Wprowadzenie do R

Poniższa notatka powstała na podstawie materiałów Kamila Dyby.

Zacznijmy od rzeczy elementarnych czyli operacji na liczbach

```
# Operacje arytmetyczne
1+1
2*2
3^2
5%%3
```

```
log(1)
```

```
## [1] 0
```

```
sqrt(3)
```

```
## [1] 1.732051
```

```
sin(x = 2*pi)
```

```
## [1] -2.449294e-16
```

```
sin(2*pi) #nazwy argumentów funkcji można pomijać
```

```
## [1] -2.449294e-16
```

```
# Przypisanie, albo tradycyjne = albo przez strzałki
a=1
a<-1
1->a
a
```

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

```
# Operatory logiczne

1==2 #uwaga na podwójny ==, szczególnie w warunkach logicznych w if
```

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

```
1==1
```

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

```
1!=1
```

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

```
1<2
```

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

```
1<=2
```

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

```
TRUE || FALSE
```

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

```
TRUE && FALSE
```

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

```
TRUE | FALSE
```

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

```
TRUE & FALSE
```

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

```
! TRUE
```

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

```
#typy zmiennych
a <- 1.2
class(a)
```

```
## [1] "numeric"
```

```
a <- "zmienna"
class(a)
```

```
## [1] "character"
```

```
a <- TRUE
class(a)
```

```
## [1] "logical"
```

R jest językiem wektorowym, to znaczy takim, w którym operacje można wykonywać na wektorach, a nie pojedynczych liczbach.

```
# Wektory

v=c(1,3,2,5,6)
v
```

```
## [1] 1 3 2 5 6
```

```
v=1:6
v=seq(from = 1, to = 50, by=3.26)
w=rep(v,each=5)
w
```

```
##  [1]  1.00  1.00  1.00  1.00  1.00  4.26  4.26  4.26  4.26  4.26  7.52
## [12]  7.52  7.52  7.52  7.52 10.78 10.78 10.78 10.78 10.78 14.04 14.04
## [23] 14.04 14.04 14.04 17.30 17.30 17.30 17.30 17.30 20.56 20.56 20.56
## [34] 20.56 20.56 23.82 23.82 23.82 23.82 23.82 27.08 27.08 27.08 27.08
## [45] 27.08 30.34 30.34 30.34 30.34 30.34 33.60 33.60 33.60 33.60 33.60
## [56] 36.86 36.86 36.86 36.86 36.86 40.12 40.12 40.12 40.12 40.12 43.38
## [67] 43.38 43.38 43.38 43.38 46.64 46.64 46.64 46.64 46.64 49.90 49.90
## [78] 49.90 49.90 49.90
```

```
rep(3,15)
```

```
##  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

```
v=seq(10,100,by=10)
v[c(1,3,5)]
```

```
## [1] 10 30 50
```

```r
v[3:6]
```

```
## [1] 30 40 50 60
```

```r
v[-c(1,3,4)]
```

```
## [1]  20  50  60  70  80  90 100
```

```r
v[-(3:5)]
```

```
## [1]  10  20  60  70  80  90 100
```

```r
v=1:3; w=5:8; c(v,w)
```

```
## [1] 1 2 3 5 6 7 8
```

```r
c(1:5,6:10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
t=seq(0,2*pi,by=0.1)
2*t
```

```
##  [1]  0.0  0.2  0.4  0.6  0.8  1.0  1.2  1.4  1.6  1.8  2.0  2.2  2.4  2.6
## [15]  2.8  3.0  3.2  3.4  3.6  3.8  4.0  4.2  4.4  4.6  4.8  5.0  5.2  5.4
## [29]  5.6  5.8  6.0  6.2  6.4  6.6  6.8  7.0  7.2  7.4  7.6  7.8  8.0  8.2
## [43]  8.4  8.6  8.8  9.0  9.2  9.4  9.6  9.8 10.0 10.2 10.4 10.6 10.8 11.0
## [57] 11.2 11.4 11.6 11.8 12.0 12.2 12.4
```

```r
t-1
```

```
##  [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3
## [15]  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7
## [29]  1.8  1.9  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9  3.0  3.1
## [43]  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.0  4.1  4.2  4.3  4.4  4.5
## [57]  4.6  4.7  4.8  4.9  5.0  5.1  5.2
```

```r
sin(t)
```

```
##  [1]  0.00000000  0.09983342  0.19866933  0.29552021  0.38941834
##  [6]  0.47942554  0.56464247  0.64421769  0.71735609  0.78332691
## [11]  0.84147098  0.89120736  0.93203909  0.96355819  0.98544973
## [16]  0.99749499  0.99957360  0.99166481  0.97384763  0.94630009
## [21]  0.90929743  0.86320937  0.80849640  0.74570521  0.67546318
## [26]  0.59847214  0.51550137  0.42737988  0.33498815  0.23924933
## [31]  0.14112001  0.04158066 -0.05837414 -0.15774569 -0.25554110
## [36] -0.35078323 -0.44252044 -0.52983614 -0.61185789 -0.68776616
## [41] -0.75680250 -0.81827711 -0.87157577 -0.91616594 -0.95160207
## [46] -0.97753012 -0.99369100 -0.99992326 -0.99616461 -0.98245261
## [51] -0.95892427 -0.92581468 -0.88345466 -0.83226744 -0.77276449
## [56] -0.70554033 -0.63126664 -0.55068554 -0.46460218 -0.37387666
## [61] -0.27941550 -0.18216250 -0.08308940
```

```r
abs(sin(t))>1/2
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [56]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
t[abs(sin(t))>1/2]
```

```
##  [1] 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2
## [18] 2.3 2.4 2.5 2.6 3.7 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9
## [35] 5.0 5.1 5.2 5.3 5.4 5.5 5.6 5.7
```

Podobnie wygodnie korzysta się z macierzy

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
matrix(1:6,2,3,byrow=TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
#do macierzy można dołączać kolumny i wiersze
A=matrix(c(1:9),3,3)
A
```

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

```
cbind(A,2:4)
```

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

```
rbind(A,2:4)
```

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

```
rbind(2:4,A)
```

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

```
#fix(A)
A+A
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18
```

```
2*A
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18
```

```
sin(A)
```

```
##           [,1]       [,2]      [,3]
## [1,] 0.8414710 -0.7568025 0.6569866
## [2,] 0.9092974 -0.9589243 0.9893582
## [3,] 0.1411200 -0.2794155 0.4121185
```

```
A%*%A
```

```
##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

```
t(A)
```

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

```
diag(1:4)
```

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

```
t(1:3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
```

```
A[2,c(1,3)]
```

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

```
A[2,]
```

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

```
A[2,2]=4
A[3,]=c(-1,0,3)
A[3,]
```

```
## [1] -1  0  3
```

```
#macierze można rozkładać, liczyć wyznacznik i odwracać
B=matrix(c(3,5,2,6,2,6,2,6,2),3,3)
B
```

```
##      [,1] [,2] [,3]
## [1,]    3    6    2
```

```
## [2,]    5    2    6
## [3,]    2    6    2
```

```
eigen(B)
```

```
## $values
## [1] 11.4661265 -5.0218624  0.5557359
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.5718400  0.4281961 -0.7764031
## [2,] -0.6328890 -0.7436426  0.1093994
## [3,] -0.5219679  0.5134625  0.6206690
```

```
det(B)
```

```
## [1] -32
```

```
solve(B)
```

```
##         [,1]          [,2]  [,3]
## [1,]  1.0000 -9.436896e-17 -1.00
## [2,] -0.0625 -6.250000e-02  0.25
## [3,] -0.8125  1.875000e-01  0.75
```

```
solve(B,b = 1:3)
```

```
## [1] -2.0000  0.5625  1.8125
```

Jak sprawdzić składnię danej funkcji? Jak uzyskać pomoc?

```
help(sqrt)
?sqrt
help.search("trigonometric")
```

Oprócz wektorów i macierzy mamy listy.

```
l=list(imie="Kamil",
       nazwisko="Dyba",
       imiona.rodzicow=list(imie.matki="Barbara",
                            imie.ojca="Czesław"),
       nr.buta=41,
       kostiumy=c("król","Robin Hood","muszkieter", "kot w butach"))
```

```
l[1]
```

```
## $imie
## [1] "Kamil"
```

```
l[[1]]
```

```
## [1] "Kamil"
```

```
l[[3]]
```

```
## $imie.matki
## [1] "Barbara"
##
## $imie.ojca
## [1] "Czesław"
```

```
l$imie
```

```
## [1] "Kamil"
```

```
l$imiona.rodzicow
```

```
## $imie.matki
## [1] "Barbara"
##
## $imie.ojca
## [1] "Czesław"
```

```
l$imiona.rodzicow$imie.ojca #listy mogą być zagnieżdzone
```

```
## [1] "Czesław"
```

Ostatnim ważnym typem zmiennych jest ramka danych (data.frame). W kolumnach mamy zmienne, w wierszach obserwacje. Różnica z macierzą jest taka, że każda kolumna może być innego typu.

```
df1=data.frame(col1=c(1,2,3), col2=c("a", "b", "a"))
df1
```

```
##   col1 col2
## 1    1    a
## 2    2    b
## 3    3    a
```

```
#Ramki danych można także wczytać z pliku csv
a=read.csv("wprowadzenie_do_R_dane.csv")
a
```

```
##   X V1 V2 V3
## 1 1  1  3  1
## 2 2  3  1  3
## 3 3  1  3  1
```

```
class(a)
```

```
## [1] "data.frame"
```

Przestrzeń robocza

```
getwd()
setwd("~")
```

Instrukcje

Instrukcje warunkowe

```
if (1<2) "mniejsze"
```

```
## [1] "mniejsze"
```

```
a=b=0
if (1 %in% 1:4)
{
    a=1;
    b=2
}
c(a,b)
```

```
## [1] 1 2
```

```
if (1!=1) "różne" else
    "równe"
```

```
## [1] "równe"
```

```
ifelse(test = 1:8 < 5, yes = "mniej", no = "wiecej")
```

```
## [1] "mniej"  "mniej"  "mniej"  "mniej"  "wiecej" "wiecej" "wiecej" "wiecej"
```

```
if (1!=1) "różne"
else "równe" # linia nie może rozpoczynać się od else
```

```
if (1!=1){
  "różne"
} else "równe" # linia nie może rozpoczynać się od else
```

Pętle

```
t=numeric(100)
for (i in 1:100) {
  t[i]=i%%5
}
t
```

```
##   [1] 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0
##  [36] 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0
##  [71] 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0
```

```
# wektor w pętli nie musi składać się z liczb

i=1; k=1;
while (i<15) {
  i=i+1
  k=2*k
}
i
```

```
## [1] 15
```

```
k
```

```
## [1] 16384
```

W R można też definiować swoje własne funkcje

```
f<-function(x,y) {
  z=x*y
  return(z+x+y)
}

f(2,3)
```

```
## [1] 11
```

```
f<-function(x,y) { z=x*y; z+x+y }
f(2,3)
```

```
## [1] 11
```

```
tryCatch(f(2), error = function(e) e)
```

```
## <simpleError in f(2): argument "y" is missing, with no default>
```

```
{function(x,y){ z=x*y; z+x+y }}(2,3)
```

```
## [1] 11
```

```
#można przypisywać do funkcji argumenty domyślne
g <- function(x, y=3){
    x+y
}
g(2)
```

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

```
g(2,4)
```

```
## [1] 6
```

Jak było powiedziane wcześniej, R jest językiem wektorowym. W związku z tym, z punktu widzenia efektywności, NIE należy używać pętli! Zamiast tego w R mamy funkcje apply. Do każdgo elementu wektora (listy) aplikujemy daną funkcję

```
#w sapply wynikiem jest wektor
sapply(1:4, sin)
```

```
## [1]  0.8414710  0.9092974  0.1411200 -0.7568025
```

```
sapply(seq(0,10,by=.1),function(x){2^x-3*x})
```

```
##   [1]   1.00000000   0.77177346   0.54869835   0.33114441   0.11950791
##   [6]  -0.08578644  -0.28428343  -0.47549521  -0.65889887  -0.83393402
##  [11]  -1.00000000  -1.15645307  -1.30260329  -1.43771117  -1.56098418
##  [16]  -1.67157288  -1.76856687  -1.85099041  -1.91779775  -1.96786803
##  [21]  -2.00000000  -2.01290615  -2.00520658  -1.97542235  -1.92196836
##  [26]  -1.84314575  -1.73713373  -1.60198083  -1.43559549  -1.23573607
##  [31]  -1.00000000  -0.72581230  -0.41041316  -0.05084469   0.35606329
##  [36]   0.81370850   1.32573253   1.89603834   2.52880901   3.22852786
##  [41]   4.00000000   4.84837540   5.77917368   6.79831061   7.91212657
##  [46]   9.12741700  10.45146506  11.89207668  13.45761803  15.15705573
##  [51]  17.00000000  18.99675080  21.15834736  23.49662123  26.02425314
##  [56]  28.75483400  31.70293013  34.88415337  38.31523605  42.01411146
##  [61]  46.00000000  50.29350160  54.91669472  59.89324245  65.24850629
##  [66]  71.00966799  77.20586026  83.86830673  91.03047210  98.72822292
##  [71] 107.00000000 115.88700320 125.43338944 135.68648491 146.69701258
##  [76] 158.51933598 171.21172051 184.83661347 199.46094420 215.15644583
##  [81] 232.00000000 250.07400641 269.46677888 290.27296982 312.59402516
##  [86] 336.53867197 362.22344103 389.77322693 419.32188841 451.01289167
##  [91] 485.00000000 521.44801282 560.53355776 602.44593963 647.38805032
##  [96] 695.57734394 747.24688205 802.64645387 862.04377682 925.72578333
## [101] 994.00000000
```

```
#w lapply wynikiem jest lista
lapply(seq(0,10,by=1),function(x){2^x-3*x})
```

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

```
##
## [[2]]
## [1] -1
##
## [[3]]
## [1] -2
##
## [[4]]
## [1] -1
##
## [[5]]
## [1] 4
##
## [[6]]
## [1] 17
##
## [[7]]
## [1] 46
##
## [[8]]
## [1] 107
##
## [[9]]
## [1] 232
##
## [[10]]
## [1] 485
##
## [[11]]
## [1] 994
```

