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

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.

Spis treści

ROZDZIAŁ 8.

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 chło… tak      czar…        19
#> 4        559         448 standardowa.…         16 chło… nie      biała        69
#> 5        489         447 mała.klasa             5 chło… tak      biała        79
#> 6        454          431 standardowa.…         8 chło… 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
`select()`	Wybór określonych kolumn
`mutate()`	Tworzenie nowych kolumn na podstawie istniejących
`rename()`	Zmiana nazw kolumn
`arrange()`	Zmiana kolejności wierszy na podstawie określonych kryteriów
`filter()`	Wybór wierszy na podstawie określonych kryteriów
`group_by()`	Grupowanie według wybranych kolumn
`summarize()`	Agregacja wartości w grupach
`left_join()`	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 Grolemunda *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.klasy        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 logarytmicznego 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.klasy          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.klasy`, 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> <chr>                          <dbl>        <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:

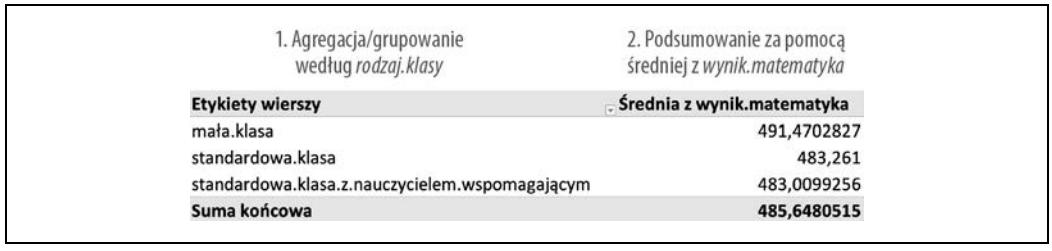
```
filter(star, wynik.czytanie >= 500)
#> # A tibble: 233 x 5
#>    wynik.matematyka wynik.czytanie rodzaj.klasy         doświadczenie… łączny.wynik
#>               <dbl>          <dbl> <chr>                       <dbl>        <dbl>
#>  1              559            522 standardowa.klasa               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*).



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
`sum()`	Suma
`n()`	Liczba wartości
`mean()`	Średnia
`max()`	Maksimum
`min()`	Minimum
`sd()`	Odchylenie standardowe

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

```
summarize(star_grouped, średnia.z.matematyki = mean(wynik.matematyka))
#>  A tibble: 3 x 2
#>   rodzaj.klasy                                    średnia.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  doświadcz… płeć  darmowe.wyż… 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 chło… tak          czar…        19
#> 4        559         448 standardowa.…         16 chło… nie          biała        69
#> 5        489         447 mała.klasa             5 chło… tak          biała        79
#> 6        454         431 standardowa.…          8 chło… tak          biała         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> <chr>           <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 ciągłego 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.
```

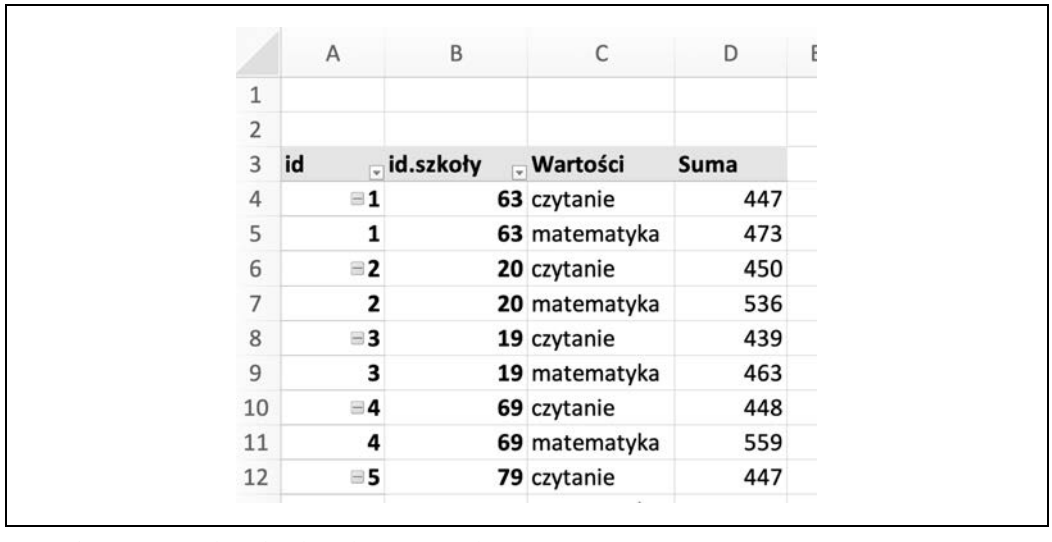
Brak konieczności uwzględniania danych w wywołaniu każdej funkcji może być na początku trochę dezorientujący, ale porównaj ostatni listing z poprzednim, a zobaczysz, o ile wydajniejsze może być to podejście. Co więcej, operator potoku może być używany również z funkcjami spoza pakietu *dplyr*. Jako przykład 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
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` określę, 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
`data`	Źródło danych
`aes`	Odwzorowanie danych na elementy wizualne (osie *x* i *y*, kolor, rozmiar itd.)
`geom`	Typ obiektu geometrycznego obserwowany na wykresie (linie, słupki, kropki itd.)

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

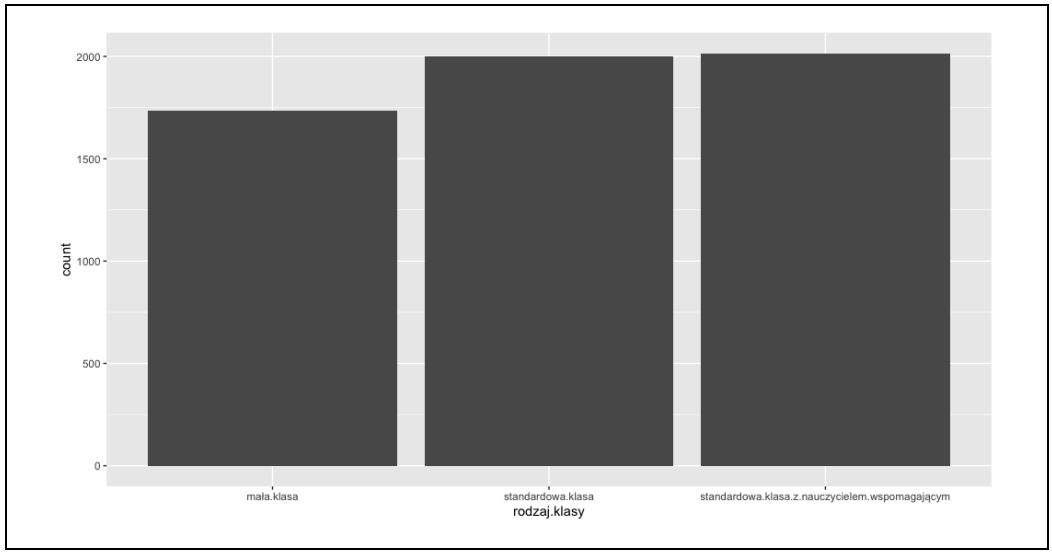
```
ggplot(data = star, ❶
       aes(x = rodzaj.klasy))+ ❷
   geom_bar() ❸
```

❶ W argumencie `data` określa się źródło danych.

❷ Odwzorowanie danych na elementy wizualne określa się za pomocą funkcji `aes()`. W tym miejscu odwzorowujemy kolumnę `rodzaj.klasy` w oś *x* wynikowego wykresu.

❸ Znając dane i ich odwzorowanie na elementy, za pomocą funkcji `geom_bar()` na wykresie umieszczamy wybrany obiekt geometryczny. Wyniki przedstawiono na rysunku 8.4.

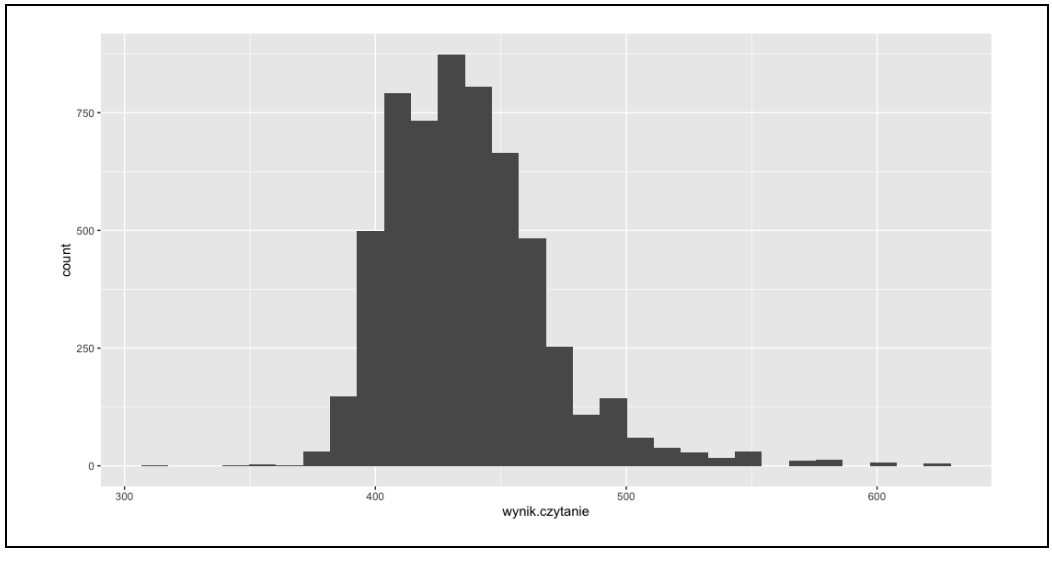
Tak jak w przypadku operatora potoku, zapisywanie każdego działania w osobnej linii nie jest konieczne, ale często preferowane ze względu na czytelność. Możliwe jest również uruchomienie całego kodu tworzącego wykres poprzez umieszczenie kursora w dowolnym miejscu bloku kodu i uruchomienie wykonania.



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:

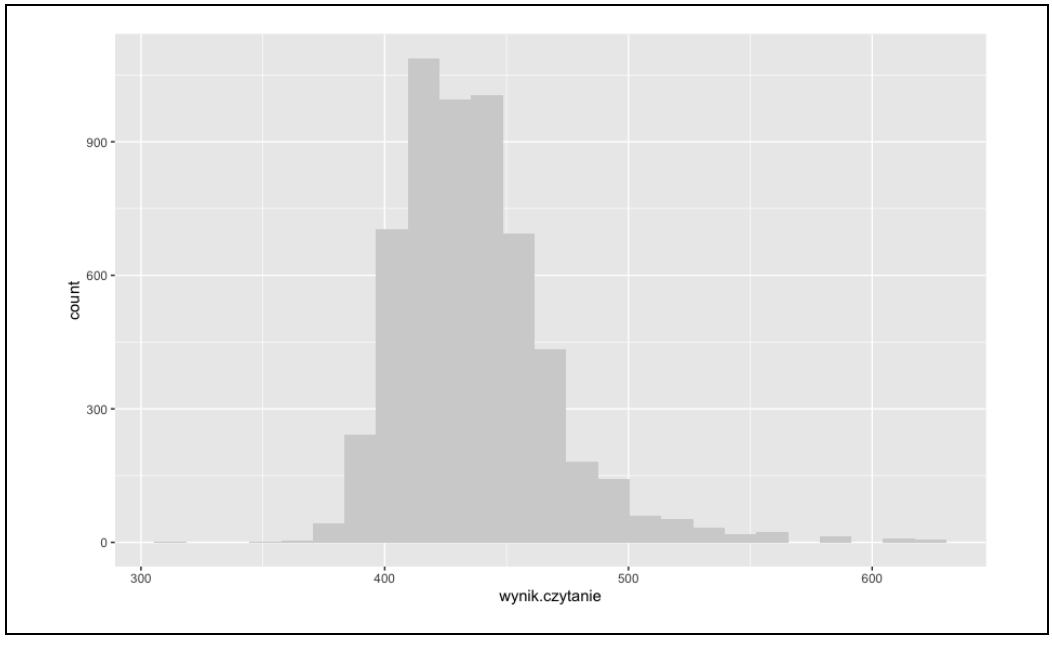
```
ggplot(data = star,aes(x = wynik.czytanie))+
  geom_histogram()

