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()
```

Out[201]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDA
0	10107	30	95.70	2	2871.00	2/24/2 (
1	10121	34	81.35	5	2765.90	5/7/2003 (
2	10134	41	94.74	2	3884.34	7/1/2003 (
3	10145	45	83.26	6	3746.70	8/25/2 (
4	10159	49	100.00	14	5205.27	10/10/2 (

5 rows × 25 columns

In [202]:

```
1 df.shape
```

Out[202]:

```
(2823, 25)
```

In [203]:

```
1 df.describe()
```

Out[203]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000
mean	10258.725115	35.092809	83.658544	6.466171	3553.889072
std	92.085478	9.741443	20.174277	4.225841	1841.865106
min	10100.000000	6.000000	26.880000	1.000000	482.130000
25%	10180.000000	27.000000	68.860000	3.000000	2203.430000
50%	10262.000000	35.000000	95.700000	6.000000	3184.800000
75%	10333.500000	43.000000	100.000000	9.000000	4508.000000
max	10425.000000	97.000000	100.000000	18.000000	14082.800000

In [204]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  ADDRESSLINE2      302 non-null    object 
 17  CITY              2823 non-null   object 
 18  STATE             1337 non-null   object 
 19  POSTALCODE        2747 non-null   object 
 20  COUNTRY           2823 non-null   object 
 21  TERRITORY         1749 non-null   object 
 22  CONTACTLASTNAME   2823 non-null   object 
 23  CONTACTFIRSTNAME  2823 non-null   object 
 24  DEALSIZE          2823 non-null   object 
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [205]:

```
1 df.isnull().sum()
```

Out[205]:

```
ORDERNUMBER           0
QUANTITYORDERED       0
PRICEEACH             0
ORDERLINENUMBER       0
SALES                 0
ORDERDATE             0
STATUS                0
QTR_ID                0
MONTH_ID              0
YEAR_ID               0
PRODUCTLINE           0
MSRP                  0
PRODUCTCODE           0
CUSTOMERNAME          0
PHONE                 0
ADDRESSLINE1          0
ADDRESSLINE2       2521
CITY                  0
STATE              1486
POSTALCODE           76
COUNTRY               0
TERRITORY          1074
CONTACTLASTNAME       0
CONTACTFIRSTNAME      0
DEALSIZE              0
dtype: int64
```

In [206]:

```
df.dtypes
```

Out[206]:

```
ORDERNUMBER          int64
QUANTITYORDERED      int64
PRICEEACH          float64
ORDERLINENUMBER      int64
SALES              float64
ORDERDATE           object
STATUS              object
QTR_ID               int64
MONTH_ID             int64
YEAR_ID              int64
PRODUCTLINE         object
MSRP                 int64
PRODUCTCODE         object
CUSTOMERNAME        object
PHONE               object
ADDRESSLINE1        object
ADDRESSLINE2        object
CITY                object
STATE               object
POSTALCODE          object
COUNTRY             object
TERRITORY           object
CONTACTLASTNAME     object
CONTACTFIRSTNAME    object
DEALSIZE            object
dtype: object
```

In [207]:

```
df_drop  = ['ADDRESSLINE1', 'ADDRESSLINE2', 'STATUS','POSTALCODE', 'CITY', 'TERRITORY',
df = df.drop(df_drop, axis=1) #Dropping the categorical uneccessary columns along with
```

In [208]:

```
df.isnull().sum()
```

Out[208]:

```
QUANTITYORDERED    0
PRICEEACH          0
ORDERLINENUMBER    0
SALES              0
ORDERDATE          0
QTR_ID             0
MONTH_ID           0
YEAR_ID            0
PRODUCTLINE        0
MSRP               0
PRODUCTCODE        0
COUNTRY            0
DEALSIZE           0
dtype: int64
```

In [209]:

```
df.dtypes
```

Out[209]:

```
QUANTITYORDERED      int64
PRICEEACH          float64
ORDERLINENUMBER      int64
SALES              float64
ORDERDATE           object
QTR_ID               int64
MONTH_ID             int64
YEAR_ID              int64
PRODUCTLINE         object
MSRP                 int64
PRODUCTCODE         object
COUNTRY             object
DEALSIZE            object
dtype: object
```

In [ ]:

```
# Checking the categorical columns.
```

In [210]:

```
df['COUNTRY'].unique()
```

Out[210]:

```
array(['USA', 'France', 'Norway', 'Australia', 'Finland', 'Austria', 'UK',
       'Spain', 'Sweden', 'Singapore', 'Canada', 'Japan', 'Italy',
       'Denmark', 'Belgium', 'Philippines', 'Germany', 'Switzerland',
       'Ireland'], dtype=object)
```

In [211]:

```
df['PRODUCTLINE'].unique()
```

Out[211]:

```
array(['Motorcycles', 'Classic Cars', 'Trucks and Buses', 'Vintage Cars',
       'Planes', 'Ships', 'Trains'], dtype=object)
```

In [212]:

```
df['DEALSIZE'].unique()
```

Out[212]:

```
array(['Small', 'Medium', 'Large'], dtype=object)
```

In [213]:

```
productline = pd.get_dummies(df['PRODUCTLINE']) #Converting the categorical columns.
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 are alot
df = df.drop(df_drop, axis=1)
```

In [216]:

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

In [217]:

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

In [218]:

```
df.dtypes #All the datatypes are converted into numeric
```

Out[218]:

```
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
MSRP                  int64
PRODUCTCODE            int8
Classic Cars          uint8
Motorcycles           uint8
Planes                uint8
Ships                 uint8
Trains                uint8
Trucks and Buses      uint8
Vintage Cars          uint8
Large                 uint8
Medium                uint8
Small                 uint8
dtype: object
```

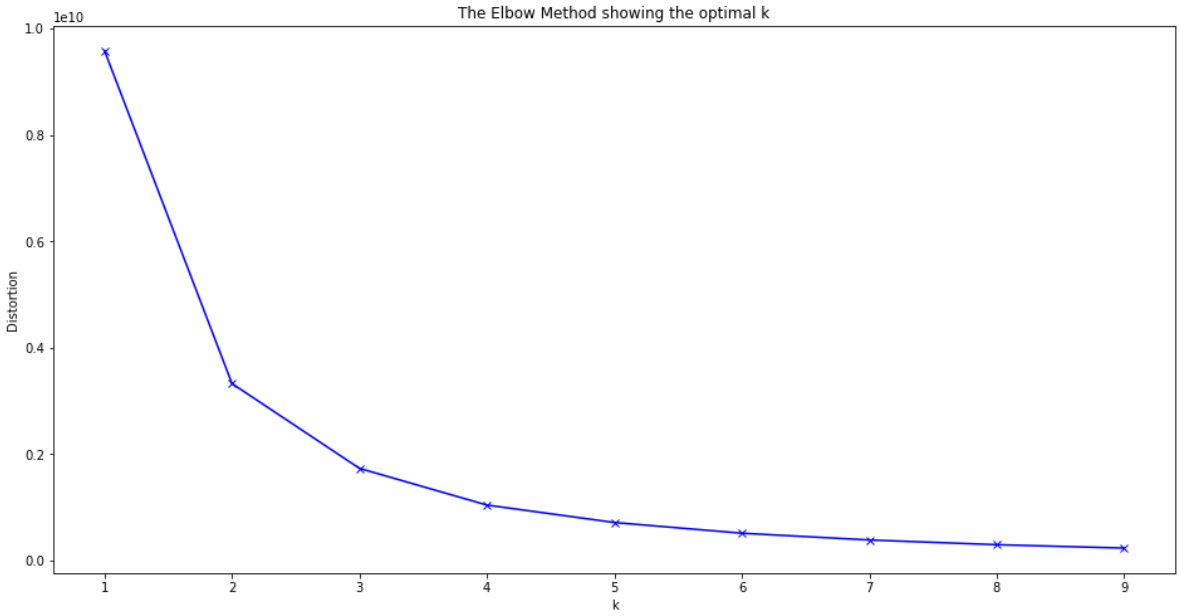
Plotting the Elbow Plot to determine the number of clusters.

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 Distortions
```

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)
```

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 easy t
```

In [230]:

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

In [231]:

```
reduced_X.head()
```

Out[231]:

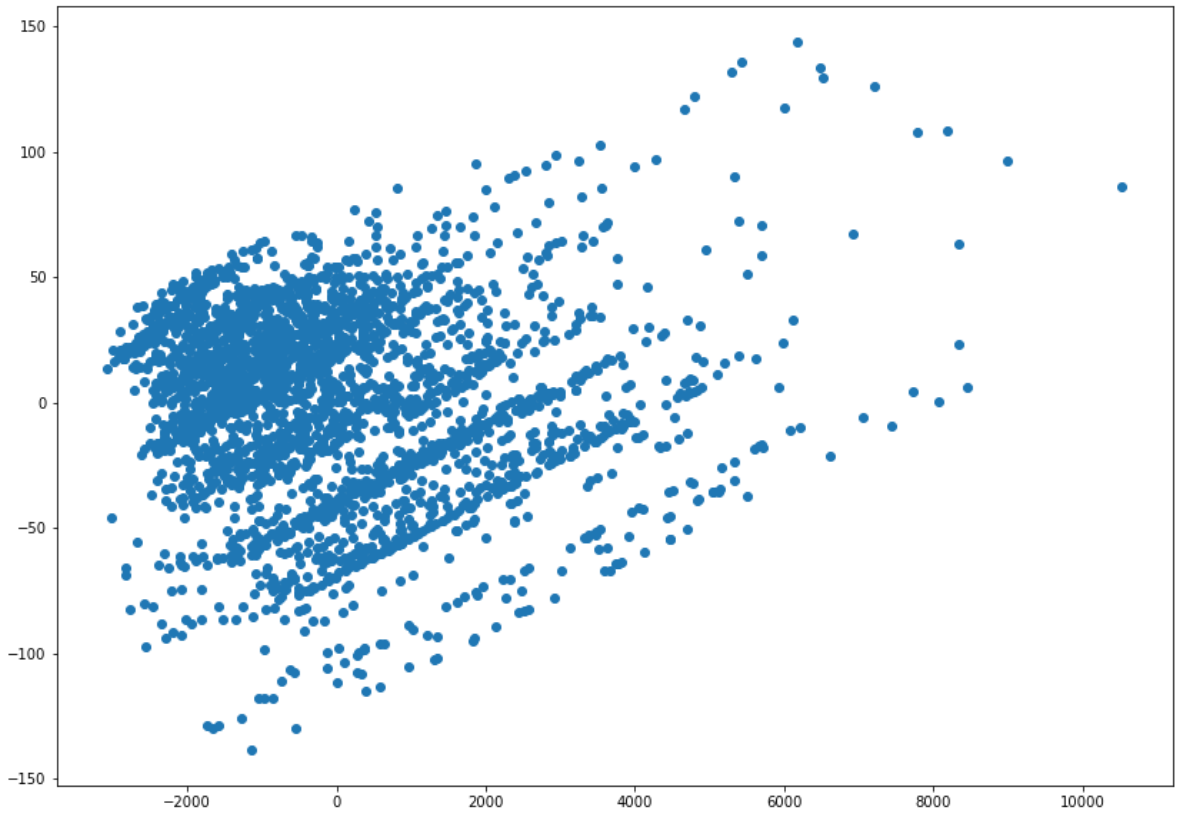
	PCA1	PCA2
0	-682.488323	-42.819535
1	-787.665502	-41.694991
2	330.732170	-26.481208
3	193.040232	-26.285766
4	1651.532874	-6.891196

In [232]:

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

Out[232]:

```
<matplotlib.collections.PathCollection at 0x218dc747880>
```



In [233]:

```
1 model.cluster_centers_ #Finding the centriods. (3 Centriods in total. Each Array contai
```

Out[233]:

```
array([[ 3.72031394e+01,  9.52120960e+01,  6.44967682e+00,
         4.13868425e+03,  2.72022161e+00,  7.09879963e+00,
         2.00379409e+03,  1.13248384e+02,  5.04469067e+01,
         3.74884580e-01,  1.15420129e-01,  9.41828255e-02,
         8.21791320e-02,  1.84672207e-02,  1.16343490e-01,
         1.98522622e-01,  2.08166817e-17,  1.00000000e+00,
        -6.66133815e-16],
       [ 3.08302853e+01,  7.00755230e+01,  6.67300658e+00,
         2.12409474e+03,  2.71762985e+00,  7.09509876e+00,
         2.00381127e+03,  7.84784199e+01,  6.24871982e+01,
         2.64813460e-01,  1.21433797e-01,  1.29480614e-01,
         1.00219459e-01,  3.87710315e-02,  9.21726408e-02,
         2.53108998e-01,  6.93889390e-18,  6.21799561e-02,
         9.37820044e-01],
       [ 4.45871314e+01,  9.98931099e+01,  5.75603217e+00,
         7.09596863e+03,  2.71045576e+00,  7.06434316e+00,
         2.00389008e+03,  1.45823056e+02,  3.14959786e+01,
         5.33512064e-01,  1.07238606e-01,  7.23860590e-02,
         2.14477212e-02,  1.07238606e-02,  1.31367292e-01,
         1.23324397e-01,  4.20911528e-01,  5.79088472e-01,
         5.55111512e-17]])
```

In [234]:

```
1 reduced_centers = pca.transform(model.cluster_centers_) #Transforming the centroids int
```

In [235]:

```
1 reduced_centers
```

Out[235]:

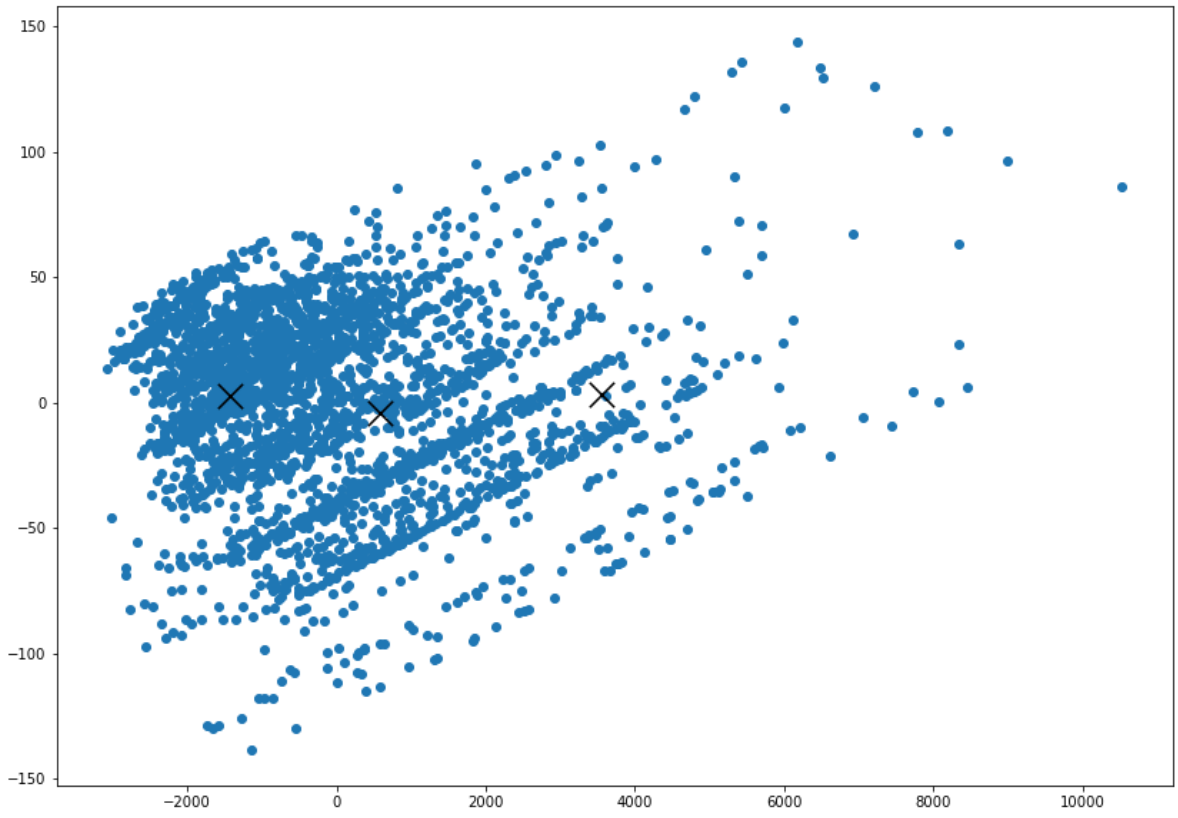
```
array([[ 5.84994044e+02, -4.36786931e+00],
       [-1.43005891e+03,  2.60041009e+00],
       [ 3.54247180e+03,  3.15185487e+00]])
```

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=300) #
```

Out[236]:

```
<matplotlib.collections.PathCollection at 0x218deb6e220>
```



In [237]:

```
reduced_X['Clusters'] = predictions #Adding the Clusters to the reduced dataframe.
```

In [238]:

```
reduced_X.head()
```

Out[238]:

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

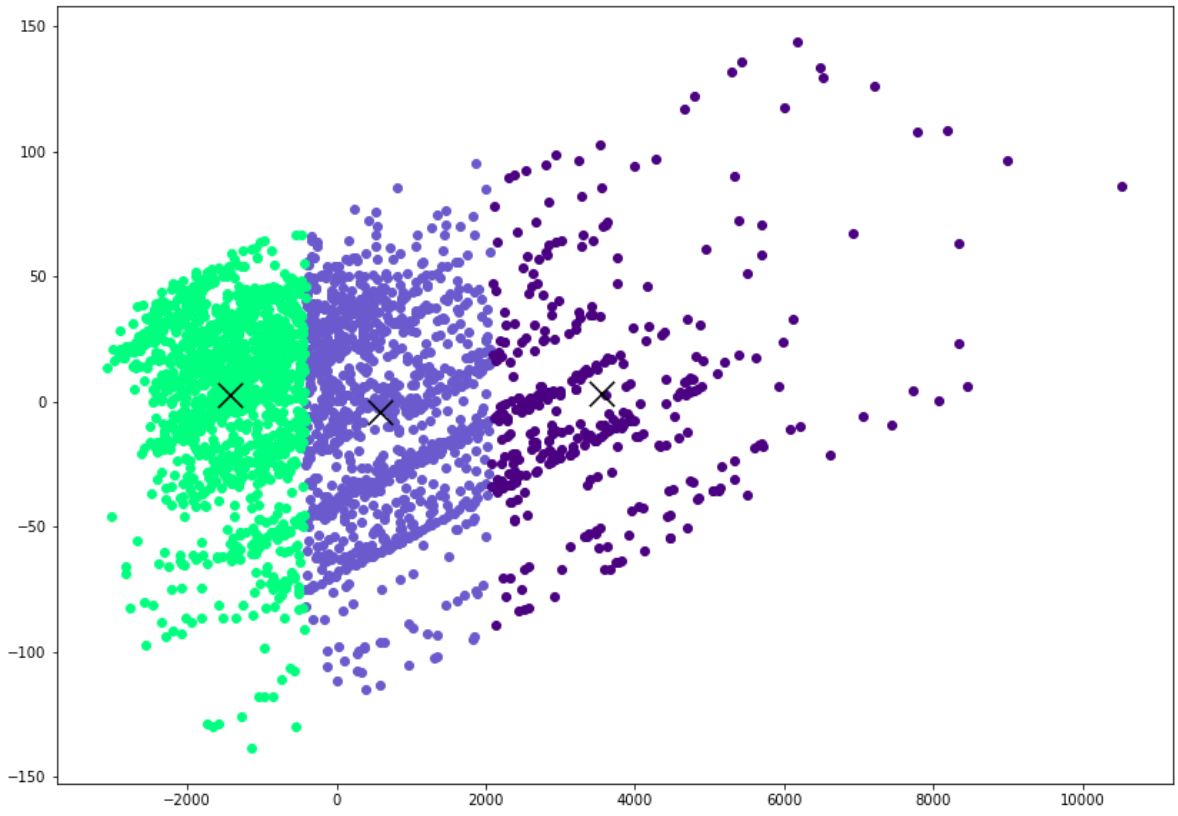
In [239]:

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


plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',marker='x',s=300)
```

Out[239]:

```
<matplotlib.collections.PathCollection at 0x218dce9e1f0>
```



In [ ]:

```
1
```

Experiment:04

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

Importing the libraries

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
```

Reading the dataset

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

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

Out[3]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insul
in',
       'BMI', 'Pedigree', 'Age', 'Outcome'],
      dtype='object')
```

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

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

Out[4]:
```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

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
```

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

In [9]:
```
print("Confusion matrix: ")
cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)
```

```
Confusion matrix:
[[134  17]
 [ 41  39]]
```

In [10]:
```
print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

```
Acccuracy  0.7489177489177489
```

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 [11]:
```
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))
```

```
58
231
Error rate 0.2510822510822511
Error rate  0.25108225108225113
```

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

```
Precision score 0.6964285714285714
```

In [13]:
```
print("Recall score ",metrics.recall_score(y_test,y_pred))
```

```
Recall score  0.4875
```

In [14]:
```
print("Classification report ",metrics.classification_report(y_test,y_pre
```

```
Classification report                precision    recall  f1-score   supp
ort

           0       0.77      0.89      0.82       151
           1       0.70      0.49      0.57        80

    accuracy                           0.75       231
   macro avg       0.73      0.69      0.70       231
weighted avg       0.74      0.75      0.74       231
```

In [ ]:
```
```

Experiment:03

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.

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.

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

In [3]:
```
df = pd.read_csv("Churn_Modelling.csv")
```

Data Preprocessing

In [4]:
```
df.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	838
2	3	15619304	Onio	502	France	Female	42	8	1596
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1255

In [5]:
```
df.shape
```

Out[5]:
```
(10000, 14)
```

In [6]:
```
df.describe()
```

Out[6]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

In [7]:
```
df.isnull()
```

Out[7]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
9995	False	False	False	False	False	False	False	False
9996	False	False	False	False	False	False	False	False
9997	False	False	False	False	False	False	False	False
9998	False	False	False	False	False	False	False	False
9999	False	False	False	False	False	False	False	False

10000 rows × 14 columns

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [10]:
```
df.dtypes
```

Out[10]:
```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [11]:
```
df.columns
```

Out[11]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCar
d',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [12]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping t
```

In [13]:
```
df.head()
```

Out[13]:

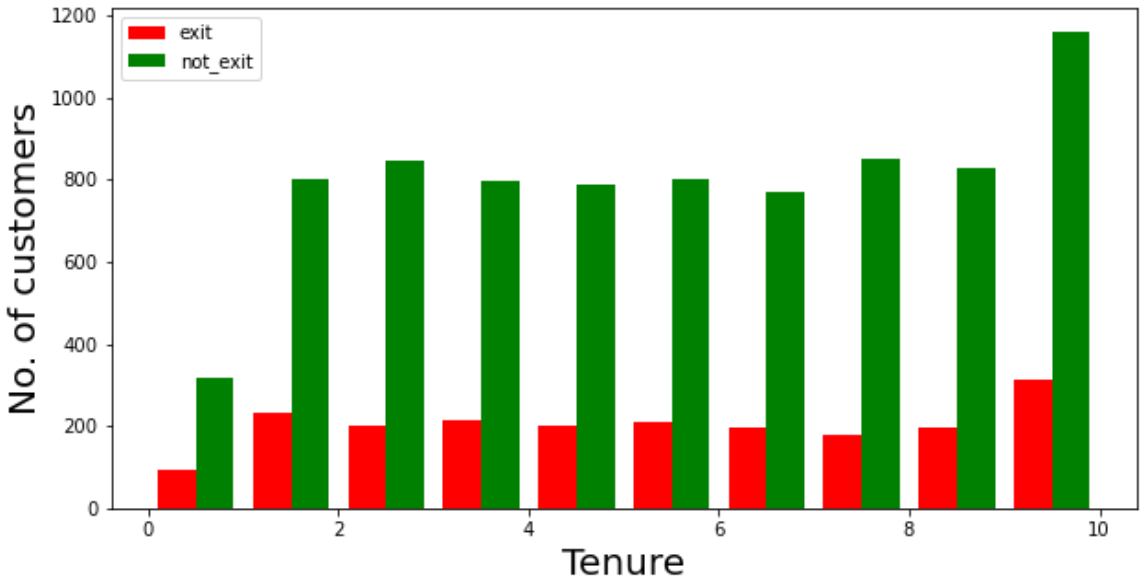
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	I
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

Data Visualization

In [14]:
```
def visualization(x, y, xlabel):
 plt.figure(figsize=(10,5))
 plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
 plt.xlabel(xlabel,fontsize=20)
 plt.ylabel("No. of customers", fontsize=20)
 plt.legend()
```

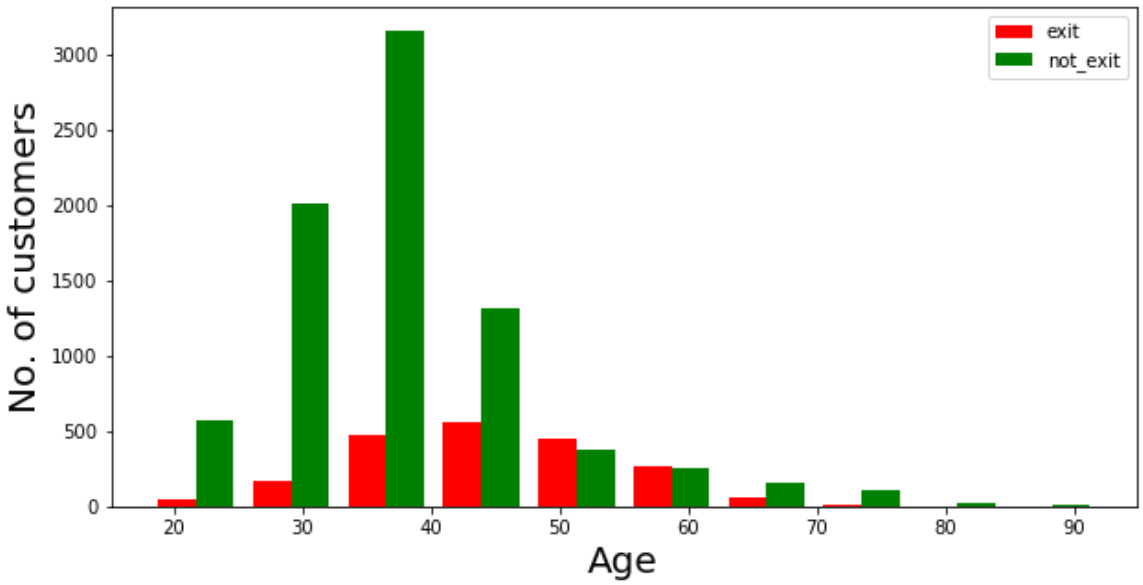
In [15]:
```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

In [16]:
```
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```



In [17]:
```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

In [18]:
```
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



In [ ]:
```
# Converting the Categorical Variables
```

In [21]:
```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
```

In [22]:
```
df = pd.concat([df,gender,states], axis = 1)
```

In [ ]:
```
#Splitting the training and testing Dataset
```

In [23]:
```
df.head()
```

Out[23]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	I
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

In [24]:
```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard
```

In [25]:
```
y = df['Exited']
```

In [26]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

In [ ]:
```
#Normalizing the values with mean as 0 and Standard Deviation as 1
```

In [27]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [28]:
```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [29]:
```
X_train
```

Out[29]:
```
array([[-2.98727399e-01,  8.71060334e-02,  1.03124628e+00, ...,
        -1.03845308e+00,  1.25879170e+00,  1.97186920e+00],
       [-2.16781127e+00,  2.76997729e-01, -1.03311404e+00, ...,
        -1.03845308e+00, -8.76580025e-01, -5.07133028e-01],
       [-1.68246905e+00,  1.32140205e+00,  1.71936639e+00, ...,
         9.62970807e-01, -1.75893559e-01, -5.07133028e-01],
       ...,
       [ 4.13796286e-01, -1.24213583e+00, -9.33877289e-04, ...,
         9.62970807e-01, -8.80538844e-01, -5.07133028e-01],
       [ 2.06602512e+00, -2.92677357e-01, -1.72123415e+00, ...,
        -1.03845308e+00,  7.14813922e-01, -5.07133028e-01],
       [ 5.17060588e-01, -1.24213583e+00, -3.44993931e-01, ...,
        -1.03845308e+00,  7.26718752e-01, -5.07133028e-01]])
```

In [30]:
```
X_test
```

Out[30]:
```
array([[-2.10585269e+00, -6.72460748e-01,  1.03124628e+00, ...,
        -1.03845308e+00,  5.76424490e-01, -5.07133028e-01],
       [-2.27107557e+00,  9.41618663e-01, -6.89053985e-01, ...,
        -1.03845308e+00,  8.06630270e-01,  1.97186920e+00],
       [ 1.05403496e+00,  8.46672815e-01, -9.33877289e-04, ...,
        -1.03845308e+00,  9.82480418e-01,  1.97186920e+00],
```

```
       ...,
       [ 5.48039879e-01,  1.03656451e+00,  1.03124628e+00, ...,
         9.62970807e-01, -3.63254588e-01, -5.07133028e-01],
       [ 1.10566711e+00,  6.56781119e-01, -1.72123415e+00, ...,
        -1.03845308e+00, -1.09186186e+00, -5.07133028e-01],
       [-1.87867122e+00, -9.57298291e-01, -1.37717409e+00, ...,
         9.62970807e-01,  1.66301871e+00, -5.07133028e-01]])
```

In [1]:
```
#Building the Classifier Model using Keras
```

In [ ]:
```
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors in
```

In [ ]:
```
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

In [ ]:
```
classifier = Sequential()
```

In [ ]:
```
#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_
```

In [ ]:
```
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "
```

In [ ]:
```
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer
```

In [ ]:
```
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics
```

In [ ]:
```
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd
```

Practical No.2

Email Spam detection

In [18]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [19]:

```
df=pd.read_csv('emails.csv')
```

In [20]:

```
df.head()
```

Out[20]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastruc
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	

5 rows × 3002 columns

In [21]:

```
df.tail()
```

Out[21]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infras
5167	Email 5168	2	2	2	3	0	0	32	0	0	...	0	0	0	0	
5168	Email 5169	35	27	11	2	6	5	151	4	3	...	0	0	0	0	
5169	Email 5170	0	0	1	1	0	0	11	0	0	...	0	0	0	0	
5170	Email 5171	2	7	1	0	2	1	28	2	0	...	0	0	0	0	
5171	Email 5172	22	24	5	1	6	5	148	8	2	...	0	0	0	0	

5 rows × 3002 columns

In [22]:

```
df.shape
```

Out[22]:

```
(5172, 3002)
```

In [23]:

```
df.isnull().sum()
```

Out[23]:

```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

In [24]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
```

In [25]:

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

In [26]:

```
from sklearn.model_selection import train_test_split
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=114
```

In [27]:

```
from sklearn.svm import SVC
from sklearn import metrics
```

In [28]:

```
X_train
```

Out[28]:

```
array([[-0.13969536, -0.12462466,  0.20256691, ..., -0.0562853 ,
         0.03067224, -0.07097072],
       [-0.22484614, -0.22951624, -0.2938948 , ..., -0.0562853 ,
        -0.32904848, -0.07097072],
       [ 2.24452652,  1.0291827 , -0.0811255 , ..., -0.0562853 ,
        -0.32904848, -0.07097072],
       ...,
       [ 0.88211402,  0.50472481, -0.1520486 , ..., -0.0562853 ,
        -0.32904848, -0.07097072],
       [ 4.6287484 ,  4.91017112,  1.05364412, ..., -0.0562853 ,
         2.54871731, -0.07097072],
       [ 2.15937574,  0.92429113,  1.33733653, ..., -0.0562853 ,
         0.39039297, -0.07097072]])
```

In [29]:

```
y_train
```

Out[29]:

```
4172    0
5151    0
3061    0
2276    0
2115    0
       ..
5149    0
1294    1
850     0
1404    1
2642    0
Name: Prediction, Length: 3620, dtype: int64
```

In [30]:

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

In [31]:

```
print("Prediction",y_pred)
```

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

In [32]:

```
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
KNN accuracy =  0.8118556701030928
```

In [33]:

```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

```
Confusion matrix [[848 274]
 [ 18 412]]
```

In [34]:

```
model = SVC(C = 1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
```

Out[34]:

```
array([[1112,   10],
       [  90,  340]], dtype=int64)
```

In [35]:

```
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

```
SVM accuracy =  0.9355670103092784
```

In [ ]:

```

```

Practical No. 1

#Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
df  = pd.read_csv("uber.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         200000 non-null  int64
 1   key                200000 non-null  object
```

```
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [6]:
```
df = df.drop(['Unnamed: 0', 'key'], axis= 1)
```

In [7]:
```
df.head()
```

Out[7]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latit
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761

In [8]:
```
df.shape
```

Out[8]:
```
(200000, 7)
```

In [9]:
```
df.dtypes
```

Out[9]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
```

```
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   fare_amount        200000 non-null  float64
 1   pickup_datetime    200000 non-null  object 
 2   pickup_longitude   200000 non-null  float64
 3   pickup_latitude    200000 non-null  float64
 4   dropoff_longitude  199999 non-null  float64
 5   dropoff_latitude   199999 non-null  float64
 6   passenger_count    200000 non-null  int64  
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
```

In [11]:
```
df.describe()
```

Out[11]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200(
mean	11.359955	-72.527638	39.935885	-72.525292	39.923890	
std	9.901776	11.437787	7.720539	13.117408	6.794829	
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823	
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	
75%	12.500000	-73.967154	40.767158	-73.963658	40.768001	
max	499.000000	57.418457	1644.421482	1153.572603	872.697628	

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [13]:
```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True
```

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

In [15]:
```
df.dtypes
```

Out[15]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [16]:
```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```

In [17]:
```
df.dtypes
```

Out[17]:
```
fare_amount                       float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude                  float64
pickup_latitude                   float64
dropoff_longitude                 float64
dropoff_latitude                  float64
passenger_count                     int64
dtype: object
```

In [18]:
```
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 [19]:
```
df.head()
```

Out[19]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latit
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.723
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.750
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.772
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.803
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.761

In [20]:
```
```

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

In [21]:
```
df.head()
```

Out[21]:

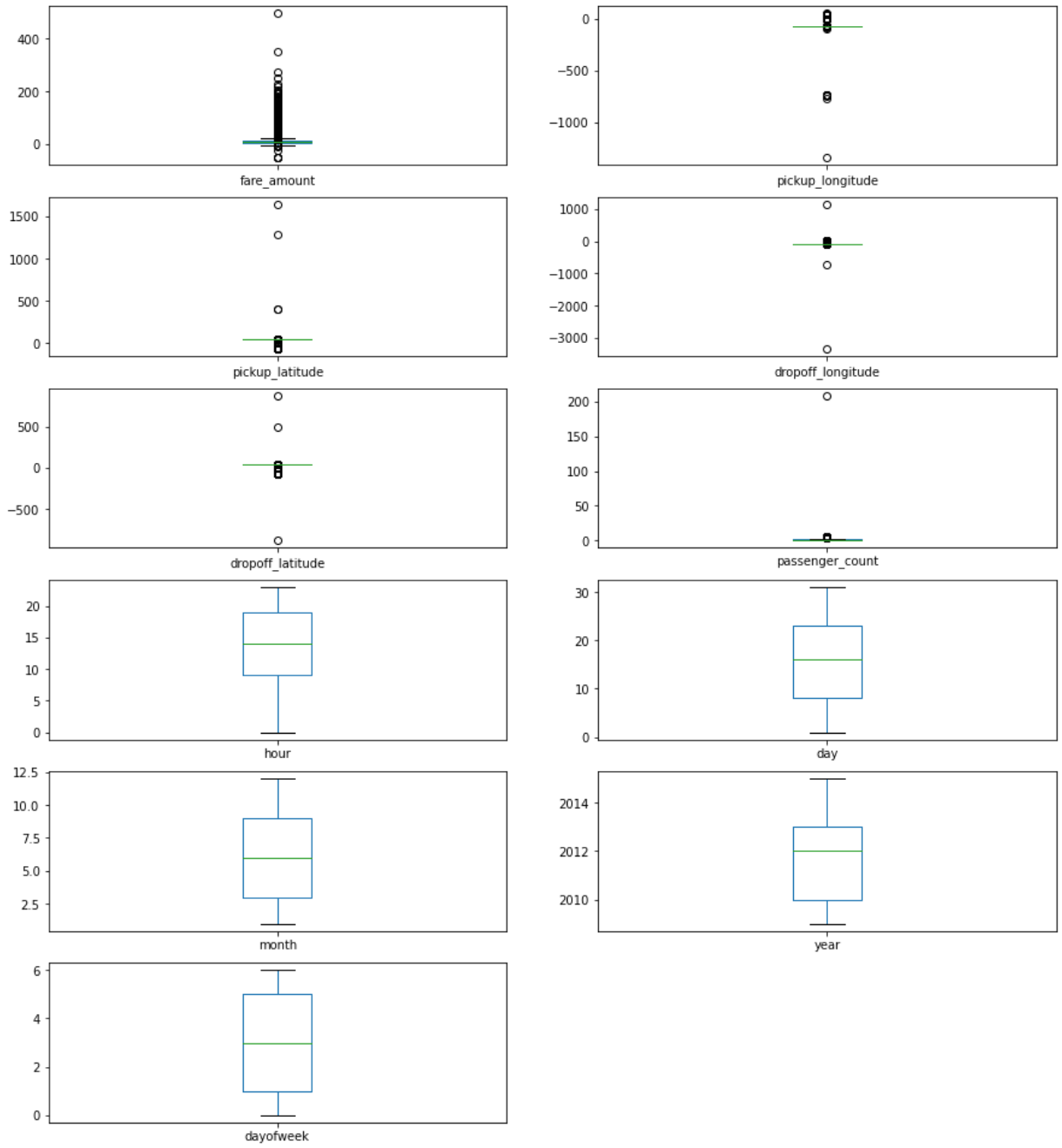
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_cc
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	

In [22]:
```
df.dtypes
```

Out[22]:
```
fare_amount          float64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
hour                   int64
day                    int64
month                  int64
year                   int64
dayofweek              int64
dtype: object
```

In [23]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))
```

Out[23]:
```
fare_amount            AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude        AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude       AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                   AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                  AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek              AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```



In [24]:

```
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 [25]:

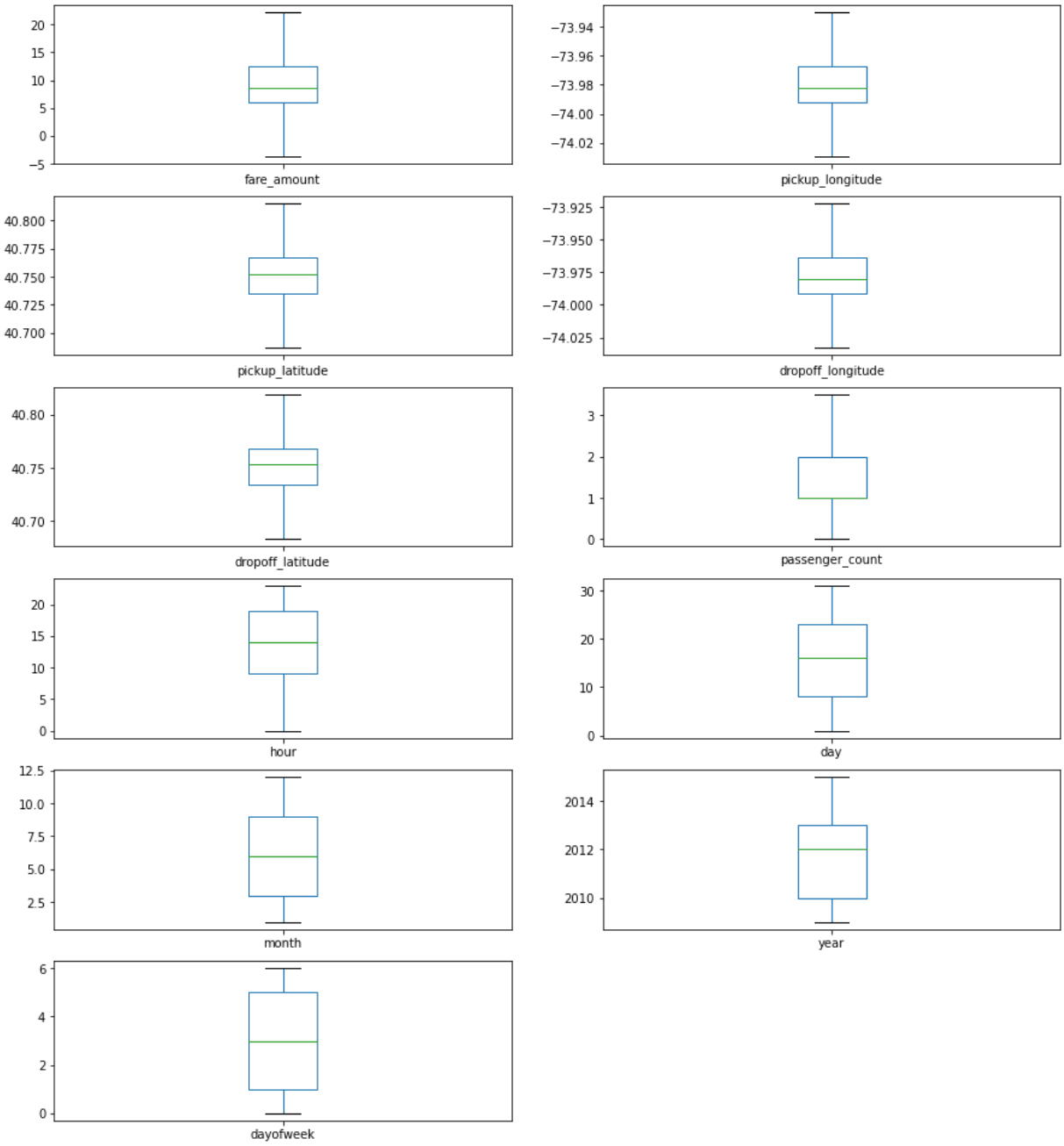
```
df = treat_outliers_all(df , df.iloc[: , 0::])
```

In [26]:

```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20))
```

Out[26]:

```
fare_amount            AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude        AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude       AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                   AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                  AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek              AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```



In [27]:

```
import haversine as hs
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pickup_latitude']
        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()
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

Out[27]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_cc
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.929786	40.744085	-73.973082	40.761247	

In [28]:

```
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 [29]:

```
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latitude < -90)
                               (df.dropoff_latitude > 90) |(df.dropoff_latitude
                               (df.pickup_longitude > 180) |(df.pickup_longitude
                               (df.dropoff_longitude > 90) |(df.dropoff_longitud
                                ]
```

In [30]:

```
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

In [31]:

```
df.head()
```

Out[31]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_cc
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.929786	40.744085	-73.973082	40.761247	

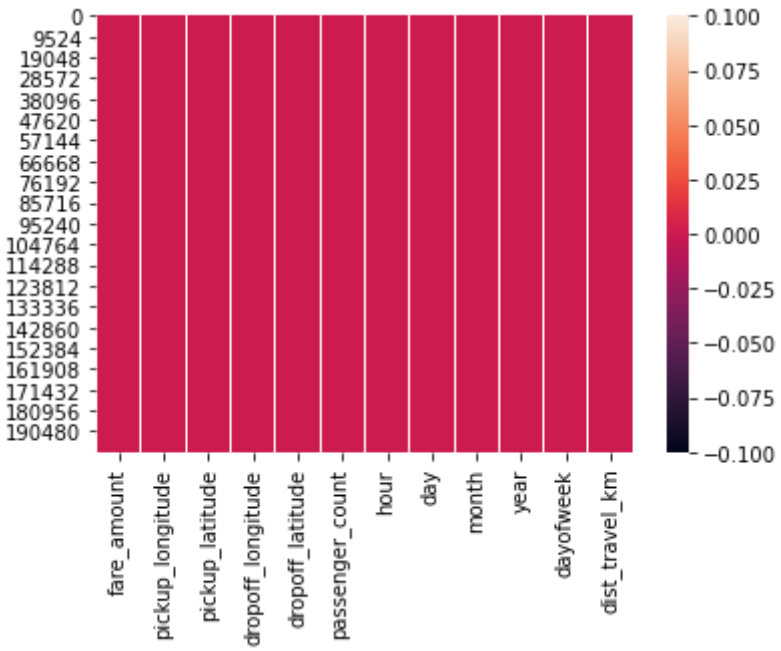
In [32]:

```
df.isnull().sum()
```

Out[32]:
```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
hour                 0
day                  0
month                0
year                 0
dayofweek            0
dist_travel_km       0
dtype: int64
```

In [33]:
```
sns.heatmap(df.isnull())
```

Out[33]:
```
<AxesSubplot:>
```



In [34]:
```
corr = df.corr()
```

In [35]:
```
corr
```

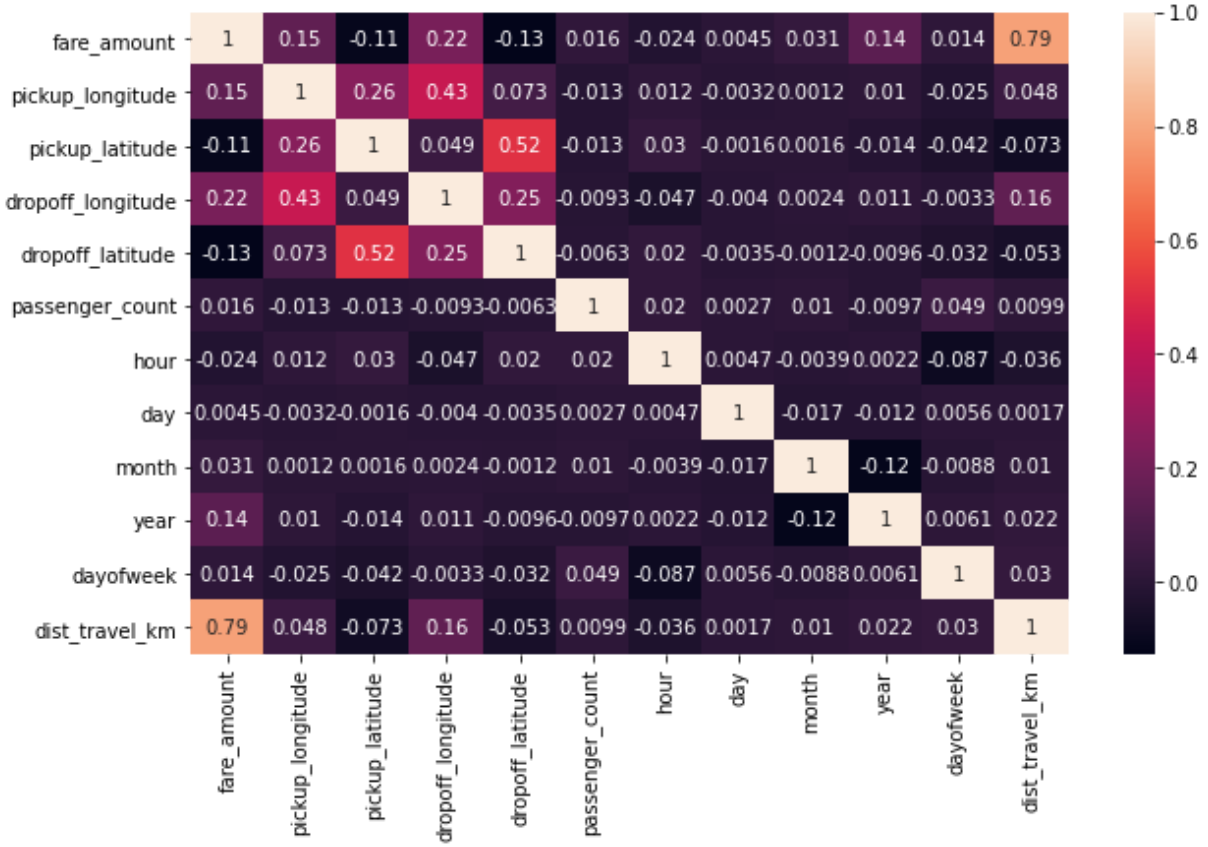
Out[35]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitud
fare_amount	1.000000	0.154069	-0.110842	0.218675	-0.1258
pickup_longitude	0.154069	1.000000	0.259497	0.425619	0.0732
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	0.5157
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	0.2456
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	1.0000
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	-0.0063
hour	-0.023623	0.011579	0.029681	-0.046558	0.0197
day	0.004534	-0.003204	-0.001553	-0.004007	-0.0034
month	0.030817	0.001169	0.001562	0.002391	-0.0011

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitud
year	0.141277	0.010198	-0.014243	0.011346	-0.0096(
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	-0.0319
dist_travel_km	0.786385	0.048446	-0.073362	0.155191	-0.0527(

In [36]:
```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True)
```

Out[36]: <AxesSubplot:>



In [182…]:
```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude',
```

In [183…]:
```
y = df['fare_amount']
```

In [184…]:
```
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)
```

In [185…]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [186…
```
regression.fit(X_train,y_train)
```

Out[186…
```
LinearRegression()
```

In [80]:
```
regression.intercept_
```

Out[80]:
```
2640.1356169149753
```

In [187…
```
regression.coef_
```

Out[187…
```
array([ 2.54805415e+01, -7.18365435e+00,  1.96232986e+01, -1.79401980e+01,
        5.48472723e-02,  5.32910041e-03,  4.05930990e-03,  5.74261856e-02,
        3.66574831e-01, -3.03753790e-02,  1.84233728e+00])
```

In [188…
```
prediction = regression.predict(X_test)
```

In [189…
```
print(prediction)
```

```
[ 5.47848314 10.11016249 12.19490542 ...  7.11952609 20.2482979
  8.82791961]
```

In [190…
```
y_test
```

Out[190…
```
155740     4.90
47070     10.00
116192    14.50
164589     6.50
154309    11.30
          ...
76552      7.70
27926     10.90
38972      6.50
120341    22.25
178449     8.10
Name: fare_amount, Length: 66000, dtype: float64
```

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

In [191…
```
from sklearn.metrics import r2_score
```

In [192…
```
r2_score(y_test,prediction)
```

Out[192…
```
0.6651880468683617
```

In [193…
```
from sklearn.metrics import mean_squared_error
```

In [194…
```
MSE = mean_squared_error(y_test,prediction)
```

In [195…
```
MSE
```

Out[195…
```
9.961516917717704
```

In [196…
```
RMSE = np.sqrt(MSE)
```

In [197…
```
RMSE
```

Out[197…
```
3.156187085348032
```

Random Forest Regression

In [198…
```
from sklearn.ensemble import RandomForestRegressor
```

In [199…
```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of tree
```

In [200…
```
rf.fit(X_train,y_train)
```

Out[200…
```
RandomForestRegressor()
```

In [201…
```
y_pred = rf.predict(X_test)
```

In [202…
```
y_pred
```

Out[202…
```
array([ 5.714 , 10.285 , 12.68  , ...,  6.338 , 19.4685,  7.712 ])
```

Metrics evaluatin for Random Forest

In [210…
```
R2_Random = r2_score(y_test,y_pred)
```

In [211…
```
R2_Random
```

Out[211…
```
0.7948374920410631
```

In [205…
```
MSE_Random = mean_squared_error(y_test,y_pred)
```

In [206…
```
MSE_Random
```

Out[206…
```
6.104112397417331
```

In [207…
```
RMSE_Random = np.sqrt(MSE_Random)
```

In [208…
```
RMSE_Random
```

Out[208…
```
2.4706501972997574
```