Assignment 4

Python Programming

Assignment Date	25 Oct 2022
Maximum Marks	2 Marks

Question 1:

1 . Importing Required Package

Solution :

```
import pandas as pd
import numpy as np
import seaborn as sbn
import matplotlib.pyplot as plt
```

Question 2:

2 . Loading the Dataset

Solution :

```
db = pd.read_csv('/Mall_Customers.csv')
Db
```

Output

Out[4]:

	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 ÃƒÂ— 5 columns

Question 3:

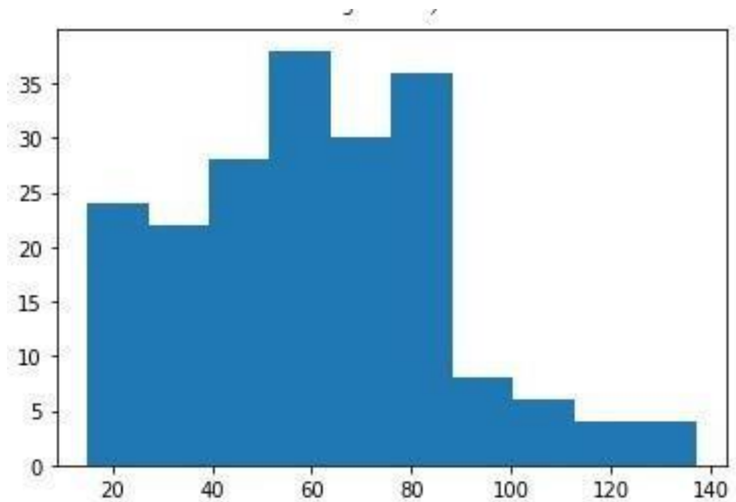
3 . Visualizations

1. UniVariate Analysis

1. Solution : plt.hist(db['Annual

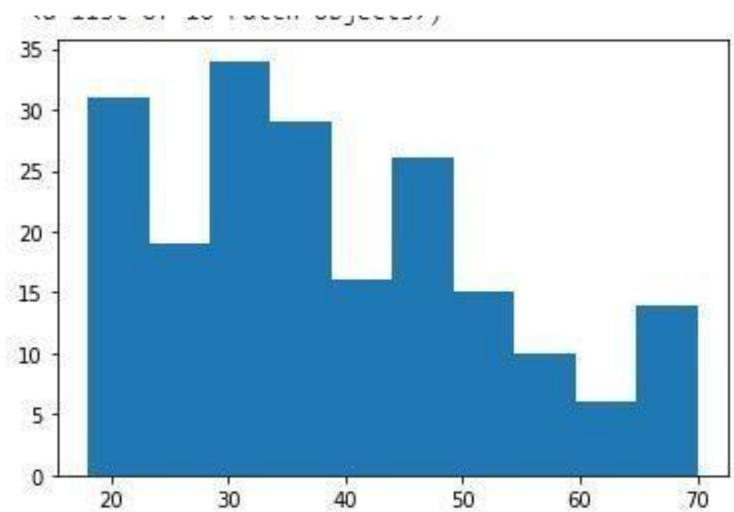
Income (k$)'])

Output :



3.1.2 Solution
plt.hist(db['Age'])

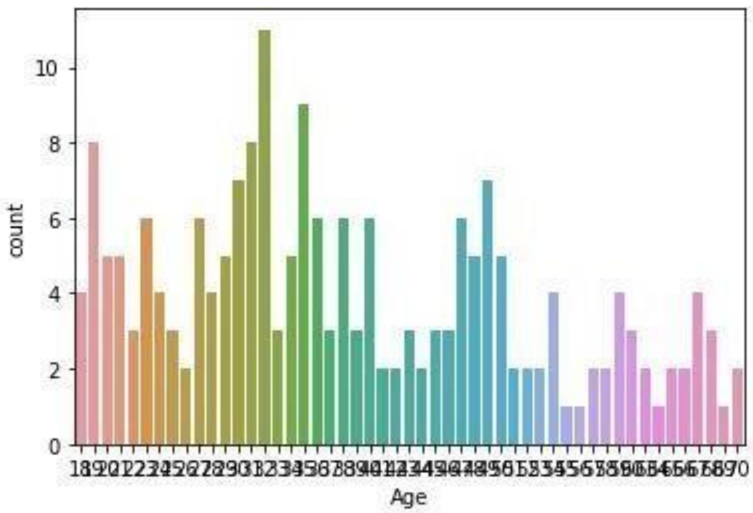
Output :



3.1.3 Solution :

sbn.countplot(db['Age'])

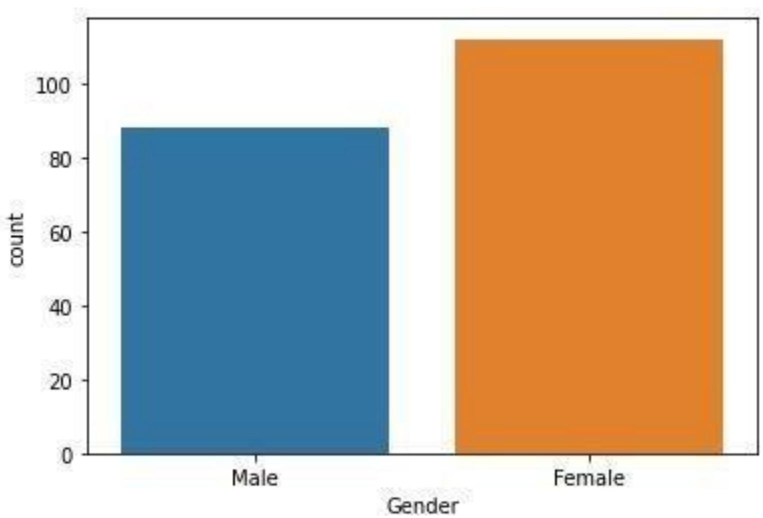
Output :



3.1.4 Solution :

sbn.countplot(db['Gender'])

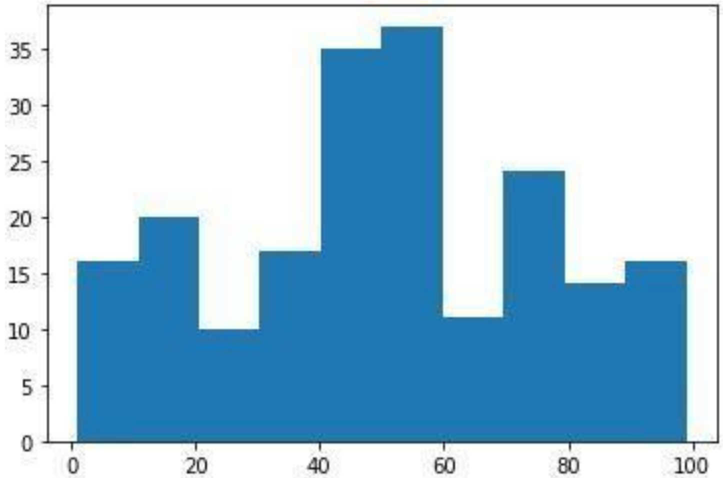
Output :



3.1.5 Solution :

plt.hist(db['Spending Score (1-100)'])

Output :

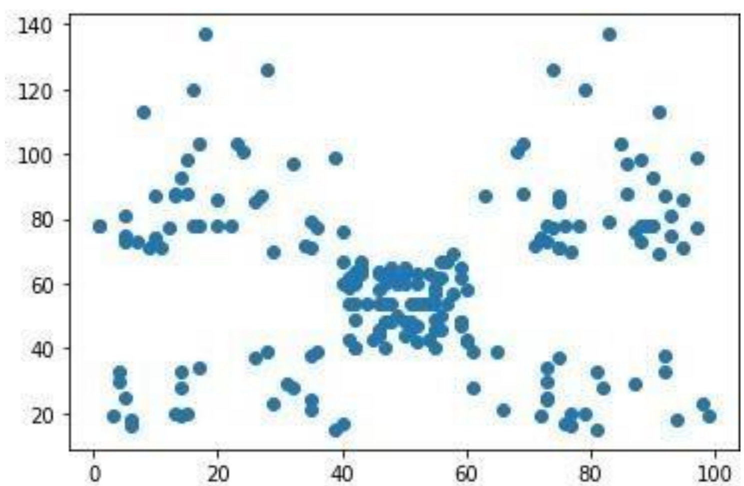


2. Bi-Variate Analysis

1. Solution :

plt.scatter(db['Spending Score (1-100)'],db['Annual Income (k$)'])

Output :



3.2.2 Solution :

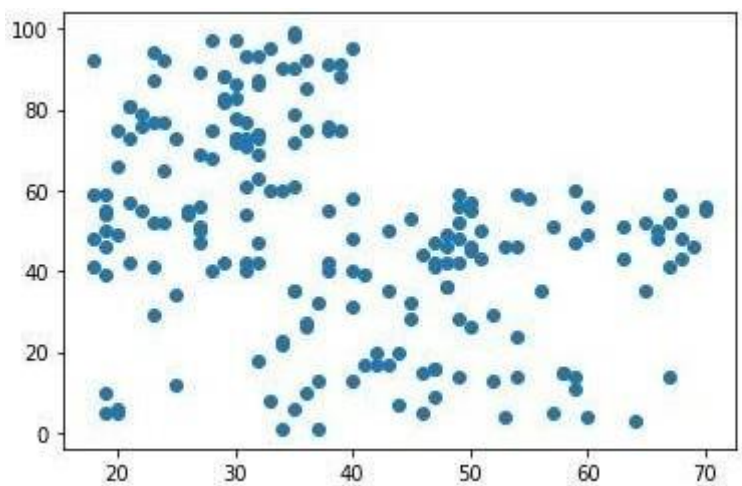
plt.scatter(db['Gender'],db['Annual Income (k$)'])

Output :

3.2.3 Solution :

plt.scatter(db['Age'],db['Spending Score (1-100)'])

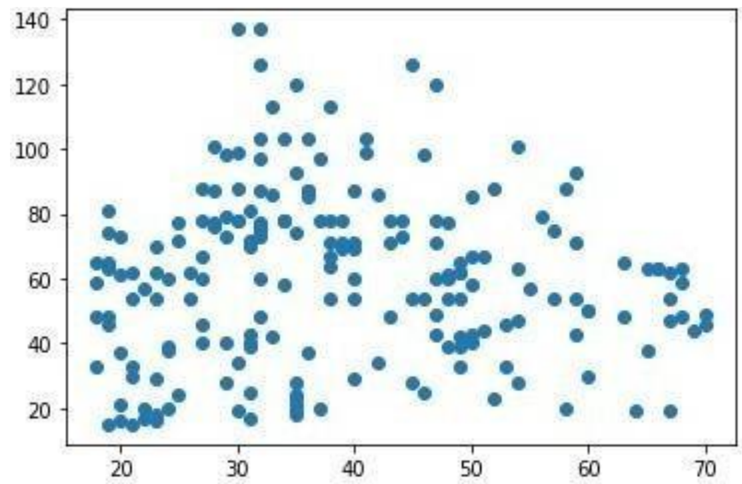
Output :



3.2.4 Solution :

plt.scatter(db['Age'],db['Annual Income (k$)'])

Output :



3.2.5 Solution：

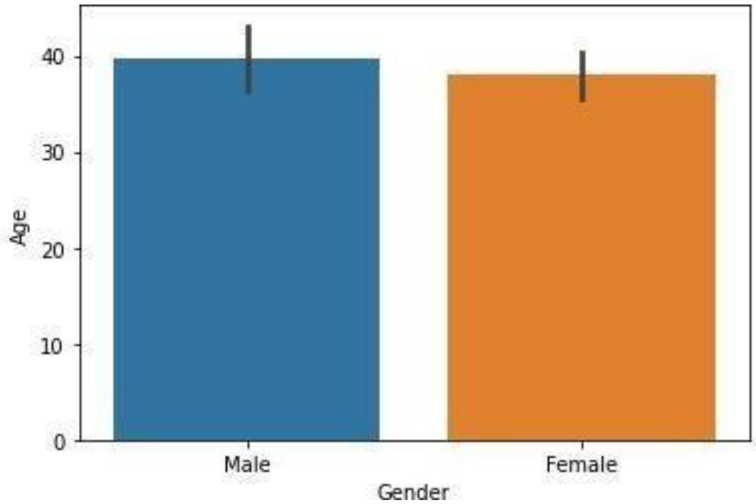
sbn.heatmap(db.corr(), annot = **True**)

Output：

3.2.6 Solution：

sbn.barplot(db['Gender'], db['Age'])

Output :

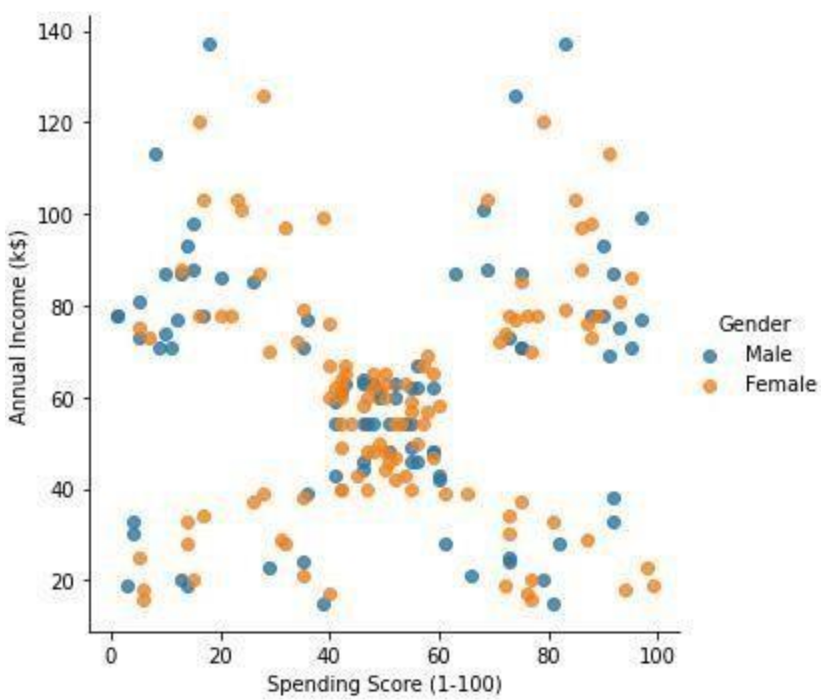


3. Multi-Variate Analysis

1. Solution :

sbn**.**lmplot("Spending Score (1-100)","Annual Income (k$)", db, hue="Gender", fit_reg**=False**);

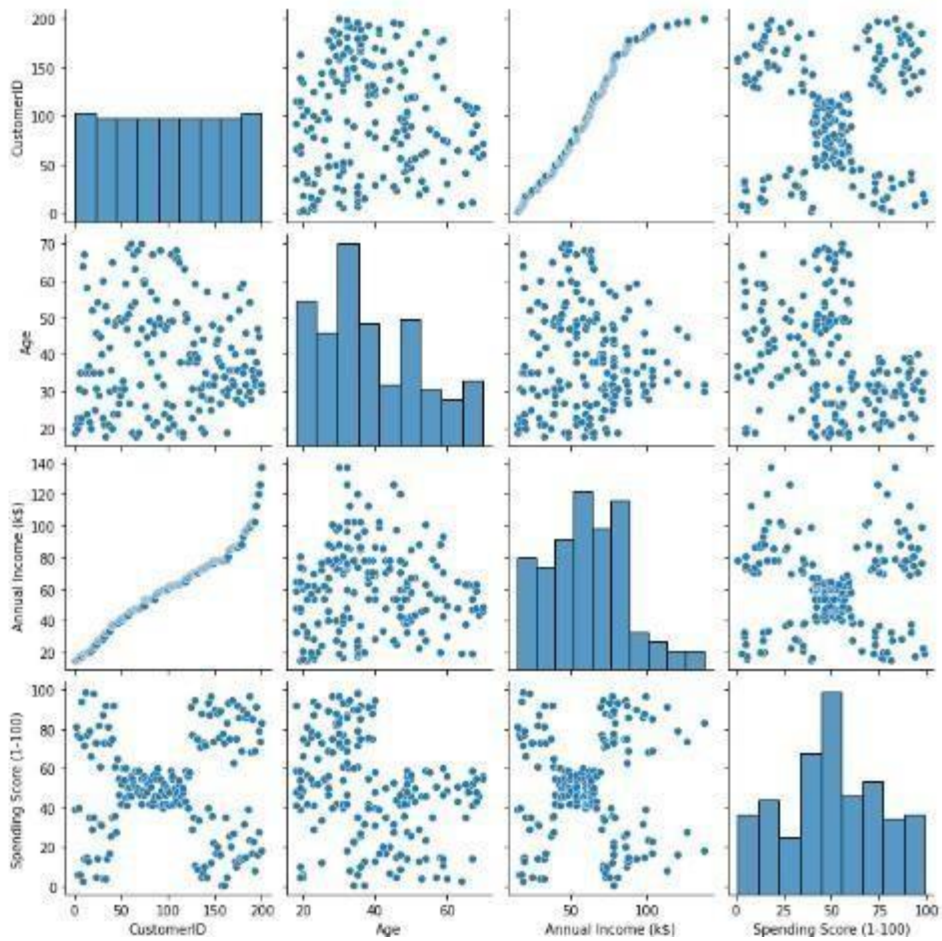
Output :



3.3.2 Solution :

sbn.pairplot(db)

Output :



Question 4:

4 . Perform descriptive statistics on the dataset

1.Solution :

db.describe()

Output :

	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

4.2 Solution :

db.dtypes

Output :

```
CustomerID                 int64
Gender                    object
Age                        int64
Annual Income (k$)         int64
Spending Score (1-100)     int64
dtype: object
```

4.3 Solution :

db.var()

Output :

```
]:  CustomerID                3350.000000
    Age                        195.133166
    Annual Income (k$)         689.835578
    Spending Score (1-100)     666.854271
    dtype: float64
```

4.4 Solution :

db.skew()

Output :

```
CustomerID                0.000000
Age                       0.485569
Annual Income (k$)        0.321843
Spending Score (1-100)   -0.047220
dtype: float64
```

4.5 Solution :

db.corr()

Output :

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
CustomerID	1.000000	-0.026763	0.977548	0.013835
Age	-0.026763	1.000000	-0.012398	-0.327227
Annual Income (k$)	0.977548	-0.012398	1.000000	0.009903
Spending Score (1-100)	0.013835	-0.327227	0.009903	1.000000

4.6 Solution :

db.std()

Output :

```
CustomerID                57.879185
Age                       13.969007
Annual Income (k$)        26.264721
Spending Score (1-100)    25.823522
dtype: float64
```

Question 5:

5. Check for Missing values and deal with them

1. Solution :

db.isna().sum()

Output :

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

5.2 Solution :

db.isna().sum().sum()

Output :

```
0
```

5.3 Solution :

db.duplicated().sum()

Output :

```
0
```

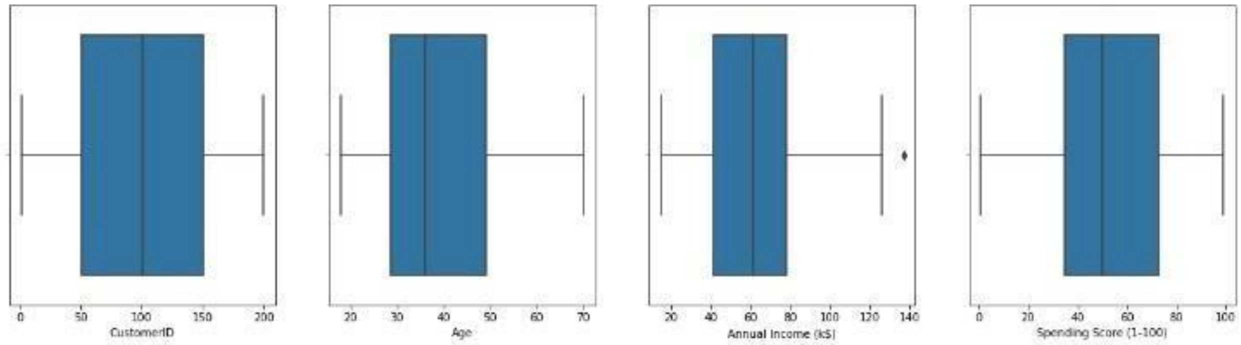
Question 6:

6 . Find the outliers and replace them outliers

1.Solution :

```
ig,ax=plt.subplots(figsize=(25,5))

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

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

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

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

Output :



6.2 Solution :

```
quantile = db.quantile(q = [0.25, 0.75])
quantile
```

Output :

	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

6.3 Solution :

```
quantile.loc[0.75]
```

Output :

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

6.4 Solution :

```
quantile.loc[0.25]
```

Output :

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

6.5 Solution :

IQR = quantile.iloc[1] - quantile.iloc[0]
IQR

Output :

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

6.6 Solution :

upper = quantile.iloc[1] + (1.5 *IQR)
upper

Output :

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

6.7 Solution :

lower = quantile.iloc[0] - (1.5* IQR)
lower

Output :

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

6.8 Solution :

db.mean()

Output :

```
CustomerID                100.50
Age                        38.85
Annual Income (k$)         60.56
Spending Score (1-100)     50.20
dtype: float64
```

9. Solution :

db['Annual Income (k$)'].max()

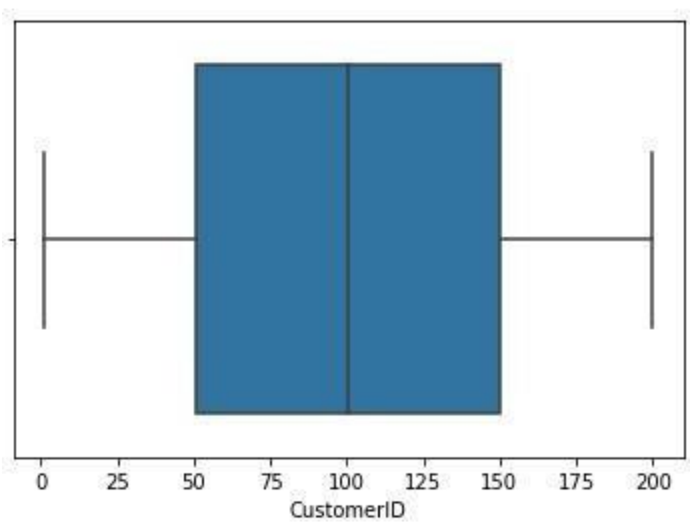
Output :

```
137
```

10. Solution :

sbn.boxplot(db['CustomerID'])

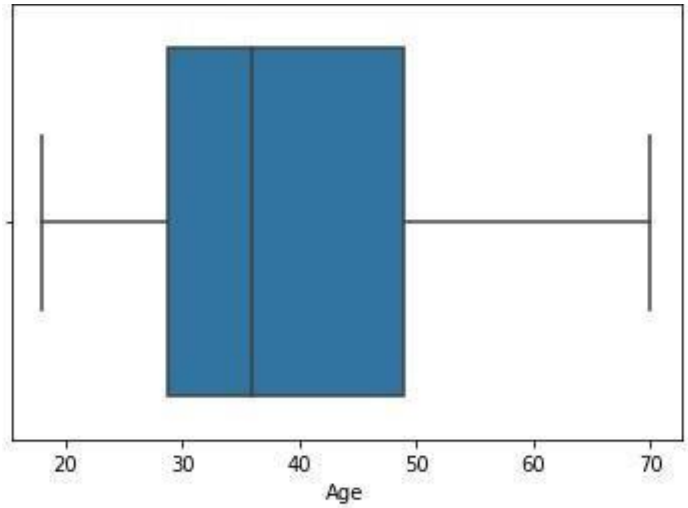
Output :



6.11 Solution :

sbn.boxplot(db['Age'])

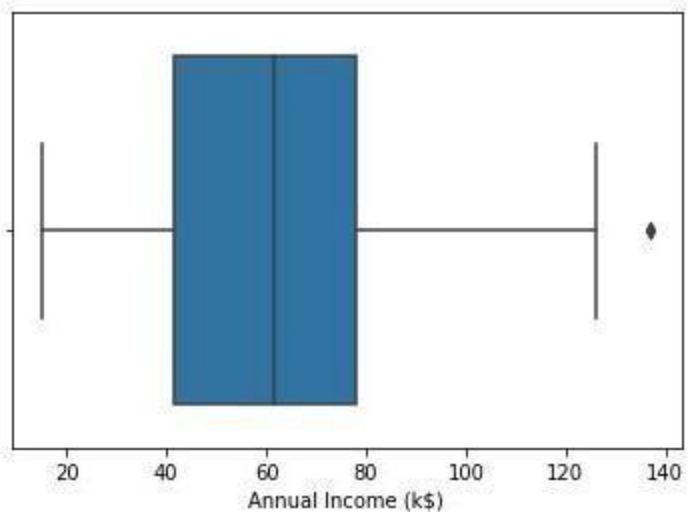
Output :



6.12 Solution :

sbn.boxplot(db['Annual Income (k$)'])

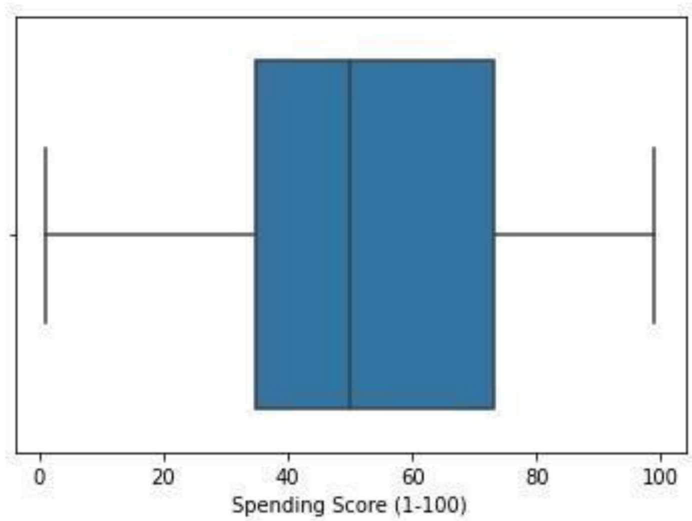
Output :



6.13 Solution :

sbn.boxplot(db['Spending Score (1-100)'])

Output :



Question 7:

7 . Check for Categorical columns and perform encoding

1.Solution :

```
db.select_dtypes(include='object').columns
```

Output :

```
Index(['Gender'], dtype='object')
```

2. Solution :

```
db['Gender'].unique()
```

Output :

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

3. Solution :

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

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15.00	39
1	2	1	21	15.00	81
2	3	0	20	16.00	6
3	4	0	23	16.00	77
4	5	0	31	17.00	40
...	...	...	...	...	...
195	196	0	35	120.00	79
196	197	0	45	126.00	28
197	198	1	32	126.00	74
198	199	1	32	60.55	18
199	200	1	30	60.55	83

200 rows ÃƒÂ— 5 columns

7.4 Solution :

db.head()

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15.0	39
1	2	1	21	15.0	81
2	3	0	20	16.0	6
3	4	0	23	16.0	77
4	5	0	31	17.0	40

Question 8:

8 . Scaling the data

1. Solution :

```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler().fit_transform(db)
```

ss

Output :

Question 9:

9 . Perform any of the clustering algorithms

1. Solution :

```
from sklearn.cluster import KMeans
TWSS = []
k = list(range(2,9))
```

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

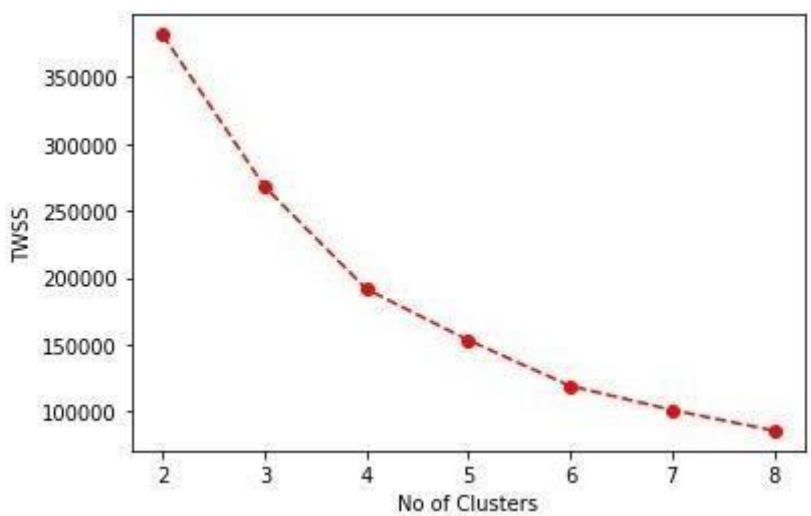
Output :

```
[381507.64738523855,
 268062.55433747417,
 191557.78099047023,
 153327.3825004856,
 119166.15727643928,
 101296.86197582977,
 85792.73210128325]
```

9.2 Solution :

```
plt.plot(k,TWSS, 'ro--')
plt.xlabel('No of Clusters')
plt.ylabel('TWSS')
```

Output :



9.3 Solution :

```
model = KMeans(n_clusters =  4)
model.fit(db)
```

Output :

```
KMeans(n_clusters=4)
```

9.4 Solution :

```
mb = pd.Series(model.labels_)
db['Cluster'] = mb
db
```

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	1	19	15.00	39	1
1	2	1	21	15.00	81	1
2	3	0	20	16.00	6	1
3	4	0	23	16.00	77	1
4	5	0	31	17.00	40	1
...	...	...	...	...	...	...
195	196	0	35	120.00	79	2
196	197	0	45	126.00	28	0
197	198	1	32	126.00	74	2
198	199	1	32	60.55	18	0
199	200	1	30	60.55	83	2

200 rows ÃƒÂ— 6 columns

9.5 Solution :

```
mb=pd.Series(model.labels_)
db.head(3)
```

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	1	19	15.0	39	1
1	2	1	21	15.0	81	1
2	3	0	20	16.0	6	1

Question 10:

10 . Add the cluster data with the primary dataset

1. Solution :

```
db['Cluster']=kmeans.labels_
db.head()
```

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	1	19	15.0	39	5
1	2	1	21	15.0	81	2
2	3	0	20	16.0	6	5
3	4	0	23	16.0	77	2
4	5	0	31	17.0	40	5

10.2 Solution :

```
db.tail()
```

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
195	196	0	35	120.00	79	6
196	197	0	45	126.00	28	1
197	198	1	32	126.00	74	6
198	199	1	32	60.55	18	1
199	200	1	30	60.55	83	6

Question 11:

11 . Split the data into dependent and independent variables

1. Solution :

```
X=db.drop('Cluster',axis=1)
Y=db['Cluster']
y=db['Cluster']
y
```

Output :

```
0      5
1      2
2      5
3      2
4      5
      ..
195    6
196    1
197    6
198    1
199    6
Name: Cluster, Length: 200, dtype: int32
```

11.2 Solution :

from sklearn.model_selection **import** train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=42)

print("Number transactions X_train dataset: ", X_train**.**shape)
print("Number transactions y_train dataset: ", y_train**.**shape)
print("Number transactions X_test dataset: ", X_test**.**shape)
print("Number transactions y_test dataset: ", y_test**.**shape)

Output :

```
Number transactions X_train dataset:  (160, 5)
Number transactions y_train dataset:  (160,)
Number transactions X_test dataset:  (40, 5)
Number transactions y_test dataset:  (40,)
```

Question 12:

12 . Split the data into training and testing

1.Solution :

X_train

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
79	80	0	49	54.0	42
197	198	1	32	126.0	74
38	39	0	36	37.0	26
24	25	0	54	28.0	14
122	123	0	40	69.0	58
...	...	...	...	...	...
106	107	0	66	63.0	50
14	15	1	37	20.0	13
92	93	1	48	60.0	49
179	180	1	35	93.0	90
102	103	1	67	62.0	59

160 rows ÃƒÂ— 5 columns

12.2 Solution :

X_test

Output :

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
95	96	1	24	60.0	52
15	16	1	22	20.0	79
30	31	1	60	30.0	4
158	159	1	34	78.0	1
128	129	1	59	71.0	11
115	116	0	19	65.0	50
69	70	0	32	48.0	47
170	171	1	40	87.0	13
174	175	0	52	88.0	13
45	46	0	24	39.0	65
66	67	0	43	48.0	50
182	183	1	46	98.0	15
165	166	0	36	85.0	75
78	79	0	23	54.0	52
186	187	0	54	101.0	24
177	178	1	27	88.0	69
56	57	0	51	44.0	50
152	153	0	44	78.0	20
82	83	1	67	54.0	41
68	69	1	19	48.0	59
124	125	0	23	70.0	29
16	17	0	35	21.0	35
148	149	0	34	78.0	22
93	94	0	40	60.0	40
65	66	1	18	48.0	59
60	61	1	70	46.0	56
84	85	0	21	54.0	57
67	68	0	68	48.0	48
125	126	0	31	70.0	77
132	133	0	25	72.0	34
9	10	0	30	19.0	72
18	19	1	52	23.0	29
55	56	1	47	43.0	41
75	76	1	26	54.0	54
150	151	1	43	78.0	17
104	105	1	49	62.0	56
135	136	0	29	73.0	88
137	138	1	32	73.0	73
164	165	1	50	85.0	26
76	77	0	45	54.0	53

12.3 Solution :

y_train

Output :

```
79     4
197    6
38     5
24     5
122    0
      ..
106    0
14     5
92     0
179    6
102    0
Name: Cluster, Length: 160, dtype: int32
```

12.14 Solution :

y_test

Output :

```
95     0
15     2
30     5
158    7
128    7
115    0
69     4
170    1
174    1
45     2
66     4
182    1
165    6
78     0
186    1
177    6
56     4
152    7
82     4
68     4
124    7
16     5
148    7
93     0
65     4
60     4
84     0
67     4
125    3
132    7
9      2
18     5
55     4
75     4
150    7
104    0
135    3
137    3
164    1
76     4
Name: Cluster, dtype: int32
```

Question 13:

13 . Build the Model

1. Solution :

from sklearn.linear_model **import** LogisticRegression
model=LogisticRegression()
model**.**fit(X_train,y_train)
from sklearn.linear_model **import** LogisticRegression
model=LogisticRegression()
model**.**fit(X_train,y_train)

Output :

```
LogisticRegression()
```

Question 14:

14 . Train the Model

Solution :

model.score(X_train,y_train)

Output :

```
0.83125
```

Question 15:

15 . Test the Model

Solution :

model.score(X_test,y_test)

Output :

```
0.675
```

Question 16:

16 . Measure the performance using Evaluation Metrics

1. Solution :

from sklearn.metrics **import** confusion_matrix,classification_report
y_pred=model.predict(X_test)
confusion_matrix(y_test,y_pred)

Output :

```
array([[5, 0, 0, 0, 0, 0, 1, 0],
       [0, 5, 0, 0, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0, 0, 0],
       [0, 0, 0, 3, 0, 0, 0, 0],
       [3, 0, 2, 0, 6, 0, 0, 0],
       [0, 0, 0, 0, 0, 3, 0, 0],
       [0, 0, 0, 1, 0, 0, 1, 0],
       [0, 6, 0, 0, 0, 0, 0, 1]])
```

16.2 Solution :

print(classification_report(y_test,y_pred))

Output :

```
              precision    recall  f1-score   support

           0       0.62      0.83      0.71         6
           1       0.45      1.00      0.62         5
           2       0.60      1.00      0.75         3
           3       0.75      1.00      0.86         3
           4       1.00      0.55      0.71        11
           5       1.00      1.00      1.00         3
           6       0.50      0.50      0.50         2
           7       1.00      0.14      0.25         7

    accuracy                           0.68        40
   macro avg       0.74      0.75      0.68        40
weighted avg       0.80      0.68      0.64        40
```