. Ujednolicenie rzędów wielkości dla zmiennych uzyskuje się np. przez ujednolicenie wartości wszystkich zmiennych pod względem zmienności mierzonej odchyleniem standardowym (medianowym odchyleniem bezwzględnym dla miar pozycyjnych) lub przez zapewnienie stałości rozstępu dla znormalizowanych wartości zmiennych. Ogólnie rzecz biorąc, ujednolicenie rzędów wielkości uzyskuje się przez wprowadzenie jednolicie określonej wartości zerowej dla wszystkich zmiennych, a następnie przeskalowanie wartości zmiennych.

Ze względu na to, że jedynymi dopuszczalnymi przekształceniami na skali przedziałowej i ilorazowej (skale metryczne) są przekształcenia liniowe, formuły normalizacyjne można wyrazić ogólnym wzorem (Jajuga i Walesiak, 2000):

$$z_{ij} = b_j x_{ij} + a_j = \frac{x_{ij} - A_j}{B_j} = \frac{1}{B_j} x_{ij} - \frac{A_j}{B_j} \ (b_j > 0), \tag{10.3}$$

gdzie: $x_{ij}(z_{ij})$ – wartość (znormalizowana wartość) j-tej zmiennej dla i-tego obiektu, A_j – parametr przesunięcia do umownego zera dla j-tej zmiennej, B_j – parametr skali dla j-tej zmiennej, $a_j = -A_j/B_j$, $b_j = 1/B_j$ – parametry dla j-tej zmiennej. Tabela 10.1 przedstawia metody normalizacji wartości zmiennych zdefiniowane ogólną formułą (10.3).

Tabela 10.1. Metody normalizacji wartości zmiennych

Typ	Nazwa formuły	Parametr		Skala pomiaru zmiennych	
		B_j	A_j	przed normalizacją	po normalizacji
n1	Standaryzacja	s_j	$\bar{x}_j$	ilorazowa lub przedziałowa	przedziałowa
n2	Standaryzacja pozycyjna	mad_j	med_j	ilorazowa lub przedziałowa	przedziałowa
n3	Unitaryzacja	r_j	$\bar{x}_j$	ilorazowa lub przedziałowa	przedziałowa
n3a	Unitaryzacja pozycyjna	r_j	med_j	ilorazowa lub przedziałowa	przedziałowa
n4	Unitaryzacja zerowana	r_j	$\min_i\{x_{ij}\}$	ilorazowa lub przedziałowa	przedziałowa
n5	Normalizacja w przedziale $[-1;1]$	$\max_i\lvert x_{ij} - \bar{x}_j\rvert$	$\bar{x}_j$	ilorazowa lub przedziałowa	przedziałowa
n5a	Normalizacja pozycyjna w przedziale $[-1;1]$	$\max_i\lvert x_{ij} - med_j\rvert$	med_j	ilorazowa lub przedziałowa	przedziałowa
n6	Przekształcenia ilorazowe	s_j	0	ilorazowa	ilorazowa
n6a		mad_j	0	ilorazowa	ilorazowa
n7		r_j	0	ilorazowa	ilorazowa
n8		$\max_i\{x_{ij}\}$	0	ilorazowa	ilorazowa
n9		$\bar{x}_j$	0	ilorazowa	ilorazowa
n9a		med_j	0	ilorazowa	ilorazowa
n10		$\sum_{i=1}^{n} x_{ij}$	0	ilorazowa	ilorazowa
n11		$\sqrt{\sum_{i=1}^{n} x_{ij}^2}$	0	ilorazowa	ilorazowa
n12	Normalizacja	$\sqrt{\sum_{i=1}^{n} (x_{ij} - \bar{x}_j)^2}$	$\bar{x}_j$	ilorazowa lub przedziałowa	przedziałowa
n12a	Normalizacja pozycyjna	$\sqrt{\sum_{i=1}^{n} (x_{ij} - med_j)^2}$	med_j	ilorazowa lub przedziałowa	przedziałowa
n13	Normalizacja z zerem usytuowanym centralnie	$\frac{r_j}{2}$	m_j	ilorazowa lub przedziałowa	przedziałowa

$x_{ij}(z_{ij})$ – wartość (znormalizowana wartość) j-tej zmiennej dla i-tego obiektu; $\bar{x}_j$ $(s_j,\ r_j)$ – średnia (odchylenie standardowe, rozstęp) dla j-tej zmiennej; $m_j = \frac{\max_i\{x_{ij}\} + \min_i\{x_{ij}\}}{2}$ – środek rozstępu (*mid-range*); $med_j = \underset{i}{med}(x_{ij})$ – mediana dla j-tej zmiennej; $mad_j = \underset{i}{mad}(x_{ij})$ – medianowe odchylenie bezwzględne dla j-tej zmiennej.

Źródło: opracowanie własne.

Normalizacja wartości zmiennych w pakiecie `clusterSim` programu R jest następująca:

```
data.Normalization(x,type="n0",normalization="column")
```

gdzie: `x` – macierz danych, `type` – typ formuły normalizacyjnej z tabeli, `normalization` – rodzaj normalizacji: `"column"` – normalizacja według zmiennych (kolumny w macierzy danych), `"row"` – normalizacja według obiektów (wiersze w macierzy danych).

Uwzględnienie argumentu `na.rm=TRUE` w funkcji `data.Normalization` pozwala na jej zastosowanie w przypadku braków danych „NA". W tabeli 10.1 przedstawiono wzory na normalizację według zmiennych. Analogiczne wzory można przedstawić dla normalizacji według obiektów. Normalizacja według obiektów ma sens w przypadku, gdy wszystkie zmienne wyrażone są w tej samej jednostce miary. Taki przypadek zachodzi np. w badaniach strukturalnych.

Etap 3. Wybór miary odległości

Funkcja $d: A \times A \to R$ (zbiór liczb rzeczywistych) jest miarą odległości wtedy i tylko wtedy, gdy spełnione są warunki:

1) nieujemności: $d(A_i, A_k) \geq 0$ dla $i, k = 1, \ldots, n$;
2) zwrotności: $d(A_i, A_k) = 0 \Leftrightarrow i = k$ dla $i, k = 1, \ldots, n$;
3) symetryczności: $d(A_i, A_k) = d(A_k, A_i)$ dla $i, k = 1, \ldots, n$.

Jeśli ponadto spełniony jest warunek nierówności trójkąta $d(A_i, A_k) \leq d(A_i, A_l) + d(A_k, A_l)$ dla $i, k, l = 1, \ldots, n$, to miara odległości nazywana jest metryką.

Miary odległości obiektów opisanych zmiennymi mierzonymi na skalach metrycznych przedstawia tab. 10.2.

Tabela 10.2. Miary odległości obiektów opisanych zmiennymi mierzonymi na skalach metrycznych

Nazwa miary	Odległość d_{ik}	Skala pomiaru
Minkowskiego ($p \geq 1$)	$\sqrt[p]{\sum_{j=1}^{m} \lvert z_{ij} - z_{kj} \rvert^p}$	metryczne
▪ miejska (Manhattan) ($p = 1$)	$\sum_{j=1}^{m} \lvert z_{ij} - z_{kj} \rvert$	metryczne
▪ euklidesowa ($p = 2$)	$\sqrt{\sum_{j=1}^{m} (z_{ij} - z_{kj})^2}$	metryczne
▪ Czebyszewa (maximum) ($p \to \infty$)	$\max_j \lvert z_{ij} - z_{kj} \rvert$	metryczne
Canberra	$\sum_{j=1}^{m} \frac{\lvert z_{ij} - z_{kj} \rvert}{(z_{ij} + z_{kj})}$	ilorazowe
Braya-Curtisa	$\frac{\sum_{j=1}^{m} \lvert z_{ij} - z_{kj} \rvert}{\sum_{j=1}^{m} (z_{ij} + z_{kj})}$	ilorazowe
GDM1	$\frac{1}{2} - \frac{\sum_{j=1}^{m} (z_{ij} - z_{kj})(z_{kj} - z_{ij}) + \sum_{j=1}^{m} \sum_{\substack{l=1 \\ l \neq i,k}}^{n} (z_{ij} - z_{lj})(z_{kj} - z_{lj})}{2 \left[\sum_{j=1}^{m} \sum_{l=1}^{n} (z_{ij} - z_{lj})^2 \cdot \sum_{j=1}^{m} \sum_{l=1}^{n} (z_{kj} - z_{lj})^2 \right]^{0,5}}$	metryczne

$i, k, l = 1, \ldots, n$ – numery obiektów, $j = 1, \ldots, m$ – numer zmiennej; m – liczba zmiennych, $x_{ij}(x_{kj}, x_{lj})$ – i-ta (k-ta, l-ta) obserwacja na j-tej zmiennej; $z_{ij}(z_{kj}, z_{lj})$ – znormalizowana wartość j-tej zmiennej dla i-tego (k-tego, l-tego) obiektu.

Źródło: opracowanie własne.

Etap 4. Wybór metody klasyfikacji

Wśród metod klasyfikacji wyróżnia się m.in. metody hierarchiczne (aglomeracyjne i deglomeracyjne), metody optymalizujące wstępny podział zbioru obiektów, w których znana jest liczba wyodrębnianych klas, por. np. (Everitt i in., 2001, rozdz. 4 i 5), a także klasyfikację spektralną.

Hierarchiczne metody aglomeracyjne

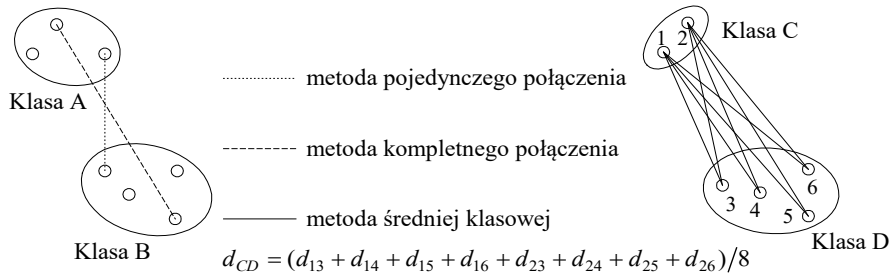
W praktycznych zastosowaniach metod klasyfikacji szczególne znaczenie mają hierarchiczne metody aglomeracyjne. Do niewątpliwych zalet tych metod należy zaliczyć to, że:

a) działają według jednej procedury (zwanej centralną procedurą aglomeracyjną);
b) wyniki klasyfikacji są przedstawione w postaci ciągu klasyfikacji (istnieje zatem możliwość kontrolowania procesu klasyfikacji);
c) wyniki klasyfikacji można przedstawić graficznie w formie dendrogramu (drzewka połączeń), wskazującego na kolejność połączeń między klasami. Uzyskana hierarchia umożliwia dokładne określenie, jak są wzajemnie usytuowane poszczególne klasy oraz obiekty w nich zawarte.

Hierarchiczna klasyfikacja aglomeracyjna rozpoczyna się od sytuacji, w której każdy obiekt badania A_i $(i = 1, \ldots, n)$ tworzy początkowo jedną klasę P_i. Algorytm centralnej procedury aglomeracyjnej jest następujący (Anderberg, 1973, s. 133; Gordon, 1987).

1. W macierzy odległości szuka się pary klas najbardziej podobnych (najmniej odległych od siebie). Załóżmy, że będą to klasy P_i oraz P_k.
2. Redukuje się liczbę klas o jeden, łącząc klasy P_i, P_k w nową.
3. Przekształca się odległości (stosownie do metody) pomiędzy połączonymi klasami P_i, P_k oraz pozostałymi klasami.
4. Powtarza się kroki 1–3, aż wszystkie obiekty znajdą się w jednej klasie.

Różnice w procedurach metod aglomeracyjnych wynikają z odmienności definiowania odległości międzyklasowej na etapie 3. Na rysunku 10.1 pokazano przykłady wyznaczania odległości międzyklasowej w przestrzeni dwuwymiarowej dla metody pojedynczego połączenia, kompletnego połączenia i średniej klasowej.



Rys. 10.1. Przykłady wyznaczania odległości międzyklasowej w przestrzeni dwuwymiarowej dla metody pojedynczego połączenia, kompletnego połączenia i średniej klasowej

Źródło: opracowanie własne.

Lance i Williams (1966; 1967; 1968) podali ogólny wzór na obliczanie odległości międzyklasowej. Odległość pomiędzy połączonymi klasami $P_i \cup P_k$ i inną klasą P_l jest zdefiniowana następująco (Everitt i in., 2001, s. 61):

$$d(P_i \cup P_k, P_l) =$$

$$\alpha_i d(P_i, P_l) + \alpha_k d(P_k, P_l) + \beta d(P_i, P_k) + \gamma |d(P_i, P_l) - d(P_k, P_l)|. \tag{10.4}$$

W formule (10.4) $\theta = (\alpha_i, \alpha_k, \beta, \gamma)$ oznacza zbiór parametrów, których wartości zależą od konkretnego wariantu metody aglomeracyjnej. W tabeli 10.3 zawarto wartości parametrów charakteryzujących hierarchiczne metody aglomeracyjne.

Tabela 10.3. Wartości parametrów dla hierarchicznych metod aglomeracyjnych

Symbol i nazwa metody	α_i	β	γ	Oznaczenie w pakiecie R
C.1. Pojedynczego połączenia (*single-link*)	0,5	0	−0,5	`single`
C.2. Kompletnego połączenia (*complete-link*)	0,5	0	0,5	`complete`
C.3. Średniej klasowej (*group average-link*)	$\frac{n_i}{n_i + n_k}$	0	0	`average`
C.4. Ważonej średniej klasowej (*weighted average-link*)	0,5	0	0	`mcquitty`
C.5. Powiększona suma kwadratów odległości (*incremental sum of squares*)	$\frac{n_i + n_l}{n_+}$	$-\frac{n_i}{n_+}$	0	`ward`
C.6. Środka ciężkości (*centroid*)	$\frac{n_i}{n_i + n_k}$	$\frac{-n_i n_k}{(n_i + n_k)^2}$	0	`centroid`
C.7. Medianowa (*median*)	0,5	−0,25	0	`median`
C.8. Giętka (*flexible*)	$0{,}5(1 - \beta)$	$\beta(< 1)$	0	`flexible`

$n_+ = n_i + n_k + n_l$, n_l – liczba obiektów w klasie P_l.

Źródło: opracowanie własne na podstawie (Gordon, 1996, s. 73; 1999, s. 79; Walesiak, 1993, s. 53).

Hierarchiczne metody deglomeracyjne

Klasyfikacja deglomeracyjna, zwana także klasyfikacją dedukcyjną, zstępującą lub klasyfikacją przez podział, rozpoczyna się od sytuacji, w której punktem wyjścia jest jedna klasa obejmująca wszystkie obiekty badania $A_1, \ldots, A_n$. W każdym kroku klasyfikacji liczba klas zwiększa się o jeden, przy czym jej zwiększenie następuje przez rozdzielenie jednej z istniejących klas. Po $n - 1$ krokach otrzymuje się liczbę klas równą liczbie obiektów badania, tzn. każdy obiekt tworzy jedną klasę.

Do grupy tej należą trzy metody Huberta działające według ogólnej procedury deglomeracyjnej oraz metody dendrytowe, tj. metodę taksonomii wrocławskiej (Florek i in., 1951; Perkal, 1953) i metoda najkrótszej sieci połączeń Prima (cyt. za: Sneath, 1957).

W tej części zostanie zaprezentowany algorytm metody zaproponowanej przez Macnaughtona-Smitha i in. (1964), w pakiecie R noszący nazwę DIANA (Kaufman i Rousseeuw, 1990, rozdz. 6), składający się z 7 kroków.

1. Dla każdej istniejącej klasy wyznaczyć parę obiektów najbardziej odległych (w pierwszej iteracji jest tylko jedna klasa obejmująca wszystkie obiekty badania). Do podziału wybieramy tę klasę, dla której odległość jest największa.
2. Dla każdego obiektu wybranej klasy P_s obliczamy średnią odległość od pozostałych obiektów tej klasy.
3. Obiekt, dla którego uzyskano maksymalną średnią odległość, tworzy zalążek nowej klasy A. Pozostałe obiekty tworzą tymczasową klasę B.
4. Dla każdego obiektu w tymczasowej klasie B obliczamy średnią odległość od pozostałych obiektów w klasie B i od obiektów w klasie A (odpowiednio $\bar{d}_{B_i}$ oraz $\bar{d}_{A_i}$).
5. Z tymczasowej klasy B trafia do tymczasowej klasy A obiekt, dla którego otrzymuje się $\max_i \left|\bar{d}_{B_i} - \bar{d}_{A_i}\right| > 0$.
6. Dla pozostałych obiektów z tymczasowej klasy B powtarzamy kroki 4 i 5. Proces podziału wybranej w kroku 1 klasy kończy się, gdy $\max_i \left|\bar{d}_{B_i} - \bar{d}_{A_i}\right| \leq 0$.
7. Kroki 1-6 powtarzane są $n-1$ razy, aż otrzyma się liczbę klas równą liczbie obiektów badania, tzn. każdy obiekt tworzy jedną klasę.

Metody optymalizujące wstępny podział zbioru obiektów

Ze względu na to, że liczba możliwych podziałów jest ogromna (zob. wzory (10.1) i (10.2)), nie jest możliwe znalezienie podziału optymalnego (optimum globalnego) z wykorzystaniem kryteriów, które są oparte na macierzy odległości lub macierzy danych (Gatnar i Walesiak, 2004, s. 331).

Algorytmy optymalizacyjne, mające charakter iteracyjny, opierają się więc na założeniu, że znany jest wstępny podział zbioru n obiektów na u klas. Zadaniem tych metod jest „poprawienie", z punktu widzenia zdefiniowanej funkcji-kryterium, wstępnego podziału zbioru obiektów. Do podstawowych metod optymalizacyjnych należą metoda k-średnich i metoda k-medoidów.

W metodzie k-średnich, w której każda klasa jest reprezentowana przez jej środek ciężkości (centroid), poszukuje się takiego podziału zbioru n obiektów na u klas, dla którego wartość miary $tr(\mathbf{W})$ ($\mathbf{W}$ – macierz kowariancji wewnątrzklasowej; tr – ślad macierzy) osiąga minimum ($\mathbf{W}= \sum_{s=1}^{u}\sum_{i=1}^{n_s}(\mathbf{z}_{si} - \bar{\mathbf{z}}_s)(\mathbf{z}_{si} - \bar{\mathbf{z}}_s)^T$, gdzie: $s = 1, \ldots, u$ – numer klasy, i – numer obiektu, n_s – liczba obiektów w klasie s, $\bar{\mathbf{z}}_s$ – m-wymiarowy wektor średnich w klasie s, m – liczba zmiennych, $\mathbf{z}_{si}$ – m-wymiarowy wektor obserwacji dla i-tego obiektu w klasie s).

W metodzie k-medoidów, w której każda klasa jest reprezentowana przez jeden z jej obiektów, będący gwiazdą klasy (*medoid, star*), poszukuje się takiego podziału zbioru n obiektów na u klas, dla którego wartość miary $C_1(n,u) = \sum_{r=1}^{u} O_3(r)$ osiąga minimum (gdzie $O_3(r) = \min_{k=1,\ldots,n_r}\left[\sum_{i=1}^{n_r}\left(d_{ri,rk}\right)^p\right]$, dla $p = 1$; $d_{ri,sk}$ – odległość między i-tym obiektem r-tej klasy i k-tym obiektem s-tej klasy; u – liczba klas; i, k – numery obiektów; $n_r(n_s)$ – liczba obiektów w klasie $r(s)$).

Metody k-średnich działają według następującego schematu (Aldenderfer i Blashfield, 1984, s. 45; Everitt i in., 2001, s. 99-100).

a. Punktem wyjścia jest wstępny podział zbioru obiektów na s klas, otrzymany np. przy użyciu dowolnej metody klasyfikacji lub ustalony losowo; dla każdej klasy wstępnego podziału oblicza się środki ciężkości oraz odległości każdego obiektu od środków ciężkości tych klas.

W pakiecie `clusterSim` dostępna jest funkcja `initial.Centers` pozwalająca ustalić początkowe środki ciężkości klas.

b. Zmienia się przyporządkowanie obiektów do klas o najbliższym środku ciężkości.
c. Oblicza się nowe środki ciężkości dla każdej klasy.
d. Powtarza się kroki b) i c) do chwili, gdy nie nastąpią przesunięcia obiektów między klasami (wartość funkcji-kryterium jakości klasyfikacji nie wykazuje istotnych zmian).

Metody *k*-średnich różnią się sposobami wyboru początkowych *k* środków ciężkości klas, stosowaną formułą odległości, strategią obliczania środków ciężkości klas. Wśród metod optymalizacji iteracyjnej Anderberg (1973, s. 156-175) wyróżnia metody: Forgy'ego, Janceya, *k*-średnich McQueena, *k*-średnich Wisharta, ISODATA Balla i Halla.

Algorytm metody *k*-medoids składa się z następujących etapów.

a. Wybierz arbitralnie lub za pomocą specjalnych procedur *k* obiektów stanowiących początkowych reprezentantów klas (*initial k medoids*).
b. Przydziel każdy pozostały obiekt ze zbioru *A* do klasy zawierającej najbliższego reprezentanta klasy (*medoid*).
c. Wprowadź w miejsce dotychczasowego reprezentanta klasy (*medoid*) inny obiekt, niebędący dotychczas reprezentantem klasy, pod warunkiem że uzyskana klasyfikacja się poprawi.
d. Powtórz kroki b) i c) do chwili, gdy nie nastąpią przesunięcia obiektów między klasami.

Szczegółowy algorytm metody *k*-medoids znajduje się w pracy Kaufmana i Rousseeuwa (1990, s. 102-104).

Klasyfikacja spektralna

Procedura klasyfikacji spektralnej (Walesiak, 2012; 2013; Walesiak i Dudek, 2009; 2010) obejmuje kolejno następujące kroki (klasyfikacja spektralna dla danych metrycznych zaproponowana została przez autorów Nga i in. (2002)).

1. Obliczenie symetrycznej macierzy podobieństw $\mathbf{A} = [A_{ik}]_{n\times n}$ (*affinity matrix*) między obiektami, dla której $A_{ii} = 0$ oraz

$$A_{ik} = exp(-\sigma \cdot d_{ik}) \text{ dla } i \neq k, \tag{10.5}$$

gdzie: σ – parametr skali, d_{ik} – miara odległości.

W kroku tym można zastosować do obliczenia elementów macierzy podobieństw A_{ik} $(i \neq k)$ estymatory jądrowe (zob. (Karatzoglou, 2006, s. 13-14) – funkcja `specc` pakietu `kernlab`; (Poland i Zeugmann, 2006) – jądro gaussowskie, jądro wielomianowe, jądro liniowe, jądro w postaci tangensa hiperbolicznego, jądro Bessela, jądro Laplace'a, jądro ANOVA, jądro łańcuchowe (dla danych tekstowych)).

W oryginalnym algorytmie klasyfikacji spektralnej dla danych metrycznych w pracy (Ng i in., 2002) zastosowano jądro gaussowskie:

$$A_{ik} = exp\left(-\frac{d_{ik}^2}{2\sigma^2}\right) \text{ dla } i \neq k, \tag{10.6}$$

gdzie: $d_{ik} = \sqrt{\sum_{j=1}^{m}(z_{ij} - z_{kj})^2}$, z_{ij}, z_{kj} – znormalizowana wartość j-tej zmiennej dla i-tego i k-tego obiektu.

2. Konstrukcja znormalizowanej macierzy Laplace'a $\mathbf{L} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ ($\mathbf{D}$ – diagonalna macierzy wag, w której na głównej przekątnej znajdują się sumy każdego wiersza z macierzy $\mathbf{A} = [A_{ik}]$, a poza główną przekątną są zera). W rzeczywistości znormalizowana macierz Laplace'a przyjmuje postać: $\mathbf{I} - \mathbf{L}$. Własności tej macierzy przedstawiono m.in. w pracy (Von Luxburg, 2006, s. 5). W algorytmie dla uproszczenia analizy pomija się macierz jednostkową $\mathbf{I}$.
3. Obliczenie wartości własnych i odpowiadających im wektorów własnych (o długości równej jeden) dla macierzy $\mathbf{L}$. Uporządkowanie wektorów własnych według malejących wartości własnych. Pierwsze u wektorów własnych (u – liczba klas) tworzy macierz $\mathbf{E} = [e_{ij}]$ o wymiarach $n \times u$.

Podobnie jak w przypadku klasycznym analizy skupień zachodzi potrzeba ustalenia optymalnej liczby klas. Odpowiedni algorytm zaproponował Girolami (2002).

Macierz podobieństw (*affinity matrix*) $\mathbf{A} = [A_{ik}]$ (dla $\sigma = 1$) poddawana jest dekompozycji $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, gdzie $\mathbf{U}$ jest macierzą wektorów własnych macierzy $\mathbf{A}$ składającą się z wektorów $\mathbf{u}_1, \ldots, \mathbf{u}_n$, a $\mathbf{\Lambda}$ jest macierzą diagonalną zawierającą wartości własne $\lambda_1, \ldots, \lambda_n$. Zatem możemy zapisać, że $\mathbf{1}_n^T\mathbf{A}\mathbf{1}_n = \mathbf{1}_n^T\{\sum_{i=1}^{n}\lambda_i\mathbf{u}_i\,\mathbf{u}_i^T\}\mathbf{1}_n = \sum_{i=1}^{n}\lambda_i\{\mathbf{1}_n^T\mathbf{u}_i\}^2$ ($\mathbf{1}_n^T$ – wektor o wymiarach $1 \times n$ zawierający wartości $1/n$). Graficzny rozkład wartości $\lambda_i\{\mathbf{1}_n^T\mathbf{u}_i\}^2$ porządkowany malejąco wyznacza optymalną liczbę skupień u (przez wyznaczenie dominujących elementów), na którą algorytm klasyfikacji spektralnej powinien podzielić zbiór badanych obiektów.

4. Przeprowadza się normalizację macierzy $\mathbf{E}$ zgodnie ze wzorem $y_{ij} = e_{ij} / \sqrt{\sum_{j=1}^{u} e_{ij}^2}$ ($i = 1, \ldots, n$ – numer obiektu, $j = 1, \ldots, u$ – numer zmiennej, u – liczba klas), otrzymując macierz $\mathbf{Y} = [y_{ij}]$ o wymiarach $n \times u$. Dzięki tej normalizacji długość każdego wektora wierszowego macierzy $\mathbf{Y} = [y_{ij}]$ jest równa jeden. W klasyfikacji spektralnej wektory własne są wykorzystywane do utworzenia nowej przestrzeni. Obiekty z pierwotnej macierzy danych $\mathbf{Z} = [z_{ij}]_{nxm}$ są odwzorowywane w nowej przestrzeni $\mathbf{Y} = [y_{ij}]_{nxu}$ na podstawie wektorów własnych. Ta przestrzeń jest zasadniczo inna, ponieważ osie nie są tutaj oryginalnymi zmiennymi, ale wymiarami, które zapewniają lepszą separowalność klas obiektów. Graficzną prezentację wybranych kroków klasyfikacji spektralnej (odpowiednio wizualizacja obiektów z macierzy $\mathbf{E} = [e_{ij}]$ oraz obiektów ze znormalizowanej macierzy $\mathbf{Y} = [y_{ij}]$) dla danych metrycznych przedstawiających strukturę dwóch klas (wizualizacja obiektów z macierzy $\mathbf{Z} = [z_{ij}]$) przedstawiono w pracy Walesiaka (2016, s. 76).
5. Macierz $\mathbf{Y}$ stanowi punkt wyjścia zastosowania klasycznych metod analizy skupień (proponuje się tutaj wykorzystanie metody k-średnich).

Parametr σ ma fundamentalne znaczenie w klasyfikacji spektralnej. W literaturze zaproponowano wiele heurystycznych sposobów wyznaczania wartości tego parametru, por. np. (Fischer i Poland, 2004; Poland i Zeugmann, 2006; Zelnik-Manor i Perona, 2004). W metodach heurystycznych wyznacza się wartość σ na podstawie pewnych statystyk opisowych macierzy odległości $[d_{ik}]$. Lepszy sposób wyznaczania parametru σ zaproponował Karatzoglou (2006). Poszukuje się takiej wartości parametru σ, która minimalizuje wewnątrzklasową sumę kwadratów odległości przy zadanej

liczbie klas u. Jest to heurystyczna metoda poszukiwania minimum lokalnego. Zbliżony koncepcyjnie algorytm znajdowania optymalnego parametru σ zaproponowali Walesiak i Dudek (2009).

Etap 5. Ustalenie liczby klas

Milligan i Cooper (1985) przetestowali na podstawie zbiorów danych o znanej strukturze klas 30 procedur pozwalających wyznaczyć liczbę klas. Przedstawiony przegląd nie wyczerpuje zbioru istniejących sposobów wyznaczania liczby klas. Między innymi po 2000 r. zaproponowano następujące indeksy oceny jakości klasyfikacji służące wyborowi liczby klas: *gap* (Tibshirani i in., 2001); *jump* (Sugar i James, 2003); *ps* (Tibshirani i Walther, 2005); *clest* (Dudoit i Fridlyand, 2002). Tabela 10.4 zawiera charakterystykę 8 indeksów często wykorzystywanych w praktyce i analizach porównawczych.

Tabela 10.4. Indeksy oceny jakości klasyfikacji służące wyborowi liczby klas

Lp.	Nazwa indeksu	Formuła	Kryterium wyboru liczby klas
1	Calińskiego i Harabasza	$G1(u) = \frac{B_u/(u-1)}{W_u/(n-u)}$, $G1(u) \in R_+$	$\hat{u} = \arg\max_u\{G1(u)\}$
2	Bakera i Huberta	$G2(u) = \frac{s(+)-s(-)}{s(+)+s(-)}$, $G2(u) \in [-1;1]$	$\hat{u} = \arg\max_u\{G2(u)\}$
3	Huberta i Levina	$C(u) = \frac{D(u) - D_{\min}}{D_{\max} - D_{\min}}$	$\hat{u} = \arg\min_u\{C(u)\}$
4	*silhouette* (sylwetkowy)	$S(u) = \frac{1}{n}\sum_{i=1}^{n} \frac{b(i)-a(i)}{\max\{a(i);b(i)\}}$, $S(u) \in [-1;1]$	$\hat{u} = \arg\max_u\{S(u)\}$
5	Krzanowskiego i Lai	$KL(u) = \left\lvert\frac{\mathrm{DIFF}_u}{\mathrm{DIFF}_{u+1}}\right\rvert$, $KL(u) \in R_+$ $\mathrm{DIFF}_u = (u-1)^{2/m}W_{u-1} - u^{2/m}W_u$	$\hat{u} = \arg\max_u\{KL(u)\}$
6	Daviesa-Bouldina	$DB(u) = \frac{1}{u}\sum_{r=1}^{u} \max_{r\neq s}\left(\frac{S_r + S_s}{d_{rs}}\right)$	$\hat{u} = \arg\min_u\{DB(u)\}$
7	Hartigana	$H(u) = \left(\frac{W_u}{W_{u+1}} - 1\right)(n-u-1)$, $H(u) \in R_+$	najmniejsze u, dla którego $H(u) \leq 10$
8	gap	$\mathrm{Gap}(u) = \frac{1}{B}\sum_{b=1}^{B} \log W_{ub} - \log W_u$, $\mathrm{Gap}(u) \in R$	najmniejsze u, dla którego $diffu \geq 0$

$\mathbf{B}_u$ – macierz kowariancji międzyklasowej; $\mathbf{W}_u$ – macierz kowariancji wewnątrzklasowej; tr – ślad macierzy; $B_u(W_u) = tr\mathbf{B}_u(tr\mathbf{W}_u)$; $r, s = 1, \ldots, u$ – numer klasy; u – liczba klas; $i, k = 1, \ldots, n$ – numer obiektu; n – liczba obiektów; m – liczba zmiennych; $s(+)$ – liczba par odległości zgodnych; $s(-)$ – liczba par odległości niezgodnych; $D(u)$ – suma wszystkich odległości wewnątrzklasowych przy podziale na u klas (jest r takich odległości); $D_{\min}(D_{\max})$ – suma r najmniejszych (największych) odległości w macierzy odległości; $a(i) = \sum_{k\in\{P_r\setminus i\}} d_{ik}/(n_r - 1)$ – średnia odległość obiektu i od pozostałych obiektów należących do klasy P_r; $b(i) = \min_{s\neq r}\{d_{iP_s}\}$, $d_{iP_s} = \sum_{k\in P_s} d_{ik}/n_s$ – średnia odległość obiektu i od obiektów należących do klasy P_s; B – liczba generowanych zbiorów obserwacji; $s_u = sd_u\sqrt{1 + 1/B}$, sd_u – odchylenie standardowe z wartości $\{\log W_{ub}\}$; $diffu = \mathrm{Gap}(u) - \mathrm{Gap}(u+1) + s_{u+1}$; $d_{rs} = \sqrt[p]{\sum_{j=1}^{m}\lvert z_{rj} - z_{sj}\rvert^p}$ – odległość między środkami ciężkości (medoidami) klas r i s ($p = 1$ – odległość miejska, $p = 2$ – odległość Euklidesa); $S_r = \sqrt[q]{\frac{1}{n_r}\sum_{i\in P_r}\sum_{j=1}^{m}\lvert x_{ij}^r - z_{rj}\rvert^q}$ – miara rozproszenia obiektów w klasie: $q = 1$ – średnia odległość obiektów w r-tej klasie od środka ciężkości (medoidy) klasy; $q = 2$ – odchylenie standardowe odległości obiektów w r-tej klasie od środka ciężkości (medoidy) klasy.

Źródło: opracowanie własne na podstawie (Caliński i Harabasz, 1974; Davies i Bouldin, 1979; Hartigan, 1975; Hubert, 1974; Hubert i Levin, 1976; Kaufman i Rousseeuw, 1990; Milligan i Cooper, 1985; Tibshirani i in., 2001).

Etap 6. Ocena wyników klasyfikacji

Testowanie braku struktury klas

Z logicznego punktu widzenia testowanie braku struktury klas powinno być na etapie trzecim procesu klasyfikacji (po ustaleniu zbioru obiektów, zbioru zmiennych i zebraniu danych statystycznych). W literaturze jednak testowanie braku struktury klas przeprowadza się na etapie oceny wyników klasyfikacji. Wynika to z ograniczonej użyteczności dostępnych testów (Everitt i in., 2001, s. 180), które w praktyce zwykle nie są stosowane.

W konstrukcji hipotezy zerowej mówiącej o braku struktury klas w badanym zbiorze obiektów wykorzystywane są m.in. modele (Gordon, 1999, s. 186-188):

- Poissona (zakładający, że obiekty są reprezentowane przez punkty, które są jednostajnie rozłożone w pewnym regionie m-wymiarowej przestrzeni),
- jednomodalny (zakładający, że m-wymiarowe obserwacje wygenerowane są z jednomodalnego rozkładu częstości),
- losowej macierzy odległości (zakładający, że elementy dolnego trójkąta macierzy odległości są uszeregowane w losowym porządku; wszystkie $(n(n-1)/2)!$ rankingi odległości są jednakowo prawdopodobne).

Zagadnienie testowania struktury klas omówili: Arnold (1979, s. 545-551), Bock (1996, s. 377-453); Everitt i in. (2001, s. 180-181), Gordon (1998, s. 22-39; 1999, s. 185-190).

Analiza replikacji (powtórzenie klasyfikacji)

Replikacja dotyczy przeprowadzenia procesu klasyfikacji zbioru obiektów na podstawie dwóch prób wylosowanych z danego zbioru danych, a następnie oceny zgodności otrzymanych rezultatów. Formalnie analizę replikacji zaproponowali McIntyre i Blashfield (1980) oraz Morey i in. (1983).

Procedura replikacji składa się z następujących etapów (Breckenridge, 2000, s. 262-263; Gordon, 1999, s. 184; Milligan, 1996, s. 368-369; Walesiak, 2008):

1. Podzielić losowo zbiór danych (zbiór n obiektów opisanych m zmiennymi) na dwa podzbiory A (podstawowy) i B (replikacyjny) opisane tym samym zbiorem zmiennych.
2. Zastosować wybraną metodę klasyfikacji (np. metodę k-średnich, k-medoidów, hierarchiczne metody aglomeracyjne) do podziału zbioru A na ustaloną liczbę klas u. Wcześniej należy podjąć decyzję dotyczącą wyboru formuły normalizacji wartości zmiennych, miary odległości oraz liczby skupień. Wyznaczyć dla danych metrycznych środki ciężkości (*centroids*) dla poszczególnych klas lub obiekty reprezentatywne dla klas (usytuowane centralnie zwane *centrotypes* lub *medoids*) dla danych metrycznych lub niemetrycznych. Obiektem usytuowanym centralnie w klasie (medoid) jest ten, dla którego suma odległości od pozostałych obiektów danej klasy jest najmniejsza.
3. Tę samą procedurę klasyfikacyjną zastosować do podziału zbioru B na u klas.
4. Obliczyć odległości obiektów ze zbioru B od środków ciężkości klas lub od obiektów usytuowanych centralnie w klasach wyznaczonych na podstawie podzbioru A. Przydzielić obiekty z podzbioru B do klas zawierających najbliższy środek ciężkości (najbliższą medoidę).

Prowadzi to do podziału podzbioru B na nie więcej niż u klas. Otrzymujemy podział zbioru B na klasy na podstawie charakterystyk (środki ciężkości lub medoidy) zbioru A.

5. Powtórzyć kroki 1-4 S razy (S – liczba symulacji).
6. Obliczyć, np. za pomocą skorygowanej miary Randa (Hubert i Arabie, 1985), średnią zgodność wyników dwóch podziałów podzbioru B. Poziom zgodności wyników dwóch podziałów podzbioru B odzwierciedla poziom stabilności przeprowadzonej klasyfikacji zbioru obiektów.

Ocena jakości klasyfikacji

Syntetyczny miernik pozwalający mierzyć m.in. ogólną jakość klasyfikacji (relatywną zwartość i separowalność klas) zaproponował Rousseeuw (1987), zob. (Kaufman i Rousseeuw, 1990, s. 83-88), zob. tab. 10.4. Indeks $S(u)$ przybiera wartości z przedziału $[-1; 1]$. Subiektywną ocenę przedziałów wartości miernika $S(u)$ zawiera tab. 10.5.

Tabela 10.5. Interpretacja wartości miernika $S(u)$

$S(u)$	Interpretacja
(0,70; 1,00]	silna struktura klas
(0,50; 0,70]	poważna struktura klas
(0,25; 0,50]	słaba struktura klas (należy zastosować inne metody klasyfikacji)
0,25 i mniej	nie odkryto struktury klas

Źródło: (Kaufman i Rousseeuw, 1990, s. 88).

Etap 7. Opis (interpretacja) i profilowanie klas

W wyniku zastosowania do klasyfikacji zbioru obiektów wybranej metody klasyfikacji otrzymuje się podział tego zbioru na klasy $P_1, \ldots, P_u$. W badaniach podstawowymi zagadnieniami stają się w związku z tym: opis (interpretacja) otrzymanych wyników i profilowanie klas.

Opis (interpretacja) otrzymanych wyników jest wskazaniem cech charakterystycznych poszczególnych klas oraz wyjaśnieniem, jakimi czynnikami różnią się wyodrębnione klasy. Podstawą opisu (interpretacji) wyodrębnionych klas są zmienne, które brały udział w procesie klasyfikacji zbioru obiektów.

W celu ułatwienia interpretacji otrzymanych rezultatów klasyfikacji wyznacza się środki ciężkości poszczególnych klas (średnie arytmetyczne obliczone z wartości pierwotnych każdej zmiennej na podstawie obiektów tworzących daną klasę) oraz odchylenia standardowe zmiennych w poszczególnych klasach. Na ten sposób rozwiązania problemu interpretacji rezultatów klasyfikacji wskazują m.in.: Hair i in. (1995, s. 443), Jajuga (1990, s. 134), Robles i Sarathy (1986) oraz Sokołowski (1992, s. 47). Oczywiście taki sposób opisu klas możliwy jest do zastosowania tylko wtedy, gdy zmienne użyte w zagadnieniu klasyfikacji zbioru obiektów są mierzone na skali przedziałowej lub ilorazowej (dla tych skal dopuszcza się użycie średniej arytmetycznej i odchylenia standardowego).

Jeśli klasyfikacja jest przeprowadzana na podstawie zmiennych mierzonych na skali porządkowej lub nominalnej, to możliwe jest wyznaczenie opisowej (werbalnej) charakterystyki poszczególnych

klas dla każdej zmiennej. Można wyznaczyć frakcje i odsetki występowania w danej klasie poszczególnych kategorii zmiennych.

Profilowanie klas. Celem profilowania klas jest wskazanie cech charakterystycznych poszczególnych klas pozwalających na wskazanie różnic pomiędzy nimi. Profilowanie klas przeprowadza się na podstawie zmiennych, które nie brały udziału w procesie klasyfikacji zbioru obiektów.

Typowymi zmiennymi stosowanymi w profilowaniu klas np. w badaniach marketingowych są zmienne demograficzne, geograficzne, socjoekonomiczne, psychograficzne i in., które charakteryzują konsumentów (nabywców) poszczególnych klas. Profilowanie przeprowadza się zwykle z wykorzystaniem takich metod, jak (Hair i in., 1998, s. 501, 513-515; Sagan, 1998, s. 180): analiza dyskryminacyjna, drzewa klasyfikacyjne, tabulacja krzyżowa (tablice kontyngencji).

10.3. Podstawowe pakiety i funkcje programu R

W tabeli 10.6 pokazano wybrane pakiety i funkcje programu R (R Core Team, 2023) wykorzystywane na poszczególnych etapach analizy skupień.

Tabela 10.6. Etapy w analizie skupień oraz funkcje programu R

Lp.	Etapy w analizie skupień	Wybrane pakiety i funkcje programu R
1	Wybór obiektów i zmiennych	Pakiet `clusterSim` (funkcja `HINoV.Mod`)
2	Wybór formuły normalizacji wartości zmiennych	Pakiet `clusterSim` (funkcja `data.Normalization`)
3	Wybór miary odległości	Pakiet `clusterSim` (funkcje: `dist.BC, dist.GDM, dist.SM`) Pakiet `stats` (funkcja `dist`) Pakiet `ade4` (funkcja `dist.binary`)
4	Wybór metody klasyfikacji	Pakiet `cluster` (funkcje: `agnes, diana, pam`) Pakiet `stats` (funkcje: `kmeans, hclust`) Pakiet `kernlab` (funkcja `specc`) Pakiet `clusterSim` (funkcje: `speccl, initial.Centers`)
5	Ustalenie liczby klas	Pakiet `clusterSim` (funkcje: `index.G1, index.G2, index.C, index.S, index.KL, index.H, index.Gap, index.DB`)
6	Ocena wyników klasyfikacji	Pakiet `clusterSim` (`funkcja replication.Mod`)
7	Opis (interpretacja) i profilowanie klas	Pakiet `clusterSim` (funkcja `cluster.Description`)

Źródło: opracowanie własne.

10.4. Zastosowania z wykorzystaniem programu R dla danych z BDL

Analizę skupień zastosowano do klasyfikacji województw Polski ze względu na zanieczyszczenie powietrza w 2021 r. opisane za pomocą następujących zmiennych:

x1 – gęstość sieci drogowej (drogi publiczne o twardej nawierzchni) w km na 100 km^2 powierzchni,
x2 – samochody osobowe zarejestrowane na 1000 ludności,
x3 – emisja zanieczyszczeń pyłowych w tonach na 1 km^2 powierzchni,

x4 – emisja dwutlenku siarki w tonach na 1 km^2 powierzchni,
x5 – emisja tlenku azotu w tonach na 1 km^2 powierzchni,
x6 – emisja tlenku węgla w tonach na 1 km^2 powierzchni.

W tabeli 10.7 przedstawiono zmienne dostępne w Banku Danych Lokalnych oraz wymagane przeliczenia.

Tabela 10.7. Zmienne dostępne w BDL oraz wymagane przeliczenia

Zmienne dostępne w BDL	ID	Wymagane przeliczenia
v1 – gęstość sieci drogowej (drogi publiczne o twardej nawierzchni) w km na 100 km^2 powierzchni	60524	x1 = v1
v2 – samochody osobowe zarejestrowane na 1000 ludności	60533	x2 = v2
v3 – emisja zanieczyszczeń pyłowych w tonach na 1 km^2 powierzchni	458168	x3 = v3
v4 – emisja dwutlenku siarki w tonach	2084	x
v5 – emisja tlenku azotu w tonach	2085	x
v6 – emisja tlenku węgla w tonach	2086	x
v7 – powierzchnia w km^2	2018	x
x4 – emisja dwutlenku siarki w tonach na 1 km^2 powierzchni	x	x4 = v4/v7
x5 – emisja tlenku azotu w tonach na 1 km^2 powierzchni	x	x5 = v5/v7
x6 – emisja tlenku węgla w tonach na 1 km^2 powierzchni	x	x6 = v6/v7

Źródło: opracowanie własne.

Skrypt 10.1 pozwala na automatyczne pozyskanie danych z Banku Danych Lokalnych.

Skrypt 10.1

```
library(bdl)
library(dplyr)
options(OutDec=",")
# stałe
rok<-2021
no_uL=2 # (2 - województwo)
# Wczytanie informacji o zmiennych z BDL
v1<-get_data_by_variable("60524",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x1=val)
v2<-get_data_by_variable("60533",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x2=val)
v3<-get_data_by_variable("458168",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x3=val)
v4<-get_data_by_variable("2084",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x4=val)
v5<-get_data_by_variable("2085",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x5=val)
v6<-get_data_by_variable("2086",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x6=val)
v7<-get_data_by_variable("2018",unitLevel=no_uL,year=rok) %>%
  dplyr::select(id,name, x7=val)
```

```
# Połączenie zmiennych w jedną tabelę (ramkę) danych
data <- v1 %>%
  full_join (v2, by=c("id","name")) %>%
  full_join (v3, by=c("id","name")) %>%
  full_join (v4, by=c("id","name")) %>%
  full_join (v5, by=c("id","name")) %>%
  full_join (v6, by=c("id","name")) %>%
  full_join (v7, by=c("id","name"))
v<-as.data.frame(data)
# Nadanie nazw wierszom
nazwy<-v[,2]
rownames(v)<-tolower(nazwy)
# Wybranie 7 kolumn z danymi oraz dodanie 3 pustych na przeliczenia
v<-v[,3:9]
for(i in 1:3){
  v<-cbind(v,rep(0,nrow(v)))
}
# Nadanie nazw kolumnom
colnames(v)<-paste("v",1:10,sep="")
# Przeliczenia
v<-mutate(v,v8=round((v4/v7),3))
v<-mutate(v,v9=round((v5/v7),3))
v<-mutate(v,v10=round((v6/v7),3))
# Usunięcie zbędnych kolumn
x<-dplyr::select(v,-c(4:7))
# Nadanie nazw wierszom i kolumnom
colnames(x)<-paste("x",1:6,sep="")
rownames(x)<-tolower(nazwy)
print(x)
# Zapisanie danych do pliku
write.table(x, file="dane_zp_2022.csv",sep=";",dec=",",
  row.names=TRUE, col.names=NA)
```

Po wykonaniu skryptu 10.1 pobrany zostaje zbiór danych z BDL dla 6 zmiennych opisujących poziom zanieczyszczenia powietrza według województw w 2021 r. (dane zostają zapisane do pliku `dane_zp_2022.csv`). Po wykonaniu polecenia `print(x)` otrzymujemy:

```
                      x1    x2   x3    x4    x5    x6
małopolskie         174,4 634,7 0,08 0,428 0,699 0,941
śląskie             178,1 654,3 0,35 2,417 2,402 8,836
lubuskie             64,5 742,5 0,04 0,113 0,201 0,139
wielkopolskie       104,7 735,7 0,05 0,240 0,302 0,149
zachodniopomorskie   62,2 657,0 0,06 0,270 0,241 0,132
dolnośląskie        107,4 699,4 0,07 0,508 0,616 0,248
opolskie             92,9 735,3 0,13 0,682 1,567 2,038
kujawsko-pomorskie  107,8 680,4 0,10 0,374 0,496 0,415
pomorskie            83,1 660,0 0,04 0,297 0,290 0,245
warmińsko-mazurskie  59,4 630,8 0,02 0,146 0,103 0,131
łódzkie             117,8 698,0 0,09 2,694 1,748 1,717
świętokrzyskie      126,2 655,2 0,11 1,449 0,960 3,064
lubelskie            96,8 683,6 0,05 0,152 0,220 0,149
podkarpackie         99,7 619,6 0,05 0,183 0,194 0,231
podlaskie            71,8 599,9 0,02 0,071 0,117 0,149
mazowieckie         114,1 742,4 0,06 0,643 0,672 0,335
```

Na podstawie danych z pliku `dane_zp_2022.csv` przeprowadzono klasyfikację 16 województw ze względu na zanieczyszczenie powietrza w 2021 r. Przeprowadzając analizę skupień w składni poleceń dla skryptu 10.2, przyjęto następującą metodykę postępowania:

- zastosowano standaryzację pozycyjną (`type="n2"`) do normalizacji wartości zmiennych; wszystkie zmienne są mierzone na skali ilorazowej;
- do wyznaczenia macierzy odległości zastosowano miarę odległości GDM1 dla danych metrycznych (funkcja `dist.GDM`, dla której `method="GDM1"`);
- zastosowano metodę klasyfikacji `pam` w powiązaniu z indeksem oceny jakości klasyfikacji gap (`index.Gap`) do podziału zbioru obiektów na klasy względnie jednorodne.

Skrypt 10.2

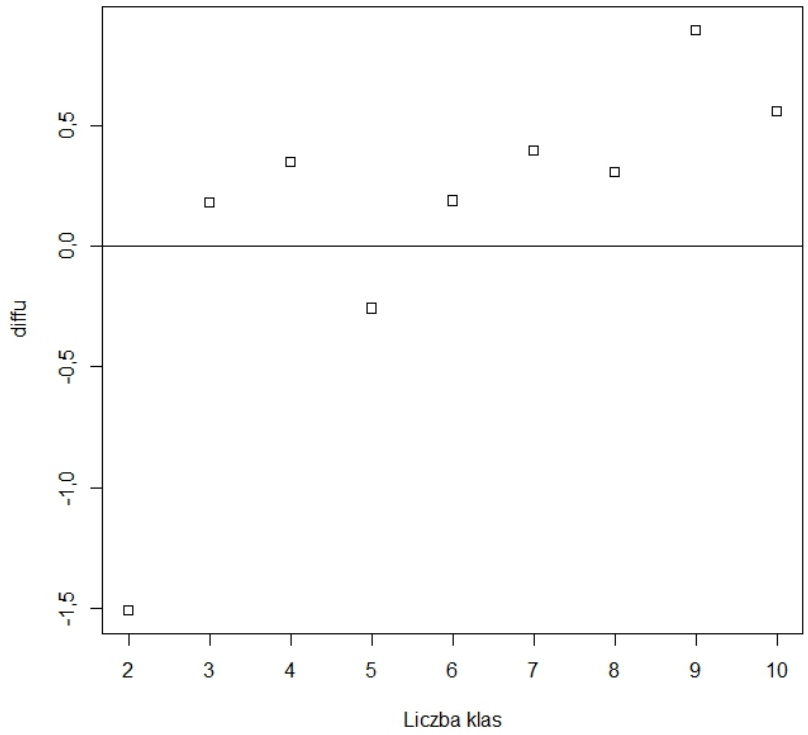
```
library(clusterSim)
set.seed(123) # Ustawienie generatora liczb losowych
# Wczytanie zbioru danych
xx<-read.csv2("dane_zp_2022.csv",header=TRUE)
x<-as.matrix(xx[,2:ncol(xx)])
options(OutDec=",")
# Wybór metody normalizacji wartości zmiennych
z<-data.Normalization(x,type="n2")
# Wybór miary odległości
z<-as.data.frame(z)
d<-dist.GDM(z,method="GDM1")
print("Ustalenie liczby klas z wykorzystaniem indeksu gap",quote=FALSE)
min_liczba_klas<-2
max_liczba_klas<-10
min<-0
clopt<-NULL
wyn<-NULL
wyniki<-array(0,c(max_liczba_klas-min_liczba_klas+1,2))
wyniki[,1]<-min_liczba_klas:max_liczba_klas
znaleziono<-FALSE
for(liczba_klas in min_liczba_klas:max_liczba_klas){
  cl1<-pam(d,liczba_klas,diss=TRUE)
  cl2<-pam(d,liczba_klas+1,diss=TRUE)
  clall<-cbind(cl1$clustering,cl2$clustering)
  Gap<-index.Gap(z,clall,reference.distribution="pc",B=10,method="pam")
  wyniki[liczba_klas-min_liczba_klas+1,2]<-diffu<-Gap$diffu
  if((wyniki[liczba_klas-min_liczba_klas+1,2]>=0)&&(!znaleziono)){
    lk<-liczba_klas
    min<-diffu
    clopt<-cl1$cluster
    wyn<-cl1$clusinfo
    znaleziono<-TRUE
  }
}
if (znaleziono){
print(paste("Minimalna liczba klas dla diffu>=0
    wynosi",lk,"dla diffu=",round(min,4)),quote=FALSE)
}else{
  print("Nie znalazłem klasyfikacji,dla której diffu>=0",quote=FALSE)
}
# Zapisanie do pliku diffu.csv wartości indeksu gap
```

```
write.table(wyniki,file="diffu.csv",sep=";",dec=",",
  row.names=TRUE,col.names=FALSE)
print("Prezentacja klasyfikacji wynikowej",quote=FALSE)
cl_wyn1<-data.frame(xx[,1],clopt)
colnames(cl_wyn1) <- c("województwa", "klasa")
print(cl_wyn1)
print("Prezentacja klasyfikacji wynikowej - uporządkowana", quote=FALSE)
cl_wyn2<-cl_wyn1[order(cl_wyn1["klasa"],decreasing=FALSE),]
cl_wyn2<-data.frame(cl_wyn2)
print(cl_wyn2)
# Zapisanie do pliku clusinfo.csv dodatkowych informacji o wyodrębnionych klasach
write.table(wyn,file="clusinfo.csv",sep=";",dec=",",row.names=TRUE,col.names=NA)
plot(wyniki,type="p",pch=0,xlab="Liczba klas",ylab="diffu",xaxt="n")
abline(h=0,untf=FALSE)
axis(1,c(min_liczba_klas:max_liczba_klas))
desc<-cluster.Description(x,clopt,"population")
print("Średnie arytmetyczne",quote=FALSE)
print(desc[,,1])
print("Odchylenia standardowe",quote=FALSE)
print(desc[,,2])
```

W wyniku zastosowania skryptu 10.2 otrzymano następujące wyniki klasyfikacji:

- ustalono liczbę klas z wykorzystaniem indeksu gap (rys. 10.2):

```
[1] Minimalna liczba klas dla diffu>=0 wynosi 3 dla diffu= 0,1824
```



Rys. 10.2. Graficzna prezentacja wartości indeksu gap

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

- zapisano przyporządkowanie województw do wyodrębnionych klas:

```
   [1] Prezentacja klasyfikacji wynikowej
           województwa klasa
1          małopolskie     1
2              śląskie     2
3             lubuskie     1
4        wielkopolskie     1
5    zachodniopomorskie     1
6         dolnośląskie     1
7              opolskie     3
8    kujawsko-pomorskie     1
9             pomorskie     1
10 warmińsko-mazurskie     1
11             łódzkie     3
12      świętokrzyskie     3
13           lubelskie     1
14         podkarpackie     1
15           podlaskie     1
16         mazowieckie     1
[1] Prezentacja klasyfikacji wynikowej - uporządkowana
           województwa klasa
1          małopolskie     1
3             lubuskie     1
4        wielkopolskie     1
5    zachodniopomorskie     1
6         dolnośląskie     1
8    kujawsko-pomorskie     1
9             pomorskie     1
10 warmińsko-mazurskie     1
13           lubelskie     1
14         podkarpackie     1
15           podlaskie     1
16         mazowieckie     1
2              śląskie     2
7             opolskie     3
11             łódzkie     3
12      świętokrzyskie     3
```

- wyznaczono i zinterpretowano charakterystyki dla poszczególnych klas (średnie arytmetyczne oraz odchylenia standardowe):

```
[1] Średnie arytmetyczne
```

```
[1] Średnie arytmetyczne
     [,1]      [,2]       [,3]     [,4]     [,5]     [,6]
[1,] "95,4917" "673,8333" "0,0533" "0,2854" "0,3459" "0,272"
[2,] "178,1"   "654,3"    "0,35"   "2,417"  "2,402"  "8,836"
[3,] "112,3"   "696,1667" "0,11"   "1,6083" "1,425"  "2,273"
```

Podkreślenia oznaczają wartości minimalne, a pogrubienia wartości maksymalne dla poszczególnych zmiennych w klasach.

```
[1] Odchylenia standardowe
```

```
   [1] Odchylenia standardowe
     [,1]      [,2]      [,3]     [,4]     [,5]     [,6]
[1,] "30,3543" "46,8166" "0,0221" "0,1662" "0,2069" "0,2194"
[2,] "0"       "0"       "0"      "0"      "0"      "0"
[3,] "14,14"   "32,7264" "0,0163" "0,8291" "0,337"  "0,5745"
```

W klasie jednoelementowej o numerze dwa znalazło się województwo śląskie o najwyższym poziomie zanieczyszczenia powietrza. Klasa pierwsza obejmuje 12 województw o najniższym poziomie zanieczyszczenia powietrza. Pozostałe 3 województwa trafiły do klasy trzeciej. Charakteryzują się one średnimi poziomami zanieczyszczenia powietrza.

Wykorzystując analizę replikacji (skrypt 10.3), oceniono poziom stabilności przeprowadzonej klasyfikacji zbioru obiektów.

Skrypt 10.3

```
library(clusterSim)
lk<-3 # Liczba klas ustalona w skrypcie 10.2
nor<-"n2" # Metoda normalizacyjna zastosowana w skrypcie 10.2
odl<-"d5" # Miara odległości zastosowana w skrypcie 10.2
set.seed(123)  # Ustawienie generatora liczb losowych
x<-read.csv2("dane_zp_2022.csv",header=TRUE,row.names=1)
x<-as.matrix(x)
options(OutDec=",")
w<-replication.Mod(x,v="m",u=lk,centrotypes="centroids",normalization=nor,
  distance=odl,method="pam",S=20)
print(w$cRand)
```

W wyniku zastosowania skryptu 10.3 otrzymano następujący wynik:

```
[1] 0,6035515
```

Poziom wartości skorygowanej miary Randa odzwierciedla stabilność przeprowadzonej klasyfikacji zbioru obiektów:

$[-\infty; 0,2]$ – podział niestabilny,
$(0,2; 0,4]$ – podział mało stabilny,
$(0,4; 0,6]$ – podział relatywnie stabilny,
$(0,6; 0,8]$ – podział stabilny,
$(0,8; 1,0]$ – podział wysoce stabilny.

Otrzymana wartość skorygowanej miary Randa świadczy o stabilnym podziale 16 województw Polski na 3 klasy ze względu na poziom zanieczyszczenia powietrza.

Literatura

Aldenderfer, M. S. i Blashfield, R. K. (1984). *Cluster Analysis*. Sage.

Anderberg, M. R. (1973). *Cluster Analysis for Applications*. Academic Press.

Arnold, S. J. (1979). A Test for Clusters. *Journal of Marketing Research*, *16*(4), 545-551. **https://doi.org/10.1177/002224377901600411**

Bock, H. H. (1996). *Probability Models and Hypotheses Testing in Partitioning Cluster Analysis*. W: P. Arabie, L. J. Hubert, G. de Soete (red.). Clustering and Classification (s. 377-453). World Scientific.

Breckenridge, J. N. (2000). Validating Cluster Analysis: Consistent Replication and Symmetry. *Multivariate Behavioral Research*, *35*(2), 261-285. **https://doi.org/10.1207/S15327906MBR3502_5**

Caliński, T. i Harabasz, J. (1974). A Dendrite Method for Cluster Analysis. *Communications in Statistics*, *3*(1), 1-27. **https://doi.org/10.1080/03610927408827101**

Carmone, F. J., Kara, A. i Maxwell, S. (1999). HINoV: A New Method to Improve Market Segment Definition by Identifying Noisy Variables. *Journal of Marketing Research*, *36*(4), 501-509. **https://doi.org/10.1177/002224379903600408**

Davies, D. L. i Bouldin, D. W. (1979). A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *1*(2), 224-227. **https://doi.org/10.1109/TPAMI.1979.4766909**

Dąbrowski, M. i Laus-Mączyńska, K. (1978). *Metody wyszukiwania i klasyfikacji informacji*. Wydawnictwa Naukowo-Techniczne.

DeSarbo, W. S., Carroll, J. D., Clark, L. A. i Green, P. E. (1984). Synthesized Clustering: A Method for Amalgamating Clustering Bases with Differential Weighting of Variables. *Psychometrika*, *49*(1), 57-78. **https://psycnet.apa.org/doi/10.1007/BF02294206**

De Soete, G. (1986). Optimal Variable Weighting for Ultrametric and Additive Tree Clustering. *Quality & Quantity: International Journal of Methodology*, *20*(2), 169-180. **https://doi.org/10.1007/BF00227423**

De Soete, G. (1988). OVWTRE: A Program for Optimal Variable Weighting for Ultrametric and Additive Tree Fitting. *Journal of Classification*, *5*, 101-104.

Donoghue, J. R. (1995). Univariate Screening Measures for Cluster Analysis. *Multivariate Behavioral Research*, *30*(3), 385-427. **https://psycnet.apa.org/doi/10.1207/s15327906mbr3003_5**

Dudoit, S. i Fridlyand, J. (2002). A Prediction-based Resampling Method for Estimating the Number of Clusters in a Dataset. *Genome Biology*, *3*(7), artykuł research0036.1. **https://doi.org/10.1186/gb-2002-3-7-research0036**

Everitt, B. S. (1974). *Cluster Analysis*. Heinemann.

Everitt, B. S., Landau, S. i Leese, M. (2001). *Cluster Analysis*. Edward Arnold.

Fischer, I. i Poland, J. (2004). *New Methods for Spectral Clustering*. Technical Report no. IDSIA-12-04. **https://repository.supsi.ch/5528/1/IDSIA-12-04.pdf**

Florek, K., Łukasiewicz, J., Perkal, J., Steinhaus, H. i Zubrzycki, S. (1951). Taksonomia wrocławska. *Przegląd Antropologiczny*, *17*, 193-210.

Fowlkes, E. B., Gnanadesikan, R. i Kettenring, J. R. (1987). Variable Selection in Clustering and Other Contexts. W: C. L. Mallows (red.), *Design, Data, and Analysis* (s. 13-34). Wiley & Sons.

Fowlkes, E. B., Gnanadesikan, R. i Kettenring, J. R. (1988). Variable Selection in Clustering. *Journal of Classification*, *5*(2), 205-228. **https://doi.org/10.1007/BF01897164**

Friedman, J. H. i Meulman, J. J. (2004). Clustering Objects on Subsets Attributes. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *66*(4), 815-849. **https://doi.org/10.1111/j.1467-9868.2004.02059.x**

Gatnar, E. i Walesiak, M. (red.). (2004). *Metody statystycznej analizy wielowymiarowej w badaniach marketingowych*. Wydawnictwo Akademii Ekonomicznej we Wrocławiu.

Girolami, M. (2002). Mercer Kernel-based Clustering in Feature Space. *IEEE Transactions on Neural Networks*, *13*(3), 780-784. **https://doi.org/10.1109/TNN.2002.1000150**

Gnanadesikan, R., Kettenring, J. R. i Tsao, S. L. (1995). Weighting and Selection of Variables for Cluster Analysis. *Journal of Classification*, *12*(1), 113-136. **https://doi.org/10.1007/BF01202271**

Gordon, A. D. (1987). A Review of Hierarchical Classification. *Journal of the Royal Statistical Society Series A: General*, *150*(2), 119-137. **https://doi.org/10.2307/2981629**

Gordon, A. D. (1996). Hierarchical Classification. W: P. Arabie, L. J. Hubert, G. de Soete (red.), *Clustering and Classification* (s. 65-121). World Scientific.

Gordon, A. D. (1998). Cluster Validation. W: C. Hayashi, N. Ohsumi, K. Yajima, Y. Tanaka, H. H. Bock, Y. Baba (red.), *Data Science, Classification, and Related Methods* (s. 22-39). Springer.

Gordon, A. D. (1999). *Classification*. Chapman and Hall/CRC.

Grabiński, T. (1992). *Metody taksonometrii*. Wydawnictwo Akademii Ekonomicznej w Krakowie.

Guyon, I. i Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, *3*, 1157-1182. **https://dl.acm.org/doi/10.5555/944919.944968**

Hair, J. F., Anderson, R. E., Tatham, R. L. i Black, W. C. (1995). *Multivariate Data Analysis with Readings* (wyd. 4). Prentice-Hall.

Hair, J. F., Anderson, R. E., Tatham, R. L. i Black, W. C. (1998). *Multivariate Data Analysis* (wyd. 5). Prentice-Hall.

Hartigan, J. A. (1975). *Clustering Algorithms*. Wiley & Sons.

Hubert, L. J. (1974). Approximate Evaluation Technique for the Single-link and Complete-link Hierarchical Clustering Procedures. *Journal of the American Statistical Association*, *69*(347), 698-704.

Hubert, L. J. i Arabie, P. (1985). Comparing Partitions. *Journal of Classification*, *2*(2-3), 193-218. **https://psycnet.apa.org/doi/10.1007/BF01908075**

Hubert, L. J. i Levin, J. R. (1976). A General Statistical Framework for Assessing Categorical Clustering in Free Recall. *Psychological Bulletin*, *83*(6), 1072-1080. **https://doi.org/10.1037/0033-2909.83.6.1072**

Jajuga, K. (1990). *Statystyczna teoria rozpoznawania obrazów*. PWN.

Jajuga, K. i Walesiak, M. (2000). Standardisation of Data Set Under Different Measurement Scales. W: R. Decker, W. Gaul (red.), *Classification and Information Processing at the Turn of the Millennium* (s. 105-112). Springer.

Karatzoglou, A. (2006). *Kernel Methods. Software, Algorithms and Applications* (rozprawa doktorska). Technische Universität Wien. **https://www.academia.edu/8080848/Kernel_Methods_Algorithms_and_Applications**

Kaufman, L. i Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley & Sons.

Korzeniewski, J. (2012). *Metody selekcji zmiennych w analizie skupień. Nowe procedury*. Wydawnictwo Naukowe Uniwersytetu Łódzkiego.

Lance, G. N. i Williams, W. T. (1966). A Generalized Sorting Strategy for Computer Classifications. *Nature*, *212*, 218. **https://www.nature.com/articles/212218a0**

Lance, G. N. i Williams, W. T. (1967). A General Theory of Classificatory Sorting Strategies. I. Hierarchical Systems. *Computer Journal*, *9*(4), 373-380. **https://doi.org/10.1093/comjnl/9.4.373**

Lance, G. N. i Williams, W. T. (1968). A General Theory of Classificatory Sorting Strategies. II. Clustering Systems. *Computer Journal*, *10*(3), 271-277. **https://doi.org/10.1093/comjnl/10.3.271**

Macnaughton-Smith, P., Williams, W. T., Dale, M. B. i Mockett, L. G. (1964). Dissimilarity Analysis: A New Technique of Hierarchical Sub-division. *Nature*, *202*, 1034-1035.

Makarenkov, V. i Legendre, P. (2001). Optimal Variable Weighting for Ultrametric and Additive Trees and *K*-means Partitioning Methods and Software. *Journal of Classification*, *18*(2), 245-271. **https://doi.org/10.1007/s00357-001-0018-x**

McIntyre, R. M. i Blashfield, R. K. (1980). A Nearest-centroid Technique for Evaluating the Minimum-variance Clustering Procedure. *Multivariate Behavioral Research*, *15*(2), 225-238. **https://psycnet.apa.org/doi/10.1207/s15327906mbr1502_7**

Milligan, G. W. (1989). A Validation Study of a Variable Weighting Algorithm for Cluster Analysis. *Journal of Classification*, *6*(1), 53-71. **https://psycnet.apa.org/doi/10.1007/BF01908588**

Milligan, G. W. (1994). Issues in Applied Classification: Selection of Variables to Cluster. *Classification Society of North America Newsletter*, November, (37).

Milligan, G. W. (1996). Clustering Validation: Results and Implications for Applied Analyses. W: P. Arabie, L. J. Hubert, G. de Soete (red.), *Clustering and Classification* (s. 341-375). World Scientific.

Milligan, G. W. i Cooper, M. C. (1985). An Examination of Procedures for Determining the Number of Clusters in a Data Set. *Psychometrika*, *50*(2), 159-179. **https://psycnet.apa.org/doi/10.1007/BF02294245**

Morey, L. C., Blashfield, R. K. i Skinner, H. A. (1983). A Comparison of Cluster Analysis Techniques Within a Sequential Validation Framework. *Multivariate Behavioral Research*, *18*(3), 309-329. **https://doi.org/10.1207/s15327906mbr1803_4**

Ng, A., Jordan, M. i Weiss, Y. (2002). On Spectral Clustering: Analysis and an Algorithm. W: T. Dietterich, S. Becker, Z. Ghahramani (red.), *Advances in Neural Information Processing Systems* 14 (s. 849-856). MIT Press.

Perkal, J. (1953). Taksonomia wrocławska. *Przegląd Antropologiczny*, *19*, 82-96.

Poland, J. i Zeugmann, T. (2006). Clustering the Google Distance with Eigenvectors and Semidefinite Programming. W: K. P. Jantke, G. Kreuzberger (red.), *Knowledge Media Technologies, First International Core-to-Core Workshop, Dagstuhl, July 23-27, 2006, Germany*. Diskussionsbeiträge, Institut für Medien und Kommunikationswisschaft, Technische Universität Ilmenau, Nr. 21, s. 61-69.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Robles, F. i Sarathy, R. (1986). Segmenting the Commuter Aircraft Market with Cluster Analysis. *Industrial Marketing Management*, *15*, 1-12. **https://www.academia.edu/93814182/Segmenting_the_commuter_aircraft_market_with_cluster_analysis**

Rousseeuw, P. J. (1987). Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of Computational and Applied Mathematics*, *20*, 53-65. **https://doi.org/10.1016/0377-0427(87)90125-7**

Sagan, A. (1998). *Badania marketingowe. Podstawowe kierunki*. Wydawnictwo Akademii Ekonomicznej w Krakowie.

Sneath, P. H. A. (1957). The application of computers to taxonomy. *Journal of General Microbiology*, *17*(1), 201-226. **https://doi.org/10.1099/00221287-17-1-201**

Sokołowski, A. (1992). Empiryczne testy istotności w taksonomii. *Zeszyty Naukowe Akademii Ekonomicznej w Krakowie*. *Monografie*, (108).

Sugar, C. A. i James, G. H. (2003). Finding the Number of Clusters in a Dataset: An Information-theoretic Approach. *Journal of the American Statistical Association*, *98*(463), 750-763. **https://doi.org/10.1198/016214503000000666**

Tibshirani, R. i Walther, G. (2005). Cluster Validation by Prediction Strength. *Journal of Computational and Graphical Statistics*, *14*(3), 511-528. **https://doi.org/10.1198/106186005X59243**

Tibshirani, R., Walther, G. i Hastie, T. (2001). Estimating the Number of Clusters in a Data Set Via the Gap Statistic. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *63*(2), 411-423. **https://doi.org/10.1111/1467-9868.00293**

Von Luxburg, U. (2006). *A Tutorial on Spectral Clustering* (Technical Report TR-149). Max Planck Institute for Biological Cybernetics.

Walesiak, M. (1993). Statystyczna analiza wielowymiarowa w badaniach marketingowych. *Prace Naukowe Akademii Ekonomicznej we Wrocławiu*, (654).

Walesiak, M. (2004). Problemy decyzyjne w procesie klasyfikacji zbioru obiektów. *Prace Naukowe Akademii Ekonomicznej we Wrocławiu*, (1010), 52-71.

Walesiak, M. (2005a). Rekomendacje w zakresie strategii postępowania w procesie klasyfikacji zbioru obiektów. W: A. Zeliaś (red.), *Przestrzenno-czasowe modelowanie i prognozowanie zjawisk gospodarczych* (s. 185-203). Wydawnictwo Akademii Ekonomicznej w Krakowie.

Walesiak, M. (2005b). Problemy selekcji i ważenia zmiennych w zagadnieniu klasyfikacji. *Prace Naukowe Akademii Ekonomicznej we Wrocławiu*, (1076), 106-118.

Walesiak, M. (2008). Ocena stabilności wyników klasyfikacji z wykorzystaniem analizy replikacji. W: J. Pociecha (red.), *Modelowanie i prognozowanie zjawisk społeczno-gospodarczych* (s. 67-72). Wydawnictwo Akademii Ekonomicznej w Krakowie.

Walesiak, M. (2009). Analiza skupień. W: M. Walesiak, E. Gatnar (red.), *Statystyczna analiza danych z wykorzystaniem programu* R (s. 407-433). Wydawnictwo Naukowe PWN.

Walesiak, M. (2012). Klastrowanie widmowe i skale pomiarowe zmiennych. *Przegląd Statystyczny*, *59*(1), 13-31. **https://doi.org/10.59139/ps.2012.01.2**

Walesiak, M. (2013). Zagadnienie doboru liczby klas w klasyfikacji spektralnej. *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, (278), 33-43.

Walesiak, M. (2016). *Uogólniona miara odległości GDM w statystycznej analizie wielowymiarowej z wykorzystaniem programu R* (wyd. 2 popr. i rozsz.). Wydawnictwo Uniwersytetu Ekonomicznego we Wrocławiu.

Walesiak, M. i Dudek, A. (2009). Odległość GDM dla danych porządkowych a klasyfikacja spektralna. *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, (84), 9-19.

Walesiak, M. i Dudek, A. (2010). Klasyfikacja spektralna z wykorzystaniem odległości GDM. *Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu*, (107), 161-171.

Walesiak, M. i Dudek, A. (2023). *clusterSim: Searching for Optimal Clustering Procedure for a Data Set*. R package version 0.51-3. **https://cran.r-project.org/web/packages/clusterSim/**

Zelnik-Manor, L. i Perona, P. (2004). Self-tuning Spectral Clustering. W: *Proceedings of the 18th Annual Conference on Neural Information Processing Systems (NIPS'04)*. **https://proceedings.neurips.cc/paper_files/paper/2004/file/40173ea48d9567f1f393b20c855bb40b-Paper.pdf**

DOI: 10.15611/2024.45.1.11

11 Analiza danych panelowych

11.1. Wprowadzenie do modelowania danych panelowych

Dane panelowe powstałe w wyniku połączenia szeregów przekrojowych i czasowych stwarzają dodatkowe możliwości w prowadzeniu analizy zjawiska w porównaniu z tym, czego możemy dokonać tylko na podstawie szeregów jednowymiarowych.

Mianem danych panelowych[1], zawierających w sobie wymiar czasowy i przekrojowy, możemy określać jedynie dane, w których we wszystkich kolejnych okresach badane są te same jednostki oznaczone w taki sposób, by można było prześledzić dla każdej z nich zmiany zachodzące w czasie. Wielkość panelu określają dwa wymiary: liczba jednostek w zbiorze (N) oraz liczba okresów badania danej jednostki (T). Jeśli dane panelowe dotyczą pojedynczych jednostek (osób, gospodarstw domowych, instytucji, przedsiębiorstw) i zostały zebrane jako wynik prowadzenia specjalnego badania w dłuższym czasie, a nie jedynie na podstawie istniejących już szeregów przekrojowych i czasowych, mamy do czynienia z mikropanelem (np. BBGD, EU-SILC, Nielsen Computer and Mobile Panel, NPD's Checkout). W przeciwnym wypadku, jeśli dane panelowe powstaną na podstawie dostępnych danych statystycznych, a jednostką obserwacji jest pewna zbiorowość (państwo, województwo, sekcja PKD, jednostka samorządowa), mówimy o makropanelu (np. PKB krajów UE w latach 2011-2018, Stopa bezrobocia na poziomie województw w latach 2010-2018).

Warto dodać, że jako swoisty panel można też potraktować dwuwymiarowe dane, w których każdą obserwację identyfikują dwa wymiary inne niż rzeczowy i czasowy (Gruszczyński, 2012, s. 267).

Wśród paneli wyróżnić można panele zbilansowane, czyli takie, w których dla wszystkich N jednostek[2] posiadamy T obserwacji, oraz panele niezbilansowane, w których nie dysponujemy pełnymi danymi (Osińska, 2007).

[1] W literaturze przedmiotu niektórzy autorzy wyraźnie rozróżniają dwa rodzaje tych dwuwymiarowych danych, definiując dane panelowe (czasowo-przekrojowe) jako zawierające więcej jednostek w zbiorowości niż badanych okresów, w odróżnieniu od danych przekrojowo-czasowych, w których dominującym wymiarem jest czas (Dańska-Borsiak, 2011; Kopczewska i in., 2016).

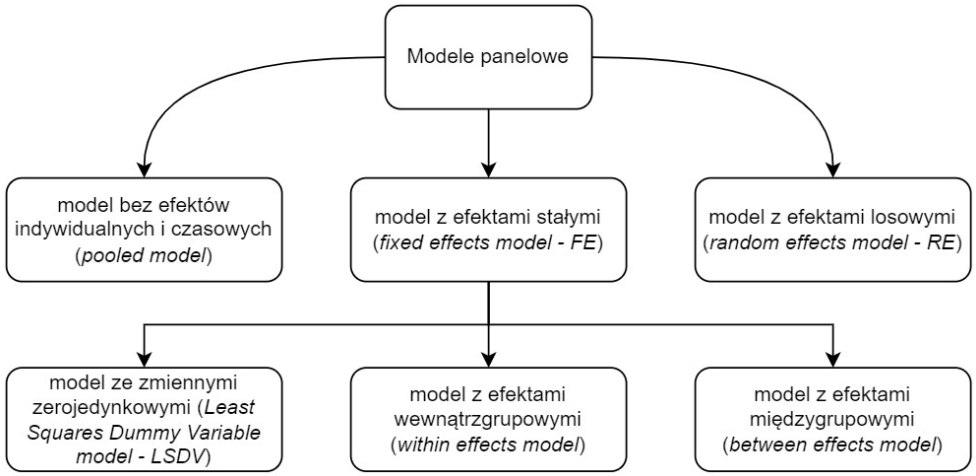
[2] W literaturze przedmiotu termin „jednostka" jest stosowany zamiennie z terminem „obiekt".

W analizie danych panelowych wykorzystuje się przede wszystkim modele panelowe. Za ich stosowaniem przemawia wiele zalet. Do najważniejszych należą:

- zwiększenie liczby obserwacji przez połączenie dwóch wymiarów danych, a tym samym zwiększenie zakresu informacji o badanych jednostkach,
- wzrost liczby stopni swobody i redukcja problemu współliniowości danych,
- umożliwienie eliminacji lub redukcji obciążenia estymatorów (efekt Mundlaka (1961)),
- łatwiejsza kontrola wpływu nieuwzględnionych w modelu zmiennych.

Wśród najistotniejszych ograniczeń w wykorzystaniu modeli panelowych wskazać można: występowanie błędów nielosowych (pokrycia, kompletności danych, braku odpowiedzi, pomiaru), zbyt krótkie szeregi czasowe, zależności przekrojowe, koszt prowadzenia badania, problem „starzenia się" panelu, a co za tym idzie – utraty jego reprezentatywności (Gruszczyński, 2012; Muszyńska, 2006).

Podstawowe typy modeli wykorzystywane do modelowania danych panelowych przestawiono na rys. 11.1.



Rys. 11.1. Podział modeli panelowych

Źródło: opracowanie własne na podstawie (Kopczewska i in., 2016; Maddala, 2013).

Modele panelowe z efektami stałymi i losowymi pozwalają na redukcję błędu pomiaru dzięki uwzględnieniu zróżnicowania wynikającego ze specyfiki badanej jednostki. Różnica pomiędzy modelami z efektami stałymi i modelami z efektami losowymi dotyczy założenia co do pochodzenia zmienności międzygrupowej[3]. W modelach z efektami stałymi informacje o różnicach pomiędzy grupami wyraża zwykle wyraz wolny (Dańska-Borsiak, 2011). Z kolei modele z efektami losowymi zakładają losowe, wyrażone poprzez składnik losowy, pochodzenie zróżnicowania pomiędzy jednostkami (Kopczewska i in., 2016). Dobór modelu zależy od rodzaju prowadzonego badania. Modele z efektami losowymi uznawane są jednak za bardziej użyteczne niż modele z efektami stałymi. Do estymacji parametrów tych pierwszych oprócz różnic między grupami wykorzystuje się bowiem różnice między okresami – źródło zmienności, które jest całkowicie eliminowane

[3] Zmienność międzygrupowa wynika z efektu grupowego (indywidualnego) oznaczającego stałą w czasie charakterystykę jednostki, która ma wpływ na zmienną objaśnianą, a jednocześnie nie jest uwzględniona *explicite* jako zmienna objaśniająca w modelu.

w modelach z dekompozycją wyrazu wolnego (Dańska, 2000). Ponadto modele z efektami losowymi mają zastosowanie we wnioskowaniu o populacji na podstawie próby, w odróżnieniu od modeli z efektami stałymi, których stosowanie zaleca się, jeżeli badaniem objęte są wszystkie lub prawie wszystkie jednostki populacji (Baltagi, 2008; Maddala, 2013). Z tym drugim przypadkiem w praktyce badań spotykamy się zdecydowanie rzadziej. Stosowanie modeli z efektami stałymi zalecane jest w przypadku tzw. paneli długich, o małej liczbie jednostek i dużej liczbie okresów badania, gdy różnice pomiędzy jednostkami daje się uchwycić w wyrazie wolnym (Muszyńska, 2006). Porównując oba typy modeli, należy wspomnieć jeszcze o dwóch własnościach. W przypadku danych panelowych z dużą liczbą jednostek (N) w modelach z efektami stałymi zachodzi konieczność dodania wielu zmiennych binarnych, co znacznie obniża liczbę stopni swobody, zwiększa szansę istnienia współliniowości i wzrostu błędów standardowych. Ponadto w modelach z efektami stałymi, w przeciwieństwie do modeli efektów losowych, nie ma możliwości uwzględnienia wśród zmiennych objaśniających zmiennej stałej w czasie (jak np. rok zatrudnienia czy płeć).

W programie R modele panelowe zostały zaimplementowane w pakiecie `plm` (Croissant i Millo, 2018).

11.2. Model bez efektów indywidualnych i czasowych

W modelu *pooled* przyjmuje się założenie o homogeniczności populacji. Parametry dla zmiennych objaśniających i stałej szacowane są bez uwzględnienia efektów indywidualnych/grupowych i czasowych. Przyjmują wartości jednakowe dla wszystkich jednostek oraz we wszystkich okresach badania. Jest to podejście reprezentujące klasyczny model regresji liniowej, nieuwzględniające własności, jakie mają dane panelowe. Postać modelu jest następująca:

$$y_{it} = \beta_0 + \beta_1 x_{1it} + \cdots + \beta_k x_{kit} + \varepsilon_{it}, \tag{11.1}$$

gdzie: y_{it} – realizacja zmiennej objaśnianej dla i-tej jednostki, w t-tym okresie ($i = 1,\ldots,N$; $t = 1,\ldots,T$), β_0 – wyraz wolny w modelu, x_{kit} – k-ta zmienna objaśniająca ($k = 1,..,p$), gdzie p oznacza liczbę zmiennych objaśniających, β_k – k-ty współczynnik kierunkowy dla k-tej zmiennej objaśniającej, ε_{it} – składnik losowy.

Zwykle jednak populacja nie jest jednorodna i stosowanie tego typu estymatora prowadzi do nieefektywnych czy nawet obciążonych oszacowań. Z tego powodu model ten jest głównie traktowany jako punkt odniesienia dla bardziej złożonych modeli panelowych, w badaniu istotności efektów indywidualnych lub czasowych.

Użycie modelu *pooled* w środowisku R jest możliwe z wykorzystaniem wbudowanej funkcji `lm()` lub `plm()` z pakietu `plm` z dodaniem argumentu `model="pooling"`. Skorzystanie z funkcji `plm()` umożliwia późniejsze porównywanie modeli panelowych między sobą. Funkcja `plm()` wymaga odpowiedniej struktury danych. Oznacza to, że domyślnie pierwsza kolumna powinna zawierać identyfikatory jednostek, a druga identyfikatory czasu. Jeśli jest inaczej, to należy za pomocą argumentu `index` podać odpowiednie nazwy kolumn.

11.3. Model z efektami stałymi

Modele z efektami stałymi są oparte na założeniu, że efekty grupowe (indywidualne) określone są przez niezmienną wielkość, nie są przypadkowe i możliwe jest ich oszacowanie. Różnice pomiędzy jednostkami badania mogą być wyrażone poprzez różne wartości stałej w modelu, uwzględniającej wpływ wszystkich niezmiennych w czasie, nieobserwowalnych czynników, specyficznych dla danej jednostki badania. Można zatem dalej przyjąć, że wszelkie zmiany zmiennej objaśnianej muszą wynikać z wpływów innych niż efekty stałe (Stock i Watson, 2019). Standardowy model *FE* można przedstawić jako:

$$y_{it} = \beta_1 x_{1it} + \cdots + \beta_k x_{kit} + \alpha_i + \varepsilon_{it}, \tag{11.2}$$

gdzie: α_i – stały w czasie efekt indywidualny dla i-tej jednostki.

Dalej omówiono trzy podstawowe typy modeli z efektami stałymi.

Model LSDV (*Least Squares Dummy Variables*)

W modelu LSDV w celu uwzględnienia stałych efektów indywidualnych wprowadzanych jest N zmiennych binarnych w taki sposób, że każda ze zmiennych przybiera wartość 1 dla danej jednostki i 0 dla pozostałych jednostek. W modelu uwzględnia się zatem bezpośrednio, obok wspólnych dla wszystkich jednostek parametrów strukturalnych, specyficzny dla każdej jednostki wyraz wolny (α_i). Odpowiada on za oddziaływanie efektów indywidualnych w danej jednostce i ma charakter stałego w czasie, deterministycznego parametru. Można przyjąć, że wyraz wolny w klasycznym rozumieniu, w modelu LSDV, podlega dekompozycji (Gruszczyński, 2012; Korol i Szczuciński, 2012). Model LSDV można przedstawić w następujący sposób[4]:

$$y_{it} = \beta_1 x_{1it} + \cdots + \beta_k x_{kit} + \sum_{j=1}^{N} \alpha_j d_{ij} + \varepsilon_{it}, \tag{11.3}$$

$$\text{przy czym } d_{ij} = \begin{cases} 0 \text{ dla } i \neq \text{j} \\ 1 \text{ dla } i = \text{j'} \end{cases} \tag{11.4}$$

gdzie: d_{ij} – zmienne binarne dla $i, j = 1, \ldots, N$.

Charakteryzuje się on prostą interpretacją parametrów oraz implementacją. Parametry modelu można szacować za pomocą KMNK. Przeszkodę w estymacji modelu może jednak stanowić znaczna liczba jednostek N. Pojawia się bowiem wymóg odwrócenia w trakcie szacunku macierzy stopnia $N + p$, gdzie p oznacza liczbę zmiennych objaśniających. Rozwiązaniem tego problemu może być *transformacja wewnątrzgrupowa* polegająca na wprowadzeniu do modelu odchyleń zmiennych od średnich grupowych w miejsce zmiennych binarnych (Gruszczyński, 2012). W wyniku jej przeprowadzenia otrzymujemy model z efektami wewnątrzgrupowymi.

[4] Jest to model jednokierunkowy (*one-way model*) – zawiera zestaw zmiennych binarnych tylko dla jednostek. Jeśli są różnice między badanymi okresami, konstruuje się model jednokierunkowy z zestawem zmiennych dla czasu lub model dwukierunkowy (*two-way model*) z dwoma zestawami zmiennych, dla jednostek i okresów (Kopczewska i in., 2016).

Model z efektami wewnątrzgrupowymi (*within effects model*)

W modelu z efektami wewnątrzgrupowymi nie uwzględnia się wyrazu wolnego, nie ma konieczności szacowania wartości efektów indywidualnych stałych (odwracania macierzy wysokiego rzędu). Postać modelu jest następująca:

$$y_{it} - \overline{y}_i = \beta_1(\mathrm{x}_{1it} - \overline{\mathrm{x}}_{1i}) + \cdots + \beta_k(\mathrm{x}_{kit} - \overline{\mathrm{x}}_{ki}) + (\varepsilon_{it} - \bar{\varepsilon}_i), \tag{11.5}$$

gdzie: $\overline{y}_i = \frac{1}{T}\sum_{t=1}^{T} y_{it}$, $\overline{x}_{ki} = \frac{1}{T}\sum_{t=1}^{T} x_{kit}$, $\overline{\varepsilon}_i = \frac{1}{T}\sum_{t=1}^{T} \varepsilon_{it}$ – odpowiednie średnie grupowe.

Parametry β mierzą wpływ zmian cech x wewnątrz grupy na zmienną y. Nieuwzględniane są natomiast zmiany pomiędzy grupami.

Istnieje możliwość doszacowania efektów indywidualnych na podstawie wzoru:

$$\hat{\alpha}_i = \overline{y}_i - \hat{\beta}_1^{FE} x_{1i} + \cdots + \hat{\beta}_k^{FE} x_{ki}, \tag{11.6}$$

gdzie: $\hat{\alpha}_i$ – oszacowanie efektu indywidualnego, $\hat{\beta}_k^{FE}$ – ocena parametru β_k.

Należy dodać, że estymacja i wnioskowanie na podstawie otrzymanych ocen efektów indywidualnych mogą zachodzić jedynie wtedy, gdy mamy do czynienia z panelem o dużej liczbie okresów (T). Model ten charakteryzuje większa liczba stopni swobody (w porównaniu z modelem LSDV zawierającym zmienne binarne), co wpływa na zmniejszenie wartości MSE i niewłaściwe oszacowania błędów standardowych parametrów. Brak wyrazu wolnego zniekształca wartość współczynnika determinacji (Kopczewska i in., 2016).

Parametry modelu z efektami wewnątrzgrupowymi szacowane są za pomocą funkcji `plm()` po dodaniu argumentu `model="within"`. Dodatkowo można wyznaczyć wartości oszacowanych efektów grupowych za pomocą funkcji `fixef()`.

Decyzja o wyborze między modelem bez efektów (*pooled*) a modelem z efektami stałymi (LSDV, *within*) może zostać podjęta na podstawie przedstawionego dalej testu Walda opartego na statystyce *F* (Gruszczyński, 2012).

Model z efektami międzygrupowymi (*between effects model*)

Model z efektami międzygrupowymi cechuje to, że uwzględnia wyraz wolny i opiera się wyłącznie na średnich grupowych zmiennych uwzględnianych w modelu. Postać modelu jest następująca:

$$\overline{y}_i = \alpha + \beta_1 \bar{x}_{1i} + \cdots + \beta_k \bar{x}_{ki} + \varepsilon_i. \tag{11.7}$$

Parametry β określają wpływ zmian pomiędzy grupami na zmienną y. W tym przypadku nie uwzględnia się zmian wewnątrz grup.

Zastosowanie tego modelu w R polega na uwzględnieniu w funkcji `plm()` argumentu `model="between"`.

11.4. Model z efektami losowymi

Model z efektami losowymi zakłada, że efekty indywidualne wynikają ze zmian losowych i w modelu są uwzględniane jako część składnika losowego. Postać modelu jest następująca:

$$y_{it} = \mu + \beta_1 x_{1it} + \cdots + \beta_k x_{kit} + \vartheta_{it}, \tag{11.8}$$

gdzie: μ – wyraz wolny, $\vartheta_{it} = \alpha_i + \varepsilon_{it}$ – łączny składnik losowy będący sumą α_i – losowych efektów indywidualnych, oraz ε_{it} – niezależnego składnika losowego.

Efekty indywidualne (α_i) poszerzają część stochastyczną modelu, nie stanowią szacowanych dla poszczególnych jednostek parametrów jak w przypadku modeli *FE*. Podlegają estymacji w rozumieniu parametrów rozkładu w próbie. Przedmiotem oceny jest ich dyspersja, wskazująca, jaka część łącznego błędu losowego wynika z niezmiennych w czasie nieobserwowalnych charakterystyk indywidualnych jednostek (Korol i Szczuciński, 2012). Przyjmuje się założenie, że efekty indywidualne są niezależne od zmiennych objaśniających. Są jednak powiązane ze składnikiem losowym, co jest przyczyną występowania jego autokorelacji, dlatego nie można szacować tego modelu metodą klasyczną. Do estymacji wykorzystuje się uogólnioną metodę najmniejszych kwadratów (UMNK). Informacji o różnicach międzygrupowych nie niosą za sobą zmienne binarne, jak w modelach *FE*, lecz wariancja międzygrupowa błędu.

Modele z efektami losowymi znajdują zastosowanie przede wszystkim przy wnioskowaniu o populacji na podstawie próby.

Estymacja parametrów modelu panelowego z efektami losowymi odbywa się przez zadeklarowanie parametru `model="random"` w funkcji `plm()`. Ponadto funkcja `ranef()` zwróci w wyniku wartości oszacowanych efektów losowych.

Przedstawione w rozdziale modele należą do klasy tzw. modeli jednokierunkowych (*one-way model*), efekty indywidualne różnią się dla poszczególnych jednostek, ale są stałe w czasie. Rozszerzenie tych modeli stanowią modele dwukierunkowe (*two-way model*). Zakładają one zróżnicowanie efektów indywidualnych również w czasie. Szczegółowe informacje na ich temat można znaleźć m.in w publikacjach Gruszczyńskiego (2012), Korola i Szczucińskiego (2012) oraz Osińskiej (2007).

11.5. Podstawowe testy stosowane w modelach panelowych

Test *F* Walda

Test *F* Walda pozwala na sprawdzenie istotności efektów indywidualnych (grupowych). Jakość modelu *FE* (LSDV, *within*) jest tu porównywana z modelem klasycznym. Weryfikowana hipoteza zakłada, że wszystkie wprowadzone do modelu *FE* wyrazy wolne są równe, niezależnie od jednostki i czasu – oznacza to brak efektów grupowych.

Hipoteza alternatywna z kolei zakłada, że wyrazy wolne są różne, a zatem istnieją podstawy do zastosowania modelu *FE*. Funkcja `pFtest()` pozwala na porównanie modelu *pooled* z modelem LSDV/*within*.

Test Hausmana

Test Hausmana porównuje model z efektami stałymi z modelem z efektami losowymi i może być wykorzystany jako wskazówka w wyborze między tymi modelami. Hipoteza zerowa zakłada, że efekty indywidualne są niezależne od zmiennych objaśniających, więc zarówno estymator *FE*, jak i estymator *RE* są nieobciążone. W takim przypadku za bardziej efektywny uznaje się estymator dla modelu z efektami losowymi. Hipoteza alternatywna zakłada, że założenie dotyczące niezależności nie jest spełnione, estymator *FE* jest nieobciążony, *RE* zaś jest obciążony lub nastąpił błąd specyfikacji modelu. Odrzucenie hipotezy zerowej wskazuje więc, że właściwszy wybór ze względu na lepsze własności modelu stanowi model z efektami stałymi (Maddala, 2013). Ten test jest dostępny w ramach funkcji `phtest()`.

W rozdziale 11 opisano jedynie podstawy dotyczące stosowania modeli panelowych, głównie pod kątem ich wykorzystania w analizie danych dostępnych w BDL. Bardziej szczegółowe informacje na temat tej metody (zaawansowane modele panelowe, dynamiczne modele liniowe dla danych panelowych) można znaleźć m.in. w pracach: Baltagiego (2008), Greene'a (2003), Gruszczyńskiego (2012), Hancka i in. (2019), Maddali (2013), Osińskiej (2007), Stocka i Watsona (2019) oraz Verbeeka (2004).

11.6. Zastosowania z wykorzystaniem programu R dla danych z BDL

W analizie danych panelowych będziemy korzystać z następujących trzech pakietów programu R (R Core Team, 2023): `tidyverse` do przekształcania i wizualizacji danych, `bdl` do pobierania danych z Banku Danych Lokalnych oraz `plm` do modelowania danych panelowych.

Stopa bezrobocia na poziomie województw w latach 2010-2018

W celu egzemplifikacji modeli opartych na danych panelowych opracujemy model objaśniający stopę bezrobocia rejestrowanego w przekroju województw dla lat 2010-2018 oznaczaną dalej przez y. Jako zmienną objaśniającą, oznaczaną przez x, przyjmiemy udział ludności w wieku produkcyjnym w ludności ogółem. Skrypt 11.1a pozwala na pobranie i połączenie danych.

Skrypt 11.1a

```
stopa_bezr<-get_data_by_variable(varId=60270, unitLevel=2, year=2010:2018) %>%
  rename(stopa_bezr=val)
lud_prod<-get_data_by_variable(varId=60566, unitLevel=2, year=2010:2018) %>%
  rename(lud_prod=val)
dane<-inner_join(stopa_bezr, lud_prod) %>%
  select(-c(id,measureUnitId,attrId,measureName,attributeDescription),
    woj=name)
print(head(dane, 20))
```

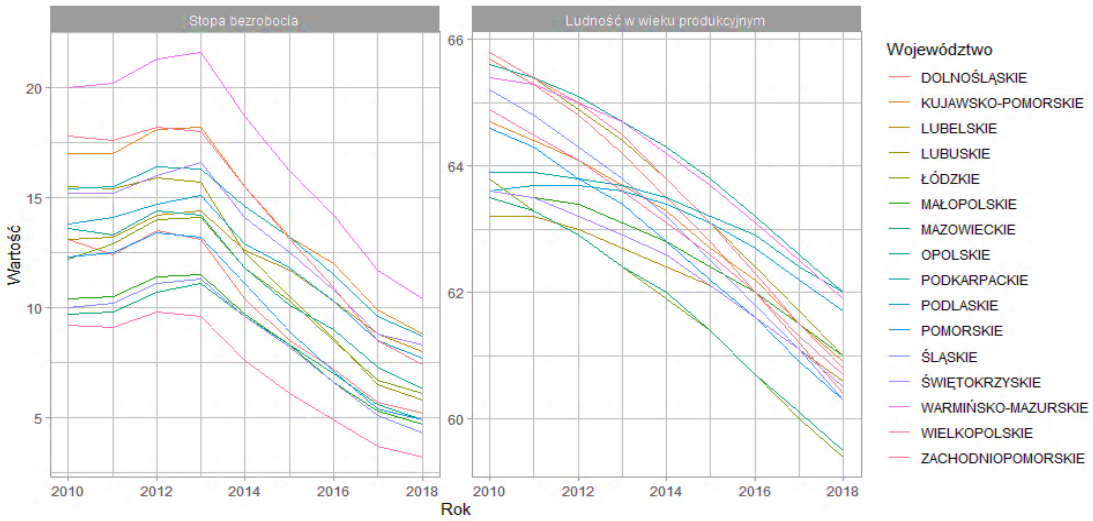
W wyniku realizacji skryptu 11.1a uzyskano zbiór o nazwie `dane` i wypisano w konsoli pierwszych 20 obserwacji.

```
# A tibble: 20 x 4
   woj         year  stopa_bezr lud_prod
   <chr>       <chr>      <dbl>    <dbl>
 1 MAŁOPOLSKIE 2010        10.4     63.6
 2 MAŁOPOLSKIE 2011        10.5     63.5
 3 MAŁOPOLSKIE 2012        11.4     63.4
 4 MAŁOPOLSKIE 2013        11.5     63.1
 5 MAŁOPOLSKIE 2014         9.7     62.8
 6 MAŁOPOLSKIE 2015         8.3     62.4
 7 MAŁOPOLSKIE 2016         6.6     62
 8 MAŁOPOLSKIE 2017         5.3     61.5
 9 MAŁOPOLSKIE 2018         4.7     61
10 ŚLĄSKIE     2010        10       65.2
11 ŚLĄSKIE     2011        10.2     64.8
12 ŚLĄSKIE     2012        11.1     64.3
13 ŚLĄSKIE     2013        11.3     63.8
14 ŚLĄSKIE     2014         9.6     63.2
15 ŚLĄSKIE     2015         8.2     62.5
16 ŚLĄSKIE     2016         6.6     61.8
17 ŚLĄSKIE     2017         5.1     61.1
18 ŚLĄSKIE     2018         4.3     60.3
19 LUBUSKIE    2010        15.5     65.8
20 LUBUSKIE    2011        15.4     65.4
```

Zebrane dane posiadają informacje o 16 województwach ($N = 16$) dla 9 lat ($T = 9$), co daje w sumie 144 obserwacje. Mamy tu do czynienia z danymi zbilansowanymi. Rysunek 11.2 przedstawia te dane (skrypt 11.1b).

Skrypt 11.1b

```
p<-dane %>%
  pivot_longer(stopa_bezr:lud_prod) %>%
  mutate(name=factor(x=name,
    levels = c("stopa_bezr", "lud_prod"), labels=c("Stopa bezrobocia",
    "Ludność w wieku produkcyjnym"), ordered=TRUE)) %>%
  ggplot(aes(x=as.numeric(year), y=value, color=woj)) +
  geom_line() +
  xlab("Rok") +
  ylab("Wartość") +
  scale_color_discrete(name="Województwo") +
  facet_wrap(~ name, scales="free") +
  theme_light()
print(p)
```



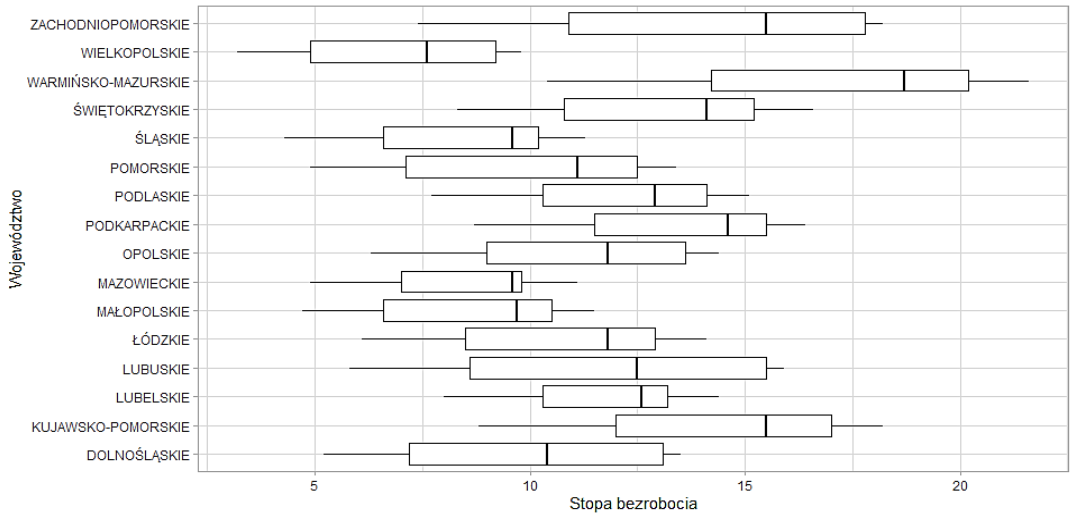
Rys. 11.2. Stopa bezrobocia oraz ludność w wieku produkcyjnym w latach 2010-2018 w przekroju województw

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Podczas analizy danych panelowych należy zwrócić uwagę na zróżnicowanie występujące w ramach jednostek. Jest ono widoczne na rys. 11.2 oraz 11.3 przedstawiających podstawowe statystyki opisowe (skrypt 11.1c).

Skrypt 11.1c

```
p<-ggplot(dane, aes(x=woj, y=stopa_bezr)) +
  geom_boxplot() +
  coord_flip() +
  xlab("Województwo") +
  ylab("Stopa bezrobocia") +
  theme_light()
print(p)
```



Rys. 11.3. Wykres pudełkowy stopy bezrobocia w latach 2010-2018 w przekroju województw

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Interpretując dane na rysunku 11.3, można zauważyć, że stopa bezrobocia np. w województwie dolnośląskim w latach 2010-2018 wahała się od 5,2 do 13,5%. Z kolei mediana stopy bezrobocia w tych latach była najniższa w województwie wielkopolskim, a najwyższa w województwie warmińsko-mazurskim. Ponadto maksymalna stopa bezrobocia w Wielkopolsce była mniejsza od najniższej stopy bezrobocia na Mazurach. Potrzebne zatem jest narzędzie matematyczne, które uwzględni to zróżnicowanie.

W pierwszej kolejności opracujemy model niezawierający informacji o specyfice danych – klasyczny model regresji liniowej. W tym przypadku struktura danych jest właściwa, to znaczy, że pierwsza kolumna zawiera identyfikatory jednostek, a druga identyfikatory czasu. W związku z tym zastosowanie funkcji `plm()` nie wymaga wprowadzenia dodatkowych argumentów (skrypt 11.1d).

Skrypt 11.1d

```
library(plm)
m_pooling<-plm(stopa_bezr~lud_prod, data=dane, model="pooling")
print(summary(m_pooling))
```

Rezultatem uruchomienia skryptu 11.1d jest wyświetlenie statystyk modelu.

```
   Pooling Model
Call:
plm(formula = stopa_bezr ~ lud_prod, data = dane, model = "pooling")
Balanced Panel: n = 16, T = 9, N = 144
Residuals:
   Min. 1st Qu.  Median 3rd Qu.    Max.
-6.1661 -2.1107  0.1296  2.0546  6.6420

Coefficients:
              Estimate Std. Error t-value  Pr(>|t|)
(Intercept) -117.07709    9.46047 -12.375 < 2.2e-16 ***
lud_prod       2.04073    0.15024  13.583 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    2252.9
Residual Sum of Squares: 979.82
R-Squared:      0.56509
Adj. R-Squared: 0.56203
F-statistic: 184.504 on 1 and 142 DF, p-value: < 2.22e-16
```

Odsetek ludności w wieku produkcyjnym opisuje 56,5% zmienności stopy bezrobocia. Wzrost tego odsetka w województwie wpływa na średni wzrost stopy bezrobocia o dwa punkty procentowe. Wyraz wolny nie ma sensownej ekonomicznej interpretacji. W następnym kroku zastosujemy model z efektami wewnątrzgrupowymi (skrypt 11.1e).

Skrypt 11.1e

```
m_within<-plm(stopa_bezr~lud_prod, data=dane, model="within")
print(summary(m_within))
```

Poniżej znajduje się rezultat kodu ze skryptu 11.1e.

```
  Oneway (individual) effect Within Model
Call:
plm(formula = stopa_bezr ~ lud_prod, data = dane, model = "within")
Balanced Panel: n = 16, T = 9, N = 144
Residuals:
    Min.  1st Qu.   Median  3rd Qu.     Max.
-3.22605 -0.69734 -0.01060  0.75191  2.90043
Coefficients:
         Estimate Std. Error t-value  Pr(>|t|)
lud_prod 2.153257   0.080071  26.892 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    1303
Residual Sum of Squares: 194.65
R-Squared:      0.85062
Adj. R-Squared: 0.8318
F-statistic: 723.164 on 1 and 127 DF, p-value: < 2.22e-16
```

W tym przypadku w modelu nie występuje wyraz wolny. Z kolei współczynnik kierunkowy dla zmiennej „odsetek ludności w wieku produkcyjnym" wynosi 2,15. Taki model wyjaśnia 85,1% zmienności stopy bezrobocia, co jest znaczną poprawą w stosunku do klasycznego modelu regresji. Z wykorzystaniem funkcji `fixef()` można sprawdzić, jakie są wartości efektów stałych dla każdego województwa.

Skrypt 11.1f

```
print(summary(fixef(m_within)))
```

Uruchomienie kodu w skrypcie 11.1f powoduje wyświetlenie następujących wartości:

```
                     Estimate Std. Error t-value  Pr(>|t|)
DOLNOŚLĄSKIE         -126.4490     5.0871 -24.857 < 2.2e-16 ***
KUJAWSKO-POMORSKIE   -121.3637     5.0658 -23.957 < 2.2e-16 ***
LUBELSKIE            -122.1454     4.9984 -24.437 < 2.2e-16 ***
LUBUSKIE             -125.1489     5.1101 -24.490 < 2.2e-16 ***
ŁÓDZKIE              -122.2423     4.9620 -24.636 < 2.2e-16 ***
MAŁOPOLSKIE          -126.0589     5.0285 -25.069 < 2.2e-16 ***
MAZOWIECKIE          -124.4534     4.9620 -25.081 < 2.2e-16 ***
OPOLSKIE             -126.8648     5.1474 -24.646 < 2.2e-16 ***
PODKARPACKIE         -122.7388     5.0817 -24.153 < 2.2e-16 ***
PODLASKIE            -123.7227     5.0676 -24.415 < 2.2e-16 ***
POMORSKIE            -125.0469     5.0339 -24.841 < 2.2e-16 ***
ŚLĄSKIE              -127.1663     5.0614 -25.125 < 2.2e-16 ***
ŚWIĘTOKRZYSKIE       -121.1881     5.0090 -24.194 < 2.2e-16 ***
WARMIŃSKO-MAZURSKIE  -120.6162     5.1394 -23.469 < 2.2e-16 ***
WIELKOPOLSKIE        -128.5851     5.0596 -25.414 < 2.2e-16 ***
ZACHODNIOPOMORSKIE   -122.7882     5.1075 -24.041 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

W praktyce są to wartości wyrazu wolnego w modelu dla każdej analizowanej jednostki. Przykładowo model dla województwa dolnośląskiego będzie miał następującą postać: $y = 2{,}15x - 126{,}45$. Niestety w analizowanym przypadku nie można nadać temu efektowi sensownej interpretacji ekonomicznej. Drugim wariantem modelu z efektami stałymi jest model uwzględniający efekt międzygrupowy (skrypt 11.1g).

Skrypt 11.1g

```
m_between<-plm(stopa_bezr~lud_prod, data=dane, model="between")
print(summary(m_between))
```

W wyniku realizacji skryptu 11.1g otrzymamy następujący rezultat:

```
   Oneway (individual) effect Between Model
Call:
plm(formula = stopa_bezr ~ lud_prod, data = dane, model = "between")
Balanced Panel: n = 16, T = 9, N = 144
Observations used in estimation: 16
Residuals:
     Min.   1st Qu.    Median   3rd Qu.      Max.
-4.411066 -2.088442 -0.072443  1.648586  4.074067
Coefficients:
            Estimate Std. Error t-value Pr(>|t|)
(Intercept) -91.6669    57.2397 -1.6015  0.13159
lud_prod      1.6371     0.9092  1.8006  0.09334 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    105.54
Residual Sum of Squares: 85.698
R-Squared:      0.18804
Adj. R-Squared: 0.13004
F-statistic: 3.24213 on 1 and 14 DF, p-value: 0.093343
```

Ten model, obliczony na podstawie zagregowanych danych, ma wartość współczynnika R^2 na poziomie zaledwie 18,8%. Współczynnik kierunkowy wynosi 1,63 i znajduje się na granicy istotności. Model z efektem międzygrupowym nie będzie zatem przedmiotem naszego dalszego zainteresowania.

Porównajmy pod względem formalnym dwa wcześniejsze modele (skrypt 11.1h).

Skrypt 11.1h

```
print(pFtest(m_within, m_pooling))
```

Wynik testu *F* Walda wygenerowanego na podstawie skryptu 11.1h znajduje się poniżej:

```
    F test for individual effects
data:  stopa_bezr ~ lud_prod
F = 34.152, df1 = 15, df2 = 127, p-value < 2.2e-16
alternative hypothesis: significant effects
```

Bardzo mała wartość p wskazuje na odrzucenie hipotezy zerowej, zatem możemy stwierdzić, że spośród tych dwóch modeli lepszy jest model z efektem wewnątrzgrupowym.

W ostatnim kroku zastosowano model z efektem losowym (skrypt 11.1i).

Skrypt 11.1i

```
m_random<-plm(stopa_bezr~lud_prod, data=dane, model="random")
print(summary(m_random))
```

Rezultatem wywołania funkcji ze skryptu 11.1i jest podsumowanie parametrów modelu:

```
   Oneway (individual) effect Random Effect Model
      (Swamy-Arora's transformation)
Call:
plm(formula = stopa_bezr ~ lud_prod, data = dane, model = "random")
Balanced Panel: n = 16, T = 9, N = 144
Effects:
                 var std.dev share
idiosyncratic 1.533   1.238 0.205
individual    5.951   2.439 0.795
theta: 0.8332
Residuals:
     Min.   1st Qu.    Median   3rd Qu.      Max.
-3.718527 -0.786526 -0.016375  0.777636  3.495260
Coefficients:
               Estimate  Std. Error z-value  Pr(>|z|)
(Intercept) -123.911080    5.047104 -24.551 < 2.2e-16 ***
lud_prod       2.149285    0.079571  27.011 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    1329.5
Residual Sum of Squares: 216.6
R-Squared:      0.83708
Adj. R-Squared: 0.83593
Chisq: 729.581 on 1 DF, p-value: < 2.22e-16
```

W odróżnieniu od modelu z efektem wewnątrzgrupowym występuje tutaj wyraz wolny. Wartość współczynnika kierunkowego w obu modelach kształtuje się na bardzo podobnym poziomie i wynosi 2,15. Współczynnik R^2 również, podobnie jak w modelu z efektem wewnątrzgrupowym, jest wysoki i wynosi 83,7%. Wartość parametru $\theta = 0{,}8332$ wskazuje, że zmienność efektów indywidualnych (województw) odpowiada za 83,3% zmienności losowej (nieopisanej przez model). W raporcie, w części poświęconej efektom (`Effects`) pojawia się tabela z wartościami `idiosyncratic` i `individual`. Kolumna `share` zawiera odpowiednio wartość międzygrupowego i wewnątrzgrupowego współczynnika determinacji. Współczynnik międzygrupowy wskazuje, w ilu procentach model tłumaczy zmiany stopy bezrobocia pomiędzy jednostkami, nie biorąc pod uwagę zmian w czasie (20,5%). Natomiast współczynnik wewnątrzgrupowy pokazuje, w jakim stopniu model wyjaśnia zmiany stopy bezrobocia w czasie, bez uwzględnienia zróżnicowania pomiędzy jednostkami (79,5%).

Z wykorzystaniem funkcji `ranef()` możemy poznać wartości efektu losowego dla każdego województwa (skrypt 11.1j).

Skrypt 11.1j

```
print(ranef(m_random))
```

Wartości efektów losowych wywołanych funkcją ze skryptu 11.1j są następujące:

```
       DOLNOŚLĄSKIE  KUJAWSKO-POMORSKIE           LUBELSKIE
         -2.2227937           2.7200138           1.9568057
          LUBUSKIE             ŁÓDZKIE         MAŁOPOLSKIE
         -0.9576791           1.8608913          -1.8463140
        MAZOWIECKIE            OPOLSKIE        PODKARPACKIE
         -0.2887057          -2.6241024           1.3839461
          PODLASKIE           POMORSKIE             ŚLĄSKIE
          0.4267657          -0.8622059          -2.9213667
    ŚWIĘTOKRZYSKIE WARMIŃSKO-MAZURSKIE       WIELKOPOLSKIE
          2.8880304           3.4503299          -4.3007969
ZACHODNIOPOMORSKIE
          1.3371815
```

W tym przypadku są to wartości wskazujące, o ile wyższa lub niższa będzie wartość stopy bezrobocia w każdej jednostce od wartości przeciętnej oszacowanej dla wszystkich jednostek. Przykładowo, stopa bezrobocia w województwie wielkopolskim jest średnio o 4,3 punktu procentowego niższa niż stopa bezrobocia w Polsce ogółem. Postać modelu dla tego województwa będzie następująca: $y = 2{,}15x - 123{,}91 - 4{,}30 = 2{,}15x - 119{,}61$.

Z wykorzystaniem testu Hausmana możemy sprawdzić, czy efekty są skorelowane ze zmienną objaśniającą, innymi słowy, który z modeli *FE* czy *RE* charakteryzuje się lepszymi własnościami (skrypt 11.1k).

Skrypt 11.1k

```
print(phtest(m_within, m_random))
```

Wynik testu ze skryptu 11.1k przedstawiony jest poniżej.

```
      Hausman Test
data:  stopa_bezr ~ lud_prod
chisq = 0.19772, df = 1, p-value = 0.6566
alternative hypothesis: one model is inconsistent
```

Nie ma podstaw do odrzucenia hipotezy zerowej – lepszy jest model z efektami losowymi.

Wynagrodzenia na poziomie powiatów w latach 2009-2018

Z wykorzystaniem danych mierzonych na poziomie powiatów zostanie opracowany model opisujący przeciętne miesięczne wynagrodzenie brutto dla lat 2009-2018. Jako zmienne objaśniające zostały wybrane następujące cechy:

- podmioty wpisane do rejestru REGON na 10 tys. ludności,
- zasięg korzystania ze środowiskowej pomocy społecznej ogółem,
- drogi gminne i powiatowe o twardej nawierzchni na 10 tys. ludności.

Pobranie danych i ich połączenie umożliwia skrypt 11.2a.

Skrypt 11.2a

```
wynagr<-get_data_by_variable(varId=64428, unitLevel=5, year=2009:2018) %>%
  select(id, name, year, wynagr=val)
podmioty<-get_data_by_variable(varId=60530, unitLevel=5, year=2009:2018) %>%
  select(id, year, podmioty=val)
pomoc_spol<-get_data_by_variable(varId=458700, unitLevel=5, year=2009:2018) %>%
  select(id, year, pomoc_spol=val)
drogi<-get_data_by_variable(varId=395404, unitLevel=5, year=2009:2018) %>%
  select(id, year, drogi=val)
dane_pow<-inner_join(wynagr, inner_join(podmioty, inner_join(pomoc_spol,
  drogi)))
```

Dane zostały pobrane z wykorzystaniem pakietu `bdl` oraz połączone w jeden zbiór danych. Łączenie odbywało się z wykorzystaniem kolumny `id`, ponieważ nazwy powiatów nie są unikalne w skali całego kraju (istnieje 10 nazw powiatów, które występują dwa razy – w różnych województwach). Z tego względu pierwsza kolumna zbioru danych zawiera unikalny identyfikator powiatu, druga nazwę powiatu, a trzecia rok. Nie jest to domyślny format danych dla funkcji `plm()`, zatem aby zastosować jakikolwiek model panelowy, trzeba dodać argument `index`, wskazując odpowiednie nazwy kolumn. Ta procedura została zaprezentowana w skrypcie 11.2b.

Skrypt 11.2b

```
m_pow_pooling<-plm(wynagr~podmioty+pomoc_spol+drogi, data=dane_pow,
  model="pooling", index=c("id", "year"))
print(summary(m_pow_pooling))
```

Rezultatem działania skryptu 11.2b jest następujący raport:

```
   Pooling Model
Call:
plm(formula = wynagr ~ podmioty + pomoc_spol + drogi, data = dane_pow,
    model = "pooling", index = c("id", "year"))
Unbalanced Panel: n = 380, T = 6-10, N = 3796
Residuals:
     Min.   1st Qu.    Median   3rd Qu.      Max.
-1388.295  -327.501   -37.126   266.071  4465.097
Coefficients:
               Estimate  Std. Error  t-value  Pr(>|t|)
(Intercept) 3521.157203   59.687717  58.9930 < 2.2e-16 ***
podmioty       0.475976    0.040944  11.6249 < 2.2e-16 ***
pomoc_spol   -76.111371    2.612892 -29.1292 < 2.2e-16 ***
drogi          1.249819    0.249247   5.0144 5.564e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    1492900000
```

```
Residual Sum of Squares: 1.017e+09
R-Squared:      0.31877
Adj. R-Squared: 0.31823
F-statistic: 591.454 on 3 and 3792 DF, p-value: < 2.22e-16
```

Na podstawie otrzymanego wyniku możemy zauważyć, że mamy do czynienia z danymi niezbilansowanymi. W zbiorze znajduje się 380 powiatów, dla których dane są obserwowane w liczbie od 6 do 10 okresów. Wynika to z utworzenia w 2013 r. miasta na prawach powiatu – Wałbrzycha. Takie dane nie stanowią przeszkody w estymacji parametrów modeli panelowych. Na podstawie wyznaczonego modelu można stwierdzić, że na przeciętne wynagrodzenie dodatnio wpływają liczba podmiotów gospodarczych oraz wskaźnik utwardzonych dróg. Z kolei odsetek osób korzystających z pomocy społecznej jest ujemnie skorelowany z wynagrodzeniem. Ten model wyjaśnia 31,9% zmienności wynagrodzenia.

W kolejnym kroku włączymy efekt wewnątrzgrupowy w celu uwzględnienia tego, że mamy do czynienia z danymi panelowymi (skrypt 11.2c).

Skrypt 11.2c

```
m_pow_within<-plm(wynagr~podmioty+pomoc_spol+drogi, data=dane_pow,
  model="within", index=c("id", "year"))
print(summary(m_pow_within))
```

W wyniku realizacji skryptu 11.2c otrzymuje się:

```
   Oneway (individual) effect Within Model
Call:
plm(formula = wynagr ~ podmioty + pomoc_spol + drogi, data = dane_pow,
    model = "within", index = c("id", "year"))
Unbalanced Panel: n = 380, T = 6-10, N = 3796
Residuals:
      Min.    1st Qu.     Median    3rd Qu.       Max.
-1235.2879  -101.9069     3.8285   101.6734  1770.9778
Coefficients:
             Estimate  Std. Error  t-value  Pr(>|t|)
podmioty      4.038529    0.094649  42.6683 < 2.2e-16 ***
pomoc_spol -131.871417    2.689888 -49.0249 < 2.2e-16 ***
drogi         3.233229    0.585898   5.5184 3.676e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    666070000
Residual Sum of Squares: 147750000
R-Squared:      0.77817
Adj. R-Squared: 0.75334
F-statistic: 3990.91 on 3 and 3413 DF, p-value: < 2.22e-16
```

Współczynniki kierunkowe przy zmiennych objaśniających nie zmieniły swoich znaków. Znacząco wzrósł współczynnik R^2, który teraz wynosi 77,8%. Z wykorzystaniem funkcji `fixef()` możemy zobaczyć, ile wynoszą efekty stałe dla poszczególnych powiatów (skrypt 11.2d).

Skrypt 11.2d

```
efekty_stale<-summary(fixef(m_pow_within))
print(head(efekty_stale))
```

Na podstawie skryptu 11.2d wyświetlono pierwszych 6 obserwacji ze zbioru zawierającego efekty stałe dla modelu z efektami wewnątrzgrupowymi.

```
              Estimate Std. Error    t-value     Pr(>|t|)
011212001000  743.61109   118.1023  6.2963286 3.436191e-10
011212006000   59.58003   128.9616  0.4619981 6.441121e-01
011212008000 -249.86549   163.4115 -1.5290566 1.263431e-01
011212009000  -27.61270   126.7306 -0.2178850 8.275317e-01
011212014000  569.75963   144.6196  3.9397115 8.321222e-05
011212019000 -421.22278   131.6855 -3.1987031 1.393106e-03
```

Z powodu naturalnych różnic pomiędzy powiatami obserwowane są także znaczące różnice w oszacowaniach efektów stałych.

Ze względu na bardzo duże zróżnicowanie danych model z efektami międzygrupowymi (*between*) ma gorsze dopasowanie aniżeli zwykły model regresji liniowej (*pooled*), więc zostanie pominięty. Przejdziemy zatem do modelu z efektami losowymi (skrypt 11.2e).

Skrypt 11.2e

```
m_pow_random<-plm(wynagr~podmioty+pomoc_spol+drogi, data=dane_pow,
  model="random", index=c("id", "year"))
print(summary(m_pow_random))
```

Poniżej przedstawiony jest wynik realizacji kodu zawartego w skrypcie 11.2e.

```
   Oneway (individual) effect Random Effect Model
    (Swamy-Arora's transformation)
Call:
plm(formula = wynagr ~ podmioty + pomoc_spol + drogi, data = dane_pow,
    model = "random", index = c("id", "year"))
Unbalanced Panel: n = 380, T = 6-10, N = 3796
Effects:
                   var  std.dev share
idiosyncratic  43291.5    208.1 0.217
individual    156251.3    395.3 0.783
theta:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.7899  0.8358  0.8358  0.8357  0.8358  0.8358
Residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1179.17  -139.57     1.37    -0.05   140.71  2312.72
Coefficients:
               Estimate  Std. Error z-value  Pr(>|z|)
(Intercept) 2003.782926  104.029938  19.262 < 2.2e-16 ***
podmioty       2.092187    0.075525  27.702 < 2.2e-16 ***
pomoc_spol  -137.026051    2.694760 -50.849 < 2.2e-16 ***
drogi          8.552091    0.430553  19.863 < 2.2e-16 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Total Sum of Squares:    688890000
Residual Sum of Squares: 216810000
R-Squared:      0.68528
Adj. R-Squared: 0.68503
Chisq: 8256.76 on 3 DF, p-value: < 2.22e-16
```

Ponownie obserwujemy nieznaczne różnice w oszacowaniach parametrów β, natomiast współczynnik R^2 wynosi 68,5% – nieznacznie mniej niż w przypadku modelu *within*. Ze względu na niezbilansowane dane współczynnik θ jest wyznaczony dla każdej jednostki. Możemy zauważyć, że te wartości są wysokie i wynoszą od 0,79 do 0,84, co wskazuje na duży odsetek zmienności losowej wyjaśnionej przez zmienność efektów indywidualnych (powiatów).

Następnie, używając funkcji `ranef()`, sprawdzamy wartości efektów losowych (skrypt 11.2f).

Skrypt 11.2f

```
efekty_losowe<-ranef(m_pow_random)
print(head(efekty_losowe))
```

Sześć pierwszych wartości efektów losowych wygenerowanych przez skrypt 11.2f.

```
011212001000 011212006000 011212008000 011212009000 011212014000 011212019000
   -99.27866   -343.18236  -1571.50263   -505.39839   -798.72732   -607.85583
```

Ze względu na specyfikę modelu wartości te znacznie różnią się od tych, które uzyskaliśmy w przypadku modelu z efektami stałymi. Korzystanie z niezbilansowanych danych panelowych wyklucza zastosowanie testów statystycznych wspomagających wybór właściwego modelu, zatem trzeba skupić się na innym kryterium, np. na współczynniku R^2 albo istotności parametrów β. Przykładowo model z efektami losowymi wyjaśnia 68,5% zmienności wynagrodzenia, podczas gdy model z efektami stałymi 77,8%. Na podstawie tego kryterium preferowany będzie model *FE*.

Literatura

Baltagi, B. (2008). *Econometric Analysis of Panel Data*. John Wiley & Sons.

Croissant, Y. i Millo, G. (2008). Panel Data Econometrics in R: The plm Package. *Journal of Statistical Software*, *27*(2), 1-43. **https://doi.org/10.18637/jss.v027.i02**

Dańska, B. (2000). Przestrzenno-czasowe modelowanie zmian w działalności produkcyjnej w Polsce. Zastosowanie modeli panelowych. W: B. Suchecki (red.), *Dane panelowe i modelowanie wielowymiarowe w badaniach ekonomicznych* (s. 213-236). Absolwent.

Dańska-Borsiak, B. (2011). *Dynamiczne modele panelowe w badaniach ekonomicznych*. Wydawnictwo Uniwersytetu Łódzkiego.

Greene, W. H. (2003). *Econometric Analysis* (wyd. 5). Pearson Education.

Gruszczyński, M. (red.). (2012). *Mikroekonometria. Modele i metody analizy danych indywidualnych*. Wolters Kluwer Polska.

Hanck, C., Arnold, M., Gerber, A. i Schmelzer, M. (2019). *Introduction to Econometrics with R*. University of Duisburg-Essen. **https://www.econometrics-with-r.org/**

Hausman, J. (1978). Specification Tests in Econometrics. *Econometrica*, *46*, 1251-1271.

Kopczewska, K., Kopczewski, T. i Wójcik, P. (2016). *Metody ilościowe w R: aplikacje ekonomiczne i finansowe*. CeDeWu.

Korol, J. i Szczuciński, P. (2012). Ekonometryczne modelowanie zróżnicowania związków w sektorze małych i średnich przedsiębiorstw w przestrzeni regionalnej. *Studia i Prace Wydziału Nauk Ekonomicznych i Zarządzania*, (26), 209-224.

Maddala, G. S. (2013). *Ekonometria*. Wydawnictwo Naukowe PWN.

Mundlak, Y. (1961). Empirical Production Function Free of Management Bias. *Journal of Farm Economics*, *43*(1), 44-56.

Muszyńska, J. (2006). Modelowanie danych panelowych. *Zeszyty Naukowe Państwowej Wyższej Szkoły Zawodowej we Włocławku*, *1*, 213-236.

Osińska, M. (red.) (2007). *Ekonometria współczesna*. Wydawnictwo Dom Organizatora.

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Stock, J. H. i Watson, M. W. (2019). *Introduction to Econometrics* (wyd. 4). Pearson.

Verbeek, M. (2004). *A Guide to Modern Econometrics*. John Wiley & Sons.

DOI: 10.15611/2024.45.1.12

12 Statystyka małych obszarów

12.1. Podstawy statystyki małych obszarów

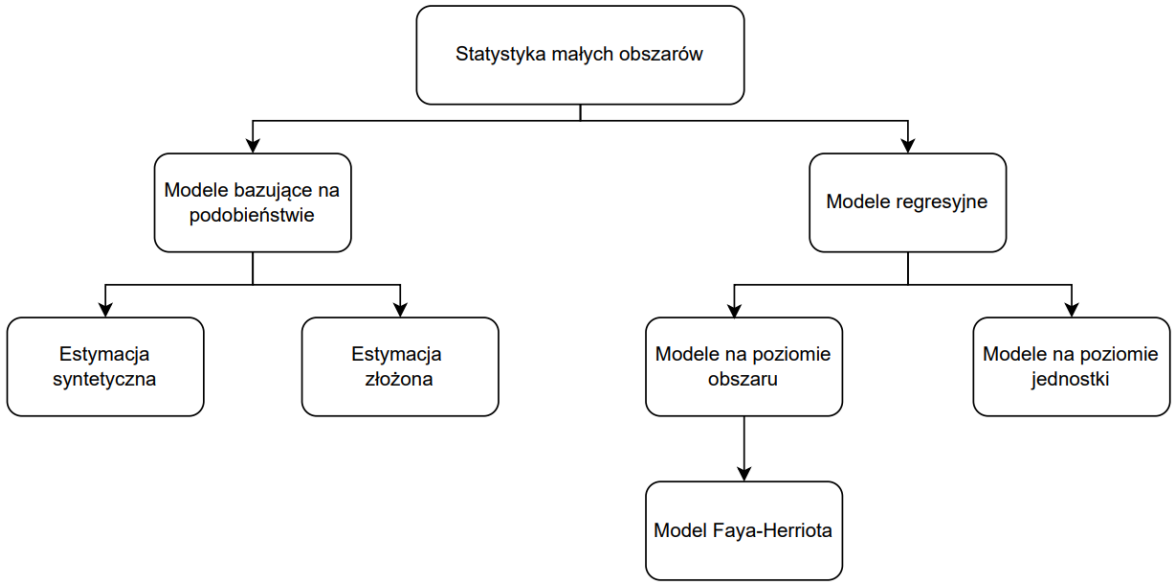
Badania próbkowe są powszechnie wykorzystywane do szacunków parametrów zarówno dla całych populacji generalnych, jak i dla ich części. W praktyce badań często pojawia się jednak potrzeba dostarczenia informacji (wiarygodnych szacunków) na bardzo niskim poziomie agregacji, dla przekrojów, których nie brano pod uwagę w momencie projektowania badania. W takich przypadkach klasyczne metody estymacji bezpośredniej, proponowane w ramach metody reprezentacyjnej, ze względu na zbyt małą liczebność próby (w skrajnym przypadku zerową) stają się bezużyteczne. Rozwiązaniem problemu może być Statystyka Małych Obszarów (SMO). Obejmuje ona zbiór metod estymacji, wykorzystujących tzw. informację wspomagającą, podchodzącą z różnych, dodatkowych źródeł danych takich jak: spisy, banki danych (np. BDL) czy rejestry administracyjne.

W SMO można wskazać trzy różne podejścia do procesu estymacji. Pierwsze – randomizacyjne (*design-based*) – odwołuje się do stosowanego w badaniu schematu losowania próby. Podejście drugie – modelowe (*model-based*), może być stosowane także w przypadku nielosowego wyboru prób. W podejściu tym jednak pojawia się problem związany z właściwą specyfikacją modelu. Można go uniknąć, stosując podejście mieszane zwane podejściem wspomaganym modelem (*model-assisted*) (Żądło, 2008).

Przekroje, dla których prowadzony jest szacunek, mogą być określone przestrzennie lub/i przedmiotowo, dlatego słowo „obszar" zastępowane jest często słowem „domena" („dziedzina") badania. Warto podkreślić, że określenia „mały obszar" czy „mała domena" odnoszą się do małej liczby jednostek w próbie. Nie dotyczą natomiast liczby jednostek w domenie czy też wielkości obszaru ze względu na zajmowaną powierzchnię.

SMO odwołuje się do dwóch rodzajów estymatorów: bezpośrednich i pośrednich. Estymatory bezpośrednie to estymatory klasyczne, znane z literatury metody reprezentacyjnej (Bracha, 1996; Molina i Rao, 2015; Zasępa, 1972; Żądło, 2015). Wykorzystują informacje dotyczące jedynie badanego małego obszaru/domeny. Estymatory pośrednie natomiast wykorzystują informacje z innych domen lub/i innych przedziałów czasowych, niebędących przedmiotem badania, co ma wpływ na poprawę efektywności estymacji. Źródłem dodatkowych informacji są tzw. zmienne pomocnicze (*auxiliary variables*), które są skorelowane ze zmienną badaną, a ich wartości znane są dla wszystkich jednostek populacji. Estymatory pośrednie są zwykle obciążone, ale w przypadku mało licznych prób cechują się niższymi wartościami wariancji niż estymatory bezpośrednie.

Rysunek 12.1 przedstawia podstawowy podział metod statystyki małych obszarów prezentowanych w tym rozdziale.



Rys. 12.1. Podział metod statystyki małych obszarów

Źródło: opracowanie własne.

Przykładem estymatora bezpośredniego, który jest najczęściej przywoływany zarówno w literaturze, jak i w praktyce badań statystycznych, jest estymator Horwitza-Thompsona (HT) (1952). Oszacowanie wartości globalnej w domenie d dla dowolnego schematu losowania uzyskuje się z wykorzystaniem wzoru

$$\hat{Y}_d^{HT} = \sum_{i=1}^{n_d} y_{di} w_{di}, \tag{12.1}$$

gdzie: $\hat{Y}_d^{HT}$ – estymator HT wartości globalnej zmiennej y w d-tej domenie, n_d – liczebność próby w d-tej domenie, y_{di} – wartość zmiennej y dla i-tej jednostki w d-tej domenie, w_{di} – wartość wagi wynikającej ze schematu losowania dla i-tej jednostki w d-tej domenie. Wartość wagi (w dużym uproszczeniu) informuje ile jednostek w populacji reprezentuje jednostka wylosowana do badania.

Estymator bezpośredni HT wykorzystuje wyłącznie dane pochodzące z próby i dla dużych wartości n_d jest nieobciążony i efektywny. Niemniej jednak mała liczebność próby sprawia, że charakteryzuje się niską precyzją szacunku. Żądło (2008) podaje następujący wzór na ocenę wariancji estymatora Horvitza-Thompsona:

$$\widehat{D}^2(\hat{Y}_d^{HT}) = \sum_{i=1}^{n_d} \sum_{k=1}^{n_d} \frac{y_i y_k}{\pi_i \pi_k} \frac{\pi_{ik} - \pi_i \pi_k}{\pi_{ik}}, \tag{12.2}$$

gdzie: y_i – wartość cechy y dla i-tej jednostki w badaniu, y_k – wartość zmiennej y dla k-tej jednostki w badaniu, π_i – prawdopodobieństwo inkluzji pierwszego rzędu jednostki i, π_k – prawdopodobieństwo inkluzji pierwszego rzędu jednostki k, π_{ik} – prawdopodobieństwo inkluzji drugiego rzędu jednostek i oraz k.

Prawdopodobieństwo inkluzji pierwszego rzędu to prawdopodobieństwo, że dana jednostka populacji znajdzie się w próbie po n losowaniach z populacji. Prawdopodobieństwo inkluzji

drugiego rzędu to prawdopodobieństwo, że dwie dane jednostki populacji znajdą się w próbie po n losowaniach z populacji.

Należy dodać, że wówczas, gdy dana domena nie jest reprezentowana w próbie ($n_d = 0$), zastosowanie tego rodzaju estymacji staje się niemożliwe. W takiej sytuacji, jak również w przypadku zbyt niskiej precyzji szacunku, rozwiązaniem może być zastąpienie estymacji bezpośredniej estymacją pośrednią. W dalszej części rozdziału zostaną przedstawione trzy rodzaje estymacji pośredniej: dwa reprezentujące podejście randomizacyjne (estymacja syntetyczna i złożona) oraz metoda estymacji reprezentująca podejście oparte na modelu Faya-Herriota. Należą one do grupy podstawowych metod SMO i stanowią punkt wyjścia do wykorzystywanych obecnie bardziej zaawansowanych technik szacunku, takich jak metoda EB wykorzystująca symulacje Monte Carlo (Molina i Rao, 2010; Wawrowski, 2020) czy regresja kwantylowa (Beręsewicz i in., 2018). Stosowanie ich wymaga jednak dostępu do danych jednostkowych, nieudostępnianych przez BDL z uwagi na poufność danych, dlatego w niniejszym rozdziale ograniczyliśmy się do przedstawienia wybranych podejść, starając się, na ich przykładzie, przede wszystkim przekazać główną ideę i możliwości, jakie stwarza włączenie do analizy metod SMO, w tym estymacji pośredniej. Czytelnikowi chcącemu poszerzyć wiedzę dotyczącą praktycznego stosowania estymacji pośredniej z wykorzystaniem programu R rekomendujemy lekturę artykułów dotyczących modeli przestrzennych (Pratesi i Salvati, 2008) oraz przestrzenno-czasowych (Marhuenda i in., 2013), które są oprogramowane w pakiecie `sae` (Molina i Marhuenda, 2015). Z kolei w pakietach `sae2` (Fay i Diallo, 2023) i `saery` (Esteban i in., 2014) dostępne są warianty modelu Faya-Herriota dla danych panelowych (Rao i Yu, 1994). Odporny model Faya-Herriota (Sinha i Rao, 2009) jest oprogramowany w pakiecie `saeRobust` (Warnholz, 2023). Z kolei pakiet `emdi` (Kreutzmann i in., 2019) jest kompleksowym interfejsem do tworzenia i wizualizacji najpopularniejszych modeli SMO. Bazując na danych z BDL, można także szacować wielomianowe modele mieszane (Lopez-Vizcaino i in., 2013) korzystając z pakietu `mme` (Lopez-Vizcaino i in., 2019). Wśród przykładowych obszarów zastosowań SMO można wyróżnić badania dotyczące ubóstwa (Molina i Rao, 2010; Wawrowski, 2014; 2016), rynku pracy (Gołata, 2004; Molina i in., 2007; Wilak, 2014), niepełnosprawności (Elazar i Conn, 2004; Klimanek i in., 2018) oraz przedsiębiorczości (Chandra i in., 2012; Dehnel, 2010; 2014; Dehnel i Wawrowski, 2020).

12.2. Ocena jakości szacunku

W ocenie jakości szacunku bierze się pod uwagę jego precyzję, która zależy od wartości błędów losowych, oraz dokładność uwzględniającą dodatkowo obciążenie estymacji (błędy nielosowe)[1]. Jedną z miar dokładności oceny estymatora jest pierwiastek kwadratowy ze średniokwadratowego błędu estymacji $RMSE(\hat{\theta}) = \sqrt{MSE(\hat{\theta})}$ mówiący, o ile przeciętnie wartości estymatora odchylają się od rzeczywistej wartości parametru[2]. W niniejszym rozdziale symbol $\hat{\theta}$ będzie oznaczał estymator dowolnej funkcji jak np. w przypadku przedstawionego powyżej estymatora bezpośredniego HT $\hat{\theta}^{HT} = \hat{Y}^{HT}$.

[1] Zagadnienie dotyczące oceny dokładności i precyzji estymacji zarówno w przypadku podejścia randomizacyjnego, jak i modelowego zostały opisane m.in. w (Cassel i in., 1977; Żądło, 2008).

[2] W praktyce badań statystycznych, dysponując tylko danymi pochodzącymi z próby można jedynie ocenić dokładność (precyzję) estymacji, szacując wartość błędu średniokwadratowego (wariancji) (Żądło, 2008, s. 24).

Wśród miar oceniających precyzję estymatora wyróżnić można średni błąd szacunku $D(\hat{\theta})$, będący pierwiastkiem z wariancji estymatora, informujący o ile przeciętnie wartości estymatora odchylają się od wartości oczekiwanej estymatora oraz względny średni błąd szacunku $CV(\hat{\theta})$ mówiący, jaki procent wartości parametru stanowi średni błąd szacunku.

$$CV(\hat{\theta}) = \frac{\sqrt{D^2(\hat{\theta})}}{\theta} = \frac{D(\hat{\theta})}{\theta}, \tag{12.3}$$

gdzie: $\hat{\theta}$ – estymator funkcji θ, $D^2(\hat{\theta})$ – wariancja estymatora $\hat{\theta}$.

W przypadku estymatorów nieobciążonych w miejsce interpretacji $RMSE(\hat{\theta})$ można stosować interpretację $D(\hat{\theta})$ (Żądło, 2008).

W badaniach prowadzonych przez statystykę publiczną powszechnie używaną miarą jakości jest ocena względnego średniego błędu szacunku – wskaźnik precyzji będący udziałem oceny średniego błędu szacunku w ocenie estymatora.

$$\widehat{CV}(\hat{\theta}) = \frac{\hat{D}(\hat{\theta})}{\hat{\theta}}. \tag{12.4}$$

Wskaźnik precyzji jest podawany w procentach i zgodnie z zaleceniami GUS do publikacji dopuszczane są oszacowania, dla których jego wartość nie przekracza 20% (Główny Urząd Statystyczny [GUS], 2013).

12.3. Estymacja syntetyczna

Estymacja syntetyczna oparta jest na założeniu, że istnieje relacja podobieństwa między badanymi domenami a populacją[3], którą domeny te współtworzą (por. rys. 12.1). Spełnienie tego założenia pozwala na rozszacowanie oceny estymatora otrzymanej dla populacji na części odpowiadające badanym domenom. Rozszacowania dokonuje się proporcjonalnie do liczby jednostek lub biorąc pod uwagę wartości cechy pomocniczej w badanych domenach. W konstrukcji pośredniego estymatora syntetycznego parametru domeny wykorzystuje się estymator bezpośredni parametru populacji, w skład której wchodzi badana domena. Jednym z podstawowych estymatorów syntetycznych jest estymator ilorazowy dany wzorem

$$\hat{Y}_d^{SYN} = \frac{X_d}{X}\hat{Y}^{HT}, \tag{12.5}$$

gdzie: X_d – wartość globalna zmiennej pomocniczej x w badanej domenie, X – wartość globalna zmiennej pomocniczej x w populacji, $\hat{Y}^{HT}$ – estymator HT wartości globalnej zmiennej y w populacji.

Wariancja estymatora przybiera z reguły niewielką wartość, ponieważ zależy wyłącznie od wariancji i kowariancji estymatora bezpośredniego wartości globalnej w populacji $\hat{Y}^{HT}$ (Gołata, 2004).

[3] Relacja podobieństwa nie musi dotyczyć całej populacji, może odnosić się do jej części – podpopulacji.

Wariancja estymatora syntetycznego daje się łatwo oszacować, jeśli znamy ocenę wariancji dla estymatora bezpośredniego:

$$\widehat{D}^2(\hat{Y}_d^{SYN}) = \left(\frac{X_d}{X}\right)^2 \widehat{D}^2(\hat{Y}^{HT}), \tag{12.6}$$

gdzie: $\widehat{D}^2(\hat{Y}^{HT})$ – ocena wariancji estymatora bezpośredniego HT w populacji.

Warto zauważyć, że ocena wskaźnika precyzji estymatora ilorazowego dla domeny będzie taka sama jak dla estymatora bezpośredniego w przypadku populacji.

Z kolei analizując obciążenie estymatora, można zauważyć, że jeśli estymator bezpośredni wykorzystany do konstrukcji pośredniego estymatora syntetycznego jest nieobciążony, to estymator syntetyczny także jest nieobciążony ($E(\hat{Y}_d^{SYN}) - Y_d = 0$). Własność ta jest prawdziwa jedynie wówczas, gdy ilorazy wartości globalnych zmiennej badanej i pomocniczej w domenie oraz w populacji są sobie równe. Jako że założenie to jest bardzo mocne, rzadko jest spełniane w praktycznym zastosowaniu estymatorów. Prowadzi to do dużych obciążeń szacunku. Rozwiązaniem problemu znacznego obciążenia estymacji może być zastosowanie estymacji złożonej.

12.4. Estymacja złożona

Jedno z podejść stosowanych w ramach estymacji złożonej reprezentują estymatory, których zastosowanie ma na celu zbilansowanie potencjalnego obciążenia estymatora syntetycznego z niestabilnością estymatora bezpośredniego przejawiającą się dużą wartością wariancji (por. rys. 12.1). Estymatory złożone konstruowane są tu jako średnia ważona z wymienionych dwóch estymatorów. Przykładem może być estymator wyrażony wzorem

$$\hat{Y}_d^{COMP} = \gamma_d \hat{Y}_d^{HT} + (1 - \gamma_d)\hat{Y}_d^{SYN}, \tag{12.7}$$

gdzie: $\hat{Y}_d^{HT}$ – estymator bezpośredni w d-tej domenie, $\hat{Y}_d^{SYN}$ – estymator syntetyczny w d-tej domenie, γ_d – odpowiednio dobrana waga w d-tej domenie ($0 \leq \gamma_d \leq 1$).

Ideą określenia wagi na podstawie posiadanych danych jest wyznaczenie takiej jej wartości, która będzie minimalizowała średni błąd średniokwadratowy estymatora złożonego:

$$\gamma_d = \frac{\widehat{D}^2(\hat{Y}_d^{SYN})}{(\widehat{D}^2(\hat{Y}_d^{SYN}) - \widehat{D}^2(\hat{Y}_d^{HT}))}, \tag{12.8}$$

gdzie: $\widehat{D}^2(\hat{Y}_d^{SYN})$ – wariancja estymatora syntetycznego w d-tej domenie, $\widehat{D}^2(\hat{Y}_d^{HT})$ – wariancja estymatora bezpośredniego w d-tej domenie.

Przedstawiona waga ma inną wartość dla każdej z badanych domen.

Trudnością jest wyznaczenie błędu oszacowania dla estymatora złożonego. Jednym z rozwiązań może być wykorzystanie metody *bootstrap*, która jednak wymaga dostępu do danych jednostkowych z badania reprezentacyjnego.

12.5. Podejście modelowe

Model Faya-Herriota (1979) jest modelem na poziomie obszaru, co oznacza, że jego aplikacja nie wymaga trudno dostępnych danych jednostkowych[4] (por. rys. 12.1). Do jej zastosowania wystarczy dostęp do danych zagregowanych, na poziomie badanej domeny/badanego obszaru. Takie dane można pozyskać np. z Banku Danych Lokalnych oraz z innych publicznych baz danych.

Budowa modelu Faya-Herriota przebiega w dwóch etapach. Zakłada się, że oszacowanie bezpośrednie ($\hat{\theta}_d$) parametru (θ_d) jest nieobciążone i można je zapisać jako:

$$\hat{\theta}_d = \theta_d + e_d, \tag{12.9}$$

gdzie: $e_d \overset{ind}{\sim} N(0, \psi_d)$.

W praktyce wariancja ψ_d nie jest znana, w związku z czym jest estymowana na podstawie próby.

Na drugim etapie model Faya-Herriota traktuje θ_d jako zmienną objaśnianą w modelu liniowym z jednym efektem losowym na poziomie obszaru:

$$\theta_d = \boldsymbol{x}_d{}'\boldsymbol{\beta} + u_d, \tag{12.10}$$

gdzie: $\boldsymbol{x}_d$ – wektor zmiennych objaśniających dla obszaru d o wymiarach $p \times 1$, $\boldsymbol{\beta}$ – wektor parametrów regresji o wymiarach $p \times 1$, u_d – efekt obszaru o $u_d \overset{iid}{\sim} N(0, \sigma_u^2)$.

W związku z tym model Faya-Herriota jest wariantem modelu liniowego z jednostkową strukturą kowariancji:

$$\theta_d = \boldsymbol{x}_d{}'\boldsymbol{\beta} + u_d + e_d. \tag{12.11}$$

Estymatorem tego modelu jest średnia ważona oszacowania bezpośredniego $\hat{Y}_d^{HT}$ oraz oszacowania syntetycznego regresyjnego $\boldsymbol{x}_d{}'\widehat{\boldsymbol{\beta}}$. Waga $\gamma_d \in \langle 0,1 \rangle$ mierzy niepewność wynikającą z opisu szacowanej wartości przez model regresyjny. W zależności od wariancji z próby ψ_d oraz wariancji międzyobszarowej σ_u^2 większy lub mniejszy udział będzie przypisywany szacunkowi bezpośredniemu.

$$\hat{Y}_d^{FH} = \boldsymbol{x}_d{}'\widehat{\boldsymbol{\beta}} + \hat{u}_d = \hat{\gamma}_d \hat{Y}_d^{HT} + (1 - \hat{\gamma}_d)\boldsymbol{x}_d{}'\widehat{\boldsymbol{\beta}},\ d = 1, \dots, D, \tag{12.12}$$

gdzie: $\hat{\gamma}_d = \frac{\hat{\sigma}_u^2}{\hat{\sigma}_u^2 + \psi_d}$, a $\widehat{\boldsymbol{\beta}}$ jest wyznaczone w wykorzystaniem ważonej metody najmniejszych kwadratów:

$$\hat{\beta} = \left(\sum_{d=1}^{D} \hat{\gamma}_d \boldsymbol{x}_d \boldsymbol{x}_d{}'\right)^{-1} \sum_{d=1}^{D} \hat{\gamma}_d \boldsymbol{x}_d \hat{Y}_d^{HT}. \tag{12.13}$$

Przy stosowaniu metod opartych na modelu ważne jest spełnienie założeń teoretycznych. W przypadku modelu Faya-Herriota chodzi szczególnie o założenie dotyczące normalności efektów losowych ($\hat{u}_d = \hat{\gamma}_d(\hat{Y}_d^{HT} - \boldsymbol{x}_d{}'\widehat{\boldsymbol{\beta}})$) oraz reszt ($r = \hat{Y}_d^{HT} - \boldsymbol{x}_d{}'\widehat{\boldsymbol{\beta}}$).

Miarą jakości uzyskanych oszacowań dla tego podejścia jest błąd średniokwadratowy. Uwzględnia on wiele źródeł zmienności i nie jest porównywalny z wariancją oszacowań bezpośrednich. Metody wyznaczania porównywalnych miar precyzji opisane są w artykule Pfeffermanna i Ben-Hura (2019).

[4] Zawartych np. w rejestrach administracyjnych czy spisach powszechnych.

12.6. Zastosowanie z wykorzystaniem programu R dla danych z BDL

W tej części będziemy korzystać z następujących pakietów programu R (R Core Team, 2023): `tidyverse` (do przetwarzania i wizualizacji danych), `bdl` (do pobierania danych), `sae` (do modelu Faya-Herriota) oraz `patchwork` (do łączenia wykresów).

Ze względu na dostępność danych prezentowane przykłady będą dotyczyły rynku pracy.

Estymacja syntetyczna – liczba osób bezrobotnych na poziomie podregionów (BAEL)

Najpierw pobieramy z BDL dane będące oszacowaniami wartości estymatora bezpośredniego – liczby osób bezrobotnych na poziomie województw oraz wskaźnika precyzji, a następnie te dane łączymy (skrypt 12.1a). Liczba osób bezrobotnych jest wyrażona w tysiącach osób, natomiast wskaźnik precyzji (zdefiniowany wzorem (12.4)) w procentach. W dalszych obliczeniach będziemy korzystać z oceny wariancji oszacowania, należy zatem przeprowadzić odpowiednie przekształcenie w celu jej wyznaczenia.

Skrypt 12.1a

