‘R’ Notes for Semester-1 (Part-a)

23BMAT019

10/13/2023

- 1 Installing R and RStudio
- 2 Working with RStudio
  - 2.1 Panes in RStudio
  - 2.2 Creating an RMarkdown File
    - 2.2.1 Viewing RMD Output
    - 2.2.2 Additional RMD Settings
  - 2.3 Writing R Code in RMD
  - 2.4 Ways of Executing R Code (R ≠ RMD)
- 3 Help Commands
- 4 RMD
  - 4.1 Headings
  - 4.2 Text Formatting
  - 4.3 Lists
    - 4.3.1 Ordered Lists
    - 4.3.2 Unordered Lists
- 5 About R
  - 5.1 Advantages of R
  - 5.2 Disadvantages of R
- 6 Arithmetic Operators in R
  - 6.1 R as a Calculator
    - 6.1.1 (Miscellaneous) Functional Translation of Arithmetic Operators
- 7 Datatypes in R
  - 7.1 Character
  - 7.2 Numeric/Integer
  - 7.3 Logical (Boolean)
  - 7.4 Complex
- 8 Maths in R
  - 8.1 Constants and Special Features
  - 8.2 Logarithms
  - 8.3 Trigonometric Functions
- 9 Working with Packages
- 10 Data Structures
  - 10.1 Vectors
    - 10.1.1 `c()` - Concatenation/Combine Function
    - 10.1.2 Creating a vector using `:`
    - 10.1.3 `seq()` - Sequence Function
    - 10.1.4 `scan()` - Scan Function to create a vector from File/User Input
    - 10.1.5 `rep()` - Used to generate a vector with repetitive pattern
    - 10.1.6 `paste()` - Element-wise concatenation of elements
    - 10.1.7 Modifying Vectors
    - 10.1.8 Cumulative Arithmetic (Left to Right)
  - 10.2 Matrices
    - 10.2.1 Arithmetic Operations on Matrices
    - 10.2.2 Matrix Features
    - 10.2.3 Matrix-specific Functions

        - 10.2.4 More ways to create a Matrix
        - 10.2.5 Class Exercises
    - 10.3 Data Frames
        - 10.3.1 Creating a Data Frame
        - 10.3.2 Built-in Data
        - 10.3.3 Working with Data Frames
        - 10.3.4 Other Important Data Frame Functions
        - 10.3.5 Extracting Elements
        - 10.3.6 Extracting Rows and Columns and Filtered Extraction
        - 10.3.7 Extracting Data Frames
        - 10.3.8 Modifying Elements in a Data Frame
        - 10.3.9 Class Exercises
        - 10.3.10 Homework
    - 10.4 Lists
        - 10.4.1 Creating a List
        - 10.4.2 Number of Elements in a List
        - 10.4.3 Naming List Elements
        - 10.4.4 Accessing Elements and Sub-elements of a List
        - 10.4.5 Modifying elements
        - 10.4.6 Class Exercises
- 11 Plotting
    - 11.1 `attach()` and `detach()`
    - 11.2 Scatter Plot: `plot()`
    - 11.3 Discrete / Isolated Points
    - 11.4 Pre-defined functions (trigonometric, exp, log)
    - 11.5 Explicit Functions `y = f(x)`
    - 11.6 Class Exercises
    - 11.7 HW
- 12 Importing and Exporting Files
    - 12.1 Directory commands
        - 12.1.1 `getwd()`
        - 12.1.2 `dir()`
        - 12.1.3 `setwd(path)`
        - 12.1.4 `file.choose()`
- 13 Analysis - I
    - 13.1 `read.csv()`
    - 13.2 `read.csv2()`
    - 13.3 `read.delim()`
    - 13.4 `read.delim2()`
    - 13.5 `read.table()`
- 14 Analysis - II
    - 14.1 Pairs Plotting (Multiple Correlation Plot)
        - 14.1.1 Additional Arguments
    - 14.2 Pie Chart
        - 14.2.1 Additional Arguments
    - 14.3 Data Structure Conversion
    - 14.4 Box-Whisker Plot
        - 14.4.1 Additional Arguments for `boxplot()`
    - 14.5 Class Exercises
- 15 Analysis - III
    - 15.1 Histograms (in Detail)
    - 15.2 Line Charts (in Detail)

1 Installing R and RStudio

- Download Page for R (https://cran.rstudio.com/)
- Download Page for RStudio (https://posit.co/download/rstudio-desktop/) or,
- Execute R Online
    - JDoodle (https://www.jdoodle.com/execute-r-online/)
    - OneCompiler (https://onecompiler.com/r)
    - tutorialspoint (https://www.tutorialspoint.com/execute_r_online.php)
    - replit (https://repl.it) (Sign-up required)

2 Working with RStudio

2.1 Panes in RStudio

There are 4 panes in RStudio:

1. **Source** (top left; edit files here)
2. **Environment/History** (top right)
3. **Console** (bottom left)
4. **Packages/Help** (bottom right)

2.2 Creating an RMarkdown File

1. **File** > **New File** > **R Markdown...**
2. (Document Section) Set **Title** and **Author**, Default Output Format **HTML**

2.2.1 Viewing RMD Output

Select **Knit** > **Knit to HTML/PDF/Word** (OR) **Ctrl**+**Shift**+**K**

2.2.2 Additional RMD Settings

Click the **Settings** (Cog) icon near *Knit* > **Output Options...**

Selecting **Include table of contents** will include the index in the output

Selecting **Number section headings** will label sections in the document e.g. **1.1.1 About R**, **1.1.2 Advantages of R**, etc. according to the heading(s)

2.3 Writing R Code in RMD

To write and run R code in an RMD file, it must be written within a **Chunk**.

To insert a Chunk at a line:

1. Press **Insert** > **R** (OR)
2. **Ctrl**+**Alt**+**I**

2.4 Ways of Executing R Code (R ≠ RMD)

1. Press **Ctrl**+**Enter** to execute the **current line** (line on which the cursor is active)
2. In the chunk, press the **Play** button to execute/run *every* line **in** the chunk
3. **Select** the lines to run and press **Ctrl**+**Enter**

4. **Knit** the RMD file as usual (also outputs the result)

In this section, using of quotes maybe optional in some cases, but it is recommended.

3 Help Commands

1. `help()` or `?`

Displays the help/manual for a **specific** R Package. The following commands are equivalent to each other:

```
# Quotes are optional
help(mean)
help("mean")
help('mean')

# Quotes are optional
? mean
? "mean"
? 'mean'
```

2. `help.search()` or `??`

Displays the manuals which contain the keyword (or the argument passed)

```
# Quotes are mandatory
help.search("mean")
help.search('mean')

# Quotes are optional
?? mean
?? "mean"
?? 'mean'
```

3. `help.start()`

Launches the General R Support Manual/Page

```
help.start()      # Launches Manual in he 4th Pane
```

4. `apropos()`

Finds all commands/functions associated with the argument passed

```
# Quotes are mandatory
apropos('mean')
```

```
##  [1] ".colMeans"     ".rowMeans"     "colMeans"      "kmeans"
##  [5] "mean"          "mean.Date"     "mean.default"  "mean.difftime"
##  [9] "mean.POSIXct"  "mean.POSIXlt"  "rowMeans"      "weighted.mean"
```

4 RMD

4.1 Headings

There are 6 types of headings in RMD (`#` , `##` , … , `######`).

Leave a space after the `#..` and enter the heading.

```
# Heading 1 (Largest)
## Heading 2
### Heading 3
#### Heading 4
##### Heading 5
###### Heading 6 (Smallest)
```

4.2 Text Formatting

Enclose text within `*` and `*` for *italicised* text.

Enclose text within `**` and `**` for **bold** text.

Enclose text within `***` and `***` for ***bold and italicised*** text.

Enclose text between two backticks (` on either side) for an `Inline block`

Enclose text between six backticks (``` on either side) for a

```
print("This program is within a code block.")
```

```
## [1] "This program is within a code block."
```

4.3 Lists

4.3.1 Ordered Lists

```
**Most Visited Areas**

1. Cafeteria
2. Science Dhaba
3. Library
4. JCR
5. Seminar Room
```

Most Visited Areas

1. Cafeteria
2. Science Dhaba
3. Library
4. JCR
5. Seminar Room

```
**Residence Blocks**

a. Allnutt North
b. Allnutt South
c. Rudra North
d. Rudra South
e. Mukarji East
f. Mukarji West
```

Residence Blocks

a. Allnutt North
b. Allnutt South
c. Rudra North
d. Rudra South
e. Mukarji East
f. Mukarji West

4.3.2 Unordered Lists

```
**Pancake Ingredients**

- Flour
- Baking Powder
- Sugar
- Salt
- Milk and Butter
- Egg
```

Pancake Ingredients

- Flour
- Baking Powder
- Sugar
- Salt
- Milk and Butter
- Egg

5 About R

R is a programming language. It was developed by **R**obert Gentleman and **R**oss Ihaka. It is a **numeric** software (fractions are expressed as decimals; which is not the case in *symbolic* software).

5.1 Advantages of R

- Open Source
- Portable Programs
- Data Analysis & Visualisation

5.2 Disadvantages of R

- Can be memory intensive

- Security issues
- Comparatively slower

6 Arithmetic Operators in R

- `+` for addition
- `-` for subtraction
- `*` for multiplication
- `/` for division
- `^` for exponentiation
- `%%` for modulo

6.1 R as a Calculator

```
15+45
```

```
## [1] 60
```

```
2^10
```

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

```
(45-7)/5
```

```
## [1] 7.6
```

6.1.1 (Miscellaneous) Functional Translation of Arithmetic Operators

We know that +, -, etc. are binary operations on the set of non-zero Complex Numbers (non-zero for generality).

And, operations on a set A is a function from $A \times A$ to another `non-empty set` (A itself if the operation is *binary*)

So, $a + b \equiv +(a, b)\ \forall a, b$

```
"+"(2, 5)
```

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

```
"*"(2, 9)
```

```
## [1] 18
```

```
"/"(8, 4)
```

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

7 Datatypes in R

There are 4 datatypes in R, namely:

1. Character
2. Numeric/Integer
3. Logical (Boolean)
4. Complex

7.1 Character

Anything within single/double quotes in R is classified as a `character`

```
# This is a comment

"Hello R!"  # This is a string
```

```
## [1] "Hello R!"
```

```
'Hello R!'  # This is also a string
```

```
## [1] "Hello R!"
```

```
# Use class() function to check the class/type of the argument
class("Hello R!")
```

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

7.2 Numeric/Integer

The class of any number is `numeric` unless it is explicitly stated to be an integer.

```
class(20)   # numeric
```

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

```
class(20L)  # integer
```

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

```
class(20.50)
```

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

7.3 Logical (Boolean)

There are two Logical values namely True and False. In R, they are denoted by either `TRUE` and `FALSE` or `T` and `F` respectively.

The boolean value `TRUE` has a numeric value of 1.

The boolean value `FALSE` has a numeric value of 0.

```
class(TRUE)     # logical
```

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

```
class(T)
```

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

```
5*T+20          # 25
```

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

```
class(5*T+20)    # numeric
```

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

7.4 Complex

Standard form of a complex number is $z < -a + bi$ where a and b are members of the set of Real numbers.

a is the *real part* whereas bi is the *imaginary part* of the complex number z.

In R, while working with `complex` datatype, the b in the complex number z has to be stated before i (i.e. `1i` and `i` are not the same, `i` could also refer to a variable)

```
# i*-i  # Gives an error
1i*-1i
```

```
## [1] 1+0i
```

```
(12+45i) * (12+2i)
```

```
## [1] 54+564i
```

```
class(42+16i)   # complex
```

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

8 Maths in R

8.1 Constants and Special Features

```
### pi
pi          # 3.1415..
```

```
## [1] 3.141593
```

```
### e
exp(1)      # 2.71.. (e^x at x=1)
```

```
## [1] 2.718282
```

```
### absolute value
abs(-45)    # 45
```

```
## [1] 45
```

```
### exponential notation
1e3         # 1 * 10^3
```

```
## [1] 1000
```

8.2 Logarithms

```
### natural
e <- exp(1)
log(e)       # 1
```

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

```
### custom base
log(343, 7)  # 3
```

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

```
### binary
log2(8)      # 3
```

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

```
### common
log10(1e2)   # 2
```

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

8.3 Trigonometric Functions

```
pi/4
```

```
## [1] 0.7853982
```

```
sin(pi/4)
```

```
## [1] 0.7071068
```

```
y <- sin(pi/4)
asin(y)     # arc sin or sin inverse of y
```

```
## [1] 0.7853982
```

9 Working with Packages

1. `search()`

Displays all installed packages

```
search()      # No arguments required
```

2. `searchpaths()`

Displays the paths of all installed packages

```
searchpaths()
```

3. `install.packages()`

Install packages with the name of the package(s) passed.

```
install.packages("ggplot2")
```

4. `library()`

This will add package(s) to the library which will enable us to use the commands/functions in the installed package(s)

```
library(ggplot2)
```

10 Data Structures

10.1 Vectors

Vectors are **homogeneous** and **1-Dimensional** data structures. Elements will be coerced according to the type precedence.

To create a vector, we can use `c()`, `:`, `seq()`, `scan()`, `paste()`, `rep()`, etc.

10.1.1 `c()` - Concatenation/Combine Function

Syntax: `c(elt1, elt2, elt3, ...)`

Different types of Assignment Operators

```
# Assignment Operator =
v1 <- c(1, 2, 3, 4)

# Left Assignment Operator <- or <<-
v2 <- c(0.3, 0.5, 0.11)

# Right Assignment Operator -> or ->>
c(12e3, 15e3, 25e3) -> v3
```

10.1.2 Creating a vector using `:`

```
# The colon operator comes in handy when the absolute difference between the elements is 1
v4 <- 1:8; v4   # Separating lines with ; allows to run them at once
```

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

```
class(v4)       # integer
```

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

Types of Vectors and Type Coercion

```
# Numeric/Integer Vector
v5 <- c(2, 3, 1, 6.5); v5   # all integers are coerced to numeric
```

```
## [1] 2.0 3.0 1.0 6.5
```

```
class(v5)     # numeric
```

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

```
# Character Vector
v6 <- c("First", "Second", TRUE, 1); v6  # all non-char coerced to char
```

```
## [1] "First"  "Second" "TRUE"   "1"
```

```
class(v6)
```

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

Type Coercion Precedence

1. Character
2. Numeric
3. Integer

10.1.3 `seq()` - Sequence Function

Syntax: `seq(from=a, to=b, by=c)`

Positional Arguments: a, b, c (default value of c is 1)

Keyword Argument: `length` (this function estimates the common difference according to the argument i.e. number of elements in the series)

```
s1 <- seq(0, 25, 5); s1
```

```
## [1]  0  5 10 15 20 25
```

```
s2 <- seq(0, 10, .5); s2
```

```
##  [1]  0.0  0.5  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0
## [16]  7.5  8.0  8.5  9.0  9.5 10.0
```

```
s3 <- seq(1, 15); s3
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

```
s4 <- seq(1, 15, length=20); s4
```

```
##  [1]  1.000000  1.736842  2.473684  3.210526  3.947368  4.684211  5.421053
##  [8]  6.157895  6.894737  7.631579  8.368421  9.105263  9.842105 10.578947
## [15] 11.315789 12.052632 12.789474 13.526316 14.263158 15.000000
```

10.1.4 `scan()` - Scan Function to create a vector from File/User Input

```
# Creating a Numeric Vector with user input
c1 <- scan() # No params => fetch (numeric) elts. from buffer input
c1; class(c1)
```

```
## numeric(0)
```

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

```
# Creating a Character Vector with user input
c2 <- scan(what=character())
c2; class(c2)
```

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

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

```
# Numeric Vector with a delimiter other than space
c3 <- scan(sep=",")    # 2 3 => 23, 2 .5 => 2.5, ...
c3; class(c3)
```

```
## numeric(0)
```

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

```
# sep="-"
c4 <- scan(sep="-")
c4; class(c4)
```

```
## numeric(0)
```

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

```
# sep="&"
c5 <- scan(what=character(), sep="&")
c5; class(c5)
```

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

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

```
# sep="|"
c6 <- scan(what=complex(), sep="|")
c6; class(c6)
```

```
## complex(0)
```

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

10.1.5 `rep()` - Used to generate a vector with repetitive pattern

Syntax: `rep(arg1, arg2)`

where arg1: A vector, arg2: A number or a vector

```
rep(1:4, 2)
```

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

```
rep(1:4, 4:1)   # Repeat the arg1[n] element arg2[n] times
```

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

```
rep(c(1,2,3), c(2,4,5))
```

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

```
rep(1:3, each=3)# Repeats each elt of arg1 3 times
```

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

10.1.6 `paste()` - Element-wise concatenation of elements

Syntax: `paste(vector_1, vector_2)`

```
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)

v3 <- paste(v1, v2)
```

10.1.6.1 Other Special Features in R

`letters` returns all lowercase letters `LETTERS` returns all UPPERCASE letters

```
letters  # lowercase alphabet
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS  # UPPERCASE alphabet
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z"
```

```
vx <- paste(1:26, LETTERS, 1:26, sep=""); vx
```

```
##  [1] "1A1"   "2B2"   "3C3"   "4D4"   "5E5"   "6F6"   "7G7"   "8H8"   "9I9"
## [10] "10J10" "11K11" "12L12" "13M13" "14N14" "15O15" "16P16" "17Q17" "18R18"
## [19] "19S19" "20T20" "21U21" "22V22" "23W23" "24X24" "25Y25" "26Z26"
```

10.1.7 Modifying Vectors

To modify a vector, use `[ ]`

- adding
- deleting
- replacing

```
# index starts at 1
# length(vector) returns the order of the vector
r=c(2,5,7,9,5,2,6); r; length(r)
```

```
## [1] 2 5 7 9 5 2 6
```

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

```
# adding element(s)
r[8:10] <- c(23,45,33)
r;length(r)
```

```
##  [1]  2  5  7  9  5  2  6 23 45 33
```

```
## [1] 10
```

```
# accessing elements
r[5]  # single element
```

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

```
r[c(1,2)]  # multiple elements
```

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

```
# replacing elements
r[5] <- 777; r
```

```
##  [1]   2   5   7   9 777   2   6  23  45  33
```

```
# replacing elements
## Method 1
r; temp <- r[1]; r[1] <- r[length(r)]; r[length(r)] <- temp; r
```

```
##  [1]   2   5   7   9 777   2   6  23  45  33
```

```
##  [1]  33   5   7   9 777   2   6  23  45   2
```

```
## Method 2
r[c(1,10)] <- r[c(10,1)]; r
```

```
##  [1]   2   5   7   9 777   2   6  23  45  33
```

```
# extracting all even places
even <- r[seq(2,length(r),2)]; even
```

```
## [1]  5  9  2 23 33
```

```
# sum of all odd places
sum(r[seq(1, length(r), 2)])
```

```
## [1] 837
```

```
# deleting elements
r=r[-1]; r      # r=r[-n]   # rm single elt with index n
```

```
## [1]   5   7   9 777   2   6  23  45  33
```

```
r <- r[-c(3,7)]  # removing multiple elts
# r <- r[c(-3,-7)]
r
```

```
## [1]   5   7 777   2   6  45  33
```

10.1.8 Cumulative Arithmetic (Left to Right)

```
my_vector = rep(1:5, 2:6); my_vector # for demonstration
```

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

```
cumsum(my_vector)  # Cumulative Sum
```

```
##  [1]  1  2  4  6  8 11 14 17 20 24 28 32 36 40 45 50 55 60 65 70
```

```
cumprod(my_vector) # Cumulative Product
```

```
##  [1]           1           1           2           4           8          24
##  [7]          72         216         648        2592       10368       41472
## [13]      165888      663552     3317760    16588800    82944000   414720000
## [19]  2073600000 10368000000
```

```
cummin(my_vector)  # Cumulative Minimum
```

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

```
cummax(my_vector)  # Cumulative Maximum
```

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

10.2 Matrices

Syntax: `matrix(x, nrow=1, ncol=1, byrow=F)`

Creates a matrix from a vector `x` (entries are ordered column-wise by default, set `byrow` to `T` or `TRUE` for row-wise ordering)

If any one of `nrow` or `ncol` is specified, the other will be computed by R.

```
x <- seq(1, 10, length=12)
matrix(x, 3, 4)
```

```
##          [,1]     [,2]     [,3]      [,4]
## [1,] 1.000000 3.454545 5.909091  8.363636
## [2,] 1.818182 4.272727 6.727273  9.181818
## [3,] 2.636364 5.090909 7.545455 10.000000
```

```
M <- matrix(x, 3, 4, byrow=T); M
```

```
##          [,1]     [,2]     [,3]      [,4]
## [1,] 1.000000 1.818182 2.636364  3.454545
## [2,] 4.272727 5.090909 5.909091  6.727273
## [3,] 7.545455 8.363636 9.181818 10.000000
```

```
# Accessing matrix entries
M[2,2]
```

```
## [1] 5.090909
```

```
M[1:2, 2]
```

```
## [1] 1.818182 5.090909
```

10.2.1 Arithmetic Operations on Matrices

```
# Matrices for demonstration
A <- matrix(seq(1, 12, length=12), 3, 4); A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
B <- matrix(seq(2, 24, length=12), 3, 4); B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    8   14   20
## [2,]    4   10   16   22
## [3,]    6   12   18   24
```

```
C <- matrix(seq(1, 10, length=12), 4, 3); C
```

```
##          [,1]     [,2]      [,3]
## [1,] 1.000000 4.272727  7.545455
## [2,] 1.818182 5.090909  8.363636
## [3,] 2.636364 5.909091  9.181818
## [4,] 3.454545 6.727273 10.000000
```

10.2.1.1 Addition `+`

```
# Element-wise
A + B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3   12   21   30
## [2,]    6   15   24   33
## [3,]    9   18   27   36
```

10.2.1.2 Subtraction `-`

```
# Element-wise
A - B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -1   -4   -7  -10
## [2,]   -2   -5   -8  -11
## [3,]   -3   -6   -9  -12
```

10.2.1.3 Multiplication

- Element-wise `*`

```
A * B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2   32   98  200
## [2,]    8   50  128  242
## [3,]   18   72  162  288
```

- Matrix Multiplication `%*%`

```
A %*% C
```

```
##          [,1]     [,2]     [,3]
## [1,] 61.27273 133.2727 205.2727
## [2,] 70.18182 155.2727 240.3636
## [3,] 79.09091 177.2727 275.4545
```

- Scalar Multiplication/Division `<constt.> * <Matrix>`

```
3 * A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3   12   21   30
## [2,]    6   15   24   33
## [3,]    9   18   27   36
```

```
1/2 * A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  0.5  2.0  3.5  5.0
## [2,]  1.0  2.5  4.0  5.5
## [3,]  1.5  3.0  4.5  6.0
```

10.2.1.4 Exponentiation `^`

```
# Element-wise
A ^ B
```

```
##      [,1]       [,2]         [,3]         [,4]
## [1,]    1      65536 6.782231e+11 1.000000e+20
## [2,]   16    9765625 2.814750e+14 8.140275e+22
## [3,]  729 2176782336 1.500946e+17 7.949685e+25
```

10.2.1.5 Division `/`

```
# Element-wise
B / A
```

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

10.2.2 Matrix Features

```
shopping_list <- matrix(round(runif(16, 1, 10)), 4); A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
M <- matrix(1:16*2, 4); M
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2   10   18   26
## [2,]    4   12   20   28
## [3,]    6   14   22   30
## [4,]    8   16   24   32
```

10.2.2.1 Number of Elements, Rows and Columns

```
length(M) # Number of Elements in M
```

```
## [1] 16
```

```
nrow(M)   # Number of rows in M
```

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

```
ncol(M)   # Number of columns in M
```

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

10.2.2.2 Row and Column Names

Syntax: `rownames(x) <- c(<rn1>, <rn2>, ..., <rnp>)` Assigns names to the rows of a Matrix `x` of order `p x q`

Syntax: `colnames(x) <- c(<cn1>, <cn2>, ..., <cnq>)` Assigns names to the columns of a Matrix `x` of order `p x q`

```
shopping_list   # Before Setting Dimension Names
```

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

```
# Setting Dimension Names
rownames(shopping_list) <- c("Clothes", "Books", "Pens", "Fruits")
colnames(shopping_list) <- c("Person_A", "Person_B", "Person_C", "Person_D")

shopping_list   # After Setting Dimension Names
```

```
##         Person_A Person_B Person_C Person_D
## Clothes        3        7        2        1
## Books          7        7        9        5
## Pens           4        6        7        7
## Fruits         2       10        5        1
```

```
rownames(B) <- paste("B_row-", 1:nrow(B), sep="")
colnames(B) <- paste("B_col-", 1:ncol(B), sep="")
```

10.2.2.3 Sums in Matrices

```
sum(M)        # Sum of all entries in M
```

```
## [1] 272
```

```
rowSums(M)    # Vector of sum of entries in each row
```

```
## [1] 56 64 72 80
```

```
rowSums(M)[2] # Sum of the 2nd Row entries
```

```
## [1] 64
```

```
colSums(M)    # Vector of sum of entries in each column
```

```
## [1]  20  52  84 116
```

```
colSums(M)[3] # Sum of the 2nd Column entries
```

```
## [1] 84
```

10.2.2.4 Averages/Means in Matrices

```
mean(M)        # Mean of all entries in M
```

```
## [1] 17
```

```
rowMeans(M)    # Vector of mean of entries in each row
```

```
## [1] 14 16 18 20
```

```
rowMeans(M)[1] # Mean of 1st Row entries
```

```
## [1] 14
```

```
colMeans(M)    # Vector of mean of entries in each column
```

```
## [1]  5 13 21 29
```

```
colMeans(M)[2] # Mean of 2nd Column entries
```

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

Similarly, `var()`, `sd()`, etc. can also be mapped on a Matrix

10.2.2.5 Matrices as Vector

Matrices are treated as Vectors in R. So, vector functions can be used on Matrices. E.g. `length()`, `sum()`, `mean()`, etc. and also the following:

```
# The cumulative sum/prod/max/min will be calculated according to the definition of the matri
        x ('byrow' parameter)
cumsum(M)
```

```
##  [1]   2   6  12  20  30  42  56  72  90 110 132 156 182 210 240 272
```

```
cumprod(M)
```

```
##  [1] 2.000000e+00 8.000000e+00 4.800000e+01 3.840000e+02 3.840000e+03
##  [6] 4.608000e+04 6.451200e+05 1.032192e+07 1.857946e+08 3.715891e+09
## [11] 8.174961e+10 1.961991e+12 5.101175e+13 1.428329e+15 4.284987e+16
## [16] 1.371196e+18
```

```
cummax(M)
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32
```

```
cummin(M)
```

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

10.2.3 Matrix-specific Functions

10.2.3.1 `det()` : Determinant of a Matrix

Syntax: `det(M)`

```
det(shopping_list)  # det shopping_list
```

```
## [1] 666
```

```
det(M)
```

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

10.2.3.2 `t()` : Transpose of a Matrix

Syntax: `t(M)`

```
t(M)    # M'
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    4    6    8
## [2,]   10   12   14   16
## [3,]   18   20   22   24
## [4,]   26   28   30   32
```

10.2.3.3 `dim()` : Dimensions (Order) of a Matrix

Syntax: `dim(M)`

```
dim(A)  # c(row, col)
```

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

10.2.3.4 `diag()` : Diagonal

10.2.3.4.1 `diag()` on a Matrix

```
# vector diagonal entries of M
diag(M)
```

```
## [1]  2 12 22 32
```

10.2.3.4.2 `diag()` on a Vector

```
# diagonal matrix with given diagonal entries
diag(c(1, 2, 3))
```

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

10.2.3.4.3 Trace of a Matrix (Sum of diagonal entries)

```
sum(diag(M))
```

```
## [1] 68
```

10.2.3.5 Inverse of a Matrix and Solution of a System of Equations

10.2.3.5.1 Inverse

Syntax: `solve(M)`

```
A <- matrix(round(runif(16, 5, 15)), 4); A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    6    7   12    8
## [2,]    8    6    8   10
## [3,]   11   10   13    9
## [4,]    9   14    9    7
```

```
solve(A)  # Inverse of A
```

```
##             [,1]         [,2]        [,3]        [,4]
## [1,] -0.27034121 -0.003937008  0.31102362 -0.08530184
## [2,]  0.04461942 -0.023622047 -0.13385827  0.15485564
## [3,]  0.14173228 -0.133858268  0.07480315 -0.06692913
## [4,]  0.07611549  0.224409449 -0.22834646  0.02887139
```

10.2.3.5.2 Solution of System of Equations

Syntax: `solve(A, B)` where `A` is the matrix of variable coefficient and `B` is the matrix of constants.

