**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)