

Duomenų analizės įvadas

2.2. dalis - R programavimas

Justas Mundeikis

VU EVAF

2019-04-12

Turinys

- 1 Loop funkcijos
- 2 rep
- 3 sweep
- 4 lapply
- 5 sapply
- 6 mapply
- 7 rapply
- 8 tapply
- 9 split
- 10 aggregate
- 11 Skirstiniai

Loop funkcijos

Loop funkcijos

Rašant skriptus, `for`, `while` ir kiti loopai yra tinkami, bet jeigu norima parašyti kodą tiesiog konsolėje, tada susiduriama su daug problemų.

- `apply`: taiko funkciją masyvo stulpeliams / eilutėms
- `lapply`: loopina per list ir paleidžia funkciją kiekvienam elementui
- `sapply`: veikia kaip ir `lapply` tik supaprastina rezultatus
- `tapply`: taiko funkciją vektorių dalims
- `mapply`: multivariatinė `lapply` versija

apply

- apply naudojama taikyti funkcijas duomenų masyvų (array: dataframe, matrix) eilutėms ar stulpeliams.
- apply iš esmės supaprastina for loop naudojimą.

```
args(apply)
## function (X, MARGIN, FUN, ...)
## NULL
# kur X yra array
# MARGIN=1 eilutėms
# MARGIN=2 stulpeliams
# FUN taikoma funkcija
```

apply

- `apply` taikymas matricų eilutėms ir stulpeliams

```
x <- matrix(1:4,2,2)
x
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
apply(x, 1, mean) # 1 - eilutėms
## [1] 2 3
apply(x, 2, mean) # 2 - stulpeliams
## [1] 1.5 3.5
apply(x, 1, sum) # 1 - eilutėms
## [1] 4 6
apply(x, 2, sum) # 2 - stulpeliams
## [1] 3 7
```

apply

- Norint pritaikyti apply funkciją daugiau dimensijų turinčiam duomenų masyvui, būtina nurodyti vektorių, kurios dimensijos išlaikomos

```
x <- array(data=rnorm(40), dim = c(2,2,10))  
# išlaikoma 1 ir 2 dimensijos,  
# jungiame per 3 dimensiją  
apply(x, c(1,2), mean)  
##           [,1]      [,2]  
## [1,] -0.4169804 -0.2678529  
## [2,] -0.2651032  0.2040710
```

apply

Exercise:

- create a sales dataframe
- calculate sum of sales using apply
 - for every month
 - for every sales-person

```
#generate sales dataframe  
set.seed(101)  
sales <- data.frame(row.names = month.abb[1:12],  
                    Marry=sample(0:100,12, replace = TRUE),  
                    John=sample(100:200,12, replace = TRUE),  
                    Felix=sample(150:300,12, replace = TRUE),  
                    Jenny=sample(0:400,12, replace = TRUE))
```


apply

Solution:

```
set.seed(101)
sales <- data.frame(row.names = month.abb[1:12],
                    Marry=sample(0:100,12, replace = TRUE),
                    John=sample(100:200,12, replace = TRUE),
                    Felix=sample(150:300,12, replace = TRUE),
                    Jenny=sample(0:400,12, replace = TRUE))

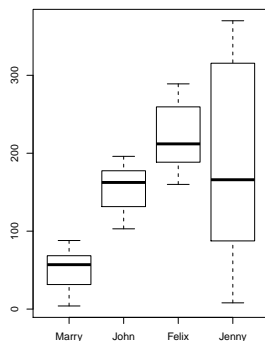
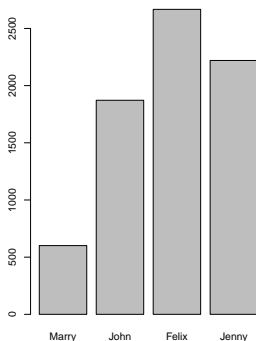
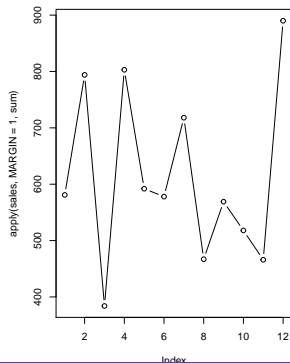
apply(sales, MARGIN=1, sum)
## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 581 794 384 803 592 578 718 467 569 518 466 890
apply(sales, MARGIN=2, sum)
## Marry John Felix Jenny
## 601 1872 2667 2220
```

apply

Exercise: build the following graphs:

- line chart with dots for sum of monthly sales (hint: use `?plot`)
- bar chart for every sales-person's sum of yearly turnover
- boxplot for every sales person's monthly sales

Solutions should look like:



apply

Boxplotas:

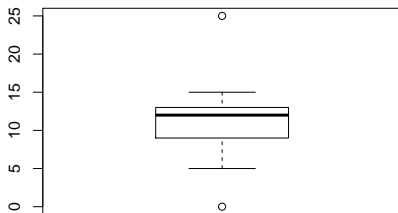
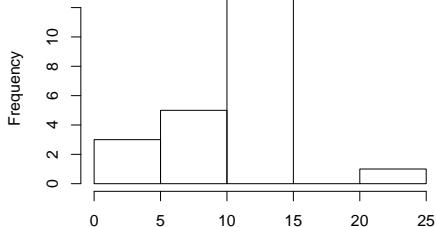
- duomenys suskirstomi didėjimo tvarka
- $1Q$ = reikšmė ties 25% visų stebėjimu (dežutės apačia)
- $2Q$ = mediana = reikšmė ties 50% visų stebėjimu (storas brūkšnys)
- $3Q$ = reikšmė ties 75% visų stebėjimu (dežutės viršus)
- $IQR = 3Q - 1Q$
- Whiskers $1Q - 1.5IQR$, $3Q + 1.5IQR$, bet brežiama iki tiek, kiek yra reiškmių
- Outlayeriai, kas žemiau, aukščiau Whiskers

apply

Boxplotas:

```
c <- c(sample(5:15, 20, replace = TRUE), 0, 25)
c
## [1] 13  6  5  9 12 13  9 13 12 10 12 11 13 13 11 15 14 12  5  8  0 25
mean(c)
## [1] 10.95455
sd(c)
## [1] 4.785666
```

Histogram of c



apply

Jeigu norima apskaičiuoti dataframe / matricų eilučių ar stulpelių sumas / vidurkius, galima naudoti jau supaprastintas funkcijas, jos veikia dar greičiau, nes yra parašytas su c++, tačiau skirtumas pasijaučia tik su labai dideliais duomenų masyvais.

- `rowSums=apply(x,1,sum)`
- `rowMeans=apply(x,1,mean)`
- `colSums=apply(x,2,sum)`
- `colMeans=apply(x,2,mean)`

apply

apply priima ... taigi galima deleguoti papildomus funkcijų parametrus.

PVZ:

- **quantile**: The generic function quantile produces sample quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1.
- **summary**: summary is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

apply

PVZ:

```
apply(sales, 2, quantile, probs=c(0,0.25 ,0.5, 0.75,1))
```

```
##      Marry   John  Felix  Jenny
## 0%      4.00 103.00 160.00   8.00
## 25%     32.25 136.25 193.25  90.25
## 50%     57.00 162.50 212.00 166.00
## 75%     67.25 175.25 254.25 310.25
## 100%    88.00 196.00 289.00 370.00
```

```
apply(sales, 2, summary, digits=0)
```

```
##           Marry  John  Felix  Jenny
## Min.           4   100    200     8
## 1st Qu.        30   100    200    90
## Median         60   200    200   200
## Mean           50   200    200   200
## 3rd Qu.        70   200    300   300
## Max.           90   200    300   400
```

rep

rep

Replicate Elements of Vectors and Lists * rep - replicates the values in x.
It is a generic function. *?rep - for more details

```
args(rep)
rep(1:4, 2)
rep(1:4, each = 2)           # not the same.
rep(1:4, c(2,2,2,2))        # same as second.
rep(1:4, c(2,1,2,1))
rep(1:4, each = 2, len = 4)  # first 4 only.
rep(1:4, each = 2, len = 10) # 8 integers plus two recycled 1's.
rep(1:4, each = 2, times = 3) # length 24, 3 complete replications
```

sweep

sweep

Returns an array obtained from an input array by sweeping out a summary statistic.

```
args(sweep)  
## function (x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)  
## NULL
```

sweep

Standartizavimas:

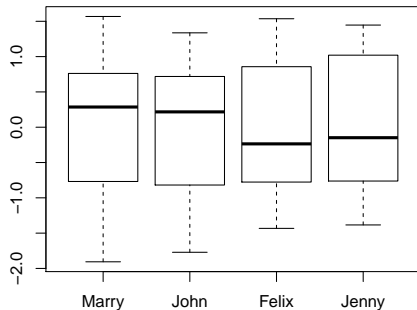
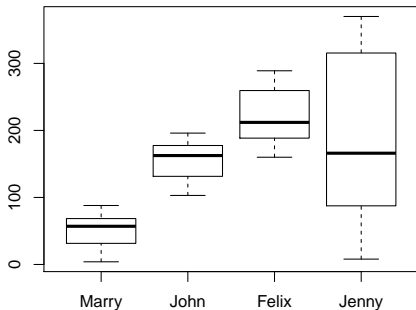
- Kartais yra patogiau naudotis standartizuotais duomenimis
- $z = \frac{X - \mu}{\sigma}$
- z - kiek vertė X yra nutolusi nuo vidurkio kartų, matuojant standartiniais nuokrypiais
- $\text{sales_mean} = \mu$
- $\text{sales_sd} = \sigma$
- taikant prielaidą dėl normaliojo skirstinio:
 - 95% visų duomenų $\pm \mu$

PVZ:

```
sales_mean <- apply(sales, 2, mean)
sales_sd <- apply(sales, 2, sd)
df1 <- sweep(sales, 2, sales_mean, "-")
df2 <- sweep(df1, 2, sales_sd, "/")
```

sweep

```
par(mfrow=c(1,2))  
boxplot(sales)  
boxplot(df2)
```



lapply

lapply

- lapply priima 3 argumentus:
 - 1 list objektą,
 - 2 funkciją arba funkcijos pavadinimą,
 - 3 galimus funkcijos papildomus argumentus
- lapply visad grąžina list klasės objektą
- Jeigu X nėra list, tada R bando paversti X list objektu.

lapply

```
set.seed(101)
x <- list(a=1:10,
          b=rnorm(50),
          c=seq(from=100, to=200, by=2))

lapply(x, mean)
## $a
## [1] 5.5
##
## $b
## [1] -0.1239708
##
## $c
## [1] 150
```


lapply

```
y <- 1:3
#taip R/lapply mato vektorių x konvertuvas jį į list objektą
#list susideda iš trijų elementų kiekvienas elementas turi savo reikšmę
as.list(1:3)
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

lapply

- `runif` generuoja tolygiojo skirstinio atsitiktinius skaičius (`runif(n, min = 0, max = 1)`)

```
lapply(y, runif)
## [[1]]
## [1] 0.1250194
##
## [[2]]
## [1] 0.02332669 0.39186128
##
## [[3]]
## [1] 0.8595986 0.7183345 0.3393950
```

lapply

Išnaudojant ... galime perleisti papildomus argumentus runif funkcijai:

```
lapply(y, runif, min=5, max=10)
## [[1]]
## [1] 5.406107
##
## [[2]]
## [1] 5.186172 8.865444
##
## [[3]]
## [1] 9.975411 5.732932 5.199224
```

lapply

lapply ir kitos apply funkcijos gali naudotis USER DEFINED FUNCTION (UDF), t.y. niekur kitur nedefinuotomis funkcijomis

```
z <- list(a=matrix(1:9, nrow=3, ncol = 3),
          b=matrix(1:4, nrow = 2, ncol=2))
#elt yra anoniminė funkcija. vietoj "elt" galėtų būti "x",
#arba "abrakadabra"
lapply(z, function(elt) elt[,1, drop=FALSE])
## $a
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
##
## $b
##      [,1]
## [1,]    1
## [2,]    2
```

lapply

Klausimas: ką generuoja ši lapply funkcija?

