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	ORDERDATE	STATUS	QTR_ID	MONT
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	

5 rows × 25 columns

In [202]:

```
df.shape
```

Out[202]:

(2823, 25)

In [203]:

```
df.describe()
```

Out[203]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	28
mean	10258.725115	35.092809	83.658544	6.466171	3553.889072	2.717676	7.092455	20
std	92.085478	9.741443	20.174277	4.225841	1841.865106	1.203878	3.656633	
min	10100.000000	6.000000	26.880000	1.000000	482.130000	1.000000	1.000000	20
25%	10180.000000	27.000000	68.860000	3.000000	2203.430000	2.000000	4.000000	20
50%	10262.000000	35.000000	95.700000	6.000000	3184.800000	3.000000	8.000000	20
75%	10333.500000	43.000000	100.000000	9.000000	4508.000000	4.000000	11.000000	20
max	10425.000000	97.000000	100.000000	18.000000	14082.800000	4.000000	12.000000	20

In [204]:

```
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]:

```
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
```

```
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',
'PHONE', 'STATE', 'CONTACTFIRSTNAME', 'CONTACTLASTNAME', 'CUSTOMERNAME', 'ORDERNUMBER']
df = df.drop(df_drop, axis=1) #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.
```

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 o
f countries.
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
cluded.
```

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
```

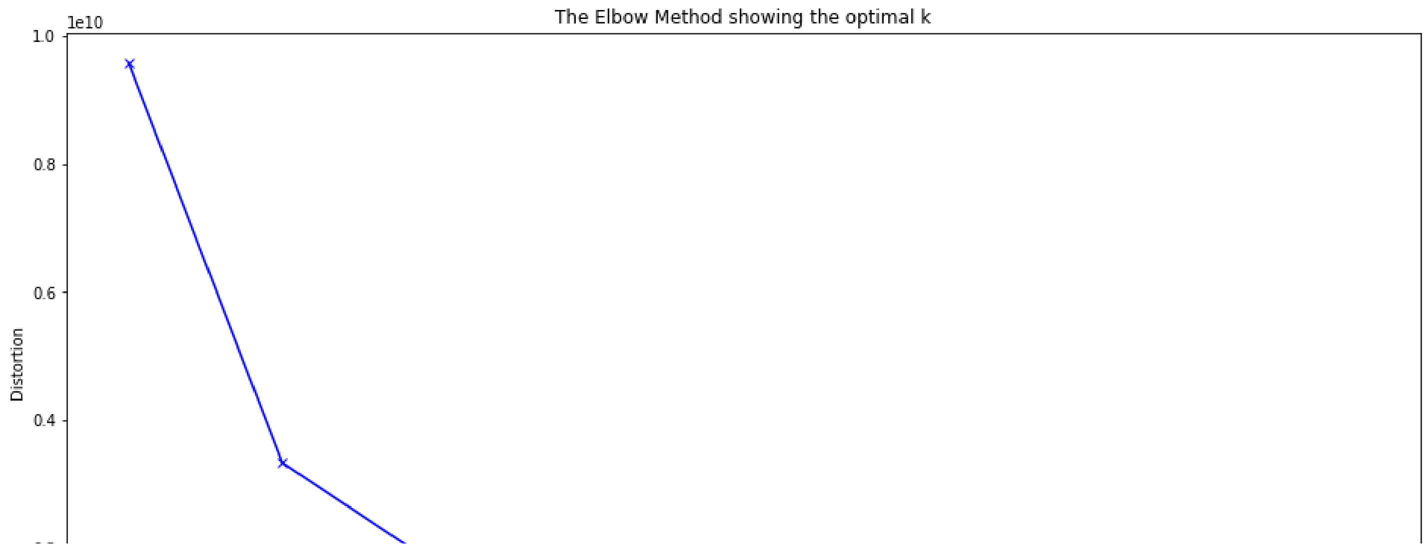
Plotting the Elbow Plot to determine the number of clusters.

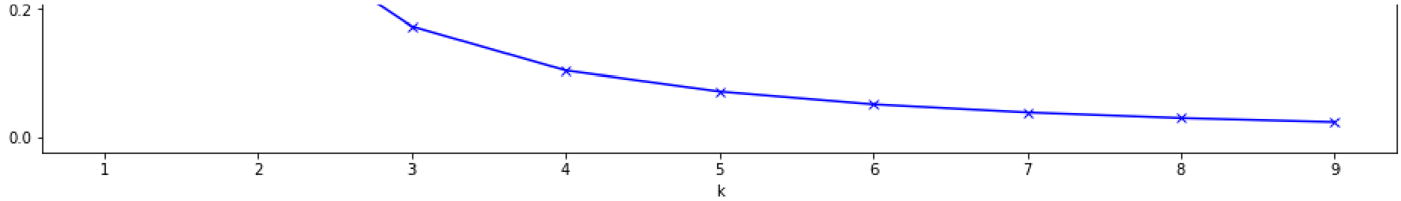
In [219]:

```
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 to visualize using Principal COmponent Analysis.
```

In [230]:

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

In [231]:

```
reduced_X.head()
```

Out[231]:

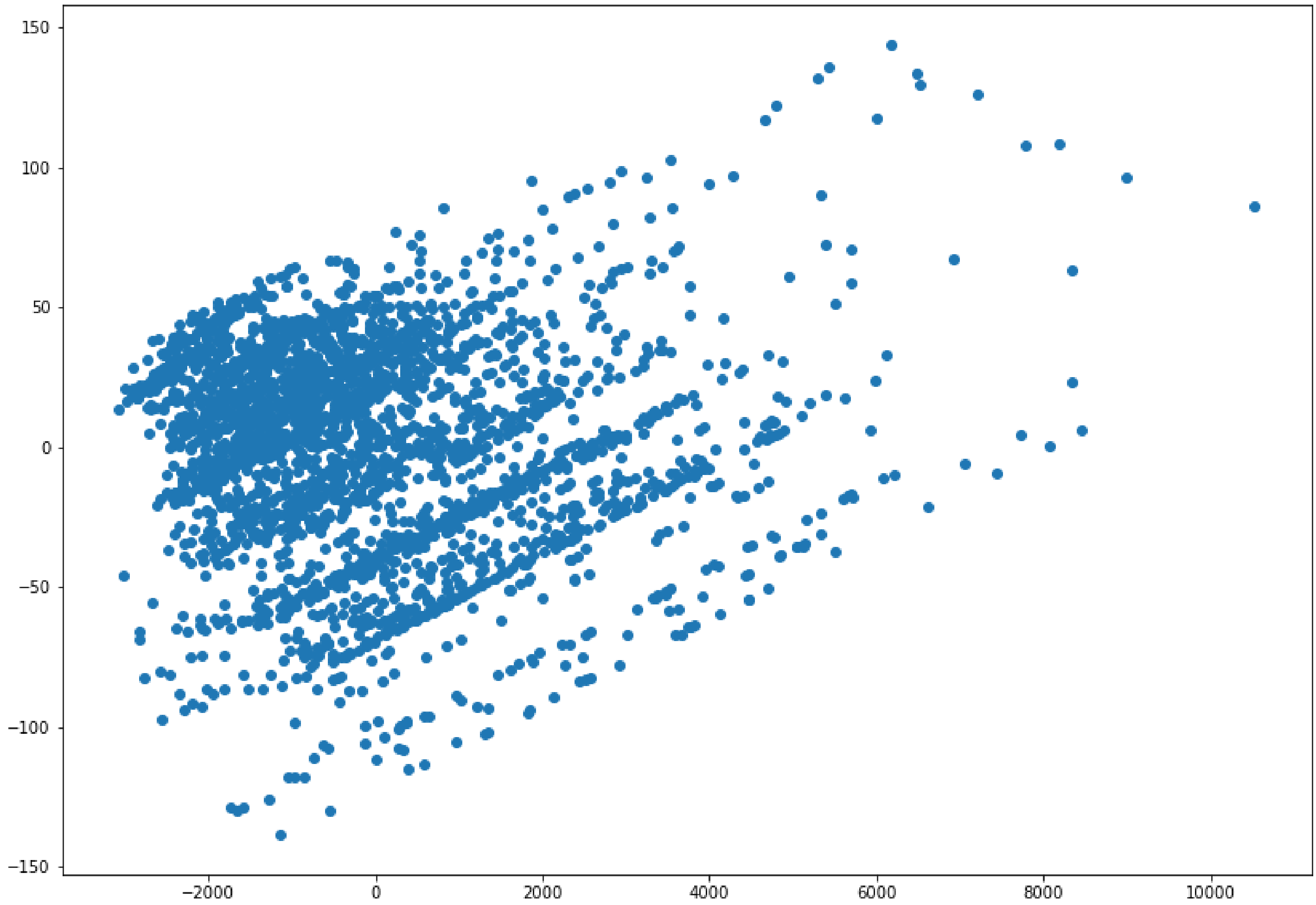
	PCA1	PCA2
0	-682.488323	-42.819535
1	-787.665502	-41.694991
2	330.732170	-26.481208
3	193.040232	-26.285766
4	1651.532874	-6.891196

In [232]:

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

Out[232]:

<matplotlib.collections.PathCollection at 0x218dc747880>



In [233]:

```
model.cluster_centers_ #Finding the centriods. (3 Centriods in total. Each Array contains
a centroids for particular feature )
```

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]:

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

In [235]:

```python
reduced_centers
```

Out[235]:

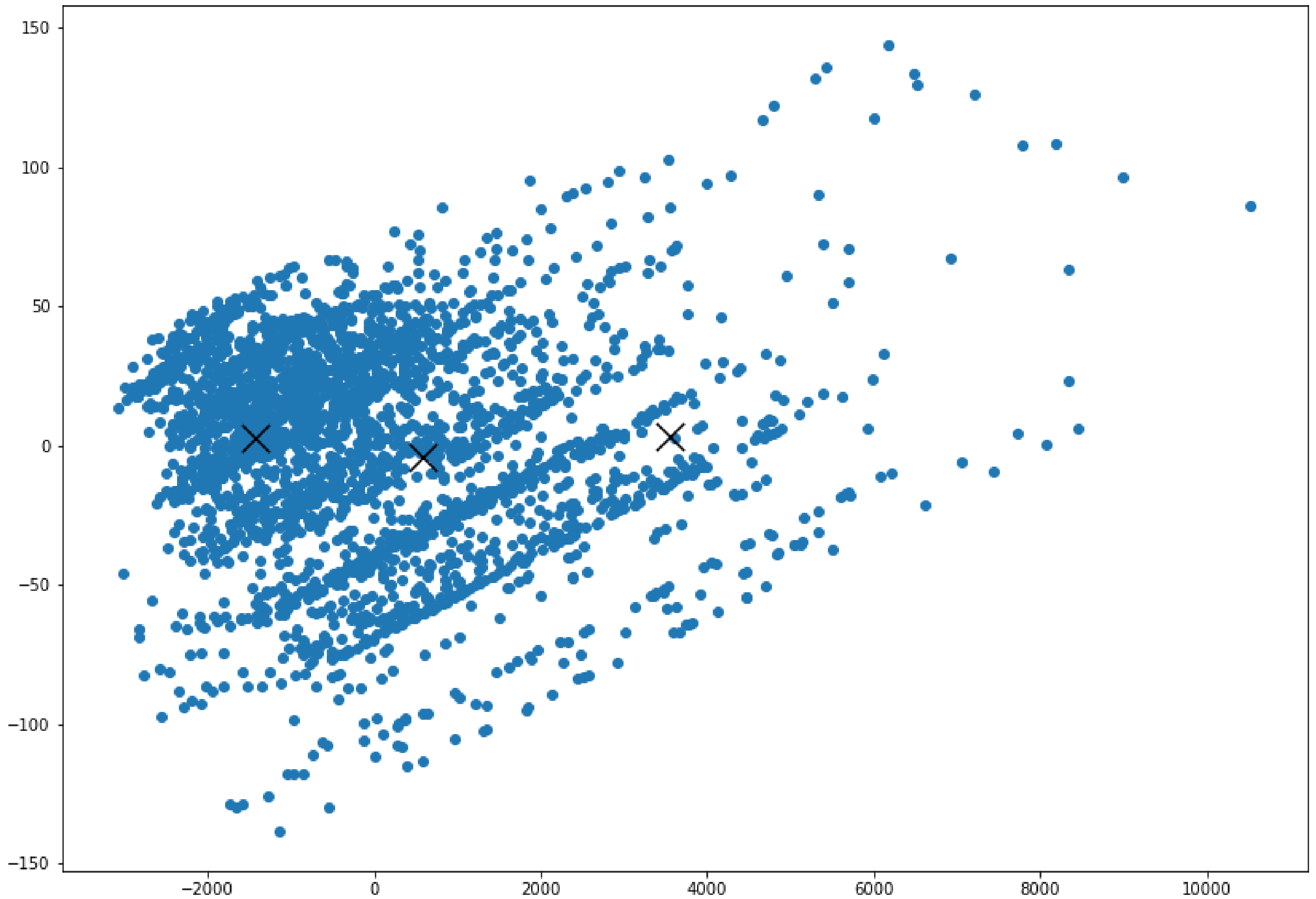
```
array([[ 5.84994044e+02, -4.36786931e+00],
       [-1.43005891e+03,  2.60041009e+00],
       [ 3.54247180e+03,  3.15185487e+00]])
```

In [236]:

```python
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) #Plotting the centriods
```

Out[236]:

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



In [237]:

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

In [238]:

```python
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

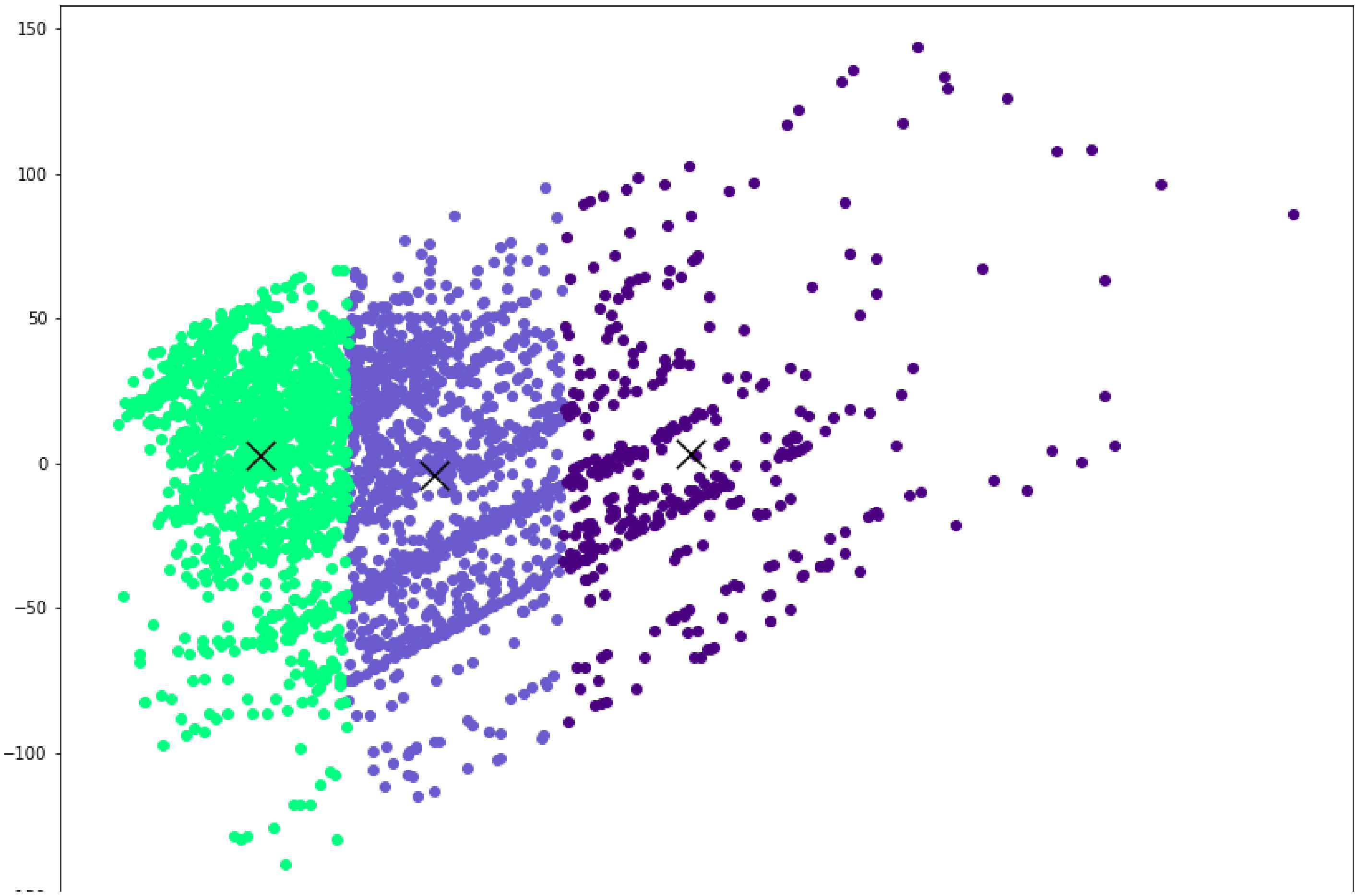
In [239]:

```python
#Plotting the clusters
plt.figure(figsize=(14,10))
#                     taking the cluster number and first column           taking the same 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)
```

Out[239]:

<matplotlib.collections.PathCollection at 0x218dce9e1f0>



In [ ]: