CSBB 311 : MACHINE LEARNING

LAB ASSIGNMENT 7 : Implementation of K-Means Clustering From Scratch and using Inbuilt Library

Submitted By:

Name: KARTIK MITTAL

Roll No: 221210056

Branch: CSE

Semester: 5th Sem

Group: 2

Submitted To: Dr. Preeti Verma

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DELHI

2024

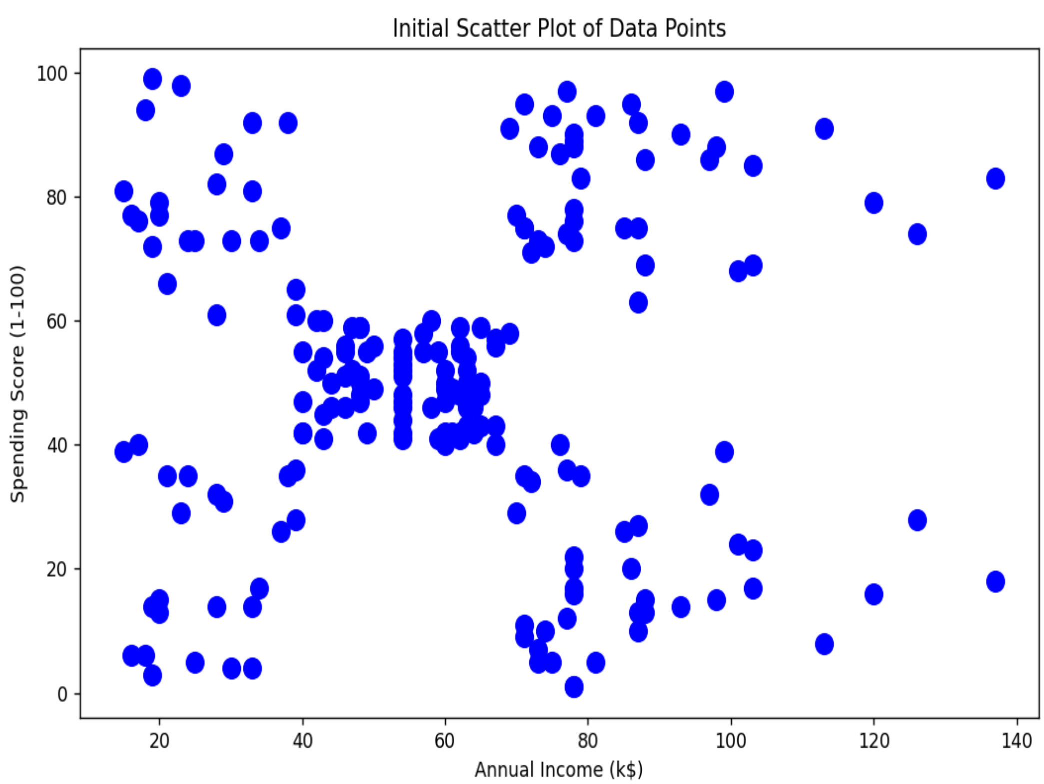
Code (Using Inbuilt Library) -

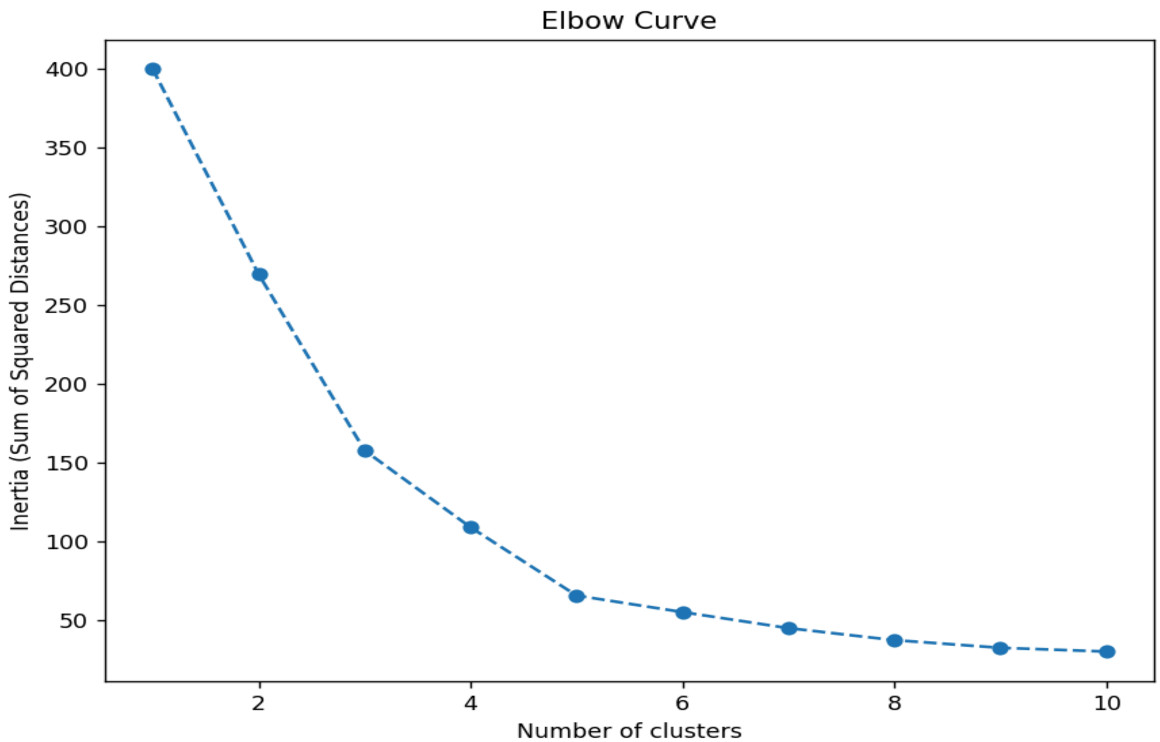
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler

# Load the data
df = pd.read_csv('Mall_Customers.csv')

# Delete rows with missing values
df.dropna(inplace=True)

# Selecting features for clustering
X = df[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Step 1: Initial scatter plot of the data points
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], color='blue', s=100)
plt.title('Initial Scatter Plot of Data Points')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()

