Assignment No.1-->Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
data = pd.read_csv('/content/uber.csv')

# Perform data preprocessing
data = data.dropna()  # Remove rows with missing values

# Split the dataset into features (X) and the target variable (y)
y = data['fare_amount']
X = data.drop(['fare_amount','pickup_datetime','key'], axis=1)


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Standardize/normalize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred_linear = linear_reg.predict(X_test)

# Random Forest Regression
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)
y_pred_rf = rf_reg.predict(X_test)

# Evaluate Linear Regression
r2_linear = r2_score(y_test, y_pred_linear)
rmse_linear = np.sqrt(mean_squared_error(y_test, y_pred_linear))
print("Linear Regression:")
print(f"R2 Score: {r2_linear:.2f}")
print(f"RMSE: {rmse_linear:.2f}")
```

```
# Evaluate Random Forest Regression
r2_rf = r2_score(y_test, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print("\nRandom Forest Regression:")
print(f"R2 Score: {r2_rf:.2f}")
print(f"RMSE: {rmse_rf:.2f}")
```

```
Linear Regression:
R2 Score: 0.00
RMSE: 9.54

Random Forest Regression:
R2 Score: 0.68
RMSE: 5.42
```

Assignment 2--> Classify the email using the binary classification method.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix


data = pd.read_csv('/content/emails.csv')
data = data.dropna()
X = data.drop(['Prediction','Email No.'], axis=1)
y = data['Prediction']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# K-Nearest Neighbors Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# Support Vector Machine Classifier
svm = SVC(kernel='linear', C=1.0)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

# Evaluation metrics
```

```
def evaluate_model(y_true, y_pred, model_name):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)

    print(f"{model_name} Performance:")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 Score: {f1:.2f}")

    cm = confusion_matrix(y_true, y_pred)
    print("Confusion Matrix:")
    print(cm)

evaluate_model(y_test, y_pred_knn, "K-Nearest Neighbors")
evaluate_model(y_test, y_pred_svm, "Support Vector Machine")
```

```
K-Nearest Neighbors Performance:
Accuracy: 0.60
Precision: 0.40
Recall: 0.89
F1 Score: 0.55
Confusion Matrix:
[[88 89]
 [ 7 59]]
Support Vector Machine Performance:
Accuracy: 0.91
Precision: 0.80
Recall: 0.89
F1 Score: 0.84
Confusion Matrix:
[[162  15]
 [  7  59]]
```

Assignment no.3 -->: Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months

```
import pandas as pd

# Load the dataset
data = pd.read_csv('/content/Churn_Modelling.csv')

# Display the first few rows of the dataset to get an overview
print(data.head())
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age
\
0          1    15634602  Hargrave          619    France  Female   42
```

```
1           2    15647311      Hill          608      Spain  Female   41

2           3    15619304      Onio          502     France  Female   42

3           4    15701354      Boni          699     France  Female   39

4           5    15737888  Mitchell          850      Spain  Female   43


   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0

# Assuming that 'Exited' is the target variable
X = data.drop(['Exited','Geography','Gender','Surname'], axis=1)  #
Features
y = data['Exited']  # Target

# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler to the training data and transform the training data
X_train = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
X_test = scaler.transform(X_test)

import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import accuracy_score, confusion_matrix

# Initialize the neural network model
model = keras.Sequential([
```

```
    keras.layers.Dense(16, input_dim=X_train.shape[1],
activation='relu'),
    keras.layers.Dense(8, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Predict on the test data
y_pred = (model.predict(X_test) > 0.5).astype(int)

# Print the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy Score: {accuracy:.2f}")

# Print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Epoch 1/10
250/250 [==============================] - 1s 1ms/step - loss: 0.5453
- accuracy: 0.7409
Epoch 2/10
250/250 [==============================] - 0s 1ms/step - loss: 0.4515
- accuracy: 0.8049
Epoch 3/10
250/250 [==============================] - 0s 1ms/step - loss: 0.4252
- accuracy: 0.8211
Epoch 4/10
250/250 [==============================] - 0s 1ms/step - loss: 0.4021
- accuracy: 0.8372
Epoch 5/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3846
- accuracy: 0.8447
Epoch 6/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3733
- accuracy: 0.8493
Epoch 7/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3674
- accuracy: 0.8496
Epoch 8/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3617
- accuracy: 0.8526
Epoch 9/10
```

```
250/250 [==============================] - 0s 1ms/step - loss: 0.3597
- accuracy: 0.8546
Epoch 10/10
250/250 [==============================] - 0s 1ms/step - loss: 0.3571
- accuracy: 0.8564
63/63 [==============================] - 0s 750us/step
Accuracy Score: 0.85
Confusion Matrix:
[[1543   64]
 [ 230  163]]
```

Assignment No.4 -->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

# Load the dataset
df = pd.read_csv("/content/diabetes.csv")
X = df.drop('Outcome', axis=1)  # Features
y = df['Outcome']  # Target variable
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.neighbors import KNeighborsClassifier

k = 5  # You can adjust the value of k
knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)

# Error Rate
error_rate = 1 - accuracy

# Precision
precision = precision_score(y_test, y_pred)
```

```
# Recall
recall = recall_score(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
print(f"Accuracy: {accuracy:.2f}")
print(f"Error Rate: {error_rate:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
Confusion Matrix:
[[79 20]
 [27 28]]
Accuracy: 0.69
Error Rate: 0.31
Precision: 0.58
Recall: 0.51
```

Assignment 5

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

```
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("/content/sales_data_sample.csv", encoding="ISO-8859-
1") #Loading the dataset.
df.head()
df.shape
df.describe()
df.info()
df.isnull().sum()
df.dtypes
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 columns having null values. Can't fill the null
values
# are there are alot of null values.
df.isnull().sum()
df.dtypes
```

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

```
<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
```

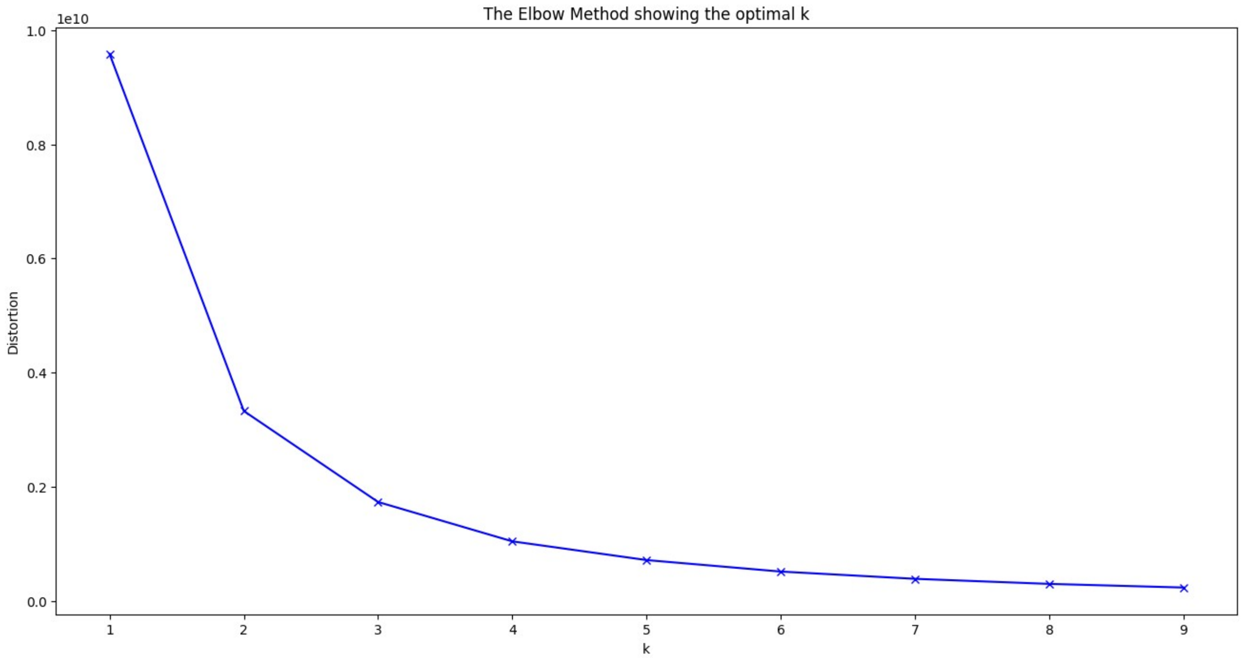
```
QUANTITYORDERED       int64
PRICEEACH           float64
ORDERLINENUMBER       int64
SALES               float64
QTR_ID                int64
MONTH_ID              int64
YEAR_ID               int64
MSRP                  int64
PRODUCTCODE            int8
Classic Cars          uint8
Motorcycles     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

Elbow plot

```
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()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870
: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
```




```
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()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

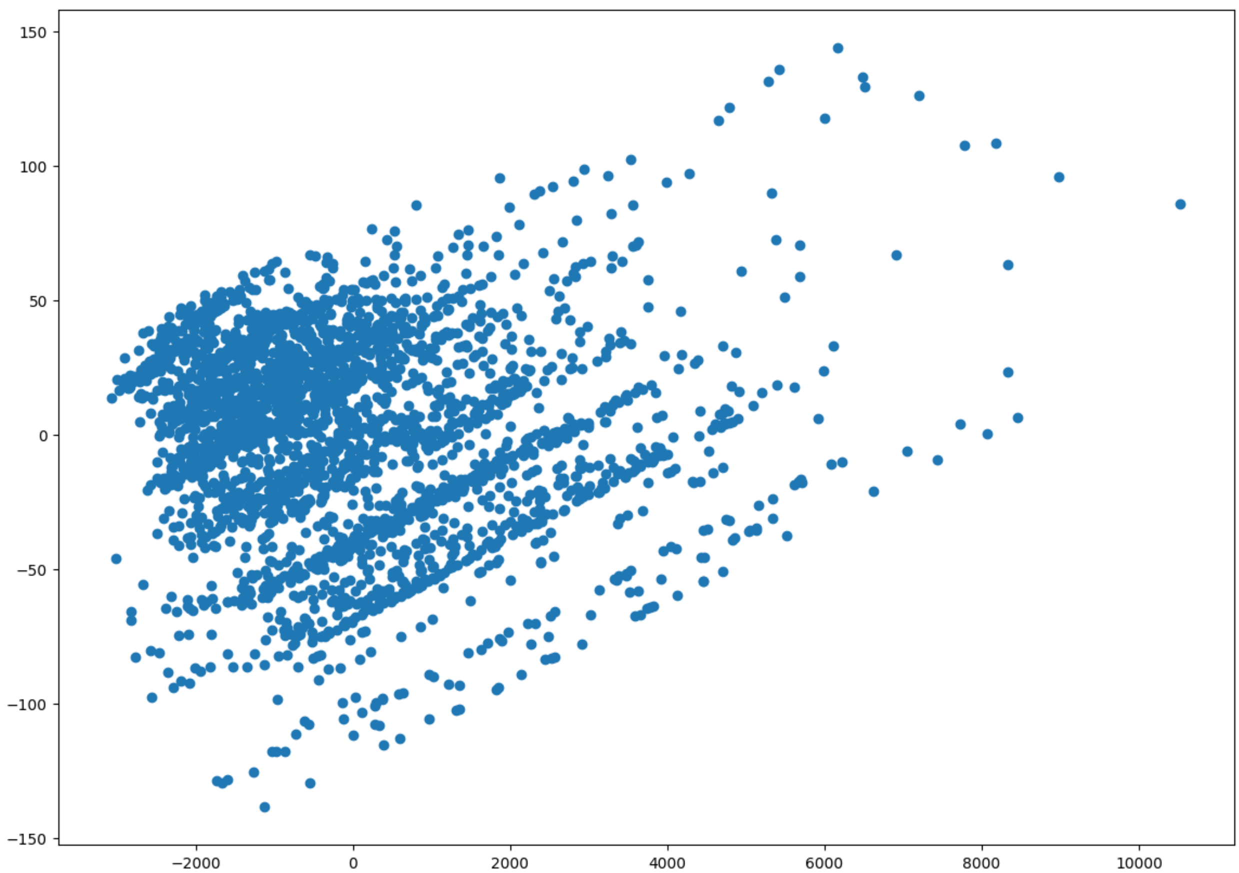
	Cluster1	Cluster2	Cluster3
0	1083	1367	373

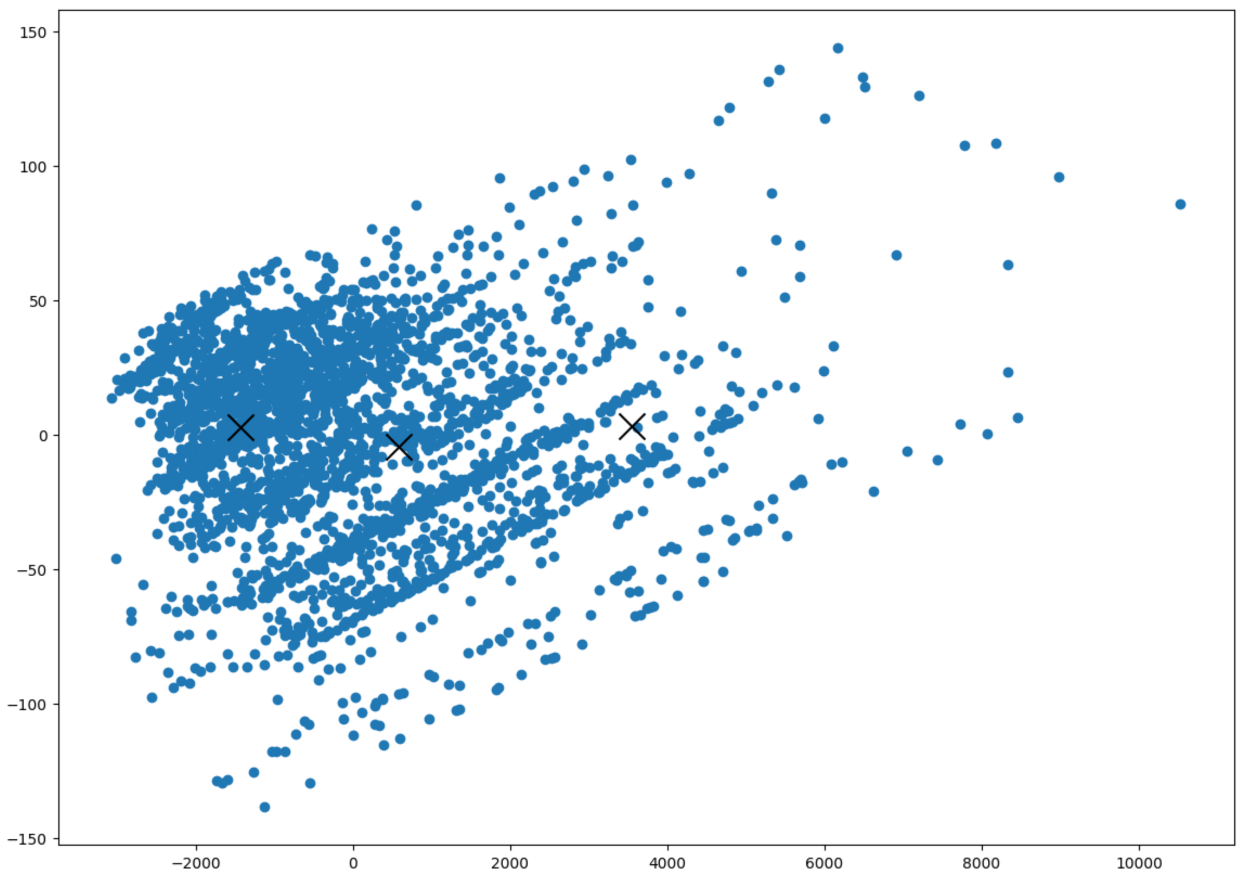
Visualization

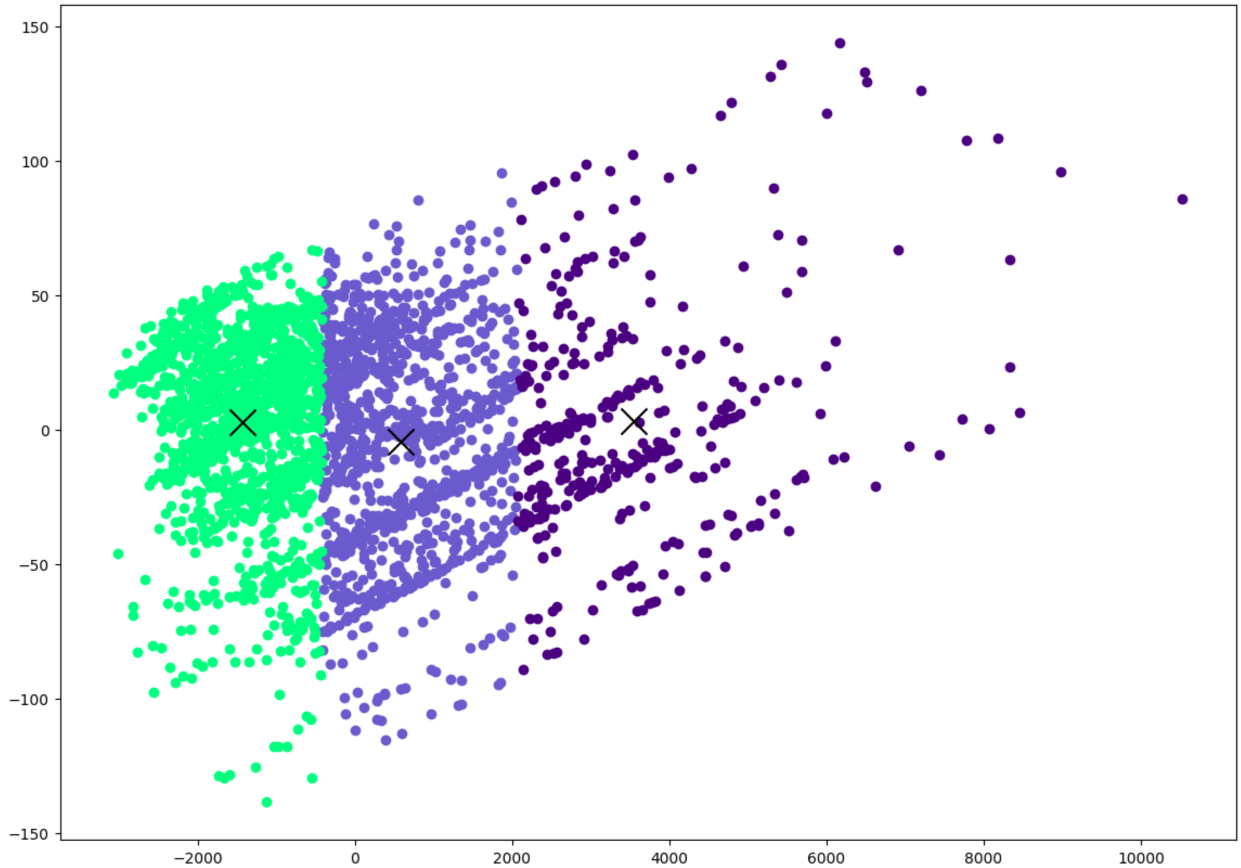
```
pca = PCA(n_components=2) #Converting all the features into 2 columns
to
# make it easy to visualize using Principal COmponent Analysis.
reduced_X
=pd.DataFrame(pca.fit_transform(X_train),columns=['PCA1','PCA2'])
#Creating
```

```
# 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
plt.figure(figsize=(14,10))
plt.scatter(reduced_X['PCA1'],reduced_X['PCA2'])
plt.scatter(reduced_centers[:,0],reduced_centers[:,1],color='black',ma
rker
='x',s=300) #Plotting the centriods
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 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',ma
rker
='x',s=300)
```

```
<matplotlib.collections.PathCollection at 0x7eb629ed97e0>
```


150
100
50
0
−50
−100
−150
−2000
0
2000
4000
6000
8000
10000


150
100
50
0
−50
−100
−150
−2000
0
2000
4000
6000
8000
10000


150
100
50
0
−50
−100
−150
−2000
0
2000
4000
6000
8000
10000