Code in class

Ali Nemati

February 21, 2022

Contents

Practicing using R as a calculator

```
my_vector <- c(1,3,5,8,10,-5,3)
length(my_vector)
```

```
## [1] 7
```

```
sum(my_vector)
```

```
## [1] 25
```

Set random seed and generate some random normal draws. It generates the same random vector each time you call the random function

Vectors, matrices, data.frames, . . .

```
set.seed(13579)
my_normal_vector <- rnorm(n=length(my_vector))
my_normal_vector
```

```
## [1] -1.2347155 -1.2528339 -0.2547780 -1.5266466  1.0971147  2.4887442  0.7794803
```

```
#

## Vector multiplication (elementwise)
my_vector * my_normal_vector
```

```
## [1]  -1.234715  -3.758502  -1.273890 -12.213173  10.971147 -12.443721   2.338441
```

```
## Inner product
## The quantity obtained by multiplying the corresponding
## coordinates of each of two vectors and adding the products.
my_vector %*% my_normal_vector
```

```
##           [,1]
## [1,] -17.61441
```

```
## Create matrix: 5 x 4
## Normal distribution: mean=10, sd=4
# The r is for "random", and it is a random variable having the specified
# distribution. For example, rnorm() function
my_matrix <- matrix(rnorm(n=5*4, mean=10, sd=4),
                    nrow=5, ncol=4)

my_matrix
```

```
##           [,1]     [,2]      [,3]      [,4]
## [1,] 10.753500 6.933320 10.987272  3.201161
## [2,]  5.894216 8.250841  6.553208 13.599404
## [3,]  8.973170 8.113796 12.639644 12.884479
## [4,] 12.984201 7.754463  9.852634  9.346003
## [5,] 11.884883 5.139338  6.189902  6.933884
```

```
## Create another matrix: 5 x 4
## rexp distribution
# rexp(m, r)-Returns a vector of m random numbers having the exponential
# distribution
gen_exp <- rexp(n=5*4, rate=1)
gen_exp
```

```
##  [1] 0.33735737 2.82539938 1.12580580 0.44286020 0.02611241 1.30599715
##  [7] 0.19481413 0.09892251 2.12785120 1.52468762 0.58872486 0.43111999
## [13] 4.14845884 0.40606674 2.44914039 0.30681808 1.29253900 0.12032613
## [19] 0.82349666 1.08946855
```

```
my_matrix_2 <- matrix(gen_exp, ncol=4, nrow=5)
my_matrix_2
```

```
##           [,1]       [,2]      [,3]      [,4]
## [1,] 0.33735737 1.30599715 0.5887249 0.3068181
## [2,] 2.82539938 0.19481413 0.4311200 1.2925390
## [3,] 1.12580580 0.09892251 4.1484588 0.1203261
## [4,] 0.44286020 2.12785120 0.4060667 0.8234967
## [5,] 0.02611241 1.52468762 2.4491404 1.0894686
```

```
## Double check the product functions (elementwise multiplication)
my_matrix * my_matrix_2
```

```
##           [,1]       [,2]      [,3]       [,4]
## [1,]  3.6277725  9.0548966  6.468480  0.9821741
## [2,] 16.6535149  1.6073804  2.825219 17.5777600
## [3,] 10.1020464  0.8026371 52.435041  1.5503396
## [4,]  5.7501857 16.5003440  4.000827  7.6964023
## [5,]  0.3103429  7.8358856 15.159939  7.5542487
```

```
## Make use of head and tail commands for sanity checks
my_huge_matrix <- matrix(rgamma(10000,
                                shape=1),
                         ncol=10)
head(my_huge_matrix)
```

```
##           [,1]      [,2]      [,3]       [,4]      [,5]       [,6]       [,7]
## [1,] 0.1585023 0.2594988 0.3045265 0.08611515 1.3593091 0.46796143 0.01747011
## [2,] 2.3844915 2.1434004 0.2076556 0.63440774 0.4867942 2.49800677 1.33030870
## [3,] 2.0498783 0.5644282 0.6914348 0.09216033 0.1677927 0.06666595 1.04110944
## [4,] 0.6544284 0.7128908 2.6595979 0.62706588 0.5343128 0.61398043 1.22492017
## [5,] 0.2356970 0.0428360 0.9375176 1.54803682 2.1684759 0.96108695 2.83205475
```

```
## [6,] 1.6757674 0.1273529 0.3218827 0.05611610 0.4491089 1.35358352 0.53563203
##            [,8]       [,9]      [,10]
## [1,] 1.11846385 0.48473271 2.56600850
## [2,] 1.00036235 1.47877854 0.27404199
## [3,] 1.31866862 0.00207013 0.01210107
## [4,] 0.67984458 0.61003525 0.69062251
## [5,] 0.09203886 3.78459495 1.74063865
## [6,] 2.12307883 0.55484225 0.64885783
```

```
tail(my_huge_matrix)
```

```
##              [,1]      [,2]      [,3]      [,4]      [,5]       [,6]      [,7]
##  [995,] 2.2065900 2.6204056 0.3617654 0.3113698 0.9299397 1.71883278 0.6832110
##  [996,] 0.1513955 0.4863449 0.6484172 1.5576926 1.6252196 0.22696258 0.1349167
##  [997,] 1.5559238 0.1786538 0.4939397 3.0058030 0.5783891 1.21986101 0.4762442
##  [998,] 0.3488299 0.3241757 0.6761012 0.8329858 0.6561114 0.04526154 0.6484562
##  [999,] 0.3982543 1.2403230 0.1033660 2.1795944 0.9295799 0.35622295 0.6209044
## [1000,] 1.5952870 0.2751454 1.0844354 0.1188376 0.2601905 1.78664851 2.0429866
##               [,8]      [,9]      [,10]
##  [995,] 0.06476413 0.3178026 0.43963115
##  [996,] 2.13606096 0.3472772 0.06109482
##  [997,] 1.99384835 0.4734880 1.22129446
##  [998,] 0.46446190 0.4860314 0.22367593
##  [999,] 0.35793411 1.4893320 1.01059960
## [1000,] 1.20263627 0.3589378 0.83118101
```

```
## Make sequence of values
1:10
```

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

```
0:10
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
10:0
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1  0
```

```
-5:5
```

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

```
seq(from=-5, to=50, by=5)
```

```
##  [1] -5  0  5 10 15 20 25 30 35 40 45 50
```

```
seq(from=-5, to=50, length=50)
```

```
##  [1] -5.0000000 -3.8775510 -2.7551020 -1.6326531 -0.5102041  0.6122449
##  [7]  1.7346939  2.8571429  3.9795918  5.1020408  6.2244898  7.3469388
## [13]  8.4693878  9.5918367 10.7142857 11.8367347 12.9591837 14.0816327
## [19] 15.2040816 16.3265306 17.4489796 18.5714286 19.6938776 20.8163265
## [25] 21.9387755 23.0612245 24.1836735 25.3061224 26.4285714 27.5510204
## [31] 28.6734694 29.7959184 30.9183673 32.0408163 33.1632653 34.2857143
## [37] 35.4081633 36.5306122 37.6530612 38.7755102 39.8979592 41.0204082
## [43] 42.1428571 43.2653061 44.3877551 45.5102041 46.6326531 47.7551020
## [49] 48.8775510 50.0000000
```

```
## Data frame versus matrix
my_df <- data.frame(my_matrix,
                    letters=c("A", "b", "c", "d","e"))

## Character vectors
my_name_vector <- c("Anne", "Bob", "Charles")

## Check the class of objects
class(my_name_vector)
```

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

```
class(my_normal_vector)
```

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

```
class(my_huge_matrix)
```

