# ASSIGNMENT – 3

## Abalone Age Prediction

| | |
|---|---|
| **ASSIGNMENT DATE** | 29-09-2022 |
| **STUDENT NAME** | Fargath Shahani M |
| **STUDENT ROLL NO.** | 913219104004 |
| **MAXIMUM MARK** | 2 Mark |

# 1. Download the dataset

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

# 2. Load the dataset

In [8]:

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

Out[8]:

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0** | M | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.1500 | 15 |
| **1** | M | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.0700 | 7 |
| **2** | F | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.2100 | 9 |
| **3** | M | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.1550 | 10 |
| **4** | I | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.0550 | 7 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4172** | F | 0.565 | 0.450 | 0.165 | 0.8870 | 0.3700 | 0.2390 | 0.2490 | 11 |
| **4173** | M | 0.590 | 0.440 | 0.135 | 0.9660 | 0.4390 | 0.2145 | 0.2605 | 10 |
| **4174** | M | 0.600 | 0.475 | 0.205 | 1.1760 | 0.5255 | 0.2875 | 0.3080 | 9 |

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **4175** | F | 0.625 | 0.485 | 0.150 | 1.0945 | 0.5310 | 0.2610 | 0.2960 | 10 |
| **4176** | M | 0.710 | 0.555 | 0.195 | 1.9485 | 0.9455 | 0.3765 | 0.4950 | 12 |

4177 rows × 9 columns

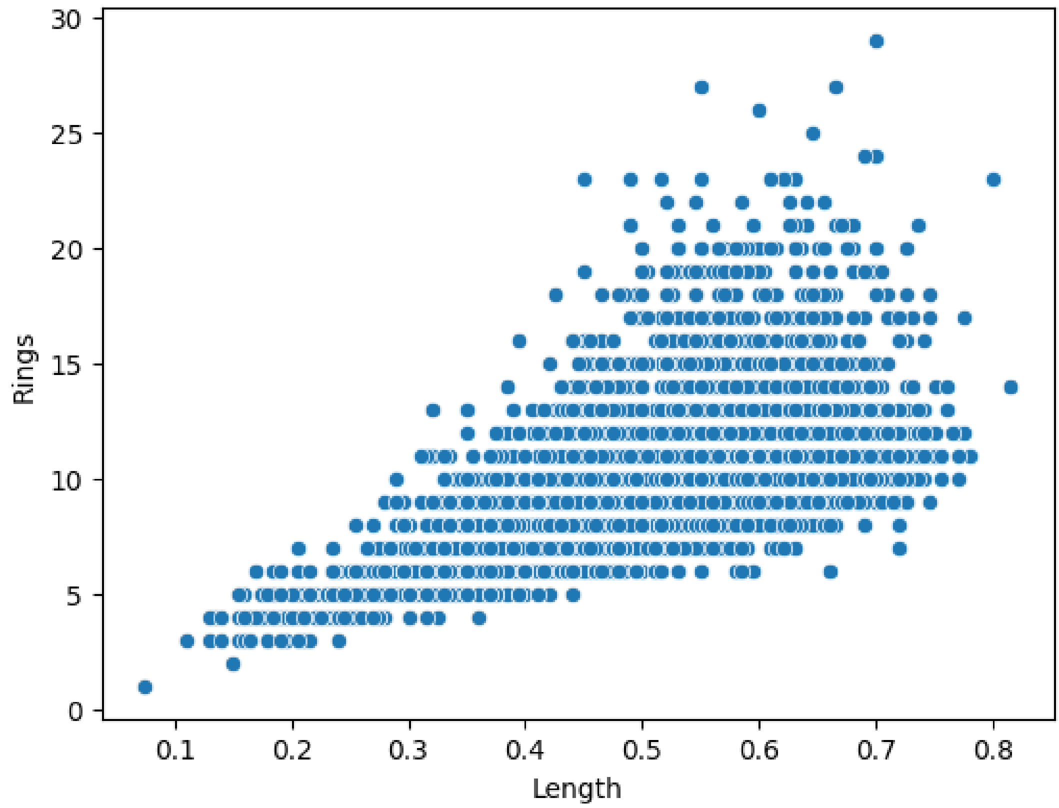
## 3. perform the visualizations

# Bivariate analysis

In [9]:

```
sns.scatterplot(x=data.Length,y=data.Rings)
```

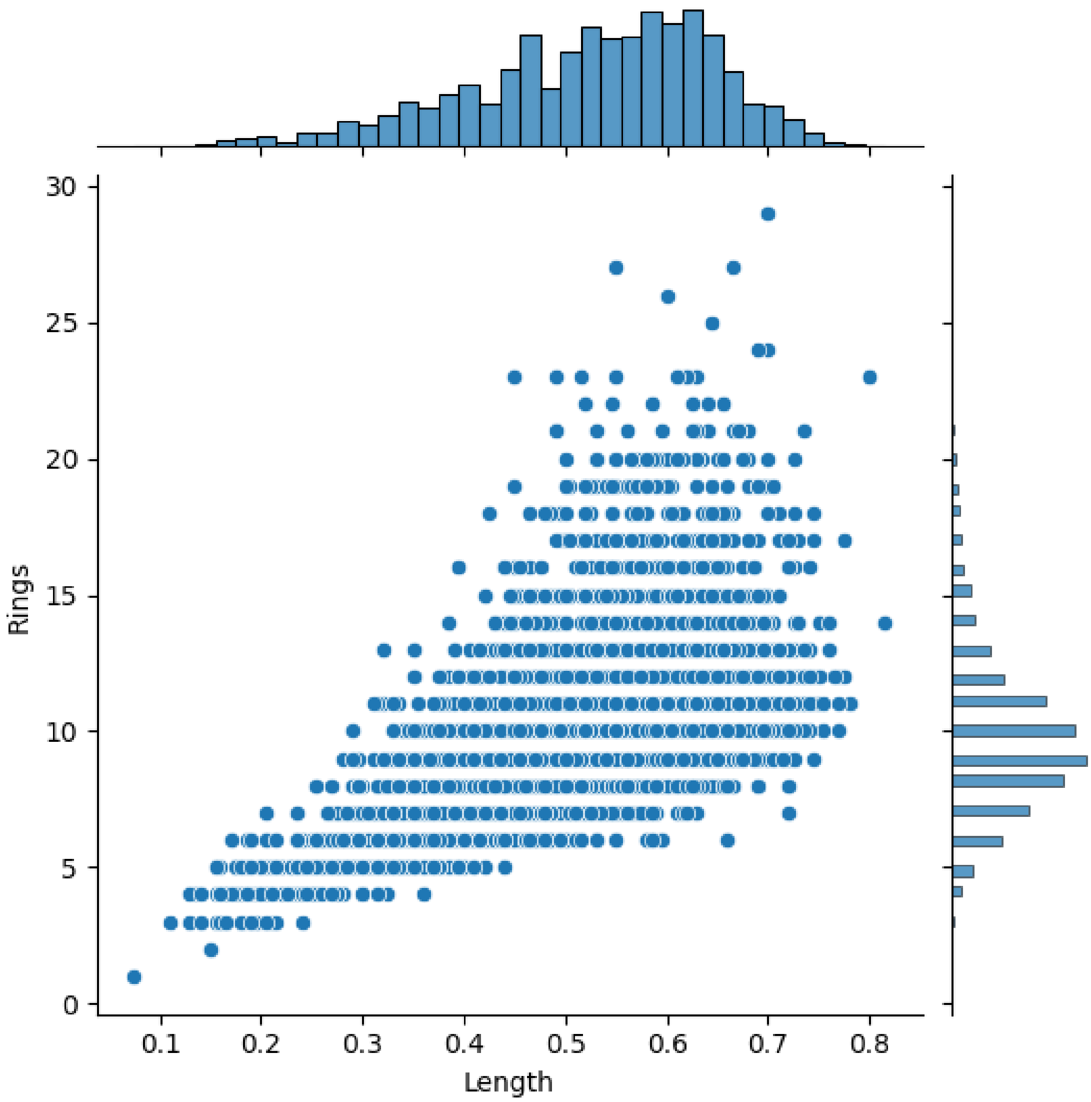
Out[9]:




In [10]:

```
sns.jointplot(x=data.Length,y=data.Rings)
```

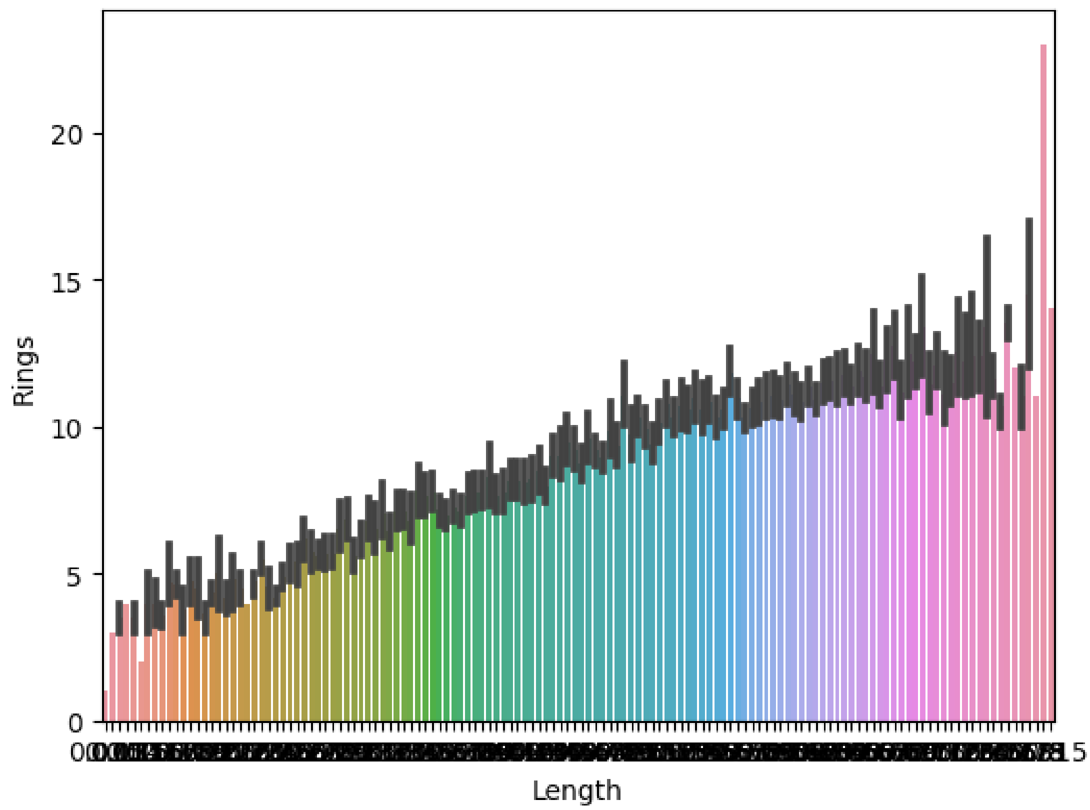
Out[10]:




In [21]:

```
sns.barplot(x=data.Length,y=data.Rings)
```

Out[21]:




## Univariate analysis

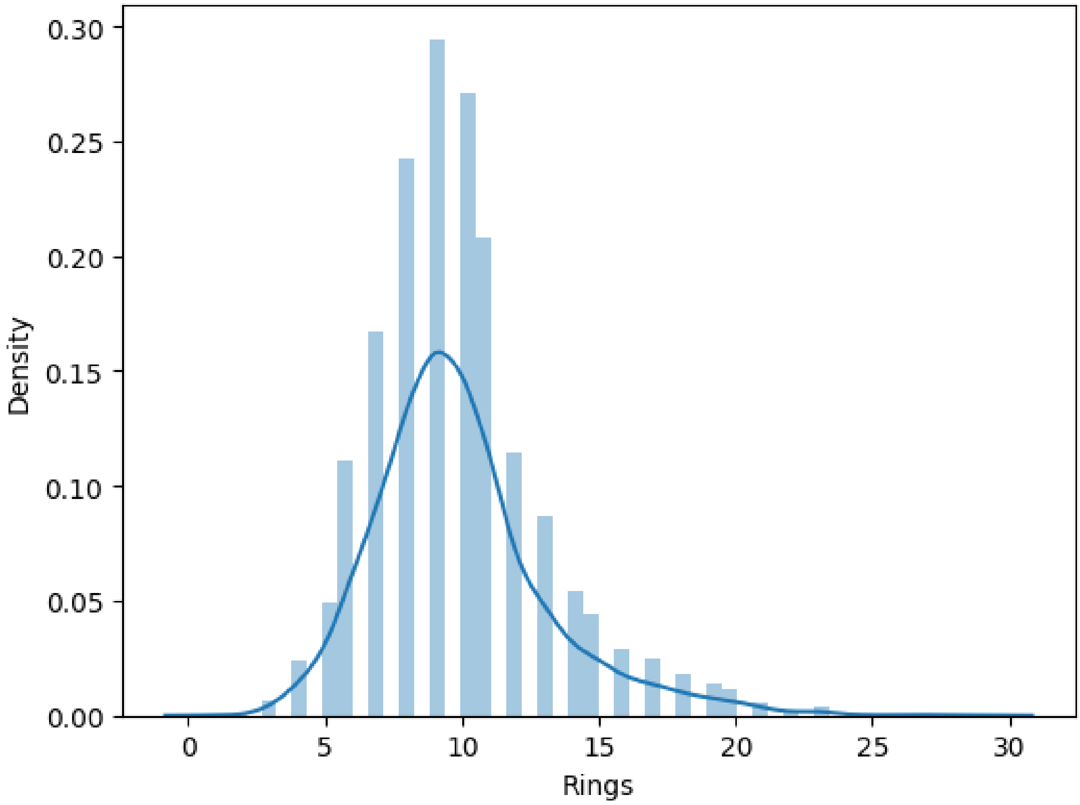
In [22]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [23]:

```
sns.distplot(data.Rings)
```

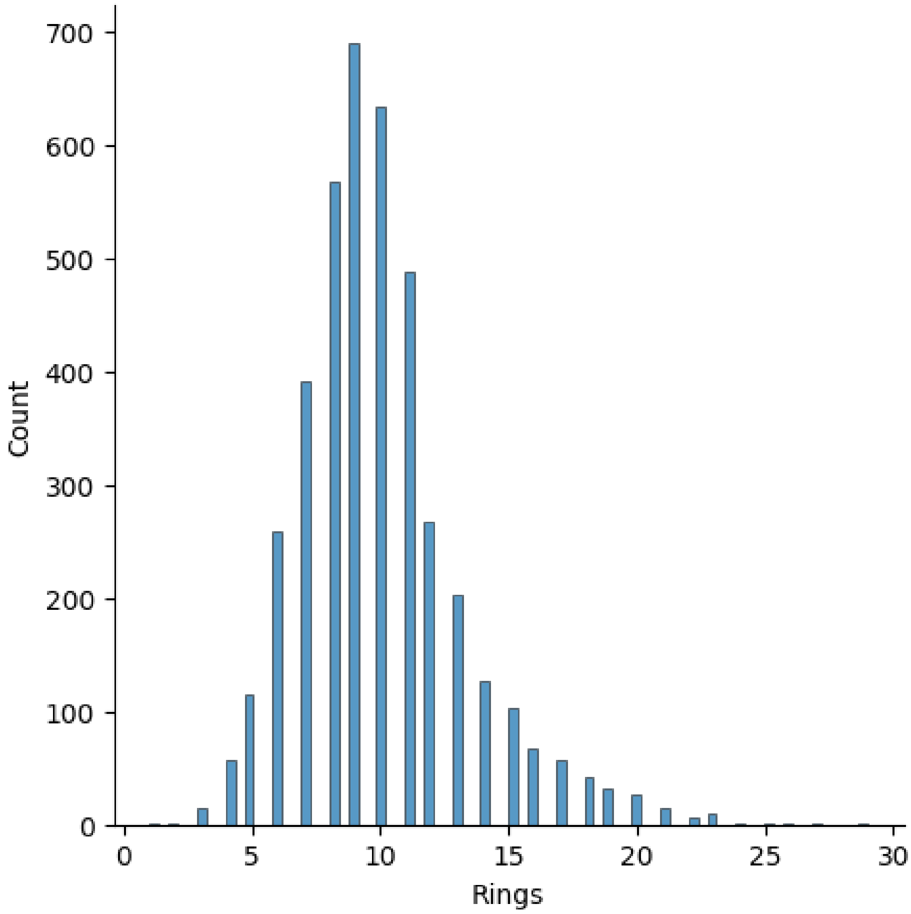
Out[23]:




