K-Means Clustering

What is K-Means?

K-Means is an unsupervised machine learning algorithm used to group data into **clusters** based on their similarity.

Key Concepts

- **Clusters:** Groups of similar data points.
- **Centroid:** The center point of each cluster.
- **WCSS (Within-Cluster-Sum-of-Squares):** Measures the compactness of clusters.

Steps in K-Means

1. Initialize the number of clusters (k).
2. Randomly assign cluster centroids.
3. Assign each data point to the nearest centroid.
4. Update centroids by calculating the mean of assigned points.
5. Repeat until cluster assignments no longer change.

Choosing the Optimal Number of Clusters

- Use the **Elbow Method**: Plot WCSS vs. number of clusters and choose the "elbow" point.

Applications of K-Means

- Customer segmentation
- Image compression

- Document clustering
- Pattern recognition

Importing the necessary libraries

In [1]:
```
import numpy as np  # For numerical computations
import matplotlib.pyplot as plt  # For data visualization
import pandas as pd  # For data manipulation
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Importing the dataset
# The dataset contains information about customers' annual income and spending score
dataset = pd.read_csv(r"D:\FSDS Material\Dataset\Clustering\Mall_Customers.csv")
X = dataset.iloc[:, [3, 4]].values  # Selecting columns 'Annual Income' and 'Spending Score'
```

In [5]:
```
dataset.shape
```

Out[5]:
```
(200, 5)
```

In [7]:
```
print(X)
```

```
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
 [ 19  14]
 [ 19  99]
 [ 20  15]
 [ 20  77]
 [ 20  13]
 [ 20  79]
 [ 21  35]
 [ 21  66]
 [ 23  29]
 [ 23  98]
 [ 24  35]
 [ 24  73]
 [ 25   5]
 [ 25  73]
 [ 28  14]
 [ 28  82]
 [ 28  32]
 [ 28  61]
 [ 29  31]
 [ 29  87]
 [ 30   4]
 [ 30  73]
 [ 33   4]
 [ 33  92]
 [ 33  14]
 [ 33  81]
 [ 34  17]
 [ 34  73]
 [ 37  26]
 [ 37  75]
 [ 38  35]
 [ 38  92]
```

```
 [ 39  36]
 [ 39  61]
 [ 39  28]
 [ 39  65]
 [ 40  55]
 [ 40  47]
 [ 40  42]
 [ 40  42]
 [ 42  52]
 [ 42  60]
 [ 43  54]
 [ 43  60]
 [ 43  45]
 [ 43  41]
 [ 44  50]
 [ 44  46]
 [ 46  51]
 [ 46  46]
 [ 46  56]
 [ 46  55]
 [ 47  52]
 [ 47  59]
 [ 48  51]
 [ 48  59]
 [ 48  50]
 [ 48  48]
 [ 48  59]
 [ 48  47]
 [ 49  55]
 [ 49  42]
 [ 50  49]
 [ 50  56]
 [ 54  47]
 [ 54  54]
 [ 54  53]
 [ 54  48]
 [ 54  52]
 [ 54  42]
 [ 54  51]
 [ 54  55]
 [ 54  41]
 [ 54  44]
```

```
 [ 54  57]
 [ 54  46]
 [ 57  58]
 [ 57  55]
 [ 58  60]
 [ 58  46]
 [ 59  55]
 [ 59  41]
 [ 60  49]
 [ 60  40]
 [ 60  42]
 [ 60  52]
 [ 60  47]
 [ 60  50]
 [ 61  42]
 [ 61  49]
 [ 62  41]
 [ 62  48]
 [ 62  59]
 [ 62  55]
 [ 62  56]
 [ 62  42]
 [ 63  50]
 [ 63  46]
 [ 63  43]
 [ 63  48]
 [ 63  52]
 [ 63  54]
 [ 64  42]
 [ 64  46]
 [ 65  48]
 [ 65  50]
 [ 65  43]
 [ 65  59]
 [ 67  43]
 [ 67  57]
 [ 67  56]
 [ 67  40]
 [ 69  58]
 [ 69  91]
 [ 70  29]
 [ 70  77]
```

```
 [ 71  35]
 [ 71  95]
 [ 71  11]
 [ 71  75]
 [ 71   9]
 [ 71  75]
 [ 72  34]
 [ 72  71]
 [ 73   5]
 [ 73  88]
 [ 73   7]
 [ 73  73]
 [ 74  10]
 [ 74  72]
 [ 75   5]
 [ 75  93]
 [ 76  40]
 [ 76  87]
 [ 77  12]
 [ 77  97]
 [ 77  36]
 [ 77  74]
 [ 78  22]
 [ 78  90]
 [ 78  17]
 [ 78  88]
 [ 78  20]
 [ 78  76]
 [ 78  16]
 [ 78  89]
 [ 78   1]
 [ 78  78]
 [ 78   1]
 [ 78  73]
 [ 79  35]
 [ 79  83]
 [ 81   5]
 [ 81  93]
 [ 85  26]
 [ 85  75]
 [ 86  20]
 [ 86  95]
```

```
 [ 87  27]
 [ 87  63]
 [ 87  13]
 [ 87  75]
 [ 87  10]
 [ 87  92]
 [ 88  13]
 [ 88  86]
 [ 88  15]
 [ 88  69]
 [ 93  14]
 [ 93  90]
 [ 97  32]
 [ 97  86]
 [ 98  15]
 [ 98  88]
 [ 99  39]
 [ 99  97]
 [101  24]
 [101  68]
 [103  17]
 [103  85]
 [103  23]
 [103  69]
 [113   8]
 [113  91]
 [120  16]
 [120  79]
 [126  28]
 [126  74]
 [137  18]
 [137  83]]
```

In [9]:
```
# Using the Elbow Method to find the optimal number of clusters
from sklearn.cluster import KMeans  # Importing the K-Means clustering model
```

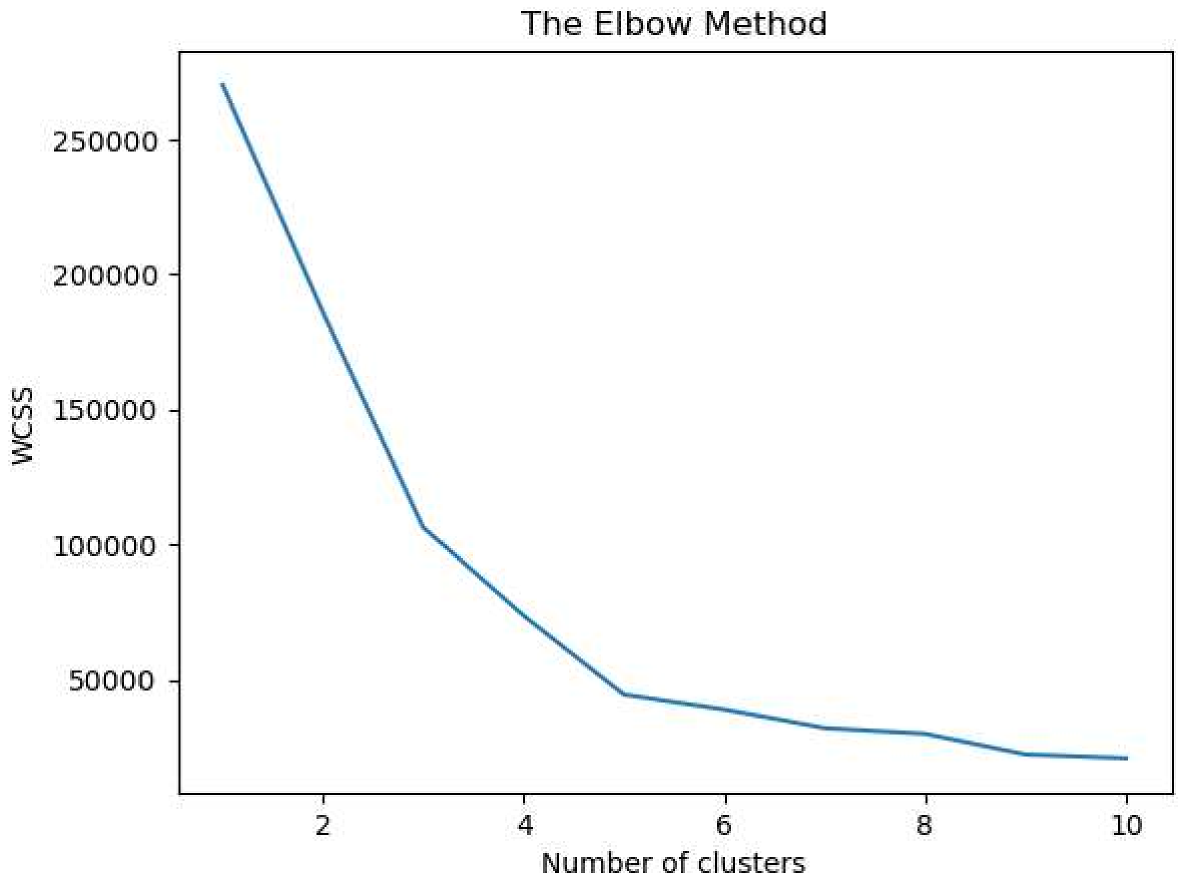
In [10]:
```
# List to store the Within-Cluster-Sum-of-Squares (WCSS) for different cluster numbers
wcss = []
```

In [13]:
```
# Check WCSS
print("WCSS values:", wcss)
```

```
WCSS values: []
```

In [15]:
```python
# Iterating over a range of cluster counts (1 to 10) to compute WCSS
for i in range(1, 11):
    # Initializing K-Means with `i` clusters
    # `init='k-means++'` ensures efficient cluster centroid initialization
    # `random_state=0` ensures reproducibility
    kmeans = KMeans(n_clusters=i, init="k-means++", random_state=0)
    kmeans.fit(X)  # Fitting K-Means to the dataset
    wcss.append(kmeans.inertia_)  # Inertia is the WCSS value for the current model
```

In [17]:
```python
# Plotting the Elbow Curve
plt.plot(range(1, 11), wcss)  # Number of clusters vs. WCSS
plt.title('The Elbow Method')  # Title for the plot
plt.xlabel('Number of clusters')  # X-axis label
plt.ylabel('WCSS')  # Y-axis label
plt.show()  # Display the plot
```



Explanation:

The "Elbow Method" involves plotting WCSS against the number of clusters.

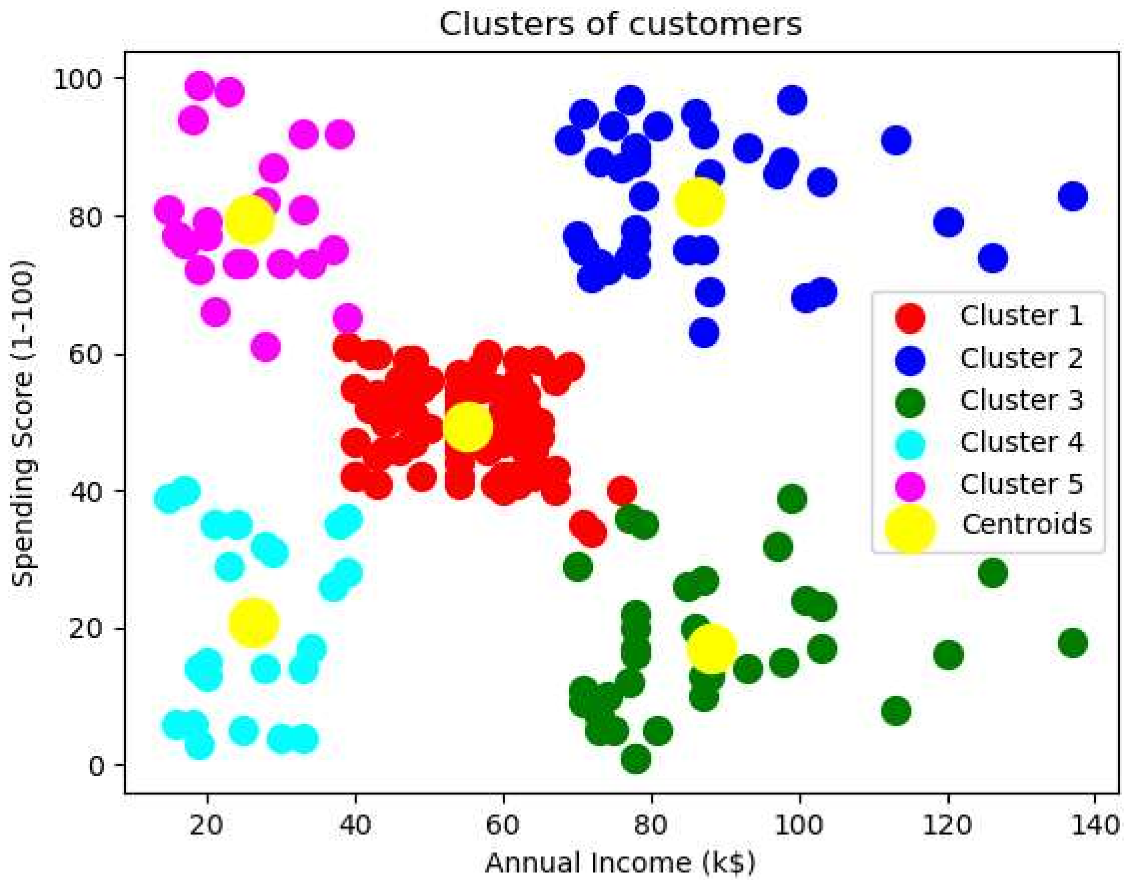
The optimal number of clusters is at the "elbow" point, where WCSS starts decreasing at a slower rate.

In [20]:
```
# Training the K-Means model on the dataset with the optimal number of clusters (e.g., 5)
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)  # 5 clusters
y_kmeans = kmeans.fit_predict(X)  # Predicting the cluster for each data point
```

In [22]:
```
# Visualizing the clusters
# Scatter plots for each cluster, identified by `y_kmeans`
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='red', label='Cluster 1')  # Cluster 1
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='blue', label='Cluster 2')  # Cluster 2
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s=100, c='green', label='Cluster 3')  # Cluster 3
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s=100, c='cyan', label='Cluster 4')  # Cluster 4
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s=100, c='magenta', label='Cluster 5')  # Cluster 5

# Highlighting the cluster centroids
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=300, c='yellow', label='Centroids')  # Centroids of the clusters

# Adding titles and labels
plt.title('Clusters of customers')  # Plot title
plt.xlabel('Annual Income (k$)')  # X-axis label
plt.ylabel('Spending Score (1-100)')  # Y-axis label
plt.legend()  # Legend to identify clusters and centroids
plt.show()  # Display the visualization
```



Explanation:

- Each cluster is represented by a different color.
- The centroids are marked in yellow and represent the central point of each cluster.
- Clustering helps to segment customers based on their spending behavior and income.

```
In [ ]:
```