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

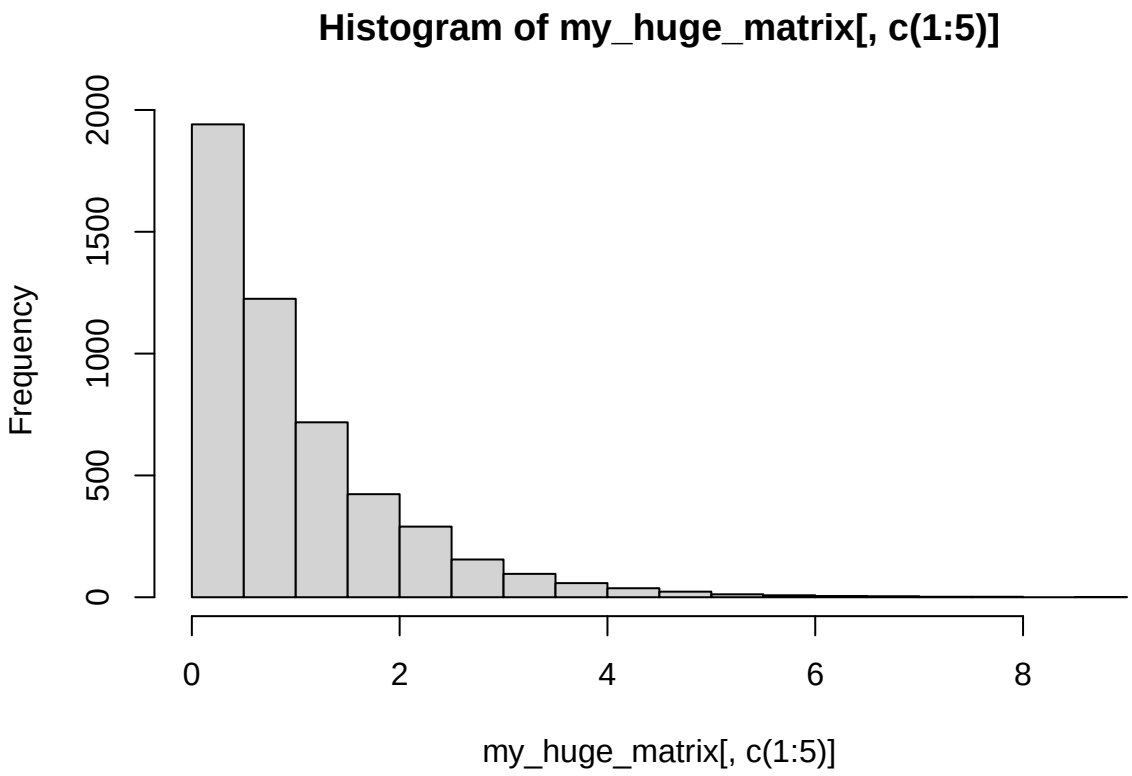
```
class(my_df)
```

```
## [1] "data.frame"
```

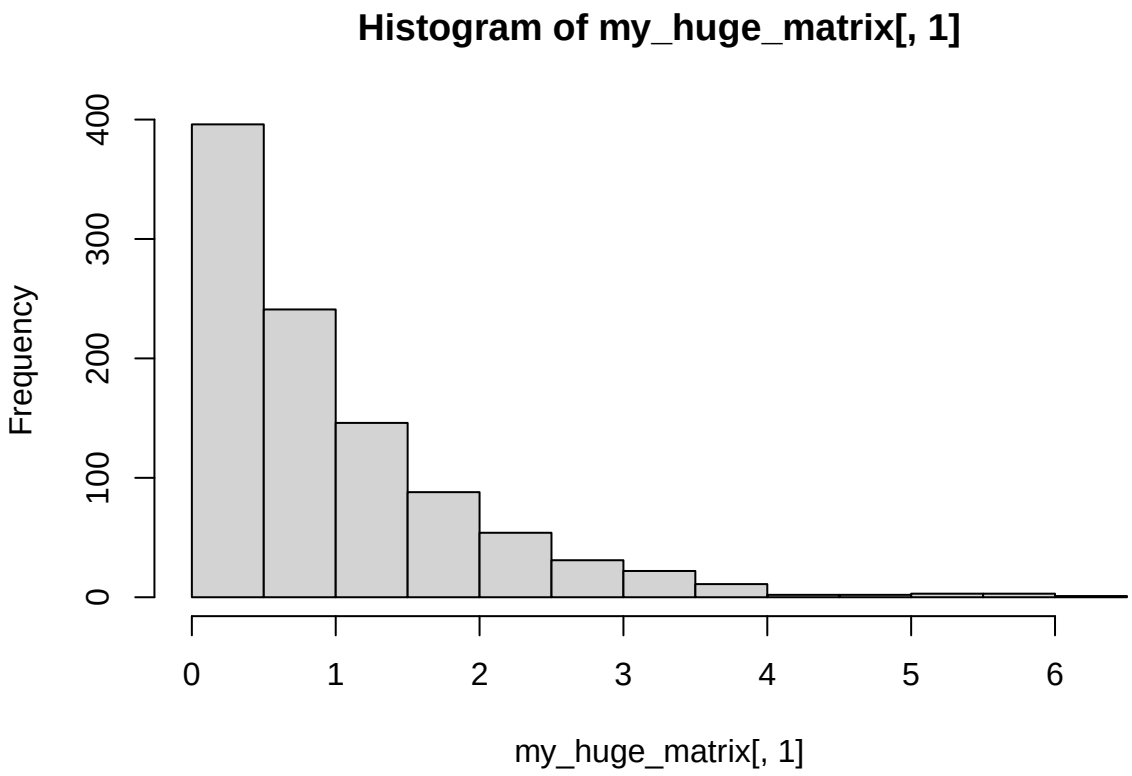
Basic plotting

Histogram

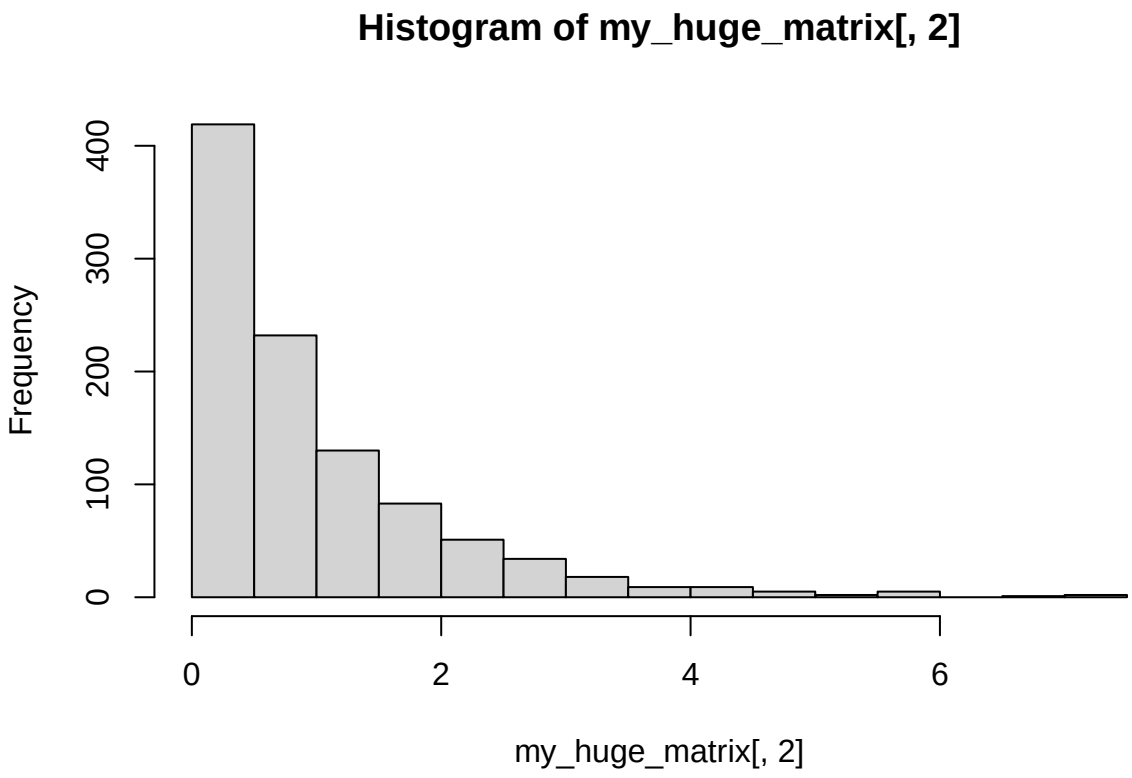
```
#-------------------------------------------------------------
##

## Histogram: hist
## ?hist
## Histogram of first five columns of my_huge_matrix

hist(my_huge_matrix[,c(1:5)] )
```



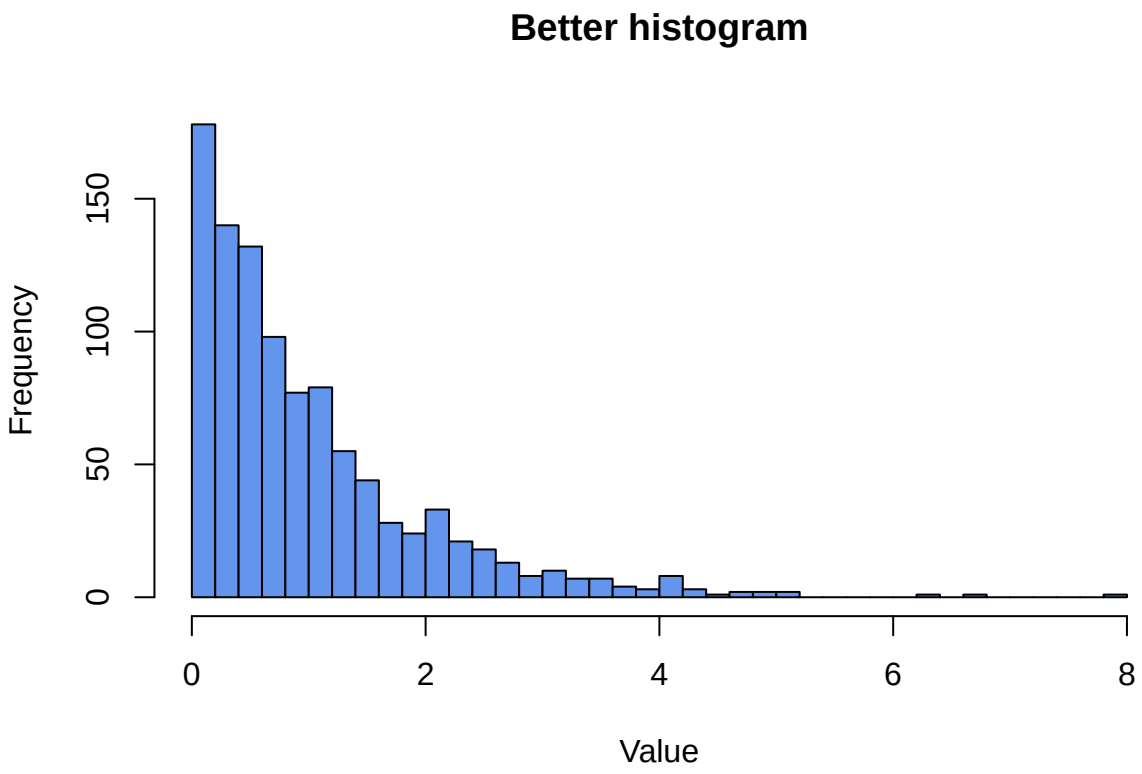

```
hist(my_huge_matrix[,1] ,w=3)
```



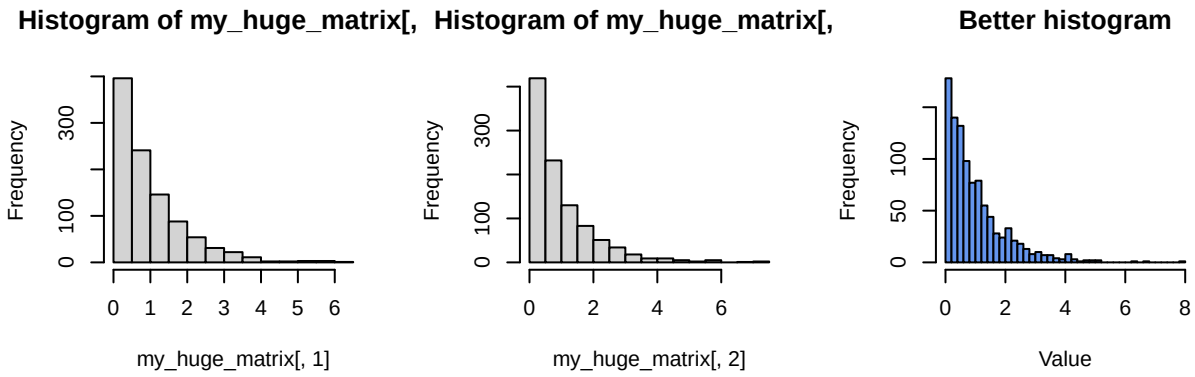

```
hist(my_huge_matrix[,2] ,w=3)
```




```
hist(my_huge_matrix[,3], main="Better histogram",
     xlab="Value", breaks=50,
     col="cornflowerblue" ,w=3)
```

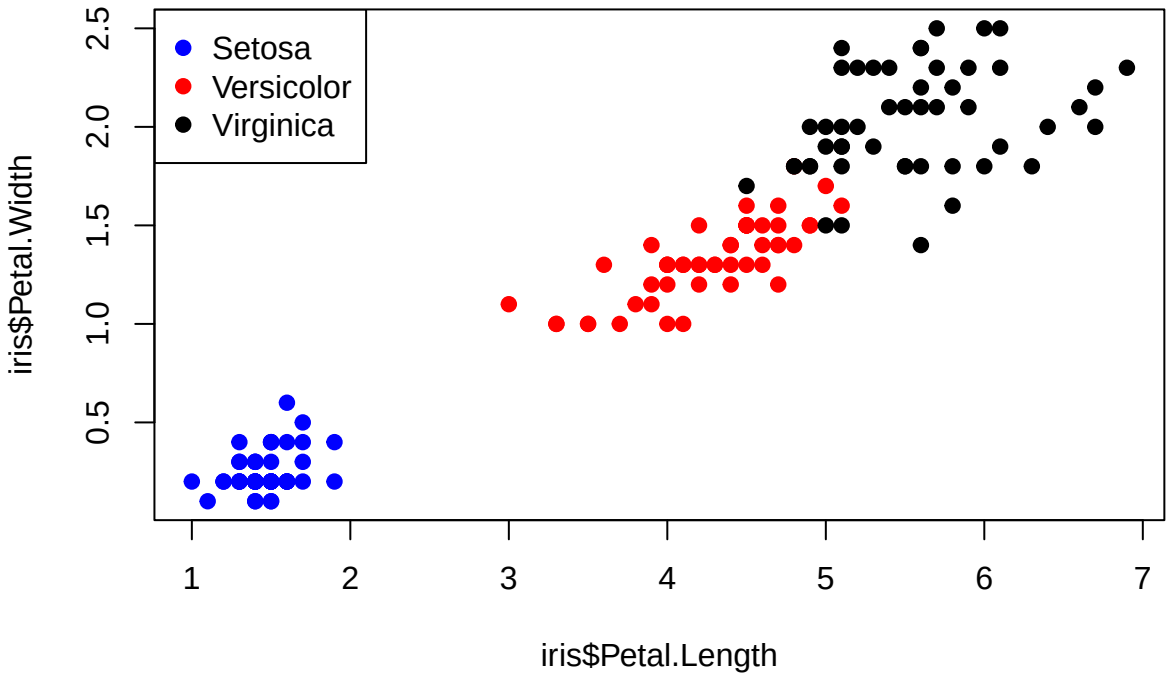



```
#To put all the 6 plots in a 2 x 3 matrix.
par(mfrow=c(2,3))
hist(my_huge_matrix[,1])
hist(my_huge_matrix[,2])
hist(my_huge_matrix[,3], main="Better histogram",
     xlab="Value", breaks=50,
     col="cornflowerblue")
```

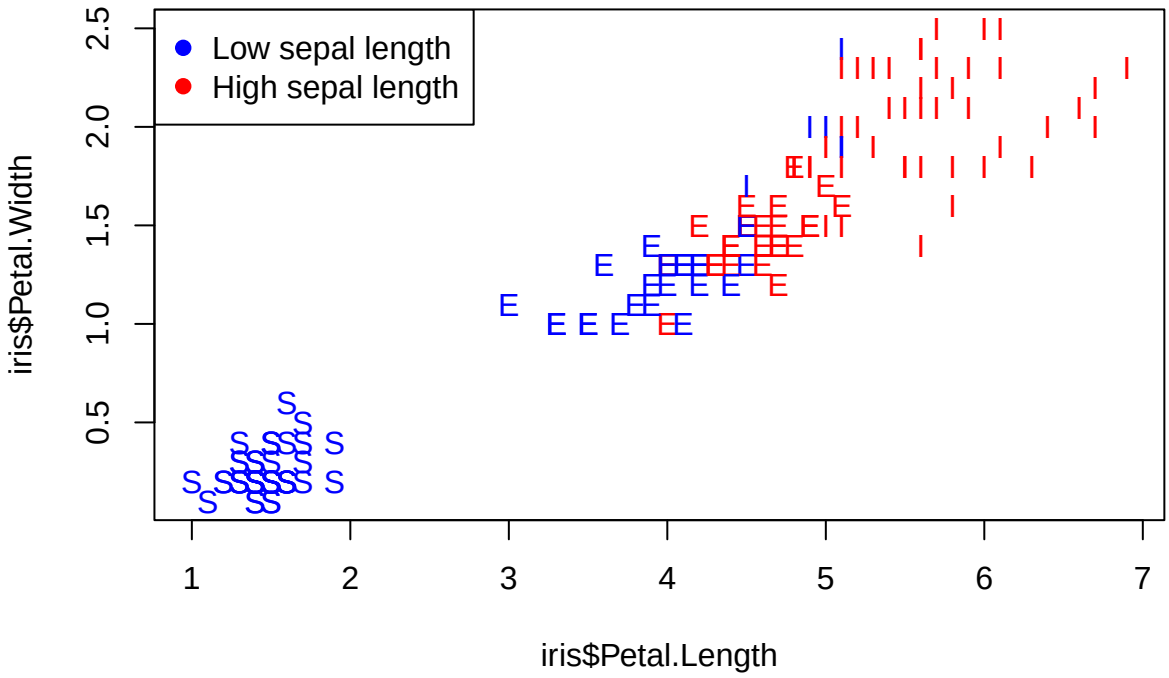


Scatterplots

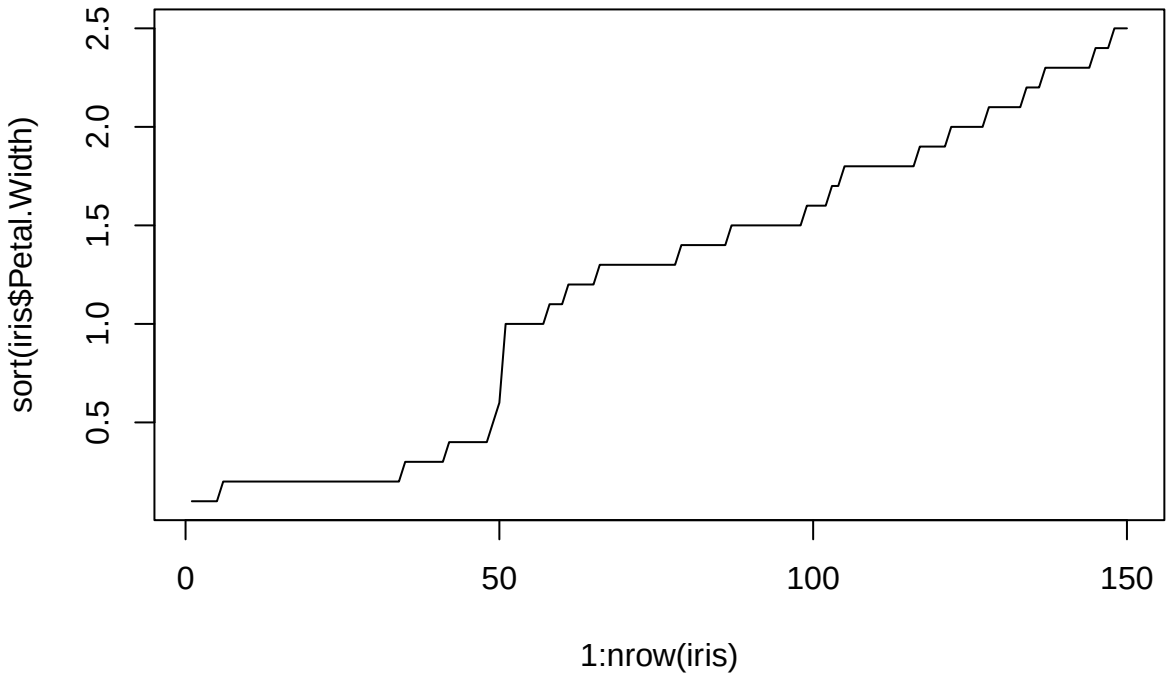
```
data(iris)
# ?plot
## Plot petal length vs petal width
col_vector <- rep(NA, nrow(iris))
setosa_index <- which(iris$Species == "setosa")
versicolor_index <- which(iris$Species == "versicolor")
virginica_index <- which(iris$Species == "virginica")
col_vector[setosa_index] <- "blue"
col_vector[versicolor_index] <- "red"
col_vector[virginica_index] <- "black"

##The pch arguement in the plot function can be varied to change the marker.
plot(x=iris$Petal.Length,
     y=iris$Petal.Width,
     col=col_vector, pch=19)
legend("topleft", pch=19,
       col=c("blue","red","black"),
       legend=c("Setosa", "Versicolor", "Virginica"))
```




```
## Plot Petal length vs width, coloring by sepal length
col_vector <- rep(NA, nrow(iris))
high_index <- which(iris$Sepal.Length >
                    mean(iris$Sepal.Length))
col_vector[high_index] <- "red"
col_vector[-high_index] <- "blue"
shape_vector <- rep(NA, nrow(iris))
shape_vector[setosa_index] <- "S"
shape_vector[versicolor_index] <- "E"
shape_vector[virginica_index] <- "I"

plot(x=iris$Petal.Length,
     y=iris$Petal.Width,
     col=col_vector, pch=shape_vector)
legend("topleft", pch=19,
       col=c("blue","red"),
       legend=c("Low sepal length",
                "High sepal length"))
```

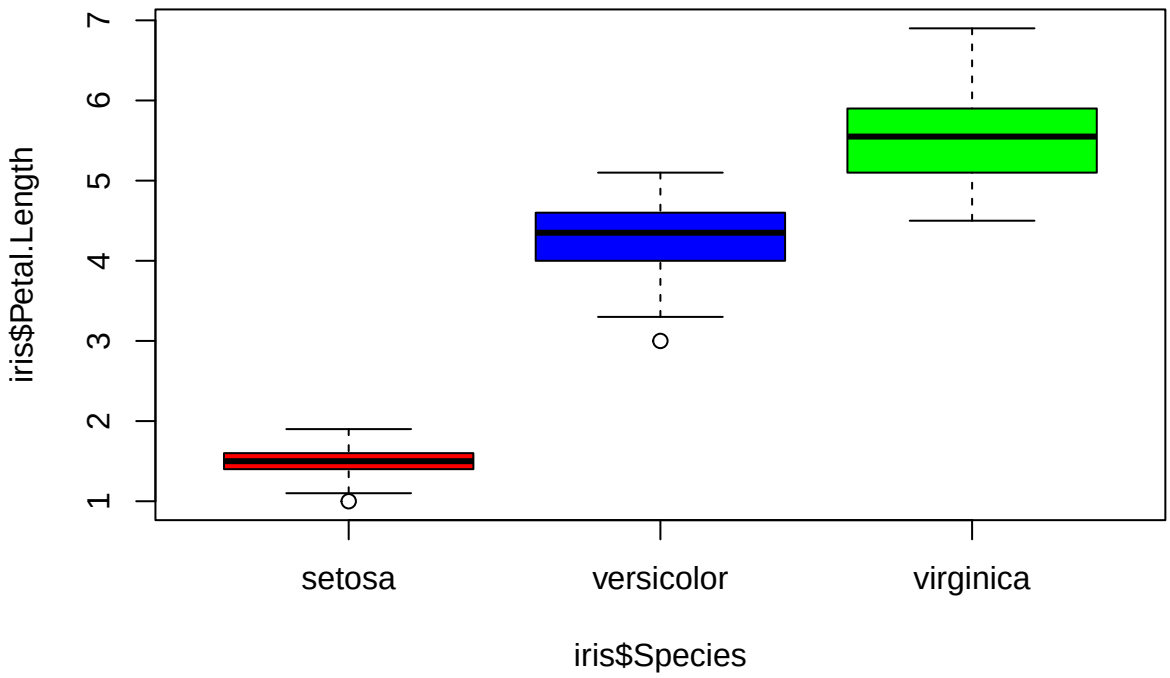



```
plot(x=1:nrow(iris), y=sort(iris$Petal.Width),
     type="l")
```

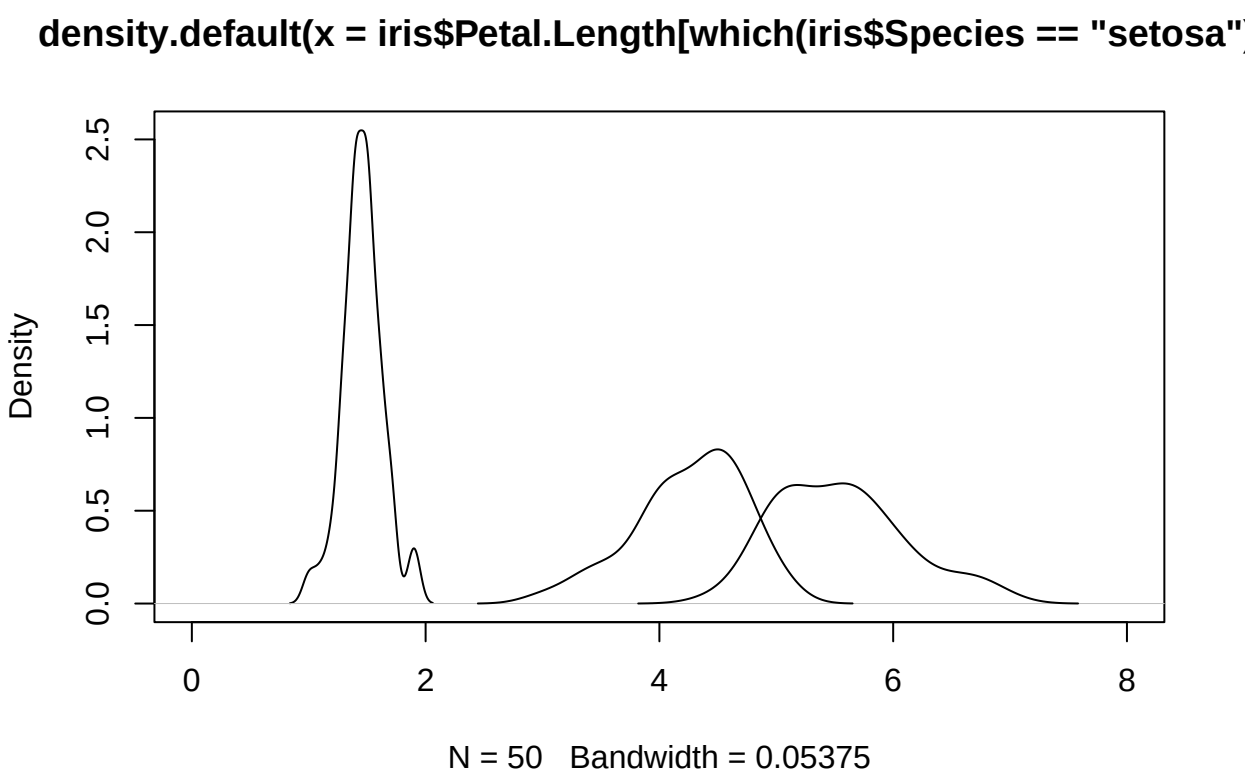


Boxplots

```
boxplot(iris$Petal.Length ~ iris$Species,
        col=c("red", "blue", "green"))
```




```
## Density plots
## Hint: which function to select species
## lines()
plot(density(iris$Petal.Length[which(iris$Species ==
                                     "setosa")]),
     xlim=c(0,8))
lines(density(iris$Petal.Length[which(iris$Species ==
                                     "virginica")]))
lines(density(iris$Petal.Length[which(iris$Species ==
                                     "versicolor")]))
```


density.default(x = iris$Petal.Length[which(iris$Species == "setosa"
Density
0.0
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8
N = 50 Bandwidth = 0.05375

Help

```
#R is case sensitive.
#R index starts from 1

####Help
#Help Home Page, Special Character Help, Search Help, Search Function (with partial name)
#
# help.start()
#
# help('$')
#
# ?"$"

# apropos('med')

example(median)
```

```
##
## median> median(1:4)                # = 2.5 [even number]
## [1] 2.5
##
## median> median(c(1:3, 100, 1000))  # = 3 [odd, robust]
## [1] 3
```

```
# help(package = "dplyr")

#Get help on a function of a package
# help("tally", package = "dplyr")
```

R basics

```
####Basics

# The arrow operator (<-) sets the left-hand side equal to the right-hand side.
# It computes a1 and assigns it to the variable a2
#a2<- a1

a1 <- 6 + 7
a2 <- a1
# View(a2)

rm(a2)

#Comments starts with #


####R Built-in Data Sets
# data()
```

```r
# Loading datasets
data(mtcars)

# Print the first 6 rows of a dataset
head(mtcars, 6)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
# ?mtcars

# Number of rows (observations)
nrow(mtcars)
```

```
## [1] 32
```

```r
# Number of columns (variables)
ncol(mtcars)
```

```
## [1] 11
```

```r
data(iris)


# You could find a large number of datasets at https://archive.ics.uci.edu/ml/datasets.php
```

Using Packages

You could find a large number of datasets at https://archive.ics.uci.edu/ml/datasets.php

Using Packages

R packages: collections of functions and data sets developed by the community. Examples: "dplyr" or "data.table"

A number of packages are automatically loaded when you start R (e.g., "base", "utils", "graphics", and "stats")

First you install packages and then you should load them using command "library()"

install & load library()

install packages() install.packages("tidyr")

library(tidyr) browseVignettes()

browseVignettes(package="packagename")

vignette(package = "ggplot2")

Unload packages

detach(packageName)

require(ggplot2)

Working Directory

current working directory where inputs are found, and outputs are sent.

getwd()

Change the current working directory.

setwd(C://file/path)

What working directory are we in?

Set working directory to a new place !!! You need to change this! setwd('/Users/kravvaz/Desktop/PH718_spring20/Lecture2')

```
### Try the following commands
getwd() ## Get working directory
```

```
## [1] "C:/Users/anemati/OneDrive - mcw.edu/UWM/Spring/PH 718 R/R code In class"
```

```
dir()   ## What else is in that directory?
```

```
##  [1] "code-in-class.html"       "code-in-class.pdf"
##  [3] "code-in-class.Rmd"        "code-in-class_files"
##  [5] "code in class.Rmd"        "comment"
##  [7] "cowplot.R"                "first package"
##  [9] "first_rmarkdown.docx"     "first_rmarkdown.pdf"
## [11] "first_rmarkdown.Rmd"      "ggplot.pdf"
## [13] "ggplot.Rmd"               "Lecture1_Rcode1.R"
## [15] "Lecture1_Rcode1.R.ipynb"  "Lecture1_Rcode2.R"
## [17] "Lecture2_Rcode.R"         "Lecture3_Rcode.R"
## [19] "Lecture4_Rcode.R"         "mypackage"
## [21] "R class 02-03-2021.r"     "R code for bio.Rproj"
## [23] "test-figure"              "Untitled.ipynb"
## [25] "W2_1_Lecture2_Rcode.R"    "W2_2_errorExamples.R"
## [27] "W2_3_first_rmarkdown.Rmd" "weel1.R"
```

```
objects()
```

```
##  [1] "a1"               "col_vector"       "gen_exp"          "high_index"
##  [5] "iris"             "mtcars"           "my_df"            "my_huge_matrix"
##  [9] "my_matrix"        "my_matrix_2"      "my_name_vector"   "my_normal_vector"
## [13] "my_vector"        "setosa_index"     "shape_vector"     "versicolor_index"
## [17] "virginica_index"
```

```
ls()
```

```
##  [1] "a1"               "col_vector"       "gen_exp"          "high_index"
##  [5] "iris"             "mtcars"           "my_df"            "my_huge_matrix"
##  [9] "my_matrix"        "my_matrix_2"      "my_name_vector"   "my_normal_vector"
## [13] "my_vector"        "setosa_index"     "shape_vector"     "versicolor_index"
## [17] "virginica_index"
```

```
rm(list=c("x", "y"))
```

```
## Warning in rm(list = c("x", "y")): object 'x' not found
```

```
## Warning in rm(list = c("x", "y")): object 'y' not found
```

```
ls()
```

```
##  [1] "a1"               "col_vector"       "gen_exp"          "high_index"
##  [5] "iris"             "mtcars"           "my_df"            "my_huge_matrix"
##  [9] "my_matrix"        "my_matrix_2"      "my_name_vector"   "my_normal_vector"
## [13] "my_vector"        "setosa_index"     "shape_vector"     "versicolor_index"
## [17] "virginica_index"
```

Data Types

- Numeric (float/double, integer, etc.)
- Character(string)
- Date (common packages: lubridate, chron)
- Logical (TRUE = 1, FALSE = 0, ==, !=)
- Factor (Character strings with preset levels.)

Example of data type

```
a1 <- 6 + 7
a2 <- a1

is.numeric(a2)
```

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

```
nchar(mtcars$hp)
```

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

```
#Stores just as a date.
date1 <- as.Date('2020-01-23')

#Internally, Date objects are stored as the number of days since January 1, 1970
#To convert a Date object to its internal form, number of days since 1/1/1970.
as.numeric(date1)
```

```
## [1] 18284
```

```
#Stores a date and time.
date2 <- as.POSIXct("2020-01-23 17:30")
#In numeric form, number of seconds since 1/1/1970.
as.numeric(date2)
```

```
## [1] 1579822200
```

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

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

```
class(iris$Species)
```

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

```
levels(iris$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

```
str(iris$Species)
```

```
##  Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Data Structures

- Vector (Groups of elements of the SAME type)
- Factor (A vector of categorical variables, encoded using integers)
- List (Store any number of items of ANY type)
- Data Frame (DATA.FRAME)(Each column is a variable, each row is an observation)
- Data Table (DATA.TABLE) (Extends and enhances the functionality of data.frames, index like a database, data.table does not turn character data into factors but data.frame does.)
- Matrix (Similar to data.frame except every element must be the SAME type)
- Array (Multidimensional vector of the SAME type)

```
v1 <- c(1, 2, 3, 4, 5, 6, 7, 8)
length(v1)
```

```
## [1] 8
```

```
unique(v1)
```

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

```
rev(v1)
```

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

```
v1[order(v1)]
```

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

```
v1[v1 > 3]
```

```
## [1] 4 5 6 7 8
```

```
v1[1:3]
```

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

```
v1[c(1,6)]
```

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

```
v1[-(2:4)]
```

```
## [1] 1 5 6 7 8
```

```
v1[v1 == 5]
```

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

```r
v1[v1 %in% c(1, 2, 5)]
```

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

```r
v1 <- c(1, 2, 3, 4, 5, 6, 7, 8)
v1 <- 1:8
v1 <- seq(from=1, to=8, by=1)

#row.names(data.frame)
#nrow()
#length()
#subset(data.frame, [condition])
#rbind()
```

Vector

```r
# with numerics from 1 up to 10
my_vector <- 1:10
my_vector
```

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

Matrix

```r
# with numerics from 1 up to 9
my_matrix <- matrix(1:9, ncol = 3)
my_matrix
```

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

data frame

```r
#First 10 elements of the built-in data frame mtcars
my_df <- mtcars[1:10,]
my_df
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
```

```
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

list

```
#Construct list with these different elements:
my_list <- list(my_vector, my_matrix, my_df)
my_list
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## [[3]]
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

Factor

```
#How to create a factor?
factor1 <- factor(c("single", "married", "married", "single"))
factor1
```

```
## [1] single  married married single
## Levels: married single
```

```
factor2 <- factor(c("single", "married", "married", "single"),
                  levels = c("single", "married", "divorced"));
factor2
```

```
## [1] single  married married single
## Levels: single married divorced
```

```r
class(factor2)
```

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

```r
levels(factor2)
```

```
## [1] "single"   "married"  "divorced"
```

```r
factor3 <- factor(c("single","married","married","single"))
str(factor3)
```

```
##  Factor w/ 2 levels "married","single": 2 1 1 2
```

```r
#How to access components of a factor?
factor3
```

```
## [1] single  married married single
## Levels: married single
```

```r
factor3[3]           # access 3rd element
```

```
## [1] married
## Levels: married single
```

```r
factor3[c(2, 4)]     # access 2nd and 4th element
```

```
## [1] married single
## Levels: married single
```

```r
factor3[-1]          # access all but 1st element
```

```
## [1] married married single
## Levels: married single
```

```r
factor3[c(TRUE, FALSE, FALSE, TRUE)]  # using logical vector
```

```
## [1] single single
## Levels: married single
```

```r
#How to modify a factor?
factor3
```

```
## [1] single  married married single
## Levels: married single
```

```
factor3[2] <- "divorced"    # modify second element;
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "divorced"): invalid factor level,
## NA generated
```

```
factor3[3] <- "widowed"    # cannot assign values outside levels
```

```
## Warning in `[<-.factor`(`*tmp*`, 3, value = "widowed"): invalid factor level, NA
## generated
```

```
factor3
```

```
## [1] single <NA>   <NA>   single
## Levels: married single
```

```
#Add a value to the level first.
levels(factor3) <- c(levels(factor3), "widowed")    # add new level
factor3[3] <- "widowed"
factor3
```

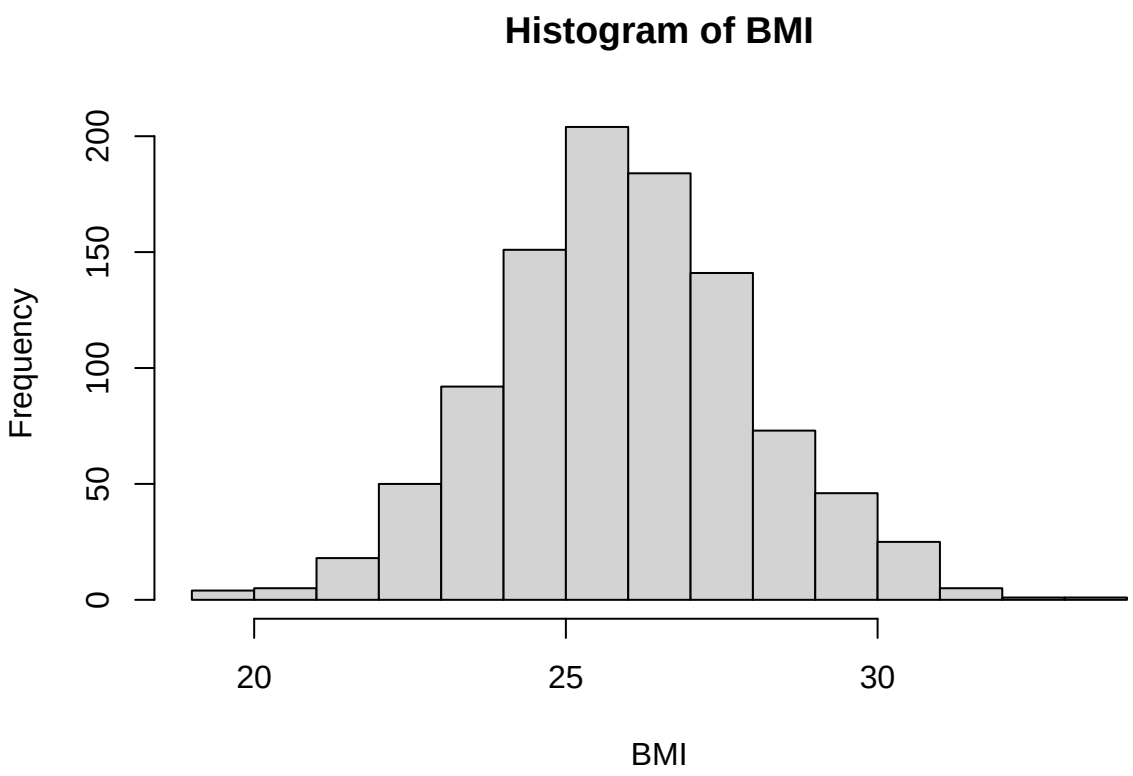
```
## [1] single  <NA>    widowed single
## Levels: married single widowed
```

```
#NA
#Missing data is encoded using value NA.
is.na(factor3) #tests whether factor3 is an NA
```

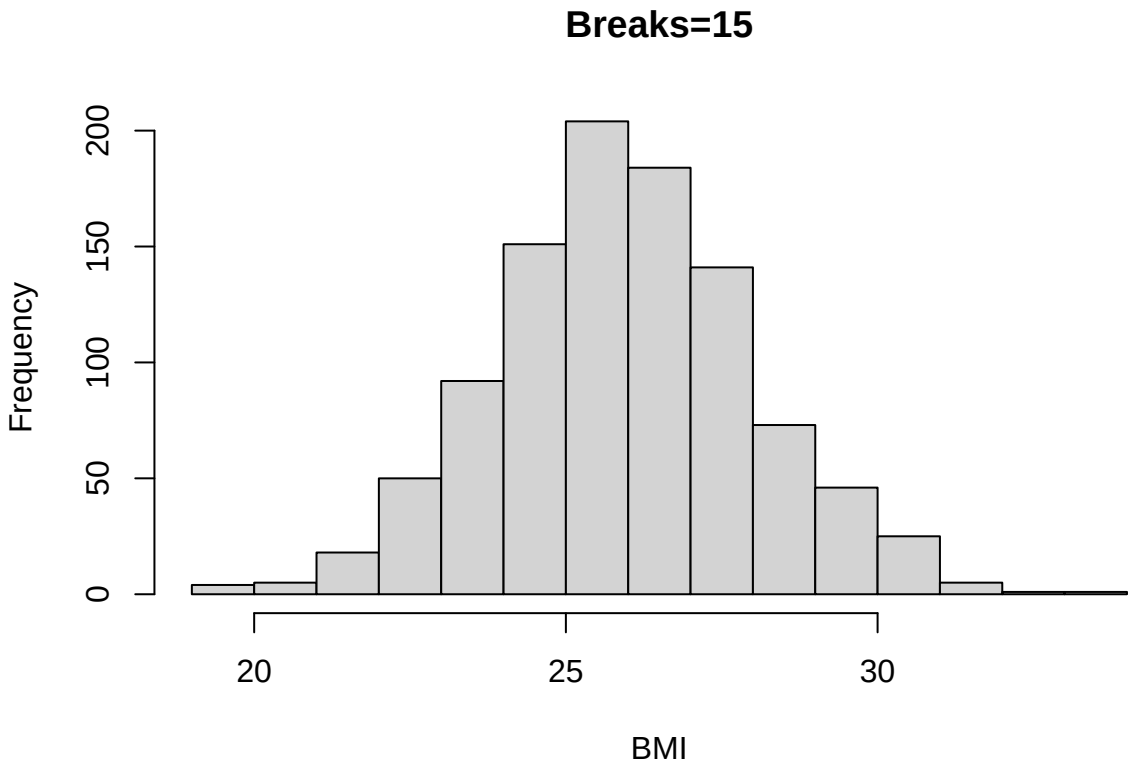
```
## [1] FALSE  TRUE FALSE FALSE
```

Plot

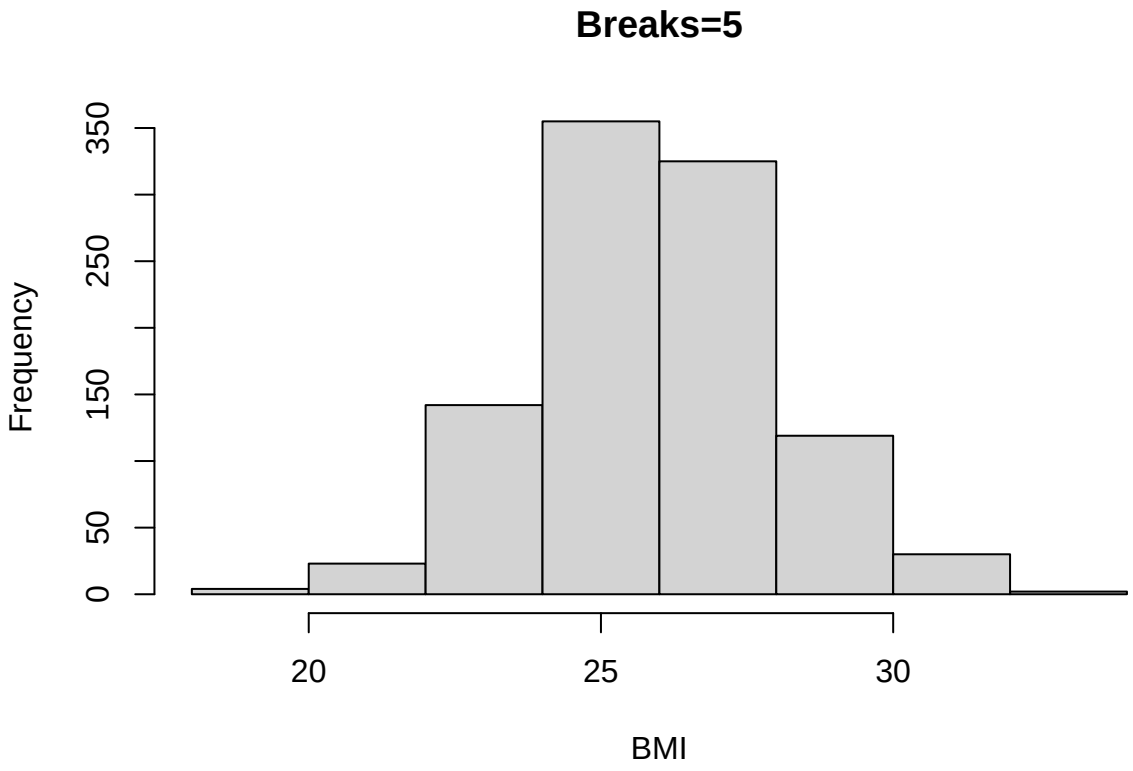
```
###
set.seed(2020)
BMI<-rnorm(n=1000, m=26, sd=2)
hist(BMI)
```



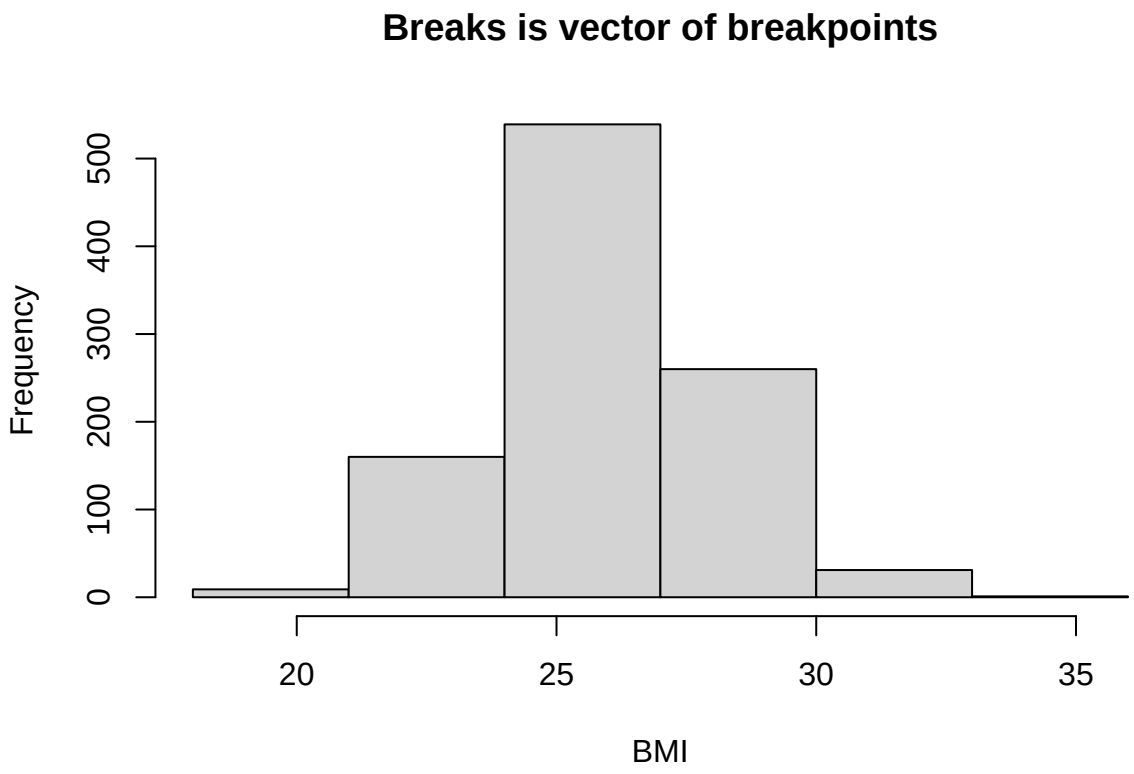

```
hist(BMI, breaks=15, main="Breaks=15")
```



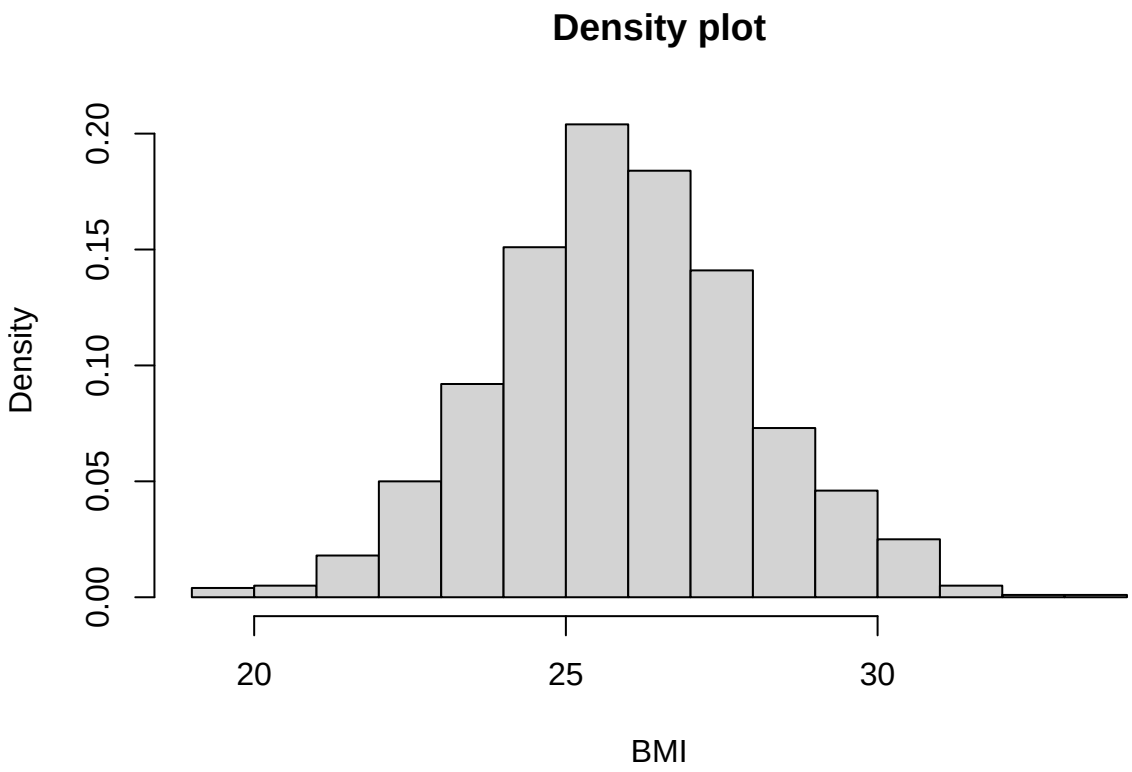

```
hist(BMI, breaks=5, main="Breaks=5")
```




```
#breaks() option with a vector gives you more control over exactly the breakpoints between bins by dict
hist(BMI, breaks=c(18, 21, 24, 27, 30, 33, 36), main="Breaks is vector of breakpoints")

hist(BMI, breaks=seq(18,36,by=3), main="Breaks is vector of breakpoints")
```




```
#Density plot: freq=FALSE option
hist(BMI, freq=FALSE, main="Density plot")
```



Functions in R

Here is a simple example

```
#
example.sum <- function(a, b) {
  return(a + b)
}

x <- 1:10
y <- 11:20
example.sum(x, y)
```

```
##  [1] 12 14 16 18 20 22 24 26 28 30
```

```
### Naming conventions. Why not write example.sum() as:
example.sum.1 <- function(x, y, z) {
  tmp <- x + y
  tmp2 <- tmp + z
  return(tmp2)
}
x <- 1:10
```

```
y <- 11:20
z <- 100
example.sum.1(x, y, z)
```

```
##  [1] 112 114 116 118 120 122 124 126 128 130
```

```
### Try this

example.diff.sum <- function(x, y) {
  newy = example.sum(x, y) - 10
  return(newy)
}
# example.diff.sum(x, y)
example.diff.sum(10, 5)
```

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

```
example.sum = function(a, b) {
  return(a + b)
}

example.diff.sum = function(c, d) {
  e = example.sum(c, d) - 10
  return(e)
}

x <- 1:10
y <- 11:20
# example.diff.sum(x, y)
example.diff.sum(10, 5)
```

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

Example together: Hardy-Weinberg problem #3.6 from book

```
HWE <- function(p) {
  stopifnot(is.numeric(p))
  stopifnot(p >= 0)
  stopifnot(p <= 1)
  prob_AA <- p ^ 2
  prob_AB <- 2 * p * (1 - p)
  prob_BB <- (1 - p) ^ 2
  return(c(prob_AA, prob_AB, prob_BB))
}
HWE(0)
```

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

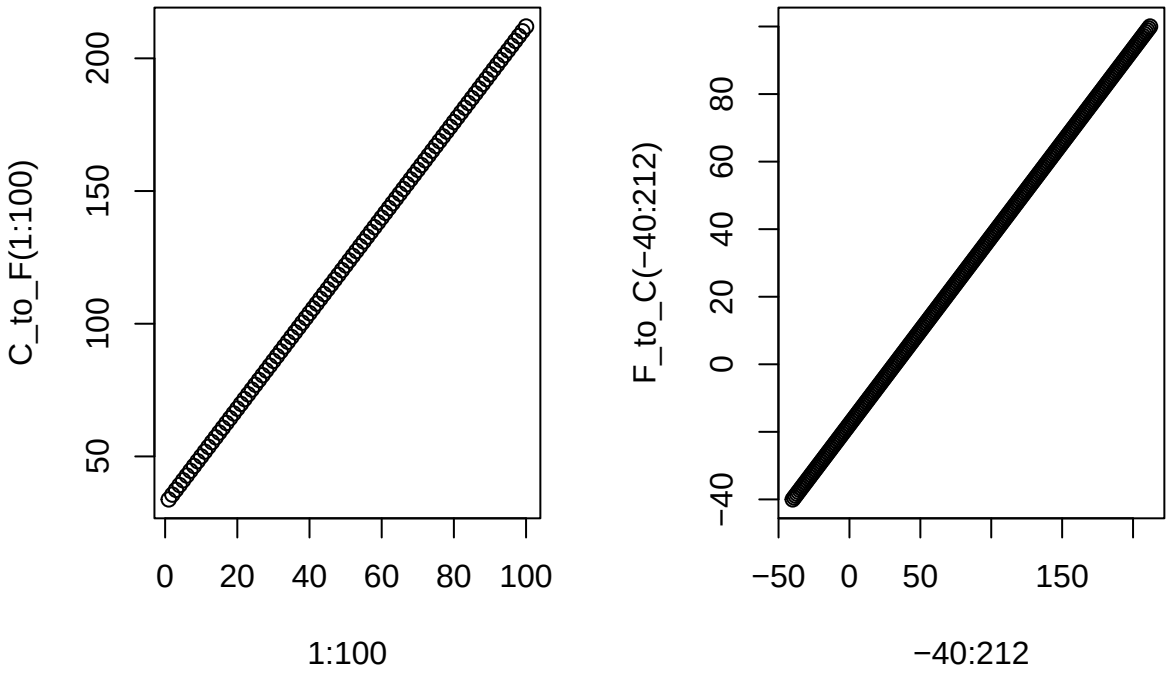
```
HWE(0.5)
```

```
## [1] 0.25 0.50 0.25
```

```
# HWE(-5) # Error
# HWE("kourosh") # Error
```

Write a function to convert Celsuis to Fahrenheit and vice versa

```
C_to_F <- function(temp) {
  return((temp * 1.8) + 32)
}

F_to_C <- function(temp) {
  return((temp - 32) / 1.8)
}

par(mfrow = c(1, 2))
plot(1:100, C_to_F(1:100))
plot(-40:212, F_to_C(-40:212))
```



type_of_conversion can be "F_to_C" or "C_to_F"

```
temp_conversion <- function(temp, type_of_conversion = "F_to_C") {
  if (!type_of_conversion %in% c("F_to_C", "C_to_F")) {
    stop("STOP!!! I can only convert C to F or F to C.")
  }
  if (type_of_conversion == "F_to_C") {
    new_temp <- F_to_C(temp)
  }
  if (type_of_conversion == "C_to_F") {
    new_temp <- C_to_F(temp)
  }
  return(list(temp = new_temp, conversion = type_of_conversion))
}

temp_conversion(100, type = "C_to_F")
```

```
## $temp
## [1] 212
##
## $conversion
## [1] "C_to_F"
```

```
temp_conversion(0, type = "F_to_C")
```

```
## $temp
## [1] -17.77778
##
## $conversion
## [1] "F_to_C"
```

Loops in R

An introductory example

This "initializes" the object "w" so that R knows what w "stands for."

w <- NULL

for" loop

```
w <- NULL
for (i in 1:10) {
  w[i] <- i + 10
}
w
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

What do you think w looks like?

```
#w <- NULL
for (hello in 1:10) {
  w[hello] <- hello
}
w
```

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

Now what does w look like?

embed loops

```
w <- NULL
counter <- 1
for (j in 11:20) {
  for (i in 1:10) {
    w[counter] <- i + j
    counter <- counter + 1
  }
}
w
```

```
##   [1] 12 13 14 15 16 17 18 19 20 21 13 14 15 16 17 18 19 20 21 22 14 15 16 17 18
##  [26] 19 20 21 22 23 15 16 17 18 19 20 21 22 23 24 16 17 18 19 20 21 22 23 24 25
##  [51] 17 18 19 20 21 22 23 24 25 26 18 19 20 21 22 23 24 25 26 27 19 20 21 22 23
##  [76] 24 25 26 27 28 20 21 22 23 24 25 26 27 28 29 21 22 23 24 25 26 27 28 29 30
```

Think through these two loops carefully.

What is the difference between this loop and the last loop?

```
w <- NULL
w
```

```
## NULL
```

```
counter <- 1
for (j in 11:20) {
  for (i in 1:10) {
    w[counter] <- i + j
  }
  counter <- counter + 1
}
w
```

```
##  [1] 21 22 23 24 25 26 27 28 29 30
```

An example of a "while" loop

```
w <- 100
z <- 5

while (w > 20) {
  w.plus.z <- w + z
  w <- w - 1
}
w
```

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

```
w <- 100
z <- 5
while (w < 20) {
  # if w > 20 => inf loop
  w.plus.z <- w + z
  w <- w + 1
}
w
```

```
## [1] 100
```

Indices and the which() function

```
x <- rnorm(100)
x[1:10]
```

```
##  [1]  0.00465041 -1.22874970 -0.14059798 -0.20732697 -0.92153058  0.36047424
##  [7]  1.66660243  1.44804634 -0.03285159 -1.62843554
```

```
x[20:30]
```

```
##  [1]  0.18713551 -0.57543743 -0.08766336  0.31167076  0.24438772 -0.45803324
##  [7]  1.32593062 -0.57952274 -0.82407695  1.08562056  1.41590793
```

```
index <- which(x < 0)

index
```

```
##  [1]   2   3   4   5   9  10  11  13  14  15  17  18  19  21  22  25  27  28  33
## [20]  35  38  41  44  45  46  49  52  57  58  60  63  64  65  66  68  72  73  74
## [39]  75  78  80  81  83  84  85  89  90  93  94  96  97 100
```

```
x[index]
```

```
##  [1] -1.22874970 -0.14059798 -0.20732697 -0.92153058 -0.03285159 -1.62843554
##  [7] -0.56266928 -0.85173525 -0.91512892 -0.44484557 -1.44896422 -0.59389701
## [13] -0.75144575 -0.57543743 -0.08766336 -0.45803324 -0.57952274 -0.82407695
## [19] -2.07172304 -0.49889987 -1.51680685 -0.62976566 -0.36027572 -0.39350268
## [25] -1.04180068 -0.63945719 -0.85649561 -0.38280458 -1.89724701 -0.69186589
## [31] -0.43251556 -0.95196059 -0.21625030 -1.04279172 -0.10986728 -1.84878973
## [37] -0.98415680 -0.21854832 -0.80092215 -0.50991847 -0.09837813 -0.83113845
## [43] -0.19782135 -0.54494659 -0.32692897 -0.25978735 -0.11579965 -0.93762421
## [49] -0.27231887 -0.47014392 -0.06493665 -0.96108831
```