The data

The dataset contains mall customer data like customer ID, gender, age, yearly income and spending score. Spending score is defined by parameters such as purchasing data or behaviour.

The goal of this analysis and modeling is to plan a data driven strategy for the marketing team, so they can target the customers correctly.

Note: Some plots were way to big to add to this document and maintain a proper formatting. Also, please zoom in where is needed. Thank you!

Imports

In [1]:
```
from sklearn.preprocessing import StandardScaler

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.cluster import MeanShift, estimate_bandwidth
```

Data info

In [2]:
```
df = pd.read_csv('Customers data.csv')
df.head()
```

Out[2]:

	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

In [3]:
```
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
```

Visualization

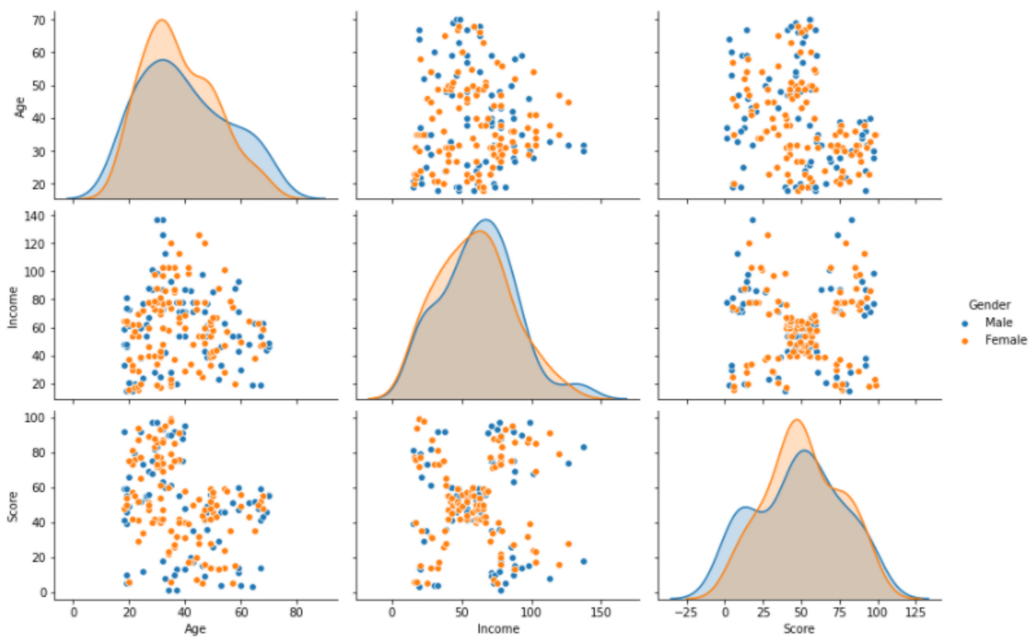
I renamed two columns for a less confusing approach to the data, dropped the gender column since it doesn't have any relation to customer segmentation and continued with other features.

In [4]:
```
df.rename(index=str, columns={'Annual Income (k$)': 'Income',
                              'Spending Score (1-100)': 'Score'}, inplace=True)
df.head()
```

Out[4]:

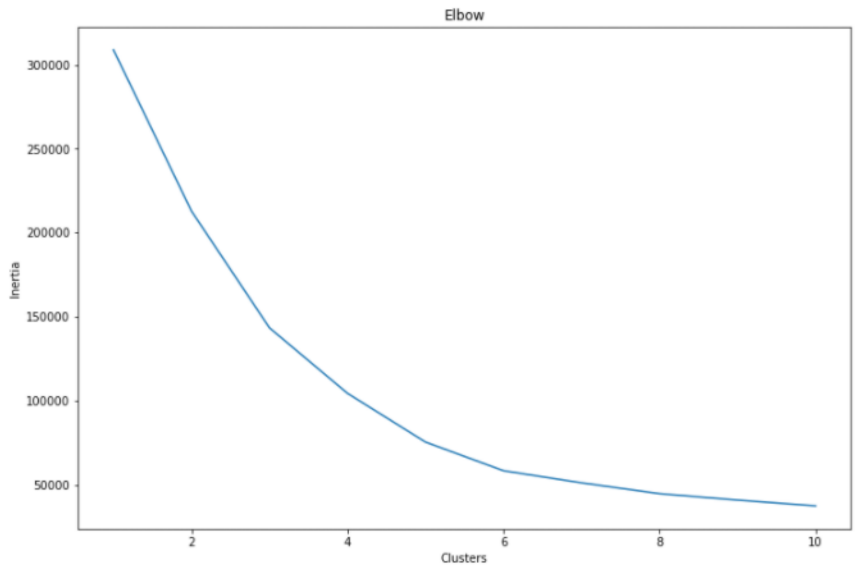
	CustomerID	Gender	Age	Income	Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

In [5]:
```
data = df.drop(['CustomerID', 'Gender'], axis=1)
sns.pairplot(df.drop('CustomerID', axis=1), hue='Gender', aspect=1.5)
plt.show()
```



kMeans

```
In [6]: clusters = []

        for i in range(1, 11):
            km = KMeans(n_clusters=i).fit(data)
            clusters.append(km.inertia_)

        fig, ax = plt.subplots(figsize=(12, 8))
        sns.lineplot(x=list(range(1, 11)), y=clusters, ax=ax)
        ax.set_title('Elbow')
        ax.set_xlabel('Clusters')
        ax.set_ylabel('Inertia')

Out[6]: Text(0, 0.5, 'Inertia')
```



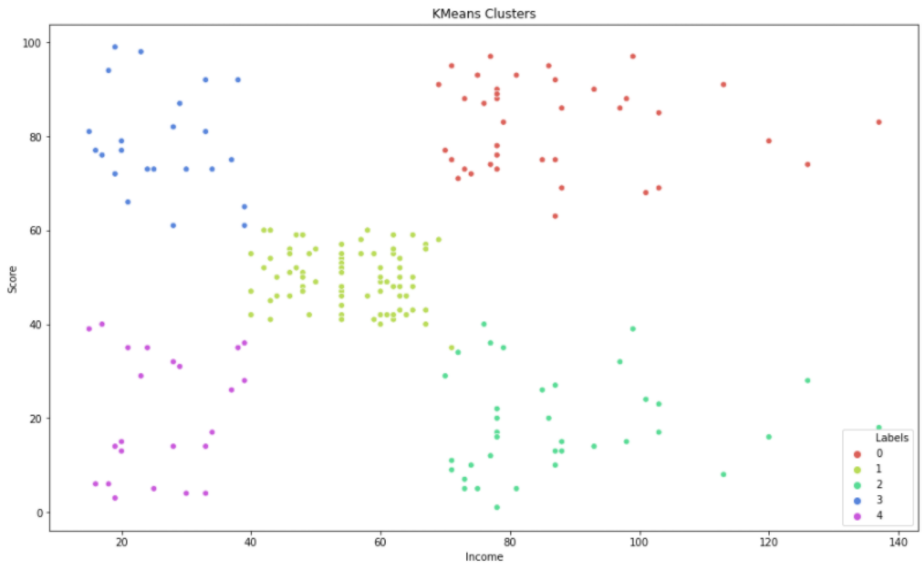
I used K-means for customer segmentation since I can quickly draw insights from the unlabeled data.

Above I used the elbow method so I can see where there's a significant change in inertia and by looking at the graph we can easily conclude that it's either 3 or 5.

I went with 5, since after plotting a 3 cluster scatter, it didn't really uncover anything.

Below you may find the 5 cluster plot, a label explanation and a swarmplot for a clearer view:

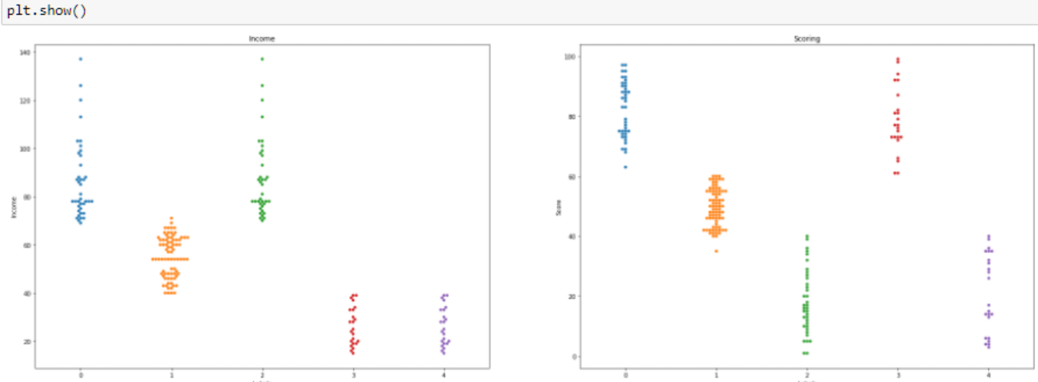
```
In [7]: kmns = KMeans(n_clusters=5).fit(data)

        data['Labels'] = kmns.labels_
        plt.figure(figsize=(15, 9))
        sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'],
                        palette=sns.color_palette('hls', 5))
        plt.title('KMeans Clusters')
        plt.show()
```



0 - low income and low spending
1 - high income and high spending
2 - mid income and mid spending
3 - high income and low spending
4 - low income and high spending

Swarm:

```
In [8]: fig = plt.figure(figsize=(30,10))
        ax = fig.add_subplot(121)
        sns.swarmplot(x='Labels', y='Income', data=data, ax=ax)
        ax.set_title('Income')

        ax = fig.add_subplot(122)
        sns.swarmplot(x='Labels', y='Score', data=data, ax=ax)
        ax.set_title('Scoring')

        plt.show()
```



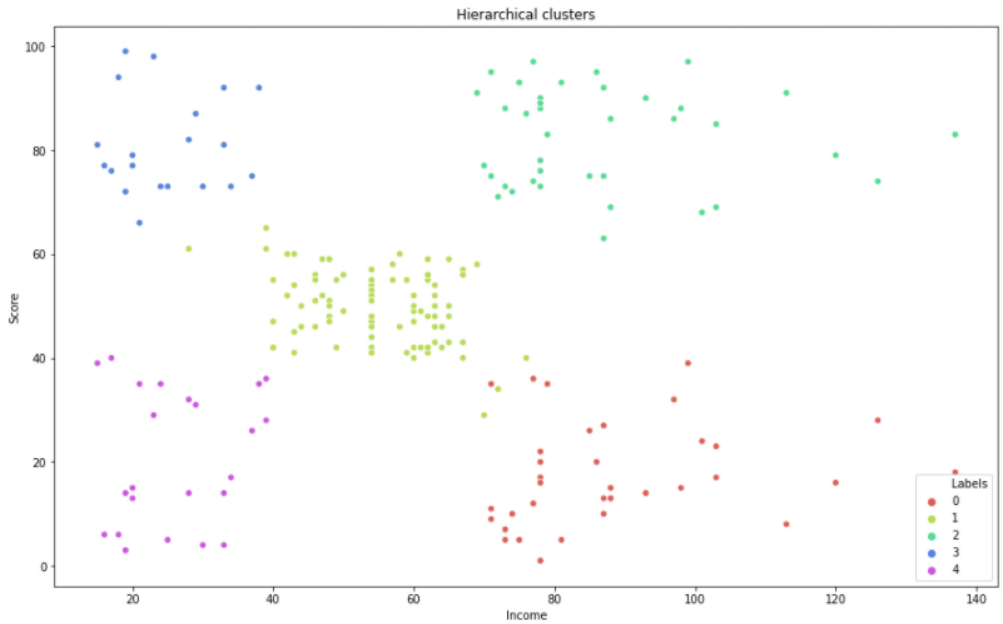
Hierarchical clustering

The second clustering technique I applied is Hierchical clustering together with average linkage as my linkage criteria.

This clustering technique will require two inputs, mainly:

- N_clusters
- linkage

In [9]:
```
# Hierarchical clustering
hclustering = AgglomerativeClustering(n_clusters=5, linkage='average').fit(data)

data['Labels'] = hclustering.labels_
plt.figure(figsize=(15, 9))
sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'],
                palette=sns.color_palette('hls', 5))
plt.title('Hierarchical clusters')
plt.show()
```



Dendrogram for the Hierarchical Clustering:

