**Linear Regression in R**

▷ Regression analysis is a very widely used statistical tool to establish a relationship model between two variables.

▷ One of these variable is called predictor variable whose value is gathered through experiments.

▷ The other variable is called response variable whose value is derived from the predictor variable.

▷ Linear regression is used to predict the value of an outcome variable *Y* based on one or more input predictor variables *X*.

▷ Mathematically a linear relationship represents a straight line when plotted as a graph.

▷ The general mathematical equation for a linear regression is −

## 𝒚 =b0 + b1 \* 𝒙

### ▷ Following is the description of the parameters used −

* y is the response variable.
* x is the predictor variable.
* b1 – slope
* b0 - intercept
* Collectively, they are called *regression coefﬁcients*.

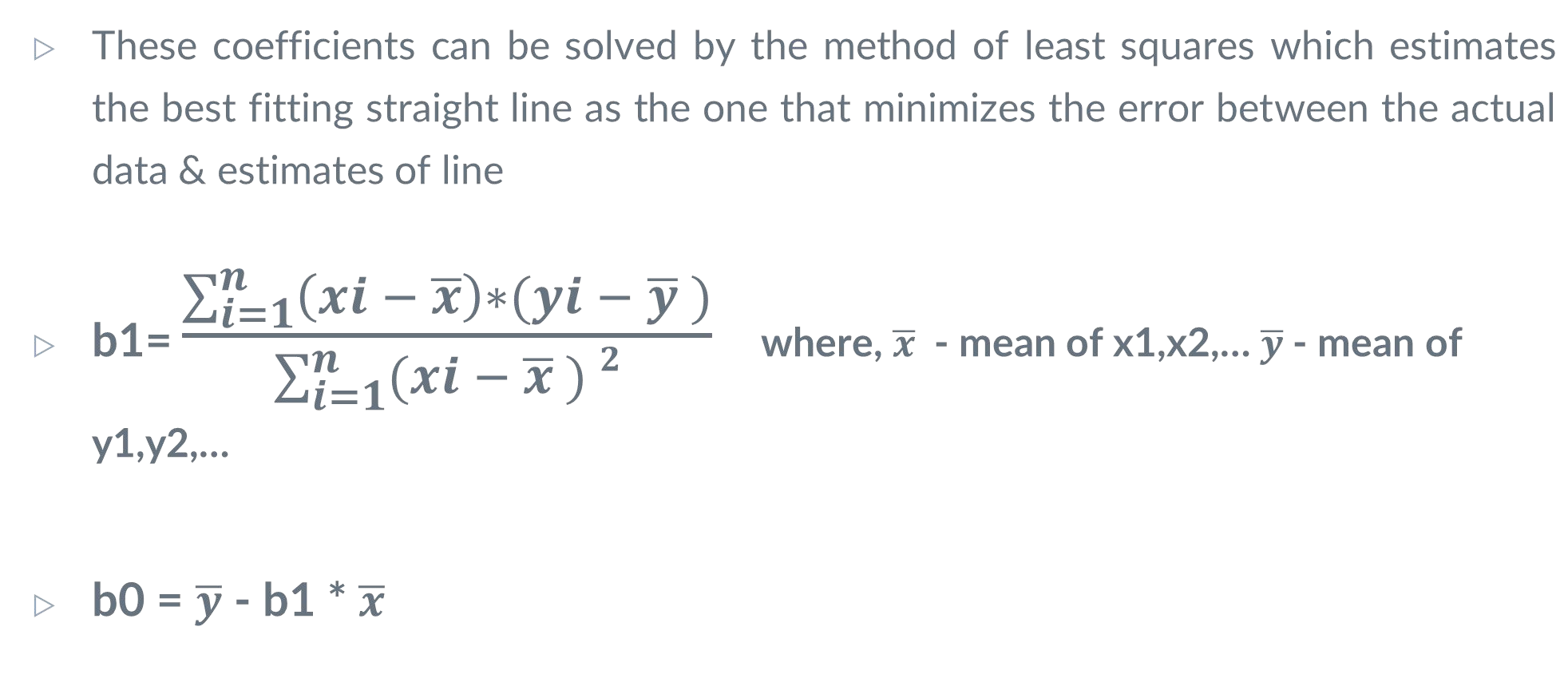
### ▷ For example, we want to predict weight (y) from height (x), the linear regression model can be represented by the following equation

Weight= b0 + b1 \* height

### ▷ b1 is called slope because it deﬁnes the slope of the line or how x translates into a y

i.e by how much y is affected by change in x

### ▷ The goal is to ﬁnd best estimates for the coefﬁcients to minimize the error in predicting y from x



▷

### If b1 > 0, then x(predictor) and y(target) have a positive relationship.

* + That is increase in x will increase y.

### If b1 < 0, then x(predictor) and y(target) have a negative relationship.

* + That is increase in x will decrease y.

## ▷ Implementation in R

##### Step 1 – load packages – tidyverse, ggplot2

library(ggplot2) library(tidyverse)

##### Step 2 – import data from CSV ﬁle – “linear.csv”

rldata<-read.csv("linear.csv") rldata

##### Step 3 - Create a relationship model using the lm() functions in R.

* + lm() Function
  + This function creates the relationship model between the predictor and the response variable.
  + Syntax
    - lm([target variable] ~ [predictor variables], data = [data source])

##### Create Relationship Model & get the Coefﬁcients

relation<-lm(weight~height,data=rldata) print(relation)

##### Get the Summary of the Relationship

summary(relation)

##### Step 4 - To predict the weight of new persons, use the predict() function in R.

* + The predict() function in R is used to predict the values based on the input data.
  + Syntax
  + The basic syntax for predict() in linear regression is −

### predict(object, newdata)

* + Following is the description of the parameters used −
    - object is the formula which is already created using the lm() function.
    - newdata is the vector containing the new value for predictor variable.

a <- data.frame(height = 170) result <- predict(relation,a) print(result)

##### Step 5 - Visualize the Regression Graphically

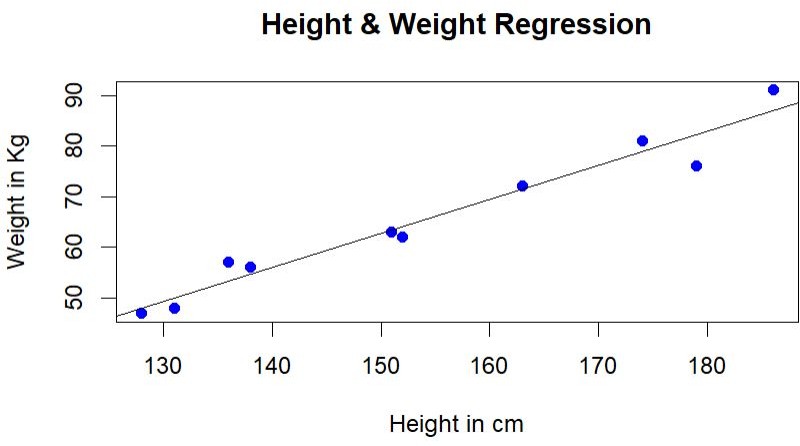
* + The plot() takes a data frame and plots the variables on it. In this case, it plots the height against the weight of the person.

plot(rldata,col = "blue",pch = 16,main = "Height & Weight Regression",ylab = "Weight in Kg",xlab = "Height in cm")

* + Then, add the line made by the linear regression with the command abline().
  + The R function abline() can be used to add vertical, horizontal or regression lines to a graph.
  + abline(a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL, coef = NULL, col = NULL, lty = NULL, lwd= NULL)
  + The abline function can accept several arguments, with defaults of NULL:
* a – The intercept of the line, as a single value
* b – The slope of a line, as a single value
* h – the y-value(s) for horizontal line(s)
* v – the x-value(s) for vertical line(s).
* coef – a vector of length two giving the intercept and slope
* reg – a regression object generated by the coef method; intended to simplifying passing trend-lines from a regression calculation
* col – optional color parameter for the line
* lty – optional line type parameter
* lwd – optional line width parameter

abline(relation)

* + When we execute the above code, it produces the following result −



# Market Basket Analysis in R

▷ The increasing volume of data and the growing importance of retail analytics made it easy for retailers to know their customers better.

▷ Data can help retailers to understand customer behavior, plan and promote products, increase sales, improve customer experience, and optimize supply chain performance.

▷ There are many algorithms and techniques used in retail that help uncover better insights and predict future events.

▷ One of the key and widely used techniques in retail is Market Basket Analysis.

▷ It works by searching for combinations of items that often happen in transactions together.

▷ Market Basket Analysis is a technique that is used to discover the association between items.

▷ In simplest terms, it allows retailers to identify a relationship between items that generally people buy together.

