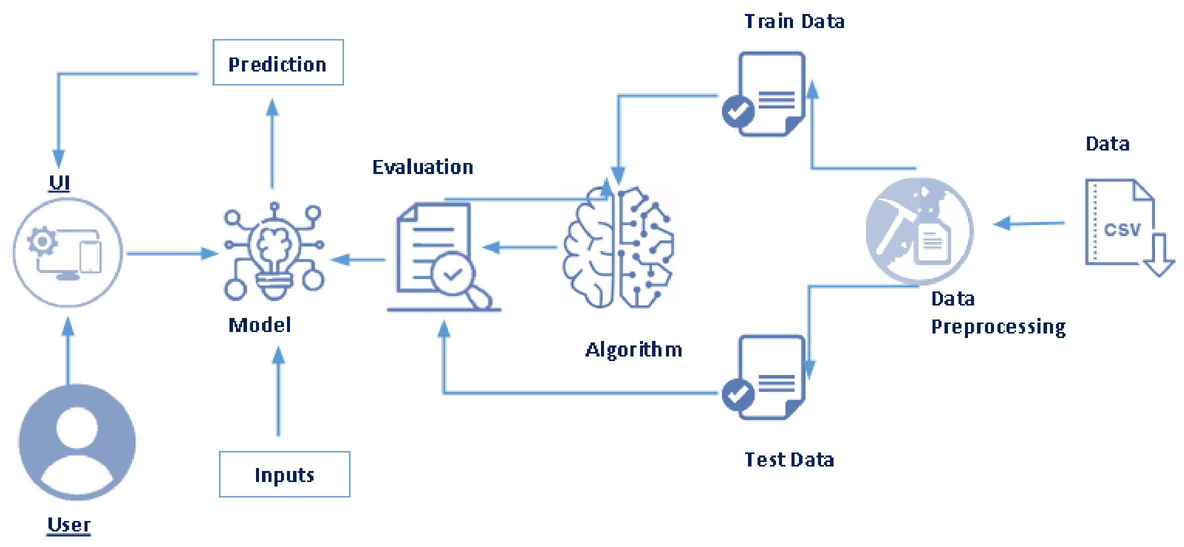
PREDICTING THE AGE OF ABALONE

Project Overview:

Abalone is a shellfish considered a delicacy in many parts of the world. An excellent source of iron and pantothenic acid, and a nutritious food resource and farming in Australia, America and East Asia. 100 grams of abalone yields more than 20% recommended daily intake of these nutrients. The economic value of abalone is positively correlated with its age. Therefore, to detect the age of abalone accurately is important for both farmers and customers to determine its price.
However, the current technology to decide the age is quite costly and inefficient. Farmers usually cut the shells and count the rings through microscopes to estimate the abalone's age. Telling the age of abalone is therefore difficult mainly because their size depends not only on their age, but on the availability of food as well. Moreover, abalone sometimes form the so-called 'stunted' populations which have their growth characteristics very different from other abalone populations This complex method increases the cost and limits its popularity. Our goal is to find out the best indicators to forecast the rings, then the age of abalone.

Architecture:



Prerequisites

To complete this project, you must require following software's, concepts and packages

● Anaconda navigator and spider:

o Refer the link below to download anaconda navigator

o Link : https://youtu.be/1ra4zH2G4o0

● Python packages:

o Open anaconda prompt as administrator

o Type "pip install numpy" and click enter.

o Type "pip install pandas" and click enter.

o Type "pip install matplotlib" and click enter.

o Type "pip install pickle-mixin" and click enter.

o Type "pip install seaborn" and click enter.

o Type "pip install Flask" and click enter.

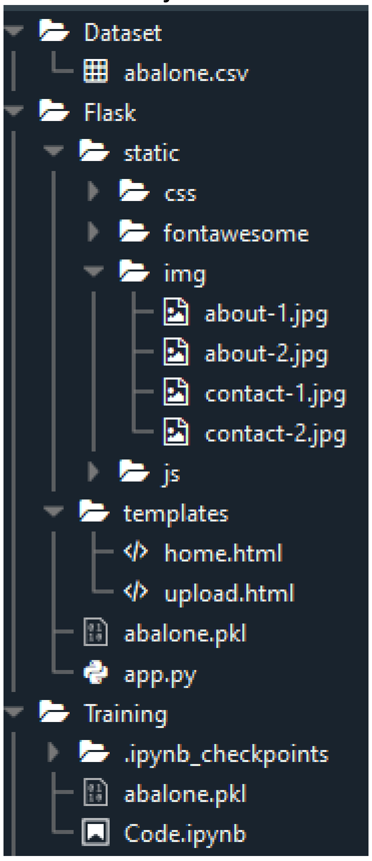
Project Workflow

- User interacts with the UI (User Interface) to upload the input features.
- Uploaded features/input is analyzed by the model which is integrated.
- Once a model analyses the uploaded inputs, the prediction is showcased on the UI.

To accomplish this, we have to complete all the activities and tasks listed below

Project Structure

Create a Project folder which contains files as shown below



```
Dataset
  abalone.csv
Flask
  static
    css
    fontawesome
    img
      about-1.jpg
      about-2.jpg
      contact-1.jpg
      contact-2.jpg
    js
  templates
    home.html
    upload.html
  abalone.pkl
  app.py
Training
  .ipynb_checkpoints
  abalone.pkl
  Code.ipynb
```

▯ We have three folders dataset, Flask and Training.
▯ Dataset has dataset abalone.csv.
▯ A python file called app.py for server side scipting.
▯ We need the model which is saved and the saved model in this content is abalone.pkl.
▯ Templates folder which contains home.html and upload.html files.
▯ Training folder has Code.ipynb where the model is created and saved.
▯ Static folder which contains css(styling), fontawesome(styling), img(images), js(Java script) folders to enhance the features of the web page

Dataset Collection

ML depends heavily on data, without data, it is impossible for an "AI" to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions

Predicting the age of abalone from physical measurements..

Predicting the age of abalone from physical measurements..

https://www.kaggle.com/rodolfomendes/abalone-dataset

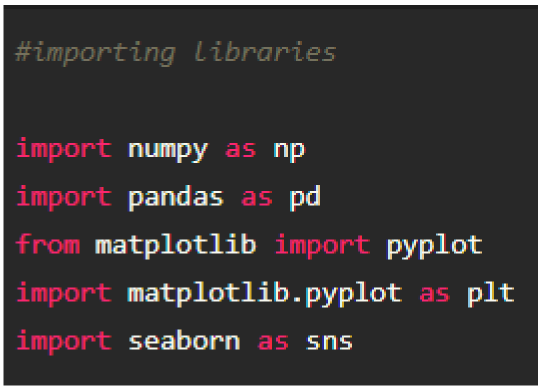
Data Pre-processing

Data Pre-processing includes the following main tasks

1. Import the Libraries.
2. Reading the dataset.
3. Exploratory Data Analysis
4. Checking for Null Values.
5. Data Visualization.
6. Label Encoding
7. Splitting the Dataset into Dependent and Independent variables.
8. Splitting Data into Train and Test

Importing the Libraries

The first step is usually importing the libraries that will be needed in the program.



```
#importing libraries

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

Pandas: It is a python library mainly used for data manipulation.

NumPy: This python library is used for numerical analysis.

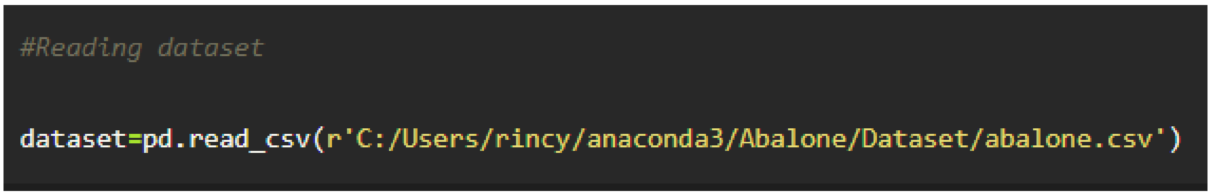
Matplotlib and Seaborn: Both are the data visualization library used for plotting graphs which will help us to understand the data.

Accuracy score: used in classification type problems and for finding accuracy.

Pickle: to serialize your machine learning algorithms and save the serialized format to a file

Reading the Dataset

- You might have your data in .csv files, .excel files or .**tsv** files or something else. But the goal is the same in all cases. If you want to analyse that data using pandas, the first step will be to read it into a data structure that's compatible with pandas.
- Let's load a .csv data file into pandas. There is a function for it, called **read_csv().**We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- names on Windows tend to have backslashes in them. But we want them to mean actual backslashes, not special characters.



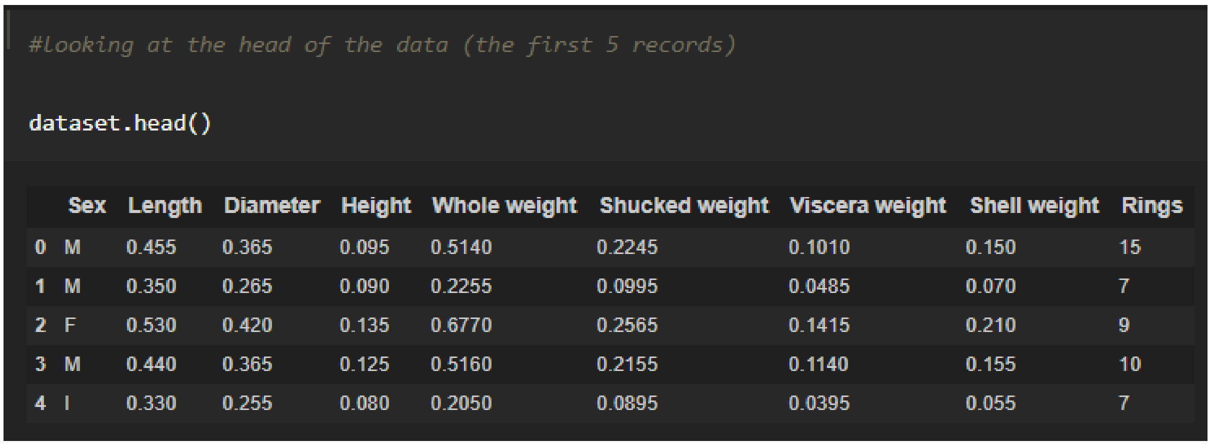
```
#Reading dataset

dataset=pd.read_csv(r'C:/Users/rincy/anaconda3/Abalone/Dataset/abalone.csv')
```

Exploratory Data Analysis

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods and used for determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

head() :To check the first five rows of the dataset, we have a function called **head( ).**



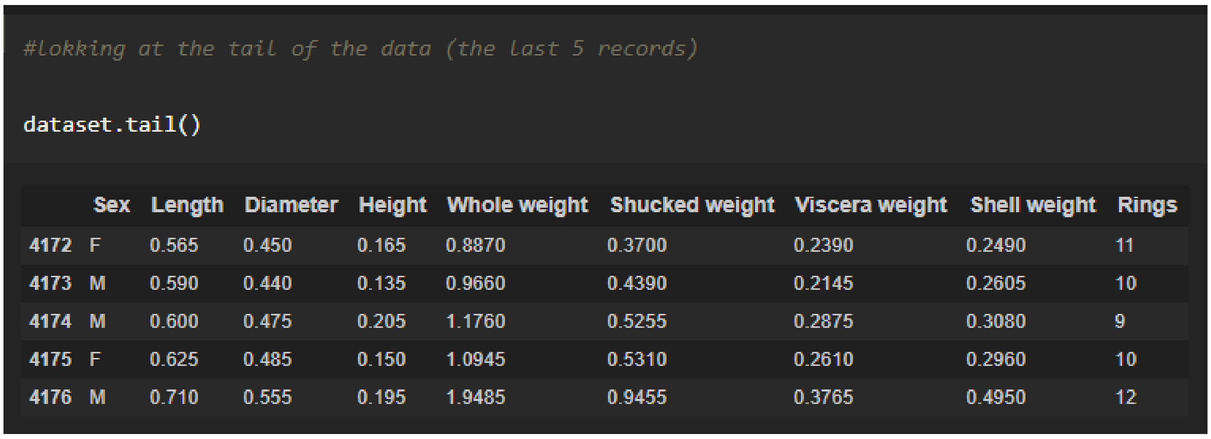
```
#looking at the head of the data (the first 5 records)

dataset.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

- This head () function returns the first 5 rows for the object based on position. It
  is useful for quickly testing if your object has the right type of data in it.

Tail(): To check the last five rows of the dataset, we have a function called **tail().**

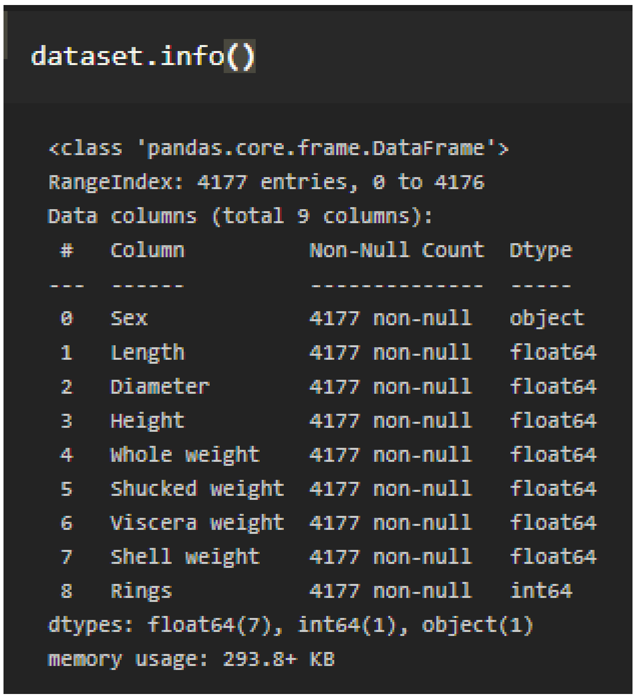


```
#lokking at the tail of the data (the last 5 records)

dataset.tail()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
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

- **Understanding Data Type and Summary of features**
  - How the information is stored in a DataFrame or Python object affects what we can do with it and the outputs of calculations as well. There are two main types of data those are numeric and text data types.
  - Numeric data types include integers and floats.
  - Text data type is known as Strings in Python, or Objects in Pandas. Strings can contain numbers and / or characters.
  - For example, a string might be a word, a sentence, or several sentences.
- Will see how our dataset is, by using the info() method.

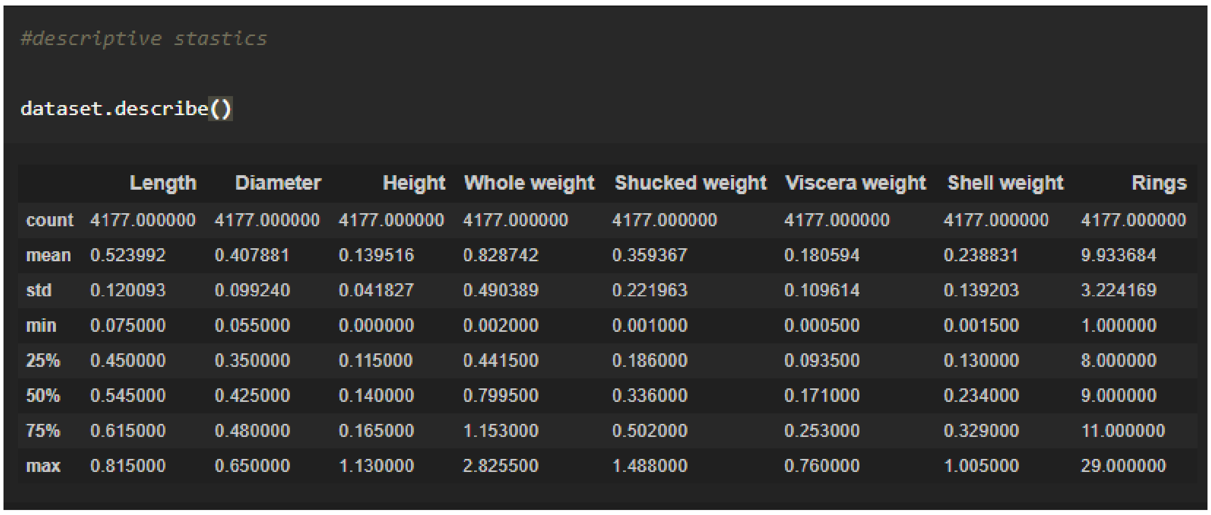


```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

- We notice that there are both numerical and categorical data present in the abalone dataset, but it is not necessary that all the continuous data which we are seeing has to be continuous in nature. There may be a case that some categorical data is in the form of numbers but when we perform info() operation we will get numerical output. So, we need to take care of those type of data also.

describe(): functions are used to compute values like count, mean, standard deviation and IQR(Inter Quantile Ranges) and give a summary of numeric type data.



```
#descriptive stastics

dataset.describe()
```

	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

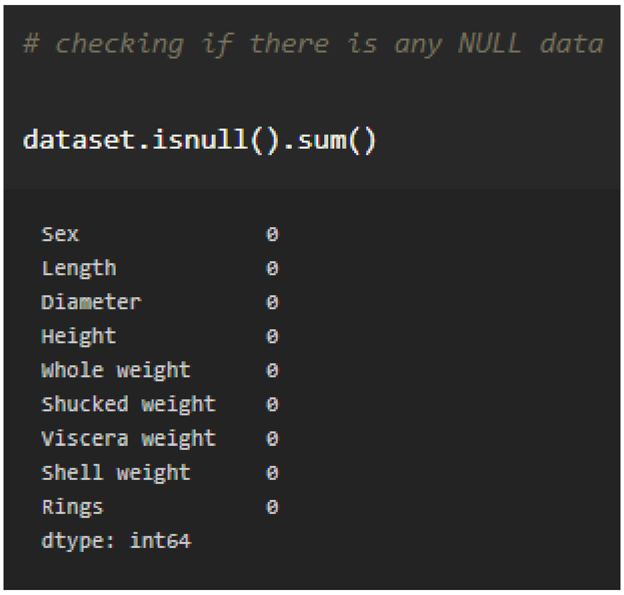
Checking for null values

1. After loading it is important to check the complete information of data as it can indication many of the hidden information such as null values in a column or a row

2.Check whether any null values are there or not. if it is present then following can be done,

a.Imputing data using Imputation method in sklearn b.Filling NaN values with mean, median and mode using fillna() method.

We will be using isnull().sum() method to see which column has missing values by counting the total sum of null values in each column.



```
# checking if there is any NULL data

dataset.isnull().sum()
```

```
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked weight    0
Viscera weight    0
Shell weight      0
Rings             0
dtype: int64
```

As our data is not having any null values, we can proceed further.

Data into Independent & Dependent Variable

- Data visualization is where a given data set is presented in a graphical format. It helps the detection of patterns, trends and correlations that might go undetected in text-based data.
- Understanding your data and the relationship present within it is just as important as any algorithm used to train your machine learning model. In fact, even the most sophisticated machine learning models will perform poorly on data that wasn't visualized and understood properly.
- To visualize the dataset we need libraries called Matplotlib and Seaborn.
- The Matplotlib library is a Python 2D plotting library which allows you to generate plots, scatter plots, histograms, bar charts etc.

Univariate Analysis

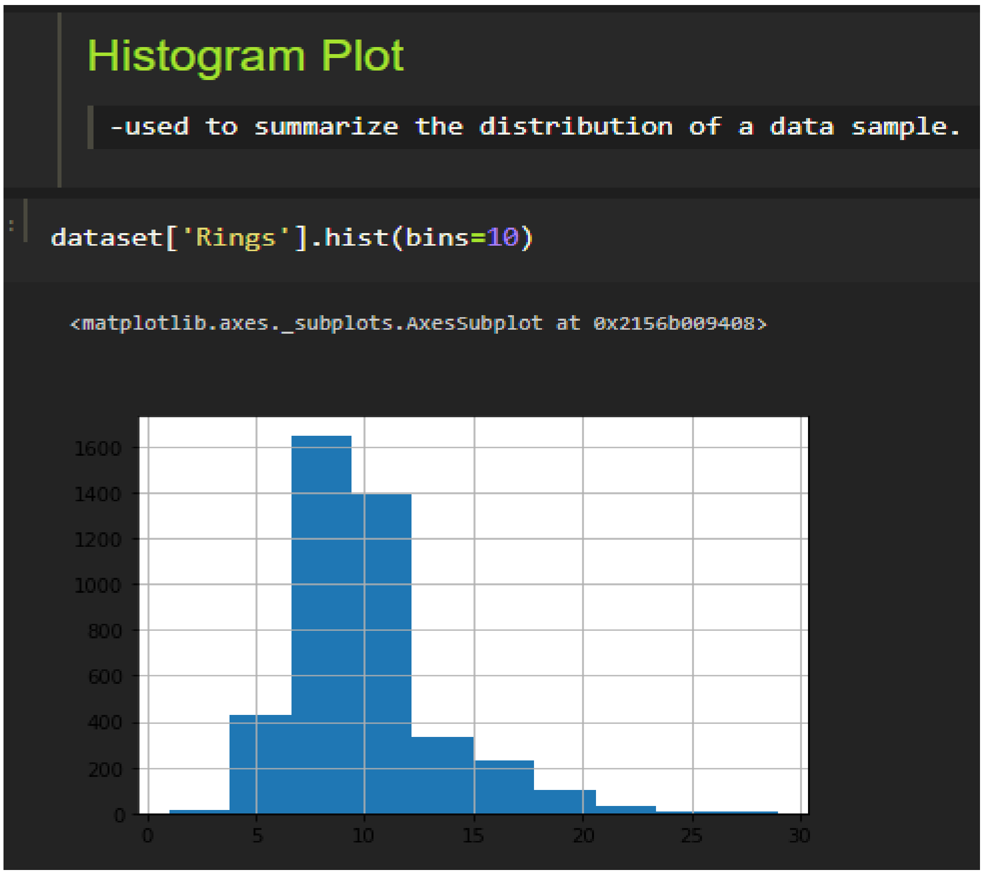
Univariate analysis is the simplest form of data analysis where the data being analyzed contains only one variable.

Bivariate Analysis

It involves the analysis of two variables (often denoted as X, Y), for the purpose of determining the empirical relationship between them.

Let's visualize our data using Matplotlib and searborn library.

1.Plotting histogram to summarize "Rings" data.



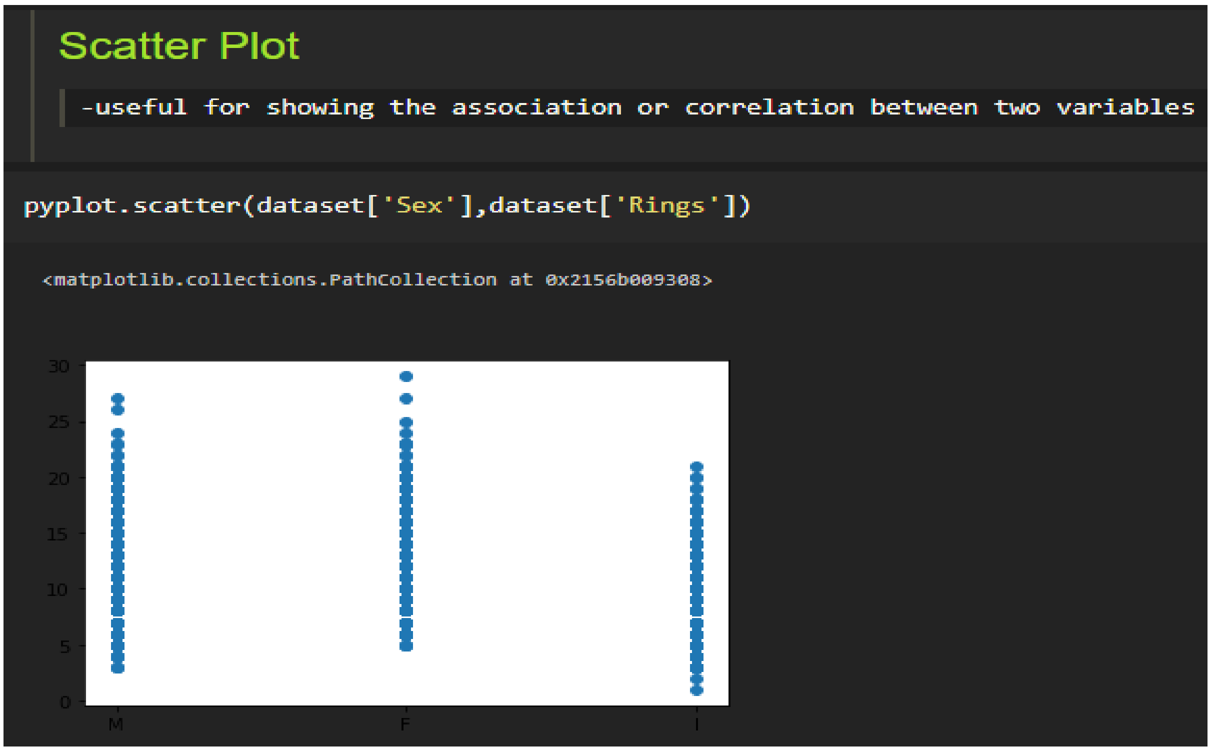
Histogram Plot

-used to summarize the distribution of a data sample.

```
dataset['Rings'].hist(bins=10)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2156b009408>
```

2. Using scatter plot to present the relationship between Sex data and Rings data.



Scatter Plot

-useful for showing the association or correlation between two variables

```
pyplot.scatter(dataset['Sex'],dataset['Rings'])
```

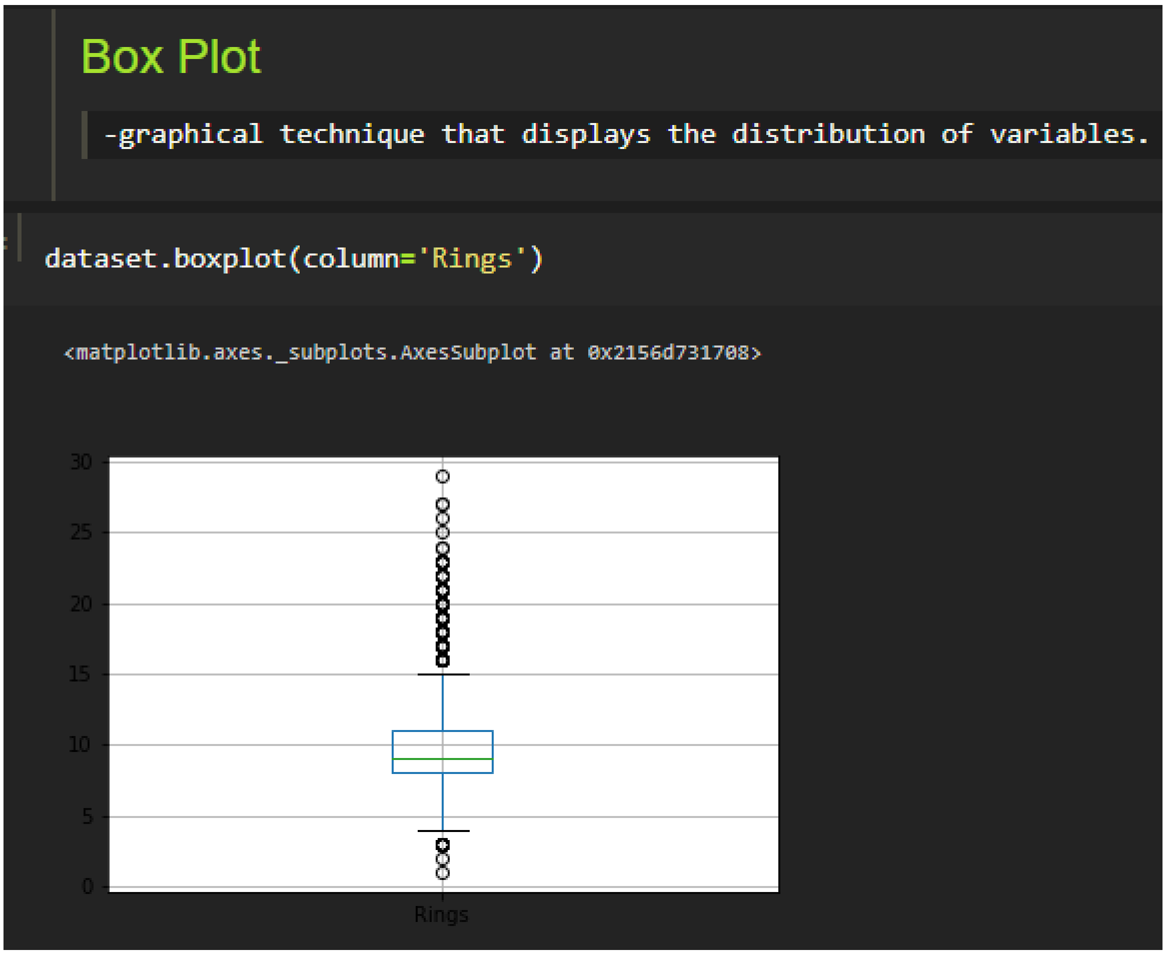
```
<matplotlib.collections.PathCollection at 0x2156b009308>
```

3.Outliers Detection using boxplot

Outliers are observations in a dataset that don't fit in some way or you can say those values are of no use and also may effect our results.

So with the help of boxplot we can visualize and check whether the data contains any outliers or not.

Box plot for Rings column



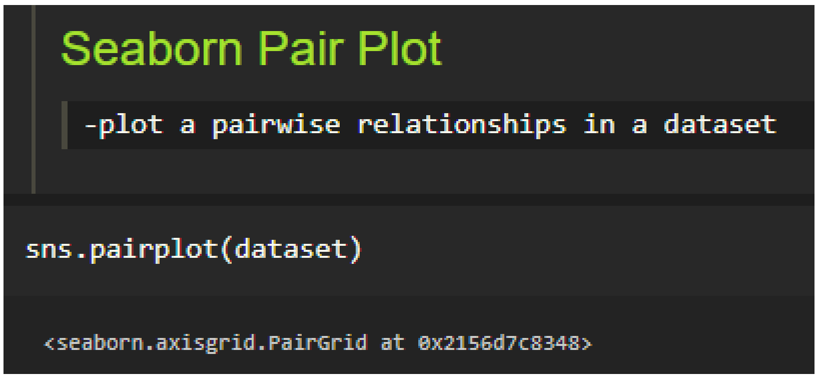
Box Plot

> -graphical technique that displays the distribution of variables.

```
dataset.boxplot(column='Rings')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2156d731708>
```

4. Drawing relationship between each column of dataset to every column using Seaborn Pair Plot.

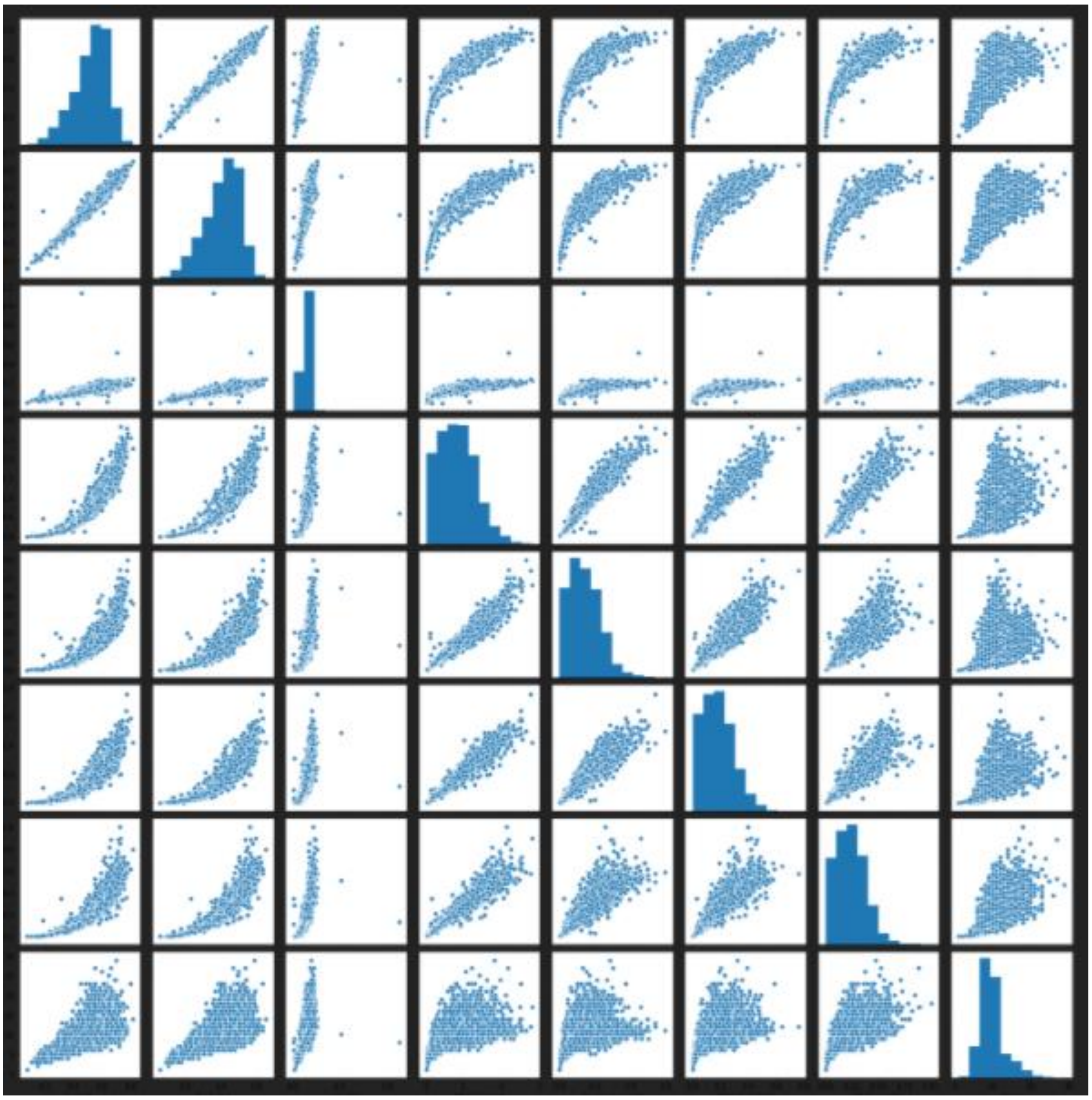


Seaborn Pair Plot

> -plot a pairwise relationships in a dataset

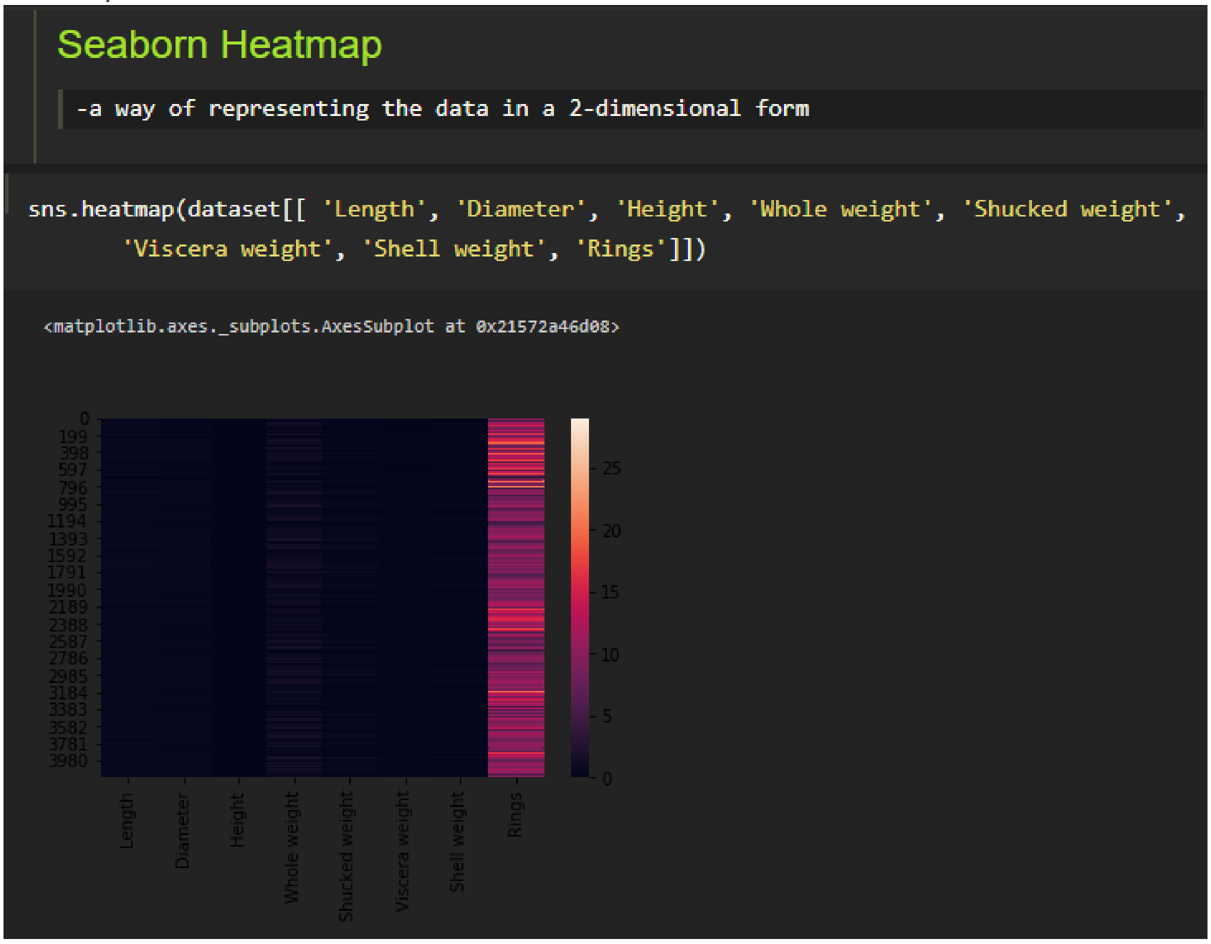
```
sns.pairplot(dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x2156d7c8348>
```



5. Seaborn Heatmap is a graphical representation of 2D (two dimensional) data. Each data value represents in a matrix and it has a special color. The color of the matrix is dependent on value. Normally, low-value show in low-intensity color and high-value show in high-

intensity color format.



Seaborn Heatmap

```
-a way of representing the data in a 2-dimensional form
```

```
sns.heatmap(dataset[[ 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'Rings']])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21572a46d08>
```

6. Label Encoding

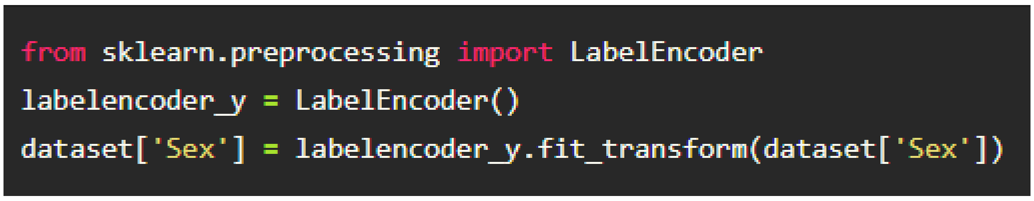
Typically, any structured dataset includes multiple columns with combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too .We need to convert each text category to numbers in order for the machine to process those using mathematical equations.

How should we handle categorical variables? There are Multiple way to handle, but will see one of it is Label Encoding.

- **Label Encoding** is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

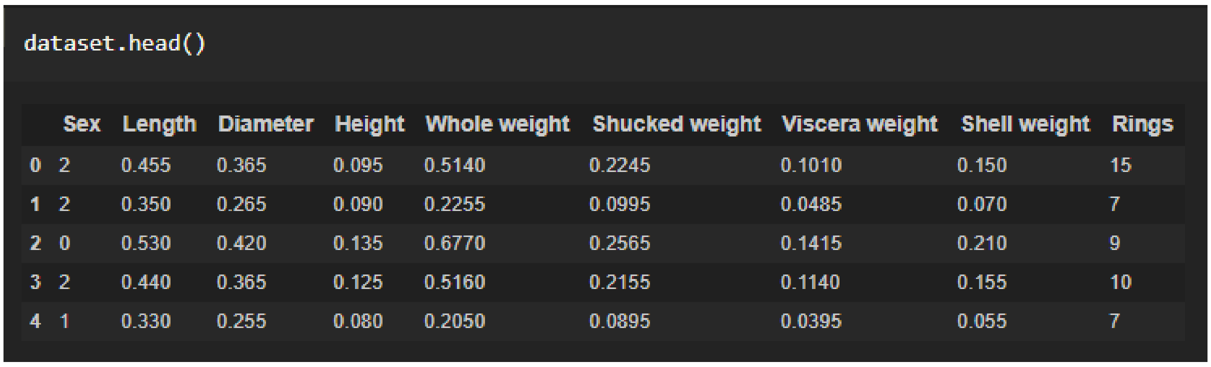
Let's see how to implement label encoding in Python using the scikit-learn library.

