QIS

COLLEGE OF ENGINEEING AND TECHNOLOGY

(AUTONOMOUS)

(Approved by AICTE., Permanently Affiliated to JNTUK, Kakinada, Accredited by NBA & UGC Recognized)
Vengamukkapalem, Pondur Road, ONGOLE-523 272, A.P.

LAB RECORD

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

SUBJECT	:	MACHINE LEARNING LAB
COURSE	:	B. TECH.
YEAR & SEM	:	III & II
A.Y.	:	2023-24

QIS COLLEGE OF ENGINEERING & TECHNOLOGY,

(AUTONOMOUS)

ONGOLE.

Department of

Name	:	
Branch	:	
Roll No.	:	
Year & Sem	:	III & II
Section	:	
Subject	:	Machine Learning Lab

Certified that this is bonafied record of the workdone

by Mr./Ms. ……………………………………………………………………………………

During the Year: 2023 - 24

Head of the Section

Ext. Examiner

1.

Staff in-charge **2.**

INDEX

EXPERIMENT-1

Aim: Write a python program to compute

- Central Tendency Measures: Mean, Median, Mode.
- Measure of Dispersion: Variance, Standard Deviation.

Source code:

```
import math
import collections
data = [2, 4, 3, 6, 4, 5]
n = len(data)

# Central tendancy Measures
#Mean
sum = 0
for i in data:
  sum += i
  mean = sum / n
print('Mean: ', mean)

#Median
data_sort = sorted(data)
if n%2 == 0:
  median = (data_sort[(n//2)-1] + data_sort[n//2])/2
else:
  median = data_sort((n//2)-1)
print('Median: ',median)

#Mode
mode = collections.Counter(data).most_common(1)[0][0]
print('Mode: ', mode)

#Dispersion Measures
#Variance
sum_var = 0
for i in data:
  sum_var += math.pow(i - mean, 2)
  var = sum_var / (n-1)
print('Variance: ', round(var,2))

#Standard deviation
std = round(math.sqrt(var),2)
print('Standard Deviation', std)
```

Output:

```
Mean:  4.0
Median:  4.0
Mode:  4
Variance:  2.0
Standard Deviation 1.41
```

Result: Successfully computed central tendency measures and dispersion measures.

EXPERIMENT-2

Aim: Study of Python Basic Libraries such as Statistics, Math and Numpy.

Source code:

Statistics Module:

```
import statistics as st
data = [5, 4, 1, 3, 2, 4, 5, 4, 5, 6]
print('Mean: ', st.mean(data))
print('Median: ', st.median(data))
print('Mode: ', st.mode(data))
print('Variance: ', round(st.variance(data),2))
print('Standard Deviation: ', round(st.stdev(data),2))
```

Output:

```
Mean:  3.9
Median:  4.0
Mode:  5
Variance:  2.32
Standard Deviation:  1.52
```

Math module:

```
import math
#constants
print("Exponential value: ", math.e)
print("Pi value: ",math.pi)
print("Infinite value: ", math.inf)
print("Not a number value: ", math.nan)

#methods
print("Logarithmic: ", math.log(math.e))
print("factorial of 5 is", math.factorial(5))
print("GCD of 64 and 42 is", math.gcd(64,42))
print("Floor of 5.67 is", math.floor(5.67))
print("Ceil of 5.67 is", math.ceil(5.67))
```

Output:

```
Exponential value:  2.718281828459045
Pi value:  3.141592653589793
Infinite value:  inf
Not a number value: nan
Logarithmic: 1.0
factorial of 5 is 120
GCD of 64 and 42 is 2
Floor of 5.67 is 5
Ceil of 5.67 is 6
```

Numpy module:

```
import numpy as np
#Creating arrays
ar1d = np.arange(11,17)
ar2d = np.arange(11,36).reshape(5,5)
print("1-D array is:")
print(ar1d)
print("2-D array is:")
print(ar2d)

#Properties
print("ar1d shape, size, dimensions are", ar1d.shape, ar1d.size,
                                                      ar1d.ndim)
print("ar2d shape, size, dimensions are", ar2d.shape, ar2d.size,
                                                      ar2d.ndim)

#indexing
print("Indexing on ar1d:", ar1d[2],ar1d[-2])
print("Indexing on ar2d:", ar2d[2][1],ar2d[1][1])

#slicing
print("Slicing on ar1d:", ar1d[2:6])
print("Slicing on ar2d:")
print(ar2d[1:4,2:4])
```

Output:

```
1-D array is:
[11 12 13 14 15 16]
2-D array is:
 [[11 12 13 14 15]
  [16 17 18 19 20]
  [21 22 23 24 25]
  [26 27 28 29 30]
  [31 32 33 34 35]]
ar1d shape, size, dimensions are (6,) 6 1
ar2d shape, size, dimensions are (5, 5) 25 2
Indexing on ar1d: 13 15
Indexing on ar2d: 22 17
Slicing on ar1d: [13 14 15 16]
Slicing on ar2d:
 [[18 19]
  [23 24]
  [28 29]]
```

Result: Successfully worked on statistics, math and numpy modules.

Experiment-3:

Aim: Study of Python Libraries for ML application such as Pandas and Matplotlib.

Source code:

Pandas module:

```
import pandas as pd
import numpy as np
from numpy import random

# Series Creation
ser1 = pd.Series(data=random.randint(10,45,size=5),
                 index=['a','b','c','d','e'])
print("Series is")
print(ser1)

#DataFrame Creation
df = pd.DataFrame(data=np.arange(101,126).reshape(5,5),
                  index=['A','B','C','D','E'],
                  columns=['U','V','W','X','Y'])
print("Data frame is")
print(df)

print("Column wise accessing")
print(df['W']['A'])
print(df['W'])
print(df[['W','X','U']])

print("Row wise accsessing")
print(df.loc['A']['X'])
print(df.loc['B'])
print(df.loc[['B','A']])

print(df.iloc[2]['X'])
print(df.iloc[1])
print(df.iloc[2:4])
```

Output:

```
Series is
a    20
b    43
c    10
d    32
e    21
dtype: int64
Data frame is
     U    V    W    X    Y
A  101  102  103  104  105
B  106  107  108  109  110
C  111  112  113  114  115
D  116  117  118  119  120
E  121  122  123  124  125
Column wise accessing
103
```

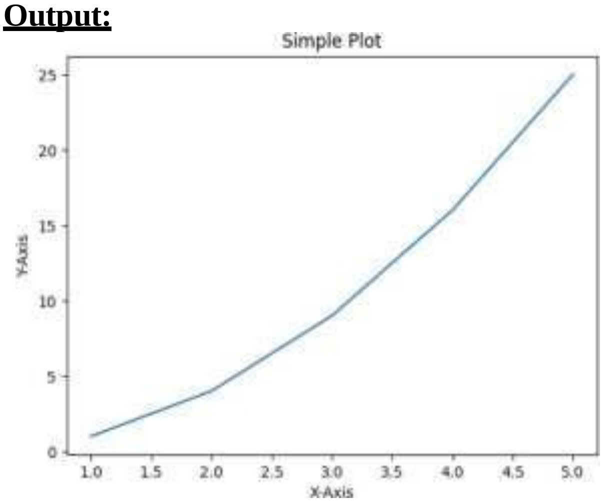
```
A    103
B    108
C    113
D    118
E    123
Name: W, dtype: int64
     W    X    U
A  103  104  101
B  108  109  106
C  113  114  111
D  118  119  116
E  123  124  121
Row wise accsessing
104
U    106
V    107
W    108
X    109
Y    110
Name: B, dtype: int64
     U    V    W    X    Y
B  106  107  108  109  110
A  101  102  103  104  105
114
U    106
V    107
W    108
X    109
Y    110
Name: B, dtype: int64
     U    V    W    X    Y
C  111  112  113  114  115
D  116  117  118  119  120
```

Matplotlib module:

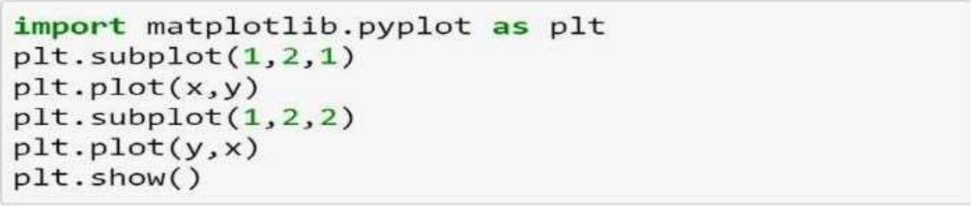
Basic Plot:

```
import matplotlib.pyplot as plt
import numpy as np
x  = np.arange(1,6)
y = x**2
plt.plot(x,y)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.title("Simple Plot")
plt.show()
```

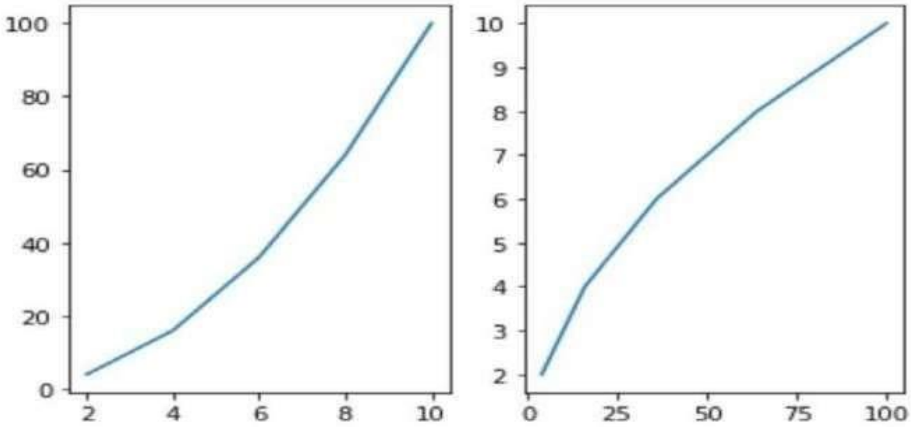
Output:



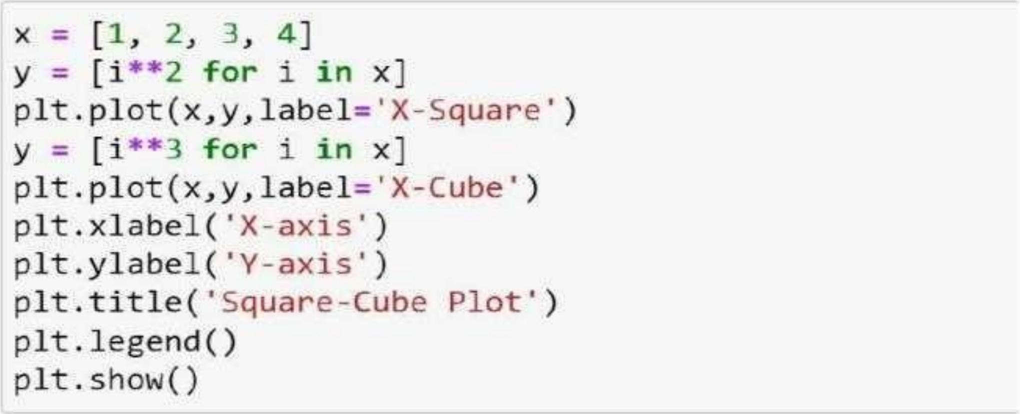
Creating subplots:



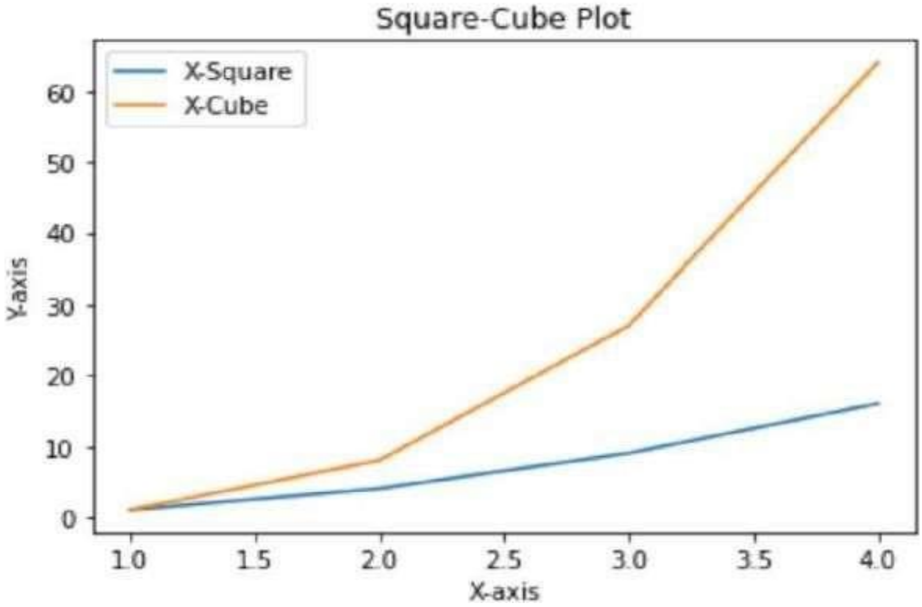
```
import matplotlib.pyplot as plt
plt.subplot(1,2,1)
plt.plot(x,y)
plt.subplot(1,2,2)
plt.plot(y,x)
plt.show()
```



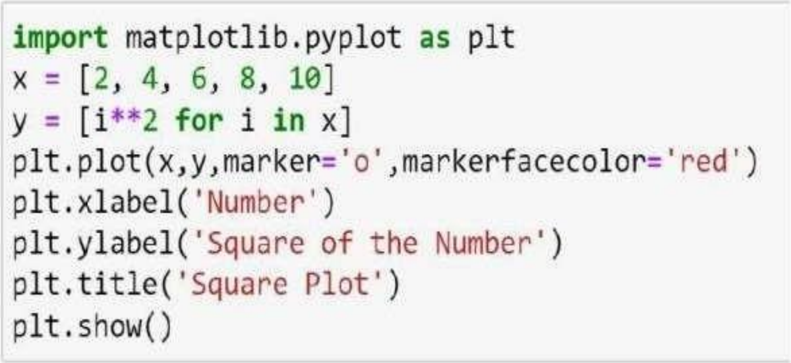
Adding legend to plot:



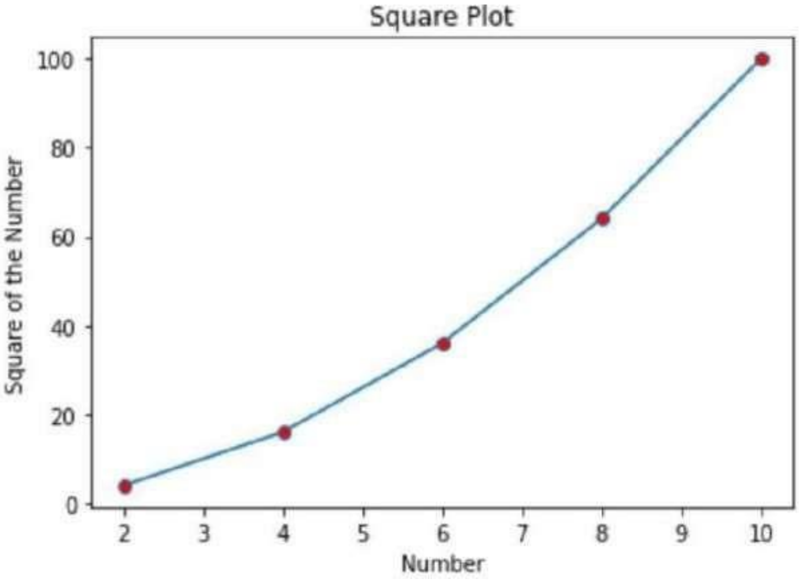
```
x = [1, 2, 3, 4]
y = [i**2 for i in x]
plt.plot(x,y,label='X-Square')
y = [i**3 for i in x]
plt.plot(x,y,label='X-Cube')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Square-Cube Plot')
plt.legend()
plt.show()
```



Adding markers to plot:

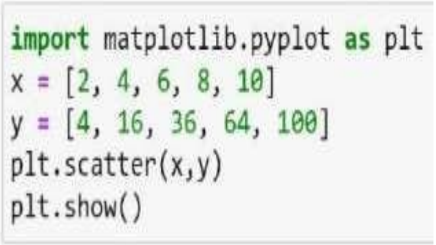


```
import matplotlib.pyplot as plt
x = [2, 4, 6, 8, 10]
y = [i**2 for i in x]
plt.plot(x,y,marker='o',markerfacecolor='red')
plt.xlabel('Number')
plt.ylabel('Square of the Number')
plt.title('Square Plot')
plt.show()
```

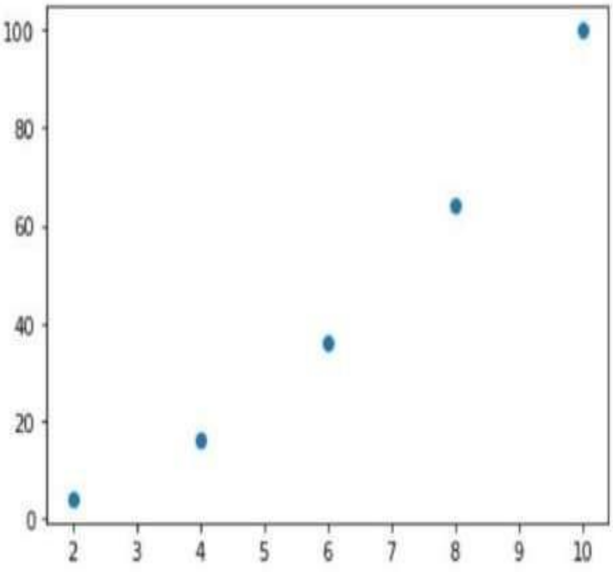


Different types of plots:

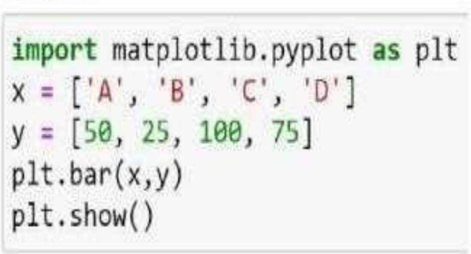
Scatter plot:



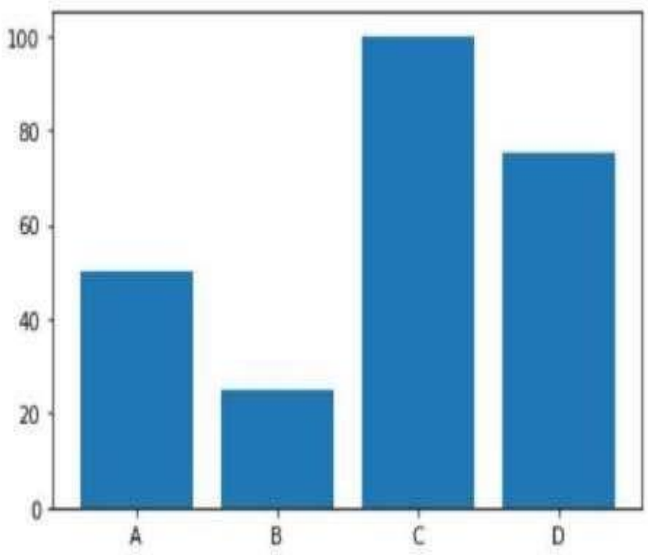
```
import matplotlib.pyplot as plt
x = [2, 4, 6, 8, 10]
y = [4, 16, 36, 64, 100]
plt.scatter(x,y)
plt.show()
```



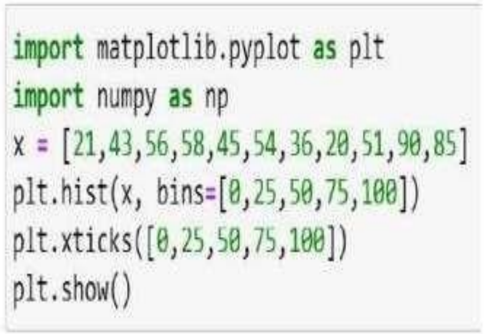
Bar plot:



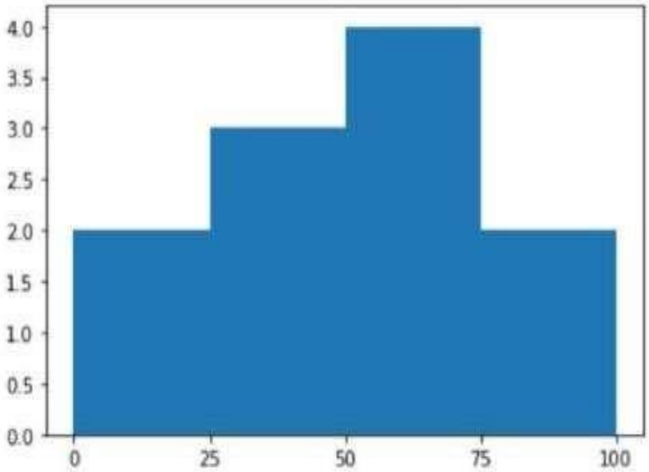
```
import matplotlib.pyplot as plt
x = ['A', 'B', 'C', 'D']
y = [50, 25, 100, 75]
plt.bar(x,y)
plt.show()
```



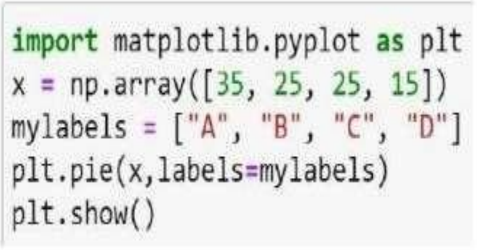
Histogram:



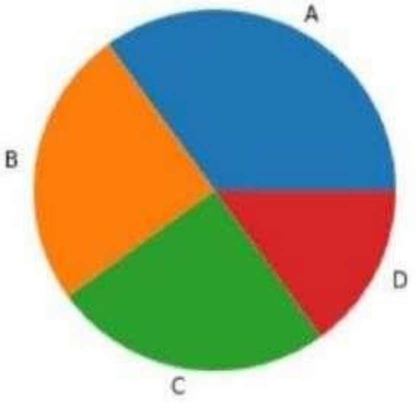
```
import matplotlib.pyplot as plt
import numpy as np
x = [21,43,56,58,45,54,36,20,51,90,85]
plt.hist(x, bins=[0,25,50,75,100])
plt.xticks([0,25,50,75,100])
plt.show()
```



Pie chart:



```
import matplotlib.pyplot as plt
x = np.array([35, 25, 25, 15])
mylabels = ["A", "B", "C", "D"]
plt.pie(x,labels=mylabels)
plt.show()
```



Result: Successfully worked on Pandas and Matplotlib libraries.

EXPERIMENT-4

Aim: Write a Python program to implement Simple Linear Regression.

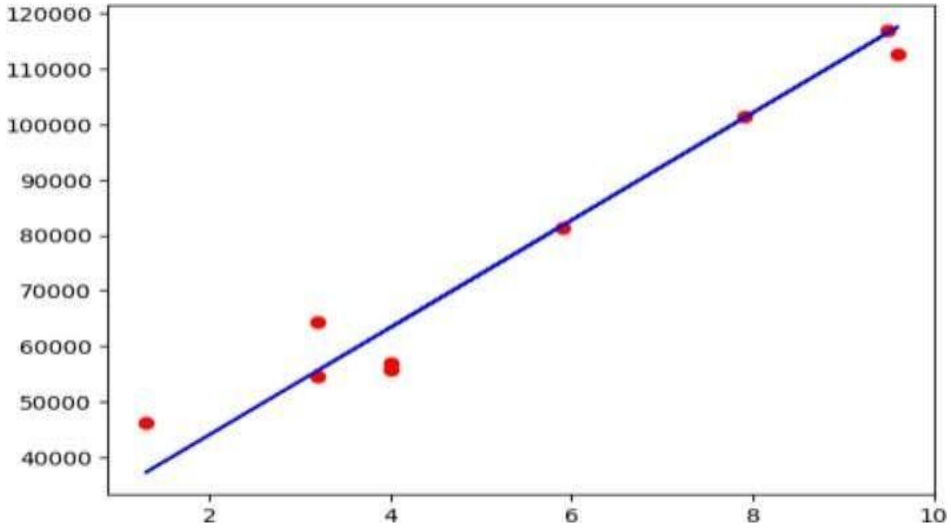
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
#Loading dataset
dataset = pd.read_csv('Salary_Data.csv')
#Feature Extraction
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:,-1].values
#splitting Train & Test data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
#Linear Rgression Model Creation
reg_model = LinearRegression()
#Model Training (Fit the model)
reg_model.fit(X_train, y_train)
#Model Prediction
y_pred = reg_model.predict(X_test)
#Finding R-Sqaure value
print("R-Sqaure value(accuracy):",r2_score(y_test,y_pred))
#Visualizing the graph
plt.scatter(X_test,y_test, color='red')
plt.plot(X_test, y_pred,color='blue')
plt.show()
```

Output:

```
R-Sqaure value (Represents accuracy): 0.9524505593397691
```



Result: Successfully implemented Simple Linear Regression model.

EXPERIMENT-5

Aim: Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

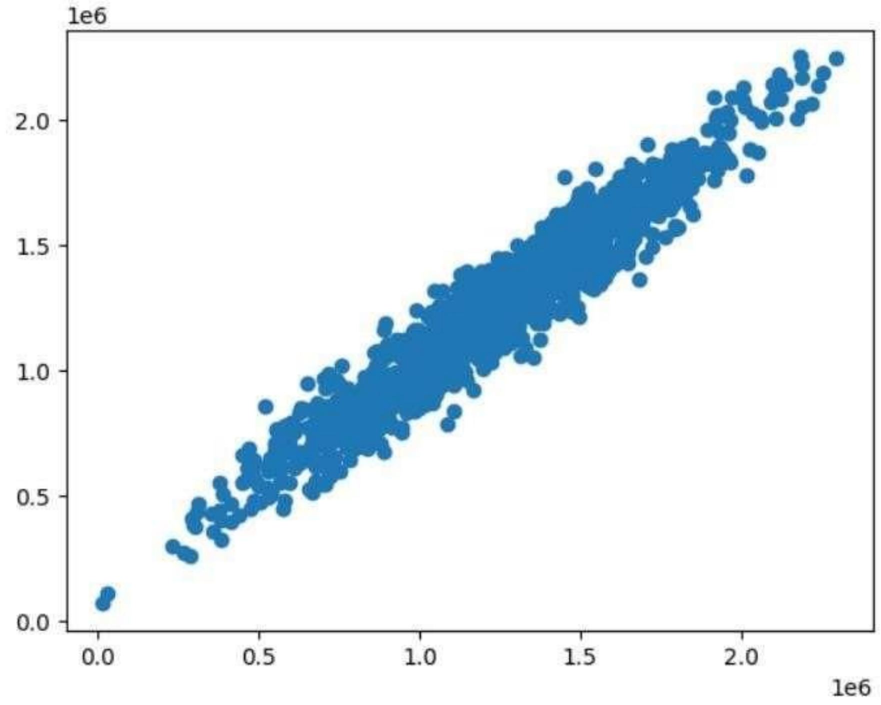
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

#Loading data set
house_df = pd.read_csv('housing.csv')
print("Housing dataset columns:")
print(house_df.columns)

#Feature Extraction
X = house_df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area 
Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
y = house_df['Price']

#Splitting Train and Test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.3)

# Linear Regresion Model Creation
reg_model = LinearRegression()

#Traing the model (fit)
reg_model.fit(X_train, y_train)

#Model Prediction
y_pred = reg_model.predict(X_test)

#Accuracy of model
print("R-Sqaure value", r2_score(y_test, y_pred))

#Visualization
plt.scatter(y_test, y_pred)
plt.show()
```

Output:

```
Housing dataset columns:
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number
of Rooms','Avg. Area Number of Bedrooms', 'Area Population', 'Price',
'Address'], dtype='object')
R-Sqaure value 0.9237308710840247
```



Result: Successfully implemented Multiple Linear Regression.

EXPERIMENT-6

Aim: Implementation of Decision tree using sklearn and its parameter tuning.

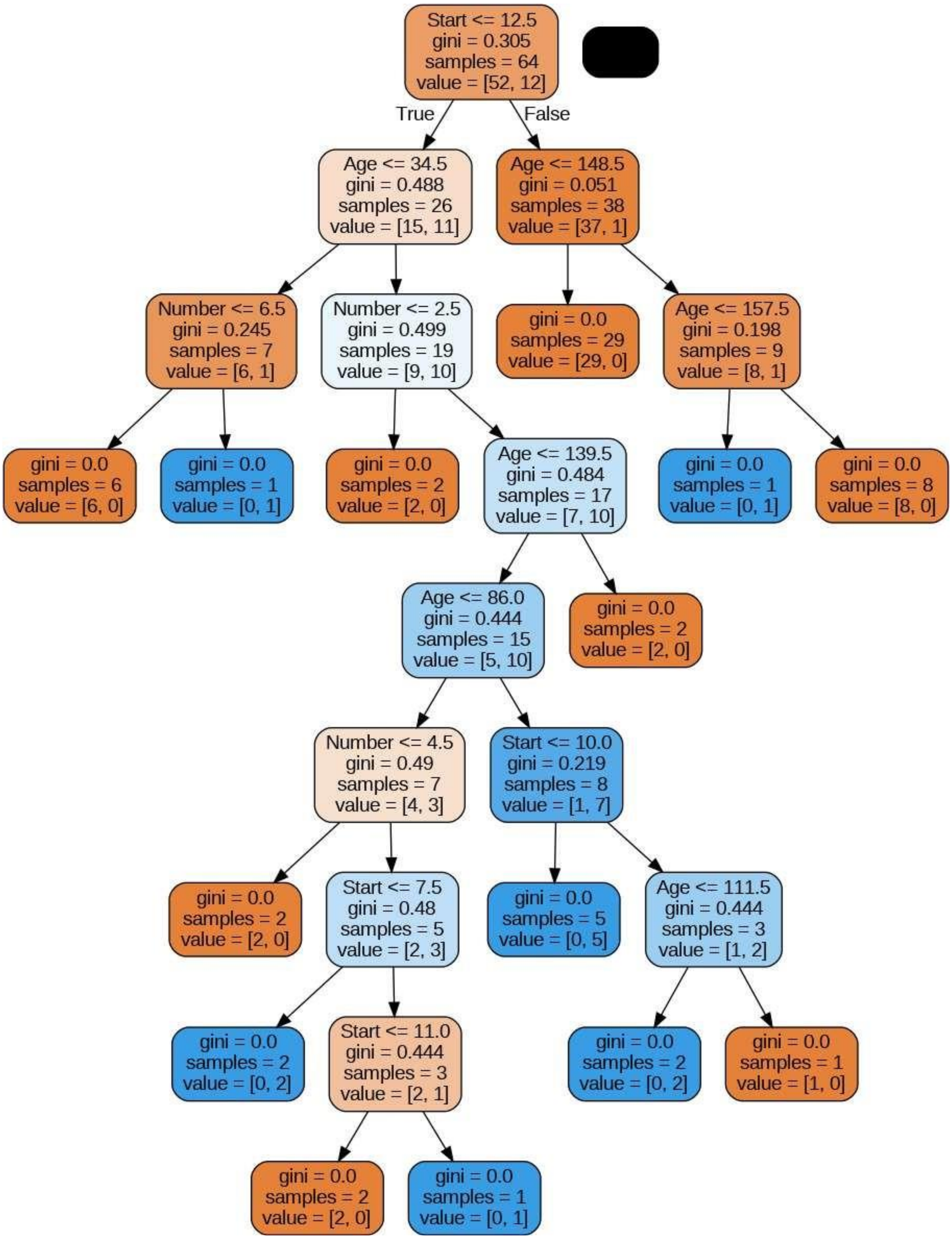
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,confusion_matrix,
                                                    accuracy_score
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydot

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

X = df.drop('Kyphosis',axis=1)
y = df['Kyphosis']

X_train,X_test,y_train,y_test= train_test_split(X, y, test_size=0.20)

dtree = DecisionTreeClassifier()

dtree.fit(X_train,y_train)

predictions = dtree.predict(X_test)

print("Accuracy Score", accuracy_score(y_test, predictions))
print("Confusion Matrix")
print(confusion_matrix(y_test,predictions))

#Visualization
features = list(df.columns[1:])
dot_data = StringIO()
export_graphviz(dtree,out_file=dot_data,feature_names=features,
                                        filled=True,rounded=True)
graph = pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

Output:

```
Accuracy Score 0.8823529411764706
Confusion Matrix
[[11  1]
 [ 1  4]]
```



Result: Successfully implemented Decision tree model.

EXPERIMENT-7

Aim: Implementation of KNN using sklearn.

Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,confusion_matrix


df = pd.read_csv("ClassifiedData.csv",index_col=0)
scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))

df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
X_train,X_test,y_train, y_test = train_test_split(scaled_features,
                                df['TARGET CLASS'], test_size=0.30)

#Initially with K=1
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(X_train,y_train)
pred1 = knn1.predict(X_test)
print("For K=1 results are:")
print(confusion_matrix(y_test,pred1))
print(classification_report(y_test,pred1))

# NOW WITH K=23
knn23 = KNeighborsClassifier(n_neighbors=23)

knn23.fit(X_train,y_train)
pred23 = knn23.predict(X_test)

print("For K=23 results are:")
print(confusion_matrix(y_test,pred23))
print(classification_report(y_test,pred23))
```

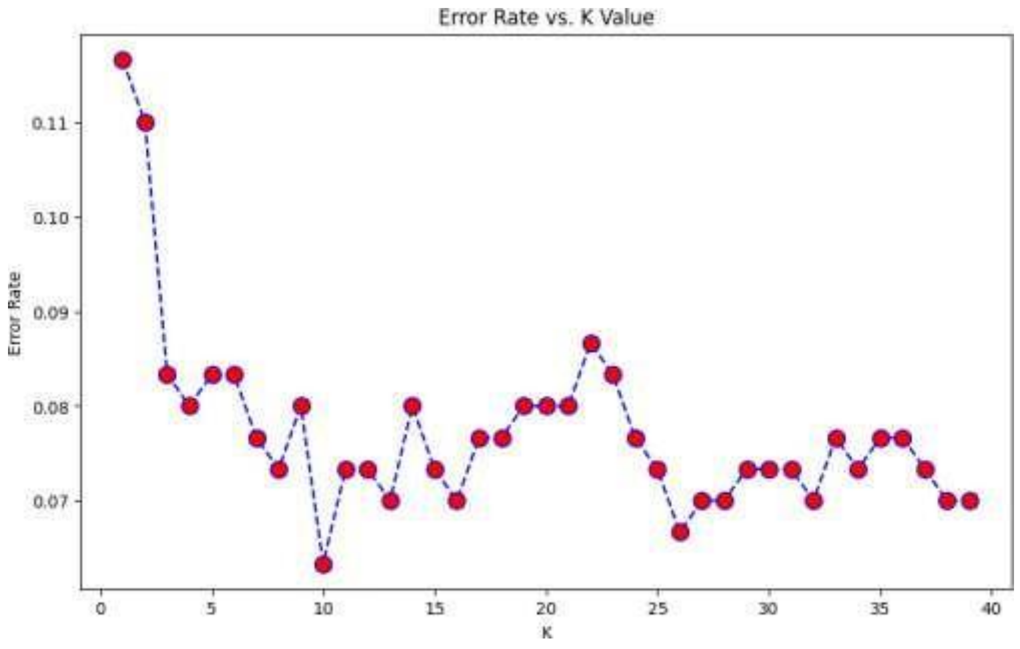
Output:

```
For K=1 results are:
[[128  17]
 [ 18 137]]
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       145
           1       0.89      0.88      0.89       155

    accuracy                           0.88       300
```

```
   macro avg       0.88      0.88      0.88       300
weighted avg       0.88      0.88      0.88       300
For K=23 results  are:
[[130  15]
 [ 10 145]]
              precision    recall  f1-score   support

           0       0.93      0.90      0.91       145
           1       0.91      0.94      0.92       155

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300
```

Choosing K Value:

```
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',
                   marker='o', markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```



Result: Successfully implemented KNN Classification model.

EXPERIMENT-8

Aim: Implementation of Logistic Regression using sklearn.

Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

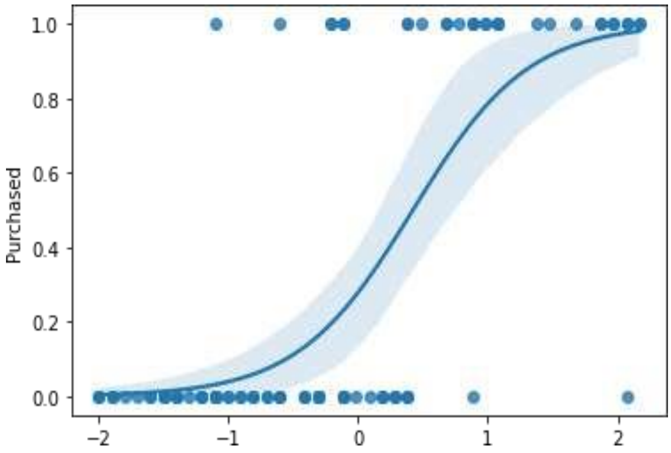
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
dataset = pd.read_csv('Social_Network_Ads.csv')
print(dataset.columns)
X = dataset[['Age', 'EstimatedSalary']]
y = dataset['Purchased']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25)
#feature scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("Accuracy Score",accuracy_score(y_test, y_pred))
```

Output:

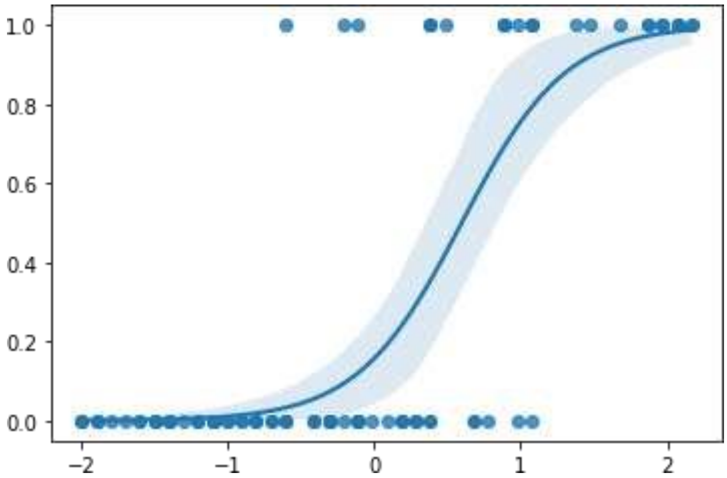
```
Index(['Age', 'EstimatedSalary', 'Purchased'], dtype='object')
Confusion Matrix
[[58  8]
 [10 24]]
Accuracy Score 0.82
```

Model Visualization:

```
sns.regplot(x=X_test[:,:-1], y=y_test, logistic=True)
```



```
sns.regplot(x=X_test[:,:-1], y=y_pred, logistic=True)
```



Result: Successfully implemented Logistic Regression Model.

EXPERIMENT-9

Aim: Implementation of K-Means Clustering.

Source code:

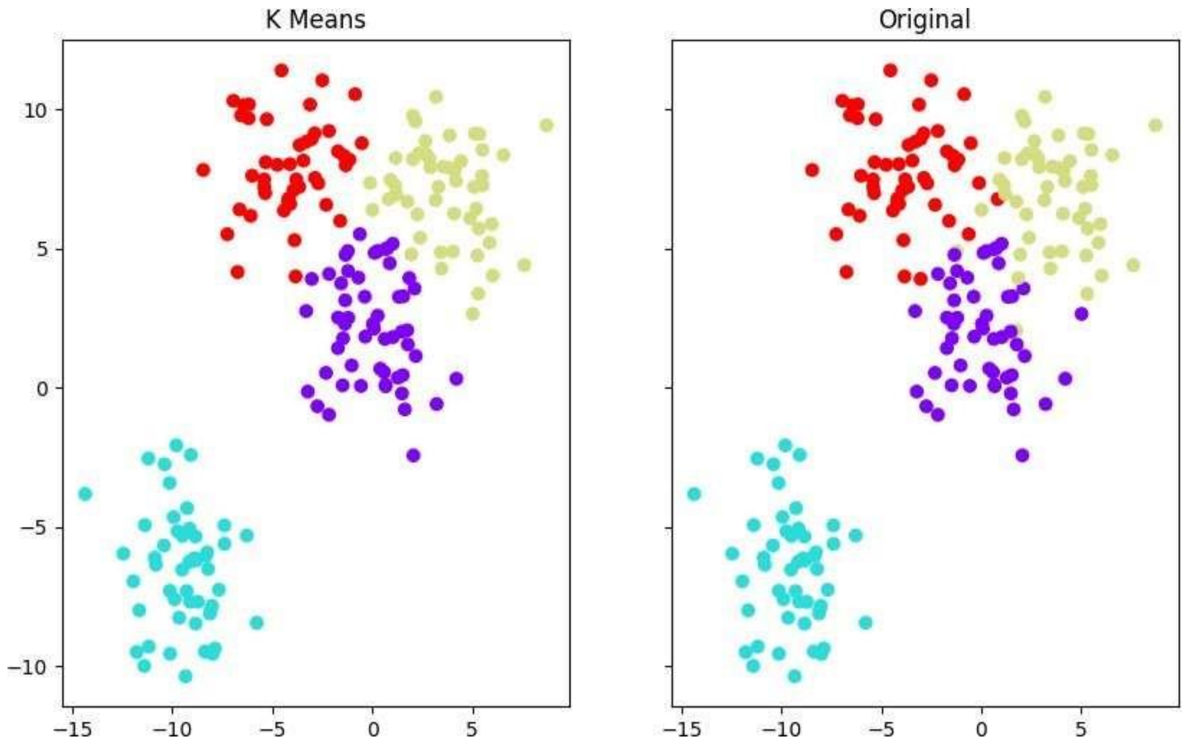
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans

from sklearn.datasets import make_blobs
data=make_blobs(n_samples=200,n_features=2,centers=4,
                               cluster_std=1.8, random_state=101)
kmeans = KMeans(n_clusters=4)
kmeans.fit(data[0])
print("K-Means Cluster Centers")
print(kmeans.cluster_centers_)

print("K-Meams Lables")
print(kmeans.labels_)
```

Output:

```
K-Means Cluster Centers
[[-0.0123077   2.13407664]
 [-9.46941837 -6.56081545]
 [ 3.71749226  7.01388735]
 [-4.13591321  7.95389851]]

K-Meams Lables
[3 2 0 2 2 1 2 0 2 0 3 0 2 2 3 0 2 0 1 3 1 0 0 1 3 1 1 0 2 2 3 1 2 0
 0 3 1 1 1 0 1 3 3 3 0 2 3 0 1 0 0 3 2 0 1 3 0 0 3 2 1 2 1 3 2 0 1 2
 2 1 2 0 1 0 1 2 2 0 3 0 0 1 2 1 0 0 0 3 0 1 1 1 1 0 0 1 2 3 1 2 0 1
 0 0 2 0 1 2 1 1 2 3 3 2 1 2 3 3 2 3 0 3 0 3 0 2 3 0 1 3 3 3 0 1 1 3
 2 3 2 0 1 2 1 3 3 2 0 1 3 3 3 3 0 2 0 3 2 2 2 0 2 0 0 3 1 3 0 2 3 0
 2 0 3 2 0 3 2 2 1 2 3 1 1 3 1 1 1 1 1 0 1 2 2 3 1 0 2 2 1 0]
```

K-Means Model Visualization:

```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels_,
                                          cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')
```



Result: Successfully implemented K-Means Clustering Model.

Experiment-10:

Aim: Performance analysis of Classification Algorithms on a specific dataset.

Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

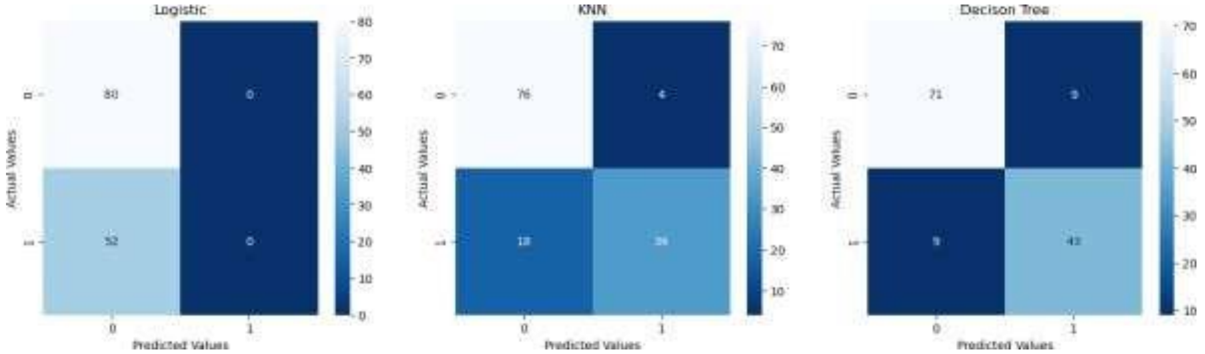
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report,
                                                        accuracy_score


df = pd.read_csv('Social_Network_Ads.csv')
print(df.columns)
X = df[['Age','EstimatedSalary']]
y = df['Purchased']
X_train,X_test,y_train,y_test= train_test_split(X, y, test_size=0.33,
                                                      random_state=42)

models=  []
models.append(LogisticRegression())
models.append(KNeighborsClassifier())
models.append(DecisionTreeClassifier())
model_list = ['Logistic','KNN','Decison Tree']
acc_list =[]
cm_list =[]

for model in models:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc_list.append(accuracy_score(y_test, y_pred))
    cm_list.append(confusion_matrix(y_test,y_pred))

result_df = pd.DataFrame({'Model':model_list,'Accuracy':acc_list})
print(result_df)
```

Output:

```
Index(['Age', 'EstimatedSalary', 'Purchased'], dtype='object')

          Model  Accuracy
0      Logistic  0.606061
1           KNN  0.833333
2  Decison Tree  0.863636
```

Confusion Matrix:

```
fig = plt.figure(figsize=(18,10))
for i in range(len(cm_list)):
    cm =  cm_list[i]
    model = model_list[i]
    sub = fig.add_subplot(2,3, i+1).set_title(model)
    cm_plot = sns.heatmap(cm, annot=True, cmap='Blues_r')
    cm_plot.set_xlabel('Predicted Values')
    cm_plot.set_ylabel('Actual Values')
```



Result: Successfully compared the performance of classification models.