

**Thakur degree college of science and commerce
Kandivali (east), Mumbai.**

Department of Computer Science

Data Science journal

SY.Bsc.CS

Academic year 2021-2022

INDEX

Practical List
1. Write a program in python to display the execution of Lists - Demo the use of append(), insert(), extend(), remove(), pop().
2. Write a program in python to display the execution of Tuples. Create a student attendance table to demo the use of tuples.
3. Write a program in python to display the execution of Sets. Demo the use of sets to display employee records of a company.
4. Write a program in python to display the execution of Dictionary. Create a dictionary set of and find the employee having highest and lowest salary.
5. Write a program in python to display the execution of Strings. Perform concat, iteration, membership and methods like upper(), join(), split(), find(), replace().
6. Compute a program in python using Matplotlib usage.
7. Compute a program in python using Numpy usage.
8. Compute a program in python using Pandas usage.
9. Implement the PCA using Python.
10. Consider a set of data points and train your model in python. The model should have 80% training data and 20% of test data.
11. Implement the supervised learning algorithm: SVM in python.
12. Implement the unsupervised learning algorithm: K means in python.
13. Implement the classification algorithm : K nearest neighbors in python.
14. Implement the Regression algorithm: Linear Regression in python.

Practical No 1

Aim: Write a program in python to display the execution of Lists - Demo the use of append(), insert(), extend(), remove(), pop().


THEORY:

- **LIST:** Lists are used to store multiple items in a single variable.
- **Append ():** Adds its argument as a single element to the end of a list. The length of the list increases by one. If you append another list onto a list, the parameter list will be a single object at the end of the list.
- **Insert ():** It is an inbuilt function in Python that inserts a given element at a given index in a list.
- **Extend ():** Extend is also an inbuilt method in python that can be used with a list or array to extend the items inside it. It is almost a similar append method of the list. When we use the extend method for array it will take two arrays and append another array at the end of the first array.
- **Remove ():** remove () is an inbuilt function in Python programming language that removes a given object from the list. It does not return any value.
- **Pop ():** The pop() method removes and returns the element at the given index (passed as an argument) from the list.

SOURCE CODE:

```
#creating list
A=[1,23,'Hi','python']
print(A)
#append()use to append the value in list
A.append(6)
print(A)
#insert() use to insert new value at a given index in the list.
A.insert(2, 1.5)
print(A)
#Remove() use to remove/delete a value from the list
A.remove(1.5)
print(A)
#POP() use to delete a value from a particular index(write index value)
A.pop(2)
print(A)
#extend() use to extend the value in list
B=['MEGHNA','BYE']
A.extend(B)
print(A)
```


1.

 *syp1.py - C:\Users\Meghna\Desktop\DATA SCI\syp1.py (3.9.1)*

File Edit Format Run Options Window Help

```
#creating list
A=[1,23,'Hi','python']
print(A)
```

OUTPUT:

 *IDLE Shell 3.9.1*

File Edit Shell Debug Options Window Help

Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

>>>

===== RESTART: C:\Users\Meghna\Desktop\DATA SCI\syp1.py =====

```
[1, 23, 'Hi', 'python']
```

2.

```
#append() use to append the value in list
A.append(6)
print(A)
```

OUTPUT:

```
[1, 23, 'Hi', 'python', 6]
```

3.

```
#insert() use to insert new value at a given index in the list.
A.insert(2, 1.5)
print(A)
```

OUTPUT:

```
[1, 23, 1.5, 'Hi', 'python', 6]
```

4.

```
#Remove() use to remove/delete a value from the list
A.remove(1.5)
print(A)
```

OUTPUT:

```
[1, 23, 'Hi', 'python', 6]
```

5.

```
#POP() use to delete a value from a particular index(write index value)
A.pop(2)
print(A)
```

OUTPUT:

```
[1,| 23, 'python', 6]
```

6.

```
#extend() use to extend the value in list
B= ['MEGHNA', 'BYE']
A.extend(B)
print(A)
```

OUTPUT:

```
[1, 23, 'python', 6, 'MEGHNA', 'BYE']
```

Practical No 2

Aim: Write a program in python to display the execution of Tuples. Create a student attendance table to demo the use of tuples.

THEORY:

- TUPLE:

Python Tuple is used to store the sequence of immutable python objects. Tuple is similar to lists since the value of the items stored in the list can be changed whereas the tuple is immutable and the value of the items stored in the tuple can not be changed.

SOURCE CODE:

```
att=()
start = True
while(start):
    name = input("Enter your name : ")
    roll = input("Enter your Rollno : ")
    lec = input("Enter your lec attended : ")
    stud = ("name: "+name,"roll no : "+roll,"lec attended : "+lec)
    att += stud
    req = int(input("enter 1 to insert more data or enter 2 to see the available data:
"))
    if(req==1):
        start = True
    else:
        print(att)
        start = False
```

OUTPUT:

```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Meghna\Desktop\DATA SCI\syp2.py =====
Enter your name : MEGHNA
Enter your Rollno : 333
Enter your lec attended : 4
enter 1 to insert more data or enter 2 to see the available data: 1
Enter your name : ABC
Enter your Rollno : 004
Enter your lec attended : 5
enter 1 to insert more data or enter 2 to see the available data: 2
('name: MEGHNA', 'roll no : 333', 'lec attended : 4', 'name: ABC', 'roll no : 004', 'lec attended : 5')
>>> |

```

SOURCE CODE:

```

Roll_no = ("1","2","3","4","5","6")

Name = ("meghna","bala","cheta","demon","eli","quinto")

Att = ()

DataS = ()

ReTry = True

print("\nStudent attendance :\n")

from typing import Tuple

import numpy as np

DataS = ((roll_no) , (name))

N = np.matrix(DataS)

print(N)

print(Att)

print("\n")

reEntry = input("Do you want add any new entries (Y/N):")

if reEntry.upper() == "N":

    pass

else :

    while(ReTry):

        print("\n\tInsert new data:")

        Name = input("Name :")

        Roll_no = input("Roll NO :")

        N =(str(Name),)

        R = (str(Roll_no),)

```



```

roll_no += tuple(R)

name += tuple(N)

DataS = ((roll_no) , (name))

N = np.matrix(DataS)

print(N)

z = input("Continue.. (Y/N)")

if z.upper() == "Y":

    ReTry = True

else:

    ReTry = False

for i in range(0,len(roll_no)):

    log = input(f"(A/P) roll no {roll_no[i]} : ")

    Att += ((roll_no[i],name[i],log.upper()),)

N = np.matrix(Att)

print("\n")

print(N)

```

OUTPUT:



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Student attendance :

[['1' '2' '3' '4' '5' '6']
 ['meghna' 'bala' 'cheta' 'demon' 'eli' 'quinto']]
()

Do you want add any new entries (Y/N):Y

      Insert new data:
Name :MEGHANA
Roll NO :333
[['1' '2' '3' '4' '5' '6' '333']
 ['meghna' 'bala' 'cheta' 'demon' 'eli' 'quinto' 'MEGHANA']]
Continue.. (Y/N)N
(A/P) roll no 1 : P
(A/P) roll no 2 : A
(A/P) roll no 3 : P
(A/P) roll no 4 : A
(A/P) roll no 5 : P
(A/P) roll no 6 : A
(A/P) roll no 333 : P

[['1' 'meghna' 'P']
 ['2' 'bala' 'A']
 ['3' 'cheta' 'P']
 ['4' 'demon' 'A']
 ['5' 'eli' 'P']
 ['6' 'quinto' 'A']
 ['333' 'MEGHANA' 'P']]
>>> |

```

Ln: 37 Col: 4

Practical No 3

Aim: Write a program in python to display the execution of Sets. Demo the use of sets to display employee records of a company.

THEORY:

SET:

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Methods for Sets

Adding elements

Insertion in set is done through `set.add()` function, where an appropriate record value is created to store in the hash table.

Union

Two sets can be merged using `union()` function or `|` operator. Both Hash Table values are accessed and traversed with merge operation perform on them to combine the elements, at the same time duplicates are removed.

Intersection

This can be done through `intersection()` or `&` operator. Common Elements are selected.

Difference

To find difference in between sets. Similar to find difference in linked list. This is done through `difference()` or `-` operator.

Clearing sets

`Clear()` method empties the whole set.

SOURCE CODE:

```
# Python program to
# demonstrate sets
# Same as {"a", "b", "c"}
myset = set(["a", "b", "c"])
print(myset)

# Adding element to the set
myset.add("d")
print(myset)

# A Python program to
# demonstrate adding elements
# in a set

# Creating a Set
people = {"Jay", "Idrish", "Archi"}

print("People:", end = " ")

print(people)

# This will add Daxit
# in the set
people.add("Daxit")

# Adding elements to the
# set using iterator
for i in range(1, 6):
    people.add(i)

print("\nSet after adding element:", end = " ")

print(people)
```

```
# Python program to
# demonstrate intersection
# of two sets

set1 = set()
set2 = set()

for i in range(5):
    set1.add(i)

for i in range(3,9):
    set2.add(i)

# Intersection using
# intersection() function

set3 = set1.intersection(set2)

print("Intersection using intersection() function")

print(set3)

# Intersection using
# "&" operator

set3 = set1 & set2

print("\nIntersection using '&' operator")

print(set3)

# Python program to
# demonstrate difference
# of two sets

set1 = set()
set2 = set()
```

```
for i in range(5):

    set1.add(i)

for i in range(3,9):

    set2.add(i)

# Difference of two sets

# using difference() function

set3 = set1.difference(set2)

print(" Difference of two sets using difference() function")

print(set3)

# Difference of two sets

# using '-' operator

set3 = set1 - set2

print("\nDifference of two sets using '-' operator")

print(set3)

# Python program to

# demonstrate clearing

# of set

set1 = {1,2,3,4,5,6}

print("Initial set")

print(set1)

# This method will remove

# all the elements of the set

set1.clear()

print("\nSet after using clear() function")

print(set1)
```

OUTPUT:

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Meghna/Desktop/DATA SCI/SY3SETS.PY =====
{'a', 'b', 'c'}
{'a', 'b', 'c', 'd'}
People: {'Idrish', 'Jay', 'Archi'}

Set after adding element: {'Jay', 'Daxit', 1, 2, 3, 4, 5, 'Idrish', 'Archi'}
Intersection using intersection() function
{3, 4}

Intersection using '&' operator
{3, 4}
Difference of two sets using difference() function
{0, 1, 2}

Difference of two sets using '-' operator
{0, 1, 2}
Initial set
{1, 2, 3, 4, 5, 6}

Set after using clear() function
set()
>>>
```

Practical No 4

Aim: Write a program in python to display the execution of Dictionary. Create a dictionary set of and find the employee having highest and lowest salary.

THEORY:

A Python dictionary is an unordered collection of data values. Unlike other data types that hold only one value as an element, a Python dictionary holds a key: value pair. The Python dictionary is optimized in a manner that allows it to access values when the key is known.

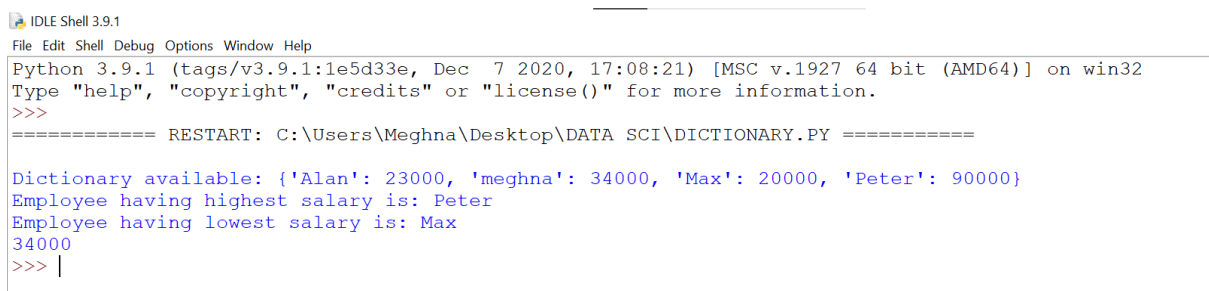
SOURCE CODE:

```
salary = {'Alan': 23000, 'meghna' : 34000, 'Max' : 20000, 'Peter' : 90000}

print("\nDictionary available:", salary)

#for getting highest salary
print("Employee having highest salary is:",max(salary, key= salary.get))
#for getting lowest salary
print("Employee having lowest salary is:",min(salary, key= salary.get))
#for getting particular employee's salary
print(salary.get('meghna'))
```

OUTPUT:



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Meghna\Desktop\DATA SCI\DICTIONARY.PY =====
Dictionary available: {'Alan': 23000, 'meghna': 34000, 'Max': 20000, 'Peter': 90000}
Employee having highest salary is: Peter
Employee having lowest salary is: Max
34000
>>> |
```

Practical No 5

Aim: Write a program in python to display the execution of Strings. Perform concat, iteration, membership and methods like upper(), join(), split(), find(), replace().

THEORY:

Strings in Python can be created using single quotes or double quotes or even triple quotes.

strings in Python are arrays of bytes representing unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

SOURCE CODE:

```
s0 = "orange"
s1 = "Hello this is python language"
s2 = "lets learn about strings in python"
#CONCATENATING --> adding two strings together
print("-----THIS IS CONCATENATION-----")
print(s1 + s2) #this is simply concatenating
print(s1 + ',' + s2) #to make it look pretty
#ITERATING --> going through the string character by character
print("-----THIS IS ITERATING-----")
for i in s0:
    print(i)
#upper() --> it will turn string to uppercase
print("-----UPPERCASE-----")
print(s0.upper())
#join() --> it will join the other string after each element while iterating
str1 = '->'
str2 = '1234'
print("-----JOINING-----")
print(str1.join(str2))
string = 'Hello'
print(" " + '.'.join(string))
#split() --> it will split the string word by word
print("-----SPLITTING-----")
print(s1.split()) #this splits word by word
print(s1.split(" ",2)) # this will split only first two words then take all other words in single inverted comma
#find() --> it will find the character or word you want and return the index
print("-----FINDING(returns index of the word you want)-----")
print(s1.find('this'))
print("-----REPLACING-----")
print(s1.replace('Hello','hey'))
```


OUTPUT

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/Meghna/Desktop/DATA SCI/SYP5STRING.PY =====
-----THIS IS CONCATENATION-----
Hello this is python languagelets learn about strings in python
Hello this is python language,lets learn about strings in python
-----THIS IS ITERATING-----
o
r
a
n
g
e
-----UPPERCASE-----
ORANGE
-----JOINING-----
1->2->3->4
H.e.l.l.o
-----SPLITTING-----
['Hello', 'this', 'is', 'python', 'language']
['Hello', 'this', 'is python language']
-----FINDING(returns index of the word you want)-----
6
-----REPLACING-----
hey this is python language
>>>
```

Practical No 6

Aim: Compute a program in python using Matplotlib usage.

THEORY:

Matplotlib is the most popular package or library in Python which is used for data visualization. By using this library we can generate plots and figures.

SOURCE CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('pokemon.csv') #usage of pandas
df.head()
df.drop('#', inplace=True, axis=1)
df.head()
df.shape #usage of numpy

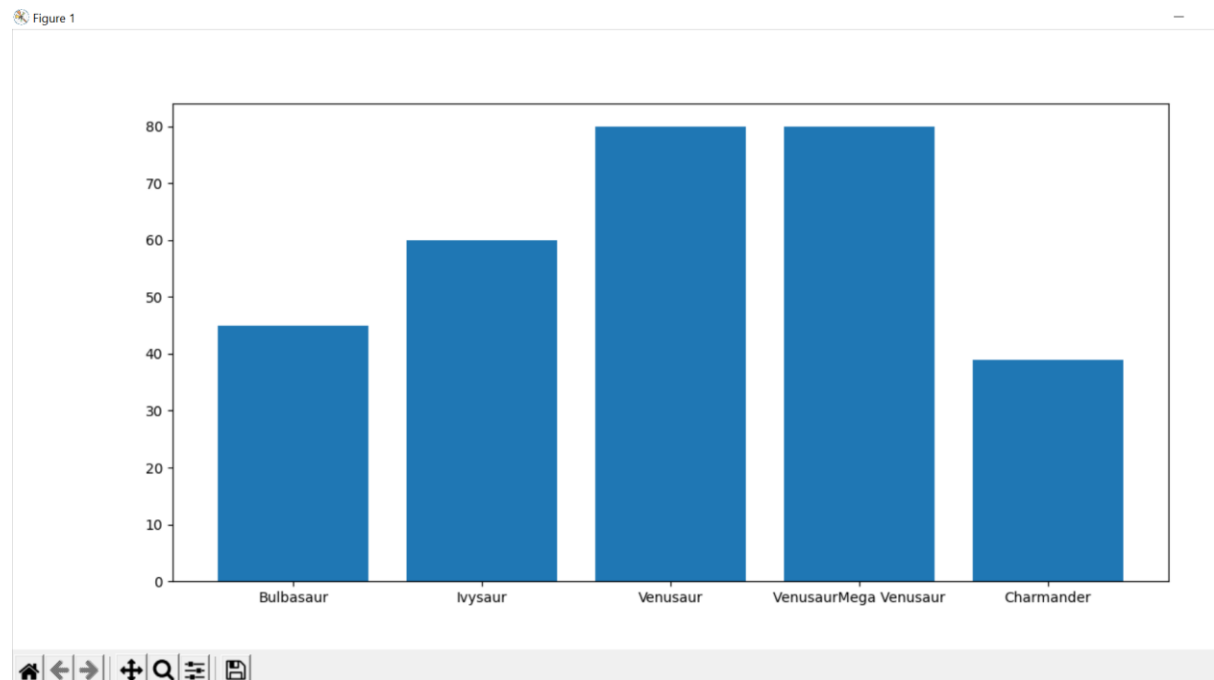
# frame['DataFrame Column'] = frame['DataFrame Column'].apply(str)
df['Type 2'] = df['Type 2'].apply(str)

df.info()

newdf = pd.DataFrame(df, columns=['Name', 'HP'])
newdf.head()
newdf.shape
plt.rcParams["figure.figsize"] = (20, 9)
plt.bar(newdf['Name'].head(), newdf['HP'].head())
plt.show()
```

OUTPUT:

```
===== RESTART: C:\Users\Meghna\Desktop\DATASCI\pokemon.py =====  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 800 entries, 0 to 799  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Name        800 non-null   object  
1   Type 1      800 non-null   object  
2   Type 2      800 non-null   object  
3   Total       800 non-null   int64  
4   HP          800 non-null   int64  
5   Attack      800 non-null   int64  
6   Defense     800 non-null   int64  
7   Sp. Atk     800 non-null   int64  
8   Sp. Def     800 non-null   int64  
9   Speed       800 non-null   int64  
10  Generation  800 non-null   int64  
11  Legendary   800 non-null   bool  
dtypes: bool(1), int64(8), object(3)  
memory usage: 69.7+ KB
```



Practical No 7

Aim: Compute a program in python using Numpy usage.

THEORY:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with arrays.

SOURCE CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('pokemon.csv') #usage of pandas
df.head()
df.drop('#', inplace=True, axis=1)
df.head()
df.shape #usage of numpy

# frame['DataFrame Column']= frame['DataFrame Column'].apply(str)
df['Type 2'] = df['Type 2'].apply(str)

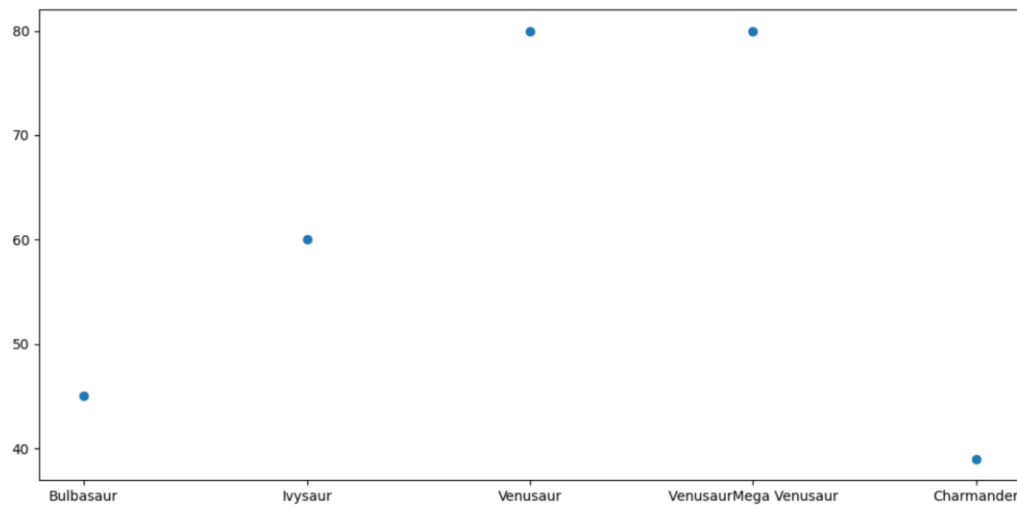
df.info()

newdf = pd.DataFrame(df, columns=['Name','HP'])
newdf.head()
newdf.shape
plt.rcParams["figure.figsize"] = (20, 9)
plt.scatter(newdf['Name'].head(), newdf['HP'].head())
plt.show()
```

OUTPUT:

```
===== RESTART: C:\Users\Meghna\Desktop\DATASCI\pokemon.py =====  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 800 entries, 0 to 799  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Name        800 non-null   object  
1   Type 1      800 non-null   object  
2   Type 2      800 non-null   object  
3   Total       800 non-null   int64  
4   HP          800 non-null   int64  
5   Attack      800 non-null   int64  
6   Defense     800 non-null   int64  
7   Sp. Atk     800 non-null   int64  
8   Sp. Def     800 non-null   int64  
9   Speed       800 non-null   int64  
10  Generation  800 non-null   int64  
11  Legendary   800 non-null   bool  
dtypes: bool(1), int64(8), object(3)  
memory usage: 69.7+ KB
```

Figure 1



Practical No 8

Aim: Compute a program in python using Pandas usage.

THEORY:

Pandas is a Python library that is used for faster data analysis, data cleaning and data pre-processing. Pandas is built on top of numpy.

SOURCE CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('pokemon.csv') #usage of pandas
df.head()
df.drop('#', inplace=True, axis=1)
df.head()
df.shape #usage of numpy

# frame['DataFrame Column']= frame['DataFrame Column'].apply(str)
df['Type 2'] = df['Type 2'].apply(str)

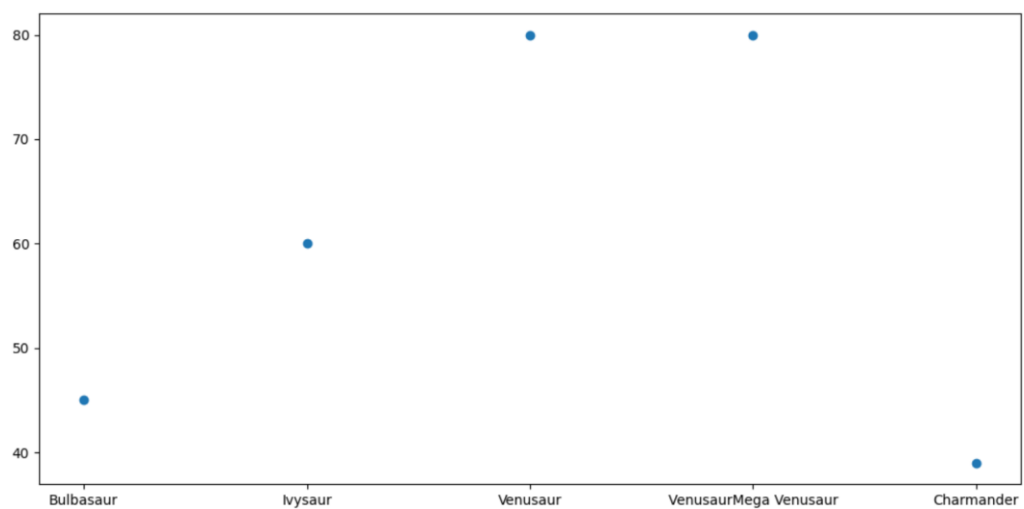
df.info()

newdf = pd.DataFrame(df, columns=['Name','HP'])
newdf.head()
newdf.shape
plt.rcParams["figure.figsize"] = (20, 9)
plt.scatter(newdf['Name'].head(), newdf['HP'].head())
plt.show()
```

OUTPUT:

```
===== RESTART: C:\Users\Meghna\Desktop\DATASCI\pokemon.py =====  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 800 entries, 0 to 799  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Name        800 non-null    object  
1   Type 1      800 non-null    object  
2   Type 2      800 non-null    object  
3   Total       800 non-null    int64  
4   HP          800 non-null    int64  
5   Attack      800 non-null    int64  
6   Defense     800 non-null    int64  
7   Sp. Atk     800 non-null    int64  
8   Sp. Def     800 non-null    int64  
9   Speed       800 non-null    int64  
10  Generation  800 non-null    int64  
11  Legendary   800 non-null    bool  
dtypes: bool(1), int64(8), object(3)  
memory usage: 69.7+ KB
```

Figure 1



Practical No 9

Aim: Implement the PCA using Python.

THEORY:

PCA stands for Principal Component Analysis which is dimensionality reduction method which is used to reduce the dimension of large data set.

In PCA we transform a large data set of variables into small parts which still contains most of information of a large set.

SOURCE CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
#Standardized the dataset features
from sklearn.preprocessing import StandardScaler
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
# load dataset into Pandas DataFrame
df = pd.read_csv(url, names=['sepal length', 'sepal width', 'petal length', 'petal width', 'target'])
df.head()
mean_sepal_length = df['sepal length'].mean()
mean_sepal_length
std_dev_sepal_length = df['sepal length'].std()
std_dev_sepal_length
(5.1-mean_sepal_length)/std_dev_sepal_length
feature = ['sepal length', 'sepal width', 'petal length', 'petal width']
x = df.loc[:,feature]
y = df.loc[:, 'target']
x = StandardScaler().fit_transform(x)
standardised_values = pd.DataFrame(x, columns=feature)
standardised_values
#PCA PROJECTION OF 2D
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pct = pca.fit_transform(x)
principal_df = pd.DataFrame(pct, columns=['pc1', 'pc2'])
#CONCATINATING TARGET
finaldf= pd.concat([principal_df, df[['target']]], axis=1)
finaldf.head()
#PLOTING 2 DIEMENTIONAL DATA
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets, colors):
    indicesToKeep = finaldf['target'] == target
    ax.scatter(finaldf.loc[indicesToKeep, 'pc1'],
               finaldf.loc[indicesToKeep, 'pc2'],
               c = color,
               s = 50)
ax.legend(targets)
ax.grid()
#SHOWS TWO SEPARATE VARIANCE PERCENT
pca.explained_variance_ratio_
```


OUTPUT

1)

```
*IDLE Shell 3.9.1*
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Meghna\Desktop\DATA SCI\PCA.py =====
>>> df.head()
   sepal length  sepal width  petal length  petal width  target
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
>>> |
```

2)

```
>>> standardised_values
   sepal length  sepal width  petal length  petal width
0      -0.900681      1.032057     -1.341272     -1.312977
1      -1.143017     -0.124958     -1.341272     -1.312977
2      -1.385353      0.337848     -1.398138     -1.312977
3      -1.506521      0.106445     -1.284407     -1.312977
4      -1.021849      1.263460     -1.341272     -1.312977
...
145      1.038005     -0.124958      0.819624      1.447956
146      0.553333     -1.281972      0.705893      0.922064
147      0.795669     -0.124958      0.819624      1.053537
148      0.432165      0.800654      0.933356      1.447956
149      0.068662     -0.124958      0.762759      0.790591

[150 rows x 4 columns]
>>> |
```

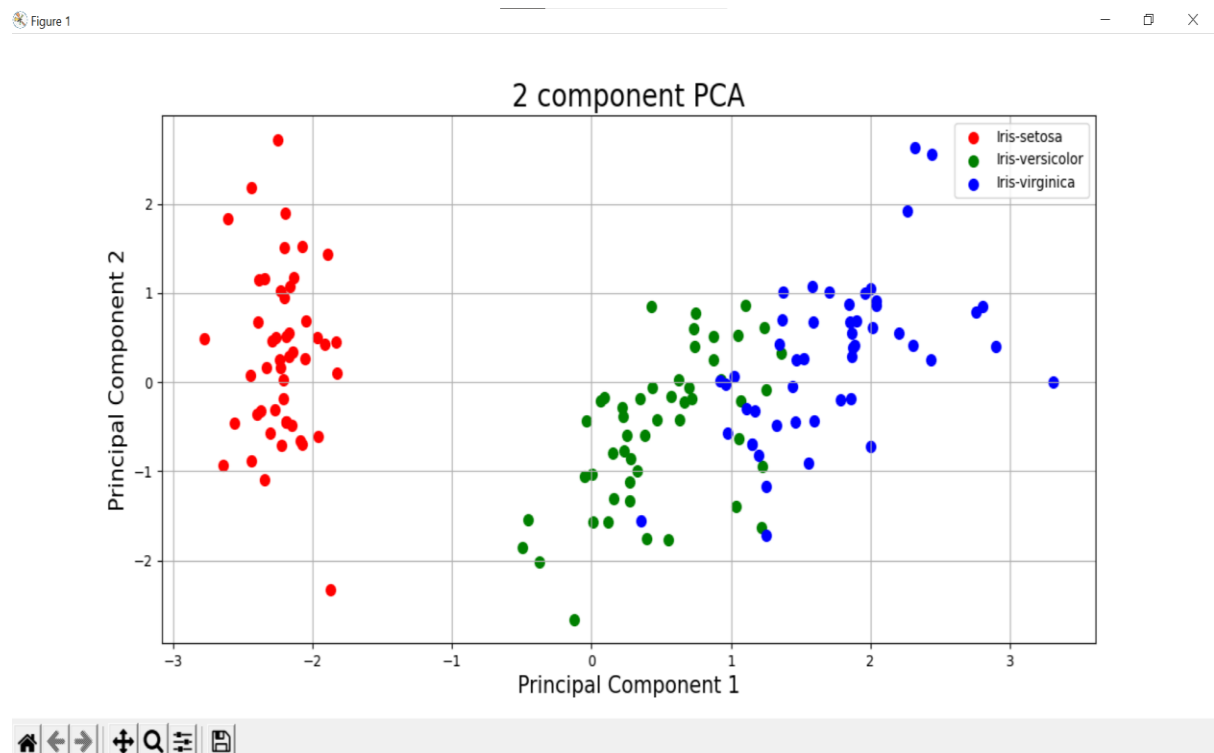
3)

```
>>> finaldf.head()
      pc1      pc2      target
0 -2.264542  0.505704  Iris-setosa
1 -2.086426 -0.655405  Iris-setosa
2 -2.367950 -0.318477  Iris-setosa
3 -2.304197 -0.575368  Iris-setosa
4 -2.388777  0.674767  Iris-setosa
>>>
```

4)

```
>>> pca.explained_variance_ratio_
array([0.72770452, 0.23030523])
>>>
```

5)



Practical No 10

Aim: Consider a set of data points and train your model in python. The model should have 80% training data and 20% of test data.

THEORY:

The training data set is a grouped set of examples that are used to fit the parameters. The test data set is the last evaluation of the final model fit on the training data set. So like the last test before it's the real deal. Testing against each other ensures the machine learning model will be more accurate.

SOURCE CODE:

```
import pandas as pd

from sklearn.datasets import load_iris

iris_data = load_iris()

df = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)

print(df)

from sklearn.model_selection import train_test_split

training_data, testing_data = train_test_split(df, test_size=0.2, random_state=25)

#random state is used to keep the data constant if it is 0 then everytime u run the
program the value of data changes.

print(f"No. of training examples: {training_data.shape[0]}")

print(f"No. of testing examples: {testing_data.shape[0]}")

training_data.head()
```

OUTPUT:

```
===== RESTART: C:\Users\Meghna\Downloads\train_test_split.py =====
    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
..                    ...             ...             ...             ...
145                  6.7             3.0             5.2             2.3
146                  6.3             2.5             5.0             1.9
147                  6.5             3.0             5.2             2.0
148                  6.2             3.4             5.4             2.3
149                  5.9             3.0             5.1             1.8

[150 rows x 4 columns]
No. of training examples: 120
No. of testing examples: 30
>>>
```

Practical No 11

Aim: Implement the supervised learning algorithm: SVM in python.