#> `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:

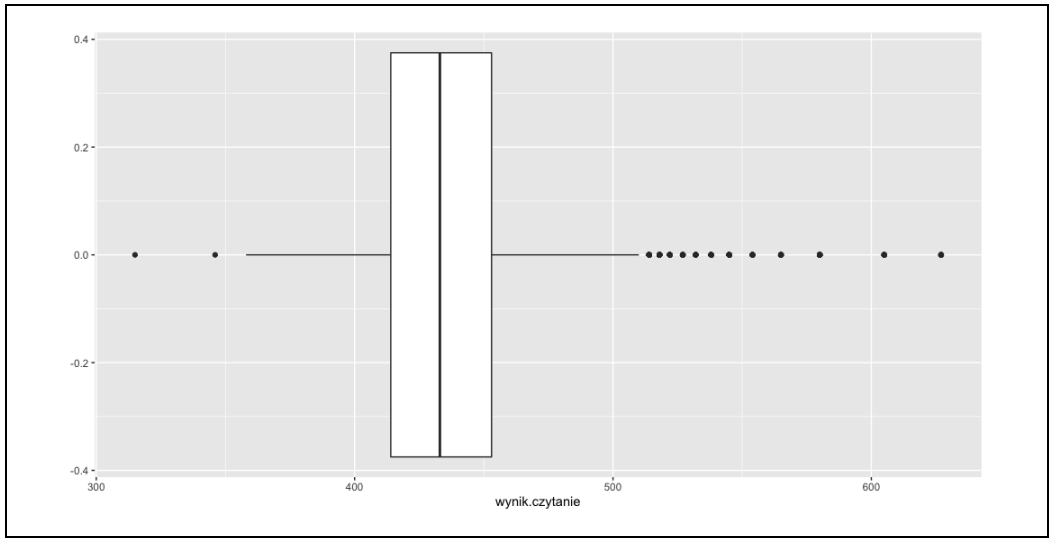
```
ggplot(data = star,aes(x = wynik.czytanie))+
  geom_boxplot()
```

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:

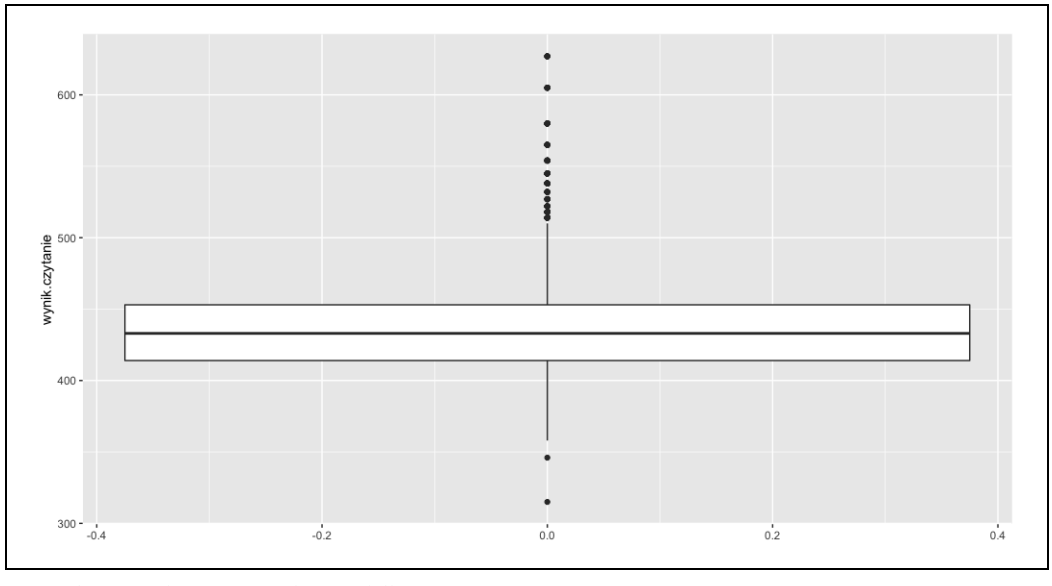
```
ggplot(data = star, aes(y = wynik.czytanie))+
  geom_boxplot()
```

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()
```



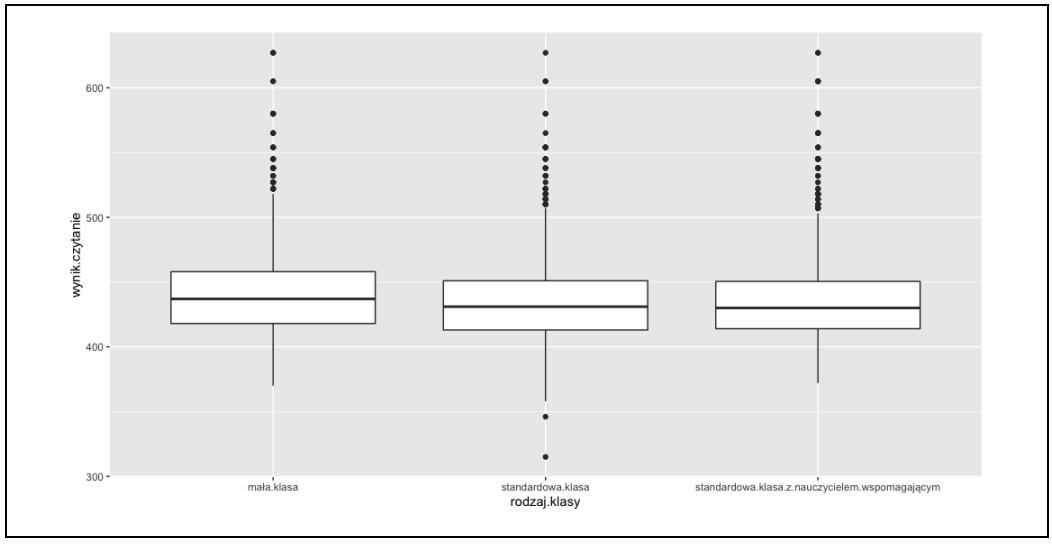
Rysunek 8.7. Wykres pudełkowy



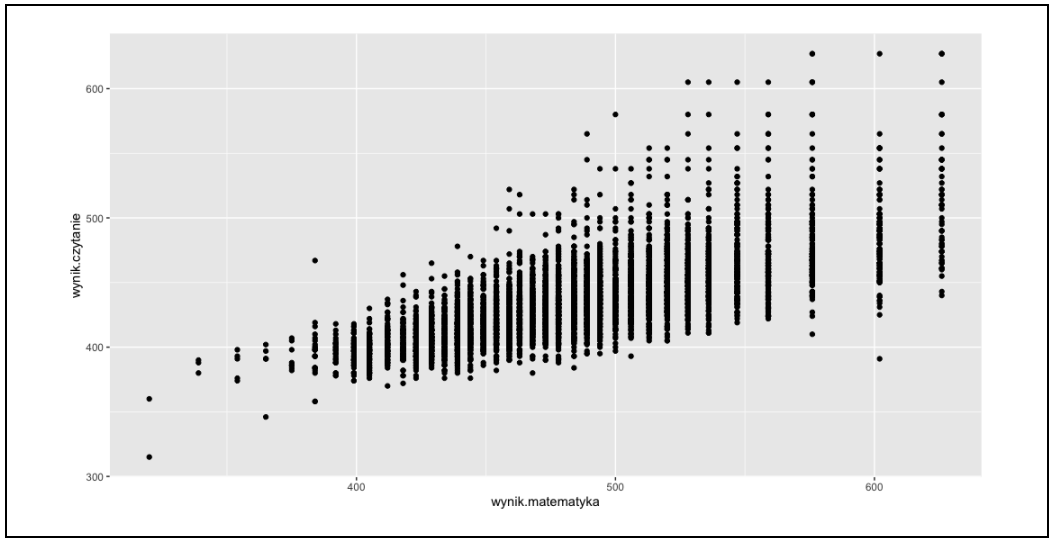
Rysunek 8.8. „Odwrócony" wykres pudełkowy

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()
```



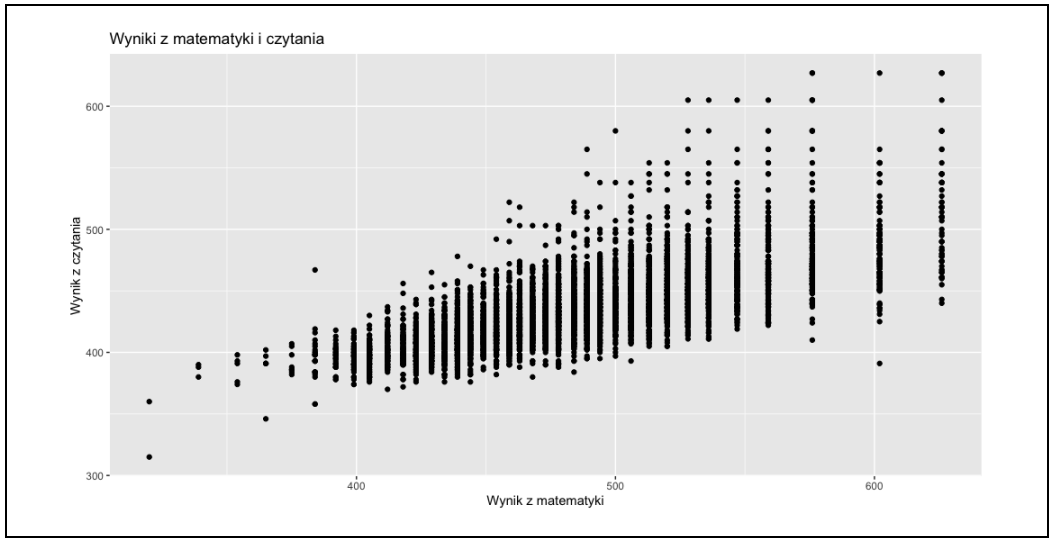
Rysunek 8.9. Wykres pudełkowy z podziałem na grupy



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 *cenzus*, znajdziesz pliki *cenzus.csv* i *cenzus-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

N

O

P

R

S

Ś

T

U

V

W

Z

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

ISBN 978-83-283-8551-1
9 788328 385511

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 59,90 zł