```
set.seed(101)
A <- matrix(sample(1:10,9),3,3)
B <- matrix(sample(1:10,9),3,3)
C <- matrix(sample(1:10,9),3,3)
MAT_LIST <- list(A,B,C)
```

```
lapply(MAT_LIST, "[", ,1)
## [[1]]
## [1] 4 1 6
##
## [[2]]
## [1] 6 8 10
##
## [[3]]
## [1] 5 1 6
```

lapply

Exercise: Use `lapply` to calculate the sum of sales for every sales-person

Solution:

```
lapply(sales, sum)
## $Marry
## [1] 601
##
## $John
## [1] 1872
##
## $Felix
## [1] 2667
##
## $Jenny
## [1] 2220
```

supply

sapply

sapply bando supaprastinti lapply rezultatus (jeigu įmanoma)

- jeigu lapply grąžintų list, kurių kiekvienas elementas yra 1 ilgumo, tada sapply grąžina vektorių
- jeigu lapply grąžintų list, kurių kiekvienas elementas yra >1 ir vienodo ilgumo, tada sapply grąžina matricą
- jeigu netinka pirma du variantai, grąžina list kaip ir lapply

supply

```
set.seed(101)
x <- list(a=1:10,
          b=rnorm(50),
          c=seq(from=100, to=200, by=2))

lapply(x, mean)
## $a
## [1] 5.5
##
## $b
## [1] -0.1239708
##
## $c
## [1] 150

supply(x, mean)
##           a           b           c
## 5.5000000 -0.1239708 150.0000000
```

sapply

```
sapply(MAT_LIST,"[,1,1, simplify = TRUE) # nepriima tuščių argumentų
## [1] 4 6 5
sapply(MAT_LIST,"[,1,1, simplify = FALSE) # nepriima tuščių argumentų
## [[1]]
## [1] 4
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 5
```

sapply

Exercise: Use `sapply` to calculate sales for every sales-person

Solution:

```
sapply(sales, sum)
## Marry   John  Felix  Jenny
##    601   1872   2667   2220
```

mapply

mapply

- mapply taiko paraleliai (vienu metu) funkcijų skirtingiems argumentams naudojantis list arba vektoriais
- “m” - multivariate

```
str(mapply)
## function (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

- FUN yra funkcija, kuri bus taikoma
- ... argumentai, kuriais naudojamosi funkcijoje
- MoreArgs kiti FUN argumentai
- SIMPLIFY ar rezultatas turėtų būti simplifikuotas kaip sapply

mapply

Jeigu norime sukurti tokį list objektą, 4 kartus rašome rep(), su argumentais 1-4 ir 4-1

```
list(rep(1,4), rep(2,3), rep(3,2), rep(4,1))  
## [[1]]  
## [1] 1 1 1 1  
##  
## [[2]]  
## [1] 2 2 2  
##  
## [[3]]  
## [1] 3 3  
##  
## [[4]]  
## [1] 4
```

mapply

Supaprastinant galima naudoti `mapply` funkciją, kurios argumentai `rep` funkcija ir du vektoriai `1:4` ir `4:1`

```
mapply(rep, 1:4, 4:1)
## [[1]]
## [1] 1 1 1 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3
##
## [[4]]
## [1] 4
```

mapply

Funkcija `noise` generuoja `n` atsitiktinių normaliojo skirstinio skaičių su vidurkiu `mean` ir standartiniu nuokrypiu `sd`

```
noise <- function(n, mean, sd){  
  rnorm(n, mean, sd)  
}  
  
noise(4,1,0.01) # veikia kaip tikėtasi  
## [1] 0.9884974 0.9972553 1.0057790 0.9860310  
# list(noise(1,1,0.01),noise(2,2,0.01),noise(3,3,0.01),noise(4,4,0.01))  
noise(1:4,1:4,0.01) # veikia ne kaip tikėtasi  
## [1] 1.007491 1.989488 3.001654 4.011298
```


mapply

Šioje vietoje galima naudotis `mapply` tam, kad funkcija priimtų argumentus iš vektorių

```
# dabar norime generuoti tokį list objektą  
# list(noise(1,1,0.01),noise(2,2,0.01),noise(3,3,0.01),noise(4,4,0.01))  
mapply(noise, 1:4, 1:4, 0.01)  
## [[1]]  
## [1] 1.011737  
##  
## [[2]]  
## [1] 1.995721 1.997402  
##  
## [[3]]  
## [1] 2.985888 2.993586 3.001125  
##  
## [[4]]  
## [1] 4.004226 4.003868 3.993122 4.001489
```

mapply

Exercise:

- Generate following matrix using `mapply`

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

rapply

rapply

rapply yra *recursive apply* ir taikomas list objektams

```
args(rapply)
## function (object, f, classes = "ANY", deflt = NULL, how = c("unlist",
##      "replace", "list"), ...)
## NULL

# PVZ:
z <- list(1,2,3,4)
rapply(z, function(x){x^2})
## [1] 1 4 9 16

# jeigu funkcija telpa į vieną eilutę, galime be {}
rapply(z, function(x) x^2)
## [1] 1 4 9 16
```

rapply

Exercise:

- Calculate the bonus for each salesperson
- $\text{Bonus} = 0.15 * \text{average sales of the year}$ (use `rapply` and either `lapply` or `sapply`)

rapply

Solutions:

```
#kai visi keliai veda į Romą :)
rapply(lapply(sales, mean), function(x) x*0.15)
##   Marry   John   Felix   Jenny
## 7.5125 23.4000 33.3375 27.7500
rapply(sapply(sales, mean, simplify = FALSE), function(x) x*0.15)
##   Marry   John   Felix   Jenny
## 7.5125 23.4000 33.3375 27.7500
rapply(sales, function(x) mean(x)*0.15)
##   Marry   John   Felix   Jenny
## 7.5125 23.4000 33.3375 27.7500
# aišku galima ir ...
apply(sales, 2, mean)*0.15
##   Marry   John   Felix   Jenny
## 7.5125 23.4000 33.3375 27.7500
sapply(sales, mean)*0.15
##   Marry   John   Felix   Jenny
## 7.5125 23.4000 33.3375 27.7500
```

rapplly

Papildomi rapplly pvz., savistudijoms

<https://www.r-bloggers.com/rapplly-function-explanation-and-examples>

tapply

tapply

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

```
str(tapply)
## function (X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
```

- X yra vektorius
- INDEX faktorius arba faktorių list
- FUN taikoma funkcija
- ... papildomi FUN argumentai
- simplify ar supaprastinti rezultatus

tapply

```

set.seed(101)
x <- c(rnorm(10), runif(10, min=5, max=15), rnorm(10, mean = 100))
x
## [1] -0.3260365 0.5524619 -0.6749438 0.2143595 0.3107692
## [6] 1.1739663 0.6187899 -0.1127343 0.9170283 -0.2232594
## [11] 12.0071155 14.5683746 7.1335200 11.6106150 14.2331888
## [16] 12.9571976 5.7121255 8.8940777 9.0645122 11.5935508
## [21] 99.8066620 99.1502453 100.0584655 99.1823296 97.9496922
## [26] 99.8362443 100.7085221 99.7320195 98.5360782 100.7444358
# Generate factors by specifying the pattern of their levels.
# gl(n, k, length = n*k, labels = seq_len(n), ordered = FALSE)
f <- gl(3,10)
f
## [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
## Levels: 1 2 3

```

tapply

Supjaustome x vektorių pagal faktorių vektorių f ir pritaikome `mean`

```
tapply(x, f, mean)
##           1           2           3
## 0.2450401 10.7774278 99.5704695
tapply(x, f, mean, simplify = FALSE)
## $`1`
## [1] 0.2450401
##
## $`2`
## [1] 10.77743
##
## $`3`
## [1] 99.57047
```

tapply

Supjaustome x vektorių pagal faktorių vektorių f ir pritaikome summary

```
tapply(x, f, summary)
## $`1`
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
## -0.6749 -0.1956   0.2626   0.2450   0.6022   1.1740
##
## $`2`
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##    5.712    8.937   11.602   10.777   12.720   14.568
##
## $`3`
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
##    97.95    99.16    99.77    99.57   100.00   100.74
```

split

split

`split` padalina vektorių arba kitą objektą į grupes priklausomai nuo faktorių arba faktorių list

- `x` vektorius / list / dataframe
- `f` faktorius arba faktorių list
- `drop` indikuoja, ar tušti faktoriai turėtų būti panaikinti

```
str(split)  
## function (x, f, drop = FALSE, ...)
```

split

```
split(x, f)
## $`1`
## [1] -0.3260365 0.5524619 -0.6749438 0.2143595 0.3107692 1.1739663
## [7] 0.6187899 -0.1127343 0.9170283 -0.2232594
##
## $`2`
## [1] 12.007115 14.568375 7.133520 11.610615 14.233189 12.957198 5.7121
## [8] 8.894078 9.064512 11.593551
##
## $`3`
## [1] 99.80666 99.15025 100.05847 99.18233 97.94969 99.83624 100.708
## [8] 99.73202 98.53608 100.74444
# dabar galima naudoti lapply / sapply
```

split

Taigi galime suskaidyti x į 3 list objektus ir tada kiekvienam atlikti `lapply`, `sapply` arba iškart `tapply`

```
lapply(split(x, f), mean)
## $`1`
## [1] 0.2450401
##
## $`2`
## [1] 10.77743
##
## $`3`
## [1] 99.57047
sapply(split(x, f), mean)
##           1           2           3
## 0.2450401 10.7774278 99.5704695
# tolygu:
tapply(x, f, mean)
##           1           2           3
## 0.2450401 10.7774278 99.5704695
```


split

- Pasitikriname ar loadina `airquality` dataset, jeigu ne `library(datasets)`
- Mūsų tikslas: apskaičiuoti 4 kintamųjų vidurkius: Ozone, Solar.R, Wind, Temp

```
head(airquality)
##      Ozone Solar.R Wind Temp Month Day
## 1      41      190  7.4   67     5   1
## 2      36      118  8.0   72     5   2
## 3      12      149 12.6   74     5   3
## 4      18      313 11.5   62     5   4
## 5      NA       NA 14.3   56     5   5
## 6      28       NA 14.9   66     5   6
```

split

Supjaustome airquality pagal mėnesius:

```
s <- split(airquality, airquality$Month)
```

split

- Pritaikome `sapply` funkciją (galima ir `lapply`, bet skaidrėse per daug vietos atima :D)
- Ką reikia pakeisti, jog negauti mėnesių ir dienų vidurkių, bei panaikinti NA?

```
sapply(s, function(x) colMeans(x))
```

```
##           5           6           7           8           9
## Ozone      NA        NA        NA        NA        NA
## Solar.R    NA 190.16667 216.483871      NA 167.4333
## Wind      11.62258  10.26667   8.941935  8.793548  10.1800
## Temp      65.54839  79.10000  83.903226  83.967742  76.9000
## Month      5.00000   6.00000   7.000000   8.000000   9.0000
## Day       16.00000  15.50000  16.000000  16.000000  15.5000
```

#tolygu

sapply(s, function(x) apply(x,2,mean))

split

Solution:

```

# pastaba: dataframe[1:n], pradėjo veikti kaip df[,1:n]
# tad kodas veikty colMeans(x[1:4], na.rm=TRUE) t.y. be ",",
# bet dirbkite tvarkingai su ",",
sapply(s, function(x) colMeans(x[,1:4], na.rm=TRUE))
##              5              6              7              8              9
## Ozone      23.61538   29.44444   59.115385   59.961538   31.44828
## Solar.R    181.29630  190.16667  216.483871  171.857143  167.43333
## Wind       11.62258   10.26667    8.941935    8.793548   10.18000
## Temp       65.54839   79.10000   83.903226   83.967742   76.90000
# ALT: sapply(s, function(x) apply(x[,1:4],2,mean, na.rm=TRUE))
# veikia ir
# sapply(s, function(x) colMeans(x[1:4], na.rm=TRUE))

```

split

Importuojame į R tikrus Lietuvos darbo užmokesčio duomenis:

(pastaba: dėl laiko stokos, duomenų surinkimas iš LSD ir EUROSTAT bus seminario uždavinyje, dabar naudotis clonuotame folderyje esančiu failu)

```
#bandant atkartoti, pasikeisti relatyvų/absoliutų adresą!!!
df <- read.csv("./duomenys_paskaitoms/Darbo_užmokestis_mėnesinis_1995_2017.
head(df,5)
```

##	Laikotarpis	Rodiklis	Tipas	Lytis
## 1	1995 Darbo užmokestis (mėnesinis)	Bruto	Vyrai ir moterys	
## 2	1995 Darbo užmokestis (mėnesinis)	Bruto	Vyrai ir moterys	
## 3	1995 Darbo užmokestis (mėnesinis)	Bruto	Vyrai ir moterys	
## 4	1995 Darbo užmokestis (mėnesinis)	Bruto	Vyrai	
## 5	1995 Darbo užmokestis (mėnesinis)	Bruto	Vyrai	
##		Sektorius	Matavimo.vienetai	
## 1	Šalies ūkis su individualiosiomis įmonėmis		EUR	
## 2	Viešasis sektorius		EUR	
## 3	Privatusis sektorius su individualiosiomis įmonėmis		EUR	
## 4	Šalies ūkis su individualiosiomis įmonėmis		EUR	
## 5	Viešasis sektorius		EUR	

split

- Supjaustom df pagal Lytis ir Sektorius
- Inspektuojam gautą objektą

```
df_s <- split(df, list(df$Lytis, df$Sektorius))
```

split

Apskaičiuojame pajamų vidurkius

```
sapply(df_s, function(x) mean(x[, "Reikšmė"], na.rm=TRUE))
```

##	Moterys.Privatusis sektorius su individualiosiomis įmonėmis	572.7300
##	Vyrai.Privatusis sektorius su individualiosiomis įmonėmis	693.2500
##	Vyrai ir moterys.Privatusis sektorius su individualiosiomis įmonėmis	436.1522
##	Moterys.Šalies ūkis su individualiosiomis įmonėmis	613.1300
##	Vyrai.Šalies ūkis su individualiosiomis įmonėmis	720.8800
##	Vyrai ir moterys.Šalies ūkis su individualiosiomis įmonėmis	460.0391
##	Moterys.Viešasis sektorius	663.2300
##	Vyrai.Viešasis sektorius	801.8000
##	Vyrai ir moterys.Viešasis sektorius	

aggregate

aggregate

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

- `x` - an R object.
- `by` - a list of grouping elements, each as long as the variables in the data frame `x`. The elements are coerced to factors before use.

```
args(aggregate)
## function (x, ...)
## NULL
?aggregate
```

aggregate

```
aggregate(airquality,
          list(airquality$Month), #jeigu nebus list() mes klaidą
          FUN=mean,
          na.rm=TRUE)
```

##	Group.1	Ozone	Solar.R	Wind	Temp	Month	Day
## 1	5	23.61538	181.2963	11.622581	65.54839	5	16.0
## 2	6	29.44444	190.1667	10.266667	79.10000	6	15.5
## 3	7	59.11538	216.4839	8.941935	83.90323	7	16.0
## 4	8	59.96154	171.8571	8.793548	83.96774	8	16.0
## 5	9	31.44828	167.4333	10.180000	76.90000	9	15.5

tik vienam kintamajam

```
aggregate(airquality$Ozone, list(airquality$Month), FUN=mean, na.rm=TRUE)
```

##	Group.1	x
## 1	5	23.61538
## 2	6	29.44444
## 3	7	59.11538
## 4	8	59.96154
## 5	9	31.44828

aggregate

Exercise:

- pritaikykite aggregate norint apskaičiuoti vidutines mėnesines pajamas Lietuvoje pagal lytį ir sektorius
- (tikėtina pakeisiu dataset, nes šiaip šis skaičiavimas nėra labai prasmingas)

```
##                               Lytis                               Sektorius
## 1      Moterys Privatusis sektorius su individualiosiomis įmonėmis
## 2      Vyrai Privatusis sektorius su individualiosiomis įmonėmis
## 3 Vyrai ir moterys Privatusis sektorius su individualiosiomis įmonėmis
## 4      Moterys      Šalies ūkis su individualiosiomis įmonėmis
## 5      Vyrai      Šalies ūkis su individualiosiomis įmonėmis
## 6 Vyrai ir moterys      Šalies ūkis su individualiosiomis įmonėmis
## 7      Moterys                               Viešasis sektorius
## 8      Vyrai                               Viešasis sektorius
## 9 Vyrai ir moterys                               Viešasis sektorius
##      x
## 1 572.7300
```

Skirstiniai

Distribucijos (lt. = skirstiniai)

Ne retai atliekant įvairius tyrimus ar skaičiuojant tikimybes statistikoje, reikės remtis tam tikrais skirstiniais.

?distributions

- dnorm
- dgamma
- beta
- dpois

ir t.t.

Distribucijos

Šioje dalyje aptarsime:

- Binomial Distribution
- Poisson Distribution
- Continuous Uniform Distribution
- Exponential Distribution
- Normal Distribution

Distribucijos

Visos distribucijos galimos su 4 funkcijomis:

```
# ?dnorm
```

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

- d density
- p cumulative distribution
- q quantile function
- r random number generation

Binominis skirstinys

Dichotomine matavimų skale matuojamų požymių reikšmių skirstinys (0/1, herbas/skaičius, moteris/vyras, išlaikė/neišlaikė)

Skirstinys yra diskretus ir apibūdinamas parametrais n ir p . Parametras $n \geq 0$ reiškia bandymų skaičių, o p – požymio tikimybę įgyti vieną iš dviejų galimų reikšmių.

Binominio skirstinio pasiskirstymo tankio funkcija (tikimybė gauti x reikmę su n bandymų ir p tikimybės reikmše):

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} \text{ kur } x = 1, 2, 3, \dots, n$$

Binominis skirstinys

Tarkime duomenų analizės teste yra 10 klausimų, kurių kiekvienas turi 4 galimus atsakymus, iš kurių tik vienas yra teisingas. Tarkime studentas atėjo visiškai nepasiruošęs ir visiškai atsitiktinai pasirinko atsakymus. Norint išlaikyti testą, reikia teisingai ataktyti j ne mažiau kaip 5 klausimus. Kokia tikimybė, jog studentas neišlaikys testo?

- $p = 1/4 = 0.25$ ir $(1-p) = 1 - 0.25 = 0.75$
- $n = 10$
- $x = 4$

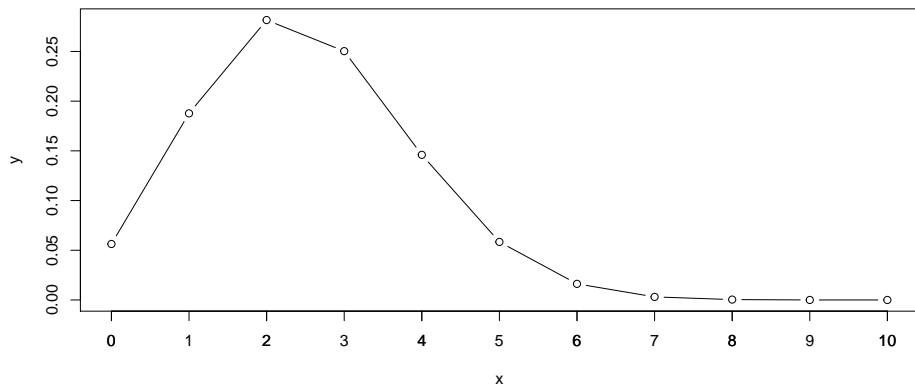
Diskreti tikimybė:

```
# tikimybė jog studentas atsakys lygiai 4 teisingai  
dbinom(x=4, size = 10, prob = 0.25)  
## [1] 0.145998
```

Binominis skirstinys

Skirstinio funkcija

```
x <- seq(from=0, to=10, by=1)
y <- dbinom(x, size=10, prob=0.25)
plot(x,y, type = "b")
axis(side = 1, at = x, labels = T)
```



Binominis skirstinys

Tačiau norint žinoti visas vertes iki 4

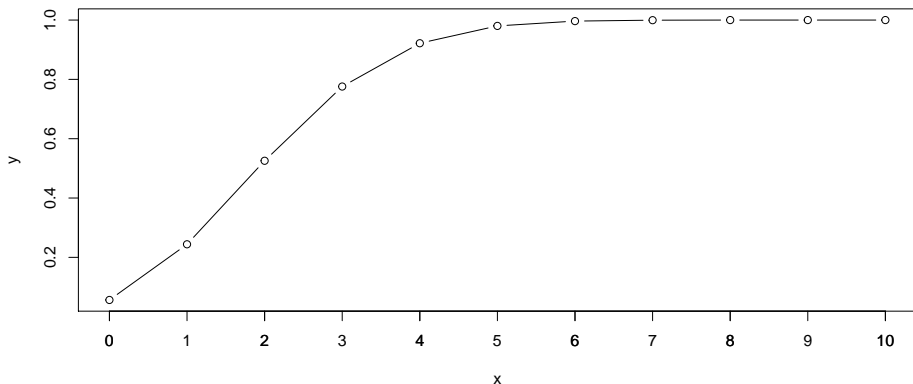
```
# todėl norint žinoti tikimybę jog studentas atsakys į 4 arba mažiau
dbinom(x=0, size = 10, prob = 0.25)+
  dbinom(x=1, size = 10, prob = 0.25)+
  dbinom(x=2, size = 10, prob = 0.25)+
  dbinom(x=3, size = 10, prob = 0.25)+
  dbinom(x=4, size = 10, prob = 0.25)
## [1] 0.9218731
# sum(mapply(dbinom, 0:4, 10,.25))

# alternatyviai galima pasinaudoti pbinom()
pbinom(q=4, size = 10, prob = 0.25, lower.tail = TRUE)
## [1] 0.9218731
# tačiau piktąjį dėstytoją domina,
# kokia tikimybė, jog studentas "praslys":
pbinom(q=4, size= 10, prob = 0.25, lower.tail = FALSE)
## [1] 0.07812691
```

Binominis skirstinys

Binominio skirstinio *tankio funkcija*

```
x <- seq(from=0, to=10, by=1)
y <- pbinom(x, size=10, prob=0.25)
plot(x,y, type = "b")
axis(side = 1, at = x, labels = T)
```



Binominis skirstinys

Tarkime dėstytojas nori nustatyti ribą, į kiek klausimų turi teisingai atsakyti studentai, kai:

- studentai turėdami 4 galimus pasirinkimus (daugiau alternatyvių atsakymų dėstytojas nenori sugalvoti, nes tingi)
- dėstytojas nenori, kad studentai praslystų pro testą didesne nei 1% tikimybe
- dėstytojas tingi galvoti daugiau nei 10 klausimų

```
qbinom(0.01, 10, 0.25, lower.tail = FALSE)
## [1] 6
pbinom(5,10,.25,lower.tail = FALSE) # galima pasitikrinti tikimybes
## [1] 0.01972771
pbinom(6,10,.25,lower.tail = FALSE) # galima pasitikrinti tikimybes
## [1] 0.003505707
```

Poisson skirstinys

Diskretus skirstinys (tikimybių pasiskirstymo dėsnis), nusakantis įvykių tikimybes įvykti per tam tikrą laiko intervalą, jeigu įvykiai vyksta pastoviu dažniu ir yra nepriklausomi vienas nuo kito.

- Vidurkis= $\bar{k} = \lambda$
- Dispersija= $\sigma^2 = \lambda$

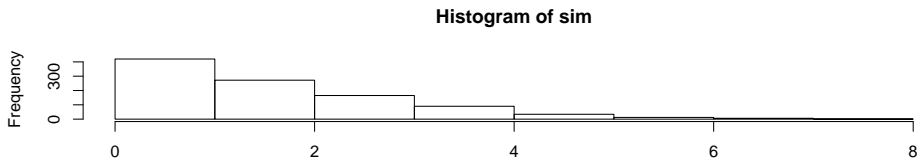
Jei per tam tikrą laiko intervalą įvyksta vidutiniškai λ įvykių, tuomet tikimybė, kad per tą laiką įvyks tiksliai x įvykių bus lygi:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \text{ kur } x = 1, 2, 3, \dots, n$$

Poisson skirstinys

Poisson distribucija, kur λ yra vidutinė įvykio tikimybė per tam tikrą laikotarpį

```
rpois(n=10, lambda = 1)
## [1] 0 1 1 1 1 2 1 1 1 1
rpois(n=10, lambda=2)
## [1] 3 3 2 6 3 3 0 1 4 4
sim <- rpois(n=1000, lambda=2)
mean(sim)
## [1] 1.976
sd(sim)^2 # dispersija=sigma^2=sd^2
## [1] 2.035459
hist(sim)
```



Poisson skirstinys

Skambučių centras per valandą sulaukia 50 skambučių. *Maximum capacity* yra 65 skambučiai per valandą. Tada skambučiai nukreipiami į alternatyvų skambučių centrą, kuriame dirba beždžionėlės, tad klientai visad lieka nepatenkinti. Klausimas, kokia yra tikimybė, jog per sekančią valandą skambučių centras sulauks: 5, 30, 60 (arba mažiau skambučių):

```
dpois(5, 50) ## 5 Pr(x=5), lambda=50
## [1] 5.022786e-16
dpois(30, 50) ## 30 Pr(x=30), lambda=50
## [1] 0.0006771985
dpois(60, 50) ## 60 Pr(x=50), lambda=50
## [1] 0.02010487
```

```
ppois(5, 50) ## 5 arba mažiau skambučių Pr(x<=5), lambda=50
## [1] 5.567756e-16
ppois(30, 50) ## 30 arba mažiau skambučių Pr(x<=30), lambda=50
## [1] 0.001594027
ppois(60, 50) ## 60 arba mažiau skambučių Pr(x<=50), lambda=50
## [1] 0.9278398
```


Poisson skirstinys

Exercise:

Kokia tikimybė, jog skambučių centras sulauks daugiau skambučių nei skambučių centro maksimalus aptarnavimo limitas?

Jeigu įmonės išsikeltas tikslas, jog nepatenkintų klientų būtų mažiau nei 0.1%, ar patartumėte vadovybei plėsti skambučių centro galimybes? Kiek papildomų darbuotojų reikia nusamdyti skambučiui centrui, jeigu 1 darbuotojas gali priimti po 5 skambučius per valandą?

Poisson skirstinys

Solution:

```
ppois(q=65, lambda = 50, lower.tail = FALSE)
## [1] 0.01726457

# Kiek papildomų skambučių reikėtų papildomai galėti priimti?
qpois(p=0.001, lambda = 50, lower.tail = FALSE)
## [1] 73

ceiling(
  (qpois(p=0.001, lambda = 50, lower.tail = FALSE) - 65) / 5
)
## [1] 2
```

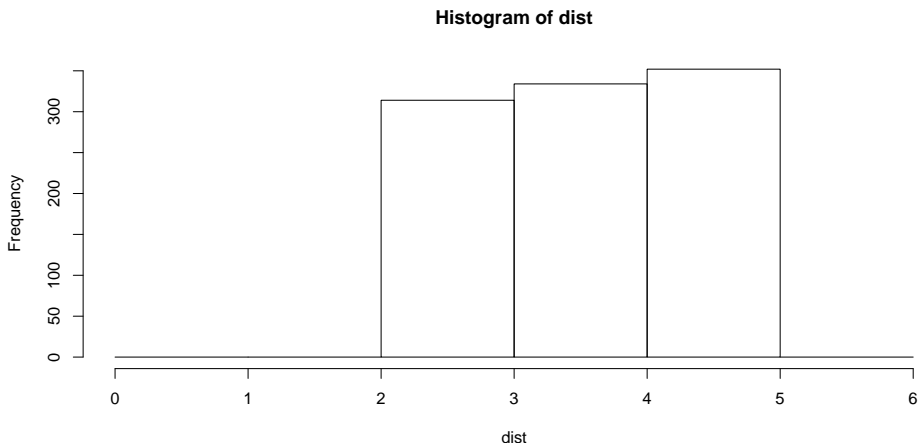
Tolygusis skirstinys (Continuous uniform distribution)

Skirstinys su vienoda tikimybe visiems skaičiams tarp a ir b . Visais kitais atvejais tikimybė $=0$.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{when } a \leq x \leq b \\ 0, & \text{else} \end{cases}$$

Tolygusis skirstinys (Continuous uniform distribution)

```
dist <- runif(n=1000, min=2, max=5)
hist(dist, breaks = seq(from=0, to=6, by=1))
```



Ekspontininis skirstinys

is the probability distribution that describes the time between events in a Poisson point process, i.e., a process in which events occur continuously and independently at a constant average rate

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x}, & \text{when } x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$f(x, \mu) = \begin{cases} \frac{1}{\mu} e^{-x/\mu}, & \text{when } x \geq 0 \\ 0, & x < 0 \end{cases}$$

Eksponentinis skirstinys

?dexp

EkspONENTINIS skirstinys

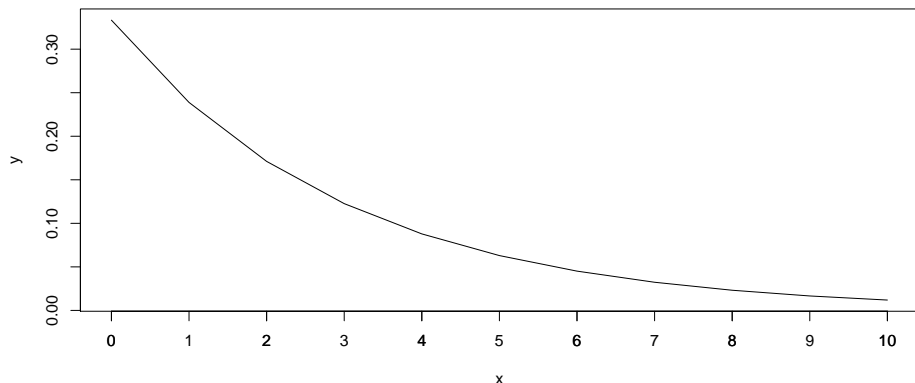
PVZ: Tarkime kasininkas aptarnauja vieną klientą per vidutiniškai 3 minutes. Žinoma, kad aptarnavimo laikas turi eksponentinį skirstinį. Kokia tikimybė sekantis klientas bus aptarnautas per mažiau nei 2 minutes

- vidutinis aptarnavimo greitis: $1/3=0.333$ klientų per minutę

```
pexp(2, rate=1/3)
## [1] 0.4865829
```

Eksponentinis skirstinys

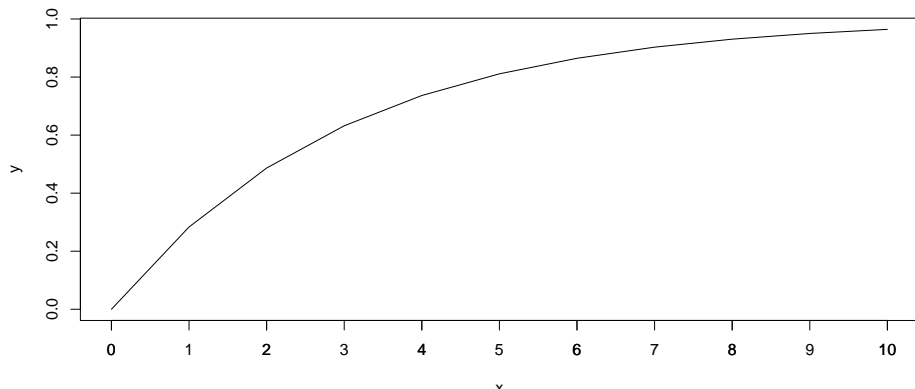
```
x <- seq(from=0, to=10, by=1)
y <- dexp(x, rate=1/3 )
plot(x,y, type = "l")
axis(side = 1, at = x,labels = T)
```



Eksponentinis skirstinys

Tankio funkcija:

```
x <- seq(from=0, to=10, by=1)
y <- pexp(x, rate=1/3 )
plot(x,y, type = "l")
axis(side = 1, at = x, labels = T)
```



Normalusis skirstinys

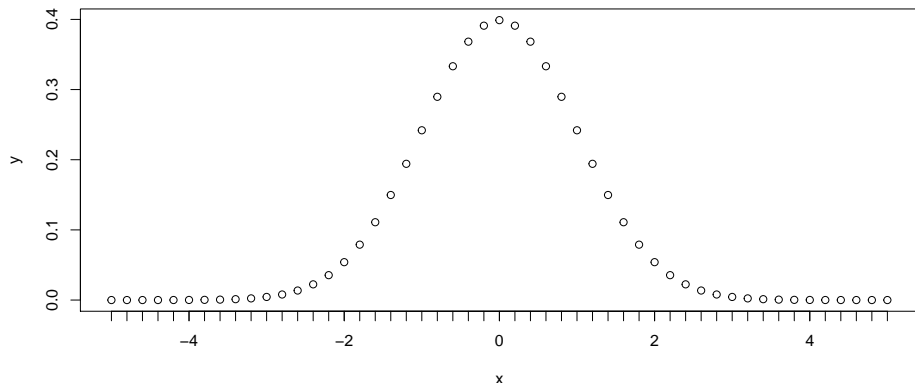
Sakysime, kad atsitiktinis dydis x turi normalųjį skirstinį, jei jo tankis

$$\varphi_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)} \text{ for } -\infty < x < \infty; -\infty < \mu < \infty, \sigma^2 > 0$$

Sakysime, kad atsitiktinis dydis x turi standartinį normalųjį skirstinį, jeigu $\mu = 0, \sigma^2 = 1$

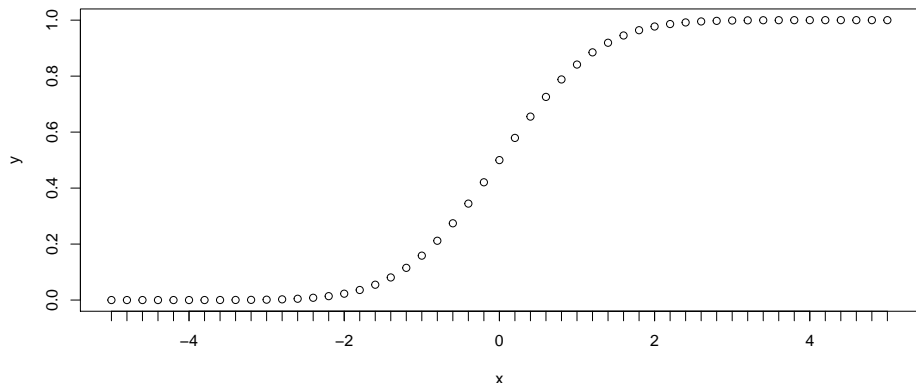
Normalusis skirstinys

```
x <- seq(from=-5, to=5, by=0.2)
y <- dnorm(x)
plot(x,y, type = "p")
axis(side = 1, at = x, labels = F)
```



Normalusis skirstinys

```
x <- seq(from=-5, to=5, by=0.2)
y <- pnorm(x)
plot(x,y, type = "p")
axis(side = 1, at = x, labels = F)
```



Normalusis skirstinys

- Tarkime yra žinoma, jog ūgis turi normalųjį skirstinį
- Kurso vidurkis: 1.70, standartinis nuokrypis: 10 cm
- Klausimai:
 - kokia tikimybė, jog auditorijoje bus 1.5m arba mažesnis asmuo
 - kokia tikimybė, jog auditorijoje bus 1.85m arba didesnis asmuo
 - iki kokio ūgio bus 95% visų studentų

Normalusis skirstinys

Solutions:

```
pnorm(q=1.5,mean=1.7,sd=0.1, lower.tail = TRUE)
## [1] 0.02275013
pnorm(q=1.85, mean=1.7,sd=0.1, lower.tail = FALSE)
## [1] 0.0668072
qnorm(p=0.95,mean=1.7, sd=0.1, lower.tail = TRUE)
## [1] 1.864485
```