

Assignment - 4

Python Programming

Assignment Date	15 Nov 2022
Project Title	Efficient Water Quality Analysis & Prediction using Machine Learning
Team ID	PNT2022TMID15978
Maximum Marks	2 Marks

Problem Statement: Customer Segmentation Analysis

Importing Modules

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

1. Download the dataset
2. Load the dataset into the tool

```
In [2]: data=pd.read_csv("Mall_Customers.csv")
```

3. Perform Below Visualizations

- Univariate Analysis
- Bi-Variate Analysis
- Multi-Variate Analysis

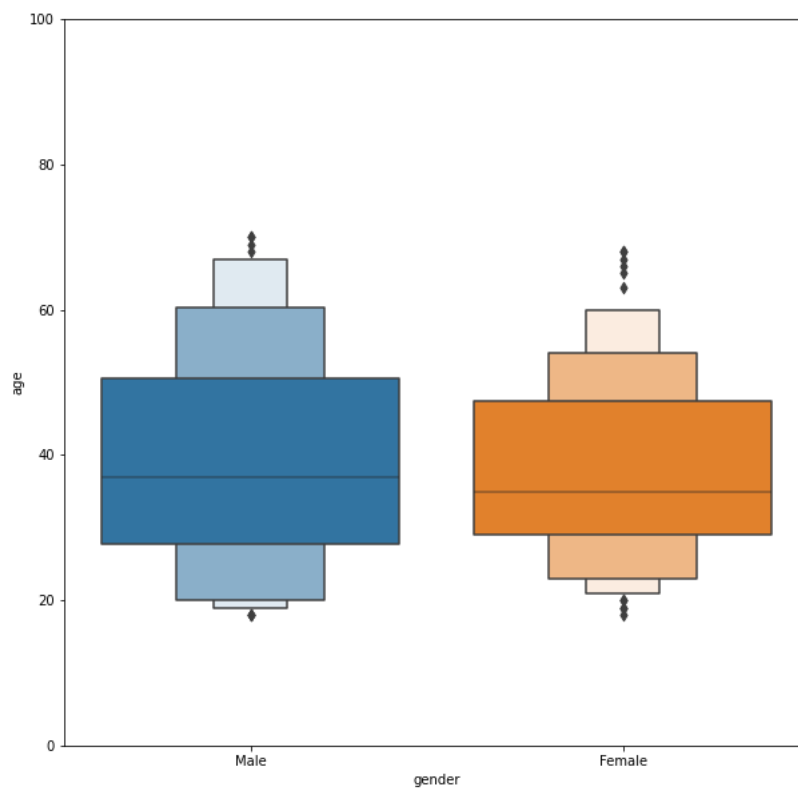
```
In [3]: data.head()
```

```
Out[3]:
```

	CustomerID	Genre	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 [8]: data.rename(columns={"CustomerID": "customer_id", "Genre": "gender", "Age": "age", "Annual Income (k$)": "annual_income",  
                             "Spending Score (1-100)": "spending_scores"}, inplace=True)
```

```
In [9]: temp = pd.concat([data['age'], data['gender']], axis=1)  
  
f, ax = plt.subplots(figsize=(10,10))  
fig = sns.boxenplot(x='gender', y="age", data=temp)  
fig.axis(ymin=0, ymax=100);
```



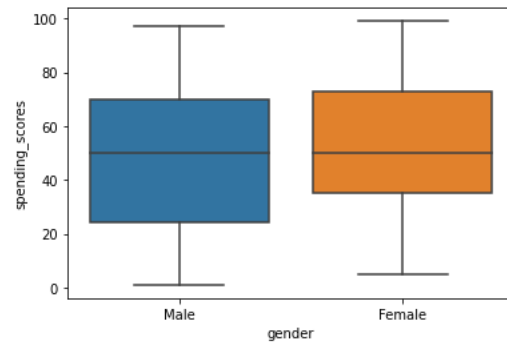
ANALYSIS

There is no difference in age of rings for male and female (18-70).

Count plot

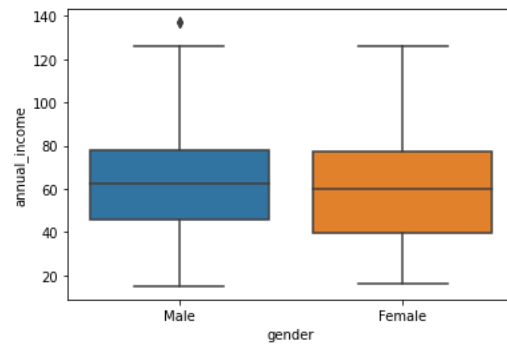
```
In [10]: sns.boxplot(x=data['gender'],y=data['spending_scores'])
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4677c22dd0>
```



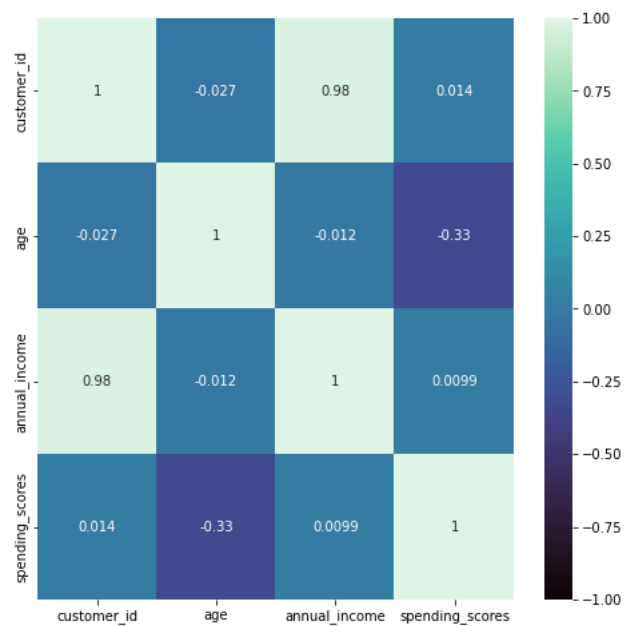
```
In [11]: sns.boxplot(x=data['gender'],y=data['annual_income'])
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4674ea2f50>
```



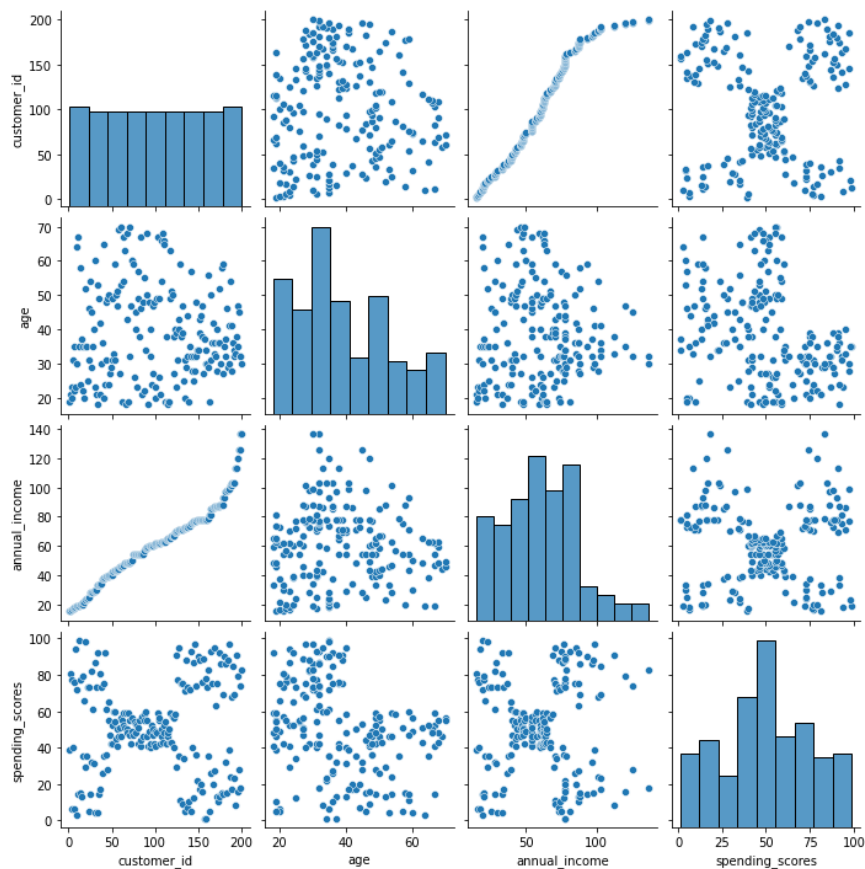
Coorelation Plot

```
In [12]: corr=data.corr()  
plt.figure(figsize=(8,8))  
sns.heatmap(corr,vmin=-1,center=0, annot = True, cmap = 'mako')
```



```
In [13]: sns.pairplot(data)
```

```
Out[13]: <seaborn.axisgrid.PairGrid at 0x7f4674cafe90>
```



4. Perform descriptive statistics on the dataset.

```
In [14]: data.head(10)
```

```
Out[14]:
```

	customer_id	gender	age	annual_income	spending_scores
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
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72

```
In [15]: data.shape
```

```
Out[15]: (200, 5)
```

```
In [16]: data.describe()
```

```
Out[16]:
```

	customer_id	age	annual_income	spending_scores
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

```
In [17]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   customer_id      200 non-null   int64
1   gender           200 non-null   object
2   age              200 non-null   int64
3   annual_income    200 non-null   int64
4   spending_scores  200 non-null   int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

5. Check for Missing values and deal with them

```
In [18]: data[data.duplicated()]
```

```
Out[18]:
```

customer_id	gender	age	annual_income	spending_scores
-------------	--------	-----	---------------	-----------------

there is no missing values and duplicates in dataframe

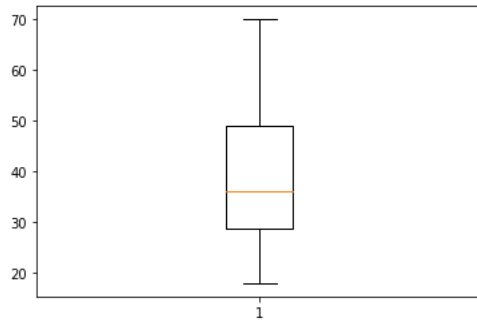
6. Find the outliers and replace them

```
In [19]: for i in data:
          if data[i].dtype=='int64':
              q1=data[i].quantile(0.25)
              q3=data[i].quantile(0.75)
              iqr=q3-q1
              upper=q3+1.5*iqr
              lower=q1-1.5*iqr
              data[i]=np.where(data[i] >upper, upper, data[i])
              data[i]=np.where(data[i] <lower, lower, data[i])
```

After removing outliers, boxplot will be like

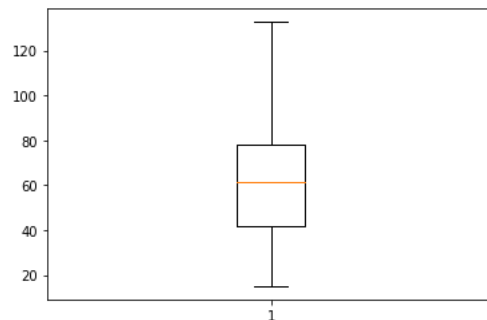
```
In [20]: plt.boxplot(data['age'])
```

```
Out[20]: {'whiskers': [<matplotlib.lines.Line2D at 0x7f467451fb10>,  
<matplotlib.lines.Line2D at 0x7f4674523090>],  
'caps': [<matplotlib.lines.Line2D at 0x7f46745235d0>,  
<matplotlib.lines.Line2D at 0x7f4674523b10>],  
'boxes': [<matplotlib.lines.Line2D at 0x7f467451f550>],  
'medians': [<matplotlib.lines.Line2D at 0x7f4672ca50d0>],  
'fliers': [<matplotlib.lines.Line2D at 0x7f4672ca5610>],  
'means': []}
```



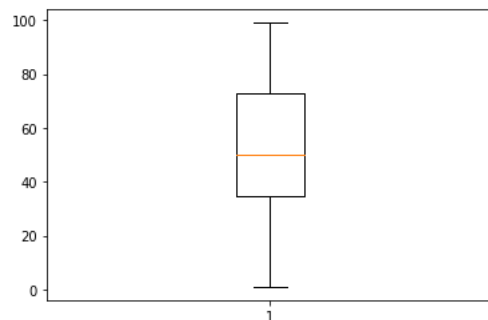
```
In [21]: plt.boxplot(data['annual_income'])
```

```
Out[21]: {'whiskers': [<matplotlib.lines.Line2D at 0x7f4672c7f650>,  
<matplotlib.lines.Line2D at 0x7f4672c7fb90>],  
'caps': [<matplotlib.lines.Line2D at 0x7f4672c86110>,  
<matplotlib.lines.Line2D at 0x7f4672c86650>],  
'boxes': [<matplotlib.lines.Line2D at 0x7f4672c7f090>],  
'medians': [<matplotlib.lines.Line2D at 0x7f4672c86bd0>],  
'fliers': [<matplotlib.lines.Line2D at 0x7f4672c8e150>],  
'means': []}
```



```
In [22]: plt.boxplot(data['spending_scores'])
```

```
Out[22]: {'whiskers': [<matplotlib.lines.Line2D at 0x7f4672bed090>,  
<matplotlib.lines.Line2D at 0x7f4672bed5d0>],  
'caps': [<matplotlib.lines.Line2D at 0x7f4672bedb10>,  
<matplotlib.lines.Line2D at 0x7f4672bf4090>],  
'boxes': [<matplotlib.lines.Line2D at 0x7f4672beaa90>],  
'medians': [<matplotlib.lines.Line2D at 0x7f4672bf4610>],  
'fliers': [<matplotlib.lines.Line2D at 0x7f4672bf4b50>],  
'means': []}
```



7. Check for categorical columns and perform encoding

```
In [23]: from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
data['gender']=encoder.fit_transform(data['gender'])
```

```
In [24]: data.head()
```

```
Out[24]:
```

	customer_id	gender	age	annual_income	spending_scores
0	1.0	1	19.0	15.0	39.0
1	2.0	1	21.0	15.0	81.0
2	3.0	0	20.0	16.0	6.0
3	4.0	0	23.0	16.0	77.0
4	5.0	0	31.0	17.0	40.0

8. Scaling the data

```
In [25]: from sklearn.preprocessing import StandardScaler
df=StandardScaler()
data1=df.fit_transform(data)
```

```
In [26]: data1
```

```
Out[26]: array([[ -1.7234121,  1.12815215, -1.42456879, -1.74542941, -0.43480148],
 [ -1.70609137,  1.12815215, -1.28103541, -1.74542941,  1.19570407],
 [ -1.68877065, -0.88640526, -1.3528021 , -1.70708307, -1.71591298],
 [ -1.67144992, -0.88640526, -1.13750203, -1.70708307,  1.04041783],
 [ -1.6541292 , -0.88640526, -0.56336851, -1.66873673, -0.39597992],
 [ -1.63680847, -0.88640526, -1.20926872, -1.66873673,  1.00159627],
 [ -1.61948775, -0.88640526, -0.27630176, -1.6303904 , -1.71591298],
 [ -1.60216702, -0.88640526, -1.13750203, -1.6303904 ,  1.70038436],
 [ -1.5848463 ,  1.12815215,  1.80493225, -1.59204406, -1.83237767],
 [ -1.56752558, -0.88640526, -0.6351352 , -1.59204406,  0.84631002],
 [ -1.55020485,  1.12815215,  2.02023231, -1.59204406, -1.4053405 ],
 [ -1.53288413, -0.88640526, -0.27630176, -1.59204406,  1.89449216],
 [ -1.5155634 , -0.88640526,  1.37433211, -1.55369772, -1.36651894],
 [ -1.49824268, -0.88640526, -1.06573534, -1.55369772,  1.04041783],
 [ -1.48092195,  1.12815215, -0.13276838, -1.55369772, -1.44416206],
 [ -1.46360123,  1.12815215, -1.20926872, -1.55369772,  1.11806095],
 [ -1.4462805 , -0.88640526, -0.27630176, -1.51535138, -0.59008772],
 [ -1.42895978,  1.12815215, -1.3528021 , -1.51535138,  0.61338066],
 [ -1.41163905,  1.12815215,  0.94373197, -1.43865871, -0.82301709],
 [ -1.39431833, -0.88640526, -0.27630176, -1.43865871,  1.8556706 ],
 [ -1.3769976 ,  1.12815215, -0.27630176, -1.40031237, -0.59008772],
 [ -1.35967688,  1.12815215, -0.99396865, -1.40031237,  0.88513158],
 [ -1.34235616, -0.88640526,  0.51313183, -1.36196603, -1.75473454],
 [ -1.32503543,  1.12815215, -0.56336851, -1.36196603,  0.88513158],
 [ -1.30771471, -0.88640526,  1.08726535, -1.24692702, -1.4053405 ],
 [ -1.29039398,  1.12815215, -0.70690189, -1.24692702,  1.23452563],
 [ -1.27307326, -0.88640526,  0.44136514, -1.24692702, -0.7065524 ],
 [ -1.25575253,  1.12815215, -0.27630176, -1.24692702,  0.41927286],
 [ -1.23843181, -0.88640526,  0.08253169, -1.20858069, -0.74537397],
 [ -1.22111108, -0.88640526, -1.13750203, -1.20858069,  1.42863343],
 [ -1.20379036,  1.12815215,  1.51786549, -1.17023435, -1.7935561 ],
 [ -1.18646963, -0.88640526, -1.28103541, -1.17023435,  0.88513158],
 [ -1.16914891,  1.12815215,  1.01549866, -1.05519534, -1.7935561 ],
 [ -1.15182818,  1.12815215, -1.49633548, -1.05519534,  1.62274124],
 [ -1.13450746, -0.88640526,  0.7284319 , -1.05519534, -1.4053405 ],
 [ -1.11718674, -0.88640526, -1.28103541, -1.05519534,  1.19570407],
 [ -1.09986601, -0.88640526,  0.22606507, -1.016849 , -1.28887582],
 [ -1.08254529, -0.88640526, -0.6351352 , -1.016849 ,  0.88513158],
 [ -1.06522456, -0.88640526, -0.20453507, -0.90180999, -0.93948177],
 [ -1.04790384, -0.88640526, -1.3528021 , -0.90180999,  0.96277471],
 [ -1.03058311, -0.88640526,  1.87669894, -0.86346365, -0.59008772],
 [ -1.01326239,  1.12815215, -1.06573534, -0.86346365,  1.62274124],
 [ -0.99594166,  1.12815215,  0.65666521, -0.82511731, -0.55126616],
 [ -0.97862094, -0.88640526, -0.56336851, -0.82511731,  0.41927286],
 [ -0.96130021, -0.88640526,  0.7284319 , -0.82511731, -0.86183865],
```

```
[ 1.5155634 ,  1.12815215, -0.7866858,  1.5523556 ,  0.69102378],
[ 1.53288413, -0.88640526,  0.15429838,  1.62904827, -1.28887582],
[ 1.55020485, -0.88640526, -0.20453507,  1.62904827,  1.35099031],
[ 1.56752558, -0.88640526, -0.34806844,  1.62904827, -1.05594645],
[ 1.5848463 , -0.88640526, -0.49160182,  1.62904827,  0.72984534],
[ 1.60216702,  1.12815215, -0.41983513,  2.01251165, -1.63826986],
[ 1.61948775, -0.88640526, -0.06100169,  2.01251165,  1.58391968],
[ 1.63680847, -0.88640526,  0.58489852,  2.28093601, -1.32769738],
[ 1.6541292 , -0.88640526, -0.27630176,  2.28093601,  1.11806095],
[ 1.67144992, -0.88640526,  0.44136514,  2.51101403, -0.86183865],
[ 1.68877065,  1.12815215, -0.49160182,  2.51101403,  0.92395314],
[ 1.70609137,  1.12815215, -0.49160182,  2.76985181, -1.25005425],
[ 1.7234121 ,  1.12815215, -0.6351352 ,  2.76985181,  1.27334719]]
```

9. Perform any of the clustering algorithms

```
In [27]: from sklearn.cluster import KMeans
```

```
In [28]: data.drop('customer_id',axis=1,inplace=True)
```

```
In [29]: km = KMeans(n_clusters=3, random_state=0)
```

```
In [30]: data['Group or Cluster'] = km.fit_predict(data)
```

```
In [31]: data.head()
```

```
Out[31]:
```

	gender	age	annual_income	spending_scores	Group or Cluster
0	1	19.0	15.0	39.0	2
1	1	21.0	15.0	81.0	2
2	0	20.0	16.0	6.0	2
3	0	23.0	16.0	77.0	2
4	0	31.0	17.0	40.0	2

```
In [32]: data['Group or Cluster'].value_counts()
```

```
Out[32]:
```

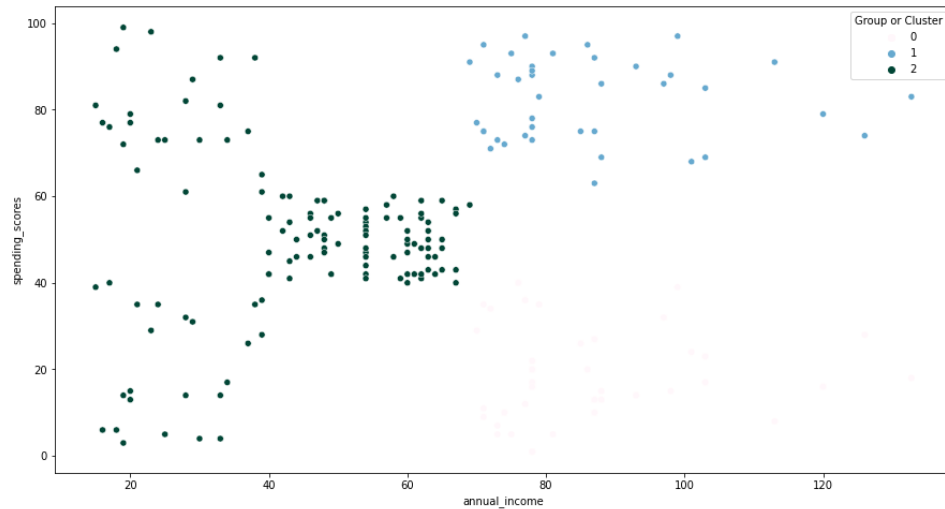
2	123
1	39
0	38

Name: Group or Cluster, dtype: int64


```
In [33]: import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(15,8))
sns.scatterplot(x=data['annual_income'],
                y=data['spending_scores'],
                hue=data['Group or Cluster'],
                palette='PuBuGn')

plt.show()
```



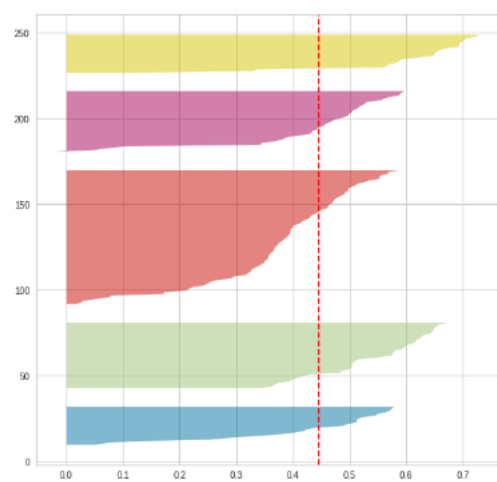
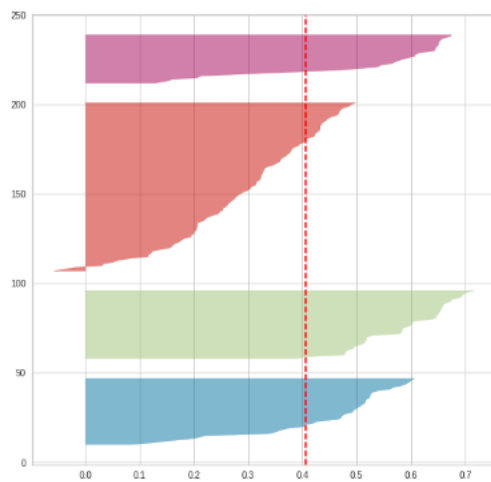
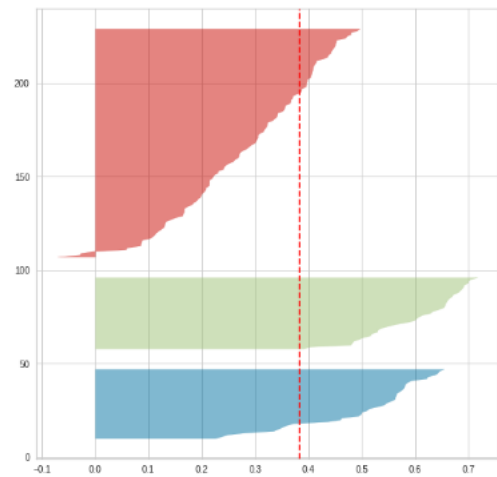
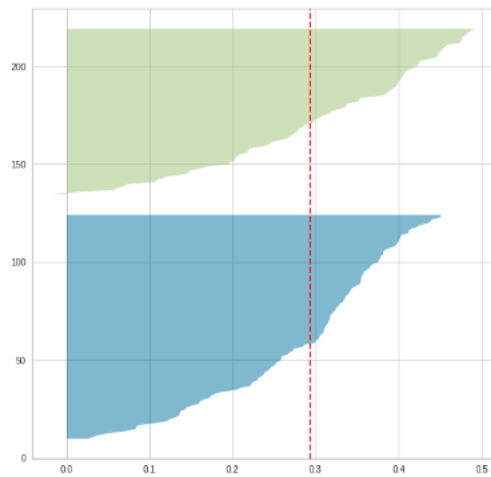
```
In [34]: from sklearn.metrics import silhouette_score, silhouette_samples
score = silhouette_score(data,
                        km.labels_,
                        metric='euclidean')

score
```

Out[34]: 0.3842057644019546

```
In [35]: import matplotlib.pyplot as plt
from yellowbrick.cluster import SilhouetteVisualizer

fig, ax = plt.subplots(2, 2, figsize=(20,20))
for i in [2, 3, 4, 5]:
    """
    Create KMeans instance for different number of clusters
    """
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=100,
                random_state=0)
    q, mod = divmod(i, 2)
    """
    Create SilhouetteVisualizer instance with KMeans instance
    Fit the visualizer
    """
    visualizer = SilhouetteVisualizer(km,
                                     colors='yellowbrick',
                                     ax=ax[q-1][mod])
    visualizer.fit(data)
```



10. Add the cluster to the primary dataset

```
df_transformed = df_initial
```

```
[ ] df_transformed['class'] = y_predicted
```

```
df_transformed
```

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	class
0	1	19.0	15.0	39	4
1	1	21.0	15.0	81	3
2	0	20.0	16.0	6	4
3	0	32.0	16.0	77	3
4	0	31.0	17.0	40	4
...
195	0	35.0	120.0	79	0
196	0	45.0	126.0	28	2
197	1	32.0	126.0	74	0
198	1	32.0	137.0	18	2
199	1	30.0	137.0	83	0

200 rows × 5 columns

11. Split Dataset into Predictors and Target

```
[ ] feature = pd.DataFrame(df_transformed.drop(['class'], axis = 1))  
label = pd.DataFrame(df_transformed['class'])
```

```
feature.head()
```

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	19.0	15.0	39
1	1	21.0	15.0	81
2	0	20.0	16.0	6
3	0	32.0	16.0	77
4	0	31.0	17.0	40

+ Code

+ Text

```
[ ] label.head()
```

	class
0	4
1	3
2	4
3	3
4	4

12. Splitting into Training and Test Datasets

```
[ ] X_train, X_test, y_train, y_test = train_test_split(feature, label, test_size = 0.2, random_state = 0)

print('X_train : ')
print(X_train)
print(X_train.shape)

print('')
print('X_test : ')
print(X_test)
print(X_test.shape)

print('')
print('y_train : ')
print(y_train)
print(y_train.shape)

print('')
print('y_test : ')
print(y_test)
print(y_test.shape)
```

```
X_train :
  Gender  Age  Annual Income (k$)  Spending Score (1-100)
134      1  20.0                73.0                    5
66       0  43.0                48.0                   50
26       0  45.0                28.0                   32
113      1  19.0                64.0                   46
168      0  36.0                87.0                   27
..      ...  ...                ...                   ...
67       0  68.0                48.0                   48
192      1  33.0               113.0                    8
117      0  49.0                65.0                   59
47       0  27.0                40.0                   47
172      1  36.0                87.0                   10
```

```
[160 rows x 4 columns]
(160, 4)
```

```
X_test :
  Gender  Age  Annual Income (k$)  Spending Score (1-100)
18      1  52.0        60.641026                    29
170     1  40.0        87.000000                    13
107     1  54.0        63.000000                    46
98      1  48.0        61.000000                    42
177     1  27.0        88.000000                    69
182     1  46.0        98.000000                    15
5       0  22.0        17.000000                    76
146     1  48.0        77.000000                    36
12      0  58.0        20.000000                    15
152     0  44.0        78.000000                    20
61      1  19.0        46.000000                    55
125     0  31.0        70.000000                    77
180     0  37.0        97.000000                    32
154     0  47.0        78.000000                    16
80      1  57.0        54.000000                    51
7       0  23.0        18.000000                    94
33      1  18.0        33.000000                    92
130     1  47.0        71.000000                     9
37      0  30.0        34.000000                    73
74      1  59.0        54.000000                    47
183     0  29.0        98.000000                    88
```

```

y_train :
      class
134      2
66      1
26      4
113     1
168     2
..     ...
67      1
192     2
117     1
47      1
172     2

[160 rows x 1 columns]
(160, 1)

y_test :
      class
18      1
170     2
107     1
98      1
177     0
182     2
5       3
146     2
12      4
152     2
61      1
125     0
180     2
154     2
80      1
7       3
33      3
130     2
37      3

```

13. Build the Model

```

[ ] rfc = RandomForestClassifier()

forest_params = [{'max_depth': list(range(10, 15)), 'max_features': list(range(0,14))}]

clf = GridSearchCV(rfc, forest_params, cv = 10, scoring='accuracy')

clf.fit(X_train, y_train)

print(clf.best_params_)

print(clf.best_score_)

```

c:\python\python39\lib\site-packages\sklearn\model_selection_validation.py:598: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), f

c:\python\python39\lib\site-packages\sklearn\model_selection_validation.py:615: FitFailedWarning: Estimator fit failed. The score on this train-test partition for these parameters will be set to nan. Details:

Traceback (most recent call last):

File "c:\python\python39\lib\site-packages\sklearn\model_selection_validation.py", line 598, in _fit_and_score

estimator.fit(X_train, y_train, **fit_params)

File "c:\python\python39\lib\site-packages\sklearn\ensemble_forest.py", line 387, in fit

trees = Parallel(n_jobs=self.n_jobs, verbose=self.verbose,

File "c:\python\python39\lib\site-packages\joblib\parallel.py", line 1041, in __call__

if self.dispatch_one_batch(iterator):

File "c:\python\python39\lib\site-packages\joblib\parallel.py", line 859, in dispatch_one_batch

self.dispatch(tasks)

File "c:\python\python39\lib\site-packages\joblib\parallel.py", line 777, in _dispatch

job = self._backend.apply_async(batch, callback=cb)

File "c:\python\python39\lib\site-packages\joblib_parallel_backends.py", line 208, in apply_async

result = ImmediateResult(func)

File "c:\python\python39\lib\site-packages\joblib_parallel_backends.py", line 572, in __init__

self.results = batch()

File "c:\python\python39\lib\site-packages\joblib\parallel.py", line 262, in __call__

return [func(*args, **kwargs)

File "c:\python\python39\lib\site-packages\joblib\parallel.py", line 262, in <listcomp>

return [func(*args, **kwargs)

File "c:\python\python39\lib\site-packages\sklearn\utils\fixes.py", line 222, in __call__

return self.function(*args, **kwargs)

14. Train the Model

```

[ ] model = RandomForestClassifier(max_depth=12, max_features=1).fit(X_train,y_train)

```

15. Test the Model

```
[ ] y_pred = model.predict(X_test)
```

```
[ ] y_pred
```

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

16. Evaluation Metrics

```
[ ] from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
cm = confusion_matrix(y_test, y_pred)  
sns.heatmap(cm, annot=True, cmap='viridis')  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	0.80	1.00	0.89	12
2	1.00	0.88	0.93	8
3	1.00	0.80	0.89	5
4	1.00	0.80	0.89	5
accuracy				0.93
macro avg				0.96
weighted avg				0.94

