Contents

R Introduction

Overview

R is a programming language and software environment for statistical analysis, graphics representation and reporting

Evolution of R

- Creator : Ross Ihaka and Robert Gentleman
- Created at : Dept of Statistics, University of Auckland, New Zealand
- Year : 1993
- Current team : Since mid-1997 there has been a core group (the “R Core Team”) who can modify the R source code archive
- License : free under GNU General Public License
- R is free software distributed under a GNU-style copy left, and an official part of the GNU project called GNU S.
- Old Name of R : S programming
- S programming language : S is a statistical programming language developed primarily by John Chambers and Rick Becker and Allan Wilks of Bell Laboratories.
- S built year : 1976
- S now implemented as R
- Read more on

https://en.wikipedia.org/wiki/R_(programming_language) (https://en.wikipedia.org/wiki/R_(programming_language))

https://en.wikipedia.org/wiki/S_(programming_language) (https://en.wikipedia.org/wiki/S_(programming_language))

https://www.r-project.org/about.html (https://www.r-project.org/about.html)

Major Features

- core R is interpreted language
- allows use of functions , branching , looping, and input and output facilities.
- allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.
- has an effective data handling and storage facility,

- provides a suite of operators for direct calculations on arrays, lists, vectors and matrices (a.k.a. vectorised operations).
- provides a large, coherent and integrated collection of tools for data analysis
- provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

Why R ? or Why to choose R programming ?

Strengths of R programming

- R has large number of packages for statistical analysis and data science
- R can be used for complex statistical modelling
- R Facilitates interaction with databases
- For statistical minded person, R is Simple and easy to understand

Limitations of R

- limited community support
- R can’t be used to create a complete end to end software

Major Libraries (Packages) in R

- R base
- tidy verse
- GGPlot2
- tidyr
- dplyr
- knitr
- caret

Installation of R

Windows Installation

- Install R base

You can download the latest Windows installer version of R from https://cran.r-project.org/bin/windows/base/ (https://cran.r-project.org/bin/windows/base/)

As it is a Windows installer (.exe) with a name “R-version-win.exe”. You can just double click and run the installer accepting the default settings. If your Windows is 32-bit version, it installs the 32-bit version. But if your windows is 64-bit, then it installs both the 32-bit and 64-bit versions. After installation you can run it After running you will get R console with basic GUI

- Install R studio

** NOTE: Install R Studio ONLY AFTER R base is installed and running

You can download R studio Desktop version from https://posit.co/products/open-source/rstudio/ (https://posit.co/products/open-source/rstudio/) and Install it.

This is a GUI tool for writing R scripts (.R) and R markmown (.Rmd) files.

R Command Prompt

Open R base (using shortcut on desktop)

It will give you command prompt to perform R operations

You can write R commands and execute them here

For example:

```
m = “Hello in R programing!”
print(m)
```

[1] “Hello in R programing!”

R Script File (.R)

Script file is collection of commands in R to run them together.

Script helps to save all the work at one place as .R file.

Comments in R Script

Single comment is written using # in the beginning of the statement

Example : # This is first comment

they are ignored by the interpreter while executing your actual script

R markdown File (.Rmd) or R notebook

This is latest type of file supported by R

R Markdown files are also known as notebook in R

It allows writing notes, explaintions along with the codes.

Markdown files have option to insert code snipets.

Output of code snipets is visible within the markdown file

Markdown file can be directly converted to HTML or PDF files

For example these notes are generated using markdown files ! :)

R Gui

When you open R base (using shortcut on desktop), R gui is launched

It gives you facility - to run the commands one by one - to run complete R script - to run part of R script - to save script - to print the script - to load/open “workspace” (workspace is any directory in your system)

R Studio

Specialised GUI to work with R scripts and Markdown files

Gives 4 panes : Source pane, Environment Pane, Console Pane and Files Pane

Read more at : https://rladiessydney.org/courses/ryouwithme/01-basicbasics-1/ (https://rladiessydney.org/courses/ryouwithme/01-basicbasics-1/)

Errors, Warnings and Messages

R reports errors, warnings, and messages in unique way.

R reports ALL a glaring red font, which makes it seem like it is scolding you.

However, seeing red text in the console is not always bad.

- Errors: Errors start with message “Error in …” When error comes script will stop running and will not complete the operation
- Warnings: Wornings start with message “Worning:” In this case script will continue running and complete the operation
- Messages: When red text is not having Error / Worning , then it is message from R There is no problem this is just a message

R Basics

2023-03-01

Different ways to Print

These are different ways to print

print function

used to print single input

print automatically adds new line ‘’ at end of the input

```
print("Hello")
```

```
## [1] "Hello"
```

```
print("World!")
```

```
## [1] "World!"
```

cat function

cat concatenate multiple objects (representation) , strings by using space as a separator

created full string is given to console

cat doesn’t add newline at the end

cat doesn’t return anything (returns NULL)

cat uses much less conversions than print

```
cat("He","llo") # 'he' and 'llo' will be concatenated with space as separator
```

```
## He llo
```

```
cat("World!") # next cat command will directly send 'World!' to console without any separator
```

```
## World!
```

paste function

Concatenate vectors after converting to character

paste returns the concatenated character vector

return value can be assigned to a variable else it is printed using print function

if return value is printed then print adds new line at the end

_

```
paste("He","llo") # 'he' and 'llo' will be concatenated with space as separator and character vector is returned. This returned vector object is printed to console using print. This adds '\n' at the end of the line.
```

```
## [1] "He llo"
```

```
paste("World!") # next paste command will create another character vector 'World!' and return it. This returned character vector is printed on console using print
```

```
## [1] "World!"
```

Use variable name directly

Only writing variable name on line in script will print current value of that variable on console

```
var1 = "Hello World!"
var1
```

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

Accept user input

readline function

Works in interactive mode ONLY

Accepts input from user and returns a character vector

If user wants to read integer/numeric then returned value needs to be converted to integer / numeric

```
yourname <- readline(prompt="What is your name? ") # Assume no input is given by the user then empty character vector is returned
```

```
## What is your name?
```

```
print(yourname) # empty character vector is printed
```

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

```
print(class(yourname)) # class of 'yourname' is 'character'
```

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

Working Directory of R

Directory which currently R studio is working on

R studio take this as default location to access files or store files

How to check current working directory ?

```
getwd()
```

```
## [1] "I:/R_practice/R_programming_notes"
```

How to change working directory ?

setwd function is used to set new directory

setwd is given path

path can use double back-slash “\ \” or single forward-slash “/”

preferably directory should not have name with spaces

NOTE : setwd doesn’t work well inside a R markdown or notebook

```
# path can use \\ or /
setwd("./inside_folder")
```

Install R packages

You can use commands or use GUI options to install any package

Install packages using commands

install.packages command helps install given package

Package name should be given in double quotes or single quotes

```
# install.packages('tidyverse')
```

Install packages using R Studio GUI

In R studio, main menu displayed on the top

Select Tools & click it

In Tools -> click “Install Packages” option

There will be a pop-up window

Here type the name of the package to be installed

Then package will be installed

Load packages in R

Here we load the packages in memory

Without loading packages, they can’t be used

While starting R, few base packages are pre loaded

```
#library(tidyverse)
```

R Decision Making

if , if - else, if - else if - else

if : An if statement consists of a Boolean expression followed by one or more statements.

if else : An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

```
x <- c("what","is","truth")
if("Truth" %in% x){
  print("Truth is found")
} else {
  print("Truth is not found")
}
```

```
## [1] "Truth is not found"
```

if…else if…else :

An if statement can be followed by an optional else if…else statement, which is very useful to test various conditions using single if…else if statement.

NOTE::

When using if, else if, else statements there are few points to keep in mind. • An if can have zero or one else and it must come after any else if's. • An if can have zero to many else if's and they must come before the else. • Once an else if succeeds, none of the remaining else if's or else's will be tested.

```
x <- c("what","is","truth")
if("Truth" %in% x){
print("Truth is found the first time")
} else if ("truth" %in% x) {
print("truth is found the second time")
} else {
print("No truth found")
}
```

```
## [1] "truth is found the second time"
```

Switch case

A switch statement allows a variable to be tested for equality against a list of values.

Syntax of switch is

switch(expression, case1, case2, case3, …)

```
choice = 2
x <- switch(choice,
"First",
"second",
"third",
"fourth"
)
print(x)
```

```
## [1] "second"
```

```
v = c(1,3)
x <- switch(length(v),
"First",
"second",
"third",
"fourth"
)
print(x)
```

```
## [1] "second"
```

R_Variables

2023-03-04

variables in R

Naming convension

A variable provides us with named storage that our programs can manipulate.

A variable in R can store an atomic vector, group of atomic vectors or a combination of many R-objects.

A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Valid variable names : var1, var, var_name2. , .var_name , var.name

In-Valid variable names : .1var, %var, _var_name2

Variable Assignment

The variables can be assigned values using leftward (<-), rightward (->) and equal to (=) operator.

Also following all operators act as assignment in R programming

<-, =, <<- :: left assignment

->, ->> :: right assignment

All following lines work in R programming

```
v1 = 10
v2 <- 20
30 -> v3
```

Printing a variable

By Name of the variable

Just writing name of variable and nothing else on single line in script will print the variable value

```
v1
```

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

By print()

print can be used to print a single variable at a time

```
print(v2)
```

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

By cat()

cat function allows to print multiple variables at a time

```
cat(v1,v2,v3)
```

```
## 10 20 30
```

Finding Variables

To know all the variables currently available in the workspace we use the ls() function. Also the ls() function can use patterns to match the variable names.

```
print(ls())
```

```
## [1] "v1" "v2" "v3"
```

pattern finding in ls()

The ls() function can use patterns to match the variable names.

```
# List the variables starting with the pattern "v".
print(ls(pattern="v"))
```

```
## [1] "v1" "v2" "v3"
```

Deleting Variables

Variables can be deleted by using the rm() function. Below we delete the variable v3. On printing the value of the variable error is thrown.

```
rm(v3)

#print(v3)
# Print gives error : Error in print(v3) : object 'v3' not found
```

Deleting all variables in environment

It can be done using rm and ls

This is used in beginning of any R script to make sure that no old variables affect the execution of the script

Following line deletes all environment variables

```
rm(list=ls())
print(ls())
```

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

Controlling Visibility of Variables

This is way to use what variables are shown in current environment.

```
rm(list = ls())
x=10

print("List after creating x")
```

```
## [1] "List after creating x"
```

```
ls()
```

```
## [1] "x"
```

```
.xyz= 90

print("List after creating .xyz")
```

```
## [1] "List after creating .xyz"
```

```
ls()
```

```
## [1] "x"
```

```
print("List of all variables")
```

```
## [1] "List of all variables"
```

```
ls(all.names = TRUE)
```

```
## [1] ".xyz" "x"
```

R Operators

Types of Operators 1 Arithmetic Operators 2 Relational Operators 3 Logical Operators 4 Assignment Operators 5 Miscellaneous Operators

Arithmetic Operators

'+' Adds two vectors

'-' Subtracts second vector from the first

'*' Multiplies both vectors elementwise

'/' Divide the first vector elements with the second vector elements elementwise

'%%' Give the remainder of the first vector with the second elementwise

'%/%' Quotient (integer part) of division of first vector with second elementwise

'^' The first vector elements raised to the exponent of second vector elements elementwise

```
x = c(15,22)
y = c(3,5)
x
```

```
## [1] 15 22
```

```
y
```

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

```
cat("\n Addition :",x+y)
```

```
##
##  Addition : 18 27
```

```
cat("\n Substraction :",x-y)
```

```
##
##  Substraction : 12 17
```

```
cat("\n Multiplication :", x*y)
```

```
##
##  Multiplication : 45 110
```

```
cat("\n Division :", x/y)
```

```
##
##  Division : 5 4.4
```

```
cat("\n Mod (remainder):",x%%y)
```

```
##
##  Mod (remainder): 0 2
```

```
cat("\n Quotient ( integer part) of division",x%/%y)
```

```
##
##  Quotient ( integer part) of division 5 4
```

```
cat("\n Power operation:",x^y)
```

```
##
##  Power operation: 3375 5153632
```

Relational Operators

Following list shows the relational operators supported by R language.

Working: Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

1. '<' elementwise check if first vector element is less than second vector element and return TRUE or FALSE for every element
2. '>' elementwise check if first vector element is greater than second vector element and return TRUE or FALSE for every element
3. '==' elementwise check if first vector element is equal to second vector element and return TRUE or FALSE for every element
4. '!=' elementwise check if first vector element is not equal to second vector element and return TRUE or FALSE for every element
5. '>=' elementwise check if first vector element is greater than equal to second vector element and return TRUE or FALSE for every element
6. '<=' elementwise check if first vector element is less than equal to second vector element and return TRUE or FALSE for every element

```
x = c(15,22,35,100)
y = c(3,5,56,100)
cat("\n Less than ::",x<y)
```

```
##
##  Less than :: FALSE FALSE TRUE FALSE
```

```
cat("\n Greater than::",x>y)
```

```
##
##  Greater than:: TRUE TRUE FALSE FALSE
```

```
cat("\n Equal ::", x==y)
```

```
##
##  Equal :: FALSE FALSE FALSE TRUE
```

```
cat("\n Not equal ::", x!=y)
```

```
##
##  Not equal :: TRUE TRUE TRUE FALSE
```

```
cat("\n Greater than equal::",x>=y)
```

```
##
##  Greater than equal:: TRUE TRUE FALSE TRUE
```

```
cat("\n Less than equal ::",x<=y)
```

```
##
##  Less than equal :: FALSE FALSE TRUE TRUE
```

Logical Operators

Following list shows the logical operators supported by R language. It is applicable only to vectors of type logical, numeric or complex. All numbers greater than 1 are considered as logical value TRUE.

Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Element wise operators :: &, | , ! (Compares all elements one by one)

Non Zero values are TRUE [Negative nos are TRUE and positive nos are TRUE]

Zero is FALSE

```
v <- c(3,0,56.23,2+2i)
t <- c(0,3,TRUE,2+3i)
z= c(TRUE,FALSE,TRUE,FALSE)
v
```

```
## [1]  3.00+0i  0.00+0i 56.23+0i  2.00+2i
```

```
t
```

```
## [1] 0+0i 3+0i 1+0i 2+3i
```

```
z
```

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

```
print(v&t)
```

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

```
print(v&z)
```

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

```
print(v|t)
```

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

```
print(v|z)
```

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

Special Logical Operators &&, || ONLY Compares only first element of vectors and return TRUE or FALSE

```
print(v&&t)
```

```
## Warning in v && t: 'length(x) = 4 > 1' in coercion to 'logical(1)'

## Warning in v && t: 'length(x) = 4 > 1' in coercion to 'logical(1)'
```

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

```
print(v||t)
```

```
## Warning in v || t: 'length(x) = 4 > 1' in coercion to 'logical(1)'
```

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

```
print(v&&z)
```

```
## Warning in v && z: 'length(x) = 4 > 1' in coercion to 'logical(1)'
```

```
## Warning in v && z: 'length(x) = 4 > 1' in coercion to 'logical(1)'
```

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

```
print(v||z)
```

```
## Warning in v || z: 'length(x) = 4 > 1' in coercion to 'logical(1)'
```

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

Working of logical operators on atomic vectors

Here both & and && also | and || return the same results

Non Zero values are TRUE [Negative nos are TRUE and positive nos are TRUE]

Zero is FALSE

```
x=10
y=0
print(x&y)
```

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

```
print(x&&y)
```

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

```
print(x|y)
```

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

```
print(x||y)
```

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

Assignment Operators <-, =, <<- :: left assignment ->, ->> :: right assignment

```
x = 10
y<-20
z<<-24

34-> n
35 ->> b
cat(x,y,z,n,b)
```

```
## 10 20 24 34 35
```

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logical computation.

Colon operator
operator = Colon operator. It creates the series of numbers in sequence and returns a vector. Each time step is +1 or -1 based on start and end values.

```
v1 = 2:8
v1
```

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

```
v4 = 10:5
v4
```

```
## [1] 10  9  8  7  6  5
```

```
v3 = -3:-10
v3
```

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

```
v4 = -23:-15
v4
```

```
## [1] -23 -22 -21 -20 -19 -18 -17 -16 -15
```

```
v5 = 11.5:15.7
v5
```

```
## [1] 11.5 12.5 13.5 14.5 15.5
```

%in% operator

%in% = This operator is used to identify if an element belongs to a vector. %in% returns a logical vector length same as the first vector, with a TRUE where that element can be found in the second argument and a FALSE where that element can not be found in second vector.

.

```
v1 <- 8
v2 <- 15
x=c(8,15)
t <- c(9,12,45,8,67,5,8)
print(v1 %in% t)
```

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

```
print(v2 %in% t)
```

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

```
print(t %in% v1)
```

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

```
print(t %in% v2)
```

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

```
x=c(8,15)
t <- c(9,12,15,8,67,5,8)
print(t %in% x)
```

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

```
print(which(t %in% x))
```

```
## [1] 3 4 7
```

%*% operator

%*% = This operator is used to get dot product of two matrices

```
M = matrix( c(2,6,5,1,10,4), nrow=2,ncol=3)
M1 = matrix( c(20,60,50,1,10,4), nrow=3,ncol=2)
print(M)
```

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

```
ans = M %*% M1 #size 2 by 2
print(ans)
```

```
##      [,1] [,2]
## [1,]  840   92
## [2,]  380   32
```

R Data Types Part 1

Data types of Variables in R

variables in any programming language are used to store information in memory

Unlike other programming languages R variables are R-objects.

Types of R-Objects in R programming:

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

Simplest R-objects are vector objects.

There are 6 types of Atomic Vectors or Basic Vectors. (NOTE: There are basic types there are more classes in R)

atomic Vector types

Logical Integer Numeric Complex Character Raw

In R, type, mode and storage mode of a R-Object may be different. So, following is details for each types of atomic vectors.

typeof	mode	storage.mode
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character
raw	raw	raw

Details for each types of atomic vectors

Create Integer type atomic vectors

Any whole number appended with L will be treated as integer by R Following is command to see all details of an integer variable Details include class, typesof, mode, and storage.mode

```
x=100000L
cat(x,class(x), typeof(x), mode(x), storage.mode(x))
```

```
## 100000 integer integer numeric integer
```

Create Numeric type atomic vectors

Any decimal point number or number (without L) will be treated as Numeric by R Following are commands to see all details of an Numeric variable Details include class, typesof, mode, and storage.mode

Numeric when number without suffix L

```
x=10
cat(x,class(x), typeof(x), mode(x), storage.mode(x))
```

```
## 10 numeric double numeric double
```

Numeric when number is decimal

```
x=10.67
cat(x,class(x), typeof(x), mode(x), storage.mode(x))
```

```
## 10.67 numeric double numeric double
```

Create Logical type atomic vectors

TRUE or FALSE will be treated as Logical by R Following are commands to see all details of an Logical variable Details include class, typesof, mode, and storage.mode

```
x=TRUE
cat(x,class(x), typeof(x), mode(x), storage.mode(x))
```

```
## TRUE logical logical logical logical
```

Create Complex type atomic vectors

Any number in form of a+bi will be treated as Complex by R Following are commands to see all details of an Complex variable Details include class, typesof, mode, and storage.mode

```
x=5+4i
cat(x,class(x), typeof(x), mode(x), storage.mode(x))
```

```
## 5+4i complex complex complex complex
```

Create Character type atomic vectors

Any single character or string will be treated as Character by R. R will give same type even if chracters are in single quotes or double quotes. Following are commands to see all details of an Character variable. Details include class, typesof, mode, and storage.mode

```
x="Hi"
cat(x,class(x), typeof(x), mode(x), storage.mode(x))
```

```
## Hi character character character character
```

Create Raw type atomic vectors

Any single character or string will be treated as Character by R. R will give same type even if chracters are in single quotes or double quotes. Following are commands to see all details of an Character variable. Details include class, typesof, mode, and storage.mode

```
x=charToRaw("Hello")
cat("bytes in Hello are :",x," details:",class(x), typeof(x), mode(x), storage.mode(x))
```

```
## bytes in Hello are : 48 65 6c 6c 6f  details: raw raw raw raw
```

Vectors

Using c() function to create vectors

In R we can use c() function also known as combine. It creates a vector

```
x_dim1 = c(10,20,30,40,50,60)
print(x_dim1)
```

```
## [1] 10 20 30 40 50 60
```

```
class(x_dim1)
```

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

Data type assignment after combine

Data types of returned value is based on highest hierarchy in all input values

Hierarchy is Logical < Integer < Numeric < Complex < Character

See output of following codes and note how the class / data types are assigned to vector

```
x_complex = c(3,0,TRUE,2+2i)
print(x_complex)
```

```
## [1] 3+0i 0+0i 1+0i 2+2i
```

```
class(x_complex)
```

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

```
x_char = c(0,"IACSD R Prog",TRUE,2+2i)
print(x_char)
```

```
## [1] "0"            "IACSD R Prog" "TRUE"         "2+2i"
```

```
class(x_char)
```

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

Using vector function to create vectors

vector function creates an empty vector of given data type and size

Following code creates vector of numeric data type of size 5

```
#create vector of numeric data type of size 5
x_vec = vector("numeric", 5)
print(x_vec)
```

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

```
#create vector of complex data type of size 5
x_vec = vector("complex", 5)
print(x_vec)
```

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

```
#create vector of logical data type of size 5
x_vec = vector("logical", 5)
print(x_vec)
```

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

```
#create vector of character data type of size 5
x_vec = vector("character", 5)
print(x_vec)
```

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

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
x <- list(1, "a", c(1,2,3), 1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [1] 1+4i
```

list used just like key value pair

Every element of list can be assigned key and value

So, value can be accessed in format list$key

```
xlist <- list(a = "Shantanu Pathak", b = 1:10)
xlist
```

```
## $a
## [1] "Shantanu Pathak"
##
## $b
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Matrix

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function

Matrix can be created by specifying number of rows and columns. “byrow = TRUE” means elements will be filled in row wise manner.

```
M = matrix( c('a','a','b','c','b','a'), nrow=2,ncol=3,byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "b"
## [2,] "c"  "b"  "a"
```

ARRAY

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
x_dim2 = array(c(10,20,30,40,50,60,70),dim=c(2,3))
print(x_dim2)
```

```
##      [,1] [,2] [,3]
## [1,]   10   30   50
## [2,]   20   40   60
```

Factors

factor- type vector contains a set of numeric codes with character-valued levels.

Regardless of the levels/labels of the factor, the numeric storage is an integer with 1 corresponding to the first level (in alph-order)

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

```
students = factor(c(100,0,100,0,0,0), levels = c(0, 100), labels = c("boy", "girl"))
students
```

```
## [1] girl boy  girl boy  boy  boy
## Levels: boy girl
```

Ordered factors

Ordinal variables are created using order=TRUE in factors

When we print the ordinal variable (ordered factor) in levels we can see the order mentioned

In following example Manager is assigned lowest level, the team lead ans then SME. This order is with respect to alphabetical order of values.

