ASSIGNMENT 4

Problem Statement: Abalone Age Prediction

Description:- Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict age. Further information, such as weather patterns and location (hence food availability) may be required to solve the problem.

Attribute Information:

Given is the attribute name, attribute type, measurement unit, and a brief description. The number of rings is the value to predict: either as a continuous value or as a classification problem.

Name / Data Type / Measurement Unit / Description

1- Sex / nominal / -- / M, F, and I (infant)

2- Length / continuous / mm / Longest shell measurement

3- Diameter / continuous / mm / perpendicular to length

4- Height / continuous / mm / with meat in shell

5- Whole weight / continuous / grams / whole abalone

6- Shucked weight / continuous / grams / weight of meat

7- Viscera weight / continuous / grams / gut weight (after bleeding)

8- Shell weight / continuous / grams / after being dried

9- Rings / integer / -- / +1.5 gives the age in years

Building a Regression Model

1. Download the dataset: Dataset
2. Load the dataset into the tool.

```
[6]  df=pd.read_csv('../abalone.csv')
```

```
[7] 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
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

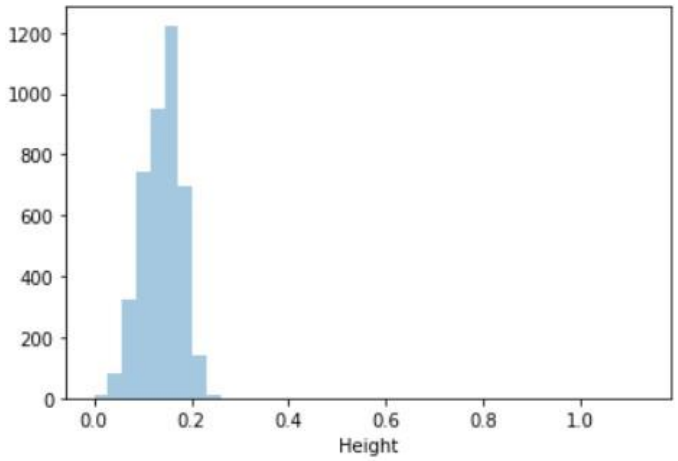
3. Perform Below Visualizations.

- **Univariate Analysis**

```
import numpy as np
import pandas as pd
#For plotting
from matplotlib import pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
[9] sns.distplot(df.Height.dropna(), kde=False, bins = 39)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarni
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f02be9339d0>
```



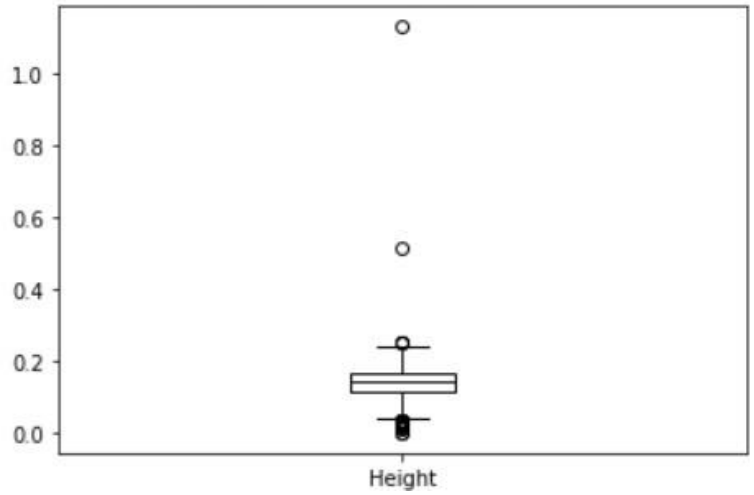
✓ 0s completed at 11:11 PM

Box plot:

```
[16] import matplotlib.pyplot as plt

     df.boxplot(column=['Height'], grid=False, color='black')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f02be793650>
```

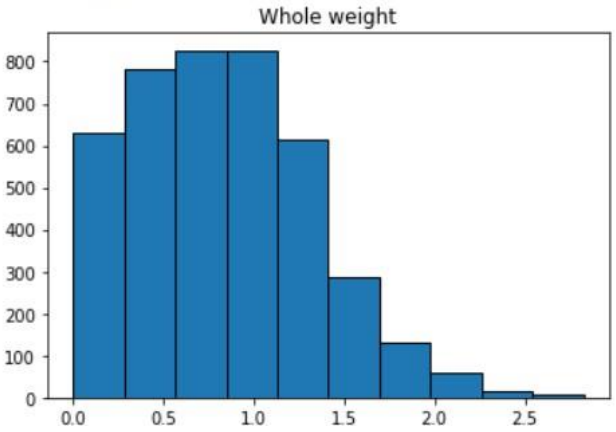


Histogram:

