Practice R

Justas Mundeikis
Data-Science.lt

2020-07-31

Contents

1 Introduction **4**

2 Installing R on Linux **4**

3 Data structures **4**
3.1 Vectors . . . 4
3.2 Vector with a sequence of numbers . . . 5
3.3 Repetition . . . 6
3.4 Vector with a sequance of letters . . . 7
3.5 Vectoriced operations . . . 8
3.6 Vectors to formulas . . . 10
3.7 Vector functions . . . 15
3.8 Rounding vector elements . . . 17
3.9 Cumulative Sums, Products, and Extremes . . . 18
3.10 Sorting or Ordering Vectors . . . 19
3.11 Vector logical testing . . . 22
3.12 Vector coersion . . . 24
3.13 Subsetting . . . 26
3.14 Logical vectors . . . 27
3.15 NAs and NANs . . . 28

4 Regular sequences **29**
4.1 Logical vectors and operators . . . 31
4.2 Missing values . . . 36
4.3 Character vectors . . . 39
4.4 Index vectors / subsetting . . . 41

5 Modes and attributes **42**

6 Factors **42**

7 Arrays and matrices **42**

8 List and Dataframes **42**

9 Data structures **42**

10 Character strings **42**

11 Tables **42**

12 Data manipulation **42**

13 Dates and times **42**
13.1 `as.Date()` . . . 42
13.2 Package "lubridate" . . . 42
13.3 Package "zoo()" . . . 42

14 Controll structures **42**
14.1 if . . . 43
14.2 for . . . 45
14.3 while, repeat loops, break and next . . . 47
14.4 Exercises . . . 50

15 Functions **51**

16 Graphical procedures **51**

17 Reading data from files **51**

18 Probability distributions **51**

19 Statistical models **51**

1 Introduction

2 Installing R on Linux

3 Data structures

3.1 Vectors

A vector is the most basic data structure in R. It contains only elements of the same type: * character elements (“name”, “grade” “a”,“b”, “c”) * integer elements (1, 2, 3L) * numerical elements (1.1, -0.1,9.999) * booleen elements (TRUE, FALSE) * complex (1+i) * raw ()

Vectors are usually created using the function `c()` [=`concatenate`]. Vectors can be assigned in multiple ways to an object, either by using `<-` operator, by using `=` operator (not suggested to use this), or by using the `assign()` function.

The type of vector can allways be checked using the function: `typeof()`. The number of elements of a vector can be checked using the function `length()`

A vector consisting of multiple vectors containing the same type of elements can also be created using `c()`.

```r
# Example
## creating vectors
x <- c(1.1,2.2,3.3,4.4,5.5)
y = c("a", "b", "c", "d")
assign("z", c(1,2,3,4,5))

## checking the type of vectors
typeof(x)
## [1] "double"
typeof(y)
## [1] "character"
typeof(z)
## [1] "double"

## checking the number of elements in the vectors
length(x)
## [1] 5
length(y)
## [1] 4
length(z)
## [1] 5

a <- c(1,2,3,4,5)
b <- c(6,7,8,9,10)
c <- c(a,b) #a and b denote object, thus no ".."
print(c)
##  [1]  1  2  3  4  5  6  7  8  9 10
```

3.2 Vector with a sequence of numbers

There are few different ways how to create a vector with a number sequence. The easiest way is to create an integer vector using :

```
# integer sequence from 1 to 10
1:10
##  [1]  1  2  3  4  5  6  7  8  9 10
# is the same as
c(1,2,3,4,5,6,7,8,9,10)
##  [1]  1  2  3  4  5  6  7  8  9 10
```

More advanced way is tu use the function `seq()`. Run the command in the console `?seq` to see the function manual page. Here are some examples of possible `seq()` usage:

```
# read the manual page
?seq

#examples
seq(from=1, to=5)
## [1] 1 2 3 4 5
seq(from=0, to=5, by=0.25 )
##  [1] 0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00 3.25 3.50
## [16] 3.75 4.00 4.25 4.50 4.75 5.00
seq(from=0, to=1, length.out=6)
## [1] 0.0 0.2 0.4 0.6 0.8 1.0
x <- c("a", "b", "c")
seq(along.with=x )
## [1] 1 2 3
seq(from=9) # the same as 1:10, or seq(from=1, to=10)
## [1] 1 2 3 4 5 6 7 8 9
seq(length.out=5.5)
## [1] 1 2 3 4 5 6
```

3.3 Repetition

```
?rep
```

```
x <- 1:5
y <- c("a", "b", "c")
rep(x,times=2)
##  [1] 1 2 3 4 5 1 2 3 4 5
rep(x, each=2)
##  [1] 1 1 2 2 3 3 4 4 5 5
rep(y, length.out=5)
## [1] "a" "b" "c" "a" "b"
```

3.3.1 Questions

1. Create / print a single number vector with the numerical element of `99`
2. Create a numerical vector with even number up to 10.
3. Create a numerical vector with numbers / integers 1 to 50 Explain what do the numbers in square brackets at the beginning of every reported line mean?
4. Concatenate three vectors: a containing integers from 1 to 10, b containing integers from 11 to 20, c containing integers from 21 to 30 by using the function `seq` or `x:x` to a new vector `d`

3.3.2 Solutions

```
# 1.
99
## [1] 99

# 2
c(2,4,6,8,10)
## [1]  2  4  6  8 10

# 3
# first alternative
seq(from=1, to=50)
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
# second alternative
1:50
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
# third alternative
# c(1,2,3,4,...,100) manual entry by hand

# the number in the brackets at the beginning of each line implicates
# the number of the element in the vector.

# 4
a <- seq(from=1, to=10, by=1)
b <- seq(from=11, to=20, by=1)
c <- seq(from=21, to=30, by=1)
d <- c(a,b,c)
print(d)
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30
```

3.4 Vector with a sequance of letters

```
letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
LETTERS
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

3.5 Vectoriced operations

In R all operations are executed in vectorized form, for example addint two vectors `c(1,2,3)` and `c(4,5,6)` will result in a vector, where each element willl be added elementwise in the vector resutling in `c(5,7,9)`

Basic arithmetic operators are: * `+` addition * `-` subtraction * `*` multiplication * `/` division * `^` or `**` exponentiation * `x %% y` modulus (x mod y) is the integer remainder. usefel for cheking if a number is odd * `x %/% y` integer division

```r
a <- c(1,2,3)
b <- c(3,4,5)

a+b
## [1] 4 6 8
a-b
## [1] -2 -2 -2
a*b
## [1]  3  8 15
a/b
## [1] 0.3333333 0.5000000 0.6000000
a^b
## [1]   1  16 243
3 %% 2 # 3 pizza pieces divided by 2 persons, 1 piece remains
## [1] 1
17 %% 2 # if remainder =1, then the number is odd, else even
## [1] 1
b %% a
## [1] 0 0 2
10 %/% 3
## [1] 3
b %/% a
## [1] 3 2 1
```

3.5.1 Exercises

1. Create two vectors a with a sequence of 1 to 5 and b with a sequence of 5 to 10 try all mathematical operators with tese two vectors.
2. Create a new object c by assigning the following arithmetical operation to it using vectors a and b from previous excersise: add to each element of vector `b` 5, then multiple each element by 3, then and divide the reuslting vector by the vector a.
3. Substituting, prove that the square of c which is equals to the sum of vectors a and b is the same as squaring the sum of vectors a and b.

3.5.2 Solutions:

```r
a <- seq(from=1, to=5, by=1)
b <- seq(from=6, to=10, by=1)

a+b
## [1]  7  9 11 13 15
a-b
## [1] -5 -5 -5 -5 -5
a*b
## [1]  6 14 24 36 50
a/b
## [1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000
a^b
## [1]       1     128    6561  262144 9765625
```

```r
b %% a
## [1] 0 1 2 1 0
b %/% a
## [1] 6 3 2 2 2


c <- ((b+5)*3)/a
print(c)
## [1] 33.0 18.0 13.0 10.5  9.0

c <- a+b
c^2
## [1]  49  81 121 169 225
(a+b)^2
## [1]  49  81 121 169 225
# proof
c^2==(a+b)^2
## [1] TRUE TRUE TRUE TRUE TRUE
```

3.6 Vectors to formulas

Mathematical predefined values / formulas:

- `pi`
- `exp()`
- `sqrt()`
- `abs()`
- `log()` see `?log`

It is quite simple using simple assign operations and mathematical operations to translate mathematical formulas.

The radius of a circle is given by a formula

$$Radius = \sqrt{\frac{Area}{\pi}}$$

Lets assume the area is given and equals 30. R knows what π is.

```
# cheking the value for pi
pi
## [1] 3.141593
# defining area
a <- 30
# creating formula for radius
r <- sqrt(a/pi)
# prining the radius resula
print(r)
## [1] 3.090194
```

3.6.1 Exercises:

1. The normally distributed density function has the equation

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

.

Given the mean ($\mu = 0$) and the standard deviation ($\sigma = 1$) calculate the probability of $x = 1.96$

2. Now instead of using single value for `x`, use a vector with a sequence from -5 to 5 by 0.1 steps. Instead of printing the result `print()` use the function `plot(x=x, y=pdf)`.

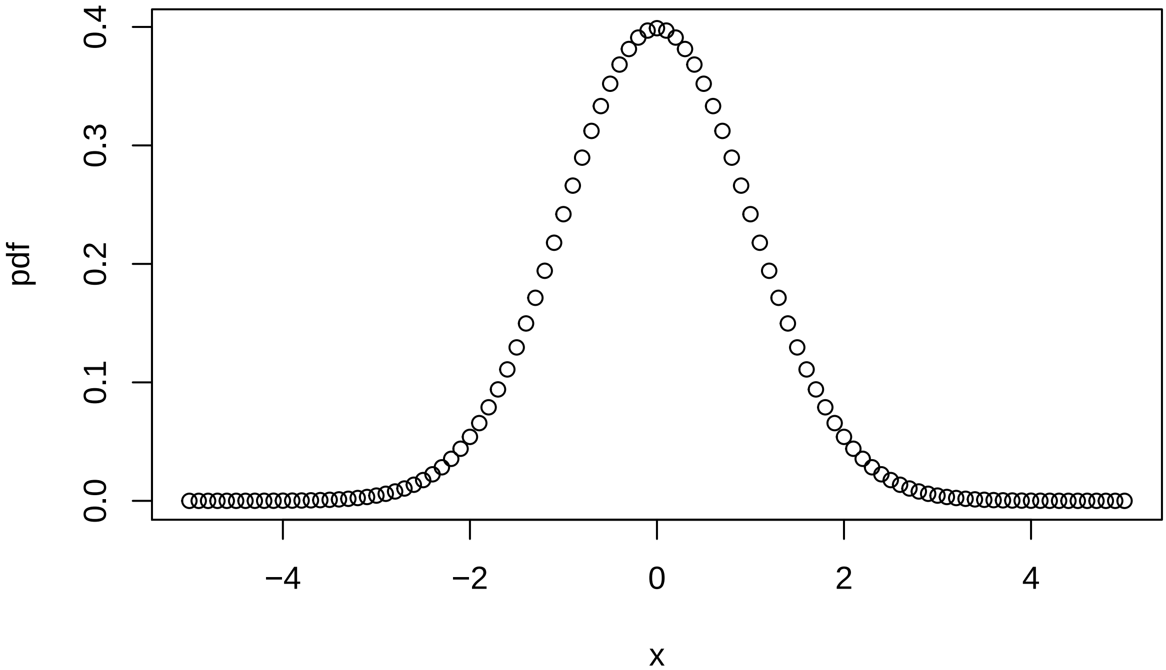
3. Use the modulo operator `%%` to find out for which of the following pairs, the first number is a multiple of the second

- 24025, 115
- 61988, 98
- 2555, 245
- 68719476736, 8

```
# defining  variables
mu <- 0
sigma <- 1
x <- 1.96
#defining formula
pdf <- 1/(sigma*sqrt(2*pi))*exp(-1/2*((x-mu)/sigma)^2)
# print the pdf
print(pdf)
## [1] 0.05844094
```

```
# defining  variables
mu <- 0
sigma <- 1
x <- seq(from=-5, to=5, by=0.1)
```

```r
#defining formula
pdf <- 1/(sigma*sqrt(2*pi))*exp(-1/2*((x-mu)/sigma)^2)
# print the pdf
plot(x=x, y=pdf)
```




```r
24025  %% 115
## [1] 105
61988 %% 98
## [1] 52
2555 %% 245
## [1] 105
68719476736 %% 8
## [1] 0
```

3. You think of taking a mortgage at a bank. The mortgage size (principle - P) equals 100000. Currently the annual interest rate of Euribor equals 6 % (monthly interest rate =0.06/12=0.005). You would like to repay the mortgage in 15 years (15*12=180 months). You would like to know what monthly payments of size M you have to pay. The formula for M is given as

$$M = P\frac{r(1+r)^n}{(1+r)^n - 1}$$

Lets assume you cannot afford to pay the monthly mortgage payment calculcated in first part. So you're interested in calculating diffrent M for different number of periods. Assume you want to calcultae M for periods between 10 and 25 years. Plot the corresping M values using `plot(x=n, y=M)`

Assume you can pay 300 euro monthly. How many monthly payments have you to pay? Assume you consider different amounts of payments of 100,200,300,400,500,600,700,800,900 euro. Plot the number of months using `plot(x=M, y=n)`

Lets assume you can only repay 650 euro. How many months have you to repay your mortgage? Note , that n can be calculated as

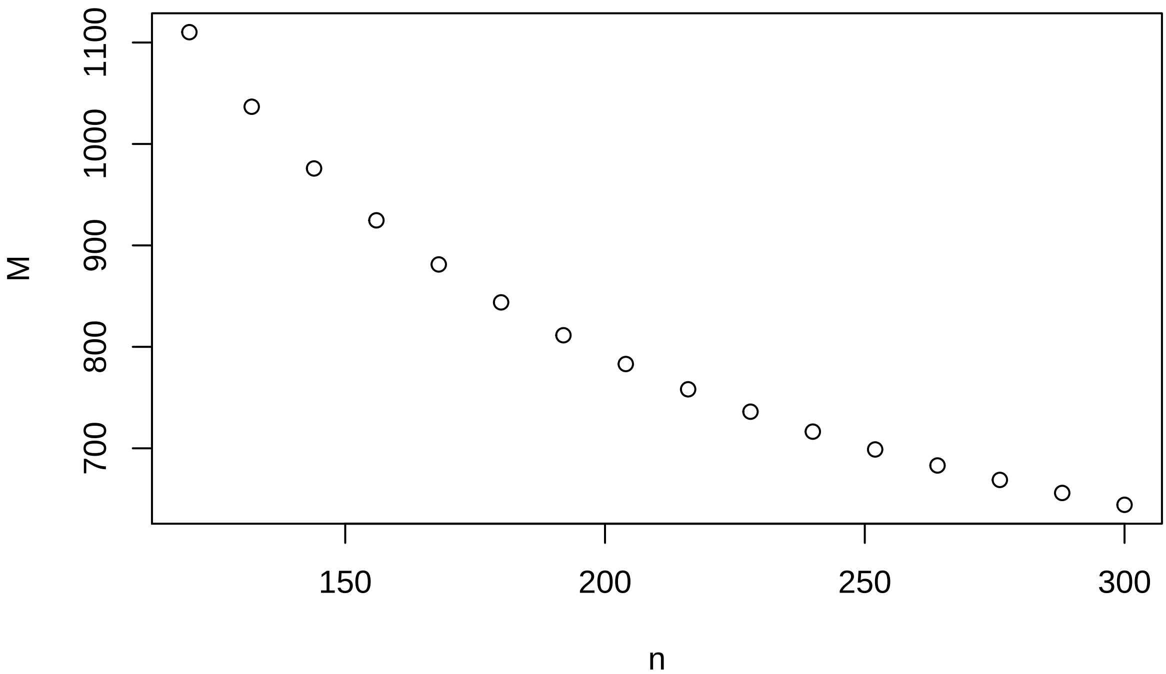
$$n = \frac{\ln\left(\frac{i}{\frac{M}{P}-i}+1\right)}{\ln(1+i)}$$

Lets assume you would can consider repaying between 100 and 1000 euro monthly (in 10 euro steps). In how many months would you have repayed your mortgage? Plot the number of months using `plot(x=M, y=n)`. Explain what happens, if you choose to repay less then 500 euro?

3.6.2 Solutions:

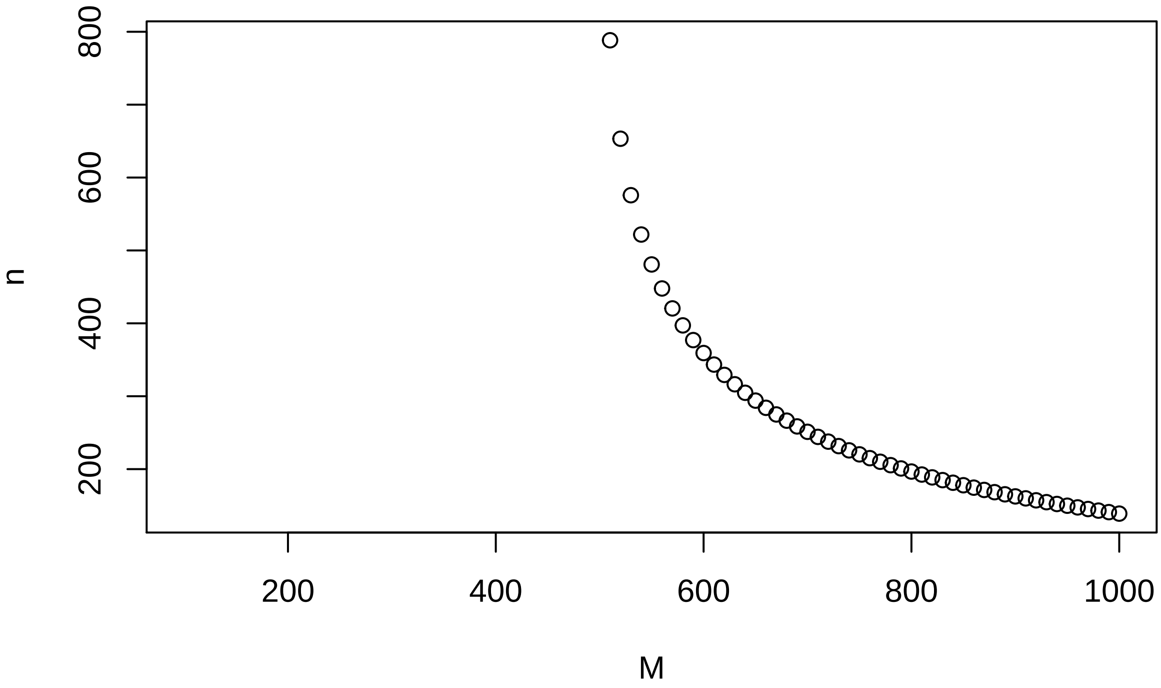
```r
P <- 100000
r <- 0.06/12
n <- 15*12
M <- P*(r*(1+r)^n)/((1+r)^n-1)
print(M)
## [1] 843.8568
```

```r
P <- 100000
r <- 0.06/12
n <- seq(from=10, to=25)*12
M <- P*(r*(1+r)^n)/((1+r)^n-1)
plot(x=n, y=M)
```




```r
P <- 100000
r <- 0.06/12
M <- 650
n <- log(r/(M/P-r)+1)/log(1+r)
years <- n %/% 12
months <- n %% 12
print(years)
## [1] 24
print(months)
## [1] 5.999973
```

```r
P <- 100000
r <- 0.06/12
M <- seq(from=100, to=1000, by=10)
n <- log(r/(M/P-r)+1)/log(1+r)
## Warning in log(r/(M/P - r) + 1): NaNs produced
plot(x=M, y=n)
```



3.6.3 Exercise

As a starting data scientist working at Lithuanian Airlines you are tasked to calculate the price for flights between Vilnius and Kaunas, Vilnius and Klaipeda and Vilnius Berlyn. Your boss gave you the following data:

- Vilnius cooridinates: Lat: 54.6870458, Long: 25.2829111
- Klaipeda Lat: 55.7127529, Long:21.1350469
- Berlin: Lat:52.5170365, Long:13.3888599

After googling for "The Great-Circle distance" you find out, that to calculate the distance, you set ϕ_1,λ_1 and ϕ_2, λ_2 to be the geographical latitude and longitude in radians of two points 1 and 2.

The geographical latitude and longitude in radians is calculated as $radians = degrees \times \frac{\pi}{180}$

Then you have to calculate central angle between them using the formula:

$$\Delta\sigma = \arccos\big(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\Delta\lambda)\big)$$

where $\Delta\lambda$ is tha absolute difference between two longitudes (use `abs()` function). Once the $\Delta\sigma$ is calcultaed, the distance can be calculated as

$$d = r\Delta\sigma$$

with r as earth radius $r = 6371$.

The total cost of each flight is set to be: * fixed cost (airport fee): 50 euro * variable cost: for a selected distance distance^(-0.5) per km of distance

So the break even price for the flight is calculated as

$$BePrice = \text{fixed cost} + distance \times \text{variable cost}$$

```
# Calculations for Vilnius --> Klaipeda
phi_1 <- 54.6870458
lambda_1 <- 25.2829111
phi_2 <- 52.5170365
lambda_2 <- 21.1350469

phi_1 <- phi_1*pi/180
phi_2 <- phi_2*pi/180
lambda_1 <- lambda_1*pi/180
lambda_2 <- lambda_2*pi/180
```

```r
delta_sigma <- acos(sin(phi_1)*sin(phi_2)+cos(phi_1)*cos(phi_2)*cos(abs(lambda_1-lambda_2)))
distance <- delta_sigma*6371

distance
## [1] 364.7552

price_vln_klp <- 50+distance*distance^(-0.5)
print(price_vln_klp)
## [1] 69.09856


# Calculations for Vilnius --> Klaipeda
phi_1 <- 54.6870458
lambda_1 <- 25.2829111
phi_2 <- 55.7127529
lambda_2 <- 13.3888599

phi_1 <- phi_1*pi/180
phi_2 <- phi_2*pi/180
lambda_1 <- lambda_1*pi/180
lambda_2 <- lambda_2*pi/180

delta_sigma <- acos(sin(phi_1)*sin(phi_2)+cos(phi_1)*cos(phi_2)*cos(abs(lambda_1-lambda_2)))
distance <- delta_sigma*6371
distance
## [1] 762.3957

price_vln_bln <- 50+distance*distance^(-0.5)
print(price_vln_bln)
## [1] 77.61151
```

3.7 Vector functions

Usually when applying a function to a vector, the function is applied element by element to the elements of the input vector and returns a result vector with the same length as the input vector.

But some functions are functions that summaryse data. Most common ones are:

min() and max() `mean()`, `median()`, `sd()` and `var()` * `length()`

Many functions take additional arguments, that allow editing the result R retrieves. Check for example `?mean`. Function `mean()` has additional argument `trim` that allows trimming the input vector from both ends. Also the function has an argument `na.rm`, if set to TRUE, all `NA` in the vector will be ommited when appplying the function.

```
# vector
x <- c(-100,1,2,3,4,5,6,7,8,9,1000)

mean(x)
## [1] 85.90909
#triming 10% of elements at the bottom and top of the vector
mean(x, trim = 0.1)
## [1] 5

#vector x with NA
x <- c(-100,1,2,3,NA,5,6,7,8,9,1000)

mean(x)
## [1] NA
# excluding NAs
mean(x, na.rm = TRUE)
## [1] 94.1
# triming 10% of elements at the bottom and top of the vector
mean(x, na.rm = TRUE, trim = 0.1)
## [1] 5.125
```

Some functions, such as `cor()` can take multiple input vectors, e.g. calculating the correlation between two vectors. Check `?cor` for additional arguments the function can take. `cor()` can take different methods: method = c("pearson", "kendall", "spearman")), which the user should select depending on the needs.

```
# loading dataset airquality into R workspace
data("airquality")

# calculating the correlation between Wind and Temp
cor(airquality$Wind, airquality$Temp)
## [1] -0.4579879

# pearson is the standard use
cor(airquality$Wind, airquality$Temp, method = "pearson")
## [1] -0.4579879
# kendall
cor(airquality$Wind, airquality$Temp, method = "kendall")
## [1] -0.3222418
# spearman
cor(airquality$Wind, airquality$Temp, method = "spearman")
## [1] -0.4465408
```

Sometimes you might need to check for `min` or `max` across multiple vectors, but the same positions within the vector, e.g. check if 3 element is greater in vector x or vector y. `max(x,y)` would retrieve one number, the maximum of both vectors x and y. To run `min` and `max` parallel between two or more vectors use `pmin()` and `pmax()`.

```
x <- c(1,9,2,11)
y <- c(2,4,6,8)
```

```
max(x,y)
## [1] 11
pmax(x,y)
## [1]  2  9  6 11
pmin(x,y)
## [1] 1 4 2 8
```

3.8 Rounding vector elements

There are 5 functions that help user round the elements of the vector or the results of summarizing function.

- round()
- floor()
- ceiling()
- truncate()
- signif()

Check their all descriptions using ?round

```
x <- c(-1,-0.99,-.751,-.5,0,0.0013,0.1,0.499,0.5,0.999)
x
##  [1] -1.0000 -0.9900 -0.7510 -0.5000  0.0000  0.0013  0.1000  0.4990  0.5000
## [10]  0.9990
# rounding without arguments
round(x)
##  [1] -1 -1 -1  0  0  0  0  0  0  1
round(x,digits = 1)
##  [1] -1.0 -1.0 -0.8 -0.5  0.0  0.0  0.1  0.5  0.5  1.0
round(x,digits = 2)
##  [1] -1.00 -0.99 -0.75 -0.50  0.00  0.00  0.10  0.50  0.50  1.00

floor(x)
##  [1] -1 -1 -1 -1  0  0  0  0  0  0
ceiling(x)
##  [1] -1  0  0  0  0  1  1  1  1  1
trunc(x)
##  [1] -1  0  0  0  0  0  0  0  0  0
signif(x,digits = 2)
##  [1] -1.0000 -0.9900 -0.7500 -0.5000  0.0000  0.0013  0.1000  0.5000  0.5000
## [10]  1.0000
```

3.9 Cumulative Sums, Products, and Extremes

4 functions return a vector whose elements are the cumulative sums, products, minima or maxima of the elements of the argument. Check the description under `?cumsum`

```
x <- c(1:3,0,10:12)

cumsum(x)
## [1]  1  3  6  6 16 27 39
cummin(x)
## [1] 1 1 1 0 0 0 0
cummax(x)
## [1]  1  2  3  3 10 11 12
cumprod(x)
## [1] 1 2 6 0 0 0 0
```

3.10 Sorting or Ordering Vectors

- `sort()` reorders a vector into ascending or descending order. See `?sort` for more information.
- `order()` order returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments
- `rank()` returns the sample ranks of the values in a vector. Ties (i.e., equal values) and missing values can be handled in several ways.

```
x <- c(1,3,5,0,2,7,2)
sort(x)
## [1] 0 1 2 2 3 5 7
sort(x, decreasing = TRUE)
## [1] 7 5 3 2 2 1 0

order(x)
## [1] 4 1 5 7 2 3 6
order(x, decreasing = TRUE)
## [1] 6 3 2 5 7 1 4

rank(x)
## [1] 2.0 5.0 6.0 1.0 3.5 7.0 3.5
```

3.10.1 Exercises

1. Load the dataset `AirPassengers` by using the command `data(AirPassengers)`, which contains monthly Airline Passenger Numbers 1949-1960. This dataset is a timeseries object, which will be covered in later lectures. Calculate:

- the mean number of passengers between 1949-1960
- the median number of passengers between 1949-1960
- the minimum number of passengers between 1949-1960
- the maximum number of passengers between 1949-1960
- the standard deviation between 1949-1960

Plot the cumulative sum of passengers between 1949-1960 using `plot(..., type="l)` by substituting ... with the formula needed.

2. What is the sum of highest 2 values in the vector: `c(1001,1000,99999,125,6898,12547,396845,15756)`?

3. What is the sum of lowest 2 values in the vector: `c(1001,1000,99999,125,6898,12547,396845,15756)`?

4. In Lithuania each parsons' income is deducted by a tax-free (NPD) amount, before being taxed with 20 percent income tax. The formula for tax-free (NPD) amount is

$$NPD = 400 - 0.19 * (\text{monthly income} - MW)$$

   where `MW` stands for the monthly minimum wage, which equals 607 euro. Calculate the after tax income (net_income) for the following set of income: `c(100,200,300,400,500,600,700,800,900,1000,1500,2000,2500,3000`

Some definitions:

- NPD is the tax free amount $NPD = 400 - 0.19 * (\text{monthly income} - MW)$
- taxable income is $income - NPD$
- labor taxes equal $(income - NPD) \times 0.2$
- net income $net_income = income - (income - NPD) \times 0.2$

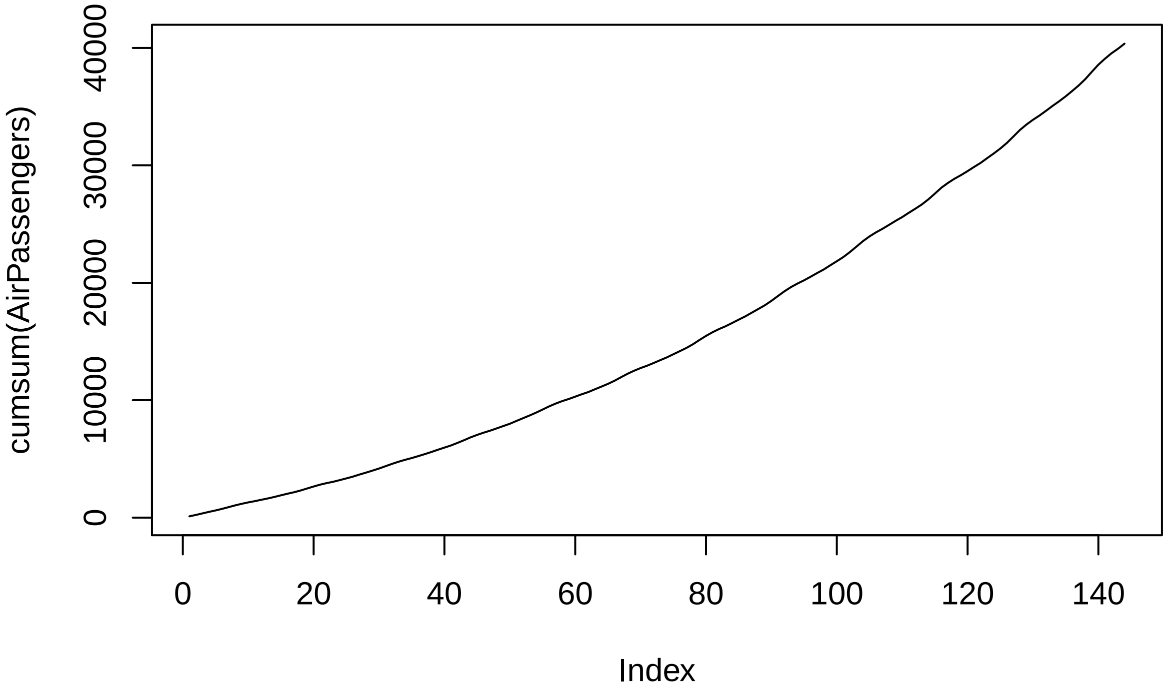
5. Calculate the tax wedge, which is defined as $taxation/income$

3.10.2 Solutions

```
mean(AirPassengers)
## [1] 280.2986
median(AirPassengers)
```

```
## [1] 265.5
min(AirPassengers)
## [1] 104
max(AirPassengers)
## [1] 622
sd(AirPassengers)
## [1] 119.9663

plot(cumsum(AirPassengers), type="l")
```




```
x <- c(1001,1000,99999,125,6898,12547,396845,15756)

# read the second entry of the vector
cumsum(sort(x, decreasing = TRUE))
## [1] 396845 496844 512600 525147 532045 533046 534046 534171

# read the second entry of the vector
cumsum(sort(x))
## [1]    125   1125   2126   9024  21571  37327 137326 534171



MW <- 607
income <- c(100,200,300,400,500,600,700,800,900,1000,1500,2000,2500,3000,3500,4000)
inc_tax <- 0.2


# after tax income is:
# income - taxable_income *0.2
# where as taxable income is
# income - tax-free income
# and tax free income is
# NPD=400-0.19*(income - MW)
```

```r
# lets start from the inside
income-MW
##  [1] -507 -407 -307 -207 -107   -7   93  193  293  393  893 1393 1893 2393 2893
## [16] 3393
# for income < MW, values turn negative, lets set them to 0
pmax(income-MW,0)
##  [1]    0    0    0    0    0    0   93  193  293  393  893 1393 1893 2393 2893
## [16] 3393
# calculating tax-free income, values above 2700 turn negative:
400-0.19*pmax(income-MW,0)
##  [1]  400.00  400.00  400.00  400.00  400.00  400.00  382.33  363.33  344.33
## [10]  325.33  230.33  135.33   40.33  -54.67 -149.67 -244.67
# lets set them also to 0
pmax(400-0.19*pmax(income-MW,0),0)
##  [1] 400.00 400.00 400.00 400.00 400.00 400.00 382.33 363.33 344.33 325.33
## [11] 230.33 135.33  40.33   0.00   0.00   0.00
# define this vector as NPD
NPD <- pmax(400-0.19*pmax(income-MW,0),0)
# calculate taxable income
income-NPD
##  [1] -300.00 -200.00 -100.00    0.00  100.00  200.00  317.67  436.67  555.67
## [10]  674.67 1269.67 1864.67 2459.67 3000.00 3500.00 4000.00
# for income less then NPD, the values turn again negative, lets set them to 0
pmax(income-NPD,0)
##  [1]    0.00    0.00    0.00    0.00  100.00  200.00  317.67  436.67  555.67
## [10]  674.67 1269.67 1864.67 2459.67 3000.00 3500.00 4000.00

net_income <- income - pmax(income-NPD,0)* inc_tax


tax_wedge <- pmax(income-NPD,0)* inc_tax /income
tax_wedge
##  [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.04000000 0.06666667
##  [7] 0.09076286 0.10916750 0.12348222 0.13493400 0.16928933 0.18646700
## [13] 0.19677360 0.20000000 0.20000000 0.20000000
```

3.11 Vector logical testing

Logical operators are :

- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to
- `==` exactly equal to
- `!=` not equal to
- `!x` Not x
- `x | y` x OR y
- `x && y`
- `x & y` x AND y
- `x || y`
- `xor(x, y)`
- `isTRUE(x)` test if X is TRUE
- `isFALSE(x)` test if X is FALSE

With R one can perform some logical tests on the elements of a vector (and on elements of other data structers as well)

```
x <- c(-9,-5,-3,0,1,5,7)
x<0
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
x<=0
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
x==0
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
x>=0
## [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
x>0
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
x!=0
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
!(x==0)
## [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

```
# && evaluates only one of the condition and only if it's TRUE will it
# evaluate the second condition. & on the other hand evaluates both expressions
# and compares them. | and || have the same difference as the above
# so if there is a condition as 3<4 || 7/"b" it wont flag an error as 3<4 is TRUE so
# 7/"b" is never evaluated. However 3<4 | 7/"b" will flag an error as you cannot
# divide by a character. && is generally used in control flow (ifelse) and & is used
# in vectorization
```

```
x <- c(TRUE, FALSE, TRUE, FALSE)
y <- c(TRUE, TRUE, FALSE, FALSE)
a <- c(T,T)
b <- c(F,F)

x&y
## [1]  TRUE FALSE FALSE FALSE
x&&y
## [1] TRUE
x|y
## [1]  TRUE  TRUE  TRUE FALSE
x||y
## [1] TRUE

isTRUE(5>0)
## [1] TRUE
isTRUE(-5>0)
```

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

```
x=letters[1:7]
print(x)
## [1] "a" "b" "c" "d" "e" "f" "g"
y=letters[11:17]
print(y)
## [1] "k" "l" "m" "n" "o" "p" "q"
x<y
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

3.12 Vector coersion

```r
a <- c(0L, 1L, 2L) # same as 0:2
b <- c(0.1, 0.2, 5)
c <- c("alfa", "beta", "gamma")
d <- c(TRUE, FALSE)

class(a)
## [1] "integer"
class(b)
## [1] "numeric"
class(c)
## [1] "character"
class(d)
## [1] "logical"

c(a,b)
## [1] 0.0 1.0 2.0 0.1 0.2 5.0
class(c(a,b))
## [1] "numeric"

c(a,c)
## [1] "0"     "1"     "2"     "alfa"  "beta"  "gamma"
class(c(a,c))
## [1] "character"

c(a,d)
## [1] 0 1 2 1 0
class(c(a,d))
## [1] "integer"

c(c,d)
## [1] "alfa"  "beta"  "gamma" "TRUE"  "FALSE"
class(c(c,d))
## [1] "character"
```

```r
x=c('blue','red','green','yellow')
is.character(x)
## [1] TRUE

x=c('blue',10,'green',20)
is.character(x)
## [1] TRUE
```

```r
a <- 1:5
b <- 1:3
c <- c("yes", "no")

a+b
## Warning in a + b: longer object length is not a multiple of shorter object
## length
## [1] 2 4 6 5 7
a*b
## Warning in a * b: longer object length is not a multiple of shorter object
## length
## [1]  1  4  9  4 10

cbind(a,b)
## Warning in cbind(a, b): number of rows of result is not a multiple of vector
## length (arg 2)
```

```
##      a b
## [1,] 1 1
## [2,] 2 2
## [3,] 3 3
## [4,] 4 1
## [5,] 5 2
cbind(a,c)
## Warning in cbind(a, c): number of rows of result is not a multiple of vector
## length (arg 2)
##      a   c
## [1,] "1" "yes"
## [2,] "2" "no"
## [3,] "3" "yes"
## [4,] "4" "no"
## [5,] "5" "yes"
rbind(a,b)
## Warning in rbind(a, b): number of columns of result is not a multiple of vector
## length (arg 2)
##   [,1] [,2] [,3] [,4] [,5]
## a    1    2    3    4    5
## b    1    2    3    1    2
rbind(a,c)
## Warning in rbind(a, c): number of columns of result is not a multiple of vector
## length (arg 2)
##   [,1]  [,2] [,3]  [,4] [,5]
## a "1"   "2"  "3"   "4"  "5"
## c "yes" "no" "yes" "no" "yes"
```

3.13 Subsetting

- `subset`
- `[]`
- `[[]]`
- `$`

```
x <- 1:10
y <- c("alpha", "beta", "gamma")
x[1]
## [1] 1
x[3:6]
## [1] 3 4 5 6
x[c(3,4,5,6)]
## [1] 3 4 5 6
x[c(1,5,9)]
## [1] 1 5 9
x[x<=5]
## [1] 1 2 3 4 5
x[x<2&x>8]
## integer(0)
x[x<2|x>8]
## [1]  1  9 10
y[c(1,3)]
## [1] "alpha" "gamma"
```

3.14 Logical vectors

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

as.numeric(x)
## [1] 1 1 0

sum(x)
## [1] 2

sum(!x)
## [1] 1
```

3.15 NAs and NANs

- `is.na()`
- `is.nan()`
- `is.finite()`
- `is.infite()`

```
x <- c(0,1,NA, 3, 4, NA)
is.na(x)
## [1] FALSE FALSE  TRUE FALSE FALSE  TRUE
sum(is.na(x))
## [1] 2
sum(!is.na(x))
## [1] 4


y <- c(0,1,2,3,NA,NaN, Inf, 1/0)
print(y)
## [1]   0   1   2   3  NA NaN Inf Inf

is.na(y)
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
is.nan(y)
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
is.finite(y)
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
is.infinite(y)
## [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

Function `which()` retrieves the TRUE indices of a logical vector.

```
x <- c(TRUE, FALSE, TRUE, FALSE, FALSE, TRUE)
which(x)
## [1] 1 3 6


x=c(12:4)
y=c(0,1,2,0,1,2,0,1,2)
which(!is.finite(x/y))
## [1] 1 4 7
```

4 Regular sequences

Exercises:

Using the `seq()` command generate following sequences:

- 1,3,5,7,9,11
- 0,9,18,27,36
- 10,5,0,-5,-10
- 100,95,90,85,80

Using the `seq()` command and `length.out` parameter generate following sequences:

- 0,5,10,15,20,25,30
- 9,18,27,36,45,54,63

Explain the difference in generating a regular sequences:

```
x <- 10
1:x-2
##  [1] -1  0  1  2  3  4  5  6  7  8
1:(x-2)
## [1] 1 2 3 4 5 6 7 8
```

Explain: what will be the result of following code and why?

```
x <- c(1,2,3,4)
y <- c(100,200,300,400)
seq(along.with=x)==seq(along.with=y)
```

Using the `rep()` function, create following regular sequences:

- 1 2 3 1 2 3 1 2 3
- 1 1 1 2 2 2 3 3 3
- 1 1 1 2 3 3 3

What does the `len` argument do? * `rep(1:3, each = 2, len = 3)` * `rep(1:3, each = 2, len=12)`

Solutions:

```
seq(from=1, to=11, by=2)
## [1]  1  3  5  7  9 11
seq(from=0, to=36, by=9)
## [1]  0  9 18 27 36
seq(from=10, to=-10, by=-5)
## [1]  10   5   0  -5 -10
seq(from=100, to=80, by=-5)
## [1] 100  95  90  85  80

seq(from=0, to=30, length.out = 7)
## [1]  0  5 10 15 20 25 30
seq(from=9, to=63, length.out = 7)
## [1]  9 18 27 36 45 54 63



rep(1:3, 3)
## [1] 1 2 3 1 2 3 1 2 3
rep(1:3, each = 3)
## [1] 1 1 1 2 2 2 3 3 3
rep(1:3, c(2,2,2))     # same as second.
## [1] 1 1 2 2 3 3

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

For the following exercises write down your answer pn a sheet of paper first, then compare it with R output

- if `x <- c(a=12, b=13, c=14, d=15)`, what is the output for the code `seq(1,10,along.with =x)`
- If `x <- seq(1,9,3)`, what is the output for the code `rep(x, each=2)`
- What is the output for the code: `seq(1,4,by=2,length.out=4)`
- What is the output for the code: `rep(letters[1:3],3)`
- What is the output for the code: `seq(from=100, to=0, by= 10)`?
- What is the output for the code: `rep(c("alpha","beta", "gamma"),each=3)`

Regular sequnces with dates The method for seq for objects of class class "Date" representing calendar dates. `seq(from, to, by, length.out = NULL, along.with = NULL, ...)`

```
## first days of years
seq(as.Date("2000-01-01"), as.Date("2020-01-01"), "1 year")
##  [1] "2000-01-01" "2001-01-01" "2002-01-01" "2003-01-01" "2004-01-01"
##  [6] "2005-01-01" "2006-01-01" "2007-01-01" "2008-01-01" "2009-01-01"
## [11] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01" "2014-01-01"
## [16] "2015-01-01" "2016-01-01" "2017-01-01" "2018-01-01" "2019-01-01"
## [21] "2020-01-01"
## first days of biannual years
seq(as.Date("2000-01-01"), as.Date("2020-01-01"), "2 year")
##  [1] "2000-01-01" "2002-01-01" "2004-01-01" "2006-01-01" "2008-01-01"
##  [6] "2010-01-01" "2012-01-01" "2014-01-01" "2016-01-01" "2018-01-01"
## [11] "2020-01-01"
## by month
seq(as.Date("2020-01-01"), as.Date("2020-12-31"), "1 month")
##  [1] "2020-01-01" "2020-02-01" "2020-03-01" "2020-04-01" "2020-05-01"
##  [6] "2020-06-01" "2020-07-01" "2020-08-01" "2020-09-01" "2020-10-01"
## [11] "2020-11-01" "2020-12-01"
## the same as:
seq(as.Date("2020-01-01"), by = "month", length.out = 12)
##  [1] "2020-01-01" "2020-02-01" "2020-03-01" "2020-04-01" "2020-05-01"
##  [6] "2020-06-01" "2020-07-01" "2020-08-01" "2020-09-01" "2020-10-01"
## [11] "2020-11-01" "2020-12-01"
## quarters
seq(as.Date("2020-01-01"), as.Date("2020-12-31"), by = "1 quarter")
## [1] "2020-01-01" "2020-04-01" "2020-07-01" "2020-10-01"
## days
seq(as.Date("2020-01-01"), as.Date("2020-01-07"), by = "1 day")
## [1] "2020-01-01" "2020-01-02" "2020-01-03" "2020-01-04" "2020-01-05"
## [6] "2020-01-06" "2020-01-07"
## negative sequence
seq(as.Date("2020-07-01"), as.Date("2020-06-25"), by = "-1 day")
## [1] "2020-07-01" "2020-06-30" "2020-06-29" "2020-06-28" "2020-06-27"
## [6] "2020-06-26" "2020-06-25"
```

4.1 Logical vectors and operators

Exercises:

Using the command `data(mtcars)` load the dataset `mtcars` in R's environment. Inspect the dataset using `View(mtcars)` Set `df <- mtcars` so that we can change the values of the data set.

- Use logical operators to output only those rows of `df` where column `mpg` is stricltly higher then 15 and stricly lower then 20, so that 15 and 20 are not included
- Use logical operators to output only those rows of `df` where column `cyl` is equal to 6 and column `vs` is not 0.
- Use logical operators to output only those rows of `df` where column `gear` or `carb` have both the value 3.
- Use logical operators to output only the even rows of data.
- Use logical operators to output only the odd rows of data.
- Use logical operators and change every third element in column mpg to 999.
- Use logical operators to output only those rows of data where columns `gear` and `carb` have the same value, but print out only the `cyl`, `gear` and `carb` columns
- Use logical operators to output only those rows of data where columns `gear` and `carb` have the value 4, but print out only the `cyl`, `gear` and `carb` columns
- Use logical operators to output only those rows of data where columns `gear` or `carb` have the value 4, but print out only the `cyl`, `gear` and `carb` columns
- Use logical operators change all values that are 999 in the `mpg` column to 0.
- Use logical operators to output only those rows of `df` where `gear` and `carb` have different values

Solutions:

```
df <- mtcars

# use first one, is sometime columnnames can change
df[df[,1]>15|df[,1]<20,]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
```

```
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
df[df$mpg>15|df$mpg<20,]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2

df[df[,2]==6 & df[,8]!=0,]
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
df[df$cyl==6 & df$vs!=0,]
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4




df[df[,10]==3& df[,11]==3,]
##              mpg cyl  disp  hp drat   wt qsec vs am gear carb
## Merc 450SE  16.4   8 275.8 180 3.07 4.07 17.4  0  0    3    3
## Merc 450SL  17.3   8 275.8 180 3.07 3.73 17.6  0  0    3    3
## Merc 450SLC 15.2   8 275.8 180 3.07 3.78 18.0  0  0    3    3
```

```r
df[df$gear==3 & df$carb==3,]
##             mpg cyl  disp  hp drat   wt qsec vs am gear carb
## Merc 450SE  16.4   8 275.8 180 3.07 4.07 17.4  0  0    3    3
## Merc 450SL  17.3   8 275.8 180 3.07 3.73 17.6  0  0    3    3
## Merc 450SLC 15.2   8 275.8 180 3.07 3.78 18.0  0  0    3    3


df[c(FALSE,TRUE),]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
df[seq(from=2, along.with = df, by=2),]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2

df[c(TRUE,FALSE),]
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
df[seq(along.with = df, by=2),]
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
```

```r
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1

df[c(F,F,T),1] <- 999
df[seq(from=3,by=3,along.with = df),1] <- 999

df[df$gear==df$carb,c(1,10,11)]
##                 mpg gear carb
## Mazda RX4      21.0    4    4
## Mazda RX4 Wag  21.0    4    4
## Merc 280       19.2    4    4
## Merc 280C      17.8    4    4
## Merc 450SE    999.0    3    3
## Merc 450SL     17.3    3    3
## Merc 450SLC    15.2    3    3
## NA               NA   NA   NA
df[df$gear==4 & df$carb==4,c(1,10,11)]
##               mpg gear carb
## Mazda RX4     21.0    4    4
## Mazda RX4 Wag 21.0    4    4
## Merc 280      19.2    4    4
## Merc 280C     17.8    4    4
## NA              NA   NA   NA
df[df$gear==4 | df$carb==4,c(1,10,11)]
##                       mpg gear carb
## Mazda RX4            21.0    4    4
## Mazda RX4 Wag        21.0    4    4
## Datsun 710          999.0    4    1
## Duster 360           14.3    3    4
## Merc 240D            24.4    4    2
## Merc 230            999.0    4    2
## Merc 280             19.2    4    4
## Merc 280C            17.8    4    4
## Cadillac Fleetwood  999.0    3    4
## Lincoln Continental  10.4    3    4
## Chrysler Imperial    14.7    3    4
## Fiat 128            999.0    4    1
## Honda Civic          30.4    4    2
## Toyota Corolla       33.9    4    1
## Camaro Z28          999.0    3    4
## Fiat X1-9            27.3    4    1
## Ford Pantera L       15.8    5    4
## Volvo 142E           21.4    4    2
## NA                     NA   NA   NA


df[df$mpg==999,1] <- 0 # the same
df$mpg[df$mpg==999] <- 0 #the same


df[df$gear!=df$carb,c(1,10,11)]
##                       mpg gear carb
```

```
## Datsun 710          0.0    4    1
## Hornet 4 Drive     21.4    3    1
## Hornet Sportabout  18.7    3    2
## Valiant             0.0    3    1
## Duster 360         14.3    3    4
## Merc 240D          24.4    4    2
## Merc 230            0.0    4    2
## Cadillac Fleetwood  0.0    3    4
## Lincoln Continental 10.4   3    4
## Chrysler Imperial  14.7    3    4
## Fiat 128            0.0    4    1
## Honda Civic        30.4    4    2
## Toyota Corolla     33.9    4    1
## Toyota Corona       0.0    3    1
## Dodge Challenger   15.5    3    2
## AMC Javelin        15.2    3    2
## Camaro Z28          0.0    3    4
## Pontiac Firebird   19.2    3    2
## Fiat X1-9          27.3    4    1
## Porsche 914-2       0.0    5    2
## Lotus Europa       30.4    5    2
## Ford Pantera L     15.8    5    4
## Ferrari Dino        0.0    5    6
## Maserati Bora      15.0    5    8
## Volvo 142E         21.4    4    2
## NA                   NA   NA   NA
```

4.2 Missing values

4.2.1 Exercises

Create the following vector `x <- c(1,2,3,4,NA,NA,7,8)` * What is the length of the vector x? Are the NA's counted? * Print out all values of `x` without the `NA` * Replace all `NA` with value of 999

Create the following vector `x <- c(1,2,3,4,NA,NA,7,8)` * What is the length of the vector x? Are the NA's counted? * Replace first `NA` with value of 5 * Replace first `NA` with value of 6

Create the following vector `x <- c(NA,2,3,4,NA,NA,7,NA)` * Count all occurences of NA

Load dataset `mtcars` save is as `df`. * Change all values of `gear` that equal 4 to `NA` * Print `df` with only those lines, that have no `NA` in `gear` * Print `df` with only those lines, that have `NA` in `gear`

Load dataset `mtcars` save is as `df`. * Change all values of `mpg` that equal 10.4 to `NA` * Calcultate the mean of `mpg`

Create following dataframe `df <- data.frame(a=c(1,2,NA), b=c(2,4,6), c=c(NA,7,8))` * Display only those rows, not containing `NA` * Display all rows containing `NA` * Display all rows, where `a` column does not contain `NA`

4.2.2 Solutions

```
x <- c(1,2,3,4,NA,NA,7,8)

length(x)
## [1] 8
x[!is.na(x)]
## [1] 1 2 3 4 7 8
x[is.na(x)] <- 999


x <- c(1,2,3,4,NA,NA,7,8)
x[is.na(x)][1] <- 5
x[is.na(x)] <- 6
print(x)
## [1] 1 2 3 4 5 6 7 8
## or
x <- c(1,2,3,4,NA,NA,7,8)
x[is.na(x)] <- c(5,6)
print(x)
## [1] 1 2 3 4 5 6 7 8


x <- c(NA,2,3,4,NA,NA,7,NA)
sum(is.na(x))
## [1] 4
is.na(x)
## [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE


df <- mtcars
df$gear[df$gear==4] <- NA ## the same
df <- mtcars
df[df[,10]==4,10] <- NA ## the same

df[!is.na(df$gear),]
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
```

```
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
df[is.na(df$gear),]
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1   NA    4
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1   NA    4
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1   NA    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0   NA    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0   NA    2
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0   NA    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0   NA    4
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1   NA    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1   NA    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1   NA    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1   NA    1
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1   NA    2
```

```
df <- mtcars
df$mpg[df$mpg==10.4] <- NA ## the same
df <- mtcars
df[df[,1]==10.4,1] <- NA ## the same

mean(df$mpg, na.rm = TRUE) ## the same
## [1] 20.73667
mean(df[!is.na(df[,1]),1]) ## the same
## [1] 20.73667
```

```
df <- data.frame(a=c(1,2,NA), b=c(2,4,6), c=c(NA,7,8))
print(df)
##    a b  c
## 1  1 2 NA
## 2  2 4  7
## 3 NA 6  8

df[complete.cases(df),] ##the same
##   a b c
## 2 2 4 7
na.omit(df) ##the same
##   a b c
## 2 2 4 7
```

```
df[!complete.cases(df),]
##    a b  c
## 1  1 2 NA
## 3 NA 6  8

na.omit(df)
##   a b c
## 2 2 4 7

df[!is.na(df[,1]),]
##   a b  c
## 1 1 2 NA
## 2 2 4  7
```

4.3 Character vectors

4.3.1 Functions nchar, paste, outer

Excercises:

- Create a character string `Hello World` and assign it to an object `text`, then calculate the number of characters in the `text` object.
- Create a character vector `c("Hello", "World")` and assign it to an object `text`, then calculate the number of characters in the `text` object
- If `x<-"Hello"` and `y<-"World"`, write the code, so that the output becomes "Hello World"
- If `text<-"Hello World, we learn R"`, write code to extcract only "Hello World"
- If `text<-"Hello World, we learn Python"`, write code to change `Python` to `R`
- If `FName <- c("John", "Mary", "Stuart", "Ann")` and `LName <- c("Delton", "Braun", "Little", "Schmitt")`, combine both vectors to a data frame.
-
  - If `FName <- c("John", "Mary", "Stuart", "Ann")` and `LName <- c("Delton", "Braun", "Little", "Schmitt", "X")` try to combine both vectors ta a data frame. Then try to column-bind both vectors. What are the differences? Why does R behave in such a way?
- First crate an empty character vector named `greek` of length 5. Then set the first element to "alpha", fifth to "epsilon" ant print out the vector. Are the not filled elements `NA`?
- Using `outer`, and `letters` functions, create following vector `"aa" "ba" "ca" "ab" "bb" "cb" "ac" "bc" "cc"`

Solutions:

```
text <- "Hello World"
nchar(text)
## [1] 11

text <- c("Hello", "World")
nchar(text)
## [1] 5 5

x<-"Hello" ; y<-"World"
paste(x,y)
## [1] "Hello World"


text<-"Hello World, we learn R"
substr(text, start=1, stop=11)
## [1] "Hello World"

text<-"Hello World, we learn Python"
sub(pattern="Python", replacement = "R", x=text)
## [1] "Hello World, we learn R"


FName <- c("John", "Mary", "Stuart", "Ann")
LName <- c("Delton", "Braun", "Little", "Schmitt")
Names <- data.frame(FName, LName)
print(Names)
##    FName   LName
## 1   John  Delton
## 2   Mary   Braun
## 3 Stuart  Little
## 4    Ann Schmitt

FName <- c("John", "Mary", "Stuart", "Ann")
```

```r
LName <- c("Delton", "Braun", "Little", "Schmitt", "X")
## cannot create a data.frame as both vectors have different length
data.frame(FName, LName)
## Error in data.frame(FName, LName): arguments imply differing number of rows: 4, 5
## cbinds both vectors, but coarses the first vector, as both vectors have different length
cbind(FName, LName)
## Warning in cbind(FName, LName): number of rows of result is not a multiple of
## vector length (arg 1)
##      FName    LName
## [1,] "John"   "Delton"
## [2,] "Mary"   "Braun"
## [3,] "Stuart" "Little"
## [4,] "Ann"    "Schmitt"
## [5,] "John"   "X"

greek <- vector(mode = "character", length = 5)
greek[1] <- "alpha"
greek[5] <- "epsilon"
print(greek)
## [1] "alpha"   ""        ""        ""        "epsilon"
is.na(greek)
## [1] FALSE FALSE FALSE FALSE FALSE

as.vector(outer(letters[1:3], letters[1:3], FUN=paste, sep=""))
## [1] "aa" "ba" "ca" "ab" "bb" "cb" "ac" "bc" "cc"
```

4.4 Index vectors / subsetting

Exercises:

- If `x<-c("aa", "ba", "ca", "ab", "bb", "cb", "ac", "bc", "cc")` write code to receive `"aa" "ab" "ac"`
- If `x<-c("aa", "ba", "ca", "ab", "bb", "cb", "ac", "bc", "cc")` write code to receive `"aa" "aa" "aa"`
- If `x<-c("aa", "ba", "ca", "ab", "bb", "cb", "ac", "bc", "cc")` write code to receive `"ab" "ba"` refence to a music group ABBA
- If `x<-c("aa", "ba", "ca", "ab", "bb", "cb", "ac", "bc", "cc")` what will be the output of `x[-c(3:7)]`
- If `x <- c("a", "b", "c" , "d")`, what does `max(x)` return?
- If `x <- c("a", "b", "c" , "d")` and `test <- c(T,F,T,F)` what does `max(x[test])` return?
- If `x <- c(1,2,3,4)` and `test <- c(T,F,T,F)` what does `sum(x[test])` return?
- If `output <- data.frame(Factory=c(LETTERS[1:5]), Production=c(10,20,30,40,50))`, print the vector of Production but only for factories "A", "C" and "D". Then Sum the output of these three factories.
- If `x <- c("a", "b", "c" , "a", "c", "d", "e")` write R code to get a an index vector with positions of "a" and "c"

```
x<-c("aa", "ba", "ca", "ab", "bb", "cb", "ac", "bc", "cc")
x[c(1,4,7)]
## [1] "aa" "ab" "ac"
x[c(1,1,1)]
## [1] "aa" "aa" "aa"
x[c(4,2)]
## [1] "ab" "ba"
x[-c(3:7)]
## [1] "aa" "ba" "bc" "cc"

x <- c("a", "b", "c" , "d")
max(x)
## [1] "d"

x <- c("a", "b", "c" , "d")
test <- c(T,F,T,F)
max(x[test])
## [1] "c"

x <- c(1,2,3,4)
test <- c(T,F,T,F)
sum(x[test])
## [1] 4

output <- data.frame(Factory=c(LETTERS[1:5]), Production=c(10,20,30,40,50))
output$Production[output$Factory %in% c("A", "C", "D")]
## [1] 10 30 40
output$Production[output$Factory=="A"|output$Factory=="C"|output$Factory=="D"]
## [1] 10 30 40
sum(output$Production[output$Factory %in% c("A", "C", "D")])
## [1] 80

x <- c("a", "b", "c" , "a", "c", "d", "e")
## would return a logical index vector
x %in% c("a", "c")
## [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE
## using command which() shows the positions of TRUE
which(x %in% c("a", "c"))
## [1] 1 3 4 5
```

5 Modes and attributes

6 Factors

7 Arrays and matrices

8 List and Dataframes

9 Data structures

10 Character strings

11 Tables

12 Data manipulation

13 Dates and times

13.1 `as.Date()`

13.2 Package “lubridate”

13.3 Package “zoo()”

14 Controll structures

meh. . .

14.1 if

The simplest `if` conditional expression is: `if (condition) command`, where `condition` evaluates to a single logical value either `TRUE` or `FALSE`. If the condition is evaluated to `TRUE` the `command` gets excecuted If the conditions is evaluated to `FALSE` the `command` is not executed.

```
## Example:
# time <=17 evaluates to TRUE, thus if executes the command
time <- 13
if(time<=17) print("Day")
## [1] "Day"

# time <=17 evaluates to FALSE, thus if stops
time <- 19
if(time<=17) print("Day")
```

A more common usage is `if (condition) command else command`

```
## Example:
# time <=17 evaluates to TRUE, thus if executes the command
time <- 13
if(time<=17) print("Day") else print("Evening")
## [1] "Day"

# time <=17 evaluates to FALSE, thus if stops
time <- 19
if(time<=17) print("Day") else print("Evening")
## [1] "Evening"
```

It is also possible combine multiple if statements using `if (condition) command else if (condition) command else command`

```
## Example:

time <- 9
if(time<=12) print("Morning") else if(time<=17) print ("Day") else print("Evening")
## [1] "Morning"


time <- 13
if(time<=12) print("Morning") else if(time<=17) print ("Day") else print("Evening")
## [1] "Day"


time <- 19
if(time<=12) print("Morning") else if(time<=17) print ("Day") else print("Evening")
## [1] "Evening"
```

Often there are multiple commands that hace to be executed using `if` statements. In such cases `{...}` are used to group different commands.

```
if (condition1) {
    expr1
    } else if (condition2) {
    expr2
    } else if  (condition3) {
    expr3
    } else {
    expr4
}
```

Example:

```r
x<-10

if(time<=12){
  print("Good morning")
  print("Have a nice morning")
} else if(time<=17) {
  print ("Good Day")
  print("Have a nice day")
} else {
  print("Good evening")
  print("Have a nice evening")
}
## [1] "Good evening"
## [1] "Have a nice evening"
```

Note, that if condition is not a vecotrized operation, this means, the condition evaluates to a single logical value. So you cannot run `if` on vectors. In such cases only the first element of the vector gets evaluated.

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

if(x<3) "low" else "high"
## Warning in if (x < 3) "low" else "high": the condition has length > 1 and only
## the first element will be used
## [1] "low"
```

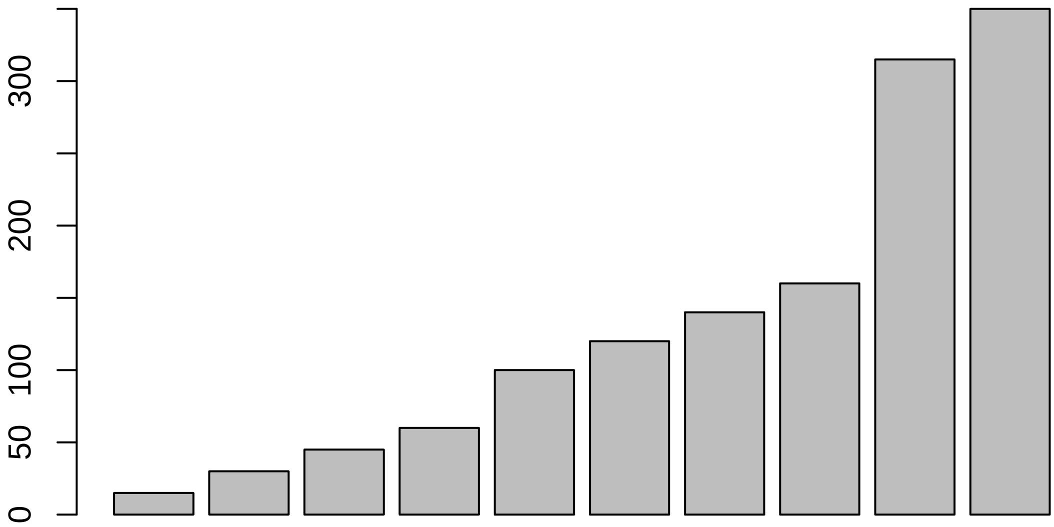
A vectorized version of the if/else construct is the `ifelse` function.

```r
x <- c(1,2,3,4,5)
ifelse(x<3, "low", "high")
## [1] "low"  "low"  "high" "high" "high"
```

Example: Assume you want calculate the sellers provisions for turnover in a shop. If turnover is bellow or equal 400, the provision rate is 15%, if turnover is higher then 400 or equal to 800, then the rate is 20%, for turnover above 800 rate is 35%. Calculate the sellers' provisions for turnovers: 100,200,...,1000. Build a simple barplot of the provisions using `barplot()` function

