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

1&2. Downloading and Loading the dataset

```
data = pd.read_csv("Mall_Customers.csv")
```

```
data
```

	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

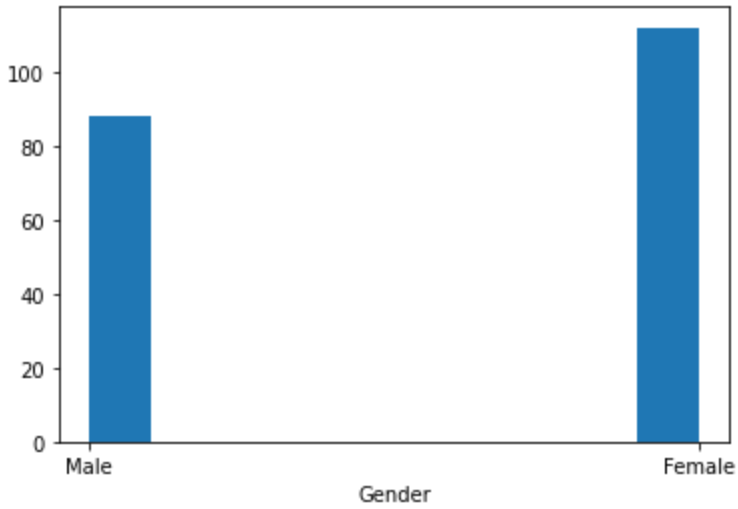
200 rows × 5 columns

3. Visualizing the analysis

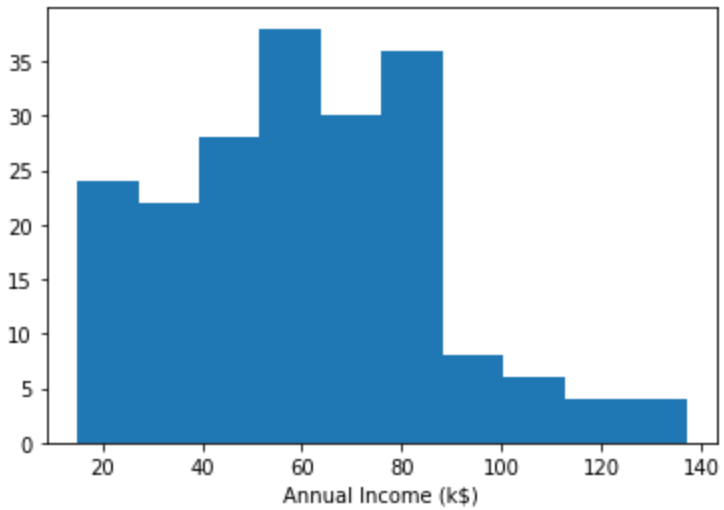
3.1 Univariate Analysis

```
plt.hist(data['Age']);
plt.xlabel('Age');
plt.show();
```

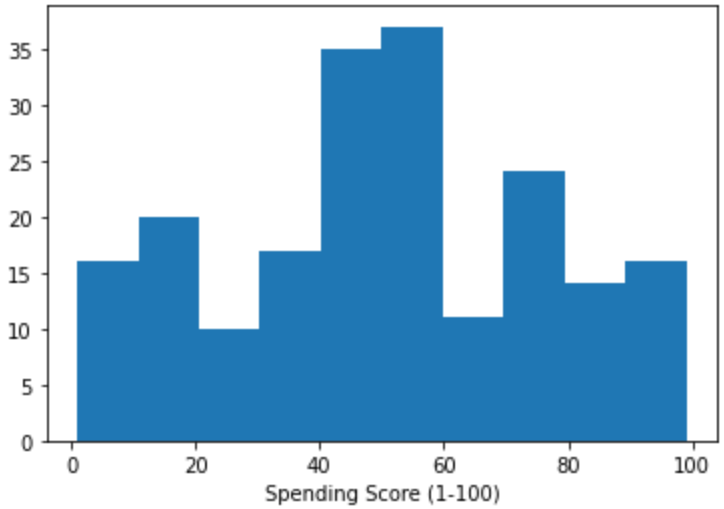

```
plt.hist(data['Gender']);
plt.xlabel('Gender');
plt.show();
```




```
plt.hist(data['Annual Income (k$)']);
plt.xlabel('Annual Income (k$)');
plt.show();
```

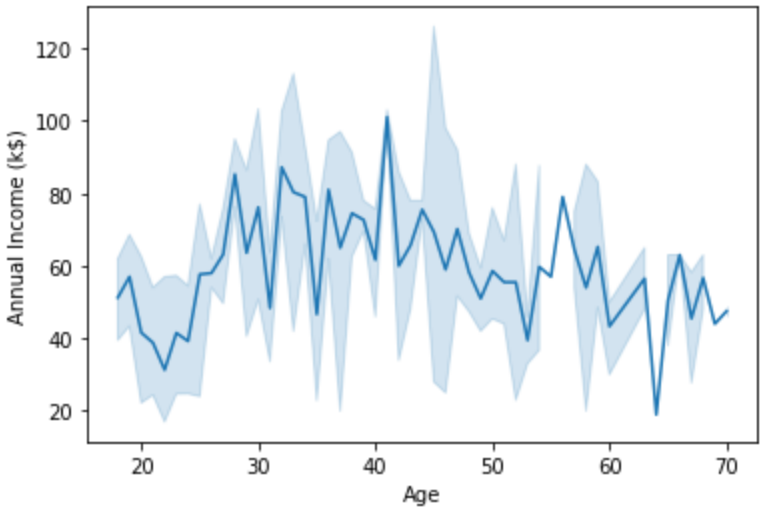



```
plt.hist(data['Spending Score (1-100)']);
plt.xlabel('Spending Score (1-100)');
plt.show();
```



3.2 Bi - Variate Analysis

```
sns.lineplot(x=data["Age"], y=data["Annual Income (k$)"]);
plt.xlabel('Age');
plt.ylabel('Annual Income (k$)');
plt.show();
```




```
sns.lineplot(x=data["Age"], y=data["Spending Score (1-100)"]);
plt.xlabel('Age');
plt.ylabel('Spending Score (1-100)');
plt.show();
```

3.3 Multi - Variate Analysis

```
sns.heatmap(data.corr(), annot=True);
```

4. Descriptive statistics

```
data.describe()
```

	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

```
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
```

```
data.columns
```

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

```
data.shape
```

```
(200, 5)
```

5. Handling missing values

```
data.isnull().any()
```

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

```
data.isnull().sum()
```

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

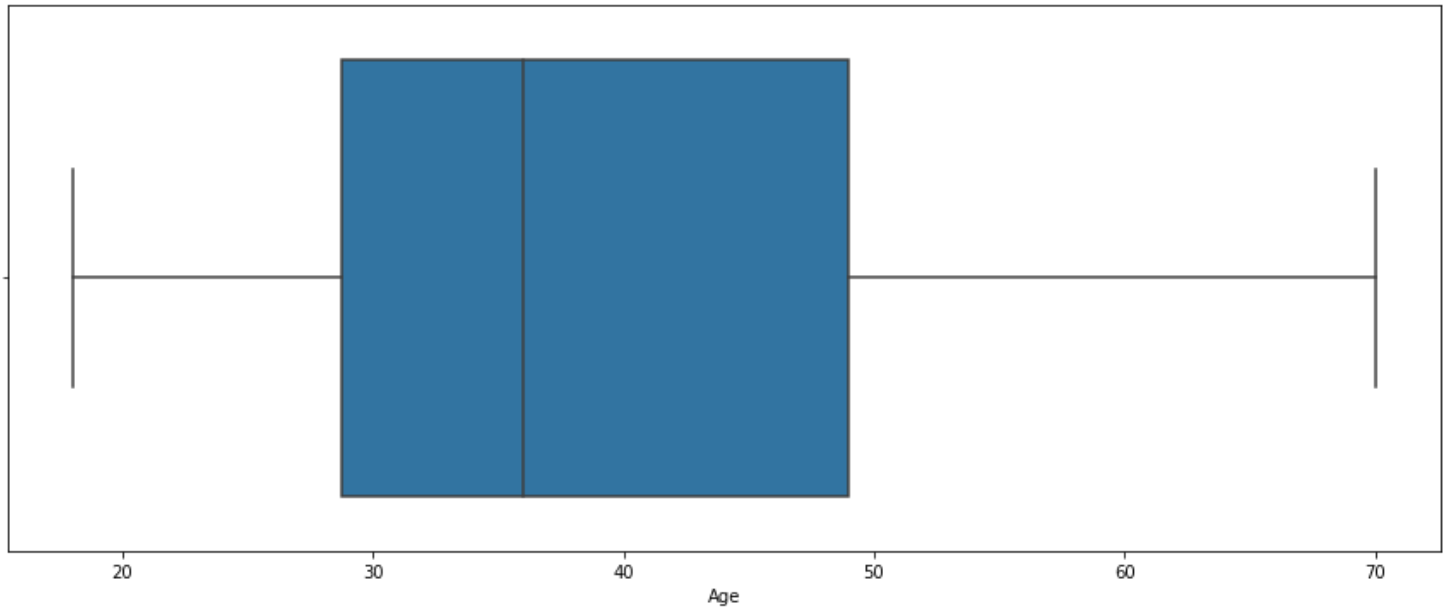
6. Finding outliers and replacing them

```
cols=['Age', 'Annual Income (k$)', 'Spending Score (1-100)']
fig, axes = plt.subplots(3, 1, figsize=(15, 20))
for i in range(3):
```

```
    sns.boxplot(ax=axes[i], data=data, x=cols[i])
plt.show()
```

```
for col in cols:
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1-(1.5*IQR)
    upper = Q3+(1.5*IQR)
    data[col]=np.where(data[col]>upper,upper,np.where(data[col]<lower,lower,data[col]))
```

```
fig, axes = plt.subplots(3, 1, figsize=(15, 20))
for i in range(3):
    sns.boxplot(ax=axes[i], data=data, x=cols[i])
plt.show()
```



7. Encoding the categorical columns

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

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

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

```
data
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19.0	15.00	39.0
1	2	1	21.0	15.00	81.0
2	3	0	20.0	16.00	6.0
3	4	0	23.0	16.00	77.0
4	5	0	31.0	17.00	40.0

8. Scaling the data

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaled_data=scaler.fit_transform(data)
scaled_data
```

```
array([[ 0.70148935, -0.88640526,  1.30256542,  0.55535083, -1.75473454],
       [ 0.71881007,  1.12815215, -0.49160182,  0.55535083,  1.6615628 ],
       [ 0.73613079, -0.88640526, -0.77866858,  0.59369717, -0.39597992],
       [ 0.75345152, -0.88640526, -0.49160182,  0.59369717,  1.42863343],
       [ 0.77077224,  1.12815215, -0.99396865,  0.6320435 , -1.48298362],
       [ 0.78809297,  1.12815215, -0.77866858,  0.6320435 ,  1.81684904],
       [ 0.80541369,  1.12815215,  0.65666521,  0.6320435 , -0.55126616],
       [ 0.82273442, -0.88640526, -0.49160182,  0.6320435 ,  0.92395314],
       [ 0.84005514, -0.88640526, -0.34806844,  0.67038984, -1.09476801],
       [ 0.85737587,  1.12815215, -0.34806844,  0.67038984,  1.54509812],
       [ 0.87469659,  1.12815215,  0.29783176,  0.67038984, -1.28887582],
       [ 0.89201732,  1.12815215,  0.010765  ,  0.67038984,  1.46745499],
       [ 0.90933804, -0.88640526,  0.36959845,  0.67038984, -1.17241113],
       [ 0.92665877, -0.88640526, -0.06100169,  0.67038984,  1.00159627],
       [ 0.94397949, -0.88640526,  0.58489852,  0.67038984, -1.32769738],
       [ 0.96130021, -0.88640526, -0.85043527,  0.67038984,  1.50627656],
       [ 0.97862094,  1.12815215, -0.13276838,  0.67038984, -1.91002079],
       [ 0.99594166, -0.88640526, -0.6351352 ,  0.67038984,  1.07923939],
       [ 1.01326239,  1.12815215, -0.34806844,  0.67038984, -1.91002079],

       [ 1.03058311, -0.88640526, -0.6351352 ,  0.67038984,  0.88513158],
       [ 1.04790384, -0.88640526,  1.23079873,  0.70873618, -0.59008772],
       [ 1.06522456, -0.88640526, -0.70690189,  0.70873618,  1.27334719],
       [ 1.08254529,  1.12815215, -1.42456879,  0.78542885, -1.75473454],
       [ 1.09986601, -0.88640526, -0.56336851,  0.78542885,  1.6615628 ],
       [ 1.11718674,  1.12815215,  0.80019859,  0.9388142 , -0.93948177],
       [ 1.13450746, -0.88640526, -0.20453507,  0.9388142 ,  0.96277471],
       [ 1.15182818,  1.12815215,  0.22606507,  0.97716054, -1.17241113],
       [ 1.16914891, -0.88640526, -0.41983513,  0.97716054,  1.73920592],
       [ 1.18646963, -0.88640526, -0.20453507,  1.01550688, -0.90066021],
       [ 1.20379036,  1.12815215, -0.49160182,  1.01550688,  0.49691598],
       [ 1.22111108,  1.12815215,  0.08253169,  1.01550688, -1.44416206],
       [ 1.23843181,  1.12815215, -0.77866858,  1.01550688,  0.96277471],
       [ 1.25575253,  1.12815215, -0.20453507,  1.01550688, -1.56062674],
       [ 1.27307326,  1.12815215, -0.20453507,  1.01550688,  1.62274124],
       [ 1.29039398, -0.88640526,  0.94373197,  1.05385321, -1.44416206],
       [ 1.30771471, -0.88640526, -0.6351352 ,  1.05385321,  1.38981187],
       [ 1.32503543,  1.12815215,  1.37433211,  1.05385321, -1.36651894],
       [ 1.34235616,  1.12815215, -0.85043527,  1.05385321,  0.72984534],
       [ 1.35967688,  1.12815215,  1.4460988 ,  1.2455849 , -1.4053405 ],
       [ 1.3769976 ,  1.12815215, -0.27630176,  1.2455849 ,  1.54509812],
```

```
[ 1.39431833, -0.88640526, -0.13276838,  1.39897025, -0.7065524 ],
[ 1.41163905, -0.88640526, -0.49160182,  1.39897025,  1.38981187],
[ 1.42895978,  1.12815215,  0.51313183,  1.43731659, -1.36651894],
[ 1.4462805 , -0.88640526, -0.70690189,  1.43731659,  1.46745499],
[ 1.46360123, -0.88640526,  0.15429838,  1.47566292, -0.43480148],
[ 1.48092195,  1.12815215, -0.6351352 ,  1.47566292,  1.81684904],
[ 1.49824268, -0.88640526,  1.08726535,  1.5523556 , -1.01712489],
[ 1.5155634 ,  1.12815215, -0.77866858,  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],
```

9. Clustering

```
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=5,random_state=42)
kmeans.fit(scaled_data)
```

```
KMeans(n_clusters=5, random_state=42)
```

```
kmeans.labels_
```

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

```
kmeans.cluster_centers_
```

```
array([[ 1.08067278,  0.14809719,  0.07477313,  1.02768443, -1.21752808],
       [-0.55195376,  1.12815215,  1.23079873, -0.50812092, -0.4128026 ],
       [-0.8380566 , -0.11157549, -1.01743084, -0.80594414,  0.45062873],
       [ 1.04747082,  0.07050951, -0.45213014,  0.97500356,  1.21414431],
       [-0.53039438, -0.88640526,  0.76344004, -0.48467666, -0.35526462]])
```

10. Add cluster data with primary dataset

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

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	1	19.0	15.0	39.0	2
1	2	1	21.0	15.0	81.0	2
2	3	0	20.0	16.0	6.0	2
3	4	0	23.0	16.0	77.0	2
4	5	0	31.0	17.0	40.0	2

11. Split data into dependent and independent variables

```
x=data.drop('Cluster',axis=1)
y=data['Cluster']
x,y
```

```
(     CustomerID  Gender   Age  Annual Income (k$)  Spending Score (1-100)
 0             1       1  19.0               15.00                    39.0
 1             2       1  21.0               15.00                    81.0
 2             3       0  20.0               16.00                     6.0
 3             4       0  23.0               16.00                    77.0
 4             5       0  31.0               17.00                    40.0
 ..          ...     ...   ...                 ...                     ...
 195         196       0  35.0              120.00                    79.0
 196         197       0  45.0              126.00                    28.0
 197         198       1  32.0              126.00                    74.0
 198         199       1  32.0              132.75                    18.0
 199         200       1  30.0              132.75                    83.0

 [200 rows x 5 columns], 0      2
 1      2
 2      2
 3      2
 4      2
       ..
 195    3
 196    0
 197    3
 198    0
 199    3
 Name: Cluster, Length: 200, dtype: int32)
```

12. Split data into training and testing

```
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(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(160, 5) (40, 5) (160,) (40,)
```

13. Build the model

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
```

14. Train the model

```
model.fit(x_train,y_train)
```

```
RandomForestClassifier()
```

15. Test the model

```
pred=model.predict(x_test)
pred
```

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

16. Performance and evaluation metrics

```
from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(y_test,pred))
```

```
Accuracy: 0.975
```

```
from sklearn.metrics import confusion_matrix,classification_report
print(confusion_matrix(y_test,pred))
```

```
[[11  0  0  0  0]
 [ 1  6  0  0  0]
 [ 0  0 11  0  0]
 [ 0  0  0  5  0]
 [ 0  0  0  0  6]]
```

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        11
```

```
              1       1.00      0.86      0.92         7
              2       1.00      1.00      1.00        11
              3       1.00      1.00      1.00         5
              4       1.00      1.00      1.00         6

       accuracy                           0.97        40
      macro avg       0.98      0.97      0.98        40
   weighted avg       0.98      0.97      0.97        40
```