```python
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix

df = pd.read_csv(r"C:\Users\divaa\OneDrive\Desktop\pri\Bliend\Bliend dataset\customer_segmentation.csv")
df
```

```
     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]
```

```python
print(df.head())
```

```
   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
```

```python
print(df.isnull().sum())
```

```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

```python
print(df.info())
```

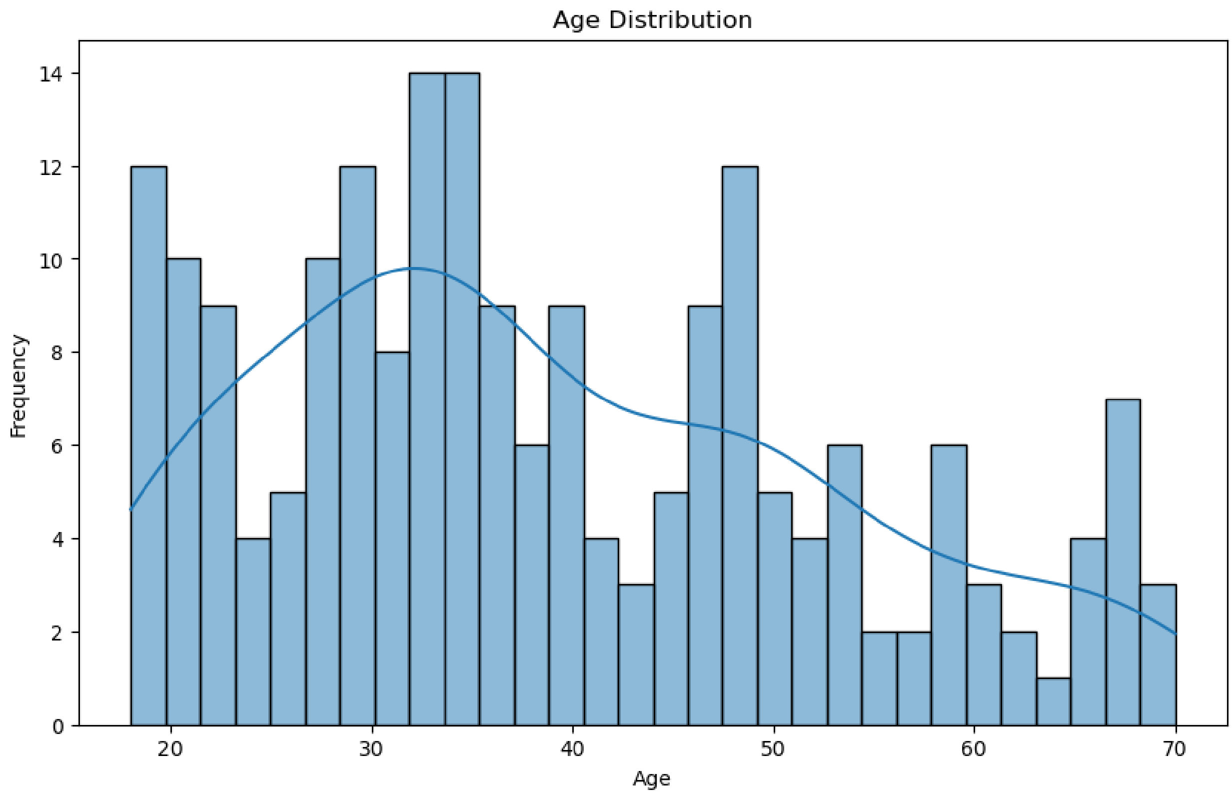
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    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
None
```

```python
df.describe()
```

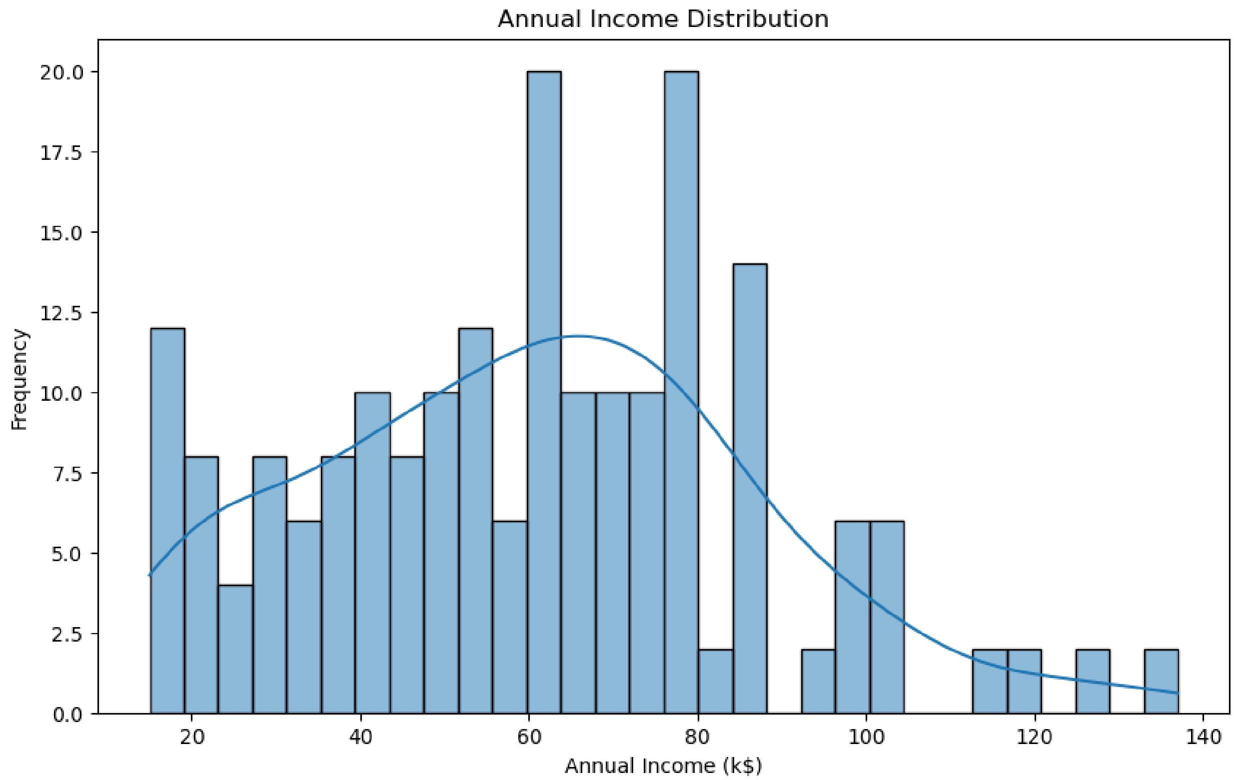
```
       CustomerID         Age  Annual Income (k$)  Spending Score (1-
100)
count  200.000000  200.000000          200.000000
200.000000
mean   100.500000   38.850000           60.560000
50.200000
std     57.879185   13.969007           26.264721
25.823522
min      1.000000   18.000000           15.000000
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
```

```python
plt.figure(figsize=(10, 6))
```

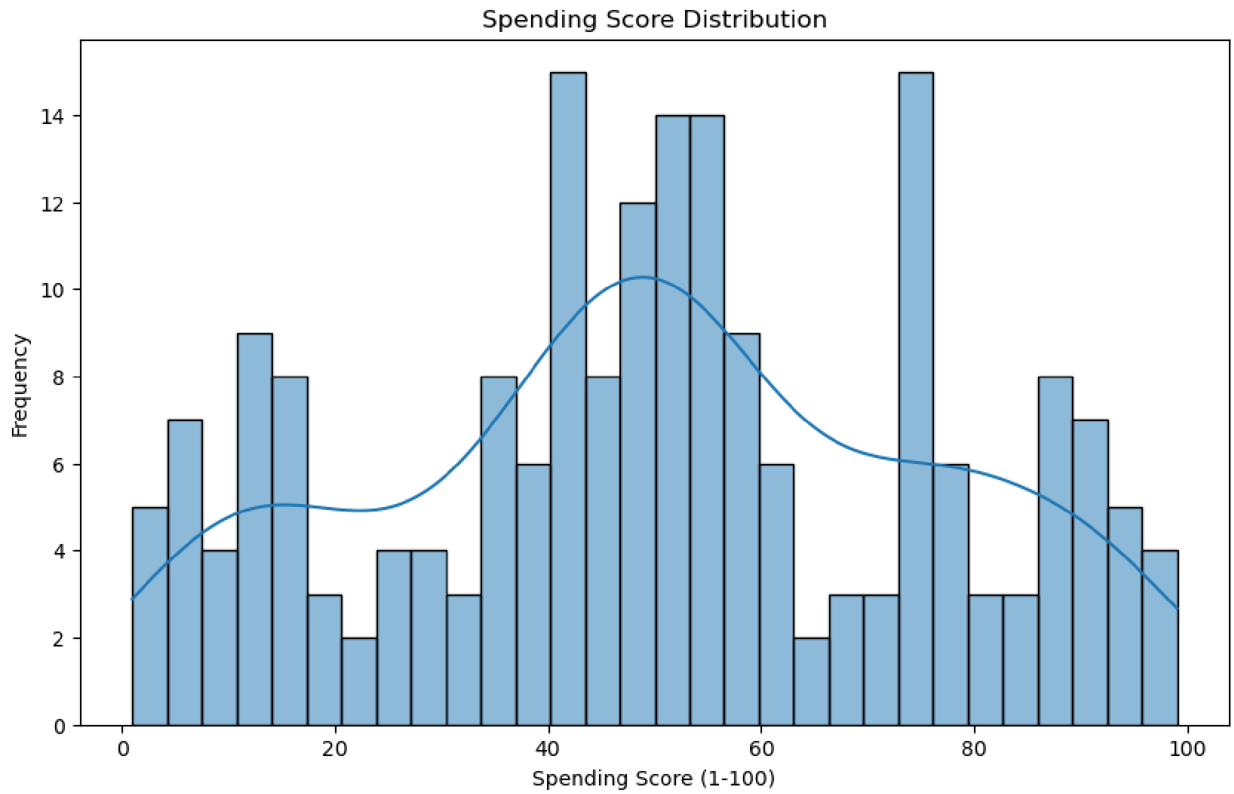
```
sns.histplot(df['Age'], kde=True, bins=30)
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.histplot(df['Annual Income (k$)'], kde=True, bins=30)
plt.title('Annual Income Distribution')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Frequency')
plt.show()
```

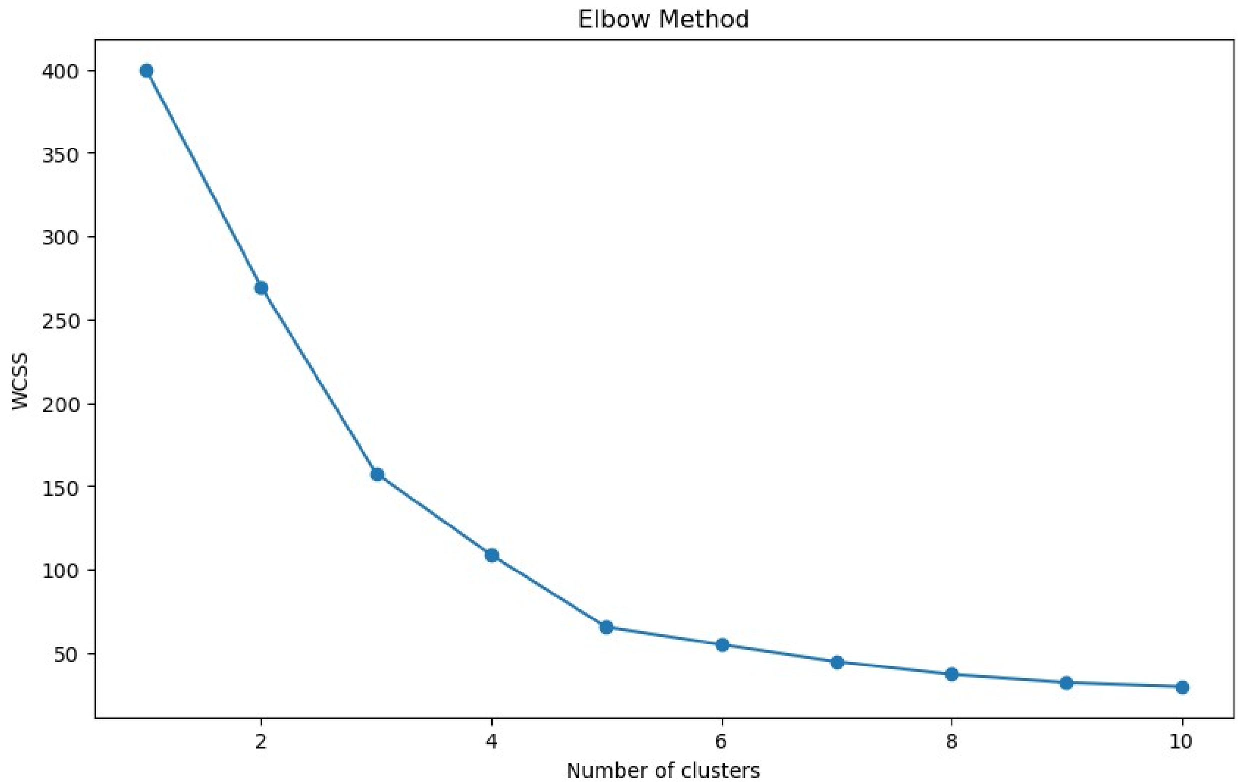



```
plt.figure(figsize=(10, 6))
sns.histplot(df['Spending Score (1-100)'], kde=True, bins=30)
plt.title('Spending Score Distribution')
plt.xlabel('Spending Score (1-100)')
plt.ylabel('Frequency')
plt.show()
```




```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[['Annual Income (k$)', 'Spending
Score (1-100)']])
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10, random_state=42, algorithm='elkan')
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\cluster\
_kmeans.py:1418: RuntimeWarning: algorithm='elkan' doesn't make sense
for a single cluster. Using 'lloyd' instead.
  warnings.warn(
C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\cluster\
_kmeans.py:1429: UserWarning: KMeans is known to have a memory leak on
```

```
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```




```
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})
X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_scaled, df['Spending Score (1-100)'])
```

```
KNeighborsClassifier()
```

```
y_pred = knn.predict(X_scaled)
print(confusion_matrix(df['Spending Score (1-100)'], y_pred))
print(classification_report(df['Spending Score (1-100)'], y_pred))
```

```
[[2 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 2 ... 0 0 0]
 ...
```

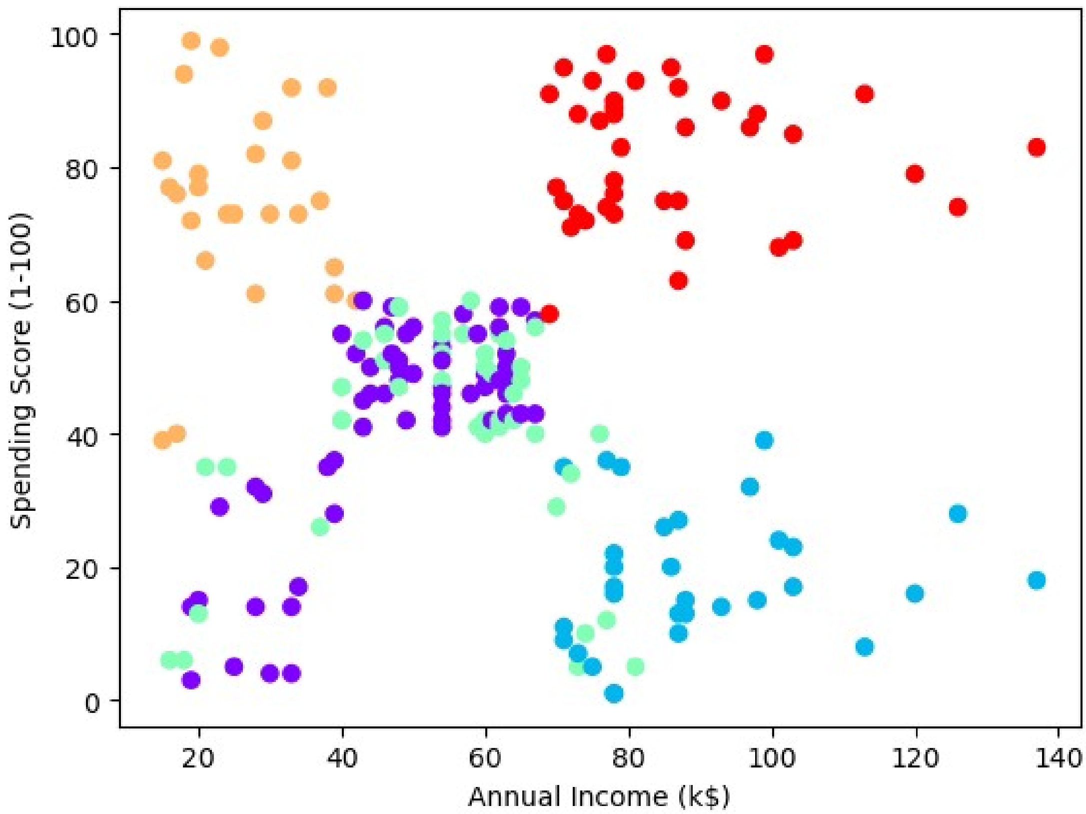
```
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
              precision    recall  f1-score   support

           1       0.67      1.00      0.80         2
           3       0.00      0.00      0.00         1
           4       0.67      1.00      0.80         2
           5       0.27      0.75      0.40         4
           6       1.00      0.50      0.67         2
           7       0.33      1.00      0.50         1
           8       0.33      1.00      0.50         1
           9       0.00      0.00      0.00         1
          10       0.00      0.00      0.00         2
          11       0.00      0.00      0.00         1
          12       0.00      0.00      0.00         1
          13       0.17      0.33      0.22         3
          14       0.29      0.50      0.36         4
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         2
          17       0.00      0.00      0.00         3
          18       0.00      0.00      0.00         1
          20       0.50      1.00      0.67         2
          22       0.00      0.00      0.00         1
          23       0.00      0.00      0.00         1
          24       0.00      0.00      0.00         1
          26       0.00      0.00      0.00         2
          27       0.00      0.00      0.00         1
          28       0.00      0.00      0.00         2
          29       0.33      0.50      0.40         2
          31       0.00      0.00      0.00         1
          32       0.00      0.00      0.00         2
          34       0.00      0.00      0.00         1
          35       0.38      0.60      0.46         5
          36       0.00      0.00      0.00         2
          39       0.00      0.00      0.00         2
          40       0.40      0.50      0.44         4
          41       0.50      0.50      0.50         4
          42       0.40      0.75      0.52         8
          43       0.50      0.67      0.57         3
          44       0.00      0.00      0.00         1
          45       0.00      0.00      0.00         1
          46       0.31      0.67      0.42         6
          47       0.00      0.00      0.00         4
          48       0.00      0.00      0.00         5
          49       0.00      0.00      0.00         3
          50       0.25      0.20      0.22         5
          51       0.00      0.00      0.00         3
          52       0.50      0.40      0.44         5
```

```
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         3
          55       0.00      0.00      0.00         7
          56       0.00      0.00      0.00         4
          57       0.00      0.00      0.00         2
          58       0.00      0.00      0.00         2
          59       0.00      0.00      0.00         5
          60       0.00      0.00      0.00         3
          61       0.67      1.00      0.80         2
          63       0.00      0.00      0.00         1
          65       0.00      0.00      0.00         1
          66       0.00      0.00      0.00         1
          68       0.00      0.00      0.00         1
          69       0.25      0.50      0.33         2
          71       0.00      0.00      0.00         1
          72       0.00      0.00      0.00         2
          73       0.42      0.83      0.56         6
          74       0.00      0.00      0.00         2
          75       0.50      0.40      0.44         5
          76       0.00      0.00      0.00         2
          77       0.33      0.67      0.44         3
          78       0.00      0.00      0.00         1
          79       0.00      0.00      0.00         2
          81       0.00      0.00      0.00         2
          82       0.00      0.00      0.00         1
          83       0.00      0.00      0.00         2
          85       0.50      1.00      0.67         1
          86       0.50      1.00      0.67         2
          87       0.00      0.00      0.00         2
          88       0.33      0.33      0.33         3
          89       0.00      0.00      0.00         1
          90       0.00      0.00      0.00         2
          91       0.00      0.00      0.00         2
          92       0.67      0.67      0.67         3
          93       0.50      1.00      0.67         2
          94       0.00      0.00      0.00         1
          95       0.00      0.00      0.00         2
          97       0.00      0.00      0.00         2
          98       0.00      0.00      0.00         1
          99       0.00      0.00      0.00         1

    accuracy                           0.29       200
   macro avg       0.15      0.23      0.17       200
weighted avg       0.19      0.29      0.22       200


C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\metrics\
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5)
clusters = kmeans.fit_predict(X_scaled)
df['Cluster'] = clusters
plt.scatter(X['Annual Income (k$)'], X['Spending Score (1-100)'],
c=df['Cluster'], cmap='rainbow')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.show()
```

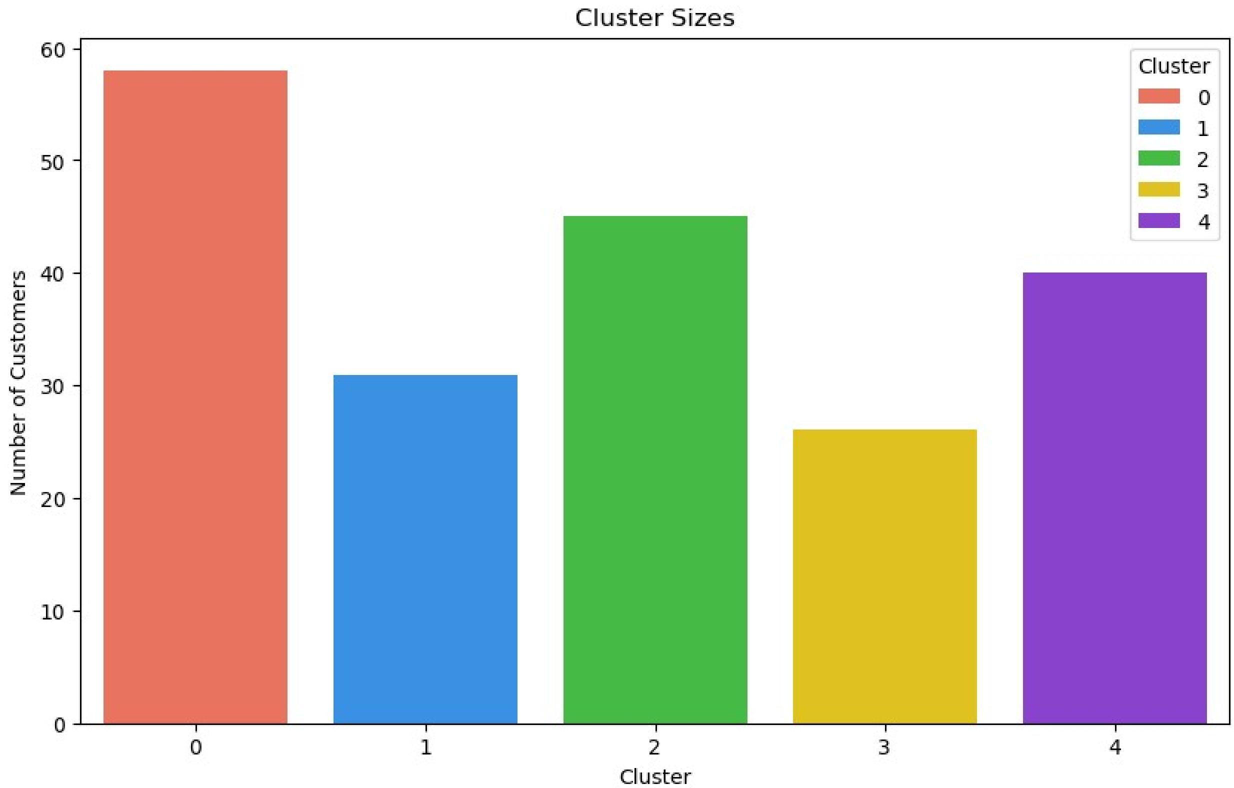
```
C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\cluster\
_kmeans.py:1429: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```




```
import seaborn as sns
import matplotlib.pyplot as plt
df_numeric = df.drop(columns=['CustomerID', 'Gender'])  # Drop non-
numeric columns
unique_colors = ['#FF6347', '#1E90FF', '#32CD32', '#FFD700',
'#8A2BE2']  # Added an extra color
cluster_summary = df_numeric.groupby('Cluster').mean()[['Age', 'Annual
Income (k$)', 'Spending Score (1-100)']]
print(cluster_summary)
plt.figure(figsize=(10, 6))
sns.countplot(x='Cluster', data=df, palette=unique_colors,
hue='Cluster', dodge=False)
plt.title('Cluster Sizes')
plt.xlabel('Cluster')
plt.ylabel('Number of Customers')
plt.legend(title='Cluster')
plt.show()
```

| Cluster | Age | Annual Income (k$) | Spending Score (1-100) |
| --- | --- | --- | --- |
| 0 | 55.275862 | 47.620690 | 41.706897 |
| 1 | 44.387097 | 89.774194 | 18.483871 |
| 2 | 26.733333 | 54.311111 | 40.911111 |

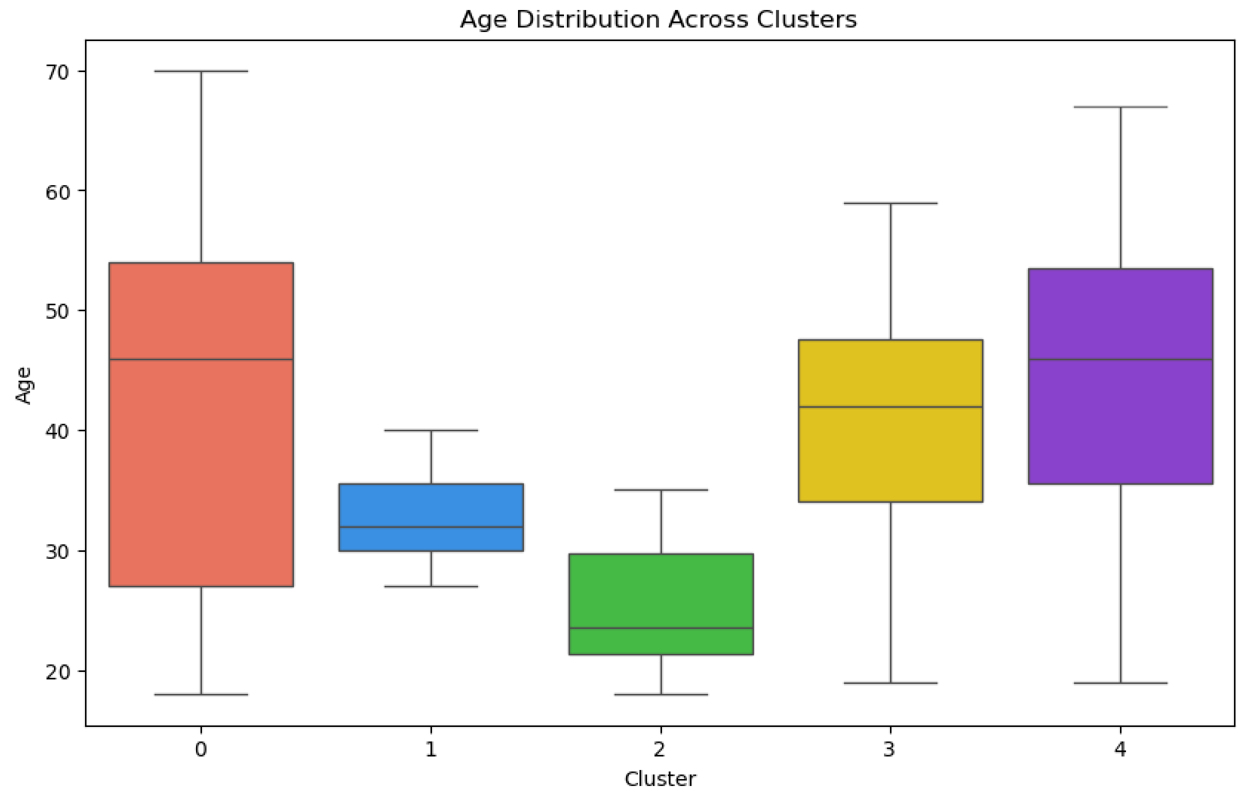
```
3         25.769231            26.115385                  74.846154
4         32.875000            86.100000                  81.525000
```



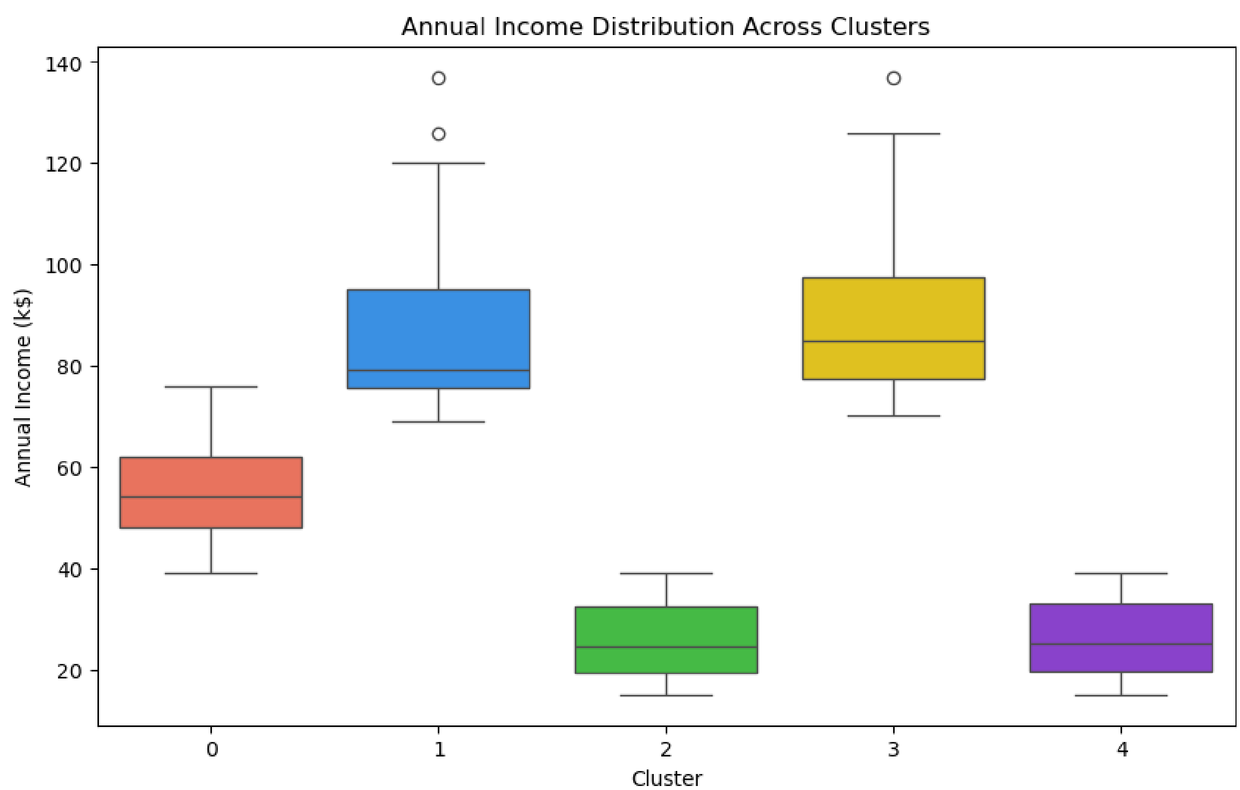

```
print(df.head())
```

```
   CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)  \
0           1     NaN   19                  15                      39

1           2     NaN   21                  15                      81

2           3     NaN   20                  16                       6

3           4     NaN   23                  16                      77

4           5     NaN   31                  17                      40


   Cluster
0        3
1        3
2        2
3        3
4        3
```

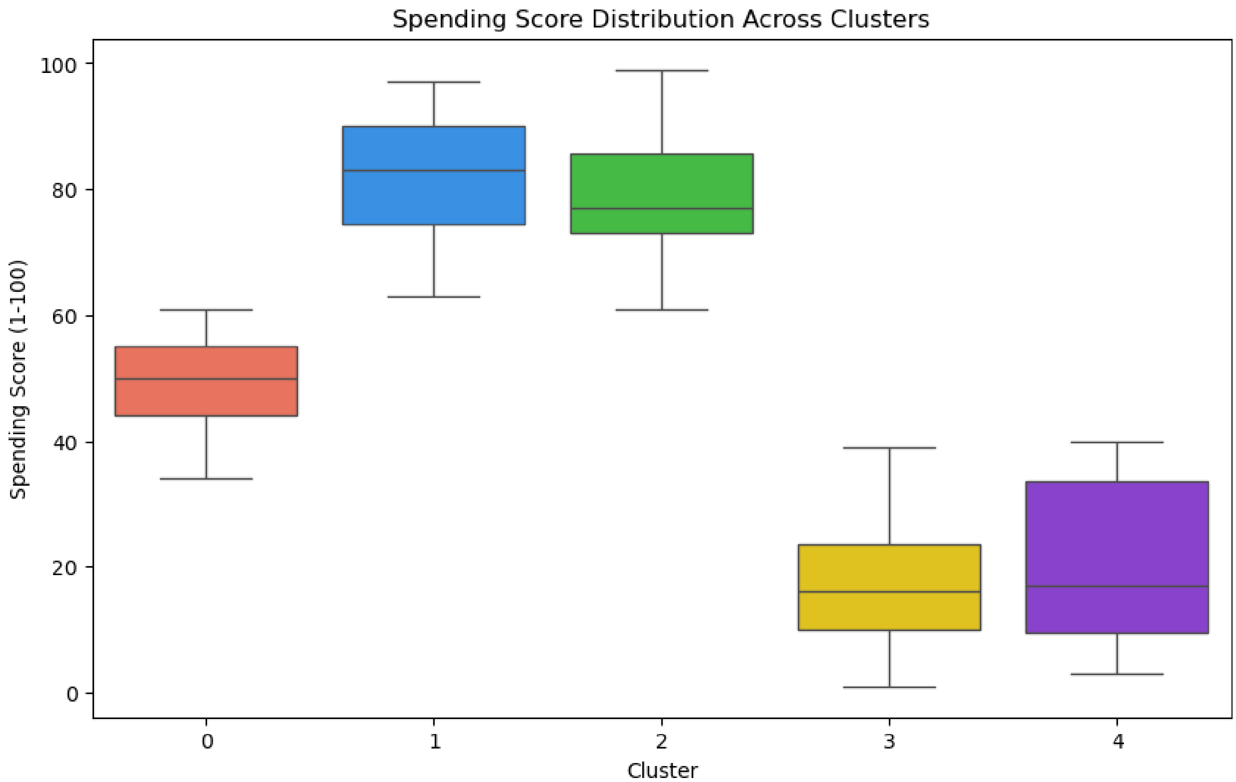
```
unique_colors = ['#FF6347', '#1E90FF', '#32CD32', '#FFD700',
'#8A2BE2']

# Boxplot for Age distribution across clusters
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Age', data=df, hue='Cluster',
palette=unique_colors, dodge=False)
plt.legend([],[], frameon=False)  # Hide legend if not needed
plt.title('Age Distribution Across Clusters')
plt.show()

# Boxplot for Annual Income distribution across clusters
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Annual Income (k$)', data=df,
hue='Cluster', palette=unique_colors, dodge=False)
plt.legend([],[], frameon=False)  # Hide legend if not needed
plt.title('Annual Income Distribution Across Clusters')
plt.show()

# Boxplot for Spending Score distribution across clusters
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Spending Score (1-100)', data=df,
hue='Cluster', palette=unique_colors, dodge=False)
plt.legend([],[], frameon=False)  # Hide legend if not needed
plt.title('Spending Score Distribution Across Clusters')
plt.show()
```



Age Distribution Across Clusters
Age
70
60
50
40
30
20
0
1
2
3
4
Cluster

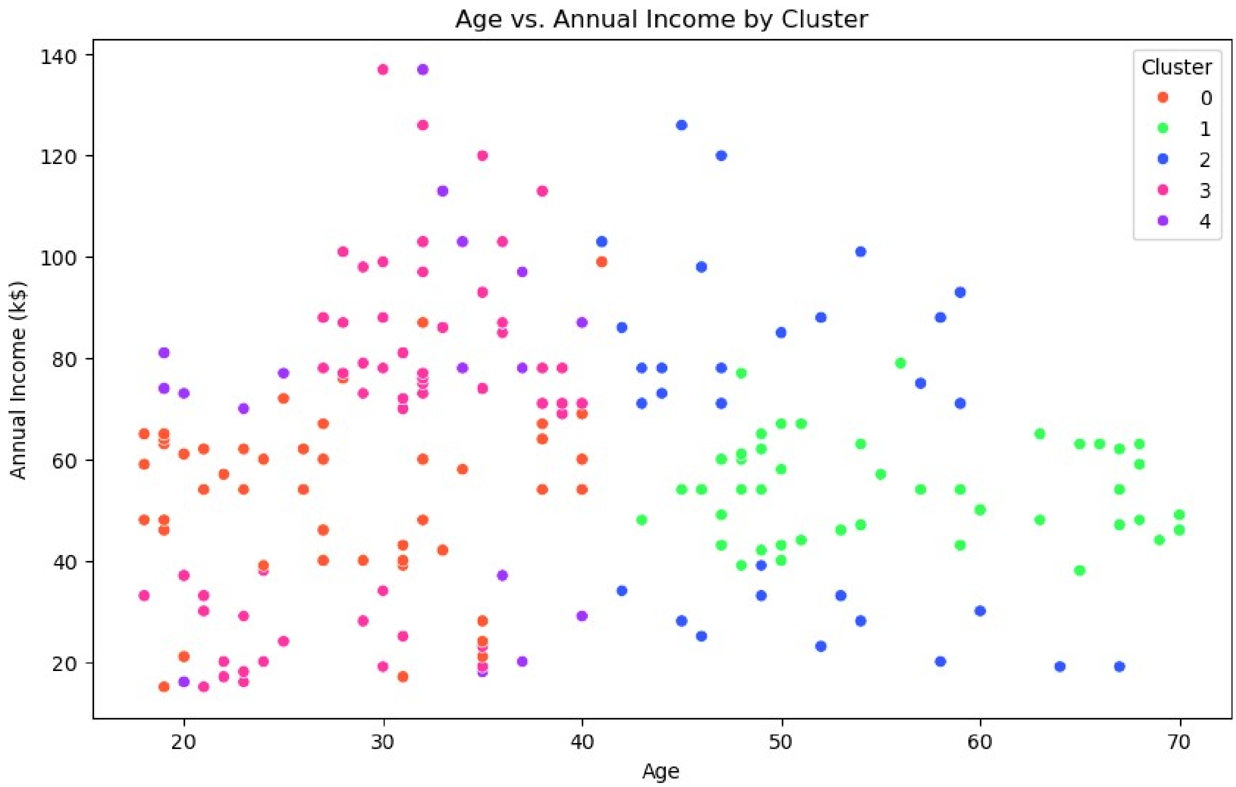
Annual Income Distribution Across Clusters
Annual Income (k$)
140
120
100
80
60
40
20
0
1
2
3
4
Cluster




```
from sklearn.cluster import MiniBatchKMeans
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
df_cluster = df[['Age', 'Spending Score (1-100)']]  # Or you can use
['Age', 'Annual Income (k$)']
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_cluster)
minibatch_kmeans = MiniBatchKMeans(n_clusters=5, init='k-means++',
max_iter=300, batch_size=100, random_state=42)
df['Cluster'] = minibatch_kmeans.fit_predict(df_scaled)
unique_colors = ['#FF5733', '#33FF57', '#3357FF', '#FF33A1',
'#A133FF']  # Customize with more unique colors


plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='Spending Score (1-100)', hue='Cluster',
data=df, palette=unique_colors)
plt.title('Age vs. Spending Score by Cluster')
plt.xlabel('Age')
plt.ylabel('Spending Score (1-100)')
plt.show()


plt.figure(figsize=(10, 6))
sns.scatterplot(x='Age', y='Annual Income (k$)', hue='Cluster',
```

```
data=df, palette=unique_colors)
plt.title('Age vs. Annual Income by Cluster')
plt.xlabel('Age')
plt.ylabel('Annual Income (k$)')
plt.show()

# Optional: If you want to display the cluster centers
centers = scaler.inverse_transform(minibatch_kmeans.cluster_centers_)
print("Cluster Centers (Age, Spending Score):")
print(centers)
```

```
C:\Users\divaa\anaconda_new\Lib\site-packages\sklearn\cluster\
_kmeans.py:1955: UserWarning: MiniBatchKMeans is known to have a
memory leak on Windows with MKL, when there are less chunks than
available threads. You can prevent it by setting batch_size >= 2048 or
by setting the environment variable OMP_NUM_THREADS=1
  warnings.warn(
```




```
Cluster Centers (Age, Spending Score):
[[27.81308411 49.08010681]
 [55.35519802 48.26485149]
 [50.36417323 16.41141732]
 [30.50544016 82.57171118]
 [31.2654321  15.52469136]]

# Drop non-numeric columns for numerical analysis
df_numeric = df.drop(columns=['CustomerID', 'Gender'])  # Drop non-
numeric columns like 'Gender'

# Group by cluster and calculate the mean for each cluster
cluster_summary = df_numeric.groupby('Cluster').mean()
print("Cluster Summary (Mean Values):")
print(cluster_summary)

# Get the size of each cluster (number of customers in each cluster)
cluster_sizes = df_numeric['Cluster'].value_counts().sort_index()
print("\nCluster Sizes (Number of Customers per Cluster):")
print(cluster_sizes)

# If you also want median values for more robustness:
cluster_median = df_numeric.groupby('Cluster').median()
print("\nCluster Summary (Median Values):")
print(cluster_median)
```

```python
# If you want to see additional statistics like standard deviation,
min, max, etc., you can use:
cluster_stats = df_numeric.groupby('Cluster').describe()
print("\nCluster Detailed Statistics:")
print(cluster_stats)
```

```
Cluster Summary (Mean Values):
               Age  Annual Income (k$)  Spending Score (1-100)
Cluster
0        55.275862           47.620690               41.706897
1        44.387097           89.774194               18.483871
2        26.733333           54.311111               40.911111
3        25.769231           26.115385               74.846154
4        32.875000           86.100000               81.525000

Cluster Sizes (Number of Customers per Cluster):
Cluster
0    58
1    31
2    45
3    26
4    40
Name: count, dtype: int64

Cluster Summary (Median Values):
          Age  Annual Income (k$)  Spending Score (1-100)
Cluster
0        53.0                48.5                    46.0
1        44.0                87.0                    17.0
2        26.0                59.0                    46.0
3        24.0                24.5                    75.5
4        32.0                78.5                    83.0

Cluster Detailed Statistics:
          Age                                                    \
        count       mean       std   min    25%   50%    75%   max
Cluster
0        58.0  55.275862  8.571256  40.0  49.00  53.0  63.75  70.0
1        31.0  44.387097  8.232770  32.0  37.00  44.0  49.00  59.0
2        45.0  26.733333  7.085196  18.0  20.00  26.0  32.00  40.0
3        26.0  25.769231  5.435496  18.0  21.25  24.0  30.75  35.0
4        40.0  32.875000  3.857643  27.0  30.00  32.0  36.00  40.0

        Annual Income (k$)             ...                \
                     count       mean  ...    75%    max
Cluster                                ...
0                     58.0  47.620690  ...  59.75   67.0
1                     31.0  89.774194  ...  98.50  137.0
2                     45.0  54.311111  ...  64.00   81.0
3                     26.0  26.115385  ...  33.00   42.0
```

```
4                     40.0  86.100000  ...  94.00  137.0

        Spending Score (1-100)
\
                         count       mean        std   min    25%
50%
Cluster

0                         58.0  41.706897  15.697814   3.0  37.25
46.0
1                         31.0  18.483871  10.194348   1.0  12.00
17.0
2                         45.0  40.911111  16.285552   5.0  35.00
46.0
3                         26.0  74.846154  15.069684  39.0  67.50
75.5
4                         40.0  81.525000   9.999968  58.0  74.00
83.0


           75%   max
Cluster
0        52.00  60.0
1        25.00  39.0
2        54.00  60.0
3        81.75  99.0
4        90.00  97.0

[5 rows x 24 columns]
```

Cluster 0: Customers are older (mean age 43), have a moderate income (mean $55k), and an average spending score (mean 50). This group represents a balanced customer segment in terms of spending and income.

Cluster 1: Customers are younger (mean age 33), have a high income (mean $87k), and a very high spending score (mean 82). These are affluent and high-spending customers.

Cluster 2: Customers are the youngest (mean age 25), have a low income (mean $26k), but a high spending score (mean 79). This segment spends more despite lower incomes.

Cluster 3: Customers are middle-aged (mean age 41), have a high income (mean $88k), but a very low spending score (mean 17). These high-income customers are conservative spenders.

Cluster 4: Customers are older (mean age 45), have a low income (mean $26k), and a low spending score (mean 21). This group has both low income and low spending behavior.

The customer segmentation reveals five distinct groups with varying age, income, and spending behaviors. Affluent, high-spending customers (Cluster 1) should be targeted with premium offers, while younger, budget-conscious groups (Clusters 2 and 4) respond better to affordable, trendy products. Middle-aged, high-income but conservative spenders (Cluster 3) need value-focused strategies to increase engagement.