K-Means Clustering Algorithm

Importing Libraries

In [7]:
```python
# Import necessary libraries
import pandas as pd   # For data manipulation and analysis
import numpy as np    # For numerical computations
import matplotlib.pyplot as plt # For plotting graphs

from sklearn.cluster import KMeans # For KMeans clustering algorithm
```

Load the Dataset

In [3]:
```python
# Read the CSV file 'Mall_Customers.csv' into a pandas DataFrame
data = pd.read_csv('Mall_Customers.csv')

# Display the first few rows of the dataset
data.head()
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Make another dataframe of column 3 & 4

In [4]:
```python
# Extract columns 3 and 4 from the DataFrame 'data'
# iloc[:, [3,4]] selects all rows and columns 3 and 4 (0-indexed)
df = data.iloc[:, [3, 4]]

# Display the first few rows of the new DataFrame 'df'
df.head()
```

Out[4]:

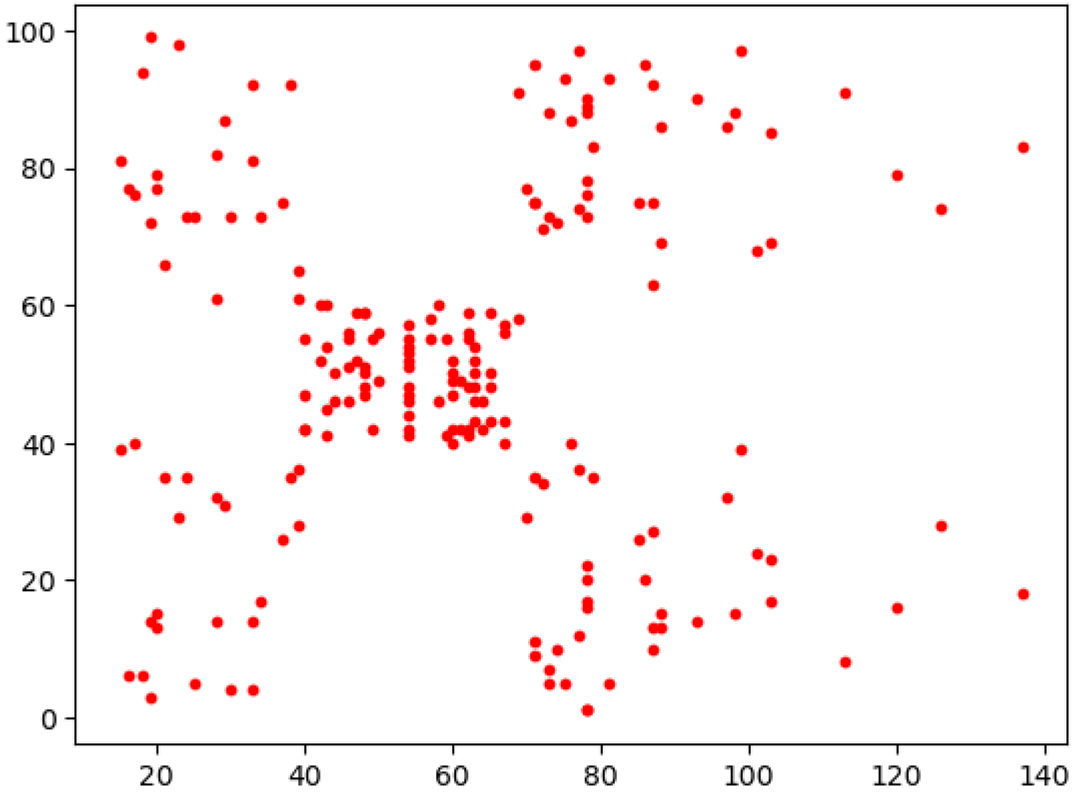
	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40

In [5]:
```python
# Check shape of dataframe
df.shape
```

Out[5]: (200, 2)

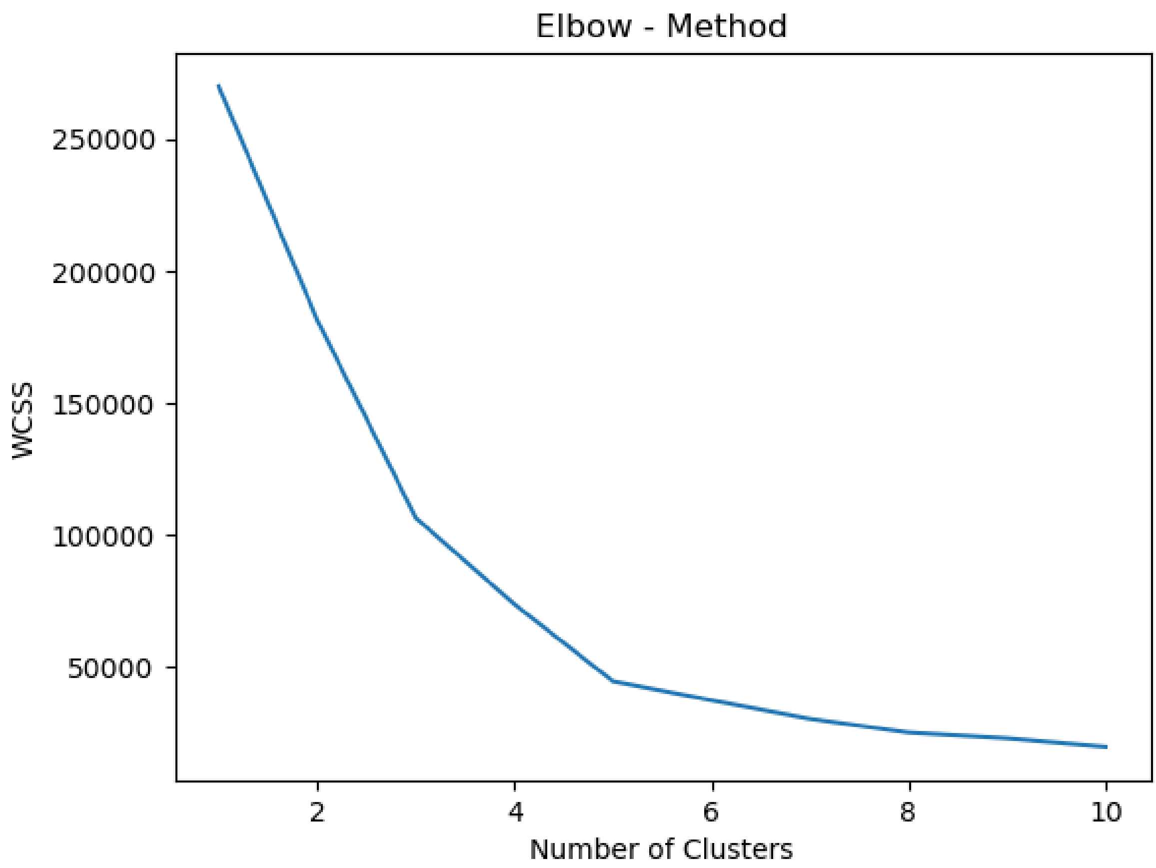
In [6]:
```python
# Create a scatter plot for df
plt.scatter(df.iloc[:, 0], df.iloc[:, 1], c='red', s=10)
```

Out[6]: <matplotlib.collections.PathCollection at 0x1104d7d50>



KMeans Clustering using Elbow Method

In [8]:
```python
# Initialize an empty list to store the WCSS values
wcss = []

# Iterate over a range of cluster numbers from 1 to 10
for i in range(1, 11):
    # Create a KMeans object with 'i' clusters
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=

    # Fit the KMeans model to the data
    kmeans.fit(df)

    # Append the WCSS value to the list
    wcss.append(kmeans.inertia_)

# Plot the number of clusters against the corresponding WCSS values
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



KMeans Clustering with Specified Clusters

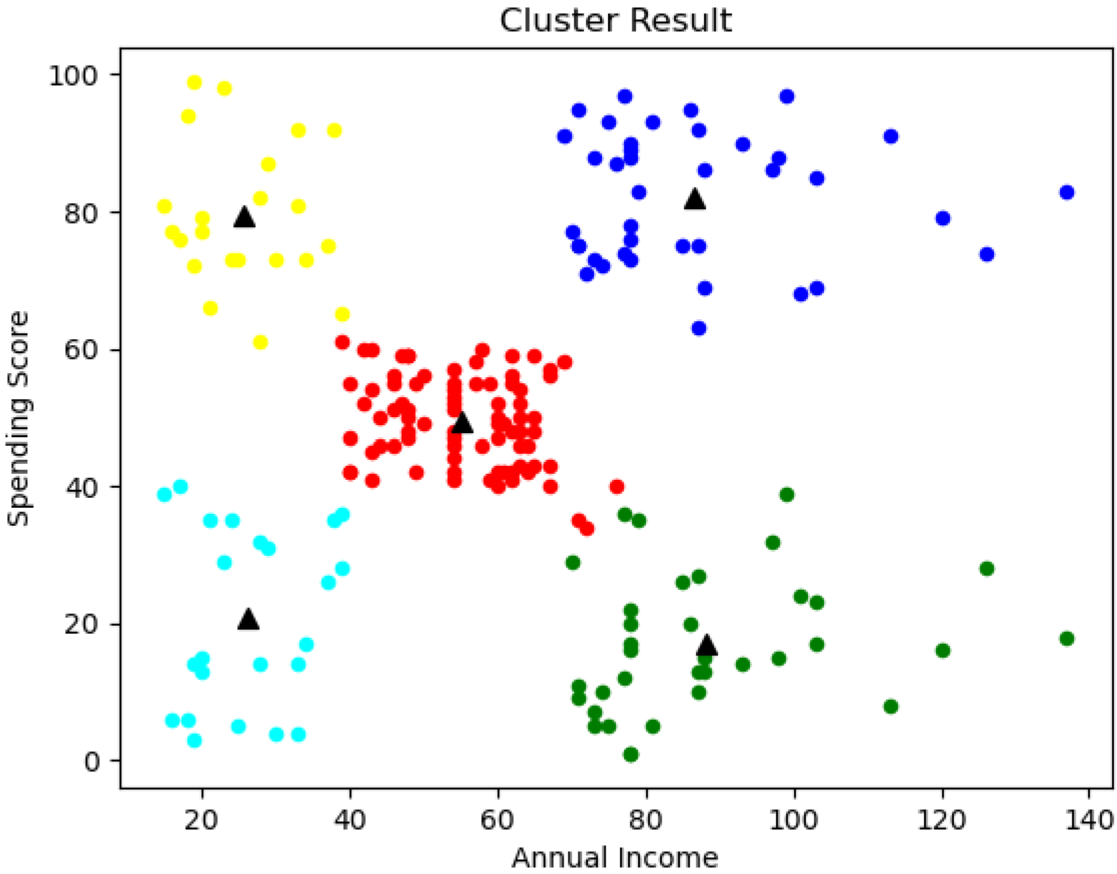
In [9]:
```
# Create a KMeans object with 5 clusters using k-means++ initialization
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10)

# Fit the KMeans model to the data and predict cluster labels for each da
labels = kmeans.fit_predict(df)
```

In [10]:
```
# Find the unique cluster labels
unique_labels = np.unique(labels)
```

Out[10]:
```
array([0, 1, 2, 3, 4], dtype=int32)
```

In [11]:
```
# Scatter plot for points in cluster 0
plt.scatter(df.iloc[labels==0, 0], df.iloc[labels==0, 1], s=20, c='red')

# Scatter plot for points in cluster 1
plt.scatter(df.iloc[labels==1, 0], df.iloc[labels==1, 1], s=20, c='green'

# Scatter plot for points in cluster 2
plt.scatter(df.iloc[labels==2, 0], df.iloc[labels==2, 1], s=20, c='blue')

# Scatter plot for points in cluster 3
plt.scatter(df.iloc[labels==3, 0], df.iloc[labels==3, 1], s=20, c='yellow

# Scatter plot for points in cluster 4
plt.scatter(df.iloc[labels==4, 0], df.iloc[labels==4, 1], s=20, c='cyan')

# Plot cluster centers
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
```

```
# Set plot title and labels
plt.title('Cluster Result')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')

# Show the plot
plt.show()
```



DBSCAN Clustering Algorithm

Importing Libraries

In [17]:
```
import numpy as np                # For numerical computations
import pandas as pd               # For data manipulation and analysis
import matplotlib.pyplot as plt   # For plotting graphs

from sklearn.cluster import KMeans    # For KMeans clustering
from sklearn.cluster import DBSCAN    # For DBSCAN clustering
```

Load the Dataset

In [18]:
```
# Read the CSV file 'Mall_Customers.csv' into a pandas DataFrame
data = pd.read_csv('Mall_Customers.csv')

# Display the first few rows of the dataset
data.head()
```

Out[18]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [19]:
```
# Check the shape of the DataFrame 'data'
data.shape
```

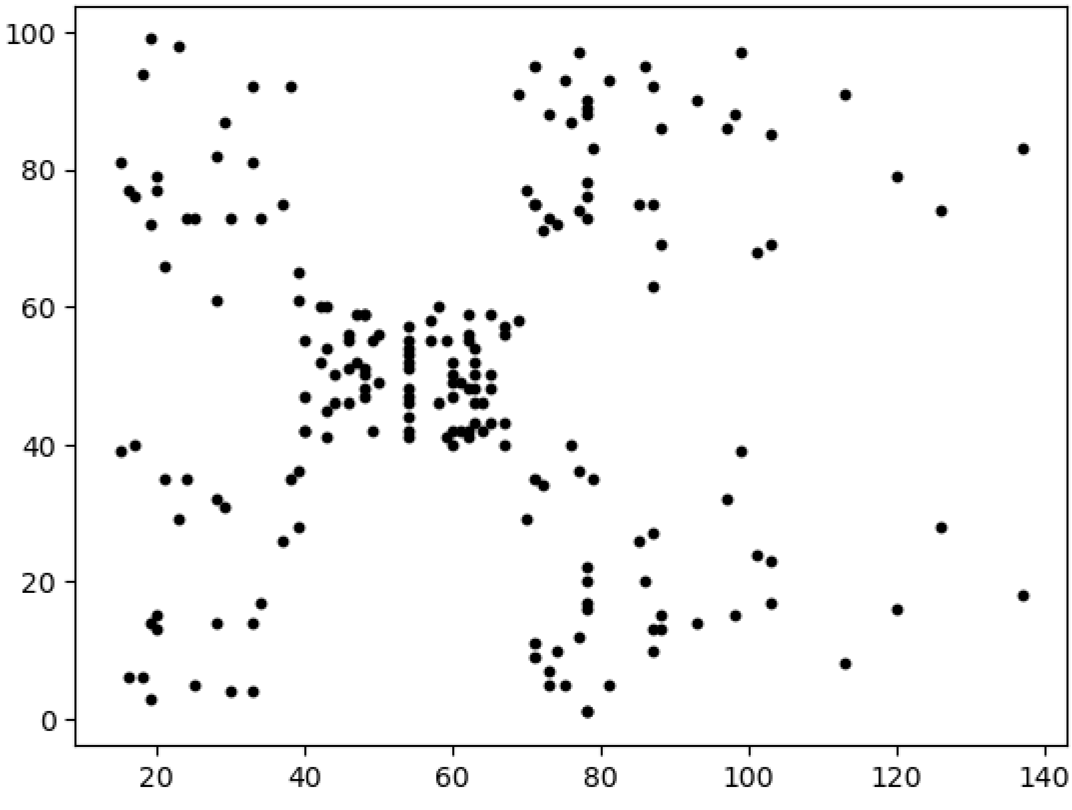
Out[19]:
```
(200, 5)
```

In [20]:
```
# Extract columns 3 and 4 from the DataFrame 'data' and convert them into
df = data.iloc[:, [3, 4]].values

# Display the resulting NumPy array
df
```

```
Out[20]:  array([[ 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],
```

```
       [ 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]])
```

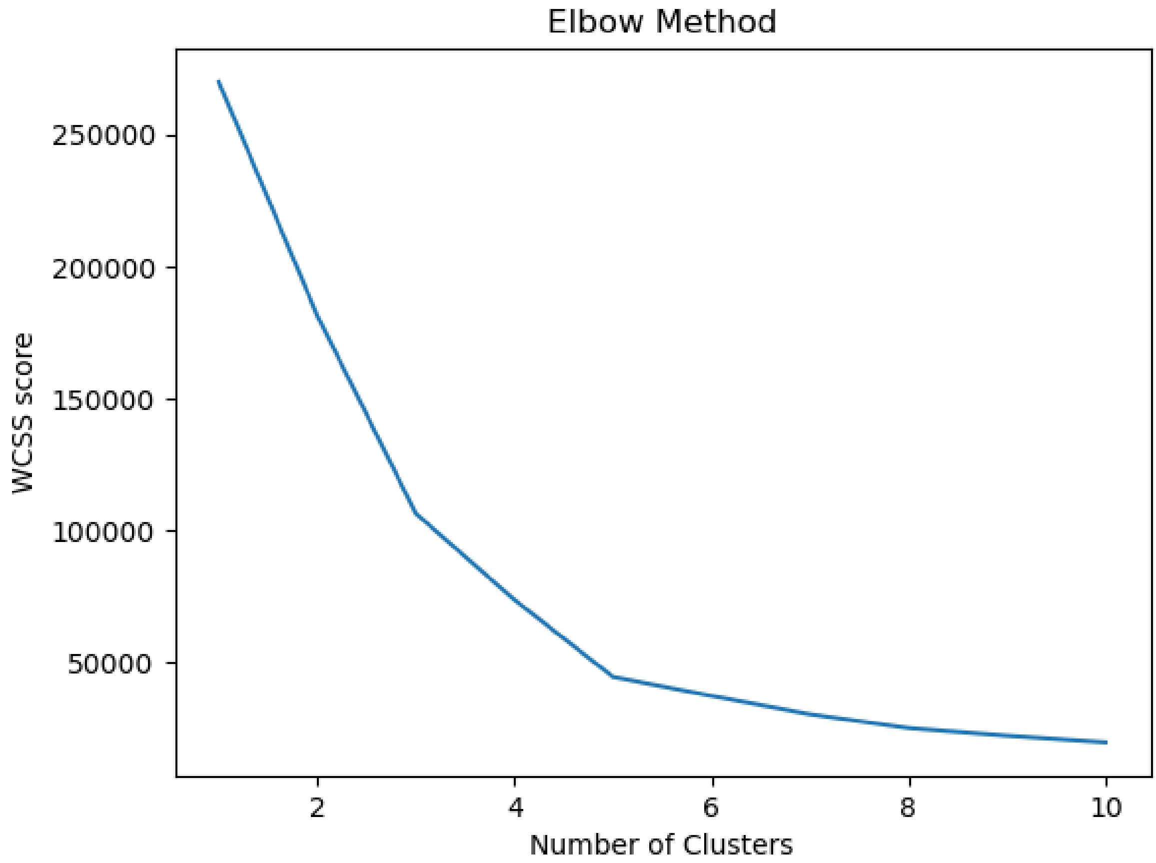
In [21]:
```
# Create a scatter plot of the data points
plt.scatter(df[:, 0], df[:, 1], s=10, c='black')

# Display the plot
plt.show()
```



Using Elbow Method for finding Optimal Clusters

In [22]:
```
# Initialize an empty list to store the within-cluster sum of squares (WC
wcss = []

# Iterate over a range of cluster numbers from 1 to 10
```

```python
for i in range(1, 11):
    # Initialize KMeans clustering with the current number of clusters
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=

    # Fit KMeans to the data and compute the WCSS score
    kmeans.fit(df)

    # Append the WCSS score to the list
    wcss.append(kmeans.inertia_)

# Plot the number of clusters against the WCSS scores
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS score')
plt.show()
```



DBSCAN Clustering

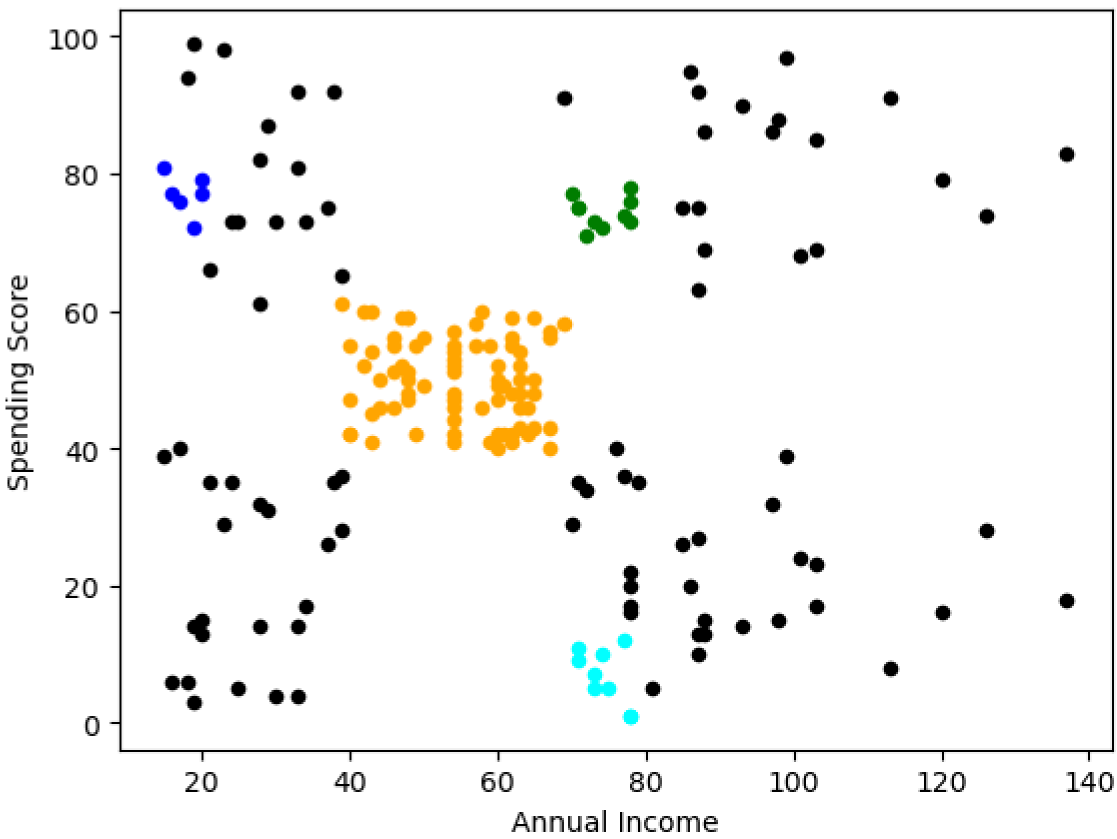
In [23]:
```python
# Initialize DBSCAN clustering algorithm with specified parameters
dbscan = DBSCAN(eps=5, min_samples=5)
```

In [24]:
```python
# Fit the DataFrame to DBSCAN Model
labels = dbscan.fit_predict(df)
```

In [26]:
```python
# Calculate the unique cluster labels
np.unique(labels)
```

Out[26]:
```
array([-1,  0,  1,  2,  3,  4])
```

In [27]:
```
# Scatter plot for points classified as noise (label = -1)
plt.scatter(df[labels == -1, 0], df[labels == -1, 1], s=20, c='black')

# Scatter plot for points in cluster 0
plt.scatter(df[labels == 0, 0], df[labels == 0, 1], s=20, c='blue')

# Scatter plot for points in cluster 1
plt.scatter(df[labels == 1, 0], df[labels == 1, 1], s=20, c='orange')

# Scatter plot for points in cluster 2
plt.scatter(df[labels == 2, 0], df[labels == 2, 1], s=20, c='green')

# Scatter plot for points in cluster 3
plt.scatter(df[labels == 3, 0], df[labels == 3, 1], s=20, c='cyan')

# Set the labels for x and y axes
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')

# Display the plot
plt.show()
```



- DBSCAN clusters the data points which are dense or close to each other.

Fuzzy C-Means Clustering Algorithm

Import Libraries

In [1]:
```
import pandas as pd   # For data manipulation and analysis
import numpy as np    # For numerical computations
import seaborn as sns # For statistical data visualization
import matplotlib.pyplot as plt # For plotting graphs
```

Load Dataset

In [2]:
```
# Read the CSV file 'housing.csv' into a pandas DataFrame
data = pd.read_csv('housing.csv')

# Display the first few rows of the dataset
data.head()
```

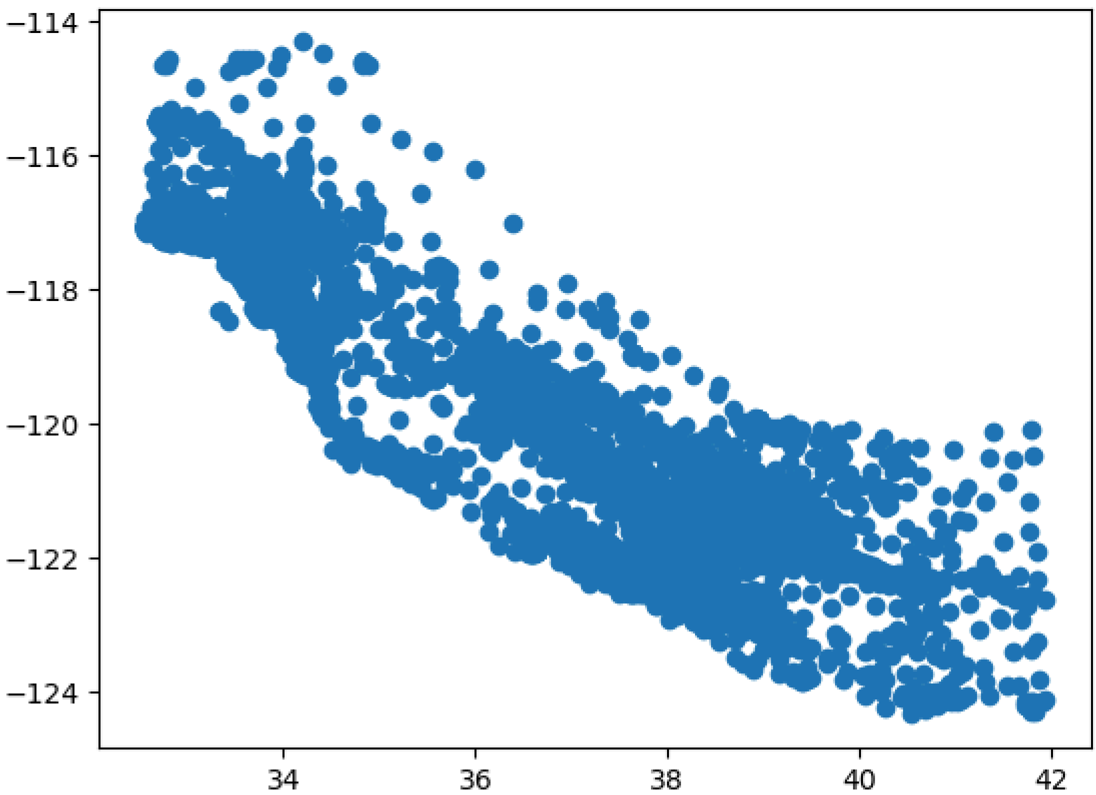
Out[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

Fuzzy C - Means Clustering

In [3]:
```
# Selecting specific columns ('latitude' and 'longitude') from the DataFr
data = data.loc[:, ['latitude', 'longitude']]

# Creating a scatter plot using latitude and longitude data
plt.scatter(data['latitude'], data['longitude'])
```

Out[3]: <matplotlib.collections.PathCollection at 0x16558eb50>



In [4]:
```
# Convert the pandas DataFrame 'data' into a NumPy array
data = np.array(data)
```

In [6]:
```
# Initialize the number of clusters (k) and the fuzziness coefficient (m)
k = 5
m = 3

# Initialize the membership matrix U with random values
U = np.random.rand(data.shape[0], k)

# Normalize the membership matrix U to ensure that each row sums to 1
U /= np.sum(U, axis=1)[:, np.newaxis]

# Display the membership matrix U
U
```

Out[6]:
```
array([[0.35642361, 0.02536057, 0.36635101, 0.02995748, 0.22190733],
       [0.06423838, 0.22233999, 0.24916174, 0.29006653, 0.17419337],
       [0.00680218, 0.59917191, 0.13855828, 0.05956693, 0.1959007 ],
       ...,
       [0.23331775, 0.21059696, 0.09073867, 0.07046755, 0.39487907],
       [0.2442473 , 0.19056832, 0.0428038 , 0.42603828, 0.0963423 ],
       [0.29794273, 0.28513548, 0.29614108, 0.09846029, 0.02232043]])
```

In [8]:
```
def cal_centroids(data, k, U, m):
    # Initialize an array to store the centroids of the clusters
    centroids = np.zeros((k, data.shape[1]))

    # Iterate over each cluster
    for i in range(k):
        # Calculate the centroid of the current cluster
        # The centroid is calculated by taking the weighted sum of data p
        # Weighted by the degree of membership to the cluster raised to t
        numerator = np.sum((U[:, i]**m)[:, np.newaxis] * data, axis=0)
```

```
        denominator = np.sum(U[:, i]**m)
        centroids[i, :] = numerator / denominator

    # Return the centroids
    return centroids
```

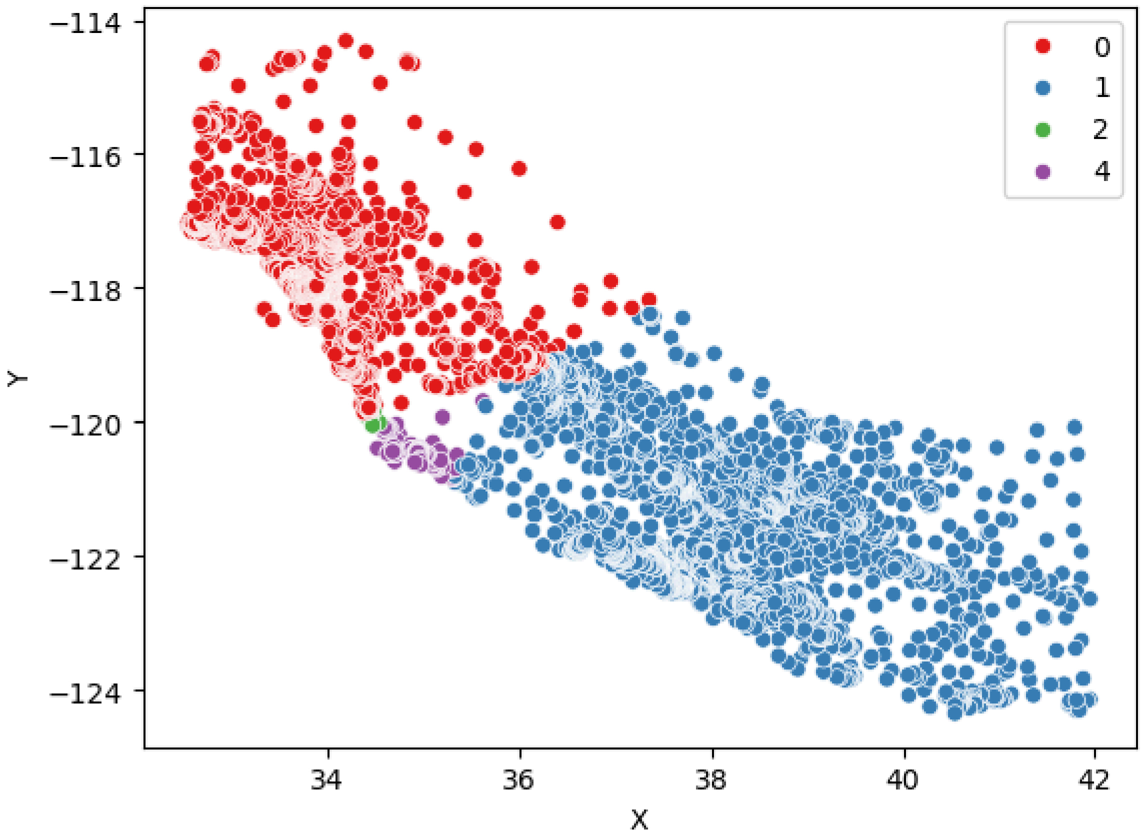
In [23]:
```
def cal_membership(data, centroids, k, m):
    # Initialize an array to store the updated membership matrix
    U_new = np.zeros((data.shape[0], k))

    # Iterate over each cluster
    for i in range(k):
        # Calculate the Euclidean distance between each data point and th
        distance = np.linalg.norm(data - centroids[i, :], axis=1)

        # Assign the inverse of the distance raised to the power of 2/(m-
        U_new[:, i] = 1 / (distance ** (2 / (m - 1)))

    # Normalize the membership matrix
    U_new /= np.sum(U_new, axis=1)[:, np.newaxis]

    # Return the updated membership matrix
    return U_new
```

In [16]:
```
# Assign cluster labels by finding the index of the maximum membership va
labels = np.argmax(U_new, axis=1)

# Display the cluster labels
labels
```

Out[16]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

In [22]:
```
# Create a DataFrame 'df' with the input data and column names 'X' and 'Y
df = pd.DataFrame(data, columns=['X', 'Y'])

# Create a scatter plot using seaborn's scatterplot function
sns.scatterplot(data=df, x='X', y='Y', hue=labels, palette='Set1')
```

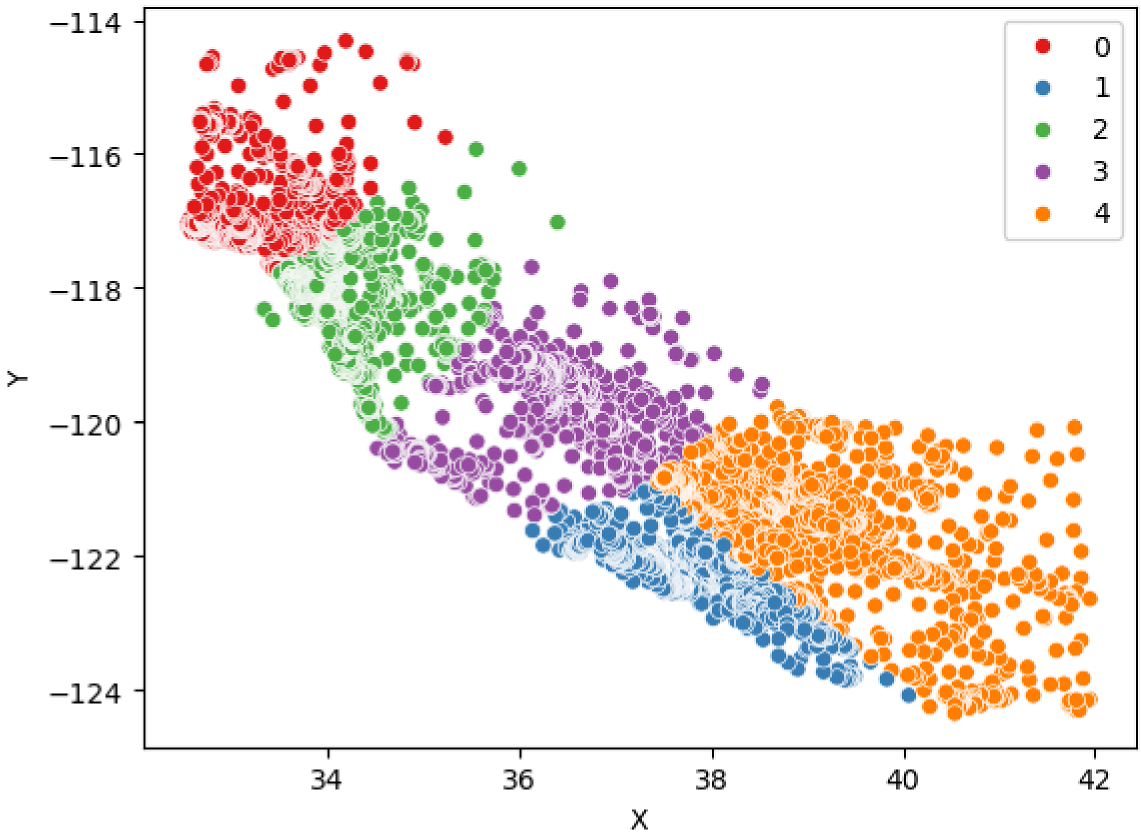
Out[22]:
```
<Axes: xlabel='X', ylabel='Y'>
```



In [24]:
```
# Set the maximum number of iterations
max_itr = 100

# Iterate over the specified maximum number of iterations
for itr in range(max_itr):
    # Update centroids based on current membership matrix U
    centroids = cal_centroids(data, 5, U, 3)

    # Update membership matrix based on current centroids
    U_new = cal_membership(data, centroids, 5, 3)

    # Check convergence criteria
    if np.linalg.norm(U_new - U) <= 0.00001:
        # If convergence is achieved, exit the loop
        break

    # Update membership matrix for the next iteration
    U = U_new

    # Assign cluster labels based on updated membership matrix
    labels = np.argmax(U_new, axis=1)
```

In [25]:
```
# Create a DataFrame 'df' with the input data and column names 'X' and 'Y
df = pd.DataFrame(data, columns=['X', 'Y'])

# Create a scatter plot using seaborn's scatterplot function
sns.scatterplot(data=df, x='X', y='Y', hue=labels, palette='Set1')
```

Out[25]:
```
<Axes: xlabel='X', ylabel='Y'>
```



−114
−116
−118
−120
−122
−124
Y
34
36
38
40
42
X
0
1
2
3
4

Hierarchical Clustering Algorithm

Importing Libraries

In [21]:
```
import pandas as pd              # For data manipulation
import numpy as np               # For numerical computations
import matplotlib.pyplot as plt  # For plotting graphs
%matplotlib inline

import scipy.cluster.hierarchy as sch     # For hierarchical clustering
from sklearn.cluster import AgglomerativeClustering  # For agglomerative
from sklearn.metrics import silhouette_score         # For silhouette sco
from sklearn.preprocessing import normalize         # For data normalizat
from sklearn.cluster import KMeans                   # For KMeans cluster

import warnings
warnings.filterwarnings('ignore')  # Ignore warnings
```

Load the Dataset

In [3]:
```
# Read the CSV file 'Mall_Customers.csv' into a pandas DataFrame
data = pd.read_csv('Mall_Customers.csv')

# Display the first few rows of the dataset
data.head()
```

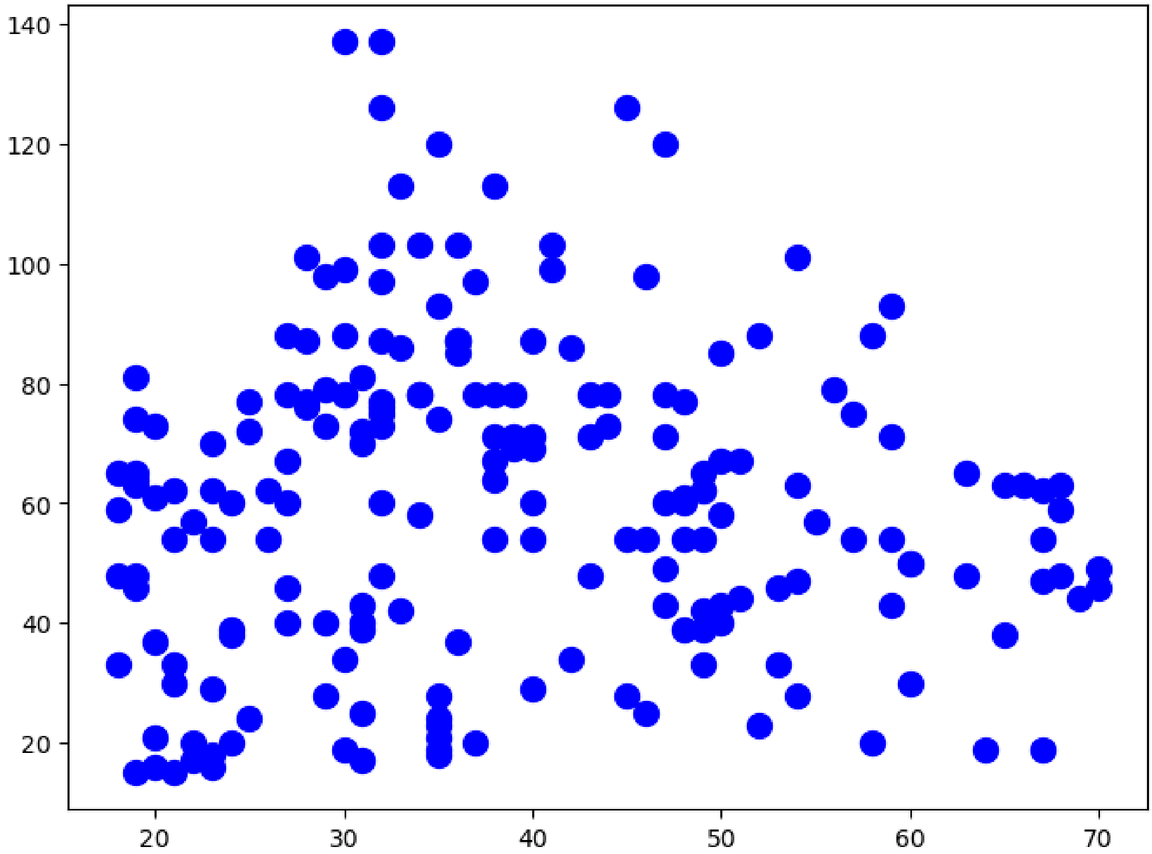
Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [4]:
```
# Select columns 'Age' and 'Annual Income (k$)' from the DataFrame 'data'
df = data.loc[:, ['Age', 'Annual Income (k$)']]
```

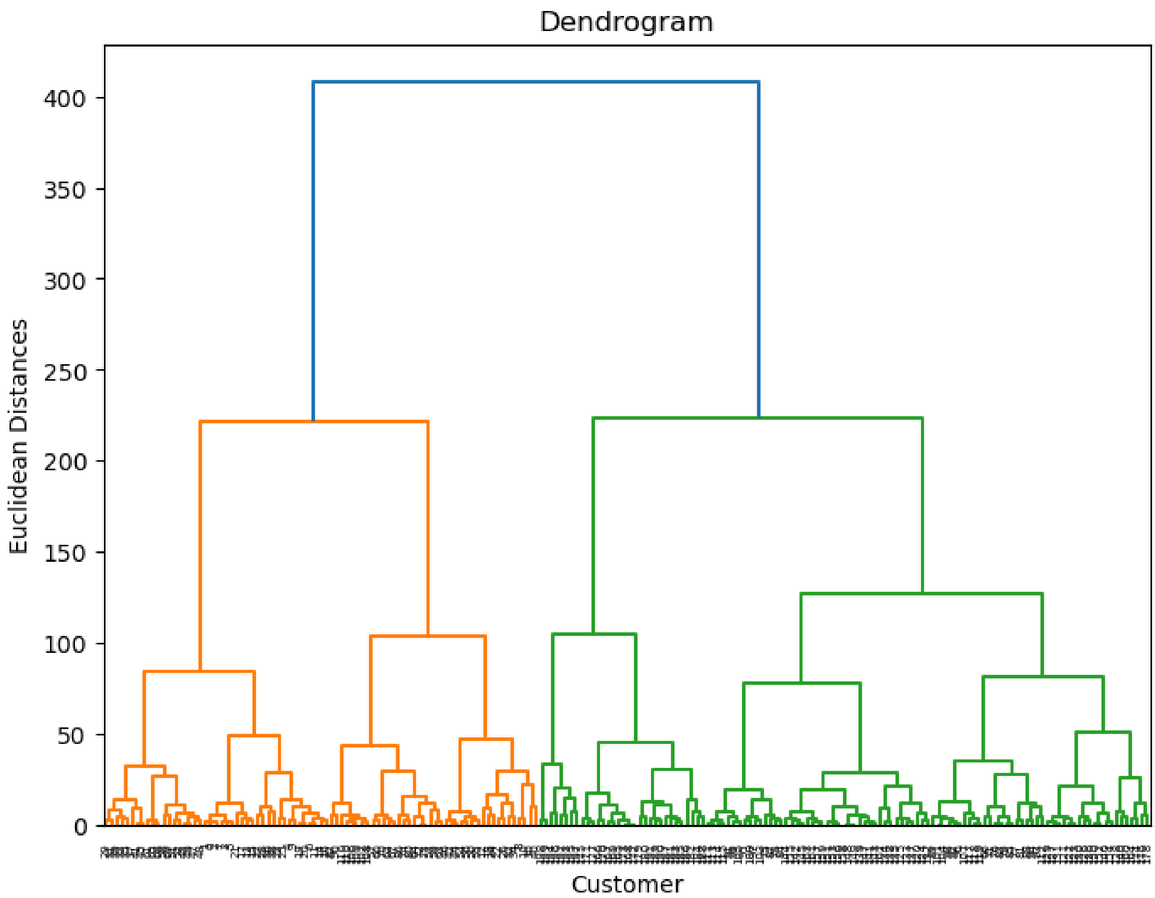
In [6]:
```
# Create a new figure with a specified size
plt.figure(figsize=(8, 6))

# Scatter plot of 'Age' against 'Annual Income (k$)'
plt.scatter(df[['Age']], df[['Annual Income (k$)']], s=100, c='blue')
```

Out[6]:
```
<matplotlib.collections.PathCollection at 0x149881790>
```



Hierarchical Clustering

In [8]:
```
# Create a new figure with a specified size
plt.figure(figsize=(8, 6))

# Generate the dendrogram using hierarchical clustering with the 'ward' m
dendrogram = sch.dendrogram(sch.linkage(df, method='ward'))

# Set the title and labels for the plot
plt.title('Dendrogram')
plt.xlabel('Customer')
plt.ylabel('Euclidean Distances')

# Display the plot
plt.show()
```



- Check for largest distance vertically without crossing any horizontal line.

In [14]:
```
# Create an AgglomerativeClustering object with 2 clusters
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', lin

# Fit the clustering model to the data and predict cluster labels for eac
cl = cluster.fit_predict(df)

cl
```

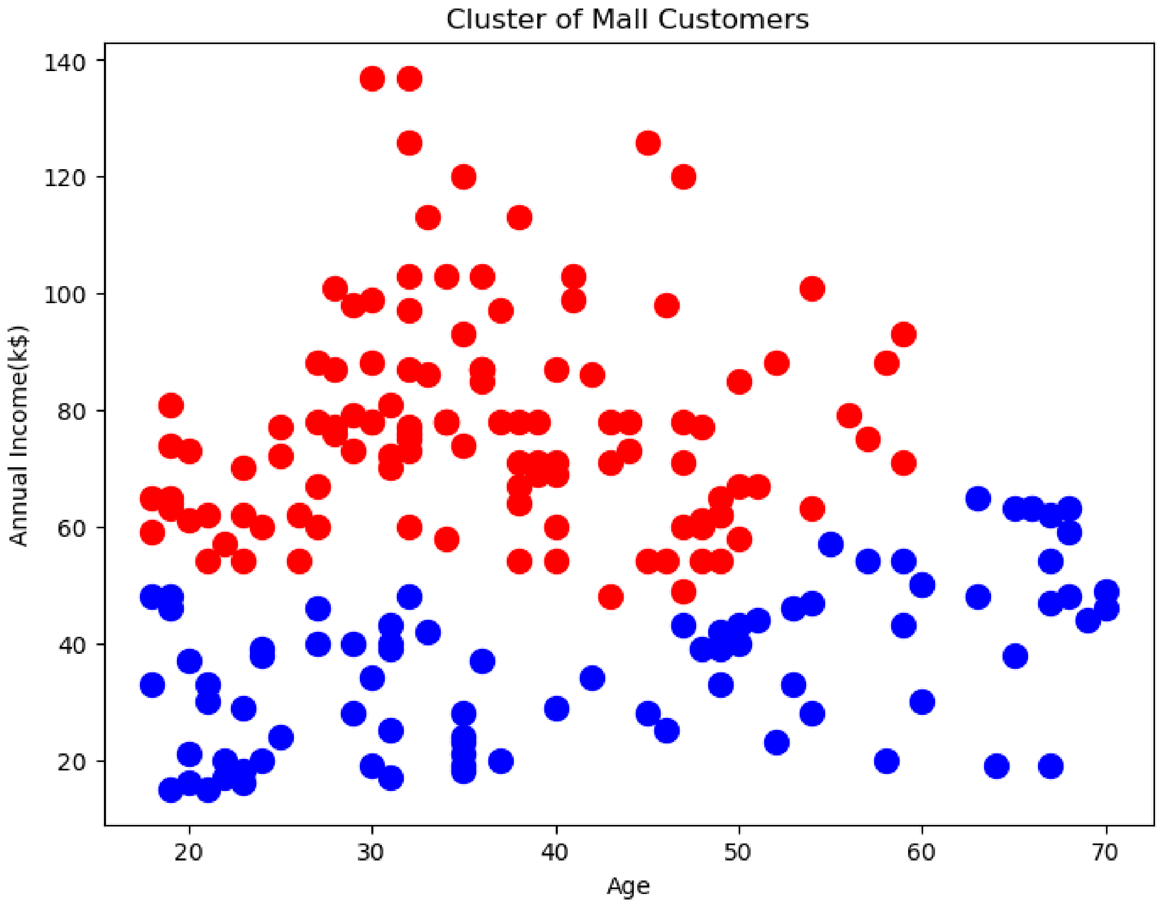
Out[14]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

In [16]:
```
# Calculate the silhouette score
silhouette = silhouette_score(df, cl)
```

Out[16]: 0.4104652474372429

In [17]:
```
# Convert the DataFrame 'df' to a Numpy array
X = df.values
```

In [18]:
```
# Create a new figure with a specified size
plt.figure(figsize=(8, 6))

# Scatter plot for points in cluster 0
plt.scatter(X[cl==0, 0], X[cl==0, 1], s=100, c='red', label='Cluster 1')

# Scatter plot for points in cluster 1
plt.scatter(X[cl==1, 0], X[cl==1, 1], s=100, c='blue', label='Cluster 2')

# Set plot title and labels
plt.title('Cluster of Mall Customers')
plt.xlabel('Age')
plt.ylabel('Annual Income(k$)')

# Show legend
plt.legend()

# Display the plot
plt.show()
```



- Silhouette Score is bad in previous Clustering so, we need to normalize the Age and Annual Income data.

Scale the Data

In [20]:
```
# Normalize the data in the Numpy array X
scaled = normalize(X)
```

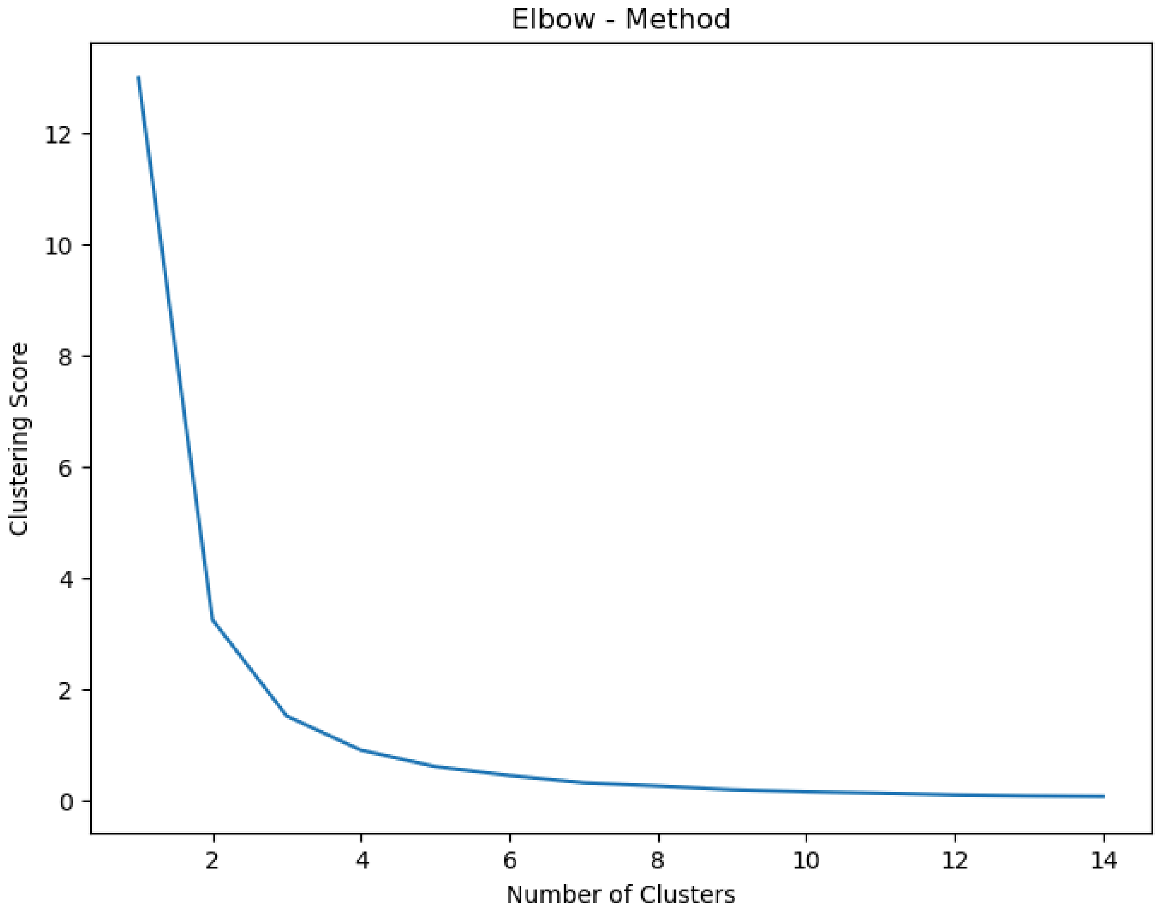
```python
# Convert the normalized data back to a pandas DataFrame
scaled = pd.DataFrame(scaled)

# Display the first few rows of the DataFrame scaled
scaled.head()
```

Out[20]:

	0	1
0	0.784883	0.619644
1	0.813733	0.581238
2	0.780869	0.624695
3	0.820905	0.571064
4	0.876812	0.480833

In [24]:

```python
# Initialize an empty list to store the WCSS values
wcss = []

# Iterate over a range of cluster numbers from 1 to 14
for i in range(1, 15):
    # Create a KMeans object with 'i' clusters using random initializatio
    kmeans = KMeans(n_clusters=i, init='random', random_state=42)

    # Fit the KMeans model to the normalized data
    kmeans.fit(scaled)

    # Append the WCSS value to the list
    wcss.append(kmeans.inertia_)

# Create a new figure with a specified size
plt.figure(figsize=(8, 6))

# Plot the number of clusters against the corresponding WCSS values
plt.plot(range(1, 15), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('Clustering Score')
plt.show()
```



In [25]:

```
# Create a KMeans object with 2 clusters and a fixed random state for rep
kmeans = KMeans(n_clusters=2, random_state=42)

# Fit the KMeans model to the normalized data
kmeans.fit(scaled)

# Predict cluster labels for each data point
pred = kmeans.predict(scaled)

# Display the predicted cluster labels
pred
```

Out[25]:

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

In [28]:

```
# Calculate the new silhouette score
silhouette = silhouette_score(scaled, pred)
```

Out[28]:

```
0.6420367225684405
```

Mean Shift Clustering Algorithm

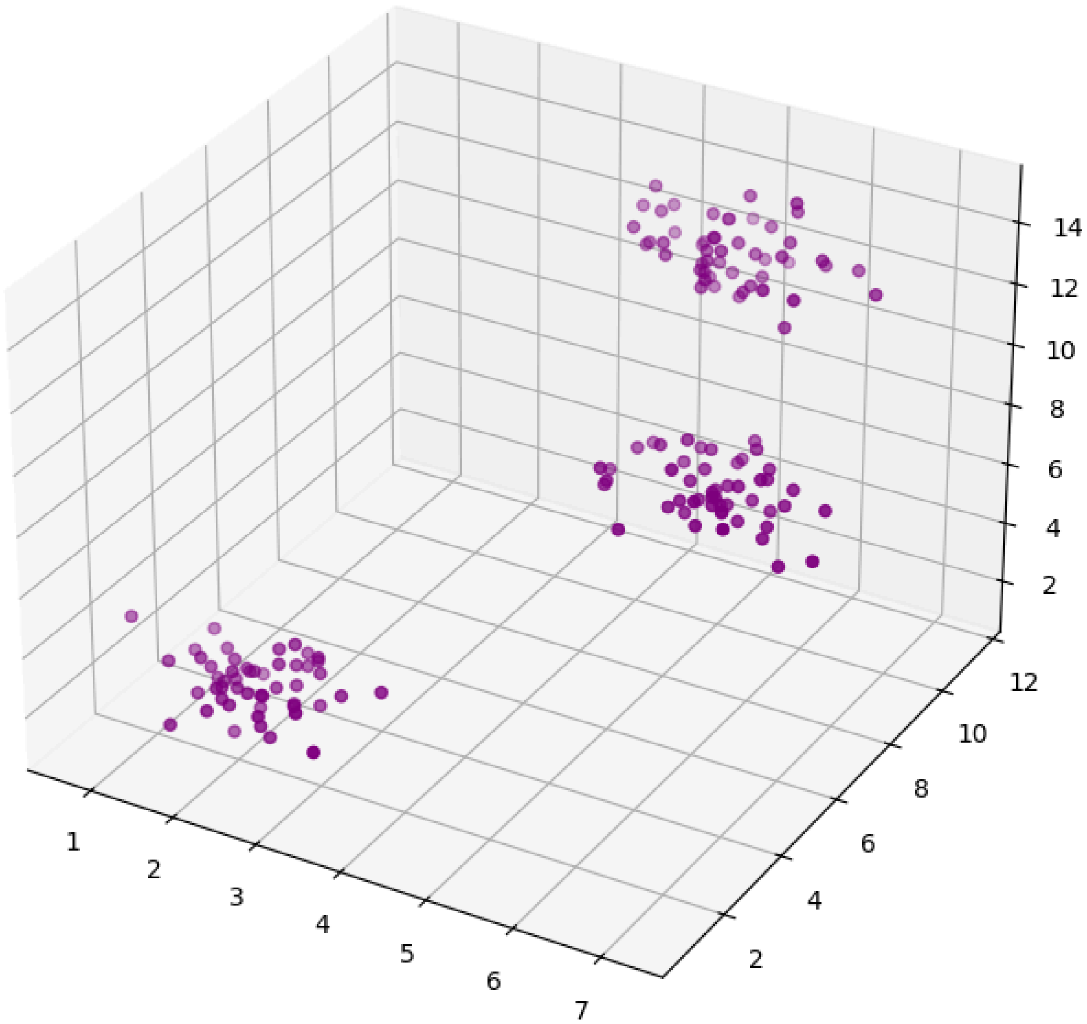
Import Necessary Libraries

In [23]:
```
import pandas as pd                    # For data manipulation and analysis
import numpy as np                     # For numerical computations
import matplotlib.pyplot as plt  # For plotting graphs

from sklearn.cluster import MeanShift                # For MeanShift cluste
from sklearn.datasets import make_blobs              # For generating sampl
from sklearn.cluster import estimate_bandwidth       # For estimating bandw

# Import Axes3D from mpl_toolkits.mplot3d for 3D plotting
from mpl_toolkits.mplot3d import Axes3D
```

Make Data Points (Blobs)

In [24]:
```
# Define the coordinates of the centers for the clusters
coordinates = [[2, 2, 3], [6, 7, 8], [5, 10, 13]]

# Generate sample data with 150 data points
# The centers of the clusters are specified by the coordinates
# The cluster_std parameter determines the standard deviation of the clus
x, _ = make_blobs(n_samples=150, centers=coordinates, cluster_std=0.60)
```

In [25]:
```
# Create a new figure with a specified size
data_fig = plt.figure(figsize=(10, 8))

# Add a 3D subplot to the figure
ax = data_fig.add_subplot(111, projection='3d')

# Create a 3D scatter plot of the data points
ax.scatter(x[:, 0], x[:, 1], x[:, 2], marker='o', color='purple')

# Display the plot
plt.show()
```



Mean Shift Clustering

In [26]:
```
# Estimate the bandwidth parameter for Mean Shift clustering
bandwidth = estimate_bandwidth(x, quantile=0.2, n_samples=500)
```

In [27]:
```
# Initialize MeanShift clustering with the estimated bandwidth and bin se
msc = MeanShift(bandwidth=bandwidth, bin_seeding=True)

# Fit the MeanShift model to the data
msc.fit(x)

# Retrieve the cluster centers
cluster_centers = msc.cluster_centers_

# Retrieve the cluster labels assigned to each data point
labels = msc.labels_

# Get the unique cluster labels
cluster_label = np.unique(labels)

# Count the number of clusters
n_clusters = len(cluster_label)
```

```
# Display the number of clusters
n_clusters
```

Out[27]: 3

Plot the Mean Shift Clusters

In [28]:
```
# Create a new figure with a specified size
msc_fig = plt.figure(figsize=(10, 8))

# Add a 3D subplot to the figure
ax = msc_fig.add_subplot(111, projection='3d')

# Create a 3D scatter plot of the data points
ax.scatter(x[:, 0], x[:, 1], x[:, 2], marker='o', color='purple')

# Plot the cluster centers as large green markers
ax.scatter(cluster_centers[:, 0], cluster_centers[:, 1], cluster_centers[
           marker='o', color='green', s=300, lw=5, zorder=10)

# Set the title of the plot
plt.title('Estimated Number of Clusters: %d' % n_clusters)

# Display the plot
plt.show()
```

Estimated Number of Clusters: 3