▷ For instance, if one person buys ‘bread’, he/she more likely to buy ‘butter’ or ‘jam’ which is predicted as a ‘go-along’ item with the purchase

▷ To implement this, associate rule mining is used.

▷ Association Rule Mining is a rule-based machine learning method to ﬁnd associations and relationships between large sets of items.

▷ This rule also shows how frequently an item occurs in the itemset based on the occurrences of other items in a transaction.

▷ Association rules are widely used to analyze basket or transaction data to discover strong rules based on the interestingness and frequency of occurrences.

▷ Association rules can be understood as the “if this, then that” rule.

▷ For example, if a user buys coffee and sugar, then he/she is likely to buy milk.

▷ Multiple techniques and algorithms are being used in Market Basket Analysis.

▷ One of the main objectives is to predict the likelihood of items being purchased together by users.

▷ APRIORI is the by far widely-used and well-known association rule algorithm.

▷ It ﬁnds frequent itemsets in transactions and identiﬁes association rules between those items.

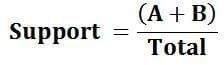
▷ It scans the database many times which leads to increased time and reduced performance as it is a computationally expensive step because of a large database.

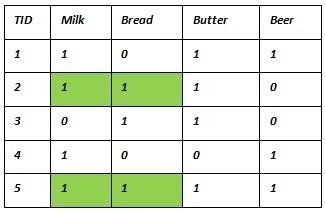
▷ The association rule has primarily three measures to decide the degree of conﬁdence, these are:

* Support
* Conﬁdence
* Lift

▷ Support:

* This is one of the important measures to determine how frequently an itemset occurs in the transaction as a percentage of all transactions.
* Support is the number of transactions that include both {A} and {B} parts as a percentage of the total number of transactions.



▷ Consider the following dataset

▷ Example to ﬁnd support for itemset {milk,bread}

▷ Support (milk->bread) = No. of transactions containing milk and bread/Total no of Transaction)

▷ Out of 5 transaction 2 transactions have both milk and bread.

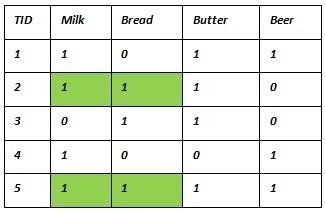
▷ So, Support (milk->bread) = (2/5) = 0.4

▷ means milk and bread are purchased together occur in 40% of all transactions.

### ▷ Conﬁdence:

* This rule is the ratio of the number of transactions that include items in {A} and {B} to the number of transactions that include items in {A}.
* It can be understood as to how often items in B appear in transactions that contain A only. It is a conditional probability.



▷ Consider the following dataset

▷ Example to ﬁnd conﬁdence for itemset {milk,bread}

▷ Conﬁdence (milk->bread) = No. of transactions containing milk and bread/Total no of Transaction containing milk)

▷ Out of 4 transactions containing milk 2 transactions have both milk and bread.

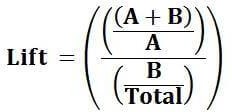
▷ So, Conﬁdence (milk->bread) = (2/4) = 0.5

▷ means that if there are 100 transactions containing milk then there will be 50 that will also contain bread.

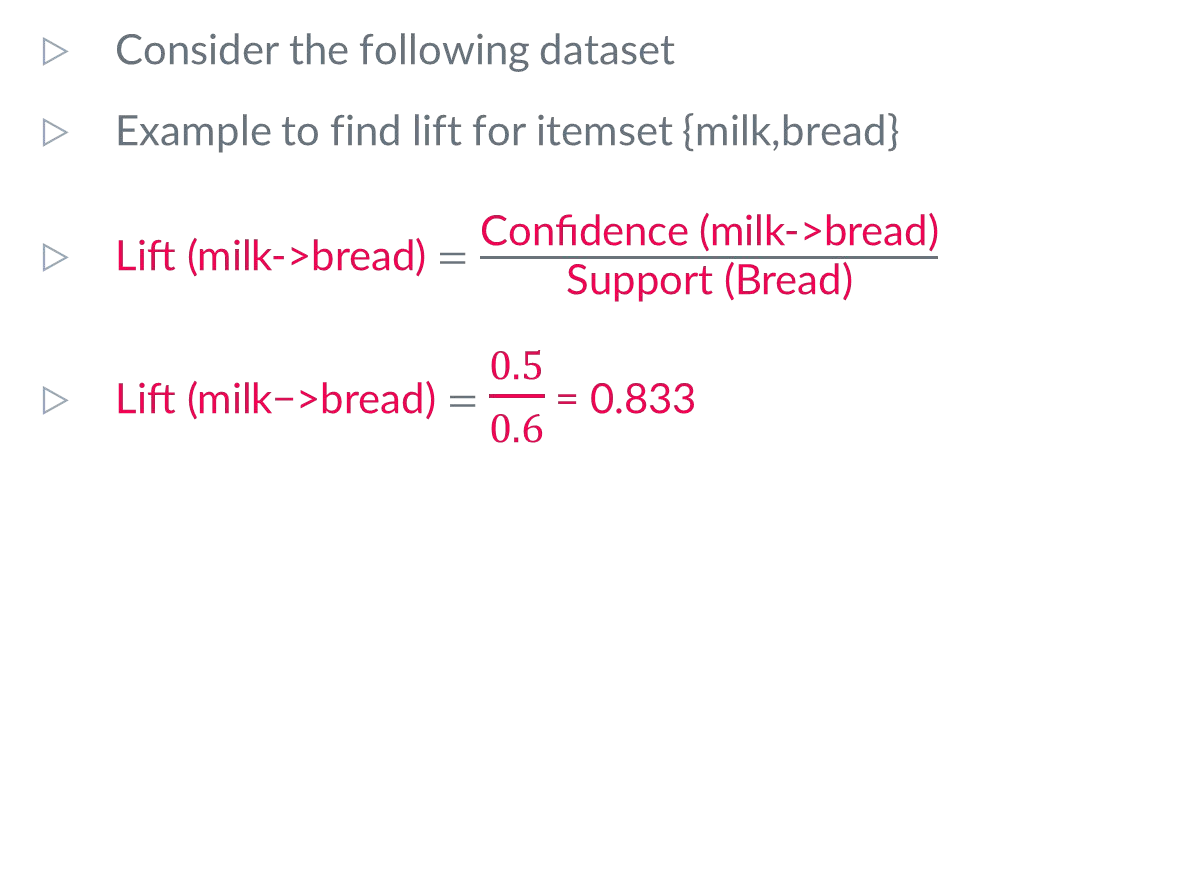
▷ conﬁdence of a rule x ⇒ y is deﬁned as: Support (x ⇒ y) / support (x).

▷ Lift:

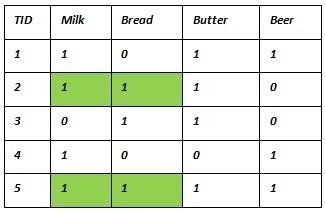
* This third measure, lift or lift ratio is the ratio of conﬁdence to expected conﬁdence.
* We can say that this rule shows us how much better a rule is at predicting the result than just assuming it.
* Greater lift value tells how strong the association is.
* It shows us the rate of conﬁdence that B will be purchased given that A was purchased.



* In other way Lift = Conﬁdence(A=>B) / Support(B)



▷



### Implementation in R

▷ **Step 1 – Load Packages – arules, arulesViz**

library(arules) library(arulesViz)

▷ **Step 2 – Load the data set – Groceries**

* The dataset used here is from the arules R package
* The Groceries data set contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The data set contains 9835 transactions and the items are aggregated to 169 categories.

data(Groceries)

▷ **Step 3 - Explore the data before making any rules**

itemFrequencyPlot(Groceries,topN=20,type="absolute")

### ▷ itemFrequencyPlot ()

* Syntax

itemFrequencyPlot(x, type = c("relative", "absolute"), weighted = FALSE, support = NULL, topN = NULL, population = NULL, popCol = "black", popLwd = 1, lift = FALSE, horiz = FALSE, names = TRUE, cex.names = graphics::par("cex.axis"), xlab = NULL, ylab = NULL, mai = NULL,...)

Arguments

* + x - the object to be plotted.
  + ... - further arguments are passed on (see graphics::barplot() from possible arguments).
  + type - a character string indicating whether item frequencies should be displayed relative of absolute.
  + weighted - should support be weighted by transactions weights stored as column "weight" in transactionInfo?
  + support - a numeric value. Only display items which have a support of at least support. If no population is given, support is calculated from x otherwise from the population. Support is interpreted relative or absolute according to the setting of type.
  + topN - a integer value. Only plot the topN items with the highest item frequency or lift (if lift = TRUE). The items are plotted ordered by descending support.
  + population - object of same class as x; if x is a segment of a population, the population mean frequency for each item can be shown as a line in the plot.
  + popCol- plotting color for population.
  + popLwd - line width for population.
  + lift - a logical indicating whether to plot the lift ratio between instead of frequencies. The lift ratio is gives how many times an item is more frequent in x than in population.
  + horiz - a logical. If horiz = FALSE (default), the bars are drawn vertically. If TRUE, the bars are drawn horizontally.
  + names- a logical indicating if the names (bar labels) should be displayed?
  + cex.names- a numeric value for the expansion factor for axis names (bar labels).
  + xlab - a character string with the label for the x axis (use an empty string to force no label).
  + ylab - a character string with the label for the y axis (see xlab).
  + mai - a numerical vector giving the plots margin sizes in inches (see ‘? par’).

▷ **Step 4 – Performing Apriori Algorithm And Generating Association Rules**

* Apply Aprori algorithm with the minimum amount of support is 0.001 and conﬁdence of 0.8 using apriori() method
* Syntax of apriori() method

apriori(data, parameter = NULL, appearance = NULL, control = NULL, ...)

* Arguments
  + data - object of class transactions. Any data structure which can be coerced into transactions (e.g., a binary matrix, a data.frame or a tibble) can also be speciﬁed and will be internally coerced to transactions.
  + parameter - object of class APparameter or named list. The default behavior is to mine rules with minimum support of 0.1, minimum conﬁdence of 0.8, maximum of 10 items (maxlen), and a maximal time for subset checking of 5 seconds (maxtime).
  + appearance -object of class APappearance or named list. With this argument item appearance can be restricted (implements rule templates). By default, all items can appear unrestricted.
  + control - object of class APcontrol or named list. Controls the algorithmic performance of the mining algorithm (item sorting, report progress (verbose), etc.)
* Get the rules

rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

#### ▷ Step 5 - Show the top 5 rules, but only 2 digits

options(digits=2) inspect(rules[1:5])

#### ▷ Step 6 - Sorting the rules

* The ﬁrst problem is that the rules are not sorted.
* We often want ﬁrst the most important rules. We want the most likely rules.
* For this we use sort() method

sort(x, decreasing = TRUE, na.last = NA, by = "support", order = FALSE, ...)

* Provides the method sort to sort elements in class associations (e.g., itemsets or rules) according to the value of measures stored in the association's slot quality (e.g., support).
* Arguments
  + x - an object to be sorted.
  + decreasing - a logical. Should the sort be increasing or decreasing? (default is decreasing)
  + na.last - na.last is not supported for associations. NAs are always put last.
  + by - a character string specifying the quality measure stored in x to be used to sort x. If a vector of character strings is speciﬁed then the additional strings are used to sort x in case of ties.
  + order - should a order vector (a permutation like order()) be returned instead of the sorted associations?
* By executing the following code, we can easily sort by trust. rules<-sort(rules, by="conﬁdence", decreasing=TRUE)
* Our top ﬁve outputs are now sorted by trust and thus the most relevant rules appear.

▷ **Step 7 – Specify more precise rules**

rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8,maxlen=3))

▷ **Step 8 - Targeting Items**

* Now that we know how to generate rules, limit the output, we want to go for rules. We could be interested in two types of targets, illustrated in an example of “all milk”:
* What are customers who can buy whole milk before buying?
  + 1) If customers buy whole milk, what are they likely to buy?
  + 2) In essence, we want the left side and the right side of the hand.

▷ We modify our apriori() function as follows on the ﬁrst question:

### rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.08), appearance = list(default="lhs",rhs="whole milk"),

control = list(verbose=F))

### rules<-sort(rules, decreasing=TRUE,by="conﬁdence") inspect(rules[1:5])

▷ We can also make the left side “full milk” and ﬁnd its background. Please note that:

* We have set the trust to 0.15 because we do not have 0.8 rules.
* To avoid leaky left-hand items, we set a minimum length of 2

rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2), appearance = list(default="rhs",lhs="whole milk"),

control = list(verbose=F))

rules<-sort(rules, decreasing=TRUE,by="conﬁdence") inspect(rules[1:5])

▷ Step 9 - **Visualization**

* Visualization is the ﬁnal step. Let’s say that you wanted a graph to map the rules.
* This can be done with another “arulesViz” library. plot(rules, method="graph")

# K-Means algorithm in R

▷ Clustering is an unsupervised learning technique.

▷ Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to ﬁnd patterns in the data.

▷ It is the task of grouping together a set of objects in a way that objects in the same cluster are more similar to each other than to objects in other clusters.

▷ Similarity is an amount that reﬂects the strength of relationship between two data objects.

▷ Clustering is mainly used for exploratory data mining.

▷ It is used in many ﬁelds such as machine learning, pattern recognition, image analysis, information retrieval, bio-informatics, data compression, and computer graphics.

▷ In k means clustering, we have to the specify the number of clusters we want the data to be grouped into.

▷ The algorithm randomly assigns each observation to a cluster, and ﬁnds the centroid of each cluster.

▷ Then, the algorithm iterates through two steps:

* Reassign data points to the cluster whose centroid is closest.
* Calculate new centroid of each cluster.

▷ These two steps are repeated till the within cluster variation cannot be reduced any further.

▷ The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

**Implementation of K-Means in R**

▷ **Step 1 : Load required packages – tidyverse, datasets, ggplot2**

* We are going to use iris dataset for this implementation.
* The “datasets” library in R already contains it.
* Just by loading the library, a data frame named iris will be made available and can be used straight away.
* “ggplot2” is included in the tidyverse package. Alternatively, you can only load ggplot2 individually.
* The ggplot2 package can be used to create various types of charts and graphs from a data frame.

▷ **Step 2 : Load dataset “iris**

* The iris dataset is already present in the datasets library.
* Just by loading the library, a data frame named iris will be made available and can be used straight away:

iris

▷ **Step 3: Get more information about iris dataset**

* If you want to take a glimpse at the ﬁrst 4 lines of rows. head(iris, 4)
* Optionally you may want to visualize the last rows of your dataset

tail(iris)

* The dimensions of the dataframe dim(iris)
* The names of the columns names(iris)
* The attributes of the dataframe attributes(iris)
* Finally, if you want the descriptive statistics summary summary(iris)
* Indexing the ﬁrst 5 rows iris[1:5,]
* Indexing the ﬁrst 4 columns iris[,1:4]
* If you want to explore the ﬁrst 10 rows of a particular column, in this case, Sepal length iris[1:10, "Sepal.Length"]
* If you want to check if a value is missing, you must use the function is.na: is.na(iris)
* To get the total number of NAs present in the dataset sum(is.na(iris))

▷ **Step 4: Plot data using ggplot() function of ggplot2 library**

* We use the function ggplot() to generate the plots.
* To build a ggplot, we will use the following basic template that can be used for different types of plots:

ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) + <GEOM\_FUNCTION>()

* + use the ggplot() function and bind the plot to a speciﬁc data frame using the **data**

argument

* + deﬁne an aesthetic mapping (using the aesthetic (aes) function), by selecting the variables to be plotted and specifying how to present them in the graph, e.g., as x/y positions or characteristics such as size, shape, color, etc.
  + add ‘geoms’ – graphical representations of the data in the plot (points, lines, bars). ggplot2 offers many different geoms; we will use some common ones today, including:
    - geom\_point() for scatter plots, dot plots, etc.
    - geom\_boxplot() for, well, boxplots!
    - geom\_line() for trend lines, time series, etc.
  + To add a geom to the plot use + operator.
* ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom\_point()

▷ **Step 5: Clustering**

* Since the initial cluster assignments are random, let us set the seed to ensure reproducibility.
* Seeds allow you to create a starting point for randomly generated numbers, so that each time your code is run, the same answer is generated.

set.seed(20)

* You can use the kmeans() function in R which uses the following syntax: kmeans(data, centers, nstart)
* where:
* data: Name of the dataset.
* centers: The number of clusters, denoted k.
* nstart: The number of initial conﬁgurations. Because it’s possible that different initial starting clusters can lead to different results, it’s recommended to use several different initial

conﬁgurations. The k-means algorithm will ﬁnd the initial conﬁgurations that lead to the smallest within-cluster variation.

* To perform k-means clustering in R we can use the built-in kmeans() function
* k value will be set as 3. K=3 is the best value for k in this case (Note: There are 3 classes in the original iris data)
* Also, there is a nstart option that attempts multiple initial conﬁgurations and reports on the best one within the kmeans function.

irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)

* Since we know that there are 3 species involved, we ask the algorithm to group the data into 3 clusters, and since the starting assignments are random, we specify nstart = 20
* This means that R will try 20 different random starting assignments and then select the one with the lowest within cluster variation.
* Let us compare the clusters with the species. table(irisCluster$cluster, iris$Species)

▷ **Step 6: Plot the data to see the clusters:**

irisCluster$cluster <- as.factor(irisCluster$cluster)

ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom\_point()

**Any Doubts?**