

O'REILLY®

Zaawansowana analiza danych

Jak przejść z arkuszy Excela
do Pythona i R



Helion 

George Mount

Tytuł oryginału: Advancing into Analytics: From Excel to Python and R

Tłumaczenie: Filip Kamiński

ISBN: 978-83-283-8551-1

© 2022 Helion S.A.

Authorized Polish translation of the English edition Advancing into Analytics ISBN 9781492094340 © 2021
Candid World Consulting, LLC.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/zaanda>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wprowadzenie	9
---------------------------	----------

Część I. Podstawy analizy danych w Excelu	17
--	-----------

1. Podstawy eksploracyjnej analizy danych	19
--	-----------

Czym jest eksploracyjna analiza danych?	19
---	----

Obserwacje	21
------------	----

Zmienne	21
---------	----

Przykład: klasyfikacja zmiennych	24
----------------------------------	----

Przypomnienie: typy zmiennych	26
-------------------------------	----

Eksploracja zmiennych w Excelu	26
--------------------------------	----

Eksploracja zmiennych kategoryjnych	27
-------------------------------------	----

Eksploracja zmiennych ilościowych	29
-----------------------------------	----

Wnioski	40
---------	----

Ćwiczenia	40
-----------	----

2. Podstawy prawdopodobieństwa	41
---	-----------

Prawdopodobieństwo i losowość	41
-------------------------------	----

Prawdopodobieństwo i przestrzeń zdarzeń elementarnych	41
---	----

Prawdopodobieństwo i eksperymenty	42
-----------------------------------	----

Prawdopodobieństwo bezwarunkowe i warunkowe	42
---	----

Rozkłady prawdopodobieństwa	42
-----------------------------	----

Dyskretne rozkłady prawdopodobieństwa	43
---------------------------------------	----

Ciągłe rozkłady prawdopodobieństwa	46
------------------------------------	----

Wnioski	53
---------	----

Ćwiczenia	53
-----------	----

3. Podstawy wnioskowania statystycznego	54
Ramy wnioskowania statystycznego	54
Zbierz reprezentatywną próbkę	55
Sformułuj hipotezy	56
Stwórz plan analizy	57
Przeanalizuj dane	59
Podejmij decyzję	62
To Twój świat... Dane się tylko w nim znajdują	68
Wnioski	69
Ćwiczenia	70
4. Korelacja i regresja	71
Korelacja nie oznacza przyczynowości	71
Wprowadzenie do korelacji	72
Od korelacji do regresji	76
Regresja liniowa w Excelu	78
Zastanówmy się raz jeszcze — pozorne związki	84
Wnioski	85
Przejdź do programowania	85
Ćwiczenia	85
5. Stos analizy danych	87
Statystyka, analiza danych, nauka o danych	87
Statystyka	87
Analiza danych	87
Analityka biznesowa	88
Nauka o danych	88
Uczenie maszynowe	88
Różne, ale nie rozłączne	89
Znaczenie stosu analizy danych	89
Arkusze kalkulacyjne	90
Bazy danych	92
Platformy analityki biznesowej	94
Języki programowania danych	94
Wnioski	95
Co dalej	96
Ćwiczenia	96

6. Pierwsze kroki w R dla użytkowników Excela	99
Pobieranie R	99
Pierwsze kroki w RStudio	99
Pakiety w R	108
Aktualizacja pakietów, RStudio i języka R	109
Wnioski	110
Ćwiczenia	110
7. Struktury danych w R	112
Wektory	112
Indeksowanie i wybór elementów z wektorów	114
Od tabel Excela do ramek danych R	115
Importowanie danych w R	117
Eksploracja ramki danych	120
Indeksowanie i wybór elementów z ramek danych	122
Zapisywanie ramek danych	123
Wnioski	124
Ćwiczenia	124
8. Przetwarzanie i wizualizacja danych w R	125
Przetwarzanie danych za pomocą dplyr	126
Operacje kolumnowe	126
Operacje wierszowe	128
Agregacja i łączenie danych	131
dplyr i potęga operatora potoku (%>%)	133
Przekształcanie danych za pomocą tidyr	135
Wizualizacja danych w ggplot2	137
Wnioski	142
Ćwiczenia	142
9. R w analizie danych	143
Eksploracyjna analiza danych	144
Testowanie hipotez	147
Test t-Studenta dla prób niezależnych	148
Regresja liniowa	150
Podział na zbiór uczący i testowy, walidacja	151
Wnioski	154
Ćwiczenia	154

Część III. Z Excela do Pythona	155
10. Pierwsze kroki w Pythonie dla użytkowników Excela	157
Pobieranie Pythona	157
Pierwsze kroki z Jupyterem	158
Moduły w Pythonie	166
Aktualizacja pakietów, Anacondy i Pythona	167
Wnioski	167
Ćwiczenia	168
11. Struktury danych w Pythonie	169
Tablice NumPy	170
Indeksowanie i wybieranie elementów z tablic NumPy	171
Ramki danych pandas	172
Importowanie danych w Pythonie	174
Eksploracja ramki danych	175
Indeksowanie i pobieranie wartości z ramek danych	177
Zapis ramek danych	178
Wnioski	178
Ćwiczenia	178
12. Przetwarzanie i wizualizacja danych w Pythonie	179
Operacje kolumnowe	180
Operacje wierszowe	182
Agregacja i łączenie danych	183
Przekształcanie danych	185
Wizualizacja danych	186
Wnioski	192
Ćwiczenia	192
13. Python w analizie danych	193
Eksploracyjna analiza danych	194
Testowanie hipotez	196
Test t-Studenta dla prób niezależnych	196
Regresja liniowa	197
Podział zbioru na zbiór treningowy i testowy oraz walidacja modelu	198
Wnioski	200
Ćwiczenia	200

14. Wnioski i kolejne kroki	201
Kolejne warstwy stosu	201
Projektowanie badań i eksperymenty biznesowe	201
Inne metody statystyczne	202
Nauka o danych i uczenie maszynowe	202
Kontrola wersji	202
Etyka	203
Idź naprzód i ciesz się danymi	203
Na pożegnanie	203
Skorowidz	204

Przetwarzanie i wizualizacja danych w R

Amerykański statystyk Ronald Thisted zażartował kiedyś, że „surowe dane tak jak surowe ziemniaki zazwyczaj wymagają oczyszczenia przed użyciem”. Przygotowanie danych wymaga czasu. Wiesz, co mam na myśli, jeśli kiedykolwiek wykonałeś następujące czynności:

- wybieranie, usuwanie lub tworzenie kolumn obliczeniowych,
- sortowanie lub filtracja wierszy,
- grupowanie i podsumowywanie wartości w grupach,
- łączenie wartości z różnych zbiorów za pomocą wspólnych pól.

Istnieją szanse, że wszystkie te czynności wykonywałeś w Excelu... *wiele razy*. Prawdopodobnie zgłębiałeś w tym celu tajniki sławnych funkcji, takich jak WYSZUKAJ.PIONOWO(), i tabel przestawnych. W tym rozdziale poznasz odpowiedniki tych narzędzi w R, zwłaszcza z pakietu *dplyr*.

Z przetwarzaniem danych często łączy się ich wizualizacja. Jak już wspomniałem, ludzie są wyjątkowo dobrzy we wzrokowym przetwarzaniu informacji. Jest to więc świetny sposób na podsumowanie danych. W tym rozdziale dowiesz się, jak wizualizować dane za pomocą wspaniałego pakietu *ggplot2*, który podobnie jak *dplyr* jest częścią kolekcji *tidyverse*. Da Ci to solidne podstawy do eksploracji i testowania relacji w języku R, które omówię w rozdziale 9. Zacznij od zaimportowania odpowiednich pakietów. W tym rozdziale skorzystamy z zestawu danych *star*, który znajdziesz w zbiorze materiałów dołączonych do tej książki (<https://ftp.helion.pl/przyklady/zaanda.zip>). Po załadowaniu bibliotek od razu wczytamy dane:

```
library(tidyverse)
library(readxl)

star <- read_excel("dane/star/star.xlsx")
head(star)
#> # A tibble: 6 x 8
#>   wynik.mat... wynik.czyt... rodzaj.klasy doświadcz... płeć darmowe... rasa id.szkoły
#>   <dbl>         <dbl> <chr>         <dbl> <chr> <chr> <chr> <dbl>
#> 1         473         447 mała.klasa         7 dzie... nie   biała 63
#> 2         536         450 mała.klasa        21 dzie... nie   czar... 20
#> 3         463         439 standardowa...     0 chłō... tak   czar... 19
#> 4         559         448 standardowa...    16 chłō... nie   biała 69
#> 5         489         447 mała.klasa         5 chłō... tak   biała 79
#> 6         454         431 standardowa...     8 chłō... tak   biała 5
```

Przetwarzanie danych za pomocą dplyr

dplyr to popularny pakiet stworzony do przetwarzania tabelarycznych struktur danych. Wiele dostępnych w nim funkcji działa w podobny sposób, a ich wywołania można ze sobą z łatwością łączyć. Tabela 8.1 przedstawia niektóre popularne funkcje z *dplyr* i ich zastosowania. W tym rozdziale omówię każdą z nich.

Tabela 8.1. Popularne funkcje z pakietu *dplyr*

Nazwa funkcji	Opis
<code>select()</code>	Wybór określonych kolumn
<code>mutate()</code>	Tworzenie nowych kolumn na podstawie istniejących
<code>rename()</code>	Zmiana nazw kolumn
<code>arrange()</code>	Zmiana kolejności wierszy na podstawie określonych kryteriów
<code>filter()</code>	Wybór wierszy na podstawie określonych kryteriów
<code>group_by()</code>	Grupowanie według wybranych kolumn
<code>summarize()</code>	Agregacja wartości w grupach
<code>left_join()</code>	Połączenie odpowiadających sobie rekordów z tabel A i B. Funkcja zwraca NA, jeżeli w tabeli B nie znaleziono pasujących rekordów.

Ze względu na ograniczenia w długości książki nie przedstawię wszystkich funkcji z pakietu *dplyr* ani nie omówię nawet wszystkich sposobów użycia funkcji, które zaprezentuję. Aby dowiedzieć się więcej o tym pakiecie, zajrzyj do książki Hadleya Wickhama i Garretta Golemunda *Język R. Kompletny zestaw narzędzi dla analityków danych* (Helion, 2017). W RStudio znajdziesz też pomocną ściągawkę, która podsumowuje powiązania pomiędzy funkcjami z pakietu *dplyr*. W RStudio wybierz *Help/Cheatsheets/Data Transformation with dplyr* (pomoc/ściągawki/transformacja danych za pomocą *dplyr*).

Operacje kolumnowe

W Excelu wybieranie i pomijanie kolumn często wymaga ich ukrywania lub usuwania. Takie działania trudno kontrolować i odtwarzać, ponieważ ukryte kolumny można łatwo przeoczyć, a te usunięte nie dają się tak łatwo odzyskać. Do wybrania kolumn z ramki danych w R możesz posłużyć się funkcją `select()`. Pierwszym argumentem `select()` (oraz innych omawianych przeze mnie funkcji) jest ramka danych, na której funkcja będzie pracować. Następnie przekazywane są dodatkowe argumenty, którymi w tym przypadku będą nazwy interesujących nas kolumn. W następujący sposób możemy wybrać na przykład kolumny `wynik.matematyka`, `wynik.czytanie`, `id.szkoły` z ramki danych `star`:

```
select(star, wynik.matematyka, wynik.czytanie, id.szkoły)
#> # A tibble: 5,748 x 3
#>   wynik.matematyka wynik.czytanie id.szkoły
#>   <dbl>           <dbl>     <dbl>
#> 1             473             447         63
#> 2             536             450         20
#> 3             463             439         19
#> 4             559             448         69
#> 5             489             447         79
#> 6             454             431          5
```

```
#> 7          423          395          16
#> 8          500          451          56
#> 9          439          478          11
#> 10         528          455          66
#> # ... with 5,738 more rows
```

Za pomocą operatora - możemy *pominąć* podane w wywołaniu select () kolumny:

```
select(star, -wynik.matematyka, -wynik.czytanie, -id.szkoły)
#> # A tibble: 5,748 x 5
#>   rodzaj.klasy   doświadczenie.nau...   płeć   darmowe.wyż...   rasa
#>   <chr>          <dbl> <chr>   <chr>          <chr>
#> 1 mała.klasa      7   dziewczy...   nie   biała
#> 2 mała.klasa     21  dziewczy...   nie   czarna
#> 3 standardowa.klasa.z...  0   chłopiec   tak   czarna
#> 4 standardowa.klasa   16  chłopiec   nie   biała
#> 5 mała.klasa      5   chłopiec   tak   biała
#> 6 standardowa.klasa   8   chłopiec   tak   biała
#> 7 standardowa.klasa.z... 17  dziewczy...   tak   czarna
#> 8 standardowa.klasa   3   dziewczy...   nie   biała
#> 9 mała.klasa     11  dziewczy...   nie   czarna
#> 10 mała.klasa    10  dziewczy...   nie   biała
#> # ... with 5,738 more rows
```

Bardziej elegancką alternatywą jest przekazanie wszystkich niechcianych kolumn w wektorze, który *następnie* pominiemy:

```
select(star, -c(wynik.matematyka, wynik.czytanie, id.szkoły))
#> # A tibble: 5,748 x 5
#>   rodzaj.klasy   doświadczenie.nau...   płeć   darmowe.wyż...   rasa
#>   <chr>          <dbl> <chr>   <chr>          <chr>
#> 1 mała.klasa      7   dziewczy...   nie   biała
#> 2 mała.klasa     21  dziewczy...   nie   czar...
#> 3 standardowa.klasa.z...  0   chłopiec   tak   czar...
#> 4 standardowa.klasa   16  chłopiec   nie   biała
#> 5 mała.klasa      5   chłopiec   tak   biała
#> 6 standardowa.klasa   8   chłopiec   tak   biała
#> 7 standardowa.klasa.z... 17  dziewczy...   tak   czar...
#> 8 standardowa.klasa   3   dziewczy...   nie   biała
#> 9 mała.klasa     11  dziewczy...   nie   czar...
#> 10 mała.klasa    10  dziewczy...   nie   biała
```

Pamiętaj, że w poprzednich przykładach po prostu wywoływałem funkcje. Nie przypisaliśmy rezultatu do żadnego obiektu.

Innym sposobem na skrócenie wywołania select () jest użycie operatora :, który pozwala wybrać wszystkie kolumny znajdujące się pomiędzy dwoma podanymi w wywołaniu kolumnami włącznie. Tym razem przypiszę wyniki do obiektu star. Poniżej wybieram wszystkie kolumny od wynik.matematyka do doświadczenie.nauczyciela włącznie:

```
star <- select(star, wynik.matematyka:doświadczenie.nauczyciela)
head(star)
#> # A tibble: 6 x 4
#>   wynik.matematyka wynik.czytanie rodzaj.klasy   doświadczenie.nauczyciela
#>   <dbl>          <dbl> <chr>          <dbl>
#> 1         473          447 mała.klasa      7
#> 2         536          450 mała.klasa     21
#> 3         463          439 standardowa.klasa.z...  0
```

```

#> 4          559          448 standardowa.klasa          16
#> 5          489          447 mała.klasa             5
#> 6          454          431 standardowa.klasa          8

```

Prawdopodobnie tworzyłeś kiedyś w Excelu kolumny obliczeniowe. W R możesz je stworzyć za pomocą funkcji `mutate()`. Stwórzmy kolumnę `nowa_kolumna`, która będzie zawierała łączny wynik testów z matematyki i czytania. W wywołaniu funkcji `mutate()` *najpierw* podaje się nazwę nowej kolumny, potem znak równości, a na końcu formułę do obliczania. W formule możesz odwoływać się do innych kolumn z ramki danych:

```

star <- mutate(star, nowa_kolumna = wynik.matematyka + wynik.czytanie)
head(star)
#> # A tibble: 6 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasa   doświadcz... nowa_kolumna
#>   <dbl>          <dbl> <chr>          <dbl>          <dbl>
#> 1         473          447 mała.klasa             7            920
#> 2         536          450 mała.klasa            21           986
#> 3         463          439 standardowa.klasa.z...  0            902
#> 4         559          448 standardowa.klasa        16           1007
#> 5         489          447 mała.klasa             5            936
#> 6         454          431 standardowa.klasa         8            885

```

Funkcja `mutate()` ułatwia tworzenie bardziej złożonych obliczeniowych kolumn, takich jak kolumny zawierające wyniki przekształcenia logarymicznego lub zmienne opóźnione. Zajrzyj do dokumentacji, aby uzyskać więcej informacji.

Nazwa `nowa_kolumna` nie jest szczególnie przydatną nazwą dla łącznego wyniku z testu. Na szczęście istnieje funkcja `rename()`, która zmienia nazwy. W drugim argumencie tej funkcji umieszcza się nową nazwę kolumny, która zastąpi starą:

```

star <- rename(star, łączny.wynik = nowa_kolumna)
head(star)
#> # A tibble: 6 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasa   dowiadcz... łączny.wynik
#>   <dbl>          <dbl> <chr>          <dbl>          <dbl>
#> 1         473          447 mała.klasa             7            920
#> 2         536          450 mała.klasa            21           986
#> 3         463          439 standardowa.klasa.z...  0            902
#> 4         559          448 standardowa.klasa        16           1007
#> 5         489          447 mała.klasa             5            936
#> 6         454          431 standardowa.klasa         8            885

```

Operacje wierszowe

Do tej pory operowaliśmy na *kolumnach*. Teraz skupimy się na *wierszach*, zwłaszcza na ich sortowaniu i filtrowaniu. W Excelu sortowanie według wielu kolumn możemy przeprowadzić za pomocą menu *Sortowanie*. Załóżmy na przykład, że chcielibyśmy posortować tę ramkę danych rosnąco według kolumny `rodzaj.klasa`, a następnie `wynik.czytanie`. Odpowiednie ustawienia w menu sortowania w Excelu pokazano na rysunku 8.1.

W R możemy to zrobić za pomocą funkcji `arrange()` z pakietu *dplyr*. Nazwy kolumn, według których ma się odbyć sortowanie, podaje się w kolejności, w jakiej chce się posortować ramkę danych:



Rysunek 8.1. Menu Sortowanie w Excelu

```

arrange(star, rodzaj.klasy, wynik.czytanie)
#> # A tibble: 5,748 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasy doświadczenie... łączny.wynik
#>   <dbl>          <dbl> <chr>          <dbl>          <dbl>
#> 1             412             370 mała.klasa          15             782
#> 2             434             376 mała.klasa          11             810
#> 3             423             378 mała.klasa           6             801
#> 4             405             378 mała.klasa           8             783
#> 5             384             380 mała.klasa          19             764
#> 6             405             380 mała.klasa          15             785
#> 7             439             382 mała.klasa           8             821
#> 8             384             384 mała.klasa          10             768
#> 9             405             384 mała.klasa           8             789
#> 10            423             384 mała.klasa          21             807
#> # ... with 5,738 more rows

```

Nazwę kolumny, względem której chcemy posortować wyniki w porządku malejącym, możemy umieścić w wywołaniu funkcji `desc()`:

```

# Sortujemy malejąco według kolumny rodzaj.klasy i rosnąco według kolumny wynik.czytanie
arrange(star, desc(rodzaj.klasy), wynik.czytanie)
#> # A tibble: 5,748 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasy          doświadczenie... łączny.wynik
#>   <dbl>          <dbl> <dbl> <chr>          <dbl>          <dbl>
#> 1             418             372 standardowa.klasa.z...     7             790
#> 2             399             374 standardowa.klasa.z...    11             773
#> 3             399             374 standardowa.klasa.z...     2             773
#> 4             354             374 standardowa.klasa.z...     7             728
#> 5             354             376 standardowa.klasa.z...     5             730
#> 6             405             376 standardowa.klasa.z...     7             781
#> 7             444             376 standardowa.klasa.z...     3             820
#> 8             399             378 standardowa.klasa.z...    14             777
#> 9             418             378 standardowa.klasa.z...     7             796
#> 10            399             380 standardowa.klasa.z...    11             779
#> # ... with 5,738 more rows

```

Tabele Excela zawierają pomocne menu rozwijane do filtrowania wartości w dowolnej kolumnie według zadanych warunków. W R do przefiltrowania ramki danych wykorzystamy, trafnie nazwaną, funkcję `filter()`. Przefiltrujmy dane w zbiorze `star` i zachowajmy tylko te rekordy, w których `rodzaj.klasy` jest równy `mała.klasa`. Pamiętaj, że sprawdzamy równość, a nie przypisujemy wartości do obiektów, dlatego korzystamy z `==`, a nie z `=`:

```

filter(star, rodzaj.klasy == 'mała.klasa')
#> # A tibble: 1,733 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasy   doświadczenie... łączny.wynik
#>   <dbl>           <dbl> <chr>          <dbl>          <dbl>
#> 1             473             447 mała.klasa         7             920
#> 2             536             450 mała.klasa        21            986
#> 3             489             447 mała.klasa         5            936
#> 4             439             478 mała.klasa        11            917
#> 5             528             455 mała.klasa        10            983
#> 6             559             474 mała.klasa         0           1033
#> 7             494             424 mała.klasa         6            918
#> 8             478             422 mała.klasa         8            900
#> 9             602             456 mała.klasa        14           1058
#> 10            439             418 mała.klasa         8            857
#> # ... with 1,723 more rows

```

W wyniku widzimy, że wywołanie `filter()` wpłynęło *tylko* na liczbę wierszy. Liczba kolumn *nie* zmieniła się. Znajdźmy teraz rekordy, w których wynik testu z matematyki wynosi co najmniej 500 pkt:

```

filter(star, wynik.czytanie >= 500)
#> # A tibble: 233 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasy   doświadczenie... łączny.wynik
#>   <dbl>           <dbl> <chr>          <dbl>          <dbl>
#> 1             559             522 standardowa.klasa         8           1081
#> 2             536             507 standardowa.klasa.z...     3           1043
#> 3             547             565 standardowa.klasa.z...     9           1112
#> 4             513             503 mała.klasa         7           1016
#> 5             559             605 standardowa.klasa.z...     5           1164
#> 6             559             554 standardowa.klasa        14           1113
#> 7             559             503 standardowa.klasa        10           1062
#> 8             602             518 standardowa.klasa        12           1120
#> 9             536             580 mała.klasa        12           1116
#> 10            626             510 mała.klasa        14           1136
#> # ... with 223 more rows

```

Możliwe jest również filtrowanie według wielu kryteriów. Możesz to tego celu wykorzystać operator `&` (*i*) i operator `|` (*lub*). Połączmy nasze dwa poprzednie kryteria za pomocą operatora `&`:

```

# Pobieramy rekordy dotyczące małych klas, w których wynik.czytanie wynosi co najmniej 500
filter(star, rodzaj.klasy == 'mała.klasa' & wynik.czytanie >= 500)
#> # A tibble: 84 x 5
#>   wynik.matematyka wynik.czytanie rodzaj.klasy   doświadczenie... łączny.wynik
#>   <dbl>           <dbl> <chr>          <dbl>          <dbl>
#> 1             513             503 mała.klasa         7           1016
#> 2             536             580 mała.klasa        12           1116
#> 3             626             510 mała.klasa        14           1136
#> 4             602             518 mała.klasa         3           1120
#> 5             626             565 mała.klasa        14           1191
#> 6             602             503 mała.klasa        14           1105
#> 7             626             538 mała.klasa        13           1164
#> 8             500             580 mała.klasa         8           1080
#> 9             489             565 mała.klasa        19           1054
#> 10            576             545 mała.klasa        19           1121
#> # ... with 74 more rows

```

Agregacja i łączenie danych

Lubię nazywać tabele przestawne mianem „WD-40 Excela”, ponieważ ułatwiają one analizę danych, umożliwiając ich „obracanie” w różnych kierunkach. W ramach przykładu odtwórz tabelę przestawną z rysunku 8.2. Pokazuje ona średni wynik z matematyki w zależności od wielkości klasy (zbiór danych *star*).

1. Agregacja/grupowanie według rodzaj.klasy	2. Podsumowanie za pomocą średniej z wynik.matematyka
Etykiety wierszy	Średnia z wynik.matematyka
mała.klasa	491,4702827
standardowa.klasa	483,261
standardowa.klasa.z.nauczycielem.wspomagającym	483,0099256
Suma końcowa	485,6480515

Rysunek 8.2. Jak działają tabele przestawne w Excelu

Jak pokazuje rysunek 8.2, tabela przestawna składa się z dwóch elementów. Najpierw zagregowałem dane według zmiennej *rodzaj.klasy*, a następnie podsumowałem wyniki za pomocą średniej z *wynik.matematyka*. W R to samo można osiągnąć w kilku krokach za pomocą kilku funkcji z pakietu *dplyr*. Najpierw zagregujemy dane za pomocą funkcji *group_by()*. W danych wyjściowych znajduje się linia `# Groups: rodzaj.klasy [3]` wskazująca, że zawartość obiektu *star_grouped* została podzielona na trzy grupy według wartości zmiennej *rodzaj.klasy*:

```
star_grouped <- group_by(star, rodzaj.klasy)
head(star_grouped)
#> # A tibble: 6 x 5
#> # Groups:   rodzaj.klasy [3]
#>   wynik.matematyka wynik.czytanie rodzaj.klasy   doświadczenie... łączny.wynik
#>   <dbl>           <dbl> <chr>           <dbl>           <dbl>
#> 1         473         447 mała.klasa           7             920
#> 2         536         450 mała.klasa           21            986
#> 3         463         439 standardowa.klasa.z...  0             902
#> 4         559         448 standardowa.klasa           16            1007
#> 5         489         447 mała.klasa           5             936
#> 6         454         431 standardowa.klasa           8             885
```

Pogrupowaliśmy nasze dane według jednej zmiennej. Teraz *podsumujmy* je za pomocą funkcji *summarize()* (możesz też skorzystać z *summarise()*). Musimy określić nazwę wynikowej kolumny oraz sposób obliczania jej wartości. W tabeli 8.2 wymieniono niektóre typowe funkcje agregacyjne.

Tabela 8.2. Przydatne funkcje agregacyjne dostępne w pakiecie *dplyr*

Funkcja	Rodzaj agregacji
<i>sum()</i>	Suma
<i>n()</i>	Liczba wartości
<i>mean()</i>	Średnia
<i>max()</i>	Maksimum
<i>min()</i>	Minimum
<i>sd()</i>	Odchylenie standardowe

Średni wynik z matematyki według rozmiaru klasy można otrzymać poprzez wywołanie `summarize()` na naszej zgrupowanej ramce danych:

```
summarize(star_grouped, srednia.z.matematyki = mean(wynik.matematyka))
#> A tibble: 3 x 2
#>   rodzaj.klasy                srednia.z.matematyki
#>   <chr>                        <dbl>
#> 1 mała.klasa                    491.
#> 2 standardowa.klasa            483.
#> 3 standardowa.klasa.z.nauczycielem.wspomagającym 483.
```

Poza pewnymi różnicami w formatowaniu wyniki te nie odbiegają od tych z rysunku 8.2.

Jeśli tabele przestawne są WD-40 Excela, to funkcja `WYSZUKAJ.PIONOWO()` jest jego taśmą klejącą, która pozwala na łatwe łączenie danych pochodzących z wielu źródeł. W oryginalnym zbiorze `star` `id.szkoły` jest identyfikatorem okręgu (dystryktu) szkolnego. Pominęliśmy tę kolumnę we wcześniejszej części tego rozdziału. Wczytajmy ją więc ponownie. Co by było, gdybyśmy oprócz identyfikatora chcieli poznać *nazwy* okręgów? Na szczęście mamy plik `districts.csv`, który zawiera te informacje. Wczytajmy więc oba pliki i opracujmy strategię ich połączenia:

```
star <- read_excel('dane/star/star.xlsx')
head(star)
#> # A tibble: 6 x 8
#>   wynik.mat... wynik.czyt... rodzaj.klasy doświadc... płeć darmowe.wyż... rasa id.szkoły
#>   <dbl>         <dbl> <chr>                <dbl> <chr> <chr>      <chr>      <dbl>
#> 1     473         447 mała.klasa          7 dzie... nie     białą     63
#> 2     536         450 mała.klasa          21 dzie... nie     czar...   20
#> 3     463         439 standardowa....  0 chł... tak     czar...   19
#> 4     559         448 standardowa....  16 chł... nie     białą     69
#> 5     489         447 mała.klasa          5 chł... tak     białą     79
#> 6     454         431 standardowa....  8 chł... tak     białą     5
```

```
districts <- read_csv('dane/star/okręgi.csv')
```

```
#> — Column specification —————
#> cols(
#>   id.szkoły = col_double(),
#>   nazwa_szkoły = col_character(),
#>   hrabstwo = col_character()
#> )
```

```
head(districts)
#> # A tibble: 6 x 3
#>   id.szkoły nazwa_szkoły hrabstwo
#>   <dbl> <chr>                <chr>
#> 1     1 Rosalia             New Liberty
#> 2     2 Montgomeryville    Topton
#> 3     3 Davy                 Wahpeton
#> 4     4 Steelton            Palestine
#> 5     6 Tolchester         Sattley
#> 6     7 Cahokia             Sattley
```

Wygląda na to, że potrzebny nam będzie odpowiednik funkcji `WYSZUKAJ.PIONOWO()`. Na podstawie wartości `id.szkoły` chcemy dopasować wartości zmiennej `nazwa_szkoły` (i ewentualnie `hrabstwo`) ze zbioru `okręgi.csv` do danych ze zbioru `star`. W R wykorzystuje się do tego celu koncepcję *złączenia*, która pochodzi z relacyjnych baz danych — tematu, o którym wspomniałem w rozdziale 5. Najbliższym

odpowiednikiem funkcji `WYSZUKAJ.PIONOWO()` jest złączenie lewostronne zewnętrzne, które w *dplyr* można wykonać za pomocą funkcji `left_join()`. W wywołaniu najpierw umieszcza się tabelę podstawową (*star*), a następnie tabelę, w której należy szukać dopasowań (*districts*). Funkcja znajdzie i zwróci dopasowanie (lub wartość NA w przypadku jego braku) dla każdego rekordu w zbiorze *star*. Aby zmniejszyć ilość informacji w konsoli, zachowam jedynie niektóre kolumny ze zbioru *star*:

```
# Złączenie lewostronne zewnętrzne tabel star i districts
left_join(select(star, id.szkoły, wynik.matematyka, wynik.czytanie), districts)
#> Joining, by = "id.szkoły"
#> # A tibble: 5,748 x 5
#>   id.szkoły wynik.matematyka wynik.czytanie nazwa_szkoły   hrabstwo
#>   <dbl>         <dbl>         <dbl>         <dbl> <chr>
#> 1         63           473           447   Ridgeville   New Liberty
#> 2         20           536           450   South Heights Selmont
#> 3         19           463           439   Bunnlevel    Sattley
#> 4         69           559           448   Hokah        Gallipolis
#> 5         79           489           447   Lake Mathews Sugar Mountain
#> 6          5           454           431   NA            NA
#> 7         16           423           395   Calimesa     Selmont
#> 8         56           500           451   Lincoln Heights Topton
#> 9         11           439           478   Moose Lake   Imbery
#> 10        66           528           455   Siglerville  Summit Hill
#> # ... with 5,738 more rows
```

Funkcja `left_join()` jest całkiem sprytna. Sama „domyśliła się”, że złączenia należy dokonać na podstawie `id.szkoły`, i dołączyła do wyniku nie tylko nazwę szkoły, ale także hrabstwo. Aby dowiedzieć się więcej o łączeniu danych, zapoznaj się z dokumentacją.

W R brakujące obserwacje są reprezentowane przez specjalną wartość NA. Wydaje się, że funkcji nie udało się znaleźć na przykład dopasowania dla nazwy szkoły w piątym okręgu. W przypadku `WYSZUKAJ.↪PIONOWO()` spowodowałoby to błąd `#N/A`. Wartość NA *nie* oznacza, że obserwacja jest równa zeru, a jedynie brak wartości. Podczas programowania w R możesz napotkać także inne specjalne wartości, takie jak `NaN` lub `NULL`. Więcej informacji na ich temat znajdziesz w dokumentacji.

dplyr i potęga operatora potoku (%>%)

Jak zaczynasz zauważać, funkcje z pakietu *dplyr* to potężne i intuicyjne narzędzie dla każdego, kto kiedyś pracował z danymi, w tym w Excelu. Jak wie każdy, kto pracował z danymi, rzadko udaje się przygotować je w jednym kroku. W ramach przykładu rozważ typowe zadanie analizy danych, które możesz wykonać na zbiorze *star*:

Znajdź średnie wyniki testu czytania w poszczególnych rodzajach klas. Posortuj je malejąco.

Wiedząc co nieco o pracy z danymi, możemy podzielić to zadanie na trzy kroki:

1. Pogrupuj dane według rodzaju klasy.
2. Znajdź średnią z testu czytania w każdej grupie.
3. Posortuj wyniki od najwyższego do najniższego.

Z pomocą pakietu *dplyr* możemy to zrobić mniej więcej tak:

```
star_grouped <- group_by(star, rodzaj.klasy)
star_avg_reading <- summarize(star_grouped, średnia.z.czytania = mean(wynik.czytanie))
```

```

star_avg_reading_sorted <- arrange(star_avg_reading, desc(średnia.z.czytania))
star_avg_reading_sorted
#> # A tibble: 3 x 2
#>   rodzaj.klasy                średnia.z.czytania
#>   <chr>                        <dbl>
#> 1 mała.klasa                    441.
#> 2 standardowa.klasa.z.nauczycielem.wspomagającym 435.
#> 3 standardowa.klasa            435.

```

W końcu otrzymaliśmy odpowiedź, ale jej znalezienie wymagało sporej liczby kroków i może być trudne do powtórzenia w przypadku innych nazw obiektów i użycia innych funkcji. Alternatywą jest połączenie kolejnych wywołań funkcji za pomocą operatora `%>%` zwanego **operatorem potoku** (ang. *pipe*). Pozwala on na przekazanie danych wyjściowych z jednej funkcji bezpośrednio do drugiej. Dzięki temu jesteśmy w stanie uniknąć ciąglego zmieniania nazw danych wejściowych i wyjściowych. Domyślny skrót klawiaturowy tego operatora to *Ctrl+Shift+M* w systemie Windows i *Cmd+Shift+M* na Macu.

Odtwórzmy poprzednie kroki za pomocą operatora potoku. Każdą funkcję umieść w osobnym wierszu. Połącz wiersze za pomocą `%>%`. Chociaż umieszczanie każdego kroku w osobnej linii nie jest konieczne, często preferuje się je ze względu na czytelność. Podczas prac z operatorem potoku nie ma również konieczności zaznaczania całego bloku kodu przed uruchomieniem. Po prostu umieść kursor w dowolnym miejscu poniższego kodu, aby go uruchomić:

```

star %>%
  group_by(rodzaj.klasy) %>%
  summarise(średnia.z.czytania = mean(wynik.czytanie)) %>%
  arrange(desc(średnia.z.czytania))
#> # A tibble: 3 x 2
#>   rodzaj.klasy                średnia.z.czytania
#>   <chr>                        <dbl>
#> 1 mała.klasa                    441.
#> 2 standardowa.klasa.z.nauczycielem.wspomagającym 435.
#> 3 standardowa.klasa            435.

```

Brak konieczności uwzględniania danych w wywołaniu każdej funkcji może być na początku trochę dezorientujący, ale porównaj ostatni listing z poprzednim, a zobaczysz, o ile wydajniejsze może być to podejście. Co więcej, operator potoku może być używany również z funkcjami spoza pakietu *dplyr*. Jako przykład wypiszmy kilka pierwszych wierszy z wyniku. Aby to zrobić, umieść wywołanie `head()` na końcu potoku:

```

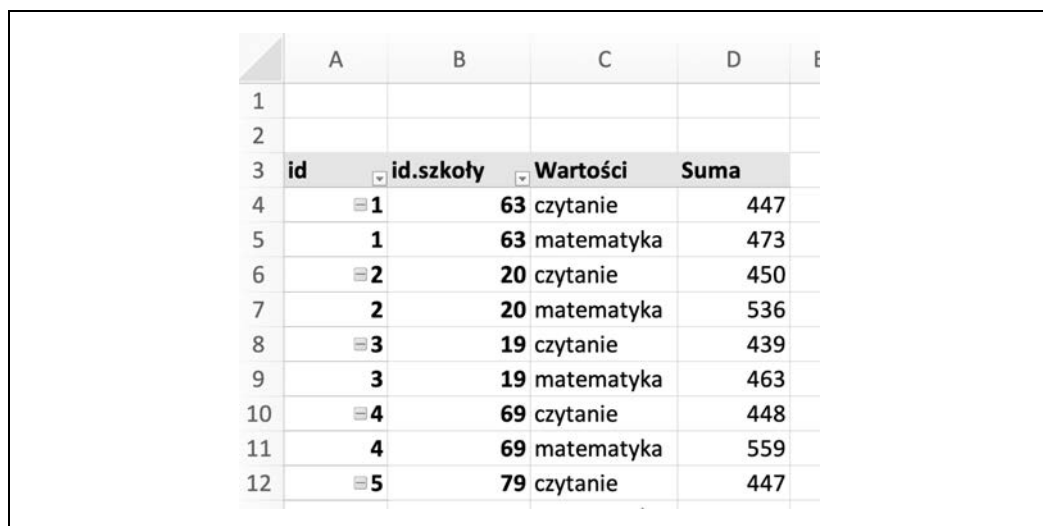
# Średnia z czytania i matematyki w każdym okręgu
star %>%
  group_by(id.szkoły) %>%
  summarise(średnia.z.czytania = mean(wynik.czytanie), średnia.z.matematyki = mean(
    ↪(wynik.matematyka)) %>%
  arrange(id.szkoły) %>%
  head()
#> # A tibble: 6 x 3
#>   id.szkoły średnia.z.czytania średnia.z.matematyki
#>   <dbl>         <dbl>         <dbl>
#> 1         1             444.             492.
#> 2         2             407.             451.
#> 3         3             441.             491.
#> 4         4             422.             468.
#> 5         5             428.             460.
#> 6         6             428.             470.

```

Przekształcanie danych za pomocą tidyr

Chociaż prawdą jest, że funkcje `group_by()` i `summarize()` pełnią rolę odpowiedników tabeli przestawnej w R, to funkcje te nie potrafią zrobić wszystkiego, co potrafi tabela przestawna w Excelu. Co by było, gdybyś poza agregacją chciał zmienić *kształt* danych lub układ wierszy i kolumn? Ramka danych `star` zawiera na przykład oddzielne kolumny dla wyników testu z matematyki i czytania (są to odpowiednio: `wynik.matematyka` i `wynik.czytanie`). Chciałbym połączyć je w jedną kolumnę o nazwie `wynik` i dodać do danych kolumnę `rodzaj.testu`, która wskazywałaby, czy obserwacja dotyczy wyniku z matematyki, czy z czytania. Chciałbym również pozostawić w danych kolumnę `id.szkoły`.

Na rysunku 8.3 pokazano jeden z możliwych sposobów realizacji tych zmian w Excelu. Zauważ, że zmieniłem nazwy pól z obszaru *Wartości* na czytanie i matematyka. Jeśli chcesz dokładniej przyjrzeć się tej tabeli, to znajdziesz ją w pliku `r08.xlsx`, który znajduje się w materiałach dołączonych do książki (<https://ftp.helion.pl/przyklady/zaanda.zip>). W tym przykładzie ponownie skorzystałem z kolumny indeksu. W przypadku jej braku tabela przestawna próbowałaby „zwinąć” wszystkie wartości na podstawie `id.szkoły`.



	A	B	C	D	E
1					
2					
3	id	id.szkoły	Wartości	Suma	
4	1	63	czytanie	447	
5	1	63	matematyka	473	
6	2	20	czytanie	450	
7	2	20	matematyka	536	
8	3	19	czytanie	439	
9	3	19	matematyka	463	
10	4	69	czytanie	448	
11	4	69	matematyka	559	
12	5	79	czytanie	447	

Rysunek 8.3. Zmiana kształtu danych `star` w Excelu

Do zmiany kształtu danych w R możesz wykorzystać pakiet `tidyr` stanowiący rdzeń kolekcji `tidyverse`. Podobnie jak w Excelu, dodanie kolumny z indeksem ułatwi zmianę kształtu danych. Kolumnę z indeksem możesz dodać za pomocą funkcji `row_number()`:

```
star_pivot <- star %>%  
  select(c(id.szkoły, wynik.czytanie, wynik.matematyka)) %>%  
  mutate(id = row_number())
```

Do zmiany kształtu ramki danych wykorzystamy funkcje `pivot_longer()` i `pivot_wider()`. Obie pochodzą z pakietu `tidyr`. Spójrz na rysunek 8.3 i zastanów się, co stało się z naszym zbiorem danych, gdy połączyliśmy wyniki z matematyki i czytania w jedną kolumnę. Czy zbiór danych wydłużył się, czy

rozszerzył? W tym przypadku dodaliśmy do niego wiersze, a więc nasz zbiór danych wydłużył się. W wywołaniu `pivot_longer()` za pomocą argumentu `cols` musimy określić, o które kolumny chcemy wydłużyć dane. Nazwę wynikowej kolumny określa się w argumencie `values_to`. Argument `names_to` pozwala nazwać kolumnę, która będzie wskazywała, czy dany wiersz odnosi się do wyniku z matematyki, czy z czytania:

```
star_long <- star_pivot %>%
  pivot_longer(cols = c(wynik.matematyka, wynik.czytanie),
               values_to = 'wynik', names_to = 'rodzaj.testu')

head(star_long)
#> # A tibble: 6 x 4
#>   id.szkoły id rodzaj.testu   wynik
#>   <dbl> <int> <chr>         <dbl>
#> 1     63     1 wynik.matematyka 473
#> 2     63     1 wynik.czytanie 447
#> 3     20     2 wynik.matematyka 536
#> 4     20     2 wynik.czytanie 450
#> 5     19     3 wynik.matematyka 463
#> 6     19     3 wynik.czytanie 439
```

Świetna robota. Ale czy istnieje sposób, aby zmienić nazwy `wynik.matematyka` i `wynik.czytanie` na `matematyka` i `czytanie`? Otóż tak. W tym celu należy skorzystać z `recode()` — kolejnej pomocnej funkcji z pakietu `dplyr`, którą można zastosować wraz z `mutate()`. Funkcja `recode()` działa nieco inaczej niż inne funkcje z tego pakietu. W jej przypadku nazwy „starych” wartości umieszcza się *przed* znakiem równości. Po znaku równości umieszcza się nowe nazwy. Za pomocą funkcji `distinct()` z `dplyr` możesz potwierdzić, że wszystkie wiersze zostały nazwane `matematyka` lub `czytanie`:

```
# Zmiana nazwy wynik.matematyka i wynik.czytanie na matematyka i czytanie
star_long <- star_long %>%
  mutate(rodzaj.testu = recode(rodzaj.testu, 'wynik.matematyka' = 'matematyka',
                              'wynik.czytanie' = 'czytanie'))

distinct(star_long, rodzaj.testu)
#> # A tibble: 2 x 1
#>   rodzaj.testu
#>   <chr>
#> 1 matematyka
#> 2 czytanie
```

Po wydłużeniu ramki danych za pomocą funkcji `pivot_wider()` możemy ją ponownie rozszerzyć. Tym razem w argumencie `values_from` określamy, z której kolumny mają pochodzić wartości dla nowo powstałych kolumn. Nazwy nowych kolumn zostaną pobrane z kolumny, której nazwa zostanie przekazana funkcji w argumencie `names_from`:

```
star_wide <- star_long %>%
  pivot_wider(values_from = 'wynik', names_from = 'rodzaj.testu')

head(star_wide)
#> # A tibble: 6 x 4
#>   id.szkoły id matematyka czytanie
#>   <dbl> <int>   <dbl>   <dbl>
#> 1     63     1     473     447
#> 2     20     2     536     450
#> 3     19     3     463     439
#> 4     69     4     559     448
#> 5     79     5     489     447
#> 6     5      6     454     431
```

Zmiana kształtu danych w R jest dość trudna. Jeśli masz wątpliwości, zadaj sobie pytania: „Czy rozszerzam, czy wydłużam dane? Jak zrobiłbym to w tabeli przestawnej?”. Znacznie uprościsz sobie kodowanie, jeśli będziesz potrafił logicznie przejść przez etapy niezbędne do osiągnięcia stanu końcowego.

Wizualizacja danych w ggplot2

Pakiet *dplyr* oferuje znacznie większe możliwości manipulacji danymi, ale na razie skupmy się na wizualizacji danych. Zwłaszcza za pomocą pakietu *ggplot2*, który jest kolejnym elementem kolekcji *tidyverse*. Pakiet *ggplot2*, wzorowany na opracowanej przez Lelanda Wilkinsona *gramatyce grafiki* i zawdzięczający jej swą nazwę, oferuje uporządkowane podejście do tworzenia wykresów, które odwzorowuje sposób, w jaki elementy mowy łączą się w zdanie (stąd słowo „gramatyka” w nazwie).

W tym podrozdziale omówię niektóre z podstawowych elementów i rodzajów wykresów, które są dostępne w *ggplot2*. Więcej informacji na temat pakietu znajdziesz w książce *ggplot2: Elegant Graphics for Data Analysis* autorstwa twórcy tego pakietu, Hadleya Wickhama (Springer, 2009). W RStudio znajdziesz również ściągawkę ułatwiającą pracę z tym pakietem. Aby się z nią zapoznać, wybierz *Help/Cheatsheets/Data Visualization with ggplot2* (pomoc/ściągawki/wizualizacja danych za pomocą *ggplot2*). Niektóre z ważnych elementów pakietu przedstawiłem w tabeli 8.3. Dostępne są też inne. Aby uzyskać więcej informacji, zajrzyj do wymienionych powyżej materiałów.

Tabela 8.3. Podstawowe elementy w *ggplot2*

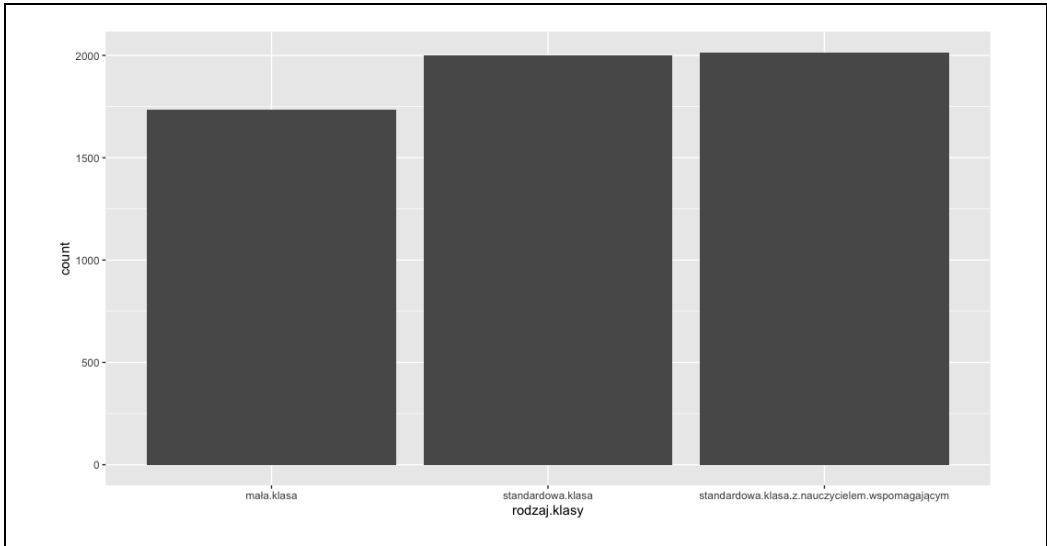
Element	Opis
<code>data</code>	Źródło danych
<code>aes</code>	Odwzorowanie danych na elementy wizualne (osie <i>x</i> i <i>y</i> , kolor, rozmiar itd.)
<code>geom</code>	Typ obiektu geometrycznego obserwowany na wykresie (linie, słupki, kropki itd.)

Zwizualizujmy liczby obserwacji dla każdego rodzaju klasy w postaci wykresu słupkowego. Zaczniemy od funkcji `ggplot()`. W jej wywołaniu musimy określić trzy elementy opisane w tabeli 8.3:

```
ggplot(data = star, ①  
       aes(x = rodzaj.klasy))+ ②  
geom_bar() ③
```

- ① W argumencie `data` określa się źródło danych.
- ② Odwzorowanie danych na elementy wizualne określa się za pomocą funkcji `aes()`. W tym miejscu odwzorowujemy kolumnę `rodzaj.klasy` w oś *x* wynikowego wykresu.
- ③ Znając dane i ich odwzorowanie na elementy, za pomocą funkcji `geom_bar()` na wykresie umieszczamy wybrany obiekt geometryczny. Wyniki przedstawiono na rysunku 8.4.

Tak jak w przypadku operatora potoku, zapisywanie każdego działania w osobnej linii nie jest konieczne, ale często preferowane ze względu na czytelność. Możliwe jest również uruchomienie całego kodu tworzącego wykres poprzez umieszczenie kursora w dowolnym miejscu bloku kodu i uruchomienie wykonania.

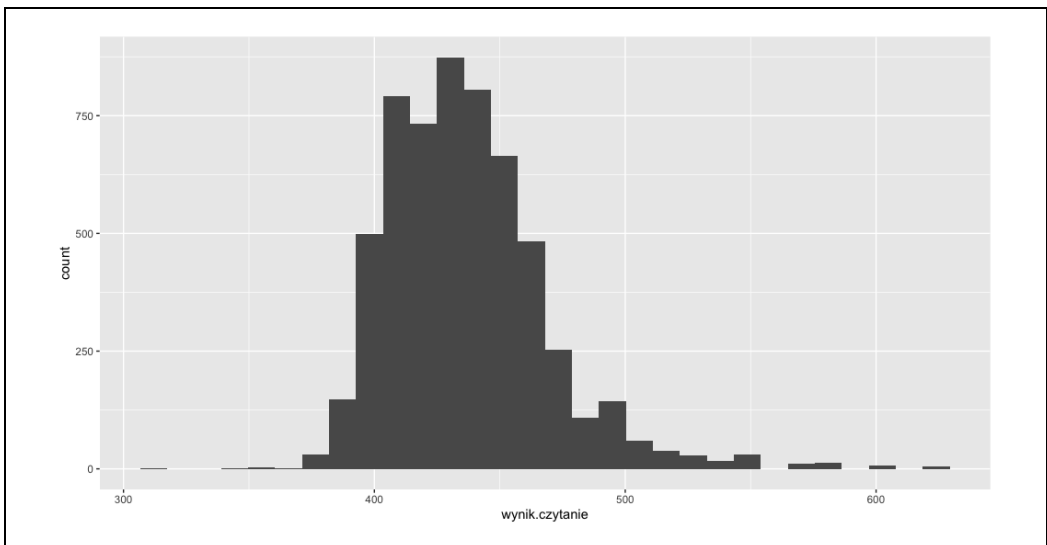


Rysunek 8.4. Wykres słupkowy w `ggplot2` (angielski opis osi y został wygenerowany automatycznie i wynika ze specyfiki działania pakietu `ggplot2`; o tym, jak go zmienić, dowiesz się w dalszej części tego rozdziału)

Dzięki modułowemu podejściu łatwo jest modyfikować wizualizacje tworzone za pomocą `ggplot2`. Możemy na przykład zmienić nasz wykres na histogram wyników testu z czytania. Wystarczy zmienić wartości na osi `x` i wykreślić wyniki za pomocą funkcji `geom_histogram()`. Powstanie w ten sposób histogram pokazany na rysunku 8.5:

```
ggplot(data = star, aes(x = wynik.czytanie)) +
  geom_histogram()

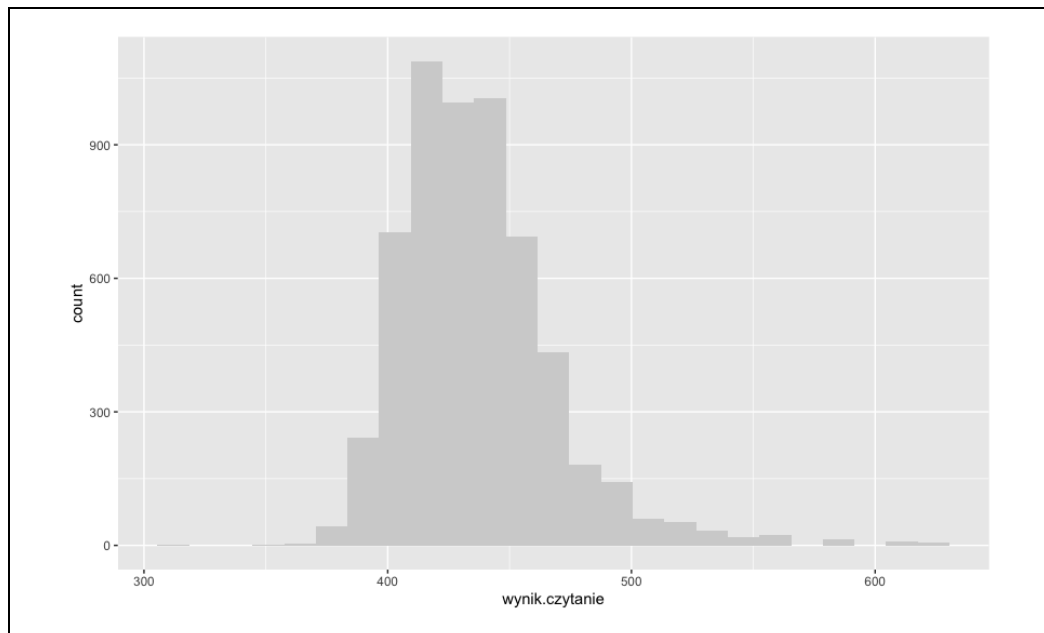
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Rysunek 8.5. Histogram w `ggplot2`

Wykresy tworzone za pomocą *ggplot2* można dostosowywać na wiele sposobów. Być może zauważyłeś, że komunikat wyjściowy wyświetlony podczas tworzenia poprzedniego wykresu poinformował nas, że na histogramie zastosowano 30 przedziałów (ang. *bins*). Za pomocą kilku dodatkowych argumentów funkcji `geom_histogram()` możemy zmienić tę liczbę na 25 i zastosować wypełnienie w kolorze różowym. W ten sposób otrzymamy histogram pokazany na rysunku 8.6:

```
ggplot(data = star, aes(x = wynik.czytanie))+  
  geom_histogram(bins = 25, fill = 'pink')
```



Rysunek 8.6. Dostosowany histogram stworzony w *ggplot2*

Za pomocą funkcji `geom_boxplot()` możesz wygenerować wykres pudełkowy (ang. *boxplot*) pokazany na rysunku 8.7:

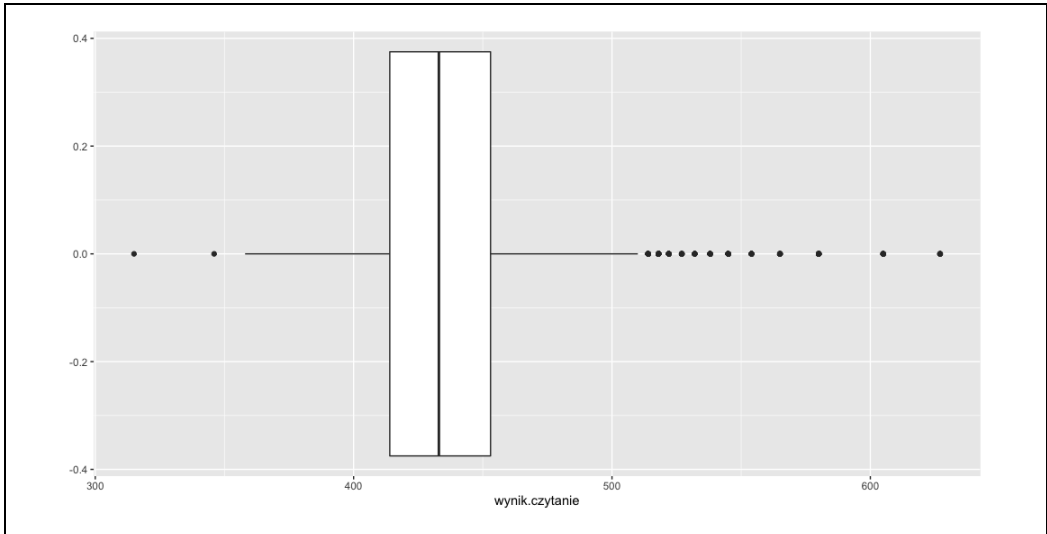
```
ggplot(data = star, aes(x = wynik.czytanie))+  
  geom_boxplot()
```

W każdym z dotychczasowych przykładów możemy „odwrócić” wykres poprzez odwzorowanie interesującej nas zmiennej na oś *y* zamiast na oś *x*. Spróbujmy to zrobić z wykresem pudełkowym. Rysunek 8.8 przedstawia wynik wywołania poniższego kodu:

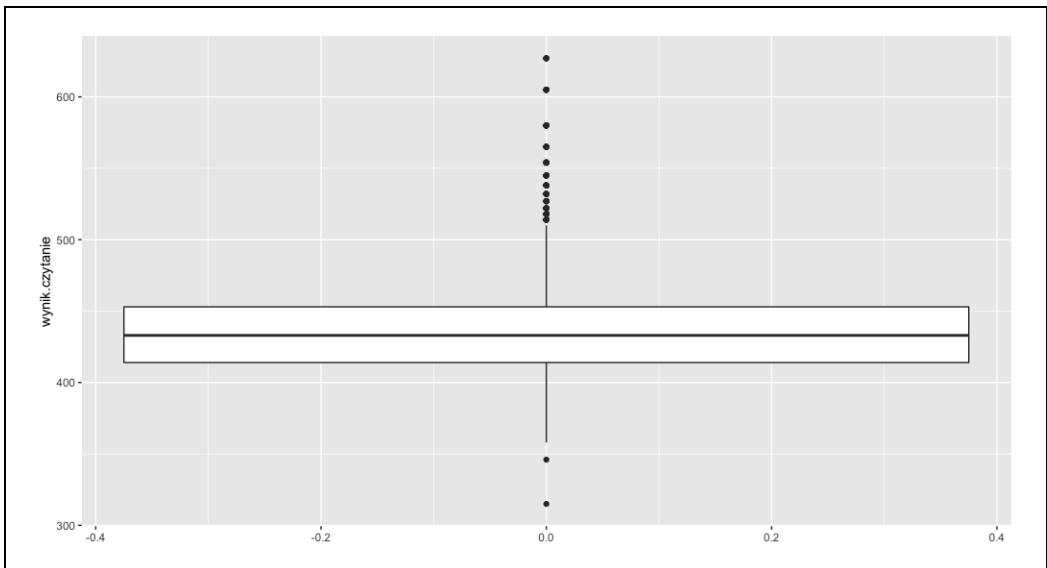
```
ggplot(data = star, aes(y = wynik.czytanie))+  
  geom_boxplot()
```

Stworzymy teraz wykres pudełkowy dla każdego rodzaju klasy. Aby to zrobić, na osi *x* przedstawimy `wynik.czytanie`, a na osi *y* rodzaj `klasy`. Efekty pokazano na rysunku 8.9:

