your_notebook

July 25, 2025

1 Customer Segmentation using Clustering Method

Field	Value
Programmer	HTET AUNG LYNN
Start date	18 July 2025
End date	25 July 2025
Purpose	Choosing best model for customer segmentation using Machine Learning CLUSTERING

1.0.1 Import modules

[1]:
```
import os
from urllib import request
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
```

[2]:
```
sns.set_context('notebook')
sns.set()
```

1.0.2 Defined functions

[3]:
```
def plot_centroids(centroids, label):
    plt.scatter(centroids[:,0], centroids[:,1], marker='*',
                s=100, color='red', label=label, lw=2)
```

[4]:
```
def plot_image(model, y_pred, algname, centroids=False, means=False):
    # Color map
    cmap = plt.get_cmap('tab20')

    # If DBSCAN, separate anomalies. Unless normal.
    if -1 in y_pred:
        clusters = np.unique(y_pred[y_pred != -1])
```

```
        anomalies = X_scaled[y_pred == -1]
    else:
        clusters = np.unique(y_pred)

    labels = [f"CL-{j+1}" for j in np.unique(clusters)]
    ncol = len(labels) + 1

    plt.figure(figsize=(8,5))
    for i in clusters:
        plt.scatter(X_scaled[y_pred == i, 0],
                    X_scaled[y_pred == i, 1],
                    c=[cmap(i)],
                    label=labels[i],
                    zorder = 1)
    if centroids:
        val = model.cluster_centers_
        plot_centroids(val, "Centroids")
    if means:
        val = model.means_
        plot_centroids(val, "Means")

    if -1 in y_pred:
        plt.scatter(anomalies[:,0], anomalies[:,1], marker='x',
                    color='red', label='Anomalies', s=100,
                    zorder = 2)

    plt.xlabel("Annual Income (k$)", weight='bold')
    plt.ylabel("Spending Score (1-100)", weight='bold')
    plt.title(f"Customer Segementation ({algname})", size=20, weight='bold')
    plt.legend(bbox_to_anchor=(0.45, -0.2), loc='center', ncol=ncol, fontsize=9)
    plt.show()
```

1.0.3 Download and load data

[5]:
```
URL = "https://drive.google.com/uc?
 ↪export=download&id=1u0eqgaojSyGCudnCKQJOxF6_a5Bg-86e"
ROOT_DIR = "module_7"
DATA = "data"
FILENAME = "mall_customers.csv"
PATH = os.path.join(ROOT_DIR, DATA + "/")
os.makedirs(PATH, exist_ok=True)
request.urlretrieve(URL,  PATH + FILENAME)
```

[5]:
```
('module_7/data/mall_customers.csv',
 <http.client.HTTPMessage at 0x790d77c80b50>)
```

[6]:
```
customer_data = pd.read_csv(PATH+FILENAME)
```

1.0.4 Check properties

[7]:
```
customer_data.head(3)
```

[7]:
```
   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
```

[8]:
```
customer_data.shape
```

[8]:
```
(200, 5)
```

[9]:
```
customer_data[customer_data.duplicated()]
```

[9]:
```
Empty DataFrame
Columns: [CustomerID, Gender, Age, Annual Income (k$), Spending Score (1-100)]
Index: []
```

[10]:
```
customer_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
```

[11]:
```
customer_data.isnull().sum()
```

[11]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

[12]:
```
customer_data.drop(columns="CustomerID", inplace=True)
```

[33]:
```
customer_data.describe()
```

[33]:
```
              Age  Annual Income (k$)  Spending Score (1-100)     Cluster
count  200.000000          200.000000              200.000000  200.000000
```

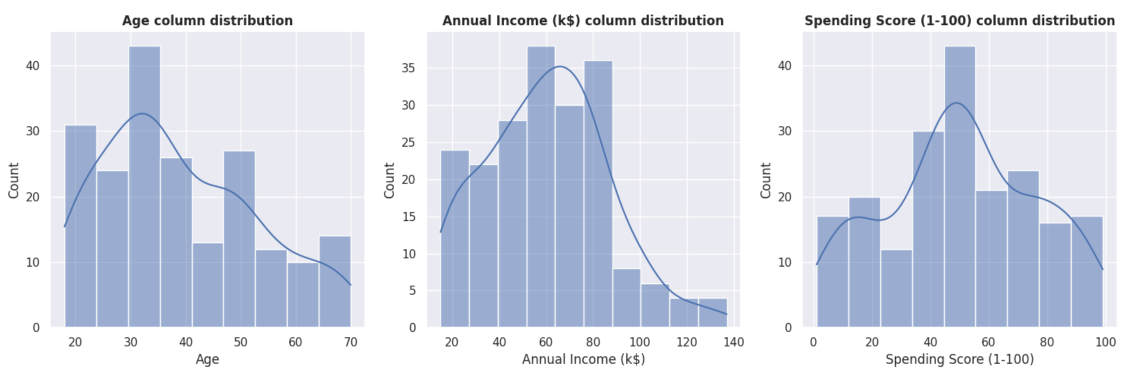
mean	38.850000	60.560000	50.200000	1.350000
std	13.969007	26.264721	25.823522	1.391828
min	18.000000	15.000000	1.000000	0.000000
25%	28.750000	41.500000	34.750000	0.000000
50%	36.000000	61.500000	50.000000	1.000000
75%	49.000000	78.000000	73.000000	2.000000
max	70.000000	137.000000	99.000000	4.000000

[36]:
```
customer_data.groupby("Gender")["Spending Score (1-100)"].agg(['min',␣
↪'max','mean'])
```

[36]:

Gender	min	max	mean
Female	5	99	51.526786
Male	1	97	48.511364

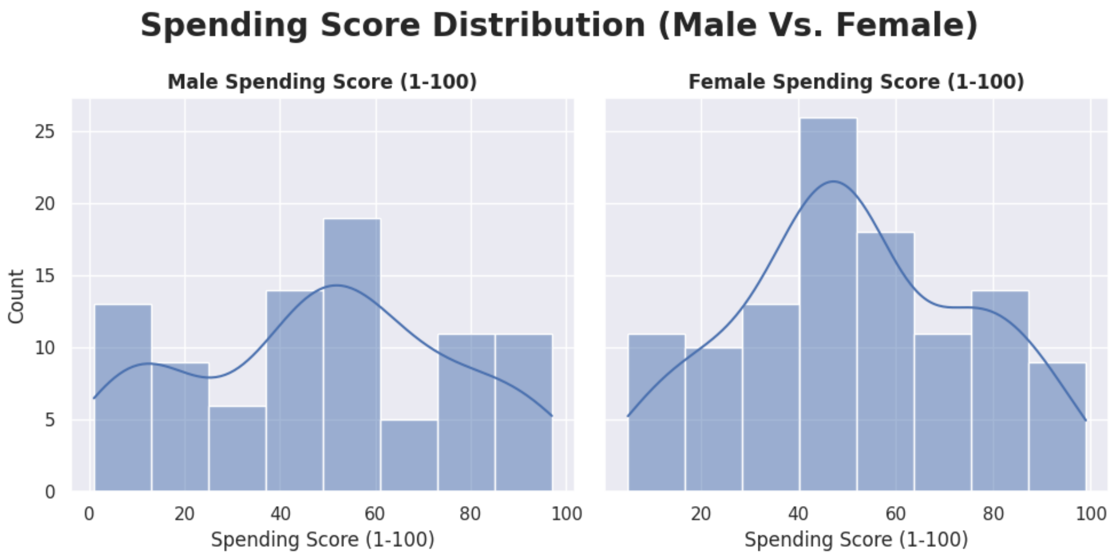
[13]:
```
cols = [z for z in customer_data.columns if z != 'Gender']
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
i = 0
for _, col in enumerate(cols):
    sns.histplot(customer_data[col], kde=True, ax=ax[_])
    ax[_].set_title(f"{col} column distribution", weight='bold')
plt.tight_layout()
plt.show()
```



[14]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5), sharey=True)
for _, g in enumerate(['Male', 'Female']):

    sns.histplot(data=customer_data[customer_data["Gender"] == g],
                 x="Spending Score (1-100)", kde=True,
                 ax=ax[_])
    ax[_].set_title(f"{g} Spending Score (1-100)", weight='bold')
```

```
plt.suptitle("Spending Score Distribution (Male Vs. Female)", weight='bold',␣
    ↪size=20)
plt.tight_layout()
plt.show()
```



[15]:
```
# Target columns: "Annual Income (k$)" and "Spending Score (1-100)"
X = customer_data.iloc[:, 2:].values
X_scaled = StandardScaler().fit_transform(X)
```

1.0.5 K-Means modelling

[16]:
```
# Plot original data
plt.scatter(X[:, 0], X[:, 1])
plt.title("Original data", size=20, weight='bold')
plt.xlabel("Annual Income (k$)", weight='bold')
plt.ylabel("Spending Score (1-100)", weight='bold')
plt.show()
```

Choose number of clusters

WCSS = Within Cluster Sum of Squares

image

[17]:
```
# Find WCSS values for different number of clusters
wcss = []

# Check by elbow graph
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

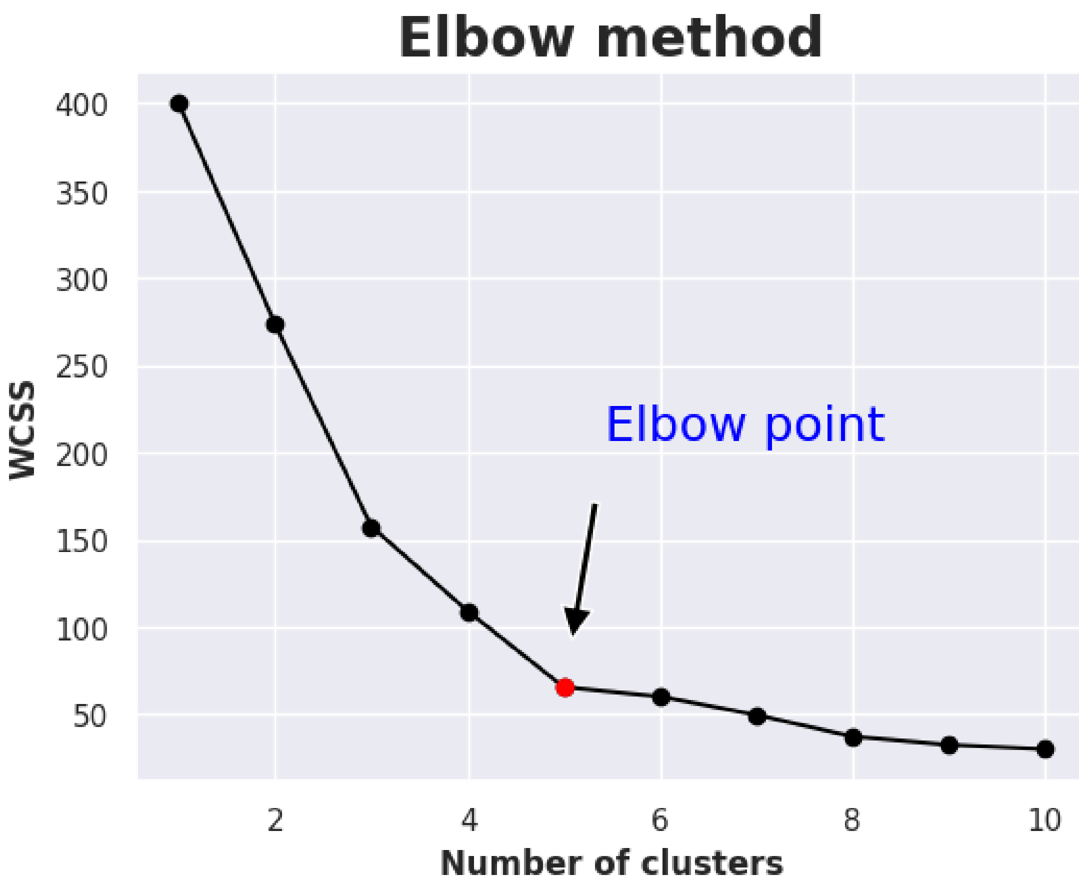
[18]:
```
wcss
```

[18]:
```
[399.99999999999994,
 273.66888662642003,
 157.70400815035939,
```

```
 109.22822707921345,
 65.56840815571681,
 60.132874871934206,
 49.668244837367965,
 37.31912287833882,
 32.495081199100916,
 30.05932269404222]
```

[19]:
```
# Plot elbow graph
plt.plot(range(1, 11), wcss, marker='o', color='black', zorder=1)
plt.scatter(5, wcss[4], color='red', zorder=2)
plt.annotate("Elbow point",
             xy = (5, wcss[4]),
             xytext = (0.5, 0.5),
             color = 'blue',
             fontsize = 18,
             textcoords = 'figure fraction',
             arrowprops = dict(facecolor='black', shrink=0.2, width=3)
             )

plt.title("Elbow method", size=20, weight='bold')
plt.xlabel("Number of clusters", weight='bold')
plt.ylabel("WCSS", weight='bold')
plt.show()
```



Optimum number of clusters is 5.

Train k-Means clustering model

[20]:
```
kmeans = KMeans(n_clusters=5, init='k-means++')

# Return labels for each data point based on cluster
y_kmean = kmeans.fit_predict(X_scaled)
print(y_kmean)
```

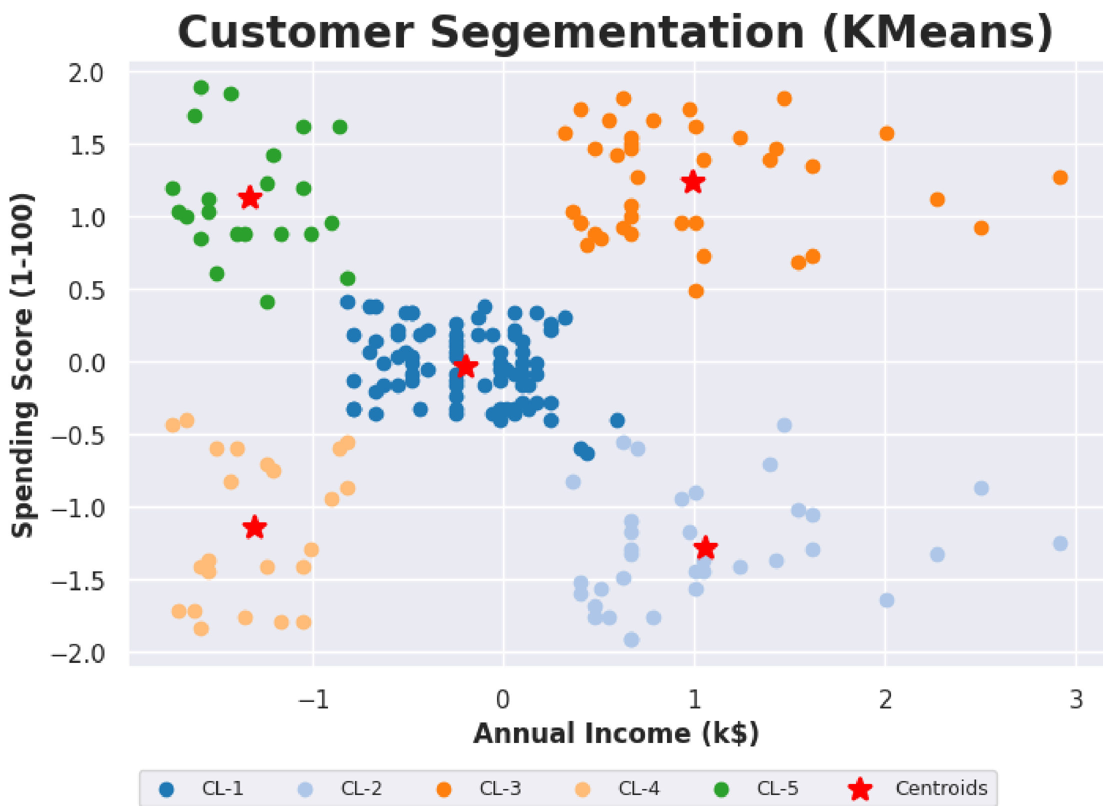
```
[3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
 4 3 4 3 4 3 0 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 0 2 1 2 1 2 0 2 1 2 1 2 1 2 1 2 0 2 1 2 1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

Visualize clusters

[21]:
```
kmeans.labels_.shape, X_scaled.shape
```

[21]: ((200,), (200, 2))

[22]:
```
plot_image(model=kmeans, y_pred=y_kmean, algname='KMeans', centroids=True )
```



[23]:
```
customer_data['Cluster'] = kmeans.labels_
customer_data.groupby('Cluster')[['Annual Income (k$)', 'Spending Score␣
↪(1-100)']].mean()
```

[23]:
```
         Annual Income (k$)  Spending Score (1-100)
Cluster
0                 55.296296               49.518519
1                 88.200000               17.114286
2                 86.538462               82.128205
3                 26.304348               20.913043
4                 25.727273               79.363636
```

[24]:
```
cluster_names = {
    0: "High Income, High Spending",
    1: "Moderate Income and Spending",
    2: "Low Income and Spending",
    3: "Low Income, High Spending",
```

```
    4: "High Income, Low Spending"}

customer_data["Customer_Type"] = customer_data["Cluster"].map(cluster_names)
customer_data.head()
```

[24]:
```
   Gender  Age  Annual Income (k$)  Spending Score (1-100)  Cluster  \
0    Male   19                  15                      39        3
1    Male   21                  15                      81        4
2  Female   20                  16                       6        3
3  Female   23                  16                      77        4
4  Female   31                  17                      40        3

              Customer_Type
0  Low Income, High Spending
1  High Income, Low Spending
2  Low Income, High Spending
3  High Income, Low Spending
4  Low Income, High Spending
```

[31]:
```
customer_data.groupby('Customer_Type')[['Annual Income (k$)', 'Spending Score␣
↪(1-100)']].mean()
```

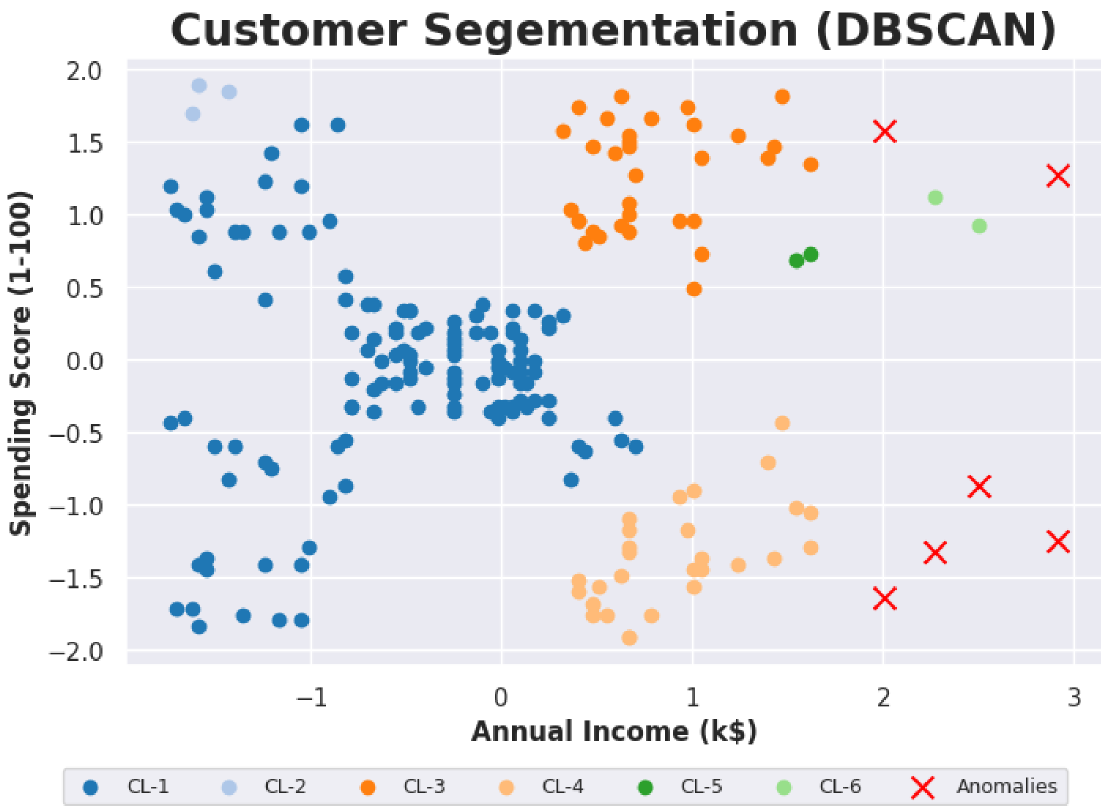
[31]:
```
                              Annual Income (k$)  Spending Score (1-100)
Customer_Type
High Income, High Spending             55.296296               49.518519
High Income, Low Spending              25.727273               79.363636
Low Income and Spending                86.538462               82.128205
Low Income, High Spending              26.304348               20.913043
Moderate Income and Spending           88.200000               17.114286
```

1.0.6 DBSCAN modelling

[25]:
```
dbscan = DBSCAN(min_samples=2, eps=10)
y_dbscan = dbscan.fit_predict(X)
# eps - small distances between instances
# minimum samples instances in its epslon-neighborhood
# -1 considers as anomaly.
np.unique(y_dbscan, return_counts=True)
```

[25]:
```
(array([-1,  0,  1,  2,  3,  4,  5]),
 array([  6, 126,   3,  33,  28,   2,   2]))
```

[26]:
```
plot_image(model=dbscan, y_pred=y_dbscan, algname='DBSCAN')
```



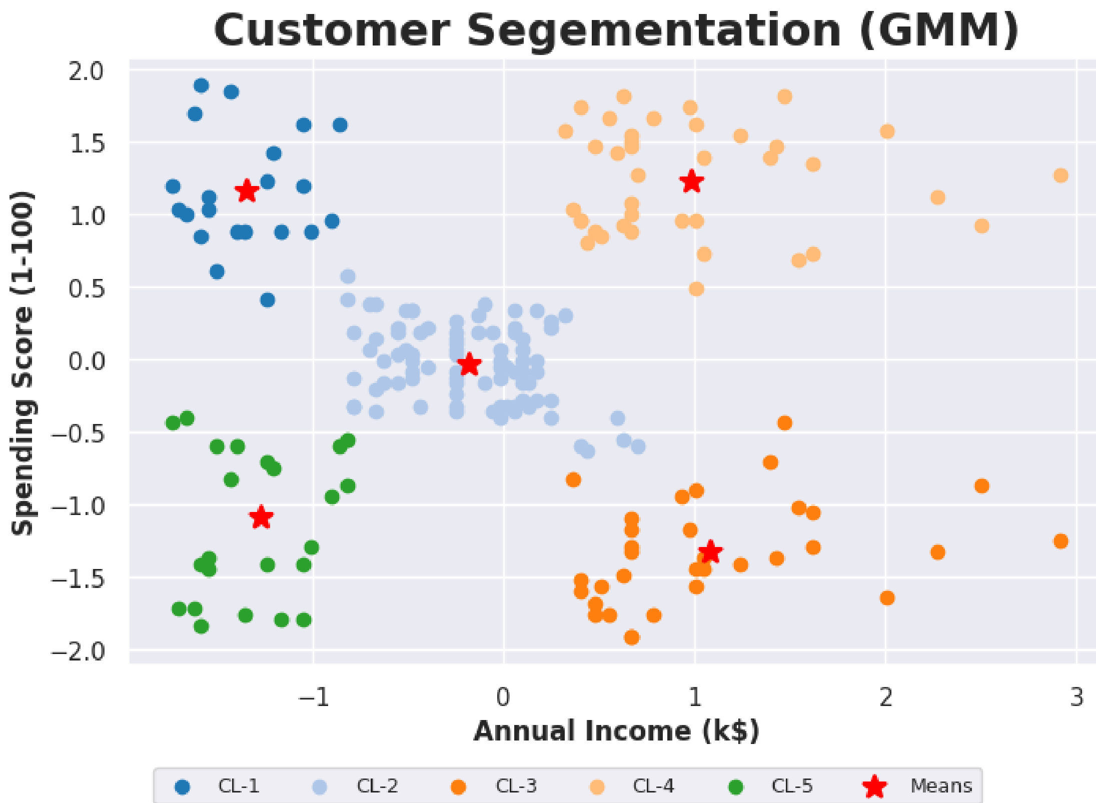
1.0.7 GMM

[27]:
```
from sklearn.mixture import GaussianMixture
```

[28]:
```
gmm = GaussianMixture(n_components=5)
y_gmm = gmm.fit_predict(X_scaled)
y_gmm
```

[28]:
```
array([4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0,
       4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 1,
       4, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 2, 3, 1, 3, 2, 3, 2, 3,
       1, 3, 2, 3, 2, 3, 2, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 1, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3])
```

[29]:
```
plot_image(gmm, y_gmm, 'GMM', means=True)
```



1.0.8 Model evaluation

[30]:
```
pd.Series({"KMeans": silhouette_score(X_scaled, y_kmean),
               "DBSCAN": silhouette_score(X_scaled, y_dbscan),
               "GMM": silhouette_score(X_scaled, y_gmm)}, name="Silhouette␣
 ↪Score") \
               .sort_values(ascending=False)
```

[30]:
```
KMeans    0.554657
GMM       0.553689
DBSCAN    0.322194
Name: Silhouette Score, dtype: float64
```

As per silhouette score, KMeans is the best mdoel for clustering; five kinds of customers are identified. 1. High Income, High Spending 2. Moderate Income and Spending 3. Low Income and Spending 4. Low Income, High Spending 5. High Income, Low Spending