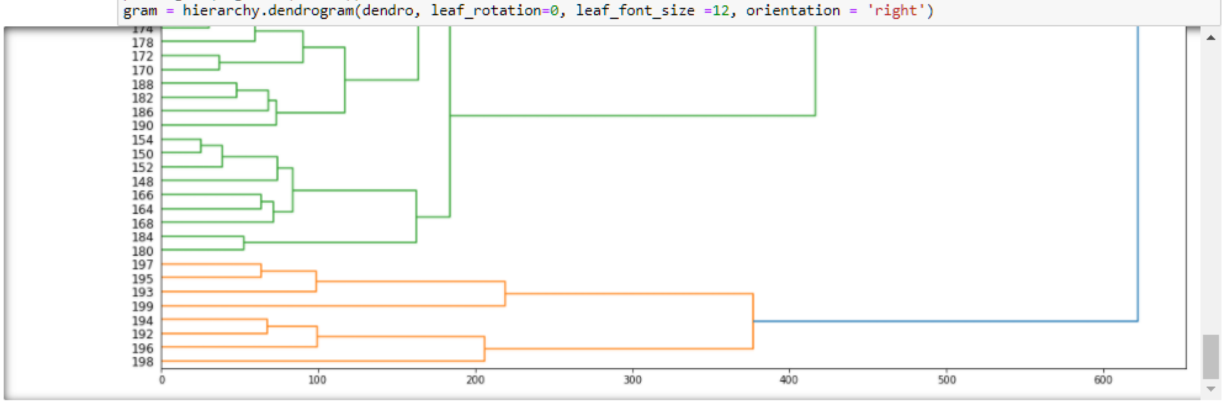
Distance matrix:

```
In [10]: # Distance matrix
         from scipy.cluster import hierarchy
         from scipy.spatial import distance_matrix

         dist = distance_matrix(data, data)
         print(dist)
```

```
[[  0.          42.05948169  33.03028913 ... 117.12813496 124.53915047
  130.17296186]
 [ 42.05948169   0.          75.01999733 ... 111.76761606 137.77880824
  122.35195135]
 [ 33.03028913  75.01999733   0.         ... 129.89226305 122.24974438
  143.78456106]
 ...
 [117.12813496 111.76761606 129.89226305 ...   0.          57.10516614
   14.35270009]
 [124.53915047 137.77880824 122.24974438 ...  57.10516614   0.
   65.06150936]
 [130.17296186 122.35195135 143.78456106 ...  14.35270009  65.06150936
    0.        ]]
```

Dendrogram:

```
In [11]: dendro = hierarchy.linkage(dist, 'average')
         plt.figure(figsize=(18, 50))
         gram = hierarchy.dendrogram(dendro, leaf_rotation=0, leaf_font_size =12, orientation = 'right')
```

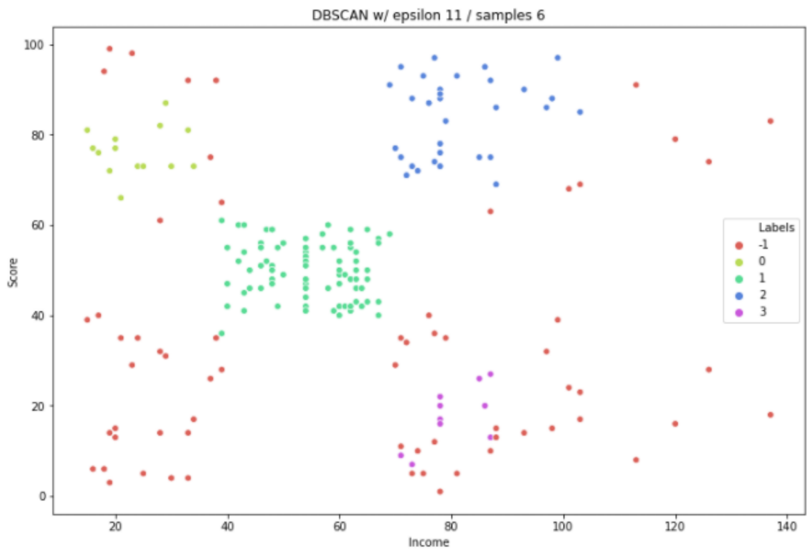


Hierarchical clustering is usually visualized as a dendrogram. The dendrogram allowed me to reconstruct the history of merges that can be seen in the depicted clustering.

DBSCAN

This technique is usually used when handling bigger datasets. Since my dataset is quite small, you will see below that this model did not perform well.

```
In [12]: #DBSCAN
         db = DBSCAN(eps=11, min_samples=6).fit(data)

         data['Labels'] = db.labels_
         plt.figure(figsize=(12, 8))
         sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'],
                         palette=sns.color_palette('hls', np.unique(db.labels_).shape[0]))

         plt.title('DBSCAN w/ epsilon 11 / samples 6')
         plt.show()
```

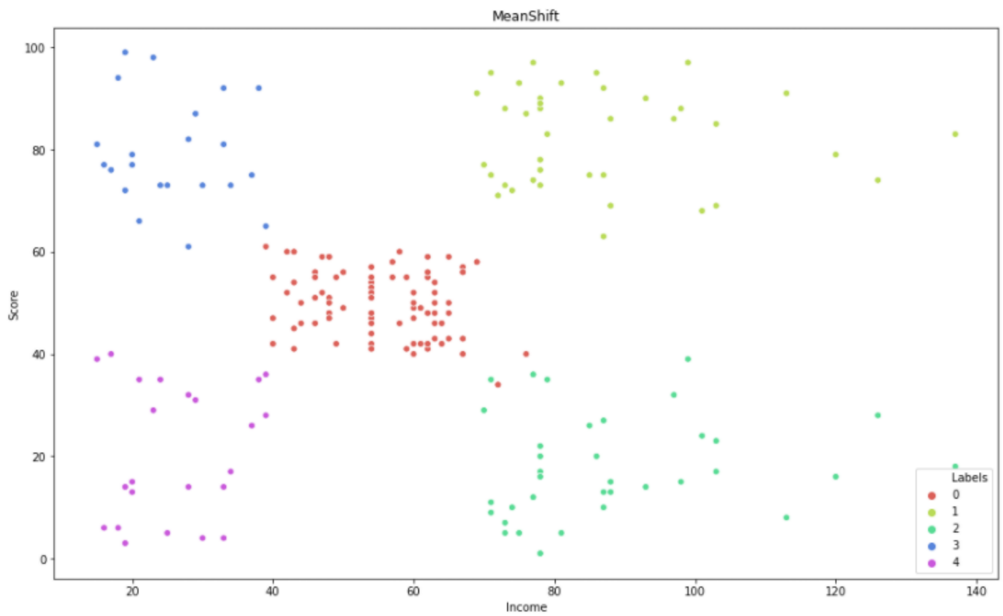


Label -1 = outliers; this means it will appear most as outliers.

MeanShift

This algorithm can automatically set the number of clusters, instead of relying on a bandwidth that dictates the size of the region to search through.

In [13]:
```
#  MeanShift
bandwidth = estimate_bandwidth(data, quantile=0.1)
ms = MeanShift(bandwidth).fit(data)

data['Labels'] = ms.labels_
plt.figure(figsize=(15, 9))
sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'],
                palette=sns.color_palette('hls', np.unique(ms.labels_).shape[0]))
plt.plot()
plt.title('MeanShift')
plt.show()
```



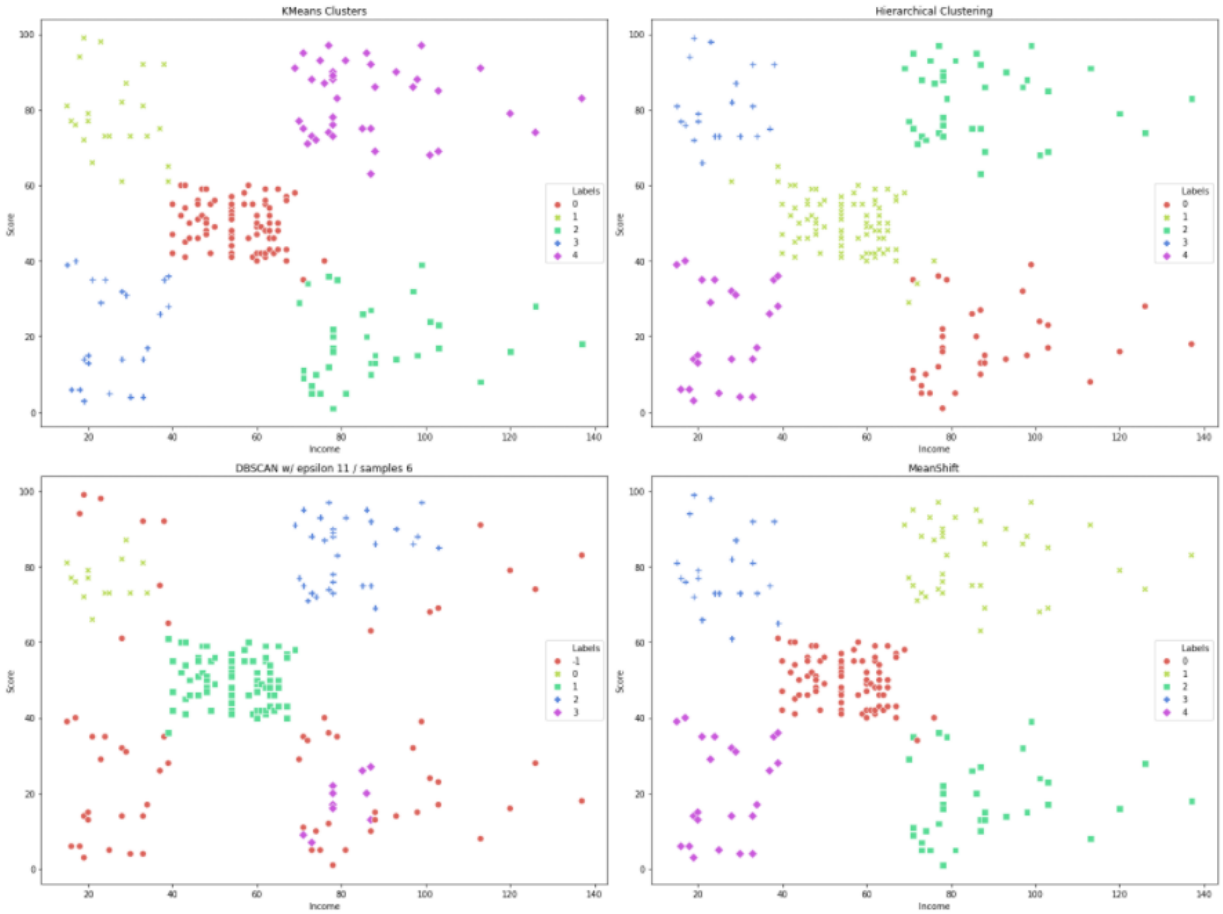
The gathering

I wrapped all the algorithms together so I could get a “top-down” view of how each clustering technique performed.

Code:

```
In [14]: fig = plt.figure(figsize=(20,15))

# Kmeans
ax = fig.add_subplot(221)

kmns = KMeans(n_clusters=5).fit(data)
data['Labels'] = kmns.labels_
sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'], style=data['Labels'],
                palette=sns.color_palette('hls', 5), s=60, ax=ax)

ax.set_title('KMeans Clusters')


# H. Clustering
ax = fig.add_subplot(222)

hclustering = AgglomerativeClustering(n_clusters=5, linkage='average').fit(data)
data['Labels'] = hclustering.labels_
sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'], style=data['Labels'],
                palette=sns.color_palette('hls', 5), s=60, ax=ax)

ax.set_title('Hierarchical Clustering')


# DBSCAN
ax = fig.add_subplot(223)

db = DBSCAN(eps=11, min_samples=6).fit(data)
data['Labels'] = db.labels_
sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'], style=data['Labels'], s=60,
                palette=sns.color_palette('hls', np.unique(db.labels_).shape[0]), ax=ax)
ax.set_title('DBSCAN w/ epsilon 11 / samples 6')


# MeanShift
ax = fig.add_subplot(224)

bandwidth = estimate_bandwidth(data, quantile=0.1)
ms = MeanShift(bandwidth).fit(data)
data['Labels'] = ms.labels_
sns.scatterplot(data['Income'], data['Score'], hue=data['Labels'], style=data['Labels'], s=60,
                palette=sns.color_palette('hls', np.unique(ms.labels_).shape[0]), ax=ax)

ax.set_title('MeanShift')

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

Result:



<u>Conclusion:</u>

Based on the resulted plots after applying **kMeans**, **Hierarchical Clustering**, **DBSCAN** and **MeanShift** techniques to my dataset, I would personally choose either kMeans or Meanshift as my go-to model in order to achieve a better customer segmentation.

Thank you!