1. Download the dataset: Dataset

2. Load the dataset

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()

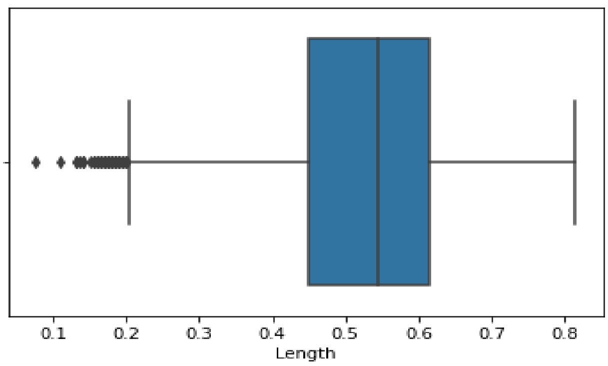
Sex Length Diameter Height Whole weight Shucked weight Viscera weight
Shell weight Rings
0 M 0.455 0.365 0.095 0.5140 0.2245 0.1010 0.150 15
1 M 0.350 0.265 0.090 0.2255 0.0995 0.0485 0.070 7 0
2 F .530 0.420 0.135 0.6770 0.2565 0.1415 0.210 9 0.
3 M 440 0.365 0.125 0.5160 0.2155 0.1140 0.155 10 0.
4 I 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)

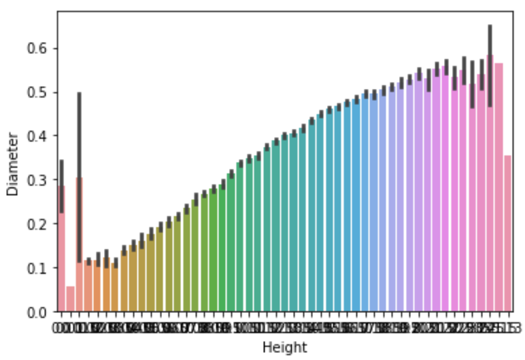
3. Perform Below Visualizations.

- **Univariate Analysis**
- **Bi - Variate Analysis**
- **Multi - Variate Analysis**

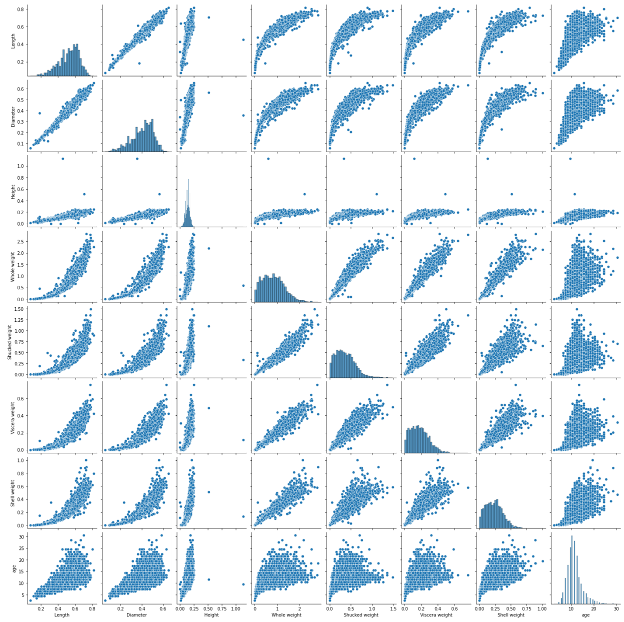
#Univariate Analysis
sbn.boxplot(df.Length)
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning



the data is significantly imbalanced
#Bi-Variant Analysis
sbn.barplot(x=df.Height,y=df.Diameter)



#Multi-Variant Analysis
sbn.pairplot(df)



4. Perform descriptive statistics on the dataset.

df.info()

```
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   age             4177 non-null   float64
dtypes: float64(8), object(1)
memory usage: 293.8+ KB
```

df.describe()

	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. 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
age             0
dtype: int64
```

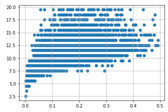
there is no missing values in dataset

```
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', 'M', 'I'}
```

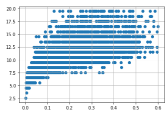
6. Find the outliers and replace the outliers

Checking for outliers

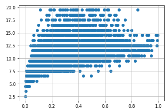
```
#Data Preprocessing
#Outlier handling
df = pd.get_dummies(df)
dummy_df = df
var = 'Viscera weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



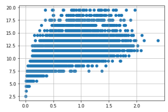
```
var = 'Shell weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



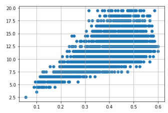
```
var = 'Shucked weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



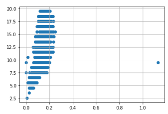
```
var = 'Whole weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



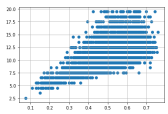
```
var = 'Diameter'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```



```
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['Height'] > 0.4) &(df['age'] < 15)].index, inplace = True)
df.drop(df[(df['Height']<0.4) & (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	age	Sex_F	Sex_I
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	16.5	0	0
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	8.5	0	0
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	10.5	1	0
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	11.5	0	0
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	8.5	0	1

```
y=df.iloc[:,-1]
y.head()
0    1
1    1
2    0
3    1
4    0
Name: Sex_M, dtype: uint8
```

9. Scale the independent variables

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x=scaler.fit_transform(x)
x
array([[-0.53701309, -0.39082366, -1.12698145, ...,  1.9433912 ,
        -0.66579302, -0.70803622],
       [-1.42965864, -1.4205279 , -1.26123393, ..., -0.95032771,
        -0.66579302, -0.70803622],
       [ 0.10059087,  0.17551367, -0.05296168, ..., -0.22689798,
         1.50196828, -0.70803622],
       ...,
       [ 0.6956879 ,  0.741851  ,  1.82657293, ..., -0.22689798,
        -0.66579302, -0.70803622],
       [ 0.90822255,  0.84482142,  0.34979574, ...,  0.13481688,
         1.50196828, -0.70803622],
       [ 1.63084038,  1.56561439,  1.55806799, ...,  0.85824661,
        -0.66579302, -0.70803622]])
```

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
(2676, 10)
```

x_test.shape
(1319, 10)
y_train.shape
(2676,)
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 :0.000000
Mean Squared Error of testing set :0.000000
```

Note: The Lower the Mean Squared Error,better the forecast.

```
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:1.00
R2 Score of testing set:1.00
```

Note: The ideal value of R-square is 1.

The closer the value of R-square to 1,better is the model fitted.