```
bezr_woj<-get_data_by_variable(varId=479083, unitLevel=2, year=2018) %>%
  select(id, name, y_ht=val)
bezr_woj_wp<-get_data_by_variable(varId=479065, unitLevel=2, year=2018) %>%
  select(id, name, y_ht_cv=val)
bezr<-inner_join(bezr_woj, bezr_woj_wp) %>%
  mutate(y_ht_se=y_ht*y_ht_cv/100, y_ht_var=y_ht_se^2)
print(head(bezr))
```

Rezultatem realizacji skryptu 12.1a jest utworzenie trzech nowych zbiorów danych i wyświetlenie w konsoli pierwszych 6 obserwacji ze zbioru `bezr`.

```
# A tibble: 6 x 6
  id           name                y_ht y_ht_cv y_ht_se y_ht_var
  <chr>        <chr>              <dbl>   <dbl>   <dbl>    <dbl>
1 011200000000 MAŁOPOLSKIE           43    10.9    4.69    22.0
2 012400000000 ŚLĄSKIE               67     8.5    5.70    32.4
3 020800000000 LUBUSKIE              13     9.2    1.20     1.43
4 023000000000 WIELKOPOLSKIE         36    12.6    4.54    20.6
5 023200000000 ZACHODNIOPOMORSKIE    28     8.9    2.49     6.21
6 030200000000 DOLNOŚLĄSKIE          42    11.1    4.66    21.7
```

Następnie wykorzystamy zmienną pomocniczą – liczbę osób w wieku produkcyjnym, do rozszacowania ocen bezpośrednich na poziom podregionów. Potrzebujemy w tym celu wartości zmiennej pomocniczej na poziomie województw i podregionów (skrypt 12.1b). Analizowane do tej pory zbiory będziemy musieli połączyć na poziomie podregionów, zatem w zbiorze `x_podreg` tworzymy identyfikator województwa. Spośród wszystkich 12 znaków zmiennej `id` pierwsze cztery to identyfikator województwa, a dwa kolejne podregionu.

Skrypt 12.1b

```
x_woj<-get_data_by_variable(varId=72292, unitLevel=2, year=2018) %>%
  select(id, name_woj=name, x_woj=val)
x_podreg<-get_data_by_variable(varId=72292, unitLevel=4, year=2018) %>%
  select(id_podreg=id, name_podreg=name, x_podreg=val)
x_podreg <- x_podreg %>%
  mutate(id=substr(id_podreg,1,4), id=str_pad(id, 12, "right", 0))
print(head(x_podreg))
```

W wyniku działania kodu w skrypcie 12.1b uzyskuje się dwa nowe zbiory danych oraz wydruk 6 pierwszych obserwacji ze zbioru `x_podreg`.

```
# A tibble: 6 x 4
  id_podreg    name_podreg             x_podreg id
  <chr>        <chr>                      <dbl> <chr>
1 011212000000 PODREGION KRAKOWSKI       471237 011200000000
2 011212100000 PODREGION MIASTO KRAKÓW   477128 011200000000
3 011212200000 PODREGION NOWOSĄDECKI     351218 011200000000
4 011212300000 PODREGION OŚWIĘCIMSKI     350306 011200000000
5 011212400000 PODREGION TARNOWSKI       301895 011200000000
6 011216900000 PODREGION NOWOTARSKI      223706 011200000000
```

W powyższym skrypcie najpierw zostały usunięte pierwsze 4 znaki z identyfikatora podregionu, a następnie po prawej stronie tego ciągu znaków dodano zera, tak aby końcowa długość tekstu wynosiła 12 znaków. Na końcu łączymy wszystkie przygotowane zbiory danych. Na podstawie takiego zbioru możemy przeprowadzić obliczenia w celu otrzymania ocen estymatora syntetycznego ilorazowego i ocen wariancji (skrypt 12.1c). Przyjmujemy założenie, że struktura podregionu jest taka sama jak województwa.

Skrypt 12.1c

```
bezr_x<-left_join(x_podreg, inner_join(x_woj, bezr), by="id")
bezr_x<-bezr_x %>%
  mutate(y_syn=x_podreg/x_woj*y_ht, y_syn_var=(x_podreg/x_woj)^2*y_ht_var,
    y_syn_se=sqrt(y_syn_var))
bezr_syn<-bezr_x %>%
  select(name_podreg, y_syn, y_syn_se)
print(head(bezr_syn))
```

Wynikiem działania skryptu 12.1c jest nowy zbiór danych `bezr_x` zawierający 19 kolumn, natomiast poniżej zaprezentowano 6 pierwszych obserwacji i 3 najważniejsze zmienne.

```
# A tibble: 6 x 3
  name_podreg             y_syn y_syn_se
  <chr>                   <dbl>    <dbl>
1 PODREGION KRAKOWSKI      9.31    1.02
2 PODREGION MIASTO KRAKÓW  9.43    1.03
3 PODREGION NOWOSĄDECKI    6.94    0.757
4 PODREGION OŚWIĘCIMSKI    6.92    0.755
5 PODREGION TARNOWSKI      5.97    0.650
6 PODREGION NOWOTARSKI     4.42    0.482
```

W rezultacie uzyskaliśmy oszacowania na poziomie podregionów dla liczby osób bezrobotnych wraz z błędem standardowym tych oszacowań. Warto dodać, że klasyczne metody estymacji nie pozwalają na dostarczenie szacunków dla analizowanej zmiennej na tak niskim poziomie jak podregiony. Zastosowanie estymacji syntetycznej proponowanej przez SMO stworzyło taką możliwość.

Estymacja złożona – liczba osób bezrobotnych na poziomie powiatów (NSP 2011)

Estymacja złożona jest możliwa, jeżeli mamy dostęp do oszacowań bezpośrednich lub możemy je wyznaczyć w przekroju domen, które są przedmiotem badania. Rozważmy zatem problem estymacji liczby osób bezrobotnych w przekroju powiatów na podstawie danych z Narodowego Spisu Powszechnego 2011. Oszacowania syntetyczne uzyskamy na podstawie danych z poziomu województw.

Najpierw pobieramy dane dotyczące oszacowań liczby osób bezrobotnych na poziomie województw oraz wartości wskaźnika precyzji. Przy okazji ograniczamy liczbę zmiennych w zbiorze danych oraz zmieniamy nazwy dwóch kolumn – z tego względu, że kolumna z pobieraną cechą ma zawsze nazwę `val`. Aby możliwe było połączenie tych zbiorów, nadajemy nowe nazwy: `woj_y_ht` dla oszacowań liczby osób bezrobotnych i `woj_y_ht_cv` dla wskaźnika precyzji. W kolejnym kroku łączymy oba zbiory i na podstawie wskaźnika precyzji wyznaczamy błąd standardowy oraz ocenę wariancji oszacowań na poziomie województw (skrypt 12.2a).

Skrypt 12.2a

```
nsp_bezr_woj_y_ht<-get_data_by_variable(varId=450241, unitLevel=2) %>%
  select(id, woj=name, woj_y_ht=val)
nsp_bezr_woj_y_cv<-get_data_by_variable(varId=450240, unitLevel=2) %>%
  select(id, woj=name, woj_y_ht_cv=val)
nsp_bezr_woj<-inner_join(nsp_bezr_woj_y_ht, nsp_bezr_woj_y_cv) %>%
  mutate(woj_y_ht_se=woj_y_ht*woj_y_ht_cv/100, woj_y_ht_var=woj_y_ht_se^2)
print(head(nsp_bezr_woj))
```

Uruchomienie skryptu 12.2a skutkuje stworzeniem 3 nowych zbiorów danych, a w konsoli zostaje wyświetlonych 6 pierwszych obserwacji ze zbioru `nsp_bezr_woj`.

```
# A tibble: 6 x 6
  id          woj             woj_y_ht woj_y_ht_cv woj_y_ht_se woj_y_ht_var
  <chr>       <chr>              <dbl>       <dbl>       <dbl>        <dbl>
1 0112000000~ MAŁOPOLSKIE       153505       0.69        1059.     1121872.
2 0124000000~ ŚLĄSKIE           228198       0.51        1164.     1354453.
3 0208000000~ LUBUSKIE           61245       0.79         484.      234097.
4 0230000000~ WIELKOPOLSKIE     160339       0.580        930.      864837.
5 0232000000~ ZACHODNIOPOMOR~   109744       0.69         757.      573403.
6 0302000000~ DOLNOŚLĄSKIE      164928       0.6          990.      979245.
```

Wartości z powyższego zbioru danych zostaną rozszacowane na poziom powiatów z wykorzystaniem estymatora syntetycznego ilorazowego.

Następnie pobieramy z BDL oszacowania stopy bezrobocia oraz ich oceny wskaźnika precyzji na poziomie powiatów, podobnie jak wcześniej zmieniając nazwy zmiennych i łącząc oba zbiory. Jako zmienną pomocniczą wykorzystamy liczbę osób w wieku produkcyjnym. Pobieramy te wartości na poziomie województw oraz powiatów. Połączenie tych zbiorów będzie możliwe po utworzeniu wspólnego klucza połączeniowego – identyfikatora województwa w zbiorze danych z wartościami dla powiatów. Przygotowane w taki sposób zbiory łączymy ze sobą, tak aby wszystkie zmienne były w jednym miejscu (skrypt 12.2b). Najpierw wyznaczymy oszacowanie syntetyczne dla liczby osób bezrobotnych na poziomie powiatów.

Skrypt 12.2b

```
nsp_bezr_pow_y_ht<-get_data_by_variable(varId=450241, unitLevel=5) %>%
  select(id, pow=name, pow_y_ht=val)
nsp_bezr_pow_y_cv<-get_data_by_variable(varId=450240, unitLevel=5) %>%
  select(id, pow=name, pow_y_ht_cv=val)
nsp_bezr_pow<-inner_join(nsp_bezr_pow_y_ht, nsp_bezr_pow_y_cv) %>%
  mutate(pow_y_ht_se=pow_y_ht*pow_y_ht_cv/100, pow_y_ht_var=pow_y_ht_se^2)
# zmienna pomocnicza
nsp_x_woj<-get_data_by_variable(varId=453979, unitLevel=2) %>%
  select(id, woj=name, x_woj=val)
nsp_x_pow<-get_data_by_variable(varId=453979, unitLevel=5) %>%
  select(id_pow=id, pow=name, x_pow=val)
# stworzenie klucza połączeniowego
nsp_x_pow<-nsp_x_pow %>%
  mutate(id=substr(id_pow,1,4), id=str_pad(id, 12, "right", 0))
# łączenie wszystkich zbiorów
nsp_bezr_x<-left_join(inner_join(nsp_x_pow, nsp_bezr_pow,
  by=c("id_pow"="id", "pow")), inner_join(nsp_x_woj, nsp_bezr_woj), by="id")
# oszacowania syntetyczne
nsp_bezr_x<-nsp_bezr_x %>%
  mutate(pow_y_syn=round(x_pow/x_woj*woj_y_ht),
  pow_y_syn_var=(x_pow/x_woj)^2*woj_y_ht_var,
  pow_y_syn_se=sqrt(pow_y_syn_var))
nsp_bezr_ht_syn <- nsp_bezr_x %>%
  select(id_pow, pow, pow_y_ht, pow_y_ht_se, pow_y_syn, pow_y_syn_se)
print(head(nsp_bezr_ht_syn))
```

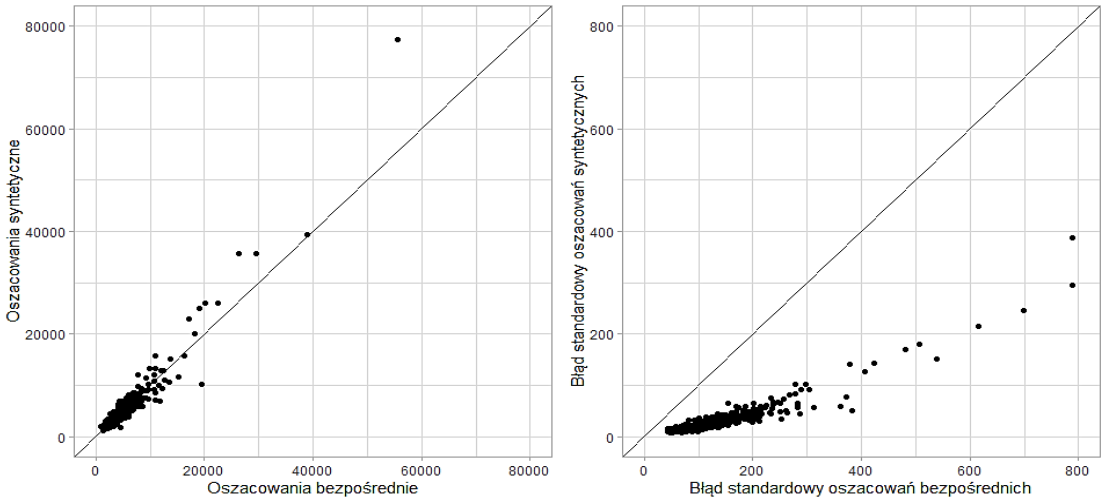
W wyniku realizacji skryptu 12.2b tworzonych jest wiele różnych zbiorów danych, natomiast najważniejszym efektem jest zbiór `nsp_bezr_x`, który zawiera wyniki przeprowadzonej estymacji syntetycznej. Poniżej zaprezentowano 6 z nich dla 6 pierwszych obserwacji.

```
# A tibble: 6 x 6
  id_pow       pow              pow_y_ht pow_y_ht_se pow_y_syn pow_y_syn_se
  <chr>        <chr>               <dbl>       <dbl>     <dbl>        <dbl>
1 011212001000 Powiat bocheński     4409       133.       4725         32.6
2 011212006000 Powiat krakowski    10729       278.      12044         83.1
3 011212008000 Powiat miechows~     2162        79.8      2242         15.5
4 011212009000 Powiat myślenic~     5601       150.       5585         38.5
5 011212014000 Powiat proszowi~     1189        53.3      1997         13.8
6 011212019000 Powiat wielicki      5050       144.       5298         36.6
```

Rysunek 12.2 przedstawia porównanie oszacowań uzyskanych na podstawie dwóch estymatorów: bezpośredniego i syntetycznego (skrypt 12.2c).

Skrypt 12.2c

```
par(mfrow=c(1,2),pty="s")
p1<-ggplot(nsp_bezr_x, aes(pow_y_ht, pow_y_syn)) +
  geom_point() +
  geom_abline(slope = 1) +
  xlim(0,80000) +
  ylim(0,80000) +
  xlab("Oszacowania bezpośrednie") +
  ylab("Oszacowania syntetyczne") +
  theme_light()
p2<-ggplot(nsp_bezr_x, aes(pow_y_ht_se, pow_y_syn_se)) +
  geom_point() +
  geom_abline(slope = 1) +
  xlim(0,800) +
  ylim(0,800) +
  xlab("Błąd standardowy oszacowań bezpośrednich") +
  ylab("Błąd standardowy oszacowań syntetycznych") +
  theme_light()
print(p1+p2)
```



Rys. 12.2. Oszacowania uzyskane na podstawie dwóch estymatorów: bezpośredniego i syntetycznego wraz z ich błędami standardowymi

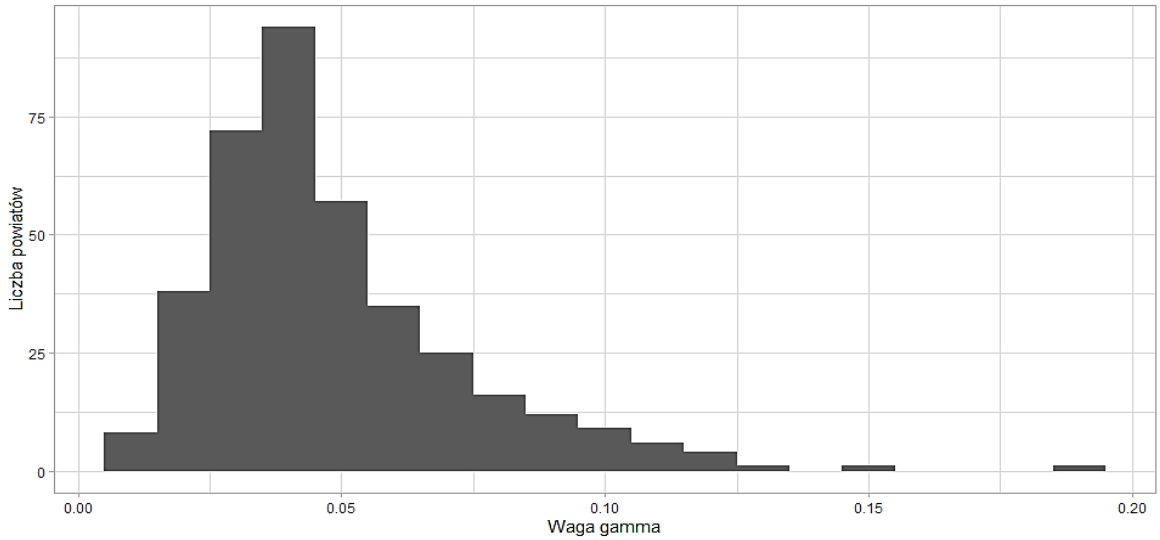
Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Można zauważyć, że wartości oszacowań liczby osób bezrobotnych w większości przypadków są zbliżone. Różnice pojawiają się dla dużych miast – w takiej sytuacji wartości są zwykle przeszacowane. Wszystkie oszacowania syntetyczne mają mniejsze błędy standardowe aniżeli oszacowania bezpośrednie.

W kolejnym kroku na podstawie ocen wariancji dla oszacowań bezpośrednich oraz syntetycznych wyznaczamy wagę γ (skrypt 12.2d). Waga ta opisana wzorem 12.8 jest wyznaczana na podstawie wariancji (niepewności) oszacowania bezpośredniego oraz syntetycznego. Wysokie wartości będą oznaczały dobrą jakość oszacowań bezpośrednich, a niskie – konieczność zastosowania podejścia wykorzystującego oszacowania syntetyczne. Rozkład wartości tej wagi prezentuje rys. 12.3.

Skrypt 12.2d

```
nsp_bezr_x<-nsp_bezr_x %>%
  mutate(gamma=pow_y_syn_var/(pow_y_syn_var+pow_y_ht_var))
p <- ggplot(nsp_bezr_x, aes(gamma)) +
  geom_histogram(binwidth = 0.01) +
  xlab("Waga gamma") +
  ylab("Liczba powiatów") +
  theme_light()
print(p)
```



Rys. 12.3. Rozkład wagi γ

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

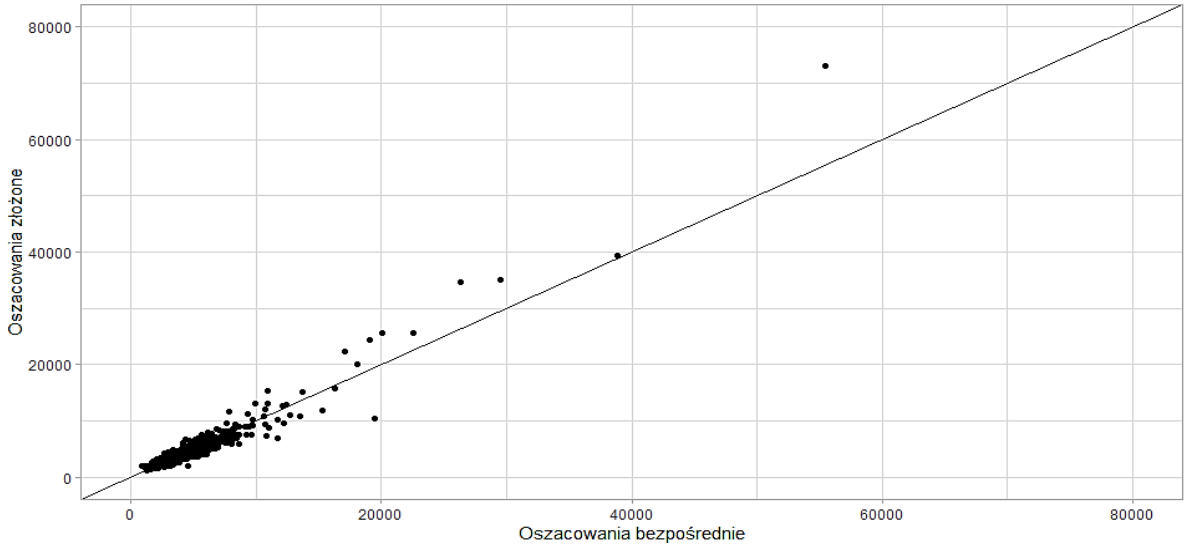
Wartości wagi γ są bardzo małe, co oznacza, że wartości estymatora złożonego będą w większości pochodziły od oszacowań syntetycznych. Wykorzystując te wartości, wyznaczamy oszacowania złożone, a uzyskane wyniki porównamy z oszacowaniami bezpośrednimi (skrypt 12.2e). Porównanie oszacowań znajduje się na rys. 12.4.

Skrypt 12.2e

```
nsp_bezr_x<-nsp_bezr_x %>%
  mutate(pow_y_comp=round(gamma*pow_y_ht+(1-gamma)*pow_y_syn))
p<-ggplot(nsp_bezr_x, aes(pow_y_ht, pow_y_comp)) +
  geom_point() +
  geom_abline(slope = 1) +
  xlim(0,80000) +
  ylim(0,80000) +
  xlab("Oszacowania bezpośrednie") +
  ylab("Oszacowania złożone") +
  theme_light()
print(p)
nsp_bezr_ht_syn_comp<-nsp_bezr_x %>%
  select(id_pow, pow, pow_y_ht, pow_y_syn, pow_y_comp)
print(head(nsp_bezr_ht_syn_comp))
```

Rezultatem działania skryptu 12.2e jest zbiór z oszacowaniami bezpośrednimi, syntetycznymi oraz złożonymi – 6 pierwszych obserwacji zostało wypisanych poniżej:

```
# A tibble: 6 x 5
  id_pow       pow                 pow_y_ht pow_y_syn pow_y_comp
  <chr>        <chr>                  <dbl>     <dbl>      <dbl>
1 011212001000 Powiat bocheński        4409      4725       4707
2 011212006000 Powiat krakowski       10729     12044      11936
3 011212008000 Powiat miechowski       2162      2242       2239
4 011212009000 Powiat myślenicki       5601      5585       5586
5 011212014000 Powiat proszowicki      1189      1997       1946
6 011212019000 Powiat wielicki         5050      5298       5283
```



Rys. 12.4. Oszacowania uzyskane na podstawie dwóch estymatorów: bezpośredniego i złożonego

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Jak można było się spodziewać, otrzymane oszacowania złożone są bardzo zbliżone do oszacowań syntetycznych. Niestety na podstawie danych z BDL nie jesteśmy w stanie wyznaczyć błędów standardowych oszacowań złożonych. W tym celu należałoby pozyskać dane jednostkowe z badania reprezentacyjnego i zastosować metodę *bootstrap*.

Model Faya-Herriota – stopa bezrobocia na poziomie powiatów (NSP 2011)

Estymator syntetyczny ilorazowy wykorzystywał jedną zmienną pomocniczą, podobnie jak estymator złożony będący średnią ważoną oszacowań bezpośrednich i syntetycznych. Model Faya-Herriota (*FH*) umożliwia uwzględnienie większej liczby zmiennych pomocniczych dzięki zastosowaniu liniowego modelu mieszanego. Stwarza to dodatkową możliwość poprawy precyzji szacunku. Wpływ na jakość oszacowań, jaki może mieć zastąpienie modelem *FH* klasycznej estymacji bezpośredniej, został zilustrowany w poniższym przykładzie. Wykorzystujemy w nim dane dotyczące szacunków stopy bezrobocia na poziomie powiatów, dokonanych na podstawie zwykłego estymatora bezpośredniego, dostępnych w BDL.

Najpierw pobieramy z banku wartości oszacowań badanej zmiennej oraz wskaźnika precyzji, wybieramy niezbędne kolumny, przy okazji zmieniając ich nazwy (podobnie jak w skrypcie 12.2a), a następnie dzielimy przez 100, w celu ujednolicenia jednostek, w których przedstawione są zmienne (skrypt 12.3a).

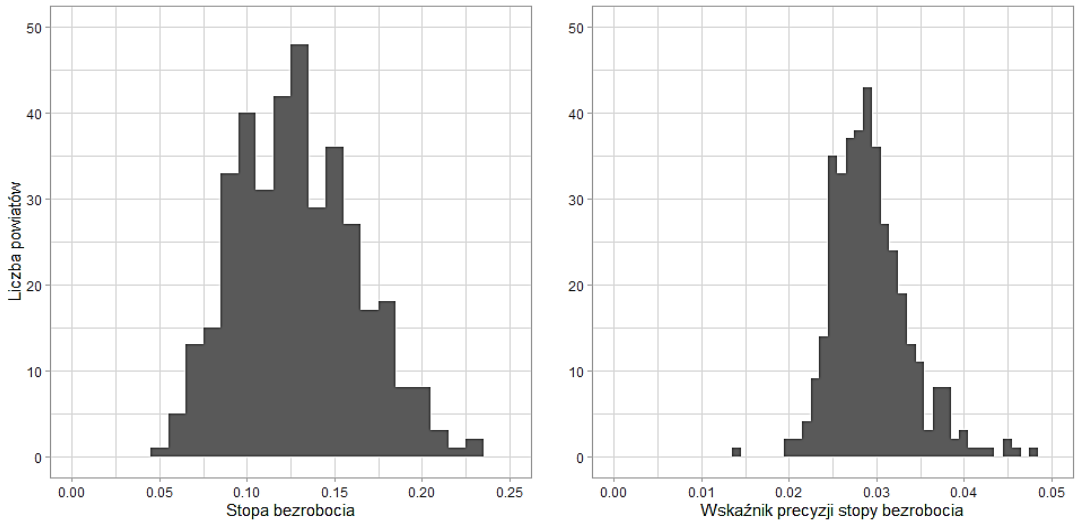
Skrypt 12.3a

```
nsp_stopa_bezr_y_ht<-get_data_by_variable(varId=450231, unitLevel=5) %>%
  dplyr::select(id, name, y_ht=val) %>%
  mutate(y_ht=y_ht/100)
nsp_stopa_bezr_y_cv<-get_data_by_variable(varId=450230, unitLevel=5) %>%
  dplyr::select(id, name, y_ht_cv=val) %>%
  mutate(y_ht_cv=y_ht_cv/100)
```

Na rysunku 12.5 można porównać rozkłady pobranych cech (skrypt 12.3b).

Skrypt 12.3b

```
r1<-ggplot(nsp_stopa_bezr_y_ht, aes(x=y_ht)) +
  geom_histogram(binwidth=0.01) +
  xlim(0,0.25) +
  ylim(0,50) +
  xlab("Stopa bezrobocia") +
  ylab("Liczba powiatów") +
  theme_light()
r2<-ggplot(nsp_stopa_bezr_y_cv, aes(x=y_ht_cv)) +
  geom_histogram(binwidth=0.001) +
  xlim(0,0.05) +
  ylim(0,50) +
  xlab("Wskaźnik precyzji stopy bezrobocia") +
  ylab("") +
  theme_light()
print(r1+r2)
```



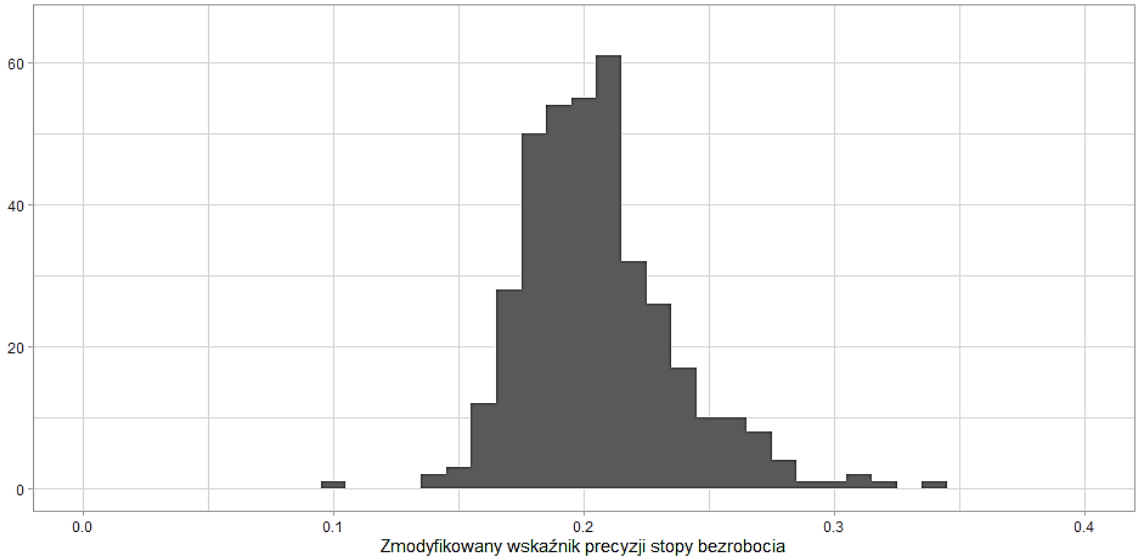
Rys. 12.5. Rozkład oszacowań stopy bezrobocia oraz wskaźnika precyzji

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Oceny wartości stopy bezrobocia znajdują się w przedziale od 5 do 25% z dominantą na poziomie 12%. Z kolei oceny wskaźnika precyzji wynoszą od 1,5 do 5%, co oznacza, że oszacowania stopy bezrobocia są dobrej jakości – błąd szacunku nigdzie nie przekracza 5% oszacowania stopy bezrobocia. Taki poziom precyzji nie wymaga stosowania do estymacji parametrów metod proponowanych przez SMO. Chcąc jednak zaprezentować sposób, w jaki można zaaplikować model *FH*, korzystając z dostępnego oprogramowania w R, dane pochodzące z BDL na potrzeby przykładu zostały poddane modyfikacji. W celu „sztucznego pogorszenia" jakości estymacji, tak by oszacowania charakteryzowały się wysokimi (nieakceptowalnymi) wartościami wskaźnika precyzji, przemnożono wartości błędu szacunku przez stałą (skrypt 12.3c). Rozkład zmodyfikowanych ocen wskaźnika precyzji pokazano na rys. 12.6.

Skrypt 12.3c

```
mnoznik<-7
nsp_stopa_bezr_y<-inner_join(nsp_stopa_bezr_y_ht, nsp_stopa_bezr_y_cv) %>%
  mutate(y_ht_se=(y_ht*y_ht_cv)*mnoznik,
         y_ht_var=y_ht_se^2, y_ht_cv2=y_ht_se/y_ht)
p <- ggplot(nsp_stopa_bezr_y, aes(x = y_ht_cv2)) +
  geom_histogram(binwidth = 0.01) +
  xlim(0,0.4) +
  ylim(0,65) +
  xlab("Zmodyfikowany wskaźnik precyzji stopy bezrobocia") +
  ylab("") +
  theme_light()
print(p)
```



Rys. 12.6. Rozkład zmodyfikowanego wskaźnika precyzji stopy bezrobocia

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Po zwiększeniu ocen błędu oszacowania wskaźnik precyzji przybiera wartości od 10 do 34%, co stanowi wskazanie do wykorzystania metod statystyki małych obszarów. Poprawa precyzji oszacowań będzie możliwa dzięki zastosowaniu modelu zawierającego zmienne pomocnicze

skorelowane z badaną cechą. Wybór takich cech może odbywać się z wykorzystaniem metod matematycznych, ale przede wszystkim powinien być poparty dogłębną analizą zjawiska i jego determinant. Istotnym kryterium jest także jakość danych. Chodzi o to, by zmienne nie były obarczone błędem losowym. Zapewnić to może m.in. wykorzystanie takich źródeł danych, jak spisy powszechne czy rejestry administracyjne.

W tym przykładzie wykorzystamy dwie cechy z Banku Danych Lokalnych:

x1 – ludność w wieku nieprodukcyjnym na 100 osób w wieku produkcyjnym,
x2 – zasięg korzystania ze środowiskowej pomocy społecznej.

Po pobraniu tych danych łączymy je z wcześniej przygotowanym zbiorem zawierającym oszacowania stopy bezrobocia (skrypt 12.3d).

Skrypt 12.3d

```
# zmienne pomocnicze
# ludność w wieku nieprodukcyjnym na 100 osób w wieku produkcyjnym
x1<-get_data_by_variable(varId=60563, unitLevel=5, year=2011) %>%
  dplyr::select(id, name, x1=val)
# zasięg korzystania ze środowiskowej pomocy społecznej
x2<-get_data_by_variable(varId=458700, unitLevel=5, year=2011) %>%
  dplyr::select(id, name, x2=val)
# połączenie zbiorów danych
nsp_stopa_bezr<-inner_join(nsp_stopa_bezr_y, inner_join(x1, x2))
nsp_zm <- nsp_stopa_bezr %>%
  dplyr::select(id, name, y_ht, y_ht_var, x1, x2)
print(head(nsp_zm))
```

W wyniku realizacji skryptu 12.3d stworzono zbiór `nsp_stopa_bezr`, który zawiera wszystkie dane wymagane do modelu Faya-Herriota, i wyświetlono w konsoli pierwszych 6 obserwacji.

```
# A tibble: 6 x 6
  id           name                y_ht y_ht_var    x1    x2
  <chr>        <chr>              <dbl>    <dbl> <dbl> <dbl>
1 011212001000 Powiat bocheński   0.092 0.000378  58.1   7.2
2 011212006000 Powiat krakowski   0.085 0.000237  56.2   4.2
3 011212008000 Powiat miechowski  0.095 0.000602  63     6.7
4 011212009000 Powiat myślenicki  0.1   0.000352  57.4   8
5 011212014000 Powiat proszowicki 0.057 0.000320  59.3   8.1
6 011212019000 Powiat wielicki    0.092 0.000339  55.8   5
```

W pierwszej kolejności sprawdźmy, jakie są parametry zwykłego modelu liniowego opisującego zależność stopy bezrobocia od wymienionych zmiennych niezależnych (skrypt 12.3e).

Skrypt 12.3e

```
model<-lm(y_ht~x1+x2, data=nsp_stopa_bezr)
print(summary(model))
```

W wyniku uruchomienia skryptu 12.3e otrzymujemy podsumowanie zawierające parametry modelu.

```
Call:
lm(formula = y_ht ~ x1 + x2, data = nsp_stopa_bezr)
Residuals:
      Min        1Q    Median        3Q       Max
-0.065559 -0.016211 -0.003288  0.014708  0.082813
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.2837999  0.0216251  13.124   <2e-16 ***
x1          -0.0038034  0.0003968  -9.585   <2e-16 ***
x2           0.0062036  0.0003732  16.623   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 0.0263 on 376 degrees of freedom
Multiple R-squared:  0.4496,    Adjusted R-squared:  0.4467
F-statistic: 153.6 on 2 and 376 DF,  p-value: < 2.2e-16
```

Obie cechy są w modelu statystycznie istotne. Wzrost liczby ludności w wieku nieprodukcyjnym wpływa na średni spadek stopy bezrobocia, natomiast wzrost zasięgu korzystania ze środowiskowej pomocy społecznej powoduje także wzrost stopy bezrobocia. Model liniowy wyjaśnia 45% zmienności stopy bezrobocia.

Zastosowanie modelu Faya-Herriota jest możliwe poprzez pakiet `sae` i funkcję `mseFH()`, która przyjmuje podobne argumenty co model liniowy, z tą różnicą, że trzeba także uwzględnić wartości wariancji oszacowań bezpośrednich. Ponadto bardzo ważna jest zgodność typów – zbiór ze zmiennymi pomocniczymi musi być macierzą. Z racji tego, że pakiet *sae* może spowodować konflikt z używaną przez nas wcześniej funkcją `select()`, nie będziemy importować wszystkich funkcji tego pakietu za pomocą funkcji `library()`, tylko poprzez zapis `nazwa_pakietu::funkcja()` bezpośrednio odwołamy się do wybranej funkcji (skrypt 12.3f).

Skrypt 12.3f

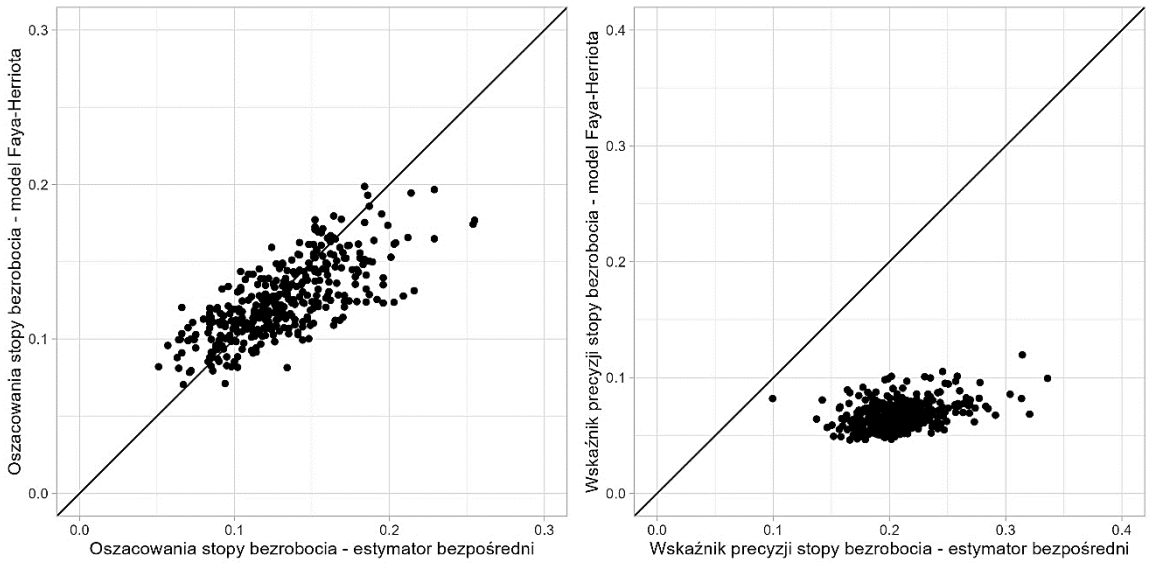
```
zmienne_x<-as.matrix(nsp_stopa_bezr[,c("x1", "x2")])
modelFH<-sae::mseFH(nsp_stopa_bezr$y_ht ~ zmienne_x, nsp_stopa_bezr$y_ht_var)
```

Obiekt wynikowy jest dwuelementową listą zawierającą oszacowania oraz parametry modelu. Korzystając z wybranych elementów tego obiektu, tworzymy nowe kolumny w zbiorze danych dla oszacowań, błędów oszacowania oraz wag γ. Na rysunku 12.7 porównano oszacowania bezpośrednie oraz pochodzące z modelu Faya-Herriota wraz ze wskaźnikami precyzji (skrypt 12.3g).

Skrypt 12.3g

```
nsp_stopa_bezr<-nsp_stopa_bezr %>%
  mutate(y_fh=as.numeric(modelFH$est$eblup),
    y_fh_mse=modelFH$mse, y_fh_se=sqrt(y_fh_mse), y_fh_cv=y_fh_se/y_fh,
    gamma=modelFH$est$fit$refvar/(modelFH$est$fit$refvar+y_ht_var))
p1<-ggplot(nsp_stopa_bezr, aes(y_ht, y_fh)) +
  geom_point() +
  geom_abline(slope=1) +
  xlim(0,0.3) +
  ylim(0,0.3) +
```

```
  xlab("Oszacowania stopy bezrobocia\n estymator bezpośredni") +
  ylab("Oszacowania stopy bezrobocia\n model Faya-Herriota") +
  theme_light()
p2<-ggplot(nsp_stopa_bezr, aes(y_ht_cv2, y_fh_cv)) +
  geom_point() +
  geom_abline(slope = 1) +
  xlim(0,0.4) +
  ylim(0,0.4) +
  xlab("Wskaźnik precyzji stopy bezrobocia\n estymator bezpośredni") +
  ylab("Wskaźnik precyzji stopy bezrobocia\n model Faya-Herriota") +
  theme_light()
print(p1 + p2)
```



Rys. 12.7. Oszacowania uzyskane na podstawie dwóch estymatorów: bezpośredniego i modelu Faya-Herriota wraz ze wskaźnikiem precyzji

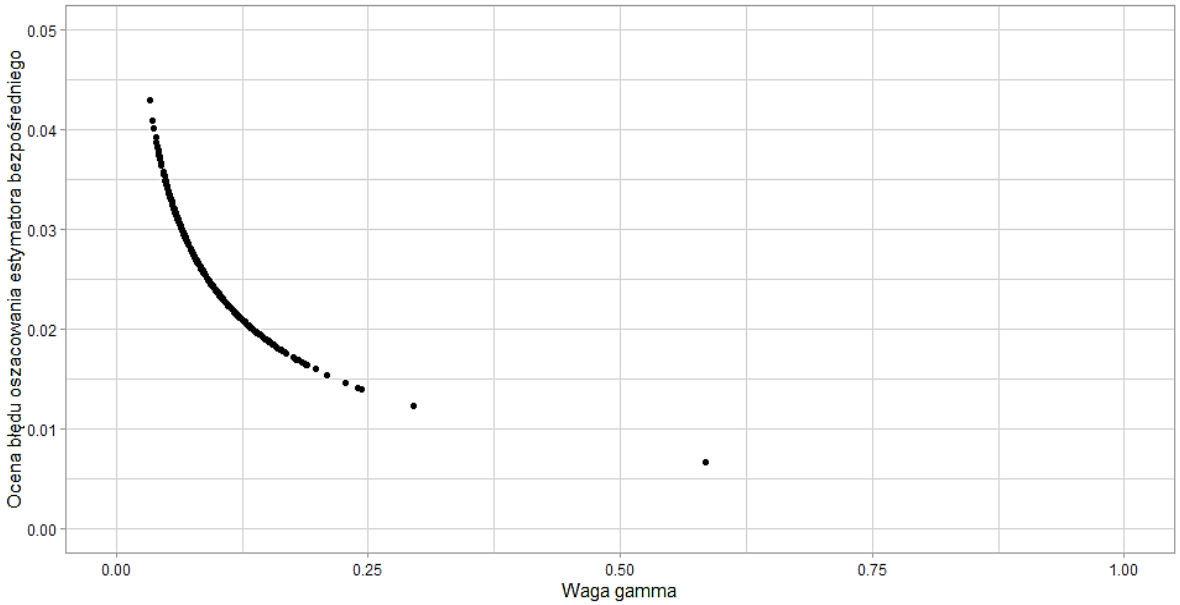
Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Wartości oszacowań są do siebie zbliżone, natomiast ich precyzja jest znacznie lepsza w przypadku zastosowania modelu Faya-Herriota. Dla modelu *FH* najwyższa wartość to niecałe 13%, co oznacza, że taki udział w wartości parametru stanowi średni błąd szacunku (dla estymacji bezpośredniej wynosi on 33%). Należy jednak pamiętać, że oceny wskaźników precyzji obliczone dla estymatora bezpośredniego oraz modelu Faya-Herriota nie oceniają precyzji w jednakowym wymiarze, co oznacza, że nie są porównywalne. W modelu *FH* uwzględnianych jest więcej źródeł zmienności ocen parametru. W celu właściwego zestawienia tych dwóch wielkości należałoby przeprowadzić dodatkowe badanie symulacyjne (Pfeffermann i Ben-Hur, 2019).

Na rys. 12.8 przedstawiono zależność pomiędzy wartościami wagi γ i błędu oszacowania estymatora bezpośredniego (skrypt 12.3h).

Skrypt 12.3h

```
P<-ggplot(nsp_stopa_bezr, aes(x=gamma, y=y_ht_se)) +
  geom_point() +
  xlim(0,1) +
  ylim(0,0.05) +
  xlab("Waga gamma") +
  ylab("Błąd oszacowania oszacowań bezpośrednich") +
  theme_light()
print(p)
```



Rys. 12.8. Waga γ i ocena błędu oszacowania estymatora bezpośredniego

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

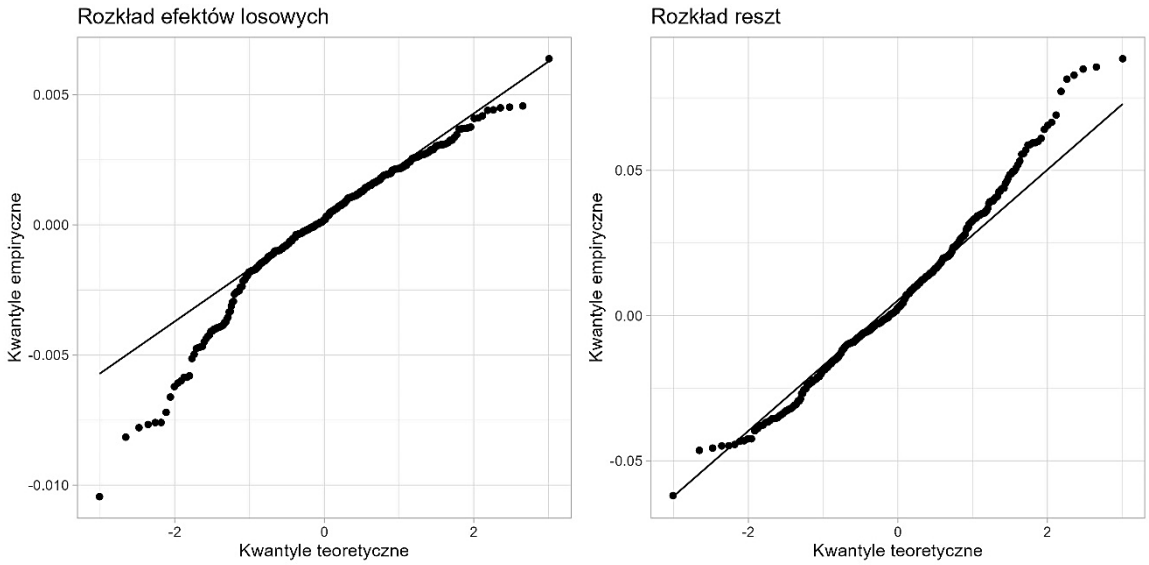
Rysunek 12.8 przedstawia zależność pomiędzy wartością wagi γ i wielkością błędu szacunku ilustrującą ideę działania modelu Faya-Herriota. Tam, gdzie błąd szacunku jest mały, waga γ przybiera wartość bliską jedności, czyli udział oszacowania bezpośredniego w ostatecznym oszacowaniu jest duży. W odwrotnej sytuacji większą część wartości estymatora stanowi oszacowanie pochodzące z zastosowania modelu.

Stosując podejście modelowe, należy pamiętać o sprawdzeniu, czy są spełnione założenia dotyczące normalności reszt oraz efektu losowego. Aby tego dokonać, musimy wyznaczyć wartości oszacowań modelowych i na ich podstawie wyznaczyć reszty oraz wartości efektów losowych (skrypt 12.3i). Normalność można zweryfikować m.in. z wykorzystaniem wykresów kwantyl-kwantyl (rys. 12.9).

Skrypt 12.3i

```
reg<-as.numeric(model.matrix(y_ht~x1+x2, data=nsp_stopa_bezr) %*%
  modelFH$est$fit$estcoef$beta)
nsp_stopa_bezr <- nsp_stopa_bezr %>%
  mutate(model_reg=reg, reszty=y_ht-model_reg, efekt=gamma*reszty)
```

```
p1 <- ggplot(nsp_stopa_bezr, aes(sample=efekt)) +
  stat_qq() +
  stat_qq_line() +
  theme_light() +
  ggtitle("Rozkład efektów losowych")
p2<-ggplot(nsp_stopa_bezr, aes(sample=reszty)) +
  stat_qq() +
  stat_qq_line() +
  theme_light() +
  ggtitle("Rozkład reszt")
print(p1+p2)
```



Rys. 12.9. Wykresy kwantyl-kwantyl rozkładu efektów losowych oraz reszt

Źródło: opracowanie własne z wykorzystaniem programu R (R Core Team, 2023).

Założenie o normalności efektów losowych i reszt jest bardzo trudne do spełnienia. Na wykresach możemy zauważyć pewne odstępstwa od rozkładu normalnego, szczególnie w ogonach rozkładu. Ten fakt może mieć negatywny wpływ na oceny błędów szacunku i tym samym pogorszenie precyzji oszacowań. Jeśli w prowadzonym badaniu analiza ujawni znaczne odstępstwa od założeń dotyczących normalności reszt oraz efektu losowego, można podjąć próbę zastosowania do estymacji parametrów np. metod odpornych uwzględniających wartości odstające (Sinha i Rao, 2009).

Literatura

Beręsewicz, M., Marchetti, S., Salvati, N., Szymkowiak, M. i Wawrowski, Ł. (2018). The Use of a Three Level M-quantile Model to Map Poverty at LAU 1 in Poland. *Journal of the Royal Statistical Society Series A: Statistics in Society*, *181*(4), 1077-1104. https://doi.org/10.1111/rssa.12349

Bracha, C. (1996). *Teoretyczne podstawy metody reprezentacyjnej*. Wydawnictwo Naukowe PWN.

Cassel, C. M., Sarndal, C. E. i Wretman, J. H. (1977). *Foundations of Inference in Survey Sampling*. John Wiley & Sons.

Chandra, H., Chambers, R. i Salvati, N. (2012). Small Area Estimation of Proportions in Business Surveys. *Journal of Statistical Computation and Simulation*, *82*(6), 783-795. **https://doi.org/10.1080/00949655.2011.554834**

Dehnel, G. (2010). *Rozwój mikroprzedsiębiorczości w Polsce w świetle estymacji dla małych domen*. Wydawnictwo Uniwersytetu Ekonomicznego w Poznaniu.

Dehnel, G. (2014). Winsorization Methods in Polish Business Survey. *Statistics in Transition New Series*, *15*(1), 97-110. **https://doi.org/10.59170/stattrans-2014-007**

Dehnel, G. i Wawrowski, Ł. (2020). Robust Estimation of Wages in Small Enterprises: The Application to Poland's Districts. *Statistics in Transition New Series*, *21*(1), 137-157. **https://doi.org/10.21307/stattrans-2020-008**

Elazar, D. i Conn, L. (2004). Small Area Estimation of Disability in Australia. *Statistics in Transition*, *6*(5), 667-684.

Esteban, M. D., Morales, D. i Perez, A. (2014). *saery: Small Area Estimation for Rao and Yu Model.* R package version 1.0. **https://github.com/cran/saery**

Fay, R. i Diallo, M. (2023). *sae2: Small Area Estimation: Time-Series Models*. R package version 1.2-1. **https://cran.r-project.org/package=sae2**

Fay, R. E. i Herriot, R. A. (1979). Estimation of income from small places: An application of James-Stein procedures to census data. *Journal of the American Statistical Association*, *74*, 269-277.

Gołata, E. (2004). *Estymacja pośrednia bezrobocia na lokalnym rynku pracy*. Wydawnictwo Uniwersytetu Ekonomicznego w Poznaniu.

Główny Urząd Statystyczny [GUS]. (2013). *Ludność. Stan i struktura demograficzno-społeczna*. Zakład Wydawnictw Statystycznych.

Horvitz, D. G. i Thompson, D. J. (1952). A Generalization of Sampling Without Replacement from a Finite Universe. *Journal of the American Statistical Association*, *47*(260), 663-685.

Klimanek, T., Szymkowiak, M. i Józefowski, T. (2018). Badanie zjawiska niepełnosprawności w przekroju powiatów województwa wielkopolskiego z wykorzystaniem metod statystyki małych obszarów. *Przegląd Statystyczny*, *65*(4), 449-472. **https://doi.org/10.5604/01.3001.0014.0604**

Kreutzmann, A., Pannier, S., Rojas-Perilla, N., Schmid, T., Templ, M. i Tzavidis, N. (2019). The R Package emdi for Estimating and Mapping Regionally Disaggregated Indicators. *Journal of Statistical Software*, *91*(7), 1-33. **https://doi.org/10.18637/jss.v091.i07**

Lopez-Vizcaino, E., Lombardia, M. J. i Morales, D. (2013). Multinomial-based Small Area Estimation of Labour Force Indicators. *Statistical Modelling*, *13*(2), 153-178. **https://doi.org/10.1177/1471082X13478873**

Lopez-Vizcaino, E., Lombardia, M. J. i Morales, D. (2019). *mme: Multinomial Mixed Effects Models*. R package version 0.1-6.

Marhuenda, Y., Molina, I. i Morales, D. (2013). Small Area Estimation with Spatio-Temporal Fay-Herriot Models. *Computational Statistics and Data Analysis*, *58*, 308-325. **https://doi.org/10.1016/j.csda.2012.09.002**

Molina, I. i Marhuenda, Y. (2015). sae: An R Package for Small Area Estimation. *The R Journal*, *7*(1), 81-98. **https://doi.org/10.32614/RJ-2015-007**

Molina, I. i Rao, J. N. K. (2010). Small Area Estimation of Poverty Indicators. *The Canadian Journal of Statistics*, *38*, 369-385.

Molina, I. i Rao, J. N. K. (2015). *Small Area Estimation*. Wiley.

Molina, I., Saei, A. i Lombardía, M. J. (2007). Small Area Estimates of Labour Force Participation Under a Multinomial Logit Mixed Model. *Journal of the Royal Statistical Society Series A: Statistics in Society*, *170*(4), 975-1000. **https://doi.org/10.1111/j.1467-985X.2007.00493.x**

Pfeffermann, D. i Ben-Hur, D. (2019). Estimation of Randomisation Mean Square Error in Small Area Estimation. *International Statistical Review*, *87*(51), 31-49. **https://doi.org/10.1111/insr.12289**

Pratesi, M. i Salvati, N. (2008). Small Area Estimation: The EBLUP Estimator Based on Spatially Correlated Random Area Effects. *Statistical Methods & Applications*, *17*(1), 113-141. **https://doi.org/10.1007/s10260-007-0061-9**

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Rao, J. N. K. i Yu, M. (1994). Small Area Estimation by Combining Time Series and Cross-sectional Data. *Canadian Journal of Statistics*, *22*(4), 511-528. **https://doi.org/10.2307/3315407**

Sinha, S. K. i Rao, J. N. K. (2009). Robust Small Area Estimation. *Canadian Journal of Statistics*, *37*(3), 381-399. **https://doi.org/10.1002/cjs.10029**

Warnholz, S. (2023). *saeRobust: Robust Small Area Estimation*. R package version 0.4.0. **https://cran.r-project.org/package=saeRobust**

Wawrowski, Ł. (2014). Wykorzystanie metod statystyki małych obszarów do tworzenia map ubóstwa w Polsce. *Wiadomości Statystyczne. The Polish Statistician*, *59*(09), 46-56. **https://doi.org/10.59139/ws.2014.09.3**

Wawrowski, Ł. (2016). The Spatial Fay-Herriot Model in Poverty Estimation, *Folia Oeconomica Stetinensia*, *16*(2), 191-202. **https://doi.org/10.1515/foli-2016-0034**

Wawrowski, Ł. (2020). Estymacja pośrednia wskaźników ubóstwa na poziomie powiatów. *Wiadomości Statystyczne. The Polish Statistician*, *65*(08), 7-26. **https://doi.org/10.5604/01.3001.0014.3524**

Wilak, K. (2014). Strukturalne modele szeregów czasowych w estymacji stopy bezrobocia w dezagregacji na województwa, płeć i wiek. *Przegląd Statystyczny*, *61*(4), 409-431.

Zasępa, R. (1972). *Metoda reprezentacyjna*. Państwowe Wydawnictwo Ekonomiczne.

Żądło, T. (2008). *Elementy statystyki małych obszarów z programem R*. Wydawnictwo Akademii Ekonomicznej im. Karola Adamieckiego.

Żądło, T. (2015). Statystyka małych obszarów w badaniach ekonomicznych: podejście modelowe i mieszane. *Prace Naukowe/Uniwersytet Ekonomiczny w Katowicach*.

Aneks
Podstawy pakietu ggplot2

Pakiet `ggplot2` to jeden z najpopularniejszych wykorzystywanych do wizualizacji danych w R, będący częścią pakietu `tidyverse`. Umożliwia tworzenie eleganckich i wysoce konfigurowalnych wykresów, stosując zasadę tzw. gramatyki grafiki (*Grammar of Graphics*). Aneks omawia podstawowe elementy składni `ggplot2` i prezentuje, jak krok po kroku tworzyć wybrane wykresy. Szczegółowe informacje dotyczące pakietu `ggplot2` można znaleźć w książce autora tego pakietu (Wickham, 2016) dostępnej *online* w wersji angielskiej. Niezwykle istotny jest także właściwy dobór wykresu do danych. Zestaw dobrych praktyk w tym zakresie jest zawarty w pozycji (Wilke, 2019).

Podstawowa składnia

Podstawowym elementem w `ggplot2` jest funkcja `ggplot()`, która tworzy obiekt wykresu. Budowanie wykresu z `ggplot2` polega na dodawaniu warstw (*layers*) przy użyciu operatora `+`. Najważniejsze elementy składni `ggplot2` to:

- dane (`data`): zbiór danych, na którym będziemy pracować,
- mapowanie (`mappings`): określają, jak zmienne ze zbioru danych są przypisywane do estetyk (`aes, aesthetics`), takich jak osie *x* oraz *y*, kolory, rozmiary itp.,
- geometrie (`geoms`): definiują rodzaj wykresu, np. punkty, linie, słupki.

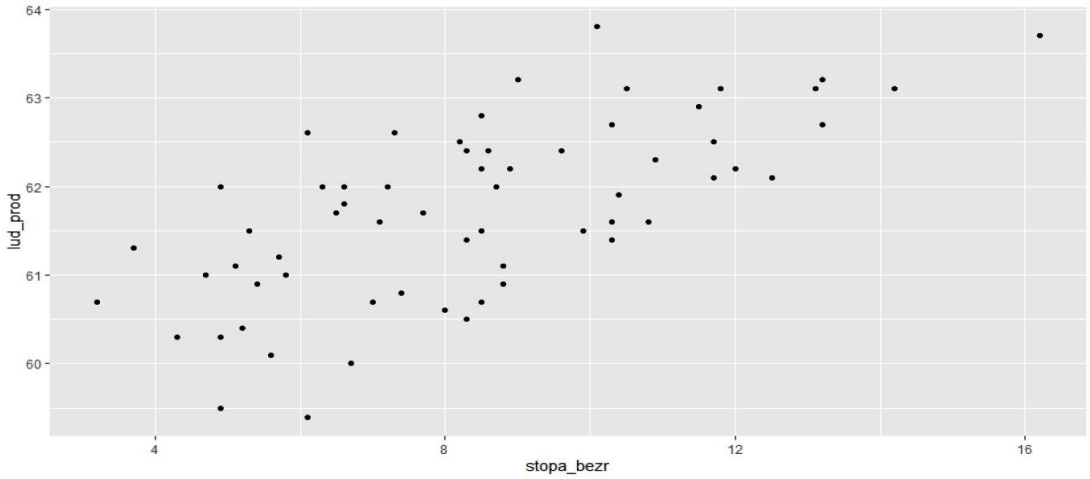
W pierwszym kroku z Banku Danych Lokalnych zostaną wczytane dane dotyczące stopy bezrobocia oraz liczby osób w wieku produkcyjnym na poziomie województw w latach 2015-2018.

