Assignment Date	01-October-2022
Student Name	MOHAMMED IJAZ J
Student Roll No	61771931032
Maximum marks	2 marks

Assignment -3
Python Programming

1. Download the dataset

```
In [1]: #importing the libraries

        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')
```

2. Load the dataset into the tool.

```
In [2]: #Loading the dataset

        d = pd.read_csv(r'Downloads/abalone.csv')
```

3. Perform Below Visualizations.

· Univariate Analysis

```
In [3]: d.head()
```

Out[3]:

	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 [4]: #Boxplot

        sns.boxplot(d['Diameter'])
```
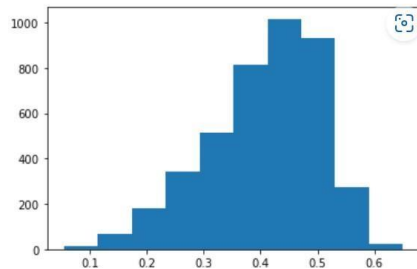
Out[4]: <AxesSubplot:xlabel='Diameter'>
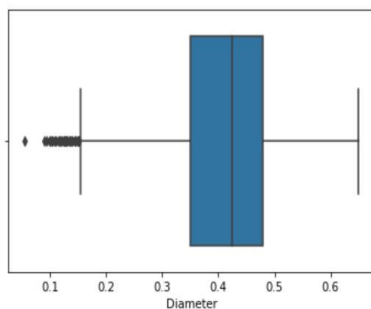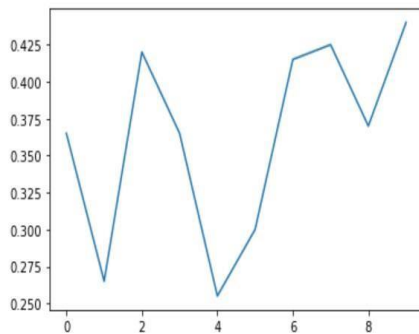
```
In [5]: #histogram

        plt.hist(d['Diameter'])

Out[5]: (array([  13.,   66.,  180.,  344.,  513.,  812., 1017.,  934.,  275.,
                   23.]),
         array([0.055 , 0.1145, 0.174 , 0.2335, 0.293 , 0.3525, 0.412 , 0.4715,
                0.531 , 0.5905, 0.65  ]),
         <BarContainer object of 10 artists>)
```



```
In [6]: #line plot

        plt.plot(d['Diameter'].head(10))

Out[6]: [<matplotlib.lines.Line2D at 0x1c2ed71d130>]
```



```
In [7]: #piechart

        plt.pie(d['Diameter'].head(),autopct='%.2f')

Out[7]: ([<matplotlib.patches.Wedge at 0x1c2ed77fd00>,
          <matplotlib.patches.Wedge at 0x1c2ed78d460>,
          <matplotlib.patches.Wedge at 0x1c2ed78db80>,
          <matplotlib.patches.Wedge at 0x1c2ed7992e0>,
          <matplotlib.patches.Wedge at 0x1c2ed799a30>],
         [Text(0.8507215626110558, 0.6973326486753676, ''),
          Text(-0.32611344931648134, 1.0505474849691026, ''),
          Text(-1.0998053664078908, -0.02069193128747144, ''),
          Text(-0.08269436219656089, -1.096887251480709, ''),
          Text(0.9758446362287218, -0.5076684409569241, '')],
         [Text(0.464029943242394, 0.3803632629138369, '21.86'),
          Text(-0.17788006326353525, 0.5730259008922377, '15.87'),
          Text(-0.5998938362224858, -0.011286507974984419, '25.15'),
          Text(-0.045106015743578656, -0.5983021371712958, '21.86'),
          Text(0.5322788924883937, -0.2769100587037768, '15.27')])
```
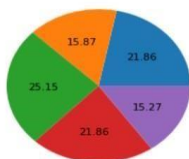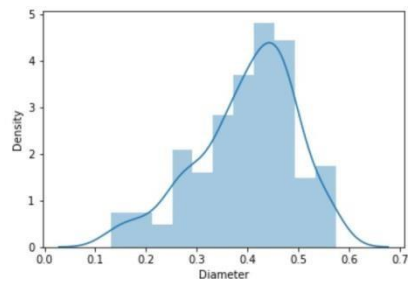
```
In [8]: #distplot

        sns.distplot(d['Diameter'].head(200))
```
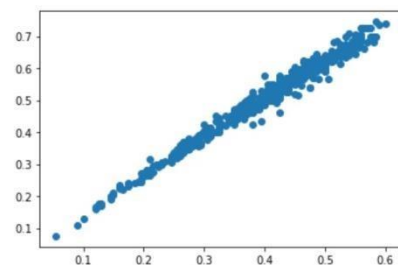
Out[8]: `<AxesSubplot:xlabel='Diameter', ylabel='Density'>`

● Bi - Variate Analysis

```
In [9]: #scatter plot

        plt.scatter(d['Diameter'].head(500),d['Length'].head(500))
```

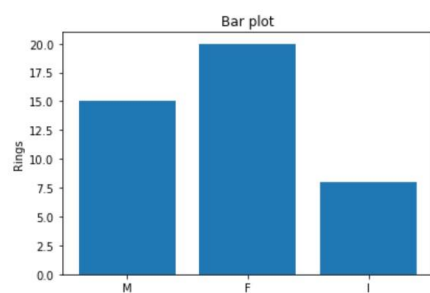Out[9]: `<matplotlib.collections.PathCollection at 0x1c2edcc2d60>`


```
In [10]: #bar plot

         plt.bar(d['Sex'].head(10),d['Rings'].head(10))

         #labelling of x,y and result

         plt.title('Bar plot')
         plt.xlabel('Diameter')
         plt.ylabel('Rings')
```

Out[10]: `Text(0, 0.5, 'Rings')`


```
In [11]: sns.barplot(d['Sex'], d['Rings'])
```

Out[11]: `<AxesSubplot:xlabel='Sex', ylabel='Rings'>`
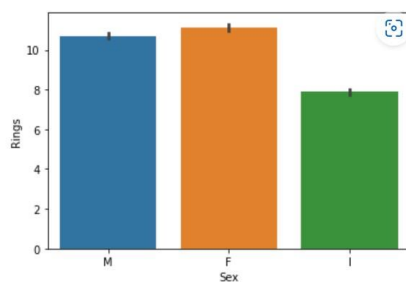
In [12]: #joint plot

sns.jointplot(d['Diameter'].head(50),d['Rings'].head(50))

Out[12]: <seaborn.axisgrid.JointGrid at 0x1c2edde3160>
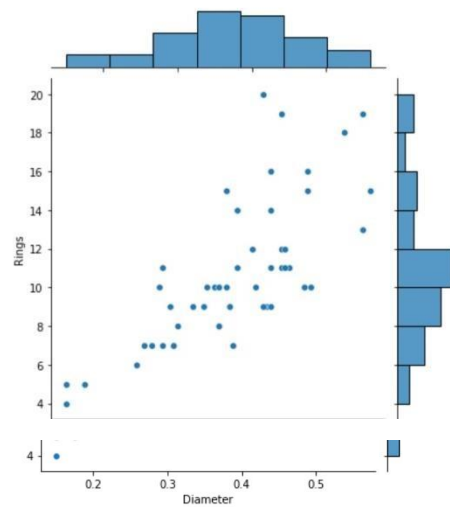
In [13]: #bar plot

sns.barplot('Diameter','Rings',hue='Sex',data=d.head())

Out[13]: <AxesSubplot:xlabel='Diameter', ylabel='Rings'>
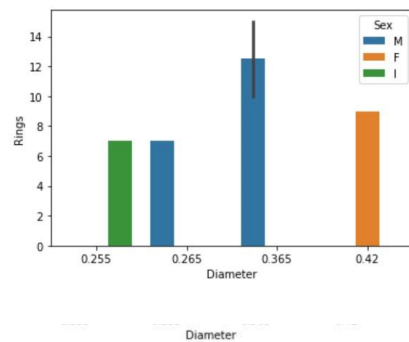
In [14]: sns.lineplot(d['Diameter'].head(),d['Rings'].head())

Out[14]: <AxesSubplot:xlabel='Diameter', ylabel='Rings'>
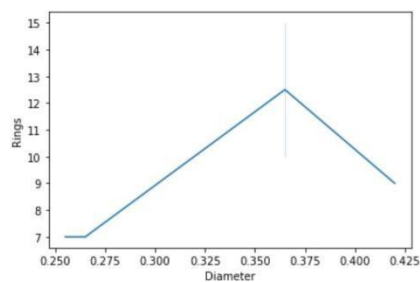
● Multi - Variate Analysis

In [15]:
```
#boxplot
sns.boxplot(d['Sex'].head(10),d['Diameter'].head(10),d['Rings'].head(10))
```

Out[15]: `<AxesSubplot:xlabel='Sex', ylabel='Diameter'>`
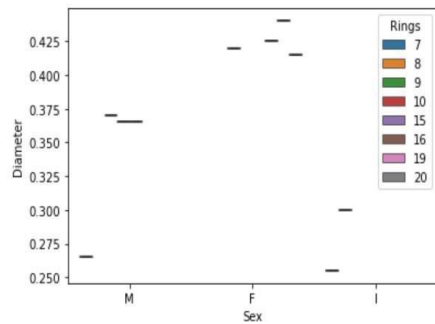
In [16]:
```
#heat map
fig=plt.figure(figsize=(8,5))
sns.heatmap(d.head().corr(),annot=True)
```

Out[16]: `<AxesSubplot:>`

In [17]:
```
#pair plot
sns.pairplot(d.head(),hue='Rings')
```

Out[17]: `<seaborn.axisgrid.PairGrid at 0x1c2edd07fd0>`
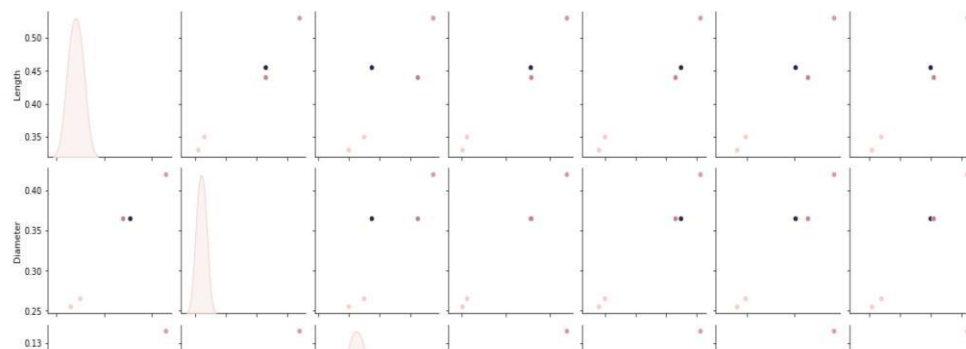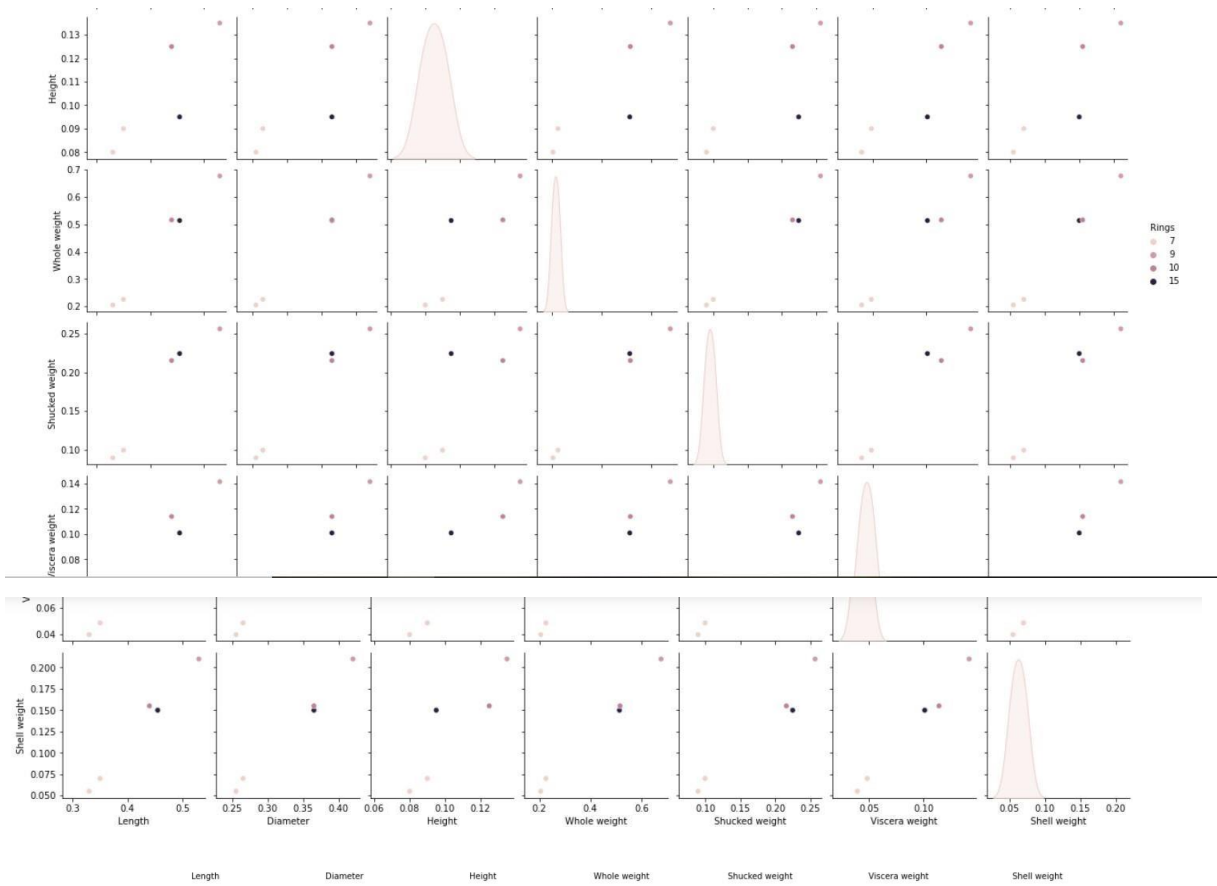
```python
In [18]: sns.pairplot(d.head())
```

Out[18]: `<seaborn.axisgrid.PairGrid at 0x1c2f243e910>`
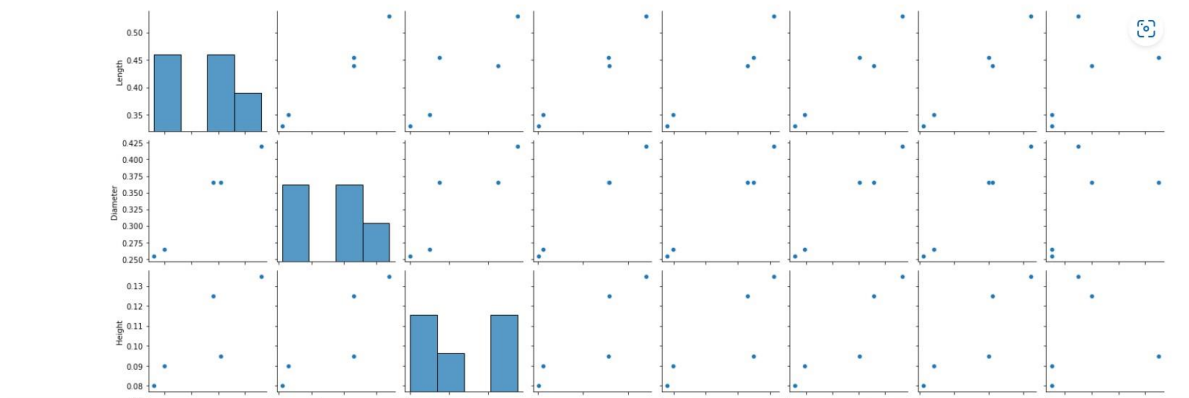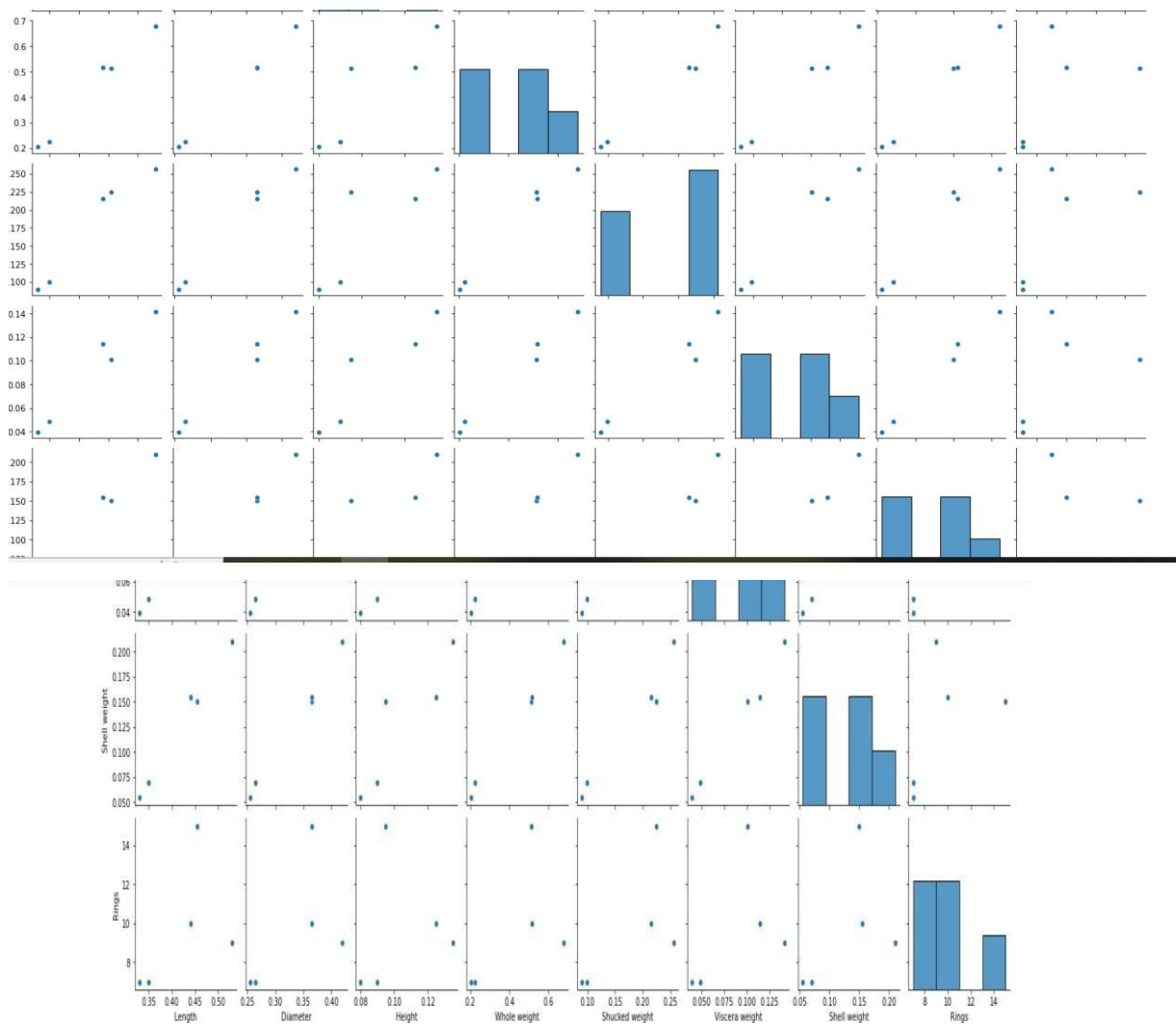
4. Perform descriptive statistics on the dataset.

In [19]: `#head`

`d.head()`

Out[19]:

	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 [20]: `#tail`

`d.tail()`

Out[20]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
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

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
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

In [21]: `d.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 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(7), int64(1), object(1)
memory usage: 293.8+ KB
```

In [22]: `d.describe()`

Out[22]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

In [23]:
```python
#mode

d.mode().T
```

Out[23]:

	0	1
Sex	M	NaN
Length	0.55	0.625
Diameter	0.45	NaN
Height	0.15	NaN
Whole weight	0.2225	NaN
Shucked weight	0.175	NaN
Viscera weight	0.1715	NaN
Shell weight	0.275	NaN
Rings	9.0	NaN

In [24]: `d.shape`

Out[24]: `(4177, 9)`

```
In [25]: #kurtosis

         d.kurt()
```

```
Out[25]: Length            0.064621
         Diameter         -0.045476
         Height           76.025509
         Whole weight     -0.023644
         Shucked weight    0.595124
         Viscera weight    0.084012
         Shell weight      0.531926
         Rings             2.330687
         dtype: float64
```

```
In [26]: #skewness

         d.skew()
```

```
Out[26]: Length           -0.639873
         Diameter         -0.609198
         Height            3.128817
         Whole weight      0.530959
         Shucked weight    0.719098
         Viscera weight    0.591852
         Shell weight      0.620927
         Rings             1.114102
         dtype: float64
```

```
In [27]: #variance

         d.var()
```

```
Out[27]: Length            0.014422
         Diameter          0.009849
         Height            0.001750
         Whole weight      0.240481
         Shucked weight    0.049268
         Viscera weight    0.012015
         Shell weight      0.019377
         Rings            10.395266
         dtype: float64
```

```
In [28]: #finding unique values for columns

         d.nunique()
```

```
Out[28]: Sex                 3
         Length            134
         Diameter          111
         Height             51
         Whole weight     2429
         Shucked weight   1515
         Viscera weight    880
         Shell weight      926
         Rings              28
         dtype: int64
```

5. Check for Missing values and deal with them.

```
In [29]: #finding missing values

         d.isna()
```

Out[29]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
4172	False	False	False	False	False	False	False	False	False
4173	False	False	False	False	False	False	False	False	False
4174	False	False	False	False	False	False	False	False	False
4175	False	False	False	False	False	False	False	False	False
4176	False	False	False	False	False	False	False	False	False

4177 rows × 9 columns

```
In [30]: d.isna().any()
```

```
Out[30]: 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 [31]: d.isna().sum()
```

```
Out[31]: Sex              0
         Length           0
         Diameter         0
         Height           0
         Whole weight     0
         Shucked weight   0
         Viscera weight   0
         Shell weight     0
         Rings            0
         dtype: int64
```
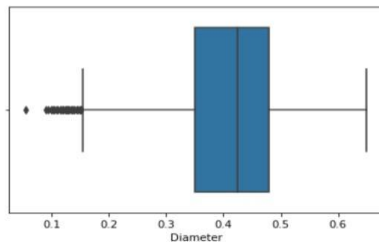
```
         dtype: bool
```

```
In [31]: d.isna().sum()
```

```
Out[31]: 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 [32]: d.isna().any().sum()
         #no missing values
```

```
Out[32]: 0
```

6. Find the outliers and replace them outliers

```
In [33]: #finding outliers
         sns.boxplot(d['Diameter'])
```

```
Out[33]: <AxesSubplot:xlabel='Diameter'>
```



```
In [34]: #handling outliers
         qnt=d.quantile(q=[0.25,0.75])
         qnt
```

Out[34]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0.25	0.450	0.35	0.115	0.4415	0.186	0.0935	0.130	8.0
0.75	0.615	0.48	0.165	1.1530	0.502	0.2530	0.329	11.0

```
In [35]: iqr=qnt.loc[0.75]-qnt.loc[0.25]
         iqr
```

```
Out[35]: Length           0.1650
         Diameter         0.1300
         Height           0.0500
         Whole weight     0.7115
         Shucked weight   0.3160
         Viscera weight   0.1595
         Shell weight     0.1990
         Rings            3.0000
         dtype: float64
```

```
In [36]: lower=qnt.loc[0.25]-(1.5*iqr)
         lower
```

```
Out[36]: Length               0.20250
         Diameter             0.15500
         Height               0.04000
         Whole weight        -0.62575
         Shucked weight      -0.28800
         Viscera weight      -0.14575
         Shell weight        -0.16850
         Rings                3.50000
         dtype: float64
```
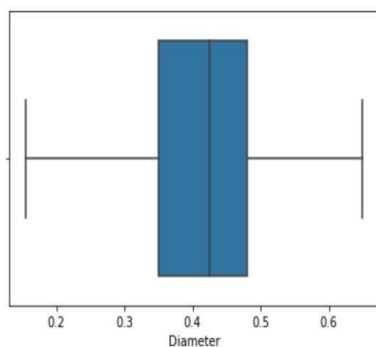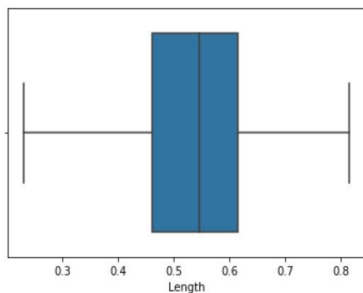
```
In [37]: upper=qnt.loc[0.75]+(1.5*iqr)
         upper
```

```
Out[37]: Length               0.86250
         Diameter             0.67500
         Height               0.24000
         Whole weight         2.22025
         Shucked weight       0.97600
         Viscera weight       0.49225
         Shell weight         0.62750
         Rings               15.50000
         dtype: float64
```

```
In [38]: # replacing outliers

         ##Diameter
         d['Diameter']=np.where(d['Diameter']<0.155,0.4078,d['Diameter'])
         sns.boxplot(d['Diameter'])
```

Out[38]: <AxesSubplot:xlabel='Diameter'>

Out[41]: <AxesSubplot:xlabel='Length'>
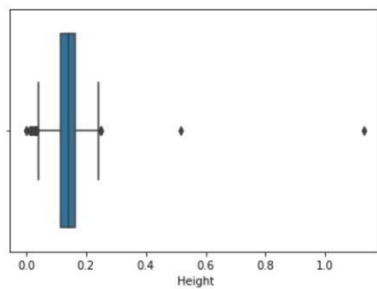


```
In [42]: ## Height

         sns.boxplot(d['Height'])
```

Out[42]: <AxesSubplot:xlabel='Height'>

In [42]: `## Height`

`sns.boxplot(d['Height'])`

Out[42]: `<AxesSubplot:xlabel='Height'>`
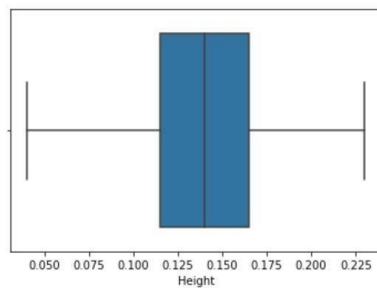

In [43]: `d['Height']=np.where(d['Height']<0.04,0.139, d['Height'])`
`d['Height']=np.where(d['Height']>0.23,0.139, d['Height'])`

`d['Height']=np.where(d['Height']>0.23,0.139, d['Height'])`

In [44]: `sns.boxplot(d['Height'])`

Out[44]: `<AxesSubplot:xlabel='Height'>`
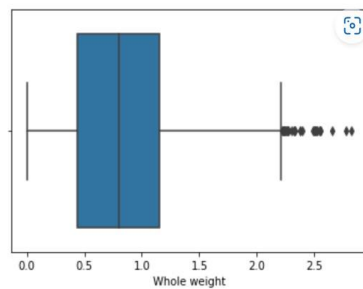

In [45]: `## Whole weight`

`sns.boxplot(d['Whole weight'])`

Out[45]: `<AxesSubplot:xlabel='Whole weight'>`

Out[45]: `<AxesSubplot:xlabel='Whole weight'>`

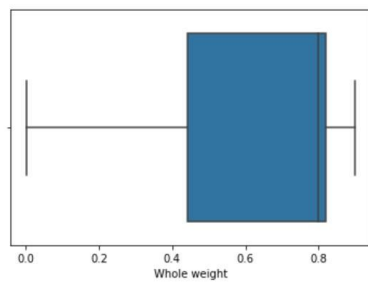In [46]: `d['Whole weight']=np.where(d['Whole weight']>0.9,0.82, d['Whole weight'])`

In [47]: `sns.boxplot(d['Whole weight'])`

Out[47]: `<AxesSubplot:xlabel='Whole weight'>`

```
In [47]: sns.boxplot(d['Whole weight'])
```
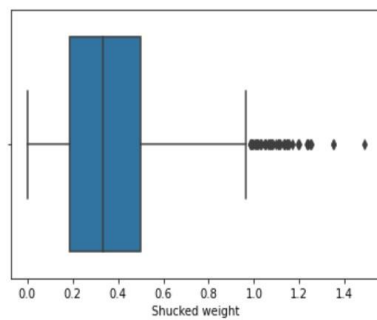
Out[47]: <AxesSubplot:xlabel='Whole weight'>


```
In [48]: ## Shucked weight
         sns.boxplot(d['Shucked weight'])
```

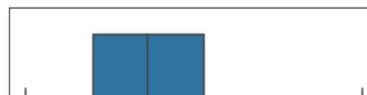Out[48]: <AxesSubplot:xlabel='Shucked weight'>

Out[48]: <AxesSubplot:xlabel='Shucked weight'>
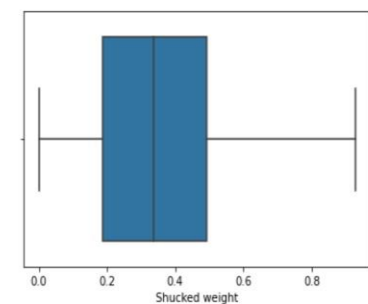


```
In [49]: d['Shucked weight']=np.where(d['Shucked weight']>0.93,0.35, d['Shucked weight'])
```

```
In [50]: sns.boxplot(d['Shucked weight'])
```

Out[50]: <AxesSubplot:xlabel='Shucked weight'>
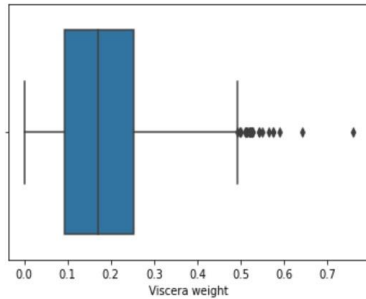

Out[50]: <AxesSubplot:xlabel='Shucked weight'>


```
In [51]: ## Viscera weight
         sns.boxplot(d['Viscera weight'])
```

Out[51]: <AxesSubplot:xlabel='Viscera weight'>

Out[51]: <AxesSubplot:xlabel='Viscera weight'>

In [52]: d['Viscera weight']=np.where(d['Viscera weight']>0.46,0.18, d['Viscera weight'])

In [53]: sns.boxplot(d['Viscera weight'])

Out[53]: <AxesSubplot:xlabel='Viscera weight'>
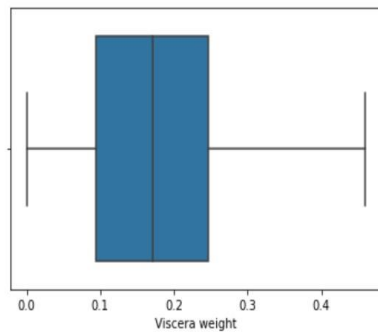
In [54]: ## Shell weight
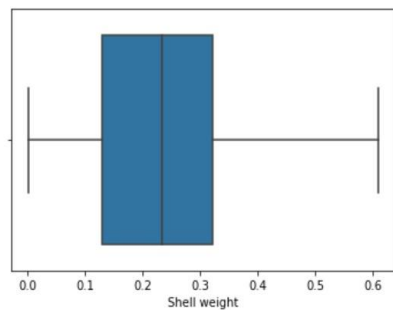   sns.boxplot(d['Shell weight'])

Out[54]: <AxesSubplot:xlabel='Shell weight'>
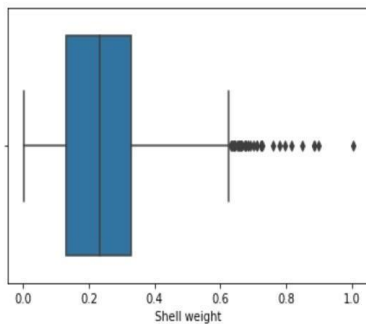
In [55]: d['Shell weight']=np.where(d['Shell weight']>0.61,0.2388, d['Shell weight'])

In [56]: sns.boxplot(d['Shell weight'])

Out[56]: <AxesSubplot:xlabel='Shell weight'>

```
In [56]: sns.boxplot(d['Shell weight'])
```

Out[56]: <AxesSubplot:xlabel='Shell weight'>

7. Check for Categorical columns and perform encoding.

```
In [57]: #one hot encoding

         d['Sex'].replace({'M':1,'F':0,'I':2},inplace=True)
         d
```

Out[57]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15
1	1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
...	...	...	...	...	...	...	...	...	...
4172	0	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	1	0.590	0.440	0.135	0.8200	0.4390	0.2145	0.2605	10
4174	1	0.600	0.475	0.205	0.8200	0.5255	0.2875	0.3080	9
4175	0	0.625	0.485	0.150	0.8200	0.5310	0.2610	0.2960	10
4176	1	0.710	0.555	0.195	0.8200	0.3500	0.3765	0.4950	12

4177 rows × 9 columns

8. Split the data into dependent and independent variables.

```
In [58]: x=d.drop(columns= ['Rings'])
         y=d['Rings']
         x
```

Out[58]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
0	1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500
1	1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100
3	1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550
...	...	...	...	...	...	...	...	...
4172	0	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490
4173	1	0.590	0.440	0.135	0.8200	0.4390	0.2145	0.2605
4174	1	0.600	0.475	0.205	0.8200	0.5255	0.2875	0.3080
4175	0	0.625	0.485	0.150	0.8200	0.5310	0.2610	0.2960
4176	1	0.710	0.555	0.195	0.8200	0.3500	0.3765	0.4950

4177 rows × 8 columns

4177 rows × 8 columns

```
In [59]: y
```

```
Out[59]: 0       15
         1        7
         2        9
         3       10
         4        7
                 ..
         4172    11
         4173    10
         4174     9
         4175    10
         4176    12
         Name: Rings, Length: 4177, dtype: int64
```

9. Scale the independent variables

```
In [60]: from sklearn.preprocessing import scale  #StandardScaler
```

```
In [61]: #Scaling the independent variables

         x = scale(x)
         x
```

```
Out[61]: array([[-0.0105225 , -0.67088921, -0.50179694, ..., -0.61037964,
                  -0.7328165 , -0.64358742],
                [-0.0105225 , -1.61376082, -1.57304487, ..., -1.22513334,
                  -1.24343929, -1.25742181],
                [-1.26630752,  0.00259051,  0.08738942, ..., -0.45300269,
                  -0.33890749, -0.18321163],
                ...,
                [-0.0105225 ,  0.63117159,  0.67657577, ...,  0.86994729,
                   1.08111018,  0.56873549],
                [-1.26630752,  0.85566483,  0.78370057, ...,  0.89699645,
                   0.82336724,  0.47666033],
                [-0.0105225 ,  1.61894185,  1.53357412, ...,  0.00683308,
                   1.94673739,  2.00357336]])
```

10. Split the data into training and testing

```
In [62]: from sklearn.model_selection import train_test_split
```

```
In [63]: #spliting data to train and test

         x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2)
         print(x_train.shape, x_test.shape)
```

```
(3341, 8) (836, 8)
```

11. Build the Model

```
In [64]: #Multiple Regression

         from sklearn.linear_model import LinearRegression

         MLR=LinearRegression()
```

12. Train the Model

```
In [65]: MLR.fit(x_train,y_train)

Out[65]: LinearRegression()
```

13. Test the Model

```
In [66]: #predcition on the test data
         y_pred=MLR.predict(x_test)
         y_pred

Out[66]: array([11.46655124,  9.2166091 ,  6.59967857,  7.81824648, 12.18984569,
                 11.44220895, 11.20545145,  8.71621092, 10.98237601,  6.83381457,
                 10.46227495,  9.10809044, 12.39359143, 14.54491772, 13.54791716,
                 10.12045364, 11.48597397,  7.73511543, 12.86466796,  8.37939955,
                  6.51920876,  8.16682072,  8.05416099, 10.22713858, 10.57995698,
                 11.31009826,  7.52742935,  9.88582514, 11.25644638, 11.38973324,
                 10.95569239, 10.28552912, 10.40475249, 10.44887526, 11.03343746,
                 10.15213587, 10.04733695,  6.54448931, 11.86305246,  6.73817965,
                  4.07354447, 11.09033543,  7.69897797,  9.56311429, 11.63006462,
                 13.17063754,  6.34451832,  7.27896893, 15.31511539,  6.92860099,
                  3.63485054,  6.80184256, 11.451762  , 10.69664795,  8.59383781,
                  7.50446583, 10.33994154, 11.85072027, 13.544946  , 10.27236403,
```

13. Test the Model

```
13.17663754,  6.34451832,  7.27896895, 15.31511539,  6.92866699,
 3.63485054,  6.80184256, 11.451762  , 10.69664795,  8.59383781,
 7.50446583, 10.33994154, 11.85072027, 13.544946  , 10.27236403,
 9.18410191,  7.7208794 , 12.33421272,  6.527156  , 11.17483778,
 7.97617745,  9.31452692,  9.56473016,  9.51077399, 12.20917888,
12.10672271,  4.70427674,  6.38943267, 10.02410014, 11.97786002,
12.77246335,  6.50139525, 10.64829499,  7.7058727 ,  6.05475715,
11.28248424, 10.75341994, 17.22835762,  9.53819376,  8.96368426,
 6.61412036, 12.00162611,  5.85400348,  4.07058709, 10.08426584,
10.15760235, 11.50892785, 10.58412873, 10.32113545, 12.98841501,
```

In [67]: #prediction in the train data
         pred=MLR.predict(x_train)
         pred

Out[67]: array([10.64104453, 11.72955404, 9.71670847, ..., 9.33031288,
                11.94411399, 9.8609076])

In [68]: from sklearn.metrics import r2_score

         acc=r2_score(y_test,y_pred)

         acc

Out[68]: 0.4331576346139585

In [69]: #test this model

14. Measure the performance using Metrics.¶

In [70]: from sklearn import metrics
         from sklearn.metrics import mean_squared_error

In [71]: np.sqrt(mean_squared_error(y_test,y_pred))

Out[71]: 2.4905110779015462

LASSO

In [72]: from sklearn.linear_model import Lasso, Ridge

In [73]: #intialising model

In [74]: #fit the model
         lso.fit(x_train,y_train)

Out[74]: Lasso(alpha=0.01, normalize=True)

In [75]: #predcition on test data

         lso_pred=lso.predict(x_test)

In [76]: #coef
         coef=lso.coef_
         coef

Out[76]: array([-0.   , 0.   , 0.   , 0.4751529 , 0.18634695,
                 0.   , 0.   , 0.8021721])

In [77]: #accuracy

         from sklearn import metrics
         from sklearn.metrics import mean_squared_error
         metrics.r2_score(y_test,lso_pred)

Out[77]: 0.3260900261255968
```

```
In [78]:  #error

          np.sqrt(mean_squared_error(y_test,lso_pred))
```

Out[78]: 2.715552909824135

RIDGE

```
In [79]:  rg=Ridge(alpha=0.01,normalize=True)
```

```
In [80]:  #fit
          rg.fit(x_train,y_train)
```

Out[80]: Ridge(alpha=0.01, normalize=True)

```
In [81]:  #predcition

          rg_pred=rg.predict(x_test)
          rg_pred
```

```
Out[81]: array([11.49838542,  9.22452452,  6.72241086,  7.80010402, 12.09475499,
                11.33701357, 11.13313  ,  8.85299136, 10.95426872,  6.83332623,
                10.48221326,  9.08348674, 12.3098871 , 14.39846005, 13.62343834,
                10.11925891, 11.53997639,  7.75730522, 12.85320604,  8.43018605,
                 6.53855123,  8.20224034,  7.58755052, 10.2671289 , 10.65653767,
                11.30141111,  7.50735436,  9.91086293, 11.27856902, 11.29021902,
                10.93344581, 10.32246436, 10.4456454 , 10.47230589, 11.05682097,
                10.1640513 , 10.10050704,  6.5623351 , 11.84100809,  6.75171646,
                 4.18665064, 11.0291328 ,  7.72116038,  9.60080953, 11.57691909,
                13.01362452,  6.35434964,  7.30414243, 15.1541625 ,  6.91515291,
                 4.16356146,  6.81943931, 11.43766939, 10.62078881,  8.65255458,
                 7.53582353, 10.44494347, 11.86697333, 13.45239251, 10.40153892,
                 9.1961334 ,  7.75332002, 12.25958727,  6.54710958, 11.17149665,
                 7.96864693,  9.37526527,  9.69298327,  9.54666379, 12.19633696,
                12.1127204 ,  4.82993146,  6.43644112,  9.93303646, 12.00590353,
                12.76127566,  6.53636246, 10.58092597,  7.71945979,  5.98307484,
                11.30577761, 10.77297947, 16.83531384,  9.62951405,  9.07577717,
                 6.65047637, 11.98056215,  5.83715385,  4.18471904, 10.66479866,
                10.18956629, 11.5066688 , 10.63940289, 10.38796727, 12.94599046,
```

```
In [82]:  #coef
          rg.coef_
```

Out[82]: array([-0.30797338, -0.75443399, 0.34843757, 0.94370227, 0.96851431,
               -1.38791368, -0.04943813, 1.70772786])

```
In [82]:  #coef
          rg.coef_
```

Out[82]: array([-0.30797338, -0.75443399, 0.34843757, 0.94370227, 0.96851431,
               -1.38791368, -0.04943813, 1.70772786])

```
In [83]:  #accuracy

          metrics.r2_score(y_test,rg_pred)
```

Out[83]: 0.43177328549243543

```
In [84]:  #error

          np.sqrt(mean_squared_error(y_test,rg_pred))
```

Out[84]: 2.4935504011542577

```
In [ ]:  # 1. Download the dataset

         #importing the libraries

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

```python
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# 2. Load the dataset into the tool.

#Loading the dataset

d = pd.read_csv(r'Downloads/abalone.csv')

# 3. Perform Below Visualizations.

# · Univariate Analysis

d.head()

#Boxplot

sns.boxplot(d['Diameter'])

#histogram

plt.hist(d['Diameter'])

#line plot
```

```python
#line plot

plt.plot(d['Diameter'].head(10))

#piechart

plt.pie(d['Diameter'].head(),autopct='%.2f')

#distplot

sns.distplot(d['Diameter'].head(200))

# ● Bi - Variate Analysis

#scatter plot

plt.scatter(d['Diameter'].head(500),d['Length'].head(500))

#bar plot

plt.bar(d['Sex'].head(10),d['Rings'].head(10))

#Labelling of x,y and result

plt.title('Bar plot')
```

```python
plt.xlabel('Diameter')
plt.ylabel('Rings')

sns.barplot(d['Sex'], d['Rings'])

#joint plot

sns.jointplot(d['Diameter'].head(50),d['Rings'].head(50))

#bar plot

sns.barplot('Diameter','Rings',hue='Sex',data=d.head())

sns.lineplot(d['Diameter'].head(),d['Rings'].head())

# ● Multi - Variate Analysis

#boxplot

sns.boxplot(d['Sex'].head(10),d['Diameter'].head(10),d['Rings'].head(10))

#heat map

fig=plt.figure(figsize=(8,5))
sns.heatmap(d.head().corr(),annot=True)

#pair plot

sns.pairplot(d.head(),hue='Rings')
```

```python
sns.pairplot(d.head(),hue='Rings')

sns.pairplot(d.head())

# 4. Perform descriptive statistics on the dataset.

#head

d.head()

#tail

d.tail()

d.info()

d.describe()

#mode

d.mode().T

d.shape

#kurtosis
```

```python
d.kurt()

#skewness

d.skew()

#variance

d.var()

#finding unique values for columns

d.nunique()

# 5. Check for Missing values and deal with them.

#finding missing values

d.isna()

d.isna().any()

d.isna().sum()

d.isna().any().sum()
#no missing values
```

```python
# 6. Find the outliers and replace them outliers

#finding outliers

sns.boxplot(d['Diameter'])

#handling outliers

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

iqr=qnt.loc[0.75]-qnt.loc[0.25]

iqr

lower=qnt.loc[0.25]-(1.5*iqr)
lower

upper=qnt.loc[0.75]+(1.5*iqr)
upper

# replacing outliers

##Diameter
d['Diameter']=np.where(d['Diameter']<0.155,0.4078,d['Diameter'])
sns.boxplot(d['Diameter'])
```

```python
## Length

sns.boxplot(d['Length'])

d['Length']=np.where(d['Length']<0.23,0.52, d['Length'])

sns.boxplot(d['Length'])

## Height

sns.boxplot(d['Height'])

d['Height']=np.where(d['Height']<0.04,0.139, d['Height'])
d['Height']=np.where(d['Height']>0.23,0.139, d['Height'])

sns.boxplot(d['Height'])

## Whole weight

sns.boxplot(d['Whole weight'])

d['Whole weight']=np.where(d['Whole weight']>0.9,0.82, d['Whole weight'])

sns.boxplot(d['Whole weight'])

## Shucked weight
```

```python
## Shucked weight

sns.boxplot(d['Shucked weight'])

d['Shucked weight']=np.where(d['Shucked weight']>0.93,0.35, d['Shucked weight'])

sns.boxplot(d['Shucked weight'])

## Viscera weight

sns.boxplot(d['Viscera weight'])

d['Viscera weight']=np.where(d['Viscera weight']>0.46,0.18, d['Viscera weight'])

sns.boxplot(d['Viscera weight'])

## Shell weight

sns.boxplot(d['Shell weight'])

d['Shell weight']=np.where(d['Shell weight']>0.61,0.2388, d['Shell weight'])

sns.boxplot(d['Shell weight'])
# 7. Check for Categorical columns and perform encoding.

#one hot encoding
```

```python
# 8. Split the data into dependent and independent variables.

x=d.drop(columns= ['Rings'])
y=d['Rings']
x

y

# 9. Scale the independent variables

from sklearn.preprocessing import scale  #StandardScaler

#Scaling the independent variables

x = scale(x)
x

# 10. Split the data into training and testing

from sklearn.model_selection import train_test_split
```

```python
# 10. Split the data into training and testing

from sklearn.model_selection import train_test_split

#spliting data to train and test

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2)
print(x_train.shape, x_test.shape)

# 11. Build the Model

#Multiple Regression

from sklearn.linear_model import LinearRegression

MLR=LinearRegression()

# 12. Train the Model

MLR.fit(x_train,y_train)

# 13. Test the Model

#predcition on the test data
y_pred=MLR.predict(x_test)
y_pred
```

```python
#prediction in the train data
pred=MLR.predict(x_train)
pred

from sklearn.metrics import r2_score

acc=r2_score(y_test,y_pred)

acc

#test this model

MLR.predict([[1,0.455,0.365,0.095,0.5140,0.2245,0.1010,0.150]])

# 14. Measure the performance using Metrics.¶

from sklearn import metrics
from sklearn.metrics import mean_squared_error

np.sqrt(mean_squared_error(y_test,y_pred))

# LASSO

from sklearn.linear_model import Lasso, Ridge

#intialising model

lso=Lasso(alpha=0.01,normalize=True)
```

```python
# LASSO

from sklearn.linear_model import Lasso, Ridge

#intialising model

lso=Lasso(alpha=0.01,normalize=True)

#fit the model
lso.fit(x_train,y_train)

#predcition on test data

lso_pred=lso.predict(x_test)

#coef
coef=lso.coef_
coef

#accuracy

from sklearn import metrics
from sklearn.metrics import mean_squared_error
metrics.r2_score(y_test,lso_pred)

#error
```