Practical 2 :- Email Spam Detection

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
data = pd.read_csv("/home/admin1/Desktop/Anurag/emails.csv")
data.head(10)
data.tail(10)
data.dtypes
data.describe()
#Selecting number of columns from the dataset
X = data.iloc[:,1:3001]
X
Y = data.iloc[:,-1].values
Y
#Perform the train test model from SKLearn Library
train_x,test_x,train_y,test_y = train_test_split(X,Y,test_size = 0.25)
svc = SVC(C=1.0,kernel='rbf',gamma='auto')
# C here is the regularization parameter. Here, L2 penalty is used(default). It is the
inverse of the strength of regularization.
# As C increases, model overfits.
# Kernel here is the radial basis function kernel.
# gamma (only used for rbf kernel) : As gamma increases, model overfits.
svc.fit(train_x,train_y)
y_pred2 = svc.predict(test_x)
print("Accuracy Score for SVC : ", accuracy_score(y_pred2,test_y))
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
print(knn.predict(X_test))
print(knn.score(X_test, y_test))
```

Practical 5 :- K-Nearest Neighbors Algorithm on diabetes.csv Dataset

```
import numpy as np
import pandas as pd
data = pd.read_csv('/home/admin1/Anurag/diabetes.csv')
data.head(5)
data.tail(5)
data.describe()
data.shape
data.boxplot()
data.isnull().sum()
for column in data.columns[1:-3]: data[column].replace(0, np.NaN, inplace = True)
data[column].fillna(round(data[column].mean(skipna=True)), inplace =
True)data.head(10)
data.hist()
X = data.iloc[:, :8] #Features
Y = data.iloc[:, 8:] #Predictor
#Perform Spliting
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, random_state=0)
#Execution of K-Nearest Neighbor
from sklearn.neighbors import KNeighborsClassifierknn = KNeighborsClassifier()
knn_fit = knn.fit(X_train, Y_train.values.ravel())
knn_pred = knn_fit.predict(X_test)
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score,
accuracy_score
print("Confusion Matrix")
print(confusion_matrix(Y_test, knn_pred))
print("Accuracy Score:", accuracy_score(Y_test, knn_pred))
print("Recall Score:", recall_score(Y_test, knn_pred))
print("F1 Score:", f1_score(Y_test, knn_pred))
print("Precision Score:",precision_score(Y_test, knn_pred))
```

Practical 6 :- K-Means on Sales Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
data = pd.read_csv('/home/admin1/Anurag/sales.csv',encoding='unicode_escape')
data.head(10)
data.tail(10)
data.dtypes
data.isnull().sum()
data.describe()
data['ORDERDATE'] = pd.to_datetime(data['ORDERDATE'])
#We need to create some features in order to create cluseters
#Recency: Number of days between customer's latest order and today's date
#Frequency : Number of purchases by the customers
#Monetary : Revenue generated by the customers
import datetime as dt
snapshot_date = data['ORDERDATE'].max() + dt.timedelta(days = 1)
data_RFM = data.groupby(['CUSTOMERNAME']).agg({
'ORDERDATE' : lambda x : (snapshot_date - x.max()).days,
'ORDERNUMBER' : 'count',
'SALES' : 'sum'
})
data_RFM.rename(columns = {
   'ORDERDATE' : 'Recency',
   'ORDERNUMBER' : 'Frequency',
   'SALES' : 'Monetary'
}, inplace=True)
```

```
#We create 4 quartile ranges
data_RFM['M'] = pd.qcut(data_RFM['Monetary'], q = 4, labels = range(1,5))
data_RFM['R'] = pd.qcut(data_RFM['Recency'], q = 4, labels = list(range(4,0,-1)))
data_RFM['F'] = pd.qcut(data_RFM['Frequency'], q = 4, labels = range(1,5))
data_RFM['RFM_Score'] = data_RFM[['R', 'M', 'F']].sum(axis=1)
data_RFM.head()
#RFM scores will display the following values:-
#RFM Score > 10 : High Value Customers
#RFM Score < 10 and RFM Score >= 6 : Mid Value Customers
#RFM Score < 6 : Low Value Customers
def rfm_level(data):
    if bool(data['RFM_Score'] >= 10):
        return 'High Value Customer'
    elif bool(data['RFM_Score'] < 10) and bool(data['RFM_Score'] >= 6):
        return 'Mid Value Customer'
    else:
        return 'Low Value Customer'
data_RFM['RFM_Level'] = data_RFM.apply(rfm_level, axis = 1)
#We display the newly created table
data_RFM.head()
#Now we will be executing KMeans
data = data_RFM[['Recency', 'Frequency', 'Monetary']]
data.head()
#Our data is not accurate so we must remove the unwanted things by performing log
transformation
data_log = np.log(data)
data_log.head()
#Now we will perform standardization
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
scaler.fit(data_log)
data_normalized = scaler.transform(data_log)
data_normalized = pd.DataFrame(data_normalized, index = data_log.index,
columns=data_log.columns)
data_normalized.describe().round(2)
#Fit KMeans and use elbow method to choose the number of clusters
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
sse = {}
for k in range(1, 21):
   kmeans = KMeans(n_clusters = k, random_state = 1)
   kmeans.fit(data_normalized)
   sse[k] = kmeans.inertia_
#Now we will plot graph using Elbow Method
plt.figure(figsize=(10,6))
plt.title('The Elbow Method')
plt.xlabel('K')
plt.ylabel('SSE')
plt.style.use('ggplot')
sns.pointplot(x=list(sse.keys()), y = list(sse.values()))
plt.text(4.5, 60, "Largest Angle", bbox = dict(facecolor = 'lightgreen', alpha = 0.5))
plt.show()
#We will make the cluster of size 5
kmeans = KMeans(n_clusters=5, random_state=1)
kmeans.fit(data_normalized)
cluster_labels = kmeans.labels_
data_rfm = data.assign(Cluster = cluster_labels)
```

```
data_rfm.head()
from sklearn.cluster import KMeans
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15,5))
sns.scatterplot(ax=axes[0], data=data, x='Recency', y='Frequency').set_title('Without
cliustering')
sns.scatterplot(ax=axes[1], data=data, x='Recency', y='Frequency',
hue=kmeans.labels_).set_title('With the Elbow method')
sns.scatterplot(ax=axes[2], data=data, x='Recency', y='Frequency',
hue=kmeans.labels_).set_title('With the Elbow method and scaled data');
```