K-Means Clustering

Mohammed Meraj TY computer 32

Importing the libraries

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [21]:
```
dataset = pd.read_csv('gaming_platform_usage.csv')
```

In [22]:
```
dataset.head(10)
```

Out[22]:

	UserType	Age	Daily Play Hours	Monthly Spend ($)
0	Developer	57	2	21
1	Streamer	40	4	43
2	Casual	20	10	179
3	Competitive	13	9	172
4	Casual	28	10	135
5	Competitive	42	9	87
6	Streamer	23	3	3
7	Developer	59	5	48
8	Streamer	39	6	89
9	Competitive	46	5	240

In [23]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   UserType           3000 non-null   object
 1   Age                3000 non-null   int64 
 2   Daily Play Hours   3000 non-null   int64 
 3   Monthly Spend ($)  3000 non-null   int64 
dtypes: int64(3), object(1)
memory usage: 93.9+ KB
```

In [24]:
```
dataset.shape
```

Out[24]:
```
(3000, 4)
```

In [27]:
```
X = dataset.iloc[:,[2,3]].values
```

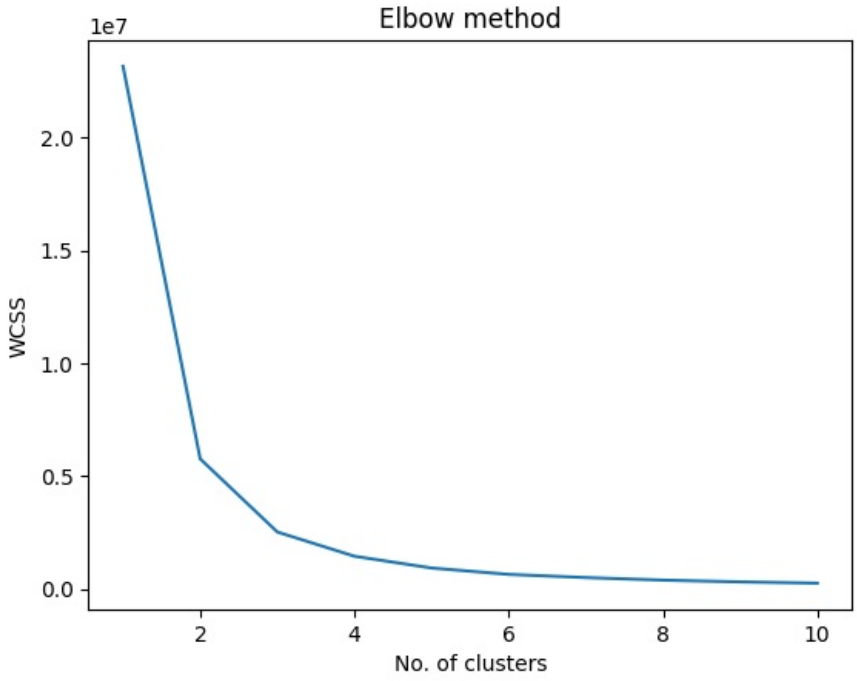
In [28]:
```
print(X)
```

```
[[  2  21]
 [  4  43]
 [ 10 179]
 ...
 [ 10  93]
 [  4 188]
 [  6  81]]
```

Using the elbow method to find the optimal number of clusters

In [42]:
```
from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
  kmeans = KMeans(n_clusters= i, init='k-means++',random_state= 42)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title("Elbow method")
```

```
plt.xlabel("No. of clusters")
plt.ylabel("WCSS")
plt.show()
```



In [ ]:

Training the K-Means model on the dataset

In [30]:
```
kmeans = KMeans(n_clusters= 5, init='k-means++',random_state= 42)
kmeans.fit(X)
```

Out[30]:
```
KMeans(n_clusters=5, random_state=42)
```

Statistics from the initialization run with the lowest SSE are available as attributes of kmeans after calling .fit()

In [31]:
```
# The lowest SSE value
kmeans.inertia_
```

Out[31]:
```
933126.6701665135
```

In [32]:
```
# Final locations of the centroid
kmeans.cluster_centers_
```

Out[32]:
```
array([[  6.82034976, 142.13831479],
       [  6.60763889,  26.03819444],
       [  6.65826772, 268.5984252 ],
       [  6.77900552,  82.40699816],
       [  6.62884927, 205.93679092]])
```

In [33]:
```
# The number of iterations required to converge
kmeans.n_iter_
```

Out[33]:
```
19
```

In [34]:
```
#Finally, the cluster assignments are stored as a one-dimensional NumPy array in kmeans.labels_
kmeans.labels_
```

Out[34]:
```
array([1, 1, 4, ..., 3, 4, 3], dtype=int32)
```

Creating Output labels for Generating Graph

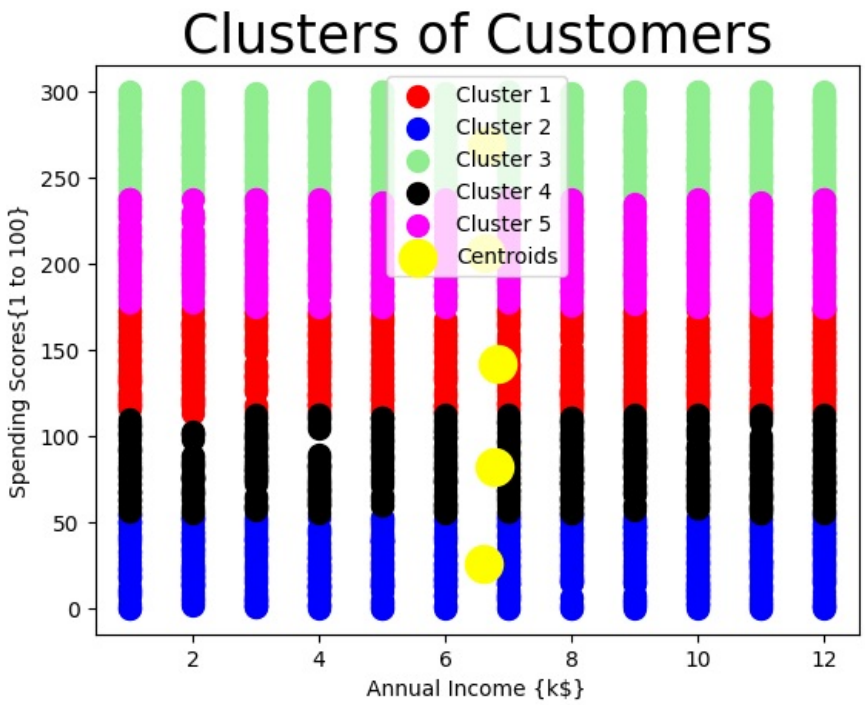
In [35]:
```
y_kmeans = kmeans.fit_predict(X)
```

In [36]:
```
print(y_kmeans)
```

```
[1 1 4 ... 3 4 3]
```

Visualising the clusters

In [37]:
```
plt.scatter(X[y_kmeans == 0,0],X[y_kmeans == 0,1],s=100, c = 'red', label ="Cluster 1")
plt.scatter(X[y_kmeans == 1,0],X[y_kmeans == 1,1],s=100, c = 'blue', label ="Cluster 2")
plt.scatter(X[y_kmeans == 2,0],X[y_kmeans == 2,1],s=100, c = 'lightgreen', label ="Cluster 3")
plt.scatter(X[y_kmeans == 3,0],X[y_kmeans == 3,1],s=100, c = 'black', label ="Cluster 4")
plt.scatter(X[y_kmeans == 4,0],X[y_kmeans == 4,1],s=100, c = 'magenta', label ="Cluster 5")
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s = 300, c = 'Yellow',label = 'Centroids'
plt.title("Clusters of Customers",size = 25)
plt.xlabel("Annual Income {k$}")
plt.ylabel("Spending Scores{1 to 100}")
plt.legend()
plt.show()
```



Internal Evaluation of Cluster

DB Score (lower is better)

In [38]:
```
from sklearn.metrics import davies_bouldin_score
davies_bouldin_score(X,y_kmeans)
```

Out[38]:
```
np.float64(0.5149092790826125)
```

External Evaluation

Homogenity Score (higher is better)

In [39]:
```
y_pred = kmeans.predict(X)
```

In [43]:
```
from sklearn.metrics.cluster import homogeneity_score
homogeneity_score(y_kmeans,y_pred)
```

Out[43]:
```
np.float64(0.9179705840639469)
```

In [47]:
```
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans

silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    y_pred = kmeans.fit_predict(X)
    silhouette_scores.append(silhouette_score(X, y_pred))

best_k = range(2, 11)[silhouette_scores.index(max(silhouette_scores))]
print(f"Optimal number of clusters: {best_k}")
```

```
Optimal number of clusters: 2
```

In [ ]: