MANGALORE UNIVERSITY

MASTER OF SCIENCE
IN
COMPUTER SCIENCE

22CSP309: Artificial Intelligence and Machine Learning Lab

SUBMITTED BY

III SEMESTER MSC
Computer Science Students

SUBMITTED TO

Chairperson
The Department of Computer Science

Examiner:

1.

2.

Mangalore University
Dept. of Post-Graduate Studies and Research in Computer Science
Mangalagangothri – 574199

INDEX

1. Write A Python Code To Demonstrate Principal Components Analysis

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

# Step 1: Create some sample data (replace this with your dataset)
data = pd.read_csv('/content/drive/MyDrive/ex1data1.txt')  # 100 samples with 3 features
np.set_printoptions(precision=4, suppress=True)
print("Formatted Array:")
print(data)
```

```
Formatted Array:
    6.1101    17.592
0    5.5277   9.13020
1    8.5186  13.66200
2    7.0032  11.85400
3    5.8598   6.82330
4    8.3829  11.88600
..      ...       ...
91   5.8707   7.20290
92   5.3054   1.98690
93   8.2934   0.14454
94  13.3940   9.05510
95   5.4369   0.61705

[96 rows x 2 columns]
```

```
# Step 2: Standardize the data
mean = np.mean(data, axis=0)
#std_dev = np.std(data, axis=0)
standardized_data = (data - mean)
print (mean)
print(standardized_data)
```

```
6.1101    8.181151
17.592    5.716709
  6.1101    17.592
0  -2.653451  3.413491
1   0.337449  7.945291
```

```
2  -1.177951  6.137291
3  -2.321351  1.106591
4   0.201749  6.169291
..       ...       ...
91 -2.310451  1.486191
92 -2.875751 -3.729809
93  0.112249 -5.572169
94  5.212849  3.338391
95 -2.744251 -5.099659

[96 rows x 2 columns]
```

```
# Step 3: Compute the covariance matrix
covariance_matrix = np.cov(standardized_data, rowvar=False)
size_cc = covariance_matrix.size
shape_cc = covariance_matrix.shape
print (size_cc, shape_cc)
print(covariance_matrix)
```

```
4 (2, 2)
[[15.089  18.3112]
 [18.3112 29.2135]]
```

```
# Step 4: Compute the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
print(eigenvalues)
print(eigenvectors)
```

```
[ 2.5254 41.7771]
[[-0.8246  0.5658]
 [ 0.5658  0.8246]]
```

```
# Step 5: Sort eigenvalues and corresponding eigenvectors in descending order
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
print(eigenvalues)
print(eigenvectors)
```

```
[41.7771  2.5254]
[[ 0.5658 -0.8246]
 [ 0.8246  0.5658]]
```

```
# Step 6: Choose the number of components (or a threshold for explained variance)
n_components = 1  # Choose the number of principal components

# Step 7: Select the top 'n_components' eigenvectors
selected_eigenvectors = eigenvectors[:, :n_components]

# Step 8: Project the data onto the selected eigenvectors to obtain the principal components
final_result = np.dot(standardized_data, selected_eigenvectors)

# Step 9: Print the final result
print("Final Result after PCA:")
print(final_result)
```

```
Final Result after PCA:
[[ 1.3135]
 [ 6.7424]
 [ 4.3942]
 [-0.4008]
 [ 5.2012]
 [-1.5271]
 [ 5.4056]
 [-0.2317]
 [-3.3356]
 [-3.4298]
 [11.456 ]
 [-3.4967]
 [ 1.3729]
 [-5.5606]
 [-3.4023]
 [-1.3669]
 [-5.9776]
 [-2.6936]
 [-0.8982]
 [-3.2529]
 [20.074 ]
 [-2.7212]
 [-1.4859]
 [-3.6523]
 [20.0425]
```

2. WRITE A LINEAR REGRESSION PROGRAM TO PERFORM WITHOUT BUILT-IN FUNCTION.

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
df=pd.read_csv('/content/drive/MyDrive/Salary_Data.csv')
df
```

output:

```
   x  y

0  1  2

1  2  4

2  3  5

3  4  4

4  5  5
```

```
import matplotlib.pyplot as plt
mean_x=df['YearsExperience'].mean()
mean_y=df['Salary'].mean()
print(mean_x)
print(mean_y)
```

output:

```
3.0
4.0
```

```
numerator = 0
denominator = 0
for i in range(len(df['YearsExperience'])):
   numerator += (df['YearsExperience'][i] - mean_x) * (df['Salary'][i] - mean_y)
   denominator += (df['YearsExperience'][i] - mean_x) ** 2
m= numerator/denominator
```

```
c=mean_y-m*mean_x
print('slope:',m)
print('intercept:',c)
```

output:

```
slope: 0.6 intercept: 2.2
```

```
predicted_y = [(m * xi) + c for xi in df['YearsExperience']]
print('predicty:',predicted_y)
```

output:

```
predicty: [2.8000000000000003, 3.4000000000000004, 4.0, 4.6, 5.2]
```

```
yr = [df['Salary'][i] - predicted_y[i] for i in range(len(df))]
print(yr)
```

output:

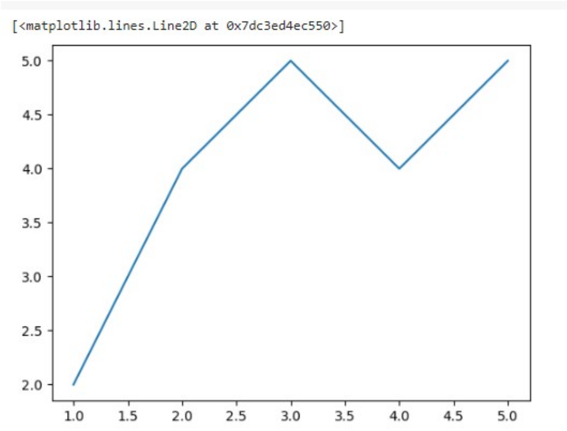
```
[-0.8000000000000003, 0.5999999999999996, 1.0, -0.5999999999999996, -
0.20000000000000018]
```

```
import numpy as np
R=np.mean(yr)
print(R)
```

output:

```
-8.881784197001253e-17
```

```
x=df['YearsExperience']
y=df['Salary']
plt.plot(x,y)
```



```
[<matplotlib.lines.Line2D at 0x7dc3ed4ec550>]
```

3. Linear regression program with built-in function.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data plotting
```

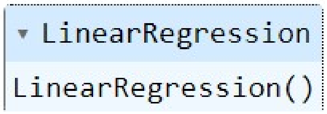
```
dataset = pd.read_csv('/content/drive/MyDrive/AI and ML/ex1data1.txt')
dataset.head()
```

```
   6.1101   17.592

0  5.5277    9.1302

1  8.5186   13.6620

2  7.0032   11.8540

3  5.8598    6.8233

4  8.3829   11.8860
```

```
X = dataset.iloc[:,:-1].values  #independent variable array
y = dataset.iloc[:,1].values  #dependent variable vector
```

```
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)
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
```



```
▾ LinearRegression
LinearRegression()
```

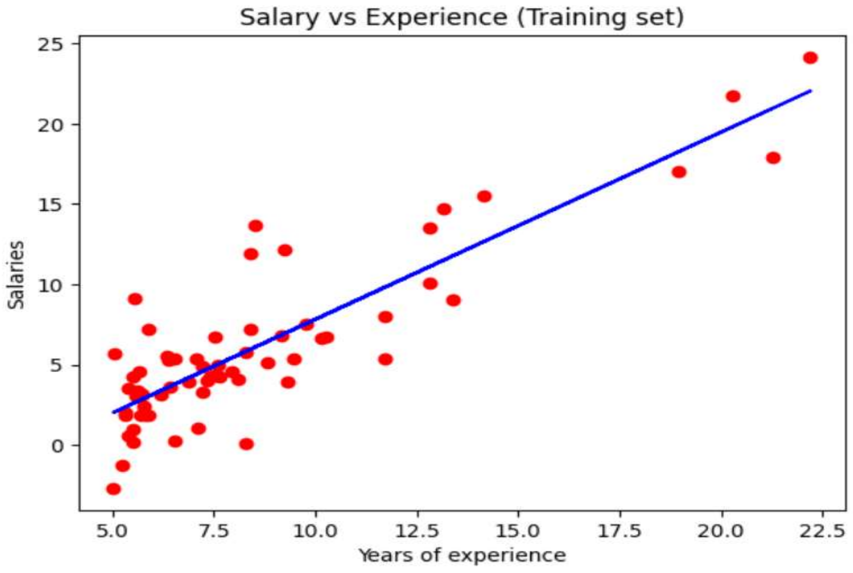
```
y_pred = regressor.predict(X_test)
y_pred
```

```
array([ 8.94195531,  6.10107029,  4.33817991,  3.48933772, 19.86861533,
        3.58423571,  2.15715616,  2.13107377,  3.58796177,  8.10242826,
        3.55757112,  2.07483361,  3.17728051,  2.75169502,  3.73618895,
        3.85635426,  3.54976969, 18.24312328,  5.74464972,  2.06924452,
        5.29472842, 13.54247042,  3.00681343,  3.17239006,  2.08740905,
        2.63630371,  6.17198179,  2.51439181,  2.22504025,  4.9615957 ,
        2.21444428,  2.85474376])
```

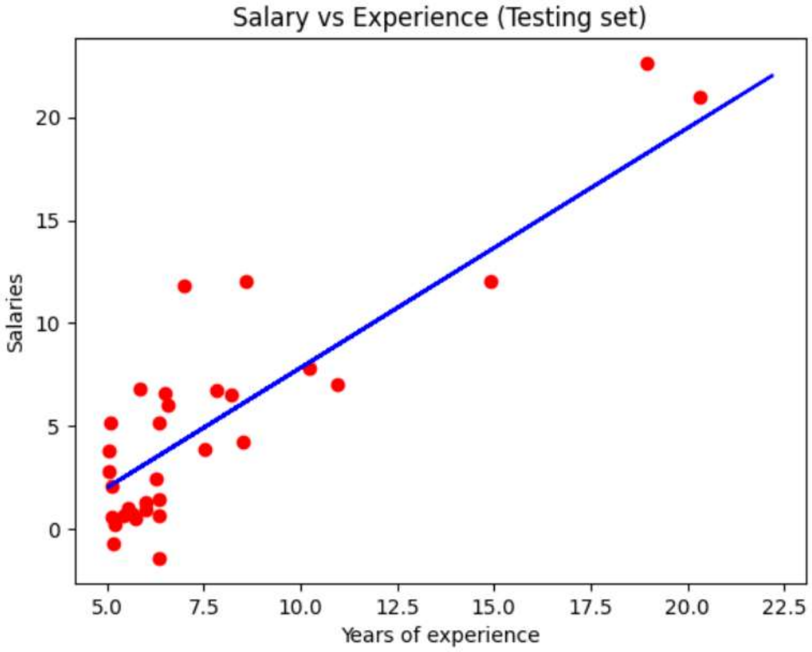
y_test

```
array([ 7.0467 ,  4.2415 , 11.854  ,  2.4756 , 20.992  ,  0.67861,
        0.56077,  2.0576 , -1.4211 ,  7.7754 ,  1.4233 ,  2.8214 ,
        1.2784 ,  0.71618,  6.5987 ,  5.9966 ,  5.1875 , 22.638  ,
        6.5426 ,  3.8166 ,  6.7318 , 12.054  ,  6.8233 ,  0.92695,
        5.1337 ,  1.0179 , 12.     ,  0.61705,  0.20421,  3.8845 ,
       -0.74279,  0.47953])
```

```
#plot for the TRAIN
plt.scatter(X_train, y_train, color='red') # plotting the observation line
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Salary vs Experience (Training set)") # stating the title of the graph
plt.xlabel("Years of experience") # adding the name of x-axis
plt.ylabel("Salaries") # adding the name of y-axis
plt.show() # specifies end of graph
```



```
#plot for the TEST
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Salary vs Experience (Testing set)")
plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

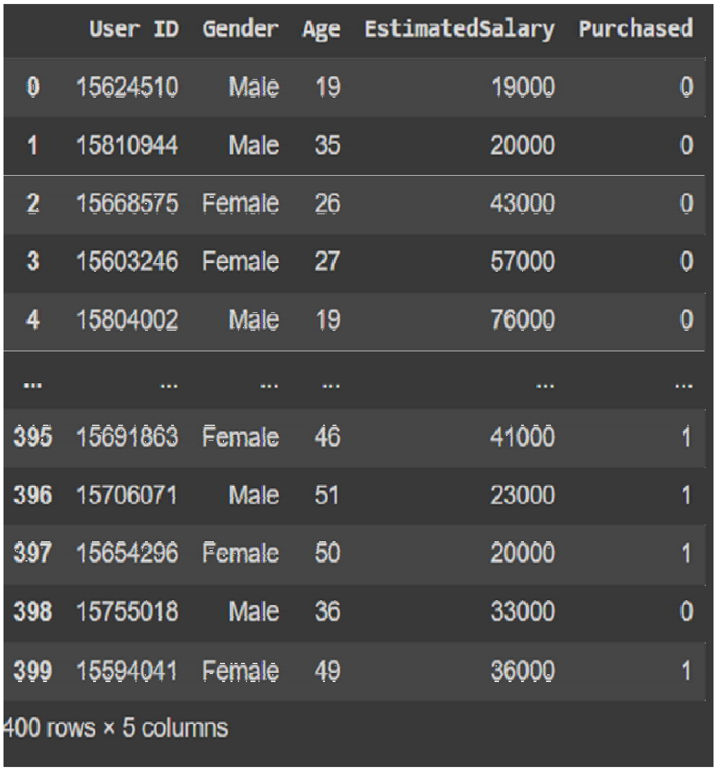


```
print("Coefficient:",regressor.coef_)
print("Intercept",regressor.intercept_)
```

```
Coefficient: [1.16439258]
Intercept -3.81629423778271
```

4. Implement Support Vector Machine (Svm) Classifier With Suitable Datasets

```
# Support Vector Machine (SVM)

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read_csv('/content/Social_Network_Ads.csv')

dataset
```



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

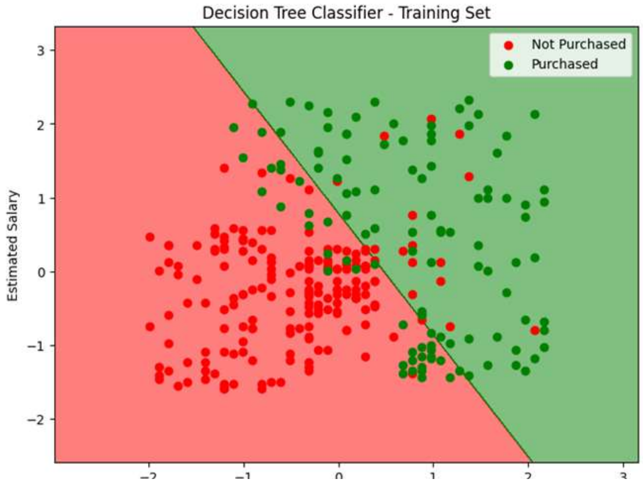
400 rows × 5 columns

```
X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into training 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)


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


# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
: SVC
SVC(kernel='linear', random_state=0)


# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred

: array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,

       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,

       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,

       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,

       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1])
```

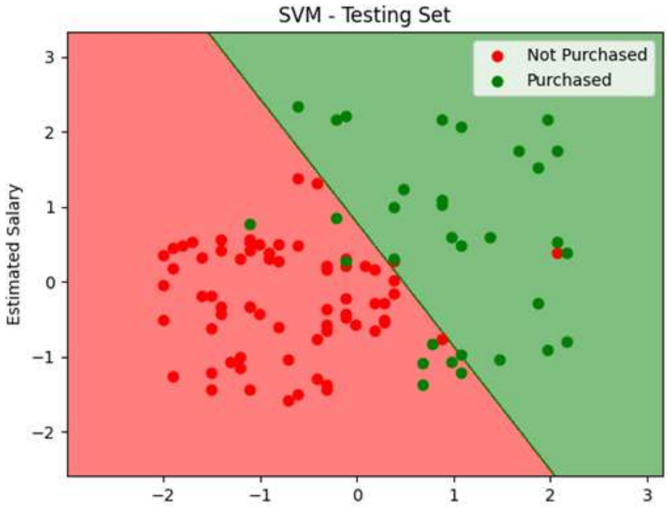
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
:array([[66,  2],
        [ 8, 24]])
```

```
# Visualizing the Training set results

from matplotlib.colors import ListedColormap
# Create a meshgrid to plot the decision boundary
X1, X2 = np.meshgrid(np.arange(start=X_train[:, 0].min() - 1, stop=X_train[:,
0].max() + 1, step=0.01), np.arange(start=X_train[:, 1].min() - 1, stop=X_train[:,
1].max() + 1, step=0.01)
# Use the classifier to predict the class labels for each point in the meshgrid
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)
# Create a color map for the plot
cmap = ListedColormap(('red', 'green'))
# Plot the training set data points
plt.figure(figsize=(8, 6))
plt.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], color='red',
label='Not Purchased')
plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], color='green',
label='Purchased')
plt.title('Decision Tree Classifier - Training Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



