Q]Classify the email using the binary classification method. Email Spam detection has two states:
a) Normal State – Not Spam,
b) Abnormal State – Spam.
Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings ('ignore')
from sklearn.model_selection import train_test_split
from sklearn import metrics

df = pd.read_csv("emails.csv")
df

df.isnull()
df.isnull().sum

df.shape
df.columns

x = df.drop(['Email No.','Prediction'],axis = 1)
y = df['Prediction']

x.shape
y.shape

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scale = scaler.fit_transform(x)
x_scale.shape

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =train_test_split(x_scale,y,test_size=0.25,random_state=0)
x_train.shape
y_train.shape

x_test.shape
y_test.shape
set(x.dtypes)

sns.countplot(x=y)

from sklearn.neighbors import KneighborsClassifier
from sklearn.metrics import ConfusionMatrixDisplay,accuracy_score,classification_report

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
```

```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)

y_test.value_counts()
accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred))

from sklearn.svm import SVC
svm = SVC(kernel = 'sigmoid')
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)
print("svm accuracy = ",accuracy_score(y_test,y_pred))

svm = SVC(kernel = 'linear')
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)
print("svm accuracy = ",accuracy_score(y_test,y_pred))

svm = SVC(kernel = 'rbf')
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)
print("svm accuracy = ",accuracy_score(y_test,y_pred))

svm = SVC(kernel = 'poly')
svm.fit(x_train,y_train)
y_pred = svm.predict(x_test)
print("svm accuracy = ",accuracy_score(y_test,y_pred))
```

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix (5 points).

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings warnings.filterwarnings('ignore')
from sklearn.model_selection
import train_test_split from sklearn import metrics
df=pd.read_csv("C:/Users/Pratibha/Downloads/archive.zip")
df.head()

df.shape
df.columns

x=df[['CreditScore','Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard','IsActiveMember',
'EstimatedSalary']]
y=df ['Exited']

sns.countplot(x=y);
y.value_counts()

from sklearn.preprocessing import StandardScaler
Scaler =StandardScaler()
x_scaled=Scaler.fit_transform(x)
x_scaled

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =train_test_split(x_scale,y,test_size=0.25,random_state=0)
x_train.shape
y_train.shape

x_test.shape
y_test.shape

from sklearn.neural_network import MLPClassifier
ann= MLPClassifier(hidden_layer_sizes=(100,100,100),random_state = 0,
max_iter=100,activation='relu')
ann.fit(x_train,y_train)
y_pred = ann.predict(x_test)
```

```
from sklearn.metrics import ConfusionMatrixDisplay,classification_report,accuracy_score
y_test.value_counts()

ConfusionMatrixDisplay.from_predictions(y_test,y_pred)


accuracy_score(y_test,y_pred)
print(classification_report(y_test,y_pred))

!pip install imbalanced-learn

from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler (random_state =0)
x_res,y_res = ros.fit_resample(x,y)
y_res.value_counts()

from sklearn.preprocessing import StandardScaler Scaler =StandardScaler()
x_scaled=Scaler.fit_transform(x_res) x_scaled

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =train_test_split(x_scaled,y_res,random_state=0,test_size=0.25)
x_res.shape

from sklearn.neural_network import MLPClassifier
ann= MLPClassifier(hidden_layer_sizes=(100,100,100),random_state = 0,
max_iter=100,activation='relu')
ann.fit(x_train,y_train)

y_pred = ann.predict(x_test)
from sklearn.metrics import ConfusionMatrixDisplay,classification_report,accuracy_score
y_test.value_counts()

ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
```

Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2

```
cur_x = 3
rate = 0.01
precision=0.00001
previous_step_size = 1
max_iters = 10000
iters = 0
gf = lambda x: (x+5) **2

while previous_ step_size > precision and iters < max_iters:
prev_x = cur_x
cur_x = cur_x – rate*gf(prev_x)
previous_step_size = abs(cur_x – prev_x)
iters = iters + 1
print(“Iteration”,iters,”\nx value is”,cur_x)
print(“The local minima occurs at “,cur_x)

cur_x = 2
rate = 0.01
precision=0.00001
previous_step_size = 1
max_iters = 10000
iters = 0
gf = lambda x: (x+3) **2

import matplotlib.pyplot as plt
gd = []

while previous_ step_size > precision and iters < max_iters:
prev_x = cur_x
cur_x = cur_x – rate*gf(prev_x)
previous_step_size = abs(cur_x – prev_x)
iters = iters + 1
print(“Iteration”,iters,”\nx value is”,cur_x)
print(“The local minima occurs at “,cur_x)
gd.append(cur_x)

print(“Local Minima”,cur_x)

plt.plot(range(9975),gd)
```