Laboratory Manual
For
Machine Learning
(3170724)

Name: **Bhumit M. Hirpara**

Branch: **Computer Engineering**

Enrollment No.: **181390107017**

Academic Year: **2021 - 2022**

BHAGWAN ARIHANT INSTITUE OF TECHNOLOGY, SURAT

BHAGWAN ARIHANT INSTITUE OF TECHNOLOGY, SURAT

CERTIFICATE

This is to certify that **MR. BHUMIT M. HIRPARA**, of **7th** semester of **COMPUTER** branch, Enrollment No **181390107017** has satisfactorily submitted his term work in **MACHINE LEARNING (3170724)** for the term ending in **2021 – 2022**.

Date: / /

Sign of Examiner **Sign of HOD**

INDEX

PRACTICAL – 1

AIM: Implement a simple calculator in Python.

Code:

```
num1 = float(input("First Number: "))
operator = input("Operator (+, -, *, /): ")
num2 = float(input("Second Number: "))

out = None

if operator == "+":
    out = num1 + num2
elif operator == "-":
    out = num1 - num2
elif operator == "*":
    out = num1 * num2
elif operator == "/":
    out = num1 / num2

print("Answer: " + str(out))
```

Output:

```
First Number: 21
Operator (+, -, *, /): +
Second Number: 30
Answer: 51.0
```

```
First Number: 21
Operator (+, -, *, /): -
Second Number: 30
Answer: -9.0
```

```
First Number: 21
Operator (+, -, *, /): *
Second Number: 30
Answer: 630.0
```

```
First Number: 30
Operator (+, -, *, /): /
Second Number: 21
Answer: 1.4285714285714286
```

PRACTICAL – 2

AIM: Implement List, Set, Tuple & Dictionary with their different methods.

1. **Operation on List**

Code:

```
print('----------- List Operation -----------', end='\n\n' )
# Declaring lists
list1 = ['Bhumit', 'Tulsi', 'Vijay', 'Unnati']
list2 = [21, 30, 27.0, 22.0]
print(list1)
print(list2)
# Length of a list
print('Length of list1: ', len(list1))
# Reversing a list
print(list1[ : :-1])
# Type checking
print(type(list1))
# Adding an element at the last
list1.append('Charmil')
print(list1)
# Adding an element at particular position
list2.insert(4, '14')
print(list2)
# Finding index of any element
print(list1.index('Tulsi'))
# Removing an element by providing an element
list1.remove('Charmil')
print(list1)
# Removing an element from last
list2.pop()
print(list2)
# Sorting list2
list2.sort()
print(list2)
# Adding elements of list2 into list1
list1.extend(list2)
print(list1)
# Clearing elements of list1
list1.clear()
print(list1)
```

Output:

```
----------- List Operation -----------

['Bhumit', 'Tulsi', 'Vijay', 'Unnati']
[21, 30, 27.0, 22.0]
Length of list1:  4
['Unnati', 'Vijay', 'Tulsi', 'Bhumit']
<class 'list'>
['Bhumit', 'Tulsi', 'Vijay', 'Unnati', 'Charmil']
[21, 30, 27.0, 22.0, '14']
1
['Bhumit', 'Tulsi', 'Vijay', 'Unnati']
[21, 30, 27.0, 22.0]
[21, 22.0, 27.0, 30]
['Bhumit', 'Tulsi', 'Vijay', 'Unnati', 21, 22.0, 27.0, 30]
[]
```

2. **Operation on Tuple**

Code:

```
print('------------- Tuple Operation -------------', end='\n\n')
tuple1 = (21, 30, 27, 22, 21)
tuple2 = ('Bhumit', 'Tulsi', 'Vijay', "Unnati")
print(tuple1, tuple2, sep='\n')
# Type checking
print(type(tuple1))
# Length of tuple1
print('Length of tuple1: ', len(tuple1))
# Adding two tuples
tuple = tuple1 + tuple2
print(tuple)
```

Output:

```
------------ Tuple Operation ------------

(21, 30, 27, 22, 21)
('Bhumit', 'Tulsi', 'Vijay', 'Unnati')
<class 'tuple'>
Length of tuple1:  5
(21, 30, 27, 22, 21, 'Bhumit', 'Tulsi', 'Vijay', 'Unnati')
```

3. **Operation on Set**

Code:

```
print('------------- Set Operation -------------', end='\n\n')
# Declaring sets
set1 = {21, 30, 27, 22}
set2 = {17, 21, 30, 3, 15}
print(set1)
print(set2)
# Type checking
print(type(set1))
# Length of set1
print('Length of set1: ', len(set1))
# Adding an element into set1
set1.add(10)
print(set1)
# Removing an element by providing an element
set1.discard(10)
print(set1)
set1.remove(22)
print(set1)
# Removing an element from last
set1.pop()
print(set1)
# Union Operation
print(set1.union(set2))
# Intersection Operation
print(set1.intersection(set2))
# Difference Operation
print(set1.difference(set2))
# Clearing elements of set1
set1.clear()
print(set1)
```

Output:

```
------------- Set Operation -------------

{27, 21, 30, 22}
{17, 3, 21, 30, 15}
<class 'set'>
Length of set1:  4
{10, 21, 22, 27, 30}
{21, 22, 27, 30}
{21, 27, 30}
{27, 30}
{17, 3, 21, 27, 30, 15}
{30}
{27}
set()
```

4. **Operation on Dictionary**

Code:

```
print('----------------- Dictionary Operation -----------------', end='\n\n')
dict = {21: 'Bhumit', 30: 'Tulsi', 27: 'Vijay', 22: 'Unnati'}
dict1 = {'Bholu': 17, 'Tulu': 3, 15: 'Vijayo'}
print(dict, dict1, sep='\n')
# Changing value of a dict1
dict1[15] = 'Dudhat'
print(dict1)
# Getting value of specified key
print(dict.get(21))
# Getting a list containing a tuple for each key value pair
print(dict.items())
# Getting a list containing the dictionary's keys
print(dict.keys())
# Getting a list of all the values in the dictionary
print(dict.values())
# Removing the element with the specified key
dict1.pop(15)
print(dict1)
# Removing the last inserted key-value pair
dict.popitem()
print(dict)
# Updating a dictionary or adding two dictionaries
dict.update(dict1)
print(dict)
```

Output:

```
----------------- Dictionary Operation -----------------

{21: 'Bhumit', 30: 'Tulsi', 27: 'Vijay', 22: 'Unnati'}
{'Bholu': 17, 'Tulu': 3, 15: 'Vijayo'}
{'Bholu': 17, 'Tulu': 3, 15: 'Dudhat'}
Bhumit
dict_items([(21, 'Bhumit'), (30, 'Tulsi'), (27, 'Vijay'), (22, 'Unnati')])
dict_keys([21, 30, 27, 22])
dict_values(['Bhumit', 'Tulsi', 'Vijay', 'Unnati'])
{'Bholu': 17, 'Tulu': 3}
{21: 'Bhumit', 30: 'Tulsi', 27: 'Vijay'}
{21: 'Bhumit', 30: 'Tulsi', 27: 'Vijay', 'Bholu': 17, 'Tulu': 3}
```

PRACTICAL – 3

AIM: List out & explain various libraries of Machine Learning with details.

To provide a structure to our discussion, we will discuss Machine Learning Libraries as follows:

Purpose	Libraries
Scientific Computation	NumPy
Tabular Data	Pandas
Data Modelling & Pre-processing	Scikit Learn
Time-Series Analysis	Stats models
Text processing	Regular Expression, NLTK
Deep Learning	TensorFlow, Pytorch

A. NumPy

- NumPy or numerical Python is arguably one of the most important Python packages for ML. Scientific computations use a ton of **matrix operations** and these operations can be pretty computationally heavy. Implementing them naively can lead to inefficient memory usage.

- NumPy arrays are a special class of arrays that do these operations within milliseconds. These arrays are implemented in C programming language. In tasks like NLP where you have a large set of vocabulary and hundreds of thousands of sentences, a single matrix can have millions of numbers. As a beginner, you have to master using this library.

B. Pandas

- In simple terms, Pandas is the Python equivalent of **Microsoft Excel**. Whenever you have tabular data, you should consider using Pandas to handle it. The good thing about Pandas is that doing operations is just a matter of a couple of lines of code. If you want to do something complex, and you find yourself thinking about a lot of code, there is a high probability that there exists a Pandas command to fulfil your wish in a line or two.

- Right from *data manipulation*, to *transform* it, to *visualize* it, Pandas does it all. If you aspire to be a Data Scientist or are looking to ace ML competitions, Pandas can reduce your workload and help you focus on the problem-solving part and not writing boilerplate code.

C. Scikit Learn

- Scikit Learn is perhaps the most popular library for ML. It provides almost every popular model – *Linear Regression, Lasso-Ridge, Logistics Regression, Decision Tree, SVMs* and a lot more. Not only that, but it provides an extensive suite of tools to pre-process data, vectorizing text using BOW, TF-IDF or hashing vectorization and many more.

- It has huge support from the community. The only drawback is that it does not support distributed computing for large scale production environment applications well. If you wish to build career as a Data Scientist or Machine Learning Engineer, this library is a must!

D. Stats models

- Stats models is another library to implement **statistical learning algorithms**. However, it is more popular for its module that helps implement time series models. You can easily decompose a time-series into its trend component, seasonal component, and a residual component.

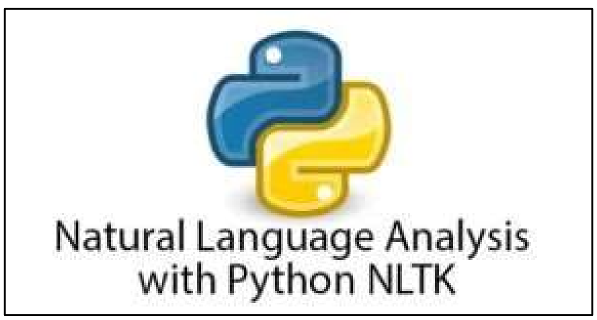
- You can also implement popular ETS methods like exponential smoothing, Holt-Winters method and models like *ARIMA* and *Seasonal ARIMA* or *SARIMA*. The only drawback is that this library does not have a lot of popularity and thorough documentation as Scikit.

E. Regex or Regular Expressions

- Regular expressions or regex is perhaps the simplest yet the most useful library for **text processing**. It helps find text according to defined strings patterns in a text. For example, if you wish to replace all the can'ts and don'ts in your text with cannot or do not, regex can do it in a jiffy.
- If you wish to find phone numbers in your text, you just have to define a pattern and regular expressions with return all the phone numbers in your text. It not only can find patterns but can also replace it with a string of your choice. Making correct matching patterns can be a little confusing in the beginning, but once you get a hang of it, it's fun!

F. NLTK

- Natural Language Toolkit is an extensive library for NLP. It is a go-to package for all your text processing needs – from word tokenization to lemmatization, stemming, dependency parsing, chunking and many more.



- Text processing is extremely important for any NLP tasks like Language Modeling, Neural machine Translation or Named Entity Recognition. It also provides a synonym bank called wordnet.

G. TensorFlow

- TensorFlow is by far currently the most popular library with extensive documentation and developer community support. It was created by **Google**. For product-based companies, TensorFlow is no brainer because of the ecosystem it provides for model prototyping to production. Tensor board a web-based visualization tool helps developers to visualize model performance, model parameters and gradients.

- A major criticism about TensorFlow in the community is its implementation of graphs. A graph is a set of operations you define. For example, c = a+b, d = c*c is a graph the does two operations on 4 variables. In python, you can perform the first step, get the value of c and then use it to calculate d. In TensorFlow, you have to compile the graph first. This means TensorFlow will first arrange all the operations and then execute them all at once.
- Unlike Python which is define by run, TensorFlow is define and run. This makes debugging cumbersome. In the recent TensorFlow summit, they have made changes to enable the define by run mode using eager execution. However, when it comes to the production environment, TensorFlow provides frameworks like TensorFlow Lite (for mobile devices) and TensorFlow Serving for deploying models.

H. Pytorch

- In a single line, Pytorch is everything TensorFlow is not. It was developed by **Facebook** as a Pythonic version of the original library Torch, which is a deep learning framework written for Lua programming language.
- Unlike TensorFlow, it was designed to be as Pythonic as possible. One major way in which it blows TensorFlow out of water is its execution of Dynamic Graphs. You can define your model components on the go. This is a blessing if you want to do research where you need this kind of flexibility with low-level APIs.
- If you are a beginner and wish to get your hands dirty, Pytorch is your thing. Since it is relatively new, it isn't as popular as TensorFlow. But the community is changing its preferences rapidly.

PRACTICAL – 4

AIM: Implement one dataset in Python & also read csv file using Python.

1. **Creating Dataset using DataFrame() Method:**

Code:

```
import pandas as pd
df = pd.DataFrame({
    'Er. No.' : [17, 3, 15, 10],
    'Name' : ['Bhumit', 'Tulsi', 'Vijay', 'Unnati'],
    'CGPA' : [9, 8.3, 8.1, 8.5]
})
df
```

Output:

	Er. No.	Name	CGPA
0	17	Bhumit	9.0
1	3	Tulsi	8.3
2	15	Vijay	8.1
3	10	Unnati	8.5

2. **Reading CSV file using read_csv:**

Code:

```
import pandas as pd
df = pd.read_csv(r'C:/Users/hbhum/student_marks.csv')
df
```

Output:

	Unnamed: 0	Gender	DOB	Maths	Physics	Chemistry	English	Biology	Economics	History	Civics
0	John	M	05/04/1988	55	45	56.0	87	21	52	89	65
1	Suresh	M	4/5/1987	75	55	NaN	64	90	61	58	2
2	Ramesh	M	25/5/1989	25	54	89.0	76	95	87	56	74
3	Jessica	F	12/8/1990	78	55	86.0	63	54	89	75	45
4	Jennifer	F	2/9/1989	58	96	78.0	46	96	77	83	53

PRACTICAL – 5

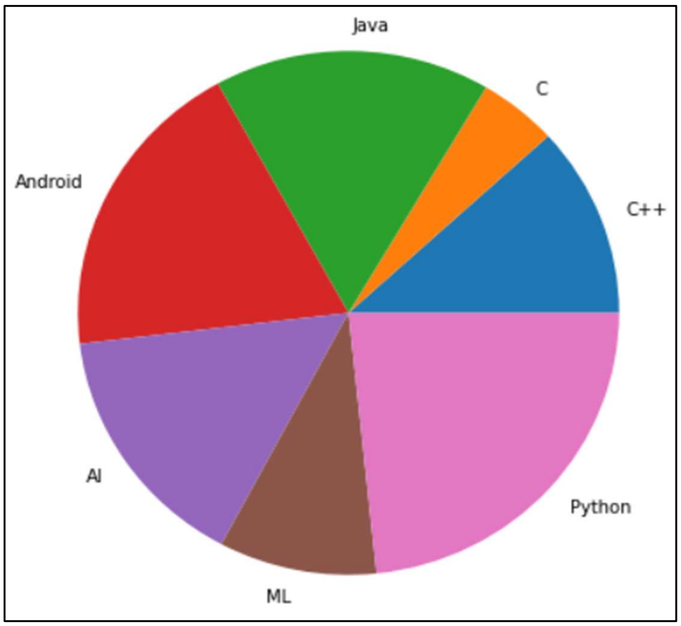
AIM: Implement Data Frame using Python & plot graph using Pandas.

1. **Pie Chart**

Code:

```
from matplotlib import pyplot as plt
import numpy as np
Subject = ['C++','C','Java','Android','AI','ML','Python']
Credit = [50,20,70,80,65,40,99]
fig = plt.figure(figsize =(10, 7))
plt.pie(Credit, labels = Subject)
plt.show()
```

Output:

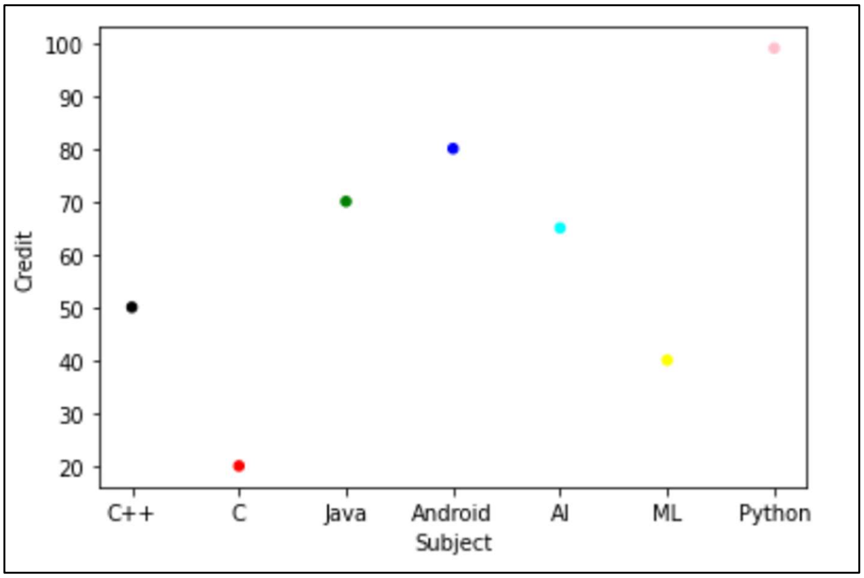


2. **Scatter Plot**

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Subject': ['C++','C','Java','Android','AI','ML','Python'],
        'Credit': [50,20,70,80,65,40,99]}
Color=['black', 'red', 'green', 'blue', 'cyan','yellow','pink']
df = pd.DataFrame(data,columns=['Subject','Credit'])
df.plot(x ='Subject', y='Credit', kind = 'scatter', color=Color)
plt.show()
```

Output:

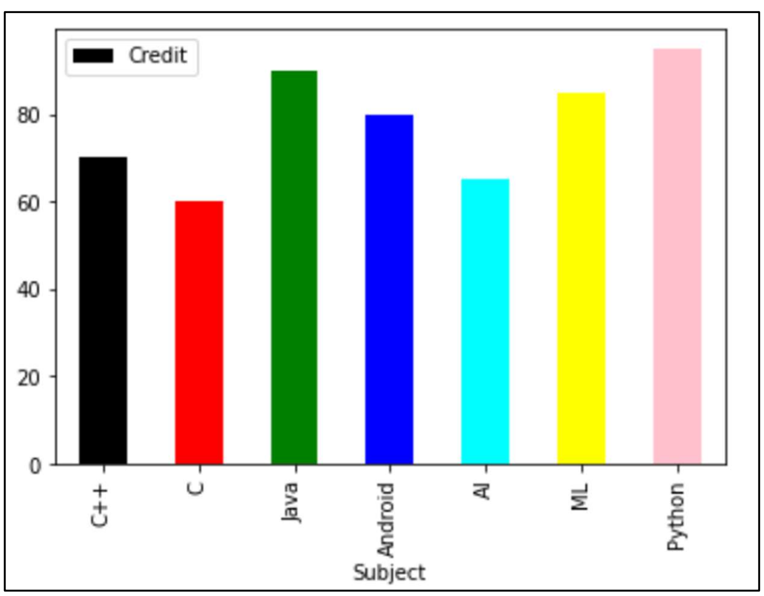


3. **Bar Chart**

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Subject': ['C++','C','Java','Android','AI','ML','Python'],
        'Credit': [70,60,90,80,65,85,95]}
Color=['black', 'red', 'green', 'blue', 'cyan','yellow','pink']
df = pd.DataFrame(data,columns=['Subject','Credit'])
df.plot(x ='Subject', y='Credit', kind = 'bar', color=Color)
plt.show()
```

Output:

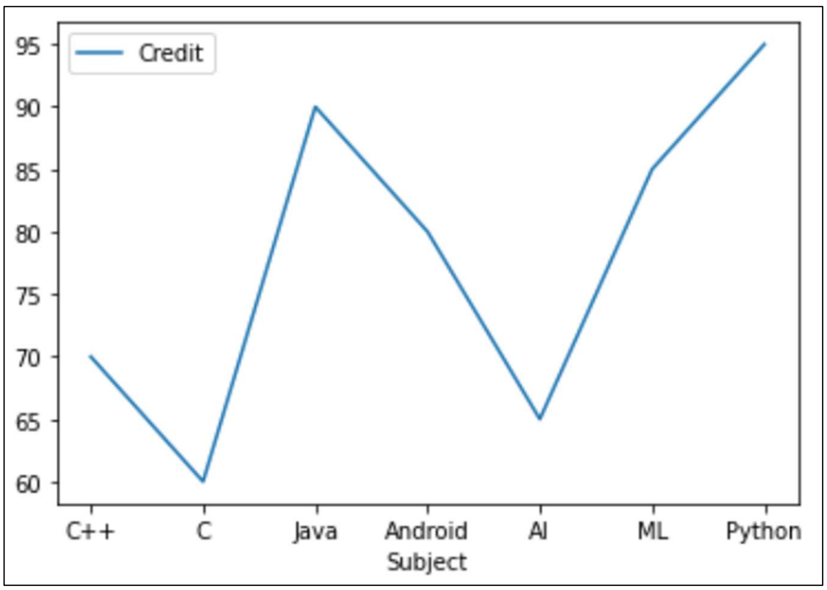


4. Line Chart

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Subject': ['C++','C','Java','Android','AI','ML','Python'],
        'Credit': [70,60,90,80,65,85,95]}
subject = pd.DataFrame(data,columns=['Subject','Credit'])
subject.plot(x ='Subject', y='Credit', kind = 'line')
plt.show()
```

Output:



PRACTICAL – 6

AIM: Implement Data Frame in Python to find missing value / replace value.

Code:

```
# importing pandas module
import pandas as pd

# loading data set
data = pd.read_csv(r'item.csv.csv')

# display the data
print("Data Before Filling Missing Values")
print(data)

# replacing missing values in quantity
# column with mean of that column
data['quantity'] = data['quantity'].fillna(data['quantity'].mean())

# replacing missing values in price column
# with median of that column
data['price'] = data['price'].fillna(data['price'].median())

# replacing missing values in bought column with
# standard deviation of that column
data['bought'] = data['bought'].fillna(data['bought'].std())

# replacing missing values in forenoon column with
# minimum number of that column
data['forenoon'] = data['forenoon'].fillna(data['forenoon'].min())

# replacing missing values in afternoon column with
# maximum number of that column
data['afternoon'] = data['afternoon'].fillna(data['afternoon'].max())
print('\nData After Filling Missing Values\n',data)
```

Output:

```
Data Before Filling Missing Values
    id       item  quantity  price  bought  forenoon  afternoon
0    1       milk       2.0   67.0   675.0     456.0        NaN
1    2      suger       1.0   90.0   586.0     365.0        NaN
2    3      chips       NaN    NaN     NaN     562.0        NaN
3    4     coffee       2.0   95.0   456.0     458.0        NaN
4    5       meat       4.0   65.0   750.0     526.0        NaN
5    6     chocos       3.0   70.0   653.0     652.0        NaN
6    7      juice       1.0   56.0   552.0       NaN      562.0
7    8        jam       NaN   78.0   632.0       NaN      625.0
8    9      bread       3.0   60.0   751.0       NaN      453.0
9   10     butter       4.0    NaN   546.0       NaN      459.0
10  11   biscuits       2.0   80.0     NaN       NaN      584.0
11  12     cheese       1.0   82.0   562.0       NaN      654.0
12  13  chocolate       NaN   71.0     NaN       NaN      563.0
```

```
Data After Filling Missing Values
    id       item  quantity  price     bought  forenoon  afternoon
0    1       milk       2.0   67.0  675.00000     456.0      654.0
1    2      suger       1.0   90.0  586.00000     365.0      654.0
2    3      chips       2.3   71.0   94.10284     562.0      654.0
3    4     coffee       2.0   95.0  456.00000     458.0      654.0
4    5       meat       4.0   65.0  750.00000     526.0      654.0
5    6     chocos       3.0   70.0  653.00000     652.0      654.0
6    7      juice       1.0   56.0  552.00000     365.0      562.0
7    8        jam       2.3   78.0  632.00000     365.0      625.0
8    9      bread       3.0   60.0  751.00000     365.0      453.0
9   10     butter       4.0   71.0  546.00000     365.0      459.0
10  11   biscuits       2.0   80.0   94.10284     365.0      584.0
11  12     cheese       1.0   82.0  562.00000     365.0      654.0
12  13  chocolate       2.3   71.0   94.10284     365.0      563.0
```

PRACTICAL – 7

AIM: Implement different pre-processing techniques:

A. Data Cleaning / Cleansing using Binning Method
B. Train Dataset & Test Dataset
C. ILOC
D. Find out Dependent & Independent variables

A. Data Cleaning / Cleansing using Binning Method

Code:

```
import numpy as np
import math
from sklearn.datasets import load_iris
from sklearn import datasets, linear_model, metrics

# load iris data set
dataset = load_iris()
a = dataset.data
b = np.zeros(150)

# take 1st column among 4 column of data set
for i in range (150):
    b[i]=a[i,1]

b=np.sort(b)  #sort the array

# create bins
bin1=np.zeros((30,5))
bin2=np.zeros((30,5))
bin3=np.zeros((30,5))

# Bin mean
for i in range (0,150,5):
    k=int(i/5)
    mean=(b[i] + b[i+1] + b[i+2] + b[i+3] + b[i+4])/5
    for j in range(5):
        bin1[k,j]=mean
print("Bin Mean: \n",bin1)
```

```
# Bin boundaries
for i in range (0,150,5):
    k=int(i/5)
    for j in range (5):
        if (b[i+j]-b[i]) < (b[i+4]-b[i+j]):
            bin2[k,j]=b[i]
        else:
            bin2[k,j]=b[i+4]
print("Bin Boundaries: \n",bin2)

# Bin median
for i in range (0,150,5):
    k=int(i/5)
    for j in range (5):
        bin3[k,j]=b[i+2]
print("Bin Median: \n",bin3)
```

Output:

```
Bin Mean:
 [[2.18 2.18 2.18 2.18 2.18]
 [2.34 2.34 2.34 2.34 2.34]
 [2.48 2.48 2.48 2.48 2.48]
 [2.52 2.52 2.52 2.52 2.52]
 [2.62 2.62 2.62 2.62 2.62]
 [2.7  2.7  2.7  2.7  2.7 ]
 [2.74 2.74 2.74 2.74 2.74]
 [2.8  2.8  2.8  2.8  2.8 ]
 [2.8  2.8  2.8  2.8  2.8 ]
 [2.86 2.86 2.86 2.86 2.86]
 [2.9  2.9  2.9  2.9  2.9 ]
 [2.96 2.96 2.96 2.96 2.96]
 [3.   3.   3.   3.   3.  ]
 [3.   3.   3.   3.   3.  ]
 [3.   3.   3.   3.   3.  ]
 [3.   3.   3.   3.   3.  ]
 [3.04 3.04 3.04 3.04 3.04]
 [3.1  3.1  3.1  3.1  3.1 ]
 [3.12 3.12 3.12 3.12 3.12]
 [3.2  3.2  3.2  3.2  3.2 ]
 [3.2  3.2  3.2  3.2  3.2 ]
 [3.26 3.26 3.26 3.26 3.26]
 [3.34 3.34 3.34 3.34 3.34]
 [3.4  3.4  3.4  3.4  3.4 ]
 [3.4  3.4  3.4  3.4  3.4 ]
 [3.5  3.5  3.5  3.5  3.5 ]
 [3.58 3.58 3.58 3.58 3.58]
 [3.74 3.74 3.74 3.74 3.74]
 [3.82 3.82 3.82 3.82 3.82]
 [4.12 4.12 4.12 4.12 4.12]]
```

```
Bin Boundaries:
 [[2.  2.3 2.3 2.3 2.3]
 [2.3 2.3 2.3 2.4 2.4]
 [2.4 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.6]
 [2.6 2.6 2.6 2.6 2.7]
 [2.7 2.7 2.7 2.7 2.7]
 [2.7 2.7 2.7 2.8 2.8]
 [2.8 2.8 2.8 2.8 2.8]
 [2.8 2.8 2.8 2.8 2.8]
 [2.8 2.8 2.9 2.9 2.9]
 [2.9 2.9 2.9 2.9 2.9]
 [2.9 2.9 3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.1 3.1]
 [3.1 3.1 3.1 3.1 3.1]
 [3.1 3.1 3.1 3.1 3.2]
 [3.2 3.2 3.2 3.2 3.2]
 [3.2 3.2 3.2 3.2 3.2]
 [3.2 3.2 3.3 3.3 3.3]
 [3.3 3.3 3.3 3.4 3.4]
 [3.4 3.4 3.4 3.4 3.4]
 [3.4 3.4 3.4 3.4 3.4]
 [3.5 3.5 3.5 3.5 3.5]
 [3.5 3.6 3.6 3.6 3.6]
 [3.7 3.7 3.7 3.8 3.8]
 [3.8 3.8 3.8 3.8 3.9]
 [3.9 3.9 3.9 4.4 4.4]]
```

```
Bin Median:
 [[2.2 2.2 2.2 2.2 2.2]
 [2.3 2.3 2.3 2.3 2.3]
 [2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5]
 [2.6 2.6 2.6 2.6 2.6]
 [2.7 2.7 2.7 2.7 2.7]
 [2.7 2.7 2.7 2.7 2.7]
 [2.8 2.8 2.8 2.8 2.8]
 [2.8 2.8 2.8 2.8 2.8]
 [2.9 2.9 2.9 2.9 2.9]
 [2.9 2.9 2.9 2.9 2.9]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.  3.  3.  3.  3. ]
 [3.1 3.1 3.1 3.1 3.1]
 [3.1 3.1 3.1 3.1 3.1]
 [3.2 3.2 3.2 3.2 3.2]
 [3.2 3.2 3.2 3.2 3.2]
 [3.3 3.3 3.3 3.3 3.3]
 [3.3 3.3 3.3 3.3 3.3]
 [3.4 3.4 3.4 3.4 3.4]
 [3.4 3.4 3.4 3.4 3.4]
 [3.5 3.5 3.5 3.5 3.5]
 [3.6 3.6 3.6 3.6 3.6]
 [3.7 3.7 3.7 3.7 3.7]
 [3.8 3.8 3.8 3.8 3.8]
 [4.1 4.1 4.1 4.1 4.1]]
```

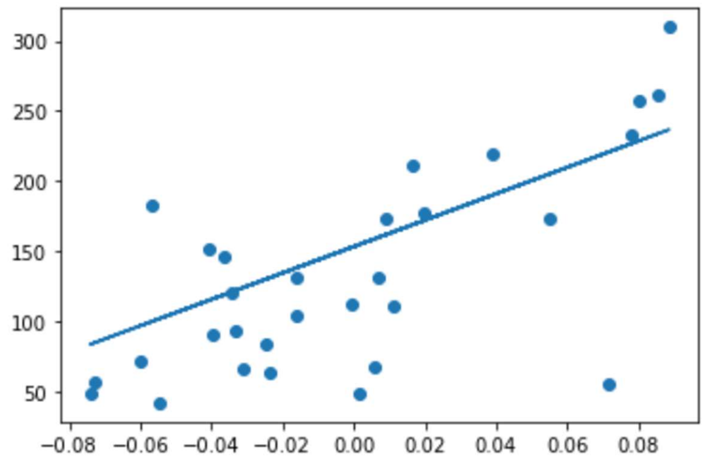
B. Train Dataset & Test Dataset

Code:

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

diabetes = datasets.load_diabetes()
diabetes_X = diabetes.data[:, np.newaxis, 2]
#print(diabetes_X)

diabetes_X_train = diabetes_X[:-30]
diabetes_X_test = diabetes_X[-30:]

diabetes_Y_train = diabetes.target[:-30]
diabetes_Y_test = diabetes.target[-30:]
model = linear_model.LinearRegression()
model.fit(diabetes_X_train, diabetes_Y_train)

diabetes_Y_Predicted = model.predict(diabetes_X_test)

print("Mean Squared error is:",
mean_squared_error(diabetes_Y_test,diabetes_Y_Predicted))
print("Weight: ", model.coef_)
print("Intercept: ", model.intercept_)

plt.scatter(diabetes_X_test, diabetes_Y_test)
plt.plot(diabetes_X_test, diabetes_Y_Predicted)
plt.show()
```

Output:

```
Mean Squared error is: 3035.0601152912695
Weight:  [941.43097333]
Intercept:  153.39713623331698
```



C. **ILOC**

Code:

```
import pandas as pd
data = pd.read_csv(r"C:\Users\Dell\Desktop\ML\nba.csv")

# retrieving rows by loc method
row1 = data.loc[3]
# retrieving rows by iloc method
row2 = data.iloc[3]
# checking if values are equal
print("Comparing Single Rows")
row1 == row2
```

Output:

```
Comparing Single Rows

Name        True
Team        True
Number      True
Position    True
Age         True
Height      True
Weight      True
College     True
Salary      True
Name: 3, dtype: bool
```

Code:

```
import pandas as pd
data = pd.read_csv(r"C:\Users\Dell\Desktop\ML\nba.csv")

# retrieving rows by loc method
row1 = data.iloc[[4, 5, 6, 7]]
# retrieving rows by loc method
row2 = data.iloc[4:8]
# comparing values
print("Comparing Multiple Rows")
row1 == row2
```

Output:

Comparing Multiple Rows

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
4	True	True	True	True	True	True	True	False	True
5	True	True	True	True	True	True	True	False	True
6	True	True	True	True	True	True	True	True	True
7	True	True	True	True	True	True	True	True	True

D. Find out Dependent & Independent variables

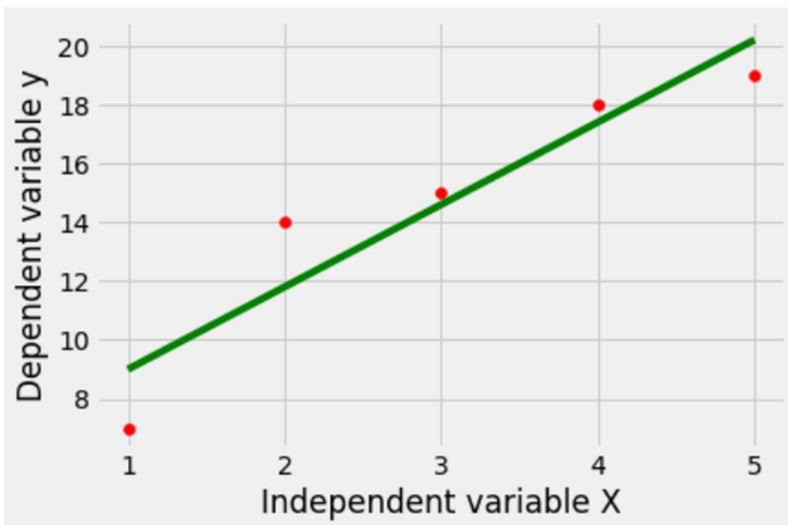
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

x = np.array([1,2,3,4,5])
y = np.array([7,14,15,18,19])
n = np.size(x)

x_mean = np.mean(x)
y_mean = np.mean(y)

Sxy = np.sum(x*y)- n*x_mean*y_mean
Sxx = np.sum(x*x)-n*x_mean*x_mean

b1 = Sxy/Sxx
b0 = y_mean-b1*x_mean
```

```
print('slope b1 is', b1)
print('intercept b0 is', b0)

y_pred = b1 * x + b0

plt.scatter(x, y, color = 'red')
plt.plot(x, y_pred, color = 'green')
plt.xlabel('Independent variable X')
plt.ylabel('Dependent variable y')
```

Output:

```
slope b1 is 2.8
intercept b0 is 6.200000000000001

Text(0, 0.5, 'Dependent variable y')
```



PRACTICAL – 8

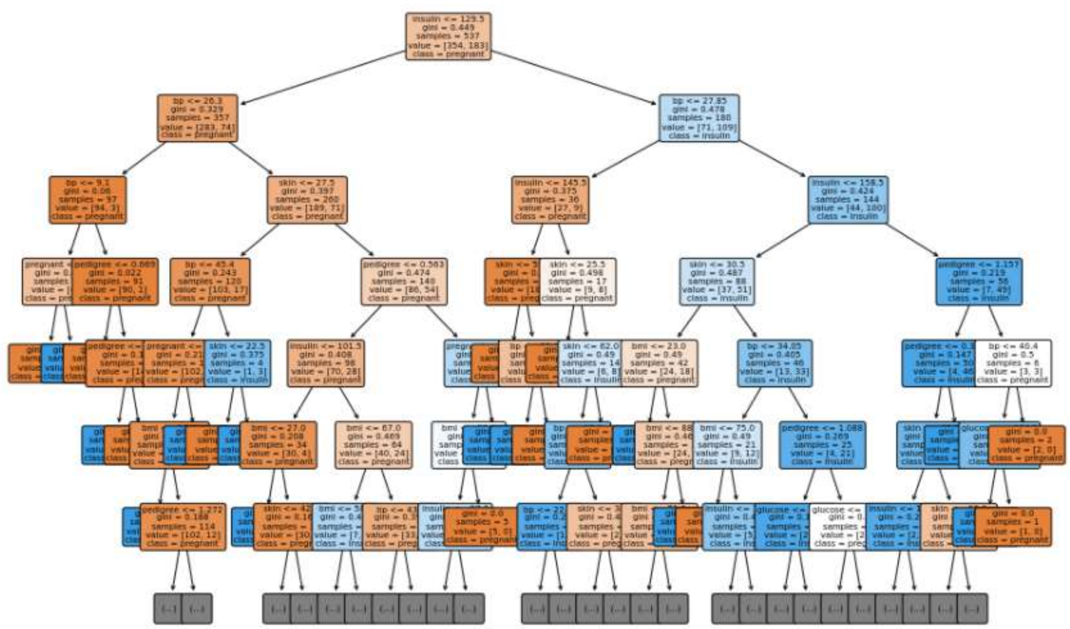
AIM: Implement Decision tree using Python.

Code:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix#for visualizing tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv(r"C:\Users\Dell\Desktop\ML\diabetes2.csv", header=None, names=col_names)
print(pima.head())
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("\nAccuracy:",metrics.accuracy_score(y_test, y_pred))
fig = plt.figure(figsize=(15, 10))
dec_tree = plot_tree(decision_tree=clf, feature_names = col_names,
            class_names =['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree'] , max_depth = 6, filled = True , impurity=True, rounded = True, fontsize = 7)
```

Output:

```
   pregnant  glucose  bp  skin  insulin   bmi  pedigree  age  label
0         6      148  72    35        0  33.6     0.627   50      1
1         1       85  66    29        0  26.6     0.351   31      0
2         8      183  64     0        0  23.3     0.672   32      1
3         1       89  66    23       94  28.1     0.167   21      0
4         0      137  40    35      168  43.1     2.288   33      1

Accuracy: 0.670995670995671
```



PRACTICAL – 9

AIM: Implement Regression technique using Python.

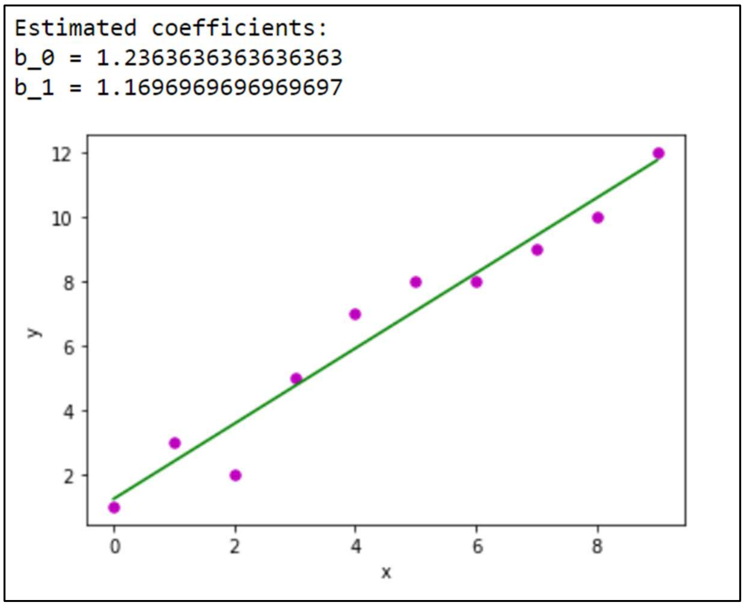
1. **Single Regression**

Code:

```
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 = "m",
    marker = "o", 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)
main()
```

Output:



2. **Multiple Regression**

Code:

```
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:

```
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: 0.720905667266178
```

PRACTICAL – 10

AIM: Implement Bay's Theorem using Python.

Code:

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X = iris.data[:, :4]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print ("Accuracy : ", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n",cm)
```

Output:

```
Accuracy :  0.9666666666666667
Confusion Matrix:
 [[ 9  0  0]
 [ 0  7  0]
 [ 0  1 13]]
```

PRACTICAL – 11

AIM: Implement K-NN algorithm using Python.

Code:

```
import sklearn
from sklearn.utils import shuffle
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import numpy as np
from sklearn import linear_model, preprocessing
data = pd.read_csv(r"C:\Users\Dell\Desktop\ML\car.data")
le = preprocessing.LabelEncoder()
buying = le.fit_transform(list(data["buying"]))
maint = le.fit_transform(list(data["maint"]))
door = le.fit_transform(list(data["door"]))
persons = le.fit_transform(list(data["persons"]))
lug_boot = le.fit_transform(list(data["lug_boot"]))
safety = le.fit_transform(list(data["safety"]))
cls = le.fit_transform(list(data["class"]))
predict = "class"
X = list(zip(buying, maint, door, persons, lug_boot, safety))
y = list(cls)
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y, test_size = 0.1)
model = KNeighborsClassifier(n_neighbors=9)
model.fit(x_train, y_train)
acc = model.score(x_test, y_test)
print('\n',acc,'\n')
predicted = model.predict(x_test)
names = ["unacc", "acc", "good", "vgood"]
for x in range(len(predicted)):
    print("Predicted: ", names[predicted[x]], "|| Actual: ", names[y_test[x]])
    n = model.kneighbors([x_test[x]], 9, True)
```

Output:

```
0.9364161849710982

Predicted:  unacc || Actual:  unacc
Predicted:  unacc || Actual:  unacc
Predicted:  good || Actual:  good
Predicted:  good || Actual:  unacc
Predicted:  good || Actual:  good
Predicted:  unacc || Actual:  acc
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  good || Actual:  good
Predicted:  vgood || Actual:  vgood
```

PRACTICAL – 12

AIM: Implement K-means algorithm using Python.

Code:

```
# importing dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
# creating data
mean_01 = np.array([0.0, 0.0])
cov_01 = np.array([[1, 0.3], [0.3, 1]])
dist_01 = np.random.multivariate_normal(mean_01, cov_01, 100)
mean_02 = np.array([6.0, 7.0])
cov_02 = np.array([[1.5, 0.3], [0.3, 1]])
dist_02 = np.random.multivariate_normal(mean_02, cov_02, 100)
mean_03 = np.array([7.0, -5.0])
cov_03 = np.array([[1.2, 0.5], [0.5, 1, 3]])
dist_03 = np.random.multivariate_normal(mean_03, cov_01, 100)
mean_04 = np.array([2.0, -7.0])
cov_04 = np.array([[1.2, 0.5], [0.5, 1, 3]])
dist_04 = np.random.multivariate_normal(mean_04, cov_01, 100)
data = np.vstack((dist_01, dist_02, dist_03, dist_04))
np.random.shuffle(data)
# function to plot the selected centroids
def plot(data, centroids):
    plt.scatter(data[:, 0], data[:, 1], marker='.',
                color='gray', label='data points')
    plt.scatter(centroids[:-1, 0], centroids[:-1, 1],
                color='black', label='previously selected centroids')
    plt.scatter(centroids[-1, 0], centroids[-1, 1],
                color='red', label='next centroid')
    plt.title('Select % d th centroid' % (centroids.shape[0]))
    plt.legend()
    plt.xlim(-5, 12)
    plt.ylim(-10, 15)
```

```
    plt.show()
# function to compute euclidean distance
def distance(p1, p2):
    return np.sum((p1 - p2) ** 2)
# initialization algorithm
def initialize(data, k):
    '''
    initialized the centroids for K-means++
    inputs:
        data - numpy array of data points having shape (200, 2)
        k - number of clusters
    '''
    ## initialize the centroids list and add
    ## a randomly selected data point to the list
    centroids = []
    centroids.append(data[np.random.randint(
        data.shape[0]), :])
    plot(data, np.array(centroids))
    ## compute remaining k - 1 centroids
    for c_id in range(k - 1):
        ## initialize a list to store distances of data
        ## points from nearest centroid
        dist = []
        for i in range(data.shape[0]):
            point = data[i, :]
            d = sys.maxsize
            ## compute distance of 'point' from each of the previously
            ## selected centroid and store the minimum distance
            for j in range(len(centroids)):
                temp_dist = distance(point, centroids[j])
                d = min(d, temp_dist)
            dist.append(d)
        ## select data point with maximum distance as our next centroid
        dist = np.array(dist)
        next_centroid = data[np.argmax(dist), :]
        centroids.append(next_centroid)
```

```
        dist = []
        plot(data, np.array(centroids))
    return centroids
# call the initialize function to get the centroids
centroids = initialize(data, k=4)
```

Output: