1.Import Libraries

In [ ]:

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

2.Load Dataset

In [ ]:

**from** google.colab **import** files

upload**=**files**.**upload()

df **=** pd**.**read\_csv('/content/abalone.csv')

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 (1).csv

In [ ]:

df**.**describe()

Out[ ]:

|  | **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 [ ]:

df**.**head()

Out[ ]:

|  | **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 |

1. Perform Below Visualizations.

∙ Univariate Analysis

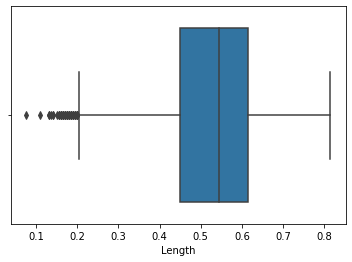
In [ ]:

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 argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

FutureWarning

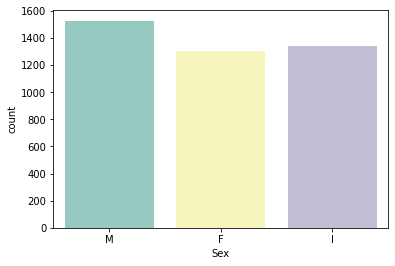
Out[ ]:



In [ ]:

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

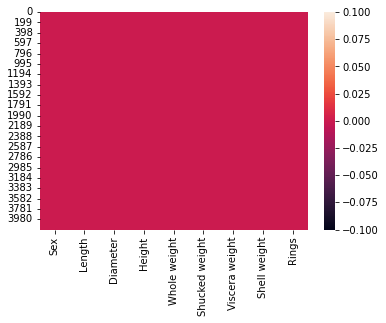
Out[ ]:



In [ ]:

sns**.**heatmap(df**.**isnull())

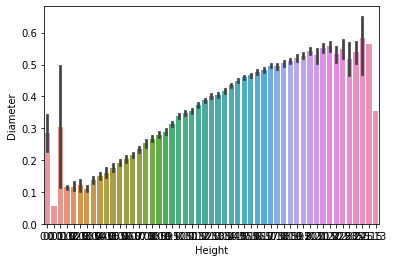
Out[ ]:



∙ Bi-Variate Analysis

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

Out[ ]:



In [ ]:

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.object` is a deprecated alias for the builtin `object`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.

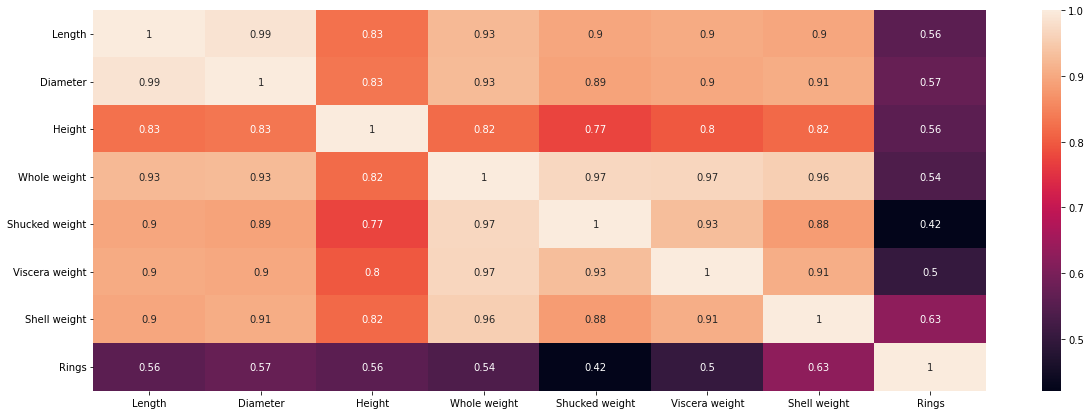
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations

In [ ]:

plt**.**figure(figsize **=** (20,7))

sns**.**heatmap(df[numerical\_features]**.**corr(),annot **=** **True**)

Out[ ]:

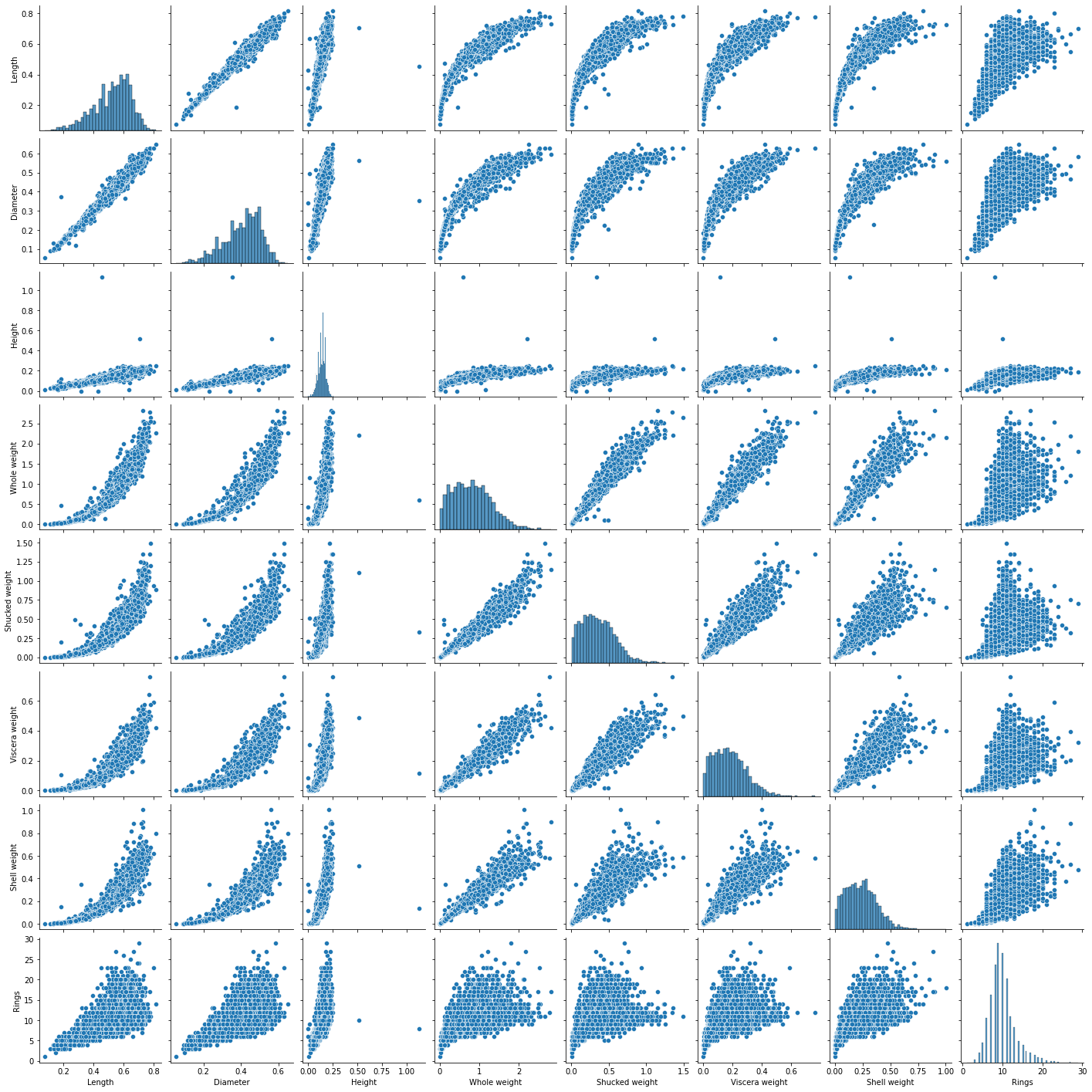


∙ Multi-Variate Analysis

In [ ]:

sns**.**pairplot(df)

Out[ ]:



1. Perform descriptive statistics on the dataset.

In [ ]:

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

Out[ ]:

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 [ ]:

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

Out[ ]:

0.13951639932966242

In [ ]:

df**.**max()

Out[ ]:

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 [ ]:

df['Sex']**.**value\_counts()

Out[ ]:

M 1528

I 1342

F 1307

Name: Sex, dtype: int64

In [ ]:

df[df**.**Height **==** 0]

Out[ ]:

|  | **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 [ ]:

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

Out[ ]:

0.5951236783694207

In [ ]:

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

Out[ ]:

0.425

In [ ]:

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

Out[ ]:

0.7190979217612694

1. Check for Missing values and deal with them.

In [ ]:

df**.**isna()**.**any()

Out[ ]:

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 [ ]:

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[ ]:

|  | **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 |

1. Find the outliers and replace them outliers

In [ ]:

q1**=**df**.**Rings**.**quantile(0.25)

q2**=**df**.**Rings**.**quantile(0.75)

iqr**=**q2**-**q1

print(iqr)

3.0

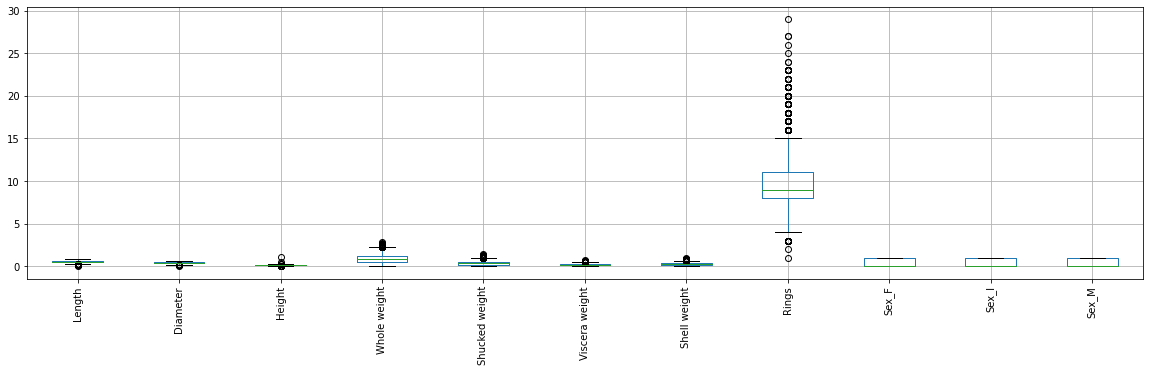
In [ ]:

df **=** pd**.**get\_dummies(df)

dummy\_df **=** df

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

Out[ ]:



In [ ]:

df['age'] **=** df['Rings']

df **=** df**.**drop('Rings', axis **=** 1)

In [ ]:

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

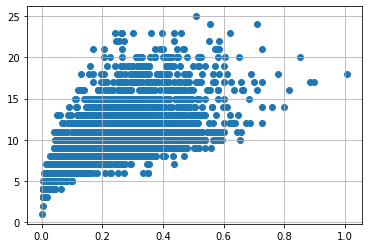
df**.**drop(df[(df['Viscera weight']**<**0.5) **&** (df['age'] **>** 25)]**.**index, inplace**=True**)

In [ ]:

var **=** 'Shell weight'

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

plt**.**grid(**True**)



1. Check for Categorical columns and perform encoding.

In [ ]:

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.object` is a deprecated alias for the builtin `object`. To silence this warning, use `object` 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/release/1.20.0-notes.html#deprecations

In [ ]:

numerical\_features

categorical\_features

Out[ ]:

Index([], dtype='object')

In [ ]:

abalone\_numeric **=** df[['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight','Viscera weight', 'Shell weight', 'age', 'Sex\_F', 'Sex\_I', 'Sex\_M']]

In [ ]:

abalone\_numeric**.**head()

Out[ ]:

|  | **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 |

1. Split the data into dependent and independent variables.

In [ ]:

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

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

y

Out[ ]:

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

1. Scale the independent variables

In [ ]:

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 [ ]:

**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]]

1. Split the data into training and testing

In [ ]:

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

Out[ ]:

array([[0.58 , 0.445, 0.125, ..., 0. , 1. , 0. ],

[0.39 , 0.3 , 0.105, ..., 0. , 1. , 0. ],

[0.63 , 0.48 , 0.16 , ..., 1. , 0. , 0. ],

...,

[0.42 , 0.305, 0.1 , ..., 0. , 1. , 0. ],

[0.475, 0.365, 0.14 , ..., 0. , 0. , 1. ],

[0.28 , 0.12 , 0.075, ..., 0. , 1. , 0. ]])

In [ ]:

y\_train

Out[ ]:

1646 9

3334 8

188 11

4030 7

2552 6

..

825 7

318 18

4107 7

3947 16

898 4

Name: age, Length: 3112, dtype: int64

1. Build the Model

In [ ]:

**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: 0.5389556158765662

1. Train the Model

In [ ]:

lm **=** LinearRegression()

lm**.**fit(X\_train, y\_train)

y\_train\_pred **=** lm**.**predict(X\_train)

y\_train\_pred

Out[ ]:

array([10.04940492, 8.17381188, 10.17705726, ..., 7.13014778,

11.1651245 , 5.25270011])

In [ ]:

X\_train

Out[ ]:

array([[0.58 , 0.445, 0.125, ..., 0. , 1. , 0. ],

[0.39 , 0.3 , 0.105, ..., 0. , 1. , 0. ],

[0.63 , 0.48 , 0.16 , ..., 1. , 0. , 0. ],

...,

[0.42 , 0.305, 0.1 , ..., 0. , 1. , 0. ],

[0.475, 0.365, 0.14 , ..., 0. , 0. , 1. ],

[0.28 , 0.12 , 0.075, ..., 0. , 1. , 0. ]])

In [ ]:

y\_train

Out[ ]:

1646 9

3334 8

188 11

4030 7

2552 6

..

825 7

318 18

4107 7

3947 16

898 4

Name: age, Length: 3112, dtype: int64

In [ ]:

**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 :4.753595

1. Test the Model

In [ ]:

y\_train\_pred **=** lm**.**predict(X\_train)

y\_test\_pred **=** lm**.**predict(X\_test)

y\_test\_pred

Out[ ]:

array([ 5.76375739, 10.86128032, 11.4225637 , ..., 4.84179968,

9.79104261, 8.3178401 ])

In [ ]:

X\_test

Out[ ]:

array([[0.255, 0.19 , 0.05 , ..., 0. , 1. , 0. ],

[0.64 , 0.5 , 0.17 , ..., 1. , 0. , 0. ],

[0.625, 0.47 , 0.18 , ..., 0. , 0. , 1. ],

...,

[0.165, 0.12 , 0.05 , ..., 0. , 1. , 0. ],

[0.5 , 0.385, 0.115, ..., 1. , 0. , 0. ],

[0.42 , 0.32 , 0.11 , ..., 0. , 1. , 0. ]])

In [ ]:

y\_test

Out[ ]:

895 6

1022 11

1498 11

3673 10

2603 9

..

2381 5

2889 8

3472 3

622 12

3015 6

Name: age, Length: 1038, dtype: int64

In [ ]:

p **=** mean\_squared\_error(y\_test, y\_test\_pred)

print('Mean Squared error of testing set :%2f'**%**p)

Mean Squared error of testing set :4.620467

1. Measure the performance using Metrics.
2. **from** sklearn.metrics **import** r2\_score
3. s **=** r2\_score(y\_train, y\_train\_pred)
4. print('R2 Score of training set:%.2f'**%**s)
5. R2 Score of training set:0.54
6. In [ ]:
7. **from** sklearn.metrics **import** r2\_score
8. p **=** r2\_score(y\_test, y\_test\_pred)
9. print('R2 Score of testing set:%.2f'**%**p)
10. R2 Score of testing set:0.52