1. Import Libraries

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

2. Load the datasets

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

Choose Files abalone.csv

- **abalone.csv**(text/csv) - 191962 bytes, last modified: 11/4/2022 - 100% done

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

```
df.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	41
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	

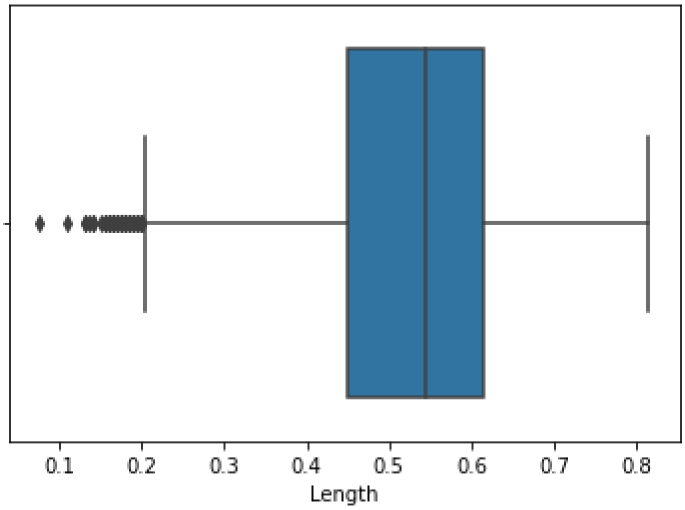
```
df.head()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shel
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	

3. Perform Below Visualizations. Univariate Analysis

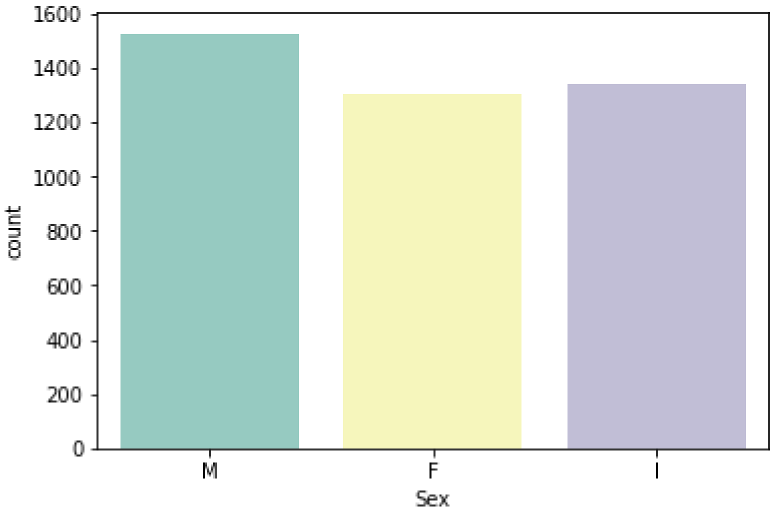
```
1 sns.boxplot(df.Length)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f79df7cd0d0>
```




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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79df704190>
```



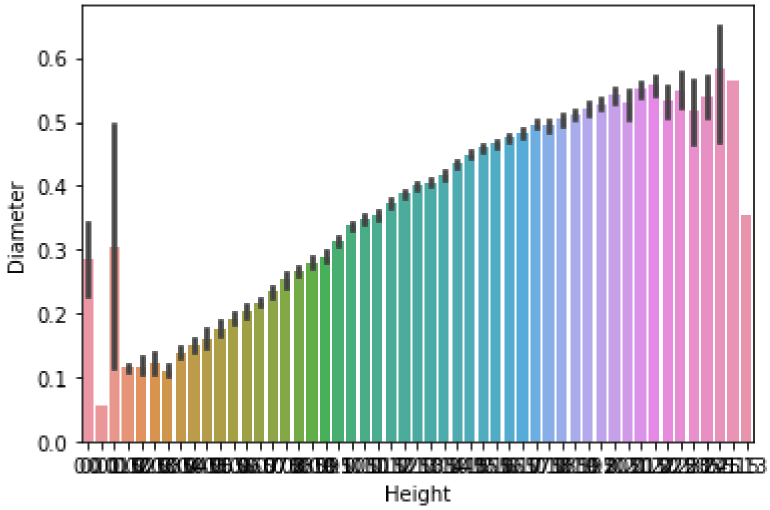

```
1 sns.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79df24a1d0>
```

Bi-Variate Analysis


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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79dc8aabd0>
```

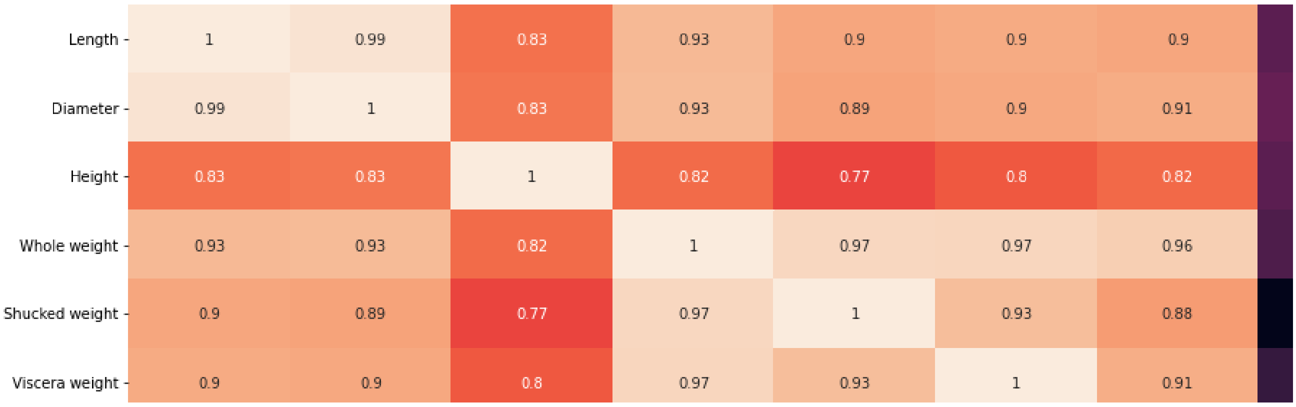



```
1 numerical_features = df.select_dtypes(include = [np.number]).columns
2 categorical_features = df.select_dtypes(include = [np.object]).columns
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: `
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
```

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

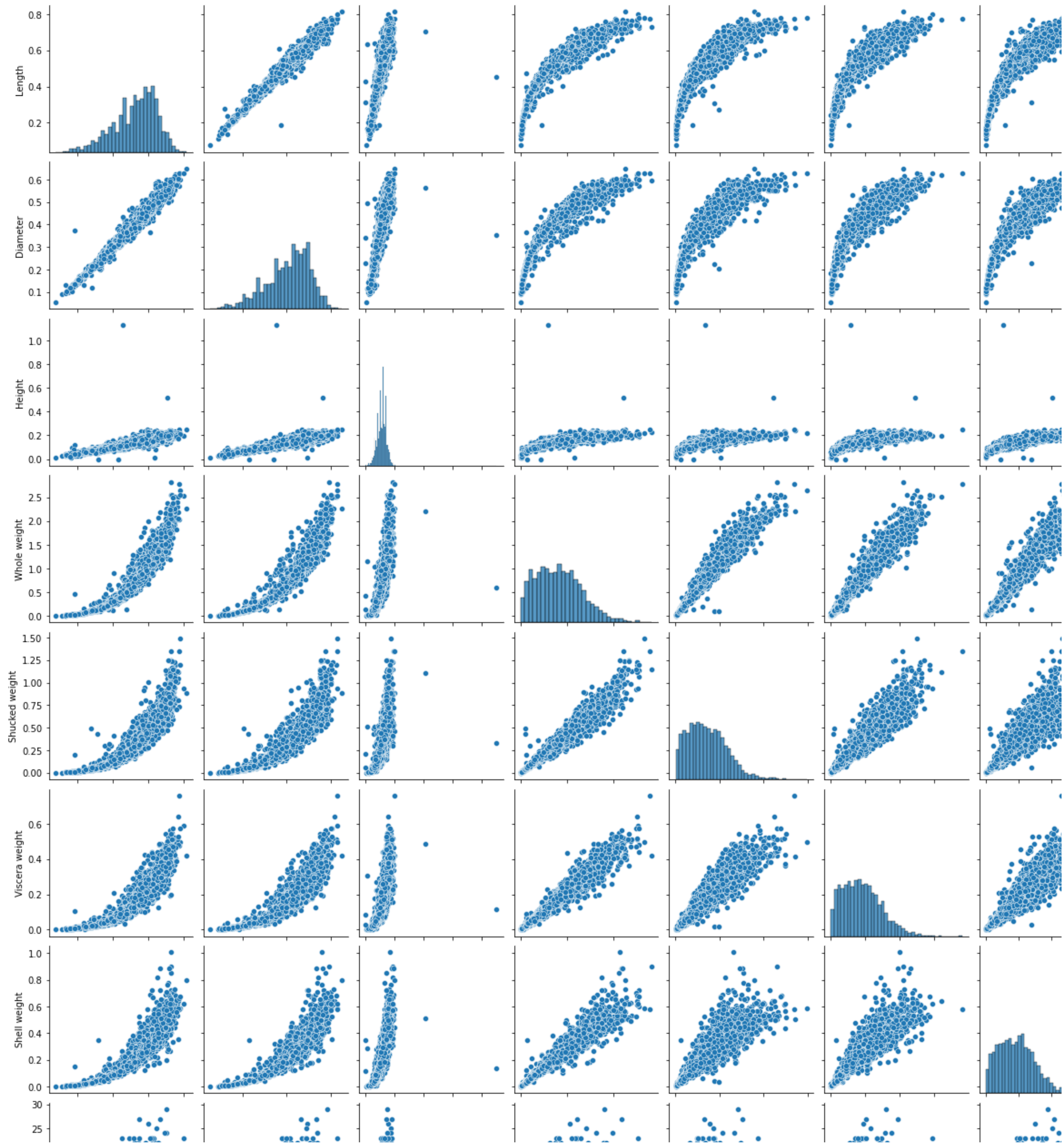
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79dc664650>
```



Multi-Variate Analysis

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

<seaborn.axisgrid.PairGrid at 0x7f79dc60e490>



4. Perform descriptive statistics on the dataset.

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

```
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
```

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

```
0.13951639932966242
```

```
1 df.max()
```

```
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
```

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

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

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

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight
1257	I	0.430	0.34	0.0	0.428	0.2065	0.0860
3996	I	0.315	0.23	0.0	0.134	0.0575	0.0285

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

```
0.5951236783694207
```

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

```
0.425
```

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

```
0.7190979217612694
```

5. Check for Missing values and deal with them.

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

```
Sex               False
Length            False
Diameter          False
```

```
Height            False
Whole weight      False
Shucked weight    False
Viscera weight    False
Shell weight      False
Rings             False
dtype: bool
```

```
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 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

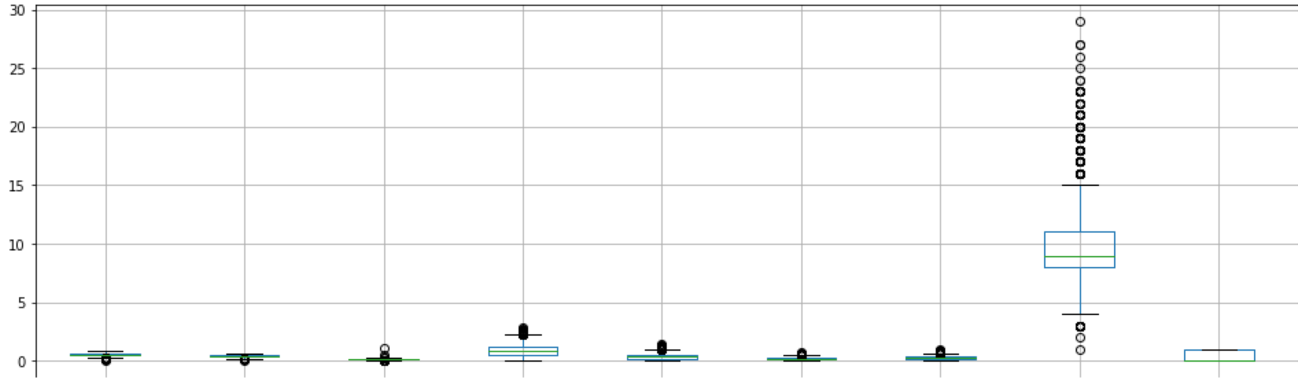
6. Find the outliers and replace them outliers

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

```
3.0
```

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

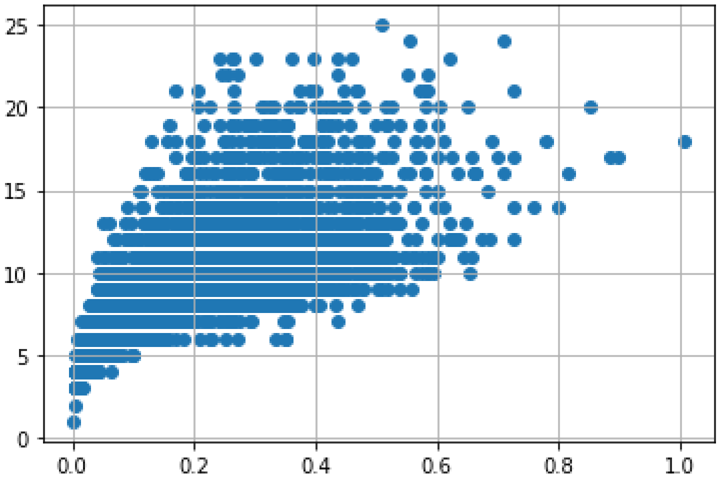
<matplotlib.axes._subplots.AxesSubplot at 0x7f79da6c3190>




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

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

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



7. Check for Categorical columns and perform encoding.

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

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: `
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
```

```
1 numerical_features
2 categorical_features
```

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

```
1 abalone_numeric = df[['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
```

```
1 abalone_numeric.head()
```

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

8. Split the data into dependent and independent variables.

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

```
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
```

9. Scale the independent variables

```
1 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
```

```
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]]
```

10. Split the data into training and testing

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

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
```

```
array([[0.63 , 0.48 , 0.15 , ..., 1.   , 0.   , 0.   ],
       [0.56 , 0.45 , 0.185, ..., 0.   , 0.   , 1.   ],
       [0.61 , 0.5  , 0.165, ..., 0.   , 0.   , 1.   ],
       ...,
       [0.64 , 0.54 , 0.175, ..., 1.   , 0.   , 0.   ],
       [0.61 , 0.485, 0.16 , ..., 0.   , 0.   , 1.   ],
       [0.575, 0.48 , 0.15 , ..., 1.   , 0.   , 0.   ]])
```

```
y_train
```

```
186     12
256     19
2156    12
```

```
3437     8
879      8
        ..
594     12
1507    10
456     15
1161     8
2665     8
Name: age, Length: 3112, dtype: int64
```

11. Build the Model

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

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

```
Accuracy of the model: 0.5460921890929435
```

12. Train the Model

```
lm = LinearRegression()
lm.fit(X_train, y_train)
y_train_pred = lm.predict(X_train)
y_train_pred
```

```
array([11.33238563, 14.39686365, 14.91438094, ..., 12.95705781,
        9.10462078, 10.32796787])
```

```
X_train
```

```
array([[0.63 , 0.48 , 0.15 , ..., 1.   , 0.   , 0.   ],
       [0.56 , 0.45 , 0.185, ..., 0.   , 0.   , 1.   ],
       [0.61 , 0.5  , 0.165, ..., 0.   , 0.   , 1.   ],
       ...,
       [0.64 , 0.54 , 0.175, ..., 1.   , 0.   , 0.   ],
       [0.61 , 0.485, 0.16 , ..., 0.   , 0.   , 1.   ],
       [0.575, 0.48 , 0.15 , ..., 1.   , 0.   , 0.   ]])
```

```
y_train
```

```
186     12
256     19
2156    12
3437     8
879      8
        ..
594     12
1507    10
```

```
456     15
1161     8
2665     8
Name: age, Length: 3112, dtype: int64
```

```
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 :4.599284
```

13. Test the Model

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

```
array([10.50313391,  8.73270156,  9.28340205, ...,  9.15795143,
        9.59125074,  8.69708903])
```

```
X_test
```

```
array([[0.485, 0.37 , 0.14 , ..., 0.   , 0.   , 1.   ],
       [0.525, 0.4  , 0.14 , ..., 0.   , 0.   , 1.   ],
       [0.63 , 0.485, 0.155, ..., 0.   , 0.   , 1.   ],
       ...,
       [0.415, 0.305, 0.105, ..., 1.   , 0.   , 0.   ],
       [0.515, 0.405, 0.12 , ..., 1.   , 0.   , 0.   ],
       [0.575, 0.445, 0.16 , ..., 0.   , 1.   , 0.   ]])
```

```
y_test
```

```
681     10
1469     8
1016     8
2723     7
3113     7
        ..
3647     8
2328    15
2325    10
471     10
3044     9
Name: age, Length: 1038, dtype: int64
```

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

```
Mean Squared error of testing set :5.050798
```

14. Measure the performance using Metrics.

```
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:0.55
```

```
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:0.50
```