Assignment 1A: Unsupervised Learning, K-means modeling

Objective: In this assignment we will build kmeans algorithm from scratch, discuss k-means variations, and identify customer profiles using k-means from sklearn.

Please do not share this material on any platform or by any other means.

Important Notes:

> Make changes to the cells that have # YOUR CODE HERE or # YOUR COMMENT HERE.Do not write your answer in anywhere else other than where it says YOUR CODE HERE (or YOUR COMMENT HERE).
>
> Your code must run without any errors start to end. Please go to menubar, select Kernel, and restart the kernel and run all cells (Restart & Run all) before submitting your work.
>
> Purpose of the assignment is to assess your knowledge and command of the data mining algorithm, python programming language, and your ability to resolve common errors. Grading is based on the code and your interpretation/comments you are submitting, not the formatting of the results.
>
> Please use the examples as a guideline, you are not expected to have the same formatting as the example, unless it is a formatting question.

Remember, there are many ways to code that can lead to the correct answer, do not hesitate to exercise your own style and python programming coventions.

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd
import random
import warnings
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

Q1: Building k-means algorithm from scratch!

K-means is one of the simpler clustering algorithms, where k is the user-specified number of clusters, where the goal is to partition the inputs into k-many sets in a way that minimizes the total sum of suaqred distances from each point to the mean of its assigned cluster.

Here is an iterative algorithm for k-means:

1. start with a set of k-means (these are points in d-dimentional space)

2. assign each point to the mean to which it is closest.

3. if no point's assignment has changed, recompute the means and return to step 2.

You may need to write multiple functions!

In [2]:
```
class kMeansScratch:
    ## The data should be a list where each element represents a single data point. For
    def __init__(self, k, max_iterations=100):
        self.k = k
        self.max_iterations = max_iterations

    def _initialize_means(self, data):
        """ Initialize k random means from the dataset. """
        self.means = random.sample(data, self.k)

    def _assign_points_to_means(self, data):
        """ Assign each point to the cluster with the closest mean. """
        clusters = {}
        for x in data: ## data is
            closest_mean = min(self.means, key=lambda mean: np.linalg.norm(np.array(x) -
            if closest_mean in clusters:
                clusters[closest_mean].append(x)
            else:
                clusters[closest_mean] = [x]
        return clusters

    def _recompute_means(self, clusters):
        """ Recompute the means of each cluster. """
        new_means = []
        for cluster in clusters:
            new_mean = np.mean(clusters[cluster], axis=0)
            new_means.append(new_mean.tolist())
        return new_means ## Return k lists

    def _check_convergence(self, new_means):
        """ Check if the algorithm has converged. """
        return set(map(tuple, self.means)) == set(map(tuple, new_means)) ## Comparing th

    def fit(self, data):
        """ The main method to fit the data to the k-means model. """
        self._initialize_means(data)
        for _ in range(self.max_iterations):
            clusters = self._assign_points_to_means(data)
            new_means = self._recompute_means(clusters)
            if self._check_convergence(new_means):
                break
            self.means = new_means
        self.clusters = clusters
```

Q2: K-means variations

What are some variations of k-means algorithms? Discuss the advantages, disadvantages, and use cases of these variations. Your discussion list should include k-means++.

k-means +++: It has a smarter to choose the initial centroids.

- Advantages: Reduces the possibility of poor clusterings due to unfortunate initial mean choices. Tends to find solutions that are closer to the global optimum than standard k-means

- Disadvantages: Only identify spherical or circular in shape like k-means. Cost-expensive in selecting initial centroids.
- Use Cases: Same use cases as k-means does. Ideal for applications where the quality of clustering is paramount, and the added computational cost in the initialization phase is acceptable

Mini-Batch K-Means: Uses small, random batches of data points to update the cluster centroids, rather than the entire dataset

- Advantages: Can handle larger datasets that do not fit into memory. And reduce the computational cost.
- Disadvantages: May lead to slightly worse results due to the randomness in mini-batches so that user must carefully consider the initial centroids
- Use Cases: Suitable for large-scale data analysis where computational efficiency is crucial, and some loss of accuracy is acceptable

Q3: Basic Customer Profiling using k-means

Customers are the only entities that pay, hence all revolves around the customer. As a data scientist, your objective is to understand the customers who purchase goods from the "Mall" store. How can you get to know your customers and how could this benefit your company?

