

Kurs R - Tutorial

Agnieszka Strzałka

2021-08-30

Contents

1 Wstęp	5
2 Skąd sięgnąć R	7
2.1 Pomoc	7
2.2 Packages	8
2.3 R Studio	9
3 Dane	11
3.1 Wczytanie danych	11
3.2 Zapisanie danych	13
3.3 Podstawowe typy danych w R	14
4 Analiza i transformacja danych	27
4.1 Format uporządkowany (wąski)	27
4.2 Pipes (potoki)	31
4.3 Sortowanie	33
4.4 Filtrowanie	36
4.5 Grupowanie i obliczanie nowych zmiennych	40
4.6 Łączenie tabel	47
5 Wykresy - pakiet ggplot2	53
5.1 Używane dane	54
5.2 Histogram, boxplot i inne	55
5.3 Barplot - wykres słupkowy	79

5.4	Średnia, mediana itp. na wykresie	85
5.5	Wykres punktowy i liniowy	96
5.6	Kolory, osie, panele	116
5.7	Motyw (theme)	149
5.8	Różne	157
5.9	Rozszerzenia ggplot2	165
6	Wykresy - pakiet podstawowy	171
6.1	Histogram - hist	171
6.2	Wykres rozrzutu - plot	174
6.3	Wykres pudełkowy - boxplot	177
6.4	Wykres słupkowy - barplot	179
6.5	Wykres mozaikowy - mosaicplot	181
6.6	Wykres kwantylowy - qqPlot	183
6.7	Diagram venna	185
6.8	Wykres korelacji - corrplot	187
6.9	Clustering - analiza skupień	189
7	Statystyka	195
7.1	Podstawowe statystyki opisowe	195
7.2	Liczby pseudolosowe	197
7.3	Podstawowe testy statystyczne	200
7.4	Modelowanie - czyli jak dopasować linię trendu	217
7.5	Analiza ANOVA	227
8	Miscellaneous	233

Chapter 1

Wstęp

Książka jest dodatkiem do kursu R i ma stanowić zbiór przykładów, które mogą zostać wykorzystane podczas pisania własnych skryptów R. Materiały będą na bieżąco aktualizowane tak aby wszystkie przedstawione przykłady były możliwe do odtworzenia.

Większość przedstawionych przykładów będzie się opierać na funkcjach zawartych w pakietach tidyverse, które tworzą spójną całość obejmującą wczytanie i analizę danych oraz ich wizualizację. Chociaż wiele z przedstawionych zadań może być wykonane korzystając z podstawowych funkcji R uważa, że nauczenie się od razu korzystania z pakietów tidyverse za bardziej korzystne gdyż zawarte w nich funkcje są często łatwiejsze i bardziej intuicyjne w stosowaniu, szczególnie gdy ktoś nie ma dużego doświadczenia w programowaniu ;)

```
##  
## R is the lingua franca of statistical research. Work in all other languages  
## should be discouraged.  
##      -- Jan de Leeuw (as quoted by Matt Pocernich on R-help)  
##      JSM 2003, San Francisco (August 2003)
```


Chapter 2

Skądściągnąć R

- Postawowe środowisko R najlepiejściągnąć bezpośrednio ze strony R project
- Polecam równieżściągnąć R Studio, które bardzo ułatwia pracę z R
- Podstawowe pakiety zostaną zainstalowane razem z R, kolejne można po brać bezpośrednio przez R Studio (zakładka Packages), bardziej bioinformatyczne pakiety znajdują się na stronie bioconductor, wiele pakietów można też pobrać bezpośrednio ze strony Github korzystając z pakietu devtools.

```
##  
## Friends don't let friends use Excel for statistics!  
## -- Jonathan D. Cryer (about problems with using Microsoft Excel for  
## statistics)  
## JSM 2001, Atlanta (August 2001)
```

2.1 Pomoc

- Pomoc do funkcji można otworzyć bezpośrednio w R wpisując: `?nazwa_funkcji` albo naciskając F1 podczas wpisywania nazwy funkcji, jednak opisy często są dość enigmatyczne i zakładają już pewne zrozumienie tematu
- Odpowiedzi na konkretne pytania najlepiej szukać w internecie wpisując po prostu: "How do sth R", najczęściej pojawiająca się strona to Stack Overflow
- Ciekawe pomysły i tutoriale pojawiają się też na stronie R-bloggers
- Interaktywny kurs R można też znaleźć na stronie datacamp, ale obecnie tylko pierwsze lekcje każdego kursu są darmowe
- Pomoc do ggplot2

- Polecam też książkę Przemysława Biecka “Przewodnik po pakiecie R”, jedna z niewielu jakie są po polsku, pierwsze rozdziały można bezpłatnie pobrać ze strony autora
- Introduction to R for Biologists
- How to make any plot in ggplot2?
- Fundamentals of Data Visualization
- R Graphics Cookbook, 2nd edition
- R for Data Science
- No i oczywiście ja chętnie pomogę :)

```
##  
## You need to get the hang of reading the online help. The information required  
## is actually there in ?dotchart --- it's just tersely and obscurely expressed. A  
## certain degree of optimism is required. You need to ***believe*** that the  
## information is there; then ask yourself "What could they possibly mean by what  
## they have written that would tell me what I need to know?".  
## -- Rolf Turner (on reading the help pages)  
## R-help (June 2013)
```

2.2 Packages

Żeby użyć funkcję z danego pakietu, który nie należy do podstawowych należy go najpierw zainstalować (Install w zakładce Packages), a potem załadować (funkcje `library(nazwa)` albo `require(nazwa)`). Można też włączać pakiety w zakładce Packages.

Dobrze jest ładować tylko te pakiety, które są faktycznie potrzebne - nie przeciążamy pamięci i niektóre nazwy funkcji mogą się powtarzać w różnych pakietach.

Jeżeli ładowanie całego pakietu jest niepotrzebne albo prowadzi do konfliktów można odwołać się do konkretnej funkcji przy pomocy `::` np. `readxl::read_excel()`.

Każdy pakiet zawiera podstawowy opis funkcji, niektóre posiadają bardziej rozbudowane przykłady analiz jakie można przy ich pomocy wykonać - vignette. Jeżeli chcemy sprawdzić które pakiety zawierają winietki można to zrobić funkcją `browseVignettes`.

2.2.1 Lista pakietów, które pojawiają się w książce (uwaga może być niekompletna)

Przed rozpoczęciem należy mieć zainstalowane następujące pakiety: ggplot2, tidyr, knitr, dplyr, readr, readxl oraz najlepiej posiadać najnowszą wersję R (4.1.0 - “Camp Pontanezen”) i R Studio (przynajmniej 1.4).

Pozostałe pakiety można pobrać tylko jeśli będą potrzebne.

Inne pakiety jakie pojawiają się to: w części dotyczącej wykresów: car, likert, gplots, VennDiagram, corrplot, hexbin, aplpack, GGally, ggthemes, ggthemr i w części dotyczącej statystyki i obróbki danych: car, MASS, nortest, modeest, moments, agricolae, drc, broom.

2.3 R Studio

Jest to obecnie najpopularniejszy dostępny edytor R, pozwalający na tworzenie projektów, pisanie i zarządzanie skryptami, pozwalający na łatwy dostęp do historii wpisywanych komend i tworzonych wykresów. Został zintegrowany z wieloma przydatnymi pakietami np. knitr, który pozwala tworzenie raportów w języku markdown, latex, prezentacji multimedialnych itp.

R Studio pozwala również na założenie projektu do konkretnego zadania, wszystkie pliki tworzone w trakcie pracy (wykresy, tabele, skrypty) zostaną umieszczone w folderze przypisany do projektu. Jeżeli nasze dane wejściowe umieścimy w tym samym miejscu nie będzie konieczne podawanie całej ścieżki dostępu przy wczytywaniu pliku. Wykorzystanie projektów ułatwia uporządkowanie pracy. Każdy projekt można też poddać kontroli wersji przy pomocy Git i Github bezpośrednio z Rstudio. Więcej informacji na ten temat można znaleźć w Happy Git with R.

Cały Tutorial to zbiór skryptów napisanych w markdown, które zostały połączone w książkę dzięki pakietowi R bookdown. Wszystkie przykłady można skopiować albo przepisać do własnych skryptów R. Trzeba jedynie pamiętać o wcześniejszym załadowaniu odpowiednich pakietów i ewentualnie danych. Samodzielne wpisywanie nazw funkcji przyśpiesza proces zapamiętywania, a korzystając z autouzupełniania trzeba tak naprawdę pamiętać 2-3 pierwsze litery :)

Chapter 3

Dane

3.1 Wczytanie danych

Dane najłatwiej wczytać z plików .txt lub .csv. Można również z plików .xlsx lub .xls ale często trwa to dłużej i może wymagać zainstalowanych innych pakietów/programów.

W każdym przypadku niezbędne jest podanie ścieżki dostępu do pliku. Jeżeli znajduje się on folderze projektu to wystarczy jego nazwa. Dobrą praktyką jest przechowywanie danych w osobnym katalogu w obrębie projektu - wtedy ścieżka może wyglądać np. data/dane_1.txt. Jeżeli plik z danymi jest przechowywany gdzie indziej konieczne jest podanie pełnej ścieżki dostępu, ale można w tym wypadku korzystać z autouzupełniania klawiszem Tab.

3.1.1 Wczytanie danych z plików tekstowych

R Studio posiada funkcję Import Dataset (zakładka Environment), która pozwala na wczytanie danych wraz z wyborem podstawowych opcji jak separator, separator dziesiętny, obecność nagłówków. Opcja ta obejmuje podstawowe funkcje R oraz pakiety readr i readxl będące częścią tidyverse. Na początek jest łatwiej korzystać z opcji Load Dataset, ponieważ po jej użyciu zostaje również wygenerowany odpowiedni kod R, który można wkleić do własnych skryptów tak aby te same lub podobne dane wczytać już bez problemu następnym razem.

Można również wczytać dane wpisując komendę (wygodne przy pisaniu skryptów) `read.table` albo `read.csv`. Argumenty: file określa nam plik, który wczytujemy, header - nagłówki, sep - separator np. “/t” to tabulator, dec - separator dziesiętny (domyślnie kropka), quote - obecność ””. W przypadku nagłówków kolumn wpisanie `header=TRUE` spowoduje, że pierwszy

wiersz naszej tabeli zostanie potraktowany jako tytuły kolumn, `header=FALSE` oznacza domyślne nazwy kolumn - V1, V2, ...

W pakiecie `readr` znajdują się analogiczne funkcje: * `read_delim` - do plików tekstowych, funkcja spróbuje zgadnąć jak rozdzielone są kolumny, można podać argumentem `delim` * `read_csv` i `read_csv2` - do plików csv, odpowiednio do danych wykorzystujących kropki i przecinki jako separatory dziesiętne * `read_tsv` i `read_table` - do plików gdzie kolumny są rodzielone przy pomocy `tab`.

Wczytane dane będą widoczne w zakładce Environment. Kliknięcie na nazwę tabeli spowoduje otwarcie widoku danych podobnego do arkusza kalkulacyjnego. Można w nim dane sortować i filtrować, ale nie edytować.

Pierwsze albo ostatnie wiersze można zobaczyć używając funkcji `head` albo `tail`. Strukturę danych pokaże funkcja `str`, a podstawowe informacje `summary`. Dobrze jest po wczytaniu danych sprawdzić czy wyglądają faktycznie tak jak miały ;)

```
dane1 <- read.table(file = "data/dane1.txt", header=TRUE, quote=""")  
  
head(dane1)  
  
##      pomiar Szczep warunki  
## 1 3.389738     A     1  
## 2 5.992065     A     1  
## 3 4.464553     A     1  
## 4 4.546082     A     1  
## 5 6.521751     A     1  
## 6 5.713088     A     1  
  
str(dane1)  
  
## 'data.frame': 1800 obs. of 3 variables:  
## $ pomiar : num 3.39 5.99 4.46 4.55 6.52 ...  
## $ Szczep : chr "A" "A" "A" "A" ...  
## $ warunki: int 1 1 1 1 1 1 1 1 1 1 ...  
  
summary(dane1)  
  
##      pomiar      Szczep      warunki  
## Min.   :1.434  Length:1800      Min.   :1.0  
## 1st Qu.:4.450  Class :character  1st Qu.:1.0  
## Median :5.478  Mode  :character  Median :1.5  
## Mean   :5.846                           Mean   :1.5  
## 3rd Qu.:6.743                           3rd Qu.:2.0  
## Max.   :27.953                          Max.   :2.0
```

3.1.2 Wczytanie danych z plików excel

Do wczytania danych z plików xlsx można wykorzystać funkcję `read_excel` z pakietu `readxl`. Przy pracy z excelem należy jednak pamiętać żeby wczytywane dane nie zawierały formuł albo połączonych komórek oraz, że funkcja wczyta dane zawarte tylko w jednym arkuszu (sheet), ale można wybrać z której przy pomocy argumentu `sheet`.

```
library(readxl)

data_excel <- read_excel('data/test.xlsx')

data_excel

## # A tibble: 16 x 3
##       A     B     C
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     1     3     4
## 3     1     4     5
## 4     1     5     6
## 5     1     6     7
## 6     1     7     8
## 7     1     8     9
## 8     1     9    10
## 9     1    10    11
## 10    1    11    12
## 11    1    12    13
## 12    1    13    14
## 13    1    14    15
## 14    1    15    16
## 15    1    16    17
## 16    1    17    18
```

3.2 Zapisanie danych

Zapisać tabelę danych można korzystając z funkcji `write.table`, wystarczy określić, co chcemy zapisać i nazwę pliku wyjściowego. Możemy zapisywać pliki w formacie np. `txt`, `csv`. Jeżeli nie określmy ścieżki dostępu plik zostanie zapisany w folderze projektu.

Można również dopisywać dane do istniejącego pliku ustawiając parametr `append = TRUE`. Domyślnie ten argument jest zawsze ustawiony jako `append =`

FALSE i jeżeli w nazwie pliku podamy istniejący plik to **zostanie on nadpisany bez ostrzeżenia**.

Można też dane zapisać w formacie rds. Można je potem otworzyć tylko w R, ale można w takim formacie zapisać każdy obiekt R, nie tylko tabele. Do zapisania danych służy funkcja saveRDS, a do wczytanie readRDS.

```
write.table(dane1, "data/dane4.txt")
```

3.3 Podstawowe typy danych w R

W R dane do zmiennej przypisujemy korzystając ze strzałki ;) <- albo ->. Można też użyć =

Do najbardziej podstawowych typów danych należą: typ liczbowy, znakowy, logiczny i czynnikowy (o tym później). Typ danych można sprawdzić funkcją **class**

Najczęściej wykorzystywane rodzaje danych to wektor i ramka danych. Poza tym to m.in. macierz i lista. W R można również pisać własne funkcje.

3.3.1 Wektor (vector)

Wektor to ciąg elementów tego samego typu np. liczbowy, znakowy. W R nawet pojedyncza cyfra jest wektorem. Wektor można stworzyć korzystając z funkcji:

- c. Sekwencję liczb można zapisać jako 1:10 - utworzy wektor liczb od 1 do 10.
- Przy bardziej skomplikowanych sekwencjach można użyć funkcji **seq**, ustawiamy start, koniec i co ile ma być kolejny element
- funkcję **rep**, podajemy jakie elementy mają zostać powtózone i ile razy. **each** - powtarzanie każdego elementu, **times** - powtarzanie całego wektora.

```
a <- c(1,2,5,8)
a
```

```
## [1] 1 2 5 8
```

```
b <- c("A", "B", "V")
b
```

```

## [1] "A" "B" "V"

c <- 1:15
c

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

d <- seq(10, 100, by=10)
d

## [1] 10 20 30 40 50 60 70 80 90 100

e <- rep(c(1,2,3), times=3)
e

## [1] 1 2 3 1 2 3 1 2 3

```

Do poszczególnych elementów wektora można się odwołać stosując nawiasy: `wektor[x]`, gdzie x to numer elementu. Można się odwołać do kilku elementów jednocześnie np. `wektor[1:3]`, `wektor[c(2,4)]`. Numeracja rozpoczyna się od 1. Elementy wektora mogą również mieć swoje nazwy i wtedy można je także wykorzystać do wyświetlania danych.

Ilość elementów wektora podaje funkcja `length`.

```

a <- c(1:5)
a[2]

## [1] 2

b <- c( "jeden" = 1, "dwa" = 2, "trzy" = 3)
b[2]

## dwa
## 2

```

```
b["dwa"]
```

```
## dwa
## 2
```

```
length(a)
```

```
## [1] 5
```

Na wektorach można wykonywać wszystkie operacje matematyczne. Przykładowo jeżeli dodamy do siebie dwa wektory to zawsze pierwszy element zostanie dodany do pierwszego elementu z drugiego wektora itd.

```
1:10 + 11:20
```

```
## [1] 12 14 16 18 20 22 24 26 28 30
```

```
1:10 * 11:20
```

```
## [1] 11 24 39 56 75 96 119 144 171 200
```

3.3.2 Ramka danych (data frame)

Ramka danych to po prostu kilka wektorów ułożonych w tabelę. Wektory mogą być różnego typu, powinny być tej samej długości. Jeżeli mają różną długość można uzupełnić brakujące elementy stosując NA - brak danych

Skoro każda kolumna ramki danych jest wektorem to można stosować na nich te same operacje matematyczne np. dodawać albo dzielić.

Ramkę danych tworzymy przy użyciu funkcji `data.frame`

Jeżeli chcemy coś zmienić w ramce danych używamy `as.data.frame`

Podstawowe informacje o ramce danych

Funkcja	Opis
<code>nrow</code>	liczba wierszy
<code>ncol</code>	liczba kolumn
<code>colnames</code>	nazwy kolumn
<code>rownames</code>	nazwy wierszy
<code>dim</code>	wymiary

```
x <- data.frame(a=1:5, b=rep("A",5), c=c(T,T,T,F,NA))
x

##   a b     c
## 1 1 A  TRUE
## 2 2 A  TRUE
## 3 3 A  TRUE
## 4 4 A FALSE
## 5 5 A    NA

colnames(x)

## [1] "a" "b" "c"
```

Do elementów ramki danych również można się odwołać przy pomocy nawiasów: `ramka[x,y]`, gdzie x to numer wiersza, a y numer kolumny. Jeżeli podamy jedynie numer kolumny otrzymamy wszystkie jej elementy.

Innym sposobem jest wykorzystanie nazw kolumn i \$ np `ramka$kolumna_1`. W ten sposób można łatwo dodać nowe kolumny do ramki danych.

Podobnie możemy usuwać kolumny i wiersze z ramki np. `ramka<-ramka[,-1]` usunie pierwszą kolumnę.

```
ramka <- data.frame(kol1=c(1:10), kol2=c(11:20))

ramka

##      kol1 kol2
## 1      1    11
## 2      2    12
## 3      3    13
## 4      4    14
## 5      5    15
## 6      6    16
## 7      7    17
## 8      8    18
## 9      9    19
## 10    10   20

ramka$kol_3 <- ramka$kol1 + ramka$kol2

ramka
```

```
##      kol1 kol2 kol_3
## 1      1    11   12
## 2      2    12   14
## 3      3    13   16
## 4      4    14   18
## 5      5    15   20
## 6      6    16   22
## 7      7    17   24
## 8      8    18   26
## 9      9    19   28
## 10     10   20   30
```

Funkcje zawarte w pakietach tidyverse (w tym `read_delim` i inne) często zamiast zwykłej ramki danych używają ulepszonej struktury zwanej tibble. W codziennym użytkowaniu nie ma między nimi wielu różnic. Tibble są bezpieczniejsze od zwykłych `data.frame`, ponieważ nigdy nie zmieniają typów danych w kolumnach. Może się jednak czasem zdarzyć że funkcje starszych pakietów nie będą akceptować tibble zamiast `data.frame` (można łatwo zmienić przy pomocy `as.data.frame`).

3.3.3 Matryca (`matrix`)

Matryca jest trochę podobna do ramki danych, ale może zawierać tylko jeden typ danych np. liczbowe. Może mieć więcej wymiarów niż dwa. Tworzymy funkcją `matrix`, zmiana istniejących danych na matrycę - `as.matrix`.

Indeksowanie z matrycach działa tak samo jak w ramkach danych - `[wiersz, kolumna]`.

```
matrix(0, 4, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
```

```
matrix(1:15, 3, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

```
(x <- matrix(1:9, 3,3))

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

x[1,3]

## [1] 7
```

3.3.4 Lista (list)

Lista to taki rozbudowany wektor ;) i najbardziej elastyczny typ danych w R. Każdy element listy może być innego rodzaju, mieć inną długość, można stworzyć listę, której elementami będzie np. ramka danych, wektor, wykres i nawet inna lista.

Listę tworzymy funkcją `list`, podobnie jak w wektorze każdy element może mieć swoją nazwę. Wiele funkcji jako swój wynik zwraca listę, więc często przydaje się wiedza jak dostać się do poszczególnych elementów.

Funkcją `str` możemy wyświetlić podsumowanie wszystkich elementów listy.

Do poszczególnych części można dostać się przez nazwy albo indeksowanie [] albo [[]]. Przy zastosowaniu nawiasów [] uzyskany element nadal będzie częścią listy.

```
lista <- list( ramka = data.frame(a=rnorm(10), b="test"),
               wektor = seq(1,16,by=3),
               tekst = "To jest lista" )

lista

## $ramka
##      a     b
## 1 -0.2023050 test
## 2 -0.1088964 test
## 3 -0.1916321 test
## 4 -1.3379562 test
## 5  0.1012737 test
## 6  1.5622984 test
## 7  0.1386039 test
## 8 -1.4607336 test
```

```

## 9   1.5207546 test
## 10 -0.3807385 test
##
## $wektor
## [1] 1 4 7 10 13 16
##
## $tekst
## [1] "To jest lista"

str(lista)

## List of 3
## $ ramka :'data.frame': 10 obs. of 2 variables:
##   ..$ a: num [1:10] -0.202 -0.109 -0.192 -1.338 0.101 ...
##   ..$ b: chr [1:10] "test" "test" "test" "test" ...
##   $ wektor: num [1:6] 1 4 7 10 13 16
##   $ tekst : chr "To jest lista"

lista$tekst

## [1] "To jest lista"

lista[3]

## $tekst
## [1] "To jest lista"

lista[[3]]

## [1] "To jest lista"

# pierwszy element wektora, będącego drugim elementem listy
lista[[2]][1]

## [1] 1

```

3.3.5 Typ liczbowy (numeric), znakowy (character), logiczny (logical)

- Typ liczbowy przechowuje liczby całkowite i rzeczywiste. Kropka dziesiętną jest kropka :). Na liczbach można łatwo prowadzić podstawowe działania matematyczne - dodawanie, odejmowanie, mnożenie itp. Takie same operacje możemy prowadzić na wektorach.

2*4

```
## [1] 8
```

```
A <- c(2,4,6)
B <- c(1,2,3)
```

```
A*B
```

```
## [1] 2 8 18
```

- Typ znakowy zawiera napisy umieszczone pomiędzy "" albo ''. Napisy można wyświetleć np. funkcją `cat`, możemy wymusić pisanie kolejnego elementu w nowej linii poprzez "\n". Sklejanie np. tekstu i liczb można wykonać funkcją `paste`.

Do pracy z typem znakowym można wykorzystać pakiet `stringr`.

```
" To jest napis "
```

```
## [1] " To jest napis "
```

```
cat("coś", "tam")
```

```
## coś tam
```

```
cat(" coś","\n","tam")
```

```
## coś
## tam
```

```
a <- 1
```

```
cat("Wynik to", a)
```

```
## Wynik to 1
```

```
paste("Wynik równa się", a)
```

```
## [1] "Wynik równa się 1"
```

- Typ logiczny przechowuje tylko wartości: TRUE (T) i FALSE (F) oraz NA (brak danych). Nazwy TRUE i FALSE są w R zastrzeżone, ale T i F już nie. Dlatego lepiej używać pełnych nazw, bo może się zdarzyć, że do T albo F zostanie przypisana jakaś zmienna.

```
wektor <- c(TRUE, FALSE, TRUE)
wektor
```

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

```
summary(wektor)
```

```
##      Mode   FALSE    TRUE
## logical     1       2
```

3.3.6 Typ czynnikowy (factor)

Służy do przechowywania wartości występujących w kilku kategoriach np. płeć, wykształcenie itp. Podczas wczytywania danych R z wykorzystaniem podstawowych funkcji, ale nie z pakietu `readr`, automatycznie zamieni kolumny zawierające tekst na typ czynnikowy, chyba że zaznaczymy opcję `stringAsFactors=FALSE`.

Typ czynnikowy jest przydatny podczas robienia wykresów, gdy chcemy pogrupować dane pod względem jakiejś kategorii. Również legendy (kolejność elementów) są tworzone w oparciu o poziomy czynnika. Może się zdarzyć, że kolejność wybrana przez R (często alfabetyczna) nie będzie odpowiednia. Można łatwo przestawić poziomy korzystając z funkcji `factor`. W tej samej funkcji argument `labels` pozwala na zmianę nazw kolejnych poziomów.

Jeżeli chcemy zmienić kolejność elementów na wykresie np. boxplotów, słupków najlepiej zrobić to zmieniając poziomy faktora w danych.

Poziomy factor można wyświetlić przy pomocy funkcji `levels`.

Do pracy z typem czynnikowym można wykorzystać pakiet `forcats` będący częścią `tidyverse`.

```
faktor <- factor(c("A", "B", "C", "A", "A", "C"))
```

```
faktor
```

```
## [1] A B C A A C
## Levels: A B C
```

```

str(faktor)

## Factor w/ 3 levels "A","B","C": 1 2 3 1 1 3

levels(faktor)

## [1] "A" "B" "C"

summary(faktor)

## A B C
## 3 1 2

# Zmiana poziomów

faktor2 <- factor(faktor, levels=c("B", "C", "A"), labels = c("BB", "CC", "AA"))
faktor2

## [1] AA BB CC AA AA CC
## Levels: BB CC AA

```

3.3.7 Funkcje

Większość operacji na danych w R wykonujemy za pomocą funkcji. Pakiety są to właściwie zbiory funkcji, często dotyczących jednego konkretnego zagadnienia.

Każda funkcja zawiera przynajmniej jeden argument, który jest niezbędny do jej działania. Nazwy wszystkich argumentów możemy sprawdzić korzystając z pomocy. Często nie jest konieczne podanie wartości wszystkich argumentów, gdyż mogą mieć ustalone wartości domyślne.

Jeżeli podajemy wartości argumentów możemy je wpisywać do funkcji kolejno (tak jak są wypisane w pomocy) albo korzystając z nazw. Jeżeli nie pamiętamy wszystkich nazw można sobie pomóc przy pomocy klawisza Tab :). Kolejne podawane argumenty należy rozdzielać przecinkami np. `funkcja(a=1, b=2, c="model")`. Do argumentu możemy też podać wektor wartości korzystając z `c` np. `funkcja(a=c(1,2,5))`

Np. funkcja licząca średnią to `mean`. Zawiera trzy argumenty:

- `x` - oznacza obiekt, z którego ma być policzona średnia
- `trim` (domyślnie 0) oznacza część obserwacji, która ma zostać przycięta z każdej strony `x`

- na.rm (domyślnie FALSE) - czy mają być brane pod uwagę wartości NA.

```
?mean
mean {base} R Documentation
Arithmetric Mean
Description
Generic function for the (trimmed) arithmetic mean.

Usage
mean(x, ...)

## Default S3 method: mean(x, trim = 0, na.rm = FALSE, ...)

wek <- c(1,5,9,2,7,3,5,9,2,4)

mean(wek)

## [1] 4.7

mean(x=wek)

## [1] 4.7

mean(x=wek, trim=0.2, na.rm=TRUE)

## [1] 4.333333
```

W R możliwe jest pisanie własnych funkcji. Nie jest to trudne, a pozwala na uniknięcie wielokrotnego powtarzanie tego samego kodu. Dużo łatwiej jest zmienić/poprawić jedną funkcję niż dwadzieścia razy wklejone te same 10 linijek kodu :)

Poniżej krótki przykład tworzenia funkcji.Więcej informacji można znaleźć np. w książce R for Data Science.

Załóżmy że chemy napisać funkcję która wczyta dane z pliku txt i doda do tego pliki nową kolumnę będącą sumą dwóch pierwszych.

Kod potrzebny żeby to wykonać:

```
data <- read.table('data/test_funkcja.txt')

data$nowa_kolumna <- data$x + data$y
```

A teraz funkcja która wykonuje te same czynności. Nazwa funkcji jest dowolna, chociaż lepiej jest nie nadpisywać już istniejących funkcji, a jej nazwa powinna opisywać to co funkcja będzie robić. W nawiasach () należy podać nazwy argumentów z ewentualnymi wartościami domyślnymi, a w nawiadach {} kod który ma zostać wykonany. Na końcu funkcji warto użyć return() - determinuje jaki wynik zwróci funkcja.

```
dodaj_kolumnę <- function(file){  
  data <- read.table(file)  
  data$nowa_kolumna <- data$x + data$y  
  return(data)  
}
```

Każdą funkcję po napisaniu należy sprawdzić

```
dodaj_kolumnę('data/test_funkcja.txt')
```

```
##      x  y nowa_kolumna  
## 1    1 11        12  
## 2    2 12        14  
## 3    3 13        16  
## 4    4 14        18  
## 5    5 15        20  
## 6    6 16        22  
## 7    7 17        24  
## 8    8 18        26  
## 9    9 19        28  
## 10 10 20        30
```


Chapter 4

Analiza i transformacja danych

W tym rozdziale zostaną przedstawione metody transformacji danych wykorzystujące tidyverse, a konkretnie pakiety dplyr i tidyr. Większość z przedstawionych operacji może być też przeprowadzona przy pomocy funkcji pakietu podstawowego albo pakietu data.table, który jest szczególnie dedykowany do bardzo dużych zbiorów danych.

Typowa praca z danymi w R zwykle obejmuje kilka podstawowych zadań: * zmiana formatu na uporządkowany - pakiet tidyr - `pivot_longer`, `pivot_wider` * sortowanie - `arrange`, `relocate` * filtrowanie - `filter`, `select`, `distinct`, `slice_*` * podział na grupy - `group_by` * obliczanie nowych zmiennych - `mutate`, `select` * łączenie tabel - `*_join`, `bind_rows`, `bind_cols`

4.1 Format uporządkowany (wąski)

Żeby wykorzystać pełnię możliwości pakietów dplyr i ggplot2 format danych powinien zostać zmieniony na uporządkowany (ang. tidy albo long). Brzmi to może skomplikowanie, ale sprowadza się do dwóch prostych zasad: 1. Każda zmienna znajduje się w osobnej kolumnie np. długość, waga, szczep, rok itp. 2. Każdy przypadek znajduje się w osobnym rzędzie np. komórka, osoba, białko itp.

Przykład danych nieuporządkowanych (szerokie) - każda kolumna to pomiar długości dla innego szczepu

szczep_1	szczep_2	szczep_3
1	10	2
3	15	4
2	11	5

Te same dane, ale uporządkowane (wąskie)

szczep	dlugosc
szczep_1	1
szczep_1	3
szczep_1	2
szczep_2	10
szczep_2	15
szczep_2	11
szczep_3	2
szczep_3	4
szczep_3	5

Pakiet tidyr zawiera dwie funkcje, które pozwalają na łatwy przejście pomiędzy danymi szerokimi a wąskimi: **pivot_wider**, **pivot_longer**.

Argumenty `pivot_longer()` * data - podstawowy - ramka danych która chcemy zmienić * `cols` - które kolumny będą zmieniane: mamy różne opcje: `everything()` - wszystkie kolumny, `c('nazwa_1', 'nazwa_2')` - podajemy nazwy kolumn po przecinku, `!nazwa_3` - po `!` podajemy nazwę kolumny która ma pozostać niezmieniona. Jeżeli kolumny do zmiany mają wspólny początek nazwy możemy użyć `starts_with('wspólna_część')` * `names_to` - nazwa kolumny w której znajdują się poprzednia nazwy kolumn * `values_to` - nazwa kolumny, w której znajdują się wartości

Warto zwrócić uwagę, że funkcje tidyr zawsze zwracają tibble zamiast zwykłej ramki danych.

```
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##      filter, lag
```

```

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

# zmiana wszystkich kolumn - najprostszy przypadek
mock_data <- data.frame(A = rnorm(100), B = rnorm(100), C = rnorm(100))
head(mock_data)

##           A         B         C
## 1 -0.19762266 1.2509268 0.9360672
## 2 -1.22342594 -0.1956236 1.0034208
## 3  0.05379449  0.5884557 0.8343494
## 4 -1.24187683  0.1496432 1.0233469
## 5 -0.21430272  0.5676162 0.1206934
## 6  0.74458424 -1.8064297 0.1115393

mock_data_long <- pivot_longer(mock_data, cols = everything(), names_to = 'key', values_to = 'val')
head(mock_data_long)

## # A tibble: 6 x 2
##   key      value
##   <chr>    <dbl>
## 1 A        -0.198
## 2 B         1.25
## 3 C         0.936
## 4 A        -1.22
## 5 B        -0.196
## 6 C         1.00

# i z powrotem
pivot_wider(mock_data_long, names_from = key, values_from = value) %>% unnest() %>% head()

## Warning: Values are not uniquely identified; output will contain list-cols.
## * Use `values_fn = list` to suppress this warning.
## * Use `values_fn = length` to identify where the duplicates arise
## * Use `values_fn = {summary_fun}` to summarise duplicates

## Warning: `cols` is now required when using unnest().
## Please use `cols = c(A, B, C)`

## # A tibble: 6 x 3
##       A         B         C
##   <dbl>    <dbl>    <dbl>
## 1 -0.198  1.250  0.936
## 2 -1.223 -0.196  1.003
## 3  0.054  0.588  0.834
## 4 -1.242  0.149  1.023
## 5 -0.214  0.568  0.121
## 6  0.745 -1.806  0.112

```

```

## 1 -0.198 1.25 0.936
## 2 -1.22 -0.196 1.00
## 3 0.0538 0.588 0.834
## 4 -1.24 0.150 1.02
## 5 -0.214 0.568 0.121
## 6 0.745 -1.81 0.112

# zmiana tylko części kolumn - argumencie cols można podać numery kolumn, nazwy albo
# użyć funkcji pomocniczych dplyr np. starts_with()
mock_data <- data.frame(A = rnorm(100), B = rnorm(100), C = rnorm(100), id = 1:100)
head(mock_data)

##           A         B         C id
## 1 -0.1770205 -0.25164542 0.5536363 1
## 2 -0.9440567 -0.54688925 -0.2698578 2
## 3 2.2879383 -0.09028388 0.1085704 3
## 4 -0.2760257 -0.66500382 0.0572416 4
## 5 0.6544091 0.23107289 -0.6537952 5
## 6 0.8398454 -2.11009313 0.8502048 6

mock_data_long <- pivot_longer(mock_data, cols = 1:3, names_to = 'key', values_to = 'value')
head(mock_data_long)

## # A tibble: 6 x 3
##       id key   value
##   <int> <chr> <dbl>
## 1     1 A     -0.177
## 2     1 B     -0.252
## 3     1 C      0.554
## 4     2 A     -0.944
## 5     2 B     -0.547
## 6     2 C     -0.270

# i z powrotem
pivot_wider(mock_data_long, names_from = key, values_from = value) %>% head()

## # A tibble: 6 x 4
##       id     A         B         C
##   <int> <dbl> <dbl> <dbl>
## 1     1 -0.177 -0.252  0.554
## 2     2 -0.944 -0.547 -0.270
## 3     3  2.29  -0.0903 0.109
## 4     4 -0.276 -0.665  0.0572
## 5     5  0.654  0.231  -0.654
## 6     6  0.840 -2.11    0.850

```

4.2 Pipes (potoki)

W R funkcje można łączyć w potoki przy pomocy operatora `%>%` (pakiet magrittr) albo `|>` (w podstawowym R od wersji 4.1.0).

Pipes można rozumieć jako zestaw instrukcji dla R, które zostaną wykonane po kolej. Kolejne funkcje połączone są `%>%`, co sprawia że wynik pierwszej funkcji jest przekazywany do następnej, zawsze jako pierwszy argument i tak dalej aż do końca potoku. Zaletą takiego podejścia jest brak konieczności tworzenia zmiennych pośrednich oraz większa czytelność kodu.

Na podstawie przykładu powyżej

```
# pipes

pivot_wider(mock_data_long, names_from = key, values_from = value) %>% unnest() %>% head(n = 3)

## Warning: `cols` is now required when using unnest().
## Please use `cols = c()``


## # A tibble: 3 x 4
##       id      A      B      C
##   <int>  <dbl>  <dbl>  <dbl>
## 1     1 -0.177 -0.252  0.554
## 2     2 -0.944 -0.547 -0.270
## 3     3  2.29   -0.0903 0.109

# zmienne pośrednie

zmienna_1 <- pivot_wider(mock_data_long, names_from = key, values_from = value)
zmienna_2 <- unnest(zmienna_1)

## Warning: `cols` is now required when using unnest().
## Please use `cols = c()``


head(zmienna_2, n = 3)

## # A tibble: 3 x 4
##       id      A      B      C
##   <int>  <dbl>  <dbl>  <dbl>
## 1     1 -0.177 -0.252  0.554
## 2     2 -0.944 -0.547 -0.270
## 3     3  2.29   -0.0903 0.109
```

```
# funkcja w funkcji - nieczytelny kod

head(unnest(pivot_wider(mock_data_long, names_from = key, values_from = value)), n = 3)

## Warning: `cols` is now required when using unnest().
## Please use `cols = c()`` 

## # A tibble: 3 x 4
##       id      A      B      C
##   <int>  <dbl>  <dbl>  <dbl>
## 1     1 -0.177 -0.252  0.554
## 2     2 -0.944 -0.547 -0.270
## 3     3  2.29  -0.0903 0.109
```

Wszystkie funkcje zawarte w pakietach tidyverse są dostosowane do pracy w potokach - pierwszym argumentem zawsze jest ramka danych. Może się jednak zdarzyć przy korzystaniu np. ze starszych pakietów, że ramka danych nie znajdzie się w pierwszym argumencie funkcji. Wtedy można użyć notacji z . albo zastosować tradycyjny zapis.

```
# wykorzystanie . do określenia gdzie ma zostać wpisany wynik funkcji znajdujący się w
# lm dopasowuje model liniowy do danych, więcej na ten temat w rozdziale statystyka
mock_data_long %>% lm(formula = value ~ key, data = .)
```

```
##
## Call:
## lm(formula = value ~ key, data = .)
##
## Coefficients:
## (Intercept)      keyB      keyC
##     0.00646    -0.07860    -0.06325
```

Końcowy wynik potoku można przypisać do nowej zmiennej na początku korzystając ze strzałki <- albo na końcu ->

```
mock_data_long %>% lm(formula = value ~ key, data = .) -> wynik
summary(wynik)
```

```
##
## Call:
## lm(formula = value ~ key, data = .)
##
## Residuals:
```

```

##      Min      1Q Median      3Q      Max
## -3.8701 -0.6131  0.0058  0.6734  2.6925
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.00646   0.09348   0.069   0.945
## keyB        -0.07860   0.13221  -0.595   0.553
## keyC        -0.06325   0.13221  -0.478   0.633
##
## Residual standard error: 0.9348 on 297 degrees of freedom
## Multiple R-squared:  0.001336, Adjusted R-squared:  -0.005389
## F-statistic: 0.1986 on 2 and 297 DF, p-value: 0.82

```

4.3 Sortowanie

Do sortowania danych można użyć funkcji `arrange()`. Jako argumenty wystarczy podać nazwy kolumn według których dane mają być posortowane. Domyslnie sortuje rosnąco. Jeżeli dana kolumna ma być posortowana malejąco należy dodać do jej nazwy `desc()`.

Natomiast do zmiany położenia kolumn w danych służy funkcja `select` lub `relocate`. `select` pozwala też na usunięcie kolumn, jako argumenty trzeba po prostu podać nazwy kolumn w takiej kolejności jak chcemy żeby znalazły się wynikach. W przypadku `relocate` kolumny nigdy nie zostaną usunięte z danych, można kontrolować pozycję przenoszonej kolumny przy pomocy argumentów `.after` i `.before`.

```

library(palmerpenguins)

# sortowanie po jednej zmiennej
penguins %>% arrange(body_mass_g)

## # A tibble: 344 x 8
##   species     island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>     <fct>       <dbl>        <dbl>          <int>        <int>
## 1 Chinstrap Dream        46.9         16.6          192        2700
## 2 Adelie    Biscoe       36.5         16.6          181        2850
## 3 Adelie    Biscoe       36.4         17.1          184        2850
## 4 Adelie    Biscoe       34.5         18.1          187        2900
## 5 Adelie    Dream         33.1         16.1          178        2900
## 6 Adelie    Torgersen    38.6          17            188        2900
## 7 Chinstrap Dream        43.2         16.6          187        2900
## 8 Adelie    Biscoe       37.9         18.6          193        2925
## 9 Adelie    Dream         37.5         18.9          179        2975

```

```

## 10 Adelie      Dream          37       16.9        185      3000
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>

# wiele zmiennych
penguins %>% arrange(species, desc(island), bill_length_mm)

## # A tibble: 344 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>           <dbl>         <dbl>            <dbl>        <int>
## 1 Adelie  Torgersen     33.5          19             190        3600
## 2 Adelie  Torgersen     34.1          18.1            193        3475
## 3 Adelie  Torgersen     34.4          18.4            184        3325
## 4 Adelie  Torgersen     34.6          21.1            198        4400
## 5 Adelie  Torgersen     34.6          17.2            189        3200
## 6 Adelie  Torgersen     35.1          19.4            193        4200
## 7 Adelie  Torgersen     35.2          15.9            186        3050
## 8 Adelie  Torgersen     35.5          17.5            190        3700
## 9 Adelie  Torgersen     35.7          17               189        3350
## 10 Adelie Torgersen     35.9          16.6            190        3050
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>

# przykład relocate
penguins %>% relocate(year, .after = island)

## # A tibble: 344 x 8
##   species island    year bill_length_mm bill_depth_mm flipper_length_mm
##   <fct>   <fct>   <int>         <dbl>         <dbl>            <int>
## 1 Adelie  Torgersen 2007      39.1          18.7          181
## 2 Adelie  Torgersen 2007      39.5          17.4          186
## 3 Adelie  Torgersen 2007      40.3          18             195
## 4 Adelie  Torgersen 2007      NA             NA             NA
## 5 Adelie  Torgersen 2007      36.7          19.3          193
## 6 Adelie  Torgersen 2007      39.3          20.6          190
## 7 Adelie  Torgersen 2007      38.9          17.8          181
## 8 Adelie  Torgersen 2007      39.2          19.6          195
## 9 Adelie  Torgersen 2007      34.1          18.1          193
## 10 Adelie Torgersen 2007      42             20.2          190
## # ... with 334 more rows, and 2 more variables: body_mass_g <int>, sex <fct>

# przenoszenie kilku zmiennych naraz z wykorzystaniem starts_with
penguins %>% relocate(starts_with('bill'), .after = body_mass_g)

## # A tibble: 344 x 8
##   species island    flipper_length_mm body_mass_g bill_length_mm bill_depth_mm
##   <fct>   <fct>            <dbl>        <dbl>         <dbl>         <dbl>
## 1 Adelie  Torgersen      33.5        3600         33.5        19.0
## 2 Adelie  Torgersen      34.1        3475         34.1        18.1
## 3 Adelie  Torgersen      34.4        3325         34.4        18.4
## 4 Adelie  Torgersen      34.6        4400         34.6        21.1
## 5 Adelie  Torgersen      34.6        3200         34.6        17.2
## 6 Adelie  Torgersen      35.1        4200         35.1        19.4
## 7 Adelie  Torgersen      35.2        3050         35.2        15.9
## 8 Adelie  Torgersen      35.5        3700         35.5        17.5
## 9 Adelie  Torgersen      35.7        3350         35.7        17.0
## 10 Adelie Torgersen      35.9        3050         35.9        16.6
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>
```

```

## # A tibble: 344 x 8
##   species island   flipper_length_mm body_mass_g bill_length_mm bill_depth_mm
##   <fct>   <fct>           <int>      <int>        <dbl>        <dbl>
## 1 Adelie  Torgersen       181        3750       39.1        18.7
## 2 Adelie  Torgersen       186        3800       39.5        17.4
## 3 Adelie  Torgersen       195        3250       40.3        18
## 4 Adelie  Torgersen       NA         NA          NA          NA
## 5 Adelie  Torgersen       193        3450       36.7        19.3
## 6 Adelie  Torgersen       190        3650       39.3        20.6
## 7 Adelie  Torgersen       181        3625       38.9        17.8
## 8 Adelie  Torgersen       195        4675       39.2        19.6
## 9 Adelie  Torgersen       193        3475       34.1        18.1
## 10 Adelie Torgersen       190        4250        42        20.2
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>

# to samo ale z select - więcej pisania ;
penguins %>% select(species, island, flipper_length_mm, body_mass_g, bill_length_mm, bill_depth_mm)

## # A tibble: 344 x 8
##   species island   flipper_length_mm body_mass_g bill_length_mm bill_depth_mm
##   <fct>   <fct>           <int>      <int>        <dbl>        <dbl>
## 1 Adelie  Torgersen       181        3750       39.1        18.7
## 2 Adelie  Torgersen       186        3800       39.5        17.4
## 3 Adelie  Torgersen       195        3250       40.3        18
## 4 Adelie  Torgersen       NA         NA          NA          NA
## 5 Adelie  Torgersen       193        3450       36.7        19.3
## 6 Adelie  Torgersen       190        3650       39.3        20.6
## 7 Adelie  Torgersen       181        3625       38.9        17.8
## 8 Adelie  Torgersen       195        4675       39.2        19.6
## 9 Adelie  Torgersen       193        3475       34.1        18.1
## 10 Adelie Torgersen       190        4250        42        20.2
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>

# albo tak
penguins %>% select(starts_with('bill'), everything())

## # A tibble: 344 x 8
##   bill_length_mm bill_depth_mm species island   flipper_length_mm body_mass_g
##   <dbl>        <dbl>   <fct>   <fct>           <int>      <int>
## 1 39.1         18.7  Adelie  Torgersen       181        3750
## 2 39.5         17.4  Adelie  Torgersen       186        3800
## 3 40.3         18    Adelie  Torgersen       195        3250
## 4 NA            NA    Adelie  Torgersen       NA          NA
## 5 36.7         19.3  Adelie  Torgersen       193        3450
## 6 39.3         20.6  Adelie  Torgersen       190        3650
## 7 38.9         17.8  Adelie  Torgersen       181        3625

```

```

## 8          39.2          19.6 Adelie Torgersen    195      4675
## 9          34.1          18.1 Adelie Torgersen    193      3475
## 10         42            20.2 Adelie Torgersen    190      4250
## # ... with 334 more rows, and 2 more variables: sex <fct>, year <int>

```

4.4 Filtrowanie

Do wybrania z tabeli tylko interesującego nas zbioru danych możemy użyć funkcji `filter`, `distinct`, `select` i `slice_*`.

`filter` pozwala na wybranie wierszy spełniających konkretne warunki. Do wpisywania warunków konieczna jest znajomość wyrażeń logicznych w R: * == równy * != nie równy * < mniejszy * <= mniejszy równy * > większy * >= większy równy * & i * | lub * %in% znajduje się w zbiorze danych (wektor)

`select` można wybrać kolejne kolumny wpisując ich nazwy albo korzystając z funkcji pomocniczych np. `starts_with`.

`distinct` pozwala na pozostawienie jedynie unikalnych kombinacji zmiennych. Jako argumenty należy podać jedynie nazwy kolumn. Domyslnie usunie wszystkie pozostałe kolumny, ale można je zostawić argumentem `.keep`. Przydatna funkcja np. do usuwania duplikatów z danych.

Natomiast do usunięcia z danych wszystkich wierszy w których pojawia się wartość NA można użyć funkcji `drop_na`.

Do wybrania jedynie fragmentu danych można też użyć funkcji `slice`. W najprostszej postaci można wybrać dane po numerach wierszy podobnie jak w indeksowaniu ramki danych przy pomocy `[]`. Dodatkowe funkcje umożliwiają większą kontrolę: `slice_head` albo `slice_tail` wybierają z danych odpowiednio pierwsze i ostatnie wiersze, liczba kontrolowana argumentem `n`. Natomiast `slice_sample` wybierze z danych losowe wiersze, liczba określona argumentem `n`, działa analogicznie do funkcji `sample`.

```
# filter - tylko Adelie z wyspy Torgersen
penguins %>% filter(species == 'Adelie', island == 'Torgersen')
```

```

## # A tibble: 52 x 8
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>           <dbl>        <dbl>          <dbl>        <dbl>
## 1 Adelie  Torgersen       39.1         18.7         181        3750
## 2 Adelie  Torgersen       39.5         17.4         186        3800
## 3 Adelie  Torgersen       40.3         18           195        3250
## 4 Adelie  Torgersen       NA            NA           NA          NA
## 5 Adelie  Torgersen       36.7         19.3         193        3450
## 6 Adelie  Torgersen       39.3         20.6         190        3650

```

```
## 7 Adelie Torgersen      38.9      17.8      181      3625
## 8 Adelie Torgersen      39.2      19.6      195      4675
## 9 Adelie Torgersen      34.1      18.1      193      3475
## 10 Adelie Torgersen     42        20.2      190      4250
## # ... with 42 more rows, and 2 more variables: sex <fct>, year <int>
```

```
# filter - tylko pingwiny cięzsze niż 4000 i płeć male
penguins %>% filter(body_mass_g >= 4000, sex == 'male')
```

```
## # A tibble: 114 x 8
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>       <dbl>        <dbl>          <dbl>        <int>
## 1 Adelie  Torgersen    39.2        19.6          195        4675
## 2 Adelie  Torgersen    34.6        21.1          198        4400
## 3 Adelie  Torgersen    42.5        20.7          197        4500
## 4 Adelie  Torgersen    46          21.5          194        4200
## 5 Adelie  Dream        39.2        21.1          196        4150
## 6 Adelie  Dream        39.8        19.1          184        4650
## 7 Adelie  Dream        44.1        19.7          196        4400
## 8 Adelie  Dream        39.6        18.8          190        4600
## 9 Adelie  Dream        42.3        21.2          191        4150
## 10 Adelie Biscoe       40.1        18.9          188        4300
## # ... with 104 more rows, and 2 more variables: sex <fct>, year <int>
```

```
# distinct - obecne wanych kombinacja species, island, sex
penguins %>% distinct(species, island, sex)
```

```
## # A tibble: 13 x 3
##   species   island   sex
##   <fct>     <fct>   <fct>
## 1 Adelie    Torgersen male
## 2 Adelie    Torgersen female
## 3 Adelie    Torgersen <NA>
## 4 Adelie    Biscoe   female
## 5 Adelie    Biscoe   male
## 6 Adelie    Dream    female
## 7 Adelie    Dream    male
## 8 Adelie    Dream    <NA>
## 9 Gentoo   Biscoe   female
## 10 Gentoo  Biscoe   male
## 11 Gentoo  Biscoe   <NA>
## 12 Chinstrap Dream   female
## 13 Chinstrap Dream   male
```

```
# usuwanie NA i potem distinct
penguins %>% drop_na() %>% distinct(species, island, sex)
```

```
## # A tibble: 10 x 3
##   species   island   sex
##   <fct>     <fct>    <fct>
## 1 Adelie    Torgersen male
## 2 Adelie    Torgersen female
## 3 Adelie    Biscoe    female
## 4 Adelie    Biscoe    male
## 5 Adelie    Dream     female
## 6 Adelie    Dream     male
## 7 Gentoo    Biscoe    female
## 8 Gentoo    Biscoe    male
## 9 Chinstrap Dream     female
## 10 Chinstrap Dream    male
```

do wyświetlenia wszystkich możliwych kombinacji zmiennych można użyć funkcji expand

```
penguins %>% drop_na() %>% distinct(species, island, sex) %>% expand(species, island)
```

```
## # A tibble: 9 x 2
##   species   island
##   <fct>     <fct>
## 1 Adelie    Biscoe
## 2 Adelie    Dream
## 3 Adelie    Torgersen
## 4 Chinstrap Biscoe
## 5 Chinstrap Dream
## 6 Chinstrap Torgersen
## 7 Gentoo    Biscoe
## 8 Gentoo    Dream
## 9 Gentoo    Torgersen
```

możliwe kombinacje + uzupełni NA w innych kolumnach tam gdzie brakuje danych

```
penguins %>% drop_na() %>% distinct(species, island, sex) %>% complete(species, island)
```

```
## # A tibble: 14 x 3
##   species   island   sex
##   <fct>     <fct>    <fct>
## 1 Adelie    Biscoe    female
## 2 Adelie    Biscoe    male
## 3 Adelie    Dream     female
## 4 Adelie    Dream     male
```

```

## 5 Adelie    Torgersen male
## 6 Adelie    Torgersen female
## 7 Chinstrap Biscoe    <NA>
## 8 Chinstrap Dream   female
## 9 Chinstrap Dream   male
## 10 Chinstrap Torgersen <NA>
## 11 Gentoo   Biscoe   female
## 12 Gentoo   Biscoe   male
## 13 Gentoo   Dream    <NA>
## 14 Gentoo   Torgersen <NA>

# wybrane wiersze od 3 do 8
penguins %>% slice(3:8)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##   <fct>   <fct>     <dbl>        <dbl>          <dbl>      <int> <fct>
## 1 Adelie  Torge~     40.3         18            195       3250  fema~
## 2 Adelie  Torge~     NA           NA             NA        NA <NA>
## 3 Adelie  Torge~     36.7         19.3          193       3450  fema~
## 4 Adelie  Torge~     39.3         20.6          190       3650  male
## 5 Adelie  Torge~     38.9         17.8          181       3625  fema~
## 6 Adelie  Torge~     39.2         19.6          195       4675  male
## # ... with 1 more variable: year <int>

# pierwsze 10 wierszy
penguins %>% slice_head(n = 10)

## # A tibble: 10 x 8
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>     <dbl>        <dbl>          <dbl>      <int>
## 1 Adelie  Torgersen  39.1         18.7          181       3750
## 2 Adelie  Torgersen  39.5         17.4          186       3800
## 3 Adelie  Torgersen  40.3         18            195       3250
## 4 Adelie  Torgersen  NA           NA             NA        NA
## 5 Adelie  Torgersen  36.7         19.3          193       3450
## 6 Adelie  Torgersen  39.3         20.6          190       3650
## 7 Adelie  Torgersen  38.9         17.8          181       3625
## 8 Adelie  Torgersen  39.2         19.6          195       4675
## 9 Adelie  Torgersen  34.1         18.1          193       3475
## 10 Adelie Torgersen  42           20.2          190       4250
## # ... with 2 more variables: sex <fct>, year <int>

```

```
# 5 losowych wierszy
penguins %>% slice_sample(n = 5)

## # A tibble: 5 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##   <fct>    <fct>        <dbl>          <dbl>            <int>      <int> <fct>
## 1 Chinstrap Dream       45.7           17              195      3650 female
## 2 Gentoo   Biscoe      48.4           14.6             213      5850 male
## 3 Adelie   Dream       37.8           18.1             193      3750 male
## 4 Adelie   Torgo       39               17.1             191      3050 female
## 5 Chinstrap Dream       51.5           18.7             187      3250 male
## # ... with 1 more variable: year <int>
```

4.5 Grupowanie i obliczanie nowych zmiennych

Jedną z najbardziej przydatnych funkcji pakietu dplyr jest **group_by**, która pozwala na wydzielenie grup w danych przed wykonaniem następnych funkcji. Argumentami **group_by** są kolejne nazwy kolumn rozdzielone przecinkami. **group_by** w żaden sposób nie zmienia danych w tabeli, jej efektem jest tylko dodanie do tabeli informacji o tym, w której kolumnie znajduje się zmienna. Grupy z danych można usunąć przy pomocy funkcji **ungroup** albo nadpisać nowym **group_by**.

Dodanie grup do danych sprawia, że wszystkie kolejne funkcje będą wykonywane na fragmentach oryginalnej tabeli. Dzięki temu można np. obliczyć średnią dla każdej grupy, odfiltrować z każdej grupy tylko wartości maksymalne, wyświetlić tylko 3 wiersze z każdej grupy, dopasować model do każdej grupy osobno itd.

group_by można łączyć z dwiema funkcjami, które służą do dodawania nowych zmiennych do tabeli - **summarise** i **mutate**, ale przydaje się również przy filtryowaniu albo sortowaniu danych.

4.5.1 **mutate**

mutate pozwala na dodanie nowych kolumn na końcu ramki danych, ale nie usuwa poprzednich kolumn. Jako argumenty w **mutate** należy podać nazwę nowej kolumny = sposób policzenia nowej kolumny.

Szczególnie przydatna jest funkcja **ifelse**, która pozwala na przygotowanie nowej zmiennej na podstawie warunku logicznego. Zawsze posiada ona trzy argumenty. Pierwszy to warunek logiczny np. kolumna == 5, drugi to wartość, która ma zostać podana w wypadku spełnienia warunku, trzeci to wartość, która ma zostać podana gdy warunek nie będzie spełniony. W wypadku gdy potrzebne jest więcej niż dwie opcje można użyć funkcji **case_when**

```
# operacje matematyczne na kolumnach
penguins %>% group_by(species, sex) %>% mutate(st_bill = bill_length_mm/bill_depth_mm)

## # A tibble: 344 x 9
## # Groups:   species, sex [8]
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>        <dbl>        <dbl>        <dbl>        <dbl>
## 1 Adelie  Torgersen     39.1       18.7       181      3750
## 2 Adelie  Torgersen     39.5       17.4       186      3800
## 3 Adelie  Torgersen     40.3        18        195      3250
## 4 Adelie  Torgersen      NA         NA         NA        NA
## 5 Adelie  Torgersen     36.7       19.3       193      3450
## 6 Adelie  Torgersen     39.3       20.6       190      3650
## 7 Adelie  Torgersen     38.9       17.8       181      3625
## 8 Adelie  Torgersen     39.2       19.6       195      4675
## 9 Adelie  Torgersen     34.1       18.1       193      3475
## 10 Adelie Torgersen      42        20.2       190      4250
## # ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
## #   st_bill <dbl>

# wartość maksymalna i obliczanie procentów
penguins %>% group_by(species, sex) %>% mutate(max_bill_depth = max(bill_depth_mm),
                                                 procent = bill_depth_mm/max_bill_depth) %>%
  relocate(procent, max_bill_depth, .after = island)

## # A tibble: 344 x 10
## # Groups:   species, sex [8]
##   species island   procent max_bill_depth bill_length_mm bill_depth_mm
##   <fct>   <fct>    <dbl>        <dbl>        <dbl>        <dbl>
## 1 Adelie  Torgersen  0.870       21.5       39.1       18.7
## 2 Adelie  Torgersen  0.841       20.7       39.5       17.4
## 3 Adelie  Torgersen  0.870       20.7       40.3        18
## 4 Adelie  Torgersen  NA          NA          NA          NA
## 5 Adelie  Torgersen  0.932       20.7       36.7       19.3
## 6 Adelie  Torgersen  0.958       21.5       39.3       20.6
## 7 Adelie  Torgersen  0.860       20.7       38.9       17.8
## 8 Adelie  Torgersen  0.912       21.5       39.2       19.6
## 9 Adelie  Torgersen  NA          NA          34.1       18.1
## 10 Adelie Torgersen  NA          NA          42        20.2
## # ... with 334 more rows, and 4 more variables: flipper_length_mm <int>,
## #   body_mass_g <int>, sex <fct>, year <int>
```

```
# zastosowanie funkcji ifelse
penguins %>% group_by(species, sex) %>% mutate(czy_jest_plec = ifelse(!is.na(sex), 'ta

## # A tibble: 344 x 9
## # Groups:   species, sex [8]
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>     <dbl>        <dbl>        <int>       <int>
## 1 Adelie  Torgersen      39.1        18.7        181       3750
## 2 Adelie  Torgersen      39.5        17.4        186       3800
## 3 Adelie  Torgersen      40.3         18          195       3250
## 4 Adelie  Torgersen       NA          NA          NA          NA
## 5 Adelie  Torgersen      36.7        19.3        193       3450
## 6 Adelie  Torgersen      39.3        20.6        190       3650
## 7 Adelie  Torgersen      38.9        17.8        181       3625
## 8 Adelie  Torgersen      39.2        19.6        195       4675
## 9 Adelie  Torgersen      34.1        18.1        193       3475
## 10 Adelie  Torgersen      42          20.2        190       4250
## # ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
## #   czy_jest_plec <chr>

# zastosowanie case_when
penguins %>% group_by(species, sex) %>% mutate(wielkosc = case_when(body_mass_g <= 3000
                                                               body_mass_g > 3000
                                                               body_mass_g >= 4000