THEORY:

A Support Vector Machine was first introduced in the 1960s and later improvised in the 1990s. It is a supervised learning machine learning classification algorithm

An SVM is implemented in a slightly different way than other machine learning algorithms. It is capable of performing classification, regression and outlier detection.

Support Vector Machine is a discriminative classifier that is formally designed by a separative hyperplane. It is a representation of examples as points in space that are mapped so that the points of different categories are separated by a gap as wide as possible. In addition to this, an SVM can also perform non-linear classification.

SOURCE CODE:

```
import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import pandas as pd

apples_oranges = pd.read_csv("apples_and_oranges.csv")

apples_oranges.head()

# create a dictionary to colour classes

color_dict = dict({'orange':'orange',

                   'apple':'green'})

# scatterplot

plt.xlabel('Weight')

plt.ylabel('Size')
```

```

plt.title('Sizes and Weights of apples and oranges')

sns.scatterplot(data=apples_oranges, x="Weight", y="Size", hue="Class", palette = color_dict)

plt.show()

# define input data

X = apples_oranges[["Weight", "Size"]]

# define target

y = apples_oranges.Class

# fitting the support vector machine using a linear kernel

from sklearn import svm

clf = svm.SVC(kernel = 'linear', C=10)

clf.fit(X, y)

b = clf.intercept_

w_1 = clf.coef_[0][0]

w_2 = clf.coef_[0][1]

b, w_1, w_2

# plotting the hyperplane and support vector lines

ax = plt.gca()

sns.scatterplot(data=apples_oranges, x="Weight", y="Size", hue="Class", palette = color_dict)

xlim = ax.get_xlim()

ylim = ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)

yy = np.linspace(ylim[0], ylim[1], 30)

YY, XX = np.meshgrid(yy, xx)

xy = np.vstack([XX.ravel(), YY.ravel()]).T

Z = clf.decision_function(xy).reshape(XX.shape)

ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,

          linestyle=['--', '-', '--'])

ax.scatter(clf.support_vectors_[0], clf.support_vectors_[1], s=100,

          linewidth=1, facecolors='none', edgecolors='k')

plt.show()

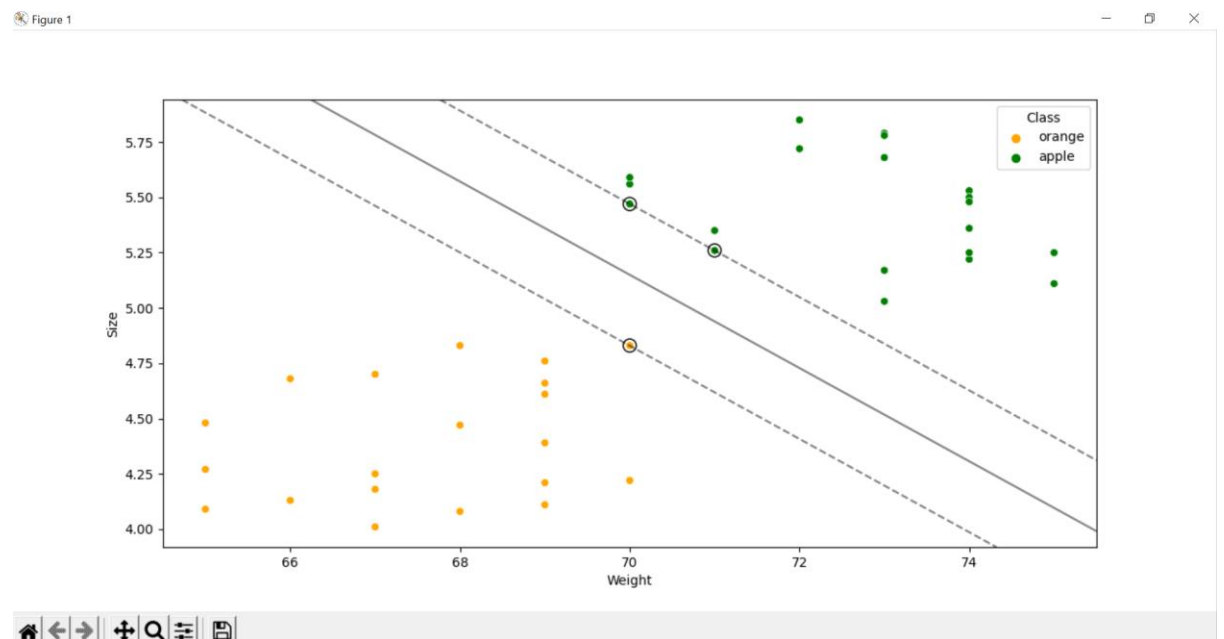
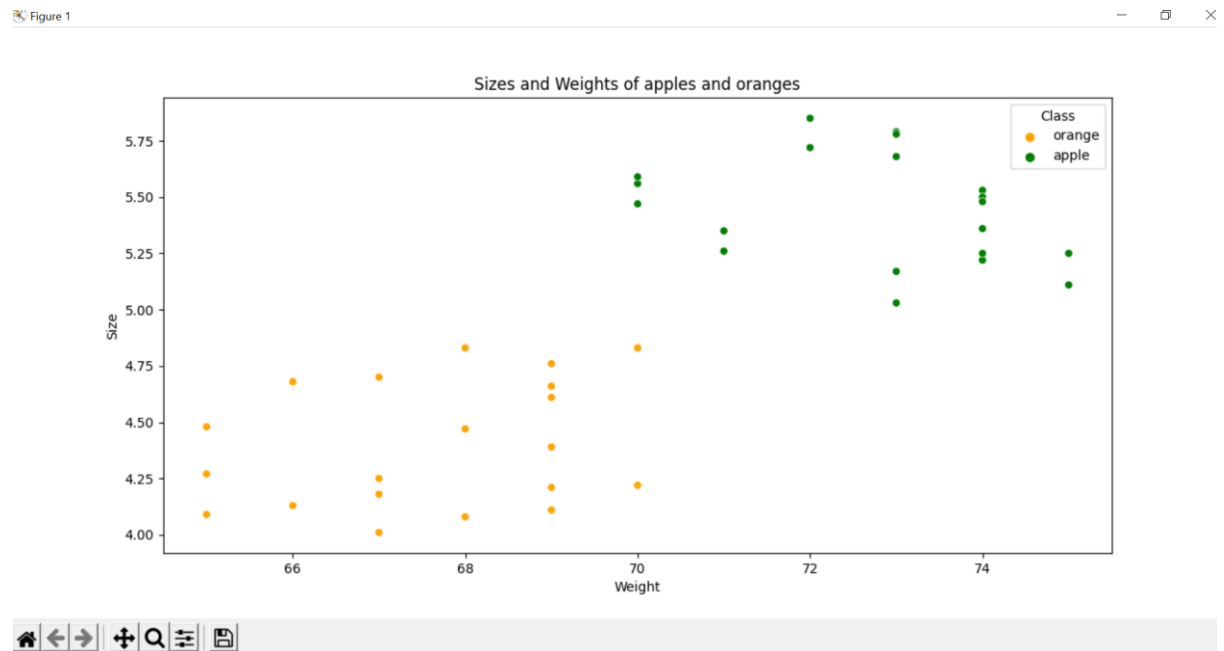
```

```
# obtain support vectors
```

```
clf.support_vectors_
```

```
clf.predict([[70, 4.6]])
```

OUTPUT :



Practical No 12

Aim: Implement the unsupervised learning algorithm: K means in python.

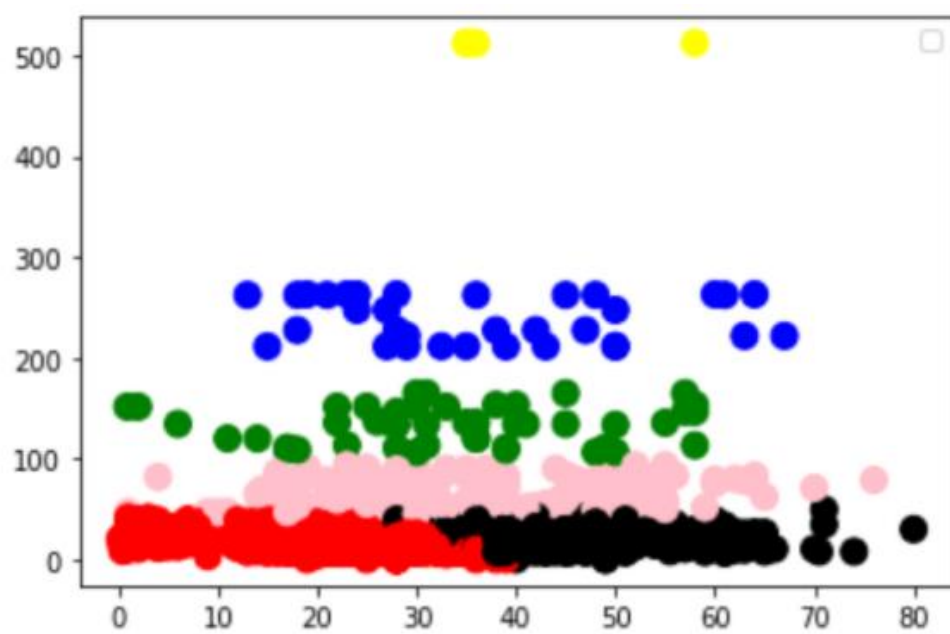
THEORY:

K-means algorithm in data mining starts with a set group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids. It halts creating and optimizing clusters when either: The centroids have stabilized — there is no change in their values because the clustering has been successful. The desired number of iterations has been achieved.

SOURCE CODE:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv('train_and_test2.csv')
df
kmeans_model = KMeans(n_clusters=6,max_iter=1000)
kmeans_model.fit(df[['Age','Fare']])
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=1000,
n_clusters=6,    n_init=10,    n_jobs=None,    precompute_distances='auto',
random_state=None, tol=0.0001, verbose=0)
color_dictionary = {0: 'red', 1: 'blue', 2: 'green' , 3: 'yellow',4: 'pink', 5: 'black'}
label_list = kmeans_model.labels_.tolist()
df['color'] = label_list

for i in color_dictionary:
    df['color'] = df['color'].replace(i,color_dictionary[i])
y = df['Fare']
x = df ['Age']
fig, ax = plt.subplots()
ax.scatter(x, y,c = df['color'], s = 100)
ax.legend()
plt.show()
OUTPUT:
```

Practical No 13

Aim: Implement the classification algorithm : K nearest neighbors in python.

THEORY:

The intuition behind the KNN algorithm is one of the simplest of all the supervised machine learning algorithms. It simply calculates the distance of a new data point to all other training data points. The distance can be of any type e.g Euclidean or Manhattan etc. It then selects the K-nearest data points, where K can be any integer. Finally it assigns the data point to the class to which the majority of the K data points belong.

SOURCE CODE:

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

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign column names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv(url, names=names)

dataset.head()

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

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)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
```

```

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

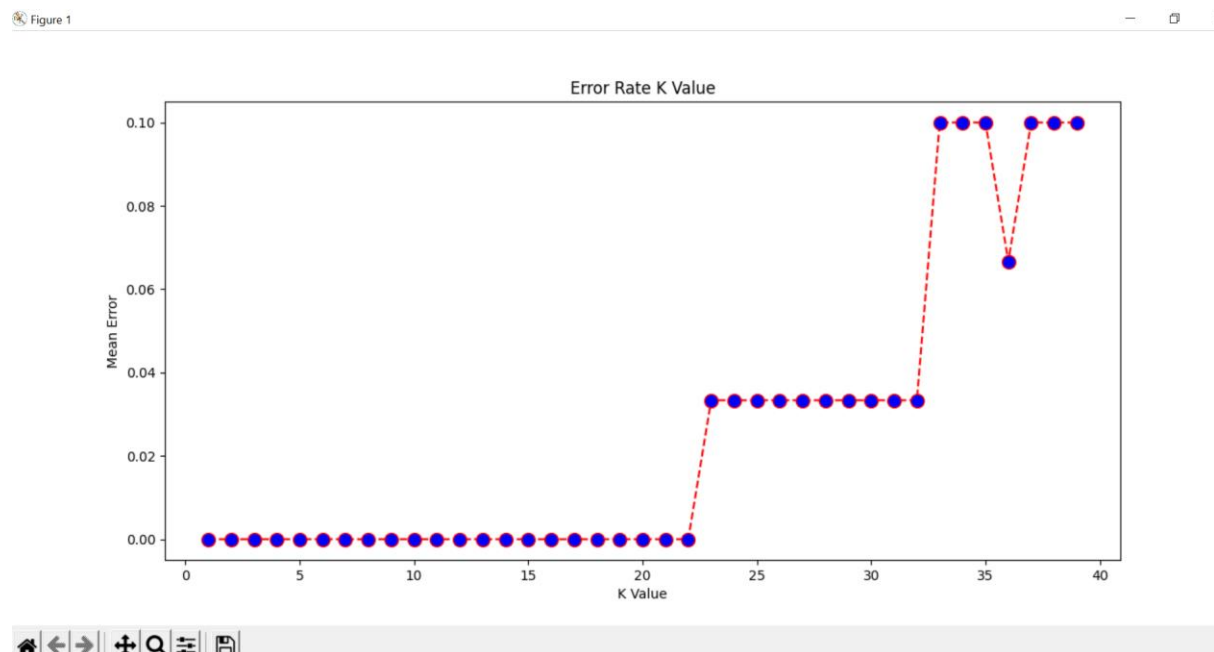
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')

```

OUTPUT



Practical No 14

Aim: Implement the Regression algorithm: Linear Regression in python.

THEORY:

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. Multiple regression is an extension of linear (OLS) regression that uses just one explanatory variable. MLR is used extensively in econometrics and nancial inference.

SOURCE CODE:

- Simple Linear Regression

```
import numpy as np

import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

def plot_regression_line(x, y, b):
```

```

# plotting the actual points as scatter plot
plt.scatter(x, y, color = "b",
            marker = "s", s = 30)

# predicted response vector
y_pred = b[0] + b[1]*x

# plotting the regression line
plt.plot(x, y_pred, color = "g")

# putting labels
plt.xlabel('x')
plt.ylabel('y')

# function to show plot
plt.show()

def main():

    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)

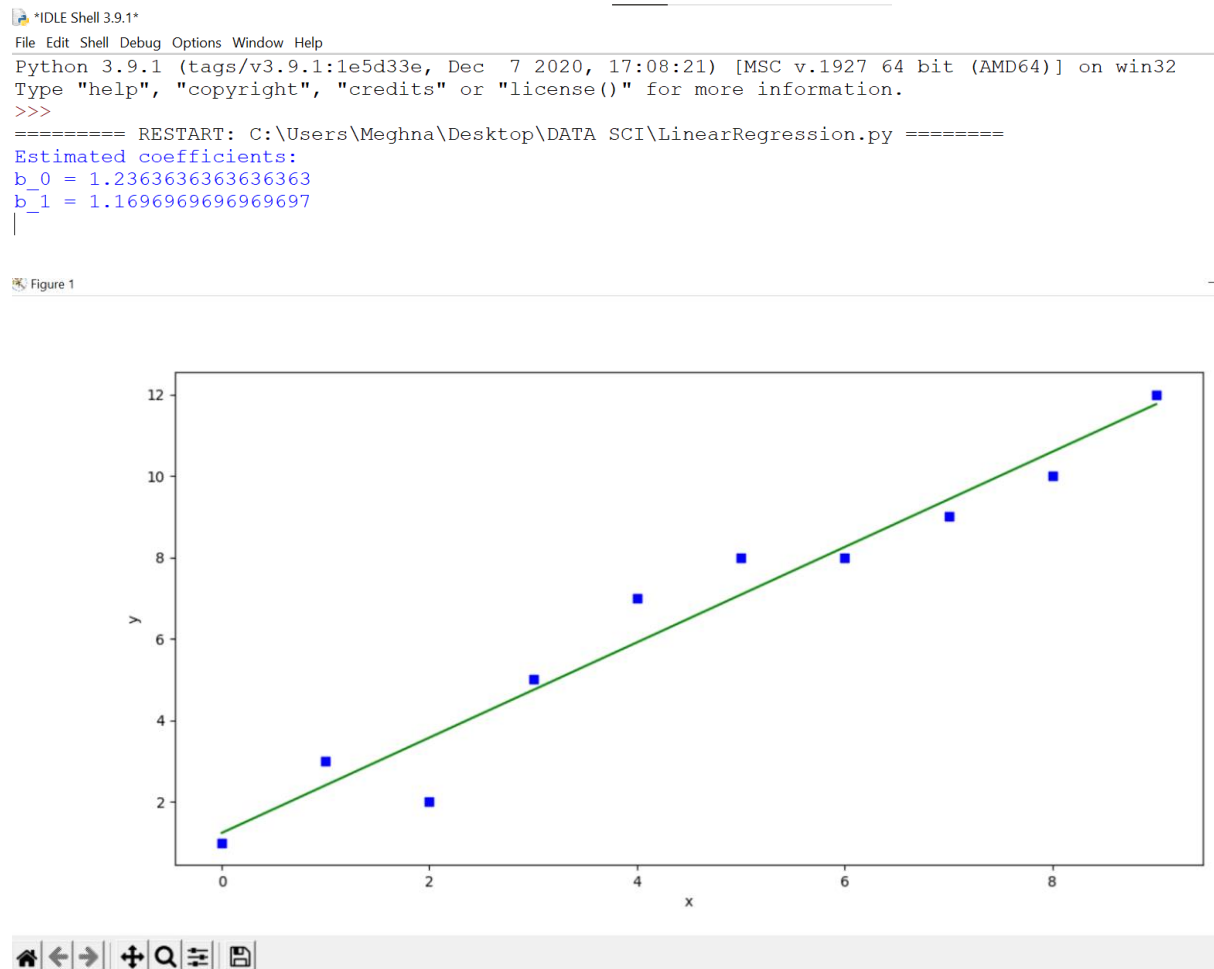
    print("Estimated coefficients:\nb_0 = {} \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()

```

OUTPUT:



- **Multiple linear Regression**

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics

# load the boston dataset
boston = datasets.load_boston(return_X_y=False)

# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
```

```

# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# regression coefficients
print('Coefficients: ', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))
# plot for residual error
## setting plot style
plt.style.use('fivethirtyeight')
## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
            color = "green", s = 10, label = 'Train data')
## plotting residual errors in test data
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
            color = "blue", s = 10, label = 'Test data')
## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)
## plotting legend
plt.legend(loc = 'upper right')
## plot title
plt.title("Residual errors")
## method call for showing the plot
plt.show()

```

OUTPUT:

```

===== RESTART: C:\Users\Meghna\Desktop\DATA SCI\mulLinearRegression.py =====
Coefficients:  [-8.95714048e-02  6.73132853e-02  5.04649248e-02  2.18579583e+00
 -1.72053975e+01  3.63606995e+00  2.05579939e-03 -1.36602886e+00
  2.89576718e-01 -1.22700072e-02 -8.34881849e-01  9.40360790e-03
 -5.04008320e-01]
Variance score: 0.7209056672661751
>>> |

```

Figure 1

