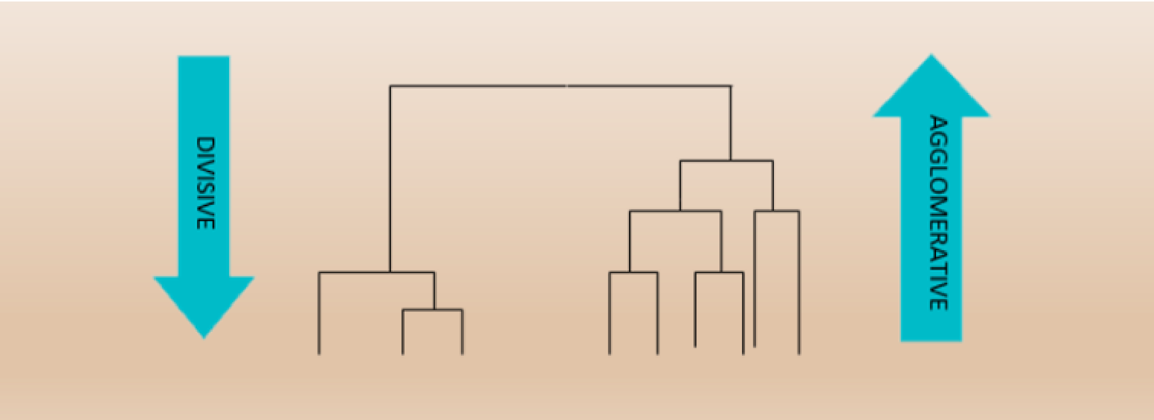
Hierarchical Clustering

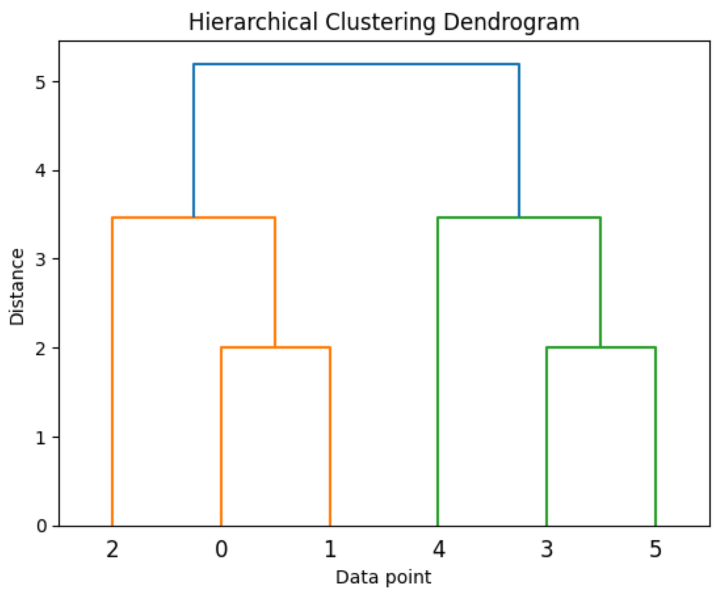
Hierarchical Clustering: Understanding Data Relationships through Dendrograms
Hierarchical Clustering is an unsupervised machine learning technique used to build a hierarchy of clusters. This method is particularly valuable for discovering relationships between data points without needing to predefine the number of clusters.

Key Features of Hierarchical Clustering:

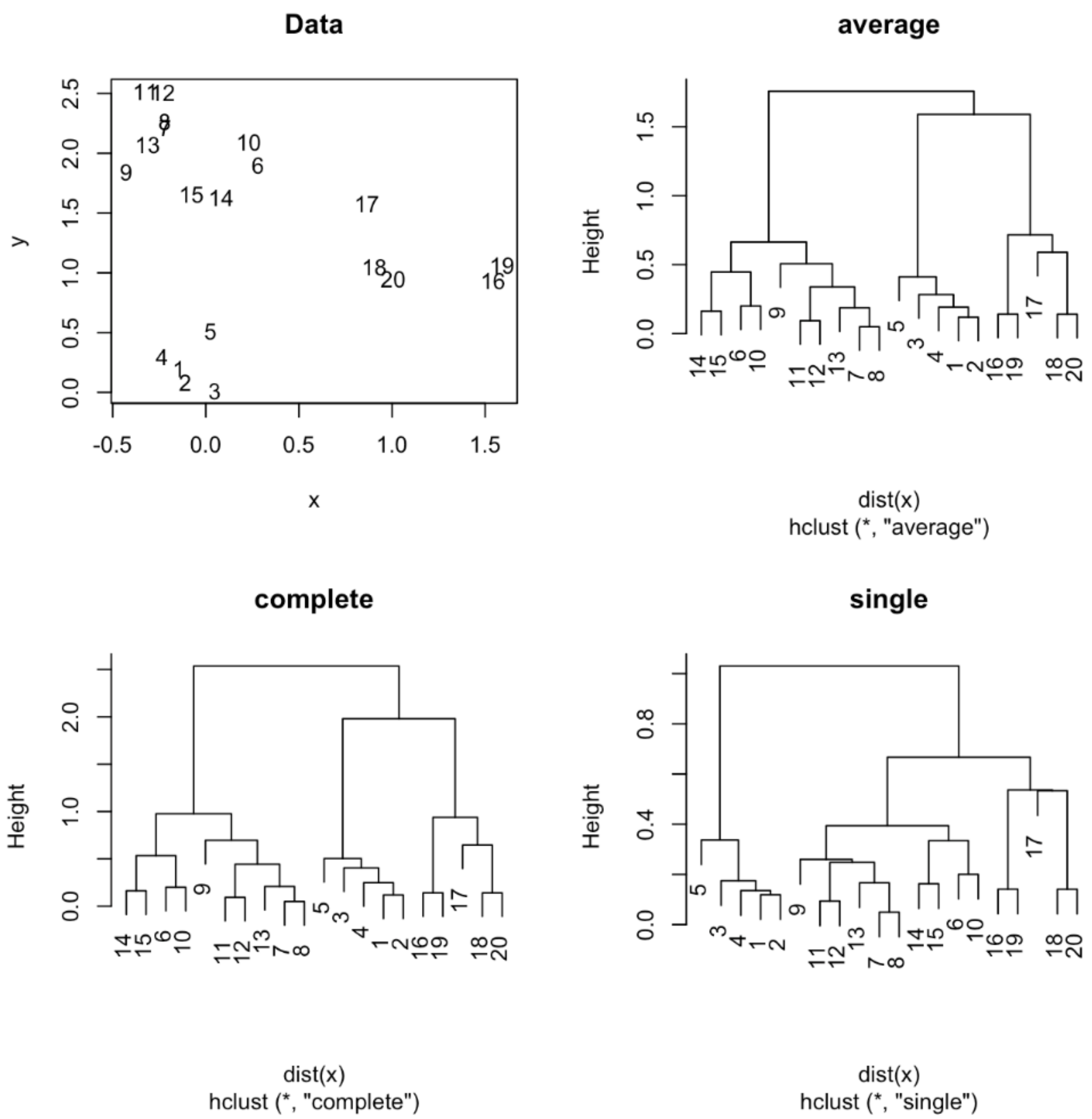
1. **Agglomerative vs. Divisive Clustering**:
   - **Agglomerative Clustering**: This is the most used approach, which follows a bottom-up strategy. It starts with each data point as its own cluster and iteratively merges the closest clusters until a single cluster is formed or a specified number of clusters is achieved.
   - **Divisive Clustering**: This top-down approach begins with all data points in one cluster and recursively splits them into smaller clusters. Although less commonly used due to its computational intensity, it can provide useful insights into data structures.



2. **Dendrogram**:
   - A dendrogram is a tree-like diagram that illustrates the arrangement of clusters formed through hierarchical clustering. Each branch represents a cluster, and the height at which branches merge indicates the distance or dissimilarity between clusters. This visualization helps interpret the data structure and the relationships among different clusters.



3. **Linkage Criteria**:
   - The linkage criteria determine how the distance between clusters is calculated during the merging process. Common methods include:
     - **Single Linkage**: Distance between the closest points of two clusters.
     - **Complete Linkage**: Distance between the farthest points of two clusters.
     - **Average Linkage**: Average distance between all points of two clusters.
     - **Ward's Linkage**: Minimizes the total within-cluster variance when merging clusters.



How Hierarchical Clustering Works:

1. **Initialization**:
   - Start with each data point as a separate cluster.
2. **Distance Calculation**:
   - Compute the distance (or similarity) between each pair of clusters.
3. **Merging Clusters**:
   - Identify the two closest clusters based on the chosen linkage criteria and merge them into a single cluster.
4. **Iteration**:
   - Repeat the distance calculation and merging steps until all data points are clustered or a specific stopping criterion is met.

Advantages of Hierarchical Clustering:

- **No Predefined Number of Clusters**: Unlike methods like K-Means, hierarchical clustering does not require specifying the number of clusters in advance.
- **Visual Representation**: The dendrogram provides an intuitive visualization of the clustering process, making it easy to interpret relationships.
- **Flexible**: It can handle different types of data and is adaptable to various distance metrics.

Limitations:

- **Computationally Intensive**: Hierarchical clustering can be slow and memory-consuming for large datasets due to the need to compute distances between all pairs of clusters.
- **Sensitivity to Noise**: Outliers can disproportionately affect the formation of clusters, leading to misleading interpretations.
- **Choice of Linkage and Distance Metrics**: The results can vary significantly based on the selected linkage criteria, necessitating careful consideration.

Applications:

- **Customer Segmentation**: Businesses often use hierarchical clustering to identify distinct customer groups based on purchasing behavior.
- **Bioinformatics**: It is commonly applied in genetic analysis to group similar genes or species based on expression patterns.
- **Document Clustering**: Hierarchical clustering can help in organizing documents into thematic clusters for information retrieval.

In summary, Hierarchical Clustering, particularly the agglomerative approach, is a powerful tool for exploring data relationships and understanding the underlying structure of datasets. Its ability to provide a visual representation of clusters through dendrograms makes it invaluable for analysis in various domains. However, the computational demands and sensitivity to outliers highlight the need for careful application in practical scenarios.

Notebook

October 21, 2024

[1]:
```
### Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.figure_factory as ff
from sklearn.metrics import confusion_matrix, accuracy_score,␣
↪classification_report
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
### Import the Dataset
df = pd.read_csv(r"C:\Users\Zahid.Shaikh\100days\60\Mall_Customers.
↪csv",header=0)
df.head()
```

[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
```

[3]:
```
df.shape ### Checking Shape
```

[3]:
```
(200, 5)
```

[4]:
```
df.describe() ### Get information of the Dataset
```

[4]:
```
       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
```

[5]:
```
df.columns ### Checking Columns
```

[5]:
```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

[6]:
```
df.info() ### Checking Information About a DataFrame
```

```
<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
```

[7]:
```
df.isnull().sum() ### Checking Null Values in the Data
```

[7]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

[8]:
```
df1 = pd.DataFrame.copy(df)
df1.shape
```

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

[9]:
```
for i in df1.columns:
    print({i:df1[i].unique()}) ### Checking Unique values in each columns
```

```
{'CustomerID': array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,
12,  13,
        14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,
        27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,
        40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,
        53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  65,
        66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,
        79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,
```

```
         92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103, 104,
        105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
        118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
        131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
        144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
        157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
        170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182,
        183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
        196, 197, 198, 199, 200], dtype=int64)}
{'Gender': array(['Male', 'Female'], dtype=object)}
{'Age': array([19, 21, 20, 23, 31, 22, 35, 64, 30, 67, 58, 24, 37, 52, 25, 46,
54,
       29, 45, 40, 60, 53, 18, 49, 42, 36, 65, 48, 50, 27, 33, 59, 47, 51,
       69, 70, 63, 43, 68, 32, 26, 57, 38, 55, 34, 66, 39, 44, 28, 56, 41],
      dtype=int64)}
{'Annual Income (k$)': array([ 15,  16,  17,  18,  19,  20,  21,  23,  24,  25,
28,  29,  30,
        33,  34,  37,  38,  39,  40,  42,  43,  44,  46,  47,  48,  49,
        50,  54,  57,  58,  59,  60,  61,  62,  63,  64,  65,  67,  69,
        70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  81,  85,  86,
        87,  88,  93,  97,  98,  99, 101, 103, 113, 120, 126, 137],
      dtype=int64)}
{'Spending Score (1-100)': array([39, 81,  6, 77, 40, 76, 94,  3, 72, 14, 99,
15, 13, 79, 35, 66, 29,
       98, 73,  5, 82, 32, 61, 31, 87,  4, 92, 17, 26, 75, 36, 28, 65, 55,
       47, 42, 52, 60, 54, 45, 41, 50, 46, 51, 56, 59, 48, 49, 53, 44, 57,
       58, 43, 91, 95, 11,  9, 34, 71, 88,  7, 10, 93, 12, 97, 74, 22, 90,
       20, 16, 89,  1, 78, 83, 27, 63, 86, 69, 24, 68, 85, 23,  8, 18],
      dtype=int64)}
```

[10]:

```python
### Finding numerical variables
colname_num = [var for var in df1.columns if df1[var].dtype!='O']
print('There are {} numerical variables\n'.format(len(colname_num)))
print('The numerical variables are :', colname_num)
```

```
There are 4 numerical variables

The numerical variables are : ['CustomerID', 'Age', 'Annual Income (k$)',
'Spending Score (1-100)']
```

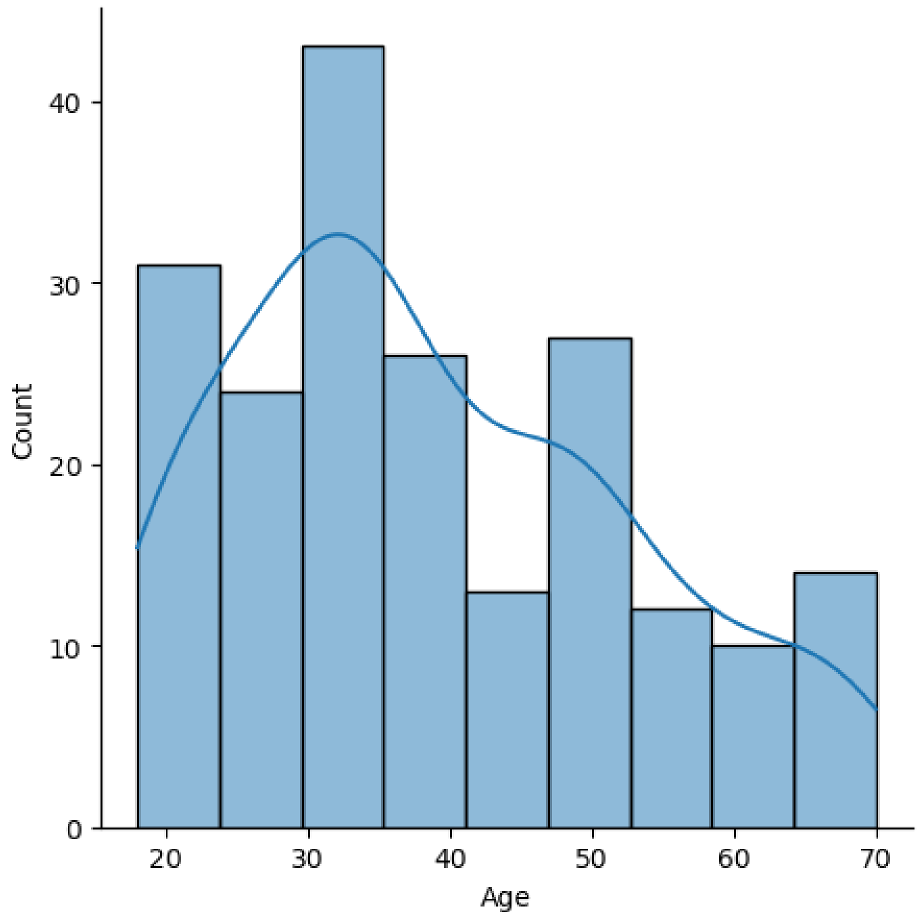
[11]:

```python
### Finding categorical variables
colname_cat = [var for var in df1.columns if df1[var].dtype=='O']
print('There are {} categorical variables\n'.format(len(colname_cat)))
print('The categorical variables are :', colname_cat)
```

```
There are 1 categorical variables

The categorical variables are : ['Gender']
```

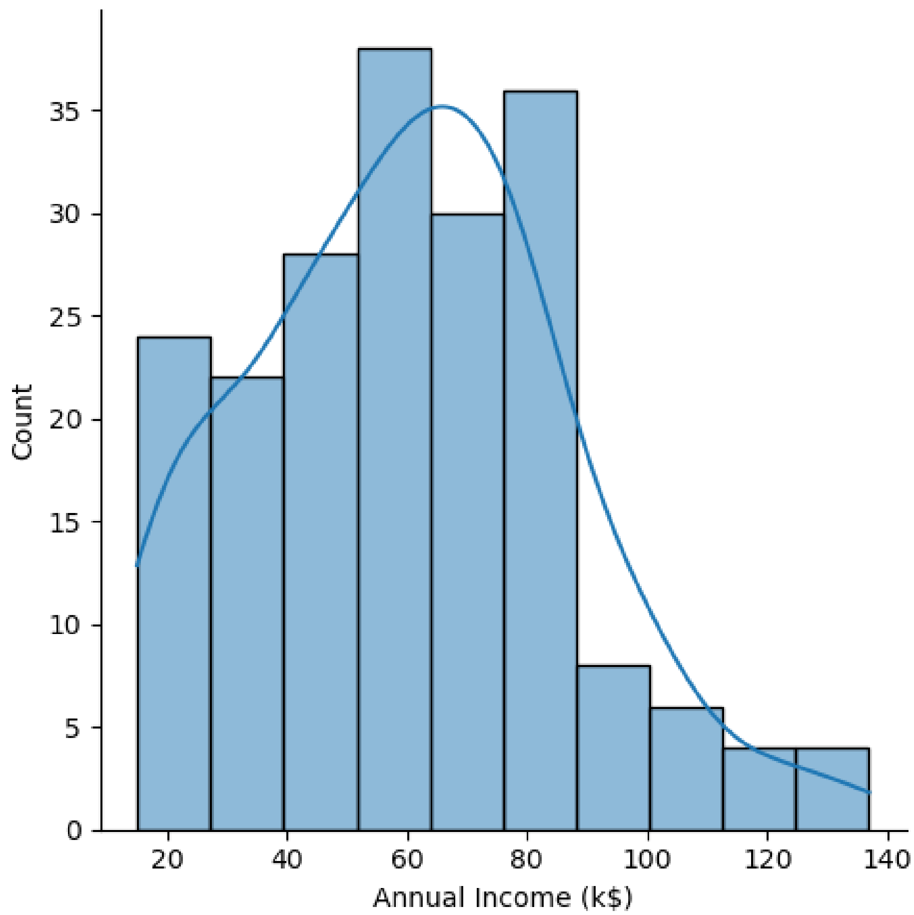
[12]:
```
### Distribution of age
sns.displot(x='Age', data=df1, kde=True)
```

[12]: <seaborn.axisgrid.FacetGrid at 0x1da8c01e360>



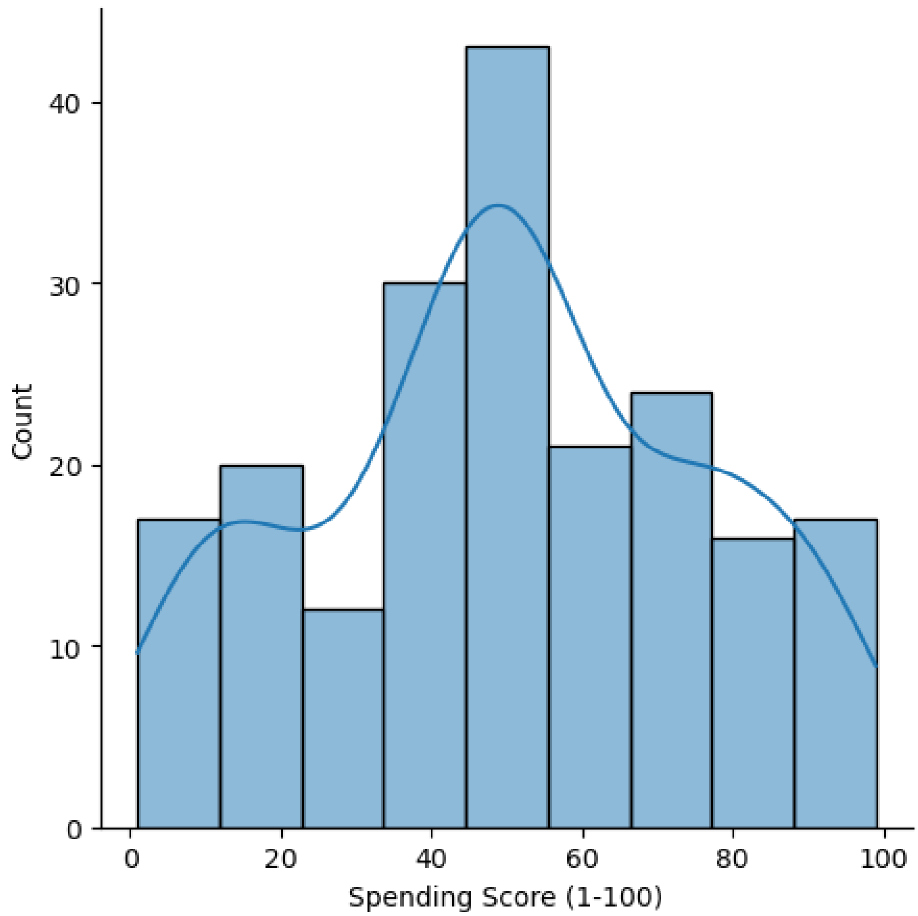
[13]:
```
### Distribution of income
sns.displot(x='Annual Income (k$)', data=df1, kde=True)
```

[13]: <seaborn.axisgrid.FacetGrid at 0x1da8c09ffe0>



[14]:
```
### Distribution of score
sns.displot(x='Spending Score (1-100)', data=df1, kde=True)
```

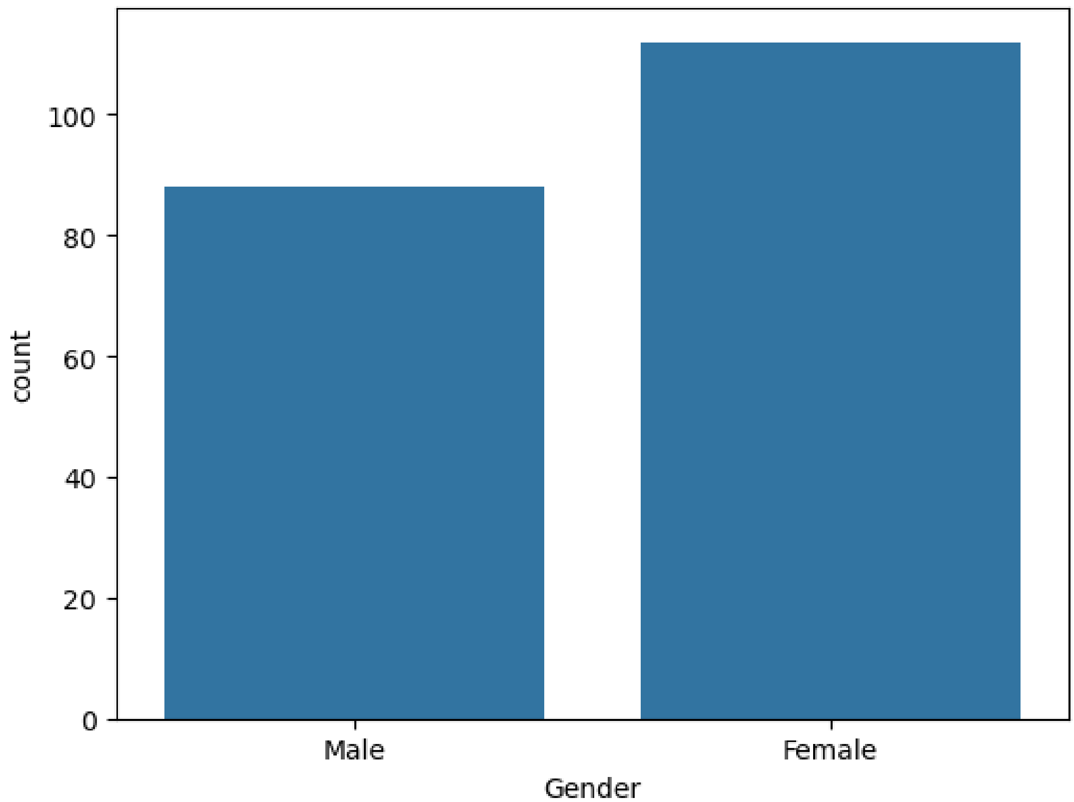
[14]: <seaborn.axisgrid.FacetGrid at 0x1da8c19bef0>



[15]:
```
# distribution of categorical variable
print(df1['Gender'].value_counts())
sns.countplot(x='Gender', data=df1)
```

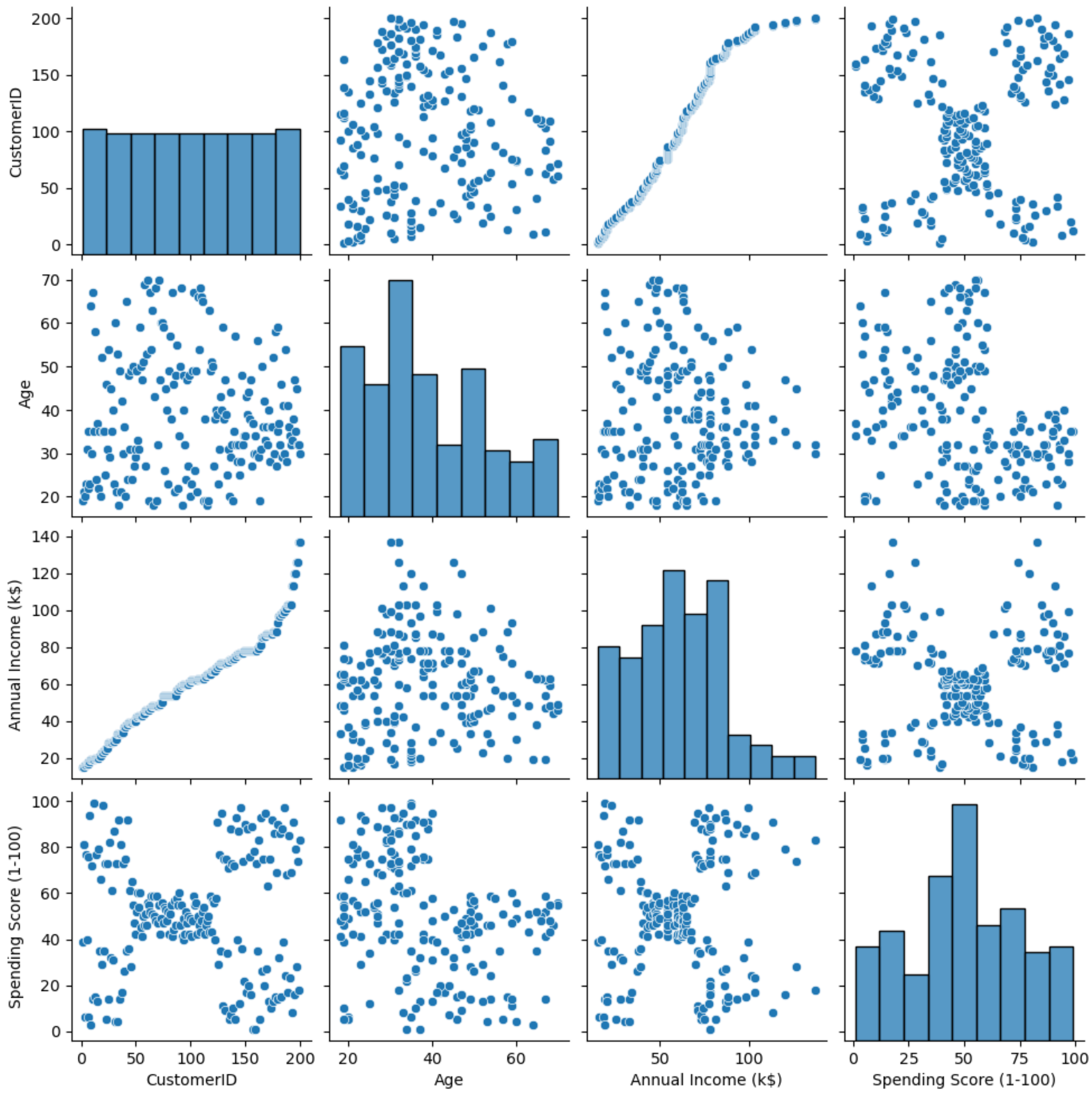
```
Gender
Female    112
Male       88
Name: count, dtype: int64
```

[15]: `<Axes: xlabel='Gender', ylabel='count'>`



[16]:
```
# Creates pairwise scatter plots for all features in the dataframe 'df1'.
sns.pairplot(df1)
```

[16]: <seaborn.axisgrid.PairGrid at 0x1da8c175a90>



[17]:
```
df2 = df1.copy()
df2.shape
```

[17]: (200, 5)

[18]:
```
### Feature sleection for the model
#Considering only 2 features (Annual income and Spending Score) and no Label␣
↪available
X = df2.iloc[:, [3,4]].values
X
```

[18]:
```
array([[ 15,  39],
       [ 15,  81],
```

```
[ 16,   6],
[ 16,  77],
[ 17,  40],
[ 17,  76],
[ 18,   6],
[ 18,  94],
[ 19,   3],
[ 19,  72],
[ 19,  14],
[ 19,  99],
[ 20,  15],
[ 20,  77],
[ 20,  13],
[ 20,  79],
[ 21,  35],
[ 21,  66],
[ 23,  29],
[ 23,  98],
[ 24,  35],
[ 24,  73],
[ 25,   5],
[ 25,  73],
[ 28,  14],
[ 28,  82],
[ 28,  32],
[ 28,  61],
[ 29,  31],
[ 29,  87],
[ 30,   4],
[ 30,  73],
[ 33,   4],
[ 33,  92],
[ 33,  14],
[ 33,  81],
[ 34,  17],
[ 34,  73],
[ 37,  26],
[ 37,  75],
[ 38,  35],
[ 38,  92],
[ 39,  36],
[ 39,  61],
[ 39,  28],
[ 39,  65],
[ 40,  55],
[ 40,  47],
[ 40,  42],
```

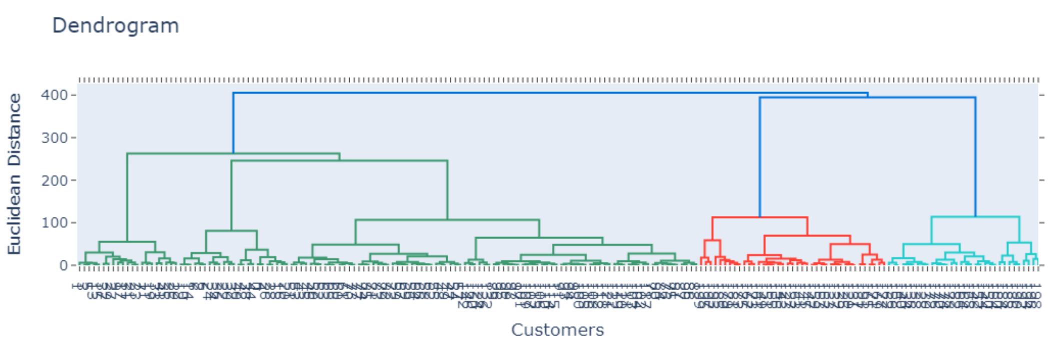
```
[ 40,  42],
[ 42,  52],
[ 42,  60],
[ 43,  54],
[ 43,  60],
[ 43,  45],
[ 43,  41],
[ 44,  50],
[ 44,  46],
[ 46,  51],
[ 46,  46],
[ 46,  56],
[ 46,  55],
[ 47,  52],
[ 47,  59],
[ 48,  51],
[ 48,  59],
[ 48,  50],
[ 48,  48],
[ 48,  59],
[ 48,  47],
[ 49,  55],
[ 49,  42],
[ 50,  49],
[ 50,  56],
[ 54,  47],
[ 54,  54],
[ 54,  53],
[ 54,  48],
[ 54,  52],
[ 54,  42],
[ 54,  51],
[ 54,  55],
[ 54,  41],
[ 54,  44],
[ 54,  57],
[ 54,  46],
[ 57,  58],
[ 57,  55],
[ 58,  60],
[ 58,  46],
[ 59,  55],
[ 59,  41],
[ 60,  49],
[ 60,  40],
[ 60,  42],
[ 60,  52],
```

```
[ 60,  47],
[ 60,  50],
[ 61,  42],
[ 61,  49],
[ 62,  41],
[ 62,  48],
[ 62,  59],
[ 62,  55],
[ 62,  56],
[ 62,  42],
[ 63,  50],
[ 63,  46],
[ 63,  43],
[ 63,  48],
[ 63,  52],
[ 63,  54],
[ 64,  42],
[ 64,  46],
[ 65,  48],
[ 65,  50],
[ 65,  43],
[ 65,  59],
[ 67,  43],
[ 67,  57],
[ 67,  56],
[ 67,  40],
[ 69,  58],
[ 69,  91],
[ 70,  29],
[ 70,  77],
[ 71,  35],
[ 71,  95],
[ 71,  11],
[ 71,  75],
[ 71,   9],
[ 71,  75],
[ 72,  34],
[ 72,  71],
[ 73,   5],
[ 73,  88],
[ 73,   7],
[ 73,  73],
[ 74,  10],
[ 74,  72],
[ 75,   5],
[ 75,  93],
[ 76,  40],
```

```
[ 76,  87],
[ 77,  12],
[ 77,  97],
[ 77,  36],
[ 77,  74],
[ 78,  22],
[ 78,  90],
[ 78,  17],
[ 78,  88],
[ 78,  20],
[ 78,  76],
[ 78,  16],
[ 78,  89],
[ 78,   1],
[ 78,  78],
[ 78,   1],
[ 78,  73],
[ 79,  35],
[ 79,  83],
[ 81,   5],
[ 81,  93],
[ 85,  26],
[ 85,  75],
[ 86,  20],
[ 86,  95],
[ 87,  27],
[ 87,  63],
[ 87,  13],
[ 87,  75],
[ 87,  10],
[ 87,  92],
[ 88,  13],
[ 88,  86],
[ 88,  15],
[ 88,  69],
[ 93,  14],
[ 93,  90],
[ 97,  32],
[ 97,  86],
[ 98,  15],
[ 98,  88],
[ 99,  39],
[ 99,  97],
[101,  24],
[101,  68],
[103,  17],
[103,  85],
```

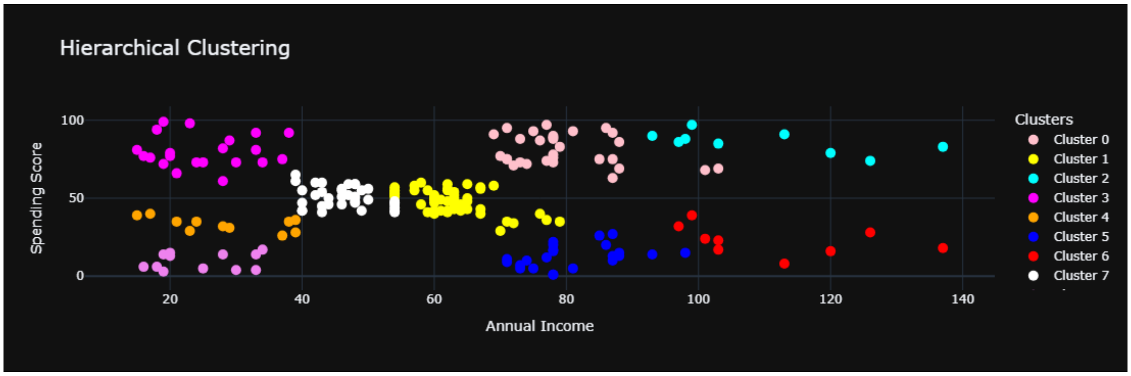
```
       [103,  23],
       [103,  69],
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
       [137,  83]], dtype=int64)
```

[51]:
```python
import plotly.figure_factory as ff
import scipy.cluster.hierarchy as sch
import numpy as np

# Perform hierarchical clustering and create the linkage matrix
linkage_matrix = sch.linkage(X, method='ward')

# Create a dendrogram
fig = ff.create_dendrogram(X, linkagefun=lambda x: linkage_matrix)

# Set the title and axis labels
fig.update_layout(
    title='Dendrogram',
    xaxis_title='Customers',
    yaxis_title='Euclidean Distance',
    font=dict(size=14)
)

# Show the figure
fig.show()
```



[59]:

```
import plotly.express as px
import plotly.graph_objects as go
from sklearn.cluster import AgglomerativeClustering

# Perform Agglomerative Clustering
hc = AgglomerativeClustering(n_clusters=9, linkage='ward')
y_hc = hc.fit_predict(X)

# Create a scatter plot using Plotly
fig = go.Figure()

# Define colors for each cluster
colors = ['pink', 'yellow', 'cyan', 'magenta', 'orange', 'blue', 'red',␣
↪'white', 'violet']

# Loop through each cluster to add scatter points
for i in range(9):
    fig.add_trace(go.Scatter(
        x=X[y_hc == i, 0],  # X coordinates for cluster i
        y=X[y_hc == i, 1],  # Y coordinates for cluster i
        mode='markers',
        marker=dict(size=10, color=colors[i]),  # Marker size and color
        name=f'Cluster {i}'  # Legend label for the cluster
    ))

# Update layout for dark theme
fig.update_layout(
    title='Hierarchical Clustering',
    title_font=dict(size=20),
    xaxis_title='Annual Income',
    yaxis_title='Spending Score',
    legend_title_text='Clusters',
    template='plotly_dark',  # Set dark theme
    hovermode='closest'  # Enable hover for better interactivity
)

# Show the figure
fig.show()
```



#####

Made with by Zahid Salim Shaikh

```
[ ]:
```

This notebook was converted with convert.ploomber.io