## # A tibble: 344 x 9
## # Groups:   species, sex [8]
##   species island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>     <dbl>        <dbl>        <int>       <int>
## 1 Adelie  Torgersen      39.1        18.7        181       3750
## 2 Adelie  Torgersen      39.5        17.4        186       3800
## 3 Adelie  Torgersen      40.3         18          195       3250
## 4 Adelie  Torgersen       NA          NA          NA          NA
## 5 Adelie  Torgersen      36.7        19.3        193       3450
## 6 Adelie  Torgersen      39.3        20.6        190       3650
## 7 Adelie  Torgersen      38.9        17.8        181       3625
## 8 Adelie  Torgersen      39.2        19.6        195       4675
## 9 Adelie  Torgersen      34.1        18.1        193       3475
## 10 Adelie  Torgersen      42          20.2        190       4250
## # ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
## #   wielkosc <chr>
```

```
# normalizacja danych przy pomocy scale
penguins %>% group_by(species, sex) %>% mutate(nor_body_mass = scale(body_mass_g))

## # A tibble: 344 x 9
## # Groups:   species, sex [8]
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>        <dbl>        <dbl>        <int>        <int>
## 1 Adelie  Torgersen     39.1       18.7        181       3750
## 2 Adelie  Torgersen     39.5       17.4        186       3800
## 3 Adelie  Torgersen     40.3        18         195       3250
## 4 Adelie  Torgersen      NA         NA          NA        NA
## 5 Adelie  Torgersen     36.7       19.3        193       3450
## 6 Adelie  Torgersen     39.3       20.6        190       3650
## 7 Adelie  Torgersen     38.9       17.8        181       3625
## 8 Adelie  Torgersen     39.2       19.6        195       4675
## 9 Adelie  Torgersen     34.1       18.1        193       3475
## 10 Adelie Torgersen      42         20.2        190       4250
## # ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
## #   nor_body_mass <dbl[,1]>

# Średnia ruchoma - konieczny pakiet zoo
penguins %>% group_by(species, island, sex) %>% arrange(species, island, sex) %>%
  mutate(mean = zoo::rollmean(body_mass_g,
                               k = 3, # szerokość okna do średniej ruchomej
                               fill = NA)) # czym uzupełnić brakujące wartości

## # A tibble: 344 x 9
## # Groups:   species, island, sex [13]
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>        <dbl>        <dbl>        <int>        <int>
## 1 Adelie  Biscoe       37.8       18.3        174       3400
## 2 Adelie  Biscoe       35.9       19.2        189       3800
## 3 Adelie  Biscoe       35.3       18.9        187       3800
## 4 Adelie  Biscoe       40.5       17.9        187       3200
## 5 Adelie  Biscoe       37.9       18.6        172       3150
## 6 Adelie  Biscoe       39.6       17.7        186       3500
## 7 Adelie  Biscoe       35         17.9        190       3450
## 8 Adelie  Biscoe       34.5       18.1        187       2900
## 9 Adelie  Biscoe       39         17.5        186       3550
## 10 Adelie Biscoe       36.5       16.6        181       2850
## # ... with 334 more rows, and 3 more variables: sex <fct>, year <int>,
## #   mean <dbl>
```

4.5.2 summarise

`summarise` tworzy nową tabelę, która będzie zawierała jedynie zmienne grupujące i nowo utworzone zmienne. Jeżeli dane nie są podzielone na żadne grupy to wynikiem `summarise` będzie tabela posiadająca tylko jeden wiersz.

Jeżeli potrzebne jest zastosowanie tej samej funkcji dla większej liczby kolumn np. `mean` można wykorzystać funkcję `across`.

Zliczenie elementów można przeprowadzić wykorzystując funkcje pomocniczą `n` albo `count`

```
# obliczanie średniej
penguins %>% group_by(species, sex) %>% filter(!is.na(sex)) %>%
  summarise(mean_mass = mean(body_mass_g))
```

```
## `summarise()` has grouped output by 'species'. You can override using the `groups`
```

```
## # A tibble: 6 x 3
## # Groups:   species [3]
##   species   sex   mean_mass
##   <fct>     <fct>     <dbl>
## 1 Adelie    female    3369.
## 2 Adelie    male     4043.
## 3 Chinstrap female    3527.
## 4 Chinstrap male     3939.
## 5 Gentoo    female    4680.
## 6 Gentoo    male     5485.
```

```
# średnia dla wszystkich zmiennych liczbowych
penguins %>% group_by(species, sex) %>% filter(!is.na(sex)) %>%
  summarise(across(.cols = where(is.numeric), # wybieramy tylko kolumny spełniające warunki
                  .fns = c(srednia = mean, odch = sd))) # lista funkcji - można każdą
```

```
## `summarise()` has grouped output by 'species'. You can override using the `groups`
```

```
## # A tibble: 6 x 12
## # Groups:   species [3]
##   species   sex   bill_length_mm_srednia bill_length_mm_odch bill_depth_mm_srednia
##   <fct>     <fct>     <dbl>           <dbl>           <dbl>
## 1 Adelie    female    37.3            2.03            17.6
## 2 Adelie    male     40.4            2.28            19.1
## 3 Chinstrap female    46.6            3.11            17.6
## 4 Chinstrap male     51.1            1.56            19.3
## 5 Gentoo    female    45.6            2.05            14.2
```

```

## 6 Gentoo male 49.5 2.72 15.7
## # ... with 7 more variables: bill_depth_mm_odch <dbl>,
## #   flipper_length_mm_srednia <dbl>, flipper_length_mm_odch <dbl>,
## #   body_mass_g_srednia <dbl>, body_mass_g_odch <dbl>, year_srednia <dbl>,
## #   year_odch <dbl>

# zliczanie z n()
penguins %>% group_by(species, sex) %>% filter(!is.na(sex)) %>%
  summarise(n = n())

## `summarise()` has grouped output by 'species'. You can override using the `groups` argument.

## # A tibble: 6 x 3
## # Groups:   species [3]
##   species   sex     n
##   <fct>     <fct> <int>
## 1 Adelie   female  73
## 2 Adelie   male    73
## 3 Chinstrap female  34
## 4 Chinstrap male    34
## 5 Gentoo   female  58
## 6 Gentoo   male    61

# zliczanie z count() - można pominąć group_by
penguins %>% filter(!is.na(sex)) %>%
  count(species, sex)

## # A tibble: 6 x 3
##   species   sex     n
##   <fct>     <fct> <int>
## 1 Adelie   female  73
## 2 Adelie   male    73
## 3 Chinstrap female  34
## 4 Chinstrap male    34
## 5 Gentoo   female  58
## 6 Gentoo   male    61

# połączenie z mutate
penguins %>% filter(!is.na(sex)) %>%
  count(species, sex) %>%
  group_by(species) %>%
  mutate(procent = n/sum(n))

```

```
## # A tibble: 6 x 4
## # Groups:   species [3]
##   species   sex     n procent
##   <fct>     <fct>   <int>    <dbl>
## 1 Adelie   female    73    0.5
## 2 Adelie   male      73    0.5
## 3 Chinstrap female    34    0.5
## 4 Chinstrap male      34    0.5
## 5 Gentoo   female    58    0.487
## 6 Gentoo   male      61    0.513
```

4.5.3 inne

```
# group_by + slice_head + arrange - dwa pierwsze wiersze z każdej grupy, posortowane po
penguins %>% group_by(species) %>% arrange(desc(body_mass_g)) %>% slice_head(n = 2)
```

```
## # A tibble: 6 x 8
## # Groups:   species [3]
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##   <fct>     <fct>       <dbl>        <dbl>          <dbl>      <dbl> <fct>
## 1 Adelie   Biscoe      43.2         19            197      4775 male
## 2 Adelie   Biscoe      41           20            203      4725 male
## 3 Chinstrap Dream       52           20.7          210      4800 male
## 4 Chinstrap Dream       52.8          20            205      4550 male
## 5 Gentoo   Biscoe      49.2          15.2          221      6300 male
## 6 Gentoo   Biscoe      59.6          17            230      6050 male
## # ... with 1 more variable: year <int>
```

Szczególnym przypadkiem funkcji tworzących nowe zmienne są `separate` i `unite` z pakietu `tidyR`. Ich działanie jest przeciwnostawne i pozwalają odpowiednio na złączenie i rozłączenie zmiennych przy pomocy separatora np. `_` albo `.`

```
# unite - domyślnie usuwa wejściowe kolumny, ale można je zostawić przy pomocy argumentu
penguins %>% unite(col = nowa_kolumn, species, sex, sep = '_')
```

```
## # A tibble: 344 x 7
##   nowa_kolumn   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <chr>         <fct>       <dbl>        <dbl>          <dbl>      <dbl>
## 1 Adelie_male   Torgersen    39.1         18.7          181      3750
## 2 Adelie_female Torgersen    39.5         17.4          186      3800
## 3 Adelie_female Torgersen    40.3          18            195      3250
```

```

## 4 Adelie_NA    Torgersen      NA      NA      NA
## 5 Adelie_female Torgersen    36.7    19.3    193    3450
## 6 Adelie_male   Torgersen    39.3    20.6    190    3650
## 7 Adelie_female Torgersen    38.9    17.8    181    3625
## 8 Adelie_male   Torgersen    39.2    19.6    195    4675
## 9 Adelie_NA    Torgersen    34.1    18.1    193    3475
## 10 Adelie_NA   Torgersen     42      20.2    190    4250
## # ... with 334 more rows, and 1 more variable: year <int>

# separate - również można wejściową kolumnę zostawić przy pomocy argumentu remove
penguins %>% unite(col = nowa_kolumna, species, sex, sep = '_') %>%
  separate(col = nowa_kolumna, into = c('gatunek', 'plec'), sep = '_')

## # A tibble: 344 x 8
##   gatunek plec   island bill_length_mm bill_depth_mm flipper_length_mm
##   <chr>   <chr>   <fct>        <dbl>        <dbl>            <int>
## 1 Adelie male   Torgersen    39.1       18.7            181
## 2 Adelie female Torgersen    39.5       17.4            186
## 3 Adelie female Torgersen    40.3        18             195
## 4 Adelie NA     Torgersen     NA          NA             NA
## 5 Adelie female Torgersen    36.7       19.3            193
## 6 Adelie male   Torgersen    39.3       20.6            190
## 7 Adelie female Torgersen    38.9       17.8            181
## 8 Adelie male   Torgersen    39.2       19.6            195
## 9 Adelie NA     Torgersen    34.1       18.1            193
## 10 Adelie NA    Torgersen     42        20.2            190
## # ... with 334 more rows, and 2 more variables: body_mass_g <int>, year <int>

```

4.6 Łączenie tabel

Do łączenia tabel w R można wykorzystać dwa typy funkcji: `bind_*` i `*_join`.

4.6.1 `bind_*`

Funkcje `bind_cols` i `bind_rows` służą do sklejania tabel ze sobą odpowiednio według kolumn (jedna obok drugiej) i wierszy (jedna pod drugą). Są szczególnie przydatne w sytuacjach gdy dane wejściowe do analizy znajdują się w kilku różnych plikach, których struktura jest taka sama (układ kolumn itp.).

Do funkcji `bind_rows`, można dodać argument `.id`, który doda nową kolumnę do danych, zawierającą nazwy pierwotnych kolumn.

```
# przykład bind_rows
data_1 <- data.frame(x = 1:3, y = letters[1:3], z = c('jeden', 'dwa', 'trzy'))
data_2 <- data.frame(x = 4:6, y = letters[10:12], z = c('jeden', 'dwa', 'trzy'))

bind_rows(data_1, data_2)

##   x y     z
## 1 1 a jeden
## 2 2 b    dwa
## 3 3 c   trzy
## 4 4 j jeden
## 5 5 k    dwa
## 6 6 l   trzy

# z ustawionym .id
bind_rows(x = data_1, y = data_2, .id = 'id') # x = itd. to podane nazwy tabel do kolumny .id

##   id x y     z
## 1  1 a jeden
## 2  2 b    dwa
## 3  3 c   trzy
## 4  4 j jeden
## 5  5 k    dwa
## 6  6 l   trzy

# przykład bind_cols
bind_cols(data_1, data_2)

## New names:
## * x -> x...1
## * y -> y...2
## * z -> z...3
## * x -> x...4
## * y -> y...5
## * ...

##   x...1 y...2 z...3 x...4 y...5 z...6
## 1     1     a jeden     4     j jeden
## 2     2     b    dwa     5     k    dwa
## 3     3     c   trzy     6     l   trzy
```

4.6.2 *_join

Druga grupa funkcji - `*_join` obejmuje funkcje, które łączą dane według pewnego klucza. Za klucz może służyć jedna albo więcej kolumn. Przykładowym kluczem może być np. nazwa genu albo kombinacja szczepu, nr komórki i czasu obserwacji.

Możliwe zastosowania funkcji `_join` * wzbogacenie tabeli o dodatkowe informacje np. dołączenie do tabeli zawierającej ekspresję genu informacje o ich funkcji, położenia na genomie * ponowne połączenie danych rozdzielonych w wyniku analizy np. dołączenie wyniku funkcji `summarise` do wejściowej tabeli * wybranie tabeli tylko wierszy występujących w obu tabelach np. wybór genów o istotnie zmienionej ekspresji z dwóch eksperymentów RNA-seq.

Dla funkcji `*_join` kluczowe jest podanie argumentu `by`, który zidentyfikuje kolumny-klucze. Gdy w obu tabelach kolumny-klucze posiadają takie same nazwy funkcja sama je odnajdzie, w przeciwnym wypadku konieczne jest podanie ich nazw w formacie `by = c('kolumna_z_tab_1' = 'kolumna_z_tab_2')`.

przykład `left_join` – do tabeli po lewej dodaje pasujące wiersze z tabeli w nawiasie `band_instruments`

```
## # A tibble: 3 x 2
##   name   plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar

band_members

## # A tibble: 3 x 2
##   name   band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles

band_instruments %>% left_join(band_members)

## Joining, by = "name"

## # A tibble: 3 x 3
```

```

##   name  plays  band
##   <chr> <chr>  <chr>
## 1 John  guitar Beatles
## 2 Paul  bass   Beatles
## 3 Keith guitar <NA>

# z by
band_instruments2 %>% left_join(band_members, by = c('artist' = 'name'))


## # A tibble: 3 x 3
##   artist plays  band
##   <chr>  <chr>  <chr>
## 1 John   guitar Beatles
## 2 Paul   bass   Beatles
## 3 Keith  guitar <NA>

# przykład inner_join - zostawia tylko wiersze z tabeli po lewej, które pasują do tabeli
band_instruments %>% inner_join(band_members)

```

```

## Joining, by = "name"

## # A tibble: 2 x 3
##   name  plays  band
##   <chr> <chr>  <chr>
## 1 John  guitar Beatles
## 2 Paul  bass   Beatles

# przykład full_join - zostawia wszystkie wiersze z obu tabel (uzupełnia brakujące wartości)
band_instruments %>% full_join(band_members)

```

```

## Joining, by = "name"

## # A tibble: 4 x 3
##   name  plays  band
##   <chr> <chr>  <chr>
## 1 John  guitar Beatles
## 2 Paul  bass   Beatles
## 3 Keith guitar <NA>
## 4 Mick  <NA>   Stones

# przykład anti_join - tylko wiersze w tabeli po lewej, które nie mają pary w tabeli po prawej
band_instruments %>% anti_join(band_members)

```

```
## Joining, by = "name"
```

```
## # A tibble: 1 x 2
##   name   plays
##   <chr> <chr>
## 1 Keith guitar
```


Chapter 5

Wykresy - pakiet `ggplot2`

```
##  
## If anything, there should be a Law: Thou Shalt Not Even Think Of Producing A  
## Graph That Looks Like Anything From A Spreadsheet.  
##      -- Ted Harding (in a discussion about producing graphics)  
##      R-help (August 2007)
```

Do tworzenia wykresów można użyć kilku pakietów:

- **graphics** - podstawowy pakiet graficzny instalowany razem z R, pełna kontrola nad wyglądem wykresu, ale często wymaga to więcej pracy niż w innych pakietach. Przydatny jeżeli chcemy coś szybko sprawdzić, a nie interesuje nas wygląd wykresu, niektóre wykresy można stworzyć tylko w tym pakiecie
- **lattice** - umożliwia łatwe tworzenie kilku wykresów na raz np. do porównania różnych cech, ale obecnie mniej popularny niż np. ggplot2
- **ggplot2** - jeden z najpopularniejszych pakietów R i mój ulubiony do przygotowywania wykresów. Oparty na grammar of graphics. Przygotowanie “ładnych” wykresów wymaga mniej pracy niż w podstawowym, ale składnia jest znaczco różna. Łatwe porównywanie i tworzenie kilku wykresów na jednym obrazku
- **plotly** - pozwala na tworzenie interaktywnych wykresów np. do wykorzystania na stronach www lub w shiny. Pakiet ggplotly umożliwia konwersję wykresów ggplot do postaci interaktywnej

Wykresy w `ggplot2` składają się z kilku elementów: `data`, `aesthetics`, `geom`, `scale`, `facet`, `theme` itd.

Pierwsze trzy są niezbędne do przygotowania wykresu.

Poszczególne elementy można ze sobą łączyć na różne sposoby, co pozwala w prosty sposób robić bardzo różne wykresy.

Podstawowa funkcja : `ggplot()`. W niej określany jest tylko data i aesthetic, pozostałe elementy dodawane są przy pomocy +

- data - określa ramkę danych, na podstawie której będzie przygotowany wykres, najlepiej żeby była w formacie ‘tidy’
- aesthetics (aes) - pokazuje (mapuje), które kolumny mają być wykorzystane np. `x=kolumna_1` dla histogramu, można też określać pod względem których zmiennych dane mają zostać pogrupowane (group), rozróżnione (color, fill, shape itp) np. `aes(x = kolumna_1, color = kolumna_2)` znaczy że dane z kolumny `_1` zostaną wykorzystane do stowrzenia osi X, a dane z kolumny `_2` posłużą do stowrzenia skali kolorów.
- geom - rodzaj wykresu - to co widzimy - np histogram, density, bar, boxplot, point, line itp., można użyć kilka jednocześnie np. point i line, histogram i density
- scale - osie wykresu np. czy mają być logarytmiczne, procentowe, miejsce start i koniec, ale też skale kolorów, wypełnienia, kształtów itp.
- facet - umożliwia podział wykresu na kilka mniejszych pod względem danej zmiennej
- theme - wygląd poszczególnych elementów wykresu (motyw) np. czcionki, rodzaj linii, kolor tła, legenda itp., istnieje sporo już przygotowanych. Można przygotować własny i zapisać - łatwo można zrobić kilka wykresów w jednym stylu

Istnieje również funkcja `qplot`, która jest podobna do funkcji `plot` w podstawowym R, ale jej możliwości są uboższe w porównaniu z `ggplot`.

Wykresy ggplot są przypisywane do zmiennych np `p <- ggplot(...)`, zmienna p przechowuje wszystkie dane dotyczące wykresu. Możemy je sprawdzić funkcją `summary`. Dodanie kolejnych elementów do wykresu np. `p + geom_point()`, jeżeli chcemy nadpisać dotychczasowy wykres - `p <- p + geom_point()`. Korzystając z + możemy dodawać tyle elementów ile chcemy.

5.1 Używane dane

Jako przykładowe dane wykorzystane zostanie tabela `penguins` dostępna w pakiecie `palmerpenguins`. Tabela zawiera dane dotyczące 344 pingwinów należących do trzech gatunków, obserwowanych na wyspach archipelagu Palmer.

5.2 Histogram, boxplot i inne

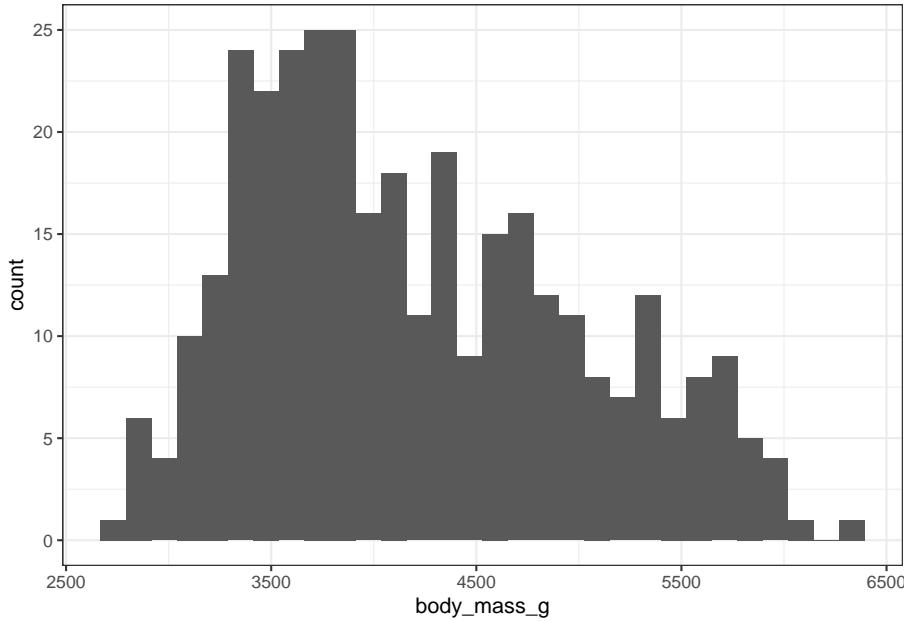
5.2.1 Histogram i density

```
library(palmerpenguins) # ładowanie pakietu z danymi
library(ggplot2) # ładowanie pakietu ggplot2
library(dplyr)
library(tidyr)
theme_set(theme_bw()) # żeby wszystkie wykresy miały białe tło, a nie szare - będzie o tym później

# histogram
p <- ggplot(data = penguins, aes(x = body_mass_g))
p + geom_histogram()

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

## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



Dla histogramu najważniejszy parametr to `binwidth` określający szerokość "słupków".

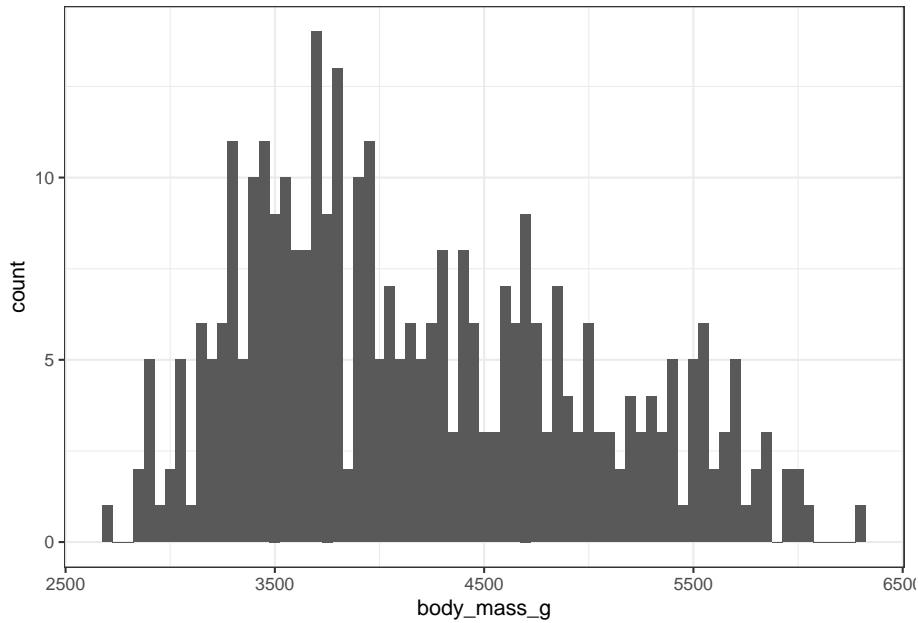
Możemy też wybrać czy chcemy żeby były zliczane ilości elementów (domyślnie), czy ma być pokazana gęstość rozkładu - w `aes` należy wpisać `y=..density..`

albo procenty $y=((..count..)/\sum(..count..))*100$ (w przypadku procentów lepiej jednak policzyć je wcześniej i podać już gotowe wartości do ggplot, w bardziej skomplikowanych przypadkach ggplot może sobie nie poradzić). Do dodania znaków % potrzebna jest zmiana parametrów osi, o tym później.

Można też zmienić kolor wypełnienia słupków - `fill` lub kolor linii - `color`.

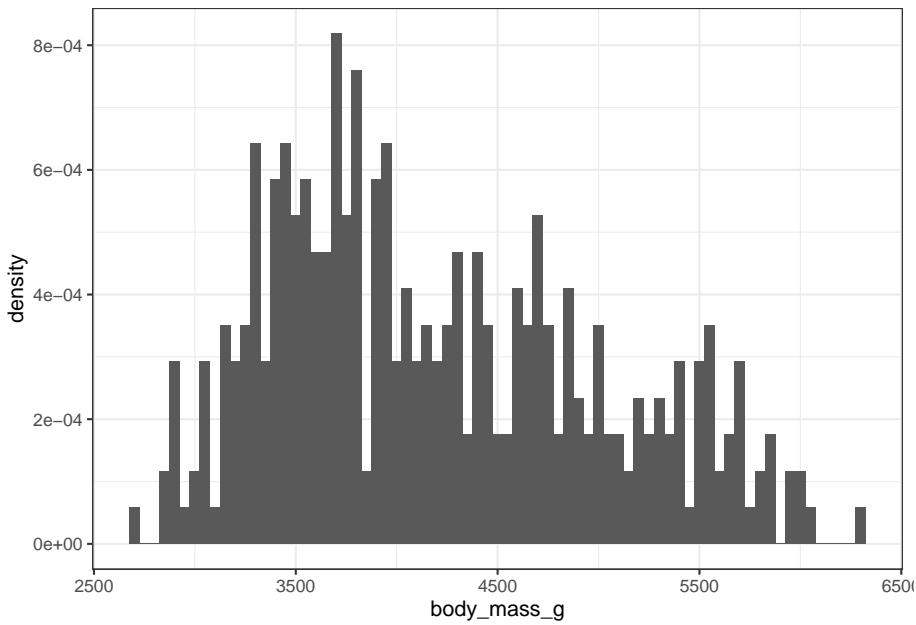
```
p + geom_histogram(binwidth = 50)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



```
# Histogram z gęstością na osi Y
p + geom_histogram(binwidth = 50, aes(y = ..density..))
```

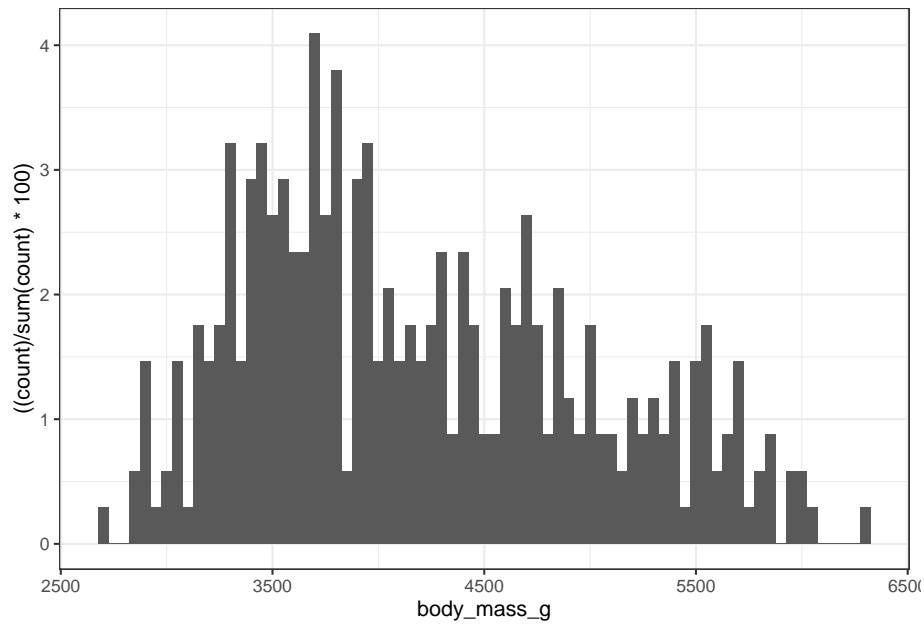
```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



```
# Histogram z wartością % na osi Y
p + geom_histogram(binwidth = 50, aes(y = ((..count..)/sum(..count..)*100)))
```

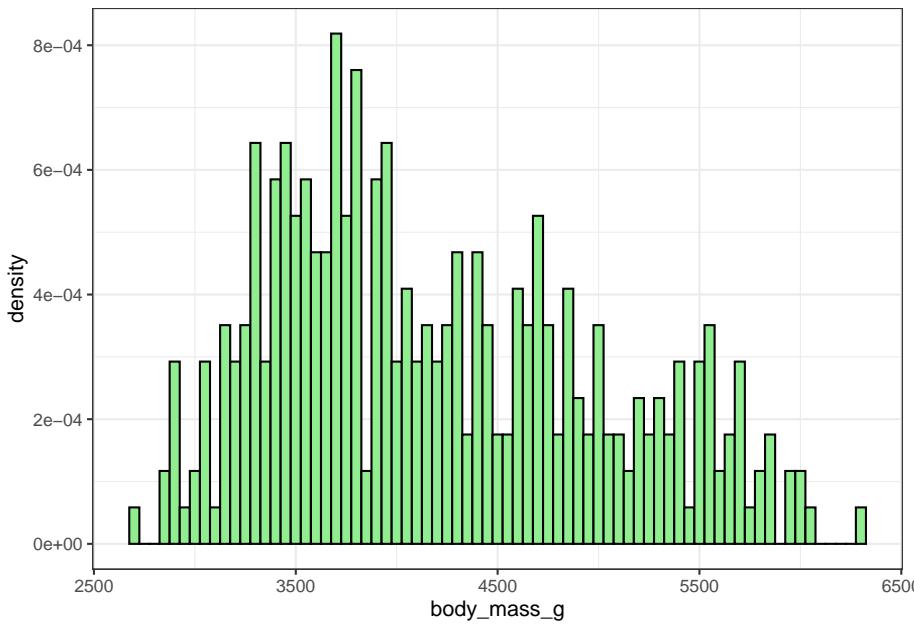


```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



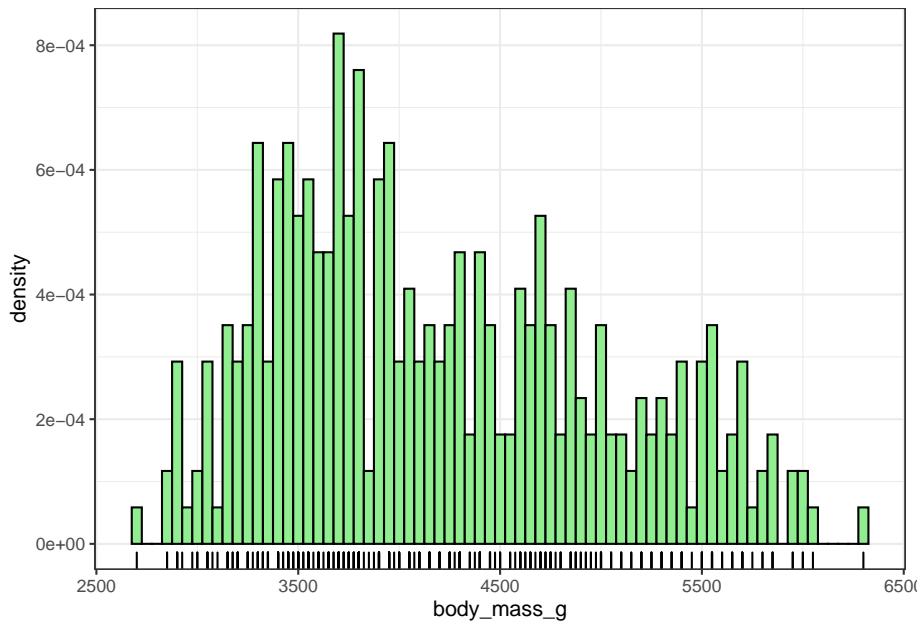
```
# Zmiana koloru wypełnienia histogramu
p + geom_histogram(binwidth = 50, aes(y = ..density..),
                    fill = "lightgreen", color = "black")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



```
# Dodanie wszystkich obserwacji do wykresu - geom_rug()
p + geom_histogram(binwidth = 50, aes(y = ..density..),
                    fill = "lightgreen", color = "black")+
    geom_rug()
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```

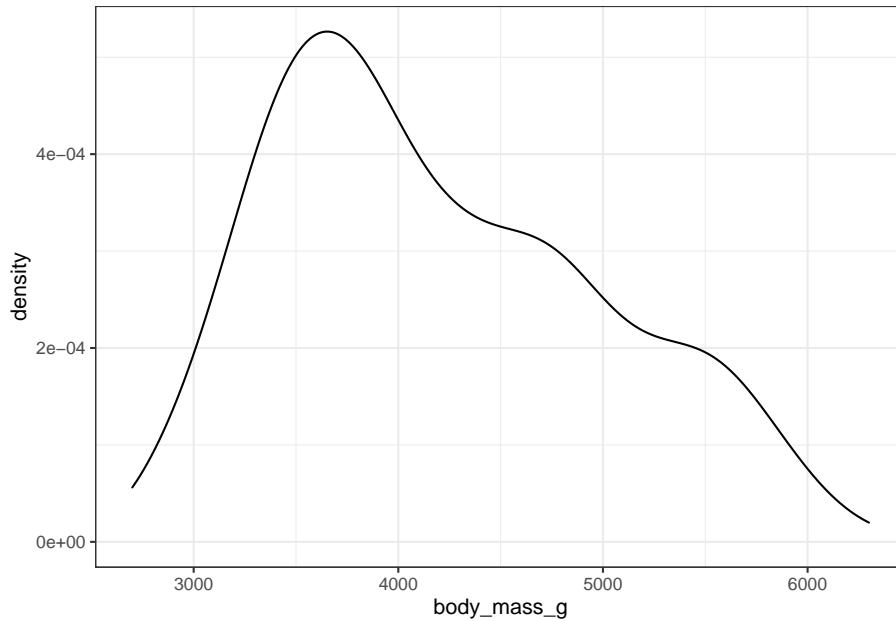


Histogram możemy łatwo zmienić na `geom_density` (gęstość) albo połączycy oba na jednym wykresie (należy pamiętać żeby ujednolicić os Y - w histogramie ustawić `y=..density..` albo w density `y = ..count..`)

Wykresy gęstości są dużo czytelniejsze od histogramów przy większej liczbie grup/kolorów na wykresie.

```
p + geom_density()
```

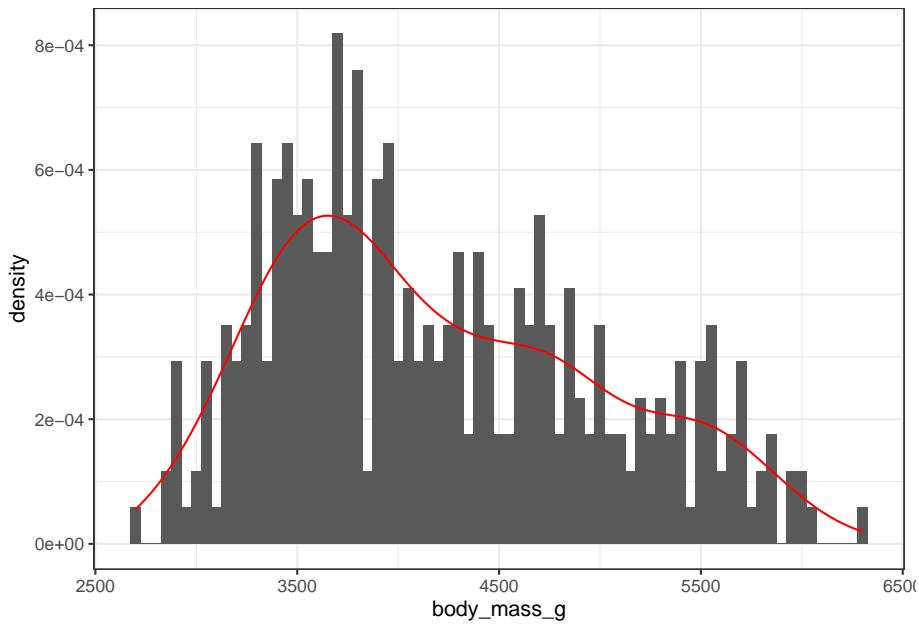
```
## Warning: Removed 2 rows containing non-finite values (stat_density).
```



```
# Wykres łączący histogram i gęstość
p + geom_histogram(binwidth = 50, aes(y = ..density..))+
  geom_density(color = "red")
```

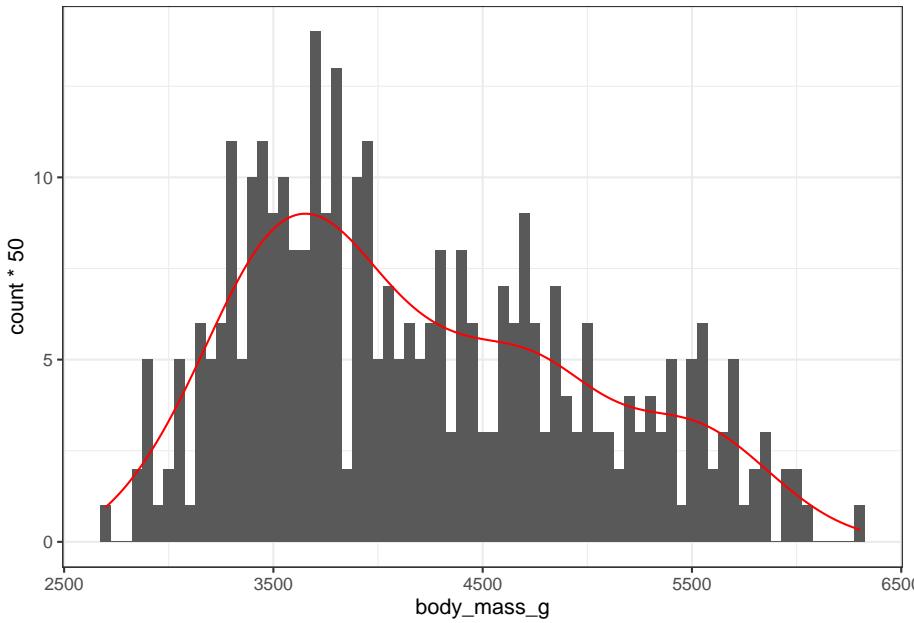


```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
## Warning: Removed 2 rows containing non-finite values (stat_density).
```



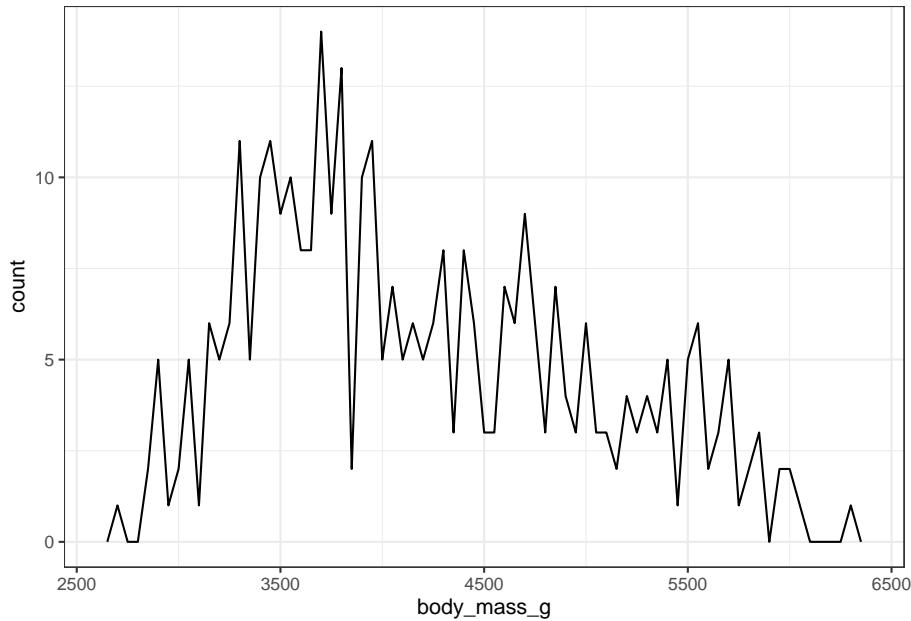
```
# albo tak, ale wtedy trzeba count w density podzielić lub pomnożyć przez konkretną liczbę
p + geom_histogram(binwidth = 50) +
  geom_density(color = "red", aes(y = ..count..*50))
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
## Warning: Removed 2 rows containing non-finite values (stat_density).
```



```
# Zamiast density można też użyć geom_freqpoly, który da bardziej "kanciasty" wykres  
# Wymaga parametru binwidth tak samo jak histogram  
p + geom_freqpoly(binwidth = 50)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



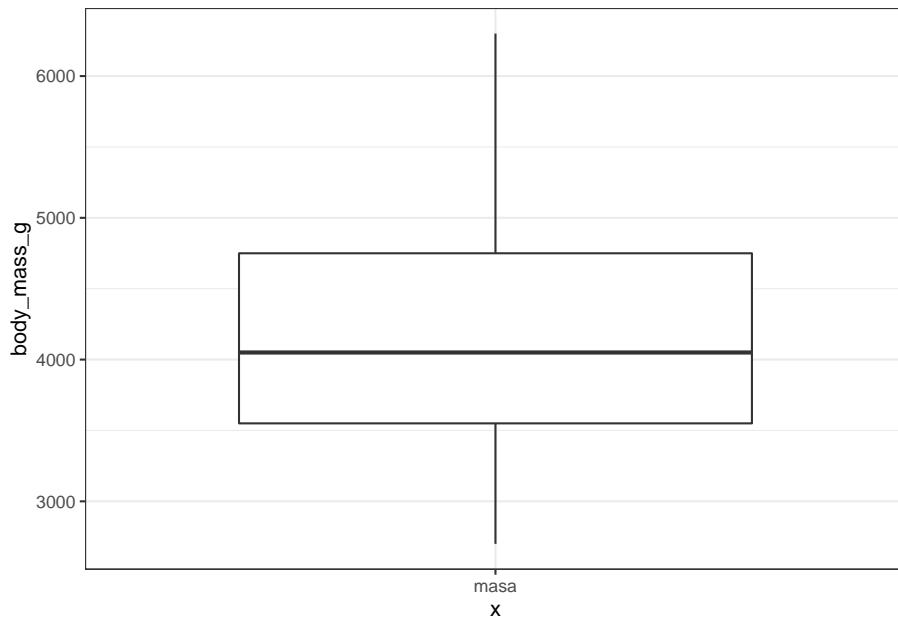
5.2.2 Wykres pudełkowy - boxplot

Na wykresie pudełkowym linia obrazuje medianę, pudełko to przestrzeń między 1 i 3 kwantylem, wąsy to zakres danych, a wszystkie punkty to obserwacje odstające.

Zamiast histogramu możemy zrobić boxplot, w tym wypadku x to nazwa szczepu, a y to mierzona cecha. Na osi X należy zawsze umieszczać zmienną jakościową, a na osi Y zmienną ilościową

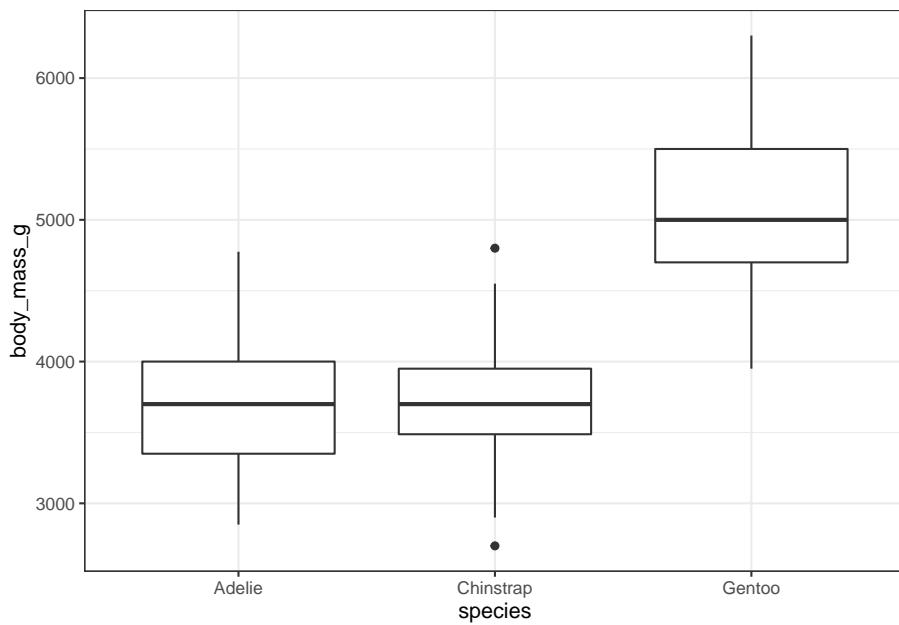
```
# pojedynczy boxplot
p <- ggplot(data = penguins, aes(x = "masa", y = body_mass_g))
p + geom_boxplot()
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```



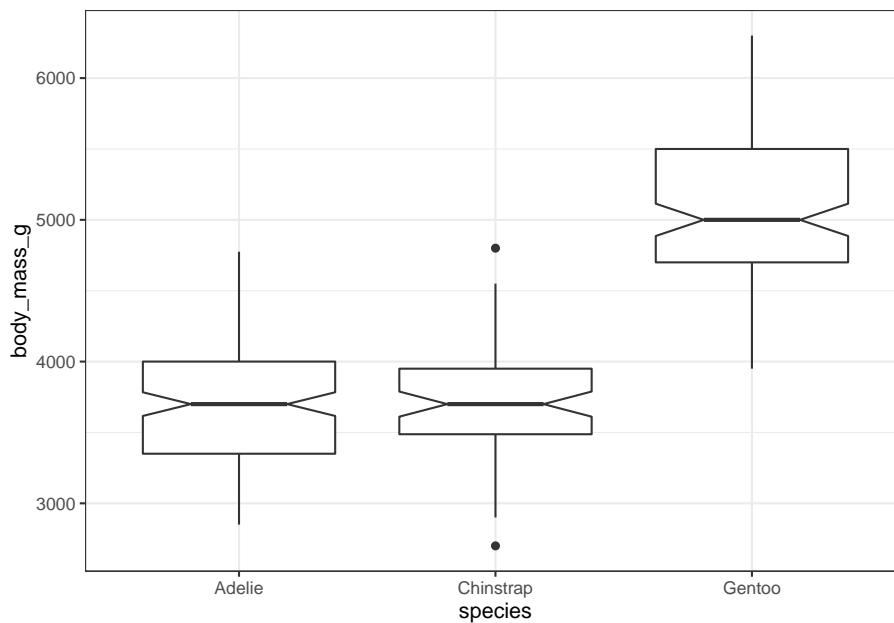
```
# boxplot dla każdej kategorii
p <- ggplot(data = penguins, aes(x = species, y = body_mass_g))
p + geom_boxplot()
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```



```
# do boxplota można dodać wcięcia, jeżeli wcięcia dwóch boxplotów na siebie nie zachodzą
# można uznać że mediany tych dwóch grup są od siebie znacznie różne
p + geom_boxplot(notch = TRUE)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```



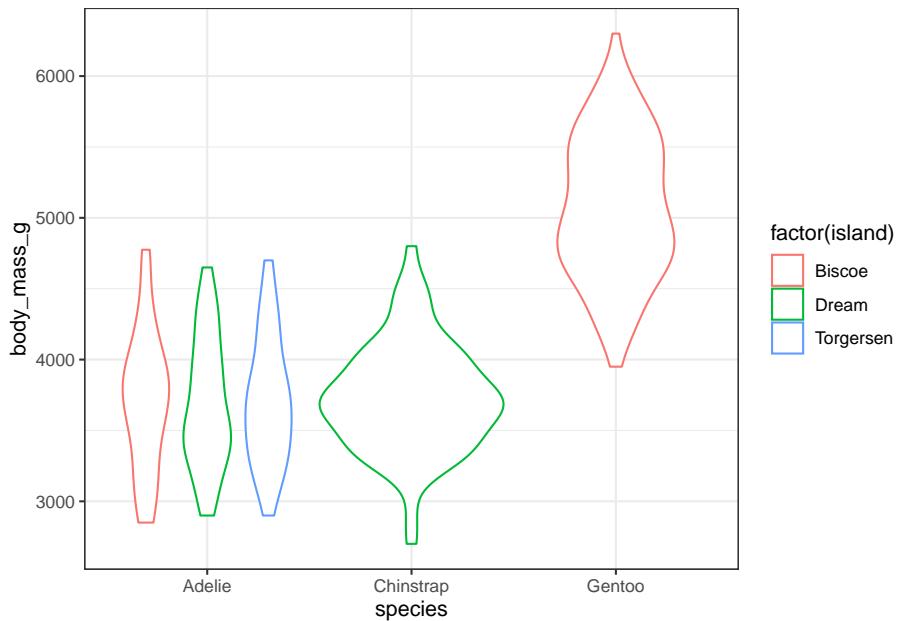
5.2.3 Wykres skrzypcowy

Odmianą boxplotów są tzw. wykresy skrzypcowe, które pozwalają też na pokazanie kształtu rozkładu - pozwala to np. na wykrycie rozkładu, który ma dwa maksima. W ggplot2 można je wygenerować funkcją `geom_violin`.

```
p <- ggplot(data = penguins, aes(x = species, y = body_mass_g))
p + geom_violin(aes(color = factor(island)))
```

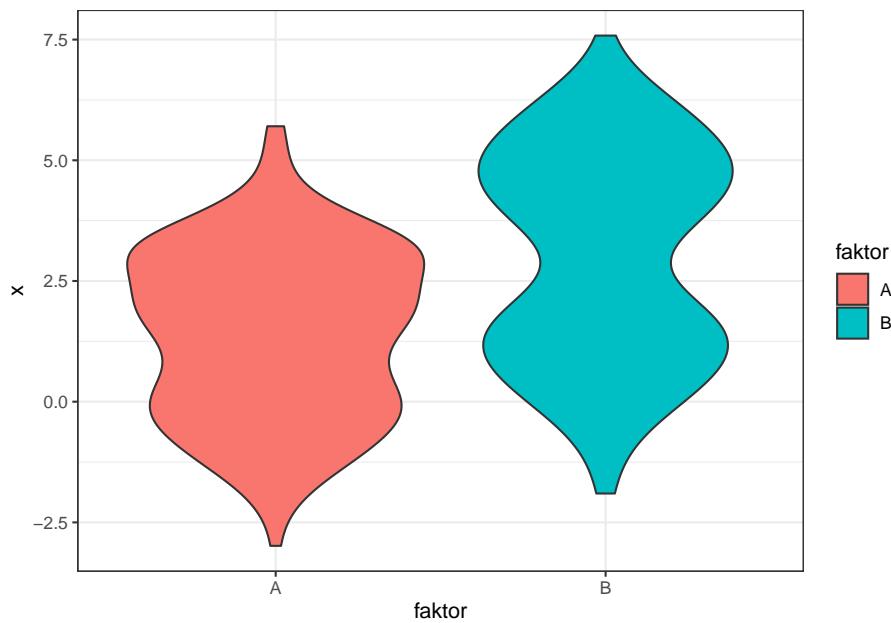


```
## Warning: Removed 2 rows containing non-finite values (stat_ydensity).
```

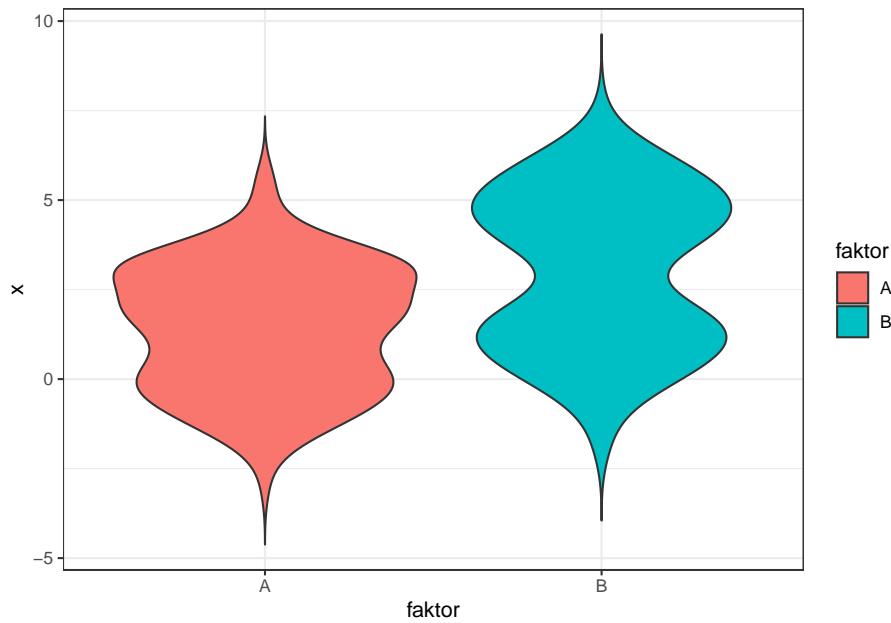


```
# przykładowe dane
dane <- data.frame(x = c(rnorm(100), rnorm(100, 3), rnorm(100, 1), rnorm(100, 5)),
                     faktor = rep(c("A", "B"), each = 200))

p <- ggplot(data = dane, aes(y = x, x = faktor, fill = faktor))
p + geom_violin()
```



```
# nie przyjęty wykres  
p + geom_violin(trim=FALSE)
```



5.2.4 Dodanie wszystkich obserwacji do boxplot/violin

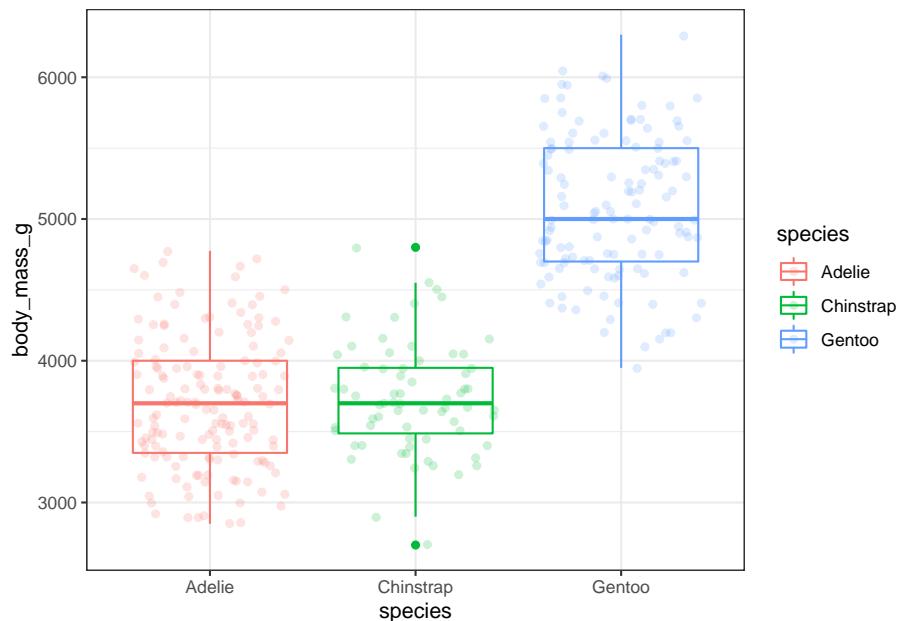
Dobrą praktyką jest przy używaniu wykresów pudełkowych lub skrzypcowych pokazanie wszystkich uzyskanych obserwacji (o ile nie ma ich za dużo). Można w tym celu użyć funkcji `geom_jitter` albo ładniejszych `geom_beeswarm` i `geom_quasirandom` z pakietu `ggbeeswarm`.

Przy pomocy argumentu `alpha` można kontrolować przezroczystość punktów - przydatne gdy jest ich dużo.

```
# Boxplot dla każdego gatunku
p <- ggplot(data = penguins, aes(x = species, y = body_mass_g))
p + geom_boxplot(aes(color = species))+
  geom_jitter(aes(color = species), alpha = 0.2)

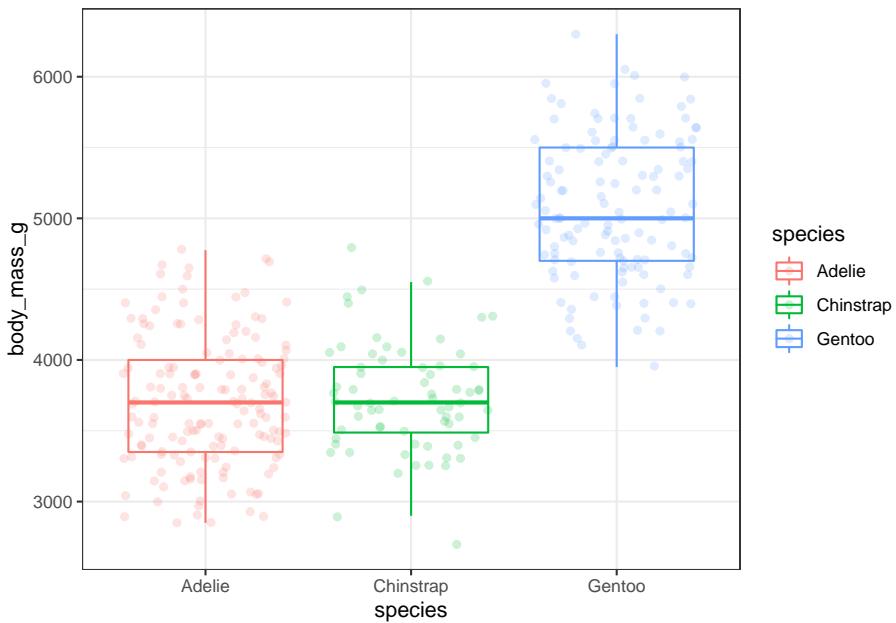
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing missing values (geom_point).
```



```
# usuwamy punkty pochodzące od boxplota
p + geom_boxplot(aes(color = species), outlier.alpha = 0)+ 
  geom_jitter(aes(color = species), alpha = 0.2)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
## Warning: Removed 2 rows containing missing values (geom_point).
```

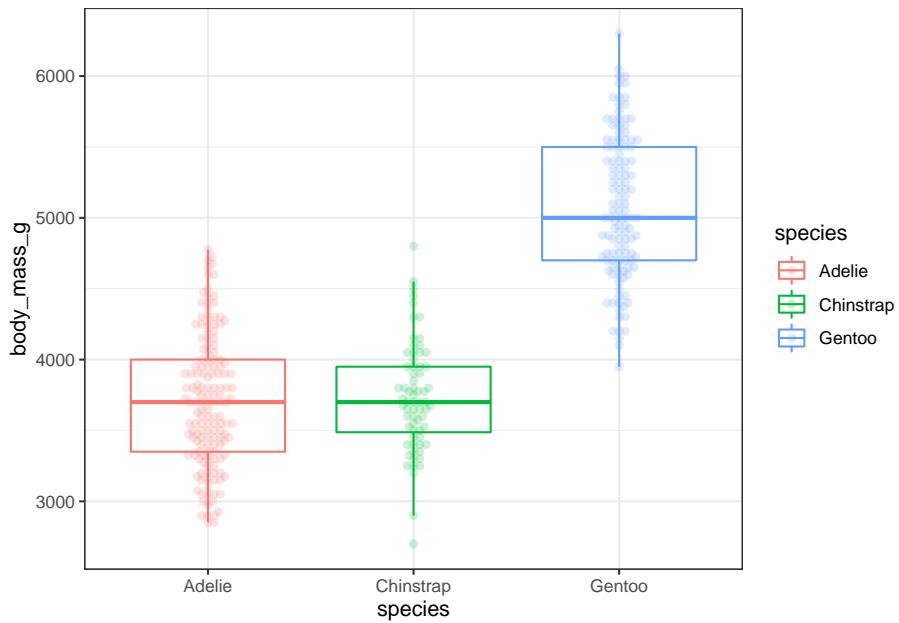


```
# beeswarm
library(ggbeeswarm)

p + geom_boxplot(aes(color = species), outlier.alpha = 0) +
  geom_beeswarm(aes(color = species), alpha = 0.2)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```

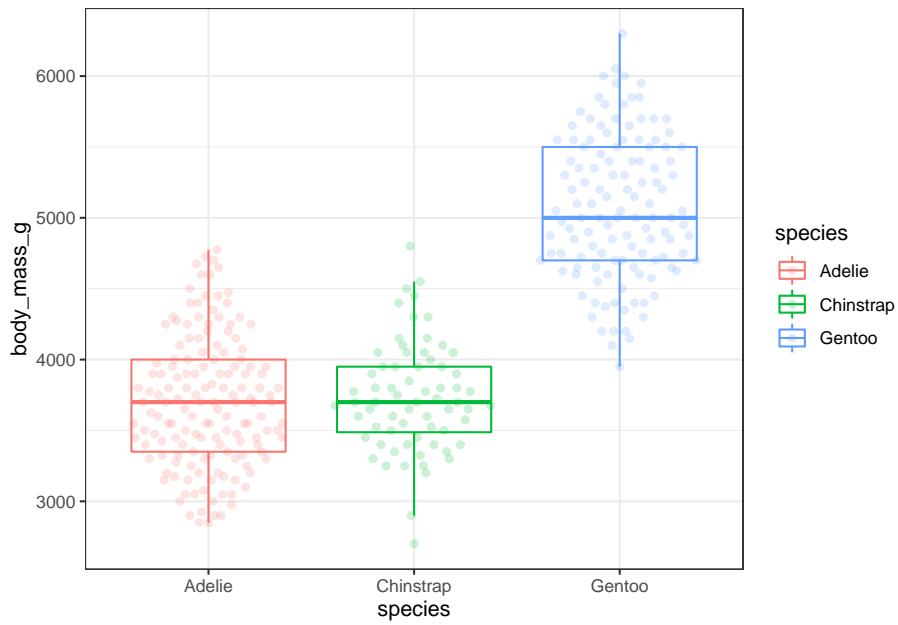
```
## Warning: Removed 2 rows containing missing values (position_beeswarm).
```



```
# quasirandom
p + geom_boxplot(aes(color = species), outlier.alpha = 0) +
  geom_quasirandom(aes(color = species), alpha = 0.2)
```

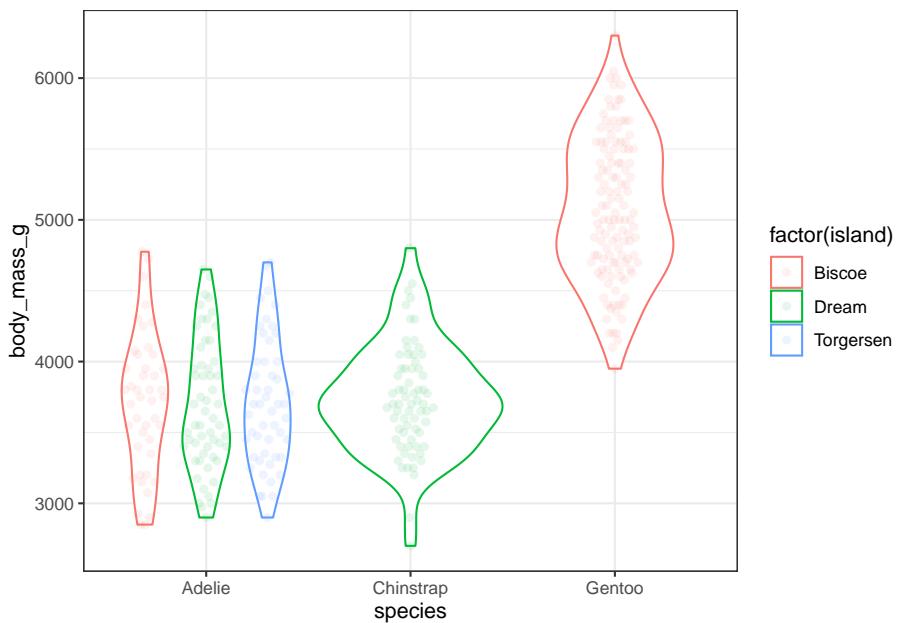
```
## Warning: Removed 2 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 2 rows containing missing values (position_quasirandom).
```



```
# quasirandom + geom_violin
p <- ggplot(data=penguins, aes(x = species, y = body_mass_g))
p + geom_violin(aes(color = factor(island)))+
  geom_quasirandom(aes(color = factor(island)),
                    dodge.width = 0.9, # pozwala na rozdzielenie kolorów na grupy
                    alpha = 0.1)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_ydensity).
## Warning: Removed 2 rows containing missing values (position_quasirandom).
```



Można też połączyć wszystko w całość korzystając z dodatkowych pakietów ggdist i gghalves, na podstawie strony: visualizing-distributions-with-raincloud-plots-with-ggplot2

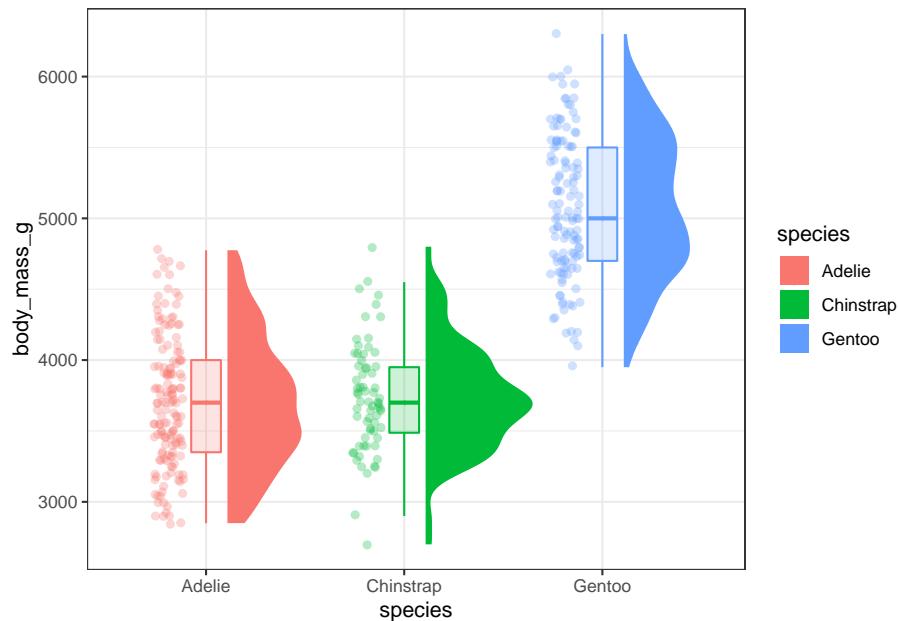
```
ggplot(penguins, aes(x = species, y = body_mass_g, color = species, fill = species)) +
  ggdist::stat_halfeye(
    adjust = .75,
    width = .6,
    .width = 0,
    justification = -.2,
    point_colour = NA
  ) +
  geom_boxplot(
    width = .15,
    outlier.shape = NA,
    alpha = 0.2
  ) +
  ## add justified jitter from the {gghalves} package
  gghalves::geom_half_point(
    ## draw jitter on the left
    side = "l",
    ## control range of jitter
    range_scale = .4,
    ## add some transparency
    alpha = .3
  )
```

```
)
```

```
## Warning: Removed 2 rows containing missing values (stat_slabinterval).

## Warning: Removed 2 rows containing non-finite values (stat_boxplot).

## Warning: Removed 2 rows containing non-finite values (stat_half_point).
```



5.2.5 Dotplot

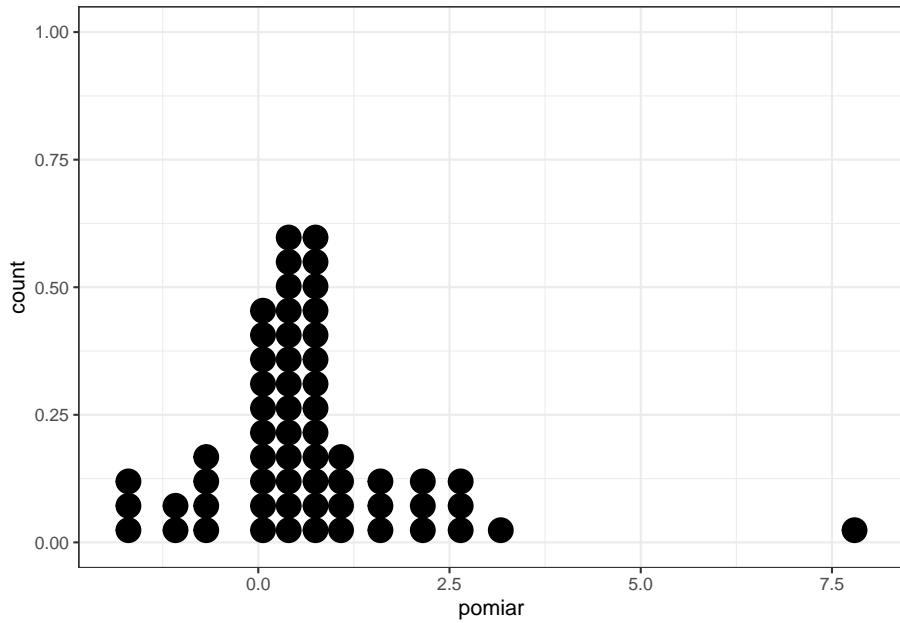
Wykres, na którym obserwacje są zaznaczane jako punkty, może być alternatywą dla histogramu albo gęstości jeżeli mamy małą liczbę danych. Punkty mogą być układane na którejś z osi albo wyśrodkowane. Podobnie jak w histogramie możemy określić parametrem `binwidth` jak mają być ułożone punkty.

```
dane_dot <- data.frame(pomiar = c(rnorm(20), rlnorm(20), runif(20)),
                         kategoria = rep(c("A","B","C"), each = 20))

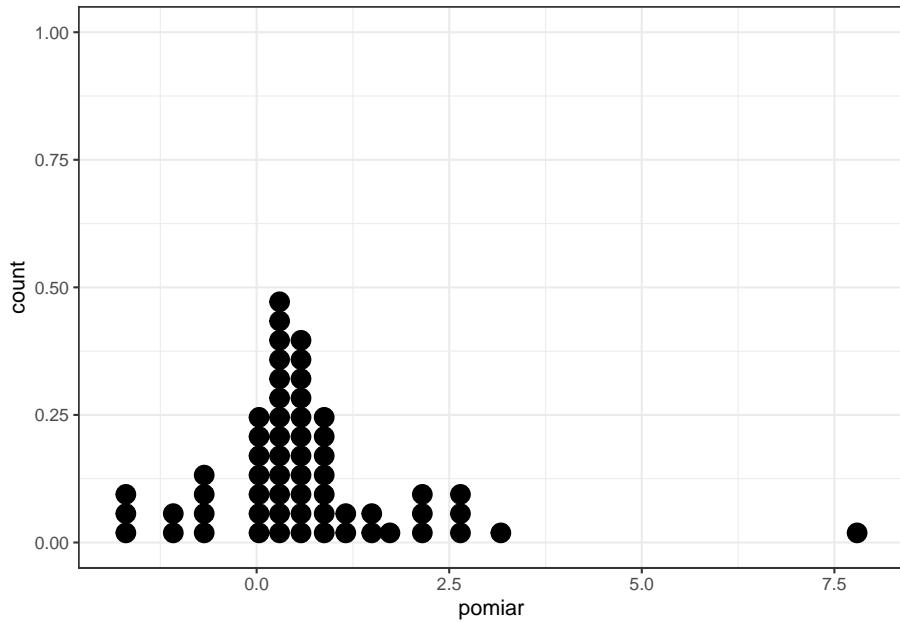
p <- ggplot(dane_dot)

p + geom_dotplot(aes(x = pomiar))
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`
```



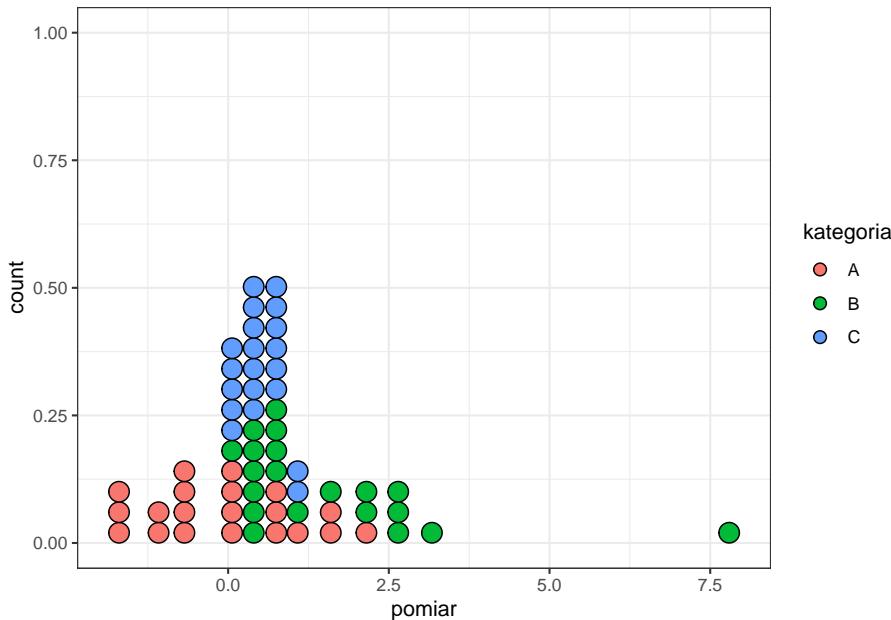
```
p + geom_dotplot(aes(x = pomiar), binwidth = 0.25)
```



```
# możemy kropki pokolorować według kategorii,
# konieczne parametry stackgroups=TRUE i binpositions="all"

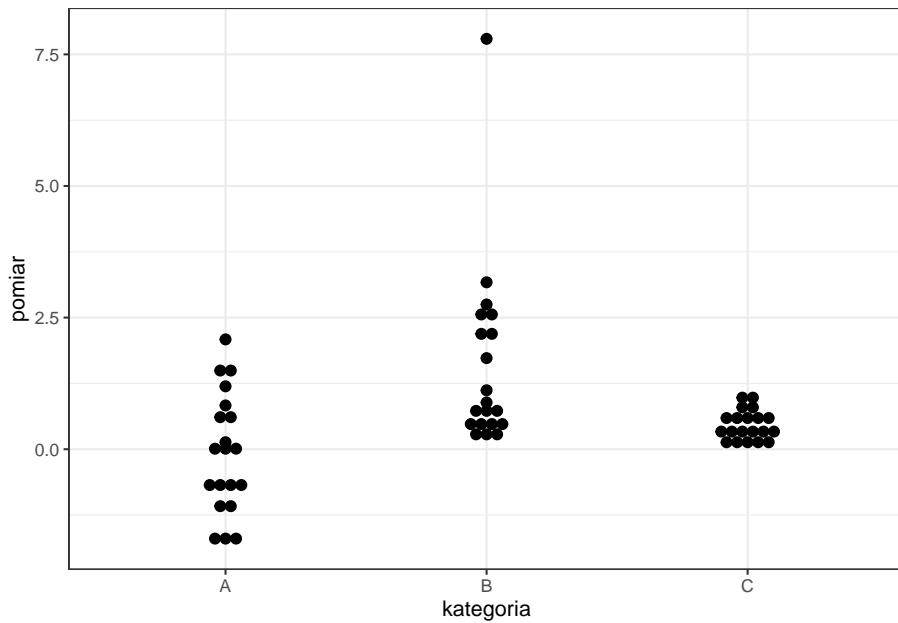
p + geom_dotplot(aes(x = pomiar, fill = kategoria), binpositions = "all", stackgroups = TRUE)

## Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.
```

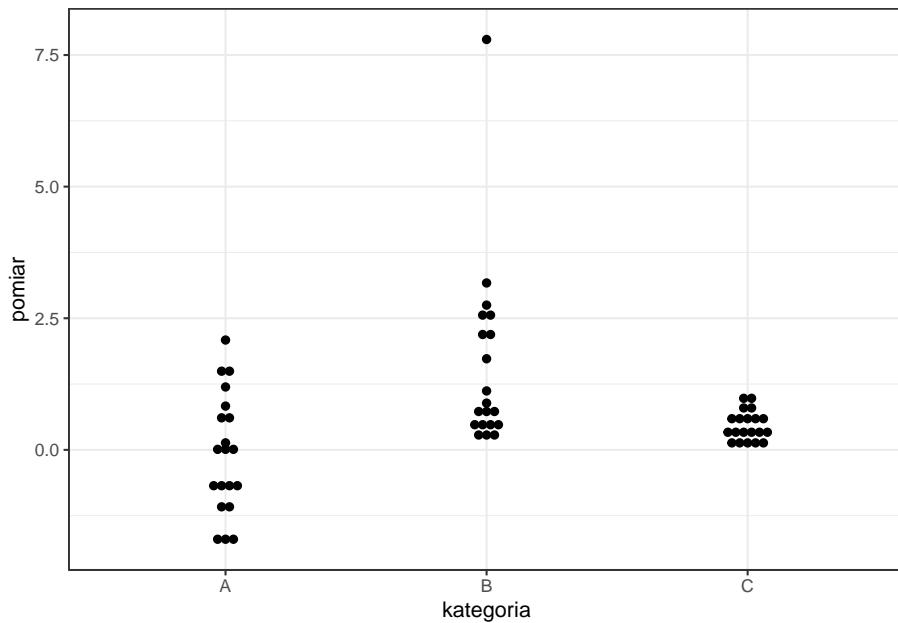


```
# albo przedstawić każdą kategorię osobno, binaxis określa w jakim kierunku układając kropki,
# stackdir czy mają być wyśrodkowane - center lub centerwhole

p + geom_dotplot(aes(y = pomiar, x = kategoria), stackdir = "center", binaxis = "y", binwidth = 0.5)
```

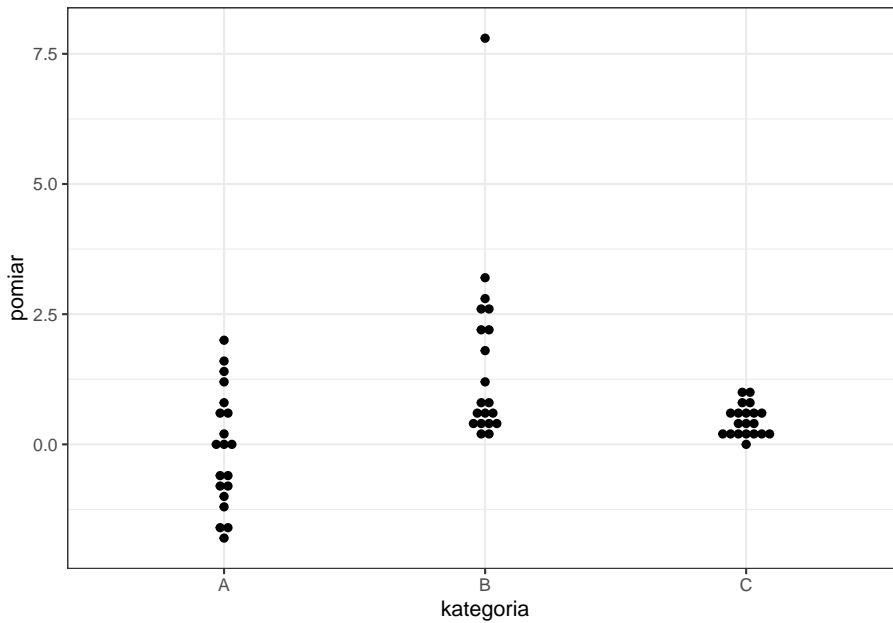


```
# zmiana wielkości kropki przez dotsize  
p + geom_dotplot(aes(y = pomiar, x = kategoria), stackdir = "center", binaxis = "y",  
                  binwidth = 0.2, dotsize = 0.75)
```



```
# można też wybrać metodę układania kropek, domyślnie wedle gęstości,
# podobnie jak histogram - method="histodot"

p + geom_dotplot(aes(y = pomiar, x = kategoria), stackdir = "center", binaxis = "y",
                  binwidth = 0.2, dotsize = 0.75, method = "histodot")
```



5.3 Barplot - wykres słupkowy

Szybkie zliczenie elementów w poszczególnych grupach możemy wykonać stosując funkcję `table`. Wystarczy podać jej kolumnę z danymi oraz kolumny zawierające wektory według których dane mają zostać podzielone na grupy. Wynikiem `table` nie jest ramka danych, więc bez przekształcenia nie można go użyć do przygotowania wykresu `ggplot`.

```
# dla jednej zmiennej
table(penguins$species)

##
##      Adelie Chinstrap     Gentoo
##          152         68        124
```

```
# dla większej liczby zmiennych
table(penguins$island, penguins$species, penguins$sex)

## , , = female
##
##          Adelie Chinstrap Gentoo
## Bischoe      22      0     58
## Dream       27     34      0
## Torgersen   24      0      0
##
## , , = male
##
##          Adelie Chinstrap Gentoo
## Bischoe      22      0     61
## Dream       28     34      0
## Torgersen   23      0      0
```

Można zauważyc, że funkcja `table`, w przeciwieństwie do `summary` pominięła wartości oznaczone jako NA. Można to zmienić używając parametr `useNA`.

```
table(penguins$island, penguins$species, penguins$sex, useNA = 'ifany')

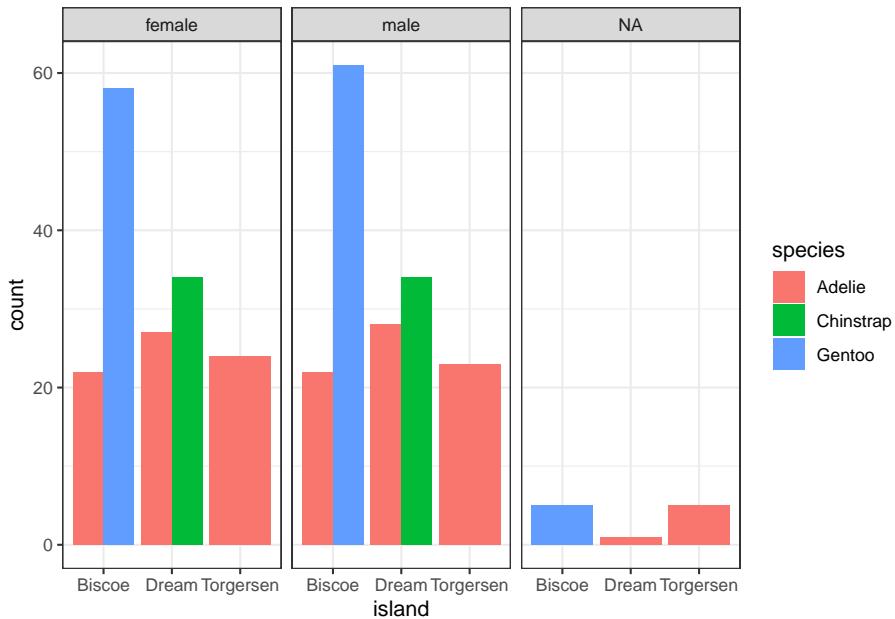
## , , = female
##
##          Adelie Chinstrap Gentoo
## Bischoe      22      0     58
## Dream       27     34      0
## Torgersen   24      0      0
##
## , , = male
##
##          Adelie Chinstrap Gentoo
## Bischoe      22      0     61
## Dream       28     34      0
## Torgersen   23      0      0
##
## , , = NA
##
##          Adelie Chinstrap Gentoo
```

```
##   Biscoe      0      0      5
##   Dream       1      0      0
##   Torgersen    5      0      0
```

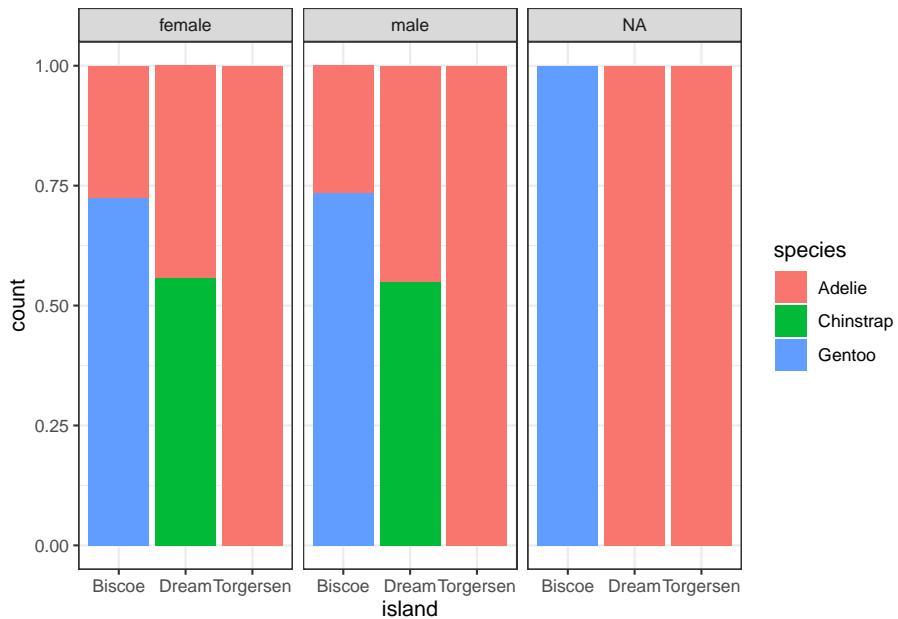
Wykres słupkowy wymaga użycia `geom_bar`. Zasady rozróżniania zmiennych według kolorów i dzielenia wykresów na części pozostają takie same.

Domyślnie również wartości NA zostaną wzięte pod uwagę podczas zliczania. Jeżeli chcemy temu zapobiec, należałoby np. usunąć je wcześniej przy pomocy funkcji `filter`.

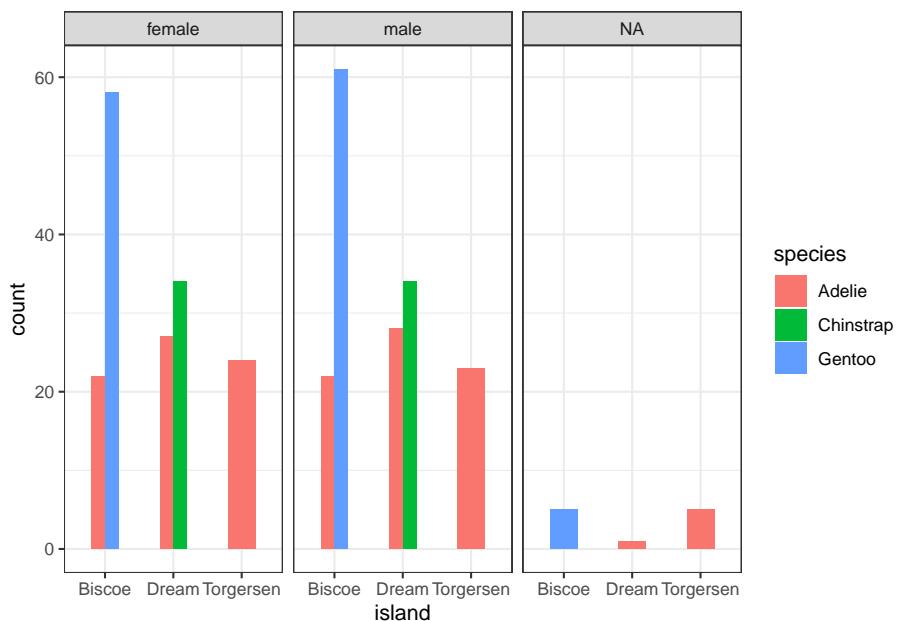
```
p <- ggplot(data = penguins, aes(x = island, fill = species))
# Słupki ustawione obok siebie
p + geom_bar(position = "dodge") + facet_wrap(~ sex)
```



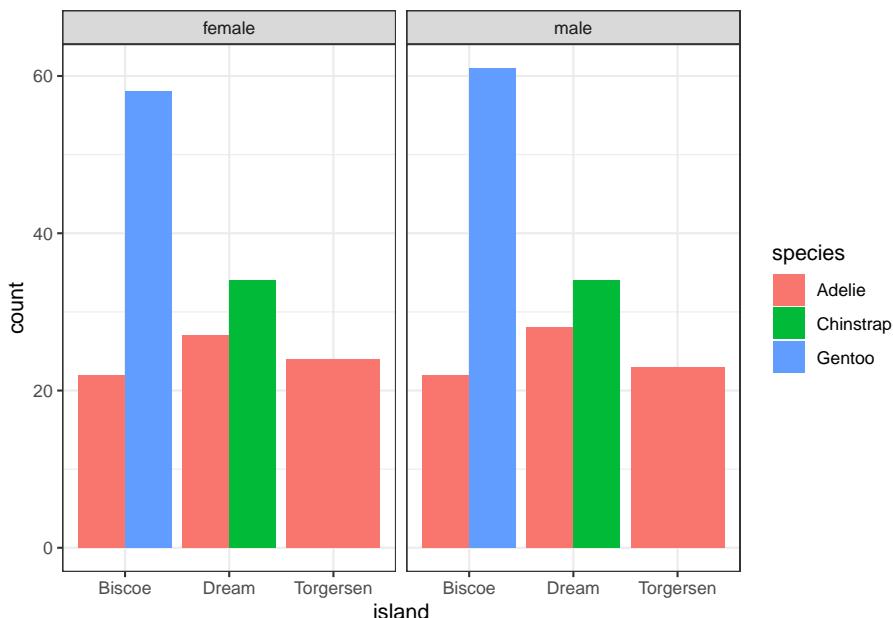
```
# Wszystkie słupki tej samej wysokości
p + geom_bar(position = "fill") + facet_wrap(~ sex)
```



```
# Szerokość słupków można zmieniać parametrem width
p + geom_bar(position = "dodge", width = 0.4) + facet_wrap(~ sex)
```

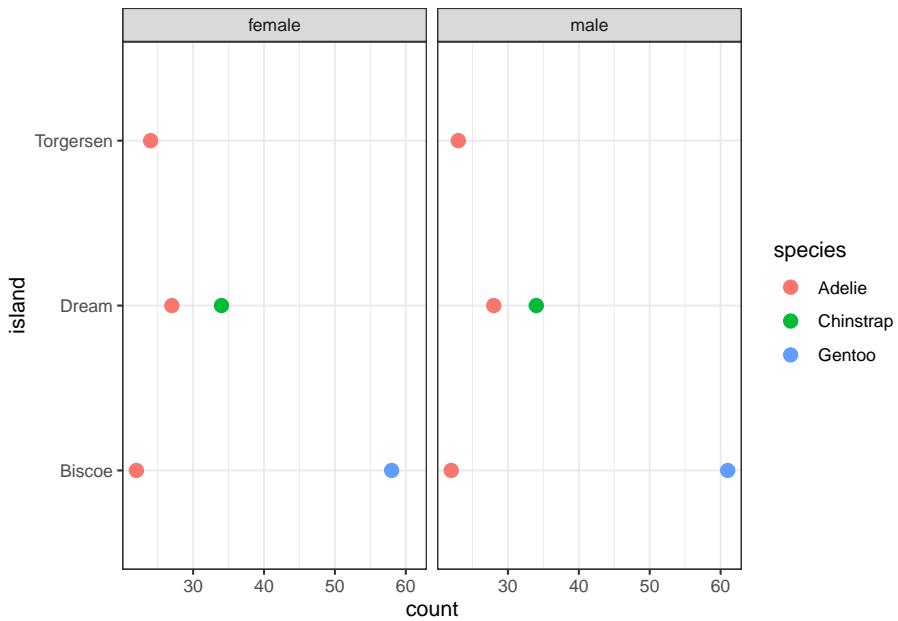


```
# Wykres bez wartości NA, filter można zrobić też wcześniej albo w obrębie ggplot
p <- ggplot(penguins %>% filter(!is.na(sex)), aes(x = island, fill = species))
p + geom_bar(position = "dodge") + facet_wrap(~ sex)
```



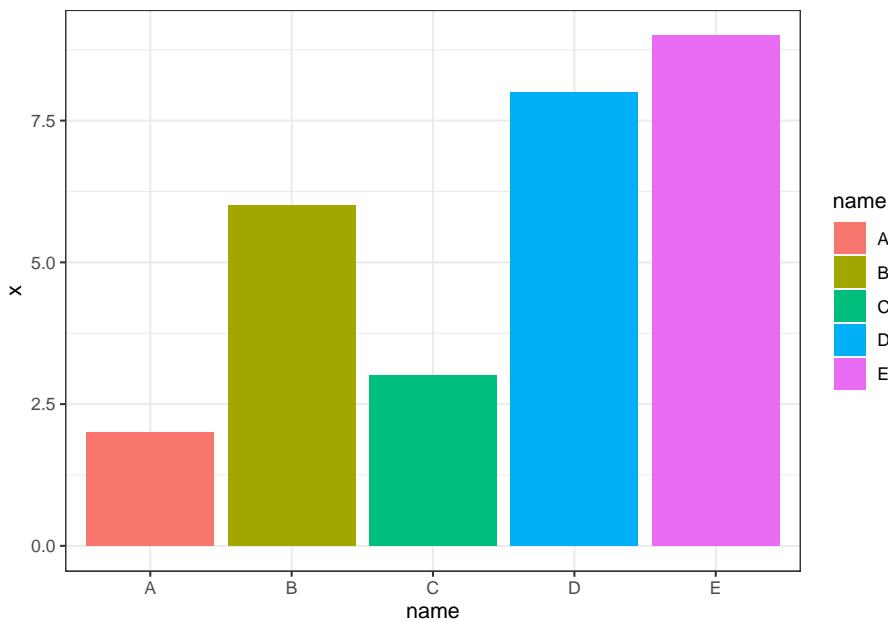
Przy dużej ilości różnokolorowych słupków wykres może być trudny do odczytania. Można wtedy rozważyć zastosowanie dotchart - czyli wykresu na którym wartości zliczenia są oznaczane przez pojedyncze punkty, a nie przez wysokość słupków. W ggplot2 do zrobienia dotchart możemy użyć `geom_point`

```
# Ustawiamy stat="bin" - oznacza że ggplot ma zliczyć częstotliwość występowania elementów,
# a nie narysować każdy z osobna i obracamy wykres - ułatwia odczytanie
p + geom_point(size=3, stat='count', aes(color=species))+
  facet_wrap(~sex)+coord_flip()
```



Nasze dane mogą też zawierać wysokości słupków. W takim wypadku należy użyć `geom_col`

```
x <- data.frame(x = c(2, 6, 3, 8, 9), name = c("A", "B", "C", "D", "E"))
p <- ggplot(data = x, aes(x = name, y = x))
p + geom_col(aes(fill = name))
```



5.4 Średnia, mediana itp. na wykresie

5.4.1 Stat_summary

Możemy również potrzebować wykres z zaznaczoną średnią i przedziałem ufności albo błędem standardowym.

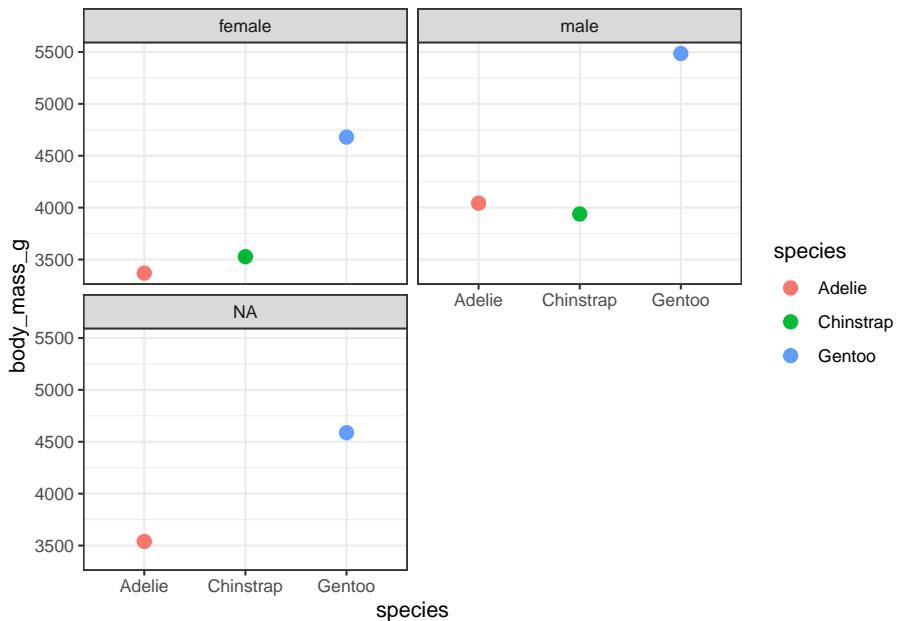
W takim wypadku możemy sami policzyć te wartości, a następnie pokazać je używając `geom_pointrange` albo `geom_crossbar`.

Można też wykorzystać `stat_summary`, który policzy je za nas.

W `stat_summary` najważniejszym argumentem jest funkcja jaką zastosujemy do podsumowania danych. Może to być albo `fun.y` - należy podać funkcję, której wynikiem jest jedna wartość np. `mean`, `median`, `sd` albo `fun.data` - należy podać funkcję, której wynikiem jest więcej wartości np. `mean_cl_boot` i `mean_cl_normal` policzą średnią i przedział ufności. Należy też dobrać odpowiedni geom: dla jednej wartości np. `point` albo `line`, dla większej: `linerange`, `crossbar`, `pointrange`, `errorbar`.

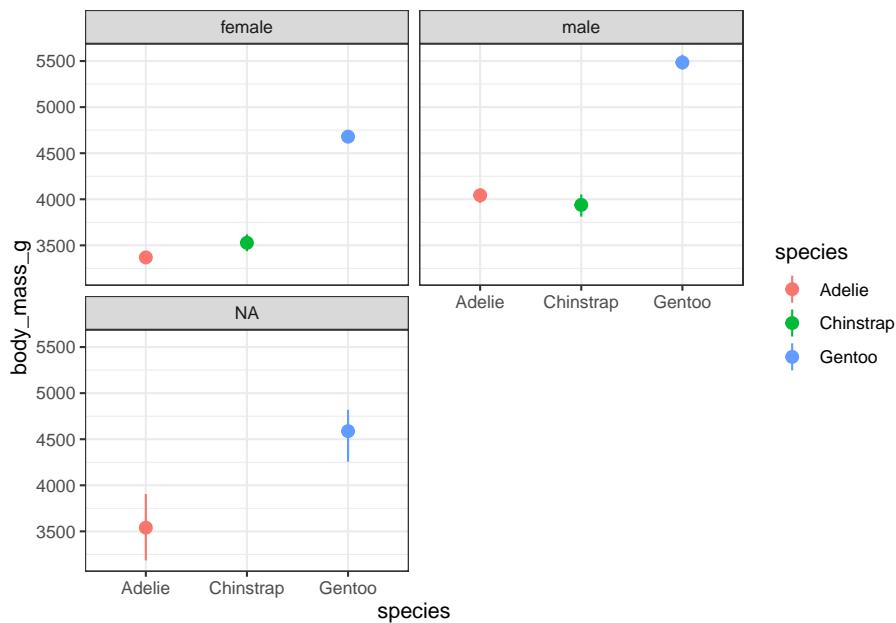
```
p <- ggplot(data = penguins, aes(x = species, y = body_mass_g, color = species))
# Wykres tylko z wartością średnią
p + stat_summary(fun = "mean", geom = "point", size = 3) +
  facet_wrap(~ sex, ncol = 2)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_summary).
```



```
# Wykres ze średnią i przedziałem ufności policzonym metodą bootstrap  
p + stat_summary(fun.data = "mean_cl_boot") +  
  facet_wrap(~ sex, ncol = 2)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_summary).
```



5.4.2 Summary z użyciem dplyr

Do samodzielnego policzenia średnich możemy wykorzystać pakiet dplyr. Pozwala on m.in. na szybkie tworzenie podsumowań danych ze względu na zmienne np. różne szczepy.

Najpierw dane są dzielone na grupy, następnie każda grupa jest poddawana działaniu pewnej funkcji lub kilku funkcji, a wynik jest zapisywany do nowej ramki danych.

W pakiecie dplyr pierwszym krokiem jest podział na grupy - `group_by` i potem przekazania wyniku do funkcji `summarize`.

```
library(dplyr)

summ <- penguins %>% group_by(species, sex) %>%
  summarize(mean = mean(body_mass_g), sdch = sd(body_mass_g),
            blad = sdch/sqrt(length(body_mass_g)),
            lower = mean-blad,
            upper = mean+blad)

## `summarise()` has grouped output by 'species'. You can override using the `groups` argument.
```

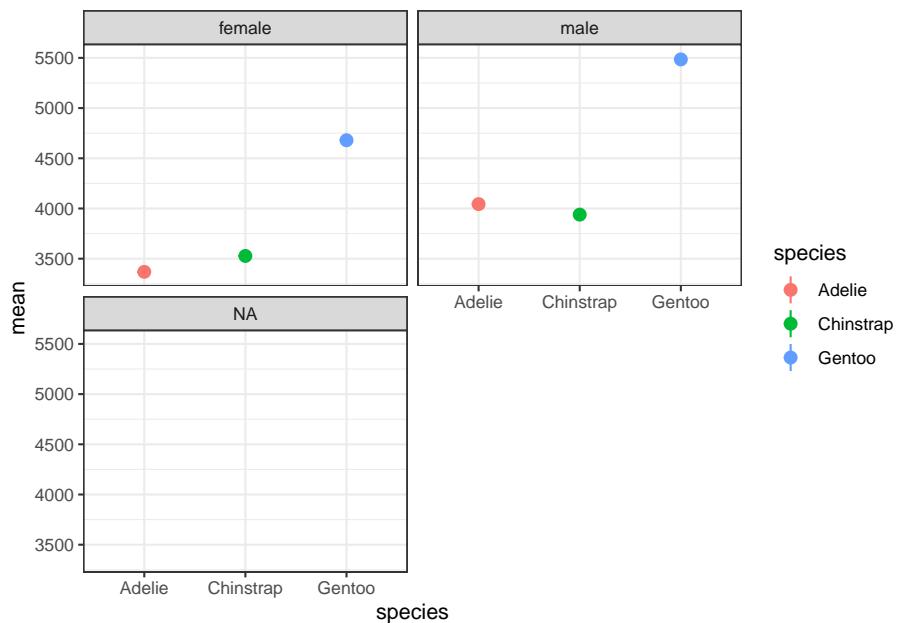
```
summ
```

```
## # A tibble: 8 x 7
## # Groups:   species [3]
##   species   sex     mean    odch   blad lower upper
##   <fct>     <fct>   <dbl>   <dbl>  <dbl> <dbl> <dbl>
## 1 Adelie   female  3369.  269.   31.5  3337. 3400.
## 2 Adelie   male    4043.  347.   40.6  4003. 4084.
## 3 Adelie   <NA>     NA     NA     NA     NA     NA
## 4 Chinstrap female  3527.  285.   48.9  3478. 3576.
## 5 Chinstrap male   3939.  362.   62.1  3877. 4001.
## 6 Gentoo   female  4680.  282.   37.0  4643. 4717.
## 7 Gentoo   male    5485.  313.   40.1  5445. 5525.
## 8 Gentoo   <NA>     NA     NA     NA     NA     NA

p <- ggplot(data = summ, aes(x = species, y = mean, ymin = lower, ymax = upper,
                           color = species)) + facet_wrap(~ sex, ncol = 2)

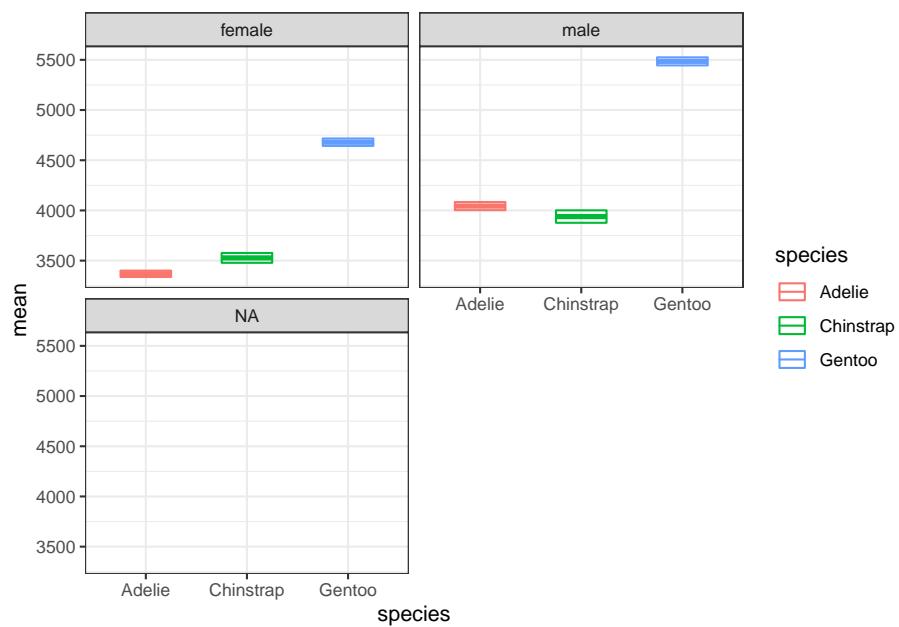
# Wykres ze średnią i błędem standardowym
p + geom_pointrange()
```

```
## Warning: Removed 2 rows containing missing values (geom_pointrange).
```



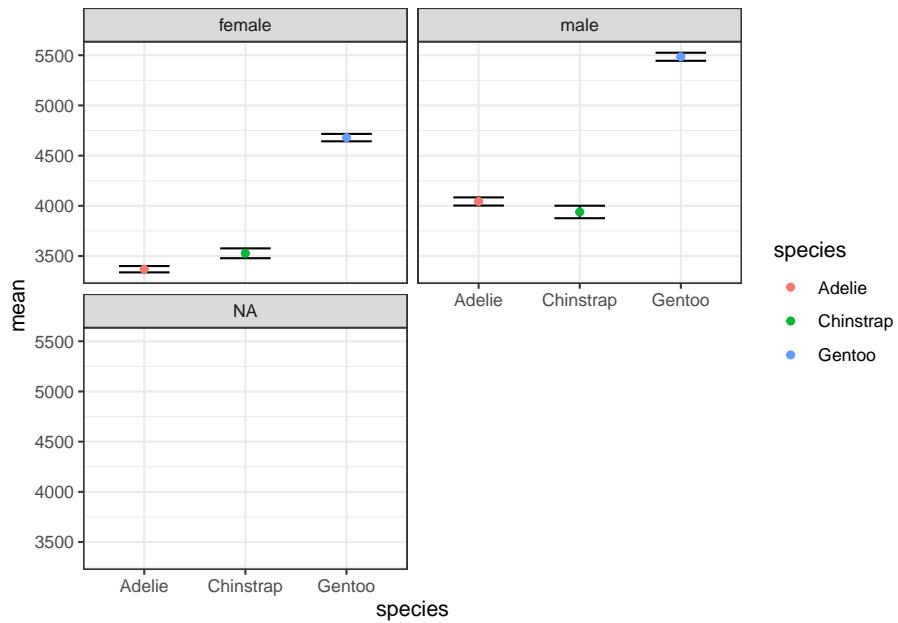
```
p + geom_crossbar(width = 0.5)
```

```
## Warning: Removed 2 rows containing missing values (geom_crossbar).
```



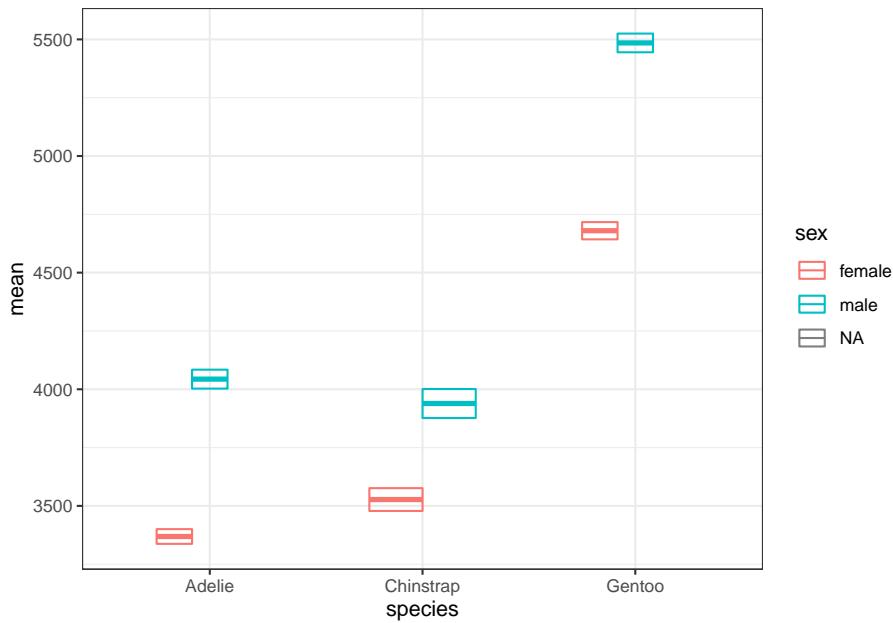
```
p + geom_errorbar(width = 0.5, color = "black") + geom_point()
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



```
# Z wykresem przedstawiającym średnie można postąpić tak samo jak z boxplotem
p <- ggplot(data = summ, aes(x = species, y = mean, ymin = lower, ymax = upper,
                               color = sex))
p + geom_crossbar(width = 0.5, position = "dodge")
```

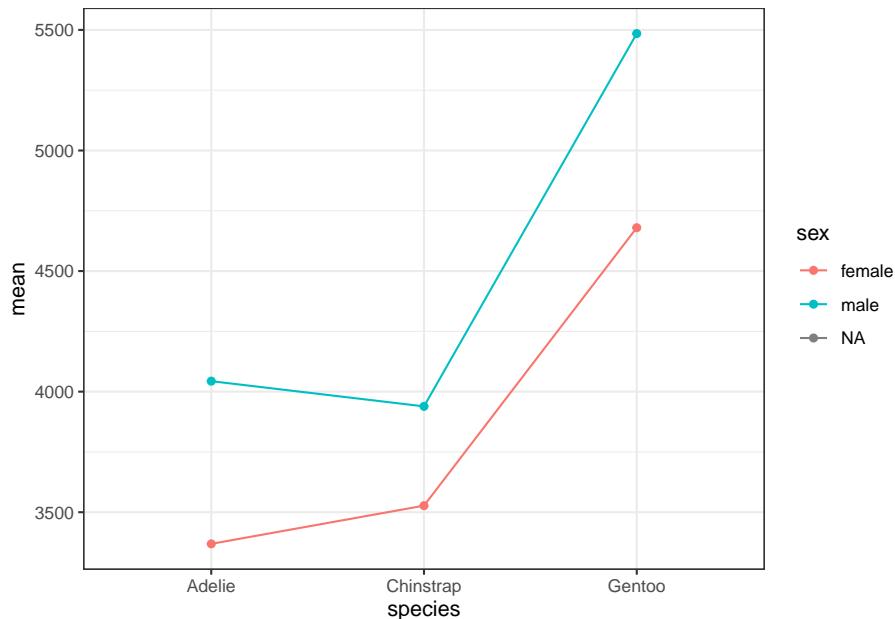
```
## Warning: Removed 2 rows containing missing values (geom_crossbar).
```



```
# Jeżeli średnie połączymy za pomocą linii otrzymamy wykres interakcji opisany w dalszej części
p <- ggplot(data = summ, aes(x = species, y = mean, color = sex))
p + geom_point() + geom_line(aes(group = sex))
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

```
## Warning: Removed 2 row(s) containing missing values (geom_path).
```



5.4.3 Słupki błędów

W publikacjach często spotyka się wykresy słupkowe z dodanym słupkiem błędu. Podobny efekt w ggplot2 można uzyskać przy pomocy `geom_errorbar` albo `geom_linerange`. Należy w `aes` podać dolną i górną granicę słupka. Jednak jeżeli słupki pochodzą np. z trzech powtórzeń danego eksperymentu to może lepiej byłoby je zaznaczyć w postaci kropek z przedziałem ufności niż rysować słupki.

Wartość średnią i ewentualny błąd standardowy albo przedział ufności możemy policzyć sami albo wykorzystać w tym celu `stat_summary`.

```
# Przykładowe dane

dane <- data.frame(kontrola = c(3,7,6), eksp_1 = c(5,9,7.5), eksp_2 = c(3,1,6), eksp_3 = c(4,8,6))

# przechodzimy do formatu tidy
library(tidyr)

dane %>% pivot_longer(cols = everything(), names_to = 'key', values_to = 'value') -> dane
head(dane)

## # A tibble: 6 x 2
##   key     value
##   <chr>   <dbl>
## 1 kontrola  3.00
## 2 eksp_1    5.00
## 3 eksp_2    3.00
## 4 eksp_3    4.00
## 5 kontrola  7.00
## 6 eksp_1    9.00
```

```

##   <chr>    <dbl>
## 1 kontrola     3
## 2 eksp_1       5
## 3 eksp_2       3
## 4 eksp_3      10
## 5 eksp_4       1
## 6 kontrola     7

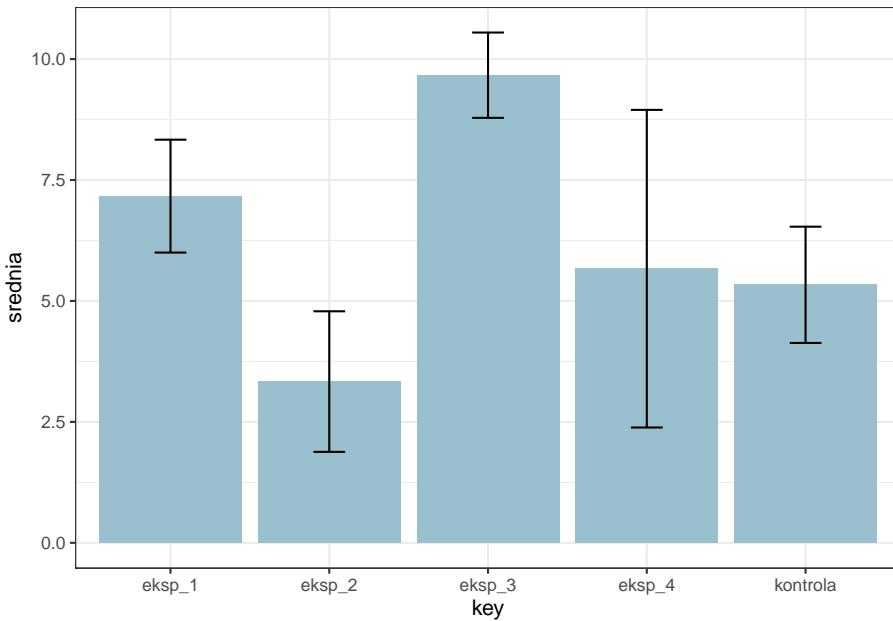
# liczymy średnią i np. błąd standardowy

podsumowanie <- dane %>% group_by(key) %>%
  summarize(
    srednia = mean(value),
    odchylenie = sd(value),
    dolny = srednia-(odchylenie/sqrt(length(value))),
    gorny = srednia+(odchylenie/sqrt(length(value))))
  
```

Wykres słupkowy z zaznaczonym błędem standardowym

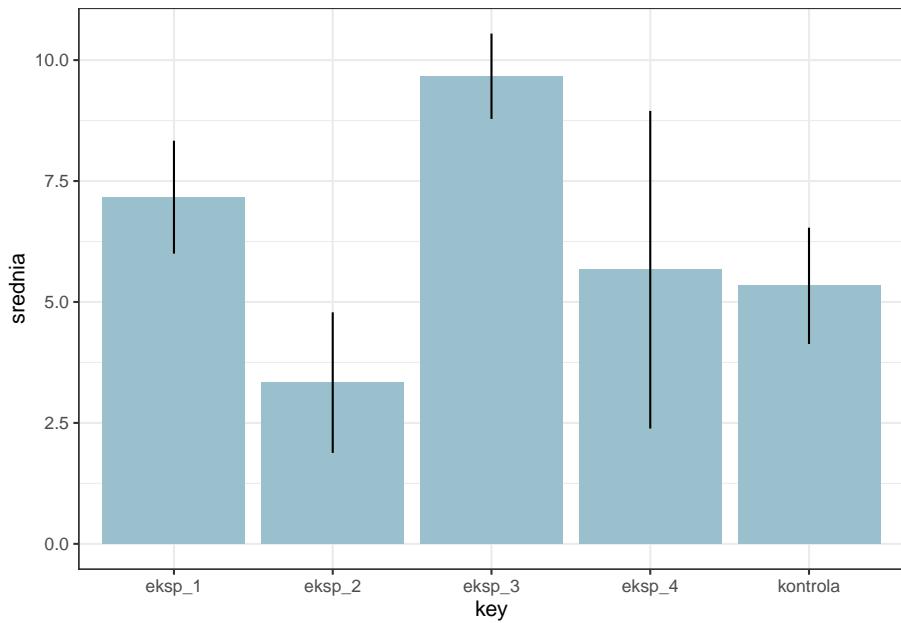
```

p <- ggplot(data = podsumowanie)
p + geom_col(aes(x = key, y = srednia), fill = "lightblue3")+
  geom_errorbar(aes(ymax = gorny, ymin = dolny, x = key), width = 0.2)
  
```

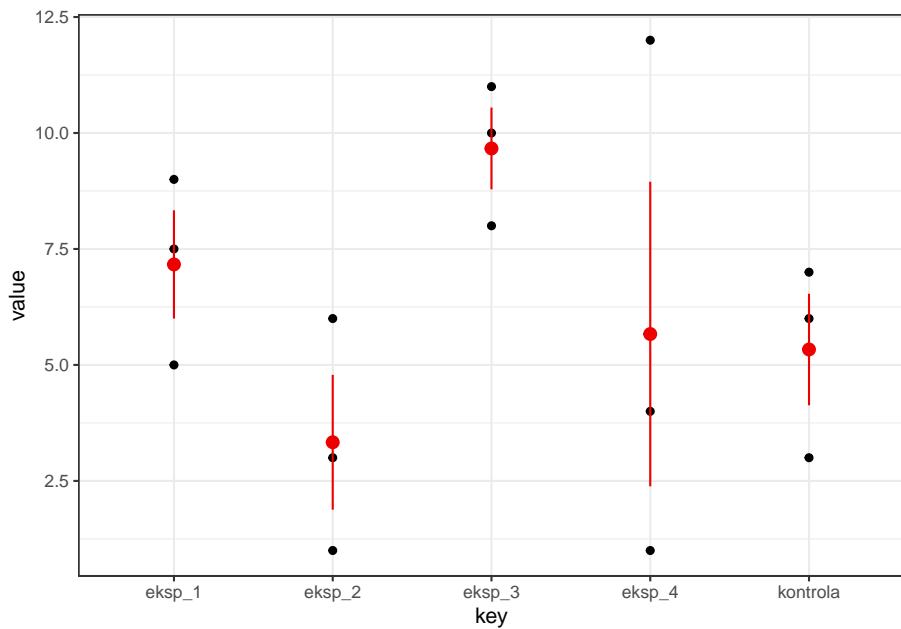


```

p + geom_col(aes(x = key, y = srednia), fill = "lightblue3")+
  geom_linerange(aes(ymin = dolny, ymax = gorny, x = key))
  
```

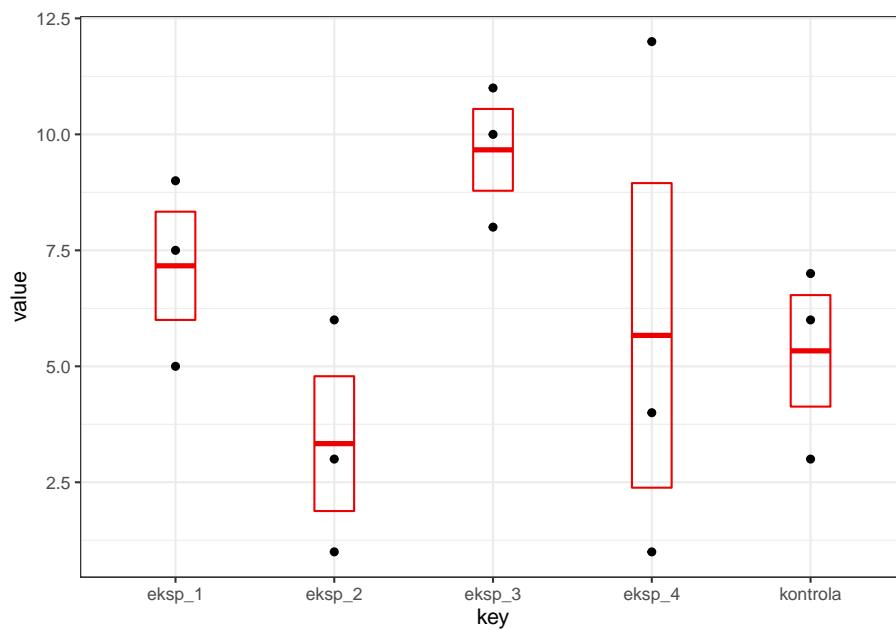


```
# Wykres punktowy wyników z zaznaczonym błędem standardowym
p + geom_point(data = dane, aes(x = key, y = value)) +
  geom_pointrange(aes(x = key, ymin = dolny, ymax = gorny, y = srednia), col = "red2")
```



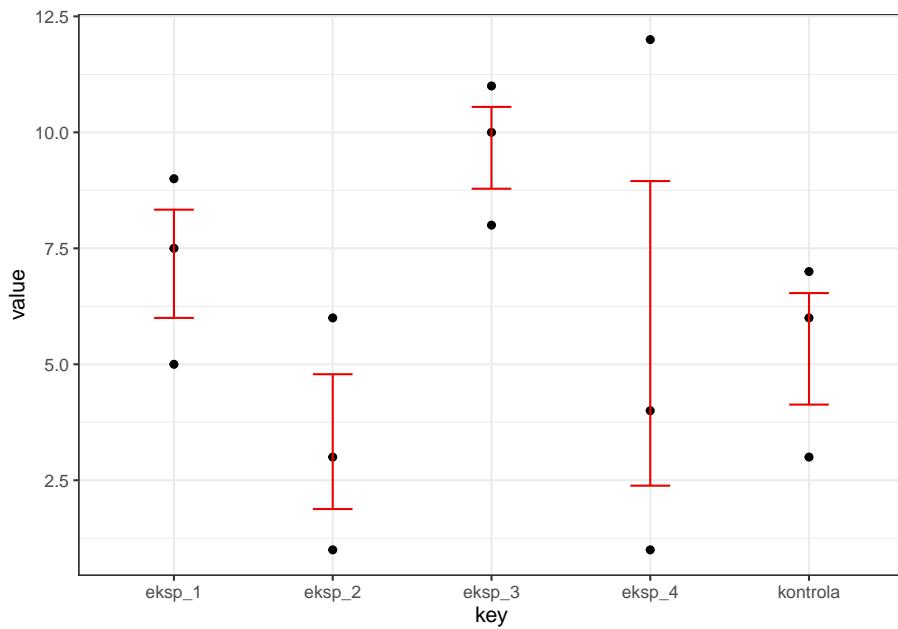
```
p + geom_point(data = dane, aes(x = key, y = value)) +
```

```
geom_crossbar(aes(x = key, ymin = dolny, ymax = gorny, y = srednia), col = "red2", width = 0.25)
```



```
p + geom_point(data = dane, aes(x = key, y = value)) +
```

```
geom_errorbar(aes(x = key, ymin = dolny, ymax = gorny, y = srednia), col = "red2", width = 0.25)
```



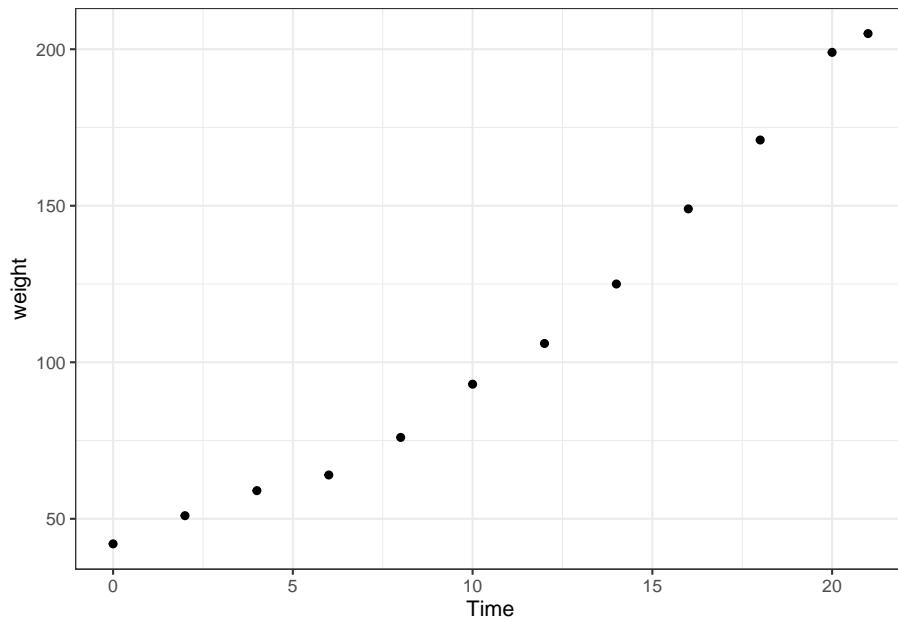
5.5 Wykres punktowy i liniowy

Do przedstawienia wykresu liniowego użyjemy zbioru danych ChickWeight, który zawiera wagę kurczaków karmionych różnymi rodzajami karmy

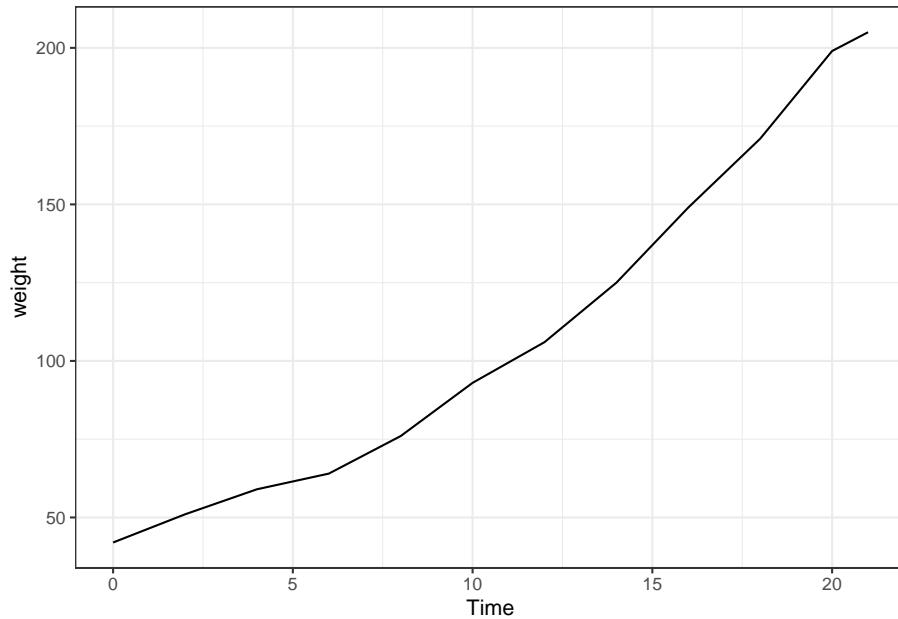
Zacznijmy od pojedynczego kurczaka

Dla `geom_point` albo `geom_line` konieczne jest w `aes` podanie `x` i `y`.

```
chick <- filter(ChickWeight, Chick == "1")
p <- ggplot(data = chick, aes(x = Time, y = weight))
p + geom_point()
```



```
p + geom_line()
```

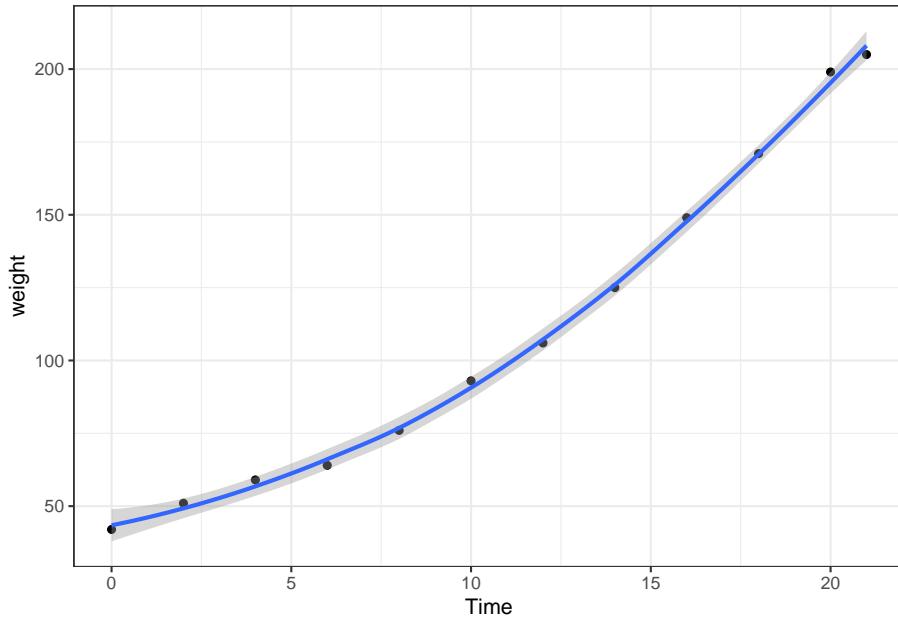


5.5.1 Dopasowanie linii trendu do wykresu punktowego

Zamiast rysować linię, możemy dopasować do wyników linię trendu (może być też inna niż liniowa np. logarytmiczna, kwadratowa, ograniczając nas tylko umiejętności pisania formuł w R ;)). Dopasowanie wykona `stat_smooth`. Domyślnie dopasuje linię do danych korzystając z własnego algorytmu (loess - lokalne wygładzanie wielomianami niskich stopni ;)), który ma za zadanie uzyskać jak najlepsze dopasowanie do danych, zaznaczy również przedział ufności. Możemy narzucić własną metodę i formułę.

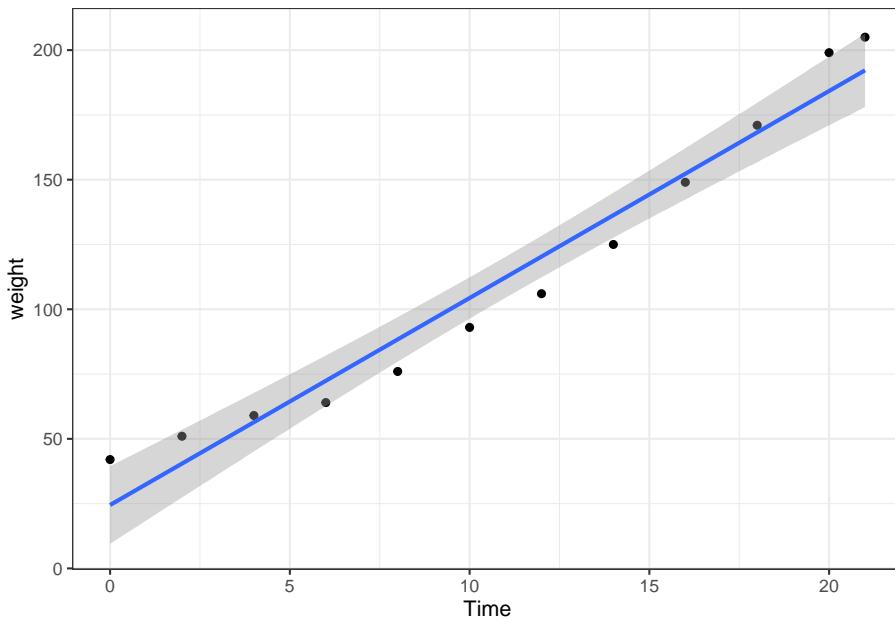
```
# metoda loess
p + geom_point() + stat_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



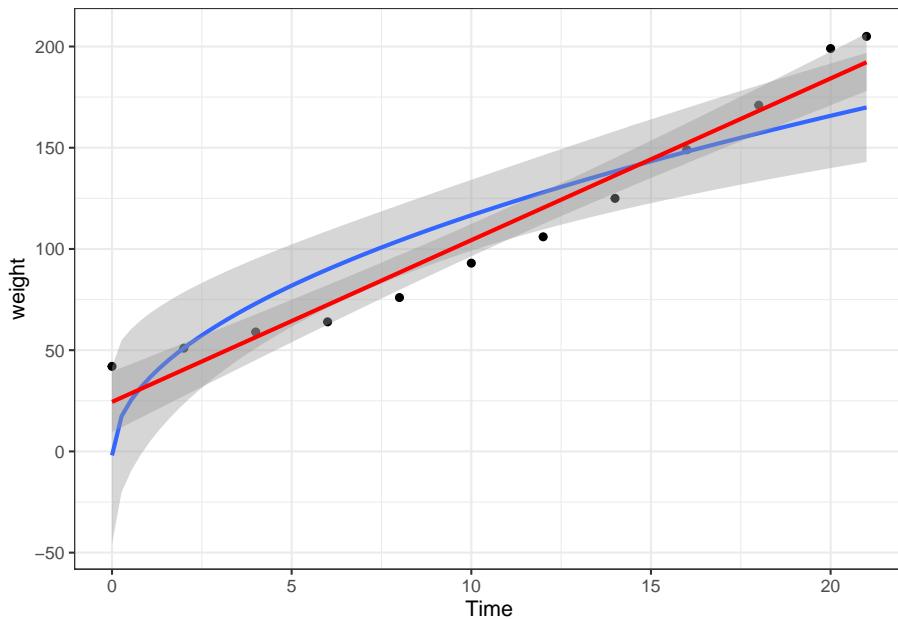
```
# metoda lm - dopasowanie do linii prostej
p + geom_point() + stat_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
# metoda lm z podaną formułą  
p + geom_point() + stat_smooth(method = "lm", formula = y~sqrt(x))+  
  stat_smooth(method = "lm", color = "red")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

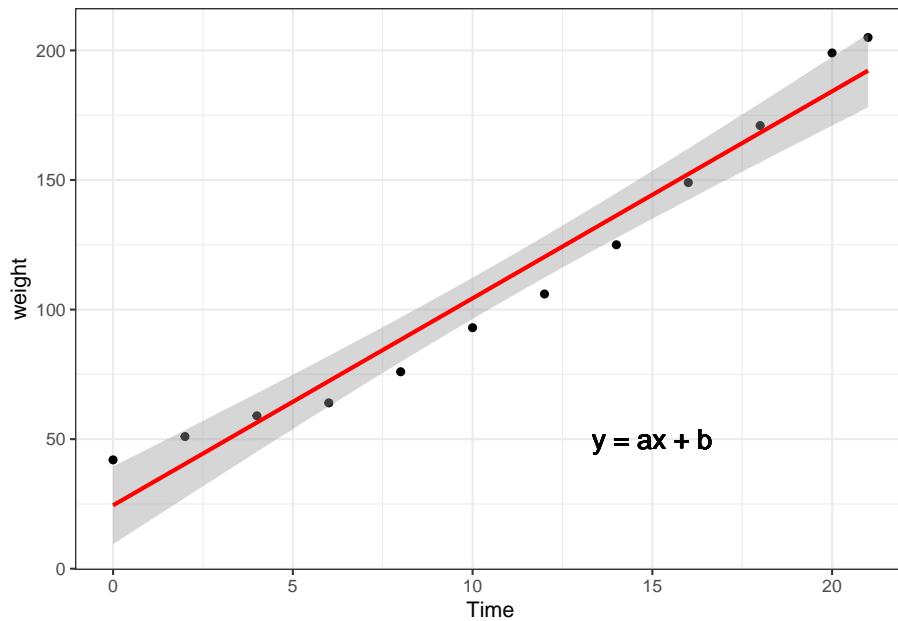


Sprawdzenie dopasowania można wykonać funkcją `lm`. Jej `summary` podaje wartość R^2 , istotność, parametry wzoru(`coefficients`).

```
summary(lm(weight ~ Time, data = chick))

##
## Call:
## lm(formula = weight ~ Time, data = chick)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -14.3202 -11.3081 - 0.3444  11.1162  17.5346 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 24.4654    6.7279   3.636  0.00456 ***
## Time        7.9879    0.5236  15.255 2.97e-08 ***
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
##
## Residual standard error: 12.29 on 10 degrees of freedom
## Multiple R-squared:  0.9588, Adjusted R-squared:  0.9547 
## F-statistic: 232.7 on 1 and 10 DF,  p-value: 2.974e-08
```

```
# dodanie tekstu do wykresu np. równanie opisujące linię trendu
p + geom_point() + stat_smooth(method = "lm", color = "red")+
  geom_text(data = NULL, x = 15, y = 50, label = "y = ax + b", size = 5) # x, y - położenie na wykresie
## `geom_smooth()` using formula 'y ~ x'
```

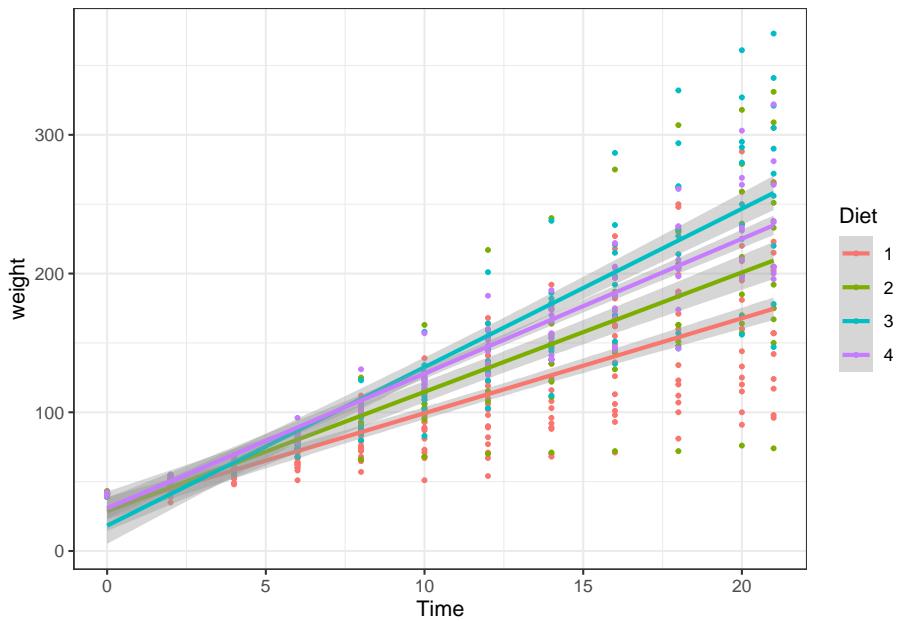


Możemy zastosować `stat_smooth` dla wszystkich danych.

`geom_point` można rozróżnić pod względem: `color`, `shape`, `size`, a `geom_line` pod względem: `color` i `linetype`.

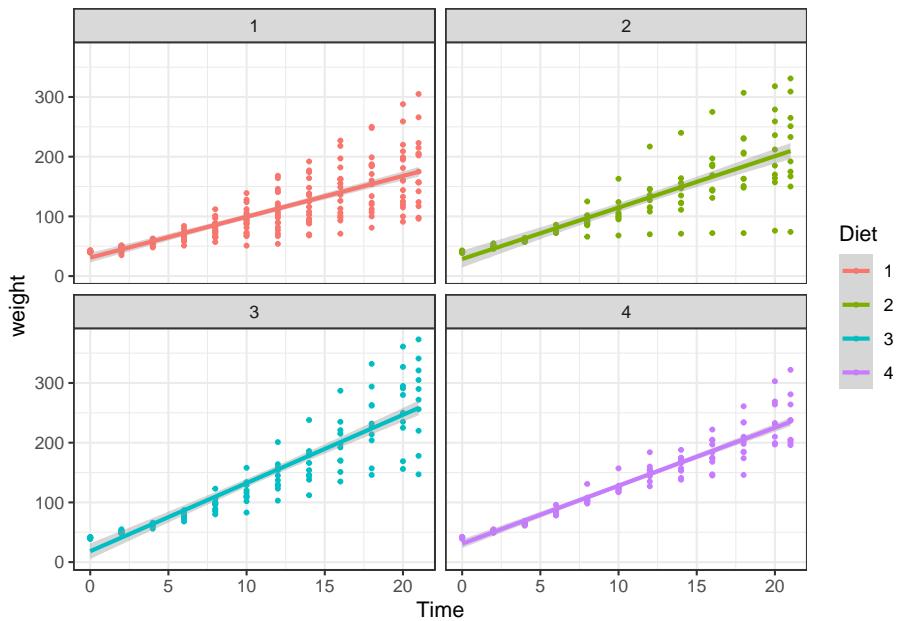
```
p <- ggplot(data = ChickWeight, aes(x = Time, y = weight, color = Diet))
p + geom_point(size = 0.75) + stat_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



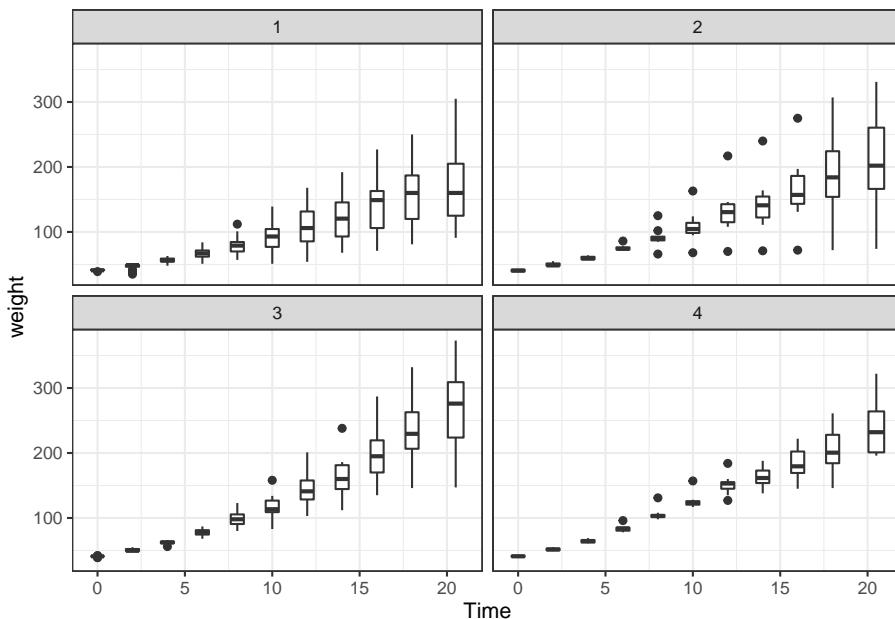
```
p + geom_point(size = 0.75) + stat_smooth(method = "lm") + facet_wrap(~Diet)

## `geom_smooth()` using formula 'y ~ x'
```



Do przedstawienia takich danych można też użyć boxplot.

```
p <- ggplot(data = ChickWeight, aes(x = Time, group = plyr::round_any(Time, 2, floor), y = weight)
p + geom_boxplot() + facet_wrap(~Diet)
```



5.5.2 Jak poradzić sobie z nadmiarem punktów na wykresie (overplotting)?

Jednym z problemów podczas stosowania wykresów punktowych może być zbyt duża liczba punktów powodująca zacieranie się informacji.

W `ggplot2` możemy skorzystać z kilku sposobów poradzenia sobie z “overplotting”. W przypadku danych ciągłych można wykorzystać parametr `alpha` pozwalający na ustawienie półprzezroczystych punktów albo wykorzystać punkty niewypełnione w środku - `shape = 1:14`.

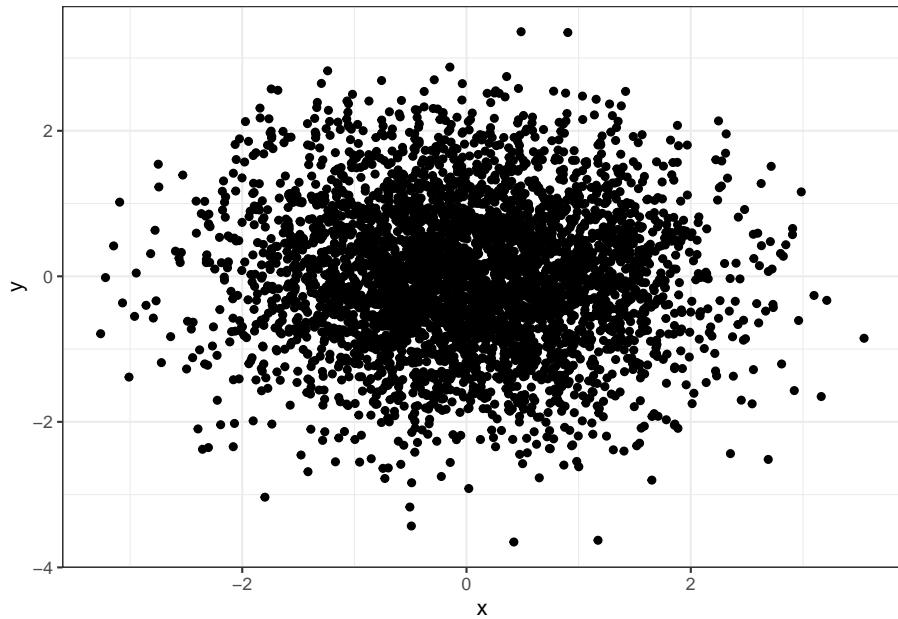
Innym sposobem może być wykorzystanie `geom_density2d`, który wylicza gęstość w układzie dwuwymiarowym i zaznacza liniami na wykresie albo `geom_bin2d`, który pozwala zrobić dwuwymiarowy histogram, możliwa kontrola `binwidth` (trzeba podać wektor dwóch wartości).

Dla danych dyskretnych pomóc może wykorzystanie `position="jitter"`, która pozwala losowo rozrzucić punkty albo zastosowanie parametru `alpha` oraz `stat_sum` - wielkość punktów zależy od ilości nakładających się punktów.

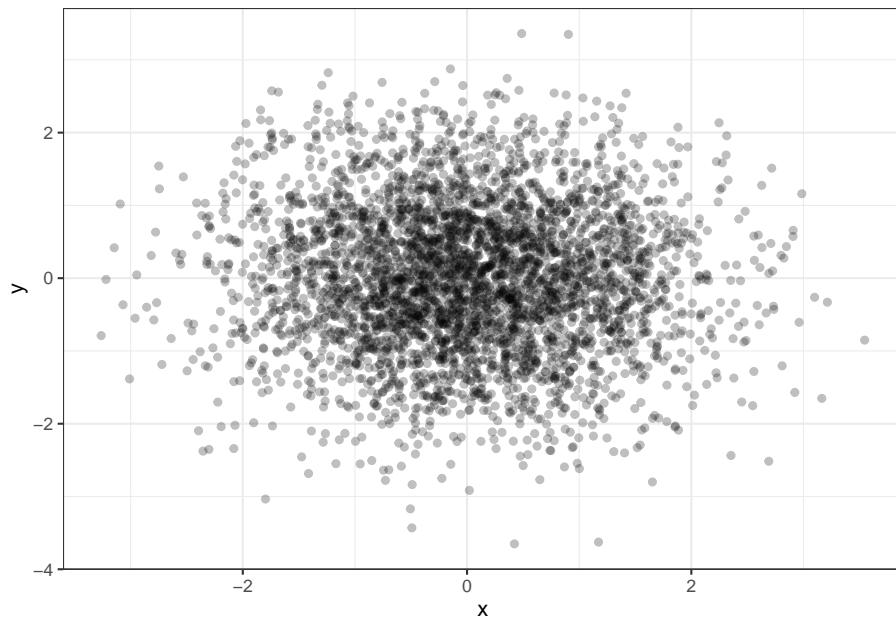
```
# przykładowe dane ciągłe

dane <- data.frame(x = rnorm(4000), y = rnorm(4000))

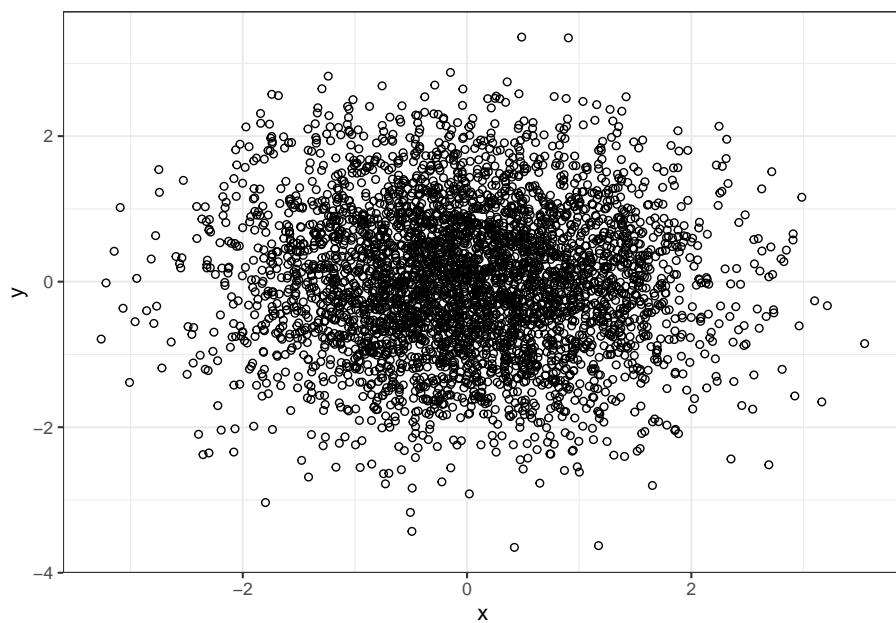
p <- ggplot(dane, aes(x = x, y = y))
p + geom_point()
```



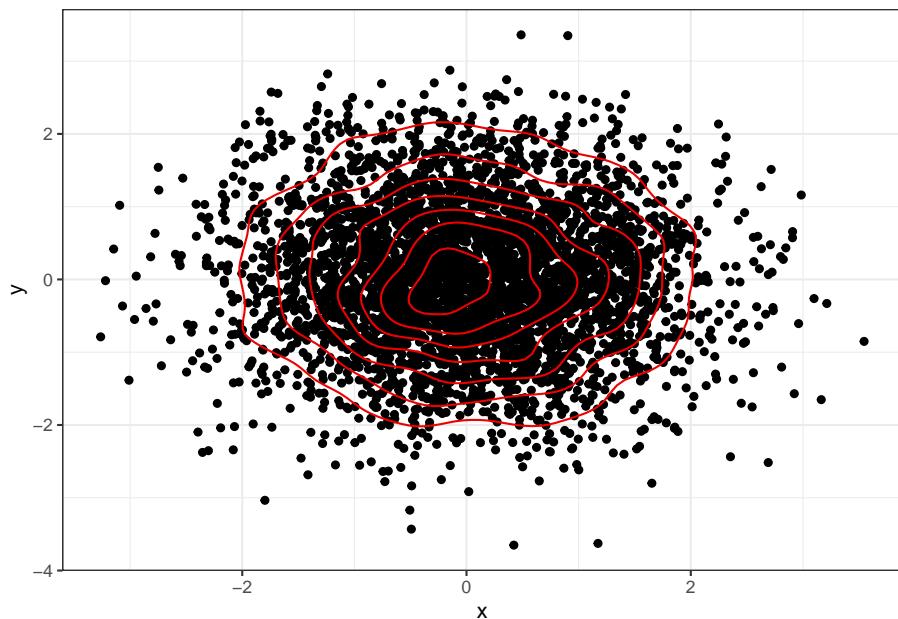
```
# zastosowanie alpha
p + geom_point(alpha = 0.25)
```



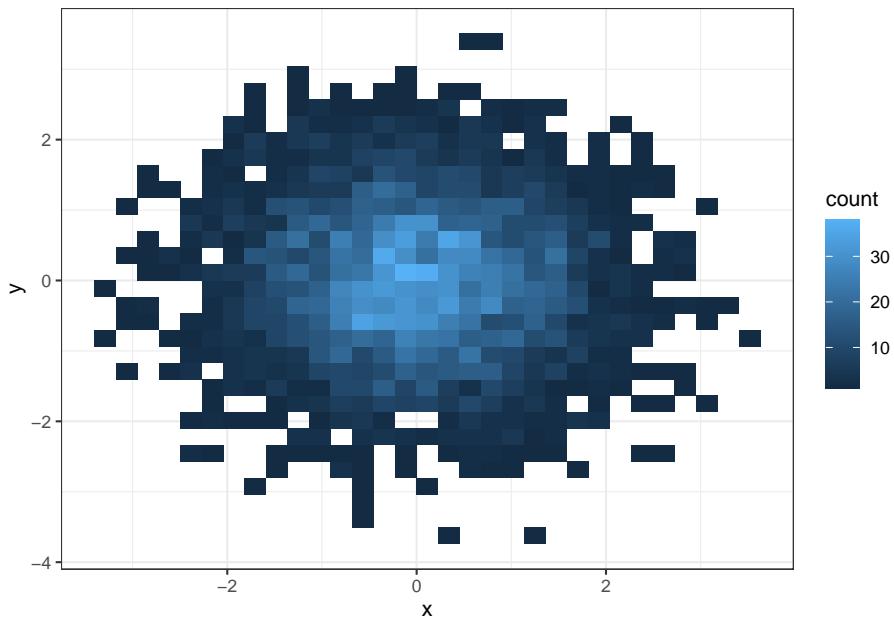
```
# puste punkty  
p + geom_point(shape = 1)
```



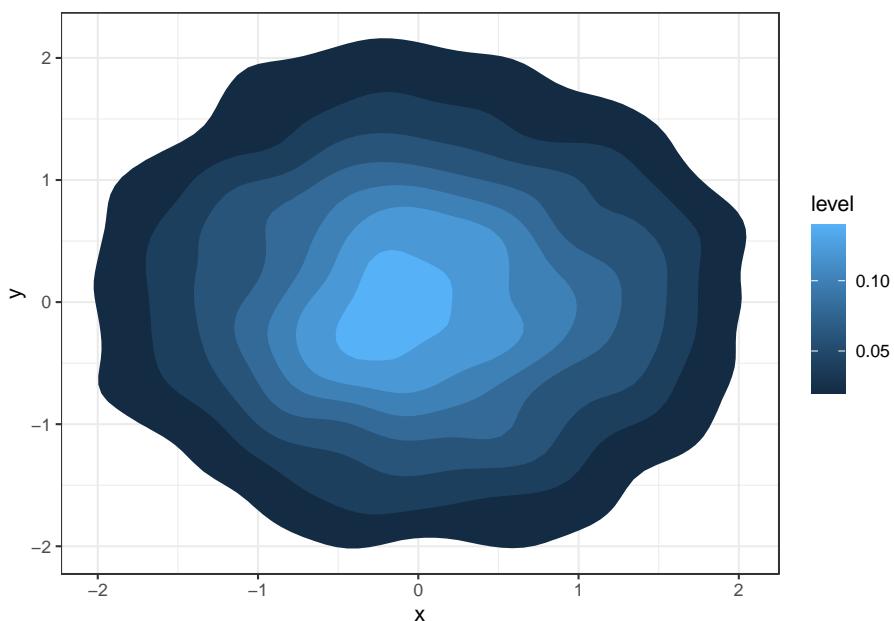
```
# geom_density2d also geom_bin2d()
p + geom_point() + geom_density2d(color = "red2")
```



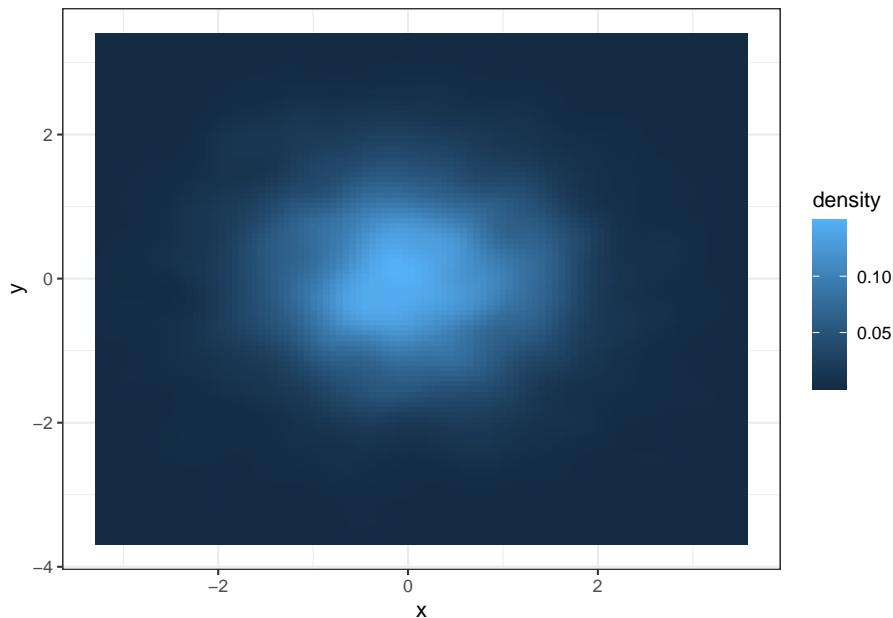
```
p + geom_bin2d()
```



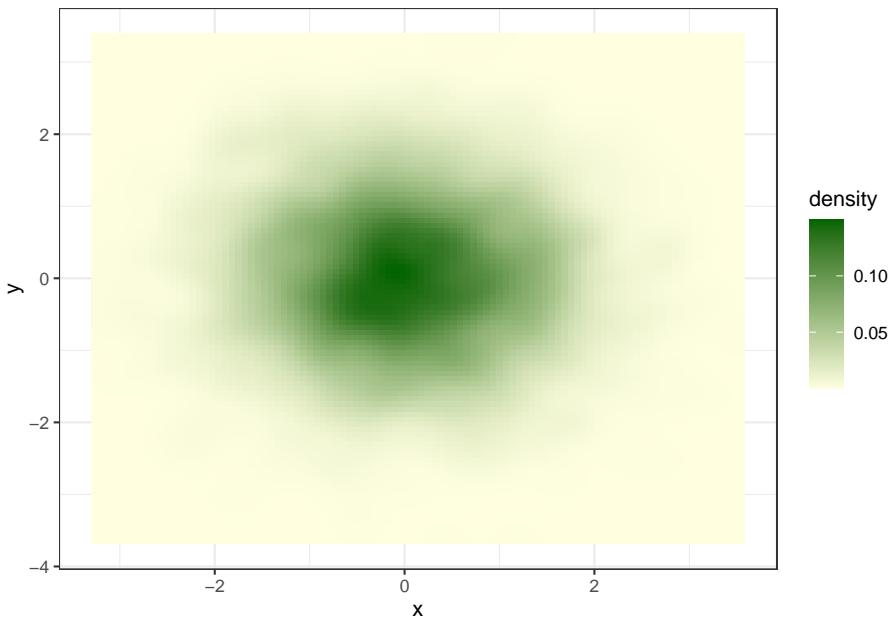
```
# można również użyć stat_density2d z innym geomem niż linia  
p + stat_density2d(geom = "polygon", aes(fill = ..level..))
```



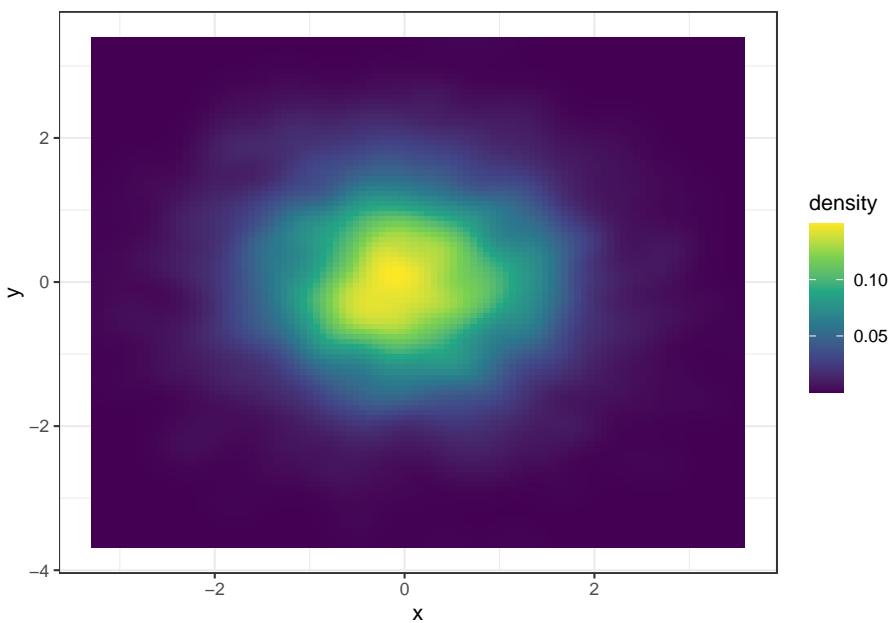
```
# linie widoczne na wykresach są wynikiem konwersji do pdf, nie będzie ich na wykresach
p + stat_density2d(geom = "tile", contour = FALSE, aes(fill = ..density..))
```



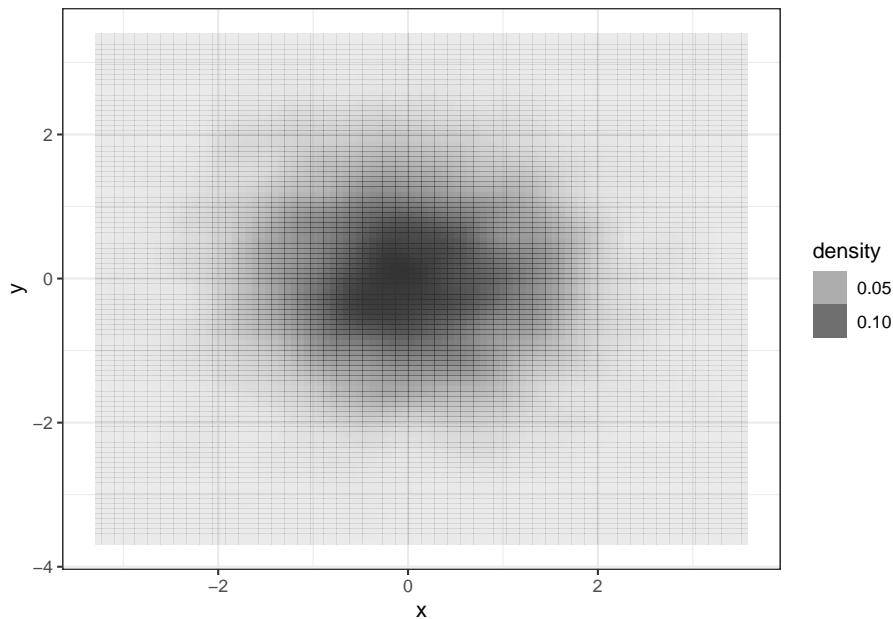
```
p + stat_density2d(geom = "tile", contour = FALSE, aes(fill = ..density..))+
  scale_fill_gradient(low = "lightyellow", high = "darkgreen")
```



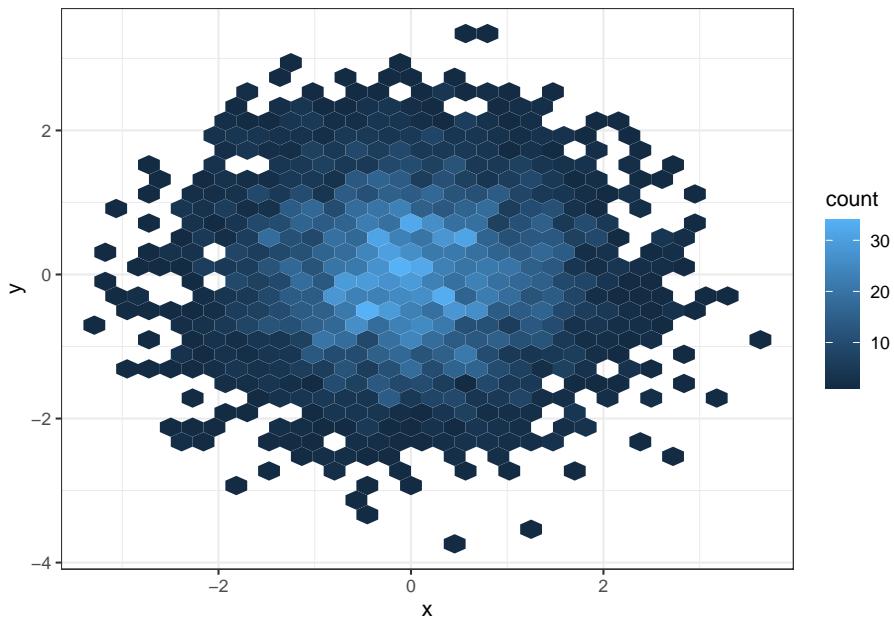
```
p + stat_density2d(geom = "tile", contour = FALSE, aes(fill = ..density..))+  
  scale_fill_viridis_c()
```



```
p + stat_density2d(geom = "tile", contour = FALSE, aes(alpha = ..density..))
```

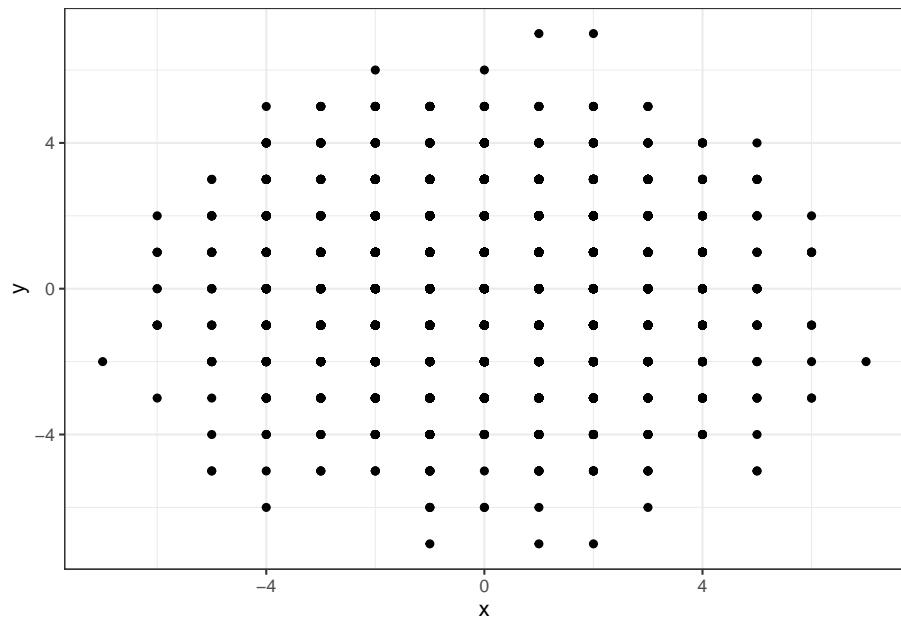


```
# zamiast kwadratów można użyć sześciokątów, wymaga zainstalowanie pakietu hexbin  
library(hexbin)  
p + stat_binhex()
```

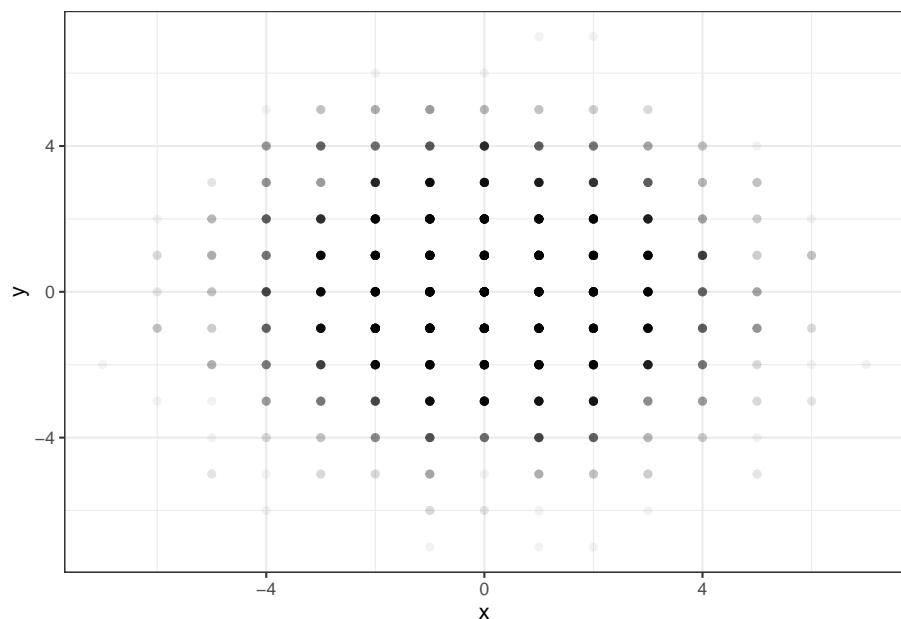


```
# dane dyskretne
dane <- round(2*dane)

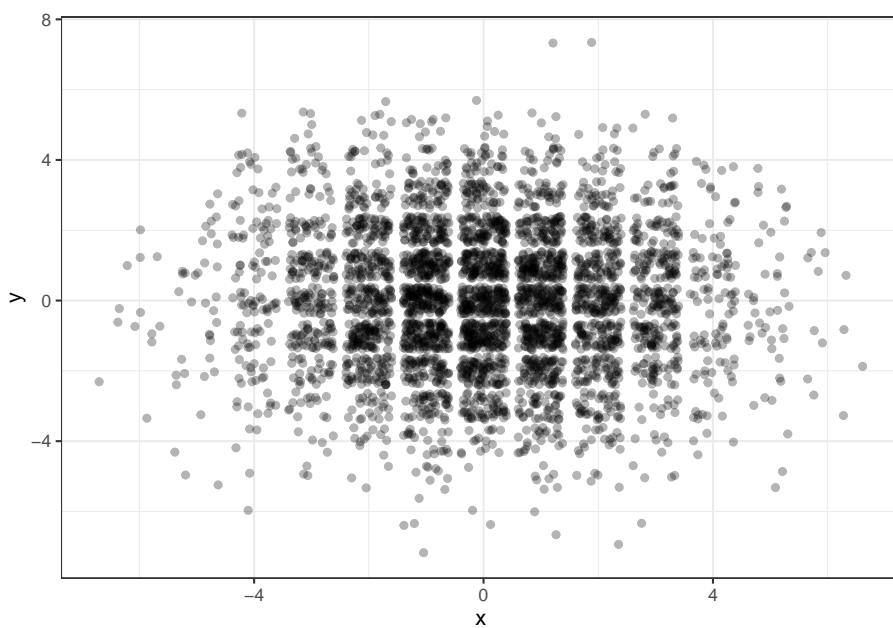
p <- ggplot(dane, aes(x = x,y = y))
p + geom_point()
```



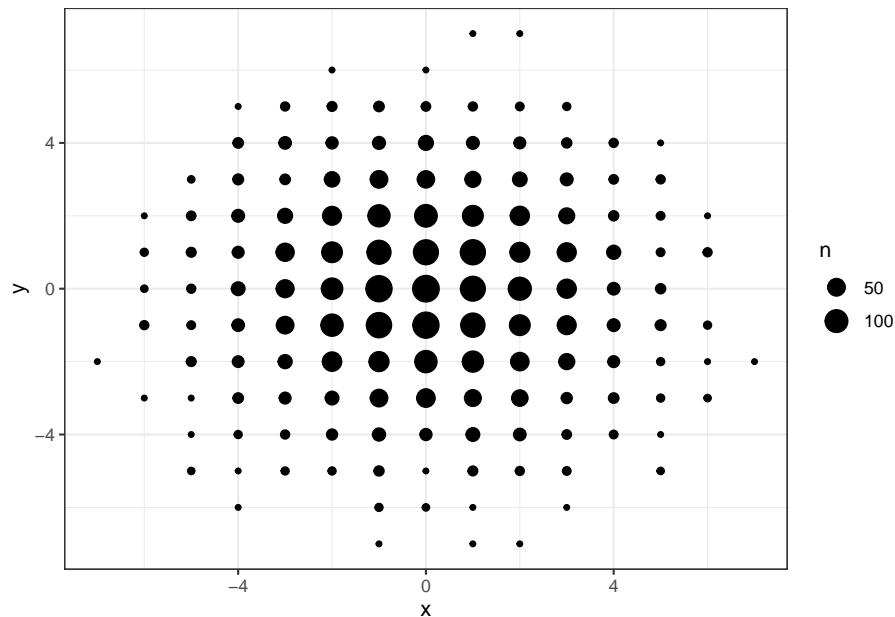
```
p + geom_point(alpha = 0.05)
```



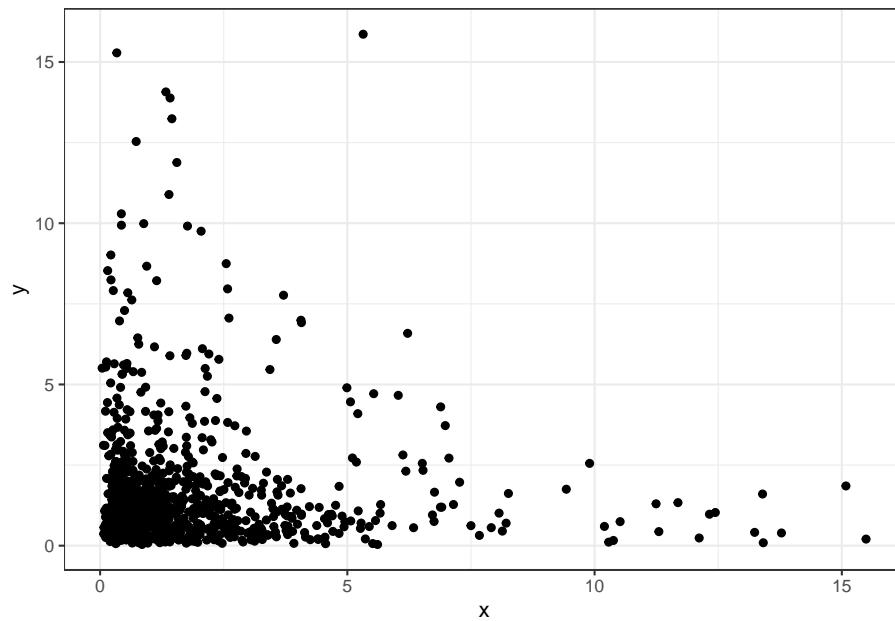
```
# wykorzystanie position = "jitter" i alpha  
p + geom_point(position = "jitter", alpha = 0.3)
```



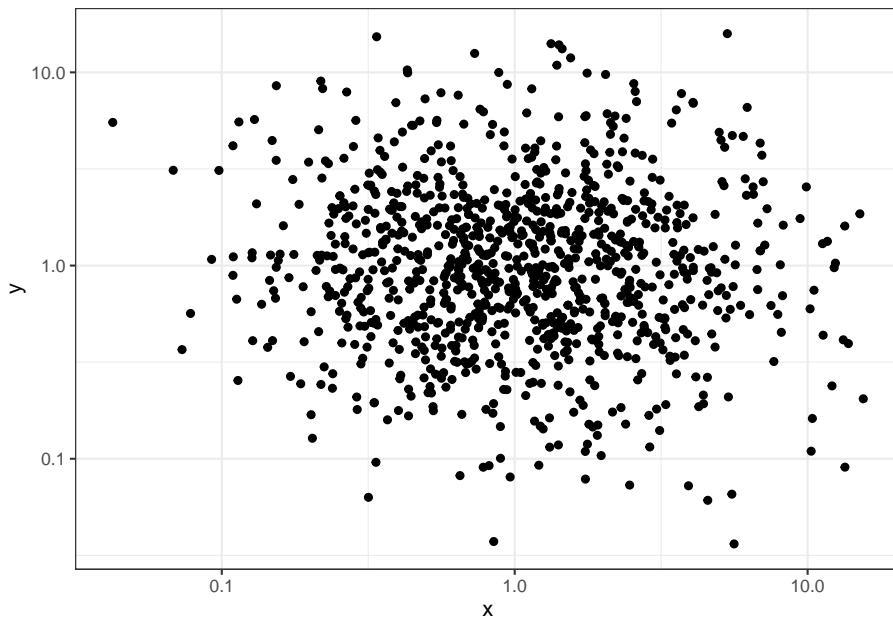
```
# z wykorzystaniem stat_sum  
p + stat_sum()
```



```
# czasem pomocą może też zmiana skali na logarytmiczną
dane <- data.frame(x = rlnorm(1000), y = rlnorm(1000))
p <- ggplot(dane, aes(x = x, y = y))
p + geom_point()
```



```
p + geom_point() + scale_y_log10() + scale_x_log10()
```



5.5.3 Wykres wstążka (ribbon)

Użycie `geom_ribbon` wymaga wstępnie podsumowania danych np. przy użyciu `dplyr`. W tym wypadku zmiennymi, pod względem których będziemy dzielić dane na grupy będą Diet i Time

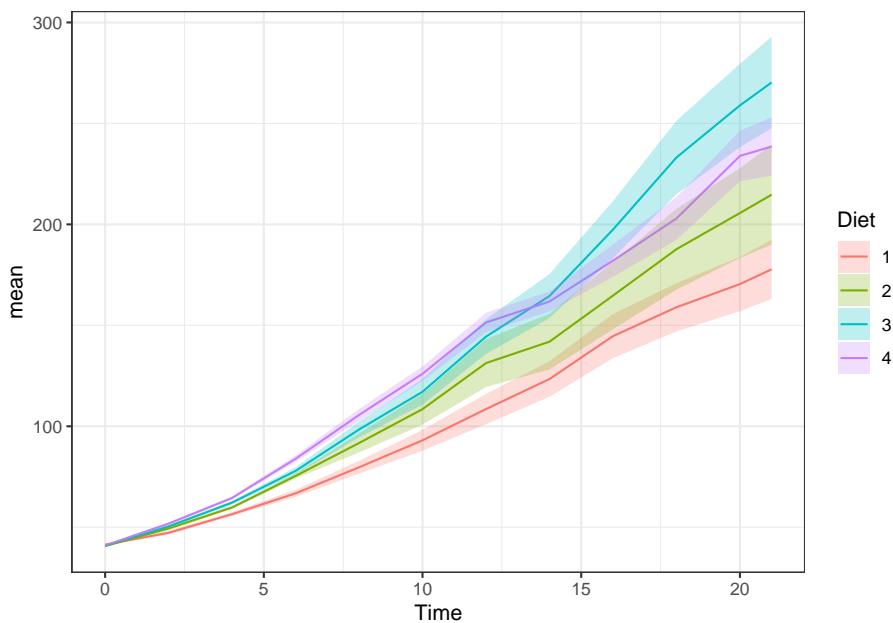
Półprzezroczystą wstążkę uzyskujemy dzięki parametrowi `alpha`.

```
library(dplyr)

summ <- ChickWeight %>% group_by(Diet, Time) %>%
  summarise(mean = mean(weight),
            sd = sd(weight),
            blad = sd/sqrt(length(weight)),
            lower = mean-blad,
            upper = mean+blad)
```

```
## `summarise()` has grouped output by 'Diet'. You can override using the `groups` argument.
```

```
p <- ggplot(data = summ, aes(x = Time, y = mean))
p + geom_line(aes(color = Diet)) +
  geom_ribbon(aes(ymin = lower, ymax = upper, fill = Diet), alpha = 0.25)
```



5.6 Kolor, osie, panele

5.6.1 Porównywanie - fill, color, shape

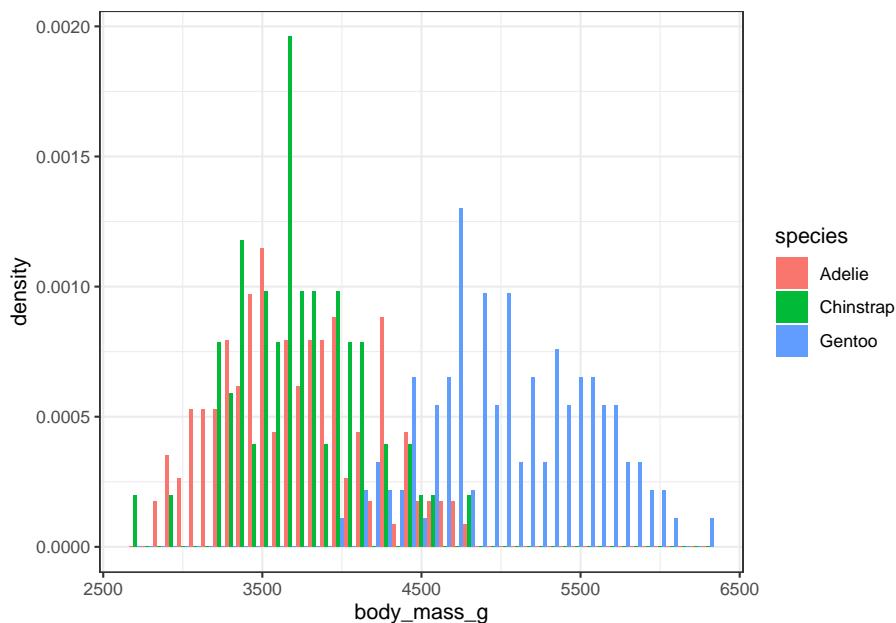
Porównanie kilku szczepów można zrobić w ten sam sposób, wybierając w `aes` - `fill=species` albo `color=species` (w zależności od geomu, fill pasuje do histogram i density, color do density, line, point, boxplot itd.)

W histogramie domyślnie dostaniemy słupki ustawione jeden na drugim, jeżeli chcemy słupki ustawione obok siebie należy ustawić parametr `position="dodge"`.

W przypadku boxplot dla kilku cech można pod x podstawić Szczep - otrzymamy wykres zawierający po jednym boxplotie dla jednego szczepu.

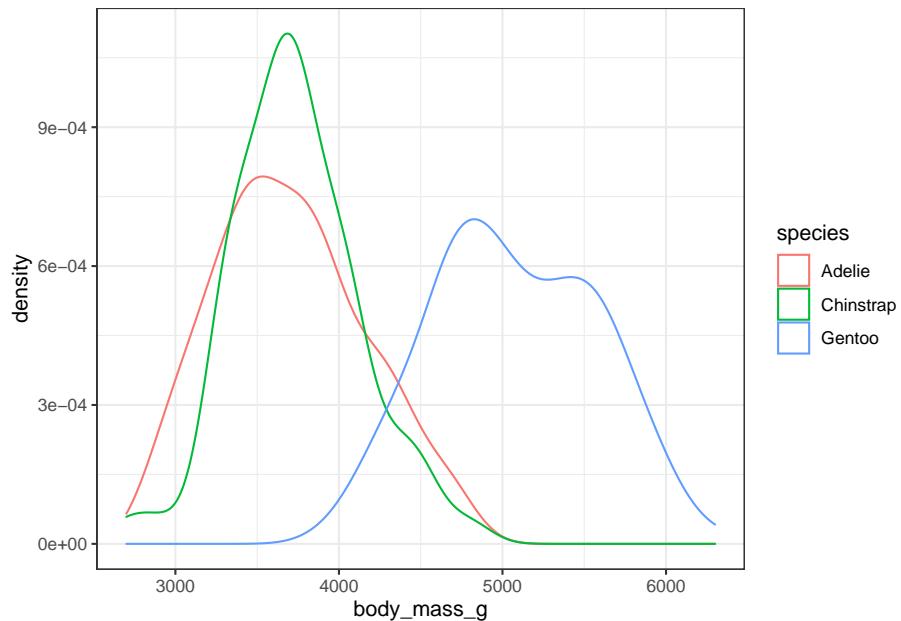
```
p <- ggplot(data = penguins, aes(x = body_mass_g))
# Histogram dla trzech szczepów
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75,
                    position="dodge")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



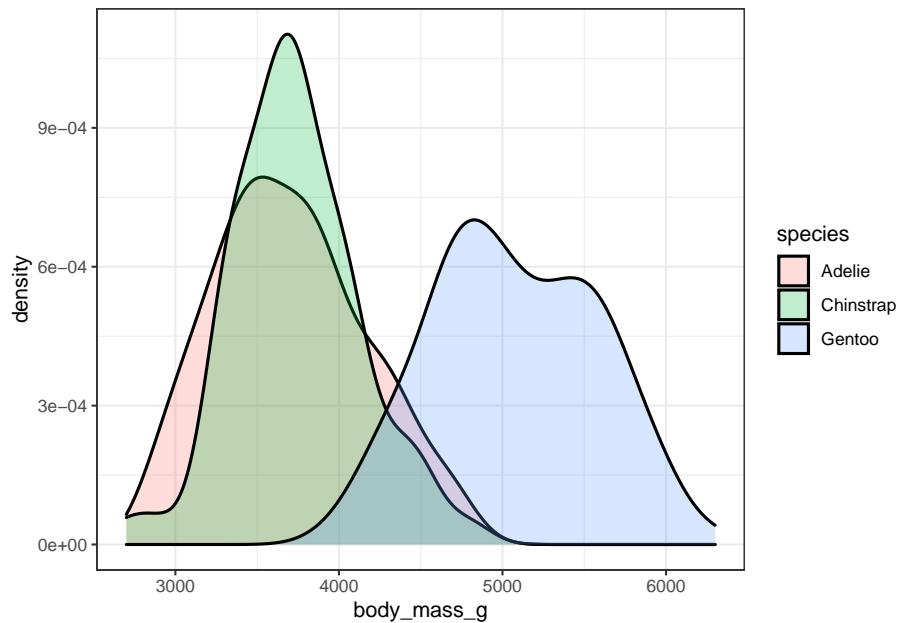
```
# Gęstość dla trzech szczepów rozróżniona przez color  
p + geom_density(aes(color = species))
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density).
```



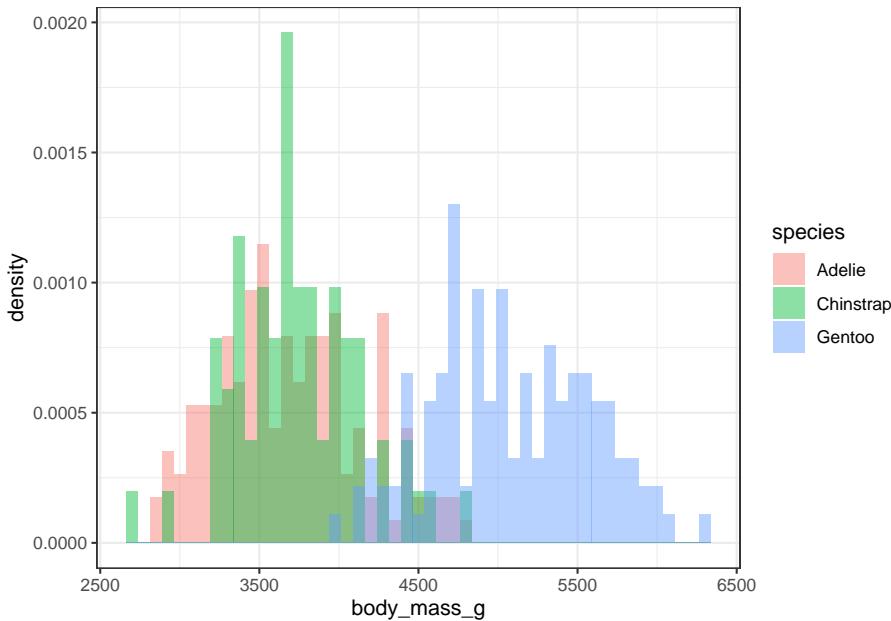
```
# Gęstość rozróżniona przez fill, półprzezroczystość uzyskujemy parametrem alpha
p + geom_density(aes(fill = species), alpha = 0.25, size = 0.75)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density).
```

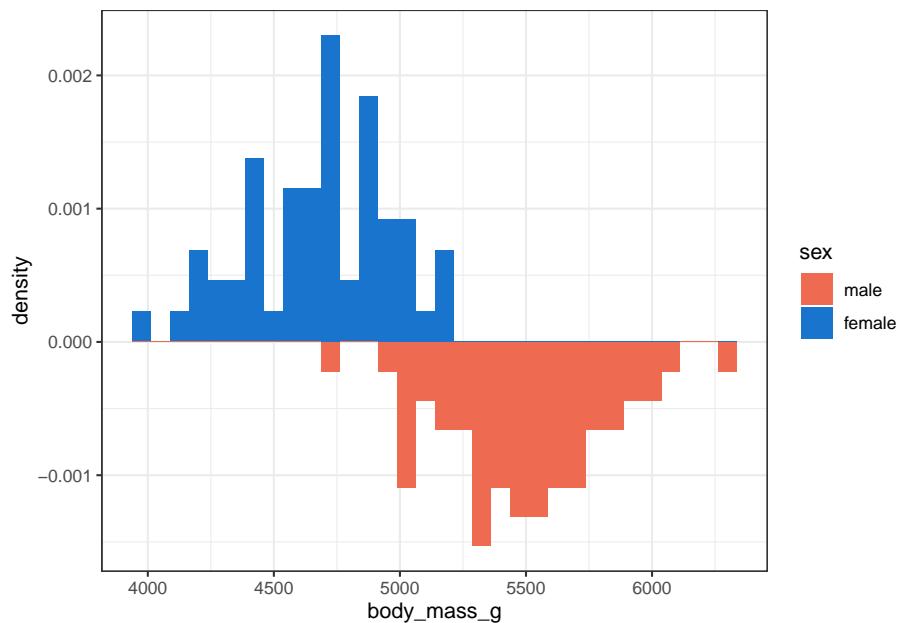


```
# Histogramy nałożone na siebie - wymaga ustawienia position = "identity"
p <- ggplot(data = penguins, aes(x = body_mass_g))
p + geom_histogram(aes(y = ..density.., fill = species), binwidth = 75, alpha = 0.45, position =
```

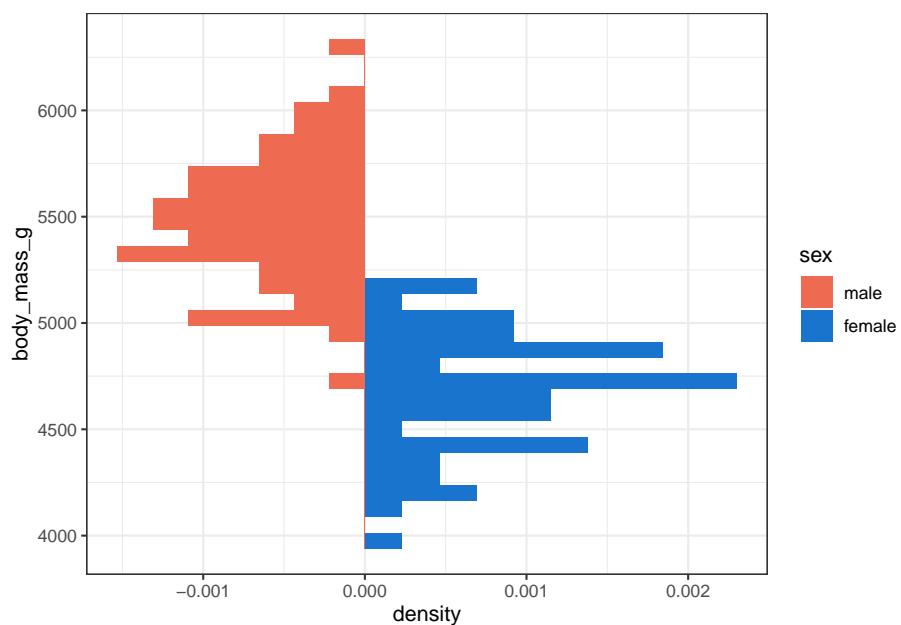
```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



```
# Histogram "back to back" wymaga ustawienia w drugim geom_histogram ujemnego y
p <- ggplot(penguins %>% filter(species == 'Gentoo', sex == 'female'),
            aes(x = body_mass_g))
p <- p + geom_histogram(aes(y = ..density.., fill = "dodgerblue3"), binwidth = 75) +
  geom_histogram(data = penguins %>% filter(species == 'Gentoo', sex == 'male'),
                aes(x = body_mass_g, y = -..density.., fill = "coral2"),
                binwidth = 75) +
  scale_fill_manual(name = "sex", # ręcznie ustawiona legenda
                    values = c("coral2" = "coral2", "dodgerblue3" = "dodgerblue3"),
                    labels = c("male", "female"))
p
```

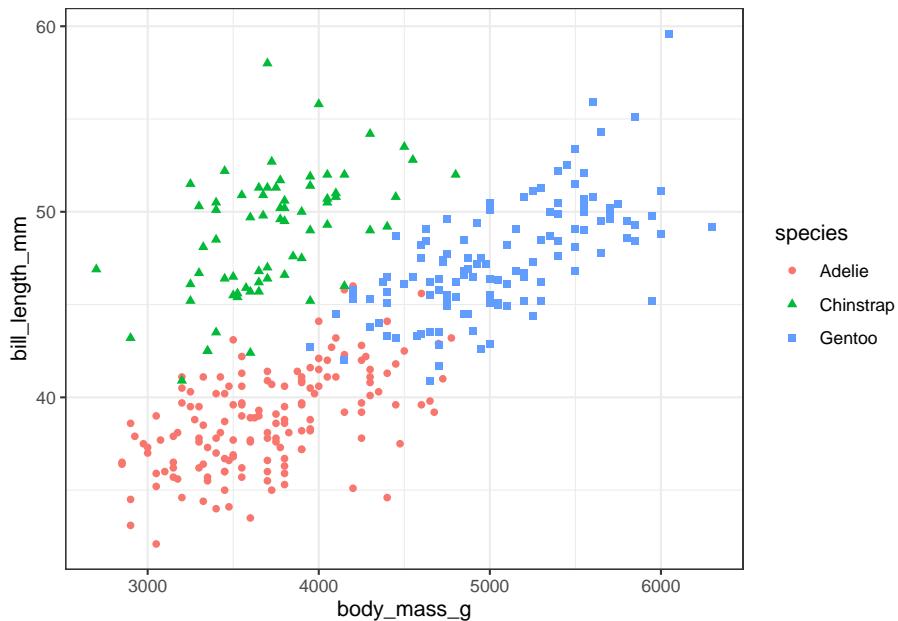


```
p + coord_flip()
```



```
# shape i linetype
p <- ggplot(penguins, aes(x = body_mass_g, y = bill_length_mm, color = species, shape = species))
p + geom_point()
```

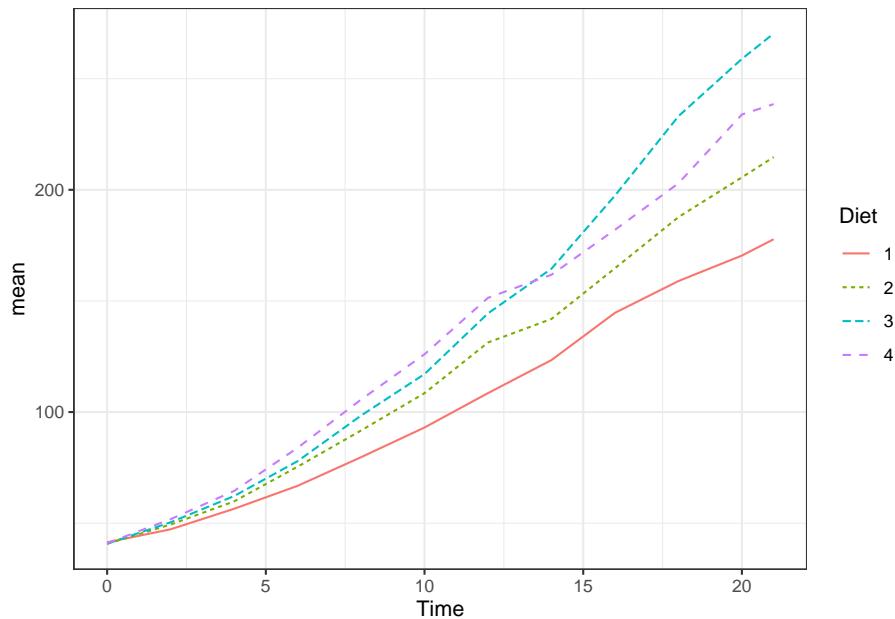
Warning: Removed 2 rows containing missing values (geom_point).



```
summ <- ChickWeight %>% group_by(Diet, Time) %>%
  summarise(mean = mean(weight))
```

`summarise()` has grouped output by 'Diet'. You can override using the `~.groups` argument.

```
p <- ggplot(data = summ, aes(x = Time, y = mean))
p + geom_line(aes(color = Diet, linetype = Diet))
```



5.6.2 Zmiana skali kolorów

Możemy zmienić domyślne kolory wykresu korzystając z funkcji `scale_colour\fill_sth`. Rodzaj funkcji zależy od rodzaju danych - ilościowe albo jakościowe.

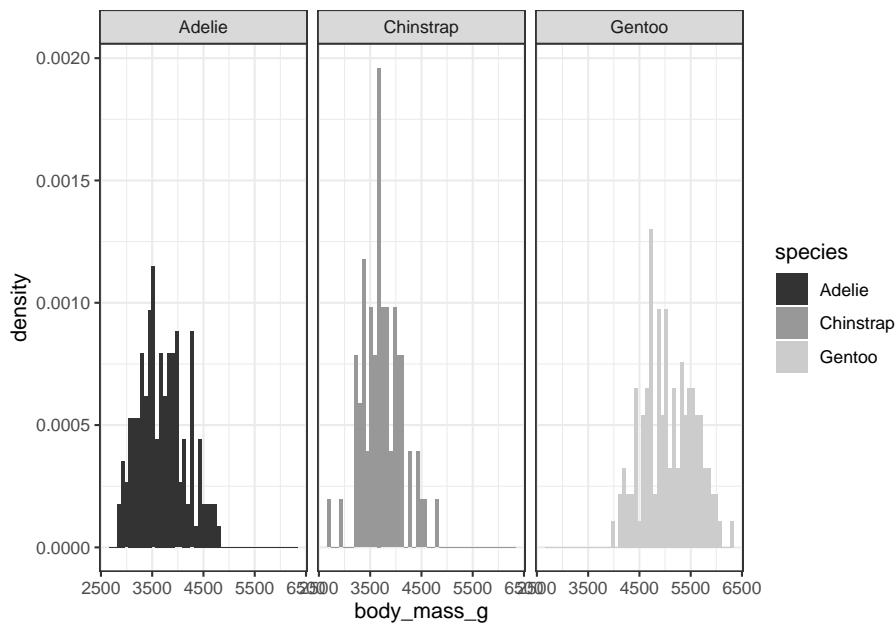
Dla danych jakościowych można użyć: `scale_color_grey` - odcień szarości, `scale_colour_brewer` - zawiera zestawy kolorów ze strony Color Brewer, `scale_color_viridis_d` - zawiera palety viridis, `scale_color_hue` - pozwala na wybranie kolorów korzystając z palety HCL. Można też ustawić własny zestaw kolorów korzystając z nazw kolorów R albo np. przez RGB, korzystając z `scale_colour_manual`

Dla danych ilościowych można wybrać: `scale_color_gradient` - gradient między dwoma kolorami, `scale_color_gradientn` - gradient pomiędzy 3 kolorami, `scale_color_gradientn` - gradient pomiędzy dowolną liczbą kolorów albo `scale_color_viridis_c`

```
# Zmienne jakościowe

# Odcienie szarości
p <- ggplot(data = penguins, aes(x = body_mass_g))
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_grey()

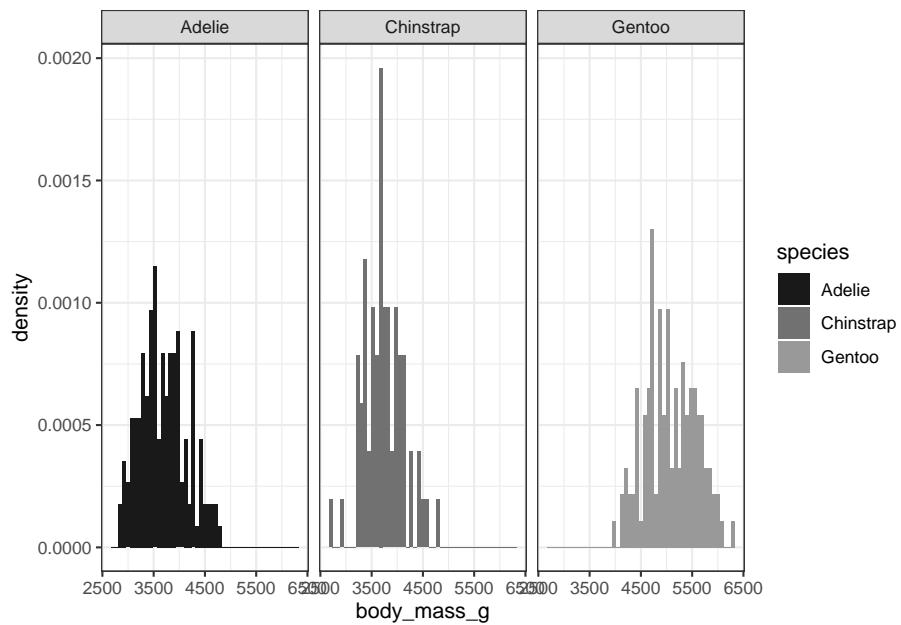
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



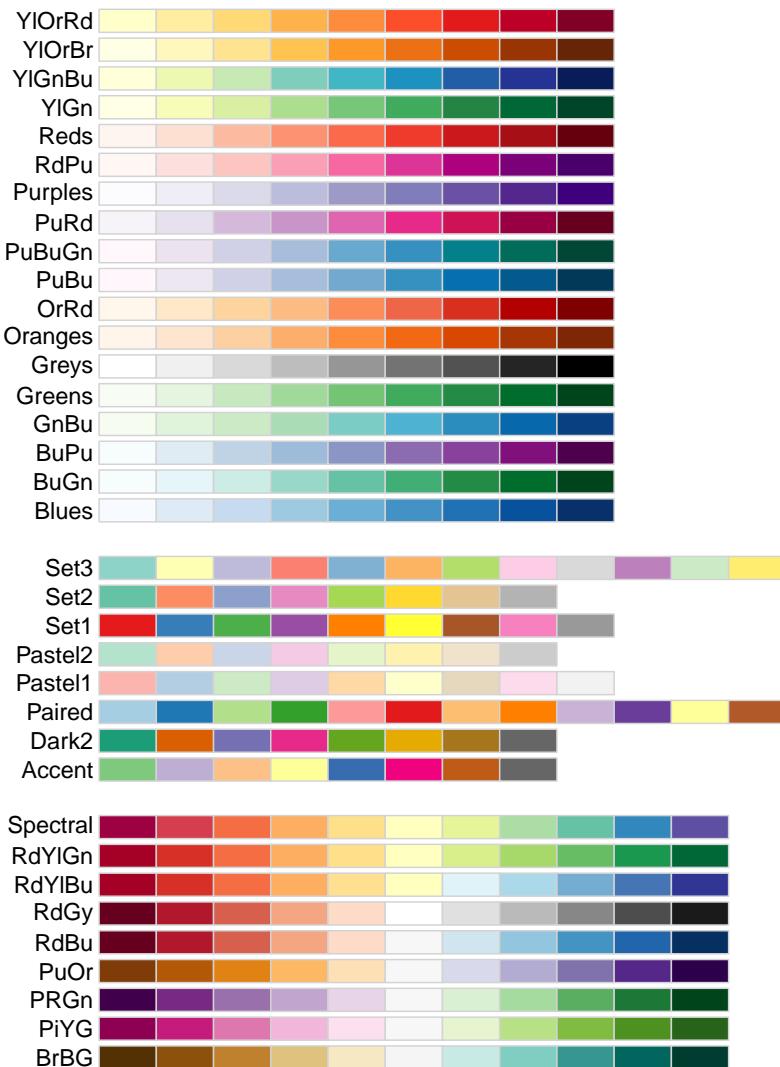
```
# można wybrać początek i koniec skali
```

```
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_grey(start = 0.1, end = 0.6)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



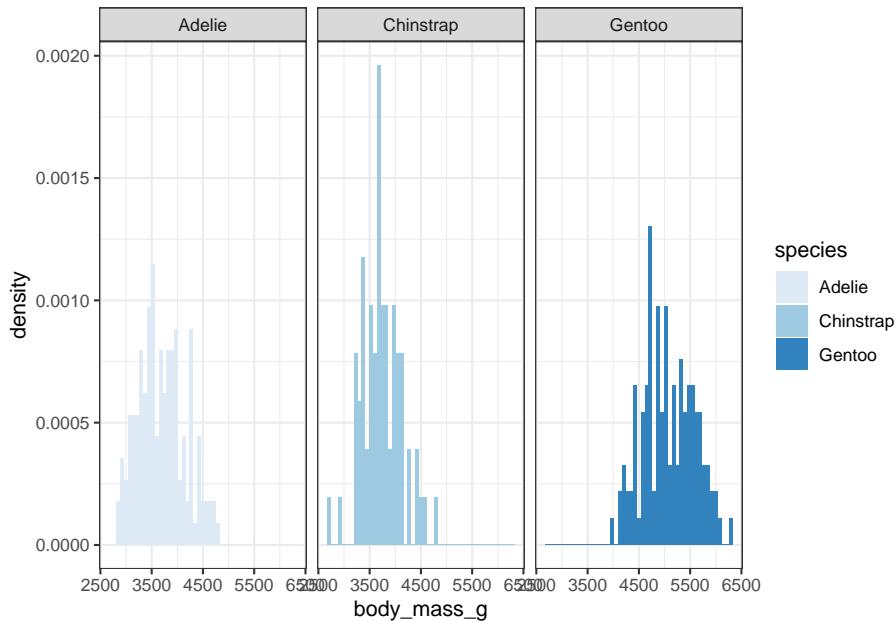
```
# skala ColorBrewer
library(RColorBrewer)
display.brewer.all()
```



```
# z użyciem ColorBrewer

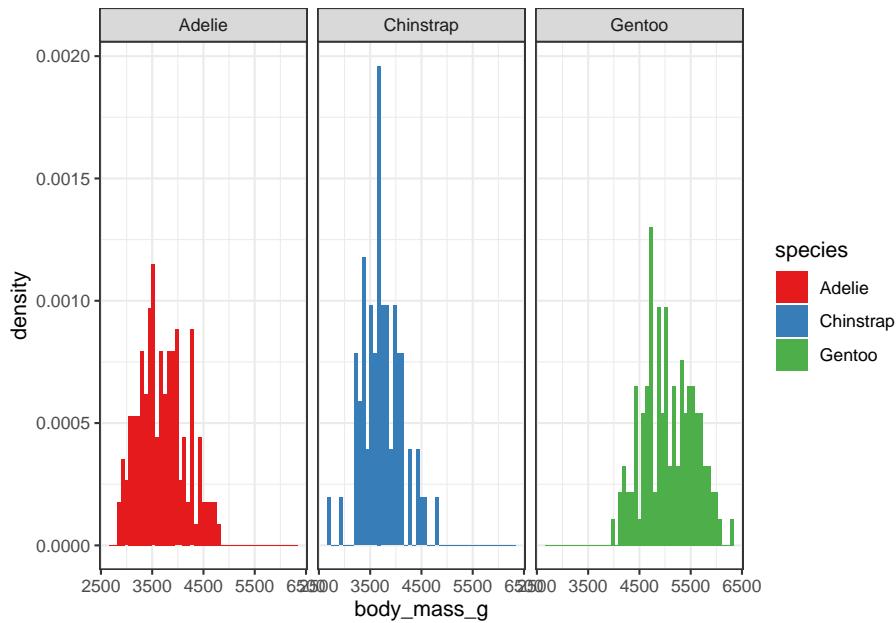
p <- ggplot(data = penguins, aes(x = body_mass_g))
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_brewer()
```

Warning: Removed 2 rows containing non-finite values (stat_bin).



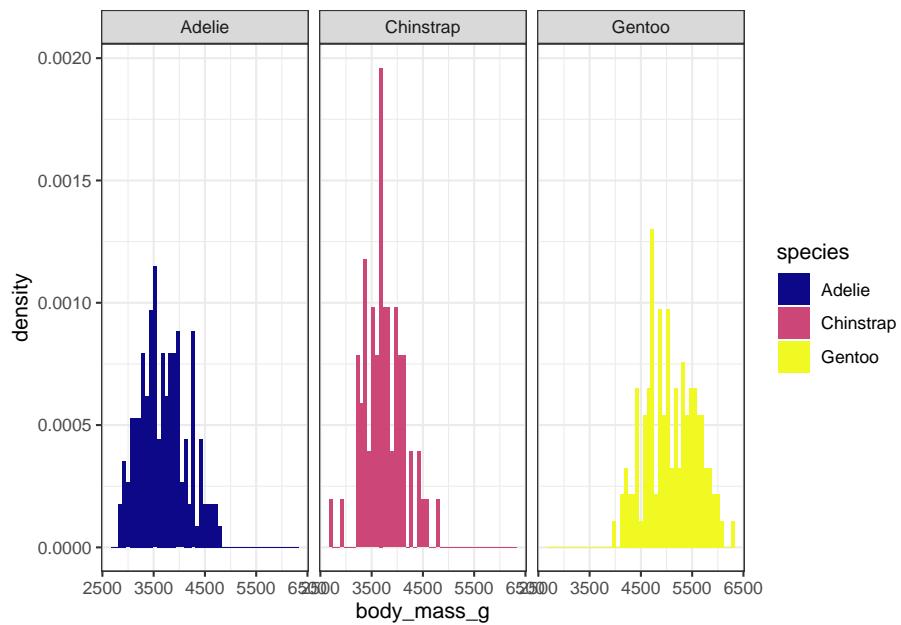
```
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_brewer(palette = "Set1")
```

Warning: Removed 2 rows containing non-finite values (stat_bin).



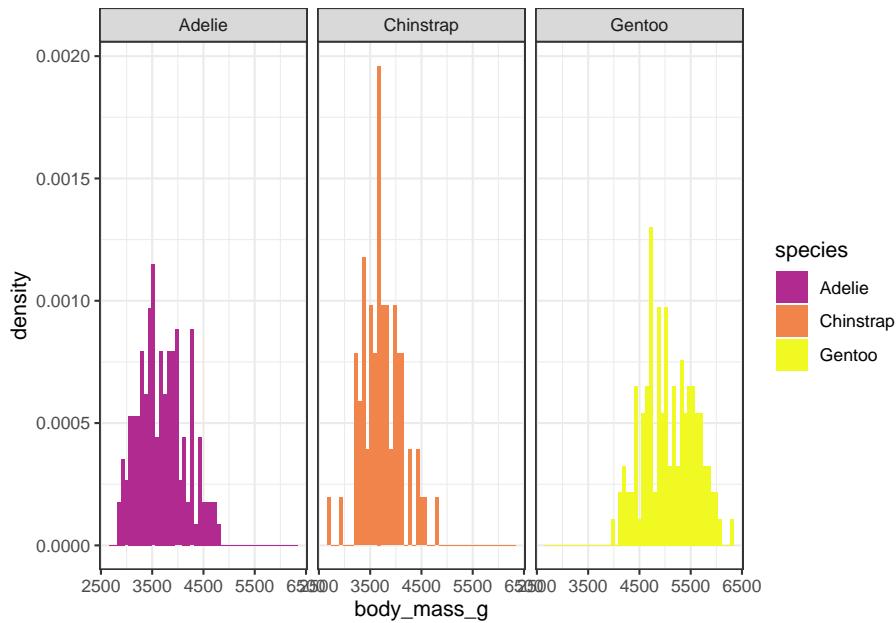
```
# z użyciem viridis, argument option przyjmuje wartości od A do H
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_viridis_d(option = 'C')
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



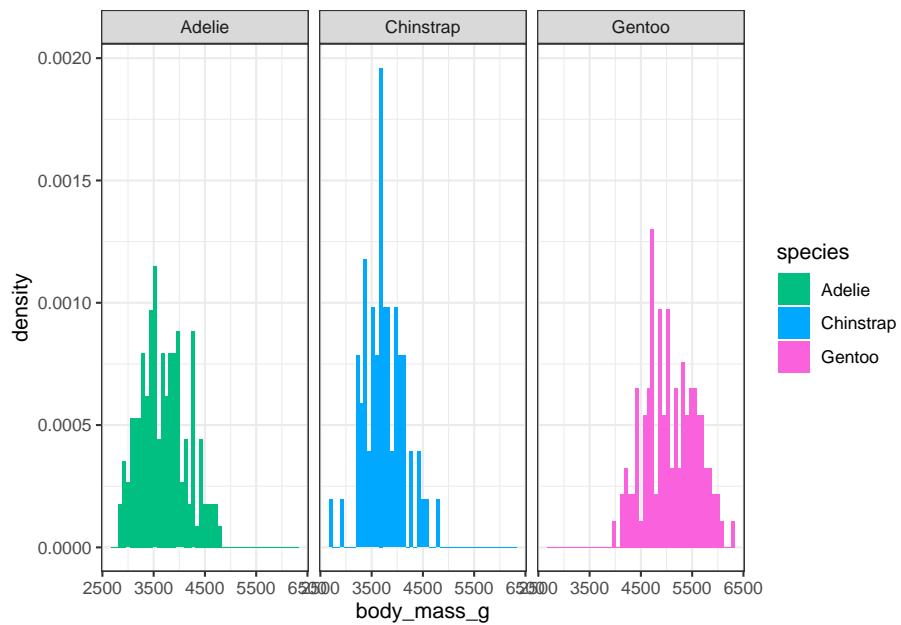
```
# zakres palety można modyfikować przy pomocy argumentów begin i end
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_viridis_d(option = 'C', begin = 0.4)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



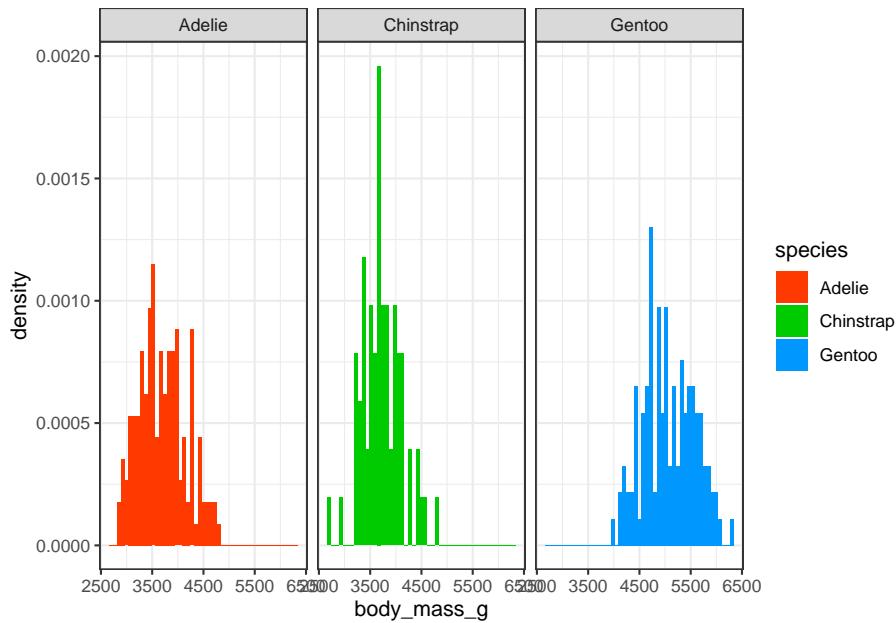
```
# skala z użyciem palety HCL - ustawiamy trzy argumenty: h(zakres barw), c(intensywność) i l(jasność)
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_hue(h = c(160,320))

## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



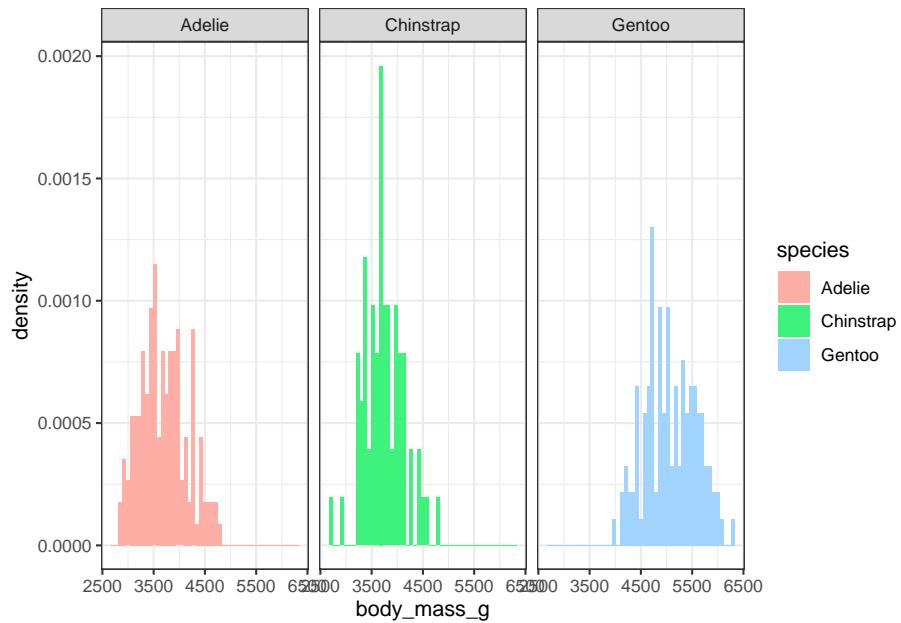
```
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_hue(c = 200)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```

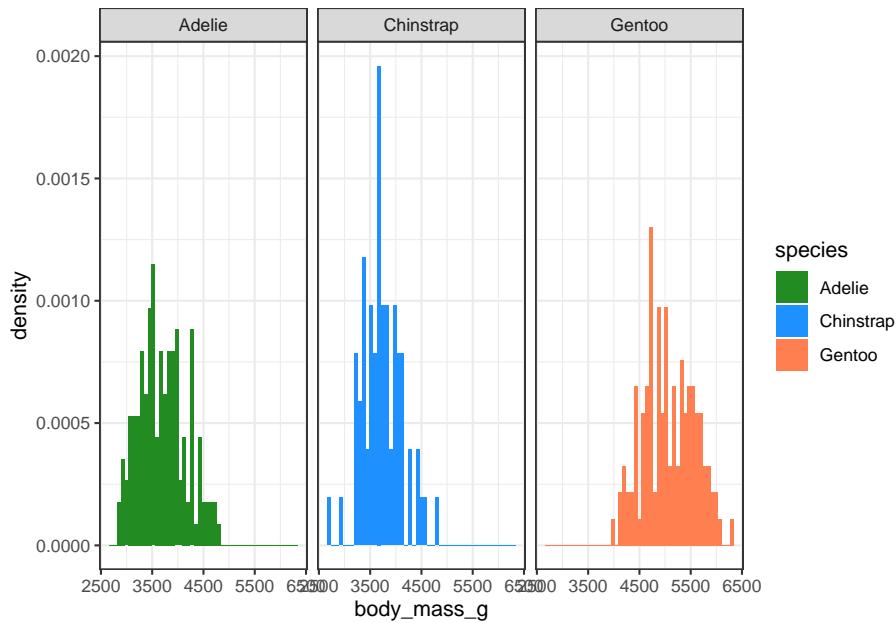


```
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75)+  
  facet_wrap(~species) + scale_fill_hue(l = 85)
```

```
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



```
# skala manualna
p + geom_histogram(aes(fill = species, y = ..density..), binwidth = 75) +
  facet_wrap(~species) + scale_fill_manual(values = c("forestgreen", "dodgerblue", "coral"))
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```

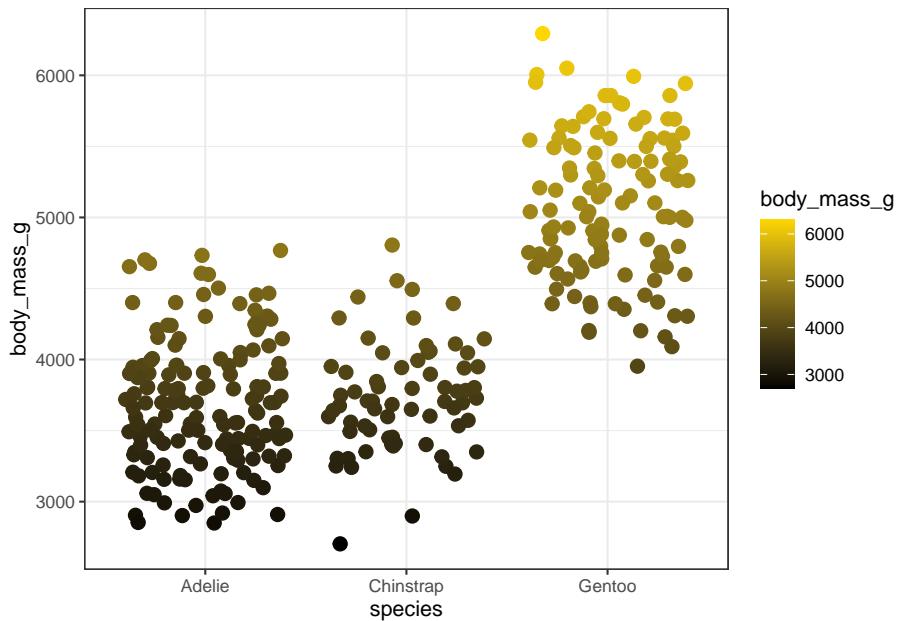


```
# Zmienne ilościowe

# domyślny gradient
p <- ggplot(penguins)
p + geom_jitter(aes(x = species, y = body_mass_g, color = body_mass_g), size = 3)

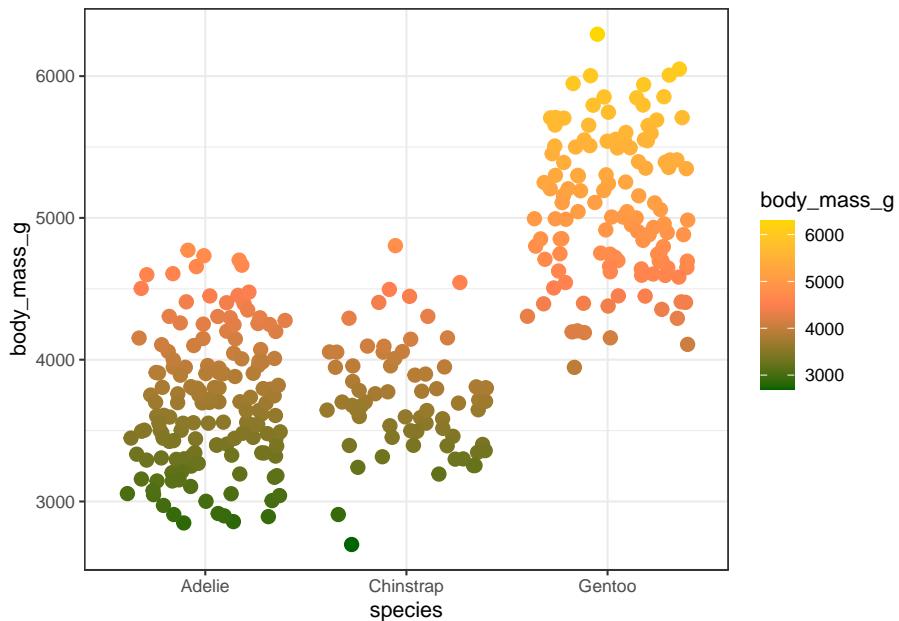
# dwa kolory
p + scale_color_gradient(low = "black", high = "gold")

## Warning: Removed 2 rows containing missing values (geom_point).
```



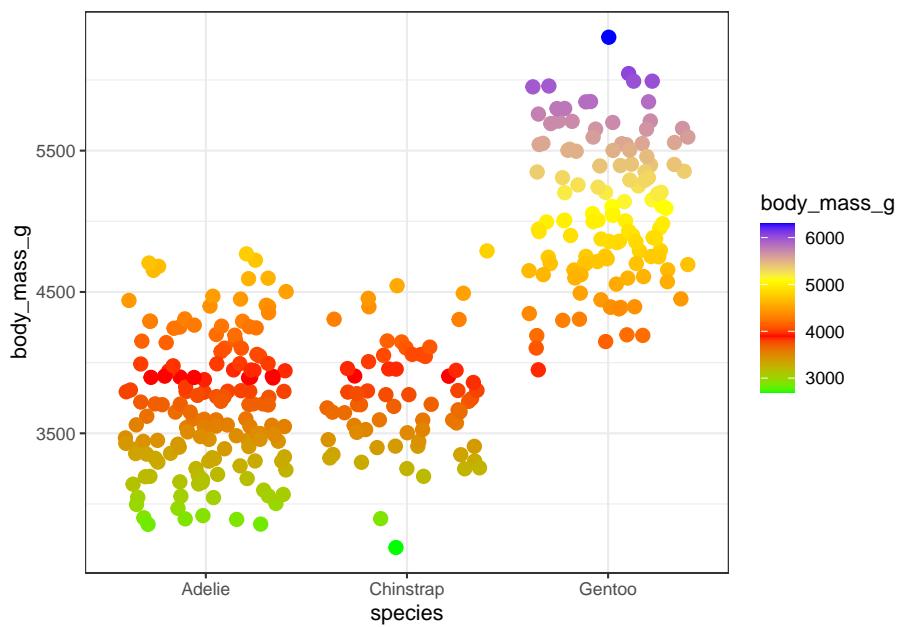
```
# trzy kolory - domyślnie założy, że wartość środkowa to zero, jeżeli jest inaczej to
p + scale_color_gradient2(low = "darkgreen", mid = "coral", high = "gold", midpoint =
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



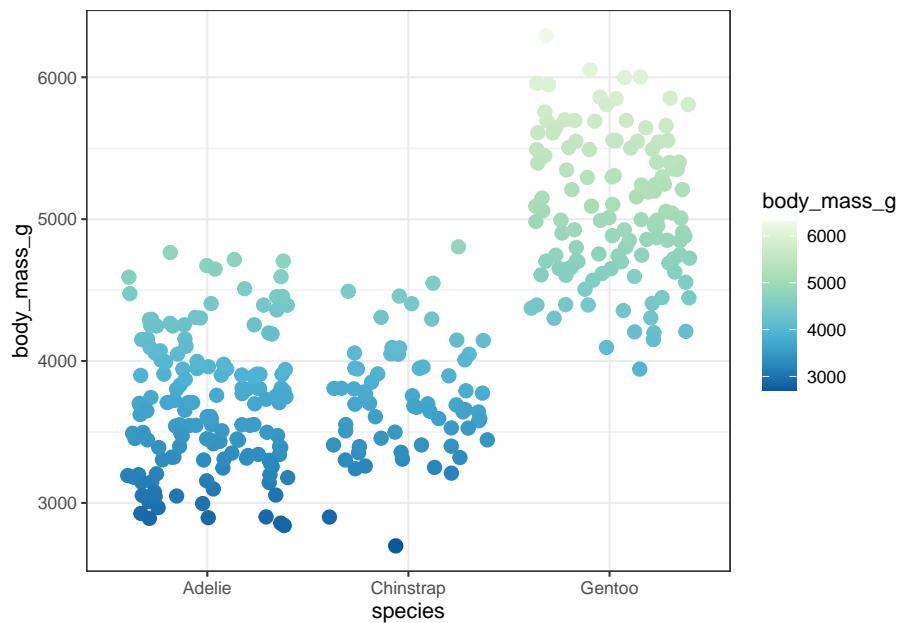
```
# więcej kolorów  
p + scale_color_gradientn(colours = c("green", "red", "yellow", "blue"))
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



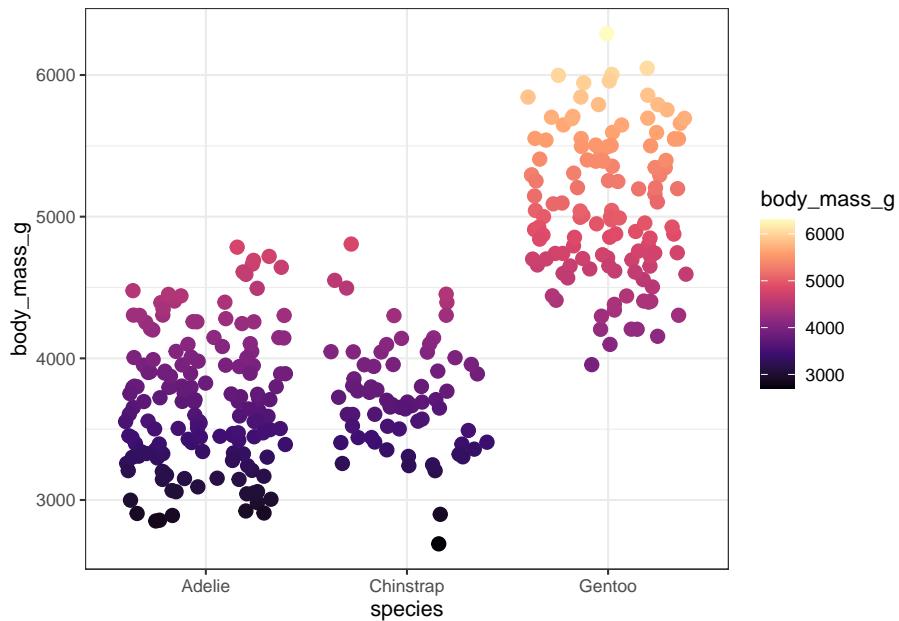
```
# ColorBrewer  
p + scale_color_distiller(palette = 4)
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

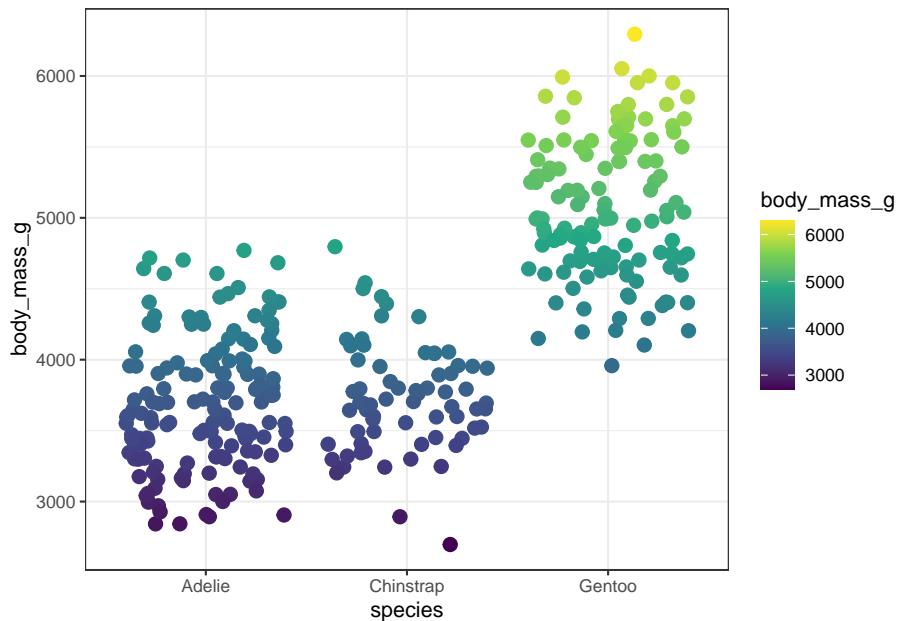


```
# viridis  
p + scale_color_viridis_c(option = 'A')
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

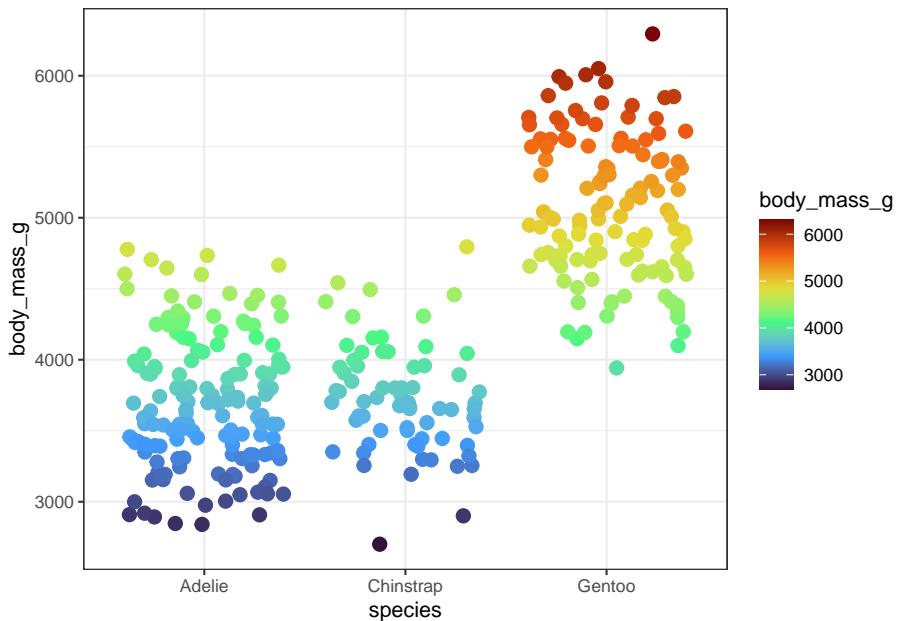


```
p + scale_color_viridis_c(option = 'D')  
## Warning: Removed 2 rows containing missing values (geom_point).
```



```
p + scale_color_viridis_c(option = 'H')

## Warning: Removed 2 rows containing missing values (geom_point).
```



Analogicznie do kolorów można ustawać skale dotyczące kształtu punktów, stylu linii itp.

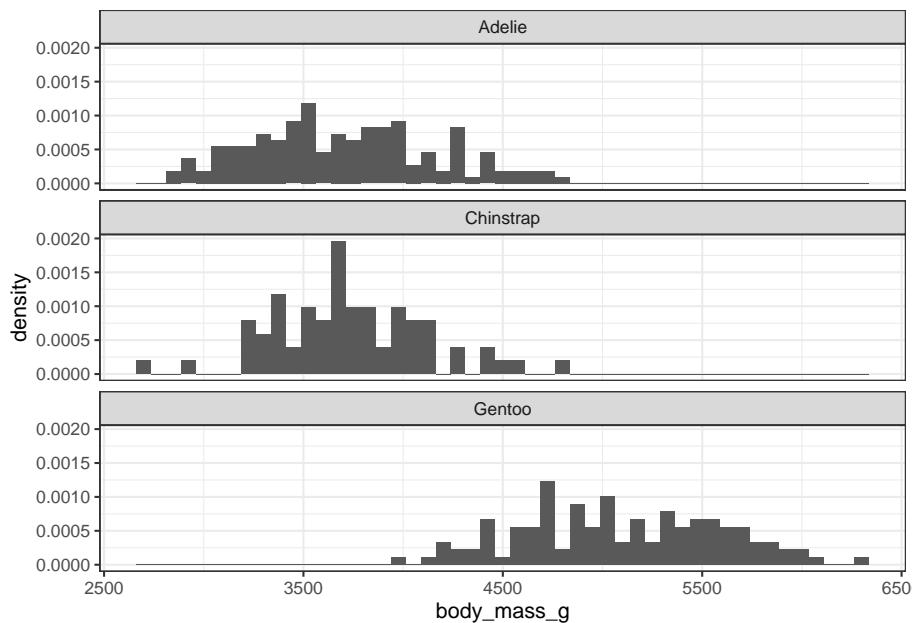
5.6.3 Podział wykresu na panele

Innym sposobem na rozróżnianie danych jest użycie paneli (facet).

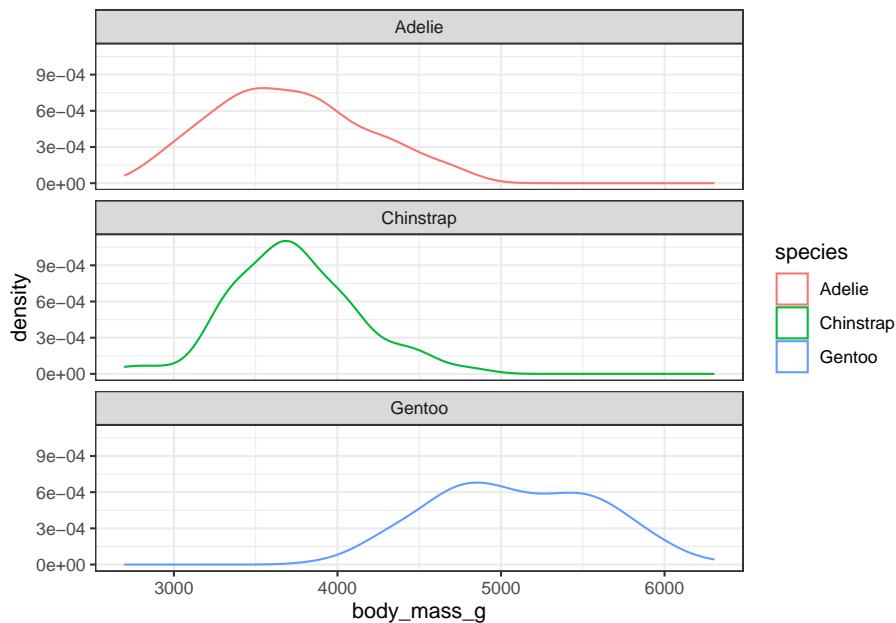
Dzielimy wykres na części pod względem zmiennej np. Szczepu używając `facet_wrap` albo `facet_grid`

`facet_grid` - rozmieści wykresy w tabeli według jednej lub dwóch zmiennych podanych w formule np. `zmienna1 ~ zmienna2` `facet_wrap` - stworzy "wstążkę" wykresów, końcową liczbę kolumn/wiersz można ustawić argumentami `ncol/nrow`. W formule po + można dodawać kolejne zmienne np. `~ zmienna1 + zmienna2`

```
p <- ggplot(data = penguins %>% drop_na(), aes(x = body_mass_g))
p + geom_histogram(aes(y = ..density..), binwidth = 75, position = "dodge") +
  facet_wrap(~species, ncol = 1)
```



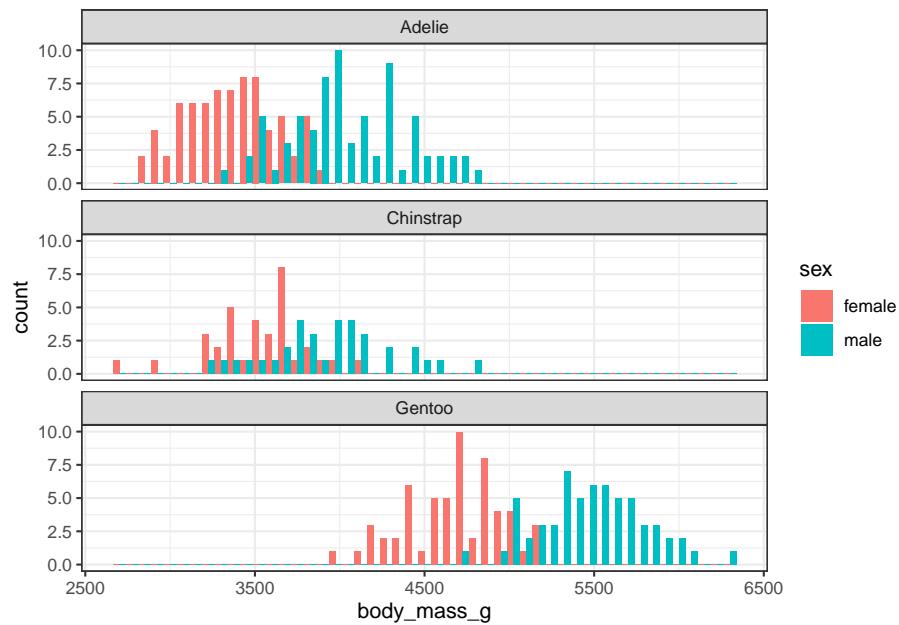
```
p + geom_density(aes(color = species)) + facet_wrap(~species, ncol = 1)
```



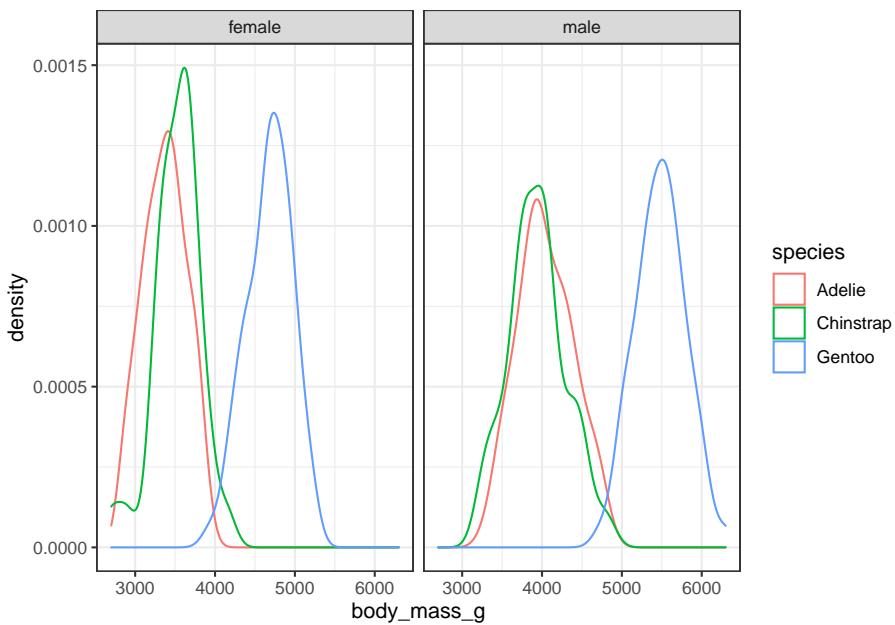
Oba sposoby można ze sobą dowolnie łączyć.

Analizujemy dane pod względem szczepu i warunków.

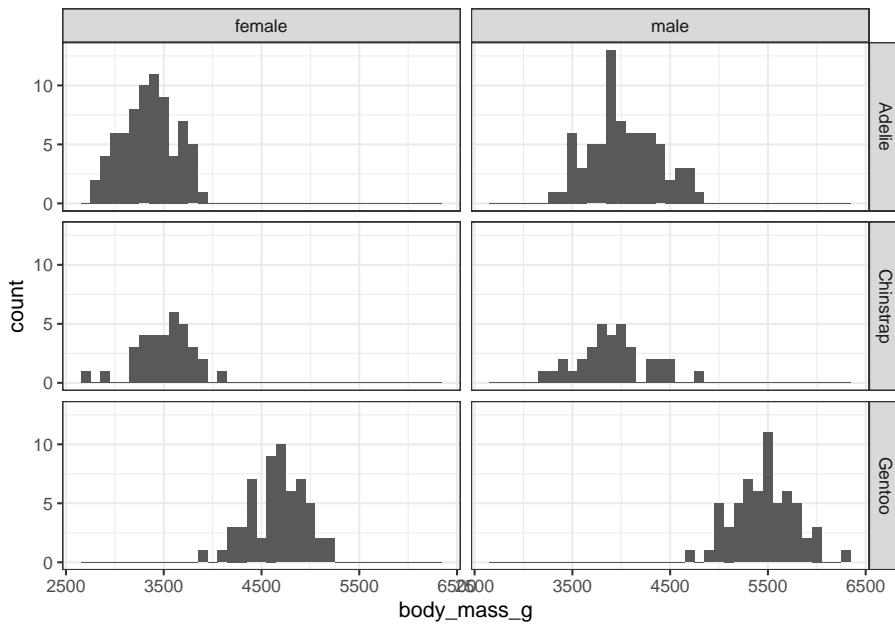
```
p <- ggplot(data = penguins %>% drop_na(), aes(x = body_mass_g))
p + geom_histogram(binwidth = 75, aes(fill = sex), position = "dodge") +
  facet_wrap(~species, ncol = 1)
```



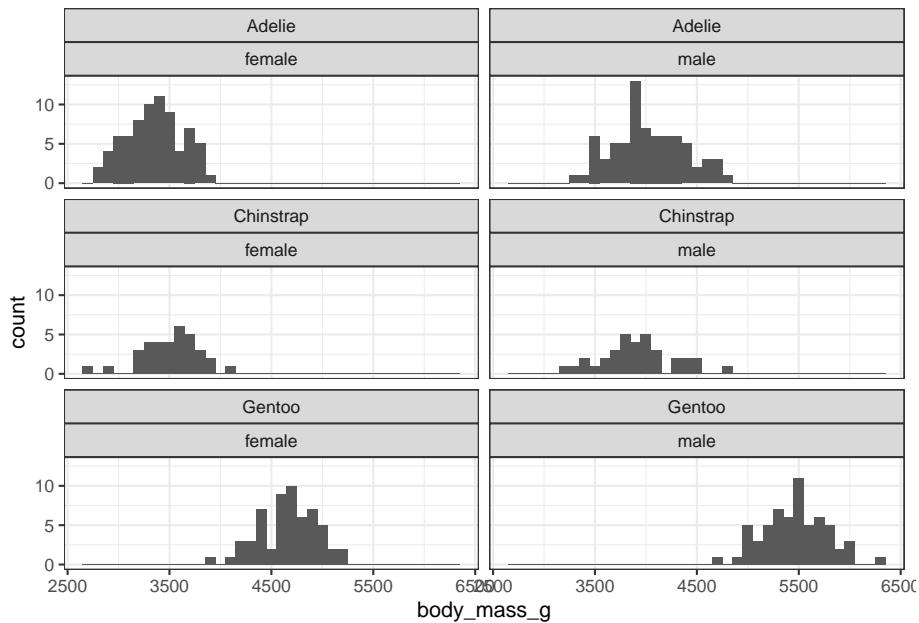
```
p + geom_density(aes(color = species)) + facet_wrap(~ sex, ncol = 2)
```



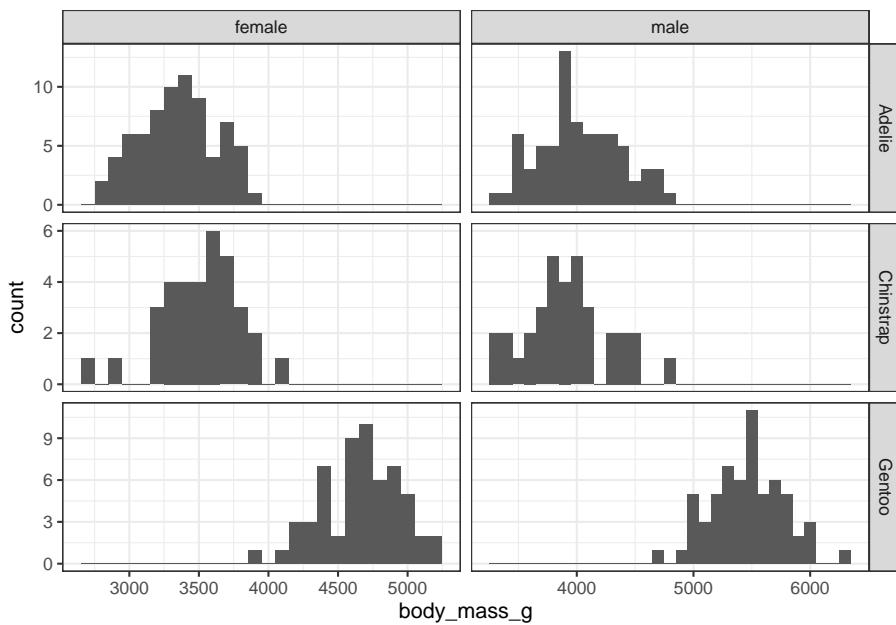
```
# Histogram podzielony pod względem speciesu i warunków - facet_grid
p + geom_histogram(binwidth = 100) + facet_grid(species ~ sex)
```



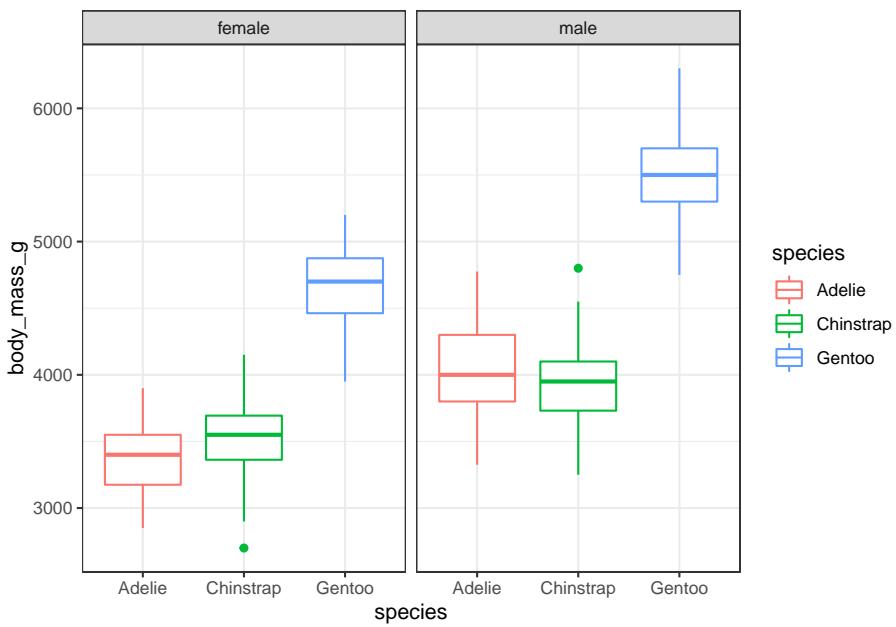
```
# Podobny efekt do facet_frid można osiągnąć stosując facet_wrap z więcej niż jednym s
# Kolejne sex dodaje się znakiem +
p + geom_histogram(binwidth = 100) + facet_wrap(~species + sex, ncol = 2)
```



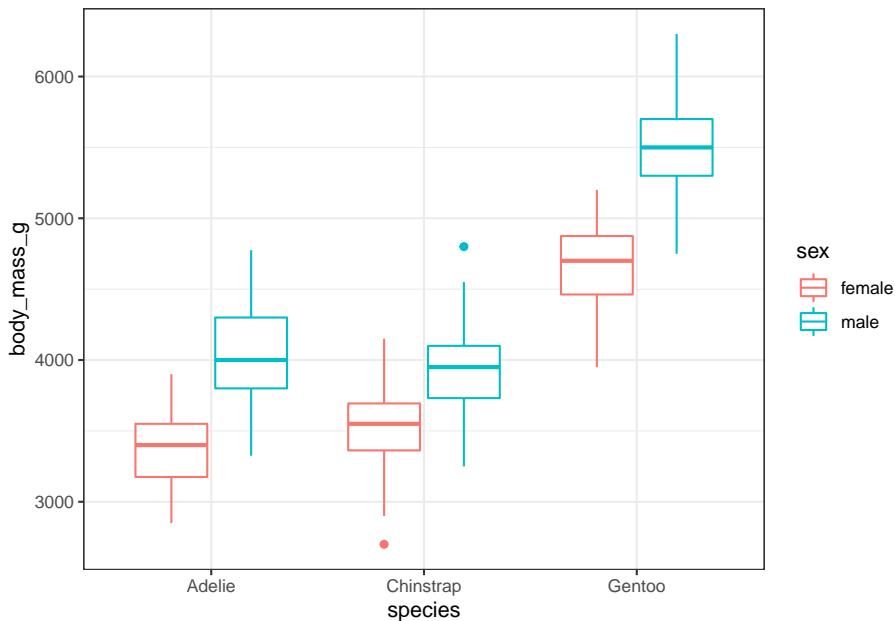
```
# Osie nie muszą być jednakowe dla wszystkich części
p + geom_histogram(binwidth = 100) + facet_grid(species ~ sex, scales = "free")
```



```
# Dzielić na części dzięki facet można każdy typ wykresu
p <- ggplot(data = penguins %>% drop_na(), aes(x = species, y = body_mass_g))
p + geom_boxplot(aes(color = species)) + facet_wrap(~ sex)
```



```
# Boxploty można również rozmiścić według jednego czynnika, a pokolorować według drugiego
p + geom_boxplot(aes(color = sex))
```



5.6.4 Zmiana skali, osi, obracanie wykresu

ggplot2 domyślnie dobiera takie parametry osi, żeby zmieściły się wszyskie dane, ale można je zmieniać używając `scale_x_continuous` albo `scale_y_continuous`.

Jeżeli chcemy tylko zmienić limity osi można to zrobić funkcją `xlim` i `ylim`.

Trzeba pamiętać, że po zmianie limitów osi najpierw zostaną usunięte wartości, które się nie mieścią, a potem policzone statystyki, więc takie wykresy jak `geom_boxplot`, `stat_smooth`, `stat_summary` i inne mogą ulec zmianie. Jeżeli chcemy tego uniknąć należy zamiast osi zmieniać układ współrzędnych - `coord_cartesian(xlim, ylim)`.

Przy ich pomocy możemy zmienić np. parametry: * `limits` - miejsce startu i końca osi np `limits = c(1,10)` * `name` - nazwa osi * `breaks` - miejsca "tick marks" * `labels` - nazwy "tick marks"

Podstawowe transformacje osi to `scale_y_log10`, `scale_y_reverse`, `scale_y_sqrt`.

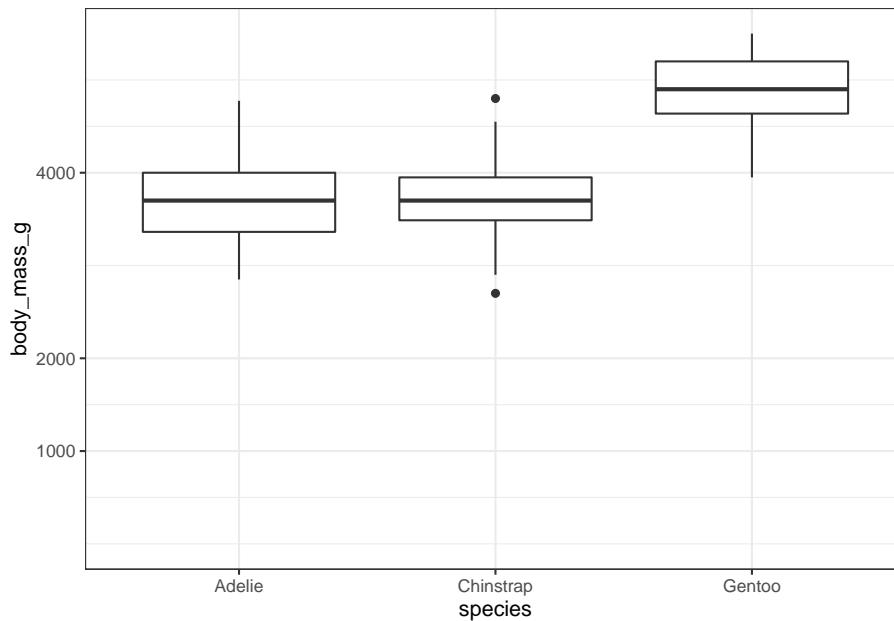
Oś % - należy wpisać `labels=percent` w scale oraz załadować pakiet scales.

Jeżeli chcemy obrócić wykres o 90 stopni możemy użyć funkcji `coord_flip`.

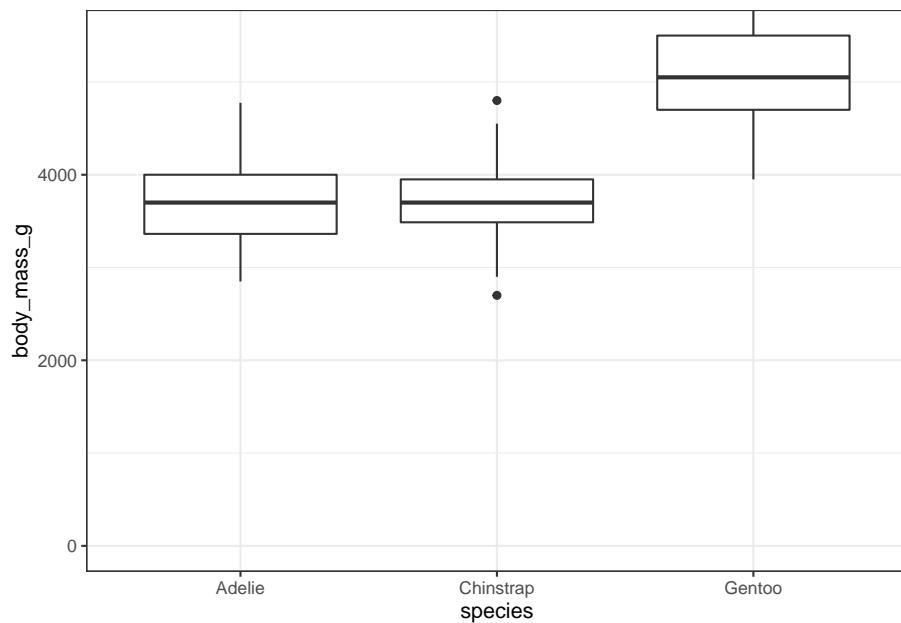
```
# Zmiana osi na przykładzie boxplot
p <- ggplot(data = penguins %>% drop_na(), aes(x = species, y = body_mass_g))
p <- p + geom_boxplot()

# Ustawienie startu i końca osi oraz miejsc podziału
p + scale_y_continuous(limits = c(0,5500), breaks = c(1000, 2000, 4000))

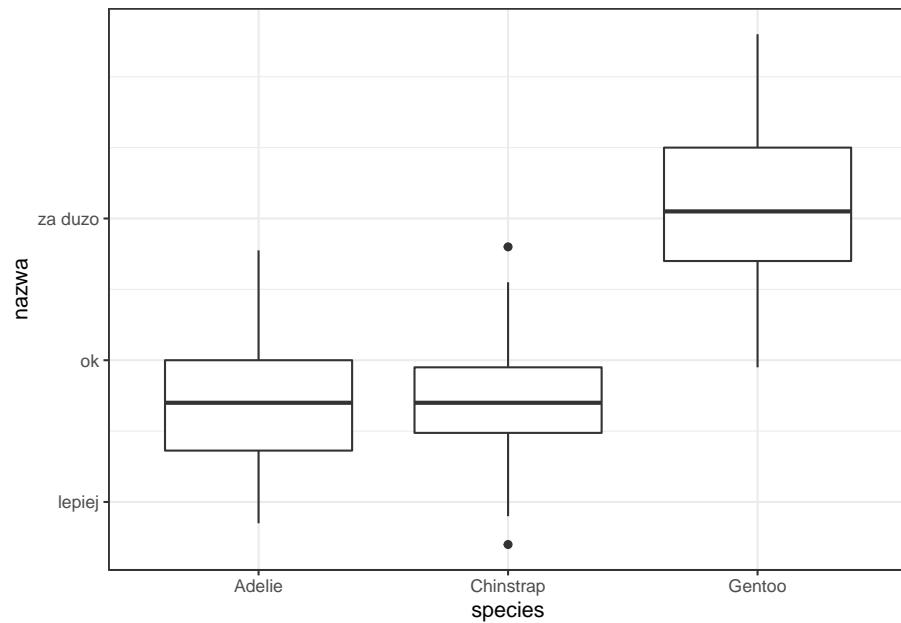
## Warning: Removed 28 rows containing non-finite values (stat_boxplot).
```



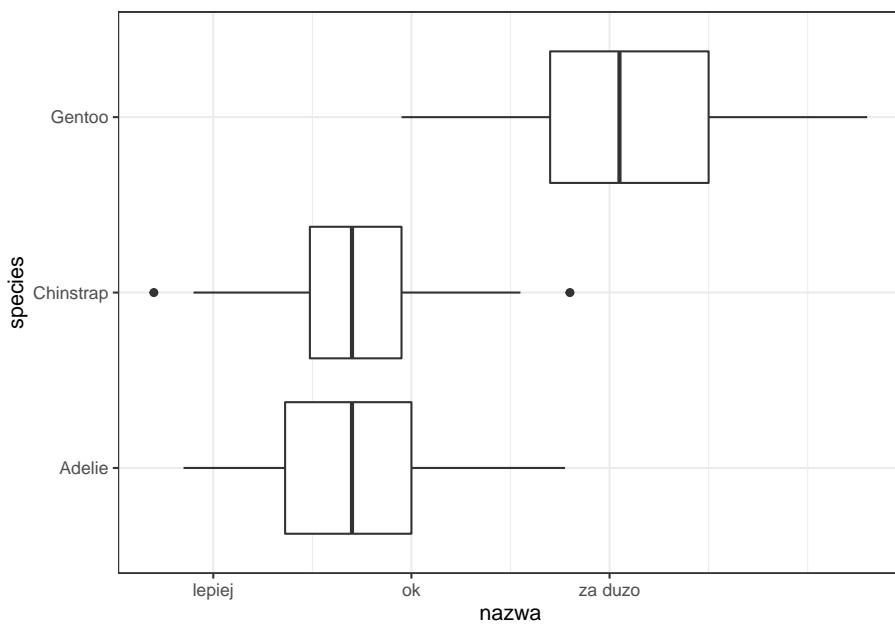
```
# przy pomocy coord_cartesian
p + coord_cartesian(ylim = c(0, 5500))
```



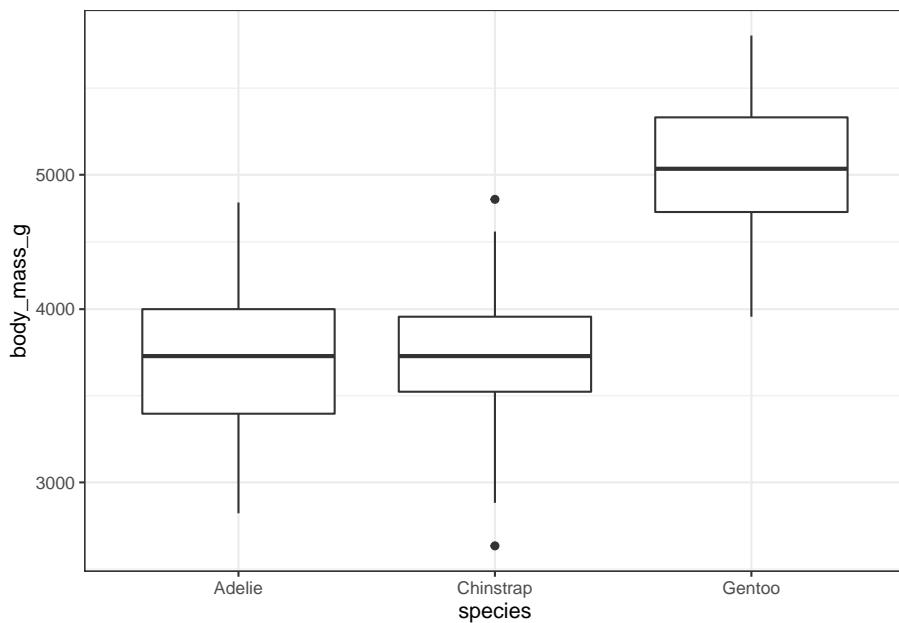
```
# Miejsca podziału nie muszą być w równych odstępach i mogą być dowolnie nazwane
p + scale_y_continuous(breaks = c(2000, 3000, 4000, 5000),
                        labels = c("malo", "lepiej", "ok", "za duzo"),
                        name = "nazwa")
```



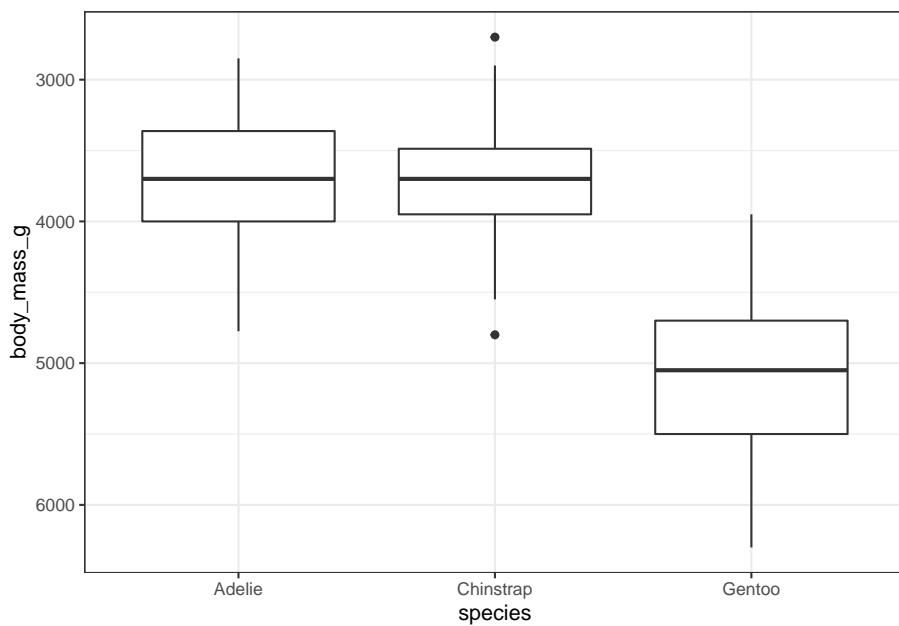
```
# Wykres obrócony  
p + scale_y_continuous(breaks = c(2000, 3000, 4000, 5000),  
                        labels = c("malo", "lepiej", "ok", "za duzo"),  
                        name = "nazwa") + coord_flip()
```



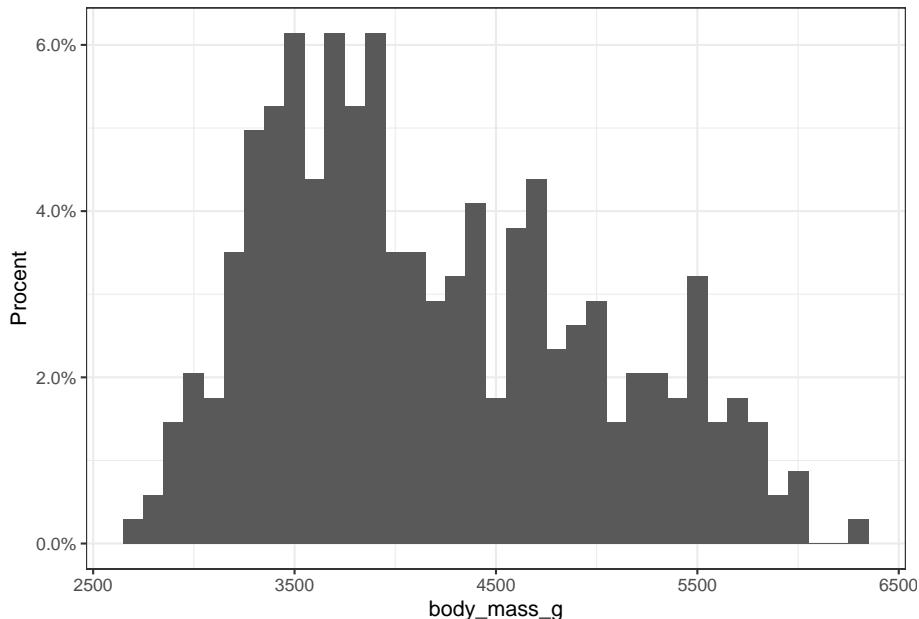
```
# Skala logarytmiczna  
p + scale_y_log10()
```



```
# Wykres "do góry nogami"  
p + scale_y_reverse()
```



```
library(scales)
# Oś procentowa
p <- ggplot(data = penguins, aes(x = body_mass_g))
p + geom_histogram(binwidth = 100, aes(y = ((..count..)/sum(..count..))))+
  scale_y_continuous(labels = percent, name = "Procent")
## Warning: Removed 2 rows containing non-finite values (stat_bin).
```



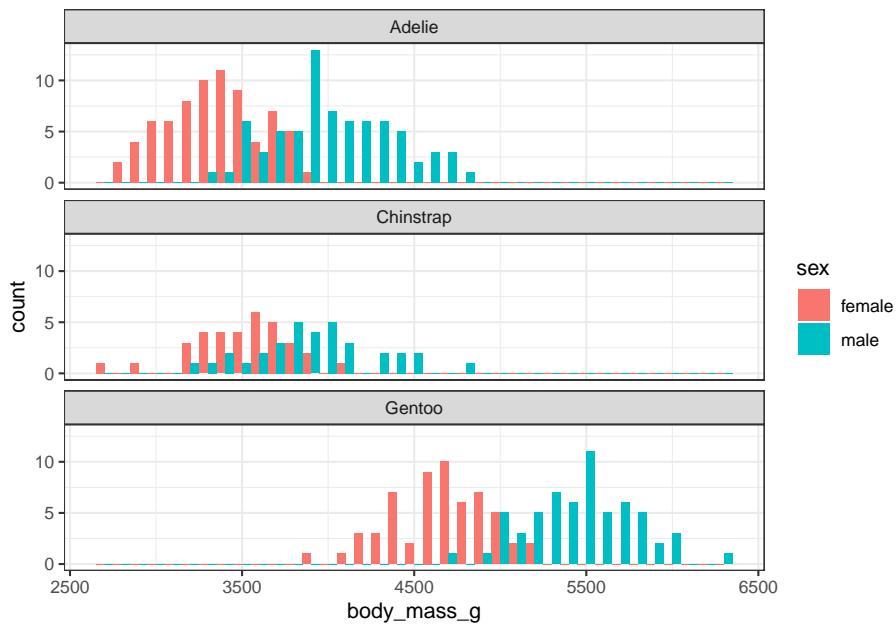
5.7 Motyw (theme)

W pakiecie `ggplot2` jest dostępnych kilka różnych motywów. Domyślnie ustawiony jest `theme_grey`, inne dostępne to `theme_bw`, `theme_minimal`, `theme_classic`, `theme_linedraw`, `theme_light`. W pakiecie `ggthemes` znajdują się dodatkowe wersje motywów, nawet (o zgrozo) `theme_excel`;) Inny pakiet zawierający gotowe motywów to `ggthemr` - można go pobrać z GitHub.

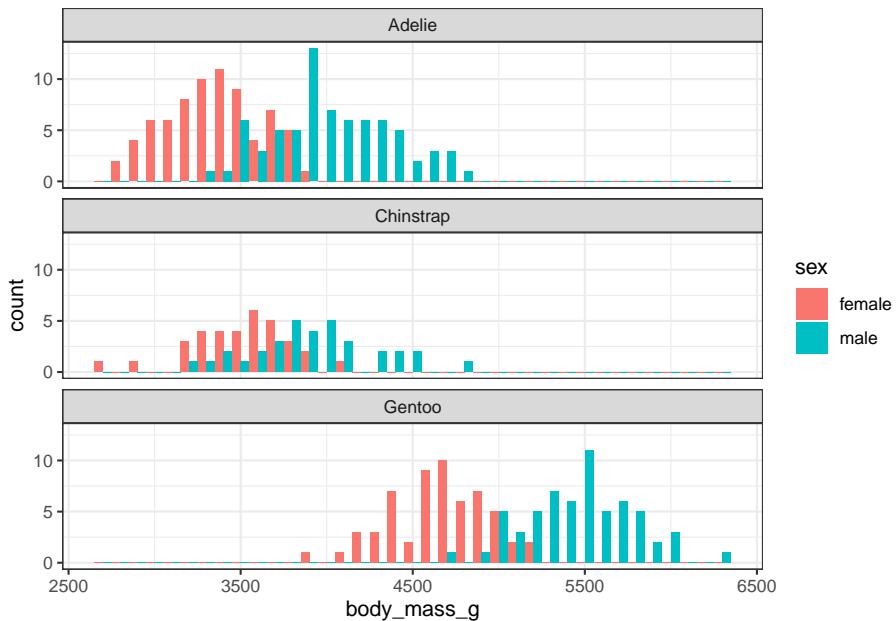
Tytuł do wykresu możemy dodać korzystając z funkcji `ggtitle`. Nazwy osi też można szybko zmienić przy pomocy `xlab` i `ylab`. Dalsze opisy można dodać korzystając z funkcji `labs` np. tag albo caption.

```
p <- ggplot(data = penguins %>% drop_na(), aes(x = body_mass_g))
p <- p + geom_histogram(binwidth = 100, aes(fill = sex), position = "dodge")+
```

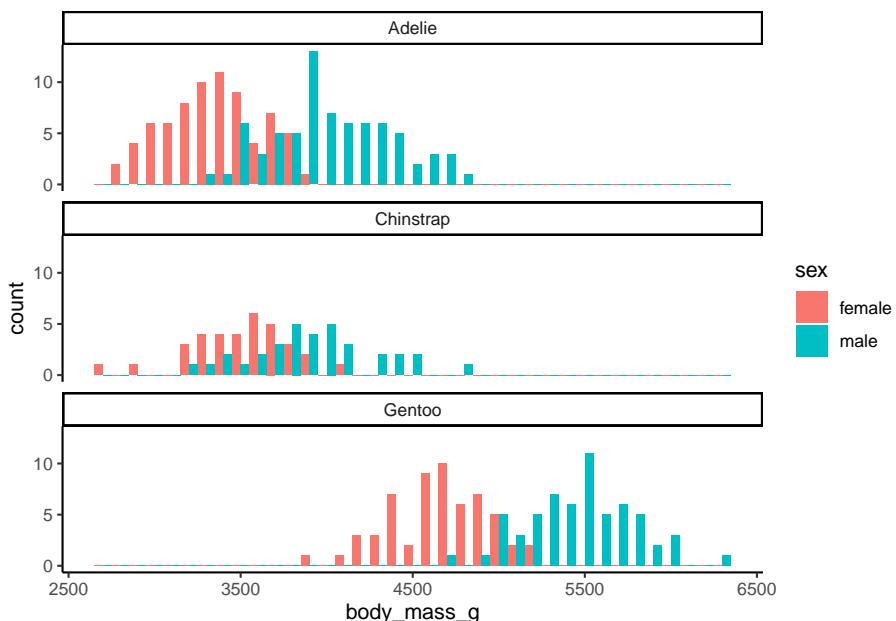
```
  facet_wrap(~species, ncol = 1)  
p
```



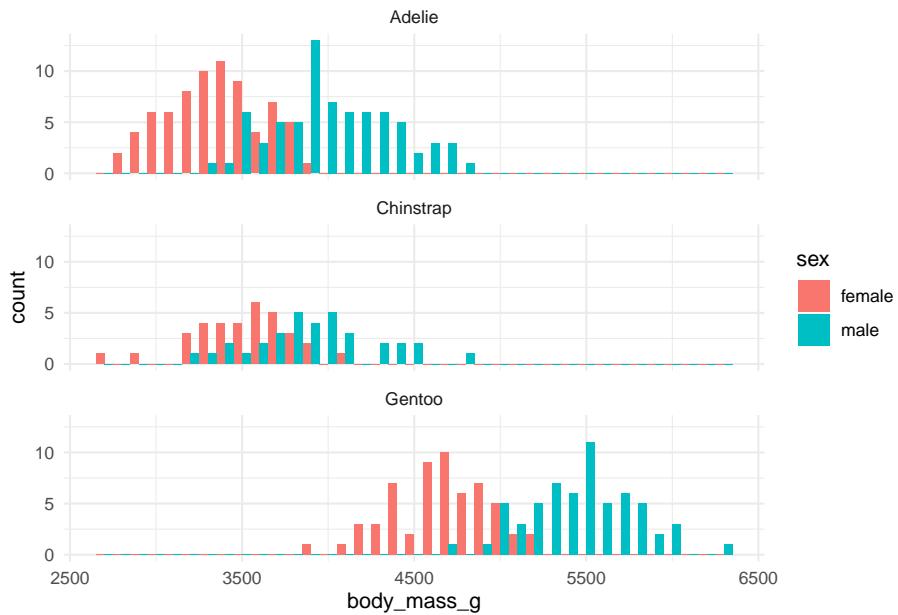
```
p + theme_bw()
```



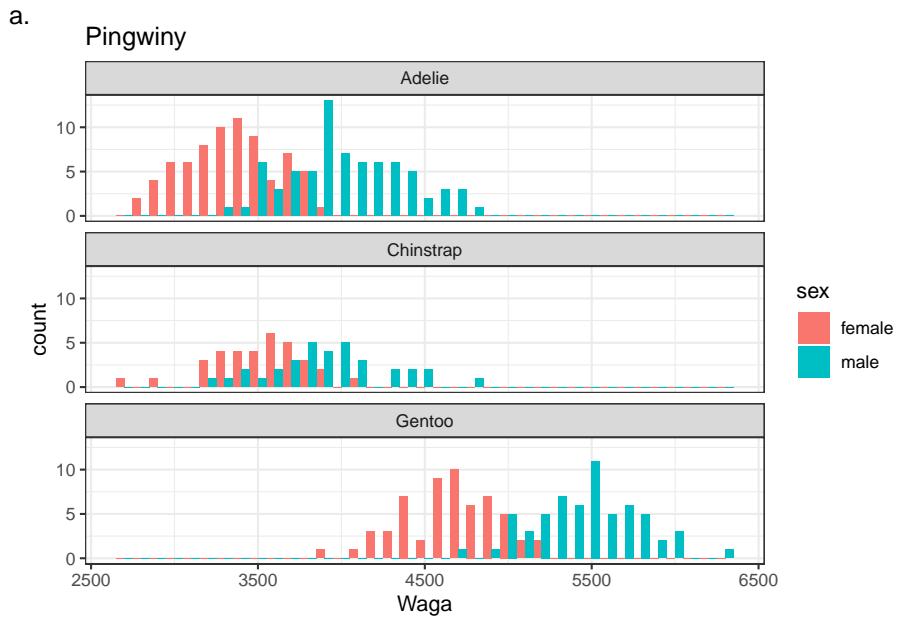
```
p + theme_classic()
```



```
p + theme_minimal()
```

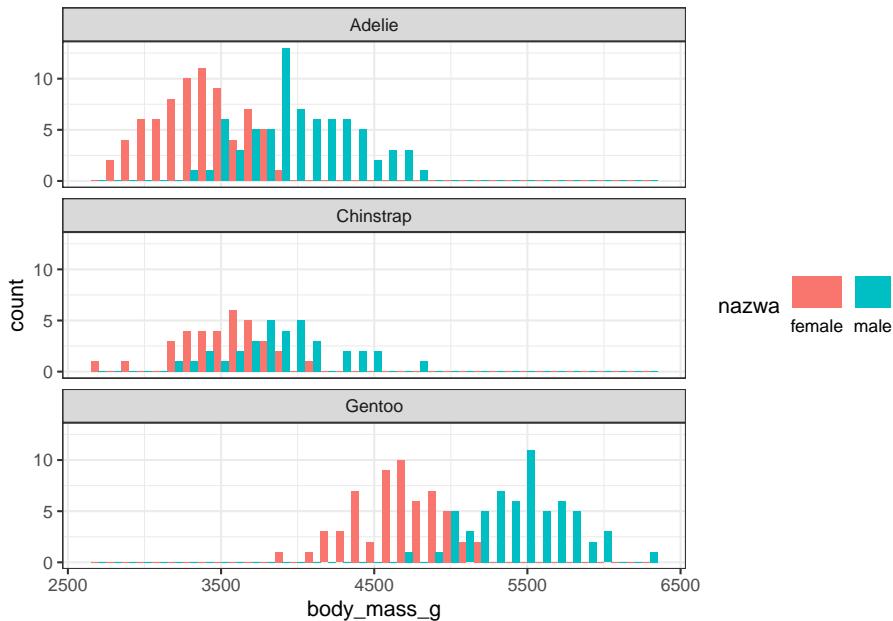


```
p + ggtitle("Pingwiny") + xlab("Waga") + labs(tag = 'a.')
```



Legendę wykresu można modyfikować przy pomocy `guide_legend` wewnątrz funkcji `scale_fill_discrete`, `scale_colour_discrete` itp. Dostępne parametry to m.in. `title`, `title.position`, `label.position`, `direction`, `nrow` i `ncol` legendy. Modyfikacje są możliwe też bezpośrednio w funkcji `theme` albo samodzielnie ustawiając przez `scale_color_manual`.

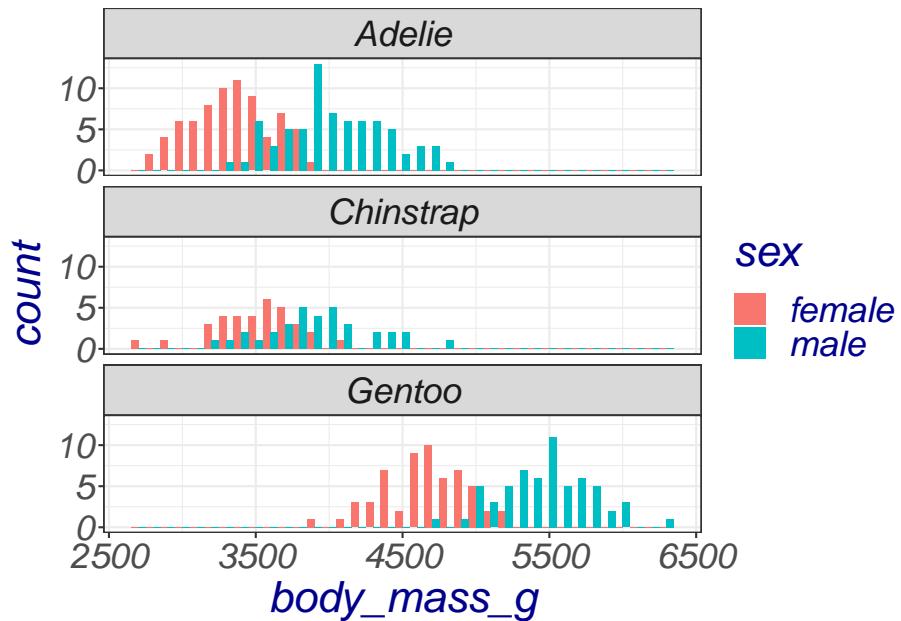
```
p + scale_fill_discrete(guide = guide_legend(title = "nazwa",
                                               title.position = "left",
                                               label.position = "bottom", ncol = 2))
```



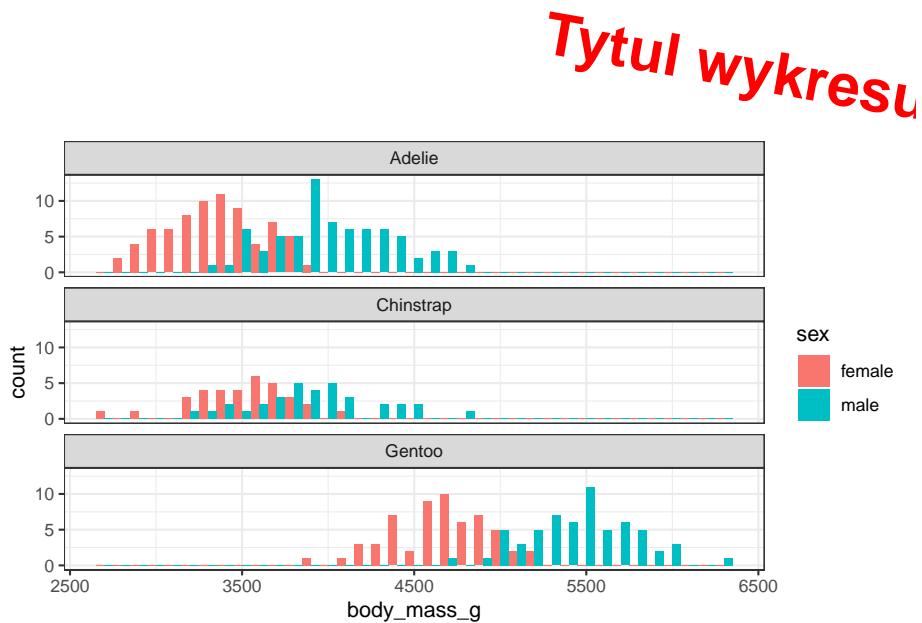
Można również modyfikować osobno każdy element wykresu np. czcionkę, kolor, linie, tło itd. przy pomocy funkcji `theme`, dużo przykładów znajduje się na stronie.

Można modyfikować jednocześnie wszystkie elementy danego rodzaju np. tekst przy pomocy `text = element_text()` albo pojedyncze części wykresu np. tytuł - `plot.title=element_text()`.

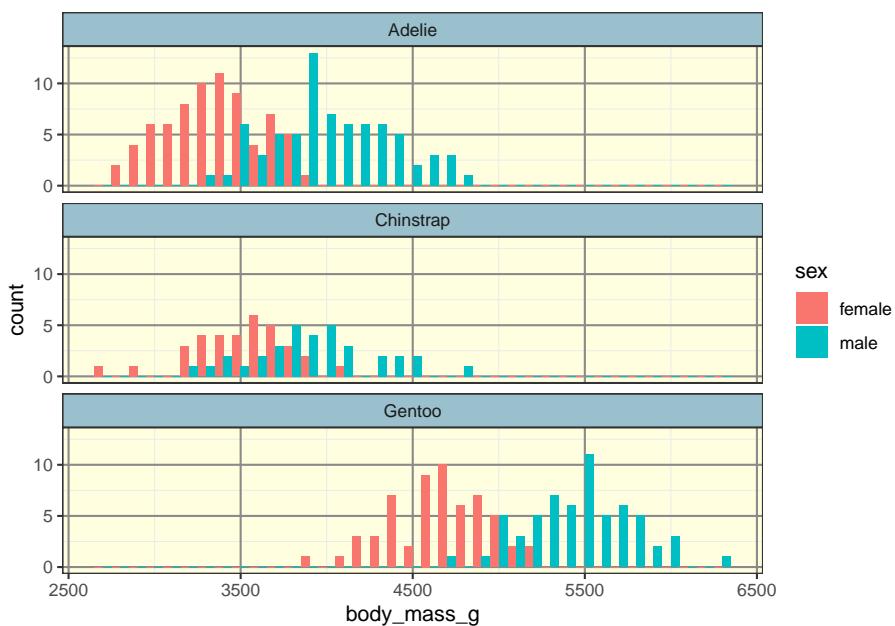
```
p + theme(text = element_text(size = 22, face = "italic", color = "darkblue"))
```



```
p + ggtitle("Tytul wykresu") + theme(plot.title = element_text(color = "red",
                                                               face = "bold", size = 30, a
```



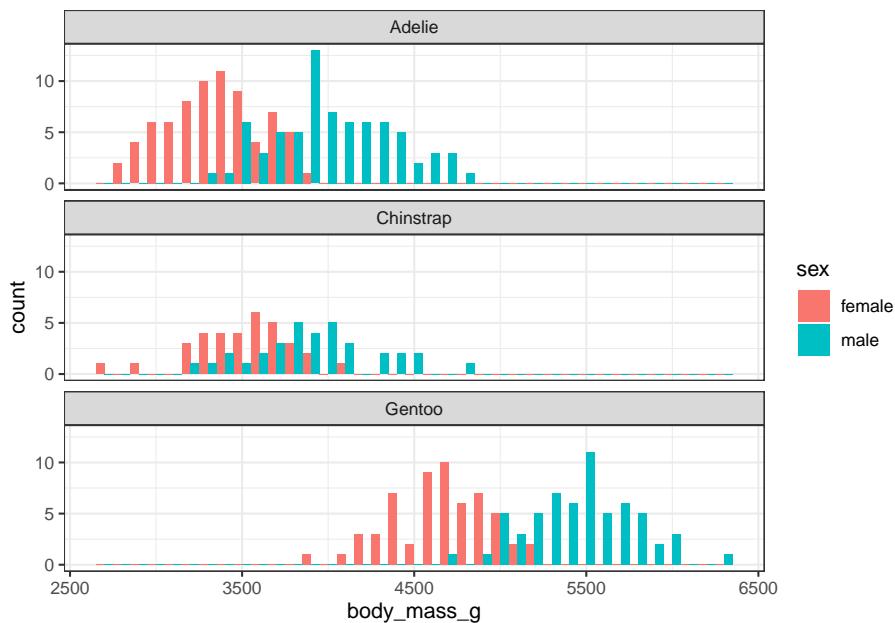
```
p + theme(panel.background = element_rect(fill = "lightyellow"),
          panel.grid.major = element_line(color = "snow4"),
          strip.background = element_rect(fill = "lightblue3"))
```



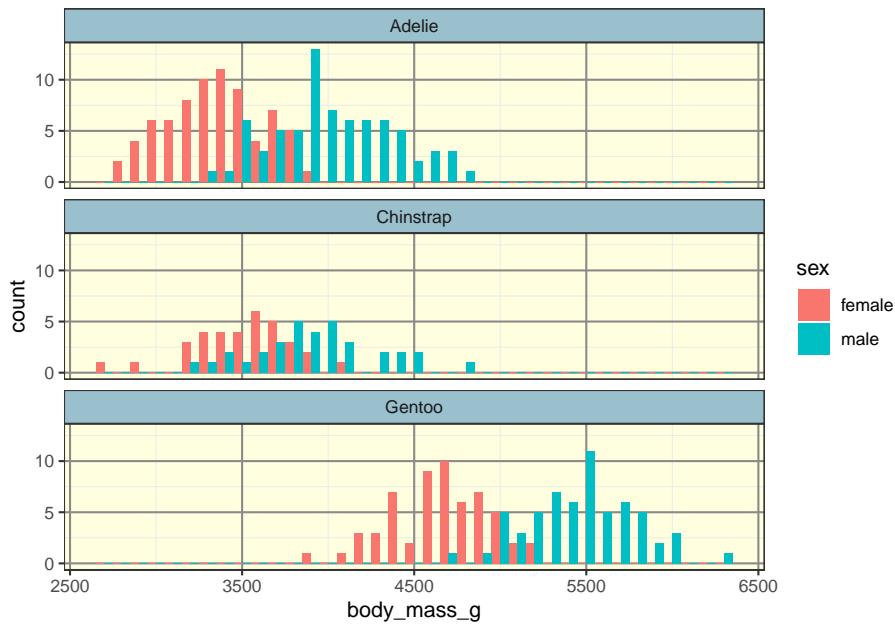
Jeżeli chcemy przygotować kilka pasujących do siebie wykresów możemy zapisać swój motyw i potem dodawać go do kolejnych wykresów.

```
motyw <- theme(panel.background = element_rect(fill = "lightyellow"),
                 panel.grid.major = element_line(color = "snow4"),
                 strip.background = element_rect(fill = "lightblue3"))

p
```



```
p + motyw
```



5.8 Różne

5.8.1 Łączenie wykresów

Pakiet `ggplot2` jest oparty o system wyświetlania kontrolowany przez pakiet `grid` (inny niż grafika z podstawowego R). Korzystając z funkcji `viewport` można z dużą dokładnością rozmieścić kilka wykresów różnych rozmiarów obok siebie, jeden na drugim itp.

W funkcji `viewport` ustawiamy parametry `width` i `height` oznaczające wymiary wykresu. Wykres zajmujący całą powierzchnię ma wymiary `1x1` oraz `x` i `y` oznaczające współrzędne środka wykresu np. `x=0.5, y=0.5` da wykres umiejscowiony na samym środku.

Dużo łatwiejszym sposobem jest wykorzystanie pakietu `patchwork`. Pozwala on na ułożenie wykresów na jednej stronie tylko przy użyciu `+`, `|` i `/`. Można też dokładnie ustalać rozmieszczenie wykresów, więcej na stronie autora pakietu

```
# Przypisujemy wykresy do zmiennych

p1 <- ggplot(data = penguins %>% drop_na(), aes(x = body_mass_g))
p1 <- p1 + geom_density(aes(color = species)) + facet_wrap(~sex, ncol = 2)

p2 <- ggplot(data = ChickWeight, aes(x = Time, y = weight, color = Diet))
p2 <- p2 + geom_point(size = 0.75) + stat_smooth(method = "lm")

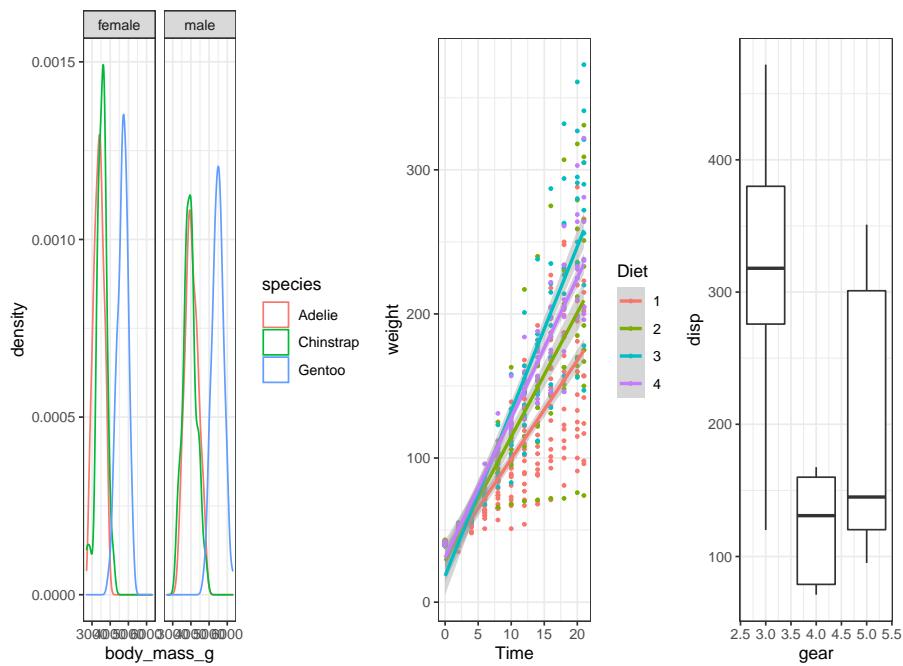
p3 <- ggplot(mtcars) + geom_boxplot(aes(gear, disp, group = gear))

# z wykorzystaniem patchwork

library(patchwork)

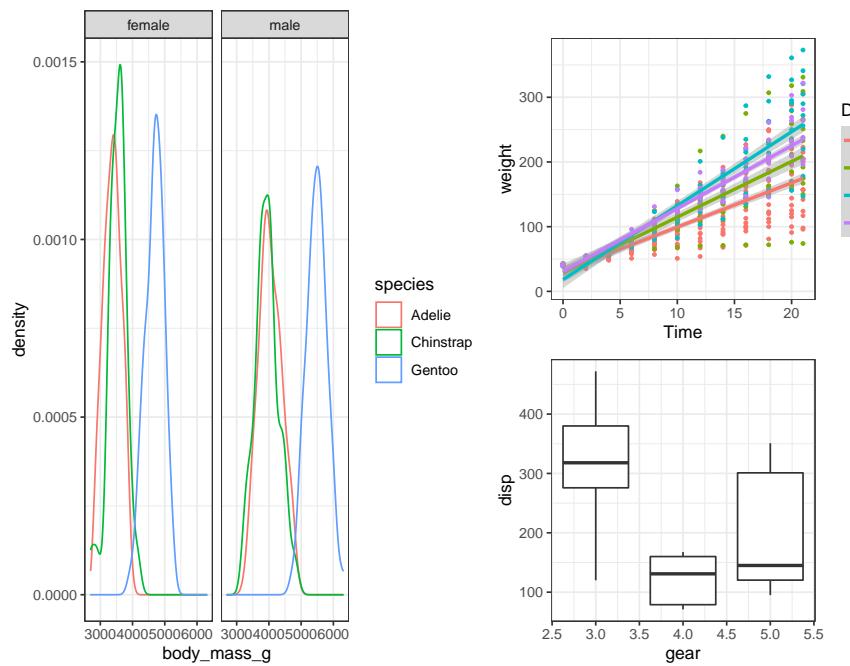
p1 + p2 + p3

## `geom_smooth()` using formula 'y ~ x'
```



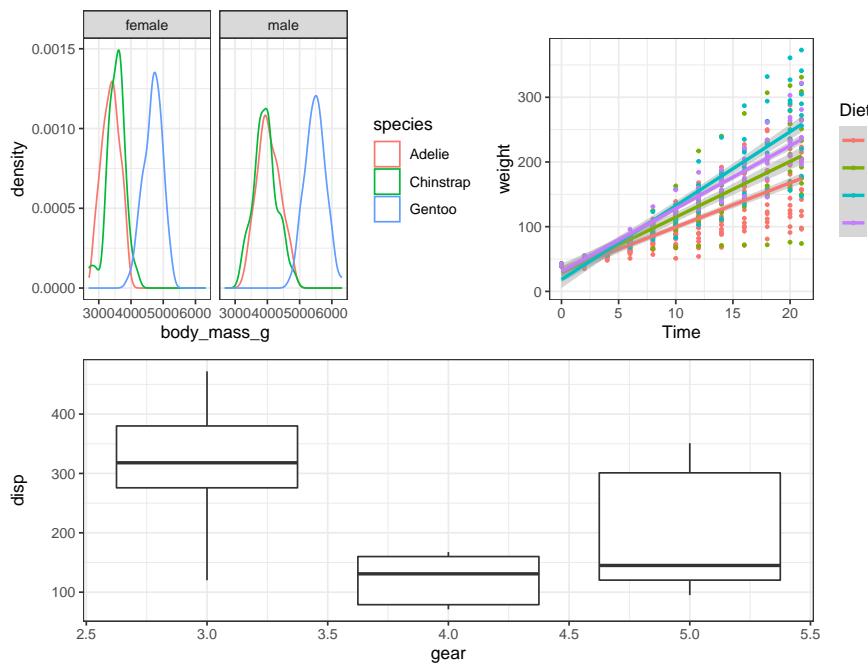
```
p1 | p2 / p3
```

```
## `geom_smooth()` using formula 'y ~ x'
```



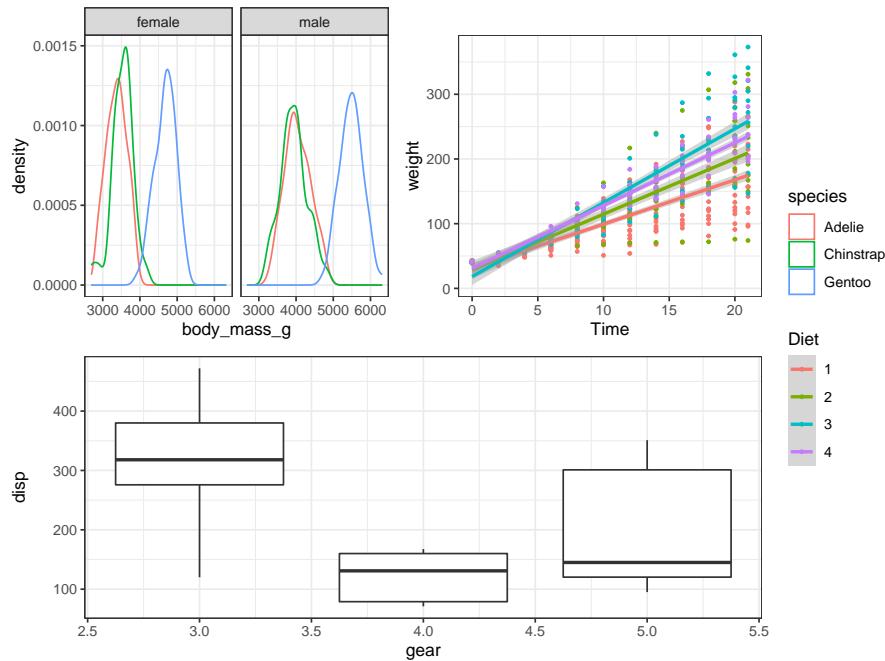
```
(p1 | p2 )/ p3
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
(p1 | p2 )/ p3 + plot_layout(guides = 'collect')
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```

layout <- "
##BBBB
AACCCC
AACCCC
"

p1 + p2 + p3 +
  plot_layout(design = layout)

## `geom_smooth()` using formula 'y ~ x'

# Z wykorzystaniem pakietu grid

library(grid)

p1 <- p1 + theme(text=element_text(size=10))
p2 <- p2 + theme(text=element_text(size=10), legend.position="bottom")

# Ustawiamy parametry viewportów
vp1 <- viewport(width=1, height=0.5, x=0.5, y=0.75)
vp2 <- viewport(width=0.4, height=0.5, x=0.2, y=0.25)
vp3 <- viewport(width=0.6, height=0.5, x=0.7, y=0.25)

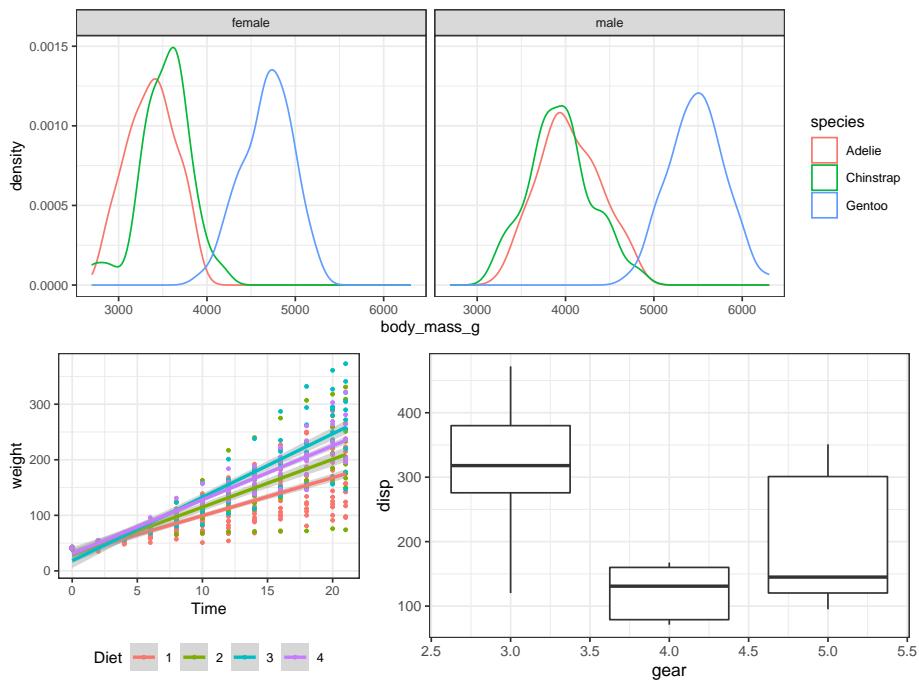
```

```
# Wyświetlamy wykresy w odpowiednich viewportach
```

```
print(p1, vp=vp1)
print(p2, vp=vp2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
print(p3, vp=vp3)
```



5.8.2 Ten sam wykres różne dane

Istnieje kilka sposobów na przygotowanie kilku takich samych wykresów, różniących się jedynie danymi.

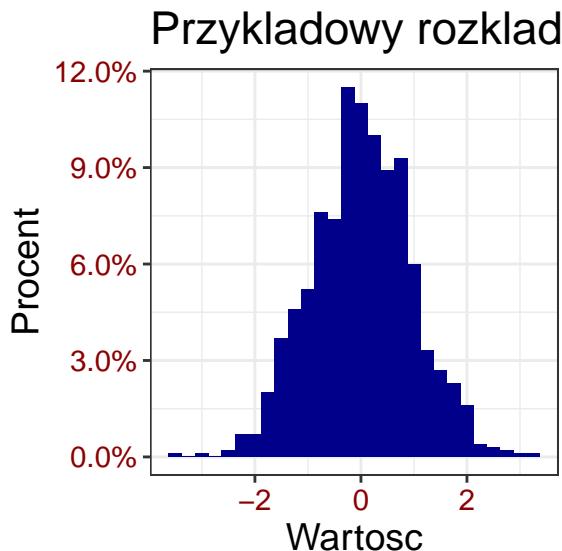
Można oczywiście ręcznie podmienić wartość parametru `data` na inny albo skorzystać z wbudowanego w pakiet `ggplot2` operatora - `%+%`.

Alternatywą jest też napisanie własnej funkcji przygotowującej konkretny wykres. Zaletą tego rozwiązania jest możliwość wpisania do funkcji odpowiednich argumentów dostosowujących wykres do konkretnej sytuacji.

