Introduction to R programming

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

Lema Logamou Seknewna, Data Scientist à AIMS RIC

25 October, 2024

Contents

Install R

1. Choose your OS (operating system):

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

2. Install Rstudio:

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

Operators

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


Rad
Deg
x!
Inv
sin
ln
π
cos
log
e
tan
√
Ans
EXP
xʸ

Arithmetic Operators

Addition: +

```
1+1
```

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

Subtraction: -

```
1-1
```

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

Multiplication: *

```
1*1
```

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

Division: /

```
1/1
```

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

Modulus (remaining of a division) : %%

```
1 %% 2
```

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

Exponent : ^ or **

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

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

Integer division: %/%

```
1035 %/% 3
```

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

Logical operators

Less than: <

```
1 < 1
```

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

Less than or equal to: <=

```
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 !!!

Not equal to: `!=`

```r
1 != 1
```

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

Négation/NON: `!`

Utiliser pour changer une proposition fausse en vraie (ou vraie en fausse)

```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: `|`

```
TRUE | TRUE
```

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

```
TRUE | FALSE
```

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

```
FALSE | FALSE
```

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

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 `_`.

```
v rsion <- 4.3.2
```

```
## Error: <text>:1:3: unexpected symbol
## 1: v rsion
##       ^
```

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

```
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 - (10.5, 55, 787)

```
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 - (9 + 3i, where "i" is the imaginary part)

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

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

```r
typeof(z)
```

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

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

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

```r
class(z1)
```

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

Character/string

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

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

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

Logical/Boolean - (TRUE or FALSE)

```r
TRUE # or T
```

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

```r
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")
# }
```

R Data Structures

The most used data types in R are

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

Scalars and vectors:

- 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;

Example 1:

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

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

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

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

```r
# sequence: ?seq
x <- seq(0, 2*pi, length.out = 90)
y <- seq(0, 2*pi, by = 0.1282283)
head(x); head(y)
```

```
## [1] 0.00000000 0.07059759 0.14119518 0.21179276 0.28239035 0.35298794
```

```
## [1] 0.0000000 0.1282283 0.2564566 0.3846849 0.5129132 0.6411415
```

```r
tail(x)
```

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

```r
range(x)
```

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

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

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

```r
rg[2]
```

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

```r
x[10]
```

```
## [1] 0.6353783
```

The length of a vector is given by:

```r
length(x)
```

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

```r
length(rg)
```

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

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

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

A scalar is a vector of length 1.

Example 2:

```r
# repeating
rep("I learn R", 5)
```

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

```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
```

```r
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(0:5), 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 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [186] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [223] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5
## [260] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [297] 5 5 5 5
```

```r
# sampling
set.seed(24102024) # fix the randomness for reproducibility.
sample(0:1, size = 100, replace = TRUE, prob = c(0.3, 0.7)) -> y
y; y == 0
```

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

```
##   [1] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
##  [13] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
##  [37]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
##  [49] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [73] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE  TRUE
```

```r
table(y)
```

```
## y
##  0  1
## 21 79
```

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

```
## [1] 21
```

```
## [1] 79
```

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

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

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

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

```r
sum(c(T, F))
```

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

Matrices:

Matrices are two dimensional data set with columns and rows.

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

```
##      [,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
```

```r

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

Matrix definition

```
##      [,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

$$\begin{cases} 2x + 2y &= 4 \\ x + 3y &= 4 \end{cases}$$

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(4, 4)
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
```

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

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

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

Eigen values/vectors

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

```
is.list(ev)
```

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

```
ev$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
```

Arrays

Arrays are data type with more than two dimensions

```
(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
```

```
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.

```
dim(aRray)
```

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

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

mylist$matrix
```

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

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

Accessing elements of a list

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

```r
mylist$Array
```

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

Accessing elements of a list

```
## $Array
## , , 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
##
##
## $matrix
##      [,1] [,2] [,3]
## [1,]    1    5    4
## [2,]    0    2    2
## [3,]    2    1    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 take any data type.

Factors

Converting a continuous variable into a categorical variable

```r
set.seed(12092024)
age <- sample(0:120, size = 100)
(brks <- seq(0, 120, by = 10))
```

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

```r
# (brks <- seq(min(age), max(age), le = 40))
age_groups <- cut(age, breaks = brks, include.lowest = TRUE)
class(age_groups)
```

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

```r
table(age_groups)
```

```
## age_groups
##    [0,10]   (10,20]   (20,30]   (30,40]   (40,50]   (50,60]   (60,70]   (70,80]
##        10         9         9         5         8         8         9         8
##   (80,90]  (90,100] (100,110] (110,120]
##         9         8         9         8
```

```r
# checking for missing values
(which(is.na(age_groups)) -> id_missing)
```

```
## integer(0)
```

```r
# convertion
age_factor <- factor(age_groups) # not necessary!
```

```r
identical(age_groups, age_factor)
```

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

```r
# count in each class/group
frequencies <- table(age_groups)
```

Data frames

A data frame a is also a list where all elements (columns) have the same length. A data frame in R is a table.

```
# converting a list into a dataframe
```

```


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

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

```
##             x y
## 1  1.56733455 4
## 2 -1.23476680 2
## 3 -1.33309877 1
## 4  1.00248238 2
## 5  1.41179396 2
## 6 -0.09788651 5
```

Data manipulation

- Missing values (`NA`)

```
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
```

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

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

```
              # 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
```

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

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

```
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
```

- NAs introduced by coercion when converting strings to numeric

```
x <- c(2, 1, 2, 7, 3, 2.5, 9, "2,7")
class(x)
```

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

```
# converting into numeric
z <- as.numeric(x)
```

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

