R – BEGINNER

Digital Scholarship HUB

University of Illinois at Chicago

digitalscholarship@uic.edu

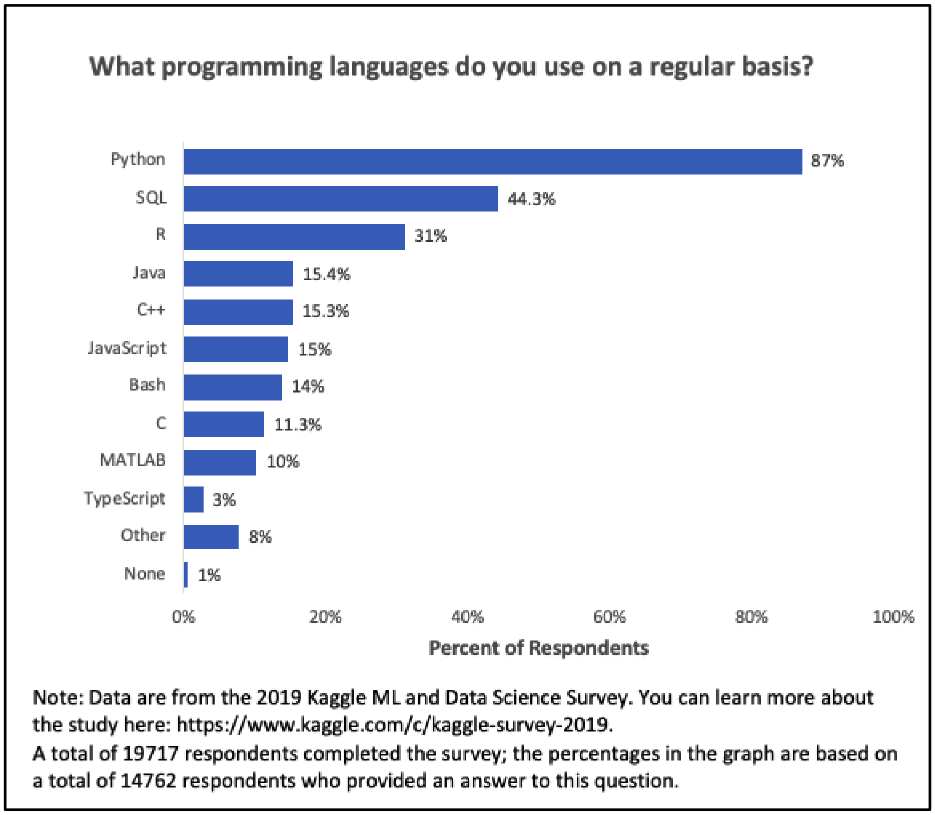
TABLE OF CONTENTS

R

R is a programming language used to perform statistical computations and implement graphics. It is open-source and free. It is generally used by statisticians for data mining, data analytics. It helps to perform data wrangling, analyzing and visualizing data easily.

R has a list of packages each of which helps you perform certain function. As of January 2017, there are more than 10,000 packages for R, the list of which you can find here. Each package can easily be downloaded and used in R.

In a 2019 survey conducted by Kaggle, R was the third most used programming language by data professionals.



What programming languages do you use on a regular basis?

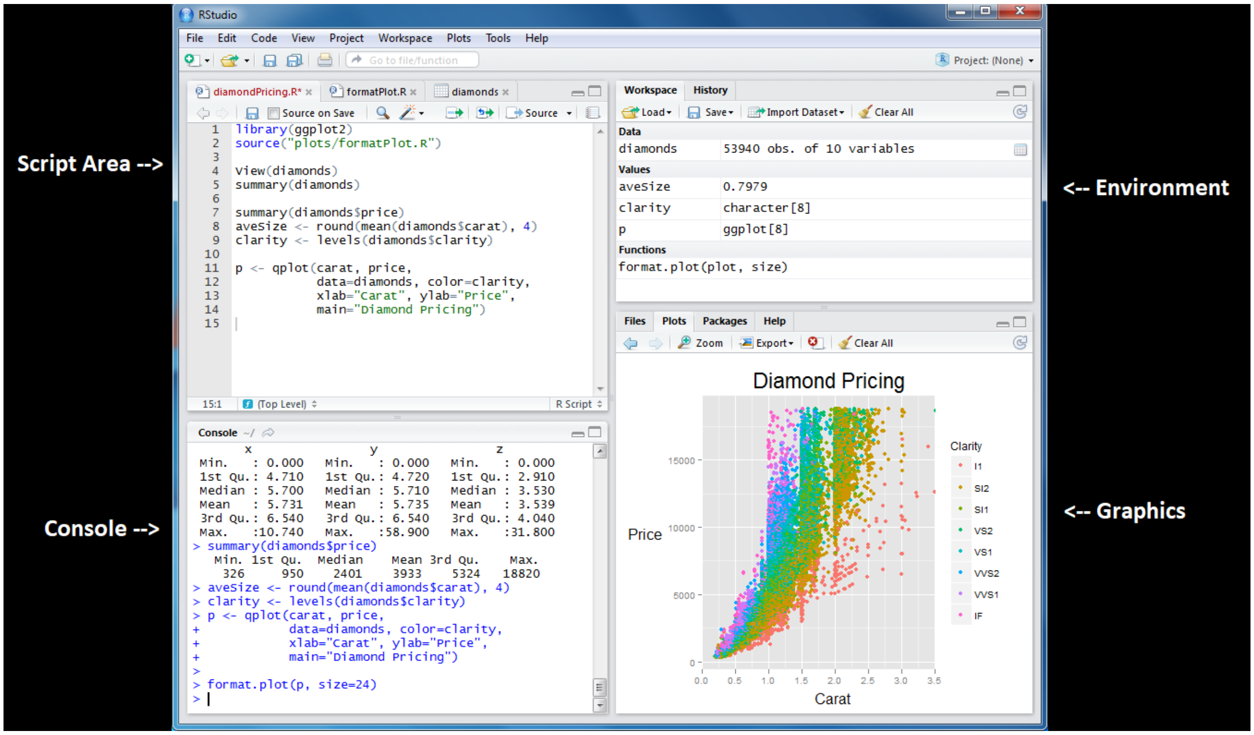
Note: Data are from the 2019 Kaggle ML and Data Science Survey. You can learn more about the study here: https://www.kaggle.com/c/kaggle-survey-2019.
A total of 19717 respondents completed the survey; the percentages in the graph are based on a total of 14762 respondents who provided an answer to this question.

Image source – Business Broadway

RStudio

RStudio is the integrated development environment (IDE) for R. It is available in two versions:

- RStudio Desktop - Regular desktop application
- RStudio Server - Runs on a remote server and accessed RStudio using a web browser



```
library(ggplot2)
source("plots/formatPlot.R")

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")

```

```
       x               y               z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
 Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> |
```

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979
clarity	character[8]
p	ggplot[8]
Functions	
format.plot(plot, size)	

Script Area – Write codes (or) scripts and run them separately

Console – Write and run the code together directly here

Environment – List of objects and variables created and present in the current session

Graphics – Displays the plots

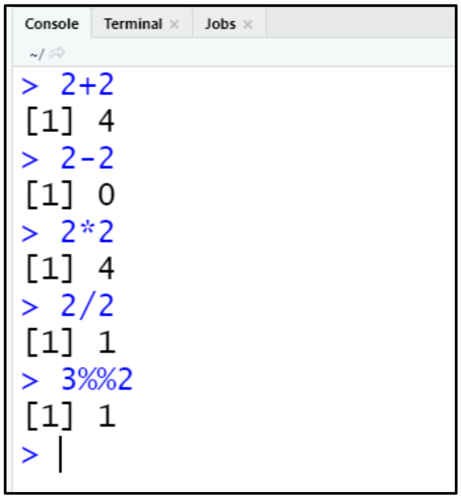
You can find the list of keyboard shortcuts for RStudio here.

Steps to install R & RStudio

- Windows – Link
- MAC - Link

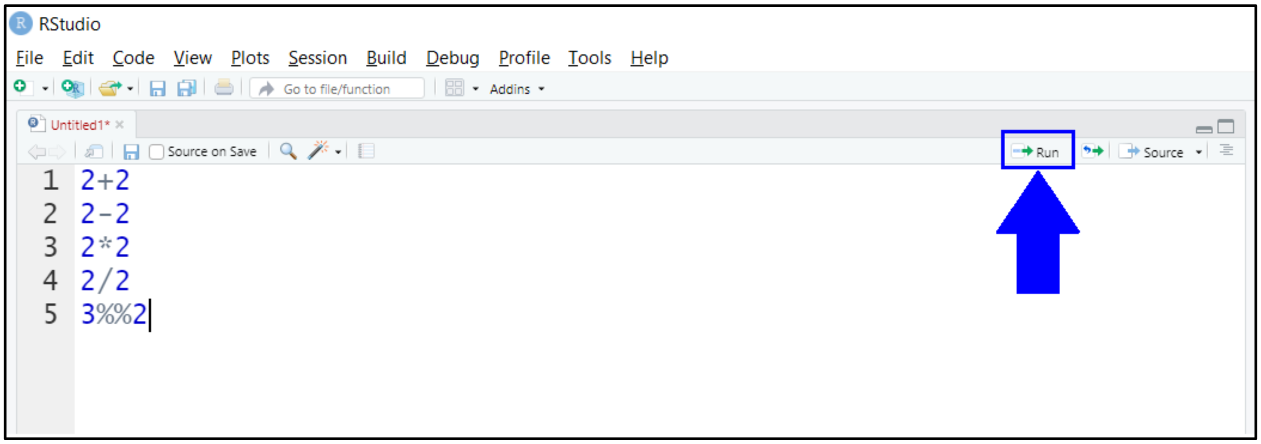
INTRODUCTION TO R

Codes can be directly run in the R console. Try running the below code to perform basic arithmetic operations of Addition (+), Subtraction (-), Multiplication (*), Division (/) and Modulo (%%) operation directly in the console.



```
> 2+2
[1] 4
> 2-2
[1] 0
> 2*2
[1] 4
> 2/2
[1] 1
> 3%%2
[1] 1
> 
```

Implementing the same code in the script area. If you do not see a file open in the script area select File → New File → R Script from the menu and then type the code in the new file that appears. Now the code in the script area (or R File) does not execute automatically, instead place the cursor on the line which needs to be executed and select **RUN** option or press **Ctrl + Enter** (for windows). To run multiple lines of code, select all the lines first and then select **RUN** option or press **Ctrl + Enter.**

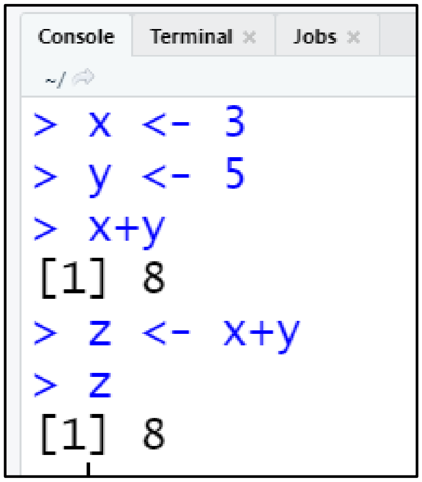


```
2+2
2-2
2*2
2/2
3%%2
```

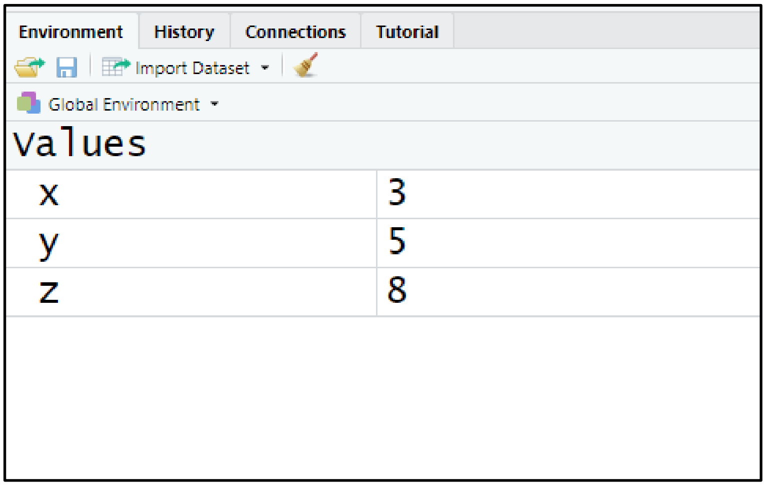
Values can be assigned to variables in R using the "**<-**" symbol. The variable is written on the left and is assigned the value on the right side. For example, to assign a value of 3 to x we can type the below code,

x <- 3

Assigning values to variables are quite useful especially if these values would be used again. Similar to the previous examples, operations can be performed on the variables to get output directly (or) the output can be stored in a different variable. Once a variable is created it will be visible under the environment section

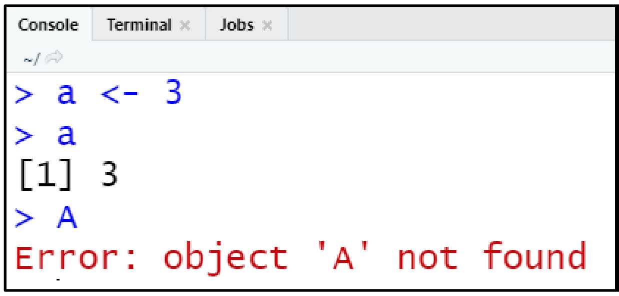


```
> x <- 3
> y <- 5
> x+y
[1] 8
> z <- x+y
> z
[1] 8
```

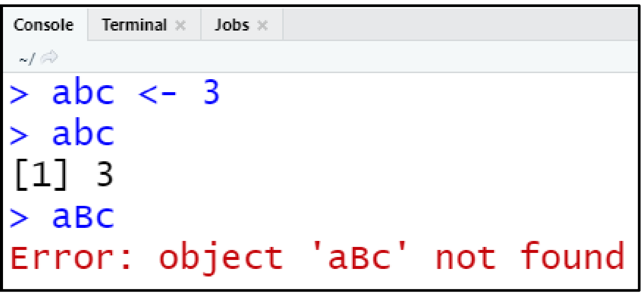


Values	
x	3
y	5
z	8

One thing to be aware of is that R is **case-sensitive**. Hence variable "a" is different from "A"



```
> a <- 3
> a
[1] 3
> A
Error: object 'A' not found
```



```
> abc <- 3
> abc
[1] 3
> aBc
Error: object 'aBc' not found
```

PACKAGES

Package - collection of R functions, data and compiled code

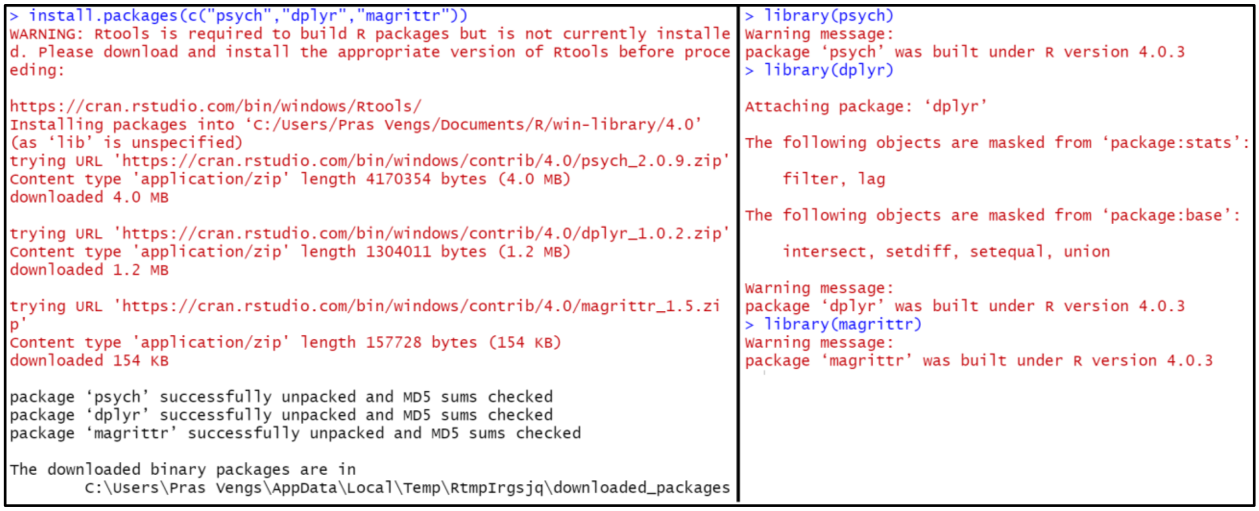
Library - The location where the packages are stored

If there is a particular functionality that you require, you can download the package from the appropriate site, and it will be stored in your library. To use the package, use the command **library()** to load the package in the current R session. Then just call the appropriate package functions

install.packages("*package_name*") – Install the package from CRAN repository

install.packages(c("*package_1*", “"*package_2*", "*package_3*")) – Install multiple packages

library("*package_name*") – Load the package in current R session

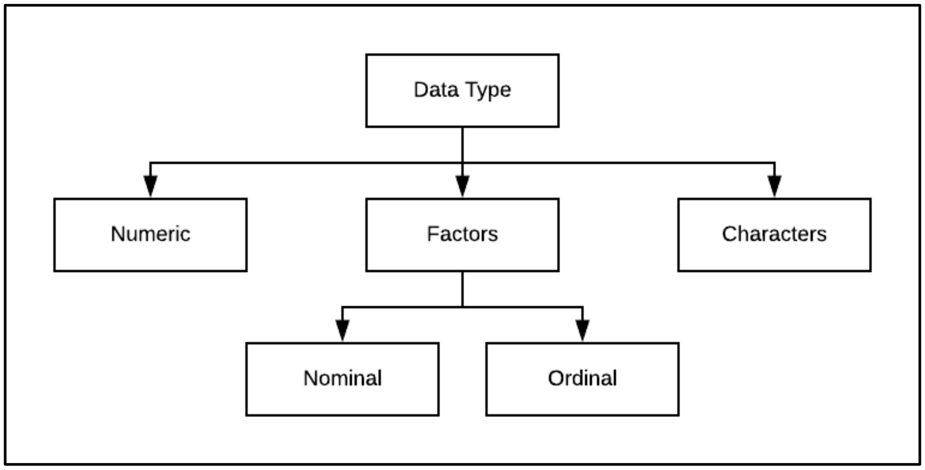


```
> install.packages(c("psych","dplyr","magrittr"))
WARNING: Rtools is required to build R packages but is not currently installe
d. Please download and install the appropriate version of Rtools before proce
eding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing packages into ‘C:/Users/Pras Vengs/Documents/R/win-library/4.0’
(as ‘lib’ is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/psych_2.0.9.zip'
Content type 'application/zip' length 4170354 bytes (4.0 MB)
downloaded 4.0 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/dplyr_1.0.2.zip'
Content type 'application/zip' length 1304011 bytes (1.2 MB)
downloaded 1.2 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/magrittr_1.5.zi
p'
Content type 'application/zip' length 157728 bytes (154 KB)
downloaded 154 KB

package ‘psych’ successfully unpacked and MD5 sums checked
package ‘dplyr’ successfully unpacked and MD5 sums checked
package ‘magrittr’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Pras Vengs\AppData\Local\Temp\RtmpIrgsjq\downloaded_packages
```

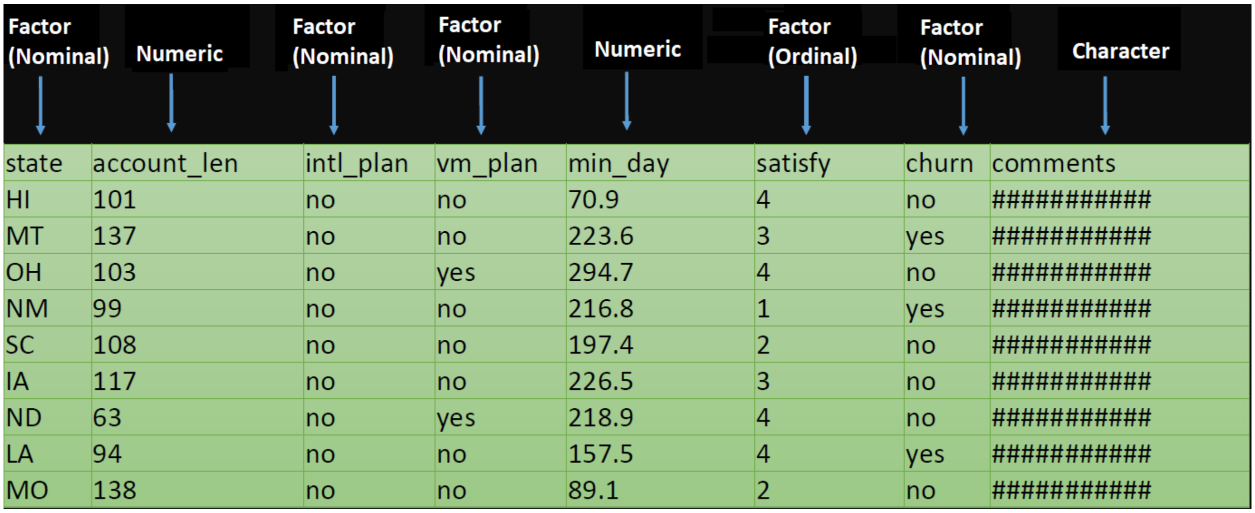
```
> library(psych)
Warning message:
package ‘psych’ was built under R version 4.0.3
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

Warning message:
package ‘dplyr’ was built under R version 4.0.3
> library(magrittr)
Warning message:
package ‘magrittr’ was built under R version 4.0.3
```

You can ignore the warnings

DATA TYPES



1. **Character** – Letter, words or strings which vary mostly and do not have any particular meaning for the data, e.g. Comments, Usernames
2. **Numeric** – Numbers which are continuous, e.g. Age, Temperature, Location coordinates
3. **Factors** – Can be numeric/character but have only a fixed set of values, e.g. Age groups, Race
   a) **Nominal –** The order of the factors does not matter since they have no significance, e.g. Country, Gender, Race
   b) **Ordinal –** These factors can be ordered since they hold a numerical significance, e.g. Satisfaction rating ("extreme dislike", "dislike", "neutral", "like", "extreme like")



Factor (Nominal)	Numeric	Factor (Nominal)	Factor (Nominal)	Numeric	Factor (Ordinal)	Factor (Nominal)	Character
state	account_len	intl_plan	vm_plan	min_day	satisfy	churn	comments
HI	101	no	no	70.9	4	no	###########
MT	137	no	no	223.6	3	yes	###########
OH	103	no	yes	294.7	4	no	###########
NM	99	no	no	216.8	1	yes	###########
SC	108	no	no	197.4	2	no	###########
IA	117	no	no	226.5	3	no	###########
ND	63	no	yes	218.9	4	no	###########
LA	94	no	no	157.5	4	yes	###########
MO	138	no	no	89.1	2	no	###########

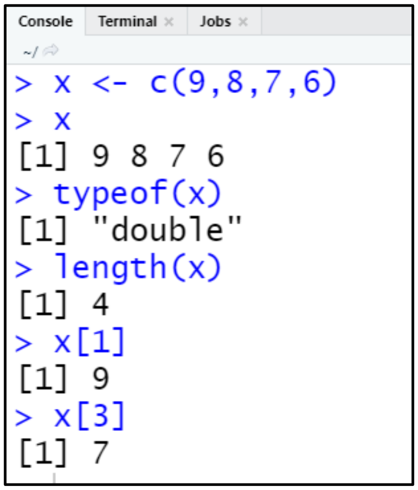
DATA STRUCTURES

1. VECTORS

Vectors are the basic data structure of R. Vectors can hold multiple values together using the concatenate **c()** function. The type of data inside a vector can be determined by using **typeof()** function and the length (or) number of elements in a vector can be found with **length()** function.

R uses one indexing, hence the position of the first component in a vector can be accessed by,

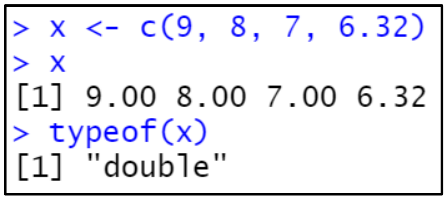
vector_name[1]



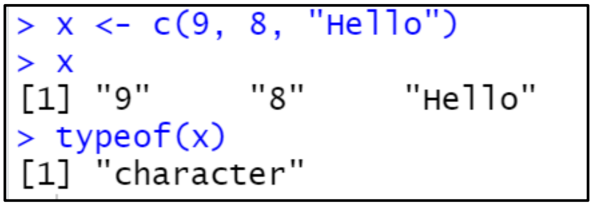
```
> x <- c(9,8,7,6)
> x
[1] 9 8 7 6
> typeof(x)
[1] "double"
> length(x)
[1] 4
> x[1]
[1] 9
> x[3]
[1] 7
```

A vector will **always** contain data of same data type. If a vector contains multiple data types the vector will convert all its values to the same data type in the below order of precedence:

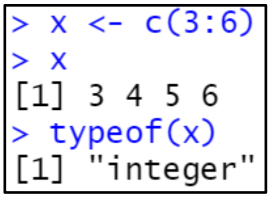
- Character
- Double (Float / Decimals)
- Integers (Round whole numbers)



```
> x <- c(9, 8, 7, 6.32)
> x
[1] 9.00 8.00 7.00 6.32
> typeof(x)
[1] "double"
```



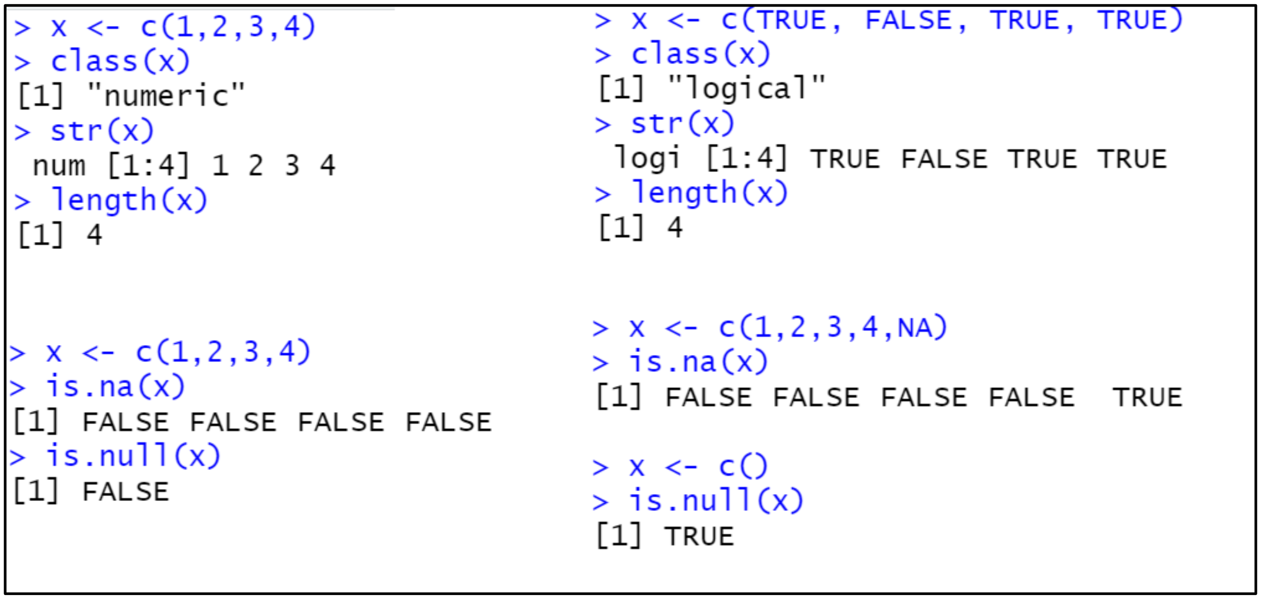
```
> x <- c(9, 8, "Hello")
> x
[1] "9"     "8"     "Hello"
> typeof(x)
[1] "character"
```



```
> x <- c(3:6)
> x
[1] 3 4 5 6
> typeof(x)
[1] "integer"
```

Analyzing a Vector:

class(*vector_name*)	-	Type of data present inside the vector
str(*vector_name*)	-	Structure of the vector
is.na(*vector_name*)	-	Checks if each element of vector is "NA"
is.null(*vector_name*)	-	Checks if the entire vector is empty
length(*vector_name*)	-	Number of elements present inside the vector



```
> x <- c(1,2,3,4)
> class(x)
[1] "numeric"
> str(x)
 num [1:4] 1 2 3 4
> length(x)
[1] 4
```

```
> x <- c(TRUE, FALSE, TRUE, TRUE)
> class(x)
[1] "logical"
> str(x)
 logi [1:4] TRUE FALSE TRUE TRUE
> length(x)
[1] 4
```

```
> x <- c(1,2,3,4)
> is.na(x)
[1] FALSE FALSE FALSE FALSE
> is.null(x)
[1] FALSE
```

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

> x <- c()
> is.null(x)
[1] TRUE
```

Subsetting a vector:

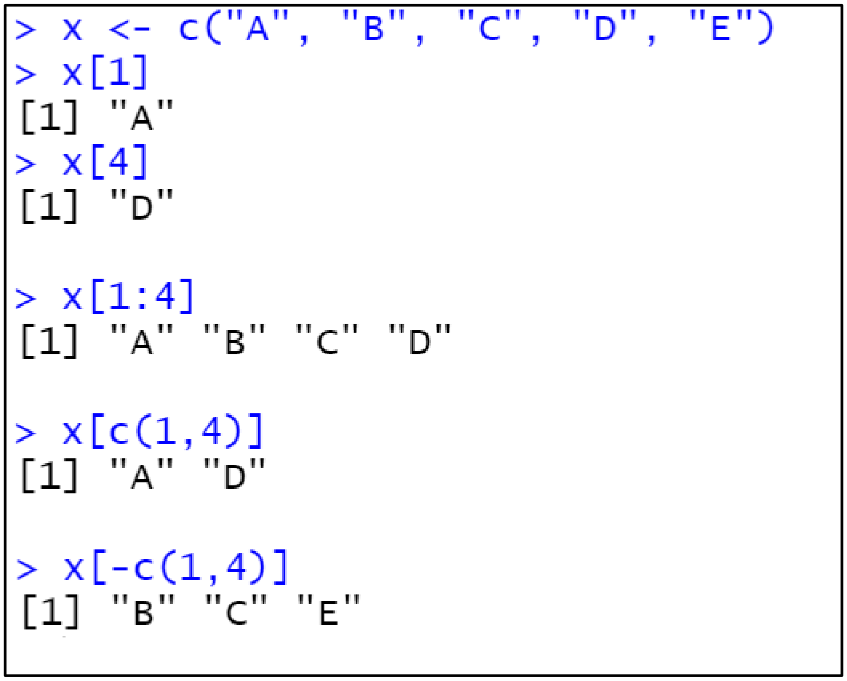
R uses one-indexing mechanism where the elements in the vector start with an index number of one.

vector_name[4] - Element at the fourth position (index) in the vector

vector_name[1:4] - Elements from positions 1 to 4 in the vector

vector_name[c(1,4)] - Elements at positions 1 & 4 only in the vector

vector_name[-c(1,4)] - All elements except those at positions 1 & 4 in the vector



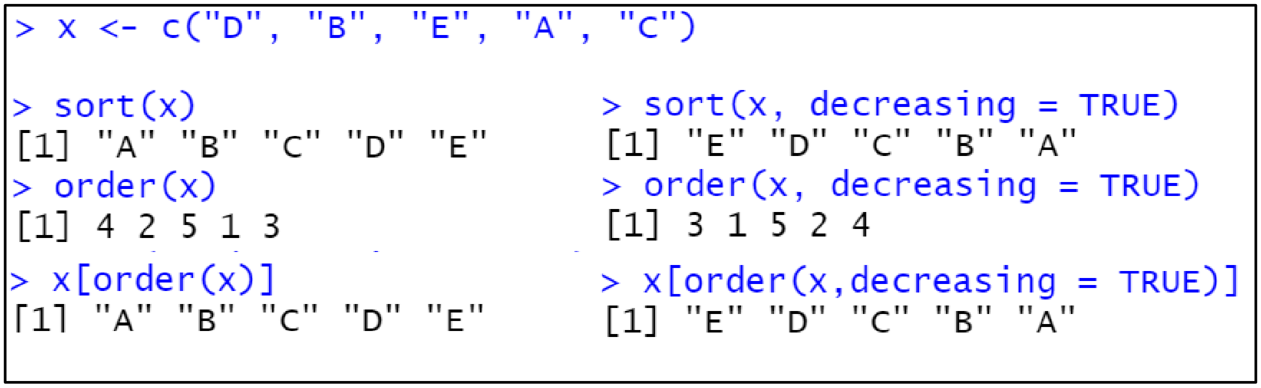
```
> x <- c("A", "B", "C", "D", "E")
> x[1]
[1] "A"
> x[4]
[1] "D"

> x[1:4]
[1] "A" "B" "C" "D"

> x[c(1,4)]
[1] "A" "D"

> x[-c(1,4)]
[1] "B" "C" "E"
```

Sorting a vector:

Sorting of a vector can be performed using two different functions

sort(*vector*) - Sorts the vector numerically or alphabetically based on vector type (ascending by default)

order(*vector*) - Returns the indices of the vector in the order they would appear when the vector is sorted (ascending by default)



```
> x <- c("D", "B", "E", "A", "C")

> sort(x)                        > sort(x, decreasing = TRUE)
[1] "A" "B" "C" "D" "E"          [1] "E" "D" "C" "B" "A"
> order(x)                       > order(x, decreasing = TRUE)
[1] 4 2 5 1 3                    [1] 3 1 5 2 4

> x[order(x)]                    > x[order(x,decreasing = TRUE)]
[1] "A" "B" "C" "D" "E"          [1] "E" "D" "C" "B" "A"
```

2. DATA FRAME:

Data frames are used for storing Data tables in R. They are two-dimensional array structure and are similar to tables where each column represents one variable. The main features to note about a data frame are:

- Columns can be of different data types
- Each column name must be unique
- Each column should be of same length i.e. contain the same number of elements

Data frames in R can be created in two ways:

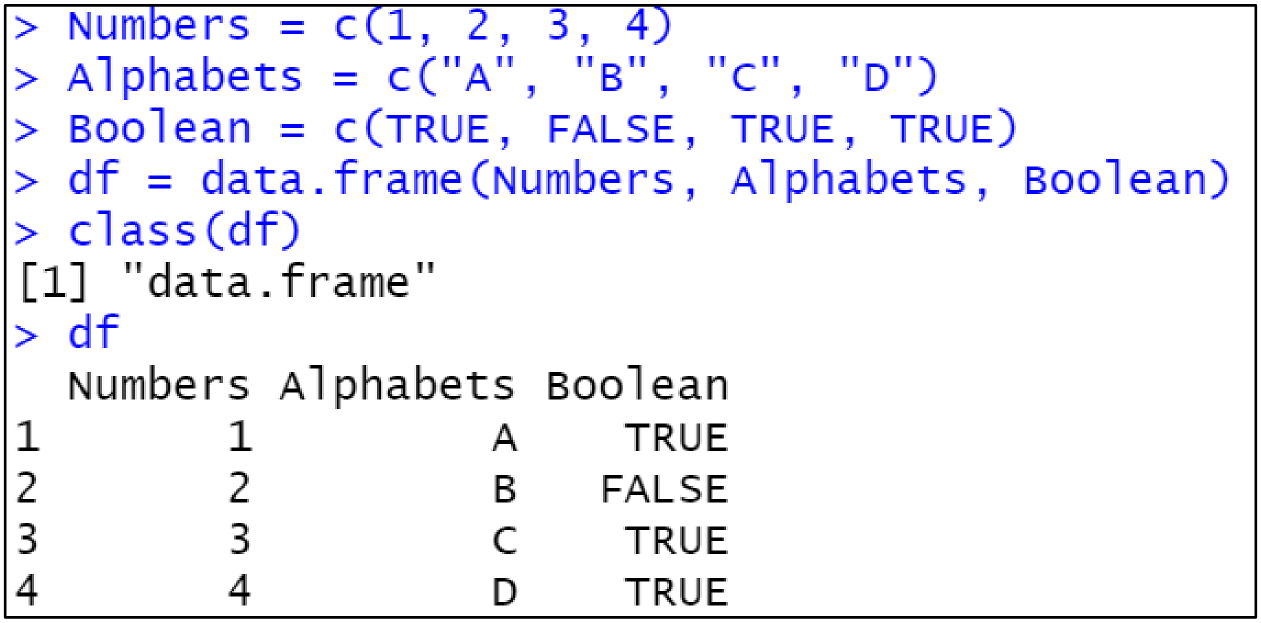
- Using data.frame() command
- Importing data from files such as .csv, .xlsx etc.

data.frame() FUNCTION:

While using the command we can follow the below syntax

data.frame(*column_1, column_2, column_3,* ……………………)

Make sure that the names of the columns are unique and are of same length.



```
> Numbers = c(1, 2, 3, 4)
> Alphabets = c("A", "B", "C", "D")
> Boolean = c(TRUE, FALSE, TRUE, TRUE)
> df = data.frame(Numbers, Alphabets, Boolean)
> class(df)
[1] "data.frame"
> df
  Numbers Alphabets Boolean
1       1         A    TRUE
2       2         B   FALSE
3       3         C    TRUE
4       4         D    TRUE
```

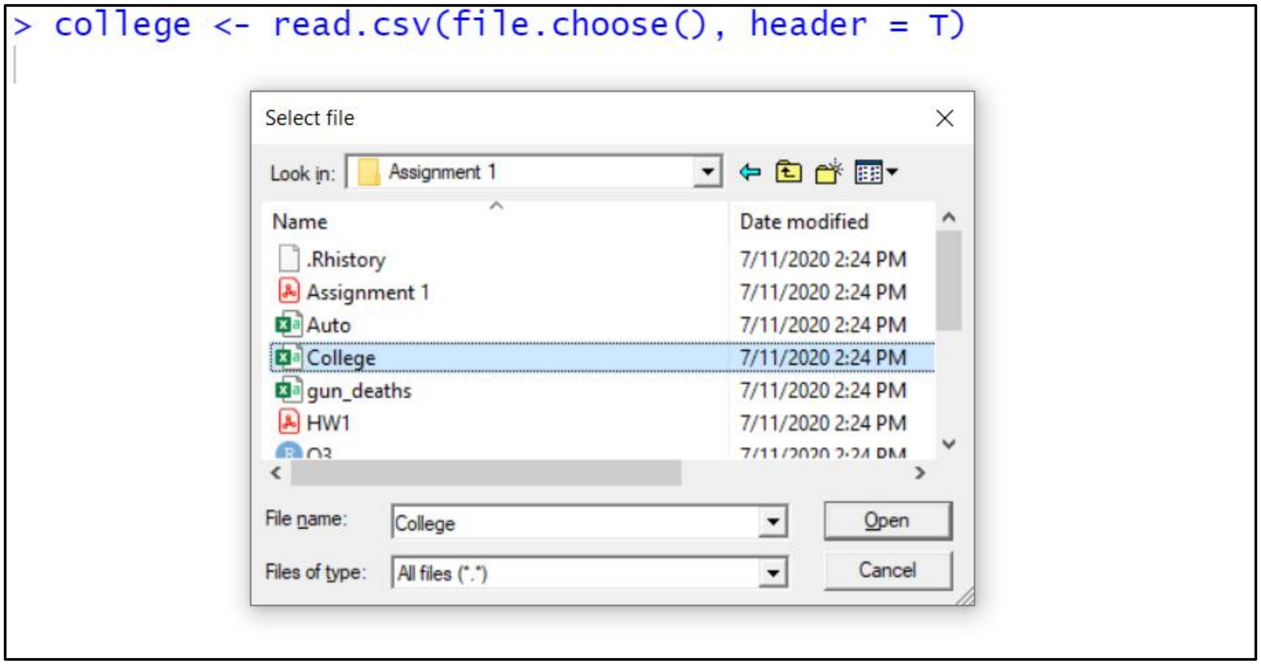
IMPORTING DATA:

There are multiple commands with various arguments to import data from different file formats into R environment. I shall show the simplest command to import a csv file as a data frame

data_frame_name <- read.csv(file.choose(), header = T)

Here, file.choose() - Allows you to choose a .csv file stored in your local desktop

header = T - Indicates the first column in the file contains column names



```
> college <- read.csv(file.choose(), header = T)
```

Select file

Look in: Assignment 1

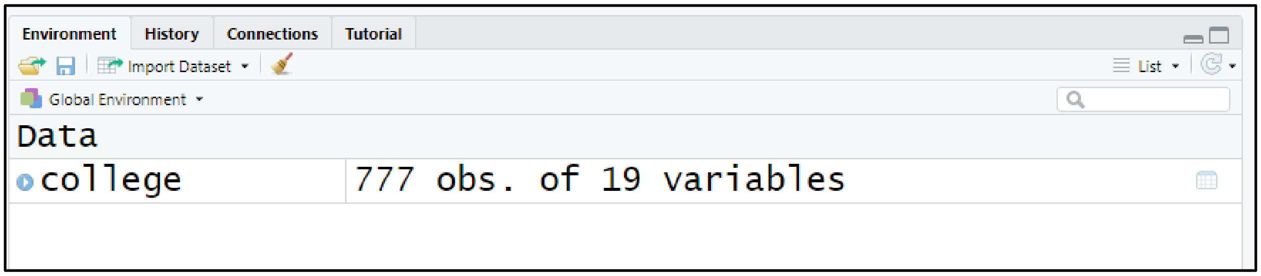
Name	Date modified
.Rhistory	7/11/2020 2:24 PM
Assignment 1	7/11/2020 2:24 PM
Auto	7/11/2020 2:24 PM
College	7/11/2020 2:24 PM
gun_deaths	7/11/2020 2:24 PM
HW1	7/11/2020 2:24 PM
Q3	7/11/2020 2:24 PM

File name: College

Files of type: All files (*.*)

Double click (or) click once and select open on your desired file to import

Once the data has been imported successfully the data frame would be visible with its name in the Environment part on the top right



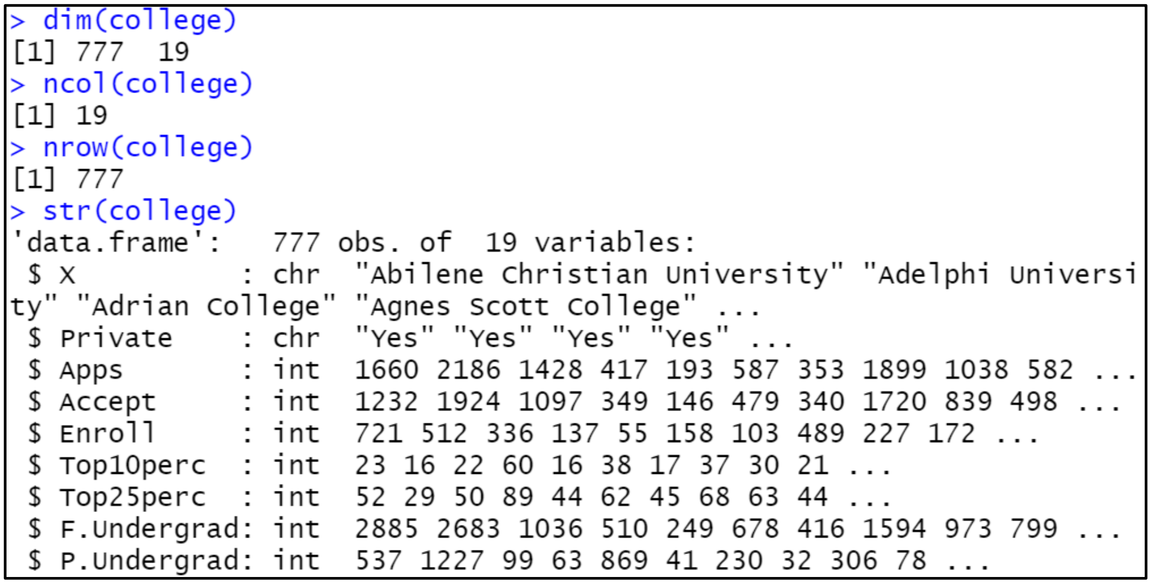
```
Data
college                777 obs. of 19 variables
```

Analyzing a data frame:

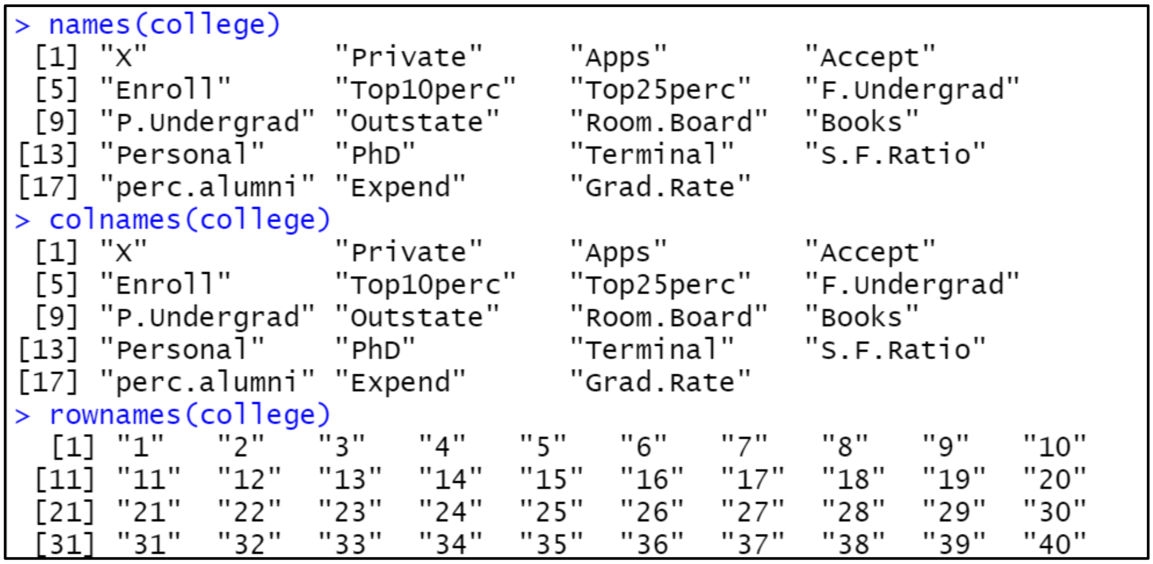
dim(*data_frame*)	-	Dimensions of the data frame in the order of (row columns)
ncol(*data_frame*)	-	Number of columns
nrow(*data_frame*)	-	Number of rows
str(*data_frame*)	-	Structure of the data frame

names(*data_frame*)
colnames(*data_frame*)
	-	Names of the columns

rownames(*data_frame*)	-	Names of the rows



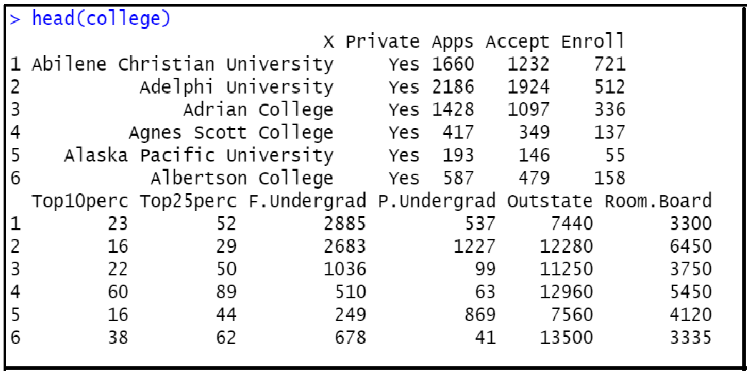
```
> dim(college)
[1] 777  19
> ncol(college)
[1] 19
> nrow(college)
[1] 777
> str(college)
'data.frame':	777 obs. of  19 variables:
 $ X          : chr  "Abilene Christian University" "Adelphi Universi
ty" "Adrian College" "Agnes Scott College" ...
 $ Private    : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ Apps       : int  1660 2186 1428 417 193 587 353 1899 1038 582 ...
 $ Accept     : int  1232 1924 1097 349 146 479 340 1720 839 498 ...
 $ Enroll     : int  721 512 336 137 55 158 103 489 227 172 ...
 $ Top10perc  : int  23 16 22 60 16 38 17 37 30 21 ...
 $ Top25perc  : int  52 29 50 89 44 62 45 68 63 44 ...
 $ F.Undergrad: int  2885 2683 1036 510 249 678 416 1594 973 799 ...
 $ P.Undergrad: int  537 1227 99 63 869 41 230 32 306 78 ...
```



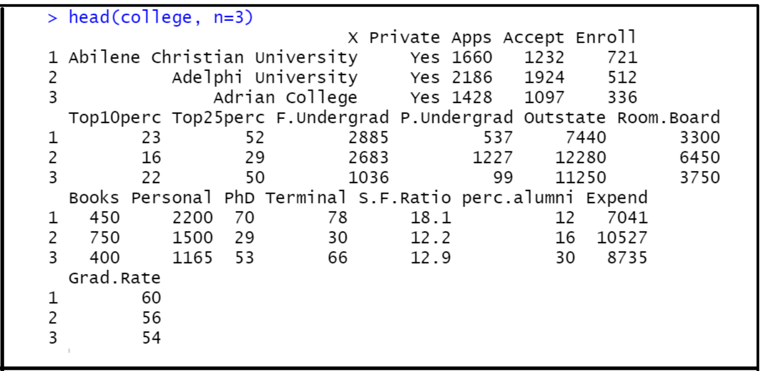
```
> names(college)
 [1] "X"           "Private"     "Apps"        "Accept"
 [5] "Enroll"      "Top10perc"   "Top25perc"   "F.Undergrad"
 [9] "P.Undergrad" "Outstate"    "Room.Board"  "Books"
[13] "Personal"    "PhD"         "Terminal"    "S.F.Ratio"
[17] "perc.alumni" "Expend"      "Grad.Rate"
> colnames(college)
 [1] "X"           "Private"     "Apps"        "Accept"
 [5] "Enroll"      "Top10perc"   "Top25perc"   "F.Undergrad"
 [9] "P.Undergrad" "Outstate"    "Room.Board"  "Books"
[13] "Personal"    "PhD"         "Terminal"    "S.F.Ratio"
[17] "perc.alumni" "Expend"      "Grad.Rate"
> rownames(college)
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
 [11] "11"  "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"
 [21] "21"  "22"  "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"
 [31] "31"  "32"  "33"  "34"  "35"  "36"  "37"  "38"  "39"  "40"
```

head(*data_frame, n*) - Data present in the first n rows of data frame (n=6 by default)

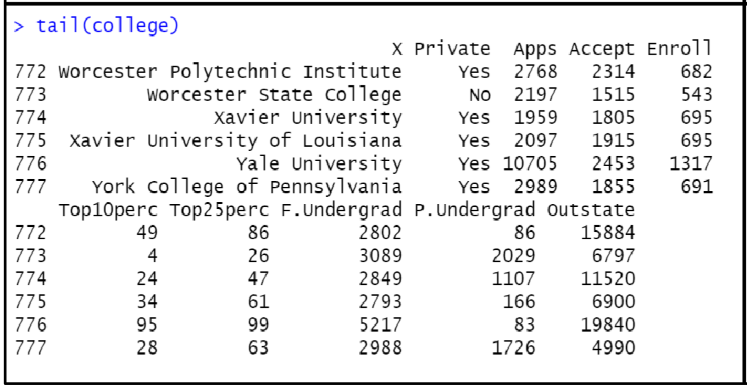
tail(*data_frame, n*) - Data present in the last n rows of data frame (n=6 by default)



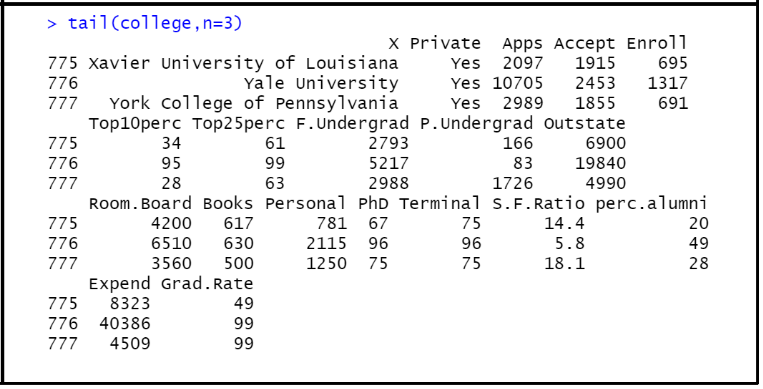
```
> head(college)
                             X Private Apps Accept Enroll
1 Abilene Christian University     Yes 1660   1232    721
2           Adelphi University     Yes 2186   1924    512
3               Adrian College     Yes 1428   1097    336
4          Agnes Scott College     Yes  417    349    137
5    Alaska Pacific University     Yes  193    146     55
6            Albertson College     Yes  587    479    158
  Top10perc Top25perc F.Undergrad P.Undergrad Outstate Room.Board
1        23        52        2885         537     7440       3300
2        16        29        2683        1227    12280       6450
3        22        50        1036          99    11250       3750
4        60        89         510          63    12960       5450
5        16        44         249         869     7560       4120
6        38        62         678          41    13500       3335
```



```
> head(college, n=3)
                             X Private Apps Accept Enroll
1 Abilene Christian University     Yes 1660   1232    721
2           Adelphi University     Yes 2186   1924    512
3               Adrian College     Yes 1428   1097    336
  Top10perc Top25perc F.Undergrad P.Undergrad Outstate Room.Board
1        23        52        2885         537     7440       3300
2        16        29        2683        1227    12280       6450
3        22        50        1036          99    11250       3750
  Books Personal PhD Terminal S.F.Ratio perc.alumni Expend
1   450     2200  70       78      18.1          12   7041
2   750     1500  29       30      12.2          16  10527
3   400     1165  53       66      12.9          30   8735
  Grad.Rate
1        60
2        56
3        54
```



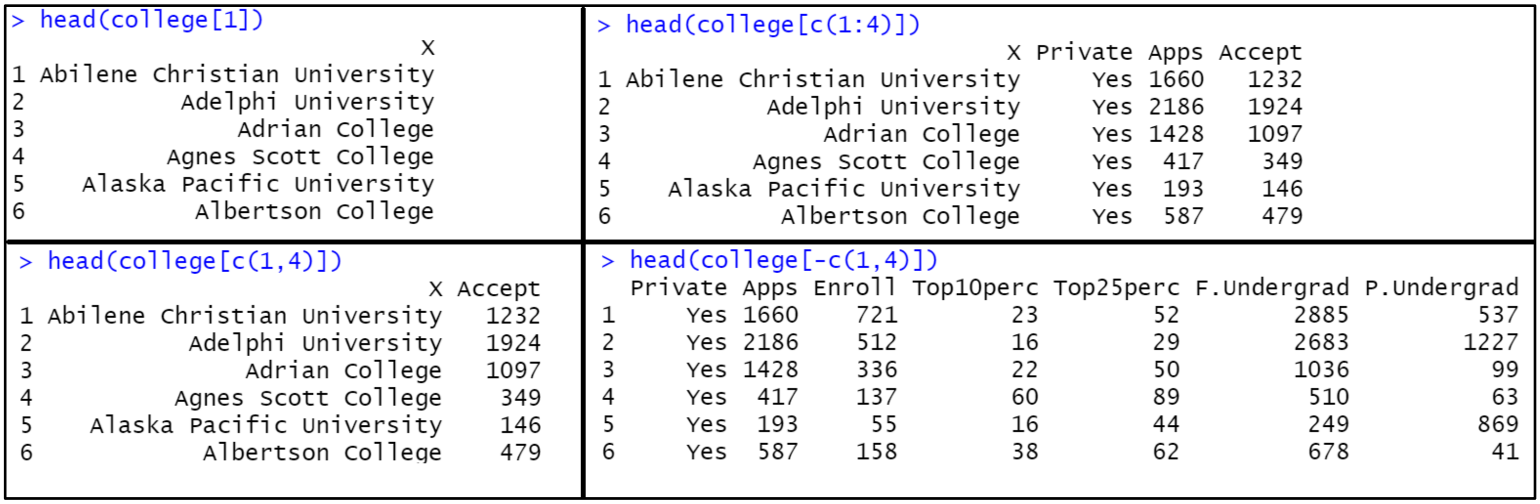
```
> tail(college)
                                  X Private  Apps Accept Enroll
772 Worcester Polytechnic Institute     Yes  2768   2314    682
773         Worcester State College      No  2197   1515    543
774               Xavier University     Yes  1959   1805    695
775  Xavier University of Louisiana     Yes  2097   1915    695
776                 Yale University     Yes 10705   2453   1317
777    York College of Pennsylvania     Yes  2989   1855    691
    Top10perc Top25perc F.Undergrad P.Undergrad Outstate
772        49        86        2802          86    15884
773         4        26        3089        2029     6797
774        24        47        2849        1107    11520
775        34        61        2793         166     6900
776        95        99        5217          83    19840
777        28        63        2988        1726     4990
```



```
> tail(college,n=3)
                                 X Private  Apps Accept Enroll
775 Xavier University of Louisiana     Yes  2097   1915    695
776                Yale University     Yes 10705   2453   1317
777   York College of Pennsylvania     Yes  2989   1855    691
    Top10perc Top25perc F.Undergrad P.Undergrad Outstate
775        34        61        2793         166     6900
776        95        99        5217          83    19840
777        28        63        2988        1726     4990
    Room.Board Books Personal PhD Terminal S.F.Ratio perc.alumni
775       4200   617      781  67       75      14.4          20
776       6510   630     2115  96       96       5.8          49
777       3560   500     1250  75       75      18.1          28
    Expend Grad.Rate
775   8323        49
776  40386        99
777   4509        99
```

Subsetting a data frame:

data_frame[1]   - Display only first column
data_frame[c(1,4)]   - Display only column one & four
data_frame[-c(1,4)]   - Display all columns except one & four
data_frame[c(1:4)]   - Display columns one to four



```
> head(college[1])
                            X
1 Abilene Christian University
2           Adelphi University
3               Adrian College
4          Agnes Scott College
5    Alaska Pacific University
6            Albertson College
```

```
> head(college[c(1:4)])
                            X Private Apps Accept
1 Abilene Christian University     Yes 1660   1232
2           Adelphi University     Yes 2186   1924
3               Adrian College     Yes 1428   1097
4          Agnes Scott College     Yes  417    349
5    Alaska Pacific University     Yes  193    146
6            Albertson College     Yes  587    479
```

```
> head(college[c(1,4)])
                            X Accept
1 Abilene Christian University   1232
2           Adelphi University   1924
3               Adrian College   1097
4          Agnes Scott College    349
5    Alaska Pacific University    146
6            Albertson College    479
```

```
> head(college[-c(1,4)])
  Private Apps Enroll Top10perc Top25perc F.Undergrad P.Undergrad
1     Yes 1660    721        23        52        2885         537
2     Yes 2186    512        16        29        2683        1227
3     Yes 1428    336        22        50        1036          99
4     Yes  417    137        60        89         510          63
5     Yes  193     55        16        44         249         869
6     Yes  587    158        38        62         678          41
```

Consider a dataframe created with the below commands:

Numbers = c(1, 2, 3, 4)

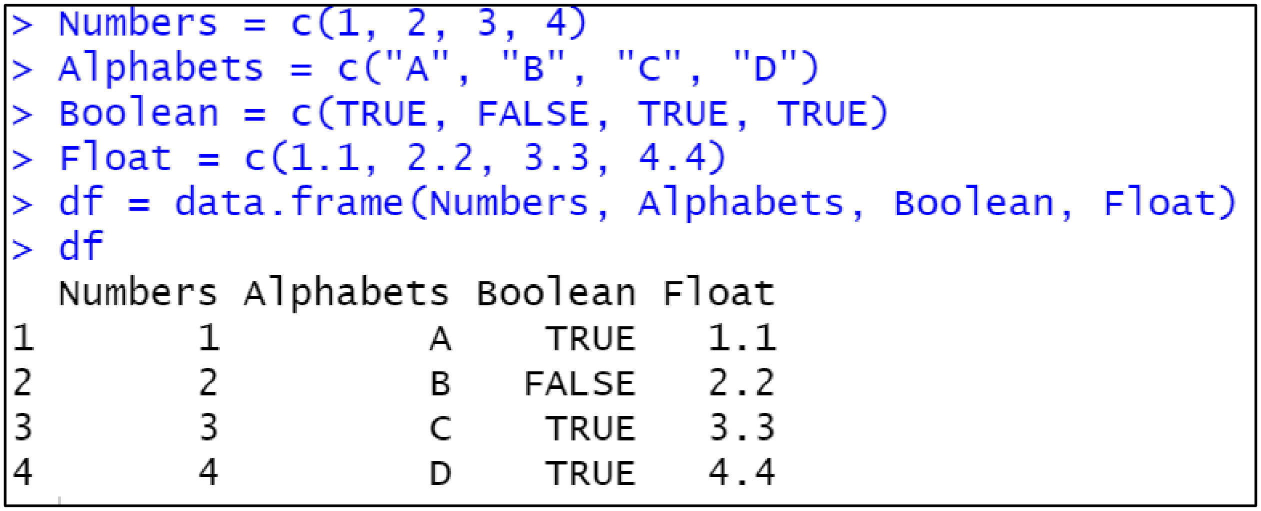
Alphabets = c("A", "B", "C", "D")

Boolean = c(TRUE, FALSE, TRUE, TRUE)

Float = c(1.1, 2.2, 3.3, 4.4)

df = data.frame(Numbers, Alphabets, Boolean, Float)

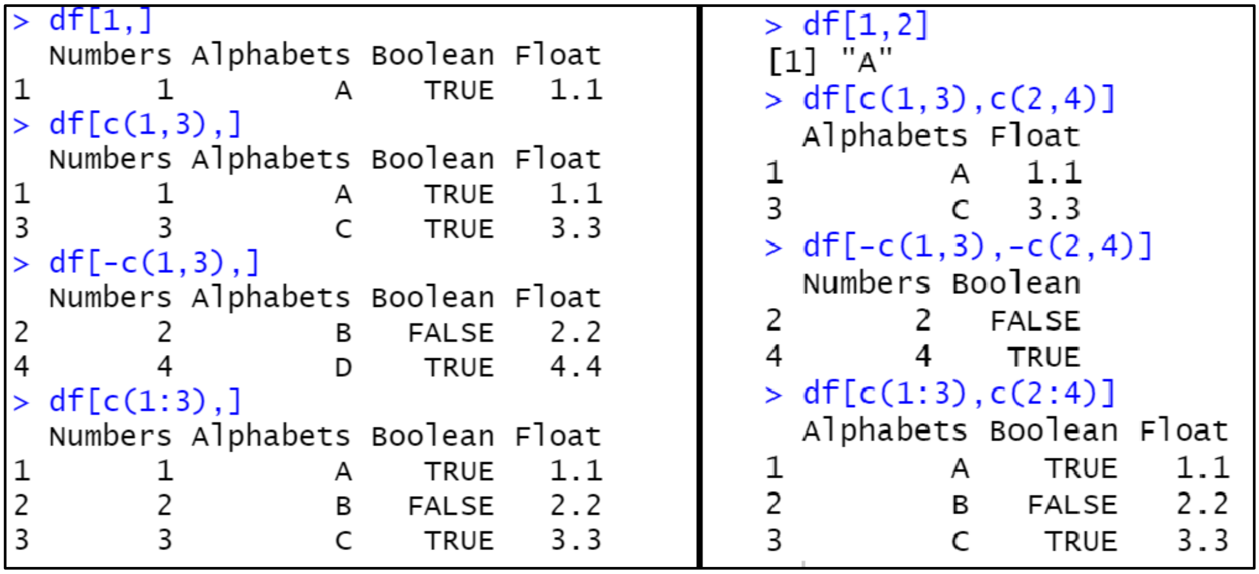
df



```
> Numbers = c(1, 2, 3, 4)
> Alphabets = c("A", "B", "C", "D")
> Boolean = c(TRUE, FALSE, TRUE, TRUE)
> Float = c(1.1, 2.2, 3.3, 4.4)
> df = data.frame(Numbers, Alphabets, Boolean, Float)
> df
  Numbers Alphabets Boolean Float
1       1         A    TRUE   1.1
2       2         B   FALSE   2.2
3       3         C    TRUE   3.3
4       4         D    TRUE   4.4
```

data_frame_name[*rows*, *columns*] – Subset based on rows and columns

Command	Rows Selected	Columns selected
df[1 ,]	One	All
df[c(1,3) ,]	One & Three	
df[-c(1,3) ,]	All except One & Three	
df[c(1:3) ,]	One to Three	
df[1 , 2]	One	Two
df[c(1,3) , c(2,4)]	One & Three	Two & Four
df[-c(1,3) , -c(2,4)]	All except One & Three	All except Two & Four
df[c(1:3) , c(2:4)]	One to Three	Two to Four



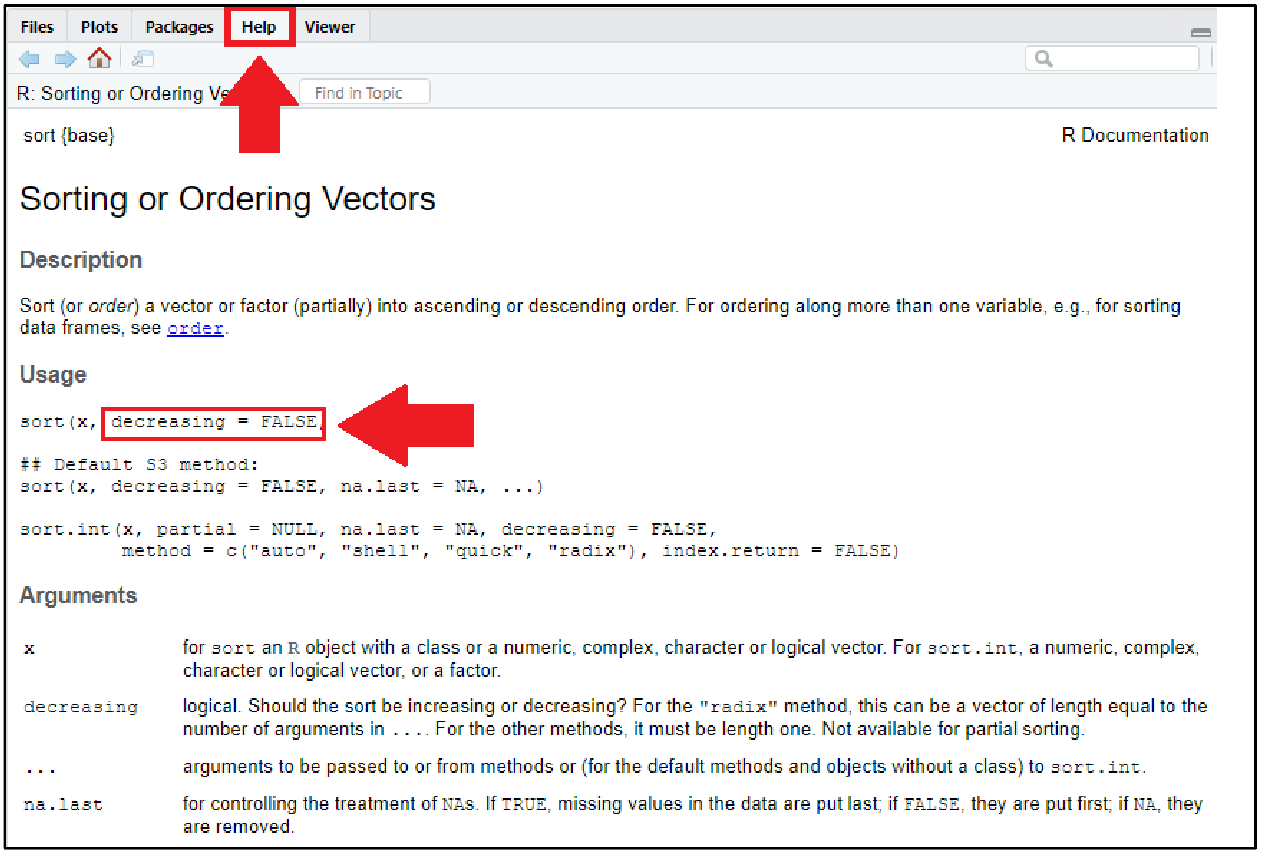
```
> df[1,]
  Numbers Alphabets Boolean Float
1       1         A    TRUE   1.1
> df[c(1,3),]
  Numbers Alphabets Boolean Float
1       1         A    TRUE   1.1
3       3         C    TRUE   3.3
> df[-c(1,3),]
  Numbers Alphabets Boolean Float
2       2         B   FALSE   2.2
4       4         D    TRUE   4.4
> df[c(1:3),]
  Numbers Alphabets Boolean Float
1       1         A    TRUE   1.1
2       2         B   FALSE   2.2
3       3         C    TRUE   3.3
```

```
> df[1,2]
[1] "A"
> df[c(1,3),c(2,4)]
  Alphabets Float
1         A   1.1
3         C   3.3
> df[-c(1,3),-c(2,4)]
  Numbers Boolean
2       2   FALSE
4       4    TRUE
> df[c(1:3),c(2:4)]
  Alphabets Boolean Float
1         A    TRUE   1.1
2         B   FALSE   2.2
3         C    TRUE   3.3
```

Help function:

help(*function_name*) – Provides detailed description of function in help window (bottom right)

E.g. Run the command **help(sort)** in the console



sort {base} R Documentation

Sorting or Ordering Vectors

Description

Sort (or *order*) a vector or factor (partially) into ascending or descending order. For ordering along more than one variable, e.g., for sorting data frames, see `order`.

Usage

```
sort(x, decreasing = FALSE, ...)

## Default S3 method:
sort(x, decreasing = FALSE, na.last = NA, ...)

sort.int(x, partial = NULL, na.last = NA, decreasing = FALSE,
         method = c("auto", "shell", "quick", "radix"), index.return = FALSE)
```

Arguments

`x`	for `sort` an `R` object with a class or a numeric, complex, character or logical vector. For `sort.int`, a numeric, complex, character or logical vector, or a factor.
`decreasing`	logical. Should the sort be increasing or decreasing? For the `"radix"` method, this can be a vector of length equal to the number of arguments in `...`. For the other methods, it must be length one. Not available for partial sorting.
`...`	arguments to be passed to or from methods or (for the default methods and objects without a class) to `sort.int`.
`na.last`	for controlling the treatment of `NA`s. If `TRUE`, missing values in the data are put last; if `FALSE`, they are put first; if `NA`, they are removed.

You will now get a complete description of the "sort" function in the help window

Points to note:

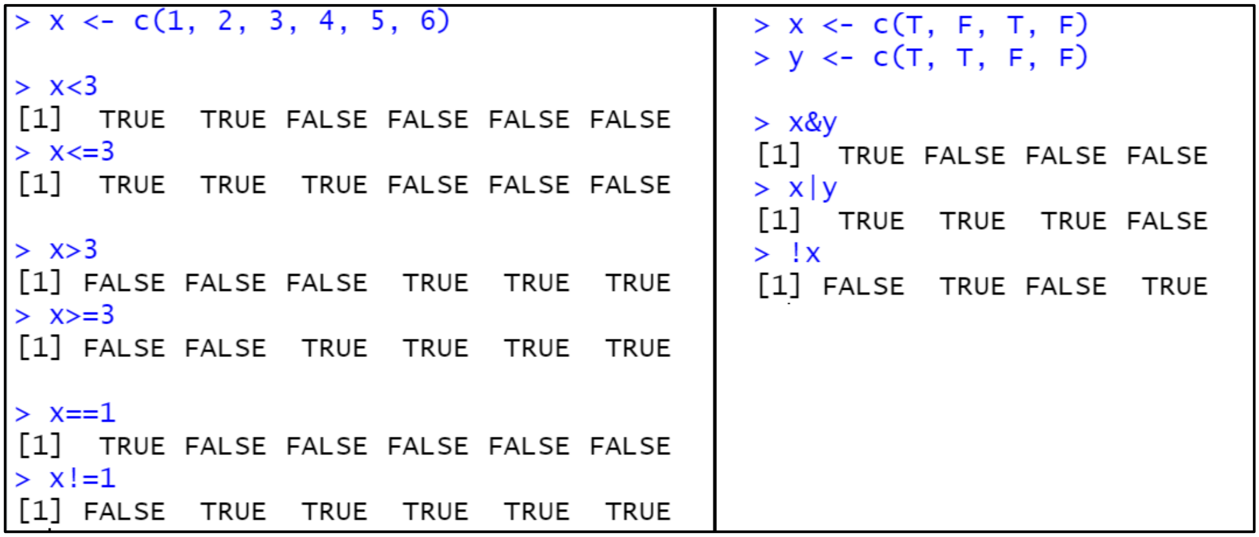
- If a function's argument is not given any value (such as x in the above picture) in the help description, this value must be compulsorily specified while running the function
- If a function's argument is given a value (decreasing = FALSE in above pic) this value is the default value considered by R. It needs to be specified compulsorily when the argument's value needs to be different

LOGICAL OPERATORS:

Provides a list of Boolean results based on operation performed

<	-	Less than
<=	-	Less than or equal to
>	-	Greater than
>=	-	Greater than or equal to
==	-	Equal to
!=	-	Not equal to
x&y	-	AND operation
x|y	-	OR operation
!x	-	NOT operation

Please note that in R the Boolean values "TRUE" & "FALSE" can also be written as "T" &"F"



```
> x <- c(1, 2, 3, 4, 5, 6)

> x<3
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE
> x<=3
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE

> x>3
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
> x>=3
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE

> x==1
[1]  TRUE FALSE FALSE FALSE FALSE FALSE
> x!=1
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
> x <- c(T, F, T, F)
> y <- c(T, T, F, F)

> x&y
[1]  TRUE FALSE FALSE FALSE
> x|y
[1]  TRUE  TRUE  TRUE FALSE
> !x
[1] FALSE  TRUE FALSE  TRUE
```

UNIVARIATE ANALYSIS

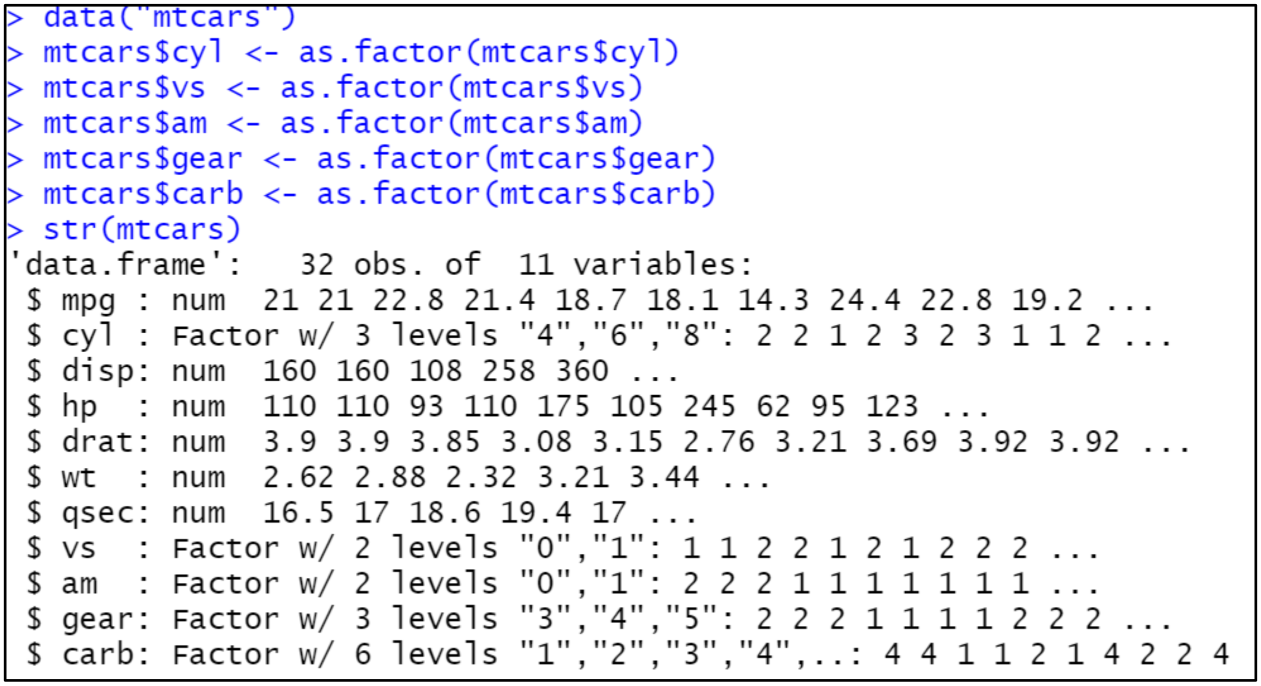
R has many in-built datasets using which we can use. You can find the available list of datasets and its description here. You can also view a detailed description of the data set using the command **help("*data_set_name*")**

1. FACTOR:

Kindly run the below command to load the in-built dataset **mtcars** in your R session:

```
library(datasets)
data("mtcars")
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$vs <- as.factor(mtcars$vs)
mtcars$am <- as.factor(mtcars$am)
mtcars$gear <- as.factor(mtcars$gear)
mtcars$carb <- as.factor(mtcars$carb)
str(mtcars)
```

The as.factor is used to convert a column of different data type to a factor data type



```
> data("mtcars")
> mtcars$cyl <- as.factor(mtcars$cyl)
> mtcars$vs <- as.factor(mtcars$vs)
> mtcars$am <- as.factor(mtcars$am)
> mtcars$gear <- as.factor(mtcars$gear)
> mtcars$carb <- as.factor(mtcars$carb)
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 1 2 2 2 ...
 $ am  : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
 $ gear: Factor w/ 3 levels "3","4","5": 2 2 2 1 1 1 1 2 2 2 ...
 $ carb: Factor w/ 6 levels "1","2","3","4",..: 4 4 1 1 2 1 4 2 2 4
```

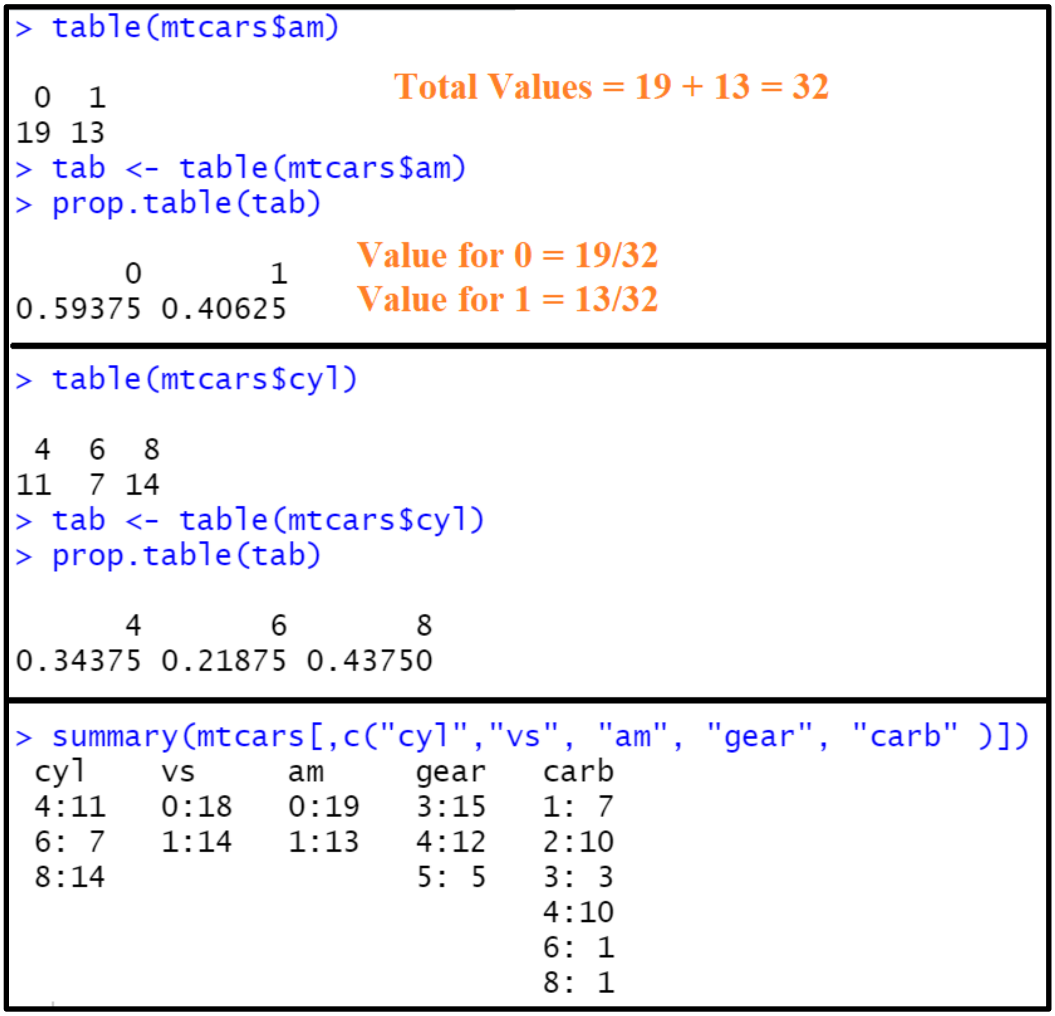
The columns **cyl, vs, am, gear, carb** are all factors with **3, 2, 2, 3, 6** individual values, respectively.

To select an individual column, you can write the code in below format using **$** symbol

Data_frame_name$*column_name*
Eg: mtcars$cyl

Below are commands to perform univariate (single variable) analysis in R

Command	Description
table(*column*)	Displays a table with number of occurrences for each value of the column
prop.table(*table*)	Displays a table with the percentage of number of occurrences for each value of the column. Kindly note the input provided for this command should be a table data type.
summary()	Complete summary of the column(s)



```
> table(mtcars$am)

 0  1 
19 13 
> tab <- table(mtcars$am)
> prop.table(tab)

      0       1 
0.59375 0.40625 
```

Total Values = 19 + 13 = 32

Value for 0 = 19/32
Value for 1 = 13/32

```
> table(mtcars$cyl)

 4  6  8 
11  7 14 
> tab <- table(mtcars$cyl)
> prop.table(tab)

      4       6       8 
0.34375 0.21875 0.43750 
```

```
> summary(mtcars[,c("cyl","vs", "am", "gear", "carb" )])
 cyl     vs      am      gear    carb  
 4:11   0:18   0:19   3:15   1: 7  
 6: 7   1:14   1:13   4:12   2:10  
 8:14                 5: 5   3: 3  
                             4:10  
                             6: 1  
                             8: 1  
```

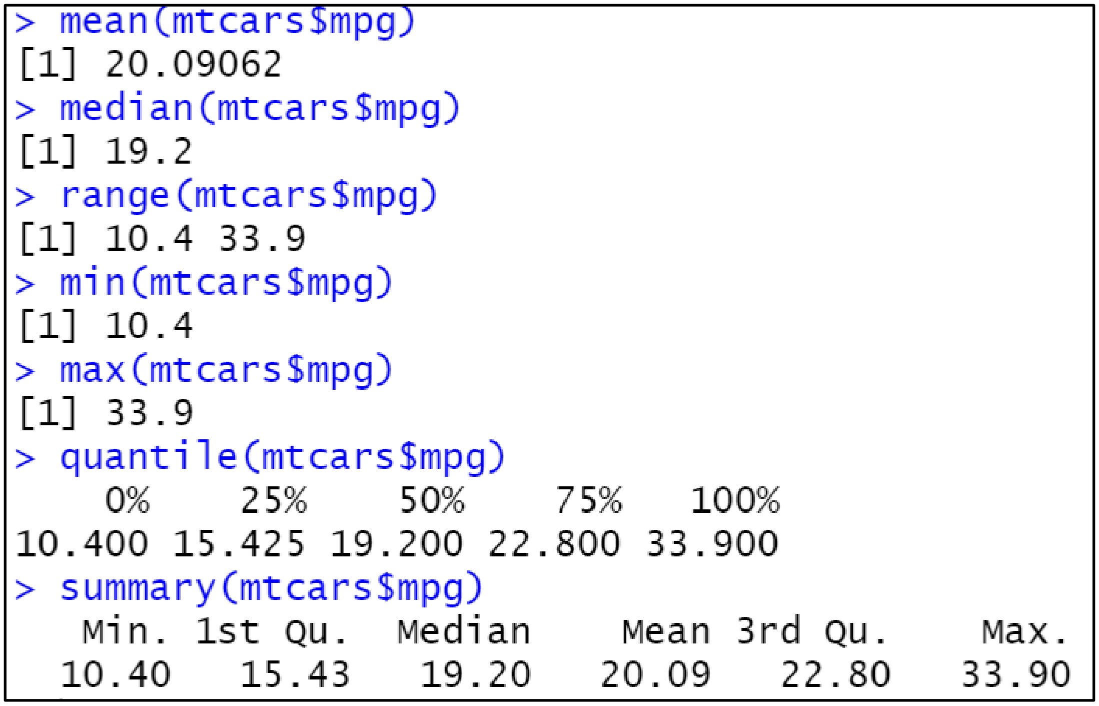
2. NUMERIC:

Command	Description
mean()	Mean value for a given list
median()	Median value for a given list
range()	Minimum and Maximum values in a list
min()	Minimum value for a given list
max()	Maximum value for a given list
quantile()	Quantile values for a given list
summary()	Complete summary of the column(s)

Quantiles divide the range of values in equal distribution. For e.g. 50% quantile represents a value such that 50 percent of the values in the given list are below this value and 50 percent of the values in the given list are above this value (which is the median)

Quantile 0% - Lowest value in the list
Quantile 50% - Median value in the list
Quantile 100% - Highest value in the list

Kindly find a video explaining in detail quantile and percentiles here.



```
> mean(mtcars$mpg)
[1] 20.09062
> median(mtcars$mpg)
[1] 19.2
> range(mtcars$mpg)
[1] 10.4 33.9
> min(mtcars$mpg)
[1] 10.4
> max(mtcars$mpg)
[1] 33.9
> quantile(mtcars$mpg)
    0%    25%    50%    75%   100% 
10.400 15.425 19.200 22.800 33.900 
> summary(mtcars$mpg)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  10.40   15.43   19.20   20.09   22.80   33.90 
```

BIVARIATE ANALYSIS

1. TWO FACTORS:

We will now use the **esoph** dataset whose description can be found here. Load the dataset using:

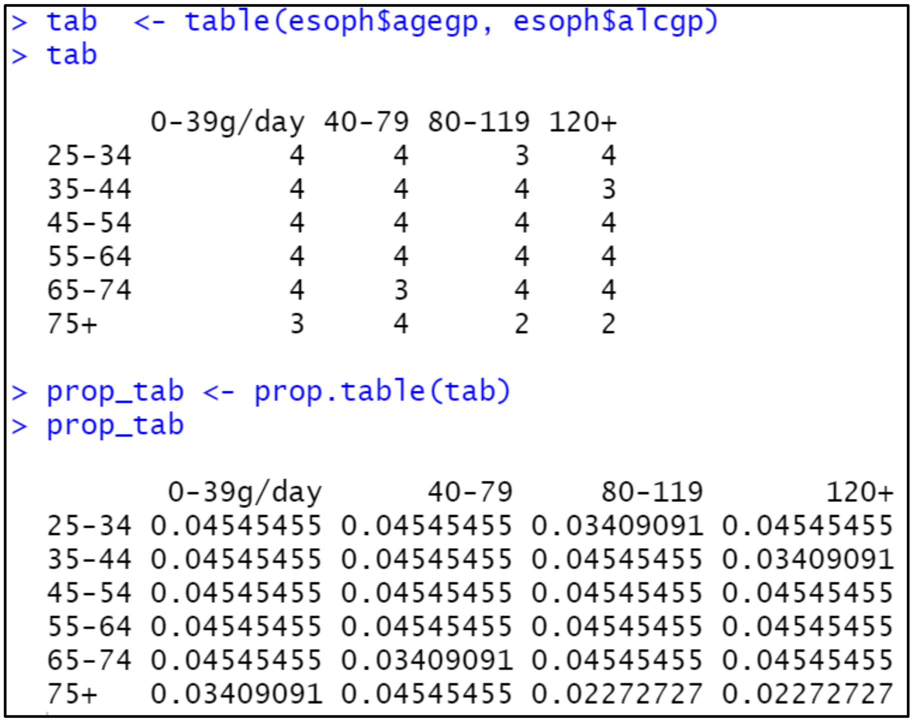
```
data(esoph)
View(esoph)
str(esoph)
```

The esoph dataset hast three factor columns:

- **agegp** – Age group
- **alcgp** – Alcohol consumption
- **tobgp** – Tobacco consumption

To compare and analyze two factors the same commands as a univariate factor analysis is used with a slight change in the arguments passed:

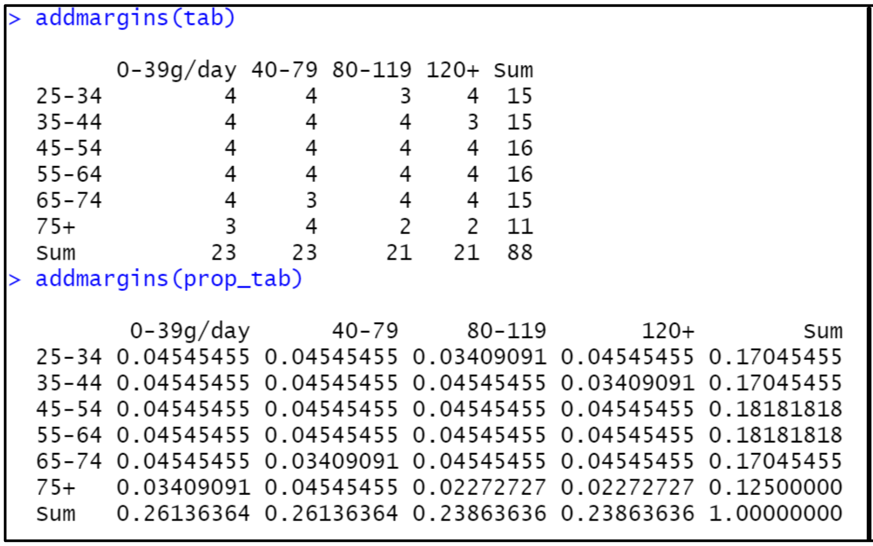
Command	Description
table(*column1*, *column2*)	Creates a cross-table with specified factor columns, displaying the counts for each combination
prop.table(*table*)	Displays a cross-table with the percentage of number of occurrences for each combination.



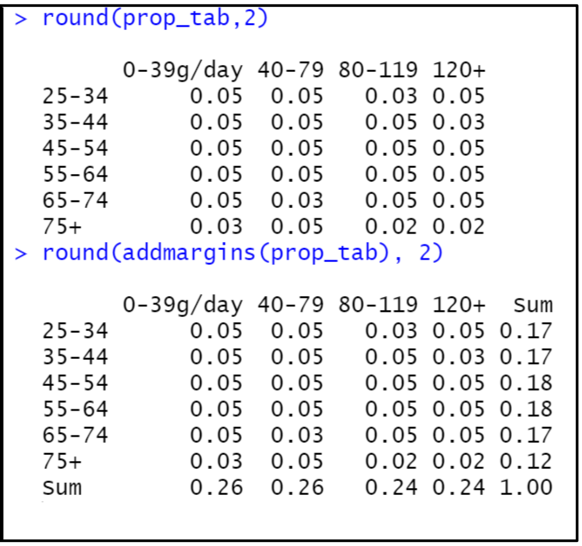
```
> tab  <- table(esoph$agegp, esoph$alcgp)
> tab

        0-39g/day 40-79 80-119 120+
  25-34         4     4      3    4
  35-44         4     4      4    3
  45-54         4     4      4    4
  55-64         4     4      4    4
  65-74         4     3      4    4
  75+           3     4      2    2

> prop_tab <- prop.table(tab)
> prop_tab

         0-39g/day      40-79     80-119       120+
  25-34 0.04545455 0.04545455 0.03409091 0.04545455
  35-44 0.04545455 0.04545455 0.04545455 0.03409091
  45-54 0.04545455 0.04545455 0.04545455 0.04545455
  55-64 0.04545455 0.04545455 0.04545455 0.04545455
  65-74 0.04545455 0.03409091 0.04545455 0.04545455
  75+   0.03409091 0.04545455 0.02272727 0.02272727
```

You can use the below two functions to perform operations on table and proportionality tables:

addmargins(*table_name*)	Adds values along rows & columns and displays under Sum
round(*table_name, decimal_places*)	Rounds the decimals to number of decimal places specified



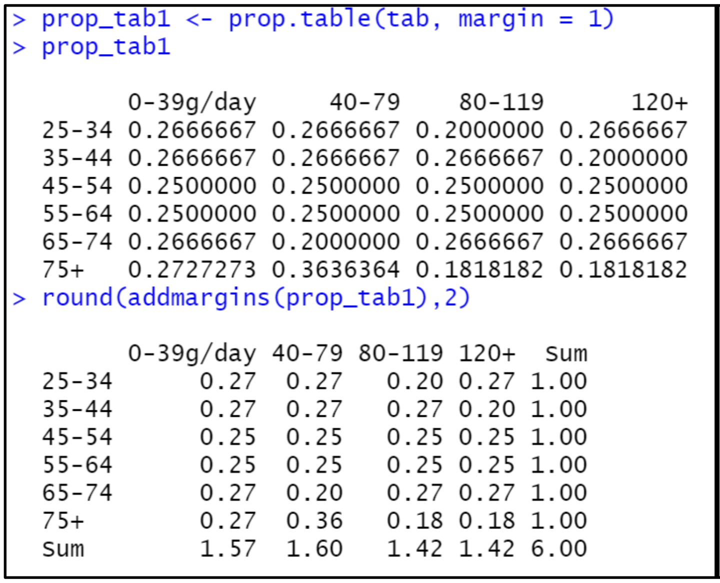
```
> addmargins(tab)

        0-39g/day 40-79 80-119 120+ Sum
  25-34         4     4      3    4  15
  35-44         4     4      4    3  15
  45-54         4     4      4    4  16
  55-64         4     4      4    4  16
  65-74         4     3      4    4  15
  75+           3     4      2    2  11
  Sum          23    23     21   21  88
> addmargins(prop_tab)

         0-39g/day      40-79     80-119       120+        sum
  25-34 0.04545455 0.04545455 0.03409091 0.04545455 0.17045455
  35-44 0.04545455 0.04545455 0.04545455 0.03409091 0.17045455
  45-54 0.04545455 0.04545455 0.04545455 0.04545455 0.18181818
  55-64 0.04545455 0.04545455 0.04545455 0.04545455 0.18181818
  65-74 0.04545455 0.03409091 0.04545455 0.04545455 0.17045455
  75+   0.03409091 0.04545455 0.02272727 0.02272727 0.12500000
  Sum   0.26136364 0.26136364 0.23863636 0.23863636 1.00000000
```



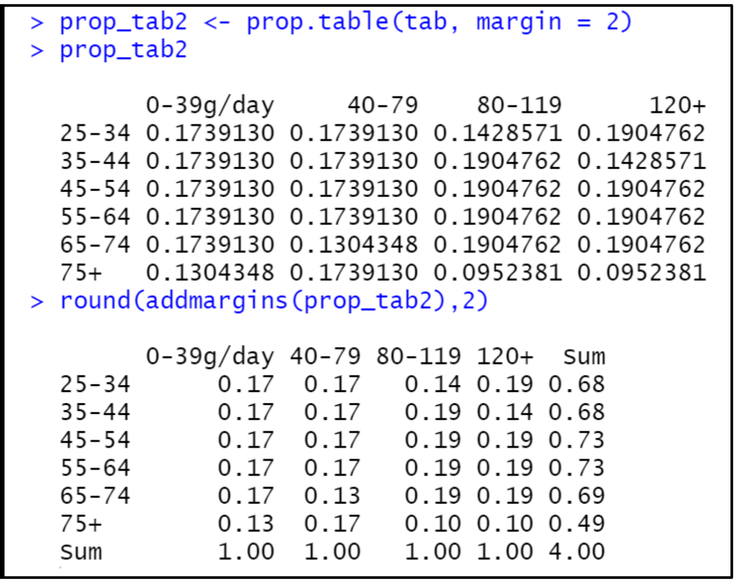
```
> round(prop_tab,2)

        0-39g/day 40-79 80-119 120+
  25-34       0.05  0.05    0.03 0.05
  35-44       0.05  0.05    0.05 0.03
  45-54       0.05  0.05    0.05 0.05
  55-64       0.05  0.05    0.05 0.05
  65-74       0.05  0.03    0.05 0.05
  75+         0.03  0.05    0.02 0.02
> round(addmargins(prop_tab), 2)

        0-39g/day 40-79 80-119 120+  sum
  25-34       0.05  0.05    0.03 0.05 0.17
  35-44       0.05  0.05    0.05 0.03 0.17
  45-54       0.05  0.05    0.05 0.05 0.18
  55-64       0.05  0.05    0.05 0.05 0.18
  65-74       0.05  0.03    0.05 0.05 0.17
  75+         0.03  0.05    0.02 0.02 0.12
  Sum         0.26  0.26    0.24 0.24 1.00
```

The **prop.table()** command has an argument "*margin*" which is used to calculate percentages along individual rows/columns

prop.table(*table*, margin=1)	Sum of each row values in the table is equal to 1
prop.table(*table*, margin=2)	Sum of each column values in the table is equal to 1



```
> prop_tab1 <- prop.table(tab, margin = 1)
> prop_tab1

       0-39g/day     40-79    80-119      120+
  25-34 0.2666667 0.2666667 0.2000000 0.2666667
  35-44 0.2666667 0.2666667 0.2666667 0.2000000
  45-54 0.2500000 0.2500000 0.2500000 0.2500000
  55-64 0.2500000 0.2500000 0.2500000 0.2500000
  65-74 0.2666667 0.2000000 0.2666667 0.2666667
  75+   0.2727273 0.3636364 0.1818182 0.1818182
> round(addmargins(prop_tab1),2)

       0-39g/day 40-79 80-119 120+  Sum
  25-34      0.27  0.27   0.20 0.27 1.00
  35-44      0.27  0.27   0.27 0.20 1.00
  45-54      0.25  0.25   0.25 0.25 1.00
  55-64      0.25  0.25   0.25 0.25 1.00
  65-74      0.27  0.20   0.27 0.27 1.00
  75+        0.27  0.36   0.18 0.18 1.00
  Sum        1.57  1.60   1.42 1.42 6.00
```



```
> prop_tab2 <- prop.table(tab, margin = 2)
> prop_tab2

       0-39g/day     40-79    80-119      120+
  25-34 0.1739130 0.1739130 0.1428571 0.1904762
  35-44 0.1739130 0.1739130 0.1904762 0.1428571
  45-54 0.1739130 0.1739130 0.1904762 0.1904762
  55-64 0.1739130 0.1739130 0.1904762 0.1904762
  65-74 0.1739130 0.1304348 0.1904762 0.1904762
  75+   0.1304348 0.1739130 0.0952381 0.0952381
> round(addmargins(prop_tab2),2)

       0-39g/day 40-79 80-119 120+  Sum
  25-34      0.17  0.17   0.14 0.19 0.68
  35-44      0.17  0.17   0.19 0.14 0.68
  45-54      0.17  0.17   0.19 0.19 0.73
  55-64      0.17  0.17   0.19 0.19 0.73
  65-74      0.17  0.13   0.19 0.19 0.69
  75+        0.13  0.17   0.10 0.10 0.49
  Sum        1.00  1.00   1.00 1.00 4.00
```

2. TWO NUMERICS:

We analyze the relationship between two numeric variables using correlation metric.

Correlation → Helps establish a relationship between the two numerical variables

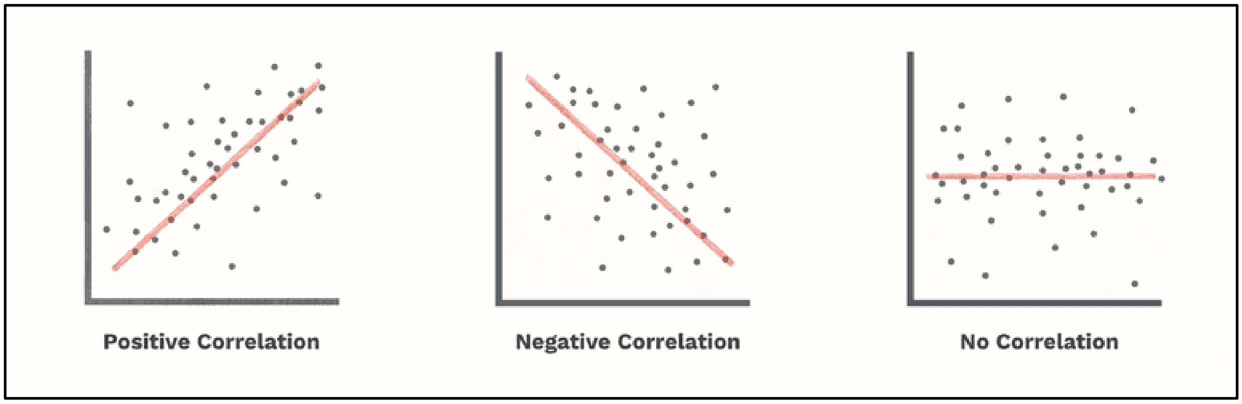
→ Has value between -1 to +1

→ Value farther from zero, more related are the two numerical variables

→ Value closer to zero, less related are the two numerical variables

→ Positive, indicates a direct relationship. Increase of one value in one variable causes an increase of value in the other variable and vice-versa

→Negative, indicates an inverse relationship. Increase of value in one variable causes a decrease of value in the other variable



Videos explaining correlation in detail – Part 1 & Part 2

Correlation is performed in R using the below command

cor(*numeric_variable1*, *numeric_variable2*)	Correlation value between the two numeric variables

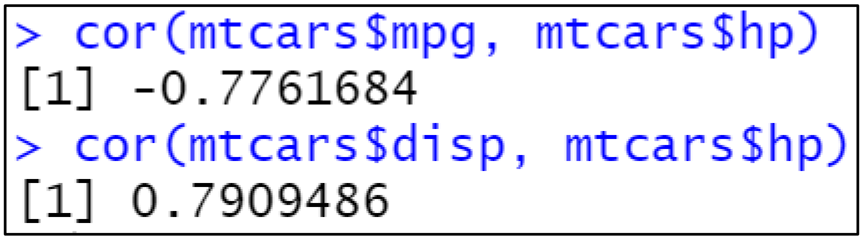
Import the **mtcars** dataset using the below commands

```
data(mtcars)
View(mtcars)
```

mtcars has a few numerical variables

- **mpg** - Miles/(US) gallon (or) Mileage
- **disp** - Displacement
- **hp** - Horsepower

Now we can run the below commands to determine the correlation relationship



```
> cor(mtcars$mpg, mtcars$hp)
[1] -0.7761684
> cor(mtcars$disp, mtcars$hp)
[1] 0.7909486
```

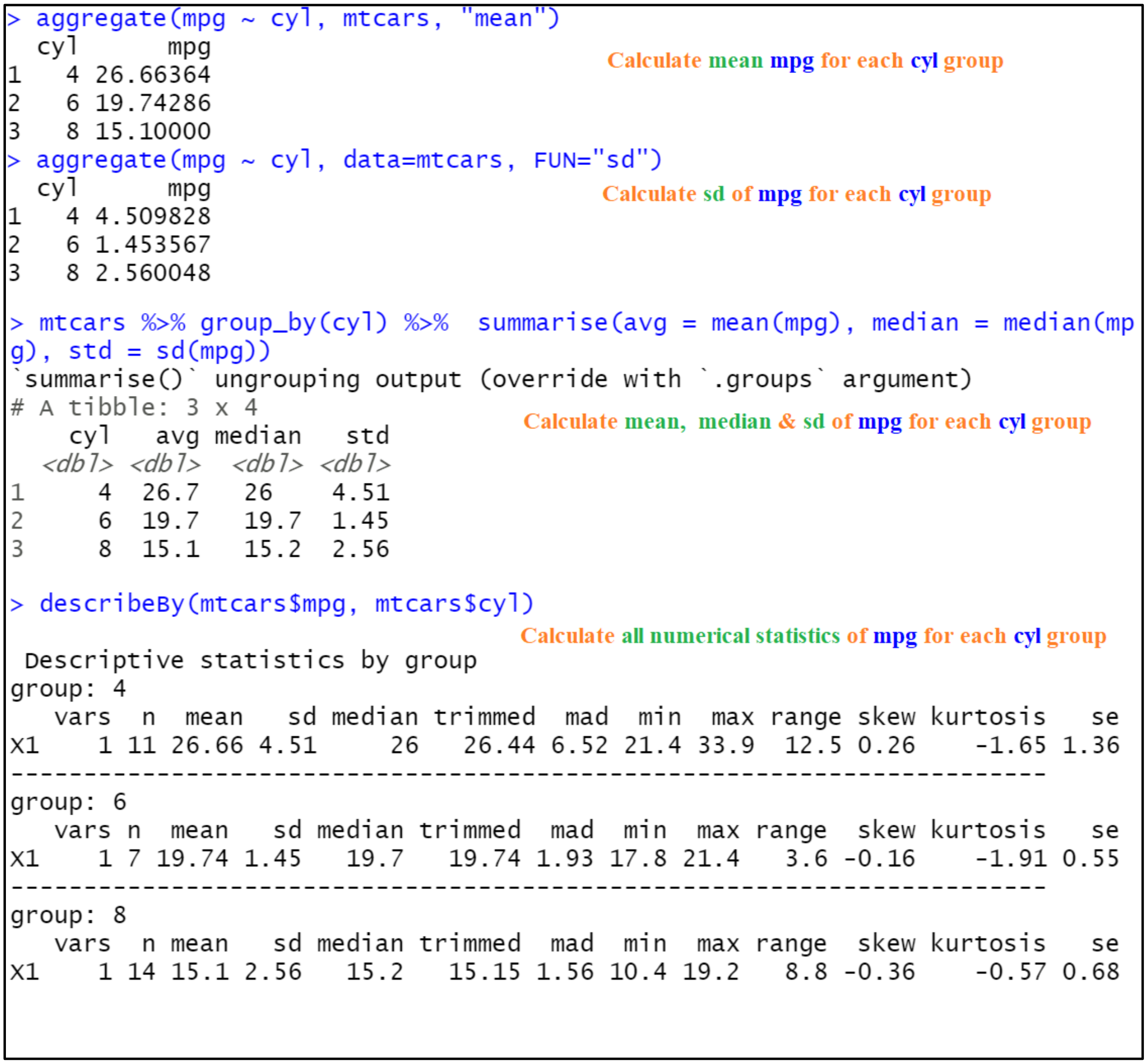
The correlation value is negative between mileage and horsepower. Thus, there is an inverse relationship between them i.e. Higher the horsepower, lower is the car mileage and vice versa.

The correlation value is positive between displacement and horsepower. Thus, there is a direct relationship between them i.e. Higher the displacement, higher is the car horsepower and same for lower values (Lower displacement indicates a lower horsepower).

3. ONE FACTOR & ONE NUMERIC:

Use below commands to analyze one factor and one numeric variable

aggregate(*Numerical_column ~ Factor_column, dataset_name, Function_to_perform*)	Perform one specific numerical analytics on each group of the factor column
summarise()	Perform multiple numerical analytics on each group of the factor column
describeBy(*Numerical_column , Factor_column*)	Perform all numerical analytics on each group of the factor column



```
> aggregate(mpg ~ cyl, mtcars, "mean")
  cyl      mpg
1   4 26.66364
2   6 19.74286
3   8 15.10000
```

Calculate mean mpg for each cyl group

```
> aggregate(mpg ~ cyl, data=mtcars, FUN="sd")
  cyl      mpg
1   4 4.509828
2   6 1.453567
3   8 2.560048
```

Calculate sd of mpg for each cyl group

```
> mtcars %>% group_by(cyl) %>%  summarise(avg = mean(mpg), median = median(mp
g), std = sd(mpg))
`summarise()` ungrouping output (override with `.groups` argument)
# A tibble: 3 x 4
    cyl   avg median   std
  <dbl> <dbl>  <dbl> <dbl>
1     4  26.7   26    4.51
2     6  19.7   19.7  1.45
3     8  15.1   15.2  2.56
```

Calculate mean, median & sd of mpg for each cyl group

```
> describeBy(mtcars$mpg, mtcars$cyl)
```

Calculate all numerical statistics of mpg for each cyl group

```
 Descriptive statistics by group
group: 4
   vars  n  mean   sd median trimmed  mad  min  max range skew kurtosis   se
X1    1 11 26.66 4.51     26   26.44 6.52 21.4 33.9  12.5 0.26    -1.65 1.36
----------------------------------------------------------------------
group: 6
   vars n  mean   sd median trimmed  mad  min  max range  skew kurtosis   se
X1    1 7 19.74 1.45   19.7   19.74 1.93 17.8 21.4   3.6 -0.16    -1.91 0.55
----------------------------------------------------------------------
group: 8
   vars  n mean   sd median trimmed  mad  min  max range  skew kurtosis   se
X1    1 14 15.1 2.56   15.2   15.15 1.56 10.4 19.2   8.8 -0.36    -0.57 0.68
```

PROBABILITY

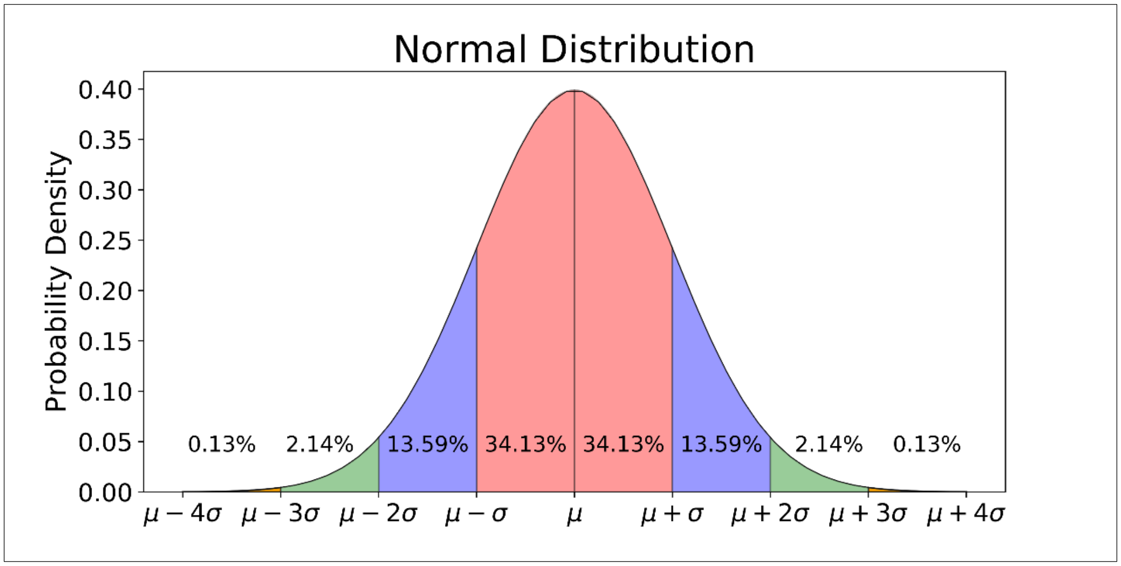
NORMAL DISTRIBUTION:

Distribution of the values creates/follows a bell curve. The distribution of the variable follows a Gaussian curve

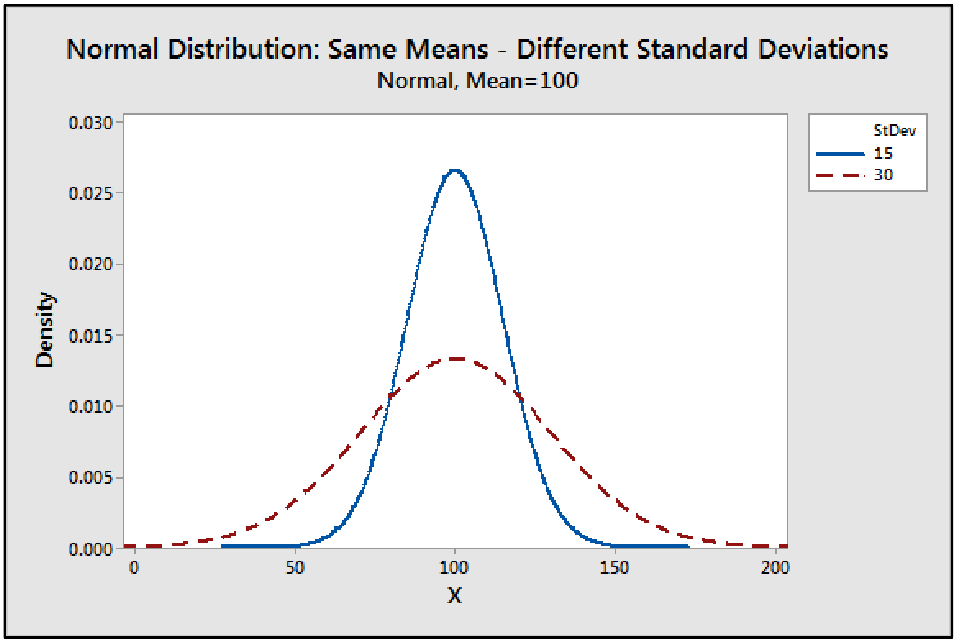
μ - Mean (Average)

Variance - Sum of square of the difference of each value from mean divided by number of values

σ – Standard Deviation (Square root of variance)



Not all normal distributions are same. They differ based on the values



To generate a normal distribution in R we can use the **rnorm()** function

Syntax: rnorm(*mean_value, sd_value, number_of_values*)

E.g.: rnorm(mean=1000, sd=250, 100)

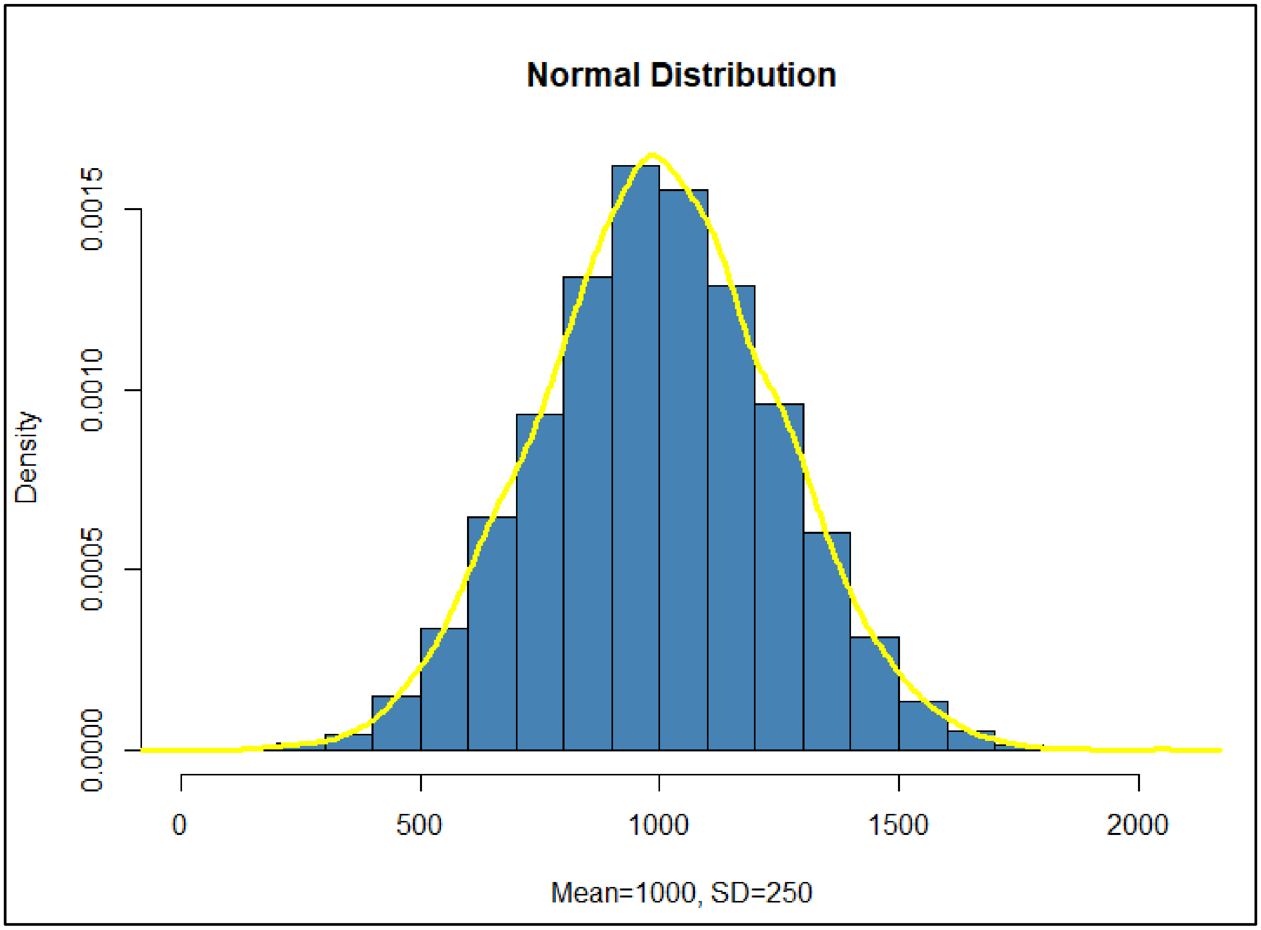
The above code will generate a list of 100 random values such that they follow a normal distribution with mean at 1000 and a standard deviation of 250

To visualize the distribution of the values try running the below set of commands

```
values <- rnorm(mean=1000, sd=250, 10000)
hist(values, main="Normal Distribution", col="steelblue", xlab="Mean=1000, SD=250", freq=F)
lines(density(values), col="yellow", lwd=3)
```

hist – Creates a histogram

lines – Draws the line describing the distribution



Z-SCORES:

Number of standard deviations the value is away from mean

$$Z = \frac{x - \mu}{\sigma}$$

Z = Z score

x = Observed value

μ = Mean

σ = Standard Deviation

For a random normal distribution with mean 100 and standard deviation (sd) of 30 the z-score for a value of 76 and 136 can be calculated as,

X = 76
Z = (76 - 100) / 30
= -0.8 SD from mean

X = 136
Z = (136 - 100) / 30
= +1.2 SD from mean

PERCENTILE:

The measure describes the percentage of values that are below the observed value in a distribution. Percentiles are different from percentage on the grounds that, percentage represents the value out of 100 but percentile indicates the percentage of values that are below the observed value.

For e.g., in a class of 100 students if a student scores 42 marks in a test out of 50,
Percentage = (42/50) X 100 = 84%

If 42 is the third highest mark it means the student has scored more marks than 97 of his/her classmates,

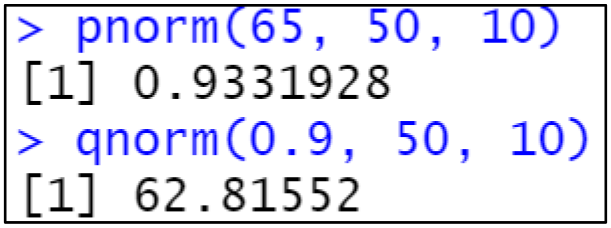
Percentile = (97/100) X 100 = 97%

Thus, the student is in the 97th percentile, indicating that 97% of the marks in the class are below the score of 42 (or) the student has scored higher than 97% of his/her classmates

Note: Percentile can never be 100. In the above example if the student was the topper, then he/she has scored better than 99 of his/her classmates and is in the 99th percentile.

Given an observed value, mean and sd you can calculate the percentile in R using **pnorm()** function. Kindly note that the result is a value between 0 and 1. To represent the percentile in terms of percentage, we will have to multiply it by 100.
Syntax: pnorm(*observed_value*, *mean*, *sd*)
E.g.: pnorm(65, 50, 10)

Similarly if you are given the percentile value, mean and sd the value can be calculated using qnorm() function.
Syntax: qnorm(*percentile_value*, *mean*, *sd*)
E.g.: qnorm(0.9, 50, 10)



```
> pnorm(65, 50, 10)
[1] 0.9331928
> qnorm(0.9, 50, 10)
[1] 62.81552
```

You can find a video explaining percentiles here.