TEAM ID : PNT2022TMID32010

PROJECT TITLE : Estimate The Crop Yield Using Data Analytics

ASSIGNMENT DATE : 31.10.22

STUDENT NAME : MANOJ J

STUDENT ROLL NUMBER : 731619104029

1.Download the dataset

In [265]:

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

2.LOAD THE DATASET

In [266]:

```
df = pd.read_csv('abalone.csv')
```

In [267]:

```
df.head
```

Out[267]:

```
<bound method NDFrame.head of       Sex  Length  Diameter  Height  Whole weight  Shucked w
eight  \
0       M   0.455     0.365   0.095        0.5140          0.2245
1       M   0.350     0.265   0.090        0.2255          0.0995
2       F   0.530     0.420   0.135        0.6770          0.2565
3       M   0.440     0.365   0.125        0.5160          0.2155
4       I   0.330     0.255   0.080        0.2050          0.0895
...   ..     ...       ...     ...           ...             ...
4172    F   0.565     0.450   0.165        0.8870          0.3700
4173    M   0.590     0.440   0.135        0.9660          0.4390
4174    M   0.600     0.475   0.205        1.1760          0.5255
4175    F   0.625     0.485   0.150        1.0945          0.5310
4176    M   0.710     0.555   0.195        1.9485          0.9455

      Viscera weight  Shell weight  Rings
0             0.1010        0.1500     15
1             0.0485        0.0700      7
2             0.1415        0.2100      9
3             0.1140        0.1550     10
4             0.0395        0.0550      7
...              ...           ...    ...
4172          0.2390        0.2490     11
4173          0.2145        0.2605     10
4174          0.2875        0.3080      9
4175          0.2610        0.2960     10
4176          0.3765        0.4950     12

[4177 rows x 9 columns]>
```

In [268]:

```
Age=1.5+df.Rings
```

```
df["Age"]=Age
df=df.rename(columns = {'whole weight':'whole_weight','Shucked weight':'Shucked_weight',
'Viscera weight':'Viscera_weight','Shell weight':'Shell_weight'})
df=df.drop(columns=["Rings"],axis=1)
df.head()
```

Out[268]:

	Sex	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Age
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	16.5
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	8.5
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	10.5
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	11.5
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	8.5

In [269]:

```
df.tail()
```

Out[269]:

	Sex	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Age
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	12.5
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	11.5
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	10.5
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	11.5
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	13.5

3. Perform Below Visualizations

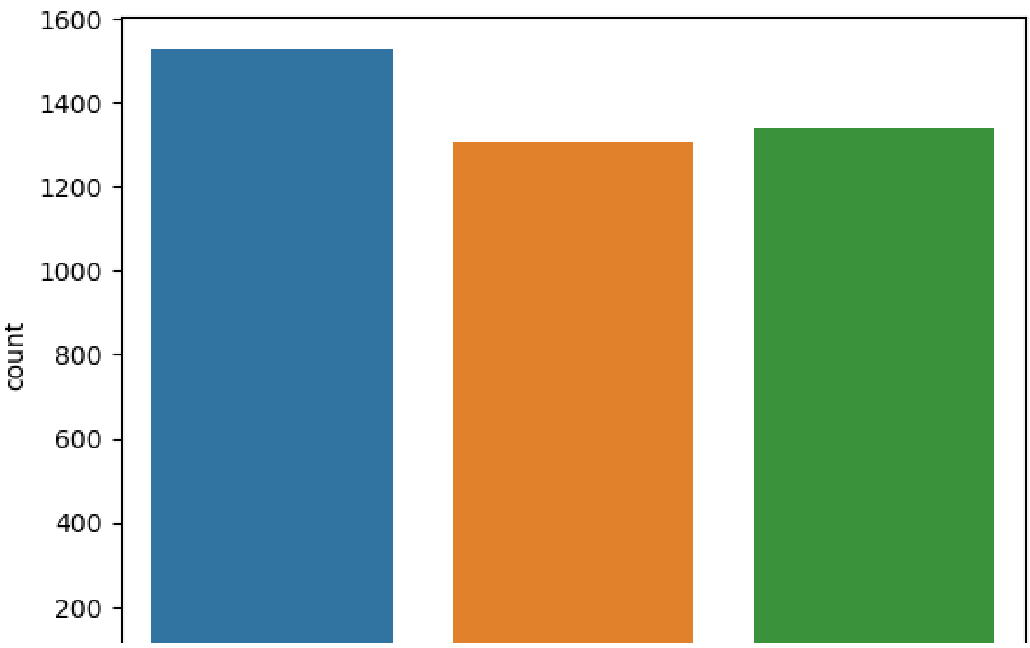
Univariate Analysis

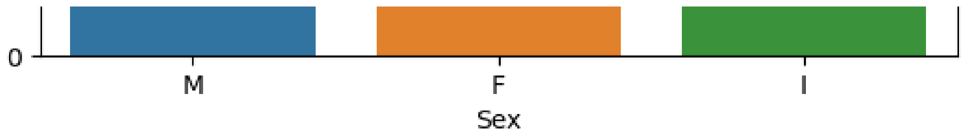
In [270]:

```
sns.countplot(x='Sex',data=df)
```

Out[270]:

<AxesSubplot:xlabel='Sex', ylabel='count'>



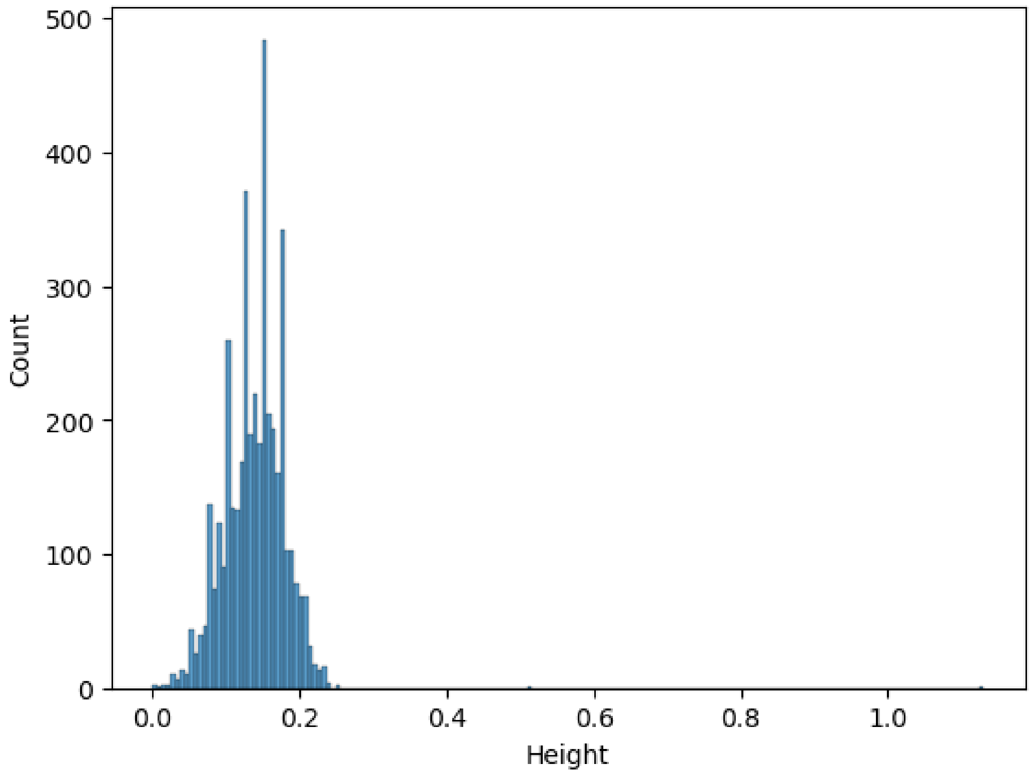


In [271]:

```
sns.histplot(df["Height"])
```

Out[271]:

```
<AxesSubplot:xlabel='Height',  ylabel='Count'>
```

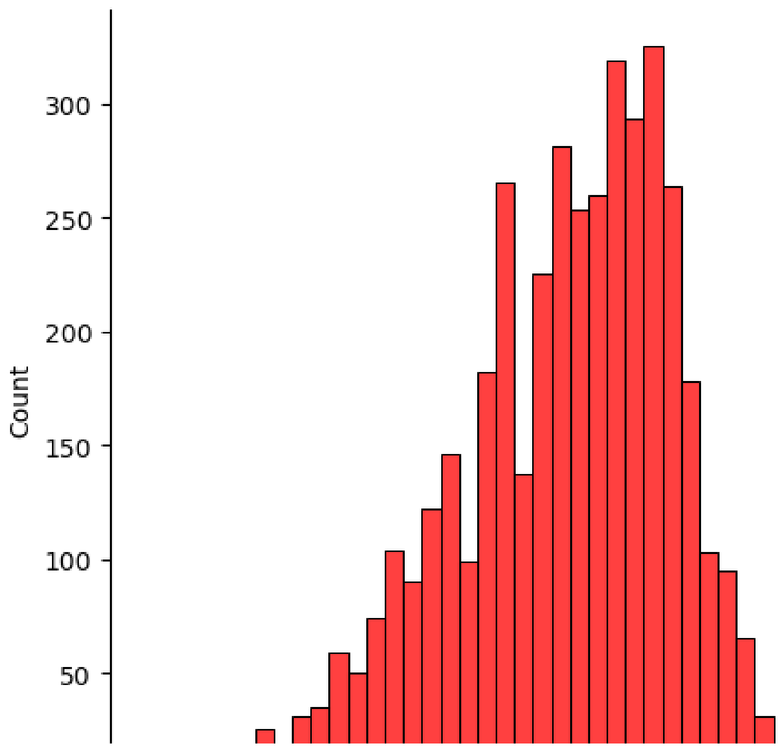


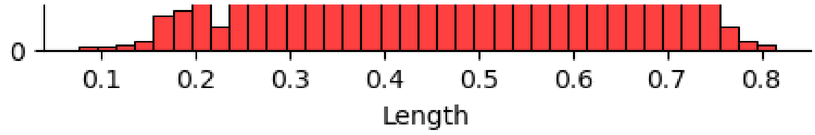
In [272]:

```
sns.displot(df["Length"],color='red')
```

Out[272]:

```
<seaborn.axisgrid.FacetGrid at 0x1af5e2f7820>
```

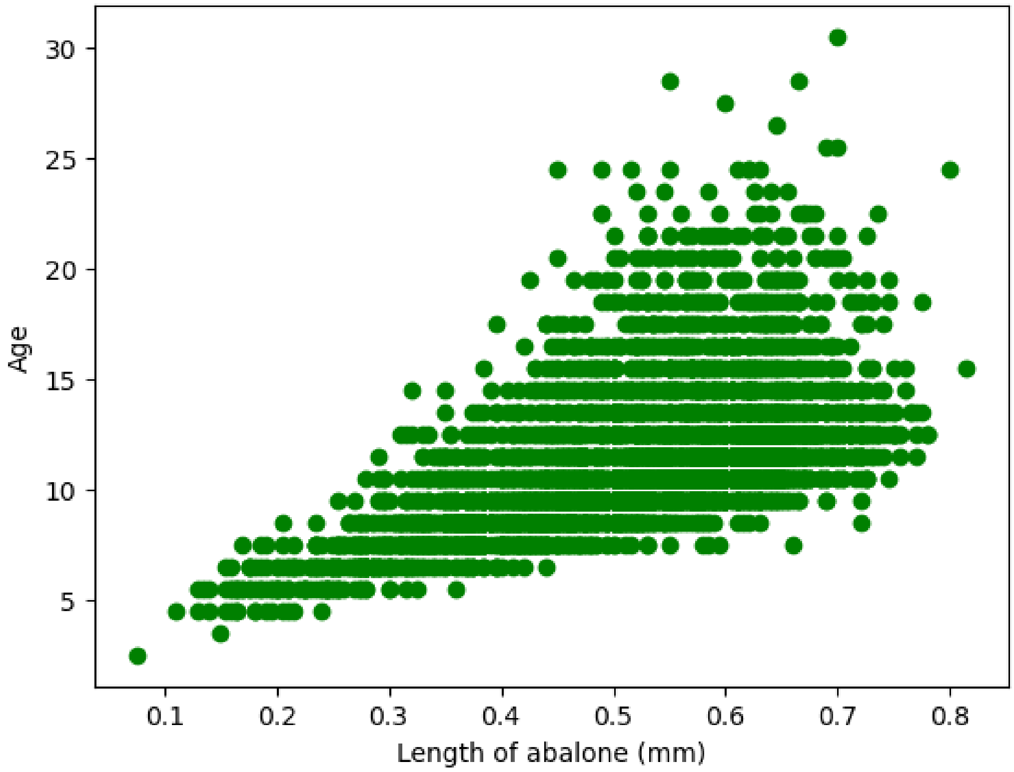




Bi-Variate Analysis

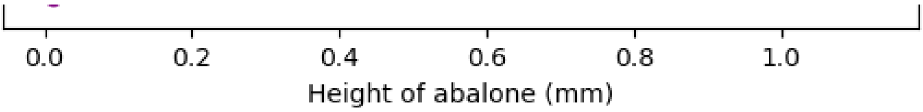
In [273]:

```
plt.scatter(df['Length'],df['Age'],c='green')
plt.xlabel('Length of abalone (mm)')
plt.ylabel('Age')
plt.show()
```



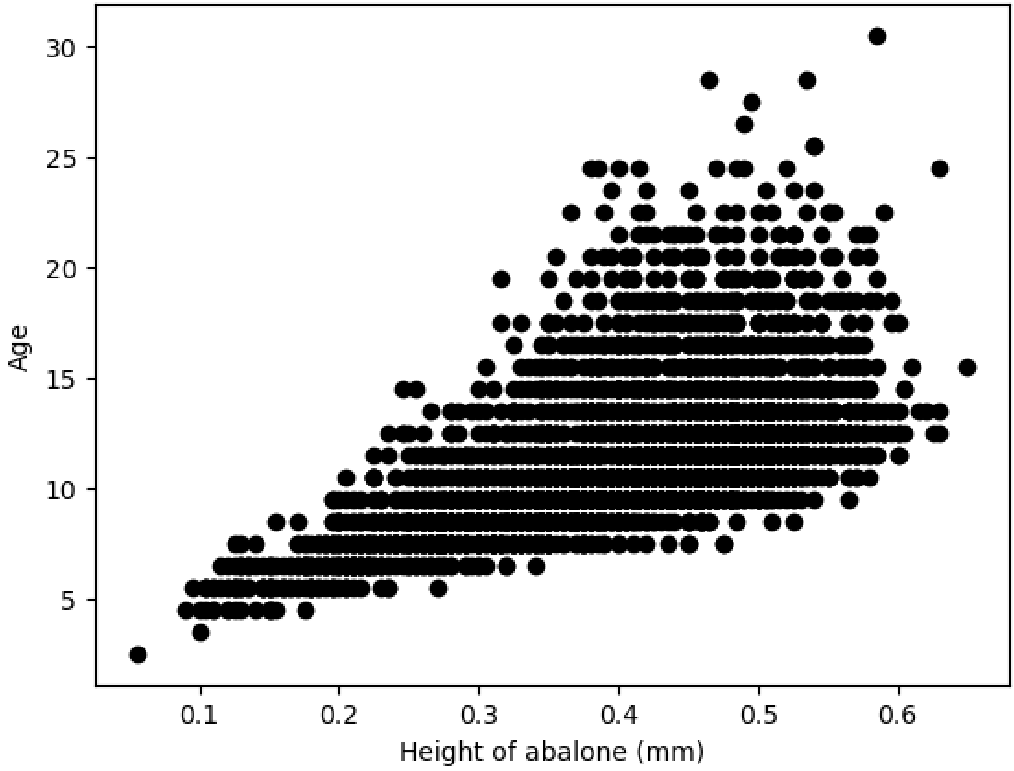
In [274]:

```
plt.scatter(df['Height'],df['Age'],c='purple')
plt.xlabel('Height of abalone (mm)')
plt.ylabel('Age')
plt.show()
```



In [275]:

```
plt.scatter(df['Diameter'],df['Age'],c='black')
plt.xlabel('Height of abalone (mm)')
plt.ylabel('Age')
plt.show()
```



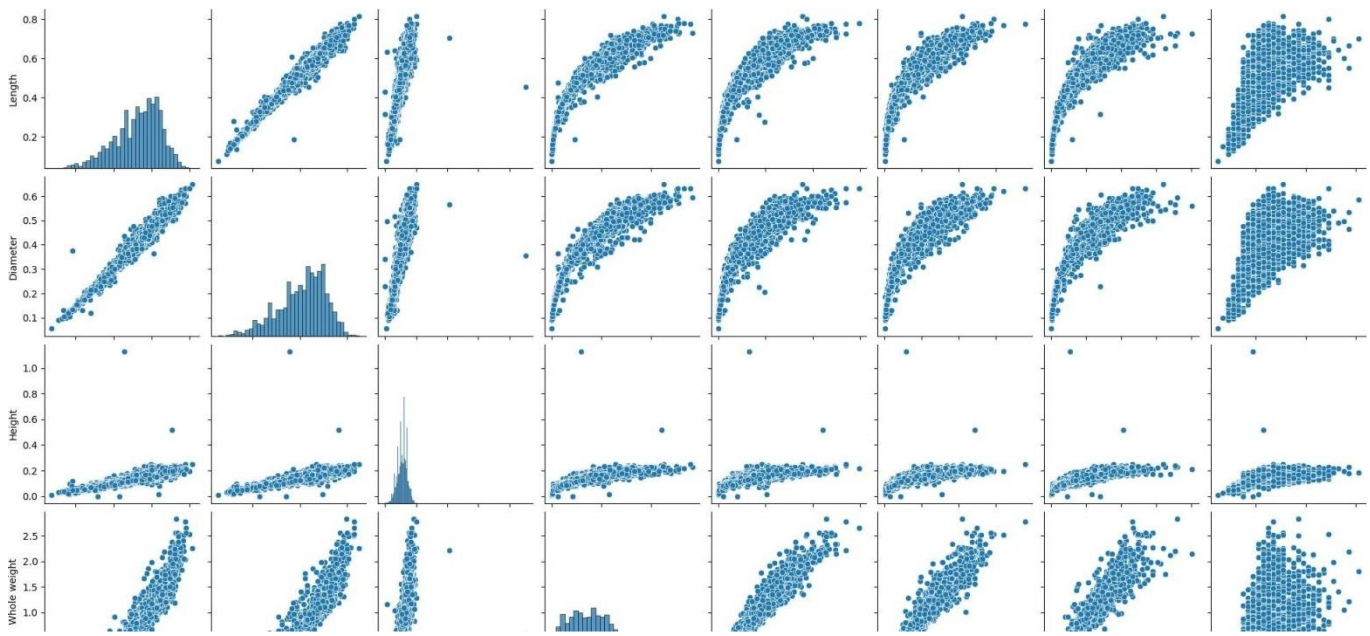
Multi-Variate Analysis

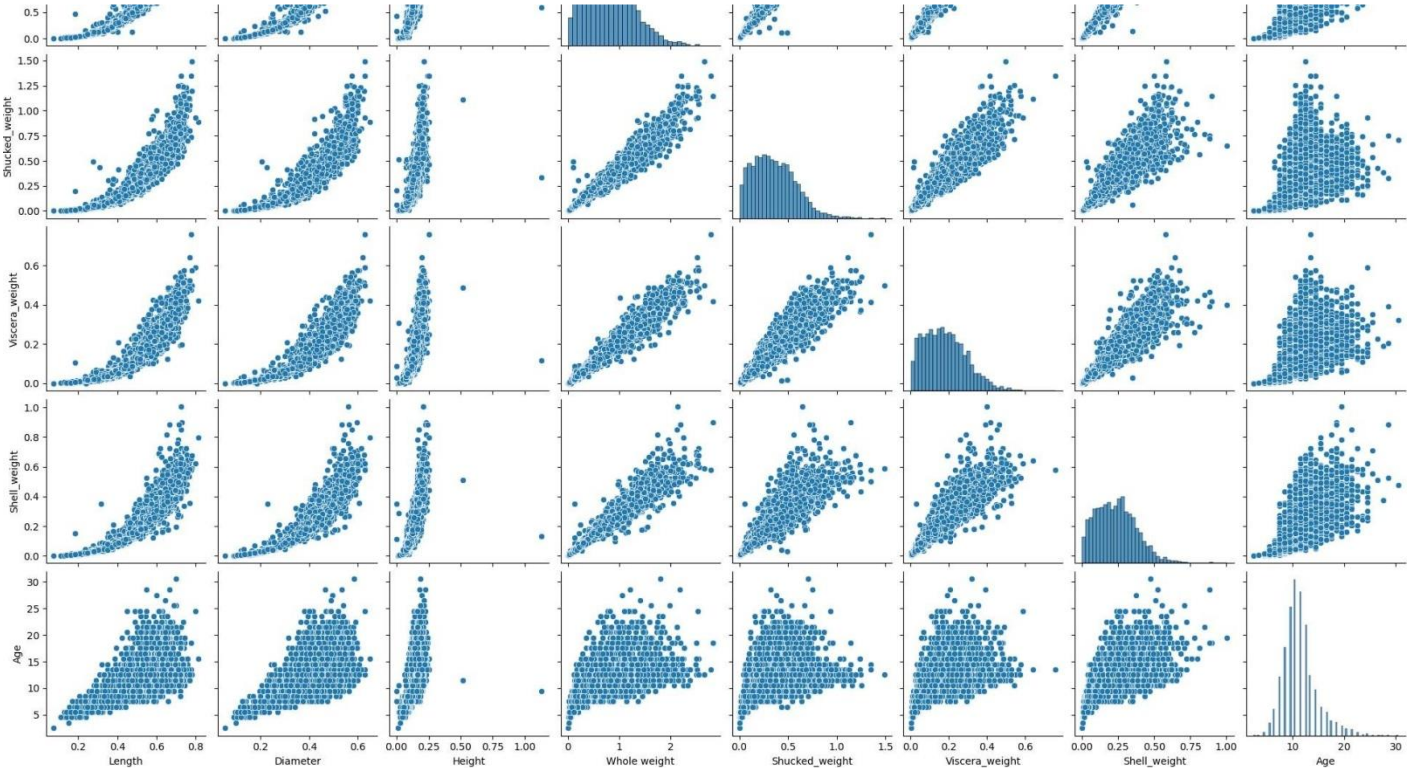
In [276]:

```
numerical_features = df.select_dtypes(include = [np.number]).columns
sns.pairplot(df[numerical_features])
```

Out[276]:

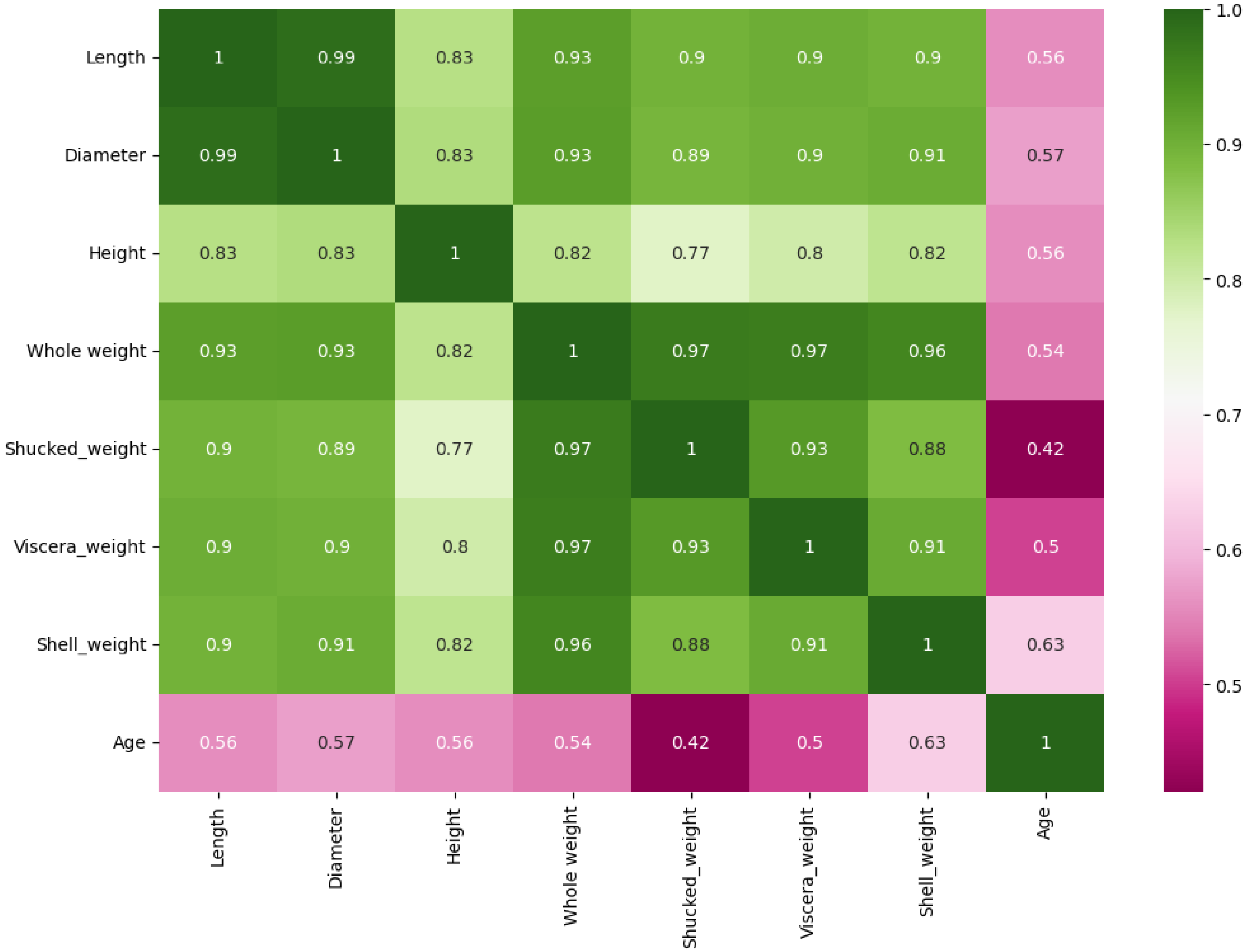
```
<seaborn.axisgrid.PairGrid at 0x1af61732d00>
```





In [277]:

```
plt.figure(figsize=(12,8));
sns.heatmap(df.corr(),cmap='PiYG',annot=True);
```



4. Perform descriptive statistics on the dataset

In [278]:

```
df.describe()
```

Out[278]:

	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Age
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	11.433684
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	2.500000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	9.500000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	10.500000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	12.500000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	30.500000

5.Check for Missing values and deal with them

In [279]:

```
df.isnull().sum()
```

Out[279]:

```
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked_weight    0
Viscera_weight    0
Shell_weight      0
Age               0
dtype: int64
```

6.Find the outliers and replace them outliers

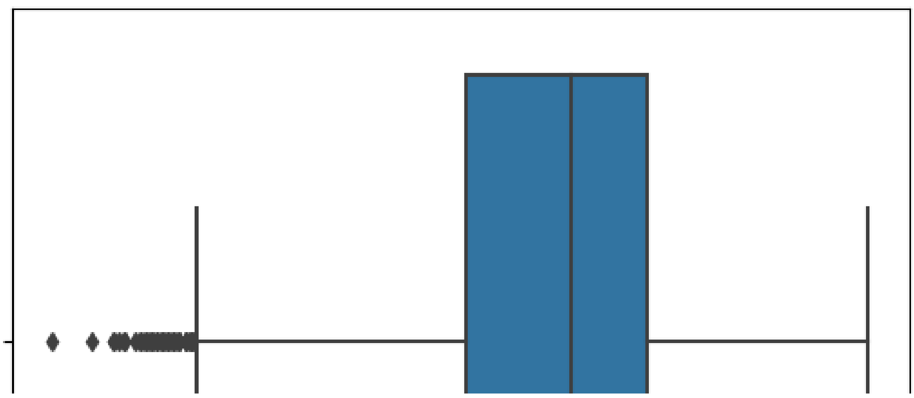
In [280]:

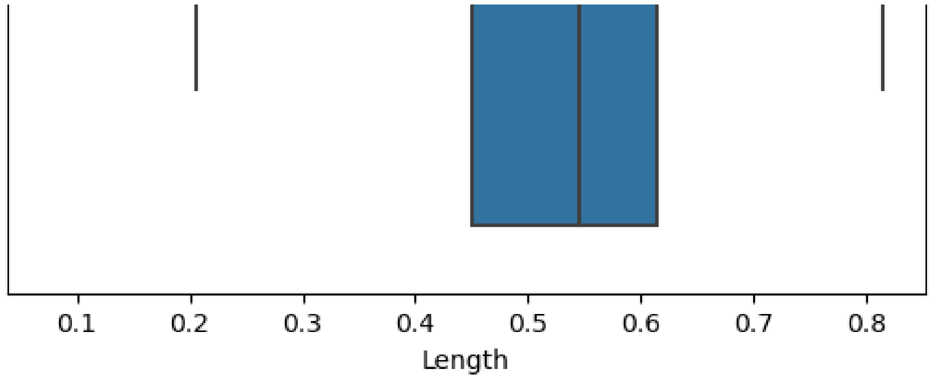
```
sns.boxplot(df['Length'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[280]:

<AxesSubplot:xlabel='Length'>





In [281]:

```
q1 = df['Length'].quantile(0.25)
q2 = df['Length'].quantile(0.75)
iqr = q2-q1
q1, q2,iqr
```

Out[281]:

```
(0.45, 0.615, 0.16499999999999998)
```

In [282]:

```
upper_limit = q2+(1.5*iqr)
lower_limit = q1-(1.5*iqr)
lower_limit, upper_limit
```

Out[282]:

```
(0.20250000000000004,  0.8624999999999999)
```

In [283]:

```
new_df = df.loc[(df['Length'] <= upper_limit) & (df['Length'] >= lower_limit)]
print('before removing outliers:', len(df))
print('after removing outliers:', len(new_df))
print('outliers:', len(df)-len(new_df))
```

```
before removing outliers: 4177
after removing outliers: 4128
outliers: 49
```

In [284]:

```
new_df = df.copy()
new_df.loc[(new_df['Length']>upper_limit), 'Length'] = upper_limit
new_df.loc[(new_df['Length']<lower_limit), 'Length'] = lower_limit
```

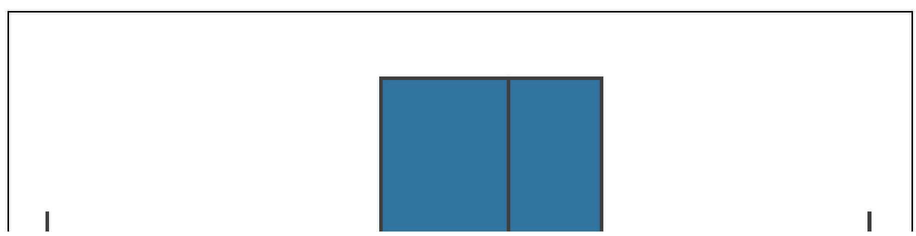
In [285]:

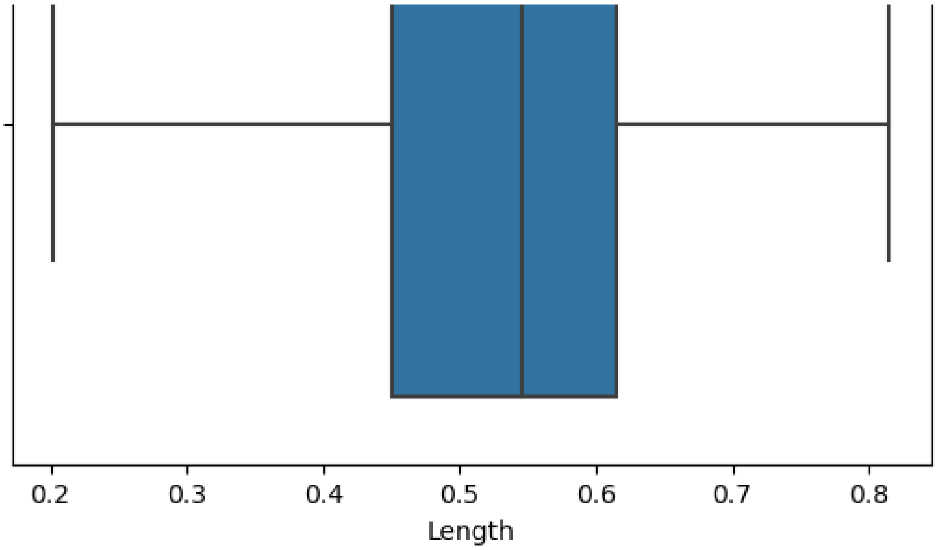
```
sns.boxplot(new_df['Length'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[285]:

```
<AxesSubplot:xlabel='Length'>
```





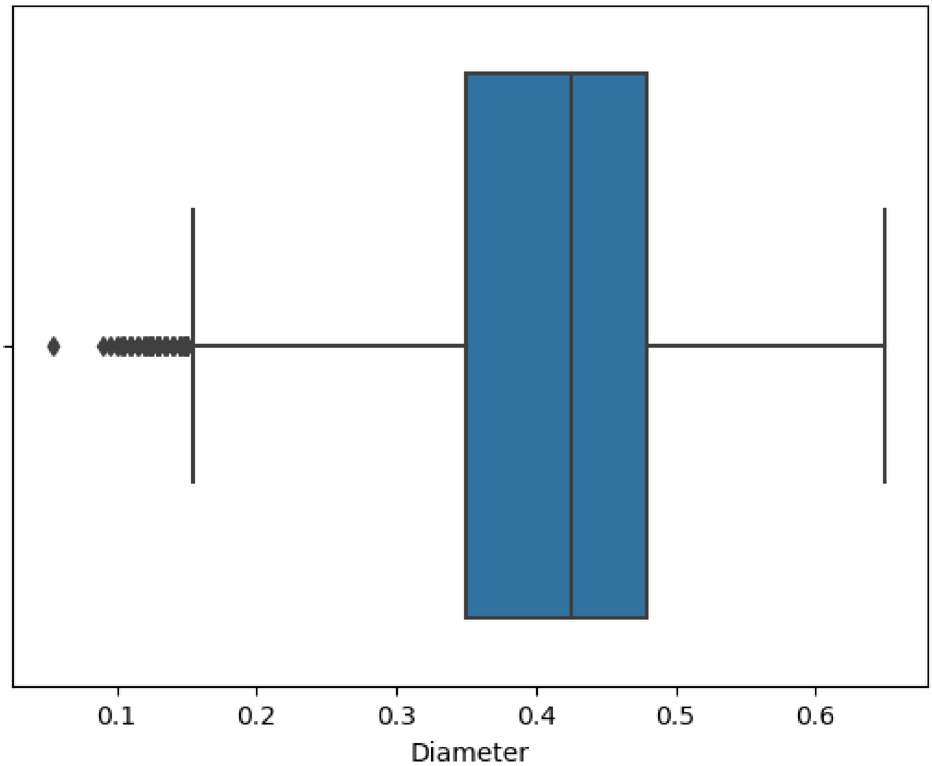
In [286]:

```
sns.boxplot(df['Diameter'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[286]:

```
<AxesSubplot:xlabel='Diameter'>
```



In [287]:

```
q1 = df['Diameter'].quantile(0.25)
q2 = df['Diameter'].quantile(0.75)
iqr = q2-q1
q1, q2, iqr
```

Out[287]:

```
(0.35, 0.48, 0.13)
```

In [288]:

```
upper_limit = q2 + (1.5*iqr)
lower_limit = q1 - (1.5*iqr)
lower_limit, upper_limit
```

Out[288]:

```
(0.15499999999999997,  0.675)
```

In [289]:

```
new_df = df.loc[(df['Diameter'] <= upper_limit) & (df['Diameter'] >= lower_limit)]
print('before removing outliers :', len(df))
print('after removing outliers :', len(new_df))
print('outliers :', len(df)-len(new_df))
```

```
before removing outliers : 4177
after removing outliers : 4118
outliers : 59
```

In [290]:

```
new_df = df.copy()
new_df.loc[(new_df['Diameter']>upper_limit), 'Diameter'] = upper_limit
new_df.loc[(new_df['Diameter']<lower_limit), 'Diameter'] = lower_limit
```

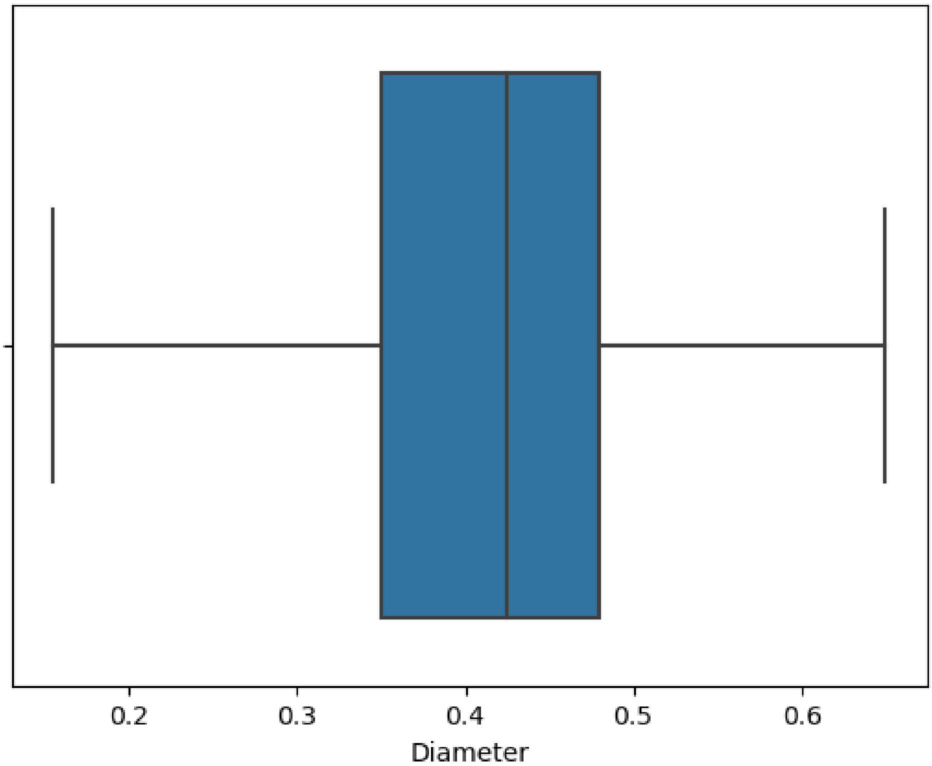
In [291]:

```
sns.boxplot(new_df['Diameter'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[291]:

```
<AxesSubplot:xlabel='Diameter'>
```



In [292]:

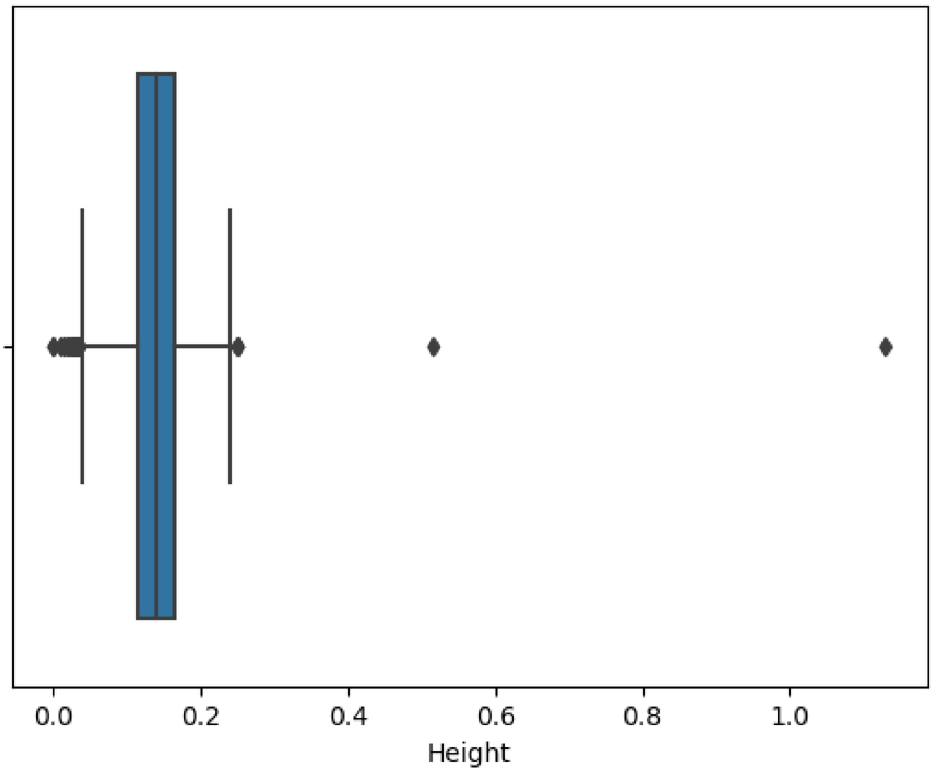
```
sns.boxplot(df['Height'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
```

```
ult in an error or misinterpretation.
  warnings.warn(
```

Out[292]:

```
<AxesSubplot:xlabel='Height'>
```



In [293]:

```
q1 = df['Height'].quantile(0.25)
q2 = df['Height'].quantile(0.75)
iqr = q2-q1
q1, q2, iqr
```

Out[293]:

```
(0.115, 0.165, 0.05)
```

In [294]:

```
upper_limit = q2 + (1.5*iqr)
lower_limit = q1 - (1.5*iqr)
lower_limit, upper_limit
```

Out[294]:

```
(0.03999999999999994,  0.24000000000000002)
```

In [295]:

```
new_df = df.loc[(df['Height'] <= upper_limit) & (df['Height'] >= lower_limit)]
print('before removing outliers :', len(df))
print('after removing outliers :', len(new_df))
print('outliers :', len(df)-len(new_df))
```

```
before removing outliers : 4177
after removing outliers : 4148
outliers : 29
```

In [296]:

```
new_df = df.copy()
new_df.loc[(new_df['Height']>upper_limit), 'Height'] = upper_limit
new_df.loc[(new_df['Height']<lower_limit), 'Height'] = lower_limit
```

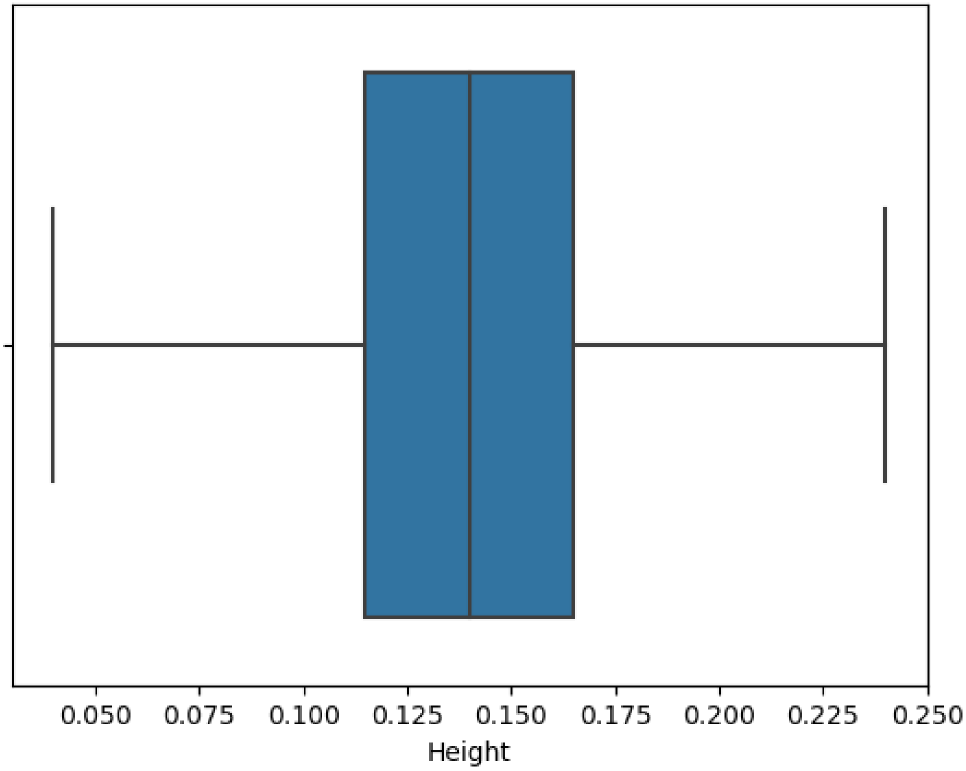
In [297]:

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

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36:  FutureWarning:  Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[297]:

<AxesSubplot:xlabel='Height'>



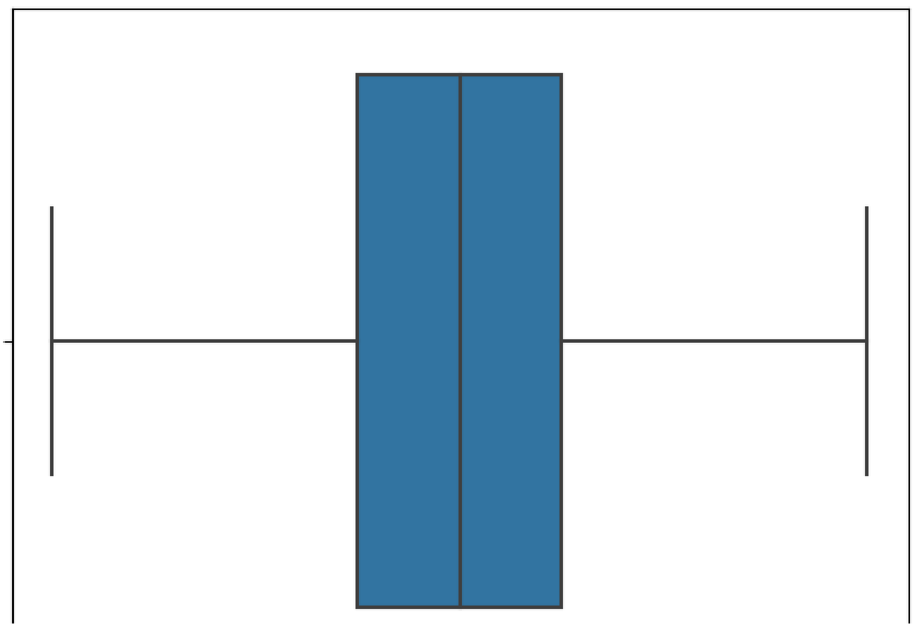
In [298]:

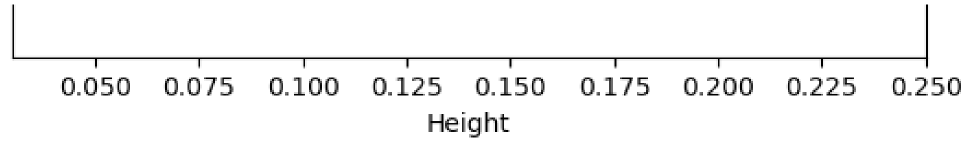
```
sns.boxplot(new_df['Height'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[298]:

<AxesSubplot:xlabel='Height'>





In [299]:

```
q1 = df['Whole weight'].quantile(0.25)
q2 = df['Whole weight'].quantile(0.75)
iqr = q2-q1
q1, q2, iqr
```

Out[299]:

```
(0.4415, 1.153, 0.7115)
```

In [300]:

```
upper_limit = q2 + (1.5*iqr)
lower_limit = q1 - (1.5*iqr)
lower_limit, upper_limit
```

Out[300]:

```
(-0.62575, 2.22025)
```

In [301]:

```
new_df = df.loc[(df['Whole weight'] <= upper_limit) & (df['Whole weight'] >= lower_limit
)]
print('before removing outliers :', len(df))
print('after removing outliers :', len(new_df))
print('outliers :', len(df)-len(new_df))
```

```
before removing outliers : 4177
after removing outliers : 4147
outliers : 30
```

In [302]:

```
new_df = df.copy()
new_df.loc[(new_df['Whole weight']>upper_limit), 'Whole weight'] = upper_limit
new_df.loc[(new_df['Whole weight']<lower_limit), 'Whole weight'] = lower_limit
```

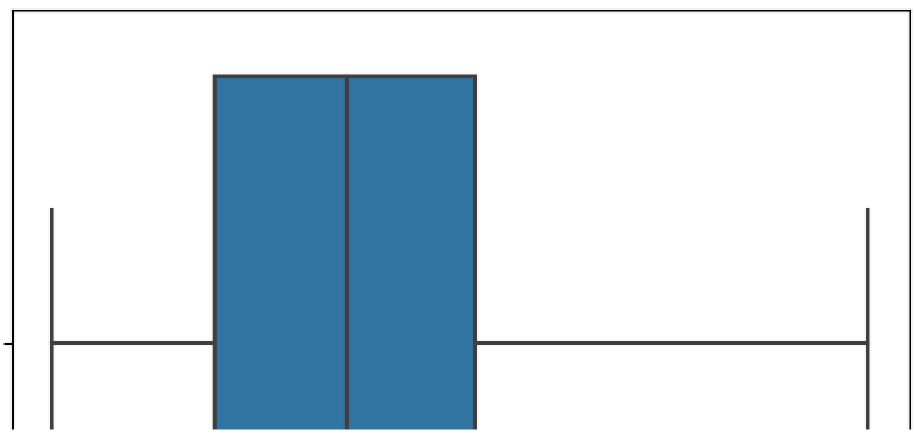
In [303]:

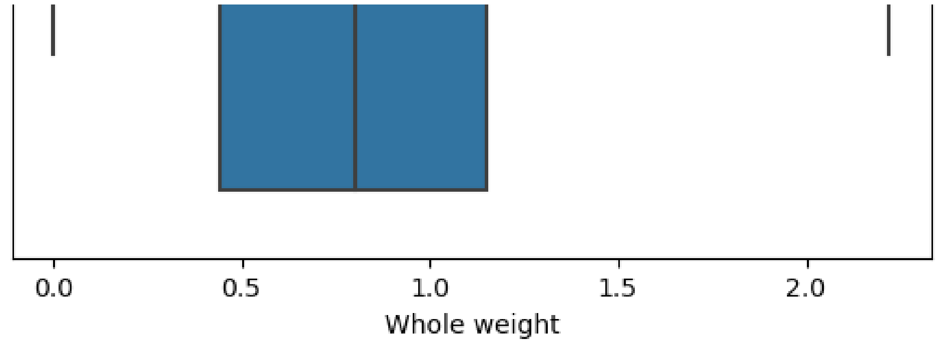
```
sns.boxplot(new_df['Whole weight'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[303]:

```
<AxesSubplot:xlabel='Whole  weight'>
```





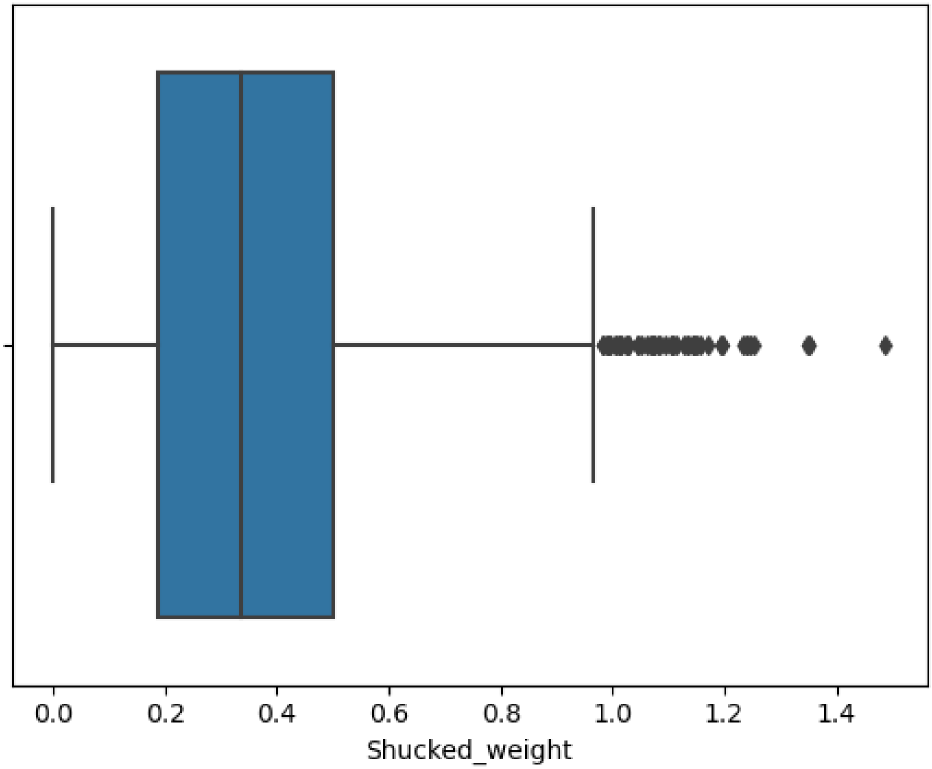
In [304]:

```
sns.boxplot(df['Shucked_weight'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[304]:

```
<AxesSubplot:xlabel='Shucked_weight'>
```



In [305]:

```
q1 = df['Shucked_weight'].quantile(0.25)
q2 = df['Shucked_weight'].quantile(0.75)
iqr = q2-q1
q1, q2, iqr
```

Out[305]:

```
(0.186, 0.502, 0.316)
```

In [306]:

```
upper_limit = q2 + (1.5*iqr)
lower_limit = q1 - (1.5*iqr)
lower_limit, upper_limit
```

Out[306]:

```
(-0.288, 0.976)
```

In [307]:

```
new_df = df.loc[(df['Shucked_weight'] <= upper_limit) & (df['Shucked_weight'] >= lower_l
imit)]
print('before removing outliers :', len(df))
print('after removing outliers :', len(new_df))
print('outliers :', len(df)-len(new_df))
```

```
before removing outliers : 4177
after removing outliers : 4129
outliers : 48
```

In [308]:

```
new_df = df.copy()
new_df.loc[(new_df['Shucked_weight']>upper_limit), 'Shucked_weight'] = upper_limit
new_df.loc[(new_df['Shucked_weight']<lower_limit), 'Shucked_weight'] = lower_limit
```

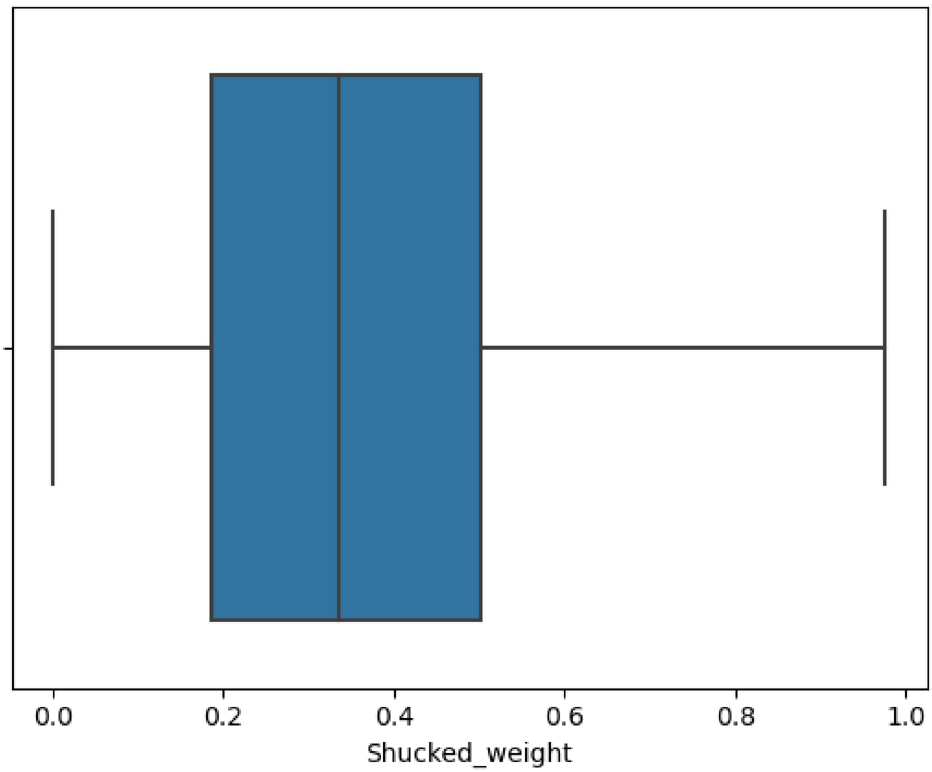
In [309]:

```
sns.boxplot(new_df['Shucked_weight'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[309]:

```
<AxesSubplot:xlabel='Shucked_weight'>
```



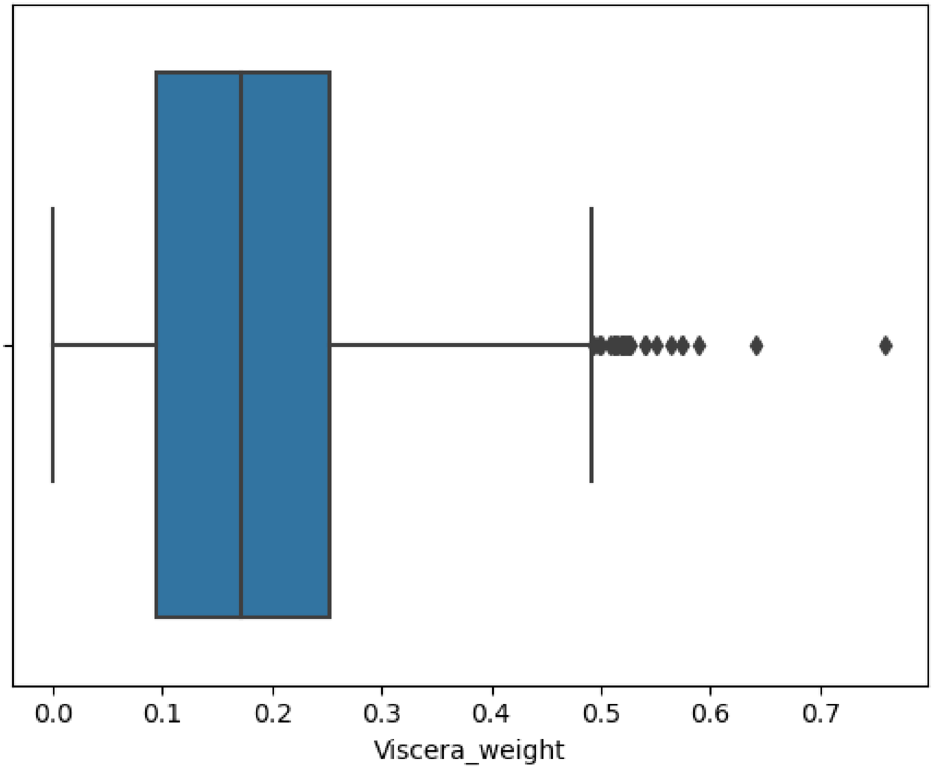
In [310]:

```
sns.boxplot(df['Viscera_weight'])
```

```
C:\Users\ELCOT\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass
the following variable as a keyword arg: x. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[310]:

```
<AxesSubplot:xlabel='Viscera_weight'>
```



7. Check for Categorical columns and perform encoding

In [311]:

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

Out[311]:

	Sex	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Age
0	1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	16.5
1	1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	8.5
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	10.5
3	1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	11.5
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	8.5
...	...	...	...	...	...	...	...	...	...
4172	0	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	12.5
4173	1	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	11.5
4174	1	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	10.5
4175	0	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	11.5
4176	1	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	13.5

4177 rows × 9 columns

In [312]:

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler
```

In [313]:

```
label_encoder = LabelEncoder()
df['Sex']= label_encoder.fit_transform(df['Sex'])
df
```

Out[313]:

	Sex	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Age
0	1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	16.5
1	1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	8.5
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	10.5
3	1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	11.5
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	8.5
...	...	...	...	...	...	...	...	...	...
4172	0	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	12.5
4173	1	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	11.5
4174	1	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	10.5
4175	0	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	11.5
4176	1	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	13.5

4177 rows × 9 columns

In [314]:

```
enc = OneHotEncoder(drop='first')

enc_df = pd.DataFrame(enc.fit_transform(df[['Sex']]).toarray())

df = df.join(enc_df)

df.head()
```

Out[314]:

	Sex	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Age	0	1
0	1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	16.5	1.0	0.0
1	1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	8.5	1.0	0.0
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	10.5	0.0	0.0
3	1	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	11.5	1.0	0.0
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	8.5	0.0	1.0

8.Split the data into dependent and independent variables

In [315]:

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
-----                ---------------  ------------
0    Length          4177 non-null   float64
1    Diameter        4177 non-null   float64
2    Height          4177 non-null   float64
3    Whole weight    4177 non-null   float64
4    Shucked_weight  4177 non-null   float64
5    Viscera_weight  4177 non-null   float64
6    Shell_weight    4177 non-null   float64
7    Age             4177 non-null   float64
8    Sex_F           4177 non-null   uint8
9    Sex_I           4177 non-null   uint8
 10  Sex_M           4177 non-null   uint8
dtypes: float64(8), uint8(3)
memory usage: 273.4 KB
```

In [316]:

```
X = x.drop(['Age'], axis = 1)
```

In [317]:

```
X.head(2)
```

Out[317]:

	Length	Diameter	Height	Whole weight	Shucked_weight	Viscera_weight	Shell_weight	Sex_F	Sex_I	Sex_M
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.15	0	0	1
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.07	0	0	1

In [318]:

```
y = x['Age']
```

In [319]:

```
y.head(2)
```

Out[319]:

```
0    16.5
1     8.5
Name: Age, dtype: float64
```

9.Scale the independent variables

In [320]:

```
scale = StandardScaler()
scaledX = scale.fit_transform(x)

print(scaledX)
```

```
[[-0.57455813 -0.43214879 -1.06442415 ... -0.67483383 -0.68801788
   1.31667716]
 [-1.44898585 -1.439929   -1.18397831 ... -0.67483383 -0.68801788
   1.31667716]
 [ 0.05003309  0.12213032 -0.10799087 ...  1.48184628 -0.68801788
  -0.75948762]
 ...
 [ 0.6329849   0.67640943  1.56576738 ... -0.67483383 -0.68801788
   1.31667716]
 [ 0.84118198  0.77718745  0.25067161 ...  1.48184628 -0.68801788
  -0.75948762]
 [ 1.54905203  1.48263359  1.32665906 ... -0.67483383 -0.68801788
   1.31667716]]
```

10. Split the data into training and testing

In [321]:

```
X.shape, y.shape
```

Out[321]:

```
((4177, 10), (4177,))
```

In [322]:

```
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)
```

In [323]:

```
print(' x_tain.shape : ',x_train.shape)
print(' y_tain.shape : ',y_train.shape)
print(' x_test.shape : ',x_test.shape)
print(' y_test.shape : ',y_test.shape)
```

```
 x_tain.shape :  (3341, 10)
 y_tain.shape :  (3341,)
 x_test.shape :  (836, 10)
 y_test.shape :  (836,)
```

10. Build the Model, 11. Train the Model , 12.Test the Model

In [324]:

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_pred = lr.predict(x_test)
```

In [325]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,make_scorer
from sklearn.model_selection import RandomizedSearchCV

rf = RandomForestRegressor()

param = {
    'max_depth':[3,6,9,12,15],
    'n_estimators':[10,50,100,150,200]
}
rf_search = RandomizedSearchCV(rf,param_distributions=param,n_iter=5,scoring=make_scorer
(mean_squared_error),n_jobs=-1,cv=5,verbose=3)
rf_search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[325]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=5, n_jobs=-1,
                   param_distributions={'max_depth': [3, 6, 9, 12, 15],
                                        'n_estimators': [10, 50, 100, 150,
                                                         200]},
                   scoring=make_scorer(mean_squared_error), verbose=3)
```

In [326]:

```
means = rf_search.cv_results_['mean_test_score']
params = rf_search.cv_results_['params']
for mean, param in zip(means, params):
    print("%f with: %r" % (mean,param))
    if mean == min(means):
        print('Best parameters with the minimum Mean Square Error are:' ,param)
```

```
4.664623 with: {'n_estimators': 200, 'max_depth': 6}
4.618707 with: {'n_estimators': 100, 'max_depth': 15}
4.644619 with: {'n_estimators': 200, 'max_depth': 15}
5.677870 with: {'n_estimators': 150, 'max_depth': 3}
4.581780 with: {'n_estimators': 100, 'max_depth': 9}
Best parameters with the minimum Mean Square Error are: {'n_estimators': 100, 'max_depth'
: 9}
```

In [327]:

```
rf = RandomForestRegressor(n_estimators=50,max_depth=6)
rf.fit(x_train,y_train)

rf_pred = rf.predict(x_test)
```

14. Measure the performance using Metrics

In [328]:

```
from sklearn import metrics
print('Linear Regression :')
print('--------------------')
print('MAE:',metrics.mean_absolute_error(y_test, lr_pred))
print('MSE:',metrics.mean_squared_error(y_test, lr_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, lr_pred)))
print('R2 Score:',metrics.r2_score(y_test,lr_pred))
print('\n\n')
```

```
Linear Regression :
--------------------
MAE: 1.5944508821770336
MSE: 4.892375672262822
RMSE: 2.211871531591024
R2 Score: 0.5480572061259404
```

In [329]:

```
from sklearn import metrics
print('Random Forest Contains:')
print('--------------------')
print('MAE:',metrics.mean_absolute_error(y_test, rf_pred))
print('MSE:',metrics.mean_squared_error(y_test, rf_pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test, rf_pred)))
print('R2 Score:',metrics.r2_score(y_test,rf_pred))
```

```
Random Forest Contains:
--------------------
MAE:  1.5580369509719958
MSE: 5.025592967383406
RMSE: 2.241783434541215
R2 Score: 0.535750997326301
```