assignment-5-3

October 5, 2023

1 Kaggle Connection & DataFrame setup

[601]:
```
!pip install -q kaggle
```

[602]:
```
!mkdir ~/.kaggle
```

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

[603]:
```
!cp kaggle.json ~/.kaggle
```

[604]:
```
! kaggle datasets download -d vjchoudhary7/
↪customer-segmentation-tutorial-in-python
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Downloading customer-segmentation-tutorial-in-python.zip to /content
  0% 0.00/1.55k [00:00<?, ?B/s]
100% 1.55k/1.55k [00:00<00:00, 2.51MB/s]
```

[605]:
```
!unzip /content/customer-segmentation-tutorial-in-python.zip
```

```
Archive:  /content/customer-segmentation-tutorial-in-python.zip
  inflating: Mall_Customers.csv
```

2 Pre-Processing

[606]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[607]:
```
df = pd.read_csv('./Mall_Customers.csv')
df.head()
```

[607]:

	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
```

[608]:
```
df.describe()
```

[608]:
```
       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
```

[609]:
```
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
```

[610]:
```
df.isnull().values.any()
```

[610]:
```
False
```

[611]:
```
df.shape
```

[611]:
```
(200, 5)
```

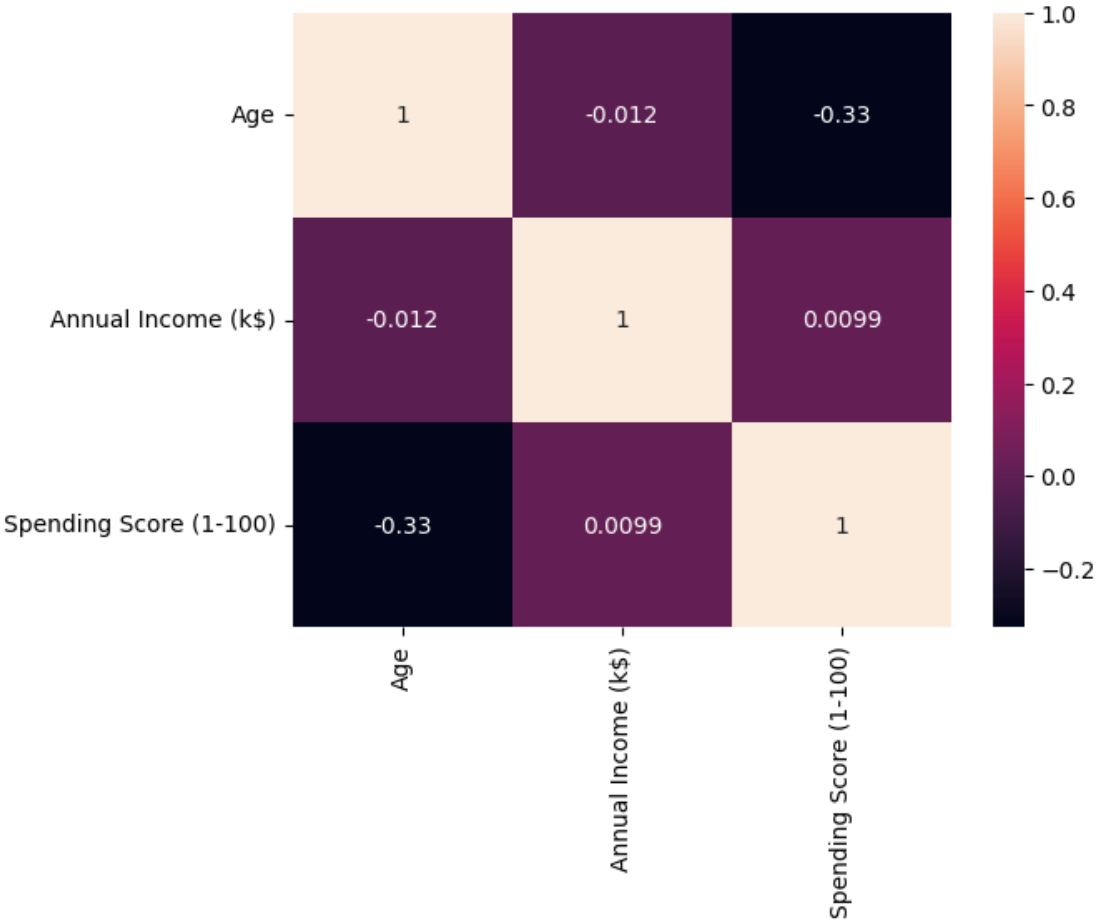
[612]:
```
# Dropping 'CustomerID' as it has no impact or connection to dataset or data␣
  ↪values
df.drop(['CustomerID'], axis=1, inplace=True)
```

[613]:
```
sns.heatmap(df.corr(), annot=True)
```

```
<ipython-input-613-6dc1c4c1753e>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
```

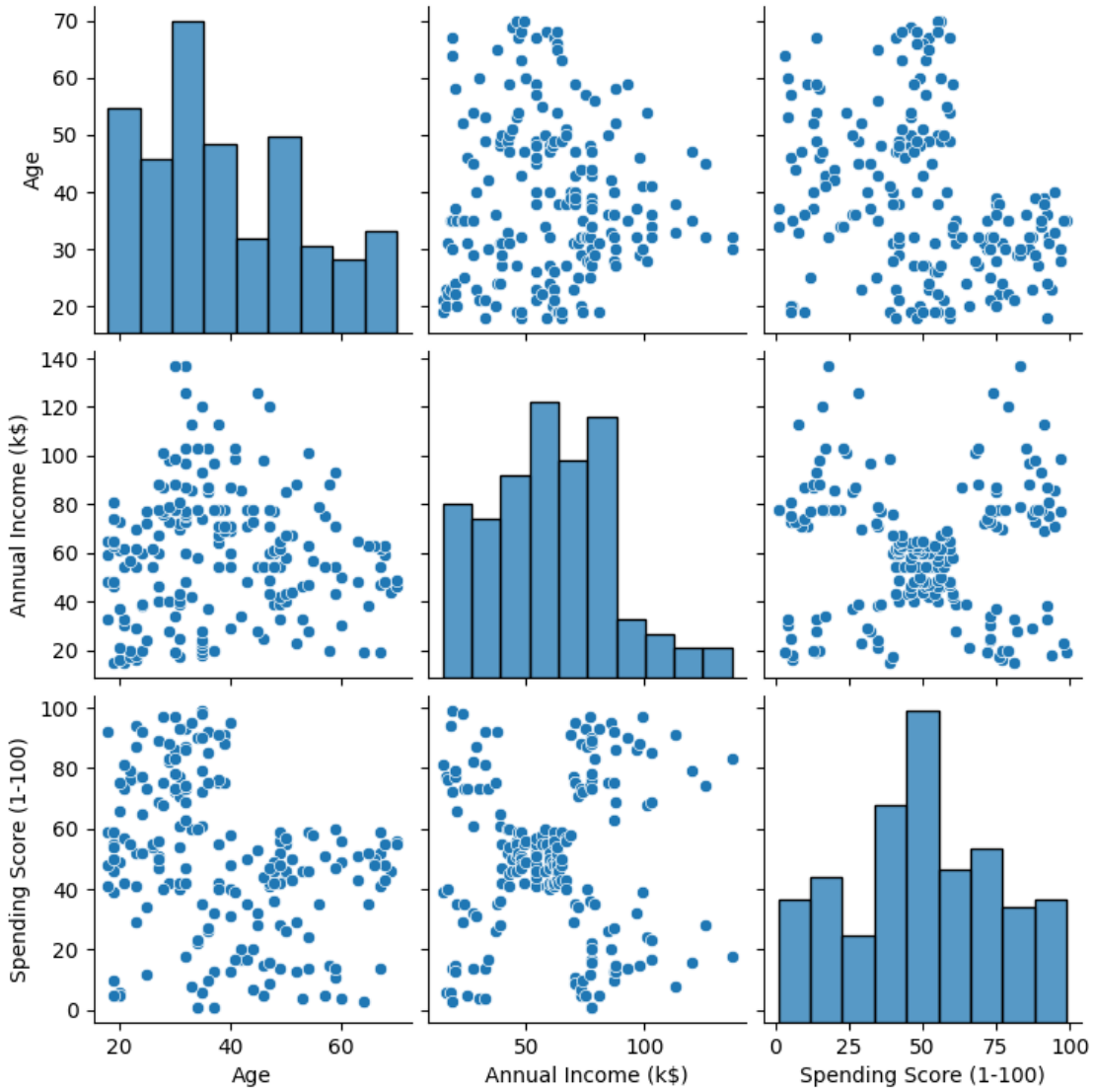
```
to silence this warning.
  sns.heatmap(df.corr(), annot=True)
```

[613]: <Axes: >



[614]:
```
sns.pairplot(df)
```

[614]: <seaborn.axisgrid.PairGrid at 0x7daddca476a0>



3 Converting Categorical Data (Columns) to Numerical

[615]:
```
df['Gender'].value_counts()
```

[615]:
```
Female    112
Male       88
Name: Gender, dtype: int64
```

[616]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

[617]:
```
# Label Encoding 'Gender' column
# '1' == 'Male' && 0 == 'Female'
df['Gender'] = le.fit_transform(df.Gender)
```

[618]:
```
df.head()
```

[618]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	19	15	39
1	1	21	15	81
2	0	20	16	6
3	0	23	16	77
4	0	31	17	40

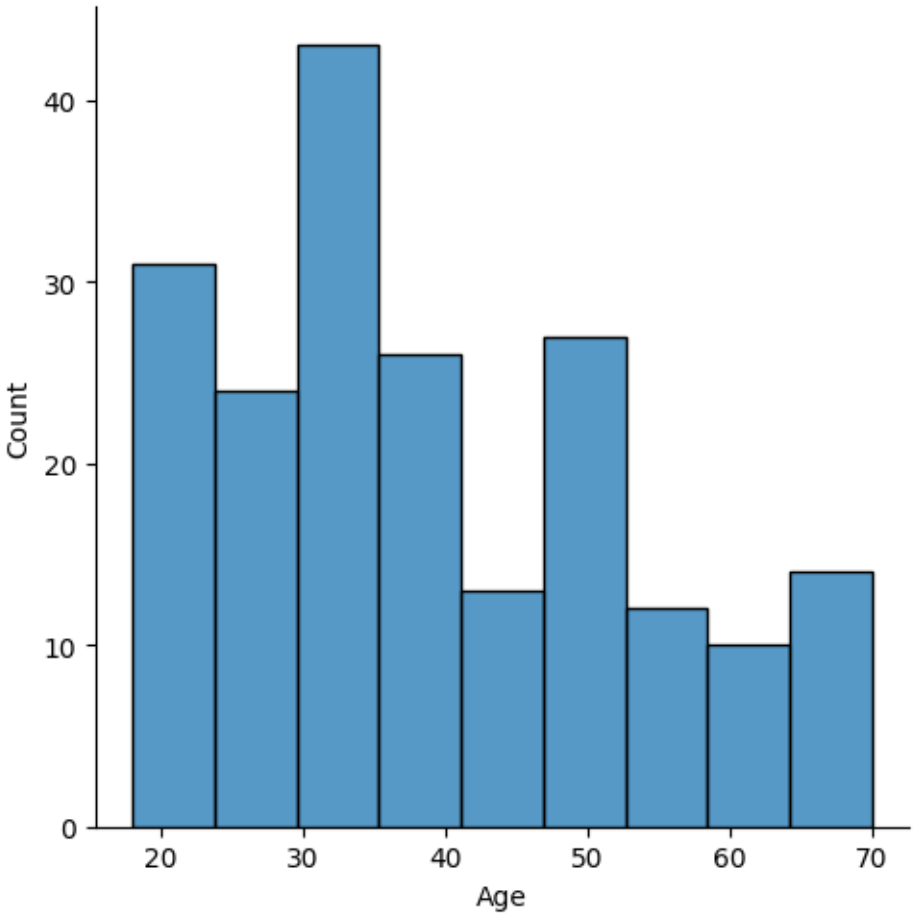
[619]:
```
df.info()
```

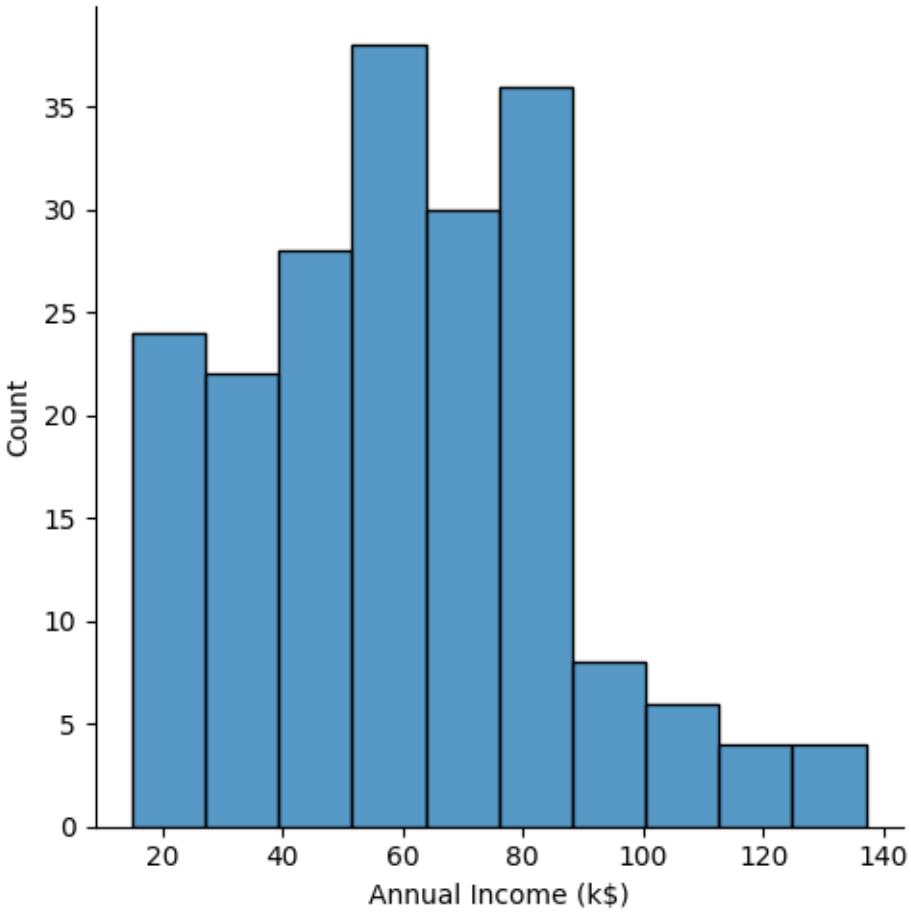
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Gender                  200 non-null    int64
 1   Age                     200 non-null    int64
 2   Annual Income (k$)      200 non-null    int64
 3   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4)
memory usage: 6.4 KB
```

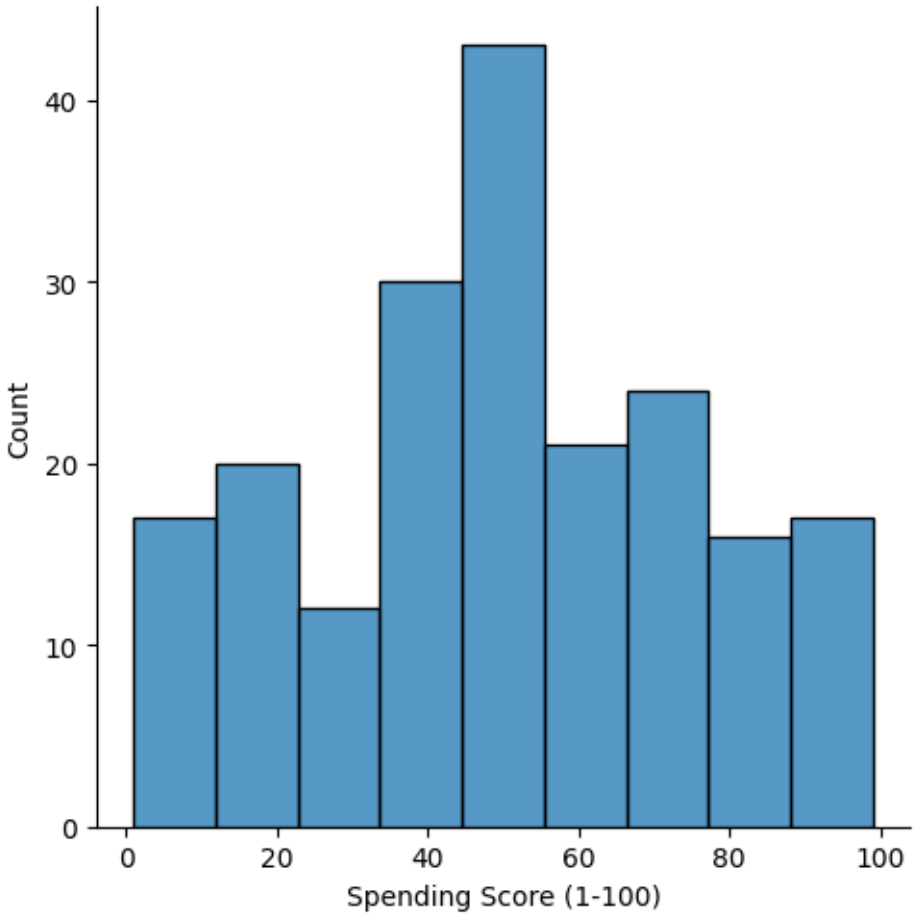
4 Data Analysis, Outlier Detection & Outlier Elimination

[620]:
```
sns.displot(df['Age'])
sns.displot(df['Annual Income (k$)'])
sns.displot(df['Spending Score (1-100)'])
```

[620]:
```
<seaborn.axisgrid.FacetGrid at 0x7daddba3dae0>
```


Count
0
10
20
30
40
20
30
40
50
60
70
Age

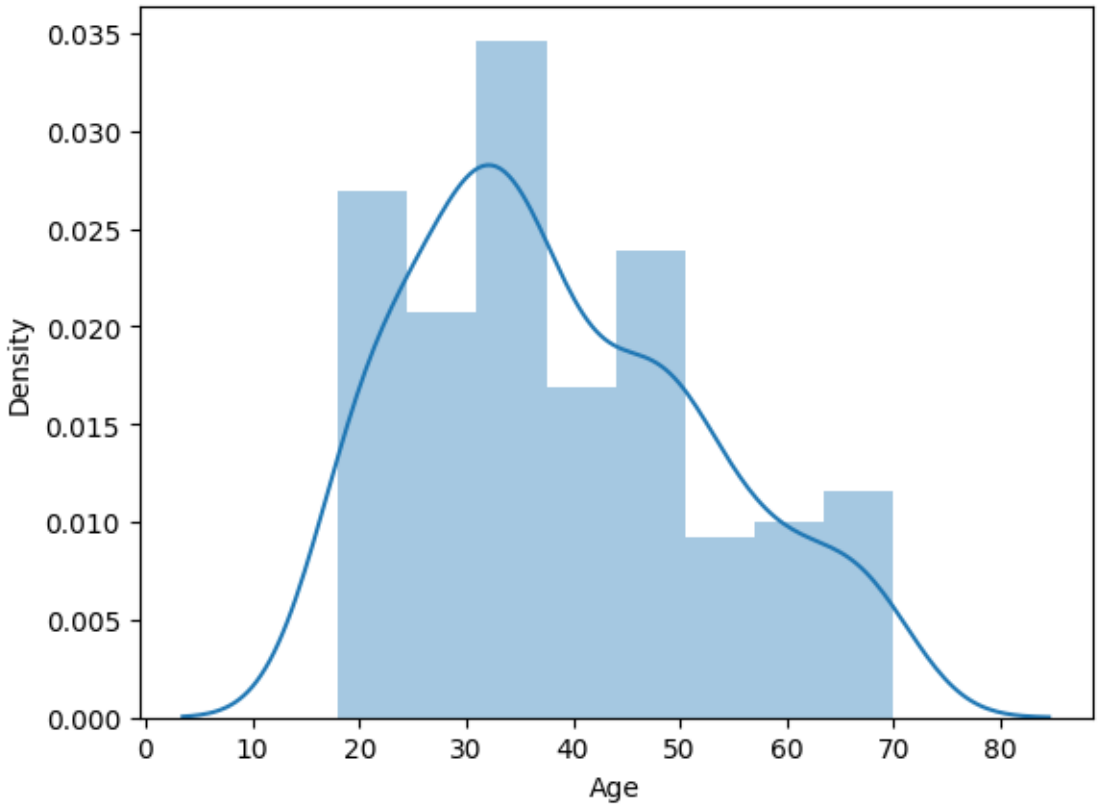

Count
0
5
10
15
20
25
30
35
20
40
60
80
100
120
140
Annual Income (k$)



[621]:
```
sns.distplot(df['Age'])
```

```
<ipython-input-621-0fafe04ea3f6>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Age'])
```

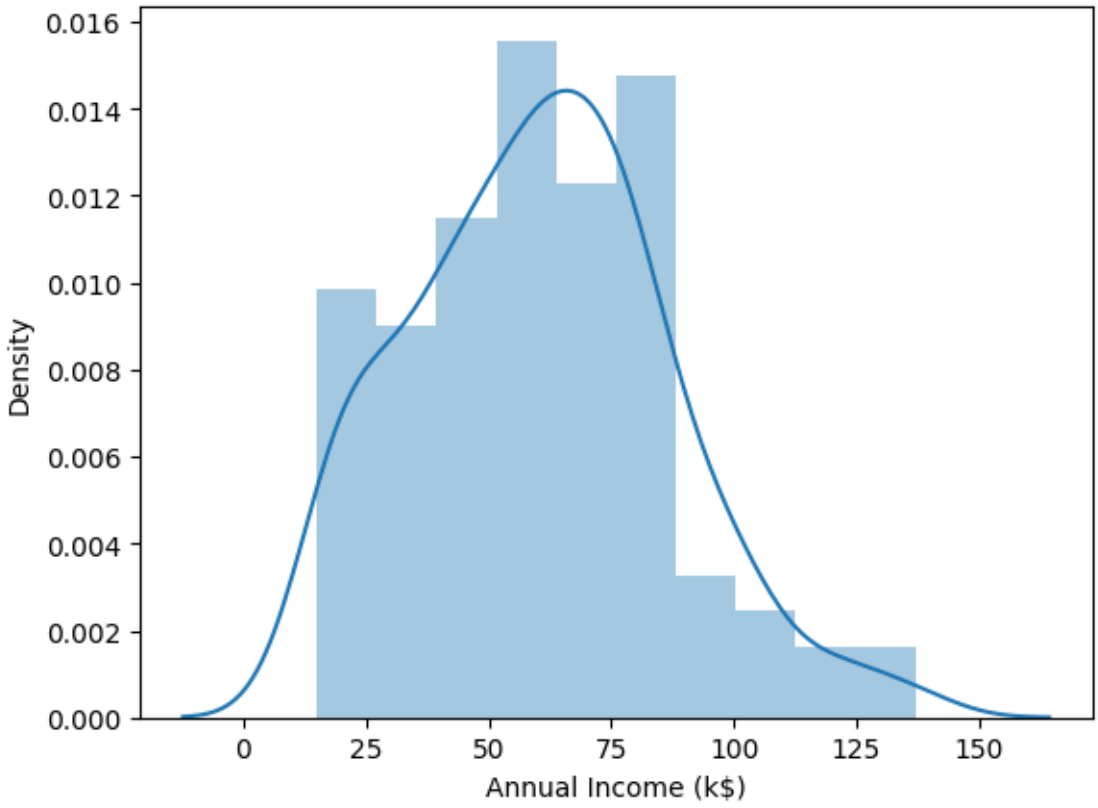
[621]: `<Axes: xlabel='Age', ylabel='Density'>`



[622]:
```
sns.distplot(df['Annual Income (k$)'])
```

```
<ipython-input-622-5c9bfeb4bab1>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Annual Income (k$)'])
```

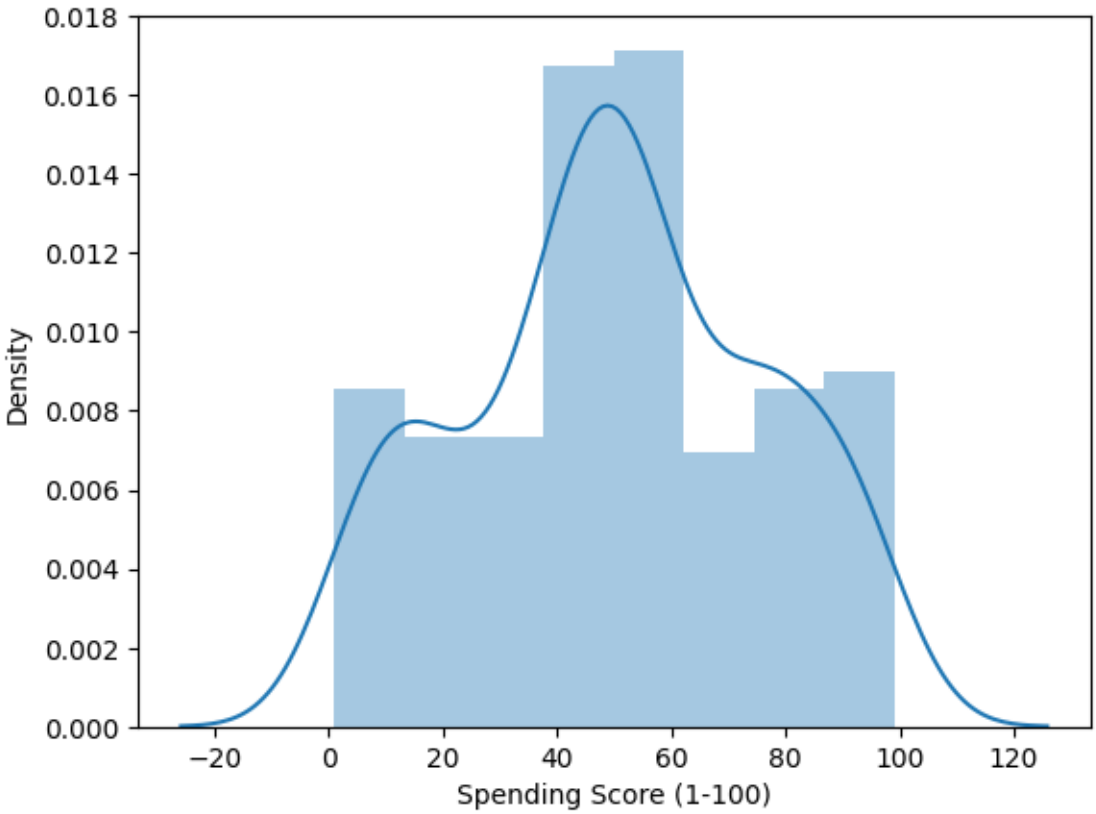
[622]: `<Axes: xlabel='Annual Income (k$)', ylabel='Density'>`



[623]:
```
sns.distplot(df['Spending Score (1-100)'])
```

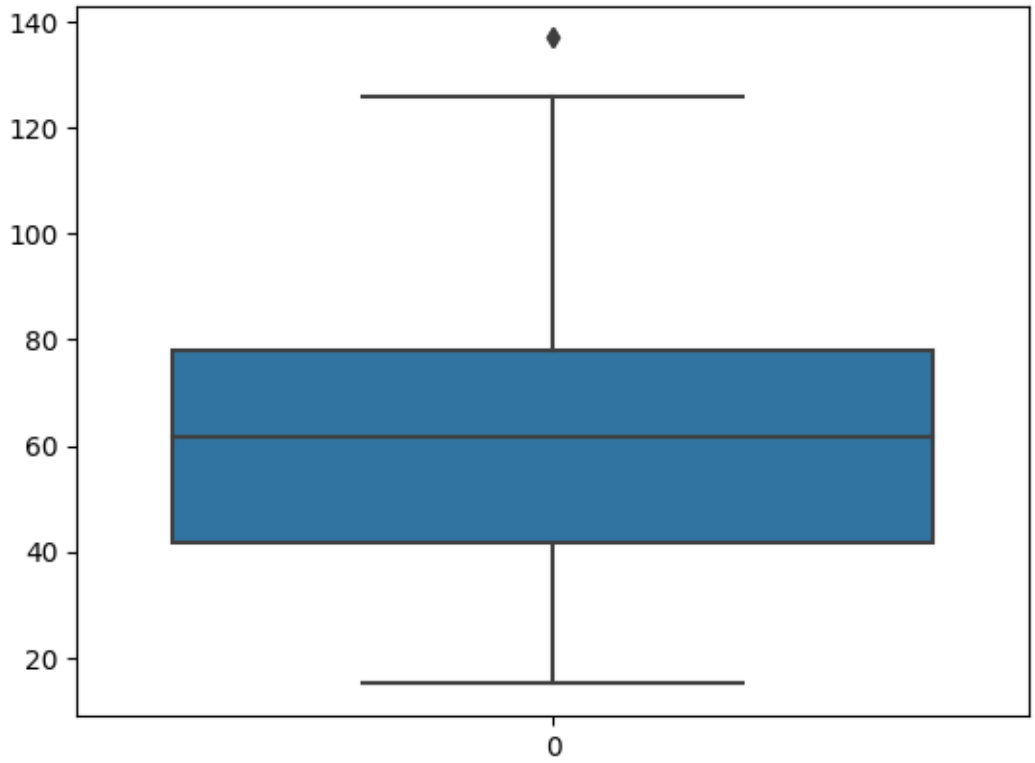
```
<ipython-input-623-beed7b40d5ab>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Spending Score (1-100)'])
```

[623]: `<Axes: xlabel='Spending Score (1-100)', ylabel='Density'>`



[624]:
```
sns.boxplot(df['Annual Income (k$)'])
```

[624]: <Axes: >



[625]:
```
Q1 = df['Annual Income (k$)'].quantile(0.25)
Q3 = df['Annual Income (k$)'].quantile(0.75)
```

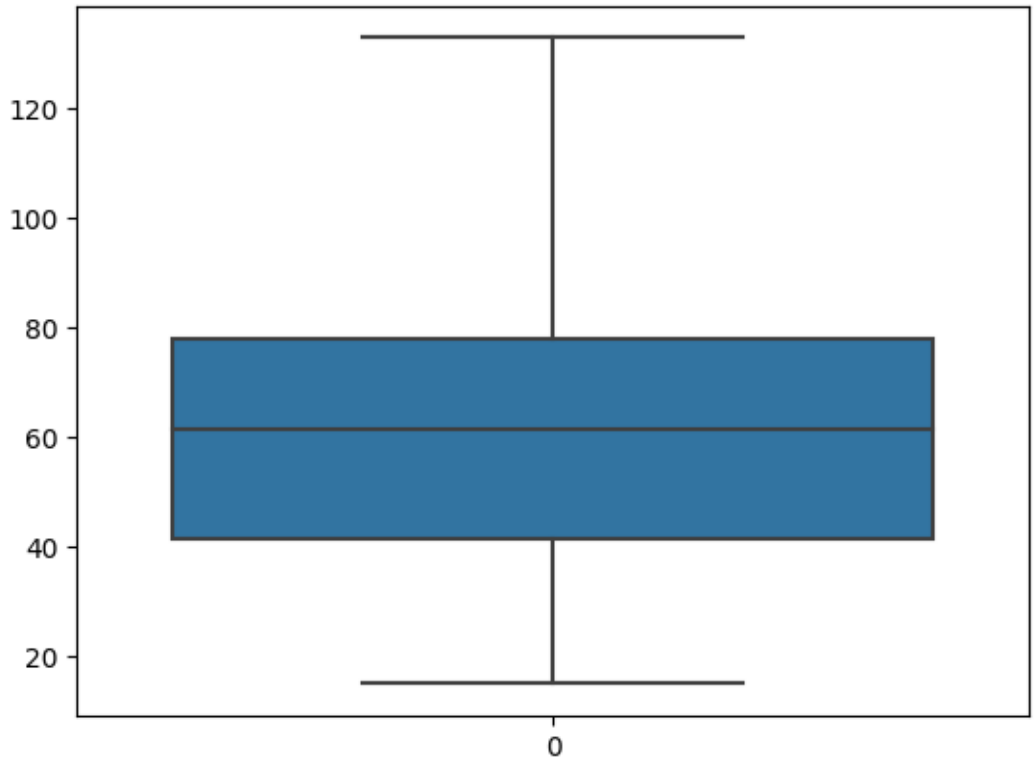
[626]:
```
IQR = Q3 - Q1
whisker_width = 1.5
```

[627]:
```
lower_whisker = Q1 - (whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
```

[628]:
```
df['Annual Income (k$)'] = np.where(df['Annual Income (k$)'] > upper_whisker,␣
 ↪upper_whisker, np.where(df['Annual Income (k$)'] < lower_whisker,␣
 ↪lower_whisker, df['Annual Income (k$)']))
```

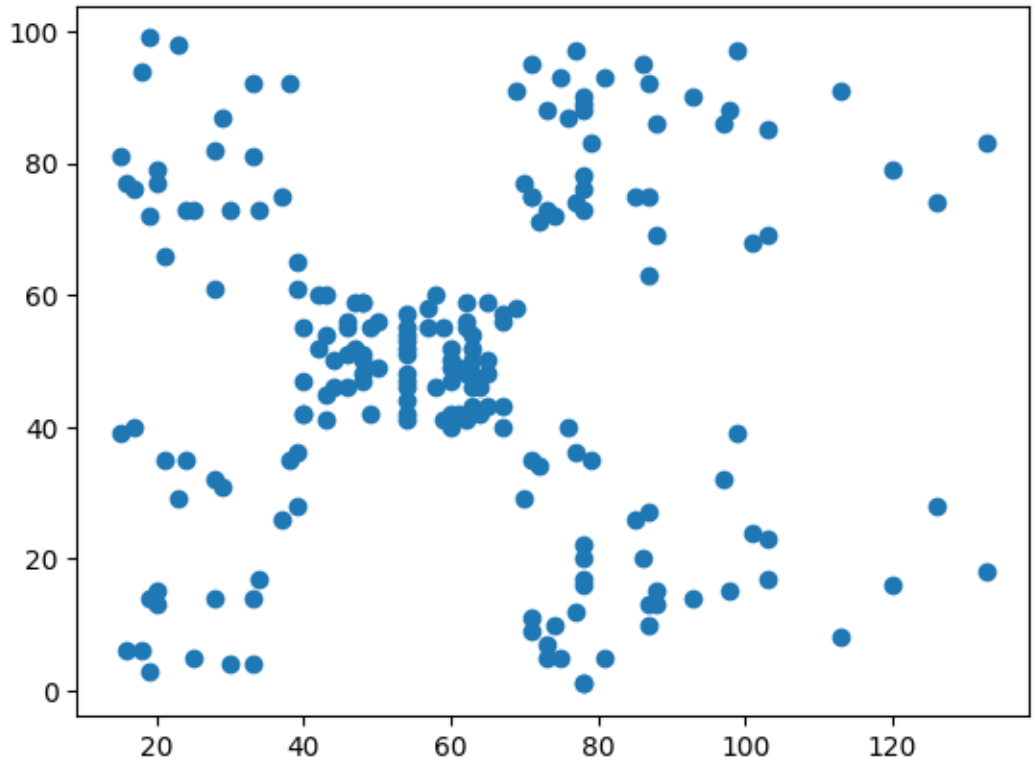
[629]:
```
sns.boxplot(df['Annual Income (k$)'])
```

[629]: <Axes: >



[630]:
```
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'])
```

[630]: <matplotlib.collections.PathCollection at 0x7daddb65d7b0>



[631]:
```
from sklearn.preprocessing import MinMaxScaler
```

[632]:
```
X_train = df.drop(['Spending Score (1-100)'], axis=1)
X_train = pd.DataFrame(MinMaxScaler().fit_transform(X_train), columns=X_train.
 ↪columns)

Y_train = df['Spending Score (1-100)']
```

[633]:
```
X_train.head(), Y_train.head()
```

[633]:
```
(   Gender       Age  Annual Income (k$)
 0     1.0  0.019231            0.000000
 1     1.0  0.057692            0.000000
 2     0.0  0.038462            0.008493
 3     0.0  0.096154            0.008493
 4     0.0  0.250000            0.016985,
 0    39
 1    81
 2     6
 3    77
 4    40
 Name: Spending Score (1-100), dtype: int64)
```

5 Finding Elbow Point (Possible 'K' value)

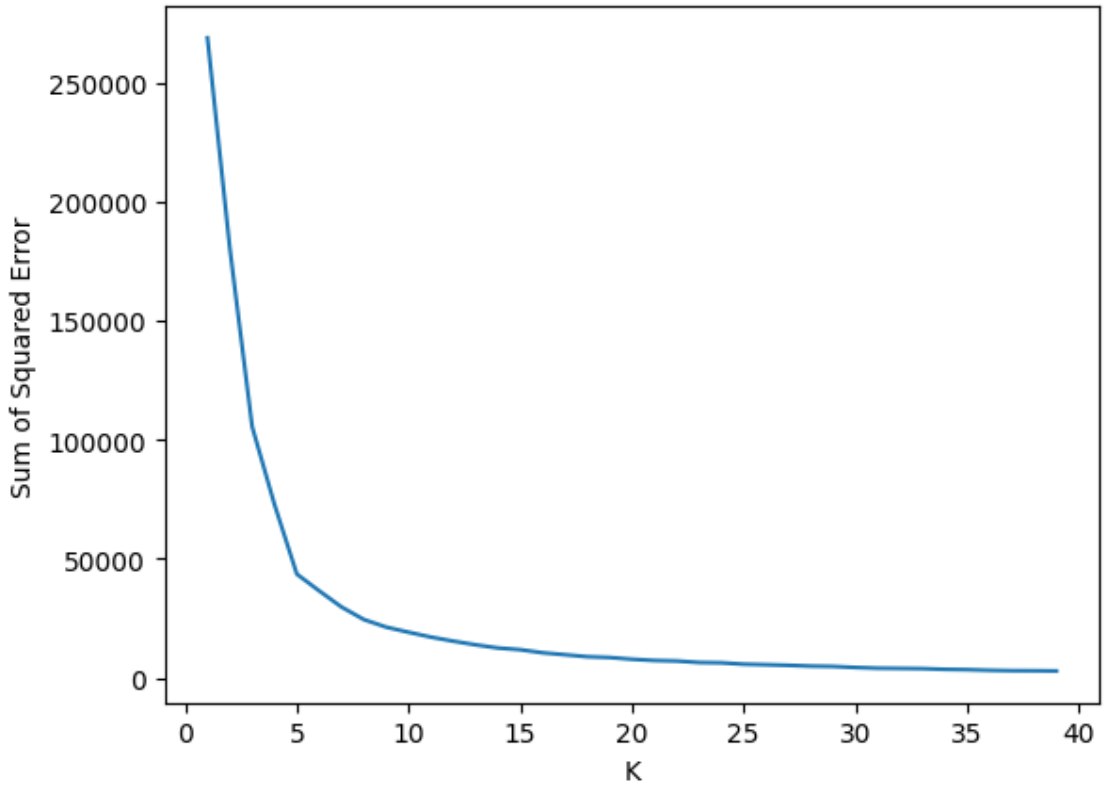
[634]:
```
from sklearn.cluster import KMeans
```

[ ]:
```
k_rng = range(1,40)
sse = []

for k in k_rng:
  km = KMeans(n_clusters=k)
  km.fit(df[['Annual Income (k$)', 'Spending Score (1-100)']])
  sse.append(km.inertia_)
```

[ ]:
```
sse
```

[637]:
```
plt.xlabel('K')
plt.ylabel('Sum of Squared Error')
plt.plot(k_rng, sse)
```

[637]: [<matplotlib.lines.Line2D at 0x7daddb6e1540>]



[638]:
```
kmeans = KMeans(n_clusters=5)
kmeans
```

[638]:
```
KMeans(n_clusters=5)
```

[639]:
```
z = kmeans.fit_predict(df[['Annual Income (k$)','Spending Score (1-100)']])
z
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[639]:
```
array([3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 4,
       3, 1, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 0, 2, 4, 2, 0, 2, 0, 2,
       4, 2, 0, 2, 0, 2, 0, 2, 0, 2, 4, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int32)
```

[640]:
```
df_2 = df
```

[641]:
```
df_2['Cluster'] = z
df_2.head()
```

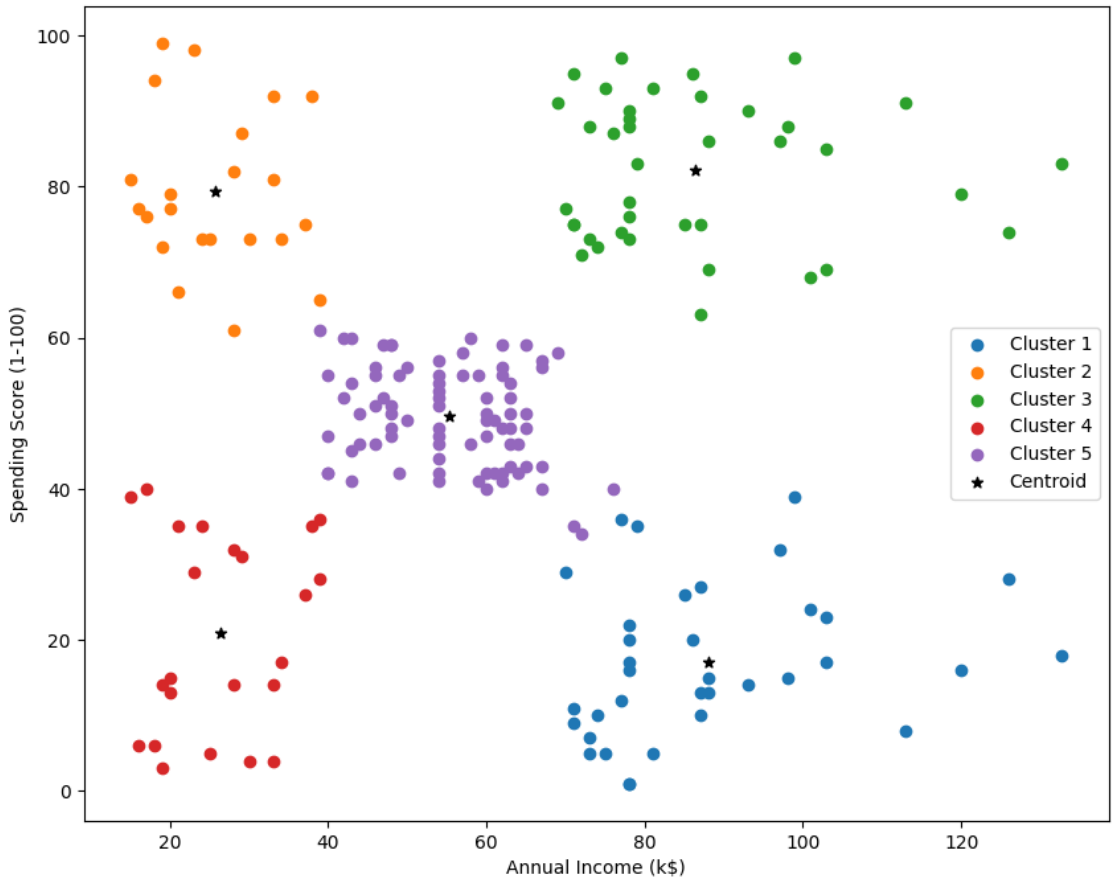
[641]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	19	15.0	39	3
1	1	21	15.0	81	1
2	0	20	16.0	6	3
3	0	23	16.0	77	1
4	0	31	17.0	40	3

[642]:
```
plt.figure(figsize=(10,8))

df1 = df_2[df_2.Cluster==0]
df2 = df_2[df_2.Cluster==1]
df3 = df_2[df_2.Cluster==2]
df4 = df_2[df_2.Cluster==3]
df5 = df_2[df_2.Cluster==4]


plt.scatter(df1['Annual Income (k$)'], df1['Spending Score (1-100)'],␣
 ↪label='Cluster 1')
```

```
plt.scatter(df2['Annual Income (k$)'], df2['Spending Score (1-100)'],␣
 ↪label='Cluster 2')
plt.scatter(df3['Annual Income (k$)'], df3['Spending Score (1-100)'],␣
 ↪label='Cluster 3')
plt.scatter(df4['Annual Income (k$)'], df4['Spending Score (1-100)'],␣
 ↪label='Cluster 4')
plt.scatter(df5['Annual Income (k$)'], df5['Spending Score (1-100)'],␣
 ↪label='Cluster 5')

plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],␣
 ↪color='black', marker='*', label='Centroid')

plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
```

[642]: <matplotlib.legend.Legend at 0x7daddb701b10>



6 Scaling & Train - Test Split

[643]:
```
from sklearn.model_selection import train_test_split
```

[664]:
```
X = df.drop(['Spending Score (1-100)'], axis=1)
X = pd.DataFrame(MinMaxScaler().fit_transform(X), columns=X.columns)
X.drop(['Cluster', 'Gender'], axis=1, inplace=True)
```

[665]:
```
y = df['Spending Score (1-100)']
```

[666]:
```
X.head(), y.head()
```

[666]:
```
(        Age  Annual Income (k$)
 0  0.019231            0.000000
 1  0.057692            0.000000
 2  0.038462            0.008493
 3  0.096154            0.008493
 4  0.250000            0.016985,
 0    39
 1    81
 2     6
 3    77
 4    40
 Name: Spending Score (1-100), dtype: int64)
```

7 KNN and Logistic Regression Modeling

[667]:
```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
```

[668]:
```
lr = LogisticRegression(max_iter=10000)
knn = KNeighborsRegressor(n_neighbors=3)
```

7.1 KNN Model

[669]:
```
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.25,␣
↪random_state=100)
```

[670]:
```
xtest.shape, xtrain.shape
```

[670]:
```
((50, 2), (150, 2))
```

[671]:
```
ytest.shape, ytrain.shape
```

[671]:
```
((50,), (150,))
```

[697]:
```
knn.fit(xtrain.values, ytrain.values)
```

[697]:
```
KNeighborsRegressor(n_neighbors=3)
```

[698]:
```
acc = knn.score(xtest.values, ytest.values)
print(f"Accuracy for the KNN model is {acc*100:.2f}%")
```

```
Accuracy for the KNN model is 56.86%
```

7.2 Logistic Regression Model

[764]:
```
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, y, test_size=0.25,␣
↪random_state=25)
```

[765]:
```
Xtest.shape, Xtrain.shape
```

[765]:
```
((50, 2), (150, 2))
```

[766]:
```
Ytest.shape, Ytrain.shape
```

[766]:
```
((50,), (150,))
```

[767]:
```
lr.fit(Xtrain.values, Ytrain.values)
```

[767]:
```
LogisticRegression(max_iter=10000)
```

[768]:
```
acc = lr.score(Xtest.values, Ytest.values)
print(f"Accuracy for the Logistic Regression model is {acc*100:.2f}%")
```

```
Accuracy for the Logistic Regression model is 8.00%
```

8 Prediction

[777]:
```
prediction1 = knn.predict([[36 , 24]])[0]
prediction2 = lr.predict([[36 , 24]])[0]
```

[778]:
```
print("KNN Model")
print(f"Age: 36, Salary(k$): 24.0, Spending Score(1-100): {prediction1:.2f}")

print('\n', "*"*10, '\n')

print("Logistic Regression Model")
print(f"Age: 36, Salary(k$): 24.0, Spending Score(1-100): {prediction2:.2f}")
```

```
KNN Model
Age: 36, Salary(k$): 24.0, Spending Score(1-100): 37.33
```

```
 **********

Logistic Regression Model
Age: 36, Salary(k$): 24.0, Spending Score(1-100): 56.00
```