#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. Check the correlation.
4. Implement linear regression and random forest regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)

In [71]:
```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [72]:
```
#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

In [73]:
```
df.head()
```

...

In [74]:
```
df.info() #To get the required information of the dataset
```

...

In [75]:
```
df.columns #TO get number of columns in the dataset
```

...

In [76]:
```
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as it isn't
```

In [77]:
```
df.head()
```

...

In [78]:
```
df.shape #To get the total (Rows,Columns)
```

...

Scanned with CamScanner

In [79]:
```
df.dtypes #To get the type of each column
```

...

In [80]:
```
df.info()
```

...

In [81]:
```
df.describe() #To get statistics of each columns
```

...

Filling Missing values

In [82]:
```
df.isnull().sum()
```

...

In [83]:
```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = T
```

In [84]:
```
df.isnull().sum()
```

...

In [85]:
```
df.dtypes
```

...

Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format

In [86]:
```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```

In [87]:
```
df.dtypes
```

...

To segregate each time of date and time

In [88]:
```
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)
```

In [89]:
```
df.head()
```

...

Scanned with CamScanner

In [90]:
```
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

df = df.drop('pickup_datetime',axis=1)
```

In [91]:
```
df.head()
```

...

In [92]:
```
df.dtypes
```

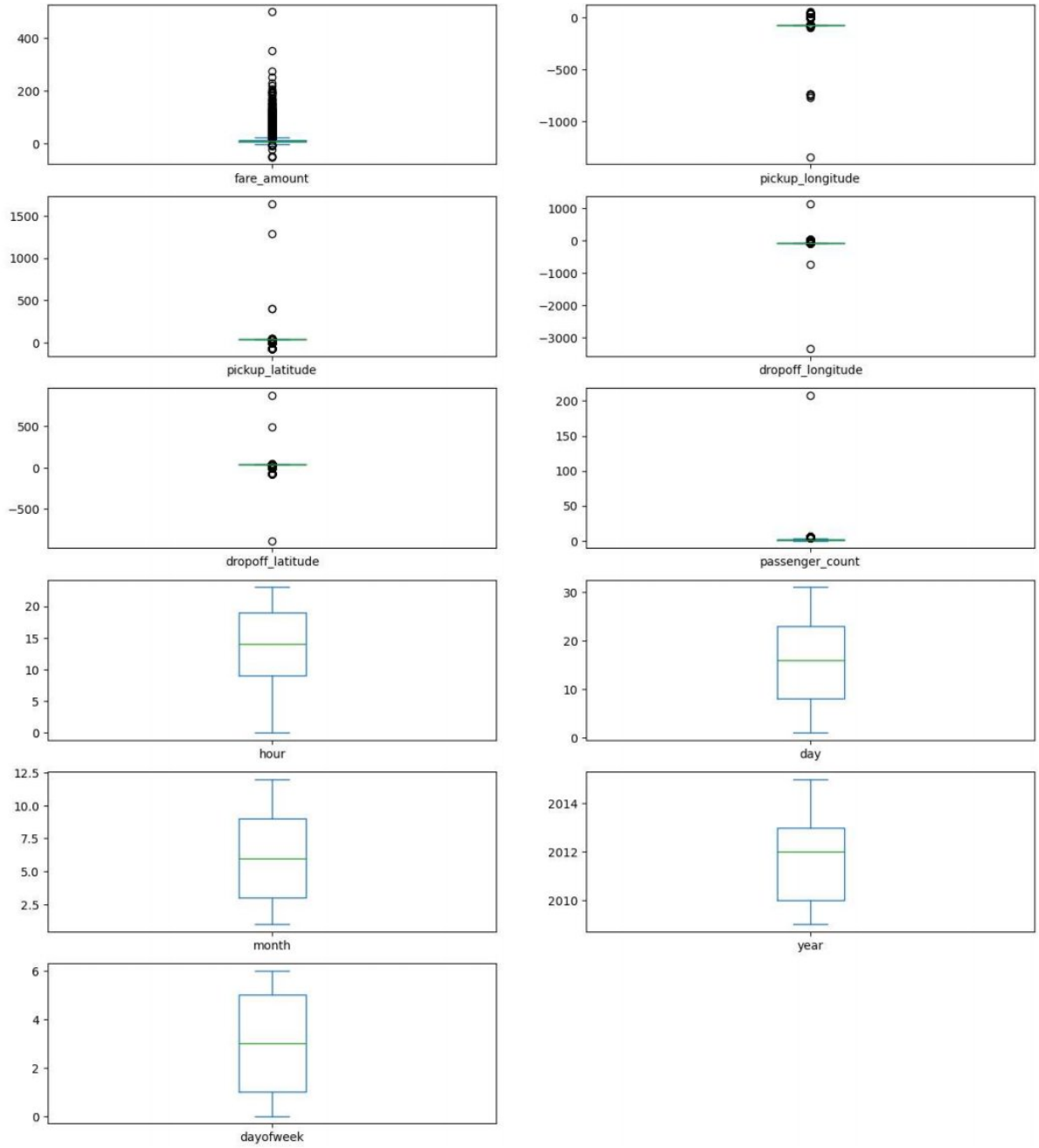
...

Checking outliers and filling them

Scanned with CamScanner

In [93]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to
```

Out[93]:
```
fare_amount            AxesSubplot(0.125,0.786098;0.352273x0.0939024)
pickup_longitude     AxesSubplot(0.547727,0.786098;0.352273x0.0939024)
pickup_latitude        AxesSubplot(0.125,0.673415;0.352273x0.0939024)
dropoff_longitude    AxesSubplot(0.547727,0.673415;0.352273x0.0939024)
dropoff_latitude       AxesSubplot(0.125,0.560732;0.352273x0.0939024)
passenger_count      AxesSubplot(0.547727,0.560732;0.352273x0.0939024)
hour                   AxesSubplot(0.125,0.448049;0.352273x0.0939024)
day                  AxesSubplot(0.547727,0.448049;0.352273x0.0939024)
month                  AxesSubplot(0.125,0.335366;0.352273x0.0939024)
year                 AxesSubplot(0.547727,0.335366;0.352273x0.0939024)
dayofweek              AxesSubplot(0.125,0.222683;0.352273x0.0939024)
dtype: object
```



Scanned with CamScanner

In [94]:
```
#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
```

In [95]:
```
df = treat_outliers_all(df , df.iloc[: , 0::])
```


Scanned with CamScanner

In [96]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot sho
```

...

In [97]:
```
pip install haversine
```

```
Requirement already satisfied: haversine in d:\anaconda\lib\site-packages (2.7.
0)
Note: you may need to restart the kernel to use updated packages.
```

In [98]:
```
import haversine as hs  #Calculate the distance using Haversine to calculate the
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pickup_latitud
        loc1=(lati1,long1)
        loc2=(lati2,long2)
        c = hs.haversine(loc1,loc2)
        travel_dist.append(c)

print(travel_dist)
df['dist_travel_km'] = travel_dist
df.head()
```

...

In [99]:
```
#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)
```

```
Remaining observastions in the dataset: (200000, 12)
```

In [100]:
```
#Finding inccorect latitude (Less than or greater than 90) and longitude (greater
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latitude < -
                                (df.dropoff_latitude > 90) |(df.dropoff_latitu
                                (df.pickup_longitude > 180) |(df.pickup_longit
                                (df.dropoff_longitude > 90) |(df.dropoff_longi
                                 ]
```

In [101]:
```
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

In [102]:
```
df.head()
```

...

In [103]:
```
df.isnull().sum()
```

...

In [104]:
```
sns.heatmap(df.isnull()) #Free for null values
```

...

Scanned with CamScanner

In [105]:
```
corr = df.corr() #Function to find the correlation
```

In [106]:
```
corr
```

...

In [107]:
```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means high
```

...

Dividing the dataset into feature and target values

In [108]:
```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitu
```

In [109]:
```
y = df['fare_amount']
```

Dividing the dataset into training and testing dataset

In [110]:
```
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

In [111]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [112]:
```
regression.fit(X_train,y_train)
```

...

In [113]:
```
regression.intercept_ #To find the linear intercept
```

...

In [114]:
```
regression.coef_ #To find the linear coeeficient
```

...

In [115]:
```
prediction = regression.predict(X_test) #To predict the target values
```

In [116]:
```
print(prediction)
```

```
[ 5.01523896  4.90594076 10.69941484 ... 13.36805853  7.0729064
  6.40670012]
```


Scanned with CamScanner

In [117]:
```
y_test
```

...

Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

In [118]:
```
from sklearn.metrics import r2_score
```

In [119]:
```
r2_score(y_test,prediction)
```

Out[119]:
```
0.662655860609985
```

In [120]:
```
from sklearn.metrics import mean_squared_error
```

In [121]:
```
MSE = mean_squared_error(y_test,prediction)
```

In [122]:
```
MSE
```

Out[122]:
```
10.02348655600728
```

In [123]:
```
RMSE = np.sqrt(MSE)
```

In [124]:
```
RMSE
```

Out[124]:
```
3.1659890328311753
```

Random Forest Regression

In [125]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [126]:
```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of t
```

In [127]:
```
rf.fit(X_train,y_train)
```

Out[127]:
```
RandomForestRegressor()
```

In [128]:
```
y_pred = rf.predict(X_test)
```

In [129]:
```
y_pred
```

Out[129]:
```
array([ 5.433 ,  4.4   , 10.8135, ..., 22.2365,  4.651 ,  6.384 ])
```

Metrics evaluatin for Random Forest

Scanned with CamScanner

In [130]:
```
R2_Random = r2_score(y_test,y_pred)
```

In [131]:
```
R2_Random
```

Out[131]:
```
0.7944206535817593
```

In [132]:
```
MSE_Random = mean_squared_error(y_test,y_pred)
```

In [133]:
```
MSE_Random
```

Out[133]:
```
6.108367018742377
```

In [134]:
```
RMSE_Random = np.sqrt(MSE_Random)
```

In [135]:
```
RMSE_Random
```

Out[135]:
```
2.4715110800363362
```


Scanned with CamScanner

Assignment 2

2. 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. Dataset link: The emails.csv dataset on the Kaggle [https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv (https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv)](https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv)

In [16]:
```
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
```

In [17]:
```
df=pd.read_csv('emails.csv')
```

In [18]:
```
df.head()
```

...

In [19]:
```
df.columns
```

...

In [20]:
```
df.isnull().sum()
```

...

In [21]:
```
df.dropna(inplace = True)
```

In [22]:
```
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
```

In [23]:
```
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, randor
```

##KNN classifier

Scanned with CamScanner

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

In [25]:
```
print("Prediction",y_pred)
```

```
Prediction [0 0 1 ... 1 1 1]
```

In [26]:
```
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
KNN accuracy =  0.8009020618556701
```

In [27]:
```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

```
Confusion matrix [[804 293]
 [ 16 439]]
```

SVM classifier

In [28]:
```
# cost C = 1
model = SVC(C = 1)

# fit
model.fit(X_train, y_train)

# predict
y_pred = model.predict(X_test)
```

In [29]:
```
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
```

Out[29]:
```
array([[1091,    6],
       [  90,  365]], dtype=int64)
```

In [30]:
```
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
SVM accuracy =  0.9381443298969072
```


Scanned with CamScanner

Assignment 5

KNN algorithm on diabetes dataset

In [1]:
```
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
```

In [2]:
```
df=pd.read_csv('diabetes.csv')
```

In [3]:
```
df.columns
```

...

Check for null values. If present remove null values from the dataset

In [4]:
```
df.isnull().sum()
```

...

In [ ]:

Outcome is the label/target, other columns are features

In [5]:
```
X = df.drop('Outcome',axis = 1)
y = df['Outcome']
```

In [6]:
```
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, rando
```

In [7]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```


Scanned with CamScanner

In [8]:
```
print("Confusion matrix: ")
cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)
```

```
Confusion matrix:
[[123  28]
 [ 37  43]]
```

In [9]:
```
print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

```
Acccuracy  0.7186147186147186
```

Classification error rate: proportion of instances misclassified over the whole set of instances. Error rate is calculated as the total number of two incorrect predictions (FN + FP) divided by the total number of a dataset (examples in the dataset.

Also error_rate = 1- accuracy

In [10]:
```
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))
```

```
65
231
Error rate 0.2813852813852814
Error rate  0.2813852813852814
```

In [11]:
```
print("Precision score",metrics.precision_score(y_test,y_pred))
```

```
Precision score 0.6056338028169014
```

In [12]:
```
print("Recall score ",metrics.recall_score(y_test,y_pred))
```

```
Recall score  0.5375
```

In [13]:
```
print("Classification report ",metrics.classification_report(y_test,y_pred))
```

```
Classification report                precision    recall  f1-score   support

           0       0.77      0.81      0.79       151
           1       0.61      0.54      0.57        80

    accuracy                           0.72       231
   macro avg       0.69      0.68      0.68       231
weighted avg       0.71      0.72      0.71       231
```


Scanned with CamScanner

Implement Gradient Descent in Python

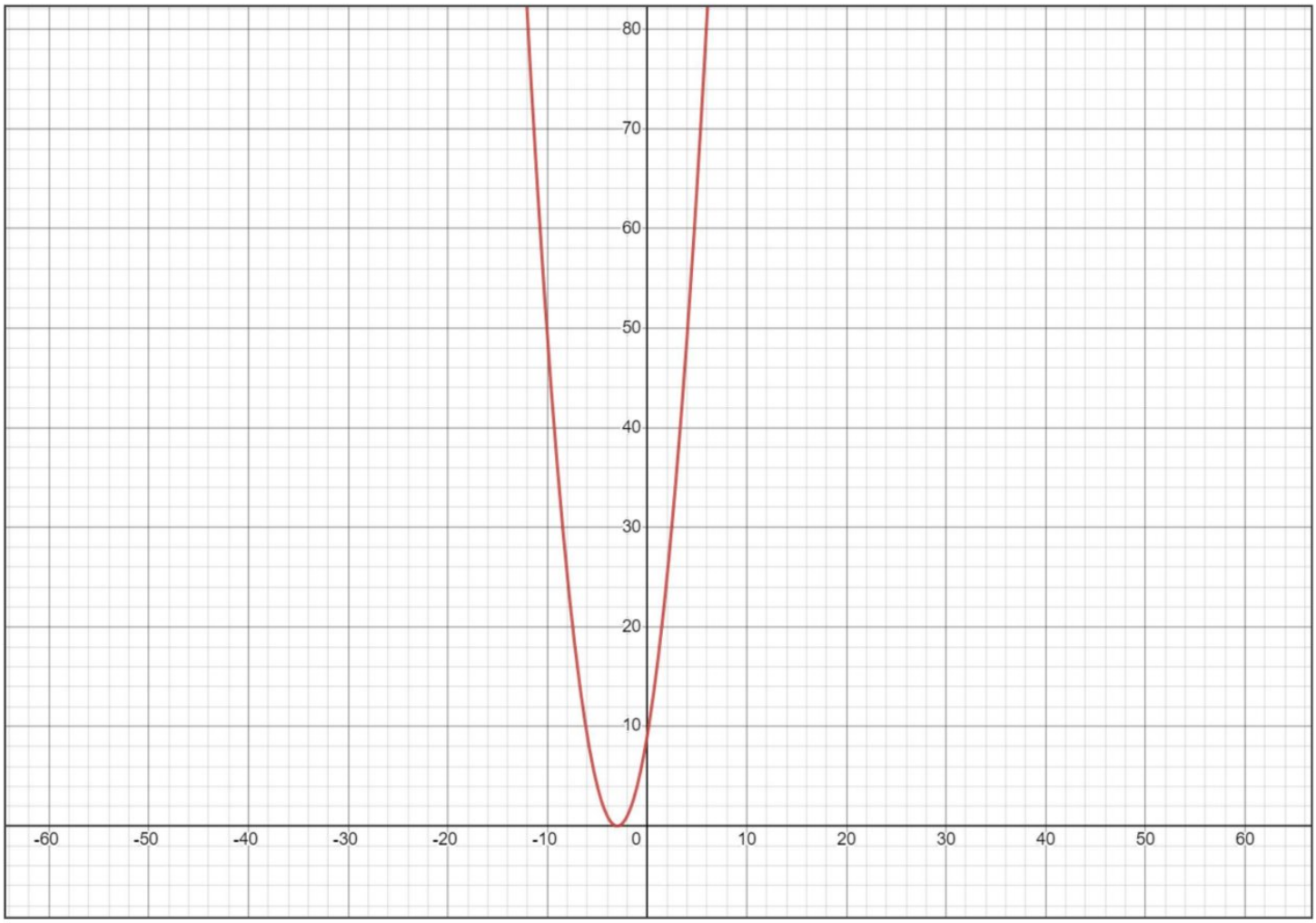
In [1]:
```
cur_x = 2 # The algorithm starts at x=2
rate = 0.01 # Learning rate
precision = 0.000001 #This tells us when to stop the algorithm
previous_step_size = 1 #
max_iters = 10000 # maximum number of iterations
iters = 0 #iteration counter
df = lambda x: 2*(x+3) #Gradient of our function

while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x #Store current x value in prev_x
    cur_x = cur_x - rate * df(prev_x) #Grad descent
    previous_step_size = abs(cur_x - prev_x) #Change in x
    iters = iters+1 #iteration count
    print("Iteration",iters,"\nX value is",cur_x) #Print iterations

print("The local minimum occurs at", cur_x)
```

```
Iteration 563
X value is -2.999942555213562
Iteration 564
X value is -2.999943704109291
Iteration 565
X value is -2.999944830027105
Iteration 566
X value is -2.999945933426563
Iteration 567
X value is -2.999947014758032
Iteration 568
X value is -2.9999480744628713
Iteration 569
X value is -2.999949112973614
Iteration 570
X value is -2.999950130714142
Iteration 571
X value is -2.999951128099859
The local minimum occurs at -2.999951128099859
```

OUTPUT

In [ ]:
```
From the output Above, we can observe the x values for the first 10
iterations- which can be cross checked with our calculation above. The algorithm
runs for 571 iterations before it terminates.
```


Scanned with CamScanner



1

$y = (x+3)^2$

Scanned with CamScanner

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

In [198]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#Importing the required libraries.
```

In [199]:
```
from sklearn.cluster import KMeans, k_means #For clustering
from sklearn.decomposition import PCA #Linear Dimensionality reduction.
```

In [200]:
```
df = pd.read_csv("sales_data_sample.csv") #Loading the dataset.
```

Preprocessing

In [201]:
```
df.head()
```

...

In [202]:
```
df.shape
```

...

In [203]:
```
df.describe()
```

...

In [204]:
```
df.info()
```

...

In [205]:
```
df.isnull().sum()
```

...

In [206]:
```
df.dtypes
```

...

In [207]:
```
df_drop  = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRI
df = df.drop(df_drop, axis=1) #Dropping the categorical uneccessary columns along
```


Scanned with CamScanner

In [208]:
```
df.isnull().sum()
```

...

In [209]:
```
df.dtypes
```

...

In [ ]:
```
# Checking the categorical columns.
```

In [210]:
```
df['COUNTRY'].unique()
```

...

In [211]:
```
df['PRODUCTLINE'].unique()
```

...

In [212]:
```
df['DEALSIZE'].unique()
```

...

In [213]:
```
productline = pd.get_dummies(df['PRODUCTLINE']) #Converting the categorical colum
Dealsize = pd.get_dummies(df['DEALSIZE'])
```

In [214]:
```
df = pd.concat([df,productline,Dealsize], axis = 1)
```

In [215]:
```
df_drop  = ['COUNTRY','PRODUCTLINE','DEALSIZE'] #Dropping Country too as there ar
df = df.drop(df_drop, axis=1)
```

In [216]:
```
df['PRODUCTCODE'] = pd.Categorical(df['PRODUCTCODE']).codes #Converting the datat
```

In [217]:
```
df.drop('ORDERDATE', axis=1, inplace=True) #Dropping the Orderdate as Month is al
```

In [218]:
```
df.dtypes #All the datatypes are converted into numeric
```

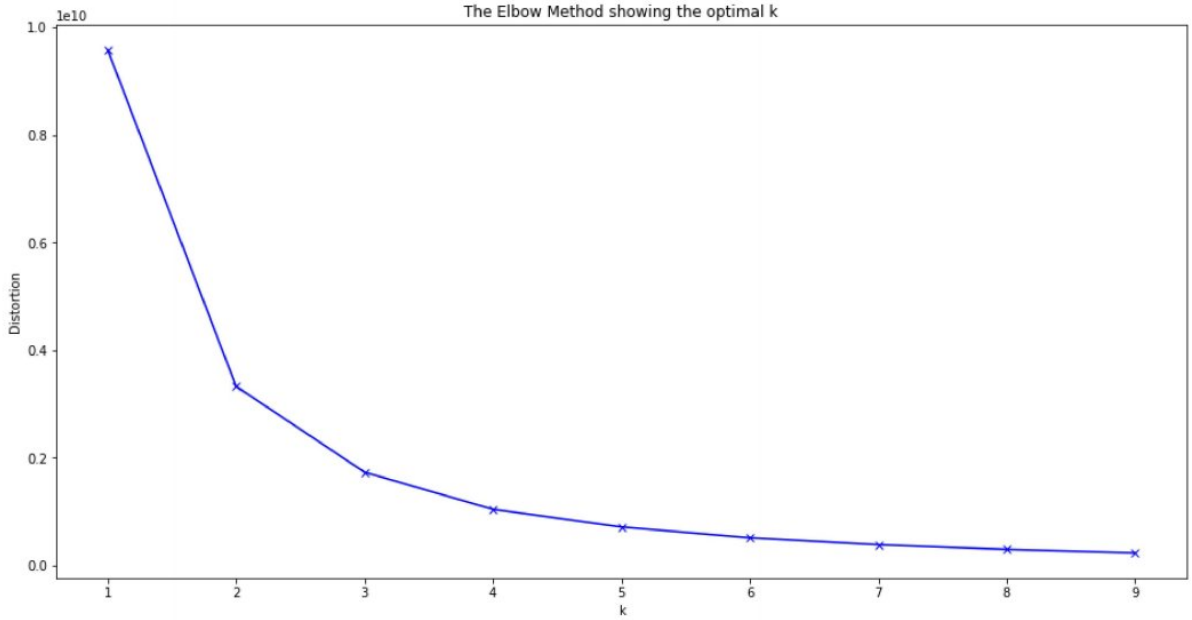
...

Plotting the Elbow Plot to determine the number of clusters.

In [219]:
```
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 Disto
```


Scanned with CamScanner

In [220]:
```
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.

In [221]:
```
X_train = df.values #Returns a numpy array.
```

In [222]:
```
X_train.shape
```

Out[222]:
```
(2823, 19)
```

In [223]:
```
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)
```

In [225]:
```
unique,counts = np.unique(predictions,return_counts=True)
```

In [226]:
```
counts = counts.reshape(1,3)
```


Scanned with CamScanner

In [227]:
```
counts_df = pd.DataFrame(counts,columns=['Cluster1','Cluster2','Cluster3'])
```

In [228]:
```
counts_df.head()
```

Out[228]:

	Cluster1	Cluster2	Cluster3
0	1083	1367	373

Visualization

In [229]:
```
pca = PCA(n_components=2) #Converting all the features into 2 columns to make it
```

In [230]:
```
reduced_X = pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2']) #Cre
```

In [231]:
```
reduced_X.head()
```

Out[231]:

	PCA1	PCA2
0	-682.488323	-42.819535
1	-787.665502	-41.694991
2	330.732170	-26.481208
3	193.040232	-26.285766
4	1651.532874	-6.891196

In [232]:
```
#Plotting the normal Scatter Plot
plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
```

...

In [233]:
```
model.cluster_centers_ #Finding the centriods. (3 Centriods in total. Each Array
```

...

In [234]:
```
reduced_centers = pca.transform(model.cluster_centers_) #Transforming the centro
```

In [235]:
```
reduced_centers
```

Out[235]:
```
array([[ 5.84994044e+02, -4.36786931e+00],
       [-1.43005891e+03,  2.60041009e+00],
       [ 3.54247180e+03,  3.15185487e+00]])
```

In [236]:
```
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=
```

...

Scanned with CamScanner

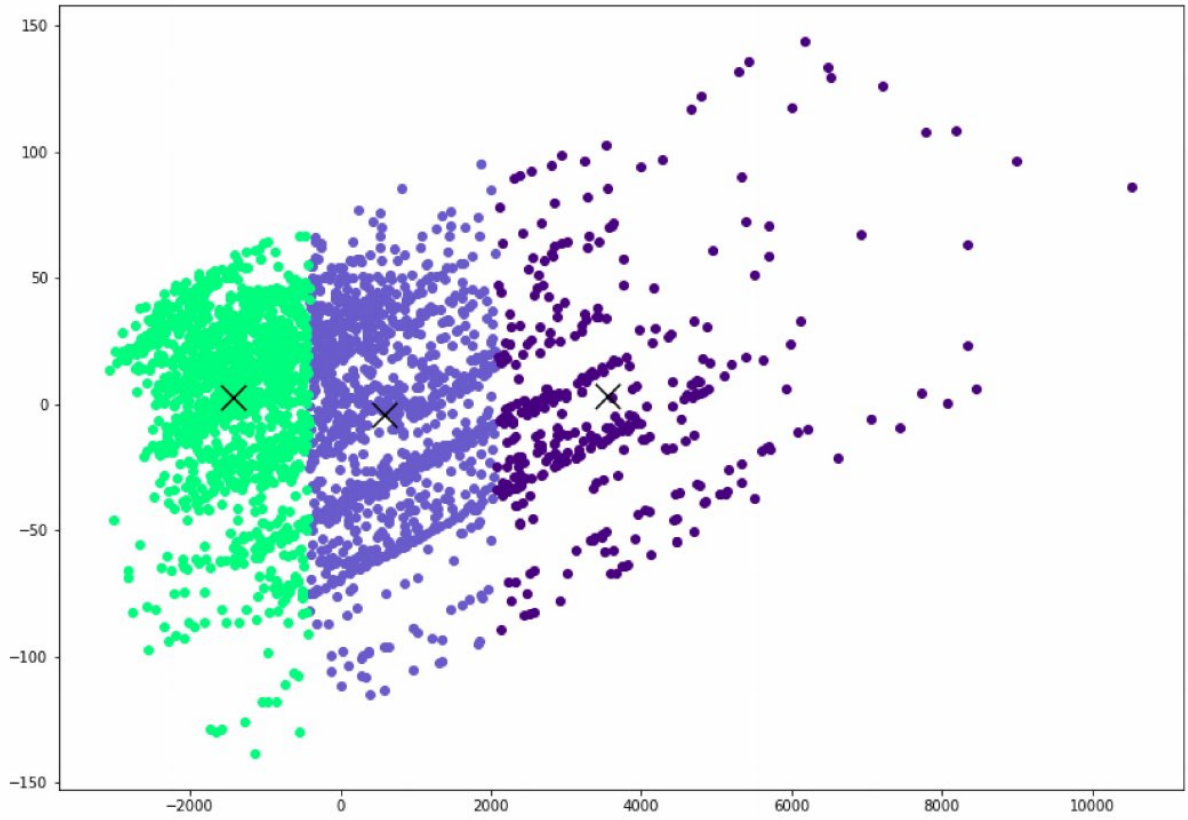
In [237]:
```
reduced_X['Clusters'] = predictions #Adding the Clusters to the reduced dataframe
```

In [238]:
```
reduced_X.head()
```

...

In [239]:
```
#Plotting the clusters
plt.figure(figsize=(14,10))
#                    taking the cluster number and first column          taking
plt.scatter(reduced_X[reduced_X['Clusters'] == 0].loc[:,'PCA1'],reduced_X[reduced
plt.scatter(reduced_X[reduced_X['Clusters'] == 1].loc[:,'PCA1'],reduced_X[reduced
plt.scatter(reduced_X[reduced_X['Clusters'] == 2].loc[:,'PCA1'],reduced_X[reduced


plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=
```

Out[239]:
```
<matplotlib.collections.PathCollection at 0x218dce9e1f0>
```



Scanned with CamScanner