```
# System of Equations: 2x+5y=12 and 7x+9y=13
A <- matrix(c(2,5,7,9), 2, byrow=T)
B <- matrix(c(12, 13), 2)
solve(A, B)
```

```
##           [,1]
## [1,] -2.529412
## [2,]  3.411765
```

10.2.3.6 Merging Matrices

10.2.3.6.1 ***R***ow-wise ***bind***ing

Merging/Appending two Matrices by Row

Condition: The number of **columns** of both matrices must be the same **Syntax**: `rbind(A, B)`

10.2.3.6.2 ***C***olumn-wise ***bind***ing

Merging/Appending two Matrices by Column

Condition: The number of **rows** of both matrices must be the same **Syntax**: `cbind(A, B)`

10.2.4 More ways to create a Matrix

1. Using `c()` and `dim()` (cannot be ordered)

```
x <- 1:9; x     # x as a vector
```

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

```
dim(x) <- c(3,3)
x               # x as a matrix
```

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

2. Using `array()` (cannot be ordered)

```
x <- array(1:20, dim=c(5, 4)); x
```

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

```
class(x)
```

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

10.2.5 Class Exercises

1. Create a 3x4 Matrix and provide dimension names

```
G <- matrix(round(runif(12, 10, 99)), 3)

rownames(G) <- c("A", "B", "C"); rownames(G)
```

```
## [1] "A" "B" "C"
```

```
colnames(G) <- c("Fans", "Books", "Clothes", "Biscuits"); colnames(G)
```

```
## [1] "Fans"     "Books"    "Clothes"  "Biscuits"
```

```
## In case of a visible pattern
rownames(G) <- paste("row", 1:nrow(G), sep=""); rownames(G)
```

```
## [1] "row1" "row2" "row3"
```

```
colnames(G) <- paste("col", 1:ncol(G), sep=""); colnames(G)
```

```
## [1] "col1" "col2" "col3" "col4"
```

2. Merge 2 Matrices row-wise and column-wise

```
M <- matrix(round(runif(12, 10, 99)), 3, 4)
N <- matrix(round(runif(20, 10, 99)), 5, 4)
rbind(M, N)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   47   21   83   53
## [2,]   79   78   75   65
## [3,]   85   50   54   68
## [4,]   54   35   68   28
## [5,]   43   73   55   70
## [6,]   45   51   74   26
## [7,]   30   47   19   47
## [8,]   32   97   92   94
```

```
dim(rbind(M, N))
```

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

```
M <- matrix(round(runif(28, 10, 99)), 7, 4)
N <- matrix(round(runif(42, 10, 99)), 7, 6)
cbind(M, N)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   44   32   84   34   26   19   89   91   31    19
## [2,]   78   35   76   70   60   95   72   56   19    15
## [3,]   45   73   14   78   43   41   43   21   72    97
## [4,]   36   36   62   11   69   45   63   77   23    56
## [5,]   46   10   42   51   34   80   58   47   27    39
## [6,]   22   43   98   40   32   81   82   44   74    65
## [7,]   75   89   88   12   64   68   15   63   57    24
```

```
dim(cbind(M, N))
```

```
## [1]  7 10
```

3. Find the inverse of a matrix and verify it

```
A <- matrix(c(12,45,27,35), 2)
A
```

```
##      [,1] [,2]
## [1,]   12   27
## [2,]   45   35
```

```
AI <- solve(A)   # Inv A
AI
```

```
##             [,1]        [,2]
## [1,] -0.04402516  0.03396226
## [2,]  0.05660377 -0.01509434
```

```
round(A %*% AI)
```

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

4. Take a system of equations having 4 equations in 4 variables. Approach it in two possible ways using solve() command

Propositional System of Equations

- $4a + 2b + 5c + 9d = 12$
- $-5a - 12b - 7c + 17d = 13$
- $2a - 4b + 13c + 11d = 53$
- $2a - 5b + 5c + 8d = 79$

```
det(A)   # non-zero
```

```
## [1] -795
```

```
### Method 1
A <- matrix(c(4,2,5,9,-5,-12,-7,17,2,-4,13,11,-2,-5,5,8), 4, byrow=T)
B <- matrix(c(12,13,53,79), ncol=1)

sol <- solve(A, B)
sol
```

```
##            [,1]
## [1,] -34.464181
## [2,]  18.887061
## [3,]   8.780336
## [4,]   7.575658
```

```
### Method 2
AI <- solve(A)
AI
```

```
##            [,1]         [,2]         [,3]       [,4]
## [1,] -0.1235380  0.141812865  0.395467836 -0.7061404
## [2,]  0.2313596 -0.120614035 -0.323464912  0.4407895
## [3,] -0.0127924 -0.053362573  0.008406433  0.1162281
## [4,]  0.1217105 -0.006578947 -0.108552632  0.1513158
```

```
sol <- AI %*% B
sol
```

```
##            [,1]
## [1,] -34.464181
## [2,]  18.887061
## [3,]   8.780336
## [4,]   7.575658
```

5. Explore the possibilities of solutions of system of equations in 2 variables

- Unique Solution

```
# x  + 2y = 5
# 3x + 4y = 6
A <- matrix(c(1, 2, 3, 4), 2, byrow=T)
B <- matrix(c(5, 6), ncol=1)
B <- c(5, 6)  # Works
solve(A, B)
```

```
## [1] -4.0  4.5
```

- No Solution

$$\frac{a_1}{a_2} = \frac{b_1}{b_2} \neq \frac{c_1}{c_2} \implies No\ solution$$

That is, no solution exists when $a_1 \cdot b_2 = a_2 \cdot b_1$ but c_is are not proportional to a_is and b_is. In other words, the variable coefficient matrix A is singular.

```
# x  + 3y = 5
# 2x + 6y = 11
A <- matrix(c(1, 3, 2, 6), 2, byrow=T); A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    6
```

```
B <- matrix(c(5, 11), ncol=1); B
```

```
##      [,1]
## [1,]    5
## [2,]   11
```

```
det(A)
```

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

```
#solve(A, B)
#Error in solve.default(A, B) : Lapack routine dgesv: system is exactly singular: U[2,2] = 0
```

- Infinite Solutions

$$\frac{a_1}{a_2} = \frac{b_1}{b_2} = \frac{c_1}{c_2} \implies \infty \, solutions$$

That is, infinite solutions exist when a_is, b_is and c_is are proportional.

Again, $a_1 \cdot b_2 = a_2 \cdot b_1$ i.e. singularity of the variable coefficient matrix A

```
# 3x + 6y = 12
# 9x + 18y = 36
A <- matrix(c(3, 6, 9, 18), 2, byrow=T); A
```

```
##      [,1] [,2]
## [1,]    3    6
## [2,]    9   18
```

```
B <- matrix(c(12, 36), ncol=1); B
```

```
##      [,1]
## [1,]   12
## [2,]   36
```

```
det(A)
```

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

```
#solve(A, B)
```

6. Create a 4x4 Matrix and:

```
A <- matrix(round(runif(16, 10, 90)), 4); A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   53   19   79   87
## [2,]   21   76   19   33
## [3,]   12   38   41   33
## [4,]   12   47   85   69
```

```
m <- nrow(A)
n <- ncol(A)
```

i. extract all 4 corners

```
tlc <- A[1,1]; tlc
```

```
## [1] 53
```

```
trc <- A[1,n]; trc
```

```
## [1] 87
```

```
brc <- A[m,n]; brc
```

```
## [1] 69
```

```
blc <- A[m,1]; blc
```

```
## [1] 12
```

```
matrix(c(tlc, trc, blc, brc), 2, byrow=T)
```

```
##      [,1] [,2]
## [1,]   53   87
## [2,]   12   69
```

ii. extract the 2x2 matrix in-between

```
A[2:3, 2:3]
```

```
##      [,1] [,2]
## [1,]   76   19
## [2,]   38   41
```

iii. interchange the (2,4)th entry by (4,2)th entry

```
## Method 1
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   53   19   79   87
## [2,]   21   76   19   33
## [3,]   12   38   41   33
## [4,]   12   47   85   69
```

```
temp <- A[2,4]
A[2,4] <- A[4,2]
A[4,2] <- temp
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   53   19   79   87
## [2,]   21   76   19   47
## [3,]   12   38   41   33
## [4,]   12   33   85   69
```

```
## Method 2
A[c(2,4), c(4,2)] = A[c(4,2), c(2,4)]
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   53   19   79   87
## [2,]   21   69   19   33
## [3,]   12   38   41   33
## [4,]   12   47   85   76
```

iv. delete the last row

```
## Method 1
A <- A[1:m-1, 1:n]; A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   53   19   79   87
## [2,]   21   69   19   33
## [3,]   12   38   41   33
```

```
## Method 2
A <- A[-nrow(A),]; A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   53   19   79   87
## [2,]   21   69   19   33
```

v. add one more column

```
A <- cbind(A, matrix(c(1:2), 2)); A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   53   19   79   87    1
## [2,]   21   69   19   33    2
```

10.3 Data Frames

Unlike other data structures (vectors and matrices), a data frame allows us to store **more than one** data type. A **Data Frame** is a **2-dimensional** data structure which can store different data types (**each column** being of the **same data type** and each column having the **same number of elements**).

10.3.1 Creating a Data Frame

One of the most convenient ways of creating a data frame is having columns (vectors) of different data types. One can use `rownames()` to assign names to the rows of a data frame and either `names()` or `colnames()` to assign names to the columns.

Syntax: `data.frame(data1, data2, ...)`

10.3.2 Built-in Data

Use `data()` for a list of built-in data

```
#trees   # Data type is also mentioned in the table
#class(trees)
#help(trees)

#mtcars

#iris
```

10.3.3 Working with Data Frames

10.3.3.1 Creating a Data Frame

```
d1 <- data.frame(S.no=1:5, Name=letters[1:5], Marks=c(12.4, 12, 10, 10.54, 11.95), Status=rep
        (T, 5)); d1
```

```
##   S.no Name Marks Status
## 1    1    a 12.40   TRUE
## 2    2    b 12.00   TRUE
## 3    3    c 10.00   TRUE
## 4    4    d 10.54   TRUE
## 5    5    e 11.95   TRUE
```

```
c1 <- letters[6:10]
c2 <- round(rnorm(5))
c3 <- c(rep(T, 3), F, F)
d2 <- data.frame(c1, c2, c3); d2
```

```
##   c1 c2    c3
## 1  f  3  TRUE
## 2  g  1  TRUE
## 3  h  1  TRUE
## 4  i  0 FALSE
## 5  j  1 FALSE
```

10.3.3.2 Length of a Data Frame (Number of Columns)

```
length(d1)
```

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

10.3.3.3 Fetching the Column Names

```
names(d1)
```

```
## [1] "S.no"   "Name"   "Marks"  "Status"
```

```
colnames(d1)
```

```
## [1] "S.no"   "Name"   "Marks"  "Status"
```

10.3.3.4 Changing/Accessing Column Names

```
colnames(d2) <- c("Initials", "Deviation", "Status")
colnames(d2); d2
```

```
## [1] "Initials"  "Deviation" "Status"
```

```
##   Initials Deviation Status
## 1        f         3   TRUE
## 2        g         1   TRUE
## 3        h         1   TRUE
## 4        i         0  FALSE
## 5        j         1  FALSE
```

10.3.3.5 Naming the Rows

```
rownames(d2) <- c(1, 2, 3, 4, 5); d2
```

```
##   Initials Deviation Status
## 1        f         3   TRUE
## 2        g         1   TRUE
## 3        h         1   TRUE
## 4        i         0  FALSE
## 5        j         1  FALSE
```

```
rownames(d2)
```

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

10.3.4 Other Important Data Frame Functions

```
#iris
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
# Quartiles from 0 (Min) to 4 (Max) alongwith the Mean, a short Statistical Summary (6 Number
        Summary)
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
##
##
##
```

```
# str for 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 ...
```

10.3.5 Extracting Elements

Syntax: `data.frame[row, column]`

```
iris[1, "Sepal.Length"]
```

```
## [1] 5.1
```

10.3.6 Extracting Rows and Columns and Filtered Extraction

```
# Extract second column
iris[,2]   # Names will not affect the indices
```

```
##   [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9 3.5
##  [19] 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
##  [37] 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
##  [55] 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7 2.2 2.5 3.2 2.8
##  [73] 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0 3.4 3.1 2.3 3.0 2.5
##  [91] 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7 3.0 2.9 3.0 3.0 2.5 2.9
## [109] 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
## [127] 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2
## [145] 3.3 3.0 2.5 3.0 3.4 3.0
```

```
iris[2]    # Data Frames are collections of columns; data frame counterpart to accessing elem
         ents in data frame
```

```
##     Sepal.Width
## 1           3.5
## 2           3.0
## 3           3.2
## 4           3.1
## 5           3.6
## 6           3.9
## 7           3.4
## 8           3.4
## 9           2.9
## 10          3.1
## 11          3.7
## 12          3.4
## 13          3.0
## 14          3.0
## 15          4.0
## 16          4.4
## 17          3.9
## 18          3.5
## 19          3.8
## 20          3.8
## 21          3.4
## 22          3.7
## 23          3.6
## 24          3.3
## 25          3.4
## 26          3.0
## 27          3.4
## 28          3.5
## 29          3.4
## 30          3.2
## 31          3.1
## 32          3.4
## 33          4.1
## 34          4.2
## 35          3.1
## 36          3.2
## 37          3.5
## 38          3.6
## 39          3.0
## 40          3.4
## 41          3.5
## 42          2.3
## 43          3.2
## 44          3.5
## 45          3.8
## 46          3.0
## 47          3.8
## 48          3.2
## 49          3.7
## 50          3.3
## 51          3.2
## 52          3.2
## 53          3.1
## 54          2.3
```

```
## 55          2.8
## 56          2.8
## 57          3.3
## 58          2.4
## 59          2.9
## 60          2.7
## 61          2.0
## 62          3.0
## 63          2.2
## 64          2.9
## 65          2.9
## 66          3.1
## 67          3.0
## 68          2.7
## 69          2.2
## 70          2.5
## 71          3.2
## 72          2.8
## 73          2.5
## 74          2.8
## 75          2.9
## 76          3.0
## 77          2.8
## 78          3.0
## 79          2.9
## 80          2.6
## 81          2.4
## 82          2.4
## 83          2.7
## 84          2.7
## 85          3.0
## 86          3.4
## 87          3.1
## 88          2.3
## 89          3.0
## 90          2.5
## 91          2.6
## 92          3.0
## 93          2.6
## 94          2.3
## 95          2.7
## 96          3.0
## 97          2.9
## 98          2.9
## 99          2.5
## 100         2.8
## 101         3.3
## 102         2.7
## 103         3.0
## 104         2.9
## 105         3.0
## 106         3.0
## 107         2.5
## 108         2.9
## 109         2.5
## 110         3.6
```

```
## 111         3.2
## 112         2.7
## 113         3.0
## 114         2.5
## 115         2.8
## 116         3.2
## 117         3.0
## 118         3.8
## 119         2.6
## 120         2.2
## 121         3.2
## 122         2.8
## 123         2.8
## 124         2.7
## 125         3.3
## 126         3.2
## 127         2.8
## 128         3.0
## 129         2.8
## 130         3.0
## 131         2.8
## 132         3.8
## 133         2.8
## 134         2.8
## 135         2.6
## 136         3.0
## 137         3.4
## 138         3.1
## 139         3.0
## 140         3.1
## 141         3.1
## 142         3.1
## 143         2.7
## 144         3.2
## 145         3.3
## 146         3.0
## 147         2.5
## 148         3.0
## 149         3.4
## 150         3.0
```

```
# However, the classes of these two are not the same
class(iris[,2])  # Treated as a vector
```

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

```
class(iris[2])   # Treated as a data frame
```

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

```
# Extracting 50 rows of 2nd column
iris[1:50, 2]
```

```
##  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9 3.5 3.8
## [20] 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2 3.5 3.6
## [39] 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3
```

```
(iris[2])[1:50,]
```

```
##  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9 3.5 3.8
## [20] 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2 3.5 3.6
## [39] 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3
```

```
## Filtered Extraction
# Extract rows having 'versicolor' species
#iris[, "Species"]   # All species
iris[, "Species" == "versicolor"]   # All versicolor species
```

```
## data frame with 0 columns and 150 rows
```

```
subset(iris, Species == "versicolor")
```

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

10.3.7 Extracting Data Frames

```
mtcars[3]   # 3rd Column; class is data.frame
```

```
##                      disp
## Mazda RX4           160.0
## Mazda RX4 Wag       160.0
## Datsun 710          108.0
## Hornet 4 Drive      258.0
## Hornet Sportabout   360.0
## Valiant             225.0
## Duster 360          360.0
## Merc 240D           146.7
## Merc 230            140.8
## Merc 280            167.6
## Merc 280C           167.6
## Merc 450SE          275.8
## Merc 450SL          275.8
## Merc 450SLC         275.8
## Cadillac Fleetwood  472.0
## Lincoln Continental 460.0
## Chrysler Imperial   440.0
## Fiat 128             78.7
## Honda Civic          75.7
## Toyota Corolla       71.1
## Toyota Corona       120.1
## Dodge Challenger    318.0
## AMC Javelin         304.0
## Camaro Z28          350.0
## Pontiac Firebird    400.0
## Fiat X1-9            79.0
## Porsche 914-2       120.3
## Lotus Europa         95.1
## Ford Pantera L      351.0
## Ferrari Dino        145.0
## Maserati Bora       301.0
## Volvo 142E          121.0
```

```
mtcars[,3]  # 3rd Column; class is numeric
```

```
##  [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
## [13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0 350.0
## [25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

10.3.7.1 Using the `$` to extract data

```
mtcars$vs   # $ will display all column names to select from
```

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

```
class(mtcars$vs)
```

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

10.3.7.2 Using `attach()` and `detach()` to extract data

`attach(df)` makes the columns *locally* accessible.

`detach(df)` removes the columns previously attached from the local scope.

```
attach(mtcars)
disp
```

```
##  [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
## [13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0 350.0
## [25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

```
detach(mtcars)
#disp # Error: object 'disp' not found
```

10.3.8 Modifying Elements in a Data Frame

```
iris$Sepal.Length[1]
```

```
## [1] 5.1
```

```
iris$Sepal.Length[1] = 3.1
iris$Sepal.Length[1]
```

```
## [1] 3.1
```

10.3.9 Class Exercises

1. Create a data frame having 4 columns, each of different data types, having minimum 5 elements in each.

```
d1 <- data.frame(S.no=1:5, Name=letters[1:5], Marks=c(12.4, 12, 10, 10.54, 11.95), Status=rep
        (T, 5)); d1
```

```
##   S.no Name Marks Status
## 1    1    a 12.40   TRUE
## 2    2    b 12.00   TRUE
## 3    3    c 10.00   TRUE
## 4    4    d 10.54   TRUE
## 5    5    e 11.95   TRUE
```

```
c1 <- letters[6:10]
c2 <- round(rnorm(5))
c3 <- c(rep(T, 3), F, F)
d2 <- data.frame(c1, c2, c3); d2
```

```
##   c1 c2    c3
## 1  f -2  TRUE
## 2  g  1  TRUE
## 3  h -1  TRUE
## 4  i  0 FALSE
## 5  j  0 FALSE
```

2. Extract those rows where Sepal.Length < 5

```
subset(iris, Sepal.Length < 5)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            3.1         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
## 7            4.6         3.4          1.4         0.3     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
## 35           4.9         3.1          1.5         0.2     setosa
## 38           4.9         3.6          1.4         0.1     setosa
## 39           4.4         3.0          1.3         0.2     setosa
## 42           4.5         2.3          1.3         0.3     setosa
## 43           4.4         3.2          1.3         0.2     setosa
## 46           4.8         3.0          1.4         0.3     setosa
## 48           4.6         3.2          1.4         0.2     setosa
## 58           4.9         2.4          3.3         1.0 versicolor
## 107          4.9         2.5          4.5         1.7  virginica
```

3. Extract those rows where Sepal.Length < 5 or Petal.Length > 1.5

```
subset(iris, Sepal.Length < 5 | Petal.Length > 1.5)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            3.1         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
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 26           5.0         3.0          1.6         0.2     setosa
## 27           5.0         3.4          1.6         0.4     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
## 35           4.9         3.1          1.5         0.2     setosa
## 38           4.9         3.6          1.4         0.1     setosa
## 39           4.4         3.0          1.3         0.2     setosa
## 42           4.5         2.3          1.3         0.3     setosa
## 43           4.4         3.2          1.3         0.2     setosa
## 44           5.0         3.5          1.6         0.6     setosa
## 45           5.1         3.8          1.9         0.4     setosa
## 46           4.8         3.0          1.4         0.3     setosa
## 47           5.1         3.8          1.6         0.2     setosa
## 48           4.6         3.2          1.4         0.2     setosa
## 51           7.0         3.2          4.7         1.4 versicolor
## 52           6.4         3.2          4.5         1.5 versicolor
## 53           6.9         3.1          4.9         1.5 versicolor
## 54           5.5         2.3          4.0         1.3 versicolor
## 55           6.5         2.8          4.6         1.5 versicolor
## 56           5.7         2.8          4.5         1.3 versicolor
## 57           6.3         3.3          4.7         1.6 versicolor
## 58           4.9         2.4          3.3         1.0 versicolor
## 59           6.6         2.9          4.6         1.3 versicolor
## 60           5.2         2.7          3.9         1.4 versicolor
## 61           5.0         2.0          3.5         1.0 versicolor
## 62           5.9         3.0          4.2         1.5 versicolor
## 63           6.0         2.2          4.0         1.0 versicolor
## 64           6.1         2.9          4.7         1.4 versicolor
## 65           5.6         2.9          3.6         1.3 versicolor
## 66           6.7         3.1          4.4         1.4 versicolor
## 67           5.6         3.0          4.5         1.5 versicolor
## 68           5.8         2.7          4.1         1.0 versicolor
## 69           6.2         2.2          4.5         1.5 versicolor
## 70           5.6         2.5          3.9         1.1 versicolor
## 71           5.9         3.2          4.8         1.8 versicolor
## 72           6.1         2.8          4.0         1.3 versicolor
## 73           6.3         2.5          4.9         1.5 versicolor
## 74           6.1         2.8          4.7         1.2 versicolor
```

```
## 75          6.4         2.9          4.3         1.3 versicolor
## 76          6.6         3.0          4.4         1.4 versicolor
## 77          6.8         2.8          4.8         1.4 versicolor
## 78          6.7         3.0          5.0         1.7 versicolor
## 79          6.0         2.9          4.5         1.5 versicolor
## 80          5.7         2.6          3.5         1.0 versicolor
## 81          5.5         2.4          3.8         1.1 versicolor
## 82          5.5         2.4          3.7         1.0 versicolor
## 83          5.8         2.7          3.9         1.2 versicolor
## 84          6.0         2.7          5.1         1.6 versicolor
## 85          5.4         3.0          4.5         1.5 versicolor
## 86          6.0         3.4          4.5         1.6 versicolor
## 87          6.7         3.1          4.7         1.5 versicolor
## 88          6.3         2.3          4.4         1.3 versicolor
## 89          5.6         3.0          4.1         1.3 versicolor
## 90          5.5         2.5          4.0         1.3 versicolor
## 91          5.5         2.6          4.4         1.2 versicolor
## 92          6.1         3.0          4.6         1.4 versicolor
## 93          5.8         2.6          4.0         1.2 versicolor
## 94          5.0         2.3          3.3         1.0 versicolor
## 95          5.6         2.7          4.2         1.3 versicolor
## 96          5.7         3.0          4.2         1.2 versicolor
## 97          5.7         2.9          4.2         1.3 versicolor
## 98          6.2         2.9          4.3         1.3 versicolor
## 99          5.1         2.5          3.0         1.1 versicolor
## 100         5.7         2.8          4.1         1.3 versicolor
## 101         6.3         3.3          6.0         2.5  virginica
## 102         5.8         2.7          5.1         1.9  virginica
## 103         7.1         3.0          5.9         2.1  virginica
## 104         6.3         2.9          5.6         1.8  virginica
## 105         6.5         3.0          5.8         2.2  virginica
## 106         7.6         3.0          6.6         2.1  virginica
## 107         4.9         2.5          4.5         1.7  virginica
## 108         7.3         2.9          6.3         1.8  virginica
## 109         6.7         2.5          5.8         1.8  virginica
## 110         7.2         3.6          6.1         2.5  virginica
## 111         6.5         3.2          5.1         2.0  virginica
## 112         6.4         2.7          5.3         1.9  virginica
## 113         6.8         3.0          5.5         2.1  virginica
## 114         5.7         2.5          5.0         2.0  virginica
## 115         5.8         2.8          5.1         2.4  virginica
## 116         6.4         3.2          5.3         2.3  virginica
## 117         6.5         3.0          5.5         1.8  virginica
## 118         7.7         3.8          6.7         2.2  virginica
## 119         7.7         2.6          6.9         2.3  virginica
## 120         6.0         2.2          5.0         1.5  virginica
## 121         6.9         3.2          5.7         2.3  virginica
## 122         5.6         2.8          4.9         2.0  virginica
## 123         7.7         2.8          6.7         2.0  virginica
## 124         6.3         2.7          4.9         1.8  virginica
## 125         6.7         3.3          5.7         2.1  virginica
## 126         7.2         3.2          6.0         1.8  virginica
## 127         6.2         2.8          4.8         1.8  virginica
## 128         6.1         3.0          4.9         1.8  virginica
## 129         6.4         2.8          5.6         2.1  virginica
## 130         7.2         3.0          5.8         1.6  virginica
```

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

4. Extract those rows where Petal.Width > 0.2 and Species=setosa

