In [109…
```
import numpy as np
import pandas as pd
```

In [110…
```
data=pd.read_csv(r"C:\Users\prasa\OneDrive\Desktop\Mall_Customers.csv")  # raw strings
# backslash is treated as an escape character  if we dont use r ,
data
```

Out[110…

	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
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

In [110…
```
data.describe(include="all")  # NaN stands for not a number
```

Out[110…

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200	200.000000	200.000000	200.000000
unique	NaN	2	NaN	NaN	NaN
top	NaN	Female	NaN	NaN	NaN
freq	NaN	112	NaN	NaN	NaN
mean	100.500000	NaN	38.850000	60.560000	50.200000
std	57.879185	NaN	13.969007	26.264721	25.823522
min	1.000000	NaN	18.000000	15.000000	1.000000
25%	50.750000	NaN	28.750000	41.500000	34.750000
50%	100.500000	NaN	36.000000	61.500000	50.000000
75%	150.250000	NaN	49.000000	78.000000	73.000000
max	200.000000	NaN	70.000000	137.000000	99.000000

In [110…
```
data.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
```

In [110…
```
data["Gender"].value_counts()
```

Out[110…
```
Female    112
Male       88
Name: Gender, dtype: int64
```

In [110…
```
#data["Gender"]=1
data
```

Out[110…

	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
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

In [110…
```
#if data.Gender=="Male":
 #   data["Gender"]=1
#else:
#    data["Gender"]=0
#data
```

In [ ]:
```
```

In [110…
```
#dummy_data=pd.get_dummies(data.Gender,prefix="Gender")
#dummy_data
```

In [110…
```
#data=data.drop('Gender',axis=1)
data["Gender"].replace({"Female":0,"Male":1},inplace=True)
data
```

Out[110…

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40
…	…	…	…	…	…
195	196	0	35	120	79
196	197	0	45	126	28
197	198	1	32	126	74
198	199	1	32	137	18
199	200	1	30	137	83

200 rows × 5 columns

In [110…

```
#data=pd.concat([data,dummy_data],axis=1)
data
```

Out[110…

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40
…	…	…	…	…	…
195	196	0	35	120	79
196	197	0	45	126	28
197	198	1	32	126	74
198	199	1	32	137	18
199	200	1	30	137	83

200 rows × 5 columns

In [110…

```
data.isnull()
```

Out[110…

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	False	False	False	False	False
1	False	False	False	False	False

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
195	False	False	False	False	False
196	False	False	False	False	False
197	False	False	False	False	False
198	False	False	False	False	False
199	False	False	False	False	False

200 rows × 5 columns

In [111...
```
data.isnull().values.any()
```

Out[111... False

In [111...
```
data.isnull().sum()
```

Out[111...
```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

In [111...
```
# we can drop the customer id column
data=data.drop(["CustomerID"],axis=1)
```

In [111...
```
data
```

Out[111...

	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
...	...	...	...	...
195	0	35	120	79
196	0	45	126	28
197	1	32	126	74
198	1	32	137	18
199	1	30	137	83

200 rows × 4 columns

In [111…
```
#from sklearn.preprocessing import MinMaxScaler
#scaler=MinMaxScaler()
#transformed_data=scaler.fit_transform(data)
#from sklearn.preprocessing import StandardScaler
#scaler=StandardScaler()
#transformed_data=scaler.fit_transform(data)
```

We are getting good results without scaling the data

In [111…
```
transformed_data=data
```

In [111…
```
new_data=pd.DataFrame(transformed_data)
new_data
```

Out[111…

	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
…	…	…	…	…
195	0	35	120	79
196	0	45	126	28
197	1	32	126	74
198	1	32	137	18
199	1	30	137	83

200 rows × 4 columns

In [111…
```
new_data.columns=data.columns
```

In [111…
```
new_data
```

Out[111…

	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

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
...	...	...	...	...
195	0	35	120	79
196	0	45	126	28
197	1	32	126	74
198	1	32	137	18
199	1	30	137	83

200 rows × 4 columns

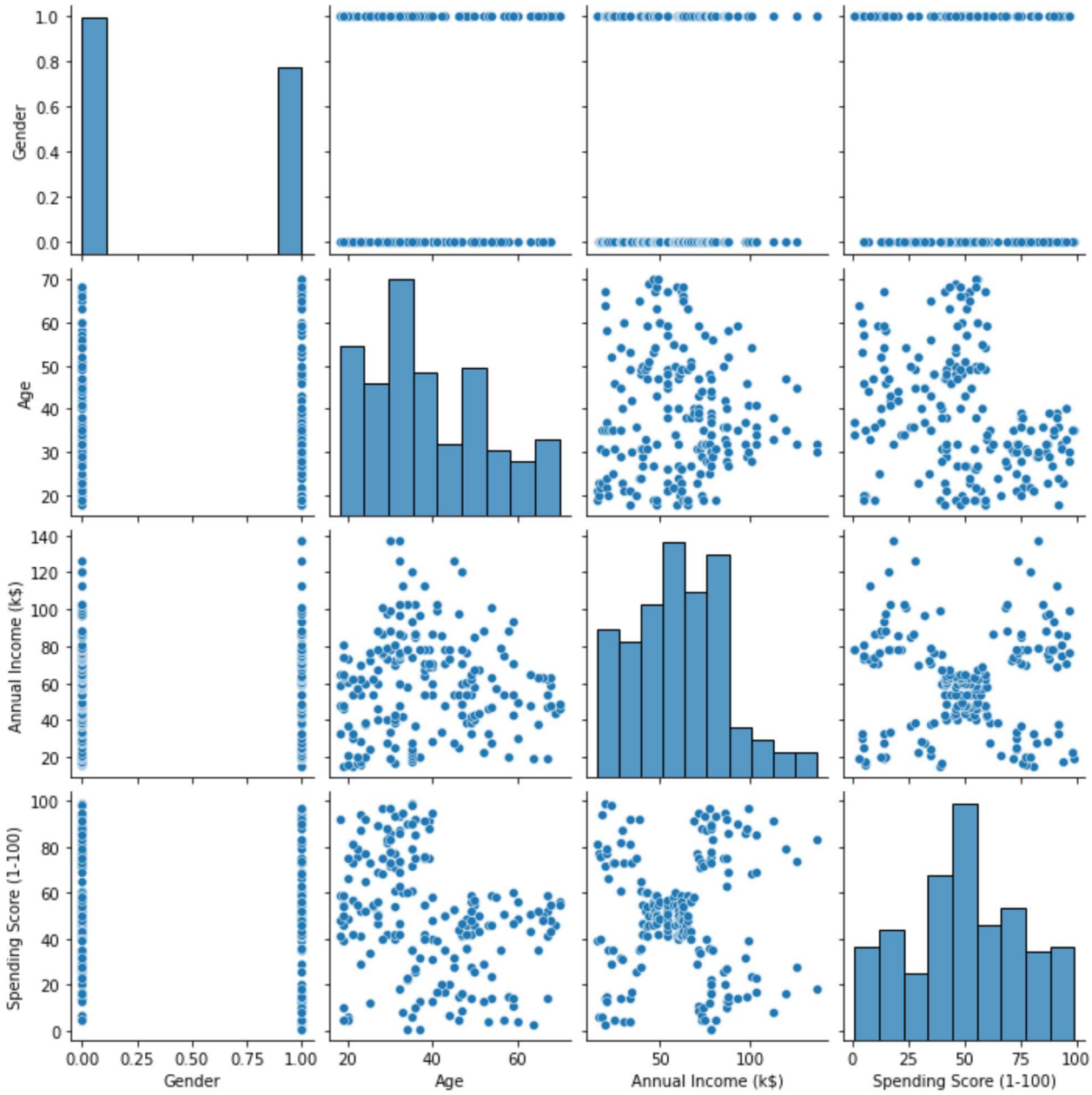
In [111…
```
new_data.corr()
```

Out[111…

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
Gender	1.000000	0.060867	0.056410	-0.058109
Age	0.060867	1.000000	-0.012398	-0.327227
Annual Income (k$)	0.056410	-0.012398	1.000000	0.009903
Spending Score (1-100)	-0.058109	-0.327227	0.009903	1.000000

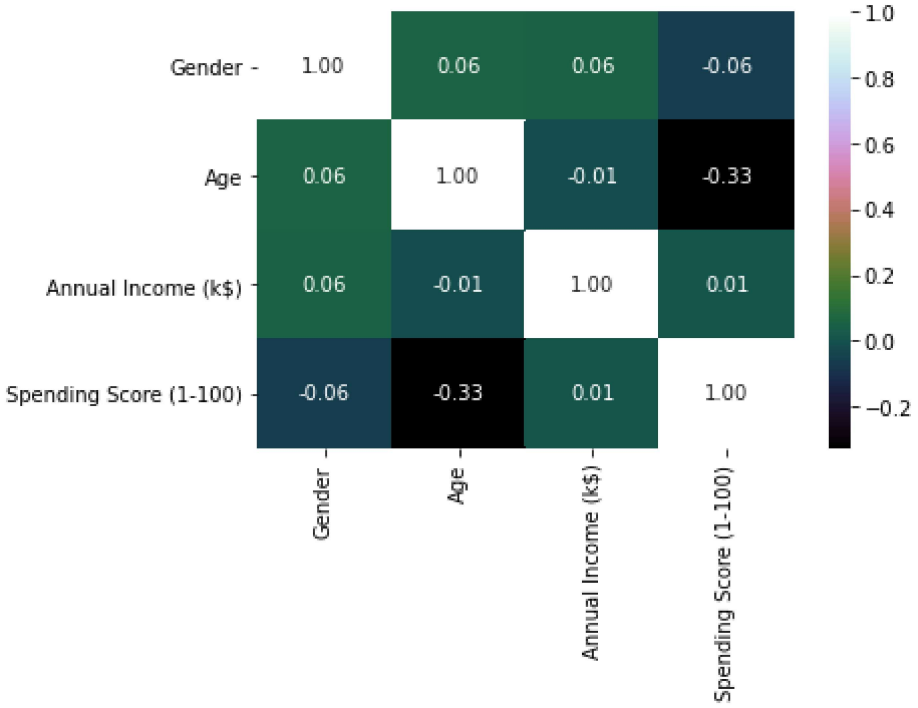
In [112…
```
import seaborn as sns
sns.pairplot(new_data)
```

Out[112… `<seaborn.axisgrid.PairGrid at 0x21237da8580>`



In [112…
```
sns.heatmap(new_data.corr(),annot=True,cmap="cubehelix",fmt="0.2f")
```

Out[112…
```
<AxesSubplot:>
```



K Means Clustering

In [112…

```
from sklearn.cluster import KMeans
WCSS=[] # within the cluster sum of squares
for i in range(1,20):
    # 300 iterations for single run and n_init=10 is no.of times kmeans algo will run o
    kmeans=KMeans(n_clusters=i,random_state=42,init="k-means++",max_iter=300,n_init=10)
    print(kmeans)
    kmeans.fit(new_data)
    WCSS.append(kmeans.inertia_)
    print(WCSS)
```

```
KMeans(n_clusters=1, random_state=42)
[308862.06000000006]
KMeans(n_clusters=2, random_state=42)
[308862.06000000006, 212889.44245524294]
KMeans(n_clusters=3, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674]
KMeans(n_clusters=4, random_state=42)
C:\Users\prasa\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning:
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks th
an available threads. You can avoid it by setting the environment variable OMP_NUM_THREA
DS=1.
  warnings.warn(
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174]
KMeans(n_clusters=5, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152]
KMeans(n_clusters=6, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505]
KMeans(n_clusters=7, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
```

```
1182424152, 58348.64136331505, 51575.27793107792]
KMeans(random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483]
KMeans(n_clusters=9, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116]
KMeans(n_clusters=10, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254]
KMeans(n_clusters=11, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725]
KMeans(n_clusters=12, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128]
KMeans(n_clusters=13, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746]
KMeans(n_clusters=14, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166]
KMeans(n_clusters=15, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166, 26112.983028083014]
KMeans(n_clusters=16, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166, 26112.983028083014, 24851.938927738935]
KMeans(n_clusters=17, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166, 26112.983028083014, 24851.938927738935, 22841.880561840113]
KMeans(n_clusters=18, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166, 26112.983028083014, 24851.938927738935, 22841.880561840113, 22249.35773132096]
KMeans(n_clusters=19, random_state=42)
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166, 26112.983028083014, 24851.938927738935, 22841.880561840113, 22249.35773132096,
21738.224406604742]
```

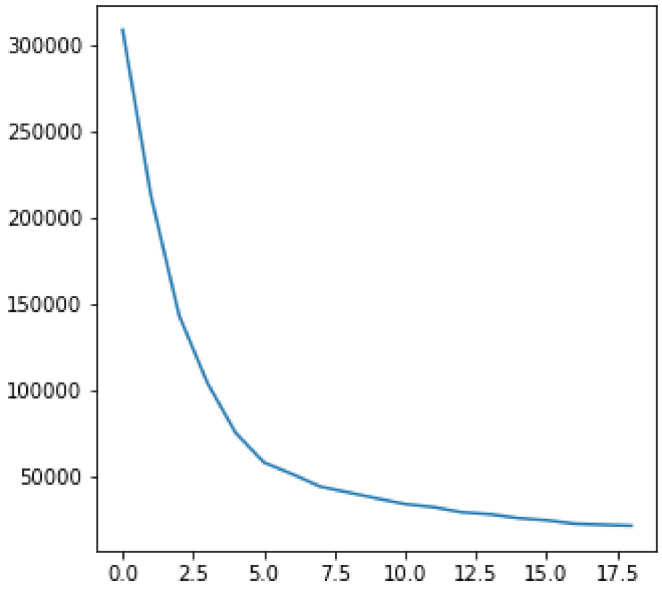
In [112…
```
print(WCSS)
```

```
[308862.06000000006, 212889.44245524294, 143391.59236035674, 104414.67534220174, 75427.7
1182424152, 58348.64136331505, 51575.27793107792, 44359.6346411483, 40942.51117006116, 3
7515.841255041254, 34221.804728234725, 32479.794685507128, 29472.951755651746, 28346.701
65945166, 26112.983028083014, 24851.938927738935, 22841.880561840113, 22249.35773132096,
21738.224406604742]
```

In [112…
```
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5))
```

```
plt.plot(WCSS)
```

Out[112…
```
[<matplotlib.lines.Line2D at 0x21239c82040>]
```



In [112…
```
kmeans=KMeans(n_clusters=5,random_state=42,max_iter=300,n_init=10)
pred_y=kmeans.fit_predict(new_data)
pred_y
```

Out[112…
```
array([0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4,
       0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 3,
       0, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 2, 1, 3, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1])
```

In [112…
```
print(kmeans.labels_)
print(kmeans.n_iter_)
```

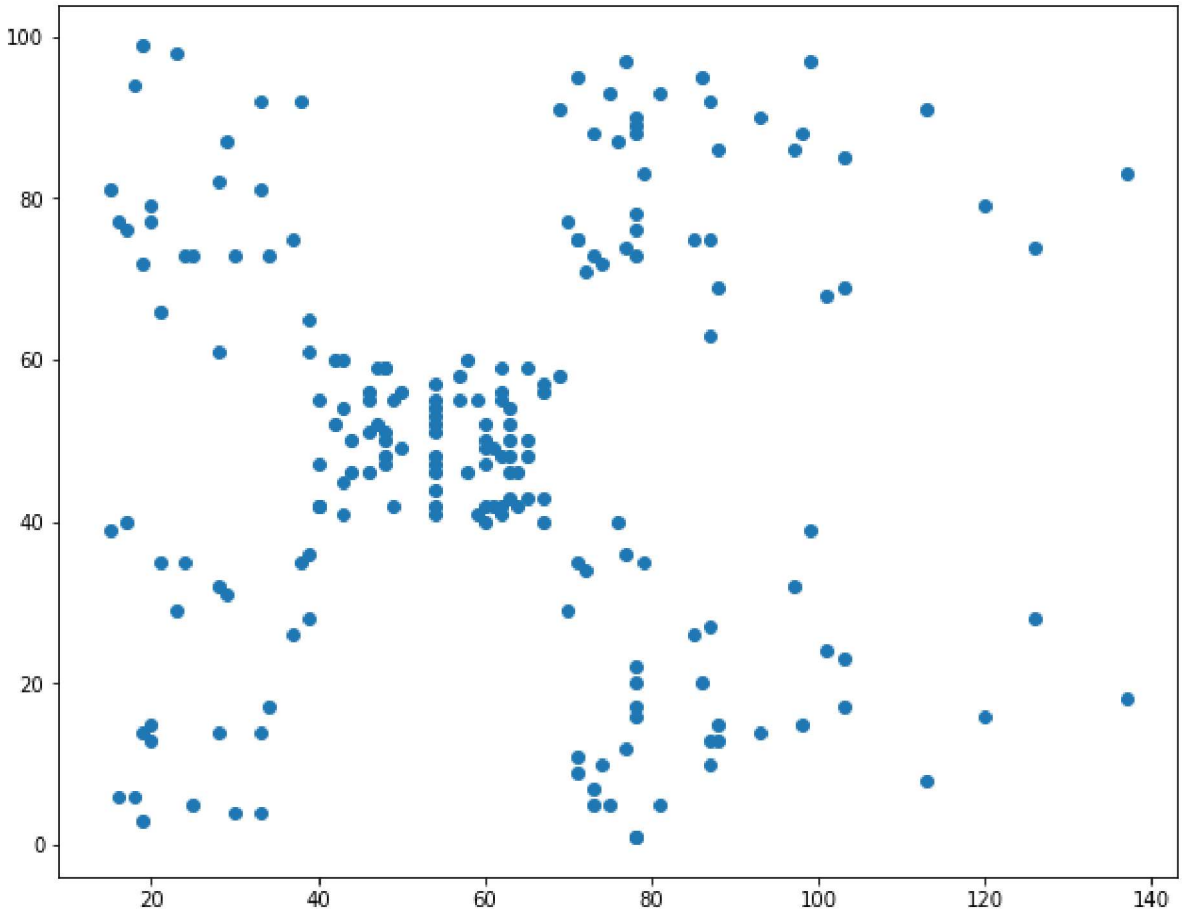
```
[0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0
 4 0 4 0 4 0 3 0 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 1 2 1 3 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1]
6
```

In [112…
```
print(kmeans.cluster_centers_)
```

```
[[ 0.39130435 45.2173913  26.30434783 20.91304348]
 [ 0.46153846 32.69230769 86.53846154 82.12820513]
 [ 0.51351351 40.32432432 87.43243243 18.18918919]
 [ 0.41772152 43.12658228 54.82278481 49.83544304]
 [ 0.40909091 25.27272727 25.72727273 79.36363636]]
```

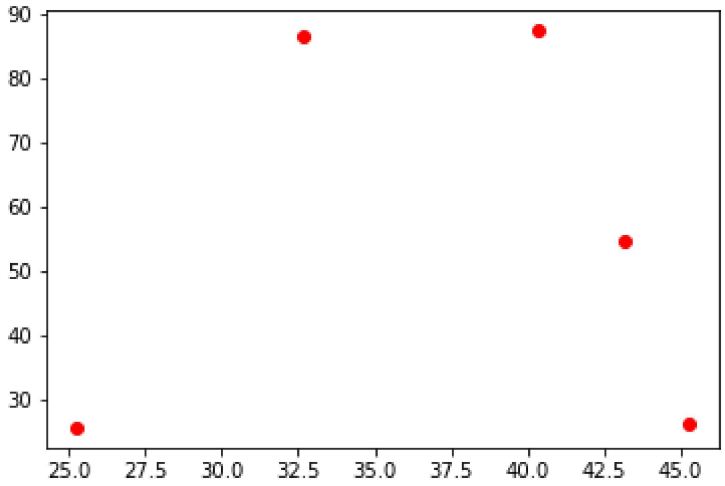
In [112…
```
plt.figure(figsize=(10,8))
plt.scatter(new_data.iloc[ :,2],new_data.iloc[ :,3])  # annual income vs spending score
```

Out[112…
```
<matplotlib.collections.PathCollection at 0x21239cdc250>
```



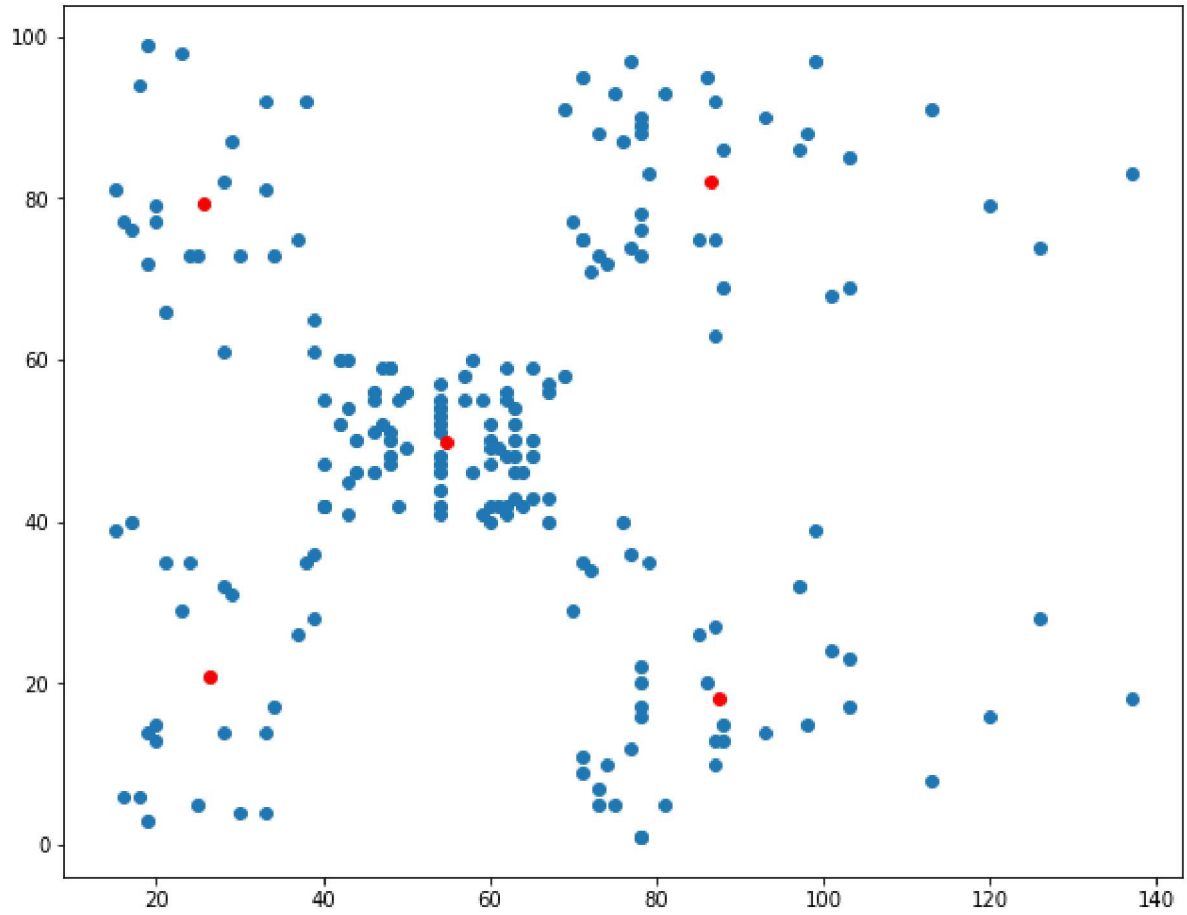
In [112…
```
plt.scatter(kmeans.cluster_centers_[:,1],kmeans.cluster_centers_[:,2],c="red")
```

Out[112…
```
<matplotlib.collections.PathCollection at 0x21239d2c5e0>
```



In [113…
```
plt.figure(figsize=(10,8))
plt.scatter(new_data.iloc[ :,2],new_data.iloc[ :,3])
plt.scatter(kmeans.cluster_centers_[:,2],kmeans.cluster_centers_[:,3],c="red")
```

Out[113…
```
<matplotlib.collections.PathCollection at 0x21239d4fbe0>
```



DBSCAN Clustering

In [113…
```
from sklearn.cluster import DBSCAN
dbscan=DBSCAN(eps=12,min_samples=4,metric="euclidean")#min samples default=5, eps defau
dbscan.fit(new_data)
y_pred=dbscan.fit_predict(new_data)
y_pred       # fit_predict calls fit method and returns labels..not a difference b/w fi
```

Out[113…
```
array([-1,  0, -1,  0,  1,  0, -1, -1,  2,  0, -1, -1,  2,  0, -1,  0,  1,
        0,  1, -1,  1,  0,  2,  0,  2,  0,  1,  0,  1,  0,  2,  0,  2,  0,
        2,  0,  2,  0,  1,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  3, -1,  3,  0,  3, -1,  3,  4,  3, -1,  3,  5,  3,
        4,  3,  5,  3,  4,  3, -1,  3,  5,  3,  0,  3,  4,  3,  4,  3,  4,
        3,  4,  3,  4,  3, -1,  3, -1,  3,  5,  3,  4,  3,  4,  3,  4,  3,
        4,  3,  4,  3,  4,  3,  4,  3,  4,  3,  4,  3,  4,  3,  4,  3, -1,
       -1, -1,  3,  4, -1, -1,  3, -1, -1, -1, -1, -1, -1], dtype=int64)
```

In [113…
```
print(dbscan.labels_)
```

```
[-1  0 -1  0  1  0 -1 -1  2  0 -1 -1  2  0 -1  0  1  0  1 -1  1  0  2  0
  2  0  1  0  1  0  2  0  2  0  2  0  2  0  1  0 -1  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  3 -1  3  0  3 -1  3  4  3 -1  3  5  3  4  3  5  3  4  3 -1  3
  5  3  0  3  4  3  4  3  4  3  4  3  4  3 -1  3 -1  3  5  3  4  3  4  3
```

```
 4  3  4  3  4  3  4  3  4  3  4  3  4  3  4  3  4  3 -1 -1 -1  3  4 -1
-1  3 -1 -1 -1 -1 -1 -1]
```

In [113…
```
print(dbscan.core_sample_indices_)
```

```
[  1   3   5   9  13  15  16  17  20  21  23  24  25  26  28  29  30  31
  32  33  34  35  37  39  42  43  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
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 125 126 127
 129 130 131 133 134 135 136 137 138 139 141 143 144 145 147 148 149 150
 151 152 153 154 155 157 159 161 162 163 165 166 167 168 170 171 173 174
 175 177 179 180 181 183 185 189]
```

In [113…
```
print(dbscan.components_)
```

```
[[  1  21  15  81]
 [  0  23  16  77]
 [  0  22  17  76]
 [  0  30  19  72]
 [  0  24  20  77]
 [  1  22  20  79]
 [  0  35  21  35]
 [  1  20  21  66]
 [  1  35  24  35]
 [  1  25  24  73]
 [  1  31  25  73]
 [  0  54  28  14]
 [  1  29  28  82]
 [  0  45  28  32]
 [  0  40  29  31]
 [  0  23  29  87]
 [  1  60  30   4]
 [  0  21  30  73]
 [  1  53  33   4]
 [  1  18  33  92]
 [  0  49  33  14]
 [  0  21  33  81]
 [  0  30  34  73]
 [  0  20  37  75]
 [  1  48  39  36]
 [  0  31  39  61]
 [  0  24  39  65]
 [  0  50  40  55]
 [  0  27  40  47]
 [  0  29  40  42]
 [  0  31  40  42]
 [  0  49  42  52]
 [  1  33  42  60]
 [  0  31  43  54]
 [  1  59  43  60]
 [  0  50  43  45]
 [  1  47  43  41]
 [  0  51  44  50]
 [  1  69  44  46]
 [  0  27  46  51]
 [  1  53  46  46]
 [  1  70  46  56]
 [  1  19  46  55]
 [  0  67  47  52]
 [  0  54  47  59]
 [  1  63  48  51]
 [  1  18  48  59]
 [  0  43  48  50]
```

```
[  0  68  48  48]
[  1  19  48  59]
[  0  32  48  47]
[  1  70  49  55]
[  0  47  49  42]
[  0  60  50  49]
[  0  60  50  56]
[  1  59  54  47]
[  1  26  54  54]
[  0  45  54  53]
[  1  40  54  48]
[  0  23  54  52]
[  0  49  54  42]
[  1  57  54  51]
[  1  38  54  55]
[  1  67  54  41]
[  0  46  54  44]
[  0  21  54  57]
[  1  48  54  46]
[  0  55  57  58]
[  0  22  57  55]
[  0  34  58  60]
[  0  50  58  46]
[  0  68  59  55]
[  1  18  59  41]
[  1  48  60  49]
[  0  40  60  40]
[  0  32  60  42]
[  1  24  60  52]
[  0  47  60  47]
[  0  27  60  50]
[  1  48  61  42]
[  1  20  61  49]
[  0  23  62  41]
[  0  49  62  48]
[  1  67  62  59]
[  1  26  62  55]
[  1  49  62  56]
[  0  21  62  42]
[  0  66  63  50]
[  1  54  63  46]
[  1  68  63  43]
[  1  66  63  48]
[  1  65  63  52]
[  0  19  63  54]
[  0  38  64  42]
[  1  19  64  46]
[  0  18  65  48]
[  0  19  65  50]
[  0  63  65  43]
[  0  49  65  59]
[  0  51  67  43]
[  0  50  67  57]
[  1  27  67  56]
[  0  38  67  40]
[  0  40  69  58]
[  1  39  69  91]
[  0  31  70  77]
[  1  43  71  35]
[  1  40  71  95]
[  1  38  71  75]
[  1  47  71   9]
[  1  39  71  75]
[  0  31  72  71]
[  1  20  73   5]
```

```
[  0  29  73  88]
[  0  44  73   7]
[  1  32  73  73]
[  1  19  74  10]
[  0  35  74  72]
[  1  32  75  93]
[  0  32  76  87]
[  1  25  77  12]
[  1  28  77  97]
[  0  32  77  74]
[  0  34  78  22]
[  1  34  78  90]
[  1  43  78  17]
[  1  39  78  88]
[  0  44  78  20]
[  0  38  78  76]
[  0  47  78  16]
[  0  27  78  89]
[  0  30  78  78]
[  0  30  78  73]
[  0  29  79  83]
[  1  19  81   5]
[  0  31  81  93]
[  0  36  85  75]
[  1  42  86  20]
[  0  33  86  95]
[  0  36  87  27]
[  1  40  87  13]
[  1  28  87  75]
[  1  36  87  92]
[  0  52  88  13]
[  0  30  88  86]
[  1  27  88  69]
[  1  35  93  90]
[  0  37  97  32]
[  0  32  97  86]
[  0  29  98  88]
[  1  30  99  97]
[  0  36 103  85]]
```

In [113…
```
n_clusters=len(set(dbscan.labels_))- (1 if -1 in dbscan.labels_ else 0)
n_clusters
```

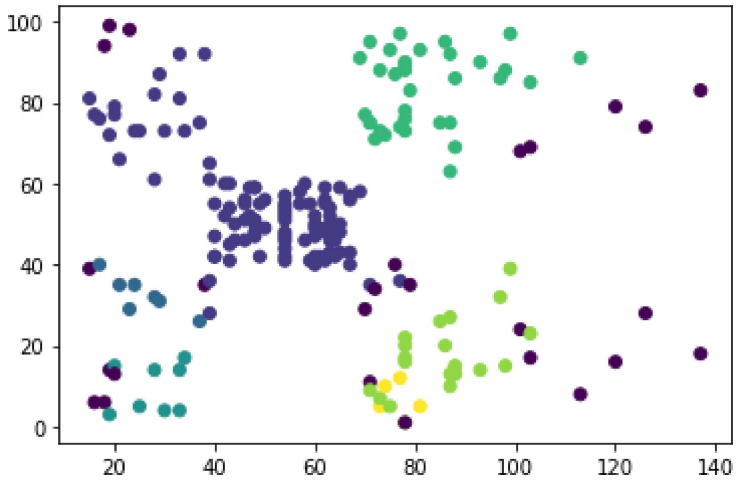
Out[113…
```
6
```

In [113…
```
print(metrics.silhouette_score(new_data,dbscan.labels_))
```

```
0.19367691397868803
```

In [113…
```
plt.scatter(new_data.iloc[:,2],new_data.iloc[:,3],c=y_pred)
```

Out[113…
```
<matplotlib.collections.PathCollection at 0x21239de9670>
```



In [ ]:

In [ ]: