

ASSIGNMENT – 3

Assignment Date	05 October 2022
Student Name	Santha rajan.N
Student Roll Number	820319104041
Maximum Marks	2 Marks

Building a Regression Model

1. Download the dataset: [Dataset](#)

```
data=pd.read_csv("abalone.csv")
```

2. Load the dataset into the tool.

```
data.head()
```

```
In [2]: data.head()
```

```
Out[2]:
```

	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

3. Perform Below Visualizations.

- Univariate Analysis

```
#univariate analysis
```

```
cols = 3
```

```
rows = 3
```

```
num_cols = data.select_dtypes(exclude='object').columns
```

```
fig = plt.figure( figsize=(cols*5, rows*5))
```

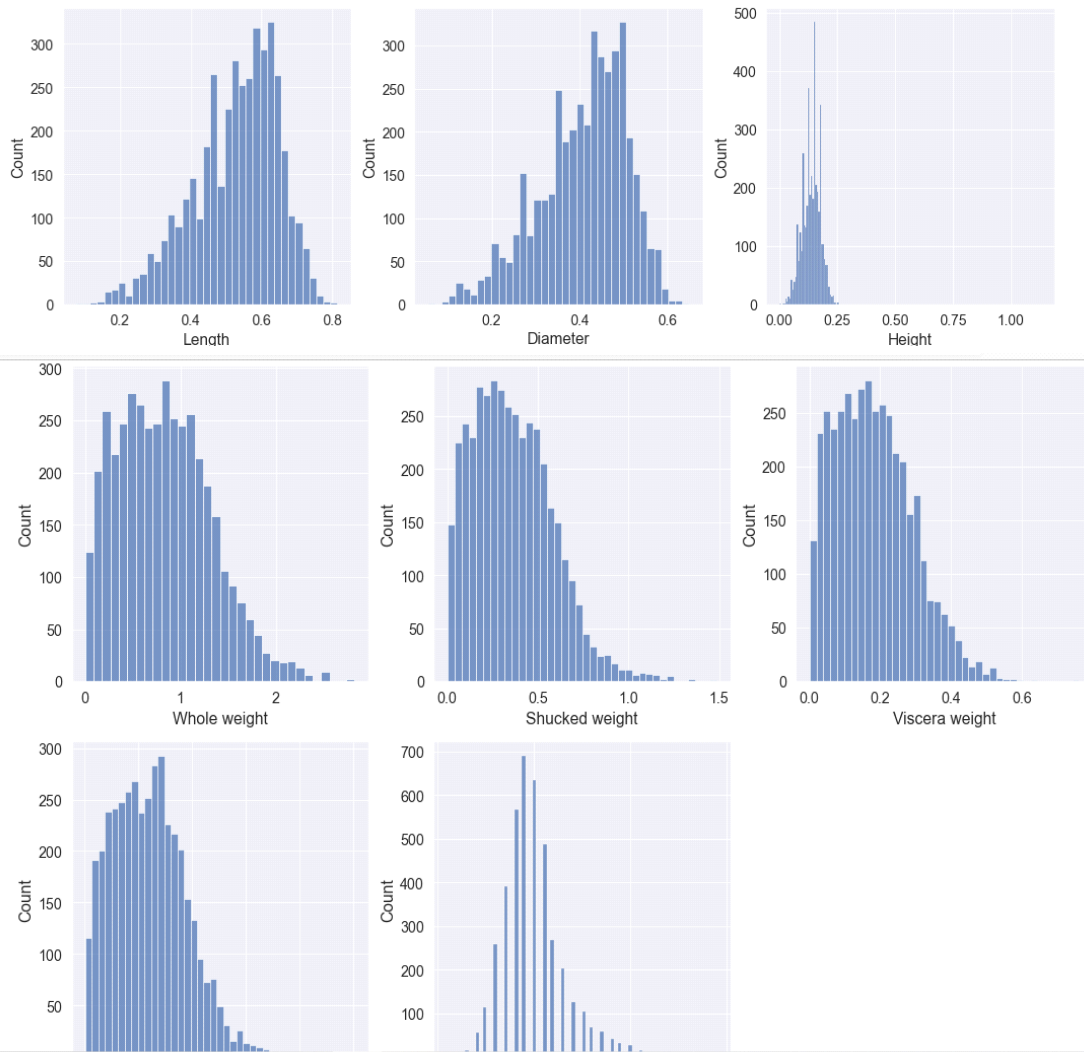
```
for i, col in enumerate(num_cols):
```

```
    ax=fig.add_subplot(rows,cols,i+1)
```

```
sns.histplot(x = data[col], ax = ax)
fig.tight_layout()
plt.show()
```

In [5]:

```
#univariate analysis
cols = 3
rows = 3
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure(figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):
    ax=fig.add_subplot(rows,cols,i+1)
    sns.histplot(x = data[col], ax = ax)
fig.tight_layout()
plt.show()
```



Bi-Variate Analysis

#Bivariate analysis

import matplotlib.pyplot **as** plt

#create scatterplot of hours vs. score

plt.scatter(data.Height, data.Diameter)

plt.title('Height vs Diameter')

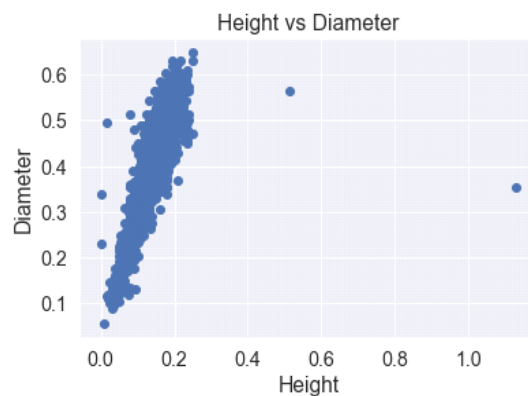
plt.xlabel('Height')

plt.ylabel('Diameter')

```
In [7]: #Bivariate analysis
import matplotlib.pyplot as plt

#create scatterplot of hours vs. score
plt.scatter(data.Height, data.Diameter)
plt.title('Height vs Diameter')
plt.xlabel('Height')
plt.ylabel('Diameter')
```

Out[7]: Text(0, 0.5, 'Diameter')

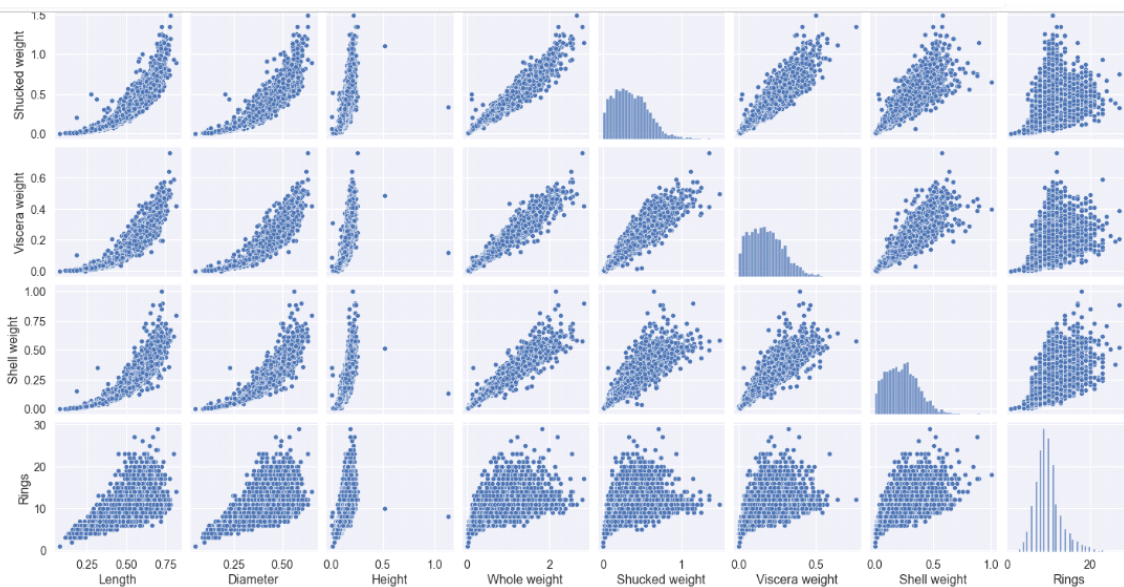
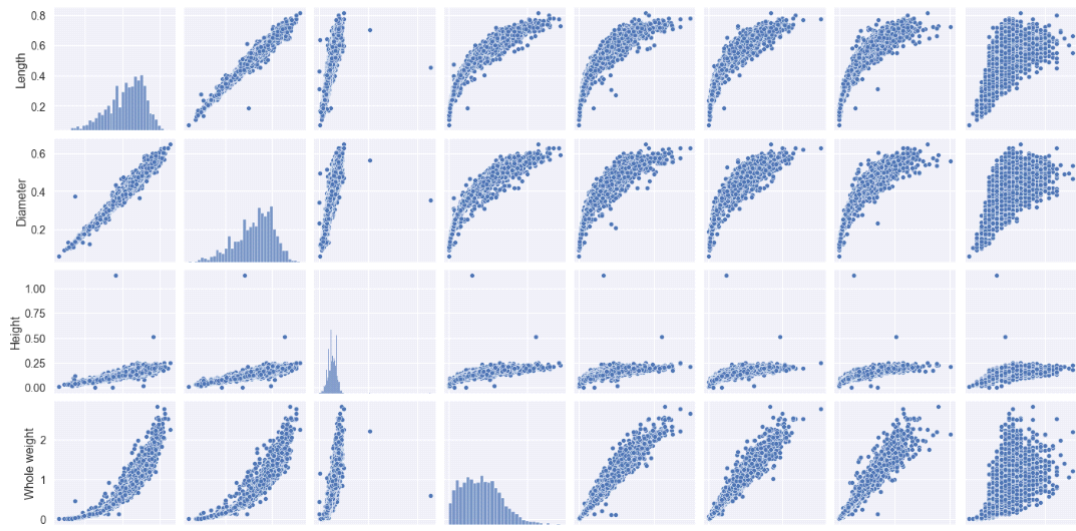


Multi-Variate Analysis

#multivariate analysis

sns.pairplot(d

```
In [8]: #multivariate analysis  
sns.pairplot(data);
```



4. Perform descriptive statistics on the dataset

`data.mean()`

`data.median()`

```
In [9]: data.mean()
```

C:\Users\Hi\AppData\Local\Temp\ipykernel_16792\983992179.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

```
data.mean()
```

Length	0.523992
Diameter	0.407881
Height	0.139516
Whole weight	0.828742
Shucked weight	0.359367
Viscera weight	0.180594
Shell weight	0.238831
Rings	9.933684

dtype: float64

```
In [10]: data.median()
```

C:\Users\Hi\AppData\Local\Temp\ipykernel_16792\3972556868.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

```
data.median()
```

Length	0.5450
Diameter	0.4250
Height	0.1400
Whole weight	0.7995
Shucked weight	0.3360
Viscera weight	0.1710
Shell weight	0.2340
Rings	9.0000

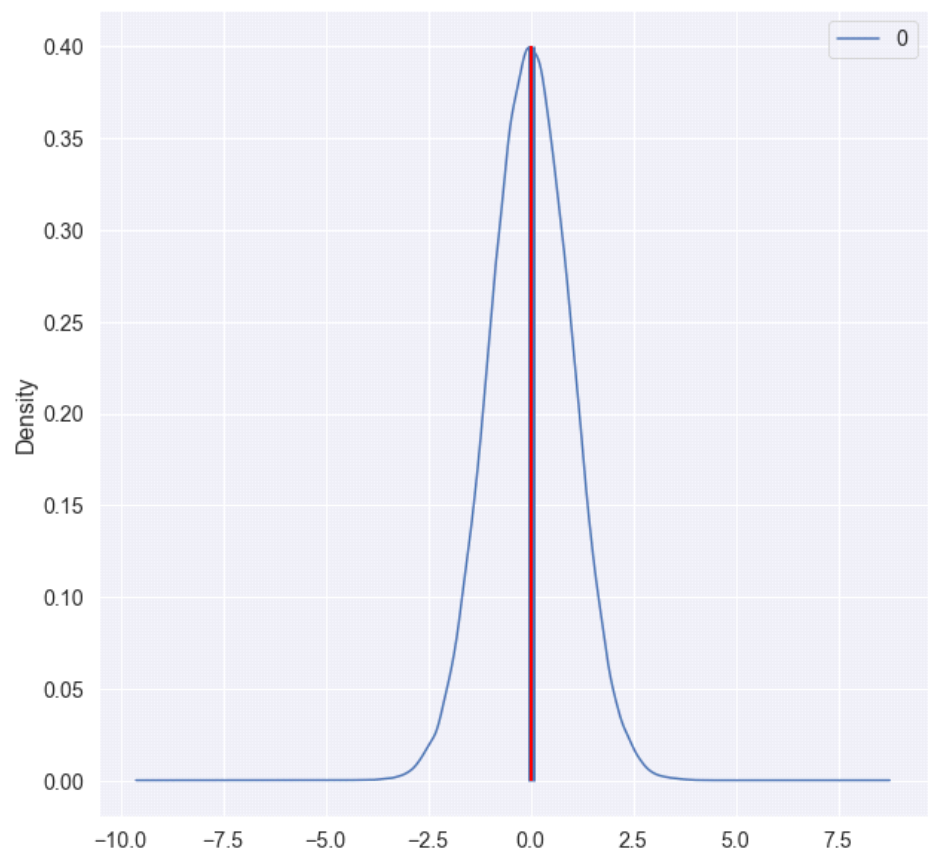
dtype: float64

```
In [11]: norm_data = pd.DataFrame(np.random.normal(size=100000))

norm_data.plot(kind="density",
               figsize=(10,10));

plt.vlines(norm_data.mean(),      # Plot black line at mean
           ymin=0,
           ymax=0.4,
           linewidth=5.0);

plt.vlines(norm_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.4,
           linewidth=2.0,
           color="red");
```



5. Check for Missing values and deal with them.

#identifying the missing value

```
df = pd.DataFrame(data)
df.isnull()
```

```
In [12]: #identifying the missing value
df = pd.DataFrame(data)
df.isnull()
```

```
Out[12]:
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...
4172	False	False	False	False	False	False	False	False	False
4173	False	False	False	False	False	False	False	False	False
4174	False	False	False	False	False	False	False	False	False
4175	False	False	False	False	False	False	False	False	False
4176	False	False	False	False	False	False	False	False	False

4177 rows × 9 columns

#filling the missing value with previous value

```
df.fillna(method='pad')
```

```
In [13]: #filling the missing value with previous value
df.fillna(method='pad')
```

```
Out[13]:
```

	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	0.1500	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
...
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

4177 rows × 9 columns

#filling null values in missing values

data[0:]

```
In [14]: #filling null values in missing values
data[0:]
```

```
Out[14]:
```

	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	0.1500	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
...
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

4177 rows × 9 columns

6. Find the outliers and replace them outliers

#identifying the outliers

```
print(df['Shell weight'].skew())
```

```
df['Shell weight'].describe()
```

```
In [15]: #identifying the outliers
print(df['Shell weight'].skew())
df['Shell weight'].describe()
```

```
Out[15]:
```

	0.6209268251392077
count	4177.000000
mean	0.238831
std	0.139203
min	0.001500
25%	0.130000
50%	0.234000
75%	0.329000
max	1.005000
Name: Shell weight, dtype: float64	

#replacing the outliers

```
print(df['Shell weight'].quantile(0.50))
print(df['Shell weight'].quantile(0.95))
df['Shell weight'] = np.where(df['Shell weight'] > 325, 140, df['Shell weight'])
df.describe()
```

```
In [16]: #replacing the outliers
print(df['Shell weight'].quantile(0.50))
print(df['Shell weight'].quantile(0.95))
df['Shell weight'] = np.where(df['Shell weight'] > 325, 140, df['Shell weight'])
df.describe()

0.234
0.48

Out[16]:
```

	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

7. Check for Categorical columns and perform encoding.

#perform encoding

```
from sklearn.compose import make_column_selector as selector
categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(data)
categorical_columns
```

```
In [17]: #perform encoding
from sklearn.compose import make_column_selector as selector

categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(data)
categorical_columns

Out[17]: ['Sex']
```

```
data_categorical = data[categorical_columns]
data_categorical.head()
```

```
In [18]: data_categorical = data[categorical_columns]
data_categorical.head()
```

```
Out[18]:
```

	Sex
0	M
1	M
2	F
3	M
4	I

8. Split the data into dependent and independent variables.

```
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df['Sex']= label_encoder.fit_transform(df['Sex'])
df['Sex'].unique()
X= data.iloc[ : , :-1].values
y= data.iloc[ : , 4].values
print(X,y)
# import packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# importing data
print(df.shape)
# head of the data
print('Head of the dataframe : ')
print(df.head())
print(df.columns)
```

```

X= df['Whole weight']
y=df['Shucked weight']
# using the train test split function
X_train,X_test,y_train,y_test = train_test_split(
X,y , random_state=104,test_size=0.25, shuffle=True)
# printing out train and test sets
print('X_train : ')
print(X_train.head())
print(X_train.shape)
print("")
print('X_test : ')
print(X_test.head())
print(X_test.shape)
print("")
print('y_train : ')
print(y_train.head())
print(y_train.shape)
print("")
print('y_test : ')
print(y_test.head())
print(y_test.shape)

```

```

In [19]: from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['Sex'].unique()

Out[19]: array([2, 0, 1])

In [20]: X= data.iloc[ : , :-1].values
y= data.iloc[ : , 4].values
print(X,y)

[['M' 0.455 0.365 ... 0.2245 0.101 0.15]
['M' 0.35 0.265 ... 0.0995 0.0485 0.07]
['F' 0.53 0.42 ... 0.2565 0.1415 0.21]
...
['M' 0.6 0.475 ... 0.5255 0.2875 0.308]
['F' 0.625 0.485 ... 0.531 0.261 0.296]
['M' 0.71 0.555 ... 0.9455 0.3765 0.495]] [0.514 0.2255 0.677 ... 1.176 1.0945 1.9485]

```

```
# importing data
print(df.shape)

# head of the data
print('Head of the dataframe : ')
print(df.head())

print(df.columns)

X= df['Whole weight']
y=df['Shucked weight']

# using the train test split function
X_train, X_test, y_train, y_test = train_test_split(
X,y , random_state=104,test_size=0.25, shuffle=True)

# printing out train and test sets

print('X_train : ')
print(X_train.head())
print(X_train.shape)

print('')
print('X_test : ')
print(X_test.head())
print(X_test.shape)

print('')
print('y_train : ')
print(y_train.head())
print(y_train.shape)

print('')
print('y_test : ')
print(y_test.head())
print(y_test.shape)
```

```
(4177, 9)
Head of the dataframe :
   Sex  Length  Diameter  Height  Whole weight  Shucked weight \
0    2    0.455    0.365    0.095    0.5140    0.2245
1    2    0.350    0.265    0.090    0.2255    0.0995
2    0    0.530    0.420    0.135    0.6770    0.2565
3    2    0.440    0.365    0.125    0.5160    0.2155
4    1    0.330    0.255    0.080    0.2050    0.0895

   Viscera weight  Shell weight  Rings
0    0.1010    0.150    15
1    0.0485    0.070    7
2    0.1415    0.210    9
3    0.1140    0.155    10
4    0.0395    0.055    7
Index(['Sex', 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
      'Viscera weight', 'Shell weight', 'Rings'],
      dtype='object')
X_train :
437    0.2520
1331    0.8730
1611    0.7625
1394    1.5210
396     0.7155
Name: Whole weight, dtype: float64
(3132,)

X_test :
4087    0.9840
1699    1.4890
1868    0.6965
2984    1.2240
5        0.3515
Name: Whole weight, dtype: float64
(1045,)
```

Activate Window
Go to Settings to activate Windows Defender

9. Scale the independent variables

#scaling

```
df_scaled = df.copy()
col_names = ['Shucked weight', 'Whole weight']
features = df_scaled[col_names]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
from sklearn.preprocessing
```

```
In [22]: #scaling
df_scaled = df.copy()
col_names = ['Shucked weight', 'Whole weight']
features = df_scaled[col_names]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(5, 10))

df_scaled[col_names] = scaler.fit_transform(features.values)
df_scaled

Out[22]:
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	2	0.455	0.365	0.095	5.906676	5.751513	0.1010	0.1500	15
1	2	0.350	0.265	0.090	5.395785	5.331204	0.0485	0.0700	7
2	0	0.530	0.420	0.135	6.195325	5.859112	0.1415	0.2100	9
3	2	0.440	0.365	0.125	5.910218	5.721251	0.1140	0.1550	10
4	1	0.330	0.255	0.080	5.359483	5.297579	0.0395	0.0550	7
...
4172	0	0.565	0.450	0.165	6.567204	6.240753	0.2390	0.2490	11
4173	2	0.590	0.440	0.135	6.707101	6.472764	0.2145	0.2605	10
4174	2	0.600	0.475	0.205	7.078980	6.763618	0.2875	0.3080	9
4175	0	0.625	0.485	0.150	6.934656	6.782112	0.2610	0.2960	10
4176	2	0.710	0.555	0.195	8.446963	8.175857	0.3765	0.4950	12

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(5, 10))
```

```
df_scaled[col_names] = scaler.fit_transform(features.values)
df_scaled
```

10. Split the data into training and testing

#testing and training

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

split the dataset

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=0)
print(X_train, X_test, y_train, y_test)
```

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

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=0)
print(X_train, X_test, y_train, y_test)
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	\
678	0	0.450	0.380	0.165	0.8165	0.2500	
3009	1	0.255	0.185	0.065	0.0740	0.0305	
1906	1	0.575	0.450	0.135	0.8245	0.3375	
768	0	0.550	0.430	0.155	0.7850	0.2890	
2781	2	0.595	0.475	0.140	1.0305	0.4925	
...	
1033	2	0.650	0.525	0.185	1.6220	0.6645	
3264	0	0.655	0.500	0.140	1.1705	0.5405	
1653	2	0.595	0.450	0.145	0.9590	0.4630	
2607	0	0.625	0.490	0.165	1.1270	0.4770	
2732	1	0.410	0.325	0.110	0.3260	0.1325	
	Viscera weight	Shell weight					
678	0.1915	0.2650					
3009	0.0165	0.0200					
1906	0.2115	0.2390					
768	0.2270	0.2330					
2781	0.2170	0.2780					
...					
1033	0.3225	0.4770					
3264	0.3175	0.2850					
1653	0.2065	0.2535					
2607	0.2365	0.3185					
2732	0.0750	0.1010					
[3968 rows x 8 columns]							
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	\
668	2	0.550	0.425	0.155	0.9175	0.2775	
1580	1	0.500	0.400	0.120	0.6160	0.2610	
3784	2	0.620	0.480	0.155	1.2555	0.5270	
463	1	0.220	0.165	0.055	0.0545	0.0215	
2615	2	0.645	0.500	0.175	1.5105	0.6735	

```

...    ...    ...    ...    ...    ...    ...
1670  0  0.610  0.485  0.150  1.2405  0.6025
3055  0  0.610  0.495  0.160  1.0890  0.4690
3366  2  0.280  0.210  0.065  0.0905  0.0350
1410  0  0.665  0.530  0.180  1.4910  0.6345
4035  1  0.520  0.410  0.140  0.5995  0.2420

```

```

Viscera weight  Shell weight

```

```

668      0.2430  0.3350
1580     0.1430  0.1935
3784     0.3740  0.3175
463      0.0120  0.0200
2615     0.3755  0.3775
...      ...
1670     0.2915  0.3005
3055     0.1980  0.3840
3366     0.0200  0.0300
1410     0.3420  0.4350
4035     0.1375  0.1820

```

```

[209 rows x 8 columns] 678    23

```

```

3009    4
1906    11
768     11
2781    10
...
1033    10
3264    12
1653    10
2607     9
2732     8
Name: Rings, Length: 3968, dtype: int64 668    13
1580     8
3784    11
463      5
2615    12
...
1670    12
3055    11
3366     5
1410    10

```

In [25]: X_train

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
678	0	0.450	0.380	0.165	0.8165	0.2500	0.1915	0.2650
3009	1	0.255	0.185	0.065	0.0740	0.0305	0.0165	0.0200
1906	1	0.575	0.450	0.135	0.8245	0.3375	0.2115	0.2390
768	0	0.550	0.430	0.155	0.7850	0.2890	0.2270	0.2330
2781	2	0.595	0.475	0.140	1.0305	0.4925	0.2170	0.2780
...
1033	2	0.650	0.525	0.185	1.6220	0.6645	0.3225	0.4770
3264	0	0.655	0.500	0.140	1.1705	0.5405	0.3175	0.2850
1653	2	0.595	0.450	0.145	0.9590	0.4630	0.2065	0.2535
2607	0	0.625	0.490	0.165	1.1270	0.4770	0.2365	0.3185
2732	1	0.410	0.325	0.110	0.3260	0.1325	0.0750	0.1010

3968 rows × 8 columns

In [26]: `x_test`

Out[26]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
668	2	0.550	0.425	0.155	0.9175	0.2775	0.2430	0.3350
1580	1	0.500	0.400	0.120	0.6160	0.2610	0.1430	0.1935
3784	2	0.620	0.480	0.155	1.2555	0.5270	0.3740	0.3175
463	1	0.220	0.165	0.055	0.0545	0.0215	0.0120	0.0200
2615	2	0.645	0.500	0.175	1.5105	0.6735	0.3755	0.3775
...
1670	0	0.610	0.485	0.150	1.2405	0.6025	0.2915	0.3085
3055	0	0.610	0.495	0.160	1.0890	0.4690	0.1980	0.3840
3366	2	0.280	0.210	0.065	0.0905	0.0350	0.0200	0.0300
1410	0	0.665	0.530	0.180	1.4910	0.6345	0.3420	0.4350
4035	1	0.520	0.410	0.140	0.5995	0.2420	0.1375	0.1820

209 rows × 8 columns

In [27]: `y_train`

Out[27]:

```
678      23
3009      4
1906     11
768      11
2781     10
..
1033     10
3264     12
1653     10
2607      9
2732      8
Name: Rings, Length: 3968, dtype: int64
```

In [28]: `y_test`

Out[28]:

```
668      13
1580      8
3784     11
463       5
2615     12
..
1670     12
3055     11
3366      5
1410     10
4035     11
Name: Rings, Length: 209, dtype: int64
```


11. Build the Model

Evaluate the model on the test data

```
predictions = model.predict(X_test)
predictions
```

```
In [30]: # Evaluate the model on the test data
          predictions = model.predict(X_test)
          predictions

Out[30]: array([17, 12, 14,  5, 10, 13,  8,  8, 12, 10,  8,  5,  8,  9,  6, 13, 10,
        15,  9,  8,  6,  5,  9, 10, 10, 10,  4, 12, 14, 11,  9,  4, 11, 18,
         6,  8,  9,  9,  7, 11, 12, 13, 14, 10, 13,  8,  9, 10, 10,  7,  9,
         6,  9, 16, 10,  6,  6,  7,  6,  6,  9,  8,  9,  7,  7, 11, 13, 13,
        12, 10, 10, 14, 10,  9, 10,  9, 10,  8,  9,  8,  9,  6,  6, 10, 12,
         9, 10, 15,  6,  5,  8,  8,  8,  6, 11,  5,  9,  9, 10, 10, 11, 13,
        10, 12,  5, 11,  8,  6, 10, 20, 10, 11, 10,  9, 16,  9,  9, 12,  5,
         9,  7,  9,  8, 11, 13,  9, 13, 12,  6,  9,  9,  8,  9, 11, 13, 10,
        10,  8,  6, 18, 14, 12,  8,  8,  9,  8,  9,  7,  7,  6,  8, 13,  8,
         8,  9,  8, 15,  7, 10,  7,  9, 10,  9,  9,  6, 20,  7,  6, 10, 11,
        10,  5,  6, 10, 21, 11,  6,  8,  6, 13, 11,  9,  7, 10, 13, 10, 10,
         8,  7,  9,  8,  9,  8, 10, 13, 13,  7,  7,  4, 15, 10, 11, 12,  9,
```

12. Train the Model

```
# Select algorithm
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import accuracy_score
```

```
model = DecisionTreeClassifier()
```

```
# Fit model to the data
```

```
model.fit(X_train, y_train)
```

Check model performance on training data

```
predictions = model.predict(X_train)
```

```
print(accuracy_score(y_train, predictions))
```

```
In [29]: # Select algorithm
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
model = DecisionTreeClassifier()
# Fit model to the data
model.fit(X_train, y_train)
# Check model performance on training data
predictions = model.predict(X_train)
print(accuracy_score(y_train, predictions))
```

1.0

13. Test the Model

Evaluate the model on the test data

```
predictions = model.predict(X_test)
predictions
```

```
In [30]: # Evaluate the model on the test data
         predictions = model.predict(X_test)
         predictions

Out[30]: array([17, 12, 14,  5, 10, 13,  8,  8, 12, 10,  8,  5,  8,  9,  6, 13, 10,
          15,  9,  8,  6,  5,  9, 10, 10, 10,  4, 12, 14, 11,  9,  4, 11, 18,
           6,  8,  9,  9,  7, 11, 12, 13, 14, 10, 13,  8,  9, 10, 10,  7,  9,
           6,  9, 16, 10,  6,  6,  7,  6,  6,  9,  8,  9,  7,  7, 11, 13, 13,
          12, 10, 10, 14, 10,  9, 10,  9, 10,  8,  9,  8,  9,  6,  6, 10, 12,
           9, 10, 15,  6,  5,  8,  8,  8,  6, 11,  5,  9,  9, 10, 10, 11, 13,
          10, 12,  5, 11,  8,  6, 10, 20, 10, 11, 10,  9, 16,  9,  9, 12,  5,
           9,  7,  9,  8, 11, 13,  9, 13, 12,  6,  9,  9,  8,  9, 11, 13, 10,
          10,  8,  6, 18, 14, 12,  8,  8,  9,  8,  9,  7,  7,  6,  8, 13,  8,
           8,  9,  8, 15,  7, 10,  7,  9, 10,  9,  9,  6, 20,  7,  6, 10, 11,
          10,  5,  6, 10, 21, 11,  6,  8,  6, 13, 11,  9,  7, 10, 13, 10, 10,
           8,  7,  9,  8,  9,  8, 10, 13, 13,  7,  7,  4, 15, 10, 11, 12,  9,
           8, 11,  7, 10, 10], dtype=int64)

In [31]: print(accuracy_score(y_test, predictions))

0.215311004784689
```

14. Measure the performance using Metrics.

import os

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin'

from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report

from sklearn.metrics import roc_auc_score

from sklearn.metrics import log_loss

X_actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]

Y_predic = [1, 0, 1, 1, 1, 0, 1, 1, 0, 0]

results = confusion_matrix(X_actual, Y_predic)

print ('Confusion Matrix :')

print(results)

print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))

print ('Classification Report : ')

print (classification_report(X_actual, Y_predic))

print('AUC-ROC:',roc_auc_score(X_actual, Y_predic))

print('LOGLOSS Value is',log_loss(X_actual, Y_predic))

```
In [33]: import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin'
```

```
In [34]: from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
X_actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
Y_predic = [1, 0, 1, 1, 1, 0, 1, 1, 0, 0]
results = confusion_matrix(X_actual, Y_predic)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))
print ('Classification Report : ')
print (classification_report(X_actual, Y_predic))
print('AUC-ROC:',roc_auc_score(X_actual, Y_predic))
print('LOGLOSS Value is',log_loss(X_actual, Y_predic))
```

```
Confusion Matrix :
[[3 3]
 [1 3]]
Accuracy Score is 0.6
Classification Report :
      precision    recall  f1-score   support

     0       0.75      0.50      0.60         6
     1       0.50      0.75      0.60         4

   accuracy          0.62
  macro avg          0.62
 weighted avg          0.65

AUC-ROC: 0.625
```