SIES COLLEGE OF ARTS, SCIENCE & COMMERCE (EMPOWERED AUTONOMOUS)

SION(W),MUMBAI-22

DEPARTMENT OF INFORMATION TECHNOLOGY

MSc(IT), SEMESTER I

Practical Journal

For the Subject

Introduction to Data Science

Submitted by

Yogesh Chatrooram Sahu

FMSC2526179

For the Academic Year

2024-2025

SIES College of Arts, Science & Commerce (Empowered Autonomous)
RISE WITH EDUCATION
NAAC REACCREDITED "A" GRADE

SIES College of Arts, Science and Commerce (Empowered Autonomous), Sion (W), Mumbai – 400 022.

Department of Information Technology

CERTIFICATE

This is to certify that Mr. **Yogesh Chatrooram Sahu,** of MSc [Information Technology] Semester - I, Seat No. **FMSC2526179** has successfully completed the practical's for the subject of **Introduction to Data Science** as a partial fulfilment of the degree **M.Sc.(I.T.)** during the academic year 2025-26.

Faculty-in-Charge

Anita Gupta

Examiner

Course Co-Ordinator

Sudha Bhagavatheeswaran

College Seal

Date:

INDEX

PRACTICAL 1

LOADING DATA FROM DIFFERENT SOURCE FILES FORMATS.

AIM: of this Practical is to Understand most commonly use file formats in Machine Learning & Data Science (use g Python)

```python
# Reading the data from CSV in Python

import pandas as pd

#df = pd.read_csv('/home/Loan_Prediction/train.csv) # Different Location File

df_sani = pd.read_csv('/content/train.csv') # Same File

# Above code will load the train.csv file in DataFrame df.
print(df_sani)
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                 Name     Sex   Age  SibSp  \
0                             Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                                Heikkinen, Miss. Laina  female  26.0      0
3          Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                              Allen, Mr. William Henry    male  35.0      0
..                                                  ...     ...   ...    ...
886                              Montvila, Rev. Juozas    male  27.0      0
887                       Graham, Miss. Margaret Edith  female  19.0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                              Behr, Mr. Karl Howell    male  26.0      0
890                                Dooley, Mr. Patrick    male  32.0      0
```

```
# Reading the data from XLSX file

import pandas as pd

df =  pd.read_excel("/content/file_example_XLS_1000.xls")
#df = pd.read_excel("/content/file_example_XLS_1000.xls", sheetname = "Invoice")

# Above code will load the sheet "Invoice" from "train.xlsx" file in DataFrame df.
print(df)
```

```
     Unnamed: 0 First Name   Last Name  Gender        Country  Age  \
0             1      Dulce       Abril  Female  United States   32
1             2       Mara   Hashimoto  Female  Great Britain   25
2             3     Philip        Gent    Male         France   36
3             4   Kathleen      Hanner  Female  United States   25
4             5    Nereida     Magwood  Female  United States   58
..          ...        ...         ...     ...            ...  ...
995         996       Roma  Lafollette  Female  United States   34
996         997     Felisa        Cail  Female  United States   28
997         998   Demetria       Abbey  Female  United States   32
998         999     Jeromy        Danz    Male  United States   39
999        1000   Rasheeda      Alkire  Female  United States   29

           Date    Id
0    15/10/2017  1562
1    16/08/2016  1582
2    21/05/2015  2587
3    15/10/2017  3549
4    16/08/2016  2468
..          ...   ...
995  15/10/2017  2654
996  16/08/2016  6525
997  21/05/2015  3265
998  15/10/2017  3265
999  16/08/2016  6125

[1000 rows x 8 columns]
```

```python
# Reading a .ZIP file in Python

# You can read a zip file by importing the “zipfile” package.
# Below is the python code which can read the “train.csv” file that is inside the “T.zip”.

# importing required modules
from zipfile import ZipFile

# specifying the zip file name
file_name = "/content/AI_Practical-main.zip"

# opening the zip file in READ mode
with ZipFile(file_name, 'r') as zip:
  # printing all the contents of the zip file
  zip.printdir()

  # extracting all the files
  print('Extracting all the files now...')
  zip.extractall()
  print('Done!')
```

```
File Name                                             Modified             Size
AI_Practical-main/                             2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 3/          2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 3/Bayes_Theorem.ipynb 2021-01-22 05:58:54         1993
AI_Practical-main/Practical Number 4/          2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 4/4_Conditional_prob.ipynb 2021-01-22 05:58:54        19018
AI_Practical-main/Practical Number 4/Joint_Probability.ipynb 2021-01-22 05:58:54         2595
AI_Practical-main/Practical Number 5/          2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 5/Rule_Based_Matcher.ipynb 2021-01-22 05:58:54       173448
AI_Practical-main/Practical Number 6/          2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 6/perform_different_operations_on_fuzzy_set (1).ipynb 2021-01-22 05:58:54         9014
AI_Practical-main/Practical Number 8/          2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 8/K_Means.ipynb 2021-01-22 05:58:54        92015
AI_Practical-main/Practical Number 9/          2021-01-22 05:58:54            0
AI_Practical-main/Practical Number 9/SVM_Implemtation.ipynb 2021-01-22 05:58:54       193288
Extracting all the files now...
Done!
```

```python
# Suppose the above text written in a file called text.txt and you want to read this so you can refer the below code.

text_file = open("text.txt", "r")
lines = text_file.read()
print(lines)
```

```
In my previous article, I introduced you to the basics of Apache Spark, different data representations
(RDD / DataFrame / Dataset) and basics of operations (Transformation and Action). We even solved a machine
learning problem from one of our past hackathons. In this article, I will continue from the place I left in
my previous article. I will focus on manipulating RDD in PySpark by applying operations
(Transformation and Actions).
```

```python
[5]  # Reading a JSON file

     import pandas as pd

     df = pd.read_json("train.json")
     print(df)
```

```
                                      Employee
0  {'id': '1', 'Name': 'Ankit', 'Sal': '1000'}
1  {'id': '2', 'Name': 'Faizy', 'Sal': '2000'}
```

```
[6] # Reading XML in python

    # For reading the data from XML file you can import xml.etree. ElementTree library.

    # importing element tree
    # under the alias of ET
    import xml.etree.ElementTree as ET

    # Passing the path of the
    # xml document to enable the
    # parsing process
    tree = ET.parse('/content/new_train.xml')

    # getting the parent tag of
    # the xml document
    root = tree.getroot()

    # printing the root (parent) tag
    # of the xml document, along with
    # its memory location
    print(root.tag)

    # printing the attributes of the
    # first tag from the parent
    print(root[0].attrib)
```

```
breakfast_menu
{}
```

```
[7] # Reading the HTML file

    #import the library used to query a website
    import urllib.request


    #specify the url
    wiki = "https://en.wikipedia.org/wiki/List_of_state_and_union_territory_capitals_in_India"


    #Query the website and return the html to the variable 'page'
    page = urllib.request.urlopen(wiki)


    #import the Beautiful soup functions to parse the data returned from the website
    from bs4 import BeautifulSoup

    #Parse the html in the 'page' variable, and store it in Beautiful Soup format
    soup = BeautifulSoup(page)



    # Use function “prettify” to look at nested structure of HTML page
    print(soup.prettify())
```

```
<!DOCTYPE html>
<html class="client-nojs vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-sticky-header-disabled vector-feature-page-tools-pinned-
 <head>
  <meta charset="utf-8"/>
  <title>
   List of state and union territory capitals in India - Wikipedia
  </title>
  <script>
   (function(){var className="client-js vector-feature-language-in-header-enabled vector-feature-language-in-main-page-header-disabled vector-feature-sticky-header-disabled vector-feature-pag
"wgMonthNames":["","January","February","March","April","May","June","July","August","September","October","November","December"],"wgRequestId":"01d8dc49-5720-46b6-9f2c-9231b02016e2","wgCanon
"wgPageViewLanguage":"en","wgPageContentLanguage":"en","wgPageContentModel":"wikitext","wgRelevantPageName":"List_of_state_and_union_territory_capitals_in_India","wgRelevantArticleId":2371868
"wgVector2022LanguageInHeader":true,"wgULSisLanguageSelectorEmpty":false,"wgWikibaseItemId":"Q3518799","wgCheckUserClientHintsHeadersJsApi":["architecture","bitness","brands","fullVersionList
"ext.cite.ux-enhancements","ext.scribunto.logs","site","mediawiki.page.ready","jquery.tablesorter","jquery.makeCollapsible","mediawiki.toc","skins.vector.js","ext.centralNotice.geoIP","ext.ce
  </script>
  <script>
   (RLQ=window.RLQ||[]).push(function(){mw.loader.impl(function(){return["user.options@12s5i",function($,jQuery,require,module){mw.user.tokens.set({"patrolToken":"+\\","watchToken":"+\\","csr
}];});});
  </script>
  <link href="/w/load.php?lang=en&amp;modules=ext.cite.styles%7Cext.uls.interlanguage%7Cext.visualEditor.desktopArticleTarget.noscript%7Cext.wikimediaBadges%7Cext.wikimediamessages.styles%7Cj
  <script async="" src="/w/load.php?lang=en&amp;modules=startup&amp;only=scripts&amp;raw=1&amp;skin=vector-2022">
  </script>
  <meta content="" name="ResourceLoaderDynamicStyles"/>
  <link href="/w/load.php?lang=en&amp;modules=site.styles&amp;only=styles&amp;skin=vector-2022" rel="stylesheet"/>
  <meta content="MediaWiki 1.43.0-wmf.27" name="generator"/>
  <meta content="origin" name="referrer"/>
  <meta content="origin-when-cross-origin" name="referrer"/>
  <meta content="max-image-preview:standard" name="robots"/>
  <meta content="telephone=no" name="format-detection"/>
  <meta content="width=1120" name="viewport"/>
  <meta content="List of state and union territory capitals in India - Wikipedia" property="og:title"/>
  <meta content="website" property="og:type"/>
  <link href="//upload.wikimedia.org" rel="preconnect"/>
  <link href="//en.m.wikipedia.org/wiki/List_of_state_and_union_territory_capitals_in_India" media="only screen and (max-width: 640px)" rel="alternate"/>
  <link href="/static/apple-touch/wikipedia.png" rel="apple-touch-icon"/>
  <link href="/static/favicon/wikipedia.ico" rel="icon"/>
  <link href="/w/rest.php/v1/search" rel="search" title="Wikipedia (en)" type="application/opensearchdescription+xml"/>
  <link href="//en.wikipedia.org/w/api.php?action=rsd" rel="EditURI" type="application/rsd+xml"/>
  <link href="https://en.wikipedia.org/wiki/List_of_state_and_union_territory_capitals_in_India" rel="canonical"/>
  <link href="https://creativecommons.org/licenses/by-sa/4.0/deed.en" rel="license"/>
```

```
[8] print(soup.title)

<title>List of state and union territory capitals in India - Wikipedia</title>
```

```
[9] print(soup.title.string)

List of state and union territory capitals in India - Wikipedia
```

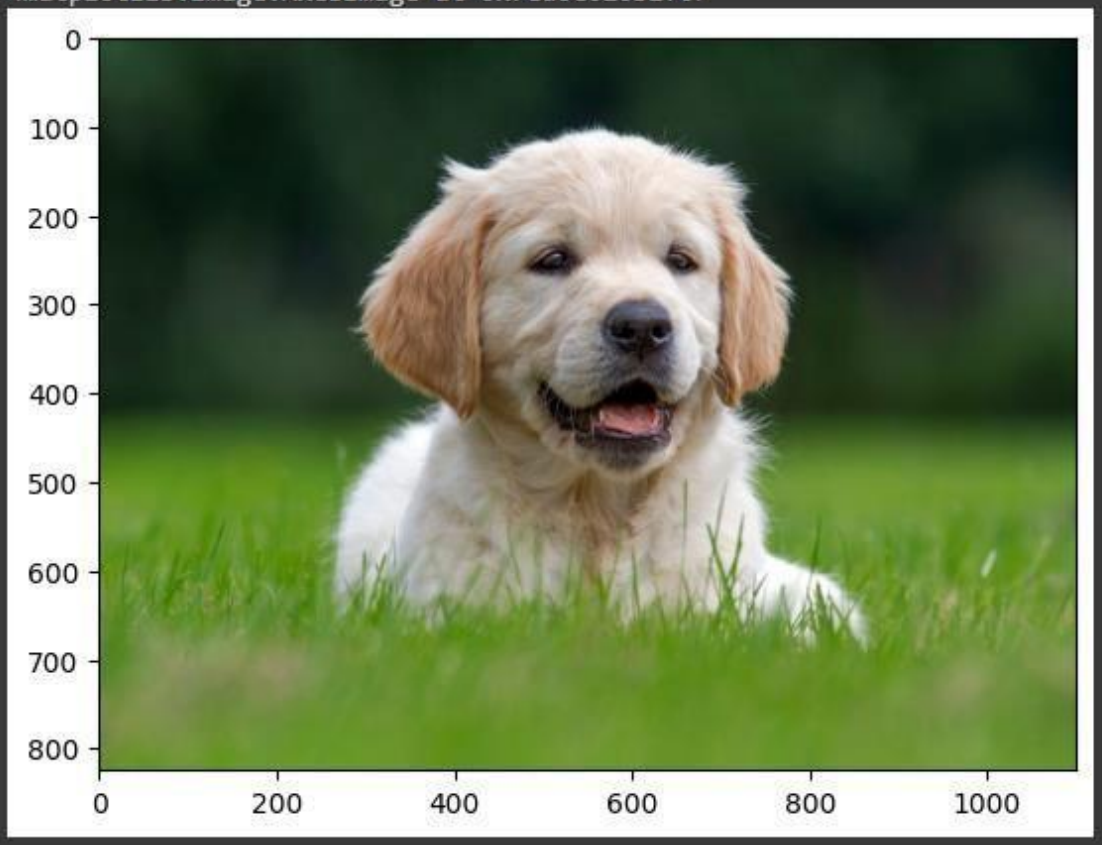
```
# finding all the tables

all_tables=soup.find_all('table')
all_tables
```

```
<td>Imphal
</td>
<td>Imphal
</td>
<td>1972
</td>
<td> –
</td></tr>
<tr>
<td><b><a href="/wiki/Meghalaya" title="Meghalaya">Meghalaya</a></b>
</td>
<td><a href="/wiki/Shillong" title="Shillong">Shillong</a>
</td>
<td>Shillong
</td>
<td>Shillong
</td>
<td>1972
</td>
<td> –
</td></tr>
<tr>
<td><b><a href="/wiki/Mizoram" title="Mizoram">Mizoram</a></b>
</td>
<td><a href="/wiki/Aizawl" title="Aizawl">Aizawl</a>
</td>
<td>Aizawl
</td>
<td><a href="/wiki/Guwahati" title="Guwahati">Guwahati</a>
</td>
<td>1987
</td>
<td> –
```

```
[12] right_table=soup.find('table', class_='wikitable sortable plainrowheaders')
     right_table
```

```
[13] # importing matplotlib modules
     import matplotlib.image as mpimg
     import matplotlib.pyplot as plt

     # Read Images
     img = mpimg.imread('dog.jpg')

     # Output Images
     plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7ed6c020b1f0>
```



```
[14] # image using PIL module

     # importing PIL
     from PIL import Image

     # Read image
     img = Image.open('dog.jpg')

     # Output Images
     img.show()

     # prints format of image
     print(img.format)

     # prints mode of image
     print(img.mode)
```

```
JPEG
RGB
```

```
[15] # importing libraries
     import pandas as pd


     # Create dataframe
     data = {'name': ['Swapnil', 'Rahul', 'Pradnya', 'Aishwarya', 'Pawan'],
             'age': [42, 52, 36, 24, 73],
             'preTestScore': [4, 24, 31, 2, 3],
             'postTestScore': [25, 94, 57, 62, 70]}
     df = pd.DataFrame(data, columns = ['name', 'age', 'preTestScore', 'postTestScore'])
     df
```

	name	age	preTestScore	postTestScore
0	Swapnil	42	4	25
1	Rahul	52	24	94
2	Pradnya	36	31	57
3	Aishwarya	24	2	62
4	Pawan	73	3	70

```
[16] # Looking Statistically on data

     # The sum of all the ages
     df['age'].sum()
```

```
227
```

```
[17] # Mean of preTestScore

     df['preTestScore'].mean()
```

```
12.8
```

```
[18] # Cumulative sum of preTestScores, moving from the rows from the top

     df['preTestScore'].cumsum()
```

	preTestScore
0	4
1	28
2	59
3	61
4	64

dtype: int64

```
[19] # Summary statistics on preTestScore
     df['preTestScore'].describe()
```

	preTestScore
count	5.000000
mean	12.800000
std	13.663821
min	2.000000
25%	3.000000
50%	4.000000
75%	24.000000
max	31.000000

dtype: float64

```
[20] # Count the number of non-NA values
     df['preTestScore'].count()
```

```
5
```

```
[21] # Minimum value of preTestScore
     df['preTestScore'].min()
```

```
2
```

```
[22] # Maximum value of preTestScore
     df['preTestScore'].max()
```

```
31
```

```
[23] # Median value of preTestScore
     df['preTestScore'].median()
```

```
4.0
```

```
[24] # Sample variance of preTestScore values
     df['preTestScore'].var()
```

```
186.7
```

```
[25] # Sample standard deviation of preTestScore values
     df['preTestScore'].std()
```

```
13.663820841916802
```

```
[26] # Skewness of preTestScore values
     df['preTestScore'].skew()
```

```
0.7433452457326751
```

```
[27] # Kurtosis of preTestScore values
     df['preTestScore'].kurt()
```

```
-2.4673543738411547
```

```
[29] # Correlation Matrix Of Values, excluding non-numeric columns
     df.select_dtypes(include=['number']).corr()
```

	age	preTestScore	postTestScore
age	1.000000	-0.105651	0.328852
preTestScore	-0.105651	1.000000	0.378039
postTestScore	0.328852	0.378039	1.000000

```
[31] # Covariance Matrix Of Values, excluding non-numeric columns
     df.select_dtypes(include=['number']).cov()
```

	age	preTestScore	postTestScore
age	340.80	-26.65	151.20
preTestScore	-26.65	186.70	128.65
postTestScore	151.20	128.65	620.30

PRACTICAL 2

EXPLORATORY DATA ANALYSIS

Aim : To achieve, understand and implement following topics

- i)Handle Missing Values
- ii)Removing Duplicates
- iii)Outlier Treatment
- iv)Normalizing and Scaling(Numerical Variables)
- v)Encoding Categorical variables (Dummy Variables)
- vi)Bivariate Analysis.

```
# Importing all the necessary libraries
import numpy as np   # numerial python
import pandas as pd  # dataframe working
import matplotlib.pyplot as plt #  plotting
import seaborn as sn # advance plotting
%matplotlib inline
```

```
df_saniya = pd.read_excel("/content/EDA Cars.xlsx")
df_saniya.head(30)
```

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CAR TYPE	CAR AGE	CITY	POSTAL CODE
0	1	125301.2425	No	F	Bachelors	Blue Collar	45.703013	Commercial	17430.0	Sports Car	7.0	Texas	42420.0
1	2	50815.44531	No	M	High School	NaN	20.591628	Private	18930.0	Minivan	1.0	Texas	42420.0
2	3	62977.82416	NaN	F	Bachelors	Clerical	33.639949	Private	NaN	SUV	1.0	Texas	42420.0
3	4	77099.96624	No	F	NaN	Lawyer	15.415676	NaN	18300.0	Sports Car	11.0	Texas	42420.0
4	5	130794.5742	No	M	High School	NaN	NaN	Commercial	28340.0	Panel Truck	10.0	Texas	42420.0
5	6	NaN	NaN	F	High School	Home Maker	48.360191	NaN	6000.0	SUV	5.0	Texas	42420.0
6	7	87460.05269	No	M	High School	Manager	45.000488	NaN	15420.0	Minivan	1.0	Texas	42420.0
7	8	NaN	Yes	F	High School	Blue Collar	15.665947	NaN	11290.0	Sports Car	1.0	Texas	NaN
8	9	@@	NaN	F	NaN	Clerical	26.392961	NaN	10030.0	SUV	1.0	Texas	42420.0
9	10	@@	Yes	M	High School	Blue Collar	27.490749	NaN	NaN	Panel Truck	NaN	Texas	42420.0
10	11	16988.72135	No	M	High School	Blue Collar	20.450016	NaN	NaN	Pickup	6.0	Texas	42420.0
11	12	@@	No	F	High School	Blue Collar	61.603208	Commercial	9360.0	Sports Car	NaN	NaN	42420.0
12	13	16352.02931	Yes	F	NaN	Home Maker	NaN	Private	10520.0	SUV	1.0	Texas	42420.0

✓ 0s completed at 1:05 PM

```
[3] df_saniya.shape

(303, 13)
```

```
[4] df_saniya.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   INDEX           303 non-null    int64
 1   INCOME          265 non-null    object
 2   MARITAL STATUS  275 non-null    object
 3   SEX             297 non-null    object
 4   EDUCATION       259 non-null    object
 5   JOB             257 non-null    object
 6   TRAVEL TIME     262 non-null    float64
 7   USE             250 non-null    object
 8   MILES CLOCKED   278 non-null    float64
 9   CAR TYPE        293 non-null    object
 10  CAR AGE         283 non-null    float64
 11  CITY            297 non-null    object
 12  POSTAL CODE     300 non-null    float64
dtypes: float64(4), int64(1), object(8)
memory usage: 30.9+ KB
```

```
df_saniya.describe()
```

	INDEX	TRAVEL TIME	MILES CLOCKED	CAR AGE	POSTAL CODE
count	303.000000	262.000000	278.000000	283.000000	300.000000
mean	139.640264	34.282098	13591.978417	6.265018	50712.196667
std	85.178422	14.910178	7167.328655	5.111218	24141.029290
min	1.000000	5.000000	1500.000000	1.000000	11435.000000
25%	62.500000	24.449874	7900.000000	1.000000	42420.000000
50%	138.000000	33.564757	12065.000000	6.000000	47150.000000
75%	213.500000	43.907339	18240.000000	10.000000	61701.000000
max	289.000000	83.617643	38000.000000	20.000000	90049.000000

```
[6] df_saniya.describe(include=['object', 'int','float'])
```

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CAR TYPE	CAR AGE	CITY	POSTAL CODE
count	303.000000	265.0	275	297	259	257	262.000000	250	278.000000	293	283.000000	297	300.000000
unique	NaN	222.0	2	2	4	8	NaN	2	NaN	6	NaN	11	NaN
top	NaN	0.0	No	F	High School	Blue Collar	NaN	Private	NaN	SUV	NaN	Houston	NaN
freq	NaN	25.0	151	165	161	96	NaN	133	NaN	93	NaN	39	NaN
mean	139.640264	NaN	NaN	NaN	NaN	NaN	34.282098	NaN	13591.978417	NaN	6.265018	NaN	50712.196667
std	85.178422	NaN	NaN	NaN	NaN	NaN	14.910178	NaN	7167.328655	NaN	5.111218	NaN	24141.029290
min	1.000000	NaN	NaN	NaN	NaN	NaN	5.000000	NaN	1500.000000	NaN	1.000000	NaN	11435.000000
25%	62.500000	NaN	NaN	NaN	NaN	NaN	24.449874	NaN	7900.000000	NaN	1.000000	NaN	42420.000000
50%	138.000000	NaN	NaN	NaN	NaN	NaN	33.564757	NaN	12065.000000	NaN	6.000000	NaN	47150.000000
75%	213.500000	NaN	NaN	NaN	NaN	NaN	43.907339	NaN	18240.000000	NaN	10.000000	NaN	61701.000000
max	289.000000	NaN	NaN	NaN	NaN	NaN	83.617643	NaN	38000.000000	NaN	20.000000	NaN	90049.000000

```
[7] df_saniya.INCOME

0      125301.2425
1      50815.44531
2      62977.82416
3      77099.96624
4      130794.5742
          ...
298    15251.52473
299    18408.39545
300            NaN
301            NaN
302              0
Name: INCOME, Length: 303, dtype: object
```

```
df_saniya[df_saniya['INCOME'] == '@@']
```

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CAR TYPE	CAR AGE	CITY	POSTAL CODE
8	9	@@	NaN	F	NaN	Clerical	26.392961	NaN	10030.0	SUV	1.0	Texas	42420.0
9	10	@@	Yes	M	High School	Blue Collar	27.490749	NaN	NaN	Panel Truck	NaN	Texas	42420.0
11	12	@@	No	F	High School	Blue Collar	61.603208	Commercial	9360.0	Sports Car	NaN	NaN	42420.0
16	17	@@	No	F	High School	Blue Collar	42.500815	Private	NaN	SUV	4.0	Texas	NaN
20	21	@@	Yes	F	Bachelors	Blue Collar	NaN	Commercial	5200.0	SUV	3.0	Texas	42420.0

```
[9] df_saniya['INCOME'] = df_saniya['INCOME'].replace(to_replace = '@@', value= np.nan)
    df_saniya['INCOME'] = df_saniya['INCOME'].astype(float)
```

```
[10] df_saniya[df_saniya['INCOME'] == '@@']
```

INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CAR TYPE	CAR AGE	CITY	POSTAL CODE

```
[11] df_saniya['TRAVEL TIME'] = df_saniya['TRAVEL TIME'].replace(to_replace = '*****', value = np.nan)
     df_saniya['TRAVEL TIME'] = df_saniya['TRAVEL TIME'].astype(float)

[12] df_saniya['MILES CLOCKED'] = df_saniya['MILES CLOCKED'].replace(to_replace = 'Na', value = np.nan)
     df_saniya['MILES CLOCKED'] = df_saniya['MILES CLOCKED'].replace(to_replace = '*****', value = np.nan)
     df_saniya['MILES CLOCKED'] = df_saniya['MILES CLOCKED'].astype(float)

[13] df_saniya.replace(to_replace='*****', value= np.nan, inplace = True)
```

```
[14] df_saniya.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   INDEX           303 non-null    int64
 1   INCOME          260 non-null    float64
 2   MARITAL STATUS  275 non-null    object
 3   SEX             297 non-null    object
 4   EDUCATION       259 non-null    object
 5   JOB             257 non-null    object
 6   TRAVEL TIME     262 non-null    float64
 7   USE             250 non-null    object
 8   MILES CLOCKED   278 non-null    float64
 9   CAR TYPE        293 non-null    object
 10  CAR AGE         283 non-null    float64
 11  CITY            297 non-null    object
 12  POSTAL CODE     300 non-null    float64
dtypes: float64(5), int64(1), object(7)
memory usage: 30.9+ KB
```

```
[15] # Check for missing values in any column or Handling missing values in dataset

     df_saniya.isnull().sum()
```

```
INDEX             0
INCOME           43
MARITAL STATUS   28
SEX               6
EDUCATION        44
JOB              46
TRAVEL TIME      41
USE              53
MILES CLOCKED    25
CAR TYPE         10
CAR AGE          20
CITY              6
POSTAL CODE       3
dtype: int64
```

```
[16] # In this exercise, we will replace the numerical columns with median values and for categorical columns, we will drop the missing values.


     #Replacing the NULL Values in numerical columns using MEDIAN
     median_income = df_saniya['INCOME'].median()
     median_travel_time = df_saniya['TRAVEL TIME'].median()
     median_miles_clocked = df_saniya['MILES CLOCKED'].median()
     median_car_age = df_saniya['CAR AGE'].median()
     median_postal_code = df_saniya['POSTAL CODE'].median()


     df_saniya['INCOME'].replace(np.nan, median_income, inplace = True)
     df_saniya['TRAVEL TIME'].replace(np.nan, median_travel_time, inplace = True)
     df_saniya['MILES CLOCKED'].replace(np.nan, median_miles_clocked, inplace = True)
     df_saniya['CAR AGE'].replace(np.nan, median_car_age, inplace = True)
     df_saniya['POSTAL CODE'].replace(np.nan, median_postal_code, inplace = True)
```

```
[17] # Replacing the NULL values in  categorical columns using MODE


     mode_sex = df_saniya['SEX'].mode().values[0]
     mode_martial_status = df_saniya['MARITAL STATUS'].mode().values[0]
     mode_education = df_saniya['EDUCATION'].mode().values[0]
     mode_job = df_saniya['JOB'].mode().values[0]
     mode_use = df_saniya['USE'].mode().values[0]
     mode_city = df_saniya['CITY'].mode().values[0]
     mode_car_type = df_saniya['CAR TYPE'].mode().values[0]


     df_saniya['SEX']= df_saniya['SEX'].replace(np.nan, mode_sex)
     df_saniya['MARITAL STATUS']= df_saniya['MARITAL STATUS'].replace(np.nan, mode_martial_status)
     df_saniya['EDUCATION']= df_saniya['EDUCATION'].replace(np.nan, mode_education)
     df_saniya['JOB']= df_saniya['JOB'].replace(np.nan, mode_job)
     df_saniya['USE']= df_saniya['USE'].replace(np.nan, mode_use)
     df_saniya['CITY']= df_saniya['CITY'].replace(np.nan, mode_city)
     df_saniya['CAR TYPE']= df_saniya['CAR TYPE'].replace(np.nan, mode_car_type)
```

```
df_saniya.isnull().sum()
```

```
INDEX            0
INCOME           0
MARITAL STATUS   0
SEX              0
EDUCATION        0
JOB              0
TRAVEL TIME      0
USE              0
MILES CLOCKED    0
CAR TYPE         0
CAR AGE          0
CITY             0
POSTAL CODE      0
dtype: int64
```

```
# Handling Duplicate Records
duplicate = df_saniya.duplicated()
print(duplicate.sum())
df_saniya[duplicate]
```

14

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CAR TYPE	CAR AGE	CITY	POSTAL CODE
69	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
70	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
71	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
72	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
73	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
74	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
75	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
76	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
77	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
78	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
79	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
80	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0
81	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pickup	5.0	Los Angeles	90049.0

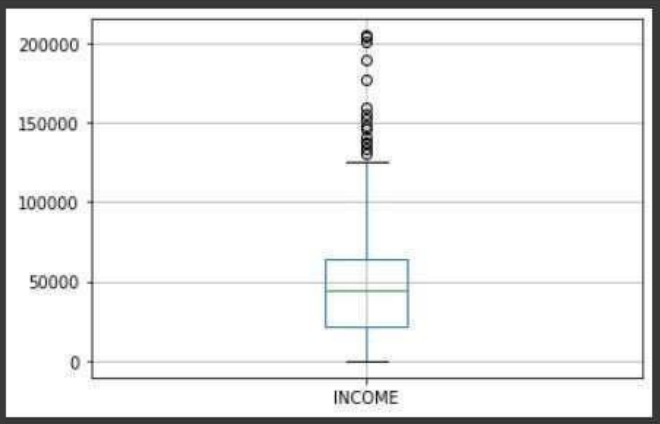
0s completed at 1:05 PM

```
[20] # code to drop the duplicate
     df_saniya.drop_duplicates(inplace=True)
```

```
[21] dup = df_saniya.duplicated()
     dup.sum()
```

```
0
```

```
[22] df_saniya.boxplot(column=['INCOME'])
     plt.show()
```

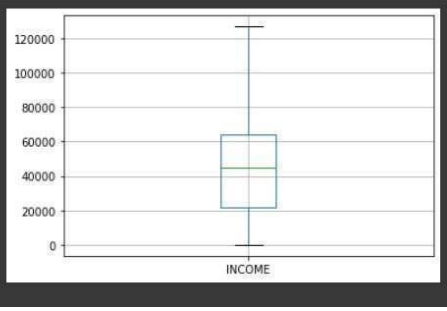



```
[31] # creating a user defined function called remove_outlier for getting the threshold value

     def remove_outlier(col):
       sorted(col)
       Q1, Q3 = col.quantile([0.25, 0.75])
       IQR = Q3-Q1
       lower_range = Q1-(1.5 * IQR)
       upper_range = Q3+(1.5 * IQR)
       return lower_range, upper_range
```

```
[24] lower_income, upper_income = remove_outlier(df_saniya['INCOME'])

     df_saniya['INCOME'] = np.where(df_saniya['INCOME'] > upper_income, upper_income, df_saniya['INCOME'])

     df_saniya['INCOME'] = np.where(df_saniya['INCOME'] < lower_income, lower_income, df_saniya['INCOME'])
```

After removing outlier, let us check it with boxplot

```
df_saniya.boxplot(column=['INCOME'])
plt.show()
```



```
[26] # if we need to find the correlation
     df_saniya.corr()
```

	INDEX	INCOME	TRAVEL TIME	MILES CLOCKED	CAR AGE	POSTAL CODE
INDEX	1.000000	-0.044968	0.016710	0.042880	-0.027206	-0.244783
INCOME	-0.044968	1.000000	0.062594	0.342164	0.267087	0.034204
TRAVEL TIME	0.016710	0.062594	1.000000	0.026915	0.140511	0.021390
MILES CLOCKED	0.042880	0.342164	0.026915	1.000000	0.127137	-0.111283
CAR AGE	-0.027206	0.267087	0.140511	0.127137	1.000000	-0.099449
POSTAL CODE	-0.244783	0.034204	0.021390	-0.111283	-0.099449	1.000000

```
# we use sklearn preprocessing using the function Standard Scaler

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

```
StandardScaler()
```

```
[28] df_saniya['INCOME'] = std_scale.fit_transform(df_saniya[['INCOME']])
     df_saniya['TRAVEL TIME'] = std_scale.fit_transform(df_saniya[['TRAVEL TIME']])
     df_saniya['CAR AGE'] = std_scale.fit_transform(df_saniya[['CAR AGE']])
     df_saniya['POSTAL CODE'] = std_scale.fit_transform(df_saniya[['POSTAL CODE']])
     df_saniya['MILES CLOCKED'] = std_scale.fit_transform(df_saniya[['MILES CLOCKED']])
```

```
[29] df_saniya.head()
```

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CAR TYPE	CAR AGE	CITY	POSTAL CODE
0	1	2.330448	No	F	Bachelors	Blue Collar	0.807947	Commercial	0.534077	Sports Car	0.137267	Texas	-0.277291
1	2	0.120293	No	M	High School	Blue Collar	-0.964473	Private	0.750925	Minivan	-1.052842	Texas	-0.277291
2	3	0.481177	No	F	Bachelors	Clerical	-0.043492	Private	-0.241513	SUV	-1.052842	Texas	-0.277291
3	4	0.900212	No	F	High School	Lawyer	-1.329803	Private	0.659849	Sports Car	0.930674	Texas	-0.277291
4	5	2.377524	No	M	High School	Blue Collar	-0.048799	Commercial	2.111280	Panel Truck	0.732322	Texas	-0.277291

```
[30] dummies = pd.get_dummies(df_saniya[['MARITAL STATUS', 'SEX', 'EDUCATION', 'JOB', 'USE', 'CAR TYPE', 'CITY']],
                              columns = ['MARITAL STATUS', 'SEX', 'EDUCATION', 'JOB', 'USE', 'CAR TYPE', 'CITY'],
                              prefix = ['married', 'sex', 'education', 'job', 'use', 'cartype', 'ciyt'], drop_first= True).head()
```

```
dummies.head()
```

	married_Yes	sex_M	education_High School	education_Masters	education_PhD	job_Clerical	job_Doctor	job_Home Maker	job_Lawyer	job_Manager	...	ciyt_Houston	ciyt_Las Vegas	ciyt_Los Angeles	ciyt_New Albany	ciyt_New York City	ciyt_Philadelph
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	
2	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	1	0	...	0	0	0	0	0	
4	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 28 columns

```
[32] columns = ["MARITAL STATUS", "SEX", "EDUCATION","JOB","USE", "CAR TYPE", "CITY"]
     df_saniya = pd.concat([df_saniya, dummies], axis=1)

     # drop original column 'fuel type' from df

     df_saniya.drop(columns, axis= 1, inplace=True )
```

```
[33] df_saniya.head()
```

	INDEX	INCOME	TRAVEL TIME	MILES CLOCKED	CAR AGE	POSTAL CODE	married_Yes	sex_M	education_High School	education_Masters	...	ciyt_Houston	ciyt_Las Vegas	ciyt_Los Angeles	ciyt_New Albany	ciyt_New York City	ciyt_Philadelphia	ciyt_San Francisco	ciyt
0	1	2.330448	0.807947	0.534077	0.137267	-0.277291	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	2	0.120293	-0.964473	0.750925	-1.052842	-0.277291	0.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	3	0.481177	-0.043492	-0.241513	-1.052842	-0.277291	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	4	0.900212	-1.329803	0.659849	0.930674	-0.277291	0.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	5	2.377524	-0.048799	2.111280	0.732322	-0.277291	0.0	1.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 34 columns

PRACTICAL 3

LINEAR REGRESSION

Aim : To Perform Linear Regression on the given dataset

```
[1]  # importing all the necessary libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as seabornInstance
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn import metrics
     %matplotlib inline
```

```
df_saniya = pd.read_csv('Summary of Weather.csv')
df_saniya
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (7,8,18,25) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

	STA	Date	Precip	WindGustSpd	MaxTemp	MinTemp	MeanTemp	Snowfall	PoorWeather	YR	...	FB	FTI	ITH	PGT	TSHDSBRSGF	SD3	RHX	RHN	RVG	WTE
0	10001	1942-7-1	1.016	NaN	25.555556	22.222222	23.888889	0.0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	10001	1942-7-2	0	NaN	28.888889	21.666667	25.555556	0.0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	10001	1942-7-3	2.54	NaN	26.111111	22.222222	24.444444	0.0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	10001	1942-7-4	2.54	NaN	26.666667	22.222222	24.444444	0.0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	10001	1942-7-5	0	NaN	26.666667	21.666667	24.444444	0.0	NaN	42	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
119035	82506	1945-12-27	0	NaN	28.333333	18.333333	23.333333	0.0	NaN	45	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
119036	82506	1945-12-28	9.906	NaN	29.444444	18.333333	23.888889	0.0	1.0	45	...	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
119037	82506	1945-12-29	0	NaN	28.333333	18.333333	23.333333	0.0	1.0	45	...	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN	NaN
119038	82506	1945-12-30	0	NaN	28.333333	18.333333	23.333333	0.0	NaN	45	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
119039	82506	1945-12-31	0	NaN	29.444444	17.222222	23.333333	0.0	NaN	45	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

119040 rows × 31 columns

```
df_saniya.shape
```

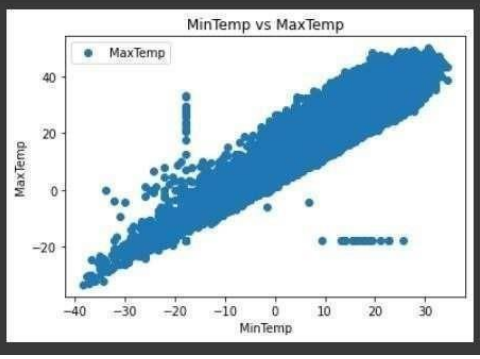
```
(119040, 31)
```

```
df_saniya.describe()
```

	STA	WindGustSpd	MaxTemp	MinTemp	MeanTemp	YR	MO	DA	DR	SPD	...	FT	FB	FTI	ITH	PGT	SD3	RHX	RHN	RVG	WT
count	119040.000000	532.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	119040.000000	533.000000	532.000000	...	0.0	0.0	0.0	0.0	525.000000	0.0	0.0	0.0	0.0	0
mean	29659.435795	37.774534	27.045111	17.789511	22.411631	43.805284	6.726016	15.797530	26.998124	20.396617	...	NaN	NaN	NaN	NaN	12.085333	NaN	NaN	NaN	NaN	Na
std	20953.209402	10.297808	8.717817	8.334572	8.297982	1.136718	3.425561	8.794541	15.221732	5.560371	...	NaN	NaN	NaN	NaN	5.731328	NaN	NaN	NaN	NaN	Na
min	10001.000000	18.520000	-33.333333	-38.333333	-35.555556	40.000000	1.000000	1.000000	2.000000	10.000000	...	NaN	NaN	NaN	NaN	0.000000	NaN	NaN	NaN	NaN	Na
25%	11801.000000	29.632000	25.555556	15.000000	20.555556	43.000000	4.000000	8.000000	11.000000	16.000000	...	NaN	NaN	NaN	NaN	8.500000	NaN	NaN	NaN	NaN	Na
50%	22508.000000	37.040000	29.444444	21.111111	25.555556	44.000000	7.000000	16.000000	32.000000	20.000000	...	NaN	NaN	NaN	NaN	11.600000	NaN	NaN	NaN	NaN	Na
75%	33501.000000	43.059000	31.666667	23.333333	27.222222	45.000000	10.000000	23.000000	34.000000	23.250000	...	NaN	NaN	NaN	NaN	15.000000	NaN	NaN	NaN	NaN	Na
max	82506.000000	75.932000	50.000000	34.444444	40.000000	45.000000	12.000000	31.000000	78.000000	41.000000	...	NaN	NaN	NaN	NaN	23.900000	NaN	NaN	NaN	NaN	Na

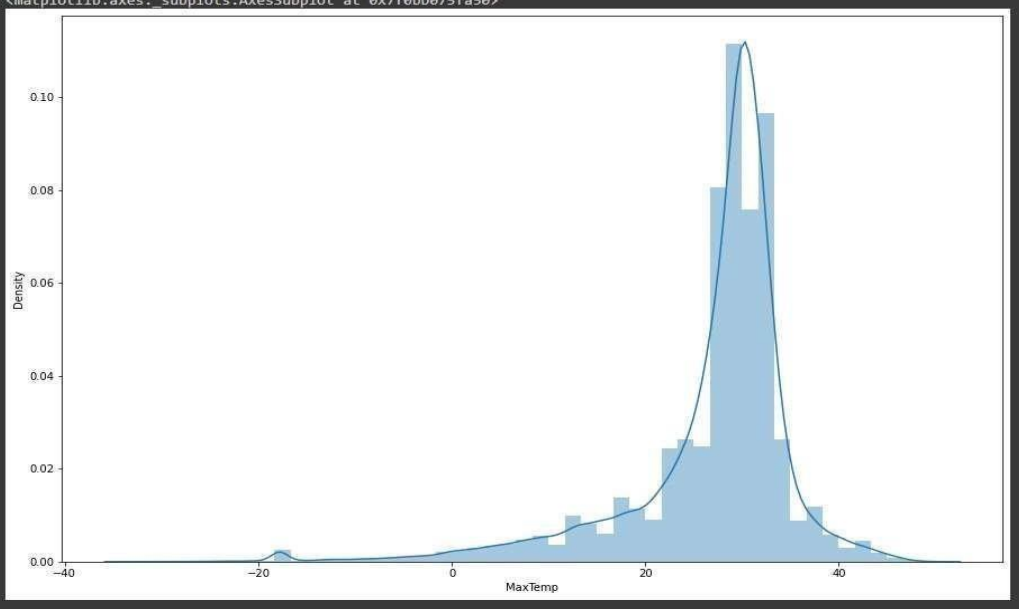
8 rows × 24 columns

```
# We have taken MinTemp and MaxTemp for doing our analysis. Below is a 2-D graph between MinTemp and MaxTemp.

df_saniya.plot(x='MinTemp', y='MaxTemp', style='o')
plt.title('MinTemp vs MaxTemp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.show()
```




```
[ ] plt.figure(figsize=(15,10))
    plt.tight_layout()
    seabornInstance.distplot(df_saniya['MaxTemp'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be r
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f0bb075fa50>
```




```
[ ] X = df_saniya['MinTemp'].values.reshape(-1,1)
    y = df_saniya['MaxTemp'].values.reshape(-1,1)
```

```
[8] X

array([[22.22222222],
       [21.66666667],
       [22.22222222],
       ...,
       [18.33333333],
       [18.33333333],
       [17.22222222]])

y

array([[25.55555556],
       [28.88888889],
       [26.11111111],
       ...,
       [28.33333333],
       [28.33333333],
       [29.44444444]])
```

```
[10] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

[11] X_train

array([[22.22222222],
       [17.77777778],
       [-9.44444444],
       ...,
       [ 3.33333333],
       [10.        ],
       [15.55555556]])

[12] X_test

array([[25.        ],
       [21.11111111],
       [17.22222222],
       ...,
       [23.88888889],
       [21.66666667],
       [22.77777778]])

y_train

array([[27.22222222],
       [32.77777778],
       [ 8.88888889],
       ...,
       [ 4.44444444],
       [14.44444444],
       [21.11111111]])
```

```
[14] y_test

array([[28.88888889],
       [31.11111111],
       [27.22222222],
       ...,
       [31.11111111],
       [31.11111111],
       [36.66666667]])
```

After splitting the data into training and testing sets, finally, the time is to train our algorithm. For that, we need to import LinearRegression class, instantiate it, and call the fit() method along with our training data.

```
[15] #training the algorithm

regressor = LinearRegression()
regressor.fit(X_train, y_train)

LinearRegression()
```

Tthe linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. To see the value of the intercept and slop calculated by the linear regression algorithm for our dataset, execute the following code.

```
#To retrieve the intercept:
print(regressor.intercept_)

#For retrieving the slope:
print(regressor.coef_)

[10.66185201]
[[0.92033997]]
```

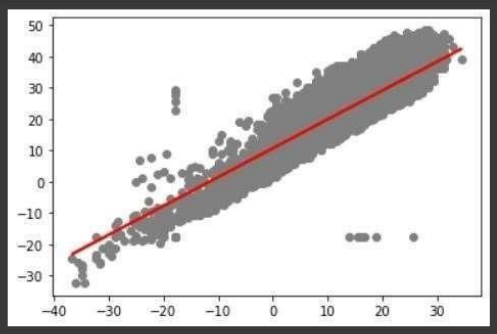
```
[ ]  y_pred = regressor.predict(X_test)
```

```
df_sani = pd.DataFrame({'Actual': y_test.flatten(), 'Predicted': y_pred.flatten()})
df_sani
```

	Actual	Predicted
0	28.888889	33.670351
1	31.111111	30.091251
2	27.222222	26.512151
3	28.888889	31.113851
4	23.333333	15.774852
...	...	...
23803	32.777778	32.136451
23804	32.222222	29.068651
23805	31.111111	32.647751
23806	31.111111	30.602551
23807	36.666667	31.625151

23808 rows × 2 columns

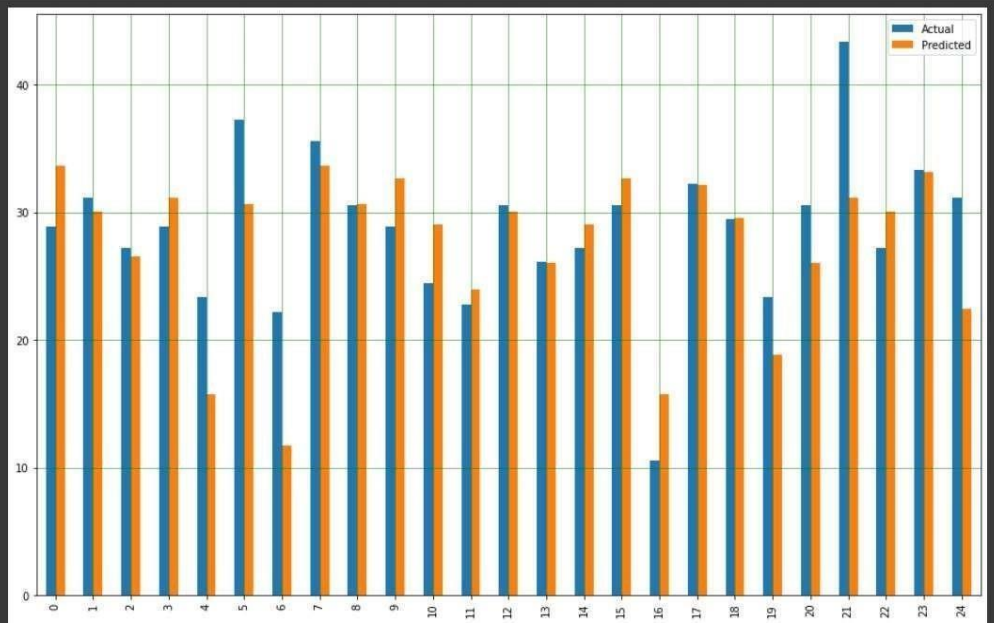
```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```



```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 3.19932917837853
Mean Squared Error: 17.631568097568447
Root Mean Squared Error: 4.198996082109204
```

```
df_sani1 = df_sani.head(25)
df_sani1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



PRACTICAL 4

LOGISTIC REGRESSION

AIM : To Perform Logistic Regression on the given dataset

```
[ ] #loading the dataset to a pandas Dataframe
    sonar_saniya = pd.read_csv('/content/sonar_data.csv', header=None)
```

```
[ ] sonar_saniya.head()
```

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	57	58	59	60
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078	R
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094	R

5 rows × 61 columns

```
[ ] # number of rows and columns
    sonar_saniya.shape
```

```
(208, 61)
```

```
[ ] sonar_saniya.describe()  #describe --> statistical measures of the data
```

	0	1	2	3	4	5	6	7	8	9	...	50	51	52	53	54
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	...	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570	0.121747	0.134799	0.178003	0.208259	...	0.016069	0.013420	0.010709	0.010941	0.009290
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152	0.118387	0.134416	...	0.012008	0.009634	0.007060	0.007301	0.007088
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500	0.007500	0.011300	...	0.000000	0.000800	0.000500	0.001000	0.000600
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425	0.097025	0.111275	...	0.008425	0.007275	0.005075	0.005375	0.004150
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100	0.152250	0.182400	...	0.013900	0.011400	0.009550	0.009300	0.007500
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600	0.233425	0.268700	...	0.020825	0.016725	0.014900	0.014500	0.012100
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000	0.682800	0.710600	...	0.100400	0.070900	0.039000	0.035200	0.044700

8 rows × 60 columns

```
sonar_saniya[60].value_counts()
```

```
M    111
R     97
Name: 60, dtype: int64
```

```
[ ] sonar_saniya.groupby(60).mean()
```

60	0	1	2	3	4	5	6	7	8	9	...	50	51	52	53	54	55
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213492	0.251022	...	0.019352	0.016014	0.011643	0.012185	0.009923	0.008914	0.0078
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137392	0.159325	...	0.012311	0.010453	0.009640	0.009518	0.008567	0.007430	0.0078

2 rows × 60 columns

```
# separating data and Labels
X = sonar_saniya.drop(columns=60, axis=1)
Y = sonar_saniya[60]
```

```
[ ] print(X)
    print(Y)
```

```
        0       1       2       3       4       5       6       7       8  \
0  0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601  0.3109
1  0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481  0.3337
2  0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771  0.5598
```

```
print(X)
print(Y)
```

```
          0       1       2       3       4       5       6       7       8   \
0    0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601  0.3109
1    0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481  0.3337
2    0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771  0.5598
3    0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276  0.0598
4    0.0762  0.0666  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467  0.3564
..      ...     ...     ...     ...     ...     ...     ...     ...     ...
203  0.0187  0.0346  0.0168  0.0177  0.0393  0.1630  0.2028  0.1694  0.2328
204  0.0323  0.0101  0.0298  0.0564  0.0760  0.0958  0.0990  0.1018  0.1030
205  0.0522  0.0437  0.0180  0.0292  0.0351  0.1171  0.1257  0.1178  0.1258
206  0.0303  0.0353  0.0490  0.0608  0.0167  0.1354  0.1465  0.1123  0.1945
207  0.0260  0.0363  0.0136  0.0272  0.0214  0.0338  0.0655  0.1400  0.1843

          9   ...      50      51      52      53      54      55      56  \
0    0.2111  ...  0.0232  0.0027  0.0065  0.0159  0.0072  0.0167  0.0180
1    0.2872  ...  0.0125  0.0084  0.0089  0.0048  0.0094  0.0191  0.0140
2    0.6194  ...  0.0033  0.0232  0.0166  0.0095  0.0180  0.0244  0.0316
3    0.1264  ...  0.0241  0.0121  0.0036  0.0150  0.0085  0.0073  0.0050
4    0.4459  ...  0.0156  0.0031  0.0054  0.0105  0.0110  0.0015  0.0072
..      ...  ...     ...     ...     ...     ...     ...     ...     ...
203  0.2684  ...  0.0203  0.0116  0.0098  0.0199  0.0033  0.0101  0.0065
204  0.2154  ...  0.0051  0.0061  0.0093  0.0135  0.0063  0.0063  0.0034
205  0.2529  ...  0.0155  0.0160  0.0029  0.0051  0.0062  0.0089  0.0140
206  0.2354  ...  0.0042  0.0086  0.0046  0.0126  0.0036  0.0035  0.0034
207  0.2354  ...  0.0181  0.0146  0.0129  0.0047  0.0039  0.0061  0.0040

         57      58      59
0    0.0084  0.0090  0.0032
1    0.0049  0.0052  0.0044
```

```
[10] from sklearn.model_selection import train_test_split

[11] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state=1)

[12] print(X.shape, X_train.shape, X_test.shape)

(208, 60) (145, 60) (63, 60)
```

```
print(X_train)
print(Y_train)
```

```
          0       1       2       3       4       5       6       7       8   \
194  0.0392  0.0108  0.0267  0.0257  0.0410  0.0491  0.1053  0.1690  0.2105
55   0.0201  0.0116  0.0123  0.0245  0.0547  0.0208  0.0891  0.0836  0.1335
93   0.0459  0.0437  0.0347  0.0456  0.0067  0.0890  0.1798  0.1741  0.1598
53   0.0293  0.0378  0.0257  0.0062  0.0130  0.0612  0.0895  0.1107  0.0973
97   0.0491  0.0279  0.0592  0.1270  0.1772  0.1908  0.2217  0.0768  0.1246
..      ...     ...     ...     ...     ...     ...     ...     ...     ...
203  0.0187  0.0346  0.0168  0.0177  0.0393  0.1630  0.2028  0.1694  0.2328
137  0.0430  0.0902  0.0833  0.0813  0.0165  0.0277  0.0569  0.2057  0.3887
72   0.0208  0.0186  0.0131  0.0211  0.0610  0.0613  0.0612  0.0506  0.0989
140  0.0412  0.1135  0.0518  0.0232  0.0646  0.1124  0.1787  0.2407  0.2682
37   0.0333  0.0221  0.0270  0.0481  0.0679  0.0981  0.0843  0.1172  0.0759

          9   ...      50      51      52      53      54      55      56  \
194  0.2471  ...  0.0089  0.0083  0.0080  0.0026  0.0079  0.0042  0.0071
55   0.1199  ...  0.0032  0.0076  0.0045  0.0056  0.0075  0.0037  0.0045
93   0.1408  ...  0.0121  0.0067  0.0032  0.0109  0.0164  0.0151  0.0070
53   0.0751  ...  0.0076  0.0065  0.0072  0.0108  0.0051  0.0102  0.0041
```

```
[14] from sklearn.linear_model import LogisticRegression

[15] lr = LogisticRegression()

[16] #training the Logistic Regression model with training data
     lr.fit(X_train, Y_train)

     LogisticRegression()
```

Model Evaluation

```
[17] from sklearn.metrics import accuracy_score

[18] #accuracy on training data
     X_train_prediction = lr.predict(X_train)
     training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

     print('Accuracy on training data : ', training_data_accuracy)

     Accuracy on training data :  0.8413793103448276
```

```
[20] #accuracy on test data
     X_test_prediction = lr.predict(X_test)
     test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

[21] print('Accuracy on test data : ', test_data_accuracy)

     Accuracy on test data :  0.7777777777777778
```

Making a Predictive System

```
input_data = (0.0307,0.0523,0.0653,0.0521,0.0611,0.0577,0.0665,0.0664,0.1460,0.2792,0.3877,0.4992,0.4981,0.4972,0.5607,0.7339,0.8230,0.9173,0.9975,0.9911,0.8240,0.645

# changing the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the np array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = lr.predict(input_data_reshaped)
print(prediction)

if (prediction[0]=='R'):
  print('The object is a Rock')
else:
  print('The object is a mine')

['M']
The object is a mine
```

PRACTICAL 5

DECISION TREE

AIM : To Implement Decision Tree on the given dataset

```
[1] # Load libraries
    import pandas as pd
    from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
    from sklearn.model_selection import train_test_split # Import train_test_split function
    from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

    #col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
    # load dataset
    df_saniya = pd.read_csv("/content/diabetes.csv")

    df_saniya.all
```

```
<bound method NDFrame._add_numeric_operations.<locals>.all of       Pregnancies  Glucose  BloodPressu
0               6      148             72             35       0  33.6
1               1       85             66             29       0  26.6
2               8      183             64              0       0  23.3
3               1       89             66             23      94  28.1
4               0      137             40             35     168  43.1
...           ...      ...            ...            ...     ...   ...
2066            1      173             74              0       0  36.8
2067            1      109             38             18     120  23.1
2068            1      108             88             19       0  27.1
2069            6       96              0              0       0  23.7
2070            1      124             74             36       0  27.8
```

```
[4] df_saniya.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2071 entries, 0 to 2070
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               2071 non-null   int64
 1   Glucose                   2071 non-null   int64
 2   BloodPressure             2071 non-null   int64
 3   SkinThickness             2071 non-null   int64
 4   Insulin                   2071 non-null   int64
 5   BMI                       2071 non-null   float64
 6   DiabetesPedigreeFunction  2071 non-null   float64
 7   Age                       2071 non-null   int64
 8   Outcome                   2071 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 145.7 KB
```

```
[5] #split dataset in features and target variable
    feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age','Glucose','BloodPressure','DiabetesPedigreeFunction']
    X = df_saniya[feature_cols] # Features
    y = df_saniya.Outcome # Target variable
```

```
X
```

	Pregnancies	Insulin	BMI	Age	Glucose	BloodPressure	DiabetesPedigreeFunction
0	6	0	33.6	50	148	72	0.627
1	1	0	26.6	31	85	66	0.351
2	8	0	23.3	32	183	64	0.672
3	1	94	28.1	21	89	66	0.167
4	0	168	43.1	33	137	40	2.288
...	...	...	...	...	...	...	...
2066	1	0	36.8	38	173	74	0.088
2067	1	120	23.1	26	109	38	0.407
2068	1	0	27.1	24	108	88	0.400
2069	6	0	23.7	28	96	0	0.190

✓ 0s completed at 7:47 PM

```
[7] y
```

```
0       1
1       0
2       1
3       0
4       1
       ..
2066    1
2067    0
2068    0
2069    0
2070    0
Name: Outcome, Length: 2071, dtype: int64
```

```
[8]  # Split dataset into training set and test set
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

Building Decision Tree Model

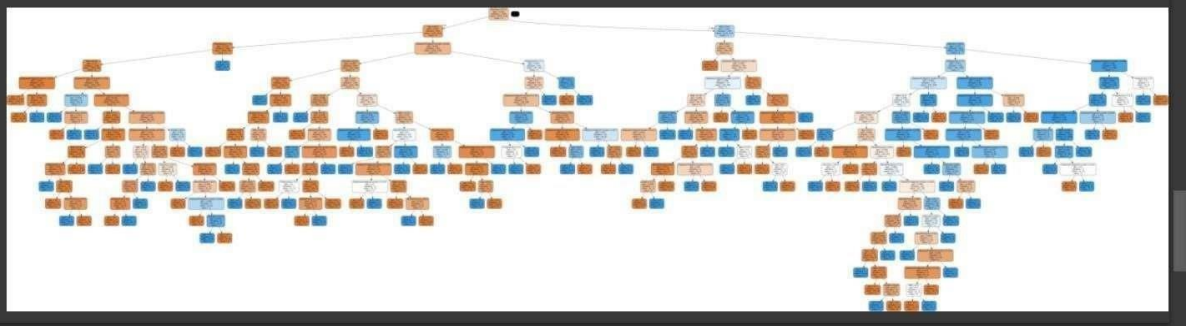
Let's create a Decision Tree Model using Scikit-learn.

```
[9]  # Create Decision Tree classifer object
     clasf = DecisionTreeClassifier()

     # Train Decision Tree Classifer
     clasf = clasf.fit(X_train,y_train)

     #Predict the response for test dataset
     y_pred = clasf.predict(X_test)
```

```
[10] # Model Accuracy, how often is the classifier correct?
     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

     Accuracy: 0.9501607717041801
```

```
[12] pip install graphviz

     Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (0.10.1)

[13] pip install pydotplus

     Requirement already satisfied: pydotplus in /usr/local/lib/python3.7/dist-packages (2.0.2)
     Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from pydotplus) (3.0.7)
```

```
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clasf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```



PRACTICAL 6

K-MEANS CLUSTERING

AIM : To Implement K-Means Clustering on the given dataset.

```
[1] import seaborn as sns
    import matplotlib.pyplot as plt
    %matplotlib inline
```

```
[2] from sklearn.datasets import make_blobs
```

In this dataset we will have 300 sample points having 2 features and there are 5 center to this blob.

```
[3] data = make_blobs(n_samples=300, n_features=2, centers=5, cluster_std=1.8,random_state=101)
    data[0].shape
```

```
(300, 2)
```

In the unsupervised learning we dont know the labels but since we are creating the dataset so we will have the label to compare the label given by the Algorithm versus the actual label.

```
[4] data[1]
```

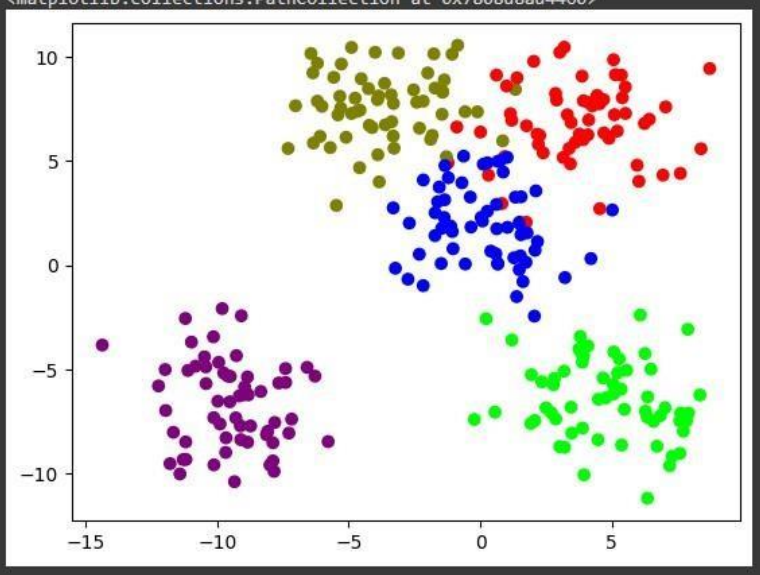
```
array([4, 0, 2, 4, 4, 0, 2, 2, 0, 3, 1, 0, 0, 2, 1, 1, 4, 4, 0, 1, 1, 4,
       4, 0, 0, 2, 1, 2, 2, 4, 0, 0, 4, 0, 0, 2, 2, 3, 3, 4, 4, 4, 2, 1,
       2, 1, 0, 4, 4, 4, 1, 1, 0, 4, 0, 3, 1, 0, 2, 3, 1, 1, 3, 1, 4, 4,
       4, 1, 1, 1, 2, 2, 1, 1, 4, 1, 3, 3, 3, 1, 3, 2, 3, 0, 2, 1, 1, 1,
       2, 0, 3, 2, 1, 2, 1, 4, 3, 0, 2, 3, 0, 4, 0, 0, 3, 3, 3, 4, 2, 2,
       2, 1, 0, 0, 0, 3, 4, 4, 1, 1, 2, 0, 3, 4, 1, 1, 1, 4, 4, 2, 3, 3,
       3, 3, 3, 3, 4, 1, 0, 0, 0, 4, 3, 4, 2, 3, 2, 1, 3, 3, 4, 4, 2, 3,
       1, 2, 4, 2, 1, 4, 2, 4, 4, 1, 3, 1, 0, 3, 0, 0, 0, 3, 1, 3, 2, 1,
       3, 0, 0, 4, 1, 2, 4, 2, 2, 0, 0, 0, 3, 4, 2, 0, 2, 3, 2, 3, 4, 0,
       2, 0, 3, 4, 2, 0, 3, 2, 0, 4, 3, 4, 4, 1, 2, 4, 1, 3, 1, 0, 3, 1,
       0, 2, 3, 0, 1, 3, 2, 4, 4, 3, 1, 3, 3, 4, 0, 0, 0, 2, 0, 2, 4, 1,
       1, 3, 4, 0, 2, 4, 2, 4, 1, 2, 3, 0, 3, 2, 1, 0, 0, 0, 2, 3, 2, 0,
       4, 2, 2, 1, 4, 1, 1, 1, 3, 3, 0, 4, 3, 3, 4, 1, 4, 3, 2, 0, 2, 4,
       1, 2, 4, 3, 2, 2, 1, 0, 3, 1, 3, 2, 1, 0])
```

Data Visualize

Lets plot out this to get a better idea what actually we are doing here.

```
[5] plt.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='brg')
```

```
<matplotlib.collections.PathCollection at 0x7808d8ad4460>
```



Using K means Clustering

Here we will import the K means algorithm from scikit learn and we will define number of clusters we want to have for this dataset.

```
[6] from sklearn.cluster import KMeans
```

Here we are doing it for n=5 (Number of clusters will be 5)

```
[7] kmeans = KMeans(n_clusters=5)
```

```
[8] kmeans.fit(data[0])
```

```
KMeans
KMeans(n_clusters=5)
```

```
[9] kmeans.cluster_centers_
```

```
array([[-3.69907513,  7.57237976],
       [ 4.83240966, -6.48200379],
       [-9.46255618, -6.63414105],
       [ 0.09806182,  1.98206252],
       [ 4.07267325,  7.01145994]])
```

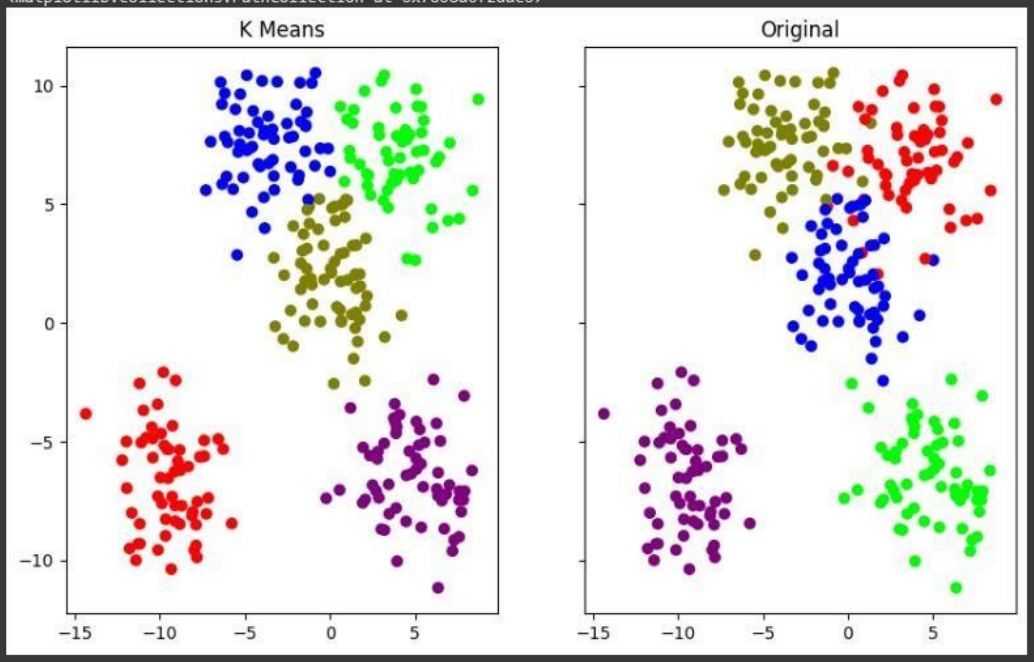
```
[10] kmeans.labels_
```

```
array([1, 3, 4, 1, 3, 3, 4, 4, 3, 4, 2, 3, 3, 4, 2, 2, 1, 1, 4, 2, 2, 1,
       1, 3, 3, 4, 2, 4, 4, 1, 3, 3, 1, 3, 3, 4, 0, 0, 0, 1, 1, 1, 4, 2,
       3, 2, 3, 1, 1, 1, 2, 2, 3, 1, 3, 0, 2, 3, 4, 0, 2, 2, 0, 2, 1, 1,
       1, 2, 2, 2, 4, 4, 2, 2, 1, 2, 0, 0, 0, 2, 0, 3, 0, 3, 4, 2, 2, 2,
       4, 3, 0, 4, 2, 3, 2, 1, 0, 3, 4, 0, 3, 1, 3, 3, 0, 0, 0, 1, 4, 3,
       0, 2, 3, 3, 3, 0, 1, 1, 2, 2, 4, 3, 0, 1, 2, 2, 2, 1, 1, 4, 0, 0,
       0, 0, 0, 0, 1, 2, 3, 3, 3, 1, 0, 1, 4, 0, 4, 2, 0, 0, 1, 1, 4, 0,
       2, 4, 1, 4, 2, 1, 4, 1, 1, 2, 0, 2, 3, 0, 3, 3, 3, 0, 2, 0, 4, 2,
       0, 3, 3, 1, 2, 4, 1, 4, 4, 3, 3, 3, 0, 1, 4, 3, 4, 0, 4, 4, 1, 3,
       4, 3, 0, 1, 4, 3, 0, 4, 3, 1, 0, 1, 1, 2, 3, 1, 2, 0, 2, 3, 0, 2,
       3, 4, 0, 3, 2, 0, 4, 1, 1, 0, 2, 0, 0, 1, 3, 3, 3, 4, 3, 4, 1, 2,
       2, 0, 1, 3, 4, 1, 4, 1, 2, 4, 0, 3, 0, 4, 2, 3, 3, 3, 4, 0, 4, 3,
       1, 4, 4, 2, 1, 2, 2, 2, 0, 0, 3, 1, 0, 0, 1, 2, 1, 0, 4, 3, 4, 1,
       2, 4, 1, 0, 4, 4, 2, 3, 0, 2, 0, 4, 2, 3], dtype=int32)
```

So, now we will visualize both the actual labels of the data and the labels which are predicted by the algorithm by plotting them on graph.

```
[11] f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))

     ax1.set_title('K Means')
     ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels_,cmap='brg')

     ax2.set_title("Original")
     ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='brg')
```

```
<matplotlib.collections.PathCollection at 0x7808d6f2dae0>
```



So we can see that the algorithm worked pretty well to make clusters for the above dataset. So you can try with other values of "K" and analyze which give a better result.

PRACTICAL 7

SUPPORT VECTOR MACHINE (SVM)

Aim: To implement SVM on given dataset

```
# print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
```

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

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109) # 70% training and 30% test
```

Generating Model

Let's build support vector machine model. First, import the SVM module and create support vector classifier object by passing argument kernel as the linear kernel in SVC() function.

Then, fit your model on train set using fit() and perform prediction on the test set using predict().

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
[11] #Import scikit-learn metrics module for accuracy calculation
     from sklearn import metrics

     # Model Accuracy: how often is the classifier correct?
     print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
```

Well, you got a classification rate of 96.49%, considered as very good accuracy.

For further evaluation, you can also check precision and recall of model.

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

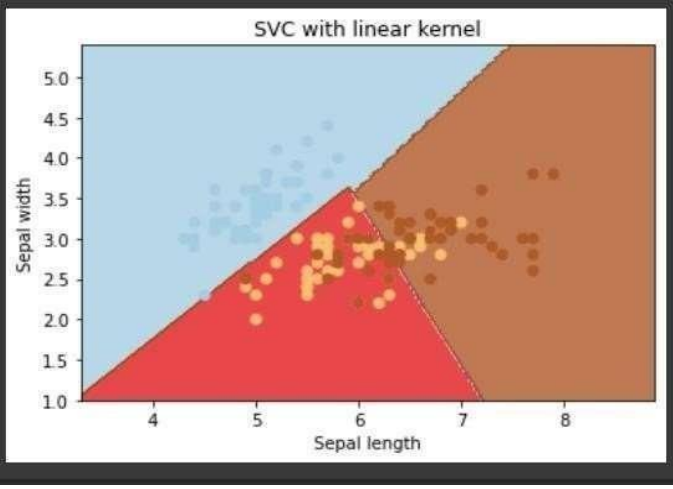
```
[13] import numpy as np
     import matplotlib.pyplot as plt
     from sklearn import svm, datasets
```

```
[14] # 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
```

```
[15] # 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').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()
```

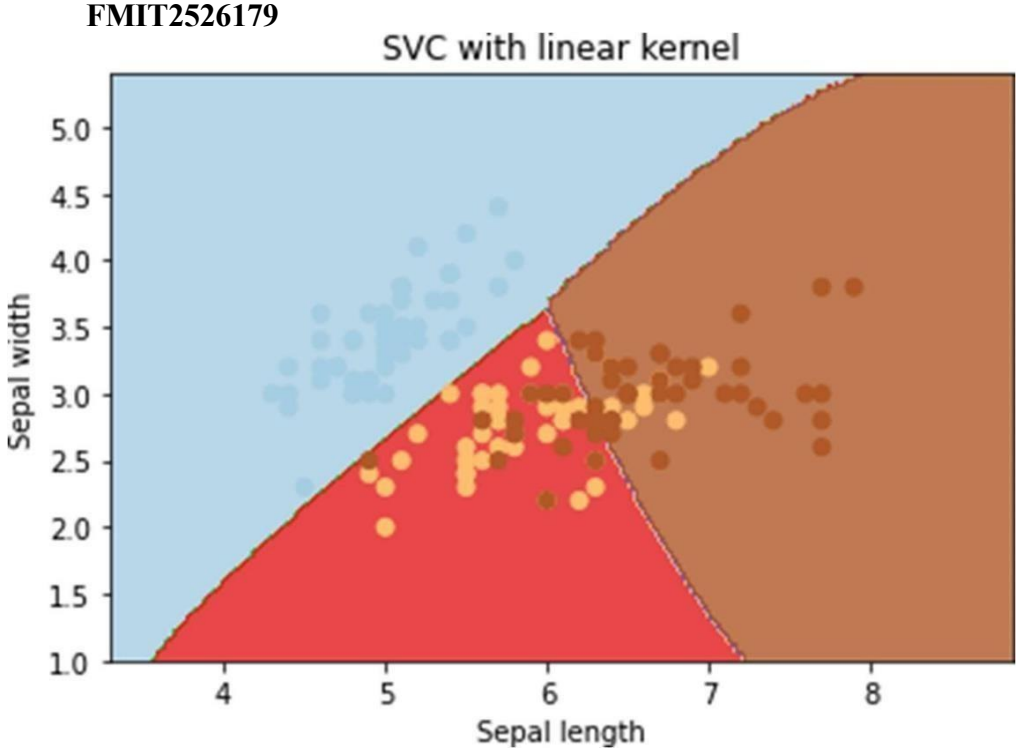



```
[18] svc = svm.SVC(kernel='rbf').fit(X, y)
```

```
[19] # 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))
```

curvish lines

```
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()
```

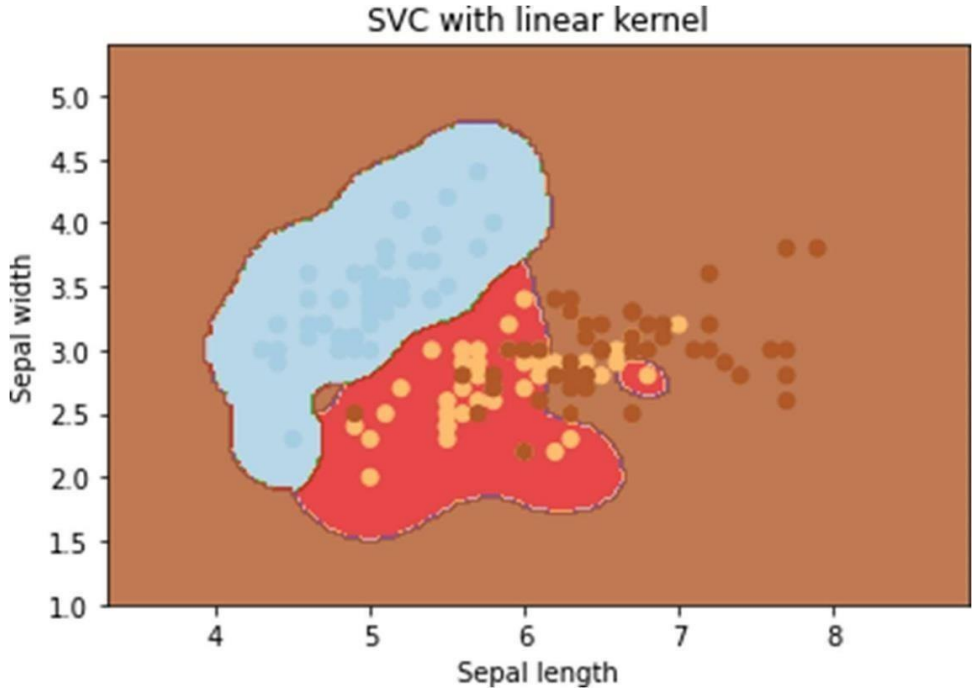



```
[21] svc = svm.SVC(kernel='rbf', C=1, gamma=10).fit(X, y)
```

low gamma value loosely coupled

```
# 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()
```


SVC with linear kernel
Sepal width
Sepal length
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
4
5
6
7
8

PRACTICAL 8

RANDOM FOREST REGRESSOR

Aim :- To Implement Random Forest Regressor on the given dataset

```
[3]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
```

```
[4]  df_saniya = pd.read_csv('audi.csv')

     X = df_saniya.iloc[:,[0,1,3,4,5,6,7,8]].values
     Y = df_saniya.iloc[:,[2]].values
```

```
[5]  print(X)
```

```
[[' A1' 2017 'Manual' ... 150 55.4 1.4]
 [' A6' 2016 'Automatic' ... 20 64.2 2.0]
 [' A1' 2016 'Manual' ... 30 55.4 1.4]
 ...
 [' A3' 2020 'Manual' ... 150 49.6 1.0]
 [' Q3' 2017 'Automatic' ... 150 47.9 1.4]
 [' Q3' 2016 'Manual' ... 150 47.9 1.4]]
```

```
print(Y)
```

```
[[12500]
 [16500]
 [11000]
 ...
```

```
[7]  from sklearn.preprocessing import LabelEncoder
     le1 = LabelEncoder()
     X[:,0] = le1.fit_transform(X[:,0])
     le2 = LabelEncoder()
     X[:,-4] = le2.fit_transform(X[:,-4])
```

```
[8]  from sklearn.preprocessing import OneHotEncoder
     from sklearn.compose import ColumnTransformer
     ct = ColumnTransformer(transformers = [('encoder',OneHotEncoder(),[2])],remainder='passthrough')
     X = ct.fit_transform(X)
```

```
print(X)
```

```
[[0.0 1.0 0.0 ... 150 55.4 1.4]
 [1.0 0.0 0.0 ... 20 64.2 2.0]
 [0.0 1.0 0.0 ... 30 55.4 1.4]
 ...
 [0.0 1.0 0.0 ... 150 49.6 1.0]
 [1.0 0.0 0.0 ... 150 47.9 1.4]
 [0.0 1.0 0.0 ... 150 47.9 1.4]]
```

```
[10] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X = sc.fit_transform(X)
```

```
print(X)
```

```
[[-0.58326752  1.2007284  -0.71233307 ...  0.35714729  0.35755001
  -0.88021837]
 [ 1.71447913 -0.83282781 -0.71233307 ... -1.57832278  1.03713001
   0.11492465]
 [-0.58326752  1.2007284  -0.71233307 ... -1.42944047  0.35755001
  -0.88021837]
 ...
 [-0.58326752  1.2007284  -0.71233307 ...  0.35714729 -0.09035499
  -1.54364705]
 [ 1.71447913 -0.83282781 -0.71233307 ...  0.35714729 -0.22163749
  -0.88021837]
 [-0.58326752  1.2007284  -0.71233307 ...  0.35714729 -0.22163749
  -0.88021837]]
```

```
[12] 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,random_state=0)
```

Training Model

```
[13] from sklearn.ensemble import RandomForestRegressor
     regression = RandomForestRegressor(random_state=0)
     regression.fit(X_train,Y_train)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the sh
  This is separate from the ipykernel package so we can avoid doing imports until
RandomForestRegressor(random_state=0)
```

```
y_pred = regression.predict(X_test)
```

```
[15] print(np.concatenate((y_pred.reshape(len(y_pred),1),Y_test.reshape(len(Y_test),1)),1))
```

```
[[14337.15 14998.  ]
 [23450.35 21950.  ]
 [27330.07 28990.  ]
 ...
 [46275.18 45995.  ]
 [31359.   30500.  ]
 [ 9929.62  8400.  ]]
```

Calculating Accuracy

```
[16] from sklearn.metrics import r2_score,mean_absolute_error
     r2_score(Y_test, y_pred)
```

```
0.9536134841307546
```

```
mean_absolute_error(Y_test,y_pred)
```

```
1538.730980670462
```

```
[18] print(y_pred)

     [14337.15 23450.35 27330.07 ... 46275.18 31359.    9929.62]
```

Reshape to 2D

```
[19] print(Y_test)

     [[14998]
      [21950]
      [28990]
      ...
      [45995]
      [30500]
      [ 8400]]

     y_pred = np.reshape(y_pred,(-1,1))
```

```
[21] mydata = np.concatenate((Y_test,y_pred),axis=1)
     dataframe = pd.DataFrame(mydata,columns=['Real Price','Predicted Price'])

     print(dataframe)

          Real Price  Predicted Price
     0        14998.0         14337.15
     1        21950.0         23450.35
     2        28990.0         27330.07
     3        25489.0         27200.98
     4        30950.0         32250.05
     ...          ...              ...
     2129     23700.0         39147.77
     2130     18000.0         16679.95
     2131     45995.0         46275.18
     2132     30500.0         31359.00
     2133      8400.0          9929.62

     [2134 rows x 2 columns]
```