```
# Przykładowe zestawy danych
a <- data.frame(x = rnorm(1000))
b <- data.frame(x = rlnorm(1000))
c <- data.frame(x = runif(1000, 0, 5))

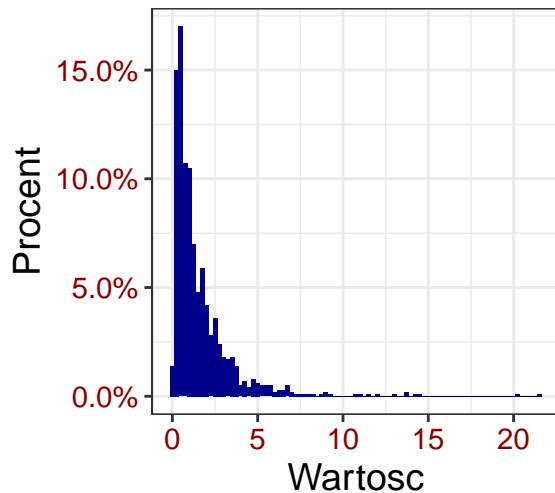
# Przygotujemy wykres składający się z kilku elementów dla danych a

p <- ggplot(data = a, aes(x = x))
p <- p + geom_histogram(binwidth = 0.25, fill = "blue4", aes(y = (..count../sum(..count..))))+
  scale_y_continuous(labels = percent, name = "Procent")+
  xlab("Wartosc")+
  ggtitle("Przykładowy rozkład")+
  theme(panel.background=element_rect(fill = "white"), text = element_text(size = 14),
        axis.text = element_text(color = "red4"))
p
```



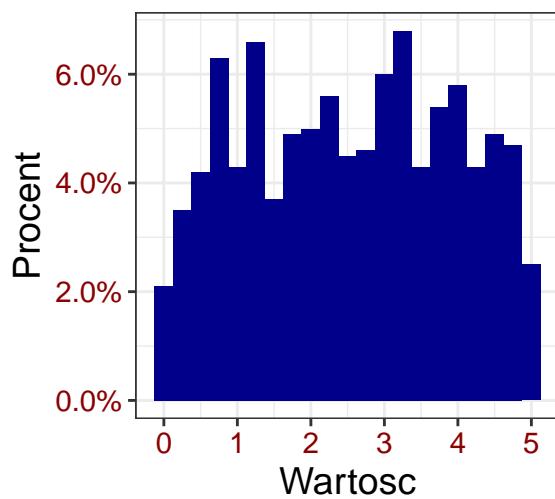
```
# taki sam wykres dla danych b
p %+% b
```

Przykładowy rozkład



```
# i c ;)  
p %+% c
```

Przykładowy rozkład



5.9 Rozszerzenia ggplot2

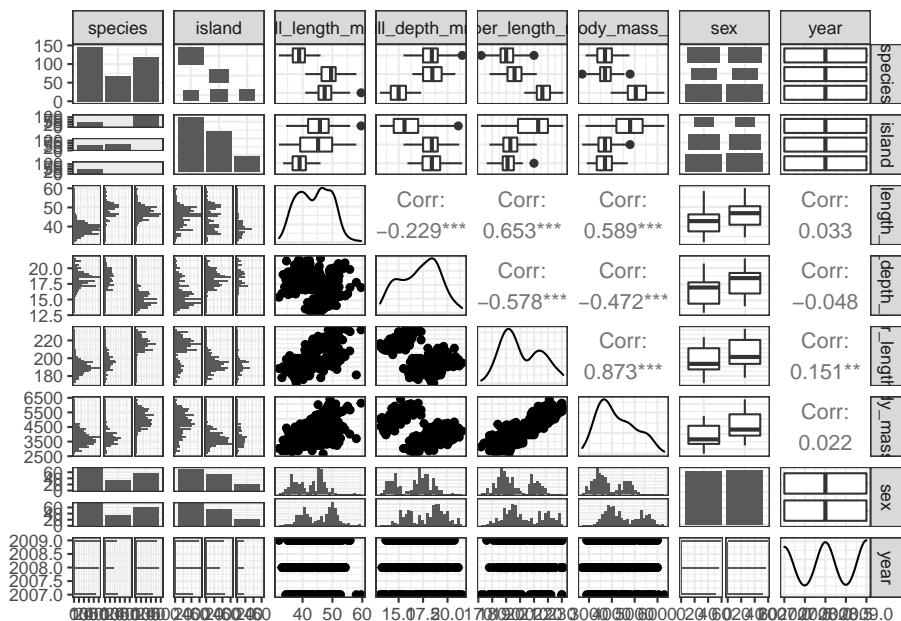
W ostatnich latach powstało bardzo wiele pakietów rozbudowujących możliwości ggplot2. Część z nich została już wspomniana wcześniej np. ggbeeswarm lub patchwork. Tutaj znajdą się inne, które również mogą okazać się przydatne. Większość dobrze udokumentowanych pakietów można znaleźć na stronie ggplot2 extensions - gallery.

5.9.1 Pakiet GGally

Pakiet GGally stanowi rozszerzenie ggplot2, zawiera kilka szablonów i pozwala na stworzenie wykresów niedostępnych w wersji podstawowej np. macierz korelacji, wykres pokazujący sieć albo macierz wykresów dla ramki danych.

5.9.1.1 Macierz wykresów - ggpairs

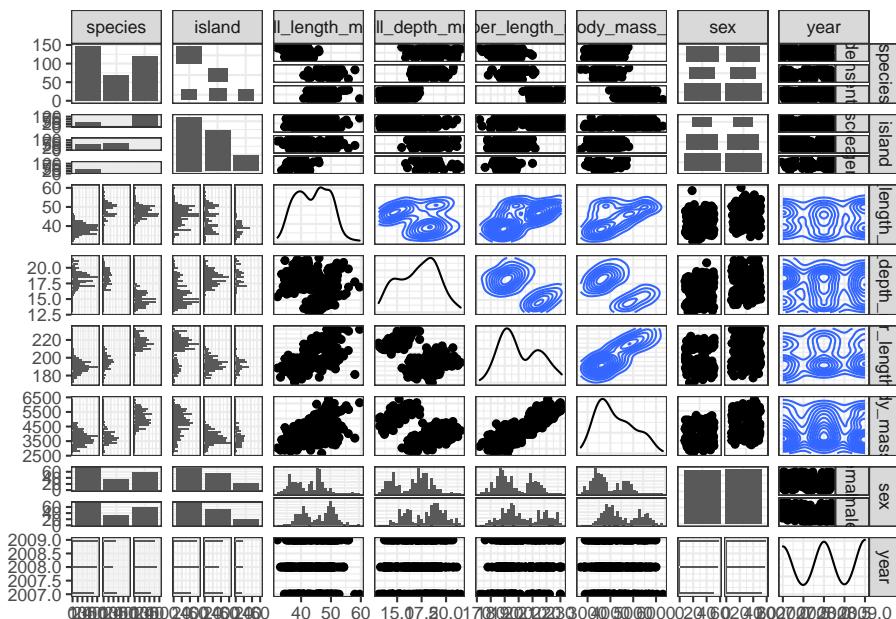
Funkcja `ggpairs` pozwala na szybką analizę danych. Jej argumentem jest ramka danych i dla każdej pary zmiennych zostanie narysowany wykres pozkazujący zależność pomiędzy nimi. Wykresy są inne w zależności od rodzaju zmiennych - liczbowe lub kategoryczne. Dla pary zmiennych liczbowych zostanie narysowany wykres rozrzutu i obliczony współczynnik korelacji. Dla pary mieszanej (liczbowo-kategorycznej) narysuje wykres pudełkowy i histogram, dla dwóch zmiennych kategorycznych wykresy słupkowe. Rodzaje rysowanych wykresów można zmieniać, można też do macierzy dodać własny wykres.



```
# zmiana rodzaju wykresu np. górnny panel pokaże wykres gęstości zamiast korelacji
# i kropkowy zamiast boxplota
ggpairs(penguins %>% drop_na(), upper = list(continuous = "density", combo = "dot"))
```

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

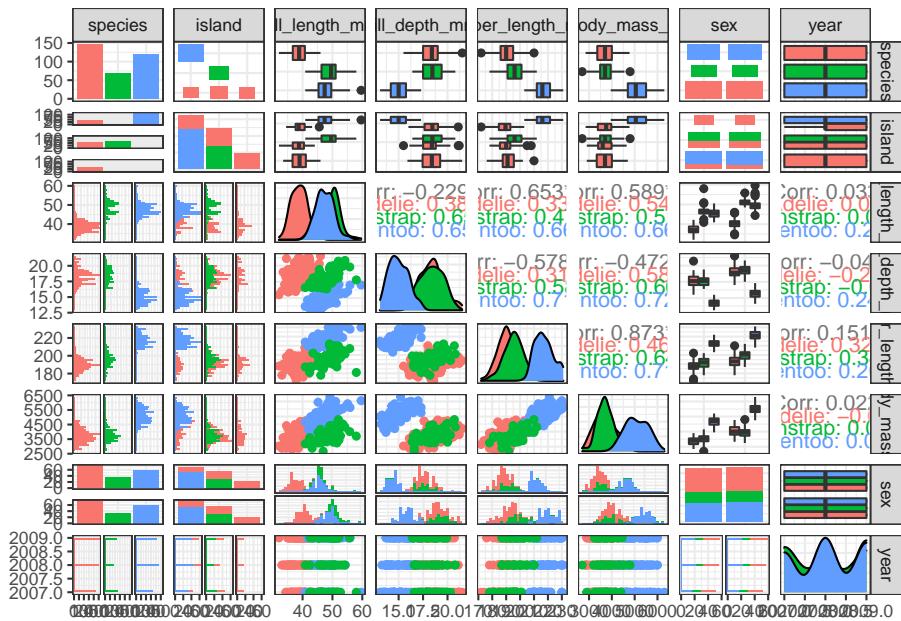
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



wykres pokolorowany według jednej z kategorii

```
ggpairs(penguins %>% drop_na(), mapping = ggplot2::aes(color = species))
```

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

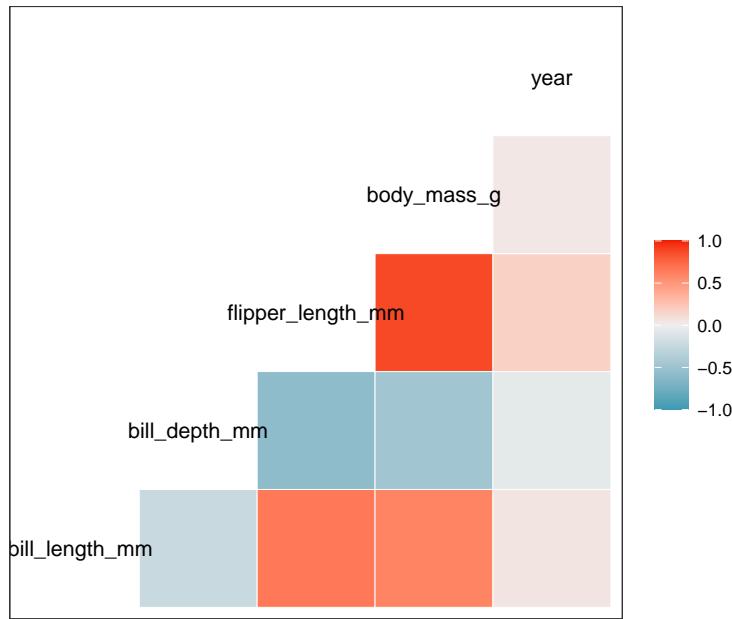


5.9.1.2 Macierz korelacji

Tworzenie macierzy korelacji jest opisane w dalszej części z wykorzystaniem pakietu corrplot, ale możliwe jest też użycie ggplot2.

```
ggcorr(penguins)
```

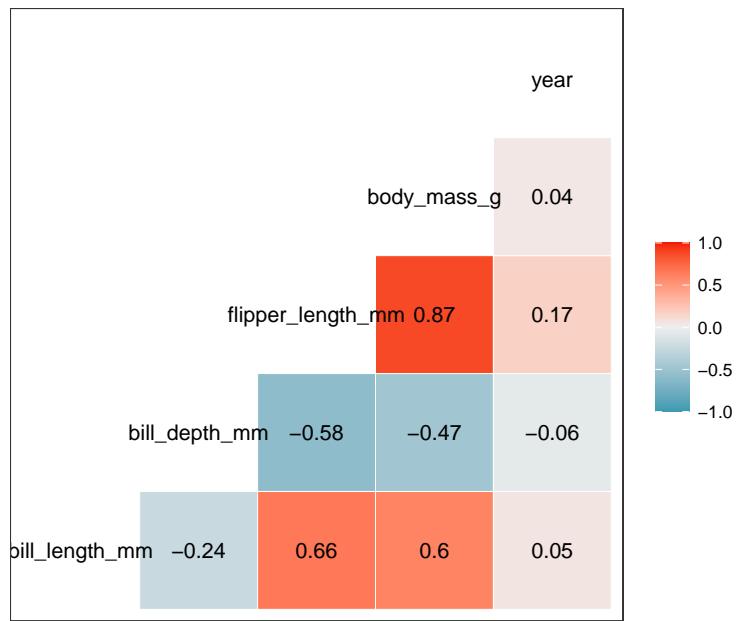
```
## Warning in ggcorr(penguins): data in column(s) 'species', 'island', 'sex' are
## not numeric and were ignored
```



```
# z wpisanymi wartościami korelacji
```

```
ggcorr(penguins, label=TRUE, label_color="black", label_round = 2)
```

```
## Warning in ggcorr(penguins, label = TRUE, label_color = "black", label_round
## = 2): data in column(s) 'species', 'island', 'sex' are not numeric and were
## ignored
```



Chapter 6

Wykresy - pakiet podstawowy

W pakiecie podstawowym każdy typ wykresu jest rysowany przy pomocy innej funkcji. Różne funkcje mogą wymagać innego typu danych do działania. Wspólne są natomiast parametry dotyczące wyświetlania i wyglądu wykresu.

Często, żeby uzyskać właściwy wygląd osi, legende musimy ją dodać samodzielnie przy pomocy funkcji `legend` lub `axis`.

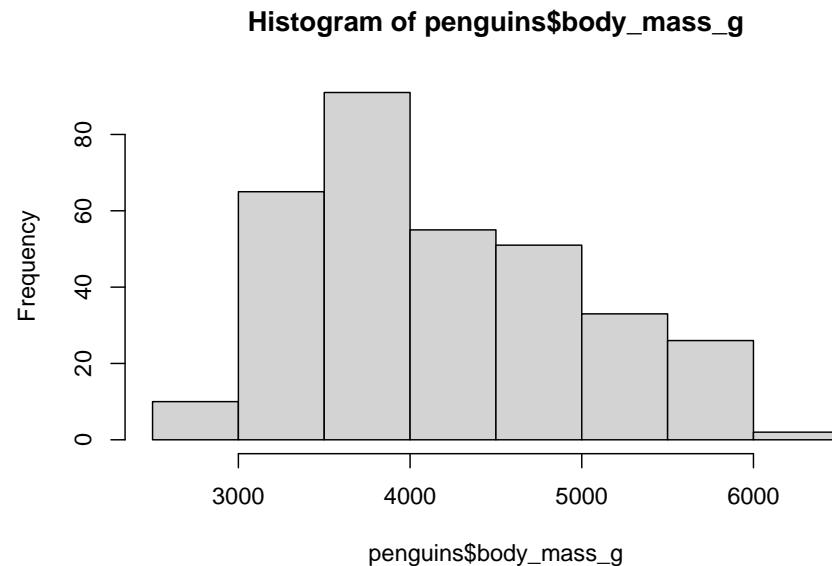
Funkcje pakietu podstawowego przydają się najbardziej, gdy trzeba coś szybko sprawdzić, gdyż wymagają mniej pisania niż analogiczne funkcje `ggplot2`. Jednakże przygotowanie ładnego wykresu korzystając tylko z pakietu podstawowego jest żmudne.

6.1 Histogram - `hist`

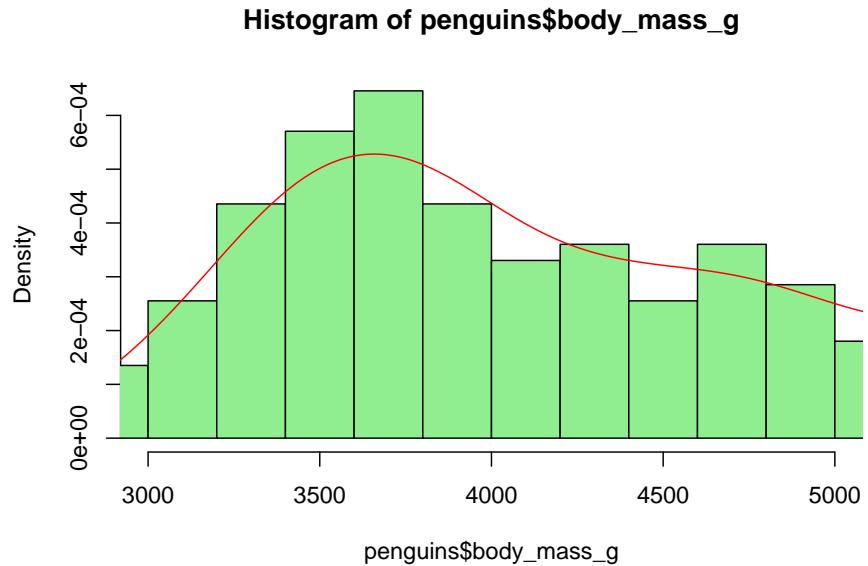
Histogram rysujemy funkcją `hist`. Należy podać wektor zawierający wartości, które mają zostać zliczone. Najważniejszym parametrem histogramu jest `breaks`, do którego można podać ilość słupków albo wektor przedziałów. `freq = TRUE` ozancza że na osi Y znajdę się zliczenie elementów w przedziałach, `FALSE` oznacza gęstości. Parametry `ylim` i `xlim` służą do zmiany startu i zakończenia osi.

Przy pomocy tej funkcji nie jest możliwe narysowanie histogramu z więcej niż jednego wektora. Możemy na niego nałożyć krzywą oznaczającą gęstość przy pomocy `lines(density(x))`. Możemy także nie rysować wykresu, ale otrzymać jego liczbową reprezentację używając `plot=FALSE`.

```
library(palmerpenguins)
penguins <- penguins %>% drop_na()
hist(penguins$body_mass_g)
```



```
hist(penguins$body_mass_g, freq = FALSE, breaks = 20, col = "lightgreen", xlim = c(3000,
```



```
# liczbowy opis histogramu - przedziały zliczenia, gęstości itd.
hist(penguins$body_mass_g, plot=FALSE)

## $breaks
## [1] 2500 3000 3500 4000 4500 5000 5500 6000 6500
##
## $counts
## [1] 10 65 91 55 51 33 26 2
##
## $density
## [1] 6.006006e-05 3.903904e-04 5.465465e-04 3.303303e-04 3.063063e-04
## [6] 1.981982e-04 1.561562e-04 1.201201e-05
##
## $mids
## [1] 2750 3250 3750 4250 4750 5250 5750 6250
##
## $xname
## [1] "penguins$body_mass_g"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

6.2 Wykres rozrzutu - plot

Wykres punktowy albo liniowy możemy otrzymać funkcją `plot`. Podajemy dwa wektory oznaczające współrzędne na osiach X i Y. Możemy określić typ wykresu `type`:

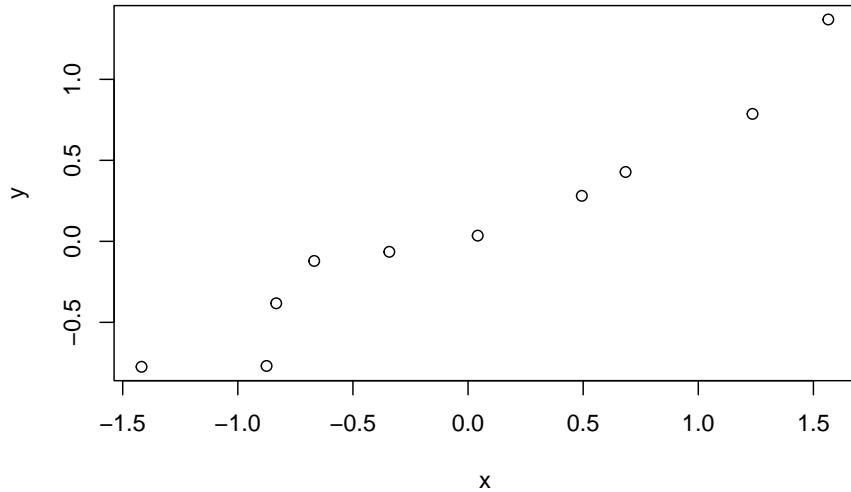
- p - punkty
- l - linia
- o - punkty i linia
- s, S - schodki
- h - linie trochę jak histogram
- n - brak

Linię trendu możemy dodać wykorzystując funkcje `lm` i `abline` (dla zależności liniowych) albo `lines`, `lm` i `predict` (nieliniowe).

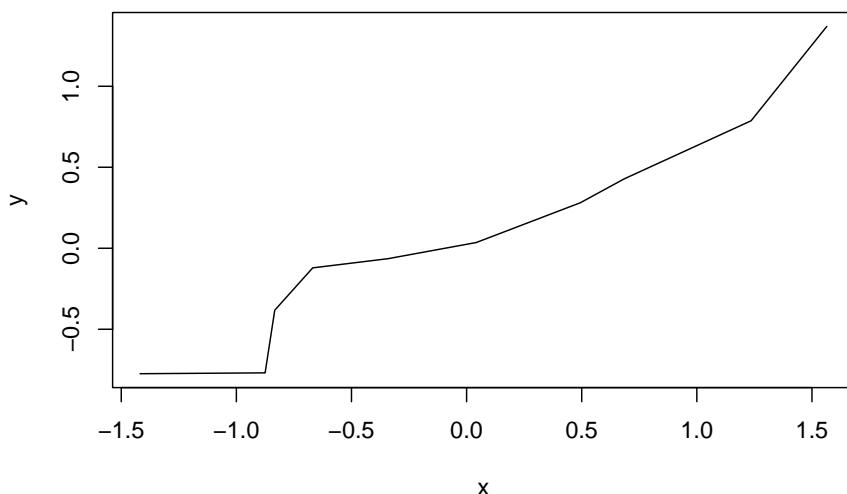
Używając `summary` i `lm` możemy wyświetlić wszystkie informacje dotyczące dopasowania.

```
x <- sort(rnorm(10))
y <- sort(rnorm(10))

plot(x,y)
```



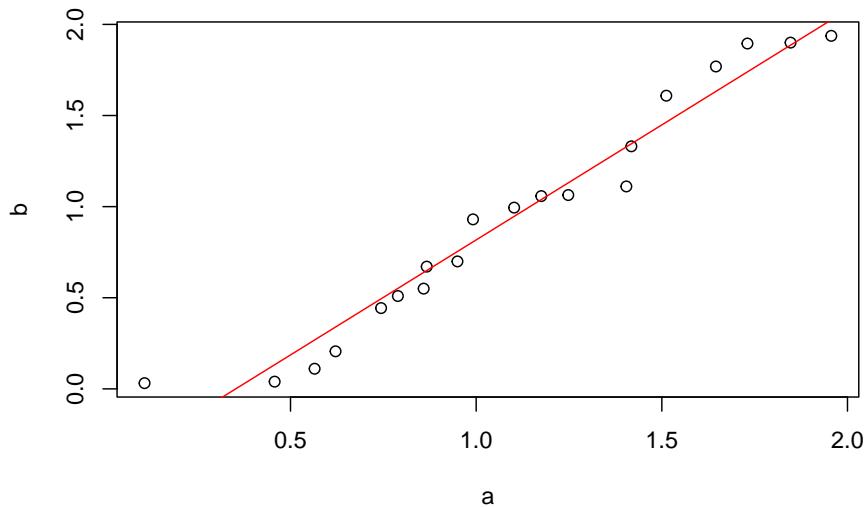
```
plot(x,y, type = "l")
```



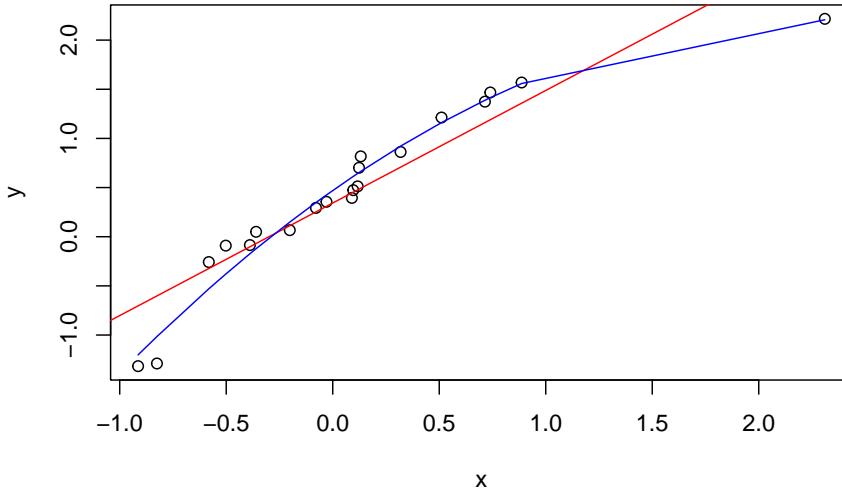
```
#posortowane, losowe liczby z rozkładu jednostajnego
a <- sort(runif(20))*2
b <- sort(runif(20))*2

plot(a,b)

# dopasowujemy prostą zależność b od a i rysujemy na wykresie
fit <- lm(b~a)
abline(fit, col = 'red')
```



```
# dane muszą być posortowane żeby działała funkcja lines
set.seed(100)
x <- sort(rnorm(20))
set.seed(300)
y <- sort(rnorm(20))
plot(x,y)
# dodajemy liniową linię trendu
abline(lm(y~x), col = "red")
# albo dopasowujemy wielomian drugiego stopnia
lines(x, predict(lm(y~poly(x, 2, raw = TRUE)), data.frame(x = x))), col = "blue")
```



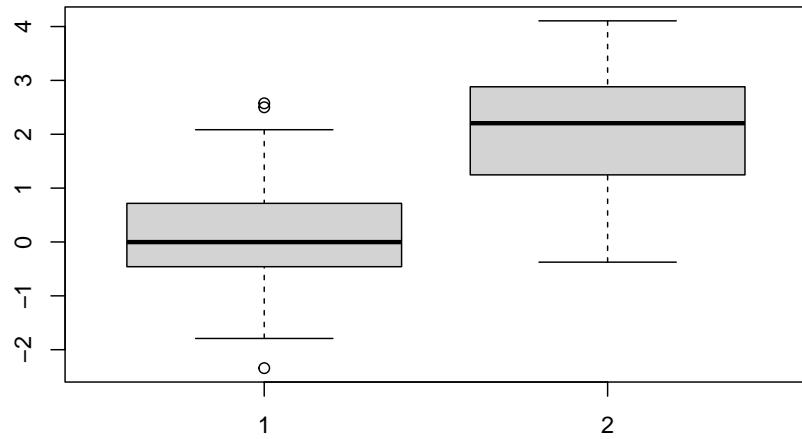
Więcej informacji na temat dopasowywania modelu do danych w części dotyczącej statystyki :)

6.3 Wykres pudełkowy - boxplot

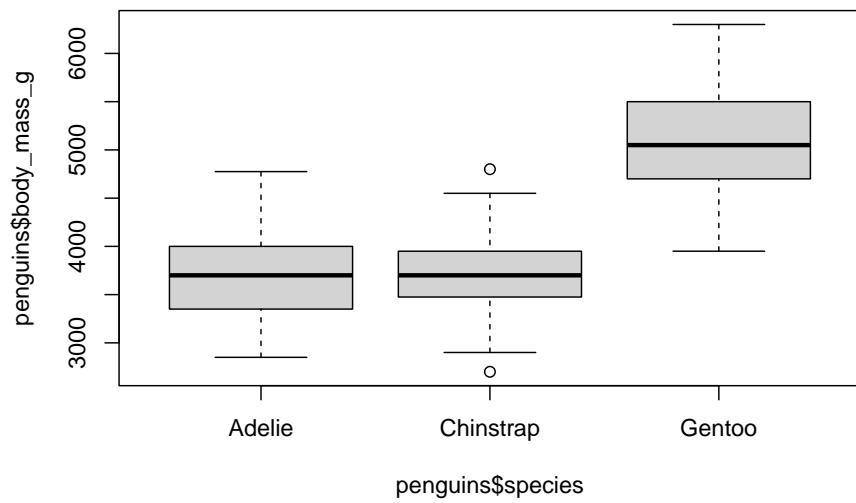
Używamy funkcji `boxplot`. Możemy podać jeden lub kilka wektorów, które posłużą do rysowania “pudełek” albo podobnie jak w `ggplot` jeden wektor z wartościami i jeden ze zmiennymi je grupującymi. Wcięcia w `boxplot`ach - `notch=TRUE`, wcięcia oznaczają przedział ufności dla mediany. Jeżeli wcięcia na siebie nie zachodzą to dwie populacje są od siebie najprawdopodobniej istotnie różne. `names` podpisy pod pudełkami.

```
# Przykładowe dane
x <- rnorm(100)
y <- rnorm(100, mean = 2)

boxplot(x, y)
```



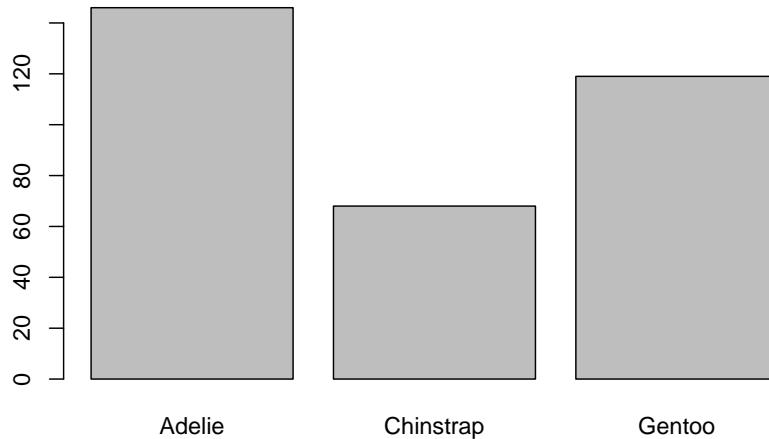
```
boxplot(penguins$body_mass_g ~ penguins$species)
```



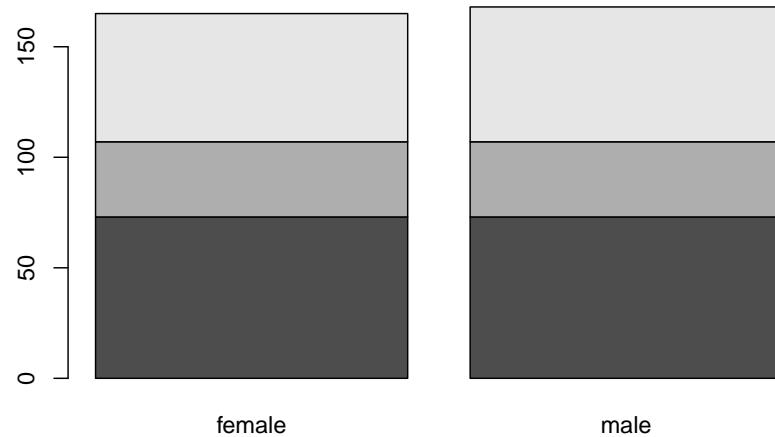
6.4 Wykres słupkowy - barplot

Rysowany przy pomocy funkcji `barplot`. Podajemy wektor albo matrycę wartości oznaczające wysokości słupków. Możemy szybko przygotować takie zliczenia funkcją `table`. W przypadku więcej niż jednego rodzaju słupków można określić czy mają być obok siebie - `beside=TRUE`, kolory zmieniamy parametrem `col`.

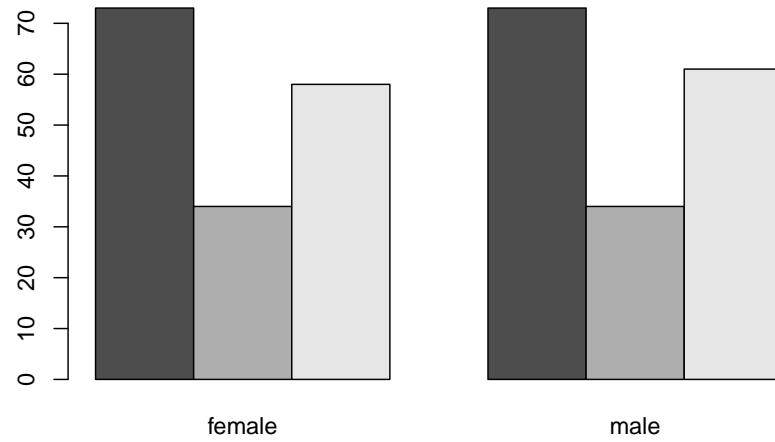
```
barplot(table(penguins$species))
```



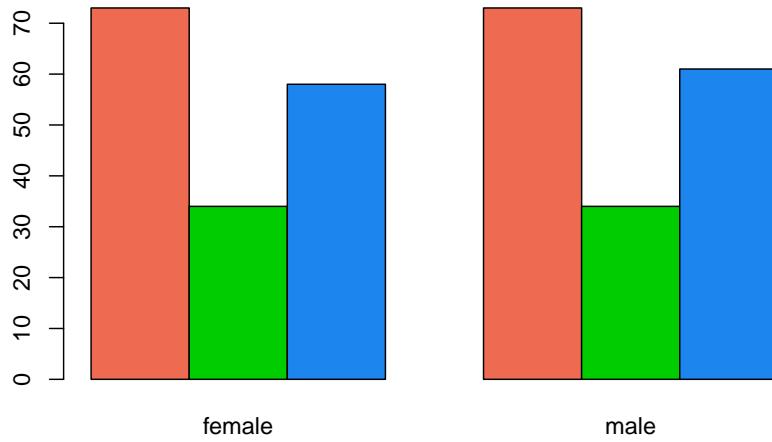
```
barplot(table(penguins$species, penguins$sex))
```



```
barplot(table(penguins$species, penguins$sex), beside = TRUE)
```



```
barplot(table(penguins$species, penguins$sex), beside = TRUE, col = c("coral2", "green3", "dodgerb
```

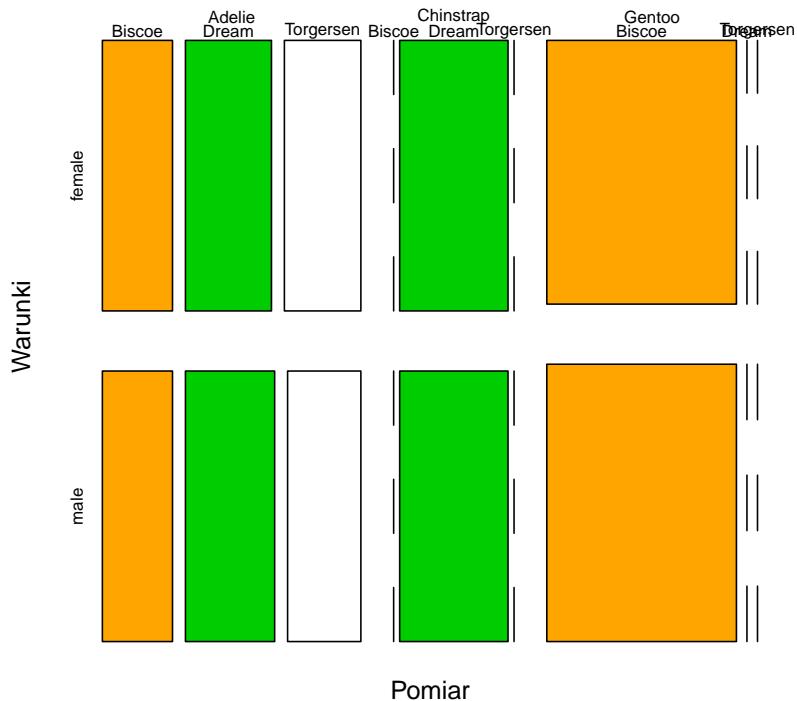


6.5 Wykres mozaikowy - mosaicplot

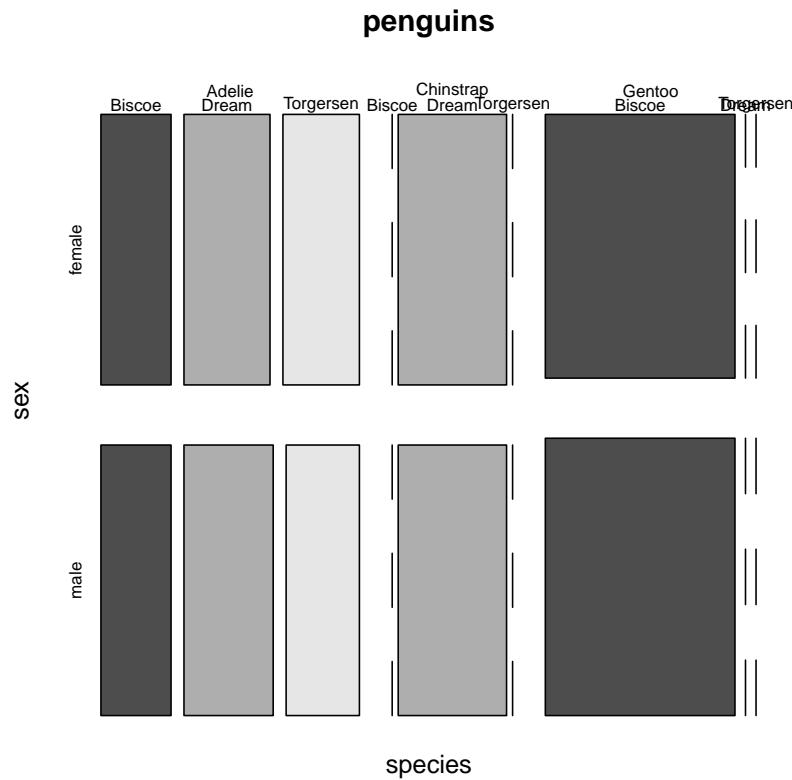
Na tym wykresie liczebności poszczególnych grup są reprezentowane przez pole powierzchni prostokąta. Można go przygotować nawet dla trzech lub więcej różnych zmiennych wyliczeniowych poprzez użycie funkcji `table` albo formuły.

```
# z funkcją table  
  
mosaicplot(table(penguins$species, penguins$sex, penguins$island),  
           col = c("orange", "green3", "white"), ylab = "Warunki", xlab = "Pomiar")
```

```
table(penguins$species, penguins$sex, penguins$island)
```



```
# z formuła  
mosaicplot(~species + sex + island, data = penguins, col = TRUE)
```



6.6 Wykres kwantylowy - qqPlot

Dopasowanie danych do rozkładu normalnego można wizualnie sprawdzić przy pomocy wykresu kwantylowego.

Funkcja qqPlot z pakietu car rysuje wykres kwantylowy z zaznaczonymi przedziałami ufności.

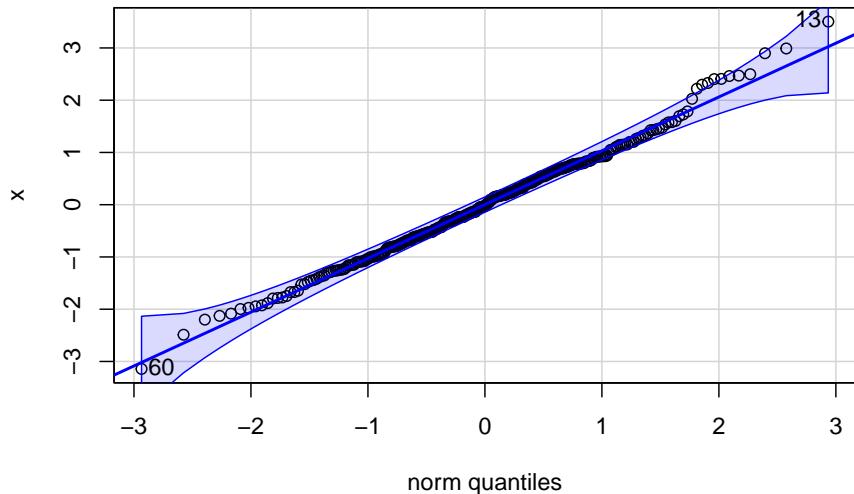
```
library(car)

## Loading required package: carData

## 
## Attaching package: 'car'
```

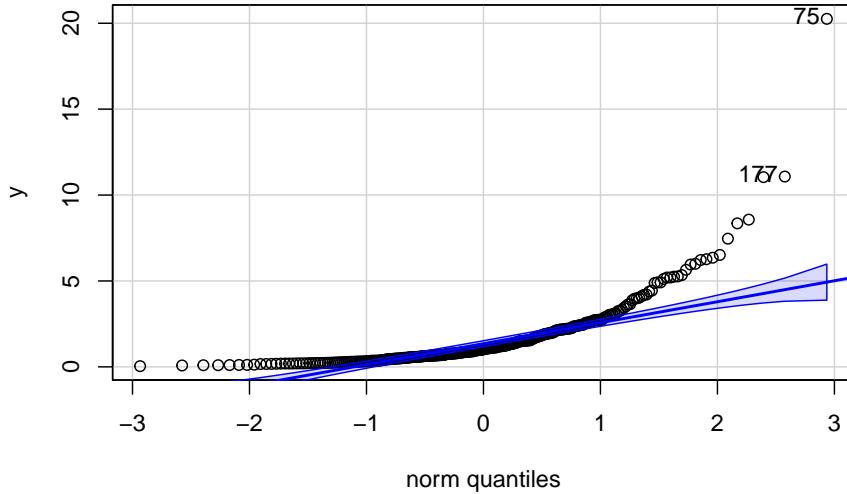
```
## The following object is masked from 'package:dplyr':  
##  
##      recode
```

```
# Dane z rozkładu normalnego  
x <- rnorm(300)  
# Dane z rozkładu lognormalnego  
y <- rlnorm(300)  
  
qqPlot(x)
```



```
## [1] 13 60
```

```
qqPlot(y)
```



```
## [1] 75 177
```

6.7 Diagram venna

Istnieje wiele różnych funkcji generujących diagramy venna. Funkcja `venn` z pakietu `gplots` wydaje się być jedną z łatwiejszych. Jej argumentem musi być lista (tworzona funkcją `list` - każdy kolejny element listy to może być coś innego: wektor, ramka danych, inna lista itp.). W tym wypadku kolejne elementy listy to wektory zawierające wartości zliczane do diagramu.

Trzeba zauważać, że wielkości obszarów nie odpowiadają liczebności grupy, może być nawet narysowany obszar z liczebnością 0.

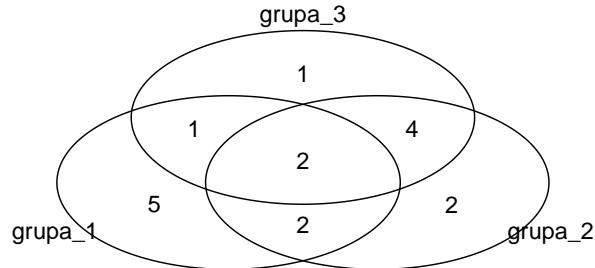
Można tworzyć diagramy z maksymalnie 5 grup, ale powyżej trzech stają się one trudne do odczytania, wtedy lepszą opcją jest narysowanie Upset plot

```
# generowanie zestawów losowych liter
set_1 <- sample(letters, 10)
set_2 <- sample(letters, 10)
set_3 <- sample(letters, 8)
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##      lowess

venn(list(grupa_1 = set_1, grupa_2 = set_2, grupa_3 = set_3))
```



Jeżeli chcielibyśmy mieć kolorowe diagramy należałoby użyć funkcji `venn.diagram` z pakietu `VennDiagram`. Ta funkcja stara się nie rysować obszarów, w których nic nie ma, efekt jej pracy jest również ładniejszy ;)

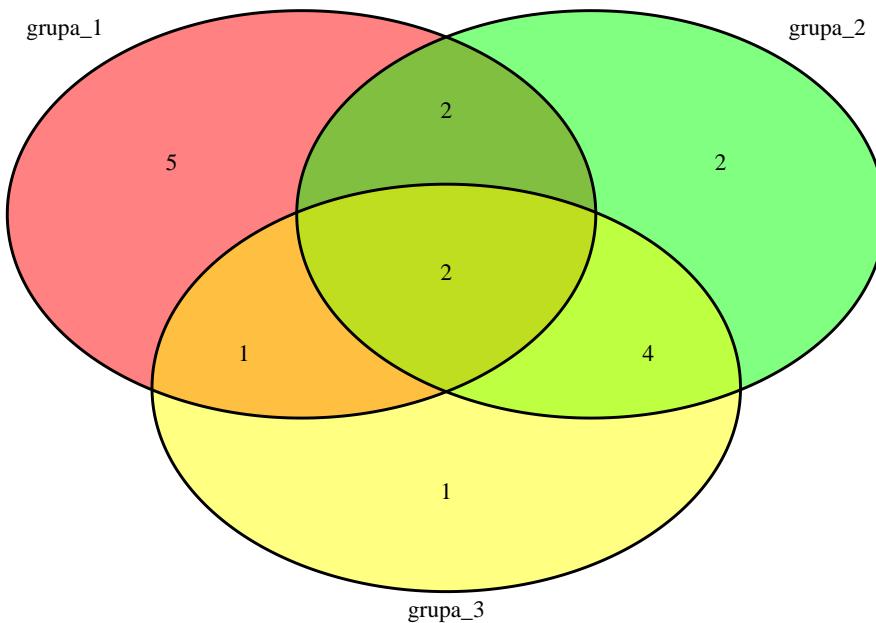
```
library(VennDiagram)

## Loading required package: futile.logger

##
## Attaching package: 'VennDiagram'

## The following object is masked from 'package:car':
##
##      ellipse
```

```
# Do wyświetlenia wymaga najpierw stworzenia obiektu typu grid
grid.newpage()
grid.draw(venn.diagram(list(grupa_1 = set_1, grupa_2 = set_2, grupa_3 = set_3),
                       fill = c("red", "green", "yellow"),
                       alpha = c(0.5, 0.5, 0.5),
                       filename = NULL))
```



6.8 Wykres korelacji - corrplot

Jeżeli dane zawierają kilka kolumn z różnymi zmiennymi do szybkiego sprawdzenie korelacji między nimi możemy użyć macierzy korelacji. Z ramki danych można ją wyznaczyć funkcją `cor`, natomiast wykres takiej macierzy możemy zrobić np. przy użyciu funkcji `corrplot` z pakietu `corrplot`.

```
# ładujemy dane dotyczące pogody
pogoda <- read.delim("data/pogoda.txt")

# tylko pierwsze cztery kolumny zawierają interesujące informacje
colnames(pogoda)

## [1] "ozon"          "naswietlenie"   "wiatr"         "temperatura"   "miesiac"
## [6] "dzien"
```

```
pogoda <- pogoda[, 1:4]
```

```
# dane zawierają brakujące informacje. Możemy je usunąć przy pomocy funkcji complete.cases
# zwraca numery wierszy nie zawierających wartości NA
pogoda <- pogoda[complete.cases(pogoda), ]
```

```
# przygotowujemy macierz korelacji
```

```
macierz <- cor(pogoda)
```

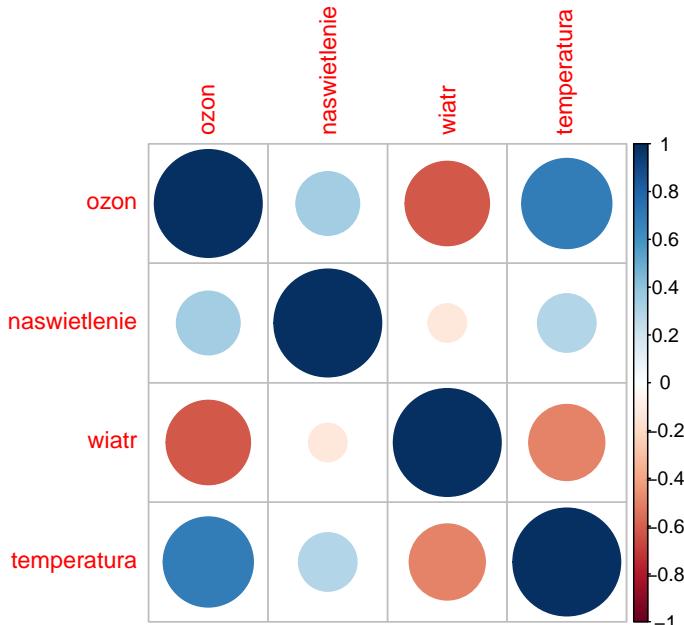
```
macierz
```

```
##          ozon naswietlenie      wiatr temperatura
## ozon      1.0000000   0.3483417 -0.6124966  0.6985414
## naswietlenie  0.3483417   1.0000000 -0.1271835  0.2940876
## wiatr      -0.6124966  -0.1271835  1.0000000 -0.4971897
## temperatura  0.6985414   0.2940876 -0.4971897  1.0000000
```

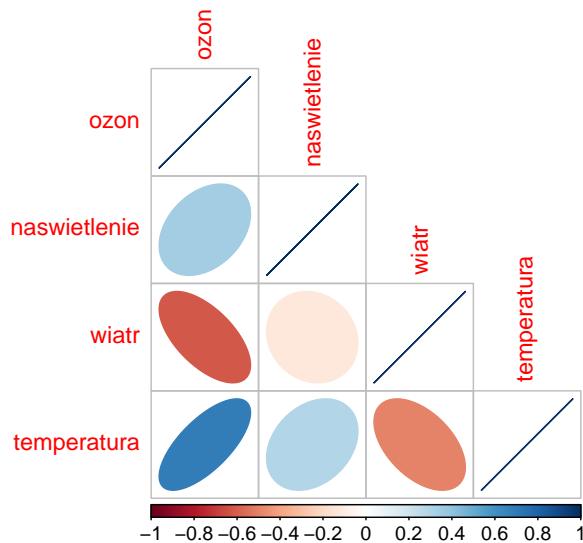
```
library(corrplot)
```

```
## corrplot 0.90 loaded
```

```
corrplot(macierz)
```



```
corrplot(macierz, method = "ellipse", type = "lower")
```



6.9 Clustering - analiza skupień

Wykorzystamy funkcję `kmeans`, która pozwala na grupowanie danych w n-liczbie klasterów. Opiera się na metodzie tzw. klasyfikacji bez nadzoru (unsupervised learning), tak aby uzyskać minimalną wariancję wewnętrz danej grupy.

Słabością tej metody jest to, że dostaniemy tyle grup ile sobie zażyczymy (niekoniecznie naprawdę istniejących), dlatego dobrze jest wypróbować kilka wartości n.

```
# Przygotowujemy dane złożone z dwóch grup
x <- c(rnorm(100,1), rnorm(100,3))
y <- c(rnorm(100,1), rnorm(100,3))

plot(x,y)
```



```

## Available components:
## [1] "cluster"      "centers"       "totss"        "withinss"      "tot.withinss"
## [6] "betweenss"    "size"          "iter"         "ifault"

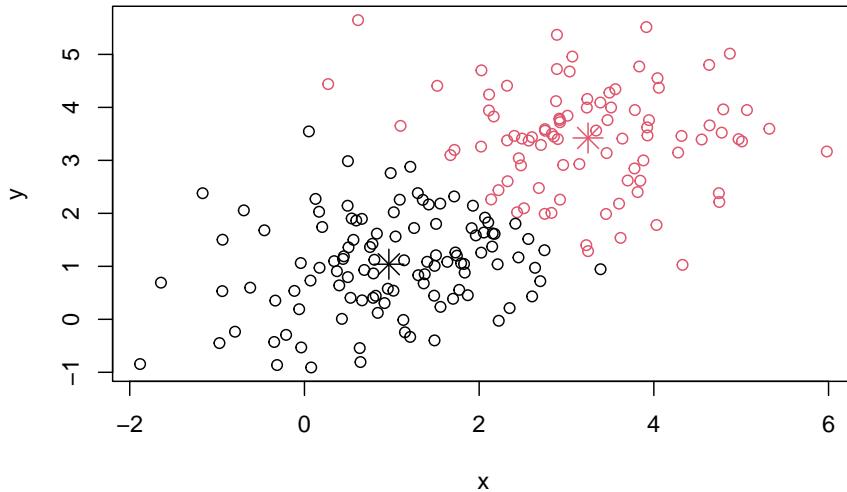
# możemy sprawdzić dopasowanie - wiemy jakie były grupy

test <- rep(c(2,1), each=100)
ok <- sum(test == clu$cluster)
ok/length(test)

## [1] 0.07

# Rysujemy ponownie wykres pokolorowany według wyznaczonych grup
plot(xy, col=clu$cluster)
# Dodajemy centra każdej z grup
points(clu$centers, col=1:2, pch=8, cex=2)

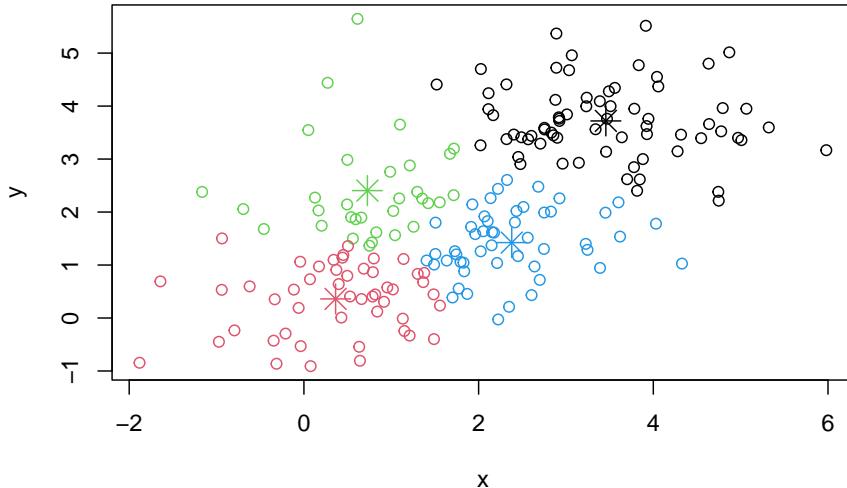
```



```

# Powtarzamy, ale z podziałem na 4 grupy
# Dane zawierały tylko 2 grupy, ale zostaną podzielone na 4
clu <- kmeans(xy, centers=4)
plot(xy, col=clu$cluster)
points(clu$centers, col=1:4, pch=8, cex=2)

```



Możemy także grupować dane według wielu zmiennych jednocześnie używając funkcji `dist` (redukuje wielowymiarowe dane do dwuwymiarowej macierzy odległości) i `hclust` (podział na grupy). Należy wybrać metodę grupowania, jedne z popularniejszych to `ward` i `complete`.

Wynik można przedstawić jako dendrogram korzystając ze zwykłej funkcji `plot`. Poszczególne grupy na wykresie zaznaczamy używając `rect.hclust`, musimy jedynie podać ilość grup.

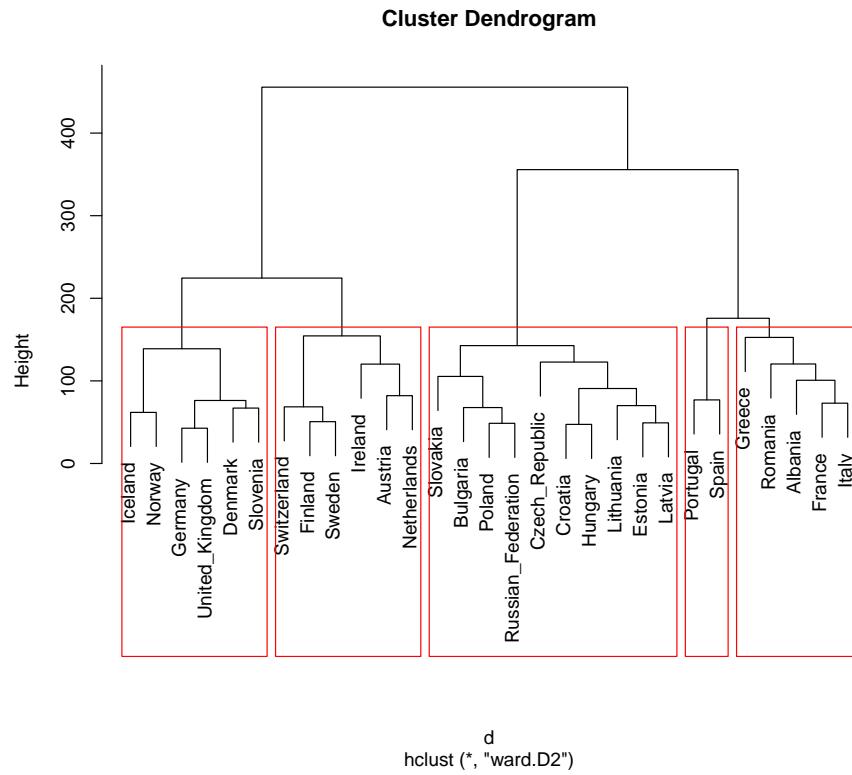
Wyświetlić grupy możemy za pomocą funkcji `cutree`, tutaj również niezbędne jest podanie ilości grup.

```
# Wykorzystamy dane spożycie z kursu R, spożycie produktów w różnych krajach
spozycie <- read.delim("data/spozycie.txt", row.names=1)

# Obliczamy macierz odległości
d <- dist(spozycie, method="euclidean")

# Grupujemy kraje
CA <- hclust(d, method="ward.D2")

# Dendrogram
plot(CA)
rect.hclust(CA, k=5, border="red")
```



```
# Podzieli krajów na 5 grup
grupy <- cutree(CA, k=5)
grupy
```

##	Albania	Austria	Bulgaria	Croatia
##	1	2	3	3
##	Czech_Republic	Denmark	Estonia	Finland
##	3	4	3	2
##	France	Germany	Greece	Hungary
##	1	4	1	3
##	Iceland	Ireland	Italy	Latvia
##	4	2	1	3
##	Lithuania	Netherlands	Norway	Poland
##	3	2	4	3
##	Portugal	Romania	Russian_Federation	Slovakia
##	5	1	3	3
##	Slovenia	Spain	Sweden	Switzerland
##	4	5	2	2
##	United_Kingdom			

Chapter 7

Statystyka

```
##  
## Robin Hankin: I'd say that without a tool like R you cannot learn statistics.  
## David Whiting: I believe Fisher and a few others managed to get by without it.  
## Peter Dalgaard: But think how far they could have got with R!  
## -- Robin Hankin, David Whiting, and Peter Dalgaard (on teaching/learning  
##      statistics with R)  
##      R-help (December 2004)
```

W tej części opieram się w dużej mierze na książce “Przewodnik po pakiecie R” P. Biecka oraz na kursie “Środowisko R od podstaw”. Nie ma ona być wprowadzeniem do statystyki jako takiej, ale jedynie pokazaniem jakie funkcje R można wykorzystać do przeprowadzenie często stosowanych analiz statystycznych.

Przystępne wprowadzenie do statystyki dla biologów znalazłam na stronie dotyczącej badania *C. elegans* - wormbook.org. Bardzo mało matematyki i wzorów, za to sporo przykładów, szkoda tylko, że nie ma nic o R ;)

7.1 Podstawowe statystyki opisowe

Skrócone podsumowanie danych liczbowych otrzymamy przy użyciu **summary** - średnia, mediana, wartość min i max, oraz pierwszy i trzeci kwantyl.

Podstawowe funkcje służące do opisu danych to:

Funkcja	Opis	Uwagi
mean	średnia	
median	mediana	

Funkcja	Opis	Uwagi
<code>sd</code>	odchylenie standardowe	
<code>var</code>	wariancja	
<code>min</code>	wartość minimalna	
<code>max</code>	wartość maksymalna	
<code>range</code>	zakres danych	
<code>IQR</code>	rozstęp kwartylowy	
<code>geometric.mean</code>	średnia geometryczna	
<code>weighted.mean</code>	średnia ważona	
<code>kurtosis</code>	kurtoza	musimy podać wektor wag do każdego elementu pakiet moments
<code>skewness</code>	skośność	pakiet moments
<code>mlv</code>	moda	pakiet modeest
<code>quantile</code>	wybrane kwantyle	należy podać, które kwantyle mają być policzone
<code>mad</code>	odchylenie medianowe	

```

wektor <- rnorm(1000, mean=2)

mean(wektor)

## [1] 2.008501

median(wektor)

## [1] 2.00913

range(wektor)

## [1] -1.253799  4.865792

quantile(wektor, c(0.25, 0.4, 0.5, 0.6, 0.75))

##      25%      40%      50%      60%      75%
## 1.271471 1.725972 2.009130 2.273003 2.737301

library(modeest)
mlv(wektor, method = 'shorth')

## [1] 1.868663

```

7.2 Liczby pseudolosowe

Podczas pracy w R często przydaje się możliwość szybkiego wygenerowania liczb z danego rozkładu. Można w ten sposób np. przetestować nową funkcję (również własną ;))

Wszystkie takie funkcje zaczynają się od r, a ich pierwszy argument to ilość liczb jaka ma zostać wygenerowana.

Jeżeli chcemy dwa razy wygenerować takie same liczby należy najpierw ustawić ziarno - `set.seed`

Funkcja	opis	parametry
<code>runif</code>	rozkład jednostajny, od 0 do 1	zmiana wartości <code>min</code> i <code>max</code> rozkładu
<code>rnorm</code>	rozkład normalny, średnia = 0, odchylenie = 1	zmiana <code>mean</code> i <code>sd</code>
<code>rlnorm</code>	rozkład log-normalny	zmiana <code>meanlog</code> i <code>sdlog</code>
<code>rexp</code>	rozkład wykładniczy	zmiana <code>rate</code>
<code>rbinom</code>	rozkład dwumianowy	ustawiamy wielkość <code>(size)</code> i prawdopodobieństwo <code>(prob)</code>

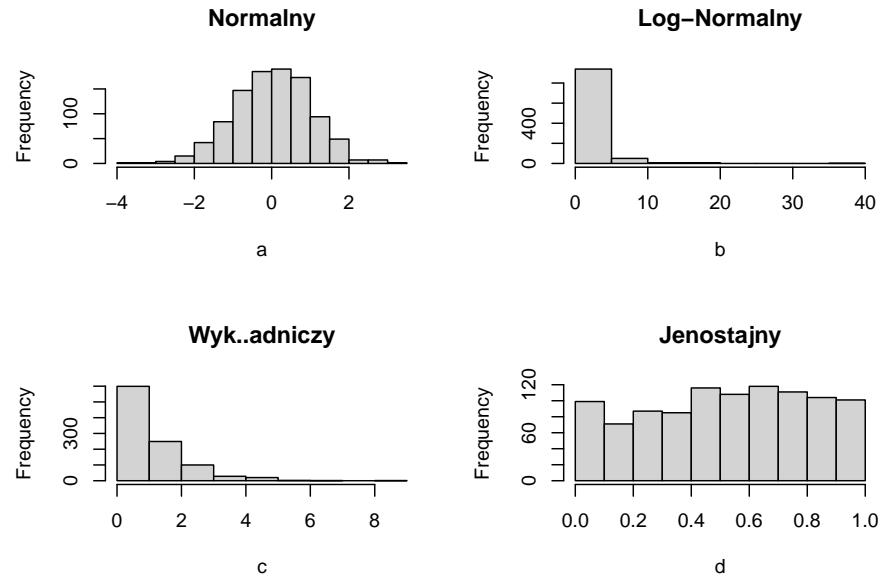
```
a <- rnorm(1000)
b <- rlnorm(1000)
c <- rexp(1000)
d <- runif(1000)

par(mfrow=c(2,2))
hist(a, main="Normalny")
hist(b, main = "Log-Normalny")
hist(c, main = "Wykładniczy")

## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...):
## conversion failure on 'Wykładniczy' in 'mbcsToSbccs': dot substituted for <c5>

## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...):
## conversion failure on 'Wykładniczy' in 'mbcsToSbccs': dot substituted for <82>

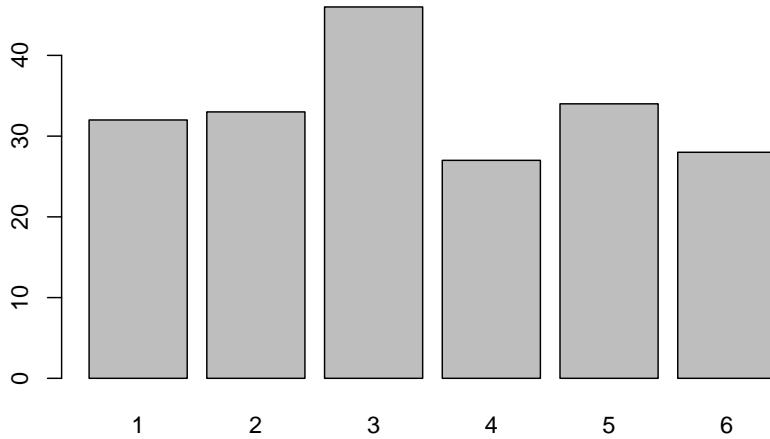
hist(d, main = "Jenostajny")
```



```
par(mfrow=c(1,1))
```

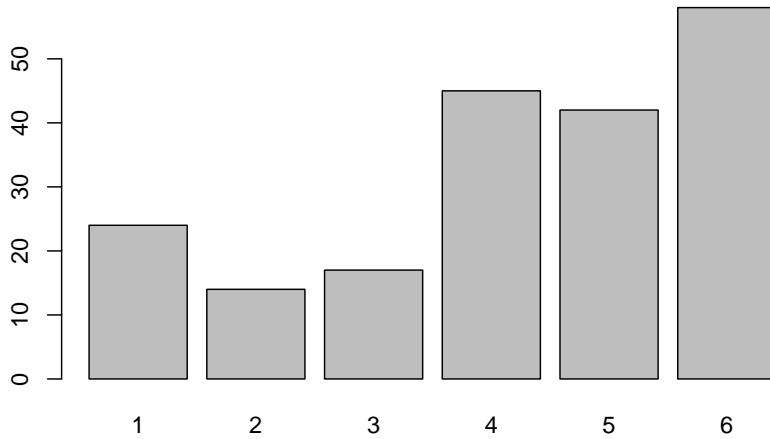
Możemy też potrzebować wektor losowych wartości z danego zakresu. Służy do tego funkcja `sample`. Pierwszym jej argumentem jest wektor, z którego mają być losowane wartości, drugim ilość elementów do losowania. Można też podać czy losowanie ma być ze zwracaniem (`replace=TRUE`) i wektor prawdopodobieństwa dla każdego elementu, jeżeli nie mają być takie same. Prawdopodobieństwa muszą sumować się do 1.

```
# Symulacja 200 rzutów kością
kosc <- sample(1:6, 200, replace = TRUE)
barplot(table(kosc))
```



Symulacja nieuczciwej kości

```
kosc2 <- sample(1:6, 200, replace = TRUE, prob = c(0.1, 0.1, 0.1, 0.2, 0.2, 0.3))
barplot(table(kosc2))
```



7.3 Podstawowe testy statystyczne

7.3.1 Test t-studenta - weryfikacja równości średnich

Można go wykonać dla jednej lub dwóch prób przy pomocy funkcji `t.test`. Dane muszą być z rozkładu normalnego, ale domyślnie nie muszą mieć równej wariancji.

Przy teście dla jednej próby należy podać wartość średniej `mu`, do której ma zostać przymierzaną próba (domyślnie wynosi 0).

Przy teście dla dwóch prób możemy również wykonać test dla prób sparowanych - `paired=TRUE` - sprawdzamy czy różnica między próbami jest różna od 0 np. dane przed i po dodaniu jakiegoś czynnika

Domyślnie wykonywany jest test dwustronny, możemy to zmienić parametrem `alternative` ustawiając "less" albo "greater".

Wynik testu podaje nam kilka wartości:

- wartość statystyki testowej
- ilość stopni swobody
- p-value - minimalny poziom istotności dla którego możemy odrzucić hipotezę zerową. Np. p-value równy 0.05 oznacza, że jeżeli odrzucimy hipotezę zerową istnieje 5% szans, że popełnimy błąd
- hipoteza alternatywna
- przedział ufności dla wyliczonej średniej albo różnicy między średnimi
- średnia z próby

Mogliśmy wyświetlić wszystkie te wartości albo jedynie interesujące nas poprzez znak \$. np. `t.test(x)$p.value`. Wynik takiego testu można też przypisać do zmiennej i wykorzystać później.

```
x <- rnorm(100)

y <- rnorm(100, mean=1)

# test dla jednej próby, porównanie do średnia równej 0
t.test(x)
```

```
##
## One Sample t-test
##
```

```
## data: x
## t = -0.31224, df = 99, p-value = 0.7555
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
## -0.2042882 0.1487354
## sample estimates:
## mean of x
## -0.02777643

t.test(y)

##
## One Sample t-test
##
## data: y
## t = 9.0505, df = 99, p-value = 1.287e-14
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
## 0.7738458 1.2084402
## sample estimates:
## mean of x
## 0.991143

# test dla dwóch prób
t.test(x,y)

##
## Welch Two Sample t-test
##
## data: x and y
## t = -7.2217, df = 190.02, p-value = 1.203e-11
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -1.2972245 -0.7406144
## sample estimates:
## mean of x mean of y
## -0.02777643 0.99114303

wynik <- t.test(x,y)
wynik$p.value

## [1] 1.203272e-11
```

```
wynik$conf.int

## [1] -1.2972245 -0.7406144
## attr(,"conf.level")
## [1] 0.95

# test t można przeprowadzić dla większej ilości grup przy pomocy pairwise.t.test,
# wartości p zostaną wtedy dostosowane do wielokrotnego powtarzania testu
pairwise.t.test(dane1$pomiar, dane1$Szczep)

## 
##  Pairwise comparisons using t tests with pooled SD
##
## data: dane1$pomiar and dane1$Szczep
##
##   A      B
## B <2e-16 -
## C <2e-16 <2e-16
##
## P value adjustment method: holm
```

Jeżeli nie wiemy z jakiego rozkładu pochodzą dane możemy wykorzystać test nieparametryczny - test Wilcoxona `wilcox.test`, też podaje wartość p.

Analogicznie dla `t.test` istnieje funkcja `pairwise.wilcox.test`

```
wilcox.test(x,y)

##
##  Wilcoxon rank sum test with continuity correction
##
## data: x and y
## W = 2379, p-value = 1.525e-10
## alternative hypothesis: true location shift is not equal to 0
```

7.3.2 Test F - weryfikacja równości wariancji

Podobny w składni do `t.test`, wykonujemy funkcją `var.test`. Również zakładamy, że dane pochodzą z rozkładu normalnego. Zastosowanie go dla danych z innych rozkładów może prowadzić do błędnych wniosków.

```
var.test(x,y)

##
## F test to compare two variances
##
## data: x and y
## F = 0.65984, num df = 99, denom df = 99, p-value = 0.0398
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
## 0.4439686 0.9806782
## sample estimates:
## ratio of variances
## 0.6598411

z <- rnorm(100, sd=3)

var.test(x,z)

##
## F test to compare two variances
##
## data: x and z
## F = 0.076738, num df = 99, denom df = 99, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
## 0.05163257 0.11405070
## sample estimates:
## ratio of variances
## 0.07673806
```

7.3.3 Testowanie zgodności z rozkładem normalnym i dopasowywanie parametrów rozkładu

Metoda graficzna to wspomniany już wykres kwantylowy (`qqPlot`).

Kilka testów, które można wykorzystać znajduje się w pakiecie `nortest` np. `shapiro.test`, `cvm.test`.

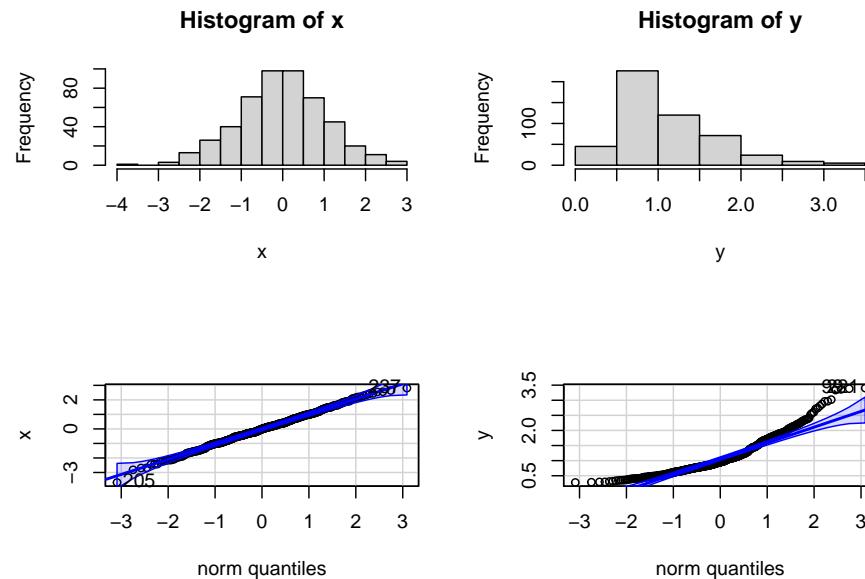
Test Shapiro-Wilka należy do bardziej popularnych, liczba obserwacji powinna mieścić się w zakresie od 3 do 5000.

```
x <- rnorm(500)
y <- rlnorm(500, sdlog = 0.5)
```

```
# histogram i wykres kwantylowy dla danych x i y
par(mfrow = c(2,2))
hist(x)
hist(y)
qqPlot(x)
```

```
## [1] 205 237
```

```
qqPlot(y)
```



```
## [1] 221 99
```

```
par(mfrow = c(1,1))
# test Shapiro-Wilka do sprawdzania zgodności z rozkładem normalnym
shapiro.test(x)
```

```
##
##  Shapiro-Wilk normality test
##
## data: x
## W = 0.99823, p-value = 0.8944
```

```
shapiro.test(y)

##
## Shapiro-Wilk normality test
##
## data: y
## W = 0.9017, p-value < 2.2e-16
```

Do sprawdzenie zgodności z zadanym rozkładem możemy wykorzystać test Kolmogorowa-Smirnowa - `ks.test`. Do funkcji należy podać wektor obserwacji oraz albo drugi wektor obserwacji (sprawdzamy czy pochodzą z takiego samego rozkładu) albo nazwę funkcji obliczającej dystrybuantę rozkładu np. `pnorm` (normalny), `plnorm` (log-normalny), `punif` (jednostajny).

Jeżeli parametry rozkładu różnią się od domyślnych to należy je podać np. `mean` i `sd` dla rozkładu normalnego.

```
x <- rnorm(1000)

x1 <- rnorm(1000)

x2 <- rnorm(1000, mean = 2)

y <- rlnorm(1000, sdlog = 0.5)

# Sprawdzamy czy obserwacje mają taki sam rozkład
ks.test(x,x1)
```

```
##
## Two-sample Kolmogorov-Smirnov test
##
## data: x and x1
## D = 0.037, p-value = 0.5004
## alternative hypothesis: two-sided
```

```
ks.test(x,x2)

##
## Two-sample Kolmogorov-Smirnov test
##
## data: x and x2
## D = 0.671, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
ks.test(x,y)

##
##  Two-sample Kolmogorov-Smirnov test
##
## data: x and y
## D = 0.64, p-value < 2.2e-16
## alternative hypothesis: two-sided

# Sprawdzamy czy x pochodzi z rozkładu normalnego czy log-normalnego
ks.test(x, pnorm)

##
##  One-sample Kolmogorov-Smirnov test
##
## data: x
## D = 0.022326, p-value = 0.7012
## alternative hypothesis: two-sided

ks.test(x, plnorm)

##
##  One-sample Kolmogorov-Smirnov test
##
## data: x
## D = 0.54985, p-value < 2.2e-16
## alternative hypothesis: two-sided

# Jeżeli nie podamy, że średnia x2 równa się 2 otrzymamy błędny wynik
ks.test(x2, pnorm)

##
##  One-sample Kolmogorov-Smirnov test
##
## data: x2
## D = 0.68387, p-value < 2.2e-16
## alternative hypothesis: two-sided

ks.test(x2, pnorm, mean = 2)

##
##  One-sample Kolmogorov-Smirnov test
```

```

##  

## data: x2  

## D = 0.020089, p-value = 0.8144  

## alternative hypothesis: two-sided

# Czy y pasuje do rozkładu normalnego czy log-normalnego  

ks.test(y, pnorm, sd = 0.5)

##  

## One-sample Kolmogorov-Smirnov test  

##  

## data: y  

## D = 0.7663, p-value < 2.2e-16  

## alternative hypothesis: two-sided

ks.test(y, plnorm, sd = 0.5

##  

## One-sample Kolmogorov-Smirnov test  

##  

## data: y  

## D = 0.02376, p-value = 0.6249  

## alternative hypothesis: two-sided

```

Do oszacowania parametrów rozkładu można wykorzystać funkcję `fitdistr` z pakietu MASS. Podajemy wektor obserwacji oraz funkcję gęstości rozkładu. Rozpoznawane funkcje to np. "normal", "log-normal", "exponential", "f", "t" itp.

```

x <- rnorm(1000, mean=20, sd=0.5)  

library(MASS)

##  

## Attaching package: 'MASS'  

##  

## The following object is masked from 'package:patchwork':  

##  

##     area  

##  

## The following object is masked from 'package:dplyr':  

##  

##     select

```

```

# obliczamy parametry rozkładu normalnego i log-normalnego dla danych x
fit <- fitdistr(x, "normal")
fit

##      mean          sd
##  20.02764577   0.51181914
##  ( 0.01618514) ( 0.01144462)

fit2 <- fitdistr(x, "log-normal")
fit2

##      meanlog        sdlog
##  2.9967861138   0.0256138133
##  (0.0008099799) (0.0005727423)

# sprawdzamy czy dane x faktycznie pochodzą z rozkładu o parametrach obliczonych wyżej
ks.test(x, pnorm, fit$estimate[1], fit$estimate[2])

## 
##  One-sample Kolmogorov-Smirnov test
##
## data: x
## D = 0.013869, p-value = 0.9906
## alternative hypothesis: two-sided

ks.test(x, pnorm, fit2$estimate[1], fit2$estimate[2])

## 
##  One-sample Kolmogorov-Smirnov test
##
## data: x
## D = 1, p-value < 2.2e-16
## alternative hypothesis: two-sided

y <- rlnorm(1000, sdlog = 0.4)
fitdistr(y, "log-normal")

##      meanlog        sdlog
##  -0.010771248   0.395992732
##  ( 0.012522390) ( 0.008854667)

```

7.3.4 Obliczanie przedziałów ufności dla średniej i błędu standardowego

Błąd standardowy obliczamy ze wzoru:

$$SEM = odch / \sqrt{n}$$

. Pokazuje estymowane odchylenie pomiędzy prawdziwą średnią populacji, a obliczoną dla próby.

W R nie ma funkcji liczącej błąd standardowy, ale można takie przeliczenie wykonać samemu albo napisać taką funkcję

```
x <- rnorm(300, mean=10, sd=0.5)

mean(x)

## [1] 10.04872

sd(x)

## [1] 0.5133524

SEM <- sd(x)/sqrt(length(x))

cat("Błąd standardowy wynosi", SEM)

## Błąd standardowy wynosi 0.02963841
```

Przedział ufności o istotności 0.95 mówi nam, że prawdopodobieństwo znalezienia estymowanego parametru w tym przedziale wynosi 95%.

Przedział ufności dla rozkładu normalnego przy poziomie istotności 0.95 wyliczamy ze wzoru:

$$ci = 1.96 * odch / \sqrt{n}$$

$$\text{średnia} + / - ci$$

1.96 to 0.975 kwantyl rozkładu normalnego.

Zamiast 1.96 możemy podstawić wartość wyliczoną z rozkładu t przy pomocy:
`qt(0.975, df)`

Przy dużej liczebności próby (ok. 300) wyjdzie na to samo.

```

set.seed(120)
x <- rnorm(100, mean=3, sd=0.75)

(srednia<-round(mean(x),2))

## [1] 3.02

round(sd(x),2)

## [1] 0.79

ci <- round(1.96*sd(x)/sqrt(length(x)),2)

lower <- srednia-ci
upper <- srednia+ci

cat("Średnia x wynosi", srednia, "w przedziale ufności", lower, upper)

## Średnia x wynosi 3.02 w przedziale ufności 2.86 3.18

ci <- round(qt(0.975, df=(length(x)-1))*sd(x)/sqrt(length(x)),2)

lower <- srednia-ci
upper <- srednia+ci

cat("Średnia x wynosi", srednia, "w przedziale ufności", lower, upper)

## Średnia x wynosi 3.02 w przedziale ufności 2.86 3.18

```

Przedział ufności dla średniej można też znaleźć w wyniku `t.test`.

7.3.5 Testowanie korelacji

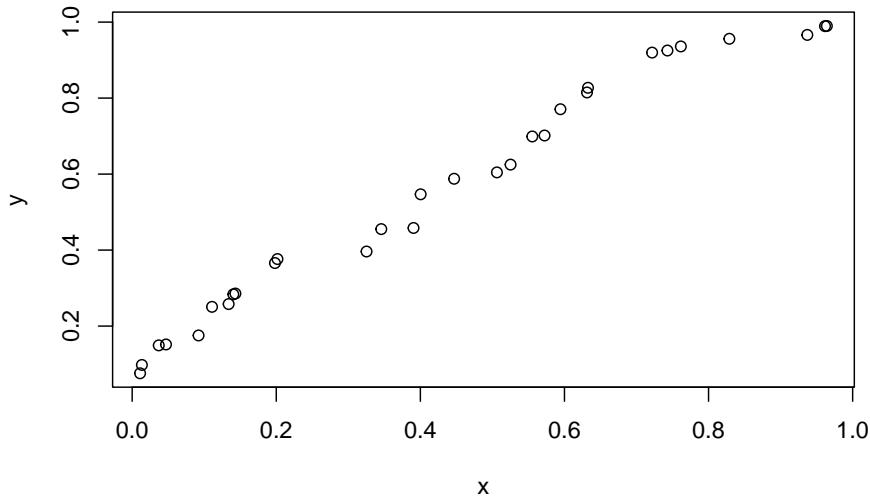
Wartość korelacji zawiera się pomiędzy -1 a 1. 0 oznacza całkowity brak korelacji. Do 0.7 korelację określamy jako silną.

Korelację obserwacji z dwóch wektorów albo całej macierzy można obliczyć przy pomocy funkcji `cor`. Domyślnie obliczona zostanie za pomocą metody Pearsona, można to zmienić na metodę Spearmana - `method="spearman"`. Metoda Spearmana jest mniej wrażliwa na obserwacje odstające.

Jeżeli chcemy poznać szczegóły dotyczące korelacji możemy użyć `cor.test`. Też można wybrać metodę, podaje również wartość p dla korelacji, przedział ufności, ilość stopni swobodi itp.

```
x <- sort(runif(30))
y <- sort(runif(30))

plot(x, y)
```



```
cor(x, y)

## [1] 0.9866882

cor(x, y, method = "spearman")

## [1] 1

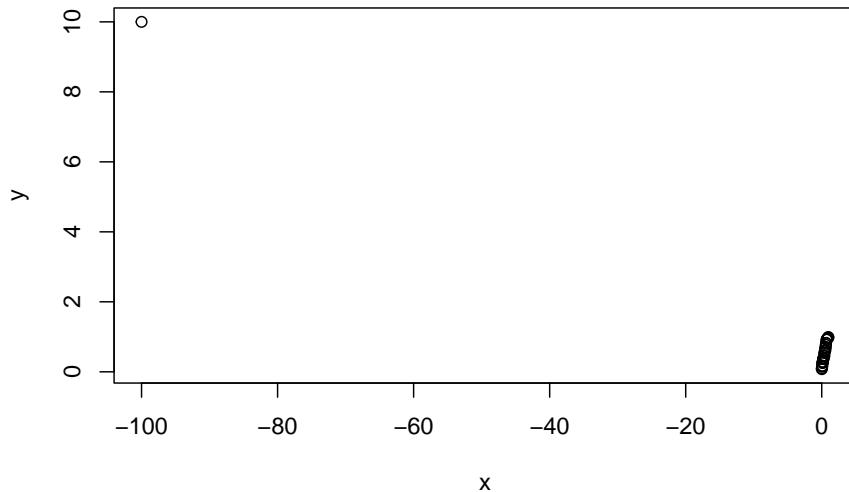
cor.test(x, y)

##
## Pearson's product-moment correlation
##
## data: x and y
## t = 32.105, df = 28, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
```

```
##  0.9719054 0.9937173
## sample estimates:
##      cor
## 0.9866882

# dodajemy wartość bardzo odstającą :)
x <- c(x, -100)
y <- c(y, 10)

plot(x, y)
```



```
# wartość korelacji wyliczona metodą spearmana zmienia się dużo mniej
cor(x, y)
```

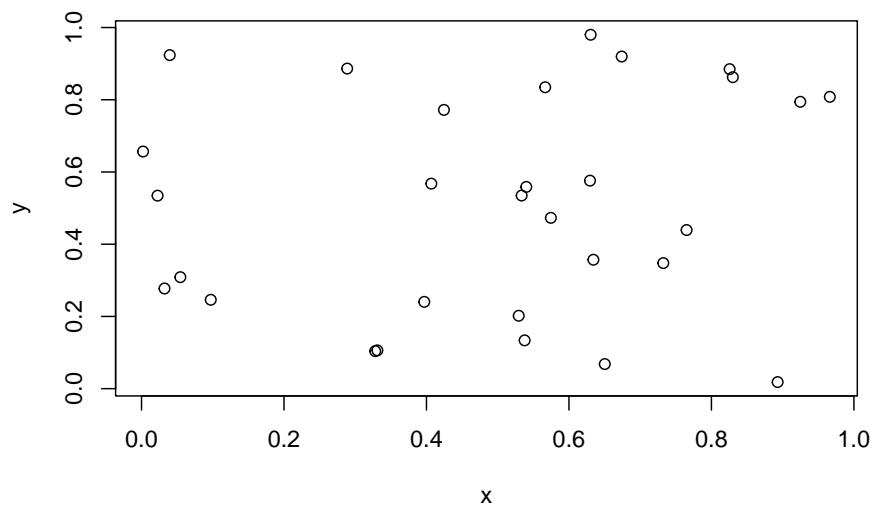
```
## [1] -0.9816699
```

```
cor(x, y, method = "spearman")
```

```
## [1] 0.8125
```

```
# wartość odstająca może też dać korelację pearsona tam gdzie jej wcale nie ma
# ale korelacja spearmana nie zmieni się tak bardzo
```

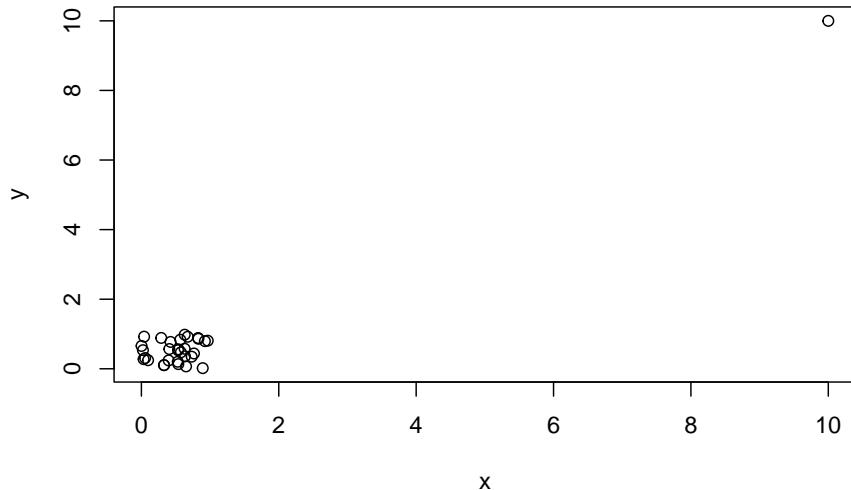
```
x <- sort(runif(30))
y <- runif(30)
plot(x, y)
```



```
cor(x, y)
```

```
## [1] 0.1557759
```

```
x <- c(x, 10)
y <- c(y, 10)
plot(x, y)
```



```
cor(x, y)
```

```
## [1] 0.9762845
```

```
cor(x, y, method="spearman")
```

```
## [1] 0.2383065
```

7.3.6 Test chi-kwadrat - zgodność rozkładu zmiennych jakościowych

Służy do weryfikacji zależności pomiędzy dwiema zmiennymi jakościowymi, funkcja `chisq.test`. Jako argument najlepiej podstawić macierz kontyngencji wyliczoną funkcją `table`. Poza wartością `p` można też sprawdzić wartości oczekiwane - `wynik$expected`. Test chi² można wykorzystać też dla większych macierzy.

Dla tablic 2x2 można również użyć dokładnego testu Fishera - `fisher.test`.

```
# Wygenerujemy przykładowe dane - takie same
x <- data.frame(jeden = sample(c("A", "B"), 200, replace = TRUE),
```

```
dwa = sample(c("grupa1","grupa2"),200, replace = TRUE))

# różne

y <- data.frame(jeden = c(sample(c("A","B"),100, replace = TRUE, prob = c(0.4,0.6)),
                         sample(c("A","B"),100, replace = TRUE, prob = c(0.7,0.3))),
                 dwa = rep(c("grupa1","grupa2"),each = 100))

(tabla_x <- table(x))

##      dwa
## jeden grupa1 grupa2
##      A     45     50
##      B     52     53

(tabla_y <- table(y))

##      dwa
## jeden grupa1 grupa2
##      A     40     77
##      B     60     23

chisq.test(tabla_x)

##
## Pearson's Chi-squared test with Yates' continuity correction
##
## data: tabla_x
## X-squared = 0.02654, df = 1, p-value = 0.8706

chisq.test(tabla_y)

##
## Pearson's Chi-squared test with Yates' continuity correction
##
## data: tabla_y
## X-squared = 26.691, df = 1, p-value = 2.387e-07

fisher.test(tabla_x)

##
```

```

## Fisher's Exact Test for Count Data
##
## data: tabela_x
## p-value = 0.7786
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
## 0.5068482 1.6595976
## sample estimates:
## odds ratio
## 0.9177044

fisher.test(tabela_y)

##
## Fisher's Exact Test for Count Data
##
## data: tabela_y
## p-value = 1.675e-07
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
## 0.1025177 0.3835088
## sample estimates:
## odds ratio
## 0.2009083

```

Jeżeli chcemy tylko sprawdzić prawdopodobieństwo możemy użyć testu propocji - `prop.test`

```

# przykładowo czy 785 sukcesów na 1500 prób jest istotnie różne od p=0.5

prop.test(785, 1500, p = 0.5)

```

```

##
## 1-sample proportions test with continuity correction
##
## data: 785 out of 1500, null probability 0.5
## X-squared = 3.174, df = 1, p-value = 0.07482
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
## 0.4976972 0.5488486
## sample estimates:
##          p
## 0.5233333

```

```
# można też wykonać test proporcji dla większej ilości danych - pairwise.prop.test

x <- c(190, 475, 350, 65)
n <- c(500, 1000, 800, 250)

pairwise.prop.test(x, n, p.adjust.method = "bonferroni")

##
## Pairwise comparisons using Pairwise comparison of proportions
##
## data: x out of n
##
##   1     2     3
## 2 0.0035 -
## 3 0.2803 0.7427 -
## 4 0.0086 7.8e-09 4.8e-06
##
## P value adjustment method: bonferroni
```

7.4 Modelowanie - czyli jak dopasować linię trendu

Zależność pomiędzy dwiema zmiennymi ilościowymi najłatwiej przedstawić na wykresie rozrzutu. Kolejnym krokiem może być próba dopasowania do danych jakiegoś modelu np. liniowego, ale może być też bardziej złożony np. model wzrostu logistycznego, Michaelisa-Menten itp.

7.4.1 Regresja liniowa

Najbardziej popularną funkcją w R do wyznaczania modeli liniowych jest `lm`. Wymaga jedynie podania formuły opisującej modelowaną przez nas zależność i ramki danych.

```
# Jako przykład wykorzystamy zbiór danych R dotyczący wzrostu drzewek pomarańczowych

orange <- Orange

head(orange, 3)

## Grouped Data: circumference ~ age | Tree
##   Tree age circumference
```

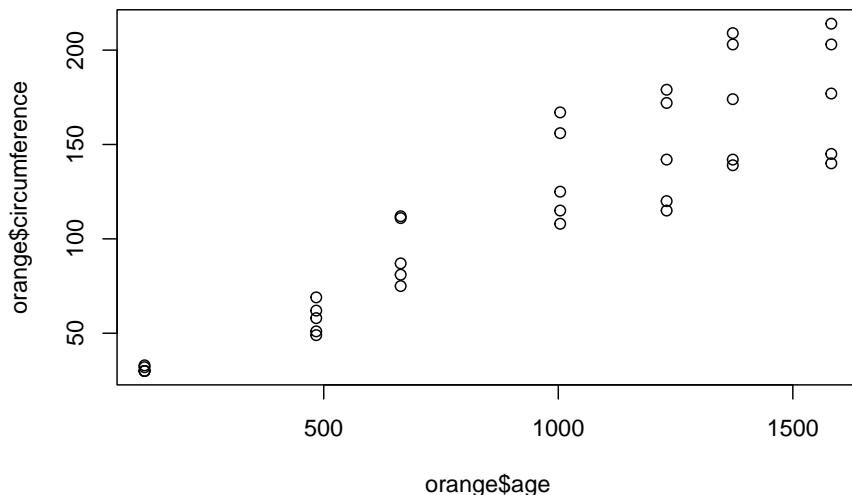
```

## 1    1 118      30
## 2    1 484      58
## 3    1 664      87

# wykres obwodu drzewa od wieku

plot(orange$age, orange$circumference)

```



Formuły zapisujemy korzystając z ~ , np.: * zależność y od x: $y \sim x$, * zależność y od x i z: $y \sim x + z$, * zależność y od x, z i interakcji pomiędzy x i z: $y \sim x + z + x:z$ albo $y \sim x * z$

```
fit <- lm(circumference~age, data = orange)
```

```
fit$coefficients
```

```

## (Intercept)      age
## 17.3996502   0.1067703

```

```
summary(fit)
```

```

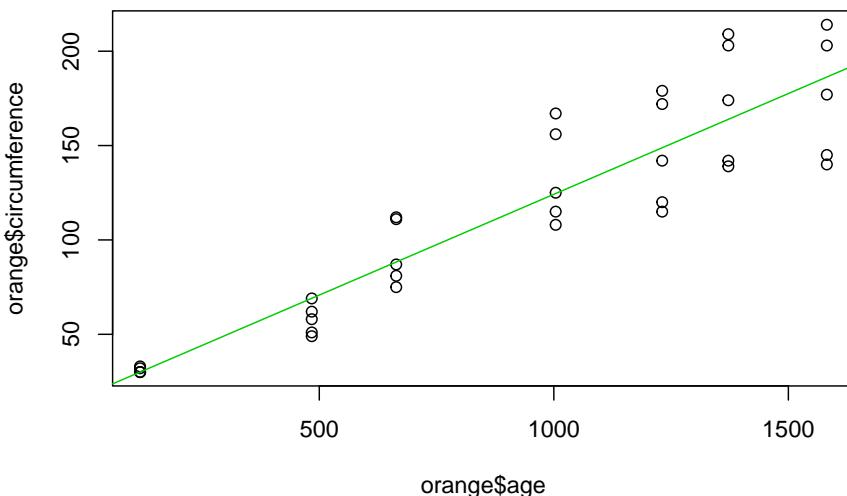
##
## Call:
##
```

```

## lm(formula = circumference ~ age, data = orange)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -46.310 -14.946 -0.076 19.697 45.111
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.399650   8.622660   2.018   0.0518 .
## age         0.106770   0.008277 12.900 1.93e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.74 on 33 degrees of freedom
## Multiple R-squared:  0.8345, Adjusted R-squared:  0.8295
## F-statistic: 166.4 on 1 and 33 DF,  p-value: 1.931e-14

plot(orange$age, orange$circumference)
abline(fit, col = "green3")

```



Jeżeli chcemy jedynie wyznaczyć równanie funkcji liniowej - $y = ax + b$, wystarczy sprawdzić współczynniki dopasowanego modelu - coefficients. Intercept oznacza miejsce przecięcia z osią Y - współczynnik b, a age to wartość przez którą należy pomnożyć wiek drzewa żeby uzyskać jego obwód.

Można powiedzić że według naszego modelu każdego roku obwód drzewa zwiększa się o wartość 0.107 mm.

Funkcja `summary` pozwala zobaczyć wszystkie istotne informacje dotyczące naszego modelu.

Pierwsza linijka podsumowania zawiera powtóżenie formuły jaką podaliśmy w funkcji.

Następnie mamy podsumowanie wartości residuals - reszt. Są to różnice pomiedzy faktycznymi wartościami a wyznaczonymi ze wzoru. Model jest dobierany tak żeby suma kwadratów reszt była jak najmniejsza.

Potem mamy tabelę z wymienionymi wszystkimi współczynnikami modelu. Istotna jest ostatnia kolumna zawierająca wartość p, wartość powyżej 0.05 sugeruje że dana zmienna nie jest istotna dla modelu i można ją z niego usunąć.

Na końcu otrzymujemy jeszcze kilka wartości oceniających dobrany model jako całość. Znajoma powinna być wartość R-squared. Przyjmuje ona wartości z zakresu od 0 do 1 i oznacza procent wariancji obecnej w danych jaka może być wyjaśniona przy pomocy danego modelu. Może być przydatna przy ocenie który z kilku modeli wybrać do opisu naszych danych. Wartość adjusted R squared uwzględnia również ilość zmiennych jakie podaliśmy w formule.

7.4.2 Zmienne jakościowe w modelu

W modelu możemy uwzględniać również zmienne jakościowe, muszą one jednak zostać zakodowane - zmienione w dane liczbowe. Funkcja `lm` domyślnie uznaje pierwszy poziom zmiennej za referencyjny, kolejne poziomy będą się do niego odnosić.

```
# zbiór danych dotyczący wzrostu kurczaków w zależności od rodzaju karmy
chick <- ChickWeight

summary(lm(weight ~ Time + Diet, data = chick))

##
## Call:
## lm(formula = weight ~ Time + Diet, data = chick)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -136.851  -17.151   -2.595   15.033  141.816 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  344.500    13.995  24.638  <2e-16 ***
## Time        -13.290     3.735  -3.568  0.0012 ** 
## Diet         52.400     4.323  12.087  <2e-16 ***
## 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

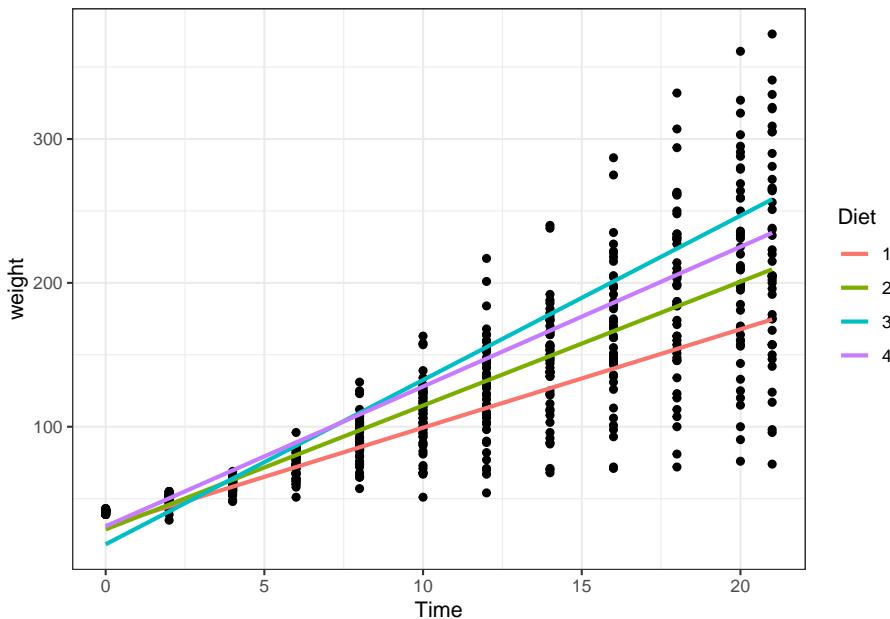
```

## (Intercept) 10.9244    3.3607    3.251  0.00122 **
## Time        8.7505    0.2218 39.451 < 2e-16 ***
## Diet2       16.1661    4.0858    3.957 8.56e-05 ***
## Diet3       36.4994    4.0858    8.933 < 2e-16 ***
## Diet4       30.2335    4.1075    7.361 6.39e-13 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 35.99 on 573 degrees of freedom
## Multiple R-squared: 0.7453, Adjusted R-squared: 0.7435
## F-statistic: 419.2 on 4 and 573 DF, p-value: < 2.2e-16

p <- ggplot(chick, aes(x = Time, y = weight))
p + geom_point() + stat_smooth(aes(color = Diet), method = "lm", se = FALSE, size = 1)

## `geom_smooth()` using formula 'y ~ x'

```



Wartości przy kolejnych typach karmy - Diet2, Diet3, Diet4, pokazują o ile średnio różnią się wartości wyznaczone dla nich od poziomu referencyjnego czyli Diet1. To znaczy, że kurczaki karmione karmą nr 2 są o 16.17 g cięższe od kurczaków karmionych karmą nr 1.

7.4.3 Pakiet drc - dose response models

Pakiet drc zawiera szereg funkcji ułatwiających dopasowywanie do danych wiele modeli popularnych w biologii np. wzrost logistyczny, rozpad eksponencjalny, Michaelisa-Menten i inne.

Podstawową funkcja to `drm`, w której podajemy formułę, podobnie jak w funkcji `lm` oraz koniecznie funkcję startową, która ma zostać użyta do poszukiwania modelu. Funkcje o kolejnych numerach różnią się ilością parametrów, które są uznawane za stałe np. MM.2 zakłada że przy $x = 0$, y też jest równe 0.

Model	Funkcja startowa
Logistyczny	LL.2, LL.3, LL.4, LL.5
Michaelisa-Menten	MM.2, MM.3
Eksponencjalny	EXD.2, EXD.3

Wykres razem z dopasowaniem można uzyskać dzięki standardowej funkcji `plot`. Można również porównać kilka modeli podając wektor ze zmienią dzielącą dane na poziomy.

```
library(drc)

##
## 'drc' has been loaded.

## Please cite R and 'drc' if used for a publication,
## for references type 'citation()' and 'citation('drc')'.

##
## Attaching package: 'drc'

## The following objects are masked from 'package:stats':
##      gaussian, getInitial

# przykładowe dane

dane <- data.frame(stezenie = c( 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.8, 1, 1.5),
                     predkosc = c( 0.3, 0.5, 0.8, 0.9, 1, 1.02, 1.09, 1.15, 1.23))

model <- drm(predkosc~stezenie, data = dane, fct = MM.2())
```

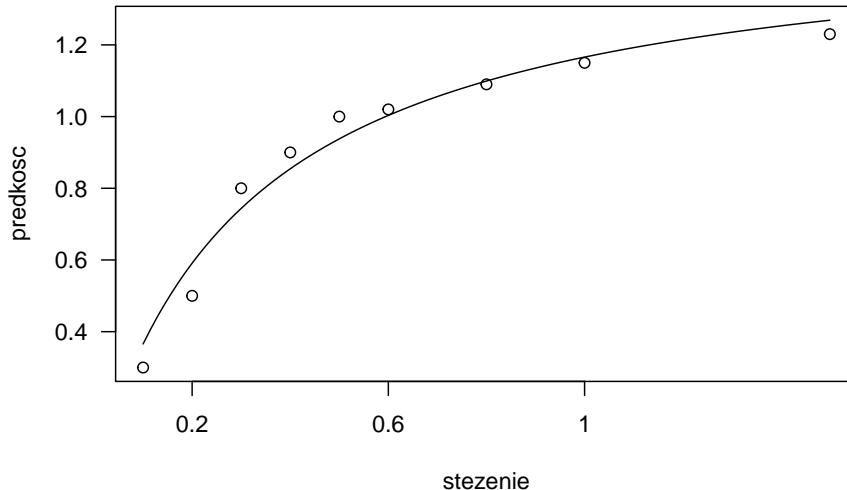
```

# d to prędkość maksymalna, e to Km
# w podsumowaniu dostajemy błąd standardowy i wartość p
summary(model)

## 
## Model fitted: Michaelis-Menten (2 parms)
##
## Parameter estimates:
##
##           Estimate Std. Error t-value p-value
## d:(Intercept) 1.540562  0.082354 18.7066 3.098e-07 ***
## e:(Intercept) 0.321284  0.047915  6.7053 0.0002761 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error:
##
## 0.05833157 (7 degrees of freedom)

plot(model, log="")

```



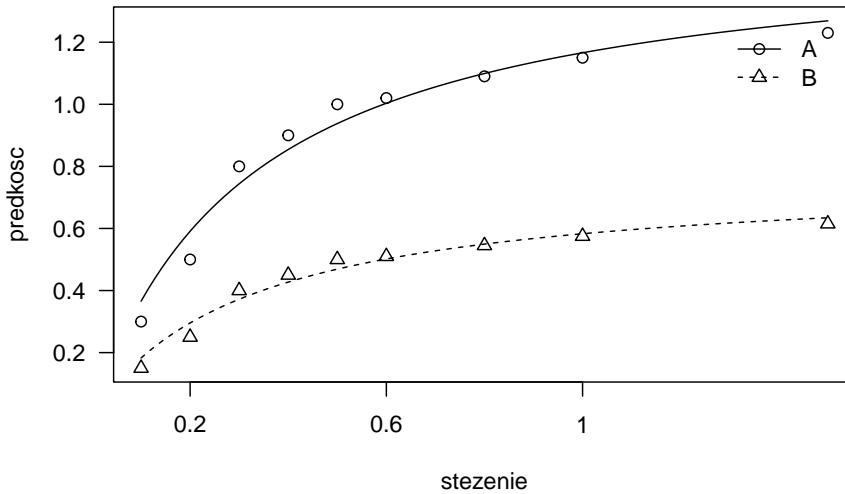
```
# przykładowe dane do porównania np. dwóch enzymów
dane1 <- dane
dane1$predkosc <- dane1$predkosc/2
dane <- rbind(dane, dane1)
dane$poziom <- rep(c("A", "B"), each = 9)

# model z dodanym podziałem na poziomy, możemy też nazwać oznaczane parametry
model <- drm(predkosc~stezenie, poziom, data = dane, fct = MM.2(names = c(d = "Vmax", e = "Km", i = "I0"), n = 2))

summary(model)

##
## Model fitted: Michaelis-Menten (2 parms)
##
## Parameter estimates:
##
##           Estimate Std. Error t-value p-value
## Vmax:A    1.540560  0.065106 23.6622 1.089e-12 ***
## Vmax:B    0.770280  0.065106 11.8311 1.122e-08 ***
## Km:A      0.321282  0.037880  8.4816 6.889e-07 ***
## Km:B      0.321282  0.075759  4.2408 0.0008227 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error:
##
## 0.04611515 (14 degrees of freedom)

plot(model, log="")
```



```
# Podobnie wygląda dopasowywanie funkcji logistycznej
# Korzystamy z przykładowych danych z pakietu drc dla wpływu herbicydów na tylakoidy

spinach.m1 <- drm(SLOPE~DOSE, CURVE, data = spinach, fct = LL.4())

summary(spinach.m1)
```

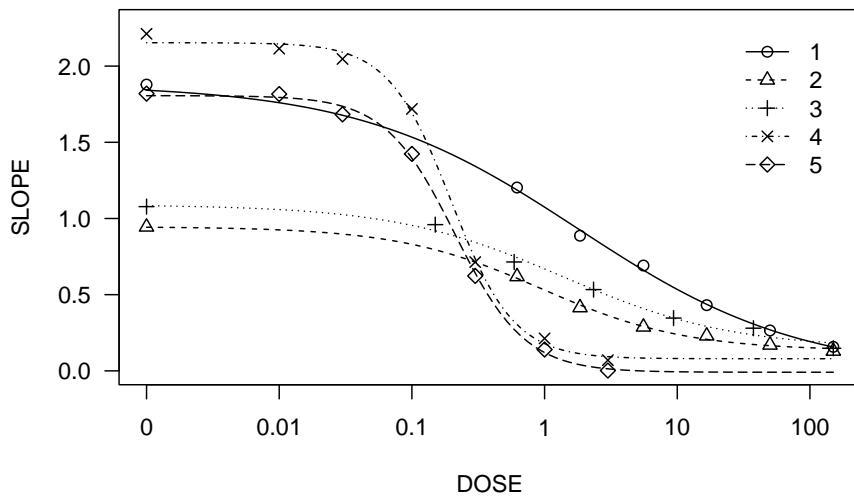
```
##
## Model fitted: Log-logistic (ED50 as parameter) (4 parms)
##
## Parameter estimates:
##
##           Estimate Std. Error t-value p-value
## b:1    0.5195192  0.0763600  6.8036 1.347e-09 ***
## b:2    0.8007959  0.2256794  3.5484 0.0006340 ***
## b:3    0.6819134  0.1285568  5.3044 8.838e-07 ***
## b:4    1.8448094  0.1663521 11.0898 < 2.2e-16 ***
## b:5    1.6507576  0.1758293  9.3884 8.857e-15 ***
## c:1   -0.0165952  0.1078254 -0.1539 0.8780472
## c:2    0.1325890  0.0471932  2.8095 0.0061561 **
## c:3    0.1464061  0.0604288  2.4228 0.0175253 *
## c:4    0.0795516  0.0394596  2.0160 0.0469555 *
## c:5   -0.0090656  0.0443536 -0.2044 0.8385337
## d:1    1.8795534  0.0423710 44.3594 < 2.2e-16 ***
```

```

## d:2  0.9460003  0.0422667 22.3817 < 2.2e-16 ***
## d:3  1.0903215  0.0405604 26.8814 < 2.2e-16 ***
## d:4  2.1535780  0.0281853 76.4079 < 2.2e-16 ***
## d:5  1.8062825  0.0292460 61.7616 < 2.2e-16 ***
## e:1  1.7949548  0.4782321  3.7533 0.0003183 *** 
## e:2  0.9455299  0.2494933  3.7898 0.0002809 *** 
## e:3  1.3730228  0.4526848  3.0331 0.0032100 ** 
## e:4  0.1973263  0.0101895 19.3657 < 2.2e-16 ***
## e:5  0.2107935  0.0138248 15.2475 < 2.2e-16 *** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error:
##
## 0.0735284 (85 degrees of freedom)

```

```
plot(spinach.m1)
```



Możliwości tego pakietu są dużo większe niż przedstawione powyżej. Autorzy przygotowali bardzo dobrą instrukcję użytkowania swojego pakietu razem ze szczegółowymi przykładami użycia wszystkich funkcji. Można ją znaleźć na stronie Biossay.

7.5 Analiza ANOVA

Test ANOVA zakłada, że nasze dane pochodzą z rozkładu normalnego, mają takie same wariancje i są niezależne.

Jeżeli nasze dane nie spełniają wymogu normalności można albo dane znormalizować (np. logarytm) albo zastosować test nieparametryczny np. test Kruskala_Wallisa albo test Friedmana.

Analizę wariancji można podzielić na jedno- i wieloczynnikową. Dane powinny zawierać wektor wartości (ilościowy), które można pogrupować wobec jednej lub więcej zmiennych jakościowych.

ANOVA w R można przeprowadzić na kilka sposobów, do popularnych należą funkcje `anova` i `aov`, które różnią się sposobem wywołania i prezentacji wyników.

Aby stwierdzić które średnie w naszym zbiorze danych różnią się można przeprowadzić tzw. testy post hoc np. test HSD Tukeya - `TukeyHSD`, `HSD.test` (pakiet `agricolae`).

```
# przykładowe dane z rozkładu normalnego, różniące się średniami

dane <- data.frame(x = c(rnorm(200,1), rnorm(200,1.3), rnorm(200, 1)),
                     y = rep(c("A","B","C"), each = 200))

# ANOVA przy pomocy funkcji aov, konieczne użycie summary dla wyświetlenia wyniku

model2 <- aov(x~y, data = dane)
summary(model2)

##           Df Sum Sq Mean Sq F value    Pr(>F)
## y          2   21.4  10.721   10.47 3.38e-05 ***
## Residuals 597  611.0   1.023
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# test Tukeya
# Otrzymujemy wartość p dla każdej pary czynników i różnicę z przedziałem ufności

TukeyHSD(model2)

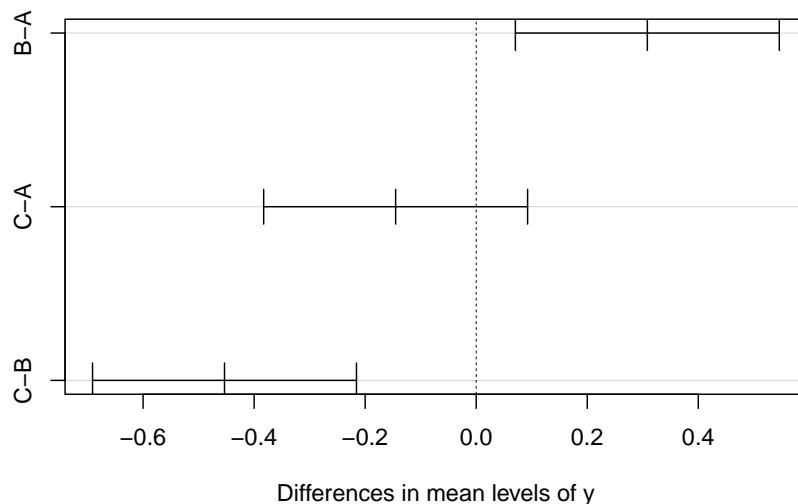
##   Tukey multiple comparisons of means
##   95% family-wise confidence level
##
##   Fit: aov(formula = x ~ y, data = dane)
##
```

```
## $y
##      diff      lwr      upr      p adj
## B-A  0.3082657  0.07056901  0.54596248 0.0068072
## C-A -0.1451069 -0.38280364  0.09258983 0.3238938
## C-B -0.4533726 -0.69106938 -0.21567591 0.0000264
```

wynik testu Tukeya można pokazać na wykresie

```
plot(TukeyHSD(model2))
```

95% family-wise confidence level



```
# Analiza dwuczynnikowa
# Dodajemy nową kolumnę do danych zawierającą losowy czynnik z - nie powinien wpływać na wynik
dane <- data.frame(dane, z = sample(c("d", "e", "f"), 600, replace = T))
```

```
model3 <- aov(lm(x~y + z, data = dane))
summary(model3)
```

```
##           Df Sum Sq Mean Sq F value    Pr(>F)
## y          2   21.4   10.721  10.472 3.39e-05 ***
## z          2     1.9     0.939    0.917      0.4
## Residuals 595  609.1    1.024
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```

TukeyHSD(model3)

## Tukey multiple comparisons of means
## 95% family-wise confidence level
##
## Fit: aov(formula = lm(x ~ y + z, data = dane))
##
## $y
##      diff      lwr      upr      p adj
## B-A  0.3082657  0.07053406  0.54599743 0.0068173
## C-A -0.1451069 -0.38283858  0.09262478 0.3239967
## C-B -0.4533726 -0.69110433 -0.21564097 0.0000265
##
## $z
##      diff      lwr      upr      p adj
## e-d -0.13293759 -0.3733960  0.1075208 0.3962997
## f-d -0.02981066 -0.2634909  0.2038696 0.9516867
## f-e  0.10312693 -0.1367823  0.3430362 0.5708251

# z analizę interakcji między y i z

model4 <- aov(lm(x~y + z + y:z, data = dane))
summary(model4)

##             Df Sum Sq Mean Sq F value    Pr(>F)
## y            2   21.4  10.721  10.416 3.58e-05 ***
## z            2     1.9   0.939   0.912   0.402
## y:z          4     0.8   0.201   0.196   0.941
## Residuals  591  608.3   1.029
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

TukeyHSD(model4)

## Tukey multiple comparisons of means
## 95% family-wise confidence level
##
## Fit: aov(formula = lm(x ~ y + z + y:z, data = dane))
##
## $y
##      diff      lwr      upr      p adj
## B-A  0.3082657  0.06988454  0.5466470 0.0070013
## C-A -0.1451069 -0.38348811  0.0932743 0.3259704

```

```

## C-B -0.4533726 -0.69175385 -0.2149914 0.0000281
##
## $z
##      diff      lwr      upr      p adj
## e-d -0.13293759 -0.3740530 0.1081778 0.3982824
## f-d -0.02981066 -0.2641293 0.2045080 0.9519415
## f-e  0.10312693 -0.1374378 0.3436917 0.5725553
##
## $`y:z`
##      diff      lwr      upr      p adj
## B:d-A:d  0.29169835 -0.23720856 0.820605261 0.7359430
## C:d-A:d -0.24652931 -0.79887024 0.305811617 0.9016398
## A:e-A:d -0.17273824 -0.73220104 0.386724559 0.9890591
## B:e-A:d  0.09317859 -0.45691785 0.643275023 0.9998503
## C:e-A:d -0.26587624 -0.82052453 0.288772062 0.8590070
## A:f-A:d -0.10214693 -0.63626789 0.431974033 0.9996283
## B:f-A:d  0.26382737 -0.29319391 0.820848659 0.8670527
## C:f-A:d -0.19632774 -0.72523465 0.332579172 0.9651189
## C:d-B:d -0.53822766 -1.07803023 0.001574904 0.0513479
## A:e-B:d -0.464443659 -1.01152427 0.082651091 0.1712732
## B:e-B:d -0.19851977 -0.73602549 0.338985955 0.9661548
## C:e-B:d -0.55757459 -1.09973789 -0.015411287 0.0383710
## A:f-B:d -0.39384528 -0.91498971 0.127299145 0.3123801
## B:f-B:d -0.02787098 -0.57246167 0.516719714 1.0000000
## C:f-B:d -0.48802609 -1.00382530 0.027773125 0.0802806
## A:e-C:d  0.07379107 -0.49598337 0.643565517 0.9999808
## B:e-C:d  0.33970790 -0.22087250 0.900288297 0.6232939
## C:e-C:d -0.01934692 -0.58439473 0.545700887 1.0000000
## A:f-C:d  0.14438238 -0.40052999 0.689294758 0.9961077
## B:f-C:d  0.51035669 -0.05702062 1.077733991 0.1173507
## C:f-C:d  0.05020158 -0.48960099 0.590004144 0.9999986
## B:e-A:e  0.26591682 -0.30168207 0.833515718 0.8740549
## C:e-A:e -0.09313800 -0.66514949 0.478873496 0.9998889
## A:f-A:e  0.07059131 -0.48153876 0.622721382 0.9999826
## B:f-A:e  0.43656561 -0.13774713 1.010878355 0.3044797
## C:f-A:e -0.02358950 -0.57067718 0.523498185 1.0000000
## C:e-B:e -0.35905482 -0.92190881 0.203799170 0.5537669
## A:f-B:e -0.19532552 -0.73796267 0.347311642 0.9710370
## B:f-B:e  0.17064879 -0.39454374 0.735841316 0.9905600
## C:f-B:e -0.28950632 -0.82701204 0.247999398 0.7605828
## A:f-C:e  0.16372931 -0.38352176 0.710980372 0.9911155
## B:f-C:e  0.52970361 -0.03992016 1.099327379 0.0919220
## C:f-C:e  0.06954850 -0.47261480 0.611711800 0.9999822
## B:f-A:f  0.36597430 -0.18368169 0.915630292 0.4929824
## C:f-A:f -0.09418081 -0.61532523 0.426963619 0.9997563
## C:f-B:f -0.46015511 -1.00474580 0.084435582 0.1761651

```

```

# test nieparametryczny Kruskala-Wallisa z pakietu podstawowego
wynik <- kruskal.test(dane$x,dane$y)
wynik

## 
##  Kruskal-Wallis rank sum test
##
## data:  dane$x and dane$y
## Kruskal-Wallis chi-squared = 18.758, df = 2, p-value = 8.448e-05

# albo z pakietu agricolae - pokazuje nie tylko czy coś się różni, ale również które grupy

wynik <- agricolae::kruskal(dane$x,dane$y)
wynik

## $statistics
##      Chisq Df   p.chisq   t.value     MSD
##  18.75789  2 8.448412e-05 1.963946 33.56365
##
## $parameters
##          test p.adjusted name.t ntr alpha
##  Kruskal-Wallis    none dane$y   3  0.05
##
## $means
##      dane.x   rank     std     r      Min      Max      Q25      Q50
## A 1.0811121 291.955 1.0485256 200 -2.0832352 3.923038 0.4085795 1.1321930
## B 1.3893779 341.575 0.9788305 200 -0.9459343 3.808629 0.7090154 1.3669592
## C 0.9360052 267.970 1.0064069 200 -1.5606246 3.654266 0.2692844 0.8887481
##      Q75
## A 1.611796
## B 1.988996
## C 1.655163
##
## $comparison
## NULL
##
## $groups
##      dane$x groups
## B 341.575      a
## A 291.955      b
## C 267.970      b
##
## attr(,"class")
## [1] "group"

```


Chapter 8

Miscellaneous

- Pakiet beep pozwala na generowanie dźwięków - funkcja `beep()` Dostępnych jest 10 różnych, można również podać własny plik .wav. Może się przydać jeżeli uruchamiamy w R jakiś dłuższy proces i chcemy wiedzieć kiedy się skończy.
- Pakiet fortunes zawiera zbiór cytatów na temat R, funkcja `fortune`
- Cytowanie R - dla całego R - wpisujemy `citation()`, dla poszczególnych pakietów `citation("nazwa_pakietu")`, podaje też cytowanie w formacie latex.

```
citation()

##
## To cite R in publications use:
##
##   R Core Team (2021). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2021},
##     url = {https://www.R-project.org/},
##   }
```

```
##  
## We have invested a lot of time and effort in creating R, please cite it  
## when using it for data analysis. See also 'citation("pkgname")' for  
## citing R packages.  
  
citation('ggplot2')  
  
##  
## To cite ggplot2 in publications, please use:  
##  
##   H. Wickham. ggplot2: Elegant Graphics for Data Analysis.  
##   Springer-Verlag New York, 2016.  
##  
## A BibTeX entry for LaTeX users is  
##  
## @Book{,  
##   author = {Hadley Wickham},  
##   title = {ggplot2: Elegant Graphics for Data Analysis},  
##   publisher = {Springer-Verlag New York},  
##   year = {2016},  
##   isbn = {978-3-319-24277-4},  
##   url = {https://ggplot2.tidyverse.org},  
## }
```