In [ ]:
```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import minmax_scale
from sklearn.model_selection import train_test_split
from keras.layers import Dropout
df=pd.read_csv("/content/drive/MyDrive/Dataset/Postures.csv")
df
```

Out[3]:

	Class	User	X0	Y0	Z0	X1	Y1	Z1	
0	0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
1	1	0	54.263880	71.466776	-64.807709	76.895635	42.462500	-72.780545	36.6
2	1	0	56.527558	72.266609	-61.935252	39.135978	82.538530	-49.596509	79.2
3	1	0	55.849928	72.469064	-62.562788	37.988804	82.631347	-50.606259	78.4
4	1	0	55.329647	71.707275	-63.688956	36.561863	81.868749	-52.752784	86.3
...	...	...	...	...	...	...	...	...	
78091	5	14	54.251127	129.177414	-44.252511	27.720784	107.810661	11.099282	-1.2
78092	5	14	54.334883	129.253842	-44.016320	27.767911	107.914808	11.069842	-30.3
78093	5	14	54.151540	129.269502	-44.173273	27.725978	108.034006	11.020347	-22.5
78094	5	14	27.915311	108.007390	10.814957	-0.910435	122.464093	-47.271248	-30.0
78095	5	14	27.898705	108.092877	11.107857	-30.031402	77.740235	-17.453099	-1.0

78096 rows × 38 columns

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

Out[4]:

	Class	User	X0	Y0	Z0	X1	
count	78096.000000	78096.000000	78096.000000	78096.000000	78096.000000	78096.000000	78(
mean	2.983738	7.959127	50.345664	85.812051	-29.984712	49.595209	
std	1.421183	4.697810	32.696173	40.204363	34.361918	32.478238	
min	0.000000	0.000000	-108.552739	-98.233756	-126.770872	-111.685241	
25%	2.000000	5.000000	29.295062	63.494432	-56.356438	28.755137	
50%	3.000000	9.000000	54.619964	86.526246	-30.864125	54.215514	
75%	4.000000	12.000000	72.488686	113.107355	-1.418803	71.762039	
max	5.000000	14.000000	190.017835	169.175464	113.345119	188.691997	

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78096 entries, 0 to 78095
Data columns (total 38 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Class   78096 non-null  int64  
 1   User    78096 non-null  int64  
 2   X0      78096 non-null  float64
 3   Y0      78096 non-null  float64
 4   Z0      78096 non-null  float64
 5   X1      78096 non-null  float64
 6   Y1      78096 non-null  float64
 7   Z1      78096 non-null  float64
 8   X2      78096 non-null  float64
 9   Y2      78096 non-null  float64
 10  Z2      78096 non-null  float64
 11  X3      78096 non-null  object 
 12  Y3      78096 non-null  object 
 13  Z3      78096 non-null  object 
 14  X4      78096 non-null  object 
 15  Y4      78096 non-null  object 
 16  Z4      78096 non-null  object 
 17  X5      78096 non-null  object 
 18  Y5      78096 non-null  object 
 19  Z5      78096 non-null  object 
 20  X6      78096 non-null  object 
 21  Y6      78096 non-null  object 
 22  Z6      78096 non-null  object 
 23  X7      78096 non-null  object 
 24  Y7      78096 non-null  object 
 25  Z7      78096 non-null  object 
 26  X8      78096 non-null  object 
 27  Y8      78096 non-null  object 
 28  Z8      78096 non-null  object 
 29  X9      78096 non-null  object 
 30  Y9      78096 non-null  object 
 31  Z9      78096 non-null  object 
 32  X10     78096 non-null  object 
 33  Y10     78096 non-null  object 
 34  Z10     78096 non-null  object 
 35  X11     78096 non-null  object 
 36  Y11     78096 non-null  object 
 37  Z11     78096 non-null  object 
dtypes: float64(9), int64(2), object(27)
memory usage: 22.6+ MB
```

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

Out[6]:

	Class	User	X0	Y0	Z0	X1	Y1	Z1	X2	Y2	...	Z8	X9	Y9
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
78091	False	False	False	False	False	False	False	False	False	False	...	False	False	False
78092	False	False	False	False	False	False	False	False	False	False	...	False	False	False
78093	False	False	False	False	False	False	False	False	False	False	...	False	False	False
78094	False	False	False	False	False	False	False	False	False	False	...	False	False	False
78095	False	False	False	False	False	False	False	False	False	False	...	False	False	False

78096 rows × 38 columns

In [ ]:
```
pd.set_option('display.max_columns', None)
```

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

Out[8]:

	Class	User	X0	Y0	Z0	X1	Y1	Z1	X2
0	0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	1	0	54.263880	71.466776	-64.807709	76.895635	42.462500	-72.780545	36.621229
2	1	0	56.527558	72.266609	-61.935252	39.135978	82.538530	-49.596509	79.223743
3	1	0	55.849928	72.469064	-62.562788	37.988804	82.631347	-50.606259	78.451526
4	1	0	55.329647	71.707275	-63.688956	36.561863	81.868749	-52.752784	86.320630

In [ ]:
```
pd.set_option('display.max_rows',None)
```

In [ ]:
```
#Converting valuesof columns to numeric
for val in list(df.columns.values):
    df[val] = pd.to_numeric(df[val], errors='coerce')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 78096 entries, 0 to 78095
Data columns (total 38 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Class   78096 non-null  int64  
 1   User    78096 non-null  int64  
 2   X0      78096 non-null  float64
 3   Y0      78096 non-null  float64
 4   Z0      78096 non-null  float64
 5   X1      78096 non-null  float64
 6   Y1      78096 non-null  float64
 7   Z1      78096 non-null  float64
 8   X2      78096 non-null  float64
 9   Y2      78096 non-null  float64
 10  Z2      78096 non-null  float64
 11  X3      77406 non-null  float64
 12  Y3      77406 non-null  float64
 13  Z3      77406 non-null  float64
 14  X4      74976 non-null  float64
 15  Y4      74976 non-null  float64
 16  Z4      74976 non-null  float64
 17  X5      65073 non-null  float64
 18  Y5      65073 non-null  float64
 19  Z5      65073 non-null  float64
 20  X6      52248 non-null  float64
 21  Y6      52248 non-null  float64
 22  Z6      52248 non-null  float64
 23  X7      38944 non-null  float64
 24  Y7      38944 non-null  float64
 25  Z7      38944 non-null  float64
 26  X8      30564 non-null  float64
 27  Y8      30564 non-null  float64
 28  Z8      30564 non-null  float64
 29  X9      23968 non-null  float64
 30  Y9      23968 non-null  float64
 31  Z9      23968 non-null  float64
 32  X10     14753 non-null  float64
 33  Y10     14753 non-null  float64
 34  Z10     14753 non-null  float64
 35  X11     32 non-null     float64
 36  Y11     32 non-null     float64
 37  Z11     32 non-null     float64
dtypes: float64(36), int64(2)
memory usage: 22.6 MB
```

In [ ]:
```
#filling missing data
df=df.fillna(df.mean())
```

In [ ]:
```
df.head(20)
```

Out[13]:

	Class	User	X0	Y0	Z0	X1	Y1	Z1	X2
0	0	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	1	0	54.263880	71.466776	-64.807709	76.895635	42.462500	-72.780545	36.621229
2	1	0	56.527558	72.266609	-61.935252	39.135978	82.538530	-49.596509	79.223743
3	1	0	55.849928	72.469064	-62.562788	37.988804	82.631347	-50.606259	78.451526
4	1	0	55.329647	71.707275	-63.688956	36.561863	81.868749	-52.752784	86.320630
5	1	0	55.142401	71.435607	-64.177303	36.175818	81.556874	-53.475747	76.986143
6	1	0	55.581184	71.641201	-63.703137	34.850565	81.352041	-54.747444	77.078512
7	1	0	34.522824	81.457318	-54.900995	55.827687	71.878788	-63.194368	86.902653
8	1	0	61.621550	10.968187	-69.134037	32.678173	81.172874	-56.994362	86.732368
9	1	0	61.401356	11.014961	-69.379418	32.527643	81.127660	-57.092473	86.421066
10	1	0	61.436613	10.992838	-69.354632	32.514926	81.082434	-57.074702	87.074330
11	1	0	61.439155	10.923622	-69.376181	32.441733	80.996422	-57.187893	86.817452
12	1	0	61.357086	10.701641	-69.449870	32.316252	80.790833	-57.426721	86.641224
13	1	0	31.375295	80.525047	-58.374065	62.815147	10.783766	-67.757236	86.828482
14	1	0	77.147708	42.547460	-72.490492	62.669492	10.684163	-67.953821	31.274939
15	1	0	77.214339	42.342892	-72.534724	62.189398	10.350602	-68.534253	31.055552
16	1	0	77.252374	42.353243	-72.567871	62.219175	10.319057	-68.511526	86.307759
17	1	0	77.331872	42.247372	-72.477275	62.072102	10.168441	-68.580401	86.207234
18	1	0	77.107568	42.109706	-72.729701	61.960891	10.225830	-68.610392	86.424993
19	1	0	77.249572	42.290797	-72.539226	86.196489	67.889384	-72.831886	31.115795

In [ ]:
```
#keep only first class
dataframe=df
dataframe=dataframe.loc[dataframe['Class'] == 1]

dataframe=dataframe.drop(['Class'], axis=1)
```

In [ ]:
```
#normalize data
listValuesToNormalize=list(dataframe.columns.values)
listValuesToNormalize.remove('User')
listValuesToNormalize
dataframe[listValuesToNormalize] = minmax_scale(dataframe[listValuesToNormaliz
```

In [ ]:
```
dataframe.head()
```

Out[16]:

	User	X0	Y0	Z0	X1	Y1	Z1	X2	Y2	Z
1	0	0.463067	0.673083	0.301926	0.554325	0.476545	0.363331	0.393129	0.661278	0.3327
2	0	0.472020	0.676255	0.315923	0.403797	0.664047	0.453686	0.563067	0.481644	0.2541
3	0	0.469340	0.677058	0.312865	0.399224	0.664482	0.449750	0.559987	0.483109	0.2510
4	0	0.467282	0.674037	0.307377	0.393535	0.660914	0.441385	0.591376	0.598328	0.2437
5	0	0.466542	0.672959	0.304998	0.391996	0.659454	0.438567	0.554141	0.477777	0.2421

In [ ]:
```
# split the dataset into input (X) and output (Y)
dataset = dataframe.values

X = dataset[:,1:].astype(float)
Y = dataset[:,0].astype(int)
```

In [ ]:
```
# converting integers to  one hot encoded
hot_encoded_y = np_utils.to_categorical(Y)
```

In [ ]:
```
seed = 1
np.random.seed(seed)
#splitting the data into 70 and 30%
X_train, X_test, y_train, y_test = train_test_split(X, hot_encoded_y, test_siz


X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=

print("the dataset has "+str(X.shape[0])+ "samples that are splitted in:")
print("- "+str(X_train.shape[0])+"samples (training set)" )
print("- "+str(X_val.shape[0])+"samples (validation set)")
print("- "+str(X_test.shape[0])+"samples (test set)")
```

```
the dataset has 16265samples that are splitted in:
- 8538samples (training set)
- 2847samples (validation set)
- 4880samples (test set)
```

In [ ]:
```
# creating model
model = Sequential()
model.add(Dense(12, input_dim=36, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.7))
model.add(Dense(15, activation='softmax'))
```

In [ ]:
```
# Compiling model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
```

In [ ]:
```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_val,y_val), epochs=140, batch_s
```

```
Epoch 1/140
427/427 [==============================] - 3s 4ms/step - loss: 0.3613 - ac
curacy: 0.0921 - val_loss: 0.2281 - val_accuracy: 0.1398
Epoch 2/140
427/427 [==============================] - 1s 3ms/step - loss: 0.2613 - ac
curacy: 0.1219 - val_loss: 0.2261 - val_accuracy: 0.1398
Epoch 3/140
427/427 [==============================] - 2s 4ms/step - loss: 0.2471 - ac
curacy: 0.1507 - val_loss: 0.2216 - val_accuracy: 0.1612
Epoch 4/140
427/427 [==============================] - 2s 4ms/step - loss: 0.2351 - ac
curacy: 0.1977 - val_loss: 0.2108 - val_accuracy: 0.2452
Epoch 5/140
427/427 [==============================] - 2s 4ms/step - loss: 0.2236 - ac
curacy: 0.2571 - val_loss: 0.2006 - val_accuracy: 0.3867
Epoch 6/140
427/427 [==============================] - 1s 3ms/step - loss: 0.2139 - ac
curacy: 0.2947 - val_loss: 0.1929 - val_accuracy: 0.3923
Epoch 7/140
```

In [ ]:
```
#test model
loss, acc =  model.evaluate(X_test, y_test, verbose=0)
print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))
```

```
Testing loss: 0.08839061856269836, acc: 0.8639343976974487
```

In [ ]:
```
#test model
loss, acc =  model.evaluate(X_test, y_test, verbose=0)
print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))
model.save('my_model_class0_predictUser.h5')
```

```
Testing loss: 0.08839061856269836, acc: 0.8639343976974487
```

In [ ]:
```
from sklearn.cluster import KMeans
```

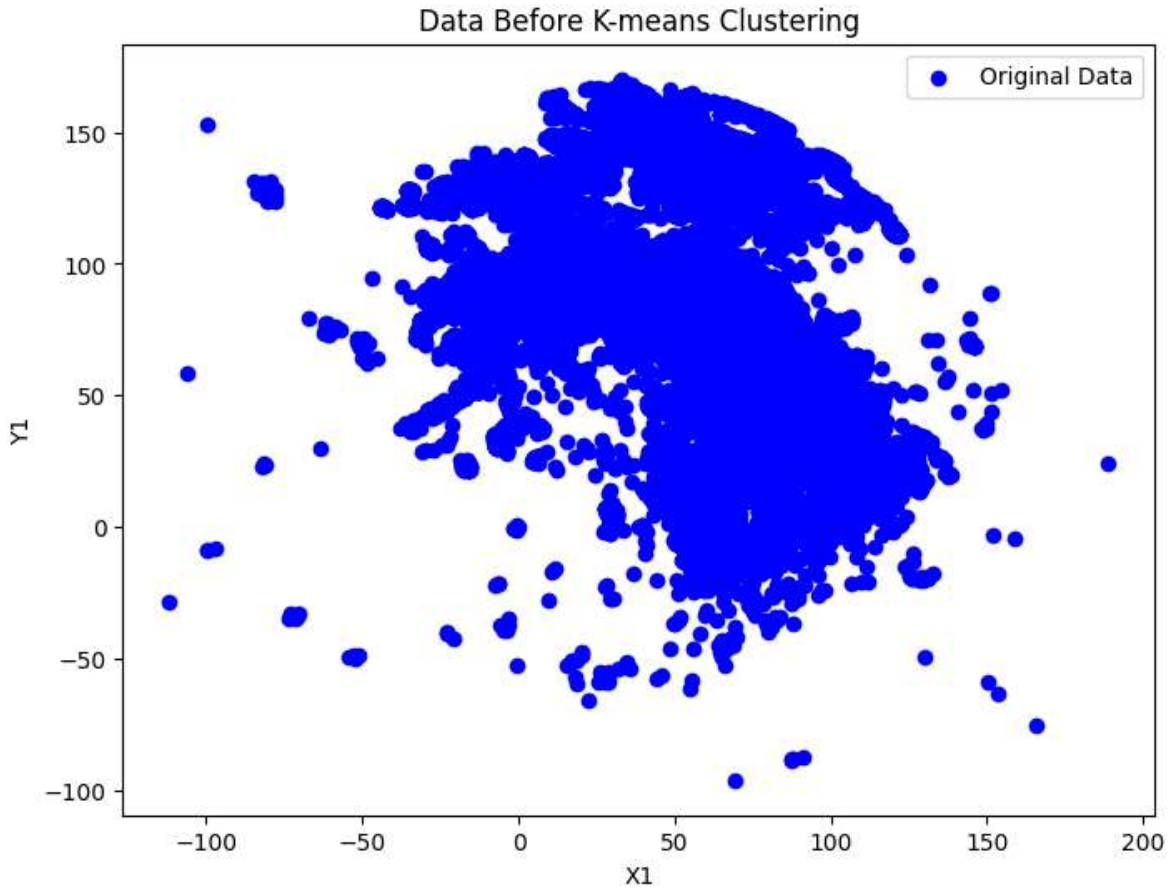
In [ ]:
```
dataframe.head(20)
```

Out[26]:

	User	X0	Y0	Z0	X1	Y1	Z1	X2	Y2	
1	0	0.463067	0.673083	0.301926	0.554325	0.476545	0.363331	0.393129	0.661278	0.332
2	0	0.472020	0.676255	0.315923	0.403797	0.664047	0.453686	0.563067	0.481644	0.254
3	0	0.469340	0.677058	0.312865	0.399224	0.664482	0.449750	0.559987	0.483109	0.251
4	0	0.467282	0.674037	0.307377	0.393535	0.660914	0.441385	0.591376	0.598328	0.243
5	0	0.466542	0.672959	0.304998	0.391996	0.659454	0.438567	0.554141	0.477777	0.242
6	0	0.468277	0.673775	0.307308	0.386713	0.658496	0.433611	0.554510	0.478345	0.242
7	0	0.384987	0.712708	0.350198	0.470338	0.614174	0.400691	0.593698	0.598787	0.246
8	0	0.492168	0.433127	0.280845	0.378053	0.657658	0.424854	0.593018	0.598765	0.245
9	0	0.491297	0.433313	0.279650	0.377453	0.657446	0.424472	0.591777	0.599221	0.244
10	0	0.491437	0.433225	0.279770	0.377402	0.657235	0.424541	0.594382	0.599563	0.247
11	0	0.491447	0.432951	0.279665	0.377110	0.656832	0.424100	0.593358	0.599735	0.245
12	0	0.491122	0.432070	0.279306	0.376610	0.655870	0.423169	0.592655	0.599248	0.244
13	0	0.372538	0.709011	0.333275	0.498194	0.328331	0.382908	0.593402	0.599208	0.245
14	0	0.553577	0.558380	0.264490	0.497613	0.327865	0.382142	0.371803	0.655453	0.306
15	0	0.553841	0.557569	0.264275	0.495699	0.326304	0.379880	0.370927	0.654049	0.304
16	0	0.553991	0.557610	0.264113	0.495818	0.326157	0.379968	0.591325	0.597762	0.241
17	0	0.554305	0.557190	0.264555	0.495231	0.325452	0.379700	0.590924	0.597073	0.240
18	0	0.553418	0.556644	0.263325	0.494788	0.325721	0.379583	0.591792	0.596798	0.241
19	0	0.553980	0.557362	0.264253	0.591403	0.595509	0.363130	0.371168	0.654134	0.305
20	0	0.553959	0.557972	0.264498	0.590871	0.595944	0.363085	0.475845	0.615777	0.294

In [ ]:
```
# Select the features for clustering let select x1 and y1
X = df[["X1", "Y1"]].values
```

In [ ]:
```
# Createinga scatter plot of the original data
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c='blue', label='Original Data')
plt.xlabel("X1")
plt.ylabel("Y1")
plt.title("Data Before K-means Clustering")
plt.legend()
plt.show()
```



In [ ]:
```
# Performig clustering using K-means algorithm
kmeans = KMeans(n_clusters=5)  # It Replace 5 with the desired number of clust
kmeans.fit(X)
```

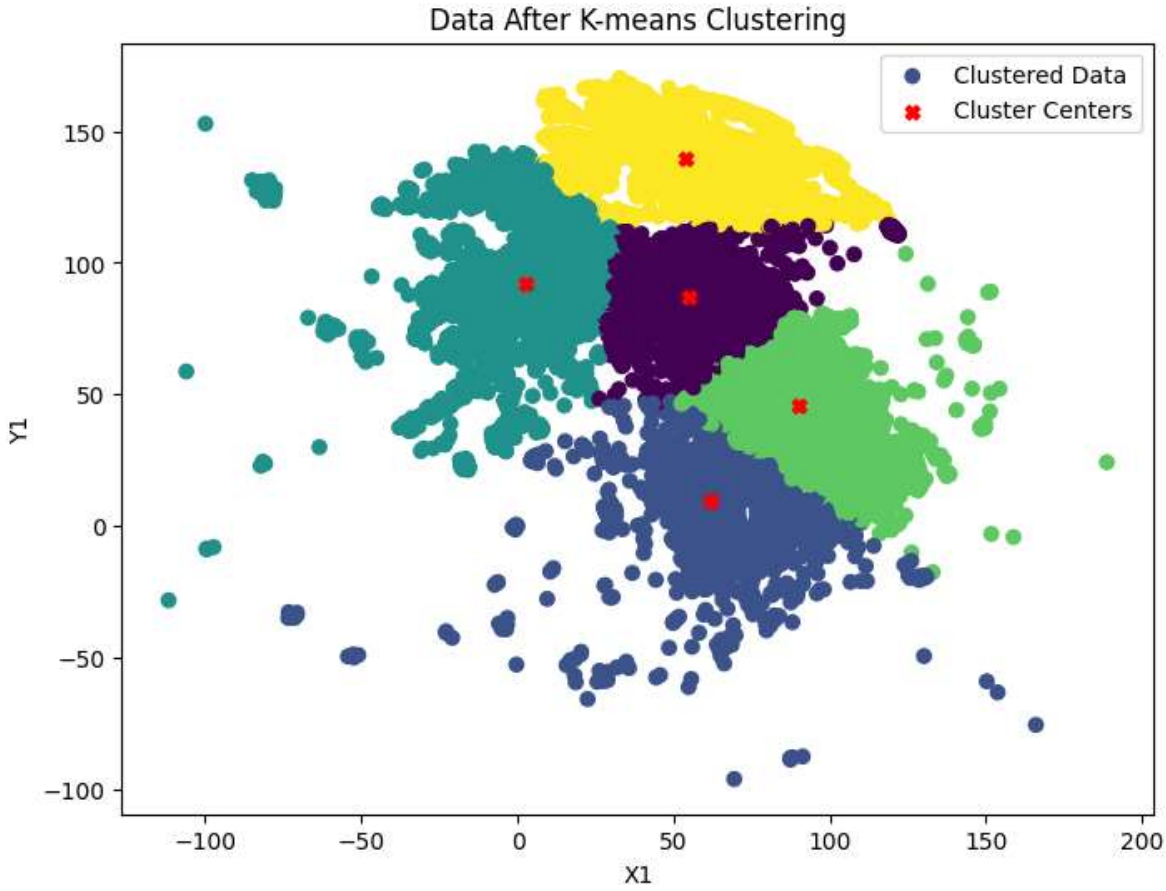
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: Futur
eWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4.
Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

Out[29]:
```
KMeans(n_clusters=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
# Get the cluster labels
cluster_labels = kmeans.labels_
```

In [ ]:
```
# Creating a scatter plot of the clustered data
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis', label='Cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c='r
plt.xlabel("X1")
plt.ylabel("Y1")
plt.title("Data After K-means Clustering")
plt.legend()
plt.show()
```



In [ ]:

In [ ]: