STATYSTYKA OPISOWA Z PAKIETEM R

Robert Kapłon

ver. 1.0

(kompilacja: 8 kwietnia 2021)

Spis treści

Wstęp

Krótkie wprowadzenie do pakietu R ma ci pomóc w poznaniu tego fantastycznego środowiska. Materiałów do nauki R nie brakuje. Znajdziesz bardzo wiele pozycji książkowych, tutoriali czy nawet filmów instruktażowych. Jednak problem osoby początkującej zawsze sprowadza się do pytania: co wybrać. I właśnie dlatego napisałem to wprowadzenie specjalnie dla ciebie. Jeżeli opanujesz umiejętności na jego poziomie, to z powodzeniem wykonasz nawet złożone analizy. Założyłem tym samym, powiesz później czy słusznie, że ma być ono twoim jedynym źródłem wiedzy na tym przedmiocie. Jeżeli jakiś treści nie rozumiesz i musisz szukać pomocy np. w Internecie, to dla mnie jest to sygnał, że ten fragment lub fragmenty muszę jeszcze dokładniej opisać. Nie zwlekaj i napisz mi o tym: `robert.kaplon@pwr.edu.pl`.

Mniej świadomość, że wprowadzenie nie aspiruje do miana kompletnego i wyczerpującego. Wiele istotnych treści pominąłem, ze wszech miar celowo, włączając w to nawet podstawowe zagadnienia. Nie chcę wywołać u ciebie efektu przytłoczenia. Oczywiście, jak skończysz pracę z tym wprowadzeniem, możesz sięgnąć do szerszych opracowań.

Pisząc, przyjąłem pewną konwencję omówienia zagadnień. Zawsze ilustruję je fragmentami kodu oraz wynikami jego uruchomienia. W wielu miejscach, a szczególnie przy wizualizacji danych, zamieszczam przykłady różniące się nieznacznie. Jeżeli je przeanalizujesz, szybko dostrzeżesz różnice między użytymi funkcjami i argumentami. Czasami takie podejście uważam za lepsze niż szczegółowy opis. Poza tym nie traktuj wykresów jako takich, które przygotowałem zgodnie z zasadami.

W tekście znajdziesz też 3 rodzaje ramek, a na końcu indeks użytych funkcji. Poniżej przeczysz, co zawiera każda z ramek.

Zapamiętaj 0.1

W takiej ramce zamieszczam informacje o szczególnej ważności. Myślałem nawet, aby nazwać ją tak z przymrużeniem oka: bez tego nie zaliczysz. Ale to sugeruje przymus i uczenie się dla oceny. Mam nadzieję, że jednak chcesz się nauczyć czegoś nowego, czegoś co może ci się przydać w późniejszej pracy zawodowej.

Strefa Eksperta 0.1

To ramka jest przeznaczona dla osób, które chcą się dowiedzieć czegoś nadprogramowo. Dlatego jest nieobowiązkowa.

> **WARTO WIEDZIEĆ**
>
> Wiedza jaką się dzielę za pośrednictwem tej ramki jest ważna, potrzebna i na pewno nie można zaliczyć jej do wiedzy ezoterycznej. Jeśli chcesz opanować R przynajmniej na dobrym poziomie zapoznaj się z nią.

Jaką strategię pracy z wprowadzeniem przyjąć. Musze to jasno napisać: samo czytanie – bez aktywnego, równoczesnego używania programu – jest niewystarczające do zdobycia biegłości w posługiwaniu się R. Dlatego gorąco cię zachęcam do przepisywania zamieszczonych tutaj

fragmentów programów, a nie ich kopiowania i wklejania. Wpisując kilka razy tą samą funkcję, po prostu ją zapamiętasz. Polecam ci również przeprowadzanie eksperymentów, gdy podczas lektury nasuną się wątpliwości. Zadawaj pytania (a co by było gdyby), zmieniaj kod i testuj.

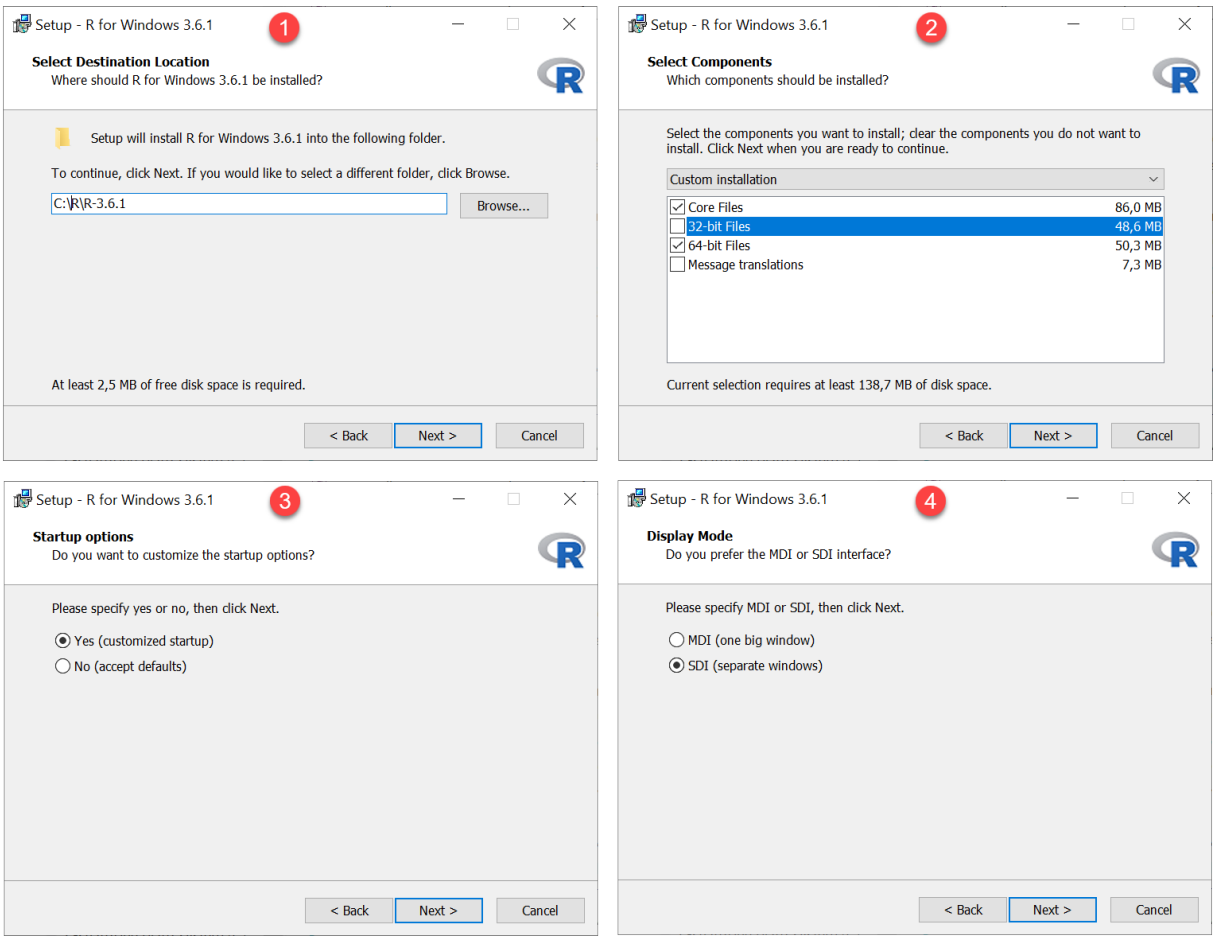
W opracowaniu wykorzystuję różne zbiory danych, które znajdują się w katalogu dane.

ROZDZIAŁ 1

Przygotowanie do pracy w środowisku R

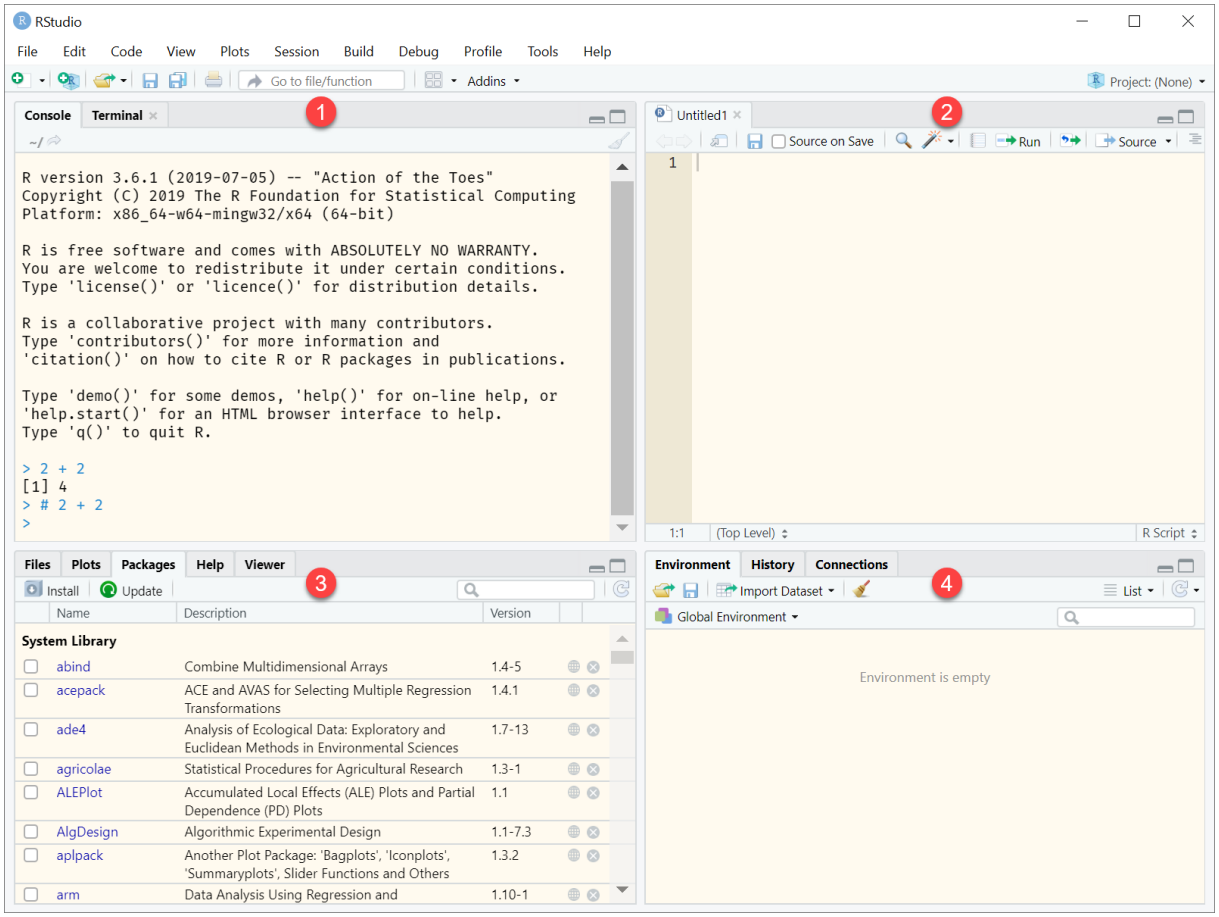
1.1. Instalacja i konfiguracja

Przygotowania rozpoczynamy od ściągnięcia i zainstalowania środowiska R. Odwiedź stronę projektu R: `http://www.r-project.org`. Z menu po lewej stronie wybierz CRAN oraz lokalizację serwera. Teoretycznie każdy URL powinien zawierać tą samą wersję programu. Pamiętaj o platformie, na której R będzie instalowany. Jeżeli wybierzesz *Windows*, to później kliknij na *base* i pobierz plik. Utwórz katalog `R` na dysku `C` i zainstaluj tam program. Dalsze kroki zamieszczam na poniższym rysunku. Zobacz, że rysunki wskazują na starszą wersję `R`. To wprowadzenie bazuje już na wersji 4.0.4.



Od pewnego czasu komunikaty w konsoli R są w j. polskim, z kolei pomoc, dokumentacja, tutoriale itp. są w j. angielskim. Taka dwujęzyczność przeszkadza, dlatego będziemy pracować z wersją angielską. Zmian dokonasz, edytując w notatniku plik `Rconsole` – najedź na plik i prawym przyciskiem myszki wybierz: otwórz za pomocą, wskazując np. notatnik (polecam Notepad++). Plik znajduje się w katalogu `etc`. U mnie pełna ścieżka to: C:\R\R-4.0.4\etc\. Gdy już otworzysz plik, odszukaj wiersz odnoszący się do definicji języka i dopisz `en`; tak powinien wyglądać uzupełniony wiersz: `language = en`.

Do tworzenia i edytowania skryptów R wykorzystamy darmowe, zintegrowane środowisko **RStudio**: `https://www.rstudio.com/products/rstudio/download/preview/`. Zainstaluj go w domyślnej lokalizacji – od tej pory będziemy uruchamiać tylko **RStudio**, którego wygląd widzisz na rysunku.



Całe okno **RStudio** podzielone jest na 4 panele główne, z których każdy ma jeszcze kilka zakładek. Położenie paneli względem siebie możesz skonfigurować, wybierając z menu: *Tools | Global Options*. Tam też znajdziesz wszystkie ustawienia. Polecam zmienić następujące:

- **General | Basic** – w *Workspace* odznacz opcję: *Restore .RData into workspace at startup*. Dla *Save workspace to .RData on exit* wybierz *Never*. Jeśli opcja jest zaznaczona (*Always*), wtedy przy każdym uruchomieniu **RStudio** ładuje do pamięci wszystkie obiekty (zawartość całego środowiska) poprzedniej sesji. Jest to źródłem wielu problemów początkujących użytkowników.
- **Code | Display** – zaznacz *Show margin* i ustaw na 100. Zaznacz też obie opcje: *Show syntax highlighting in console input* oraz *Highlight R function calls*.

W **Oknie 1** widzisz konsolę R. Pozwala ona na interaktywną pracę. Znak większości > oznacza gotowość programu. Wpisz niej `2+2` i naciśnij Enter. Nastąpi natychmiastowa interpretacja i zobaczysz wynik. Jeśli dowolny zapis poprzedzisz znakiem #, to poinformujesz program, że ma go traktować jak komentarz – spróbuj wpisać: `#2+2`. Tryb interaktywny jest użyteczny wtedy, gdy chcesz coś szybko obliczyć lub sprawdzić, czy składnia jest poprawna. Musisz jeszcze wiedzieć, że oprócz widocznego znaku > czasami będzie pojawiał się znak plusa. Program wtedy czeka na dokończenie składni. Wykonaj działanie odejmowania dwóch liczb (5 – 4), ale w następujących krokach: wpisz `5 -`, naciśnij enter, a następnie wpisz 4. Gdy chcesz wyjść z trybu oczekiwania, naciśnij klawisz `Esc`.

Tryb interaktywny nie sprawdza się nawet w wypadku bardzo krótkich analizy. Dlatego cały

skrypt z naszą analizą będziemy tworzyć w **Oknie 2**. Więcej o tworzeniu, edycji i uruchamianiu skryptów znajdziesz w rozdziale 1.2, w którym nauczysz się tworzyć projekty.

W **Oknie 3** mamy kilka zakładek. Omówię dwie: pakiety (*packages*) i pomoc (*help*). Pakiety są bardzo mocną stroną R. Tworzą je zarówno profesjonaliści jak i entuzjaści języka. Pakiety zapewniają funkcjonalność niespotykaną w innych programach do analiz statystycznych. Wyobraź sobie, że wiele pomysłów na analizę danych, które być może przyjdą ci do głowy, zostało już zaimplementowanych właśnie w postaci pakietów. Mamy do dyspozycji 17370 pakietów, które są dostępne w repozytorium CRAN na stronie R. Jeśli chcesz z któregoś skorzystać, to musisz go najpierw zainstalować. Z zakładki *Packages* wybierasz *Install*, a następnie wpisujesz nazwy interesujących cię pakietów. Spróbuj wpisać pakiet *tidyverse*, który będziemy używać bardzo często.

Zapamiętaj 1.1

- Pakiety instalujesz tylko raz – podobnie jak każdy program.
- Jeśli chcesz użyć pakietu, wtedy przy każdym uruchomieniu programu **RStudio** musisz go wczytać za pomocą funkcji: `library("tutaj_nazwa_pakietu")`.

Zazwyczaj pakiety ładujemy na samym początku pisanego skryptu. Choć możesz też wpisać to w konsoli, jednak takiego podejścia nie polecam. W poniższym przykładzie wczytuję pakiet `tidyverse`.

```
> library(tidyverse) # Wczytujesz każdorazowo
```

Przejdę teraz do omówienia pomocy, którą widzisz w jednej z zakładek **Okna 3**. Możesz tę pomoc przeglądać albo w okienku wyszukiwarki wpisać nazwę funkcji czy też frazę. Przyznam, że szukając informacji postępuję trochę inaczej i wydaje mi się, że szybciej. Wpisuję w konsoli nazwę interesującej mnie funkcji, poprzedzając ją znakiem zapytania. Wtedy automatycznie, w oknie pomocy, pojawia się pełna informacja. Spróbuj wpisać: `?log`. Jeśli nie znasz dokładnej nazwy funkcji, a chcesz wyszukać wszystko, co dotyczy pewnej frazy, użyj podwójnego pytajnika `??`. Może się zdarzyć, że we frazie pojawia się spacja, wtedy użyj cudzysłowów, np. `??"log normal"`.

Pomocy możesz również poszukiwać w dokumentacji i opracowaniach dostępnych na stronach:

- Manuale: `http://cran.r-project.org/manuals.html`
- Pozostałe dokumenty: `http://cran.r-project.org/other-docs.html`
- Tytuły książek: `http://www.r-project.org/doc/bib/R-books.html`

W **Oknie 4** widzisz zawartość tzw. środowiska (*Environment*). Składają się na niego obiekty, które tworzysz w ramach sesji R. Oprócz listy z nazwami tych obiektów, masz informację o ich rodzaju i rozmiarze. Jeżeli klikniesz na obiekt o strukturze tablicy, to pojawi się podgląd zawartości w postaci tabeli, jaką znasz z arkuszy kalkulacyjnych.

W tej części użyłem dwóch bardzo ważnych słów, które nie zdefiniowałem – obiekt i funkcja. Zapamiętaj dwie fundamentalne reguły R.

Zapamiętaj 1.2

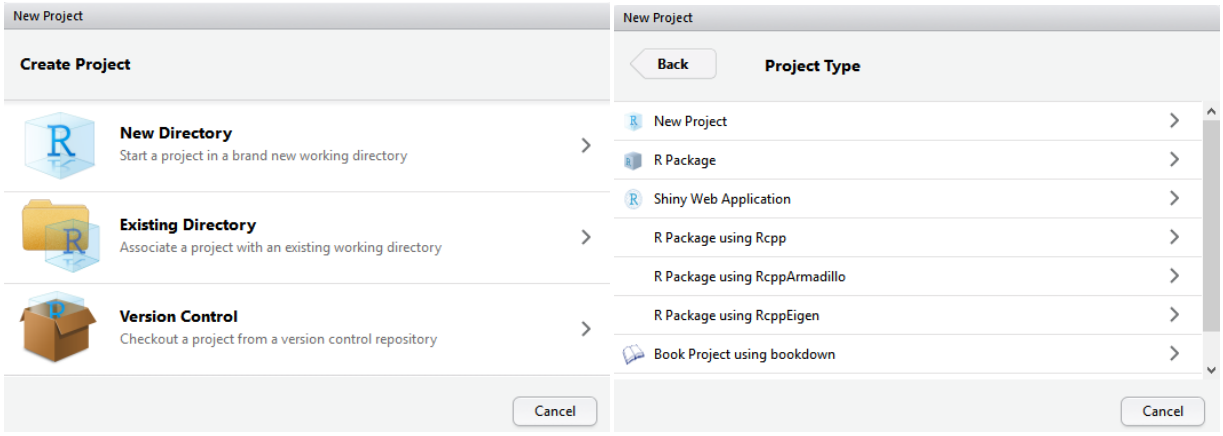
Dwa fundamenty R

- Wszystko w R jest obiektem.
- Wszystko co dzieje się w R, jest następstwem wywołania funkcji.

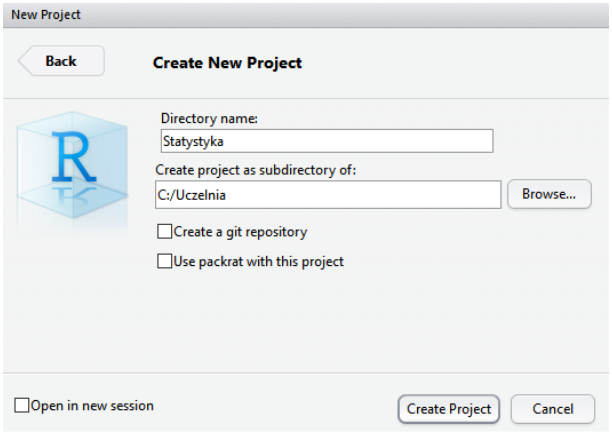
1.2. Tworzenie projektu

Najlepszym sposobem organizacji pracy w **RStudio** są projekty. Projekt jest tak naprawdę katalogiem, w którym przechowujemy pliki i inne katalogi. Pozwala nam to uporządkować pracę. Możemy utworzyć tam katalog o nazwie `dane` i przechowywać w nim wszystkie zbiory danych, które wykorzystujemy w analizie. W innym katalogu możemy przechowywać pliki graficzne wykresów, a w kolejnym raporty i prezentacje. W konsekwencji tzw. przestrzenią roboczą jest katalog projektu, a nie domyślnie ustalona przez **R** lokalizacja na dysku. Zapytaj program o tę lokalizację wpisując w konsoli `getwd()`.

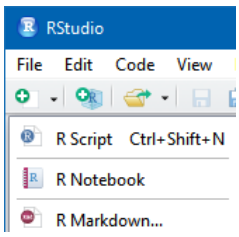
Stwórzmy przykładowy projekt i nazwijmy go: *Statystyka*. Z menu **File | New Project** wybierz: *New Directory*, a następnie *New Project*. Ten etap prezentuję na poniższych oknach.



W kolejnym kroku podaj nazwę projektu (*Statystyka*) i wybierz jego lokalizację na dysku, np. `"C:/Uczelnia"`.



Od teraz całą zawartość katalogu *Statystyka* możesz przenosić między komputerami (sprawdź ponownie lokalizację przestrzeni roboczej). Aby uruchomić **RStudio** i otworzyć projekt, wystarczy że klikniesz na plik `Statystyka.Rproj`, który znajduje się w tym katalogu. Nie ma w nim chyba najważniejszych plików – skryptów z twoimi programami. Aby taki utworzyć, kliknij na ikonkę zielonego plusa i wybierz `R Script`, jak podpowiada ci poniższy rysunek.



Plik zapisz pamiętając o rozszerzeniu: po kropce dodaj dużą literę R. Zauważ, że domyślna lokalizacja zapisu to właśnie twój projekt. Ja utworzyłem plik o nazwie: `zad_rozdz1.R`. W zależności od potrzeby możesz utworzyć wiele takich plików. Od tej pory każdą, nawet najmniejszą analizę, będziemy zapisywać w edytorze plików (**Okno 2**).

W edytorze plików możesz wpisać cokolwiek i dopóki nie prześlesz tego do konsoli, edytor jest zwykłym notatnikiem. Zatem napisz `2+2` i użyj kombinacji klawiszy: `CTRL+Enter`. To ona przekazuje całą aktywną linię do konsoli – nie ma znaczenia, w której kolumnie jest kursor. Jeśli natomiast chcesz przekazać więcej linii, to zaznacz je i wtedy użyj tego skrótu. Z innymi, dostępnymi skrótami klawiaturowymi możesz zapoznać się wybierając z menu `Help` i `Keyboard Shortcuts`.

Zapamiętaj 1.3

Skróty klawiaturowe

- `Ctrl+ENTER` – wiersz edytora skryptów jest przekazywany do konsoli, a następnie wykonywane są zawarte w nim polecenia. Działa również, jeśli zaznaczysz kilka wierszy.
- ↑/↓ – użycie strzałek w konsoli przywołuje wcześniej napisane komendy.

ROZDZIAŁ 2

Struktury danych

2.1. Typy fundamentalne

Liczby są fundamentalnymi obiektami w R. Zapewne znasz podział na liczby: naturalne, całkowite, wymierne, niewymierne, rzeczywiste i zespolone. W pracy ograniczymy się do **typu całkowitego** (*integer*) oraz **rzeczywistego** (*double*). Zapamiętaj, że w liczbach rzeczywistych będziemy oddzielać kropką części całkowite od dziesiętnych, np. 2.77. Muszę wspomnieć również o **typie liczbowym** (*numeric*), który odnosi się do dwóch pierwszych typów. W statystyce **zmienne ilościowe** będą jednym z tych typów, np. wynagrodzenie, wiek, liczba dzieci w gospodarstwie domowym.

Kolejnym obiektem podstawowym są łańcuchy znaków, zwane również napisami, o których mówimy, że są **typu znakowego**. Wykorzystujemy je do przechowywania informacji tekstowych, które ujmujemy w cudzysłowy, np. `"a"` czy `"zgadzam się"`. Ten typ, w statystyce, odpowiada **zmiennym kategorialnym**, ale mierzonym na skali nominalnej. Bardziej złożonym typem, który pozwala na uwzględnienie zarówno skali nominalnej jak i porządkowej jest typ **czynnikowy** (*factor*). Jednak nie jest on typem fundamentalnym.

Pracując z danymi musisz mieć pewność, z jakim typem masz do czynienia. Od tego zależy, jakie statystyki możesz policzyć. Bardzo często R poinformuje cię o tym, wypisując angielską nazwę (wcześniej umieściłem je w nawiasie). Możemy też bezpośrednio zapytać o to, używając funkcji `typeof()` lub `mode()`. Skopiuj poniższa linijkę i uruchom. Pamiętaj, aby z kopiowanego fragmentu usunąć znak >.

```
> c(typeof(7L), typeof(7), mode(7), typeof(1.5), typeof("b. dobrze"))
[1] "integer"   "double"    "numeric"   "double"    "character"
```

Zapewne winny jestem ci wyjaśnienia, bo przecież liczba 7 jest całkowita, a R napisał, że jest rzeczywista (*double*). Okazuje się, że bardzo często liczby które widzimy jako całkowite, R zapisuje i traktuje jak liczby rzeczywiste – robi to celowo. Jeżeli chcesz mieć liczbę całkowitą, to dodaj dużą literę L na końcu jak w powyższym przykładzie.

Chociaż obiekt typu **logicznego** nie ma swojego odpowiednika w statystyce, to jednak muszę o nim wspomnieć. Obiekt taki może przechowywać jedną z dwóch wartości: prawda albo fałsz. W programie za te dwa stany odpowiadają słowa: `TRUE` i `FALSE`. Ponieważ w R wielkość liter ma znaczenie, więc nie zostaną rozpoznane takie słowa jak `true` czy `True` – znane z innych języków programowania. Zapewne wiesz, że określenia słowne (polskie, angielskie) przyjmowane są dla naszej wygody, gdyż komputer przechowuje liczby: 0 i 1 odpowiednio dla fałszu i prawdy. Zapytam z przymrużeniem oka: jeżeli dodasz 3 prawdy, to co otrzymasz? Nasz program jest nad wyraz konsekwentny:

```
> TRUE + TRUE + TRUE
[1] 3
```

Naszą listę musimy jeszcze uzupełnić o **wartości i symbole specjalne**. Na pewno spotkasz:

- `Inf` – wartość nieskończona
- `NaN` (*not a number*) – wyrażenie nieoznaczone, np. wykonując działanie $\frac{0}{0}$)
- `NA` (*not available*) – jeśli mamy do czynienia z brakującymi danymi, wtedy ten fakt

zostanie odnotowany przez NA

Przeanalizuj poniższe przykłady, aby zobaczyć, w jakich sytuacjach wartości i symbole mogą się pojawić. Zapewne zauważasz, że kolejny raz użyłem funkcji `c()` złączającej elementy. Co więcej powstały obiekt nazwałem wektorem. Chyba najwyższa pora, abyśmy przeszli do jego opisu. Obiecuję, że już w podrozdziale 2.3 skupimy się na wektorach. Musimy jeszcze zapoznać się z bardzo ważnym zagadnieniem tworzenia obiektów nazwanych, czemu poświęcimy kolejny podrozdział.

```
> log(0) # oblicz logarytm naturalny z 0
[1] -Inf

> sqrt(-1) # (dla liczb rzeczywistych nie istnieje)
[1] NaN

> c(3, 5, 9, NA) # ostatni element wektora traktowany jako brak danych
[1]  3  5  9 NA
```

2.2. Tworzenie obiektów nazwanych

Do tej pory wpisywaliśmy wyrażenia (w konsoli lub edytorze), które następnie były przetwarzane przez R, a wynik wyświetlany w konsoli, np.

```
> 2 + log(5)
[1] 3.60944
```

Jeżeli wartość tego wyrażenia chcesz wykorzystać później, w jakimś innym miejscu swojej analizy, to musisz go zapisać w pamięci komputera. Aby to zrobić, utwórz **obiekt nazwany** w dwóch krokach: wybierz nazwę, a następnie wykorzystując **operator przypisania** <-, przypisz tej nazwie obiekt. Zamiast operatora przypisania możesz użyć znaku =. Nie polecam takiej zamiany, gdyż prawie wszyscy – z historycznych względów – używają operatora przypisania. Im szybciej się do niego przyzwyczaisz, tym łatwiejszy w interpretacji będzie kod. Wybrałem nazwę `suma` dla obiektu i wpisałem w konsoli:

```
> suma <- 2 + log(5)
```

Być może zastanawiasz się, dlaczego nie widzisz wyniku (jak wcześniej). Otóż R domyślnie nie wyświetla wartości przypisanej do `suma`. Trzeba go do tego zmusić, wpisując nazwę. Możemy tak zrobić, gdyż wynik został zapamiętany w pamięci:

```
> suma
[1] 3.60944
```

Powiem ci jeszcze o jednym sposobie wyświetlania obiektów. Ujmij całe wyrażenie w nawisy okrągłe (`(suma <- 2 + log(5))`). Przyznam, że podczas pisania programów raczej z tego nie korzystam, chociaż tutaj od tego nie stronię. Przyświecał mi jeden cel – oszczędność miejsca.

Powinienem poruszyć kwestię wyboru nazwy dla obiektu, gdyż nie ma tutaj pełnej dowolności. W R nazwy mogą składać się z ciągu liter, cyfr, kropki, podkreślenia. Nie możesz nazw zaczynać od cyfr, używać znaków specjalnych (np. %, #) oraz słów kluczowych języka R (np. `if`, `else`, `TRUE`, `FALSE`, `NA`). Przykładem poprawnych nazw są: `grupaWiek`, `grupa.wiek`, `grupa_wiek`, `GrupaWiek`. Jak już wiesz, w R jest istotna wielkość liter, dlatego nazwa pierwsza nie jest równoważna nazwie ostatniej.

2.3. Wektory

Wektor jest ciągiem elementów tego samego typu. Dlatego tworząc wektor, nie możesz typów mieszać. Zapewne pamiętasz z podrozdziału 2.1, że do wyboru masz typ: liczbowy, znakowy, czynnikowy, logiczny. Jeśli nie zastosujesz się do tego wymogu i zechcesz utworzyć wektor składający się z liczby 5.7 i słowa `"czytaj"`, wtedy R dokona tzw. koercji, czyli ujednolicenia typu. W konsekwencji liczba 5.7 zostanie potraktowana jako łańcuch znaków, a wektor będzie typu znakowego.

2.3.1. Tworzenie wektorów.

Wektory w R utworzysz na wiele sposobów. My skupimy się na dwóch:

1. `c()` – funkcja łącząca (*combine*) elementy, np. `c(4, 5, 3.7)`;
2. `:` – operator tworzący ciąg arytmetyczny o różnicy 1, np. `2:5` tworzy wektor: `(2, 3, 4, 5)`

Przeanalizuj poniższe przykłady.

```
> ## 1. Funkcja c() i wektory rożnych typów
> c(1, 5, 6, -2.34) # typ liczbowy
[1]  1.00  5.00  6.00 -2.34

> c("zdecydowanie", "raczej nie", "trudno sie zdecydowac") # typ znakowy
[1] "zdecydowanie"          "raczej nie"            "trudno sie zdecydowac"

> c(TRUE, FALSE, FALSE) # typ logiczny
[1]  TRUE FALSE FALSE

> c(1, 7, "zarobki") # niedozwolone mieszanie typów: R uzgodni typ (będzie to znakowy)
[1] "1"       "7"       "zarobki"

> ## 2. Operator :
> 3:10 # utwórz wektor od 3 do 10
[1]  3  4  5  6  7  8  9 10

> c(3:10) # to samo co wyżej, ale tak nie piszemy: c() nie ma uzasadnienia
[1]  3  4  5  6  7  8  9 10

> 7:2
[1] 7 6 5 4 3 2

> c(3:10, 4:2, -3) # łączymy 3 wektory; tak, liczba jest wektorem
 [1]  3  4  5  6  7  8  9 10  4  3  2 -3
```

2.3.2. Operacje na wektorach.

W R masz do dyspozycji następujące operatory:

- operatory arytmetyczne: `+`, `-`, `*`, `/`, `^`, (dodawanie, odejmowanie, mnożenie, dzielenie, potęgowanie); przykładowo:

  `x+y`, `x-y`, `x*y`, `x/y`, `x^y`

- operatory logiczne: `!`, `|`, `&` (negacja, alternatywa, koniunkcja); alternatywę i koniunkcję dla skalarów przedstawiają odpowiednio symbole: `||`, `&&`; przykładowo:

  `!x`, `x|y`, `x&y`, gdy skalary: `x||y`, `x&&y`

- operatory relacyjne: `>`, `<`, `>=`, `<=`, `==`, `!=` (większy, mniejszy, większy bądź równy, mniejszy bądź równy, równy, różny); przykłady

```
x > y,  x < y,  x >= y,  x <= y,  x == y,  x != y
```

- operator binarny `%in%`, który oznacza *należy* i zastępuje matematyczny symbol $\in$; przykład

```
x %in% y
```

Zacznijmy od **operatorów arytmetycznych**. Jeżeli masz wektory takiej samej długości, to działania wykonywane są element po elemencie, np. dla operatora dodawania: $[1,4]+[2,5]=[3,9]$. Przeanalizuj poniższe przykłady.

```
> x <- c(4, 7, 3, 9)
> y <- c(0, 3, 1, 5)
> x + y
[1]  4 10  4 14

> 3 * x
[1] 12 21  9 27

> (2 * x - y^2) / x
[1]  2.000000  0.714286  1.666667 -0.777778
```

Na tym mógłbym zakończyć opis operacji na wektorach, gdyby nie to, że R wykona operacje również wtedy, gdy wektory mają różną długość – uważaj na to. Działa to tak, że krótszy z wektorów jest powielany tyle razy, aby zrównał się z długością tego dłuższego. Jest to tzw. ***reguła zawijania*** (*recycling rule*). Przykładowo, chcesz dodać dwa wektory: $[5,7,3]$ i $[1,3,7,4,9,2]$. Pierwszy zostanie powielony dwa razy, więc ostatecznie będzie miał postać: $[5,7,3,5,7,3]$. Dopiero teraz zostanie dodany do drugiego. Nie zostaniesz o tym fakcie poinformowany przez program, bo długość jednego jest wielokrotnością długości drugiego. Zobacz: pierwszy i drugi mają długości 3 i 6 odpowiednio, a 6 jest wielokrotnością 3.

Zapewne zastanawiasz się, co dzieje się w sytuacji, gdy długość większego nie jest wielokrotnością długości mniejszego. Reguła zawijania zostanie zastosowana, ale o tym R poinformuje cię odpowiednim komunikatem. Może spróbujesz dodać jakieś dwa wektory, aby wygenerować taki komunikat?

Zanim przejdziemy do przykładów, zastanówmy się, w jaki sposób mnożona jest liczba przez wektor. Otóż liczba to *de facto* wektor z jednym elementem. Dlatego reguła zawijania tutaj też obowiązuje. Pomnożyć liczbę 2 przez wektor $[5,7,3]$, to inaczej pomnożyć dwa wektory: $[2,2,2]$ i $[5,7,3]$ element po elemencie. Jaki będzie wynik? Na końcu będzie 6?

```
> x <- c(5, 7, 3) # długość 3
> y <- c(1, 3, 7, 4, 9, 2) # długość 6
> x + y # zawijanie bez komunikatu, bo 6/3 jest całkowite
[1]  6 10 10  9 16  5

> y <- c(1, 3, 7, 4, 9, 2, 100) # długość 7
> x + y # zawijanie z ostrzeżeniem

Warning in x + y: longer object length is not a multiple of shorter object length
[1]   6  10  10   9  16   5 105

> x + 1000
[1] 1005 1007 1003
```

```
> x <- c(8,  2,  4,  12, 10,  6)
> y <- c(1,  5,  9,  0,  3,  -5)
> z <- c(8, 9, 0)
> z / x # reguła zawijania (dla którego wektora?)
[1] 1.000000 4.500000 0.000000 0.666667 0.900000 0.000000
```

Jeżeli użyjesz **operatora logicznego, relacyjnego lub binarnego** to w wyniku zawsze otrzymasz wektor logiczny z kombinacją stanów: `TRUE`, `FALSE`. Zanim przejdziemy do przykładów, które rozwieją twoje wątpliwości odnośnie tych operatorów, chciałbym zwrócić twoją uwagę na dwie kwestie. Pierwsza – używaj operatorów logicznych `|`, `&` tylko do wektorów logicznych. Jeżeli zapiszesz `x|y`, miej pewność, że zarówno `x` jak i `y` są typu logicznego. Druga – operator relacyjny równy (`==`) ma dwa znaki równości. Często, na samym początku przygody z R, jest on zastępowany jednym znakiem równości, co jest źródłem wielu błędów.

```
> ## Rozważmy krótki przykład: dzienny utarg z wizyt
> ## w kolejnych dniach tygodnia. Pracuś?
> stawka <- c(100, 70, 90, 150, 120, 110, 130) # koszt wizyty u specjalisty
> ilePacjent <- c(3, 5, 4, 4, 1, 7, 3) # liczba przyjętych pacjentów
> utarg <- stawka * ilePacjent
> utarg
[1] 300 350 360 600 120 770 390

> utarg != 150 # który utarg jest różny od 150
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE

> utarg == 150 # który utarg jest równy 150
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE

> utarg >= 300 # który utarg przynajmniej 300
[1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE

> (utarg > 350) | (utarg < 200) # który utarg większy od 350 lub mniejszy od 200
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

> (utarg < 350) & (utarg > 250) # który utarg między 250 a 350
[1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

> 150 %in% stawka # czy 150 należy do zbioru wartości stawka
[1] TRUE

> c(70, 300) %in% stawka # która wartość wektora należy do stawka
[1]  TRUE FALSE
```

Zauważ, że w niektórych przykładach R wykonuje dwa kroki. Wynikiem pierwszego są dwa wektory logiczne: `utarg > 350` oraz `utarg < 200`. Drugi krok to już operacja z wykorzystaniem logicznego lub (`|`). Użyliśmy go, bo mamy 2 wektory logiczne.

Prowadzisz sklep internetowy. Poniżej dla 8 produktów masz: ceny netto [PLN], stawki podatku vat [%]. Wykorzystaj te dane i policz dla każdego produktu: (a) wartość naliczonego podatku vat (b) cenę brutto. Wiesz, że cena netto powstała po naliczeniu 25% marży. Jaka była cena przed dodaniem marży?

```
cenaNetto <- c(12, 17, 34, 78, 95, 47, 27, 77)
vat <- c(0, 23, 7, 8, 5, 0, 23, 22)
```

2.3.3. Odwołania do elementów wektora. Do każdego elementu wektora możesz się odwołać, wskazując indeks (pozycję) tego elementu, który ujmujesz w nawiasy kwadratowe. Tak naprawdę `[` jest tzw. operatorem indeksowania. Musisz wiedzieć, że numerowanie elementów w R zaczyna się od 1 (w niektórych językach od 0, np. w Python). Dla przykładu weźmy zdefiniowany wektor `utarg` z poprzedniego podrozdziału. Jeżeli napiszesz w konsoli: `utarg[5]` to zostanie wzięty 5 element wektora `utarg`, czyli wartość 120.

Bardzo ważna jest umiejętność wyboru elementów, dlatego poniżej zamieszczam kilka możliwości, dla jakiegoś hipotetycznego wektora `x`:

1. `x[liczba]` – wybierz element będący na pozycji `liczba`, np. `liczba=2`, wtedy wybrany zostanie drugi element;
2. `x[-liczba]` – minus oznacza: weź wszystko oprócz elementu o indeksie `liczba`;
3. `x[wektor]` – elementami obiektu `wektor` jest wektor indeksów, np. `x[c(1, 8, 3)]` oznacza wybranie elementów o indeksach: 1, 8 i 3 (dokładnie w takiej kolejności);
4. `x[-wektor]` – wybierz wszystkie elementy oprócz tych, których indeksy zawiera `wektor`;
5. `x[wektor_logiczny]` – `wektor_logiczny` musi mieć taką samą długość jak wektor `x`; `TRUE` oznacza wzięcie elementu, `FALSE` przeciwnie; jeżeli `x` ma 3 elementy, to np. `x[c(TRUE, TRUE, FALSE)]` zwróci pierwsze dwa elementy i pominie ostatni.

Zobacz, jak te różne strategie wyboru możemy zastosować do wcześniej zdefiniowanego wektora `utarg`.

```
> utarg
[1] 300 350 360 600 120 770 390

> utarg[3] # wybierz 3 elment
[1] 360

> utarg[3:5] # wybierz elementy od 3 do 5, bo 3:5 = c(3, 4, 5)
[1] 360 600 120

> utarg[c(1, 3, 4, 5)] # wybiera elementy o wskazanych indeksach
[1] 300 360 600 120

> utarg[-2] # wybierz wszystkie oprócz elementu na pozycji 2
[1] 300 360 600 120 770 390

> utarg[-c(1, 4)] # wybierz wszystkie bez elementu na pozycji: 1 i 4
[1] 350 360 120 770 390

> doktor <- c(1, 3, 4, 7)
> utarg[doktor] # wybierz elementy o indeksach zapisanych w wektorze: doktor
[1] 300 360 600 390

> utarg[c(TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE)] # wybierz element tam gdzie TRUE
[1] 300 360 600 120
```

Postaraj się jeszcze przeanalizować poniższe przykłady – celowo nie zamieszczam wyników.

```
> utarg[utarg > 300]
> utarg[utarg > 350 | utarg < 200]
> utarg[utarg < 350 & utarg > 250]
```

Ponieważ wiesz, w jaki sposób odwoływać się do dowolnych elementów wektora, więc zmiana takich elementów jest prosta. Wystarczy, że przypiszesz starym elementom wartości nowe. Załóżmy, że wysokość utargu przekraczająca 400 ma mieć wartość 1000. Dodatkowo element pierwszy i trzeci zamienimy odpowiednio na 0 i 1.

```
> ## Aby nie nadpisywać naszego wektora utarg, tworzymy kopię
> utarg2 <- utarg
> utarg2[utarg2 > 400] <- 1000 # krok 1
> utarg2[c(1, 3)] <- c(0, 1) # krok 2
> utarg2
[1]    0  350    1 1000  120 1000  390
```

Znasz kilka sposobów odwołania się do elementów wektora. Wykorzystaj każdy z nich, aby wybrać drugi i czwarty element wektora: `c(12, 55, 34, 97, 23)`. Niektóre z podejść są bardziej efektywne, a inne mniej – które?

2.3.4. Wybrane funkcje dla wektorów. Zapewne zgodzisz się, że operatory arytmetyczne dają ograniczone możliwości przetwarzania wektorów. Dlatego jeśli zapytasz, jak policzyć logarytm z elementów wektora, jak znaleźć wartość największą wektora czy jego długość, odpowiem: skorzystaj z wbudowanych funkcji. Poniżej zamieszczam podstawowe funkcje matematyczne, których argumentami mogą być również wektory. W tym wypadku operacje wykonywane są element po elemencie. Poniżej tabeli znajdziesz krótkie przykłady ich wykorzystania.

Tabela 2.1. Funkcje matematyczne dla wektorów

Funkcja	Opis
`log(x)`	Logarytm naturalny z x
`exp(x)`	Liczba e podniesiona do potęgi x
`log(x, n)`	Logarytm z x przy podstawie n
`sqrt(x)`	Pierwiastek kwadratowy z x
`factorial(x)`	$x! = 1 \cdot 2 \cdot \ldots \cdot x$
`choose(n, k)`	Symbol Newtona $\frac{n!}{k!(n-k)!}$
`abs(x)`	Wartość bezwzględna z x
`round(x, digits=n)`	Zaokrągla x do n miejsc po przecinku

```
> ## Przykłady wykorzystania funkcji matematycznych dla wektorów
> x <- rnorm(5) # generuje 5 liczb losowych z rozkładu normalnego N(0,1)
> x
[1] -2.0757530 -1.4697311 -0.3569707  0.1132655 -0.0785911

> round(x, 1) # zaokrągl do 1 miejsca po przecinku
[1] -2.1 -1.5 -0.4  0.1 -0.1

> exp(x) # oblicz wartości funkcji e w punktach x
[1] 0.125462 0.229987 0.699793 1.119929 0.924418

> abs(x) # oblicz wartości bezwzględne
[1] 2.0757530 1.4697311 0.3569707 0.1132655 0.0785911

> log(abs(x)) # najpier oblicz wartość bezwzględną, później log. naturalny
```

```
[1]  0.730324  0.385079 -1.030102 -2.178021 -2.543497
```

Kontynuujemy przegląd funkcji wbudowanych w R. Teraz poznasz tzw. funkcje agregujące. Czytając opis funkcji spróbuj wyobrazić sobie sytuację, w której mogłyby się przydać. Przykładowo, masz wektor z wartościami 100 transakcji zrealizowanych za pomocą karty. Jaką informację możesz uzyskać, posługując się tymi funkcjami? Następnie zapoznaj się z przykładami, które zamieściłem poniżej.

Tabela 2.2. Funkcje agregujące dla wektorów

Funkcja	Opis
`length(x)`	Długość (liczba elementów) wektora x
`max(x, na.rm = FALSE)`	Największa wartość z x; usunie braki danych, gdy `na.rm = TRUE`
`min(x, na.rm = FALSE)`	Najmniejsza wartość z x; usunie braki danych, gdy `na.rm = TRUE`
`sum(x, na.rm = FALSE)`	Suma wszystkich wartości x; usunie braki danych, gdy `na.rm = TRUE`
`prod(x, na.rm = FALSE)`	Iloczyn wszystkich wartości x; usunie braki danych, gdy `na.rm = TRUE`
`sort(x,decreasing = FALSE)`	Sortuje (rosnąco) wartości x; gdy `TRUE` – malejąco
`sample(x, n, replace = TRUE)`	Losowanie `n` elementów wektora x ze zwracaniem (`replace=TRUE`) lub bez zwracania (`replace = FALSE`)
`which(x)`	Zwraca te indeksy wektora logicznego x, które mają wartość `TRUE`, np. `which(x == 5)` podaje indeksy wektora x równe 5.
`is.na(x)`	Zwraca wektor logiczny, w którym `TRUE` pojawia się tylko wtedy, gdy jest brakująca obserwacja. Jeśli $x = [1, \text{NA}, 5, \text{NA}]$ to operacja `is.na(x)` zwróci `FALSE, TRUE, FALSE, TRUE`
`unique(x)`	Usuwa duplikaty wektora x. Jeśli $x = [1,3,2,1,3,2,1]$ to operacja `unique(x)` zwróci $[1,3,2]$
`table(x)`	Zwraca tabelę kontyngencji. Jeżeli $x = [\text{"}a\text{"}, \text{"}b\text{"}, \text{"}a\text{"}, \text{"}a\text{"}]$ to `table(x)` zwróci $a = 3, b = 1$

```
> ## Przykład: z wektora wartości od 1 do 100 wylosuj 10 liczb
> set.seed(76) # ustaw ziarno generatora (gwarantuje identyczność losowania)
> los <- sample(1:100, 10, replace = FALSE)
> los
 [1]  1 28 50 74 16 87 21 78  6 10

> max(los)
[1] 87

> sum(los)
[1] 371

> sort(los) # argument decreasing pominięty, dlatego użyty domyślny FALSE
 [1]  1  6 10 16 21 28 50 74 78 87

> zestawienie <- c(range(los), sum(los), length(los))
> zestawienie
[1]   1  87 371  10

> sum(los)/length(los) # oblicz średnią arytmetyczną
[1] 37.1
```

Na szczególną uwagę zasługuje funkcja `which()`, która z pewnością przyda ci się wiele razy. Zapamiętaj: funkcja ta zwraca indeksy (pozycje elementów). Jeżeli interesują cię nie tylko indeksy, ale również elementy które odpowiadają tym indeksom, to potrzebujesz dodatkowego kroku. Ale po kolei – spójrz na poniższy przykład.

```
> los
 [1]  1 28 50 74 16 87 21 78  6 10

> which(los > 65) # które elementy są większe od 65, pokaż ich indeksy
[1] 4 6 8
```

Zauważ, że indeksom: 4, 6, 8 odpowiadają wartości 74, 87, 78, które faktycznie są większe od 65. Jak już pisałem, R zwrócił tylko te indeksy. A jak wyświetlić te wartości? Podejście jest identyczne, jak przy odwoływaniu się do elementów wektora. Musisz w nawiasach kwadratowych umieścić wektor, którego wartościami są numery indeksów:

```
> los[which(los > 65)]
[1] 74 87 78
```

W wcześniejszym zadaniu podałem ceny i stawki vat dla 8 produktów. Poniżej jeszcze raz je zamieszczam.

```
cenaNetto <- c(12, 17, 34, 78, 95, 47, 27, 77)
vat <- c(0, 23, 7, 8, 5, 0, 23, 22)
```

Wykorzystaj wyniki tego zadania, aby policzyć: (a) sumę cen netto (b) pierwiastek kwadratowy z cen netto (c) średnią arytmetyczną cen netto. Ponadto zaokrąglij do 1 miejsca po przecinku wszystkie ceny brutto. Następnie powiedz, które produkty mają cenę brutto mniejszą od 50 PLN. Dodatkowo wyświetl w konsoli te ceny.

2.4. Czynniki

W rozdziale o wektorach napisałem, że mogą one być typu: liczbowego (naturalne, rzeczywiste), logicznego i znakowego. Jest jeszcze jeden typ, trochę podobny do tego ostatniego – **typ czynnikowy** (*factor*). Jego wprowadzenie do struktur danych R jest powiązane z koncepcją zmiennych kategorialnych (nominalne, porządkowe) występujących w statystyce. Jak się przekonasz, ten typ może być naprawdę użyteczny. Czasami jest wręcz niezbędny, jeśli chcesz oszacować parametry modelu (np. modelu regresji) z takiego typu zmiennymi albo umieścić na wykresie kategorie zmiennej w odpowiednim porządku.

Aby utworzyć obiekt typu czynnikowego, użyj funkcji `factor()`, a za obligatoryjny argument `x` przyjmij wektor (dowolnego typu). Spójrz na trzy warianty:

```
factor(x)
factor(x, ordered = TRUE)
factor(x, levels = ..., labels = ..., ordered = ...)
```

Argumentu opcjonalnego `ordered = TRUE` użyj wtedy, gdy zmienna ma charakter porządkowy. Z kolei `levels` (poziomy) i `labels` (etykiety) będą przydatne, jeżeli sami chcemy poziomom przypisać etykiety. Szczegółowo wyjaśnię to na przykładzie za chwilę. Teraz zapamiętaj tylko, że każdemu poziomowi (każdej liczbie) przyporządkowujesz etykiety (słowny opis), np. 1 ⟶mężczyzna, 2 ⟶kobieta.

Przykład zaczniemy od zdefiniowania wektora typu znakowego, a następnie zamienimy go na czynnik.

```
> mieszka <- c("miasto", "miasto", "wieś", "miasto") # wektor typu znakowego
> mieszka <- factor(mieszka) # zamiana na czynnik
> mieszka
[1] miasto miasto wieś   miasto
Levels: miasto wieś
```

W drugim przykładzie zakładamy, że ktoś dostarczył ci wektor numeryczny z opisem: 0 – miasto, 1 – wieś. Jak zamienić go na czynnik, prezentuję na poniższym przykładzie.

```
> mieszka <- c(0, 0, 1, 0)
> mieszka
[1] 0 0 1 0

> # Zobacz: teraz pojawi się Levels
> factor(mieszka, levels = c(0, 1), labels = c("miasto", "wies"))
[1] miasto miasto wies   miasto
Levels: miasto wies
```

W ostatnim przykładzie rozważymy sytuację, w której mamy zmienną porządkową–wykształcenie. W wypadku takiej zmiennej nie możemy zapomnieć o właściwym porządku dla jej poziomów. Dlatego musimy dodać argument, który informuje o tym: `ordered = TRUE`.

```
> wyksztalcenie <- c("podstawowe", "wyzsze",  "srednie", "wyzsze")
> wyksztalcenie <- factor(wyksztalcenie, levels = c("podstawowe", "srednie", "wyzsze"),
+                 labels = c("podstawowe", "srednie", "wyzsze"),
+                 ordered = TRUE)
> wyksztalcenie
[1] podstawowe wyzsze     srednie    wyzsze
Levels: podstawowe < srednie < wyzsze
```

Poniższy wektor opisuje wykształcenie. Skopiuj kod do swojego skryptu i go uruchom.

```
## Losujemy 20 liczba ze zbioru: 1, 2, 3, 4
set.seed(1234) # Ustawienie ziarna generatora liczb (można pominąć)
(edu <- sample(1:4, 20, replace = TRUE))
 [1] 4 4 2 2 1 4 3 1 1 2 4 4 2 3 2 2 2 3 2 4
```

Twoje zadanie polega na zamianie wektora liczbowego edu na wektor typu czynnik. Kodowanie przebiega według schematu: 1 – podstawowe, 2 – średnie, 3 – licencjat, 4 – magisterium.

2.5. Zadania

Zad. 1. Wykorzystaj odpowiednie funkcje i utwórz wektory, nadając im nazwy (wymyślone). Wektory mają składać się z następujących elementów:

a) $1, 4, 6, 13, -10$
b) "czy", "to", "jest", "wektor z NA"
c)★ $1, 3, 5, \ldots, 101$.
d)★ $1, 8, 15, \ldots, 106$. Wykorzystaj funkcję `seq()`. Znajdziesz ją w pomocy.
e)★ $4, 4, 4, 4, 7, 7, 7, 7, 9, 9, 9, 9$. Wykorzystaj funkcję `rep()`. Znajdziesz ją w pomocy.
f)★ $4, 7, 9, 4, 7, 9, 4, 7, 9, 4, 7, 9, 4, 7, 9, 4, 7, 9$. Tutaj również będzie przydatna funkcja `rep()`.

Następnie dla każdego podaj: długość (liczba elementów), typ, element najmniejszy i największy. Wartości wektorów posortuj. Skorzystaj z odpowiednich funkcji.

Zad. 2. Wykorzystaj poniższy skrypt do wygenerowania wektora `cena` w PLN.

```
set.seed(1313)
cena <- rnorm(100, mean=50, sd=10)
```

Następnie zaokrąglij cenę do dwóch miejsc po przecinku. Zdefiniuj nowy wektor, którego wartości będą ceną wyrażoną w EURO; przyjmij kurs wymiany na poziomie 4.57 PLN/EUR. Nowy wektor zaokrąglij do liczb całkowitych, a następnie:

a) znajdź jego wartość największą i najmniejszą;
b) podaj liczbę jego unikalnych elementów, później je posortuj i wyświetl w konsoli R;
c) wykorzystaj wzory i oblicz: sumę elementów ($\sum_{i=1}^{n} x_i$), średnią arytmetyczną ($\frac{1}{n}\sum_{i=1}^{n} x_i$) i geometryczną ($\sqrt[n]{x_1 \cdot x_2 \cdot \ldots x_n}$). Pamiętasz zależność między pierwiastkowaniem a potęgowaniem?
d) podaj liczbę wartości: (1) większą od 13 EUR, (2) mniejszą od 15 EUR i większą od 10 EUR.

Zad. 3. Wykonaj procedurę rekodowania poniższego wektora numerycznego `zwierz` zgodnie ze schematem: 1–`koza`, 5–`pies`, 8–`kot`. Następnie sprawdź, jak często (procentowo) pojawiają się poszczególne zwierzęta.

```
zwierz <- sample(c(1, 5, 8), 77, replace = TRUE)
```

Zad. 4. Studenci ustosunkowali się do stwierdzenia: Politechnika Wrocławska jest najlepszą uczelnią, dlatego tu studiuję. Wyniki tego krótkiego badania mamy zapisane w poniższym wektorze `opinia`:

```
opinia <- sample(1:5, sample(175:277, 1), replace = TRUE,
                 prob = c(0.05, 0.1, 0.15, 0.3, 0.4))
```

- Zamień wartości numeryczne wektora `opinia` na opisy słowne: 1–`zdecydowanie nie zgadzam sie`, 2–`raczej sie nie zgadzam`, 3–`nie mam zdania`, 4–`raczej sie zgadzam`, 5–`zdecydowanie sie zgadzam`. Czy uwzględnisz tutaj charakter porządkowy zmiennej? Od tej pory wektor `opinia` zawiera opisy słowne.
- Ilu studentów wzięło udział w badaniu?
- Wypisz indeksy tych studentów, którzy udzielili skrajnych odpowiedzi (1 lub 5 w oryginalnym wektorze).
- Oblicz odsetek odpowiedzi.

Zad. 5. Niech `x` będzie wektorem numerycznym:

```
x <- rchisq(57, df = 10, ncp = 30)
```

Oblicz wartości poniższych wyrażeń. Pamiętaj, że zachodzi relacja między notacją matematyczną x_i, a elementami wektora w R $x[i]$, czyli : $x_1 = x[1]$, $x_2 = x[2]$, $x_3 = x[3]$ itd. W poniższych wzorach $\bar{x}$ oznacza średnią arytmetyczną.

- $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$
- $w = \frac{N}{(N-2)(N-1)s^3}\sum_{i=1}^{N}(x_i - \bar{x})^3$

ROZDZIAŁ 3

Ramki danych, wczytywanie i zapisywanie danych

Ramki danych mają strukturę dwuwymiarową, na którą składają się wiersze i kolumny. Każda kolumna odpowiada jednej zmiennej. Wiersze natomiast nazywamy przypadkami albo obserwacjami. Przykładowo: jedna kolumna może składać się z liczb rzeczywistych odnoszących się do masy ciała, druga zawierać nazwy miejscowości, a trzecia opisywać wykształcenie. Jeśli mamy 100 badanych, to mamy również 100 wiersz. Tak zorganizowany zbiór danych znasz np. z arkuszy kalkulacyjnych. Ponieważ tak zorganizowana struktura danych jest najczęściej wykorzystywana przy analizach statystycznych, dlatego w tym rozdziale poznasz, jak taką ramkę:

- utworzyć, wykorzystując funkcję `data.frame()`;
- poddać podstawowym operacjom takim jak wybór zmiennych lub obserwacji do analizy;
- zrestrukturyzować, zamieniając zmienne na obserwacje – to przekształcenie w **wąską postać** ramki danych. Poznasz również procedurę odwrotną – przekształcanie do **szerokiej postaci**.
- zaimportować do R, gdy jest ona przechowywana w pliku zewnętrznym;
- zapisać do pliku.

3.1. Tworzenie ramek danych

W R ramkę danych utworzysz, używając funkcji

```
data.frame(col1, col2, ...)
```

w której każdy `col` jest wektorem kolumnowym o dowolnym typie, ale identycznej długości. Jeśli wektory będą różnić się długością, wtedy R zastosuje regułę zawijania, którą już znasz (zob. str. 14). Utwórzmy, na dwa sposoby, ramkę danych z trzech wektorów.

```
> ## Definiujemy wektory i tworzymy ramkę danych
> sok <- c("kubus", "pysio", "leon", "bobo frut")
> cena <- c(1.2, 1.35, 1.65, 1.99)
> cukier <- c(11.5, 12, 10, 9.6)
> dfSok <- data.frame(sok, cena, cukier)
> dfSok
        sok cena cukier
1     kubus 1.20   11.5
2     pysio 1.35   12.0
3      leon 1.65   10.0
4 bobo frut 1.99    9.6

> ## Drugi, możliwy sposób
> dfSok2 <- data.frame(sok = c("kubus", "pysio", "leon", "bobo frut"),
+                      cena = c(1.2, 1.35, 1.65, 1.99),
```

```
+                     cukier = c(11.5, 12, 10, 9.6))
> dfSok2
        sok cena cukier
1     kubus 1.20   11.5
2     pysio 1.35   12.0
3      leon 1.65   10.0
4 bobo frut 1.99    9.6
```

Zapamiętaj 3.1

Podgląd na ramkę danych

Jeśli chcesz podglądnąć fragment ramki danych lub poznać jej strukturę użyj funkcji: `head()` oraz `str()` (lub `glimpse()`). Gdy masz dużą ramkę danych (bardzo wiele wierszy i kolumn), nie ma sensu wpisywanie w konsoli nazwy ramki danych i wyświetlanie jej zawartości. R i tak wyświetli tylko niewielką część i najprawdopodobniej nie zobaczysz wszystkich nazw wierszy.

Możesz też wyświetlić nazwy wszystkich zmiennych (kolumn) używając funkcji `names()`.

Utwórz ramkę danych o nazwie `dfPerfumy`, która będzie miała 3 kolumny: `nazwa`, `cena`, `liczba_opinii`. Wartości zmiennych uzupełnij swoimi ulubionymi (przynajmniej 3) zapachami. Wykorzystaj jakąś stronę, by uzyskać dane (np. notino.pl). Przykładowo pierwszy wiersz mógłby zawierać: `ROJA PARFUMS Fetish Parfum pour Homme, 2025, 10`.

Dodatkowo wykorzystaj odpowiednie funkcje, aby:

- podać liczbę wierszy i kolumn oraz nazwy zmiennych.
- Wyświetl też strukturę ramki danych.

3.2. Wybór przypadków i zmiennych do analizy

Dość często chcemy pracować na pewnym podzbiorze ramki danych. Najprostszym przykładem może być wybór zmiennej, dla której chcemy policzyć średnią. Mamy tutaj kilka możliwości. Przedstawię tylko dwie:

1. `moja_ramka$nazwa_kolumny` – po nazwie ramki umieszczasz operator `$`, po którym z kolei podajesz nazwę kolumny. Jeżeli w RStudio napiszesz nazwę ramki danych, a po niej umieścisz symbol `$`, wtedy pojawia się lista rozwijana z nazwą kolumn. Zacznij pisać nazwę lub najedź na właściwą i wciśnij ENTER.
2. `moja_ramka[, numer_kolumny]` – podajesz numer kolumny[1], w której znajduje się zmienna. Jeżeli pracujesz z dużą ramką danych i dodajesz lub usuwasz kolumny, zmieniasz ich kolejność, to numery kolumn też ulegają zmianie. Dlatego preferujemy pierwsze podejście. To jednak ma tę przewagę, że możesz wskazywać numery wierszy i kolumn, bo ogólna postać wygląda nastepująco: `moja_ramka[numery_wierszy, numery_kolumn]`. Brak numeru oznacza: weź wszystko.

Weźmy ramkę danych `dfSok` z poprzedniego rozdziału i wypróbujmy te odwołania:

[1] Zamiast numeru możesz wpisać też nazwę kolumny albo wektor z nazwami, np. `moja_ramka[, c("nazwa1", "nazwa2")]`

```
> ## Wybieramy zmienną: cena, na dwa sposoby
> dfSok$cena # 1. sposób
[1] 1.20 1.35 1.65 1.99

> dfSok[, 2] # 2. sposób
[1] 1.20 1.35 1.65 1.99

> ## Możesz utworzyć wektor o wartościach z kolumny cena i ramki dfSok
> cenaSok <- dfSok$cena
> cenaSok
[1] 1.20 1.35 1.65 1.99
```

Poniżej zamieszczam kilka przykładów, w których wybieram wiersze lub kolumny. Wykorzystuję tutaj drugie podejście oparte na wartościach numerycznych indeksów. Bez wpisywania do R spróbuj przewidzieć wyniki. Koniecznie zapisz je na kartce, aby później sprawdzić poprawność (wpisz do R).

```
> dfSok[1, 2]
> dfSok[1, ]
> dfSok[1, c(1, 3)]
> dfSok[c(2,4), 1]
> dfSok$cukier[2:3]
```

WARTO WIEDZIEĆ

Jeżeli chcesz dodać pojedynczą kolumnę do ramki danych, użyj operatora `$` lub funkcji `data.frame()`. Wykorzystajmy operator `$`, aby dodać kolumnę o nazwie `witC` do ramki `dfSok` o wartościach [24,20,18,32]. Z kolei funkcję użyjemy do dodania kolejnej kolumny `witA` o wartościach [240,230,180,257].

```
> ## Operator
> dfSok$witC <- c(24, 20, 18, 32)
> dfSok
        sok cena cukier witC
1     kubus 1.20   11.5   24
2     pysio 1.35   12.0   20
3      leon 1.65   10.0   18
4 bobo frut 1.99    9.6   32

> ## Funkcja data.frame()
> dfSok <- data.frame(dfSok, witA = c(240, 230, 180, 257))
> dfSok
        sok cena cukier witC witA
1     kubus 1.20   11.5   24  240
2     pysio 1.35   12.0   20  230
3      leon 1.65   10.0   18  180
4 bobo frut 1.99    9.6   32  257
```

W praktyce dość często pracujemy na podzbiorze danych, który jest konsekwencją nałożenia warunków. Przykładowo analizę chcemy wykonać dla kobiet z wykształceniem wyższym, które nie przekroczył czterdziestego roku życia. Chociaż powyższe podejście pozwala stworzyć taki

podzbiór, to jednak procedura może wydawać się dość skomplikowana dla osoby początkującej. Na szczęście z pomocą przychodzi pakiet `dplyr`[2]. W założeniu twórców pakietu każda funkcja wykonuje jedno zadanie. Dlatego jednej funkcji potrzebujemy do wyboru zmiennych (kolumn) i jednej do wyboru przypadków (wierszy).

3.2.1. Wybór przypadków – `filter()`.

Funkcja `filter()` wybiera przypadki z ramki danych `rd`, które spełniają określone warunki. Ogólna składnia jest następująca:

```
filter(rd, warunek_1, warunek_2, ...)
```

Wpisywane po przecinku warunki traktowane są jak koniunkcja warunków – wszystkie muszą być jednocześnie spełnione. Gdy interesuje nas alternatywa (lub), wtedy wystarczy użyć operatora `|`. Zerknij na poniższy przykład ilustrujący.

```
> library(tidyverse) # Jeśli wcześniej nie został wczytany
> ## Z ramki dfSok wybierz wiersze: cena poniżej 1.7 zł, cukier przynajmniej 11
> filter(dfSok, cena < 1.7, cukier >= 11)
    sok cena cukier witC witA
1 kubus 1.20   11.5   24  240
2 pysio 1.35   12.0   20  230
```

Oczywiście powyższy wyniki został tylko wyświetlony. Jeśli powstałą ramkę chcesz użyć w dalszej analizie, musisz przypisać jej nazwę. O obiektach nazwanych mówiliśmy w rozdz. 2.2.

Poniżej przedstawiam bardziej złożone przykłady ilustrujące zachowanie funkcji `filter()`. Jak zwykle potrzebuję zbioru danych, który tym razem wygeneruję przy użyciu poniższego kodu – skopiuj go i wklej do swojego skryptu. Nie zapomnij go uruchomić.

```
## Generujemy wektory do ramki danych
ileObser <- 20
set.seed(12345) # użyj, jeśli chcesz mieć identyczne wartości
plec <- sample(c("k", "m"), ileObser, replace=TRUE, prob=c(0.7,0.3))
wiek <- sample(c(20:60), ileObser, replace=TRUE)
mieszka <- sample(c("miasto", "wies"), ileObser, replace=TRUE, prob=c(0.7,0.3))
papierosy <- sample(0:10, ileObser, replace=TRUE, prob=c(0.9, rep(0.2, times=10)))
wwwGodziny <- sample(0:15, ileObser, replace=TRUE)
```

Wygenerowane wektory danych posłużą mi do stworzenia ramki danych. Nazwa zbioru danych powinna mówić coś o samych danych. Kieruję się jednak celem dydaktycznym – łatwiej dostrzec różnice przy krótszych nazwach – i nazywam ramkę po prostu `x`.

```
> ## Tworzę ramkę danych:
> x <- data.frame(plec, wiek, mieszka, papierosy, wwwGodziny)
> head(x) # Wyświetl 6 pierwszych wierszy
  plec wiek mieszka papierosy wwwGodziny
1    m   59  miasto         6          6
2    m   58  miasto         1          6
3    m   57    wies         3         14
4    m   49    wies         4          4
5    k   20  miasto         2         13
6    k   31  miasto         2          6
```

[2] W rozdziale dotyczącym eksploracyjnej analizy danych będziemy z tego pakietu korzystać. Jak się przekonamy, zaimplementowana tam filozofia i podejście do przetwarzania danych czynią ten proces bardzo przejrzystym i dość prostym.

1. Wybierz tylko mężczyzn

```
> filter(x, plec == "m")
```

```
  plec wiek mieszka papierosy wwwGodziny
1    m   59  miasto         6          6
2    m   58  miasto         1          6
3    m   57    wies         3         14
4    m   49    wies         4          4
5    m   31    wies         1          9
6    m   22  miasto         6         12
7    m   32    wies        10          2
8    m   51  miasto         4          2
```

2. Wybierz tych, którzy wypalają więcej niż 5 papierosów dziennie

```
> filter(x, papierosy > 5)
```

```
  plec wiek mieszka papierosy wwwGodziny
1    m   59  miasto         6          6
2    k   27  miasto         6         11
3    m   22  miasto         6         12
4    m   32    wies        10          2
5    k   60    wies         9          0
```

3. Wyłącz z analizy tych, którzy wypalają 0 lub 1 papierosa dziennie

```
> # Znak: != oznacza różne od
> filter(x, papierosy != 0, papierosy != 1)
```

```
   plec wiek mieszka papierosy wwwGodziny
1     m   59  miasto         6          6
2     m   57    wies         3         14
3     m   49    wies         4          4
4     k   20  miasto         2         13
5     k   31  miasto         2          6
6     k   27  miasto         6         11
7     m   22  miasto         6         12
8     k   28  miasto         4         11
9     m   32    wies        10          2
10    k   39  miasto         4          4
11    k   60    wies         9          0
12    m   51  miasto         4          2
```

4. Wybierz: niepalących, którzy spędzają przed internetem przynajmniej 8 godzin

```
> filter(x, papierosy == 0, wwwGodziny >= 8)
```

```
  plec wiek mieszka papierosy wwwGodziny
1    k   39    wies         0         11
2    k   35    wies         0         10
3    k   51    wies         0          8
4    k   53  miasto         0         10
```

5. Wybierz niepalące kobiety ze wsi

```
> filter(x, plec == "k", mieszka == "wies", papierosy == 0)

  plec wiek mieszka papierosy wwwGodziny
1    k   39    wies         0         11
2    k   35    wies         0         10
3    k   35    wies         0          3
4    k   51    wies         0          8
```

6. Wyłącz osoby miedzy 30 a 50 rokiem życia

```
> filter(x, wiek > 50 | wiek < 30)

   plec wiek mieszka papierosy wwwGodziny
1     m   59  miasto         6          6
2     m   58  miasto         1          6
3     m   57    wies         3         14
4     k   20  miasto         2         13
5     k   27  miasto         6         11
6     m   22  miasto         6         12
7     k   28  miasto         4         11
8     k   51    wies         0          8
9     k   53  miasto         0         10
10    k   60    wies         9          0
11    m   51  miasto         4          2
```

W tym miejscu powinniśmy wspomnieć o operatorze `%in%`, który oznacza: *należy* i zastępuje matematyczny symbol $\in$. Jest on szczególnie użyteczny wtedy, gdy chcemy, aby wartości zmiennej należały do zbioru, np. `x %in% c("zielony", "czerwony")`.

3.2.2. Wybór zmiennych – `select()`.

Zmienne z ramki danych `rd` wybieramy, wykorzystując funkcję `select()`. Składnia ma następująca postać:

```
select(rd, zmienna_1, zmienna_2, ...)
```

Po przecinku wypisujemy nazwy zmiennych, które chcemy wziąć. Jeśli interesuje nas zakres zmiennych, wtedy użyjemy dwukropka. Prześledź poniższe przykłady.

1. Wybierz zmienne: `wiek, papierosy, wwwGodziny`

```
> y <- select(x, wiek, papierosy, wwwGodziny)
> head(y, 2) # wyświetl tylko 2 wiersze

  wiek papierosy wwwGodziny
1   59         6          6
2   58         1          6
```

2. Wybierz wszystkie zmienne: od `wiek` do `papierosy`

```
> y <- select(x, wiek:papierosy) # zakres zmiennych
> head(y, 2) # wyświetl tylko 2 wiersze

  wiek mieszka papierosy
1   59  miasto         6
2   58  miasto         1
```

3. Wybierz pierwszą zmienną, oraz zmienne od `mieszka` do `wwwGodziny`

```
> y <- select(x, plec, mieszka:wwwGodziny)
> head(y, 2) # wyświetl tylko 2 wiersze

  plec mieszka papierosy wwwGodziny
1    m  miasto         6          6
2    m  miasto         1          6
```

WARTO WIEDZIEĆ

Wybór przypadków i zmiennych

Czasami chcemy wybrać zarówno wiersze jak i kolumny. Wtedy możemy wykonać dwa kroki. Dla przykładu wybierzmy zmienne `plec` i `papierosy` oraz te respondentki, które wypalają mniej niż 5 papierosów.

```
> krok1 <- select(x, plec, papierosy) # najpierw zmienne
> krok2 <- filter(krok1, plec == "k", papierosy < 5) # później przypadki
> krok2
   plec papierosy
1     k         2
2     k         2
3     k         0
4     k         4
5     k         0
6     k         4
7     k         0
8     k         0
9     k         0
10    k         0
```

Zauważ, że w pierwszym kroku użyliśmy funkcji `select()`, by w kolejnym wynik tej operacji poddać następnemu przekształceniu, czyli działaniu funkcji `filter()`. Oczywiście nic nie stoi na przeszkodzie, aby to zrobić w jednym kroku, jednak zapis jest mniej czytelny.

```
filter(select(x, plec, papierosy), plec == "k", papierosy < 5)
```

Wykorzystaj utworzoną w tym rozdziale ramkę danych `x` i (a) wybierz mężczyzn (b) wybierz mężczyzn, którzy mieszkają w mieście (c) wybierz tych, którzy wypalają 4 lub 6 papierosów (d) wybierz osoby, które wypalają przynajmniej 6 papierosów (e) wybierz osoby, których wiek jest różny od 35 lat (f) wybierz zmienne: `wiek, papierosy` a następnie osoby, które wypalają 6 papierosów, a ich wiek nie przekroczył 30 lat.

3.2.3. Operator potoku `%>%`.

W ostatnim przykładzie użyliśmy funkcji `select()`, a wynik tej operacji poddaliśmy kolejnemu przekształceniu z wykorzystaniem funkcji `filter()`. Jeżeli chcemy to zrobić w jednym kroku, wtedy zapiszemy:

```
>   filter(select(x, plec, papierosy), plec == "k", papierosy < 5)
```

Takie podejście, na tzw. „cebulkę”, już staje się mało czytelne. Jeżeli kroków będzie więcej, to takiej strategii powinniśmy zdecydowanie unikać. Wobec tego masz dwie możliwości. Pierwsza – przypisujesz krokom pośrednim nazwy, podobnie jak to robiliśmy wcześniej. Druga

– wykorzystujesz **operatora potoku**. Jest on bardzo ważny, dlatego szczegóły zamieszczam w ramce.

Zapamiętaj 3.2

Operator potoku

Operator do przetwarzania potokowego `%>%` przekazuje wszystko co znajduje się po jego lewej stronie jako pierwszy argument funkcji, która jest po stronie prawej. Wyrażenie `x %>% f(y)` jest równoważne zapisowi `f(x,y)`. Możemy powiedzieć: `x` ma być pierwszym argumentem funkcji `f(y)`, czyli `f(x,y)`. Zastosujmy to podejście do wcześniejszego przykładu.

```
x %>%
    select(plec, papierosy) %>%
    filter(plec == "k", papierosy < 5)
```

Niech `x` będzie wektorem, a `dfX` ramką danych. Przepisz poniższe wyrażenia wykorzystując operator potoku. (a) `mean(x, na.rm = TRUE)` (b) `length(which(x > 10))` c) `sum(is.na(x))` d) `select(filter(select(dfX, pyt_1:pyt:5), pyt_3 > 6 | pyt_4 < 3), pyt_1, pyt_2)`

3.3. Restrukturyzacja danych

Poniższe ramki danych zawierają te same dane, ale zorganizowane są w inny sposób. W zależności od rodzaju analizy dane wejściowe powinny mieć jedną z tych form. Dlatego w tym rozdziale pokażę ci, w jaki sposób strukturę ramki po lewej stronie zamienić na ramkę po stronie prawej – lub odwrotnie.

owoc	sty	lut	mar
kiwi	13	17	21
cytryna	11	20	14

`pivot_longer()` →

← `pivot_wider()`

owoc	miesiąc	sztuki
kiwi	Sty	13
cytryna	Sty	11
kiwi	Lut	17
cytryna	Lut	20
kiwi	Mar	21
cytryna	Mar	14

Do takiej restrukturyzacji wykorzystamy dwie funkcje z pakietu `tidyr`. Mają one składnię:

```
pivot_longer(rd, cols, names_to = "nazwaKol", values_to = "nazwaKolWart")
pivot_wider(rd, id_cols = NULL, names_from = nazwaKol, values_from = nazwaKolWart)
```

Funkcja `pivot_longer()` zbiera kolumny (np. miesiące z tabeli po lewej stronie), a następnie je łączy. To **długa postać** ramki danych. Argumentami funkcji są:

`cols` – nazwy kolumn ramki `rd`, które chcemy zebrać. Może to być wektor `c(sty, lut, mar)`, zakres `sty:mar` itp. Negację uzyskamy, dając przed: - albo `!`. Mówimy wtedy: nie bierz tych zmiennych. Tutaj nie dajemy cudzysłowów, bo odwołujemy się do istniejących nazw kolumn;

`names_to` – wymyślona nazwa kolumny, w której znają się nazwy zebranych kolumn, np. `names_to = "miesiac"`. A tutaj nazwy nie ma w `rd`, więc będzie cudzysłów;

`values_to` – wymyślona nazwa kolumny dla wartości, które mają zebrane kolumny, np. `values_to = "sztuki"`.

Przeanalizuj poniższy przykład. Następnie zaproponuj 2 inne sposoby zapisu wartości dla argumentu funkcji `cols`.

```
> ramkaLewa
     owoc sty lut mar
1    kiwi  13  17  21
2 cytryna  11  20  14
```

```
> ramkaPrawa <- pivot_longer(ramkaLewa,
+                      cols = c(sty, lut, mar),
+                      names_to = "miesiac",
+                      values_to = "sztuki")
> ramkaPrawa
# A tibble: 6 x 3
  owoc    miesiac sztuki
  <chr>   <chr>    <dbl>
1 kiwi    sty         13
2 kiwi    lut         17
3 kiwi    mar         21
4 cytryna sty         11
5 cytryna lut         20
6 cytryna mar         14
```

Z kolei funkcja `pivot_wider()` działa dokładnie odwrotnie do funkcji `pivot_longer()`. Rozciąga jedną z kolumn na kilka. To **szeroka postać** ramki danych. Tutaj `id_cols` to kolumny, które identyfikują obserwacje – domyślnie brane są wszystkie. Musimy jednak podać nazwę kolumny `names_from`, którą chcemy rozciągnąć na nowe kolumny (np. `miesiąc`). Za `values_from` bierzemy nazwę kolumny, której wartości wypełnią nowo utworzone kolumny (np. `sztuki`). Zastosuj tę funkcję do tabeli po prawej stronie, a miesiące zostaną rozciągnięte na kilka kolumn. W ten sposób otrzymasz tabelę po lewej stronie. Spójrz na poniższy przykład.

```
> ramkaPrawa
# A tibble: 6 x 3
  owoc    miesiac sztuki
  <chr>   <chr>    <dbl>
1 kiwi    sty         13
2 kiwi    lut         17
3 kiwi    mar         21
4 cytryna sty         11
5 cytryna lut         20
6 cytryna mar         14
```

```
> pivot_wider(ramkaPrawa,
+           names_from = miesiac,
+           values_from = sztuki)
# A tibble: 2 x 4
  owoc      sty   lut   mar
  <chr>   <dbl> <dbl> <dbl>
1 kiwi       13    17    21
2 cytryna    11    20    14
```

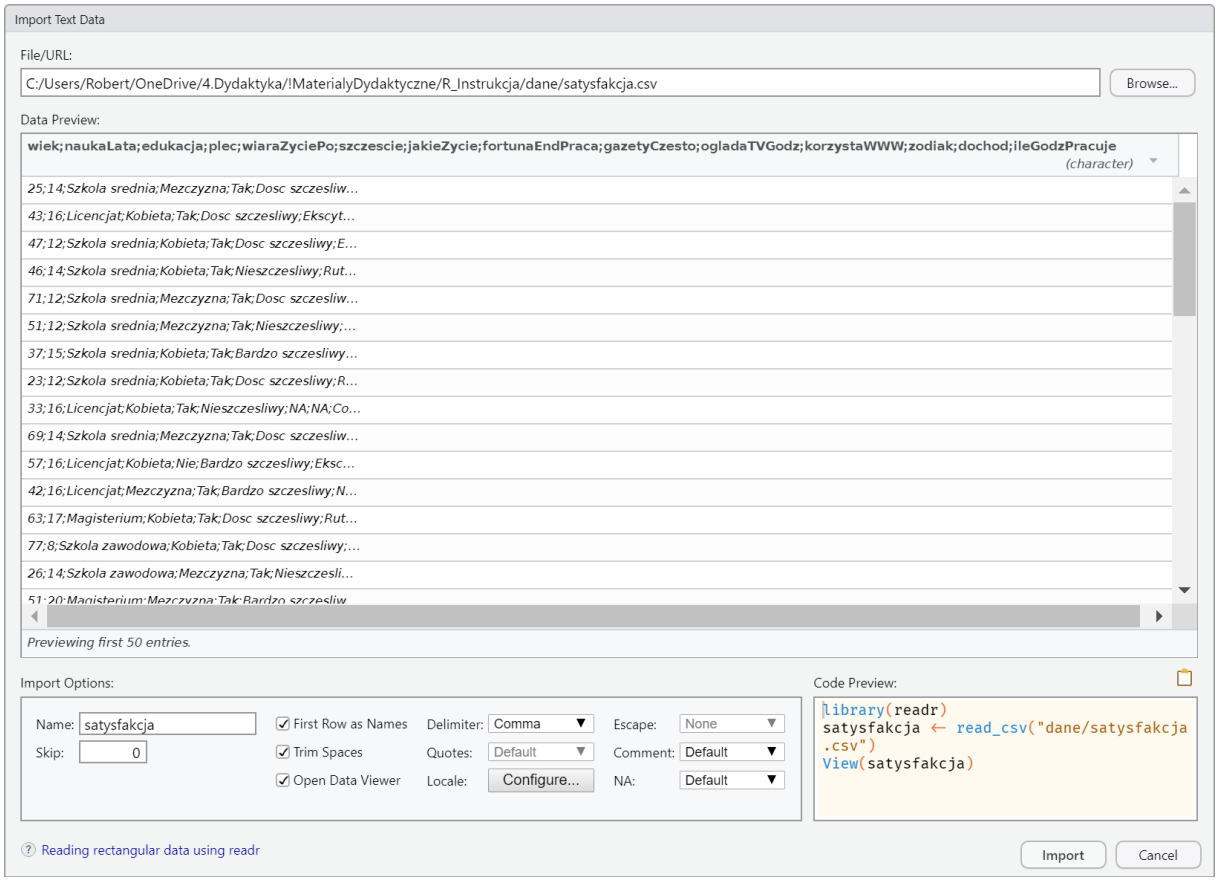
Poniższa tabela pokazuje m.in. ceny pewnych produktów w dwóch miastach. Użyj jej, aby utworzyć ramkę danych. Następnie przekształć tę ramkę tak, aby miała wąską postać. Wykonaj również procedurę odwrotną.

produkt	tłuszcz	konserwant	Wrocław	Warszawa
jogurt	3%	tak	3	4
ser	25%	nie	21	28
mleko	3%	nie	2	3

3.4. Wczytywanie i zapisywanie danych

W analizach statystycznych wykorzystujemy dane, które najczęściej przechowywane są w plikach zewnętrznych. Przykładem mogą być pliki tekstowe o rozszerzeniach: `txt`, `dat`, `csv` lub pliki w formacie `xls` i `xlsx`. W tym rozdziale nauczysz się, w jaki sposób zaimportować takie pliki do **R**. Dodatkowo pokażę ci, jak wykonać procedurę odwrotną, tzw. zapisać plik w wybranym formacie.

Do wczytywania czy też importu plików wykorzystamy kreator[3] wbudowany w **RStudio**. Prześledźmy jego działanie na przykładzie wczytania zbioru danych `satysfakcja.csv`. W oknie 4 (zob. str. 7) klikamy na ikonę *Import Dataset* i z menu rozwijanego wybieramy *From Text (readr)*. Ta opcja pozwala tylko na wczytanie plików tekstowy. Gdy pojawi się okno importu wybierz *Browse...* i interesujący nas plik. W konsekwencji otrzymasz okno, które zamieszczam i omawiam poniżej.



1. *Data Preview* – podgląd wczytywanego zbioru. Jeśli domyślne ustawienia są prawidłowe, wtedy każda zmienna znajduje się w osobnej kolumnie. W naszym wypadku mamy tylko jedną kolumnę (typu `character`), w której są wszystkie zmienne. Oznacza to, że pewne opcje należy zmienić.

[3] Bardziej zaawansowani użytkownicy z niego nie korzystają. Po prostu bardziej efektywna jest praca, gdy w skrypcie samemu się te funkcji wpisuje.

2. *Name* – nazwa wczytywanego zbioru. Domyślnie przyjmowana jest taka sama, jak nazwa pliku. Preferujemy krótsze nazwy dla ramek danych, dlatego zmień na `saty`.
3. *First Row as Names* – w pierwszym wierszu są nazwy zmiennych, dlatego ta opcja powinna być zaznaczona.
4. *Open Data Viewer* – po imporcie nie chcemy, aby automatycznie uruchomił się podgląd ramki danych. Możemy z takiego podglądu skorzystać później.
5. *Delimiter* – wybierasz znak separacji kolumn. Przecinek (*Comma*) nie został użyty w tym pliku, dlatego kolumny nie zostały właściwie odseparowane. Patrząc na podgląd danych widzimy, że tym separatorem będzie średnik (*Semicolon*). Po zmianie otrzymasz właściwą strukturę. Do wyboru masz jeszcze: tabulator *Tab*, biały znak (*Whitespace*) albo inny znak (*Other...*), który musisz podać.
6. *Locale* – wybierasz sposób kodowania znaków. Jeśli na podglądzie widzisz tzw. krzaczki, zamiast polskich znaków, wtedy możesz zmienić domyślne kodowanie. W systemie Windows sprawdzi się `CP1250`.
7. *NA* – jeśli braki danych oznaczone są w nietypowy sposób, musisz zmienić wartość domyślną. Przykładowo w pliku mamy zmienną masa ciała i wszystkie braki zostały oznaczone jako −1. Wtedy z listy rozwijanej wybierz np. 0, a we fragmencie kodu (*Code Preview*s), gdzie pojawiło się: `na = "0"`, podmień 0 na −1.
8. *Code Preview* – widok kodu, który uruchamia procedurę importu. Zaczyna się ona od wczytania pakietu `readr`. Jeśli w bieżącej sesji **RStudio** pakiet był już wczytywany, możesz tę linijkę usunąć. W kolejnej pojawia się właściwa funkcja z tego pakietu, która wczytuje dane. Zobacz, jak zmienia się składnia, gdy zmieniasz opcje importu.

Po uruchomieniu importu, składnia widoczna w oknie *Code Preview* jest przekazywana do konsoli **R**. Skopiuj ją do swojego skryptu. Jeśli w kolejnej sesji **RStudio** zechcesz pracować z tym samym zbiorem danych, wystarczy, że uruchomisz ten skrypt. Tym samym nie musisz korzystać z kreatora importu. Zobacz, na poniższym rysunku, jak mógłby wyglądać początek twojego skryptu.

Pliki w formacie `xls` i `xlsx` zaimportujesz, używając tego samego kreatora. Jednak tym razem, po kliknięciu na ikonę *Import Dataset*, wybierasz *From Excel*. Opcje importu pozwalają na wybór importowanego arkusza – jeśli w skoroszycie Excela jest ich więcej niż jeden. Dodatkowo możemy określić zakres importowanych komórek, jeśli z jakiś względów ten domyślny zakres chcemy zmienić. Zauważ, że tym razem wykorzystujemy pakiet `readxl`.

Przejdźmy teraz do eksportu pliku. Ponieważ nie ma kreatora pozwalającego zapisać ramkę danych do pliku, dlatego sami musimy sami użyć odpowiedniej funkcji. Choć możliwości jest bardzo dużo, wspomnę tylko o jednej, która zapisuje ramkę danych do pliku tekstowego `csv`. Składnia jest dość prosta, gdyż wymaga od nas podania nazwy pliku oraz ramki danych, którą chcemy wyeksportować. Poniżej zamieszczam funkcję (znajduje się w pakiecie `readr`) i jej wymagane argumenty.

```
write_excel_csv2(nazwa_ramki, "nazwa_pliku")
```

Wcześniej zaimportowaliśmy zbiór danych i zapisaliśmy pod nazwą `saty`. Teraz zapiszę go w tym samym katalogu `dane`, pod nazwą: `satystakcjaZapis.csv`. Zobacz, jak łatwo mogę zapisać ramkę danych. Przyjąłem tutaj, że w bieżącej sesji nie wczytałem wymaganego pakietu.

```
> library(readr)
> write_excel_csv2(saty, "dane/satystakcjaZapis.csv")
```

Weź dowolny plik z danymi (ze swojego komputera) w formacie `xlsx` lub `xls` i zaimportuj go. Przyjmij następującą nazwę dla ramki danych: `mojPlik`. Następnie tę ramkę zapisz w formacie `csv` pod nazwą `mojPlikKopia`. W kolejnym kroku ponownie zaimportuj zapisany plik tekstowy `mojPlikKopia.csv`. Tym razem przyjmij nazwę dla ramki danych: `mojPlikCSV`.

3.5. Zadania

Zad. 1. Zaciągnięto kredyt hipoteczny w wysokości K PLN na okres L lat. Spłata następuje w cyklu miesięcznym, w równych ratach przy rocznej stopie oprocentowania równej r. Szczegóły zawiera poniższy skrypt, skopiuj go do swojego pliku.

```
r <- 0.05 #oprocentowanie roczne
rr <- 1+r/12
K <- 300000 #kwota kredytu
L <- 20 #ile lat
N <- 12*L #liczba rat (ile miesięcy)
n <- 1:N #wektor zawierający kolejne okresy
rataKredytu <- K*rr^N*(rr-1)/(rr^N-1)
zadluzenie <- K*(rr^N-rr^n)/(rr^N-1)
odsetki <- K*(rr^N-rr^(n-1))/(rr^N-1)*(rr-1)
rataKapitalu <- rataKredytu - odsetki
```

Utwórz ramkę danych o nazwie `kredyt`, której kolumnami będą wektory: `rataKapitalu`, `odsetki`, `rataKredytu`, `zadluzenie`. Użyj funkcji `class()` by sprawdzić, czy utworzony obiekt faktycznie jest ramką danych. Następnie wykorzystaj odpowiednie funkcje i:

a) wyświetl pierwszych 10 wierszy;
b) pokaż strukturę ramki;
c) jaki jest wymiar ramki danych;
d) wyświetl w konsoli wiersze: (a) od 100 do 125 (b) pierwszych 20 (c) ostatnich 30 (d) od 20 do 30 i od 50 do 60 (e) co dziesiąty wiersz (10, 20, 30 itd.)
e) oblicz sumaryczną wielkość zapłaconych odsetek, rat kredytu i rat kapitałowych.
f)★ od którego okresu wysokość raty kapitałowej (`rataKapitalu`) zaczyna przewyższać wysokość spłacanych odsetek (`odsetki`)? Może przydać się funkcja `which()`.

Zad. 2. Poniższa ramka danych zawiera informacje o masie [kg] i wysokości [cm] ciała.

```
medic <- data.frame(
c(82.5, 65.1, 90.5, 80.9, 74, 74.4, 73.5, 75.6, 70.1, 61.8, 80.6, 82.2, 54.1, 60),
c(181, 169, 178, 189, 178, 175, 173, 187, 175, 165, 185, 178, 162, 185))
```

a) Zmień domyślne nazwy kolumn na: `masa` i `wysokosc`.
b) Utwórz dodatkową kolumnę (o nazwie `BMI`), której wartościami będzie wskaźnik masy ciała.

Wskaźnik ten obliczamy ze wzoru: $\frac{\text{masa[kg]}}{(\text{wysokosc[m]})^2}$. Uwaga: w danych mamy wysokość w centymetrach, a we wskaźniku w metrach. Poniżej efekt końcowy (wyświetlam tylko 4 wiersze)

c)★ utwórz wektor, który będzie przyjmował wartość 1 – gdy BMI < 18.5, wartość 2 – gdy BMI ∈ [18.5,24.99], wartość 3 – gdy BMI >24.99. Nazwij ten wektor `waga`. Wskazówka: jeżeli `x <- c(30,10,25)` i napiszemy `x > 20` to otrzymamy ciąg: `TRUE, FALSE, TRUE`. Jeżeli zapiszemy `(x > 20)+ 1` to otrzymamy: `2,1,2`. Pamiętasz o rzutowaniu typów?

d) Wektor `waga` z poprzedniego punktu ma postać: 3, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 1. Wykorzystując czynniki, dokonaj kodowania według schematu: 1 – niedowaga, 2 – prawidlowa, 3 – nadwaga. Pamiętaj, że to zmienna porządkowa. Następnie włącz tą zmienną do ramki danych `medic`. Efekt poniżej:

```
> head(medic, 4)
```

```
  masa wysokosc     BMI       waga
1 82.5      181 25.1824    nadwaga
2 65.1      169 22.7933 prawidlowa
3 90.5      178 28.5633    nadwaga
4 80.9      189 22.6477 prawidlowa
```

e) Dodaj wiersz do ramki danych, który będzie odnosił się do jakiejś wymyślonej osoby.

f) Wykorzystaj operatory `[, $` by wybrać różnymi sposobami: (i) jedną kolumnę `masa` (ii) dwie dowolne kolumny.

g) Wybierz te wiersze dla których `BMI` jest większe od 23.

h) Które osoby (nr wiersza) mają wagę prawidłową?

Zad. 3. Umieść w katalogu swojego projektu plik z danymi `satysfakcja.dat` z badań sondażowych, a następnie wczytaj go do R. Przyjmij `saty` dla tej ramki danych. W poniższych poleceniach wykorzystaj operator potoku.

a) Wybierz te zmienne, które odnoszą się do edukacji.

b) Wybierz osoby, które wierzą w życie po śmierci i są spod znaku lwa.

c) Wybierz te osoby, których znak zodiaku zaczyna się od litery B. Na pewno potrafisz je wymienić.

d) Weź wszystkie zmienne od `wieku` do `płci` włącznie, a następnie z tego zbioru wybierz osoby, które uczyły się więcej niż 19 lat.

e) W zbiorze danych są zmienne, które mają charakter porządkowy (np. częstotliwość czytania gazet). Choć są reprezentowane w postaci czynników, to jednak nie uwzględniono tam charakteru porządkowego. Popraw to.

f)★ Wybierz wszystkie zmienne typu numerycznego (użyj funkcji `sapply()`). Dla nich policz sumę i znajdź wartości największe i najmniejsze.

g) Wybierz kilka zmiennych (o różnym typie) i wykonaj na nich operacje wykorzystując funkcje: `sum()`, `table()`, `unique()`. Pamiętaj: nie wszystkie funkcje można wykorzystać do wszystkich zmiennych.

ROZDZIAŁ 4 Rachunek prawdopodobieństwa

4.1. Kombinatoryka

W tym rozdziale wykorzystamy schematy kombinatoryczne, aby utworzyć zbiory, a następnie obliczyć ich liczbę. W obliczeniu liczby zbiorów przydatne będą dwie funkcje (zob. tab. 2.1): silnia – `factorial()` oraz symbol Newtona –`choose()`.

Korzystając z R, możemy w łatwy sposób konstruować zbiory. Napisałem kilka prostych funkcji, które znajdziesz w pliku `kombinatoryka.R`. Wystarczy, że w swoim skrypcie uruchomisz poniższe linijki:

```
> library(RcppAlgos)
```

Pierwsza wczytuje wymagany pakiet, który wcześniej zainstaluj. Druga uruchamia skrypt `kombinatoryka.R`. Pamiętaj, aby skopiować go do katalogu `dane` w swoim projekcie. Listę dostępnych funkcji zamieszczam poniżej.

```
PokazKombinacje(zbior, k, powtorz = FALSE) # Kombinacje bez powtórzeń
PokazKombinacje(zbior, k, powtorz = TRUE)  # Kombinacje z powtórzeniami
PokazWariacje(zbior, k, powtorz = FALSE)   # Wariacje bez powtórzeń
PokazWariacje(zbior, k, powtorz = TRUE)    # Wariacje z powtórzeniami
PokazPermutacje(zbior, ileRazyPowt = NULL)    # Permutacje bez powtórzeń
PokazPermutacje(zbior, ileRazyPowt = wektor)  # Permutacje z powtórzeniami
```

Ze zbioru `zbior` wybieramy `k` elementów, z powtórzeniami (`TRUE`) lub bez powtórzeń (`FALSE`). Ten zbiór jest wektorem, składającym się z unikalnych elementów. Jeżeli wykorzystujemy funkcję pokazującą wszystkie permutacje z powtórzeniami, to `ileRazyPowt` jest wektorem typu liczbowego, o długości odpowiadającej liczbie elementów zbioru. Jego wartości mówią, ile razy dany element zbioru jest powtarzany. Weźmy przykład: jeśli `zbior = c("a", "j", "z")` to `ileRazyPowt = c(3, 6, 1)` pokazuje, że a powtarzamy – 3 razy, `j` – 6 razy, natomiast `z` – 1 raz.

```
> source("dane/kombinatoryka.R", encoding = "UTF-8")
```

W kolejnych rozdziałach zilustruję działanie tych 6 funkcji kombinatorycznych. Będziemy konstruować zbiory i obliczać liczbę jego elementów.

4.1.1. Kombinacje.

Mamy zbiór składający się z 5 elementów, z których chcemy zrobić paczkę.

```
> zbior <- c("banan", "jabłko", "kiwi", "czekolada", "książka")
```

Każda paczka składa się z 2 elementów. Elementy w paczce nie mogą się powtarzać. Liczbę możliwych paczek tworzymy, wykorzystując funkcję:

```
> (kombPaczka <- PokazKombinacje(zbior, 2, powtorz = FALSE))
      [,1]        [,2]
 [1,] "banan"     "jabłko"
 [2,] "banan"     "kiwi"
```

```
 [3,] "banan"     "czekolada"
 [4,] "banan"     "książka"
 [5,] "jabłko"    "kiwi"
 [6,] "jabłko"    "czekolada"
 [7,] "jabłko"    "książka"
 [8,] "kiwi"      "czekolada"
 [9,] "kiwi"      "książka"
[10,] "czekolada" "książka"
```

W każdym wierszu mamy jedną paczkę. A więc pierwsza paczka składa się z banana i czekolady, a ostatnia z kiwi i książki. Jeżeli elementy w paczce mogą się powtarzać (np. będą 2 czekolady), wtedy argument funkcji `powtorz` ustawiamy na `TRUE`.

```
> (kombPaczkaPowtorz <- PokazKombinacje(zbior, 2, powtorz = TRUE))
      [,1]        [,2]
 [1,] "banan"     "banan"
 [2,] "banan"     "jabłko"
 [3,] "banan"     "kiwi"
 [4,] "banan"     "czekolada"
 [5,] "banan"     "książka"
 [6,] "jabłko"    "jabłko"
 [7,] "jabłko"    "kiwi"
 [8,] "jabłko"    "czekolada"
 [9,] "jabłko"    "książka"
[10,] "kiwi"      "kiwi"
[11,] "kiwi"      "czekolada"
[12,] "kiwi"      "książka"
[13,] "czekolada" "czekolada"
[14,] "czekolada" "książka"
[15,] "książka"   "książka"
```

Pierwsza część zadania za nami. Teraz obliczymy liczbę możliwych paczek. Zrobimy to na dwa sposoby. Pierwszy – wykorzystamy utworzony obiekt, który zawiera wszystkie możliwe paczki. Liczba możliwości odpowiada liczbie wierszy tego obiektu (`kombPaczkaPowtorz`). Policzymy ją, wykorzystując funkcję `nrow()`. Drugi – wykorzystuje wzory kombinatoryczne, które znasz. Wyniki zamieszczam poniżej dla kombinacji z powtórzeniami i bez powtórzeń.

```
> ## Liczba możliwości (bez powtórzeń)
> nrow(kombPaczka)
[1] 10

> ## ze wzoru
> choose(5, 2)
[1] 10
```

```
> ## Liczba możliwości (z powtórzeniami)
> nrow(kombPaczkaPowtorz)
[1] 15

> ## ze wzoru
> choose(5 + 2 - 1, 2)
[1] 15
```

4.1.2. Wariacje. Windą jadą trzy osoby. Budynek ma 5 pięter. Na ile sposobów mogą wysiąść, jeżeli (a) każdy wysiada na innym piętrze (b) nie ma żadnych ograniczeń. Ponieważ liczba wariacji jest duża, dlatego wyświetlam 6 pierwszych możliwości.

```
> zbior <- paste0("pietro_", 1:5)
> zbior
[1] "pietro_1" "pietro_2" "pietro_3" "pietro_4" "pietro_5"

> pietroWar <- PokazWariacje(zbior, 3, powtorz = FALSE)
```

```
> head(pietroWar) # wyświetl tylko 6
     [,1]       [,2]       [,3]
[1,] "pietro_1" "pietro_2" "pietro_3"
[2,] "pietro_1" "pietro_2" "pietro_4"
[3,] "pietro_1" "pietro_2" "pietro_5"
[4,] "pietro_1" "pietro_3" "pietro_2"
[5,] "pietro_1" "pietro_3" "pietro_4"
[6,] "pietro_1" "pietro_3" "pietro_5"

> pietroWarPowt <- PokazWariacje(zbior, 3, powtorz = TRUE)
> head(pietroWarPowt) # wyświetl tylko 6
     [,1]       [,2]       [,3]
[1,] "pietro_1" "pietro_1" "pietro_1"
[2,] "pietro_1" "pietro_1" "pietro_2"
[3,] "pietro_1" "pietro_1" "pietro_3"
[4,] "pietro_1" "pietro_1" "pietro_4"
[5,] "pietro_1" "pietro_1" "pietro_5"
[6,] "pietro_1" "pietro_2" "pietro_1"
```

Podobnie jak wcześniej liczbę możliwych sposobów policzymy na dwa sposoby.

```
> ## Liczba możliwości (bez powtórzeń)
> nrow(pietroWar)
[1] 60

> ## ze wzoru
> choose(5, 3) * factorial(3)
[1] 60
```

```
> ## Liczba możliwości (z powtórzeniami)
> nrow(pietroWarPowt)
[1] 125

> ## ze wzoru
> 5^3
[1] 125
```

4.1.3. Permutacje. Dla permutacji bez powtórzeń rozważmy następujący przykład. Do samolotu wchodzą: mama, tata i dziecko. Jeżeli idą gęsiego, to na ile sposobów mogą się ustawić.

```
> zbior <- c("mama", "tata", "dziecko")
> (rodzinaPerm <- PokazPermutacje(zbior))
     [,1]     [,2]     [,3]
[1,] "mama"   "tata"   "dziecko"
[2,] "mama"   "dziecko" "tata"
[3,] "tata"   "mama"   "dziecko"
[4,] "tata"   "dziecko" "mama"
[5,] "dziecko" "mama"   "tata"
[6,] "dziecko" "tata"   "mama"
```

Tutaj również liczbę możliwych ustawień wyznaczamy na dwa sposoby.

```
> ## Liczba możliwości (bez powtórzeń)
> nrow(rodzinaPerm)
[1] 6

> ## ze wzoru
> factorial(3)
[1] 6
```

Zilustruję teraz zachowanie funkcji wyświetlającej permutacje z powtórzeniami. Weźmy przykład: ile różnych wyrazów (mogą nie istnieć) możemy utworzyć ze słowa: ALLAN. Zbiór,

jako argument wejściowy funkcji, jest wektorem typu znakowego. Musi on składać się z liter tego słowa. Zapewne pamiętasz, że dla zbiorów mamy: $\{a, a\} = \{a\}$. Dlatego nasz zbiór składa się z unikalnych liter: `A`, `L` i `N`. Każda z nich w słowie `ALLAN` powtarza się, co uwzględniamy w wartości argumentu `ileRazyPowt`.

```
> ## ALLAN
> zbior <- c("A", "L", "N")
> (wyrazPerm <- PokazPermutacje(zbior, ileRazyPowt = c(2, 2, 1)))
      [,1] [,2] [,3] [,4] [,5]
 [1,] "A"  "A"  "L"  "L"  "N"
 [2,] "A"  "A"  "L"  "N"  "L"
 [3,] "A"  "A"  "N"  "L"  "L"
 [4,] "A"  "L"  "A"  "L"  "N"
 [5,] "A"  "L"  "A"  "N"  "L"
 [6,] "A"  "L"  "L"  "A"  "N"
 [7,] "A"  "L"  "L"  "N"  "A"
 [8,] "A"  "L"  "N"  "A"  "L"
 [9,] "A"  "L"  "N"  "L"  "A"
[10,] "A"  "N"  "A"  "L"  "L"
[11,] "A"  "N"  "L"  "A"  "L"
[12,] "A"  "N"  "L"  "L"  "A"
[13,] "L"  "A"  "A"  "L"  "N"
[14,] "L"  "A"  "A"  "N"  "L"
[15,] "L"  "A"  "L"  "A"  "N"
[16,] "L"  "A"  "L"  "N"  "A"
[17,] "L"  "A"  "N"  "A"  "L"
[18,] "L"  "A"  "N"  "L"  "A"
[19,] "L"  "L"  "A"  "A"  "N"
[20,] "L"  "L"  "A"  "N"  "A"
[21,] "L"  "L"  "N"  "A"  "A"
[22,] "L"  "N"  "A"  "A"  "L"
[23,] "L"  "N"  "A"  "L"  "A"
[24,] "L"  "N"  "L"  "A"  "A"
[25,] "N"  "A"  "A"  "L"  "L"
[26,] "N"  "A"  "L"  "A"  "L"
[27,] "N"  "A"  "L"  "L"  "A"
[28,] "N"  "L"  "A"  "A"  "L"
[29,] "N"  "L"  "A"  "L"  "A"
[30,] "N"  "L"  "L"  "A"  "A"
```

Jeżeli połączymy litery, to zobaczymy słowa. Poniżej zamieszczam rozwiązanie. Składni nie tłumaczę – po prostu skopiuj ją do swojego skryptu i uruchom. Możesz w podobny sposób poprawić rozwiązanie zadania o kolejce do samolotu – zmień tylko `collapse = "-"`. Wygląda lepiej?

```
> apply(wyrazPerm, 1, paste0, collapse = "")
 [1] "AALLN" "AALNL" "AANLL" "ALALN" "ALANL" "ALLAN" "ALLNA" "ALNAL" "ALNLA" "ANALL" "ANLAL"
[12] "ANLLA" "LAALN" "LAANL" "LALAN" "LALNA" "LANAL" "LANLA" "LLAAN" "LLANA" "LLNAA" "LNAAL"
[23] "LNALA" "LNLAA" "NAALL" "NALAL" "NALLA" "NLAAL" "NLALA" "NLLAA"
```

Wyznaczmy jeszcze liczbę możliwości – również na dwa sposoby.

```
> ## Liczba możliwości (bez powtórzeń)
> nrow(wyrazPerm)
[1] 30
```

```
> ## ze wzoru
> factorial(5)/(factorial(2) * factorial(2))
[1] 30
```

4.2. Funkcje R w rachunku prawdopodobieństwa

Wiele funkcji rozkładów prawdopodobieństwa znajduje się w pakiecie `stats`, który wczytywany jest zawsze podczas uruchamiania R. Zawsze możesz sięgnąć do pomocy (wpisz: `?Distributions`), aby zapoznać się z pełną listą. W tym rozdziale opiszę 4 rozkład ciągłe: normalny, t-Studenta, jednostajny, wykładniczy oraz 2 rozkłady dyskretne: dwumianowy i Poissona.

W ramach każdego rozkładu wyróżniamy 4 funkcje:

- gęstość lub rozkład prawdopodobieństwa (`d` – *density*);
- dystrybuantę (`p` – *probability*);
- kwantyle (`q` – *quantile*);
- generator liczb pseudolosowych (`r` – *random*).

Przedrostek który widzisz w nawiasie (wzięty z angielskich nazw) określa, co zwraca funkcja. Przykładowo dla rozkładu normalnego trzonem jest wyraz `norm`, a poprzedzając go przedrostkiem, otrzymamy funkcje: `dnorm()`, `pnorm()`, `qnorm()`, `rnorm()`. W ostatniej przedrostkiem jest `r`, więc funkcja generuje liczby z rozkładu normalnego. Zobaczmy jak wyglądają funkcje i ich argumenty (niektóre pominąłem) dla wybranych rozkładów.

Rozkład normalny: `mean` – średnia, `sd` – odchylenie standardowe; są wartości domyślne.

```
dnorm(x, mean = 0, sd = 1)
pnorm(x, mean = 0, sd = 1)
qnorm(p, mean = 0, sd = 1)
rnorm(n, mean = 0, sd = 1)
```

Rozkład normalny ma dwa parametry: średnią (*mean*) i odchylenie standardowe (*sd*). Gdy tych parametrów nie podasz, wtedy przyjmowane są wartości domyślne widoczne powyżej w definicji funkcji. Argumentami na pierwszej pozycji są: `x` – punkt w którym chcemy obliczyć wartość funkcji gęstości (`d`) albo wartość dystrybuanty (`p`), `p` – prawdopodobieństwo (rząd kwantyla), `n` – ile wygenerować obserwacji. Działanie pierwszych trzech funkcji zilustruję przykładem. Generowanie liczb losowych pozostawiam jako ćwiczenie.

```
> ## Przykład dla funkcji rozkładu normalnego
> ## Uwaga1: jeśli zachowamy kolejność wpisywania, słowa mean i sd można pominąć
> ## Uwaga2: zawsze w zapisie N(a, b), b jest wariancją, dlatego sd = sqrt(b)
> dnorm(0) # średnia i odch. standardowe domyślne, czyli 0 i 1
[1] 0.398942

> dnorm(0, mean = 10, sd = 15)
[1] 0.0212965

> pnorm(0) # Pr(X <= 0), gdzie X - N(0, 1)
[1] 0.5

> pnorm(3, 6, 10) # Pr(X <= 3), gdzie X - N(6, 10^2)
[1] 0.382089

> qnorm(0.7) # Oblicz a, by F(a) = 0.7
```

```
[1] 0.524401

> qnorm(0.7, 50, 25) # Oblicz a, by F(a) = 0.7 ale X - N(50, 25^2)
[1] 63.11
```

Ponieważ funkcje w R są zazwyczaj zwektoryzowane, dlatego argumentami mogą być także wektory. Zobacz jak szybko można obliczyć prawdopodobieństwa dla kilku wartości.

```
> pnorm(c(-1, -0.5, 2.1, 3.5), mean = 0.5, sd = 3) # Oblicz dyst. w punktach
[1] 0.308538 0.369441 0.703099 0.841345
```

Rozkład t-Studenta: `df` – liczba stopni swobody; nie ma domyślnej wartości. `df`

```
dt(x, df)
pt(x, df)
qt(p, df)
rt(n, df)
```

Rozkład jednostajny: `min` i `max` – odpowiednio dolny i górny kraniec przedziału; widoczne wartości są domyślne.

```
dunif(x, min = 0, max = 1)
punif(x, min = 0, max = 1)
qunif(p, min = 0, max = 1)
runif(n, min = 0, max = 1)
```

Rozkład wykładniczy: `rate` – parametr rozkładu (lambda); są wartości domyślne.

```
dexp(x, rate = 1)
pexp(x, rate = 1)
qexp(p, rate = 1)
rexp(n, rate = 1)
```

Rozkład normalny, t-studenta, jednostajny i wykładniczy należą do rodziny rozkładów ciągłych. Opis funkcji które przedstawiłem, a dotyczący przedrostków, ma również zastosowanie do rozkładów dyskretnych. Pamiętaj o jednej istotnej różnicy: wszystkie funkcje z przedrostkiem `d` (*density*), dla rozkładów dyskretnych, zwracają wartość prawdopodobieństwa. Powtórzmy: dla ciągłych – wartość funkcji gęstości, dla dyskretnych – wartość prawdopodobieństwa.

Rozkład dwumianowy: `size` – liczba prób (eksperymentów), `prob` – prawdopodobieństwo sukcesu w pojedynczej próbie; nie ma wartości domyślnej dla `size, prob`.

```
dbinom(x, size, prob)
pbinom(x, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)
```

Wykorzystajmy rozkład dwumianowy do rozwiązania zadania. W teście masz 20 stwierdzeń i musisz rozstrzygnąć, czy są one prawdziwe. Załóżmy, że odpowiedzi zaznaczasz losowo („strzelasz”). Prawdopodobieństwo zaznaczenia poprawnej odpowiedzi w jednym pytaniu wynosi 0.5. Obliczmy prawdopodobieństwo tego, że przynajmniej 7 odpowiedzi będzie prawidłowych.

```
> dbinom(7, size = 20, prob = 0.5) # Obliczamy Pr(X=7),  a chcemy Pr(X>=7)
[1] 0.0739288

> (prawd <- dbinom(7:20, size = 20, prob = 0.5)) # Pr(X=7), Pr(X=8), ..., Pr(X=20)
```

```
 [1] 7.39288e-02 1.20134e-01 1.60179e-01 1.76197e-01 1.60179e-01 1.20134e-01 7.39288e-02
 [8] 3.69644e-02 1.47858e-02 4.62055e-03 1.08719e-03 1.81198e-04 1.90735e-05 9.53674e-07
```

```
> sum(prawd) # Sumujemy -  to jest nasza odpowiedź
[1] 0.942341
```

Możemy to zadanie rozwiązać, wykorzystując dystrybuantę: $F(x) = \mathbb{P}(X \leqslant x)$. W zadaniu mamy obliczyć:

$$\mathbb{P}(X \geqslant 7) = \mathbb{P}(X > 6) = 1 - \mathbb{P}(X \leqslant 6) = 1 - F(6)$$

więc

```
> 1 - pbinom(6, 20, 0.5)
[1] 0.942341
```

Rozkład Poissona: `lambda` – parametr w rozkładzie poissona; nie ma wartości domyślnej

```
dpois(x, lambda)
ppois(x, lambda)
qpois(p, lambda)
rpois(n, lambda)
```

4.3. Zadania

Kombinatoryka

Zad. 1. Wiemy, że pin do karty kredytowej składa się z czterech różnych cyfr: $2, 7, 9, 5$. Niestety nie pamiętamy ich kolejności. Jaka jest maksymalna liczba prób, pozwalająca tę kolejność ustalić?

Zad. 2. Ogrodnik ma 10 sadzonek róż: 5 – białych, 3 – czerwone, 2 – herbaciane. Jeśli musi posadzić je w rzędzie, na ile sposobów może to zrobić.

Zad. 3. Dziecko ma 12 klocków: 5 czarnych, 3 czerwone, 3 białe i 1 zielony. Układa je w kolejności. Na ile sposobów może to zrobić.

Zad. 4. Mamy 7 różnych prezentów, które chcemy rozdzielić między 10 osób. Na ile sposobów możemy to zrobić, jeżeli każda z osób może otrzymać co najwyżej jeden prezent.

Zad. 5. Ile wyrazów można utworzyć ze słowa: BALLADA? Rozwiązać dwoma sposobami – wykorzystując różne schematy kombinatoryczne.

Zad. 6. Studenci kierunku Zarządzania muszą wybrać delegację składającą się z 3 osób. Chęć bycia członkiem takiej delegacji zgłosiło 12 osób. Ile takich delegacji można utworzyć.

Zad. 7. Mamy 6 kanapek, które rozdzielamy między 10 studentów. Na ile sposobów możemy to zrobić?

Zad. 8. Musimy zatrudnić 8 nauczycieli. Na ile sposobów możemy to zrobić, jeżeli rozważamy tylko 4 szkoły. Jak zmieni się liczba możliwości, jeżeli w każdej ze szkół należy zatrudnić dokładnie po 2 nauczycieli?

Prawdopodobieństwo

Zad. 9. Grupa składa się z 20 studentów, w której jest 7 kobiet. Otrzymała ona 6 biletów, które zostały rozdzielone drogą losowania. Jakie jest prawdopodobieństwo tego, że w grupie szczęśliwców znajdą się dokładnie 3 kobiety? Oblicz również prawdopodobieństwo tego, że w grupie znajdzie się przynajmniej jedna kobieta.

Zad. 10. Winda rusza z 7 pasażerami i zatrzymuje się na 10 piętrach. Jakie jest prawdopodobieństwo zdarzenia, że żadnych dwóch pasażerów nie opuści windy na tym samym piętrze?

Zad. 11. Płatności można dokonać dwoma kartami: Visa i MasterCard. Wiemy, że 24 procent klientów ma kartę Visa, a 61 procent ma kartę MasterCard. Z kolei 11 procent ma obie karty. Ile procent klientów ma jakąkolwiek kartę. Potraktuj procenty jak prawdopodobieństwa.

Zad. 12. Prawdopodobieństwo, że cena pewnego towaru pójdzie jutro w górę wynosi 0,3, a prawdopodobieństwo, że cena miedzi pójdzie w górę wynosi 0,2. Wiadomo ponadto, że w 6% przypadków obie ceny – towaru i miedzi – idą w górę. Student twierdzi, że cena towaru i miedzi są zależne, gdyż prawdopodobieństwo wzrostu obu jest dodatnie, równe 0,06. Z kolei studentka uważa, że obie ceny są niezależne. Kto ma rację?

Zad. 13. Szacuje się, że prawdopodobieństwo zawarcie kontraktu z firmą A (zdarzenie A) jest równe 0,45. Uważa się też, że gdyby firma zawarła kontrakt z firmą A, to prawdopodobieństwo zawarcia kontraktu z firmą B (zdarzenie B) należałoby ocenić na 0,90. Jakie jest prawdopodobieństwo, że firma zawrze oba kontrakty?

Zad. 14. W Politechnice Wrocławskiej studiuje prawie 29 tys. osób. Prawdopodobieństwo tego, że losowo wybrany student pierwszego roku będzie miał telefon Samsunga wynosi 0.47. Z kolei prawdopodobieństwo tego, że wybrany student będzie na pierwszym roku wynosi 0.32. Wiemy również, że 56.2% studentów wyższych lat ma telefon Samsunga.

a) Oblicz prawdopodobieństwo tego, że losowo wybrany student będzie miał telefon Samsunga.

b) Zakładając, że student ma telefon Samsunga, oblicz prawdopodobieństwo, że nie jest studentem pierwszego roku?

c) Zakładając, że student nie ma telefonu Samsunga, oblicz prawdopodobieństwo, że nie jest studentem pierwszego roku?

Zmienne losowe

Zad. 15. Dla zestandaryzowanego rozkładu normalnego i rozkładu t-studenta o 15 stopniach swobody

a) wyznacz kwantyle rzędu $p = 0.85$, $p = 0.99$, $p = 0.27$.

b) oblicz prawdopodobieństwa: $\mathbb{P}(X > 1.8)$, $\mathbb{P}(X \geqslant 2.47)$.
Do obliczeń wykorzystaj odpowiednie funkcje R.

Zad. 16. Pewien bank ma atrakcyjny program kart kredytowych. Klienci, którzy spełniają wymagania, mogą otrzymać taką kartę na preferencyjnych warunkach. Analiza danych historycznych pokazała, że 35% wszystkich wniosków zostaje odrzuconych ze względu na niespełnienie wymagań. Załóżmy, że przyjęcie lub odrzucenie wniosku jest zmienna losową o rozkładzie Bernoulliego. Jeśli próbę losową stanowi 20 wniosków jakie jest prawdopodobieństwo tego, że:

a) dokładnie trzy wnioski zostaną odrzucone;

b) 10 wniosków zostanie przyjętych;

c) przynajmniej 10 wniosków zostanie przyjętych.

Zad. 17. Salon samochodowy rejestruje dzienną sprzedaż nowego modelu samochodu *Shinari*. Wyniki obserwacji doprowadziły do wniosku, że rozkład liczby sprzedanych samochodów w ciągu dnia można przybliżyć rozkładem Poissona:

$$\mathbb{P}(X = x|\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}, \qquad x = 0, 1, 2, \ldots$$

z parametrem $\lambda = 5$. Oblicz prawdopodobieństwo tego, że salon:

a) nie sprzeda ani jednej sztuki;
b) sprzeda dokładnie 5 sztuk;
c) sprzeda przynajmniej jedną sztukę;
d) sprzeda przynajmniej 2 sztuki ale mniej niż 5;
e) sprzeda 5 sztuk przy założeniu, że sprzedał już ponad 3 sztuki.

Zad. 18. Niech X i Y będą zmiennymi losowymi o rozkładzie normalny $\mathcal{N}(0,1)$ i $\mathcal{N}(0,4)$ odpowiednio. Bez wykonywania obliczeń podaj wartości prawdopodobieństwa lub zdefiniuj relację ($>, <, =, \geqslant, \leqslant$):

a) $\mathbb{P}(X > 0) = \ldots$

b) $\mathbb{P}(Y > 0) = \ldots$

c) $\mathbb{P}(X > 2) \ldots \mathbb{P}(X \leqslant -2)$

d) $\mathbb{P}(0 < Y \leqslant 3) \ldots \mathbb{P}(2 < Y < 5)$

e) $\mathbb{P}(-1 < X \leqslant 1) \ldots \mathbb{P}(0 < X \leqslant 2)$

f) $\mathbb{P}(0 < Y \leqslant 2) \ldots \mathbb{P}(0 < X \leqslant 2)$

g) $\mathbb{P}\left(\frac{3\ln(2)}{8} < Y < 1.5\right) \ldots \mathbb{P}\left(\frac{3\ln(2)}{8} < X < 1.5\right)$

Sprawdź swoje rozwiązania obliczając powyższe prawdopodobieństwa. Użyj odpowiednich funkcji R.

ROZDZIAŁ 5

Przekształcanie i analiza danych z wykorzystaniem narzędzi *tidyverse*

5.1. Sześć podstawowych funkcji pakietu `dplyr`

Ramkę danych możemy dowolnie przekształcać, np.:

- wybrać wiersze lub kolumny – to już omówiliśmy w rozdz. 3.2);
- sortować wiersze w porządku rosnącym albo malejącym;
- tworzyć nowe zmienne (kolumny);
- agregować zmienne z wykorzystaniem statystyk opisowych (np. średniej, mediany itp.).

Do wykonania tych operacji wystarczy 6 funkcji z pakietu `dplyr` – znasz już 2 z nich. Jednak wszystkie zamieszczam w tab. 5.1.

Tabela 5.1. Podstawowe funkcje z pakietu `dplyr`

Funkcja	Opis
`select(rd, zm1, zm5)`	Wybiera z ramki danych `rd` zmienne `zm1` i `zm5`. Zapisując `zm1:zm5` wybieramy te zmienne oraz wszystkie będące między nimi. Dając minus przed nazwą, wykluczamy ją.
`filter(rd, war_1, war_2)`	Wybiera te wiersze, które jednocześnie spełniają warunki: `war_1` i `war_2`. Przecinek pełni rolę operatora logicznego: `&`. Możemy użyć operatora lub: `|`.
`mutate(rd, nowaZm = ...)`	Dodaje zmienną o nazwie `nowaZm`. W miejsce kropek wstawiamy wyrażenie tworzące zmienną, np. `zm1/2`.
`arrange(rd, zm2, desc(zm4))`	Sortuje wiersze w porządku rosnącym dla zmiennej `zm2` i malejącym dla zmiennej `zm4`.
`summarise(rd, statOpis = fun(zm3))`	Oblicza wartość statystyki opisowej `fun` dla zmiennej `zm3` i zapisuje ją pod nazwą `statOpis`, np. aby obliczyć średnią: `mean(zm3)`
`group_by(rd, zm3, zm6)`	Dzieli ramkę danych na grupy, jakie tworzą kombinacje poziomów 2 zmiennych kategorialnych: `zm3`, `zm6`. Jeśli pierwsza ma 2 poziomy, a druga 3, to mamy 6 grup.

Na tej liście znajduje się funkcja `group_by()`, która pozwala wykonać przekształcenia w podziale na grupy. Na przykład chcesz policzyć średnią wartość wynagrodzenia, ale w podziale na poziomy wykształcenia (podstawowe, średnie, wyższe). Wtedy przed krokiem, w którym liczysz tą średnią, umieszczasz funkcję grupującą. Jej argumentem będzie nazwa zmiennej (albo zmiennych).

Zapamiętaj, bo to ważne: funkcję `group_by()` stosujesz do zmiennych kategorialnych. Możesz też wykorzystać ją do zmiennych ilościowych dyskretnych (licznikowych), jeżeli liczba unikalnych wartości jest nieduża. Przykładem jest zmienna mówiąca o liczbie dzieci czy też kart kredytowych w gospodarstwie domowym. Na pewno nie użyjesz tej funkcji do zmiennej wynagrodzenie lub stopa bezrobocia.

5.2. Dodawanie zmiennych do ramki danych

Do ramki danych dodajemy zmienne lub nadpisujemy już istniejące przy użyciu funkcji `mutate()`. Nowe kolumny pojawią się na końcu ramki danych. Ciekawą jej cechą, i wcale nie taką oczywistą, jest możliwość tworzenia zmiennych na bazie aktualnie tworzonych. Pokażę to na przykładzie i poniższej ramce danych.

```
> # Tworzymy ramkę danych: badanie
> pracownik <- c("kierownik", "wykonawczy", "wykonawczy", "kierownik")
> rokUrodz <- c(1987, 1975, 1997, 1970)
> wynUSD <- c(57000, 40200, 21450, 21900)
> wynPoczUSD <- c(27000, 18750, 12000, 13200)
> badanie <- data.frame(pracownik, rokUrodz, wynUSD, wynPoczUSD)
> badanie
   pracownik rokUrodz wynUSD wynPoczUSD
1  kierownik     1987  57000      27000
2 wykonawczy     1975  40200      18750
3 wykonawczy     1997  21450      12000
4  kierownik     1970  21900      13200
```

Do ramki danych `badanie` dodamy kolejne zmienne, a powstałą w ten sposób ramkę, zapiszemy pod nazwą `badanieNew`. Wykonamy następujące kroki:

- Utworzymy zmienną `wiek`, wykorzystując rok urodzenia (`rokUrodz`).
- Utworzymy zmienną `wynDiffUSD`, której wartości są różnicą między obecnym wynagrodzeniem (`wynUSD`), a wynagrodzeniem początkowym (`wynPoczUSD`).
- Zmienną utworzoną w poprzednim punkcie wyrazimy w PLN – przyjmując kurs wymiany na poziomie 3.7 PLN/USD – i zapiszemy jako zmienną `wynDiffPLN`.

Przeanalizuj poniższe rozwiązanie. Zwróć uwagę na wspomnianą przeze mnie cechę, która pozwala utworzyć zmienną `wynDiffPLN` na bazie aktualnie tworzonej `wynDiffUSD`.

```
> badanieNew <- mutate(badanie, wiek = 2019 - rokUrodz,
+                              wynDiffUSD = wynUSD - wynPoczUSD,
+                              wynDiffPLN = 3.7 * wynDiffUSD)
> badanieNew
   pracownik rokUrodz wynUSD wynPoczUSD wiek wynDiffUSD wynDiffPLN
1  kierownik     1987  57000      27000   32      30000     111000
2 wykonawczy     1975  40200      18750   44      21450      79365
3 wykonawczy     1997  21450      12000   22       9450      34965
4  kierownik     1970  21900      13200   49       8700      32190
```

Użyj ramki danych `badanie` i wykonaj podobne zadanie. Dodaj zmienną: (a) wiek (jaki mamy teraz rok?), (b) wynagrodzenie wyrażone w euro, (c) wzrost procentowy wynagrodzenia. Użyj funkcji `arrange()` i posortuj rosnąco utworzoną zmienną w pkt. (c). Tak zmodyfikowaną ramkę zapisz pod nazwą `badanieNowe`.

5.3. Rekodowanie zmiennych

Każdą zmienną kategorialną możemy zakodować na kilka sposobów. Przykładowo dla zmiennej płeć mamy 3 możliwości. Pierwsza – przyjąć: `k` i `m`; druga – użyć wartości liczbowych, np. `0` i `1`, trzecia – wziąć pełne nazwy: `kobieta` i `mężczyzna`. Jeżeli korzystasz z gotowych zbiorów danych, to warto znać sposób zmiany schematu kodowania. Pojawiają się również sytuacje, w których niektóre z poziomów chcemy połączyć, np. uważamy, że liczba poziomów dla stanowiska zatrudnienia jest zbyt duża. Właśnie rekodowanie, jak sama nazwa wskazuje, polega na zmianie sposobu kodowania zmiennej.

Niemal każdą operację w R możesz przeprowadzić na wiele różnych sposobów – dotyczy to również rekodowania, które wykonasz za pomocą funkcji tworzącej czynniki `factor()`. Omawiałem ją w rozdz. 2.4. Często łatwiej i szybciej poradzimy sobie z tym procesem, jeżeli użyjemy funkcji `recode()` lub `recode_factor()` z pakietu `dplyr`. W poniższej składni pierwszym argumentem jest wektor, w którym przechowywane są wartości zmiennej rekodowanej.

```
recode(zmienna, "obecny_1" = "nowy_1", "obecny_2" = "nowy_2",.default = NULL)
recode_factor(zmienna, "obecny_1" = "nowy_1", "obecny_2" = "nowy_2",
    .ordered = TRUE, .default = NULL)
```

Wartości `obecny` chcemy rekodować (zamienić) na `nowy`. Łańcuch znaków ujmujemy w cudzysłowy. Jeżeli wartością obecną są liczby, to zamiast cudzysłowów, musimy użyć odwróconego apostrofu `` ` ``, np: `` `2` ``. Pierwsza z funkcji zachowuje porządek pierwotnej zmiennej, jeśli jest ona porządkowa. Druga natomiast, ma dodatkowy argument `.ordered`, który pozwala na zmianę tego porządku – porządek ten ustala kolejność, w jakiej wpisujemy wartości `nowy`. Ostatni argument zmienimy z `NULL` na własną wartość, gdy wszystkie pozostałe, niewymienione poziomy mają ją przyjąć. Przykładowo możemy napisać: `.default = "inne"`.

Pora na przykłady. Mamy zmienną typu czynnik (*factor*) o następujących poziomach

```
> ## Losujemy 10 liter ze zbioru: a,b,c
> set.seed(777)
> owoc <- factor(sample(letters[1:3], 10, replace = TRUE))
> owoc
 [1] a b b b c b a a b c
Levels: a b c
```

które chcemy zrekodować na poziomy odpowiadające nazwom owoców: arbuz, banan i czereśnia:

```
> ## Rekodujemy, wykorzystując pakiet dplyr
> library(dplyr)
> recode(owoc, "a" = "arbuz", "b" = "banan", "c" = "czereśnia")
 [1] arbuz     banan     banan     banan     czereśnia banan     arbuz     arbuz     banan
[10] czereśnia
Levels: arbuz banan czereśnia
```

Powyższą funkcję możemy też wykorzystać do połączenia kategorii, np.

```
> ## łączymy kategorie:  a i b - owoc południowy, c - owoc krajowy
> recode(owoc, "a" = "owoc południowy", "b" = "owoc południowy", "c" = "owoc krajowy")
 [1] owoc południowy owoc południowy owoc południowy owoc południowy owoc krajowy
 [6] owoc południowy owoc południowy owoc południowy owoc południowy owoc krajowy
Levels: owoc południowy owoc krajowy

> ## II sposób: przypisuję, aby nie wyświetlać wyniku
```

```
> temp <- recode(owoc, "c" = "owoc krajowy", .default = "owoc południowy")
```

Jeżeli rekodowana zmienna ma wartości liczbowe, wtedy ujmujemy je w odwrócony cudzysłów ``. W poniższym przykładzie mamy zmienną:

```
> (wykszt <- factor(sample(1:3, 10, replace = TRUE)))
 [1] 2 1 3 3 3 2 2 2 2 3
Levels: 1 2 3
```

która przyjmuje wartości: 1, 2 i 3. Oznaczają one odpowiednio wykształcenie: podstawowe, średnie i wyższe. Wiemy, że zmienna wykształcenie jest zmienną porządkową, więc ten fakt uwzględnimy w rekodowaniu i wykorzystamy drugą ze wspomnianych funkcji. Zwróć uwagę na dodatkowy argument.

```
> recode_factor(wykszt, `1` = "podstawowe", `2` = "średnie", `3` = "wyższe",
+               .ordered = TRUE)
 [1] średnie    podstawowe wyższe     wyższe     wyższe     średnie    średnie    średnie
 [9] średnie    wyższe
Levels: podstawowe < średnie < wyższe
```

W ostatnim przykładzie zrekodujemy numery miesiąca na odpowiadające im kwartały. Najpierw wygenerujemy dane.

```
> # Tworzymy wektor miesiac
> set.seed(123)
> (miesiac <- sample(1:12, 30, replace=TRUE))
 [1]  3  3 10  2  6 11  5  4  6  9 10 11  5  3 11  9 12  9  9  3  8 10  7 10  9  3  4  1 11  7
```

Chcemy zastąpić wartości od 1 do 3: `I_kw`, od 4 do 6: `II_kw` itd. Całą procedurę rekodowania musimy poprzedzić podziałem zmiennej `miesiac` na kategorie odpowiadające kwartałom. Użyjemy funkcji `cut()`. Spróbuj

```
> (miesiacKat <- cut(miesiac, breaks = c(0,3,6,9,12))) # breaks - punkty podziału
 [1] (0,3]  (0,3]  (9,12] (0,3]  (3,6]  (9,12] (3,6]  (3,6]  (3,6]  (6,9]  (9,12] (9,12] (3,6]
[14] (0,3]  (9,12] (6,9]  (9,12] (6,9]  (6,9]  (0,3]  (6,9]  (9,12] (6,9]  (9,12] (6,9]  (0,3]
[27] (3,6]  (0,3]  (9,12] (6,9]
Levels: (0,3] (3,6] (6,9] (9,12]
```

Właściwa procedura rekodowania wygląda następująco.

```
> recode_factor(miesiacKat, "(0,3]" = "I_kw", "(3,6]" = "II_kw",
+     "(6,9]" = "III_kw", "(9,12]" = "IV_kw", .ordered = TRUE)
 [1] I_kw   I_kw   IV_kw  I_kw   II_kw  IV_kw  II_kw  II_kw  II_kw  III_kw IV_kw  IV_kw  II_kw
[14] I_kw   IV_kw  III_kw IV_kw  III_kw III_kw I_kw   III_kw IV_kw  III_kw IV_kw  III_kw I_kw
[27] II_kw  I_kw   IV_kw  III_kw
Levels: I_kw < II_kw < III_kw < IV_kw
```

Masz wektor z następującymi wartościami (skopiuj to do swojego skryptu):

```
uczycLubie <- c(5, 4, 1, 2, 3, 2, 1)
```

W nim przechowujesz odpowiedzi na pytanie z kwestionariusza ankiety: czy lubisz się uczyć? Wykorzystaj funkcję `recode()` i zrekoduj tą zmienną według schematu: 1 – nie, 2 – raczej nie, 3 – średnio, 4 – raczej tak, 5 – tak. Po rekodowaniu wyświetl zmienną `uczycLubie`.

5.4. Obliczanie statystyk opisowych

W tym i następnych rozdziałach będziemy posługiwali się danymi `bankFull.csv`. Zostały one zebrane przez pewien bank portugalski, podczas kampanii marketingu bezpośredniego. Pochodzą z *UCI machine learning repository*. Wczytaj je i zapisz w obiekcie o nazwie `bank`. Zamieszczam tabelę z oryginalnym opisem zmiennych. Tłumaczeniem zajmę się bezpośrednio w tekście.

Tabela 5.2. Zmienne zbioru danych bank

Zmienna	Opis
age	(numeric)
job	(categorical) type of job: "admin.","unknown","unemployed","management","housemaid", "entrepreneur", ...
marital	(categorical) marital status: "married","divorced","single"; note: "divorced" means divorced or widowed
education	(categorical) "unknown","secondary","primary","tertiary"
default	(binary) has credit in default? "yes","no"
balance	(numeric) average yearly balance, in euros
housing	(binary) has housing loan? "yes","no"
loan	(binary) has personal loan? "yes","no"
contact	(categorical) contact communication type: "unknown","telephone","cellular"
day	(numeric) last contact day of the month
month	(categorical) last contact month of year: "jan", "feb", "mar", ..., "nov", "dec"
duration	(numeric) last contact duration, in seconds
campaign	(numeric) number of contacts performed during this campaign and for this client, includes last contact
pdays	(numeric) number of days that passed by after the client was last contacted from a previous campaign, 1 means client was not previously contacted
previous	(numeric) number of contacts performed before this campaign and for this client
poutcome	(categorical) outcome of the previous marketing campaign: "unknown","other","failure","success"
y	(binary) has the client subscribed a term deposit? "yes","no"

5.4.1. Statystyki dla zmiennych ilościowych.

W tej części omówię dwie, ostatnie funkcje z tabeli 5.1: `summarise()` i `group_by()`. Z kolei statystyki podsumowujące znajdziesz w tabeli 5.3.

Przypomnę, że widoczne w funkcjach wartości argumentów są domyślne. Jeśli mają mieć taką domyślną wartość, to możesz ją pominąć. Przykładowo zapis `mean(x)` jest równoważny zapisowi `mean(x, na.rm = FALSE)`. Jeśli w zbiorze pojawi się choćby jedna brakująca wartość, wtedy wywołanie funkcji z takimi argumentami zwraca `NA`. Działanie to prezentuję poniżej.

```
> ## domyślnie na.rm=FALSE
> x <- c(1, 5, 3, NA, 4, NA, 7)
> mean(x)
[1] NA
```

```
> ## Zmieniamy na: na.rm=TRUE
> ## każąc wyrzucić braki z obliczeń
> mean(x, na.rm = TRUE)
[1] 4
```

Zanim przyjdziemy do zasadniczych przykładów, obliczmy kurtozę, współczynnik asymetrii oraz kwantyle rzędu: 0.25, 0.30, 0.45, 0.95 dla zmiennej saldo rachunku (`balance`) z ramki danych `bank` (zob. tab. 5.2).

```
> ##  Zbiór danych: bank
> library(e1071) # kurtoza i asymetria
> skewness(bank$balance, type = 2)
[1] 8.36031
```

Tabela 5.3. Funkcje statystyk opisowych

Funkcja	Opis
`mean(x, trim = 0, na.rm = FALSE)`	Średnia lub średnia ucięta (`trim` - odsetek pominiętych) z wektora `x`; gdy `na.rm = TRUE`, wtedy brakujące dane są pomijane w obliczeniach
`median(x, na.rm = FALSE)`	Mediana z wektora `x`
`var(x, na.rm = FALSE)`	Wariancja z wektora `x`
`sd(x, na.rm = FALSE)`	Odchylenie standardowe z wektora `x`
`quantile(x, probs=seq(0, 1, 0.25), na.rm = FALSE)`	Kwantyle rzędu: 0, 0.25, 0.5, 0.75, 1
`IQR(x, na.rm = FALSE)`	Rozstęp międzykwartylowy, czyli różnica: `quantile(x, 0.75) - quantile(x, 0.25)`
`skewness(x, na.rm = FALSE, type = 3)`	Współczynnik asymetrii z pakietu `e1071`. Typ 3 (domyślny) używany jest w programie MINITAB i BMDP, natomiast typ 2 pojawia się w programach IBM SPSS i SAS.
`kurtosis(x, na.rm = FALSE, type = 3)`	Współczynnik spłaszczenia z pakietu `e1071`.
`cor(x, y)`	Współczynnik korelacji Pearsona między zmiennymi `x` i `y`. Jeżeli nie podamy `y`, wtedy `x` musi mieć przynajmniej 2 kolumny. Możemy obliczyć inne współczynniki ustawiając argument `method = "kendall"` lub `"spearman"`. Braki danych obsługujemy argumentem `use` z jedną wartością: `all.obs`, `complete.obs`, `na.or.complete`, `pairwise.complete.obs`.
`summary(x)`	Podaje wartości: największą, najmniejszą, średnią, medianę, pierwszy i trzeci kwartyl, liczbę braków danych

```
> kurtosis(bank$balance, type = 2)
[1] 140.752

> # percentyle: 25, 30, 45,  i 95 dla zm. balance
> quantile(bank$balance, probs = c(0.25, 0.3, 0.45, 0.95))
 25%  30%  45%  95%
  72  131  352 5768
```

Wiesz już, w jaki sposób używać funkcji statystyk opisowych, gdy argumentem wejściowym jest wektor. Teraz możemy przejść do funkcji `summarise()`. W następnym przykładzie policzymy wartości następujących statystyk: średniej, mediany, asymetrii i kurtozy dla zmiennej saldo rachunku (`balance`). Są dwie wersje tego przykładu: z operatorem potokowym (zob. str. 29) i bez niego.

```
> ## Z wykorzystaniem operatora %>%
> bank %>%
+   summarise(srednia = mean(balance),
+     mediana = median(balance),
+     wspAsymetrii = skewness(balance),
+     wspSplaszcz = kurtosis(balance))
  srednia mediana wspAsymetrii wspSplaszcz
1    1362     448         8.36       140.7
```

```
> ## Bez operatora %>%
> summarise(bank,
+     srednia = mean(balance),
+     mediana = median(balance),
+     wspAsymetrii = skewness(balance),
+     wspSplaszcz = kurtosis(balance))
  srednia mediana wspAsymetrii wspSplaszcz
1    1362     448         8.36       140.7
```

Zauważ, że jeśli nie używamy operatora `%>%`, wtedy na pierwszej pozycji `summarise()` musi znaleźć się nazwa ramki danych. W dalszej kolejności pojawiają się interesujące nas funkcje, które poprzedzamy wymyślonymi nazwami kolumn.

Sama funkcja `summarise()` nie byłaby specjalnie użyteczna, gdyby jej wywołanie ograniczało się do tak prostych zadań jak w powyższym przykładzie. Jej elastyczność docenimy, gdy użyjemy funkcji `group_by()` z tego samego pakietu. Funkcja ta tworzy grupy składające się z poziomów zmiennych kategorialnych. Przykładowo jeżeli jej argumentem będzie nazwa

zmiennej odnoszącej się do edukacji (`education`), wtedy R zapamięta, że funkcję `summarise()` należy zastosować do każdego poziomu tej zmiennej, czyli do każdego poziomu wykształcenia. Tutaj również obowiązuje ta sama składnia: jeżeli nie korzystasz z operatora `%>%`, wtedy pierwszym argumentem `group_by()` jest nazwa ramki danych. Zobaczmy jak wyglądają statystyki dla salda rachunku w zależności od poziomu edukacji:

```
> ## Z wykorzystaniem operatora %>%
> bank %>%
+   group_by(education) %>%
+   summarise(srednia = mean(balance),
+             mediana = median(balance),
+     wspAsymetrii = skewness(balance))
# A tibble: 4 x 4
  education srednia mediana wspAsymetrii
  <chr>       <dbl>   <dbl>        <dbl>
1 primary     1251.     403         8.85
2 secondary   1155.     392         8.53
3 tertiary    1758.     577         7.43
4 unknown     1527.     568         7.73
```

```
> ## Bez operatora %>%
> krok1 <- group_by(bank, education)
> summarise(krok1,
+     srednia = mean(balance),
+     mediana = median(balance),
+     wspAsymetrii = skewness(balance))
# A tibble: 4 x 4
  education srednia mediana wspAsymetrii
  <chr>       <dbl>   <dbl>        <dbl>
1 primary     1251.     403         8.85
2 secondary   1155.     392         8.53
3 tertiary    1758.     577         7.43
4 unknown     1527.     568         7.73
```

5.4.2. Statystyki dla zmiennych kategorialnych. Podstawowa analiza danych kategorialnych sprowadza się do zliczania wystąpień każdej kategorii. Umiesz już taką operację wykonać, bo znasz funkcję `table()`. Będziemy jednak kontynuować pracę z pakietem `dplyr`. Pokażę ci, w jaki sposób oprócz częstości umieszczać dodatkowo w tabeli tzw. częstości względne (odsetki, procenty). Nauczysz się również budować tabele kontyngencji, zwane tabelami krzyżowymi lub wielodzielczymi. Te umiejętności przydadzą ci się, gdy przejdziemy do tworzenia wykresów.

Aby zbudować tabelę składającą się wyłącznie z liczebności, musimy najpierw użyć funkcji grupującej `group_by()`, której argumentem będzie nazwa zmiennej lub zmiennych, dla których tworzymy tabelę[1]. W drugim kroku wykorzystamy funkcję `summarise()`, wewnątrz której pojawi się funkcja specjalna `n()`. Jej rola sprowadza się do zliczenia wystąpień każdej kategorii zmiennej lub kombinacji kategorii – jeżeli mamy więcej zmiennych. Zapamiętaj, że możesz jej użyć tylko wewnątrz funkcji: `summarise()`, `mutate()` i `filter()`.

Zbudujemy tabelę dla zmiennej edukacja (`education`) oraz tabelę krzyżową dla dwóch zmiennych: edukacja (`education`) i zgoda na lokatę terminową (`y`). Obie zmienne znajdują się w ramce danych `bank`.

[1] Szybciej otrzymasz liczebności, jeśli użyjesz funkcji `count()`. Przykładowo dla zmiennej wykształcenie kod ma postać: `count(bank, education)`. To podejście jednak nie jest tak elastyczne, jak jednoczesne użycie `group_by()` i `summarise()`. Czy po lekturze tego rozdziału potrafisz to uzasadnić na przykładzie?

```
> ## Tabela 1: education
> bank %>%
+     group_by(education) %>%
+     summarise(ile = n())
# A tibble: 4 x 2
  education   ile
  <chr>     <int>
1 primary    6851
2 secondary 23202
3 tertiary  13301
4 unknown    1857
```

```
> ## Tabela 2: education i y
> bank %>%
+     group_by(education, y) %>%
+     summarise(ile = n())
# A tibble: 8 x 3
# Groups:   education [4]
  education y       ile
  <chr>     <chr> <int>
1 primary   no     6260
2 primary   yes     591
3 secondary no    20752
4 secondary yes    2450
5 tertiary  no    11305
6 tertiary  yes    1996
7 unknown   no     1605
8 unknown   yes     252
```

Zauważ, że w wyniku operacji zwrócona została ramka danych. To oznacza, że możesz do niej dodawać kolejne zmienne za pomocą funkcji `mutate()`. Korzystając z tej możliwości, dodamy odsetki. Zapamiętaj, że sumują się do 1 w obrębie kategorii ostatniej zmiennej, jaka pojawia się wewnątrz funkcji `group_by()`. Która ze zmiennych powinna być ostatnia, zależy wyłącznie od ciebie. Przeanalizuj poniższe przykłady pod kątem sumowania do 1.

```
> ## Tabela 1: sumowanie do 1 dla y
> bank %>%
+     group_by(education, y) %>%
+     summarise(ile = n()) %>%
+     mutate(odset = ile/sum(ile))
# A tibble: 8 x 4
# Groups:   education [4]
  education y       ile  odset
  <chr>     <chr> <int>  <dbl>
1 primary   no     6260 0.914
2 primary   yes     591 0.0863
3 secondary no    20752 0.894
4 secondary yes    2450 0.106
5 tertiary  no    11305 0.850
6 tertiary  yes    1996 0.150
7 unknown   no     1605 0.864
8 unknown   yes     252 0.136
```

```
> ## Tabela 2: sumowanie do 1 dla education
> bank %>%
+     group_by(y, education) %>%
+     summarise(ile = n()) %>%
+     mutate(odset = ile/sum(ile))
# A tibble: 8 x 4
# Groups:   y [2]
  y     education   ile  odset
  <chr> <chr>     <int>  <dbl>
1 no    primary    6260 0.157
2 no    secondary 20752 0.520
3 no    tertiary  11305 0.283
4 no    unknown    1605 0.0402
5 yes   primary     591 0.112
6 yes   secondary  2450 0.463
7 yes   tertiary   1996 0.377
8 yes   unknown     252 0.0476
```

Rozszerz którąś z tabel o dodatkową kolumnę, która będzie pokazywała procenty, np. 91.4 (bez znaku %). Dodatkowo zaokrąglij te wartości do jednego miejsca po przecinku.

5.4.3. Przykład wykorzystania funkcji z pakietu `dplyr`.

Na przykładzie zbioru danych `bank` pokażę ci, w jaki sposób poznane funkcje z pakietu `dplyr()` pozwalają na przeprowadzenie analizy wymagającej wielu kroków. W rozwiązaniu, które znajdziesz poniżej, zwróć uwagę na operator potokowy – idealnie wpisuje się w sekwencję tych działań. Spróbuj samodzielnie wykonać poniższe kroki analizy, a dopiero później zapoznaj sie z rozwiązaniem. Nie ma lepszego sposobu na naukę. A to lista kroków:

1. Interesują nas te obserwacje (klienci), dla których wynik ostatniej kampanii marketingowej (poutcome) zakończył się sukcesem lub porażką. Pozostałe poziomy wyrzucamy, bo okazuje się, że takie są.
2. Wszystkie obliczenia wykonujemy w podziale na zmienne: edukacja (education), wynik ostatniej kampanii marketingowej (poutcome) oraz zgoda na lokatę terminową (y).
3. Obliczamy liczebność (Liczebnosc) i procenty (Procent) dla zmiennych z punktu 2, a dla zmiennej saldo na rachunku (balance) liczymy średnią i medianę.
4. W interpretacji i porównaniu otrzymanych statystyk chcemy ograniczyć się tylko do tych wartości, którym odpowiada zgoda na lokatę terminową – dlatego pozostałe usuwamy.
5. Dla ułatwienia porównania sortujmy kolumnę z wartościami procentowymi Procent.
6. Zawężamy wyniki do wartości większych od 50 dla kolumny Procent.

W analizie tego przykładu zachęcam do uruchamiania poniższego kodu partiami i obserwowania, jak zmieniają się wyniki w zależności od dodawania kolejnych linijek.

```
> bank %>%
+  filter(poutcome=="failure" | poutcome == "success") %>% # wybierz: sukces lub porażka
+   group_by(education, poutcome, y) %>%
+   summarise(sredniaSaldo = mean(balance),
+             medianaSaldo = median(balance),
+             Liczebnosc = n()) %>%
+   mutate(Procent = 100*Liczebnosc/sum(Liczebnosc)) %>%
+   filter(y == "yes") %>%                                  # weź tylko yes
+  select(-y) %>%                                  # skoro y=yes, to wyrzuć zmienną y
+   arrange(desc(Procent)) %>%                              # posortuj malejąco
+   filter(Procent > 50)
# A tibble: 4 x 6
# Groups:   education, poutcome [4]
  education poutcome sredniaSaldo medianaSaldo Liczebnosc Procent
  <chr>     <chr>           <dbl>        <dbl>      <int>   <dbl>
1 unknown   success         2211.          973         55    67.9
2 tertiary  success         2302.          925        409    65.8
3 secondary success         1610.          703        433    64.1
4 primary   success         2487.         1388         81    60.9
```

Zinterpretujmy pierwszy wiersz: wśród osób o nieznanej edukacji (education), które na wcześniejszą kampanię marketingową odpowiedziały pozytywnie (poutcome), prawie 68% wyraziło zgodę na utworzenie lokaty terminowej (y). Czyżby nadwyżka gotówki na koncie (balance) w porównaniu z innymi? Wynik należy interpretować z pewną ostrożnością, gdyż liczebność tej grupy jest niewielka.

Wykorzystaj ramkę danych bank do porównania czasów rozmów (contact) prowadzonych przez telefon (telephone) i telefon komórkowy (cellular). W porównaniu wykorzystaj czasy wyrażone w minutach (w pliku są w sekundach) i oblicz: średnią, medianę, kwartyle, rozstęp międzykwartylowy i odchylenie standardowe. Powtórz analizę w podziale na wybrane zmienne kategorialne. Może jakaś grupa osób jest szczególnie rozgadana?

5.5. Zadania

Zad. 1. W zbiorze danych `WholesaleCustomers.txt` mamy kolumny odpowiadające: kanałowi dystrybucji, regionom oraz rocznym wydatkom na produkty świeże, mleczne, spożywcze itd.

a) Oblicz medianę rocznych wydatków dla wszystkich produktów, ze względu na kanał (`Channel`) i region (`Region`).

b) Oblicz wartości tych statystyk opisowych, które pozwalają powiedzieć coś o rozkładzie wydatków na produkty mleczne i spożywcze. Wykonaj obliczenia dla każdego regionu, a wyniki porównaj.

c) Oblicz co dwudziesty percentyl wydatków na wszystkie produkty z uwzględnieniem kanału dystrybucji. Dane posortuj tak, aby powstała tabela pozwoliła porównać regiony ze sobą. Wyciągnij wnioski.

d) Zbuduj tabelę przedstawiającą kanały dystrybucji i region.

e) Czy występuje zależność między produktami ze względu na wielkość wydatków? Posłuż się odpowiednią miarą korelacji.

Zad. 2. Do realizacji poniższych punktów wykorzystaj dane `mieszkania.txt`.

a) Dodaj do ramki danych kolumnę o nazwie `cena_m2` – to cena za metr kwadratowy.

b) Ile ofert jest w każdym mieście? Policz odsetki.

c) Zbuduj tabelę, w której znajdzie się rozkład liczby ofert ze względu na rok i miasto. Dodaj kolumnę odsetek ofert. W obrębie której kategorii odsetki powinny sumować się do 1?

d) Stwórz taką samą tabelę jak w pkt. c) z tą różnicą, że każde miasto będzie miało ten sam zakres lat. Następnie przekształć powstałą ramkę tak, aby w kolumnach znalazły się nazwy miast, a wartościami były odsetki zaokrąglone do 2 miejsc po przecinku. Wnioski? Spójrz na rezultat:

```
# A tibble: 3 x 4
    rok Krakow Warszawa Wroclaw
  <int>  <dbl>    <dbl>   <dbl>
1  2009   0.03     0.06    0.02
2  2010   0.16     0.21    0.17
3  2011   0.81     0.73    0.81
```

e) Które miasto ma najdroższe mieszkania? Czy dynamika cen w każdym mieście jest podobna? Jak silna jest zależność między miastami w odniesieniu do dynamiki cen?

Zad. 3. Zaimportuj zbiór danych `AutoSprzedam.dat` do R. Przypisz go do obiektu o nazwie `auto` – tam ma się nazywać ramka danych.

a) Zmień nazwę pierwszej kolumny `NrOferty`.

b) Stwórz ramkę danych `df_niePolska`, w której oferty nie będą uwzględniały Polski jako kraju pochodzenia samochodu.

c) Stwórz ramkę danych o nazwie `df_kraje3`, w której oferty będą uwzględniały tylko 3 najczęściej występujące kraje pochodzenia (pamiętaj o usunięciu nieużywanych poziomów czynnika – `droplevels()`). Powstałą ramkę zapisz do pliku. Sprawdź poprawność importu.

d) Utwórz ramkę danych `df_kolor`, która zawiera tylko samochody w kolorze `czarny-metallic`. Usuń kolumnę odnoszącą się do koloru samochodu.

e) Dodaj do ramki danych `auto` zmienną, która będzie ceną sprzedaży w EUR, z dokładnością do jednego miejsca po przecinku. Kurs wymiany to 4.31 PLN/EUR.

f)★ Utwórz ramkę danych `df_akcyza`, która będzie składała się z 4 zmiennych: `PojemnoscSkokowa`, `CenaPLN`, `Akcyza` oraz `CenaAkcyza`. Pierwsze 2 zmienne występują w oryginalnym zbiorze danych `auto`. Z kolei akcyzę (`Akcyza`) wyliczamy według następującego schematu: samochody o pojemności nie przekraczającej 2000 cm^3 są opodatkowane stawką 3.1%, powyżej tej pojemności obowiązuje stawka 18.6%. Ostatnia ze zmiennych `CenaAkcyza` jest sumą ceny i akcyzy.

Uwzględnij następujący fakt: akcyza dotyczy tylko samochodów sprowadzanych z zagranicy. Dlatego musi być równa 0, jeśli `KrajPochodzenia` to Polska. Wyniki dla pierwszych 6 wierszy:

```
  KrajPochodzenia PojemnoscSkokowa CenaPLN Akcyza CenaAkcyza
1          Niemcy             1900   27900    865      28765
2          Polska             2000   28000      0      28000
3          Polska             1781   25500      0      25500
4          Polska             1991   29900      0      29900
5         Francja             2946   29800   5543      35343
6          Niemcy             1800   21400    663      22063
```

g) Ze zbioru danych `auto` usuń te obserwacje, dla których `RodzajPaliwa` to: `hybryda` lub `napęd elektryczny`. Następnie zrekoduj tą zmienną na dwa poziomy: benzyna i olej napędowy.

ROZDZIAŁ 6 Wizualizacja danych z pakietem ggplot2

Rdzeniem silnika graficznego w R jest pakiet `grDevices`. Daje on podstawowe możliwości w odniesieniu do kolorów i czcionek oraz wyboru formatu wyjściowego grafiki. Na nim z kolei bazują dwa systemy (pakiety) graficzne: system tradycyjny `graphics` oraz system `grid`. Różnią się od siebie tak bardzo, że tworzą odrębne światy graficzne języka R. W tym opracowaniu skupię się na drugim systemie[1].

Na bazie systemu `grid` powstał pakiet `lattice` i `ggplot2`. `lattice` w swoim czasie zyskał dużą popularność z dwóch powodów. Po pierwsze – w odróżnieniu od systemu tradycyjnego wiele elementów wykresu jest formatowanych. Dlatego wykresy wyglądają lepiej i możemy ich użyć bezpośrednio np. w publikacji. Po drugie – relatywnie łatwo utworzymy wykresy z wieloma zmiennymi (tzw. wykresy warunkowe).

W pakiecie `ggplot2` zaimplementowano koncepcję gramatyki grafiki Wilkinsona – stąd w początkowej nazwie widzimy `gg`. Gramatyka to zbiór reguł mówiących o zasadach tworzenia grafiki. Odpowiada ona na pytanie: czym jest grafika statystyczna. Spójna koncepcja i łatwość tworzenia wykresów sprawiają, że użytkownicy języka R najczęściej sięgają po ten pakiet. Dlatego w tym rozdziale poznasz podstawy tworzenia wykresów z wykorzystaniem tego pakietu.

6.1. Schemat budowy wykresu

Budowa wykresu sprowadza się do nakładania kolejnych warstw. Do nich zaliczamy m.in.: **estetykę**, **geometrię**, **skale**, **współrzędne** i **panele**. Takie sekwencyjne podejście do rysowania wykresu powoduje, że kolejna warstwa może zasłonić, bądź zmienić, elementy warstwy wcześniejszej. Poniżej znajdziesz krótki opis każdej z warstw[2]. Zwróć szczególną uwagę na pierwsze dwie warstwy, które są niezbędne do utworzenia wykresu. Jeśli jednak chcesz mieć większą kontrolę nad ostatecznym wyglądem wykresu, zapoznaj się z pozostałymi – potraktuj je jako materiał nieobowiązkowy.

Mapowanie estetyk

Pierwszym elementem jest tzw. estetyka (*aesthetic*). Odwzorowuje ona dane w atrybuty wykresu takie jak: kolor, kształt, rozmiar. Mówi nam o roli jaką każda zmienna pełni na wykresie. Na przykład jedna zmienna będzie odwzorowana na osi X, druga na osi Y, a trzecia pojawi się w legendzie (w postaci koloru, a może kształtu lub rozmiaru).

Budowę wykresu zaczynamy od funkcji `ggplot()`. Pierwszym jej argumentem jest nazwa ramki danych. Kolejne odnoszą się do elementów estetycznych – przydziel role poszczególnym

[1] Możesz zobaczyć, jak wygląda przykładowy wykres utworzony w systemie tradycyjnym. Wystarczy, że wpiszesz w konsoli: `plot(rnorm(100), type = "b")`. Porównaj go z wykresami w tym rozdziale.

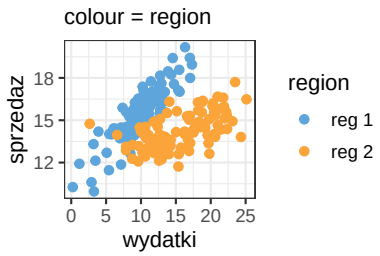
[2] Dokumentacja `ggplot2` jest również dostępna pod oficjalnym adresem: `http://ggplot2.tidyverse.org/reference/`

zmiennym, zgodnie z poniższą składnią. W miejsce kropek wstaw nazwy zmiennych ramki danych. Zwróć również uwagę na skrót *aes* pochodzący od angielskiego słowa *aesthetic*.

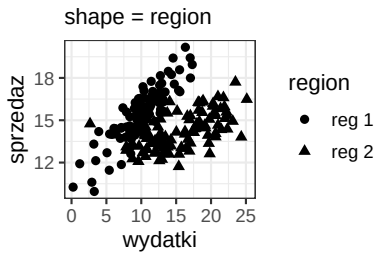
```
ggplot(ramkaDanych, aes(x = ..., y = ..., colour = ..., fill = ..., shape = ...))
```

Zapewne nie wszystkie elementy estetyki będziemy wykorzystywać jednocześnie. Rozważmy prosty przykład, w którym chcemy wizualizować relacje między wydatkami na reklamę a wielkością sprzedaży. W tej sytuacji przyjmujemy, że na osi X będą `wydatki`, a więc `x = wydatki`. Z kolei na osi Y znajdzie się zmienna `sprzdaz`. To wystarczy do stworzenia prostego wykresu. Załóżmy jednak, że firma działa w 3 województwach i ten fakt również powinniśmy uwzględnić na wykresie. Ta zmienna znajdzie się zapewne w legendzie, ale pojawia sie pytanie: który element estetyczny powinniśmy wybrać? To jest ściśle powiązane z drugą warstwą (elementami geometrycznymi), o której piszę poniżej. Bo jeśli zdecydujemy się na punkty, jako element geometryczny, wtedy możemy je różnicować kolorem (`colour`) albo kształtem (`shape`). Mamy również możliwość wyboru obu estetyk. Spójrz na poniższe warianty składni i odpowiadające im wykresy.

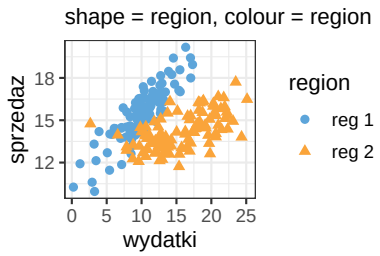
```
ggplot(wydSprz,
    aes(x = wydatki,
        y = sprzedaz,
   colour = region))
```




```
ggplot(wydSprz,
    aes(x = wydatki,
        y = sprzedaz,
    shape = region))
```




```
ggplot(wydSprz,
    aes(x = wydatki,
        y = sprzedaz,
    shape = region,
   colour = region))
```



Elementy geometryczne

Drugim elementem (warstwą) jest atrybut geometryczny. Mówi nam o tym, w jakie elementy geometryczne będą odwzorowywane zmienne. Możemy wybrać punkty, linie, słupki itp. Funkcje odpowiadające geometrii[3] zaczynają się od `geom_`. Drugi człon, który dodajemy, opowiada nazwie elementowi geometrycznemu. Punkty, które widzisz na powyższych wykresach, dodałem za pomocą funkcji `geom_point()`. I to dosłownie – użyłem znaku +. W taki sposób będziemy dodawać kolejne warstwy. Spójrz na poniższą składnię, której wynikiem jest utworzony wyżej wykres – ten po prawej stronie.

```
ggplot(wydSprz, aes(x = wydatki, y = sprzedaz, shape = region, colour = region)) +
    geom_point()
```

Zwróć uwagę na brak argumentów funkcji `geom_point()`. Zazwyczaj elementy geometryczne ich nie wymagają. Jeśli jednak chcesz zmienić np. rozmiar punktu, wtedy musisz zmienić wartość domyślną tego argumentu. Listę najczęściej wykorzystywanych elementów geometrycznych zamieszczam w tabeli 6.1. Omówię je w dalszej części opracowania. Argumentami opcjonalnymi nie musisz się przejmować – potraktuje je jako materiał nieobowiązkowy.

Jeśli jednak chcesz mieć kontrolę nad ostatecznym wyglądem wykresu, spróbuj zapamiętać następujący schemat. Argument `size` odpowiada wielkości punktu lub grubości linii – przyjmuje wartości rzeczywiste dodatnie (np. 0.75, 2). Kolor linii lub punktu zmienisz używając

[3] Liczba tych funkcji jest duża i wynosi: 53. Najważniejsze zamieszczam w tabeli 6.1. Z nich będziemy korzystać.

argumentu `colour`. Przykład: `geom_point(size = 4, colour = "#a0589d")`. Podobną funkcję pełni `fill`, ale odpowiada za wypełnienie czegoś – może to być słupek, pole pod krzywą albo punkt z pustym środkiem (np. okrąg, trójkąt). Linii nie wypełniamy tylko kolorujemy. Rodzaj linii definiuje argument `linetype`. Choć będę o tym pisał, to wiedz, że możesz podstawić wartości naturalne od 0 do 6. Rodzaj punktu `shape` ustawisz wpisując liczbę naturalną od 0 do 25. Pojawiająca się wszędzie `alpha` odpowiada nasyceniu koloru i przyjmuje wartości z przedziału $[0,1]$, gdzie największa wartość oznacza całkowite pokrycie. Przykład: `geom_point(size=4, shape=20, colour="blue", alpha=0.3)`.

Tabela 6.1. Funkcje elementów geometrycznych

Funkcja	Rysuje	Argumenty opcjonalne
`geom_point()`	Punkty	`size`, `shape`, `colour`, `fill`, `alpha`
`geom_line()`	Linie	`size`, `linetype`, `colour`, `alpha`
`geom_vline(xintercept = ...)`	Linie pionowe	`size`, `linetype`, `colour`, `alpha`
`geom_hline(yintercept = ...)`	Linie poziome	`size`, `linetype`, `colour`, `alpha`
`geom_smooth()`	Krzywą wygładzoną	`method`, `se`, `size`, `linetype`, `colour`, `alpha`
`geom_segment(aes(xend=..., yend=...))`	Linie od ... do ...	`size`, `linetype`, `colour`, `alpha`
`geom_bar()`, `geom_col()`	Słupki	`position`, `linetype`, `colour`, `fill`, `alpha`
`geom_histogram()`	Histogram	`bins`, `linetype`, `colour`, `fill`, `alpha`
`geom_density()`	Gęstość	`size`, `linetype`, `colour`, `fill`, `alpha`
`geom_boxplot()`	Pudełko-wąsy	`size`, `linetype`, `colour`, `fill`, `alpha`, `coef`, `outlier.size`, `outlier.shape`, `outlier.colour`, `outlier.fill`, `outlier.alpha`
`stat_function(fun = ...)`	Funkcję	`args=list(...)`, `size`, `linetype`, `colour`, `alpha`
`stat_ecdf()`	Dystrybuantę	`size`, `linetype`, `colour`, `alpha`
`stat_qq()`	Punkty kwantyl-kwantyl	`size`, `shape`, `colour`, `fill`, `alpha`
`stat_qq_line()`	Linię kwantyl-kwantyl	`size`, `linetype`, `colour`, `alpha`

Obok elementów geometrycznych, w tabeli 6.1, widzisz też elementy statystyczne, które zaczynają się od `stat_`. Ich zadaniem jest przekształcanie zmiennych z wykorzystaniem elementów statystyki i odwzorowanie wyniku w elementy geometryczne.

Skale

Kolejnym elementem są skale (zaczynają się od `scale_`). Pozwalają one kontrolować sposób, w jaki dane są odwzorowywane w elementy estetyczne. W tym miejscu możemy decydować o kolorystyce, kształcie obiektów, rozmiarze, zakresie wartości danych itd. Skoro argumenty funkcji z tabeli 6.1, zapytasz, pozawalają zmieniać kolorystykę, to po co mi kontrola nad skalami? Musisz jednak wiedzieć, że te argumenty ustawiają jeden i ten sam kolor dla danego obiektu geometrycznego. Przypomnij sobie wykres o wydatkach i sprzedaży, na którym mieliśmy dwa kolory punktów zależne od regionu. Zmieniłem je z wartości domyślnych, wykorzystując skale – inaczej się nie da.

Pamiętaj, że skale są elementem opcjonalnym. Zmień je wtedy, gdy domyślne ustawienia ci nie odpowiadają. Lista funkcji jest długa (obecnie 121), więc zaprezentuję tylko niektóre. Dodatkowo ograniczę się do pewnego schematu. Ponieważ funkcje składają się z 3 członów oddzielonych znakiem podkreślenia – a pierwszy już znasz (`scale_`) – więc w tabeli 6.2 pojawią się nazwy dwóch następnych. W ramach każdego wiersza utworzysz tyle funkcji, ile jest

możliwych powiązań między drugim a trzecim członem. Ważne: wybór elementu dla drugiego członu związany jest z tym, co chcesz zmienić. Jeśli będzie to rozmiar linii bądź punktu wybierzesz `size`, wypełnienie słupka to `fill`, kolor natomiast odpowiada elementowi `colour` itd.

Rozważmy jeszcze przykład tworzenia takich funkcji. W ostatnim wierszu tabeli 6.2 mamy 2 możliwości dla drugiego członu – (`x`, `y`), i tyle samo dla trzeciego – (`continuous`, `discrete`). Dlatego utworzymy 4 funkcje: (1) `scale_x_continuous` (2) `scale_x_discrete` (3) `scale_y_continuous` (4) `scale_y_discrete`.

Tabela 6.2. Funkcje dla skal

Drugi człon	Trzeci człon	Szczegóły
`colour, fill`	`brewer`	Kolorystyka z pakietu `RColorBrewer`. Wymagany argument `palette` i znajomość jego wartości, np. `scale_colour_brewer(palette = "Set1")`
`colour, fill`	`manual`	Ręcznie ustawiany argument koloru `values`. Wymagana znajomość kodu koloru w zapisie szesnastkowym (HEX), np. zmiana dwóch kolorów: `scale_fill_manual(values = c("#69b4ff", "#377eb8"))`
`size, shape, linetype`	`manual`	Ręcznie ustawiany argument `values` dla rozmiaru, kształtu lub typu linii, np. `scale_linetype_manual(values = 1)`
`x, y`	`continuous, discrete`	Ręcznie ustawiany przynajmniej jeden z argumentów: `breaks`, `labels`, `limits`, np. `scale_y_continuous(breaks = seq(1, 13, 3), limits = c(4, 10))`. Dane poza zakresem są ustawiane na `NA`.

Do skal włączymy jeszcze dwie ważne funkcje.

```
labs(x = ..., y = ..., fill = ..., colour = ..., shape = ...)
ggtitle(label = ..., "podtytul")
```

Pierwsza zmienia domyśle opisy dla osi X i Y oraz legendy. W wypadku opisu legendy wybieramy te elementy estetyczne, które użyliśmy w funkcji `ggplot()`. Druga z funkcji dodaje tytuł wykresu (`label`) oraz opcjonalnie podtytuł. Przykładowo dla wykresu o wydatkach i sprzedaży, tego po lewej stronie, wpisałem:

```
labs(x = "wydatki", y = "sprzedaz", colour = "region") +
ggtitle(label = NULL, "region") # usuń duży tytułu (dlatego NULL), dodaj podtytuł
```

Do poniższego fragmentu wykresu dodaj warstwy, które zmienią kolor i kształt punktów. Zmienna `region` ma 2 kategorie. Dodaj również tytuł wykresu.

```
ggplot(wydSprz, aes(x = wydatki, y = sprzedaz, shape = region, colour = region)) +
  geom_point(size = 2)
```

Współrzędne

Opcjonalną warstwą są też współrzędne, których funkcje zaczynają się od `coord_`. Szczególnie przydatne są trzy. Pozwalają one definiować zakresy dla osi X i Y, ustawiać proporcję między osiami oraz zamieniać osie (transponować). Widoczne wartości argumentów w tabeli 6.3 są domyślne.

Tabela 6.3. Funkcje dla współrzędnych

Funkcja	Opis
`coord_cartesian(xlim = NULL, ylim = NULL, expand = TRUE)`	Zmienia zakres współrzędnych X i Y. Dane poza zakresem są niewidoczne, ale nie jak w wypadku skal ustawione na NA. Argument `expand = TRUE` dodaje niewielkie wartości do zakresu.
`coord_fixed(ratio = 1, xlim = NULL, ylim = NULL, expand = TRUE)`	Zmienia proporcje `ratio` między osiami. Dodatkowo pozwala na zmianę zakresu współrzędnych.
`coord_flip(xlim = NULL, ylim = NULL, expand = TRUE)`	Zamienia współrzędne miejscami (transponuje). Dodatkowo pozwala na zmianę zakresu współrzędnych.

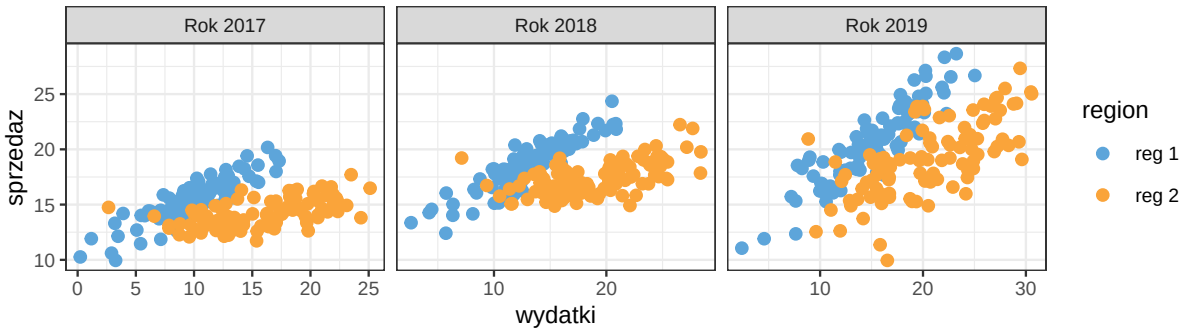
Panele

Za pomocą paneli (*facet*) możemy przedstawić wiele wykresów cząstkowych. Idea polega na tym, że dane dzielone są ze względu na poziomy zmiennej kategorialnej i dla każdego poziomu tworzony jest osobny wykres. Możemy z tego skorzystać, zamiast przenosić zmienną do tzw. legendy. Częściej jednak posługujemy się tym rozwiązaniem, gdy w legendzie jest już zmienna, a my chcemy dodać kolejną. Do wyboru mamy dwie funkcje:

```
facet_wrap(~ zm_1, nrow = NULL, ncol = NULL, scales = "fixed")
facet_grid(zm_1 ~ zm_2, scales = "fixed", space = "fixed")
```

Wymaganym argumentem jest nazwa zmiennej kategorialnej lub dyskretnej `zm_`. Kolejne, opcjonalne argumenty pozwalają kontrolować liczbę wykresów w wierszu (`nrow`) albo w kolumnie (`ncol`). Domyślnie skale osi są takie same we wszystkich panelach (`fixed`). Możemy to zmienić, przyjmując: `free_x`, `free_y` albo `free`. Wtedy skale będą różne dla wybranej osi albo dla wszystkich osi. Poniższy wykres otrzymałem dodając warstwę, którą widzisz poniżej. Choć różne skale dla wydatków raczej nie mają tutaj uzasadnienia, to zrobiłem to celowo – zwróć uwagę na osie X i Y w 3 panelach.

```
facet_wrap(~ rok, scales = "free_x")
```



Z kolei funkcja `facet_grid()` przedstawia wykresy w postaci macierzowej. W wierszu znajdą się poziomy zmiennej `zm_1`, w kolumnie natomiast poziomy zmiennej `zm_2`. Z tej funkcji nie będziemy korzystać.

6.2. Wybrane wykresy

W tej części praktycznej pokażę ci, w jaki sposób budować najpopularniejsze wykresy. Do takich zaliczam wykresy: rozrzutu (punktowe), liniowe, słupkowe. Do tej grupy włączę też wykresy pozwalające przedstawić rozkład zmiennej ilościowej za pomocą: funkcji gęstości, dystrybuanty, rozkładu kwantyl-kwantyl czy wykresu pudełko-wąsy. Wszystkie funkcje, których będziemy potrzebować, zamieściłem w tabeli 6.1 na stronie 57. Staram się nie używać

argumentów opcjonalnych i ograniczam się do dwóch pierwszych warstw (estetyki i geometrii). Nieraz zmuszony jestem użyć argumentów opcjonalnych czy też warstw. Dobrym przykładem argumentu opcjonalnego, który zmieniam, jest rozmiar punktu (`size`). Zdarza się, że zmienna kategorialna ma wiele kategorii, których etykiety nie mieszczą się na osi X. Wtedy zamieniam współrzędne miejscami dodając warstwę `coord_flip()`. Bez tych zabiegów wykresy byłyby nieczytelne. Ostatnim obowiązkowym elementem są podpisy osi – rzadko można z nich zrezygnować.

Muszę jeszcze wspomnieć o związku między kodem a wykresami, które widzisz. Zawsze do finalnego wykresu dodaję warstwę odnoszącą się do tematu `theme_bw()` – aby usunąć szare tło – i zmieniam domyślną kolorystykę. Tego nie zobaczysz w kodzie, bo celowo tego nie pokazuję. Dlatego jeśli uruchomisz zamieszczone tutaj skrypty, nie otrzymasz identycznego wykresu. Pamiętaj również o wczytaniu odpowiednich pakietów – sugeruję wczytać zbiorczy pakiet `tidyverse`. W tym rozdziale będziemy pracowali z ramką danych `bank`, którą omówiłem na stronie 48.

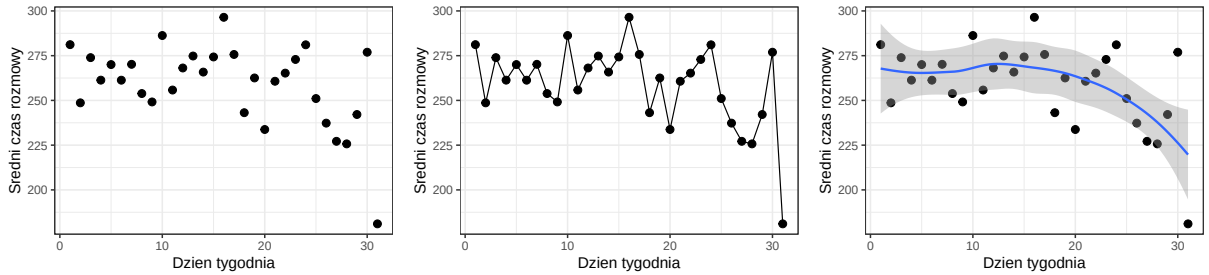
6.2.1. Punkty i linie.

Już wiesz, że wykresy tworzymy nakładając kolejne warstwy. Tym samym możemy najpierw narysować punkty, później je połączyć linią, a ostatecznie nanieść krzywą wygładzoną. W poniższym przykładzie wykorzystamy odpowiednie funkcje z tab. 6.1. Użyjemy ramki danych `dayCzas`, którą utworzymy na podstawie ramki `bank`.

```
> ## Ramka danych: bank; tworzymy ramkę dayCzas
> dayCzas <- bank %>%
+    group_by(day) %>%
+    summarise(czas = mean(duration))
> head(dayCzas)
# A tibble: 6 x 2
    day  czas
  <int> <dbl>
1     1  281.
2     2  249.
3     3  274.
4     4  261.
5     5  270.
6     6  261.
```

W powyższej ramce danych mamy średni czas rozmowy w każdym dniu miesiąca. Chcemy to przedstawić na wykresie. W pierwszym kroku określamy tzw. estetykę – która zmienna będzie na jakiej osi. Niech na osi X będzie zmienna `day`, a na osi Y znajdzie się zmienna `czas`. Drugi krok to wybór elementu geometrycznego. Stwórzmy 3 różne wykresy.

```
> p1 <- ggplot(dayCzas, aes(x = day, y = czas)) +
+ geom_point(size = 3) +
+ labs(x = "Dzien tygodnia", y = "Sredni czas rozmowy")
> p1                  # wykres po lewej stronie
> p1 + geom_line()    # wykres środkowy
> p1 + geom_smooth()  # wykres po prawej stronie
```



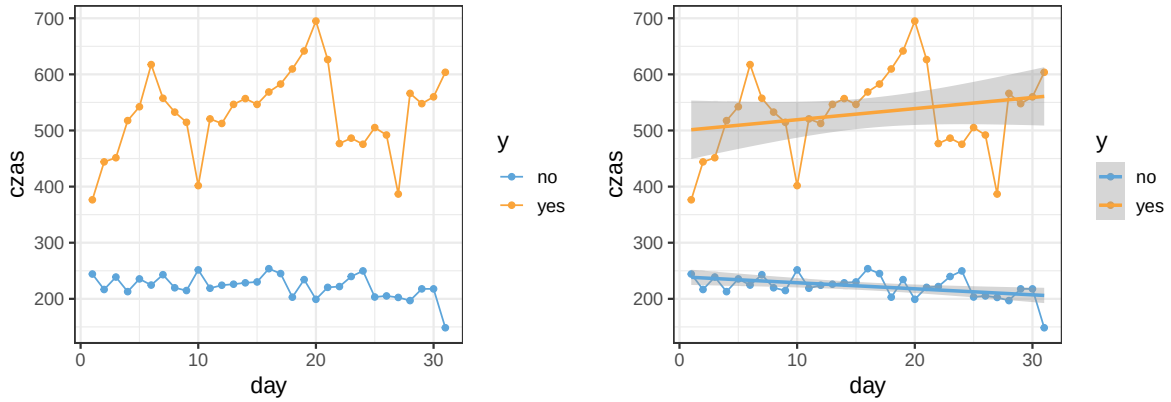
Jak widzisz, pierwszy wykres składa się tylko z punktów. Aby go wyświetlić wpisujemy `p1`. Do tego wykresu dodajemy kolejną warstwę geometryczną – linię. To zapewne ułatwi nam interpretację. Do punktów możemy dodać inny element geometryczny – krzywą wygładzoną. W ten sposób otrzymujemy ostatni wykres. Do punktów dodasz również dowolną warstw. Spróbuj dodać do siebie wszystkie 3 elementy geometryczne (punkty). Proponuję również zmienić krzywą wygładzoną na linię prostą pisząc: `geom_smooth(method = "lm")`.

Zbudujemy wykres podobny do poprzednich wykresów. Różnica będzie polegać na dodaniu zmiennej kategorialnej `y`, odpowiadającej zgodzie na lokatę terminową. W konsekwencji pojawią się dwie linie i dwie grupy punktów odpowiadające poziomom dodanej zmiennej: `yes`, `no`. Zanim jednak przystąpimy do budowy wykresu, musimy przygotować dane.

```
> dayCzas2 <- bank %>%
+    group_by(day, y) %>%
+    summarise(czas = mean(duration))
> head(dayCzas2, 4)
# A tibble: 4 x 3
# Groups:   day [2]
    day y      czas
  <int> <chr> <dbl>
1     1 no     244.
2     1 yes    376.
3     2 no     217.
4     2 yes    444.
```

Dodatkowa zmienna wymaga od nas dodania kolejnego elementu estetycznego. Jeśli chcemy dokonać rozróżnienia między kategoriami zmiennej `y` za pomocą koloru, wtedy napiszemy: `colour=y`. Nie jest to jedyna możliwość. Możemy np. dobrać też różne kształty dla punktów, co osiągniemy, pisząc `shape=y`. Niech nie zmyli nas nazwa tej zmiennej. Nie ma ona nic wspólnego z elementem estetycznym `y=...`. To jest zwykły zbieg okoliczności. Decydujemy się na pierwsze rozwiązanie i wybieramy kolor jako podstawę rozróżnienia kategorii.

```
> p2 <- ggplot(dayCzas2, aes(x = day, y = czas, colour = y)) +
+       geom_point() + geom_line()
> p2                              # wykres po lewej
> p2 + geom_smooth(method = "lm") # wykres po prawej
```



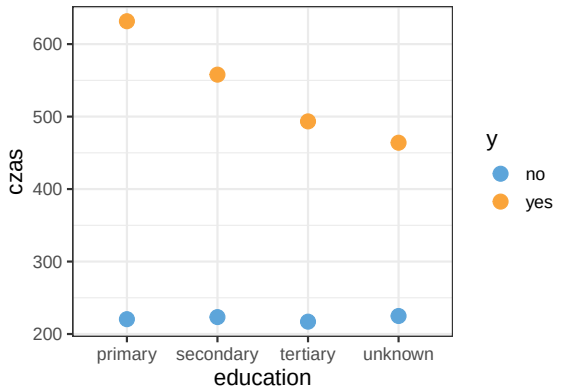
Dodając kolejną warstwę w postaci linii, otrzymaliśmy zamierzony efekt, tzn. punkty zostały połączone linią poziomą. Ale zastanów się: dlaczego R nie połączył tych punktów linią pionową? Przecież technicznie też jest to możliwe. O tym zadecydował typ liczbowy zmiennej `day` i zmiennej `czas`. R w tym wypadku nie miał żadnych wątpliwości. Wobec tego co się stanie, jeśli zamiast zmiennej ilościowej na osi X będziemy mieli zmienną kategorialną? Przeanalizuj poniższą ramkę danych i odpowiadający jej wykres punktów. Jak myślisz, jaki będzie efekt dodania linii?

Zapamiętaj 6.1

Aby ułatwić analizę wykresów, zrezygnowałem z opisów osi. Widzisz tam oryginalne nazwy z ramki danych. Pamiętaj, że opisy są niezbędne. Spróbuj do wykresu `p2` dodać:

```
labs(x = "Poziom edukacji", y = "Średni czas rozmowy", colour = "Lokata")
```

```
> eduCzas <- bank %>%
+    group_by(education, y) %>%
+    summarise(czas = mean(duration))
# A tibble: 8 x 3
# Groups:   education [4]
  education y      czas
  <chr>     <chr> <dbl>
1 primary   no     220.
2 primary   yes    632.
3 secondary no     223.
4 secondary yes    558.
5 tertiary  no     217.
6 tertiary  yes    493.
7 unknown   no     225.
8 unknown   yes    464.
```

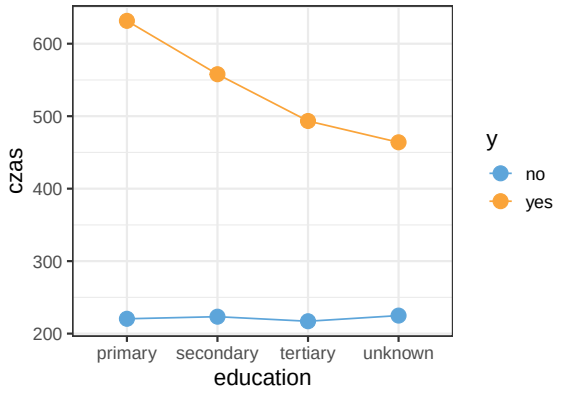


W zależności od tego co chcemy zaakcentować na wykresie, mamy do wyboru dwie możliwości. Pierwsza – łączymy punkty linią pionową. W ten sposób zwracamy większą uwagę na różnice między decyzjami odnośnie depozytów. Interpretując taki wykres możemy powiedzieć, że wraz ze wzrostem poziomu edukacji, skraca się dystans między czasem rozmów osób zgadzających się i niezgadzających się na lokatę terminową. Druga możliwość – łączymy punkty linią poziomą, gdy bardziej chcemy zaakcentować zmianę czasu rozmowy w zależności od poziomu wykształcenia. W tej sytuacji powiemy, że czas rozmów osób godzących się na lokatę terminową wyraźnie spada wraz ze wzrostem wykształcenia. Tego nie można powiedzieć o tych, którzy nie zdecydowali się na taką lokatę. Niezależnie od wykształcenia, ten czasu utrzymuje się na podobnym poziomie.

Powyższe uwagi prowadzą nas do oczywistego wniosku: musimy w elemencie estetycznym

group podać nazwę tej zmiennej, w obrębie której punkty mają być połączone. Decydując się na drugą możliwość tą zmienną jest y.

```
> ggplot(eduCzas, aes(x = education,
+                     y = czas,
+                     colour = y,
+                     group = y)) +
+ geom_line() +
+ geom_point(size = 4)
```

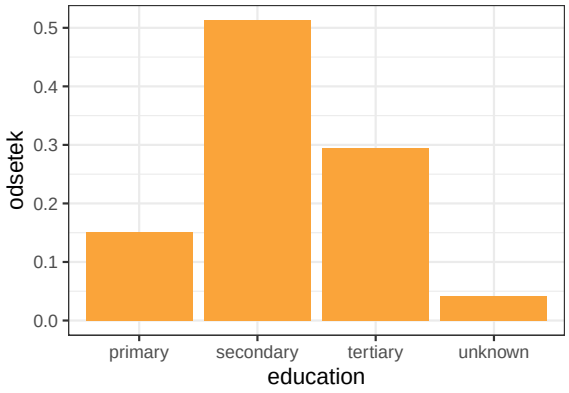


6.2.2. Wykresy słupkowe. Wykres słupkowy rysujemy nanosząc warstwę geometryczną za pomocą `geom_bar()` albo `geom_col()`. Różnice między tymi funkcjami wyjaśniam w ramce: warto wiedzieć. Skupimy się tylko na drugiej funkcji i następującej strategii tworzenia wykresów słupkowych. Jeśli chcemy wizualizować statystyki zmiennych kategorialnych tj. liczebność, odsetki lub procenty, wtedy musimy stworzyć odpowiednią tabelę. Tworzyliśmy je w rozdz. 5.4.2 i 5.4.3. Chodzi o to, aby każdej wartości odpowiadał jeden słupek, bo tego wymaga funkcja `geom_col()`. Dla tak przygotowanej ramki budujemy wykres.

Wykorzystamy tę strategię i zbudujemy wykres, który będzie przedstawiał rozkład edukacji. Przyjmijmy, że interesującą nas statystyką jest odsetek. Poniżej tworzymy tabelę tabEdu, a następnie wykres. Zwróć uwagę na elementy estetyczne. Zamiana osi nie spowoduje, że otrzymasz to, o czym myślisz. Jeśli osie chcesz zamienić miejscami, to użyj warstwy odpowiedzialnej za współrzędne: `coord_flip()`.

```
> tabEdu <- bank %>% count(education) %>%
+   mutate(odsetek  = n/sum(n))
> tabEdu # ramka do zwizualizowania
  education     n odsetek
1   primary  6851  0.1515
2 secondary 23202  0.5132
3  tertiary 13301  0.2942
4   unknown  1857  0.0411

> p4 <- ggplot(tabEdu, aes(x = education, y = c
+       geom_col()
```



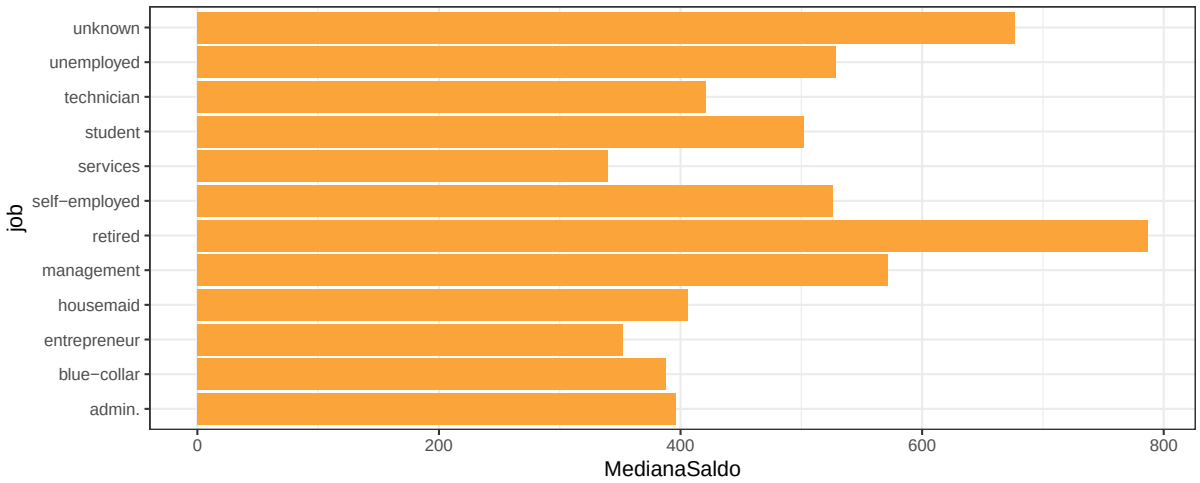
> **Warto wiedzieć**
>
> **Budowa wykresów słupkowych: `geom_bar()` a `geom_col()`**
>
> Niezależnie od tego, którą z poniższych funkcji zastosujesz, otrzymasz identyczny rezultat.
>
> ```
> geom_col()
> geom_bar(stat = "identity")
> ```
>
> Bez opcjonalnego argumentu druga funkcja najpierw wywołuje funkcję zliczającą wystąpienia wszystkich poziomów zmiennej kategorialnej, a dopiero później tworzy słupki odpowiadające tym liczebnościom. Tak wywołana funkcja wymaga też

> oryginalnej, nieprzetworzonej ramki bank. Zazwyczaj jesteśmy zainteresowani wartościami względnymi (odsetkami, procentami), więc musimy je policzyć. To z kolei narzuca na nas konieczność wpisania argumentu opcjonalnego. Czy nie szybciej będzie, jak użyjemy `geom_col()`?

Wysokość słupka może odpowiadać wybranej statystyce dla zmiennej ciągłej. Jako miarę agregacji możemy wziąć średnią, medianę, kwantyl itd. Zobaczmy jak kształtuje się mediana salda rachunku (`balance`) w zależności od kategorii zatrudnienia (`job`). Aby kategorie na osi X nie zachodziły na siebie, zamienimy osie miejscami, dodając `coord_flip()`.

```
> balJob <- bank %>%
+    group_by(job) %>%
+    summarise(MedianaSaldo = median(balance))
> head(balJob, 3)
# A tibble: 3 x 2
  job          MedianaSaldo
  <chr>               <dbl>
1 admin.                396
2 blue-collar           388
3 entrepreneur          352
```

```
> ggplot(balJob, aes(x=job, y=MedianaSaldo)) + geom_col() + coord_flip()
```



Jeśli zmienna kategorialna ma wiele poziomów, rozważ wykres punkty zamiast wykresu słupkowego. Taki wykres już umiesz zbudować. W poniższym przykładzie, który bazuje na ramce `balJob`, uwzględniam dwie nowe funkcje. Pierwsza `reorder()` sortuje kategorie zmiennej dyskretnej `jakaZm` w kolejności `kolejnosc`. Taki zabieg na pewno ułatwi nam interpretację wykresu. Oczywiście ma to sens wtedy, gdy zmienna zmierzona jest na skali nominalnej. Zauważ, że do poprzedniego wykresu słupkowego również mogliśmy ją użyć. Druga funkcja pojawiła się w tabeli 6.1, ale jej nie omówiłem. Chodzi mi o `geom_segment()`, która pozwala dodać linie pionowe lub poziome w zakresie od do. W naszym przykładzie te linie będą pełniły rolę linii wiodących – ułatwiają powiązanie punktu z kategorią. Musimy tylko podać gdzie się kończą x i y, bo swój początek mają już w narysowanym, czarnym punkcie.

```
> p5 <- ggplot(balJob, aes(x = MedianaSaldo, y = reorder(job, MedianaSaldo)))
> p5 <- p5 + geom_point(size=3)
> p5  + labs(x = "Mediana salda", y = "Kategoria zatrudnienia") # wykres po lewej stronie
> p5 + geom_segment(aes(yend = job, xend = 0))                  # wykres po prawej stronie
```

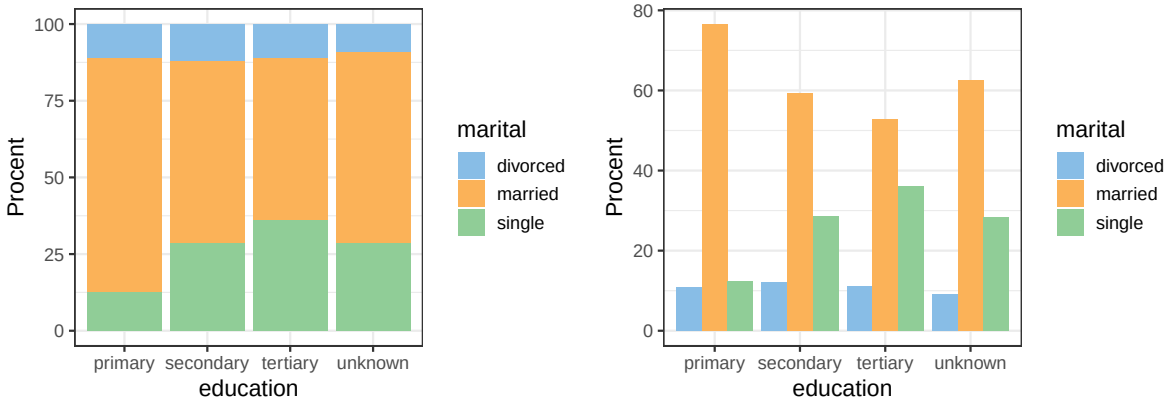
Powinienem jeszcze omówić sytuację, w której dodajemy kolejną zmienną kategorialną i umieszczamy ją w tzw. legendzie. Modyfikacje poprzednich przykładów będą niewielkie. Pierwsza z nich jest dość oczywista, bo wymaga od nas dołączenia kolejnego elementu estetycznego. Jeśli poziomy zmiennej będziemy różnicować wypełnieniem słupka, wtedy w `aes` zapiszemy `fill = nazwa_zmiennej`. Druga modyfikacja ma charakter preferencji wizualnych. Jeżeli chcemy otrzymać wykres słupkowy zestawiony (*stack*), w którym słupki umieszczane są jeden na drugim, wtedy nic nie musimy zmieniać. Jest to domyślne zachowanie funkcji. Jeśli jednak chcemy otrzymać wykres słupkowy zgrupowany, gdzie słupki pojawią się obok siebie, wtedy takie zachowanie wymusimy pisząc: `geom_col(position = "dodge")`.

W przykładzie ilustrującym pokażę, jak rozkładają się wartości procentowe dla zmiennej stan cywilny (`marital`) w zależności do wykształcenia (`education`). Konstruując tabelę, stanowiącą dane wejściowe dla wykresu, przyjmuję, że to kategorie zmiennej stan cywilny sumują się do 100%.

```
> eduMarit <- bank %>%
+    group_by(education, marital) %>%
+    summarise(Liczebnosc = n()) %>%
+    mutate(Procent = 100*Liczebnosc/sum(Liczebnosc))
> head(eduMarit, 3)
# A tibble: 3 x 4
# Groups:   education [1]
  education marital  Liczebnosc Procent
  <chr>     <chr>         <int>   <dbl>
1 primary   divorced        752    11.0
2 primary   married        5246    76.6
3 primary   single          853    12.5
```

Teraz możemy zbudować wykresy i to w dwóch wariantach.

```
> p6 <- ggplot(eduMarit, aes(x = education, y = Procent, fill = marital))
> p6 + geom_col()             # wykres po lewej stronie; domyślnie jest position = "stack"
> p6 + geom_col(position = "dodge") # wykres po prawej stronie
```



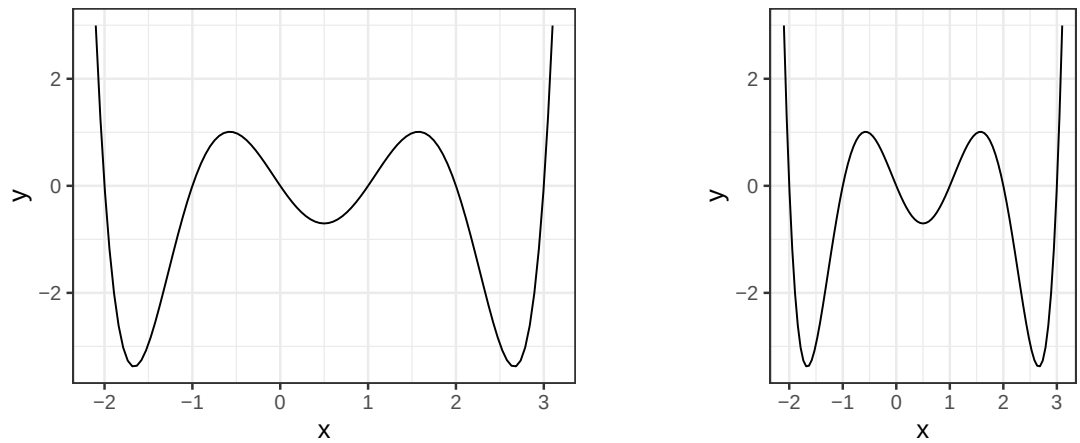
6.2.3. Wykresy rozkładu zmiennej. Rozkład zmiennej możemy przedstawić w postaci histogramu, funkcji gęstości estymatora jądrowego, dystrybuanty empirycznej, wykresu pudełko-wąsy i kwantyl-kwantyl (zob. tab. 6.1). Te wykresy budujemy dla zmiennych, których mamy realizację – dane. W R mamy też możliwości narysowania wykresu dowolnej funkcji zadanej równaniem. Od tego zaczniemy.

Wykresy funkcji

Czasami chcemy narysować rozkład teoretyczny i porównać go z empirycznym. W tym wypadku użyjemy funkcji `stat_function()`. Za jej pomocą narysujemy wykres dowolnej krzywej. Możemy wybrać funkcję już zaimplementowaną w R (np.: `sin()`, `log()`, `dnorm()` itp.) lub też samemu ją stworzyć. Zacznijmy od tej drugiej możliwości.

Naszym zadaniem jest narysowanie wykresu wielomianu 6 stopnia, którego postać zapiszemy w obiekcie o nazwie `Wielom6`. Pakiet `ggplot2` wymaga, by dane wejściowe były ramką danych. W naszym przykładzie wystarczy podać dwie wartości x, które są tożsame z krańcami przedziału – u nas będzie to przedział $(-2.1, 3.1)$. W poniższym przykładzie zwróć uwagę na skalowanie osi. Jeśli chcemy mieć gwarancję, że proporcja odległości na osi X do odległości na osi Y będzie 1:1, wtedy musimy użyć funkcji `coord_fixed(ratio = 1)`.

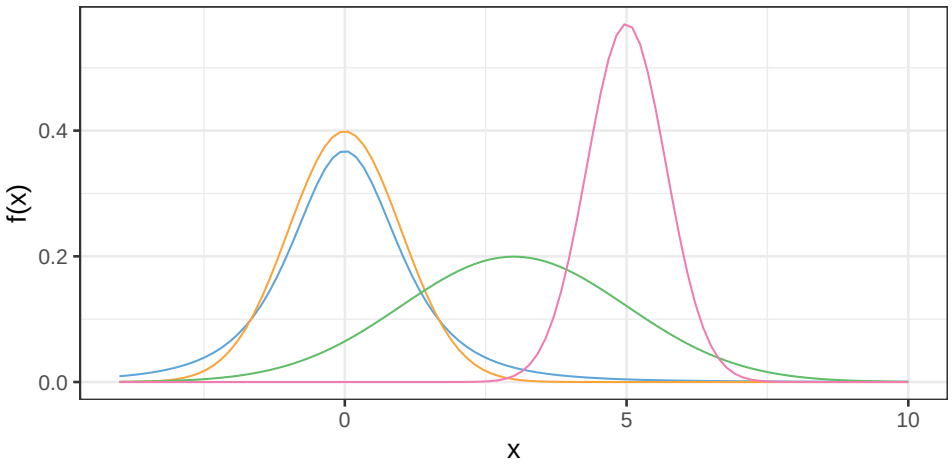
```
> ## Rysowanie wielomianu, o zadanie postaci funkcyjnej
> Wielom6 <- function(x) 0.2*x^6 - 0.6*x^5 - x^4 + 3*x^3 + 0.8*x^2 - 2.4*x
> wyk0 <- ggplot(data.frame(x = c(-2.1, 3.1)), aes(x = x))
> wyk <- wyk0 + stat_function(fun = Wielom6)
> wyk # odcinek [0,1] na osi X jest dłuższy od identycznego na osi Y
> wyk + coord_fixed(ratio = 1) #gwarancja, że długości będą identyczne
```



Przejdźmy teraz do narysowania kilku funkcji gęstości na tym samym wykresie. Postępowanie zasadniczo nie różni się od zaprezentowanego powyżej. Jednak zmuszeni jesteśmy do uwzględnienia argumentu opcjonalnego funkcji `stat_function()`, jeśli chcemy zmienić domyślne parametry rysowanych gęstości.

Narysujemy funkcję gęstość rozkładu t-Studenta. Wykorzystamy funkcję gęstości `dt()`, przyjmując liczbę 3 stopnie swobody (zob. str. 40). Następnie narysujemy trzy funkcje gęstości rozkładu normalnego za pomocą `dnorm()` (zob. str. 39). Za średnie przyjmiemy wartości: $0, 3, 5$, a za odchylenia standardowe: $1, 2, 0.7$. Taką możliwość daje nam argument opcjonalny `arg=list(..)`, w którym zamiast kropek podajemy nazwy parametrów i ich wartości. To znakomity moment, abyśmy użyli argumentu opcjonalnego `colour`, pozwalającego zmienić kolor linii.

```
> ## Rysowanie różnych funkcji gęstości
> dfX <- data.frame(osX = c(-4, 10)) #przedział zmienności x
> mdf <- ggplot(dfX, aes(x = osX)) + xlab("x") + ylab("f(x)")
> mdf + stat_function(fun = dt, colour = "#5DA5DA", args = list(df = 3)) +
+       stat_function(fun = dnorm, colour = "#FAA43A") + # normalny N(0,1)
+     stat_function(fun = dnorm, colour = "#60BD68", args = list(mean = 3, sd = 2)) +
+     stat_function(fun = dnorm, colour = "#F17CB0", args = list(mean = 5, sd = 0.7))
```



Histogram

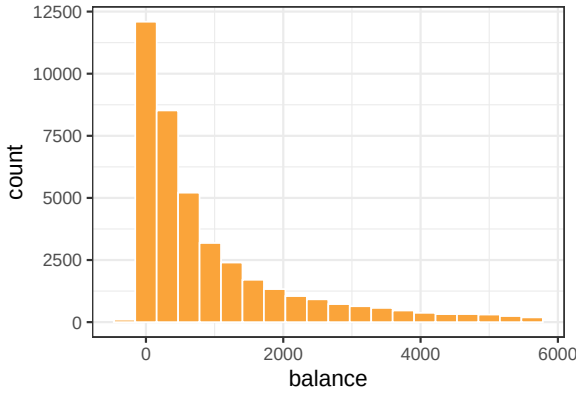
Prezentację rozkładów empirycznych rozpoczniemy od histogramu i funkcji `geom_histogram()`, która pojawiła się w tabeli 6.1. Domyślna wartość argumentu `bins` jest równa 30. Jeżeli chcesz tę liczbę przedziałów zmienić, wpisz po prostu swoją wartość – poeksperymentuj.

Stwórzmy histogram dla zmiennej saldo rachunku (`balance`) z ramki danych `bank`. Ze względów wizualnych ograniczmy się do obserwacji leżących między 5 a 95 percentylem. Nowa ramka będzie miała nazwę: `bankCut`.

```
> ## Ramka danych: bank; przygotowanie danych: balance między 5 a 95 percentylem
> (quantBalance <- quantile(bank$balance, probs = c(0.05, 0.95)))
  5%  95%
-172 5768

> bankCut <- bank %>%
+    filter(balance > quantBalance[1], balance < quantBalance[2])
```

```
> p7 <- ggplot(bankCut, aes(x = balance))
> p7  + geom_histogram(bins = 20)
```



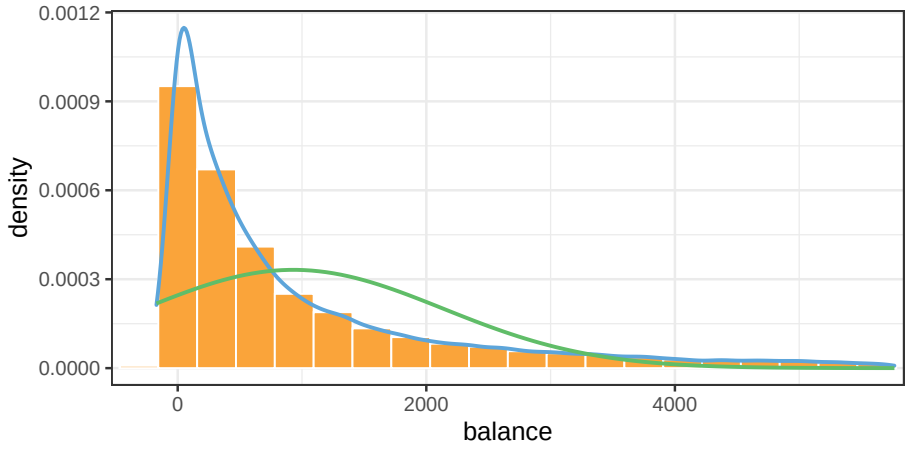
Strefa Eksperta 6.1

Dodawanie funkcji gęstości do histogramu

Dla histogramu przyjmowane są liczebności na osi rzędnej (Y). Jeśli zechcemy na histogram nanieść jakąkolwiek funkcję gęstości, wtedy pojawi sie problem jednostek. W konsekwencji funkcji gęstości nie będzie po prostu widać (podobnie jak: liczebności vs. częstości). Dlatego musimy zmienić skalę dla histogramu z domyślnej (liczebności) na skalę odpowiadającą gęstości. Odbywa się to poprzez dodatkowy argument: `aes(y = ..density..)`.

Poniżej ten sam przykład z naniesioną gęstością teoretyczną oraz gęstością estymatora jądrowego (o tym w kolejnym rozdziale). Przyjmijmy, że będzie to rozkład normalny ze średnią 930 i odchyleniem standardowym 1203 – wartości obliczone z próby. Jak widzisz, rozkład empiryczny w ogóle nie przypomina rozkładu normalnego (kolor zielony).

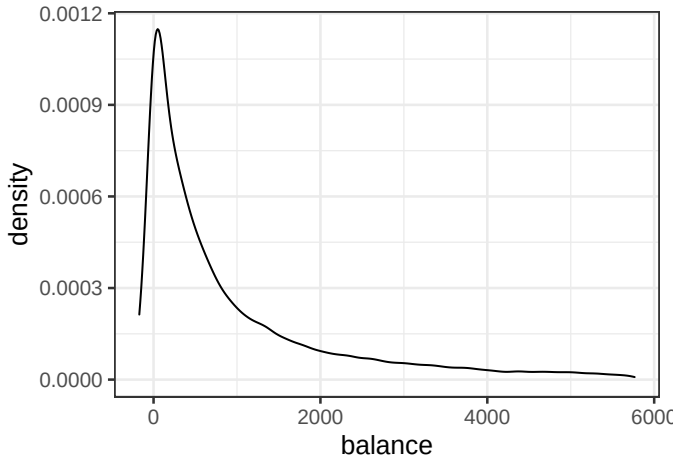
```
> p8 <- ggplot(bankCut, aes(x = balance))
> p8  + geom_histogram(aes(y = ..density..), fill = "#FAA43A", colour = "white", bins = 20) +
+       geom_density(colour = "#5DA5DA", size = 1) +
+       stat_function(fun = dnorm,
+                     args = list(mean = 930, sd = 1203),
+                     colour = "#60BD68", size = 1) +
+       scale_x_continuous(expand = c(0.01, 0.01))
```



Estymator gęstości jądrowej

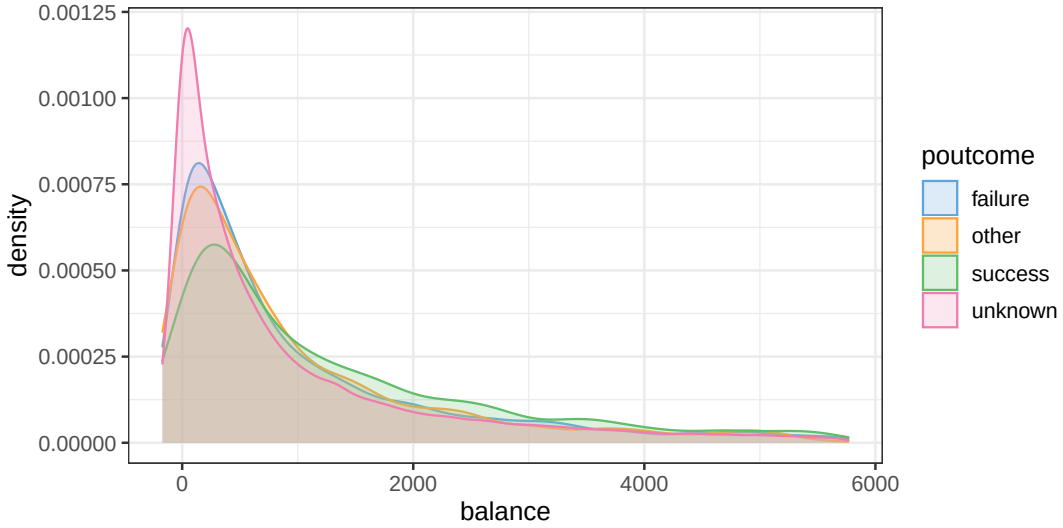
Funkcja gęstości estymatora jądrowego `geom_density()` pozwala dokładniej przedstawić rozkład empiryczny zmiennej. Wykorzystajmy uprzednio przygotowaną ramkę danych `bankCut` i zbudujmy taki wykres dla zmiennej saldo rachunku (`balance`).

```
> p8 <- ggplot(bankCut, aes(x = balan
> p8  + geom_density()
```



W dość prosty sposób możemy rozszerzyć powyższą składnię i nanieść kilka gęstości. Załóżmy, że chcemy zobaczyć, jak wyglądają gęstości ze względu na wynik ostatniej kampanii marketingowej (`poutcome`). Jeśli chcemy wypełnić gęstości kolorem, wtedy potrzebujemy dodatkowego elementu estetycznego `fill`, który będzie wskazywał na zmienną `poutcome`. W tej sytuacji warto zmienić intensywność wypełnienia kolorem, sterując argumentem opcjonalnym $\alpha \in (0,1)$, np. `geom_density(alpha = 0.4)`. Mamy też możliwość zmiany koloru linii samej krzywej, jeżeli użyjemy kolejnego elementu estetycznego `colour`. Przyjmijmy, że kolory wypełnienia i linii będą identyczne. Poeksperymentuj z poniższym kodem, usuwając jeden z elementów estetycznych.

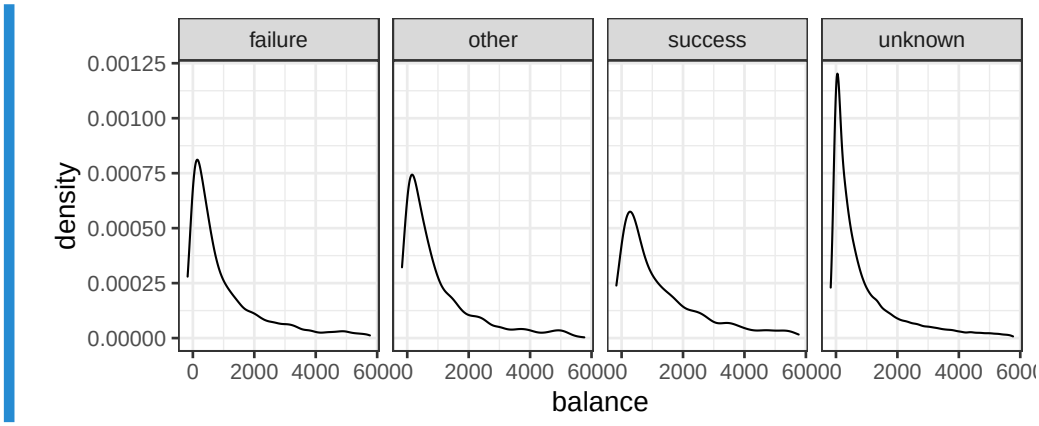
```
> p9 <- ggplot(bankCut, aes(x = balance, fill = poutcome, colour = poutcome))
> p9  + geom_density(alpha = 0.3)
```



WARTO WIEDZIEĆ

Zobacz, jak łatwo utworzyć alternatywną grafikę z wykorzystaniem paneli.

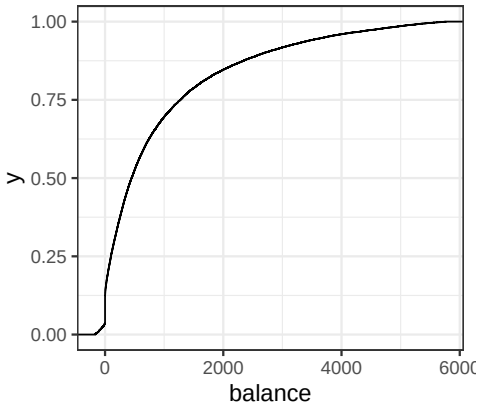
```
> ggplot(bankCut, aes(x = balance)) + facet_wrap(~ poutcome, nrow = 1) +
+                                     geom_density()
```



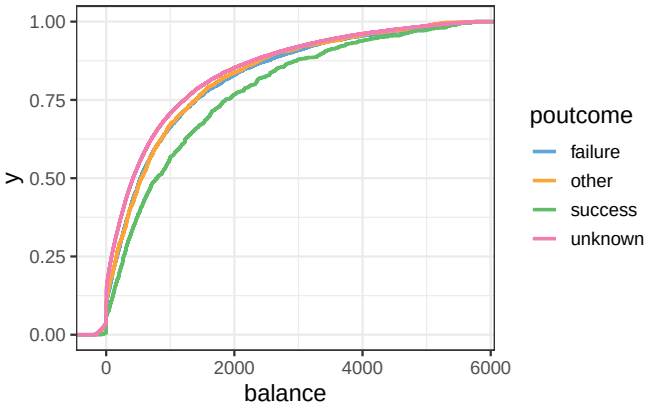
Dystrybuanta empiryczna

Dystrybuantę empiryczną narysujemy przy użyciu funkcji `stat_ecdf()`. Ostatni jej człon to skrót angielskiej nazwy: *empirical cumulative distribution function*. Do jej narysowania używane są linie, więc jeśli chcemy porównać rozkłady w grupach – jak to miało miejsce we wcześniejszym przykładzie – wtedy musimy zadecydować o sposobie ich odróżniania: kolor czy rodzaj linii. W zależności od decyzji, wybierzemy jeden z elementów estetycznych: `colour` lub `linetype`. Dopuszczalne jest użycie obu naraz – sprawdź. Poniżej zamieszczam dwa przykłady.

```
> ggplot(bankCut, aes(x = balance)) +
+ stat_ecdf()
```




```
> ggplot(bankCut, aes(x = balance, colour = poutcome)) +
+ stat_ecdf()
```



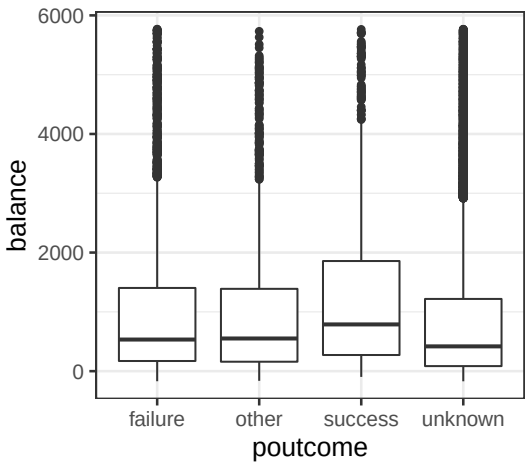
Wykres pudełko-wąsy

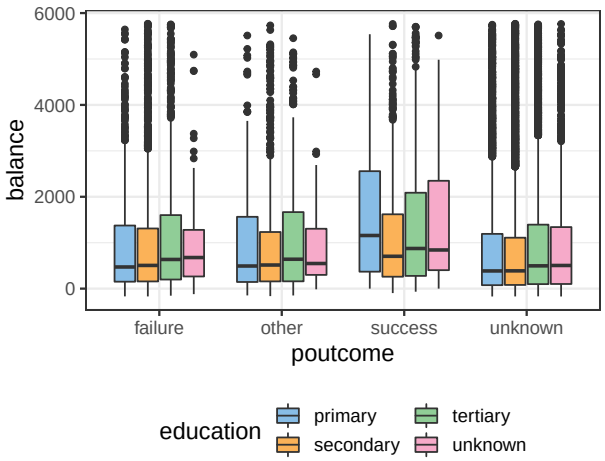
Kolejną funkcją pozwalającą ocenić rozkład zmiennej, jest `geom_boxplot()`. Rysuje ona wykres pudełko-wąsy wykorzystując statystyki pozycyjne. W definicji `ggplot()` oś X reprezentowana jest przez zmienną kategorialną, natomiast oś Y przez zmienną ciągłą – nie odwrotnie. Jeśli mamy życzenie zamienić osie, wtedy użyjmy wspomnianej wcześniej funkcji `coord_flip()`.

Kontynuujemy badanie rozkładu zmiennej saldo rachunku w zależności od wyników ostatniej kampanii marketingowej. Na drugim wykresie uwzględnimy jeszcze zmienną odnoszącą sie do poziomu edukacji (`education`). W ramach ćwiczenia zbuduj wykres, na którym w panelach pojawią się poziomy edukacji. Pamiętaj, że w tej sytuacji, będziemy mieć jeden element estetyczny mniej.

```
> ggplot(bankCut, aes(x = poutcome, y = bal
+ geom_boxplot()
```

```
> ggplot(bankCut, aes(x = poutcome, y = balance,
+         fill = education)) + geom_boxplot()
```





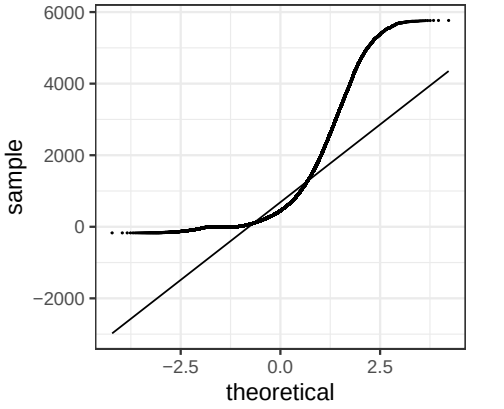
Wykres kwantyl-kwantyl

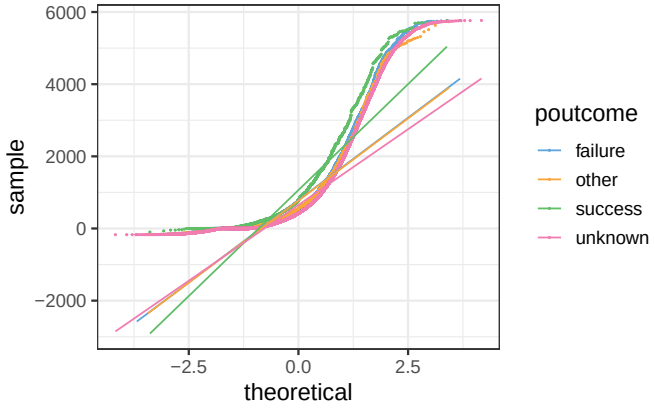
Aby sprawdzić, jak bardzo rozkład z próby jest podobny do wybranego rozkładu teoretycznego, możemy posłużyć się wykresem kwantyl-kwanty i funkcją `geom_qq()`. Swą nazwę zawdzięcza budowie: na osi X umieszczone są kwantyle teoretyczne, a na osi Y kwantyle empiryczne. Im większe podobieństwo między kwantylami, tym bardziej rozkład empiryczny jest zbliżony do rozkładu teoretycznego. W tej ocenie pomoże nam funkcja `geom_qq_line()`, która nanosi prostą na taki wykres. Idealne podobieństwo zaobserwujemy wtedy, gdy punkty będą leżały na prostej.

Jako rozkład teoretyczny możesz wybrać dowolny rozkład. W praktyce, najczęściej porównujemy rozkład naszej zmiennej z rozkładem normalnym. Dlatego domyślnym argumentem obu funkcji są kwantyle rozkładu normalnego `qnorm`. Poprzednie wykresy pokazały, że rozkład salda rachunku znacznie odbiega od rozkładu normalnego. To powinno być bardzo widoczne na wykresie. Oprócz rozkładu samego salda, zbadamy jego rozkład w zależności od kampanii marketingowej.

```
> ggplot(bankCut, aes(sample = balance))+
+ geom_qq() + geom_qq_line()
```

```
> ggplot(bankCut, aes(sample = balance,
+                     colour = poutcome)) +
+ geom_qq() + geom_qq_line()
```





6.3. Zadania

Zad. 1. Porównaj kształt funkcji gęstości rozkładu normalnego i rozkładu t-studenta, dla różnych stopni swobody, np. 3, 8, 20. W tym celu narysuj na jednym wykresie odpowiednie funkcje gęstości.

Zad. 2. Wykorzystaj zbiór danych `WholesaleCustomers.txt` i oblicz korelacje między wydatkami na produkty mleczne a pozostałymi produktami. Wartości tych korelacji wizualizować. Wersja trudniejsza: w analizie uwzględnić regiony. Wyniki zinterpretować.

Zad. 3. W pliku `stopaBezrobocia.xlsx` znajdują się wartości stopy bezrobocia (skorygowane sezonowo) dla 30 krajów w podziale na: mężczyzn, kobiety i młodych poniżej 25 roku życia. Fragment danych przedstawia poniższa tabela.

Kraj	Mężczyźni	Kobiety	Młodzi < 25 lat
Belgium	7,6	6,4	19,1
Bulgaria	7,1	6,5	16,0
Czech Republic	3,0	4,1	10,1
Denmark	5,9	6,4	12,9
Germany	4,4	3,4	6,8
…	…	…	…

Wykorzystaj dane i:

a) zaproponuj sposób wizualizacji danych;
b) przeprowadź analizę (włącznie z analizą porównawczą) z wykorzystaniem miar tendencji centralnych, miar rozproszenia oraz miar pozycyjnych;
c) zbuduj histogram dla wybranej grupy,
d) porównaj rozkłady stopy bezrobocia dla 3 grup – wykreśl funkcje gęstości estymatora jądrowego (na jednym wykresie).
e) wykreśl dystrybuanty empiryczne dla 3 grup.
f) wykorzystaj wykresy kwantyl-kwanty i odpowiedz na pytanie, czy rozkłady stopy bezrobocia tych 3 grup podlegają rozkładowi normalnemu.

Zad. 4. Badacze rynku wiedzą, że muzyka może wpływać na nastrój, a w konsekwencji na decyzje zakupowe konsumentów. W pewnym sklepie rejestrowano liczbę sprzedanych butelek win: francuskich, włoskich oraz innych, gdy nie puszczano muzyki lub puszczano muzykę: francuską akordeonową lub włoską strunową. Wyniki badań przedstawia poniższa tabela.

	Muzyka			
Wino	Żadna	Francuska	Włoska	Suma
Francuskie	30	39	30	99
Włoskie	11	1	19	31
Inne	43	35	35	113
Suma	84	75	84	243

a) Zinterpretuj wyniki w tabeli.
b) Narysuj, a następnie zinterpretuj wykres, na którym pojawią się zmienne: muzyka i wino. Czy są niezależne?

Zad. 5. Menadżerowie firmy zainteresowani są ustaleniem relacji między sprzedażą pewnego produktu a nakładami na jego promocję. W tym celu w 20 losowo wybranych punktach handlowych przeprowadzono akcję promocyjną. W poniższej tabeli zamieszczone został koszty promocji oraz sprzedaż w ujęciu wartościowym.

Punkt	Sprzedaż tys. PLN]	Promocja [PLN]
P 1	121.4	2688
P 2	143.8	2690
P 3	149.5	3401
P 4	104.5	2761
P 5	128.2	2812
P 6	115.4	2394
P 7	105.7	2402
P 8	148.6	3034
P 9	93.8	2551
P 10	168.6	3139
P 11	150.4	2834
P 12	143.3	2796
P 13	86.5	2371
P 14	135.5	2683
P 15	122.5	2610
P 16	122.3	2744
P 17	122.2	2342
P 18	126.7	2941
P 19	129.2	2406
P 20	140.3	3332

a) Zbuduj wykres i na jego postawie określ naturę związku między zmiennymi. Spróbuj przewidzieć wartość współczynnika korelacji Pearsona.
b) Oblicz współczynniki korelacji: Pearsona i rang Spearmana. Wyniki zinterpretuj.
c) Przemnóż sprzedaż przez 1000 i oblicz współczynnik korelacji Pearsona. Czy się zmienił? Odpowiedź uzasadnij odwołując się do teorii.

Zad. 6. Wykorzystaj raport `aktywnosc_ekonomiczna_ludnosci_polski_iv_kwartal_2018` i:
a) znajdź 2 wykresy, które według ciebie powinny być poprawione. Zaproponuj lepszy sposób wizualizacji i przedstaw do na wykresie.
b) wybierz 2 tabele, których wartości zwizualizujesz.

Zad. 7. Dla każdego zadania z rozdziału 5 utwórz wykresy. Chodzi o zadania końcowe zaczynające się na stronie 53.

ROZDZIAŁ 7

Estymacja i testowanie hipotez

7.1. Estymacja przedziałowa średniej μ i proporcji p

Do zbudowania przedziału ufności dla średniej wykorzystamy następującą funkcję:

```
t.test(x, conf.level = 0.95)
```

Ma ona dwa argumenty. Pierwszy `x` – jest wektorem obserwacji, drugi `conf.level` – odnosi się do poziomu ufności. O tej funkcji powiem więcej, gdy przejdziemy do testowania hipotez statystyczny. Jedyną informację na jakiej musisz sie skupić to przedział ufności, który może zaczynać się następująco: 95 percent confidence interval. Zobacz poniższy kod.

```
> wydatki <- c(30, 25, 16, 22, 34, 54, 23, 28, 23, 21)
> t.test(wydatki, conf.level = 0.9)

One Sample t-test

data:  wydatki
t = 8, df = 9, p-value = 2e-05
alternative hypothesis: true mean is not equal to 0
90 percent confidence interval:
 21.5 33.7
sample estimates:
mean of x
     27.6
```

W wypadku estymacji proporcji użyjemy pakietu `Hmisc`, w którym funkcja:

```
binconf(x, n, alpha=0.05, method=c("wilson", "exact", "asymptotic", "all"))
```

pozwala oszacować przedział w sposób dokładny (`exact`) lub go aproksymować. Jeżeli próba jest duża, wtedy można założyć, że statystyka na bazie której taki przedział jest konstruowany, ma rozkład normalny (`asymptotic`). Przy niewielkich próbach najlepiej użyć metody Wilsona (nawet bardziej preferowana niż szacowanie dokładne). Wymagane argumenty funkcji to: `x` – liczba sukcesów, `n` – liczba obserwacji.

Zobaczmy, jak wyglądają oszacowania przedziałów w zależności od rozmiaru próby i wykorzystanej metody estymacji.

```
> ## gdy pakiet niezainstalowany użyj: install.packages("Hmisc")
> library(Hmisc)
> binconf(150, 300,  alpha = 0.05, method = "all") # różnice marginalne
           PointEst Lower Upper
Exact           0.5 0.442 0.558
Wilson          0.5 0.444 0.556
Asymptotic      0.5 0.443 0.557

> binconf(20, 40,  alpha = 0.05, method = "all") # większe różnice, bo próba mała
           PointEst Lower Upper
Exact           0.5 0.338 0.662
```

```
Wilson          0.5 0.352 0.648
Asymptotic      0.5 0.345 0.655
```

7.2. Testowanie hipotez

7.2.1. Testy dla frakcji labo proporcji. Do weryfikacji hipotezy o frakcji w jednej populacji lub równości frakcji w wielu populacjach wykorzystamy następującą funkcję:

```
prop.test(x, n, p = NULL, alternative = c("two.sided", "less", "greater"),
          conf.level = 0.95, correct = TRUE)
```

w której:

- `x` – liczba sukcesów (gdy jest jedna populacja) lub wektor liczby sukcesów (przynajmniej dwie populacje); alternatywnie możesz podać tabelę, przy czym pierwsza kolumna odnosi się do sukcesu, a druga do porażki (kolejność jest ważna);
- `n` – liczba prób (rozmiar próby); nie podajesz, jeżeli wcześniejszym argumentem jest tabela;
- `p` – weryfikowane prawdopodobieństwo/frakcja sukcesów; gdy nie podasz, wtedy domyślnie przyjmowane jest 0.5;
- `correct` – czy poprawka Yatesa na ciągłość ma być uwzględniona.

Wykorzystamy powyższą funkcję i zbiór danych `satysfakcja.dat`, aby zweryfikować dwie hipotezy.

Hipoteza 1: Frakcja osób wierzących w życie po śmierci (`wiaraZyciePo`) jest równa 3/4 (0.75).

Hipoteza 2: Frakcja osób niewietrzących w życie po śmierci (`wiaraZyciePo`) jest taka sama w każdej grupie wykształcenia (`edukacja`). Innymi słowy grupy odnoszące się do poziomu wykształcenia są jednorodne pod względem braku wiary w życie po śmierci.

```
> ## Hipoteza 1
> saty <- read.table("dane/satysfakcja.dat", header=TRUE, sep="\t")
> table(saty$wiaraZyciePo)

Nie Tak
236 974

> prop.test(974, 236+974, p=0.75)

1-sample proportions test with continuity correction

data:  974 out of 236 + 974, null probability 0.75
X-squared = 19, df = 1, p-value = 1e-05
alternative hypothesis: true p is not equal to 0.75
95 percent confidence interval:
 0.78 0.83
sample estimates:
  p
0.8
```

```
>
> ## Hipoteza 2
> (tab <- table(saty$edukacja, saty$wiaraZyciePo))

                 Nie Tak
  Licencjat       29 164
  Magisterium     33  76
  Policealna      13  67
  Szkola srednia 122 529
  Szkola zawodowa 39 138

> ## Zobaczmy jak wygląda rozkład frakcji
> prop.table(tab, margin = 1) # procent sumuje się do 100 w wierszach

                  Nie  Tak
  Licencjat      0.15 0.85
  Magisterium    0.30 0.70
  Policealna     0.16 0.84
  Szkola srednia 0.19 0.81
  Szkola zawodowa 0.22 0.78

> prop.test(tab) #weryfikacja drugiej hipotezy

5-sample test for equality of proportions without continuity correction

data:  tab
X-squared = 12, df = 4, p-value = 0.02
alternative hypothesis: two.sided
sample estimates:
prop 1 prop 2 prop 3 prop 4 prop 5
  0.15   0.30   0.16   0.19   0.22
```

7.2.2. Testy niezależności χ^2. Weryfikację hipotezy o niezależności dwóch zmiennych kategorialny przeprowadzamy za pomocą testu χ^2 Pearsona, który został zaimplementowany w funkcji `chisq.test()`. Wymaganym argumentem tej funkcji jest tabela utworzona z dwóch zmiennych, w której komórki są liczebnościami.

Załóżmy, że chcemy zweryfikować hipotezę, że poziom deklarowanego szczęścia (`szczescie`) oraz wykształcenie (`edukacja`) są niezależne. Wykorzystujemy zbiór danych `satysfakcja.dat` oraz interesujące nas zmienne, aby zapisać:

```
> (tab <- table(saty$edukacja, saty$szczescie))

                 Bardzo szczesliwy Dosc szczesliwy Nieszczesliwy
  Licencjat                     79             100            14
  Magisterium                   30              67            12
  Policealna                    32              39             9
  Szkola srednia               185             381            85
  Szkola zawodowa               42             102            33

> chisq.test(tab)

Pearson's Chi-squared test
```

```
data:  tab
X-squared = 26, df = 8, p-value = 0.001
```

Jeśli sądzimy, że asymptotyczny rozkład statystyki testowej odbiega od rozkładu χ^2, wtedy możemy wykorzystać procedurę symulacji do oszacowania *p-value*. Wystarczy, że użyjemy dodatkowych argumentów funkcji `chisq.test()`. Ilustruję to przykładem.

```
> chisq.test(tab, simulate.p.value = TRUE, B = 2000) # B -ile powtórzeń w Monte Carlo

Pearson's Chi-squared test with simulated p-value (based on 2000 replicates)

data:  tab
X-squared = 26, df = NA, p-value = 0.001
```

7.2.3. Testy zgodności z rozkładem normalnym. W estymacji przedziałowej czy weryfikacji niektórych hipotez statystycznych przyjmujemy założenie, że rozkład badanej cechy jest normalny. Do weryfikacji takiego założenia możemy wykorzystać dostępną w pakiecie podstawowym `stats` funkcję `shapiro.test()`, która odnosi się do testu Shapiro-Wilka. Jeśli zdecydujemy się na pakiet `nortest`, wtedy mamy do dyspozycji dodatkowych 5 testów: `ad.test()` (Anderson-Darling), `cvm.test()` (Cramer-von Mises), `lillie.test()` (Kolmogorov-Smirnov z poprawką Lillieforsa), `pearson.test()` (Pearson χ^2, `sf.test()` (Shapiro-Francia). Argumentem wejściowym we wszystkich funkcjach jest wektor, którego wartości odpowiadają badanej zmiennej.

Zweryfikujmy hipotezę, że liczba godzin pracy (`ileGodzPracuje`) ma rozkład normalny. Hipoteza alternatywna głosi, że zmienna ma rozkład inny niż rozkład normalny. Dla porównania użyjemy dwóch testów.

```
> ## Jeżeli pakiet nie jest zainstalowany, to użyj: install.packages("nortest")
> library(nortest)
> shapiro.test(saty$ileGodzPracuje) # Shapiro-Wilk

Shapiro-Wilk normality test

data:  saty$ileGodzPracuje
W = 0.9, p-value <2e-16

> ad.test(saty$ileGodzPracuje) # Anderson-Darling

Anderson-Darling normality test

data:  saty$ileGodzPracuje
A = 18, p-value <2e-16
```

7.2.4. Testy dla wartości średniej lub mediany. Hipotezę o wartości średniej weryfikujemy w jednej, dwóch lub wielu populacjach. Pierwsza możliwość zakłada, że średnia jest równa zadanej wartości. Druga grupa hipotez odnosi się do testowania równości wartości średnich w dwóch niezależnych próbach[1]. Jeżeli obserwacje zestawione są w pary, wtedy mówimy, że próby są zależne, a weryfikowana hipoteza zakłada, że różnica między wartościami średnimi jest równa zadanej wartości (najczęściej to zero). Te trzy warianty hipotez

[1] Rozszerzeniem, na przypadek wielu prób, jest analiza wariancji ANOVA.

weryfikujemy tą samą funkcją – zmieniamy, w zależności od potrzeby, jej argumenty:

```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95)
```

gdzie:

`x,y`	–	wartości numeryczne zmiennej lub zmiennych
`alternative`	–	test dwustronny (`two.sided`), jednostronny dla alternatywnej hipotezy (`less` - lewostronny; `greater` - prawostronny);
`mu`	–	założenie dotyczące wartości średniej;
`paired`	–	czy obserwacje są zestawione w pary;
`var.equal`	–	czy założono równość wariancji; w wypadku braku równości zostanie wykorzystana korekta Welcha.

W poniższych przykładach będę wykorzystywał zbiór danych `satysfakcja.dat`. Zwracam uwagę na dwie sprawy. Pierwsza – przy weryfikacji hipotez rezygnuję ze sprawdzania, czy badana cecha lub cechy mają rozkład normalny. Przykłady ilustrują tylko wykorzystanie funkcji `t.test()`. Druga – jeśli nie wspomnę o poziomie istotności, to zakładam, że $\alpha = 0.05$.

Zweryfikujmy hipotezę, że liczba godzin pracy (`ileGodzPracuje`) jest różna od 8 godzin. Stawiamy hipotezę zerową i alternatywną: $H_0: \mu = 8$ i $H_1: \mu \neq 8$.

```
> ## H_1: m =/= 8
> t.test(saty$ileGodzPracuje, mu=8, alternative = "two.sided")

One Sample t-test

data:  saty$ileGodzPracuje
t = -12, df = 1209, p-value <2e-16
alternative hypothesis: true mean is not equal to 8
95 percent confidence interval:
 6.5 6.9
sample estimates:
mean of x
      6.7
```

Oszacowana średnia liczba godzin pracy wyniosła: 6.69. Ponieważ p-wartość = 0, więc hipotezę H_0 odrzucamy na korzyść hipotezy alternatywnej (przy uwzględnieniu $\alpha = 0.05$). Daje nam to podstawę do twierdzenia, że liczba godzin pracy jest różna od 8.

Interesujące może być również pytanie, czy przeciętna liczba godzin pracy jest taka sama dla kobiet i mężczyzn (`plec`). Prowadzi nas ono do następujących hipotez: $H_0: \mu_k = \mu_M$ i $H_1: \mu_k \neq \mu_M$, gdzie indeks odnosi się do kobiet bądź mężczyzn. W tym wypadku mamy do czynienia z dwoma niezależnymi próbami. Argumentami wejściowymi funkcji `t.test()` mogłyby być wektory `x` i `y` reprezentujące odpowiednio liczbę godzin pracy mężczyzn i kobiet. Biorąc pod uwagę konieczność przygotowania takich wektorów (na podstawie ramki danych), wygodniej jest użyć formuły. Zobacz poniżej.

```
> ## ogólna formuła: zmiennaIlościowa ~ zmiennaKategorialna
> t.test(ileGodzPracuje ~ plec, data=saty) # data= nazwaZbioruDanych

Welch Two Sample t-test

data:  ileGodzPracuje by plec
t = -0.2, df = 1182, p-value = 0.8
```

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.47  0.38
sample estimates:
  mean in group Kobieta mean in group Mezczyzna
                    6.7                     6.7
```

Zwróćmy uwagę na przyjęte *implicite* założenie, że wariancje w obu grupach nie są jednakowe. Świadczy o tym wykorzystany tzw. test Welcha (korekta na `df`). Jeśli chcemy z tej korekty zrezygnować, powinniśmy hipotezę o równości wariancji w obu grupach zweryfikować, czyli $H_0\colon \sigma_K^2 = \sigma_M^2$ wobec $H_1\colon \sigma_K^2 \neq \sigma_M^2$. W sprawdzeniu tej hipotezy pomocna będzie funkcja `var.test()`, której odpowiada test F oparty na statystyce F-Snedecora. Również i tutaj musimy sprawdzić założenia jego stosowalności (jest nim rozkład normalny cech).

```
> ## Krok 1: hipoteza o równości wariancji
> var.test(ileGodzPracuje ~ plec, data=saty)

F test to compare two variances

data:  ileGodzPracuje by plec
F = 1, num df = 561, denom df = 647, p-value = 0.9
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.86 1.19
sample estimates:
ratio of variances
                 1
```

Powyższy test nie daje podstaw do odrzucenia hipotezy zerowej, więc w kolejnym kroku zbadamy równość średnich z pominięciem korekty (zakładamy równość wariancji).

```
>  ## Krok 2: hipoteza o równości średnich, ale bez korekty
>  t.test(ileGodzPracuje ~ plec, data=saty, var.equal=TRUE)

Two Sample t-test

data:  ileGodzPracuje by plec
t = -0.2, df = 1208, p-value = 0.8
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.47  0.38
sample estimates:
  mean in group Kobieta mean in group Mezczyzna
                    6.7                     6.7
```

Gdy rozkład cechy nie spełnia założeń i musimy zrezygnować z testu `t.test()`, wtedy możemy posłużyć się testem nieparametrycznym Wilcoxona i funkcją `wilcox.test()`. Jej argumenty wejściowe prawie pokrywają sie z omówionymi argumentami funkcji `t.test()`. Zobacz na poniższy przykład.

```
> ## Testy nieparametryczne; alternatywa dla t.test()
> # Przykład 1: H0: m <= 8; H1: m > 8
> wilcox.test(saty$ileGodzPracuje, mu=8, alternative = "two.sided")

Wilcoxon signed rank test with continuity correction
```

```
data:  saty$ileGodzPracuje
V = 2e+05, p-value <2e-16
alternative hypothesis: true location is not equal to 8

> # Przykład 2: H0: m_k = m_M; H1: m_K =/= m_M
> wilcox.test(ileGodzPracuje ~ plec, data=saty)

Wilcoxon rank sum test with continuity correction

data:  ileGodzPracuje by plec
W = 2e+05, p-value = 0.8
alternative hypothesis: true location shift is not equal to 0
```

7.3. Zadania

Zad. 1. Dla zmiennej saldo rachunku (`balance`) ze zbioru danych `bankFull.csv` zbudować przedział ufności dla średniej. Wynik zinterpretować.

Zad. 2. Dla zmiennej mówiącej o nieregulowaniu należności (`default`) ze zbioru danych `bank` zbudować przedział ufności. Wynik zinterpretować.

Zad. 3. Dla wybranego zbioru danych z katalogu `dane` oraz wybranych zmiennych zweryfikować wszystkie omówione w tym rozdziale hipotezy statystyczne. Pamiętać o sformułowaniu hipotezy zerowej i alternatywnej, założeniach oraz interpretacji wyników.

Bibliografia

[1] Bache, K. and Lichman, M. (2013), 'UCI machine learning repository'.
URL: *http://archive.ics.uci.edu/ml*

[2] Chang, W. (2013), *R Graphics Cookbook*, Oreilly and Associate Series, O'Reilly Media, Incorporated.

[3] Crawley, M. (2012), *The R Book*, Wiley.

[4] Gągolewski, M. (2014), *Programowanie w języku R*, Wydawnictwo Naukowe PWN.

[5] Maindonald, J. and Braun, W. (2010), *Data Analysis and Graphics Using R: An Example-Based Approach*, Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press.

[6] Matloff, N. and Matloff, N. (2011), *The Art of R Programming: A Tour of Statistical Software Design*, No Starch Press.

[7] Muenchen, R. A. (2011), *R for SAS and SPSS Users*, Springer Series in Statistics and Computing, Springer.

[8] Murrell, P. (2011), *R Graphics, Second Edition*, Chapman & Hall/CRC the R series, Chapman & Hall/CRC Press, Boca Raton, FL.

[9] Spector, P. (2008), *Data Manipulation with R*, Springer.

[10] Wickham, H. (2009), *ggplot2: Elegant Graphics for Data Analysis*, Springer.

[11] Wickham, H. (2014), 'Tidy data', *Journal of Statistical Software, Articles* **59**(10), 1–23.
URL: *https://www.jstatsoft.org/v059/i10*

[12] Wilkinson, L. (2005), *The Grammar of Graphics*, Springer-Verlag, Berlin, Heidelberg.

Indeks funkcji