# Step 2: Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

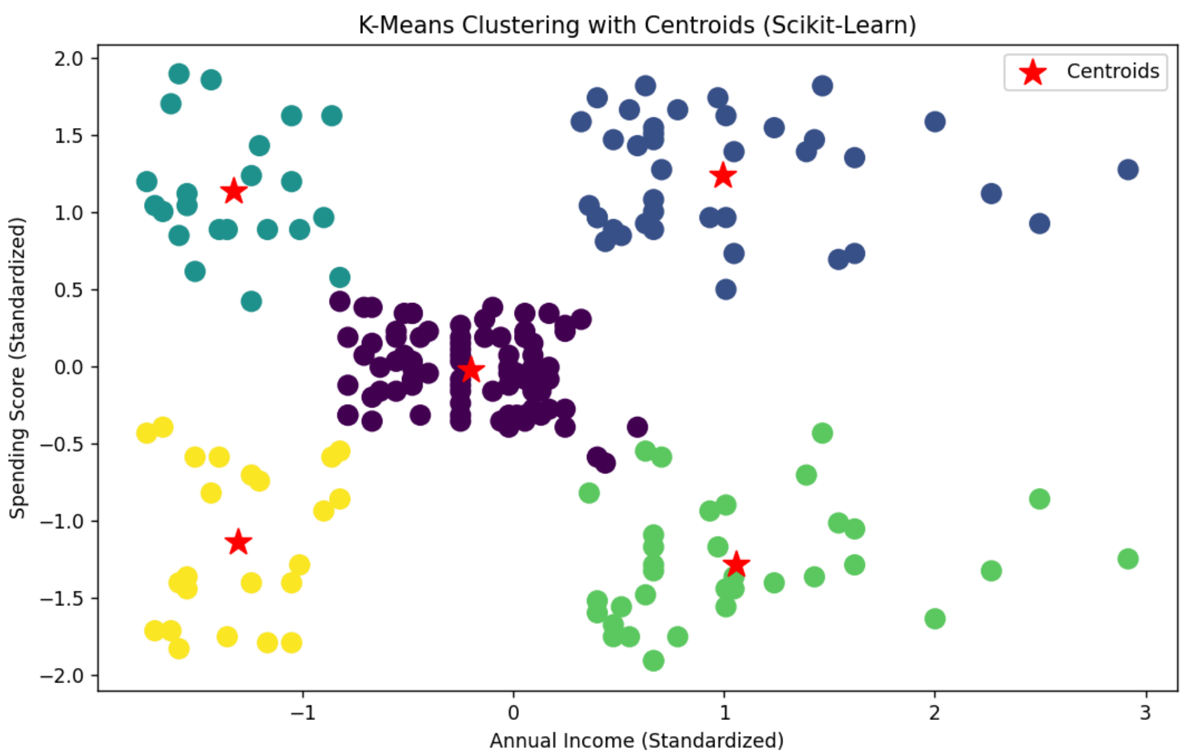
```python
# Step 3: Elbow curve to find the optimal number of clusters
inertia = []
K = range(1, 11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(8, 6))
plt.plot(K, inertia, marker='o', linestyle='--')
plt.title('Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.show()

# Step 4: Applying K-Means with the optimal number of clusters (e.g., 5 from elbow curve)
optimal_clusters = 5
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
kmeans.fit(X_scaled)

# Step 5: Visualizing the clusters with centroids
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans.labels_, cmap='viridis', s=100)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color='red',
            marker='*',s=200, label='Centroids')
plt.title('K-Means Clustering with Centroids (Scikit-Learn)')
plt.xlabel('Annual Income (Standardized)')
plt.ylabel('Spending Score (Standardized)')
```

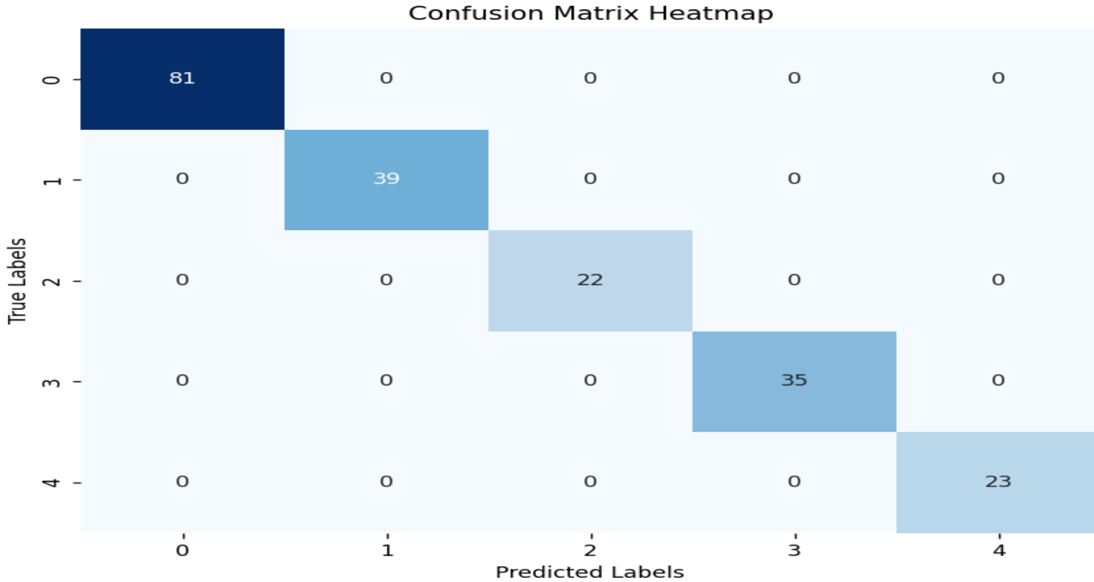
```python
plt.legend()
plt.show()

# Step 6: Confusion matrix heatmap (assuming cluster predictions vs true labels)
# In this case, we don't have true labels, but we can use the predicted labels for simulation
y_true = kmeans.labels_  # In practice, this should be true labels if available
y_pred = kmeans.predict(X_scaled)

# Create confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)

# Plotting the confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix Heatmap')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Output cluster centers
print("Cluster centers (standardized):")
print(kmeans.cluster_centers_)
```

Output -




Elbow Curve
Inertia (Sum of Squared Distances)
Number of clusters
400
350
300
250
200
150
100
50
2
4
6
8
10


K-Means Clustering with Centroids (Scikit-Learn)
Centroids
Spending Score (Standardized)
Annual Income (Standardized)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1
0
1
2
3



Code (From Scratch) -

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler

# Load the data
df = pd.read_csv('Mall_Customers.csv')

# Delete rows with missing values
df.dropna(inplace=True)

# Selecting features for clustering
X = df[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Step 1: Initial scatter plot of the data points
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], color='blue', s=100)
plt.title('Initial Scatter Plot of Data Points')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

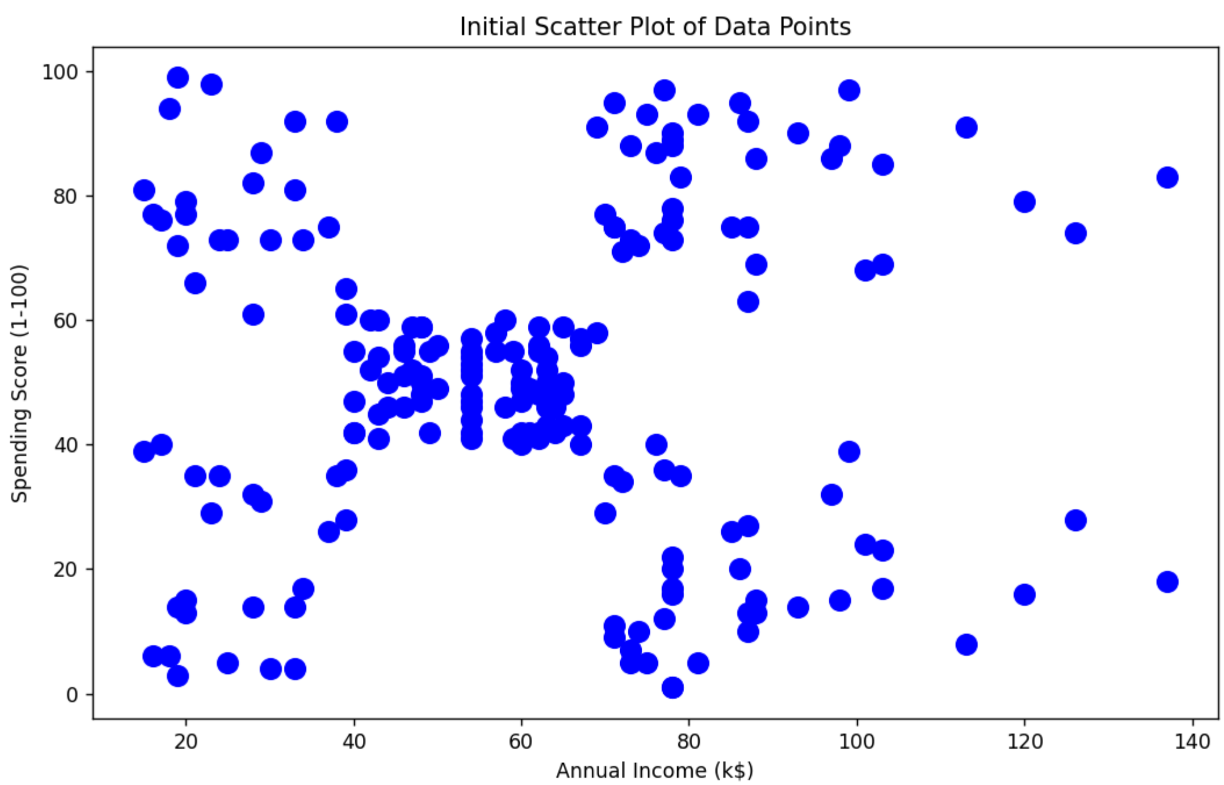
```python
# Function to calculate the Euclidean distance between points
def euclidean_distance(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

# K-Means Clustering from scratch
class KMeansScratch:
    def __init__(self, n_clusters, max_iter=100):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        # Randomly initialize cluster centers
        np.random.seed(42)
        random_idx = np.random.permutation(X.shape[0])
        self.centroids = X[random_idx[:self.n_clusters]]

        for i in range(self.max_iter):
            # Assign clusters
            self.labels = self.assign_clusters(X)

            # Compute new centroids
            new_centroids = self.calculate_centroids(X)

            # If centroids do not change, break
            if np.all(self.centroids == new_centroids):
                break
            self.centroids = new_centroids

```

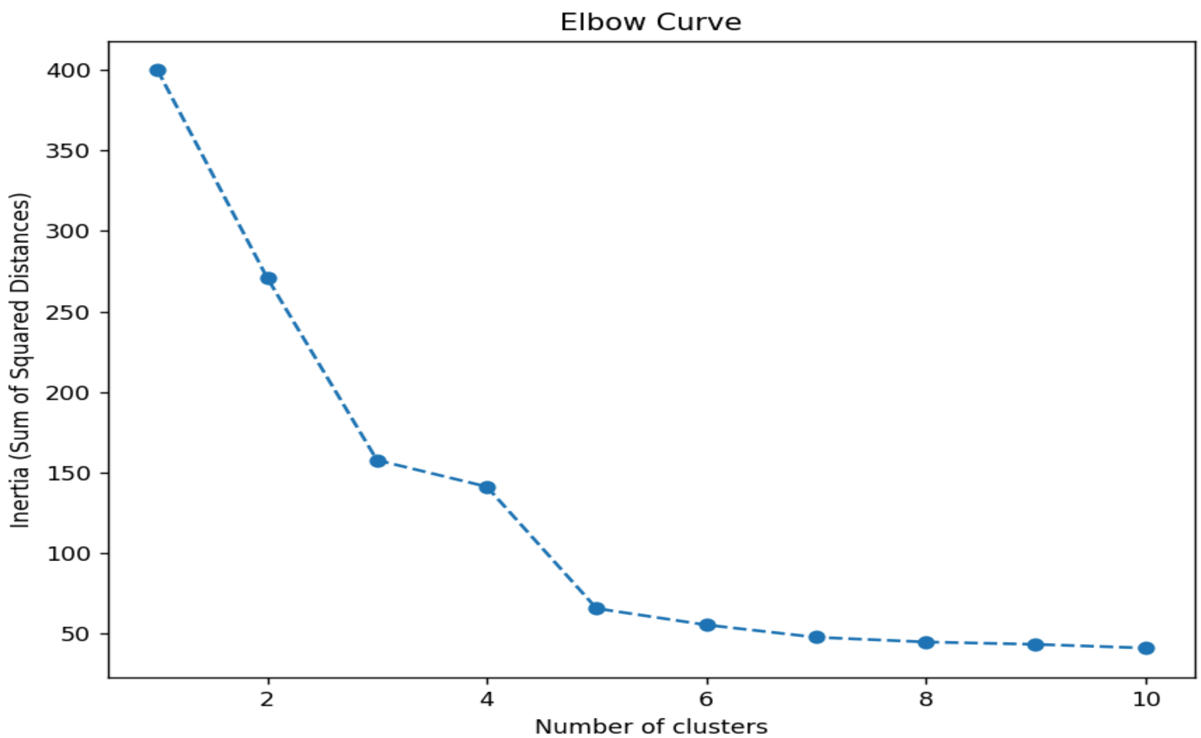
```python
    def assign_clusters(self, X):
        # Assign each point to the nearest centroid
        labels = []
        for point in X:
            distances = [euclidean_distance(point, centroid) for centroid in self.centroids]
            # np.argmin(distances) returns the index of the minimum value in the array
            labels.append(np.argmin(distances))
        return np.array(labels)

    def calculate_centroids(self, X):
        # Compute the centroids as the mean of the points assigned to each cluster
        centroids = np.zeros((self.n_clusters, X.shape[1]))
        for idx in range(self.n_clusters):
            # Get all points that belong to cluster idx
            points = X[self.labels == idx]
            centroids[idx] = np.mean(points, axis=0) if len(points) > 0 else self.centroids[idx]
        return centroids

    def predict(self, X):
        return self.assign_clusters(X)

# Step 2: Plot the elbow curve to find optimal number of clusters
inertia = []
for k in range(1, 11):
    kmeans = KMeansScratch(n_clusters=k)
    kmeans.fit(X_scaled)
```

```python
    # Inertia is the sum of squared distances to the nearest centroid
    inertia_val = np.sum([euclidean_distance(X_scaled[i],
        kmeans.centroids[kmeans.labels[i]])**2 for i in range(X_scaled.shape[0])])
    inertia.append(inertia_val)

# Plot the elbow curve
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.title('Elbow Curve')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.show()

# Step 3: Applying K-Means with the optimal number of clusters (e.g., 5 from elbow curve)
kmeans_scratch = KMeansScratch(n_clusters=5)
kmeans_scratch.fit(X_scaled)

# Step 4: Visualizing the clusters with centroids
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans_scratch.labels, cmap='viridis', s=100)
plt.scatter(kmeans_scratch.centroids[:, 0], kmeans_scratch.centroids[:, 1],
            color='red', marker='*', s=200, label='Centroids')
plt.title('K-Means Clustering with Centroids (Scratch)')
plt.xlabel('Annual Income (Standardized)')
plt.ylabel('Spending Score (Standardized)')
plt.legend()
plt.show()
```

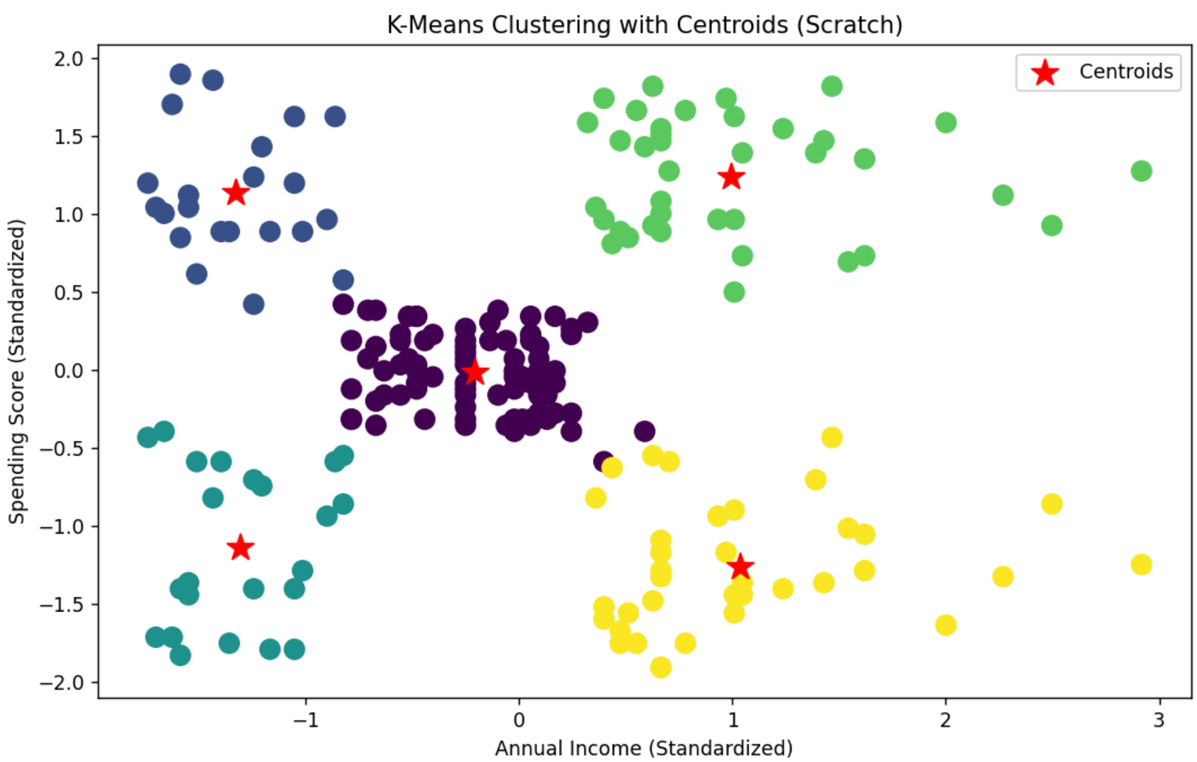
```
# Step 5: Confusion matrix heatmap (assuming cluster predictions vs true labels)
# Here, since we don't have true labels, we simulate a confusion matrix using the predicted labels
y_true = kmeans_scratch.labels  # Predicted labels (in real scenario, compare with actual labels)
y_pred = kmeans_scratch.predict(X_scaled)

conf_matrix = confusion_matrix(y_true, y_pred)

# Plotting the confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix Heatmap')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()

# Output cluster centers
print("Cluster centers (standardized):")
print(kmeans_scratch.centroids)
```

Output -

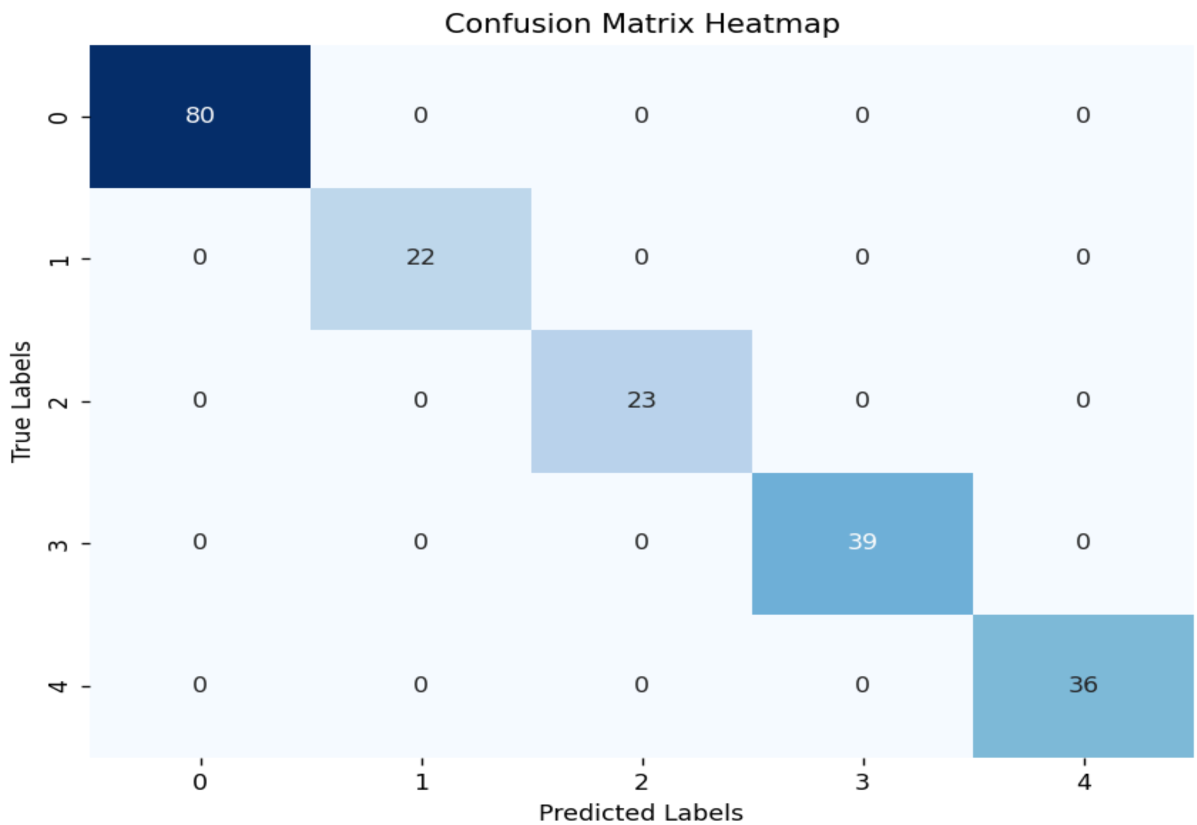




Elbow Curve
Inertia (Sum of Squared Distances)
400
350
300
250
200
150
100
50
2
4
6
8
10
Number of clusters



K-Means Clustering with Centroids (Scratch)
Centroids
Spending Score (Standardized)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1
0
1
2
3
Annual Income (Standardized)



Conclusion -

- The inbuilt K-Means (e.g., Scikit-learn) is more reliable and accurate because it has optimized algorithms and edge-case handling, ensuring better predictions, especially with large dataset.
- So inbuilt has better prediction results.