```
[17] import matplotlib.pyplot as plt

     df.hist(column='Whole weight', grid=False, edgecolor='black')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f02be1700d0>]],
      dtype=object)
```



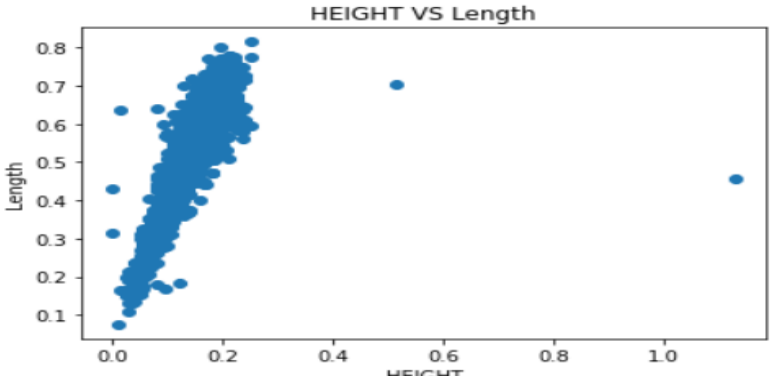
· Bi-Variate Analysis

1. Scatterplots

```
[21] import matplotlib.pyplot as plt

     #create scatterplot of hours vs. score
     plt.scatter(df.Height, df.Length)
     plt.title('HEIGHT VS Length')
     plt.xlabel('HEIGHT')
     plt.ylabel('Length')
```

```
Text(0, 0.5, 'Length')
```



2. Correlation Coefficients

A Pearson Correlation Coefficient is a way to quantify the linear relationship between two variables.

We can use the corr() function in pandas to create a correlation matrix:

```
#create correlation matrix
df.corr()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
Length	1.000000	0.986812	0.827554	0.925261	0.897914	0.903018	0.897706	0.556720
Diameter	0.986812	1.000000	0.833684	0.925452	0.893162	0.899724	0.905330	0.574660
Height	0.827554	0.833684	1.000000	0.819221	0.774972	0.798319	0.817338	0.557467
Whole weight	0.925261	0.925452	0.819221	1.000000	0.969405	0.966375	0.955355	0.540390
Shucked weight	0.897914	0.893162	0.774972	0.969405	1.000000	0.931961	0.882617	0.420884
Viscera weight	0.903018	0.899724	0.798319	0.966375	0.931961	1.000000	0.907656	0.503819
Shell weight	0.897706	0.905330	0.817338	0.955355	0.882617	0.907656	1.000000	0.627574

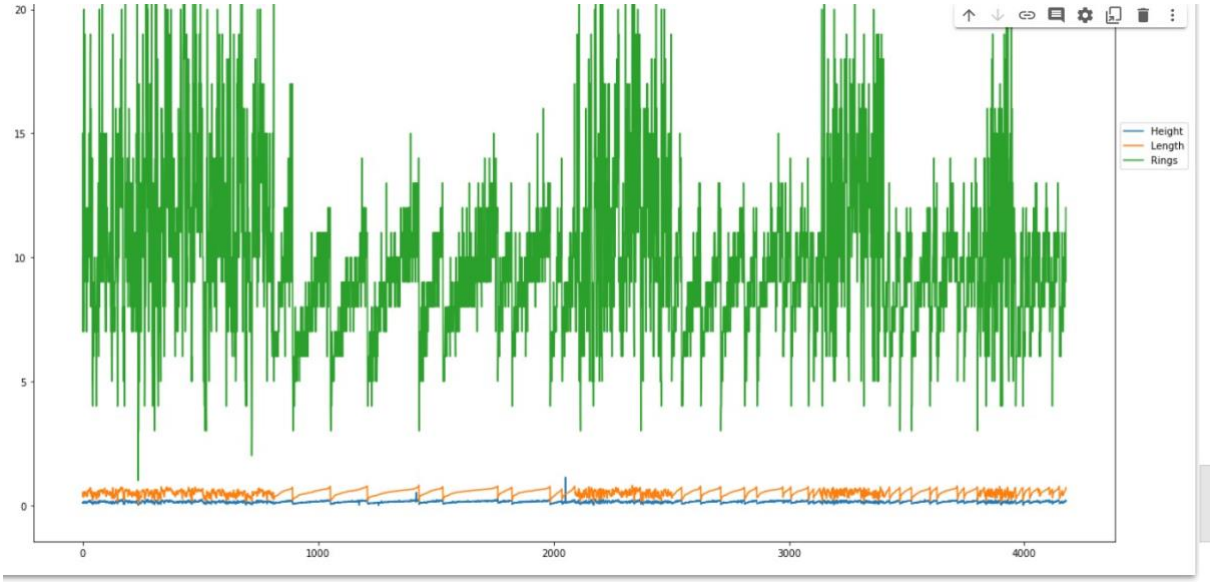
✓ 0s completed at 11:20 PM

· **Multi-Variate Analysis**

Profile Plot

Profile plot, used to shows the variation in each of the variables, by plotting the value of each of the variables for each of the samples.

```
ax = df[["Height","Length","Rings"]].plot(figsize=(20,15))
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```



4. Perform descriptive statistics on the dataset.

Mean median and standard deviation.

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

```
0.041827056607257274
```

```
[15] #calculate median
     df['Height'].median()
```

```
0.14
```

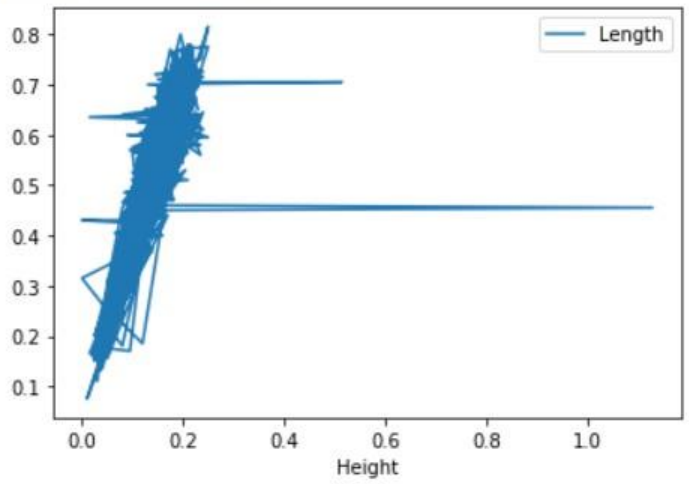
```
[14] #calculate standard deviation
     df['Height'].std()
```

```
0.041827056607257274
```

IQR(difference between 75% and 25% quartile)

```
df.plot(x="Height", y="Length")
IQR=df["Length"].quantile(0.75)-df["Length"].quantile(0.25)
IQR
```

```
0.16499999999999998
```



5. Check for Missing values and deal with them.

```
df.isnull()
```

	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

No missing values.

6. Find the outliers and replace them outliers

Outliers Identification

There are different ways and methods of identifying outliers, but we are only going to use some of the most popular techniques:

- Skewness

Skewness

the skewness value should be within the range of -1 to 1 for a normal distribution, any major changes from this value may indicate the presence of outliers.

```
print('skewness value of ring: ',df['Rings'].skew())
print('skewness value of height: ',df['Height'].skew())
```

```
skewness value of ring:  1.114101898355677
skewness value of height:  3.1288173790659615
```

value of 3.12 shows the variable has been rightly skewed,

indicating the presence of outliers.

Outliers Treatment

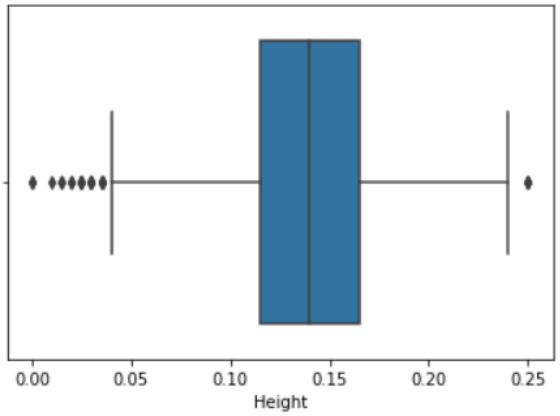
- Trimming.

in this method, we removed and completely drop all the outliers, the line of code below creates an index for all data points and drop the index values.

```
[36] Q1 = df['Height'].quantile(0.10)
     Q3 = df['Height'].quantile(0.90)
     IQR = Q3 - Q1
     whisker_width = 1.5
     lower_whisker = Q1 - (whisker_width*IQR)
     upper_whisker = Q3 + (whisker_width*IQR)
     index=df['Height'][(df['Height']>upper_whisker)|(df['Height']<lower_whisker)].index
     df.drop(index,inplace=True)
```

```
sns.boxplot(df['Height'],data=df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f02bc5345d0>
```



✓ 0s completed at 11:44 PM

7. Check for Categorical columns and perform encoding.

There are no categorial data:

8. Split the data into dependent and independent variables

Splitting the Dataset into the Independent Feature Matrix:

```
X = df.iloc[:, :-1].values
print(X)
```

```
[['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]]
```

Extracting the Dataset to Get the Dependent Vector

```
Y = df.iloc[:, -1].values
print(Y)
```

```
[15  7  9 ...  9 10 12]
```

. 9. Scale the independent variables

It is a pre-processing step. This technique used to **normalize** the range of independent variables. Variables that are used to determine the target variable are known as features.

1. **MIN-MAX SCALING**

   In min-max scaling or min-man normalization, we re-scale the data to a range of [0,1] or [-1,1].

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

2. STANDARDIZATION

In this, we scale the features in such a way that the distribution has mean=0 and variance=1.

$$z = \frac{x - \mu}{\sigma}$$

```
from sklearn import preprocessing

#extracting values which we want to scale
x = df.iloc[:, 1:4].values
print ("\n ORIGIONAL VALUES: \n\n", x)
#MIN-MAX SCALER
min_max_scaler = preprocessing.MinMaxScaler(feature_range =(0, 1))
new_x= min_max_scaler.fit_transform(x)
print ("\n VALUES AFTER MIN MAX SCALING: \n\n", new_x)
Standardisation = preprocessing.StandardScaler()
new_x= Standardisation.fit_transform(x)
print ("\n\n VALUES AFTER STANDARDIZATION : \n\n", new_x)
```

```
ORIGIONAL VALUES:

[[0.455 0.365 0.095]
[0.35  0.265 0.09 ]
[0.53  0.42  0.135]
...
[0.6   0.475 0.205]
[0.625 0.485 0.15 ]
[0.71  0.555 0.195]]

VALUES AFTER MIN MAX SCALING:

[[0.51351351 0.5210084  0.38      ]
[0.37162162 0.35294118 0.36      ]
[0.61486486 0.61344538 0.54      ]
...
[0.70945946 0.70588235 0.82      ]
[0.74324324 0.72268908 0.6       ]
```

10. Split the data into training and testing

```
from sklearn.model_selection import train_test_split

training_data, testing_data = train_test_split(df, test_size=0.2, random_state=25)

print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")
```

```
No. of training examples: 3340
No. of testing examples: 835
```

11. Build the Model

USING LINEAR REGRESSION :

```
from sklearn import preprocessing
y = df['Rings'].values.reshape(-1, 1)
X = df['Height'].values.reshape(-1, 1)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=1/3,random_state=0)
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

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

```
LinearRegression()
```

12. Train the Model

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

```
LinearRegression()
```

```
[63] y_pred = regressor.predict(X_test)
     y_pred
```

```
array([[ 8.95492379],
       [10.00093634],
       [ 9.47793007],
       ...,
       [14.44648972],
       [ 8.17041437],
       [ 9.73943321]])
```

```
[64] y_test
```

```
array([[ 7],
       [10],
       [11],
       ...,
       [15],
       [ 9],
       [11]])
```

```
[65] #plot for the TRAIN

     plt.scatter(X_train, y_train, color='red') # plotting the observation line

     plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line

     plt.title("Salary vs Experience (Training set)") # stating the title of the graph

     plt.xlabel("Years of experience") # adding the name of x-axis
     plt.ylabel("Salaries") # adding the name of y-axis
     plt.show() # specifies end of graph
```

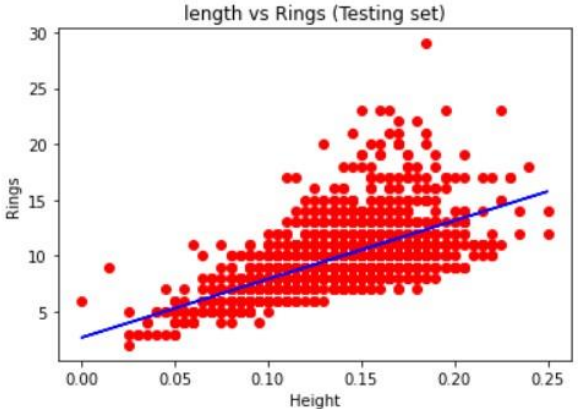
13. Test the Model

```
#plot for the TEST

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line

plt.title("length vs Rings (Testing set)")

plt.xlabel("Height")
plt.ylabel("Rings")
plt.show()
```



14. Measure the performance using Metrics

```
[67] from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
[68] mae = mean_absolute_error(y_test, y_pred)
     mse = mean_squared_error(y_test, y_pred)
     rmse = np.sqrt(mse)
```

```
print(f'Mean absolute error: {mae:.2f}')
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}')
```

```
Mean absolute error: 1.85
Mean squared error: 6.34
Root mean squared error: 2.52
```

.