```
#dla macierzy możemy zdecydować czy idziemy po kolumnach, wierszach czy elementach
A=matrix(1:16,4,4,byrow=TRUE)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

```
apply(A, MARGIN = 1, FUN = sum)
```

```
## [1] 10 26 42 58
```

```
apply(A, MARGIN = 2, FUN = sum)
```

```
## [1] 28 32 36 40
```

```
apply(A,1:2,exp)
```

```
##              [,1]         [,2]         [,3]         [,4]
## [1,] 2.718282e+00 7.389056e+00 2.008554e+01 5.459815e+01
## [2,] 1.484132e+02 4.034288e+02 1.096633e+03 2.980958e+03
## [3,] 8.103084e+03 2.202647e+04 5.987414e+04 1.627548e+05
## [4,] 4.424134e+05 1.202604e+06 3.269017e+06 8.886111e+06
```

Podstawowe statystyki opisowe

```
mean(...)
var(...) # obciażona czy nieobciążona?
median(...)
min(...)
max(...)
quantile(..., 0.95)
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
attach(iris)
by(iris[1:4], INDICES = Species, function(M) apply(M,2,mean))
```

```
## Species: setosa
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        5.006        3.428        1.462        0.246
## ------------------------------------------------------------
## Species: versicolor
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        5.936        2.770        4.260        1.326
## ------------------------------------------------------------
## Species: virginica
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        6.588        2.974        5.552        2.026
```

```
by(iris[1:4], Species, function(M) {apply(M,2,function(v){c(sr=mean(v),war=var(v))})})
```

```
## Species: setosa
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## sr      5.006000   3.4280000   1.46200000  0.24600000
## war     0.124249   0.1436898   0.03015918  0.01110612
## ------------------------------------------------------------
## Species: versicolor
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## sr     5.9360000  2.77000000    4.2600000  1.32600000
## war    0.2664327  0.09846939    0.2208163  0.03910612
## ------------------------------------------------------------
## Species: virginica
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## sr     6.5880000   2.9740000    5.5520000  2.02600000
## war    0.4043429   0.1040041    0.3045878  0.07543265
```

Dużo ładniej wygląda to w pakiecie **dplyr**

```
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
```

```
iris %>% group_by(Species) %>%
  summarise_each(c("mean", "sd"))
```

```
## # A tibble: 3 x 9
##      Species Sepal.Length_mean Sepal.Width_mean Petal.Length_mean
##       <fctr>             <dbl>            <dbl>             <dbl>
## 1     setosa             5.006            3.428             1.462
## 2 versicolor             5.936            2.770             4.260
## 3  virginica             6.588            2.974             5.552
## # ... with 5 more variables: Petal.Width_mean <dbl>,
## #   Sepal.Length_sd <dbl>, Sepal.Width_sd <dbl>, Petal.Length_sd <dbl>,
## #   Petal.Width_sd <dbl>
```

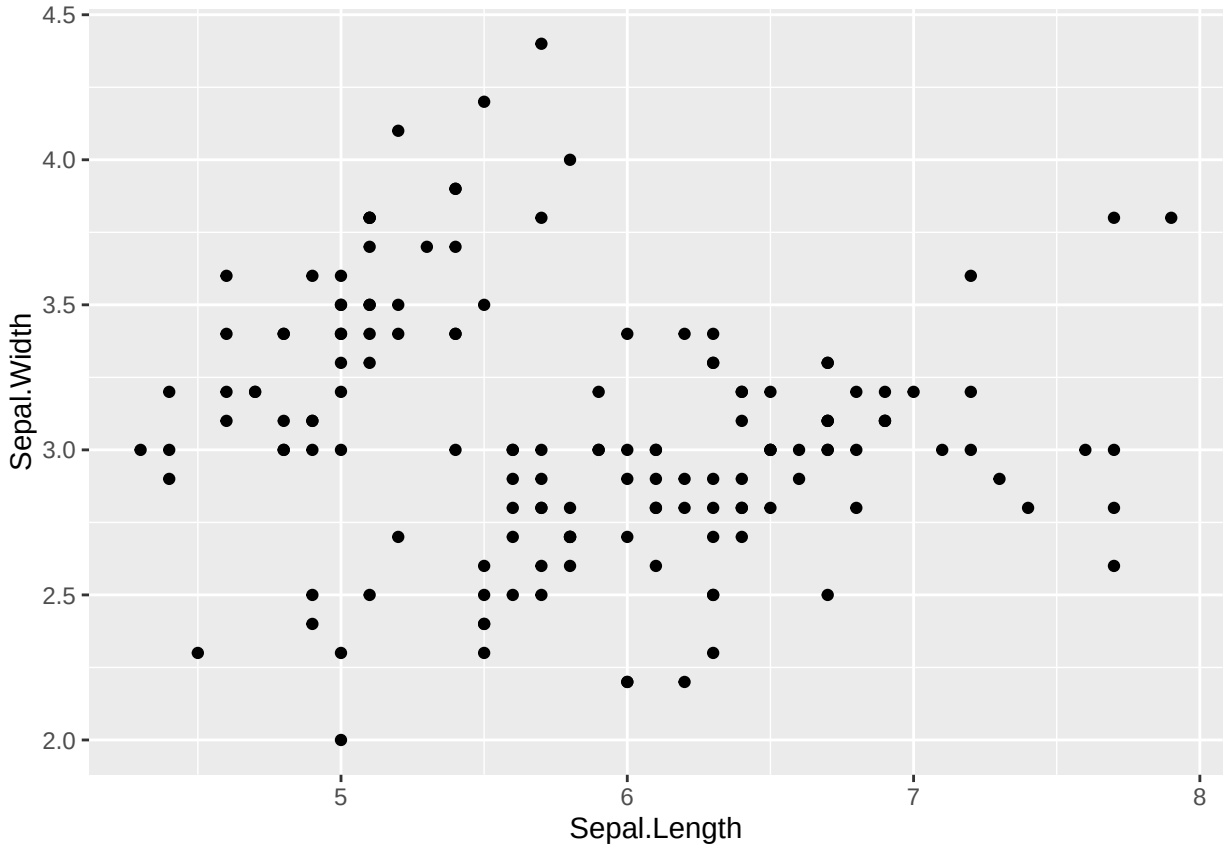
Rysowanie wykresów

W R mamy wiele sposobów na rysowanie wykresów. Podstawowy pakiet *__graphics__ jest łatwy w użyciu (można o nim przeczytać w notatkach Kamila Dyby), ale my pobawimy się czymś odrobinę bardziej zaawansowanym, ale za to produkującym ładniejsze wykresy :)

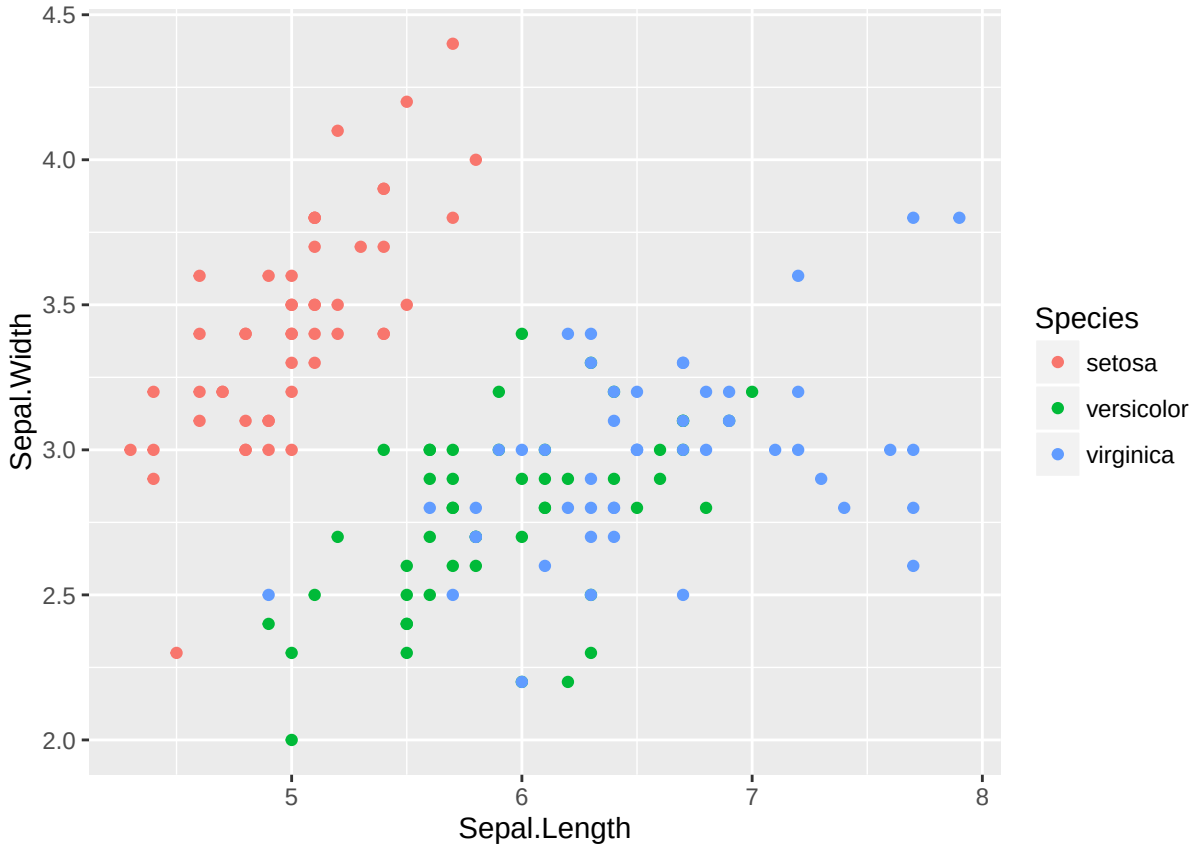
Pakiet ggplot. Podstawowe rzeczy:

- dane muszą być w postaci data.frame
- wybieramy typ wykresu jaki nas interesuje (patrz ściągawka)
- łączymy wymiary (oś x, oś y, kolor itd) z konkretnymi zmiennymi z data.frame
- grammar of graphics

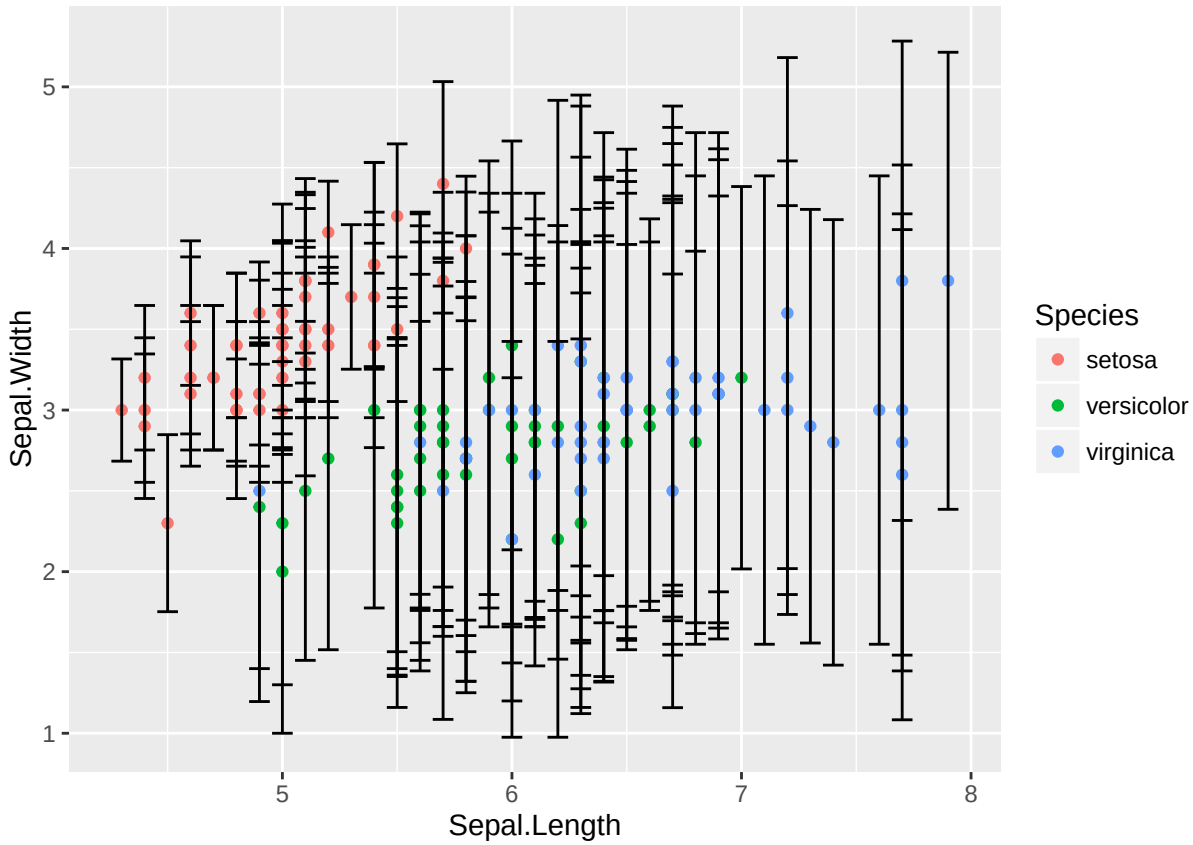
```
library(ggplot2)
#zwykly scatterplot
ggplot(iris) +
    geom_point(aes(x=Sepal.Length, y=Sepal.Width))
```



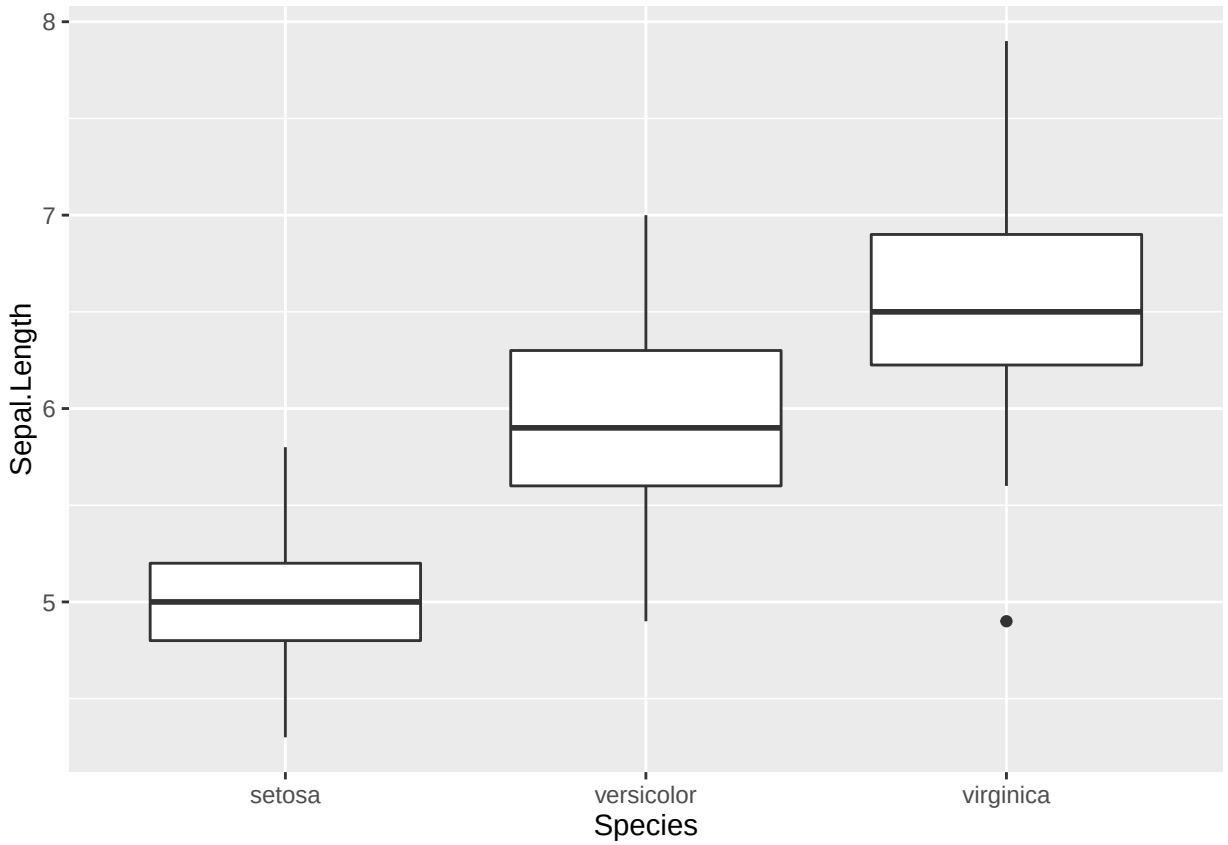

```
#scatterplot z dodanym kolorem w zaleznosci od gatunku
ggplot(iris) +
    geom_point(mapping = aes(x=Sepal.Length, y=Sepal.Width, col=Species))
```



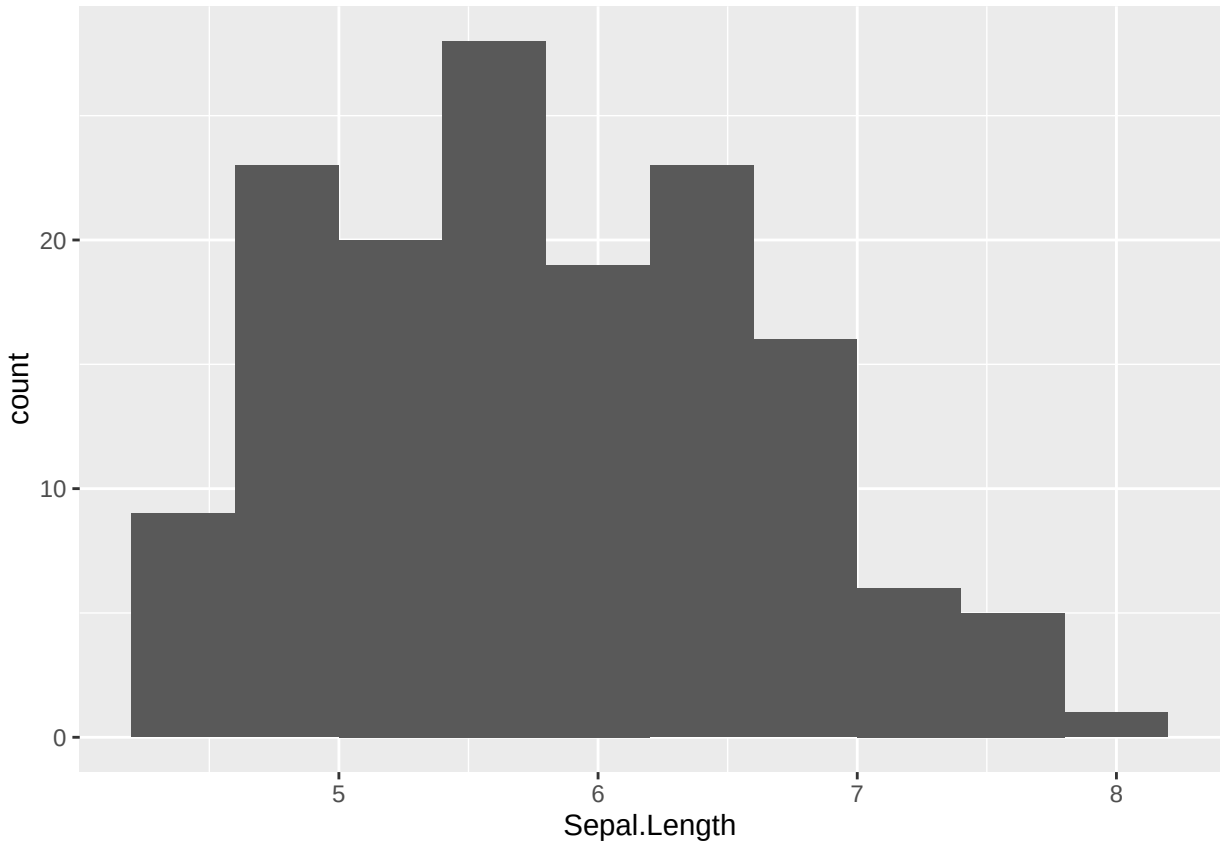

```
#dodanie błedów
ggplot(iris) +
    geom_point(aes(x=Sepal.Length, y=Sepal.Width, col=Species)) +
    geom_errorbar(aes(x=Sepal.Length, ymin=Sepal.Width-sqrt(Petal.Width), ymax=Sepal.Width+sqrt(Petal.W
```




```
#boxplot
ggplot(iris) +
    geom_boxplot(aes(x=Species, y=Sepal.Length))
```

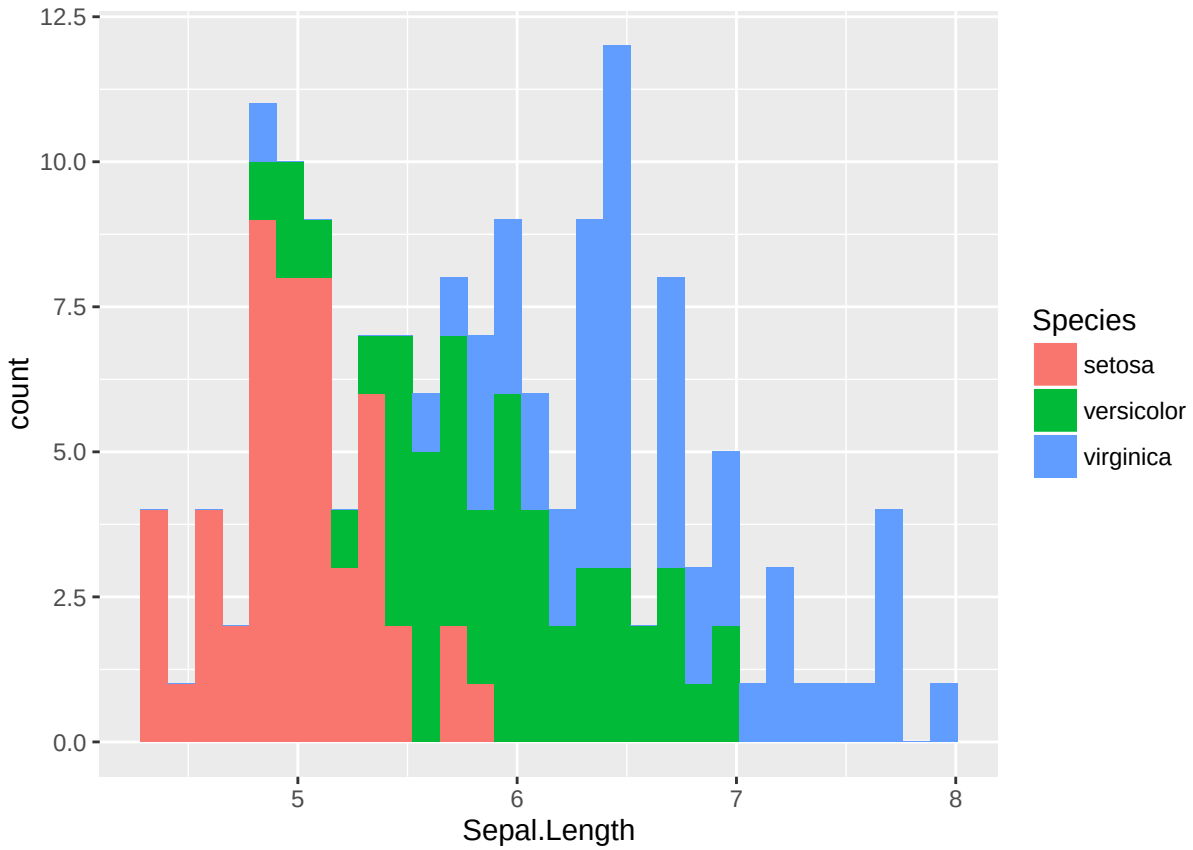



```
#histogram
ggplot(iris) +
    geom_histogram(aes(x=Sepal.Length), bins=10)
```



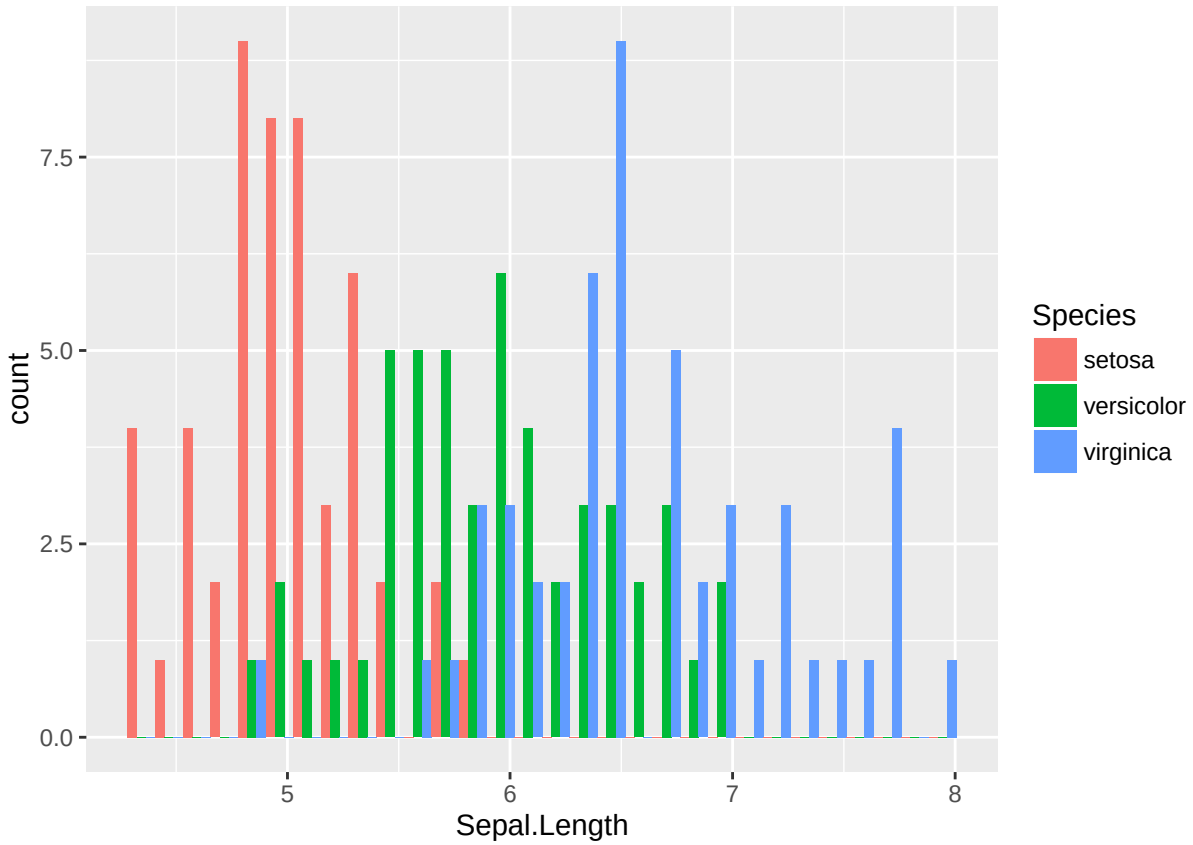

```
#histogram w podziale na gatunki
ggplot(iris) +
    geom_histogram(aes(x=Sepal.Length, fill=Species))
```

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



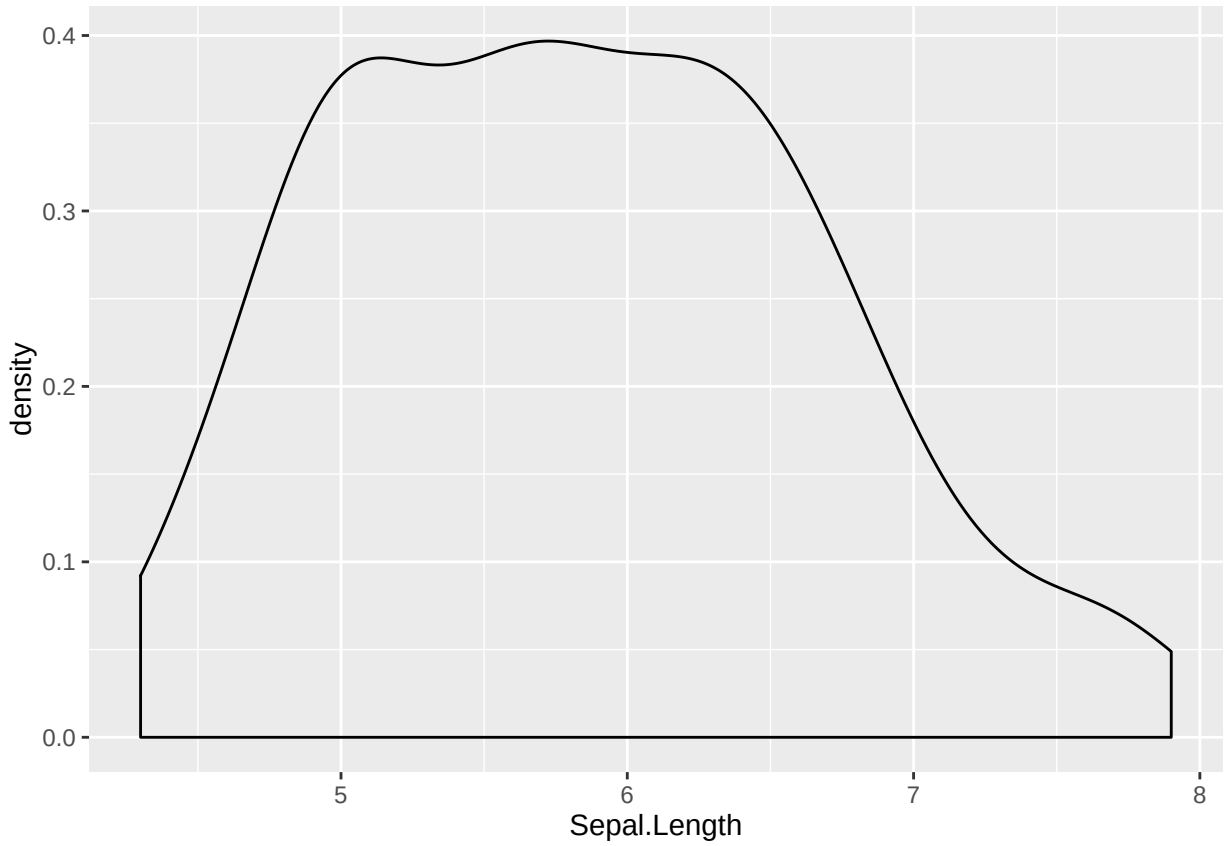

```
#histogram w podziale na gatunki rozłącznie
ggplot(iris) +
    geom_histogram(aes(x=Sepal.Length, fill=Species), position = "dodge")
```

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

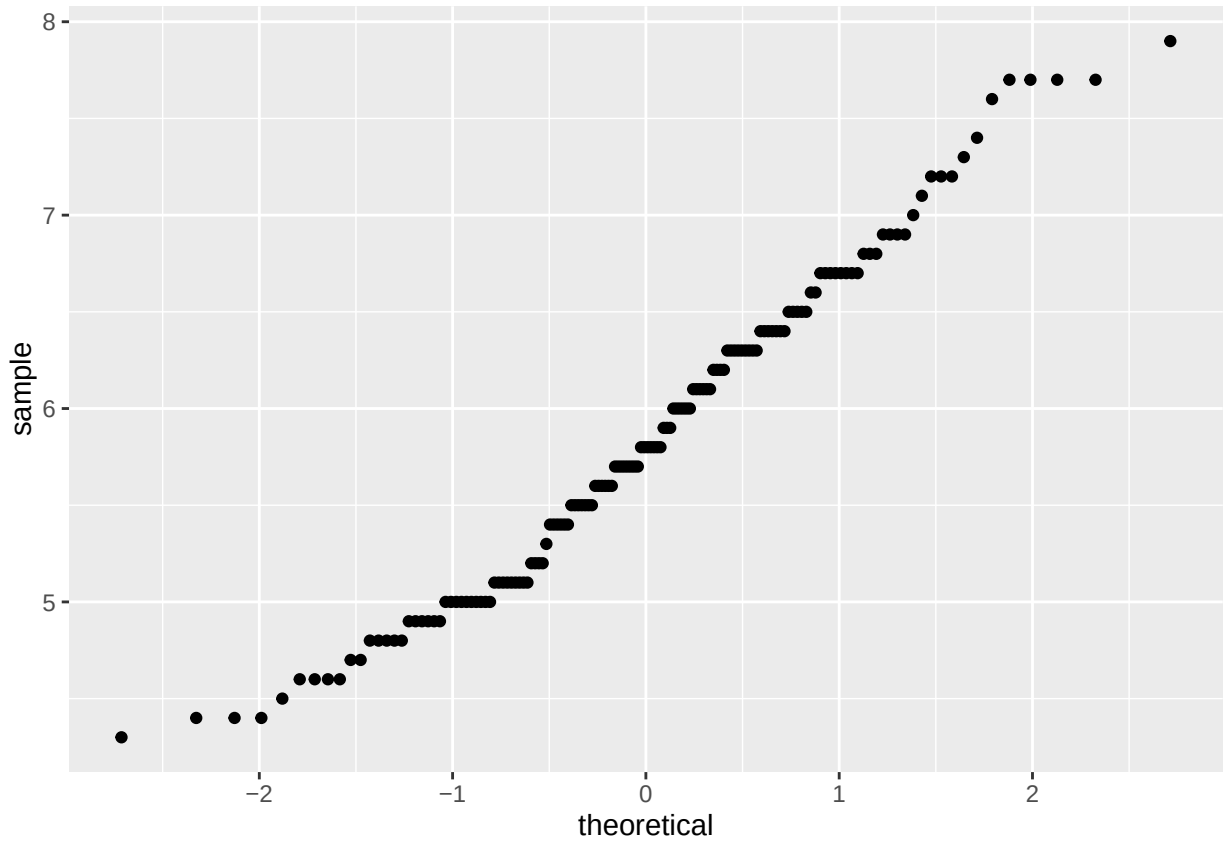



```
#zawsze można sprawdzić dokumentację
?geom_histogram
```

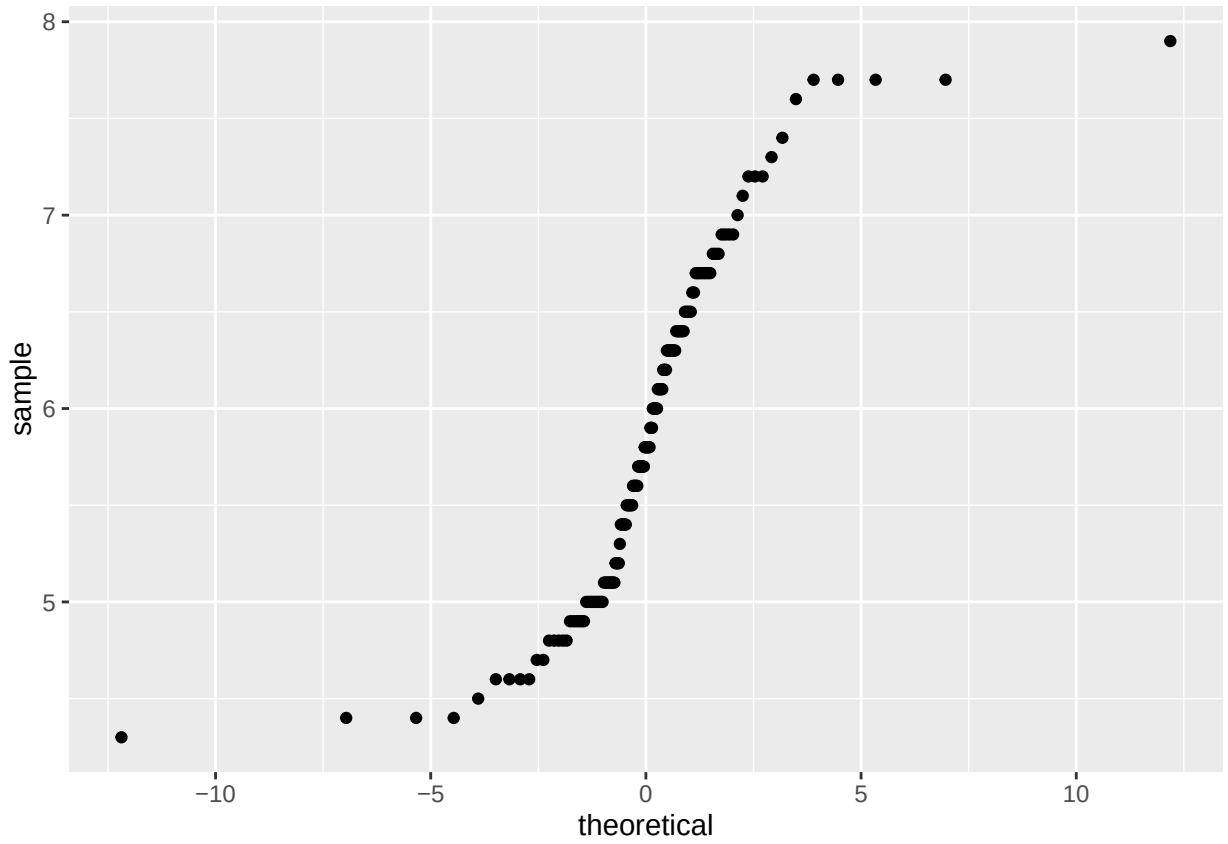
```
#jądrowy estymator gęstości
ggplot(iris) +
    geom_density(aes(x=Sepal.Length))
```




```
#wykres qqnorm
ggplot(iris) +
    stat_qq(aes(sample=Sepal.Length))
```




```
#wykres qq w odniesieniu do dowolnego rozkładu, tutaj Studenta z 2 stopniami swobody
ggplot(iris) +
    stat_qq(aes(sample=Sepal.Length), distribution = "qt", dparams = 2)
```



Rozkłady prawdopodobieństwa w R

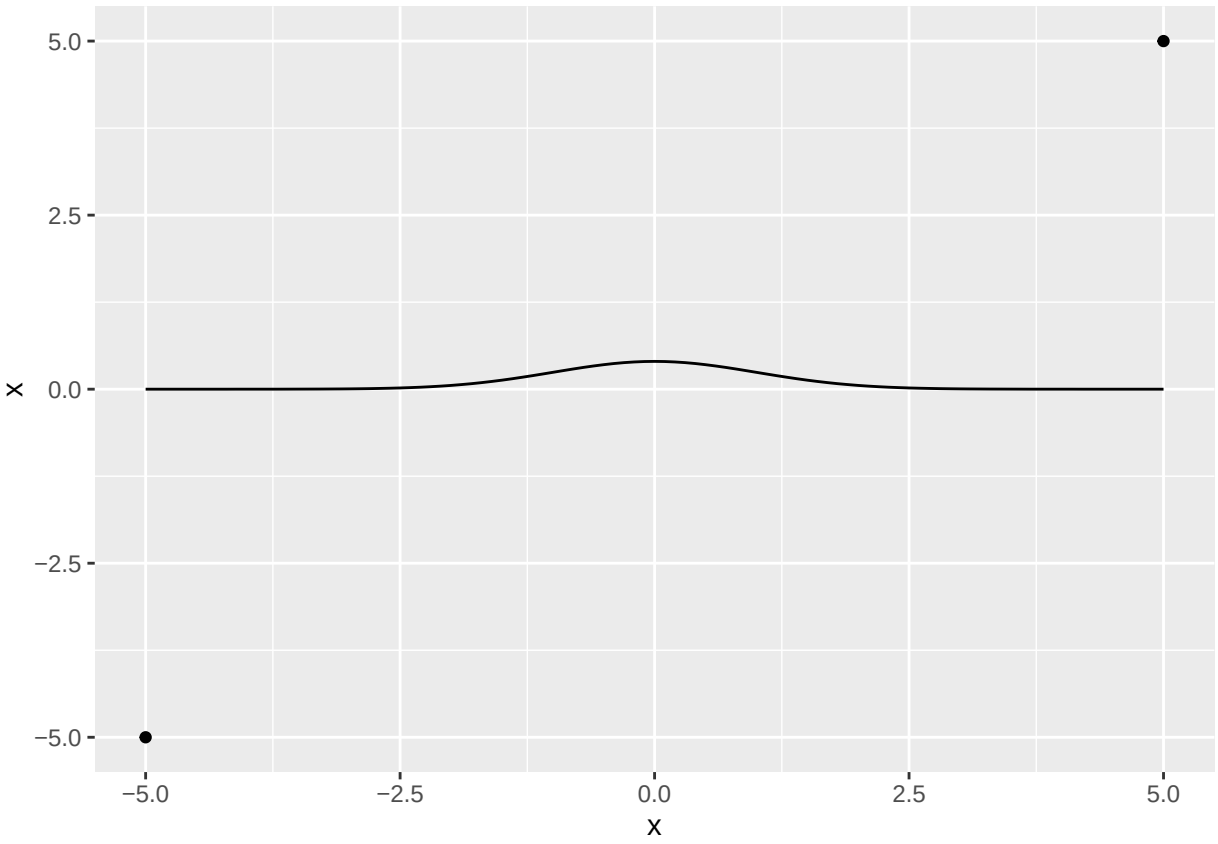
```
pdf + normal -> pnorm
density + exp -> dexp
quantile + t -> qt
random + chisq -> rchsiq
```

Więcej info w pomocy

```
?distributions
```

Gęstość rozkładu normalnego

```
ggplot(data.frame(x = c(-5, 5)), aes(x=x, y=x)) +
  geom_point(mapping = aes(x=x)) +
  stat_function(fun = dnorm)
```



Zapisywanie grafiki do pliku

```
p = ggplot(data.frame(x = c(-5, 5)), aes(x)) + stat_function(fun = dnorm)
ggsave(filename = "plik.png", plot = p, width = 10, height = 10)

#mozna tez zapisać ostatni wykres
ggsave(filename = "plik.png", width = 10, height = 10)
```

Przykład symulacji statystycznych

```
set.seed(7564) # dzięki ziarnu każdy może powtórzyć symulacje
A=matrix(c(rexp(1000),-rexp(1000)),20,100,byrow=TRUE)
dim(A)
```

```
## [1]  20 100
```

```
x<- c(T,F,T)
mean(x)
```

```
## [1] 0.6666667
```

```
x<-rnorm(20)
shapiro.test(x)$p.value<0.05
```

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

```
mean(apply(A,MARGIN = 2, function(v) {
  shapiro.test(v[1:10]+v[11:20])$p.value<=0.05
}))
```

```
## [1] 0.12
```

Jaki wniosek z mini symulacji?

Inne użyteczne pakiety

1. dplyr - przetwarzanie data.frame
2. readr, haven - wczytywanie plików z danymi w różnych formatach
3. RMarkdown - wsparcie dla markdown-a w R

i wiele, wiele innych...