```
library(bdl)
library(tidyverse)
stopa_bezr <- get_data_by_variable(varId=60270, unitLevel=2, year=2015:2018) %>%
  select(id, name, year, stopa_bezr=val)
lud_prod <- get_data_by_variable(varId=60566, unitLevel=2, year=2015:2018) %>%
  select(id, name, year, lud_prod=val)
dane <- inner_join(stopa_bezr, lud_prod, by=join_by(id, name, year))
```

Wykres punktowy/rozrzutu

Aby stworzyć prosty wykres punktowy (*scatter plot*), należy użyć funkcji `ggplot()` w połączeniu z geometrią `geom_point()`.

```
ggplot(data=dane, mapping=aes(x=stopa_bezr, y=lud_prod)) +
  geom_point()
```



Rys. A1. Wykres punktowy stopy bezrobocia i liczby osób w wieku produkcyjnym

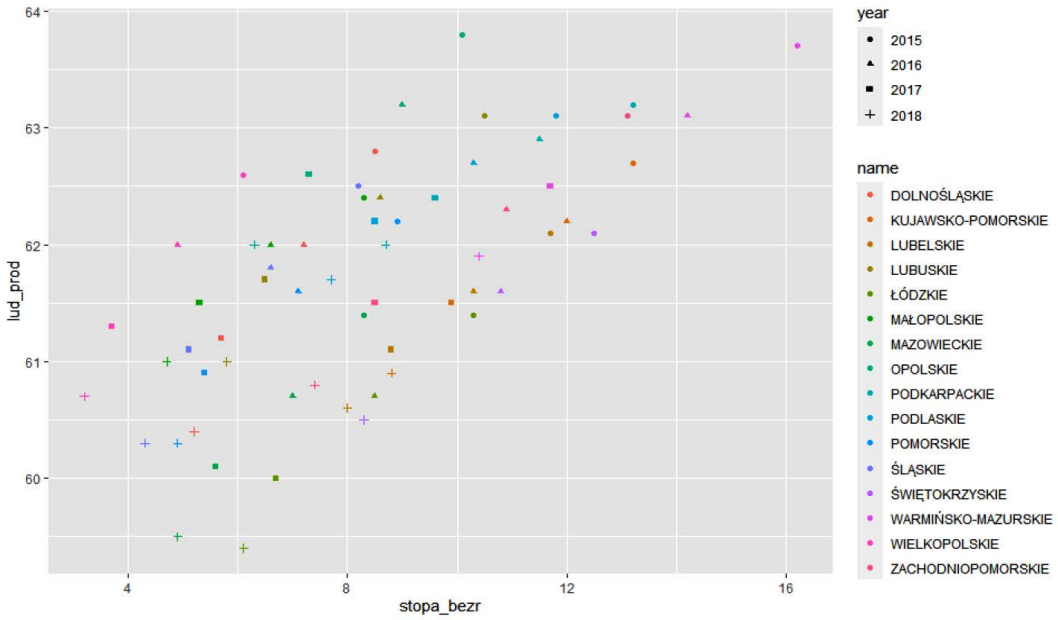
Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

W tym przykładzie:

- `data = dane` określa zbiór danych,
- `aes(x=stopa_bezr, y=lud_prod)` definiuje mapowanie, gdzie kolumna `stopa_bezr` jest umieszczona na osi *x*, a kolumna `lud_prod` jest na osi *y*,
- `geom_point()` dodaje warstwę geometryczną, która rysuje punkty.

Wykres można rozszerzyć wykres, dodając więcej estetyk, takich jak kolor (`color`) i kształt (`shape`):

```
ggplot(data=dane, mapping=aes(x=stopa_bezr, y=lud_prod, color=name, shape=year)) +
  geom_point()
```

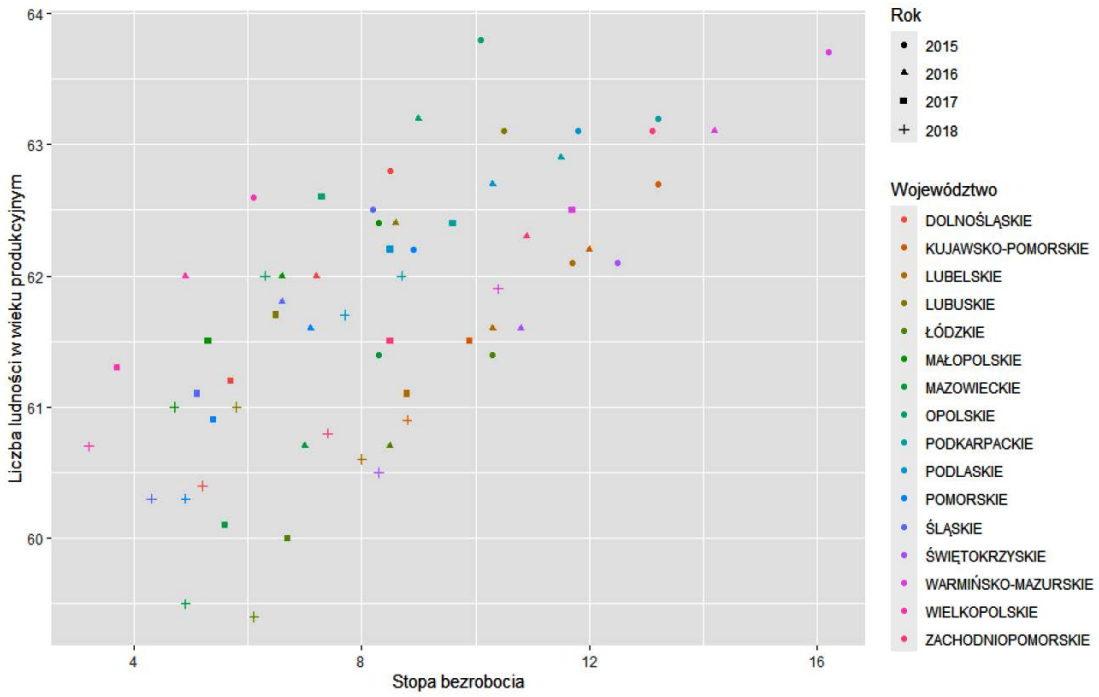


Rys. A2. Wykres punktowy stopy bezrobocia i liczby osób w wieku produkcyjnym z uwzględnieniem roku i województwa

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

W kolejnym kroku zmodyfikujemy etykiety osi oraz tytuł przy legendzie. W tym celu korzystamy z funkcji `xlab()` oraz `ylab()` dla osi *x* oraz *y*, natomiast dla pozostałych estetyk konieczne jest zastosowanie funkcji z przedrostkiem `scale_`, po którym podajemy nazwę estetyki (`color/shape`) oraz typ cechy (`discrete/continuous`).

```
ggplot(data=dane, mapping=aes(x=stopa_bezr, y=lud_prod, color=name, shape=year)) +
  geom_point() +
  xlab("Stopa bezrobocia") +
  ylab("Liczba ludności w wieku produkcyjnym") +
  scale_color_discrete(name = "Województwo") +
  scale_shape_discrete(name = "Rok")
```

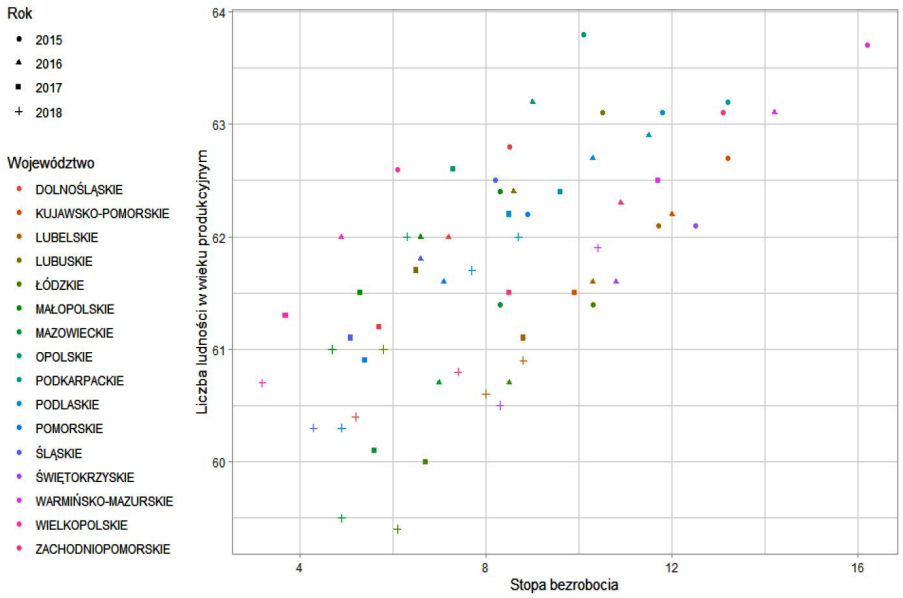


Rys. A3. Wykres punktowy stopy bezrobocia i liczby osób w wieku produkcyjnym z uwzględnieniem roku i województwa oraz z właściwymi etykietami

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Pakiet `ggplot2` pozwala na dostosowywanie wyglądu wykresów za pomocą funkcji gotowych motywów, takich jak `theme_bw()`, `theme_minimal()`, `theme_classic()` itp. Dodatkowo za pomocą funkcji `theme()` można edytować dowolny element wykresu m.in. pozycję legendy. Co istotne, funkcji `theme()` należy zawsze używać po zastosowaniu gotowego motywu, ponieważ w przeciwnym przypadku zastosowane przez nas modyfikacje wyglądu wykresu zostaną zresetowane.

```
ggplot(data=dane, mapping=
aes(x=stopa_bezr, y=lud_prod, color=name, shape=year)) +
  geom_point() +
  xlab("Stopa bezrobocia") +
  ylab("Liczba ludności w wieku produkcyjnym") +
  scale_color_discrete(name = "Województwo") +
  scale_shape_discrete(name = "Rok") +
  theme_light() +
  theme(legend.position = "left")
```

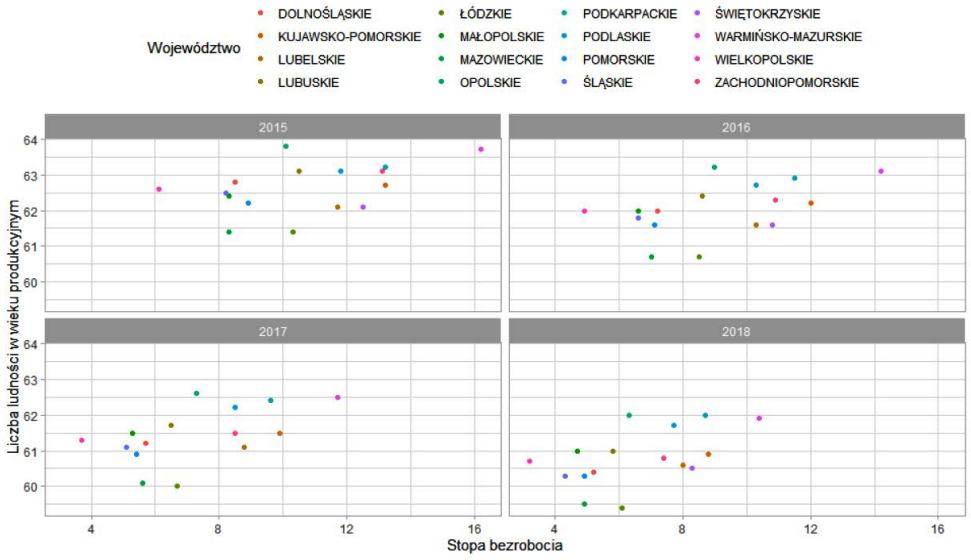


Rys. A4. Wykres punktowy stopy bezrobocia i liczby osób w wieku produkcyjnym z uwzględnieniem roku i województwa oraz z właściwymi etykietami i motywem

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Facetowanie umożliwia tworzenie wielu wykresów na podstawie podgrup danych. Można to zrobić za pomocą `facet_wrap()` lub `facet_grid()`.

```
ggplot(data=dane, mapping=aes(x=stopa_bezr, y=lud_prod, color=name)) +
  geom_point() +
  xlab("Stopa bezrobocia") +
  ylab("Liczba ludności w wieku produkcyjnym") +
  scale_color_discrete(name = "Województwo") +
  facet_wrap(~ year) +
  theme_light() +
  theme(legend.position = "top")
```



Rys. A5. Wykres punktowy stopy bezrobocia i liczby osób w wieku produkcyjnym z uwzględnieniem województwa i roku na osobnych wykresach

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

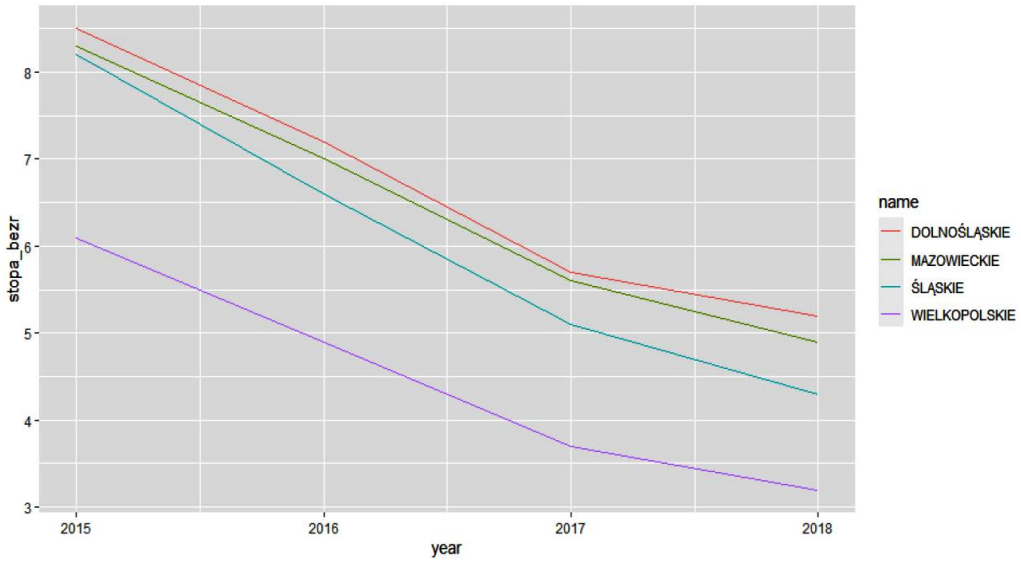
W powyższym przykładzie `facet_wrap(~ year)` tworzy osobne wykresy dla każdej wartości roku. Jest to szczególnie przydatne w sytuacji, kiedy zastosowanie zbyt wielu estetyk na jednym wykresie pozbawia go czytelności.

Na przykładzie wykresu rozrzutu został przedstawiony proces tworzenia prostego wykresu i dodawania do niego kolejnych elementów modyfikujących jego wygląd. W dalszej części tego aneksu pozostałe typy wykresów zostaną jedynie krótko scharakteryzowane, czytelnika zaś zachęcamy do modyfikacji tych przykładów według własnych upodobań.

Wykres liniowy

Wykres liniowy jest przydatny do wizualizacji zmian wartości w czasie lub w innej ciągłej zmiennej. W pakiecie `ggplot2` do tworzenia wykresów liniowych używamy geometrii `geom_line()`. Utworzenie właściwego wykresu liniowego wymaga, aby cecha na osi *x* miała format liczbowy, zatem w odniesieniu do wykorzystywanego zbioru danych należy przeprowadzić odpowiednie rzutowanie. Dodatkowo spośród 16 województw wybieramy 4, aby wykres był bardziej czytelny.

```
dane %>%
  mutate(year=as.numeric(year)) %>%
  filter(name %in% c("DOLNOŚLĄSKIE","MAZOWIECKIE","ŚLĄSKIE",
    "WIELKOPOLSKIE")) %>%
  ggplot(data = ., aes(x=year, y=stopa_bezr, color=name)) +
  geom_line()
```



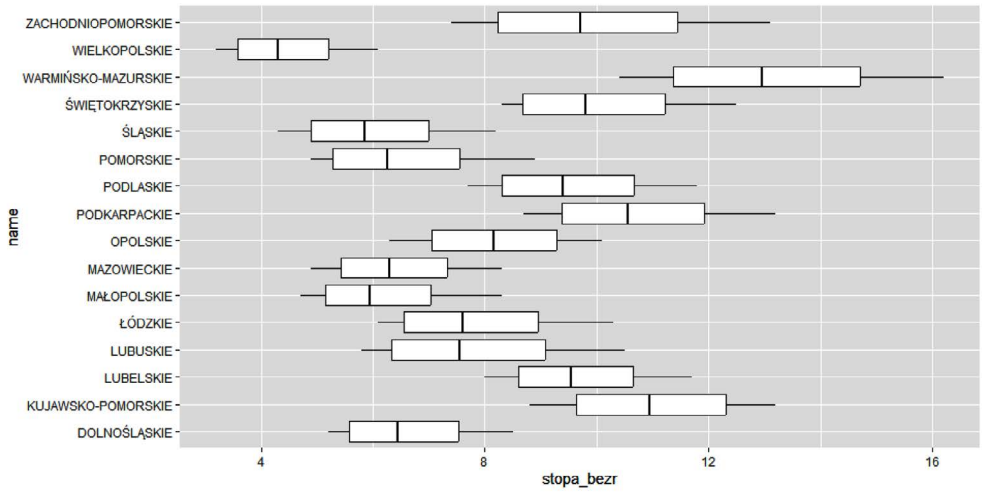
Rys. A6. Stopa bezrobocia w latach 2015-2018 dla wybranych województw

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Wykres pudełkowy

Wykresy pudełkowe (*boxplots*) są użyteczne do wizualizacji rozkładu danych i identyfikacji wartości odstających. Można je tworzyć za pomocą `geom_boxplot()`.

```
ggplot(data = dane, aes(x=stopa_bezr, y=name)) +
  geom_boxplot()
```



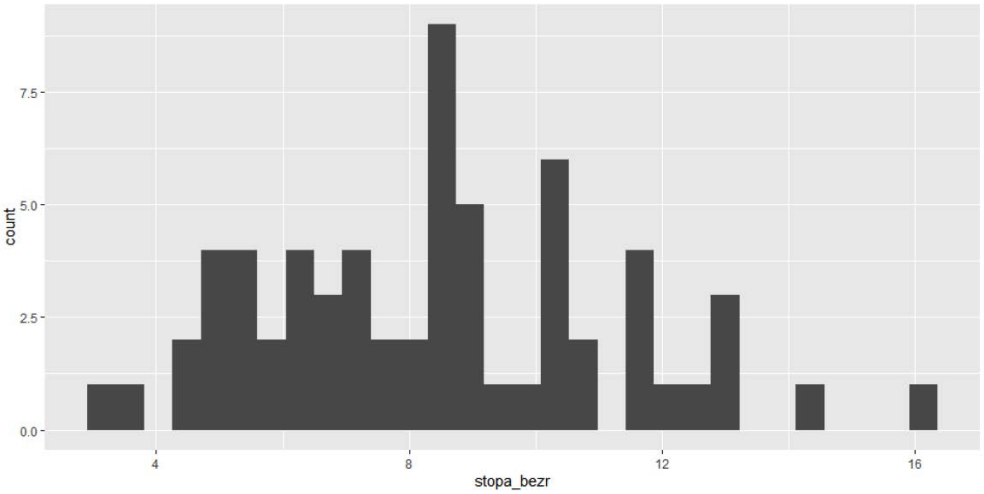
Rys. A7. Rozkład stopy bezrobocia w latach 2015-2018 w ramach województw

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Histogram

Histogram jest wykorzystywany do wizualizacji rozkładu jednej zmiennej ilościowej, przedstawiając częstotliwość występowania wartości w określonych przedziałach (*bins*). Do tworzenia histogramów w `ggplot2` używamy geometrii `geom_histogram()`.

```
ggplot(data = dane, aes(x=stopa_bezr)) +
  geom_histogram()
```

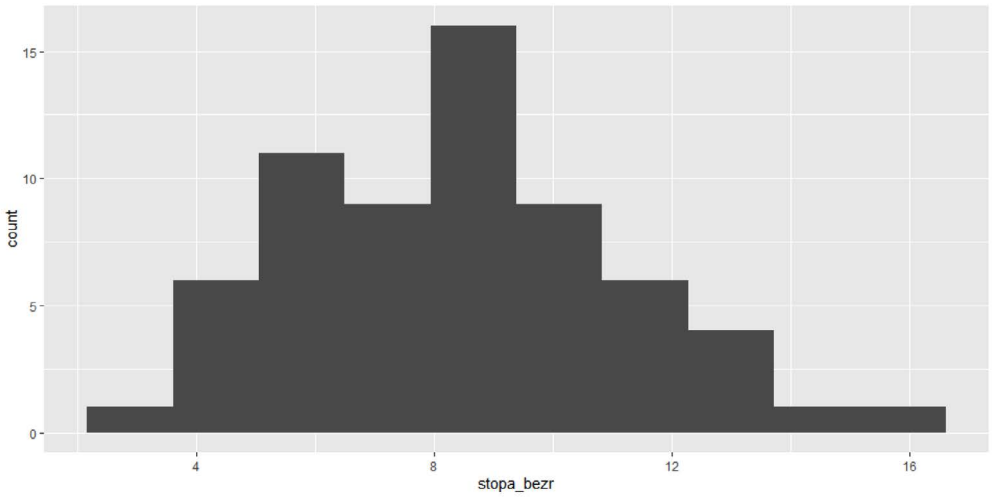


Rys. A8. Rozkład stopy bezrobocia – ustawienia domyślne

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Domyślnie funkcja `geom_histogram()` niezależnie od danych tworzy zawsze 15 słupków. Możemy zmienić tę wartość za pomocą argumentu `bins`.

```
ggplot(data = dane, aes(x=stopa_bezr)) +
  geom_histogram(bins = 10)
```

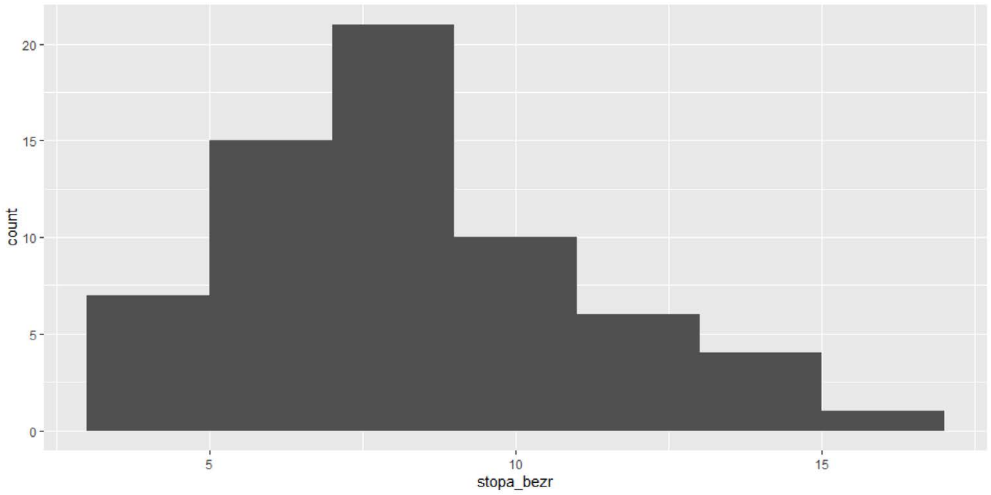


Rys. A9. Rozkład stopy bezrobocia – 10 słupków

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Alternatywnie, zamiast podawać liczbę słupków, można także określić ich szerokość z wykorzystaniem argumentu `binwidth`.

```
ggplot(data = dane, aes(x=stopa_bezr)) +
  geom_histogram(binwidth = 2)
```



Rys. A10. Rozkład stopy bezrobocia – słupki o szerokości 2 punktów procentowych

Źródło: opracowanie własne w wykorzystaniem programu R (R Core Team, 2023).

Podsumowanie

Pakiet `ggplot2` oferuje duże możliwości dostosowywania i tworzenia wykresów. Podstawy obejmują zrozumienie składni `ggplot2`, czyli danych, mapowań, geometrii i warstw. Dzięki temu można tworzyć zarówno proste, jak i zaawansowane wizualizacje, dostosowane do indywidualnych potrzeb analitycznych.

Literatura

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. **https://www.R-project.org**

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer. **https://ggplot2-book.org/**

Wilke, C. O. (2019). *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. O'Reilly Media. **https://clauswilke.com/dataviz/**

Spis rysunków

Spis tabel