```
ggplot(data=star, aes(x = wynik.matematyka, y = wynik.czytanie))+  
  geom_point()
```



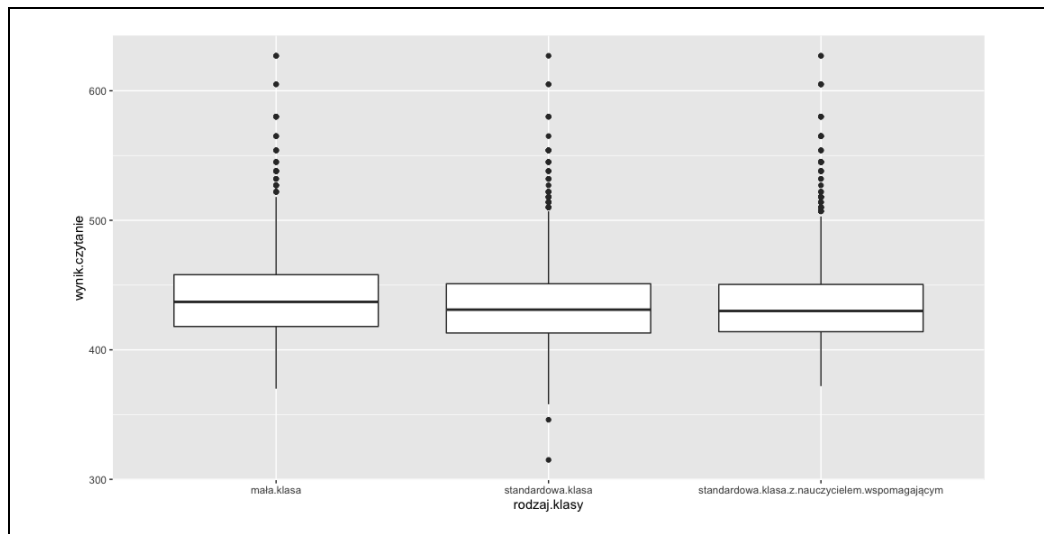
Rysunek 8.7. Wykres pudełkowy



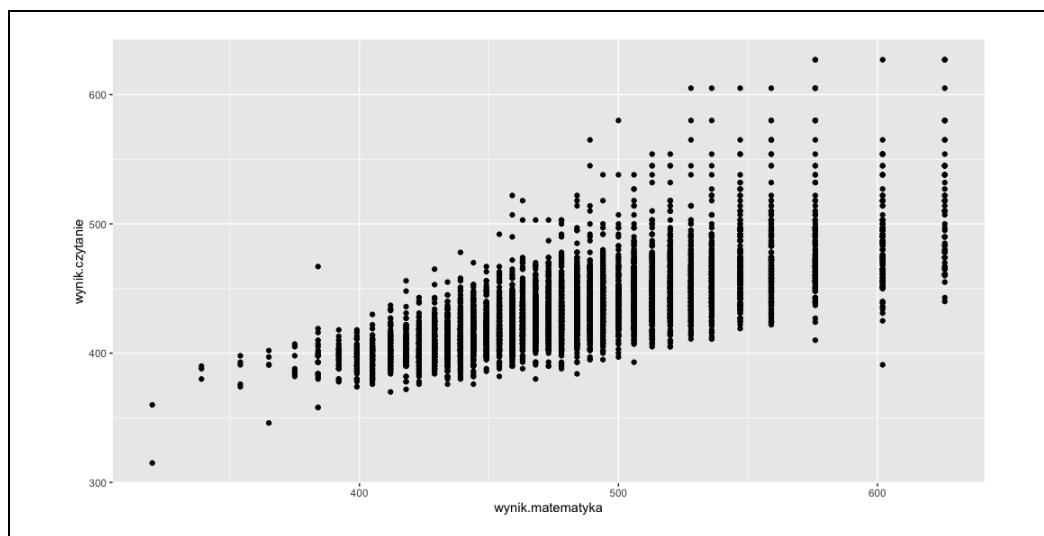
Rysunek 8.8. „Odwrócony” wykres pudełkowy

W podobny sposób możemy wykorzystać funkcję `geom_point()` do stworzenia wykresu rozrzutu wyników z matematyki i czytania (zmapowanych na oś x i y). W wyniku otrzymasz wykres pokazany na rysunku 8.10:

```
ggplot(data=star,aes(x = wynik.matematyka,y = wynik.czytanie))+
  geom_point()
```

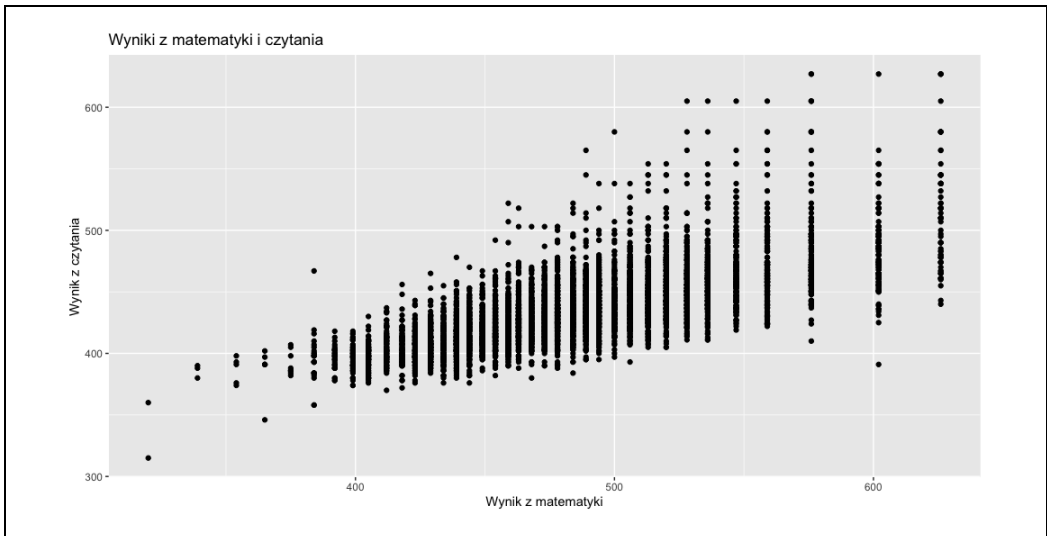
Rysunek 8.9. Wykres pudełkowy z podziałem na grupy



Rysunek 8.10. Wykres rozrzutu

Za pomocą kilku dodatkowych funkcji z pakietu `ggplot2` możemy dodać do rysunku etykiety osi x i y oraz tytuł (w ten sposób można też dodać polskie etykiety do poprzednich wykresów z tego rozdziału). Efekty uruchomienia poniższego kodu pokazano na rysunku 8.11:

```
ggplot(data = star, aes(x = wynik.matematyka, y = wynik.czytanie))+
  geom_point() +
  xlab('Wynik z matematyki') + ylab('Wynik z czytania')+
  ggtitle('Wyniki z matematyki i czytania')
```



Rysunek 8.11. Wykres rozrzutu z niestandardowymi etykietami osi i tytułem

Wnioski

Pakiety *dplyr* i *ggplot2* potrafią wiele więcej, ale to, co zaprezentowałem, wystarczy, aby przejść do właściwego zadania, którym jest eksploracja i testowanie relacji w danych. Zagadnienia te będą tematem rozdziału 9.

Ćwiczenia

W zbiorze materiałów dołączonych do tej książki (<https://ftp.helion.pl/przyklady/zaanda.zip>), w podfolderze *census*, znajdziesz pliki *census.csv* i *census-oddziały.csv*. Wczytaj je w R i wykonaj następujące czynności:

1. Posortuj dane rosnąco według regionu i oddziału oraz malejąco według liczby ludności. (Aby to zrobić, musisz połączyć ze sobą zbiory danych). Wyniki zapisz w arkuszu Excela.
2. Usuń pole kod.pocztowy z połączonego zbioru danych.
3. Utwórz nową kolumnę *gęstość.zaludnienia*, która będzie równa liczbie ludności podzielonej przez powierzchnię.
4. Zwizualizuj relację między powierzchnią a populacją dla wszystkich obserwacji z 2015 roku.
5. Znajdź całkowitą liczbę ludności w każdym regionie w 2015 roku.
6. Utwórz tabelę zawierającą nazwy stanów i liczbę ludności, w której liczba ludności z każdego roku od 2010 do 2015 przechowywana jest w osobnej kolumnie.

Skorowidz

A

agregacja, 131, 183
aktualizacja pakietów, 109, 167
alfa, 58
alias, 170
 plt, 179
 sns, 173
Anaconda, 166, 167
analityka biznesowa, 88, 94
analiza, 57
 dwuwymiarowa, 72
 jednowymiarowa, 72
 wariancji, 81
 what-if, 67
 what-if przedziału ufności, 68
analiza danych, 87
 eksploracyjna, 19, 60, 144, 194
 Python, 193
Analysis ToolPak, 33
 regresja, 80
 statystyka opisowa, 34
 test t, 61
API, application programming
 interface, 117
arkusze kalkulacyjne, 90

B

bazy danych, 92
białe znaki, 163
biblioteka NumPy, 170
biblioteka
 pandas, 171, 172, 179
 seaborn, 173, 179, 186

błąd

standardowy, 64, 81, 83
systematyczny, 55

C

centralne twierdzenie
 graniczne, 51
Code, 161
CRAN, 108

D

data mining, 67
dodatek Data Analysis ToolPak,
 33
dokument PEP8, 164
dominanta, 29
dyrektywy, 166

E

edytor skryptów, 102
eksploracja
 ramki danych, 175
 zmiennych ilościowych, 29
 zmiennych kategorialnych, 27
eksploracyjna analiza danych, 19
etyka, 203

F

funkcja
 abs, 105
 arrange, 126, 128
 array, 170

bind_cols, 153
c, 113, 123
chdir, 174
cor, 148
count, 144, 184
crosstab, 194
data, 116
desc, 129
describe, 121, 194
describeBy, 145
dim, 152
displot, 196
distinct, 136
facet_wrap, 145
factor, 116
file.exists, 118
filter, 126, 129
fit, 152
geom_bar, 137
geom_boxplot, 139
geom_histogram, 138
geom_smooth, 150
get_dataset_names, 173
getcwd, 174
ggplot, 137, 150
glance, 152
glimpse, 121
group_by, 126, 131, 135
head, 134, 199
INDEKS, 114, 122
initial_split, 152
is.data.frame, 116
is.vector, 113
isfile, 174
left_join, 126, 133

linear_reg, 152
LinearRegression, 199
linregress, 197
lm, 150, 152
load_data set, 173
masy prawdopodobieństwa, 48
max, 131, 184
MAX, 33
mean, 131, 184
mean_squared_error, 200
MEDIANA, 30
melt, 185
min, 131, 184
MIN, 33
MODUŁ.LICZBY, 105
mutate, 126, 128, 136
n, 131
ODCH.STAND.POPUL, 33
ODCH.STANDARD.PRÓBK, 33
pairplot, 197
pairs, 148
pivot_longer, 135
pivot_wider, 135, 136, 144
predict, 153, 199
r2_score, 200
read_csv, 119, 175, 193
read_excel, 174
read_xlsx, 120
readr, 120
recode, 136
regplot, 198
regr.fit, 199
rename, 126, 128
row_number, 135
ROZKŁ.NORMALNY, 48
rsq, 153
sd, 131
select, 126
sqrt, 102, 166
ŚREDNIA, 30
std, 184
str, 113, 116
sum, 131, 184
summarize, 126, 131, 135

summary, 121, 150
t.test, 148
testing, 152
tidy, 152
train_test_split, 199
training, 152
ttest_ind, 196
type, 165
View, 120
WARIANCJA.POP, 33
WARIANCJA.PRÓBK, 33
write_csv, 123
write_xlsx, 123
WSP.KORELACJI, 73
WYST.NAJCZĘŚCIEJ, 30
WYST.NAJCZĘŚCIEJ.TABL, 30
WYSZUKAJ.PIONOWO, 93, 132
funkcje agregacyjne, 131

G

graficzny interfejs użytkownika, GUI, 94

H

hipoteza
alternatywna, 57
badawcza, 55, 56
statystyczna, 55, 56
zerowa, 57, 62
hipotezy
testowanie, 147, 196
histogram, 36, 138, 188
niestandardowy, 188
rozkładu normalnego, 46
z podziałem na grupy, 37

I

importowanie danych, 117, 119
indeksowanie
tablice, 171
wektory, 114

instalowanie pakietów, 166
interfejs programowania aplikacji, API, 117
istotność statystyczna, 58, 77

J

język
Markdown, 161
Python, 94, 157
R, 94, 99
SQL, 93
VBA, 91
języki programowania danych, 94
Jupyter, 158
Notebook, 160

K

klasyfikacja zmiennych, 24
kolekcje, 169
kolumny, 20, 92
kontrola wersji, 202
korelacja, 71, 85
dodatnia, 73
ujemna, 72
kreator importu danych, 119
krzywa dzwonowa, 46

L

linia trendu, 79
lista, 169
Looker, 94
losowość, 41

Ł

Łączenie
danych, 131, 183
metod, 184

M

macierz korelacji, 74
Markdown, 161
mediana, 29

- metoda, 165
 - corr, 196
 - describe, 175
 - drop, 180
 - metoda groupby, 183
 - head, 173
 - iloc, 177
 - info, 175
 - loc, 177
 - merge, 184
 - najmniejszych kwadratów, 82
 - rename, 181
 - sort_values, 182
 - upper, 165
 - write_csv, 178
 - write_xlsx, 178
 - metody statystyczne, 202
 - miary
 - tendencji centralnej, 29
 - zmienności, 31
 - model liniowy, 77
 - moduł, *Patrz* pakiet, biblioteka
 - moduł matplotlib, 179
- N**
- nauka o danych, 88
 - NumPy, 170
- O**
- obiekty, 106
 - obiekty typu Series, 180
 - obserwacje, 21, 92
 - odchylenie, 31
 - standardowe, 32
 - operacje
 - kolumnowe, 126, 180
 - wierszowe, 128, 182
 - operator
 - \$, 123
 - &, 130
 - :, 122
 - ?, 163
 - |, 130
 - ~, 145
 - potoku, 133
 - zakresu, 177
 - operatory
 - arytmetyczne, 104, 162
 - porównania, 105
- P**
- pakiet
 - tidyr, 135
 - dplyr, 126, 133
 - ggplot2, 137
 - openpyxl, 174
 - pyplot, 179
 - scikit-learn, 193
 - seaborn, 173
 - statsmodels, 197, 200
 - tidymodels, 152, 154
 - pakiety, 108, 166
 - aktualizacja, 109
 - pandas, 171
 - ramki danych, 172
 - plan analizy, 57
 - platformy analityki biznesowej, 94
 - R
 - pliki
 - .csv, 117
 - .ipynb, 158
 - .py, 158
 - pole, 92
 - Power BI, 94
 - Power Pivot, 92
 - Power Query, 91
 - Power View, 92
 - poziom istotności, 58
 - pozorny związek, 84
 - prawdopodobieństwo, 41
 - bezwarunkowe, 42
 - warunkowe, 42
 - prawo wielkich liczb, 52
 - programowanie obiektowe, OOP, 180
 - próbka, 55
 - przedział ufności, 63, 64, 150
 - analiza what-if, 68
 - obliczanie, 65
 - przekształcanie danych, 135, 185
 - przestrzeń zdarzeń
 - elementarnych, 41
 - przyczynowość, 71
 - p-wartość, 81
 - Python, 157
 - agregacja, 183
 - aktualizacja, 167
 - analizowanie danych, 193
 - importowanie danych, 174
 - łączenie danych, 183
 - moduły, 166
 - operacje kolumnowe, 180
 - operacje wierszowe, 182
 - operatory arytmetyczne, 162
 - pakiety, 166
 - przekształcanie danych, 185
 - struktury danych, 169
 - typy danych, 165
 - wizualizacja danych, 186
- R**
- agregacja, 131
 - analiza danych, 143
 - importowanie danych, 117, 119
 - instalacja, 100, 117
 - komentarze, 104
 - łączenie danych, 131
 - operacje kolumnowe, 126
 - operacje wierszowe, 128
 - operatory arytmetyczne, 104
 - operatory porównania, 105
 - pakiety, 108
 - przekształcanie danych, 135
 - przetwarzanie danych, 126
 - ramki danych, 115
 - struktury danych, 112
 - typy danych, 107
 - wizualizacja danych, 137
 - wykres, 103

ramka danych, 115
 eksploracja, 175
pandas, 181
ramki danych pandas, 172
regresja
 liniowa, 76, 78, 150, 197
 liniowa wielowymiarowa, 83
reguła trzech sigm, 47, 49
rekord, 92
relacyjne bazy danych, 92
reprezentatywna próbka, 55
R-kwadrat, 153
rozkład normalny, 46, 50, 63
 standardowy, 63
rozkład prawdopodobieństwa
 skumulowany, 44
rozkład t-Studenta, 63
rozkłady prawdopodobieństwa, 42
 ciągłe, 46
 dyskretne, 43
rozrzut, 141
rozsęp międzykwartyłowy, 38
równanie regresji, 82
 liniowej, 77
RStudio, 99, 118

S

słownik, 169, 182
SQL, structured query language,
 93
statystyka, 87
 testowa, 63
statystyki opisowe, 29, 34
stos analizy danych, 87, 89
struktury danych
 w R, 112
 w Pythonie, 169
system kontroli wersji, 202
system
 pip, 166
 zarządzania relacyjną bazą
 danych, 93

Ś

średnia arytmetyczna, 29

T

tabela, 92
 dwukierunkowa, 28
 jednokierunkowa, 27
 przetawna, 27
Tableau, 94
tablice NumPy, 170
 indeksowanie, 171
 wybieranie elementów, 171
test dwustronny, 59
test t-Studenta, 61, 69
 dla prób niezależnych, 58,
 148, 196
testowanie hipotez, 59, 67, 147,
 196
typy
 danych, 107
 obiektów, 165
 zmiennych, 26

U

uczenie maszynowe, 88, 202

V

VBA, Visual Basic for
 Applications, 91

W

walidacja, 151, 198
wariancja, 31
wartości odstające, 38
wartość
 krytyczna, 63
 oczekiwana, 52
wektory, 112
 indeksowanie, 114
 wybór elementów, 114

wiersze, 20, 92
wizualizacja danych, 137, 186
wnioskowanie statystyczne, 54
współczynnik
 kierunkowy, 77
 korelacji, 73, 84, 197
 korelacji Pearsona, 72
 R-kwadrat, 83, 153
wybieranie elementów
 tablice, 171
wybór elementów
 wektory, 114
wykres
 fasetowy, 145
 przetawny, 36
 pudełkowy, 38, 39, 139, 140,
 189
 punktowy, 72, 191
 rozrzutu, 72, 75, 79, 141, 149,
 199
 Skrzynka i wąsy, 38
 słupkowy, 29
 typu pairplot, 198

Z

zakres, 31
zarządzanie pakietami, 166, 167
zbiór
 danych, 20
 testowy, 151, 198
 treningowy, 198
 uczący, 151
 wektorów, 115
zintegrowane środowisko
 programistyczne, IDE, 99
złączenie lewostronne, 133
zmiennie, 21, 92, 106
 ilościowe, 23, 29
 kategorialne, 22, 27
zmiennie
 niezależne, 57
 zależne, 57
zmiennność, 83
znak równości, 164

PROGRAM PARTNERSKI

— GRUPY HELION —

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Eksploracyjna analiza danych? I w Excelu, i w Pythonie!

Sukces przedsiębiorstwa zależy od jakości podejmowanych decyzji. Spośród strategii, które wspierają ten proces, na szczególną uwagę zasługuje zastosowanie analizy danych. Jest to jednak dość złożona dziedzina. Podstawowym narzędziem wielu analityków danych jest arkusz kalkulacyjny. Ma on tę zaletę, że ułatwia solidne zrozumienie prawideł statystyki i analizy danych. Po zdobyciu takich podstaw warto jednak pójść dalej i nauczyć się eksploracyjnej analizy danych za pomocą języków programowania.

Dzięki tej książce przejście od pracy z arkuszami Excela do samodzielnego tworzenia kodu w Pythonie i R będzie płynne i bezproblemowe. Rozpoczniesz od ugruntowania swoich umiejętności w Excelu i dogłębnego zrozumienia podstaw statystyki i analizy danych. Ułatwi Ci to rozpoczęcie pisania kodu w języku R i w Pythonie. Dowiesz się, jak dokładnie przebiega proces oczyszczania danych i ich analizy w kodzie napisanym w języku R. Następnie zajmiesz się poznawaniem Pythona. Jest to wszechstronny, łatwy w nauce i potężny język programowania, ulubiony język naukowców i... analityków danych. Nauczysz się płynnego przenoszenia danych z Excela do programu napisanego w Pythonie, a także praktycznych metod ich analizy. Dzięki ćwiczeniom, które znajdziesz w końcowej części każdego rozdziału, utrwalisz i lepiej zrozumiesz prezentowane treści.

W książce:

- badanie relacji między danymi za pomocą Excela
- stosowanie Excela w analizach statystycznych i badaniu danych
- podstawy języka R
- proces oczyszczania i analizy danych w R
- przenoszenie danych z Excela do kodu Pythona
- pełna analiza danych w Pythonie

George Mount założył i prowadzi Stringfest Analytics, firmę konsultingową specjalizującą się w analizie danych. Współpracował z wiodącymi bootcampami, platformami edukacyjnymi i organizacjami. Regularnie wypowiada się na tematy dotyczące nauki i analizy danych, a także rozwoju pracowników. Mieszka w Cleveland w stanie Ohio.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

Sprawdź nasze szkolenia!
SZKOLENIA
AKADEMIA IT & BUSINESS
WWW.SZKOLENIA.HELION.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶
ISBN 978-83-283-8551-1
9 788328 385511

