Introduction to R programming

African Institute for Mathematical Sciences - National Institute of Statistics of Rwanda

Dr. Lema Logamou SeknewnaSenior Data Scientist

09 September, 2025

Contents

What is R?

R a statistical programming language created in 1992 by two statisticians from the Auchkand University (New Zealand), Ross Ihaka & Robert Gentlemen. The first version (1.0) was released in February 2000. The name R comes from their first names' initials. The R language come from the S language also based on Fortran was developped between 1975 and 1976 at Bell Laboratories. The current version of R is 4.5.1.

Install R 4.5.1.

Choose your OS (operating system):

- Windows
- macOS: Apple silicon (M1-3)
- macOS: Intel
- and for Linux Debian, Fedora/Redhat, Ubuntu

Install Rstudio 2025.05.1513.

- Windows
- macOS
- Ubuntu 20/Debian 11
- Ubuntu 22/Debian 12
- Ubuntu 24

Setting up R & R markdown

Required packages

If you do not know how to install a package, refer to the section Packages.

- tinytex: to render a pdf document

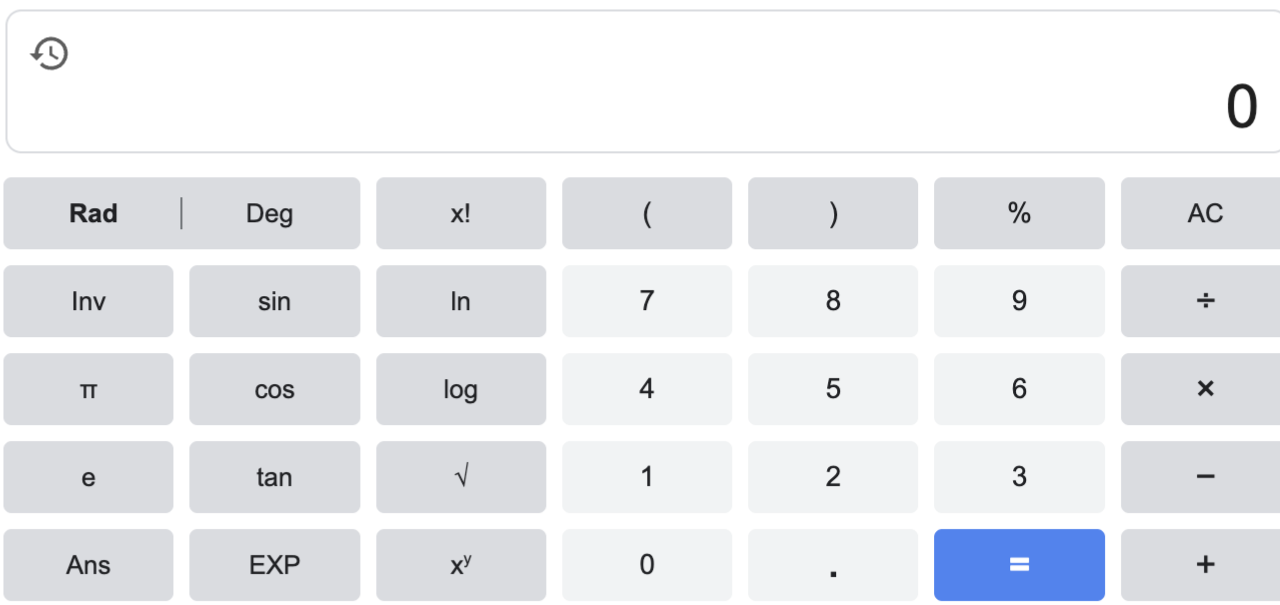
```
# the following code installs tinytex if it is not installed already.
if (!"tinytex" %in% rownames(installed.packages())){
  install.packages("tinytex")
}

library(tinytex)
install_tinytex() # to install LaTeX
```

- Rtools 64 - aarch64
- You can also access R online from Posit Cloud

Training Agenda

Operators

R is a calculator because you can perform all operations in the R console.



Arithmetic Operators

Addition: +

```
1+1
```

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

Subtraction: `-`

```r
1-1
```

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

Multiplication: `*`

```r
1*1
```

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

Division: `/`

```r
1/1
```

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

Modulus (remaining of a division) : `%%`

```r
1 %% 2
```

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

Exponent : `^` or `**`

```r
2 ^ 10 # or 2 ** 10
```

```
## [1] 1024
```

Integer division: `%/%`

```r
1035 %/% 3
```

```
## [1] 345
```

Logical operators

Less than: `<`

```r
1 < 1
```

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

Less than or equal to: `<=`

```r
1 <= 1
```

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

Greater than: `>`

```r
1 > 1
```

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

Greater than or equal to: >=

```r
1 >= 1
```

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

Exactly equal to: ==

```r
"R" == "r"
```

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

R est sensible à la casse !!!

The equality operator can also be used to match one element with multiple elements

```r
"Species" == c("Sepal.Length", "Sepal.Width", "Petal.Length",
                   "Petal.Width", "Species")
```

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

Not equal to: !=

```r
1 != 1
```

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

Negation/NOT: !

Used to change a `TRUE` condition to `FALSE` (respectively a `FALSE` condition to `TRUE`)

```r
!TRUE # or !T
```

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

```r
!FALSE # or !F
```

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

```r
!(T & F) # this is TRUE
```

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

```r
!(F | T) # is FALSE
```

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

AND: &

```r
TRUE & TRUE
```

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

```r
TRUE & FALSE
```

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

```r
FALSE & FALSE
```

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

OR: `|`

```r
TRUE | TRUE
```

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

```r
TRUE | FALSE
```

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

```r
FALSE | FALSE
```

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

Value Matching

In R, we also have inbuilt functions that help to match element of a given vector. The first function is `match()`. You can check the documentation with `help("match")` or `?match`. Read that: **`match` returns a vector of the positions of (first) matches of its first argument in its second.**

```r
match("Species", c("Sepal.Length", "Sepal.Width", "Petal.Length",
                   "Petal.Width", "Species"))
```

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

The second function `%in%` check the existence of a value in a given vector (of values).

```r
"Species" %in% c("Sepal.Length", "Sepal.Width", "Petal.Length",
                   "Petal.Width", "Species")
```

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

R object and assignment

In R we can use `<-`, `=` (single equal sign !) and `->` to assign a value to a variable.

A variable name:

- can begin with a character or dot(s). Ex: `a <- 1`, `0 -> .a`
- should not contain space. Replace empty space with `_` or a dot `.`.

```r
v rsion <- 4.3.2
```

```
## Error in parse(text = input): <text>:1:3: unexpected symbol
## 1: v rsion
##       ^
```

- can contain numbers. Ex: `a1 <- 1`.

```r
a <- 1
b <- 2
0 -> .a
a1 = .a
```

Data types

In R we have the following data types: * numeric * integer * complex * character * logical * raw * factor

Numeric/double

Examples of numberic numbers are 10.5, 55, 787, pi

```
PI <- pi; class(PI); typeof(PI)
```

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

```
## [1] "double"
```

```
n <- 55; class(n); typeof(n)
```

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

```
## [1] "double"
```

Integer

- (1L, 55L, 100L, where the letter `L` declares this as an integer).
- Check the class of n <- 55L. What do you see?

```
n <- 55L
class(n)
```

```
## [1] "integer"
```

Complex

An example of a complex number is $9 + 3i$, where i is the imaginary part. Multiplying a real number by `1i`, transforms it to complex.

```
z <- 9 + 3i
class(z)
```

```
## [1] "complex"
```

```
typeof(z)
```

```
## [1] "complex"
```

```
z1 <- a + 1i*b
print(z1)
```

```
## [1] 1+2i
```

```
class(z1)
```

```
## [1] "complex"
```

Character/string

```
string <- "I am Learning R"
class(string)
```

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

Remember!! `LeaRning` is different from `Learning`.

Logical/Boolean - (TRUE or FALSE)

```
TRUE # or T
```

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

```
FALSE # or F
```

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

Logical output can also be an outcome of a test. Example: if we want to check if `"LeaRning" == "Learning"`

```r
"LeaRning" == "Learning"
```

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

Raw

```r
text <- "I am learning R."
(raw_text <- charToRaw(text))
```

```
##  [1] 49 20 61 6d 20 6c 65 61 72 6e 69 6e 67 20 52 2e
```

```r
class(raw_text)
```

```
## [1] "raw"
```

Converting raw to text:

```r
rawToChar(raw_text)
```

```
## [1] "I am learning R."
```

Factors

They are a data type that is used to refer to a qualitative relationship like colors, good & bad, course or movie ratings, etc. They are useful in statistical modeling.

```r
Gender <- factor(c("Female", "Male"))
print(Gender)
```

```
## [1] Female Male
## Levels: Female Male
```

```r
class(Gender)
```

```
## [1] "factor"
```

Logical

```r
v <- TRUE
w <- FALSE

class(v); typeof(v)
```

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

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

```r
!v
```

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

```r
isTRUE(w)
```

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

```r
# if (isTRUE(v)) {
#   print("This code is compiled")
# }
```

Create object

- Numeric object

```r
n <- 10
x <- numeric(n) # creates a numeric object of size n
print(x)
```

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

```r
# assigning values to x:
x[1] <- 2.5
print(x)
```

```
##  [1] 2.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

- Integer

```r
n <- 10
x <- integer(n) # creates a numeric object of size n
print(x)
```

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

```r
class(x)
```

```
## [1] "integer"
```

```r
# assigning values to x:
x[1] <- 2.5 # R will automatically convert integer to numeric
class(x)
```

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

```r
print(x)
```

```
##  [1] 2.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Convertions:

To convert data in R we can use function starting with `as.` + data type from the `base` package.

- Numeric to character
- Character to numeric
- Factor to character
- Character to factor
-

R Data Structures

The most used data types in R are

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Scalars and vectors (1D):

- A scalar is any number in N, Z, D, Q, R, or C (Quantum Mechanics)
- Vectors: collection of objects of the same type. A vector can also be a sequence;

Let create a vector with elements of different types to see how R will deal with them.

- Numerics and characters

```
# ?c
v <- c(1, "R", T, FALSE, NA)
# print v
print(v)
```

```
## [1] "1"     "R"     "TRUE"  "FALSE" NA
```

```
# what is the class of v?
class(v)
```

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

R converts everything in character type except NA which is common to numeric and character.

- Numeric and logical

```
v2 <- c(1, 4, 8, FALSE, TRUE, FALSE, FALSE, TRUE, "R" == "r")
print(v2)
```

```
## [1] 1 4 8 0 1 0 0 1 0
```

Here, R converts everything into numeric. FALSE is 0 and TRUE is 1.

- Create a sequence with `seq()` function

```
x <- seq(0, 2*pi, length.out = 90)
length(x)
```

```
## [1] 90
```

```
unique(round(diff(x), 5)) # to get the common difference
```

```
## [1] 0.0706
```

```
y <- seq(0, 2*pi, by = 0.07059759)
n <- 20
head(x, n = n); head(y, n = n)
```

```
##  [1] 0.00000000 0.07059759 0.14119518 0.21179276 0.28239035 0.35298794
##  [7] 0.42358553 0.49418311 0.56478070 0.63537829 0.70597588 0.77657346
## [13] 0.84717105 0.91776864 0.98836623 1.05896382 1.12956140 1.20015899
## [19] 1.27075658 1.34135417
```

```
##  [1] 0.00000000 0.07059759 0.14119518 0.21179277 0.28239036 0.35298795
##  [7] 0.42358554 0.49418313 0.56478072 0.63537831 0.70597590 0.77657349
## [13] 0.84717108 0.91776867 0.98836626 1.05896385 1.12956144 1.20015903
## [19] 1.27075662 1.34135421
```

```
tail(x)
```

```
## [1] 5.930197 6.000795 6.071393 6.141990 6.212588 6.283185
```

```
range(x)
```

```
## [1] 0.000000 6.283185
```

If you want the difference between the max and the min of x

```
diff(c(4, 6))
```

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

```
diff(range(x)) # the same as max(x) - min(x)
```

```
## [1] 6.283185
```

```
rg <- range(x)
rg[1]
```

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

```
rg[2]
```

```
## [1] 6.283185
```

```
x[11.8] == x[11] # R considers the floor as index.
```

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

```
x[9.123486785] == x[9]
```

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

```
as.integer(9.123486785) # this is what R does when the index is not an integer.
```

```
## [1] 9
```

The length of a vector is given by:

```
length(x)
```

```
## [1] 90
```

```
length(rg)
```

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

```
a <- 9
length(a)
```

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

A scalar is a vector of length 1.

Example 2:

```
# repeating
rep("I learn R", 10)
```

```
##  [1] "I learn R" "I learn R" "I learn R" "I learn R" "I learn R" "I learn R"
##  [7] "I learn R" "I learn R" "I learn R" "I learn R"
```

```
rep(c(0, 1), 10)
```

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

```
v <- rep(0, 10)
v <- numeric(10)
v[10] <- NA
v
```

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

```r
# repetition
rep(c(0:1), c(50, 50))
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```r
rep(c(0:1), each = 50)
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Random generation

```r
# sampling
set.seed(123) # fix the randomness for reproducibility.
sample(0:1, size = 200, replace = TRUE, prob = c(1/3, 1-1/3)) -> y
print(y)
```

```
##   [1] 1 0 1 0 0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 0 1 1 0
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 0 0 1 0 1 0 1
##  [75] 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 1 1 0
## [112] 1 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 1 1
## [149] 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1
## [186] 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1
```

What does the following code do?

```r
sum(y == 0) # gives the count of zeros
```

```
## [1] 62
```

```r
sum(y == 1) # gives the count of ones
```

```
## [1] 138
```

The `table()` function does the work for you

```r
table(y)
```

```
## y
##   0   1
##  62 138
```

Proportions: `prop.table()` on `table()`

```r
tab <- table(y)
prop.table(tab)
```

```
## y
##    0    1
## 0.31 0.69
```

```r
mean(y == 0)
```

```
## [1] 0.31
```

```r
mean(y == 1)
```

```
## [1] 0.69
```

```r
sum(y == 0); sum(!y == 0)
```

```
## [1] 62
```

```
## [1] 138
```

```r
as.numeric(TRUE)
```

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

```r
as.numeric(FALSE)
```

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

```r
set.seed(123)
participants <- sample(c("Female", "Male", "Child"), size = 120, replace = TRUE)
head(participants, 20) # displays the first 20 elements of the sample
```

```
##  [1] "Child"  "Child"  "Child"  "Male"   "Child"  "Male"   "Male"   "Male"
##  [9] "Child"  "Female" "Male"   "Male"   "Female" "Male"   "Child"  "Female"
## [17] "Child"  "Child"  "Female" "Female"
```

```r
table(participants)
```

```
## participants
##  Child Female   Male
##     40     36     44
```

```r
participants
```

```
##   [1] "Child"  "Child"  "Child"  "Male"   "Child"  "Male"   "Male"   "Male"
##   [9] "Child"  "Female" "Male"   "Male"   "Female" "Male"   "Child"  "Female"
##  [17] "Child"  "Child"  "Female" "Female" "Female" "Female" "Child"  "Male"
##  [25] "Child"  "Male"   "Female" "Male"   "Child"  "Male"   "Female" "Child"
##  [33] "Child"  "Female" "Child"  "Male"   "Female" "Child"  "Female" "Female"
##  [41] "Male"   "Child"  "Child"  "Female" "Child"  "Female" "Child"  "Male"
##  [49] "Female" "Male"   "Female" "Female" "Child"  "Female" "Male"   "Female"
##  [57] "Female" "Child"  "Female" "Male"   "Female" "Child"  "Female" "Child"
##  [65] "Male"   "Child"  "Male"   "Male"   "Child"  "Male"   "Male"   "Child"
##  [73] "Child"  "Female" "Male"   "Male"   "Female" "Male"   "Female" "Female"
##  [81] "Male"   "Child"  "Child"  "Female" "Male"   "Female" "Male"   "Female"
##  [89] "Child"  "Child"  "Male"   "Child"  "Female" "Male"   "Male"   "Child"
##  [97] "Male"   "Female" "Child"  "Child"  "Child"  "Male"   "Male"   "Child"
## [105] "Female" "Female" "Child"  "Male"   "Male"   "Male"   "Male"   "Male"
## [113] "Male"   "Child"  "Male"   "Female" "Male"   "Male"   "Male"   "Child"
```

Count of females

```r
sum(participants == "Female")
```

```
## [1] 36
```

Proportion of females

```r
mean(participants == "Female")
```

```
## [1] 0.3
```

From our survey

```r
Gender <- c("Female", "Male", "Male", "Male", "Female", "Female", "Male",
"Male", "Female", "Female", "Male", "Male", "Male", "Male", "Female",
"Female", "Male", "Female", "Female", "Male", "Female", "Male",
"Male", "Male", "Male", "Female", "Femaale", "Male", "Female",
"Male", "Male", "Female", "Male", "Female", "Malle", "Male", "Female",
"Female", "Femmale", "Female", "Female", "Female", "Female", "Male",
"Male", "Male", "Male")

# counts
sum(Gender == "Female")
```

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

```r
sum(Gender != "Female") # length(Gender) - sum(Gender == "Female")
```

```
## [1] 27
```

```r
# proportions
mean(Gender == "Female")
```

```
## [1] 0.4255319
```

```r
mean(Gender != "Female") # 1 - mean(Gender == "Female")
```

```
## [1] 0.5744681
```

```r
# using table function
table(Gender)
```

```
## Gender
## Femaale  Female Femmale    Male   Malle
##       1      20       1      24       1
```

```r
# prop.table()
prop.table(table(Gender))
```

```
## Gender
##   Femaale    Female   Femmale      Male     Malle
## 0.0212766 0.4255319 0.0212766 0.5106383 0.0212766
```

```r
Gender[Gender == "Femaale"] <- "Female"
Gender[Gender == "Femmale"] <- "Female"
Gender[Gender == "Malle"] <- "Male"

# checking if the Gender variable has only 2 classes
table(Gender)
```

```
## Gender
## Female   Male
##     22     25
```

Sub-setting using sample function

```r
set.seed(76)
table(sample(Gender, size = 20))
```

```
##
## Female   Male
```

```
##     10     10
```

Mimicking the LUDO game

```r
table(sample(6, size = 10000, replace = TRUE))
```

```
##
##    1    2    3    4    5    6
## 1697 1688 1706 1594 1665 1650
```

```r
prop.table(table(sample(6, size = 10000, replace = TRUE)))
```

```
##
##      1      2      3      4      5      6
## 0.1649 0.1663 0.1717 0.1692 0.1669 0.1610
```

```r
k <- 2
v <- c(1, 0, 3)

# addition
k + v
```

Operations

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

```r
v + k
```

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

```r
# subtraction
k - v
```

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

```r
v - k
```

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

```r
# multiplication
k * v
```

```
## [1] 2 0 6
```

```r
v * k
```

```
## [1] 2 0 6
```

```r
# division
v / k
```

```
## [1] 0.5 0.0 1.5
```

```r
k / v # is the inverse of v/k
```

```
## [1] 2.0000000       Inf 0.6666667
```

```r
k <- 1/2 # scalar
v <- c(pi, 0, 1, 4)
w <- c(0, pi, pi, 0, 0)

# product of scalar and vector
k*v; v*k
```

```
## [1] 1.570796 0.000000 0.500000 2.000000
```

```
## [1] 1.570796 0.000000 0.500000 2.000000
```

```r
# raise a vector to a power k?
v^k
```

```
## [1] 1.772454 0.000000 1.000000 2.000000
```

```r
# v + v
v + w
```

```
## Warning in v + w: longer object length is not a multiple of shorter object
## length
```

```
## [1] 3.141593 3.141593 4.141593 4.000000 3.141593
```

```r
v + v
```

```
## [1] 6.283185 0.000000 2.000000 8.000000
```

```r
v^w
```

```
## Warning in v^w: longer object length is not a multiple of shorter object length
```

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

Matrices (2D):

Matrices are two dimensional data set with columns and rows.

```r

(A <- matrix(1:25, ncol = 5, nrow = 5)) # byrow = F by default
```

Matrix definition

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

```r
(B <- matrix(1:25, nrow = 5, ncol = 5, byrow = T)) # ncol = 5 is optional.
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
## [5,]   21   22   23   24   25
```

Exercise:

Define the following matrix in R

```
$$
\begin{pmatrix}
1 & 0 & 0 & 1\\
0 & 1 & 1 & 1\\
0 & 0 & 1 & 1\\
\end{pmatrix}
$$
```

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

```r
# define the matrix above
v <- c(1, rep(0, 2), 1, 0, rep(1, 3), rep(0, 2), rep(1, 2))
A = matrix(v, 3, byrow = TRUE)
# displaying A
print(A)
```

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

```r
# let's check the dimension of A
dim(A)
```

```
## [1] 3 4
```

Define a matrix using the `dim()` function which stands for `dimension` of a given matrix.

```r
v <- c(1, rep(0, 2), 1, 0, rep(1, 3), rep(0, 2), rep(1, 2))
dim(v) <- c(4, 3)
v <- t(v)
v
```

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

Define a 0 matrix

```r
n <- 6
p <- 8

(zeros <- rep(0, n*p)) # also numeric(n*p)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 0 0 0 0 0 0 0 0
```

```r
matrix(zeros, nrow = n, ncol = p)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    0    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0    0    0
## [6,]    0    0    0    0    0    0    0    0
```

```r
dim(zeros) <- c(n, p)
print(zeros)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    0    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0    0    0
```

```
## [4,]    0    0    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0    0    0
## [6,]    0    0    0    0    0    0    0    0
```

```r
# very short way is:
matrix(0, nrow = n, ncol = p)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    0    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0    0    0
## [6,]    0    0    0    0    0    0    0    0
```

```r
(A <- matrix(c(1, 0, 2, 5, 2, 1, 4, 2, 0), nrow = 3))
```

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

```r
(B <- matrix(c(2, 5, 2, 3, 1, 1, 0, 1, 1), nrow = 3))
```

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

Matrix from vectors We can also construct a matrix from vectors $M = (v_1, v_2, v_3)$ using the `cbind` and `rbind` functions.

```r
v1 <- c(1, 0, 2); v2 <- c(5, 2, 1); v3 <- c(4, 2, 0)
(M1 <- cbind(v1, v2, v3))
```

```
##      v1 v2 v3
## [1,]  1  5  4
## [2,]  0  2  2
## [3,]  2  1  0
```

```r
(M2 <- rbind(v1, v2, v3))
```

```
##    [,1] [,2] [,3]
## v1    1    0    2
## v2    5    2    1
## v3    4    2    0
```

```r
class(M1)
```

```
## [1] "matrix" "array"
```

```r
class(M2)
```

```
## [1] "matrix" "array"
```

Matrix using `dim` function ! `dim` is also called to check the dimension of a matrix, a data frame or an array.

```r
M3 <- c(1, 5, 4, 0, 2, 2, 2, 1, 0)
dim(M3) <- c(3, 3) # sets the dimensions of M3
dim(M3) # shows the dimensions of M3
```

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

```r
M3
```

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

```r
class(M3);
```

```
## [1] "matrix" "array"
```

Matrix operations

- Transpose

```r
(A_T <- t(A))
```

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

- Addition

```r
A + B
```

```
##      [,1] [,2] [,3]
## [1,]    3    8    4
## [2,]    5    3    3
## [3,]    4    2    1
```

- Substraction

```r
A - B
```

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

- Multiplication

```r
# number of columns in A: dim(A)[2], or ncol(A).
# number of rows in A: dim(A)[1], or nrow(A)
dim(A)[2] == ncol(A)
```

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

```r
ncol(A) == nrow(B)
```

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

```r
A %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   35   12    9
## [2,]   14    4    4
```

```
## [3,]    9    7    1
```

- Inverse

```r
# I want to get the inverse of A
(A_inv <- solve(A))
```

```
##      [,1] [,2] [,3]
## [1,]   -1  2.0    1
## [2,]    2 -4.0   -1
## [3,]   -2  4.5    1
```

```r
A %*% A_inv # is to check if A_inv is really the inverse of A.
```

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

Solving a system of equations

$$\begin{cases} 2x + 2y & = 0 \\ x + 3y & = 2 \end{cases}$$

The matrix of the equation system is: $A = \begin{pmatrix} 2 & 2 \\ 1 & 3 \end{pmatrix}$ and the right hand side of the equation is $b = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$. We can use the `solve` function to have the solutions.

```r
A1 <- matrix(c(2, 2, 1, 3), nrow = 2, byrow = TRUE)
b <- c(0, 2)
solve(A1, b)
```

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

```r
# A1*A1 # point-wise multiplication.
```

- Division: multiply a matrix by the inverse of another. $B/A = BA^{-1}$

```r
B %*% A_inv
```

```
##      [,1] [,2] [,3]
## [1,]    4 -8.0   -1
## [2,]   -5 10.5    5
## [3,]   -2  4.5    2
```

```r
round(A_inv %*% A, 0)
```

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

Eigen values/vectors (basis of Principal Component Analysis) Requirements:

- `A` should be a square matrix of dimension n.
- The eigen values λ are solutions of the characteristic polynomial

$$P_A(\lambda) = \det(A - \lambda I_n) = 0, \quad n \in \mathbb{N}.$$

```r
ev <- eigen(A)  # gives a list of eigen values and
                # eigen vectors

print(ev)
```

Eigen values/vectors

```
## eigen() decomposition
## $values
## [1]  4.7664355 -1.4836116 -0.2828239
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.8535725 -0.3668743  0.2177685
## [2,] -0.3052279 -0.4631774 -0.6431613
## [3,] -0.4221966  0.8067651  0.7341120
```

```r
print(ev$values)
```

```
## [1]  4.7664355 -1.4836116 -0.2828239
```

```r
P <- ev$vectors
round(solve(P) %*% A %*% P, 4)
```

```
##        [,1]    [,2]    [,3]
## [1,] 4.7664  0.0000  0.0000
## [2,] 0.0000 -1.4836  0.0000
## [3,] 0.0000  0.0000 -0.2828
```

Example:

```r
A <- matrix(c(2, 1, 0, 3), ncol = 2, byrow = TRUE)
ev <- eigen(A)
ev$values
```

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

```r
ev$vectors
```

```
##           [,1] [,2]
## [1,] 0.7071068    1
## [2,] 0.7071068    0
```

Exercise for tomorrow

1. remove the last column of the iris data. Save it in `dta`

```r
# TODO
```

2. Calculate the average of each column of the remaining data `dta`. Save it in `avg`

```r
# TODO
```

3. Scale `dta` by removing the average of each column. Name it it `scaled_dta`

```r
# TODO
```

4. Calculate the eigen values and eigen vectors of `t(scaled_dta) %*% scaled_dta`

```r
# TODO
```

5. Projection on the first two axis

```r
# TODO
```

Arrays

Arrays are data type with more than two dimensions

```r
(aRray <- array(1:24, dim = c(3, 4, 2)))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
```

```r
class(aRray)
```

```
## [1] "array"
```

An example of array is NetCDF data with for instance: * Longitude as column names (n) * Latitude as row names (p) * 3rd dimension could the time. For each time, we have a $n \times p$ matrix.

```r
dim(aRray)
```

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

```r
aRray[1, 1, 2] # element at i=1, j=1 from the second matrix
```

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

The dimension: row position, column position, matrix level

Lists

A list is a collection of object of different types. The sizes of elements could be different.

```r
L <- list()

A <- matrix(0, 5, 6)

L$A <- A
print(L)
```

```
## $A
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
```

```r
# adding a seqeuence to L
L$my_seq <- sample(10, 5)
print(L)
```

```
## $A
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## $my_seq
## [1]  7 10  8  6  4
```

```r
# add a boolean
L$Bool <- TRUE
print(L)
```

```
## $A
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## $my_seq
## [1]  7 10  8  6  4
##
## $Bool
## [1] TRUE
```

```r
# adding the iris
L[["iris"]] <- head(iris, 10)
print(L)
```

```
## $A
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## $my_seq
## [1]  7 10  8  6  4
##
## $Bool
## [1] TRUE
##
## $iris
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
```

```
## 3           4.7         3.2         1.3         0.2  setosa
## 4           4.6         3.1         1.5         0.2  setosa
## 5           5.0         3.6         1.4         0.2  setosa
## 6           5.4         3.9         1.7         0.4  setosa
## 7           4.6         3.4         1.4         0.3  setosa
## 8           5.0         3.4         1.5         0.2  setosa
## 9           4.4         2.9         1.4         0.2  setosa
## 10          4.9         3.1         1.5         0.1  setosa
```

```r
# let's check the number elements in L
length(L)
```

```
## [1] 4
```

```r
# adding 5th element to list L
L[[5]] <- "I am learning R"
print(L)
```

```
## $A
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## $my_seq
## [1]  7 10  8  6  4
##
## $Bool
## [1] TRUE
##
## $iris
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5         1.4         0.2  setosa
## 2           4.9         3.0         1.4         0.2  setosa
## 3           4.7         3.2         1.3         0.2  setosa
## 4           4.6         3.1         1.5         0.2  setosa
## 5           5.0         3.6         1.4         0.2  setosa
## 6           5.4         3.9         1.7         0.4  setosa
## 7           4.6         3.4         1.4         0.3  setosa
## 8           5.0         3.4         1.5         0.2  setosa
## 9           4.4         2.9         1.4         0.2  setosa
## 10          4.9         3.1         1.5         0.1  setosa
##
## [[5]]
## [1] "I am learning R"
```

Getting the `names` of element of L?

```r
names(L)
```

```
## [1] "A"      "my_seq" "Bool"   "iris"   ""
```

Renaming the 5th element of L?

```r
names(L)[5] <- "String"
print(L)
```

```
## $A
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
##
## $my_seq
## [1]  7 10  8  6  4
##
## $Bool
## [1] TRUE
##
## $iris
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
##
## $String
## [1] "I am learning R"
```

```r
mylist <- list("matrix" = A,
               "sequence" = x,
               "Bool" = TRUE
               # ,
               # "Array" = aRray
               )

mylist$matrix
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
```

```r
class(mylist[[1]])
```

Accessing elements of a list

```
## [1] "matrix" "array"
```

```r
mylist$Array
```

```
## NULL
```

```r
mylist[c("Array", "matrix")]
```

Accessing elements of a list

```
## $<NA>
## NULL
##
## $matrix
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
```

Data Frames

A data frame is a table of n number of rows (observations) and p number of columns (features or variables). Variables can be of any data type.

- Converting a continuous variable into a categorical variable

```r
set.seed(123)
n <- 120
df <- data.frame(
  "ID" = paste0("Particip_", 1:n),
  "age" = sample(0:120, size = n)
)


head(df, 10)
```

```
##             ID age
## 1   Particip_1  30
## 2   Particip_2  78
## 3   Particip_3  50
## 4   Particip_4  13
## 5   Particip_5  66
## 6   Particip_6  41
## 7   Particip_7  49
## 8   Particip_8  42
## 9   Particip_9 100
## 10 Particip_10 117
```

```r
brks <- seq(0, 120, by = 10)
df$age_groups <- cut(df$age, breaks = brks, include.lowest = TRUE)

head(df, 20)
```

```
##             ID age age_groups
## 1   Particip_1  30    (20,30]
## 2   Particip_2  78    (70,80]
## 3   Particip_3  50    (40,50]
## 4   Particip_4  13    (10,20]
## 5   Particip_5  66    (60,70]
## 6   Particip_6  41    (40,50]
```

```
## 7   Particip_7  49    (40,50]
## 8   Particip_8  42    (40,50]
## 9   Particip_9 100   (90,100]
## 10 Particip_10 117  (110,120]
## 11 Particip_11  24    (20,30]
## 12 Particip_12  89    (80,90]
## 13 Particip_13  90    (80,90]
## 14 Particip_14  68    (60,70]
## 15 Particip_15 108  (100,110]
## 16 Particip_16  56    (50,60]
## 17 Particip_17  91   (90,100]
## 18 Particip_18   8     [0,10]
## 19 Particip_19  92   (90,100]
## 20 Particip_20  98   (90,100]
```

Exercise:

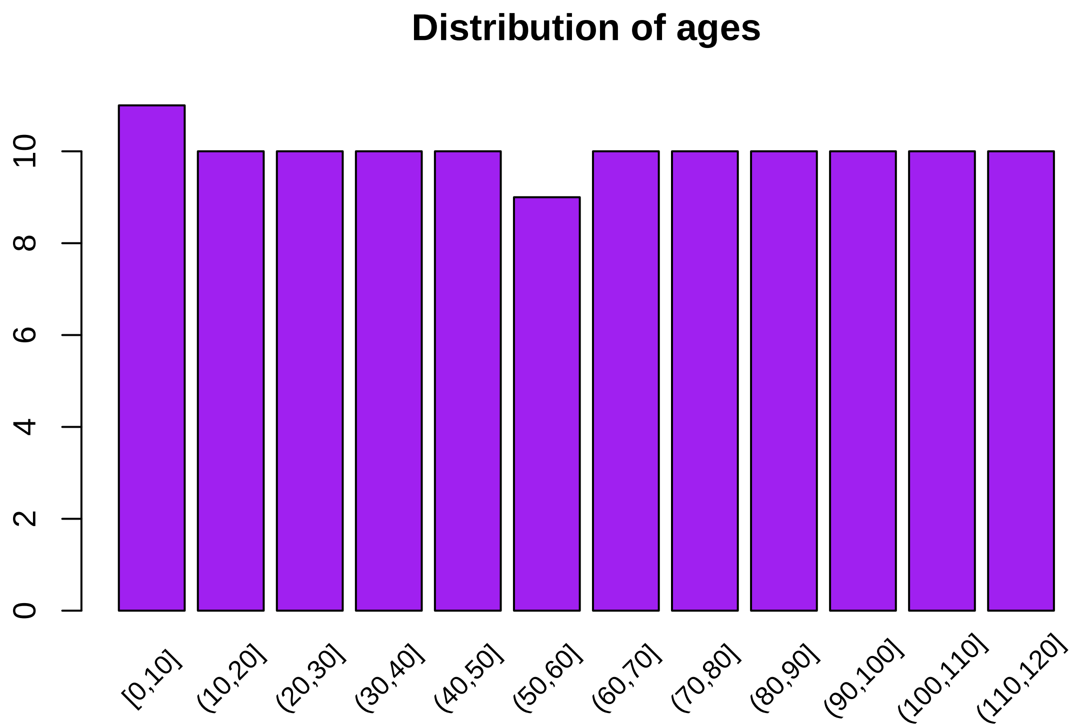
1. Provide the counts by age groups

```r
(counts <- table(df$age_groups))
```

```
##
##    [0,10]   (10,20]   (20,30]   (30,40]   (40,50]   (50,60]   (60,70]   (70,80]
##        11        10        10        10        10         9        10        10
##   (80,90]  (90,100] (100,110] (110,120]
##        10        10        10        10
```

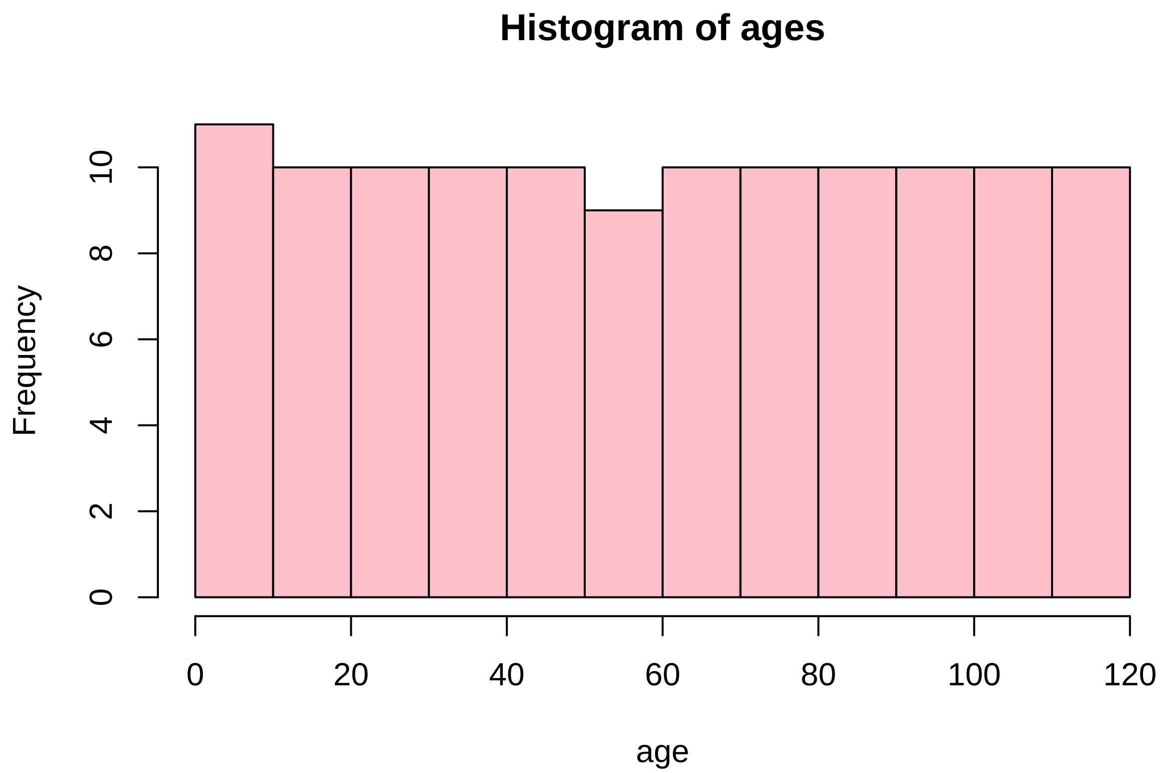
2. Make a `barplot` of age groups

```r
x <- barplot(counts, col = "purple", main = "Distribution of ages", xaxt="n")
text(x = x, y = -1.5, cex = 0.8, xpd=TRUE, srt=45, labels = names(counts))
```

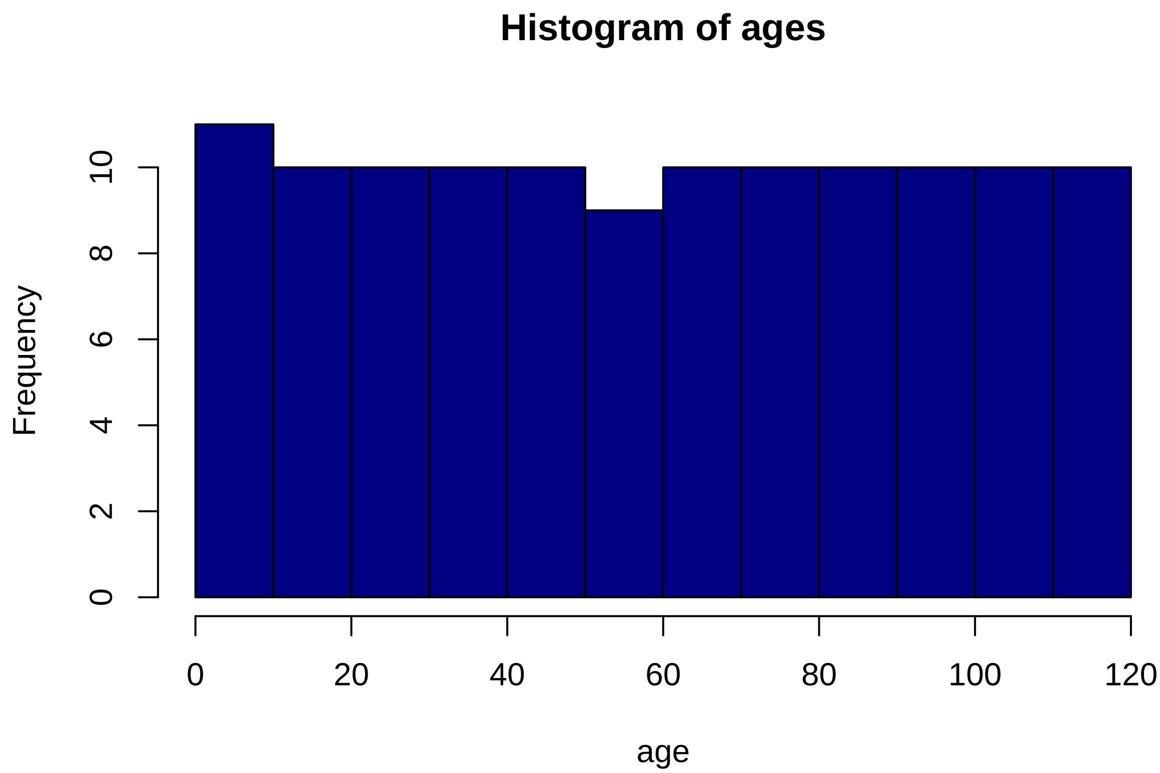


3. Make a histogram of ages

```
hist(df$age, main = "Histogram of ages", xlab = "age", col = "pink", breaks = 12)
```




```r
with(df, hist(age, main = "Histogram of ages", col = "navyblue", breaks = 12)) # alternative
```



4. Display the IDs of participants in the $(70, 80]$ age group.

```
df[df$age_groups == "(70,80]", "ID"]
```

```
##  [1] "Particip_2"  "Particip_21" "Particip_28" "Particip_29" "Particip_31"
##  [6] "Particip_37" "Particip_52" "Particip_83" "Particip_91" "Particip_97"
```

```
df$ID[df$age_groups == "(70,80]"]
```

```
##  [1] "Particip_2"  "Particip_21" "Particip_28" "Particip_29" "Particip_31"
##  [6] "Particip_37" "Particip_52" "Particip_83" "Particip_91" "Particip_97"
```

```
# df[70:80, ]
```

```
l <- split(df, df$age_groups)
names(l)
```

```
##  [1] "[0,10]"    "(10,20]"   "(20,30]"   "(30,40]"   "(40,50]"   "(50,60]"
##  [7] "(60,70]"   "(70,80]"   "(80,90]"   "(90,100]"  "(100,110]" "(110,120]"
```

```
l$`(70,80]`$ID
```

```
##  [1] "Particip_2"  "Particip_21" "Particip_28" "Particip_29" "Particip_31"
##  [6] "Particip_37" "Particip_52" "Particip_83" "Particip_91" "Particip_97"
```

```
df2 <- data.frame(x = rnorm(10), y = rpois(10, 2))
head(df2)
```

Create a data frame using the `data.frame()` function

```
##              x y
## 1 -0.138891362 4
## 2  0.005764186 2
## 3  0.385280401 3
## 4 -0.370660032 1
## 5  0.644376549 3
## 6 -0.220486562 1
```

Data manipulation

- Missing values (`NA`)

```r
x <- c(NA, 1, 2, NA, 3, NA, 3.55)
which(is.na(x)) # means: which of the elements of x are missing
```

```
## [1] 1 4 6
```

```r
which(x >= 2) # means: which of the elements of x are greater than or
```

```
## [1] 3 5 7
```

```r
              # equal to 2.
# which(x != NA) wrong way to check for non-missing values
which(!is.na(x)) # means: which of the elements of x are not missing
```

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

```r
mis_id <- which(is.na(x))
x[mis_id]
```

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

```r
x[is.na(x)] <- mean(x[which(!is.na(x))]) # Good but could be shorter
x[is.na(x)] <- mean(x, na.rm = TRUE)

print(x)
```

```
## [1] 2.3875 1.0000 2.0000 2.3875 3.0000 2.3875 3.5500
```

```r

x <- c(2, 1, 2, 7, "$8", 3, 2.5, 9, "2,7")
print(x)
```

NAs introduced by coercion when converting strings to numeric

```
## [1] "2"   "1"   "2"   "7"   "$8"  "3"   "2.5" "9"   "2,7"
```

```r
class(x)
```

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

```r
y <- as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

```r
print(y)
```

```
## [1] 2.0 1.0 2.0 7.0  NA 3.0 2.5 9.0  NA
```

```r
id <- which(is.na(y))
x[id[1]] <- 8
```

```r
x[id[2]] <- 2.7

y <- as.numeric(x)

# gsub
x <- c(2, 1, 2, ",7", "$8", 3, "&2.5", 9, "2,7")
x <- gsub("\\$|\\&", "", x)
print(x)
```

```
## [1] "2"   "1"   "2"   ",7"  "8"   "3"   "2.5" "9"   "2,7"
```

```r
x <- gsub("\\,", ".", x)

y <- as.numeric(x)
print(y)
```

```
## [1] 2.0 1.0 2.0 0.7 8.0 3.0 2.5 9.0 2.7
```

Outliers detection

- **Inter-Quartile Range (IQR) method**

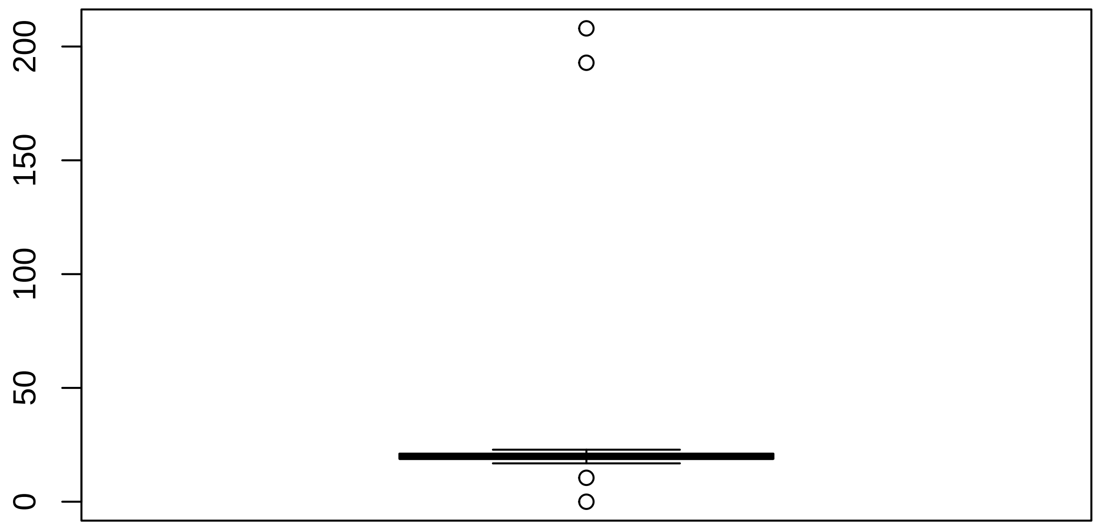
A number outside the following intervall is an outlier

$$[Q_1 - 1.5 \times IQR, Q_3 + 1.5 \times IQR]$$

where

- Q_1 is the first quartile (25% quartile)
- Q_2 is the median or the second quartile (50% quartile)
- Q_3 is the third quartile (75% quartile)
- $IQR = Q_3 - Q_1$

```r
x <- c(19.103, 19.632, 22.494, 20.113, 10.5, 22.744, 20.737, 17.976,
       18.901, 192.87, 21.959, 0.001, 20.641, 20.177, 19.111, 22.859,
       207.97, 16.853, 21.122, 19.244, 18.291, 19.651, 18.358, 18.834, 19)

boxplot(x_copy <- x)
```



Here

```
Q <- as.numeric(quantile(x, probs = c(0.25, 0.5, 0.75)))
d <- as.data.frame(t(Q))
names(d) <- paste0("Q", 1:length(Q))
rownames(d) <- "stats"
d$IQR <- d$Q3 - d$Q1
d$LowerLim <- d$Q1 - 1.5*d$IQR
d$UpperLim <- d$Q3 + 1.5*d$IQR
d
```

```
##           Q1     Q2     Q3   IQR LowerLim UpperLim
## stats 18.834 19.632 21.122 2.288   15.402   24.554
```

Checking for all elements of x if they are outliers or not.

```
(d_out <- data.frame(x = x, outlier = x < d$LowerLim | x > d$UpperLim))
```

```
##          x outlier
## 1   19.103   FALSE
## 2   19.632   FALSE
## 3   22.494   FALSE
## 4   20.113   FALSE
## 5   10.500    TRUE
## 6   22.744   FALSE
## 7   20.737   FALSE
## 8   17.976   FALSE
## 9   18.901   FALSE
```

```
## 10 192.870    TRUE
## 11  21.959   FALSE
## 12   0.001    TRUE
## 13  20.641   FALSE
## 14  20.177   FALSE
## 15  19.111   FALSE
## 16  22.859   FALSE
## 17 207.970    TRUE
## 18  16.853   FALSE
## 19  21.122   FALSE
## 20  19.244   FALSE
## 21  18.291   FALSE
## 22  19.651   FALSE
## 23  18.358   FALSE
## 24  18.834   FALSE
## 25  19.000   FALSE
```

Identifying outliers:

- using the `d_out` table

```
x[d_out$outlier]
```

```
## [1]  10.500 192.870   0.001 207.970
```

- using the `boxplot()` function

```
(bxpt <- boxplot(x, range = 1.5, plot = FALSE)) # we do not plot
```

```
## $stats
##        [,1]
## [1,] 16.853
## [2,] 18.834
## [3,] 19.632
## [4,] 21.122
## [5,] 22.859
##
## $n
## [1] 25
##
## $conf
##          [,1]
## [1,] 18.90899
## [2,] 20.35501
##
## $out
## [1]  10.500 192.870   0.001 207.970
##
## $group
## [1] 1 1 1 1
##
## $names
## [1] "1"
```

The object `bxpt` is a list of 6 elements with a column named `out` which stands for `outliers` (this is the vector of outliers). You all know how to identify and element in a list using `$out` or `[["out"]]`.

```r
x[d_out$outlier] # ours
```

```
## [1]  10.500 192.870   0.001 207.970
```

```r
bxpt$out
```

```
## [1]  10.500 192.870   0.001 207.970
```

```r
bxpt[["out"]]
```

```
## [1]  10.500 192.870   0.001 207.970
```

Now, we can consider all the outliers as missing values (`NA`s). First, we need to locate them.

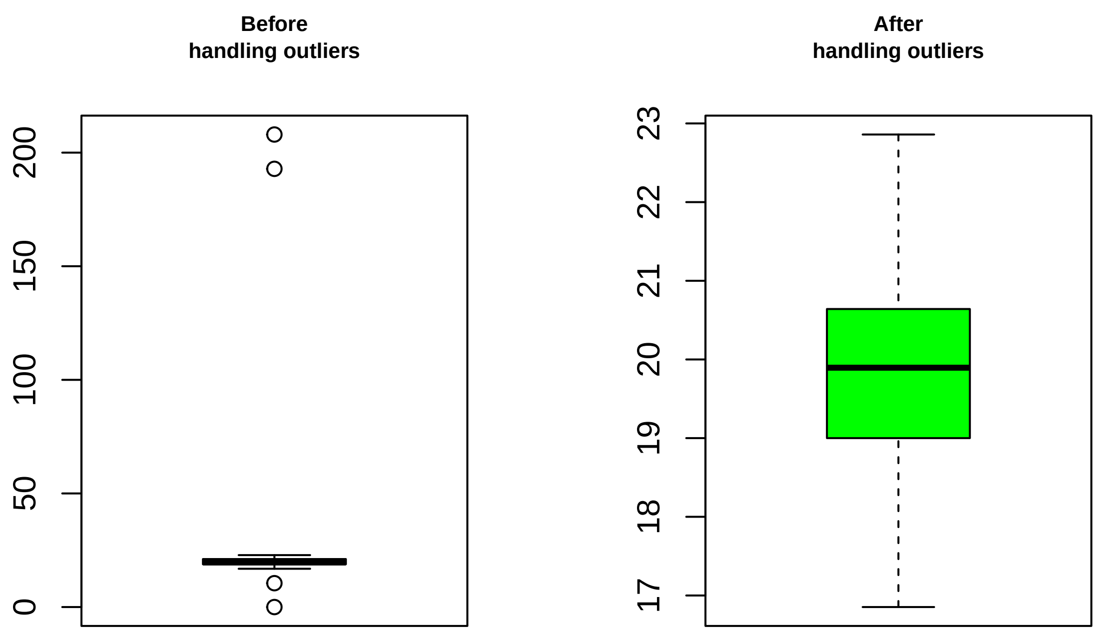
- Method 1

```r
# find all outliers in x
x[d_out$outlier] <- NA # or x[x %in% bxpt$out]
x[d_out$outlier] <- mean(x, na.rm = TRUE)
```

- Method 2

```r
x[d_out$outlier] <- mean(x[!d_out$outlier])
```

```r
# boxplot of imputed x
par(mfrow = c(1, 2))
boxplot(x_copy, main = "Before\nhandling outliers", cex.main = 0.7)
boxplot(x, main = "After\nhandling outliers", col = "green", cex.main = 0.7)
```



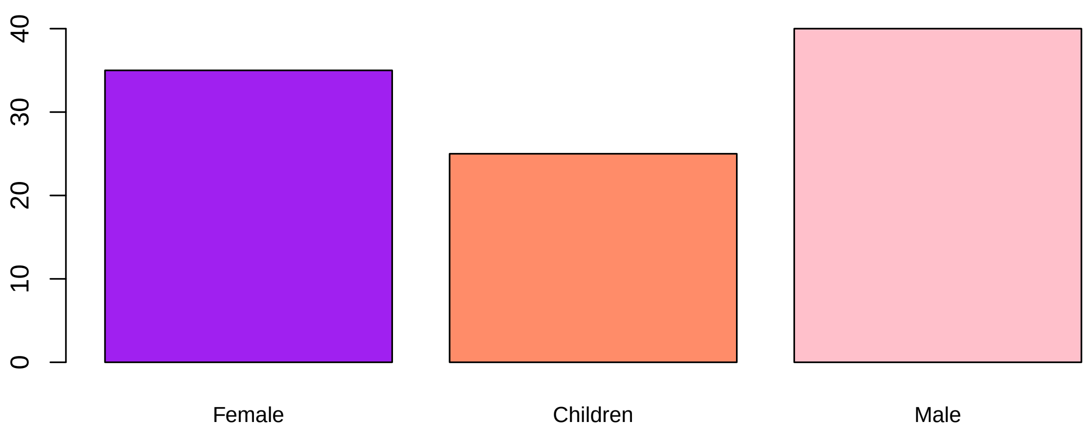
As you can see, there is no more outlier.

Data simulation and visualization

Charts in R

Bar chart/plot

```r
frequencies <- c(Female = 35, Children = 25, Male = 40)
barplot(frequencies, cex.names=0.8,
        names.arg = names(frequencies), # optional
        horiz = FALSE, col = c("purple", "salmon1", "pink"))
```

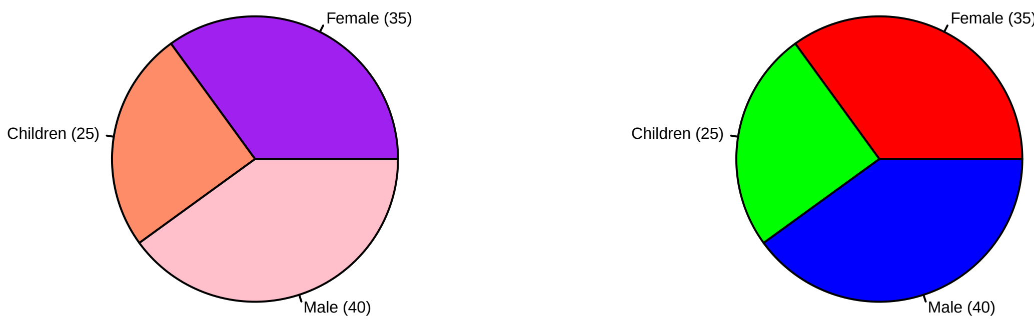



```r
# use horiz = TRUE to have horizontal bars
```

The argument `cex.names` reduces the size of x-labels. Low values, say `cex.names=0.6`, forces R to show all the labels.

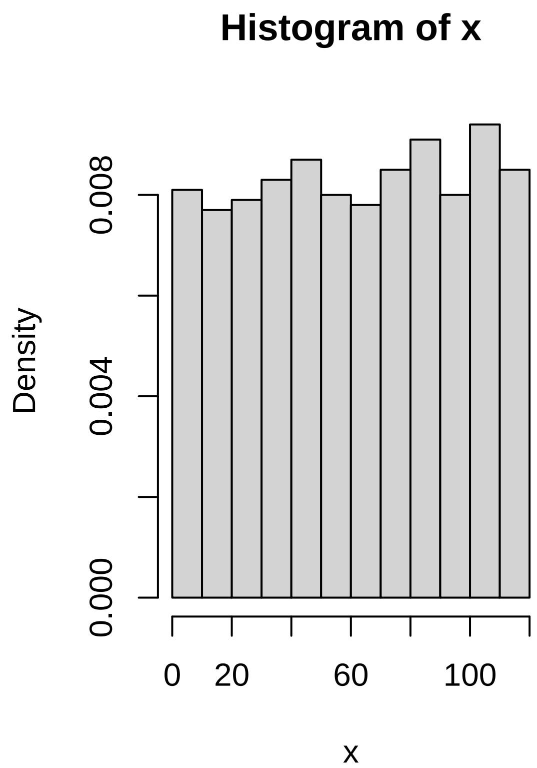
Pie chart/plot

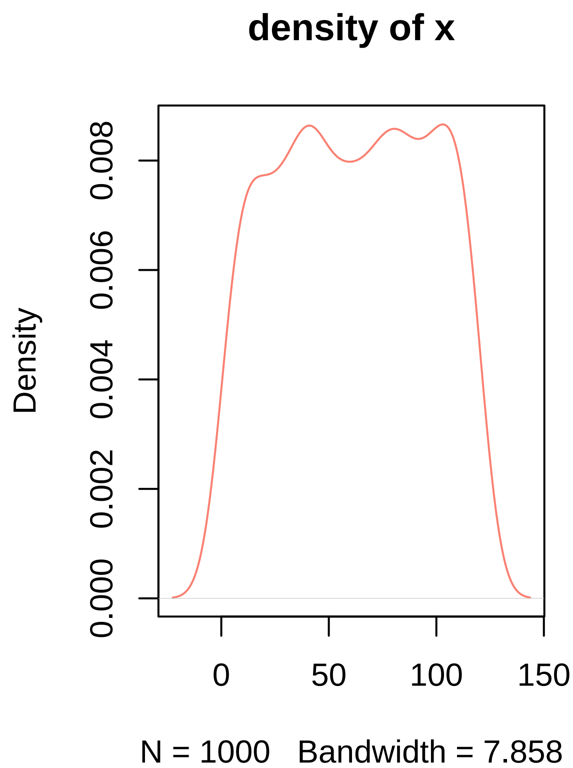
```r
par(mfrow = c(1, 2))
pie(frequencies, cex = 0.5,
    labels = paste0(names(frequencies), " (", frequencies, ")"),
    col = c("purple", "salmon1", "pink"))

pie(frequencies, cex = 0.5,
    labels = paste0(names(frequencies), " (", frequencies, ")"),
    col = rainbow(length(frequencies)))
```



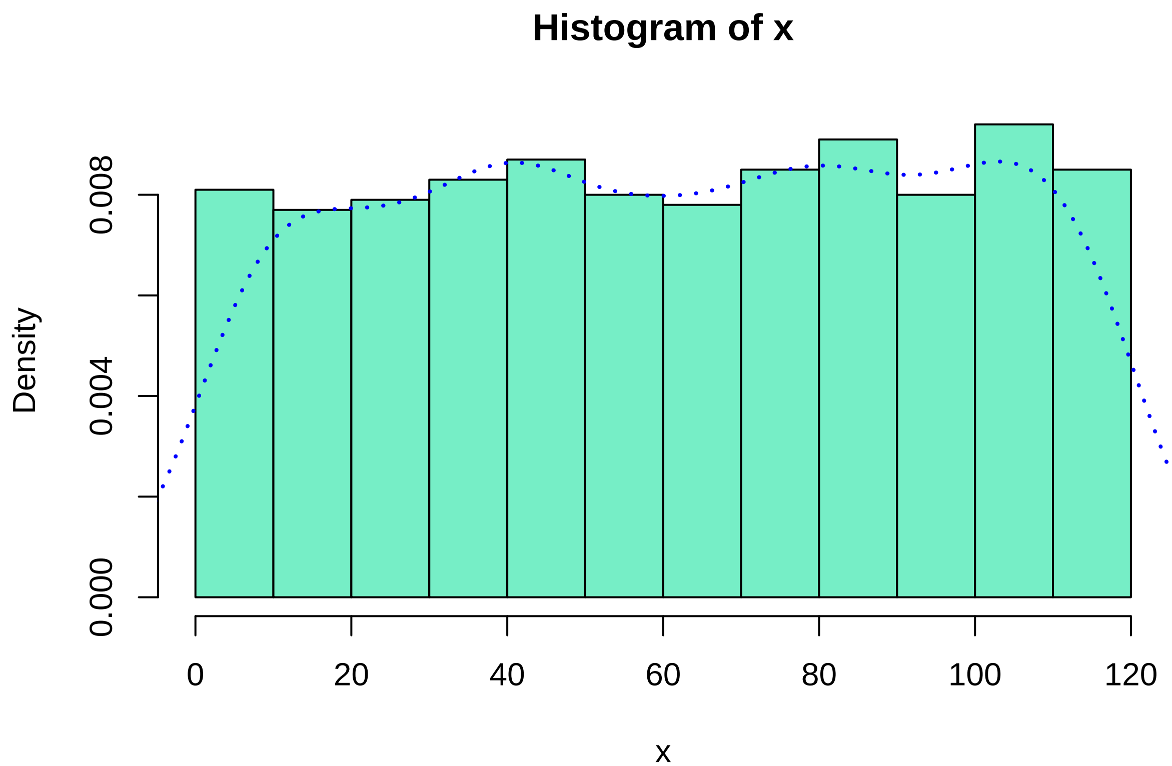
Histograms

```
set.seed(12092024)
x <- sample(1:120, size = 1000, replace = TRUE)
par(mfrow = c(1,2))
hist(x, probability = TRUE) # use probability = TRUE to have densities
        # instead of counts (frequencies)
# Density plots
plot(density(x), col = "salmon", main = "density of x")
```

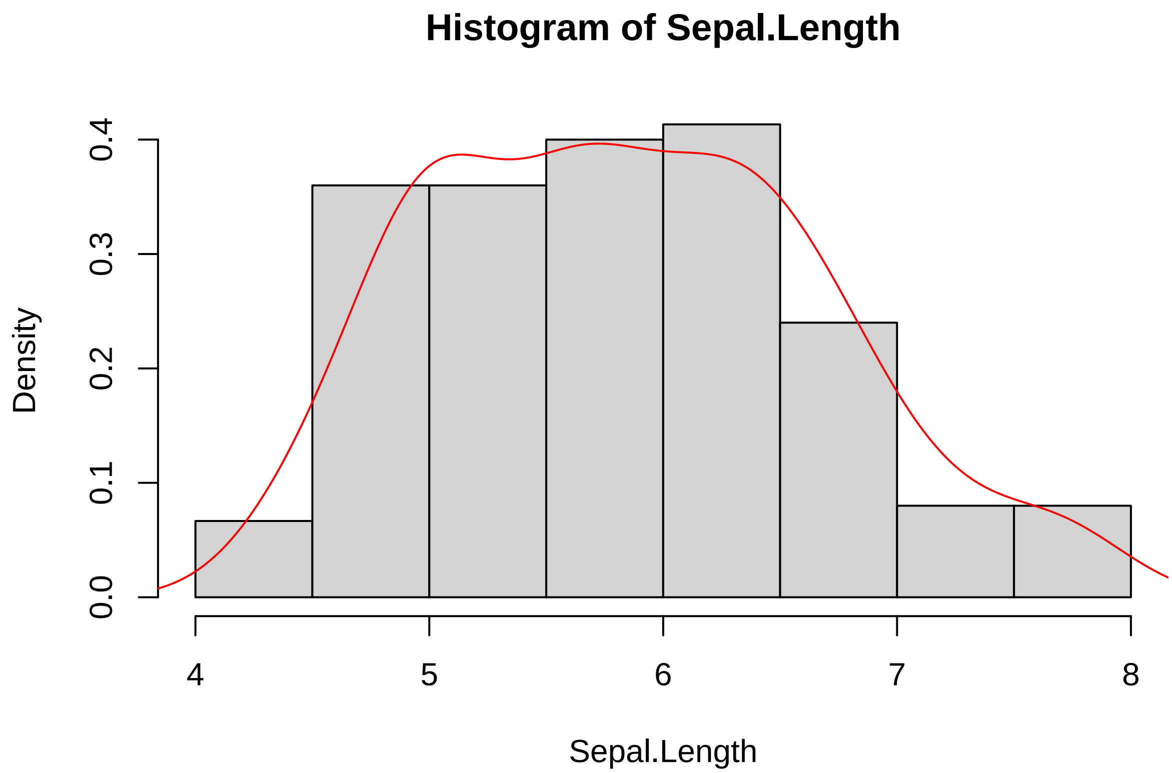




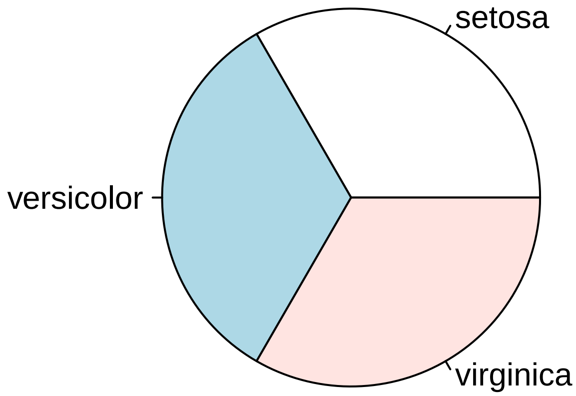

```r
hist(x, probability = TRUE, col = colors()[10])
lines(density(x), col = "blue", lwd = 2, lty = 3)
```




```
with(iris, hist(Sepal.Length, probability = TRUE))
lines(density(iris$Sepal.Length), col = "red")
```

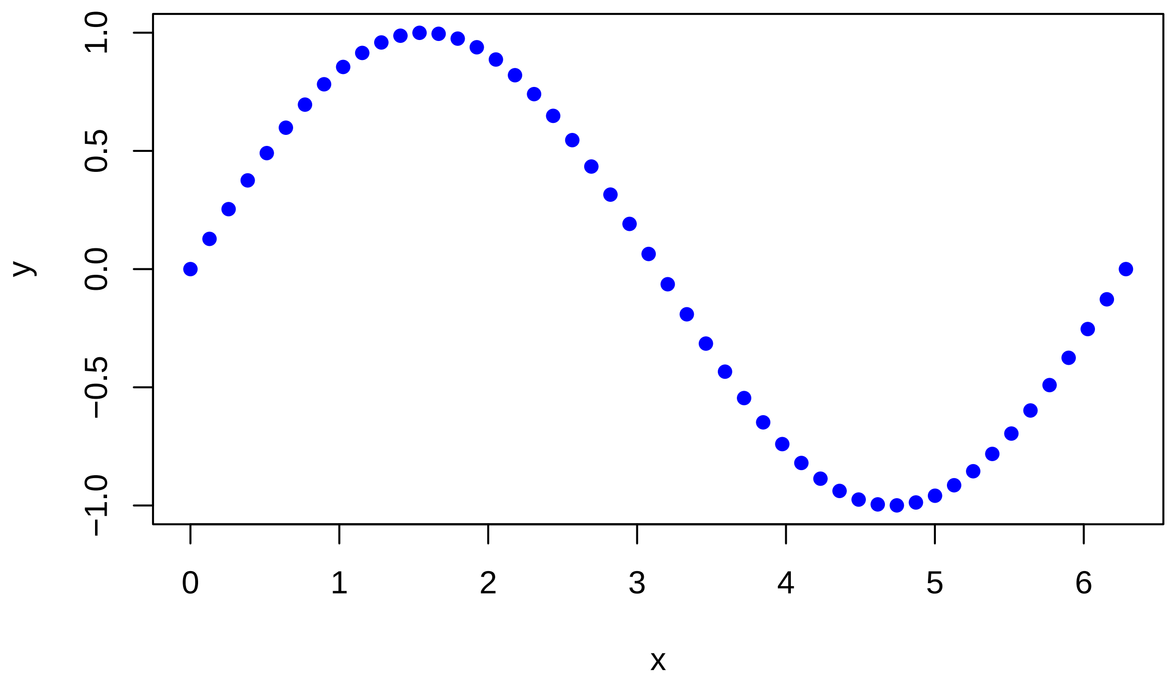



```
pie(table(iris$Species))
```



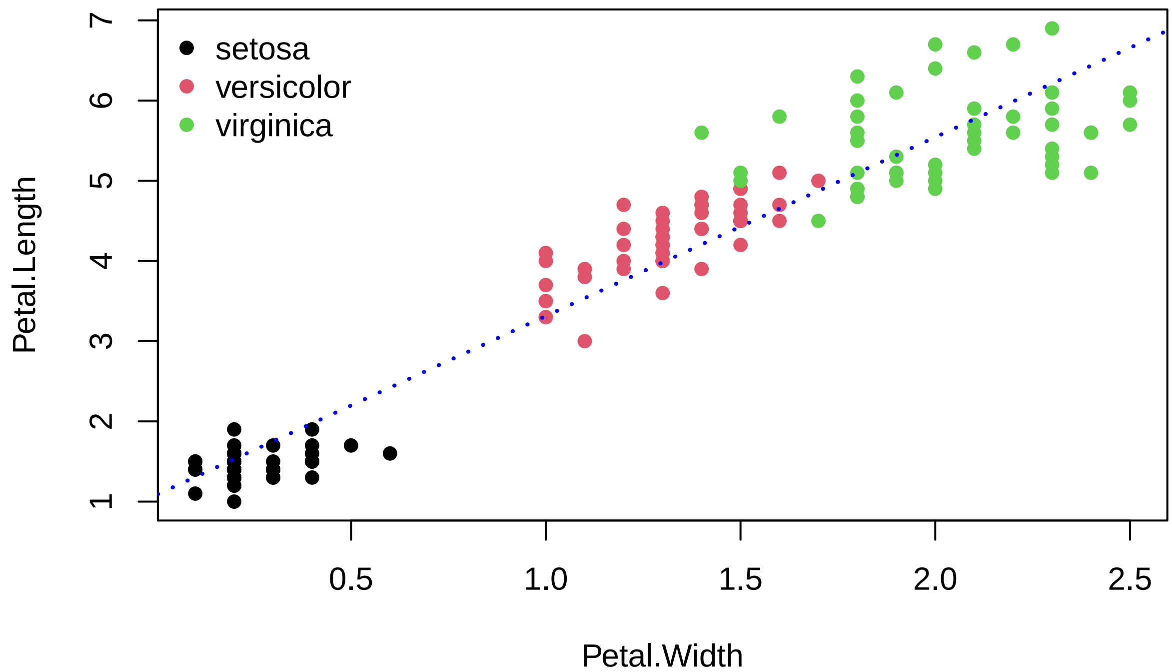
Scatter plot

```
x <- seq(0, 2*pi, le = 50)
y <- sin(x)
# z <- cos(x)
# tg <- tan(x)
plot(x, y, pch = 16, col = "blue")
```



Exercise: Make a scatter plot of `Petal.Length` and `Petal.Width` colored by `Species` using `iris` data frame.

```
f <- Petal.Length ~ Petal.Width
lr <- lm(f, data = iris)
plot(f, data = iris, col = Species, pch = 16)
abline(lr, col = "blue", lwd = 2, lty = 3)
legend("topleft", legend = unique(iris$Species), col = unique(iris$Species), pch = 16, bty = "n")
```



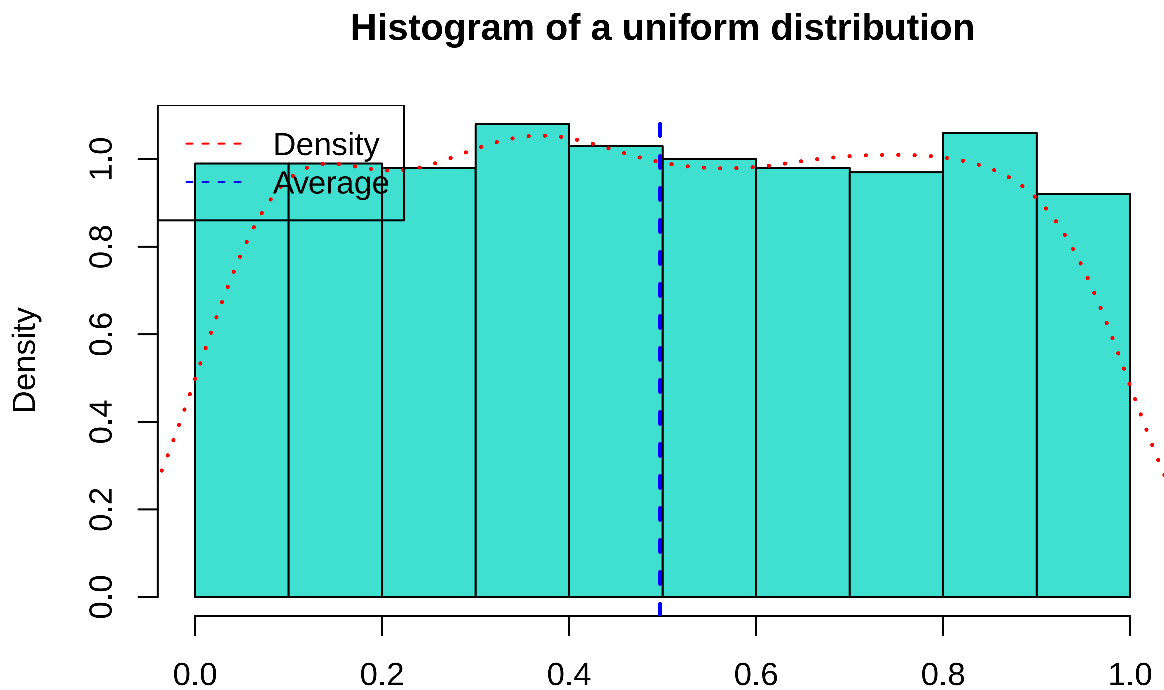
Distribution simulations

Uniform distribution

The p.d.f of the uniform distribution is:

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{else} \end{cases}$$

```
set.seed(123) # just for reproducibility
x <- runif(1000)
hist(x, probability = TRUE, xlab = NULL,
     main = "Histogram of a uniform distribution", col = "turquoise")
# using the lines function draw a line on top of the existing histogram
lines(density(x), col = "red", lwd = 2, lty = 3)
# adding a vertical line that represents the mean (average)
abline(v = mean(x), col = "blue", lty = 2, lwd = 2)
# adding the legend to our plot
legend("topleft", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"))
```



- Statistics

```
# the mean
(mx <- mean(x))
```

```
## [1] 0.4972778
```

```
# variance
(vx <- var(x)) # unbiased variance
```

```
## [1] 0.082647
```

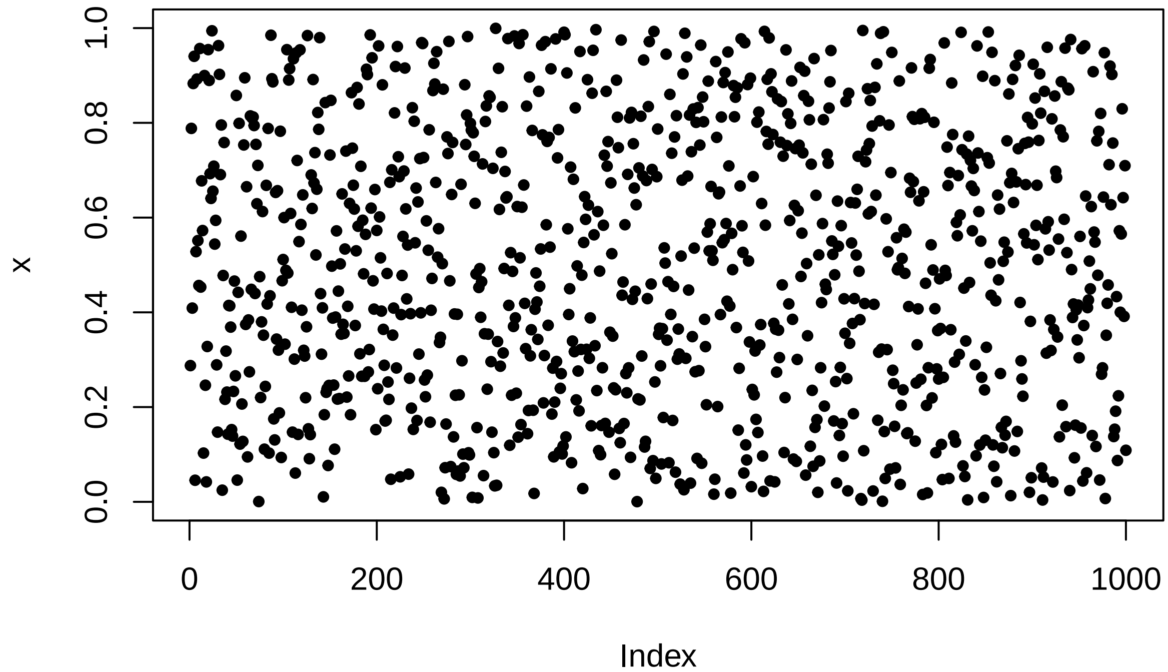
```
mean(x^2) - mx^2
```

```
## [1] 0.08256435
```

```
# standard deviation
sdx <- sd(x)

# coefficient of dispiersion
sdx/mx
```

```
## [1] 0.5781153
```

```
plot(x, pch = 20)
```



Let's test if the uniform distribution is normal.

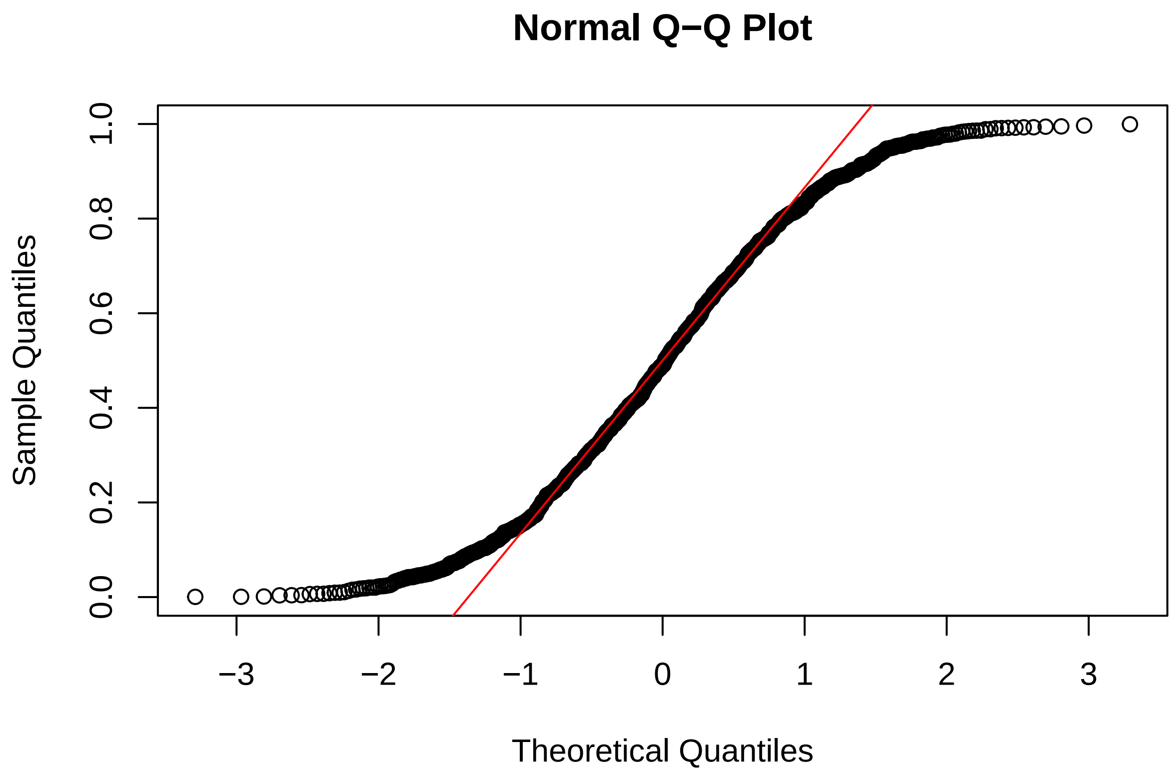
```
# Kolmogorov-Smirnov test
ks.test(x, "pnorm")
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.50019, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
# is the distribution uniform?
ks.test(x, "punif") # check if the p-value > 0.05. If yes, x is uniform.
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.014051, p-value = 0.9891
## alternative hypothesis: two-sided
```

```
# test using plot
qqnorm(x)
qqline(x, col = "red")
```

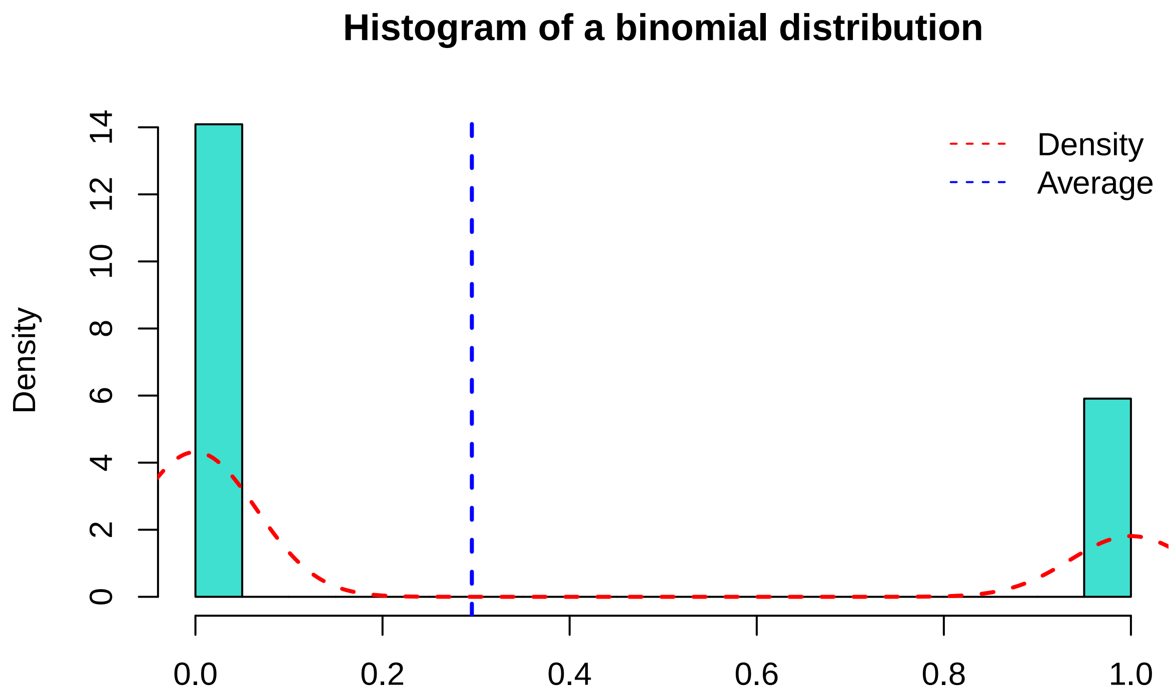


Binomial distribution

A given variable X follows a binomial distribution of parameters n and p if:

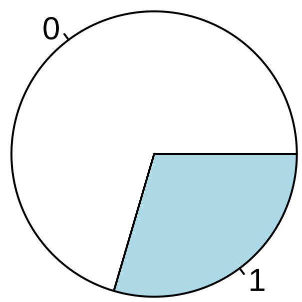
$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k \in \{0, 1\}.$$

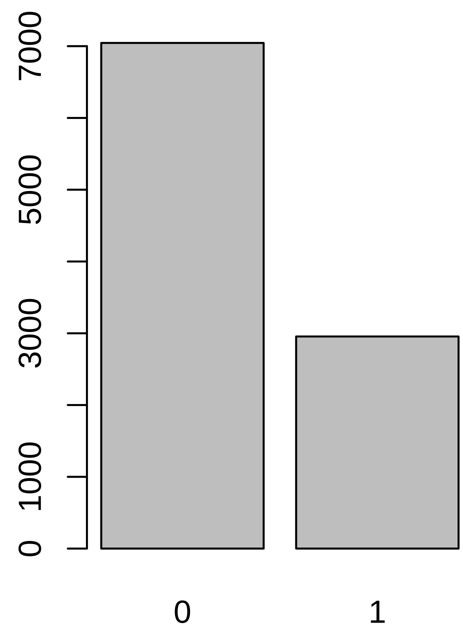
```
rbinom_dist <- rbinom(n = 10000, size = 1, 0.3)
hist(rbinom_dist, probability = TRUE, main = "Histogram of a binomial distribution",
     col = "turquoise", breaks = 20, xlab = NULL)
lines(density(rbinom_dist), col = "red", lwd = 2, lty = 2)
abline(v = mean(rbinom_dist), col = "blue", lty = 2, lwd = 2) # vertical line
legend("topright", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"), bty = "n")
```



A histogram is not appropriate for this distribution. We use the bar charts or pie charts instead.

```
par(mfrow = c(1, 2))
pie(table(rbinom_dist))
barplot(table(rbinom_dist))
```





Gaussian distribution

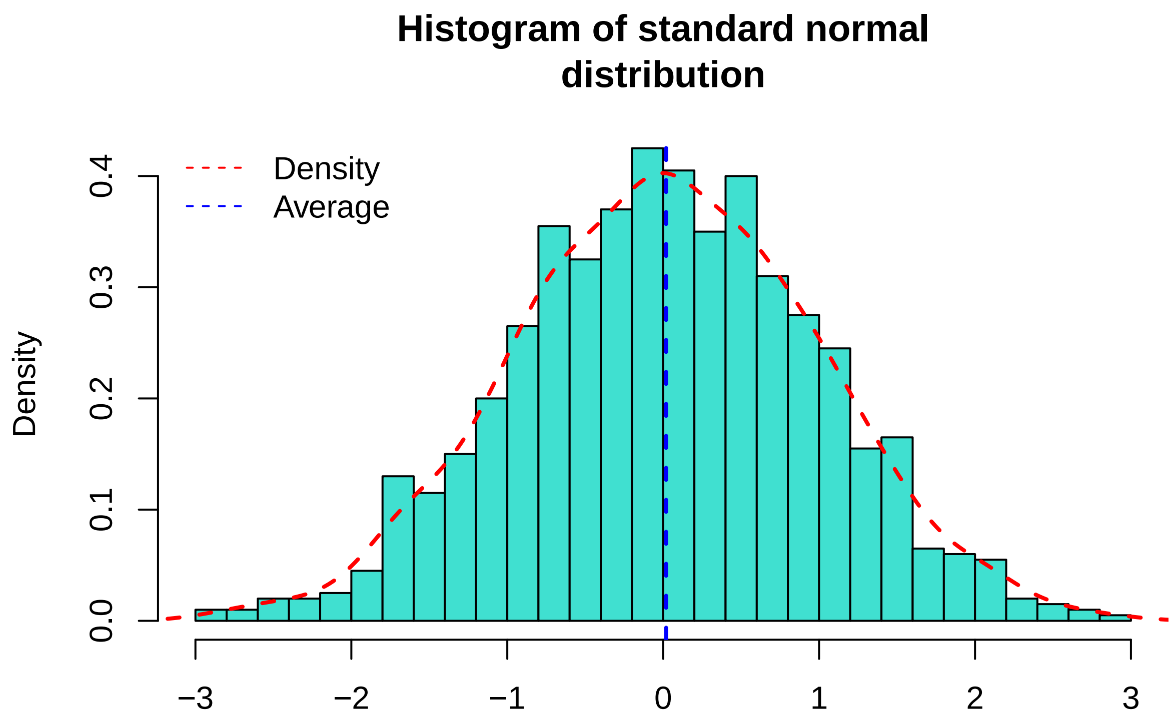
The probability density function for a normal distribution with parameters μ and σ^2 is given by:

LaTeX code:

`f_X(x) = \frac{1}{\sigma^2\sqrt{2\pi}}\exp\left\{-\frac{1}{2\sigma^2}\left(x -\mu\right)^2\right\}, \qua`

$$f_X(x) = \frac{1}{\sigma^2\sqrt{2\pi}}\exp\left\{-\frac{1}{2\sigma^2}\left(x -\mu\right)^2\right\}, \quad x \in R.$$

```r
set.seed(13092024)
gauss_dist <- rnorm(1000, mean = 0, sd = 1)
hist(gauss_dist, probability = TRUE, breaks = 30, xlab = NULL,
     main = "Histogram of standard normal\ndistribution", col = "turquoise")
lines(density(gauss_dist), col = "red", lwd = 2, lty = 2)
abline(v = mean(gauss_dist), col = "blue", lty = 2, lwd = 2)
legend("topleft", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"), bty = "n")
```



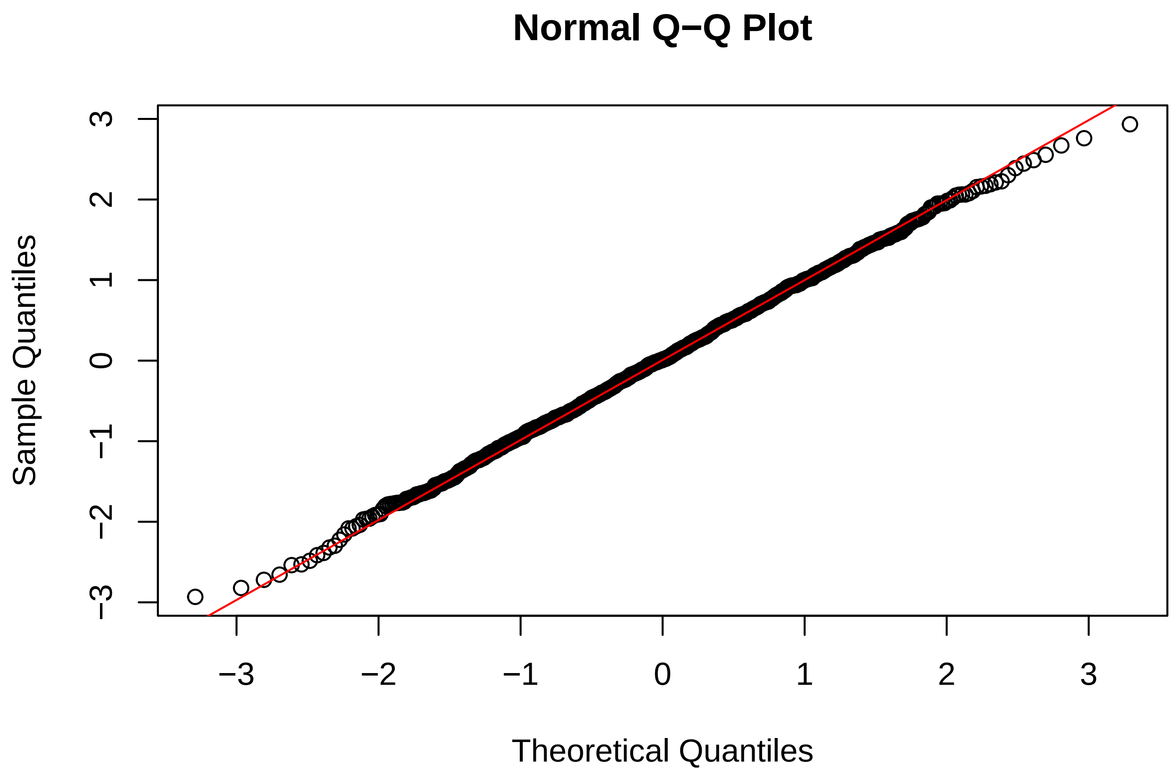

```r
# Kolmogorov-Smirnov test
ks.test(gauss_dist, "pnorm")
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  gauss_dist
## D = 0.019659, p-value = 0.8343
## alternative hypothesis: two-sided
```

```r
# is the distribution uniform?
ks.test(gauss_dist, "punif") # check if the p-value > 0.05. If yes, x is uniform.
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  gauss_dist
## D = 0.49381, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```r
# test using plot
qqnorm(gauss_dist)
qqline(gauss_dist, col = "red")
```



- Statistics

```
mean(gauss_dist)
```

```
## [1] 0.01907579
```

```
sd(gauss_dist)
```

```
## [1] 0.9686031
```

Scatter plot to show relationship between two variables

Equation:

$$y_i = \alpha + \beta x_i + \varepsilon_i, \quad i = 1, 2, \ldots, n$$

where $\varepsilon_i \sim \mathcal{N}(0, 1)$. The coefficients α and β are to be estimated.

Data simulation

```
set.seed(123)
x <- rnorm(1000, mean = 5, sd = 1.5)
y <- rnorm(1000) # another normal distribution.
error_term <- rnorm(1000)
z <- 4 + 8*x + error_term # linear dependence between x and z (this is
# just a simulation. In real-world data analysis the relationships between
# variables are not known in advance).
```

```r
# construction of data frame from x and z
head(df <- data.frame(x, z), 10)
```

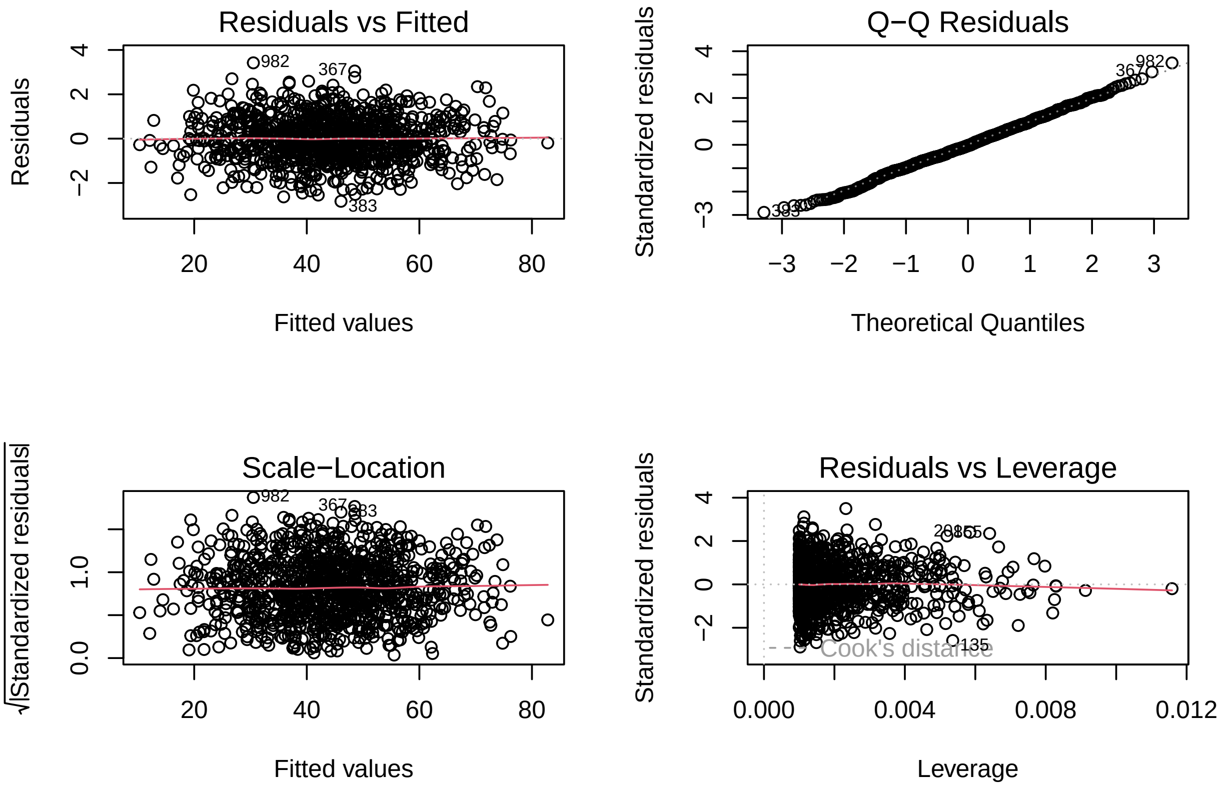
```
##            x        z
## 1  4.159287 36.76269
## 2  4.654734 41.47481
## 3  7.338062 62.16291
## 4  5.105763 46.06533
## 5  5.193932 45.72559
## 6  7.572597 63.96551
## 7  5.691374 47.72410
## 8  3.102408 28.17558
## 9  3.969721 37.80378
## 10 4.331507 38.09129
```

Simple Linear Model

```r
# fitting the model
LR <- lm(z ~ x, data = df)
# the estimated coefficients
coefs <- coefficients(LR)

# summary
summary(LR)
```

```
##
## Call:
## lm(formula = z ~ x, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8254 -0.6397 -0.0310  0.6588  3.4195
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.04376    0.10906   37.08   <2e-16 ***
## x            7.98729    0.02082  383.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9787 on 998 degrees of freedom
## Multiple R-squared:  0.9933, Adjusted R-squared:  0.9933
## F-statistic: 1.472e+05 on 1 and 998 DF,  p-value: < 2.2e-16
```

```r
# model diagnostic
par(mfrow = c(2, 2))
plot(LR)
```



Estimated model: $\hat{y} = 4.0437603 + 7.987287x$. For a new value of x, say $x = 15$, estimated value for y should be: $\hat{y} = 4.0437603 + 4.0437603 \times 15 = 123.8530646$. We use the `predict()` function to make predictions from our model.

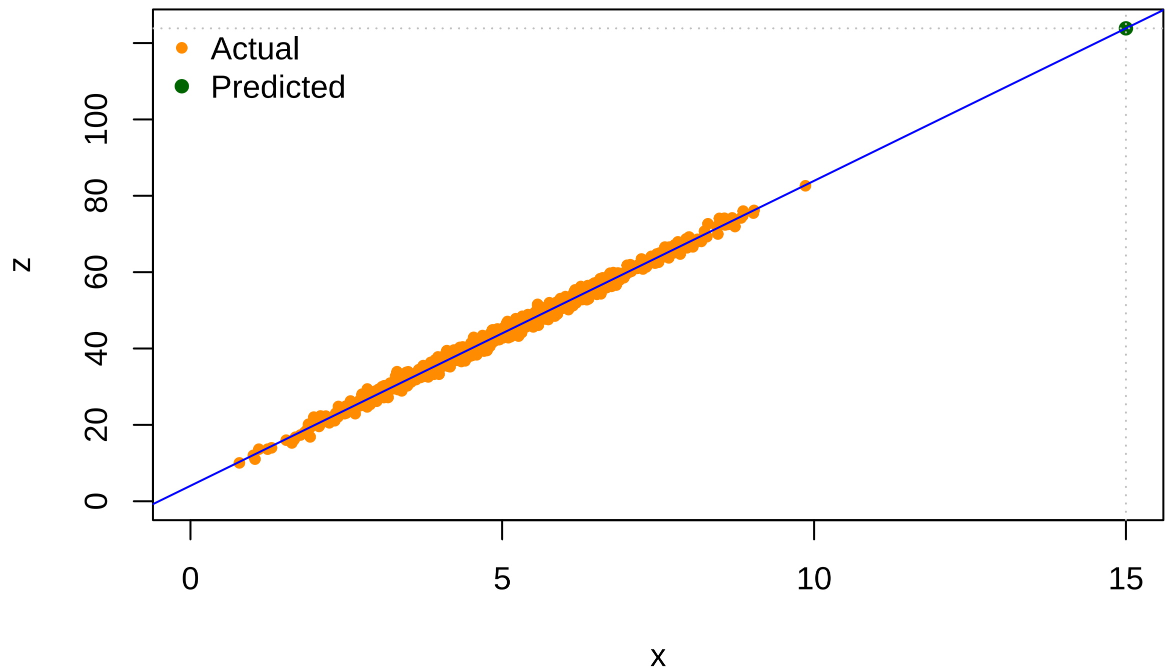
```
# Prediction
new_data <- data.frame(x = 15)
4 + 8*new_data$x # is the theoretical value of y for x = 15
```

```
## [1] 124
```

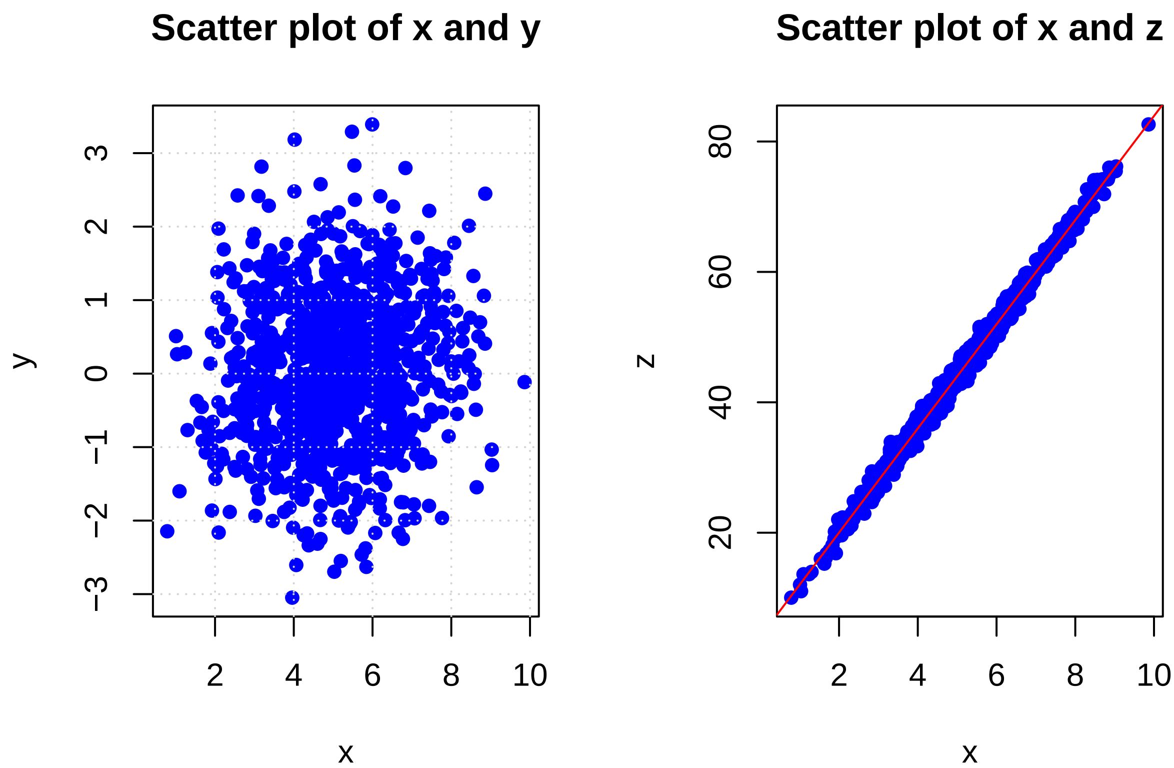
```
(new_data$predicted <- predict(object = LR, newdata = new_data))
```

```
##        1
## 123.8531
```

```
# plot of the predicted value and existing ones
plot(predicted ~ x, data = new_data, col = "darkgreen", xlim = c(0, 15),
     ylim = c(0, new_data$predicted), pch = 16, ylab = "z")
points(z ~ x, data = df, col = "darkorange", pch = 20)
legend("topleft", c("Actual", "Predicted"), col = c("darkorange", "darkgreen"),
       pch = c(20, 16), bty = "n")
abline(LR, col = "blue")
abline(v = new_data$x, h = new_data$predicted, col = "gray", lty = 3)
```




```r
par(mfrow = c(1, 2))
plot(x, y, main = "Scatter plot of x and y", col = "blue", pch = 16); grid()
plot(x, z, main = "Scatter plot of x and z", col = "blue", pch = 16)
abline(LR, col = "red")
```




```
cor(x, z)
```

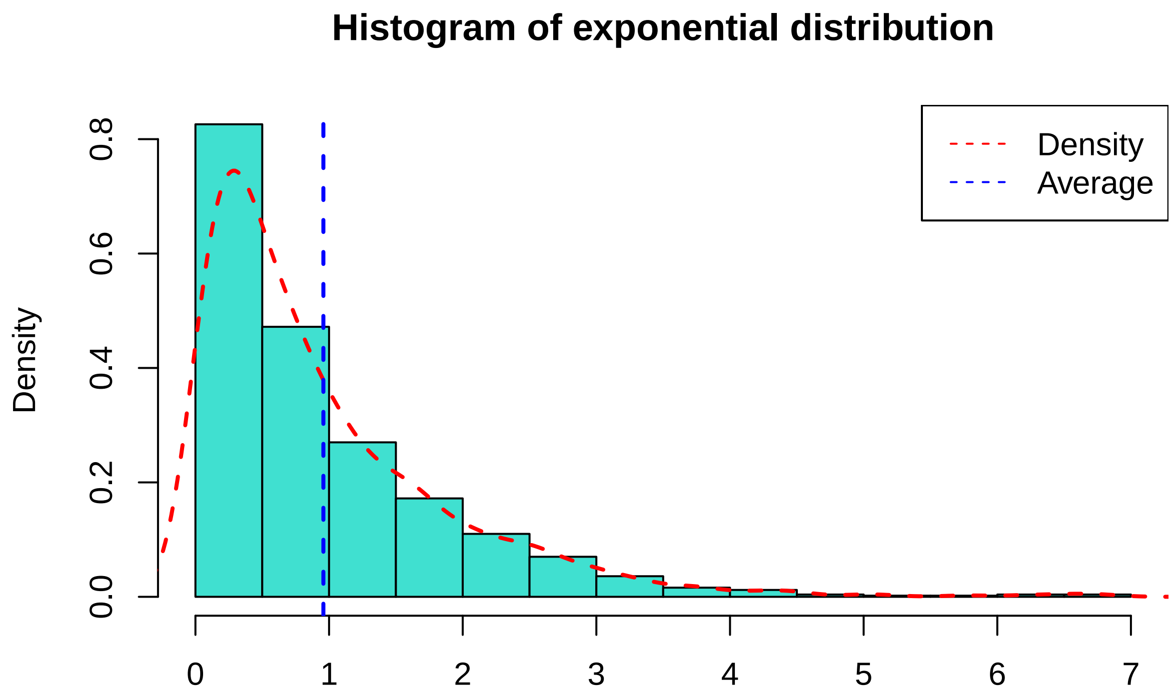
```
## [1] 0.9966282
```

```
cor(x, y)
```

```
## [1] 0.08647944
```

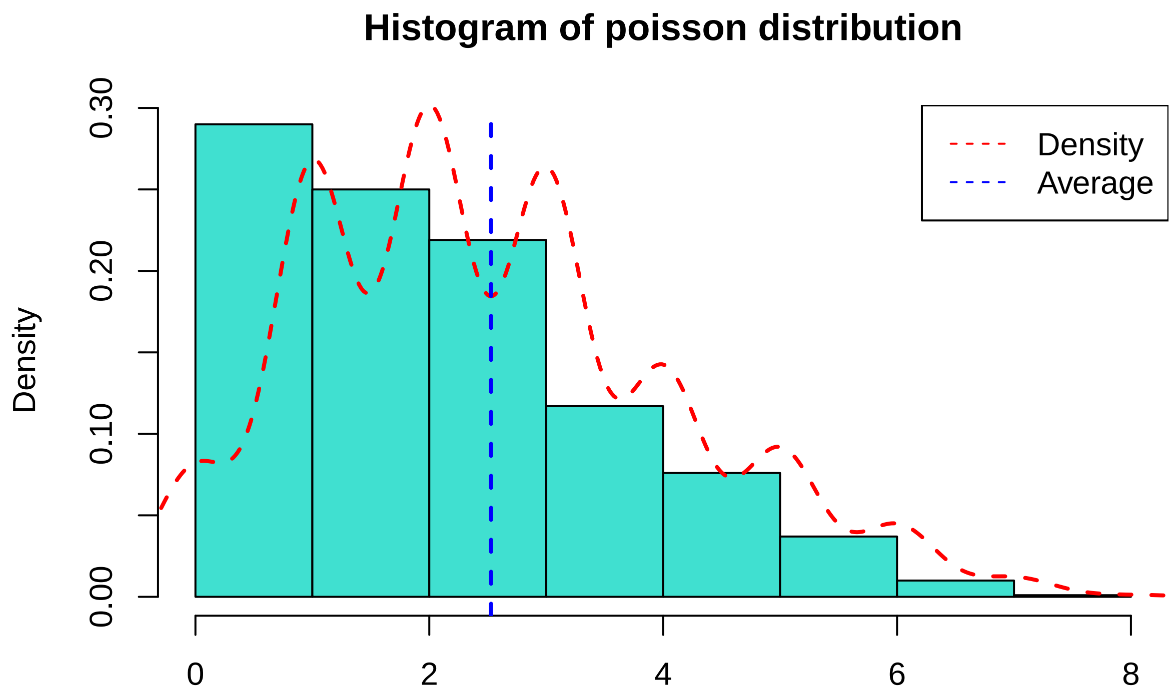
Exponential distribution

```
set.seed(13092024)
exp_dist <- rexp(1000, rate = 1)
hist(exp_dist, probability = TRUE, xlab = NULL,
     main = "Histogram of exponential distribution", col = "turquoise")
lines(density(exp_dist), col = "red", lwd = 2, lty = 2)
abline(v = mean(exp_dist), col = "blue", lty = 2, lwd = 2)
legend("topright", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"))
```



Poisson distribution

```
set.seed(13092024)
pois_dist <- rpois(1000, lambda = 2.5)
hist(pois_dist, probability = TRUE, main = "Histogram of poisson distribution",
     col = "turquoise", xlab = NULL)
lines(density(pois_dist), col = "red", lwd = 2, lty = 2)
abline(v = mean(pois_dist), col = "blue", lty = 2, lwd = 2)
legend("topright", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"))
```



Flow Controls:

if / else

```
if (condition/Boolean expression){
  ## code to be executed
}
```

Example

```r
x <- 3

if (x < 4) {
  print(TRUE)
} else {
  print(FALSE)
}
```

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

```r
# alternative 1
if (x < 4) print(TRUE) else print(FALSE)
```

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

```r
# alternative 2
x <- c(3, 4, 6, 7, 0, 4)
```

```r
ifelse(x < 4, T, F) # this is a vectorized function (can be applied on vectors)
```

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

We can embed `if` to `if` and `else`.

```r
set.seed(123)
marks <- round(rnorm(50, mean = 60, sd = 15), 2)
```

Exercise: write a if statement to check for each value of marks if they fall in the following categories

- $[0-60)$ -> Fail. Here we check if `m < 60` or `marks[i] < 60`.
- $[60-70)$ -> Pass
- $[70-80)$ -> Good Pass
- $[80-85)$ -> Very Good Pass
- ≥ 85 Distinction

1. Taking values from the `marks` directly

```r
category <- character(0)

for (m in marks){
  if (m >= 0 & m < 60){
    cat("Fail ")
    category <- c(category, "Fail") # category.append("Fail") from python
  } else if (m >= 60 & m < 70){
    cat("Pass ")
    category <- c(category, "Pass")
  } else if (m >= 70 & m < 80){
    cat("Good Pass ")
    category <- c(category, "Good Pass")
  } else if (m >= 80 & m < 85){
    cat("Very Good Pass ")
    category <- c(category, "Very Good Pass")
  } else {
    cat("Distinction ")
    category <- c(category, "Distinction")
  }
}
```

```
## Fail Fail Very Good Pass Pass Pass Distinction Pass Fail Fail Fail Good Pass Pass Pass Pass Fail Dis
```

```r
head(df_marks <- data.frame(marks, category), 10)
```

```
##    marks       category
## 1  51.59           Fail
## 2  56.55           Fail
## 3  83.38 Very Good Pass
## 4  61.06           Pass
## 5  61.94           Pass
## 6  85.73    Distinction
## 7  66.91           Pass
## 8  41.02           Fail
## 9  49.70           Fail
## 10 53.32           Fail
```

2. Using indexes

```r
category2 <- character()
for (i in 1:length(marks)){
  if(marks[i] >= 0 & marks[i] < 60){
    cat("F ")
    category2[i] <- "F"
  } else if (marks[i] >= 60 & marks[i] < 70){
    cat("P ")
    category2[i] <- "P"
  } else if (marks[i] >= 70 & marks[i] < 80){
    cat("GP ")
    category2[i] <- "GP"
  }else if (marks[i] >= 80 & marks[i] < 85){
    cat("VGP ")
    category2[i] <- "VGP"
  } else {
    cat("D ")
    category2[i] <- "D"
  }
}
```

```
## F F VGP P P D P F F F GP P P P F D P F GP F F F F F F F GP P F GP P F GP GP GP GP P F F F F F F D GP
```

```r
df_marks$category2 <- category2
head(df_marks, 10)
```

```
##    marks       category category2
## 1  51.59           Fail         F
## 2  56.55           Fail         F
## 3  83.38 Very Good Pass       VGP
## 4  61.06           Pass         P
## 5  61.94           Pass         P
## 6  85.73    Distinction         D
## 7  66.91           Pass         P
## 8  41.02           Fail         F
## 9  49.70           Fail         F
## 10 53.32           Fail         F
```

3. Using `ifelse` (vectorized function)

```r
category3 <- ifelse(marks >= 0 & marks < 60, "Fail",
                    ifelse (marks >= 60 & marks < 70, "Pass",
                            ifelse(marks >= 70 & marks < 80, "Good Pass",
                                   ifelse(marks >= 80 & marks < 85, "Very Good Pass", "Distinction"))))

print(category3)
```

```
##  [1] "Fail"           "Fail"           "Very Good Pass" "Pass"
##  [5] "Pass"           "Distinction"    "Pass"           "Fail"
##  [9] "Fail"           "Fail"           "Good Pass"      "Pass"
## [13] "Pass"           "Pass"           "Fail"           "Distinction"
## [17] "Pass"           "Fail"           "Good Pass"      "Fail"
## [21] "Fail"           "Fail"           "Fail"           "Fail"
## [25] "Fail"           "Fail"           "Good Pass"      "Pass"
## [29] "Fail"           "Good Pass"      "Pass"           "Fail"
## [33] "Good Pass"      "Good Pass"      "Good Pass"      "Good Pass"
## [37] "Pass"           "Fail"           "Fail"           "Fail"
```

```
## [41] "Fail"           "Fail"           "Fail"           "Distinction"
## [45] "Good Pass"      "Fail"           "Fail"           "Fail"
## [49] "Good Pass"      "Fail"
```

```
df_marks$category == category3
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [46] TRUE TRUE TRUE TRUE TRUE
```

Assignment: Write an R function that will take as an input a vector of marks and `return` a data frame of marks and categories (`F`, `P`, `GP`, `VGP`, and `D`). The function should be able make some plots (`pie`, `barplot`).

Loops

- for loops

```
for (i in vector){
  ## code to be executed
}
```

```
m <- 10
for (i in 1:m) print(i)
```

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

```
for (i in 1:10){
  # for each element, let's check if it is a multiple of 3
  if (i %% 3 == 0){
    cat(i, "is a multiple of 3\n")
  } else if (i %% 2 == 0) {
    cat(i, "is a multiple of 2\n")
  } else {
    cat(i, "is neither a multiple of 2 nor 3.\n")
  }
}
```

```
## 1 is neither a multiple of 2 nor 3.
## 2 is a multiple of 2
## 3 is a multiple of 3
## 4 is a multiple of 2
## 5 is neither a multiple of 2 nor 3.
## 6 is a multiple of 3
## 7 is neither a multiple of 2 nor 3.
## 8 is a multiple of 2
## 9 is a multiple of 3
## 10 is a multiple of 2
```

```
for (i in 1:m) {
  print(i)
}
```

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

Exercise 1:

Write a for loop that checks each of the first 10 positive integers if it is odd or even.

```
## TODO
```

Exercise 2:

Using `for loop`, import all `CSV` from the `data_files` folder.

```
# checking the working directory
# getwd()
# simple ls like in bash
dir("./data_list/", pattern = ".csv") # list of elements of in a directory
```

```
## character(0)
```

```
# Exercise: write a for loop to import all
# csv files in a list.
(file_names <- dir("./data_list/", pattern = ".csv"))
```

```
## character(0)
```

Hints: Importing files from the working directory

- We need a path/url when the file to be loaded is not in the working directory.
- We construct a path by combining strings. See the example below.

```
string1 <- "." # working directory (root where the script is saved)
string2 <- "folder" # folder in the working directory
string3 <- "subfolder" # sub-folder in folder
paste(string1, string2, string3, sep = "/")
```

```
## [1] "./folder/subfolder"
```

```
paste0(string1, "/", string2, "/", string3)
```

```
## [1] "./folder/subfolder"
```

Importing files from a folder located in my working directory

while

```
while (condition){
  ## code to be executed
```

```
  # increment
}
```

```r
# Initialize i
i <- 0
while (i <= 10) {
  print(i*2)

  i <- i+10
}
```

```
## [1] 0
## [1] 20
```

Exercises

1. Write a program that will tell the user YOU WON! and exit if they get 5 three times on a row.
2. Write a program that run continuously an ask a user to input a number between 0 and 9 and provide the multiplication table by 2 and asks the user to stop or continue.

Hint: Use the function `readline(prompt = "Enter a number: ")` to interact with the user.

```
number <- readline(prompt = "Entrer un nombre: ") # conversion is needed.
```

repeat

Syntax of the repeat loop:

```
# increment i or anything else
i <- 0

repeat{
  # execute a code

  # increment
  i <- i + 1

  # stopping criteria
  if ( something happens ){
    break # repeat until something happens
  }
}
```

```r
i <- 0
repeat{
  print(i)
  i <- i + 1
  if (i > 10) break # repeat until condition holds.
}
```

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

```
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Apply Functions Over Array Margins

apply

The `apply()` function return a vector or array or list of values obtained by applying a function to margins of an array or matrix.

```
A <- c(1:4)
dim(A) <- c(2, 2)
A
```

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

```
avg <- function(x){
  sum(x)/length(x)
}

v <- 1:10

avg(x = v)
```

```
## [1] 5.5
```

```
# iris[-5]
apply(iris[-5], MARGIN = 2, summary)#/nrow(iris[-5]) # MARGIN = 2 means column-wise
```

```
##         Sepal.Length Sepal.Width Petal.Length Petal.Width
## Min.        4.300000    2.000000        1.000    0.100000
## 1st Qu.     5.100000    2.800000        1.600    0.300000
## Median      5.800000    3.000000        4.350    1.300000
## Mean        5.843333    3.057333        3.758    1.199333
## 3rd Qu.     6.400000    3.300000        5.100    1.800000
## Max.        7.900000    4.400000        6.900    2.500000
```

sapply: use ?sapply to check the documentation.

```
sapply(A, sum) # does not apply for matrices
```

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

Calculating the mean of each from iris data

```
coef_disp <- function(x){ # x is a numeric vector
  sd(x)/mean(x)
}

get_range <- function(x) diff(range(x))

minmax <- function(x) c(min = min(x), max = max(x))
```

```r
sapply(Filter(is.numeric, iris), get_range)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          3.6          2.4          5.9          2.4
```

```r
sapply(Filter(is.numeric, iris), function(x) c(min = min(x),
                                                max = max(x),
                                                std_dev = sd(x),
                                                Q1 = quantile(x, 0.25),
                                                Med = quantile(x, .5),
                                                Q3 = quantile(x, .75),
                                                disp = coef_disp(x)))
```

```
##         Sepal.Length Sepal.Width Petal.Length Petal.Width
## min        4.3000000   2.0000000    1.0000000   0.1000000
## max        7.9000000   4.4000000    6.9000000   2.5000000
## std_dev    0.8280661   0.4358663    1.7652982   0.7622377
## Q1.25%     5.1000000   2.8000000    1.6000000   0.3000000
## Med.50%    5.8000000   3.0000000    4.3500000   1.3000000
## Q3.75%     6.4000000   3.3000000    5.1000000   1.8000000
## disp       0.1417113   0.1425642    0.4697441   0.6355511
```

```r
iris0 <- iris

iris0$Sepal.Length[10] <- NA
sapply(Filter(is.numeric, iris0), mean, na.rm = TRUE)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.849664     3.057333     3.758000     1.199333
```

The sapply function can also return a list if the outputs are not of the same length.

```r
sapply(iris, summary)
```

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

```r
df <- data.frame(replicate(10, rnorm(1000)))
L <- as.list(df) # converting data frame to list.
```

```r
sapply(L, avg)
```

```
##           X1           X2           X3           X4           X5           X6 
##  0.022972504  0.034160787 -0.024801530 -0.008415118 -0.030320654  0.039747511 
##           X7           X8           X9          X10 
## -0.024418261 -0.007606869  0.026690273 -0.048200721
```

```r
sapply(1:10, function(x) x^2)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

lapply:

The `lapply()` function returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X

```r
a <- lapply(iris[-5], mean) # MARGIN = 2 means column-wise
write.csv(a, "a.csv")
unlist(a)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width 
##     5.843333     3.057333     3.758000     1.199333
```

tapply: check the documentation using ?tapply

```r
is.factor(iris$Species) # checking if the column named Species is a factor.
```

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

```r
tapply(iris$Sepal.Length, iris[[5]], mean)
```

```
##     setosa versicolor  virginica 
##      5.006      5.936      6.588
```

vapply: check the documentation

```r
vapply(X = as.list(iris[-5]), quantile, FUN.VALUE =
       c("0%" = 0, "25%" = 0, "50%" = 0, "75%" = 0, "100%" = 0))
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## 0%            4.3         2.0         1.00         0.1
## 25%           5.1         2.8         1.60         0.3
## 50%           5.8         3.0         4.35         1.3
## 75%           6.4         3.3         5.10         1.8
## 100%          7.9         4.4         6.90         2.5
```

Define functions in R

Syntax to write/define a function in R:

```
function_name <- function(arg1, arg2, ...){
  # code to be executed
}
```

```r
toss_coin <- function(){
  face <- sample(6, size = 1)
```

```
  return(face)
}


faces <- c()
for (i in 1:100) {
  faces[i] <- toss_coin()
}
```

```
pp <- function(x) return(x+1)
i <- 1
(i <- pp(i))
```

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

Exercises

1. Write a function that takes an `x` as argument and detects NA then replaces them by the mean

```
replace_missing <- function(x, fun){

}

replace_missing(x, fun = mean)
```

```
## NULL
```

2. Draw the flowchart of the quadratic equation $ax^2 + bx + c = 0$ and write an R function that give solutions and comment according to the values of the discriminant.

Packages

A package is a well documented collection of functions, compiled code and data sets. Packages are created to make specific functionality easy.

How to install a package?

From CRAN (check the lisk of available package)

If a package is not in the `installed.packages()`, matrix of installed packages, one can install it using the command `install.packages("package_name")`.

```
head(data.frame(installed.packages())[c(1, 3:5)], 5)
```

```
##       Package Version Priority
## abind   abind   1.4-8     <NA>
## AER       AER  1.2-14     <NA>
## aion     aion   1.5.0     <NA>
## arkhe   arkhe  1.11.0     <NA>
## ash       ash  1.0-15     <NA>
##                                                                       Depends
## abind                                                             R (>= 1.5.0)
## AER   R (>= 3.0.0), car (>= 2.0-19), lmtest, sandwich (>= 2.4-0),\nsurvival (>= 2.37-5), zoo
## aion                                                                R (>= 3.3)
## arkhe                                                               R (>= 3.5)
## ash                                                                       <NA>
```

Notice that the matrix of installed packages has a column named `Package` that is easily accessible by the command `installed.packages()[,"Package"]` or `rownames(installed.packages())`.

```
all_packages <- installed.packages()[,"Package"]
head(all_packages, 10) # only displaying the first 20 packages by alp. order
```

```
##         abind           AER          aion         arkhe           ash
##       "abind"         "AER"        "aion"       "arkhe"         "ash"
##   AsioHeaders       askpass    assertthat     backports          base
## "AsioHeaders"     "askpass"  "assertthat"   "backports"        "base"
```

Having the list of all packages, we can check if a package is in it.

```
"pacman" %in% all_packages
```

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

It looks like the package is not installed yet. Using if control-flow, we can check if a package is missing and then install it using the `install.package()` function.

```
if (!"pacman" %in% all_packages) {
  install.packages("pacman", repos = "http://cran.us.r-project.org")
}
```

```
##
## The downloaded binary packages are in
##  /var/folders/x6/rdmyg9yd5cq432r1z8p90p6r0000gn/T//RtmpUcuJ8S/downloaded_packages
```

Loading a package using `library()` function from the `base` package.

```
library(pacman)
# library(tidyverse)

plist <- c("kairos", "ggplot2", "tidyverse", "tidyr", "dplyr", "stringr")

pacman::p_load(plist, character.only = TRUE)
# library(mice)
```

R is displaying messages when loading the tidyverse package. You would not want to have it displayed in your report.

Prenvent R from displaying warnings when loading a packages

Do the following setting

`{r warning=FALSE, message=FALSE}`

```
library(pacman)
```

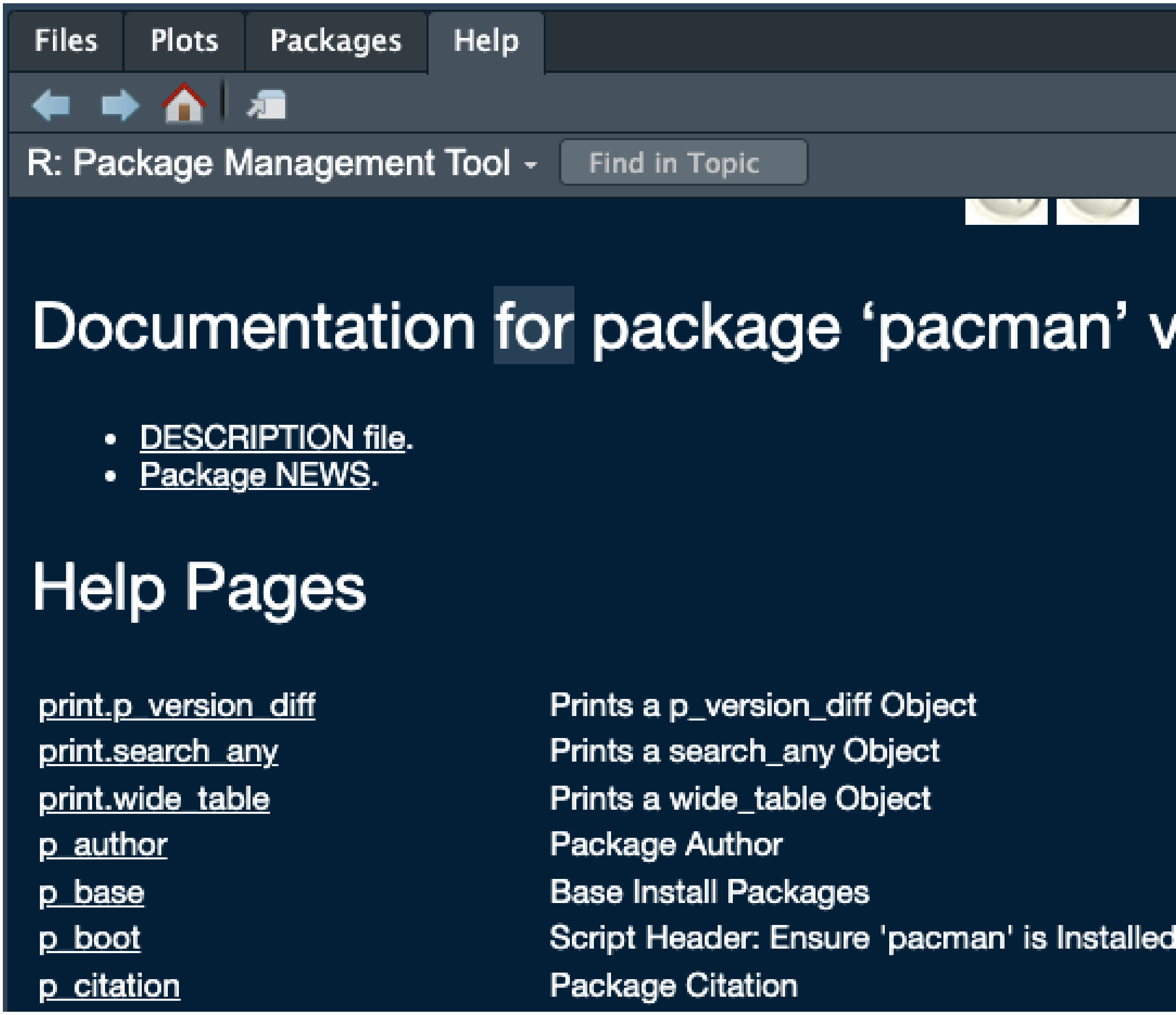
Package documentation

To get the documentation for a specific package that you already installed. Use the command `help(package = "the_package_name")`

Let's get help for the `pacman` package

```
help(package = "pacman")
```

We can have the entire documentation displayed in `File | Plots | Packages | Help` pane.



Functions from a specific package

To access all the functions and data from a given package, we need to load it in R using the `library(the_package)` or `require(the_package)`. The `pacman` package give more flexibility by loading a list of packages and if there any on the list that is not install, `pacman` does the installation for you.

The command to load a list of packages with `pacman` is as follows:

```r
pkg_list <- c("tidyverse", "ggplot2", "lubridate", "flextable", "tictoc")
p_load(pkg_list, character.only = TRUE)
```

Import data in R

Inbuilt data

The iris data set exist already in the R environment. We can import data in R from different sources:

```
# access iris data
data("iris")
data("spam", package = "kernlab")
# displaying the first 6 rows
# help(iris)
# data structure
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# descriptive statistics
summary(iris)
```

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

```
length(colnames(iris))
```

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

```
ncol(iris)
```

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

Checking the classes of all variables in the a given data frame

```
library(tictoc)

dtypes <- c()

tic() # you can also use start <- Sys.time()
for ( i in 1:ncol(iris)){
 dtypes[i] <- class(iris[[i]])
}
toc() # and end <- Sys.time()
```

```
## 0.009 sec elapsed
```

```r
# and calculate the difference between end and start
print(dtypes)
```

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

```r
unique(dtypes)
```

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

from a package without loading it using the `library` function.

```r
data("spam", package = "kernlab")
# data structure
str(spam[1:10])
```

```
## 'data.frame':    4601 obs. of  10 variables:
##  $ make    : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ address : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ all     : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ num3d   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ our     : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
##  $ over    : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ remove  : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ internet: num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ order   : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ mail    : num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
```

Comma Separated Value file

To import a CSV file in R we can use:

- `read.csv()` function from `base` package

```r
dta <- read.csv("./data/iris.csv")
head(dta)
```

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

Notice that the column `Species` is seen as `character`. We can force the conversion by setting the argument `stringsAsFactors` to `TRUE` in `read.csv()`.

- `read_csv()` function from `readr` package already loaded together with `tidyverse`.

```r
dta <- read_csv("./data/iris.csv")
```

```
## Rows: 150 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): Species
## dbl (4): Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
##
```

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(dta)
```

```
## # A tibble: 6 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <chr>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## 3          4.7         3.2          1.3         0.2 setosa
## 4          4.6         3.1          1.5         0.2 setosa
## 5          5           3.6          1.4         0.2 setosa
## 6          5.4         3.9          1.7         0.4 setosa
```

! Always check the errors, warnings and messages to make your report look good.

- Using `import` function from `rio` package that can detect file extension and load it.

```
library(rio)
```

```
##
## Attaching package: 'rio'

## The following object is masked from 'package:dimensio':
##
##     export

## The following object is masked from 'package:aion':
##
##     convert
```

```
head(import("data/iris.xlsx"), 2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

Exercise: import all the csv files in a list using a for loop.

```
data_list <- list() # creating an empty list.
# check the files names in data/csv
dir("./data/csv/")
```

```
## character(0)
```

```
# import
# TODO
```

Pipe

- `%>%` from `magrittr` package or *`|>` from `base` package.
- Library: `tidyverse` or `dplyr`
- Shortcut: `Crtl + Shift + M`
- Why is it useful?

`f(g(h(x)))` is equivalent to `x %>% h() %>% g() %>% f()`

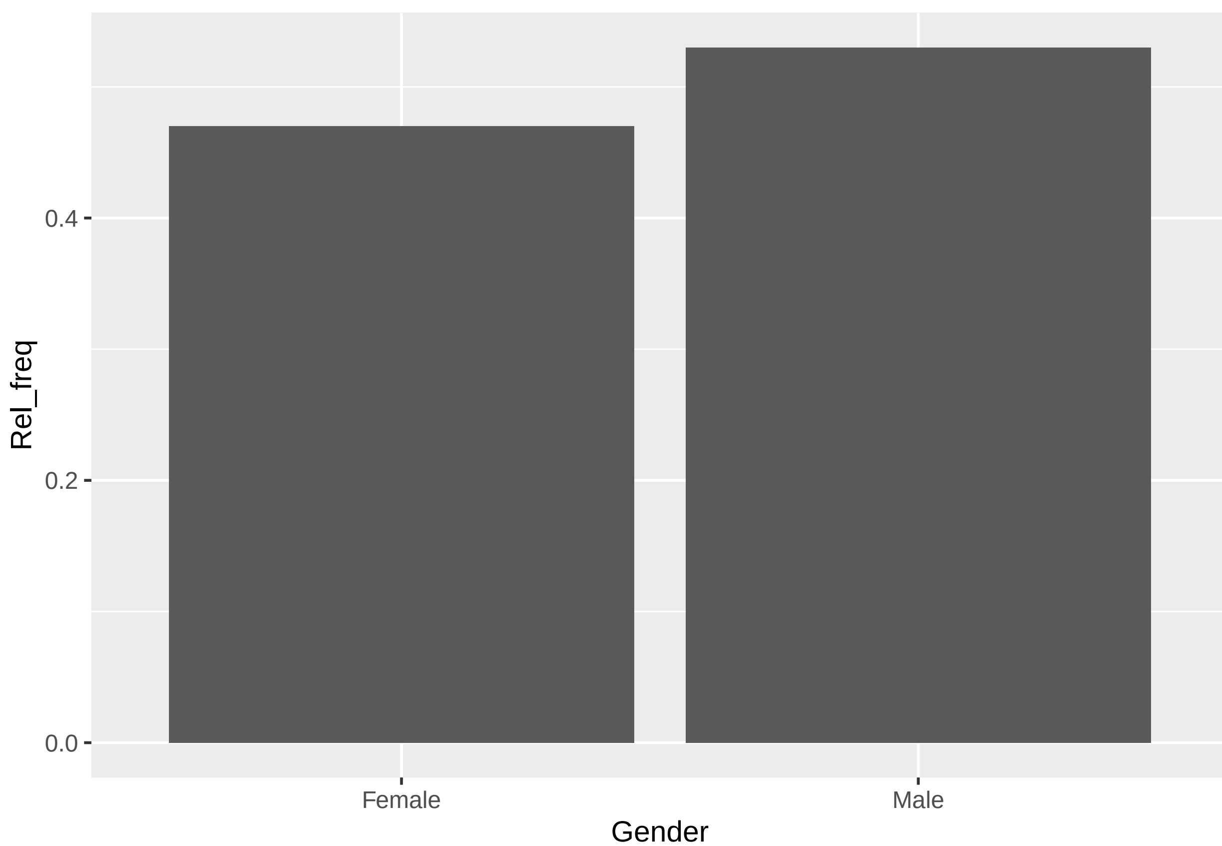
```r
library(dplyr)
library(ggplot2)
iris %>% group_by(Species) %>%
  summarise(mean = mean(Petal.Width))
```

```
## # A tibble: 3 x 2
##   Species     mean
##   <fct>      <dbl>
## 1 setosa     0.246
## 2 versicolor 1.33
## 3 virginica  2.03
```

```r
iris %>% group_by(Species) %>%
  summarise_if(is.numeric, list(mean))
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33
## 3 virginica          6.59        2.97         5.55       2.03
```

```r
Gender %>% table() %>% data.frame() %>%
  rename("Gender" = 1, Count = Freq) %>%
  mutate(Rel_freq = round(Count/sum(Count), 2)) %>%
  # using ggplot to plot
  ggplot(aes(x = Gender, y = Rel_freq)) + geom_col()
```



Instead of

```
summarise(group_by(iris, Species), mean = mean(Petal.Width))
```

```
## # A tibble: 3 x 2
##   Species     mean
##   <fct>      <dbl>
## 1 setosa     0.246
## 2 versicolor 1.33
## 3 virginica  2.03
```

Data manipulation

Data manipulation with tidyverse

When we load `tidyverse`, we load extra packages like: `broom`, `conflicted`, `cli`, `dbplyr`, `dplyr`, `dtplyr`, `forcats`, `ggplot2`, `googledrive`, `googlesheets4`, `haven`, `hms`, `httr`, `jsonlite`, `lubridate`, `magrittr`, `modelr`, `pillar`, `purrr`, `ragg`, `readr`, `readxl`, `reprex`, `rlang`, `rstudioapi`, `rvest`, `stringr`, `tibble`, `tidyr` and `xml2`.

So, it is not necessary to load `tidyverse` and then load any of the dependencies above.

The package

It is part of `tidyverse` package that allows user to perform data manipulation easily using grammar which makes it simpler to use and with the use of pipe, the code is cleaner.

The most used functions from `dplyr` are:

- `select`: Returns a subset of columns from a data frame

```
# consecutive
iris %>% select(1:4)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
## 16           5.7         4.4          1.5         0.4
## 17           5.4         3.9          1.3         0.4
## 18           5.1         3.5          1.4         0.3
## 19           5.7         3.8          1.7         0.3
## 20           5.1         3.8          1.5         0.3
## 21           5.4         3.4          1.7         0.2
## 22           5.1         3.7          1.5         0.4
```

```
## 23          4.6         3.6          1.0         0.2
## 24          5.1         3.3          1.7         0.5
## 25          4.8         3.4          1.9         0.2
## 26          5.0         3.0          1.6         0.2
## 27          5.0         3.4          1.6         0.4
## 28          5.2         3.5          1.5         0.2
## 29          5.2         3.4          1.4         0.2
## 30          4.7         3.2          1.6         0.2
## 31          4.8         3.1          1.6         0.2
## 32          5.4         3.4          1.5         0.4
## 33          5.2         4.1          1.5         0.1
## 34          5.5         4.2          1.4         0.2
## 35          4.9         3.1          1.5         0.2
## 36          5.0         3.2          1.2         0.2
## 37          5.5         3.5          1.3         0.2
## 38          4.9         3.6          1.4         0.1
## 39          4.4         3.0          1.3         0.2
## 40          5.1         3.4          1.5         0.2
## 41          5.0         3.5          1.3         0.3
## 42          4.5         2.3          1.3         0.3
## 43          4.4         3.2          1.3         0.2
## 44          5.0         3.5          1.6         0.6
## 45          5.1         3.8          1.9         0.4
## 46          4.8         3.0          1.4         0.3
## 47          5.1         3.8          1.6         0.2
## 48          4.6         3.2          1.4         0.2
## 49          5.3         3.7          1.5         0.2
## 50          5.0         3.3          1.4         0.2
## 51          7.0         3.2          4.7         1.4
## 52          6.4         3.2          4.5         1.5
## 53          6.9         3.1          4.9         1.5
## 54          5.5         2.3          4.0         1.3
## 55          6.5         2.8          4.6         1.5
## 56          5.7         2.8          4.5         1.3
## 57          6.3         3.3          4.7         1.6
## 58          4.9         2.4          3.3         1.0
## 59          6.6         2.9          4.6         1.3
## 60          5.2         2.7          3.9         1.4
## 61          5.0         2.0          3.5         1.0
## 62          5.9         3.0          4.2         1.5
## 63          6.0         2.2          4.0         1.0
## 64          6.1         2.9          4.7         1.4
## 65          5.6         2.9          3.6         1.3
## 66          6.7         3.1          4.4         1.4
## 67          5.6         3.0          4.5         1.5
## 68          5.8         2.7          4.1         1.0
## 69          6.2         2.2          4.5         1.5
## 70          5.6         2.5          3.9         1.1
## 71          5.9         3.2          4.8         1.8
## 72          6.1         2.8          4.0         1.3
## 73          6.3         2.5          4.9         1.5
## 74          6.1         2.8          4.7         1.2
## 75          6.4         2.9          4.3         1.3
## 76          6.6         3.0          4.4         1.4
```

```
## 77           6.8         2.8          4.8         1.4
## 78           6.7         3.0          5.0         1.7
## 79           6.0         2.9          4.5         1.5
## 80           5.7         2.6          3.5         1.0
## 81           5.5         2.4          3.8         1.1
## 82           5.5         2.4          3.7         1.0
## 83           5.8         2.7          3.9         1.2
## 84           6.0         2.7          5.1         1.6
## 85           5.4         3.0          4.5         1.5
## 86           6.0         3.4          4.5         1.6
## 87           6.7         3.1          4.7         1.5
## 88           6.3         2.3          4.4         1.3
## 89           5.6         3.0          4.1         1.3
## 90           5.5         2.5          4.0         1.3
## 91           5.5         2.6          4.4         1.2
## 92           6.1         3.0          4.6         1.4
## 93           5.8         2.6          4.0         1.2
## 94           5.0         2.3          3.3         1.0
## 95           5.6         2.7          4.2         1.3
## 96           5.7         3.0          4.2         1.2
## 97           5.7         2.9          4.2         1.3
## 98           6.2         2.9          4.3         1.3
## 99           5.1         2.5          3.0         1.1
## 100          5.7         2.8          4.1         1.3
## 101          6.3         3.3          6.0         2.5
## 102          5.8         2.7          5.1         1.9
## 103          7.1         3.0          5.9         2.1
## 104          6.3         2.9          5.6         1.8
## 105          6.5         3.0          5.8         2.2
## 106          7.6         3.0          6.6         2.1
## 107          4.9         2.5          4.5         1.7
## 108          7.3         2.9          6.3         1.8
## 109          6.7         2.5          5.8         1.8
## 110          7.2         3.6          6.1         2.5
## 111          6.5         3.2          5.1         2.0
## 112          6.4         2.7          5.3         1.9
## 113          6.8         3.0          5.5         2.1
## 114          5.7         2.5          5.0         2.0
## 115          5.8         2.8          5.1         2.4
## 116          6.4         3.2          5.3         2.3
## 117          6.5         3.0          5.5         1.8
## 118          7.7         3.8          6.7         2.2
## 119          7.7         2.6          6.9         2.3
## 120          6.0         2.2          5.0         1.5
## 121          6.9         3.2          5.7         2.3
## 122          5.6         2.8          4.9         2.0
## 123          7.7         2.8          6.7         2.0
## 124          6.3         2.7          4.9         1.8
## 125          6.7         3.3          5.7         2.1
## 126          7.2         3.2          6.0         1.8
## 127          6.2         2.8          4.8         1.8
## 128          6.1         3.0          4.9         1.8
## 129          6.4         2.8          5.6         2.1
## 130          7.2         3.0          5.8         1.6
```

```
## 131          7.4         2.8          6.1         1.9
## 132          7.9         3.8          6.4         2.0
## 133          6.4         2.8          5.6         2.2
## 134          6.3         2.8          5.1         1.5
## 135          6.1         2.6          5.6         1.4
## 136          7.7         3.0          6.1         2.3
## 137          6.3         3.4          5.6         2.4
## 138          6.4         3.1          5.5         1.8
## 139          6.0         3.0          4.8         1.8
## 140          6.9         3.1          5.4         2.1
## 141          6.7         3.1          5.6         2.4
## 142          6.9         3.1          5.1         2.3
## 143          5.8         2.7          5.1         1.9
## 144          6.8         3.2          5.9         2.3
## 145          6.7         3.3          5.7         2.5
## 146          6.7         3.0          5.2         2.3
## 147          6.3         2.5          5.0         1.9
## 148          6.5         3.0          5.2         2.0
## 149          6.2         3.4          5.4         2.3
## 150          5.9         3.0          5.1         1.8
```

```
iris %>% select(Sepal.Length:Petal.Width)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
## 16           5.7         4.4          1.5         0.4
## 17           5.4         3.9          1.3         0.4
## 18           5.1         3.5          1.4         0.3
## 19           5.7         3.8          1.7         0.3
## 20           5.1         3.8          1.5         0.3
## 21           5.4         3.4          1.7         0.2
## 22           5.1         3.7          1.5         0.4
## 23           4.6         3.6          1.0         0.2
## 24           5.1         3.3          1.7         0.5
## 25           4.8         3.4          1.9         0.2
## 26           5.0         3.0          1.6         0.2
## 27           5.0         3.4          1.6         0.4
## 28           5.2         3.5          1.5         0.2
## 29           5.2         3.4          1.4         0.2
## 30           4.7         3.2          1.6         0.2
## 31           4.8         3.1          1.6         0.2
```

```
## 32          5.4         3.4          1.5         0.4
## 33          5.2         4.1          1.5         0.1
## 34          5.5         4.2          1.4         0.2
## 35          4.9         3.1          1.5         0.2
## 36          5.0         3.2          1.2         0.2
## 37          5.5         3.5          1.3         0.2
## 38          4.9         3.6          1.4         0.1
## 39          4.4         3.0          1.3         0.2
## 40          5.1         3.4          1.5         0.2
## 41          5.0         3.5          1.3         0.3
## 42          4.5         2.3          1.3         0.3
## 43          4.4         3.2          1.3         0.2
## 44          5.0         3.5          1.6         0.6
## 45          5.1         3.8          1.9         0.4
## 46          4.8         3.0          1.4         0.3
## 47          5.1         3.8          1.6         0.2
## 48          4.6         3.2          1.4         0.2
## 49          5.3         3.7          1.5         0.2
## 50          5.0         3.3          1.4         0.2
## 51          7.0         3.2          4.7         1.4
## 52          6.4         3.2          4.5         1.5
## 53          6.9         3.1          4.9         1.5
## 54          5.5         2.3          4.0         1.3
## 55          6.5         2.8          4.6         1.5
## 56          5.7         2.8          4.5         1.3
## 57          6.3         3.3          4.7         1.6
## 58          4.9         2.4          3.3         1.0
## 59          6.6         2.9          4.6         1.3
## 60          5.2         2.7          3.9         1.4
## 61          5.0         2.0          3.5         1.0
## 62          5.9         3.0          4.2         1.5
## 63          6.0         2.2          4.0         1.0
## 64          6.1         2.9          4.7         1.4
## 65          5.6         2.9          3.6         1.3
## 66          6.7         3.1          4.4         1.4
## 67          5.6         3.0          4.5         1.5
## 68          5.8         2.7          4.1         1.0
## 69          6.2         2.2          4.5         1.5
## 70          5.6         2.5          3.9         1.1
## 71          5.9         3.2          4.8         1.8
## 72          6.1         2.8          4.0         1.3
## 73          6.3         2.5          4.9         1.5
## 74          6.1         2.8          4.7         1.2
## 75          6.4         2.9          4.3         1.3
## 76          6.6         3.0          4.4         1.4
## 77          6.8         2.8          4.8         1.4
## 78          6.7         3.0          5.0         1.7
## 79          6.0         2.9          4.5         1.5
## 80          5.7         2.6          3.5         1.0
## 81          5.5         2.4          3.8         1.1
## 82          5.5         2.4          3.7         1.0
## 83          5.8         2.7          3.9         1.2
## 84          6.0         2.7          5.1         1.6
## 85          5.4         3.0          4.5         1.5
```

```
## 86            6.0          3.4          4.5          1.6
## 87            6.7          3.1          4.7          1.5
## 88            6.3          2.3          4.4          1.3
## 89            5.6          3.0          4.1          1.3
## 90            5.5          2.5          4.0          1.3
## 91            5.5          2.6          4.4          1.2
## 92            6.1          3.0          4.6          1.4
## 93            5.8          2.6          4.0          1.2
## 94            5.0          2.3          3.3          1.0
## 95            5.6          2.7          4.2          1.3
## 96            5.7          3.0          4.2          1.2
## 97            5.7          2.9          4.2          1.3
## 98            6.2          2.9          4.3          1.3
## 99            5.1          2.5          3.0          1.1
## 100           5.7          2.8          4.1          1.3
## 101           6.3          3.3          6.0          2.5
## 102           5.8          2.7          5.1          1.9
## 103           7.1          3.0          5.9          2.1
## 104           6.3          2.9          5.6          1.8
## 105           6.5          3.0          5.8          2.2
## 106           7.6          3.0          6.6          2.1
## 107           4.9          2.5          4.5          1.7
## 108           7.3          2.9          6.3          1.8
## 109           6.7          2.5          5.8          1.8
## 110           7.2          3.6          6.1          2.5
## 111           6.5          3.2          5.1          2.0
## 112           6.4          2.7          5.3          1.9
## 113           6.8          3.0          5.5          2.1
## 114           5.7          2.5          5.0          2.0
## 115           5.8          2.8          5.1          2.4
## 116           6.4          3.2          5.3          2.3
## 117           6.5          3.0          5.5          1.8
## 118           7.7          3.8          6.7          2.2
## 119           7.7          2.6          6.9          2.3
## 120           6.0          2.2          5.0          1.5
## 121           6.9          3.2          5.7          2.3
## 122           5.6          2.8          4.9          2.0
## 123           7.7          2.8          6.7          2.0
## 124           6.3          2.7          4.9          1.8
## 125           6.7          3.3          5.7          2.1
## 126           7.2          3.2          6.0          1.8
## 127           6.2          2.8          4.8          1.8
## 128           6.1          3.0          4.9          1.8
## 129           6.4          2.8          5.6          2.1
## 130           7.2          3.0          5.8          1.6
## 131           7.4          2.8          6.1          1.9
## 132           7.9          3.8          6.4          2.0
## 133           6.4          2.8          5.6          2.2
## 134           6.3          2.8          5.1          1.5
## 135           6.1          2.6          5.6          1.4
## 136           7.7          3.0          6.1          2.3
## 137           6.3          3.4          5.6          2.4
## 138           6.4          3.1          5.5          1.8
## 139           6.0          3.0          4.8          1.8
```

```
## 140          6.9         3.1          5.4         2.1
## 141          6.7         3.1          5.6         2.4
## 142          6.9         3.1          5.1         2.3
## 143          5.8         2.7          5.1         1.9
## 144          6.8         3.2          5.9         2.3
## 145          6.7         3.3          5.7         2.5
## 146          6.7         3.0          5.2         2.3
## 147          6.3         2.5          5.0         1.9
## 148          6.5         3.0          5.2         2.0
## 149          6.2         3.4          5.4         2.3
## 150          5.9         3.0          5.1         1.8
```

```r
# Exclusion
iris %>% select(-5)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
## 16           5.7         4.4          1.5         0.4
## 17           5.4         3.9          1.3         0.4
## 18           5.1         3.5          1.4         0.3
## 19           5.7         3.8          1.7         0.3
## 20           5.1         3.8          1.5         0.3
## 21           5.4         3.4          1.7         0.2
## 22           5.1         3.7          1.5         0.4
## 23           4.6         3.6          1.0         0.2
## 24           5.1         3.3          1.7         0.5
## 25           4.8         3.4          1.9         0.2
## 26           5.0         3.0          1.6         0.2
## 27           5.0         3.4          1.6         0.4
## 28           5.2         3.5          1.5         0.2
## 29           5.2         3.4          1.4         0.2
## 30           4.7         3.2          1.6         0.2
## 31           4.8         3.1          1.6         0.2
## 32           5.4         3.4          1.5         0.4
## 33           5.2         4.1          1.5         0.1
## 34           5.5         4.2          1.4         0.2
## 35           4.9         3.1          1.5         0.2
## 36           5.0         3.2          1.2         0.2
## 37           5.5         3.5          1.3         0.2
## 38           4.9         3.6          1.4         0.1
## 39           4.4         3.0          1.3         0.2
```

```
## 40          5.1         3.4          1.5         0.2
## 41          5.0         3.5          1.3         0.3
## 42          4.5         2.3          1.3         0.3
## 43          4.4         3.2          1.3         0.2
## 44          5.0         3.5          1.6         0.6
## 45          5.1         3.8          1.9         0.4
## 46          4.8         3.0          1.4         0.3
## 47          5.1         3.8          1.6         0.2
## 48          4.6         3.2          1.4         0.2
## 49          5.3         3.7          1.5         0.2
## 50          5.0         3.3          1.4         0.2
## 51          7.0         3.2          4.7         1.4
## 52          6.4         3.2          4.5         1.5
## 53          6.9         3.1          4.9         1.5
## 54          5.5         2.3          4.0         1.3
## 55          6.5         2.8          4.6         1.5
## 56          5.7         2.8          4.5         1.3
## 57          6.3         3.3          4.7         1.6
## 58          4.9         2.4          3.3         1.0
## 59          6.6         2.9          4.6         1.3
## 60          5.2         2.7          3.9         1.4
## 61          5.0         2.0          3.5         1.0
## 62          5.9         3.0          4.2         1.5
## 63          6.0         2.2          4.0         1.0
## 64          6.1         2.9          4.7         1.4
## 65          5.6         2.9          3.6         1.3
## 66          6.7         3.1          4.4         1.4
## 67          5.6         3.0          4.5         1.5
## 68          5.8         2.7          4.1         1.0
## 69          6.2         2.2          4.5         1.5
## 70          5.6         2.5          3.9         1.1
## 71          5.9         3.2          4.8         1.8
## 72          6.1         2.8          4.0         1.3
## 73          6.3         2.5          4.9         1.5
## 74          6.1         2.8          4.7         1.2
## 75          6.4         2.9          4.3         1.3
## 76          6.6         3.0          4.4         1.4
## 77          6.8         2.8          4.8         1.4
## 78          6.7         3.0          5.0         1.7
## 79          6.0         2.9          4.5         1.5
## 80          5.7         2.6          3.5         1.0
## 81          5.5         2.4          3.8         1.1
## 82          5.5         2.4          3.7         1.0
## 83          5.8         2.7          3.9         1.2
## 84          6.0         2.7          5.1         1.6
## 85          5.4         3.0          4.5         1.5
## 86          6.0         3.4          4.5         1.6
## 87          6.7         3.1          4.7         1.5
## 88          6.3         2.3          4.4         1.3
## 89          5.6         3.0          4.1         1.3
## 90          5.5         2.5          4.0         1.3
## 91          5.5         2.6          4.4         1.2
## 92          6.1         3.0          4.6         1.4
## 93          5.8         2.6          4.0         1.2
```

```
## 94           5.0         2.3          3.3         1.0
## 95           5.6         2.7          4.2         1.3
## 96           5.7         3.0          4.2         1.2
## 97           5.7         2.9          4.2         1.3
## 98           6.2         2.9          4.3         1.3
## 99           5.1         2.5          3.0         1.1
## 100          5.7         2.8          4.1         1.3
## 101          6.3         3.3          6.0         2.5
## 102          5.8         2.7          5.1         1.9
## 103          7.1         3.0          5.9         2.1
## 104          6.3         2.9          5.6         1.8
## 105          6.5         3.0          5.8         2.2
## 106          7.6         3.0          6.6         2.1
## 107          4.9         2.5          4.5         1.7
## 108          7.3         2.9          6.3         1.8
## 109          6.7         2.5          5.8         1.8
## 110          7.2         3.6          6.1         2.5
## 111          6.5         3.2          5.1         2.0
## 112          6.4         2.7          5.3         1.9
## 113          6.8         3.0          5.5         2.1
## 114          5.7         2.5          5.0         2.0
## 115          5.8         2.8          5.1         2.4
## 116          6.4         3.2          5.3         2.3
## 117          6.5         3.0          5.5         1.8
## 118          7.7         3.8          6.7         2.2
## 119          7.7         2.6          6.9         2.3
## 120          6.0         2.2          5.0         1.5
## 121          6.9         3.2          5.7         2.3
## 122          5.6         2.8          4.9         2.0
## 123          7.7         2.8          6.7         2.0
## 124          6.3         2.7          4.9         1.8
## 125          6.7         3.3          5.7         2.1
## 126          7.2         3.2          6.0         1.8
## 127          6.2         2.8          4.8         1.8
## 128          6.1         3.0          4.9         1.8
## 129          6.4         2.8          5.6         2.1
## 130          7.2         3.0          5.8         1.6
## 131          7.4         2.8          6.1         1.9
## 132          7.9         3.8          6.4         2.0
## 133          6.4         2.8          5.6         2.2
## 134          6.3         2.8          5.1         1.5
## 135          6.1         2.6          5.6         1.4
## 136          7.7         3.0          6.1         2.3
## 137          6.3         3.4          5.6         2.4
## 138          6.4         3.1          5.5         1.8
## 139          6.0         3.0          4.8         1.8
## 140          6.9         3.1          5.4         2.1
## 141          6.7         3.1          5.6         2.4
## 142          6.9         3.1          5.1         2.3
## 143          5.8         2.7          5.1         1.9
## 144          6.8         3.2          5.9         2.3
## 145          6.7         3.3          5.7         2.5
## 146          6.7         3.0          5.2         2.3
## 147          6.3         2.5          5.0         1.9
```

```
## 148          6.5         3.0          5.2         2.0
## 149          6.2         3.4          5.4         2.3
## 150          5.9         3.0          5.1         1.8
```

```r
iris %>% select(-Species)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
## 15           5.8         4.0          1.2         0.2
## 16           5.7         4.4          1.5         0.4
## 17           5.4         3.9          1.3         0.4
## 18           5.1         3.5          1.4         0.3
## 19           5.7         3.8          1.7         0.3
## 20           5.1         3.8          1.5         0.3
## 21           5.4         3.4          1.7         0.2
## 22           5.1         3.7          1.5         0.4
## 23           4.6         3.6          1.0         0.2
## 24           5.1         3.3          1.7         0.5
## 25           4.8         3.4          1.9         0.2
## 26           5.0         3.0          1.6         0.2
## 27           5.0         3.4          1.6         0.4
## 28           5.2         3.5          1.5         0.2
## 29           5.2         3.4          1.4         0.2
## 30           4.7         3.2          1.6         0.2
## 31           4.8         3.1          1.6         0.2
## 32           5.4         3.4          1.5         0.4
## 33           5.2         4.1          1.5         0.1
## 34           5.5         4.2          1.4         0.2
## 35           4.9         3.1          1.5         0.2
## 36           5.0         3.2          1.2         0.2
## 37           5.5         3.5          1.3         0.2
## 38           4.9         3.6          1.4         0.1
## 39           4.4         3.0          1.3         0.2
## 40           5.1         3.4          1.5         0.2
## 41           5.0         3.5          1.3         0.3
## 42           4.5         2.3          1.3         0.3
## 43           4.4         3.2          1.3         0.2
## 44           5.0         3.5          1.6         0.6
## 45           5.1         3.8          1.9         0.4
## 46           4.8         3.0          1.4         0.3
## 47           5.1         3.8          1.6         0.2
## 48           4.6         3.2          1.4         0.2
```

```
## 49           5.3          3.7           1.5          0.2
## 50           5.0          3.3           1.4          0.2
## 51           7.0          3.2           4.7          1.4
## 52           6.4          3.2           4.5          1.5
## 53           6.9          3.1           4.9          1.5
## 54           5.5          2.3           4.0          1.3
## 55           6.5          2.8           4.6          1.5
## 56           5.7          2.8           4.5          1.3
## 57           6.3          3.3           4.7          1.6
## 58           4.9          2.4           3.3          1.0
## 59           6.6          2.9           4.6          1.3
## 60           5.2          2.7           3.9          1.4
## 61           5.0          2.0           3.5          1.0
## 62           5.9          3.0           4.2          1.5
## 63           6.0          2.2           4.0          1.0
## 64           6.1          2.9           4.7          1.4
## 65           5.6          2.9           3.6          1.3
## 66           6.7          3.1           4.4          1.4
## 67           5.6          3.0           4.5          1.5
## 68           5.8          2.7           4.1          1.0
## 69           6.2          2.2           4.5          1.5
## 70           5.6          2.5           3.9          1.1
## 71           5.9          3.2           4.8          1.8
## 72           6.1          2.8           4.0          1.3
## 73           6.3          2.5           4.9          1.5
## 74           6.1          2.8           4.7          1.2
## 75           6.4          2.9           4.3          1.3
## 76           6.6          3.0           4.4          1.4
## 77           6.8          2.8           4.8          1.4
## 78           6.7          3.0           5.0          1.7
## 79           6.0          2.9           4.5          1.5
## 80           5.7          2.6           3.5          1.0
## 81           5.5          2.4           3.8          1.1
## 82           5.5          2.4           3.7          1.0
## 83           5.8          2.7           3.9          1.2
## 84           6.0          2.7           5.1          1.6
## 85           5.4          3.0           4.5          1.5
## 86           6.0          3.4           4.5          1.6
## 87           6.7          3.1           4.7          1.5
## 88           6.3          2.3           4.4          1.3
## 89           5.6          3.0           4.1          1.3
## 90           5.5          2.5           4.0          1.3
## 91           5.5          2.6           4.4          1.2
## 92           6.1          3.0           4.6          1.4
## 93           5.8          2.6           4.0          1.2
## 94           5.0          2.3           3.3          1.0
## 95           5.6          2.7           4.2          1.3
## 96           5.7          3.0           4.2          1.2
## 97           5.7          2.9           4.2          1.3
## 98           6.2          2.9           4.3          1.3
## 99           5.1          2.5           3.0          1.1
## 100          5.7          2.8           4.1          1.3
## 101          6.3          3.3           6.0          2.5
## 102          5.8          2.7           5.1          1.9
```

```
## 103          7.1         3.0          5.9         2.1
## 104          6.3         2.9          5.6         1.8
## 105          6.5         3.0          5.8         2.2
## 106          7.6         3.0          6.6         2.1
## 107          4.9         2.5          4.5         1.7
## 108          7.3         2.9          6.3         1.8
## 109          6.7         2.5          5.8         1.8
## 110          7.2         3.6          6.1         2.5
## 111          6.5         3.2          5.1         2.0
## 112          6.4         2.7          5.3         1.9
## 113          6.8         3.0          5.5         2.1
## 114          5.7         2.5          5.0         2.0
## 115          5.8         2.8          5.1         2.4
## 116          6.4         3.2          5.3         2.3
## 117          6.5         3.0          5.5         1.8
## 118          7.7         3.8          6.7         2.2
## 119          7.7         2.6          6.9         2.3
## 120          6.0         2.2          5.0         1.5
## 121          6.9         3.2          5.7         2.3
## 122          5.6         2.8          4.9         2.0
## 123          7.7         2.8          6.7         2.0
## 124          6.3         2.7          4.9         1.8
## 125          6.7         3.3          5.7         2.1
## 126          7.2         3.2          6.0         1.8
## 127          6.2         2.8          4.8         1.8
## 128          6.1         3.0          4.9         1.8
## 129          6.4         2.8          5.6         2.1
## 130          7.2         3.0          5.8         1.6
## 131          7.4         2.8          6.1         1.9
## 132          7.9         3.8          6.4         2.0
## 133          6.4         2.8          5.6         2.2
## 134          6.3         2.8          5.1         1.5
## 135          6.1         2.6          5.6         1.4
## 136          7.7         3.0          6.1         2.3
## 137          6.3         3.4          5.6         2.4
## 138          6.4         3.1          5.5         1.8
## 139          6.0         3.0          4.8         1.8
## 140          6.9         3.1          5.4         2.1
## 141          6.7         3.1          5.6         2.4
## 142          6.9         3.1          5.1         2.3
## 143          5.8         2.7          5.1         1.9
## 144          6.8         3.2          5.9         2.3
## 145          6.7         3.3          5.7         2.5
## 146          6.7         3.0          5.2         2.3
## 147          6.3         2.5          5.0         1.9
## 148          6.5         3.0          5.2         2.0
## 149          6.2         3.4          5.4         2.3
## 150          5.9         3.0          5.1         1.8
```

- `filter`: Extracts a subset of rows from a data frame based on logical conditions
- `arrange`: Reorders rows of a data frame according to a variable or column
- `rename`: Makes it easier to rename variables in a data frame
- `mutate`: Computes transformations of variables in a data frame
- `summarize`: Collapses a group into a single row

Data manipulation with tibble

Data manipulation with reshape2

```
library(reshape2)
library(ggplot2)

view1 <- iris %>% group_by(Species) %>%
  summarise_if(is.numeric, list(mean = mean, sd = sd)) %>%
  # using reshape2 package to reshape from wide to long format
  # the tidyr::pivot_longer function can also be used
  melt(id.vars = "Species") %>%
  separate(variable, into = c("variable", "metric"), sep = "_")

view1
```

```
##       Species     variable metric     value
## 1      setosa Sepal.Length   mean 5.0060000
## 2  versicolor Sepal.Length   mean 5.9360000
## 3   virginica Sepal.Length   mean 6.5880000
## 4      setosa  Sepal.Width   mean 3.4280000
## 5  versicolor  Sepal.Width   mean 2.7700000
## 6   virginica  Sepal.Width   mean 2.9740000
## 7      setosa Petal.Length   mean 1.4620000
## 8  versicolor Petal.Length   mean 4.2600000
## 9   virginica Petal.Length   mean 5.5520000
## 10     setosa  Petal.Width   mean 0.2460000
## 11 versicolor  Petal.Width   mean 1.3260000
## 12  virginica  Petal.Width   mean 2.0260000
## 13     setosa Sepal.Length     sd 0.3524897
## 14 versicolor Sepal.Length     sd 0.5161711
## 15  virginica Sepal.Length     sd 0.6358796
## 16     setosa  Sepal.Width     sd 0.3790644
## 17 versicolor  Sepal.Width     sd 0.3137983
## 18  virginica  Sepal.Width     sd 0.3224966
## 19     setosa Petal.Length     sd 0.1736640
## 20 versicolor Petal.Length     sd 0.4699110
## 21  virginica Petal.Length     sd 0.5518947
## 22     setosa  Petal.Width     sd 0.1053856
## 23 versicolor  Petal.Width     sd 0.1977527
## 24  virginica  Petal.Width     sd 0.2746501
```

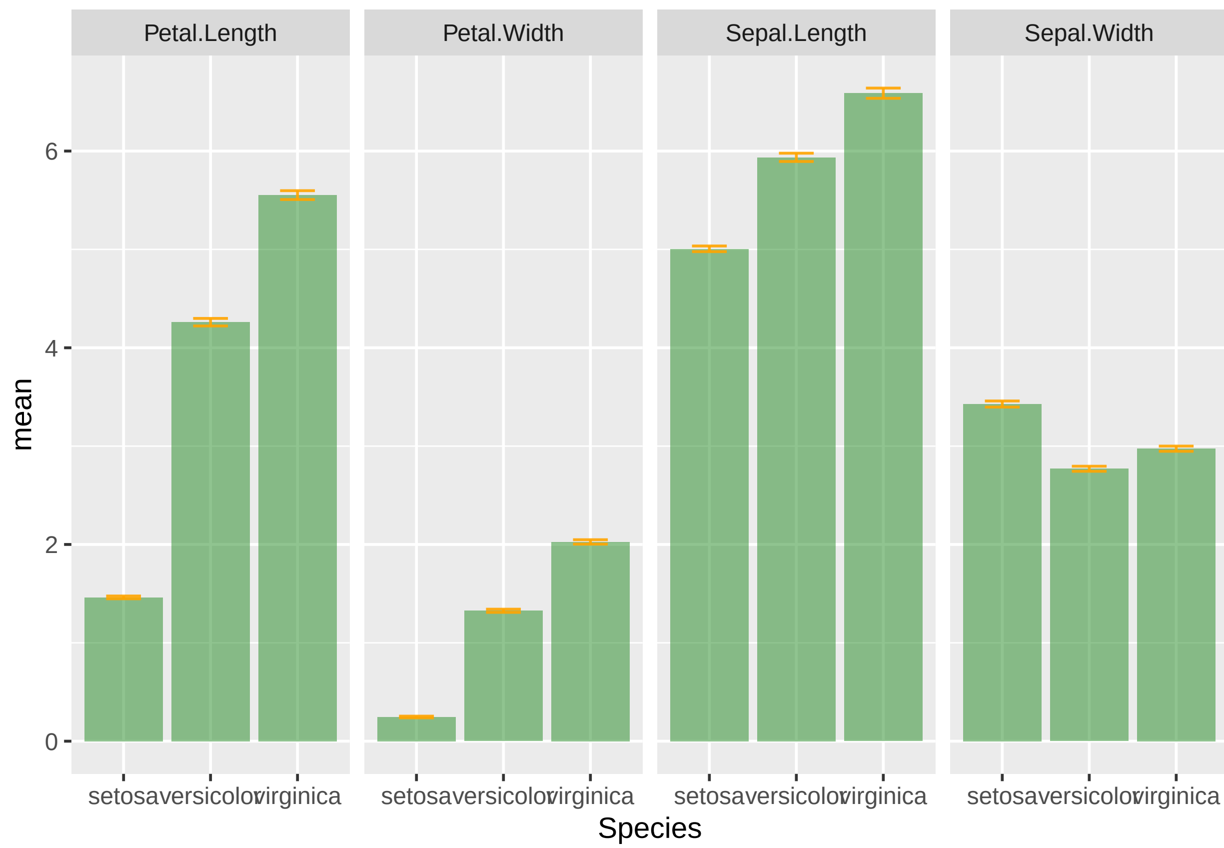
Now if we want to have `mean` and `sd` as columns, the `tidyr::pivot_wider` can be used.

```
view2 <- view1 %>%
  pivot_wider(names_from = metric, values_from = value) %>%
  mutate(std_error = sd/sqrt(nrow(iris)))

view2
```

```
## # A tibble: 12 x 5
##    Species    variable      mean    sd std_error
##    <fct>      <chr>        <dbl> <dbl>     <dbl>
##  1 setosa     Sepal.Length  5.01 0.352    0.0288
##  2 versicolor Sepal.Length  5.94 0.516    0.0421
##  3 virginica  Sepal.Length  6.59 0.636    0.0519
```

```
##   4 setosa     Sepal.Width  3.43  0.379   0.0310
##   5 versicolor Sepal.Width  2.77  0.314   0.0256
##   6 virginica  Sepal.Width  2.97  0.322   0.0263
##   7 setosa     Petal.Length 1.46  0.174   0.0142
##   8 versicolor Petal.Length 4.26  0.470   0.0384
##   9 virginica  Petal.Length 5.55  0.552   0.0451
##  10 setosa     Petal.Width  0.246 0.105   0.00860
##  11 versicolor Petal.Width  1.33  0.198   0.0161
##  12 virginica  Petal.Width  2.03  0.275   0.0224
```

```
view2 %>% ggplot() +
  geom_bar(aes(x = Species, y = mean), stat="identity", fill="forestgreen", alpha = 0.5) +
  geom_errorbar(aes(x=Species,
                    ymin = mean - std_error,
                    ymax = mean + std_error),
                width = 0.4, colour="orange",
                alpha = 0.9, linewidth = 0.5) +
  facet_wrap(~variable, ncol = 4)
```



Data display with kabbleExtra

- https://bookdown.org/yihui/rmarkdown-cookbook/kableextra.html

Create beautiful tables with flextable

```r
library(broom)
library(tidyverse)
library(flextable)
tidy(LR) %>% select(-5) %>%  flextable() %>% autofit() %>% theme_box()
```

term	estimate	std.error	statistic
(Intercept)	4.043760	0.10906295	37.0773
x	7.987287	0.02081526	383.7227

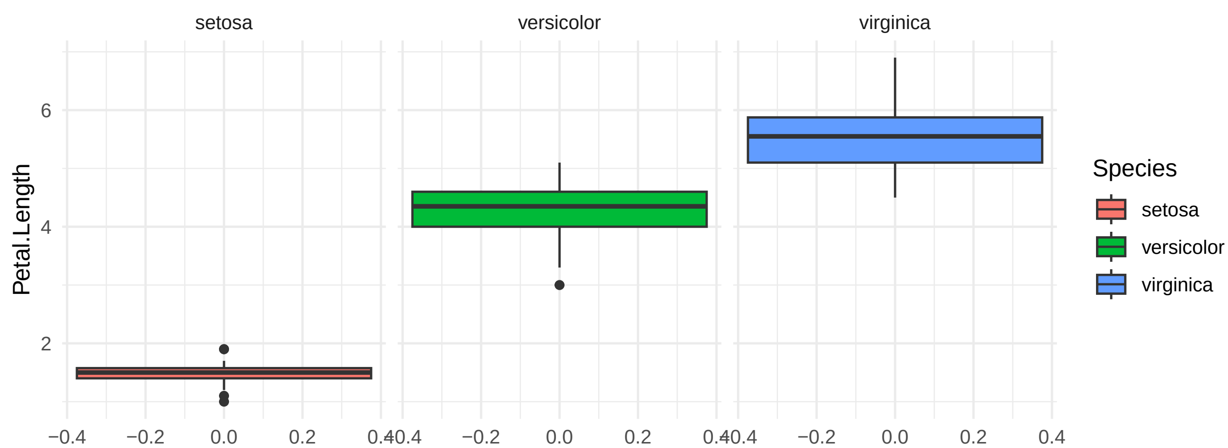
Manipulate Microsoft Word and PowerPoint Documents with `officer`

- https://ardata-fr.github.io/officeverse/

Visualization

Data visualization with ggplot2

```r
iris %>%
  ggplot(aes(y = Petal.Length,  fill = Species)) +
  geom_boxplot() +
  facet_grid(~Species) +
  theme_minimal()
```

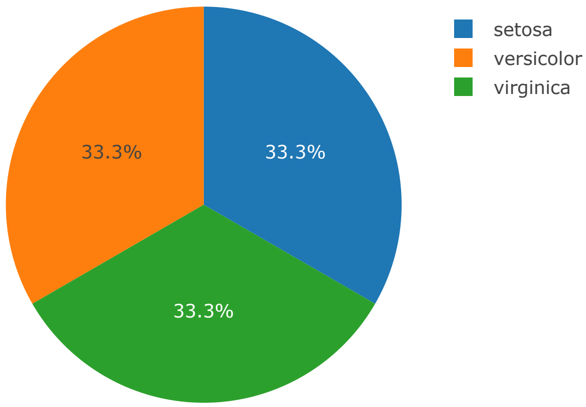


- https://bookdown.org/ozancanozdemir/introduction-to-ggplot2/

Data visualization with plotly

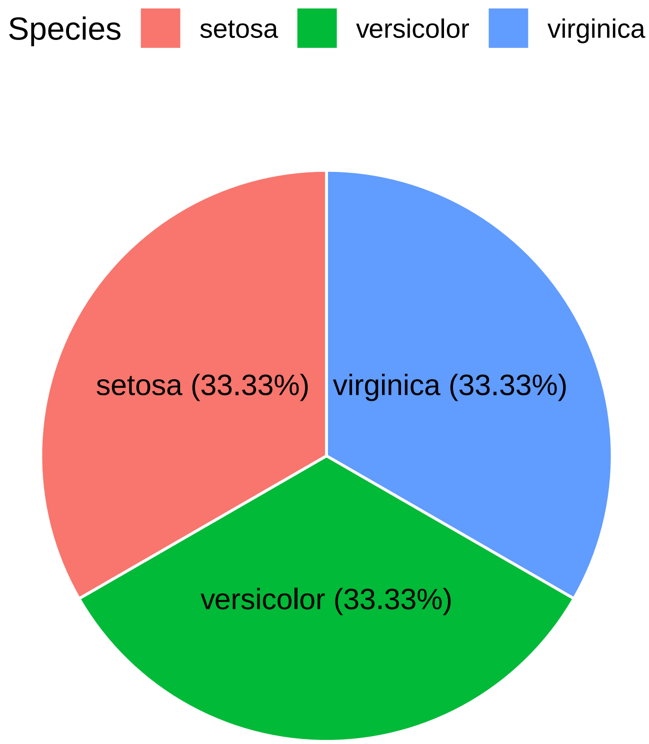
- https://plotly.com/r/
- https://bookdown.org/ronsarafian/IntrotoDS/plotting.html

```r
library(plotly)
library(ggpubr)
plot_ly(data = iris %>% count(Species),
                labels = ~Species, values = ~n, type = "pie")
```




```
ggpie(data = iris %>% count(Species) %>%
        mutate(label = paste0(Species, " (", round(100* n/sum(n), 2), "%)")),
```

```
    x = "n", label = "label",
    color = "white",
    fill = "Species", lab.pos = "in")
```



R advanced

Regular expressions

- https://www.datacamp.com/tutorial/regex-r-regular-expressions-guide
- https://jfjelstul.github.io/regular-expressions-tutorial/
- https://www.geeksforgeeks.org/regular-expressions-in-r/

Unsupervised & Supervised Learning

Principal Component Analysis

Clustering: K-means, Hierarchical Clustering

K-Nearest Neighbor

Simple Linear Regression

Logistic Regression

Latex in Rstudio (R markdown/Quarto markdown)

The variance of a real-valued variables `$X = (X_1, \ldots, X_n)$` is given by:

The variance of a real-valued variables $X = (X_1, \dots, X_n)$ is given by:

```
$$
\textrm{Var(X)} =
\left[\frac{1}{n}\sum_{i=1}^n\left(X_i-\frac{1}{n}
\sum_{i=1}^nX_i\right)^2\right]^\frac{1}{2}
$$
```

$$
\textrm{Var(X)} = \left[\frac{1}{n}\sum_{i=1}^n\left(X_i-\frac{1}{n}\sum_{i=1}^nX_i\right)^2\right]^\frac{1}{2}
$$