```r
turnover <- seq(from=100, to=1000, by=100)
provisions <- ifelse(turnover<=400, turnover*0.15,
                     ifelse(turnover<=800, turnover*0.2, turnover*0.35))

barplot(provisions)
```



14.2 for

The `for` loop conctruction has the follwoing form: `for (variable in vector) command`

where `variable` is the loop variable, `vector` is a vector expression and `command` is often a grouped (`{...}`) expression with its subexpressions using the dummyvariable `variable`. `command` is repeatedly evaluated as `variable` ranges through the values in the vector result of `vector`.

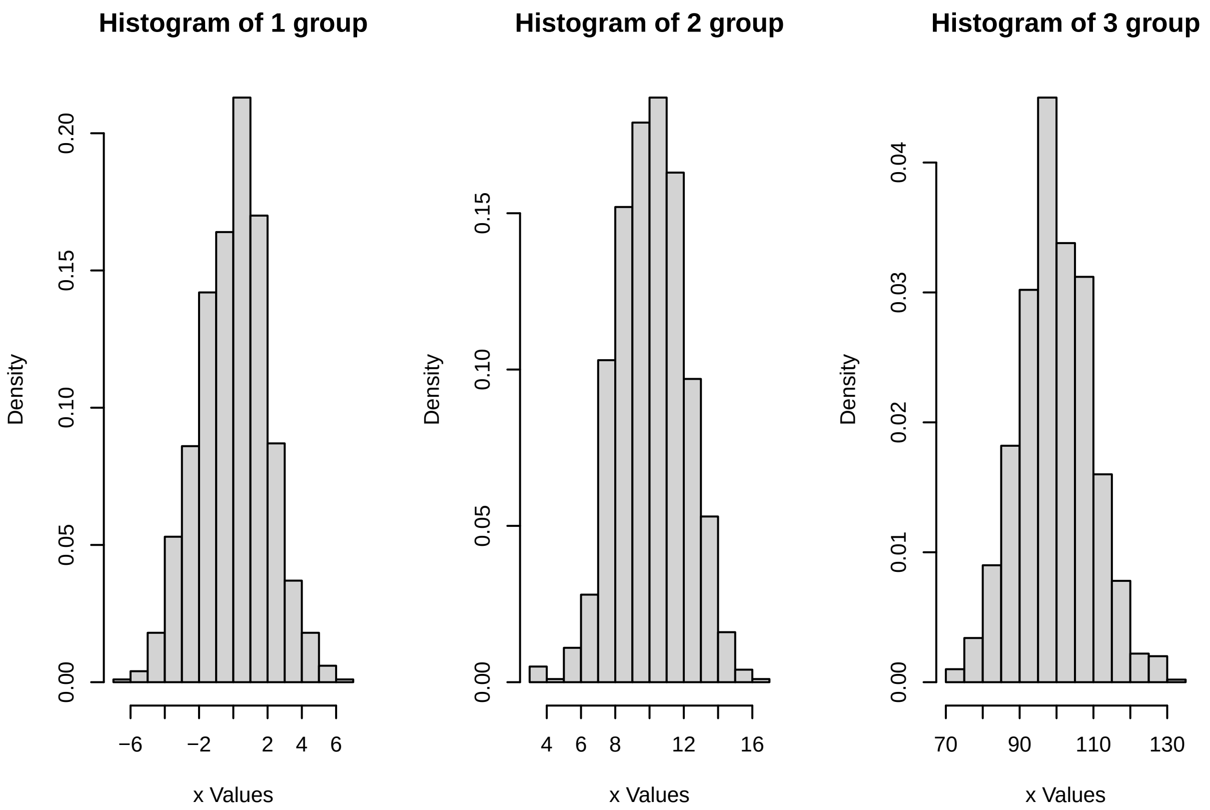
```
for (i in 1:5) {
  print(i)
}
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5


for (variable in c("a", "b", "c")) {
  print(variable)
}
## [1] "a"
## [1] "b"
## [1] "c"
```

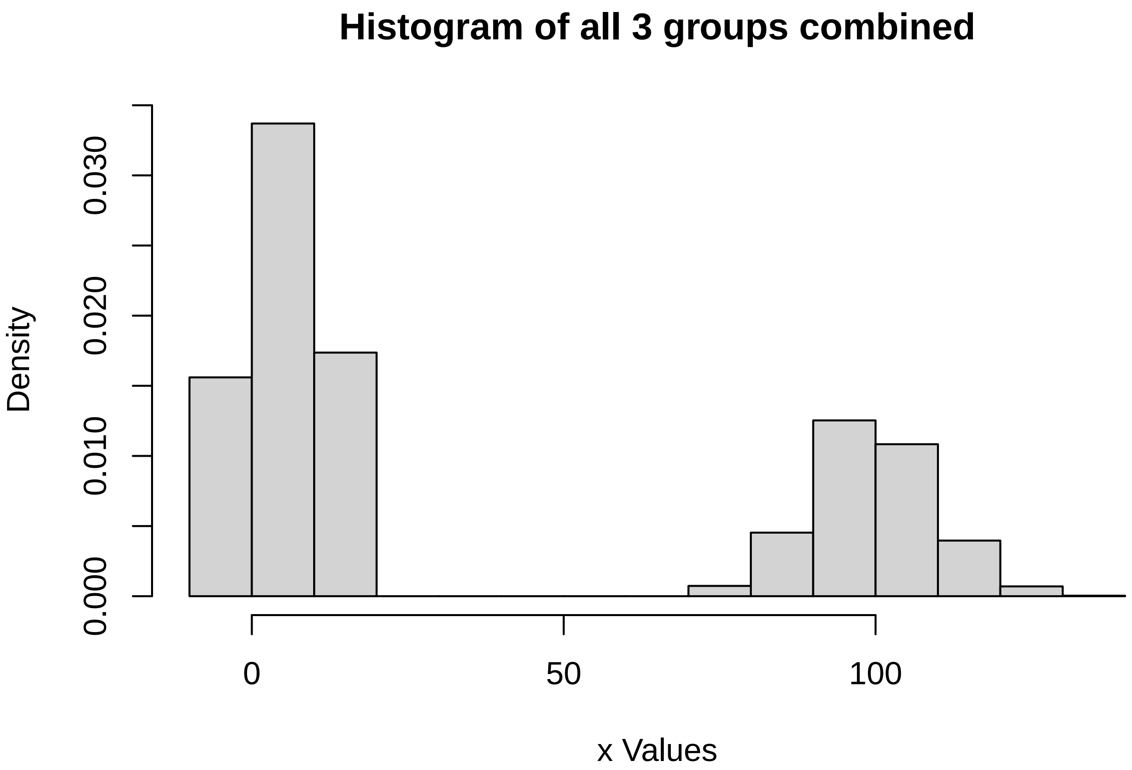
Example

```
df <- data.frame(x=c(rnorm(n=1000, mean=0, sd=2),
                     rnorm(n=1000, mean=10, sd=2),
                     rnorm(n=1000, mean=100, sd=10)),
                 ind=rep(c(1,2,3), each=1000))
x_split <- split(df$x, df$ind)

par(mfrow=c(1,3))
for(i in 1:length(x_split)){
  hist(x_split[[i]],
       freq = F,
       main=paste("Histogram of ", i," group", sep="" ),
       xlab="x Values")
}
```




```r
## for comparison
## one cannot distingish if there is a significant difference between 1 and 2 group
par(mfrow=c(1,1))
hist(df$x,
     freq = F,
     main="Histogram of all 3 groups combined",
     xlab="x Values")
```



14.3 while, repeat loops, break and next

`while (condition) command` and `repeat command` are two other looping facilities in R. These loops are more often used when trying to optimize some calculations.

`break` is the only command, that can stop the `repeat` loop, but it also stops all other loops. `next` statement is used to jump to the next loop cycle.

Here example:

After the Chernobyl explosion, radiation near the reactors was about 300000 micro Sieverts per hour. The half-life of cesium 137 is 30.17 years. How many years it would take, for the radiation near Chernobyl to have the safe radiation levels of 2 micro Sieverts per hour? Fore more information check here

```
x <- 300000 # micro Sieverts at reactor 4
normal <- 2 # micro Sieverts
halflife <- 30 # years
year <- 0 # initial value


while(x>normal){
  year <- year+halflife
  x <- round(x*0.5,1)
  print(c(years=year, rest_value=x))
}
##      years rest_value
##         30     150000
##      years rest_value
##         60      75000
##      years rest_value
##         90      37500
##      years rest_value
##        120      18750
##      years rest_value
##        150       9375
##      years rest_value
##      180.0     4687.5
##      years rest_value
##      210.0     2343.8
##      years rest_value
##      240.0     1171.9
##      years rest_value
##        270        586
##      years rest_value
##        300        293
##      years rest_value
##      330.0      146.5
##      years rest_value
##      360.0       73.2
##      years rest_value
##      390.0       36.6
##      years rest_value
##      420.0       18.3
##      years rest_value
##      450.0        9.2
##      years rest_value
##      480.0        4.6
##      years rest_value
##      510.0        2.3
##      years rest_value
##      540.0        1.1
```

Alternative with `repeat`

```r
x <- 300000 # micro Sieverts at reactor 4
normal <- 2 # micro Sieverts
halflife <- 30 # years
year <- 0 # initial value


repeat{
  year <- year+halflife
  x <- round(x*0.5,1)
  print(c(years=year, rest_value=x))
  if(x<normal){
    break
  }
}
##      years rest_value
##         30     150000
##      years rest_value
##         60      75000
##      years rest_value
##         90      37500
##      years rest_value
##        120      18750
##      years rest_value
##        150       9375
##      years rest_value
##      180.0     4687.5
##      years rest_value
##      210.0     2343.8
##      years rest_value
##      240.0     1171.9
##      years rest_value
##        270        586
##      years rest_value
##        300        293
##      years rest_value
##      330.0      146.5
##      years rest_value
##      360.0       73.2
##      years rest_value
##      390.0       36.6
##      years rest_value
##      420.0       18.3
##      years rest_value
##      450.0        9.2
##      years rest_value
##      480.0        4.6
##      years rest_value
##      510.0        2.3
##      years rest_value
##      540.0        1.1
```

An example with `next`

```r
# setting seed for reprudacability of results
set.seed(666)
## selecting 20 random numbers between 0 and 10
x <-runif(n=20,min=0, max=10)

for(i in 1:length(x)){
  if(x[i]<5){
    next
```

```
  }
  print(x[i])
}
## [1] 7.743685
## [1] 9.780138
## [1] 7.426119
## [1] 9.787284
## [1] 7.758931
## [1] 8.112564
## [1] 8.916374
```

14.4 Exercises

- Write R code, so that if

15 Functions

16 Graphical procedures

17 Reading data from files

18 Probability distributions

19 Statistical models