QIS

COLLEGEOFENGINEEINGANDTECHNOLOGY

(AUTONOMOUS)

(Approvedby AICTE, Permanently Affiliated to JNTUK, Kakinada, Accredited by NBA & UGC Recognized)
Vengamukkapalem, PondurRoad, ONGOLE-523272,A.P.

LABRECORD

DEPARTMENT OF COMPUTERSCIENCE&ENGINEERING

SUBJECT	:	**MACHINELEARNINGLAB**
COURSE	:	**B. TECH.**
YEAR&SEM	:	**III&II**
A.Y.	:	**2024-25**

QISCOLLEGEOFENGINEERING&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 work done

By Mr./Ms.……………………………………………………………………………………………

During the Year: 2024-25

Head of the Section **Ext. Examiner**

1.

Staff in- charge **2.**

INDEX

Expt .No.	Date	Title of the Experiment	Page No.	Signature Of the Lecturer	Remarks
1		Write a python program to compute • Central Tendency Measures: Mean, Median, mode. • Measure of Dispersion: Variance, Standard Deviation.			
2		Study of Python Basic Libraries such as Statistics, Math and Numpy.			
3		Study of Python Libraries for ML application such as Pandas and Matplotlib			
4		Write a Python program to implement Simple Linear Regression			
5		Implementation of Multiple Linear Regression for House Price Prediction using sklearn			
6		Implementation of Decision tree using sklearn and its parameter tuning			
7		ImplementationofKNNusing sklearn			
8		Implementation of Logistic Regression using sklearn			
9		Implementation of K-Means Clustering			
10		Performance analysis of Classification Algorithms on a specific dataset			

EXPERIMENT-1

Central Tendencies in Statistics are the numerical values that are used to represent mid-value or central value a large collection of numerical data.

Mean:-The mean is the average value of all the values in a dataset. To calculate the mean value of a dataset, we first need to find the sum of all the values and then divide the sum of all the values by the total number of values.

Median :-The median of numeric data is the value that lies in the middle when we sort the data. The data may be sorted in ascending or descending order, the median remains the same.

Mode:-The mode is the most frequent observation (or observations) in a sample.

Measures of Dispersion: Variance Standard Deviation:-The variance is a measure of how far individual (numeric) values in a dataset are from the mean or average value. The variance is often used to quantify spread or dispersion. Spread is a characteristic of a sample or population that describes how much variability there is in it. To calculate the variance in a dataset, we first need to find the difference between each individual value and the mean.The variance is the average of the squares of those differences.

Standard Deviation:-The standard deviation measures the amount of variation or dispersion of a set of numeric values. Standard deviation is the square root of variance $\boldsymbol{\sigma^2}$

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)
```

```
#DispersionMeasures
#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('StandardDeviation',std)
```

Output:

Mean:4.0
Median:4.0
Mode:4
Variance:2.0
StandardDeviation1.41

Result: Successfully computed central tendency measures and dispersion measures.

EXPERIMENT-2

Statistics:-The statistics library provides functions for calculating mathematical statistics of numeric data. **statistics**: Provides simple statistical functions like mean, median, mode, variance, and standard deviation. import statistics as stats
Math: The math library provides access to mathematical functions **math**: Offers a wide range of mathematical functions, including trigonometric, exponential, and logarithmic functions. .import math.
Numpy:-The numpy library is used for working with arrays and provides a large set of mathematical functions to operate on these arrays, **it is** Essential for numerical operations on arrays and includes statistical functions, array manipulations, and more.
import numpy as np

Aim: Study of Python Basic Libraries suchas 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('StandardDeviation:',round(st.stdev(data),2))
```

Output:

```
Mean:3.9
Median:4.0
Mode:5
Variance:2.32
StandardDeviation:1.52
```

Math module:

```
Import math
 #constants
print("Exponential value: ", math.e)
print("Pi value: ",math.pi)
print("Infinite value: ", math.inf)
print("Notanumbervalue:",math.nan)

#methods
print("Logarithmic: ", math.log(math.e))
print("factorialof5is",math.factorial(5))
print("GCD of 64and42is",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 5is 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-Darrayis:") print(ar2d)

#Properties
print("ar1dshape,size,dimensionsare",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("Indexingonar2d:",ar2d[2][1],ar2d[1][1])

#slicing
print("Slicingonar1d:",ar1d[2:6])
print("Slicing on ar2d:")
print(ar2d[1:4,2:4])
```

Output:

```
1-D array is:
[11 12 13 1415 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]]

ar1dshape,size,dimensionsare(6,)61
ar2dshape,size,dimensionsare(5,5)252
Indexing on ar1d:1315

Indexing on ar2d:2217
```

Slicing on ar1d:[13141516]
Slicing on ar2d:
[[1819]
 [2324]
 [2829]]

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

EXPERIMENT-3

a) **Pandas Library Module:**The Pandas library is used for data manipulation and analysis. It provides data structures like Series and Data Frame, which are essential for handling structured data. It is Essential for data manipulation and analysis. Provides powerful data structures, like Series and Data Frame for handling structured data. Creating and manipulating Data Frames. Filtering and grouping data. Handling missing values.

import pandas as pd

b) **Matplotlib Library Module:-**The matplotlib library is used for creating static, animated, and interactive visualizations in Python. A versatile library for creating various types of visualizations. Creating line plots, bar plots, histograms, scatter plots, and box plots. Customizing plots with titles, labels, legends, and grid lines. These libraries are foundational tools in the machine learning workflow, enabling effective data analysis and visualization, which are crucial for understanding data patterns and insights before building machine learning models.

import matplotlib. pyplot as plt

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

Source code:

Pandasmodule:

```
importpandasaspd import
numpy as np
fromnumpyimportrandom

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

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

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

print ("Rowwise 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 frameis
U V W X
YA10110210310410
5
B106107108109110
C111112113114115
D116117118119120
E1211 22123124125
Columnwise 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

Ro wiseaccsessing
w
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

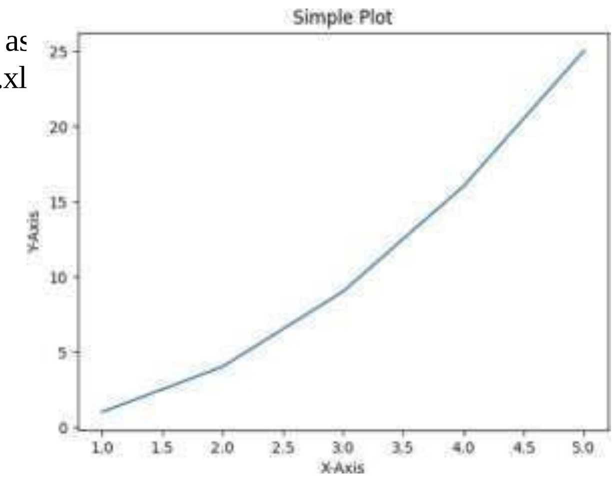
U V W X Y

C111112113114115
D116117118119120

Matplotlib module:

BasicPlot:

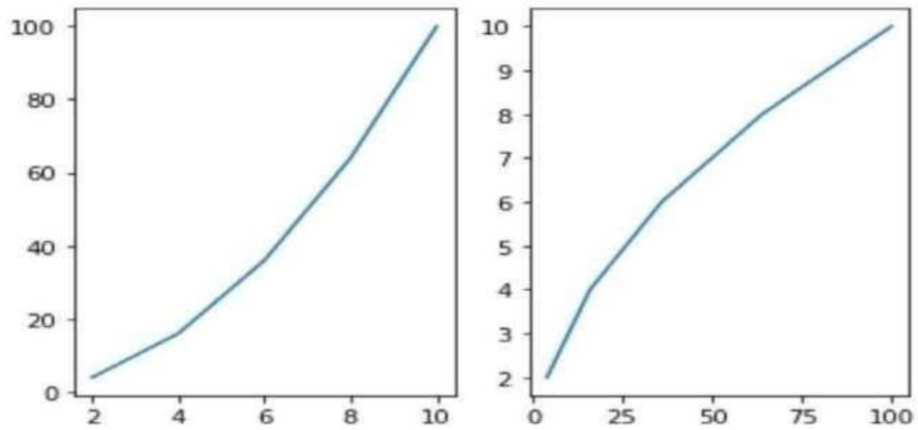
importmatplotlib.pyplotasplt import numpy as
x = np.arange(1,6)y = x**2 plt.plot(x,y) plt.xl
plt.title("SimplePlot") plt.show()



Output:

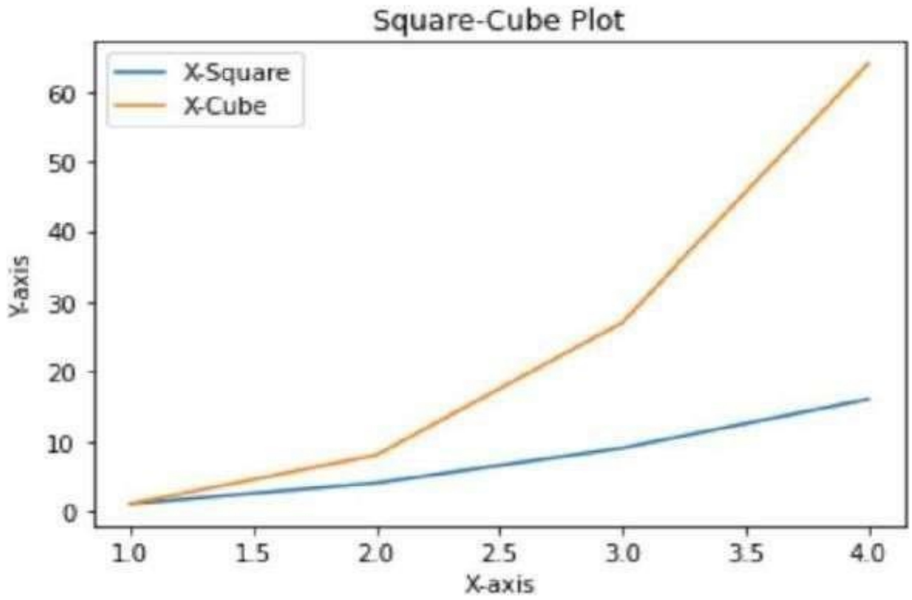
Creating subplots:

```
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()
```



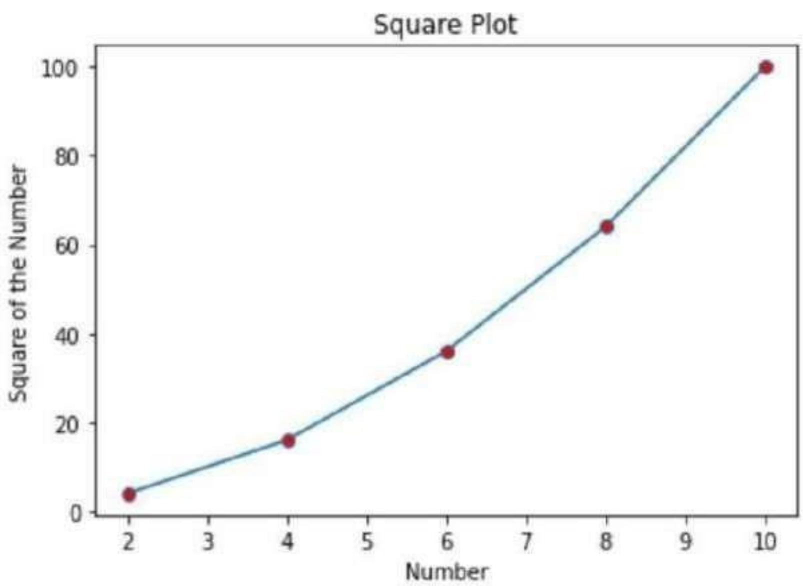
Addinglegendtoplot:

```
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:

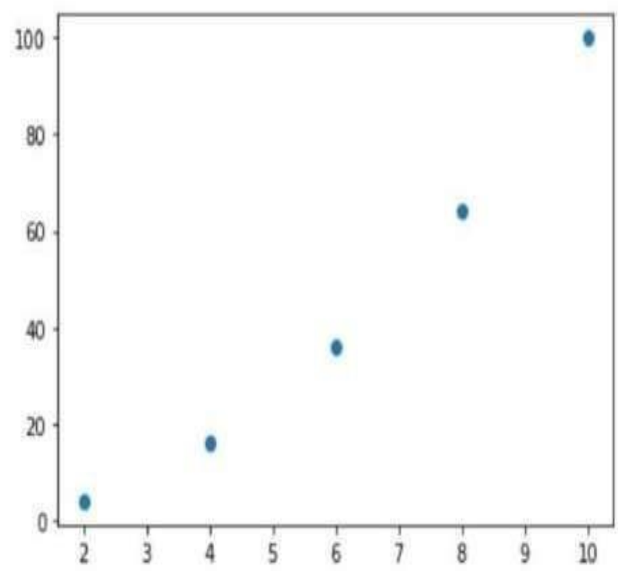
```
import matplotlib.pyplot as plt
x = [2, 4, 6, 8, 10]
y = [i**2 for i in x]
plt.plot(x,y,marker='o',markerfacecolor='red')
plt.xlabel('Number')
plt.ylabel('Square of the Number')
plt.title('Square Plot')
plt.show()
```



Different types of plots:

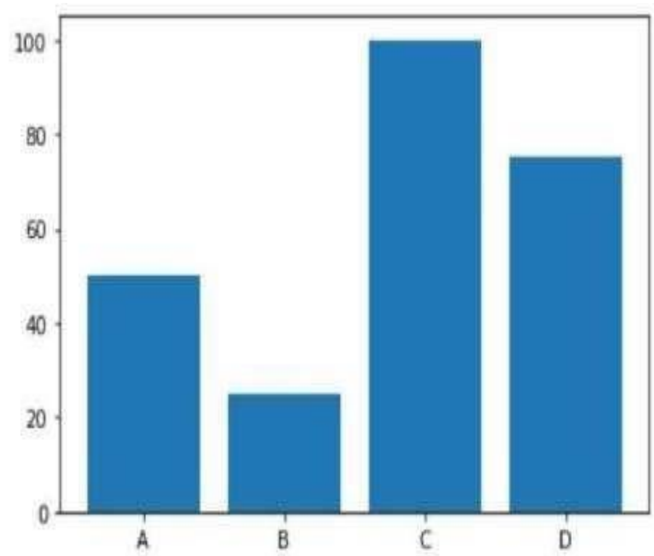
Scatter plot:

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



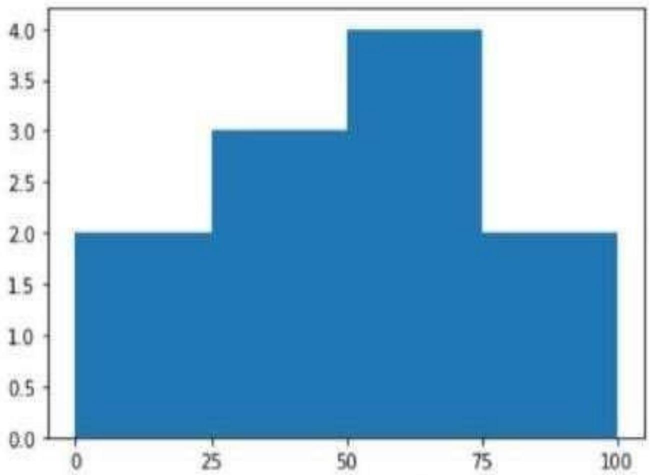
Bar plot:

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



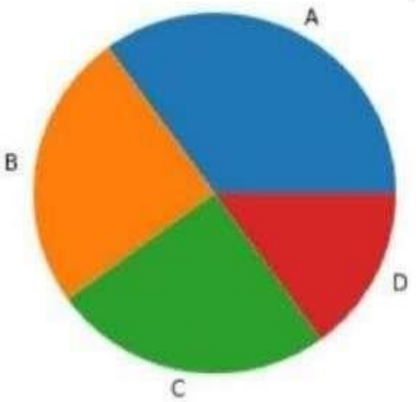
Histogram:

```
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: Successfullyworked onPandasandMatplotlib libraries.

EXPERIMENT-4

Regression is a statistical method used to understand the relationship between a dependent (target) variable and one or more independent (predictor) variables.
The goal of regression is to model the relationship between the variables so that the dependent variable can be predicted from the independent variable(s). Regression analysis is widely used for prediction, forecasting, and identifying causal relationships.
Types of Regression

1. **Simple Linear Regression**: Involves one independent variable.
2. **Multiple Linear Regression**: Involves more than one independent variable.
3. **Polynomial Regression**: Models a non-linear relationship between the dependent and independent variables.
4. **Logistic Regression**: Used for binary classification problems.
5. **Ridge, Lasso, and Elastic Net Regression**: Regularization techniques to handle multicollinearity and overfitting.

Regression: A technique for modeling the relationship between a dependent variable and one or more independent variables
Simple Linear Regression: Models the relationship between two variables using a straight line, where one variable is dependent and the other is independent.
Implementation: Can be done manually by calculating the slope and intercept using statistical methods or using libraries like scikit-learn for a more streamlined approach.

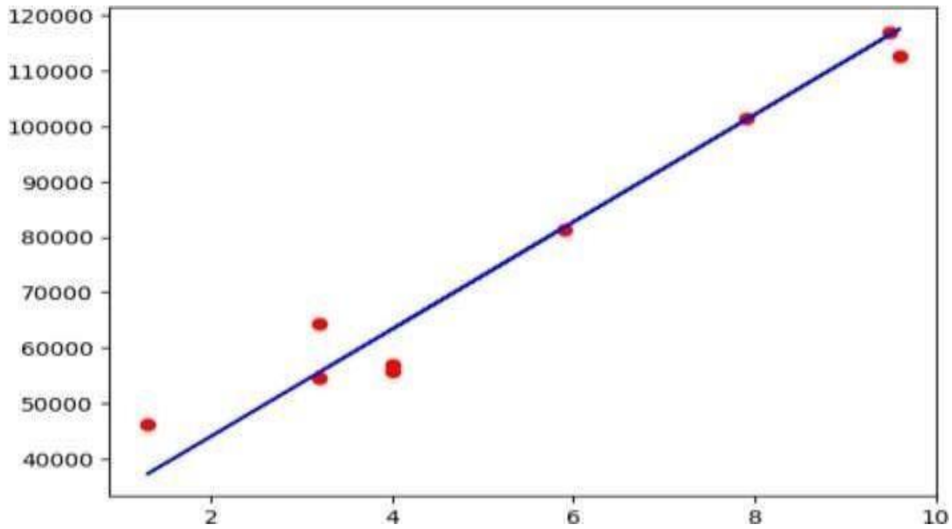
Aim: Write a Python program to implement Simple Linear Regression.
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link

Source code:

```
Import  numpy  as np
importpandasaspd
importmatplotlib.pyplotasplt
fromsklearn.model_selectionimporttrain_test_split from
sklearn.linear_model import LinearRegression from sklearn.metrics
import r2_score
#Loadingdataset
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()
#ModelTraining(Fitthemodel)
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-Sqaurevalue(Representsaccuracy):0.9524505593397691



Result: SuccessfullyimplementedSimpleLinearRegressionmodel.

EXPERIMENT-5

Multiple linear regression: It refers to a statistical technique that is used to predict the outcome of a variable based on the value of two or more variables. It is sometimes known simply as multiple regression, and it is an extension of linear regression.
The variable that we want to predict is known as the dependent variable, while the variables we use to predict the value of the dependent variable are known as independent or explanatory variables.
The following steps are involved in Multiple linear regression
Collect Data: Gather and inspect the dataset.
Explore and Clean Data: Handle missing values, outliers, and encode categorical data.
Select and Engineer Features: Choose relevant features and potentially create new ones.
Split the Data: Divide the dataset into training and testing sets.
Implement the Model: Use scikit-learn to create and fit the model.
Evaluate the Model: Assess the model's performance using metrics like MSE and R-squared.
Make Predictions: Use the model to predict house prices on new data.
(Optional) Tune and Deploy: Improve the model and deploy it for use in real-world applications.
Import Required Libraries

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

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

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

Sourcecode:

```
import numpy as np
importpandasaspd
importmatplotlib.pyplotasplt import seaborn as
sns
fromsklearn.model_selectionimporttrain_test_split from
sklearn.linear_model import LinearRegression from sklearn.metrics
import r2_score

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

#FeatureExtraction
X=house_df[['Avg.AreaIncome','Avg.AreaHouseAge','Avg.Area Number of Rooms',
          'Avg.AreaNumberofBedrooms','AreaPopulation']] y =
```

house_df['Price']

#SplittingTrainandTestdata
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size= 0.3)

#LinearRegresionModelCreation reg_model =
LinearRegression()

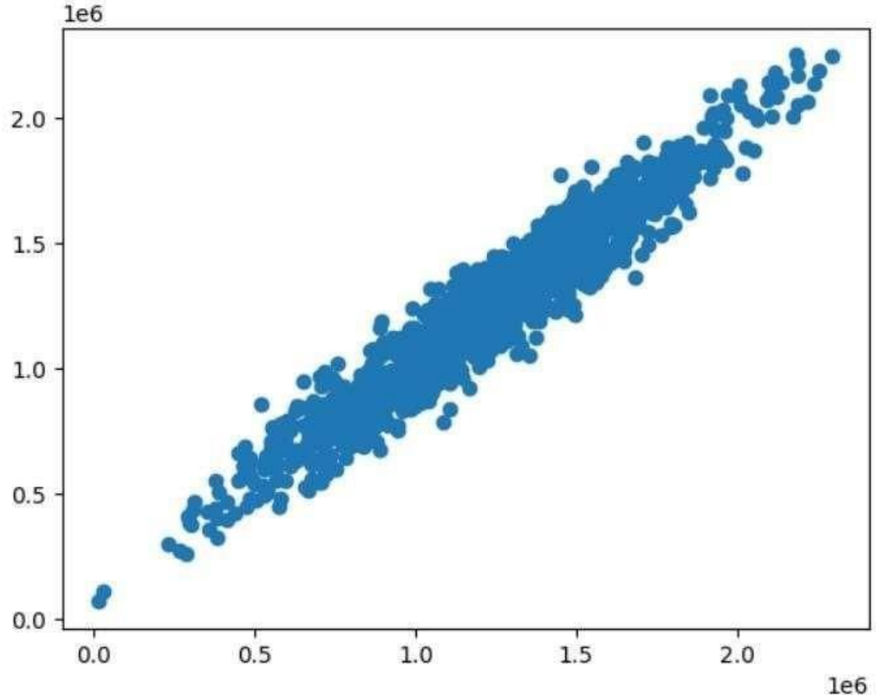
#Traing the model (fit)
reg_model.fit(X_train,y_train)

#ModelPrediction
y_pred=reg_model.predict(X_test)

#Accuracyofmodel
print("R-Sqaurevalue",r2_score(y_test,y_pred))

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

Output:
Housingdatasetcolumns:
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number ofRooms','Avg.AreaNumberofBedrooms','AreaPopulation','Price', 'Address'], dtype='object')
R-Sqaurevalue0.9237308710840247



Result: SuccessfullyimplementedMultipleLinearRegression.

EXPERIMENT-6

A decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data. It is a tree-like structure where each internal node tests on attribute, each branch corresponds to attribute value and each leaf node represents the final decision or prediction. The decision tree algorithm falls under the category of supervised learning.
They can be used to solve both **regression** and **classification problems**.
There are specialized terms associated with decision trees that denote various components and facets of the tree structure and decision-making procedure. :

- **Root Node:** A decision tree's root node, which represents the original choice or feature from which the tree branches, is the highest node.
- **Internal Nodes (Decision Nodes)**: Nodes in the tree whose choices are determined by the values of particular attributes. There are branches on these nodes that go to other nodes.
- **Leaf Nodes (Terminal Nodes)**: The branches' termini, when choices or forecasts are decided upon. There are no more branches on leaf nodes.
- **Branches (Edges)**: Links between nodes that show how decisions are made in response to particular circumstances.
- **Splitting**: The process of dividing a node into two or more sub-nodes based on a decision criterion. It involves selecting a feature and a threshold to create subsets of data.
- **Parent Node**: A node that is split into child nodes. The original node from which a split originates.
- **Child Node**: Nodes created as a result of a split from a parent node.
- **Decision Criterion**: The rule or condition used to determine how the data should be split at a decision node. It involves comparing feature values against a threshold.
- **Pruning**: The process of removing branches or nodes from a decision tree to improve its generalisation and prevent overfitting.

We have two popular attribute selection measures:

1. Information Gain
2. Gini Index

1. Information Gain:
When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.

- Suppose S is a set of instances,
- A is an attribute
- Sv is the subset of S
- *v* represents an individual value that the attribute *A* can take and Values (A) is the set of all possible values of A, then
  Gain(S,A) = Entropy(S)–∑vA|Sv||S|.Entropy(Sv)

Entropy: is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.
Suppose S is a set of instances, A is an attribute, Sv is the subset of S with A = v, and Values (A) is the set of all possible values of A, then
Gain(*S*,*A*)=*Entropy*(*S*)–∑*v*ϵ*Values*(*A*)|*S*||*Sv*|.*Entropy*(*Sv*)

<u>Aim:</u> Implementation of Decision tree using sklearn and its parameter tuning.
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL

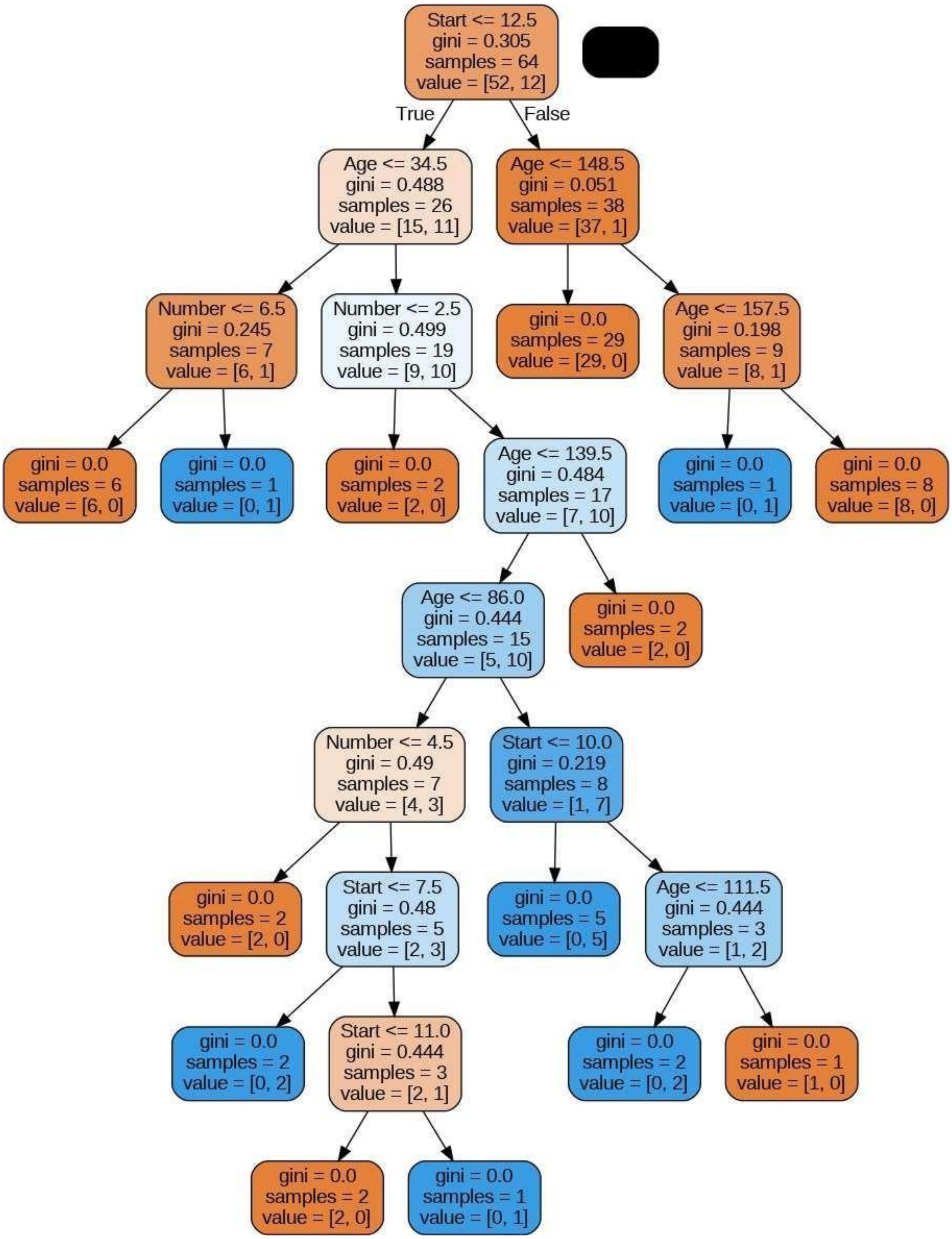
00Nebt1?usp=share_link

Sourcecode:

```
importpandasaspd import
numpy as np
importmatplotlib.pyplotasplt import seaborn as
sns
%matplotlibinline
fromsklearn.model_selectionimporttrain_test_split from sklearn.tree import
DecisionTreeClassifier
fromsklearn.metricsimportclassification_report,confusion_matrix,
                                                        accuracy_score
fromIPython.displayimportImage from six
import StringIO
fromsklearn.treeimportexport_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("AccuracyScore",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:

```
AccuracyScore0.8823529411764706 Confusion
Matrix
[[111]
 [14]]
```



Result: SuccessfullyimplementedDecisiontree model.

EXPERIMENT-7

KNN is a simplest Supervised machine Learning and non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real-world datasets do not follow mathematical theoretical assumptions. The lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and the testing phase slower and costlier.
In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm.

Aim: Implementation of KNN using sklearn.
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link
Sourcecode:

```
import pandas as pd
importseabornassns
importmatplotlib.pyplotasplt import numpy as
np
%matplotlibinline
from sklearn.preprocessing import StandardScaler
fromsklearn.model_selectionimporttrain_test_split from
sklearn.neighbors import KNeighborsClassifier
fromsklearn.metricsimportclassification_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('TARGETCLASS',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['TARGETCLASS'],test_size=0.30)

#InitiallywithK=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))

#NOWWITHK=23
knn23=KNeighborsClassifier(n_neighbors=23)

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

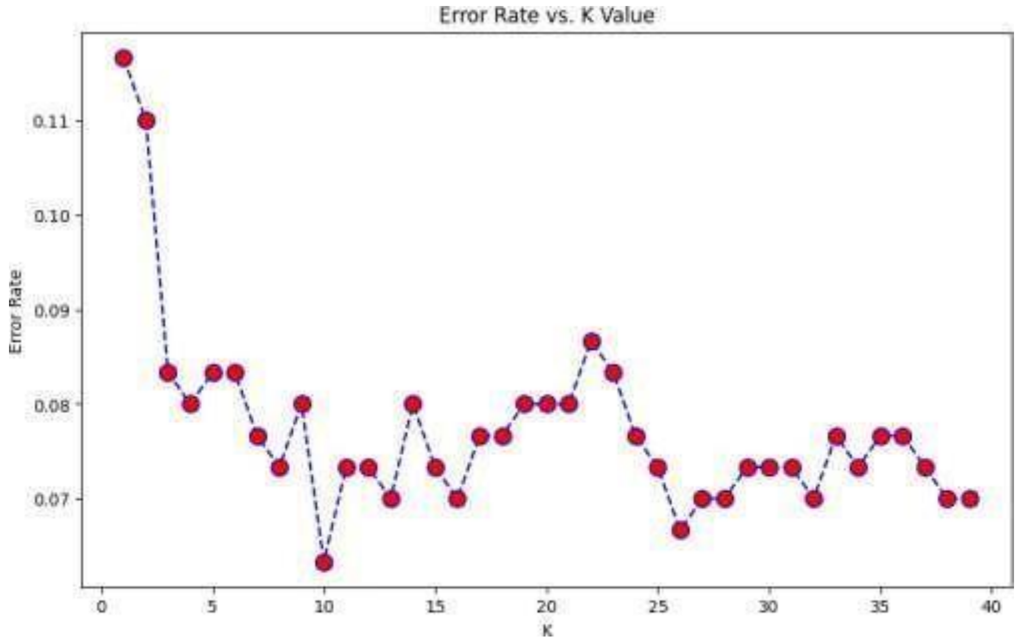
```
print("For K=23 results are:")
print(confusion_matrix(y_test,pred23))
print(classification_report(y_test,pred23))
```

Output:

```
For K=1 results are:  [[128
17]
  [18137]]

              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

   macroavg        0.88      0.88      0.88       300
 weightedavg       0.88      0.88      0.88       300
ForK=23results    are:
[[130 15]
  [10145]]
              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
    macroavg       0.92      0.92      0.92       300
 weightedavg       0.92      0.92      0.92       300
```

Choosing K Value:

```
error_rate=[]
foriinrange(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('ErrorRate')
```



Result: SuccessfullyimplementedKNNClassification model.

EXPERIMENT-8

Logistic regression is a supervised **classification** model known as the **logit** model. It estimates the **probability** of something occurring, like 'will buy' or 'will not buy,' based on a dataset of independent variables.
The outcome should be a categorical or a discrete value.
The outcome can be either a 0 and 1, true and false, yes and no, and so on.
The model does not give an exact 0 and 1 but a value between 0 and 1. Unlike linear regression, which fits a regression line, logistic regression fits an 'S'-shaped logistic function(**Sigmoid function**).
Logistic regression is a binary classification technique, the values predicted should fall close to either 0 or 1. This is why a sigmoid function is convenient.
In mathematical terms:

$$p(x) = \frac{1}{1+e^{-z}}$$

Where:
p(x) is the predicted probability that the output for a given **x** is equal to 1.
z is the linear function since logistic regression is a linear classifier which translates to:

$\mathbf{z = b_0 + b_1x_1 + ... + b_rx_r}$

Where:

$\mathbf{b_0}$, $\mathbf{b_1}$ ...$\mathbf{b_r}$ are the model's **predicted eights** or **coefficients**.
x the feature values.
To implement logistic regression with Scikit-learn, you need to follow the following steps.
Import the packages**,** classes, and functions.
Load the data.
Exploratory Data Analysis(**EDA**).
Transform the data if necessary.
Fit the classification model.
Evaluate the performance model.

Aim: **Implementation of Logistic Regression using sklearn.**
Datasetslink:https://drive.google.com/drive/folders/15XG8HzPdMaWgGYv5DGG4uN4KL00Nebt1?usp=share_link
Sourcecode:

```
Import numpy as np
importmatplotlib.pyplotasplt
import pandas as pd
import seaborn as sns
fromsklearn.model_selectionimporttrain_test_split from
sklearn.preprocessing import StandardScaler from
sklearn.linear_model import LogisticRegression
fromsklearn.metricsimportconfusion_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 =
```

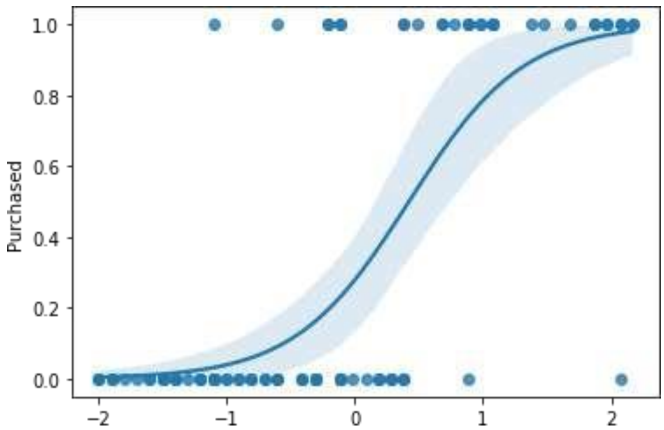
LogisticRegression()
classifier.fit(X_train,y_train)y_pred =
classifier.predict(X_test) print("Confusion Matrix")
print(confusion_matrix(y_test,y_pred))
print("AccuracyScore",accuracy_score(y_test,y_pred))

Output:

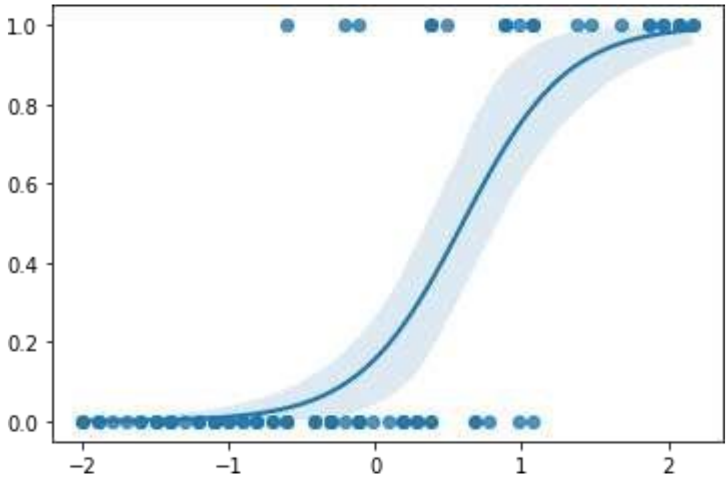
Index(['Age','EstimatedSalary','Purchased'],dtype='object')
ConfusionMatrix
[[58 8]
 [1024]]
AccuracyScore0.82

ModelVisualization:

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

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

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 allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training. 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.

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

Aim: Implementation of K-Means Clustering.

Source code:

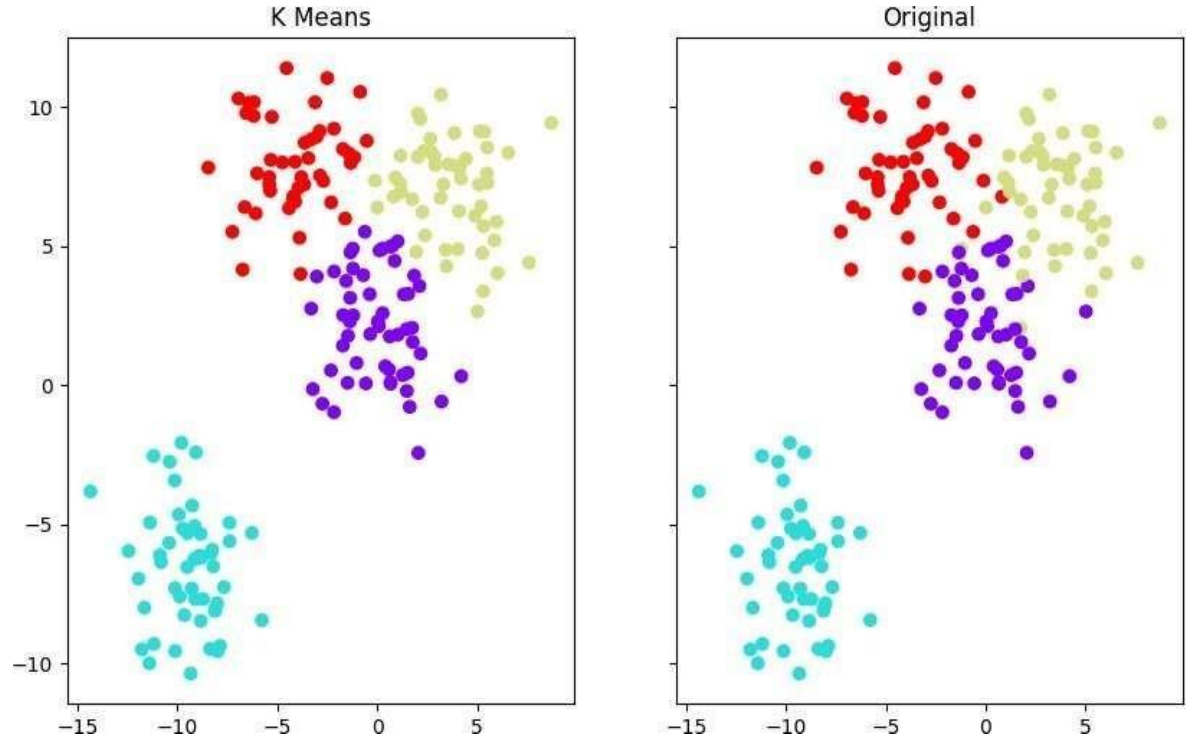
```
Import sea born as sns
Import matplotlib.pyplot as plt
%matplotlib in line
From sklearn.cluster import K-Means

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-MeansClusterCenters") print(kmeans.cluster_centers_)

print("K-MeamsLables")
print(kmeans.labels_)
```

Output:

```
K-MeansClusterCenters
[[-0.0123077      2.13407664]
 [-9.46941837    -6.56081545]
 [3.71749226      7.01388735]
 [-4.13591321     7.95389851]]

K-MeamsLables
[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   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   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   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   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   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-MeansModelVisualization:

```
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

Analyzing the performance of classification algorithms on clinical datasets involves steps such as preprocessing the data, choosing appropriate algorithms, training the models, evaluating them, and visualizing performance metrics. Here's an outline for Python code to carry out these steps using a clinical dataset, with common algorithms like Logistic Regression, Decision Trees, Random Forest, Support Vector Machines, and k-Nearest Neighbors.

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

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

Sourcecode:

```
Import pandas as pd
import numpy as npb
importmatplotlib.pyplot as plt
import seaborn as sns
fromsklearn.model_selectionimporttrain_test_split from sklearn import
preprocessing
fromsklearn.linear_modelimportLogisticRegression from
sklearn.neighbors import KNeighborsClassifier from sklearn.tree
import DecisionTreeClassifier from sklearn import metrics
                    fromsklearn.metricsimportconfusion_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','DecisonTree'] 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)
```

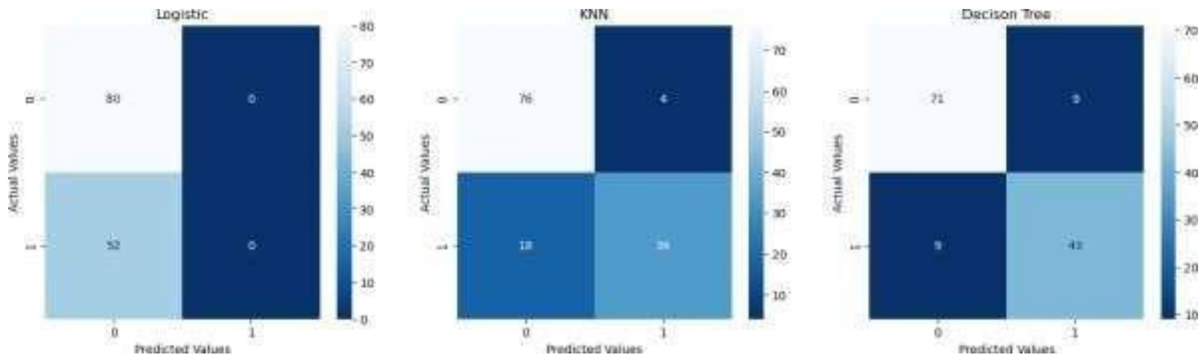
Output:

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

	ModelAccuracy
0	Logistic0.606061
1	KNN0.833333
2	DecisonTree0.863636

Confusion Matrix:

```
fig=plt.figure(figsize=(18,10))
for i in range(len(cm_list)):
    cm =
    cm_list[i]model=mode
    l_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: Successfullycomparedtheperformanceofclassification models.