Essential library download and import , and ignore the warnings

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

Load The dataset

```
df=pd.read_csv("chimpanzee.csv")

df.head()
```

```
                                            sequence  class
0  ATGCCCCAACTAAATACCGCCGTATGACCCACCATAATTACCCCCA...      4
1  ATGAACGAAAATCTATTCGCTTCATTCGCTGCCCCCACAATCCTAG...      4
2  ATGGCCTCGCGCTGGTGGCGGTGGCGACGCGGCTGCTCCTGGAGGC...      4
3  ATGGCCTCGCGCTGGTGGCGGTGGCGACGCGGCTGCTCCTGGAGGC...      4
4  ATGGGCAGCGCCAGCCCGGGTCTGAGCAGCGTGTCCCCCAGCCACC...      6
```

```
df.shape
```

```
(1682, 2)
```

Check the missing value

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

```
sequence    0
class       0
dtype: int64
```

there is no missing value finding

Encode the main sequece of DNA

```
from sklearn.preprocessing import LabelEncoder

def encode_sequence_label(seq):
    le = LabelEncoder()
    return le.fit_transform(list(seq))
```

```
df['encoded'] = df['sequence'].apply(encode_sequence_label)

df1 = pd.DataFrame(df['encoded'].to_list())

df1

      0      1      2      3      4      5      6      7      8      9
\
0         0      3      2      1      1      1      1      0      0
1
1         0      3      2      0      0      1      2      0      0
0
2         0      3      2      2      1      1      3      1      2
1
3         0      3      2      2      1      1      3      1      2
1
4         0      3      2      2      2      1      0      2      1
2
...     ...    ...    ...    ...    ...    ...    ...    ...    ...
...
1677      0      3      2      1      3      2      0      2      1
2
1678      0      3      2      1      3      2      0      2      1
2
1679      0      3      2      0      0      2      1      2      0
1
1680      0      3      2      0      1      3      2      2      0
0
1681      0      3      2      3      3      2      1      1      1
0

      ...  18911  18912  18913  18914  18915  18916  18917  18918
18919  \
0     ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
1     ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
2     ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
3     ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
4     ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
...   ...    ...    ...    ...    ...    ...    ...    ...    ...     .
..
1677  ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
1678  ...    NaN    NaN    NaN    NaN    NaN    NaN    NaN    NaN
NaN
```

```
1679  ...     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
NaN
1680  ...     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
NaN
1681  ...     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
NaN

      18920
0       NaN
1       NaN
2       NaN
3       NaN
4       NaN
...     ...
1677    NaN
1678    NaN
1679    NaN
1680    NaN
1681    NaN

[1682 rows x 18921 columns]
```

```
df1.shape
```

```
(1682, 18921)
```

Split the datset for build the model

```
x=df1.fillna(0)
y=df["class"]

x_train, x_test, y_train, y_test = train_test_split(x,
y,train_size=0.70, test_size=0.2, random_state=100)

x_train.shape
```

```
(1177, 18921)
```

```
x_test.shape
```

```
(337, 18921)
```

```
y_train.shape
```

```
(1177,)
```

```
y_test.shape
```

```
(337,)
```

Build the model

```
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(x_train, y_train)

KNeighborsClassifier(n_neighbors=6)

y_pred = knn.predict(x_test)

y_pred.shape

(337,)
```

Chcek the clssification report

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.50      0.38      0.43        53
           1       0.36      0.56      0.44        36
           2       0.50      0.23      0.31        35
           3       0.75      0.19      0.30        48
           4       0.37      0.19      0.25        59
           5       0.83      0.25      0.38        20
           6       0.39      0.80      0.52        86

    accuracy                           0.42       337
   macro avg       0.53      0.37      0.38       337
weighted avg       0.49      0.42      0.39       337

[[20  6  1  0  4  0 22]
 [ 5 20  2  1  2  0  6]
 [ 1  9  8  2  4  0 11]
 [ 2  7  0  9  4  0 26]
 [ 5  3  0  0 11  0 40]
 [ 3  4  2  0  2  5  4]
 [ 4  6  3  0  3  1 69]]
```

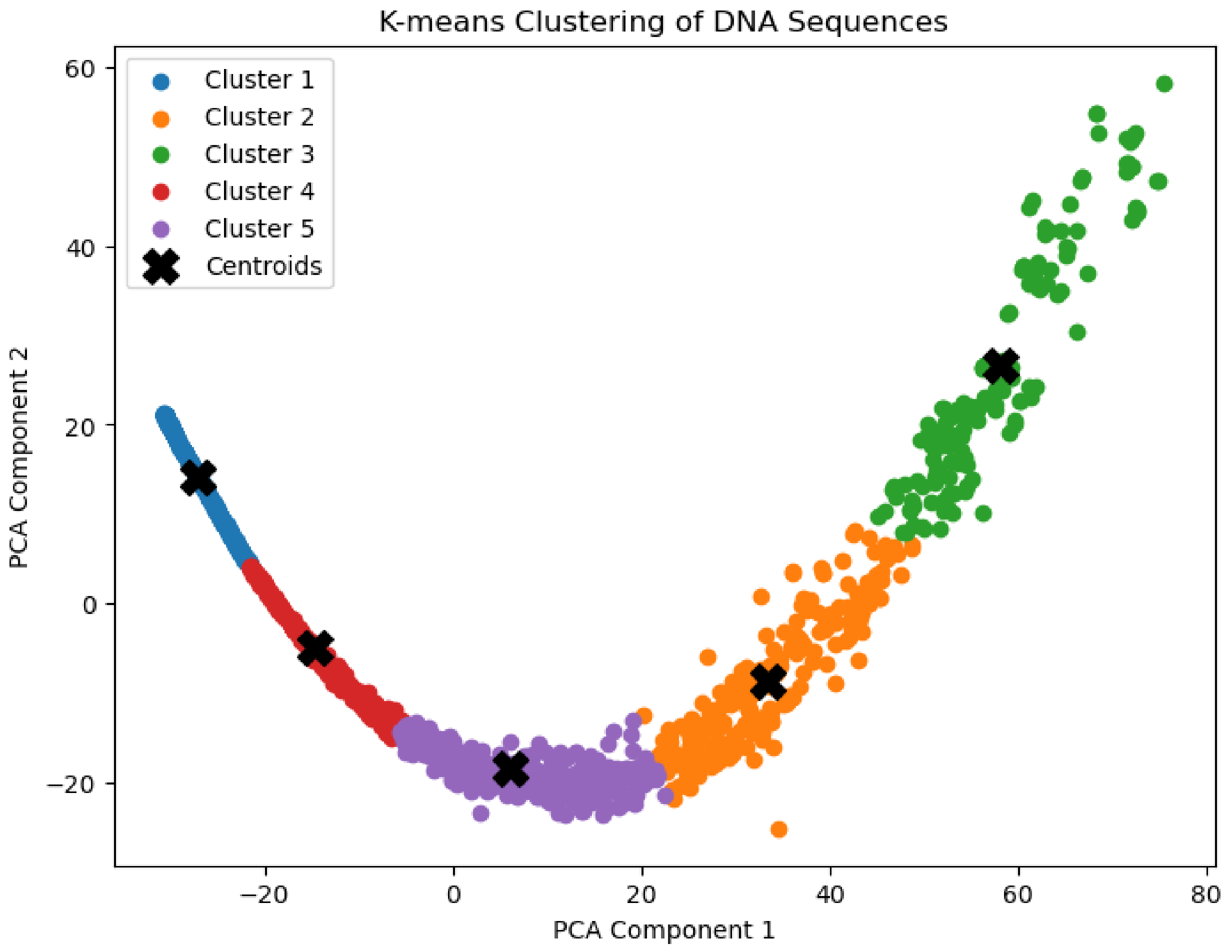
K_means Clustering

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
x_pca = pca.fit_transform(x)

k = 5
kmeans = KMeans(n_clusters=k, random_state=100)
labels = kmeans.fit_predict(x_pca)
```

```
plt.figure(figsize=(8,6))
for cluster in range(k):
    plt.scatter(x_pca[labels == cluster, 0], x_pca[labels == cluster,
1], label=f'Cluster {cluster+1}')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], s=200, c='black', marker='X', label='Centroids')
plt.title('K-means Clustering of DNA Sequences')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```



```
print("PCA Components (directions):")
print(pca.components_)

PCA Components (directions):
[[ 1.11711836e-03 -9.31690280e-04  6.02611237e-05 ...  1.72653491e-04
   0.00000000e+00  1.15102327e-04]
 [ 2.31322394e-04  5.49447341e-04  6.37676645e-04 ...  4.01712755e-04
   0.00000000e+00  2.67808503e-04]]
```

```python
print("\nExplained variance ratio (how much info each component
keeps):")
print(pca.explained_variance_ratio_)
```

```
Explained variance ratio (how much info each component keeps):
[0.2040294  0.06774289]
```