

Week-1

AIM :Extract data from different file formats and display the summary statistics.

Description:

PANDAS:

Pandas is a data manipulation and analysis library for Python. It is widely used for data analysis and manipulation. It provides data structures and functions needed to work with structured data.

Pandas provides two main data structures they are

1)series :Series is a one-dimensional labeled array

2)DataFrame :Dataframe is a two-dimensional labeled data structure with columns of potentially different types.

Reading and Writing Data:

-Pandas supports reading data from various sources such as CSV, Excel, SQL, and more.

-You can also write DataFrames to these file formats using the `to_csv`, `to_excel`, and other similar functions.

Data Cleaning:

-Pandas provides functions for cleaning and preprocessing data, including handling missing values, renaming columns, and changing data types.

Data Manipulation:

-Pandas provides functions for data manipulation, including merging, grouping, and pivoting data.

-You can also apply various mathematical and statistical functions to the data.

Data Visualization:

-Pandas integrates well with data visualization libraries such as Matplotlib and Seaborn.

-You can easily create bar plots, line plots, histograms, and more.

Performance:

-Pandas is optimized for performance, but it can be slow when dealing with large datasets.

-To improve performance, you can use vectorized operations, use the optimized data structures such as Categorical and make use of the `dtype` argument to specify data types.

-Pandas is a powerful tool for data analysis and manipulation in Python.

It provides a rich set of functions and data structures that make it easy to work with structured data in Python.

Extracting data from different file formats and displaying summary statistics requires different tools and techniques depending on the format of the data. Here are some common file formats and ways to extract data and generate summary statistics:

CSV file: CSV files are commonly used for tabular data. To extract data from a CSV file, you can use a library like Pandas in Python or `read.csv()` in R. display the summary statistics using the `describe()` method.

Excel file: Excel files are often used for data storage and analysis. To extract data from an Excel file, you can use a library like Pandas in Python or `read.xlsx()` in R. display the summary statistics using the `describe()` method.

JSON file: JSON files are often used for storing and exchanging data in web applications. To extract data from a JSON file, you can use a library like json in Python or jsonlite in R. method.

XML file: XML files are often used for storing structured data in web applications. To extract data from an XML file, you can use a library like xml.etree.ElementTree in Python or XML in R. The code will load the XML file named 'data.xml', parse the data, and display the element tags and attributes.

Text file: Text files are often used for storing unstructured data, such as logs or plain text. To extract data from a text file, you can use a library like open() in Python or readLines() in R.

Zip file: Zipped (compressed) files take up less storage space and can be transferred to other computers more quickly than uncompressed files. To extract data from a zip file, you can use a library like zipfile in Python or unzip in R.

#PROGRAM

For csv extraction:

```
import pandas as pd

df = pd.read_csv('covid_19_india.csv')

print(df.describe())
```

output:

	Sno	Cured	Deaths	Confirmed
count	18110.000000	1.811000e+04	18110.000000	1.811000e+04
mean	9055.500000	2.786375e+05	4052.402264	3.010314e+05
std	5228.051023	6.148909e+05	10919.076411	6.561489e+05
min	1.000000	0.000000e+00	0.000000	0.000000e+00
25%	4528.250000	3.360250e+03	32.000000	4.376750e+03
50%	9055.500000	3.336400e+04	588.000000	3.977350e+04
75%	13582.750000	2.788698e+05	3643.750000	3.001498e+05
max	18110.000000	6.159676e+06	134201.000	6.363442e+06

Sno	Date	Time	State/UnionTerritory	ConfirmedIndianNational	ConfirmedForeignNational	Cured	Deaths	Confirmed
0	1	2020-01-30	6:00 PM	Kerala	1	0	0	0
1	2	2020-01-31	6:00 PM	Kerala	1	0	0	0
2	3	2020-02-01	6:00 PM	Kerala	2	0	0	0

3	4	2020-02-02	6:00 PM	Kerala	3	0	0	0	3
4	5	2020-02-03	6:00 PM	Kerala	3	0	0	0	3

For Excel extraction:

```
import pandas as pd
df = pd.read_excel('Demo.xlsx')
print(df.describe())
df.head()
```

output:

```
Year    Sales    Rating
count  75.000000  75.000000  75.000000
mean   2016.000000 10644.000000  0.542400
std     0.821995 11960.184397  0.286006
min     2015.000000  300.000000  0.050000
25%     2015.000000 2350.000000  0.350000
50%     2016.000000 6200.000000  0.480000
75%     2017.000000 16400.000000 0.865000
max     2017.000000 63700.000000 1.000000

Year    Category    Product    Sales    Rating
0       2017    Components    Chains  20000  0.75
1       2015    Clothing       Socks   3700  0.22
2       2017    Clothing       Bib-Shorts  4000  0.22
3       2015    Clothing       Shorts  13300  0.56
4       2017    Clothing       Tights  36000  1.00
```

For JSON file:

```
import json

with open('data.json') as f:

    data = json.load(f)

print(data)
```

output:

```
{['_id': '5c0a28d7a647437fd3d3a6aa', 'index': 0, 'guid': '832f1af4-fb18-4ba8-b032-f4cbf343dbe9', 'isActive': False, 'balance': '$3,806.93', 'picture': 'http://placeholder.it/32x32', 'age': 20, 'eyeColor': 'green', 'name': 'Grace Berry', 'gender': 'female', 'company': 'RODEMCO', 'email': 'graceberry@rodemco.com', 'phone': '+1 (814) 555-3298', 'address': '698 Kansas Place, Bethpage, Louisiana, 7695', 'about': 'Nostrud consectetur elit occaecat dolore incididunt est pariatur amet occaecat excepteur tempor do. Aute minim dolore aute voluptate ea quis nostrud aliqua est incididunt. Dolor sit aliqua ipsum sit eiusmod minim ullamco. Sint aliquip mollit velit laborum. Id ea incididunt aliquip consectetur sint dolore cupidatat. Esse culpa sint ut Lorem ullamco velit dolore sit sit aute tempor. Est duis consequat tempor amet in do nostrud irure veniam aute velit dolore.\r\n', 'registered' ]}
```

For Xml file:

```
import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')

root = tree.getroot()

for child in root:

    print(child.tag, child.attrib)
```

output:

```
book {'id': 'bk101'}
book {'id': 'bk102'}
book {'id': 'bk103'}
book {'id': 'bk104'}
book {'id': 'bk105'}
book {'id': 'bk106'}
book {'id': 'bk107'}
book {'id': 'bk108'}
```

```
book {'id': 'bk109'}
```

```
book {'id': 'bk110'}
```

```
book {'id': 'bk111'}
```

```
book {'id': 'bk112'}
```

For txt file:

```
with open('/content/sample1.txt') as f:
```

```
    lines = f.readlines()
```

```
print(lines)
```

output:

```
['Utilitatis causa amicitia est quaesita.\n', 'Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Collatio igitur ista te nihil iuvat. Honesta oratio, Socratica, Platonis etiam.
Primum in nostrane potestate est, quid meminerimus? Duo Reges: constructio interrete.
Quid, si etiam iucunda memoria est praeteritorum malorum? Si quidem, inquit, tollerem,
sed relinquo. An nisi populari fama?\n', '\n', 'Quamquam id quidem licebit iis existimare, qui
legerint. Summum a vobis bonum voluptas dicitur. At hoc in eo M. Refert tamen, quo modo.
Quid sequatur, quid repugnet, vident. Iam id ipsum absurdum, maximum malum neglegi.']
```

For zip file:

```
import zipfile
```

```
import pandas as pd
```

```
with zipfile.ZipFile('archive.zip', 'r') as zip_ref:
```

```
    zip_ref.extractall()
```

```
df = pd.read_csv('archive.zip')
```

```
print(df.describe())
```

output:

```
the      to      ect      and      for \
```

```

count 5172.000000 5172.000000 5172.000000 5172.000000 5172.000000
mean   6.640565   6.188128   5.143852   3.075599   3.124710
std    11.745009   9.534576   14.101142   6.045970   4.680522
min     0.000000   0.000000   1.000000   0.000000   0.000000
25%     0.000000   1.000000   1.000000   0.000000   1.000000
50%     3.000000   3.000000   1.000000   1.000000   2.000000
75%     8.000000   7.000000   4.000000   3.000000   4.000000
max    210.000000  132.000000  344.000000  89.000000  47.000000

```

of a you hou in ... \

```

count 5172.000000 5172.000000 5172.000000 5172.000000 5172.000000 ...
mean   2.627030  55.517401   2.466551   2.024362  10.600155 ...
std    6.229845  87.574172   4.314444   6.967878  19.281892 ...
min     0.000000   0.000000   0.000000   0.000000   0.000000 ...
25%     0.000000  12.000000   0.000000   0.000000   1.000000 ...
50%     1.000000  28.000000   1.000000   0.000000   5.000000 ...
75%     2.000000  62.250000   3.000000   1.000000  12.000000 ...
max    77.000000 1898.000000  70.000000  167.000000 223.000000 ...

```

connevey jay valued lay infrastructure \

```

count 5172.000000 5172.000000 5172.000000 5172.000000 5172.000000
mean   0.005027   0.012568   0.010634   0.098028   0.004254
std    0.105788   0.199682   0.116693   0.569532   0.096252
min     0.000000   0.000000   0.000000   0.000000   0.000000
25%     0.000000   0.000000   0.000000   0.000000   0.000000
50%     0.000000   0.000000   0.000000   0.000000   0.000000
75%     0.000000   0.000000   0.000000   0.000000   0.000000
max     4.000000   7.000000   2.000000  12.000000   3.000000

```

	military	allowing	ff	dry	Prediction
count	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000
mean	0.006574	0.004060	0.914733	0.006961	0.290023
std	0.138908	0.072145	2.780203	0.098086	0.453817
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	1.000000	0.000000	1.000000
max	4.000000	3.000000	114.000000	4.000000	1.000000

[8 rows x 3001 columns]

Week-2

Aim: Write a program that extracts the words(Features) used in a sentence.

Description:

NLTK (Natural Language Toolkit) Library is a suite that contains libraries and programs for statistical language processing. It is one of the most powerful NLP libraries, which contains packages to make machines understand human language and reply to it with an appropriate response.

NLTK is used for tokenization of words, POS, Tokenization, Stemming, Lemmatization, Punctuation, Character count, word count, WordNet, Word Embedding, seq2seq model, etc.

Stop words are words in any language or corpus that occur frequently. For some NLP tasks, they do not provide any additional or valuable information to the text containing them. Words like a, they, the, is, an, etc. are usually considered stop words.

Natural Language Tool kit comes with a stopwords corpus containing word lists for many languages

Word_tokenize is a function in Python that splits a given sentence into words using the NLTK library..In Natural Language Processing, tokenization divides a string into a list of tokens. Tokens come in handy when finding valuable patterns and helping to replace sensitive data components with nonsensitive ones.

