Experiment-1

AIM: Identification and Installation of python environment towards the machine learning, installing python modules/Packages Import scikitlearn, keras and tensorflows etc.

DESCRIPTION:

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

DOWNLOADING PYTHON

Step 1: Go to https://www.python.org/

The following page would be visible

Step 2: Click the **Download Python 3.9.6** button. The following pop-up window titled **Opening python-3.96-amd64.exe** will appear.

Click the **Save File** button.

The file named **python-3.9.6-amd64.exe** should start downloading into your standard download folder.

Step 3: Move this file to a more permanent location, so that you can install Python (and reinstall it easily later, if necessary).

Step 4: Feel free to explore this webpage further; if you want to just continue the installation, you can terminate the tab browsing this webpage.

Step 5: Start the **Installing** instructions directly below.

Installing

1. Double-click the icon labeling the file **python-3.9.6-amd64.exe**.

A **Python 3.9.6 (64-bit) Setup** pop-up window will appear.

Ensure that **both** the **Install launcher for all users (recommended)** and the **Add Python 3.9 to PATH** checkboxes at the bottom are checked: typically only first is checked by default.

If the Python Installer finds an earlier version of Python installed on your computer, the **Install Now** message may instead appear as **Upgrade Now** (and the checkboxes will not appear).

2. Highlight the **Install Now** (or **Upgrade Now**) message, and then click it.

When run, a **User Account Control** pop-up window may appear on your screen. I could not capture its image, but it asks, **Do you want to allow this app to make changes to your device**.

3. Click the **Yes** button.

A new **Python 3.9.6 (64-bit) Setup** pop-up window will appear with a **Setup Progress** message and a progress bar.

During installation, it will show the various components it is installing and move the progress bar towards completion. Soon, a new **Python 3.9.6 (64-bit) Setup** pop-up window will appear with a **Setup was successfuly** message.

4. Click the **Close** button.

Python should now be installed.

INSTALLING SCIKIT-LEARN

1. Go to the command prompt
2. Type pip install scikit-learn
3. The package will be downloaded

INSTALLING KERAS

1. Go to the command prompt
2. Install numpy, pandas, matplotlib, scikit learn before installing keras
3. Type pip install keras
4. The package will be downloaded

INSTALLING TENSORFLOW

1. Go to the command prompt
2. Type pip install tensorflow
3. The package will be downloaded

IMPORTING SCIKITLEARN

Command – import sklearn

IMPORTING KERAS

Command – import keras

IMPORTING TENSORFLOW

Command – import tensorflow

CONCLUSION:

Successfully installed modules/packages needed for machine learning.

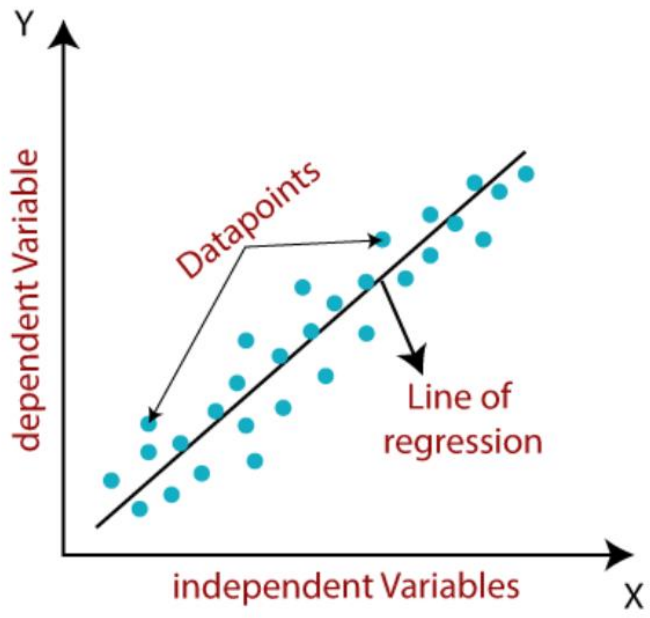
Experiment-2

AIM: Build linear regression model using gradient descent, least squares, polynomial, LASSO and RIDGE approaches also compare all the algorithms and draw a table for all the metrics.

DESCRIPTION:

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Gradient Descent:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

Least Square Method :

The least squares method is a common technique used in machine learning to fit a linear regression model to a given dataset. The goal is to find the line that best fits the data by minimizing the sum of the squared differences between the predicted values and the actual values.

The basic idea is to find the slope and intercept of the line that minimize the sum of the squared errors (the difference between the predicted and actual values). This can be done using a mathematical formula, which involves taking the derivative of the sum of squared errors with respect to the parameters (slope and intercept) and setting them to zero.

Once the optimal values of the parameters have been found, they can be used to make predictions for new input values.

The least squares method assumes that the errors in the data are normally distributed and that the variance of the errors is constant across all values of the input variable. It is also sensitive to outliers, which can significantly affect the fitted line.

Lasso Method:

The Lasso method is a technique used in linear regression to reduce the complexity of the model and prevent overfitting. It is a type of regularization that adds a penalty term to the cost function of the linear regression model, which helps in shrinking the coefficient values of some features to zero. This, in turn, helps in feature selection and simplifying the model.

The Lasso method involves adding a penalty term proportional to the absolute value of the coefficients to the cost function. This penalty term forces the model to shrink the coefficients towards zero and select only the most relevant features. The amount of shrinkage is controlled by the regularization parameter, which can be tuned using cross-validation techniques.

The Lasso method is particularly useful when dealing with high-dimensional data, where the number of features is much larger than the number of observations. It has been widely used in applications such as gene expression analysis, image processing, and natural language processing, among others.

Ridge regression:

Ridge regression is a regularization technique that is commonly used in linear regression to prevent overfitting. It is also known as L2 regularization because it adds a penalty term to the sum of squared residuals that is proportional to the square of the magnitude of the coefficients.

The Ridge regression method works by adding a penalty term to the cost function, which shrinks the coefficients towards zero. The amount of shrinkage is controlled by a hyperparameter called lambda (λ). Higher values of λ result in more shrinkage and lower values result in less shrinkage. The goal is to find the optimal value of λ that minimizes the sum of squared residuals while also preventing overfitting.

The ridge regression algorithm is similar to the standard linear regression algorithm, but with an additional penalty term added to the cost function. The resulting optimization problem can be solved using various methods, such as gradient descent or closed-form solutions.

Ridge regression can be used when there are many features in the dataset, or when the features are highly correlated with each other. It helps to reduce the variance in the model and improve its generalization performance on unseen data.

CODE & OUTPUT:

Importing the libraries

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Salary_Data.csv')

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

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

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
```

Training the Simple Linear Regression model on the Training set

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```

Predicting the Test set results

```
y_pred = regressor.predict(X_test)
```

Visualising the Training set results

```
plt.scatter(X_train, y_train, color = 'red')
```

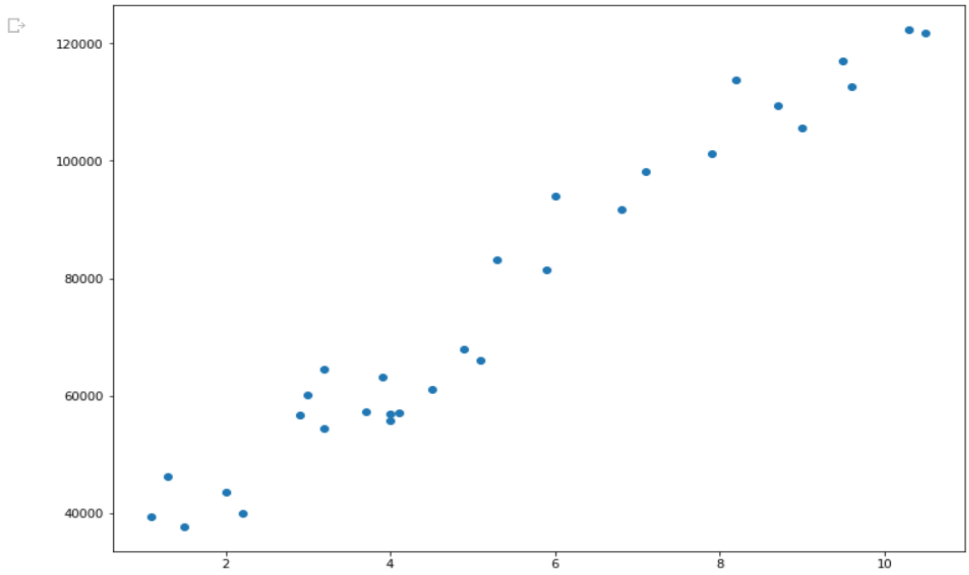
```
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Visualising the Test set results

```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Implementing the model

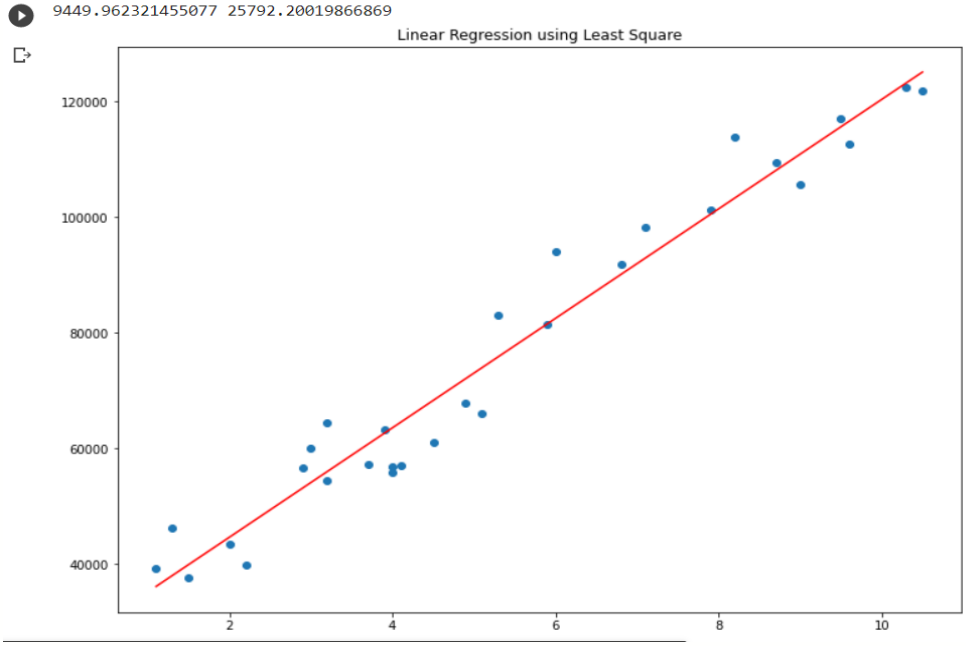
```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#rcParams => You can dynamically change the default rc (runtime configuration)
#settings in a python script or interactively from the python shell.
#All rc settings are stored in a dictionary-like variable called matplotlib.rcParams,
#which is global to the matplotlib package. See matplotlib.rcParams for a full list of configur
able rcParams.

plt.rcParams['figure.figsize']=(12.0,9.0)

#preprocessing the input data
data=pd.read_csv('data.csv')
#The iloc() function in python is defined in the Pandas module
#that helps us to select a specific row or column from the data set.
# Using the iloc method in python, we can easily retrieve any particular
#value from a row or column by using index values.
X=data.iloc[:,0]
Y=data.iloc[:,1]
plt.scatter(X,Y)
plt.show()
```



Linear Regression using Least Squares Method

```
#building the model
X_mean = np.mean(X)
Y_mean = np.mean(Y)

num=0
den=0
for i in range(len(X)):
  num+=(X[i]-X_mean)*(Y[i]-Y_mean)
  den+=(X[i]-X_mean)**2
m=num/den
c=Y_mean - m*X_mean

print(m,c)

#Making predictions
Y_pred = m*X+c
plt.scatter(X,Y)
plt.title('Linear Regression using Least Square')
plt.plot([min(X),max(X)],[min(Y_pred),max(Y_pred)],color="red")
plt.show()
```

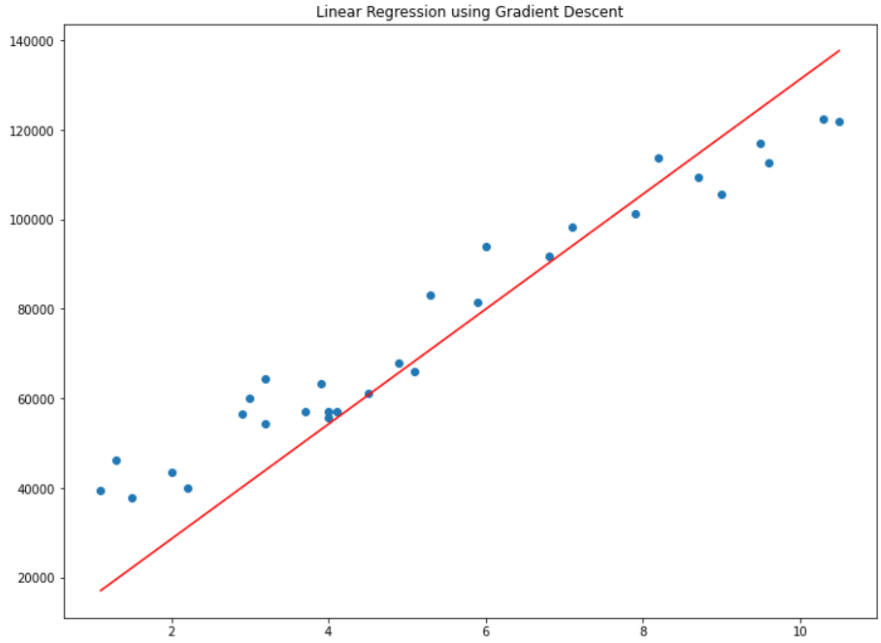
```
9449.962321455077 25792.20019866869
```



Linear Regression using Gradient Descent Method

```
#building the model

m=0 #slope
c=0 #intercept

L=0.0001 #learning rate
epochs=1000 #the number iterations performed gradient descent

n=float(len(X)) # no.of elements in X

#performing gradient descent
for i in range(epochs):
  Y_pred=m*X+c #The current predicted value  of Y
  D_m=(-2/n)* sum(X*(Y-Y_pred)) #  Derivative w.r.t m
  D_c=(-2/n)* sum(Y-Y_pred) #Derivative w.r.t c
  m=m-L * D_m # update m
  c=c-L * D_c # update  c
print(m,c)
#Making predictions
Y_pred=m*X+c
plt.scatter(X,Y)
plt.title('Linear Regression using Gradient Descent')
plt.plot([min(X),max(X)],[min(Y_pred),max(Y_pred)],color="red")
plt.show()
```

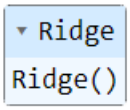
```
12836.600965885045 2915.2044856014018
```



Ridge Regression

```
# import necessary libraries

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


dataset=pd.read_csv('Salary_Data.csv')
x=dataset.iloc[:,:-1]
y=dataset.iloc[:,-1]

# split the data 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=0.25,random_state=0)

# standardize the features

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# create a ridge regression object

from sklearn.linear_model import Ridge
las=Ridge(alpha=1.0)
```

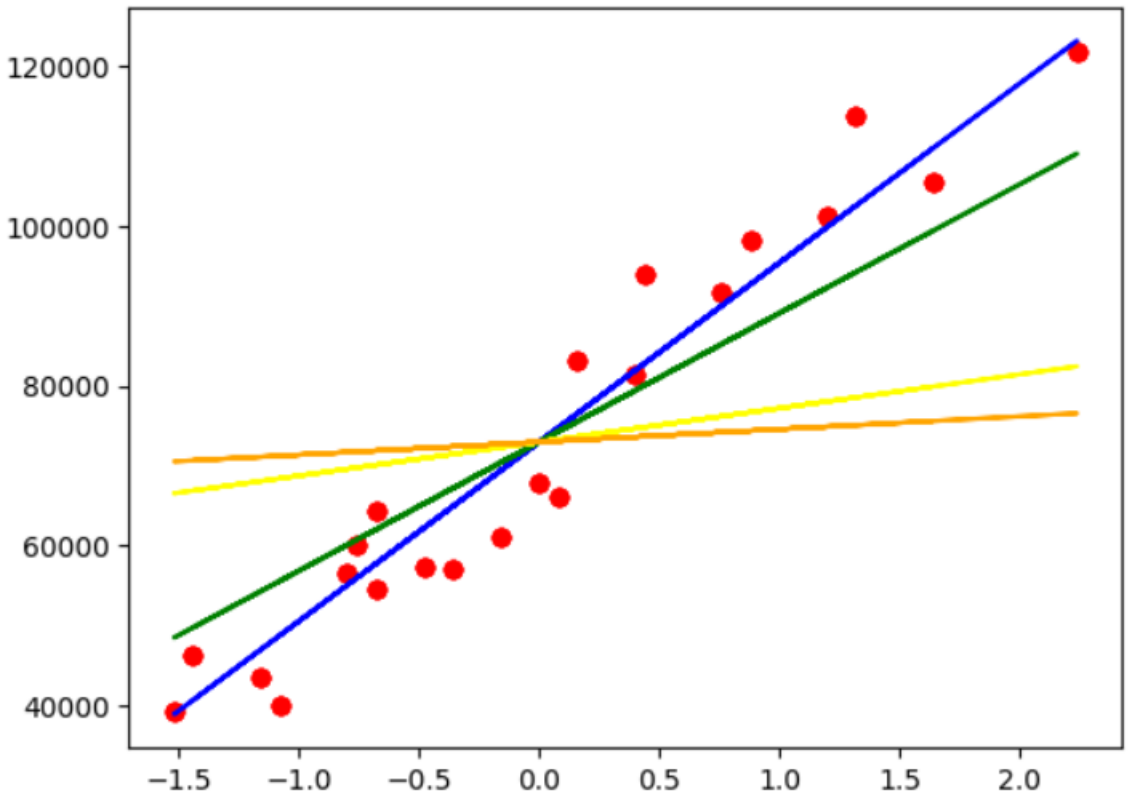
```
# fit the model on the training data

las.fit(x_train,y_train)
```




```
# predict the target values for the test data

y_pred=las.predict(x_test)
```

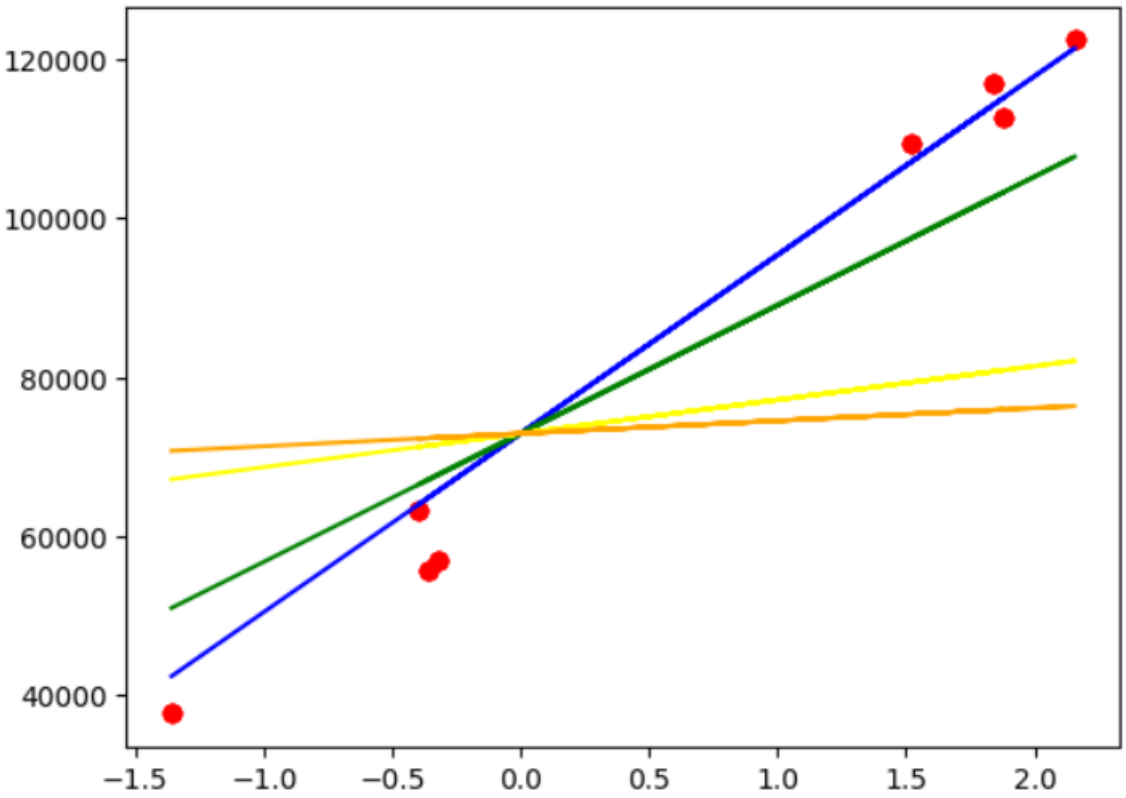
Visualizing training data

```
las=Ridge(alpha=1.0)
las.fit(x_train,y_train)
plt.scatter(x_train,y_train,color="red")
plt.plot(x_train,las.predict(x_train),color="blue")
las=Ridge(alpha=10)
las.fit(x_train,y_train)
plt.scatter(x_train,y_train,color="red")
plt.plot(x_train,las.predict(x_train),color="green")
las=Ridge(alpha=100)
las.fit(x_train,y_train)
plt.scatter(x_train,y_train,color="red")
plt.plot(x_train,las.predict(x_train),color="yellow")
las=Ridge(alpha=300)
las.fit(x_train,y_train)
plt.scatter(x_train,y_train,color="red")
plt.plot(x_train,las.predict(x_train),color="orange")
```



Visualizing testing data

```
las=Ridge(alpha=1.0)
las.fit(x_train,y_train)
plt.scatter(x_test,y_test,color="red")
plt.plot(x_test,las.predict(x_test),color="blue")
las=Ridge(alpha=10)
las.fit(x_train,y_train)
plt.scatter(x_test,y_test,color="red")
plt.plot(x_test,las.predict(x_test),color="green")
las=Ridge(alpha=100)
las.fit(x_train,y_train)
plt.scatter(x_test,y_test,color="red")
plt.plot(x_test,las.predict(x_test),color="yellow")
las=Ridge(alpha=300)
las.fit(x_train,y_train)
plt.scatter(x_test,y_test,color="red")
plt.plot(x_test,las.predict(x_test),color="orange")
```



Lasso Regression

```
# import necessary libraries

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


dataset=pd.read_csv('Salary_Data.csv')
x=dataset.iloc[:,:-1]
y=dataset.iloc[:,-1]


# split the data 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=0.25,random_state=0)
# standardize the features

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
# create a Lasso regression object

from sklearn.linear_model import Lasso
las=Lasso(alpha=0.1)

# fit the model on the training data

las.fit(x_train,y_train)
```

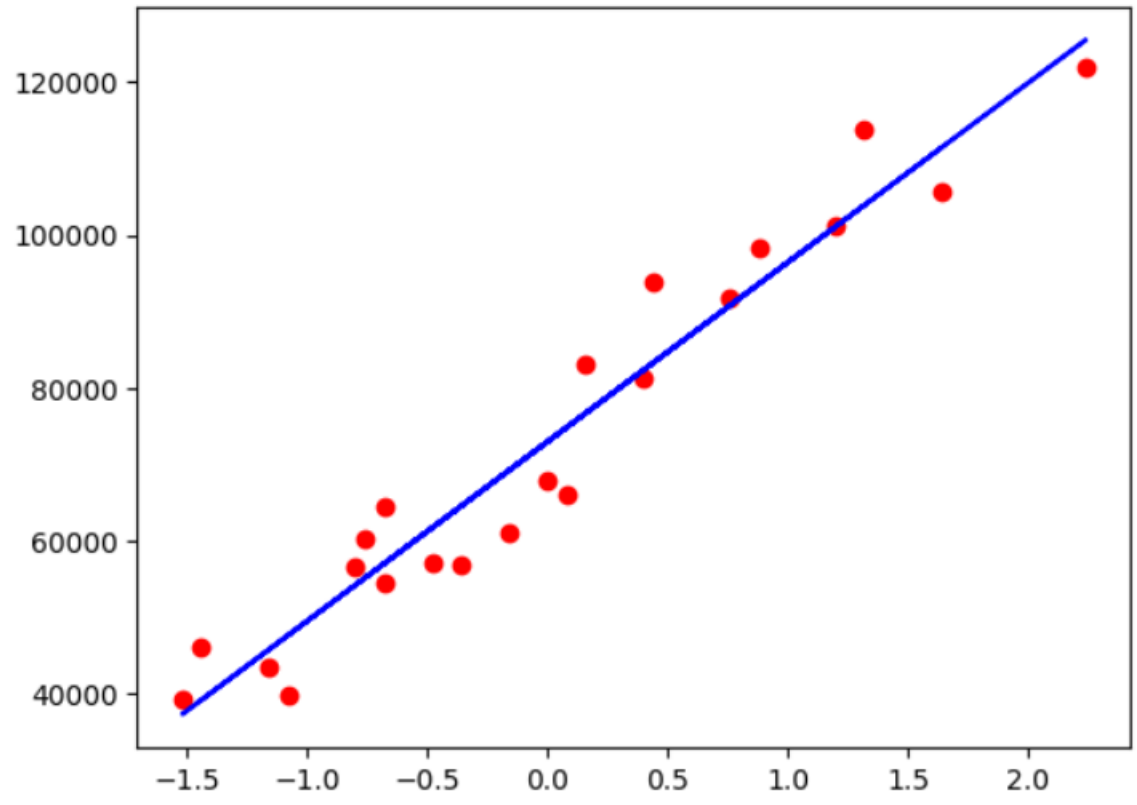
```
Lasso
Lasso(alpha=0.1)
```

```
# predict the target values for the test data

y_pred=las.predict(x_test)
# print the R-squared score of the model

print("R-squared score: {:.2f}".format(las.score(x_test, y_test)))
```

```
R-squared score: 0.98
```

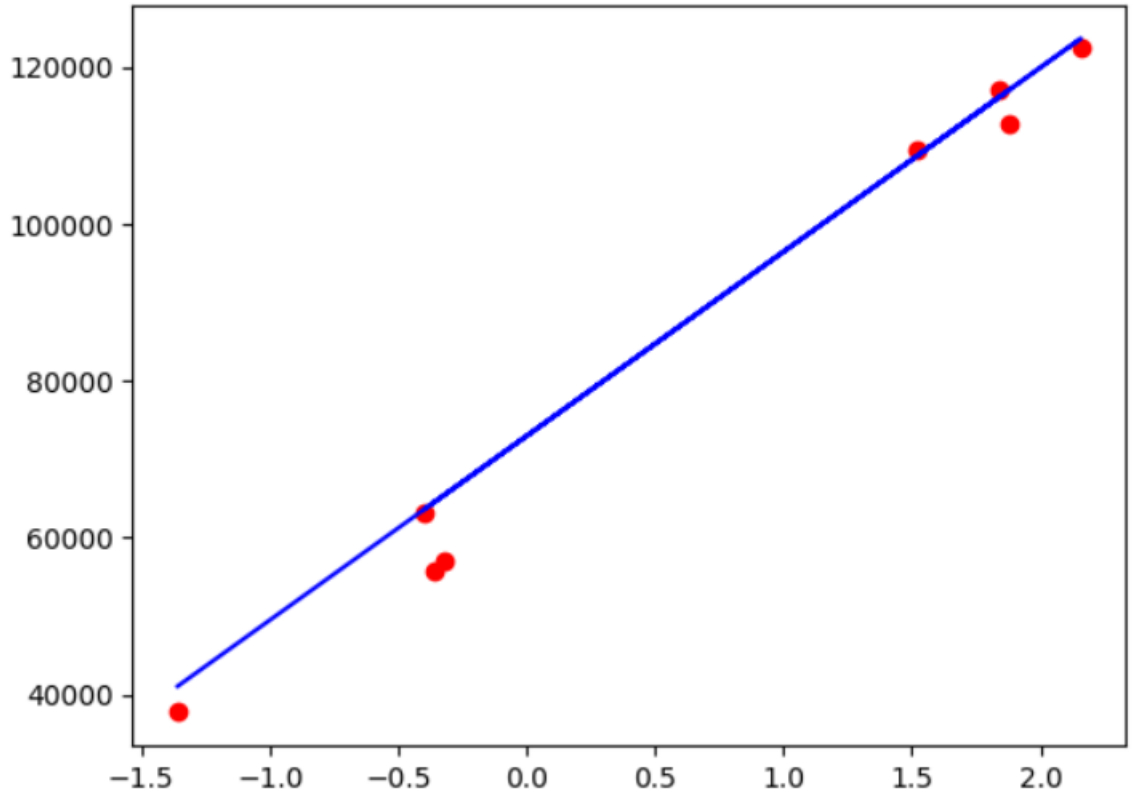
Visualizing Training data

```
plt.scatter(x_train,y_train,color="red")
plt.plot(x_train,las.predict(x_train),color="blue")
```



Visualizing Testing data

```
plt.scatter(x_test,y_test,color="red")
plt.plot(x_test,las.predict(x_test),color="blue")
```



CONCLUSION:

1.Illustrated linear regression model using gradient descent, least squares, polynomial, LASSO and RIDGE approaches.

2.Visualised the regression lines obtained.

Experiment-3

AIM: To implement the Decision Tree algorithm for a given data set and visualize the output.

DESCRIPTION:

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

CODE & OUTPUT:

Implementing the model

```
#importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into training set and test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

Training the decision tree classification model on the training set

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Making the confusion matrix

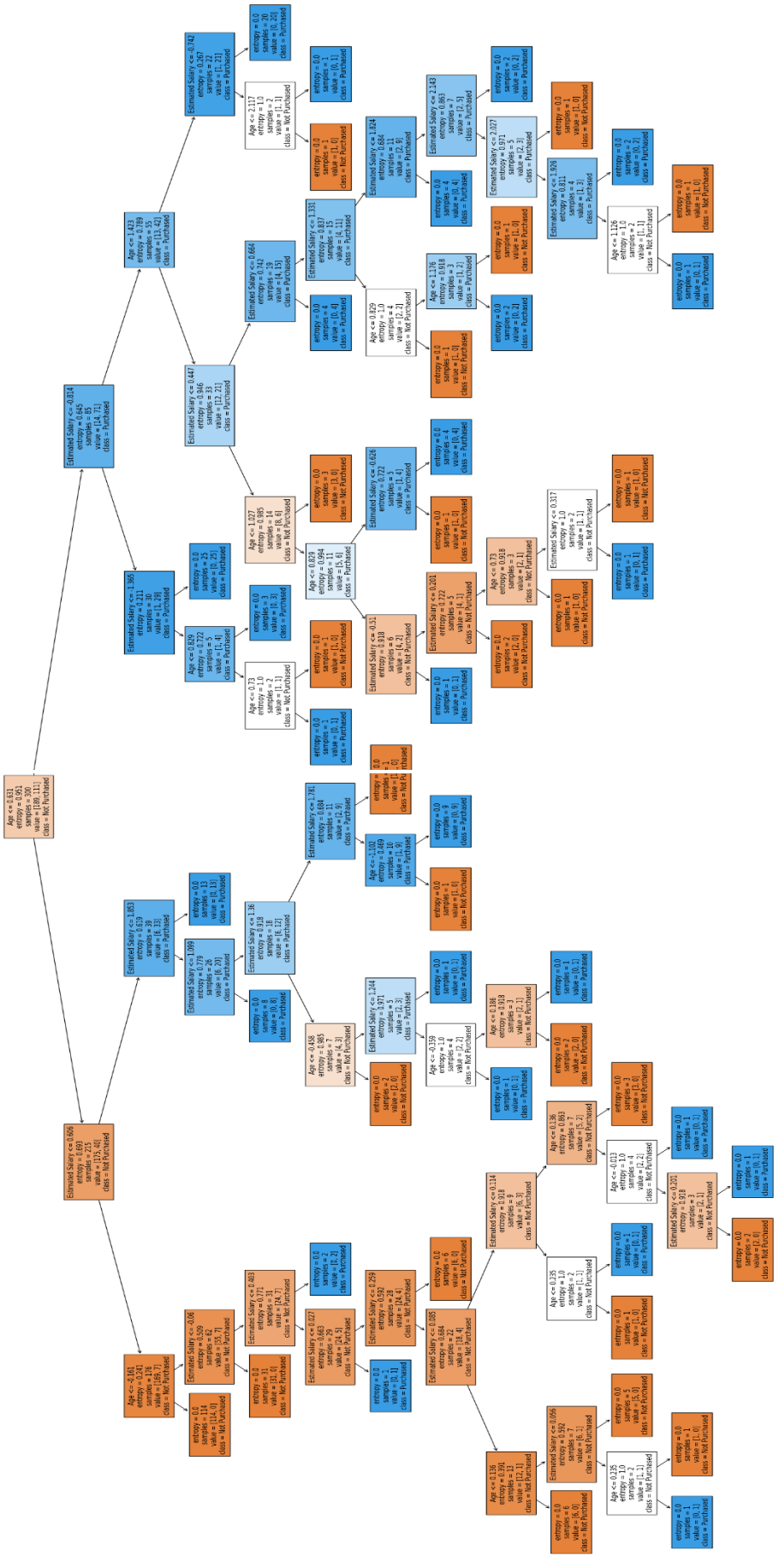
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
# Finding precision and recall
from sklearn.metrics import precision_score, recall_score
c=precision_score(y_test, y_pred)
d=recall_score(y_test, y_pred)
```

```
print(c,end="\n")
print(d)
```

```
[[62  6]
 [ 3 29]]
Accuracy = 0.91
Precision = 0.8285714285714286
Recall = 0.90625
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt


# Visualizing the decision tree for training set
plt.figure(figsize=(50,50))
plot_tree(classifier, filled=True, feature_names=['Age', 'Estimated Salary'], class_names=['No
t Purchased', 'Purchased'])
plt.show()
```


Age <= 0.181
entropy = 0.951
samples = 300
value = [180, 111]
class = Not Purchased
Estimated Salary <= 0.605
entropy = 0.969
samples = 215
value = [120, 40]
class = Not Purchased
Estimated Salary <= -0.814
entropy = 0.645
samples = 85
value = [4, 71]
class = Purchased
Age <= 0.181
entropy = 0.207
samples = 116
value = [169, 7]
class = Not Purchased
Estimated Salary <= -0.06
entropy = 0.309
samples = 62
value = [55, 7]
class = Not Purchased
entropy = 0.0
samples = 114
value = [14, 0]
class = Not Purchased
entropy = 0.0
samples = 11
value = [11, 0]
class = Not Purchased
Estimated Salary <= 0.403
entropy = 0.772
samples = 31
value = [24, 7]
class = Not Purchased
Estimated Salary <= 0.259
entropy = 0.592
samples = 29
value = [24, 4]
class = Not Purchased
entropy = 0.0
samples = 2
value = [0, 2]
class = Purchased
Estimated Salary <= 0.027
entropy = 0.65
samples = 27
value = [24, 5]
class = Not Purchased
entropy = 0.0
samples = 3
value = [0, 3]
class = Purchased
Estimated Salary <= 0.853
entropy = 0.503
samples = 39
value = [6, 33]
class = Purchased
Estimated Salary <= 1.099
entropy = 0.811
samples = 26
value = [6, 20]
class = Purchased
entropy = 0.0
samples = 13
value = [0, 13]
class = Purchased
entropy = 0.0
samples = 8
value = [0, 8]
class = Purchased
Estimated Salary <= 1.36
entropy = 0.971
samples = 18
value = [6, 12]
class = Purchased
Age <= 0.438
entropy = 0.845
samples = 7
value = [4, 3]
class = Not Purchased
entropy = 0.0
samples = 2
value = [0, 2]
class = Purchased
Estimated Salary <= 1.244
entropy = 0.971
samples = 5
value = [2, 3]
class = Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Age <= 0.159
entropy = 1.0
samples = 4
value = [2, 2]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Estimated Salary <= 1.181
entropy = 0.684
samples = 11
value = [2, 9]
class = Purchased
Age <= 1.107
entropy = 1.469
samples = 4
value = [1, 3]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 3
value = [0, 3]
class = Purchased
entropy = 0.0
samples = 7
value = [0, 7]
class = Purchased
Age <= 0.225
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Estimated Salary <= 0.665
entropy = 0.918
samples = 3
value = [2, 1]
class = Not Purchased
Age <= 0.188
entropy = 0.918
samples = 3
value = [1, 2]
class = Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
entropy = 0.0
samples = 2
value = [2, 0]
class = Not Purchased
Estimated Salary <= 0.114
entropy = 0.918
samples = 3
value = [2, 1]
class = Not Purchased
Age <= 0.136
entropy = 0.863
samples = 2
value = [1, 2]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Age <= 0.235
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
Age <= 0.013
entropy = 1.0
samples = 4
value = [2, 2]
class = Not Purchased
entropy = 0.0
samples = 3
value = [3, 0]
class = Not Purchased
Estimated Salary <= 0.201
entropy = 0.918
samples = 2
value = [2, 1]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
entropy = 0.0
samples = 2
value = [2, 0]
class = Not Purchased
Age <= 0.136
entropy = 0.391
samples = 7
value = [1, 6]
class = Purchased
Estimated Salary <= 0.055
entropy = 0.592
samples = 3
value = [1, 2]
class = Purchased
entropy = 0.0
samples = 6
value = [0, 6]
class = Not Purchased
Age <= 0.235
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 5
value = [5, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Age <= 1.423
entropy = 0.759
samples = 55
value = [13, 42]
class = Purchased
Estimated Salary <= 0.447
entropy = 0.946
samples = 33
value = [12, 21]
class = Purchased
Estimated Salary <= 1.385
entropy = 0.811
samples = 30
value = [1, 29]
class = Purchased
Age <= 0.829
entropy = 1.522
samples = 3
value = [1, 4]
class = Purchased
entropy = 0.0
samples = 25
value = [0, 25]
class = Purchased
Age <= 0.73
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 3
value = [0, 3]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Age <= 1.227
entropy = 0.985
samples = 14
value = [8, 6]
class = Not Purchased
Age <= 0.829
entropy = 0.994
samples = 11
value = [5, 6]
class = Purchased
entropy = 0.0
samples = 3
value = [3, 0]
class = Not Purchased
Estimated Salary <= 0.626
entropy = 0.722
samples = 8
value = [1, 4]
class = Purchased
Estimated Salary <= 0.51
entropy = 0.918
samples = 3
value = [4, 2]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
Estimated Salary <= 0.201
entropy = 1.522
samples = 5
value = [4, 1]
class = Not Purchased
entropy = 0.0
samples = 4
value = [0, 4]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
Age <= 0.73
entropy = 0.918
samples = 3
value = [2, 1]
class = Not Purchased
Estimated Salary <= 0.317
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
entropy = 0.0
samples = 2
value = [2, 0]
class = Not Purchased
Estimated Salary <= 0.742
entropy = 0.297
samples = 22
value = [1, 21]
class = Purchased
Age <= 2.117
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
entropy = 0.0
samples = 20
value = [0, 20]
class = Purchased
Estimated Salary <= 0.664
entropy = 0.439
samples = 13
value = [1, 12]
class = Purchased
Estimated Salary <= 1.331
entropy = 1.037
samples = 5
value = [1, 4]
class = Purchased
entropy = 0.0
samples = 4
value = [0, 4]
class = Purchased
Age <= 0.829
entropy = 1.0
samples = 4
value = [2, 2]
class = Not Purchased
Estimated Salary <= 1.824
entropy = 0.684
samples = 3
value = [2, 9]
class = Purchased
Age <= 1.116
entropy = 0.918
samples = 3
value = [1, 2]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 2
value = [0, 2]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
Estimated Salary <= 2.243
entropy = 0.863
samples = 4
value = [2, 5]
class = Purchased
Estimated Salary <= 2.027
entropy = 0.971
samples = 3
value = [2, 3]
class = Purchased
Estimated Salary <= 1.926
entropy = 0.811
samples = 2
value = [1, 3]
class = Purchased
entropy = 0.0
samples = 1
value = [0, 2]
class = Purchased
Age <= 1.126
entropy = 1.0
samples = 2
value = [1, 1]
class = Not Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased
entropy = 0.0
samples = 1
value = [1, 0]
class = Not Purchased
entropy = 0.0
samples = 1
value = [0, 1]
class = Purchased

CONCLUSION:

1.Illustrated Decision Tree algorithm and visualised the output.

2.The accuracy, precision and recall were found to be 0.91,0.83 and 0.9 respectively.

Experiment-4

AIM: To implement the K-Nearest Neighbours algorithm for a given data set and visualize the output.

DESCRIPTION:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

CODE & OUTPUT:

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

Training the K-NN model on the Training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
print(classifier.predict(sc.transform([[30,87000]])))
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Making the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
# Finding precision and recall
from sklearn.metrics import precision_score, recall_score
c=precision_score(y_test, y_pred)
d=recall_score(y_test, y_pred)
print(c,end="\n")
print(d)
```

```
[[64  4]
 [ 3 29]]
Accuracy = 0.93
Precision = 0.8787878787878788
Recall = 0.90625
```

CONCLUSION:

1.Illustrated KNN algorithm and visualised the output.

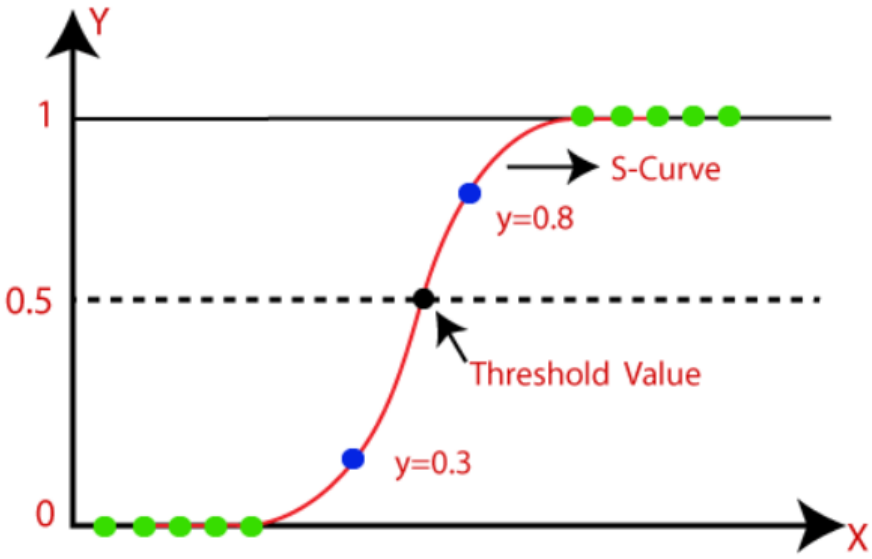
2.The accuracy, precision and recall were found to be 0.93,0.88 and 0.91 respectively.

Experiment-5

AIM: To implement logistic regression algorithm for a given data set and visualize the output.

DESCRIPTION:

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



CODE & OUTPUT:

Importing the libraries

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

Importing the dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train)
print(X_test)
```

Training the Logistic Regression model on the Training set

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Predicting a new result

```
print(classifier.predict(sc.transform([[30,87000]])))
```

Predicting the Test set results

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
# Finding precision and recall
from sklearn.metrics import precision_score, recall_score
c=precision_score(y_test, y_pred)
d=recall_score(y_test, y_pred)
print(c,end="\n")
print(d)
```

```
[[65  3]
 [ 8 24]]
Accuracy = 0.89
Precision = 0.8888888888888888
Recall = 0.75
```

CONCLUSION:

1.Illustrated logistic regression algorithm and visualised the output.
2.The accuracy, precision and recall were found to be 0.89,0.89 and 0.75 respectively.

Experiment-6

AIM: To implement Naïve Bayes classifier algorithm for a given data set and calculate accuracy, precision and recall for given dataset

DESCRIPTION:

Naïve bayes is based on the bayes theorem that is used to solve classification problem by following a probabilistic approach.

It is based on the idea that predicator variables in a Machine learning model are independent of each other.

CODE & OUTPUT:

```
#Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
#Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.25, random_state = 0)

print(X_train)
print(y_train)
print(X_test)
print(y_test)
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

print(X_train)
print(X_test)

#Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
```

```
classifier.fit(X_train, y_train)
```

```
GaussianNB()
```

```
#Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
#Predicting the Test set results
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len
(y_test),1)),1))
```

```
[0]
```

```
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[65  3]
 [ 7 25]]
0.9
```

```
# Finding precision and recall
from sklearn.metrics import precision_score, recall_score
c=precision_score(y_test, y_pred)
d=recall_score(y_test, y_pred)
print("precision score: ",c,end="\n")
print("Recall score: ",d)
```

```
precision score:  0.8928571428571429
Recall score:  0.78125
```

CONCLUSION:

1.Illustrated Naïve Bayes algorithm and visualised the output.
2.The precision and recall were found to be 0.89 and 0.78 respectively.

Experiment-7

AIM:

Build the decision tree classifier compare its performance with ensemble techniques like random forest, bagging, boosting and voting Demonstrate it with different decision trees.

DESCRIPTION:

Decision tree classifiers are popular in machine learning as they provide an intuitive way to visualize and understand the decision-making process. However, decision tree classifiers can suffer from overfitting, and their performance can be improved by using ensemble techniques such as random forests, bagging, boosting, and voting. Random forests build multiple decision trees by randomly selecting a subset of features and data points for each tree. Bagging, on the other hand, builds multiple decision trees on bootstrapped samples of the data and aggregates the results. Boosting builds decision trees sequentially by adjusting the weights of misclassified data points, while voting combines the predictions of multiple decision trees. These ensemble techniques can improve the performance of decision tree classifiers by reducing overfitting and increasing the stability and accuracy of the model. Demonstrating the performance of different decision trees and ensemble techniques on a given dataset can help in selecting the best model for the problem at hand.

CODE & OUTPUT:

```
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
x = iris.data[:, :4]
y = iris.target


cols = ['Ensemble method', 'Accuracy']
summary = []


from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2
5, random_state=1)


#Bagging
name = 'bagging'
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

model = BaggingClassifier()
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])


#Gradient Boosting
name = 'Gradient Boostng'
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()
model.fit(x_train, y_train)


y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])


#Staking
name = 'Stacking'
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

base_learners = [
    ('rf_1', RandomForestClassifier()),
    ('rf_2', KNeighborsClassifier(n_neighbors=5))
]

model = StackingClassifier(estimators = base_learners)
model.fit(x_train, y_train)


y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])


summary = pd.DataFrame(summary, columns=cols)
summary
```

	Ensemble method	Accuracy
0	bagging	0.973684
1	Gradient Boostng	0.973684
2	Stacking	0.973684

CONCLUSION:

In conclusion, decision tree classifiers are a useful tool in machine learning for understanding the decision-making process of a model. However, they can suffer from overfitting and limited performance. Ensemble techniques like random forest, bagging, boosting, and voting can improve the performance of decision tree classifiers by reducing overfitting and increasing stability and accuracy.

Experiment-8

AIM: Implementation of Gradient Descent Algorithm using Tensor Flow.

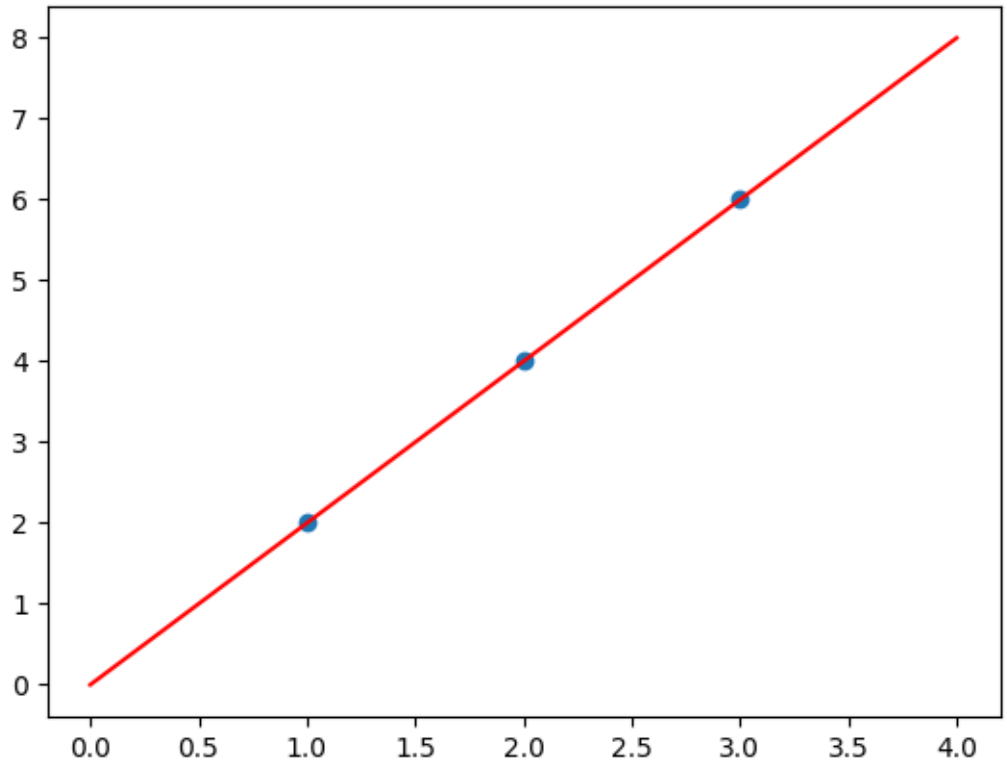
DESCRIPTION:

- Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.
- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively update the values to reach the minimum cost function.

CODE & OUTPUT:

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

# Define your data and model
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]
w = tf.Variable(1.0)

# Define the loss function
def loss(x, y):
    y_pred = x * w
    return tf.reduce_mean(tf.square(y_pred - y))

# Define the optimizer and learning rate
optimizer = tf.optimizers.SGD(learning_rate=0.01)

# Define the training loop
def train(x, y):
    with tf.GradientTape() as tape:
        l = loss(x, y)
    gradients = tape.gradient(l, [w])
    optimizer.apply_gradients(zip(gradients, [w]))

# Train the model for a certain number of epochs
for epoch in range(100):
    train(x_data, y_data)
```

```
# Plot the data points and the learned linear function
x_plot = np.linspace(0, 4, 10)
y_plot = w.numpy() * x_plot
plt.scatter(x_data, y_data)
plt.plot(x_plot, y_plot, color='red')
plt.show()
```



CONCLUSION:

In this conversation, I provided an example code for performing gradient descent using TensorFlow in Python. The code defines a simple linear model and minimizes the mean squared error between the predicted **y** values and the true **y** values using gradient descent. I also provided an updated code that plots the training data points and the learned linear function. This example demonstrates the power of TensorFlow for performing numerical optimization and machine learning tasks.

Experiment-9

AIM: Demonstration of clustering algorithms - k-Means, Agglomerative and DBSCAN to classify for the standard datasets.

DESCRIPTION:

K-Means:

K-means clustering algorithm is a popular unsupervised machine learning technique used for clustering data points into K number of clusters based on their similarity. The algorithm iteratively partitions the data into K clusters by first randomly initializing K cluster centroids, then assigning each data point to the nearest centroid based on its Euclidean distance, and finally moving the centroids to the mean of the data points in each cluster. This process is repeated until convergence, which occurs when the centroids no longer move significantly or a maximum number of iterations is reached. K-means is widely used in applications such as customer segmentation, image segmentation, and anomaly detection, and it can be extended to handle non-spherical clusters. However, it is sensitive to the initial placement of the centroids, and choosing the right value of K is important for obtaining good results.

Agglomerative Clustering:

Agglomerative clustering is another popular technique in unsupervised machine learning used for clustering data points. Unlike the K-means algorithm, which starts with K centroids, agglomerative clustering starts with each data point as a separate cluster and iteratively merges the most similar clusters until only one cluster remains. This process continues until a stopping criterion is met, such as reaching a desired number of clusters or a certain level of similarity. The similarity between clusters is measured using a linkage criterion, which can be based on different metrics such as Euclidean distance, Manhattan distance, or correlation. Agglomerative clustering is a hierarchical clustering technique that results in a dendrogram, which shows the hierarchical relationships between clusters. Agglomerative clustering is useful in applications such as gene expression analysis, image segmentation, and document clustering, and it can handle non-spherical and irregularly shaped clusters. However, it can be computationally expensive for large datasets and may suffer from the chaining effect, where small clusters are merged too early, leading to less optimal results.

DBSCAN:

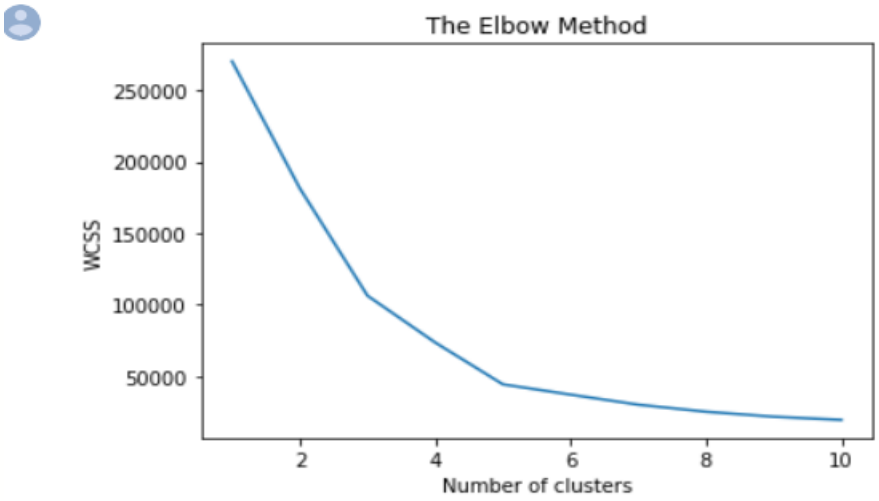
Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm that groups data points based on their density. It is a density-based clustering technique that can identify clusters of arbitrary shape and is robust to noise. The algorithm works by defining a neighborhood around each data point based on a distance metric and a user-defined radius. The algorithm then classifies each point as a core

point, border point, or noise point based on the number of points within its neighborhood. The core points are then used to form clusters, and border points are assigned to their nearest core point. The noise points are not assigned to any cluster. DBSCAN is useful in applications such as image segmentation, anomaly detection, and customer segmentation. It does not require a predetermined number of clusters and can handle noisy data. However, it is sensitive to the choice of parameters, such as the radius and the minimum number of points required to form a cluster, and it can be computationally expensive for large datasets.

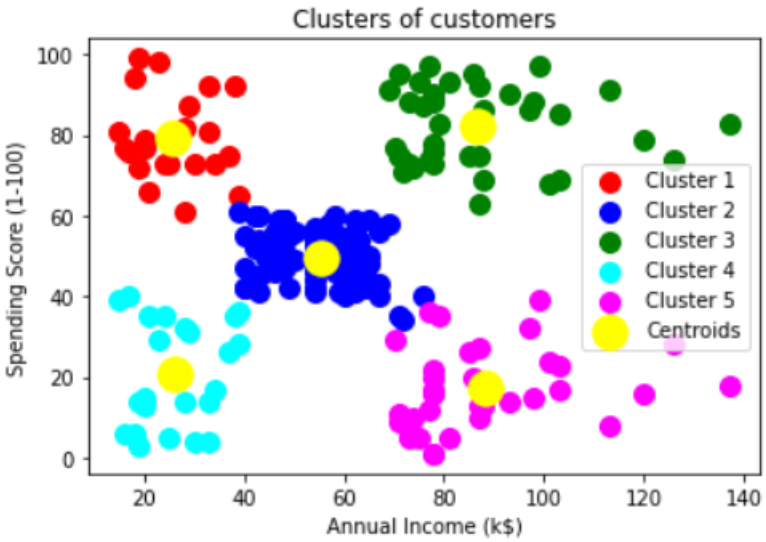
CODE & OUTPUT:

K-Means

```
#Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
#Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-
means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



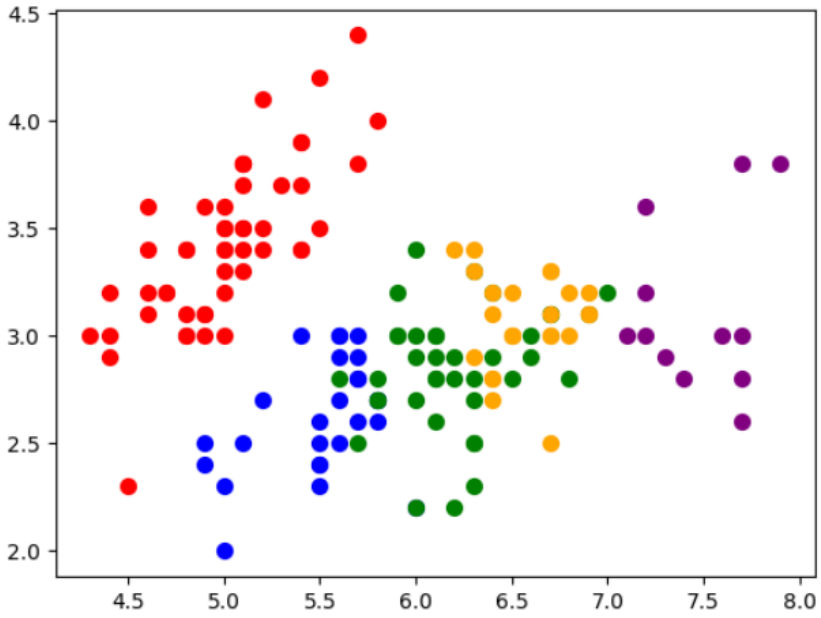
```
#Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

#Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red
', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blu
e', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'gre
en', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cya
n', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'mag
enta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1
], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



Agglomerative Clustering :

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
```

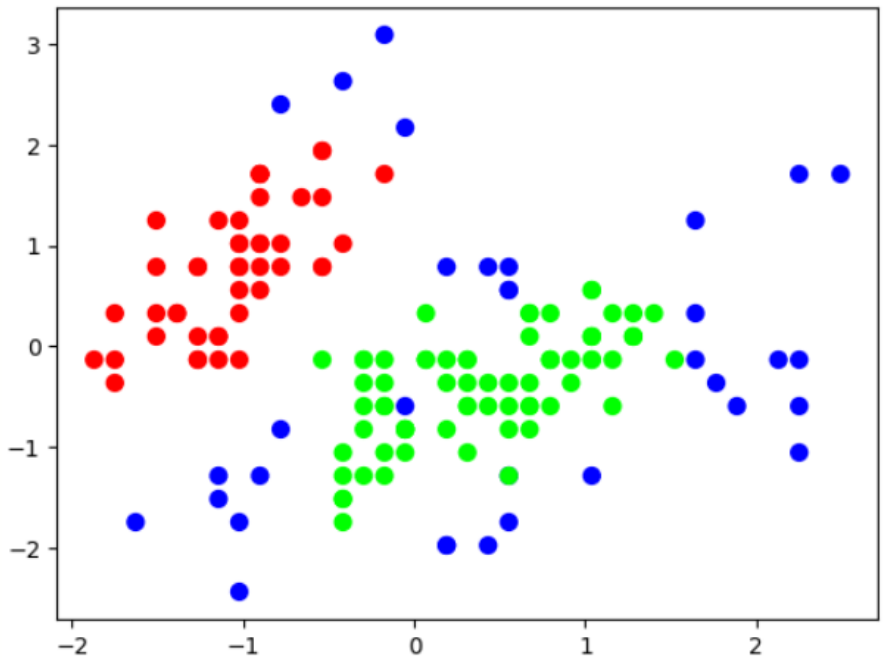
```
iris = load_iris()
X = iris.data


dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))


model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', lin
kage='ward')
model.fit(X)
labels = model.labels_


plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o', color='
red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o', color='
blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o', color='
green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o', color='
purple')
plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o', color='
orange')
plt.show()
```



DBSCAN :

```
from sklearn.datasets import load_iris
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
```

```
iris = load_iris()
X = iris.data
X = StandardScaler().fit_transform(X)

dbscan = DBSCAN(eps=0.5, min_samples=5)

dbscan.fit(X)
labels = dbscan.labels_

colors = np.array(['#ff0000', '#00ff00', '#0000ff'])

plt.scatter(X[:, 0], X[:, 1], c=colors[labels], s=50)
plt.savefig('dbscan.png', dpi=600)
plt.show()
```



CONCLUSION:

In conclusion, clustering algorithms such as k-Means, Agglomerative, and DBSCAN provide powerful tools for grouping data points with similar characteristics. Each algorithm has its own strengths and weaknesses, and the choice of algorithm depends on the characteristics of the data and the problem being addressed.