In [24]:

```
sns.displot(data.Rings)
```

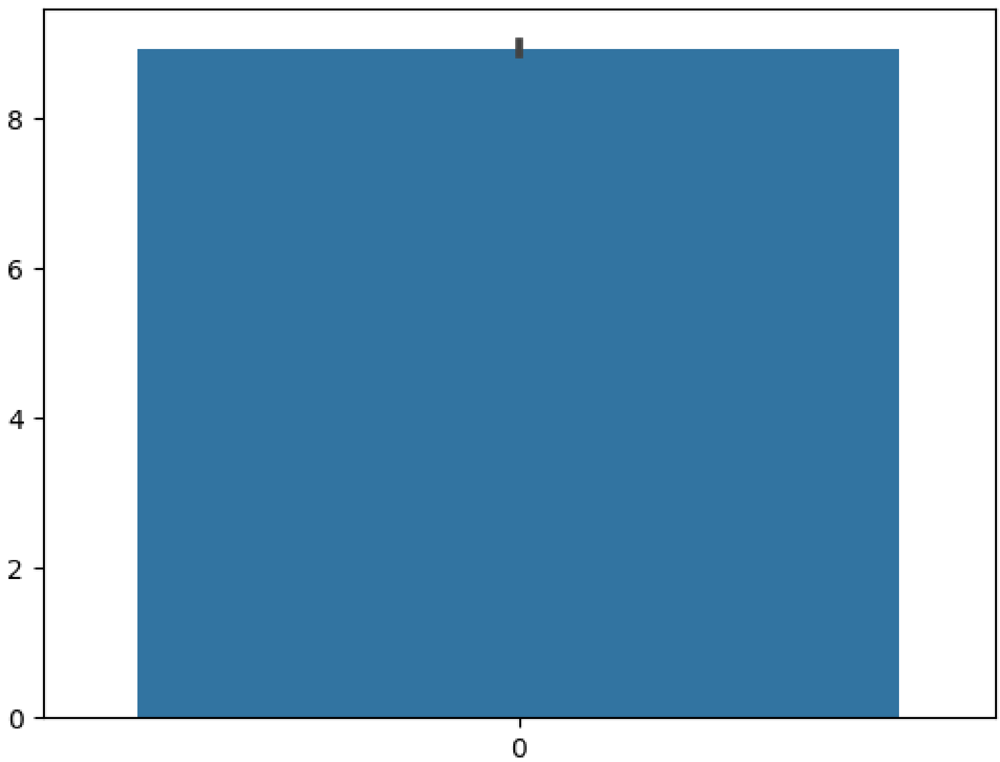
Out[24]:




In [111]:

```
sns.barplot(data.Rings)
```

Out[111]:



# 4. perform descriptive statistics

In [27]:

```
data.head()
```

Out[27]:

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0** | M | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 15 |
| **1** | M | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 7 |
| **2** | F | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 9 |
| **3** | M | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 10 |
| **4** | I | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 7 |

In [67]:

```
data.tail()
```

Out[67]:

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **4172** | 0 | 0.565 | 73 | 0.165 | 0.8870 | 0.3700 | 0.2390 | 0.2490 | 10 |
| **4173** | 2 | 0.590 | 71 | 0.135 | 0.9660 | 0.4390 | 0.2145 | 0.2605 | 9 |
| **4174** | 2 | 0.600 | 78 | 0.205 | 1.1760 | 0.5255 | 0.2875 | 0.3080 | 8 |
| **4175** | 0 | 0.625 | 80 | 0.150 | 1.0945 | 0.5310 | 0.2610 | 0.2960 | 9 |
| **4176** | 2 | 0.710 | 94 | 0.195 | 1.9485 | 0.9455 | 0.3765 | 0.4950 | 11 |

In [68]:

```
data.info()
```

```
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   int32
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   int64
 3   Height          4177 non-null   float64
```

```
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(6), int32(1), int64(2)
memory usage: 277.5 KB
```

In [69]:

```
data.shape
```

Out[69]:

```
(4177, 9)
```

# measure of tendency

In [70]:

```
data.mean()
```

Out[70]:

```
Sex                1.052909
Length             0.523992
Diameter          64.576969
Height             0.139516
Whole weight       0.828742
Shucked weight     0.359367
Viscera weight     0.180594
Shell weight       0.238831
Rings              8.933445
dtype: float64
```

In [71]:

```
data.mode()
```

Out[71]:

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2.0 | 0.550 | 73.0 | 0.15 | 0.2225 | 0.175 | 0.1715 | 0.275 | 8.0 |
| **1** | NaN | 0.625 | NaN | NaN | NaN | NaN | NaN | NaN | NaN |

In [72]:

```
data.median()
```

Out[72]:

```
Sex                1.0000
Length             0.5450
Diameter          68.0000
Height             0.1400
Whole weight       0.7995
Shucked weight     0.3360
Viscera weight     0.1710
Shell weight       0.2340
Rings              8.0000
dtype: float64
```

In [73]:

```
data.skew()
```

Out[73]:

```
Sex              -0.098155
Length           -0.639873
Diameter         -0.607999
Height            3.128817
Whole weight      0.530959
Shucked weight    0.719098
Viscera weight    0.591852
Shell weight      0.620927
Rings             1.108353
dtype: float64
```

In [74]:

```
data.kurtosis()
```

Out[74]:

```
Sex               -1.514387
Length             0.064621
Diameter          -0.054859
Height            76.025509
Whole weight      -0.023644
Shucked weight     0.595124
Viscera weight     0.084012
Shell weight       0.531926
Rings              2.283203
dtype: float64
```

In [75]:

```
data.std()
```

Out[75]:

```
Sex                0.822240
Length             0.120093
Diameter          19.841382
Height             0.041827
Whole weight       0.490389
Shucked weight     0.221963
Viscera weight     0.109614
Shell weight       0.139203
Rings              3.222790
dtype: float64
```

In [76]:

```
data.var()
```

Out[76]:

```
Sex                 0.676079
Length              0.014422
Diameter          393.680437
Height              0.001750
Whole weight        0.240481
Shucked weight      0.049268
Viscera weight      0.012015
Shell weight        0.019377
Rings              10.386374
dtype: float64
```

# 5.check the missing values and deal with them

In [77]:

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

Out[77]:

```
Sex               False
Length            False
Diameter          False
Height            False
Whole weight      False
Shucked weight    False
Viscera weight    False
Shell weight      False
Rings             False
dtype: bool
```

In [78]:

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

Out[78]:

```
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked weight    0
Viscera weight    0
Shell weight      0
Rings             0
dtype: int64
```

In [79]:

```
data.dropna()
```

Out[79]:

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 0.455 | 56 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.1500 | 14 |
| **1** | 2 | 0.350 | 36 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.0700 | 6 |
| **2** | 0 | 0.530 | 67 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.2100 | 8 |
| **3** | 2 | 0.440 | 56 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.1550 | 9 |
| **4** | 1 | 0.330 | 34 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.0550 | 6 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **4172** | 0 | 0.565 | 73 | 0.165 | 0.8870 | 0.3700 | 0.2390 | 0.2490 | 10 |

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **4173** | 2 | 0.590 | 71 | 0.135 | 0.9660 | 0.4390 | 0.2145 | 0.2605 | 9 |
| **4174** | 2 | 0.600 | 78 | 0.205 | 1.1760 | 0.5255 | 0.2875 | 0.3080 | 8 |
| **4175** | 0 | 0.625 | 80 | 0.150 | 1.0945 | 0.5310 | 0.2610 | 0.2960 | 9 |
| **4176** | 2 | 0.710 | 94 | 0.195 | 1.9485 | 0.9455 | 0.3765 | 0.4950 | 11 |

4177 rows × 9 columns

# 6. Find the outliers and replace them outliers

In [80]:

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

Out[80]:

| | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0.25** | 0.0 | 0.450 | 53.0 | 0.115 | 0.4415 | 0.186 | 0.0935 | 0.130 | 7.0 |
| **0.75** | 2.0 | 0.615 | 79.0 | 0.165 | 1.1530 | 0.502 | 0.2530 | 0.329 | 10.0 |

## 7.Check the categorical columns and perform the encoding

In [81]:

```
from sklearn.preprocessing import LabelEncoder
```

In [82]:

```
le=LabelEncoder()
```

In [83]:

```
data["Sex"]=le.fit_transform(data['Sex'])
data["Rings"]=le.fit_transform(data['Rings'])
data["Diameter"]=le.fit_transform(data['Diameter'])
data.head()
```

Out[83]:

|   | Sex | Length | Diameter | Height | Whole weight | Shucked weight | Viscera weight | Shell weight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 0.455 | 56 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 14 |
| **1** | 2 | 0.350 | 36 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 6 |
| **2** | 0 | 0.530 | 67 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 8 |
| **3** | 2 | 0.440 | 56 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 9 |
| **4** | 1 | 0.330 | 34 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 6 |

# 8. split the data into dependent and independent variables

In [84]:

```
x=data.iloc[:,:-1].values
y=data.iloc[:,-1].values
x
```

Out[84]:

```
array([[2.000e+00, 4.550e-01, 5.600e+01, ..., 2.245e-01, 1.010e-01,
        1.500e-01],
       [2.000e+00, 3.500e-01, 3.600e+01, ..., 9.950e-02, 4.850e-02,
        7.000e-02],
       [0.000e+00, 5.300e-01, 6.700e+01, ..., 2.565e-01, 1.415e-01,
        2.100e-01],
       ...,
       [2.000e+00, 6.000e-01, 7.800e+01, ..., 5.255e-01, 2.875e-01,
        3.080e-01],
       [0.000e+00, 6.250e-01, 8.000e+01, ..., 5.310e-01, 2.610e-01,
        2.960e-01],
       [2.000e+00, 7.100e-01, 9.400e+01, ..., 9.455e-01, 3.765e-01,
        4.950e-01]])
```

In [85]:

```
y
```

Out[85]:

```
array([14,  6,  8, ...,  8,  9, 11], dtype=int64)
```

In [86]:

```
print(x.shape,y.shape)
```

```
(4177, 8) (4177,)
```

# 9. Scale the independent variables

In [87]:

```
from sklearn.preprocessing import scale
```

In [88]:

```
x=scale(x)
x
```

Out[88]:

```
array([[ 1.15198011, -0.57455813, -0.43232856, ..., -0.60768536,
         -0.72621157, -0.63821689],
       [ 1.15198011, -1.44898585, -1.44044354, ..., -1.17090984,
         -1.20522124, -1.21298732],
       [-1.28068972,  0.05003309,  0.12213469, ..., -0.4634999 ,
         -0.35668983, -0.20713907],
       ...,
       [ 1.15198011,  0.6329849 ,  0.67659793, ...,  0.74855917,
          0.97541324,  0.49695471],
       [-1.28068972,  0.84118198,  0.77740943, ...,  0.77334105,
          0.73362741,  0.41073914],
       [ 1.15198011,  1.54905203,  1.48308992, ...,  2.64099341,
          1.78744868,  1.84048058]])
```

# 10. Split the data into training and testing

In [89]:

```
from sklearn.model_selection import train_test_split
```

In [90]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
te=0)
```

In [91]:

```
x_train.shape
```

Out[91]:

```
(3341, 8)
```

In [92]:

```
x_test.shape
```

Out[92]:

```
(836, 8)
```

# 11. Build the model

In [93]:

```
from sklearn.linear_model import LinearRegression
```

In [94]:

```
regressor=LinearRegression()

regressor.fit(x_train,y_train)
```

Out[94]:

LinearRegression()
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [95]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [96]:

```
model=DecisionTreeClassifier()
```

In [97]:

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

Out[97]:

DecisionTreeClassifier()
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

# 12. Train and the model

In [98]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [99]:

```
knn=KNeighborsClassifier(n_neighbors=5)
```

In [100]:

```
knn.fit(x_train,y_train)
knn.fit(x_test,y_test)
```

Out[100]:

KNeighborsClassifier()
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

# 14. Measure the performance using metrics

In [101]:

```
from sklearn.naive_bayes import GaussianNB
```

In [102]:

```
nb=GaussianNB()
nb.fit(x_train,y_train)
```

Out[102]:

GaussianNB()
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [103]:

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [104]:

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

Out[104]:

```
array([ 8,  7, 10,  3, 10, 10,  6,  7,  5, 10,  6,  4,  6,  7,  3,  9,  5,
        9, 10,  6,  5,  4,  6,  5,  8,  8,  2, 10,  8,  8,  5,  2,  9, 10,
        6,  7,  6,  9,  6, 10,  9,  7,  8,  8,  9,  6,  9, 10,  8,  5,  8,
```

```
         5,  6, 10,  7,  6,  3,  5,  5,  5,  9,  8,  8,  6,  5, 10, 10, 10,
         7, 10,  7, 26, 10,  8,  8,  7,  8,  7, 10,  7,  9,  4,  5,  9,  8,
         6,  9, 10,  2,  5,  8,  8,  6,  5,  8,  4,  7,  5,  9, 10, 10,  7,
        10,  9,  3, 10,  7,  5,  5,  9, 10, 10,  7,  8,  8, 10, 10,  9,  4,
         7,  5,  8,  5,  6,  8,  9, 10, 10,  5, 10, 10,  7,  7, 10, 10,  9,
         7,  7,  5, 10,  7,  7,  5,  8,  9,  8,  6,  7,  6,  4,  7, 23,  5,
        10,  5,  5,  9,  5,  9,  5,  8,  4,  7,  9,  3,  9,  5,  4,  8,  8,
         8,  2,  6,  9, 10, 10,  3,  6,  6,  8,  7,  9,  8,  7,  8,  8,  9,
         7,  4,  7,  8,  7,  6,  9,  8,  7,  5,  7,  2, 10,  6,  7, 10,  8,
        10,  9,  3, 10,  7,  7,  3,  8,  8,  8,  9,  8,  8, 10,  3, 10,  8,
        10,  6,  9, 10,  9, 10,  7, 11,  9,  8,  7,  4, 10,  7,  9,  7,  3,
         7,  8,  7,  9,  8, 10,  8,  8, 10,  6,  9, 10, 10,  6, 10, 10,  8,
         7,  5,  7,  8, 10, 10,  7,  9,  4,  7,  6,  6,  6,  5,  5,  8,  6,
         5,  6,  4, 10, 10,  6,  9,  7,  4, 10,  5, 10, 10,  8,  6, 10,  8,
         9,  7,  7,  4,  7,  5,  7,  4,  3,  8, 10,  6,  6, 23,  7,  6,  6,
         8,  8,  8,  8, 10, 10,  6,  8, 10,  9,  7,  4,  6,  8, 10,  8,  6,
        10,  9,  7,  7,  8, 10, 10,  7, 23,  2, 10,  8, 10,  6,  7, 10,  8,
         6,  7, 10,  9,  8,  6,  4,  8,  9,  5,  4,  8, 10, 10, 10,  7, 10,
        10,  6,  7,  7,  8,  6,  9, 10,  3,  4,  8,  8,  7,  9,  5,  9,  6,
        10,  9, 10,  6,  7,  2,  7,  8,  9, 10,  3, 10, 10,  7,  6,  4, 10,
         8,  7,  7, 19,  5, 10,  6,  8,  7,  6,  8,  5, 10,  6, 10,  3, 10,
        10,  7,  7,  6, 10,  8, 10,  8,  9,  9, 10, 10,  7,  7,  8,  9,  4,
         8,  8, 10,  9,  9,  7, 10,  7,  8,  6,  6, 10,  8,  8, 10,  6,  4,
         9,  6,  5,  5,  9,  3,  7,  9,  2,  4,  9,  4,  5,  6,  9,  8,  6,
         9,  6, 10,  7,  8,  7, 10,  4, 19,  6,  7,  8,  8,  6, 10,  7,  6,
         6,  6,  5, 10,  7, 10,  6,  8, 10, 10,  8,  7,  7, 10,  2,  8,  8,
         8,  4,  5,  6,  4,  9,  9,  6,  9,  5,  9,  9,  9,  7, 10,  5,  4,
         7,  9, 10,  9,  6,  6,  9, 10, 10,  8,  8,  8, 10,  9,  4, 10,  6,
         7,  7,  7,  6,  6,  6,  8,  3,  8, 10, 10,  9, 10,  6,  8,  6,  9,
         5,  8,  7,  6,  9,  9,  8, 10,  6, 10,  9,  6,  7,  8,  9,  8,  8,
         9, 10,  8, 10,  3,  6,  8,  8,  6,  8,  8,  7,  8,  7,  6,  8,  8,
        10,  2,  9,  4,  3,  4,  7,  8, 26,  9,  7, 10,  4,  6, 10,  5,  9,
        10,  2,  6,  4, 10,  7, 10,  8, 10, 10,  7,  9,  4,  8, 10,  6, 10,
         8,  7,  8, 10,  7,  5, 10,  8,  7, 10,  5,  8,  9,  8,  6,  9,  6,
         8,  6,  5,  6,  5,  6,  8,  3, 10,  6,  8,  7, 10, 10,  9,  8,  7,
         5,  8, 10,  7,  6,  6,  6,  8,  5, 10,  6,  6,  9,  9,  7,  6,  9,
         8,  7,  6,  6,  4,  4,  8,  8, 10, 10, 10,  3,  6,  6,  4,  8, 10,
         9,  8,  9,  7,  7,  6,  8,  9,  7, 10,  9,  9,  8, 10,  7, 10, 10,
         8,  7,  6,  5,  8,  9,  9,  7,  8,  7,  6, 10,  9,  4,  4,  4,  8,
         5,  7,  9,  7,  6,  6,  9,  3,  7,  5,  9,  4, 10,  7,  8,  9,  8,
        10,  9, 10,  7,  8,  7, 10,  6,  4,  8,  8,  4,  6,  8,  5,  9,  8,
         7,  6,  8,  8,  5, 10,  6, 10,  7,  5,  8,  9,  5,  4,  5,  4,  8,
         8, 10,  9,  8,  6,  4,  6,  8, 10,  8,  7,  5,  7,  5,  8, 10,  8,
         6,  5,  6,  8,  7,  6,  9,  8,  8, 10, 10,  8,  9,  8,  8,  8,  2,
         7,  6,  6,  6,  4,  4,  6,  7, 10,  6,  7,  6,  7,  8,  8, 23,  7,
         8,  9,  8,  7,  4,  8, 10,  4,  6,  8,  9,  7,  4,  4,  8,  9,  6,
         6,  9,  3], dtype=int64)
```

In [105]:

```
accuracy_score(y_test,pred)
```

Out[105]:

```
0.24162679425837322
```