Samp.txt file :

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

IBM has a rich history with machine learning. One of its own, Arthur Samuel, is credited for coining the term, “machine learning” with his research (PDF, 481 KB) (link resides outside IBM) around the game of checkers. Robert Nealey, the self-proclaimed checkers master, played the game on an IBM 7094

^% computer in 1962, and he lost to the computer. Compared to what can be done today, this feat seems trivial, but it's considered a major milestone in the field of artificial intelligence

#PROGRAM

```
import string
import nltk
nltk.download("punkt")
from nltk.corpus import stopwords
nltk.download("stopwords")
from nltk.tokenize import word_tokenize
def words_extractor(strl):
    print("length of strl:"+str(len(strl.split()))))
    strl=strl.lower()
    strl=strl.translate(str.maketrans("", "", string.punctuation))
    str_tokens=word_tokenize(strl)
    stop=stopwords.words('english')
    str_filtered_tokens=[]
    for i in str_tokens:
        if i not in stop :
            str_filtered_tokens.append(i)
    print(str_filtered_tokens)
    print("length of filtered strl:"+str(len(str_filtered_tokens)))
f=open('sample1.txt','r')
strl=f.read()
words_extractor(strl)
```

OUTPUT:

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data] Unzipping tokenizers/punkt.zip.

length of strl:88

['utilitatis', 'causa', 'amicitia', 'est', 'quaesita', 'lorem', 'ipsum', 'dolor', 'sit', 'amet',
'consectetur', 'adipiscing', 'elit', 'collatio', 'igitur', 'ista', 'te', 'nihil', 'iuvat', 'honestas', 'oratio',
'socratica', 'platonis', 'etiam', 'primum', 'nostrane', 'potestate', 'est', 'quid', 'meminerimus',
'duo', 'reges', 'constructio', 'interrete', 'quid', 'si', 'etiam', 'iucunda', 'memoria', 'est',
'praeteritorum', 'malorum', 'si', 'quidem', 'inquit', 'tollerem', 'sed', 'relinquo', 'nisi', 'populari',
'fama', 'quamquam', 'id', 'quidem', 'licebit', 'iis', 'existimare', 'qui', 'legerint', 'summum',
'vobis', 'bonum', 'voluptas', 'dicitur', 'hoc', 'eo', 'refert', 'tamen', 'quo', 'modo', 'quid',
'sequatur', 'quid', 'repugnet', 'vident', 'iam', 'id', 'ipsum', 'absurdum', 'maximum', 'malum',
'neglegi']

length of filtered strl:82

[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Unzipping corpora/stopwords.zip.

WEEK-3

Aim: Write a program for edge detection to extract edge based features from a sample image.

DISCRIPTION:

Edge Detection:

Edge detection is a technique of image processing used to identify points in a digital image with discontinuities, simply to say, sharp changes in the image brightness.

These points where the image brightness varies sharply are called the edges (or boundaries) of the image.

Edge detection algorithms typically work by computing the gradient of the image, which measures the rate of change of intensity in different directions. The gradient can be calculated using various methods, such as the Sobel, Prewitt, or Canny edge detectors. These algorithms apply a set of convolution filters to the image to highlight the areas of high gradient, which correspond to the edges.

The output of an edge detection algorithm is a binary image that shows the locations of the detected edges. This information can be used to extract edgebased features from the image, such as the length, orientation, and curvature of the edges. These features can be useful for various applications, such as object recognition, segmentation, and tracking in computer vision.

IMAGE:



OUTOUT: -

The resultant image is stored /saved in another file named "edge_detection_result.jpg"



WEEK-4

Aim:

Write a program to extract SURF/SIFT feature descriptors from a sample image.

Description:

SIFT:

SIFT (Scale-Invariant Feature Transform) is a feature detection algorithm in computer vision that can be used for object recognition, image stitching, and other applications. SIFT algorithm helps locate the local features in an image, commonly known as the 'keypoints' of the image. These keypoints are scale & rotation invariants that can be used for various computer vision applications, like image matching, object detection, scene detection, etc. While there is no built-in implementation of SIFT in OpenCV for Python due to patent issues, there are third-party implementations available, such as Mahotas.

Libraries:

Cv2:

cv2 is a Python library for computer vision tasks that is built on top of the OpenCV (Open Source Computer Vision Library) C++ library. It provides a wide range of functions for image and video processing, object detection and recognition, feature extraction, and more.

Here are some of the key features of cv2:

- Reading and writing image and video files in various formats
- Image and video processing and manipulation
- Object detection and recognition
- Feature detection and extraction
- Camera calibration and 3D reconstruction
- Machine learning and deep learning integration

matplotlib:

matplotlib is a Python library for creating static, animated, and interactive visualizations in Python. It provides a variety of tools for creating plots, charts,

and other types of visualizations, and can be used in conjunction with NumPy and Pandas for data analysis.

pyplot is a sublibrary of matplotlib that provides a high-level interface for creating plots and visualizations in Python. It provides a variety of functions for creating different types of plots, including line plots, scatter plots, bar charts, histograms, and more.

Program:

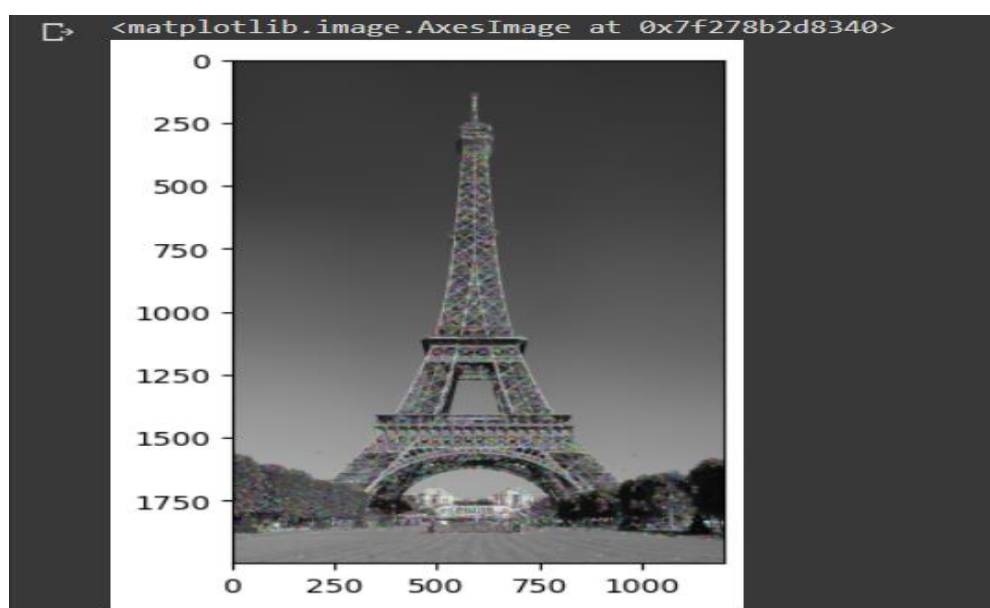
```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

#reading image
img1 = cv2.imread('eif_1.jpg')
gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

#keypoints
sift = cv2.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1, None)

img_1 = cv2.drawKeypoints(gray1, keypoints_1, img1)
plt.imshow(img_1)
```

Output:



Program:

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

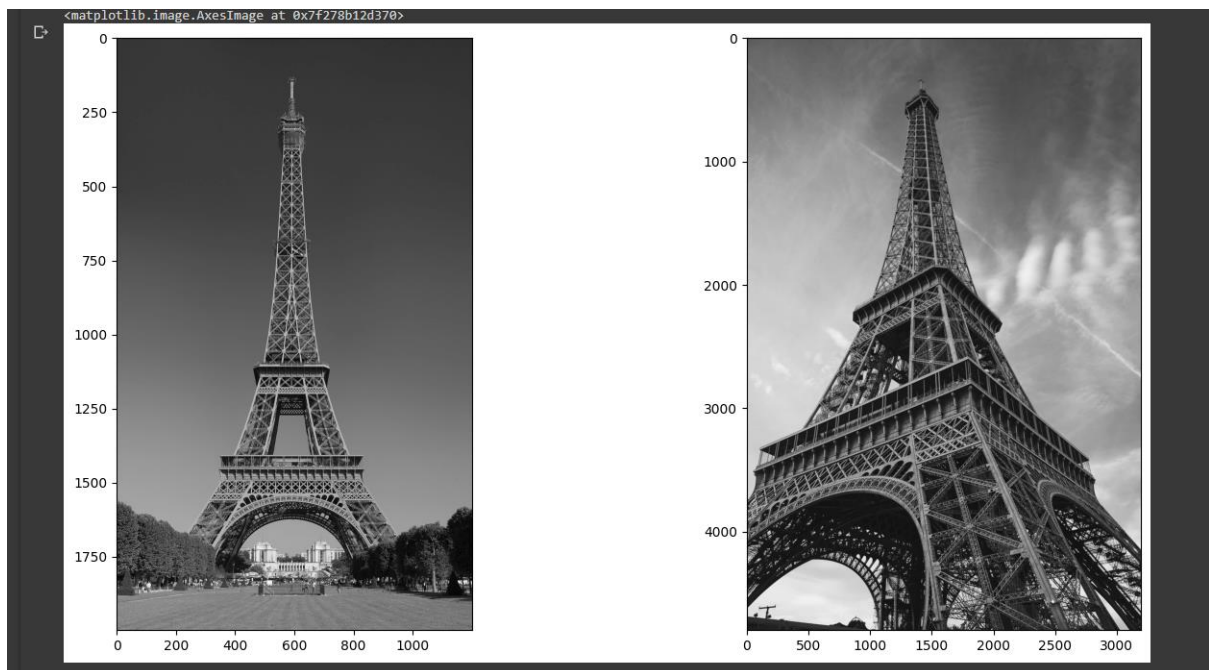
# read images
img1 = cv2.imread('eif_1.jpg')
img2 = cv2.imread('eif_2.jpg')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

figure, ax = plt.subplots(1, 2, figsize=(16, 8))

ax[0].imshow(img1, cmap='gray')
ax[1].imshow(img2, cmap='gray')
```

Output:



Program:

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# read images
img1 = cv2.imread('eif_1.jpg')
img2 = cv2.imread('eif_2.jpg')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

#sift
sift = cv2.xfeatures2d.SIFT_create()

keypoints_1, descriptors_1 = sift.detectAndCompute(im
g1, None)
keypoints_2, descriptors_2 = sift.detectAndCompute(im
g2, None)

len(keypoints_1), len(keypoints_2)
```

Output:

```
↳ (7312, 106026)
```

Program:

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# read images
img1 = cv2.imread('eif_1.jpg')
img2 = cv2.imread('eif_2.jpg')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

#sift
sift = cv2.xfeatures2d.SIFT_create()
```



```

keypoints_1, descriptors_1 = sift.detectAndCompute(im
g1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(im
g2,None)

