Savitribai Phule Pune University

सावित्रीबाई फुले पुणे विद्यापीठ

T.Y.B.C.A.

(Science)

DSE II

BCA 357- Laboratory (Data Mining)

Workbook

Semester V

(From Academic Year 2021)

Name_____Roll No._____

College_____Division _____

Academic Year_____

From the Chairman's Desk

It gives me a great pleasure to present this workbook prepared by the Board of studies in Computer Applications.

The workbook has been prepared with the objectives of bringing uniformity in implementation of lab assignments across all affiliated colleges, act as a ready reference for both fast and slow learners and facilitate continuous assessment using clearly defined rubrics.

The workbook provides, for each of the assignments, the aims, pre-requisites, related theoretical concepts with suitable examples wherever necessary, guidelines for the faculty/lab administrator, instructions for the students to perform assignments and a set of exercises divided into three sets.

I am thankful to the Chief Editor and the entire team of editors appointed. I am also thankful to the members of BOS. I thank everyone who have contributed directly or indirectly for the preparation of the workbook.

Constructive criticism is welcome and to be communicated to the Chief Editor. Affiliated colleges are requested to collect feedbacks from the students for the further improvements.

I am thankful to Hon. Vice Chancellor of Savitribai Phule Pune University Prof. Dr. Nitin Karmalkar and the Dean of Faculty of Science and Technology Prof. Dr. M G Chaskar for their support and guidance.

Prof. Dr. S. S. Sane

Chairman, BOS in Computer Applications

SPPU, Pune

Chairperson and Editor:

Dr. Dipali Meher — PES Modern College of Arts, Science and Commerce, Ganeshkhind Pune 16

Prepared and Compiled by:

Sr. No.	Assignment Name	Teacher Name	College Name
1.	Assignment 1 : R Programming	Prof. Kavita Khoje	Haribhai V. Desai College, Pune
2.	Assignment 2: Data Pre-processing	Dr. Sonali Nemade	Dr. D. Y. Patil Arts, Commerce and Science College, Pimpri
3.	Assignment 3: Classification	Prof. Sanjay Wani & Dr. Dipali Meher	Women's College of Home Science and BCA, Loni PES Modern College of Arts, Science and Commerce, Ganeshkhind Pune 16
4.	Assignment 4: Association Rules	Prof. Sachin Mhaske	Rajarshi Shahu Mahavidyalaya, Deolali-Pravara.
5.	Assignment 5: Regression Analysis and Outlier detection	Dr. Dipali Meher	PES Modern College of Arts, Science and Commerce, Ganeshkhind Pune 16
6.	Assignment 6: Clustering	Prof. Sachin Mhaske & Dr. Sonali Nemade	Rajarshi Shahu Mahavidyalaya, Deolali-Pravara. Dr. D. Y. Patil Arts, Commerce and Science College, Pimpri

Reviewed By

1. Dr. Madhukar Shelar — KTHM College, Nashik 02
2. Dr. Pallawi Bulakh — PES Modern College, of Arts, Science and Commerce, Ganeshkhind Pune 16
3. Dr. Sayyad Razak — Ahmednagar College, Ahmednagar

ABOUT THE WORK BOOK

This lab-book is intended to be used by T.Y.BCA(Science) students for **BCA 357-Laboratory (Data Mining)**, Laboratory Assignments in Semester V. This workbook is designed by considering all the practical concepts topics mentioned in syllabus

The objectives of this workbook are

1) Defining the scope of the course.
2) To bring the uniformity in the practical conduction and implementation in all colleges affiliated to SPPU.
3) To have continuous assessment of the course and students.
4) Providing ready reference for the students during practical implementation.
5) Provide more options to students so that they can have good practice before facing the examination.
6) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

How to use this book?

The workbook is divided into 6 assignments. Each assignment has three SETs .It is mandatory for students to complete SETA ,SETB and SET C in given slot.

Instructions to the students

Please read the following instructions carefully and follow them.

Students are expected to carry this book every time when they come to the lab for computer science practical.

1. Students should prepare themselves before hand for the Assignment by reading these materials.

2. Instructor will specify which problems to solve in the lab during the allotted slot and student should complete the verified by the instructor. How every student should spend additional hours in Lab and at home to cover as many problems as possible given in this workbook.

3. Students will be assessed for each exercise on as scale from 0 to5 as indicated below.

i)	Not done	0
ii)	Incomplete	1
iii)	Late Complete	2
iv)	Needs improvement	3
v)	Complete	4
vi)	Well Done	5

Guide lines for Instructors

1. Explain the assignment and related concepts in around ten minutes so using whiteboard if required or by demonstrating the software.

2. You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.

3. The value should also be entered on assignment completion page of the respective Lab Course.

Guidelines for Lab Administrator

You have to ensure appropriate hardware and software is available to each student in the Lab.

The operating system and software requirement so server side and also client side areas given below:

1) Server and Client Side-(Operating System)Linux (Ubuntu/RedHat/Fedora/ Centos)

2) Jupyter/Spyder notebook or Anaconda Navigator or any IDLE of python on any platform.

3) R Software

INDEX

Sr. No.	Assignment Name	Page Number
1	R Programming Basics, Programs using List, Matrix, String and Factors, Program using data frame and visualization	14
2	Data Preprocessing	42
3	Classification	52
4	Association Rules	76
5	Regression Analysis and Outlier detection	85
6	Clustering	108

Assignment Completion Sheet

Subject: DSE II BCA 357- Laboratory (Data Mining)			
Sr. No.	**Assignment Name**	**Marks (Out of 5)**	**Teacher's Sign**
1	R Programming,		
2	Data Preprocessing		
3	Classification		
4	Association Rules		
5	Regression Analysis and Outlier detection		
6	Clustering		
	Total out of 30		
	Total out of 15		

CERTIFICATE

This is to certify that

*Mr./Ms.*_____

*Has successfully completed BCA-357DSEII Laboratory (Data Mining) course in the year*_____ *and his/her seat number is*_____ .

*He/She has scored marks*_____*out of 15.*

Instructor *H.O.D./Coordinator*

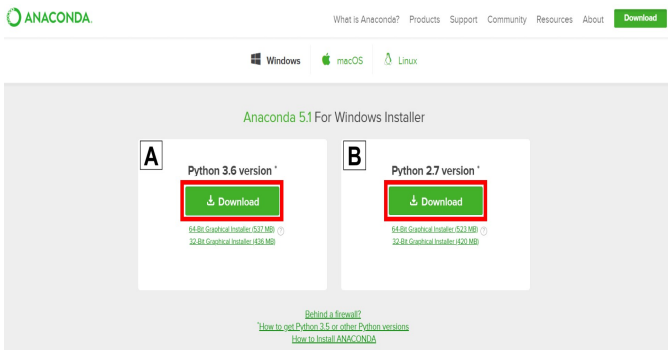
Internal Examiner *External Examiner*

Required Software's : R, Anaconda

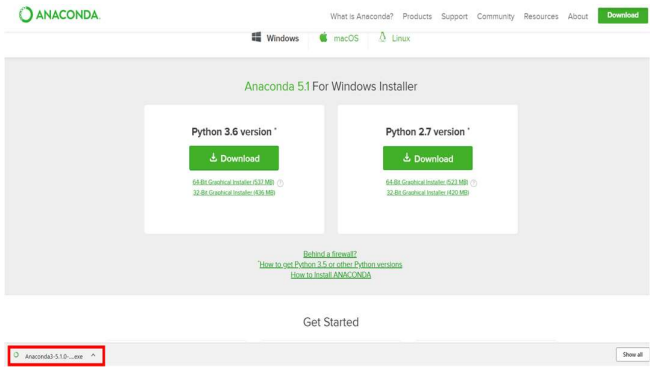
For this Data Pre-processing script, We are going to use Anaconda Navigator and specifically Jypyter NoteBook or Spider(IDE) or Pycharm(IDE).

Installation Steps of Anaconda Navigator:

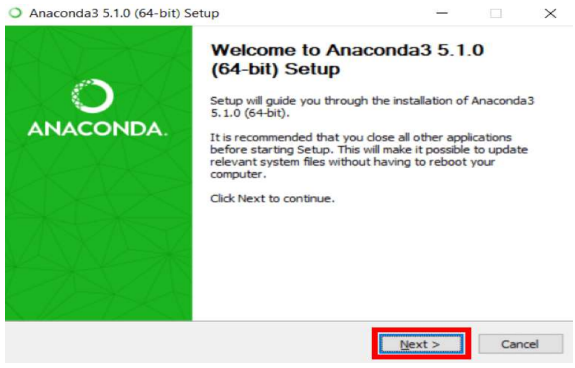
1. Go to the Anaconda Website and choose a Python 3.x graphical installer (A) or a Python 2.x graphical installer (B). If you aren't sure which Python version you want to install, choose Python 3. Do not choose both.



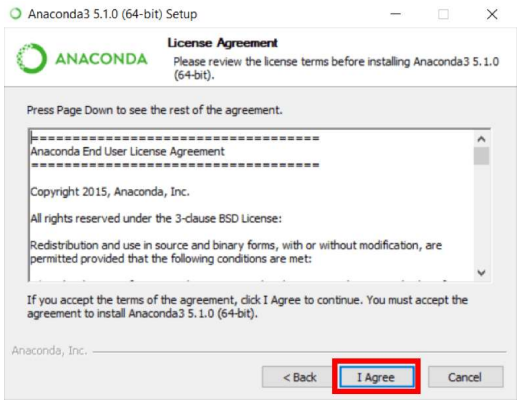
2. Locate your download and double click it.



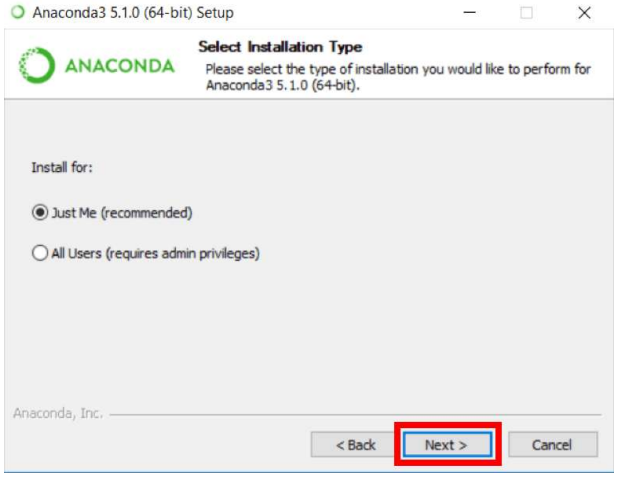
When the screen below appears, click on Next.



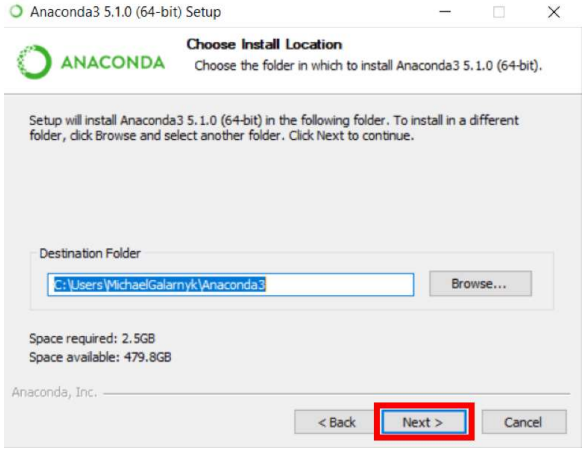
3. Read the license agreement and click on I Agree.



4. Click on Next.



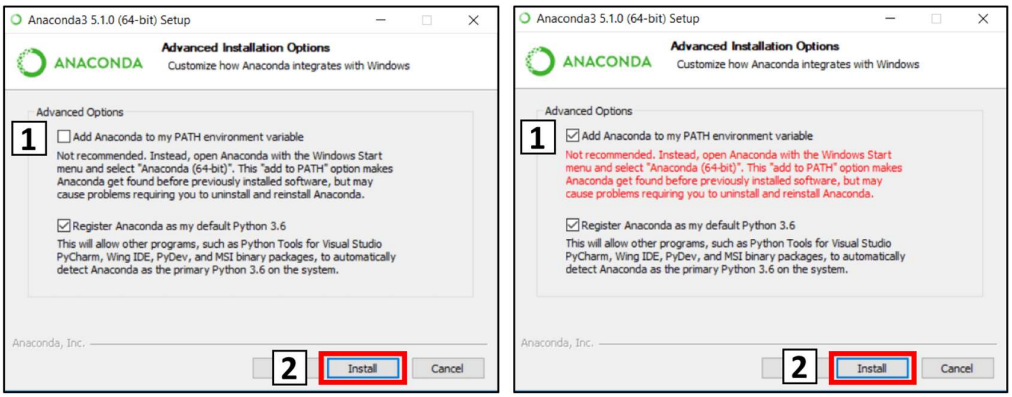
5. Note your installation location and then click Next

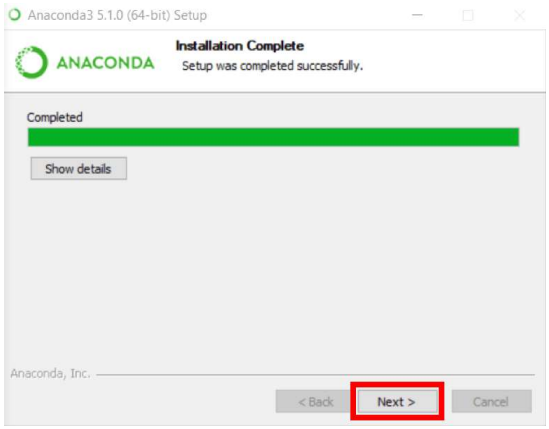


6. This is an important part of the installation process. The recommended approach is to not check the box to add Anaconda to your path. This means you will have to use Anaconda Navigator or the Anaconda Command Prompt (located in the Start Menu under "Anaconda") when you wish to use Anaconda (you can always add Anaconda to your PATH later if you don't check the box). If you want to be able to use Anaconda in your

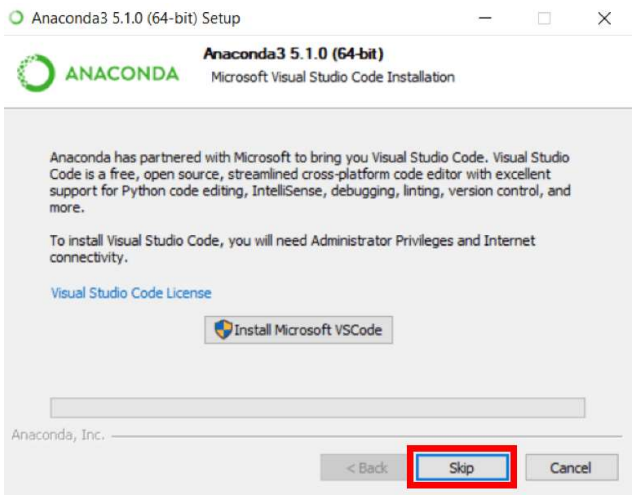
command prompt (or git bash, cmder, powershell etc), please use the alternative approach and check the box.

Recommended Approach **Alternative Approach**

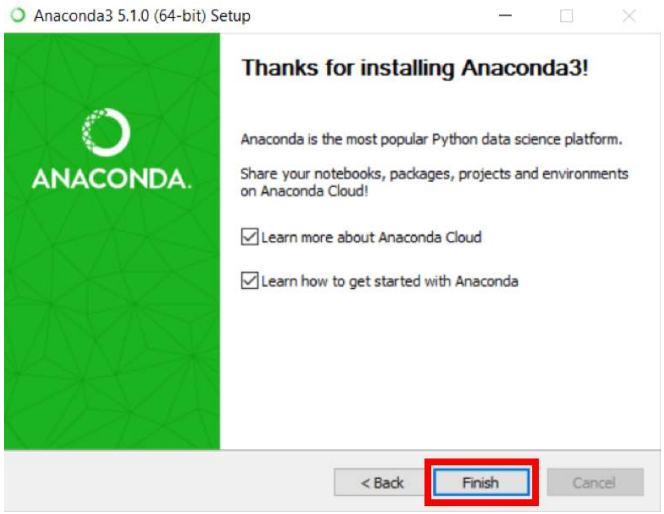




7. You can install Microsoft VSCode if you wish, but it is optional.



8. Click on Finish.



Machine Learning libraries:

1. **Numpy :** Numpy is a general purpose array processing package. It provides a high performance multidimension array object and tools for working with these array. Numpy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structure to python that guarantee efficient calculations with array and matrices and it supplies an extensive library of high level mathematical functions that operate on these arrays and metrices.

Numpy used much less memory to store data and it provides a mechanism of specifying the data types.

Numpy operations:

1. Vector-vector multiplication.
2. Matrix-Matrix and Matrix-vector multiplication.
3. Element wise and array wise comparison.
4. Applying functions element-wise to a vector or matrix.
5. Linear Algebra operations.

2. **Pandas :** Pandas is an open source library that allows to you perform data manipulation and analysis in python. Pandas provide an easy way to create, manipulate and wrangle the data. It is built on top of Numpy, means it needs Numpy to operate.

3. Scikit-learn: Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical Modeling including classification, regression, clustering and dimensionality reduction.

Please note that sklearn is used to build machine learning models. It should not be used for reading the data, manipulating and summarizing it. There are better libraries for that (e.g. NumPy, Pandas etc.)

4. **Sklearn:** Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.

It features various classification, regression and clustering algorithms and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

5. **Seaborn:** Seaborn is a data visualization library built on top of matplotlib and integrated with pandas data structure in python. Visualization is the central part of seaborn which helps in exploration and understanding.

6. **Matplotlib:** Matplotlib is a Python 2D plotting library that is used to plot any type of charts in Python. It can deliver publication-quality figures in numerous hard copy formats and interactive environments across platforms (IPython shells, Jupyter notebook, web application servers, etc.).

Assignment 1: R Programming Basics Program using List, Matrix, String and Factors Program using data frame and visualization

Objectives:

Students will be able to

- Understand data structures available in R programming.
- Understand how data structures are used for writing programmes in R
- Learn working of R programming for data visualization.
- How to implement data structures in R, step by step
- Understand use of inbuilt data sets and how to perform operations on it.

Prerequisites:

- ✓ Knowledge of C, CPP Programming languages
- ✓ https://www.statmethods.net/r-tutorial/index.html
- ✓ https://www.listendata.com/2014/06/getting-started-with-r.html

Introduction :

R programming is one of the most popular languages used in data science, statistical computations or scientific research. R programming is very efficient and user-friendly widely used in machine learning. It provides flexibility in doing big statistical operations with a few lines of code. R programming provides following data structures

1. Vectors
2. Lists
3. Arrays
4. Matrix
5. Data Frames
6. Factors

What is R Vector?

A vector is a sequence of elements that share the same data type. These elements are known as components of a vector.

R vector is the basic data structure, which plays an essential role in R programming. Vectors in R are same as the arrays in C language which hold multiple data values of the same type. One major key point is that in R the indexing of the vector will start from '1' and not from '0'. We can create numeric vectors and character vectors as well.

R vector comes in two parts: **Atomic vectors** and **Lists.**

- These data structures differ in the type of their elements: All elements of an atomic vector must be of the same type, whereas the elements of a list can have different types.
- They have three common properties, i.e., **function type, function length**, and **attribute function**.

Types of Atomic Vector

1. Numeric Data Type - Decimal values are referred to as numeric data types in R.

2. Integer Data Type - A numeric value with no fraction called integer data is represented by “Int”.

```
# R program to create numeric Vectors
# Creation of vectors using c() function.
v1 <- c(1,2,3,4)
typeof(v1)          # display type of vector
```

Output

```
[1] "double"
```

```
v2<- c(1.1, 2.1, 3.1, 4.1)
typeof(v2)
```

Output

```
[1] "double"
```

```
# by using 'L' we can specify that we want integer values.
        v3 <- c(1L, 2L, 3L, 4L)
        typeof(v3)
```

Output

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

3. Character Data Type - The character is held as the one-byte integer in memory

```
# R program to create Character Vectors

# by default numeric values are converted into characters
        v4 <- c('Lab', '357', 'data mining', '35')
        typeof(v4) # displaying type of vector
```

Output

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

4. Logical Data Type - A logical data type returns either of the two values – TRUE or FALSE based on which condition is satisfied and NA for NULL values.
   For Example:

```
# R program to create Logical Vectors

#Creating logical vector using c() function
    V5 <- c(TRUE, FALSE, TRUE, NA)
    typeof(V5)
```

Output:

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

How to create a vector in R?

Different ways to create vector in R

1. **Using c() function** - In R, we use c() function to create a vector. This function returns a one-dimensional array or simply vector. The c() function is a generic function which combines its argument. All arguments are restricted with a common data type which is the type of the returned value. There are various other ways to create a vector in R, which are as follows:

```
# we can use the c function to combine the values as a vector.
# By default the type will be double
V1 <- c(1, 2, 3, 4)
cat('using c function', V1, '\n')

Output:
using c function 1 2 3 4
```

2. **Using the ':' Operator:** We can create a vector with the help of the colon operator to create vector of continuous values

```
V2 <- 1:5
cat('using colon', V2)
Output:
using colon 1 2 3 4 5
```

3. **Using the seq() function -** In R, we can create a vector with the help of the seq() function. A sequence function creates a sequence of elements as a vector. There are also two ways in this. The first way is to set the step size and the second method is by setting the length of the vector.

```
Seq_V <- seq(1, 4, length.out = 6)
cat('using seq() function', Seq_V, '\n')
Output:
        using seq() function 1 1.6 2.2 2.8 3.4 4
Seq_V2 <- seq(1, 4, by = 0.5)
cat('using seq() function by', Seq_V2, '\n')
Output:
        using seq() function by 1 1.5 2.0 2.5 3.0 4
```

How to Access Elements of R Vectors?

With the help of vector indexing, we can access the elements of vectors. Indexing denotes the position where the values in a vector are stored. This indexing can be performed with the help of integer, character or logic.

```
# R program to access elements of a Vector
# accessing elements with an index number.
X <- c(1, 5, 10, 1, 12)
cat('Using Subscript operator', X[2], '\n')
Output:
Using Subscript operator 5
# by passing a range of values inside the vector index.
Y <- c(14, 18, 12, 11, 17)
cat('Using combine() function', Y[c(4, 1)], '\n')
Output:
Using combine() function 11 14
# using logical expressions
Z <- c(5, 2, 1, 4, 4, 3)
cat('Using Logical indexing', Z[Z>4])
Output:
Using Logical indexing 5
```

```
#indexing using character vector
char_vec<-c("shubham"=101,"ram"=102,"varsha"=103)
cat('Using character vector', char_vec[“ram”])
Output:
Using character vector 102
#indexing using logical vector it returns the values of those positions whose
#corresponding position has a logical vector TRUE.
a<-c(1,2,3,4,5,6)
a[c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE)]
Output:
[1] 1 3 4 6
```

Vector Operations

In R, there are various operation which is performed on the vector. We can add, subtract, multiply or divide two or more vectors from each other. In data science, R plays an important role, and operations are required for data manipulation. There are the following types of operation which are performed on the vector.

1. **Combining vector in R :** The c() function is not only used to create a vector, but also it is also used to combine two vectors. By combining one or more vectors, it forms a new vector which contains all the elements of each vector. Let see an example to see how c() function combines the vectors.

```
num = c(1, 2, 3, 4)
str = c("one", "two", "three", "Four")
c(num,str)
Output:

[1] "1"     "2"     "3"     "4"     "one"   "two"   "three" "Four"
```

2. **Arithmetic Operation on Vectors in R:** We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors. We can add, subtract, multiply, or divide two vectors.

```
a<-c(1,3,5,7)
b<-c(2,4,6,8)
a+b  #addition
Output
            [1]  3  7 11 15
a-b  #subtraction
Output
            [1] -1 -1 -1 -1
a/b  #division #a % / % b is used to give integer division
Output
            [1] 0.5000000 0.7500000 0.8333333 0.8750000
a%%b  #Reminder operation
Output
[1] 1 3 5 7
```

3. **Logical Index Vector:** With the help of the logical index vector in R, we can form a new vector from a given vector. This vector has the same length as the original vector. The

vector members are TRUE only when the corresponding members of the original vector are included in the slice; otherwise, it will be false.

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
L<-c(TRUE,FALSE,TRUE,TRUE,FALSE,FALSE)
names[b]

Output
            [1] "Ram"   "Nisha" "Siya"
```

4. **Numeric Index**: In R, we specify the index between square braces [ ] for indexing a numerical value. If our index is negative, it will return us all the values except for the index which we have specified. For example, specifying [-3] will prompt R to convert -3 into its absolute value and then search for the value which occupies that index.

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[2]
Output
          [1] "Aryan"

names[-4]
Output
             [1] "Ram"    "Aryan"  "Nisha"  "Radha"  "Gunjan"

names[15]
Output
            NA
```

5. **Duplicate Index**: The index vector allows duplicate values. Hence, the following retrieves a member twice in one operation

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[c(2,4,4,3)]
Output
          [1] "Aryan" "Siya"  "Siya"  "Nisha"
```

6. **Range Index:** Range index is used to slice our vector to form a new vector. For slicing, we used colon(:) operator. Range indexes are very helpful for the situation involving a large operator.

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[1:3]
Output
[1] "Ram"   "Aryan" "Nisha"
```

7. **Out of Order Index:** In R, the index vector can be out-of-order. Below is an example in which a vector slice with the order of first and second values reversed

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[c(2,1,3,4,5,6)]
Output
            [1] "Aryan"  "Ram"    "Nisha"  "Siya"   "Radha"  "Gunjan"
```

8. **Sorting element of the vector**: **sort()** function is used with the help of which we can sort the values in ascending or descending order.

```
# R program to sort elements of a Vector
 # Creation of Vector
num <- c(8, 2, 7, 1, 11, 2)

# Sort in ascending order
Asc_num <- sort(num)
cat('ascending order', Asc_num, '\n')
```

Output

```
ascending order 1  2  2  7  8 11
```

```
# sort in descending order by setting decreasing as TRUE
 Desc_num<- sort(num, decreasing = TRUE)
cat('descending order', Desc_num)
```

Output

```
descending order 11  8  7  2  2  1
```

9. **Names vectors members**

```
 # We first create our vector of characters as
lib=c("TensorFlow","PyTorch")
lib
```

Output

```
[1] "TensorFlow" "PyTorch"
```

```
# Once our vector of characters is created, we name the first vector member as
"Start" and the #second member as "End" as:
names(lib)=c("Start","End")
lib
```

Output

```
      Start         End
"TensorFlow"    "PyTorch"
```

```
#We retrieve the first member by its name as follows:
lib["Start"]
```

Output

```
 Start
"TensorFlow"
```

Applications of vectors

1. In machine learning for principal component analysis vectors are used. They are extended to eigenvalues and eigenvector and then used for performing decomposition in vector spaces.

2. The inputs which are provided to the deep learning model are in the form of vectors. These vectors consist of standardized data which is supplied to the input layer of the neural network.
3. In the development of support vector machine algorithms, vectors are used.
4. Vector operations are utilized in neural networks for various operations like image recognition and text processing.

What is R List?

In R, lists are the second type of vector. Lists are the objects of R which contain elements of different types such as number, vectors, string and another list inside it. It can also contain a function or a matrix as its elements. A list is a data structure which has components of mixed data types. We can say, a list is a generic vector which contains other objects.

How to create List?

The process of creating a list is the same as a vector. In R, the vector is created with the help of c() function. Like c() function, there is another function, i.e., list() which is used to create a list in R. A list avoid the drawback of the vector which is data type. We can add the elements in the list of different data types.

```
# Create a list containing strings, numbers, vectors and a logical values.
list_data <- list("Ram", "Sham", c(1,2,3), TRUE, 1.03)
print(list_data)

Output
$Rscript main.r
[[1]]
[1] "Ram"
[[2]]
[1] "Sham"
[[3]]
[1] 1 2 3
[[4]]
[1] TRUE
[[5]]
[1] 1.03
```

Giving a name to list elements

R provides a easy way for accessing elements, i.e., by giving the name to each element of a list. By assigning names to the elements, we can access the element easily. There are only three steps to print the list data corresponding to the name:

1. Creating a list.
2. Assign a name to the list elements with the help of names() function.
3. Print the list data.

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),
   list("BCA","MCA","BSc"))
```

```
# Giving names to the elements in the list.
names(list_data) <- c("Students", "Marks", "Course")
  # Show the list.
print(list_data)
```

Output

```
$Students
[1] "Ram"  "Sham" "Raj"
$Marks
     [,1] [,2] [,3]
[1,]   40   60   90
[2,]   80   70   80
$Course
$Course[[1]]
[1] "BCA"
$Course[[2]]
[1] "MCA"
$Course[[3]]
[1] "BSc"
```

How to access R List elements?

R provides two ways to access the elements of a list.

1. First one is the indexing method performed in the same way as a vector.
2. In the second one, we can access the elements of a list with the help of names. It will be possible only with the named list.; we cannot access the elements of a list using names if the list is normal.

Example 1. Accessing elements of List using index

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),    list("BCA","
MCA","BSc"))
# Accessing the first element of the list.
print(list_data[1])
 Output
[[1]]
[1] "Ram"  "Sham" "Raj"
# Accessing the third element. The third element is also a list, so all its elements will be printed.
print(list_dat
Output

[[1]]
[[1]][[1]]
[1] "BCA"
[[1]][[2]]
[1] "MCA"
[[1]][[3]]
[1] "BSc"
```

Example 2. Accessing elements of List using name

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),    list("BCA","MCA","BSc"))
# Giving names to the elements in the list.
names(list_data) <- c("Student", "Marks", "Course")
# Accessing the first element of the list.
print(list_data["Student"])
Output
          $Student
          [1] "Ram"  "Sham" "Raj"

print(list_data$Marks)
Output

     [,1] [,2] [,3]
[1,]   40   60   90
[2,]   80   70   80

print(list_data)
Output

  $Student
[1] "Ram"  "Sham" "Raj"
$Marks
     [,1] [,2] [,3]
[1,]   40   60   90
[2,]   80   70   80
$Course
$Course[[1]]
[1] "BCA"
$Course[[2]]
[1] "MCA"
$Course[[3]]
[1] "BSc"
```

R allows us to add, delete, or update elements in the list. We can update an element of a list from anywhere, but elements can add or delete only at the end of the list. To remove an element from a specified index, we will assign it a null value. We can update the element of a list by overriding it from the new value.

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),    list("BCA","MCA"
,"BSc"))
# Giving names to the elements in the list.
names(list_data) <- c("Student", "Marks", "Course")
# Adding element at the end of the list(you can use append function also).
list_data[4] <- "Pune"  #list_data <- append(list_data,"Pune")
print(list_data[4])
Output
[[1]]
[1] "Pune"

# Removing the last element.
list_data[4] <- NULL
  # Printing the 4th Element.
print(list_data[4])
Output
$<NA>
NULL
  # Updating the 3rd Element.
list_data[3] <- "BCA Science"
print(list_data[3])
Output
$Course
[1] "BCA Science"
```

Converting list to vector : There is a drawback with the list, i.e., we cannot perform all the arithmetic operations on list elements. To remove this, drawback R provides unlist() function. This function converts the list into vectors. In some cases, it is required to convert a list into a vector so that we can use the elements of the vector for further manipulation. The unlist() function takes the list as a parameter and change into a vector.

```
# Creating lists.
list1 <- list(1:5)
print(list1)
Output
        [[1]]
        [1] 1 2 3 4 5
list2 <-list(11:15)
print(list2)
Output
        [[1]]
        [1] 11 12 13 14 15
# Converting the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
print(v1)
Output
        [1] 1 2 3 4 5
 print(v2)
Output
        [1] 11 12 13 14 15
#adding the vectors
result <- v1+v2
print(result)
Output
        [1] 12 14 16 18 20
```

How to merge lists in R programming?

R allows us to merge one or more lists into one list. Merging is done with the help of the list() function also. To merge the lists, we have to pass all the lists into list function as a parameter, and it returns a list which contains all the elements which are present in the lists.

```
# Creating two lists.
Even_list <- list(2,4,6,8,10)
Odd_list <- list(1,3,5,7,9)

# Merging the two lists.
merged.list <- list(Even_list,Odd_list)

# Printing the merged list.
print(merged.list)
```

Output

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

[[1]][[2]]
[1] 4

[[1]][[3]]
[1] 6

[[1]][[4]]
[1] 8

[[1]][[5]]
[1] 10


[[2]]
[[2]][[1]]
[1] 1

[[2]][[2]]
[1] 3

[[2]][[3]]
[1] 5

[[2]][[4]]
[1] 7

[[2]][[5]]
[1] 9
```

What is array in R programming?

In R, arrays are the data objects which allow us to store data in more than two dimensions. In R, an array is created with the help of the **array()** function. This array() function takes a vector as an input and to create an array it uses vectors values in the **dim** parameter.

For example- if we will create an array of dimension (2, 3, 4) then it will create 4 rectangular matrices of 2 row and 3 columns.

```
#Create two vectors of different lengths.
 vector1 <- c(1,2,3)
 vector2 <- c(10,11,12,13,14,15)

 # Take these vectors as input to the array.
 result <- array(c(vector1,vector2),dim = c(3,3,2))
 print(result)
```

Output

```
, , 1

     [,1] [,2] [,3]
[1,]    1   10   13
[2,]    2   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    1   10   13
[2,]    2   11   14
[3,]    3   12   15
```

Naming Columns and Rows

We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
#Create two vectors of different lengths.
 vector1 <- c(1,2,3)
 vector2 <- c(10,11,12,13,14,15)

column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
 # Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames =
list(row.names,column.names,matrix.names))
print(result)
```

Output

```
, , Matrix1
     COL1 COL2 COL3
ROW1    1   10   13
ROW2    2   11   14
ROW3    3   12   15

, , Matrix2

     COL1 COL2 COL3
ROW1    1   10   13
ROW2    2   11   14
ROW3    3   12   15
```

Accessing Array elements

```
#Create two vectors of different lengths.
 vector1 <- c(1,2,3)
 vector2 <- c(10,11,12,13,14,15)
 column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
 # Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames =
list(row.names,column.names,matrix.names))

# Print the third row of the second matrix of the array.
print(result[3,,2])
Output
COL1 COL2 COL3
  3   12   15
 # Print the element in the 1st row and 3rd column of the 1st matrix.
 print(result[1,3,1])
 Output
 [1] 13
 # Print the 2nd Matrix.
 print(result[,,2])
 Output
     COL1 COL2 COL3
 ROW1    1   10   13
 ROW2    2   11   14
 ROW3    3   12   15
```

Matrix in R

In R, a two-dimensional rectangular data set is known as a matrix. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation.

In the R matrix, elements are arranged in a fixed number of rows and columns. The matrix elements are the real numbers. In R, we use matrix function, which can easily reproduce the memory representation of the matrix. In the R matrix, all the elements must share a common basic type.

```
#Syntax to create matrix
#matrix(data, nrow, ncol, byrow, dim_name)
matrix1<-matrix(c(11, 13, 15, 12, 14, 16),nrow =2, ncol =3, byrow = TRUE)
matrix1
Output
    [,1] [,2] [,3]
[1,]   11   13   15
[2,]   12   14   16
# Elements are arranged sequentially by row.
M <- matrix(c(1:12), nrow = 4, byrow = TRUE)
print(M)
```

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

# Elements are arranged sequentially by column.
N <- matrix(c(1:12), nrow = 4, byrow = FALSE)
print(N)
Output
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
P <- matrix(c(1:12), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
Output
     col1 col2 col3
row1    1    2    3
row2    4    5    6
row3    7    8    9
row4   10   11   12
```

Accessing matrix elements

There are three ways to access the elements from the matrix.

1. We can access the element which presents on nth row and mth column.
2. We can access all the elements of the matrix which are present on the nth row.
3. We can also access all the elements of the matrix which are present on the mth column.

```
# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3")
#Creating matrix
R <- matrix(c(1:12), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print(R)
 Output
      col1 col2 col3
row1    1    2    3
row2    4    5    6
row3    7    8    9
row4   10   11   12

#Accessing element present on 3rd row and 2nd column
print(R[3,2])
 Output
            [1] 8
```

```
#Accessing element present in 3rd row
print(R[3,])
Output
            col1 col2 col3
             7    8    9
  #Accessing element present in 2nd column
print(R[,2])
Output
            row1 row2 row3 row4
              2    5    8   11
```

Matrix Computations : Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix. The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

```
> # Create two 2x3 matrices.
> matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
> print(matrix1)
Output
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
 > matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
> print(matrix2)
Output
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
> # Add the matrices.
> result <- matrix1 + matrix2
> cat("Result of addition","\n")
Result of addition
> print(result)
Output
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
> # Subtract the matrices
> result <- matrix1 - matrix2
> cat("Result of subtraction","\n")
Result of subtraction
> print(result)
Output
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
> # Multiply the matrices.
> result <- matrix1 * matrix2
> cat("Result of multiplication","\n")
Result of multiplication
> print(result)
Output
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24
> result <- matrix1 / matrix2
> cat("Result of division","\n")
Result of division
> print(result)
Output
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Applications of matrix

1. In geology, Matrices takes surveys and plot graphs, statistics, and used to study in different fields.
2. Matrix is the representation method which helps in plotting common survey things.
3. In robotics and automation, Matrices have the topmost elements for the robot movements.
4. Matrices are mainly used in calculating the gross domestic products in Economics, and it also helps in calculating the capability of goods and products.
5. In computer-based application, matrices play a crucial role in the creation of realistic seeming motion.

Data Frames in R

- A data frame is a two-dimensional structure similar to array or a table in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a special case of the list in which each component has equal length.
- A data frame is used to store data table and the vectors which are present in the form of a list in a data frame, are of equal length.
- In a simple way, it is a list of equal length vectors. A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.
- There are following characteristics of a data frame.
    - The columns name should be non-empty.
    - The rows name should be unique.
    - The data which is stored in a data frame can be a factor, numeric, or character type.
    - Each column contains the same number of data items

How to create data frame?

In R, the data frames are created with the help of frame() function of data. This function contains the vectors of any type such as numeric, character, or integer. In below example, we create a data frame that contains employee id (integer vector), employee name(character vector), salary(numeric vector), and starting date(Date vector).

```
# Creating the data frame.
emp.data<- data.frame(
employee_id = c (101:105),
employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
sal = c(40000, 35000, 20000, 25000, 30000),

starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-01",
    "2020-03-05")),
stringsAsFactors = FALSE
)
# Printing the data frame.
print(emp.data)
```

Output

	employee_id	employee_name	sal	starting_date
1	101	Ram	40000	2020-01-01
2	102	Sham	35000	2019-09-01
3	103	Neha	20000	2021-01-01
4	104	Siya	25000	2019-05-01
5	105	Sumit	30000	2020-03-05

Note : The argument 'stringsAsFactors' is an argument to the 'data.frame()' function in R. It is a logical that indicates whether strings in a data frame should be treated as factor variables or as just plain strings

Extracting data from data frame

The data of the data frame is very crucial for us. To manipulate the data of the data frame, it is essential to extract it from the data frame. We can extract the data in three ways which are as follows:

1. We can extract the specific columns from a data frame using the column name.

```
    # Creating the data frame.
    emp.data<- data.frame(
    employee_id = c (101:105),
    employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
    sal = c(40000, 35000, 20000, 25000, 30000),

    starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-
    01", "2020-03-05")),
    stringsAsFactors = FALSE
    )
# Extracting specific columns from a data frame
final <- data.frame(emp.data$employee_id,emp.data$sal)
print(final)
```

Output

	emp.data.employee_id	emp.data.sal
1	101	40000
2	102	35000
3	103	20000
4	104	25000
5	105	30000

2. We can extract the specific rows also from a data frame.

```
# Creating the data frame.
      emp.data<- data.frame(
      employee_id = c (101:105),
      employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
      sal = c(40000, 35000, 20000, 25000, 30000),
      starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-01", "2020-03-05")),
      stringsAsFactors = FALSE
      )
      # Extracting first row from a data frame
      final <- emp.data[1,]
      print(final)
```

Output

```
   employee_id employee_name   sal   starting_date
1        101          Ram     40000    2020-01-01
```

```
1.          # Extracting last two row from a data frame
2.          final <- emp.data[4:5,]
3.          print(final)
```

Output

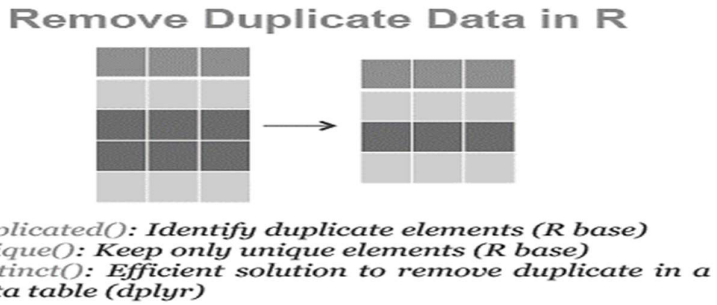
```
              employee_id employee_name   sal   starting_date
4        104          Siya      25000    2019-05-01
5        105         Sumit       30000    2020-03-05
```

3. We can extract the specific rows corresponding to specific columns

```
# Creating the data frame.
emp.data<- data.frame(
employee_id = c (101:105),
employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
sal = c(40000, 35000, 20000, 25000, 30000),
starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-01", "2020-03-05")),
stringsAsFactors = FALSE
)
# Extracting 2nd and 3rd row corresponding to the 1st and 4th column
final <- emp.data[c(2,3),c(1,4)]
print(final)
```

Output

```
  employee_id starting_date
2        102    2019-09-01
3        103    2021-01-01
```



```
> companies <- data.frame(Shares = c("TCS", "Reliance", "HDFC Bank", "Infosys", "Reliance"),
+ Price = c(3200, 1900, 1500, 2200, 1900))
> companies
     Shares Price
1       TCS  3200
2  Reliance  1900
3 HDFC Bank  1500
4    Infosys 2200
5  Reliance  1900
> cat("After removing Duplicates ", "\n")
After removing Duplicates
> companies[duplicated(companies),]
    Shares Price
5 Reliance  1900
> unique(companies)
     Shares Price
1       TCS  3200
2  Reliance  1900
3 HDFC Bank  1500
4   Infosys  2200
```

What is Factors in R?

- The factor is a data structure which is used for fields which take only predefined finite number of values.
- These are the variable which takes a limited number of different values.
- These are the data objects which are used to store categorical data as levels.
- It can store both integers and strings values, and are useful in the column that has a limited number of unique values.
- By default R always sorts levels in alphabetical order.

Attributes of Factor

- **X**
  It is the input vector which is to be transformed into a factor.
- **levels**
  It is an input vector that represents a set of unique values which are taken by x.
- **labels**
  It is a character vector which corresponds to the number of labels.
- **Exclude**
  It is used to specify the value which we want to be excluded,
- **ordered**
  It is a logical attribute which determines if the levels are ordered.
- **nmax**
  It is used to specify the upper bound for the maximum number of level.

How to create factor in R ?
Factors are created using the **factor ()** function by taking a vector as input

Example :

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name)
Output

[1] Ram  Siya Raj  Sham Ram
Levels: Raj Ram Sham Siya
```

Notice that in the output, we have four levels “Raj”, “Ram” ,”Sham” and “Siya”. This does not contain the redundant “Ram” that was occurring twice in our input vector.

In order to add this missing level to our factors, we use the “levels” attribute as follows:

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name, levels=c(“Ram”,”Siya”,”Raj”,”Sham”))
Output

[1] Ram  Siya Raj  Sham Ram
Levels: Raj Ram Sham Siya
```

In order to provide abbreviations or ‘labels’ to our levels, we make use of the labels argument as follows –

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name, levels=c("Ram","Siya","Raj","Sham"),labels = c("R1","S1","R2","S2"))
Output

[1] R1 S1 R2 S2 R1
Levels: R1 S1 R2 S2
```

if you want to exclude any level from your factor, you can make use of the exclude argument.

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name, levels=c("Ram","Siya","Raj","Sham"),exclude="Ram")
Output
[1] <NA> Siya Raj  Sham <NA>
Levels: Siya Raj Sham
```

Accessing component of Factor in R

There are various ways to access the elements of a factor in R. Some of the ways are as follows:

```
# Create vector of compass
>compass <- c("East","West","East","North")
>data <- factor(compass)
>data
```

Output

```
[1] East  West  East  North
Levels: East North West

>data[4] #Accessing the 4th element
```

Output

```
[1] North
Levels: East North West

>data[c(2,3)] #Accessing the 2nd & 3rd element
```

Output

```
[1] West East
Levels: East North West
```

```
data[-1] #Accessing everything except 1st element
```

Output

```
[1] West  East  North
Levels: East North West

data[c(TRUE, FALSE, TRUE, TRUE)] #Accessing using Logical Vector
```

Output

```
[1] East  East  North
Levels: East North West
```

The levels() method of the specific factor vector can be used to extract the individual levels in the form of independent strings. It returns the number of elements equivalent to the length of the vector.

Syntax:
levels (fac-vec)[fac-vec]

Since, these levels are each returned as strings, therefore they belong to the same data type. Hence, they are combined now to form an atomic vector using the c() method.

Syntax:
c (vec1 , vec2)

where vec1 and vec2 are vectors belonging to same data type

```
> fac1 <- factor(letters[1:3])
> print ("Factor1 : ")
[1] "Factor1 : "
> print (fac1)
[1] a b c
Levels: a b c
> sapply(fac1,class)
[1] "factor" "factor" "factor"
> fac2 <- factor(c(1:4))
> print ("Factor2 : ")
[1] "Factor2 : "
> print (fac2)
[1] 1 2 3 4
Levels: 1 2 3 4
> sapply(fac2,class)
[1] "factor" "factor" "factor" "factor"
>
> # extracting levels of factor1
> level1 <- levels(fac1)[fac1]
>
> # extracting levels of factor2
> level2 <- levels(fac2)[fac2]
```

```
> # combine into one factor
> combined <- factor(c( level1,level2 ))
> print ("Combined Factor : ")
[1] "Combined Factor : "
> print (combined)
[1] a b c 1 2 3 4
Levels: 1 2 3 4 a b c
>
> sapply(combined,class)
[1] "factor" "factor" "factor" "factor" "factor" "factor" "factor"
```

Factor Functions in R

Some of these functions; *is.factor(), as.factor(), is.ordered(), as.ordered().*

- *is.factor()* checks if the input is present in the form of factor and returns a Boolean value (TRUE or FALSE).
- as.factor() takes the input (usually a vector) and converts it into a factor.
- is.ordered() checks if the factor is ordered and returns boolean TRUE or FALSE.
- The as.ordered() function takes an unordered function and returns a factor that is arranged in order.

Working with Dataset

1. **Dataset built into R**

This dataset is built into R ,to get information on the dataset is by typing ? before the dataset. Information will appear on the 4th panel under the Help tab.

```
?women
```

To see the whole dataset, use the command, View(*data_frame*). In this case, it would be View(women). A new tab in the Source panel called women will appear. You should see 15 rows and 2 columns of data entries.

2. **View part of dataset**

   If you want to see only a portion of the dataset, the function head( ) or tail( ) will do the job.

   The function head(*data_frame[,nL]*) will show the first 6 rows of the dataset. (nL : where is total number of rows you want followed by L)

```
head(women)
```

   The function tail(*data_frame*) will show the last 6 rows of the dataset.

```
tail(women)
```

3. **Viewing entries tied to the variable**
   Suppose we want to work on the data of women having height greater than 60

```
ans <- women[women$height>60,]
ans
```

4. **Ordering data frame by variable**
   Use the function called order( ) to order the data frame in descending or ascending order. The syntax in this case is: *data_frame*[order(*data_frame*$*variable*),]

   Suppose we want to order the women data in ascending order of weight of the women

```
data=women
sorted_data=data[order(data$weight),]
sorted_data
```

   To order the data frame, in descending order by weight of the women, put negative sign in front of the target vector.

```
data=women
sorted_data=data[order(-data$weight),]
sorted_data
```

R Data Visualization

In R, we can create visually appealing data visualizations by writing few lines of code. For this purpose, we use the diverse functionalities of R. Data visualization is an efficient technique for

gaining insight about data through a visual medium. With the help of visualization techniques, a human can easily obtain information about hidden patterns in data that might be neglected.

By using the data visualization technique, we can work with large datasets to efficiently obtain key insights about it.

Standard Graphics

R standard graphics are available through package graphics, include several functions which provide statistical plots, like:

- Scatterplots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is −

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
# Get the input values.
input <- mtcars[,c('wt','mpg')]

 # Give the chart file a name.
 png(file = "scatterplot.png")

 # Plot the chart for cars with weight between 2.5 to 5 and mileage #between 15 and 30.
 plot(x = input$wt,y = input$mpg,
     xlab = "Weight",
     ylab = "Milage",
     xlim = c(2.5,5),
     ylim = c(15,30),
     main = "Weight vs Milage"
 )

 # Save the file.
 dev.off()
```

Pie charts

The Pie charts are created with the help of pie () function, which takes positive numbers as vector input. Additional parameters are used to control labels, colors, titles, etc.

There is the following syntax of the pie() function:

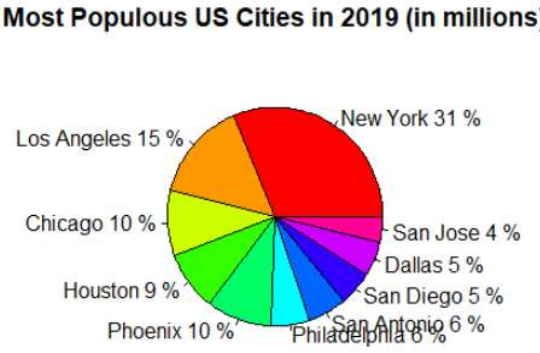
pie(X, Labels, Radius, Main, Col, Clockwise)

Here,

1. **X** is a vector that contains the numeric values used in the pie chart.
2. **Labels** are used to give the description to the slices.
3. **Radius** describes the radius of the pie chart.
4. **Main** describes the title of the chart.
5. **Col** defines the color palette.
6. **Clockwise** is a logical value that indicates the clockwise or anti-clockwise direction in which slices are drawn.

```
Cities <- c("New York", "Los Angeles", "Chicago", "Houston", "Phoenix",
+            "Philadelphia", "San Antonio", "San Diego", "Dallas", "San Jose")
> Population <- c(8.60, 4.06, 2.68, 2.40, 2.71, 1.58, 1.57, 1.45, 1.40, 1.03 )
> top_ten <- data.frame(Cities, Population)
> top_ten
pct <- round(100*top_ten$Population/sum(top_ten$Population))
> # Draw pie chart
> pie(top_ten$Population,
+     labels = paste(top_ten$Cities, sep = " ", pct, "%"),
+     col = rainbow(length(top_ten$Population)),
+     main = "Most Populous US Cities in 2019 (in millions)")
```

Output:



- **Bar plots**

  The barplot default is a vertical bar graph with black borders and gray filling. The bar graph is drawn in the same order as the data frame entries.

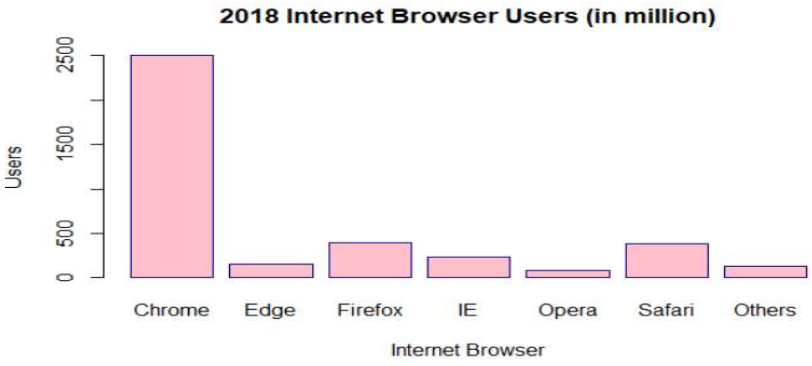
To add a title, labels on the axes and color to your bar graph, we use the following arguments.

- main = "*Header of the graph*"
- xlab = "*x-axis label*"
- ylab = "*y-axis label*"
- name.arg = *vector* (used for labelling each of the bar graphs)
- border = "*bar graph border color*"
- col = "*color to fill bar graph*"

```
barplot(height = IB$Users,
        main = "2018 Internet Browser Users (in million)",
        xlab = "Internet Browser",
        ylab = "Users",
        names.arg = IB$Browser,
        border = "dark blue",
        col = "pink")
```

```
Browser <- c("Chrome", "Edge", "Firefox", "IE",
+              "Opera", "Safari", "Others")
> Users <- c(2502.4, 150.78, 395.83, 238.05, 86.49, 387.65, 134.8)
> IB <- data.frame(Browser, Users)
> IB
  > barplot(height = IB$Users,
+         main = "2018 Internet Browser Users (in million)",
+         xlab = "Internet Browser",
+         ylab = "Users",
+         names.arg = IB$Browser,
+         border = "dark blue",
+         col = "pink")
```

Output:



Set A

1. Write a R program to add, multiply and divide two vectors of integer type. (vector length should be minimum 4)
2. Write a R program to calculate the multiplication table using a function.
3. Write a R program to sort a list of strings in ascending and descending order.
4. Write a script in R to create a list of employees and perform the following:
   a. Display names of employees in the list.
   b. Add an employee at the end of the list.
   c. Remove the third element of the list.

Set B

1. Write a R program to reverse a number and also calculate the sum of digits of that number.
2. Write a R program to calculate the sum of two matrices of given size.
3. Write a R program to concatenate two given factors.
4. Write a R program to create a data frame using two given vectors and display the duplicate elements
5. Write a R program to perform the following:
   a. Display all rows of the data set having weight greater than 120.
   b. Display all rows of data set in ascending order of weight.
   (Use inbuilt data set woman)
6. Write a R program to perform the following:
   a. Display all the cars having mpg more than 20.
   b. Subset the data set by mpg column for values greater than 15.0
   c. Display all cars having four gears.
   (Use inbuilt data set mtcar)

Set C

1. Write a R Program to perform the following:
   a. Create a Scattered plot to compare wind speed and temperature.
   b. Create a Scattered plot to show the relationship between ozone and wind values by giving appropriate values to colour argument.
   c. Create a Bar plot to show the ozone level for all the days having temperature > 70.
   (Use inbuilt dataset air quality)

Signature of the Instructor: **Date:**

Assignment Evaluation

0: Not Done	2: Late Complete	4: Complete
1: Complete	3. Needs Improvemen	5: Well Done

Assignment 2: Data Pre-processing

Objectives:

Students will be able to

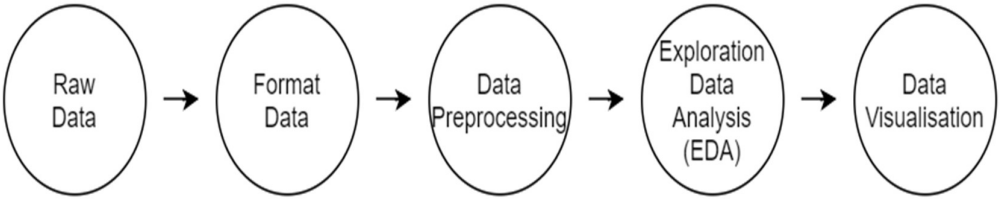
- To understand the need for data pre-processing.
- To identify different phases of data pre-processing such as data cleaning, data integration, Data transformation and data reduction

Prerequisites:

- ✓ Basic knowledge of Python Programming
- ✓ https://fdocuments.in/document/unit-1-data-preprocessing-data-preprocessing-learning-objectives-understand-why-preprocess-the-data-understand-how-to-clean-the-data-understand-how.html
- ✓ https://parteekbhatia.com/wp-content/uploads/2020/01/Chapter4.pdf

Introduction:

Pre-processing refers to the changes applied to our data before feeding it to the ML algorithm. Data pre-processing is a technique that is used to convert the created or collected (raw) data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis or processing by ML model. Following figure shows transformation processing performed on raw data before, during and after applying ML techniques:



Importance of Data Pre-processing:

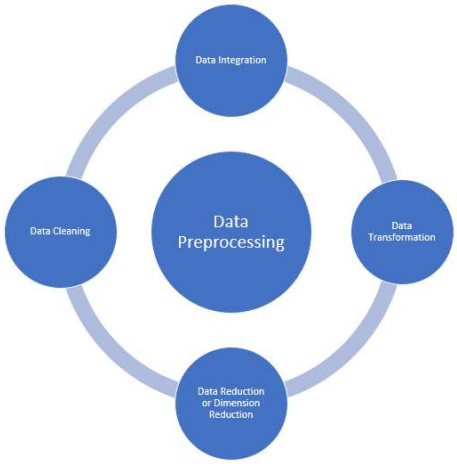
Pre-processing of data is mainly to check the data quality. The quality can be checked by the following

Accuracy: To check whether the data entered is correct or not.
Completeness: To check whether the data is available or not recorded.
Consistency: To check whether the same data is kept in all the places that do or do not match.
Timeliness: The data should be updated correctly.
Believability: The data should be trustable.
Interpretability: The understandability of the data.

Data Pre-processing:

There are seven significant steps in data pre-processing in Machine Learning:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation



1. Data Integration:-

The process of combining multiple sources into a single dataset. The Data integration process is one of the main components in data management. There are some problems to be considered during data integration.

1. **Schema integration**: Integrates metadata (a set of data that describes other data) from different sources.
2. **Entity identification problem:** Identifying entities from multiple databases. For example, the system or the use should know student _id of one database and student_name of another database belongs to the same entity.
3. **Detecting and resolving data value concepts**: The data taken from different databases while merging may differ. Like the attribute values from one database may differ from another database. For example, the date format may differ like "MM/DD/YYYY" or "DD/MM/YYYY".

2. Data Cleaning:-

Data cleaning is the process of preparing raw data for analysis by removing bad data, organizing the raw data, and filling in the null values. Ultimately, cleaning data prepares the data for the process of data mining when the most valuable information can be pulled from the data set.

a. **Missing Data:**
   This situation arises when some data is missing in the data. It can be handled in various ways. Some of them are:
   Ignore the tuples:
   This approach is suitable only when the dataset we have is quite large and multiple values are missing within a tuple.
   Fill the Missing values:
   There are various ways to do this task. You can choose to fill the missing values manually, by attribute mean or the most probable value.

b. **Noisy Data:**
   Noisy data is a meaningless data that can't be interpreted by machines. It can be generated due to faulty data collection, data entry errors etc. It can be handled in following ways :
   Binning Method:

This method works on sorted data in order to smooth it. The whole data is divided into segments of equal size and then various methods are performed to complete the task. Each segmented is handled separately. One can replace all data in a segment by its mean or boundary values can be used to complete the task.

First sort data and partition

Then one can smooth by bin mean, median and boundaries.

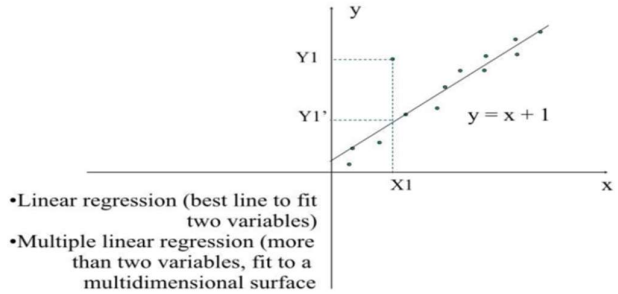
For Example:

Sorted data for price (in dollars): 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34

* Partition into bins:
  - Bin 1: 4, 8, 9, 15
  - Bin 2: 21, 21, 24, 25
  - Bin 3: 26, 28, 29, 34
* Smoothing by bin means:
  - Bin 1: 9, 9, 9, 9
  - Bin 2: 23, 23, 23, 23
  - Bin 3: 29, 29, 29, 29
* Smoothing by bin boundaries:
  - Bin 1: 4, 4, 4, 15
  - Bin 2: 21, 21, 25, 25
- Bin 3: 26, 26, 26, 34

c. Regression Method:

Here data can be smoothed by fitting the data to a function. Linear regression involves finding the "best" line to fit two attributes, so that one attribute can be used to predict the other. Multiple linear regression is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface.



3. Data reduction:

This process helps in the reduction of the volume of the data which makes the analysis easier yet produces the same or almost the same result. This reduction also helps to reduce storage space. There are some of the techniques in data reduction are Dimensionality reduction, Numerosity reduction, Data compression.

1. **Dimensionality reduction:**

   This process is necessary for real-world applications as the data size is big. In this process, the reduction of random variables or attributes is done so that the

dimensionality of the data set can be reduced. Combining and merging the attributes of the data without losing its original characteristics. This also helps in the reduction of storage space and computation time is reduced. When the data is highly dimensional the problem called "Curse of Dimensionality" occurs.

2. **Numerosity Reduction:**
   In this method, the representation of the data is made smaller by reducing the volume. There will not be any loss of data in this reduction.
3. **Data compression:**
   The compressed form of data is called data compression. This compression can be lossless or lossy. When there is no loss of information during compression it is called lossless compression. Whereas lossy compression reduces information but it removes only the unnecessary information.

4. Data Transformation:

The change made in the format or the structure of the data is called data transformation. This step can be simple or complex based on the requirements. There are some methods in data transformation.

1. **Smoothing**: With the help of algorithms, we can remove noise from the dataset and helps in knowing the important features of the dataset. By smoothing we can find even a simple change that helps in prediction.
2. **Aggregation**: In this method, the data is stored and presented in the form of a summary. The data set which is from multiple sources is integrated into with data analysis description. This is an important step since the accuracy of the data depends on the quantity and quality of the data. When the quality and the quantity of the data are good the results are more relevant.
3. **Discretization**: The continuous data here is split into intervals. Discretization reduces the data size. For example, rather than specifying the class time, we can set an interval like (3 pm-5 pm, 6 pm-8 pm).
4. **Normalization:** It is the method of scaling the data so that it can be represented in a smaller range. Example ranging from -1.0 to 1.0.

5. Training-Test Data Split :-

Just before training supervised machine learning model, this is usually the last step of data pre-processing in which for a given data set, after undergoing all cleaning and transformation, is divided into two parts – one for training of machine learning model and second for testing the trained model. Though there is no rule of thumb, but usually training-test split is done randomly at 80%-20% ratio. While splitting data set, care has to be taken that there is no loss of information in training data set.

Execution of Data Pre-processing methods using Python commonly involves following steps:

- Importing the libraries
- Importing the Dataset
- Handling of Missing Data
- Handling of Categorical Data

- Splitting the dataset into training and testing datasets
- Feature Scaling

1. **Importing the libraries**

```
import numpy as np
import pandas as pd
import seaborn as sns
```

2. **Importing the Dataset**

There are several ways to import a dataset. The simplest one is importing the dataset from a .csv file. We use the read_csv method of the pandas library to read a local CSV file as a dataframe.

First of all, let us have a look at the dataset we are going to use for this particular example. You can download or take this dataset from : **https://github.com/tarunlnmiit/machine_learning/blob/master/DataPreprocessing.csv**

```
dataset = pd.read_csv('Data.csv')
```

output:

	Region	Age	Income	Online Shopper
2	India	49	86400	No
3	Brazil	32	57600	Yes
4	USA	35	64800	No
5	Brazil	43	73200	No
6	USA	45		Yes
7	India	40	69600	Yes
8	Brazil		62400	No
9	India	53	94800	Yes
10	USA	55	99600	No
11	India	42	80400	Yes

3. **Extracting dependent and independent variables:**

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are **Country, Age**, and **Salary**, and one is a dependent variable which is **Purchased**.

4. **Extracting independent variable:**

To extract an independent variable, we will use **iloc[ ]** method of Pandas library. It is used to extract the required rows and columns from the dataset.

```
x= dataset.iloc[:,:-1].values
```

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

5. **Extracting dependent variable:**

To extract dependent variables, again, we will use Pandas .iloc[] method.

```
y= dataset.iloc[:,3].values
```

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

6. Dealing with Missing Data:

handling the missing data and replace missing values with nan from numpy and replace with mean of all the other values

```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer = imputer.fit(X[:, 1:])
X[:, 1:] = imputer.transform(X[:, 1:])
```

After execution of this code, the independent variable *X* will transform into the following.

Output:

X - NumPy object array (read only)

	0	1	2
0	India	49.0	86400.0
1	Brazil	32.0	57600.0
2	USA	35.0	64800.0
3	Brazil	43.0	73200.0
4	USA	45.0	76533.3333…
5	India	40.0	69600.0
6	Brazil	43.7777777…	62400.0
7	India	53.0	94800.0
8	USA	55.0	99600.0
9	India	42.0	80400.0

7. **Handling of Categorical Data**

In this dataset we can see that we have two categorical variables namely Region variable and the Online Shopper variable. These two variables are categorical variables because simply they contain categories. The Region contains three categories. It's **India, USA & Brazil** and the online shopper variable contains two categories. ***Yes*** and ***No*** that's why they're called categorical variables.

You can guess that since machine learning models are based on mathematical equations you can naturally understand that it would cause some problem if we keep the text here in the categorical variables in the equations because we would only want numbers in the equations. So that's why we need to encode the categorical variables. That is to encode the text that we have here into numbers. To do this we use the following code snippet.

```
# encode categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
```

```
rg = ColumnTransformer([("Region", OneHotEncoder(), [0])], remainder =
'passthrough')
X = rg.fit_transform(X)
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)
```

After execution of this code, the independent variable *X* and dependent variable *Y* will transform into the following.

X - NumPy object array (read only)

	0	1	2	3	4
0	0.0	1.0	0.0	49.0	86400.0
1	1.0	0.0	0.0	32.0	57600.0
2	0.0	0.0	1.0	35.0	64800.0
3	1.0	0.0	0.0	43.0	73200.0
4	0.0	0.0	1.0	45.0	76533.3333…
5	0.0	1.0	0.0	40.0	69600.0
6	1.0	0.0	0.0	43.7777777…	62400.0
7	0.0	1.0	0.0	53.0	94800.0
8	0.0	0.0	1.0	55.0	99600.0
9	0.0	1.0	0.0	42.0	80400.0

8. **Splitting the Dataset into Training and Testing Datasets**

Any machine learning algorithm needs to be tested for accuracy. In order to do that, we divide our data set into two parts: **training set** and **testing set.** As the name itself suggests, we use the training set to make the algorithm learn the behaviours present in the data and check the correctness of the algorithm by testing on testing set. In Python, we do that as follows:

```
# splitting the dataset into training set and test set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
random_state=0)
```

Here, we are taking training set to be 80% of the original data set and testing set to be 20% of the original data set. This is usually the ratio in which they are split. But, you can come across sometimes to a 70–30% or 75–25% ratio split. But you don't want to split it 50–50%. This can lead to **Model Overfitting.** For now, we are going to split it in 80–20% ratio.

Python Program:

```
import pandas as pd
dataset = pd.read_csv("play_tennis.csv")          # (Download  dataset from
github.com)
dataset
```

	outlook	temp	humidity	wind	play
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

y = dataset.play
x = dataset.drop('play',axis=1)
print(y)

```
0      No
1      No
2     Yes
3     Yes
4     Yes
5      No
6     Yes
7      No
8     Yes
9     Yes
10    Yes
11    Yes
12    Yes
13     No
Name: play, dtype: object
```

print(x)

```
      outlook  temp humidity    wind
0       Sunny   Hot     High    Weak
1       Sunny   Hot     High  Strong
2    Overcast   Hot     High    Weak
3        Rain  Mild     High    Weak
4        Rain  Cool   Normal    Weak
5        Rain  Cool   Normal  Strong
6    Overcast  Cool   Normal  Strong
7       Sunny  Mild     High    Weak
8       Sunny  Cool   Normal    Weak
9        Rain  Mild   Normal    Weak
10      Sunny  Mild   Normal  Strong
11   Overcast  Mild     High  Strong
12   Overcast   Hot   Normal    Weak
13       Rain  Mild     High  Strong
```

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
print(x_train)

	outlook	temp	humidity	wind
8	Sunny	Cool	Normal	Weak
5	Rain	Cool	Normal	Strong
2	Overcast	Hot	High	Weak
0	Sunny	Hot	High	Weak
12	Overcast	Hot	Normal	Weak
4	Rain	Cool	Normal	Weak
11	Overcast	Mild	High	Strong
7	Sunny	Mild	High	Weak
13	Rain	Mild	High	Strong
6	Overcast	Cool	Normal	Strong
10	Sunny	Mild	Normal	Strong

print(x_test)

	outlook	temp	humidity	wind
9	Rain	Mild	Normal	Weak
1	Sunny	Hot	High	Strong
3	Rain	Mild	High	Weak

9. Feature Scaling:

Feature Scaling or Standardization: It is **a step of Data Pre Processing that is applied to independent variables or features of data**. It basically helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

Min-Max Normalization: This technique re-scales a feature or observation value with distribution value between 0 and 1.

$$X_{\text{new}} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

Standardization: It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1.

$$X_{\text{new}} = \frac{X_i - X_{\text{mean}}}{\text{Standard Deviation}}$$

Python Program:

```
import pandas as pd
import sklearn
from sklearn import preprocessing as per
df = pd.read_csv("Data.csv")
print(df.head())
scaler = per.MinMaxScaler(feature_range=(0,1))
```

```
rescaleData = scaler.fit_transform(df)
rescaleData =
pd.DataFrame(rescaleData,index=df.index,columns=df.columns)
print(rescaleData)
```

Set A

1. Write a python program to find all null values in a given data set and remove them. (Download dataset from github.com)
2. Write a python program the Categorical values in numeric format for a given dataset.

Set B

1. Write a python program to rescale the data between 0 and 1. (use inbuilt dataset)
2. Write a python program to splitting the dataset into training and testing set.

Set C

1. Write a python program to implement complete data pre-processing in a given data set.(missing value, encoding categorical value, Splitting the dataset into the training and test sets and feature scaling.(Download dataset from github.com).

Signature of the Instructor: **Date:**

Assignment Evaluation

0: Not Done	2: Late Complete	4: Complete
1: Complete	3. Needs Improvement	5: Well Done

Assignment 3: Classification

Objectives:

Students will be able to

- Understand classification process and its different terminologies
- Demonstrate and apply an understanding of the basic classification algorithmic methods that support knowledge discovery
- Be able to evaluate what has been learned through the application of the appropriate statistics used in classification techniques.

Prerequisites:

- ✓ Basic knowledge of Python programming
- ✓ https://www.upgrad.com/blog/classification-in-data-mining/
- ✓ https://byjus.com/commerce/meaning-and-objectives-of-classification-of-data/

Introduction:

Classification is a data-mining technique that **assigns categories to a collection of data to aid in more accurate predictions and analysis**. Classification is one of several methods intended to make the analysis of very large datasets effective.

There are two forms of data analysis that can be used for extracting models describing important classes or to predict future data trends. These two forms are as follows −

- Classification
- Prediction

Classification models predict categorical class labels; and prediction models predict continuous valued functions. For example, we can build a classification model to categorize bank loan applications as either safe or risky, or a prediction model to predict the expenditures in dollars of potential customers on computer equipment given their income and occupation.

What is classification?

Following are the examples of cases where the data analysis task is Classification −

- A bank loan officer wants to analyse the data in order to know which customer (loan applicant) is risky or which are safe.
- A marketing manager at a company needs to analyse a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

What is prediction?

Following are the examples of cases where the data analysis task is Prediction −

Suppose the marketing manager needs to predict how much a given customer will spend during a sale at his company. In this example we are bothered to predict a numeric value. Therefore the data analysis task is an example of numeric prediction. In this case, a model or a predictor will be constructed that predicts a continuous-valued-function or ordered value.

Note − Regression analysis is a statistical methodology that is most often used for numeric prediction.

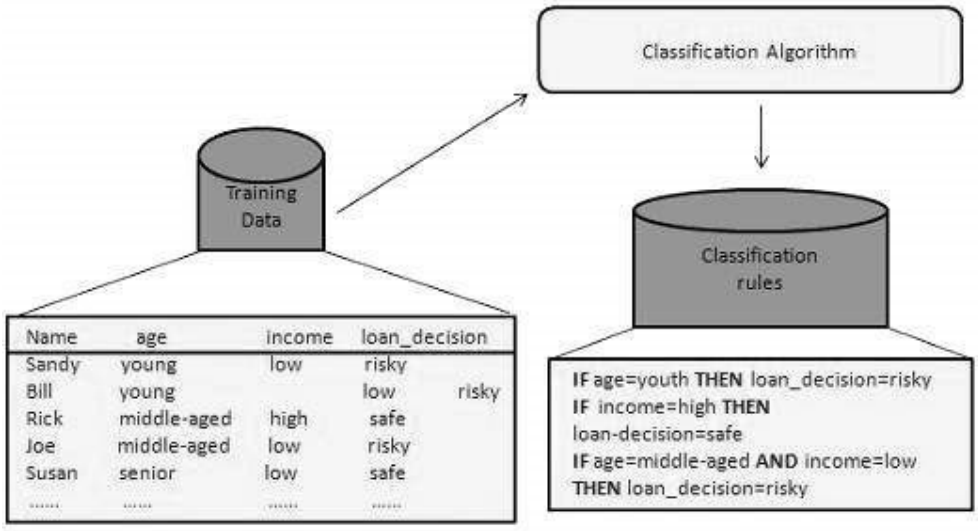
How Does Classification Works?

With the help of the bank loan application that we have discussed above, let us understand the working of classification. The Data Classification process includes two steps −

- Building the Classifier or Model
- Using Classifier for Classification

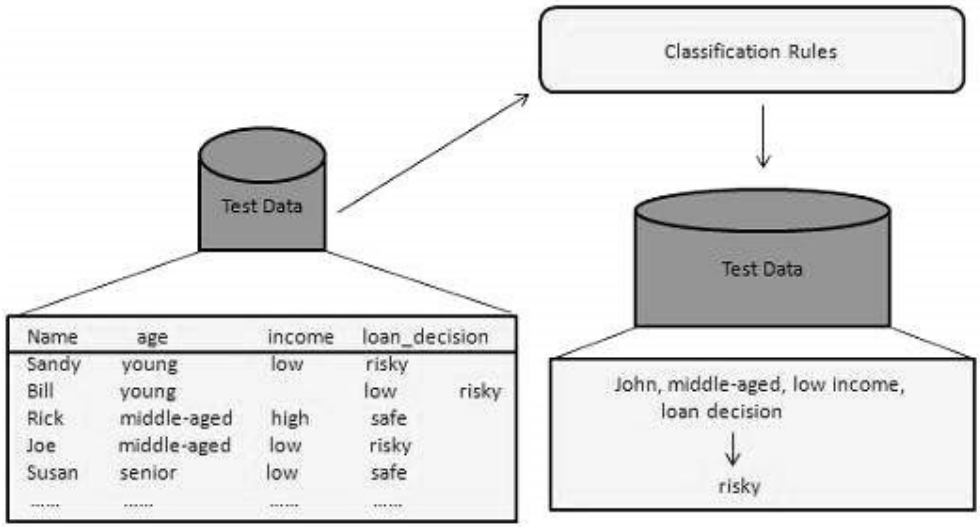
Building the Classifier or Model

- This step is the learning step or the learning phase.
- In this step the classification algorithms build the classifier.
- The classifier is built from the training set made up of database tuples and their associated class labels.
- Each tuple that constitutes the training set is referred to as a category or class. These tuples can also be referred to as sample, object or data points.



Using Classifier for Classification

In this step, the classifier is used for classification. Here the test data is used to estimate the accuracy of classification rules. The classification rules can be applied to the new data tuples if the accuracy is considered acceptable.



Classification and Prediction Issues

The major issue is preparing the data for Classification and Prediction. Preparing the data involves the following activities −

- **Data Cleaning** − Data cleaning involves removing the noise and treatment of missing values. The noise is removed by applying smoothing techniques and the problem of missing values is solved by replacing a missing value with most commonly occurring value for that attribute.

- **Relevance Analysis** − Database may also have the irrelevant attributes. Correlation analysis is used to know whether any two given attributes are related.
- **Data Transformation and reduction** − The data can be transformed by any of the following methods.
    - **Normalization** − The data is transformed using normalization. Normalization involves scaling all values for given attribute in order to make them fall within a small specified range. Normalization is used when in the learning step, the neural networks or the methods involving measurements are used.
    - **Generalization** − The data can also be transformed by generalizing it to the higher concept. For this purpose we can use the concept hierarchies.

Note − Data can also be reduced by some other methods such as wavelet transformation, binning, histogram analysis, and clustering.

Comparison of Classification and Prediction Methods

Here is the criteria for comparing the methods of Classification and Prediction −

- **Accuracy** − Accuracy of classifier refers to the ability of classifier. It predict the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.
- **Speed** − This refers to the computational cost in generating and using the classifier or predictor.
- **Robustness** − It refers to the ability of classifier or predictor to make correct predictions from given noisy data.
- **Scalability** − Scalability refers to the ability to construct the classifier or predictor efficiently; given large amount of data.
- **Interpretability** − It refers to what extent the classifier or predictor understands.

Data Mining algorithms for Classification are as follows

1) Decision Tree

2) Naïve Bayes

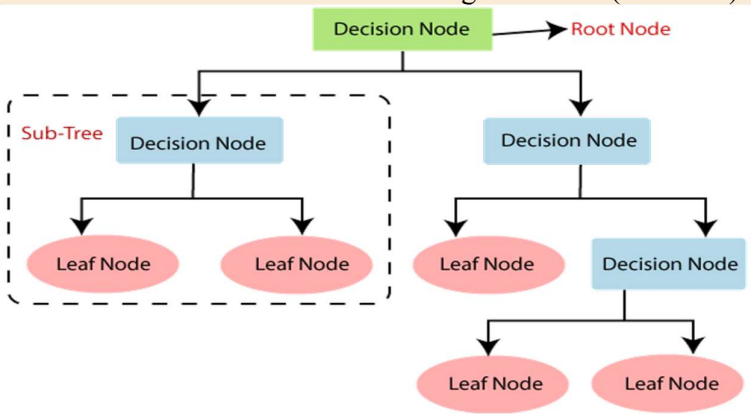
3) Support vector Machine

1) Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:
-

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

 Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

 Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

 Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

 Branch/Sub Tree: A tree formed by splitting the tree.

 Pruning: Pruning is the process of removing the unwanted branches from the tree.

 Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

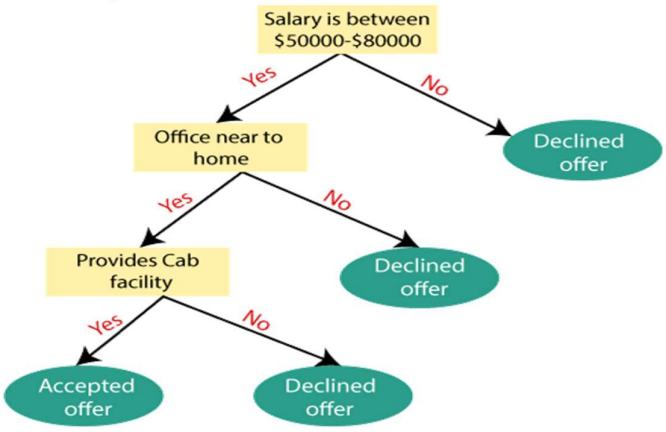
In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Minister of India (1947-2020)

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

Gini Index= $1- \sum_j P_j^2$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Advantages of the Decision Tree

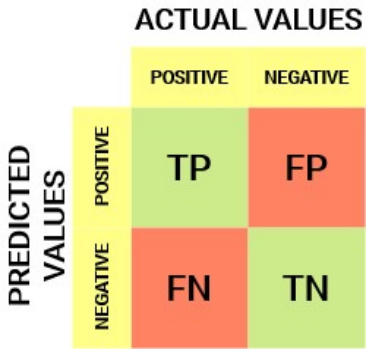
- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Confusion Matrix:

- A confusion matrix is a summary of prediction results on a classification problem.
- The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix.
- The confusion matrix shows the ways in which your classification model is confused when it makes predictions.
- A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.



- The target variable has two values: **Positive** or **Negative**
- The **columns** represent the **actual values** of the target variable
- The **rows** represent the **predicted values** of the target variable

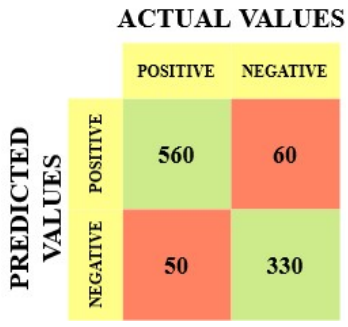
True Positive (TP)
- The predicted value matches the actual value
- The actual value was positive and the model predicted a positive value

True Negative (TN)
- The predicted value matches the actual value
- The actual value was negative and the model predicted a negative value

False Positive (FP) –
- The predicted value was falsely predicted
- The actual value was negative but the model predicted a positive value

False Negative (FN) –
- The predicted value was falsely predicted
- The actual value was positive but the model predicted a negative value

Example:
Let me give you an example to better understand this. Suppose we had a classification dataset with 1000 data points. We fit a classifier on it and get the below confusion matrix:



- True Positive (TP) = 560; meaning 560 positive class data points were correctly classified by the model
- True Negative (TN) = 330; meaning 330 negative class data points were correctly classified by the model

- False Positive (FP) = 60; meaning 60 negative class data points were incorrectly classified as belonging to the positive class by the model
- False Negative (FN) = 50; meaning 50 positive class data points were incorrectly classified as belonging to the negative class by the model

Classification Accuracy:
Classification Accuracy is given by the relation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:
Recall is defined as the ratio of the total number of correctly classified positive classes divide by the total number of positive classes. Or, out of all the positive classes, how much we have predicted correctly.

$$Recall = \frac{TP}{TP + FN} \text{ or } \frac{\text{True Positive}}{\text{Actual Results}}$$

Precision:
Precision is defined as the ratio of the total number of correctly classified positive classes divided by the total number of predicted positive classes. Or, out of all the predictive positive classes, how much we predicted correctly.

$$Precision = \frac{TP}{TP + FP} \text{ or } \frac{\text{True Positive}}{\text{Predictive Results}}$$

F-score or F1-score:
It is difficult to compare two models with different Precision and Recall. So to make them comparable, we use F-Score. It is the Harmonic Mean of Precision and Recall. As compared to Arithmetic Mean, Harmonic Mean punishes the extreme values more.

$$F\text{-}score = \frac{2 * Recall * Precision}{Recall + Precision}$$

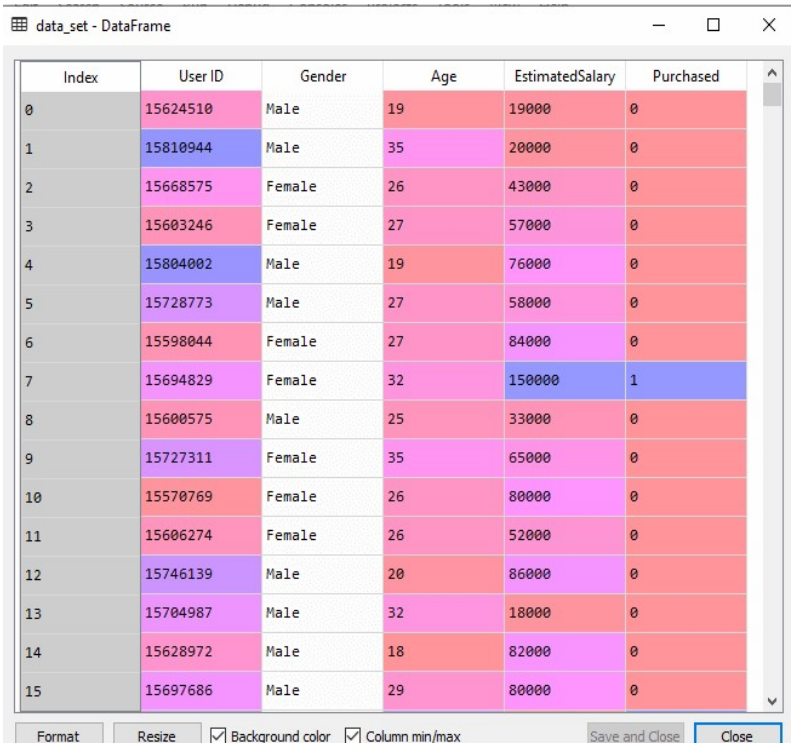
Python Implementation of Decision Tree
Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as KNN SVM, Logistic Regression, etc.
Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

1. Data Pre-Processing Step:
importing libraries

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
  #importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

1. #Fitting Decision Tree classifier to the training set
2. From sklearn.tree **import** DecisionTreeClassifier
3. classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
4. classifier.fit(x_train, y_train)

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy':** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random_state=0":** For generating the random states.

Below is the output for this:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
                random_state=0, splitter='best')
```

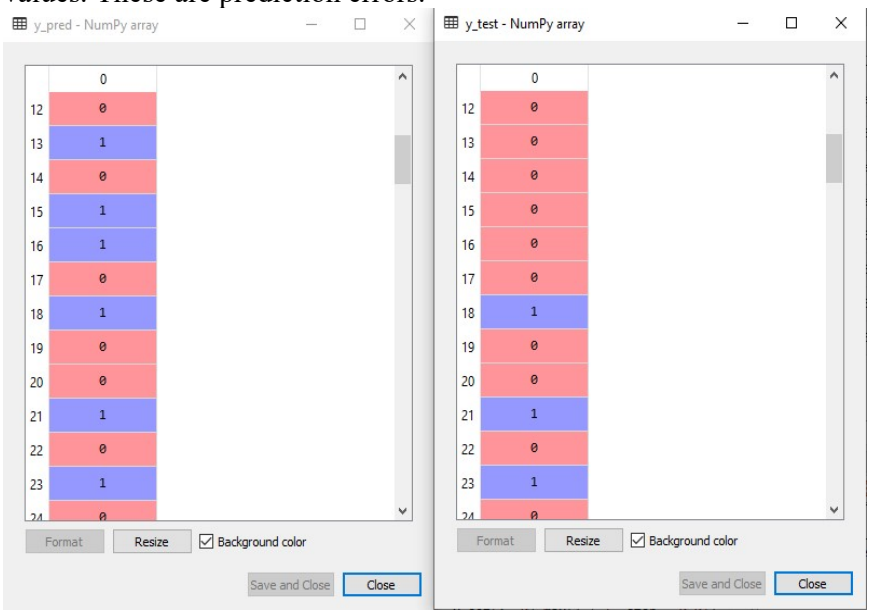
3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector **y_pred.** Below is the code for it:

1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)

Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

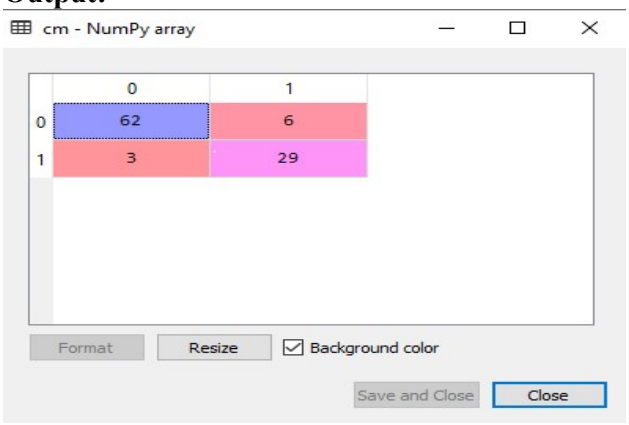


4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

```
1. #Creating the Confusion matrix
2. from sklearn.metrics import confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)
```

Output:



In the above output image, we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and**62+29=91 correct predictions. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.**

5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Logistic Regression. Below is the code for it:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, ste
p  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
,
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
      c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

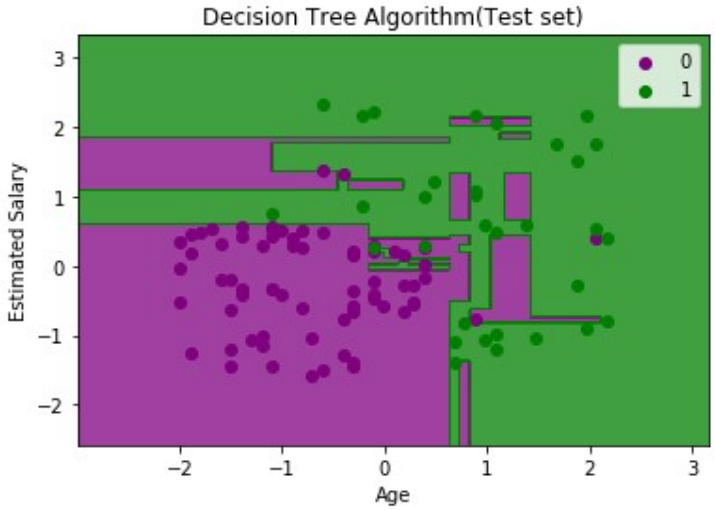
Output:

The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.
As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

6. Visualizing the test set result:
Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, ste
p  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)
,
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



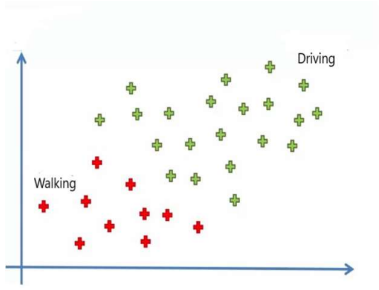
As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

2) Naive Bayes Algorithm

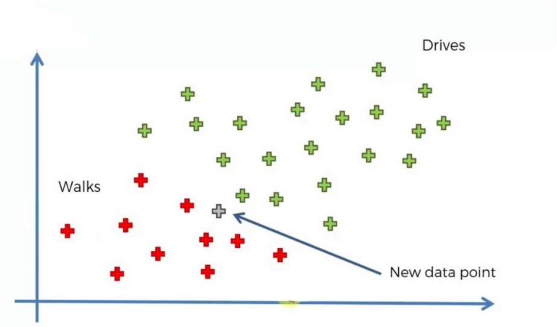
Naive Bayes is a classification algorithm that works based on the Bayes theorem. Before explaining about Naive Bayes, first, we should discuss Bayes Theorem. Bayes theorem is used to find the probability of a hypothesis with given evidence.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

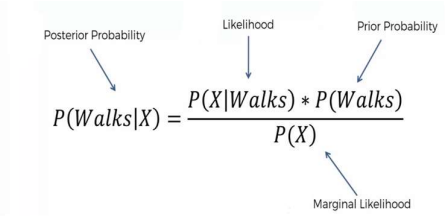
In this, using Bayes theorem we can find the probability of A, given that B occurred. A is the hypothesis and B is the evidence.
P(B|A) is the probability of B given that A is True.
P(A) and P(B) is the independent probabilities of A and B.



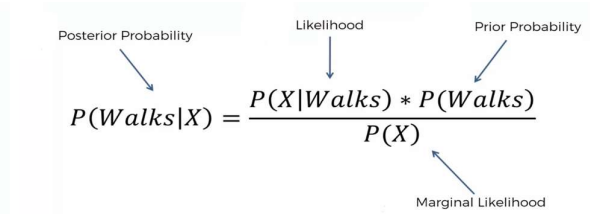
The concept behind the algorithm
Let's understand the concept of the Naive Bayes Theorem through an example. We are taking a dataset of employees in a company, our aim is to create a model to find whether a person is going to the office by driving or walking using salary and age of the person.



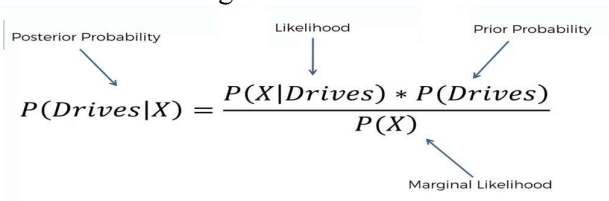
In the above, we can see 30 data points in which red points belong to those who are walking and green belongs to those who are driving. Now let's add a new data point into it. Our aim is to find the category that the new point belongs to.



Note that we are taken age on the X-axis and Salary on the Y-axis. We are using the Naive Bayes algorithm to find the category of the new data point. For this, we have to find the posterior probability of walking and driving for this data point. After comparing, the point belongs to the category having a higher probability.
The posterior probability of walking for the new data point is :



also for the driving is :

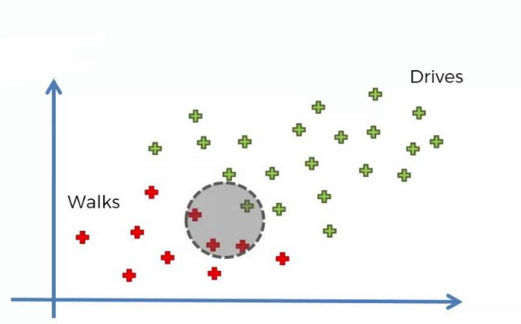


Steps involved in Naive Bayes algorithm
Step 1: We have to find all the probabilities required for the Bayes theorem for the calculation of posterior probability
P(Walks) is simply the probability of those who walk among all

$$P(Walks) = \frac{Number\ of\ Walkers}{Total\ Observations}$$

$$P(Walks) = \frac{10}{30}$$

In order to find the marginal likelihood, P(X), we have to consider a circle around the new data point of any radii including some red and green points.



$$P(X) = \frac{Number\ of\ Similar\ Observations}{Total\ Observations}$$

$$P(X) = \frac{4}{30}$$

P(X|Walks) can be found by :

$$P(X|Walks) = \frac{Number\ of\ Similar\ Observations\ Among\ those\ who\ Walk}{Total\ number\ of\ Walkers}$$

$$P(X|Walks) = \frac{3}{10}$$

Now we can find the posterior probability using the Bayes theorem,

$$P(Walks|X) = \frac{\frac{3}{10} * \frac{10}{30}}{\frac{4}{30}} = 0.75$$

Step 2: Similarly we can find the posterior probability of Driving, and it is 0.25
Step 3: Compare both posterior probabilities. When comparing the posterior probability, we can find that P(walks|X) has greater values and the new point belongs to the walking category.

Implementation of Naive Bayes

Now let's implement Naive Bayes using python

We are using the Social network ad dataset. The dataset contains the details of users in a social networking site to find whether a user buys a product by clicking the ad on the site based on their salary, age, and gender.

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0

Let's start the programming by importing essential libraries required

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
```

Importing of dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
```

Since our dataset containing character variables we have to encode it using LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
```

Train test Split

We are performing a train test split on our dataset. We are providing the test size as 0.20, that means our training sample contains 320 training set and test sample contains 80 test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Feature scaling

Next, we are doing **feature scaling** to the training and test set of independent variables

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

Training the Naive Bayes model on the training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
Let's predict the test results
y_pred = classifier.predict(X_test)
Predicted and actual value –
y_pred

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0

y_test

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0

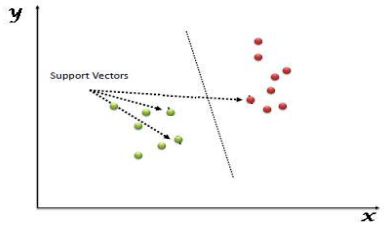
For the first 8 values, both are the same. We can evaluate our matrix using the confusion matrix and accuracy score by comparing the predicted and actual test values
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
confusion matrix –
ac – **0.9125**

	0	1
0	55	3
1	4	18

Accuracy is good. Note that, you can achieve better results for this problem using different algorithms.

3) Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).

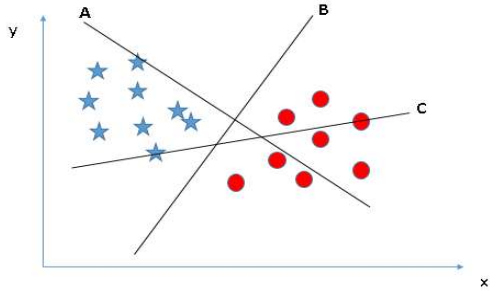


Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

You can look at support vector machines and a few examples of their working here.

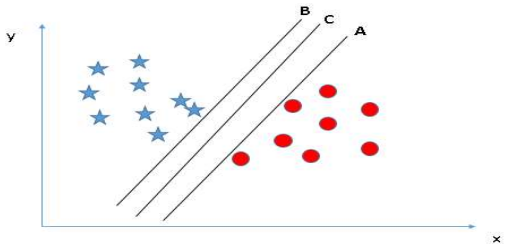
Working:

- **Identify the right hyper-plane (Scenario-1):** Here, we have three hyper-planes (A, B, and C). Now, identify the right hyper-plane to classify stars and circles.

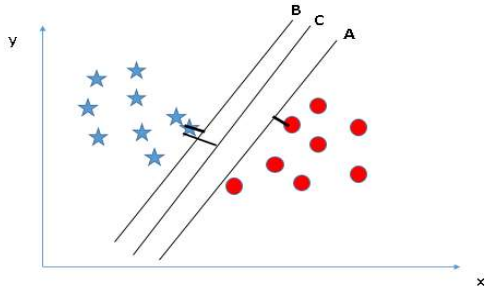


You need to remember a thumb rule to identify the right hyper-plane: "Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.

- **Identify the right hyper-plane (Scenario-2):** Here, we have three hyper-planes (A, B, and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

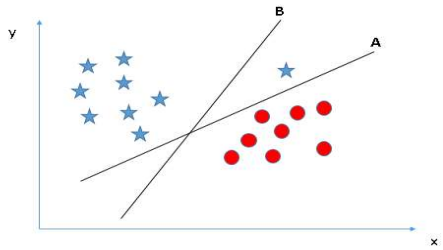


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Let's look at the below



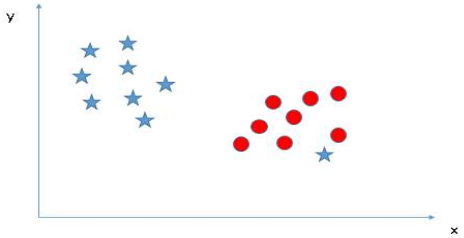
Above, you can see that the margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss-classification.

- **Identify the right hyper-plane (Scenario-3):**Hint: Use the rules as discussed in previous section to identify the right hyper-plane



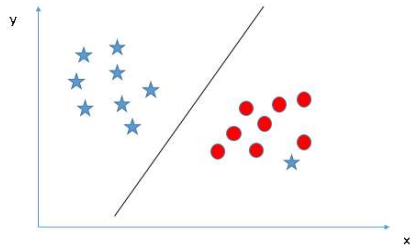
Some of you may have selected the hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin. Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

- **Can we classify two classes (Scenario-4)?:** Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other(circle) class as an outlier.

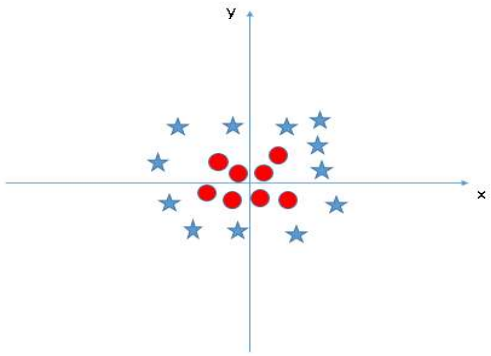


- one star at other end is like an outlier for star class. The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin. Hence, we can say, SVM

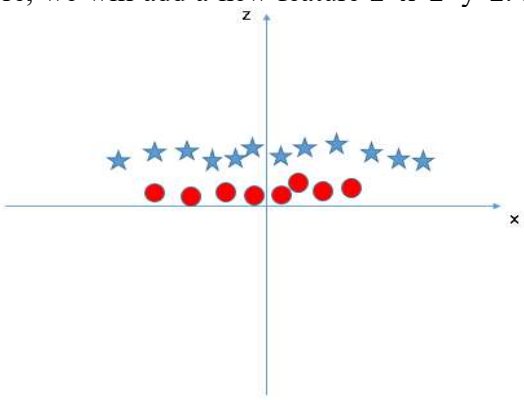
classification is robust to outliers.



- **Find the hyper-plane to segregate to classes (Scenario-5):** In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



- SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:

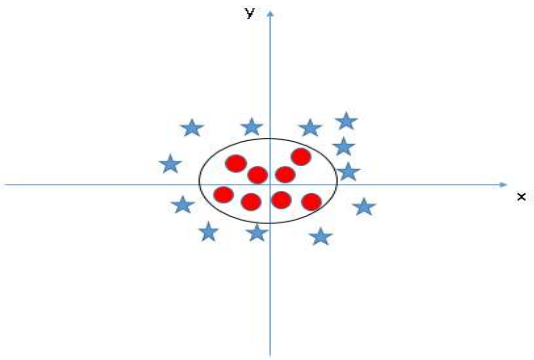


In above plot, points to consider are:

- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.

In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But, another burning question which arises is, should we need to add this feature manually to have a hyper-plane. No, the SVM algorithm has a technique called the **kernel trick**. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:

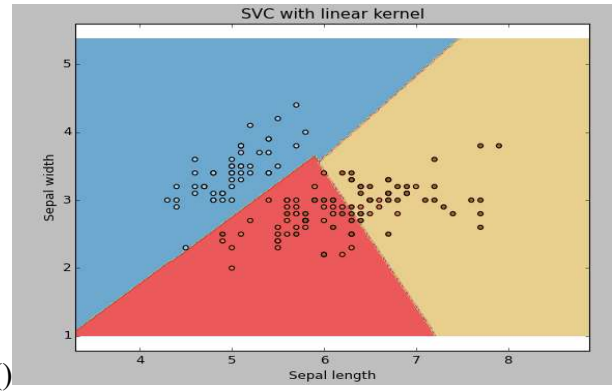


Now, let's look at the methods to apply SVM classifier algorithm in a data science challenge. You can also learn about the working of Support Vector Machine in video format from this Machine Learning Certification.

Support Vector Machine(SVM) code in Python

Example: Have a linear SVM kernel

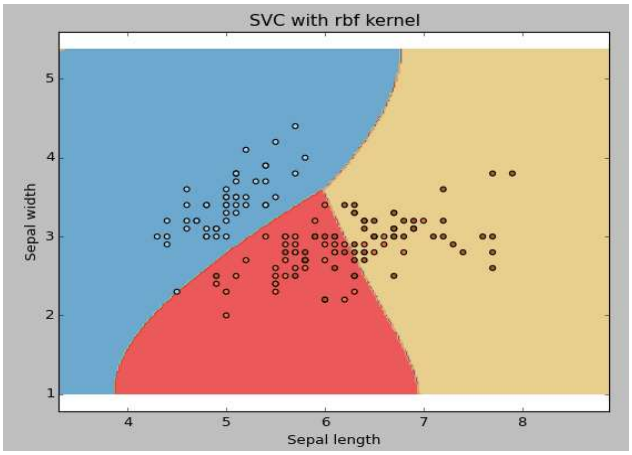
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma=0).fit(X, y)
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
```



plt.show()

Example: Use SVM rbf kernel
Change the kernel type to rbf in below line and look at the impact.

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

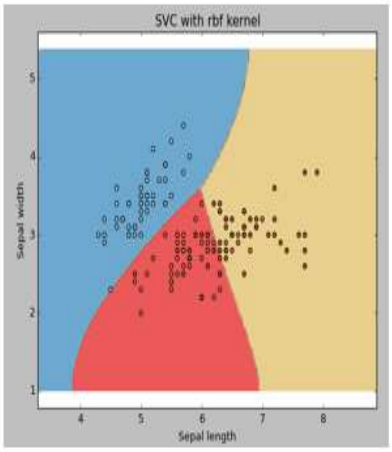


For linear SVM kernel if you have a large number of features (>1000) because it is more likely that the data is linearly separable in high dimensional space. Also, you can use RBF but do not forget to cross-validate for its parameters to avoid over-fitting.

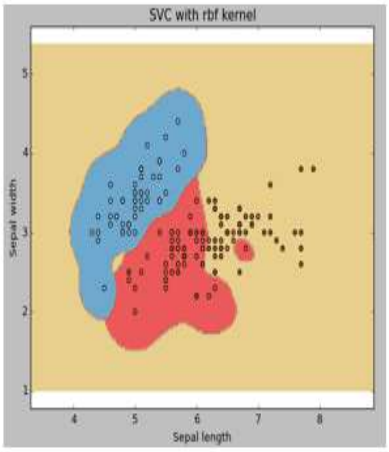
gamma: Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. Higher the value of gamma, will try to exact fit the as per training data set i.e. generalization error and cause over-fitting problem.
Example: Let's difference if we have gamma different gamma values like 0, 10 or 100.

svc = svm.SVC(kernel='rbf', C=1,gamma=0).fit(X, y)

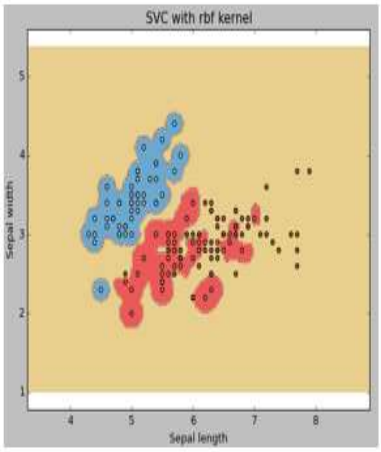
gamma =0



gamma =10

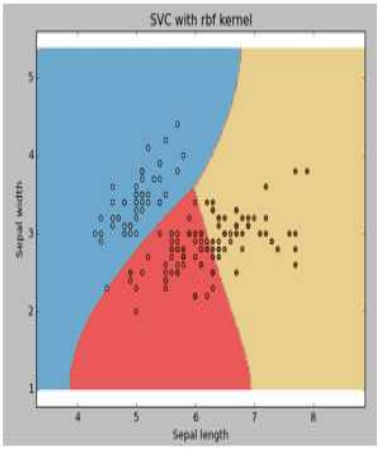


gamma =100

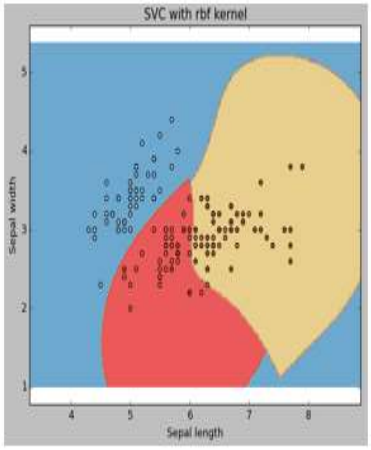


C: Penalty parameter C of the error term. It also controls the trade-off between smooth decision boundaries and classifying the training points correctly.

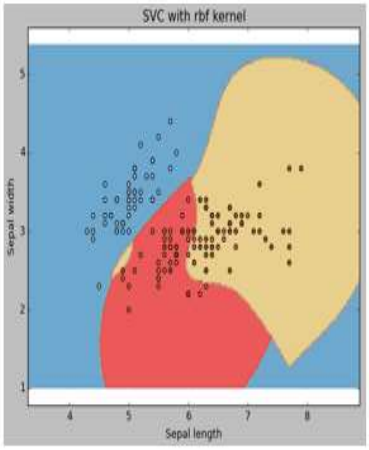
c =1



C =100



c =1000



We should always look at the cross-validation score to have effective combination of these parameters and avoid over-fitting.

Set A

1) Write a Python program build Decision Tree Classifier using Scikit-learn package for diabetes data set (download database from https://www.kaggle.com/uciml/pima-indians-diabetes-database)
2) Write a Python program build Decision Tree Classifier for shows.csv from pandas and predict class label for show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7? Create a csv file as shown in https://www.w3schools.com/python/python_ml_decision_tree.asp

Set B

1) Consider following dataset
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']. Use Naïve Bayes algorithm to predict[0:Overcast, 2:Mild] tuple belongs to which class whether to play the sports or not.

Set C :

1) Write a Python program to build SVM model to Cancer dataset. The dataset is available in the scikit-learn library. Check the accuracy of model with precision and recall.

Signature of the Instructor: **Date:**

Assignment Evaluation

0: Not Done	2: Late Complete	4: Complete
1: Complete	3. Needs Improvement	5: Well Done

Assignment 4: Association Rules

Objectives:

Students will be able to

- Learn about Apriori algorithm and association rule mining
- Learn support, confidence and lift
- Understand how to apply Apriori algorithm on data set.

Prerequisites:

- ✓ Basic knowledge of Python Programming
- ✓ https://thecleverprogrammer.com/2020/11/16/apriori-algorithm-using-python/
- ✓ https://towardsdatascience.com
- ✓ https://www.ncbi.nlm.nih.gov/

Introduction:

ARM (Associate Rule Mining) is one of the important techniques in data science. In ARM, the frequency of patterns and associations in the dataset is identified among the item sets then used to predict the next relevant item in the set. This ARM technique is mostly used in business decisions according to customer purchases.

- Example: In Walmart, if Sonali buys Milk and Bread, the chances of him buying Butter are predicted by the Associate Rule Mining technique.
- In this module, We are building an apriori algorithm from jupyter notebook on the given dataset and used it as a recommendation system for customer purchases.

Example:

Suppose we have a record of 1 thousand customer transactions, and we want to find the Support, Confidence, and Lift for two items e.g. burgers and ketchup. Out of one thousand transactions, 100 contain ketchup while 150 contain a burger. Out of 150 transactions where a burger is purchased, 50 transactions contain ketchup as well. Using this data, we want to find the support, confidence, and lift.

1) Support

Support refers to the default popularity of an item and can be calculated by finding number of transactions containing a particular item divided by total number of transactions. Suppose we want to find support for item B. This can be calculated as:

- Support(B) = (Transactions containing (B))/(Total Transactions)

For instance if out of 1000 transactions, 100 transactions contain Ketchup then the support for item Ketchup can be calculated as:

- Support(Ketchup) = (Transactions containing Ketchup)/(Total Transactions)

Support(Ketchup) = 100/1000

= 10%

2) Confidence

Confidence refers to the likelihood that an item B is also bought if item A is bought. It can be calculated by finding the number of transactions where A and B are bought together, divided by total number of transactions where A is bought. Mathematically, it can be represented as:

- Confidence(A→B) = (Transactions containing both (A and B))/(Transactions containing A)

Coming back to our problem, we had 50 transactions where Burger and Ketchup were bought together. While in 150 transactions, burgers are bought. Then we can find likelihood of buying ketchup when a burger is bought can be represented as confidence of Burger -> Ketchup and can be mathematically written as:

- Confidence(Burger→Ketchup) = (Transactions containing both (Burger and Ketchup))/(Transactions containing A)

  Confidence(Burger→Ketchup) = 50/150

  = 33.3%

3) Lift

Lift(A -> B) refers to the increase in the ratio of sale of B when A is sold. Lift(A –> B) can be calculated by dividing Confidence(A -> B) divided by Support(B). Mathematically it can be represented as:

- Lift(A→B) = (Confidence (A→B))/(Support (B))

Coming back to our Burger and Ketchup problem, the Lift(Burger -> Ketchup) can be calculated as:

- Lift(Burger→Ketchup) = (Confidence (Burger→Ketchup))/(Support (Ketchup))

  Lift(Burger→Ketchup) = 33.3/10

  = 3.33

Lift basically tells us that the likelihood of buying a Burger and Ketchup together is 3.33 times more than the likelihood of just buying the ketchup. A Lift of 1 means there is no association between products A and B. Lift of greater than 1 means products A and B are more likely to be bought together. Finally, Lift of less than 1 refers to the case where two products are unlikely to be bought together.

APRIORI Algorithm

- Apriori is an algorithm used for Association Rule Mining. It searches for a series of frequent sets of items in the datasets. It builds on associations and correlations between the itemsets. It is the algorithm behind “You may also like” where you commonly saw in recommendation platforms.

- Apriori Algorithm is a Machine Learning algorithm that is used to gain insight into the structured relationships between different items involved. It's a data mining technique that is used for mining frequent itemsets and relevant association rules.
- Example: Recommending products based on your purchased items. You can see this in different e-commerce websites. (Recommendation system).

Steps Involved in Apriori Algorithm

For large sets of data, there can be hundreds of items in hundreds of thousands transactions. The Apriori algorithm tries to extract rules for each possible combination of items. For instance, Lift can be calculated for item 1 and item 2, item 1 and item 3, item 1 and item 4 and then item 2 and item 3, item 2 and item 4 and then combinations of items e.g. item 1, item 2 and item 3; similarly, item 1, item2, and item 4, and so on.

As you can see from the above example, this process can be extremely slow due to the number of combinations. To speed up the process, we need to perform the following steps:

1) Set a minimum value for support and confidence. This means that we are only interested in finding rules for the items that have certain default existence (e.g. support) and have a minimum value for co-occurrence with other items (e.g. confidence).

2) Extract all the subsets having higher value of support than minimum threshold.

3) Select all the rules from the subsets with confidence value higher than minimum threshold.

4) Order the rules by descending order of Lift.

Implementing Apriori Algorithm with Python

In this section we will use the Apriori algorithm to find rules that describe associations between different products given 7500 transactions over the course of a week at a French retail store. The dataset can be downloaded from the following link:

https://drive.google.com/file/d/1am9V3mFEnbuen7EanzwLoRKO3qFXZ5By/view?usp=sharing

Datasets are freely available on (https://www.kaggle.com/) this site.

We do not need to write the script to calculate support, confidence, and lift for all the possible combination of items. We will use an off-the-shelf library where all of the code has already been implemented.

The apyori library download and install the library in the default path for your Python libraries before proceeding.

Steps to implement Apriori algorithm in Python:

1) Import the Libraries

```
In [5]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from apyori import apriori
```

In the script above we import pandas, numpy, pyplot, and apriori libraries.

2) Import the Dataset

Now let's import the dataset and see what we are working with. Download the dataset and place it in the "Datasets" folder of the "E" drive (or change the code below to match the path of the file on your computer) and execute the following script:

```
In [6]: store_data = pd.read_csv('store_data.csv')
```

Let's call the head() function to see how the dataset looks:

```
In [7]: store_data.head()
```

A snippet of the dataset is shown in the below screenshot. If you carefully look at the data, we can see that the header is actually the first transaction. Each row corresponds to a transaction and each column corresponds to an item purchased in that specific transaction. The NaN tells us that the item represented by the column was not purchased in that specific transaction.

Out[7]:

	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	froze smooth
0	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

In this dataset there is no header row. But by default, pd.read_csv function treats first row as header. To get rid of this problem, add header=None option to pd.read_csv function, as shown below:

```
In [8]: store_data = pd.read_csv('store_data.csv',header=None)
```

Now execute the head() function:

```
In [9]: store_data.head()
```

In this updated output you will see that the first line is now treated as a record instead of header as shown below:

Out[9]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon	antioxydant juice	frozen smoothie	spina
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Now we will use the Apriori algorithm to find out which items are commonly sold together, so that store owner can act to place the related items together or advertise them together in order to have increased profit.

3) Data Pre-processing

The Apriori library we are going to use requires our dataset to be in the form of a list of lists, where the whole dataset is a big list and each transaction in the dataset is an inner list within the outer big list. Currently we have data in the form of a pandas dataframe. To convert our pandas dataframe into a list of lists, execute the following script:

```
In [10]: records = []
         for i in range(0, 7501):
             records.append([str(store_data.values[i,j]) for j in range(0, 20)])
```

4) Applying Apriori

The next step is to apply the Apriori algorithm on the dataset. To do so, we can use the apriori class that we imported from the apyori library. The apriori class requires some parameter values to work. The first parameter is the list of list that you want to extract rules from. The second parameter is the min_support parameter.

This parameter is used to select the items with support values greater than the value specified by the parameter. Next, the min_confidence parameter filters those rules that have confidence greater than the confidence threshold specified by the parameter. Similarly, the min_lift parameter specifies the minimum lift value for the short listed rules. Finally, the min_length parameter specifies the minimum number of items that you want in your rules.

Let's suppose that we want rules for only those items that are purchased at least 5 times a day, or 7 x 5 = 35 times in one week, since our dataset is for a one-week time period. The support for those items can be calculated as 35/7500 = 0.0045. The minimum confidence for the rules is 20% or 0.2. Similarly, we specify the value for lift as 3 and finally min_length is 2 since we want at least two products in our rules. These values are mostly just arbitrarily chosen, so you can play with these values and see what difference it makes in the rules you get back out.

Execute the following script:

```
In [11]: association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)
         association_results = list(association_rules)
```

In the second line here we convert the rules found by the apriori class into a list since it is easier to view the results in this form.

5) Viewing the Results

Let's first find the total number of rules mined by the apriori class. Execute the following script:

```
In [12]: print(len(association_results))
         48
```

The script above should return 48. Each item corresponds to one rule.

Let's print the first item in the association_rules list to see the first rule. Execute the following script:

```
In [13]: print(association_results[0])
```

The below output should look like this:

RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004532728969470737, ordered_statistics=[OrderedStatistic(items_base=frozenset({'light cream'}), items_add=frozenset({'chicken'}), confidence=0.29059829059829057, lift=4.84395061728395)])

The first item in the list is a list itself containing three items. The first item of the list shows the grocery items in the rule.

For instance from the first item, we can see that light cream and chicken are commonly bought together. This makes sense since people who purchase light cream are careful about what they eat hence they are more likely to buy chicken i.e. white meat instead of red meat i.e. beef. Or this could mean that light cream is commonly used in recipes for chicken.

The support value for the first rule is 0.0045. This number is calculated by dividing the number of transactions containing light cream divided by total number of transactions. The confidence level for the rule is 0.2905 which shows that out of all the transactions that contain light cream, 29.05% of the transactions also contain chicken. Finally, the lift of 4.84 tells us that chicken is 4.84 times more likely to be bought by the customers who buy light cream compared to the default likelihood of the sale of chicken.

- The following script displays the rule, the support, the confidence, and lift for each rule in a more clear way:

If you execute the below script, you will see all the rules returned by the apriori class. The first four rules returned by the apriori class look like this:

```
In [16]: for item in association_results:

             # first index of the inner list
             # Contains base item and add item
             pair = item[0]
             items = [x for x in pair]
             print("Rule: " + items[0] + " -> " + items[1])

             #second index of the inner list
             print("Support: " + str(item[1]))

             #third index of the list located at 0th
             #of the third index of the inner list

             print("Confidence: " + str(item[2][0][2]))
             print("Lift: " + str(item[2][0][3]))
             print("====================================")
```

Output:

```
Rule: escalope -> mushroom cream sauce
Support: 0.005732568990801226
Confidence: 0.3006993006993007
Lift: 3.790832696715049
====================================
Rule: escalope -> pasta
Support: 0.005865884548726837
Confidence: 0.3728813559322034
Lift: 4.700811850163794
====================================
Rule: herb & pepper -> ground beef
Support: 0.015997866951073192
Confidence: 0.3234501347708895
Lift: 3.2919938411349285
```

We have already discussed the first rule. Let's now discuss the second rule. The second rule states that mushroom cream sauce and escalope are bought frequently. The support for mushroom cream sauce is 0.0057. The confidence for this rule is 0.3006 which means that out of all the transactions containing mushroom, 30.06% of the transactions are likely to contain escalope as well. Finally, lift of 3.79 shows that the escalope is 3.79 more likely to be bought by the customers that buy mushroom cream sauce, compared to its default sale.

Set A

1) Write a Python Programme to read the dataset ("Iris.csv"). dataset download from (https://archive.ics.uci.edu/ml/datasets/iris) and apply Apriori algorithm.
2) Write a Python Programme to apply Apriori algorithm on Groceries dataset. Dataset can be downloaded from (https://github.com/amankharwal/Website-data/blob/master/Groceries_dataset.csv)
   Also display support and confidence for each rule.

Set B

1) Write a Python program to read "StudentsPerformance.csv" file. Solve following:
   - To display the shape of dataset.
   - To display the top rows of the dataset with their columns.
   - To display the number of rows randomly.
   - To display the number of columns and names of the columns.
   Note: Download dataset from following link : (https://www.kaggle.com/spscientist/students-performance-in-exams?select=StudentsPerformance.csv)

2) Write a Python program for Apriori Algorithm using ARM. And Print the Rule, Support, Confidence and Lift.
   - (Set Min_Support = 0.0040, Min_Confidence=0.2, Min_Lift=3, Min_Length=2)

   Note: Download dataset from following link : (https://www.kaggle.com/irfanasrullah/groceries)

Set C

1) Write a Python Program to implement Apriori algorithm for following data set. Create csv file in excel. Apply Apriori algorithm and print the association results.

Wine	Chips	Bread	Butter	Milk	Apple
Wine	Chips		Butter	Milk	
		Bread	Butter	Milk	Apple
		Bread		Milk	Apple
Wine	Chips	Bread		Milk	
Wine	Chips		Butter	Milk	Apple
Wine	Chips		Butter		Apple
Wine	Chips	Bread	Butter	Milk	Apple
Wine	Chips	Bread	Butter	Milk	Apple
Wine	Chips		Butter	Milk	
			Butter		Apple
Wine		Bread	Butter	Milk	Apple
Wine	Chips	Bread	Butter	Milk	Apple
Wine	Chips	Bread		Milk	Apple

	Chips	Bread		Milk	Apple
		Bread	Butter	Milk	
Wine		Bread	Butter	Milk	Apple
Wine	Chips	Bread	Butter		Apple
Wine	Chips	Bread	Butter		
Wine	Chips	Bread	Butter	Milk	Apple
Wine	Chips	Bread	Butter	Milk	Apple
	Chips	Bread	Butter	Milk	Apple

Signature of the Instructor: **Date:**

Assignment Evaluation

0: Not Done	2: Late Complete	4: Complete
1: Complete	3. Needs Improvement	5: Well Done

Assignment 5: Regression Analysis and outlier detection

Objectives:

Students will be able to

- Understand regression and its types
- Understand what linear regression is used for
- Learn working of regression and its types
- How to implement regression in Python, step by step
- Understand what are outliers and its types
- Methods used for detection of outliers.

Prerequisites:

- ✓ https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- ✓ https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/introduction-to-trend-lines/a/linear-regression-review
- ✓ Machine Learning Basics: Simple Linear Regression | by Gurucharan M K | Towards Data Science

Introduction:

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).

Regression can also help predict sales for a company based on weather, previous sales, GDP growth, or other types of conditions.

Regression searches for relationships among variables.

For example, you can observe several employees of some company and try to understand how their salaries depend on the features, such as experience, level of education, role, city they work in, and so on.

This is a regression problem where data related to each employee represent one observation. The presumption is that the experience, education, role, and city are the independent features, while the salary depends on them.

Similarly, you can try to establish a mathematical dependence of the prices of houses on their areas, numbers of bedrooms, distances to the city center, and so on.

Generally, in regression analysis, you usually consider some phenomenon of interest and have a number of observations. Each observation has two or more features. Following the assumption that (at least) one of the features depends on the others, you try to establish a relation among them.

User has to find a function that maps some features or variables to others sufficiently well.

The dependent features are called the dependent variables, outputs, or responses.

The independent features are called the independent variables, inputs, or predictors.

Regression problems usually have one continuous and unbounded dependent variable. The inputs, however, can be continuous, discrete, or even categorical data such as gender, nationality, brand, and so on.

It is a common practice to denote the outputs with y and inputs with x. If there are two or more independent variables, they can be represented as the vector $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of inputs.

Need of Regression:

1) Regression model is used to answer whether and how some phenomenon influences the other or how several variables are related. For example, you can use it to determine *if* and *to what extent* the experience or gender impact salaries.
2) Regression is also useful to forecast a response using a new set of predictors. For example, you could try to predict electricity consumption of a household for the next hour given the outdoor temperature, time of day, and number of residents in that household.

Regression is used in many different fields: economy, computer science, social sciences, and so on. Its importance rises every day with the availability of large amounts of data and increased awareness of the practical value of data.

Two types of regression

1) Simple linear regression : This regression uses one independent variable to explain or predict the outcome of the dependent variable Y
2) Multiple linear regression : This regression uses two or more independent variables to predict the outcome.

The general form of each type of regression is:

- **Simple linear regression:** $Y = a + bX + u$
- **Multiple linear regression:** $Y = a + b_1X_1 + b_2X_2 + b_3X_3 + ... + b_tX_t + u$

Where:

- Y = the variable that you are trying to predict (dependent variable).
- X = the variable that you are using to predict Y (independent variable).
- a = the intercept.
- b = the slope.
- u = the regression residual.

Linear Regression

Most widely used technique.

Problem Formulation

When implementing linear regression of some dependent variable y on the set of independent variables $\mathbf{x} = (x_1, \ldots, x_r)$, where r is the number of predictors, you assume a linear relationship between y and $\mathbf{x}$: $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_r x_r + \varepsilon$. This equation is the regression equation. β_0, β_1, …, β_r are the regression coefficients, and ε is the random error.

Linear regression calculates the estimators of the regression coefficients or simply the predicted weights, denoted with b_0, b_1, …, b_r. They define the estimated regression function $f(\mathbf{x}) = b_0 + b_1 x_1 + \cdots + b_r x_r$. This function should capture the dependencies between the inputs and output sufficiently well.

The estimated or predicted response, $f(\mathbf{x}_i)$, for each observation $i = 1, \ldots, n$, should be as close as possible to the corresponding actual response y_i. The differences $y_i - f(\mathbf{x}_i)$ for all observations $i = 1, \ldots, n$, are called the residuals. Regression is about determining the best predicted weights, that is the weights corresponding to the smallest residuals.

To get the best weights, you usually minimize the sum of squared residuals (SSR) for all observations $i = 1, \ldots, n$: $\text{SSR} = \Sigma_i (y_i - f(\mathbf{x}_i))^2$. This approach is called the method of ordinary least squares.

Regression Performance

The variation of actual responses y_i, $i = 1, \ldots, n$, occurs partly due to the dependence on the predictors $\mathbf{x}_i$. However, there is also an additional inherent variance of the output.

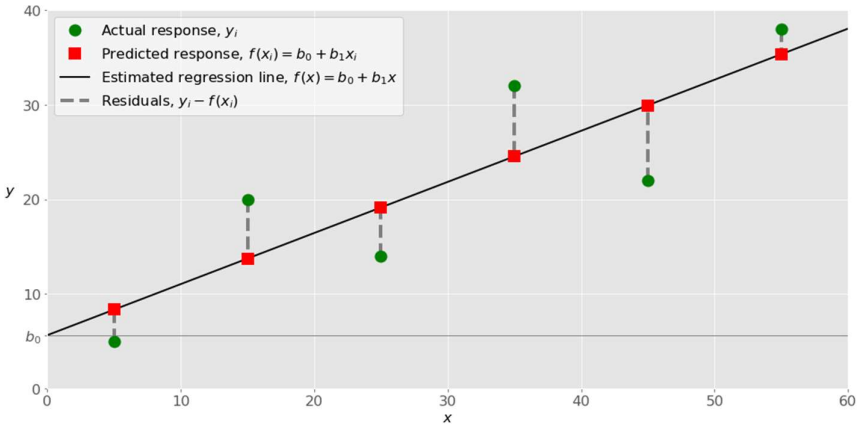
The coefficient of determination, denoted as R^2, tells you which amount of variation in y can be explained by the dependence on $\mathbf{x}$ using the particular regression model. Larger R^2 indicates a better fit and means that the model can better explain the variation of the output with different inputs.

The value $R^2 = 1$ corresponds to SSR = 0, that is to the perfect fit since the values of predicted and actual responses fit completely to each other.

Simple Linear Regression

Simple or single-variate linear regression is the simplest case of linear regression with a single independent variable, $\mathbf{x} = x$.

The following figure illustrates simple linear regression:



Example of Simple Linear Regression

When implementing simple linear regression, you typically start with a given set of input-output (x-y) pairs (green circles). These pairs are your observations. For example, the leftmost observation (green circle) has the input $x = 5$ and the actual output (response) $y = 5$. The next one has $x = 15$ and $y = 20$, and so on.

The estimated regression function (black line) has the equation $f(x) = b_0 + b_1x$. Your goal is to calculate the optimal values of the predicted weights b_0 and b_1 that minimize SSR and determine the estimated regression function. The value of b_0, also called the intercept, shows the point where the estimated regression line crosses the y axis. It is the value of the estimated response $f(x)$ for $x = 0$. The value of b_1 determines the slope of the estimated regression line.

The predicted responses (red squares) are the points on the regression line that correspond to the input values. For example, for the input $x = 5$, the predicted response is $f(5) = 8.33$ (represented with the leftmost red square).

The residuals (vertical dashed gray lines) can be calculated as $y_i - f(\mathbf{x}_i) = y_i - b_0 - b_1x_i$ for $i = 1, \ldots, n$. They are the distances between the green circles and red squares. When you implement linear regression, you are actually trying to minimize these distances and make the red squares as close to the predefined green circles as possible.

Implementing Linear Regression in Python

Python Packages for Linear Regression

1) Scipy Package
2) Numpy
3) Scikit-learn package

The package **NumPy** is a fundamental Python scientific package that allows many high-performance operations on single- and multi-dimensional arrays. It also offers many mathematical routines. it's open source package.

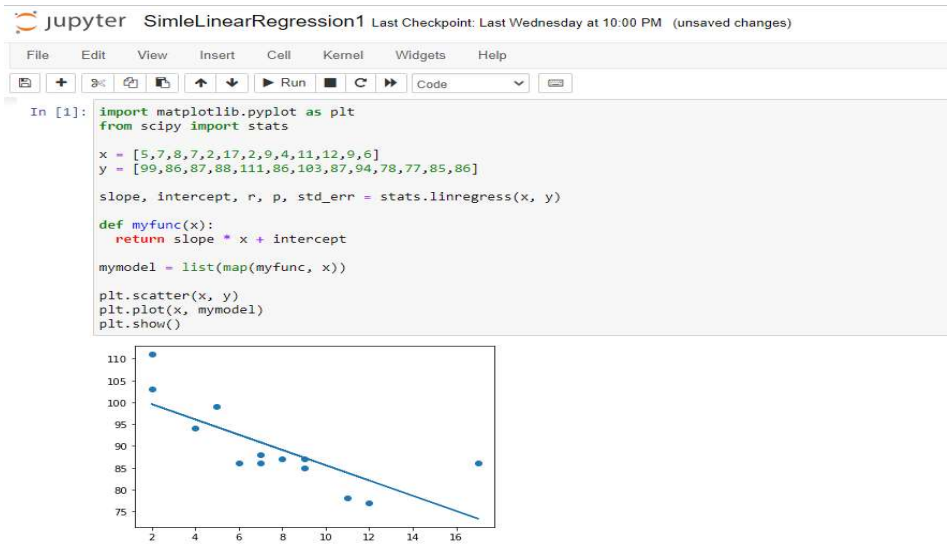
The package scikit-learn is a widely used Python library for machine learning, built on top of NumPy and some other packages. It provides the means for preprocessing data, reducing dimensionality, implementing regression, classification, clustering, and more. Like NumPy, scikit-learn is also open source.

Simple Linear Regression with Scipy package

Steps:

1) Import the required modules
   import matplotlib.pyplot as plt
   from scipy import stats
2) Create the arrays that represent the values of the x and y axis:
   x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
   y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
3) Execute a method that returns some important key values of Linear Regression:
   slope, intercept, r, p, std_err = stats.linregress(x, y)
4) Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed.
   def myfunc(x):
     return slope * x + intercept
5) Run each value of the x array through the function. This will result in a new array with new values for the y-axis
   mymodel = list(map(myfunc, x))
6) Draw the original scatter plot:
   plt.scatter(x, y)
7) Draw the line of linear regression:
   plt.plot(x, mymodel)
8) Display the Diagram
   plt.show()

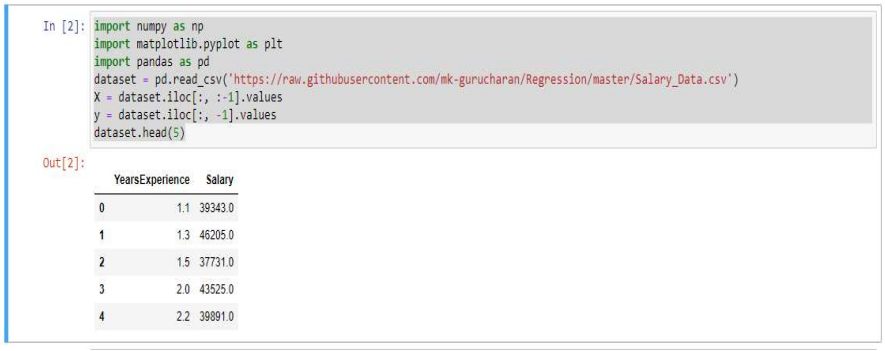
following is the complete programme with output



Simple Linear Regression with numpy package

Steps

1) Importing the libraries
   import the ***pandas*** library that will be used to store the data in a Pandas DataFrame. The ***matplotlib*** is used to plot graphs.
   import numpy as np
   import matplotlib.pyplot as plt
   import pandas as pd
2) Importing the dataset
   download the dataset from my github repositary which contains the data as "Salary_Data.csv". The variable *X* will store the "*Years of Experience*" and the variable *Y* will store the "*Salary*". The dataset.head(5)is used to visualize the first 5 rows of the data.
   import numpy as np
   import matplotlib.pyplot as plt
   import pandas as pd
   dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Regression/master/Salary_Data.csv')
   X = dataset.iloc[:, :-1].values
   y = dataset.iloc[:, -1].values
   dataset.head(5)



```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        dataset = pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Regression/master/Salary_Data.csv')
        X = dataset.iloc[:, :-1].values
        y = dataset.iloc[:, -1].values
        dataset.head(5)

Out[2]:
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

3) Splitting the dataset into the Training set and Test set
   split the dataset into the Training set, on which the Linear Regression model will be trained and the Test set, on which the trained model will be applied to visualize the results. In this the test_size=0.2 denotes that *20%* of the data will be kept as the *Test set* and the remaining *80%* will be used for training as the *Training set*.
   from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

4) Training the Simple Linear Regression model on the Training set
   the class LinearRegression is imported and is assigned to the variable *"regressor"*. The regressor.fit() function is fitted with **X_train** and *Y_train* on which the model will be trained.

   from sklearn.linear_model import LinearRegression
   regressor = LinearRegression()
   regressor.fit(X_train, y_train)

5) Predicting the Test set results
the regressor.predict() function is used to predict the values for the Test set and the values are stored to the variable y_pred.
y_pred = regressor.predict(X_test)
6) Comparing the Test Set with Predicted Values
a Pandas DataFrame is created to compare the Salary values of both the original Test set (*y_test*) and the predicted results (*y_pred*).
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
```

	Real Values	Predicted Values
0	63218.0	62638.405203
1	46205.0	37548.649766
2	43525.0	44303.583922
3	61111.0	68428.348766
4	112635.0	117642.869046
5	67938.0	72288.311141

In this step the predicted salaries are very close to the real salary values and it can be concluded that the model has been well trained.
7) Visualizing the results
we visualize the results of the *Real* and the *Predicted* Salary values along with the Linear Regression Line on a graph that is plotted.

```
plt.scatter(X_test, y_test, color = 'red')
plt.scatter(X_test, y_pred, color = 'green')
plt.plot(X_train, regressor.predict(X_train), color = 'black')
plt.title('Salary vs Experience (Result)')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.show()
```

```
plt.scatter(X_test, y_test, color = 'red')
plt.scatter(X_test, y_pred, color = 'green')
plt.plot(X_train, regressor.predict(X_train), color = 'black')
plt.title('Salary vs Experience (Result)')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.show()
```

In the above graph In this graph, the Real values are plotted in "*Red*" color and the Predicted values are plotted in "*Green*" color. The Linear Regression line that is generated is drawn in "*Black*" color.
So we have successfully build a ***Simple Linear Regression*** model that predicts the 'Salary' of an employee based on their 'Years of Experience' and visualize the results.

Simple Linear Regression With scikit-learn Package

There are five basic steps for implementing linear regression:

1) Import the packages and classes you need.
   import the package numpy and the class LinearRegression from sklearn.linear_model:

   import numpy as np

   from sklearn.linear_model import LinearRegression

   The fundamental data type of NumPy is the array type called numpy.ndarray. The rest of this article uses the term **array** to refer to instances of the type numpy.ndarray.

   The class sklearn.linear_model.LinearRegression will be used to perform linear and polynomial regression and make predictions accordingly.

2) Provide data to work with and eventually do appropriate transformations.
   The inputs (regressors, x) and output (predictor, y) should be arrays (the instances of the class numpy.ndarray) or similar objects. This is the simplest way of providing data for regression:

   x= np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

   y = np.array([5, 20, 14, 32, 22, 38])

   Now, you have two arrays: the input x and output y. You should call .reshape() on x because this array is required to be **two-dimensional**, or to be more precise, to have **one column and as many rows as necessary**. That's exactly what the argument (-1, 1) of .reshape() specifies.

   This is how x and y look now:

```
print(x)
```

```
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
```

```
print (y)
```

```
[ 5 20 14 32 22 38]
```

x has two dimensions, and x.shape is (6, 1), while y has a single dimension, and y.shape is (6,).

3) Create a regression model and fit it with existing data.
create an instance of the class LinearRegression, which will represent the regression model:

This statement creates the variable model as the instance of LinearRegression.

model = LinearRegression()

Several optional parameters to LinearRegression are as follows:

- **fit_intercept** is a Boolean (True by default) that decides whether to calculate the intercept b_0 (True) or consider it equal to zero (False).
- **normalize** is a Boolean (False by default) that decides whether to normalize the input variables (True) or not (False).
- **copy_X** is a Boolean (True by default) that decides whether to copy (True) or overwrite the input variables (False).
- **n_jobs** is an integer or None (default) and represents the number of jobs used in parallel computation. None usually means one job and -1 to use all processors.

This example uses the default values of all parameters. It's time to start using the model. First, you need to call .fit() on model:

model.fit(x, y)

With .fit(), you calculate the optimal values of the weights b_0 and b_1, using the existing input and output (x and y) as the arguments. In other words, .fit() **fits the model**. It returns self, which is the variable model itself. That's why you can replace the last two statements with this one:

model = LinearRegression().fit(x, y)

4) Check the results of model fitting to know whether the model is satisfactory.
Once you have your model fitted, you can get the results to check whether the model works satisfactorily and interpret it.You can obtain the coefficient of determination (R^2) with .score() called on model:

r_sq = model.score(x, y)

print('coefficient of determination:', r_sq)

```
r_sq = model.score(x, y)
```

```
print('coefficient of determination:', r_sq)
```

```
coefficient of determination: 0.715875613747954
```

When you're applying .score(), the arguments are also the predictor x and regressor y, and the return value is R^2.

The attributes of model are .intercept_, which represents the coefficient, b_0 and .coef_, which represents b_1:

```
print('intercept:', model.intercept_)
```

```
intercept: 5.633333333333329
```

```
print('slope:', model.coef_)
```

```
slope: [0.54]
```

The code above illustrates how to get b_0 and b_1. You can notice that .intercept_ is a scalar, while .coef_ is an array.

The value $b_0 = 5.63$ (approximately) illustrates that your model predicts the response 5.63 when x is zero. The value $b_1 = 0.54$ means that the predicted response rises by 0.54 when x is increased by one.

You should notice that you can provide y as a two-dimensional array as well. In this case, you'll get a similar result. This is how it might look:

```
new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
print('intercept:', new_model.intercept_)
print('slope:', new_model.coef_)
```

```
intercept: [5.63333333]
slope: [[0.54]]
```

As you can see, this example is very similar to the previous one, but in this case, .intercept_ is a one-dimensional array with the single element b_0, and .coef_ is a two-dimensional array with the single element b_1.

5) Apply the model for predictions.

Once there is a satisfactory model, you can use it for predictions with either existing or new data. To obtain the predicted response, use .predict():

```
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

When applying .predict(), you pass the regressor as the argument and get the corresponding predicted response.

This is a nearly identical way to predict the response:

```
y_pred = model.intercept_ + model.coef_ * x
print('predicted response:', y_pred, sep='\n')
```

```
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

In this case, you multiply each element of x with model.coef_ and add model.intercept_ to the product.

The output here differs from the previous example only in dimensions. The predicted response is now a two-dimensional array, while in the previous case, it had one dimension.

Multiple linear regression:

Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is $f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2$. It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights b_0, b_1, and b_2 such that this plane is as close as possible to the actual responses and yield the minimal SSR.

The case of more than two independent variables is similar, but more general. The estimated regression function is $f(x_1, \ldots, x_r) = b_0 + b_1x_1 + \cdots + b_rx_r$, and there are $r + 1$ weights to be determined when the number of inputs is r.

Multiple Linear Regression With scikit-learn

Multiple linear regression can be implemented following the same steps as you would for simple regression.

Steps 1 and 2: Import packages and classes, and provide data

First, you import numpy and sklearn.linear_model.LinearRegression and provide known inputs and output:

```
import numpy as np

from sklearn.linear_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]

y = [4, 5, 20, 14, 32, 22, 38, 43]

x, y = np.array(x), np.array(y)

Print x and y

print(x)

print(y)
```

In multiple linear regression, x is a two-dimensional array with at least two columns, while y is usually a one-dimensional array. This is a simple example of multiple linear regression, and x has exactly two columns.

3) create a model and fit it

The next step is to create the regression model as an instance of LinearRegression and fit it with .fit():

```
model = LinearRegression().fit(x, y)
```

The result of this statement is the variable model referring to the object of type LinearRegression. It represents the regression model fitted with existing data.

4) Get result

You can obtain the properties of the model the same way as in the case of simple linear regression:

```
r_sq = model.score(x, y)

print('coefficient of determination:', r_sq)

print('intercept:', model.intercept_)
```

```
print('slope:', model.coef_)
```

You obtain the value of R^2 using .score() and the values of the estimators of regression coefficients with .intercept_ and .coef_. Again, .intercept_ holds the bias b_0, while now .coef_ is an array containing b_1 and b_2 respectively.

In this example, the intercept is approximately 5.52, and this is the value of the predicted response when $x_1 = x_2 = 0$. The increase of x_1 by 1 yields the rise of the predicted response by 0.45. Similarly, when x_2 grows by 1, the response rises by 0.26.

5) predict response

Predictions also work the same way as in the case of simple linear regression:

```
y_pred = model.predict(x)

print('predicted response:', y_pred, sep='\n')
```

The predicted response is obtained with .predict(), which is very similar to the following:

```
y_pred = model.intercept_ + np.sum(model.coef_ * x, axis=1)

print('predicted response:', y_pred, sep='\n')

x_new = np.arange(10).reshape((-1, 2))
```

You can predict the output values by multiplying each column of the input with the appropriate weight, summing the results and adding the intercept to the sum.

You can apply this model to new data as well:

```
print(x_new)

y_new = model.predict(x_new)

print(y_new)
```

The final code will be:

```
import numpy as np
from sklearn.linear_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)
print(x)
model = LinearRegression().fit(x, y)
r_sq = model.score(x, y)
print('coefficient of determination:', r_sq)

print('intercept:', model.intercept_)

print('slope:', model.coef_)
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')

y_pred = model.intercept_ + np.sum(model.coef_ * x, axis=1)
print('predicted response:', y_pred, sep='\n')
x_new = np.arange(10).reshape((-1, 2))

print(x_new)
y_new = model.predict(x_new)
print(y_new)
```

Output

```
[[ 0  1]
 [ 5  1]
 [15  2]
 [25  5]
 [35 11]
 [45 15]
 [55 34]
 [60 35]]
coefficient of determination: 0.8615939258756775
intercept: 5.52257927519819
slope: [0.44706965 0.25502548]
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
predicted response:
[ 5.77760476  8.012953   12.73867497 17.9744479  23.97529728 29.4660957
 38.78227633 41.27265006]
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
[ 5.77760476  7.18179502  8.58598528  9.99017554 11.3943658 ]
```

Outlier Detection:

What is outlier: Methods used for outlier detection:

Outliers are **observations in a dataset that don't fit in some way**.

Outliers are generally defined as samples that are exceptionally far from the mainstream of data. There is no rigid mathematical definition of what constitutes an outlier; determining whether or not an observation is an outlier is ultimately a subjective exercise.
An outlier may also be explained as a piece of data or observation that deviates drastically from the given norm or average of the data set. An outlier may be caused simply by chance, but it may also indicate measurement error or that the given data set has a heavy-tailed distribution.

Therefore, Outlier Detection may be defined as the process of detecting and subsequently excluding outliers from a given set of data. There are no standardized Outlier identification methods as these are largely dependent upon the data set. Outlier Detection as a branch of data mining has many applications in data stream analysis.

In statistics outlier can be defined as "An Outlier is that observation which is significantly different from all other observations."

It is something odd one out value or observation.

Example:

In a class, we have 100 students and one student who always scores marks on the higher side concerning other students and its score is not much dependent on the Difficulty level of the exam. So, here we consider that student as an outlier.

Learning outliers involved following three parts:

A) Treating with outliers:
B) Outlier Detection
C) Different techniques used for outlier detection and removal.

A) Treating with outliers:

a. **Trimming:** It excludes the outlier values from our analysis. By applying this technique our data becomes thin when there are more outliers present in the dataset. Its main advantage is its **fastest** nature.
b. **Capping:** In this technique, we cap our outliers data and make the limit i.e, above a particular value or less than that value, all the values will be considered as outliers, and the number of outliers in the dataset gives that capping number. **For Example,** if you're working on the income feature, you might find that people above a certain income level behave in the same way as those with a lower income. In this case, you can cap the income value at a level that keeps that intact and accordingly treat the outliers.
c. **Treat outliers as a missing value:** By assuming outliers as the missing observations, treat them accordingly i.e, same as those of missing values.
d. **Discretization:** In this technique, by making the groups we include the outliers in a particular group and force them to behave in the same manner as those of other points in that group. This technique is also known as **Binning**.

B) Outlier Detection:
For Normal distributions: Use empirical relations of Normal distribution.
For Skewed distributions: Use Inter-Quartile Range (IQR) proximity rule.
For Other distributions: Use percentile-based approach.

C) Techniques used for outlier detection and removal

a. **Z-score technique:** Z score is also called standard score. This score helps to understand if a data value is greater or smaller than mean and

how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean.
Z score=(x-means)/standard deviation

If the z score of a data point is more than 3, it indicates that the data point is quite different from the other data points. Such a data point can be an outlier.
For example, in a survey, it was asked how many children a person had.
Suppose the data obtained from people is
1, 2, 2, 2, 3, 1, 1, 15, 2, 2, 2, 3, 1, 1, 2
Here we can see 15 is the outlier
Steps
1) Import necessary libraries
2) Calculate mean standard deviation
3) Calculate z score if z score >3 the print it is an outlier

```
import numpy as np
data = [1, 2, 2, 2, 3, 1, 1, 15, 2, 2, 2, 3, 1, 1, 2]
mean = np.mean(data)
std = np.std(data)
print('mean of the dataset is', mean)
print('std. deviation is', std)
threshold = 3
outlier = []
for i in data:
    z = (i-mean)/std
    if z > threshold:
        outlier.append(i)
print('outlier in dataset is', outlier)
```

```
mean of the dataset is 2.6666666666666665
std. deviation is 3.3598941782277745
outlier in dataset is [15]
```

Detecting outlier with csv file as input
Create csv data file in excel as follows:

	cgpa	Placement_exam_marks	placed
1	6.7	11	0
2	7.6	29	0
3	6.1	6	1
4	6.8	24	0
5	3.5	20	0
6	7.1	10	0
7	7.2	28	0
8	6.1	13	0
9	6.9	25	0
10	6.2	20	0

11	6.7	13	0
12	7.7	23	0
13	6.1	24	0
14	8.8	12	0
15	6.1	23	0
16	6.7	27	0
17	5.6	28	0
18	5.1	23	0
19	5.9	21	0
20	5.6	14	0
21	6.5	15	0
22	7.1	16	0
23	3.4	18	0
24	6.8	10	0
25	3.4	20	0

Steps:

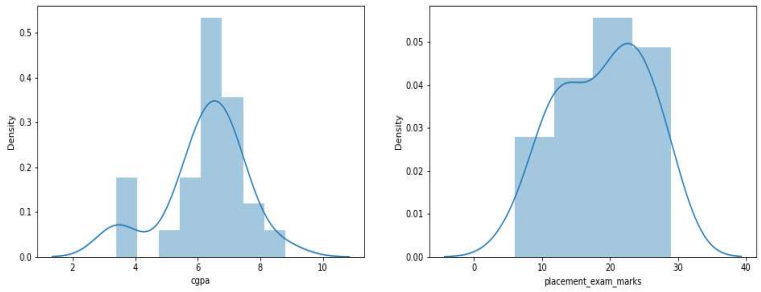
1) Importing packages
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```
2) Read and load the data set
```
df = pd.read_csv('d:\dmdataset\placement.csv')
df.sample(25)
```
3) Plot the distribution plots for the features
```
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['cgpa'])
plt.subplot(1,2,2)
sns.distplot(df['placement_exam_marks'])
plt.show()
```
4) Finding the boundary values
```
print("Highest allowed",df['cgpa'].mean() + 3*df['cgpa'].std())

print("Lowest allowed",df['cgpa'].mean() - 3*df['cgpa'].std())
```
5) Finding the outliers
```
df[(df['cgpa'] > 8.80) | (df['cgpa'] < 5.11)]
```
6) Trimming of outliers
```
new_df = df[(df['cgpa'] < 8.80) & (df['cgpa'] > 5.11)]
new_df
```
7) Capping of outliers
```
upper_limit = df['cgpa'].mean() + 3*df['cgpa'].std()
lower_limit = df['cgpa'].mean() - 3*df['cgpa'].std()
```
8) Now apply the capping

```
df['cgpa'] = np.where(
    df['cgpa']>upper_limit,
    upper_limit,
    np.where(
        df['cgpa']<lower_limit,
        lower_limit,
        df['cgpa']    ))
```

9) See the statistics using describe function
   df['cgpa'].describe()

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('d:\dmdataset\placement.csv')
df.sample(25)
import warnings
warnings.filterwarnings('ignore')
plt.figure(figsize=(16,5))
plt.subplot(1,2,1)
sns.distplot(df['cgpa'])
plt.subplot(1,2,2)
sns.distplot(df['placement_exam_marks'])
plt.show()
print("Highest allowed",df['cgpa'].mean() + 3*df['cgpa'].std())
print("Lowest allowed",df['cgpa'].mean() - 3*df['cgpa'].std())
df[(df['cgpa'] > 8.80) | (df['cgpa'] < 5.11)]
new_df = df[(df['cgpa'] < 8.80) & (df['cgpa'] > 5.11)]
new_df
upper_limit = df['cgpa'].mean() + 3*df['cgpa'].std()
lower_limit = df['cgpa'].mean() - 3*df['cgpa'].std()
df['cgpa'] = np.where(
    df['cgpa']>upper_limit,
    upper_limit,
    np.where(
        df['cgpa']<lower_limit,
        lower_limit,
        df['cgpa']
    )
)
df['cgpa'].describe()
```

Output:



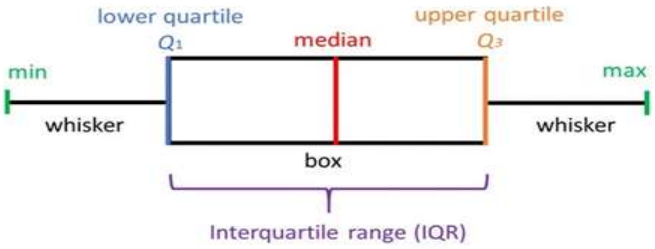

```
Highest allowed 10.132343734867616
Lowest allowed 2.323656265132385

count    25.000000
mean      6.228000
std       1.301448
min       3.400000
25%       5.900000
50%       6.500000
75%       6.900000
max       8.800000
Name: cgpa, dtype: float64
```

b. **IQR based filtering:** Used when our data distribution is skewed.

Each dataset can be divided into quartiles. The first quartile point indicates that 25% of the data points are below that value whereas second quartile is considered as median point of the dataset. The inter quartile method finds the outliers on numerical datasets by following the procedure below.

- Find the first quartile, Q1.
- Find the third quartile, Q3.
- Calculate the IQR. IQR= Q3-Q1.
- Define the normal data range with lower limit as Q1–1.5*IQR and upper limit as Q3+1.5*IQR.
- Any data point outside this range is considered as outlier and should be removed for further analysis.

The concept of quartiles and IQR can best be visualized from the boxplot. It has the minimum and maximum point defined as Q1–1.5*IQR and Q3+1.5*IQR respectively. Any point outside this range is outlier.



Data set can be downloaded from Github: path is
airbnb-listings.csv · GitHub
steps
1) Import the libraries

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

2) Read the input csv file

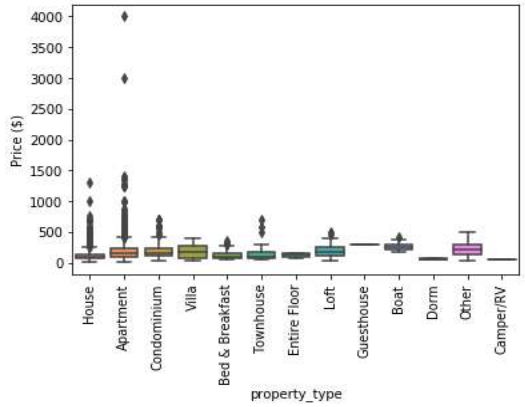
```
df=pd.read_csv('d:\dmdataset\listings.csv')
```

3) Remove special sign like '$' from the price column.(data preprocessing)

```
def remove_sign(x,sign):
    if type(x) is str:
        x = float(x.replace(sign,'').replace(',',''))
    return x
```

4) See the initial distribution in boxplots.

```
df=df[['price','property_type']]
df=pd.DataFrame(df)
df.price = df.price.apply(remove_sign,sign='$')
sns.boxplot(y='price', x='property_type',data=df)
plt.xticks(rotation=90)
plt.ylabel('Price ($)')
```

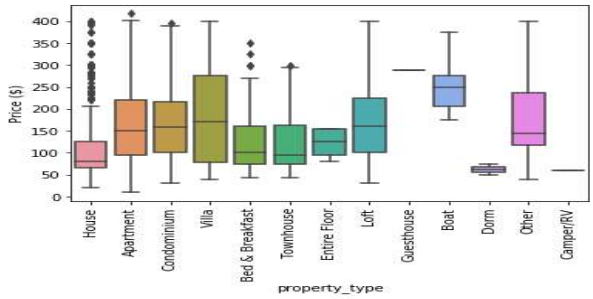


5) Create function to implement IQR method.

```
def remove_outlier_IQR(df):
    Q1=df.quantile(0.25)
    Q3=df.quantile(0.75)
    IQR=Q3-Q1
    df_final=df[~((df<(Q1-1.5*IQR)) | (df>(Q3+1.5*IQR)))]
    return df_final
```

6) Revisit the boxplot after outlier removal. The indices of the bad data points are determined and those are removed from the initial dataset.

```
df_outlier_removed=remove_outlier_IQR(df.price)
df_outlier_removed=pd.DataFrame(df_outlier_removed)
ind_diff=df.index.difference(df_outlier_removed.index)

for i in range(0, len(ind_diff),1):
    df_final=df.drop([ind_diff[i]])
    df=df_final

sns.boxplot(y='price', x='property_type',data=df_final)
plt.xticks(rotation=90)
plt.ylabel('Price ($)')
```



As seen in the boxplot, the majority of the outliers are removed. One can also perform this IQR method in individual rental type and that will remove all the deviant points and result in a cleaner boxplot.

7) Check number of outliers removed. The total number of outliers determined by this process.

```
len(ind_diff)
output:125
```

c. Percentile: This technique works by setting a particular threshold value, which decides based on our problem statement. While we should remove the outliers using capping, then that particular method is known as **Winsorization**. Here we have to always maintain **symmetry** on both sides means if remove 1% from the right then in the left we also drop by 1%.

Steps

1) Import necessary dependencies

```
import numpy as np
import pandas as pd
```

2) Read and Load the dataset

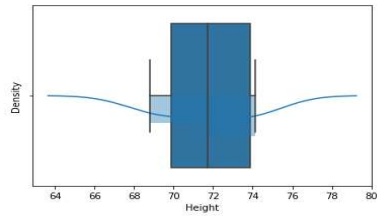
```
df = pd.read_csv('d:\dmdataset\wh.csv')
df.sample(5)
```

3) Plot the distribution plot of “height” feature

```
sns.distplot(df['Height'])
```

4) Plot the box-plot of “height” feature

```
sns.boxplot(df['Height'])
```



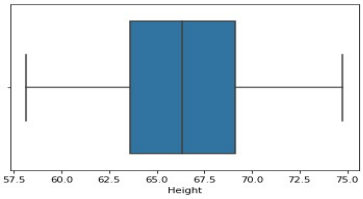
5) Finding upper and lower limit

```
upper_limit = df['Height'].quantile(0.99)
lower_limit = df['Height'].quantile(0.01)
```

6) Apply trimming

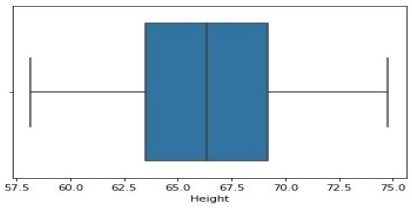
```
new_df = df[(df['Height'] <= 74.78) & (df['Height'] >= 58.13)]
```

Compare the distribution and box-plot after trimming
sns.distplot(new_df['Height'])
sns.boxplot(new_df['Height'])



7) Apply Capping (Winsorization)
df['Height'] = np.where(df['Height'] >= upper_limit,
upper_limit,np.where(df['Height']<= lower_limit, lower_limit,df['Height']))
8) Compare the distribution and box-plot after capping
sns.distplot(df['Height'])

sns.boxplot(df['Height'])



Set A : Simple Linear Regression

1) Consider following observations/data. And apply simple linear regression and find out estimated coefficients b0 and b1.(use numpy package)
x= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9,11,13]

y = ([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,16, 18]

2) Consider following observations/data. And apply simple linear regression and find out estimated coefficients b1 and b1 Also analyse the performance of the model (Use sklearn package)
x = np.array([1,2,3,4,5,6,7,8])
y = np.array([7,14,15,18,19,21,26,23])

3) Consider the student data set It can be downloaded from: https://drive.google.com/open?id=1oakZCv7g3mlmCSdv9J8kdSaqO5_6dIOw
Write a programme in python to apply simple linear regression and find out mean absolute error, mean squared error and root mean squared error.

Set B: Multiple Linear Regression

1) Write a python program to implement multiple Linear Regression model for a car dataset.

Dataset can be downloaded from: https://www.w3schools.com/python/python_ml_multiple_regression.asp

2) Write a python programme to implement multiple linear regression model for stock market data frame as follows:
Stock_Market = {'Year': [2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2017,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016,2016],
'Month': [12, 11,10,9,8,7,6,5,4,3,2,1,12,11,10,9,8,7,6,5,4,3,2,1],
'Interest_Rate': [2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75],
'Unemployment_Rate': [5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,5.8,6.1,6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1],
'Stock_Index_Price': [1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,965,943,958,971,949,884,866,876,822,704,719] }
And draw a graph of stock market price verses interest rate.

Set C: Outlier Detection

1) Write a programme in python to print the number of outliers. Generate 200 samples, from a normal distribution, cantered around the value 100, with a standard deviation of 5.

Signature of the Instructor: **Date:**

Assignment Evaluation

0: Not Done	2: Late Complete	4: Complete
1: Complete	3. Needs Improvement	5: Well Done

Assignment 6: Clustering

Objectives:

Students will be able to

- Discover structures and patterns in high-dimensional data.
- Group data with similar patterns together.
- Describe a case where clustering is appropriate, and what insight it might extract from the data.
- Explain the K-means clustering algorithm with its interpretation.
- Use the elbow method to choose the number of clusters for K-means.
- Visualize the output of K-means clustering in Python using coloured scatter plots.

Prerequisites:

- ✓ Basic knowledge of Python Programming
- ✓ https://ubc-dsci.github.io/introduction-to-datascience/clustering.html
- ✓ https://www.inf.ed.ac.uk/teaching/courses/bio2/lectures09/clustering.pdf

Introduction:

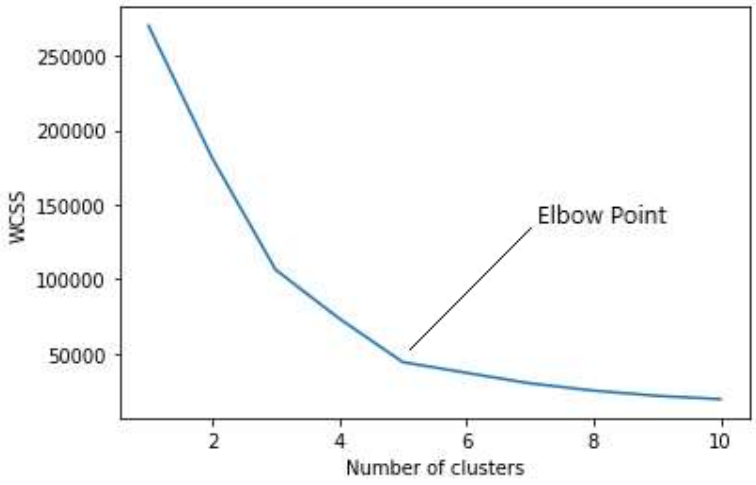
Clustering is an unsupervised machine learning technique. It is the process of division of the dataset into groups in which the members in the same group possess similarities in features. The commonly used clustering algorithms are K-Means clustering, Hierarchical clustering, Density-based clustering, Model-based clustering, etc.

K-Means Clustering:

It is the simplest and commonly used iterative type unsupervised learning algorithm. In this, we randomly initialize the K number of centroids in the data (the number of k is found using the Elbow method) and iterates these centroids until no change happens to the position of the centroid. Let's go through the steps involved in K means clustering for a better understanding. 1) Select the number of clusters for the dataset (K) 2) Select K number of centroids 3) By calculating the Euclidean distance or Manhattan distance assign the points to the nearest centroid, thus creating K groups 4) Now find the original centroid in each group 5) Again reassign the whole data point based on this new centroid, then repeat step 4 until the position of the centroid doesn't change. Finding the optimal number of clusters is an important part of this algorithm. A commonly used method for finding optimal K value is Elbow Method.

Elbow Method:

In the Elbow method, we are actually varying the number of clusters (K) from 1 – 10. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.



Python Implementation of K-means Clustering Algorithm:

let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id, Gender, Age, Annual Income ($), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Importing Libraries

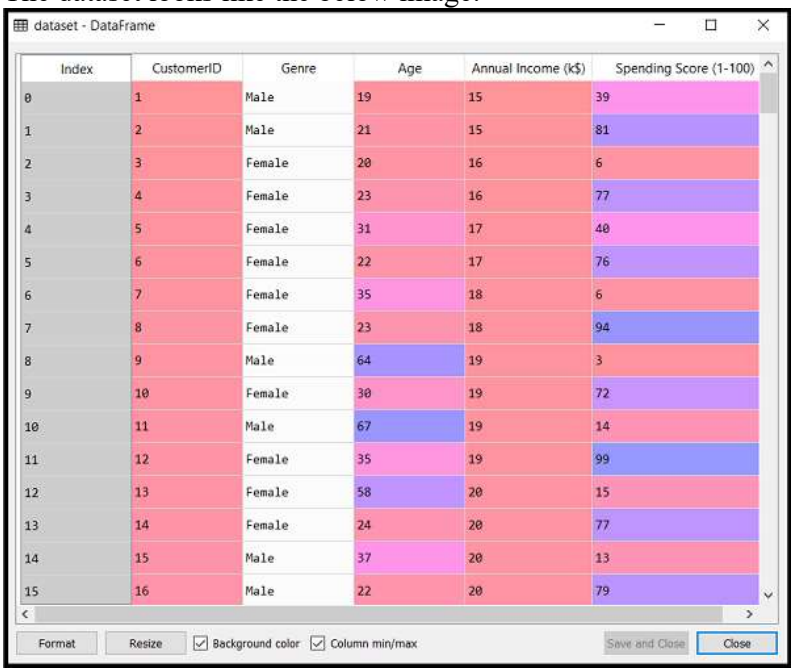
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

Importing the Dataset:

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')
```

The dataset looks like the below image:



Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

Extracting Independent Variables:

```
x = dataset.iloc[:, [3, 4]].values
```

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Finding the optimal number of clusters using the elbow method:

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

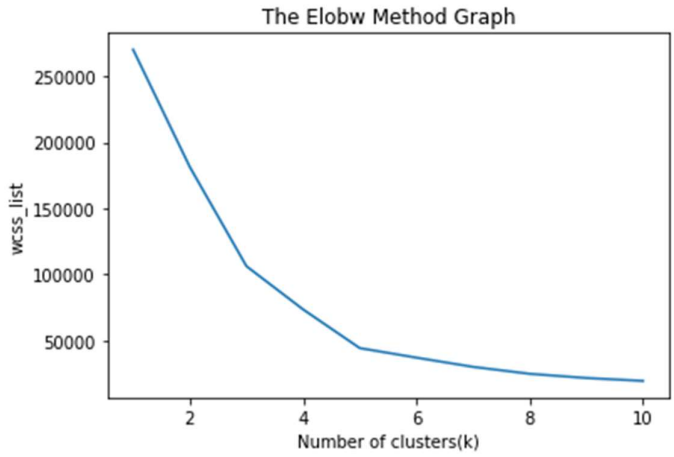
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10^{th} value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

After executing the above code, we will get the below output:



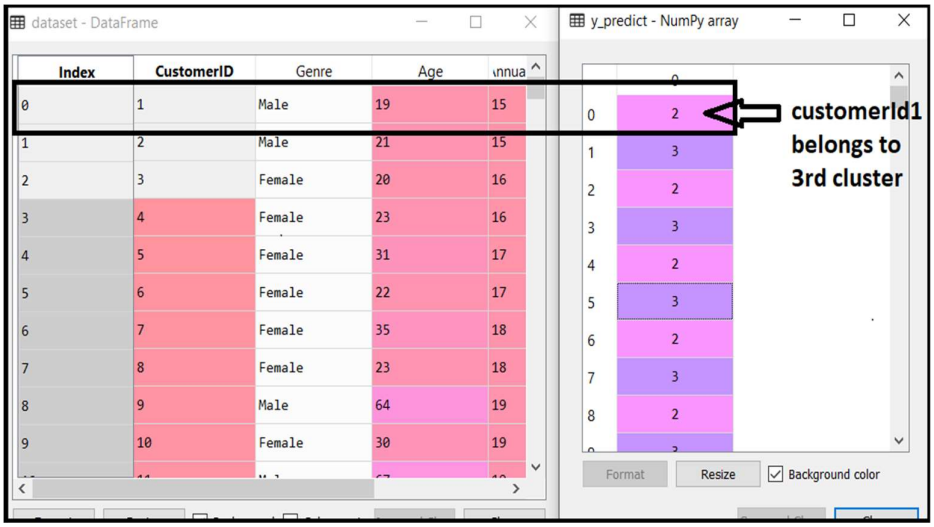
Training the K-means algorithm on the training dataset

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y_predict** to train the model.

By executing the above lines of code, we will get the y_predict variable.

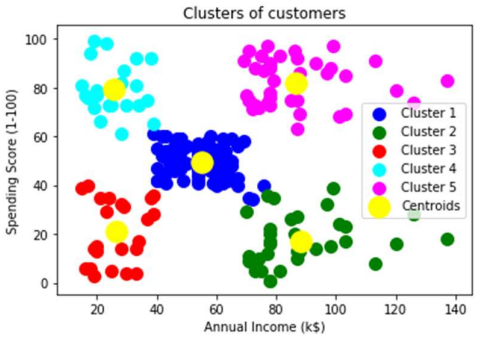


Visualizing the Clusters

```
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the mtp.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.

After executing the above code, we will get the below output:



Hierarchical clustering:

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

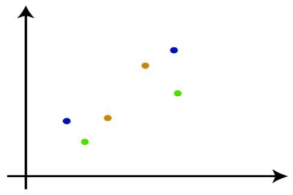
As we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

Agglomerative Hierarchical clustering

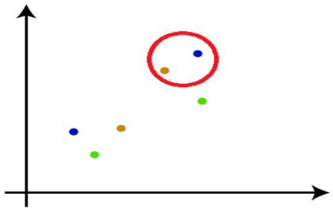
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.

How the Agglomerative Hierarchical clustering Work?

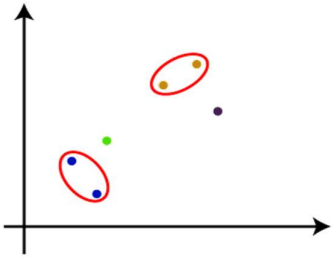
Step-1: Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.



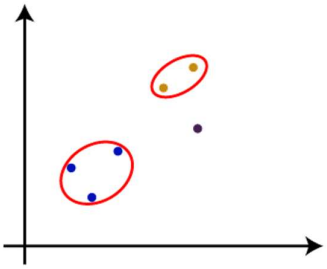
Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.



Step-3: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.



Step-4: Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:



Step-5: Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

Working of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

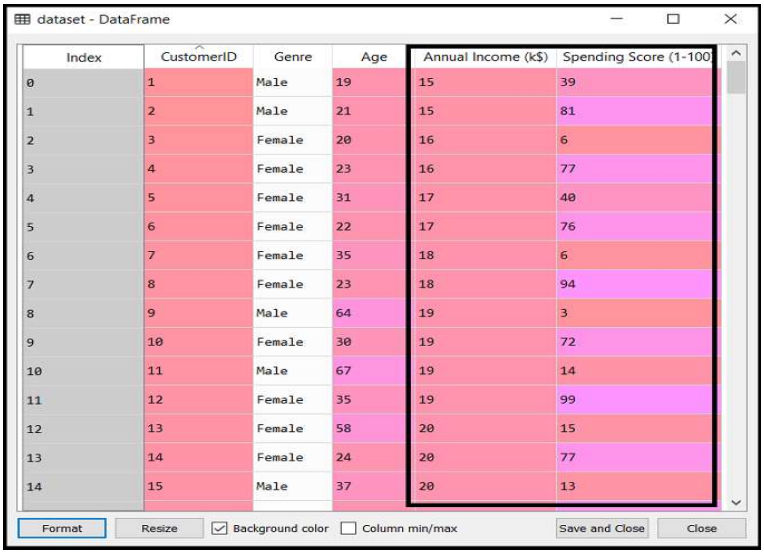
Python Implementation of Agglomerative Hierarchical Clustering

Importing the libraries

```
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Importing the dataset

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')
```



Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13

Extracting the matrix of features

```
x = dataset.iloc[:, [3, 4]].values
```

Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters. So, we are considering the Annual income and spending score as the matrix of features.

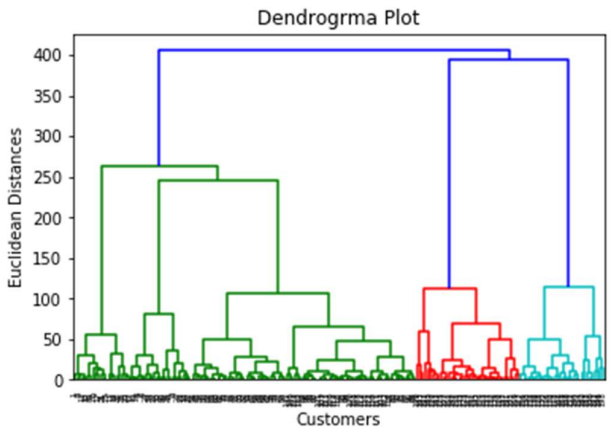
Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use **scipy** library as it provides a function that will directly return the dendrogram.

```
#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
```

In the above lines of code, we have imported the **hierarchy** module of scipy library. This module provides us a method **shc.denrogram(),** which takes the **linkage()** as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the x(matrix of features), and method "**ward**," the popular method of linkage in hierarchical clustering.

By executing the above lines of code, we will get the below output**:**



Training the hierarchical clustering model

```
#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x)
```

Visualizing the clusters

```
#visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

By executing the above lines of code, we will get the below output:

Set A

1. Write a python program to implement k-means algorithm to build prediction model (Use Credit Card Dataset CC GENERAL.csv Download from kaggle.com)
2. Write a python program to implement hierarchical Agglomerative clustering algorithm. (Download Customer.csv dataset from github.com).

Set B

1. Write a python program to implement k-means algorithms on a synthetic dataset.
2. Write a python program to implement hierarchical clustering algorithm. (Download Wholesale customers data dataset from github.com).

Set C

1. Write a python program to implement Agglomerative clustering on a synthetic dataset. (use inbuilt Iris data set).

Signature of the Instructor: **Date:**

Assignment Evaluation

0: Not Done	2: Late Complete	4: Complete
1: Complete	3. Needs Improvement	5: Well Done

References:

- ✓ https://realpython.com/linear-regression-in-python/
- ✓ Machine Learning Basics: Simple Linear Regression | by Gurucharan M K | Towards Data Science
- ✓ Linear Regression (Python Implementation) - GeeksforGeeks
- ✓ Solving Linear Regression in Python - GeeksforGeeks
- ✓ Simple Linear Regression in Python | Simple Linear Regression Tutorial (analyticsvidhya.com)
- ✓ Simple Linear Regression: A Practical Implementation in Python - AskPython
- ✓ https://datatofish.com/multiple-linear-regression-python/
- ✓ How to Detect and Remove Outliers | Outlier Detection And Removal (analyticsvidhya.com)
- ✓ Outlier Detection: An Introduction To Its Techniques (digitalvidya.com)
- ✓ https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
- ✓ https://www.javapoint.com
- ✓ https://www.tutorialspoint.com/data_mining/dm_classification_prediction.htm
- ✓ https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
- ✓ https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
- ✓ https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
- ✓ https://github.com/Eligijus112/decision-tree-python
- ✓ https://archive.ics.uci.edu/ml/datasets/Zoo