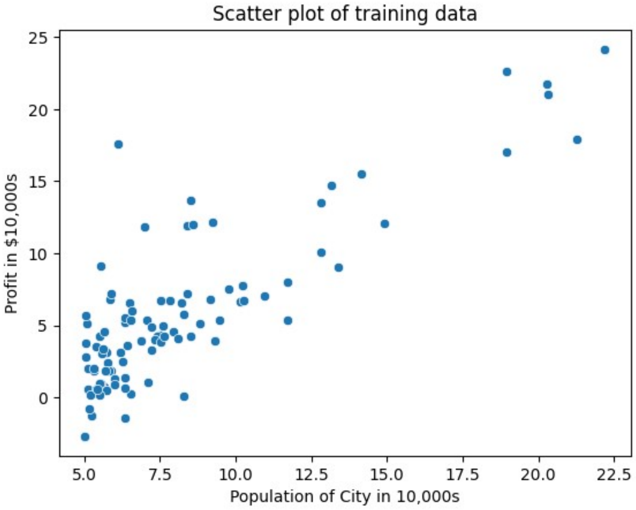
```
# Visualizing the Testing set results
from matplotlib.colors import ListedColormap
# Create a meshgrid of feature values
X1, X2 = np.meshgrid(np.arange(start = X_test[:, 0].min() - 1, stop = X_test[:,
0].max() + 1, step = 0.01),np.arange(start = X_test[:, 1].min() - 1, stop = X_test[:,
1].max() + 1, step = 0.01))
# Use the trained classifier to make predictions on the meshgrid points
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)
# Create a colormap for the two classes
cmap = ListedColormap(('red', 'green'))
# Plot the contour filled by the predictions
plt.contourf(X1, X2, Z, alpha = 0.5, cmap = cmap)
# Scatter plot the actual data points
plt.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], color = 'red', label = 'Not
Purchased')
plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], color = 'green', label =
'Purchased')
# Add labels and legend
plt.title('SVM - Testing Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
# Show the plot
plt.show()
```



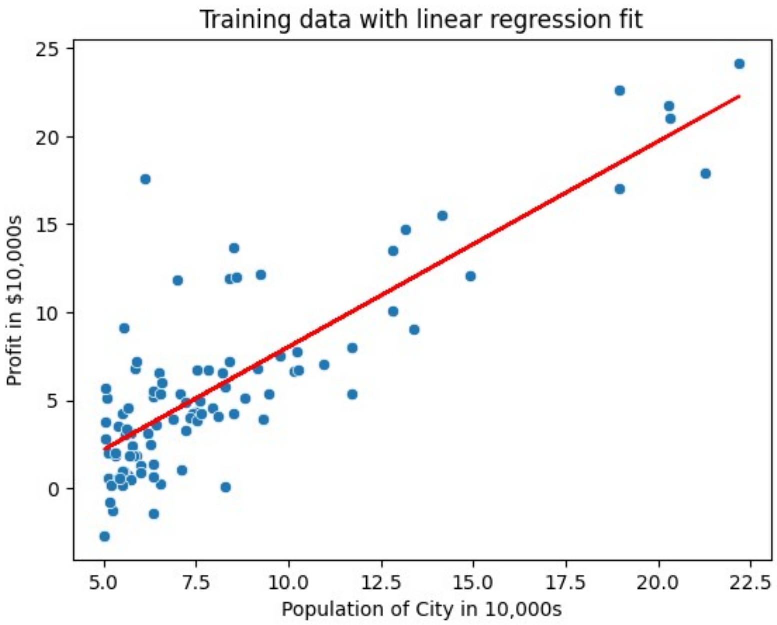
```
from sklearn.metrics import accuracy_score
print ('Accuracy : ', accuracy_score(y_test, y_pred))
: Accuracy :  0.9
```

5. Implement Gradient descent algorithm for the single variable

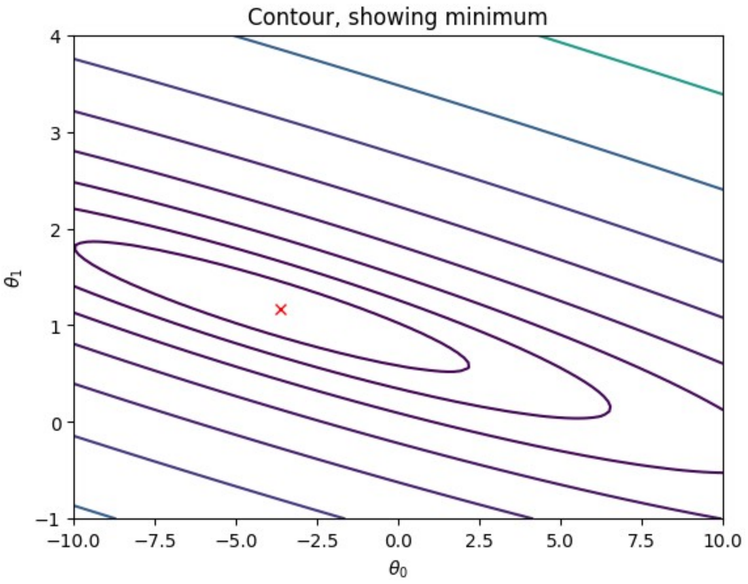
```
from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# %matplotlib inline

df = pd.read_csv('/content/drive/MyDrive/Copy of ex1data1.txt', sep=',', header=None)
df.columns = ['population', 'profit']
df

ax = sns.scatterplot(x='population', y='profit', data=df)
ax.set(xlabel='Population of City in 10,000s', ylabel='Profit in $10,000s', title='Scatter plot
of training data')
```



```
m = df.shape[0]
X = np.hstack((np.ones((m,1)), df.population.values.reshape(-1,1)))
y = np.array(df.profit.values).reshape(-1,1)
theta = np.zeros(shape=(X.shape[1],1))

iterations = 1500
alpha = 0.01
print(m, X)



def compute_cost_one_variable(X, y, theta):
    m = y.shape[0]
    h = X.dot(theta)
    J = (1/(2*m)) * (np.sum((h - y)**2))
    return J

J = compute_cost_one_variable(X, y, theta)
print('With theta = [0 ; 0]\nCost computed =', J)
print('Expected cost value (approx) 32.07')

J = compute_cost_one_variable(X, y, [[-1],[2]])
print('With theta = [-1 ; 2]\nCost computed =', J)
print('Expected cost value (approx) 54.24')

def gradient_descent(X, y, theta, alpha, num_iters):
    m = y.shape[0]
    J_history = np.zeros(shape=(num_iters, 1))

    for i in range(0, num_iters):
        h = X.dot(theta)
        diff_hy = h - y

        delta = (1/m) * (diff_hy.T.dot(X))
        theta = theta - (alpha * delta.T)
        J_history[i] = compute_cost_one_variable(X, y, theta)

    return theta, J_history

theta, _ = gradient_descent(X, y, theta, alpha, iterations)
print('Theta found by gradient descent:\n', theta)
print('Expected theta values (approx)\n -3.6303\n  1.1664')

ax = sns.scatterplot(x='population', y='profit', data=df)
plt.plot(X[:,1], X.dot(theta), color='r')
ax.set(xlabel='Population of City in 10,000s', ylabel='Profit in $10,000s', title='Training
data with linear regression fit')
```



```
y_pred = np.array([1, 3.5]).dot(theta)
f'For population = 35,000, we predict a profit of {y_pred[0]*10000}'

y_pred = np.array([1, 7]).dot(theta)
f'For population = 70,000, we predict a profit of {y_pred[0]*10000}'

theta0_vals = np.linspace(-10, 10, 100)
theta1_vals = np.linspace(-1, 4, 100)

J_vals = np.zeros(shape=(len(theta0_vals), len(theta1_vals)))

for i in range(0, len(theta0_vals)):
    for j in range(0, len(theta1_vals)):
        J_vals[i,j] = compute_cost_one_variable(X, y, [[theta0_vals[i]], [theta1_vals[j]]])

ax = plt.contour(theta0_vals, theta1_vals, np.transpose(J_vals), levels=np.logspace(-2,3,20))
plt.plot(theta[0,0], theta[1,0], marker='x', color='r')
plt.xlabel(r'$\theta_0$')
plt.ylabel(r'$\theta_1$')
plt.title('Contour, showing minimum')
```



```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(df.population.values.reshape(-1,1),
          df.profit.values.reshape(-1,1))

print(lin_reg.intercept_, lin_reg.coef_)
```

6. Implement Text Preprocessing

1. **Removing punctuations**
2. **Removing digits**
3. **Removing Special characters**
4. **TokenizatioN**
5. **Lemmatization**
6. **Removal of Stopwords**
7. **Demojize**

```
from google.colab import drive
drive.mount('/content/drive')

!pip install contractions
!pip install emoji
!pip install nltk

import pandas as pd
import numpy as np
from nltk.tokenize import word_tokenize
import statistics
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('whitespace')
nltk.download('wordnet')
import json
from nltk import word_tokenize
from nltk.corpus import stopwords
import string
import contractions
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
import re
from nltk.tokenize import TweetTokenizer
import emoji
import regex
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
```

```
#read dataset
dft=pd.read_csv("/content/drive/MyDrive/principle of data science lab/sample_text
- Sheet1.csv")
dft
```

```
                tweet_id            text     task1
0     1123757263427186690     hate wen females hit ah nigga with tht bro □□,...     HOF
1     1123733301397733380     RT @airjunebug: When you're from the Bay but y... HOF
2     1123734094108659712     RT @DonaldJTrumpJr: Dear Democrats: The Americ...
      NOT
3     1126951188170199049     RT @SheLoveTimothy: He ain't on drugs he just ... HOF
4     1126863510447710208     RT @TavianJordan: Summer '19 I'm coming for yo...
      NOT
...   ...     ...     ...
995   1126798721025544193     RT @prodnose: Good morning, everyone.\nFollowi...
      NOT
996   1126833089190219777     @cheezitking123 this what you get for tryna ge...     NOT
997   1130037092845670400     earphones ko □□□□□□□     NOT
998   1127028455651123201     RT @nj_linguist: @realgonegirl @elivalley I th...     NOT
999   1130285076858789889     i'm tired as fuck. and man, physically ain't S...     HOF
1000 rows × 3 columns
```

```
#tokenization
dft['Text_Tokenized'] = dft['text'].str.lower().apply(word_tokenize)
dft
```

```
        tweet_id        text    task1   Text_Tokenized
0       1123757263427186690     hate wen females hit ah nigga with tht bro □□,...    HOF
                [hate, wen, females, hit, ah, nigga, with, tht...
1       1123733301397733380     RT @airjunebug: When you're from the Bay but y... HOF
        [rt,
2       1123734094108659712     RT @DonaldJTrumpJr: Dear Democrats: The Americ...
        NOT     [rt,
3       1126951188170199049     RT @SheLoveTimothy: He ain’t on drugs he just ... HOF
        [rt,
4       1126863510447710208     RT @TavianJordan: Summer ‘19 I’m coming for yo...
        NOT     [rt,
```

```
#Preprocessing
ps =PorterStemmer()
lemmatiser = WordNetLemmatizer()
english_stopwords = stopwords.words('english')
exclude = set(string.punctuation)
text=dft['text'].iloc[:40]
def preprocess(text):
  text=contractions.fix(text.lower(),slang=True)
  text =re.sub("@ ?[A-Za-z0-9_]+", "", text)
  text= re.sub(r'\d+', '', text)
  text=re.sub(r'$', '', text)
  text= re.sub(r'’','', text )
  text=re.sub('<.*?>','',text)
  text=re.sub(r'http\S+', '', text)
  tokens = word_tokenize(text)
  tokens = [lemmatiser.lemmatize(t) for t in tokens]
  tokens = [t for t in tokens if t not in english_stopwords]
  return text
```

```
dft['preprocessed_text'] = dft['text'].apply(preprocess)
dft['preprocessed_text']
```

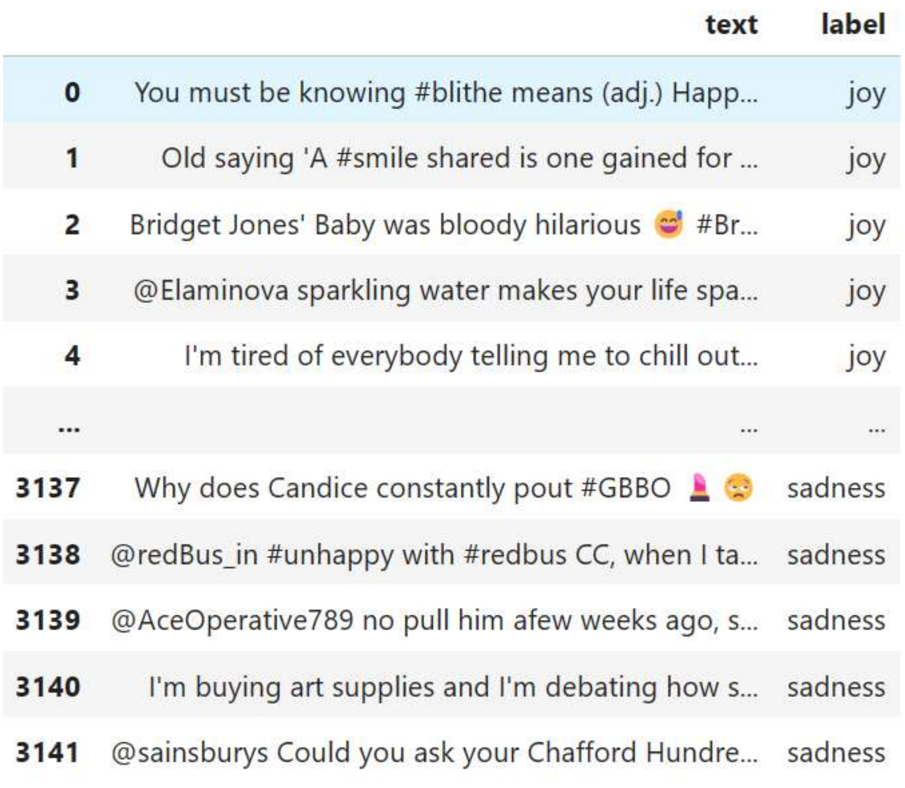
```
0      hate wen females hit ah nigga with tht bro □□,...
1      rt : when you are from the bay but you are rea...
2      rt : dear democrats: the american people are n...
3      rt : he are not on drugs he just bored. i be d...
4      rt : summer ‘ i am coming for you ! no boring ...
```

```
import emoji
def remove_emojis(text):
   temp = emoji.demojize(text,delimiters=("",""))
   temp = temp.replace("_", "")
   return temp

dft['clean'] = dft['preprocessed_text'].apply(lambda x: remove_emojis(x))
dft['clean']=dft['clean'].apply(lambda X: preprocess(X))
dft['clean']
```

```
0      hate wen females hit ah nigga with tht bro fac...
1      rt : when you are from the bay but you are rea...
2      rt : dear democrats: the american people are n...
3      rt : he are not on drugs he just bored. i be d...
4      rt : summer ‘ i am coming for you ! no boring ...
```

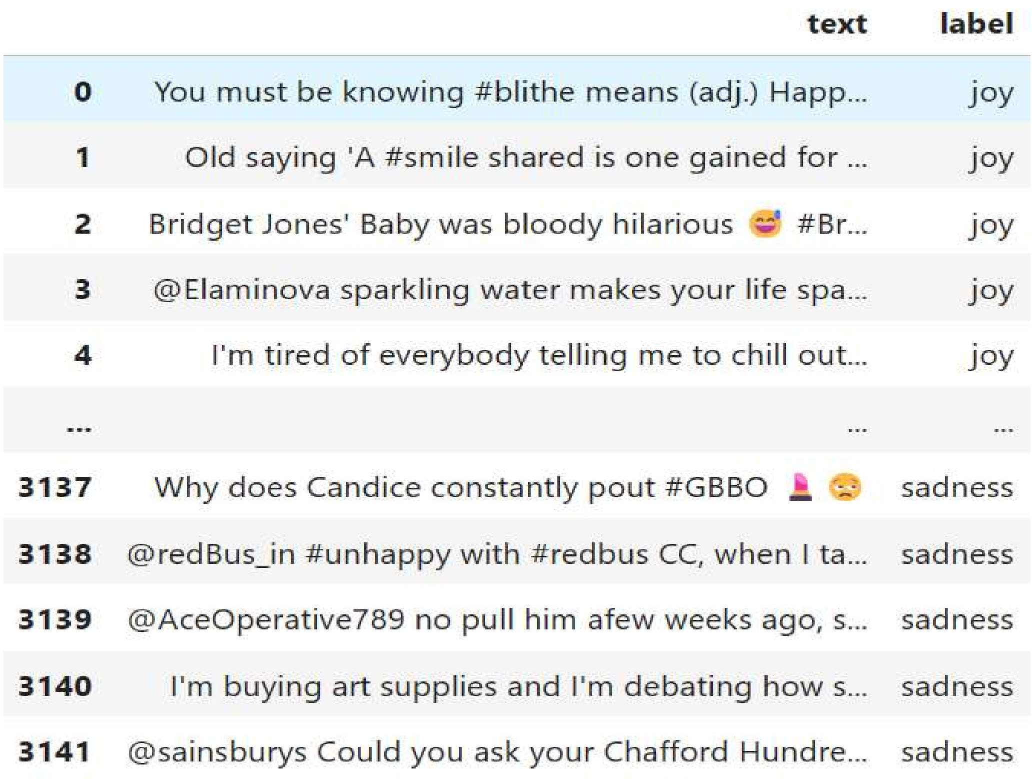
7. Implement feature extraction using TF-IDF of the models(SVM, KNN, Logistic Regression, Decision tree)

```
from google.colab import drive
drive.mount('/content/drive')

!pip install emoji
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
import string
import emoji
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import re
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report


nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Load your CSV file into a pandas DataFrame
csv_file_path = '/content/drive/MyDrive/lab programs/emotion-labels-test.csv'  # Replace with the path to your CSV file
df = pd.read_csv(csv_file_path)
df
```



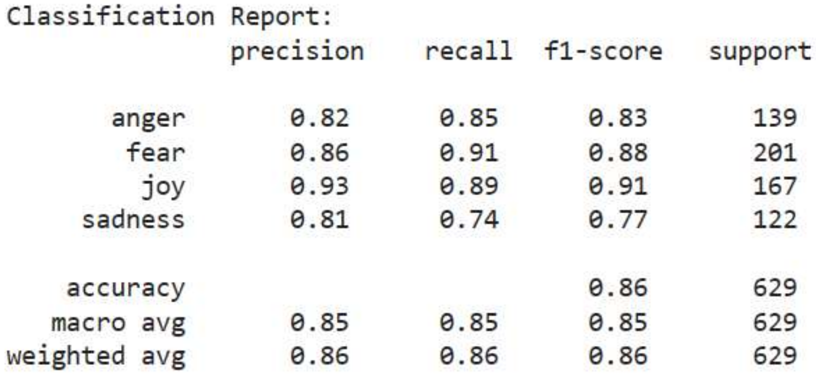
	text	label
0	You must be knowing #blithe means (adj.) Happ...	joy
1	Old saying 'A #smile shared is one gained for ...	joy
2	Bridget Jones' Baby was bloody hilarious 😅 #Br...	joy
3	@Elaminova sparkling water makes your life spa...	joy
4	I'm tired of everybody telling me to chill out...	joy
...	...	...
3137	Why does Candice constantly pout #GBBO 💄 😣	sadness
3138	@redBus_in #unhappy with #redbus CC, when I ta...	sadness
3139	@AceOperative789 no pull him afew weeks ago, s...	sadness
3140	I'm buying art supplies and I'm debating how s...	sadness
3141	@sainsburys Could you ask your Chafford Hundre...	sadness

```
#Pre-Processing
ps = PorterStemmer()
lemmatiser = WordNetLemmatizer()
english_stopwords = set(stopwords.words('english'))
exclude = set(string.punctuation)

def preprocess(text):
    # Convert to lowercase
    text = text.lower()

    # Remove digits
    text = re.sub(r'\d+', '', text)

    # Remove HTML tags
    text = re.sub('<.*?>', '', text)

    # Remove URLs
    text = re.sub(r'http\S+', '', text)

    # Remove non-ASCII characters
    text = text.encode("ascii", "ignore").decode()

    # Remove punctuation
    text = ''.join(ch for ch in text if ch not in exclude)

    # Tokenize, lemmatize, and stem
    tokens = word_tokenize(text)
    tokens = [lemmatiser.lemmatize(t) for t in tokens]
    tokens = [ps.stem(t) for t in tokens]

    # Remove stopwords
    tokens = [t for t in tokens if t not in english_stopwords]

    # Join tokens back into text
    text = " ".join(tokens)

    # Demojize and replace underscores
    text = emoji.demojize(text, delimiters=(" ", " "))
    text = text.replace("_", " ")
    return text

df['processed_text'] = df['text'].astype(str).apply(preprocess)
df
```



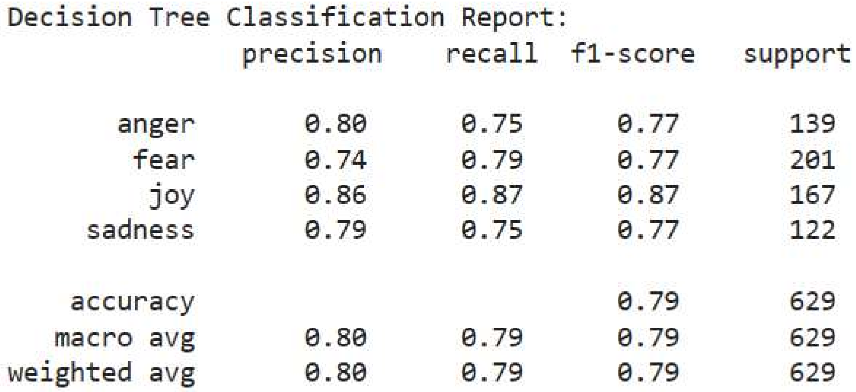
	text	label
0	You must be knowing #blithe means (adj.) Happ...	joy
1	Old saying 'A #smile shared is one gained for ...	joy
2	Bridget Jones' Baby was bloody hilarious 😅 #Br...	joy
3	@Elaminova sparkling water makes your life spa...	joy
4	I'm tired of everybody telling me to chill out...	joy
...	...	...
3137	Why does Candice constantly pout #GBBO 💄 😒	sadness
3138	@redBus_in #unhappy with #redbus CC, when I ta...	sadness
3139	@AceOperative789 no pull him afew weeks ago, s...	sadness
3140	I'm buying art supplies and I'm debating how s...	sadness
3141	@sainsburys Could you ask your Chafford Hundre...	sadness

```
X = df['processed_text']
y = df['label']


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)


svm_model = SVC(kernel='linear')
svm_model.fit(X_train_vec, y_train)

# Decision Tree
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train_vec, y_train))

# k-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_vec, y_train)


# Logistic Regression
lr_model = LogisticRegression()
lr_model.fit(X_train_vec, y_train)

# Classification report
y_pred = svm_model.predict(X_test_vec)
report = classification_report(y_test, y_pred)print("Classification Report:\n",
report)
```



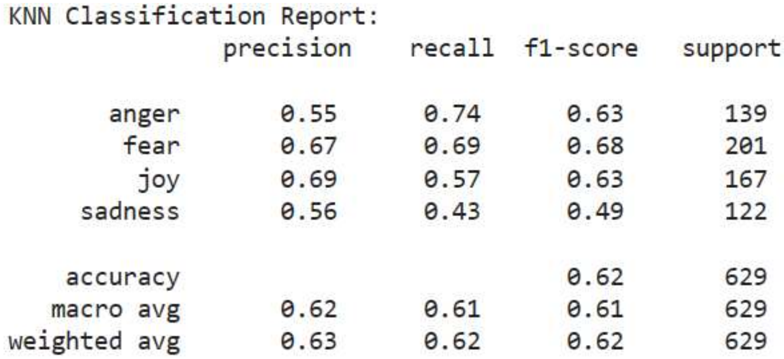
```
Classification Report:
              precision    recall  f1-score   support

       anger       0.82      0.85      0.83       139
        fear       0.86      0.91      0.88       201
         joy       0.93      0.89      0.91       167
     sadness       0.81      0.74      0.77       122

    accuracy                           0.86       629
   macro avg       0.85      0.85      0.85       629
weighted avg       0.86      0.86      0.86       629
```

```
# Classification report for Decision Tree
y_pred_dt = dt_model.predict(X_test_vec)
report_dt = classification_report(y_test, y_pred_dt)
print("Decision Tree Classification Report:\n", report_dt)
```



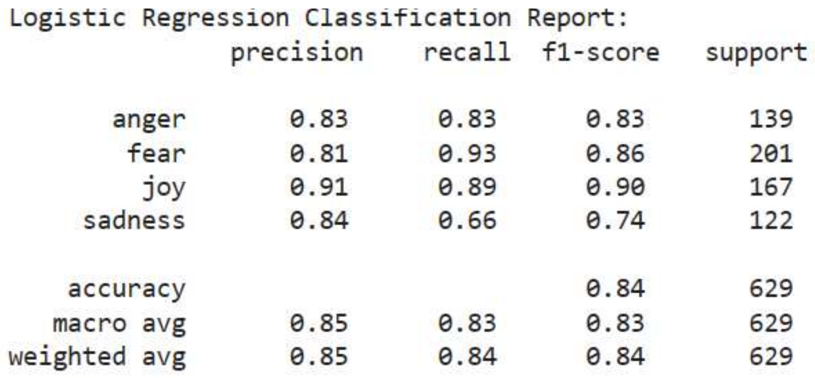
```
Decision Tree Classification Report:
               precision    recall  f1-score   support

       anger       0.80      0.75      0.77       139
        fear       0.74      0.79      0.77       201
         joy       0.86      0.87      0.87       167
     sadness       0.79      0.75      0.77       122

    accuracy                           0.79       629
   macro avg       0.80      0.79      0.79       629
weighted avg       0.80      0.79      0.79       629
```

```
# Classification report for KNN
y_pred_knn = knn_model.predict(X_test_vec)
report_knn = classification_report(y_test, y_pred_knn)
print("KNN Classification Report:\n", report_knn)
```



```
KNN Classification Report:
               precision    recall  f1-score   support

       anger       0.55      0.74      0.63       139
        fear       0.67      0.69      0.68       201
         joy       0.69      0.57      0.63       167
     sadness       0.56      0.43      0.49       122

    accuracy                           0.62       629
   macro avg       0.62      0.61      0.61       629
weighted avg       0.63      0.62      0.62       629
```

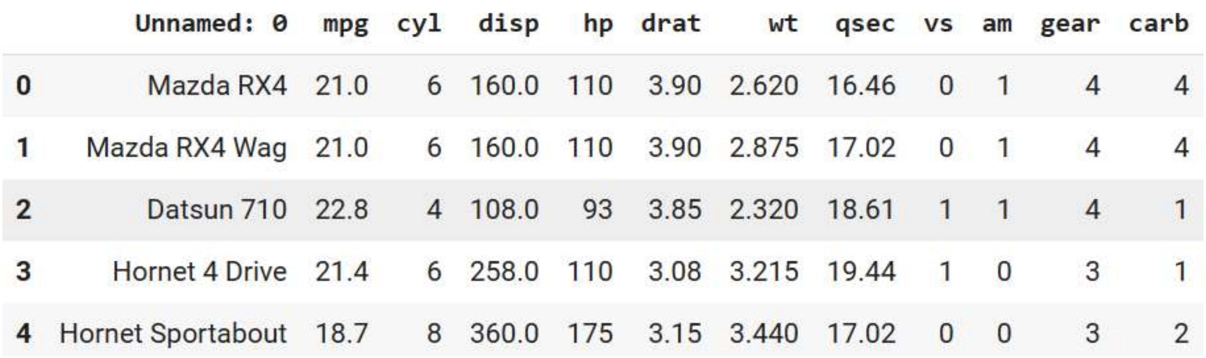
```
# Classification report for Logistic Regression
y_pred_lr = lr_model.predict(X_test_vec)
report_lr = classification_report(y_test, y_pred_lr)
print("Logistic Regression Classification Report:\n", report_lr)
```



```
Logistic Regression Classification Report:
               precision    recall  f1-score   support

       anger       0.83      0.83      0.83       139
        fear       0.81      0.93      0.86       201
         joy       0.91      0.89      0.90       167
     sadness       0.84      0.66      0.74       122

    accuracy                           0.84       629
   macro avg       0.85      0.83      0.83       629
weighted avg       0.85      0.84      0.84       629
```

8. **Write a program to perform Decision Tree classifier with built-in function on the dataset**

```
from google.colab import drive
drive.mount('/content/drive')

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

data = pd.read_csv('/content/drive/MyDrive/Principles of data science/poo.csv')
data
```



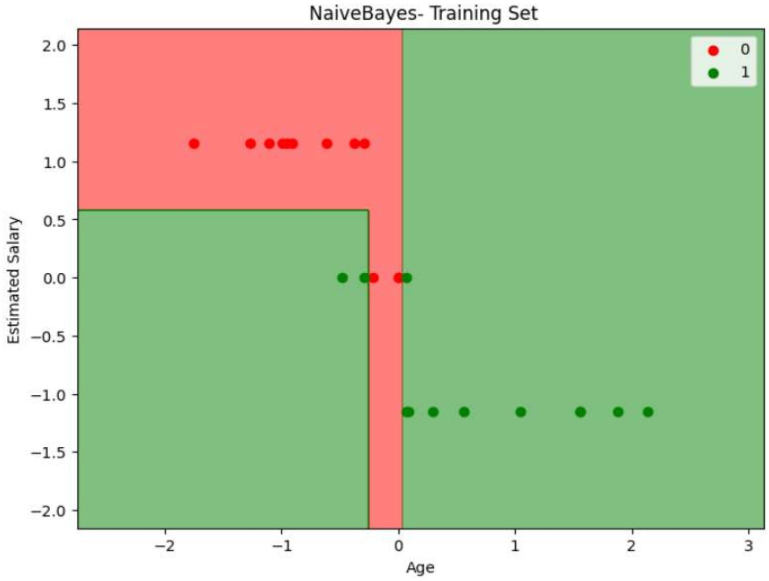
	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

```
X = data.iloc[:,[1,2]].values
y = data.iloc[:,8].values

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)

#feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train,y_train)

!pip install metrics

#prediction
y_pred = classifier.predict(X_test)
```

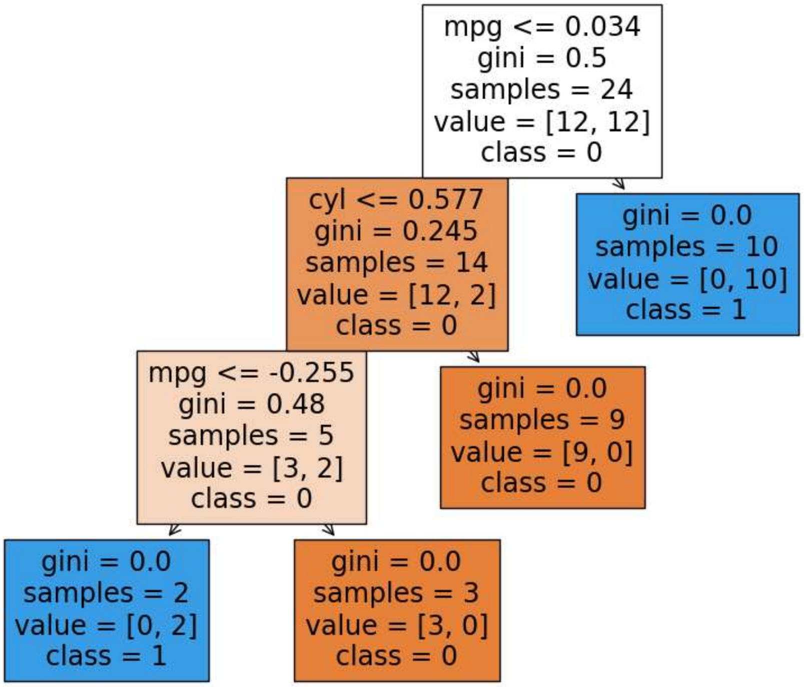
```
#Accuracy
from sklearn import metrics
print('Accuracy Score:', metrics.accuracy_score(y_test,y_pred))
```

Accuracy score is 0.875

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

# Visualizing the Training set results
from matplotlib.colors import ListedColormap

# Create a meshgrid to plot the decision boundary
X1, X2 = np.meshgrid(np.arange(start=X_train[:, 0].min() - 1, stop=X_train[:,
0].max() + 1, step=0.01),
                     np.arange(start=X_train[:, 1].min() - 1, stop=X_train[:, 1].max() + 1,
step=0.01))

# Use the classifier to predict the class labels for each point in the meshgrid
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a color map for the plot
cmap = ListedColormap(('red', 'green'))

# Plot the training set data points
plt.figure(figsize=(8, 6))
plt.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], color='red', label='0')
plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], color='green', label='1')
plt.title(' NaiveBayes- Training Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

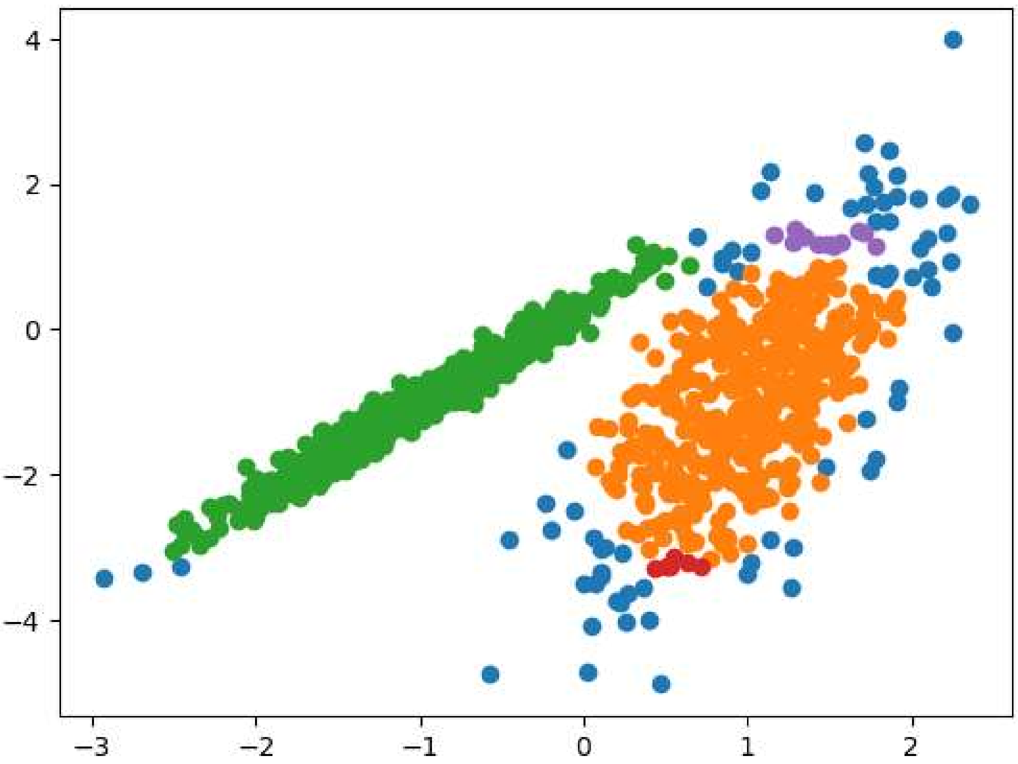


```
!pip install scikit-learn

from sklearn.tree import DecisionTreeClassifier, plot_tree

# Plotting the Decision Tree graph
plt.figure(figsize=(10, 8))
plot_tree(classifier, feature_names=['mpg', 'cyl'], class_names=['0', '1'],
filled=True)
plt.show()
```



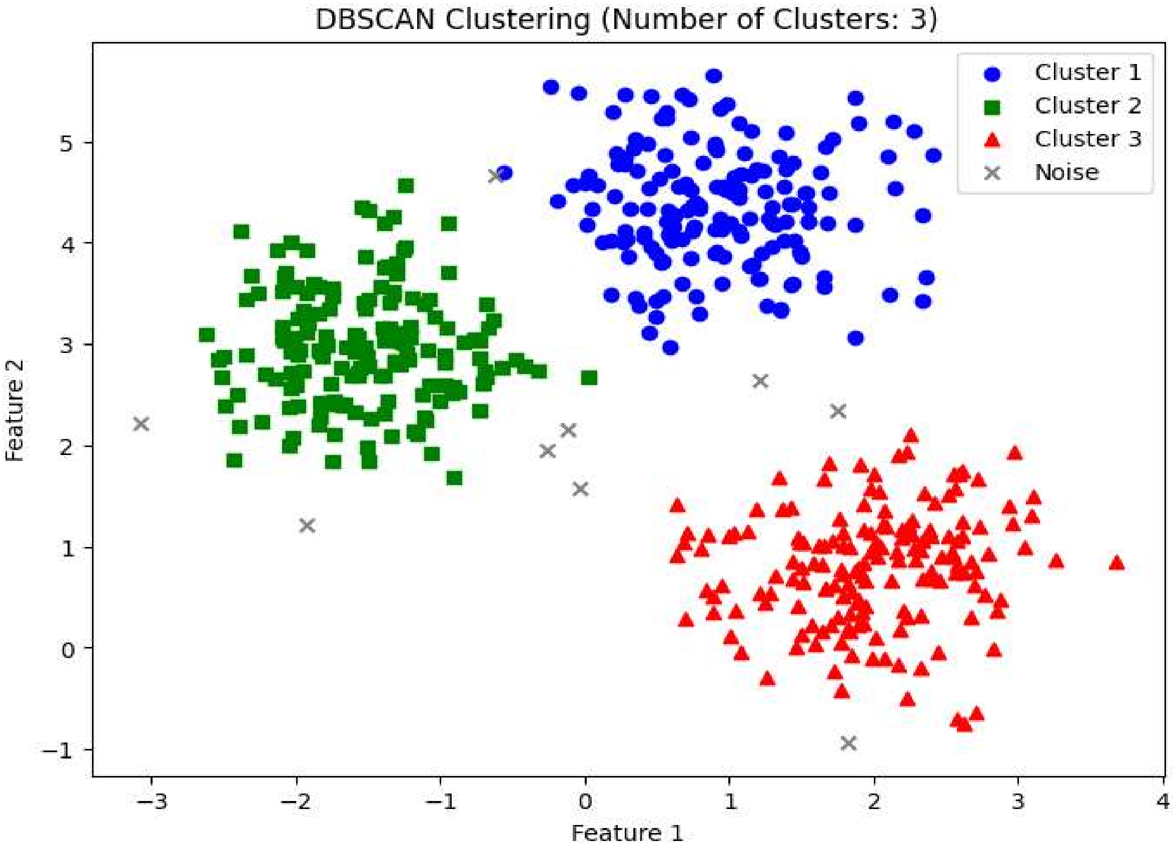
9. Write a program to perform Density Based Clustering with built-in function on the dataset

```
from numpy import unique
from numpy import where
from matplotlib import pyplot
from sklearn.datasets import make_classification
from sklearn.cluster import DBSCAN
# initialize the data set we'll work with
training_data, _ = make_classification(
        n_samples=1000,
        n_features=2,
        n_informative=2,
        n_redundant=0,
        n_clusters_per_class=1,
        random_state=4
)



# define the model
dbscan_model = DBSCAN(eps=0.25, min_samples=9)

# train the model
dbscan_model.fit(training_data)

# assign each data point to a cluster
dbscan_result = dbscan_model.fit_predict(training_data)

# get all of the unique clusters
dbscan_clusters = unique(dbscan_result)

# plot the DBSCAN clusters
for dbscan_cluster in dbscan_clusters:
   # get data points that fall in this cluster
   index = where(dbscan_result == dbscan_cluster)
   # make the plot
   pyplot.scatter(training_data[index, 0], training_data[index, 1])
  # show the DBSCAN plot
pyplot.show()
```



```
#another method
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate sample data with blobs
X,_ = make_blobs(n_samples=500, centers=3, cluster_std=0.6,
random_state=0)

# DBSCAN clustering
db = DBSCAN(eps=0.5, min_samples=5)
y_db = db.fit_predict(X)

# Number of clusters in labels, ignoring noise if present (-1)
n_clusters_ = len(set(y_db)) - (1 if -1 in y_db else 0)

# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X[y_db == 0][:, 0], X[y_db == 0][:, 1], c='blue', marker='o', label='Cluster
1')
plt.scatter(X[y_db == 1][:, 0], X[y_db == 1][:, 1], c='green', marker='s',
label='Cluster 2')
```

```
plt.scatter(X[y_db == 2][:, 0], X[y_db == 2][:, 1], c='red', marker='^', label='Cluster 3')
plt.scatter(X[y_db == -1][:, 0], X[y_db == -1][:, 1], c='gray', marker='x', label='Noise')
plt.legend(loc='best')
plt.title(f"DBSCAN Clustering (Number of Clusters: {n_clusters_})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

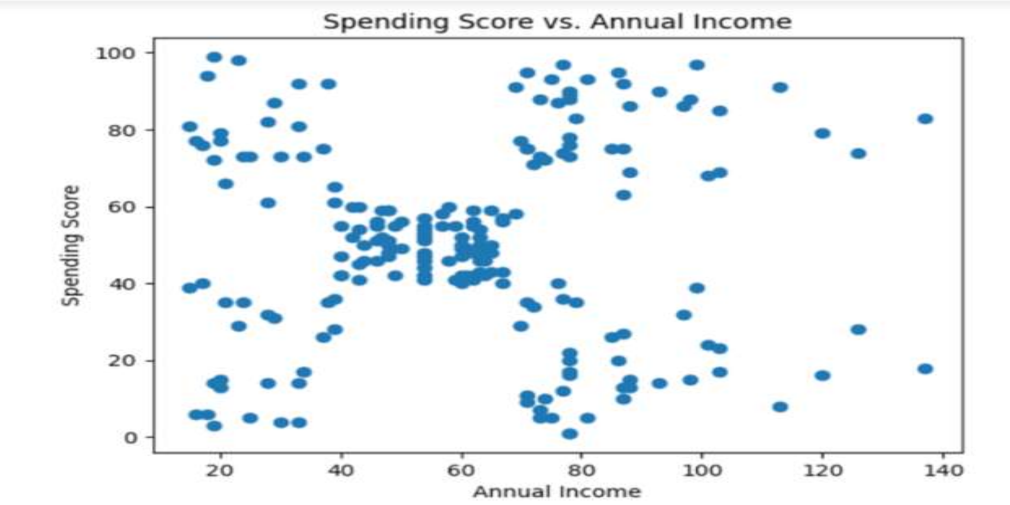


10. Implement K-Means Clustering with built-in function on the dataset

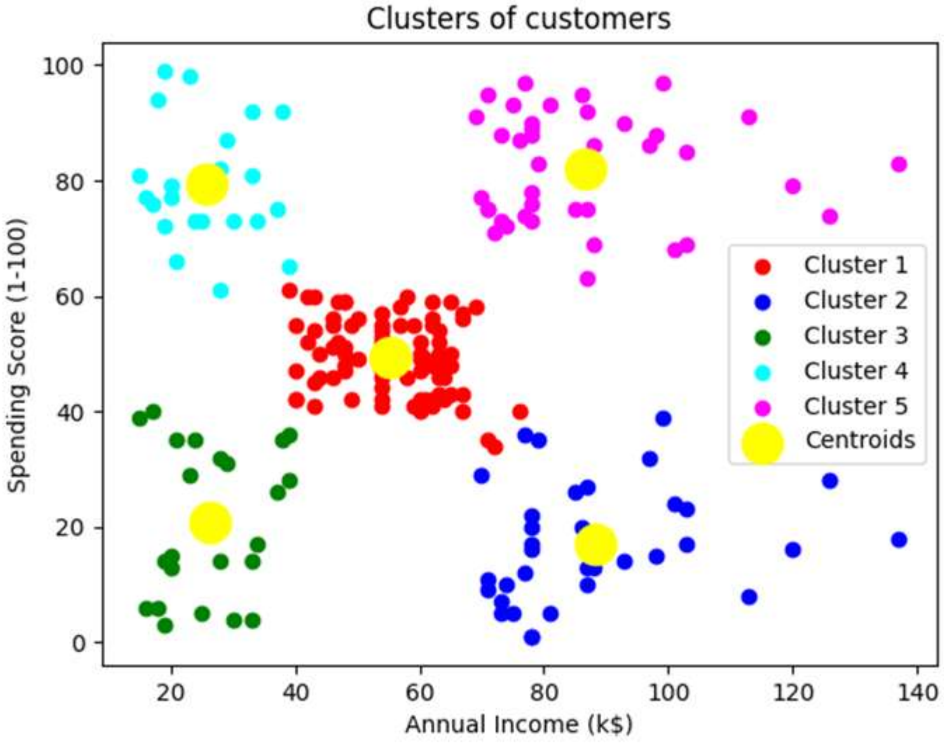
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv(&#39;/content/drive/MyDrive/DATA SCIENCE
LAB/Mall_Customers.csv&#39;)
dataset
```

```
CustomerID Genre Age Annual Income (k$) Spending Score (1-100)

0 1 Male 19 15 39

198 199 Male 32 137 18

199 200 Male 30 137 83

200 rows × 5 columns
```

```
import matplotlib.pyplot as plt

# Example data points
spending_scores =dataset.iloc[:,3].values
annual_incomes = dataset.iloc[:, 4].values

# Plotting the scatter plot
plt.scatter(spending_scores,annual_incomes)
plt.title(&#39;Spending Score vs. Annual Income&#39;)
plt.xlabel(&#39;Annual Income&#39;)
plt.ylabel(&#39;Spending Score&#39;)
plt.show()
```



```
X = dataset.iloc[:, [3, 4]].values

# Fitting K-Means to the dataset
from sklearn.cluster import KMeans
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], c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], c = 'green', label = 'Cluster
3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], c = 'magenta', 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()
```


Clusters of customers
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

11. Implement K-Nearest Neighbour classifier with built-in function on the dataset

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
data_set= pd.read_csv('User_Data.csv')
print(data_set)
```

```
       User ID  Gender  Age  EstimatedSalary  Purchased
0    15624510      Male   19         19000          0
1    15810944      Male   35         20000          0
2    15668575  Female   26         43000      0
3    15603246  Female   27         57000      0
4    15804002      Male   19         76000          0
..       ...       ...  ...          ...       ...
395  15691863  Female   46         41000          1
396  15706071      Male   51         23000          1
397  15654296  Female   50         20000          1
398  15755018      Male   36         33000          0
399  15594041  Female   49         36000          1
[400 rows x 5 columns]
```

```
x= data_set.iloc[:, [2,3]].values

y= data_set.iloc[:, 4].values
```

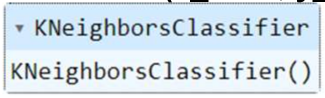
```
# Splitting the dataset into training and test set.
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.15,
random_state=0)
```

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

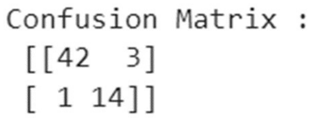
```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```



```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

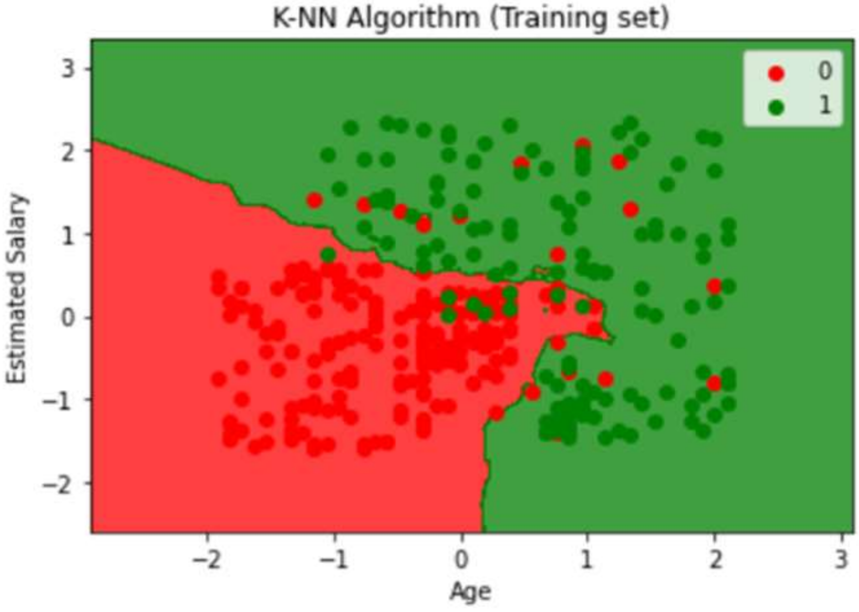
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
print(y_pred)
```

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print('Confusion Matrix : \n',cm)
```

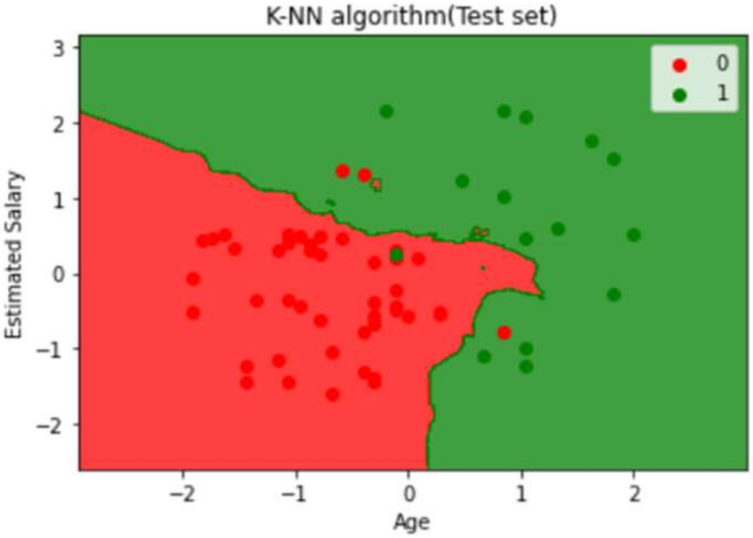


```
Confusion Matrix :
 [[42  3]
 [ 1 14]]
```

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```



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

```
Accuracy :  0.9333333333333333
```