Assi – 1

```
1) import pandas as pd
2) import numpy as np
3) import matplotlib.pyplot as plt
4) import seaborn as sns
5) df = pd.read_csv("C:\\Users\\sspm-engg\\Downloads\\archive\\uber.csv")
6) df
7) df.head()
8) df.tail()
9) df.info()
10) df.columns
11) df=df.drop(['Unnamed: 0', 'key'],axis=1)
12) df
13) df.head()
14) df.shape
15) df.dtypes
16) df.describe()
17) df.isnull().sum()
18) df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
19) df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)
20) df.isna().sum()
21) df.dtypes
22) df.pickup_datetime = pd.to_datetime(df.pickup_datetime,errors='coerce')
23) df.dtypes
24) df=df.assign(hour = df.pickup_datetime.dt.hour, day = df.pickup_datetime.dt.day, month =
    df.pickup_datetime.dt.month, year = df.pickup_datetime.dt.year, dayofweek =
    df.pickup_datetime.dt.dayofweek)
25) df.head()
26) df.drop('pickup_datetime',axis=1)
27) from math import*
28) from math import radians, sin, cos, asin, sqrt
```

29) def distance_formula(longitude1, latitude1, longitude2, latitude2):

travel_dist = []

for i in range(len(longitude1)):

Convert degrees to radians

lon1, lat1 = radians(longitude1[i]), radians(latitude1[i])

lon2, lat2 = radians(longitude2[i]), radians(latitude2[i])

Haversine formula

dlon = lon2 - lon1

dlat = lat2 - lat1

a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2

c = 2 * asin(sqrt(a))

distance_km = 6371 * c # 6371 km = Earth's radius

travel_dist.append(distance_km)

return travel_dist

30) df['dist_travel_km'] = distance_formula(df['pickup_longitude'].to_numpy(), df['pickup_latitude'].to_numpy(), df['dropoff_longitude'].to_numpy(), df['dropoff_latitude'].to_numpy())

31) df.dtypes

32) df.plot(kind="box", subplots=True, layout=(6, 2), figsize=(15, 20))

plt.show()

33) def remove_outlier(df1, col):

Q1 = df1[col].quantile(0.25)

Q3 = df1[col].quantile(0.75)

IQR = Q3 - Q1 # interquartile range

lower_whisker = Q1 - 1.5 * IQR

upper_whisker = Q3 + 1.5 * IQR

df1[col] = np.clip(df1[col], lower_whisker, upper_whisker)

return df1

34) def treat_outliers_all(df1,col_list):

for c in col_list:

df1 = remove_outlier(df1, c)

return df1

```
35) numeric_cols = df.select_dtypes(include=['number']).columns
36) df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))
37) corr = df.corr()
38) corr
39) fig, ax = plt.subplots(figsize=(10, 6))
40) sns.heatmap(df.corr(), annot=True, cmap='coolwarm', ax=ax)
41) plt.show()
42) y = df['fare_amount']
43) x = df[['pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
  'dropoff_latitude', 'passenger_count']]
44) from sklearn.model_selection import train_test_split
x = x.fillna(x.mean())
y = y.fillna(y.mean())
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)
45) df.head()
46) from sklearn.linear_model import LinearRegression
regression = LinearRegression()
47) regression.fit(x_train,y_train)
48) regression.intercept_
49) regression.coef_
50) prediction = regression.predict(x_test)
print(prediction)
51) y_test
52) from sklearn.metrics import r2_score
53) r2_score(y_test,prediction)
54) from sklearn.metrics import mean_squared_error
55) MSE = mean_squared_error(y_test,prediction)
56) MSE
57) RMSE = np.sqrt(MSE)
58) RMSE
59) from sklearn.ensemble import RandomForestRegressor
```

60) rf = RandomForestRegressor(n_estimators=100)
61) rf.fit(x_train,y_train)
62) y_pred = rf.predict(x_test)
63) y_pred
64) R2_Random = r2_score(y_test,y_pred)
65) R2_Random
66) MSE_Random = mean_squared_error(y_test,y_pred)
67) MSE_Random
68) RMSE_Random = np. sqrt(MSE_Random)
69) RMSE_Random

Assi – 2

1) import pandas as pd
   import numpy as np
   from sklearn.svm import SVC
   from sklearn.neighbors import KNeighborsClassifier
   from sklearn import metrics
2) df=pd.read_csv("C:\\Users\\sspm-engg\\Desktop\\emails.csv")
3) df
4) df.head()
5) df.shape
6) df.columns
7) df.dtypes
8) df.describe()
9) s=df.isnull().sum()
   s
10) s[s>0]
11) df.isnull().any()
12) df.dropna (inplace = True)

```
13) df.drop(['Email No.'],axis=1,inplace=True)
14) df.head()
15) df.columns
16) df.Prediction.unique()
17) df['Prediction']=df['Prediction'].replace({0:'Not spam',1:'spam'})
18) df
19) x=df.drop(columns='Prediction',axis=1)
    y=df['Prediction']
20) x.columns
21) y.head()
22) from sklearn.preprocessing import scale
    x=scale(x)
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
23) from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=7)
24) knn.fit(x_train,y_train) y_pred=knn.predict(x_test)
25) print("prediction:\n")
    print(y_pred)
26) M = metrics.accuracy_score(y_test, y_pred)
    print("KNN accuracy:", M)
27) C=metrics.confusion_matrix(y_test,y_pred)
    print("Confusion matrix:",C)
28) model=SVC(C=1)
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
29) metrics.confusion_matrix(y_true=y_test,y_pred=y_pred)
30) print("SVM accuracy=",metrics.accuracy_score(y_test,y_pred))
```

Assi- 3

1) import pandas as pd
   import numpy as np
   from sklearn.preprocessing import StandardScaler
   from sklearn.neighbors import KNeighborsClassifier
   from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
2) df=pd.read_csv("C:\\Users\\Admin\\Downloads\\diabetes.csv")
3) df
4) df.head()
5) df.info()
6) df.shape
7) df.columns
8) df.describe()
9) df.isna().sum()
10) x=df.drop('Outcome',axis=1)
11) y=df['Outcome']
12) df.head()
13) from sklearn.model_selection import train_test_split
    x_train, x_test, y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
14) scaler=StandardScaler()
    x_train_scaled=scaler.fit_transform(x_train)
    x_test_scaled=scaler.transform(x_test)
15) k=5
    Knn=KNeighborsClassifier(n_neighbors=k)
    Knn.fit(x_train_scaled,y_train)
16) y_pred=Knn.predict(x_test_scaled)
17) conf_matrix=confusion_matrix(y_test,y_pred)
    accuracy=accuracy_score(y_test,y_pred)
    error_rate=1-accuracy
    precision=precision_score(y_test,y_pred)
    recall=recall_score(y_test,y_pred)
18) print("Confusion Matrix:")
    print(conf_matrix)

```
19) print("Accuracy:", accuracy)
20) print("Error Rate:",error_rate)
21) print("Precision:",precision)
22) print("Recall:",recall)
```

Assi – 4

```
1) import pandas as pd
   import numpy as np
   import matplotlib.pyplot as plt
   import seaborn as sns
   from sklearn.cluster import KMeans,k_means
   from sklearn.decomposition import PCA
2) df=pd.read_csv("C:\\Users\\Admin\\Downloads\\sales_data_sample.csv",encoding='latin1')
3) df.head()
4) df.shape
5) df.describe()
6) df.info()
7) df.isnull().sum()
8) df.dtypes
   df=df.drop(['ADDRESSLINE1','ADDRESSLINE2','STATUS','POSTALCODE','CITY','TERRITORY','STAT
   E'],axis=1, errors='ignore')
9) df.isnull().sum()
10) df.dtypes
11) df['COUNTRY'].unique()
12) df['PRODUCTLINE'].unique()
13) df['DEALSIZE'].unique()
14) productline=pd.get_dummies(df['PRODUCTLINE'])
    Dealsize=pd.get_dummies(df['DEALSIZE'])
15) df=pd.concat([df,productline,Dealsize],axis=1)
16) df_drop=['COUNTRY','PRODUCTLINE','DEALSIZE']
```

```
    df=df.drop(df_drop,axis=1)
17) df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes
18) df.drop('ORDERDATE', axis=1, inplace=True, errors='ignore')
19) df.dtypes
20) X = pd.get_dummies(df, drop_first=True)
21) distortions = []
    K = range(1, 11)
    for k in K:
    kmeanModel = KMeans(n_clusters=k, random_state=42)
    kmeanModel.fit(X)
    distortions.append(kmeanModel.inertia_)
22) plt.figure(figsize=(16,8))
    plt.plot(K, distortions, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Distortion')
    plt.title('The Elbow Method showing the optimal k')
    plt.show()
23) X_train = df.values
24) X_train.shape
25) import pandas as pd
    from sklearn.model_selection import train_test_split
    X = df.select_dtypes(include=['number'])
    X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
    from sklearn.cluster import KMeans
26) model = KMeans(n_clusters=3, random_state=2)
    model = model.fit(X_train)
    predictions = model.predict(X_train)
27) unique,counts = np.unique(predictions,return_counts=True)
28) counts = counts.reshape(1,3)
29) counts_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3'])
30) counts_df.head()
31) pca = PCA(n_components=2)
32) reduced_X = pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2'])
33) reduced_X.head()
```

```
34) plt.figure(figsize=(14,10))
    plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
35) model.cluster_centers_
36) reduced_centers = pca.transform(model.cluster_centers_)
37) reduced_centers
38) plt.figure(figsize=(14,10))
    plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
    plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300)
39) reduced_X['Clusters'] = predictions
40) reduced_X.head()
41) plt.figure(figsize=(14,10))
    plt.scatter(reduced_X[reduced_X['Clusters'] ==
    0].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] == 0].loc[:,'PCA2'],color='slateblue')
    plt.scatter(reduced_X[reduced_X['Clusters'] ==
    1].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] == 1].loc[:,'PCA2'],color='springgreen')
    plt.scatter(reduced_X[reduced_X['Clusters'] ==
    2].loc[:,'PCA1'],reduced_X[reduced_X['Clusters'] == 2].loc[:,'PCA2'],color='indigo')
    plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300)
```