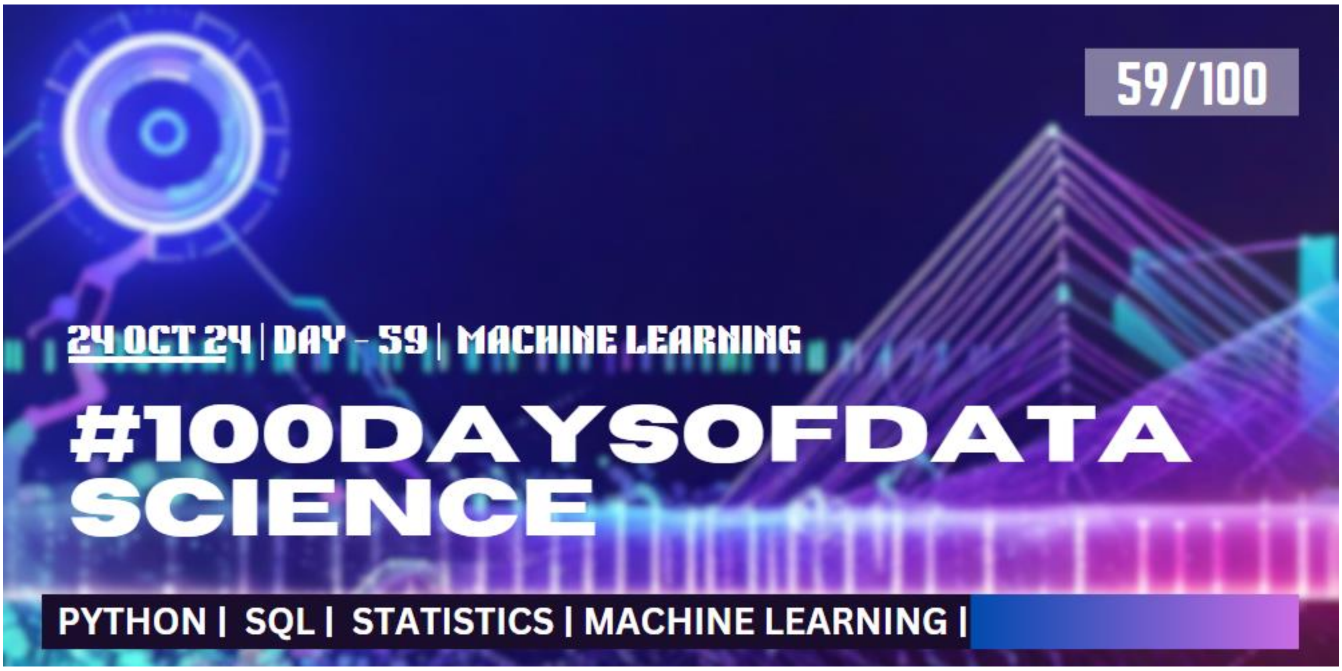
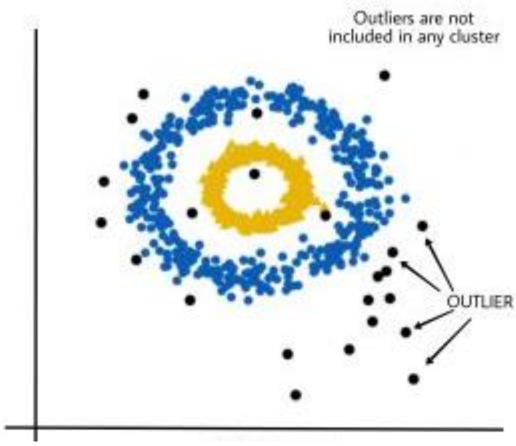


DBSCAN Clustering



DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised machine learning algorithm designed to identify clusters of arbitrary shape by grouping closely packed points together and distinguishing outliers. Unlike K-Means, it doesn't require you to predefine the number of clusters, making it ideal for noisy datasets.

• **Key Features of DBSCAN:**

1. **Density-Based Clustering**: DBSCAN forms clusters based on the density of points in the dataset. It identifies regions with a high concentration of points and separates them from regions with low point density.
2. **Outlier Detection**: Points that don't belong to any cluster are classified as outliers (noise). This feature makes DBSCAN highly effective for detecting anomalies in datasets.
3. **Cluster Flexibility**: DBSCAN can find clusters of arbitrary shapes, which is particularly useful for real-world data where clusters are often non-spherical and vary in size.

• **Parameters of DBSCAN:**

- **eps (ε)**: The maximum distance between two points to consider them neighbors. A smaller eps creates more compact clusters, while a larger eps might result in fewer, larger clusters.
- **min_samples**: The minimum number of points required to form a dense region (core points). This parameter helps distinguish between core points and outliers.

• **How DBSCAN Works:**

1. **Core Points**: Points that have at least min_samples points within a distance of eps are identified as core points.
2. **Cluster Expansion**: DBSCAN begins with a core point and expands the cluster by including all neighboring points within the eps distance. If any of the neighbors are also core points, their neighbors are added to the cluster as well, forming a larger cluster.
3. **Border Points**: Points that fall within the neighborhood of a core point but don't have enough neighbors to form their own cluster are called border points and are assigned to the nearest core point's cluster.
4. **Noise**: Points that don't belong to any cluster are labeled as noise (outliers), effectively handled by DBSCAN.

• **Advantages of DBSCAN:**

- **No Predefined Clusters**: Unlike K-Means, you don't need to specify the number of clusters beforehand.
- **Outlier Detection**: DBSCAN automatically identifies and separates noise or outliers, making it useful for datasets with anomalies.
- **Cluster Shapes**: DBSCAN can find clusters of various shapes and sizes, making it more flexible than algorithms like K-Means that assume spherical clusters.
- **Works with Non-Normalized Data**: DBSCAN can work with data that is not normalized, providing flexibility in feature scaling.

• **Challenges:**

- **Parameter Tuning**: Choosing the right eps and min_samples can be difficult and requires domain knowledge or experimentation.
- **Varied Density**: DBSCAN struggles with datasets where clusters have varying densities, as a single eps value may not fit all clusters.
- **High Dimensionality**: The performance of DBSCAN may degrade with high-dimensional data, as it becomes challenging to measure distances meaningfully.

• **Applications:**

- **Anomaly Detection**: DBSCAN is widely used in fraud detection, network intrusion detection, and identifying outliers in financial transactions.
- **Customer Segmentation**: DBSCAN can help businesses identify distinct groups of customers based on their behavior, without the need to predefine the number of customer segments.
- **Image Segmentation**: DBSCAN can be used for segmenting images by grouping similar pixels together based on density.

• **Limitations:**

- **Sensitive to eps**: The choice of eps significantly impacts the clustering result, and there is no one-size-fits-all value. Selecting a poor eps value can lead to either too many clusters or failing to identify meaningful ones.
- **Difficulty with Varying Density**: DBSCAN performs poorly when the dataset contains clusters with widely varying densities.
- **Scalability**: Although DBSCAN is efficient with small datasets, it can become computationally expensive for very large datasets, especially in higher dimensions.

In conclusion, DBSCAN is a robust algorithm for density-based clustering and outlier detection. It is well-suited for applications where the number of clusters is unknown, and there are irregular cluster shapes or outliers. However, selecting the right parameters and handling datasets with varying densities remain key challenges.

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\59\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]:
```
### 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)
```

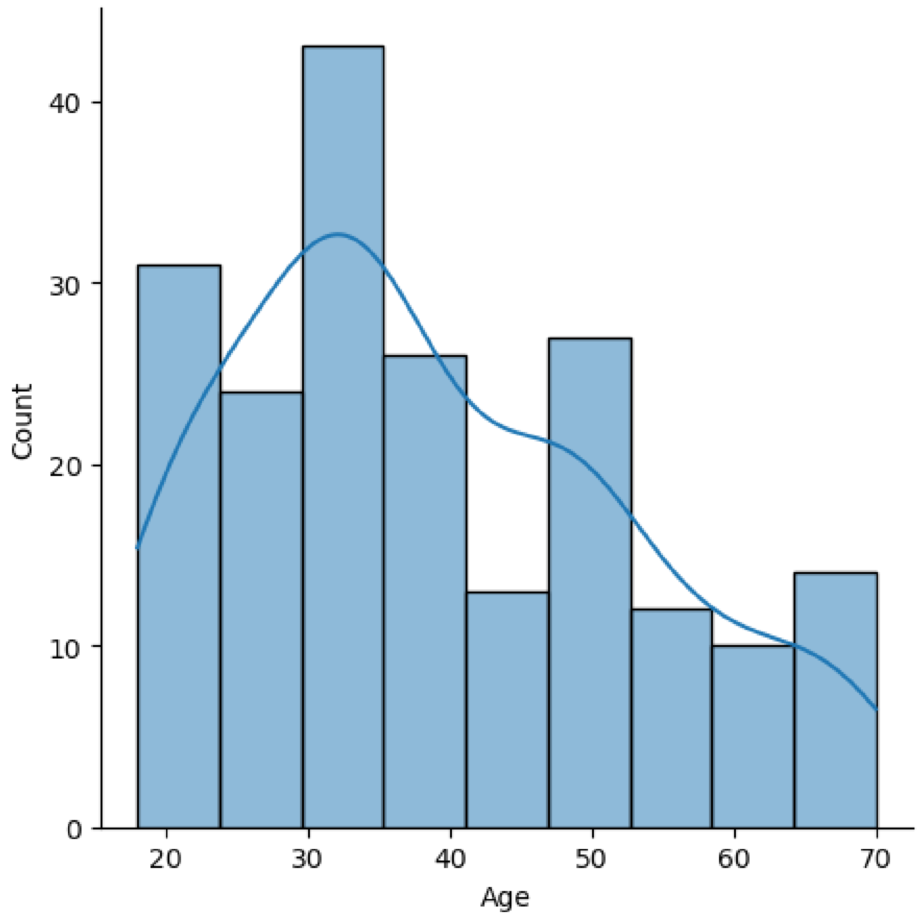
```
There are 4 numerical variables

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

[11]:
```
### 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']
```

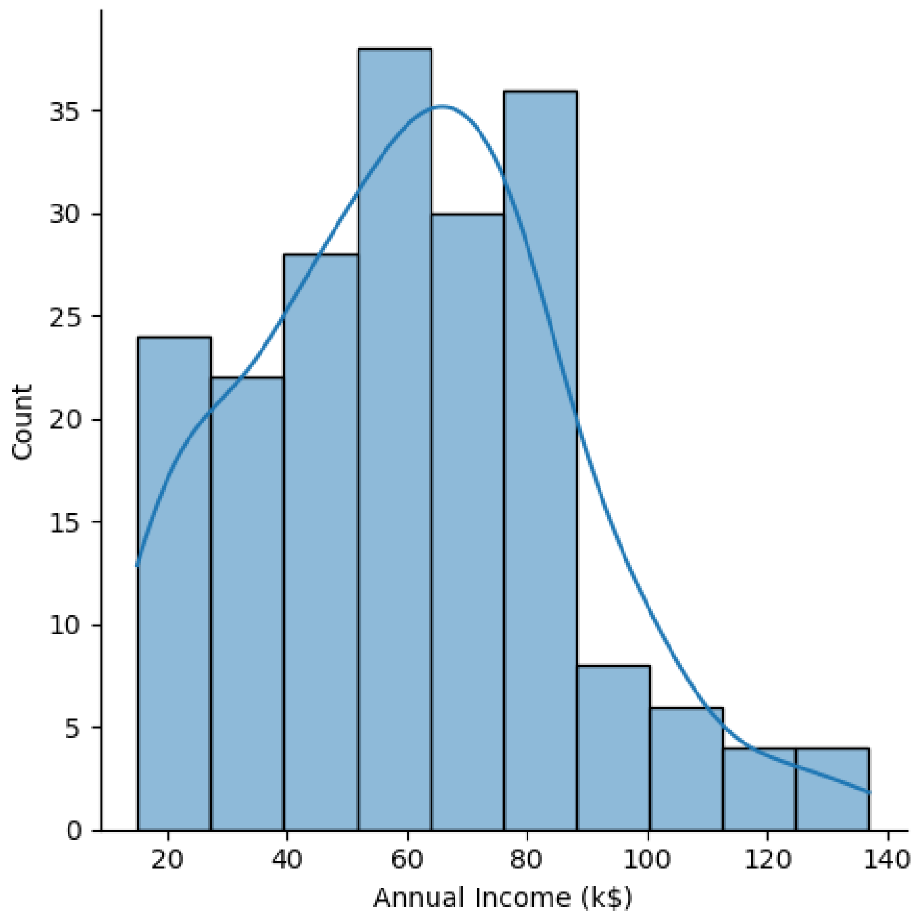
[12]:
```python
### Distribution of age
sns.displot(x='Age', data=df1, kde=True)
```

[12]: <seaborn.axisgrid.FacetGrid at 0x1e14b5866f0>



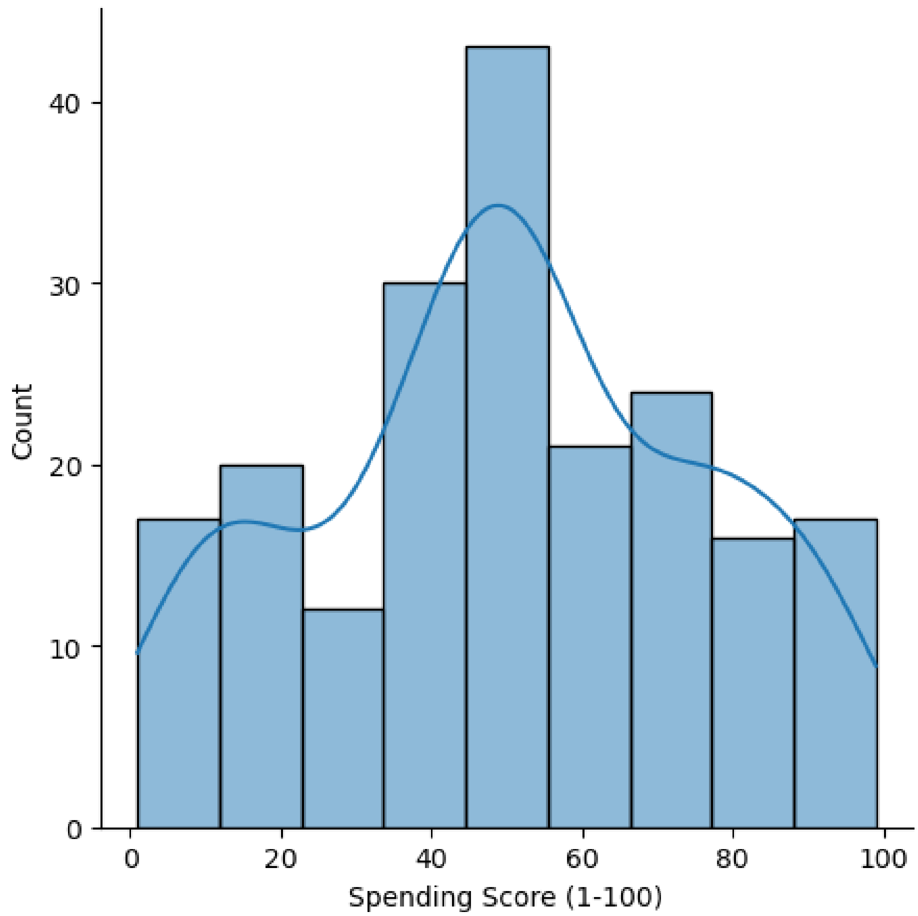
[13]:
```python
### Distribution of income
sns.displot(x='Annual Income (k$)', data=df1, kde=True)
```

[13]: <seaborn.axisgrid.FacetGrid at 0x1e14c753230>



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

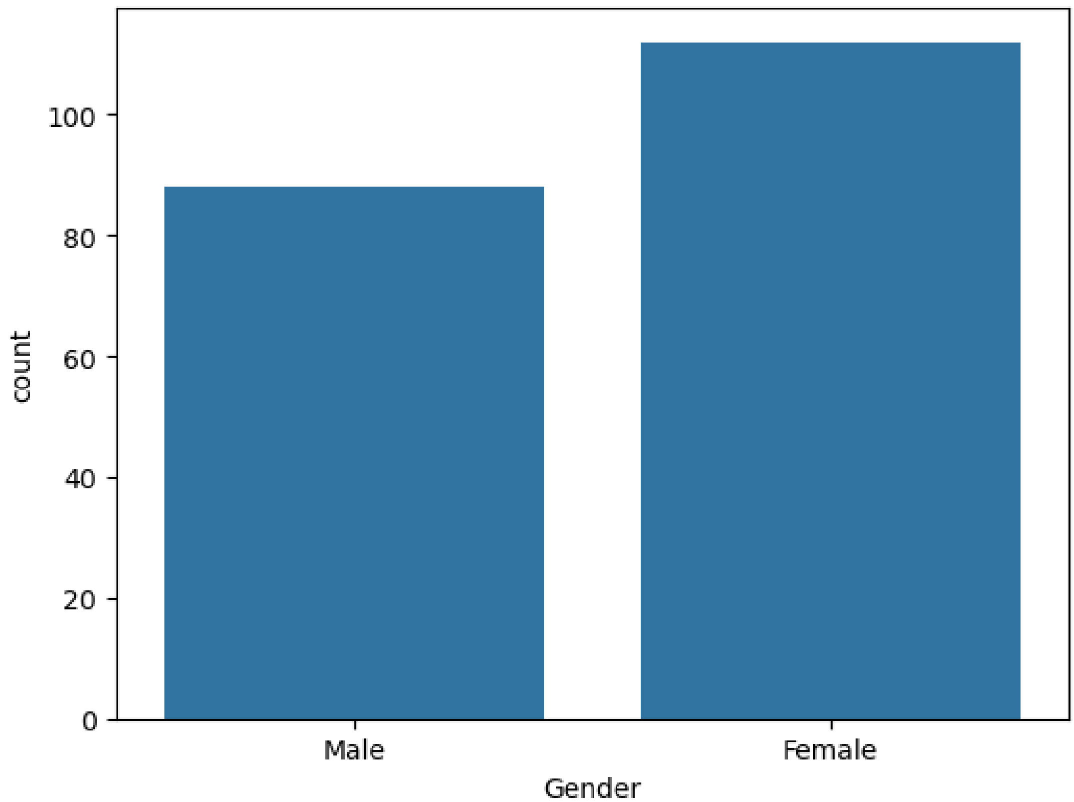
[14]: <seaborn.axisgrid.FacetGrid at 0x1e14fc0fcb0>



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

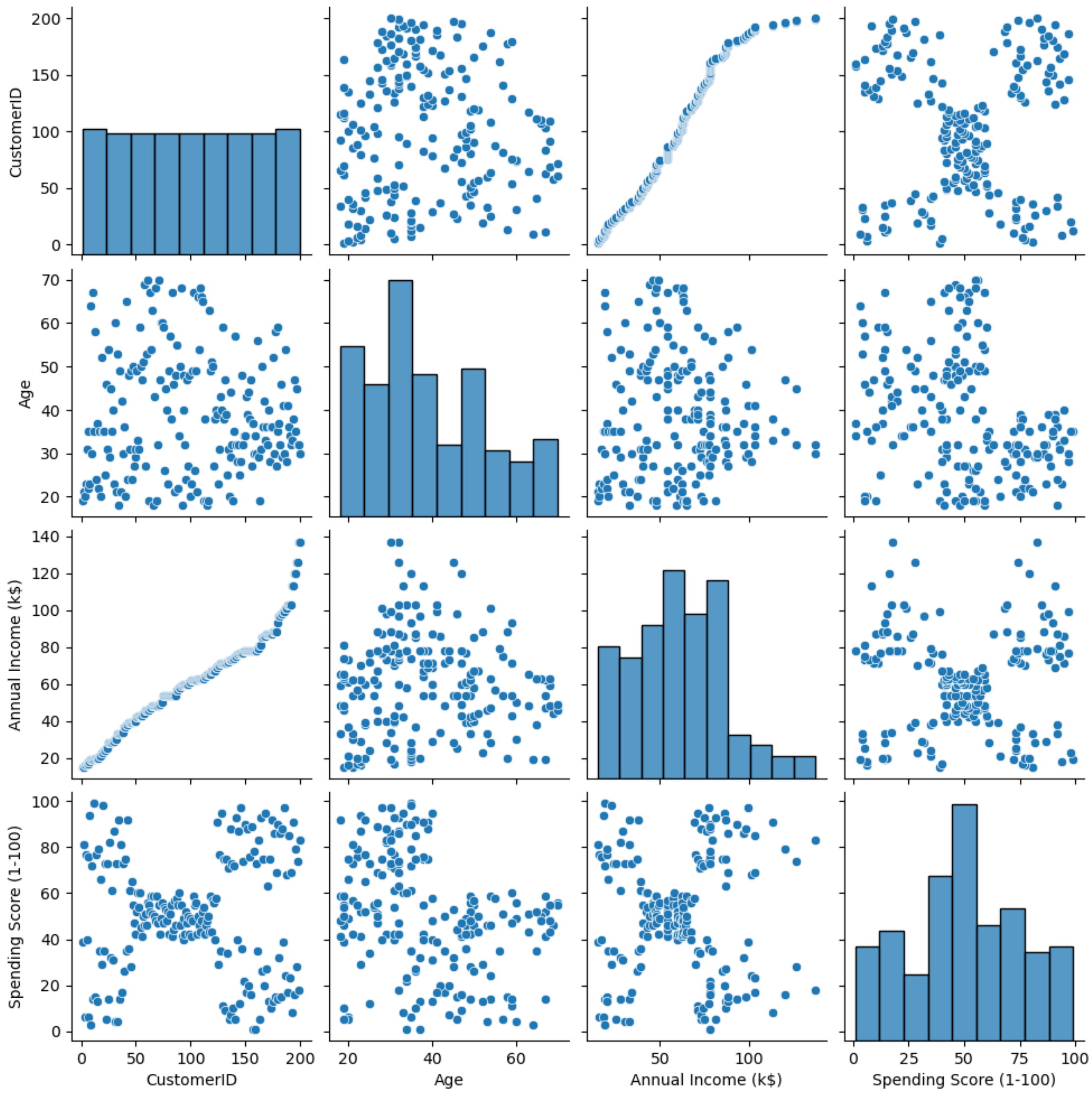
```
Gender
Female    112
Male       88
Name: count, dtype: int64
```

[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 0x1e14ca03a70>



[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)
```

[19]:
```
from sklearn.cluster import DBSCAN  # Importing DBSCAN algorithm from␣
↪scikit-learn

# Initializing DBSCAN model with:
# eps: The maximum distance between two samples for them to be considered as in␣
↪the same neighborhood.
# min_samples: The number of samples (or total weight) in a neighborhood for a␣
↪point to be considered as a core point.
# metric: The distance metric to use, in this case, Euclidean distance.
db = DBSCAN(eps=3, min_samples=4, metric='euclidean')

# Fitting the DBSCAN model to the data X
model = db.fit(X)

# Extracting the labels assigned to each data point by the DBSCAN algorithm
# -1 indicates that the point is considered noise (outlier)
label = model.labels_

# Printing the cluster labels for each data point
label
```

[19]:
```
array([-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,  0,  0,  0,  0, -1, -1,  0, -1,  0, -1,  0,  0,
       -1,  0, -1, -1,  0, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1, -1,  2,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  2,
        3,  3, -1,  3, -1, -1,  4, -1, -1, -1,  4,  5,  4, -1,  4,  5, -1,
        5,  4, -1,  4,  5, -1, -1,  6, -1, -1, -1,  7, -1,  6, -1,  6, -1,
        7, -1,  6, -1,  7, -1,  7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        8, -1,  8, -1,  8, -1,  8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1], dtype=int64)
```

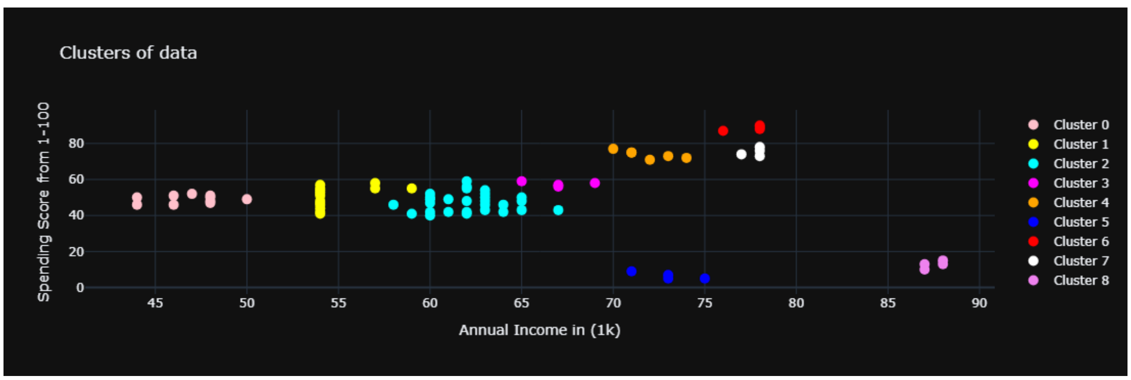
[20]:
```
from sklearn import metrics  # Importing metrics module for performance␣
↪evaluation
```

```
# Creating a boolean array `sample_cores` to identify core points in the DBSCAN␣
↪clustering
# Initializing all elements to False (i.e., assuming no core points initially)
sample_cores = np.zeros_like(label, dtype=bool)

# Setting the entries corresponding to core points (as identified by DBSCAN) to␣
↪True
# `db.core_sample_indices_` contains the indices of core samples (i.e., core␣
↪points)
sample_cores[db.core_sample_indices_] = True

# Calculating the number of clusters found by DBSCAN
# `set(label)` gives unique labels assigned by DBSCAN
# `-1` represents noise points, so we subtract 1 from the count if there is any␣
↪noise (-1)
n_clusters = len(set(label)) - (1 if -1 in label else 0)

# Printing the number of clusters detected
print('No of clusters:', n_clusters)
```

```
No of clusters: 9
```

[21]:
```
import plotly.graph_objs as go
import plotly.express as px

# Fit the DBSCAN model and predict the cluster labels
y_means = db.fit_predict(X)

# Create scatter plots for each cluster
trace0 = go.Scatter(x=X[y_means == 0, 0], y=X[y_means == 0, 1], mode='markers',␣
↪marker=dict(size=10, color='pink'), name='Cluster 0')
trace1 = go.Scatter(x=X[y_means == 1, 0], y=X[y_means == 1, 1], mode='markers',␣
↪marker=dict(size=10, color='yellow'), name='Cluster 1')
trace2 = go.Scatter(x=X[y_means == 2, 0], y=X[y_means == 2, 1], mode='markers',␣
↪marker=dict(size=10, color='cyan'), name='Cluster 2')
trace3 = go.Scatter(x=X[y_means == 3, 0], y=X[y_means == 3, 1], mode='markers',␣
↪marker=dict(size=10, color='magenta'), name='Cluster 3')
trace4 = go.Scatter(x=X[y_means == 4, 0], y=X[y_means == 4, 1], mode='markers',␣
↪marker=dict(size=10, color='orange'), name='Cluster 4')
trace5 = go.Scatter(x=X[y_means == 5, 0], y=X[y_means == 5, 1], mode='markers',␣
↪marker=dict(size=10, color='blue'), name='Cluster 5')
trace6 = go.Scatter(x=X[y_means == 6, 0], y=X[y_means == 6, 1], mode='markers',␣
↪marker=dict(size=10, color='red'), name='Cluster 6')
trace7 = go.Scatter(x=X[y_means == 7, 0], y=X[y_means == 7, 1], mode='markers',␣
↪marker=dict(size=10, color='white'), name='Cluster 7')
```

```python
trace8 = go.Scatter(x=X[y_means == 8, 0], y=X[y_means == 8, 1], mode='markers',␣
↪marker=dict(size=10, color='violet'), name='Cluster 8')

# Combine all traces
data = [trace0, trace1, trace2, trace3, trace4, trace5, trace6, trace7, trace8]

# Set up layout with dark theme, titles, and axis labels
layout = go.Layout(
    title='Clusters of data',
    xaxis=dict(title='Annual Income in (1k)'),  # X-axis label
    yaxis=dict(title='Spending Score from 1-100'),  # Y-axis label
    showlegend=True,  # Show the legend for cluster labels
    template='plotly_dark'  # Apply dark theme template
)

# Create the figure object and plot it
fig = go.Figure(data=data, layout=layout)
fig.show()
```



#####

Made with by Zahid Salim Shaikh

[ ]:

This notebook was converted with convert.ploomber.io