Assignment-4

Assignment Date	04 October 2022
Student Name	Gunasekaran A
Student Roll Number	2127190801023
Maximum Marks	2 Marks

Question-1:

Download the dataset

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Question-2:

Load the dataset

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

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

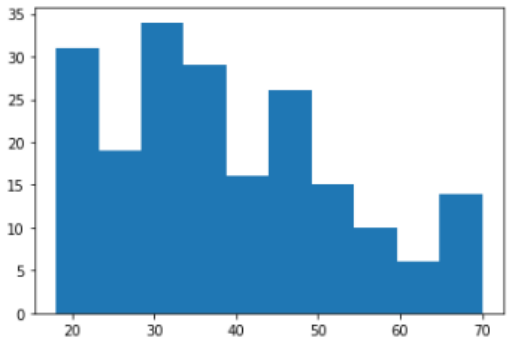
Question 3:

Perform Below Visualizations.

1) Univariate Analysis

In [4]:
```
plt.hist(data['Age'])
```

Out[4]:
```
(array([31., 19., 34., 29., 16., 26., 15., 10.,  6., 14.]),
 array([18. , 23.2, 28.4, 33.6, 38.8, 44. , 49.2, 54.4, 59.6, 64.8, 70. ]),
 )
```



2) Bi-variate analysis

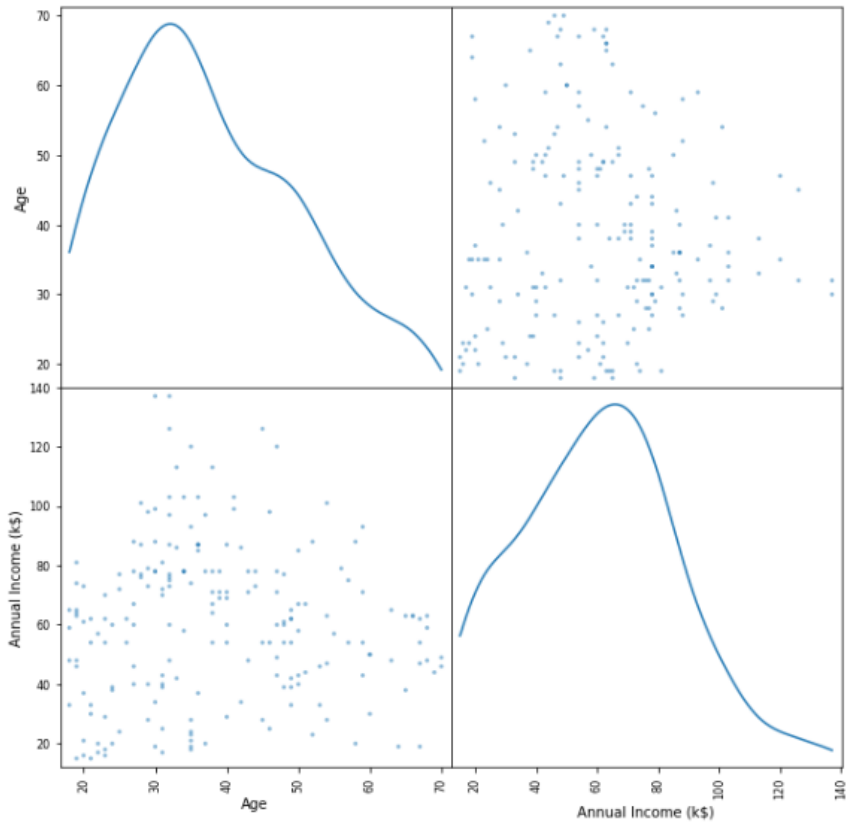
In [5]:
```
plt.scatter(data['Gender'],data['Annual Income (k$)'])
```

Out[5]:

3) Multi-variate analysis

In [6]:
```
pd.plotting.scatter_matrix(data.loc[:,"Age":"Annual Income (k$)"] , diagonal ="kde", figsize=(10,10))
```

Out[6]:
```
array([[,
        ],
       [,
        ]],
      dtype=object)
```



Question 4:

Perform descriptive statistics on the dataset

In [7]:
```
data.describe()
```

Out[7]:

	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

In [8]:
```
data.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
CustomerID	200.0	100.50	57.879185	1.0	50.75	100.5	150.25	200.0
Age	200.0	38.85	13.969007	18.0	28.75	36.0	49.00	70.0
Annual Income (k$)	200.0	60.56	26.264721	15.0	41.50	61.5	78.00	137.0
Spending Score (1-100)	200.0	50.20	25.823522	1.0	34.75	50.0	73.00	99.0

Question 5:

Check for missing values

and deal with them

In [9]:
```
data.isna().sum()
```

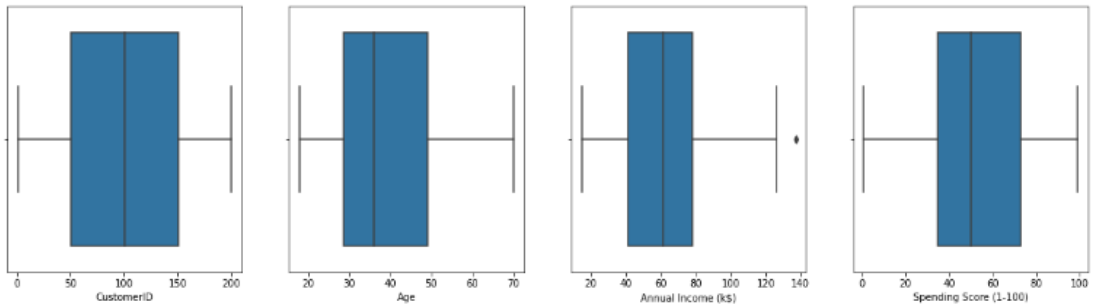
Out[9]:
```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

Question 6:

Find the outliers and replace them outliers

In [10]:
```
fig,ax=plt.subplots(figsize=(25,5))

plt.subplot(1, 5, 2)
sns.boxplot(x=data['Age'])

plt.subplot(1, 5, 3)
sns.boxplot(x=data['Annual Income (k$)'])

plt.subplot(1, 5, 4)
sns.boxplot(x=data['Spending Score (1-100)'])

plt.subplot(1, 5, 1)
sns.boxplot(x=data['CustomerID'])
```

Out[10]:



Handling Outlier

In [11]:
```
quant=data.quantile(q=[0.25,0.75])
quant
```

Out[11]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
0.25	50.75	28.75	41.5	34.75
0.75	150.25	49.00	78.0	73.00

In [12]:
```
quant.loc[0.75]
```

Out[12]:
```
CustomerID                150.25
Age                        49.00
Annual Income (k$)         78.00
Spending Score (1-100)     73.00
Name: 0.75, dtype: float64
```

In [13]:
```
quant.loc[0.25]
```

Out[13]:
```
CustomerID                50.75
Age                       28.75
Annual Income (k$)        41.50
Spending Score (1-100)    34.75
Name: 0.25, dtype: float64
```

In [14]:
```
iqr=quant.loc[0.75]-quant.loc[0.25]
iqr
```

Out[14]:
```
CustomerID                99.50
Age                       20.25
Annual Income (k$)        36.50
Spending Score (1-100)    38.25
dtype: float64
```

In [15]:
```
low=quant.loc[0.25]-(1.5 *iqr)
low
```

Out[15]:
```
CustomerID               -98.500
Age                       -1.625
Annual Income (k$)       -13.250
Spending Score (1-100)   -22.625
dtype: float64
```

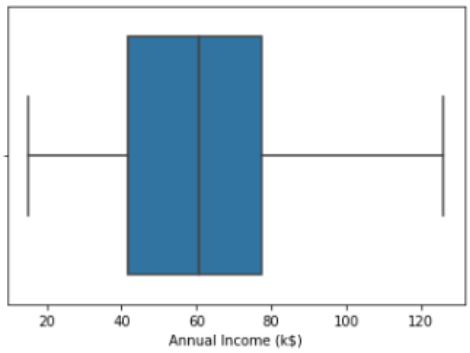
In [16]:
```
up=quant.loc[0.75]+(1.5 *iqr)
up
```

Out[16]:
```
CustomerID              299.500
Age                      79.375
Annual Income (k$)      132.750
Spending Score (1-100)  130.375
dtype: float64
```

In [18]:
```
data['Annual Income (k$)']= np.where(data['Annual Income (k$)']>132,60,data['Annual Income (k$)'])
```

In [19]:
```
sns.boxplot(x=data['Annual Income (k$)'])
```

Out[19]:



Question 7:

Check for Categorical columns and perform encoding.

In [20]:
```
data.info()
```

```
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 [21]:
```
data['Gender'].unique()
```

Out[21]:
```
array(['Male', 'Female'], dtype=object)
```

In [22]:
```
data['Gender'].replace({'Male':1,"Female":0},inplace=True)
```

In [23]:
```
data
```

Out[23]:

	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	60	18
199	200	1	30	60	83

200 rows × 5 columns

Question 8:

Scaling the data

In [24]:
```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
```

In [25]:
```
df=sc.fit_transform(data.iloc[:,1:])
```

In [26]:
```
df
```

Out[26]:
```
array([[1.        , 0.01923077, 0.        , 0.3877551 ],
       [1.        , 0.05769231, 0.        , 0.81632653],
       [0.        , 0.03846154, 0.00900901, 0.05102041],
       [0.        , 0.09615385, 0.00900901, 0.7755102 ],
       [0.        , 0.25      , 0.01801802, 0.39795918],
       [0.        , 0.07692308, 0.01801802, 0.76530612],
       [0.        , 0.32692308, 0.02702703, 0.05102041],
       [0.        , 0.09615385, 0.02702703, 0.94897959],
       [1.        , 0.88461538, 0.03603604, 0.02040816],
       [0.        , 0.23076923, 0.03603604, 0.7244898 ],
       [1.        , 0.94230769, 0.03603604, 0.13265306],
       [0.        , 0.32692308, 0.03603604, 1.        ],
       [0.        , 0.76923077, 0.04504505, 0.14285714],
       [0.        , 0.11538462, 0.04504505, 0.7755102 ],
       [1.        , 0.36538462, 0.04504505, 0.12244898],
       [1.        , 0.07692308, 0.04504505, 0.79591837],
       [0.        , 0.32692308, 0.05405405, 0.34693878],
       [1.        , 0.03846154, 0.05405405, 0.66326531],
       [1.        , 0.65384615, 0.07207207, 0.28571429],
       [0.        , 0.32692308, 0.07207207, 0.98979592],
       [1.        , 0.32692308, 0.08108108, 0.34693878],
       [1.        , 0.13461538, 0.08108108, 0.73469388],
       [0.        , 0.53846154, 0.09009009, 0.04081633],
       [1.        , 0.25      , 0.09009009, 0.73469388],
       [0.        , 0.69230769, 0.11711712, 0.13265306],
       [1.        , 0.21153846, 0.11711712, 0.82653061],
       [0.        , 0.51923077, 0.11711712, 0.31632653],
       [1.        , 0.32692308, 0.11711712, 0.6122449 ],
       [0.        , 0.42307692, 0.12612613, 0.30612245],
       [0.        , 0.09615385, 0.12612613, 0.87755102],
       [1.        , 0.80769231, 0.13513514, 0.03061224],
       [0.        , 0.05769231, 0.13513514, 0.73469388],
       [1.        , 0.67307692, 0.16216216, 0.03061224],
       [1.        , 0.        , 0.16216216, 0.92857143],
       [0.        , 0.59615385, 0.16216216, 0.13265306],
       [0.        , 0.05769231, 0.16216216, 0.81632653],
       [0.        , 0.46153846, 0.17117117, 0.16326531],
       [0.        , 0.23076923, 0.17117117, 0.73469388],
       [0.        , 0.34615385, 0.1981982 , 0.25510204],
       [0.        , 0.03846154, 0.1981982 , 0.75510204],
       [0.        , 0.90384615, 0.20720721, 0.34693878],
       [1.        , 0.11538462, 0.20720721, 0.92857143],
       [1.        , 0.57692308, 0.21621622, 0.35714286],
       [0.        , 0.25      , 0.21621622, 0.6122449 ],
       [0.        , 0.59615385, 0.21621622, 0.2755102 ],
       [0.        , 0.11538462, 0.21621622, 0.65306122],
       [0.        , 0.61538462, 0.22522523, 0.55102041],
       [0.        , 0.17307692, 0.22522523, 0.46938776],
       [0.        , 0.21153846, 0.22522523, 0.41836735],
       [0.        , 0.25      , 0.22522523, 0.41836735],
       [0.        , 0.59615385, 0.24324324, 0.52040816],
       [1.        , 0.28846154, 0.24324324, 0.60204082],
       [0.        , 0.25      , 0.25225225, 0.54081633],
       [1.        , 0.78846154, 0.25225225, 0.60204082],
       [0.        , 0.61538462, 0.25225225, 0.44897959],
       [1.        , 0.55769231, 0.25225225, 0.40816327],
       [0.        , 0.63461538, 0.26126126, 0.5       ],
```

```
[1.        , 0.13461538, 0.55855856, 0.1122449 ],
[1.        , 0.19230769, 0.55855856, 0.97959184],
[1.        , 0.57692308, 0.55855856, 0.35714286],
[0.        , 0.26923077, 0.55855856, 0.74489796],
[0.        , 0.30769231, 0.56756757, 0.21428571],
[1.        , 0.30769231, 0.56756757, 0.90816327],
[1.        , 0.48076923, 0.56756757, 0.16326531],
[1.        , 0.40384615, 0.56756757, 0.8877551 ],
[0.        , 0.5       , 0.56756757, 0.19387755],
[0.        , 0.38461538, 0.56756757, 0.76530612],
[0.        , 0.55769231, 0.56756757, 0.15306122],
[0.        , 0.17307692, 0.56756757, 0.89795918],
[1.        , 0.36538462, 0.56756757, 0.        ],
[0.        , 0.23076923, 0.56756757, 0.78571429],
[1.        , 0.30769231, 0.56756757, 0.        ],
[0.        , 0.23076923, 0.56756757, 0.73469388],
[0.        , 0.73076923, 0.57657658, 0.34693878],
[0.        , 0.21153846, 0.57657658, 0.83673469],
[1.        , 0.01923077, 0.59459459, 0.04081633],
[0.        , 0.25      , 0.59459459, 0.93877551],
[1.        , 0.61538462, 0.63063063, 0.25510204],
[0.        , 0.34615385, 0.63063063, 0.75510204],
[1.        , 0.46153846, 0.63963964, 0.19387755],
[0.        , 0.28846154, 0.63963964, 0.95918367],
[0.        , 0.34615385, 0.64864865, 0.26530612],
[1.        , 0.26923077, 0.64864865, 0.63265306],
[1.        , 0.42307692, 0.64864865, 0.12244898],
[1.        , 0.19230769, 0.64864865, 0.75510204],
[1.        , 0.34615385, 0.64864865, 0.09183673],
[1.        , 0.34615385, 0.64864865, 0.92857143],
[0.        , 0.65384615, 0.65765766, 0.12244898],
[0.        , 0.23076923, 0.65765766, 0.86734694],
[1.        , 0.76923077, 0.65765766, 0.14285714],
[1.        , 0.17307692, 0.65765766, 0.69387755],
[1.        , 0.78846154, 0.7027027 , 0.13265306],
[1.        , 0.32692308, 0.7027027 , 0.90816327],
[0.        , 0.36538462, 0.73873874, 0.31632653],
[0.        , 0.26923077, 0.73873874, 0.86734694],
[1.        , 0.53846154, 0.74774775, 0.14285714],
[0.        , 0.21153846, 0.74774775, 0.8877551 ],
[0.        , 0.44230769, 0.75675676, 0.3877551 ],
[1.        , 0.23076923, 0.75675676, 0.97959184],
[0.        , 0.69230769, 0.77477477, 0.23469388],
[1.        , 0.19230769, 0.77477477, 0.68367347],
[0.        , 0.44230769, 0.79279279, 0.16326531],
[0.        , 0.34615385, 0.79279279, 0.85714286],
[0.        , 0.30769231, 0.79279279, 0.2244898 ],
[0.        , 0.26923077, 0.79279279, 0.69387755],
[1.        , 0.28846154, 0.88288288, 0.07142857],
[0.        , 0.38461538, 0.88288288, 0.91836735],
[0.        , 0.55769231, 0.94594595, 0.15306122],
[0.        , 0.32692308, 0.94594595, 0.79591837],
[0.        , 0.51923077, 1.        , 0.2755102 ],
[1.        , 0.26923077, 1.        , 0.74489796],
[1.        , 0.26923077, 0.40540541, 0.17346939],
[1.        , 0.23076923, 0.40540541, 0.83673469]])
```

Question 9:

Perform any of the clustering algorithms

Kmeans_clustering

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

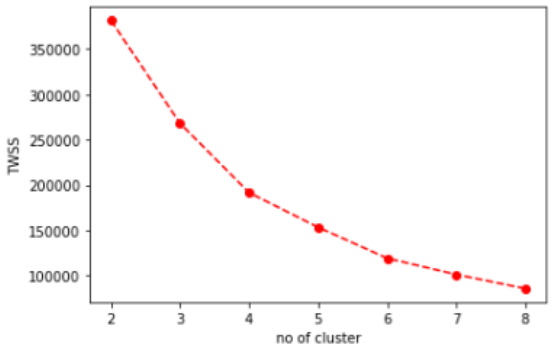
In [28]:
```
TWSS=[]
k=list(range(2,9))

for i in k:
    kmeans=KMeans(n_clusters=i,init='k-means++')
    kmeans.fit(data)
    TWSS.append(kmeans.inertia_)
```

In [29]:
```
TWSS
```

Out[29]:
```
[381550.6840684068,
 268082.56760639744,
 191612.56821803437,
 153394.66603206735,
 119223.63779954854,
 101364.2432178932,
 85819.89345888031]
```

In [30]:
```
plt.plot(k,TWSS,'ro--')
plt.xlabel('no of cluster')
plt.ylabel('TWSS')
```

Out[30]:
```
Text(0, 0.5, 'TWSS')
```



In [31]:
```
#selecting  4 clusters

model=KMeans(n_clusters=4)

model.fit(data)
```

Out[31]:
```
KMeans(n_clusters=4)
```

In [32]:
```
model.labels_
```

Out[32]:
```
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, 1, 1, 1, 1, 1, 0, 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, 3, 0, 3, 0, 3, 0, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3])
```

In [33]:
```
mb=pd.Series(model.labels_)
```

In [34]:
```
data.head(3)
```

Out[34]:

	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

Question 10:
Add the cluster data with the primary dataset

In [35]:
```
data['clust']=mb
```

In [36]:
```
data.head()
```

Out[36]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	clust
0	1	1	19	15	39	1
1	2	1	21	15	81	1
2	3	0	20	16	6	1
3	4	0	23	16	77	1
4	5	0	31	17	40	1

In [37]:
```
data.tail()
```

Out[37]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	clust
195	196	0	35	120	79	3
196	197	0	45	126	28	2
197	198	1	32	126	74	3
198	199	1	32	60	18	2
199	200	1	30	60	83	3

Question 11:
Split the data into dependent and independent variables.

In [38]:
```
#dependent
y= data['clust']
y
```

Out[38]:
```
0      1
1      1
2      1
3      1
4      1
      ..
195    3
196    2
197    3
198    2
199    3
Name: clust, Length: 200, dtype: int32
```

In [39]:
```
#independent
x= data.drop(columns=['CustomerID','clust'],axis=1)
x.head()
```

Out[39]:

	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

In [52]:
```
x.tail()
```

Out[52]:

	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	60	18
199	1	30	60	83

Question 12:

Split the model into training and testing

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

In [ ]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Question 13:

Build the model.

In [41]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [42]:
```
rf=RandomForestClassifier()
```

Question 14:

Train the model.

In [117…
```
rf.fit(x_train,y_train)
```

Out[117… RandomForestClassifier()

Question 15:

Test the model.

In [118…
```
#prediction
pred=rf.predict(x_test)
```

Question 16:

Measure the performance using Metrics.

In [119…
```
# Accuracy of DI model
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

Out[119… 0.975

In [120…
```
#confusion matrix
from sklearn import metrics
metrics.confusion_matrix(y_test,pred)
```

Out[120…
```
array([[13,  0,  0,  0],
       [ 0, 10,  0,  0],
       [ 1,  0,  8,  0],
       [ 0,  0,  0,  8]], dtype=int64)
```