```
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
CustomerID                200 non-null int64
Gender                    200 non-null object
Age                       200 non-null int64
Annual Income (k$)        200 non-null int64
Spending Score (1-100)    200 non-null int64
```

There are 200 entries, visualized for 200 customers.

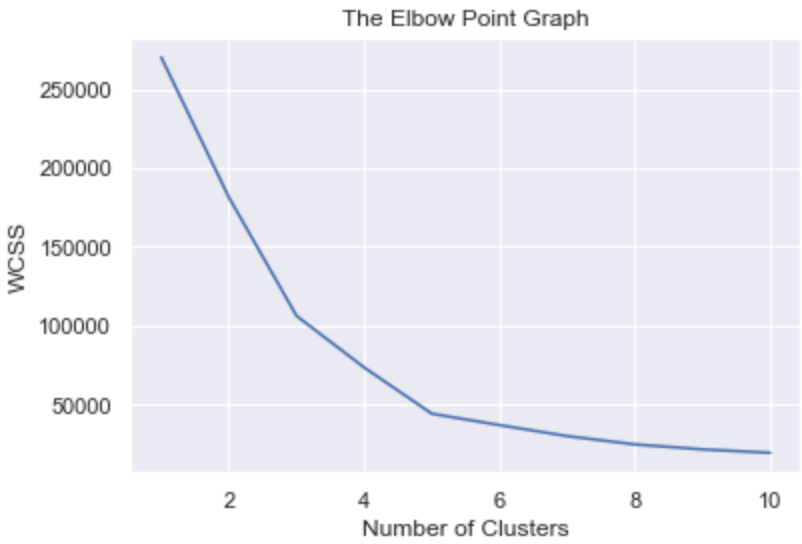
Spending score is a score computed for each of the mall's clients based on several criteria including income, the number of times per week they come to the mall and the money they spent in a year. This score is between 1–100. Our target in this model will be to divide the customers into a reasonable number of segments and determine the segments of the mall customers.

We actually do not know the number of clusters. The Elbow method is one of the robust ones used to find out the optimal number of clusters. In this method, the sum of distances of observations from their cluster centroids, called Within-Cluster-Sum-of-Squares (WCSS). This is computed as

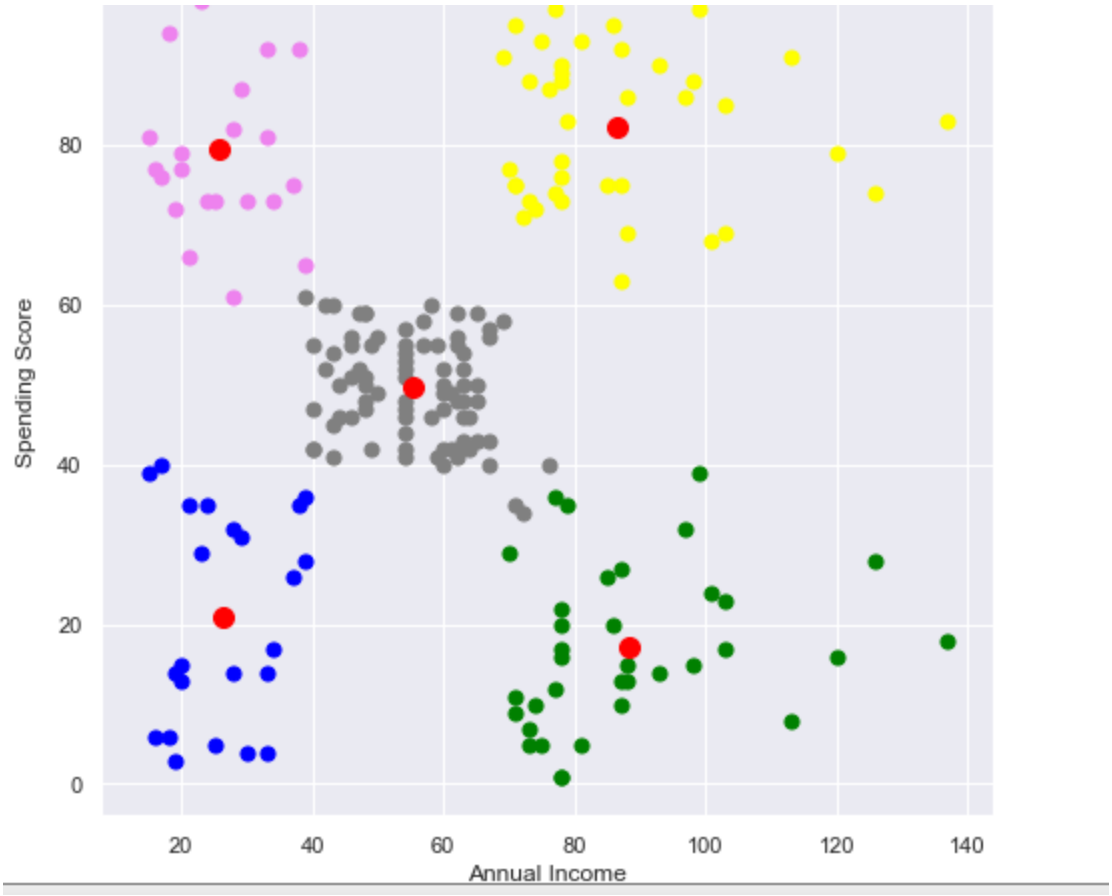
$$WCSS = \sum_{i \in n} (X_i - Y_i)^2$$

where Yi is centroid for observation Xi. By definition, this is geared towards maximizing number of clusters, and in limiting case each data point becomes its own cluster centroid

```
#KMeans class from the sklearn library.
from sklearn.cluster import KMeans
wcss=[]
#this loop will fit the k-means algorithm to our data and
#second we will compute the within cluster sum of squares and #appended to our wcss list.
for i in range(1,11):
     kmeans = KMeans(n_clusters=i, init ='k-means++', max_iter=300,  n_init=10,random_state=0 )
#i above is between 1-10 numbers. init parameter is the random
#initialization method
#we select kmeans++ method. max_iter parameter the maximum number of iterations there
#can be to find the final clusters when the K-meands algorithm is running.
#we enter the default value of 300
#the next parameter is n_init which is the number of times the
#K_means algorithm different initial centroid.
kmeans.fit(X)
#kmeans algorithm fits to the X dataset
wcss.append(kmeans.inertia_)
#kmeans inertia_ attribute is:  Sum of squared distances of samples
#to their closest cluster center.
#4.Plot the elbow graph
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



We are going to use the fit predict method that returns for each observation which cluster it belongs to. The cluster to which client belongs and it will return this cluster numbers into a single vector that is called y K-means



```
"""Optimum Number of Clusters = 5

Training the k-Means Clustering Model
"""

kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)
#'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
#Determines random number generation for centroid initialization. Use an int to make the randomness deterministic


# return a label for each data point based on their cluster
Y = kmeans.fit_predict(X)
```

```
"""5 Clusters -  0, 1, 2, 3, 4

Visualizing all the Clusters
"""

# plotting all the clusters and their Centroids

plt.figure(figsize=(8,8))
plt.scatter(X[Y==0,0], X[Y==0,1], s=50, c='green', label='Cluster 1')
plt.scatter(X[Y==1,0], X[Y==1,1], s=50, c='gray', label='Cluster 2')
plt.scatter(X[Y==2,0], X[Y==2,1], s=50, c='yellow', label='Cluster 3')
plt.scatter(X[Y==3,0], X[Y==3,1], s=50, c='violet', label='Cluster 4')
plt.scatter(X[Y==4,0], X[Y==4,1], s=50, c='blue', label='Cluster 5')


# plot the centroids
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100,
            c='red', label='Centroids')

plt.title('Customer Groups')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```