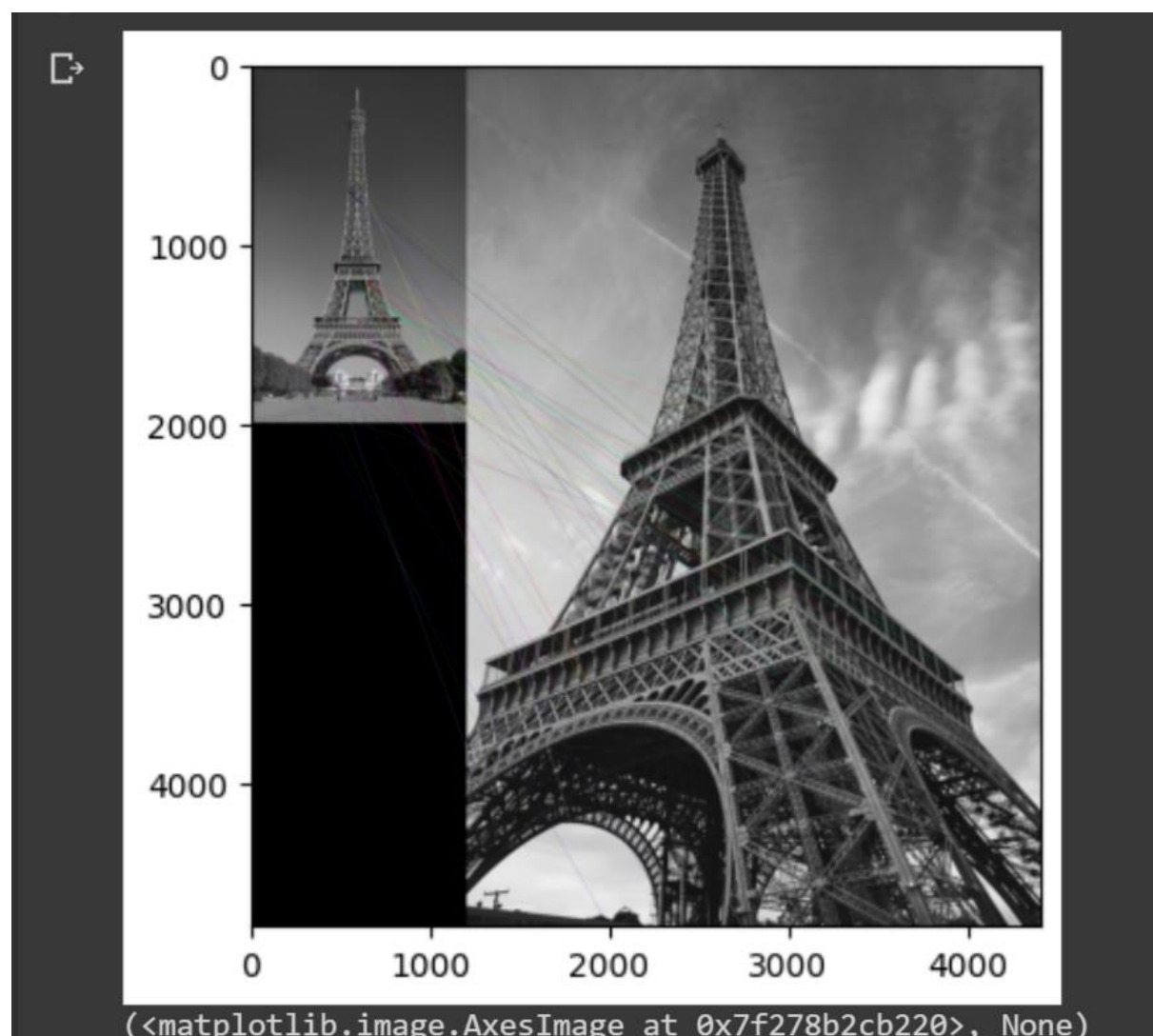
#feature matching
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)

img3 = cv2.drawMatches(img1, keypoints_1, img2, keypo
ints_2, matches[:50], img2, flags=2)
plt.imshow(img3),plt.show()

```

Output:



WEEK-5

Aim:

Write a program to perform Exploratory Data Analysis on real time datasets.

- a) Univariate Analysis
- b) Multivariate Analysis
- c) Visualization using correlation matrix

Description:

Exploratory Data Analysis (EDA) is a process of analyzing, summarizing, and visualizing data to gain insights into its underlying patterns and relationships. It involves applying various statistical and visualization techniques to gain a better understanding of the data and identify any outliers or anomalies.

The main goal of EDA is to discover and explore relationships and patterns in the data, which can then be used to inform further analysis and modeling. EDA can be applied to any type of data, including numerical, categorical, and textual data.

Some common techniques used in EDA include:

- Summary statistics: Descriptive statistics such as mean, median, mode, range, and standard deviation can provide a quick overview of the data.
- Data visualization: Graphical representations such as histograms, box plots, scatter plots, and heat maps can help identify patterns and relationships in the data.
- Outlier detection: Outliers are data points that are significantly different from the rest of the data. Identifying and removing them can help improve the accuracy of the analysis.
- Correlation analysis: Correlation measures the strength and direction of the relationship between two variables. Correlation analysis can help identify variables that are strongly related and may be useful in predicting outcomes.
- Dimensionality reduction: Dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-SNE can help reduce the complexity of high-dimensional data and identify underlying patterns.

Overall, EDA is an essential step in any data analysis project as it provides a foundation for further analysis and modeling. It allows data scientists to understand the characteristics of the data and identify any potential issues or limitations that may impact the accuracy of the analysis.

Program:

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

```
names = ['sepal_length', 'sepal_width', 'petal_length',
        'petal_width', 'species']
df = pd.read_csv('iris.data', names=names)
df.head()
```

Output:



	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Program:

```
df.shape
```

Output:

```
(150, 5)
```

Univariate Analysis:

Program:

```
df['species'].unique()
```

```
df['species'].value_counts()
```

```
df['species'].value_counts(normalize=True)
```

Output:

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

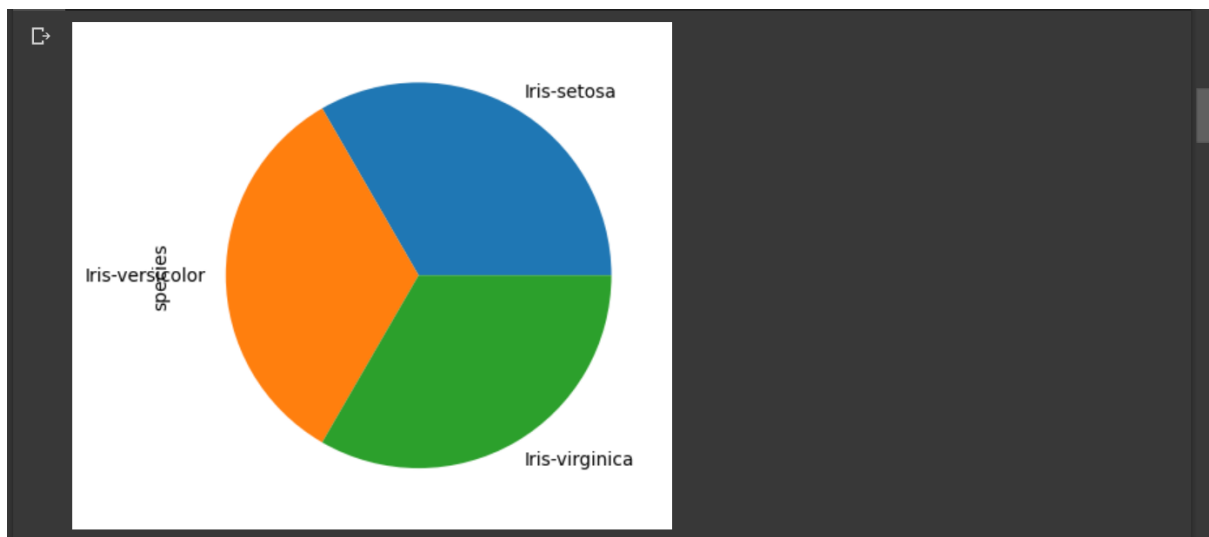
```
Iris-setosa      50  
Iris-versicolor  50  
Iris-virginica   50  
Name: species, dtype: int64
```

```
Iris-setosa      0.333333  
Iris-versicolor  0.333333  
Iris-virginica   0.333333  
Name: species, dtype: float64
```

Program:

```
df['species'].value_counts(normalize=True).plot.pie()  
plt.show()
```

Output:



Program:

```
df_setosa=df.loc[df['species']=='setosa']  
df_versicolor=df.loc[df['species']=='versicolor']  
df_virginica=df.loc[df['species']=='virginica']
```

```
df_setosa.head()
```

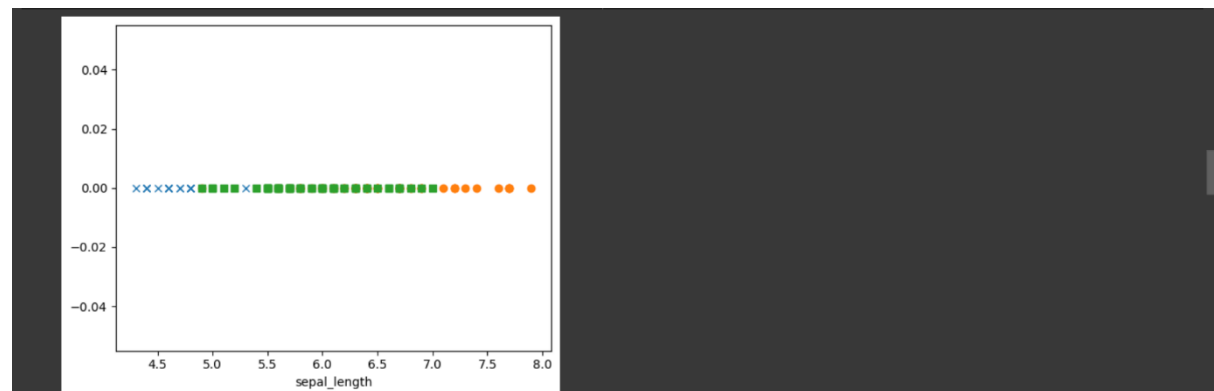
Output:

```
sepal_length  sepal_width  petal_length  petal_width  species
```

Program:

```
plt.plot(df_setosa['sepal_length'],np.zeros_like(df_s  
etosa['sepal_length']), 'x')  
plt.plot(df_virginica['sepal_length'],np.zeros_like(d  
f_virginica['sepal_length']), 'o')  
plt.plot(df_versicolor['sepal_length'],np.zeros_like(  
df_versicolor['sepal_length']), '^')  
plt.xlabel('sepal_length')  
plt.show()
```

Output:

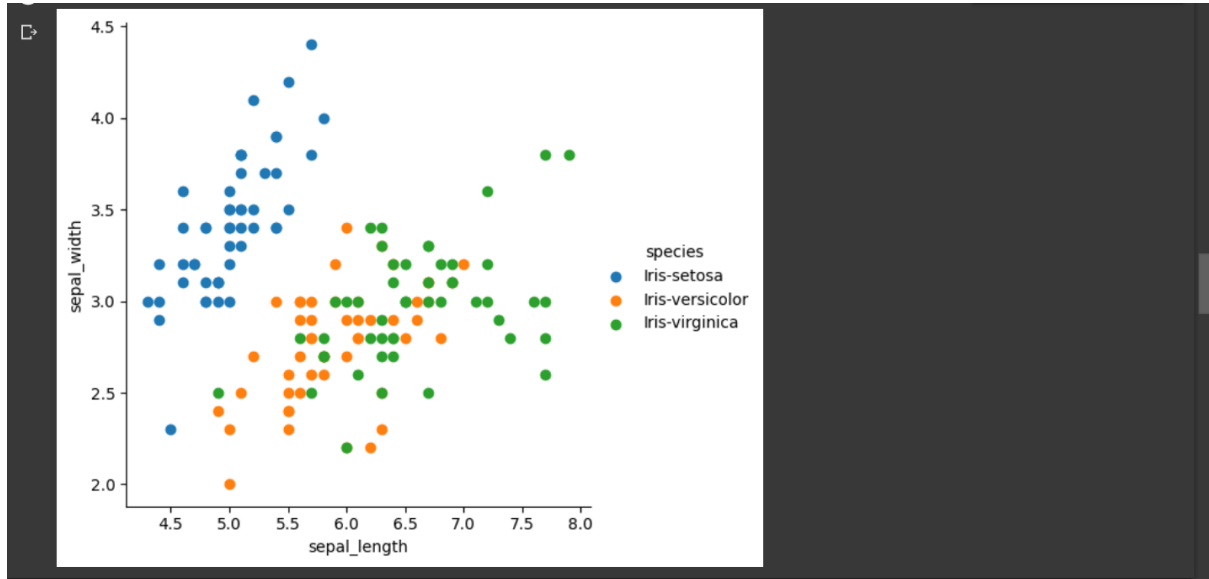


Bivariate Analysis:

Program:

```
sns.FacetGrid(df,hue="species",height=5).map(plt.scatter,"sepal_length","sepal_width").add_legend();  
plt.show()
```

Output:



Multivariate Analysis:

Program:

```
sns.pairplot(df,hue="species",height=5)  
plt.show()
```

Output:



Visualization using correlation matrix

Program:

```
df.corr()
```

Output:

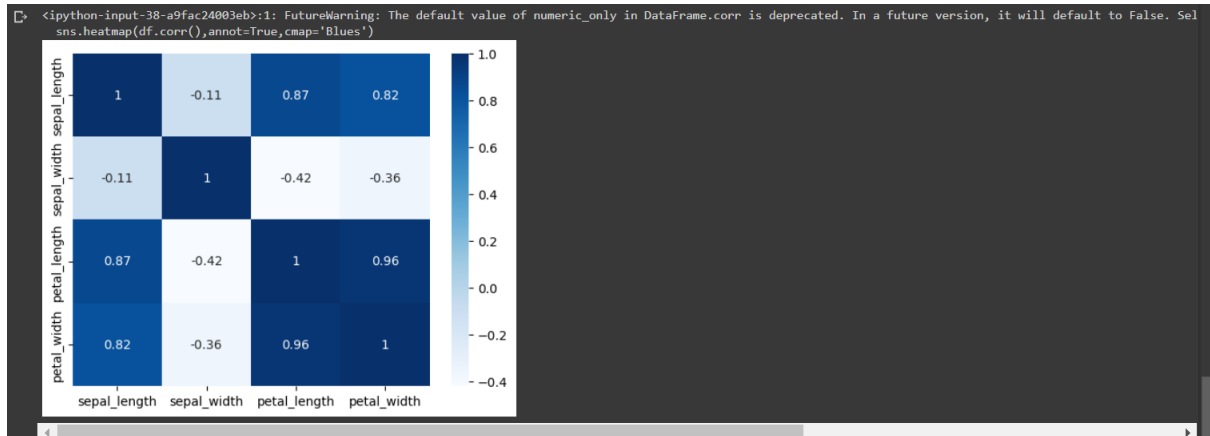
```
<ipython-input-37-2f6f668aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or sj
df.corr()
```

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

Program:

```
sns.heatmap(df.corr(),annot=True,cmap='Blues')  
plt.show()
```

Output:



WEEK-6

Aim:

Write a program to perform Dimensionality Reduction using Principle Component Analysis techniques on real time data sets.

Description:

PCA can be used for various purposes, such as:

1. Data compression: PCA can be used to compress high-dimensional data into a lower-dimensional space without losing too much information.
2. Visualization: PCA can be used to visualize high-dimensional data in a lower-dimensional space.
3. Data pre-processing: PCA can be used as a pre-processing step to reduce the dimensionality of the data before applying other machine learning algorithms.

Libraries:

Sklearn:

The scikit-learn library (often abbreviated as sklearn) is a popular Python library used for machine learning tasks such as classification, regression, and clustering. It is built on top of NumPy, SciPy, and matplotlib, and provides a wide range of machine learning algorithms, tools, and utilities.

Here are some key features of scikit-learn:

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

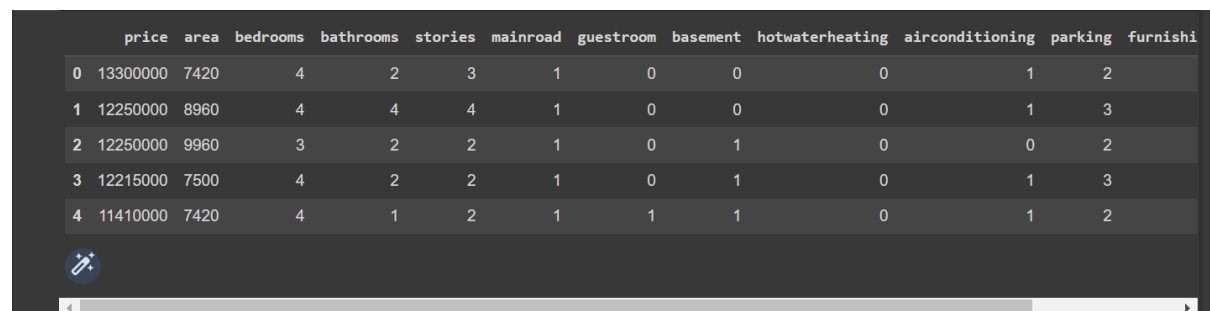
Program:

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

```
data=pd.read_csv("Housing.csv")
```

```
#preprocessing
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['mainroad']= le.fit_transform(data['mainroad'])
data['guestroom']= le.fit_transform(data['guestroom'])
)
data['basement']= le.fit_transform(data['basement'])
data['hotwaterheating']= le.fit_transform(data['hotwaterheating'])
data['airconditioning']= le.fit_transform(data['airconditioning'])
data['furnishingstatus']= le.fit_transform(data['furnishingstatus'])
data['preferred']= le.fit_transform(data['preferred'])
)
```

Output:



	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	furnishi
0	13300000	7420	4	2	3	1	0	0	0	1	2	
1	12250000	8960	4	4	4	1	0	0	0	1	3	
2	12250000	9960	3	2	2	1	0	1	0	0	2	
3	12215000	7500	4	2	2	1	0	1	0	1	3	
4	11410000	7420	4	1	2	1	1	1	0	1	2	

Program:

```
x_1=data.iloc[:, :-1]
y_1=data.iloc[:, -1]
```

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x_1, y_1, test_size= 0.2, random_state=0)
```

```
from sklearn.tree import DecisionTreeClassifier  
clf = DecisionTreeClassifier()  
  
clf = clf.fit(x_train,y_train)  
  
y_pred = clf.predict(x_test)
```

```
from sklearn import metrics  
ac_1 = metrics.accuracy_score(y_test, y_pred)  
print("Accuracy:",ac_1)
```

Output:

```
➞ Accuracy: 0.7706422018348624
```

Program:

```
from sklearn.preprocessing import StandardScaler  
  
sc = StandardScaler()  
x_train = sc.fit_transform(x_train)  
x_test = sc.transform(x_test)
```

```
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=4)  
x_train = pca.fit_transform(x_train)  
x_test = pca.transform(x_test)
```

```
clf = DecisionTreeClassifier()  
  
clf = clf.fit(x_train,y_train)  
  
y_pred = clf.predict(x_test)
```

```
ac_2 = metrics.accuracy_score(y_test, y_pred)
print("Accuracy after using pca:",ac_2)
```

Output:

```
☞ Accuracy after using pca: 0.7522935779816514
```

WEEK - 7

Aim: Write a program to perform Linear Discriminant Analysis for binary classification considering a real time dataset

Description:

Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. For example, we have two classes and we need to separate them efficiently.

Classes can have multiple features. Using only a single feature to classify them may result in some overlapping as shown in the below figure. So, we will keep on increasing the number of features for proper classification.

Program:

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

Load the dataset

```
#Load the Dataset
```

```
datas = pd.read_csv("./ionosphere_data.csv")
```

```
print(datas.head())
```

```
print(datas.columns)
```

```
print(datas.tail())
```

Output:

```

Index(['column_a', 'column_b', 'column_c', 'column_d', 'column_e', 'column_f',
      'column_g', 'column_h', 'column_i', 'column_j', 'column_k', 'column_l',
      'column_m', 'column_n', 'column_o', 'column_p', 'column_q', 'column_r',
      'column_s', 'column_t', 'column_u', 'column_v', 'column_w', 'column_x',
      'column_y', 'column_z', 'column_aa', 'column_ab', 'column_ac',
      'column_ad', 'column_ae', 'column_af', 'column_ag', 'column_ah',
      'column_ai'],
      dtype='object')

```

	column_a	column_b	column_c	column_d	column_e	column_f	column_g	\
346	True	False	0.83508	0.08298	0.73739	-0.14706	0.84349	
347	True	False	0.95113	0.00419	0.95183	-0.02723	0.93438	
348	True	False	0.94701	-0.00034	0.93207	-0.03227	0.95177	
349	True	False	0.90608	-0.01657	0.98122	-0.01989	0.95691	
350	True	False	0.84710	0.13533	0.73638	-0.06151	0.87873	

	column_h	column_i	column_j	...	column_z	column_aa	column_ab	\
346	-0.05567	0.90441	-0.04622	...	-0.04202	0.83479	0.00123	
347	-0.01920	0.94590	0.01606	...	0.01361	0.93522	0.04925	
348	-0.03431	0.95584	0.02446	...	0.03193	0.92489	0.02542	
349	-0.03646	0.85746	0.00110	...	-0.02099	0.89147	-0.07760	
350	0.08260	0.88928	-0.09139	...	-0.15114	0.81147	-0.04822	

	column_ac	column_ad	column_ae	column_af	column_ag	column_ah	\
346	1.00000	0.12815	0.86660	-0.10714	0.90546	-0.04307	
347	0.93159	0.08168	0.94066	-0.00035	0.91483	0.04712	
348	0.92120	0.02242	0.92459	0.00442	0.92697	-0.00577	
349	0.82983	-0.17238	0.96022	-0.03757	0.87403	-0.16243	
350	0.78207	-0.00703	0.75747	-0.06678	0.85764	-0.06151	

	column_ai
346	g
347	g
348	g
349	g
350	g

[5 rows x 35 columns]

Check For Null Values

```
datas.isnull().values.any()
```

Output:

False

Renaming Attribute to Target

```

datas.rename(columns = {'column_ai' : 'Target'}, inplace = True)
print(datas['Target'])

```

Output:

```
0      g
1      b
2      g
3      b
4      g
..
346    g
347    g
348    g
349    g
350    g
Name: Target, Length: 351, dtype: object
```

Class Balancing Data Set:

```
print(datas['Target'].value_counts())
```

Output:

```
g      225
b      126
Name: Target, dtype: int64
```

```
res1 = pd.DataFrame()
res2 = pd.DataFrame()
for i in range(0, 351):
    if datas.loc[i, 'Target'] == 'g':
        res1 = res1.append(datas.loc[i], ignore_index = True)
    elif datas.loc[i, 'Target'] == 'b':
        res2 = res2.append(datas.loc[i], ignore_index = True)
```

```
print(res1)
```

Output:

```
   column_a  column_b  column_c  column_d  column_e  column_f  column_g  \
0         1.0         0.0   0.99539 -0.05889   0.85243   0.02306   0.83398
1         1.0         0.0   1.00000 -0.03365   1.00000   0.00485   1.00000
2         1.0         0.0   1.00000 -0.02401   0.94140   0.06531   0.92106
3         1.0         0.0   0.97588 -0.10602   0.94601 -0.20800   0.92806
4         1.0         0.0   0.96355 -0.07198   1.00000 -0.14333   1.00000
..      ...      ...      ...      ...      ...      ...      ...
220        1.0         0.0   0.83508   0.08298   0.73739 -0.14706   0.84349
221        1.0         0.0   0.95113   0.00419   0.95183 -0.02723   0.93438
222        1.0         0.0   0.94701 -0.00034   0.93207 -0.03227   0.95177
223        1.0         0.0   0.90608 -0.01657   0.98122 -0.01989   0.95691
224        1.0         0.0   0.84710   0.13533   0.73638 -0.06151   0.87873

   column_h  column_i  column_j  ...  column_z  column_aa  column_ab  \
0  -0.37708   1.00000   0.03760  ...  -0.51171   0.41078   -0.46168
1  -0.12062   0.88965   0.01198  ...  -0.40220   0.58984   -0.22145
2  -0.23255   0.77152  -0.16399  ...  -0.65158   0.13290   -0.53206
3  -0.28350   0.85996  -0.27342  ...  -0.81634   0.13659   -0.82510
4  -0.21313   1.00000  -0.36174  ...  -0.65440   0.57577   -0.69712
..      ...      ...      ...      ...      ...      ...      ...
220  -0.05567   0.90441  -0.04622  ...  -0.04202   0.83479   0.00123
221  -0.01920   0.94590   0.01606  ...   0.01361   0.93522   0.04925
222  -0.03431   0.95584   0.02446  ...   0.03193   0.92489   0.02542
223  -0.03646   0.85746   0.00110  ...  -0.02099   0.89147   -0.07760
224   0.08260   0.88928  -0.09139  ...  -0.15114   0.81147   -0.04822

   column_ac  column_ad  column_ae  column_af  column_ag  column_ah  Target
0    0.21266  -0.34090   0.42267  -0.54487   0.18641  -0.45300      g
1    0.43100  -0.17365   0.60436  -0.24180   0.56045  -0.38238      g
2    0.02431  -0.62197  -0.05707  -0.59573  -0.04608  -0.65697      g
3    0.04606  -0.82395  -0.04262  -0.81318  -0.13832  -0.80975      g
4    0.25435  -0.63919   0.45114  -0.72779   0.38895  -0.73420      g
..      ...      ...      ...      ...      ...      ...      ...
220   1.00000   0.12815   0.86660  -0.10714   0.90546  -0.04307      g
221   0.93159   0.08168   0.94066  -0.00035   0.91483   0.04712      g
222   0.92120   0.02242   0.92459   0.00442   0.92697  -0.00577      g
223   0.82983  -0.17238   0.96022  -0.03757   0.87403  -0.16243      g
224   0.78207  -0.00703   0.75747  -0.06678   0.85764  -0.06151      g
```

```
[225 rows x 35 columns]
```

```
fres1 = pd.DataFrame(res1.sample(n = 126, ignore_index = True))
df = [fres1, res2]
df = pd.concat(df, ignore_index = True)
print(df)
```

```
df = pd.DataFrame(df.sample( n= 252, ignore_index = True))
```

```
print(df['Target'].value_counts())
```

Output:

```
b    126
g    126
Name: Target, dtype: int64
```

Splitting Data:

```
X = df[df.columns[ : 34]]
```

```
Y = df['Target']
```

```
X_train, X_test , Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state
= 23)
```

Standardizing Data:

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

Applying LDA :

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
from sklearn.metrics import mean_squared_error
```

```
lda = LDA(n_components = 1 )
```

```
X_train = lda.fit_transform(X_train, Y_train)
```

```
X_test = lda.transform(X_test)
```

Building And Training Model :

```
from sklearn.ensemble import RandomForestClassifier as RFC
```

```
rfc = RFC(max_depth = 2, random_state = 21)
```



```
rfc.fit(X_train, Y_train)
```

Testing Model:

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

```
y_pred
```

Output:

```
array(['g', 'b', 'g', 'b', 'g', 'g', 'g', 'g', 'b', 'b', 'b', 'g', 'g',  
      'g', 'b', 'g', 'b', 'b', 'g', 'g', 'g', 'b', 'b', 'g', 'b', 'b',  
      'g', 'b', 'g', 'b', 'b', 'g', 'g', 'b', 'g', 'b', 'g', 'b', 'g',  
      'g', 'b', 'g', 'g', 'g', 'g', 'b', 'g', 'b', 'b', 'b', 'g'],  
      dtype=object)
```

Evaluating Model :

```
cm = confusion_matrix(Y_test, y_pred)
```

```
sc = accuracy_score(Y_test, y_pred)
```

```
print("Accuracy Score : ", sc*100)
```

```
print("Confusion Matrix \n", cm)
```

Output:

```
Accuracy Score : 86.27450980392157  
Confusion Matrix  
[[22  6]  
 [ 1 22]]
```

WEEK-8

AIM: Write a program to implement the linear regression for a sample training data set stored as a .CSV file.

DESCRIPTION:

Linear regression is a quiet and simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression. If there is a single input variable (x), such linear regression is called simple linear regression. And if there is more than one input variable, such linear regression is called multiple linear regression. The linear regression model gives a sloped straight line describing the relationship within the variables.

FUNCTIONS:

LinearRegression:

Syntax:

```
sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False, copy_X=True)
```

Parameters :

fit_intercept : [boolean, Default is True] Whether to calculate intercept for the model.

normalize : [boolean, Default is False] Normalisation before regression.

copy_X : [boolean, Default is True] If true, make a copy of X else overwritten.

Fit:

Syntax : `my_linear_regressor.fit(x_train, y_train)`

Where `x_train` is features of training data

`Y_train` is target/label of training data

Predict:

Syntax : `my_linear_regressor.predict(x_test)`

Where `x_test` is features of test data

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
import pandas as pd
df = pd.read_csv("1.01. Simple linear regression.csv")
df
```

Output:

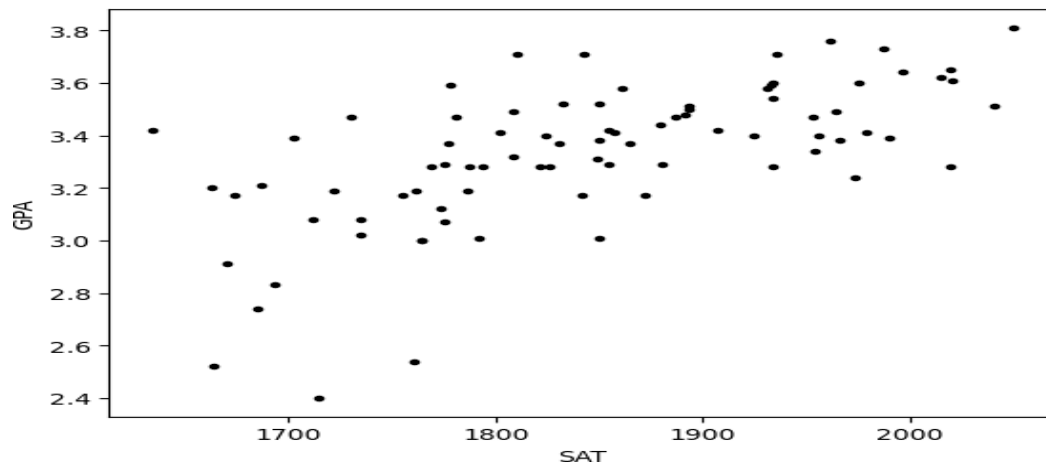
	SAT	GPA
0	1714	2.40
1	1664	2.52
2	1760	2.54
3	1685	2.74
4	1693	2.83
...
79	1936	3.71
80	1810	3.71
81	1987	3.73
82	1962	3.76
83	2050	3.81

84 rows × 2 columns

Program:

```
import matplotlib.pyplot as plt
plt.xlabel('SAT')
plt.ylabel('GPA')
plt.scatter(df.SAT,df.GPA,color='black', marker='.')
```

Output: <matplotlib.collections.PathCollection at 0x1b9745a03d0>



Program:

```
from sklearn import linear_model
model=linear_model.LinearRegression()
model.fit(df[['SAT']],df.GPA)
```

Output:

```
LinearRegression()
```

Program:

```
import numpy as np
model.predict(np.array([[3300]]))
```

Output:

```
array([5.73881086])
```

Program:

```
m=model.coef_
print(m)
```

Output:

```
[0.00165569]
```

Program:

```
c=model.intercept_
print(c)
```

Output:

```
0.2750402996602803
```

Program:

```
y=m*3300+c
print(y)
```

Output:

[5.73881086]

Program:

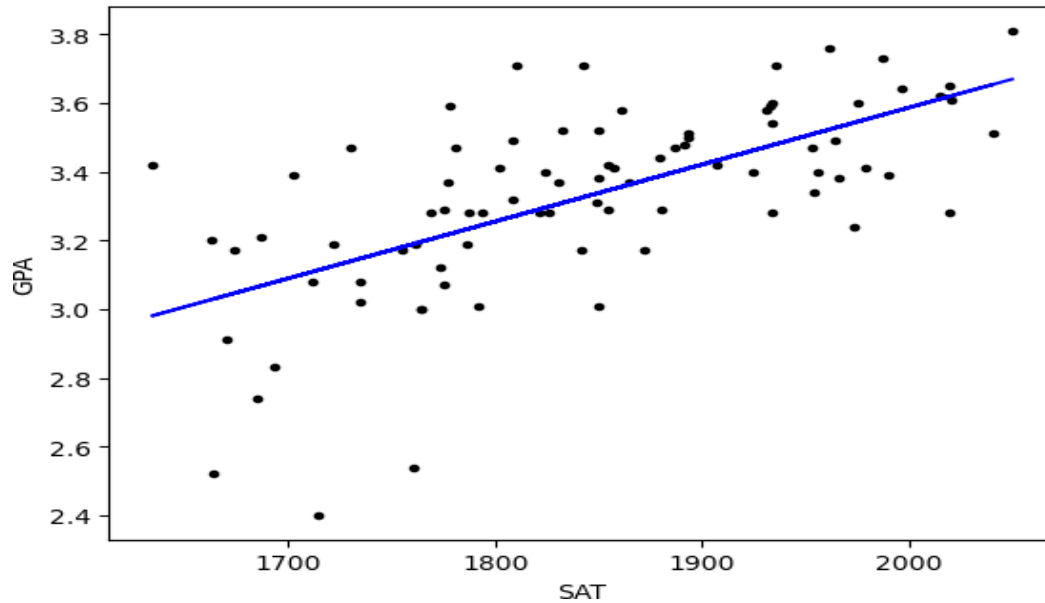
```
plt.xlabel('SAT')
```

```
plt.ylabel('GPA')
```

```
plt.scatter(df.SAT,df.GPA,color='black', marker='.')
```

```
plt.plot(df.SAT,model.predict(df[['SAT']]),color='blue')
```

Output: [<matplotlib.lines.Line2D at 0xb97447b7f0>]



WEEK-9

Aim: Write a program to implement the Non-linear Regression for a sample training data set stored as .CSV file. Compute Mean Square Error by considering few test data sets.

Description:

Non-linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables, where the relationship is not a simple linear function. In contrast to linear regression, which assumes that the relationship between the variables is linear, non-linear regression models can capture more complex relationships, such as curves or other non-linear patterns.

Non-linear regression involves estimating the parameters of a non-linear function that best fits the data. This can be done using various methods, such as the least-squares method or maximum likelihood estimation. The non-linear function can be any type of non-linear equation, such as polynomial, exponential, logarithmic, or trigonometric functions.

Polynomial Features:-

Polynomial features are those features created by raising existing features to an exponent.

Syntax:

```
class sklearn.preprocessing.PolynomialFeatures(degree=2, *, interaction_only=False, include_bias=True, order='C')
```

Nystroem:

Syntax:-

```
class sklearn.kernel_approximation.Nystroem(kernel='rbf', *, gamma=None, coef0=None, degree=None, kernel_params=None, n_components=100, random_state=None, n_jobs=None)
```

Methods:-

<u>fit</u> (X[, y])	Compute number of output features.
<u>fit transform</u> (X[, y])	Fit to data, then transform it.
<u>get feature names out</u> ([input_features])	Get output feature names for transformation.
<u>get params</u> ([deep])	Get parameters for this estimator.
<u>set output</u> (*[, transform])	Set output container.

<u>set params</u> (**params)	Set the parameters of this estimator.
<u>transform</u> (X)	Transform data to polynomial features.

Program:

```
import pandas as pd

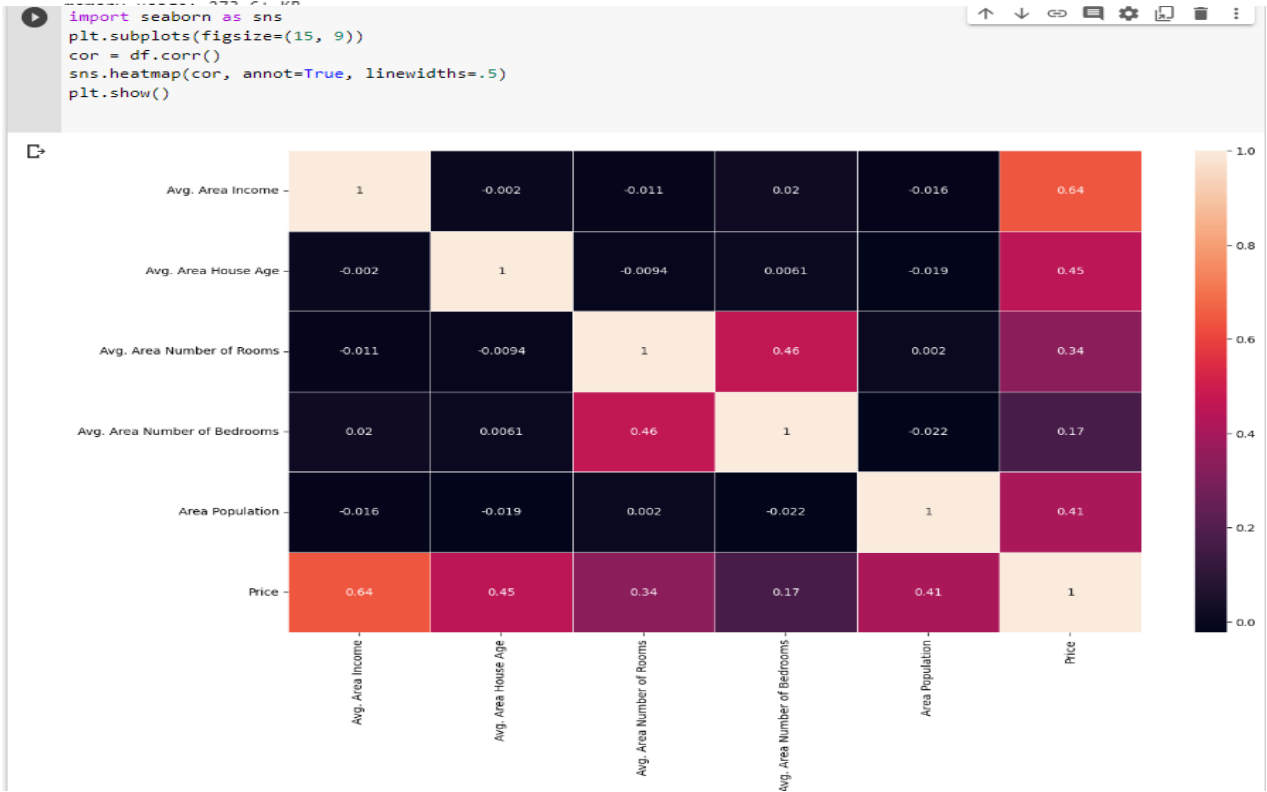
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
df=pd.read_csv('USA_Housing.csv')
df.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

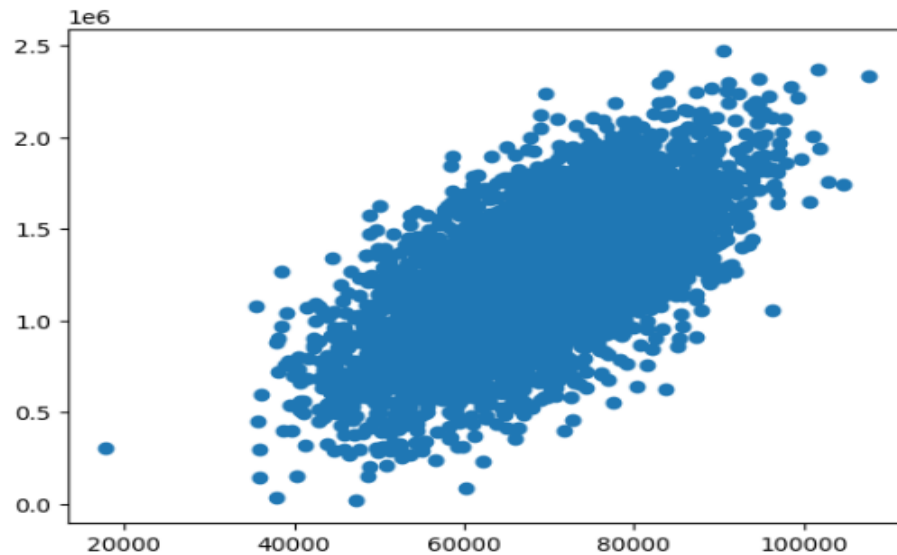
```
[4] df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Avg. Area Income                      5000 non-null   float64
1   Avg. Area House Age                   5000 non-null   float64
2   Avg. Area Number of Rooms              5000 non-null   float64
3   Avg. Area Number of Bedrooms           5000 non-null   float64
4   Area Population                        5000 non-null   float64
5   Price                                 5000 non-null   float64
6   Address                               5000 non-null   object
dtypes: float64(6), object(1)
```



```
▶ X=df['Avg. Area Income']  
y=df['Price']  
plt.scatter(X,y)
```

```
↳ <matplotlib.collections.PathCollection at 0x7f4508243b80>
```



```
[11] from sklearn.model_selection import train_test_split  
X1,x1,Y1,y1=train_test_split(X,y,test_size=0.5)  
x2,x3,y2,y3=train_test_split(X1,Y1,test_size=0.2)
```

```
▶ from sklearn.preprocessing import PolynomialFeatures  
from sklearn.linear_model import LassoCV  
from sklearn.pipeline import make_pipeline  
  
import warnings  
warnings.filterwarnings("ignore")  
from sklearn.metrics import mean_squared_error,r2_score
```



```

lasso_eps = 0.0001
lasso_alpha=100
lasso_iter=2000
degree_min = 2
degree_max = 8

rmse=[]
test_score=[]
r2score=[]
X_train, X_test, y_train, y_test = train_test_split(x1,y1,test_size=0.2)
for degree in range(degree_min,degree_max+1):

    model = make_pipeline(PolynomialFeatures(degree, interaction_only=False),LassoCV(eps=lasso_eps,n_alphas=lasso_alpha,max_iter=lasso_iter,cv=5))
    model.fit(np.array(X_train).reshape(-1,1),y_train)
    test_pred = np.array(model.predict(np.array(X_test).reshape(-1,1)))
    rmse.append(np.sqrt(mean_squared_error(y_test, test_pred)))
    test_score.append(model.score(np.array(X_test).reshape(-1,1),y_test))
    r2score.append(r2_score(y_test,test_pred ))

```

```
5] max(test_score)
```

```
0.37550227273206005
```

```
7] min(rmse)
```

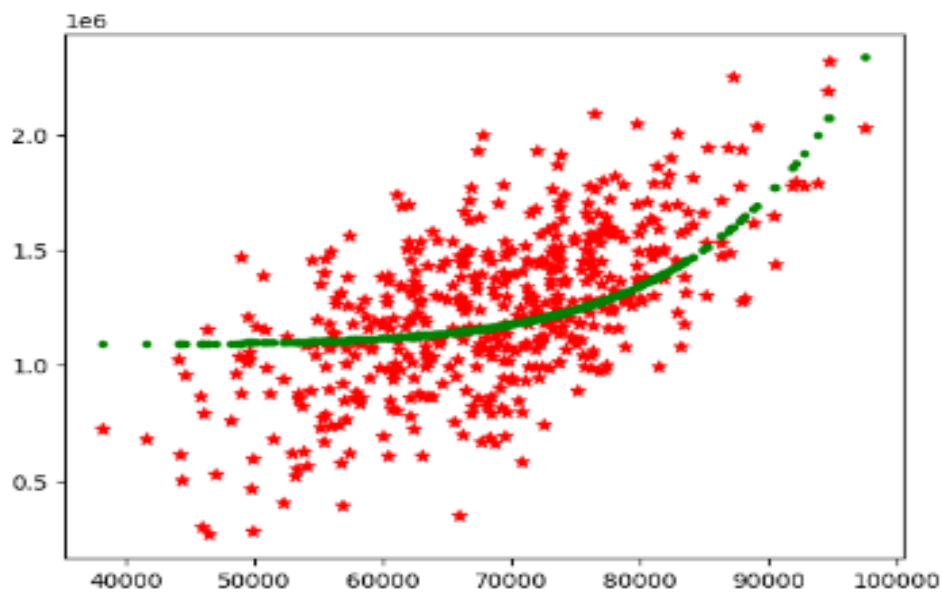
```
267736.18495455384
```

```

plt.scatter(X_test,y_test,color='red',marker="*")
plt.scatter(X_test,test_pred,color='green',marker=".")

```

```
<matplotlib.collections.PathCollection at 0x7f450124d250>
```



```
[ ] from sklearn.kernel_approximation import Nystroem
from sklearn.linear_model import LinearRegression
nystroem_regression = make_pipeline( Nystroem(n_components=40), LinearRegression(),)

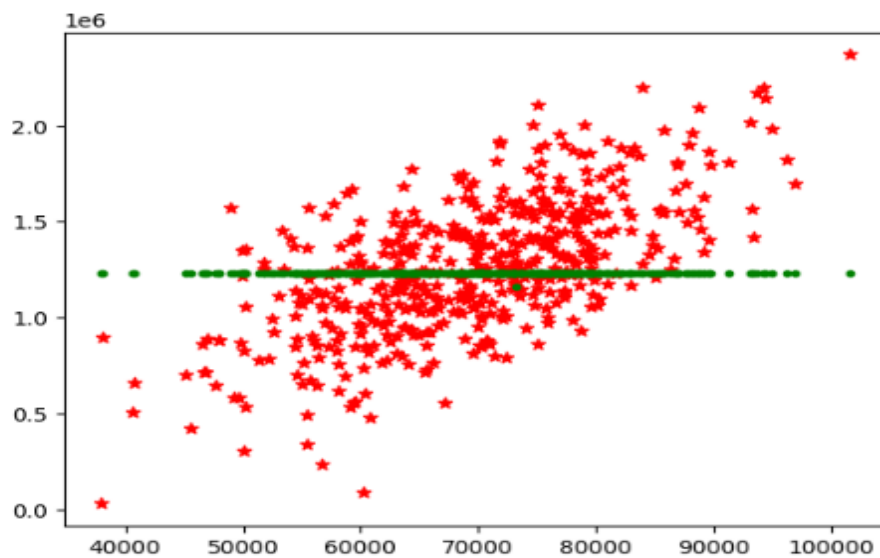
nystroem_regression.fit(np.array(x2).reshape(-1,1), y2)
target_predicted = nystroem_regression.predict(np.array(x3).reshape(-1,1))

mse = mean_squared_error(y3, target_predicted)
error=np.sqrt(mse)
print("MSE= ",error)
```

➤ MSE= 352467.48295791325

```
[22] plt.scatter(x3,y3,color='red',marker="*")
plt.scatter(x3,target_predicted,color='green',marker=".")
```

<matplotlib.collections.PathCollection at 0x7f450119bd00>



WEEK-10

Aim:

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Description:

Libraries:

Sklearn:

The scikit-learn library (often abbreviated as sklearn) is a popular Python library used for machine learning tasks such as classification, regression, and clustering. It is built on top of NumPy, SciPy, and matplotlib, and provides a wide range of machine learning algorithms, tools, and utilities.

Here are some key features of scikit-learn:

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

GaussianNB in sklearn:

Gaussian Naive Bayes is a classification algorithm that is part of the scikit-learn library in Python. It is based on the Bayes theorem and assumes that the probability of each feature value is independent of the other feature values. This means that the presence of a particular feature in a class is unrelated to the presence of any other feature.

The GaussianNB class in scikit-learn implements the Gaussian Naive Bayes algorithm for classification. It assumes that the features are normally distributed and uses the mean and variance of each feature for each class to make predictions.

Program:

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

```
data=pd.read_csv("Breast cancer data.csv")
```

```
data.head()
```

Output:

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0

Program:

```
x = data.drop('diagnosis', 1)
y = data['diagnosis']
```

```
x.head()
```

Output:

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030

Program:

```
#dropping unwanted features
x = x.drop('mean_perimeter', 1)
x = x.drop('mean_area',1)
```

```
x.head()
```

Output:

	mean_radius	mean_texture	mean_smoothness
0	17.99	10.38	0.11840
1	20.57	17.77	0.08474
2	19.69	21.25	0.10960
3	11.42	20.38	0.14250
4	20.29	14.34	0.10030

Program:

```
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)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

Output:

<div> <div> GaussianNB </div> <div> GaussianNB() </div> </div>
--

Program:

```
y_pred = classifier.predict(x_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy
_score
```

```
cm = confusion_matrix(y_test, y_pred)
cm
```

Output:

<div> <div> array([[39, 8], [2, 65]]) </div> </div>
--

Program:

```
ac = accuracy_score(y_test,y_pred)
ac
```

Output:

```
0.9122807017543859
```

WEEK-11

Aim:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

Description:

Libraries:

Sklearn:

The scikit-learn library (often abbreviated as sklearn) is a popular Python library used for machine learning tasks such as classification, regression, and clustering. It is built on top of NumPy, SciPy, and matplotlib, and provides a wide range of machine learning algorithms, tools, and utilities.

Here are some key features of scikit-learn:

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

KNeighborsClassifier in sklearn:

KNeighborsClassifier is a classification algorithm that is part of the scikit-learn library in Python. It is a type of instance-based learning or lazy learning where the algorithm simply stores the training data and makes predictions by comparing the new data to the stored training data.

The KNeighborsClassifier class in scikit-learn implements the k-nearest neighbours algorithm for classification. It works by finding the k nearest neighbours of a new data point and assigning it to the class that is most common among those neighbours. The number k is a hyperparameter that needs to be set before training the model.

Program:

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

```
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
```

```
data = pd.read_csv('iris.data', names=names)
```

```
data.head()
```

Output:



	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Program:

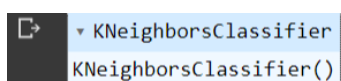
```
x = data.drop('Class', 1)
y = data['Class']
```

```
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)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, me
tric = 'minkowski', p = 2)
classifier.fit(x_train, y_train)
```

Output:



```
▼ KNeighborsClassifier
KNeighborsClassifier()
```


Program:

```
y_pred = classifier.predict(x_test)
```

```
y_test
```

Output:

```
114    Iris-virginica
62     Iris-versicolor
33      Iris-setosa
107    Iris-virginica
7       Iris-setosa
100    Iris-virginica
40      Iris-setosa
86     Iris-versicolor
76     Iris-versicolor
71     Iris-versicolor
134    Iris-virginica
51     Iris-versicolor
73     Iris-versicolor
54     Iris-versicolor
63     Iris-versicolor
37      Iris-setosa
78     Iris-versicolor
90     Iris-versicolor
45      Iris-setosa
16      Iris-setosa
121    Iris-virginica
66     Iris-versicolor
24      Iris-setosa
8       Iris-setosa
126    Iris-virginica
22      Iris-setosa
44      Iris-setosa
97     Iris-versicolor
93     Iris-versicolor
26      Iris-setosa
Name: Class, dtype: object
```

Program:

```
y_pred
```

Output:

```
array(['Iris-virginica', 'Iris-versicolor', 'Iris-setosa',  
      'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',  
      'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',  
      'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',  
      'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',  
      'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',  
      'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',  
      'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',  
      'Iris-versicolor', 'Iris-setosa'], dtype=object)
```

Program:

```
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
cm = confusion_matrix(y_test, y_pred)  
cm
```

Output:

```
array([[11,  0,  0],  
       [ 0, 13,  0],  
       [ 0,  0,  6]])
```

Program:

```
total_predictions=0  
correct_predictions=0  
wrong_predictions=0  
  
for i in range(3):  
    for j in range(3):  
        total_predictions = total_predictions + cm[i][j]  
print("total predictions are ")  
print(total_predictions)  
  
for i in range(3):  
    for j in range(3):  
        if(i==j):  
            correct_predictions = correct_predictions + cm[i][j]
```

```
print("correct predictions are ")
print(correct_predictions)

for i in range(3):
    for j in range(3):
        if(i!=j):
            wrong_predictions = wrong_predictions + cm[i][j]
]
print("wrong predictions are ")
print(wrong_predictions)
```

Output:

```
total predictions are
30
correct predictions are
30
wrong predictions are
0
```

Program:

```
ac = accuracy_score(y_test,y_pred)
ac
```

Output:

```
1.0
```

WEEK-12

AIM: Write a program to implement the Support Vector Machine algorithm to classify the iris dataset. Print both correct and wrong predictions.

DESCRIPTION:

Support vector machines (SVMs) are powerful yet flexible supervised machine learning methods used for classification, regression, and, outliers' detection. SVMs are very efficient in high dimensional spaces and generally are used in classification problems. SVMs are popular and memory efficient because they use a subset of training points in the decision function.

The main goal of SVMs is to divide the datasets into number of classes in order to find a **maximum marginal hyperplane (MMH)** which can be done in the following two steps –

- Support Vector Machines will first generate hyperplanes iteratively that separates the classes in the best way.
- After that it will choose the hyperplane that segregate the classes correctly.

Some important concepts in SVM are as follows – • **Support Vectors** – They may be defined as the datapoints which are closest to the hyperplane. Support vectors help in deciding the separating line.

- **Hyperplane** – The decision plane or space that divides set of objects having different classes.
- **Margin** – The gap between two lines on the closet data points of different classes is called margin.

Classification of SVM

- Scikit-learn provides three classes namely **SVC**, **NuSVC** and **LinearSVC** which can perform multiclass-class classification.

SVC:

It is C-support vector classification whose implementation is based on libsvm. The module used by scikitlearn is sklearn.svm.SVC. This class handles the multiclass support according to one-vs-one scheme.

Syntax:

```
class
sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, p
obability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-
1, decision_function_shape='ovr', break_ties=False, random_state=None)
```

Methods:

decision_function (X)	Evaluate the decision function for the samples in X.
fit (X, y[, sample_weight])	Fit the SVM model according to the given training data.
get_params ([deep])	Get parameters for this estimator.
predict (X)	Perform classification on samples in X.
predict_log_proba (X)	Compute log probabilities of possible outcomes for samples in X.
predict_proba (X)	Compute probabilities of possible outcomes for samples in X.
score (X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params (**params)	Set the parameters of this estimator.

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import load_iris
iris=load_iris()
dir(iris)
```

```
['DESCR', 'data', 'data_module', 'feature_names', 'filename', 'frame', 'target', 'target_names']
```

iris.data

```
array([[5.1, 3.5, 1.4, 0.2], [4.9, 3. , 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2],..... [5.9, 3. ,
5.1, 1.8]])
```

```
ar=iris.data
```

```
df=pd.DataFrame(ar,columns=iris.feature_names)
```

```
df.head()
```

index	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
y=iris.target y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..... 2, 2, 2])
```

```
df['target']=y df.head()
```

index	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df[df.target==0].count()
```

```
sepal length (cm) 50
sepal width (cm) 50 petal
length (cm) 50 petal
width (cm) 50
target 50 dtype:
int64
```

```
df[df.target==1].count()
```

```
sepal length (cm) 50
sepal width (cm) 50
petal length (cm) 50
petal width (cm) 50
target 50 dtype:
int64
```

```
df[df.target==2].count()
```

```
sepal length (cm) 50
sepal width (cm) 50
petal length (cm) 50
petal width (cm) 50
target 50 dtype:
int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 150 entries, 0 to 149 Data
columns (total 5 columns):
```

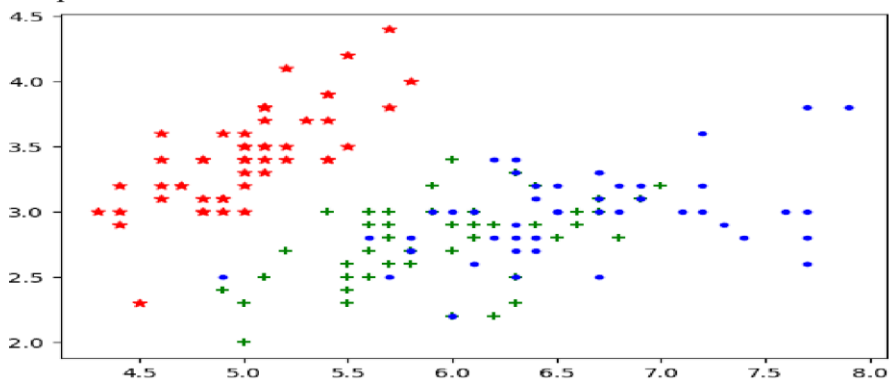
```
#   Column          Non-Null Count  Dtype
--  --
0   sepal length (cm)  150 non-null    float64
1   sepal width (cm)   150 non-null    float64
2   petal length (cm)  150 non-null    float64
3   petal width (cm)   150 non-null    float64
4   target             150 non-null    int64
```

dtypes: float64(4), int64(1)
memory usage: 6.0 KB

```
df0=df[df.target==0]
```

```
plt.ylabel='sepal width (cm)'  
plt.scatter(df0['sepal length (cm)'],df0['sepal width (cm)'],marker="*",color="red")  
plt.scatter(df1['sepal length (cm)'],df1['sepal width (cm)'],marker="+",color="green")  
plt.scatter(df2['sepal length (cm)'],df2['sepal width (cm)'],marker=".",color="blue")
```

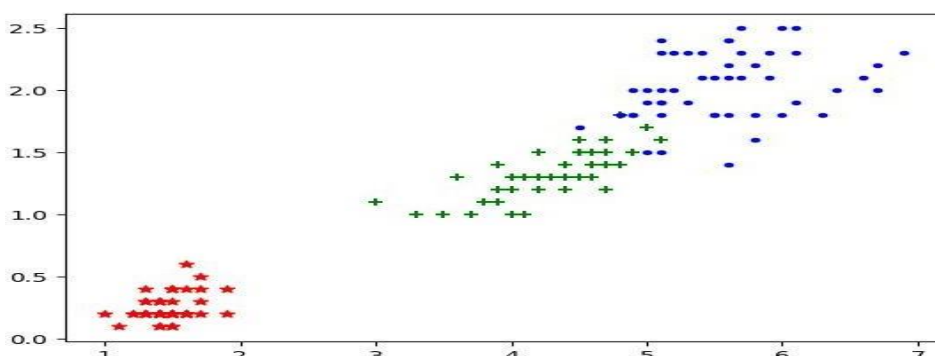
<matplotlib.collections.PathCollection at 0x7fe24b6e0f70>



```
plt.xlabel='petal length (cm)'  
plt.ylabel='petal width (cm)'  
plt.scatter(df0['petal length (cm)'],df0['petal width (cm)'],marker="*",color="red")  
plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],marker="+",color="green")  
plt.scatter(df2['petal length (cm)'],df2['petal width (cm)'],marker=".",color="blue")
```

<matplotlib.collections.PathCollection at 0x7fe248d8b6d0>

```
df1=df[df.target==1]  
df2=df[df.target==2] plt.xlabel='sepal  
length (cm)'
```



```
X=df.drop(['sepal length (cm)', 'sepal width (cm)', 'target'],axis=1) X.head()
```

index	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2

2	1.3	0.2
3	1.5	0.2
4	1.4	0.2

```
y=df.target
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=None,shuffle=True)
from sklearn.svm import SVC model=SVC()
model.fit(X_train,y_train)
model.score(X_test,y_test)
```

0.9333333333333333

```
from sklearn.model_selection import
cross_val_score
```

array([0.96666667, 0.96666667, 0.93333333, 0.93333333, 1.])

```
res.mean()
```

0.96

```
from sklearn.model_selection import GridSearchCV
model1=GridSearchCV(SVC(gamma='auto'),{'C':[1,5,10,20],'kernel':['rbf','linear']},return_train_score=False)
model1.fit(X,y)
model1.best_score_
```

0.96

```
model1.best_params_
{'C': 1, 'kernel': 'rbf'}
```

```
y_predicted=model1.predict(X_test) from
sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_predicted) cm
```

array([[9, 0, 0], [0, 11, 0], [0, 1, 9]])

WEEK-14

Aim: Write a program to perform Model diagnosis and tuning on any real time dataset using ensemblers:

a)RandomForestClassifier

b)BaggingClassifier

c)GradientBoostClassifier

Description:

Tune Hyperparameters with GridSearchCV

Grid Search uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters. This makes the processing time-consuming and expensive based on the number of hyperparameters involved.

In GridSearchCV, along with Grid Search, cross-validation is also performed. Cross-Validation is used while training the model. As we know that before training the model with data, we divide the data into two parts – **train data** and **test data**. In cross-validation, the process divides the train data further into two parts – the **train data** and the **validation data**.

a) RandomForestClassifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Syntax:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=
0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=
True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
```

b)BaggingClassifier

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a

decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

```
class sklearn.ensemble.BaggingClassifier(estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0, base_estimator='deprecated')
```

c)GradientBoostClassifier

This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes` regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced.

```
class sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
df=pd.read_csv('titanic.csv')
df.head()
```

index	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925	NaN	S
3	4	1	1	Futelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	NaN	S

```
df.info()
```

#	Column	Non-Null
Count	Dtype	

```

-----
0  PassengerId    891 non-null    int64
1  Survived      891 non-null    int64
2  Pclass        891 non-null    int64
3  Name          891 non-null    object
4  Sex           891 non-null    object
5  Age           714 non-null    float64
6  SibSp         891 non-null    int64
7  Parch         891 non-null    int64
8  Ticket        891 non-null    object
9  Fare          891 non-null    float64
10 Cabin         204 non-null    object
11 Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)

```

```
df.drop(['PassengerId', 'Name', 'Ticket', 'Fare', 'Cabin'], axis=1, inplace=True)
df.columns
```

```
Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Embarked'],
      dtype='object')
```

```
df1=df.fillna({'Age':df['Age'].mean()})
df1.corr()
```

index	Survived	Pclass	Age	SibSp	Parch
Survived	1.0	-0.33848103596101514	-0.06980851528714313	-0.035322498885735576	0.08162940708348335
Pclass	-0.33848103596101514	1.0	-0.3313387740824206	0.08308136284568686	0.018442671310748508
Age	-0.06980851528714313	-0.3313387740824206	1.0	-0.23262458644442344	-0.17919091526675796
SibSp	-0.035322498885735576	0.08308136284568686	-0.23262458644442344	1.0	0.41483769862015624
Parch	0.08162940708348335	0.018442671310748508	-0.17919091526675796	0.41483769862015624	1.0

```
df1.drop(['Embarked', 'SibSp', 'Parch'], axis=1, inplace=True)
df1.columns
```

```
Index(['Survived', 'Pclass', 'Sex', 'Age'], dtype='object')
```

```

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df1['Sex']=le.fit_transform(df1['Sex'])

X=df1[['Pclass', 'Sex']]
y=df1['Survived']
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)

```

a) USING RANDOM FOREST CLASSIFIER:

```

from sklearn.ensemble import RandomForestClassifier
model_RFC = RandomForestClassifier()
param_grid = {'n_estimators': [20, 50, 100, 200],
              'max_depth': [None, 2, 5, 10],
              'min_samples_split': [2, 5, 10]}
grid_search_RFC = GridSearchCV(model_RFC, param_grid, cv=5)
grid_search_RFC.fit(X_train, y_train)
best_model = grid_search_RFC.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print("Best parameters are : ", grid_search_RFC.best_params_)

```

Accuracy: 0.7653631284916201

Best parameters are : {'max_depth': 2, 'min_samples_split': 5, 'n_estimators': 20}

b) USING BAGGING CLASSIFIER:

```

from sklearn.ensemble import BaggingClassifier
from sklearn.svm import SVC
bagging_clf = BaggingClassifier(estimator=SVC())
param_grid = {'n_estimators': [10, 50, 100], 'max_samples': [0.5, 1.0], 'max_features': [0.5, 1.0],
              'estimator__C': [0.1, 1, 10], 'estimator__kernel': ['linear', 'poly', 'rbf']}
grid_search_BC = GridSearchCV(bagging_clf, param_grid, cv=5)
grid_search_BC.fit(X_train, y_train)
best_model = grid_search_BC.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print("Best parameters are : ", grid_search_BC.best_params_)

```

Accuracy: 0.7821229050279329

Best parameters are : {'estimator__C': 10, 'estimator__kernel': 'rbf', 'max_features': 0.5, 'max_samples': 0.5, 'n_estimators': 100}

c) USING GRADIENT BOOST CLASSIFIER:

```

from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
param_grid = {'n_estimators': [10, 50, 100], 'learning_rate': [0.1, 0.01, 0.001], 'max_depth': [2, 3, 4],
              'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'max_features': [None, 'sqrt', 'log2']}
grid_search_GBC = GridSearchCV(gbc, param_grid, cv=5)
grid_search_GBC.fit(X_train, y_train)
best_model = grid_search_GBC.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print("Best parameters are : ", grid_search_GBC.best_params_)

```

Accuracy: 0.7653631284916201

Best parameters are : {'learning_rate': 0.1, 'max_depth': 2, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 10}

WEEK-15

Aim: Implement a single neural network and test for different logic gates

Description:

A single-layer neural network represents the most simple form of neural network, in which there is only one layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes, or in some cases, one receiving node. This single-layer design was part of the foundation for systems which have now become much more complex.

One of the early examples of a single-layer neural network was called a “perceptron.” The perceptron would return a function based on inputs, again, based on single neurons in the physiology of the human brain. In some senses, perceptron models are much like “logic gates” fulfilling individual functions: A perceptron will either send a signal, or not, based on the weighted inputs. Another type of single-layer neural network is the single-layer binary linear classifier, which can isolate inputs into one of two categories.

Single-layer neural networks can also be thought of as part of a class of feedforward neural networks, where information only travels in one direction, through the inputs, to the output. Again, this defines these simple networks in contrast to immensely more complicated systems, such as those that use backpropagation or gradient descent to function.

Program:

```
from keras.models import Sequential

from keras.layers import Dense

#Define the neural network

model=Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))

#Compile the model

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

#Define the training data

X_train_AND=[[0, 0], [0, 1], [1, 0], [1, 1]]
y_train_AND=[0, 0, 0, 1]

X_train_OR=[[0, 0], [0, 1], [1, 0], [1, 1]]
y_train_OR=[0, 1, 1, 1]

X_train_XOR=[[0, 0], [0, 1], [1, 0], [1, 1]]
y_train_XOR=[0, 1, 1, 0]

# Train the model for each logic gate

model.fit(X_train_AND, y_train_AND, epochs=1008, verbose=0)
model.fit(X_train_OR, y_train_OR, epochs=1800, verbose=0)
model.fit(X_train_XOR, y_train_XOR, epochs=1000, verbose=0)

#Test the model for each logic gate
print(model.predict(X_train_AND))
print("AND gate:")
print("OR gate:")
print(model.predict(X_train_OR))
print(model.predict(X_train_XOR))
print("XOR gate:")
```

output:

```
1/1 [=====] - 0s 70ms/step
[[0.31626716]
 [0.55467665]
 [0.4234677 ]
 [0.6641874 ]]
AND gate:
OR gate:
1/1 [=====] - 0s 41ms/step
[[0.31626716]
```

```
[0.55467665]
[0.4234677 ]
[0.6641874 ]]
1/1 [=====] - 0s 39ms/step
[[0.31626716]
 [0.55467665]
 [0.4234677 ]
 [0.6641874 ]]
```

XOR gate: