```
#importing the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#reading the dataset by calling pandas
data = pd.read_csv("rna seq .csv")
```

```
#gives the shape of the data i.e how many rows and columns it contains
data.shape
```

```
(653, 24)
```

```
#gives the head part of the data
data.head()
```

	Target seq	Start	End	Sequence	G	U	bi	uni	duplex	Pos1	...	Pos18	Dif_5-3	Content+	Content-	Cons+	Cons-
0	M60857	195	213	AUUAUCCACUGUUUUUGGA	3	9	-7.0	-1.9	-28.1	-1.1	...	-2.4	-1.3	2	6	2	6
1	M60857	197	215	AAAUUAUCCACUGUUUUUG	2	9	-0.7	0.0	-24.2	-0.9	...	-2.1	-1.2	1	6	1	5
2	M60857	199	217	CAAAAUUAUCCACUGUUUU	1	8	-1.5	0.0	-24.2	-2.1	...	-0.9	1.2	2	5	3	2
3	M60857	201	219	CACAAAAUUAUCCACUGUU	1	6	-0.6	0.0	-26.7	-2.1	...	-0.9	1.2	3	3	3	3
4	M60857	203	221	GCCACAAAAUUAUCCACUG	2	4	-0.1	0.0	-30.3	-3.4	...	-2.1	1.3	4	2	2	3

5 rows × 24 columns

```
#returns the information of the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 653 entries, 0 to 652
Data columns (total 24 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Target seq  653 non-null    object
 1   Start       653 non-null    int64
 2   End         653 non-null    int64
 3   Sequence    653 non-null    object
 4   G           653 non-null    int64
 5   U           653 non-null    int64
 6   bi          653 non-null    float64
 7   uni         653 non-null    float64
 8   duplex      653 non-null    float64
 9   Pos1        653 non-null    float64
 10  Pos2        653 non-null    float64
 11  Pos6        653 non-null    float64
 12  Pos13       653 non-null    float64
 13  Pos14       653 non-null    float64
 14  Pos18       653 non-null    float64
 15  Dif_5-3     653 non-null    float64
 16  Content+    653 non-null    int64
 17  Content-    653 non-null    int64
 18  Cons+       653 non-null    int64
 19  Cons-       653 non-null    int64
 20  Cons_Sum    653 non-null    int64
 21  Hyb19       653 non-null    float64
 22  target      653 non-null    float64
 23  Activity    653 non-null    float64
dtypes: float64(13), int64(9), object(2)
memory usage: 122.6+ KB
```

```
#describes the statistical measures of the data
data.describe()
```

	Start	End	G	U	bi	uni	duplex	Pos1	Pos2	Pos6	...	
count	653.000000	653.000000	653.000000	653.000000	653.000000	653.000000	653.000000	653.000000	653.000000	653.000000	...	653
mean	972.715161	990.715161	4.362940	5.422665	-7.446095	-1.106432	-34.111639	-2.060337	-2.086371	-2.149923	...	-2
std	769.976871	769.976871	1.975689	2.326608	4.678564	1.491547	4.365093	0.763692	0.768782	0.762777	...	(
min	1.000000	19.000000	0.000000	0.000000	-26.600000	-7.900000	-49.900000	-3.400000	-3.400000	-3.400000	...	-3
25%	329.000000	347.000000	3.000000	4.000000	-9.900000	-1.900000	-36.700000	-2.400000	-2.400000	-2.400000	...	-2
50%	779.000000	797.000000	4.000000	5.000000	-7.000000	-0.300000	-33.700000	-2.100000	-2.100000	-2.100000	...	-2
75%	1504.000000	1522.000000	6.000000	7.000000	-4.000000	0.000000	-31.000000	-1.300000	-1.300000	-2.100000	...	-2
max	5524.000000	5542.000000	12.000000	13.000000	4.100000	0.000000	-22.300000	-0.900000	-0.900000	-0.900000	...	-(

8 rows × 22 columns

```
#checking the presence of missing values
data.isnull().sum()
```

```
Target seq    0
Start         0
End           0
Sequence      0
G             0
U             0
bi            0
uni           0
duplex        0
Pos1          0
Pos2          0
Pos6          0
Pos13         0
Pos14         0
Pos18         0
Dif_5-3       0
Content+      0
Content-      0
Cons+         0
Cons-         0
Cons_Sum      0
Hyb19         0
target        0
Activity      0
dtype: int64
```

```
#the column named TARGET SEQ is dropped because it contains text and assigned to the variable rna
rna = data.drop('Target seq', axis = 1)
```

```
#the column named sequence is dropped because it contains text and assigned to the variable rna_seq
rna_seq = rna.drop('Sequence', axis = 1)
```

```
#collecting the informaton of the dataset after dropping the 2 columns
rna_seq.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 653 entries, 0 to 652
Data columns (total 22 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Start     653 non-null    int64
 1   End       653 non-null    int64
 2   G         653 non-null    int64
 3   U         653 non-null    int64
 4   bi        653 non-null    float64
 5   uni       653 non-null    float64
 6   duplex    653 non-null    float64
 7   Pos1      653 non-null    float64
 8   Pos2      653 non-null    float64
 9   Pos6      653 non-null    float64
 10  Pos13     653 non-null    float64
 11  Pos14     653 non-null    float64
 12  Pos18     653 non-null    float64
 13  Dif_5-3   653 non-null    float64
 14  Content+  653 non-null    int64
 15  Content-  653 non-null    int64
 16  Cons+     653 non-null    int64
 17  Cons-     653 non-null    int64
 18  Cons_Sum  653 non-null    int64
 19  Hyb19     653 non-null    float64
 20  target    653 non-null    float64
 21  Activity  653 non-null    float64
```

```
dtypes: float64(13), int64(9)
memory usage: 112.4 KB
```

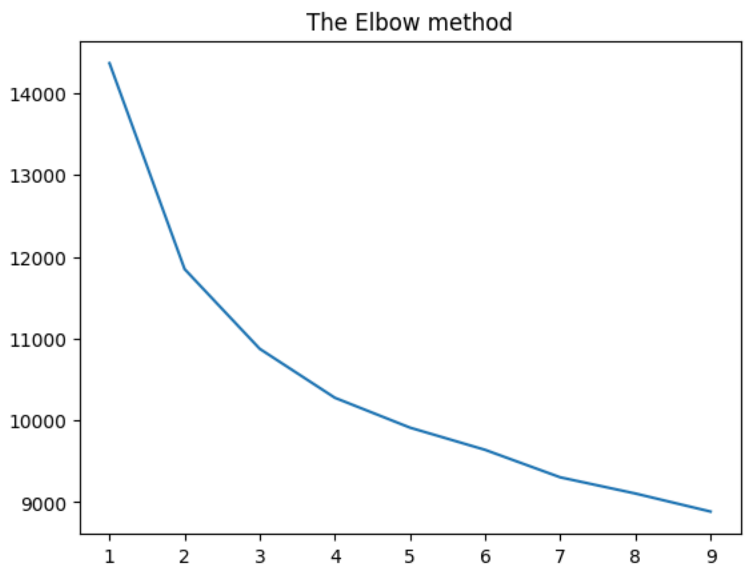
```
# scaling the data with the StandardScalar function of sklearn library
# to make mean as 0 and std. deviation as 1
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# fitting the data to standardscaler and storing the scaled data in the variable data_scaled
data_scaled = scaler.fit_transform(rna_seq)
```

```
#importing the kmeans from sklearn
from sklearn.cluster import KMeans
#this line creates a sequence of integers from 1 to 9 , representing the possible values for the number of clusters (k) to evaluate.
k_meansclus = range(1,10)
#This line initializes an empty list named wcss to store the Within-Cluster Sum of Squares (WCSS) for each k value.
#WCSS is a measure of how well data points are grouped within their assigned clusters in K-Means clustering.
wcss = []
#intializing the for loop
for k in k_meansclus:
#this line creates a new KMeans model, specifying the number of clusters (n_clusters) to be k
    km = KMeans(n_clusters = k)
#This line fits the KMeans model to the data
    km.fit(data_scaled)
#km.inertia_: After fitting the KMeans model for a specific k, this line accesses the model's inertia_ attribute. This attribute stores
#The WCSS value is then appended to the wcss list, effectively storing the WCSS for each evaluated k (number of clusters).
    wcss.append(km.inertia_)
```

```
# Within-Cluster Sum of Squares (WCSS)
wcss
```

```
[14366.000000000004,
 11848.386915668325,
 10874.064452966397,
 10275.626986045678,
 9910.629300270894,
 9639.140221903133,
 9303.259613419987,
 9105.283643636898,
 8884.604690950462]
```

```
#visualising the elbow plot
plt.title('The Elbow method')
#plotting the eblow method by clusters on x axis and wcss on y axis
plt.plot(k_meansclus, wcss)
plt.show()
```




```
#this line creates a new instance of the KMeans class
#n_clusters  parameter specifies the number of clusters we want the model to identify in the  data.
#max_iter parameter sets the maximum number of iterations the K-Means algorithm will perform. During each iteration, the algorithm refi
km1 = KMeans(n_clusters = 3, max_iter = 100, random_state = 0)
#this line fits the KMeans model (km1) to the data stored in the data_scaled variable.
km1.fit(data_scaled)
```

```
KMeans
KMeans(max_iter=100, n_clusters=3, random_state=0)
```

```
#The fit part ensures the KMeans model is properly fitted to  data (data_scaled) in case it wasn't fitted before.
#The predict part predicts the cluster labels for the data points in data_scaled
Y = km1.fit_predict(data_scaled)
Y
```

```
array([1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 2, 2, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2, 1, 1, 2, 0, 1, 0, 2, 2,
       2, 1, 2, 0, 1, 0, 0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 2, 0, 0, 2, 2, 2,
       2, 2, 2, 2, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 0, 2, 2, 2, 2, 2, 2,
       2, 1, 0, 1, 1, 0, 0, 0, 1, 2, 0, 1, 1, 0, 1, 2, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 2, 2, 0, 2, 0, 0, 2, 1, 2, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 2, 2, 1, 0, 1, 2, 1, 1, 0, 2, 0, 1, 2, 2, 2, 2, 2,
       2, 1, 2, 0, 0, 1, 0, 0, 1, 2, 1, 0, 0, 2, 0, 0, 0, 1, 1, 1, 2, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 2, 2, 2, 1, 2, 2, 2, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 2, 1, 2, 2, 0, 2,
       2, 2, 2, 0, 0, 1, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
       2, 0, 2, 2, 0, 0, 2, 0, 2, 0, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 2, 2, 0, 0, 2, 2, 2, 0, 1, 0, 1, 0, 0, 1, 2, 0, 2, 1,
       2, 0, 2, 2, 0, 0, 2, 2, 1, 1, 2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 0, 1, 1, 1, 2, 2, 1, 0, 0,
       0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 0, 1, 2, 2, 2, 2, 0,
       2, 2, 2, 1, 0, 0, 0, 2, 2, 2, 2, 0, 0, 1, 1], dtype=int32)
```

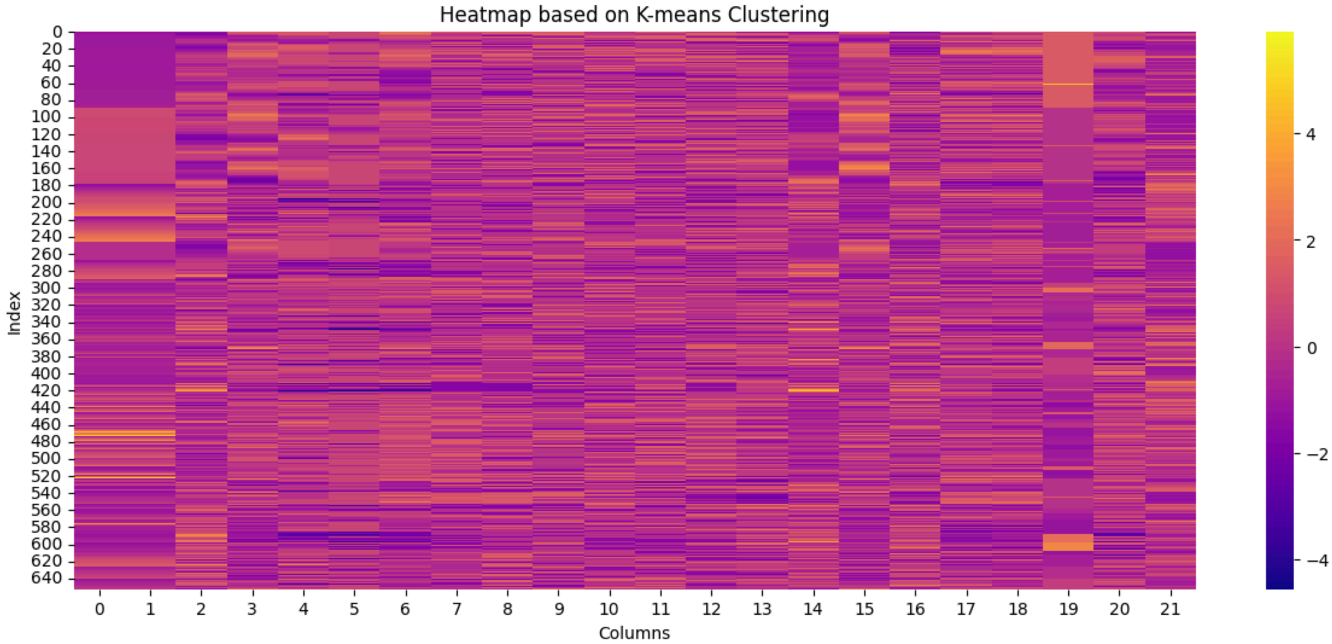
```
#this returns the centers of each clusters
km1.cluster_centers_
```

```
array([[ 3.34668680e-01,  3.34668680e-01, -1.13260426e-01,
         7.55738438e-02,  1.71252733e-01,  1.53631314e-01,
         3.21667281e-01, -2.56059590e-01, -6.25567155e-02,
         4.37246791e-04, -7.90155827e-02,  1.15562704e-02,
         5.70557530e-01,  6.09717948e-01, -1.55826387e-01,
        -9.11274595e-02,  3.32271278e-01, -4.13545952e-01,
        -4.31326995e-01, -2.16103639e-01,  2.03586862e-01,
         4.54658789e-01],
       [-1.22803536e-01, -1.22803536e-01, -5.03005599e-01,
         5.43837383e-01,  4.31060781e-01,  4.17255122e-01,
         4.88040834e-01,  5.34522711e-01,  2.88646792e-01,
         3.00167094e-01,  3.98958033e-01,  3.36162075e-01,
        -2.69396782e-01, -5.90521273e-01, -5.35600049e-01,
         6.12870023e-01, -7.69908183e-01,  8.13315686e-01,
         9.18081560e-01,  9.58812182e-02,  3.02960883e-01,
        -6.68513059e-01],
       [-3.15641031e-01, -3.15641031e-01,  9.28680499e-01,
        -9.33889657e-01, -9.06911772e-01, -8.59698163e-01,
        -1.21806664e+00, -4.23399061e-01, -3.41983265e-01,
        -4.53667443e-01, -4.83827447e-01, -5.24635585e-01,
        -4.47511834e-01, -2.14904668e-02,  1.04159049e+00,
        -7.88533285e-01,  6.64559638e-01, -6.08407647e-01,
        -7.39903540e-01,  1.78788394e-01, -7.61985101e-01,
         3.28328024e-01]])
```

```
#he using np.array which likely converts the variable data_scaled into a NumPy array, assuming data_scaled is currently not a NumPy arra
data_imp = np.array(data_scaled)
data_imp
```

```
array([[-1.01082431e+00, -1.01082431e+00, -6.90384385e-01, ...,
         1.41975683e+00,  1.09992617e+00, -5.71225347e-01],
       [-1.00822484e+00, -1.00822484e+00, -1.19692484e+00, ...,
         1.41975683e+00,  1.09992617e+00, -1.05008917e+00],
       [-1.00562537e+00, -1.00562537e+00, -1.70346529e+00, ...,
         1.41975683e+00,  6.04196784e-01,  2.57948261e-01],
       ...,
       [-9.29520157e-04, -9.29520157e-04,  1.33577743e+00, ...,
         4.17095938e-01,  1.53556714e+00, -2.08060162e-01],
       [ 4.58609721e-02,  4.58609721e-02,  3.22696521e-01, ...,
         4.17095938e-01,  5.74152579e-01, -5.61583793e-01],
       [ 1.58937995e-01,  1.58937995e-01, -1.19692484e+00, ...,
         4.17095938e-01,  9.79749346e-01,  3.38294541e-01]])
```

```
plt.figure(figsize = (15,6))#defining the size of the plot
#creates a heatmap visualization using the Seaborn library (sns) to represent the data in data_imp.
sns.heatmap(data_imp, cmap='plasma')
plt.title('Heatmap based on K-means Clustering') # title of the plot
plt.xlabel('Columns') # label on x axis
plt.ylabel('Index') # label on y axis
plt.show() # visualizing the heatmap
```



Start coding or generate with AI.