You are given the "MallCustomers.csv" dataset for this analysis. Start with the steps below, and feel free to research customer profiling online.

1. Analyze, plot and discuss the data
2. Use the "Annual Income (k$) ","Spending Score (1-100) " features, and build a sklearn.cluster.KMeans model for k in (1,11). Use k-means++ to initialize your model, and random_state=42.
3. Plot the elbow chart and discuss the results
4. Discuss other ways to evaluate the results of your model (not general comments, but specific to your results).
5. Plot the "Annual Income (k$)","Spending Score (1-100)" and use your clustering model predictions to color the customers.
6. Write a short summary of all your findings and your interpretation of the results.

Analyze, plot and discuss the data

In [3]:
```
## Fetch the data
customer_data = pd.read_csv('CustomerData.csv')
```

In [4]:
```
## Check the data
customer_data.head(5)
```

Out[4]:

	CustomerID	Gender	Age	Spending Score (1-100)	Annual Income ($)
0	1	Male	19	39	$15,000.00
1	2	Male	21	81	$15,000.00

2	3	Female	20	6	$16,000.00
3	4	Female	23	77	$16,000.00
4	5	Female	31	40	$17,000.00

- CustomerID: Identifier for each customer.
- Gender: Gender of the customer (Male/Female).
- Age: Age of the customer.
- Spending Score (1-100): A score assigned to the customer based on their spending behavior.
- Annual Income ($): Annual income of the customer.

In [5]:
```
## Check data type and missing values
data_types = customer_data.dtypes
missing_values = customer_data.isnull().sum()

data_types, missing_values
```

Out[5]:
```
(CustomerID                 int64
 Gender                    object
 Age                        int64
 Spending Score (1-100)     int64
 Annual Income ($)         object
 dtype: object,
 CustomerID                0
 Gender                    0
 Age                       0
 Spending Score (1-100)    0
 Annual Income ($)         0
 dtype: int64)
```

In [6]:
```
# Cleaning the 'Annual Income ($)' column
# Removing the dollar sign and commas, and converting it to a float
customer_data['Annual Income ($)'] = customer_data['Annual Income ($)'].replace('[\$,]',

# Check the cleaned column
customer_data['Annual Income ($)'].head()
```

Out[6]:
```
0    15000.0
1    15000.0
2    16000.0
3    16000.0
4    17000.0
Name: Annual Income ($), dtype: float64
```

In [7]:
```
## Descriptive statistics
summary_statistics = customer_data.describe()
summary_statistics
```

Out[7]:

	CustomerID	Age	Spending Score (1-100)	Annual Income ($)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	50.200000	60560.000000
std	57.879185	13.969007	25.823522	26264.721165
min	1.000000	18.000000	1.000000	15000.000000
25%	50.750000	28.750000	34.750000	41500.000000
50%	100.500000	36.000000	50.000000	61500.000000
75%	150.250000	49.000000	73.000000	78000.000000

max	200.000000	70.000000	99.000000	137000.000000

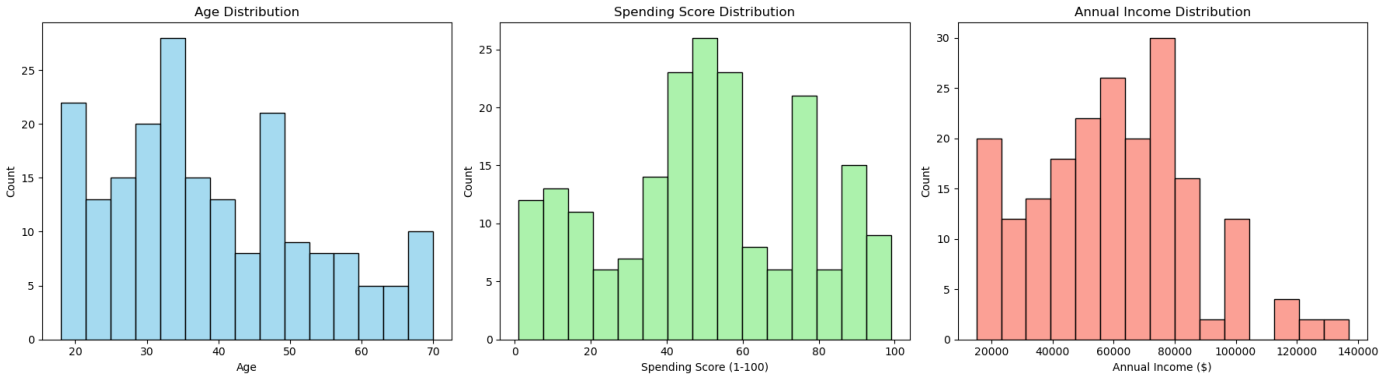
In [8]:
```
# Creating histograms for Age, Spending Score, and Annual Income
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Histogram for Age
sns.histplot(customer_data['Age'], bins=15, ax=axes[0], color='skyblue')
axes[0].set_title('Age Distribution')

# Histogram for Spending Score
sns.histplot(customer_data['Spending Score (1-100)'], bins=15, ax=axes[1], color='lightg
axes[1].set_title('Spending Score Distribution')

# Histogram for Annual Income
sns.histplot(customer_data['Annual Income ($)'], bins=15, ax=axes[2], color='salmon')
axes[2].set_title('Annual Income Distribution')

plt.tight_layout()
plt.show()
```



- Age Distribution: The age distribution is somewhat uniformly spread with a slight increase in frequency in the late 20s to early 30s age range. This suggests a diverse range of customers in terms of age.
- Spending Score Distribution: The distribution of the spending score is fairly uniform across different scores. There's no clear skewness, indicating a balanced mix of low, medium, and high spenders.
- Annual Income Distribution: The income distribution is skewed towards the lower end, with a peak around $50,000-60,000$. Higher incomes are less frequent in the dataset.

In [9]:
```
# Scatter plot for Annual Income vs. Spending Score
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Annual Income ($)', y='Spending Score (1-100)', data=customer_data, h
plt.title('Annual Income vs. Spending Score')
plt.xlabel('Annual Income ($)')
plt.ylabel('Spending Score (1-100)')
plt.legend(title='Gender')
plt.show()
```

- A cluster of customers with low annual income and low to moderate spending scores.
- A cluster with high annual income and moderate to high spending scores.
- Interestingly, there's a group with moderate annual income but very high spending scores.
- These clusters suggest different types of consumer behavior that might be interesting for targeted marketing strategies.

Build K-Means ++

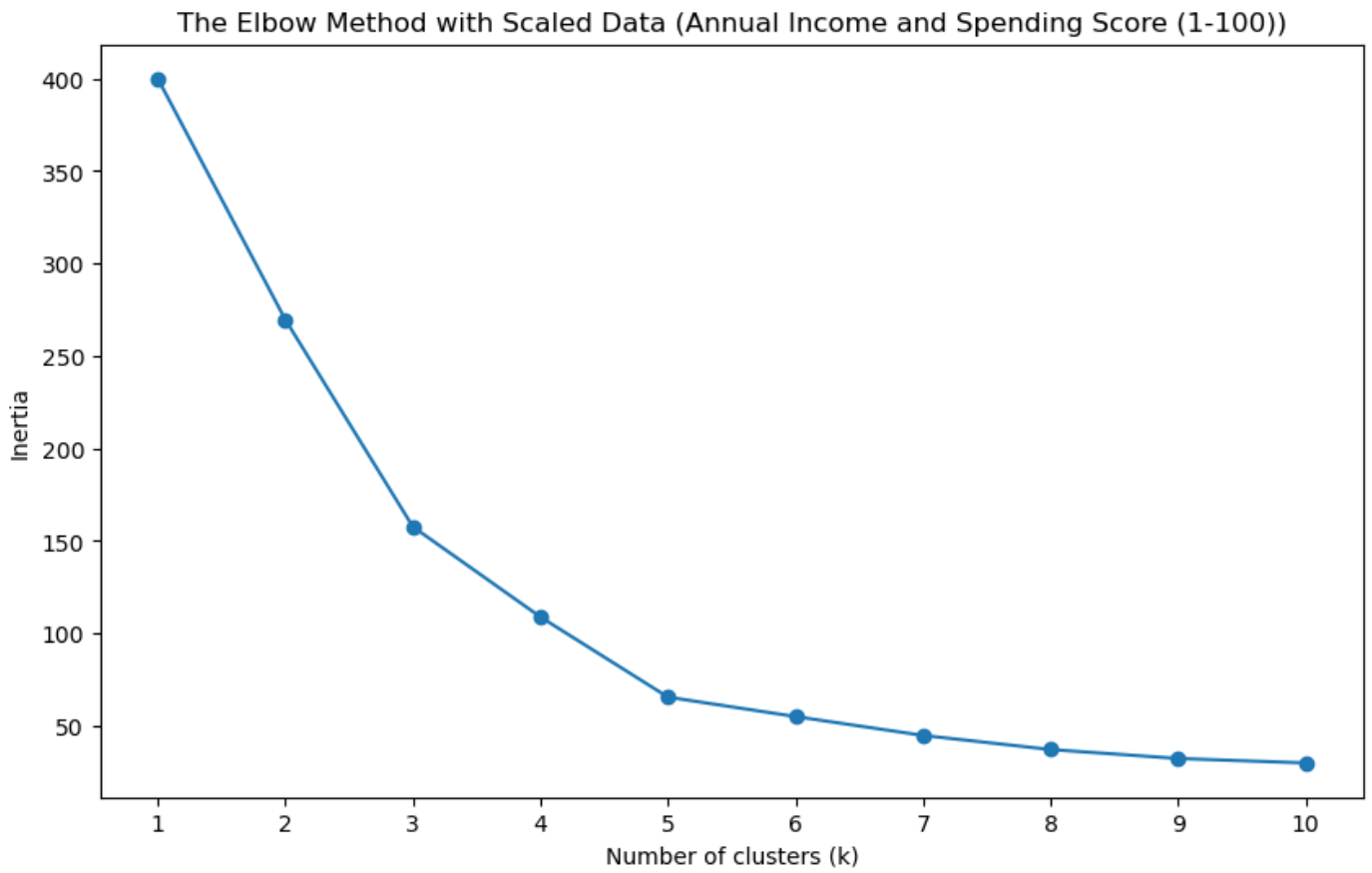
In [10]:
```
# Scale the two features required for k_means
scaled_features = scale(customer_data[['Annual Income ($)', 'Spending Score (1-100)']])
```

In [11]:
```
# Building KMeans models for k in range(1, 11) with the scaled data
inertia_scaled = []
for k in range(1, 11):
    kmeans_scaled = KMeans(n_clusters=k, init='k-means++', random_state=42)
    kmeans_scaled.fit(scaled_features)
    inertia_scaled.append(kmeans_scaled.inertia_)
```

Plot the elbow chart and discuss the results

In [12]:
```
# Plotting the inertia to observe the 'Elbow' for the scaled data
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia_scaled, marker='o')
plt.title('The Elbow Method with Scaled Data (Annual Income and Spending Score (1-100))'
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.xticks(range(1, 11))
plt.show()

# Returning the inertia values as well for the scaled data
inertia_scaled
```



Out[12]:
```
[399.99999999999994,
 269.69101219276394,
 157.70400815035947,
 108.92131661364355,
 65.56840815571681,
 55.057348270385994,
 44.86475569922556,
 37.22818767758588,
 32.39226763033117,
 29.981897788243693]
```

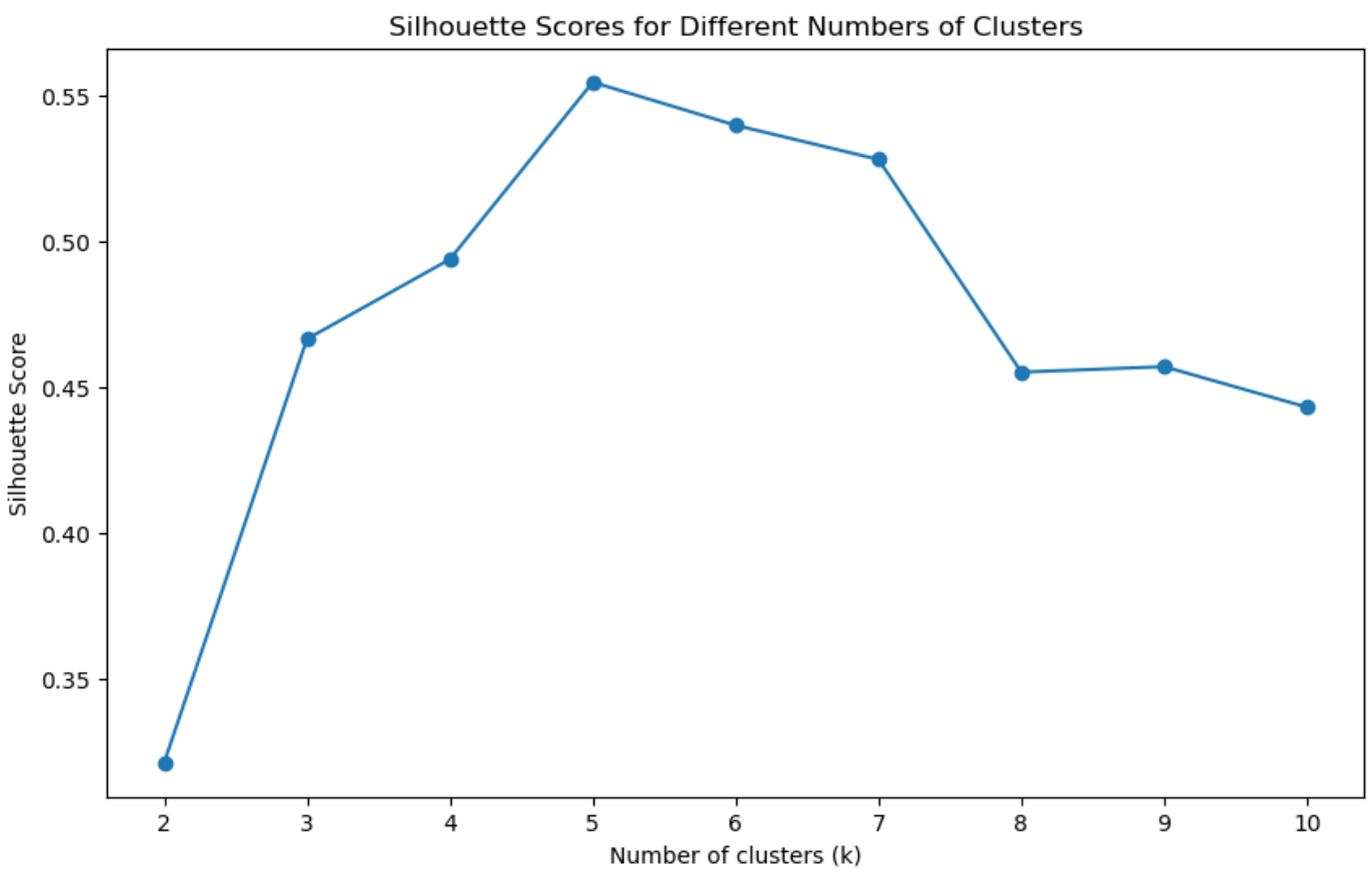
The elbow appears to be around k=5, where inertia = 65, suggesting that 5 clusters might be a suitable choice.

Other evaluation of model

In [13]:
```
# Calculating silhouette scores for each k
silhouette_scores = []
for k in range(2, 11):  # Silhouette score is not defined for k=1
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    kmeans.fit(scaled_features)
    score = silhouette_score(scaled_features, kmeans.labels_)
    silhouette_scores.append(score)

# Plotting the silhouette scores
plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.title('Silhouette Scores for Different Numbers of Clusters')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Silhouette Score')
plt.xticks(range(2, 11))
plt.show()

# Returning the silhouette scores
silhouette_scores
```



Out[13]:
```
[0.3212707813918878,
 0.46658474419000145,
 0.4939069237513199,
 0.5546571631111091,
 0.5398800926790663,
 0.5281492781108291,
 0.4552147906587443,
 0.4570853966942764,
 0.4431713026508046]
```

In this case, the highest silhouette score is observed for k=5 (approximately 0.5547), suggesting that the clustering configuration with 5 clusters is quite strong in terms of both cohesion and separation. This result aligns well with the Elbow Method we observed earlier. Other evaluation method could be a cluster visualization to check whether the clusters demonstrate reasonable clusters

Cluster visualization

In [14]:
```
# Building the KMeans model with k=5 (as suggested by both Elbow Method and Silhouette S
kmeans_final = KMeans(n_clusters=5, init='k-means++', random_state=42)
kmeans_final.fit(scaled_features)

# Predicting the clusters
clusters = kmeans_final.predict(scaled_features)
```

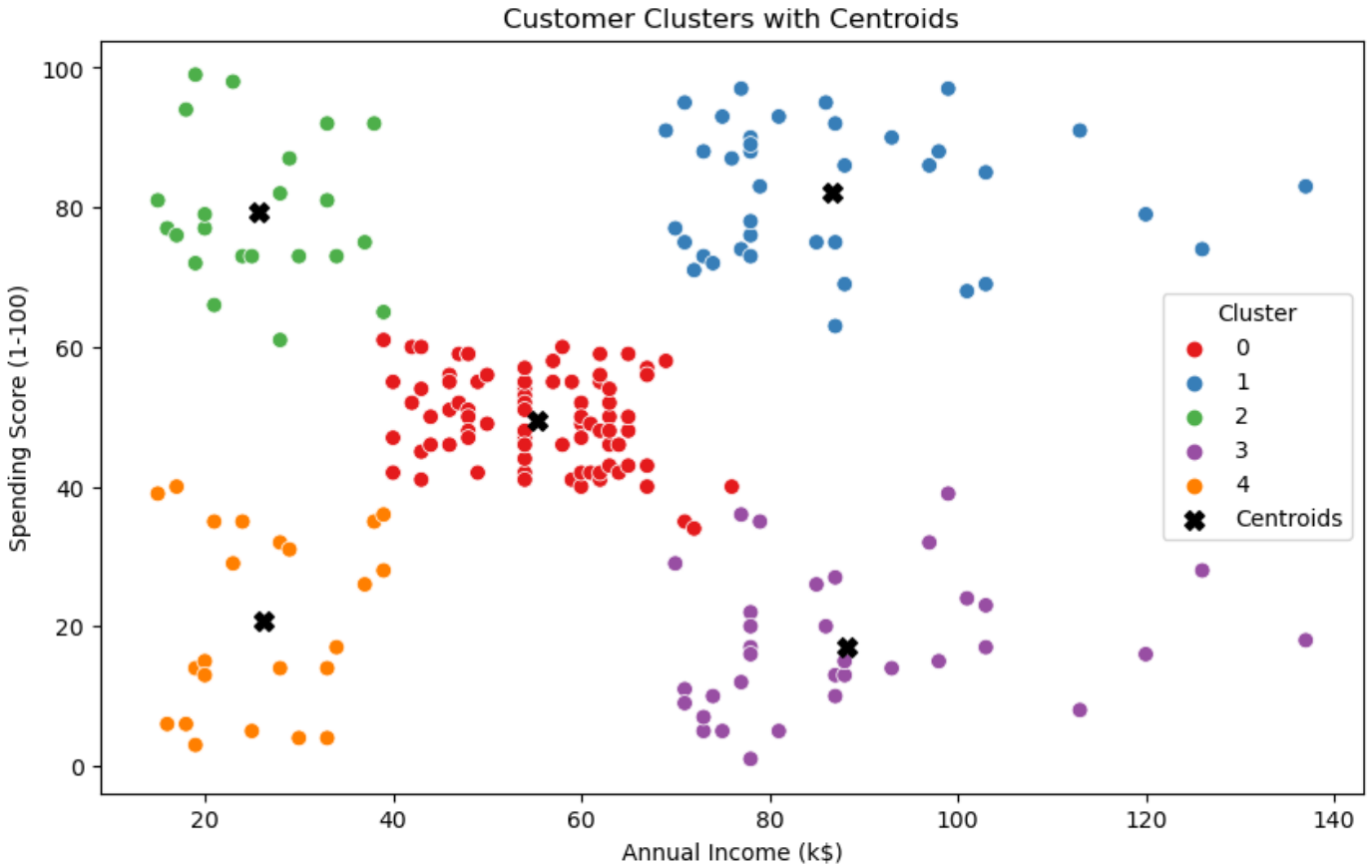
In [15]:
```
# Extracting the centroids of the clusters
centroids = kmeans_final.cluster_centers_

# Correcting the scaling transformation for the centroids
mean_vals = customer_data[['Annual Income ($)', 'Spending Score (1-100)']].mean().values
std_vals = customer_data[['Annual Income ($)', 'Spending Score (1-100)']].std().values

# Applying the inverse transformation to scale back the centroids
centroids_original_scale = centroids * std_vals + mean_vals
```

```
# Convert centroids to "k$" for annual income
centroids_original_scale[:, 0] = centroids_original_scale[:, 0] / 1000
```

In [16]:
```
# Plotting the clusters along with the centroids
plt.figure(figsize=(10, 6))
sns.scatterplot(x=customer_data['Annual Income ($)']/1000, y=customer_data['Spending Sco
                hue=clusters, palette='Set1', s=50)
plt.scatter(centroids_original_scale[:, 0], centroids_original_scale[:, 1],
            s=70, c='black', marker='X', label='Centroids')
plt.title('Customer Clusters with Centroids')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend(title='Cluster')
plt.show()
```



Summary

From the plot, we can observe the following patterns in the clusters:

- Cluster 1 (red): Customers with medium annual income and medium spending scores.
- Cluster 2 (orange):Customers with low annual income and low spending scores.
- Cluster 3 (green): Customers with low annual income and high spending scores.
- Cluster 4 (blue): Customers with high annual income and high spending scores.
- Cluster 5 (purple): Customers with high annual income but low spending scores.

These clusters can provide valuable insights for targeted marketing strategies. For example, clusters with high spending scores regardless of income (clusters 3 and 4) might be targeted for more premium products, while clusters with lower spending scores (clusters 2 and 5) might be more responsive to different types of marketing approaches or promotions

End of Assignment 1A

In [ ]: