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

```
file=pd.read_csv("abalone.csv")
df=pd.DataFrame(file)
df.head()
```

	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

```
df['age'] = df['Rings']+1.5
df = df.drop('Rings', axis = 1)
```

bold text

3. Perform Below Visualizations.

● Univariate Analysis ● Bi - Variate Analysis ● Multi - Variate Analysis

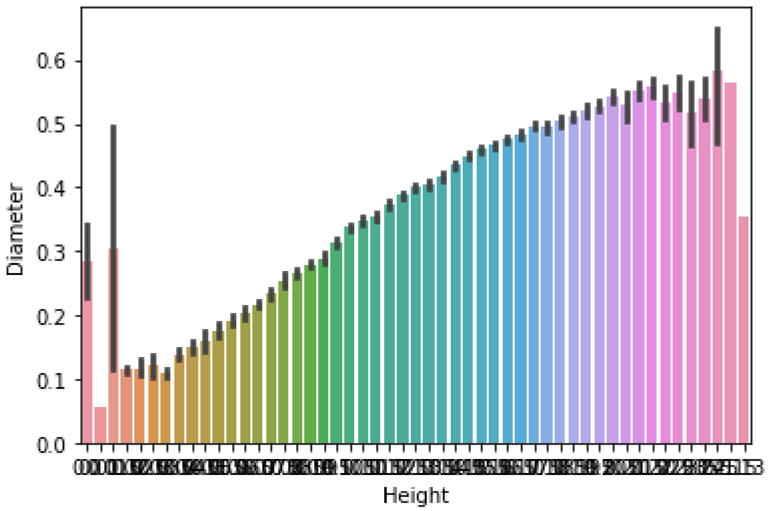
```
#Univariate Analysis
sbn.boxplot(df.Length)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: P
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0b6faf690>
```

the data is significantly imbalanced

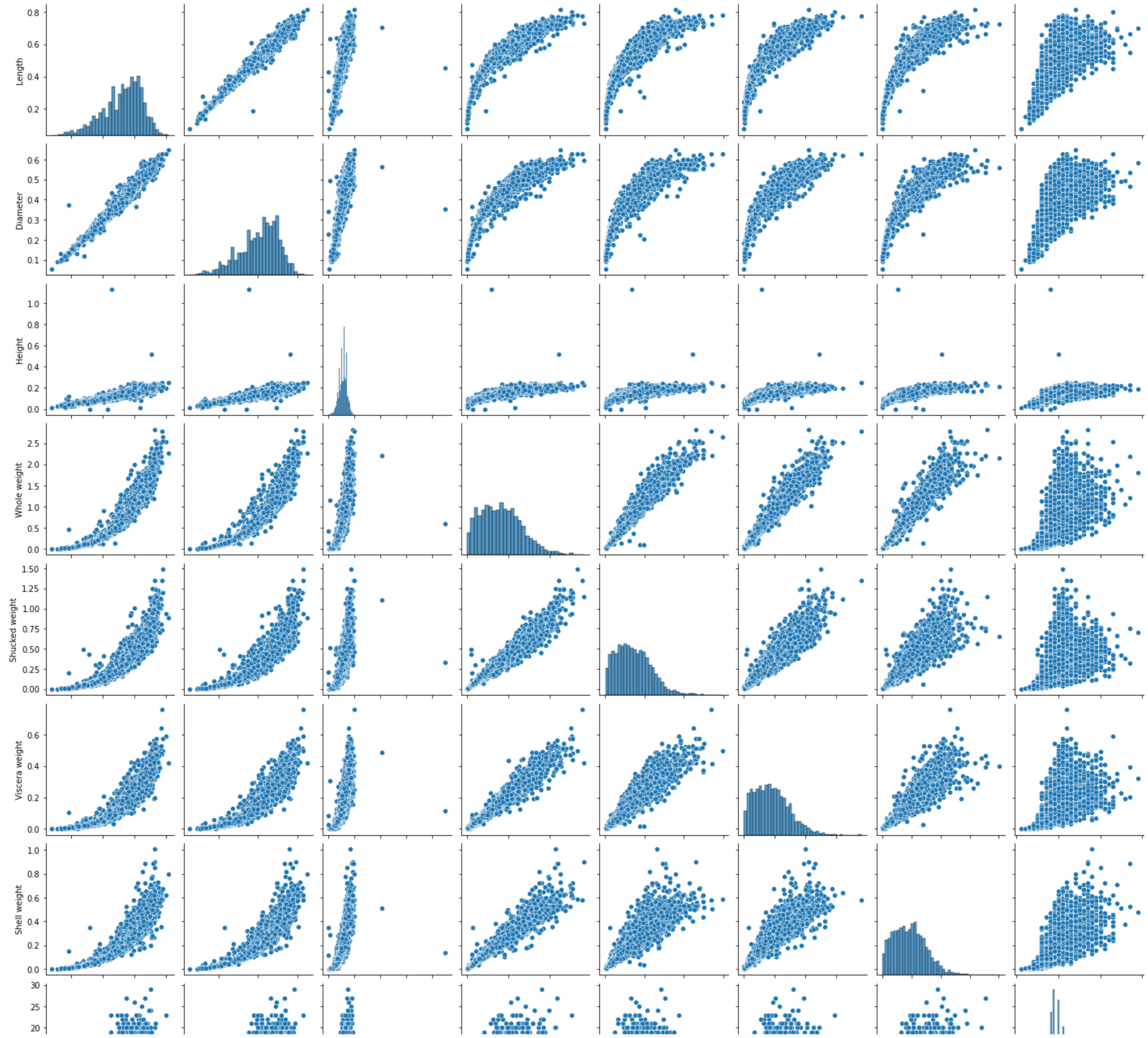
```
#Bi-Variant Analysis
sbn.barplot(x=df.Height,y=df.Diameter)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa0b6a65ed0>
```




```
#Multi-Variant Analysis
sbn.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fa0b68233d0>
```



4. Perform descriptive statistics on the dataset.

```
df.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)
```

```
df.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight
count	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
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000

5. Handle the Missing values.

```
df.isna().sum()
```

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

there is no missing values in dataset

```
for i in df:
    if df[i].dtype=='object' or df[i].dtype=='category':
        print("unique of "+i+" is "+str(len(set(df[i])))+" they are "+str(set(df[i])))
```

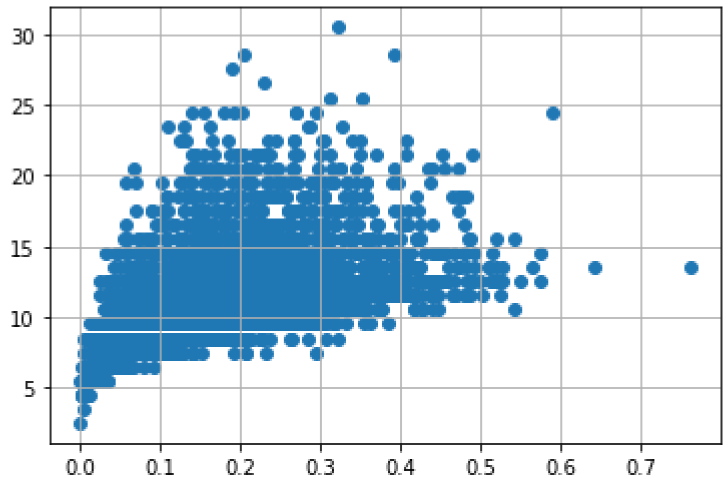
```
unique of Sex is 3 they are {'F', 'I', 'M'}
```

6. Find the outliers and replace the outliers

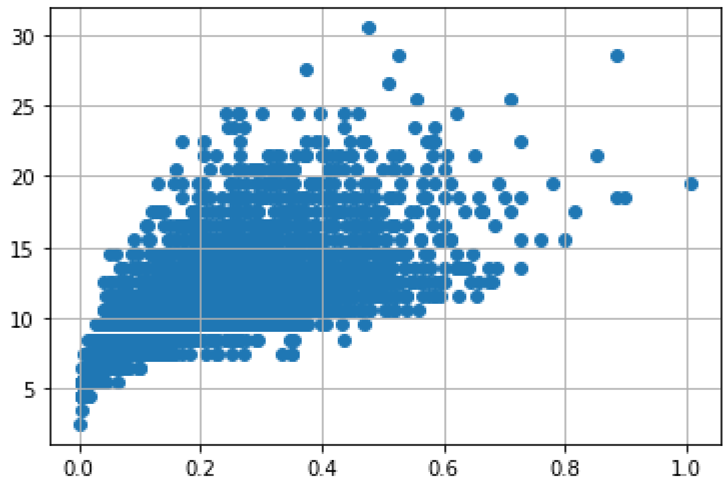
Checking for outliers

```
#Data Preprocessing
#Outlier handling
df = pd.get_dummies(df)
dummy_df = df
```

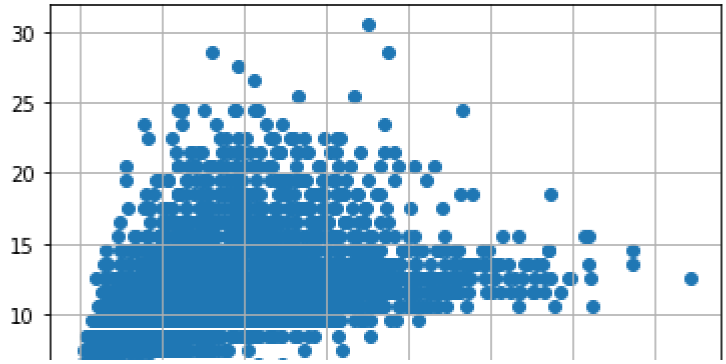
```
var = 'Viscera weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



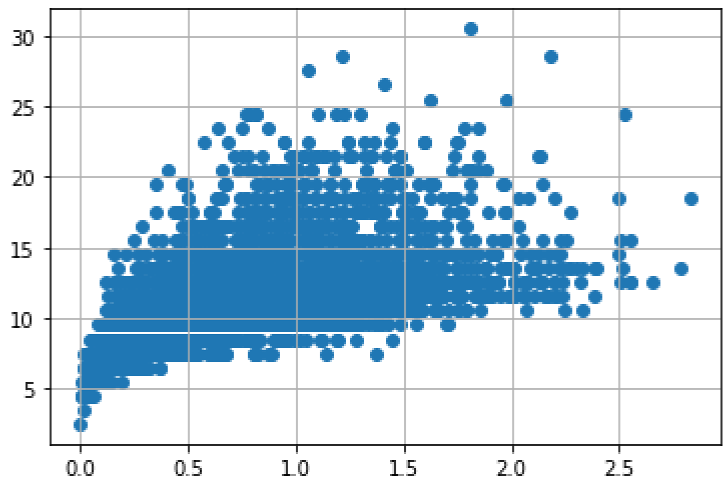

```
var = 'Shell weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



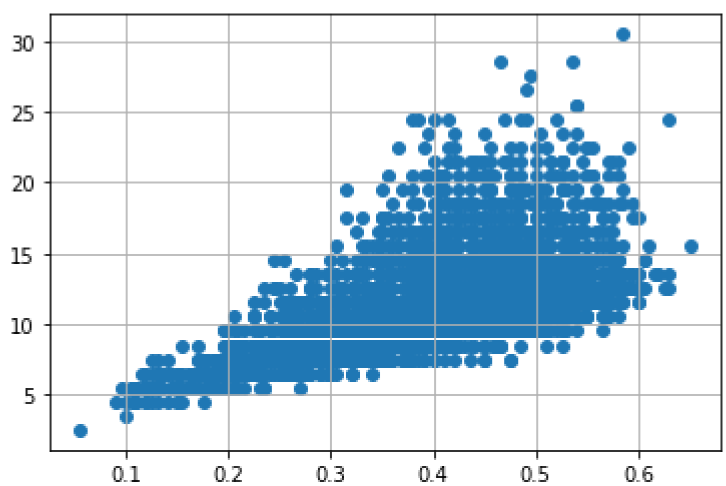

```
var = 'Shucked weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```




```
var = 'Whole weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```

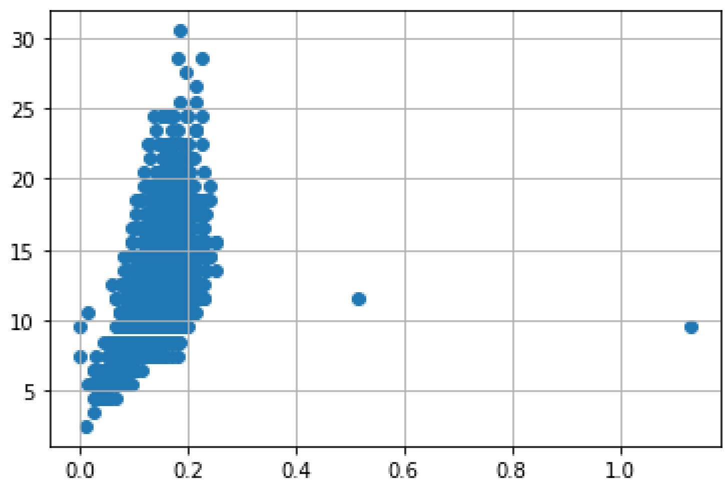



```
var = 'Diameter'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```

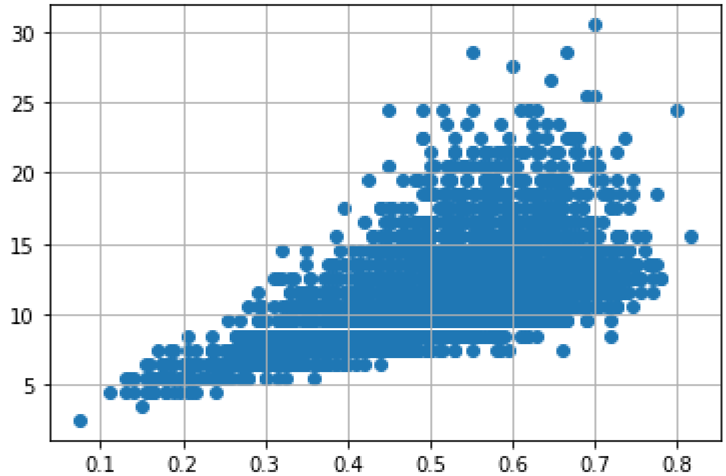



```
var = 'Height'
plt.scatter(x = df[var], y = df['age'])
```

```
plt.grid(True)
```



```
var = 'Length'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



Removing outliers

```
df.drop(df[(df['Viscera weight'] > 0.5) &
          (df['age'] < 20)].index, inplace = True)
df.drop(df[(df['Viscera weight']<0.5) & (
df['age'] > 25)].index, inplace = True)


df.drop(df[(df['Shell weight'] > 0.6) & (df['age'] < 25)].index, inplace = True)
df.drop(df[(df['Shell weight']<0.8) & (df['age'] > 25)].index, inplace = True)


df.drop(df[(df['Shucked weight'] >= 1) & (df['age'] < 20)].index, inplace = True)
df.drop(df[(df['Viscera weight']<1) & (df['age'] > 20)].index, inplace = True)
```

```
df.drop(df[(df['Diameter'] <0.1) & (df['age'] < 5)].index, inplace = True)
df.drop(df[(df['Diameter']<0.6) & (df['age'] > 25)].index, inplace = True)
df.drop(df[(df['Diameter']>=0.6) & (df['age'] < 25)].index, inplace = True)

df.drop(df[(df['Length'] <0.1) &(df['age'] < 5)].index, inplace = True)
df.drop(df[(df['Length']<0.8) & (df['age'] > 25)].index, inplace = True)
df.drop(df[(df['Length']>=0.8) & (df['age'] < 25)].index, inplace = True)
```

7. Check for Categorical columns and perform encoding.

```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
for i in df:
    if df[i].dtype=='object' or df[i].dtype=='category':
        df[i]=encoder.fit_transform(df[i])
```

8. Split the data into dependent and independent variables

```
x=df.iloc[:,:-1]
x.head()
```

	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	0.150	0	0	1
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	0	0	1
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	1	0	0
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	0	0	1

```
y=df.iloc[:,-1]
y.head()
```

```
0    16.5
1     8.5
2    10.5
3    11.5
4     8.5
Name: age, dtype: float64
```

9. Scale the independent variables

```
from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()
x=scaler.fit_transform(x)
```

```
x
```

```
array([[-0.53692652, -0.39073707, -1.04466898, ..., -0.66606428,
        -0.70790322,  1.33664164],
       [-1.42965157, -1.42053875, -1.16838344, ..., -0.66606428,
        -0.70790322,  1.33664164],
       [ 0.10073423,  0.17565386, -0.05495325, ...,  1.50135661,
        -0.70790322, -0.74814368],
       ...,
       [ 0.69588426,  0.74204479,  1.67704928, ..., -0.66606428,
        -0.70790322,  1.33664164],
       [ 0.90843784,  0.84502496,  0.31619015, ...,  1.50135661,
        -0.70790322, -0.74814368],
       [ 1.63112002,  1.56588614,  1.42962035, ..., -0.66606428,
        -0.70790322,  1.33664164]])
```

10. Split the 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=0.33)
x_train.shape
```

```
(2677, 10)
```

```
x_test.shape
```

```
(1319, 10)
```

```
y_train.shape
```

```
(2677,)
```

```
y_test.shape
```

```
(1319,)
```

MODEL

Linear regression

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train, y_train)
```

```
LinearRegression()
```

```
y_train_pred = lm.predict(x_train)
y_test_pred = lm.predict(x_test)
from sklearn.metrics import mean_absolute_error, mean_squared_error
s = mean_squared_error(y_train, y_train_pred)
print('Mean Squared Error of training set :%2f'%s)

p = mean_squared_error(y_test, y_test_pred)
print('Mean Squared Error of testing set :%2f'%p)
```

```
Mean Squared Error of training set :3.785284
Mean Squared Error of testing set :3.239985
```

```
from sklearn.metrics import r2_score
s = r2_score(y_train, y_train_pred)
print('R2 Score of training set:%.2f'%s)

p = r2_score(y_test, y_test_pred)
print('R2 Score of testing set:%.2f'%p)
```

```
R2 Score of training set:0.52
R2 Score of testing set:0.55
```