```
x[which(is.na(z))] <- 2.7
z[which(is.na(z))] <- 2.7
z; x
```

```
## [1] 2.0 1.0 2.0 7.0 3.0 2.5 9.0 2.7
```

```
## [1] "2"   "1"   "2"   "7"   "3"   "2.5" "9"   "2.7"
```

- Outliers detection

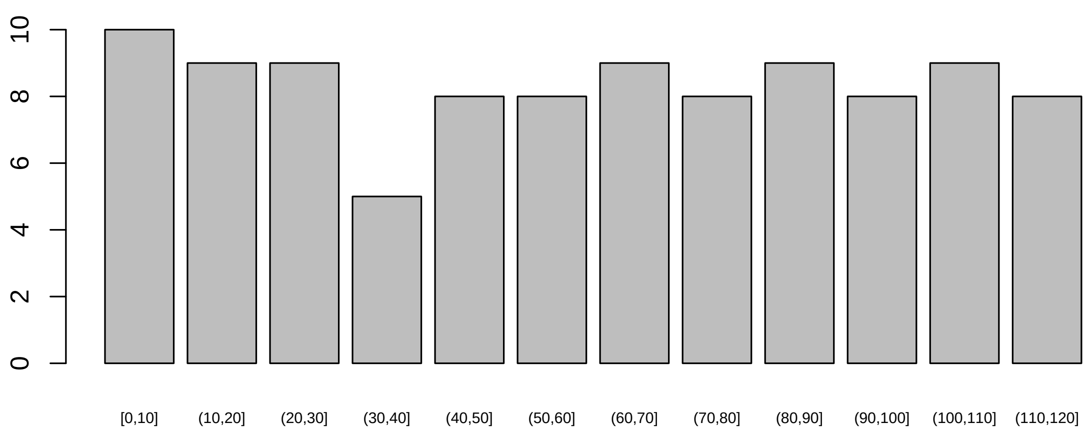
Plots from a data frame

Data simulation and visualization

Charts in R

Bar chart/plot

```
barplot(frequencies, cex.names=0.6) # 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.

- Number of observations in a subset: $\sum_{i=1}^{n} I_{\{\text{age}_i \geq 80\}}$
- Percentage of observations in a subset: $\frac{100}{n}\sum_{i=1}^{n} I_{\{\text{age}_i \geq 80\}}$
- n is the sample size.

```
sum(age >= 80)
```

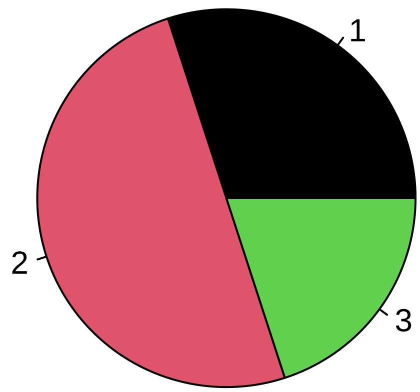
```
## [1] 35
```

```r
mean(age >= 80) # I get the relative frequency
```

```
## [1] 0.35
```

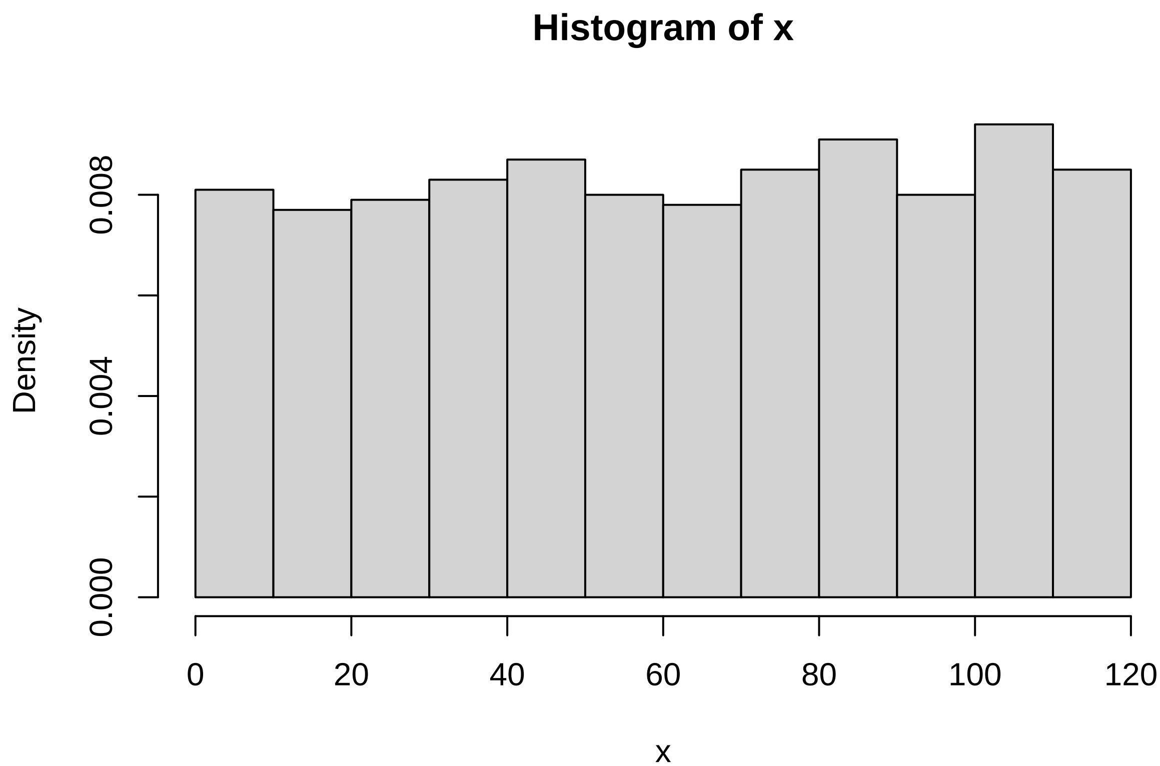
Pie chart/plot

```r
pie(c(30, 50, 20), col = 1:3)
```

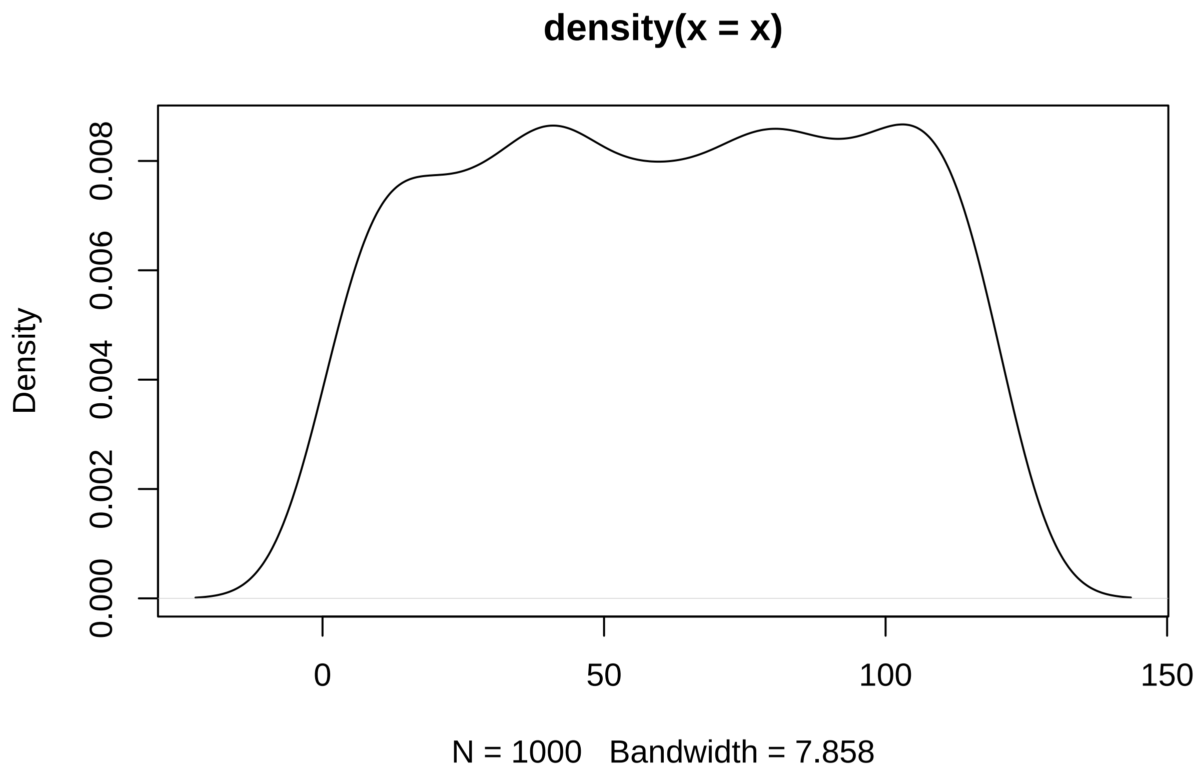


Histograms

```r
set.seed(12092024)
x <- sample(1:120, size = 1000, replace = TRUE);
hist(x, probability = TRUE) # use probability = TRUE to have densities
```

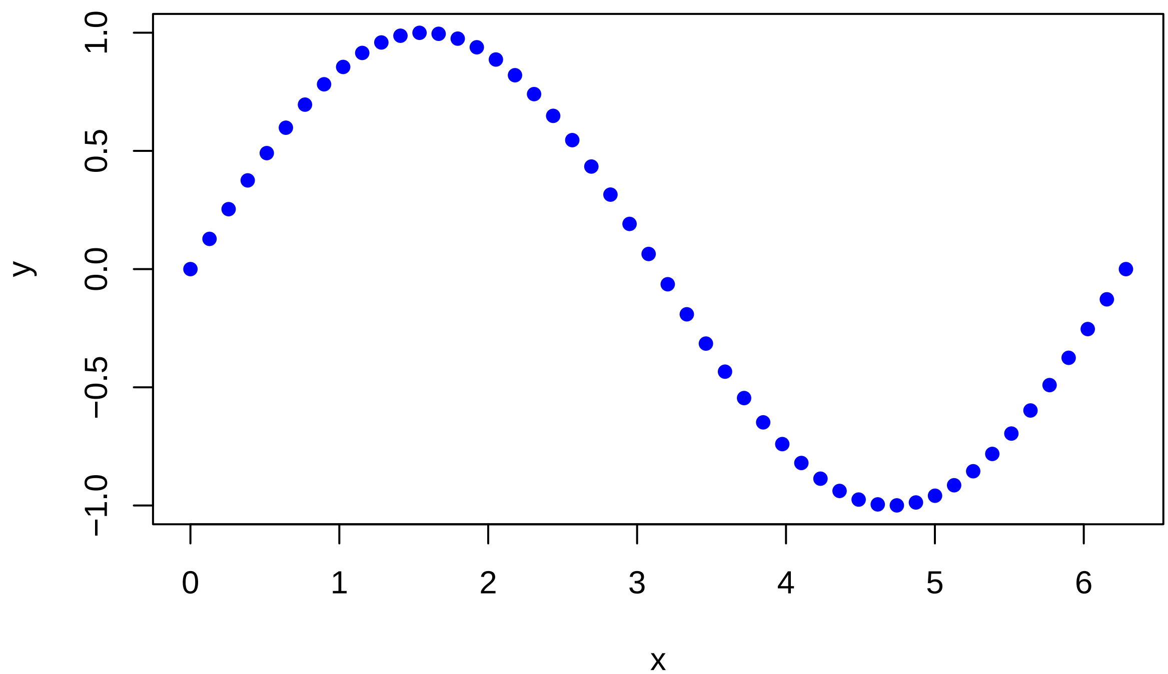



```r
        # instead of counts (frequencies)
# Density plots
plot(density(x))
```



Scatter plot

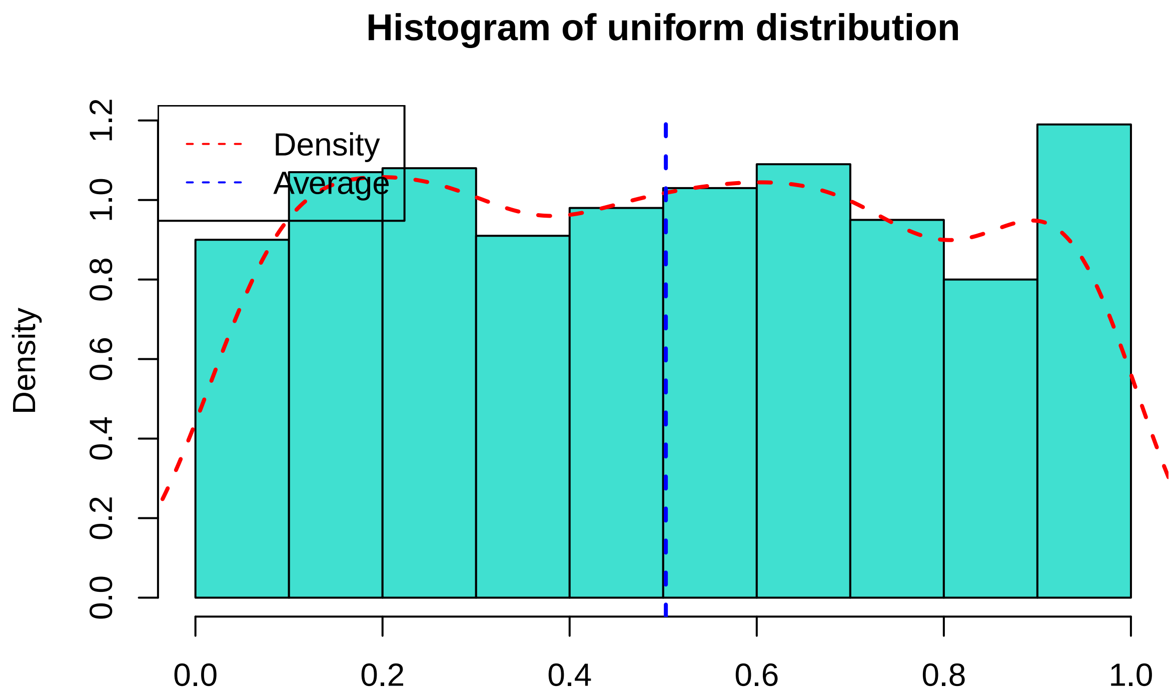
```
x <- seq(0, 2*pi, le = 50)
y <- sin(x)
z <- cos(x)
tg <- tan(x)
plot(x, y, pch = 16, col = "blue")
```



Distribution simulations

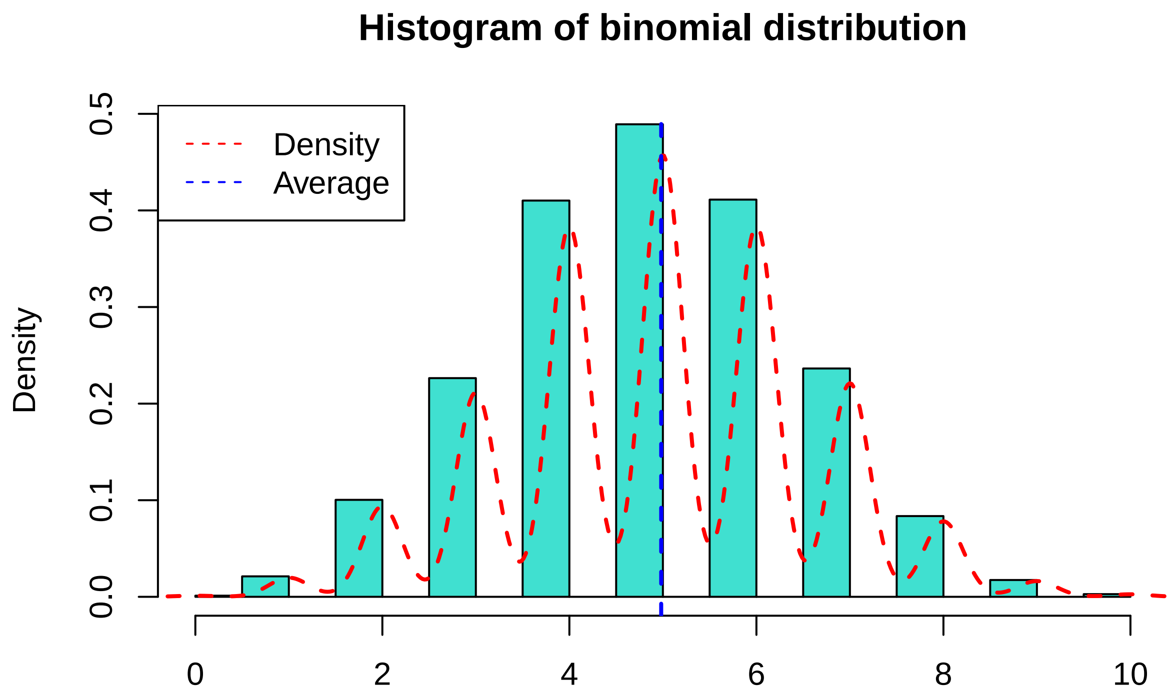
Uniform distribution

```
set.seed(13092024)
unif_dist <- runif(1000)
hist(unif_dist, probability = TRUE, xlab = NULL,
     main = "Histogram of uniform distribution", col = "turquoise")
lines(density(unif_dist), col = "red", lwd = 2, lty = 2)
abline(v = mean(unif_dist), col = "blue", lty = 2, lwd = 2)
legend("topleft", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"))
```



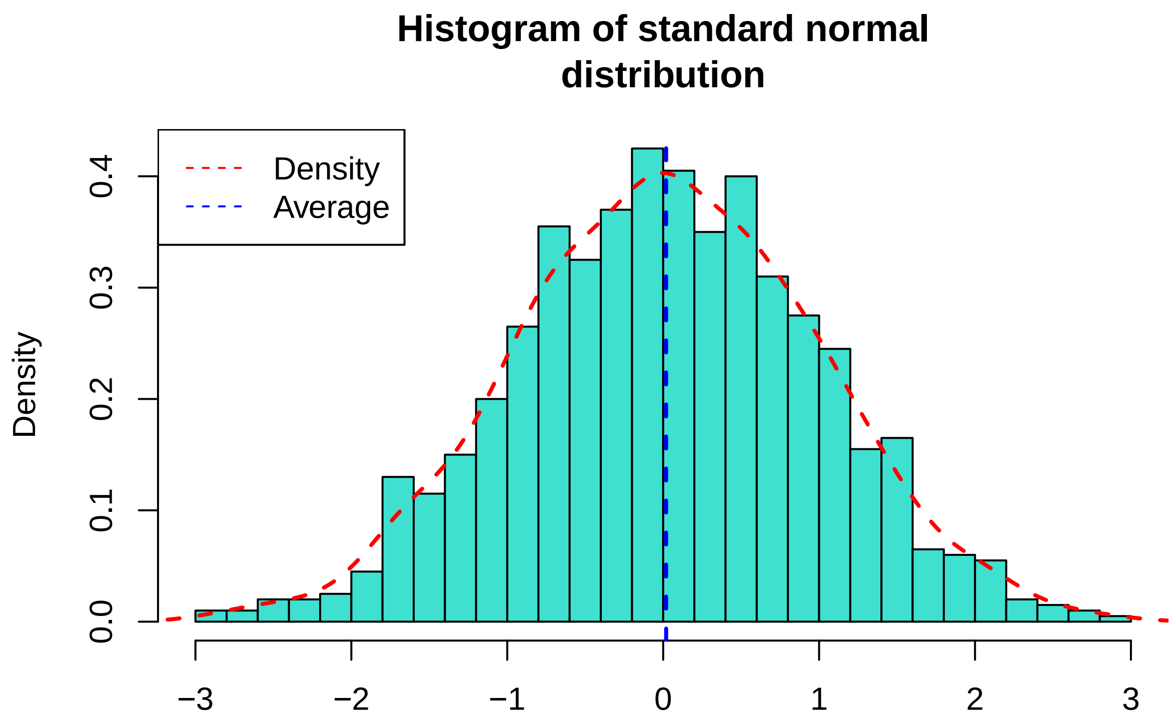
Binomial distribution

```
rbinom_dist <- rbinom(10000, 10, 0.5)
hist(rbinom_dist, probability = TRUE, main = "Histogram of 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("topleft", lty = c(2, 2),
       col = c("red", "blue"), legend = c("Density", "Average"))
```



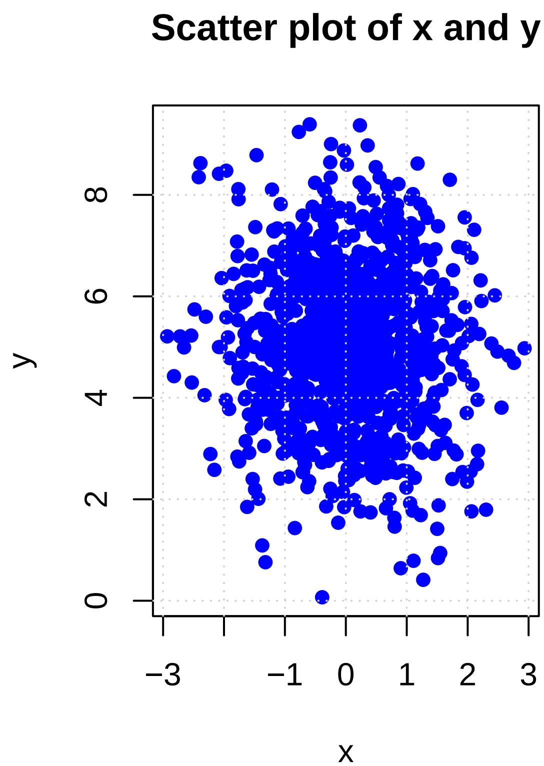
Gaussian distribution

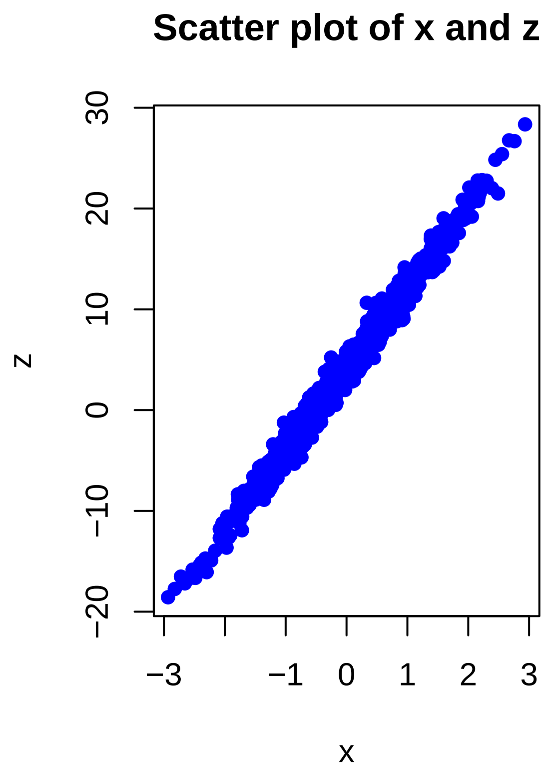
```
set.seed(13092024)
gauss_dist <- rnorm(1000, mean = 0, sd = 1)
hist(gauss_dist, probability = TRUE, breaks = 30, xlab = NULL,
     main = "Histogram of standard normal\ndistribution", col = "turquoise")
lines(density(gauss_dist), col = "red", lwd = 2, lty = 2)
abline(v = mean(gauss_dist), col = "blue",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



Scatter plot to show relationship between two variables

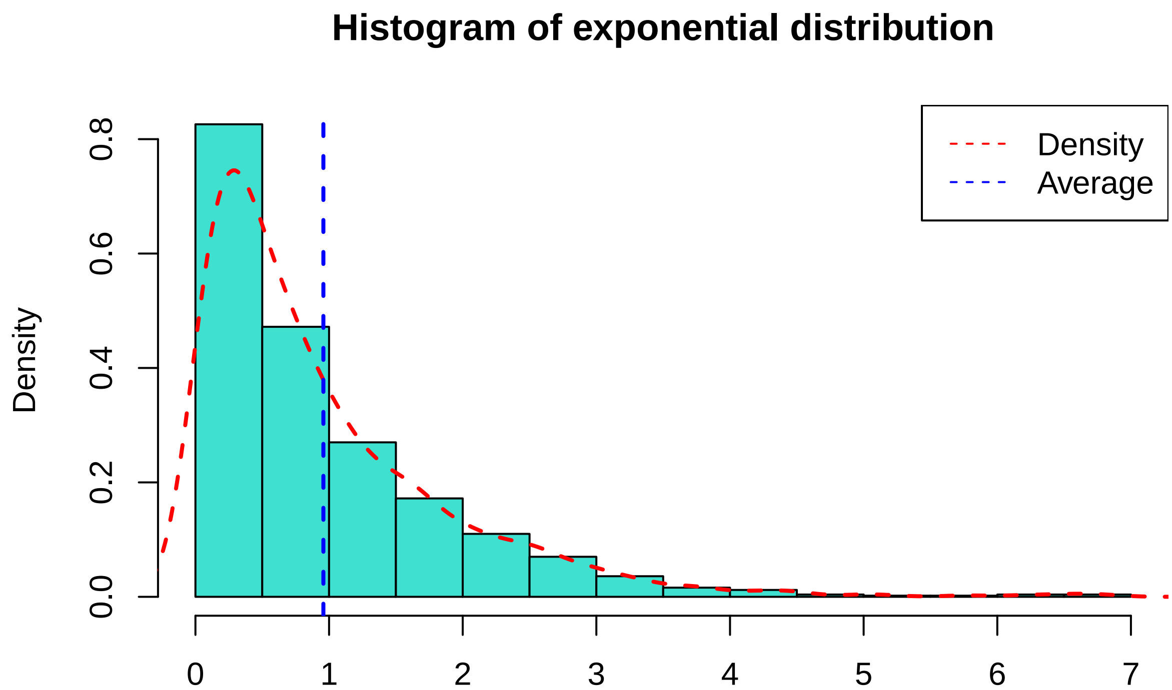
```
set.seed(13092024)
x <- rnorm(1000); y <- rnorm(1000, mean = 5, sd = 1.5)
z <- 4 + 8*x + rnorm(1000) # linear dependence between x and z
par(mfrow = c(1, 2))
plot(x, y, main = "Scatter plot of x and y", col = "blue", pch = 16); grid()
plot(x, z, main = "Scatter plot of x and z", col = "blue", pch = 16)
```





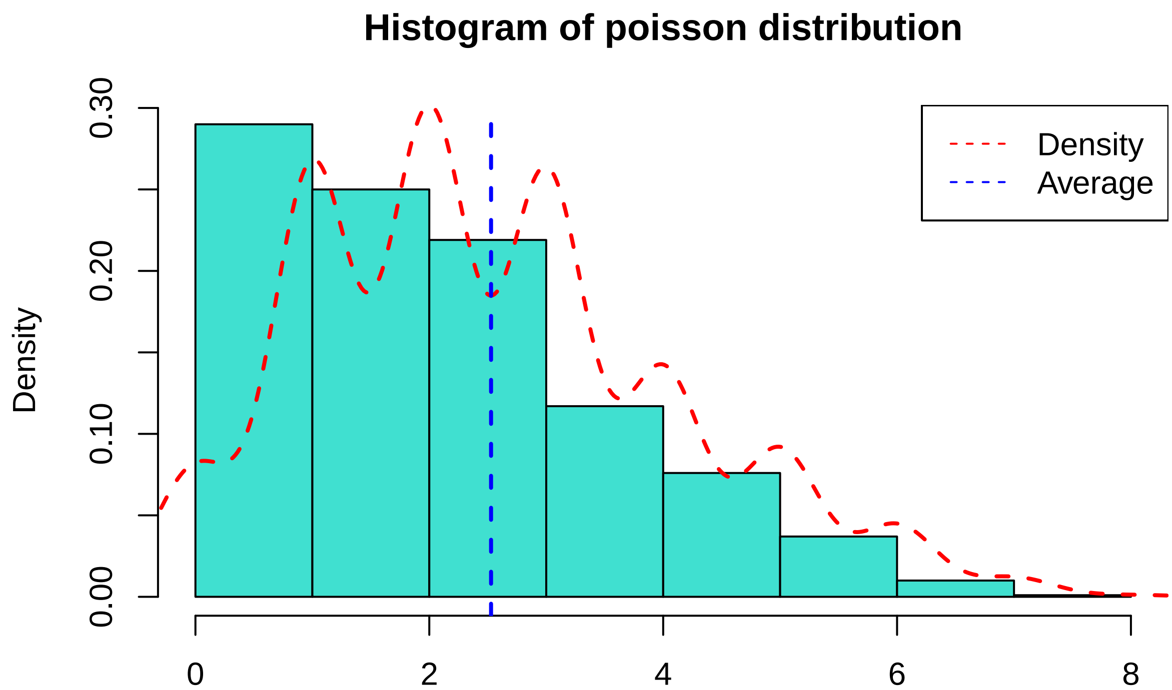
Exponential distribution

```
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
# one line
ifelse(x < 4, T, F)
```

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

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

```r
if (x < 4){
  if (x != 0){
    print("x is not equal to zero.")
  } else {
    print("x is equal to zero")
  }

  print("x is less than 4")
} else {
  if (x > 1){
    print("x is greater than 1.")
  } else {
    print("x is less than or equal to 1")
  }
  print("x is greater than 4.")
}
```

```
## [1] "x is not equal to zero."
## [1] "x is less than 4"
```

Loops

- for loops

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

```r
m <- 6
for (i in 1:m) print(i)
```

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

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

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

Exercise 1:

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

```r
## TODO
```

Exercise 2:

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

```
# checking the working directory
getwd()
```

```
## [1] "/Users/logamouseknewnalema/Library/CloudStorage/GoogleDrive-lseknewna@aimsric.org/My Drive/INDAB
```

```
# simple ls like in bash
dir()
```

```
##  [1] "a.csv"
##  [2] "aims-logo.jpg"
##  [3] "calc.png"
##  [4] "donnees_hospitalieres"
##  [5] "mypackages.R"
##  [6] "pipe.jpeg"
##  [7] "R_Intro_DrLema_files"
##  [8] "R_Intro_DrLema.html"
##  [9] "R_Intro_DrLema.pdf"
## [10] "R_Intro_DrLema.Rmd"
## [11] "R_Intro_IndabaX_Chad.html"
## [12] "R_Intro_IndabaX_Chad.Rmd"
## [13] "rsconnect"
## [14] "Screenshot 2024-02-20 at 17.05.25.png"
## [15] "Screenshot 2024-09-27 at 14.37.41.png"
## [16] "Screenshot 2024-09-30 at 15.36.49.png"
## [17] "Screenshot 2024-09-30 at 15.39.23.png"
## [18] "Screenshot 2024-10-24 at 10.24.29.png"
## [19] "Screenshot 2024-10-24 at 10.25.09.png"
## [20] "Screenshot 2024-10-24 at 10.28.26.png"
## [21] "Screenshot 2024-10-24 at 10.34.32.png"
## [22] "Screenshot 2024-10-24 at 10.37.26.png"
## [23] "Screenshot 2024-10-24 at 10.46.49.png"
## [24] "telecharger_donnees_admissions_med.ipynb"
```

```
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"
```

```r
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.

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

```
## Entrer un nombre:
```

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.

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

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

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

v <- 1:10

avg(x = v)
```

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

```r
# 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.

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

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

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.
sapply(L, avg)
```

```
##           X1           X2           X3           X4           X5           X6
##  0.002825105  0.020374063  0.055152188 -0.044140208 -0.016019744  0.050948281
##           X7           X8           X9          X10
## -0.030072130  0.015579582 -0.007629547  0.019322313
```

```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
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

```r
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 collection of data and functions with their documentations.

```r
# install.packages("pacman", dependencies = TRUE)
# install.packages("ggplot2")
# rownames(installed.packages())
```

Prenvent R from display warning when loading a packages

Do the following setting

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

```r
library(pacman)
source("mypackages.R")
```

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:

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

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

```r
data_list <- list() # creating an empty list.
dir()
```

```
##  [1] "a.csv"
##  [2] "aims-logo.jpg"
##  [3] "calc.png"
##  [4] "donnees_hospitalieres"
##  [5] "mypackages.R"
##  [6] "pipe.jpeg"
##  [7] "R_Intro_DrLema_files"
##  [8] "R_Intro_DrLema.html"
##  [9] "R_Intro_DrLema.pdf"
## [10] "R_Intro_DrLema.Rmd"
## [11] "R_Intro_IndabaX_Chad.html"
```

```
## [12] "R_Intro_IndabaX_Chad.Rmd"
## [13] "rsconnect"
## [14] "Screenshot 2024-02-20 at 17.05.25.png"
## [15] "Screenshot 2024-09-27 at 14.37.41.png"
## [16] "Screenshot 2024-09-30 at 15.36.49.png"
## [17] "Screenshot 2024-09-30 at 15.39.23.png"
## [18] "Screenshot 2024-10-24 at 10.24.29.png"
## [19] "Screenshot 2024-10-24 at 10.25.09.png"
## [20] "Screenshot 2024-10-24 at 10.28.26.png"
## [21] "Screenshot 2024-10-24 at 10.34.32.png"
## [22] "Screenshot 2024-10-24 at 10.37.26.png"
## [23] "Screenshot 2024-10-24 at 10.46.49.png"
## [24] "telecharger_donnees_admissions_med.ipynb"
```

```r
# check the files names in data/csv
dir("./data/csv/")
```

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

```r
# import
# TODO
```

Pipe: `%>%` or `|>`

- Library: `tidyverse` or `dplyr`
- Shortcut: `Crtl + Shift + M`
- Why is it useful?

$$f(g(h(x))) \quad \text{is equivalent to} \quad x \ \%{>}\% \ h() \ \%{>}\% \ g() \ \%{>}\% \ f()$$

```r
library(tidyverse)
```

```r
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
```

Instead of

```r
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
```

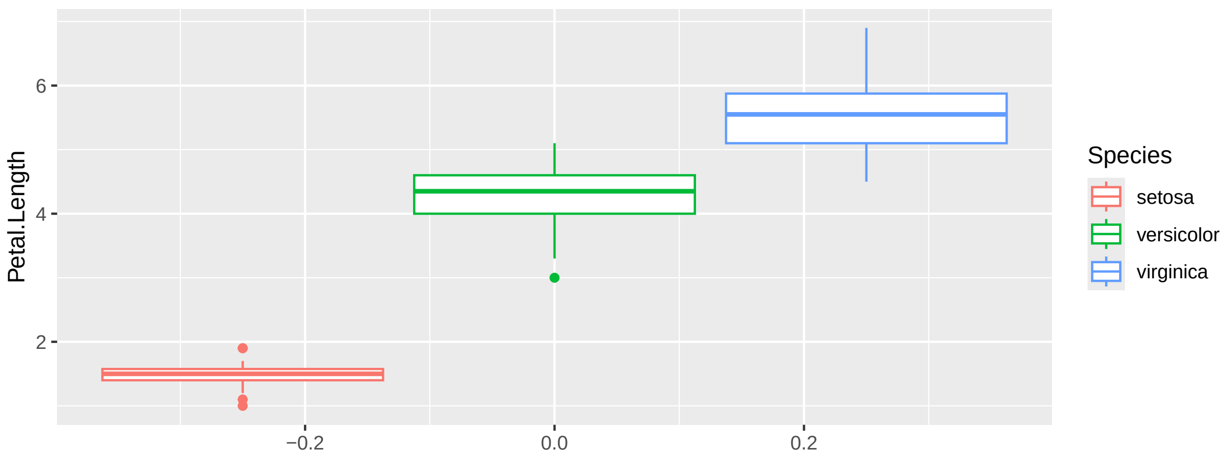
Data manipulation

Data manipulation with tidyverse

Data manipulation with tibble

Data manipulation with reshape2

```r
library(ggplot2)
iris %>% ggplot(aes(y = Petal.Length,  col = Species)) + geom_boxplot()
```

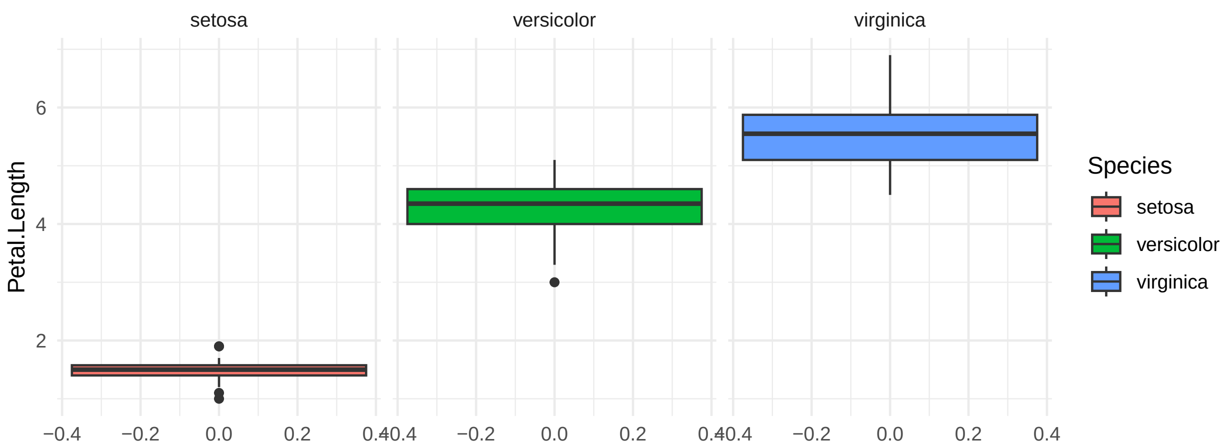


Data display with kabbleExtra

Data display with officer, ...

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

R advanced

Regular expressions

Unsupervised & Supervised Learning

Principal Component Analysis

Clustering: K-means, Hierarchical Clustering

K-Nearest Neighbor

Simple Linear Regression

Logistic Regression

Machine Learning

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 $X = (X_1, \ldots, X_n)$ is given by:

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

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

Include bash code in Rstudio

```bash
#!/bin/bash
for ((i=1; i<=10; i++)); do
  echo $i
done
```

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