Introduction

This project utilizes the K-Means Clustering algorithm to segment customers based on their Annual Income and Spending Scorefrom a mall dataset.

Goal is to identify distinct customer groups that can be targeted with specific marketing strategies or business insights. We will preprocess the data, determine the optimal number of clusters, apply the K-Means algorithm, visualize the results, and interpret the clusters.

In [86]:

```
pip install kneed
```

```
Requirement already satisfied: kneed in /usr/local/lib/python3.10/dist-packages (0.8.5)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.10/dist-packages (
from kneed) (1.26.4)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (f
rom kneed) (1.13.1)
```

In [87]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from kneed import KneeLocator
```

Here NumPy and Pandas are used for **data manipulation and handling.**

Matplotlib and Seaborn are for creating visualizations.

StandardScaler helps **scale the features** to ensure all features contribute equally to the clustering.

LabelEncoder is used to **convert categorical values (like Gender) to numerical values.**

KMeans is the **clustering algorithm** that will be used to divide customers into groups.

KneeLocator helps identify the **"elbow"** point in the WCSS (Within-Cluster Sum of Squares) graph **to find the optimal number of clusters.**

IMPORTING THE DATASET

In [88]:

```
data = pd.read_csv('/content/Mall_Customers.csv')
print(data)
```

```
     CustomerID  Gender  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
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
```

TO DISPLAY FIRST FEW ROWS OF THE DATA STRUCTURE

TO DISPLAY FIRST FEW ROWS OF THE DATA STRUCTURE

In [89]:

```
data.head()
```

Out[89]:

	CustomerID	Gender	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

TO DISPLAY LAST FEW ROWS OF THE DATA STRUCTURE

In [90]:

```
data.tail()
```

Out[90]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

TO FIND NUMBER OF ROWS AND COLUMNS

In [91]:

```
data.shape
```

Out[91]:

```
(200, 5)
```

TO PROVIDE STATISTICAL SUMMAIRES OF DATA STRUCTURE

In [92]:

```
data.describe()
```

Out[92]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

DISPLAY BASIC INFORMATION ABOUT THE COLUMNS INCLUDING DATATYPES.

In [93]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

CHECK FOR ANY NULL VALUES TO ENUSRE DATA QUALITY

In [94]:

```
data.isnull().sum()
```

Out[94]:

	0
CustomerID	0
Gender	0
Age	0
Annual Income (k$)	0
Spending Score (1-100)	0

dtype: int64

to Encode categorical columns like Gender

beacuse without encoding of feature column gender into float it is not possible to calculate the correlation values unless all the values are numerical.

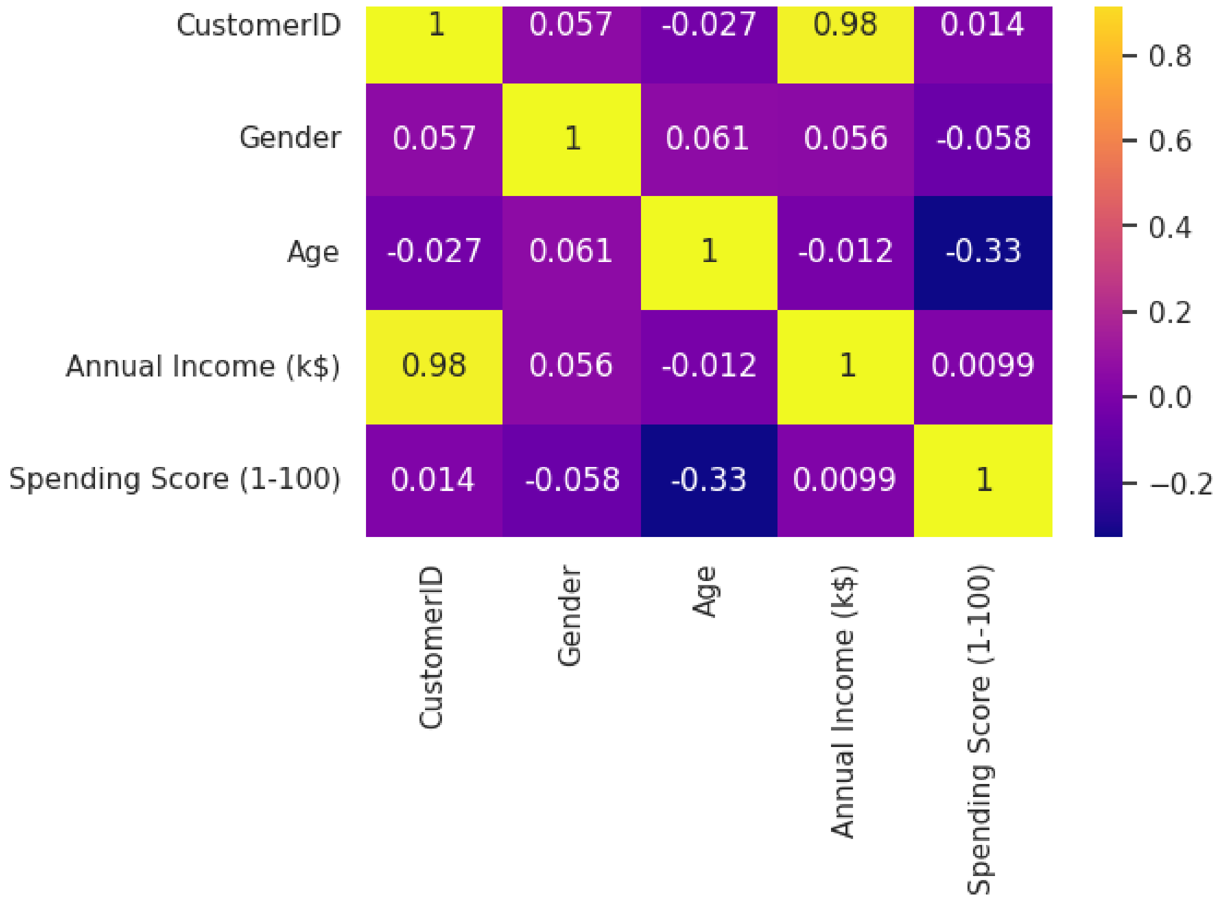
The Gender column is encoded from text ('Male'/'Female') into numerical values (0/1) because most machine learning algorithms, including K-Means, require numerical input.

The LabelEncoder converts 'Male' to 1 and 'Female' to 0.

ThE below matrix helps to identify relationships between features, guiding us to choose relevant features for clustering.

In [95]:

```
# Encode categorical columns like Gender
data['Gender'] = LabelEncoder().fit_transform(data['Gender'])

# Re-generate heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(data.corr(), annot=True, cmap='plasma')
plt.title('Feature Correlation Matrix (with Encoded Gender)')
plt.show()
```



SELECTING FEATURES FOR CLUSTERING

Choosing the Annual Income Column & Spending Score column

In [96]:

```
X = data.iloc[:,[3,4]].values
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 [97]:

```
# Scaling the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print("Scaled Data:")
print(X_scaled)
```

```
Scaled Data:
[[-1.73899919 -0.43480148]
 [-1.73899919  1.19570407]
 [-1.70082976 -1.71591298]
 [-1.70082976  1.04041783]
 [-1.66266033 -0.39597992]
 [-1.66266033  1.00159627]
 [-1.62449091 -1.71591298]
 [-1.62449091  1.70038436]
 [-1.58632148 -1.83237767]
 [-1.58632148  0.84631002]
 [-1.58632148 -1.4053405 ]
 [-1.58632148  1.89449216]
 [-1.54815205 -1.36651894]
 [-1.54815205  1.04041783]
 [-1.54815205 -1.44416206]
 [-1.54815205  1.11806095]
 [-1.50998262 -0.59008772]
 [-1.50998262  0.61338066]
 [-1.43364376 -0.82301709]
 [-1.43364376  1.8556706 ]
 [-1.39547433 -0.59008772]
 [-1.39547433  0.88513158]
 [-1.3573049  -1.75473454]
 [-1.3573049   0.88513158]
 [-1.24279661 -1.4053405 ]
 [-1.24279661  1.23452563]
 [-1.24279661 -0.7065524 ]
 [-1.24279661  0.41927286]
 [-1.20462718 -0.74537397]
 [-1.20462718  1.42863343]
 [-1.16645776 -1.7935561 ]
 [-1.16645776  0.88513158]
 [-1.05194947 -1.7935561 ]
 [-1.05194947  1.62274124]
 [-1.05194947 -1.4053405 ]
 [-1.05194947  1.19570407]
 [-1.01378004 -1.28887582]
 [-1.01378004  0.88513158]
```

```
 [-0.89927175 -0.93948177]
 [-0.89927175  0.96277471]
 [-0.86110232 -0.59008772]
 [-0.86110232  1.62274124]
 [-0.82293289 -0.55126616]
 [-0.82293289  0.41927286]
 [-0.82293289 -0.86183865]
 [-0.82293289  0.5745591 ]
 [-0.78476346  0.18634349]
 [-0.78476346 -0.12422899]
 [-0.78476346 -0.3183368 ]
 [-0.78476346 -0.3183368 ]
 [-0.70842461  0.06987881]
 [-0.70842461  0.38045129]
 [-0.67025518  0.14752193]
 [-0.67025518  0.38045129]
 [-0.67025518 -0.20187212]
 [-0.67025518 -0.35715836]
 [-0.63208575 -0.00776431]
 [-0.63208575 -0.16305055]
 [-0.55574689  0.03105725]
 [-0.55574689 -0.16305055]
 [-0.55574689  0.22516505]
 [-0.55574689  0.18634349]
 [-0.51757746  0.06987881]
 [-0.51757746  0.34162973]
 [-0.47940803  0.03105725]
 [-0.47940803  0.34162973]
 [-0.47940803 -0.00776431]
 [-0.47940803 -0.08540743]
 [-0.47940803  0.34162973]
 [-0.47940803 -0.12422899]
 [-0.4412386   0.18634349]
 [-0.4412386  -0.3183368 ]
 [-0.40306917 -0.04658587]
 [-0.40306917  0.22516505]
 [-0.25039146 -0.12422899]
 [-0.25039146  0.14752193]
 [-0.25039146  0.10870037]
 [-0.25039146 -0.08540743]
 [-0.25039146  0.06987881]
 [-0.25039146 -0.3183368 ]
 [-0.25039146  0.03105725]
 [-0.25039146  0.18634349]
 [-0.25039146 -0.35715836]
 [-0.25039146 -0.24069368]
 [-0.25039146  0.26398661]
 [-0.25039146 -0.16305055]
 [-0.13588317  0.30280817]
 [-0.13588317  0.18634349]
 [-0.09771374  0.38045129]
 [-0.09771374 -0.16305055]
 [-0.05954431  0.18634349]
 [-0.05954431 -0.35715836]
 [-0.02137488 -0.04658587]
 [-0.02137488 -0.39597992]
 [-0.02137488 -0.3183368 ]
 [-0.02137488  0.06987881]
 [-0.02137488 -0.12422899]
 [-0.02137488 -0.00776431]
 [ 0.01679455 -0.3183368 ]
 [ 0.01679455 -0.04658587]
 [ 0.05496398 -0.35715836]
 [ 0.05496398 -0.08540743]
 [ 0.05496398  0.34162973]
 [ 0.05496398  0.18634349]
 [ 0.05496398  0.22516505]
 [ 0.05496398 -0.3183368 ]
 [ 0.09313341 -0.00776431]
 [ 0.09313341 -0.16305055]
 [ 0.09313341 -0.27951524]
 [ 0.09313341 -0.08540743]
```

```
 [ 0.09313341  0.06987881]
 [ 0.09313341  0.14752193]
 [ 0.13130284 -0.3183368 ]
 [ 0.13130284 -0.16305055]
 [ 0.16947227 -0.08540743]
 [ 0.16947227 -0.00776431]
 [ 0.16947227 -0.27951524]
 [ 0.16947227  0.34162973]
 [ 0.24581112 -0.27951524]
 [ 0.24581112  0.26398661]
 [ 0.24581112  0.22516505]
 [ 0.24581112 -0.39597992]
 [ 0.32214998  0.30280817]
 [ 0.32214998  1.58391968]
 [ 0.36031941 -0.82301709]
 [ 0.36031941  1.04041783]
 [ 0.39848884 -0.59008772]
 [ 0.39848884  1.73920592]
 [ 0.39848884 -1.52180518]
 [ 0.39848884  0.96277471]
 [ 0.39848884 -1.5994483 ]
 [ 0.39848884  0.96277471]
 [ 0.43665827 -0.62890928]
 [ 0.43665827  0.80748846]
 [ 0.4748277  -1.75473454]
 [ 0.4748277   1.46745499]
 [ 0.4748277  -1.67709142]
 [ 0.4748277   0.88513158]
 [ 0.51299713 -1.56062674]
 [ 0.51299713  0.84631002]
 [ 0.55116656 -1.75473454]
 [ 0.55116656  1.6615628 ]
 [ 0.58933599 -0.39597992]
 [ 0.58933599  1.42863343]
 [ 0.62750542 -1.48298362]
 [ 0.62750542  1.81684904]
 [ 0.62750542 -0.55126616]
 [ 0.62750542  0.92395314]
 [ 0.66567484 -1.09476801]
 [ 0.66567484  1.54509812]
 [ 0.66567484 -1.28887582]
 [ 0.66567484  1.46745499]
 [ 0.66567484 -1.17241113]
 [ 0.66567484  1.00159627]
 [ 0.66567484 -1.32769738]
 [ 0.66567484  1.50627656]
 [ 0.66567484 -1.91002079]
 [ 0.66567484  1.07923939]
 [ 0.66567484 -1.91002079]
 [ 0.66567484  0.88513158]
 [ 0.70384427 -0.59008772]
 [ 0.70384427  1.27334719]
 [ 0.78018313 -1.75473454]
 [ 0.78018313  1.6615628 ]
 [ 0.93286085 -0.93948177]
 [ 0.93286085  0.96277471]
 [ 0.97103028 -1.17241113]
 [ 0.97103028  1.73920592]
 [ 1.00919971 -0.90066021]
 [ 1.00919971  0.49691598]
 [ 1.00919971 -1.44416206]
 [ 1.00919971  0.96277471]
 [ 1.00919971 -1.56062674]
 [ 1.00919971  1.62274124]
 [ 1.04736914 -1.44416206]
 [ 1.04736914  1.38981187]
 [ 1.04736914 -1.36651894]
 [ 1.04736914  0.72984534]
 [ 1.23821628 -1.4053405 ]
 [ 1.23821628  1.54509812]
 [ 1.390894   -0.7065524 ]
 [ 1.390894    1.38981187]
```

```
 [ 1.42906343 -1.36651894]
 [ 1.42906343  1.46745499]
 [ 1.46723286 -0.43480148]
 [ 1.46723286  1.81684904]
 [ 1.54357172 -1.01712489]
 [ 1.54357172  0.69102378]
 [ 1.61991057 -1.28887582]
 [ 1.61991057  1.35099031]
 [ 1.61991057 -1.05594645]
 [ 1.61991057  0.72984534]
 [ 2.00160487 -1.63826986]
 [ 2.00160487  1.58391968]
 [ 2.26879087 -1.32769738]
 [ 2.26879087  1.11806095]
 [ 2.49780745 -0.86183865]
 [ 2.49780745  0.92395314]
 [ 2.91767117 -1.25005425]
 [ 2.91767117  1.27334719]]
```

StandardScaler is used to scale the features (Annual Income and Spending Score).

Scaling ensures both features have the same scale (mean of 0, standard deviation of 1).

This is important for K-Means, as the algorithm uses distance calculations (Euclidean distance) between points, and unscaled data could bias the clustering process.

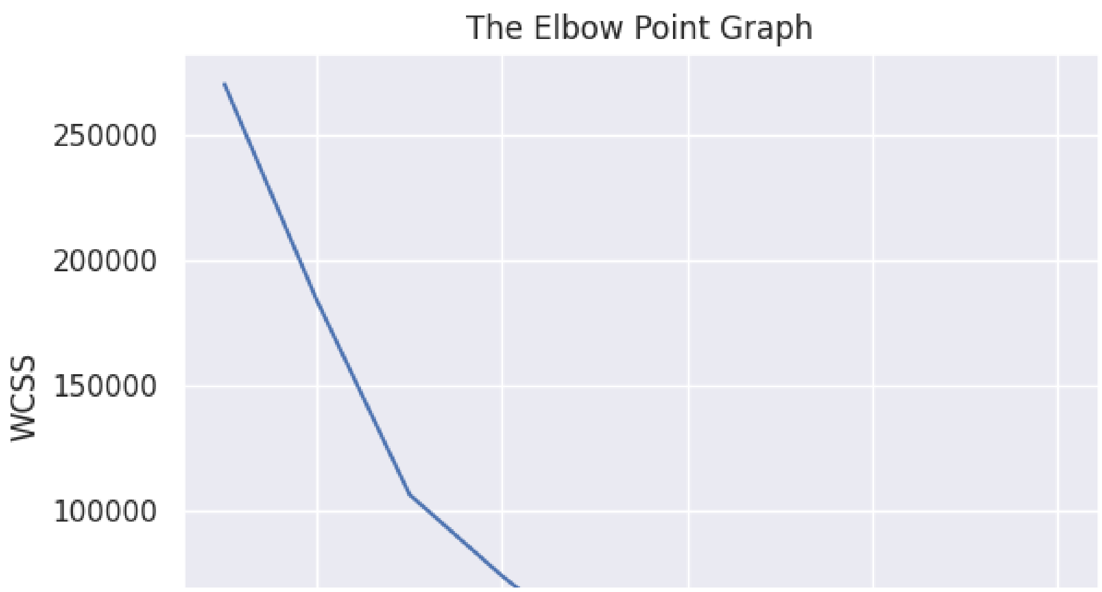
FINDING THE OPTIMUM NUMBER OF CLUSTERS USING ELBOW METHOD

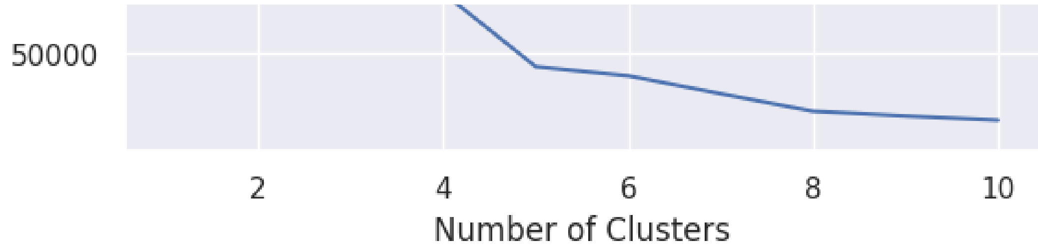
In [98]:

```
# finding wcss(Within-Cluster Sum of Squares) value for different number of clusters

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_)
```

In [99]:

```
#Elbow graph

sns.set()
plt.plot(range(1,11), wcss)
plt.title('The Elbow Point Graph')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```





In [100]:

```
# Optimal number of clusters using KneeLocator
knee = KneeLocator(range(1, 11), wcss, curve="convex", direction="decreasing")
optimal_clusters = knee.knee
print(f"Optimal number of clusters: {optimal_clusters}")
```

```
Optimal number of clusters: 5
```

From the graph, it is seen that the optimum number of clusters is 5. To cross-validate this, the above method using the "kneedle" technique was applied for verification.

Next, the k-Means Clustering model is being trained.

TRAINING THE K-MEANS MODEL

In [101]:

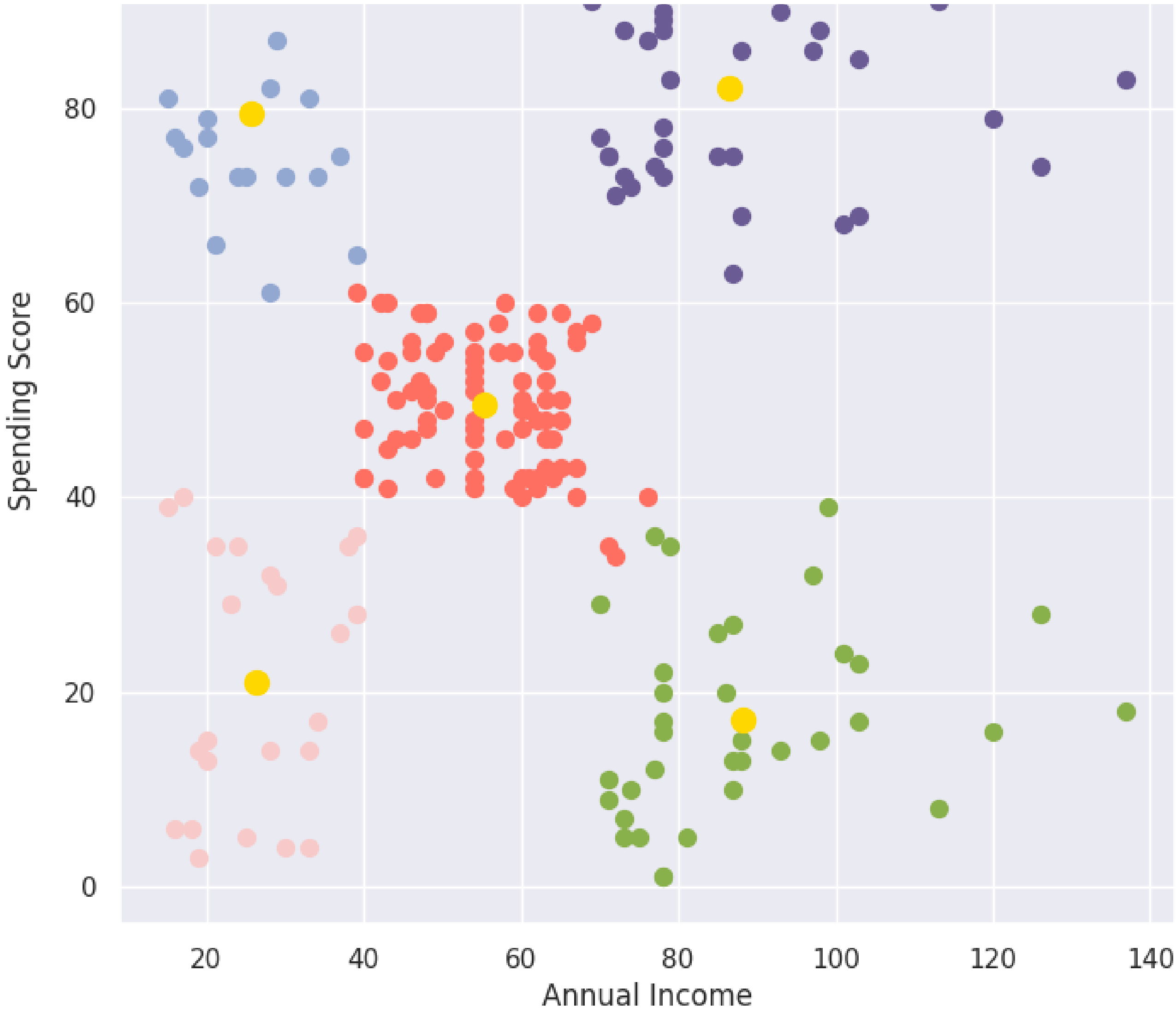
```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0)

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

print(Y)
```

```
[3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
 4 3 4 3 4 3 0 3 4 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 1 2 1 0 1 2 1 2 1 0 1 2 1 2 1 2 1 2 1 0 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1]
```

There are 5 Clusters - 0, 1, 2, 3, 4

In [102]:

```
# 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='#FF6F61', label='Cluster 1')
plt.scatter(X[Y==1,0], X[Y==1,1], s=50, c='#6B5B95', label='Cluster 2')
plt.scatter(X[Y==2,0], X[Y==2,1], s=50, c='#88B04B', label='Cluster 3')
plt.scatter(X[Y==3,0], X[Y==3,1], s=50, c='#F7CAC9', label='Cluster 4')
plt.scatter(X[Y==4,0], X[Y==4,1], s=50, c='#92A8D1', label='Cluster 5')

# plot the centroids
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s=100, c='#FFD70
0', label='Centroids')

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



PROFILING EACH CLUSTER AND PLOTTING OTHER PLOT

In [103]:

```
# Cluster Profiles
data = data.copy()
data['Cluster'] = Y
for i in range(optimal_clusters):
    print(f"\nCluster {i} Statistics:")
    print(data[data['Cluster'] == i].describe())


# Optional plot: Interactive Visualization using Plotly
import plotly.express as px
fig = px.scatter(data, x='Annual Income (k$)', y='Spending Score (1-100)',
                 color=Y.astype(str), title='Clusters of Customers',
                 labels={'color': 'Cluster'})
fig.show()
```

```
Cluster 0 Statistics:
       CustomerID     Gender        Age  Annual Income (k$)  \
count   81.000000  81.000000  81.000000           81.000000
mean    86.320988   0.407407  42.716049           55.296296
std     24.240889   0.494413  16.447822            8.988109
min     44.000000   0.000000  18.000000           39.000000
25%     66.000000   0.000000  27.000000           48.000000
50%     86.000000   0.000000  46.000000           54.000000
75%    106.000000   1.000000  54.000000           62.000000
max    143.000000   1.000000  70.000000           76.000000

       Spending Score (1-100)  Cluster
count               81.000000     81.0
mean                49.518519      0.0
std                  6.530909      0.0
```

```
min                34.000000      0.0
25%                44.000000      0.0
50%                50.000000      0.0
75%                55.000000      0.0
max                61.000000      0.0

Cluster 1 Statistics:
       CustomerID     Gender        Age  Annual Income (k$)  \
count   39.000000  39.000000  39.000000           39.000000
mean   162.000000   0.461538  32.692308           86.538462
std     22.803509   0.505035   3.728650           16.312485
min    124.000000   0.000000  27.000000           69.000000
25%    143.000000   0.000000  30.000000           75.500000
50%    162.000000   0.000000  32.000000           79.000000
75%    181.000000   1.000000  35.500000           95.000000
max    200.000000   1.000000  40.000000          137.000000

       Spending Score (1-100)  Cluster
count               39.000000     39.0
mean                82.128205      1.0
std                  9.364489      0.0
min                 63.000000      1.0
25%                 74.500000      1.0
50%                 83.000000      1.0
75%                 90.000000      1.0
max                 97.000000      1.0

Cluster 2 Statistics:
       CustomerID     Gender        Age  Annual Income (k$)  \
count   35.000000  35.000000  35.000000           35.000000
mean   164.371429   0.542857  41.114286           88.200000
std     21.457325   0.505433  11.341676           16.399067
min    125.000000   0.000000  19.000000           70.000000
25%    148.000000   0.000000  34.000000           77.500000
50%    165.000000   1.000000  42.000000           85.000000
75%    182.000000   1.000000  47.500000           97.500000
max    199.000000   1.000000  59.000000          137.000000

       Spending Score (1-100)  Cluster
count               35.000000     35.0
mean                17.114286      2.0
std                  9.952154      0.0
min                  1.000000      2.0
25%                 10.000000      2.0
50%                 16.000000      2.0
75%                 23.500000      2.0
max                 39.000000      2.0

Cluster 3 Statistics:
       CustomerID     Gender        Age  Annual Income (k$)  \
count    23.00000  23.000000  23.000000           23.000000
mean     23.00000   0.391304  45.217391           26.304348
std      13.56466   0.499011  13.228607            7.893811
min       1.00000   0.000000  19.000000           15.000000
25%      12.00000   0.000000  35.500000           19.500000
50%      23.00000   0.000000  46.000000           25.000000
75%      34.00000   1.000000  53.500000           33.000000
max      45.00000   1.000000  67.000000           39.000000

       Spending Score (1-100)  Cluster
count               23.000000     23.0
mean                20.913043      3.0
std                 13.017167      0.0
min                  3.000000      3.0
25%                  9.500000      3.0
50%                 17.000000      3.0
75%                 33.500000      3.0
max                 40.000000      3.0

Cluster 4 Statistics:
       CustomerID     Gender        Age  Annual Income (k$)  \
count   22.000000  22.000000  22.000000           22.000000
```

```
mean     23.090909   0.409091  25.272727             25.727273
std      13.147185   0.503236   5.257030              7.566731
min       2.000000   0.000000  18.000000             15.000000
25%      12.500000   0.000000  21.250000             19.250000
50%      23.000000   0.000000  23.500000             24.500000
75%      33.500000   1.000000  29.750000             32.250000
max      46.000000   1.000000  35.000000             39.000000

       Spending Score (1-100)  Cluster
count               22.000000     22.0
mean                79.363636      4.0
std                 10.504174      0.0
min                 61.000000      4.0
25%                 73.000000      4.0
50%                 77.000000      4.0
75%                 85.750000      4.0
max                 99.000000      4.0
```

CONCLUSION :

This project successfully segments customers into 5 distinct groups based on Annual Income and Spending Score using K-Means Clustering. By profiling these clusters, we gain valuable insights into customer behavior, which can be useful for targeted marketing and customer engagement strategies. The optimal number of clusters was determined using both the Elbow Method and KneeLocator for accuracy. The final model has been saved and can be reused for future predictions or insights.