In [1]:

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

In [2]:

```
from google.colab import files
upload=files.upload()
df = pd.read_csv('abalone.csv')
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving abalone.csv to abalone.csv
```

In [3]:

```
df.describe()
```

Out[3]:

	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 [4]:

```
df.head()
```

Out[4]:

	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 [5]:

```
#Perform visualisations
#Univariate analysis
```

In [6]:

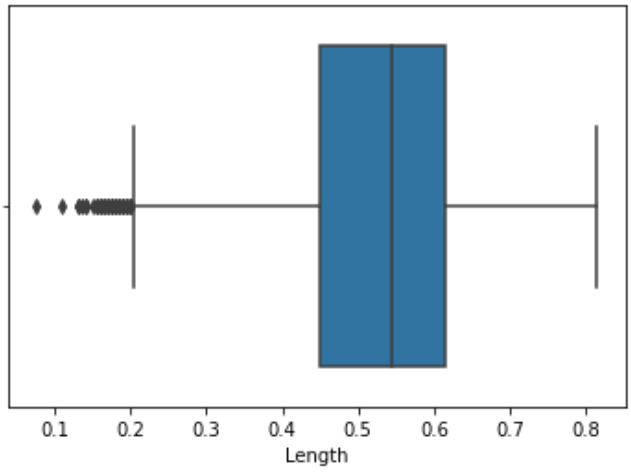
```
sns.boxplot(df.Length)
```

```
/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 argu
ment will be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  FutureWarning
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2aa021d550>
```

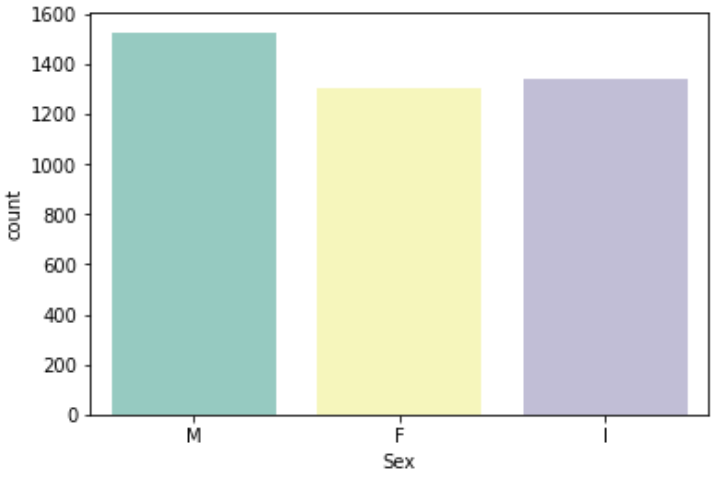


In [7]:

```
sns.countplot(x = 'Sex', data = df, palette = 'Set3')
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2aa010bf10>
```

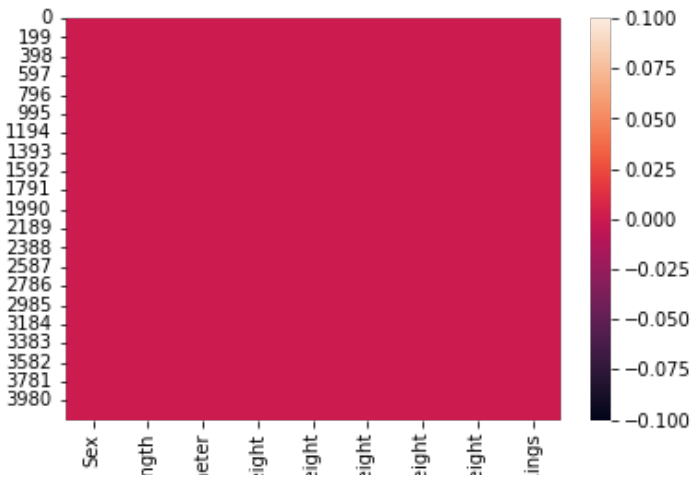


In [8]:

```
sns.heatmap(df.isnull())
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a9fc55950>
```



Lei Diam He Whole we Shucked we Viscera we Shell we R

In [9]:

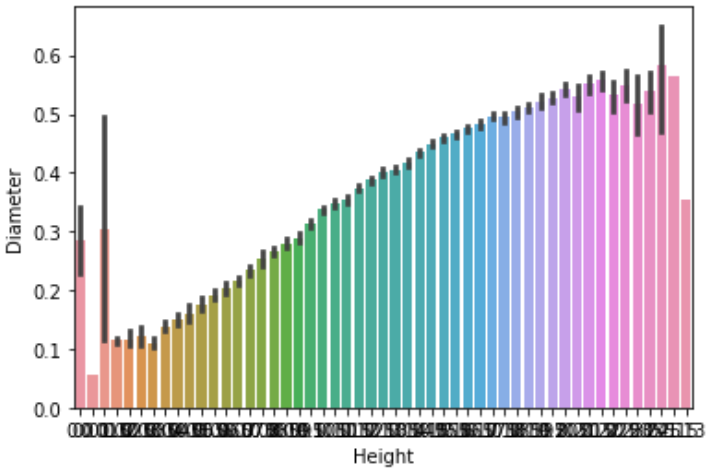
```
#Bivariate analysis
```

In [10]:

```
sns.barplot(x=df.Height,y=df.Diameter)
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a9d309ed0>
```



In [11]:

```
numerical_features = df.select_dtypes(include = [np.number]).columns
categorical_features = df.select_dtypes(include = [np.object]).columns
```

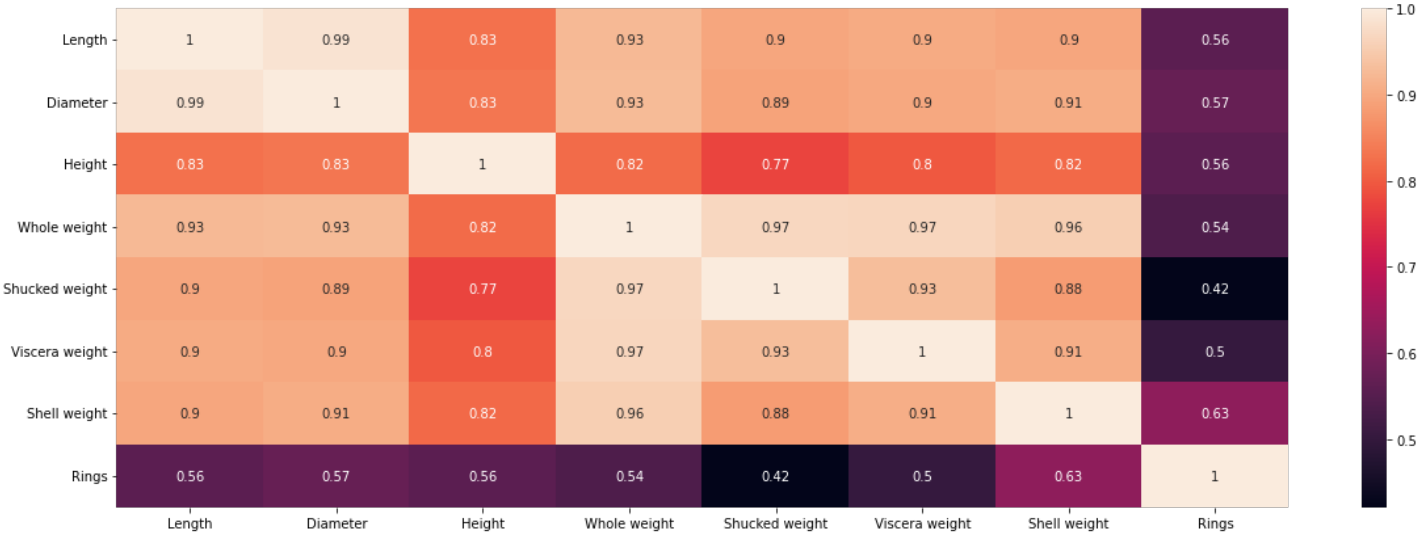
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: `np.o
bject` is a deprecated alias for the builtin `object`. To silence this warning, use `obje
ct` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
```

In [12]:

```
plt.figure(figsize = (20,7))
sns.heatmap(df[numerical_features].corr(),annot = True)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a9d070090>
```



In [13]:

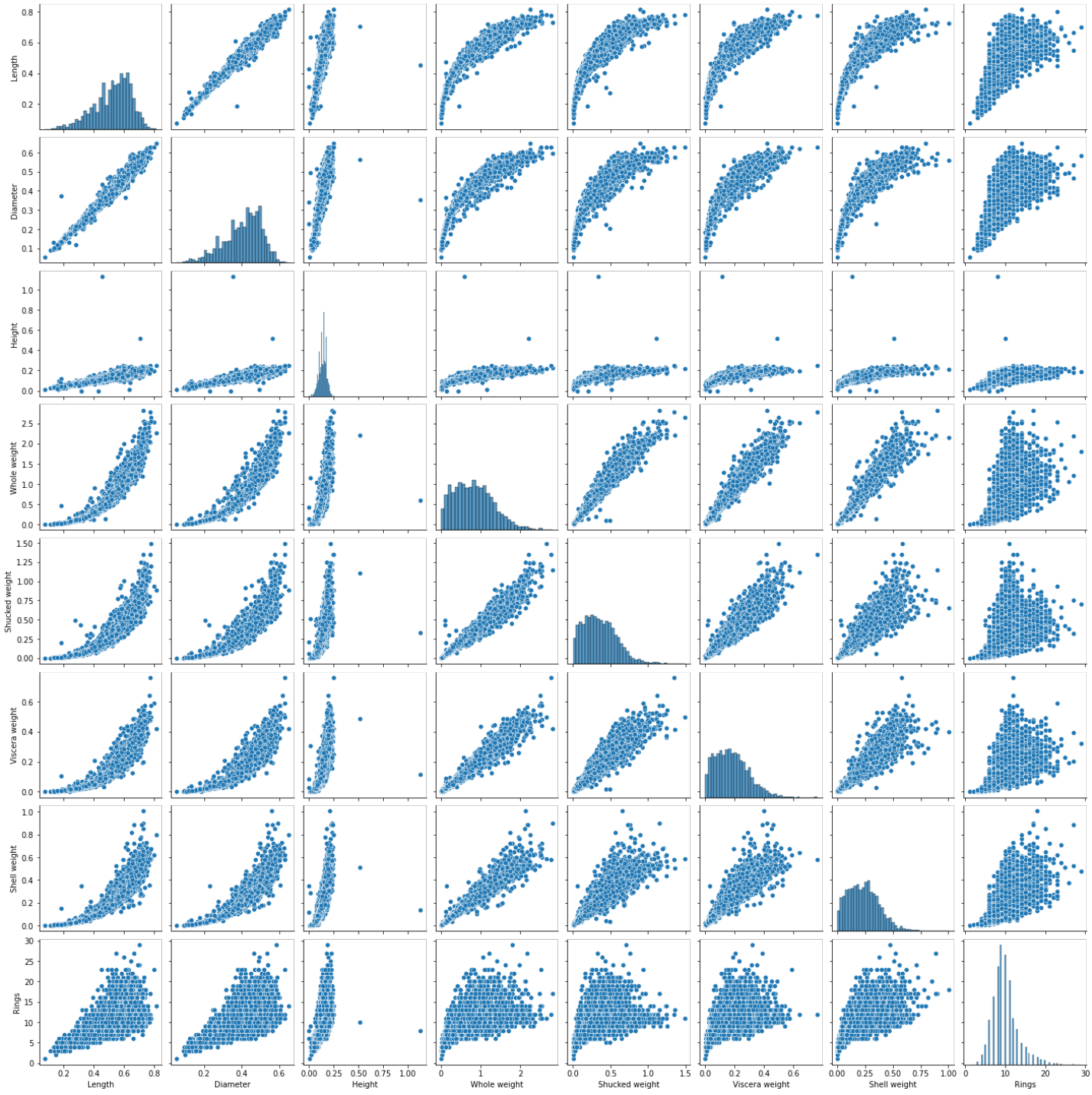
```
#Multivariate Analysis
```

In [14]:

```
sns.pairplot(df)
```

Out[14]:

```
<seaborn.axisgrid.PairGrid at 0x7f2aa032b710>
```



In [15]:

```
#Perform descriptive model on the dataset
```

In [16]:

```
df['Height'].describe()
```

Out[16]:

```
count    4177.000000
mean        0.139516
std         0.041827
```

```
min         0.000000
25%         0.115000
50%         0.140000
75%         0.165000
max         1.130000
Name: Height, dtype: float64
```

In [17]:

```
df['Height'].mean()
```

Out[17]:

```
0.13951639932966242
```

In [18]:

```
df.max()
```

Out[18]:

```
Sex                    M
Length             0.815
Diameter            0.65
Height              1.13
Whole weight      2.8255
Shucked weight     1.488
Viscera weight      0.76
Shell weight       1.005
Rings                 29
dtype: object
```

In [19]:

```
df['Sex'].value_counts()
```

Out[19]:

```
M    1528
I    1342
F    1307
Name: Sex, dtype: int64
```

In [20]:

```
df[df.Height == 0]
```

Out[20]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
1257	I	0.430	0.34	0.0	0.428	0.2065	0.0860	0.1150	8
3996	I	0.315	0.23	0.0	0.134	0.0575	0.0285	0.3505	6

In [23]:

```
df['Shucked weight'].kurtosis()
```

Out[23]:

```
0.5951236783694207
```

In [24]:

```
df['Diameter'].median()
```

Out[24]:

```
0.425
```

In [22]:

```
df['Shucked weight'].skew()
```

df['Shucked weight'].skew()

Out[22]:

```
0.7190979217612694
```

In [25]:

```
#Missing values
```

In [26]:

```
df.isna().any()
```

Out[26]:

```
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 [27]:

```
missing_values = df.isnull().sum().sort_values(ascending = False)
percentage_missing_values = (missing_values/len(df))*100
pd.concat([missing_values, percentage_missing_values], axis = 1, keys= ['Missing values'
, '% Missing'])
```

Out[27]:

	Missing values	% Missing
Sex	0	0.0
Length	0	0.0
Diameter	0	0.0
Height	0	0.0
Whole weight	0	0.0
Shucked weight	0	0.0
Viscera weight	0	0.0
Shell weight	0	0.0
Rings	0	0.0

In [28]:

```
#Find the outliers
```

In [29]:

```
q1=df.Rings.quantile(0.25)
q2=df.Rings.quantile(0.75)
iqr=q2-q1
```

In [30]:

```
print(iqr)
```

```
3.0
```

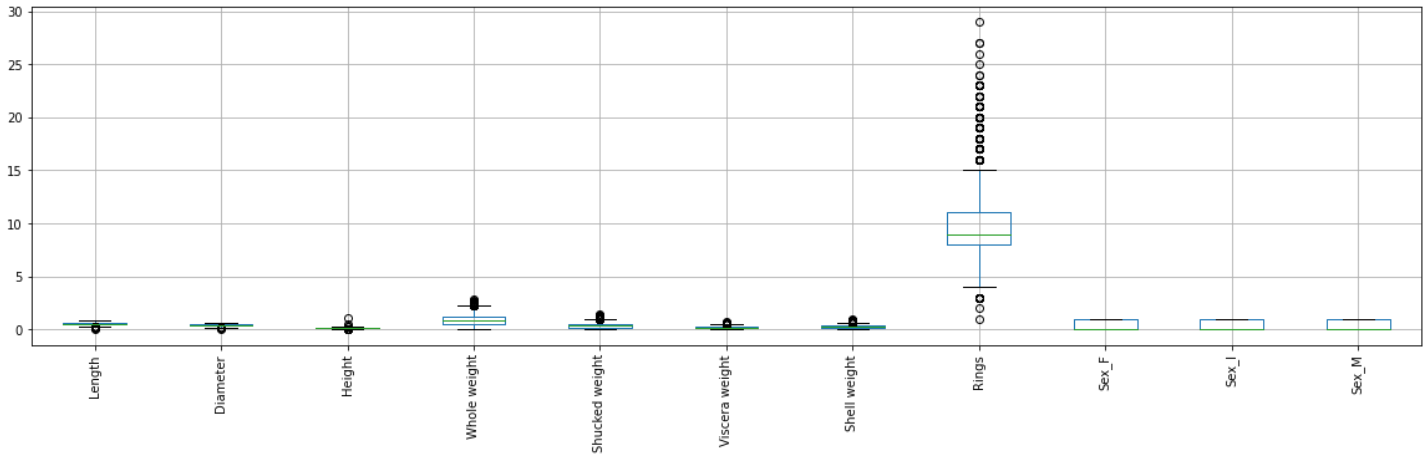
In [31]:

```
df = pd.get_dummies(df)
```

```
dummy_df = df
df.boxplot( rot = 90, figsize=(20,5))
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2a9b2a30d0>
```



In [39]:

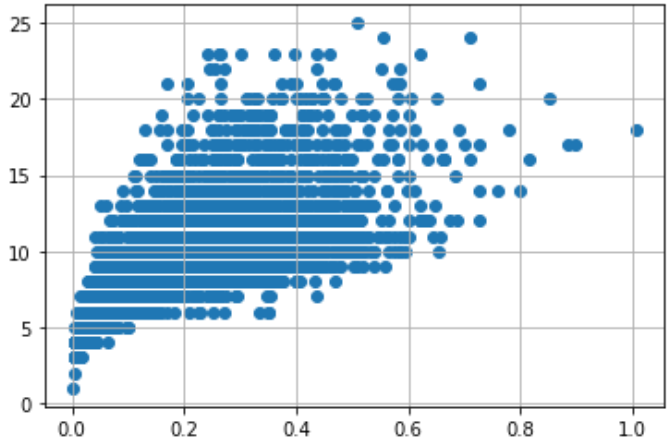
```
df['age'] = df['Rings']
df = df.drop('Rings', axis = 1)
```

In [40]:

```
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)
```

In [41]:

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



In [42]:

```
#Check for categorical columns and perform encoding
```

In [45]:

```
numerical_features = df.select_dtypes(include = [np.number]).columns
categorical_features = df.select_dtypes(include = [np.object]).columns
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: `np.o
bject` is a deprecated alias for the builtin `object`. To silence this warning, use `obje
ct` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/releas
e/1.20.0-notes.html#deprecations
```

In [46]:

```
numerical_features
categorical_features
```

Out[46]:

```
Index([], dtype='object')
```

In [47]:

```
abalone_numeric = df[['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight','Viscera weight', 'Shell weight', 'age', 'Sex_F', 'Sex_I', 'Sex_M']]
```

In [48]:

```
abalone_numeric.head()
```

Out[48]:

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

In [49]:

```
#Dependent and Independent Variables
```

In [50]:

```
x = df.iloc[:, 0:1].values
```

In [52]:

```
y = df.iloc[:, 1]
```

In [53]:

```
y
```

Out[53]:

```
0       0.365
1       0.265
2       0.420
3       0.365
4       0.255
        ...
4172    0.450
4173    0.440
4174    0.475
4175    0.485
4176    0.555
Name: Diameter, Length: 4150, dtype: float64
```

In [54]:

```
#Scaling the Independent Variables
print ("\n ORIGINAL VALUES: \n\n", x,y)
```

```
 ORIGINAL VALUES:

 [[0.455]
 [0.35 ]
 [0.53 ]
 ...
 [0.6  ]
```

```
 [0.625]
 [0.71 ]] 0       0.365
1        0.265
2        0.420
3        0.365
4        0.255
         ...
4172     0.450
4173     0.440
4174     0.475
4175     0.485
4176     0.555
Name: Diameter, Length: 4150, dtype: float64
```

In [55]:

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler(feature_range =(0, 1))
new_y= min_max_scaler.fit_transform(x,y)
print ("\n VALUES AFTER MIN MAX SCALING: \n\n", new_y)
```

```
 VALUES AFTER MIN MAX SCALING:

 [[0.51351351]
 [0.37162162]
 [0.61486486]
 ...
 [0.70945946]
 [0.74324324]
 [0.85810811]]
```

In [57]:

```
#Split the data into Training and Testing
X = df.drop('age', axis = 1)
y = df['age']
```

In [60]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import SelectKBest
standardScale = StandardScaler()
standardScale.fit_transform(X)

selectkBest = SelectKBest()
X_new = selectkBest.fit_transform(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size = 0.25)
X_train
```

Out[60]:

```
array([[0.525, 0.41 , 0.135, ..., 1.   , 0.   , 0.   ],
       [0.275, 0.175, 0.09 , ..., 0.   , 1.   , 0.   ],
       [0.68 , 0.56 , 0.195, ..., 1.   , 0.   , 0.   ],
       ...,
       [0.55 , 0.425, 0.15 , ..., 0.   , 1.   , 0.   ],
       [0.35 , 0.26 , 0.09 , ..., 0.   , 0.   , 1.   ],
       [0.57 , 0.42 , 0.14 , ..., 0.   , 0.   , 1.   ]])
```

In [61]:

```
y_train
```

Out[61]:

```
2983     8
1764     5
888     11
2029     9
3096     9
```

```
        ..
279     11
584     11
581     14
3315     9
2835     8
Name: age, Length: 3112, dtype: int64
```

In [63]:

```
# Build the model
# Linear Regression
```

In [64]:

```
from sklearn import linear_model as lm
from sklearn.linear_model import LinearRegression
model=lm.LinearRegression()
results=model.fit(X_train,y_train)
```

In [65]:

```
accuracy = model.score(X_train, y_train)
print('Accuracy of the model:', accuracy)
```

```
Accuracy of the model: 0.5354279264706927
```

In [74]:

```
#Training the model
lm = LinearRegression()
lm.fit(X_train, y_train)
y_train_pred = lm.predict(X_train)
y_train_pred
```

Out[74]:

```
array([ 8.203125,  6.34375 , 11.046875, ...,  9.359375,  8.09375 ,
        9.90625 ])
```

In [67]:

```
X_train
```

Out[67]:

```
array([[0.525, 0.41 , 0.135, ..., 1.   , 0.   , 0.   ],
       [0.275, 0.175, 0.09 , ..., 0.   , 1.   , 0.   ],
       [0.68 , 0.56 , 0.195, ..., 1.   , 0.   , 0.   ],
       ...,
       [0.55 , 0.425, 0.15 , ..., 0.   , 1.   , 0.   ],
       [0.35 , 0.26 , 0.09 , ..., 0.   , 0.   , 1.   ],
       [0.57 , 0.42 , 0.14 , ..., 0.   , 0.   , 1.   ]])
```

In [68]:

```
y_train
```

Out[68]:

```
2983     8
1764     5
888     11
2029     9
3096     9
        ..
279     11
584     11
581     14
3315     9
2835     8
Name: age, Length: 3112, dtype: int64
```

In [75]:

```
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)
```

```
Mean Squared error of training set :4.696701
```

In [70]:

```
#Testing the model
```

In [71]:

```
y_train_pred = lm.predict(X_train)
y_test_pred = lm.predict(X_test)
```

In [72]:

```
y_test_pred
```

Out[72]:

```
array([ 7.125    ,  9.       , 10.59375 , ...,  8.15625 ,  7.078125,
        9.609375])
```

In [80]:

```
X_test
```

Out[80]:

```
array([[0.35 , 0.26 , 0.095, ..., 0.   , 1.   , 0.   ],
       [0.53 , 0.42 , 0.17 , ..., 1.   , 0.   , 0.   ],
       [0.525, 0.425, 0.16 , ..., 1.   , 0.   , 0.   ],
       ...,
       [0.35 , 0.265, 0.11 , ..., 0.   , 0.   , 1.   ],
       [0.425, 0.34 , 0.105, ..., 0.   , 1.   , 0.   ],
       [0.605, 0.47 , 0.165, ..., 0.   , 0.   , 1.   ]])
```

In [81]:

```
y_test
```

Out[81]:

```
3813     8
2581     6
49       9
384     10
3832    14
        ..
3065    11
724     11
2311     7
1444     6
1006    11
Name: age, Length: 1038, dtype: int64
```

In [76]:

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

```
Mean Squared error of testing set :4.994425
```

In [77]:

```
#Measure the performance using metrices
```

In [78]:

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

```
R2 Score of training set:0.54
```

In [79]:

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

```
R2 Score of testing set:0.51
```