# Вовед во R

## Елементи од јазикот, работа со распределби, дескриптивни статистики, визуелизација, и оценување на параметри

Дарио Ѓорѓевски[1]
gjorgjevski.dario@students.finki.ukim.mk

[1]Факултет за компјутерски науки и инженерство
Универзитет Св. Кирил и Методиј, Скопје

31 јануари 2017 год.

# Содржина



# Што е R?

R е:

- Колекција софтверски алатки кои служат за:
  - Читање на податоци и нивно манипулирање;
  - Вршење пресметки;
  - Статистичка анализа;
  - Приказ на резултати.
- Open-source верзија на програмскиот јазик S: јазик за манипулација на *објекти*;
- Платформа за развој и имплементација на (статистички) алгоритми.

R, заедно со пакети од имплементирани алгоритми и функционалности, може да бидат симнати од `cran.r-project.org`.

# Интерактива сесија

## Читање команди

Со започнување на R, започнува и интерактивна сесија. Корисниците внесуваат *команди* во конзолата на R. Кога R е спремен да прочита команда, печати prompt ‘>’. Командите се состојат од *изрази*. Изразите

- се одделени со точка-зарпика (`;`) или newline;
- може да бидат групирани во поголеми изрази, наречени *блокови*, со употреба на `{` и `}`.

Коментари може да бидат вклучени со тараба (`#`) и важат сè до крајот на редот.

Корисникот внесува команда и истата ја евалуира со притискање на ↵. Вредноста на командата се печати назад во конзолата; на пр.:

```
> 2 + 2
[1] 4
>
```

Prompt-от ‘>’ ни кажува дека R е спремен да вчита нова команда. Доколку внесена команда не е комплетна, prompt-от се променува во ‘+’, и останува така сè до комплетирање на командата.

# Како работи R?

R е програмски јазик кој работи со *објекти*. Дури и променливите и функциите се објекти и истите може да бидат користени како било кој друг објект.

- Објектите се креираат во меморија и зачувуваат во датотека `.RData`;
- Извршените команди се зачувуваат во датотека `.Rhistory`, и може да бидат изминати со ↑ и ↓;
- Лошите команди може да бидат прекинати со Esc или Ctrl + C;
- Команди може да бидат зачувани во датотека со екстензија `.R` и евалуирани со `source(file, ...)`;
- Сесијата се прекинува со повик на функцијата `q` или испраќање на `EOF` сигнал (Ctrl + D).

# R како калкулатор

R може да извршува стандардни аритметички пресметки:

```
> exp(-2)
[1] 0.1353353
> 2 * 3 * 4* 5
[1] 120
> pi  # R знае за бројот π
[1] 3.141593
> 1000 * (1 + 0.075)^2 - 1000
[1] 155.625
> sin(pi)
[1] 1.224647e-16  # 1,22 × 10^−16 ≈ 0
> 8 %/% 3 + 8 %% 5  # Целобројно делење и остаток
[1] 5
```

# Доделување вредности

Често сакаме одредени вредности да ги зачуваме во променливи. За да ја доделиме вредноста 10 на симболот `x`, пишуваме

```
> x <- 10
```

и притискаме ↵.
Еквивалентно, можеме да ја искористиме функцијата

```
> assign("x", 10)
```

Видлив резултат од извршената команда нема, но сега `x` има вредност 10 и може да се користи во понатамошни изрази:

```
> x
[1] 10
> x + x
[1] 20
> sqrt(x)
[1] 3.162278
```

# Case sensitivity и имиња на променливи

R е *case sensitive*: `x` и `X` не се однесуваат на истиот објект.
Идентификаторите

- може да содржат букви, цифри, долна црта (‘_’), и точка (‘.’);
- не смее да започнуваат со цифра, долна црта, или точка следена од цифра.

Некои имиња се резервирани од R и не може да бидат користени како идентификатори: `if else repeat while function`, ...

# Објекти

## Основни информации

Статистички анализи не е погодно да се вршат на едноставни броеви. R работи така што креира *објекти* користејќи различни функции за нивно креирање и манипулирање.

- Вектори од
  - броеви;
  - логички вредности;
  - карактери (strings);
  - комплексни броеви.
- Матрици и $n$-димензионални низи;
- Листи: произволни колекции од објекти од било кој тип;
- Data frames: листи со правоаголна структура;
- Функции;
- Специјален објект `NULL`.

# Објекти

## Workspace (работна околина)

За време на една сесија, објектите се зачувуваат по име. Функцијата

```
> ls()
```

ги печати имињата на сите моментално зачувани објекти во *workspace*-от (работната околина). Објекти може да бидат отстранети со функцијата

```
> rm(x)              # Избриши ја променливата x.
> rm(list=ls())      # Избриши ги сите објекти.
```

На крајот на секоја сесеија, корисникот е прашан дали состојбата на работната околина да биде зачувана во датотеката `.RData`.

# Содржина



Векторите се *наједноставен* тип на објект во R. Постојат 4 основни типови на вектори:

1. Нумерички вектори;
2. Вектори од карактери;
3. Вектори од логички вредности;
4. Вектори од комплексни броеви.

За дефинирање на вектор `x` со броевите 10,40, 5,60, 3,10, 6,40, и 21,70, користиме

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

За да ја испечатиме содржината на `x` користиме

```
> x
[1] 10.4 5.6 3.1 6.4 21.7
```

`[1]` на почетокот ни го кажува индексот на првиот елемент. Како што можеме да забележиме, R користи 1-based indexing.
Да пристапиме до елемент на дадена позиција користиме

```
> x[1]
[1] 10.4
> x[5]
[1] 21.7
```

Со функцијата `c`

```
> y <- c(x, 0, x)
```

можеме и да креираме две копии од `x` со 0 на средината.

- Пресметките вообичаено се вршат по-компонентно:

```
> 1 / x
[1] 0.09615385 0.17857143 0.32258065 0.15625000
[5] 0.04608295
```

- Пократките вектори се „рециклираат“ да се совпаднат со подолгите:

```
> x + c(1, 2)
[1] 11.4  7.6  4.1  8.4 22.7
Warning message:
In x + c(1, 2) :
  longer object length is not a multiple
  of shorter object length
```

# Вектори

## Функции кои работат со нумерички вектори

- Некои функции враќаат повеќе вредности—една за секој елемент на векторот (`sin`, `cos`, `round`, `exp`, `log`, ...):

```
> round(x)
[1] 10  6  3  6 22
```

- Некои функции враќаат една вредност (`sum`, `prod`, `length`, ...):

```
> sum(x) + prod(x)
[1] 25121.15
```

- Некои функции се „специјални“ (`sort`, `cumsum`, `range`, ...)

Комплексните броеви се означуваат како `5 + 6i`. Мора да внимаваме, R не го прави тоа автоматски за нас:

```
> sqrt(-17)
[1] NaN
Warning message:
In sqrt(-17) : NaNs produced
> sqrt(-17 + 0i)
[1] 0+4.123106i
> log(-5 + 6i)  # Комплексен логаритам.
[1] 2.055437+2.265535i
```

# Вектори

## Генерирање на секвенци

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1, 10)  # Еквивалентно со претходното.
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1, 10, by=2)
[1] 1 3 5 7 9
> seq(1, 10, length=5)
[1]  1.00  3.25  5.50  7.75 10.00
> rep(x, 2)
 [1] 10.4  5.6  3.1  6.4  21.7 10.4  5.6  3.1  6.4 21.7
```

Вектор од карактери со должина 1 се добива со ставање на збор во двојни наводници:

```
> # c1 ќе биде вектор од карактери со должина 1
> c1 <- "Canberra"
```

Можеме пак да ја искористиме функцијата `c`:

```
> c2 <- c(z, "Syndey", "Melbourne", "Newcastle"); c2
[1] "Canberra"  "Syndey"    "Melbourne" "Newcastle"

> c3 <- c("Perth", c2); c3
[1] "Perth"     "Canberra"  "Syndey"    "Melbourne"
[5] "Newcastle"
```

- Вектор од логички вредности ги содржи вредностите `TRUE`, `FALSE`, и `NA`
- Ваквите вектори се добиваат од *условни изрази*:

  ```
  > temp <- x > 13
  ```

  Овој израз го зема секој елемент од `x`, и го споредува со 13. Резултатот е логички вектор со истата должина како `x`, кој на позиција $i$ ја има вредноста од споредбата `x[i] > 13`.
- Оператори `>`, `>=`, `<`, `<=`, `==`, `!=`, `&`, `|`, `&&`, `||`.
- `&` и `|` враќаат вектори добиени по-компонентно, додека `&&` и `||` враќаат една вредност добиена од првите елементи.

# Вредности кои недостасуваат

Понекогаш точната вредност на одреден елемент во векторот не ја знаеме. Таквите вредности се претсавени со `NA`.
Можеме да провериме дали некоја вредност е `NA` користејќи ја функцијата `is.na`:

```
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE
> w <- c(1:3, rep(NA, 2), 5)
> w
[1]  1  2  3 NA NA  5
> is.na(w)
[1] FALSE FALSE FALSE  TRUE  TRUE FALSE
```

Веќе видовме како да индексираме вектори и да пристапиме до еден елемент. На сличен начин можеме да пристапиме и до повеќе елементи истовремено, така што наместо со една ќе индексираме со повеќе вредности:

- `> ex1 <- w[!is.na(w)]`

  ги зема елементите кои *не* се `NA`.

- `> ex2 <- w[1:3]`

  ги зема елементите на позиции 1, 2, и 3.

- `> ex3 <- w[-c(1, 4)]`

  ги зема *сиш̄е* елементи *освен* тие на позиции 1 и 4.

Модифицирањето е исто како индексирањето, само што истовремено доделуваме вредност:

```
# Првиот два елементи ќе имаат вредност 5.
> x[1:2] <- 5
> x
[1]  5.0  5.0  3.1  6.4 21.7
> # Им додаваме 5 на сите помали елементи.
> x[x < 5] <- x[x < 5] + 5
> x
[1]  5.0  5.0  8.1  6.4 21.7
```

Факторите се специјални типови на податоци кои се користат за категорични податоци, на пр.: пол, социјален статус, боја на очи, и сл.

- Внатрешната репрезентација е како нумерички вектор со вредности $1, 2, \ldots, k$, каде $k$ е бројот на *нивоа* во факторот;
- Може да бидат подредени или неподредени;
- Освен нумеричкиот вектор, факторот содржи и вектор од карактери кој го опишува секое ниво.

# Фактори

## Пример

Нека некоја анкета биде спроведена на 200 лица од машки и 300 лица од женски пол. Најдобар начин да ги зачуваме половите е со користење на фактор:

```
> gender <- c(rep("Male", 200), rep("Female", 300))
> gender <- factor(gender)
> levels(gender)
[1] "Female" "Male"
```

Внатре во R овој фактор е зачуван како

| | |
|---|---|
| 1 | Female |
| 2 | Male |

Матрицата

- е дводимензионална низа од броеви;
- има редици и колони;
- е често користена во статистиката.

Во R матриците се претставуваат како вектори *со димензии.*

```
> m <- rnorm(12)       # Генерирај 12 случајни броја.
> dim(m) <- c(3, 4)    # Подреди ги во 3 × 4 матрица.
> m
            [,1]       [,2]       [,3]      [,4]
[1,]  1.152181249  0.2933196 -0.7658679 -0.014726
[2,] -0.354144751  1.0523346  0.6052899 -1.080053
[3,]  0.003613231 -0.2068498 -2.5782982 -1.881613
```

# Матрици

## Функциите dim и matrix

Функцијата dim се користи за читање или промена на димензиите на објект. Кога вектор има димензија $m \times n$, R истиот го третира како матрица.

Друг начин за креирање на матрици е со matrix:

```
> n <- rnorm(10)  # Генерирај 10 случајни броеви.
> matrix(n, nrow=5, ncol=2, byrow=TRUE)
            [,1]       [,2]
[1,] -0.05156321 -0.73200194
[2,]  0.75949863 -0.38434349
[3,]  1.22055333  0.41306536
[4,]  0.39608629  0.41769997
[5,]  0.47927784 -0.08826759
```

Аргументот `byrow=TRUE` му кажува на R да ја пополни матрицата редица по редица наместо колона по колона.
Корисни функции за матрици се `nrow`, `ncol`, `rownames`, и `colnames`.

```
> nrow(m)
[1] 3
> rownames(m)
NULL
> rownames(m) <- c("A", "B", "C")
> m
          [,1]       [,2]       [,3]      [,4]
A  1.152181249  0.2933196 -0.7658679 -0.014726
B -0.354144751  1.0523346  0.6052899 -1.080053
C  0.003613231 -0.2068498 -2.5782982 -1.881613
```

# Матрици

## Транспонирање, додавање редици или колони

Функцијата `t` врши транспонирање:

```
> t(m)
                A           B            C
[1,]  1.1521812 -0.3541448  0.003613231
[2,]  0.2933196  1.0523346 -0.206849760
[3,] -0.7658679  0.6052899 -2.578298206
[4,] -0.0147260 -1.0800535 -1.881613177
```

За спојување на вектори и матрици ги користиме функциите `rbind` (додавање редици) и `cbind` (додавање колони). Доколку спојуваме матрици со матрици, димензиите на истите мора да се совпаѓаат. Доколку спојуваме вектори со било што, векторите се рециклираат или кратат сè додека не се добие соодветната димензија.

# Матрици

## Пример со rbind и cbind

```
> X1 <- matrix(1:12, nrow=3, ncol=4, byrow=TRUE)
> X2 <- matrix(20:27, nrow=2, ncol=4)
> rbind(X1, X2)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   20   22   24   26
[5,]   21   23   25   27
> cbind(t(X1), t(X2), c(1, 2))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   20   21    1
[2,]    2    6   10   22   23    2
[3,]    3    7   11   24   25    1
[4,]    4    8   12   26   27    2
```

# Матрици

## Индексирање

Нека имаме $5 \times 6$ матрица:

```
> X <- matrix(1:30, nrow=5)
> dimnames(X) <- list(letters[1:5], LETTERS[1:6])
> X
  A  B  C  D  E  F
a 1  6 11 16 21 26
b 2  7 12 17 22 27
c 3  8 13 18 23 28
d 4  9 14 19 24 29
e 5 10 15 20 25 30
```

Можеме да пристапиме до вредноста во редица 3, колона 2 со

```
> X[3, 2]
[1] 8
> X["c", "B"]
[1] 8
```

Цели димензии се индексираат така што истите се *испушташат*:

```
> X[, 2]          # Пристапи до втората колона.
 a  b  c  d  e
 6  7  8  9 10
> X[c(1, 3), ]    # Пристапи до првата и третата редица.
   A  B  C  D  E  F
a  1  6 11 16 21 26
c  3  8 13 18 23 28
```

Индексирањето може да се врши и со матрици (или низи). На пример, да пристапиме до елементите `X[1, 3]`, `X[2, 2]`, и `X[3, 1]`, можеме да дефинираме матрица која по редици ќе ги има соодветните индекси.

```
> idx <- matrix(c(1:3, 3:1), nrow=3)
> idx
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
> X[idx]
[1] 11  7  3
> X[idx] <- 0  # Замени ги елементите со 0.
```

# Матрици

## Операции

```
> A <- matrix(1:9, nrow=3)
> B <- matrix(10:18, nrow=3)
> A + B                 # По-компонентно собирање.
> A %*% B               # Производ на матрици
> A * B                 # По-компонентно множење.
> crossprod(A, B)       # A^T B.
> 1 / A                 # По-компонентно делење.
> solve(A)              # A^{-1}.
> eigen(A)              # Сопствени вредности и вектори.
```

# Низи

Низите се обопштување на матриците во $d$ димензии. Се креираат со функцијата `array`, или со доделување на повеќе од 2 димензии на вектор.

```
> array(runif(12), c(2, 3, 2))  # 2 × 3 × 2 димензии.
, , 1
          [,1]      [,2]      [,3]
[1,] 0.6671513 0.3530644 0.7217998
[2,] 0.4465717 0.2270993 0.2396116

, , 2
          [,1]      [,2]      [,3]
[1,] 0.6682330 0.5903150 0.7933734
[2,] 0.4730066 0.1273569 0.2701172
```

Индексирањето се врши на ист начин како кај матриците.

# Листи

## Креирање со `list`

Листите

- се подредена колекција од (именувани) компоненти—произволни објекти;
- користат `[ ]` за пристап до подлисти;
- користат `[[ ]]` или `$` за пристап до индивидуални компоненти;
- се креираат од функцијата `list`.

Едноставен пример за листа:

```
> L1 <- list(name="Fred", wife="Mary", num.children=3)
```

Забележуаме дека секоја компонента има име.

# Листи

## Индексирање

Компонентите на листите може да бидат индексирани по број или по име (доколку се именувани).

```
> L1$name
[1] "Fred"
> L1[["wife"]]
[1] "Mary"
> L1$num == L1[[3]]   # Парцијално совпаѓање.
[1] TRUE
```

До подлисти се пристапува со `[ ]`.

```
> L1[c(1, 2)]
$name
[1] "Fred"


$wife
[1] "Mary"
```

# Листи

## Имиња и конкатенација

Имињата на компонентите на една листа можеме да ги видиме или измениме со функцијата `names`:

```
> names(L1)[1] <- "first.name"
> names(L1)
[1] "first.name"   "wife"          "num.children"
```

Листи може да бидат конкатенирани исто како вектори:

```
> c(L1, list(last.name="Smith"))
$first.name
[1] "Fred"

...   # Останати компоненти.

$last.name
[1] "Smith"
```

Data frames

- се листи со правоаголна (матрична) структура—секоја компонента има иста должина;
- претставува податочно множество: колоните ги претставуваат атрибутите, а редиците елементите на примерокот.

Се креираат со `data.frame` на ист начин како `list`:

```
> genders <- factor(sample(c("M", "F"), 10, replace=TRUE))
> weights <- rnorm(10, 75, 5)
> df <- data.frame(gender=genders, weight=weights)
```

# Data frames

## Индексирање

Data frames може да бидат индексирани и како листи и како матрици.

```
> df$gender  # Пристапи до половите.
 [1] F F F M F M F M F M
Levels: F M
> df[[1]]      # Еквивалентно со погорното.
> df[2]        # Подлиста која содржи само weights.
> df[1, 2]     # Елемент на позиција (1,2).
[1] 69.59721
> df[df$gender == "M", ]
   genders  weights
4        M 72.46704
6        M 76.68369
8        M 78.13764
10       M 68.60033
```

# Коерција на типови

Кога функција е повикана со аргументи, R се обидува да изврши *коерција*: да ги промени типовите така што функцијата би работела.

```
> x <- 1:5          # x е нумерички вектор.
> x[2] <- "foo"     # Векторот може да чува само еден тип,
> x                 # па R ќе ги направи сите карактери.
[1] "1"   "foo" "3"   "4"   "5"
```

Експлицитна коерција може да се врши со функциите од типот `as.*` (`as.numeric`, `as.data.frame`, ...). Подредувањето е грубо `logical` $\prec$ `numeric` $\prec$ `complex` $\prec$ `character` $\prec$ `list` (да се потсетиме дека матриците се „специјални“ вектори, а data frames „специјални“ листи).

# Контролни структури

## if израз

`if` изразот служи за условно извршување на изрази и има облик

```
if ( condition )
  true_expr
else
  false_expr
```

каде `{true,false}_expr` се изрази кои се извршуваат кога условот е `TRUE` односно `FALSE`. Бидејќи `condition` треба да биде вектор од логички вредности со должина 1, препорачано е да се користат операторите `&&` и `||` наместо `&` и `|`.

# Контролни структури

## if израз

Вредноста на if изразот е вредноста на извршениот израз во однос на условот.

```
> if (runif(1) >= .5) {
+   "Yes"
+ } else {
+   "No"
+ }
[1] "No"
> x <- -5
> y <- if (x >= 0) x else -x
> y
[1] 5
```

# Контролни структури

## `ifelse` функција—векторизиран `if` израз

Доколку сакаме условно да примениме операции на сите елементи од вектор, најпогодно е користењето на `ifelse` функцијата.

```
> v <- c(1, 1, 2, 3, 5:10)
> ifelse(v == 1 | v >= 8, v + 10, -v)
[1] 11 11 -2 -3 -5 -6 -7 18 19 20
> ifelse(1:length(v) %% 2, v, -v)
[1]   1  -1   2  -3   5  -6   7  -8   9 -10
> ifelse(round(sqrt(v)) == sqrt(v), sqrt(v), NA)
[1]  1  1 NA NA NA NA NA NA  3 NA
```

Синтаксата е

```
for (var in seqn)
  expr
```

каде **seqn** е вектор или листа. Вредноста на `for` изразот е `NULL`.

```
> for (i in 1:5) {
+   print(exp(i))
+ }
[1] 2.718282
[1] 7.389056
[1] 20.08554
[1] 54.59815
[1] 148.4132
```

# Контролни структури

## while изрази за циклуси

Синтаксата е

```
while (condition)
  expr
```

За да провериме колку пати треба да влечеме еден од 100 елементи дур не извлечеме 20, можеме да извршиме

```
> niter <- 1
> num <- sample(1:100, 1)
> while (num != 20) {
+   num <- sample(1:100, 1)
+   niter <- niter + 1
+ }
> niter
[1] 6
```

# Контролни структури

## next, break, и repeat

- `next` ја прескокнува моменталната итерација на еден циклус и продолжува со наредната итерација;
- `break` излегува од циклус;
- `repeat expr` претставува бесконечен циклус: ја извршува `expr` постојано.

Во случај на вгнездени циклуси, `next` и `break` работат само со циклусот на моменталното ниво.

Синтаксата за дефинирање функција е

```
function(arglist)
  funcbody
```

каде `function` е клучен збор.

- `arglist` е листа од формални аргументи одделени со запирка. Формален аргумент може да биде симбол, израз од облик ‘`symbol=default_expression`’, или специјалниот формален аргумент ‘`...`’.
- `funcbody` може да биде било кој израз. Најчесто, тоа е блок како специјален израз.
- Функциите обично сакаме да ги именуваме, па користиме

```
funcname <- function(arglist)
  funcbody
```

- Функција се повикува со операторот `(argvalues)`, каде `argvalues` се вредностите на аргументите одделени со запирка.
- Формалните аргументи на функцијата се заменуваат со дадените вредности, и `funcbody` се евалуира со овие вредности.
- Вредноста на функцијата е вредноста на последниот евалуиран израз во неа. За предвремено да излеземе од функција и да вратиме вредност, користиме `return(value)`.
- Совпаѓањето на аргументи се врши првин по име, па потоа по позиција.

# Функции

## Пример

```
> pow <- function(x, y=2) {
+   if (y < 0) {
+     (1 / x)^(-y)
+   } else {
+     x^y
+   }
+ }
> pow(5)
[1] 25
> pow(5, x=3)
[1] 243
> pow(y=-2, 10)
[1] 0.01
```

# Функции од фамилијата на `apply`

Овие функции служат за примена на функции на сите елементи од вектори или листи.

- `lapply`: зема вектор или листа и враќа листа;
- `sapply`: исто како `lapply`, само се обидува да го упрости (simplify) резултатот во вектор (или матрица);
- `apply`: се користи на низи (и матрици како нивни специјални случаи);
- `tapply`: се користи за креирање таблици групирани по фактори и агрегирани по дадена функција.

# Функции од фамилијата на apply

## Пример со lapply и lapply

```
> head(iris, 2)
  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

> numeric.cols <- sapply(iris, is.numeric); numeric.cols
Sepal.Length  Sepal.Width Petal.Length  Petal.Width   Species
        TRUE         TRUE         TRUE         TRUE     FALSE

> lapply(iris[numeric.cols], mean)
$Sepal.Length   $Petal.Length
[1] 5.843333    [1] 3.758

$Sepal.Width    $Petal.Width
[1] 3.057333    [1] 1.199333
```

# Функции од фамилијата на apply

## Пример со apply

R ги содржи функции rowMeans и colMeans за сметање на просек по редици односно колони на матрици. Истите е тривијално да се дефинираат преку apply. Освен низа и функција која ќе биде применета, apply прима уште еден аргумент: димензијата по која ќе биде применета функцијата (овој аргумент е познат како *марџина*).

```
> our.rowMeans <- function(X) {
+   apply(X, 1, mean)
+ }
> X <- matrix(1:6, 3, 2)
> X
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> our.rowMeans(X)
[1] 2.5 3.5 4.5
```

# Функции од фамилијата на apply

## Пример со tapply

tapply креира таблица така што примероците ги групира по фактори, а групите агрегира со дадена функција.

```
> library(MASS)  # Вчитување на библиотека.
> # Сакаме автомобилите да ги групираме по производител
> # и за секоја група да пресметаме просечна цена.
> tapply(Cars93$Price, Cars93$Manufacturer, mean)
      Acura        Audi         BMW       Buick
   24.90000    33.40000    30.00000    21.62500
> # Доколку дополнително сакаме да групираме и по тип:
> tapply(Cars93$Price, list(Cars93$Manufacturer, Cars93$Type),
+        mean)
              Compact Large Midsize Small Sporty   Van
Acura              NA    NA    33.9 15.90     NA    NA
Audi             29.1    NA    37.7    NA     NA    NA
```

# Содржина



# Формат на податоци

Во R вградена е поддршка за вчитување податоци зачувани во текстуални формати. Најчесто користени формати се

- CSV: comma-separated values (вредностите во една редица се одделени со запирка, редиците со newline);
- TSV: tab-separated values (вредностите во една редица се одделение со tab, редиците со newline).

Вчитувањето на други формати, на пр. формати од Microsoft Excel (`.xls{,x}`) или Weka (`.arff`) се врши преку библиотеки како `gdata` и `foreign`.

# Вчитување на текстуални датотеки

Нека е дадена следната датотека:

```
% cat Data/Trees.csv | head -n 7
"Girth","Height","Volume"
8.3,70,10.3
8.6,65,10.3
8.8,63,10.2
10.5,72,16.4
10.7,81,18.8
10.8,83,19.7
11,66,15.6
```

При вчитување на вакви датотеки, треба да внимаваме на тоа како се одделени вредностите, дали првиот ред или колона се користат како имиња, дали е користена децимална точка или запирка, и сл.

# Вчитување на текстуални датотеки

## `read.table` и варијанти

- `read.table` е основната функција за вчитување текстуални датотеки. Нејзиното однесување е целосно кориснички дефинирано. Видете `?read.table` за целосни информации.
- `read.csv` и `read.delim` се специфични верзии на `read.table` за CSV односно TSV датотеки.

Датотеката `trees.csv` би ја вчитале како

```
> trees <- read.csv("Data/Trees.csv")
> head(trees, 3)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
```

# Содржина



Нека $X$ е случајна променлива со дадена распределба. Распределбата на $X$ дефинира две функции: *густина* $p(x)$ и *функција на распределба* $F(x) = \Pr(X \leq x)$, за кои важи

$$F(x) = \int_{-\infty}^{x} p(x)\,\mathrm{d}\,x$$

$$p(x) = \frac{\mathrm{d}\,F}{\mathrm{d}\,x}.$$

Дополнително можеме да дефинираме *функција на квантили* $Q\colon [0,1] \to \mathbb{R}$ како

$$Q(p) = F^{-1}(p) = \inf\{x \in \mathbb{R} : p \leq F(x)\}.$$

# Функции за работа со веројатносни распределби во R

R знае за сите стандардни распределби, и ги содржи сите претходно наведени функции како и функција за генерирање случајни броеви. Имињата на функциите следат стандарден формат (`name` треба да се замени со името на распределбата):

- `rname(n, ...)`: генерирање на $n$ случајни броеви од распределба;
- `pname(x, ...)`: вредноста на $F(x)$ за распределбата;
- `dname(x, ...)`: вредноста на $p(x)$ за распределбата;
- `qname(p, ...)`: вредноста на $Q(p)$ за распределбата.

# Имиња на распределби во R

| Име на распределба | Име во R | Параметри |
|---|---|---|
| нормална | `norm` | `mean, sd` |
| $t$-распределба | `t` | `df` |
| $F$-распределба | `f` | `df1, df2` |
| $\chi^2$ | `chisq` | `df` |
| Пуасонова | `pois` | `lambda` |
| експоненцијална | `exp` | `rate` |
| гама | `gamma` | `shape, rate` |
| биномна | `binom` | `size, prob` |
| геометриска | `geom` | `prob` |
| негативна биномна | `nbinom` | `size, prob` |
| хипергеометриска | `hyper` | `m, n, k` |
| рамномерна | `unif` | `min, max` |
| ⋮ | ⋮ | ⋮ |

# Функции за работа со веројатносни распределби во R

## Пример

```
> x <- rpois(10^5, 3)   # 10^5 броеви од X ~ Pois(λ = 3)
> dpois(4, 3)           # Pr(X = 4)
[1] 0.1680314
> length(x[x == 4]) / length(x)   # Емпириска распределба
[1] 0.16969
> x <- rnorm(10^5, mean=3, sd=2)   # X ~ N(μ = 3, σ^2 = 4)
> x <- rnorm(10^5)                 # X ~ N(μ = 0, σ^2 = 1)
> length(x[x >= -3 & x <= 3]) / length(x)   # Правило на 3σ
[1] 0.99738
> pbinom(3, size=10, 0.4)   # Pr(Y ≤ 3) за Y ~ B(n = 10, p = 0.4)
[1] 0.3822806
> pbinom(3, size=10, 0.4, lower.tail=FALSE)   # Pr(Y > 3)
[1] 0.6177194
```

# Содржина



# Модел на примерок

**Дефиниција (Случаен примерок)**

Нека на популација $\Omega$ биде дефинирано обележје $X$, и $X_1$ биде вредноста на $X$ набљудувана во првиот, $X_2$ во вториот, ..., $X_n$ во $n$-тиот опит. Тогаш $X_1, X_2, \ldots, X_n$ се i.i.d. случајни променливи со распределба иста како $X$. Случајниот вектор $(X_1, X_2, \ldots, X_n)$ го нарекуваме *случаен примерок*.

Оваа дефиниција ни овозможува понатаму формален третман на статистичките методи.

# Дескриптивни статистики

- Просек на примерок: $\bar{X} := \frac{1}{n}\sum_{k=1}^{n} X_k$
- Дисперзија на примерок: $S^2 := \frac{1}{n-1}\sum_{k=1}^{n}(X_k - \bar{X})^2$
- Стандардна девијација на примерок: $S := \sqrt{S^2}$
- Ковариjанса и корелација на два примерока:

$$Q_{X,Y} := \frac{1}{n-1}\sum_{i=1}^{k}(X_k - \bar{X})(Y_k - \bar{Y})$$

$$r_{X,Y} := \frac{Q_{X,Y}}{S_X S_Y}$$

- Квантили, опсег, и интер-квантилен опсег

# Десктиптивни статистики во R

R содржи функции за наоѓање на сите десктиптивни статистики.

```
> x <- rnorm(10^4, 5, 3)  # 10^4 реализации од N(μ = 5, σ^2 = 9)
> mean(x)
[1] 5.009368
> sd(x)
[1] 3.01369
> var(x)
[1] 9.082328
> quantile(x)
      0%       25%       50%       75%      100%
-5.585404  2.983599  4.974200  7.047683 15.984251
> range(x)
[1] -5.585404 15.984251
> IQR(x)  # Интер-квантилен опсег.
[1] 4.064084
```

# Десктиптивни статистики во R

```
> y <- 2*x + 3
> corr(x, y)
> cor(x, y)
[1] 1  # Совршена линеарна врска (позитивна).
> z <- rpois(10^4, lambda=5)
> cor(x, z)
[1] -0.003875863  # Слаба корелација.
> x <- runif(200, min=-1, max=1)
> y <- abs(x)
> cor(x, y)
[1] 0.04368372  # Слаба корелација ⇏ независност.
```

# Пресметување на мода во R

R нема вградена функција за наоѓање на мода на податоци. Но, ние можеме наивно решение да имплементираме сами:

```
> our.mode <- function(x) {
+   ux <- unique(x)
+   tab <- tabulate(match(x, ux))
+   ux[tab == max(tab)]
+ }

> x <- c(5, 6, 7, 7, 1, 1, 1, 3, 4, 5, 7)
> our.mode(x)
[1] 7 1  # Би-модални податоци.
```

Видете како работат функциите `unique`, `tabulate`, и `match`. Обидете се да ги разберете во контекст на `our.mode`.

# Сумарни статистики во R

Доколку реализацијата на случајниот примерок ни е зачувана во соодветен објект (најчесто data frame), можеме со една команда да испечатиме сумарни статистики.

```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

# Сумарни статистики во R

```
> library(pastecs); stat.desc(iris)
              Sepal.Length  Sepal.Width Petal.Length ...
nbr.val       150.00000000 150.00000000  150.0000000 ...
nbr.null        0.00000000   0.00000000    0.0000000 ...
nbr.na          0.00000000   0.00000000    0.0000000 ...
min             4.30000000   2.00000000    1.0000000 ...
max             7.90000000   4.40000000    6.9000000 ...
range           3.60000000   2.40000000    5.9000000 ...
sum           876.50000000 458.60000000  563.7000000 ...
median          5.80000000   3.00000000    4.3500000 ...
mean            5.84333333   3.05733333    3.7580000 ...
SE.mean         0.06761132   0.03558833    0.1441360 ...
CI.mean.0.95    0.13360085   0.07032302    0.2848146 ...
var             0.68569351   0.18997942    3.1162779 ...
std.dev         0.82806613   0.43586628    1.7652982 ...
coef.var        0.14171126   0.14256420    0.4697441 ...
```

# Содржина



# `plot` како основна функција за цртање

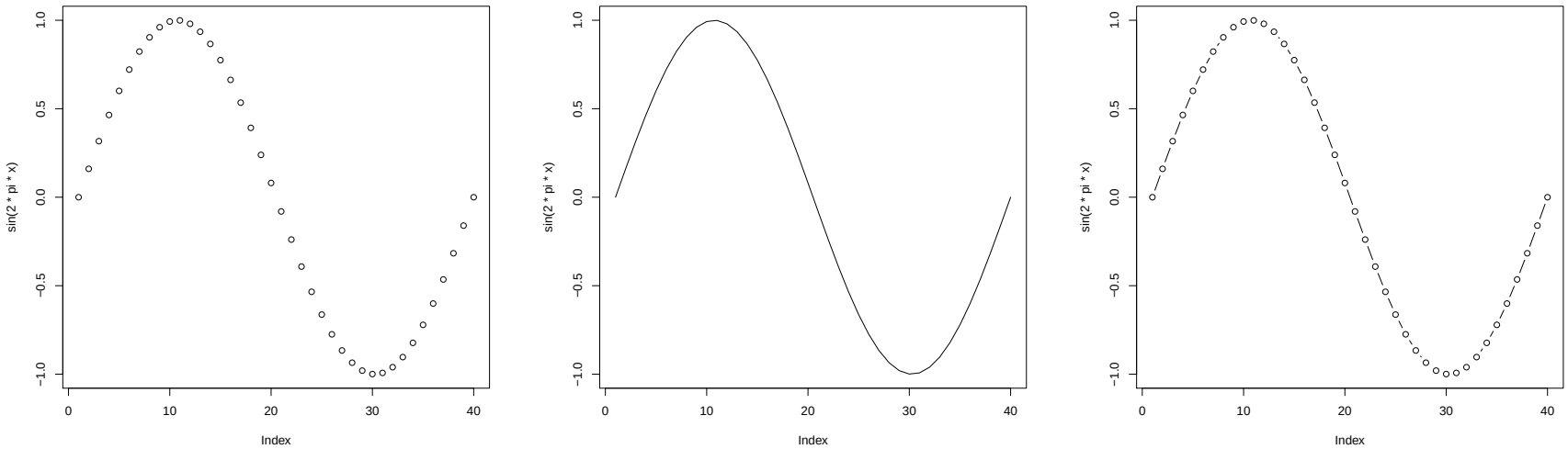
Основната функција за цртање на екран во R е `plot`. Можните аргументи на `plot` вклучуваат:

- `x` и `y`: координати (`y` може да се испушти);
- `xlim=c(lo, hi)` и `ylim=c(lo, hi)`: опсег на вредностите на координатните оски;
- `main` и `sub`: наслов и поднаслов;
- `xlab` и `ylab`: опис на оските;
- `type="c"`: тип (`"p"`, `"h"`, ...);
- `lty` и `lwd`: тип и ширина на линија;
- `pch`: знак за бележење на точки;
- `col`: боја;
- ...и многу други. Видете `?plot`, `?plot.default`, и `?par`.

# plot како основна функција за цртање

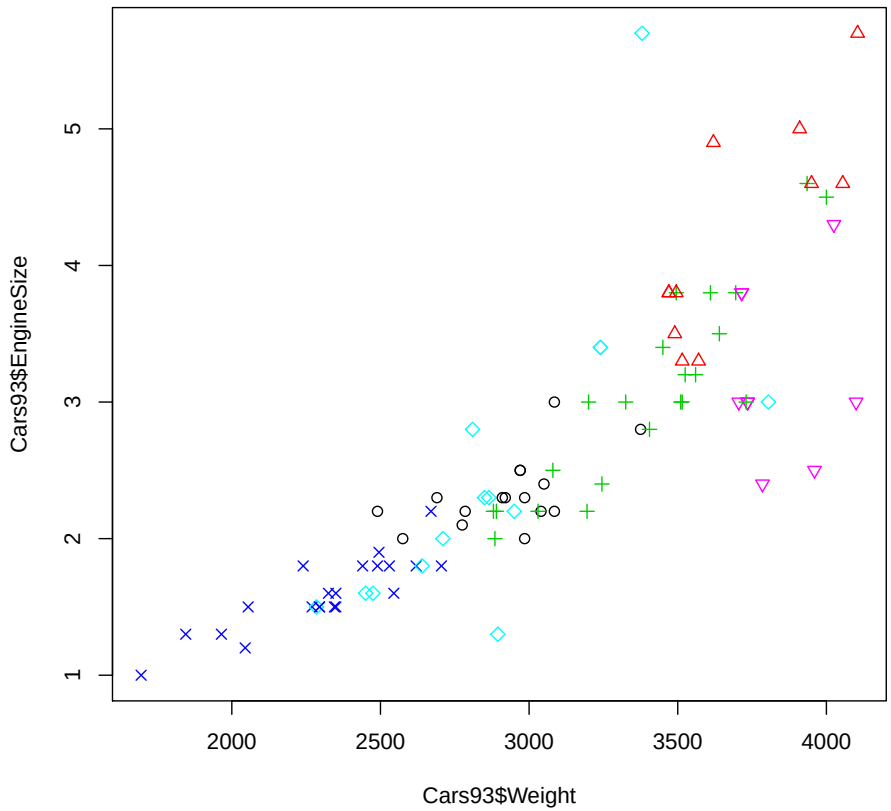
## Пример

```
> x <- seq(0, 1, length=30)
> plot(sin(2 * pi * x))                    # Точки.
> plot(sin(2 * pi * x), type="l")          # Линии.
> plot(sin(2 * pi * x), type="b")          # Точки и линии.
```




Слика: Резултати од различните plot типови.

```
> plot(Cars93$Weight, Cars93$EngineSize,
+       col=as.numeric(Cars93$Type),
+       pch=as.numeric(Cars93$Type))
```



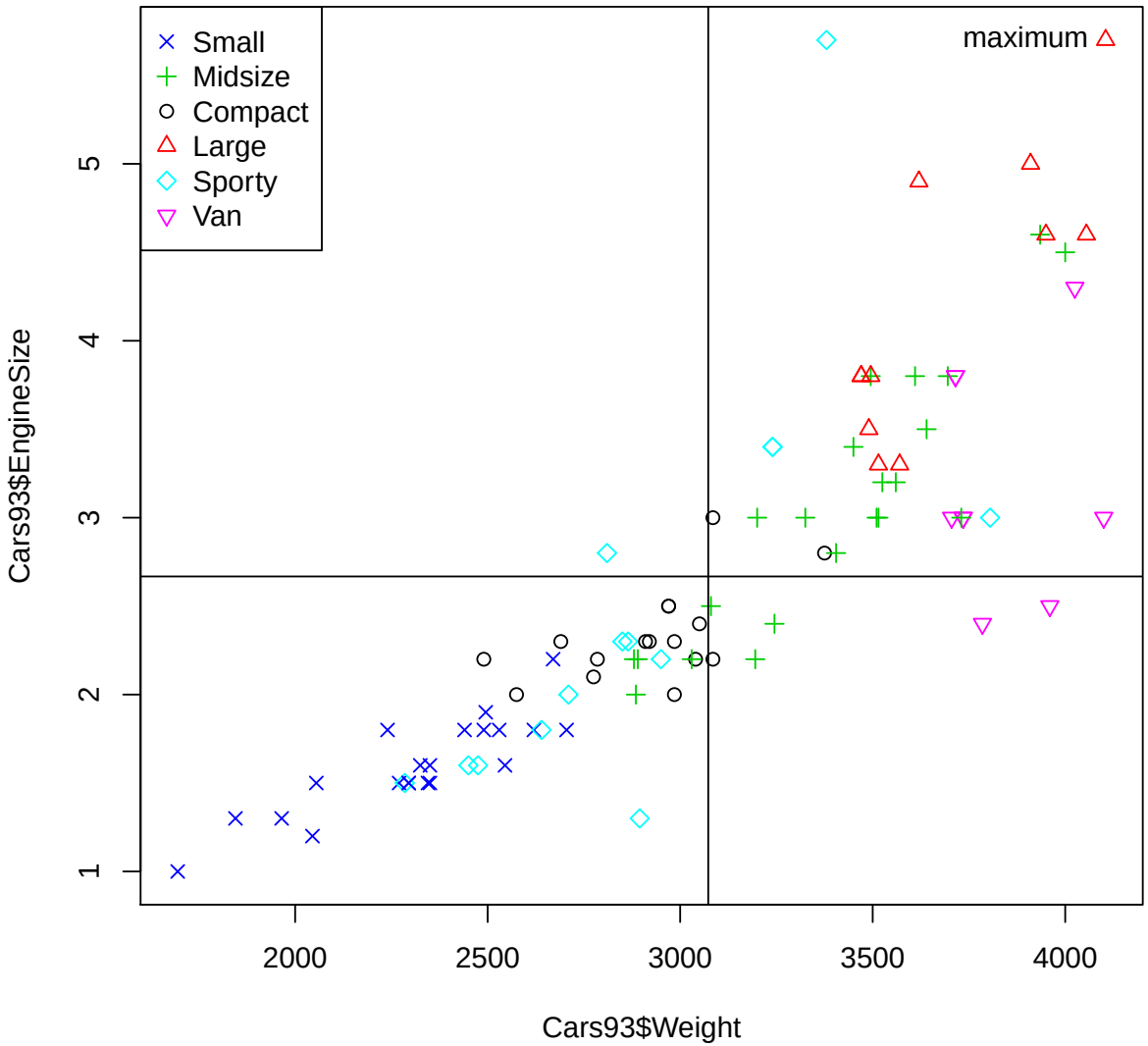

Слика: Тежина наспрема големина на мотор.

# Додавање на текст, линии, легенда...

Постојат функции кои додаваат елементи на веќе отворен цртеж.

- `text`: додавање на текст;
- `legend`: додавање на легенда;
- `lines` и `abline`: додавање на произволни и прави линии соодветно;
- `points` и `rug`: додавање на точки односно rugs.

```
> text(max(Cars93$Weight), max(Cars93$EngineSize),
+      "maximum", pos="2")  # pos="2" значи „лево од“
> abline(v=mean(Cars93$Weight))
> abline(h=mean(Cars93$EngineSize))
> legend("topleft", legend=unique(Cars93$Type),
+        pch=unique(Cars93$Type),
+        col=unique(Cars93$Type))
```

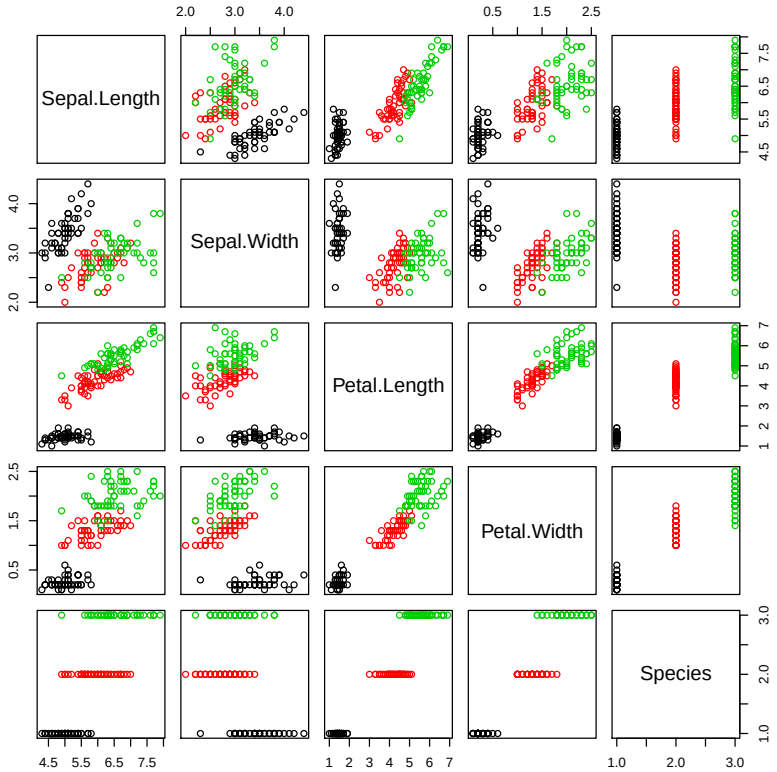



Слика: Цртежот со додаден текст и легенда.

# Матрица од цртежи

Доколку `plot` ја повикаме со data frame, на сликата ќе биде исцртена матрица од цртежи—еден за секој пар колони. Истото може да се постигне со функцијата `pairs` за произволни податоци.

```
> plot(iris, col=iris$Species)   # или pairs(iris)
```



# Хистограми

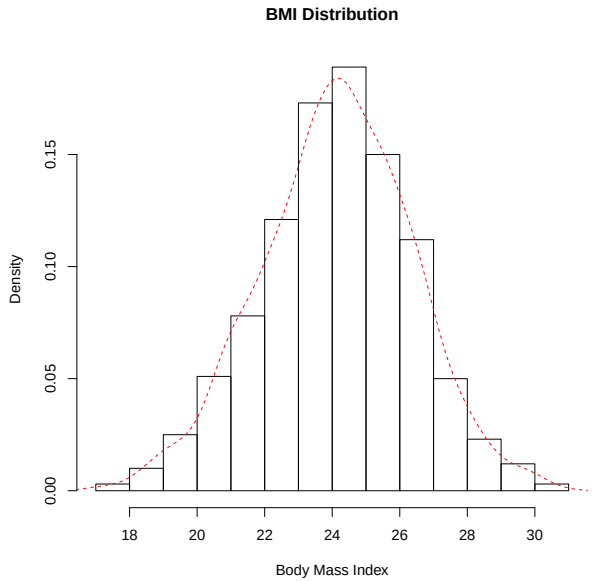
Функцијата за цртање хистограми е `hist`. Освен тоа што го црта хистограмот, враќа и листа со информации за него.

- `x`: вектор за кој да се направи хистограм.
- `breaks`: контрола на групирањето вредности:
  - нумерички вектор со опсег за секоја група;
  - број на групи (опсезите автоматски ќе бидат одредени);
  - име на алгоритам за групирање (`"Sturges"`, `"Scott"`, `"Freedman-Diaconis"`; default `"Sturges"`).
- `freq`: ако е `TRUE`, се цртаат фреквенции, инаку се црта густина. По default е `TRUE` кога `breaks` се на еднаква оддалеченост.
- `plot`: ако е `FALSE`, хистограмот нема да биде исцртан. Кога е `TRUE`, може да се пратат дополнителни аргументи како `col`, `xlim`, `ylim`, ...

