B. K. BIRLA COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS), KALYAN

DEPARTMENT OF INFORMATION TECHNOLOGY

Student Name:	
Student ID:	
Class:	
Subject:	

B. K. BIRLA COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS), KALYAN

(Affiliated to University of Mumbai)

KALYAN-MAHARASHTRA-421301

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that Mr/Ms____________________bearing Seat. No: **(_________)**, in class ________________has successfully completed practical of the subject____________________

Teacher's Signature: ____________________

Place:

Date: **College Seal**

INDEX

Assignment

1. **Print *"Hello World!"* in Python.**

```
[24] print("Hello World!")
```

```
Hello World!
```

2. **Print following:**
   "Hi!"
   "Python is fun to learn!"

```
[25] print("Hi!")
     print("Python is fun to learn!")
```

```
Hi!
Python is fun to learn!
```

3. **Write comment:**
   This is a comment
   Written in
   More than one line

```
[26] print("This is a comment \nwritten in more than one line")
```

```
This is a comment
written in more than one line
```

4. **Define variables a,b,c and assign them values integer, float and string(Machine Learning), respectively.**

```
[27] a = 1
     b = 0.2
     c = "Machine Learning"
     print(a)
     print(b)
     print(c)
```

```
1
0.2
Machine Learning
```

5. **Print following using c**
   I Like ML

```
[28] print("I like ML")
```

```
I like ML
```

6. **Add a and b, store it in d. Check type of d.**

```
[29] a = 2
     b = 3
     d = a+b
     print(d)
     print(type(d))

5
<class 'int'>
```

7. **Convert b to integer by casting.**

```
[30] b = 10.0
     b = int(b)
     print(b)

10
```

8. **Str1 = “HelloWorld!”. Slice the string to remove first 2 and last 2.**

```
[31] Str1 = "Hello World!"
     print(Str1[1:9])

ello Wor
```

9. **Convert the str1 to upper case.**

```
[32] print(Str1.upper())

HELLO WORLD!
```

10. **Remove white space from str1.**

```
[33] Str1 = Str1.replace(" ", "")
     print(Str1)

HelloWorld!
```

11. **Separate out two words from str1 and assign it to str2 and str3.**

```
[34] Str1 = "Hello World!"
     str2 = Str1.split(" ")[0]
     str3 = Str1.split(" ")[1]
     print(str2)
     print(str3)

Hello
World!
```

12. Combine str2 and str3 to form str4 so that str4 is same as str1.

```
[35] str4 = str2 + " " + str3
     print(str4)

     Hello World!
```

13. Let quantity=3, price=60, rate=20
Print following sentence using variables instead of actual values
"I bought 3 apples at 60 Rs with the rate of 20 Rs per apple.

```
[36] quantity = 3
     price = 60
     rate = 20
     print("I bought",quantity,"apples at",price,"Rs with the rate of",rate,"Rs per apple.")

     I bought 3 apples at 60 Rs with the rate of 20 Rs per apple.
```

14. Create a list named fruit containing names of 7 fruit.

```
[37] fruits = ["Apple","Strawberry","Mango","Cherry","Grapes","Watermelon","Papaya"]
     print(fruits)

     ['Apple', 'Strawberry', 'Mango', 'Cherry', 'Grapes', 'Watermelon', 'Papaya']
```

15. Print 3rd item of the list fruit.

```
[38] print(fruits[2])

     Mango
```

16. Print name of all elements from 4th to 6th elements of the list.

```
[39] print(fruits[2:6])

     ['Mango', 'Cherry', 'Grapes', 'Watermelon']
```

17. Check if "Kiwi" exist in the list.

```
[40] print("kiwi" in fruits)

     False
```

18. Change the first element of the list to "blackcurrant".

```
[41] fruits = ["Apple","Strawberry","Mango","Cherry","Grapes","Watermelon","Papaya"]
     fruits[0] = "Blackcurrent"
     print(fruits)

     ['Blackcurrent', 'Strawberry', 'Mango', 'Cherry', 'Grapes', 'Watermelon', 'Papaya']
```

19. Add new fruit to the end of the list.

```
[42] fruits.append("Banana")
     print(fruits)

['Blackcurrent', 'Strawberry', 'Mango', 'Cherry', 'Grapes', 'Watermelon', 'Papaya', 'Banana']
```

20. Remove 3rd element from the list.

```
[43] fruits.remove("Mango")
     print(fruits)

['Blackcurrent', 'Strawberry', 'Cherry', 'Grapes', 'Watermelon', 'Papaya', 'Banana']
```

21. Print all elements of the list on a new line using *for* statement.

```
[44] for i in fruits:
       print(i)

Blackcurrent
Strawberry
Cherry
Grapes
Watermelon
Papaya
Banana
```

22. Print all elements of the list on a new line using *while* statement.

```
fruits = ["Apple","Strawberry","Mango","Cherry","Grapes","Watermelon","Papaya"]
i = 0
while i < len(fruits):
  print(fruits[i])
  i+=1

Apple
Strawberry
Mango
Cherry
Grapes
Watermelon
Papaya
```

23. Sort the list alphabetically descending order.

```
[46] fruits = ["Apple","Strawberry","Mango","Cherry","Grapes","Watermelon","Papaya"]
     fruits.sort(reverse=True)
     print(fruits)

['Watermelon', 'Strawberry', 'Papaya', 'Mango', 'Grapes', 'Cherry', 'Apple']
```

24. Create a dictionary with the name dictbikes with keys: brand, model, engineCC and values: Royal Enfield, Thunder bird, 2021, 350.

```
[47] dictbikes = {"Brand":"Royal Enfield", "Model":"Thunder bird", "Year": 2021, "EngineCC":350}
     print(dictbikes)

     {'Brand': 'Royal Enfield', 'Model': 'Thunder bird', 'Year': 2021, 'EngineCC': 350}
```

25. Print model of the dictbikes.

```
[48] print(dictbikes["Model"])

     Thunder bird
```

26. Change the year to 2022.

```
[49] dictbikes["Year"] = 2022
     print(dictbikes)

     {'Brand': 'Royal Enfield', 'Model': 'Thunder bird', 'Year': 2022, 'EngineCC': 350}
```

27. Add new key: color and value: red

```
[50] dictbikes["color"] = "red"
     print(dictbikes)

     {'Brand': 'Royal Enfield', 'Model': 'Thunder bird', 'Year': 2022, 'EngineCC': 350, 'color': 'red'}
```

28. Remove year from dictionary.

```
[51] dictbikes.pop("Year")
     print(dictbikes)

     {'Brand': 'Royal Enfield', 'Model': 'Thunder bird', 'EngineCC': 350, 'color': 'red'}
```

29. Print values from the dictionary using *for* construct.

```
[52] for j,k in dictbikes.items():
       print(j,":",k)

     Brand : Royal Enfield
     Model : Thunder bird
     EngineCC : 350
     color : red
```

30. Write code to check and print if a is equal to b, less than b, greater than b for a=21, b=3.

```
[53] a = 21
     b = 3
     if(a>b):
       print("a is greater than b")
     elif(a<b):
       print("a is less than b")
     else:
       print("a is equal to b")
       print("a is equal to b")

     a is greater than b
```

31. Print all even number less than 20 using *while* construct.

```
[54] i = 0
     while i < 20:
       if i % 2 == 0:
         print(i)
       i += 1

0
2
4
6
8
10
12
14
16
18
```

32. Print all even number less than 20, except for those which are divisible by 3 using *while* statement.

```
[55] i = 0
     while i <= 20:
       if i % 2 == 0 and i % 3 != 0:
         print(i)
       i += 1

2
4
8
10
14
16
20
```

33. Print all even number less than 20 using *for* construct.

```
[56] for i in range(1,20):
       if i%2==0:
         print(i)

2
4
6
8
10
12
14
16
18
```

Practical No. 1

Title: Study of numPy and Pandas library in Python

Aim: To study numPy and Pandas library in Python.

Tools: anaconda, Python 3.7, Jupiter Notebook

Theory:

Q. What is numPy?

NumPy is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

Q. Why use numPy?

NumPy arrays are faster and more compact than Python lists. An array consumes less memory and is convenient to use. NumPy uses much less memory to store data and it provides a mechanism of specifying the data types. This allows the code to be optimized even further.

Q. What is pandas?

Pandas is an open-source library that allows to you perform data manipulation and analysis in Python. Pandas Python library offers data manipulation and data operations for numerical tables and time series. Pandas provide an easy way to create, manipulate, and wrangle the data. It is built on top of NumPy, means it needs NumPy to operate.

Q. Why use Pandas?

Use of Pandas in Python has following advantages:

- Easily handles missing data
- It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure
- It provides an efficient way to slice the data
- It provides a flexible way to merge, concatenate or reshape the data
- It includes a powerful time series tool to work with

In a nutshell, Pandas is a useful library in data analysis. It can be used to perform data manipulation and analysis. Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

Assignment:

1. Install numPy
   pip install numpy

```
pip install numpy
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
```

2. Import numPy library
   import numpy as np

```
import numpy as np
```

3. Create a 1-D array A.
   A = np.array([1,2,3,4,5,6,7,8,9])

```
A = np.array([1,2,3,4,5,6,7,8,9])
```

4. Print elements and type of A.
   print(A)
   print(type(A))

```
print(A)
print(type(A))

[1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
```

5. Create a 2-D array B. Print its elements and type.
   B = np.array([[1,2,3], [4,5,6], [7,8,9]])
   print(B)
   print(type(B))

```
B = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(B)
print(type(B))

[[1 2 3]
 [4 5 6]
 [7 8 9]]
<class 'numpy.ndarray'>
```

6. Create a 3-D array C. Print its elements and type.
C=np.array([[[1,2],[3,4],[5,6]],[[7,8],[9,10],[11,12]],[[13,14],[15,16],[17,18]]])
print(C)
print(type(C))

```
C=np.array([[[1,2],[3,4],[5,6]],[[7,8],[9,10],[11,12]],[[13,14],[15,16],[17,18]]])
print(C)
print(type(C))
```

```
[[[ 1  2]
  [ 3  4]
  [ 5  6]]

 [[ 7  8]
  [ 9 10]
  [11 12]]

 [[13 14]
  [15 16]
  [17 18]]]
<class 'numpy.ndarray'>
```

7. Print 3rd element of A
print(A[3])

```
print(A[3])
```

```
4
```

8. Print 2nd element of 3rd row of B
print(B[2,2])

```
print(B[2,2])
```

```
9
```

9. Print 1st element of 2nd array of the 2nd array from C.
print(C[2,2,1])

```
print(C[2,2,1])
```

```
18
```

10. Print last elements of C.
print(B[-1,-1,-1])

```
print(C[-1,-1,-1])
```

```
18
```

11. Slice elements from index 1 to 5 of A.
print(A[1:5])

```
print(A[1:5])
```

```
[2 3 4 5]
```

12. Slice elements from the beginning to 4th element of A
print(A[:5])

```
print(A[:5])
```

```
[1 2 3 4 5]
```

13. Slice elements from 3rd element to the end of A.
print(A[3:])

```
print(A[3:])
```

```
[4 5 6 7 8 9]
```

14. From the 2nd element of B, slice elements from 2 to 3.
print(B[2,2:4])

```
print(B[2,2:4])
```

```
[9]
```

15. Create an array fruit that contains names of the fruits.
fruit = np.array(['banana','apple','cherry','orange'])

```
fruit = np.array(['banana','apple','cherry','orange'])
print(fruit)
```

```
['banana' 'apple' 'cherry' 'orange']
```

16. Reshape array A into a new 2-D array A1.
A1 = A.reshape(3,3)

```
A1 = A.reshape(3,3)
print(A1)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

17. Flatten the array C.

C1 = C.reshape(-1)

```
C1 = C.reshape(-1)
print(C1)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18]
```

18. Iterate through all elements of A to print them.

for x in A:

print(x)

```
for x in A:
    print(x)
```

```
1
2
3
4
5
6
7
8
9
```

19. Iterate through all elements of B to print them.

for i in B:

for j in i:

print(j)

```
for i in B:
    for j in i:
        print(j)
```

```
1
2
3
4
5
6
7
8
9
```

20. Iterate through all elements of C to print them.

for i in C:
  for j in i:
    for k in j:
      print(k)

```
for i in C:
    for j in i:
        for k in j:
            print(k)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

21. Create an array A3 from A, where all elements of A3 are 5 times elements of A. Join A and A3 into A3

A3 = 5*A
print(A)
print(A3)
A3 = np.concatenate((A,A3))

```
A3 = 5*A
print(A)
print(A3)
A3 = np.concatenate((A,A3))
```

```
[1 2 3 4 5 6 7 8 9]
[ 5 10 15 20 25 30 35 40 45]
```

22. Create two 2-D arrays and perform element wise multiplication.

P = np.array([[1,2,3], [4,5,6]])
Q = np.array([[2, -7, 5], [-6, 2, 0]])
R = np.multiply(P,Q)
print(R)

```
P = np.array([[1,2,3], [4,5,6]])
Q = np.array([[2, -7, 5], [-6, 2, 0]])
R = np.multiply(P,Q)
print(R)
```

```
[[  2 -14  15]
 [-24  10   0]]
```

23. Create two 2-D arrays and perform multiplication.

P = np.array([[1,2,3], [4,5,6]])
Q = np.array([[2, -7], [-6, 2], [1,5]])
R = np.dot(P,Q)
print(R)

```
P = np.array([[1,2,3], [4,5,6]])
Q = np.array([[2, -7], [-6, 2], [1,5]])
R = np.dot(P,Q)
print(R)
```

```
[[ -7  12]
 [-16  12]]
```

24. Find square of all elements of B.

np.square(B)

```
np.square(B)
```

```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

25. Find addition of all elements of C.

np.sum(C)

```
np.sum(C)
```

```
171
```

26. Find 1/√(e^x-1) for all elements of C, where x is an elements of C.
print(np.reciprocal(np.sqrt((np.exp(C)-1))))

```
print(np.reciprocal(np.sqrt((np.exp(C)-1))))

[[[7.62873978e-01 3.95623107e-01]
  [2.28901063e-01 1.36591948e-01]
  [8.23629462e-02 4.98488882e-02]]

 [[3.02111611e-02 1.83187118e-02]
  [1.11096821e-02 6.73809996e-03]
  [4.08680557e-03 2.47875979e-03]]

 [[1.50344089e-03 9.11882345e-04]
  [5.53084455e-04 3.35462647e-04]
  [2.03468373e-04 1.23409805e-04]]]
```

27. Install and import pandas
pip install pandas
import pandas as pd

```
[37] pip install pandas

Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.25.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)

import pandas as pd
```

28. Read data from csv file into a dataframe df
df = pd.read_csv(r'/content/dataset.csv')

```
df = pd.read_csv(r'/content/dataset.csv')
```

29. Display top 10 rows of the data.
df.head(10)

```
df.head(10)
```

	Name	Age	Grade
0	Alice	18	A
1	Bob	17	A+
2	Charlie	16	B
3	David	18	C
4	Emma	17	F
5	Frank	16	A
6	Grace	18	B+
7	Henry	17	D
8	Isabella	16	A+
9	Kate	18	A+

30. Find the deatails of the data in datafrme.

df.info()

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    20 non-null     object
 1   Age     20 non-null     int64
 2   Grade   20 non-null     object
dtypes: int64(1), object(2)
memory usage: 608.0+ bytes
```

31. Display 'Grade' column of the data.

df['Grade']

```
df['Grade']
```

```
0      A
1     A+
2      B
3      C
4      F
5      A
6     B+
7      D
8     A+
9     A+
10     C
11     D
12     F
13    A+
14     A
15     B
16     C
17    B+
18     C
19     A
Name: Grade, dtype: object
```

32. Find the details of the 10th entry of dataframe.

df.iloc[9]

```
df.iloc[9]
```

```
Name     Kate
Age        18
Grade      A+
Name: 9, dtype: object
```

33. Display the details of the player whose age is greater than 16.

df[df['Age']>16]

```
df[df['Age']>16]
```

	Name	Age	Grade
0	Alice	18	A
1	Bob	17	A+
3	David	18	C
4	Emma	17	F
6	Grace	18	B+
7	Henry	17	D
9	Kate	18	A+
10	Liam	17	C
12	Noh	18	F
13	Olivia	17	A+
15	Quinn	18	B
16	Rachel	17	C
18	Taylor	18	C
19	Charliee	17	A

34. Display the details of the player whose age is greater than 16 and having Garde A or A+.

df[df['Age']>16 & df['Grade'].isin(['A', 'A+'])]

```
df[df['Age']>16 & df['Grade'].isin(['A', 'A+'])]
```

	Name	Age	Grade
0	Alice	18	A
1	Bob	17	A+
2	Charlie	16	B
3	David	18	C
4	Emma	17	F
5	Frank	16	A
6	Grace	18	B+
7	Henry	17	D
8	Isabella	16	A+
9	Kate	18	A+
10	Liam	17	C
11	Jack	16	D
12	Noh	18	F
13	Olivia	17	A+
14	Peter	16	A
15	Quinn	18	B
16	Rachel	17	C
17	Saumel	16	B+

35. Check if any empty cell is in the dataframe. Find the row with empty name entry and remove if any.

df.isnull().sum()

```
df.isnull().sum()
```

```
Name     0
Age      0
Grade    0
dtype: int64
```

Conclusion:

numPy library is very useful for defining, performing mathematical operations and manipulating arrays. Pandas library efficiently handles the data by fetching it into a dataframe.

Practical No. 2

Title: Univariate Linear regression using python

Aim: To implement Univariate Linear Regression using NumPy and pandas

Tools: anaconda, Python 3.7, Jupiter Notebook

Theory:

Q. Write equation for hypothesis for one sample.

$h_\theta(x) = \theta_0+\theta_1*x$
θ_0, θ_1 – parameters
x - feature

Q. Write equation for squared error function.

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Q. Write equation for Gradient Descent.

$$\begin{aligned}&\text{repeat until convergence } \{\\ &\quad \theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)\\ &\}\end{aligned}$$

Correct: Simultaneous update

$$\begin{aligned}&\text{temp0} := \theta_0 - \alpha\tfrac{\partial}{\partial\theta_0}J(\theta_0, \theta_1)\\ &\text{temp1} := \theta_1 - \alpha\tfrac{\partial}{\partial\theta_1}J(\theta_0, \theta_1)\\ &\theta_0 := \text{temp0}\\ &\theta_1 := \text{temp1}\end{aligned}$$

Q. Write matrix implementation of gradient descent.

$\theta = \theta - \alpha\,(X^T(h-y))/m$

Assignment:

1. The dataset contains the record of the profit earned by a food truck in cities with population mentioned in the first column. Construct a linear regression model that can predict the profit if the population of a city is known.

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

```
[2] df = pd.read_csv("/content/profit.txt")
```

```
[3] df.head()
```

	6.1101	17.592
0	5.5277	9.1302
1	8.5186	13.6620
2	7.0032	11.8540
3	5.8598	6.8233
4	8.3829	11.8860

```
[4] df = pd.read_csv("/content/profit.txt", header = None)
```

```
df.head(10)
```

	0	1
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233
5	8.3829	11.8860
6	7.4764	4.3483
7	8.5781	12.0000
8	6.4862	6.5987
9	5.0546	3.8166

```
[6] df.describe()
```

	0	1
count	97.000000	97.000000
mean	8.159800	5.839135
std	3.869884	5.510262
min	5.026900	-2.680700
25%	5.707700	1.986900
50%	6.589400	4.562300
75%	8.578100	7.046700
max	22.203000	24.147000

```
[7]  X = df.iloc[:,0]
```

```
[8]  X
```

```
0      6.1101
1      5.5277
2      8.5186
3      7.0032
4      5.8598
        ...
92     5.8707
93     5.3054
94     8.2934
95    13.3940
96     5.4369
Name: 0, Length: 97, dtype: float64
```

```
[9]  # Sample Size
     m = X.shape[0]
     print(m)
```

```
97
```

```
[10] type(X)
```

```
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool=False, fastpath:
bool=False) -> None

/usr/local/lib/python3.10/dist-packages/pandas/core/series.py
One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
```

```
[11]  X = X.values
```

```
[12]  type(X)
```

```
numpy.ndarray
```

```
[13]  X = X.reshape(m,1)
```

```
[14]  X.shape
```

```
(97, 1)
```

```
[15] X
```

```
[ 6.3534],
[ 5.4069],
[ 6.8825],
[11.708 ],
[ 5.7737],
[ 7.8247],
[ 7.0931],
[ 5.0702],
[ 5.8014],
[11.7   ],
[ 5.5416],
[ 7.5402],
[ 5.3077],
[ 7.4239],
```

```
[16] y = df.iloc[:,1]
```

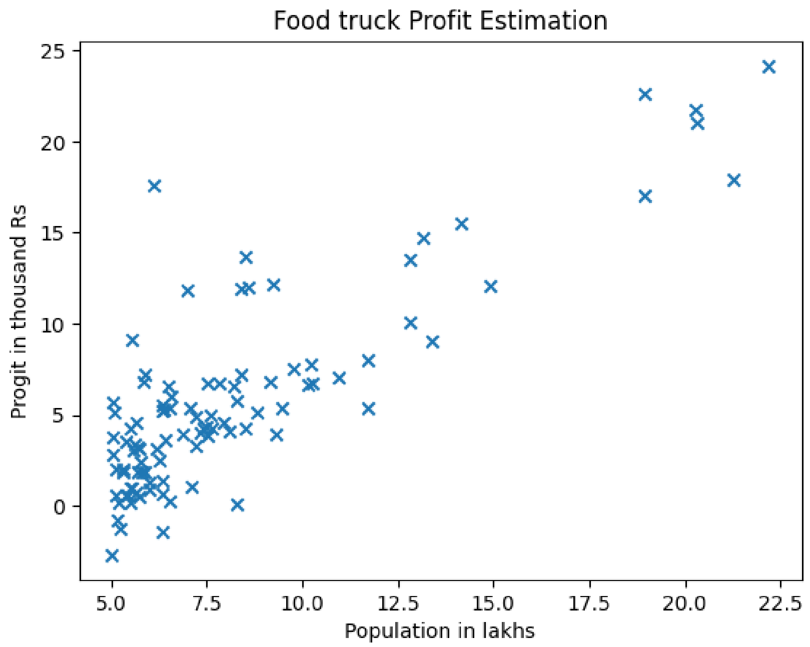
```
[17] y = (y.values).reshape(m,1)
```

```
[18] y.shape
```

```
(97, 1)
```

```
plt.scatter(X,y,marker='x')
plt.xlabel('Population in lakhs')
plt.ylabel('Progit in thousand Rs')
plt.title('Food truck Profit Estimation')
```

```
Text(0.5, 1.0, 'Food truck Profit Estimation')
```



[20]
```
X.shape
```

```
(97, 1)
```

[21]
```
y.shape
```

```
(97, 1)
```

[22]
```
col1 = np.ones((m,1))
col1
```

```
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
          [1.],
```

[23]
```
X = np.hstack((col1,X))
```

[24]
```
print(X)
```

```
 [ 1.       6.3534]
 [ 1.       5.4069]
 [ 1.       6.8825]
 [ 1.      11.708 ]
 [ 1.       5.7737]
 [ 1.       7.8247]
 [ 1.       7.0931]
 [ 1.       5.0702]
 [ 1.       5.8014]
 [ 1.      11.7    ]
 [ 1.       5.5416]
 [ 1.       7.5402]
 [ 1.       5.3077]
 [ 1.       7.4239]
 [ 1.       7.6031]
 [ 1.       6.3328]
```

```
[25] Theta = np.zeros((2,1))
     J = 0
     alpha = 0.1
     print(Theta)
```

```
[[0.]
 [0.]]
```

```
[26] # Hypothesis
     h = np.dot(X,Theta)
     print(h)
```

```
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
```

```
[27] J = np.sum(np.square(h-y))/(2*m)
     print(J)
```

```
32.072733877455676
```

```
[28] Theta[0]-(alpha/m)*np.sum(h-y)
```

```
array([0.58391351])
```

```
[29] Theta[1]-(alpha/m)*np.sum((h-y)*(X[:,1].reshape(m,1)))
```

```
array([6.53288497])
```

```
[30] for i in range(100):
       h = np.dot(X,Theta)
       J = np.sum(np.square(h-y))/(2*m)
       Theta[0] = Theta[0]-(alpha/m)*np.sum(h-y)
       Theta[1] = Theta[1]-(alpha/m)*np.sum((h-y)*(X[:,1].reshape(m,1)))
```

```
[31] Theta
```

```
array([[-5.88287103e+84],
       [-5.85588437e+85]])
```

```
[32] X_predict = 4.5
```

```
[33] y_predict = np.array([1,X_predict])*Theta
```

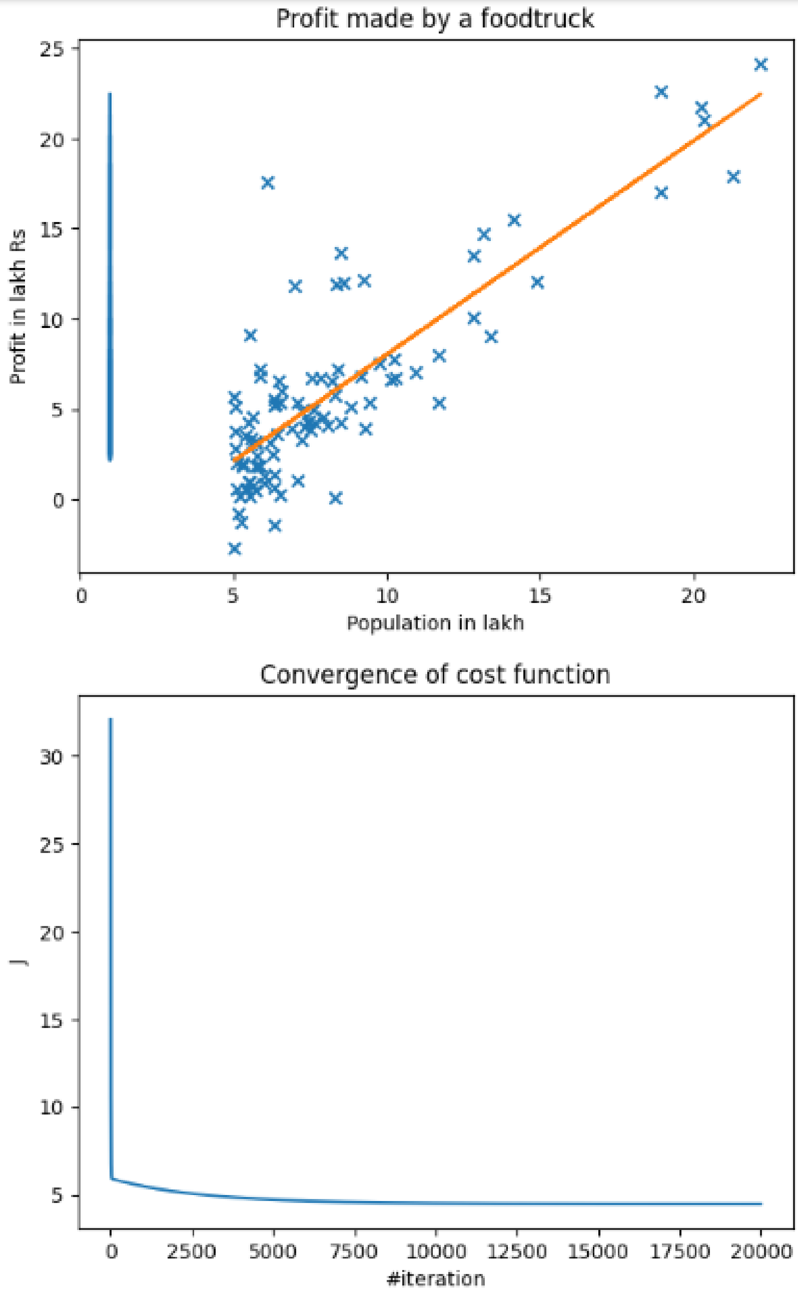
```
[34] def computeCost(X,y,Theta):
       m = y.shape[0]
       h = np.dot(X,Theta)
       J = np.sum(np.square(h-y))/(2*m)
       return[h,J]
```

```
[35] def gradientDescent(X,y,Theta,alpha):
       m = y.shape[0]
       h = computeCost(X,y,Theta)[0]
       J = computeCost(X,y,Theta)[1]
       Theta[0] = Theta[0]-(alpha/m)*np.sum(h-y)
       Theta[1] = Theta[1]-(alpha/m)*np.sum((h-y)*(X[:,1].reshape(m,1)))
       return [J,Theta]
```

```
[36] def trainLinearRegression (X,y,alpha,noIter,printIter):
       Theta = np.zeros((2,1))
       jHistory = []
       for i in range(noIter):
         J = gradientDescent(X,y,Theta,alpha)[0]
         jHistory.append(J)
         if (i%printIter==0):
           print('Iteration = ',i)
           print('Cost = ',J)

       plot1 = plt.figure(1)
       plt.scatter(X[:,1],y,marker='x')
       plt.plot(X,np.dot(X,Theta))
       plt.xlabel('Population in lakh')
       plt.ylabel('Profit in lakh Rs')
       plt.title('Profit made by a foodtruck')

       plot1 = plt.figure(2)
       plt.plot(list(range(noIter)),jHistory)
       plt.xlabel('#iteration')
       plt.ylabel('J')
       plt.title('Convergence of cost function')

       plt.show()

       return Theta
```

```
Theta = trainLinearRegression(X,y,0.001,20000,1000)
```

```
Iteration =  0
Cost =  32.072733877455676
Iteration =  1000
Cost =  5.480269332020323
Iteration =  2000
Cost =  5.176562563777922
Iteration =  3000
Cost =  4.964790400326137
Iteration =  4000
Cost =  4.817123460031757
Iteration =  5000
```



```
[40] def predProfit(population):
       X_pred = np.array([1,population])
       y_pred = np.dot(X_pred,Theta)
       print('The profit in the city in thousand Rs can be',y_pred)
```

```
[41] predProfit(6.1)
```

```
The profit in the city in thousand Rs can be [3.42355166]
```

Conclusion:

Linear Regression is a powerful and widely used technique for modelling relationships between variables, its successful application requires careful consideration of assumptions, data quality, model evaluation, and interpretation of results to derive meaningful insights and make informed decisions.

Practical No. 3

Title: Logistic Regression using Python

Aim: To implement Logistic Regression using NumPy and pandas

Tools: Anaconda, Python 3.7, Jupiter Notebook

Theory:

Q. Write equation for hypothesis for one sample.

$h_\theta (x)$ = sigmoid(θ_0+θ_1*x)
θ_0, θ_1 – parameters
x – feature
sigmoid(z) = $1/(1+e^{-z})$

Q. Write equation for error function.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

$$m = number\ of\ samples$$

Q. Write equation for Gradient Descent.

repeat until convergence {

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Q. Write matrix implementation of gradient descent.

$\theta = \theta - \alpha\ (X^T(h-y))/m$

Assignment:

1. The dataset contains the record of marks scored in exams (exam1 and exam2) by candidates appearing in two entrance tests and whether or not they got admitted to University previous year. Construct a Logistic regression model that can predict if the student based on their scores this year can get admitted to the University?

```python
[15] import pandas as pd
```

```python
[16] import matplotlib.pyplot as plt
```

```python
[17] import numpy as np
```

```python
[18] df = pd.read_csv('/content/studentData.txt')
```

```python
[19] df.head()
```

	34.62365962451697	78.0246928153624	0
0	30.286711	43.894998	0
1	35.847409	72.902198	0
2	60.182599	86.308552	1
3	79.032736	75.344376	1
4	45.083277	56.316372	0

Next steps: Generate code with df | View recommended plots

```python
[20] df = pd.read_csv('/content/studentData.txt')
```

```python
[21] df.head()
```

	34.62365962451697	78.0246928153624	0
0	30.286711	43.894998	0
1	35.847409	72.902198	0
2	60.182599	86.308552	1
3	79.032736	75.344376	1
4	45.083277	56.316372	0

Next steps: Generate code with df | View recommended plots

```
df.describe()
```

	34.62365962451697	78.0246928153624	0
count	99.000000	99.000000	99.000000
mean	65.957614	66.102779	0.606061
std	19.302009	18.638875	0.491108
min	30.058822	30.603263	0.000000
25%	51.297736	47.978125	0.000000
50%	67.319257	66.589353	1.000000
75%	80.234877	79.876423	1.000000
max	99.827858	98.869436	1.000000

```
[23] X = df.iloc[:,0:2]
```

```
[24] X = X.values
```

```
[25] type(X)
```

```
numpy.ndarray
```

```
[26] X.shape
```

```
(99, 2)
```

```
y = df.iloc[:,2]
y = y.values
print(y.shape)
y = y.reshape(99,1)
print(y.shape)
```

```
(99,)
(99, 1)
```

```python
pos=y==1
neg=y==0
print(pos)
print(neg)
```

```
[False]
[ True]
[ True]
[ True]
[False]
[False]
[False]
```

```python
[29] np.where(pos)
```

```
(array([ 2,  3,  5,  6,  7,  8, 11, 12, 14, 15, 17, 18, 20, 23, 24, 25, 29,
        30, 32, 36, 39, 41, 45, 46, 47, 48, 49, 50, 51, 55, 57, 58, 59, 65,
        67, 68, 70, 71, 72, 73, 74, 75, 76, 79, 80, 81, 82, 83, 84, 86, 87,
        89, 90, 92, 93, 94, 95, 96, 97, 98]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]))
```

```python
[30] np.where(pos)[0]
```

```
array([ 2,  3,  5,  6,  7,  8, 11, 12, 14, 15, 17, 18, 20, 23, 24, 25, 29,
       30, 32, 36, 39, 41, 45, 46, 47, 48, 49, 50, 51, 55, 57, 58, 59, 65,
       67, 68, 70, 71, 72, 73, 74, 75, 76, 79, 80, 81, 82, 83, 84, 86, 87,
       89, 90, 92, 93, 94, 95, 96, 97, 98])
```

```python
[31] X[np.where(pos)[0]]
```

```
array([[60.18259939, 86.3085521 ],
       [79.03273605, 75.34437644],
       [61.10666454, 96.51142588],
       [75.02474557, 46.55401354],
       [76.0987867 , 87.42056972],
       [84.43281996, 43.53339331],
       [82.30705337, 76.4819633 ],
       [69.36458876, 97.71869196],
       [53.97105215, 89.20735014],
       [69.07014406, 52.74046973],
       [70.66150955, 92.92713789],
       [76.97878373, 47.57596365],
       [89.67677575, 65.79936593],
       [77.92409145, 68.97235999],
       [62.27101367, 69.95445795],
       [80.19018075, 44.82162893],
       [61.37928945, 72.80788731],
       [85.40451939, 57.05198398],
```

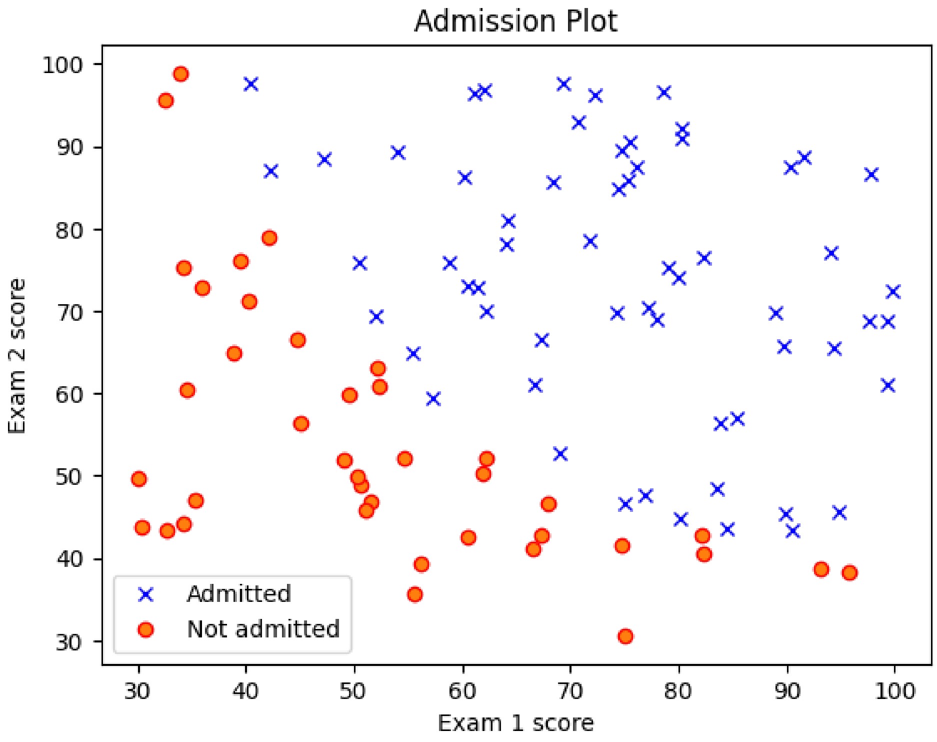
[32] plt

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```

[33]

```
plt.plot(X[np.where(pos)[0],0], X[np.where(pos)[0],1], 'x', mec='b')
plt.plot(X[np.where(neg)[0],0], X[np.where(neg)[0],1], 'o', mec='r')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.title('Admission Plot')
plt.legend(['Admitted', 'Not admitted'])
```

```
<matplotlib.legend.Legend at 0x7e54fc7c6350>
```



[34]

```
m = y.shape[0]
```

[35]

```
print(np.ones((m,1)))
```

```
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
```

```
[36] X = np.concatenate([np.ones((m,1)),X],axis=1)
```

```
[37] X
```

```
[ 1.        , 94.44336777, 65.56892161],
[ 1.        , 82.36875376, 40.61825516],
[ 1.        , 51.04775177, 45.82270146],
[ 1.        , 62.22267576, 52.06099195],
[ 1.        , 77.19303493, 70.4582    ],
[ 1.        , 97.77159928, 86.72782233],
[ 1.        , 62.0730638 , 96.76882412],
[ 1.        , 91.5649745 , 88.69629255],
[ 1.        , 79.94481794, 74.16311935],
[ 1.        , 99.27252693, 60.999031  ],
[ 1.        , 90.54671411, 43.39060181],
```

```
[38] X.shape[1]
```

```
3
```

```
[39] Theta = np.zeros((X.shape[1],1))
```

```
[40] Theta
```

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

```
[41] z = np.dot(X,Theta)
```

```
[42] def sigmoid(z):
       g = 1/(1+np.exp(-z))
       return g
```

```
[43] p = np.array([[0],[0]])
     sigmoid(p)
```

```
array([[0.5],
       [0.5]])
```

```
[44] h = sigmoid(z)
```

```
[45] J = np.sum(-y*np.log(h)-(1-y)*np.log(1-h))/m
```

```
[46] J

0.6931471805599453
```

```
[47] def costFuction(X,y,Theta):
       e = 0.00001
       z = np.dot(X,Theta)
       h = sigmoid(z)
       J = np.sum(-y*np.log(h+e)-(1-y)*np.log(1-h+e))/m
       return [h,J]
```

```
[49] def gradientDescent(X,y,Theta,alpha):
       noOfParam = Theta.shape[0]
       h = costFuction(X,y,Theta)[0]
       J = costFuction(X,y,Theta)[1]
       for i in range(noOfParam):
         Theta[i,0] = Theta[i,0]-alpha*np.sum((h-y)*X[:,i].reshape(m,1))/m
         return [h,J,Theta]
```

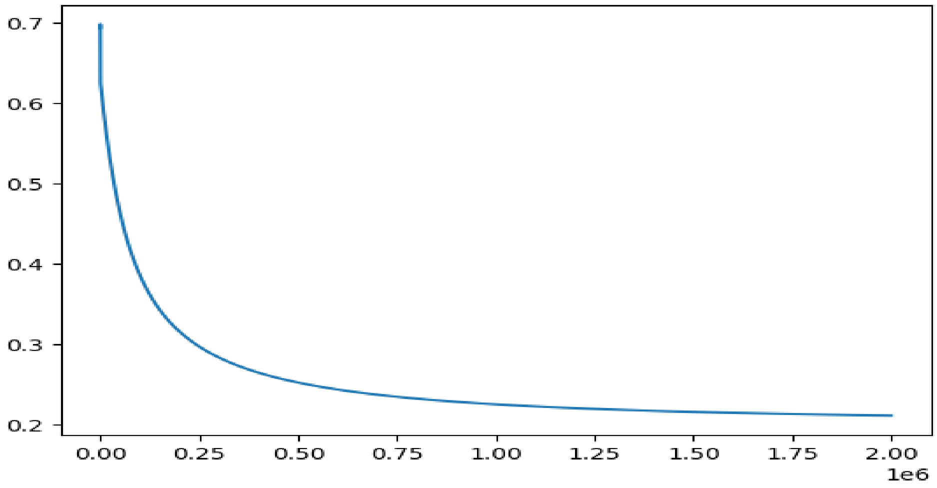
```
[50] jHistory = []
     print(jHistory)

     []
```

```
[48] def learningLogisticRegression(X,y,alpha,noOfIter):
       jHistory = []
       Theta = np.zeros((X.shape[1],1))
       for i in range(noOfIter):
         J = gradientDescent(X,y,Theta,alpha)[1]
         print(J)
         Theta  = gradientDescent(X,y,Theta,alpha)[2]
         jHistory.append(J)

       plt.plot(list(range(noOfIter)),jHistory)
```

```
[51] def learningLogisticRegression(X,y,alpha,noOfIter,printIter):
       jHistory = []
       e = 0.0000001
       Theta = np.zeros((X.shape[1],1))
       for i in range(noOfIter):
         h = 1/(1+np.exp(-np.dot(X,Theta)))
         J = np.sum(-y*np.log(h+e)-(1-y)*np.log(1-h+e))/m
         Theta = Theta-alpha*np.dot(X.transpose(),(h-y))/m
         jHistory.append(J)
         if (i%printIter==0):
           print('Iteration = ',i)
           print('Cost = ',J)
           print('Theta = ',Theta)

       plt.plot(list(range(noOfIter)),jHistory)
       return Theta
```

```
Theta = learningLogisticRegression(X,y,0.001,2000000,10000)
```

```
Iteration =  1930000
Cost =  0.21167231258707633
Theta =  [[-18.76967244]
 [  0.15449314]
 [  0.15071318]]
Iteration =  1940000
Cost =  0.211603045721821
Theta =  [[-18.79598929]
```




```
[53] Theta
```

```
array([[-18.95059175],
       [  0.1559417 ],
       [  0.15217058]])
```

```
def predict(X_pred):
  y_pred = 1/(1+np.exp(-np.dot(X_pred,Theta)))
  print(y_pred)
  if (y_pred<0.5):
    print('Sorry this candidate can not be admitted')
  else:
    print('Congratulations, there is possibility of you getting admitted to the college')
```

```
[55] X_predict = np.array([1,80,80])
     Theta
     predict(X_predict)
```

```
[0.99665984]
Congratulations, there is possibility of you getting admitted to the college
```

Conclusion:

The logistic regression model provides valuable insights into the relationship between exam scores and admission probability, offering a practical tool for predicting student admission to the University based on their performance in entrance exams. However, it's essential to consider the model's limitations and continue refining it for better accuracy and generalization to real-world scenarios.

Practical No. 4

Title: KNN using Python

Aim: To implement K-Nearest Neighbors using NumPy and pandas

Tools: Anaconda, Python 3.7, Jupiter Notebook

Theory:

Q. Write in shorts, how does KNN work?

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

The KNN Algorithm

- Load the data
- Initialize K to your chosen number of neighbors
- For each example in the data
  - Calculate the distance between the query example and the current example from the data.
  - Add the distance and the index of the example to an ordered collection
- Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances
- Pick the first K entries from the sorted collection
- Get the labels of the selected K entries
- If regression, return the mean of the K labels
- If classification, return the mode of the K labels

KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

Q. How to select the best value for k

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Following things should be kept in mind:

1. As we decrease the value of K to 1, our predictions become less stable.
2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
3. In cases where we are taking a majority vote among labels, we usually make K an odd number to have a tiebreaker.

Assignment:

1. The dataset contains the record of sepal length, sepal width, petal length, petal width and their class amongst, setosa, versicolor and virginica for Iris flower. Construct a KNN model that can predict the class of a new flower based on the above information.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
[2] df = pd.read_csv("/content/iris.csv")
```

```
[3] df.head()
```

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

Next steps: Generate code with df View recommended plots

```
df.describe()
```

	5.1	3.5	1.4	0.2
count	149.000000	149.000000	149.000000	149.000000
mean	5.848322	3.051007	3.774497	1.205369
std	0.828594	0.433499	1.759651	0.761292
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
[5] headername = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
```

```
[6] df = pd.read_csv("/content/iris.csv", names = headername)
```

```python
df.head()
```

	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

```python
df.describe()
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```python
[9] X = df.iloc[:,:-1].values
```

```python
X
```

```
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
```

```
[11] y = df.iloc[:,-1].values
```

```
[12] y
```

```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
```

```
[13] label,unique = pd.factorize(y)
```

```
[14] label
```

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

```
[15] unique
```

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

```
[16] X_train,X_test,y_train,y_test = train_test_split(X,label)
```

```
[17] def takeSecond(elem):
       return elem[1]
```

```
def KNNClassify(X_test,X_train=X_train,y_train=y_train,k=8):
  min_dist = []
  for i,point in enumerate(X_train):
    d0 = (point[0]-X_test[0])**2
    d1 = (point[1]-X_test[1])**2
    d2 = (point[2]-X_test[2])**2
    d3 = (point[3]-X_test[3])**2
    dist = np.sqrt(np.sum(d0+d1+d2+d3))
    min_dist.append((i,dist))

  min_dist.sort(key=takeSecond)
  neighbours = min_dist[:k]
  idx = []
  for tup in neighbours:
    idx.append(tup[0])

  output = y_train[idx]
  values,count = np.unique(output,return_counts=True)
  max_idx = np.argmax(count)
  return values[max_idx]
```

```
[19] predictions =list(map(KNNClassify,X_test))
```

```
[20] correctcount = 0
     for i in range(len(y_test)):
       if predictions[i] == y_test[i]:
         correctcount +=1


     accuracy = correctcount*100/len(y_test)
     print('Acuuracy = ', accuracy)

     Acuuracy =  97.36842105263158
```

```
[21] from sklearn import metrics as met
```

```
met.confusion_matrix(y_test, predictions)

array([[16,  0,  0],
       [ 0,  9,  1],
       [ 0,  0, 12]])
```

```
[23] met.accuracy_score(predictions,y_test)

     0.9736842105263158
```

Conclusion:

Implementing K-Nearest Neighbors (KNN) with the Iris dataset demonstrates its effectiveness in classification tasks. By adjusting parameters and scaling features appropriately, KNN can offer accurate predictions based on instance similarity. Its straightforward and instinctive approach makes it a valuable tool, especially for datasets like Iris with clear clusters. However, its performance may vary depending on the chosen K value and feature quality. In summary, KNN proves to be a versatile and dependable method for classification, striking a balance between simplicity and effectiveness.

Practical No. 5

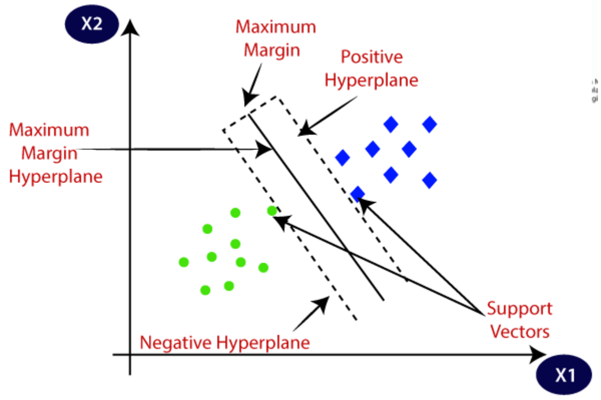
Title: SVM using python

Aim: To implement Support Vector Machine using Sci-kit learn

Tools: Anaconda, Python 3.7, Jupiter Notebook

Theory:

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification and regression challenges. Horithm iwever, it is mostly used in classification problems. The goal of the SVM algos to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.



In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

After adding the regularization parameter, the cost functions looks as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

SVM can be of two types:

<u>Linear SVM</u>: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

The SVM Algorithm

- Load the important libraries
- Import dataset and extract the X variables and Y separately.
- Divide the dataset into train and test
- Perform scaling on the dataset
- Initializing the SVM classifier model
- Fit the SVM classifier model
- Make prediction
- Evaluate model's performance

Assignment:

1. An automobile company wants to advertise their new launched car to potential customers. The potential customers can be identified based on the dataset that includes the age, salary, etc. Construct a SVM model that can decide whether an advertisement should be sent to a particular person based on the above information.

```
[27] import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
```

```
[28] df = pd.read_csv("/content/Social_Network_Ads.csv")
```

```
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Next steps: Generate code with df View recommended plots

```python
df.describe()
```

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

```python
[31] df['Age'].isna().sum()
```

```
0
```

```python
[32] df['EstimatedSalary'].isna().sum()
```

```
0
```

```python
[33] df['Purchased'].isna().sum()
```

```
0
```

```python
[34] X = df.iloc[:,2:4].values
```

```python
[35] X.shape
```

```
(400, 2)
```

```python
[36] y = df.iloc[:,4].values
```

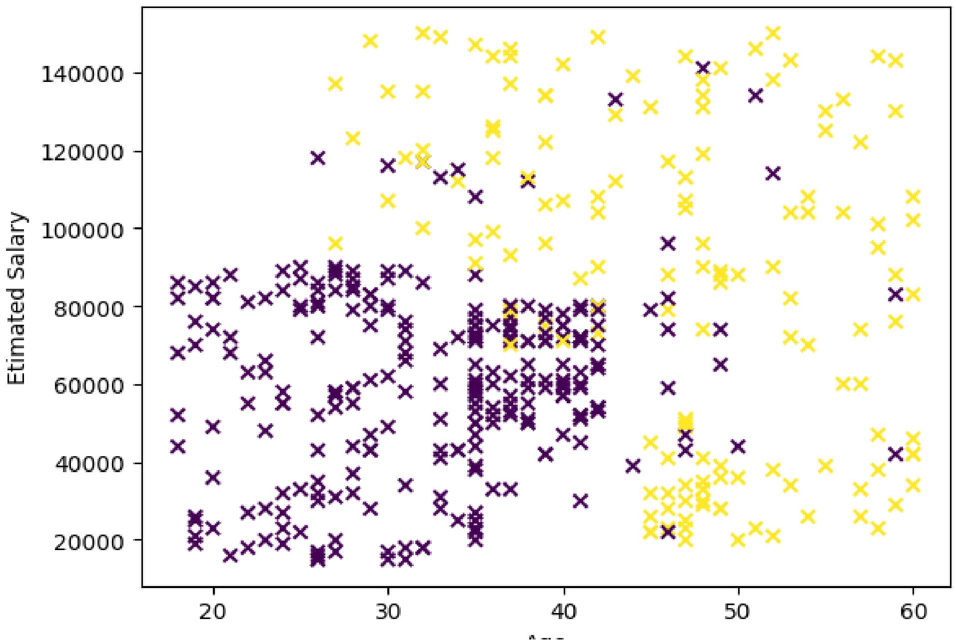
```python
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

[38]
```
y.shape
```

```
(400,)
```

```
import matplotlib.pyplot as plt
plt.scatter(X[:,0:1],X[:,1:2],c=y,marker='x',)
plt.xlabel('Age')
plt.ylabel('Etimated Salary')
plt.show()
```



[40]
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3)
```

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

```
X_train
```

```
       [ 2.09617497,  1.83842186],
       [-0.2317536 , -1.45535938],
       [ 1.12620473, -0.88643353],
       [ 1.70818687,  1.83842186],
       [ 0.35022854,  0.10170084],
       [-0.71673872,  0.61073976],
       [ 0.35022854,  0.10170084],
       [-0.03775956, -0.34745114],
       [-1.00772979, -0.31750768],
       [ 2.19317199, -0.67682927],
       [ 1.12620473, -0.97626392],
       [-0.52274467,  1.98813918],
       [-0.32875063, -1.30564205],
       [ 1.02920771,  2.16779998],
       [-1.29872087,  0.46102243],
       [ 0.83521366, -0.28756421],
       [ 1.22320175,  0.58079629],
       [-1.20172384, -1.06609432],
       [-0.2317536 , -0.31750768],
       [-0.03775956,  0.07175738],
```

```
[43] X_test
```

```
        [-0.30299055, -0.80225562],
        [-0.30299055, -0.35028062],
        [-1.13563633,  1.28812874],
        [ 0.15959044, -0.43502593],
        [-0.21047435, -0.57626812],
        [ 1.91739821,  1.99433967],
        [-1.41318493, -0.18079   ],
        [ 0.06707424, -0.03954781],
        [-0.21047435, -1.11298843],
        [-1.69073352,  0.38417875],
        [ 0.34462284,  0.21468812],
        [ 0.80720383, -0.60451656],
        [-1.41318493,  0.271185  ],
        [ 0.89972003,  0.4971725 ],
        [-0.48802295, -0.60451656],
        [-1.32066873, -1.50846655],
        [-1.04312013,  0.32768187],
        [-0.58053914,  0.38417875],
        [-0.85808774,  2.13558186],
```

```
[44] from sklearn.svm import SVC
```

```
[45] classifier = SVC(kernel = 'linear', random_state=0)
```

```
[46] classifier.fit(X_train,y_train)
```

```
SVC
SVC(kernel='linear', random_state=0)
```

```
[47] y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
print(confusion_matrix(y_test,y_pred))
print('Accuracy = ', accuracy_score(y_test,y_pred))
```

```
[[67  3]
 [23 27]]
Accuracy =  0.7833333333333333
```

```
[49] classifier = SVC(kernel = 'poly', random_state=0,degree = 2)
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print('Accuracy = ', accuracy_score(y_test,y_pred))
```

```
[[69  1]
 [27 23]]
Accuracy =  0.7666666666666667
```

```
[50] classifier = SVC(kernel = 'poly', random_state=0,degree = 3)
     classifier.fit(X_train,y_train)
     y_pred = classifier.predict(X_test)
     print(confusion_matrix(y_test,y_pred))
     print('Accuracy = ', accuracy_score(y_test,y_pred))
```

```
[[67  3]
 [23 27]]
Accuracy =  0.7833333333333333
```

```
[51] classifier = SVC(kernel = 'rbf', random_state=0)
     classifier.fit(X_train,y_train)
     y_pred = classifier.predict(X_test)
     print(confusion_matrix(y_test,y_pred))
     print('Accuracy = ', accuracy_score(y_test,y_pred))
```

```
[[66  4]
 [ 9 41]]
Accuracy =  0.8916666666666667
```

```
[52] classifier = SVC(kernel = 'rbf', random_state=0,C=3)
     classifier.fit(X_train,y_train)
     y_pred = classifier.predict(X_test)
     print(confusion_matrix(y_test,y_pred))
     print('Accuracy = ', accuracy_score(y_test,y_pred))
```

```
[[66  4]
 [ 9 41]]
Accuracy =  0.8916666666666667
```

Conclusion:

SVM is a sophisticated algorithm that can act as a linear and non-linear algorithm through kernels. However, when dealing with SVM, tuning the hyper-parameters and selecting the kernel is crucial, and the time taken during the training phase is high.

Practical No. 6

Title: K-Means clustering using python

Aim: To implement K-Means clustering using sci-kit learn library.

Tools: Anaconda, Python 3.7, Jupiter Notebook

Theory:

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. K-Means Clustering groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process. It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Algorithm:

- Select the number K to decide the number of clusters.
- Select random K points or centroids. (It can be other from the input dataset).
- Assign each data point to their closest centroid, which will form the predefined K clusters.
- Calculate the variance and place a new centroid of each cluster.
- Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- If any reassignment occurs, then go to step-4 else go to FINISH.
- The model is ready.

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. There are different ways to find the optimal number of clusters, elbow method is one of them.

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. WCSS stands for Within Cluster Sum of Squares, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS = \sum_{P_i \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{P_i \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{P_i \text{ in CLuster3}} distance(P_i\ C_3)^2$$

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Assignment:

1. The dataset contains the records of different countries that includes different numbers related to child mortality, exports, health, imports, income, etc. Let the United Nations want to design development plan for the different groups of countries. The groups to be formed based on the economic wellbeing of a country. Write a program in python to train a model using K-Means clustering to group the countries based on the dataset available.

```
[14] import pandas as pd
```

```
[15] df = pd.read_csv("/content/Country_data.csv")
```

```
df.head()
```

	country	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	Afghanistan	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	Albania	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	Algeria	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	Angola	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	Antigua and Barbuda	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200

```
df.describe()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
count	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000	167.000000
mean	38.270060	41.108976	6.815689	46.890215	17144.688623	7.781832	70.555689	2.947964	12964.155689
std	40.328931	27.412010	2.746837	24.209589	19278.067698	10.570704	8.893172	1.513848	18328.704809
min	2.600000	0.109000	1.810000	0.065900	609.000000	-4.210000	32.100000	1.150000	231.000000
25%	8.250000	23.800000	4.920000	30.200000	3355.000000	1.810000	65.300000	1.795000	1330.000000
50%	19.300000	35.000000	6.320000	43.300000	9960.000000	5.390000	73.100000	2.410000	4660.000000
75%	62.100000	51.350000	8.600000	58.750000	22800.000000	10.750000	76.800000	3.880000	14050.000000
max	208.000000	200.000000	17.900000	174.000000	125000.000000	104.000000	82.800000	7.490000	105000.000000

```python
[18] df['child_mort'].isnull().sum()
```

```
0
```

```python
[19] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167 entries, 0 to 166
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   country      167 non-null    object
 1   child_mort   167 non-null    float64
 2   exports      167 non-null    float64
 3   health       167 non-null    float64
 4   imports      167 non-null    float64
 5   income       167 non-null    int64
 6   inflation    167 non-null    float64
 7   life_expec   167 non-null    float64
 8   total_fer    167 non-null    float64
 9   gdpp         167 non-null    int64
dtypes: float64(7), int64(2), object(1)
memory usage: 13.2+ KB
```

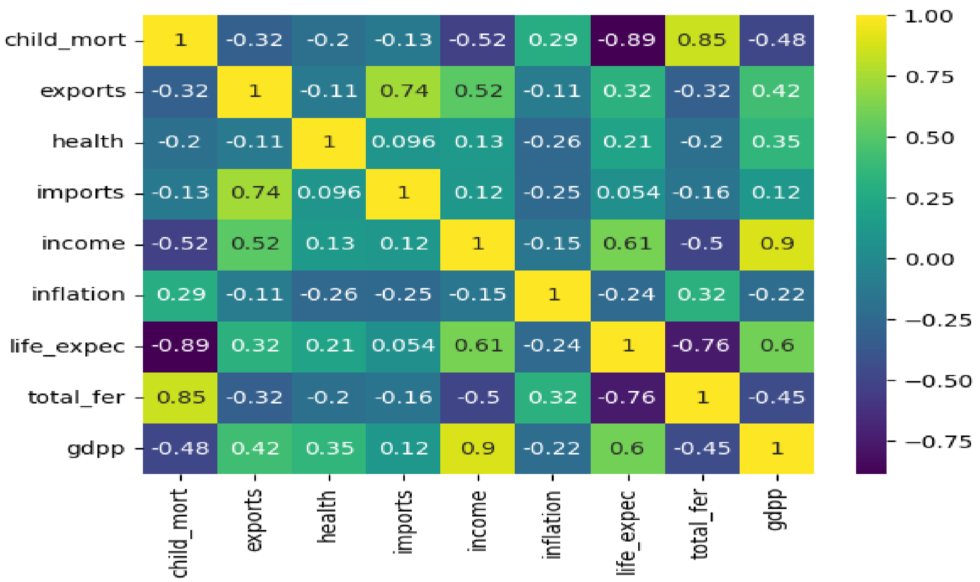
```python
[20] df = df.drop('country', axis=1)
     df.corr()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
child_mort	1.000000	-0.318093	-0.200402	-0.127211	-0.524315	0.288276	-0.886676	0.848478	-0.483032
exports	-0.318093	1.000000	-0.114408	0.737381	0.516784	-0.107294	0.316313	-0.320011	0.418725
health	-0.200402	-0.114408	1.000000	0.095717	0.129579	-0.255376	0.210692	-0.196674	0.345966
imports	-0.127211	0.737381	0.095717	1.000000	0.122406	-0.246994	0.054391	-0.159048	0.115498
income	-0.524315	0.516784	0.129579	0.122406	1.000000	-0.147756	0.611962	-0.501840	0.895571
inflation	0.288276	-0.107294	-0.255376	-0.246994	-0.147756	1.000000	-0.239705	0.316921	-0.221631
life_expec	-0.886676	0.316313	0.210692	0.054391	0.611962	-0.239705	1.000000	-0.760875	0.600089
total_fer	0.848478	-0.320011	-0.196674	-0.159048	-0.501840	0.316921	-0.760875	1.000000	-0.454910
gdpp	-0.483032	0.418725	0.345966	0.115498	0.895571	-0.221631	0.600089	-0.454910	1.000000

```python
[21] import seaborn as sns
```

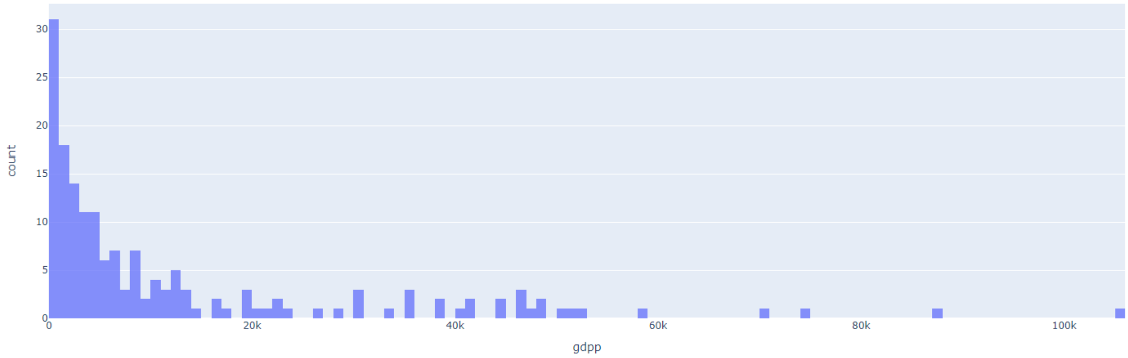
```python
sns.heatmap(df.corr(), annot=True, cmap='viridis')
```

```
<Axes: >
```



```
[23] import plotly.express as exp
```

```
exp.histogram(data_frame= df, x='gdpp',nbins=167, opacity=0.75, barmode='overlay')
```




```
[26] df['child_mort'].mean()
```

```
38.27005988023952
```

```
[27] df['child_mort'].max()
```

```
208.0
```

```
df.drop('gdpp', axis=1)
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer
0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82
1	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65
2	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89
3	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16
4	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13
...	...	...	...	...	...	...	...	...
162	29.2	46.6	5.25	52.7	2950	2.62	63.0	3.50
163	17.1	28.5	4.91	17.6	16500	45.90	75.4	2.47
164	23.3	72.0	6.84	80.2	4490	12.10	73.1	1.95
165	56.3	30.0	5.18	34.4	4480	23.60	67.5	4.67
166	83.1	37.0	5.89	30.9	3280	14.00	52.0	5.40

167 rows × 8 columns

```
[32] from sklearn.preprocessing import MinMaxScaler
```

```
[33] scaler = MinMaxScaler()
```

```
[35] scaled_data = scaler.fit_transform(df.drop('gdpp',axis=1))
```

```
[36] scaled_data
```

```
array([[0.42648491, 0.04948197, 0.35860783, ..., 0.12614361, 0.47534517,
        0.73659306],
       [0.06815969, 0.13953104, 0.29459291, ..., 0.08039922, 0.87179487,
        0.07886435],
       [0.12025316, 0.1915594 , 0.14667495, ..., 0.1876906 , 0.87573964,
        0.27444795],
       ...,
       [0.10077897, 0.35965101, 0.31261653, ..., 0.15072544, 0.8086785 ,
        0.12618297],
       [0.26144109, 0.1495365 , 0.20944686, ..., 0.25700028, 0.69822485,
        0.55520505],
       [0.39191821, 0.18455558, 0.25357365, ..., 0.16828389, 0.39250493,
        0.670347  ]])
```

```python
df.drop('gdpp', axis=1).columns
```

```
Index(['child_mort', 'exports', 'health', 'imports', 'income', 'inflation',
       'life_expec', 'total_fer'],
      dtype='object')
```

```python
[41] data = pd.DataFrame(scaled_data,columns=df.drop('gdpp', axis=1).columns)
```


```python
[41] data = pd.DataFrame(scaled_data,columns=df.drop('gdpp', axis=1).columns)
```


```python
[42] df.head()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	90.2	10.0	7.58	44.9	1610	9.44	56.2	5.82	553
1	16.6	28.0	6.55	48.6	9930	4.49	76.3	1.65	4090
2	27.3	38.4	4.17	31.4	12900	16.10	76.5	2.89	4460
3	119.0	62.3	2.85	42.9	5900	22.40	60.1	6.16	3530
4	10.3	45.5	6.03	58.9	19100	1.44	76.8	2.13	12200

Next steps: Generate code with df View recommended plots

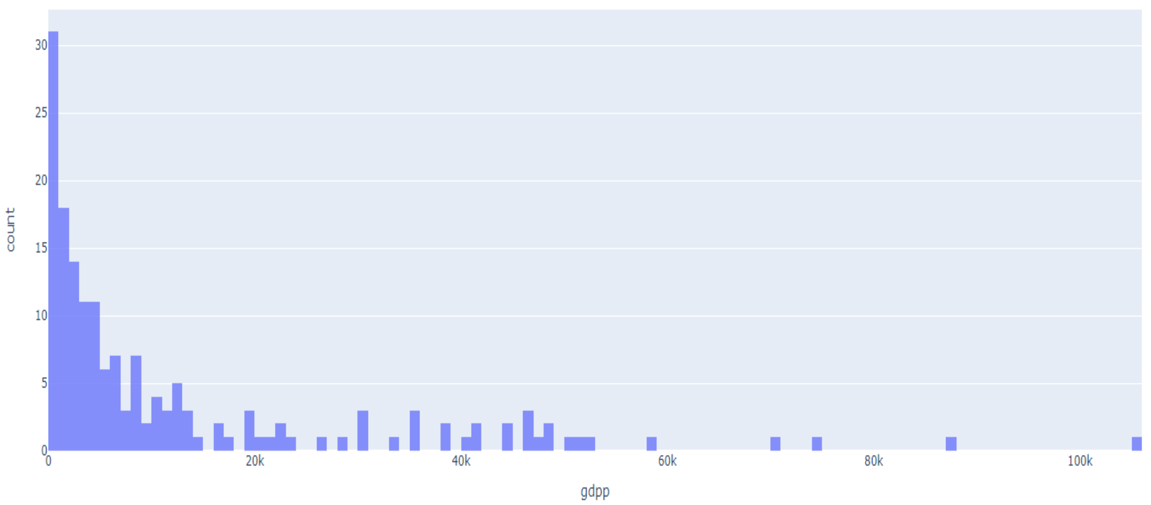
```python
[43] data['gdpp'] = df['gdpp']
```

```python
[44] data.head()
```

	child_mort	exports	health	imports	income	inflation	life_expec	total_fer	gdpp
0	0.426485	0.049482	0.358608	0.257765	0.008047	0.126144	0.475345	0.736593	553
1	0.068160	0.139531	0.294593	0.279037	0.074933	0.080399	0.871795	0.078864	4090
2	0.120253	0.191559	0.146675	0.180149	0.098809	0.187691	0.875740	0.274448	4460
3	0.566699	0.311125	0.064636	0.246266	0.042535	0.245911	0.552268	0.790221	3530
4	0.037488	0.227079	0.262275	0.338255	0.148652	0.052213	0.881657	0.154574	12200

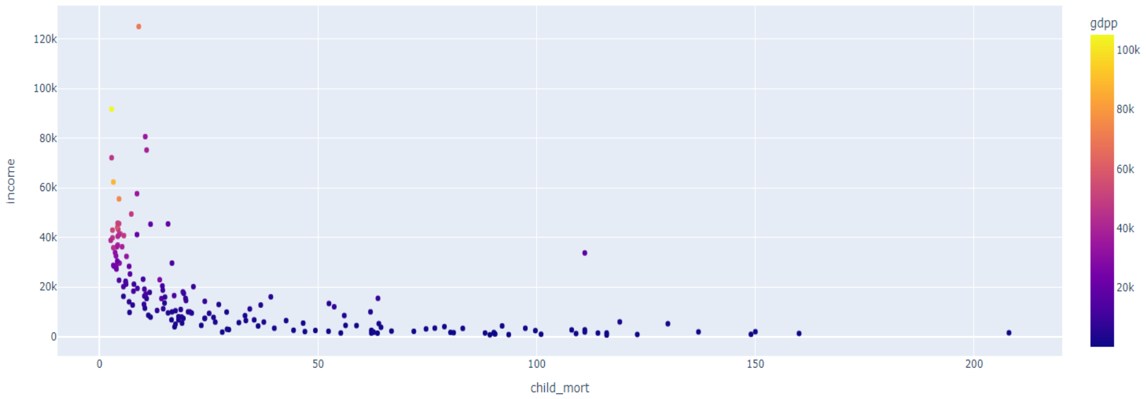
Next steps: Generate code with data View recommended plots

```python
exp.histogram(data_frame=df, x = 'gdpp',nbins=167,opacity=0.75,barmode='overlay')
```



```
[46] import matplotlib.pyplot as plt
```

```
exp.scatter(data_frame = df, x='child_mort', y='income', color='gdpp')
```




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

```
[50] k_means = KMeans(n_clusters=5)
```

```
[51] k_means.fit(data.drop('gdpp',axis=1))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

```
▾ KMeans
KMeans(n_clusters=5)
```

```
[52] k_means.labels_
```

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

```
[53] k_means.inertia_
```

```
13.68293200611038
```

```
k_means = KMeans(n_clusters=4)
k_means.fit(data.drop('gdpp', axis=1))
k_means.labels_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning

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

```
[55] k_means.inertia_
```

```
15.288570852888972
```

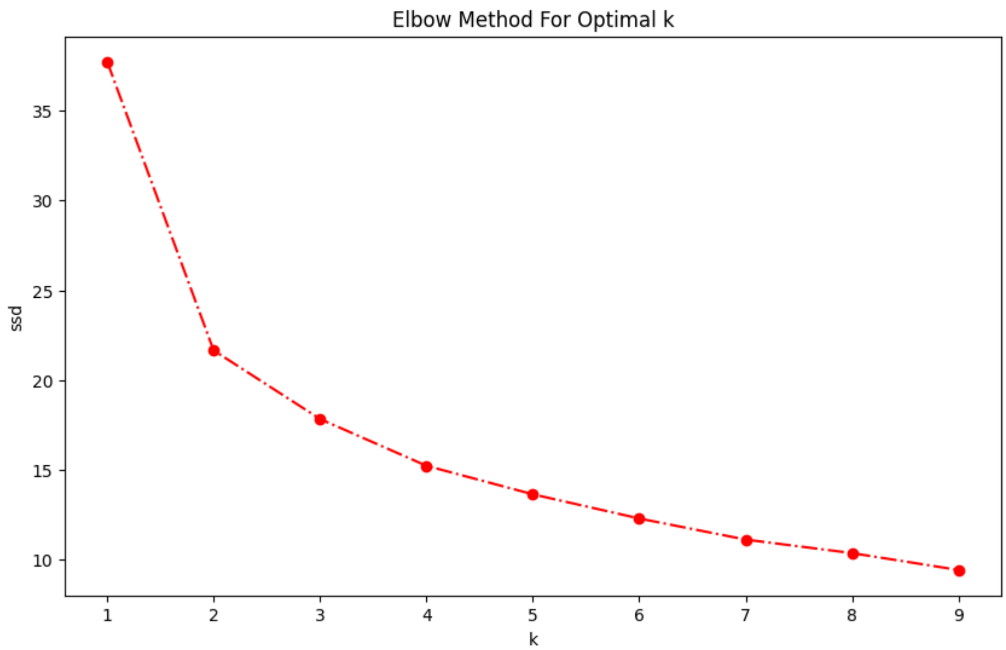
```
K = range(1,10)
ssd = []
for k in K:
 k_means = KMeans(n_clusters=k)
 k_means.fit(data.drop('gdpp', axis=1))
 ssd.append(k_means.inertia_)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
```

```
[57] ssd
```

```
[37.71822358882465,
 21.65310536770243,
 17.835238481489867,
 15.229773305595078,
 13.643426627008767,
 12.295820845330235,
 11.121687840370727,
 10.368572390657242,
 9.42957464212973]
```

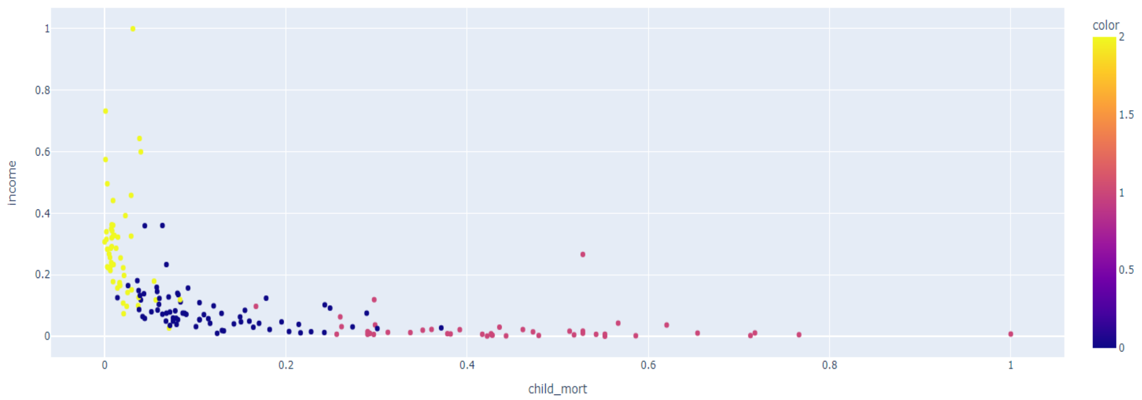
```
plt.figure(figsize=(10,6))
plt.plot(K,ssd, 'ro-.')
plt.xlabel('k')
plt.ylabel('ssd')
plt.title('Elbow Method For Optimal k')
plt.show()
```



```
[60] k_means = KMeans(n_clusters = 3)
     k_means.fit(data.drop('gdpp',axis=1))
     pred = k_means.labels_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
```

```
[62] exp.scatter(data_frame=data, x='child_mort',y='income',color=pred)
```



Conclusion:

The elbow methods shows that K=3 is the optimum value of K. The output image shows the three different clusters, of countries based on the data, with different colors.

Practical No. 7

Title: PCA using python

Aim: To implement Principal Component Analysis for data visualization using Sci-kit learn

Tools: Anaconda, Python 3.7, Jupiter Notebook

Theory:

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data. PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The transformed new features or the output of PCA are the Principal Components. The number of these PCs is either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

<u>Steps for PCA algorithm</u>:

- Getting the dataset
- Representing data into a structure
- Standardizing the data
- Calculating the new features or Principal Components
- Remove less or unimportant features from the new dataset.
- Display data with PCAs

Assignment:

1. A medical research group is working on Parkinson 's disease and want to develop an application for early detection of Parkinson's. The dataset has 755 features. Train a model to reduce the dimensionality and visualize the data.

```
[19] import pandas as pd
```

```
df = pd.read_csv("/content/pd_speech_features.csv")
```

```
df.head()
```

	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	meanPeriodPulses	stdDevPeriodPulses	locPctJitter	...	tqwt_kurtosisValue_
0	0	1	0.85247	0.71826	0.57227	240	239	0.008064	0.000087	0.00218	...	
1	0	1	0.76686	0.69481	0.53966	234	233	0.008258	0.000073	0.00195	...	
2	0	1	0.85083	0.67604	0.58982	232	231	0.008340	0.000060	0.00176	...	
3	1	0	0.41121	0.79672	0.59257	178	177	0.010858	0.000183	0.00419	...	
4	1	0	0.32790	0.79782	0.53028	236	235	0.008162	0.002669	0.00535	...	

5 rows × 755 columns

```
[22] df.shape
```

```
(756, 755)
```

```
[23] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Columns: 755 entries, id to class
dtypes: float64(749), int64(6)
memory usage: 4.4 MB
```

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

```
class
1    564
0    192
Name: count, dtype: int64
```

PCA for Visualization step 1 : Standardization

```
[25] X = df.iloc[:,:-1]
```

```
[26] X.shape
```

```
(756, 754)
```

```
[27] y = df.iloc[:,-1]
```

```
y.shape
```

```
(756,)
```

```
[29] from sklearn.preprocessing import StandardScaler
```

```
[30] scaler = StandardScaler()
```

```
[31] X_scaled = scaler.fit_transform(X)
```

```
[32] X_scaled.var()
```

```
1.0
```

```
[33] X_scaled.mean()
```

```
-1.553891536095293e-15
```

Step 2 : PCA with PC=2

```
[34] from sklearn.decomposition import PCA
```

```
[35] pca2 = PCA(n_components=2)
```

```
[36] principal_component = pca2.fit_transform(X_scaled)
```

```
principal_component.shape
```

```
(756, 2)
```

```
[38] type(principal_component)
```

```
numpy.ndarray
```

```
[39] principal_component
```

```
array([[-10.0473721 ,   1.47107525],
       [-10.63772497,   1.58374737],
       [-13.51618516,  -1.25354307],
       ...,
       [  8.27026448,   2.39128224],
       [  4.01176032,   5.41225399],
       [  3.99311363,   6.0724144 ]])
```

```
[40] principalDF = pd.DataFrame(data=principal_component, columns=['PC1','PC2'])
```

```
principalDF.head()
```

	PC1	PC2
0	-10.047372	1.471075
1	-10.637725	1.583747
2	-13.516185	-1.253543
3	-9.155084	8.833595
4	-6.764470	4.611463

```
[42] principalDF['class'] = df['class']
```

```
[43] principalDF.head()
```

	PC1	PC2	class
0	-10.047372	1.471075	1
1	-10.637725	1.583747	1
2	-13.516185	-1.253543	1
3	-9.155084	8.833595	1
4	-6.764470	4.611463	1

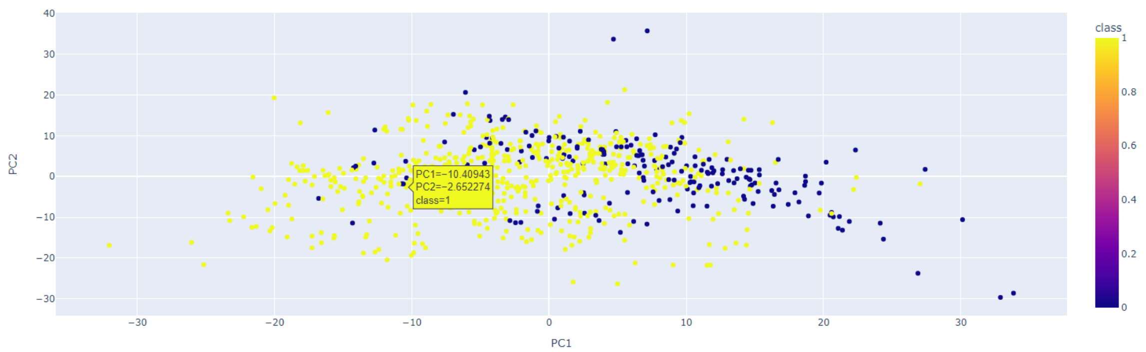
Next steps: Generate code with principalDF View recommended plots

Step 3: Visualizing the data

```
[44] import matplotlib.pyplot as plt
```

```
[45] import plotly.express as exp
```

```
[46] exp.scatter(data_frame=principalDF,x='PC1',y='PC2', color='class')
```



Conclusion:

The principal component analysis is a widely used unsupervised learning method to perform dimensionality reduction. The data with lower dimension can be visualized and interpreted.

Practical No. 8

Title: Random-Forest clustering using python

Aim: To implement random-forest using sci-kit learn library.

Tools: Anaconda, Python 3.7, Jupiter Notebook

Theory:

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems. Random forest is an ensemble method that works on the Bagging principle.

Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as row sampling. This step of row sampling with replacement is called bootstrap. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as aggregation.

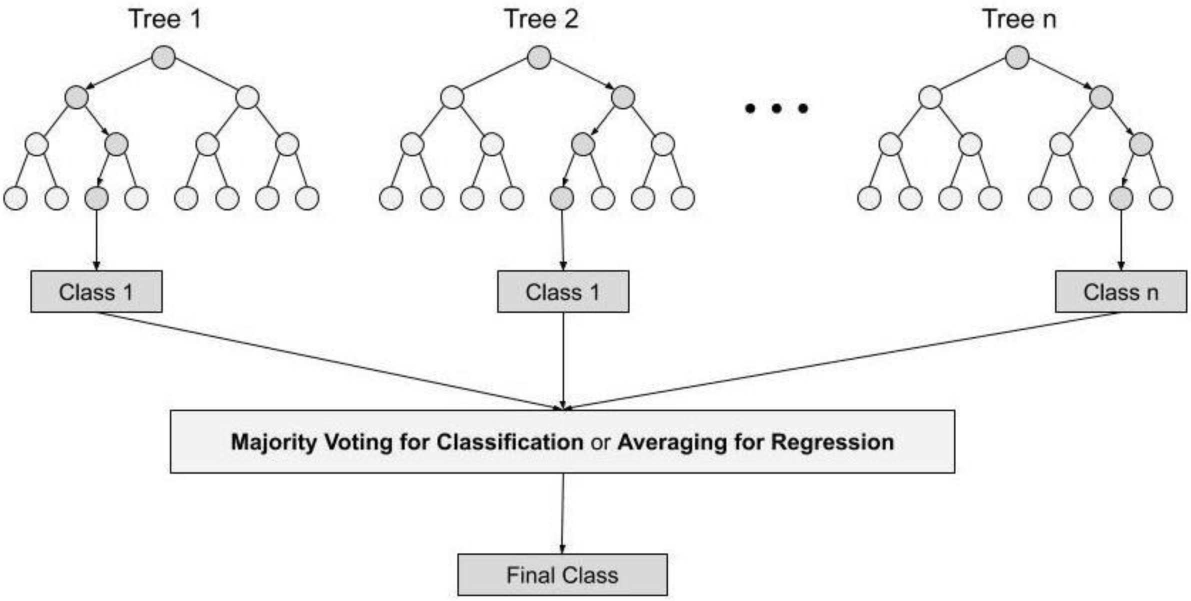
Important Features of Random Forest

1. Diversity- Not all attributes/variables/features are considered while making an individual tree, each tree is different.
2. Immune to the curse of dimensionality- Since each tree does not consider all the features, the feature space is reduced.
3. Parallelization-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.
4. Train-Test split- In a random forest we don’t have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.
5. Stability- Stability arises because the result is based on majority voting/ averaging.

Algorithm:

- In Random forest n number of random records are taken from the data set having k number of records.

- Individual decision trees are constructed for each sample.
- Each decision tree will generate an output.
- Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.



Assignment:

1. The dataset contains the historical data of temperature for a city taken from meteorological department. It contains the month of the year, day of the month, day of the week, maximum temperature of the previous two days, prediction of the temperature by a colleague, etc. Write a python program to predict the temperature of the city for the next day.

```
[1] import pandas as pd
    import numpy as np
```

```
df= pd.read_csv("/content/temps.csv")
```

```
[3] df.head()
```

	year	month	day	week	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend
0	2016	1	1	Fri	45	45	45.6	45	43	50	44	29
1	2016	1	2	Sat	44	45	45.7	44	41	50	44	61
2	2016	1	3	Sun	45	44	45.8	41	43	46	47	56
3	2016	1	4	Mon	44	41	45.9	40	44	48	46	53
4	2016	1	5	Tues	41	40	46.0	44	46	46	46	41

Next steps: Generate code with df View recommended plots

```python
[4] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 348 entries, 0 to 347
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   year            348 non-null    int64  
 1   month           348 non-null    int64  
 2   day             348 non-null    int64  
 3   week            348 non-null    object 
 4   temp_2          348 non-null    int64  
 5   temp_1          348 non-null    int64  
 6   average         348 non-null    float64
 7   actual          348 non-null    int64  
 8   forecast_noaa   348 non-null    int64  
 9   forecast_acc    348 non-null    int64  
 10  forecast_under  348 non-null    int64  
 11  friend          348 non-null    int64  
dtypes: float64(1), int64(10), object(1)
memory usage: 32.8+ KB
```

```python
[5] df.shape
```

```
(348, 12)
```

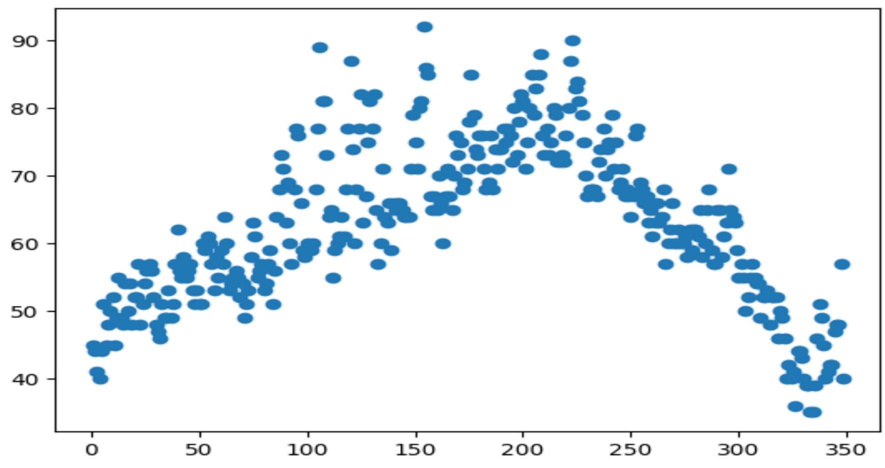
```python
[6] df.describe()
```

	year	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend
count	348.0	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000	348.000000
mean	2016.0	6.477011	15.514368	62.652299	62.701149	59.760632	62.543103	57.238506	62.373563	59.772989	60.034483
std	0.0	3.498380	8.772982	12.165398	12.120542	10.527306	11.794146	10.605746	10.549381	10.705256	15.626179
min	2016.0	1.000000	1.000000	35.000000	35.000000	45.100000	35.000000	41.000000	46.000000	44.000000	28.000000
25%	2016.0	3.000000	8.000000	54.000000	54.000000	49.975000	54.000000	48.000000	53.000000	50.000000	47.750000
50%	2016.0	6.000000	15.000000	62.500000	62.500000	58.200000	62.500000	56.000000	61.000000	58.000000	60.000000
75%	2016.0	10.000000	23.000000	71.000000	71.000000	69.025000	71.000000	66.000000	72.000000	69.000000	71.000000
max	2016.0	12.000000	31.000000	117.000000	117.000000	77.400000	92.000000	77.000000	82.000000	79.000000	95.000000

```python
[7] import matplotlib.pyplot as plt
```

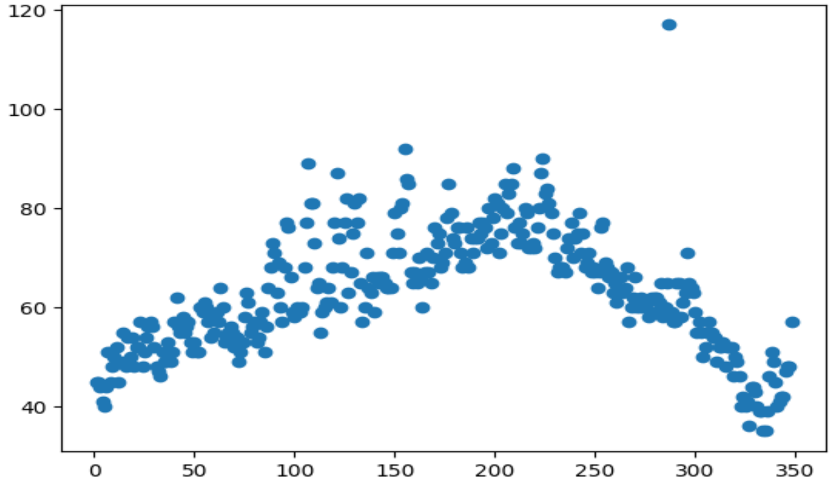
```python
plt.scatter(list(range(1,349)),df['actual'])
```

```
<matplotlib.collections.PathCollection at 0x79c352dd9780>
```



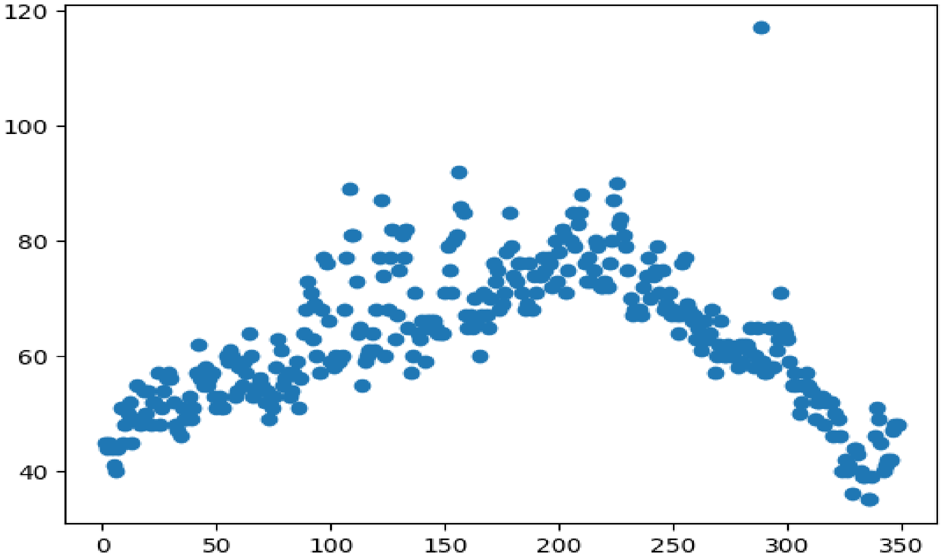
```
[9] plt.scatter(list(range(1,349)), df['temp_1'])
```

<matplotlib.collections.PathCollection at 0x79c350cf76d0>



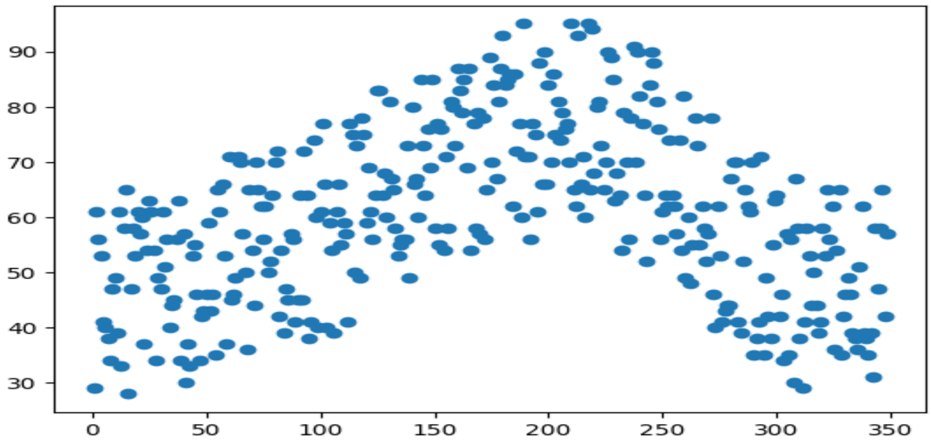

```
plt.scatter(list(range(1,349)),df['temp_2'])
```

<matplotlib.collections.PathCollection at 0x79c350b9d420>



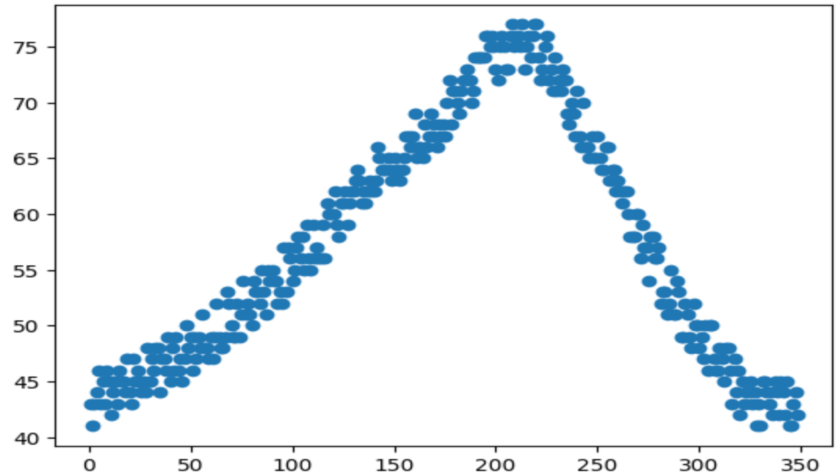

```
plt.scatter(list(range(1,349)),df['friend'])
```

<matplotlib.collections.PathCollection at 0x79c350c030a0>



```
[12] plt.scatter(list (range(1,349)), df['forecast_noaa'])
```

```
<matplotlib.collections.PathCollection at 0x79c350a92200>
```




```
[13] df = pd.get_dummies (df)
```

```
[14] df.head()
```

	year	month	day	temp_2	temp_1	average	actual	forecast_noaa	forecast_acc	forecast_under	friend	week_Fri	week_Mon	week_Sat	week_Sun	week_Thurs	week_Tues	week_Wed
0	2016	1	1	45	45	45.6	45	43	50	44	29	True	False	False	False	False	False	False
1	2016	1	2	44	45	45.7	44	41	50	44	61	False	False	True	False	False	False	False
2	2016	1	3	45	44	45.8	41	43	46	47	56	False	False	False	True	False	False	False
3	2016	1	4	44	41	45.9	40	44	48	46	53	False	True	False	False	False	False	False
4	2016	1	5	41	40	46.0	44	46	46	46	41	False	False	False	False	False	True	False

Next steps: Generate code with df   View recommended plots

```
[15] y = np.array(df['actual'])
```

```
[16] y.shape
```

```
(348,)
```

```
[17] df = df.drop('actual',axis= 1)
```

```
[18] df=df.drop('forecast_noaa',axis=1)
```

```
[19] df=df.drop('forecast_acc',axis=1)
```

```
[20] df=df.drop('forecast_under',axis=1)
```

```
df.head()
```

	year	month	day	temp_2	temp_1	average	friend	week_Fri	week_Mon	week_Sat	week_Sun	week_Thurs	week_Tues	week_Wed
0	2016	1	1	45	45	45.6	29	True	False	False	False	False	False	False
1	2016	1	2	44	45	45.7	61	False	False	True	False	False	False	False
2	2016	1	3	45	44	45.8	56	False	False	False	True	False	False	False
3	2016	1	4	44	41	45.9	53	False	True	False	False	False	False	False
4	2016	1	5	41	40	46.0	41	False	False	False	False	False	True	False

Next steps: Generate code with df   View recommended plots

```
list(df.columns)
```

```
['year',
 'month',
 'day',
 'temp_2',
 'temp_1',
 'average',
 'friend',
 'week_Fri',
 'week_Mon',
 'week_Sat',
 'week_Sun',
 'week_Thurs',
 'week_Tues',
 'week_Wed']
```

```
[23] feature_list = list(df.columns)
     feature_list
```

```
['year',
 'month',
 'day',
 'temp_2',
 'temp_1',
 'average',
 'friend',
 'week_Fri',
 'week_Mon',
 'week_Sat',
 'week_Sun',
 'week_Thurs',
 'week_Tues',
 'week_Wed']
```

```
[24] X = np.array(df)
```

```
[25] X.shape
```

```
(348, 14)
```

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

```
[27] Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.25, random_state=5)
```

```
[28] Xtrain.shape
```

```
(261, 14)
```

```
Xtest.shape
```

```
(87, 14)
```

[30] ytrain.shape

```
(261,)
```

[31] ytest.shape

```
(87,)
```

[32]
```python
from sklearn.ensemble import RandomForestRegressor
```

[33]
```python
rf = RandomForestRegressor (n_estimators =1000, random_state=5)
```

[34]
```python
rf.fit(Xtrain, ytrain)
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=1000, random_state=5)
```

```python
pred = rf.predict(Xtest)
```

[36]
```python
Xtest
```

```
array([[2016, 3, 20, ..., False, False, False],
       [2016, 10, 10, ..., False, False, False],
       [2016, 1, 22, ..., False, False, False],
       ...,
       [2016, 6, 1, ..., False, False, True],
       [2016, 10, 28, ..., False, False, False],
       [2016, 10, 17, ..., False, False, False]], dtype=object)
```

[37]
```python
pred
```

```
array([59.623, 62.315, 51.65 , 64.56 , 51.607, 60.182, 74.887, 75.718,
       55.504, 70.655, 43.338, 62.038, 57.479, 59.425, 63.764, 56.61 ,
       61.048, 52.145, 47.788, 57.432, 60.507, 70.825, 53.867, 55.995,
       57.865, 54.432, 80.204, 64.037, 56.775, 67.511, 72.67 , 77.568,
       43.023, 76.931, 67.315, 63.328, 61.973, 64.87 , 73.914, 52.022,
       66.578, 64.74 , 68.803, 56.845, 65.434, 54.95 , 59.507, 63.718,
       76.863, 61.49 , 60.201, 75.102, 60.257, 56.672, 78.504, 46.64 ,
       76.748, 71.4  , 65.934, 43.258, 52.133, 52.193, 49.924, 68.145,
       59.181, 60.904, 48.882, 67.446, 69.365, 75.903, 68.548, 70.963,
       69.887, 82.467, 63.193, 78.481, 59.788, 61.614, 55.459, 42.213,
       57.004, 44.794, 55.774, 75.549, 78.072, 61.485, 61.853])
```

```python
ytest
```

```
array([55, 60, 57, 66, 53, 57, 59, 82, 56, 77, 42, 58, 55, 51, 57, 55, 61,
       48, 45, 53, 65, 67, 51, 51, 61, 58, 83, 60, 53, 66, 75, 75, 42, 68,
       75, 59, 64, 66, 81, 52, 65, 61, 76, 51, 64, 54, 57, 66, 68, 64, 58,
       70, 60, 51, 72, 50, 71, 71, 67, 40, 48, 54, 49, 65, 57, 62, 45, 72,
       76, 71, 65, 78, 75, 67, 68, 73, 62, 60, 57, 41, 53, 51, 52, 75, 75,
       65, 60])
```

```
[39] errors = abs(pred - ytest)
```

```
[40] print('Mean absolute error =', round(np.mean(errors), 2), 'degrees')
```

```
Mean absolute error = 3.8 degrees
```

```
[41] import sklearn.metrics as met
```

```
[42] met.median_absolute_error(pred,ytest)
```

```
3.3599999999999994
```

```
[43] mape = 100*errors/ytest
```

```
[44] accuracy = 100-np.mean (mape)
     print(accuracy)
```

```
93.81245775201744
```

Conclusion:

The random forest algorithm can be used with good accuracy for the prediction of the temperature using the given data. The performance of the model can be improved further hyper parameter tuning.