```
designation <- factor(c("SME", "Team Lead","Manager"), ordered =TRUE)
designation
```

```
## [1] SME       Team Lead Manager
## Levels: Manager < SME < Team Lead
```

DataFrame

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the data.frame() function.

```
M1 = matrix(c(1,2,3,4),nrow=2)
#M1
df <- as.data.frame(M1)
df
```

	V1 <dbl>	**V2** <dbl>
	1	3
	2	4

2 rows

R_lists_factors

2023-03-06

List Operations

Create a list

```
x <- list(1, "a", TRUE, 1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

Create empty list

```
x <- vector("list", length = 5)
x
```

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

```
length(x)
```

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

```
class(x)
```

```
## [1] "list"
```

Access element of list and element within element

```
x <- list(1, "a", c(1,2,3), 1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [1] 1+4i
```

```
# Complete second element from list
x[2]
```

```
## [[1]]
## [1] "a"
```

```
# Value of the second element
x[[2]]
```

```
## [1] "a"
```

```
x <- list(1, "a", c(1,2,3), 1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [1] 1+4i
```

```
x[[3]][3] # within 3rd element access 3rd sub element
```

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

```
x[[3]][-2] # within 3rd element access all elements except 2nd element(-2)
```

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

Modify element in list

```
x <- list(1, "a", c(1,2,3), 1+4i)
x[1] = 3
x[[3]] = "name" #c(1,3,5)
x
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] "name"
##
## [[4]]
## [1] 1+4i
```

Create array of lists

```
x= list(1,"kk")
y = list(c(1,2,3),list(78,78),2+67i,3,"lll")
a = array(c(x,y))
class(a)
```

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

```
a
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "kk"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [[4]][[1]]
## [1] 78
##
## [[4]][[2]]
## [1] 78
##
##
## [[5]]
## [1] 2+67i
##
## [[6]]
## [1] 3
##
## [[7]]
## [1] "lll"
```

Convert vector to list using as.list()

```
x <- 1:10 # range operator special behavior as dtypes integer
class(x)
```

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

```
x <- as.list(x)
class(x)
```

```
## [1] "list"
```

List as key value pair

```
# list is just like key value pair
xlist <- list(a = "Shantanu Pathak", b = 1:10)#, data = head(iris))
xlist
```

```
## $a
## [1] "Shantanu Pathak"
##
## $b
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Factors

factor- type vector contains a set of numeric codes with character-valued levels.

Regardless of the levels/labels of the factor, the numeric storage is an integer with 1 corresponding to the first level (in alph-order)

```
students = factor(c(100,0,100,0,0,0), levels = c(0, 100), labels = c("boy", "girl"))
students
```

```
## [1] girl boy  girl boy  boy  boy
## Levels: boy girl
```

```
grades = factor(c("A","B","A+","A","B","B","A","A+"))
grades
```

```
## [1] A  B  A+ A  B  B  A  A+
## Levels: A A+ B
```

```
nlevels(grades)
```

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

Convert factor to numeric

```
as.numeric(students)
```

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

```
 1 + as.numeric(students)
```

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

Ordered Factor (Ordinal Variables)

Alphabetical Order

```
designation <- factor(c("Manager", "Team Lead","SME"), ordered =TRUE)
designation
```

```
## [1] Manager   Team Lead SME
## Levels: Manager < SME < Team Lead
```

User given Order

User can provide the order of values using levels option.

In levels, first value is lowest level and then in increasing way other values are there.

In following example levels = c(“SME”, “Team Lead”,“Manager”). So, SME is lowest then Team lead and then top level is manager.

Same order can be seen when we convert the factor to numeric, manager is assigned highest numeric value 3 then team lead 2 and then SME is 1 smallest value.

```
designation <- factor(c("Manager", "Team Lead","SME"), ordered =TRUE, levels = c("SME", "Team
Lead","Manager"))
designation
```

```
## [1] Manager   Team Lead SME
## Levels: SME < Team Lead < Manager
```

```
as.numeric(designation)
```

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

```
val<-factor(c("r1","r2","r1","r2","r2","r3","r1"),ordered = TRUE,levels = c("r3", "r2","r1"))
val
```

```
## [1] r1 r2 r1 r2 r2 r3 r1
## Levels: r3 < r2 < r1
```

```
as.numeric(val)
```

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

R Data Types Part 2

Code

Hide

```
x_arr = c(1, "a", TRUE, 1+4i)
x_arr
x <- list(1, "a", TRUE, 1+4i)
x
```

Hide

```
x <- vector("list", length = 5) # empty list
x
length(x)
class(x)
```

Hide

```
x <- 1:10 # range operator special behavior as dtypes integer
class(x)
x <- as.list(x)
class(x)
```

Creating Sequences by ':' operator

Hide

```
print(3:12)
print(8.5:4.5)
print(-12:1)
print(c(1, 1:3, c(5, 8), 13))
```

Hide

```
print(10.45:5)
a = c(10:20)
a
```

Hide

```
print(c(1, 1:3, c(5, 8), 13))
```

Hide

```
x_dim2 = array(1:9,dim=c(3,3))
print(x_dim2)
```

Creating Sequences by seq class

Similar to range function in python

Hide

```
print(seq.int(3, 12))
print(seq.int(3, 12, 0.5))
print(seq.int(0.1, 0.01, -0.01))
print(seq.int(2,100,2))
print(seq.int(99,90,-2))
```

Hide

```
#print(seq.int(10,20,-1))
```

Sequence by seq_len function

It takes ‘n’ as parameter. And creates sequence from 1:n

Hide

```
print(seq_len(10))
```

Sequence by seq_along function

It is useful when we want to create sequence of length of input array

Hide

```
v = c(45,56,66,34,23,100,450)
print(v)
seq_along(v)
print(seq_len(length(v)))
```

Creating Vectors by Repeatition using rep() function

Hide

```
rep(1:5, 3)
rep(1:5, each = 3)
rep(1:5, times = 1:5)
rep(1:5, length.out = 7)
rep.int(1:5, 3)
rep_len(1:5, 7)
```

Hide

```
rep(1:5, times = c(5,3,8,1,1))
# first 5 times ,, 2nd 3 times , 3rd 8 times, 4th 1 , 5 th 1
```

R Matrix Operations

Matrix

Matrix can be created by specifying number of rows and columns. “byrow = TRUE” means elements will be filled in row wise manner.

```
M = matrix( c('a','a','b','c','b','a'), nrow=2,ncol=3,byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "b"
## [2,] "c"  "b"  "a"
```

If “byrow = TRUE” is not gievn then elements will be filled in column wise manner. In this example elements 3 to 14 are arranged in 4 rows.

```
M <- matrix(c(1:12), nrow = 4)
print(M)
```

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

```
M <- matrix(c(1:14), nrow = 4)
```

```
## Warning in matrix(c(1:14), nrow = 4): data length [14] is not a sub-multiple or
## multiple of the number of rows [4]
```

```
print(M)
```

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

In case rows or columns are not specified then, matrix with single column and no of rows equal to number of elements is created.

```
M <- matrix(c(3:14))
print(M)
```

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

Accessing elements of a Matrix

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
cat("\n Print element row 1 col 3 M[1,3]",M[1,3])
```

```
##
##  Print element row 1 col 3 M[1,3] 11
```

```
cat("\n Print row 1 M[1,]",M[1,])
```

```
##
##  Print row 1 M[1,] 3 7 11
```

```
cat("\n Print col 3 M[,3]",M[,3])
```

```
##
##  Print col 3 M[,3] 11 12 13 14
```

```
M[2,3] = 20  #Assigning value 20 to the element at 2nd row and 3rd column
cat("\n After assigning 20 to M[2,3] \n")
```

```
##
##  After assigning 20 to M[2,3]
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   20
## [3,]    5    9   13
## [4,]    6   10   14
```

```
M[3,] = 300 # This is vectorized ... all elements in row 3 will be 300
cat("\n After assigning 300 to row 3 \n")
```

```
##
##  After assigning 300 to row 3
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   20
## [3,]  300  300  300
## [4,]    6   10   14
```

```
M[,2] = 23.4
cat("\n After assigning 23.4 to column 2 \n")
```

```
##
##  After assigning 23.4 to column 2
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3 23.4   11
## [2,]    4 23.4   20
## [3,]  300 23.4  300
## [4,]    6 23.4   14
```

```
M <- matrix(c(3:14), nrow = 4)
print(M - 30) # create a copy of matrix
```

```
##      [,1] [,2] [,3]
## [1,]  -27  -23  -19
## [2,]  -26  -22  -18
## [3,]  -25  -21  -17
## [4,]  -24  -20  -16
```

```
print(M) # original matrix is still the same
```

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

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
M[2,2] = M[2,2] + 10
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4   18   12
## [3,]    5    9   13
## [4,]    6   10   14
```

```
M[3,] = M[3,] - 30
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4   18   12
## [3,]  -25  -21  -17
## [4,]    6   10   14
```

```
M[,1] = M[,1] * 0.1
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]  0.3    7   11
## [2,]  0.4   18   12
## [3,] -2.5  -21  -17
## [4,]  0.6   10   14
```

```
M = M + 10101
print(M)
```

```
##         [,1]  [,2]  [,3]
## [1,] 10101.3 10108 10112
## [2,] 10101.4 10119 10113
## [3,] 10098.5 10080 10084
## [4,] 10101.6 10111 10115
```

Change Matrix Elements based on Condition

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
print(M[M < 6])
```

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

```
M[M < 6] = 100
print("Modified Array is ")
```

```
## [1] "Modified Array is "
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]  100    7   11
## [2,]  100    8   12
## [3,]  100    9   13
## [4,]    6   10   14
```

```
M[M < 100] = M[M < 100] * 20
print("Modified Array is ")
```

```
## [1] "Modified Array is "
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]  100  140  220
## [2,]  100  160  240
## [3,]  100  180  260
## [4,]  120  200  280
```

```
# modify elements in 2nd row to 200 if element is less than 10
M <- matrix(c(3:14), nrow = 4)
M[2,][M[2,] < 10] = 200
M
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]  200  200   12
## [3,]    5    9   13
## [4,]    6   10   14
```

Add new column to matrix

here we use 'cbind()' function

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
M = cbind(M, c(0,0,0,0))

print("Modified matrix")
```

```
## [1] "Modified matrix"
```

```
print(M)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3    7   11    0
## [2,]    4    8   12    0
## [3,]    5    9   13    0
## [4,]    6   10   14    0
```

```
M = cbind(M, c(50,51,52))
```

```
## Warning in cbind(M, c(50, 51, 52)): number of rows of result is not a multiple
## of vector length (arg 2)
```

```
print(M)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    3    7   11    0   50
## [2,]    4    8   12    0   51
## [3,]    5    9   13    0   52
## [4,]    6   10   14    0   50
```

```
M = cbind(M, -1)
print(M)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    3    7   11    0   50   -1
## [2,]    4    8   12    0   51   -1
## [3,]    5    9   13    0   52   -1
## [4,]    6   10   14    0   50   -1
```

Add new row to matrix

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
M = rbind(M , c(0))
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
## [3,]    5    9   13
## [4,]    6   10   14
## [5,]    0    0    0
```

```
M = rbind(M , c(100,200,300,400))
```

```
## Warning in rbind(M, c(100, 200, 300, 400)): number of columns of result is not a
## multiple of vector length (arg 2)
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
## [3,]    5    9   13
## [4,]    6   10   14
## [5,]    0    0    0
## [6,]  100  200  300
```

```
M = rbind(M , c(-1,-2))
```

```
## Warning in rbind(M, c(-1, -2)): number of columns of result is not a multiple of
## vector length (arg 2)
```

```
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
## [3,]    5    9   13
## [4,]    6   10   14
## [5,]    0    0    0
## [6,]  100  200  300
## [7,]   -1   -2   -1
```

Reshape Matrix

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
print("Matrix dimensions are ")
```

```
## [1] "Matrix dimensions are "
```

```
print(dim(M))
```

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

```
dim(M) = c(2,6)
print("New dimensions are")
```

```
## [1] "New dimensions are"
```

```
print(dim(M))
```

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

```
# NOT Allowed
#dim(M) = c(2,2) # Less values in target
#print("New dimensions are")
#print(dim(M))
#dim(M) = c(1,14) # More values in target
#print("New dimensions are")
#print(dim(M))
M
```

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

Transpose of Matrix

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
print("transpose is")
```

```
## [1] "transpose is"
```

```
t(M)
```

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

Removing Elements from Matrix

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
print("Removing  third row")
```

```
## [1] "Removing  third row"
```

```
M[-3,]
```

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

```
print("Removing  third & 4 th row")
```

```
## [1] "Removing  third & 4 th row"
```

```
M[c(-3,-4),]
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
```

```
# Original matrix is not modified!!
print("Removing  Second column")
```

```
## [1] "Removing  Second column"
```

```
M[,-2]
```

```
##      [,1] [,2]
## [1,]    3   11
## [2,]    4   12
## [3,]    5   13
## [4,]    6   14
```

```
print("Removing  1 &3 column")
```

```
## [1] "Removing  1 &3 column"
```

```
M[,c(-1,-3)]
```

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

```
print("Removing one row and one column ")
```

```
## [1] "Removing one row and one column "
```

```
M[-1,-2]
```

```
##      [,1] [,2]
## [1,]    4   12
## [2,]    5   13
## [3,]    6   14
```

Seach An element in Matrix Find index of an element in matrix

```
M <- matrix(c(3:14), nrow = 4)
print(M)
```

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

```
print("Index of 9")
```

```
## [1] "Index of 9"
```

```
which(M == 9)
```

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

```
print("Elements excluding 9")
```

```
## [1] "Elements excluding 9"
```

```
M[-which(M == 9)] # remove single element from matrix
```

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

```
print("Index of even nos")
```

```
## [1] "Index of even nos"
```

```
which(M %% 2 ==0)
```

```
## [1]  2  4  6  8 10 12
```

```
print("Elements excluding even nos")
```

```
## [1] "Elements excluding even nos"
```

```
M[- which(M %%2 ==0)] # remove all even nos from the matrix
```

```
## [1]  3  5  7  9 11 13
```

Matrix Operations

```
M1 <- matrix(c(1:9), nrow = 3)
M2 <- matrix(c(11:19), nrow = 3)
cat("\n After addition \n")
```

```
##
##  After addition
```

```
print(M1+M2)
```

```
##      [,1] [,2] [,3]
## [1,]   12   18   24
## [2,]   14   20   26
## [3,]   16   22   28
```

```
cat("\n After Substraction \n")
```

```
##
##  After Substraction
```

```
print(M1-M2)
```

```
##      [,1] [,2] [,3]
## [1,]  -10  -10  -10
## [2,]  -10  -10  -10
## [3,]  -10  -10  -10
```

```
cat("\n After elementwise Multiplication \n")
```

```
##
##  After elementwise Multiplication
```

```
print(M1*M2)
```

```
##      [,1] [,2] [,3]
## [1,]   11   56  119
## [2,]   24   75  144
## [3,]   39   96  171
```

```
cat("\n After Matrix Multiplication \n")
```

```
##
##  After Matrix Multiplication
```

```
print(M1 %*% M2)
```

```
##      [,1] [,2] [,3]
## [1,]  150  186  222
## [2,]  186  231  276
## [3,]  222  276  330
```

```
cat("\n After Division \n")
```

```
##
##  After Division
```

```
print(M1/M2)
```

```
##            [,1]      [,2]      [,3]
## [1,] 0.09090909 0.2857143 0.4117647
## [2,] 0.16666667 0.3333333 0.4444444
## [3,] 0.23076923 0.3750000 0.4736842
```

```
cat("\n After Matrix Multiplication with scalar (constant)\n")
```

```
##
##  After Matrix Multiplication with scalar (constant)
```

```
print(M1 * 4)
```

```
##      [,1] [,2] [,3]
## [1,]    4   16   28
## [2,]    8   20   32
## [3,]   12   24   36
```

R Loops

Basic Loops in R programming

Repeat Loop

The Repeat loop executes the same code again and again until specifically taken out by the programmer.

NOTE If no specific *break* statement is written, then this will be infinite loop.

```
v = c("Inside", "loop")
cnt = 1
repeat{
  print(v)
  cnt = cnt+1
  if(cnt > 5){
    break
  }
}
```

```
## [1] "Inside" "loop"
## [1] "Inside" "loop"
## [1] "Inside" "loop"
## [1] "Inside" "loop"
## [1] "Inside" "loop"
```

While Loop

Loop runs till the test condition is TRUE. When test condition is FALSE then loop stops.

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

```
cnt = 1
while (cnt <= 4)
{
  print(cnt)
  cnt = cnt + 1
}
```

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

For Loop

It works on every element of the given vector or array

One element from the given vector is taken in every iteration

```
v1 = c(10,20,30)
for( i in v1){
  print(i)
}
```

```
## [1] 10
## [1] 20
## [1] 30
```

```
v <- c("IACSD","is", "in","Pune")
for ( i in v) {
  print(i)
}
```

```
## [1] "IACSD"
## [1] "is"
## [1] "in"
## [1] "Pune"
```

Break statement

break statement will break the flow of the loop which is immediate.

NOTE If there is loop within loop then control will be taken out of only one loop.

```
for ( i in 1:4){
  print(i)
  if(i == 2)
    break
}
```

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

```
for ( i in 1:4){
  for (j in 10:25){
    print(j)
    if(j == 12)
      break
  }
}
```

```
## [1] 10
## [1] 11
## [1] 12
## [1] 10
## [1] 11
## [1] 12
## [1] 10
## [1] 11
## [1] 12
## [1] 10
## [1] 11
## [1] 12
```

Next Statement

The next statement in R programming language is useful when we want to skip the current iteration of a loop without terminating it.

```
v = LETTERS[1:6]
for ( i in v){
  if (i == "D"){
    next
  }
  print(i)
}
```

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

Making Loops Faster

lapply function

It is used to execute a given function on every element of input vector or array or list

Example:: In following code there is list with two sequences 1 to 5 and 5 to 10

Now lapply() will apply mean function on both elements from the list

```
x <- list(a = 1:5, b = 5:10)
x$a
```

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

```
x$b
```

```
## [1]  5  6  7  8  9 10
```

```
lapply(x, mean)# find mean of each element in x
```

```
## $a
## [1] 3
##
## $b
## [1] 7.5
```

lapply with multiple arguments

lapply can apply a function taking multiple arguments to a given vector or array or list

In following code runif function with two arguments min and max is applied on every element of a given vector or array or list

```
x <- 1:4
lapply(x, runif, min = 10, max = 80)
```

```
## [[1]]
## [1] 74.18367
##
## [[2]]
## [1] 64.15528 67.47851
##
## [[3]]
## [1] 25.40679 58.09359 52.95168
##
## [[4]]
## [1] 62.13321 39.83950 54.79650 53.17861
```

sapply (simplify lapply)

sapply() calls lapply() on its input and then applies the following algorithm:

If the result is a list where every element is length 1, then a vector is returned

If the result is a list where every element is a vector of the same length (> 1), a matrix is returned.

If it can’t figure things out, a list is returned

Example

```
x <- list(a = 1:5, b = 11:15)
sapply(x, mean)
```

```
##  a  b
##  3 13
```

R Dataframe Functions

Create Dataframe from matrix / array

as.data.frame()

```
M1 = matrix(c(1,2,3,4),nrow=2)
#M1
df <- as.data.frame(M1)
df
```

V1 <dbl>	V2 <dbl>
1	3
2	4

2 rows

```
df$V1
```

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

```
str(df)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ V1: num  1 2
##  $ V2: num  3 4
```

```
is.data.frame(df)
```

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

```
nrow(df)
```

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

```
ncol(df)
```

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

```
length(df)
```

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

```
colnames(df)
```

```
## [1] "V1" "V2"
```

```
names(df)
```

```
## [1] "V1" "V2"
```

```
M1 = matrix(c(1,2,3,4),nrow=2)
m_df <- as.data.frame(M1)
colnames(m_df) = c('c1','c2') # change column names
print(m_df)
```

```
##   c1 c2
## 1  1  3
## 2  2  4
```

```
str(m_df)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ c1: num  1 2
##  $ c2: num  3 4
```

```
ncol(m_df)
```

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

```
#nominal varible doesn't have any order
emp = c("John Doe", "Peter Gynn", "Jolie Hope")
#Ordinal varible has order
designation <- factor(c("Manager", "Team Lead","SME"), ordered =TRUE,          levels = c("SM
E", "Team Lead","Manager"))
salary <- c(41000, 33400, 26800)
startdate <- as.Date(c("2010-11-1", "2008-3-25", "2007-3-14"))
employee.data <- data.frame(emp, designation,salary, startdate)

str(employee.data)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ emp        : chr  "John Doe" "Peter Gynn" "Jolie Hope"
##  $ designation: Ord.factor w/ 3 levels "SME"<"Team Lead"<..: 3 2 1
##  $ salary     : num  41000 33400 26800
##  $ startdate  : Date, format: "2010-11-01" "2008-03-25" ...
```

```
employee.data
```

emp <chr>	designation <ord>	salary <dbl>	startdate <date>
John Doe	Manager	41000	2010-11-01
Peter Gynn	Team Lead	33400	2008-03-25
Jolie Hope	SME	26800	2007-03-14

3 rows

```
employ.data <- data.frame(emp, salary, startdate, stringsAsFactors=TRUE)
str(employ.data)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ emp      : Factor w/ 3 levels "John Doe","Jolie Hope",..: 1 3 2
##  $ salary   : num  41000 33400 26800
##  $ startdate: Date, format: "2010-11-01" "2008-03-25" ...
```

```
# variaable name can have .
employ.data <- data.frame(emp, salary, startdate, stringsAsFactors=FALSE)
# select rows based on condition
employ.data[employ.data$salary>35000,]
```

	emp <chr>	salary <dbl>	startdate <date>
1	John Doe	41000	2010-11-01

1 row

```
employ.data$salary
```

```
## [1] 41000 33400 26800
```

```
employ.data[c('emp','salary')]
```

emp <chr>	salary <dbl>

emp <chr>	salary <dbl>
John Doe	41000
Peter Gynn	33400
Jolie Hope	26800

3 rows

```
employ.data[employ.data$salary>35000,c('emp','salary')]
```

	emp <chr>	salary <dbl>
1	John Doe	41000

1 row

Reading TEXT / CSV into Data frame

read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", …)

GET HELP –> ?read.csv

read.csv2(file, header = TRUE, sep = ";", quote = "\"", dec = ",", fill = TRUE, comment.char = "", …)

```
df = read.csv("76_attributes_heartdiseases.csv")
summary(df)
```

```
##        V1               V2          V3              V4               V5
##  Min.   :  1.00   Min.   :0   Min.   :29.00   Min.   :0.0000   Min.   :-9
##  1st Qu.: 75.25   1st Qu.:0   1st Qu.:48.00   1st Qu.:0.0000   1st Qu.:-9
##  Median :151.50   Median :0   Median :55.00   Median :1.0000   Median :-9
##  Mean   :151.52   Mean   :0   Mean   :54.41   Mean   :0.6773   Mean   :-9
##  3rd Qu.:227.75   3rd Qu.:0   3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:-9
##  Max.   :298.00   Max.   :0   Max.   :77.00   Max.   :1.0000   Max.   :-9
##        V6           V7               V8              V9
##  Min.   :-9   Min.   :-18.000   Min.   :-9.000   Min.   :  1.000
##  1st Qu.:-9   1st Qu.: -9.000   1st Qu.:-9.000   1st Qu.:  3.000
##  Median :-9   Median : -9.000   Median :-9.000   Median :  3.000
##  Mean   :-9   Mean   : -9.096   Mean   :-8.869   Mean   :  4.433
##  3rd Qu.:-9   3rd Qu.: -9.000   3rd Qu.:-9.000   3rd Qu.:  4.000
##  Max.   :-9   Max.   : -9.000   Max.   : 4.000   Max.   :130.000
##       V10             V11               V12             V13
##  Min.   :  1.0   Min.   :  0.000   Min.   : -9.0   Min.   :-9.000
##  1st Qu.:120.0   1st Qu.:  0.000   1st Qu.:212.0   1st Qu.:-9.000
##  Median :130.0   Median :  1.000   Median :244.0   Median :-9.000
##  Mean   :130.3   Mean   :  2.823   Mean   :246.8   Mean   :-8.727
##  3rd Qu.:140.0   3rd Qu.:  1.000   3rd Qu.:277.0   3rd Qu.:-9.000
##  Max.   :200.0   Max.   :253.000   Max.   :564.0   Max.   :30.000
##       V14             V15              V16                V17
##  Min.   :-9.0   Min.   :-9.00   Min.   :-9.00000   Min.   :-9.000
##  1st Qu.: 0.0   1st Qu.: 0.00   1st Qu.: 0.00000   1st Qu.:-9.000
##  Median :10.0   Median :15.00   Median : 0.00000   Median :-9.000
##  Mean   :16.5   Mean   :14.62   Mean   : 0.05319   Mean   :-8.078
##  3rd Qu.:30.0   3rd Qu.:30.00   3rd Qu.: 0.00000   3rd Qu.:-9.000
##  Max.   :99.0   Max.   :54.00   Max.   : 1.00000   Max.   : 1.000
##       V18              V19              V20             V21
##  Min.   :0.0000   Min.   : 0.000   Min.   : 1.0   Min.   : 1.00
##  1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.: 3.0   1st Qu.: 8.00
##  Median :1.0000   Median : 2.000   Median : 7.0   Median :15.00
##  Mean   :0.6099   Mean   : 1.071   Mean   : 6.5   Mean   :15.94
##  3rd Qu.:1.0000   3rd Qu.: 2.000   3rd Qu.:10.0   3rd Qu.:22.00
##  Max.   :1.0000   Max.   :11.000   Max.   :23.0   Max.   :82.00
##       V22             V23                V24               V25
##  Min.   : 0.00   Min.   :-9.00000   Min.   :-9.0000   Min.   :-9.0000
##  1st Qu.:82.00   1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.: 0.0000
##  Median :82.00   Median : 0.00000   Median : 0.0000   Median : 0.0000
##  Mean   :81.43   Mean   :-0.03191   Mean   : 0.2695   Mean   : 0.1809
##  3rd Qu.:83.00   3rd Qu.: 0.00000   3rd Qu.: 1.0000   3rd Qu.: 0.0000
##  Max.   :84.00   Max.   : 1.00000   Max.   : 1.0000   Max.   : 1.0000
##       V26                V27               V28            V29
##  Min.   :-9.00000   Min.   :-9.00000   Min.   :1.00   Min.   : 1.800
##  1st Qu.: 0.00000   1st Qu.: 0.00000   1st Qu.:1.00   1st Qu.: 6.500
##  Median : 0.00000   Median : 0.00000   Median :1.00   Median : 8.500
##  Mean   : 0.03546   Mean   : 0.06028   Mean   :1.08   Mean   : 8.405
##  3rd Qu.: 0.00000   3rd Qu.: 0.00000   3rd Qu.:1.00   3rd Qu.:10.075
##  Max.   : 1.00000   Max.   : 1.00000   Max.   :9.00   Max.   :15.000
##       V30              V31             V32           V33
##  Min.   :-9.000   Min.   :  3.0   Min.   : 71   Min.   : 40.00
```

```
##  1st Qu.: 0.000   1st Qu.:  7.0   1st Qu.:132   1st Qu.: 65.00
##  Median : 3.000   Median :  9.5   Median :153   Median : 74.00
##  Mean   : 1.507   Mean   : 11.3   Mean   :149   Mean   : 75.95
##  3rd Qu.: 6.000   3rd Qu.: 12.0   3rd Qu.:165   3rd Qu.: 85.00
##  Max.   :15.000   Max.   :175.0   Max.   :202   Max.   :190.00
##       V34             V35             V36             V37
##  Min.   : 84.0   Min.   : 26.00   Min.   : 78.0   Min.   :  0.00
##  1st Qu.:152.0   1st Qu.: 70.00   1st Qu.:120.0   1st Qu.: 80.00
##  Median :168.0   Median : 80.00   Median :130.0   Median : 85.00
##  Mean   :167.3   Mean   : 79.09   Mean   :131.2   Mean   : 84.04
##  3rd Qu.:183.5   3rd Qu.: 85.00   3rd Qu.:140.0   3rd Qu.: 90.00
##  Max.   :232.0   Max.   :130.00   Max.   :200.0   Max.   :110.00
##       V38              V39              V40             V41
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.000   Min.   :-9.000
##  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.000   1st Qu.: 1.000
##  Median :0.0000   Median :0.00000   Median :0.800   Median : 2.000
##  Mean   :0.3227   Mean   :0.03652   Mean   :1.037   Mean   : 1.433
##  3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:1.600   3rd Qu.: 2.000
##  Max.   :1.0000   Max.   :1.80000   Max.   :6.200   Max.   : 3.000
##       V42              V43             V44              V45          V46
##  Min.   : -9.000   Min.   :  0.00   Min.   :-9.0000   Min.   :-9   Min.   :-9
##  1st Qu.: -9.000   1st Qu.: 90.25   1st Qu.: 0.0000   1st Qu.:-9   1st Qu.:-9
##  Median : -9.000   Median :117.50   Median : 0.0000   Median :-9   Median :-9
##  Mean   : -6.564   Mean   :121.29   Mean   : 0.4539   Mean   :-9   Mean   :-9
##  3rd Qu.: -9.000   3rd Qu.:150.00   3rd Qu.: 1.0000   3rd Qu.:-9   3rd Qu.:-9
##  Max.   :200.000   Max.   :270.00   Max.   : 3.0000   Max.   :-9   Max.   :-9
##       V47          V48          V49          V50              V51
##  Min.   :-9   Min.   :-9   Min.   :-9   Min.   :-9.000   Min.   :-9.000
##  1st Qu.:-9   1st Qu.:-9   1st Qu.:-9   1st Qu.:-9.000   1st Qu.: 3.000
##  Median :-9   Median :-9   Median :-9   Median :-9.000   Median : 3.000
##  Mean   :-9   Mean   :-9   Mean   :-9   Mean   :-8.787   Mean   : 4.369
##  3rd Qu.:-9   3rd Qu.:-9   3rd Qu.:-9   3rd Qu.:-9.000   3rd Qu.: 7.000
##  Max.   :-9   Max.   :-9   Max.   :-9   Max.   : 7.000   Max.   : 7.000
##       V52          V53          V54             V55              V56
##  Min.   :-9   Min.   :-9   Min.   :-9.00   Min.   : 1.000   Min.   : 1.00
##  1st Qu.:-9   1st Qu.:-9   1st Qu.:-9.00   1st Qu.: 3.000   1st Qu.: 8.00
##  Median :-9   Median :-9   Median :-9.00   Median : 7.000   Median :15.00
##  Mean   :-9   Mean   :-9   Mean   :-8.78   Mean   : 6.571   Mean   :16.24
##  3rd Qu.:-9   3rd Qu.:-9   3rd Qu.:-9.00   3rd Qu.:10.000   3rd Qu.:23.00
##  Max.   :-9   Max.   :-9   Max.   :11.00   Max.   :29.000   Max.   :82.00
##       V57             V58             V59             V60
##  Min.   : 0.00   Min.   :0.0000   Min.   :1.000   Min.   :1.000
##  1st Qu.:82.00   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:1.000
##  Median :82.00   Median :0.0000   Median :1.000   Median :1.000
##  Mean   :81.15   Mean   :0.9184   Mean   :1.043   Mean   :1.145
##  3rd Qu.:83.00   3rd Qu.:2.0000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :84.00   Max.   :4.0000   Max.   :2.000   Max.   :2.000
##       V61             V62             V63                V64
##  Min.   :-9.00   Min.   :-9.000   Length:282         Min.   :-9.000
##  1st Qu.: 1.00   1st Qu.:-9.000   Class :character   1st Qu.:-9.000
##  Median : 1.00   Median :-9.000   Mode  :character   Median :-9.000
```

```
##  Mean   : 1.06   Mean   :-8.858                     Mean   :-8.784
##  3rd Qu.: 1.00   3rd Qu.:-9.000                     3rd Qu.:-9.000
##  Max.   : 2.00   Max.   : 1.000                     Max.   : 2.000
##       V65               V66              V67             V68             V69
##  Min.   :-9.0000   Min.   :-9.000   Min.   :1.000   Min.   :1.000   Min.   :1
##  1st Qu.: 1.0000   1st Qu.:-9.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1
##  Median : 1.0000   Median :-9.000   Median :1.000   Median :1.000   Median :1
##  Mean   : 0.9468   Mean   :-8.784   Mean   :1.174   Mean   :1.124   Mean   :1
##  3rd Qu.: 1.0000   3rd Qu.:-9.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1
##  Max.   : 2.0000   Max.   : 2.000   Max.   :2.000   Max.   :2.000   Max.   :1
##       V70             V71              V72             V73
##  Min.   :1.000   Min.   : 1.000   Min.   :1.000   Min.   :-9.0000
##  1st Qu.:1.000   1st Qu.: 1.000   1st Qu.:1.000   1st Qu.: 1.0000
##  Median :1.000   Median : 1.000   Median :1.000   Median : 1.0000
##  Mean   :1.007   Mean   : 1.177   Mean   :1.404   Mean   : 0.8901
##  3rd Qu.:1.000   3rd Qu.: 1.000   3rd Qu.:1.000   3rd Qu.: 1.0000
##  Max.   :3.000   Max.   :11.000   Max.   :8.000   Max.   : 4.0000
##       V74          V75                V76
##  Min.   :-9   Length:282         Length:282
##  1st Qu.:-9   Class :character   Class :character
##  Median :-9   Mode  :character   Mode  :character
##  Mean   :-9
##  3rd Qu.:-9
##  Max.   :-9
```

```
#df_sample = read.csv("sample3.txt",sep="")
#df_sample
df_sample = read.csv("sample1.txt",sep="|")
df_sample
```

c1 <int>	c2 <int>	c3 <int>
1	23	233
2	45	254
3	67	555

3 rows

```
df = read.csv("76_attributes_heartdiseases.csv")
summary(df$V73)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -9.0000  1.0000  1.0000  0.8901  1.0000  4.0000
```

```
df = read.csv("76_attributes_heartdiseases.csv",na.strings=c("-9","-18"))
summary(df$V73)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   1.000   1.000   1.000   1.142   1.000   4.000       7
```

```
summary(df)
```

```
##        V1               V2          V3              V4            V5
##  Min.   :  1.00   Min.   :0   Min.   :29.00   Min.   :0.0000   Mode:logical
##  1st Qu.: 75.25   1st Qu.:0   1st Qu.:48.00   1st Qu.:0.0000   NA's:282
##  Median :151.50   Median :0   Median :55.00   Median :1.0000
##  Mean   :151.52   Mean   :0   Mean   :54.41   Mean   :0.6773
##  3rd Qu.:227.75   3rd Qu.:0   3rd Qu.:61.00   3rd Qu.:1.0000
##  Max.   :298.00   Max.   :0   Max.   :77.00   Max.   :1.0000
##
##     V6             V7               V8              V9
##  Mode:logical   Mode:logical   Min.   :3.000   Min.   :  1.000
##  NA's:282       NA's:282       1st Qu.:3.000   1st Qu.:  3.000
##                                Median :3.000   Median :  3.000
##                                Mean   :3.333   Mean   :  4.433
##                                3rd Qu.:3.500   3rd Qu.:  4.000
##                                Max.   :4.000   Max.   :130.000
##                                NA's   :279
##       V10             V11               V12             V13
##  Min.   :  1.0   Min.   :  0.000   Min.   :126.0   Min.   : 0.00
##  1st Qu.:120.0   1st Qu.:  0.000   1st Qu.:213.0   1st Qu.:10.00
##  Median :130.0   Median :  1.000   Median :244.0   Median :20.00
##  Mean   :130.3   Mean   :  2.823   Mean   :249.5   Mean   :16.67
##  3rd Qu.:140.0   3rd Qu.:  1.000   3rd Qu.:277.5   3rd Qu.:25.00
##  Max.   :200.0   Max.   :253.000   Max.   :564.0   Max.   :30.00
##                                    NA's   :3       NA's   :279
##       V14             V15             V16             V17
##  Min.   : 0.00   Min.   : 0.00   Min.   :0.0000   Min.   :1
##  1st Qu.: 0.00   1st Qu.: 0.00   1st Qu.:0.0000   1st Qu.:1
##  Median :10.00   Median :15.00   Median :0.0000   Median :1
##  Mean   :16.96   Mean   :15.04   Mean   :0.1505   Mean   :1
##  3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.:0.0000   3rd Qu.:1
##  Max.   :99.00   Max.   :54.00   Max.   :1.0000   Max.   :1
##  NA's   :5       NA's   :5       NA's   :3        NA's   :256
##       V18              V19             V20             V21
##  Min.   :0.0000   Min.   : 0.000   Min.   : 1.0   Min.   : 1.00
##  1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.: 3.0   1st Qu.: 8.00
##  Median :1.0000   Median : 2.000   Median : 7.0   Median :15.00
##  Mean   :0.6099   Mean   : 1.071   Mean   : 6.5   Mean   :15.94
##  3rd Qu.:1.0000   3rd Qu.: 2.000   3rd Qu.:10.0   3rd Qu.:22.00
##  Max.   :1.0000   Max.   :11.000   Max.   :23.0   Max.   :82.00
##
##       V22             V23               V24             V25
##  Min.   : 0.00   Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:82.00   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :82.00   Median :0.00000   Median :0.0000   Median :0.0000
##  Mean   :81.43   Mean   :0.03214   Mean   :0.3357   Mean   :0.2464
##  3rd Qu.:83.00   3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :84.00   Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
##                  NA's   :2         NA's   :2        NA's   :2
##       V26           V27             V28            V29              V30
##  Min.   :0.0   Min.   :0.000   Min.   :1.00   Min.   : 1.800   Min.   : 0.000
##  1st Qu.:0.0   1st Qu.:0.000   1st Qu.:1.00   1st Qu.: 6.500   1st Qu.: 3.000
```

```
##  Median :0.0   Median :0.000   Median :1.00   Median : 8.500   Median : 5.500
##  Mean   :0.1   Mean   :0.125   Mean   :1.08   Mean   : 8.405   Mean   : 4.911
##  3rd Qu.:0.0   3rd Qu.:0.000   3rd Qu.:1.00   3rd Qu.:10.075   3rd Qu.: 7.500
##  Max.   :1.0   Max.   :1.000   Max.   :9.00   Max.   :15.000   Max.   :15.000
##  NA's   :2     NA's   :2                                       NA's   :69
##       V31             V32           V33              V34
##  Min.   :  3.0   Min.   : 71   Min.   : 40.00   Min.   : 84.0
##  1st Qu.:  7.0   1st Qu.:132   1st Qu.: 65.00   1st Qu.:152.0
##  Median :  9.5   Median :153   Median : 74.00   Median :168.0
##  Mean   : 11.3   Mean   :149   Mean   : 75.95   Mean   :167.3
##  3rd Qu.: 12.0   3rd Qu.:165   3rd Qu.: 85.00   3rd Qu.:183.5
##  Max.   :175.0   Max.   :202   Max.   :190.00   Max.   :232.0
##
##       V35             V36             V37              V38
##  Min.   : 26.00   Min.   : 78.0   Min.   :  0.00   Min.   :0.0000
##  1st Qu.: 70.00   1st Qu.:120.0   1st Qu.: 80.00   1st Qu.:0.0000
##  Median : 80.00   Median :130.0   Median : 85.00   Median :0.0000
##  Mean   : 79.09   Mean   :131.2   Mean   : 84.04   Mean   :0.3227
##  3rd Qu.: 85.00   3rd Qu.:140.0   3rd Qu.: 90.00   3rd Qu.:1.0000
##  Max.   :130.00   Max.   :200.0   Max.   :110.00   Max.   :1.0000
##
##       V39               V40             V41             V42
##  Min.   :0.00000   Min.   :0.000   Min.   :1.000   Min.   :105.0
##  1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:153.8
##  Median :0.00000   Median :0.800   Median :2.000   Median :173.0
##  Mean   :0.03652   Mean   :1.037   Mean   :1.583   Mean   :162.8
##  3rd Qu.:0.00000   3rd Qu.:1.600   3rd Qu.:2.000   3rd Qu.:182.0
##  Max.   :1.80000   Max.   :6.200   Max.   :3.000   Max.   :200.0
##                                    NA's   :4       NA's   :278
##       V43              V44           V45             V46             V47
##  Min.   :  0.00   Min.   :0.0000   Mode:logical   Mode:logical   Mode:logical
##  1st Qu.: 90.25   1st Qu.:0.0000   NA's:282       NA's:282       NA's:282
##  Median :117.50   Median :0.0000
##  Mean   :121.29   Mean   :0.6594
##  3rd Qu.:150.00   3rd Qu.:1.0000
##  Max.   :270.00   Max.   :3.0000
##                   NA's   :6
##    V48             V49               V50           V51           V52
##  Mode:logical   Mode:logical   Min.   :3     Min.   :3.000   Mode:logical
##  NA's:282       NA's:282       1st Qu.:6     1st Qu.:3.000   NA's:282
##                                Median :7     Median :3.000
##                                Mean   :6     Mean   :4.659
##                                3rd Qu.:7     3rd Qu.:7.000
##                                Max.   :7     Max.   :7.000
##                                NA's   :278   NA's   :6
##    V53               V54             V55              V56
##  Mode:logical   Min.   : 2.00   Min.   : 1.000   Min.   : 1.00
##  NA's:282       1st Qu.: 2.75   1st Qu.: 3.000   1st Qu.: 8.00
##                 Median : 6.50   Median : 7.000   Median :15.00
##                 Mean   : 6.50   Mean   : 6.571   Mean   :16.24
##                 3rd Qu.:10.25   3rd Qu.:10.000   3rd Qu.:23.00
```

```
##                Max.   :11.00   Max.   :29.000   Max.   :82.00
##                NA's   :278
##       V57              V58               V59              V60
##  Min.   : 0.00   Min.   :0.0000   Min.   :1.000   Min.   :1.000
##  1st Qu.:82.00   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:1.000
##  Median :82.00   Median :0.0000   Median :1.000   Median :1.000
##  Mean   :81.15   Mean   :0.9184   Mean   :1.043   Mean   :1.145
##  3rd Qu.:83.00   3rd Qu.:2.0000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :84.00   Max.   :4.0000   Max.   :2.000   Max.   :2.000
##
##       V61              V62           V63                 V64
##  Min.   :1.000   Min.   :1     Length:282         Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1     Class :character   1st Qu.:1.000
##  Median :1.000   Median :1     Mode  :character   Median :1.000
##  Mean   :1.205   Mean   :1                        Mean   :1.167
##  3rd Qu.:1.000   3rd Qu.:1                        3rd Qu.:1.000
##  Max.   :2.000   Max.   :1                        Max.   :2.000
##  NA's   :4       NA's   :278                      NA's   :276
##       V65              V66              V67              V68              V69
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1
##  Median :1.000   Median :1.000   Median :1.000   Median :1.000   Median :1
##  Mean   :1.163   Mean   :1.167   Mean   :1.174   Mean   :1.124   Mean   :1
##  3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1
##  Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :1
##  NA's   :6       NA's   :276
##       V70              V71               V72              V73
##  Min.   :1.000   Min.   : 1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.: 1.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :1.000   Median : 1.000   Median :1.000   Median :1.000
##  Mean   :1.007   Mean   : 1.177   Mean   :1.404   Mean   :1.142
##  3rd Qu.:1.000   3rd Qu.: 1.000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :3.000   Max.   :11.000   Max.   :8.000   Max.   :4.000
##                                                   NA's   :7
##    V74              V75                V76
##  Mode:logical   Length:282         Length:282
##  NA's:282       Class :character   Class :character
##                 Mode  :character   Mode  :character
##
##
##
##
```

```
df_sample = read.csv("sample2.txt",skip=2,sep="|")
df_sample
```

c1	c2	c3
<int>	<int>	<int>
1	23	233

c1 <int>	c2 <int>	c3 <int>
2	45	254
3	67	555

3 rows

```
df_sample = read.csv("sample2.txt",skip=2,sep="|",blank.lines.skip=FALSE)
df_sample
```

c1 <int>	c2 <int>	c3 <int>
1	23	233
2	45	254
NA	*NA*	*NA*
NA	*NA*	*NA*
3	67	555

5 rows

```
df_sample = read.csv("sample2.txt",skip=0)
df_sample
```

This.is.a.sample.file <chr>
We need to skip first two lines
c1|c2|c3
1|23|233
2|45|254
3|67|555

5 rows

Few more IMP operations in read.csv

stringsAsFactors –> Default to TRUE. String columns will be treated as factors

```
If FALSE then Strings will be kept intact
```

encoding –> For handing unicode or other encoding problems

nrows –> integer: the maximum number of rows to read in.

```
       Negative and other invalid values are ignored.

       Useful when loading very large files
```

```
print(ncol(df))
```

```
## [1] 76
```

```
print(nrow(df))
```

```
## [1] 282
```

```
summary(df)
```

```
##        V1               V2          V3              V4            V5
##  Min.   :  1.00   Min.   :0   Min.   :29.00   Min.   :0.0000   Mode:logical
##  1st Qu.: 75.25   1st Qu.:0   1st Qu.:48.00   1st Qu.:0.0000   NA's:282
##  Median :151.50   Median :0   Median :55.00   Median :1.0000
##  Mean   :151.52   Mean   :0   Mean   :54.41   Mean   :0.6773
##  3rd Qu.:227.75   3rd Qu.:0   3rd Qu.:61.00   3rd Qu.:1.0000
##  Max.   :298.00   Max.   :0   Max.   :77.00   Max.   :1.0000
##
##     V6             V7               V8              V9
##  Mode:logical   Mode:logical   Min.   :3.000   Min.   :  1.000
##  NA's:282       NA's:282       1st Qu.:3.000   1st Qu.:  3.000
##                                Median :3.000   Median :  3.000
##                                Mean   :3.333   Mean   :  4.433
##                                3rd Qu.:3.500   3rd Qu.:  4.000
##                                Max.   :4.000   Max.   :130.000
##                                NA's   :279
##       V10             V11               V12             V13
##  Min.   :  1.0   Min.   :  0.000   Min.   :126.0   Min.   : 0.00
##  1st Qu.:120.0   1st Qu.:  0.000   1st Qu.:213.0   1st Qu.:10.00
##  Median :130.0   Median :  1.000   Median :244.0   Median :20.00
##  Mean   :130.3   Mean   :  2.823   Mean   :249.5   Mean   :16.67
##  3rd Qu.:140.0   3rd Qu.:  1.000   3rd Qu.:277.5   3rd Qu.:25.00
##  Max.   :200.0   Max.   :253.000   Max.   :564.0   Max.   :30.00
##                                    NA's   :3       NA's   :279
##       V14             V15             V16             V17
##  Min.   : 0.00   Min.   : 0.00   Min.   :0.0000   Min.   :1
##  1st Qu.: 0.00   1st Qu.: 0.00   1st Qu.:0.0000   1st Qu.:1
##  Median :10.00   Median :15.00   Median :0.0000   Median :1
##  Mean   :16.96   Mean   :15.04   Mean   :0.1505   Mean   :1
##  3rd Qu.:30.00   3rd Qu.:30.00   3rd Qu.:0.0000   3rd Qu.:1
##  Max.   :99.00   Max.   :54.00   Max.   :1.0000   Max.   :1
##  NA's   :5       NA's   :5       NA's   :3        NA's   :256
##       V18              V19             V20             V21
##  Min.   :0.0000   Min.   : 0.000   Min.   : 1.0   Min.   : 1.00
##  1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.: 3.0   1st Qu.: 8.00
##  Median :1.0000   Median : 2.000   Median : 7.0   Median :15.00
##  Mean   :0.6099   Mean   : 1.071   Mean   : 6.5   Mean   :15.94
##  3rd Qu.:1.0000   3rd Qu.: 2.000   3rd Qu.:10.0   3rd Qu.:22.00
##  Max.   :1.0000   Max.   :11.000   Max.   :23.0   Max.   :82.00
##
##       V22             V23               V24             V25
##  Min.   : 0.00   Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:82.00   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :82.00   Median :0.00000   Median :0.0000   Median :0.0000
##  Mean   :81.43   Mean   :0.03214   Mean   :0.3357   Mean   :0.2464
##  3rd Qu.:83.00   3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :84.00   Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
##                  NA's   :2         NA's   :2        NA's   :2
##       V26           V27             V28            V29              V30
##  Min.   :0.0   Min.   :0.000   Min.   :1.00   Min.   : 1.800   Min.   : 0.000
##  1st Qu.:0.0   1st Qu.:0.000   1st Qu.:1.00   1st Qu.: 6.500   1st Qu.: 3.000
```

```
##  Median :0.0   Median :0.000   Median :1.00   Median : 8.500   Median : 5.500
##  Mean   :0.1   Mean   :0.125   Mean   :1.08   Mean   : 8.405   Mean   : 4.911
##  3rd Qu.:0.0   3rd Qu.:0.000   3rd Qu.:1.00   3rd Qu.:10.075   3rd Qu.: 7.500
##  Max.   :1.0   Max.   :1.000   Max.   :9.00   Max.   :15.000   Max.   :15.000
##  NA's   :2     NA's   :2                                       NA's   :69
##       V31             V32           V33              V34
##  Min.   :  3.0   Min.   : 71   Min.   : 40.00   Min.   : 84.0
##  1st Qu.:  7.0   1st Qu.:132   1st Qu.: 65.00   1st Qu.:152.0
##  Median :  9.5   Median :153   Median : 74.00   Median :168.0
##  Mean   : 11.3   Mean   :149   Mean   : 75.95   Mean   :167.3
##  3rd Qu.: 12.0   3rd Qu.:165   3rd Qu.: 85.00   3rd Qu.:183.5
##  Max.   :175.0   Max.   :202   Max.   :190.00   Max.   :232.0
##
##       V35              V36             V37              V38
##  Min.   : 26.00   Min.   : 78.0   Min.   :  0.00   Min.   :0.0000
##  1st Qu.: 70.00   1st Qu.:120.0   1st Qu.: 80.00   1st Qu.:0.0000
##  Median : 80.00   Median :130.0   Median : 85.00   Median :0.0000
##  Mean   : 79.09   Mean   :131.2   Mean   : 84.04   Mean   :0.3227
##  3rd Qu.: 85.00   3rd Qu.:140.0   3rd Qu.: 90.00   3rd Qu.:1.0000
##  Max.   :130.00   Max.   :200.0   Max.   :110.00   Max.   :1.0000
##
##       V39               V40             V41             V42
##  Min.   :0.00000   Min.   :0.000   Min.   :1.000   Min.   :105.0
##  1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:153.8
##  Median :0.00000   Median :0.800   Median :2.000   Median :173.0
##  Mean   :0.03652   Mean   :1.037   Mean   :1.583   Mean   :162.8
##  3rd Qu.:0.00000   3rd Qu.:1.600   3rd Qu.:2.000   3rd Qu.:182.0
##  Max.   :1.80000   Max.   :6.200   Max.   :3.000   Max.   :200.0
##                                    NA's   :4       NA's   :278
##       V43              V44           V45             V46             V47
##  Min.   :  0.00   Min.   :0.0000   Mode:logical   Mode:logical   Mode:logical
##  1st Qu.: 90.25   1st Qu.:0.0000   NA's:282       NA's:282       NA's:282
##  Median :117.50   Median :0.0000
##  Mean   :121.29   Mean   :0.6594
##  3rd Qu.:150.00   3rd Qu.:1.0000
##  Max.   :270.00   Max.   :3.0000
##                   NA's   :6
##    V48             V49               V50           V51           V52
##  Mode:logical   Mode:logical   Min.   :3     Min.   :3.000   Mode:logical
##  NA's:282       NA's:282       1st Qu.:6     1st Qu.:3.000   NA's:282
##                                Median :7     Median :3.000
##                                Mean   :6     Mean   :4.659
##                                3rd Qu.:7     3rd Qu.:7.000
##                                Max.   :7     Max.   :7.000
##                                NA's   :278   NA's   :6
##    V53               V54             V55              V56
##  Mode:logical   Min.   : 2.00   Min.   : 1.000   Min.   : 1.00
##  NA's:282       1st Qu.: 2.75   1st Qu.: 3.000   1st Qu.: 8.00
##                 Median : 6.50   Median : 7.000   Median :15.00
##                 Mean   : 6.50   Mean   : 6.571   Mean   :16.24
##                 3rd Qu.:10.25   3rd Qu.:10.000   3rd Qu.:23.00
```

```
##                Max.   :11.00   Max.   :29.000   Max.   :82.00
##                NA's   :278
##       V57             V58              V59             V60
##  Min.   : 0.00   Min.   :0.0000   Min.   :1.000   Min.   :1.000
##  1st Qu.:82.00   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:1.000
##  Median :82.00   Median :0.0000   Median :1.000   Median :1.000
##  Mean   :81.15   Mean   :0.9184   Mean   :1.043   Mean   :1.145
##  3rd Qu.:83.00   3rd Qu.:2.0000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :84.00   Max.   :4.0000   Max.   :2.000   Max.   :2.000
##
##       V61             V62          V63                V64
##  Min.   :1.000   Min.   :1     Length:282         Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1     Class :character   1st Qu.:1.000
##  Median :1.000   Median :1     Mode  :character   Median :1.000
##  Mean   :1.205   Mean   :1                        Mean   :1.167
##  3rd Qu.:1.000   3rd Qu.:1                        3rd Qu.:1.000
##  Max.   :2.000   Max.   :1                        Max.   :2.000
##  NA's   :4       NA's   :278                      NA's   :276
##       V65             V66             V67             V68             V69
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1
##  Median :1.000   Median :1.000   Median :1.000   Median :1.000   Median :1
##  Mean   :1.163   Mean   :1.167   Mean   :1.174   Mean   :1.124   Mean   :1
##  3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1
##  Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :2.000   Max.   :1
##  NA's   :6       NA's   :276
##       V70             V71              V72             V73
##  Min.   :1.000   Min.   : 1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.: 1.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :1.000   Median : 1.000   Median :1.000   Median :1.000
##  Mean   :1.007   Mean   : 1.177   Mean   :1.404   Mean   :1.142
##  3rd Qu.:1.000   3rd Qu.: 1.000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :3.000   Max.   :11.000   Max.   :8.000   Max.   :4.000
##                                                   NA's   :7
##    V74             V75                V76
##  Mode:logical   Length:282         Length:282
##  NA's:282       Class :character   Class :character
##                 Mode  :character   Mode  :character
##
##
##
##
```

Create subset of a dataframe

We can use subset() function. HELP –> ?subset

Arguments:

data / data frame

condition based on which to create the subset

select –> specify the array of columns to be selected

```
df_1 = subset(df, df$V1 > 200)
head(df_1)
```

	V1 <int>	V2 <int>	V3 <int>	V4 <int>	V5 <lgl>	V6 <lgl>	V7 <lgl>	V8 <int>	V9 <int>
185	201	0	60	0	*NA*	*NA*	*NA*	*NA*	4
186	202	0	63	0	*NA*	*NA*	*NA*	*NA*	2
187	203	0	42	1	*NA*	*NA*	*NA*	*NA*	3
188	204	0	66	1	*NA*	*NA*	*NA*	*NA*	2
189	205	0	54	1	*NA*	*NA*	*NA*	*NA*	2
190	206	0	69	1	*NA*	*NA*	*NA*	*NA*	3

6 rows | 1-10 of 77 columns

```
df_1 = subset(df, df$V1 > 200, select = c("V1","V2"))
head(df_1)
```

	V1 <int>	V2 <int>
185	201	0
186	202	0
187	203	0
188	204	0
189	205	0
190	206	0

6 rows

```
nrow(df_1)
```

```
## [1] 98
```

```
head(mtcars)
```

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>	am <dbl>
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1

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

6 rows | 1-10 of 12 columns

```
mtcars[mtcars$mpg > 20 , ]
```

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>	vs <dbl>	am <dbl>
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0

1-10 of 14 rows | 1-10 of 12 columns Previous **1** 2 Next

```
mtcars[mtcars$mpg > 20 , c('mpg')]
```

```
##  [1] 21.0 21.0 22.8 21.4 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26.0 30.4 21.4
```

```
mtcars[(mtcars$mpg > 20) & (mtcars$cyl != 6), c('wt','cyl','mpg')]
```

	wt <dbl>	cyl <dbl>	mpg <dbl>
Datsun 710	2.320	4	22.8
Merc 240D	3.190	4	24.4

	wt <dbl>	cyl <dbl>	mpg <dbl>
Merc 230	3.150	4	22.8
Fiat 128	2.200	4	32.4
Honda Civic	1.615	4	30.4
Toyota Corolla	1.835	4	33.9
Toyota Corona	2.465	4	21.5
Fiat X1-9	1.935	4	27.3
Porsche 914-2	2.140	4	26.0
Lotus Europa	1.513	4	30.4

1-10 of 11 rows Previous **1** 2 Next

Select all Numeric Columns or All String/Character Columns

```
library("ggplot2")
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
head(mpg)
```

manufacturer <chr>	model <chr>	displ <dbl>	year <int>	cyl <int>	trans <chr>	drv <chr>	cty <int>	hwy <int>	fl <chr>
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p
audi	a4	1.8	1999	4	manual(m5)	f	21	29	p
audi	a4	2.0	2008	4	manual(m6)	f	20	31	p
audi	a4	2.0	2008	4	auto(av)	f	21	30	p
audi	a4	2.8	1999	6	auto(l5)	f	16	26	p
audi	a4	2.8	1999	6	manual(m5)	f	18	26	p

6 rows | 1-10 of 11 columns

```
Filter(is.numeric, mpg)
```

displ <dbl>	year <int>	cyl <int>	cty <int>	hwy <int>
1.8	1999	4	18	29
1.8	1999	4	21	29

displ <dbl>	year <int>	cyl <int>	cty <int>	hwy <int>
2.0	2008	4	20	31
2.0	2008	4	21	30
2.8	1999	6	16	26
2.8	1999	6	18	26
3.1	2008	6	18	27
1.8	1999	4	18	26
1.8	1999	4	16	25
2.0	2008	4	20	28

1-10 of 234 rows Previous **1** 2 3 4 5 6 ... 24 Next

```
Filter(is.character, mpg)
```

manufacturer <chr>	model <chr>	trans <chr>	drv <chr>	fl <chr>	class <chr>
audi	a4	auto(l5)	f	p	compact
audi	a4	manual(m5)	f	p	compact
audi	a4	manual(m6)	f	p	compact
audi	a4	auto(av)	f	p	compact
audi	a4	auto(l5)	f	p	compact
audi	a4	manual(m5)	f	p	compact
audi	a4	auto(av)	f	p	compact
audi	a4 quattro	manual(m5)	4	p	compact
audi	a4 quattro	auto(l5)	4	p	compact
audi	a4 quattro	manual(m6)	4	p	compact

1-10 of 234 rows Previous **1** 2 3 4 5 6 ... 24 Next

```
M1 = matrix(c(1234,2235,67,85),nrow=2)
m1_df <- as.data.frame(M1)
colnames(m1_df) = c('roll_no','marks_mod1')
m1_df
```

roll_no <dbl>	marks_mod1 <dbl>

roll_no <dbl>	marks_mod1 <dbl>
1234	67
2235	85

2 rows

```
M2 = matrix(c(1234,2235,75,68),nrow=2)
m2_df <- as.data.frame(M2)
colnames(m2_df) = c('roll_no','marks_mod2')
m2_df
```

roll_no <dbl>	marks_mod2 <dbl>
1234	75
2235	68

2 rows

```
merge(m1_df,m2_df,by = "roll_no")
```

roll_no <dbl>	marks_mod1 <dbl>	marks_mod2 <dbl>
1234	67	75
2235	85	68

2 rows

Merge Dataframes

```
M1 = matrix(c(1234,2235,67,85),nrow=2)
m1_df <- as.data.frame(M1)
colnames(m1_df) = c('roll_no','marks_mod1')
m1_df
```

roll_no <dbl>	marks_mod1 <dbl>
1234	67
2235	85

2 rows

```
M2 = matrix(c(1234,2235,75,68),nrow=2)
m2_df <- as.data.frame(M2)
colnames(m2_df) = c('r_no','marks_mod2')
m2_df
```

r_no <dbl>	marks_mod2 <dbl>
1234	75
2235	68

2 rows

```
M3 = matrix(c(1234,2235,50,45),nrow=2)
m3_df <- as.data.frame(M3)
colnames(m3_df) = c('r_no','marks_mod3')
m3_df
```

r_no <dbl>	marks_mod3 <dbl>
1234	50
2235	45

2 rows

```
merge(m1_df,m2_df,by.x = "roll_no", by.y = "r_no")
```

roll_no <dbl>	marks_mod1 <dbl>	marks_mod2 <dbl>
1234	67	75
2235	85	68

2 rows

```
merge(m2_df,m3_df,by = "r_no")
```

r_no <dbl>	marks_mod2 <dbl>	marks_mod3 <dbl>
1234	75	50
2235	68	45

2 rows

Store Datframe to a File

```
write.csv(m2_df,"new_df.csv")
```

R_tidyverse

2023-03-08

Core tidyverse

The core tidyverse includes the packages that you're likely to use in everyday data analyses. As of tidyverse 1.3.0, the following packages are included in the core tidyverse:

ggplot2

ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.

dplyr

dplyr provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges

tidyr

tidyr provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable.

readr

readr provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes.

purrr

purrr enhances R's functional programming (FP) toolkit by providing a complete and consistent set of tools for working with functions and vectors. Once you master the basic concepts, purrr allows you to replace many for loops with code that is easier to write and more expressive

tibble

tibble is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not. Tibbles are data.frames that are lazy and surly: they do less and complain more forcing you to confront problems earlier, typically leading to cleaner, more expressive code

stringr

stringr provides a cohesive set of functions designed to make working with strings as easy as possible. It is built on top of stringi, which uses the ICU C library to provide fast, correct implementations of common string manipulations.

forcats

forcats provides a suite of useful tools that solve common problems with factors. R uses factors to handle categorical variables, variables that have a fixed and known set of possible values.

NOTE: Run library(tidyverse) to load the core tidyverse and make it available in your current R session. The

tidyverse also includes **many other packages** with more specialised usage. They are **not loaded automatically** with library(tidyverse), so you'll need to load each one with its own call to library()

R_dplyr

2023-03-08

dplyr

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges

loading dplyr

use library(dplyr)

remember the warnings about replacing base R functions

```
#install.packages("tidyverse")
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.2.2
```

```
##
## 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
```

Common Functions (verbs) of data manupulation

```
mutate() adds new variables that are functions of existing variables
select() picks variables based on their names.
filter() picks cases based on their values.
summarise() reduces multiple values down to a single summary.
arrange() changes the ordering of the rows.
```

These all combine naturally with group_by() which allows you to perform any operation “by group”.

All verbs work similarly:

```
The first argument is a data frame.

The subsequent arguments describe what to do with the data frame, using the variable names (w
ithout quotes).

The result is a new data frame.
```

load data for manupulation

Load students mathematics performance data

```
# Load students mathematics performance data
stud.data = read.csv("student-mat.csv")
nrow(stud.data) # print number of rows in data
```

```
## [1] 395
```

```
colnames(stud.data) # print all column names from data
```

```
##  [1] "school"     "sex"        "age"        "address"    "famsize"
##  [6] "Pstatus"    "Medu"       "Fedu"       "Mjob"       "Fjob"
## [11] "reason"     "guardian"   "traveltime" "studytime"  "failures"
## [16] "schoolsup"  "famsup"     "paid"       "activities" "nursery"
## [21] "higher"     "internet"   "romantic"   "famrel"     "freetime"
## [26] "goout"      "Dalc"       "Walc"       "health"     "absences"
## [31] "G1"         "G2"         "G3"
```

Filter rows with filter()

filter() allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame.

Simple condition

```
# filter all students staying in Urban area ( address = 'U')
urban.stud = filter(stud.data , address == 'U')
nrow(urban.stud)
```

```
## [1] 307
```

Match multiple values

```
# filter all students whose father's job is teacher or health
f.stud = filter(stud.data , Fjob %in% c('health' , 'teacher'))
nrow(f.stud)
```

```
## [1] 47
```

Multiple conditions (and &)

```
# filter students who are from urban area and whose father's job is teacher or health
comb.stud = filter(stud.data , address == 'U' &
                   Fjob %in% c('health' , 'teacher'))
nrow(comb.stud)
```

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

Multiple conditions (or |)

```
# filter students who are from urban area or whose Mother is at home
comb.stud = filter(stud.data , address == 'U' |
                   Mjob == 'at_home')
nrow(comb.stud)
```

```
## [1] 329
```

NOTE Common Error The easiest mistake to make is to use = instead of == when testing for equality. When this happens you'll get an informative error.

So use == when you want to compare and = when you want to assign a value

Reference : https://r4ds.had.co.nz/transform.html (https://r4ds.had.co.nz/transform.html)

Arrange rows with arrange()

arrange() changes order of rows. It takes a data frame and a set of column names (or more complicated expressions) to order by.

If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns.

Use desc() to re-order by a column in descending order

Missing values are always sorted at the end

ascending arrange

```
# arange students ascending by their age
age.stud = arrange(stud.data , age)
age.stud$age
```

```
##   [1] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
##  [26] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
##  [51] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
##  [76] 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [101] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [126] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [151] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [176] 16 16 16 16 16 16 16 16 16 16 16 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [201] 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [226] 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [251] 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [276] 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
## [301] 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
## [326] 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
## [351] 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 19 19 19 19 19 19 19 19 19
## [376] 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 20 20 20 21 22
```

descending arrange

```
# arrange students descending by their age
age.stud = arrange(stud.data , desc(age))
age.stud$age
```

```
##   [1] 22 21 20 20 20 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19 19
##  [26] 19 19 19 19 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
##  [51] 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
##  [76] 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
## [101] 18 18 18 18 18 18 18 18 18 18 18 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [126] 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [151] 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [176] 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17 17
## [201] 17 17 17 17 17 17 17 17 17 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [226] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [251] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [276] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
## [301] 16 16 16 16 16 16 16 16 16 16 16 16 16 15 15 15 15 15 15 15 15 15 15 15 15
## [326] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
## [351] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
## [376] 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
```

arrange by multiple columns

```
# arrange students ascending by their age and descending by father's education
age.stud = arrange(stud.data , age, desc(Fedu) )
head(age.stud[c('age','Fedu')])
```

```
##   age Fedu
## 1  15    4
## 2  15    4
## 3  15    4
## 4  15    4
## 5  15    4
## 6  15    4
```

Select columns with select()

It's not uncommon to get datasets with hundreds or even thousands of variables. In this case, the first challenge is often narrowing in on the variables you're actually interested in. select() allows you to rapidly zoom in on a useful subset using operations based on the names of the variables.

options in select

starts_with –> match pattern to start of column name

ends_with –> match pattern to end of column name

contains –> match a pattern in complete column name

matches –> find matching columns using regular expressions

num_range("y",1:5) –> y1,y2,y3,y4,y5

one_of –> select one of the columns of array

any_of –> used to check if the column is present or not

```
colnames(stud.data)
```

```
##  [1] "school"     "sex"        "age"        "address"    "famsize"
##  [6] "Pstatus"    "Medu"       "Fedu"       "Mjob"       "Fjob"
## [11] "reason"     "guardian"   "traveltime" "studytime"  "failures"
## [16] "schoolsup"  "famsup"     "paid"       "activities" "nursery"
## [21] "higher"     "internet"   "romantic"   "famrel"     "freetime"
## [26] "goout"      "Dalc"       "Walc"       "health"     "absences"
## [31] "G1"         "G2"         "G3"
```

In following example columns school and all columns starting with "G" are selected

```
col.select = select(stud.data, school, starts_with("G"))
names(col.select)
```

```
## [1] "school"   "guardian" "goout"    "G1"       "G2"       "G3"
```

Add new variables with mutate()

Besides selecting sets of existing columns, it's often useful to add new columns that are functions of existing

columns. That's the job of mutate().

mutate() always adds new columns at the end of your dataset so we'll start by creating a narrower dataset so we can see the new variables.

In following example new column total is added at end Total is total grades, sum of G1, G2 and G3.

```
col.select = select(stud.data, school, starts_with("G"))
col.select = mutate(col.select, total = G1 + G2 + G3)
names(col.select)
```

```
## [1] "school"   "guardian" "goout"    "G1"       "G2"       "G3"       "total"
```

Useful creation functions

There are many functions for creating new variables that you can use with mutate(). The key property is that the function must be vectorised. it must take a vector of values as input, return a vector with the same number of values as output. There's no way to list every possible function that you might use, but here's a selection of functions that are frequently useful:

- Arithmetic operators: +, -, *, /, ^.

These are all vectorised, using the so called "recycling rules". If one parameter is shorter than the other, it will be automatically extended to be the same length. This is most useful when one of the arguments is a single number: air_time / 60, hours * 60 + minute, etc.

Arithmetic operators are also useful in conjunction with the aggregate functions you'll learn about later. For example, x / sum(x) calculates the proportion of a total, and y - mean(y) computes the difference from the mean.

- Logical comparisons, <, <=, >, >=, !=, and ==
- Modular arithmetic: %/% (integer division) and %% (remainder),

where x == y * (x %/% y) + (x %% y).

Modular arithmetic is a handy tool because it allows you to break integers up into pieces. For example, you can compute hour and minute from time variable

- Ranking:

there are a number of ranking functions, but you should start with min_rank(). It does the most usual type of ranking (e.g. 1st, 2nd, 2nd, 4th). The default gives smallest values the small ranks; use desc(x) to give the largest values the smallest ranks.

Grouped summaries with summarise()

The last key verb is summarise(). It collapses a data frame to a single row.

summarise() is not terribly useful unless we pair it with group_by(). This changes the unit of analysis from the complete dataset to individual groups. Then, when you use the dplyr verbs on a grouped data frame they'll be automatically applied "by group".

```
# Find the average grades in G1 for each level of mother's education
medu.groups = group_by(stud.data, Medu)
summarize(medu.groups, avg.G1 = mean(G1))
```

```
## # A tibble: 5 × 2
##    Medu avg.G1
##   <int>  <dbl>
## 1     0  12
## 2     1   9.75
## 3     2  10.6
## 4     3  10.6
## 5     4  11.9
```

Useful summarize functions

min(x) - minimum value of vector x .

max(x) - maximum value of vector x .

mean(x) - mean value of vector x .

median(x) - median value of vector x .

quantile(x, p) - pth quantile of vector x .

sd(x) - standard deviation of vector x .

var(x) - variance of vector x .

IQR(x) - Inter Quartile Range (IQR) of vector x .

diff(range(x)) - total range of vector x .

special functions

first(x) - The first element of vector x .

last(x) - The last element of vector x .

nth(x, n) - The nth element of vector x .

n() - The number of rows in the data.frame or group of observations that summarise() describes.

n_distinct(x) - The number of unique values in vector

Combining multiple operations with the pipe

pipe operator

%>% is pipe operator in dplyr

Here output of first operation is passed as input to next step

for example x %>% f(y) means x is passed to function f(y)

Behind the scenes, x %>% f(y) turns into f(x, y), and x %>% f(y) %>% g(z) turns into g(f(x, y), z) and so on.

Advantages of pipe operator

- avoid complex codes
- avoids use of lot of temporary variables

Example of pipe usage

```
# Find the average grades in G1 for each level of mother's education
stud.data %>%
 group_by(Medu) %>%
  summarize(avg.G1 = mean(G1))
```

```
## # A tibble: 5 × 2
##    Medu avg.G1
##   <int>  <dbl>
## 1     0  12
## 2     1   9.75
## 3     2  10.6
## 4     3  10.6
## 5     4  11.9
```

R_ggplot2

GGPlot graph Format

There is a specificformat to create ggplot graphs

ggplot (data =) + < GEOM_FUNCTION > (mapping = aes(< MAPPINGS >))

geom_functions

geom_line() , geom_point() , geom_histgram(),

geom_col(), geom_bar()

geom_boxplot(), geom_violin(), geom_density()

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     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
## 15           5.8         4.0          1.2         0.2     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
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     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
## 28           5.2         3.5          1.5         0.2     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
## 32           5.4         3.4          1.5         0.4     setosa
## 33           5.2         4.1          1.5         0.1     setosa
## 34           5.5         4.2          1.4         0.2     setosa
## 35           4.9         3.1          1.5         0.2     setosa
## 36           5.0         3.2          1.2         0.2     setosa
## 37           5.5         3.5          1.3         0.2     setosa
## 38           4.9         3.6          1.4         0.1     setosa
## 39           4.4         3.0          1.3         0.2     setosa
## 40           5.1         3.4          1.5         0.2     setosa
## 41           5.0         3.5          1.3         0.3     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
## 49           5.3         3.7          1.5         0.2     setosa
## 50           5.0         3.3          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
```

```
data = iris
```

Install ggplot2 if not installed

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

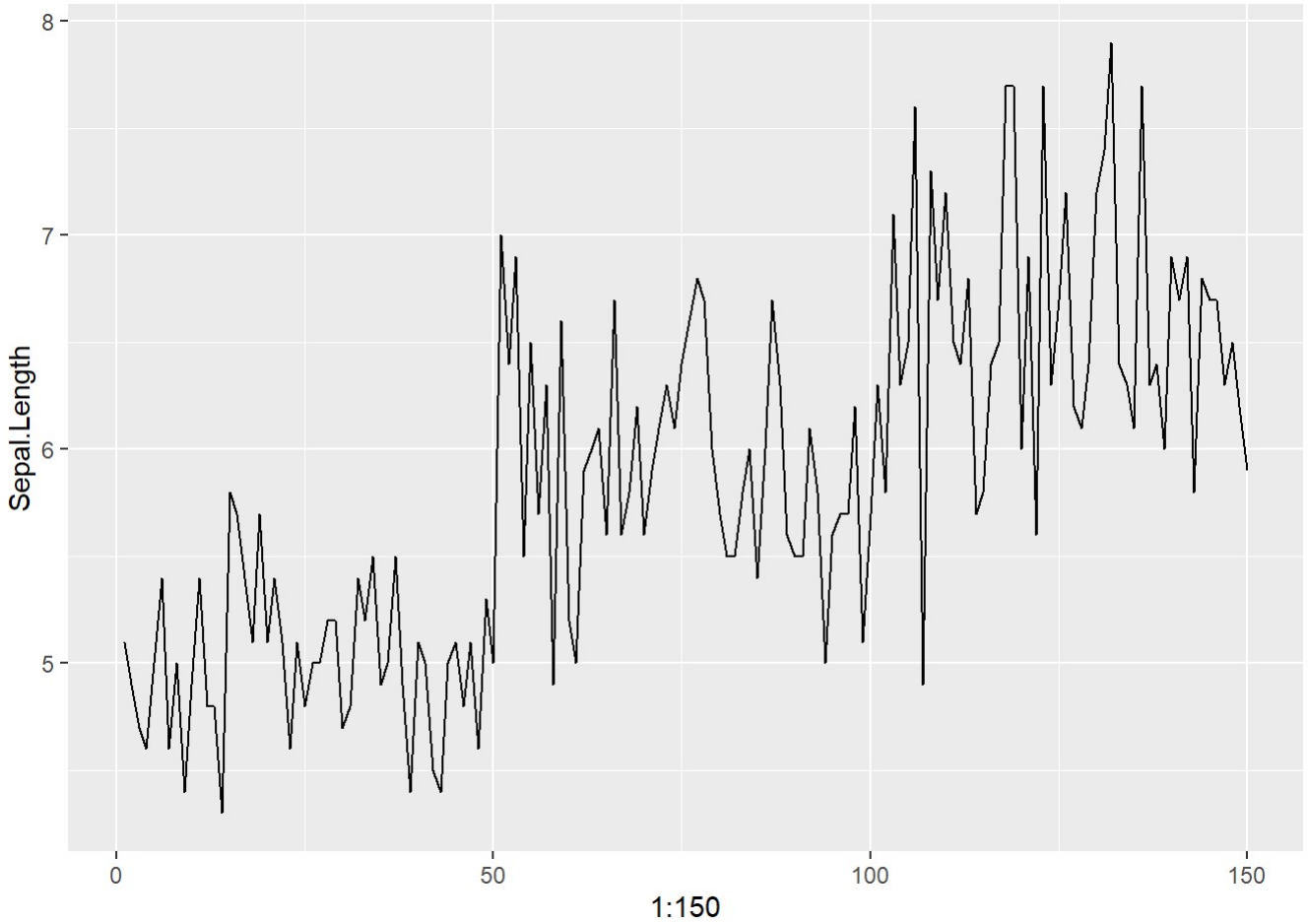
Load the ggplot2 library

```
library(ggplot2)
```

Line plot

describe the trend

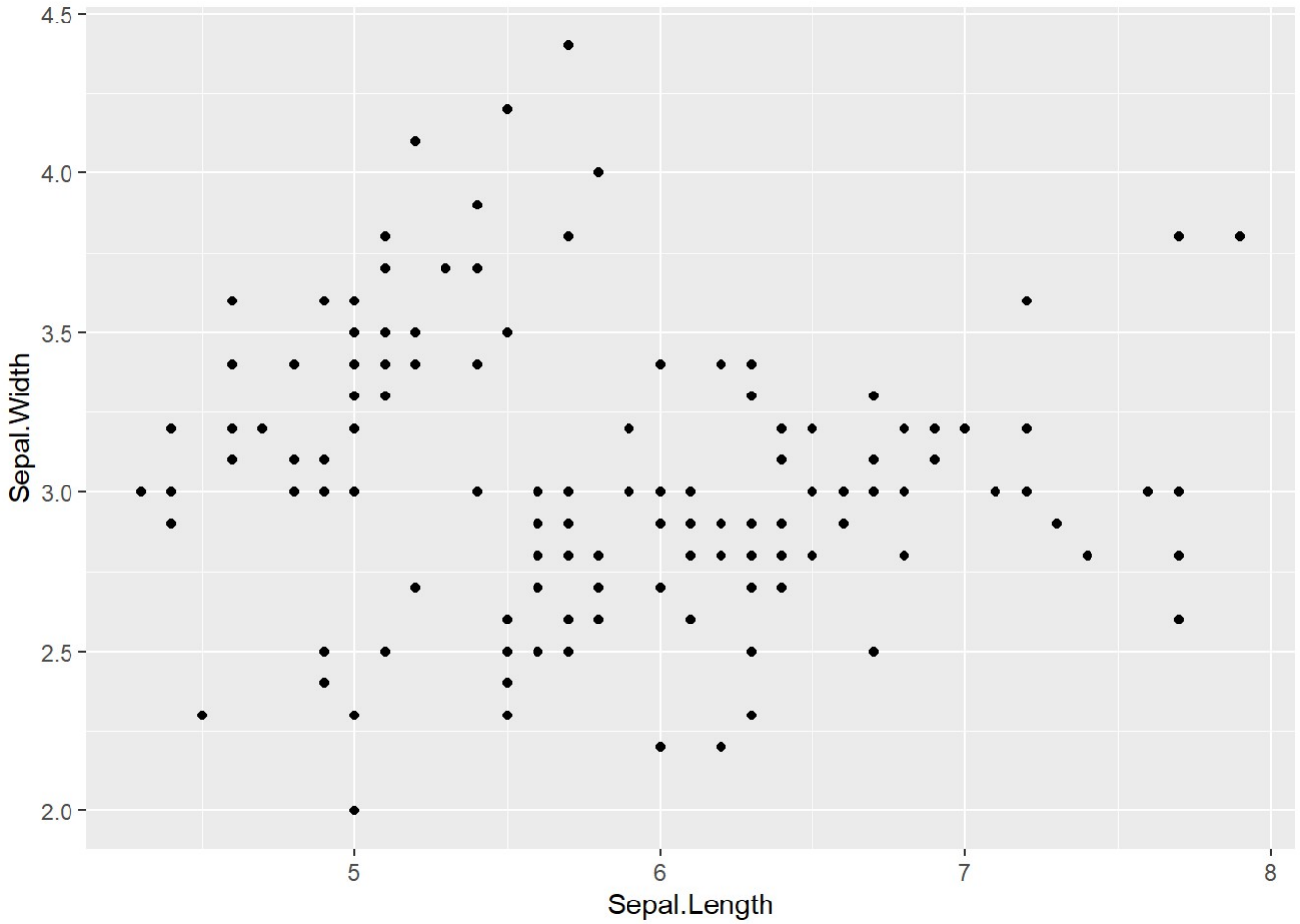
```
ggplot(data, aes(1:150, Sepal.Length)) + geom_line()
```



Scatter plot

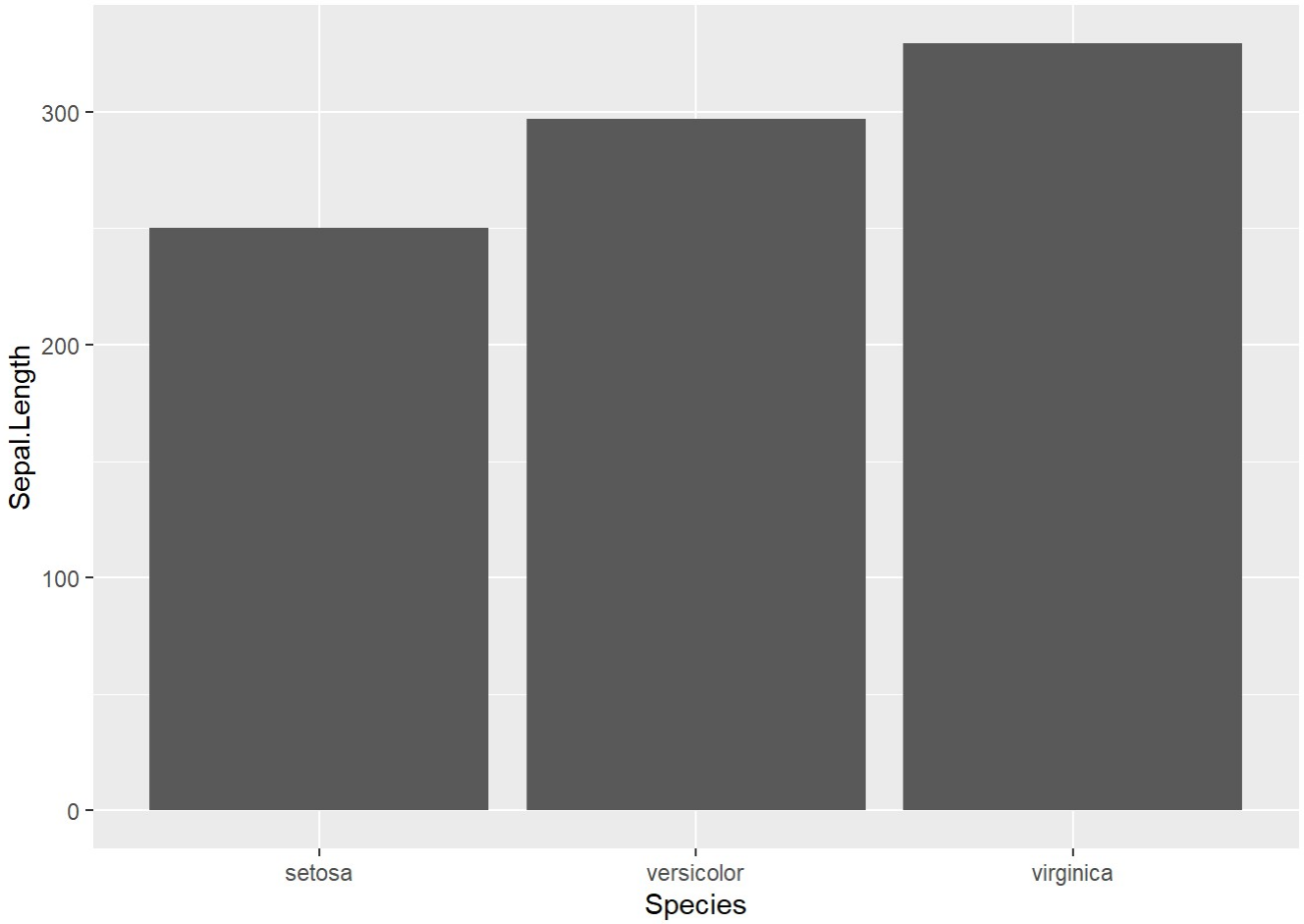
Understand relation between x & y

```
ggplot(data, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point()
```



Bar chart

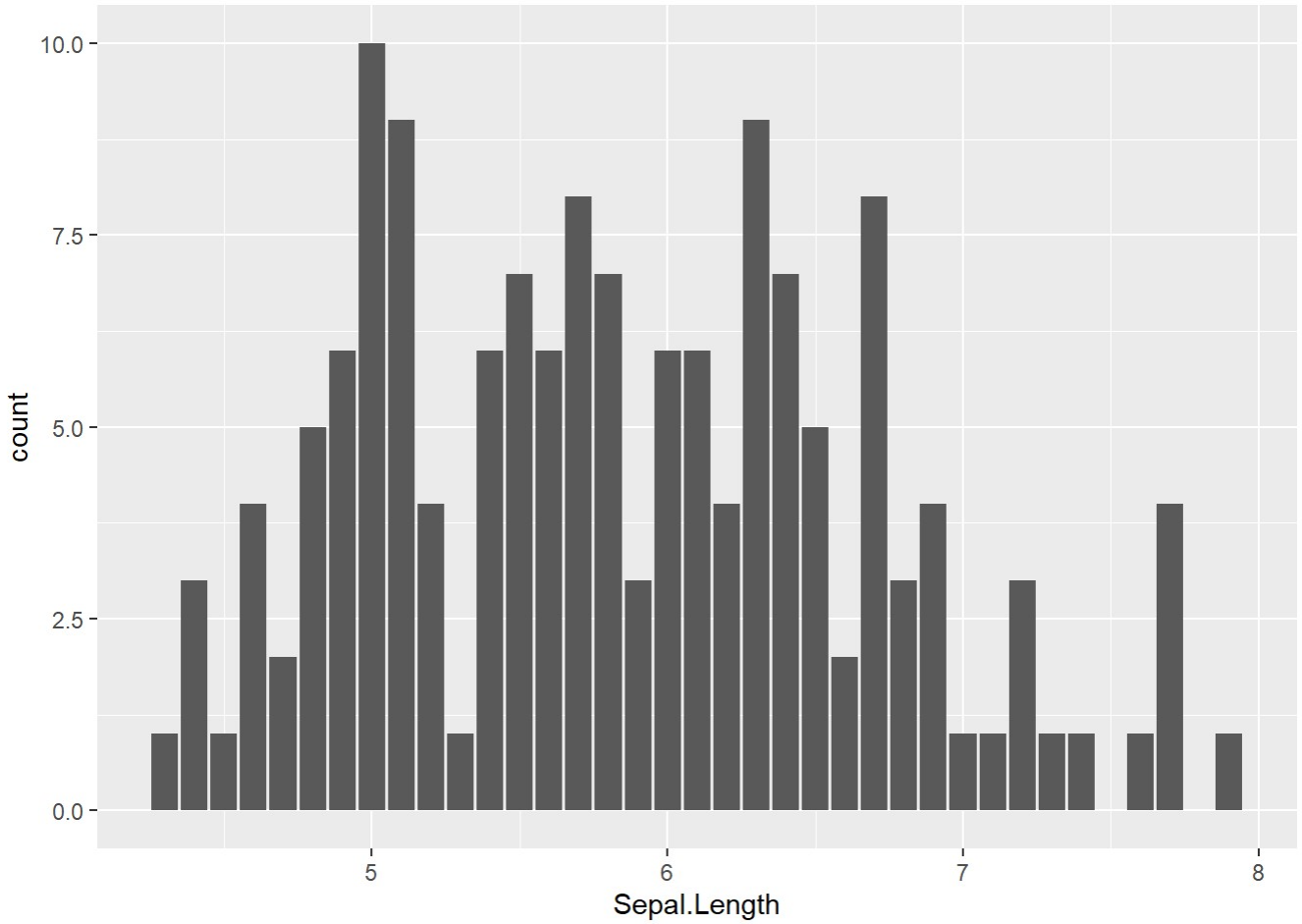
```
ggplot(data, aes(x = Species, y = Sepal.Length)) +
  geom_col()
```



Histogram

Count of each Sepal length occurrence

```
ggplot(data, aes(x = Sepal.Length)) +
  geom_bar()
```

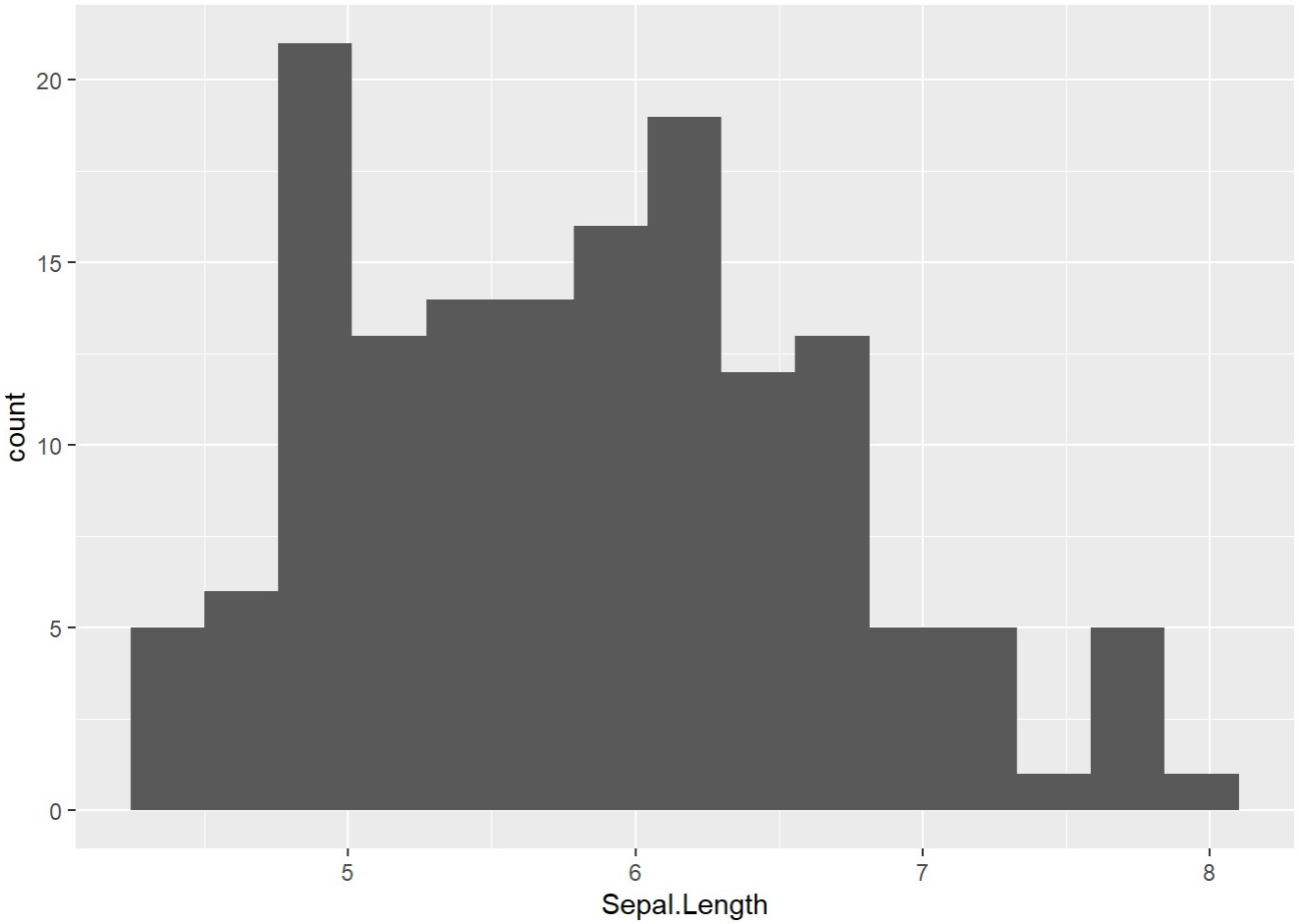


Histogram

Count of each Sepal length occurrence

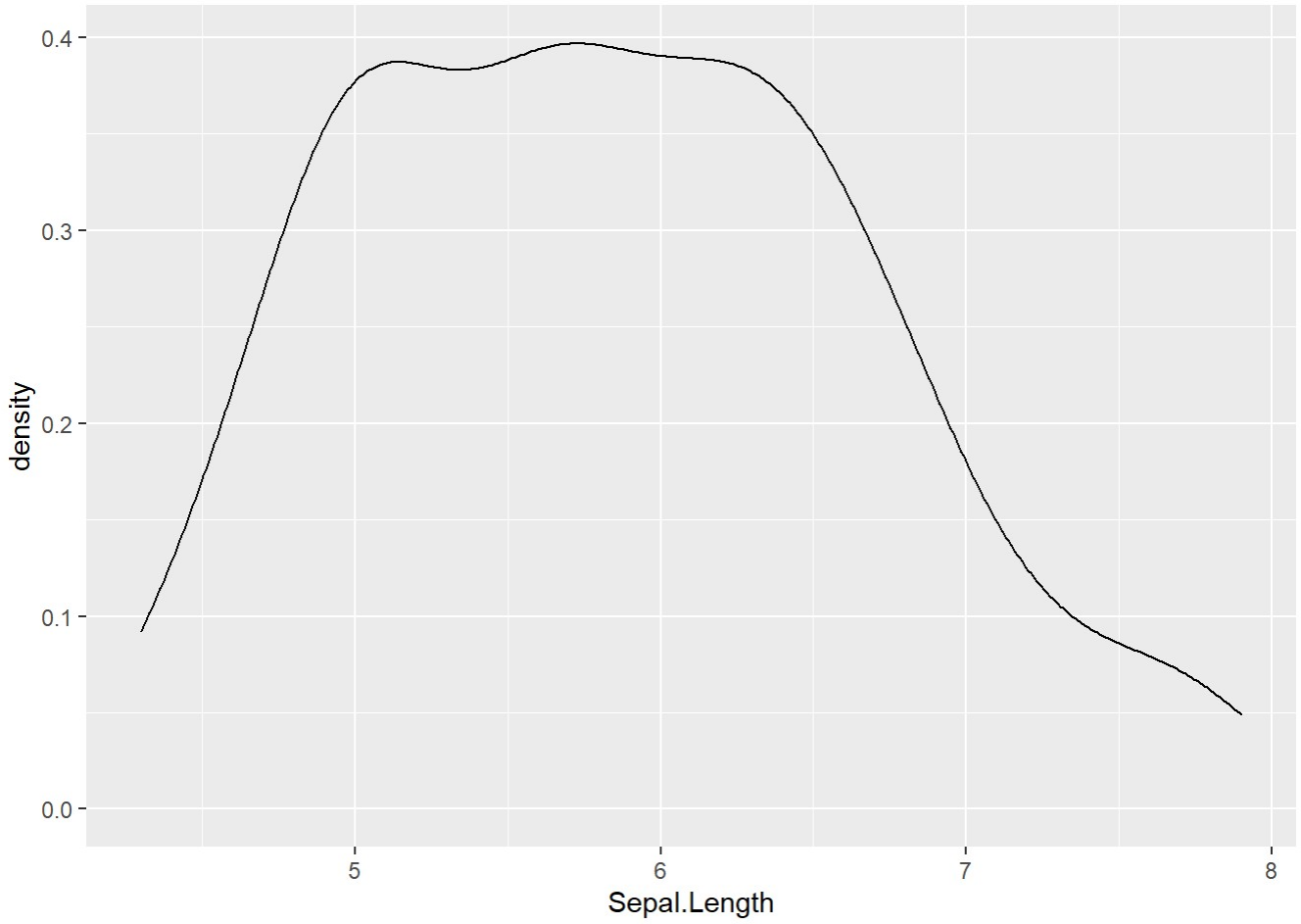
here can control the groups

```
ggplot(data, aes(Sepal.Length)) +
  geom_histogram(bins = 15)
```



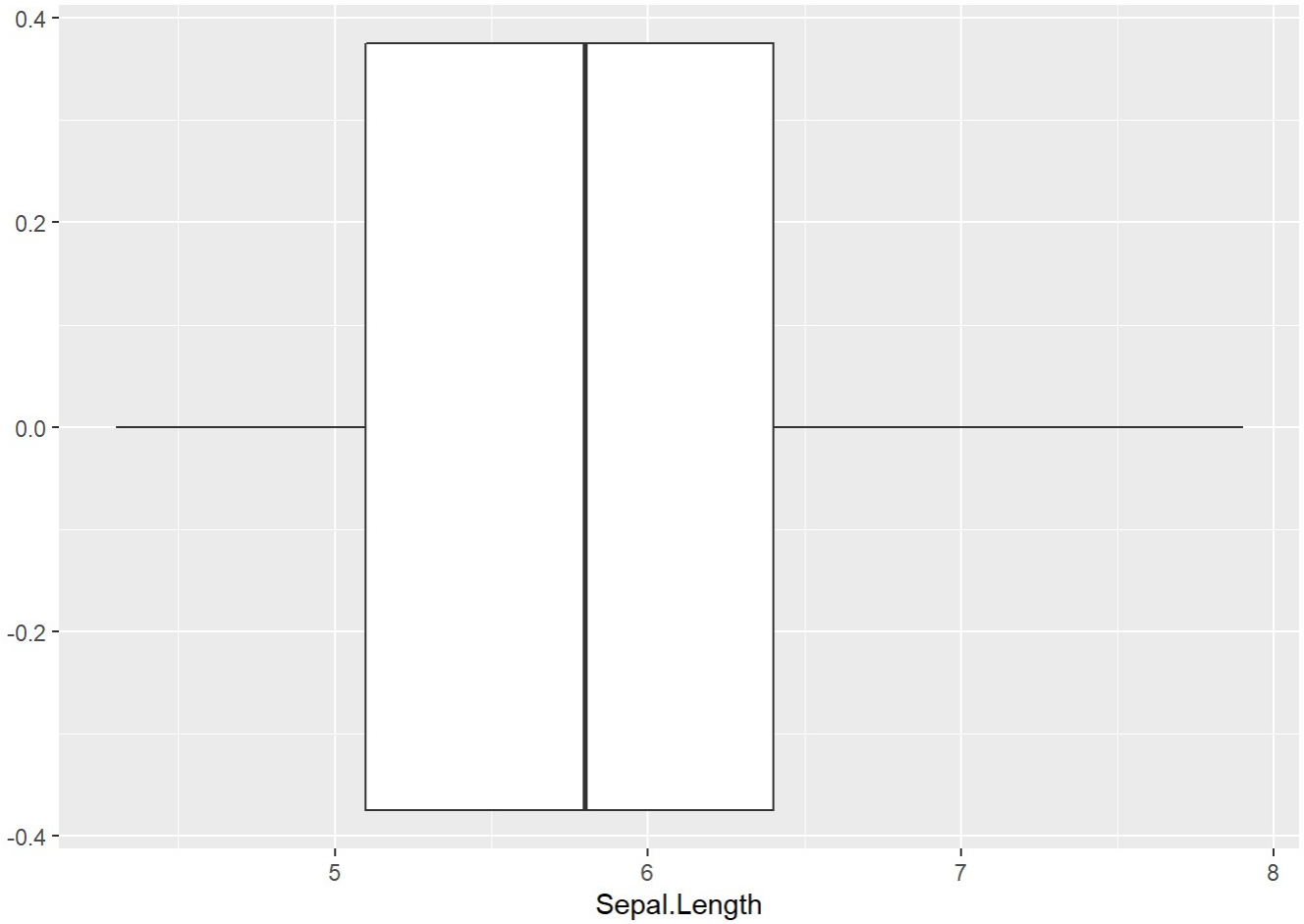
Density plot

```
ggplot(data, aes(Sepal.Length)) +
  geom_density()
```



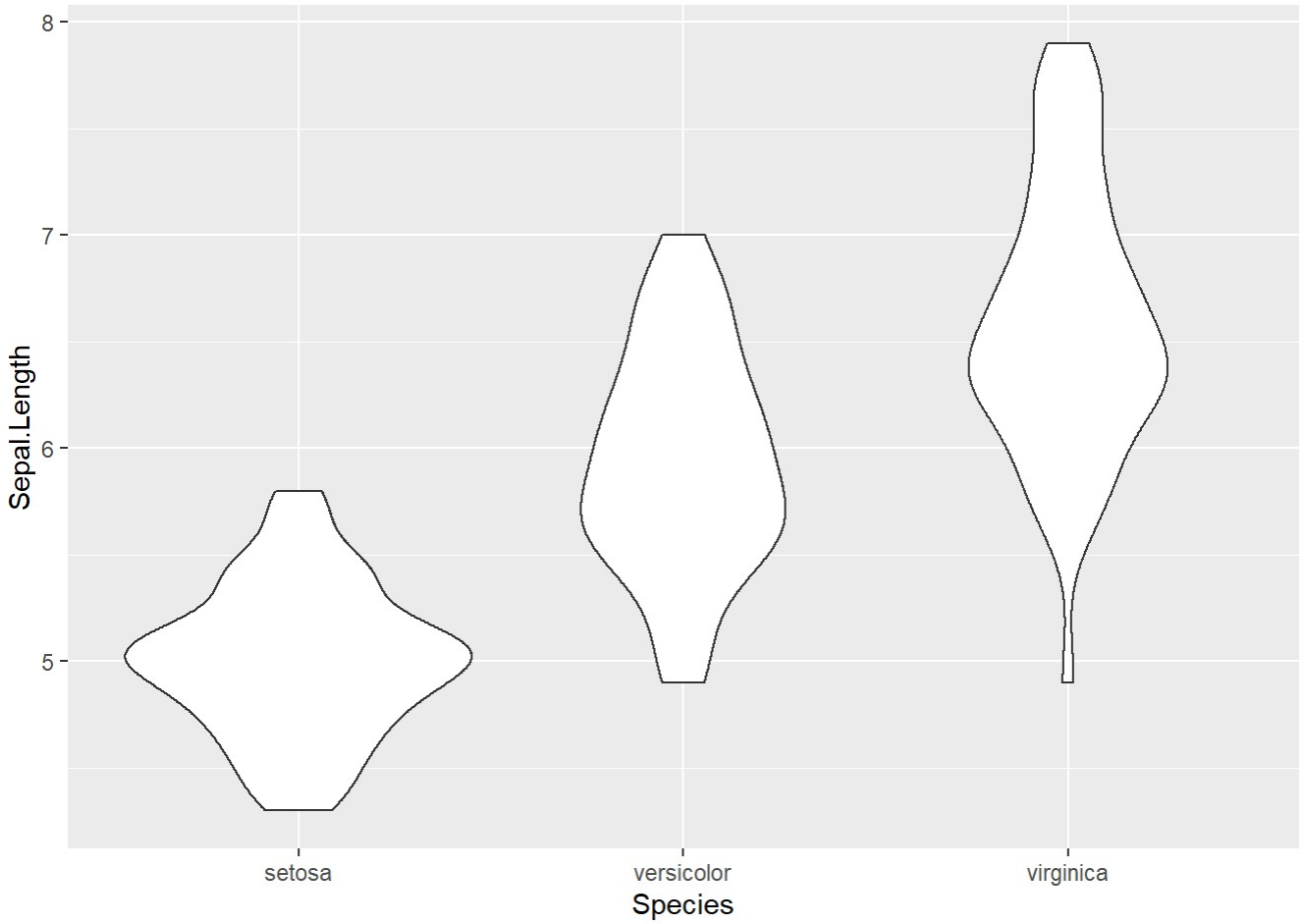
Box plot

```
ggplot(data, aes(x = Sepal.Length)) +
  geom_boxplot()
```



Understand distribution of Sepal length for different species

```
ggplot(data, aes(x = Species, y = Sepal.Length)) +
  geom_violin()
```



How to set labels and titles

```
ggplot(data, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() + labs(x='x axis label', y='y axis label',
                      title='title', subtitle='subtitle',
                      caption='image using ggplot2')
```

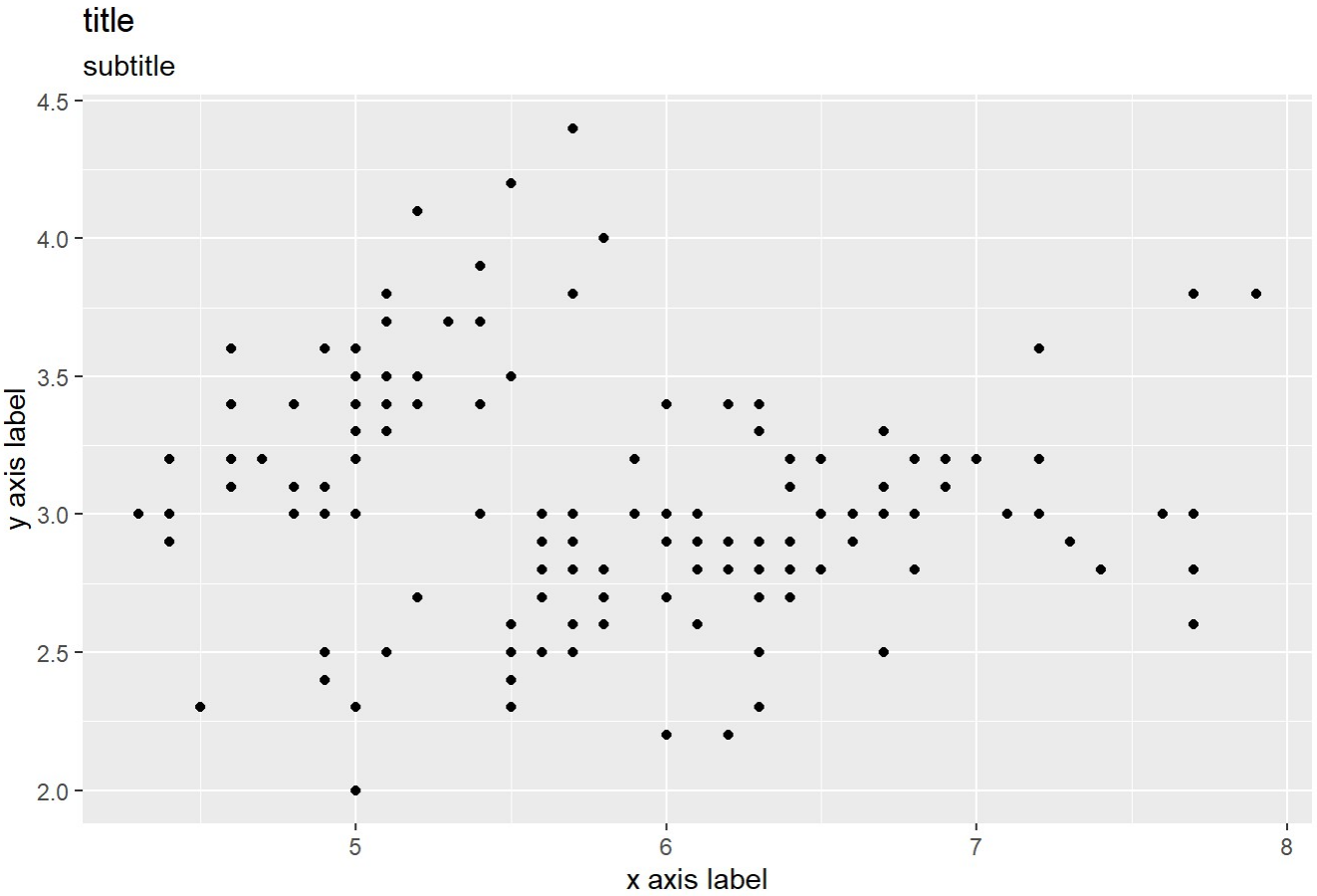
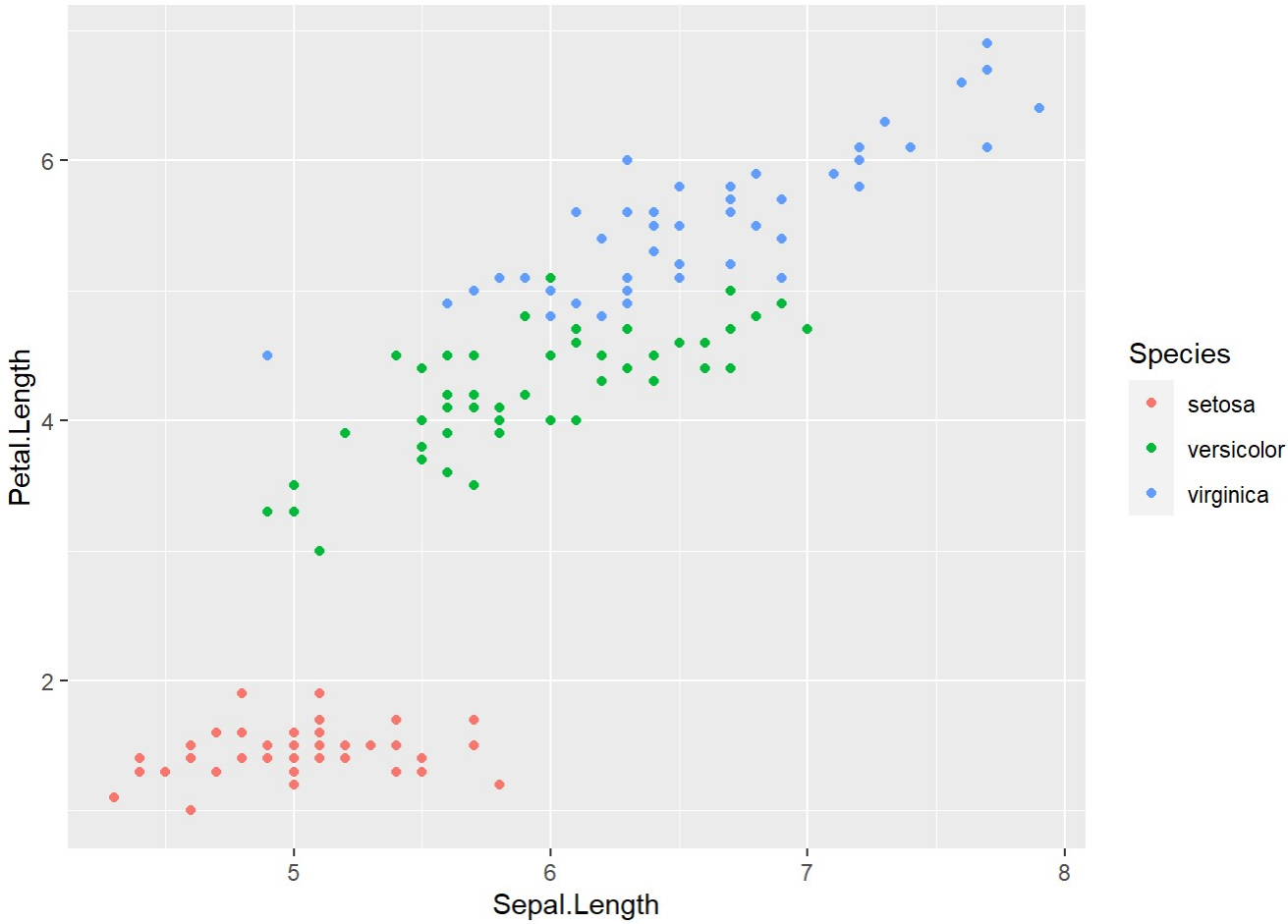


image using ggplot2

Assign the graph to a variable

```
IrisPlot <- ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) + geom_point()
print(IrisPlot)
```

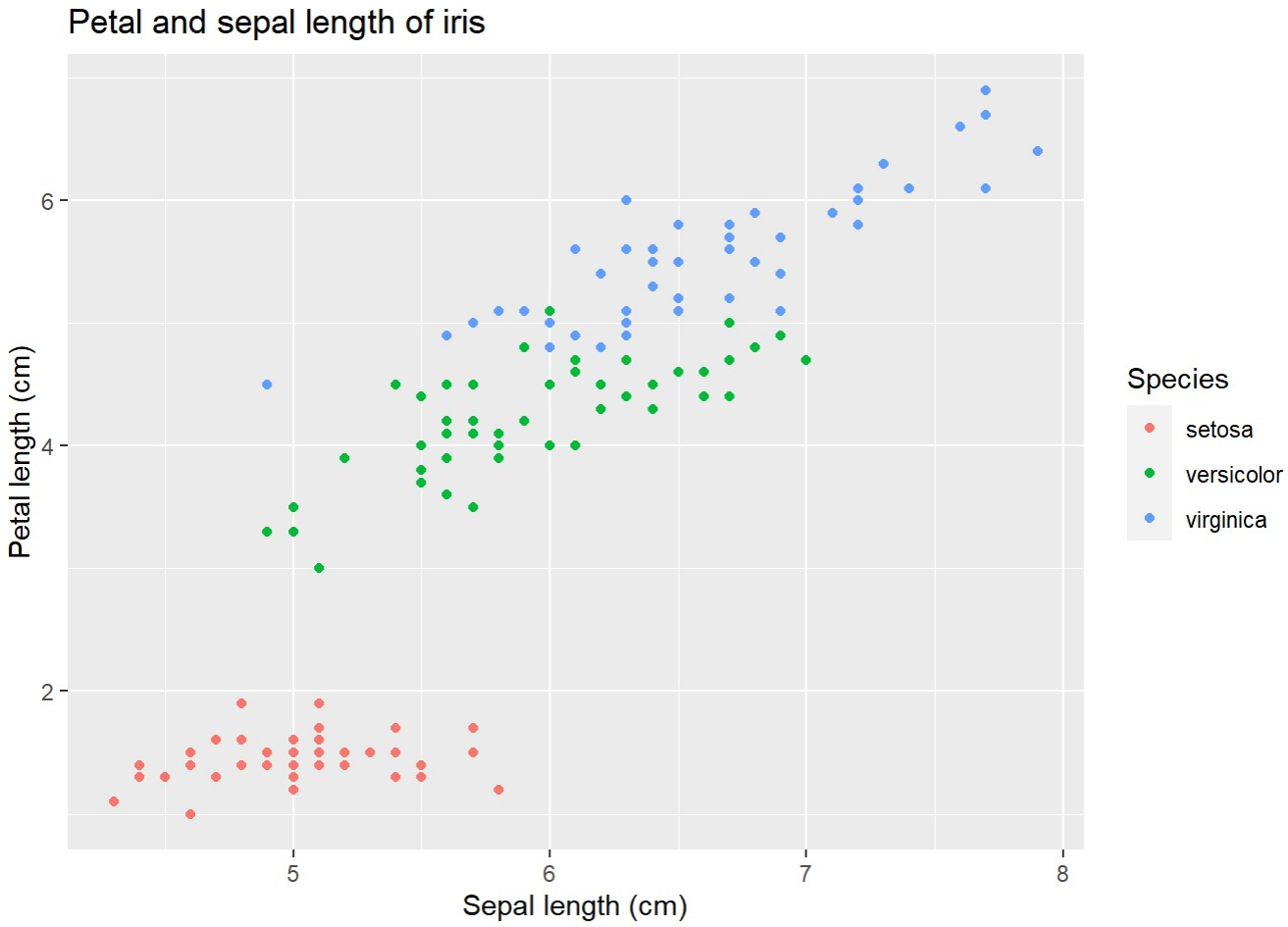


Modify x, y labels and title

labs() will allows specifying custom x axis and y axis labels

ggtitle() allows to set title for the graph

```
print(IrisPlot + labs(y="Petal length (cm)", x = "Sepal length (cm)")
   + ggtitle("Petal and sepal length of iris"))
```



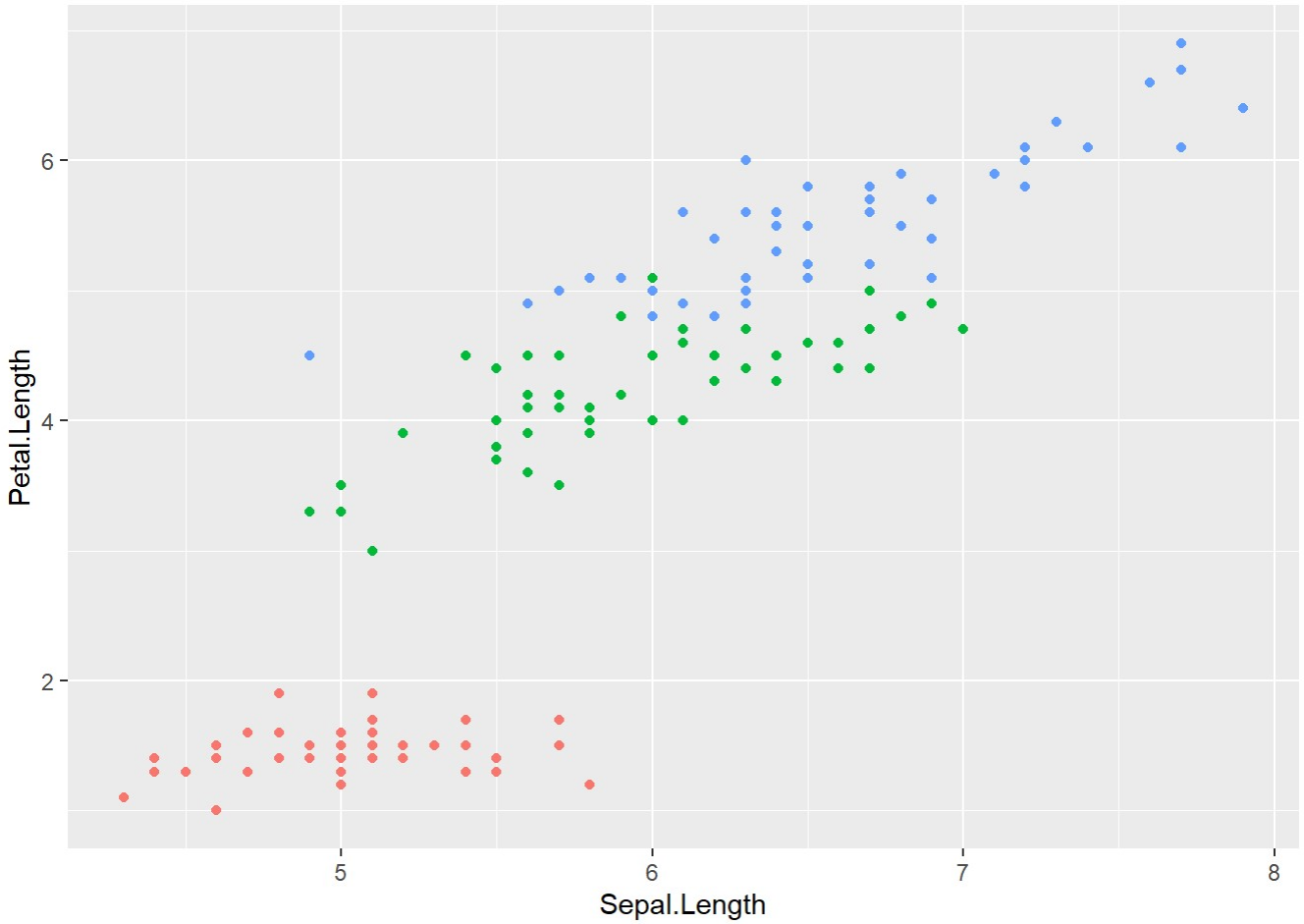
Handling Legends

Remove the legend

Remove the legend with the help of property “legend.position”

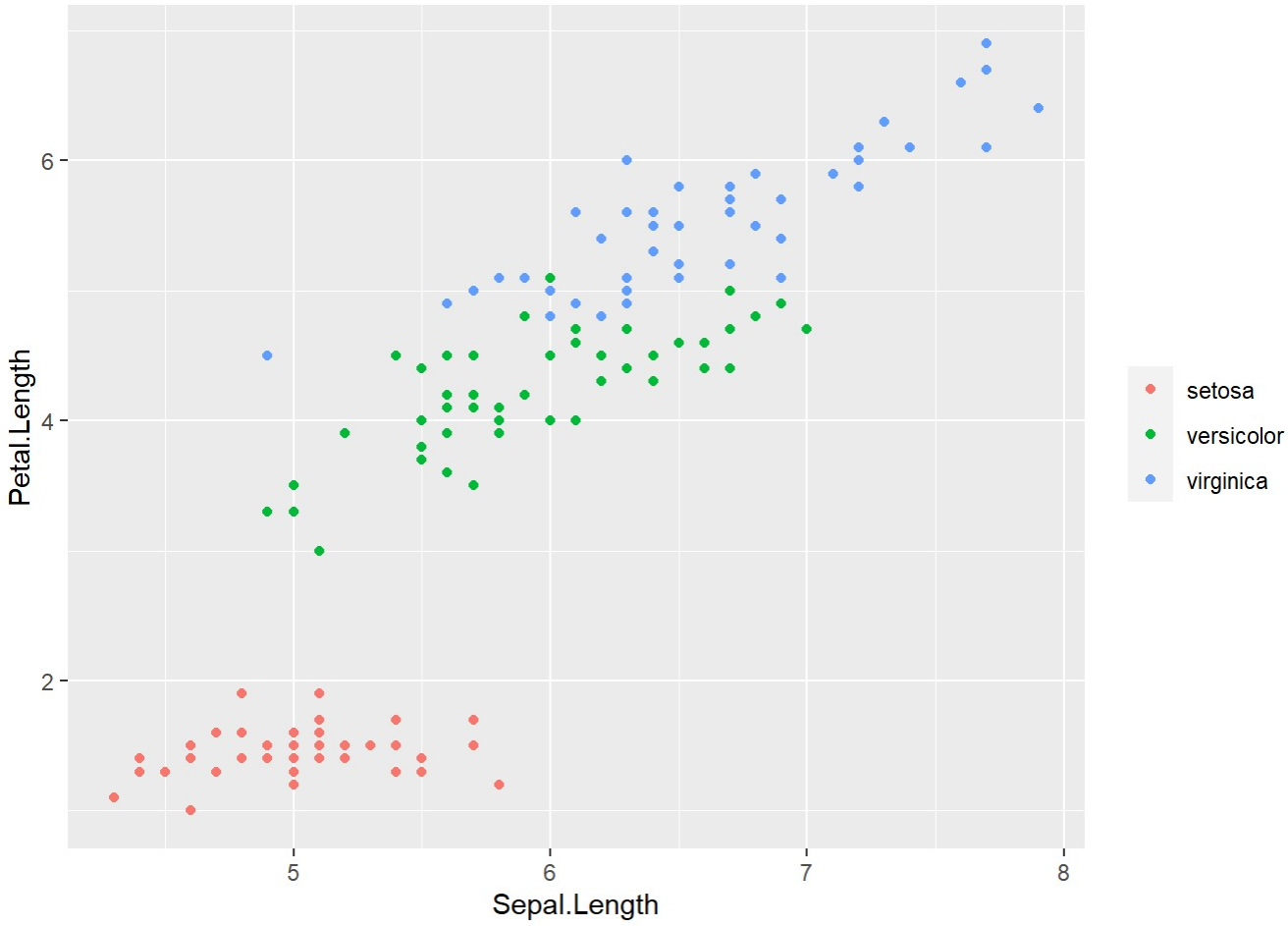
Set it to value “none” and we will not see the legends

```
# Assign basic graph to a variable
graph1 = ggplot(iris, aes(Sepal.Length, Petal.Length, colour=Species)) + geom_point()
# Remove Legend
graph1 + theme(legend.position="none")
```



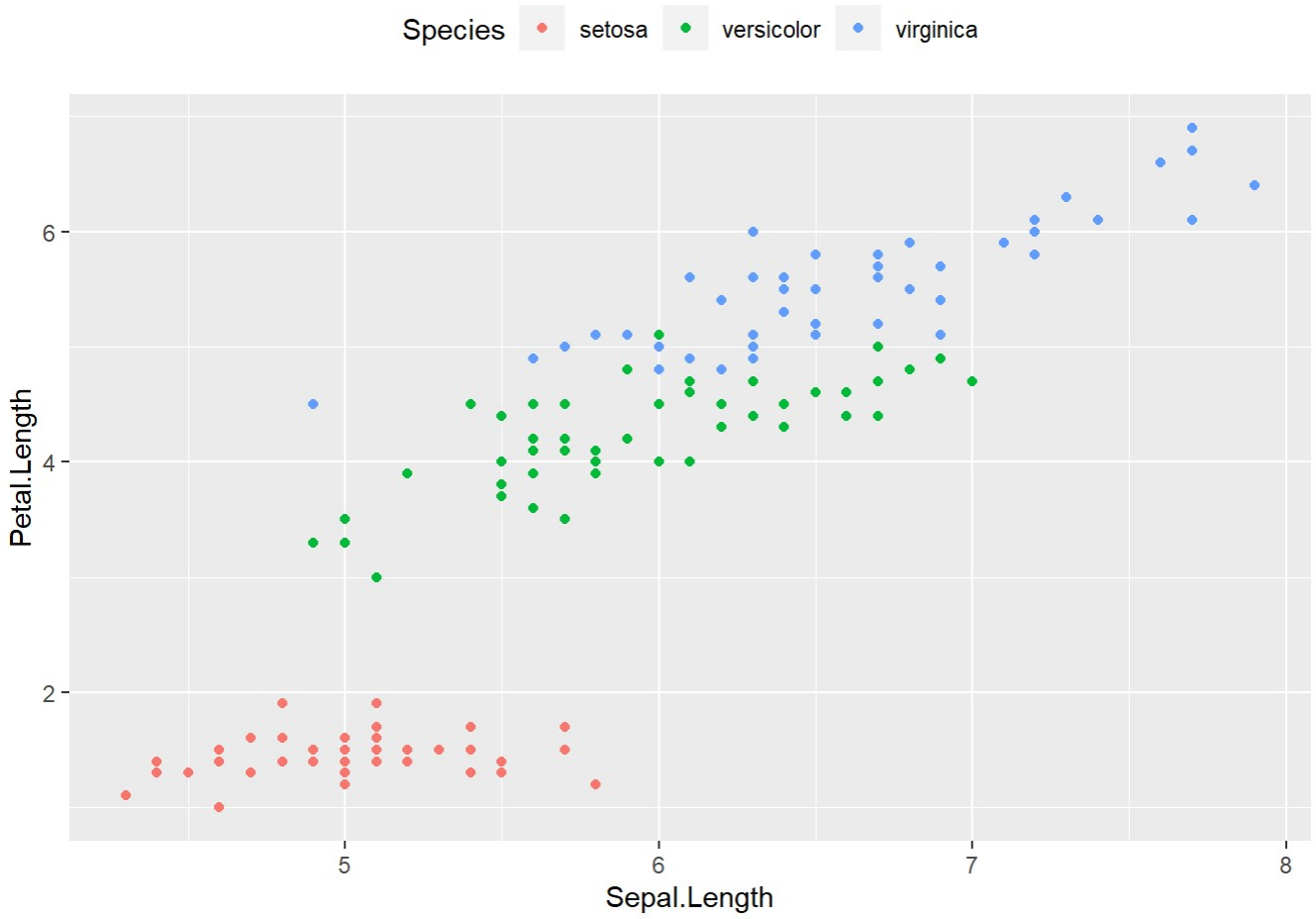
Hide the legend title

```
# Hide the legend title
graph1 + theme(legend.title=element_blank())
```



Set legend position

```
#Change the legend position
graph1 + theme(legend.position="top")
```




```
graph1 + theme(legend.position="bottom")
```


Petal.Length
6
4
2
5
6
7
8
Sepal.Length
Species
setosa
versicolor
virginica

EXTRA

R_Vectors

2023-03-06

Vectors in R

Single Element Vector

Even when you write just one value in R, it becomes a vector of length 1 called as atomic vectors

```
# Atomic vector of type character.
print("abc");
```

```
## [1] "abc"
```

```
# Atomic vector of type double.
print(12.5)
```

```
## [1] 12.5
```

```
# Atomic vector of type integer.
print(63L)
```

```
## [1] 63
```

```
# Atomic vector of type logical.
print(TRUE)
```

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

```
# Atomic vector of type complex.
print(2+3i)
```

```
## [1] 2+3i
```

```
# Atomic vector of type raw.
print(charToRaw('hello'))
```

```
## [1] 68 65 6c 6c 6f
```

Multiple Elements Vector

Using colon operator with numeric data

```
# Creating a sequence from 5 to 13.
v <- 5:13
print(v)
```

```
## [1]  5  6  7  8  9 10 11 12 13
```

```
# Creating a sequence from 6.6 to 12.6.
v <- 6.6:12.6
print(v)
```

```
## [1]  6.6  7.6  8.6  9.6 10.6 11.6 12.6
```

```
# If the final element specified does not belong to the sequence then it is discarded.
v <- 3.8:11.4
print(v)
```

```
## [1]  3.8  4.8  5.8  6.8  7.8  8.8  9.8 10.8
```

Using sequence (Seq.) operator

```
# Create vector with elements from 5 to 9 increment by 0.4.
print(seq(5, 9, by=0.4))
```

```
##  [1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

```
# Another way seq.int
print(seq.int(3, 7, 0.5))
```

```
## [1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
```

```
# Another way seq.int to generate sequence in reverse order
print(seq.int(99,90,-2))
```

```
## [1] 99 97 95 93 91
```

Sequence by seq_len function

It takes ‘n’ as parameter. And creates sequence from 1:n

```
print(seq_len(10))
```

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

Sequence by seq_along function

It is useful when we want to create sequence of length of input array

```
v = c(45,56,66,34,23,100,450)
print(v)
```

```
## [1]  45  56  66  34  23 100 450
```

```
seq_along(v)
```

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

```
print(seq_len(length(v)))
```

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

Creating Vectors by Repetition using rep() function

```
rep(1:5, 3)
```

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

```
rep(1:5, each = 3)
```

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

```
rep(1:5, times = 1:5)
```

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

```
rep(1:5, length.out = 7)
```

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

```
rep.int(1:5, 3)
```

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

```
rep_len(1:5, 7)
```

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

```
rep(1:5, times = c(5,3,8,1,1))
```

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

```
# first 5 times ,, 2nd 3 times , 3rd 8 times, 4th 1 , 5 th 1
```

Accessing Vector Elements

Elements of a Vector are accessed using indexing.

The [ ] brackets are used for indexing.

Indexing starts with position 1.

Giving a negative value in the index drops that element from result.

TRUE, FALSE or 0 and 1 can also be used for indexing.

```
# Accessing vector elements using position.
days <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- days[c(2,3,6)]
print(u)
```

```
## [1] "Mon" "Tue" "Fri"
```

```
# Accessing vector elements using logical indexing.
v <- days[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)
```

```
## [1] "Sun" "Fri"
```

```
# Accessing vector elements using negative indexing.
# So all elements except given negative indexes are selected
x <- days[c(-2,-5)]
print(x)
```

```
## [1] "Sun" "Tue" "Wed" "Fri" "Sat"
```

```
# Accessing vector elements using 0/1 indexing.
y <- days[c(0,0,0,0,0,0,1)]
print(y)
```

```
## [1] "Sun"
```

Vector Manipulation

Vector Arithmetic

```
# Create two vectors.
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)


# Vector addition.
add.result <- v1+v2
print(add.result)
```

```
## [1]  7 19  4 13  1 13
```

```
# Vector substraction.
sub.result <- v1-v2
print(sub.result)
```

```
## [1] -1 -3  4 -3 -1  9
```

```
# Vector multiplication.
multi.result <- v1*v2
print(multi.result)
```

```
## [1] 12 88  0 40  0 22
```

```
# Vector division.
divi.result <- v1/v2
print(divi.result)
```

```
## [1] 0.7500000 0.7272727       Inf 0.6250000 0.0000000 5.5000000
```

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)

# When applied with operations then v2 values are recycled
# V2 becomes c(4,11,4,11,4,11)


add.result <- v1+v2
print(add.result)
```

```
## [1]  7 19  8 16  4 22
```

```
sub.result <- v1-v2
print(sub.result)
```

```
## [1] -1 -3  0 -6 -4  0
```

Vector Element Sorting

Elements in a vector can be sorted using the sort() function.

```
v <- c(3,8,4,5,0,11, -9, 304)


# Sort the elements of the vector.
sort.result <- sort(v)
print(sort.result)
```

```
## [1]  -9   0   3   4   5   8  11 304
```

```
# Sort the elements in the reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

```
## [1] 304  11   8   5   4   3   0  -9
```

```
# Sorting character vectors.
v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
print(sort.result)
```

```
## [1] "Blue"   "Red"    "violet" "yellow"
```

```
# Sorting character vectors in reverse order.
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

```
## [1] "yellow" "violet" "Red"    "Blue"
```

R Functions

Functions in R

An R function is created by using the keyword function.

The basic syntax of an R function definition is as follows:

function_name <- function(arg_1, arg_2, …) { # Function body

}

Function Components: The different parts of a function are: • Function Name: This is the actual name of the function. It is stored in R environment as an object with this name.

• Arguments: An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

• Function Body: The function body contains a collection of statements that defines what the function does.

• Return Value: The return value of a function is the last expression in the function body to be evaluated.

User Defined Function

These are used for customization by the user. Repeated set of instructions are put in function to be used again It makes programs readable and easy to understand

An R function is created by using the keyword function.

User define function Without argument

Here function is created without any arguments. It will have process inside it. This function may or may not return any value

```
display = function(){
  print("This is a function ....")
}
display
```

```
## function(){
##   print("This is a function ....")
## }
```

```
display() # calling the function
```

```
## [1] "This is a function ...."
```

User define function With argument

Function can work better when user can some arguments

For example in following code function is taking two arguments. Here all arguments are compulsory.

```
add = function(a,b){
  # all functions can handle array / matrix/a single value/vector
  return(a+b)
}
add(23,45)
```

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

```
add(c(3,4),c(6,7))
```

```
## [1]  9 11
```

```
add(3+8i,7+4i)
```

```
## [1] 10+12i
```

With Default argument

Default argument means a default value is given to argument.

If user doesn't pass some values then default value of that argument is taken

Default arguments should be on rightmost side

All other arguments are compulsory

```
pow = function(n,p=2){
  return(n^p)
}
pow(23)
```

```
## [1] 529
```

```
pow(34,5)
```

```
## [1] 45435424
```

Calling a Function with Argument Values (by position and by name)

The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

```
# Create a function with arguments.
new.function <- function(a,b,c) {
result <- a*b+c
print(result)
}


# Call the function by position of arguments.
new.function(5,3,11)
```

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

```
# Call the function by names of the arguments.
new.function(a=11,b=5,c=3)
```

```
## [1] 58
```

Lazy Evaluation of Function

When functions are not checked for syntax errors like in compiled languages.

here syntax will give error when it is executed

R is interpreted language, so line by line code is checked and executed

In following code line 196, only one value is passed when 2 are expected, still doesn't give error because inside the function 2nd value is not used. So even if second value is not present it doesn't matter!

```
square = function(a,b){
  print(a^2)
}
square(2)
```

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

```
square(2,56)
```

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

Built in Functions

Simple examples of in-built functions are seq(), mean(), max(), sum(x)and paste(…) etc. They are directly called by user written programs. You can refer most widely used R functions on following link,

https://cran.r-project.org/doc/contrib/Short-refcard.pdf (https://cran.r-project.org/doc/contrib/Short-refcard.pdf)

```
# Create a sequence of numbers from 32 to 44.
print(seq(32,44))
```

```
##  [1] 32 33 34 35 36 37 38 39 40 41 42 43 44
```

```
# Find mean of numbers from 25 to 82.
print(mean(25:82))
```

```
## [1] 53.5
```

```
# Find sum of numbers frm 41 to 68.
print(sum(41:68))
```

```
## [1] 1526
```

```
x = c(45,-34,23,11.33)
abs(x)
```

```
## [1] 45.00 34.00 23.00 11.33
```

```
x = c(1,4,6)
log(x)
```

```
## [1] 0.000000 1.386294 1.791759
```

```
x = c(1,4,6)
exp(x) # base is e .. e^no
```

```
## [1]   2.718282  54.598150 403.428793
```

```
x = c(1,4,6)
sqrt(x)
```

```
## [1] 1.00000 2.00000 2.44949
```

```
x = c(1,4,6)
factorial(x)
```

```
## [1]   1  24 720
```

```
x = c(1,4,6)
sin(x)
```

```
## [1]  0.8414710 -0.7568025 -0.2794155
```

```
x = c(1,4,6)
cos(x)
```

```
## [1]  0.5403023 -0.6536436  0.9601703
```

```
x = c(1,0,-5,10,-34,0,0,100,-90)
sign(x)
```

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

```
x=c(10,-20,-30,80,-90,40,20.1,0)
sign(x)
```

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

Match Function match returns a vector of the positions of (first) matches of its first argument in its second.

%in% internally uses match

```
x=c(10,20,30,10,20)
y=c(20,20,40)
match(x,y)
```

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

```
match(y,x)
```

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

```
ux <- unique(x)
ux
```

```
## [1] 10 20 30
```

```
match(x, ux)
```

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

```
x=c(10:30)
y=c(10,20,60)
x
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

```
y
```

```
## [1] 10 20 60
```

```
match(y,x)
```

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

```
x=c(10,20,30,40,50)
y=c(20,4,60,50,30)
x
```

```
## [1] 10 20 30 40 50
```

```
y
```

```
## [1] 20  4 60 50 30
```

```
match(x,y)
```

```
## [1] NA  1  5 NA  4
```

Tabulate Function

tabulate() function takes the integer-valued vector bin and counts the number of times each integer occurs in it

```
tabulate(c(-5,0,3,5,4,5,4,10))
```

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

```
tabulate(c(3,5,4,5,4,100),nbins=4)
```

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

```
tabulate(c(3,5,4,8),nbins=4)
```

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

```
tabulate(c(3,5,4,8),nbins=10)
```

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

Example usage of match and tabulate to find frequency of values in given vector

```
# given a vector x
x=c(10,300,30,10,300)
x
```

```
## [1]  10 300  30  10 300
```

```
#unique values in x
ux <- unique(x)
ux
```

```
## [1]  10 300  30
```

```
# matched indexes for unique values in x
match(x, ux)
```

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

```
#element wise fequency of x
tabulate(match(x, ux))
```

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

NOTE ABOUT OPERATORS

Every operator is a function in R programming

```
'+'(90,45)
```

```
## [1] 135
```

```
'*'(34,56)
```

```
## [1] 1904
```

R Data Type Detailed Functions

Array Functions

Sum function

```
x=c(1,2,3)
sum(x)
```

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

```
x_dim2 = array(1:9,dim=c(3,3))
sum(x_dim2)
```

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

```
r = array(c(10,67,-30,0,50,60),dim=c(2,3))
r
```

```
##      [,1] [,2] [,3]
## [1,]   10  -30   50
## [2,]   67    0   60
```

```
sum(r)
```

```
## [1] 157
```

```
length(r)
```

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

```
max(r)
```

```
## [1] 67
```

```
min(r)
```

```
## [1] -30
```

Length function

```
x_dim1=c(10,20,30,40,50,60)
x_dim1
```

```
## [1] 10 20 30 40 50 60
```

```
length(x_dim1)
```

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

```
x_dim2 = array(1:9,dim=c(3,3))
x_dim2
```

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

```
length(x_dim2)
```

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

Modify the length of Array

```
x = c(10,20,30,40,50)
#x = c(34:89)
print(x)
```

```
## [1] 10 20 30 40 50
```

```
length(x) = 3
print(x)
```

```
## [1] 10 20 30
```

```
x = c(10,20,30,40,50)
length(x) = 10
print(x)
```

```
##  [1] 10 20 30 40 50 NA NA NA NA NA
```

```
x_vec = vector("numeric", 5)
length(x_vec) = 10
print(x_vec)
```

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

Modify the elements in an Array

```
x_vec = vector("numeric", 5)
x_vec
```

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

```
x_vec[3] = 100
x_vec
```

```
## [1]   0   0 100   0   0
```

```
x_vec = vector("numeric", 5)
x_vec
```

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

```
x_vec[1:3] = 89
x_vec
```

```
## [1] 89 89 89  0  0
```

```
x_vec = vector("numeric", 5)
x_vec
```

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

```
x_vec[c(1,3,5)] = 99
x_vec
```

```
## [1] 99  0 99  0 99
```

Apply function Apply function has three arguments:: X, MARGIN and FUN.

X: is input array / data MARGIN: MARGIN=1, it applies over rows, whereas with MARGIN=2, it works over columns. FUN: Function to be applied on complete array / data of values. This function can be system defined or user defined.

```
# Use apply to calculate the sum of the rows across all the matrices.
x_dim2 = array(1:12,dim=c(4,3))
print(x_dim2)
```

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

```
result <- apply(x_dim2, c(1), length)
print(result)
```

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

```
result <- apply(x_dim2, c(1), sum)
print(result)
```

```
## [1] 15 18 21 24
```

```
result <- apply(x_dim2, c(2), sum)
print(result)
```

```
## [1] 10 26 42
```

** Apply function on 3D array **

```
# Use apply to calculate the sum of the rows across all the matrices.
x_dim3 = array(1:12,dim=c(2,2,2))
print(x_dim3)
```

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

```
result <- apply(x_dim3, c(1), length)
print(result)
```

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

```
result <- apply(x_dim3, c(1), sum)
print(result)
```

```
## [1] 16 20
```

```
result <- apply(x_dim3, c(1,2), sum)
print(result)
```

```
##      [,1] [,2]
## [1,]    6   10
## [2,]    8   12
```

```
# WAP to create 3 by 3 array
# Print sum of each row and each column
x=array(seq.int(10,30,3),dim=c(3,3))
print(x)
```

```
##      [,1] [,2] [,3]
## [1,]   10   19   28
## [2,]   13   22   10
## [3,]   16   25   13
```

```
result = apply(x, 1, sum)
print(result)
```

```
## [1] 57 45 54
```

```
result = apply(x, 2, sum)
print(result)
```

```
## [1] 39 66 51
```

```
c(apply(x, 1, sum),apply(x, 2, sum))
```

```
## [1] 57 45 54 39 66 51
```

```
# Create 4,3 array
# find maximum element in each row  -> max()
# find min element in each column-> min()
```

```
x1 = array(c(1:9))
x1
```

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

```
sum(x1)
```

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

```
apply(x1, 1, sum)
```

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

```
x2 = array(c(1:9), dim=c(9,1))
x2
```

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

```
apply(x2, 1, sum)
```

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

```
apply(x2, 2, sum)
```

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

```
sum(x2)
```

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

Names to array Elements

```
x = c(apple = 1, "banana" = 2, "kiwi fruit" = 3, 4)
print(x)
```

```
##      apple     banana kiwi fruit
##          1          2          3          4
```

```
x <- 11:14  # ->
names(x) <- c("First", "Second", "Third", "Fourth")
print(x)
```

```
##  First Second  Third Fourth
##     11     12     13     14
```

```
print(names(x))
```

```
## [1] "First"  "Second" "Third"  "Fourth"
```

Indexing in Vectors

```
# vectorised operation , operation performed on every element of the array
x <- (1:10) ^ 2

print(x)
```

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

```
x[2]
```

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

```
x[c(1,5,7)]
```

```
## [1]  1 25 49
```

```
x[c(8,5,7)]
```

```
## [1] 64 25 49
```

```
x[1:3]
```

```
## [1] 1 4 9
```

```
x[-2] # exclude element at index 2
```

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

```
x[c(-1,-4)]
```

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

```
#x[c(-1,-4,5)] #--> gives error
x[c(TRUE, TRUE, FALSE)] # repeated the same array
```

```
## [1]   1   4  16  25  49  64 100
```

```
x[100]
```

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

```
x[-4.99] # floor rounding off
```

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

Indexing in 2D array

```
x=array(seq.int(10,30,3),dim=c(3,3))
print(x)
```

```
##      [,1] [,2] [,3]
## [1,]   10   19   28
## [2,]   13   22   10
## [3,]   16   25   13
```

```
x[5]
```

```
## [1] 22
```

```
x[2,1] # NOT using c() --> it is index in 2D array
```

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

```
x[c(3,1),c(2,1)] # using C() --> it is index in 1D array
```

```
##      [,1] [,2]
## [1,]   25   16
## [2,]   19   10
```

```
x[c(3,1),]
```

```
##      [,1] [,2] [,3]
## [1,]   16   25   13
## [2,]   10   19   28
```

```
x <- (1:5) ^ 2
x
```

```
## [1]  1  4  9 16 25
```

```
x[c(TRUE, FALSE, FALSE, TRUE)]
```

```
## [1]  1 16 25
```

```
x[c(TRUE)]
```

```
## [1]  1  4  9 16 25
```

```
x[c(TRUE, FALSE)] # Broadcasting
```

```
## [1]  1  9 25
```

How to use condition to select elements from an Array

```
x <- (1:5) ^ 2
x
```

```
## [1]  1  4  9 16 25
```

```
x[c(TRUE, FALSE, TRUE, FALSE, TRUE)]
```

```
## [1]  1  9 25
```

```
x>10 # vectorized operations
```

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

```
x[x>10]
```

```
## [1] 16 25
```

```
x[!x>10]
```

```
## [1] 1 4 9
```

Which Function

```
x <- (1:5) ^ 2
x
```

```
## [1]  1  4  9 16 25
```

```
print(which(x>5)) # return index of elements which satisfy the condition
```

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

```
print(x[which(x>5)])
```

```
## [1]  9 16 25
```

```
print(which.min(x)) # index of minimum element in array
```

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

```
print(x[which.min(x)])
```

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

```
print(which.max(x)) # index of max element
```

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

```
print(x[which.max(x)])
```

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

Vector Operation

Adding scalar to vector will add to all elements of that vector.

```
x = 1:5
print( x + 1)
```

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

Adding two vectors of same length will do element wise addition

```
x = 1:5
x1 = 11:15
print(x+x1)
```

```
## [1] 12 14 16 18 20
```

Adding two vectors of different length will repeat small vector over longer vector

```
x = 1:5
y = 1:10
print(x)
```

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

```
print(y)
```

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

```
print(x+y)
```

```
##  [1]  2  4  6  8 10  7  9 11 13 15
```

```
print(x-y)
```

```
##  [1]  0  0  0  0  0 -5 -5 -5 -5 -5
```

```
print(x*y)
```

```
##  [1]  1  4  9 16 25  6 14 24 36 50
```

```
x = 1:5
y = 11:18
x
```

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

```
y
```

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

```
x+y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

```
## [1] 12 14 16 18 20 17 19 21
```

Deleting Elements from an Array

There is no direct function to delete elements.

We can exclude the elements by using negative index as we did earlier.

```
x = c(10,20,30,40,50)
print(x)
```

```
## [1] 10 20 30 40 50
```

```
print("Delete second element")
```

```
## [1] "Delete second element"
```

```
x = x[-2]
print(x)
```

```
## [1] 10 30 40 50
```

```
print("Delete multiple elements, index 1 and 4 of new array")
```

```
## [1] "Delete multiple elements, index 1 and 4 of new array"
```

```
x = x[c(-1,-4)]
print(x)
```

```
## [1] 30 40
```

Cleaning the Environment

```
ls()
```

```
##  [1] "r"      "result" "x"      "x_dim1" "x_dim2" "x_dim3" "x_vec"  "x1"
##  [9] "x2"     "y"
```

```
rm(list = ls())
ls()
```

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

Controlling Visibility of Variables

This is way to use what variables are shown in current environment.

```
rm(list = ls())
x=10

print("List after creating x")
```

```
## [1] "List after creating x"
```

```
ls()
```

```
## [1] "x"
```

```
.xyz= 90

print("List after creating .xyz")
```

```
## [1] "List after creating .xyz"
```

```
ls()
```

```
## [1] "x"
```

```
print("List of all variables")
```

```
## [1] "List of all variables"
```

```
ls(all.names = TRUE)
```

```
## [1] ".xyz" "x"
```

List all variables of Environment When you want to remind yourself of all the variables you've created in the environment, use ls().

```
ls()
```

```
## [1] "x"
```

Deleting Variables from Environment Variable can be deleted using rm() You can use any variable name you have created

```
ls()
```

```
## [1] "x"
```

```
rm(yourname)
```

```
## Warning in rm(yourname): object 'yourname' not found
```

```
ls()
```

```
## [1] "x"
```

List Operations and Functions

```
x <- list(1, "a", c(1,2,3), 1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [1] 1+4i
```

```
x[2]
```

```
## [[1]]
## [1] "a"
```

```
x[[2]]
```

```
## [1] "a"
```

```
x <- list(1, "a", c(1,2,3), 1+4i)
x[1] = 3
x[[3]] = "name" #c(1,3,5)
x
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] "name"
##
## [[4]]
## [1] 1+4i
```

```
x <- list(1, "a", c(1,2,3), 1+4i)
x[3]= c(10)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 10
##
## [[4]]
## [1] 1+4i
```

```
x <- list(1, "a", c(1,2,3), 1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [1] 1+4i
```

```
x[[3]][3]
```

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

```
x[[3]][-2] #
```

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

```
x[[2]]
```

```
## [1] "a"
```

```
x= list(1,"kk")
y = list(c(1,2,3),list(78,78),2+67i,3,"lll")
a = array(c(x,y))
class(a)
```

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

```
a
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "kk"
##
## [[3]]
## [1] 1 2 3
##
## [[4]]
## [[4]][[1]]
## [1] 78
##
## [[4]][[2]]
## [1] 78
##
##
## [[5]]
## [1] 2+67i
##
## [[6]]
## [1] 3
##
## [[7]]
## [1] "lll"
```

return all the keys in list

```
# list is just like key value pair
xlist <- list(a = "Shantanu Pathak", b = 1:10, data = head(iris))
names(xlist) # return all the keys in list
```

```
## [1] "a"    "b"    "data"
```

```
xlist$a
```

```
## [1] "Shantanu Pathak"
```

```
xlist$b
```

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

```
xlist$data
```

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

6 rows

Factors Functions

factor- type vector contains a set of numeric codes with character-valued levels.

Regardless of the levels/labels of the factor, the numeric storage is an integer with 1 corresponding to the first level (in alph-order)

```
students = factor(c(100,0,100,0,0,0), levels = c(0, 100), labels = c("boy", "girl"))
students
```

```
## [1] girl boy  girl boy  boy  boy
## Levels: boy girl
```

```
as.numeric(students)
```

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

```
 1 + as.numeric(students)
```

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

```
grades = factor(c("A","B","A+","A","B","B","A","A+"))
grades
```

```
## [1] A  B  A+ A  B  B  A  A+
## Levels: A A+ B
```

```
as.numeric(grades)
```

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

```
nlevels(grades)
```

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

```
iris$Species
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa
##   [7] setosa     setosa     setosa     setosa     setosa     setosa
##  [13] setosa     setosa     setosa     setosa     setosa     setosa
##  [19] setosa     setosa     setosa     setosa     setosa     setosa
##  [25] setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa
##  [37] setosa     setosa     setosa     setosa     setosa     setosa
##  [43] setosa     setosa     setosa     setosa     setosa     setosa
##  [49] setosa     setosa     versicolor versicolor versicolor versicolor
##  [55] versicolor versicolor versicolor versicolor versicolor versicolor
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor
##  [67] versicolor versicolor versicolor versicolor versicolor versicolor
##  [73] versicolor versicolor versicolor versicolor versicolor versicolor
##  [79] versicolor versicolor versicolor versicolor versicolor versicolor
##  [85] versicolor versicolor versicolor versicolor versicolor versicolor
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor
##  [97] versicolor versicolor versicolor versicolor virginica  virginica
## [103] virginica  virginica  virginica  virginica  virginica  virginica
## [109] virginica  virginica  virginica  virginica  virginica  virginica
## [115] virginica  virginica  virginica  virginica  virginica  virginica
## [121] virginica  virginica  virginica  virginica  virginica  virginica
## [127] virginica  virginica  virginica  virginica  virginica  virginica
## [133] virginica  virginica  virginica  virginica  virginica  virginica
## [139] virginica  virginica  virginica  virginica  virginica  virginica
## [145] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

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

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

```
nlevels(iris$Species)
```

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

```
as.numeric(iris$Species)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

```
designation <- factor(c("Manager", "Team Lead","SME"), ordered =TRUE)
designation
```

```
## [1] Manager   Team Lead SME
## Levels: Manager < SME < Team Lead
```

```
designation <- factor(c("Manager", "Team Lead","SME"), ordered =TRUE, levels = c("SME", "Team
Lead","Manager"))
designation
```

```
## [1] Manager   Team Lead SME
## Levels: SME < Team Lead < Manager
```

```
as.numeric(designation)
```

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

```
val<-factor(c("r1","r2","r1","r2","r2","r3","r1"),ordered = TRUE,levels = c("r3", "r2","r1"))
val
```

```
## [1] r1 r2 r1 r2 r2 r3 r1
## Levels: r3 < r2 < r1
```

```
as.numeric(val)
```

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