1)Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

1. **Pre-process the dataset.**
2. **Identify outliers.**
3. **Implement linear regression**
5. **Evaluate the model using scores like R2, RMSE, etc.**

```
#import library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#importing the dataset
df = pd.read_csv("uber.csv")

# 1. Pre-process the dataset.
df.head()
df.info() #To get the required information of the dataset
df.columns #TO get number of columns in the dataset
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as it isn't required
df.shape #To get the total (Rows,Columns)
df.dtypes #To get the type of each colum
df.info()
df.describe() #To get statistics of each columns

# Filling Missing values
df.isnull().sum()
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)
df.isnull().sum()
df.dtypes

# Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
df.dtypes
# To segregate each time of date and time
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)
df.head()
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column
df = df.drop('pickup_datetime',axis=1)
df.head()
df.dtypes

#Checking outliers and filling them
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to check t he outliers
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1
def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1
df = treat_outliers_all(df , df.iloc[: , 0::])
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot shows that dataset is free from outliers
#Uber doesn't travel over 130 kms so minimize the distance
df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
```

```
print("Remaining observastions in the dataset:", df.shape)
#Finding inccorect latitude (Less than or greater than 90) and longitude (greater than or less than 180)
incorrect_coordinates =
df.loc[(df.pickup_latitude > 90)
|(df.pickup_latitude < -90) |
(df.dropoff_latitude > 90)
|(df.dropoff_latitude < - 90) |
(df.pickup_longitude > 180)
|(df.pickup_longitude < -180) |
(df.dropoff_longitude > 90)
|(df.dropoff_longitude < -90) ]
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
df.head()
df.isnull().sum()
sns.heatmap(df.isnull()) #Free for null values
corr = df.corr() #Function to find the correlation
corr
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True)
#Correlation Heatmap (Light values means highly corr elated)

# Dividing the dataset into feature and target values
x =
df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude','passenger_count','hour','day','month','year','dayofweek','dist_travel_km']]
y = df['fare_amount']

#Dividing the dataset into training and testing dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(x,y,test_size = 0.33)

# Linear Regression
from sklearn.linear_model import
LinearRegression regression =
LinearRegression()
regression.fit(X_train,y_train)
regression.intercept_ #To find the linear intercept
regression.coef_ #To find the linear coeeficient
prediction = regression.predict(X_test) #To predict the target values
print(prediction)
y_test

# Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error
from sklearn.metrics import r2_score
r2_score(y_test,prediction)
from sklearn.metrics import mean_squared_error
MSE =
mean_squared_error(y_test,prediction)
MSE
RMSE = np.sqrt(MSE)
RMSE

#Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
rf =
RandomForestRegressor(n_estimators=100)
#Here n_estimators means number of trees you want to build before making the prediction
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
y_pred

#Metrics evaluatin for Random Forest
R2_Random = r2_score(y_test,y_pred)
R2_Random
MSE_Random =
mean_squared_error(y_test,y_pred)
MSE_Random
RMSE_Random = np.sqrt(MSE_Random)
RMSE_Random
```

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.svm import SVC
from sklearn import metrics
df=pd.read_csv('emails.csv')
df.head()
df.columns
df.isnull().sum()
df.dropna(inplace = True)
df.drop(['Email No.'],axis=1,inplace=True) X =
df.drop(['Prediction'],axis = 1) y =
df['Prediction']
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size = 0.3,
random_state = 42)
from sklearn.neighbors import
KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Prediction",y_pred)
print("KNN accuracy =
",metrics.accuracy_score(y_test,y_pred))
print("Confusion
matrix",metrics.confusion_matrix(y_test,y_pr
ed))
# cost C = 1
model = SVC(C = 1)
# fit
model.fit(X_train, y_train)
# predict
y_pred = model.predict(X_test)
metrics.confusion_matrix(y_true=y_test,
y_pred=y_pred)
print("SVM accuracy =
",metrics.accuracy_score(y_test,y_pred))
```

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

```
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.svm import SVC
from sklearn import metrics
df=pd.read_csv('diabetes.csv')
df.columns

# Check for null values. If present remove null
values from the dataset
df.isnull().sum()

# Outcome is the label/target, other columns
are features
X = df.drop('Outcome',axis = 1)
y = df['Outcome']
from sklearn.preprocessing import scale
X = scale(X)

# split into train and test
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size = 0.3,
random_state = 42)
```

```
from sklearn.neighbors import
KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("Confusion matrix: ")

cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)

print("Acccuracy
",metrics.accuracy_score(y_test,y_pred))

total_misclassified = cs[0,1] + cs[1,0]
print(total_misclassified)

total_examples =
cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]
print(total_examples)

print("Error
rate",total_misclassified/total_examples)
print("Error rate ",1-
metrics.accuracy_score(y_test,y_pred))
print("Precision
score",metrics.precision_score(y_test,y_pred)
)
print("Recall score
",metrics.recall_score(y_test,y_pred))
print("Classification report
",metrics.classification_report(y_test,y_pred))
```

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method and visualize clusters.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Importing the required libraries.

from sklearn.cluster import KMeans, k_means
#For clustering
from sklearn.decomposition import PCA
#Linear Dimensionality reduction.
df = pd.read_csv("sales_data_sample.csv")
#Loading the dataset.
df.head()
df.shape
df.describe()
df.info()
df.isnull().sum()
df.dtypes
#Dropping the categorical uneccessary
columns along with c olumns having null
values. Can't fill the null values are there are
alot of null values.
df_drop = ['ADDRESSLINE1', 'ADDRESSLINE2',
'STATUS','POSTALCODE', 'CITY', 'TERRITORY',
'PHONE', 'STATE', 'CONTACTFIRSTNAME',
'CONTACTLASTNAME', 'CUSTOMERNAME',
'ORDERNUMBER'] df = df.drop(df_drop,
axis=1)
df.isnull().sum()
df.dtypes
# Checking the categorical columns.
df['COUNTRY'].unique()
df['PRODUCTLINE'].unique()

df['DEALSIZE'].unique()
productline =
pd.get_dummies(df['PRODUCTLINE'])
#Converting the categorical columns. Dealsize
= pd.get_dummies(df['DEALSIZE'])
df = pd.concat([df,productline,Dealsize], axis =
1)

df_drop =
['COUNTRY','PRODUCTLINE','DEALSIZE']
#Dropping Country too as there are alot o f
countries.
df = df.drop(df_drop, axis=1)

df['PRODUCTCODE'] =
pd.Categorical(df['PRODUCTCODE']).codes
#Converting the datatype.
```

df.drop('ORDERDATE', axis=1, inplace=True) #Dropping the Orderdate as Month is already in cluded.

df.dtypes #All the datatypes are converted into numeric
Plotting the Elbow Plot to determine the number of clusters.

distortions = [] # Within Cluster Sum of Squares from the centroid
K = range(1,10)
	for k in K:
		kmeanModel = KMeans(n_clusters=k)
		kmeanModel.fit(df)
		distortions.append(kmeanModel.inertia_)
#Appeding the intertia to the Distortions

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()
# As the number of k increases Inertia decreases.
#Observations: A Elbow can be observed at 3 and after that the curve decreases gradually.
X_train = df.values #Returns a numpy array.
X_train.shape

model = KMeans(n_clusters=3,random_state=2)
#Number of cluster = 3
model = model.fit(X_train) #Fitting the values to create a model.
predictions = model.predict(X_train)

#Predicting the cluster values (0,1,or 2)
unique,counts = np.unique(predictions,return_counts=True)
counts = counts.reshape(1,3)

counts_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3'])
counts_df.head()

# Visualization
pca = PCA(n_components=2)
#Converting all the features into 2 columns to make it easy to visualize using Principal COmponent Analysis.

reduced_X = pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2']) #Creating a DataFrame.

reduced_X.head()

#Plotting the normal Scatter Plot
plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
model.cluster_centers_ #Finding the centriods. (3 Centriods in total. Each Array contains a centroids for particular feature)

reduced_centers = pca.transform(model.cluster_centers_)
#Transforming the centroids into 3 in x and y coordinates

reduced_centers

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)
#Pl otting the centroids
reduced_X['Clusters'] = predictions
 #Adding the Clusters to the reduced dataframe.
reduced_X.head()

#Plotting the clusters
plt.figure(figsize=(14,10))

# taking the cluster number and first column taking the sa me cluster number and second column Assigning the color
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)