```
subset(iris, Petal.Width > 0.2 & Species == "setosa")
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
## 18          5.1         3.5          1.4         0.3  setosa
## 19          5.7         3.8          1.7         0.3  setosa
## 20          5.1         3.8          1.5         0.3  setosa
## 22          5.1         3.7          1.5         0.4  setosa
## 24          5.1         3.3          1.7         0.5  setosa
## 27          5.0         3.4          1.6         0.4  setosa
## 32          5.4         3.4          1.5         0.4  setosa
## 41          5.0         3.5          1.3         0.3  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 44          5.0         3.5          1.6         0.6  setosa
## 45          5.1         3.8          1.9         0.4  setosa
## 46          4.8         3.0          1.4         0.3  setosa
```

10.3.10 Homework

1. Delete/Add any column

```
# deleting a column
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  3.1 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 ...
```

```
iris <- iris[, -1]

str(iris)
```

```
## 'data.frame':    150 obs. of  4 variables:
##  $ 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 ...
```

```
# adding a column
iris$Sepal.Length = round(runif(150, 3, 5), 1)
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ 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 ...
##  $ Sepal.Length: num  4.4 3.1 4.9 4.6 3.2 3 3.3 4.8 4.8 4.1 ...
```

2. Delete/Add some row

```
# deleting a row
iris <- iris[-nrow(iris),]

str(iris)
```

```
## 'data.frame':    149 obs. of  5 variables:
##  $ 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 ...
##  $ Sepal.Length: num  4.4 3.1 4.9 4.6 3.2 3 3.3 4.8 4.8 4.1 ...
```

```
# add a row
iris <- rbind(iris, c(4.4, 2.1, 1.6, 0.2, "setosa"))
```

```
## Warning in `[<-.factor`(`*tmp*`, ri, value = "0.2"): invalid factor level, NA
## generated
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Width : chr  "3.5" "3" "3.2" "3.1" ...
##  $ Petal.Length: chr  "1.4" "1.4" "1.3" "1.5" ...
##  $ Petal.Width : chr  "0.2" "0.2" "0.2" "0.2" ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sepal.Length: chr  "4.4" "3.1" "4.9" "4.6" ...
```

```
View(iris)
```

10.4 Lists

10.4.1 Creating a List

Syntax: `list(obj1, obj2, ...)`

`obj` (Objects) can be anything (vectors, matrices, data frames, lists)

```
obj1 = 1:10
obj2 = month.name
obj3 = matrix(1:16, 4)
obj4 = head(iris)  # First 6 rows of df 'iris'
obj5 = tail(iris)  # Last 6 rows of df 'iris'

p1 = list(obj1, obj2, obj3, obj4, obj5); p1
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
##
## [[4]]
##   Sepal.Width Petal.Length Petal.Width Species Sepal.Length
## 1         3.5          1.4         0.2  setosa          4.4
## 2           3          1.4         0.2  setosa          3.1
## 3         3.2          1.3         0.2  setosa          4.9
## 4         3.1          1.5         0.2  setosa          4.6
## 5         3.6          1.4         0.2  setosa          3.2
## 6         3.9          1.7         0.4  setosa            3
##
## [[5]]
##     Sepal.Width Petal.Length Petal.Width   Species Sepal.Length
## 145         3.3          5.7         2.5 virginica          3.1
## 146           3          5.2         2.3 virginica          3.3
## 147         2.5            5         1.9 virginica            5
## 148           3          5.2           2 virginica          3.4
## 149         3.4          5.4         2.3 virginica          4.9
## 150         4.4          2.1         1.6      <NA>       setosa
```

10.4.2 Number of Elements in a List

```
length(p1)
```

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

10.4.3 Naming List Elements

```
names(p1) = c("vect1", "vect2", "Matrix", "dataframe1", "dataframe2")
names(p1)
```

```
## [1] "vect1"      "vect2"      "Matrix"     "dataframe1" "dataframe2"
```

```
p1
```

```
## $vect1
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $vect2
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
##
## $Matrix
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
##
## $dataframe1
##   Sepal.Width Petal.Length Petal.Width Species Sepal.Length
## 1         3.5          1.4         0.2  setosa          4.4
## 2           3          1.4         0.2  setosa          3.1
## 3         3.2          1.3         0.2  setosa          4.9
## 4         3.1          1.5         0.2  setosa          4.6
## 5         3.6          1.4         0.2  setosa          3.2
## 6         3.9          1.7         0.4  setosa            3
##
## $dataframe2
##     Sepal.Width Petal.Length Petal.Width   Species Sepal.Length
## 145         3.3          5.7         2.5 virginica          3.1
## 146           3          5.2         2.3 virginica          3.3
## 147         2.5            5         1.9 virginica            5
## 148           3          5.2           2 virginica          3.4
## 149         3.4          5.4         2.3 virginica          4.9
## 150         4.4          2.1         1.6      <NA>       setosa
```

10.4.4 Accessing Elements and Sub-elements of a List

```
# Elements of a list
p1[[2]]
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

```
p1$vect1
```

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

```
# Sub-Elements of a List
(p1[[2]])[9]
```

```
## [1] "September"
```

```
(p1[[3]])[4,4]
```

```
## [1] 16
```

10.4.5 Modifying elements

```
p1[[2]][9] = "Sept."
p1[[2]][9]
```

```
## [1] "Sept."
```

10.4.6 Class Exercises

1. Modify the 16 number by 26 in the matrix element of the list.

```
obj1 = 1:10
obj2 = month.name
obj3 = matrix(1:16, 4)
obj4 = head(iris)  # First 6 rows of df 'iris'

p1 = list(obj1, obj2, obj3, obj4); p1
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
##
## [[4]]
##   Sepal.Width Petal.Length Petal.Width Species Sepal.Length
## 1         3.5          1.4         0.2  setosa          4.4
## 2           3          1.4         0.2  setosa          3.1
## 3         3.2          1.3         0.2  setosa          4.9
## 4         3.1          1.5         0.2  setosa          4.6
## 5         3.6          1.4         0.2  setosa          3.2
## 6         3.9          1.7         0.4  setosa            3
```

```
names(p1) = c("vect1", "vect2", "Matrix", "dataframe")

p1[[3]][4,4] = 26
p1[[3]][4,4]
```

```
## [1] 26
```

```
p1$Matrix[4,4] = 25
p1$Matrix[4,4]
```

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

```
p1
```

```
## $vect1
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $vect2
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
##
## $Matrix
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   25
##
## $dataframe
##   Sepal.Width Petal.Length Petal.Width Species Sepal.Length
## 1         3.5          1.4         0.2  setosa          4.4
## 2           3          1.4         0.2  setosa          3.1
## 3         3.2          1.3         0.2  setosa          4.9
## 4         3.1          1.5         0.2  setosa          4.6
## 5         3.6          1.4         0.2  setosa          3.2
## 6         3.9          1.7         0.4  setosa            3
```

2. Add one more element to the list

```
p1[[length(p1)+1]] = seq(1,100, length=12)
names(p1)[length(p1)] = "Sequence"
p1[[length(p1)]]
```

```
##  [1]   1  10  19  28  37  46  55  64  73  82  91 100
```

3. Remove the first element of the list

```
p1 <- p1[-1]
p1; length(p1)
```

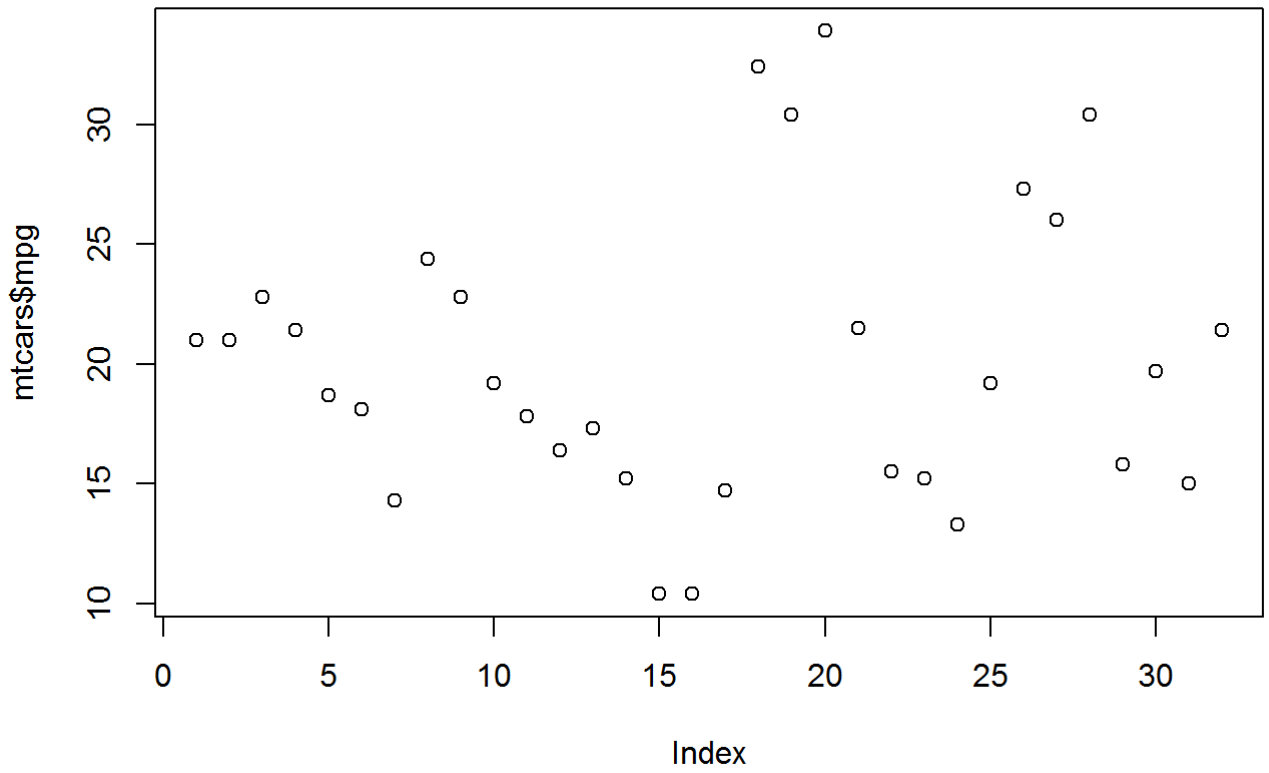
```
## $vect2
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
##
## $Matrix
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   25
##
## $dataframe
##   Sepal.Width Petal.Length Petal.Width Species Sepal.Length
## 1         3.5          1.4         0.2  setosa          4.4
## 2           3          1.4         0.2  setosa          3.1
## 3         3.2          1.3         0.2  setosa          4.9
## 4         3.1          1.5         0.2  setosa          4.6
## 5         3.6          1.4         0.2  setosa          3.2
## 6         3.9          1.7         0.4  setosa            3
##
## $Sequence
##  [1]   1  10  19  28  37  46  55  64  73  82  91 100
```

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

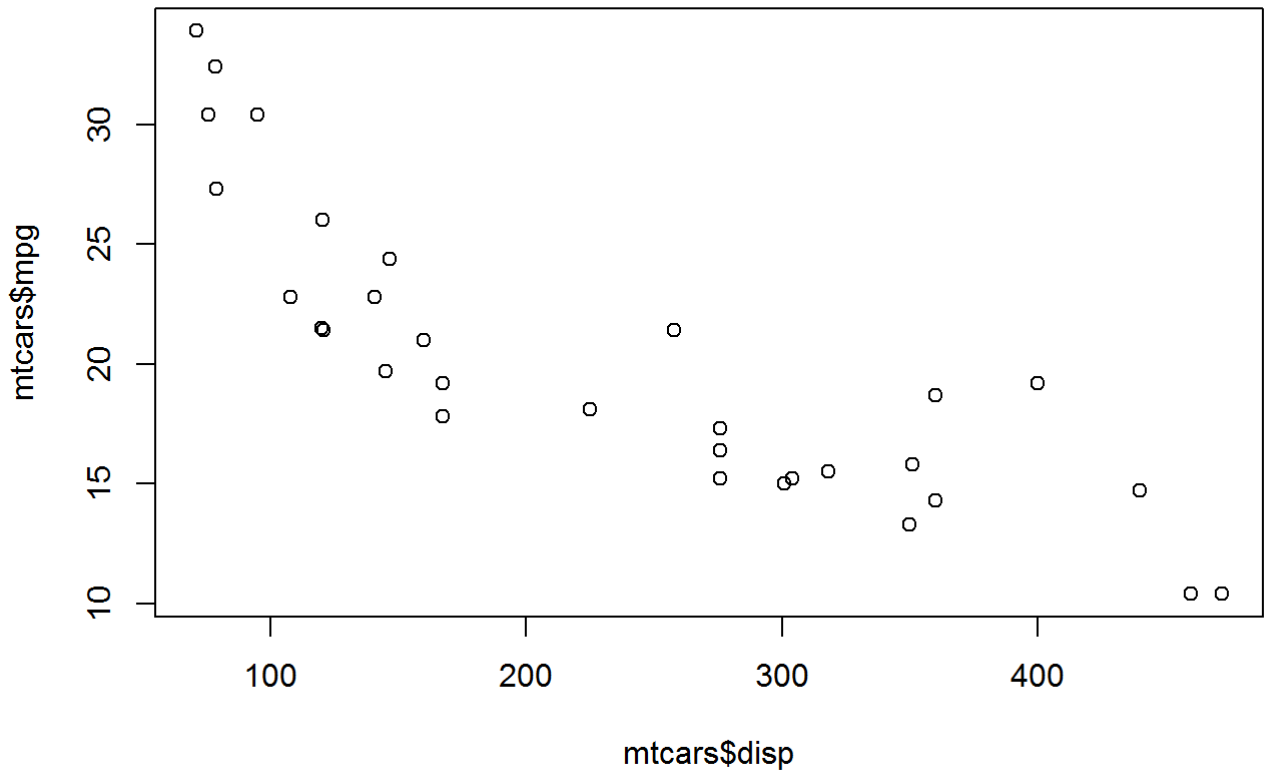
11 Plotting

11.1 attach() and detach()

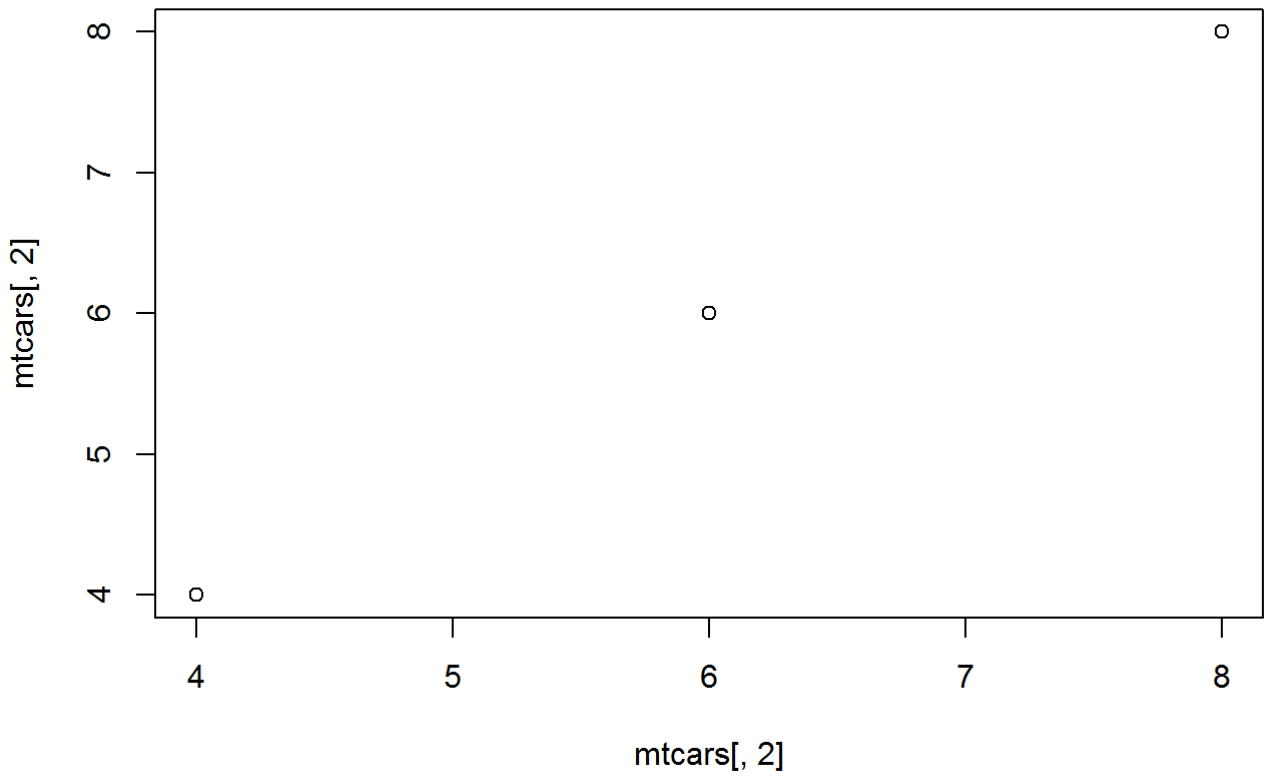
```
# Plotting with attach() and detach()
attach(mtcars)
plot(mtcars$mpg)
```




```
detach(mtcars)

# Plotting without attach/detach

plot(mtcars$disp, mtcars$mpg)
```




```
plot(mtcars[,2], mtcars[,2])
```



```
# plot() cannot accept data.frame values, instead a vector is passed into it in this case
```

11.2 Scatter Plot: `plot()`

Creates a scatter plot

Syntax:

`plot(x, y, ...)` or `plot(y~x, ...)` is used to plot discrete points

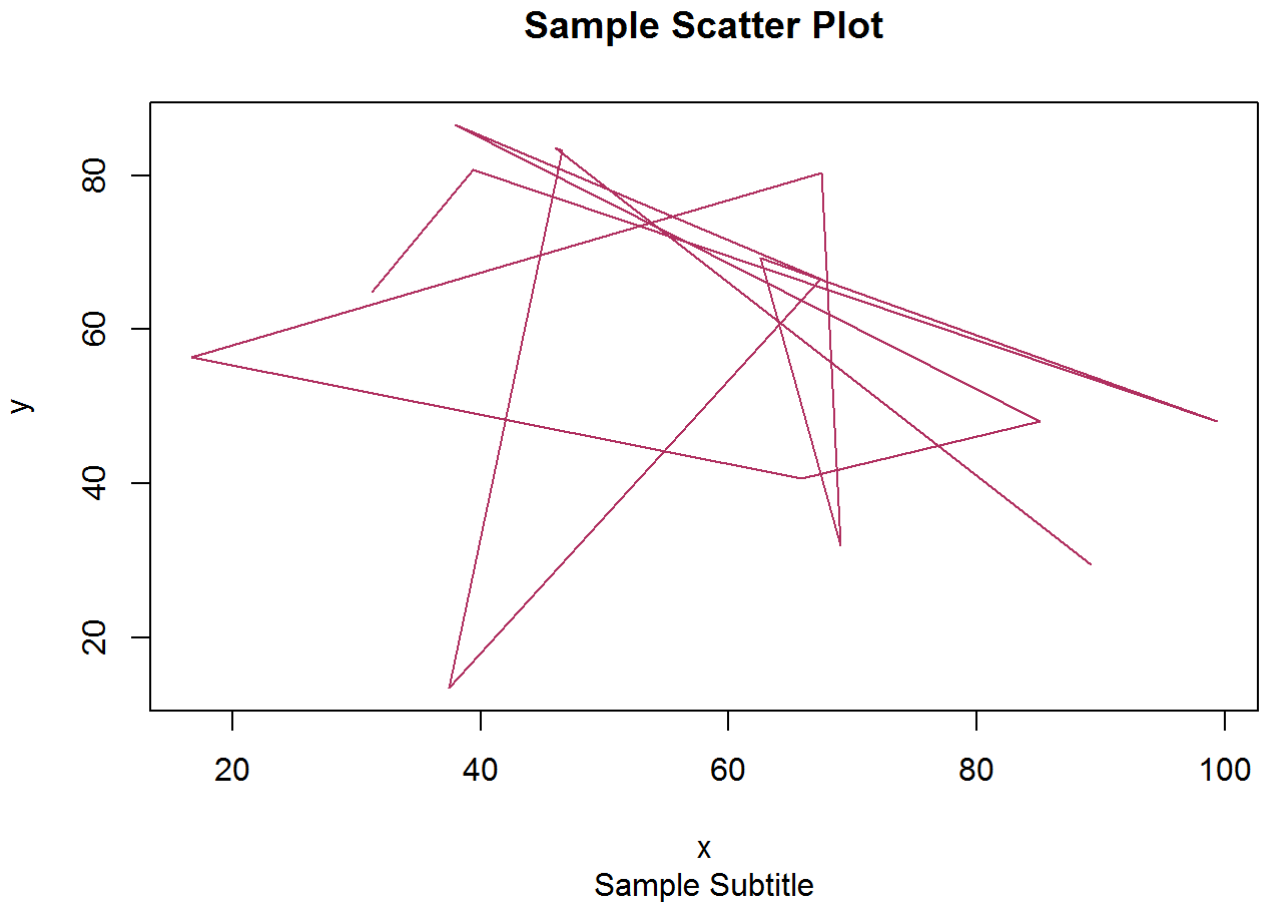
Additional Arguments:

1. `col` : changes the colour of the dots
2. `pch` : changes the `pch` (plotting character) default is a hollow circle (pch=1)
3. `main` : changes the title of the plot
4. `type` : type of plot a. `p` - default, point b. `l` - **l**ine c. `o` - **o**verlapping/ overplotted of line and point
5. `cex` : **c**haracter **ex**pansion (if plot has points)
6. `lty` : **l**ine **ty**pe (if plot has lines)
7. `lwd` : **l**ine **wi****d**th (if plot has lines)
8. `sub` : **Sub**title
9. `xlab` : x-label
10. `ylab` : y-label
11. `xlim` : restrict the domain (start and end points) i.e. c(a, b)
12. `ylim` : restrict the range (start and end points) i.e. c(a, b) Use `help(plot)` to explore more.

11.3 Discrete / Isolated Points

Plot (1,2), (3,5), (-3,6), (0,2)

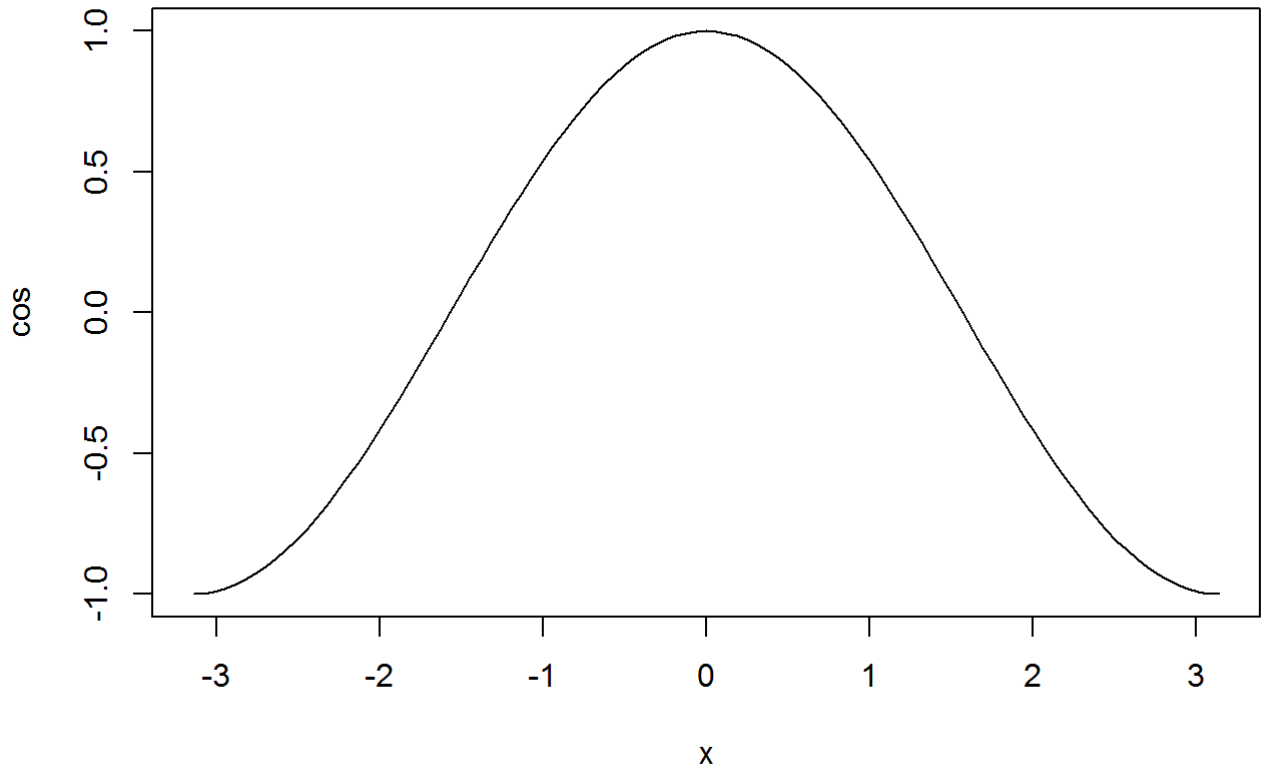
```
x <- c(1, 3, -3, 0)
y <- c(2, 5, 6 , 2)

x <- runif(15, 1, 100)
y <- runif(15, 1, 100)

#plot(y~x)
#plot(x, y, col="red", pch=8, cex=0.4, main="Sample Scatter Plot", type="p")
plot(x, y, col="maroon", pch=8, cex=0.4, main="Sample Scatter Plot", sub="Sample Subtitle", t
         ype="l", lty=1, lwd=1)
```



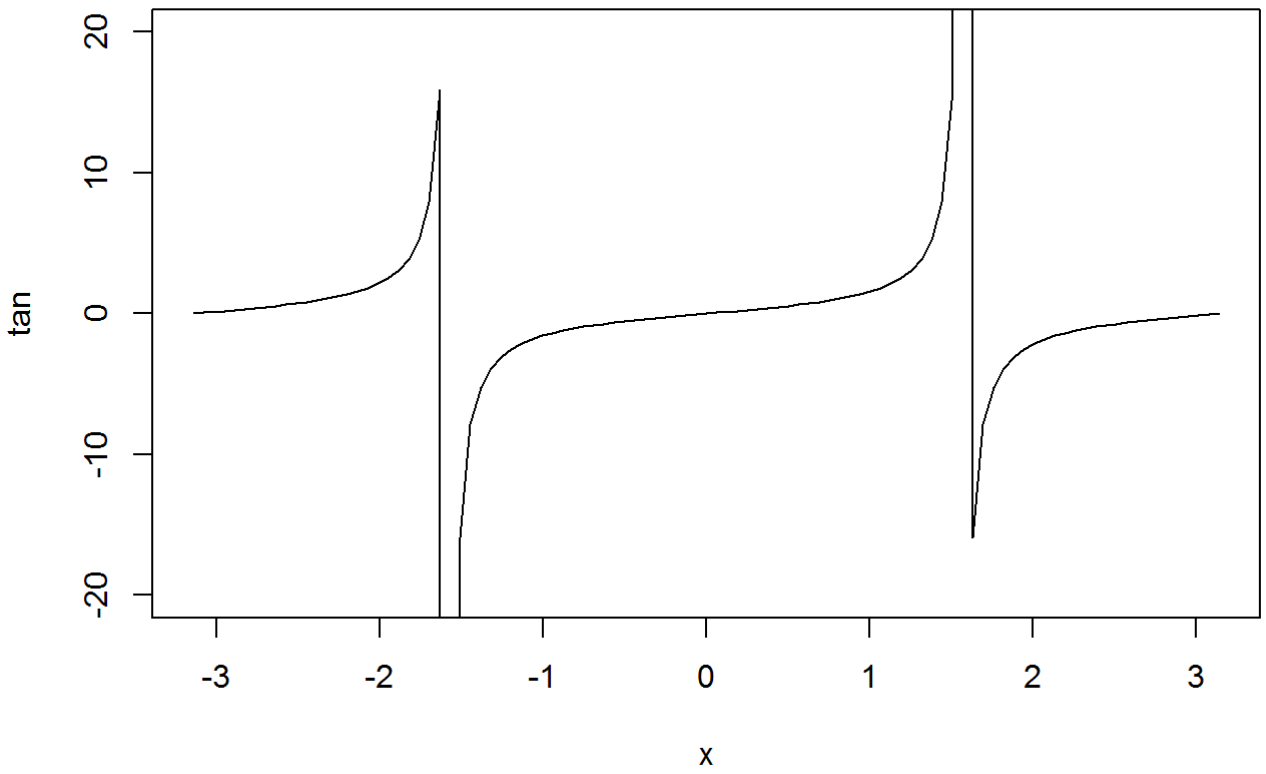
11.4 Pre-defined functions (trigonometric, exp, log)

Syntax: `plot(f, a, b)`

```
plot(cos, -pi, pi)
```



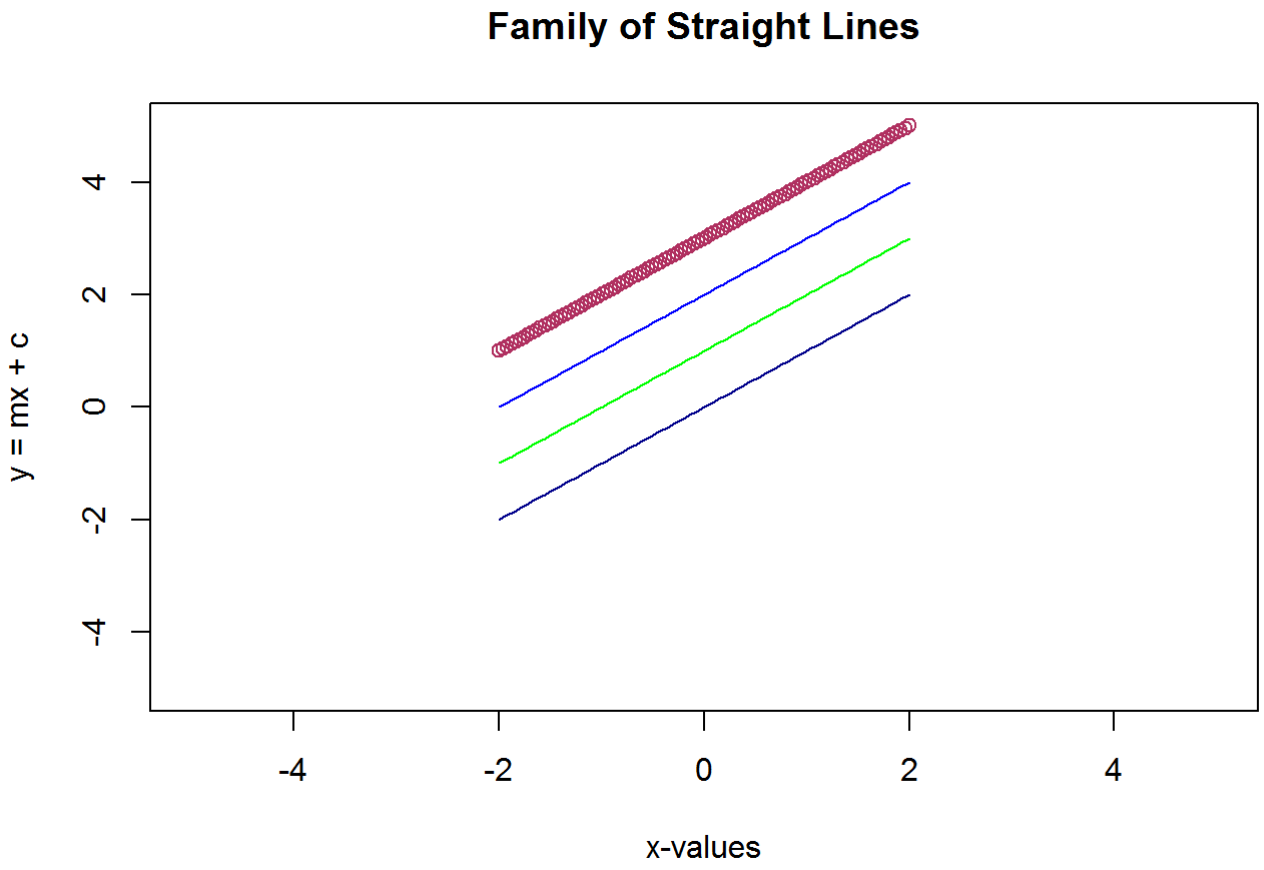

```
plot(tan, -pi, pi, ylim=c(-20,20))
```



11.5 Explicit Functions `y = f(x)`

Plot the family of curves $y = x + c$, f.s. $c \in \mathbf{R}$

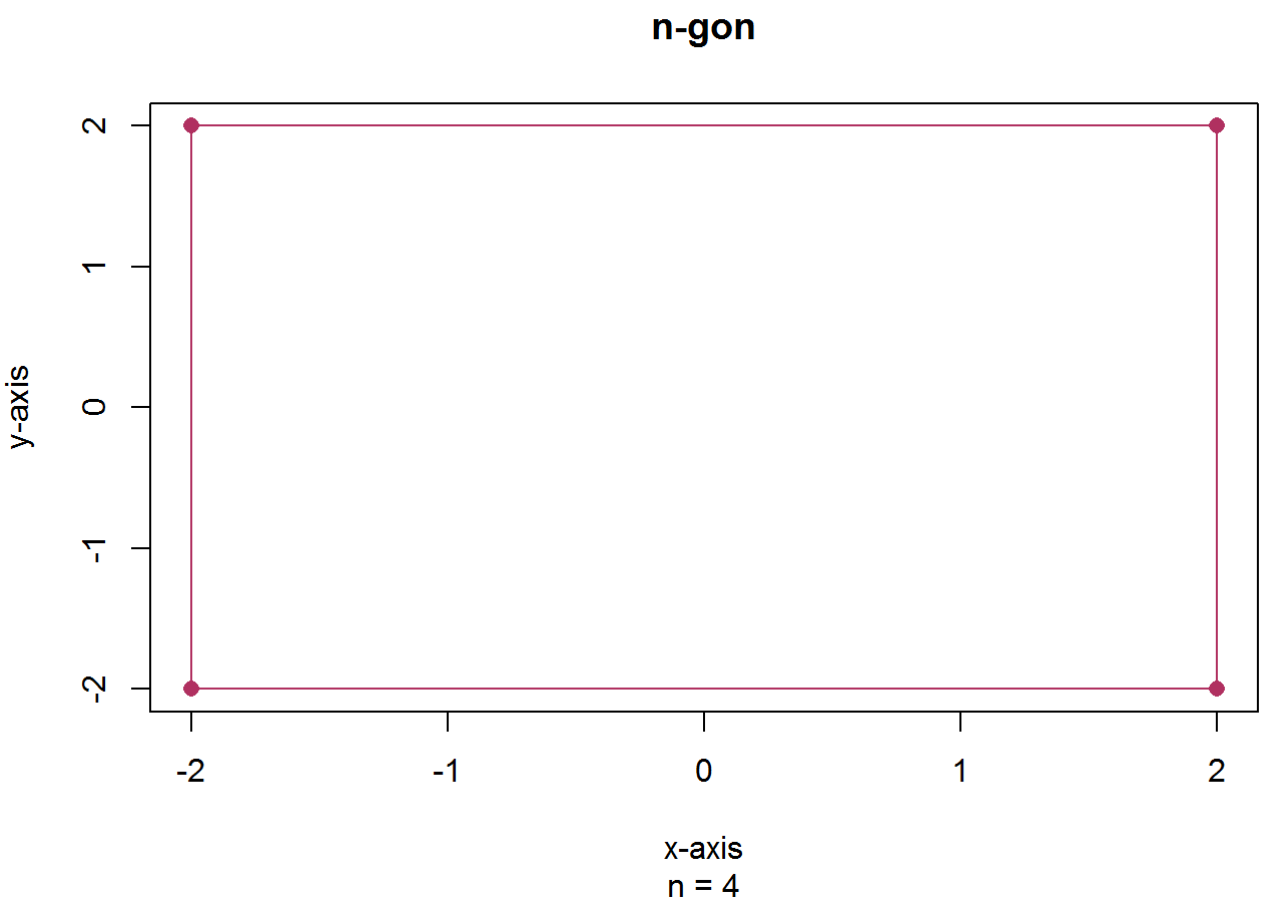
```
x = seq(-2, 2, length=100)
plot(x, x,
     type="l",
     col="darkblue",
     main="Family of Straight Lines",
     xlab="x-values",
     ylab="y = mx + c",
     ylim=c(-5, 5),
     xlim=c(-5, 5))
# To overlap the curves on the same graph, we can use lines() and points()commands
# plot(x, x+1, type="l", col="darkblue")
lines(x, x+1, col="green")
points(x, x+3, col="maroon")
lines(x, x+2, col="blue")
```



11.6 Class Exercises

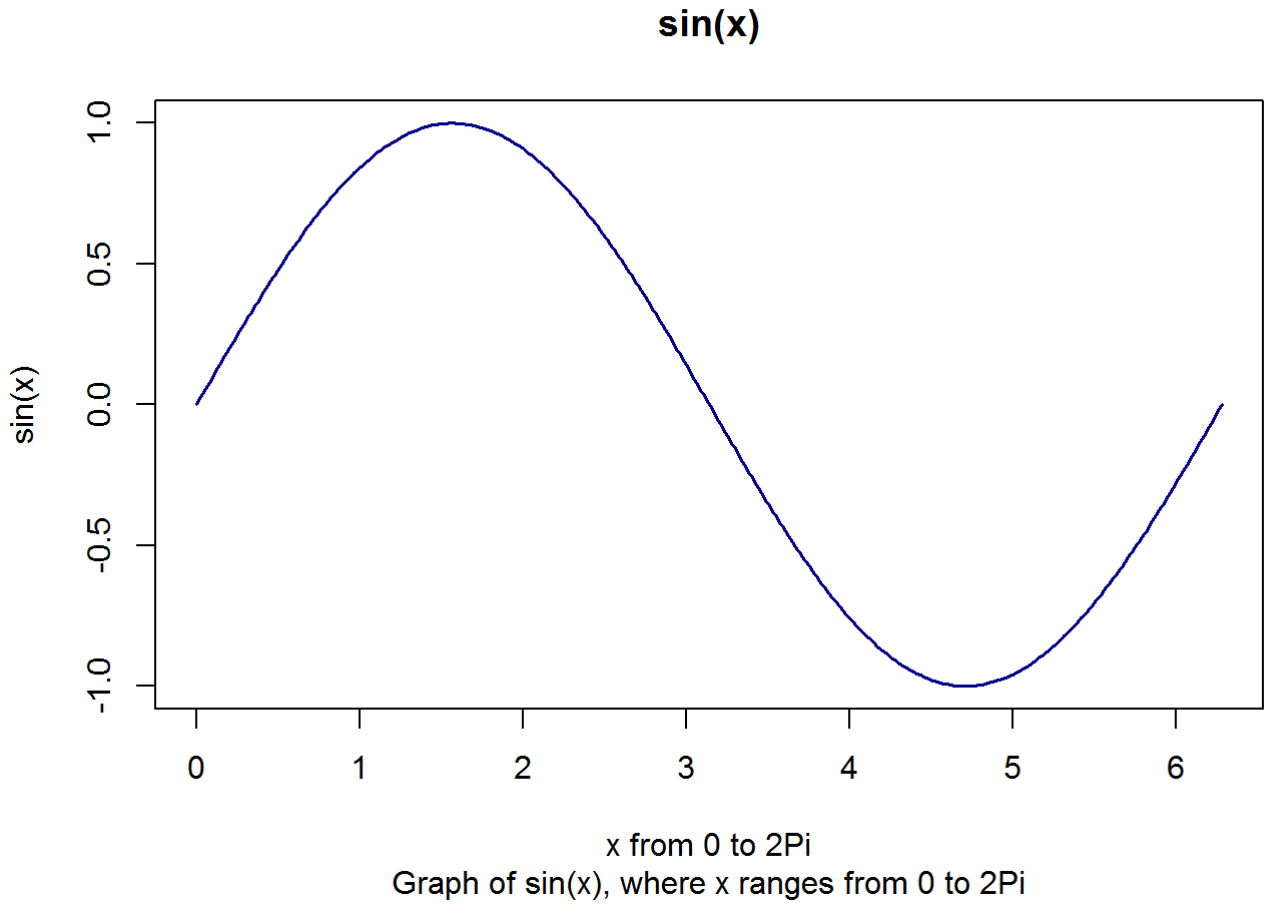
1. Plot an n-gon with $n > 3$

```
x <- c(-2,  2, 2, -2, -2)
y <- c(-2, -2, 2,  2, -2)

plot(x, y, col="maroon", pch=16, cex=1.1, main="n-gon", sub="n = 4", type="o", lty=1, lwd=1,
         xlab="x-axis", ylab="y-axis")
```



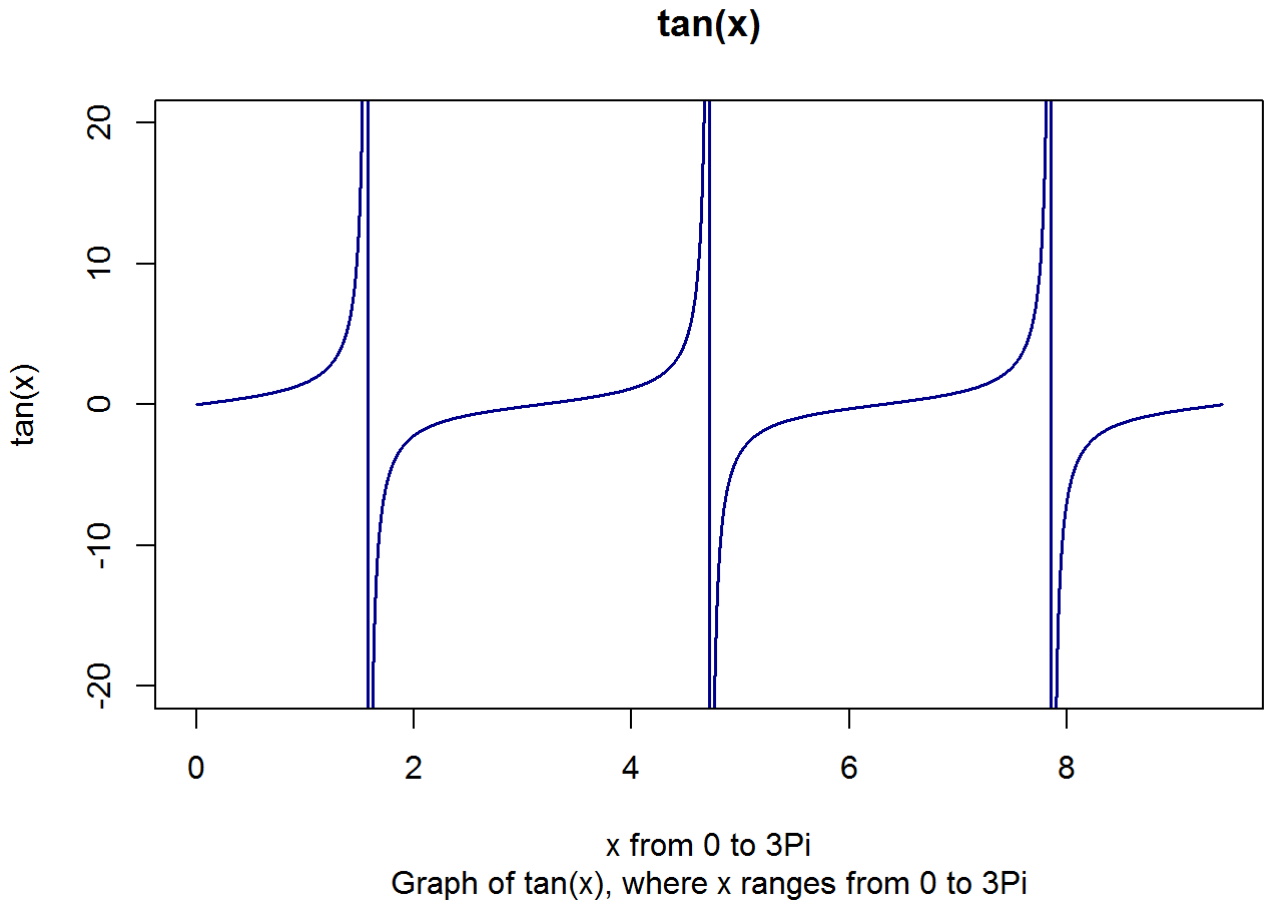
2. Plot $y = \sin(x),\ x \in [0, 2\pi]$

```
x = seq(0, 2*pi, length=300)
y = sin(x)

plot(y~x, main="sin(x)", xlab="x from 0 to 2Pi", ylab="sin(x)", sub="Graph of sin(x), where x
         ranges from 0 to 2Pi", col="darkblue", type="l", lwd=1.6)
```



3. Plot $y = \tan(x),\ x \in [0, 3\pi]$

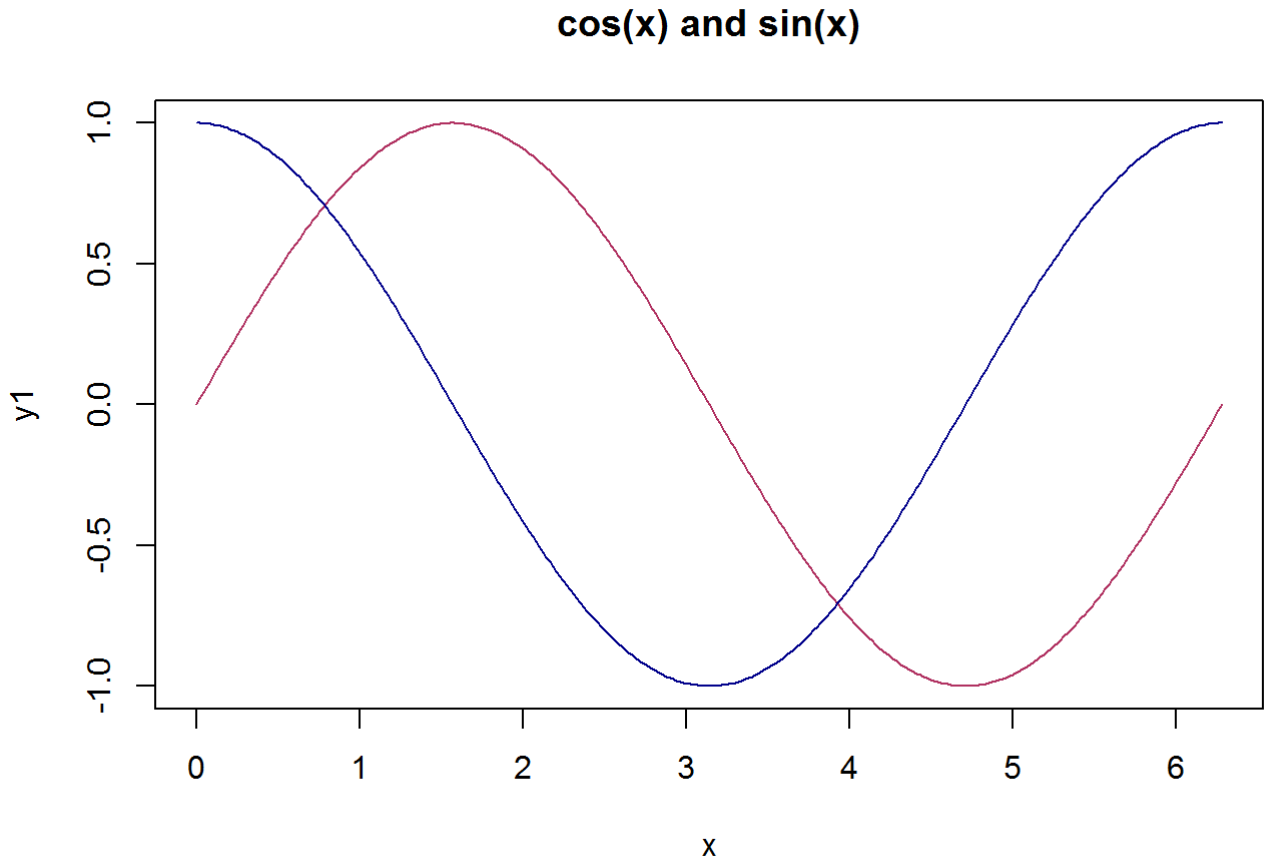
```
x = seq(0, 3*pi, length=10000)
y = tan(x)

plot(y~x, main="tan(x)", xlab="x from 0 to 3Pi", ylab="tan(x)", sub="Graph of tan(x), where x
         ranges from 0 to 3Pi", col="darkblue", type="l", lwd=1.6, ylim=c(-20, 20))
```



Graph of tan(x), where x ranges from 0 to 3Pi

4. Plot $y = \sin(x)$ and $y = \cos(x)$ together

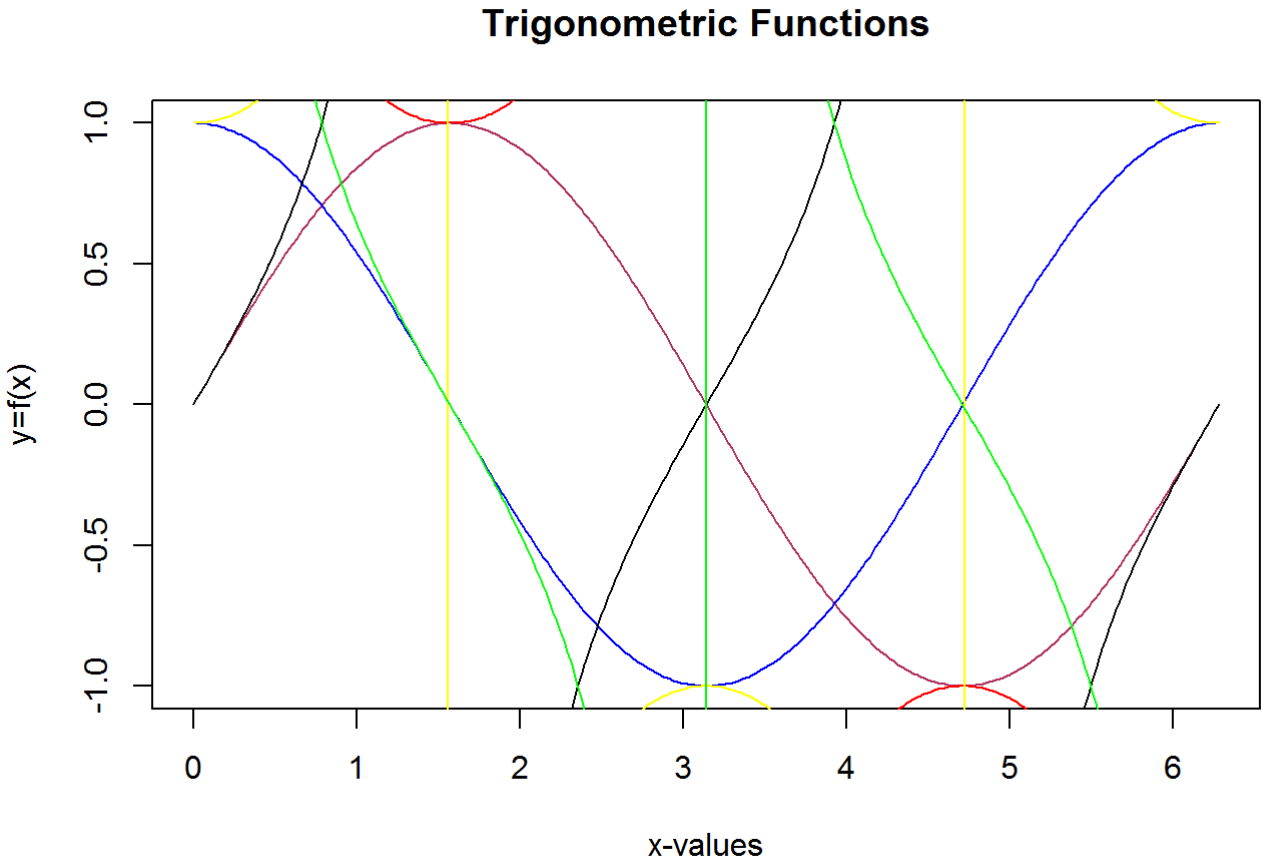
```
x = seq(0, 2*pi, length=300)
y1 = sin(x)
y2 = cos(x)

plot(y1~x,
     type="l",
     main="cos(x) and sin(x)",
     col="maroon")
lines(y2~x,
      col="darkblue")
```



11.7 HW

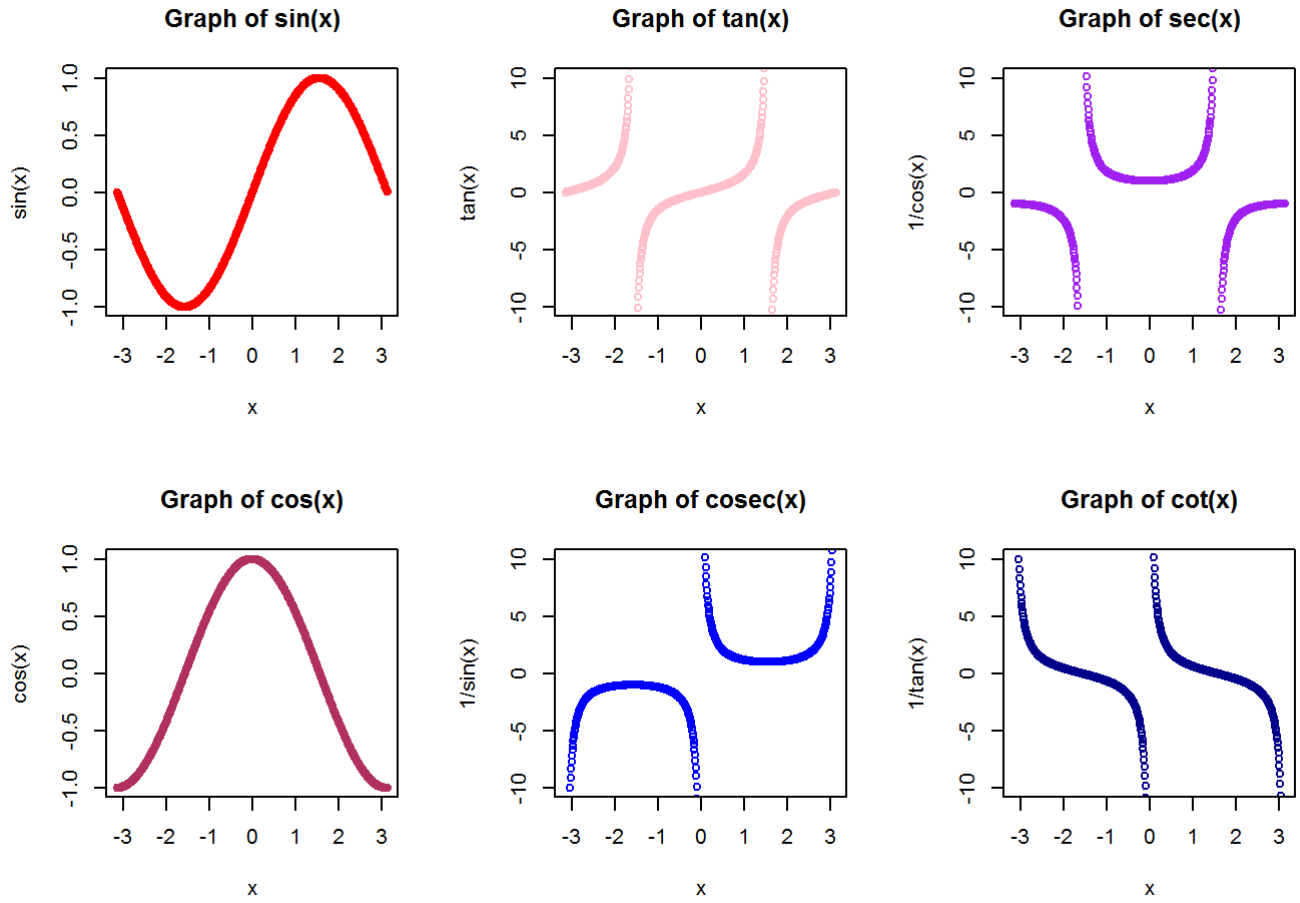
1. Explore how to plot more than one function one graph

```
x = seq(0, 2*pi, length=300)
y1 = sin(x)
y2 = cos(x)
y3 = tan(x)
y4 = 1/y1
y5 = 1/y2
y6 = 1/y3

plot(y1~x,
     main="Trigonometric Functions",
     xlab="x-values",
     ylab="y=f(x)",
     type="l",
     col="maroon")

# lines() is used for overlapping curves with type="l"
lines(y2~x, col="blue")
lines(y3~x, col="black")
lines(y4~x, col="red")
lines(y5~x, col="yellow")
lines(y6~x, col="green")
```



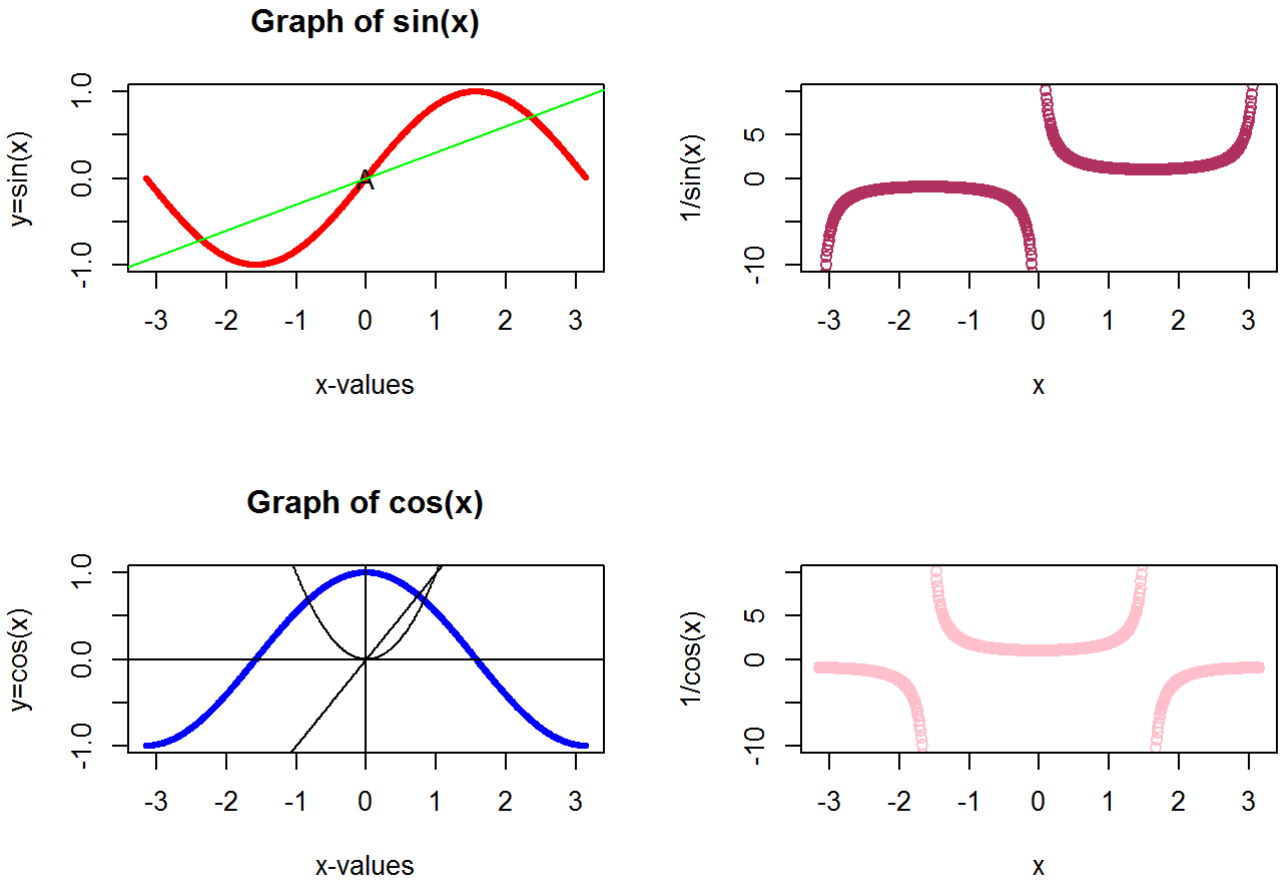
2. Explore how to plot all trigonometric functions in different graphs but together in 2 columns and 3 rows

```
x = seq(-pi, pi, 0.01)

layout(matrix(1:6, 2))
plot(x, sin(x),
     main="Graph of sin(x)",
     cex=0.8,
     col="red")

plot(x, cos(x),
     main="Graph of cos(x)",
     cex=0.8,
     col="maroon")

plot(x, tan(x),
     main="Graph of tan(x)",
     cex=0.8,
     col="pink",
     ylim=c(-10, 10))

plot(x, 1/sin(x),
     main="Graph of cosec(x)",
     cex=0.8,
     col="blue",
     ylim=c(-10, 10))

plot(x, 1/cos(x),
     main="Graph of sec(x)",
     cex=0.8,
     col="purple",
     ylim=c(-10, 10))

plot(x, 1/tan(x),
     main="Graph of cot(x)",
     cex=0.8,
     col="darkblue",
     ylim=c(-10, 10))
```



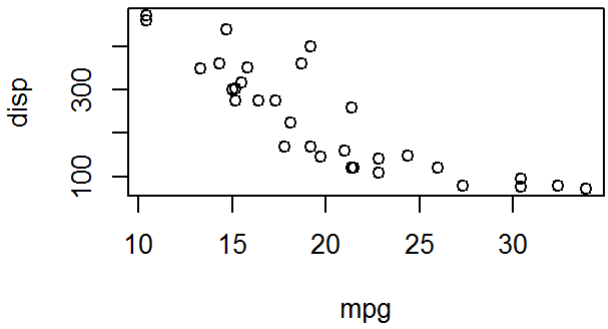
3. Plot 4 Functions together in a 2x2 matrix margin

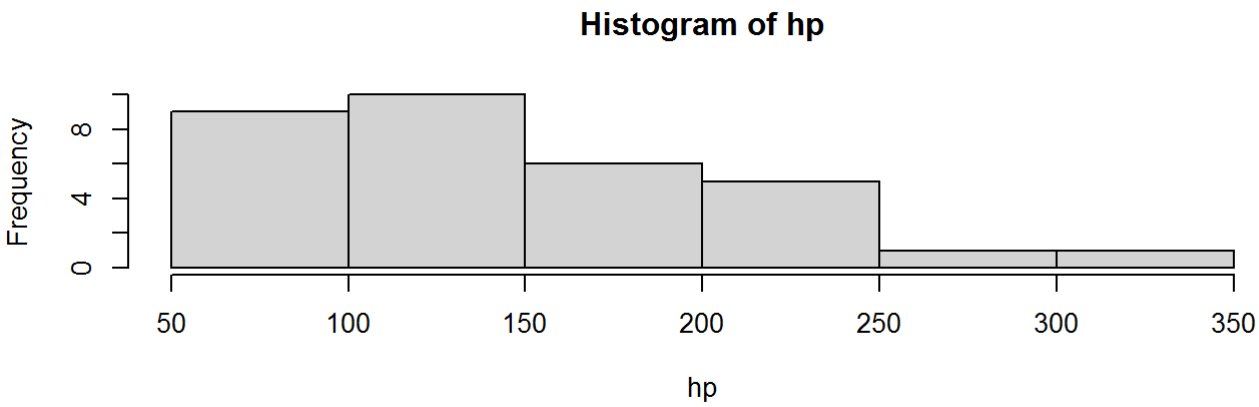
```r
# Option (1/2)
# par() command
x = seq(-pi, pi, 0.01)
par(mfcol=c(2, 2)) # creates space
plot(sin(x)~x,
     col="red",
     main="Graph of sin(x)",
     ylab="y=sin(x)",
     xlab="x-values",
     cex=0.5)
text(0, 0, "A")

# To add a best-fit line to scatter plot
# syntax: abline(a=intercept, b=slope, h=horizontal_line, v=vertical_line)
abline(lm(sin(x)~x), col="green")

# lm(): linear model, which provides the value of the slope and intercept

plot(cos(x)~x,
     col="blue",
     main="Graph of cos(x)",
     ylab="y=cos(x)",
     xlab="x-values",
     cex=0.5)
lines(x, x)
lines(x, x^2)
abline(h=0, v=0) # To add a straight line of which you know the slope and intercept
#abline(h=c(0, 1), v=0) # Multiple Lines

plot(1/sin(x)~x,
     col="maroon",
     ylim=c(-10, 10))

plot(1/cos(x)~x,
     col="pink",
     ylim=c(-10, 10))
```



4. Plot 3 different graphs in a 2x2 matrix layout

```
# Option (2/2)
# layout() command
# layout(matrix(c(i1, i2, ..., in), m, n, byrow=F))

attach(mtcars)
layout(matrix(c(1, 2, 3, 3), 2, 2, byrow=T))
plot(mpg, disp)
pie(wt)
hist(hp)
```

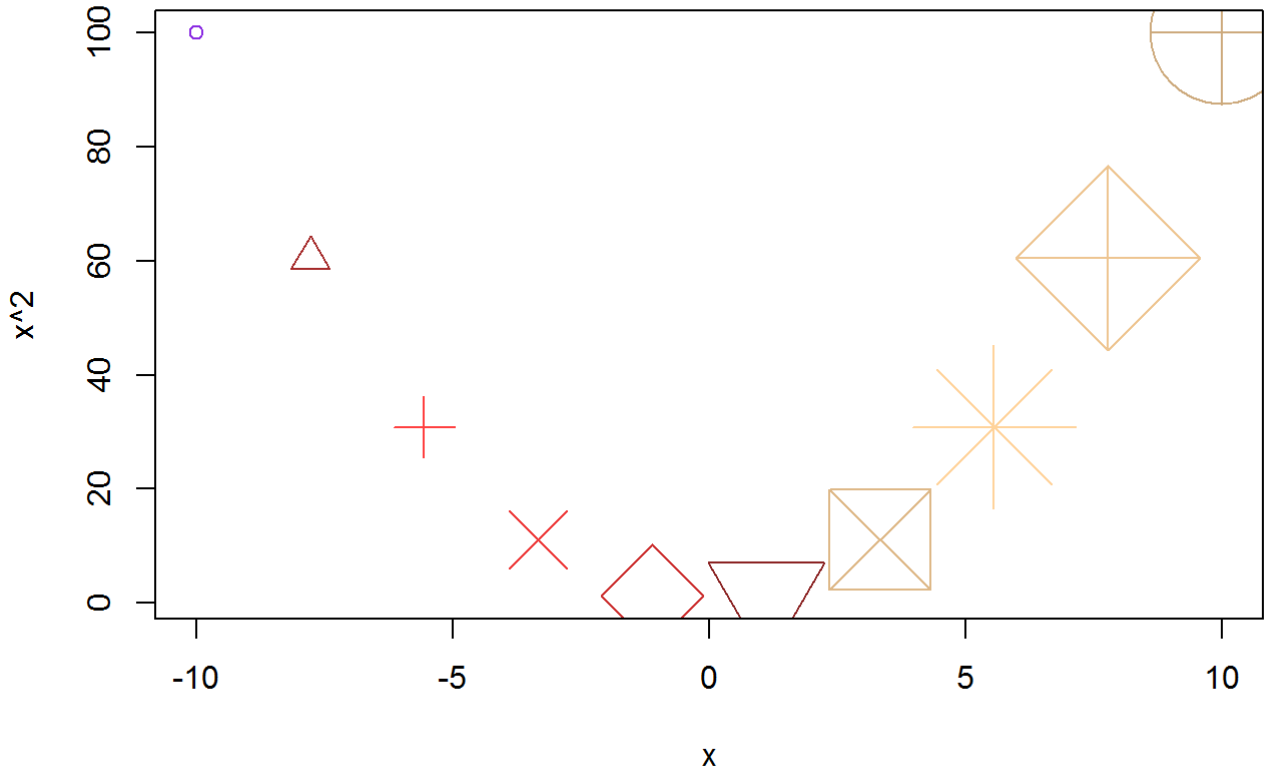





```
detach(mtcars)
```

5. Make a plot showing all the different symbols in the same graph

```
x <- seq(-10, 10, length=10)
plot(x, x^2, pch=1:10, col=colors()[31:40], cex=1:10)
```



12 Importing and Exporting Files

12.1 Directory commands

12.1.1 `getwd()`

This returns the current working directory **get w**orking **d**irectory

```
getwd()
```

12.1.2 `dir()`

This returns all items in the directory (if empty then it lists for the current directory).

```
dir()
# Folders do not have a file extension.
```

12.1.3 `setwd(path)`

To change the working directory to `path`

```
main <- getwd()
dir()
setwd("merged")
getwd()
dir()
setwd("D:\\Semester I\\SEC - R\\merged")
dir()
setwd(main)
getwd()
```

12.1.4 `file.choose()`

```
# file.choose() enables us to choose the file on our own at any location on the Disk/State Dr
ive
scan(file.choose())
scan("D:\\Semester I\\SEC - R\\nums.txt")
scan("D:\\Semester I\\SEC - R\\nums.csv", sep=",")
```

13 Analysis - I

13.1 `read.csv()`

This is used when the data is **separated by commas** and numbers use **periods as decimals**

```
d1 <- read.csv("D:\\Semester I\\SEC - R\\nums.csv") # First row is the header
#read.csv(file.choose(), header=F)

class(d1) # data frame
d1
```

13.2 `read.csv2()`

This is used when the data is **separated by semicolons** and numbers use **commas as decimals**

13.3 `read.delim()`

This is used when the numbers use **periods as decimals**

13.4 `read.delim2()`

This is used when the numbers use **commas as decimals**

13.5 `read.table()`

All entries are clubbed in a single column. Default separator is ""

```
read.csv("data1.csv", sep=",")
read.csv2("data1.csv", sep=",")
read.delim("data1.csv", sep=".")
read.delim2("data1.csv", sep=",")
read.table("data1.csv", sep=",")
```

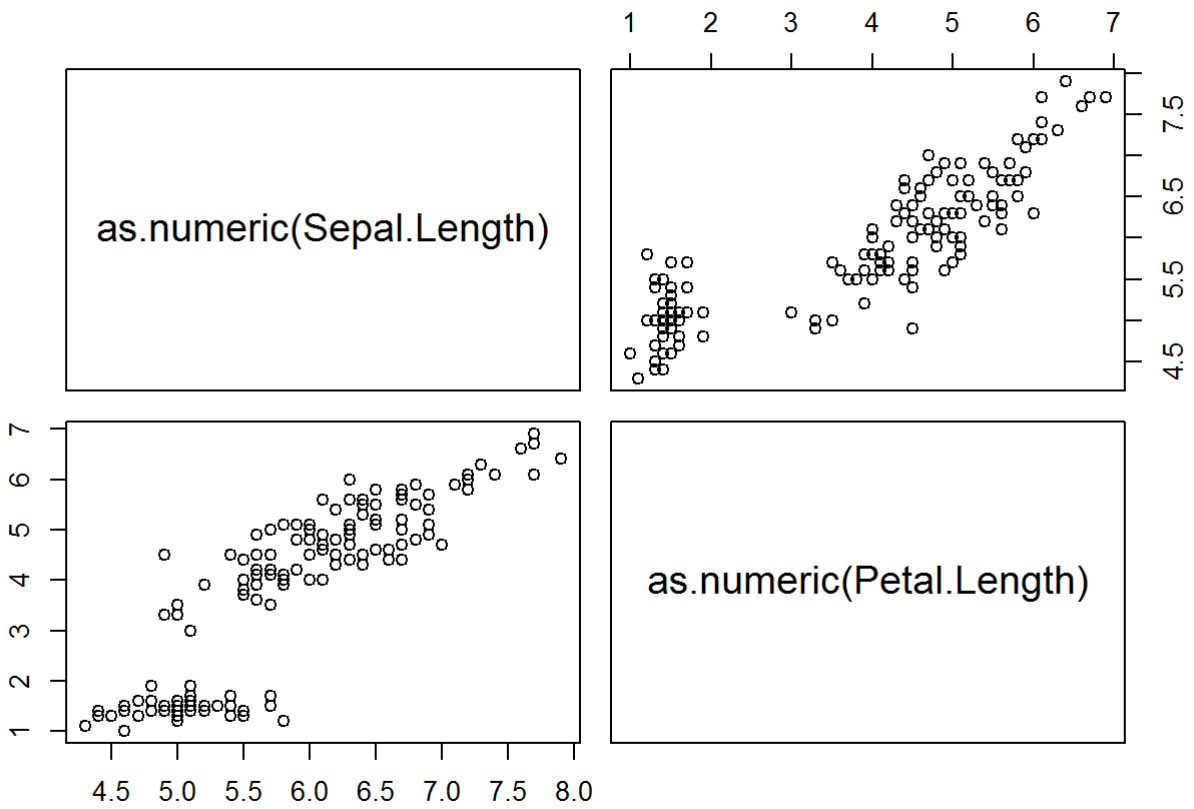
14 Analysis - II

14.1 Pairs Plotting (Multiple Correlation Plot)

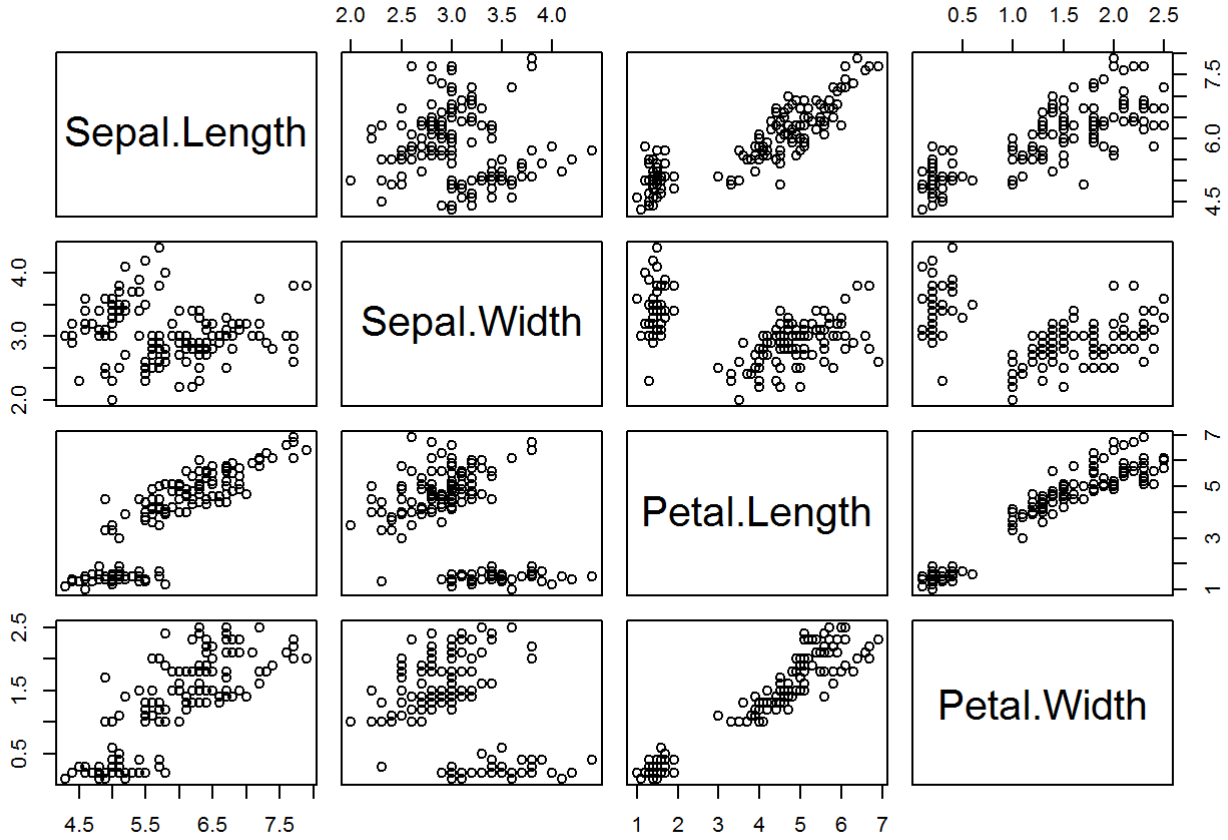
Syntax: `pairs(~x+y+..., data=our_data, additional arguments)`

Till now we were working with `plot()` command which gives us the scatter plot for two variables. If we want to get the scatter plot for more than two variables and for each pair, we may use the `pairs()` command.

```
rm(iris) # Remove to reload iris
View(iris) # View the output on a separate window
#pairs(~iris[1]+iris[2], iris) # Data should be Numeric
attach(iris)
pairs(~as.numeric(Sepal.Length)+as.numeric(Petal.Length), iris)
```




```
detach(iris)
pairs(iris[-5])
```



Pairs plot provide us the matrix of scatter plots. It plots all the pairwise combinations for the mentioned columns of the data.

14.1.1 Additional Arguments

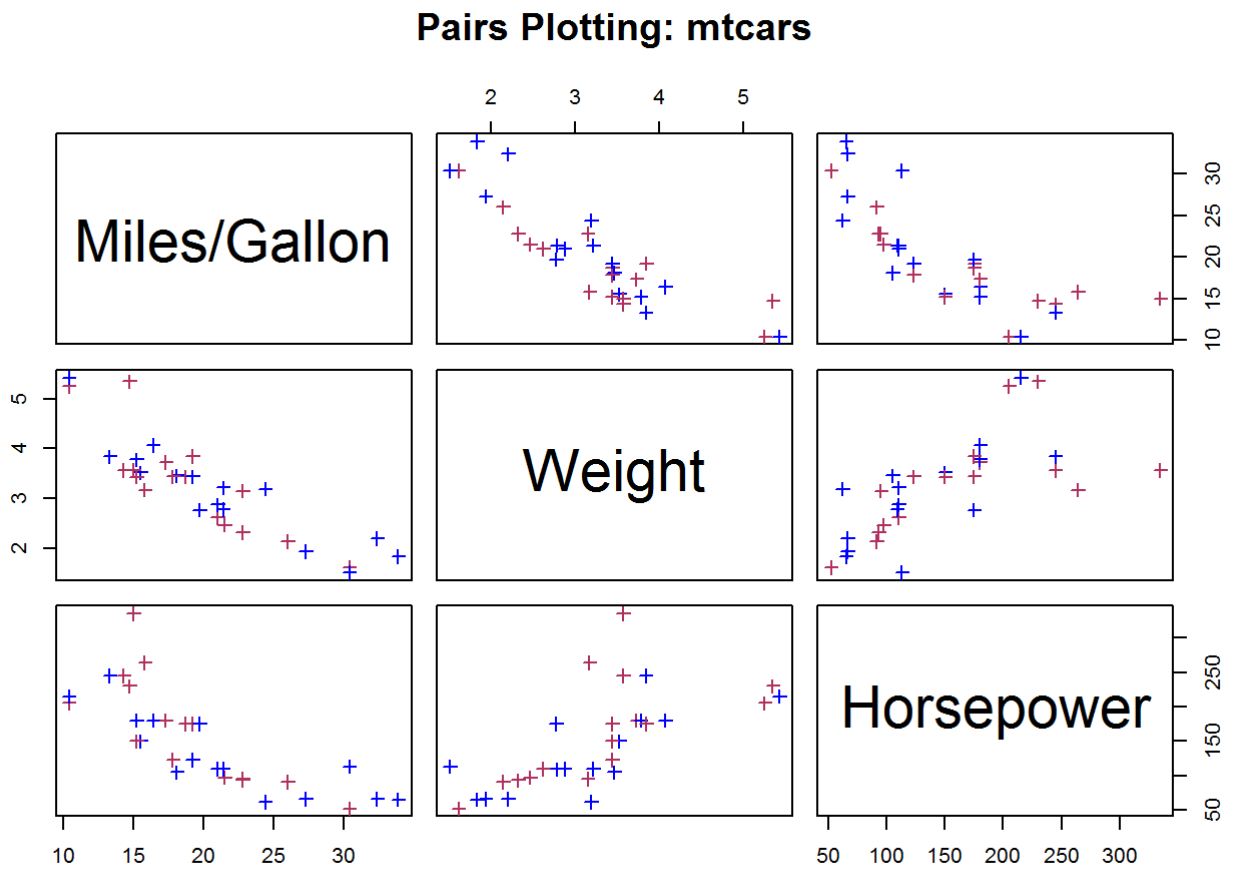
14.1.1.1 Similar to Scatter Plot commands

- `main`
- `sub`
- `col`
- `pch`
- `cex`

14.1.1.2 New Arguments

- `font.labels`
- `labels`
- `cex.labels`
- `font.labels`

```
View(mtcars)

# Make a scatter plot for each pair of mpg, wt and hp
attach(mtcars)
pairs(~mpg+wt+hp, mtcars,
      main="Pairs Plotting: mtcars",
      col=c("maroon", "blue"),
      labels=c("Miles/Gallon", "Weight", "Horsepower"),
      cex.labels=2.75,
      pch=3)
```




```
detach(mtcars)
```

14.2 Pie Chart

It displays proportional data through the use of `pie()` command.

Syntax: `pie(data, ...)`

% Area = $\frac{x_i}{\sum x} \cdot 100 \ \forall i$

14.2.1 Additional Arguments

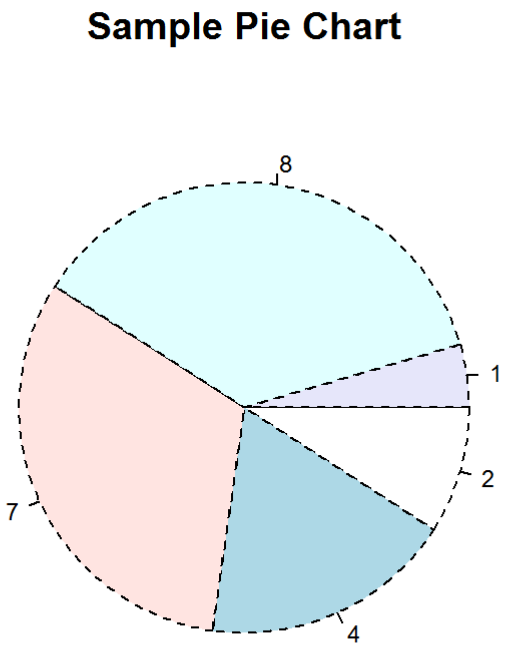
- `labels`
- `col`
- `main`
- `clockwise` (default is FALSE)

- `init.angle`

NOTE:

Orientation	Angle (in degrees)
Clockwise	0
Counter-clockwise	90

```
x = c(2, 4, 7, 8, 1) # Numeric Vector
pie(x, labels=x,
    main="Sample Pie Chart",
    clockwise=T,
    init.angle=0,
    cex=0.75, lty=2)
```



14.3 Data Structure Conversion

Use `as.<datatype>(<data>)`

14.4 Box-Whisker Plot

This type of plot is useful when we want to compare multiple samples. It shows the 5-number summary in a compact manner.

Syntax: `boxplot(data, ...)`

14.4.1 Additional Arguments for `boxplot()`

- `range=0` to extend the whisker to the minimum and maximum values.
- `horizontal=F` to get horizontal box plot
- `title()` to set labels; can be used outside boxplot() (xlab, ylab, main)
- `col`

```
# Vector
x = round(runif(10, 1, 100)); x
```

```
##  [1] 78 60 21  9 47 73 89 26 35 62
```

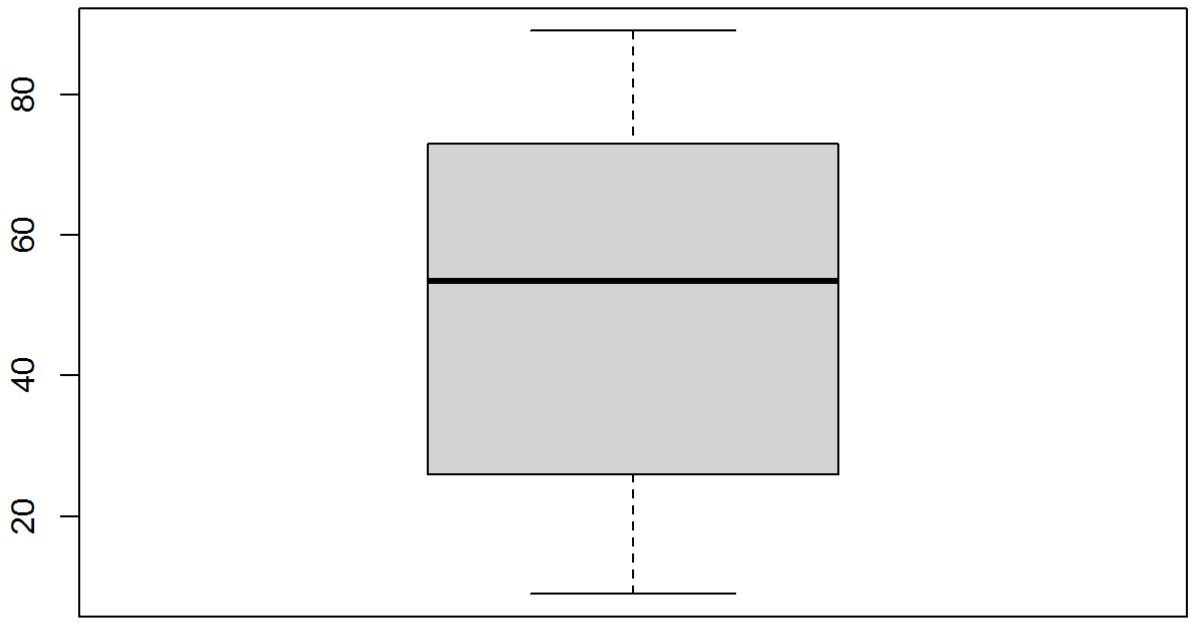
```
summary(x) # 6 number summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   28.25   53.50   50.00   70.25   89.00
```

```
fivenum(x) # 5 number summary
```

```
## [1]  9.0 26.0 53.5 73.0 89.0
```

```
boxplot(x)
```



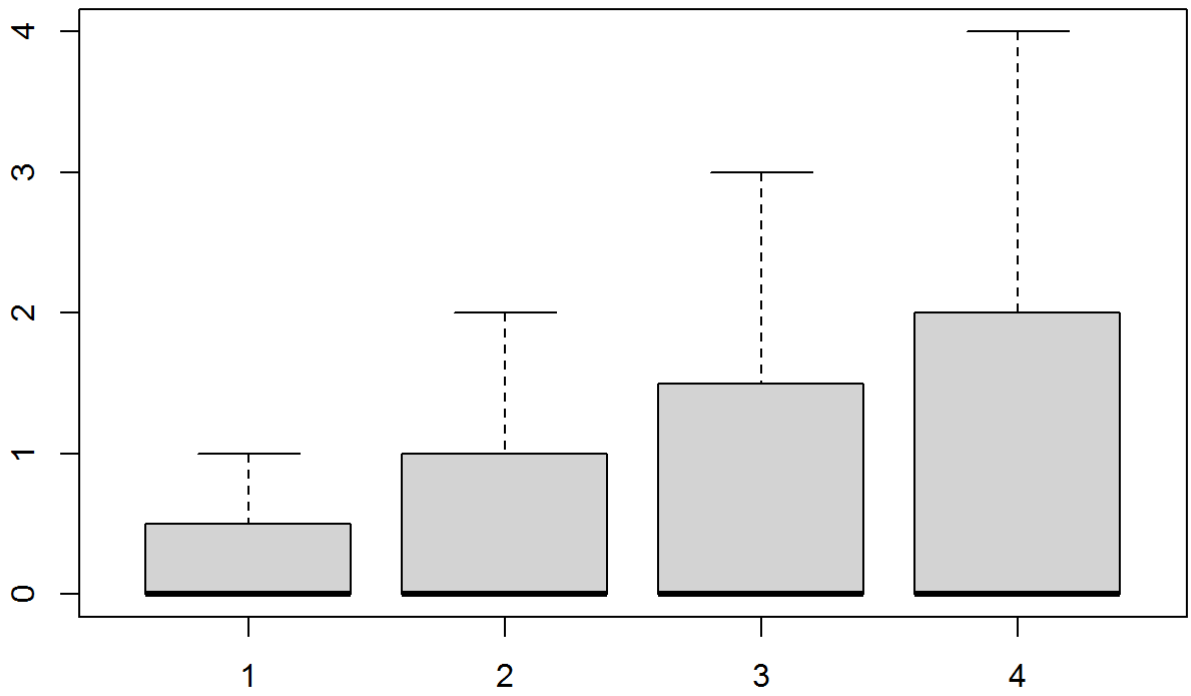
```
# Matrix
d = diag(1:4); d
```

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

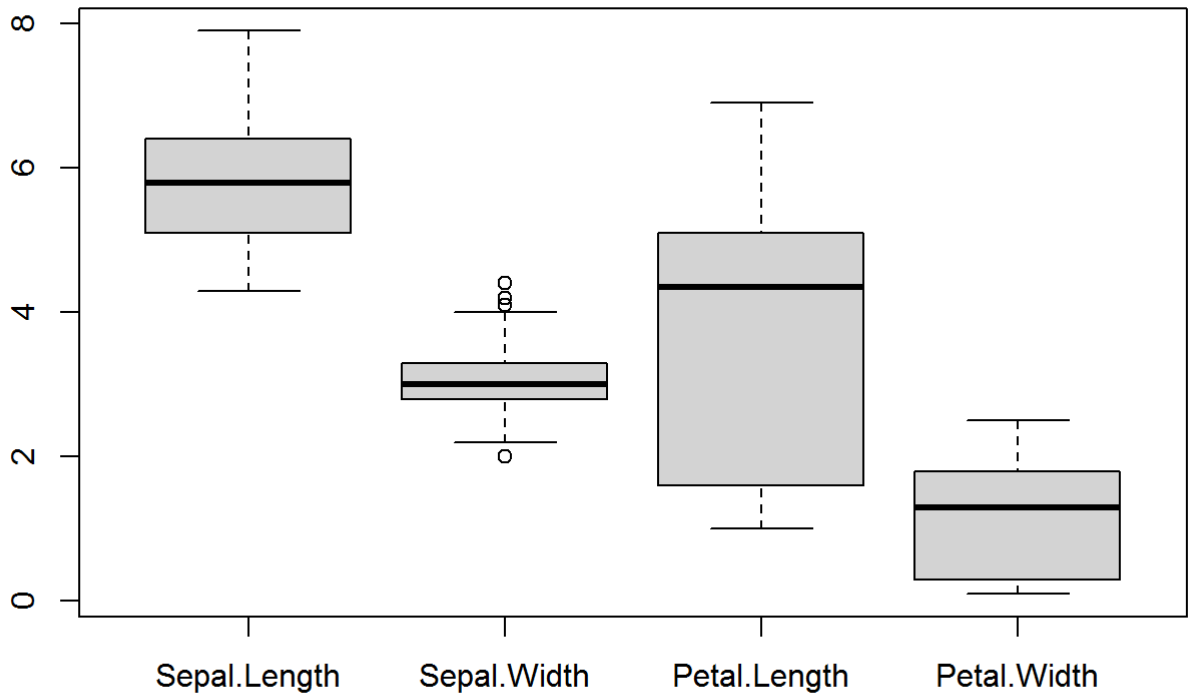
```
summary(d)
```

```
##        V1              V2              V3              V4
##  Min.   :0.00   Min.   :0.0   Min.   :0.00   Min.   :0
##  1st Qu.:0.00   1st Qu.:0.0   1st Qu.:0.00   1st Qu.:0
##  Median :0.00   Median :0.0   Median :0.00   Median :0
##  Mean   :0.25   Mean   :0.5   Mean   :0.75   Mean   :1
##  3rd Qu.:0.25   3rd Qu.:0.5   3rd Qu.:0.75   3rd Qu.:1
##  Max.   :1.00   Max.   :2.0   Max.   :3.00   Max.   :4
```

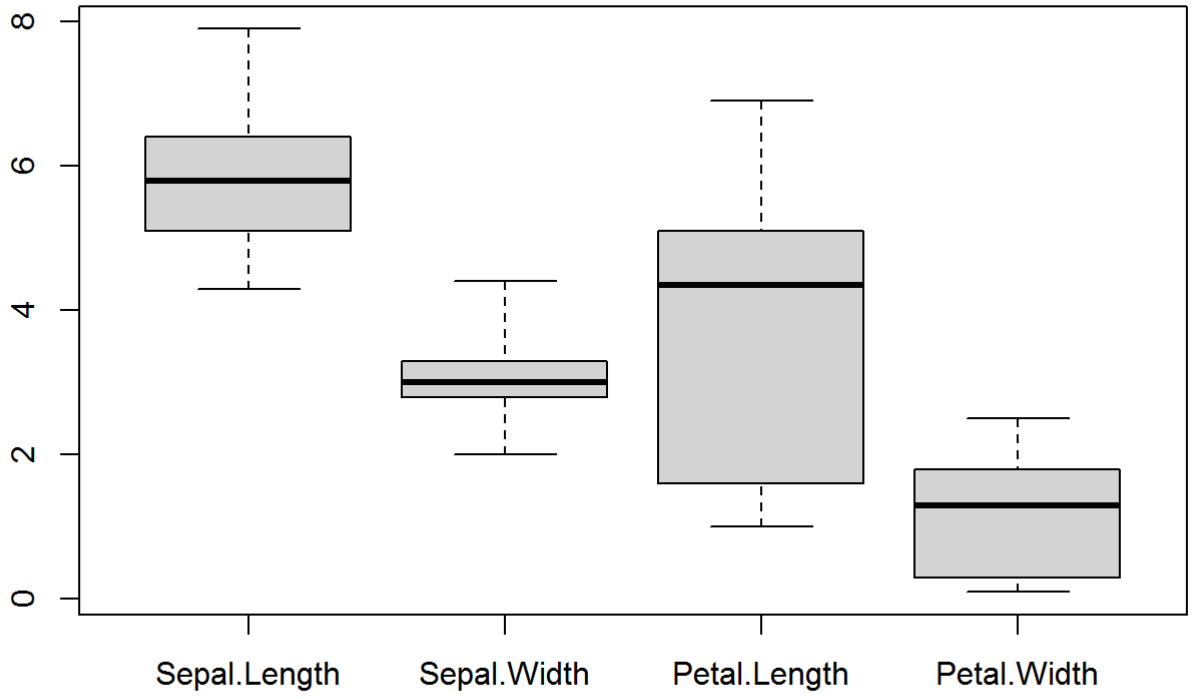
```
boxplot(d)
```



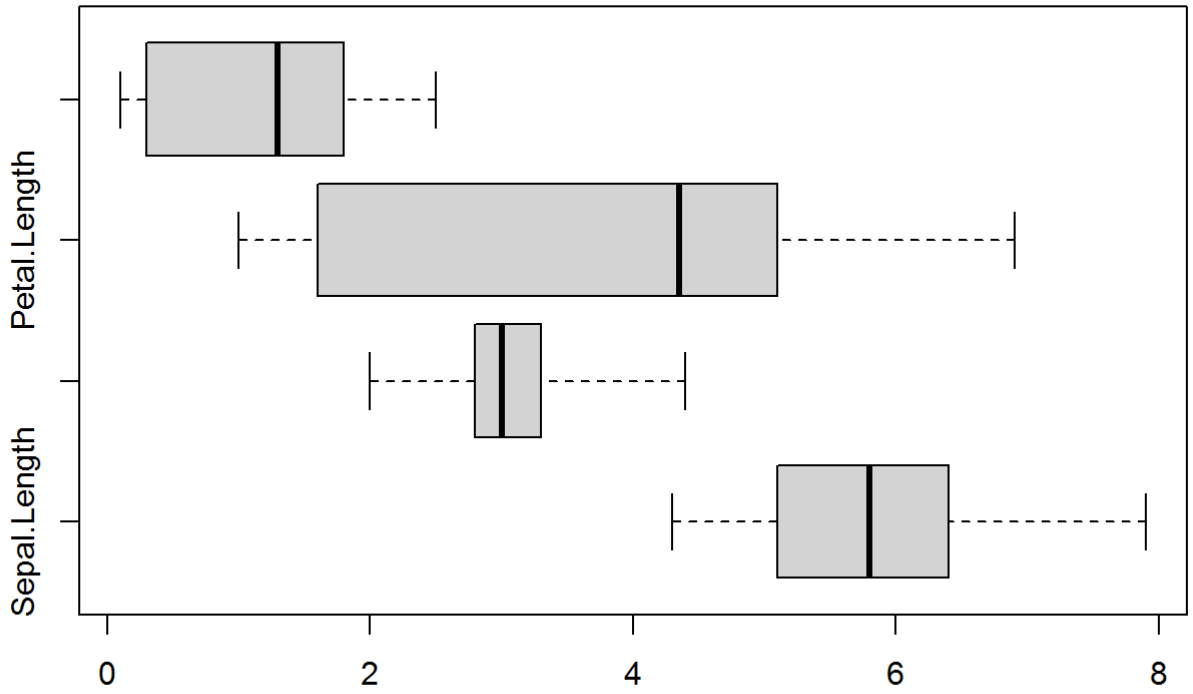

```
# Data Frame
i1 = iris[-5]
boxplot(i1) # Outliers
```



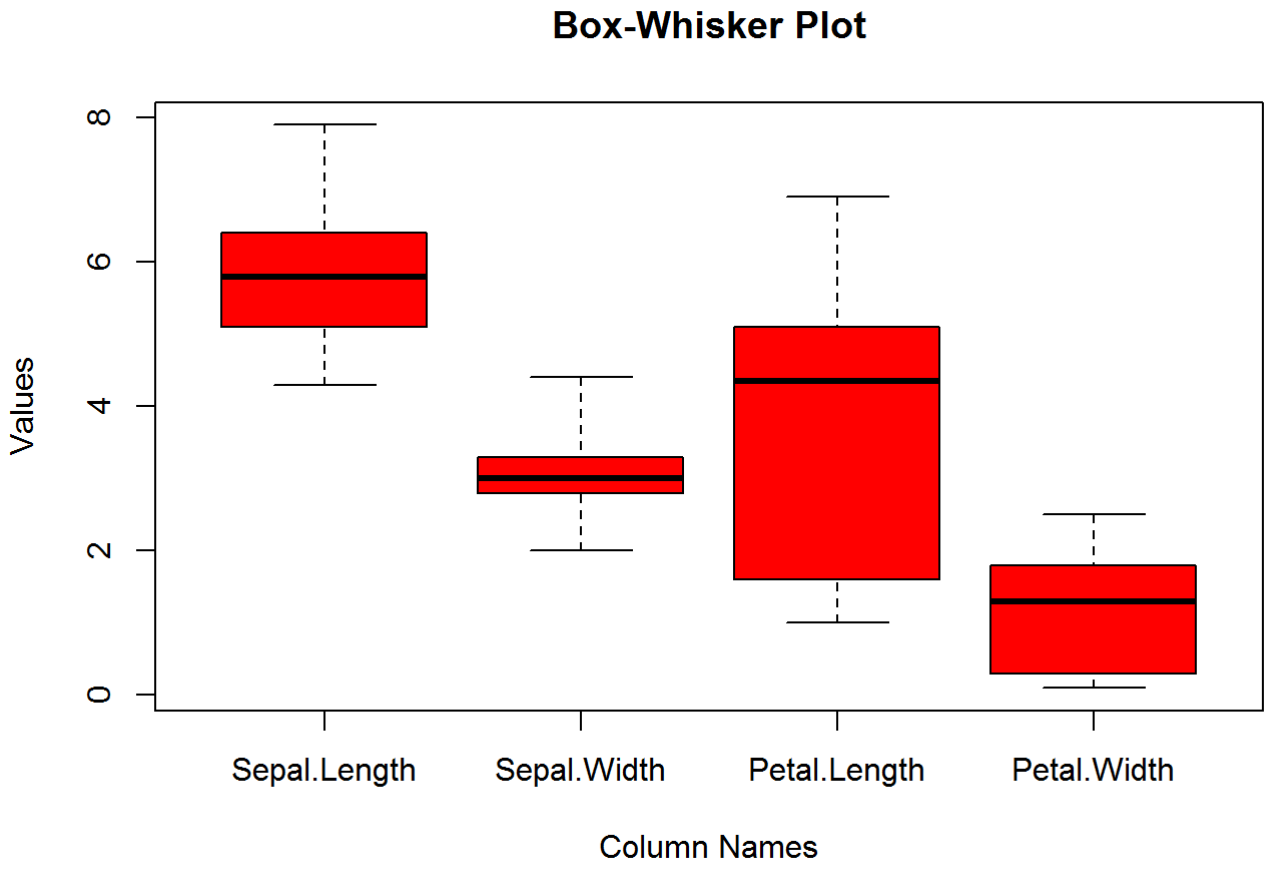

```
boxplot(i1, range=0) # No Outliers
```



```
boxplot(i1, range=0, horizontal=T) # No Outliers; Horizontal Box-plot (some row names are omm
        itted)
```




```
boxplot(i1, range=0, col="red")
title(ylab = "Values", xlab="Column Names", main="Box-Whisker Plot")
```



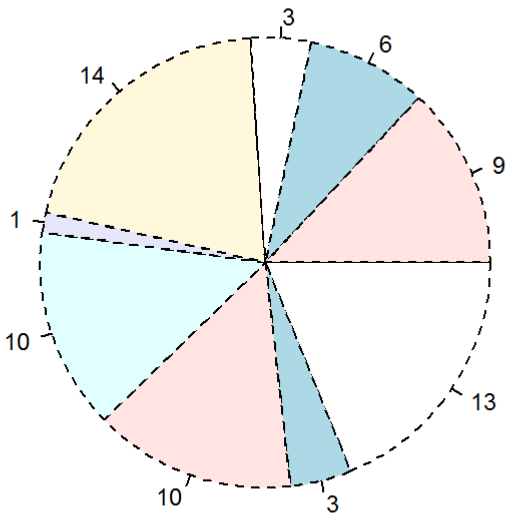
14.5 Class Exercises

a. Define a 3x3 Matrix in `A` and plot the pie chart

```
A = matrix(round(runif(9, 1, 16)), 3); A
```

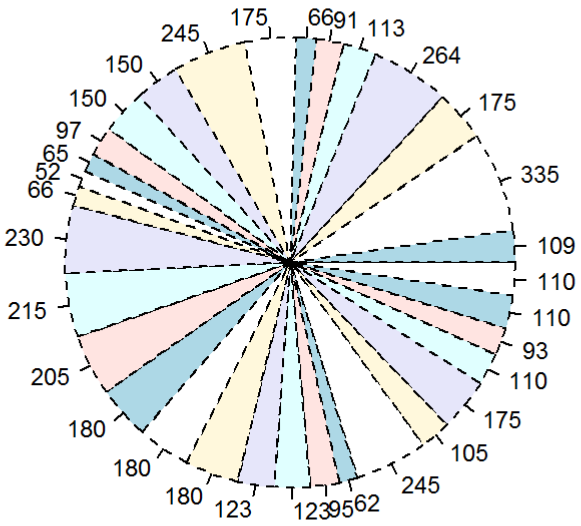
```
##      [,1] [,2] [,3]
## [1,]   13   10    3
## [2,]    3    1    6
## [3,]   10   14    9
```

```
pie(A, labels=A,
    main="Pie Chart from a 3x3 Matrix",
    clockwise=T,
    init.angle=0,
    cex=0.75, lty=2) # Plots column-wise
```



b. Take a data frame and plot a pie chart

```
# With attach() and detach()
attach(mtcars)
pie(hp, labels=hp,
    main="HP from mtcars",
    clockwise=T,
    init.angle=0,
    cex=0.75, lty=2)
```




```
detach(mtcars)

# Without attach() and detach()
s <- iris[1:10, 1:4]; s; class(s)
```

```
##    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
```

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

```
class(s[1])
```

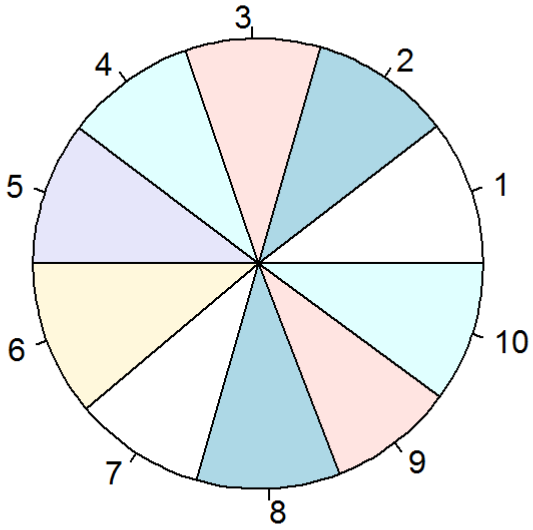
```
## [1] "data.frame"
```

```
d <- as.matrix(s[1]); d; class(d)
```

```
##    Sepal.Length
## 1           5.1
## 2           4.9
## 3           4.7
## 4           4.6
## 5           5.0
## 6           5.4
## 7           4.6
## 8           5.0
## 9           4.4
## 10          4.9
```

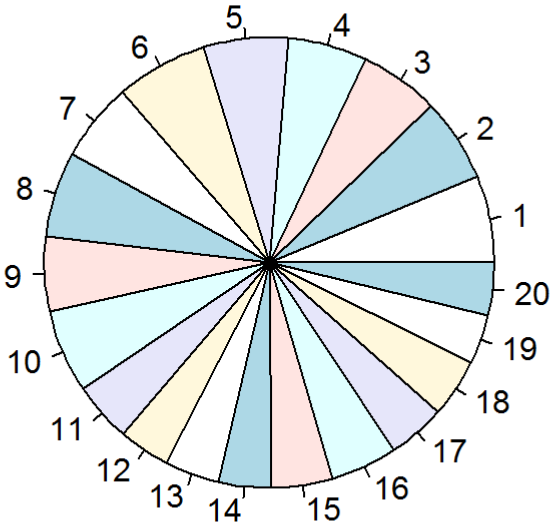
```
## [1] "matrix" "array"
```

```
pie(d)
```



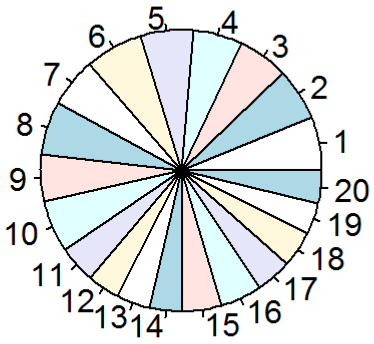
c. Make a pie chart for any two columns of a data frame

```
s <- iris[1:10, 1:4]
v1 <- as.matrix(s[1])
v2 <- as.matrix(s[2])
pie(c(v1, v2))
```



d. Explore how to change the radius of the pie chart

```
s <- iris[1:10, 1:4]
v1 <- as.matrix(s[1])
v2 <- as.matrix(s[2])
pie(c(v1, v2), radius=0.5)
```

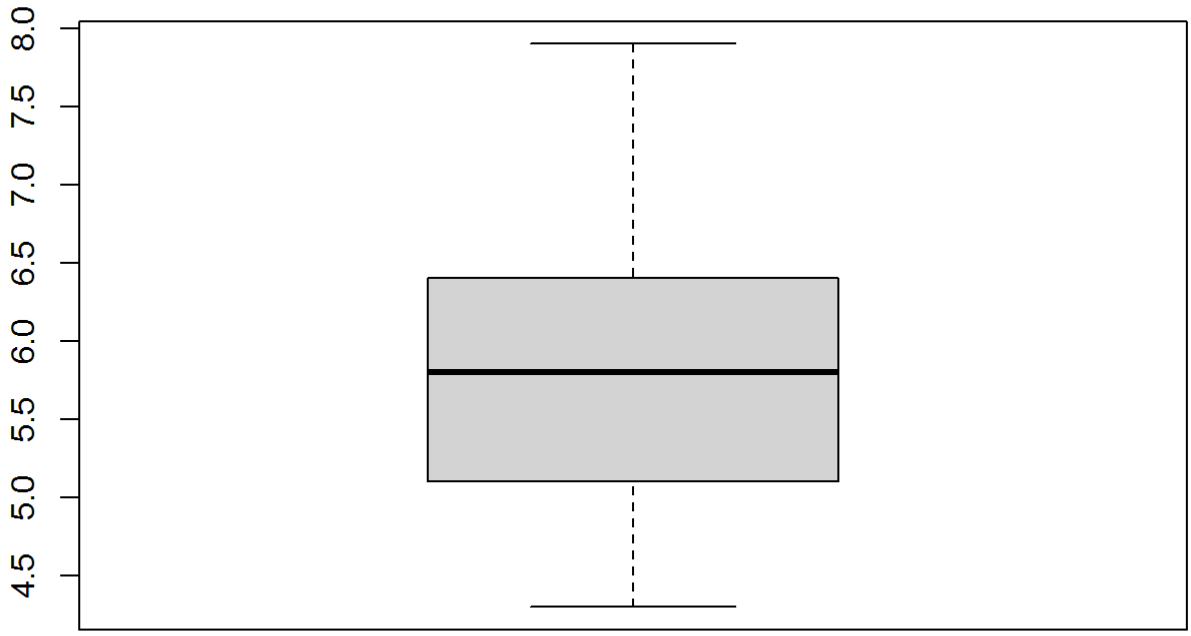


e. Create boxplots for the first column of each type of species in data frame `iris`

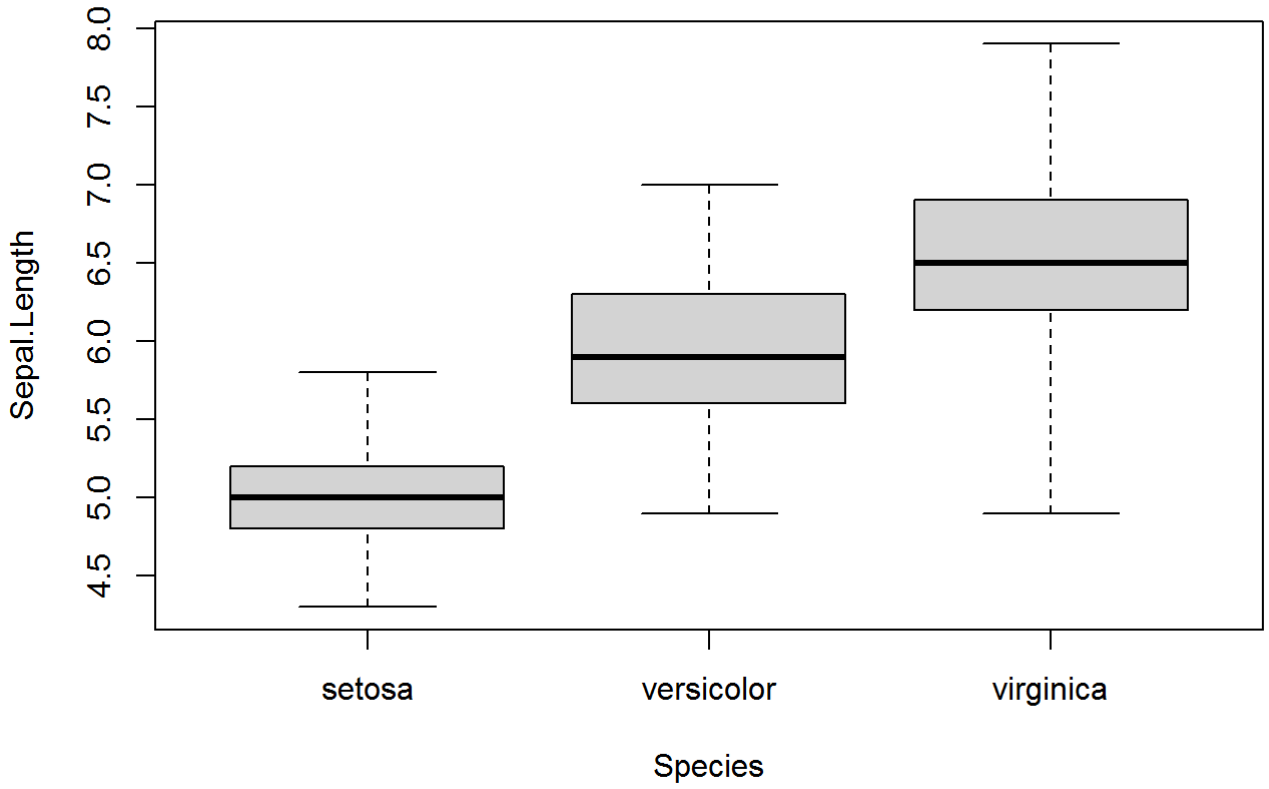
Syntax: `boxplot(response~predictor, data=dataname)`

From `dataname` divide the entries of `response` in terms of `predictor`.

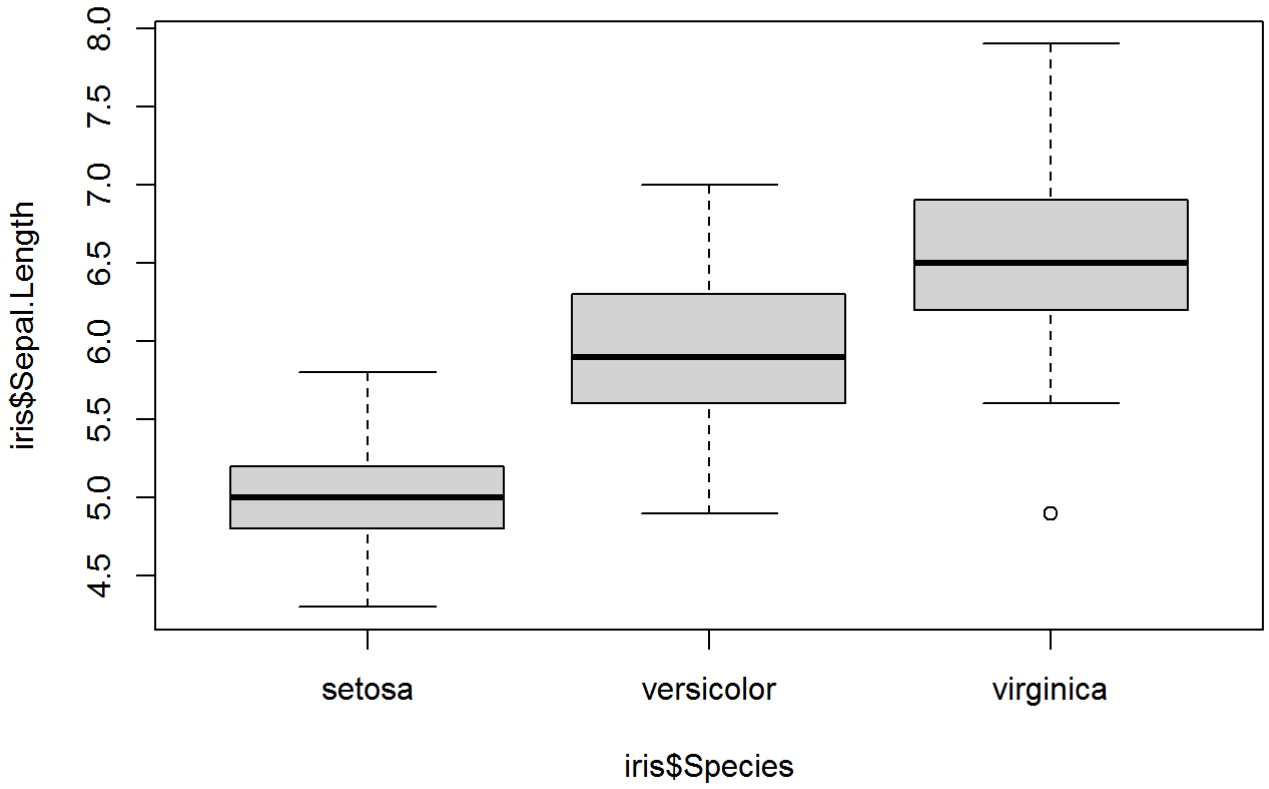
```
# 1st Method
View(iris)
attach(iris) # Only for directly accessing columns of a data frame
boxplot(Sepal.Length) # Boxplot for entire first col
```



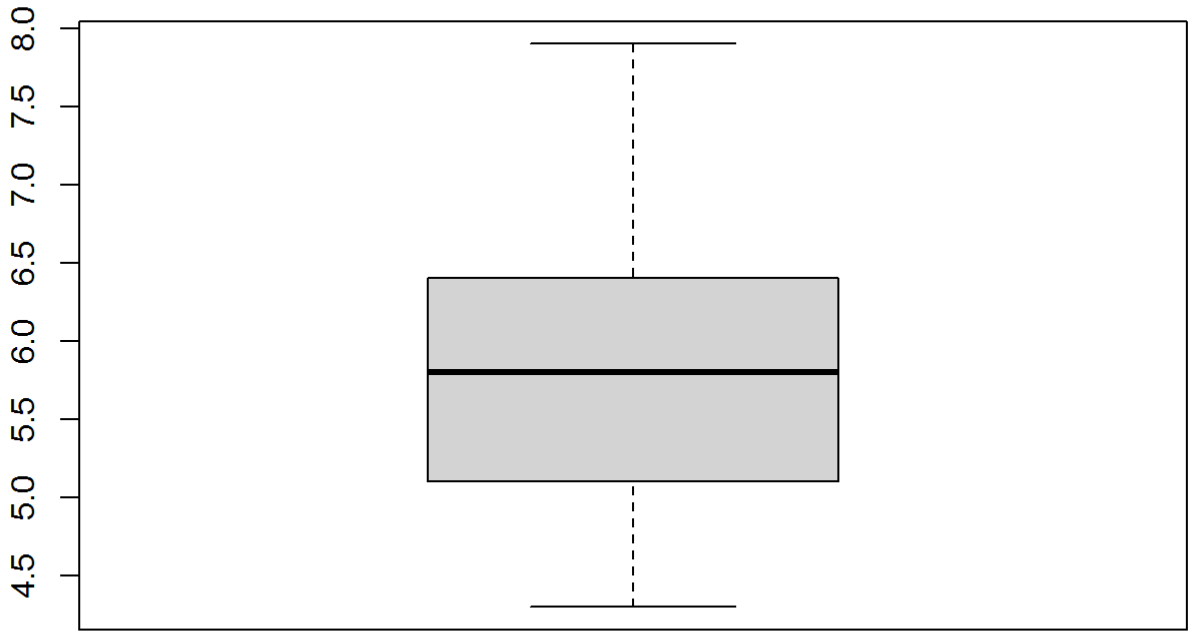

```
boxplot(Sepal.Length~Species, data=iris, range=0) # Boxplot for entire first col but specific
        species
detach(iris)

# 2nd Method
boxplot(Sepal.Length~Species, data=iris, range=0)
```




```
# 3rd Method
boxplot(iris$Sepal.Length~iris$Species)
```




```
# 4th Method (***)
# Syntax: with(expression, data=dataname, ...)
# This will help us plot anything (boxplot, histogram, pie, etc.)
with(boxplot(Sepal.Length), data=iris)
```



15 Analysis - III

#@ Stem Leaf Plot, Histograms and Line Charts

```
data1 = c(3, 3, 5, 2, 2, 6, 4 , 9, 8, 3, 7, 2)
table(data1)  # occurrences of each element
```

```
## data1
## 2 3 4 5 6 7 8 9
## 3 3 1 1 1 1 1 1
```

```
stem(data1)
```

```
##
##   The decimal point is at the |
##
##   2 | 000000
##   4 | 00
##   6 | 00
##   8 | 00
```

```
stem(data1, scale=2)  # scale parameter is used to increase the bin range
```

```
##
##   The decimal point is at the |
##
##   2 | 000
##   3 | 000
##   4 | 0
##   5 | 0
##   6 | 0
##   7 | 0
##   8 | 0
##   9 | 0
```

Q. On World Obesity Day, suppose in a school a teacher decided to measure the weight of any 10 students whom she feels might have obesity. Make the stem and leaf plot.

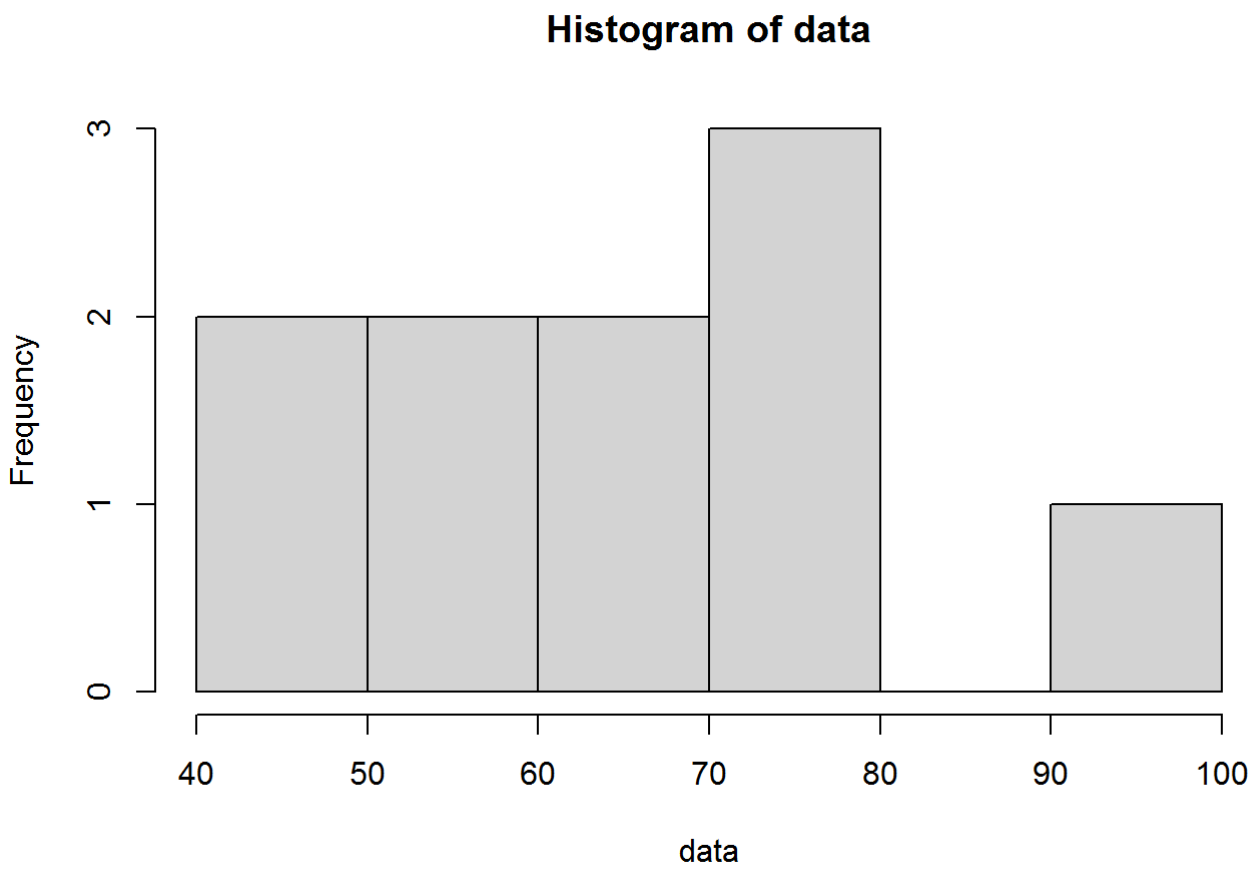
```
data = c(54, 43, 67, 76, 45, 59, 66, 78, 80, 92)
table(data)
```

```
## data
## 43 45 54 59 66 67 76 78 80 92
##  1  1  1  1  1  1  1  1  1  1
```

```
stem(data)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   4 | 35
##   5 | 49
##   6 | 67
##   7 | 68
##   8 | 0
##   9 | 2
```

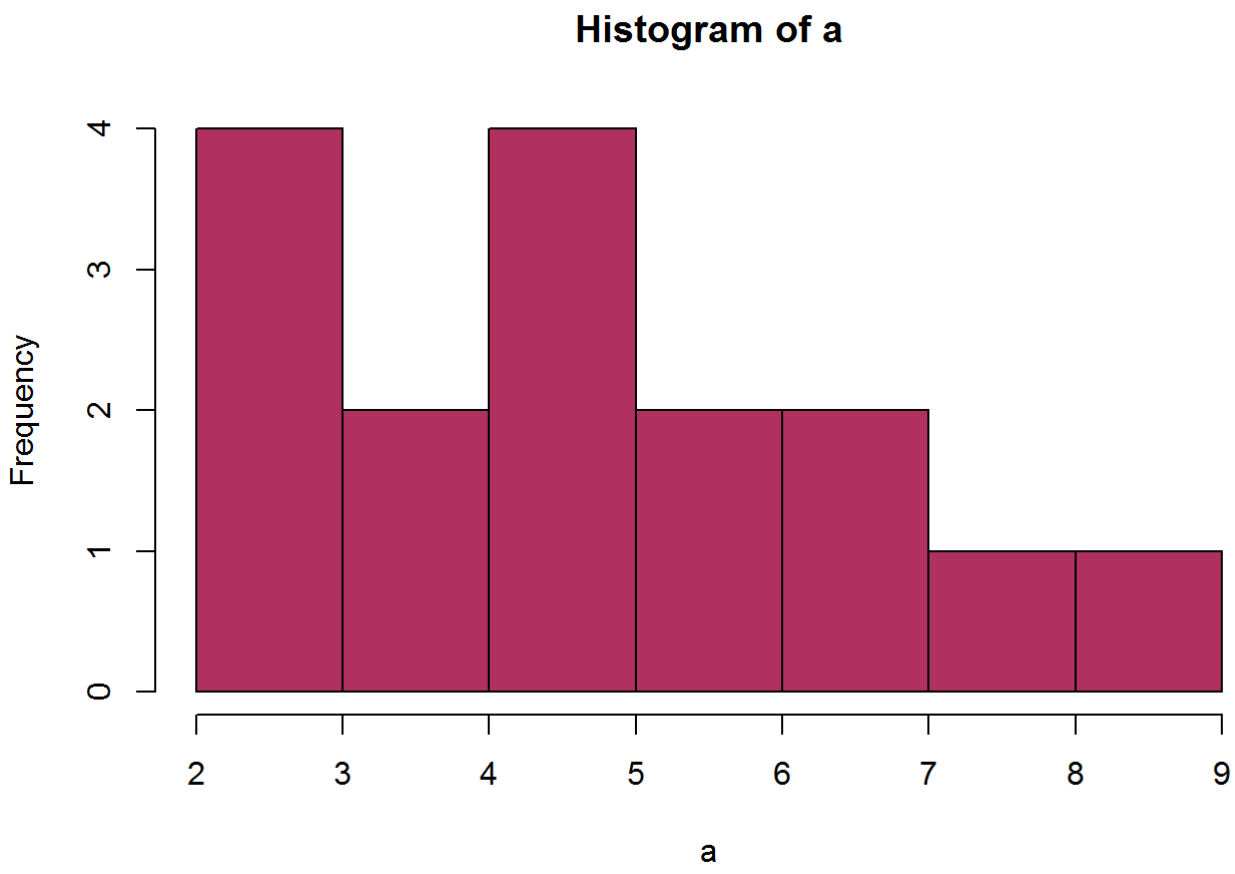
```
hist(data)
```



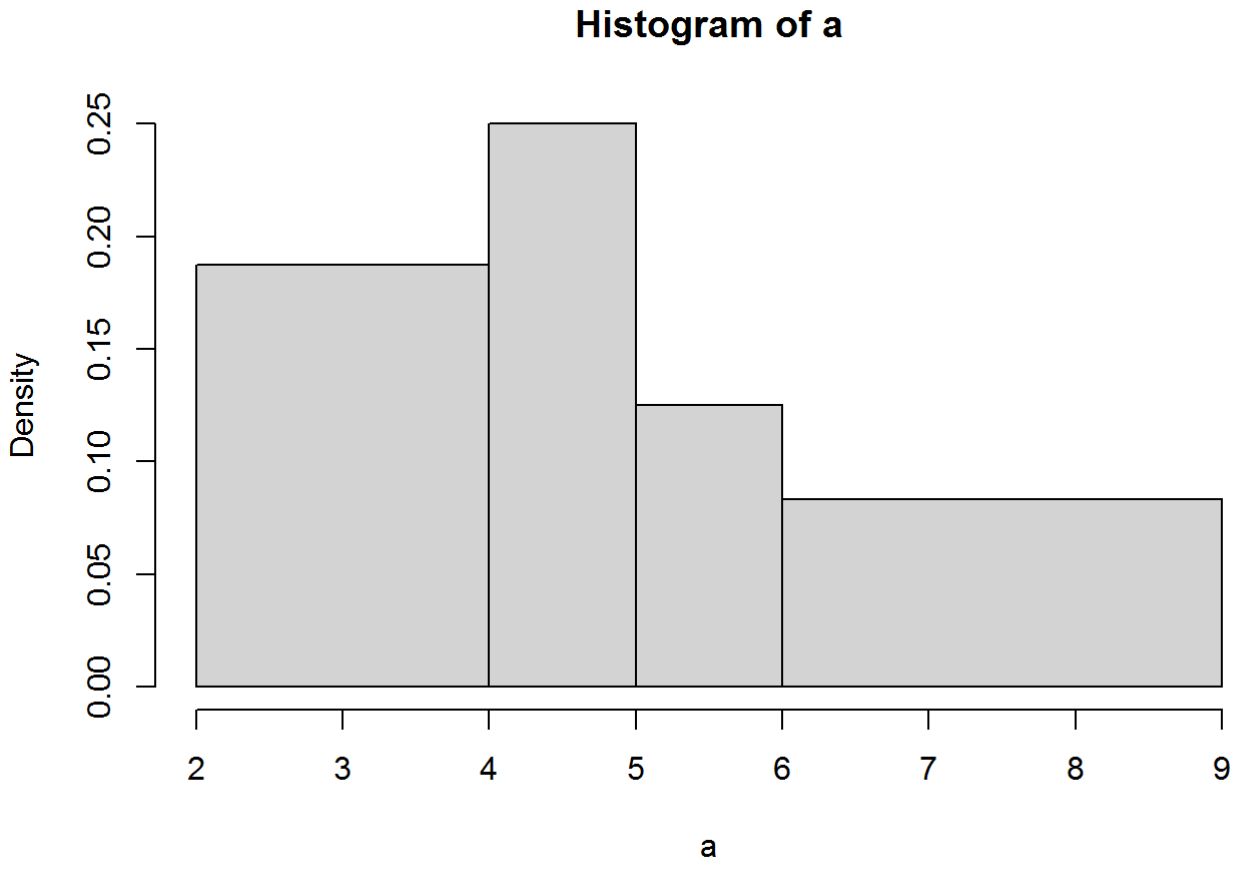

```
a = c(2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 6, 6, 7, 7, 8, 9)
table(a)
```

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

```
hist(a, breaks=2:9, col="maroon")
```

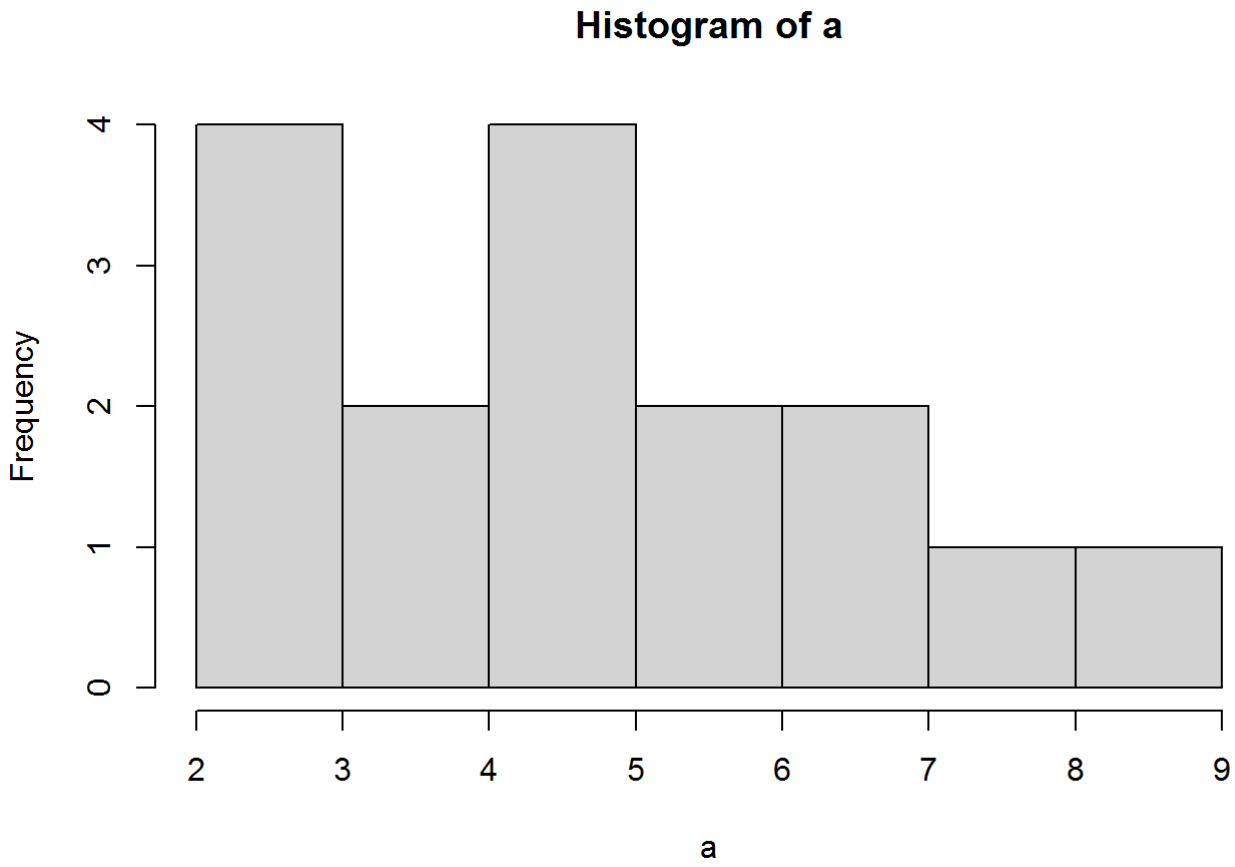



```
hist(a, breaks=c(2,4,5,6,9))
```

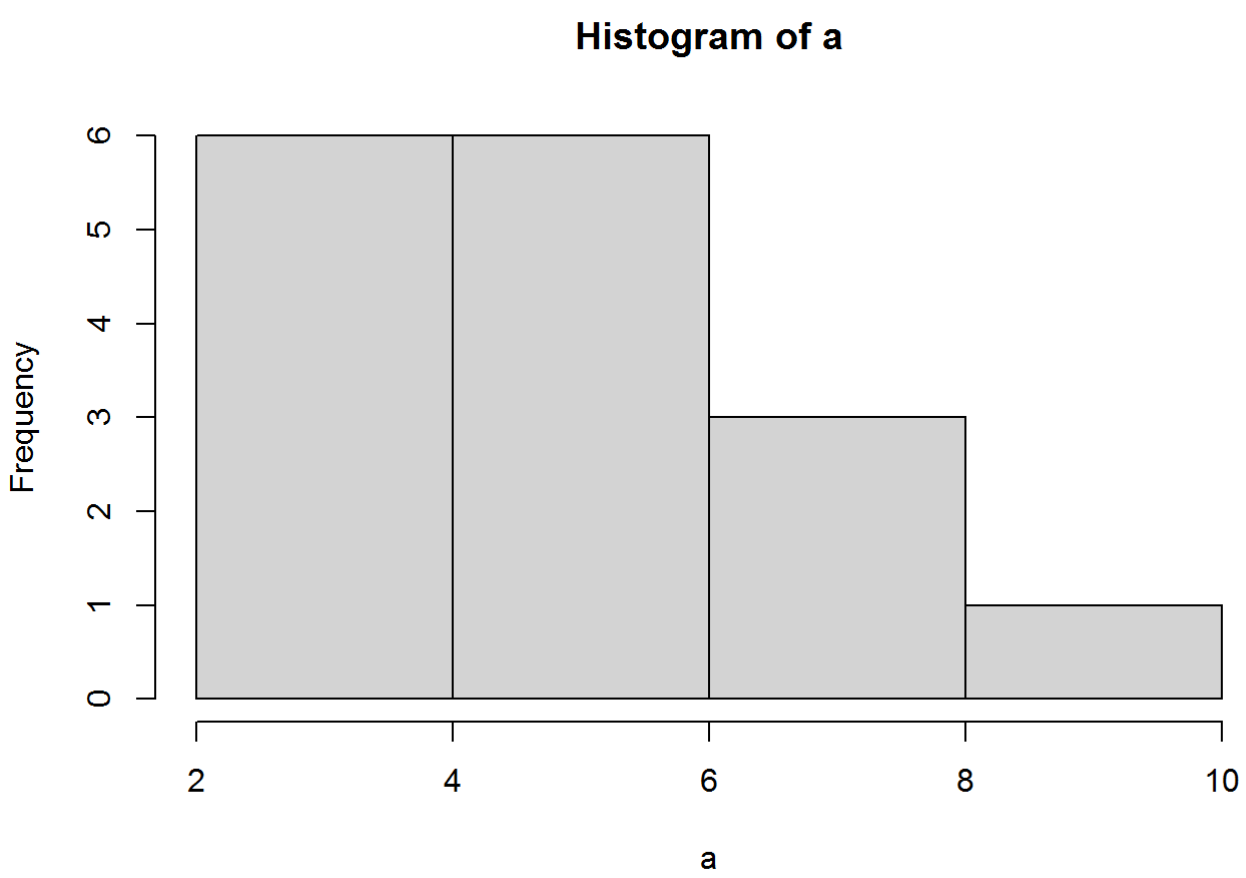


```
hist(a, breaks=c(2,9))
```


```
hist(a, breaks="st")
```



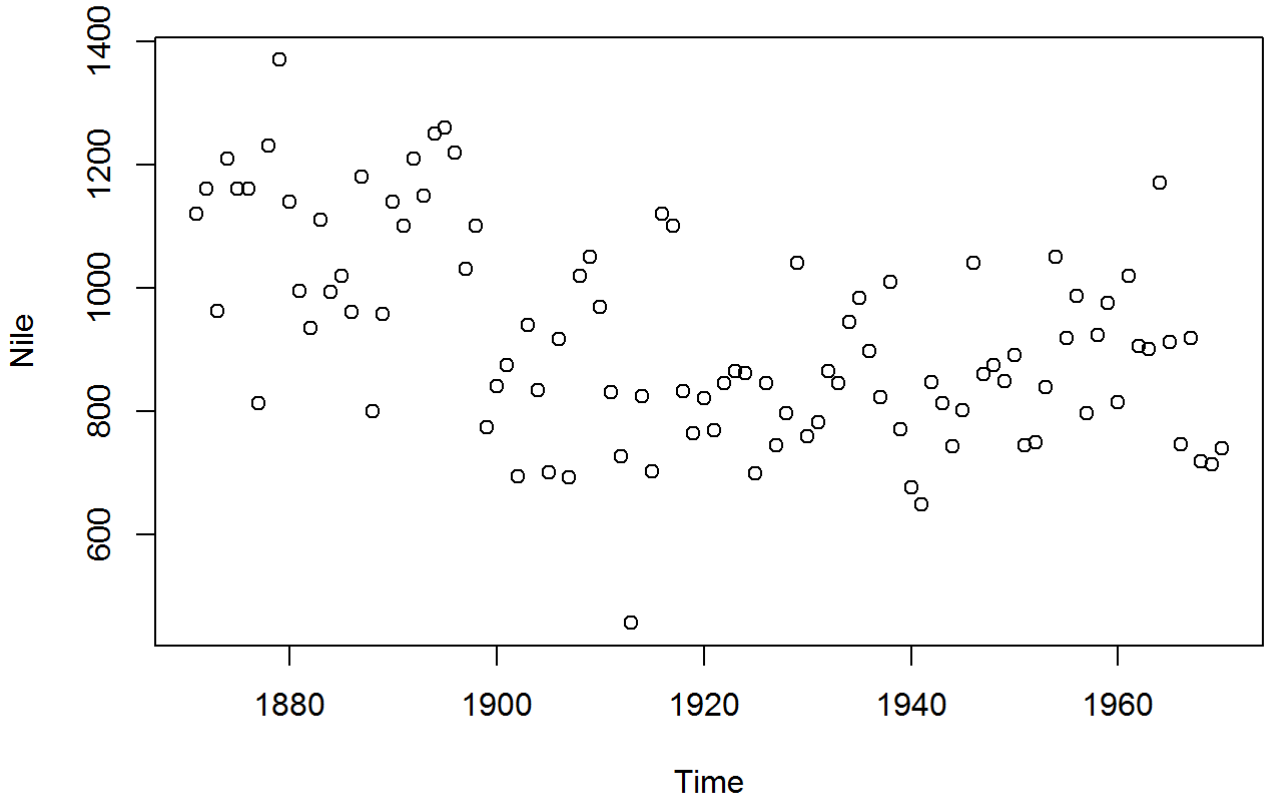

```
hist(a, breaks="sc")
hist(a, breaks="fr")
```



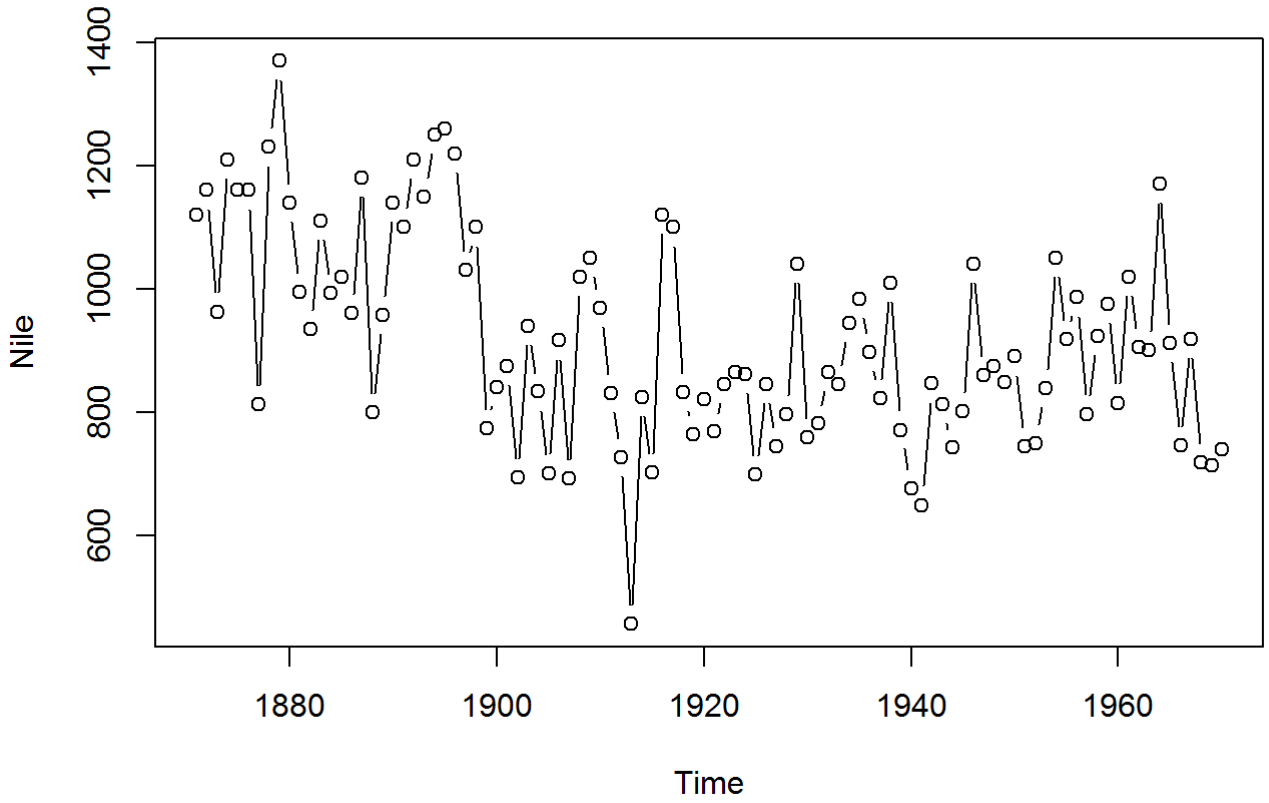

```
# frequency (freq) = T => Density
# xlim, ylim, col, main, xlab, ylab
```

#@ Line Chart

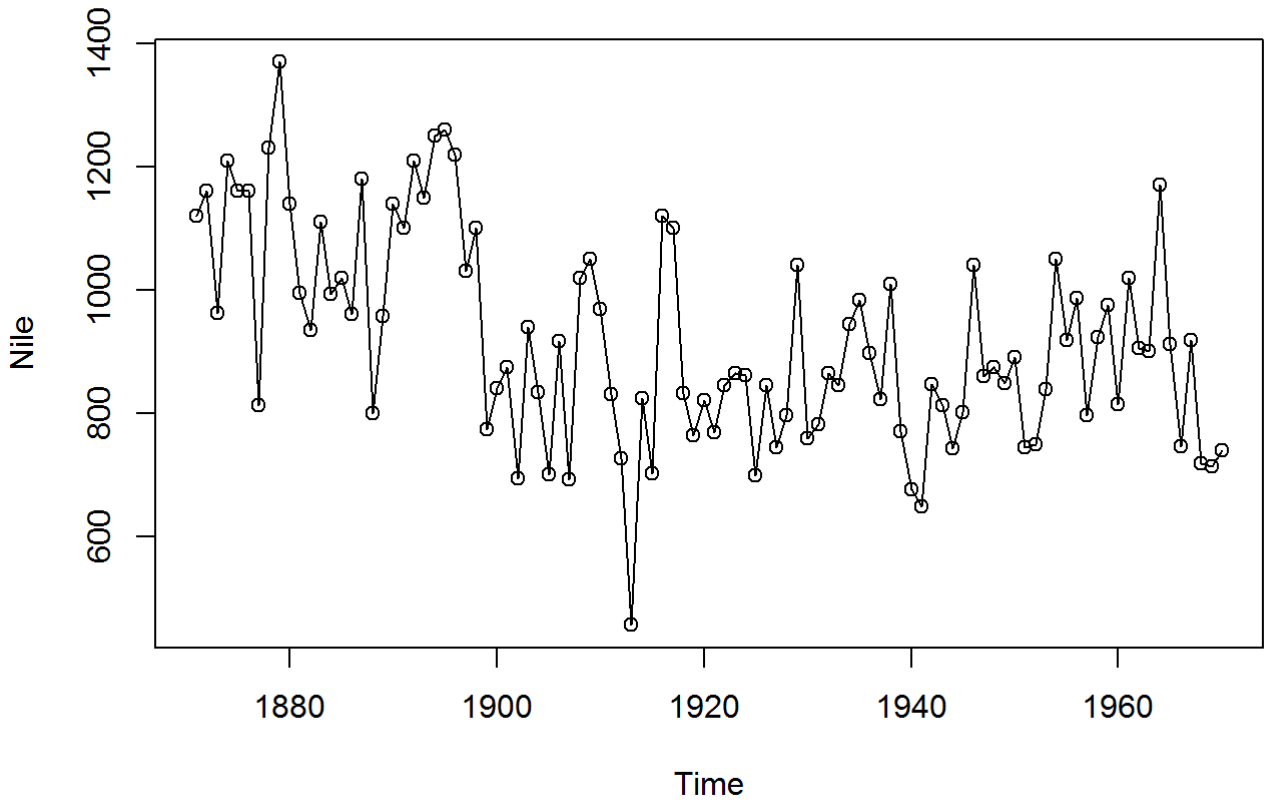
```
plot(Nile, type="p")
```



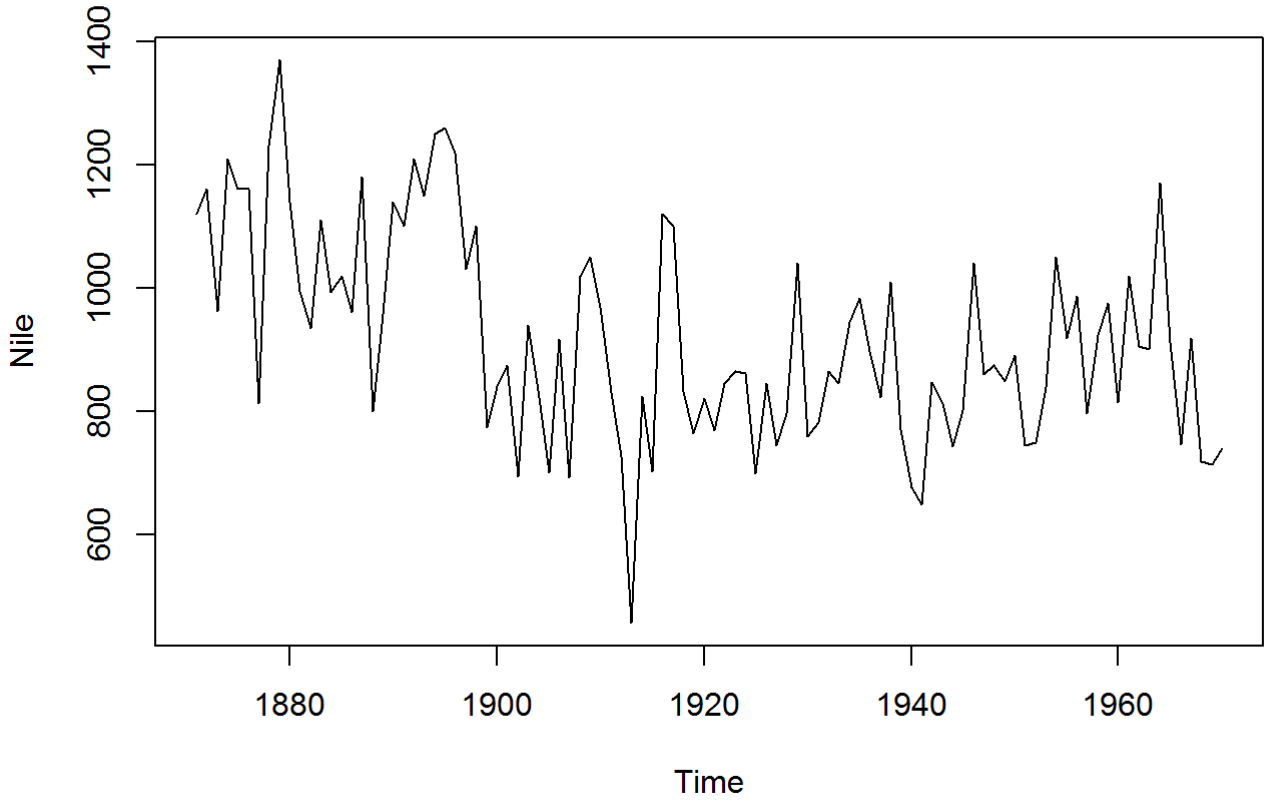

```
plot(Nile, type="b")
```



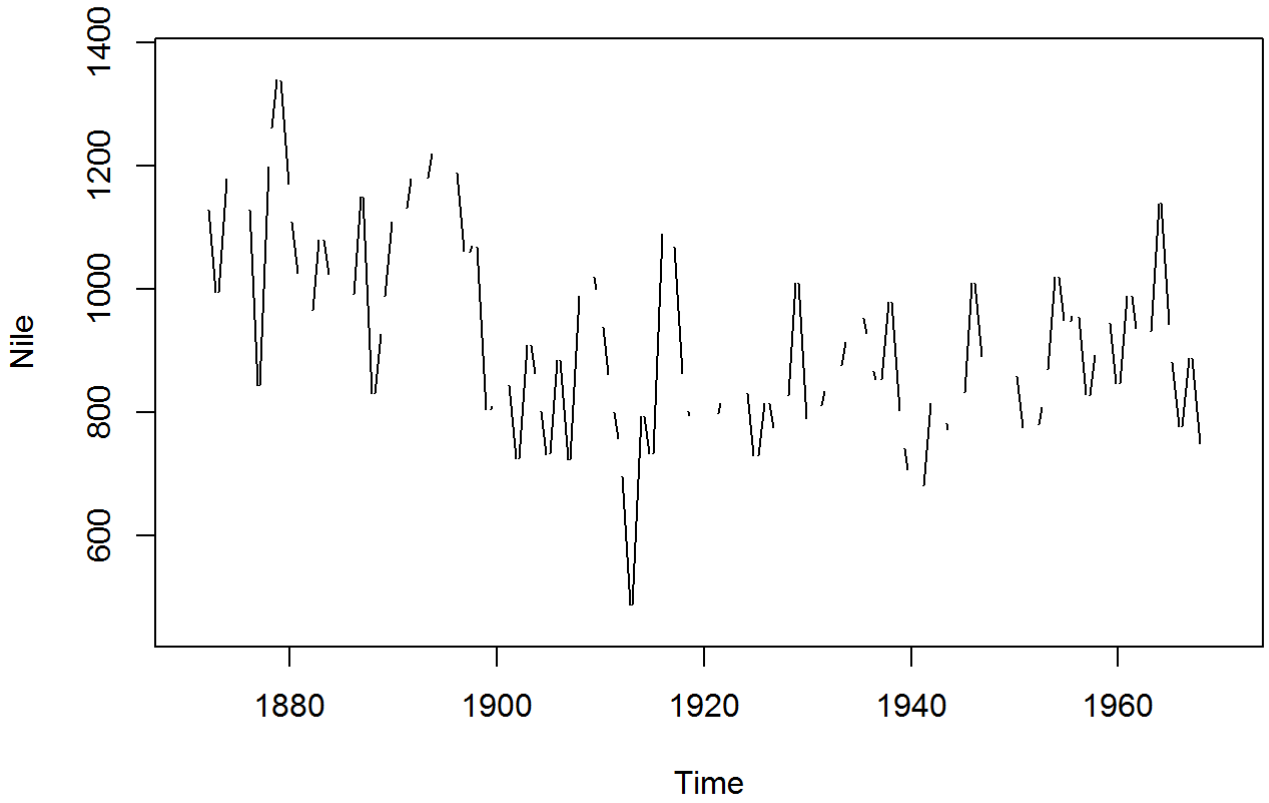
```
plot(Nile, type="o")
```



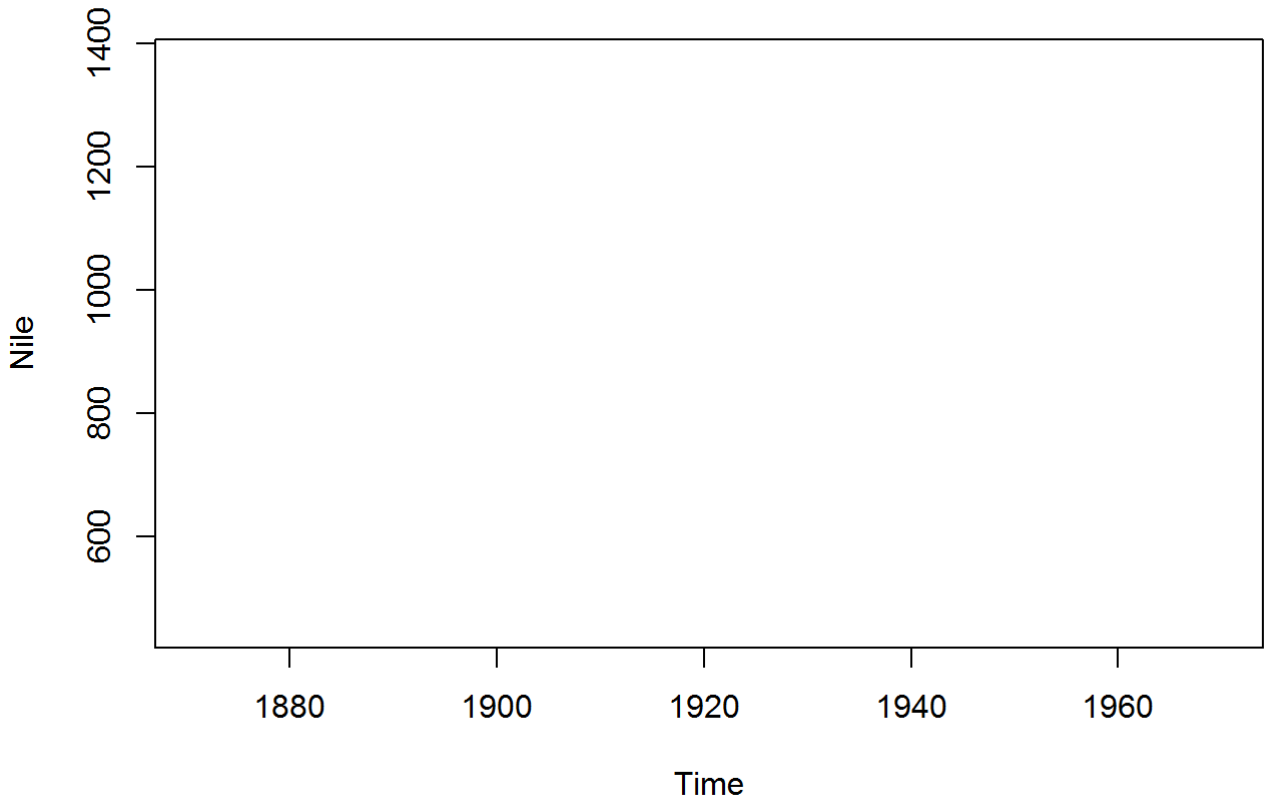

```
plot(Nile, type="l")
```




```
plot(Nile, type="c")
```



```
plot(Nile, type="n")
```



15.1 Histograms (in Detail)

Refer to Page 184 of Book PDF

15.2 Line Charts (in Detail)

Refer to Page 262 of Book PDF