# Хистограми

## Пример

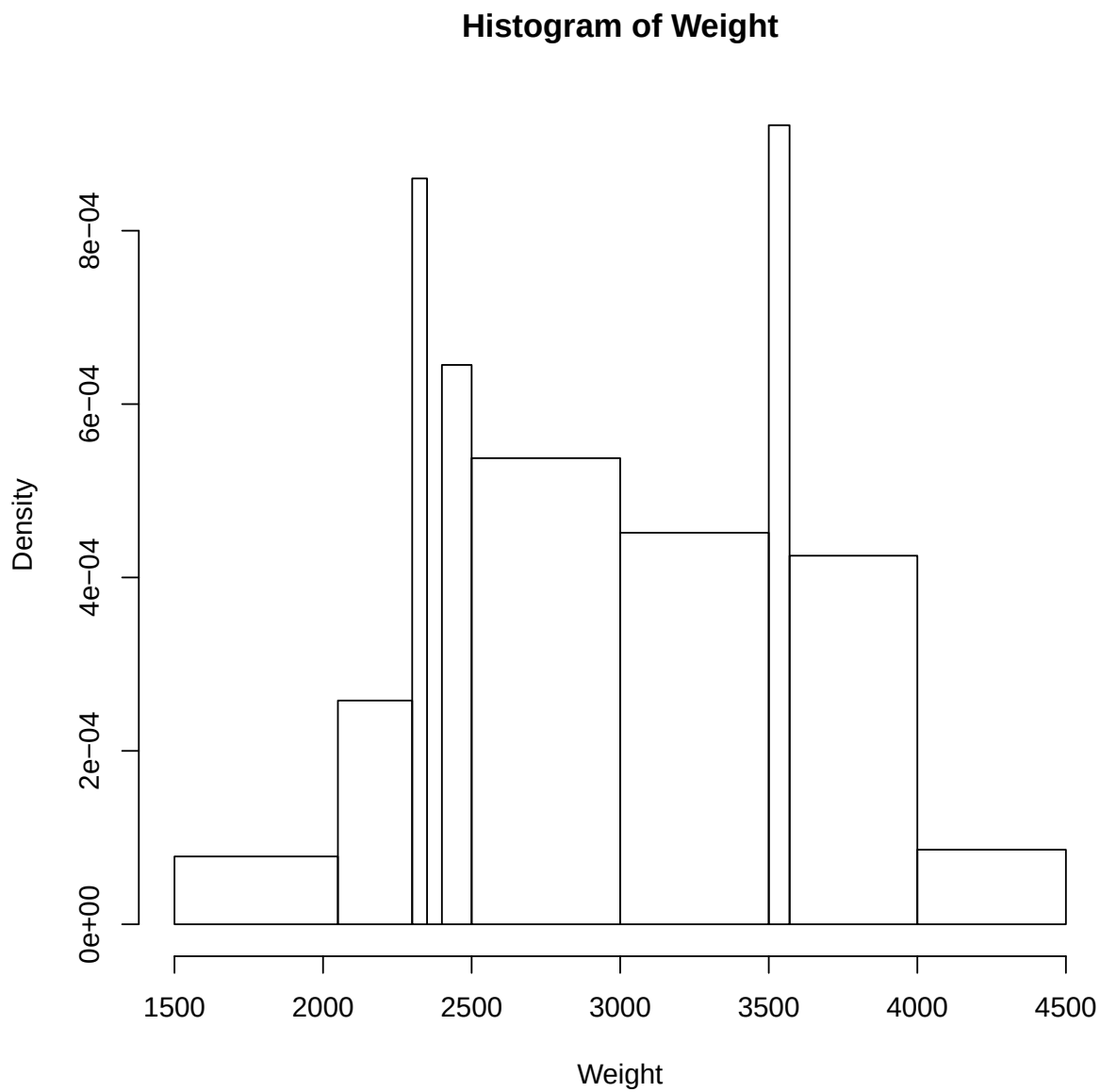
```
> BMI <- rnorm(10^3, mean=24.2, sd=2.2)
> BMI.hist <- hist(BMI, freq=FALSE, xlab="Body Mass Index",
+                  main="BMI Distribution")
> lines(density(BMI), lty=2, col="red")
> sum(diff(BMI.hist$breaks) * BMI.hist$density) # ∫_{-∞}^{∞}(hist) d x
[1] 1
```



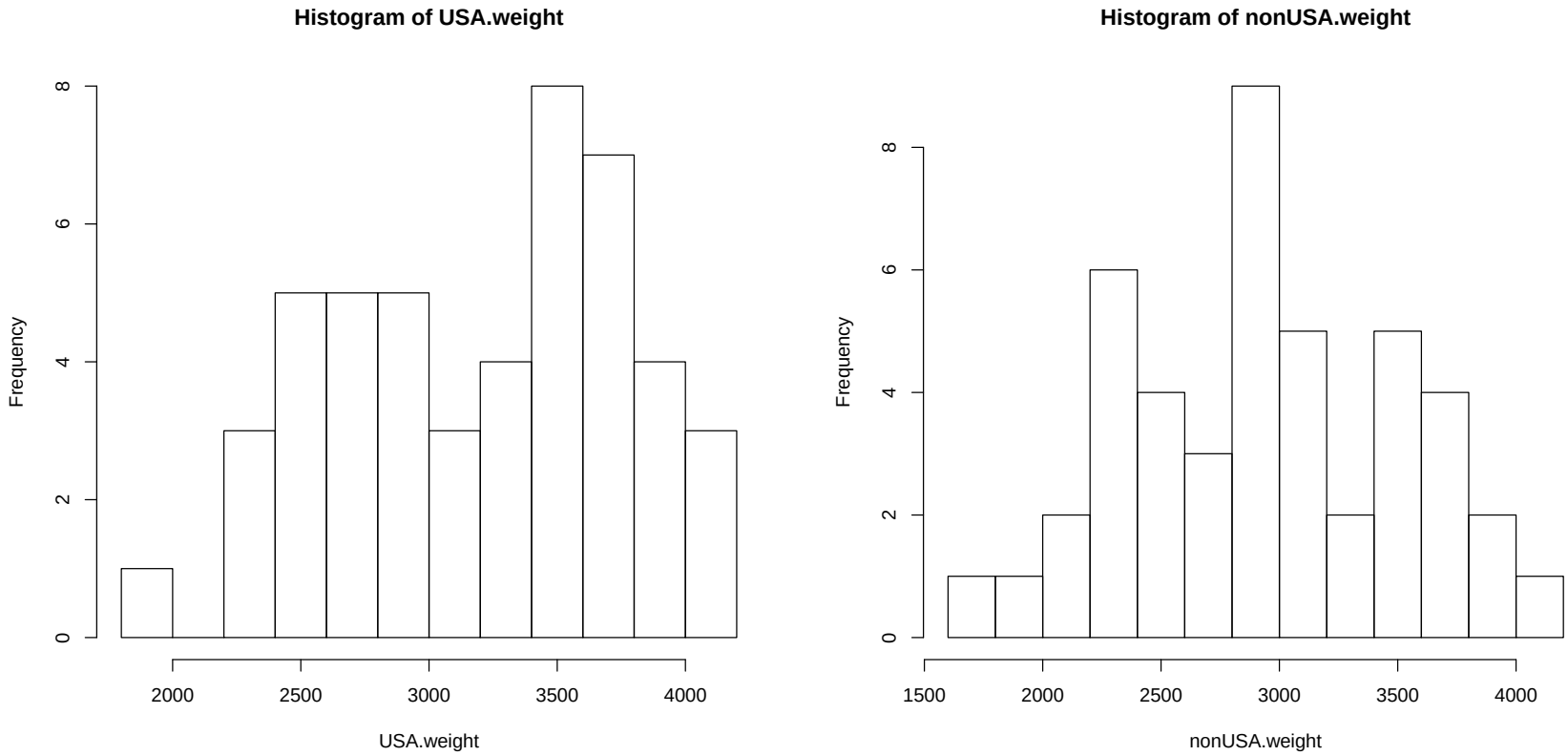
# Хистограми

## Пример

```
> hist(Cars93$Weight,
+      breaks=c(1500, 2050, 2300, 2350, 2400, 2500,
+      3000, 3500, 3570, 4000, 4500),
+      xlab="Weight", main="Histogram of Weight")
> USA.weight <- Cars93$Weight[Cars93$Origin == "USA"]
> nonUSA.weight <- Cars93$Weight[Cars93$Origin == "non-USA"]
> hist(USA.weight, breaks=10)
> hist(nonUSA.weight, breaks=10)
```




Слика: Хистограм од `Cars93$Weight`.




Слика: Тежини на автомобилите од САД и од други земји.

# Boxplots

Boxplots се користат за графички приказ на групи преку нивните квантили.

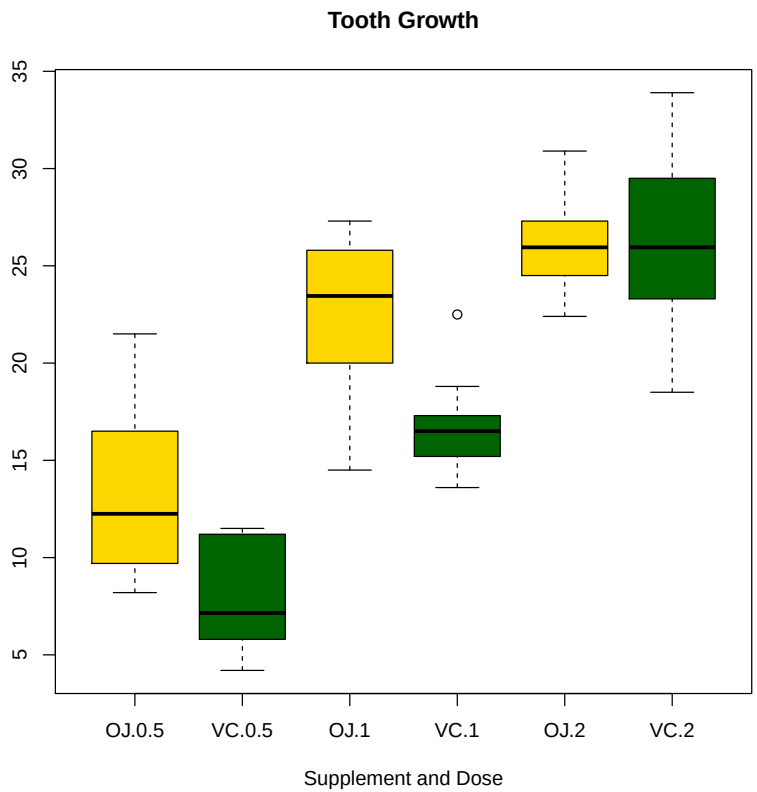
- Освен box, секој boxplot содржи и вертикални линии (*whiskers*) кои ја кажуваат варијабилноста во пониските квантили.
- Box-от го зафаќа целиот IQR, каде медијаната е означена со затемнета линија.
- Whiskers-ите се протегаат сè до $1.5 \cdot \text{IQR}$.
- Податоците надвор од $1.5 \cdot \text{IQR}$ се познати како *outliers*.

Во R тие се цртаат со функцијата `boxplot`. Таа како аргумент прифаќа *формула*: објект од типот на `value ~ group`, што кажува да се нацрта boxplot за `value` за секоја вредност од `group`.

# Boxplots

## Пример

```
> boxplot(len ~ supp * dose, data=ToothGrowth,
+         col=c("gold", "darkgreen"),
+         main="Tooth Growth", xlab="Supplement and Dose")
> boxplot(trees$Girth, main="Tree Girth", xlab="Girth")
```




# Q–Q (квантил–квантил) plot

Q–Q plot е метод за графичка споредба на две распределби преку споредба на нивните квантили.

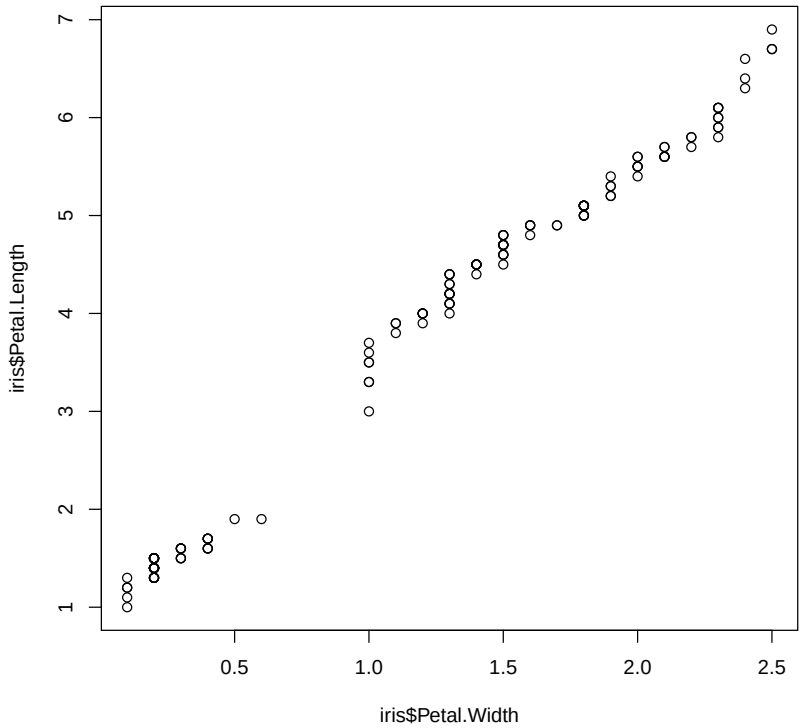
- Една точка $(x, y)$ во Q–Q plot-от соодветствува на еден од квантилите на втората распределба ($y$-координата) наспрема истиот квантил од првата распределба ($x$-координата).
- Ако квантилите се слични, точките од Q–Q plot-от би лежеле близу линијата $y = x$.
- Доколку постои линеарна зависност помеѓу распределбите, точките од Q–Q plot-от би лежеле на линија, но во општ случај не на $y = x$.

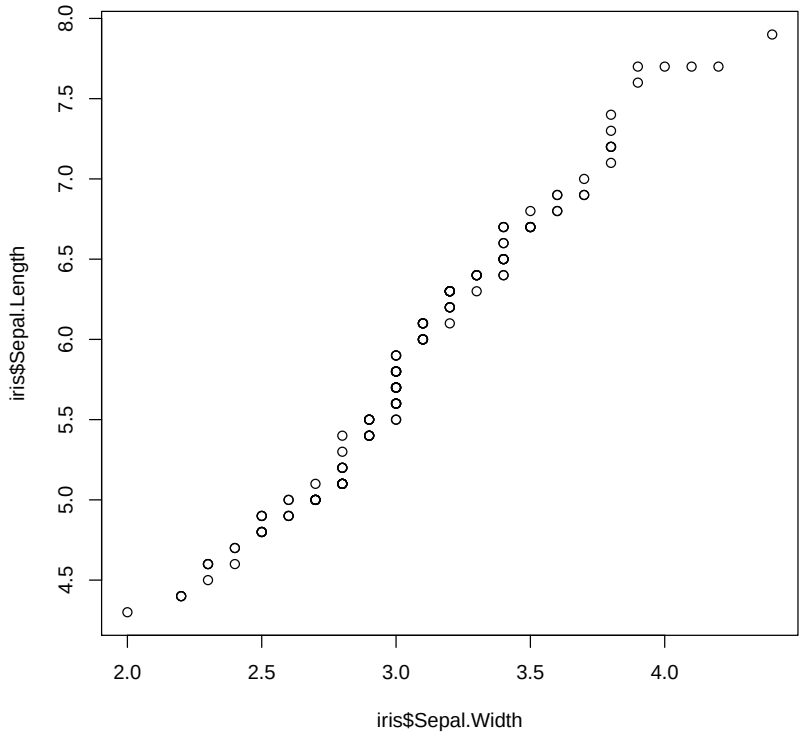
Во R се користат функциите `qqplot`, `qqnorm`, и `qqline`.

# Q–Q plot

## Пример

```
> qqplot(iris$Petal.Width, iris$Petal.Length)
> qqplot(iris$Sepal.Width, iris$Sepal.Length)
```

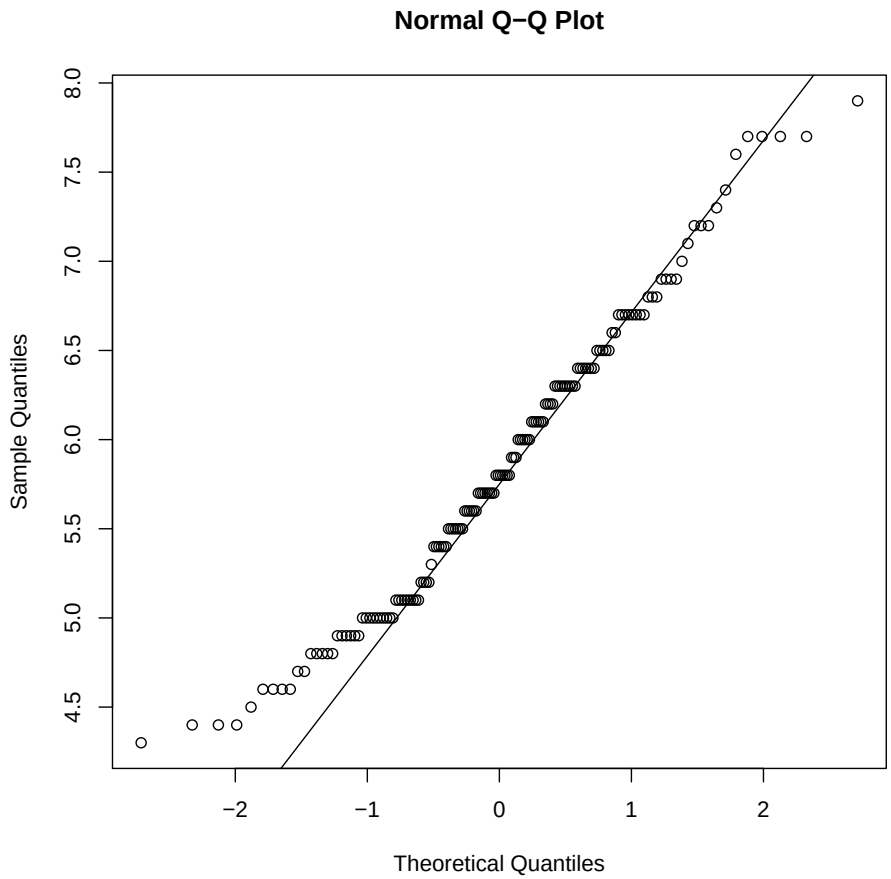


# Q–Q plot

## Пример

```
> qqnorm(iris$Sepal.Length)
> qqline(iris$Sepal.Length)
```



# Централна гранична теорема

Централната гранична теорема е една од фундаменталните теореми во веројатноста. Таа, освен тоа што е од теориски интерес, наоѓа голема примена во статистиката.

**Теорема (Централна гранична теорема, Линдеберг–Леви)**

*Нека* $(X_1, X_2, \ldots, X_n)$ *биде случаен примерок од обележје* $X$ *со средна вредност* $\mu$ *и варијанса* $\sigma^2 < \infty$. *Тогаш*

$$\sqrt{\frac{n}{\sigma^2}}\left(\left(\frac{\sum_{k=1}^{n} X_i}{n}\right) - \mu\right) \xrightarrow{d} \mathcal{N}(0,1),$$

*каде* $\xrightarrow{d}$ *означува конвергенција во распределба.*

# Централна гранична теорема

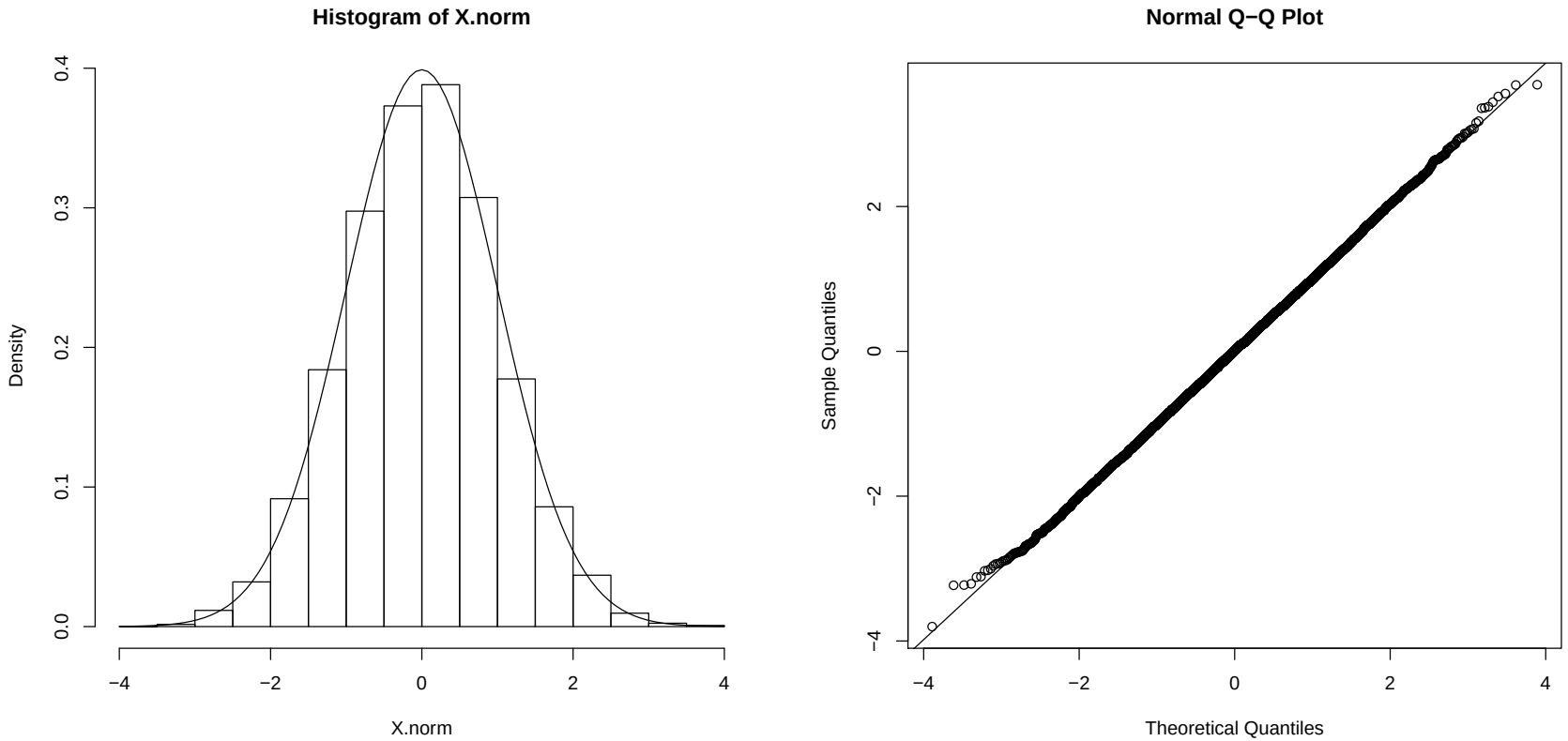
## Илустрација

За да ја илустрираме централната гранична теорема, ќе ја исцртаме распределбата на стандардизираната средна вредност од сто $\mathcal{U}(0,1)$ случајни променливи.

```
> m <- 10^4
> n <- 100
> X <- matrix(runif(m * n), nrow=m)
> X.norm <- (apply(X, 1, sum) - 0.5 * n) / sqrt(n / 12)
> # Верификација со хистограм
> hist(X.norm, freq=FALSE)
> curve(dnorm(x), add=TRUE)
> # Верификација со Q-Q plot
> qqnorm(X.norm)
> qqline(X.norm)
```

# Централна гранична теорема

## Илустрација




Слика: Верификација на централната гранична теорема.

# Содржина



# Метод на моменти

Методот на моменти врши оценување на параметрите на една распределба преку изедначување на теориските со моментите на примерокот.

## Дефиниција (Метод на моменти)

Нека $(X_1, X_2, \ldots, X_n)$ е случаен примерок од обележје $X$ чија распределба зависи од параметри $\theta_1, \theta_2, \ldots, \theta_m$. Оценувачи по метод на моменти $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_m$ се добиваат така што $m$ теориски моменти од распределбата на $X$ ќе се изедначат со истите $m$ моменти на примерокот, и равенките ќе се решат по непознатите параметри.

# Метод на моменти

## Пример во R

Веќе ги имаме разработено сите потребни функции за методот на моменти да го примениме во R. На пример, знаеме дека ако $X \sim \text{Pois}(\lambda)$, тогаш $\mathbb{E}[X] = \lambda$ и $\mathbb{D}[X] = \lambda$. Оттука добиваме

$$\hat{\lambda} = \bar{X}, \text{ или}$$
$$\hat{\lambda} = S^2.$$

```
> lambda <- 5
> x <- rpois(10^5, lambda)
> mean(x)
[1] 5.00487
> var(x)
[1] 5.002956
```

# Метод на моменти

## Пример во R

Ако $X \sim \mathcal{N}(\mu, \sigma^2)$, добиваме дека

$$\hat{\mu} = \bar{X}$$

$$\hat{\sigma}^2 = \frac{n-1}{n} S^2.$$

```
> n <- 10^4
> mu <- 5; sd <- 2   # μ = 5, σ^2 = 4
> x <- rnorm(n, mu, sd)
> mean(x)
[1] 4.994426
> (n - 1) / n * var(x)
[1] 3.942171
```

# Метод на маскимална подобност

Методот на максимална подобност е специјален случај на *максимум а̄по̄сте̄риори* оценувач кој го наоѓа параметарот кој ја максимизира подобноста на дадената реализација на примерокот.

## Дефиниција (Метод на максимална подобност)

Нека $(x_1, x_2, \ldots, x_n)$ биде реализација на случајниот примерок $(X_1, X_2, \ldots, X_n)$ од обележјето $X$. Распределбата на $X$ припаѓа на фамилија распределби $\{p(\cdot \mid \theta) : \theta \in \Theta\}$. Ако ги фиксираме $(x_1, x_2, \ldots, x_n)$, и $p(x_1, x_2, \ldots, x_n \mid \theta)$ ја гледаме како функција од $\theta$, $\mathcal{L}(\theta \mid x_1, x_2, \ldots, x_n)$, тогаш оценувач за $\theta$ по методот на максимална подобност е

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta \mid x_1, x_2, \ldots, x_n).$$

# Метод на максимална подобност

Бидејќи случајниот примерок се состои од независни и идентично распределени случајни променливи,

$$\mathcal{L}(\theta \mid x_1, x_2, \ldots, x_n) = p(x_1, x_2, \ldots, x_n \mid \theta) = \prod_{k=1}^{n} p(x_k \mid \theta).$$

Често е поедноставно да се максимизира

$$\ln \mathcal{L}(\theta \mid x_1, x_2, \ldots, x_n) = \sum_{k=1}^{n} \ln p(x_k \mid \theta).$$

Оценувачите добиени по методот на максимална подобност поседуваат добри статистички својства и се често користени во пракса.

# Метод на максимална подобност

## Пример

Во R, ќе ја користиме функцијата `mle` од библиотеката `stats4`. Од нас единствено се бара да дефинираме функција која ќе пресметува $-\ln \mathcal{L}(\theta \mid x_1, x_2, \ldots, x_n)$ (да го забележиме минусот напред). Доколку имаме примерок од експоненцијална распределба:

```
> library(stats4)
> x <- rexp(10^4, rate=3)
> nll <- function(lambda=0.1) {
+   -sum(dexp(x, rate=lambda, log=TRUE))   # -∑_{k=1}^{n} ln p(x_k | λ).
+ }
> mle(nll)
Coefficients:
  lambda
3.075704
```

# Метод на максимална подобност

## Пример

Функцијата `mle` врши нумеричка оптимизација на дефинираната од нас функција `nll` и го враќа минимумот.

- Потребна е почетна вредност за параметрите, која може да биде зададена во дефиницијата на функцијата или преку аргументот `start`.
- Најчесто, почетната вредност не игра никаква улога.

```
> y <- rgamma(10^4, shape=3, rate=2)
> nll <- function(shape, rate) {
+   -sum(dgamma(y, shape, rate, log=TRUE))
+ }
> mle(nll, start=list(shape=1, rate=1))
Coefficients:
   shape     rate
3.013568 2.013409
```