As we have to convert only the text class category columns, we first select it then we will implement Label Encoding to it.



```
from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
dataset['Sex'] = labelencoder_y.fit_transform(dataset['Sex'])
```

We notice the output of the above code, after performing label encoding alphabetical classes is converted to numeric.



```
dataset.head()
```

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

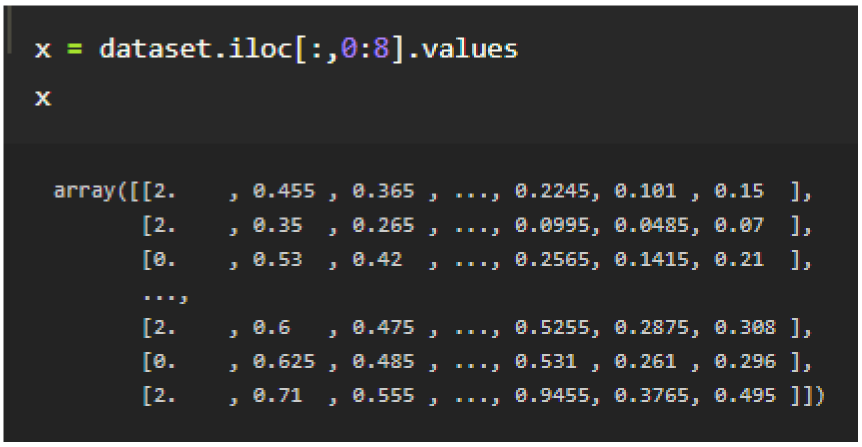
7: Splitting the Dataset into Dependent and Independent variable.

- In machine learning, the concept of dependent variable (y) and independent variables(x) is important to understand. Here, Dependent variable is nothing but output in dataset and independent variable is all inputs in the dataset.
- With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.

To read the columns, we will use **iloc** of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

- Let's split our dataset into independent and dependent variables.
  1. The independent variable in the dataset would be considered as 'x' and the 'Sex', 'Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell weight' columns would be considered as independent variable.
  2. The dependent variable in the dataset would be considered as 'y' and the 'Rings' column is considered as dependent variable.
  Now we will split the data of independent variables,

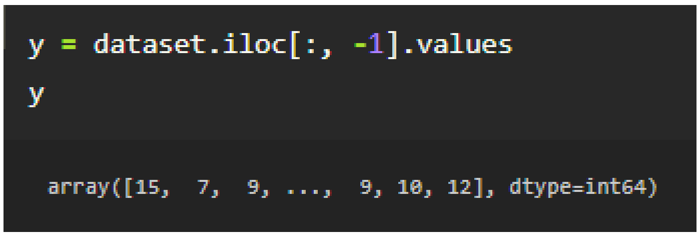
Independent_variables:



```
x = dataset.iloc[:,0:8].values
x
```

```
array([[2.    , 0.455 , 0.365 , ..., 0.2245, 0.101 , 0.15  ],
       [2.    , 0.35  , 0.265 , ..., 0.0995, 0.0485, 0.07  ],
       [0.    , 0.53  , 0.42  , ..., 0.2565, 0.1415, 0.21  ],
       ...,
       [2.    , 0.6   , 0.475 , ..., 0.5255, 0.2875, 0.308 ],
       [0.    , 0.625 , 0.485 , ..., 0.531 , 0.261 , 0.296 ],
       [2.    , 0.71  , 0.555 , ..., 0.9455, 0.3765, 0.495 ]])
```

Dependent_variables:



```
y = dataset.iloc[:, -1].values
y
```

```
array([15,  7,  9, ...,  9, 10, 12], dtype=int64)
```

In the above code we are creating DataFrame of the independent variable **x** with our selected columns and for dependent variable **y** we are only taking the **Rings** column.

Split the dataset into Train set and Test set

- When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.
- But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you split the data in a random manner. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is,'train_test_split.' Using this we can easily split the dataset into the training and the testing datasets in various proportions.
- The train-test split is a technique for evaluating the performance of a machine learning algorithm.
- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.
- In general you can allocate 80% of the dataset to training set and the remaining 20% to test set.We will create 4 sets— x_train (training part of the matrix of features), x_test (test part of the matrix of features), y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.
- There are a few other parameters that we need to understand before we use the class:
- **test_size** — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset
- **train_size** — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
- **random_state** — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the Random_state class, which will become the number generator. If you don't pass anything, the Random_state instance used by np.random will be used instead.
- Now split our dataset into train set and test using train_test_split class from scikit learn library.

```
# train test split

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
```

Model Building

Predictive modeling is a mathematical approach to create a statistical model to forecast future behavior based on input test data.

Steps involved in predictive modeling:

Algorithm Selection:

When we have the structured dataset, and we want to estimate the continuous or categorical outcome then we use supervised machine learning methodologies like regression and classification techniques. When we have unstructured data and want to predict the clusters of items to which a particular input test sample belongs, we use unsupervised algorithms. An actual data scientist applies multiple algorithms to get a more accurate model.

Train Model:

After assigning the algorithm and getting the data handy, we train our model using the input data applying the preferred algorithm. It is an action to determine the correspondence between independent variables, and the prediction targets.

Model Prediction:

We make predictions by giving the input test data to the trained model. We measure the accuracy by using a cross-validation strategy or ROC curve which performs well to derive model output for test data.

Train and Test the Model using Decision Tree & Random Forest Regressor.

There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have may be Classification algorithms are Regression algorithms.

Example:

1. Decision Tree Regression / Classification.

2. Random Forest Regression / Classification.

You will need to train the datasets to run smoothly and see an incremental improvement in the prediction rate.

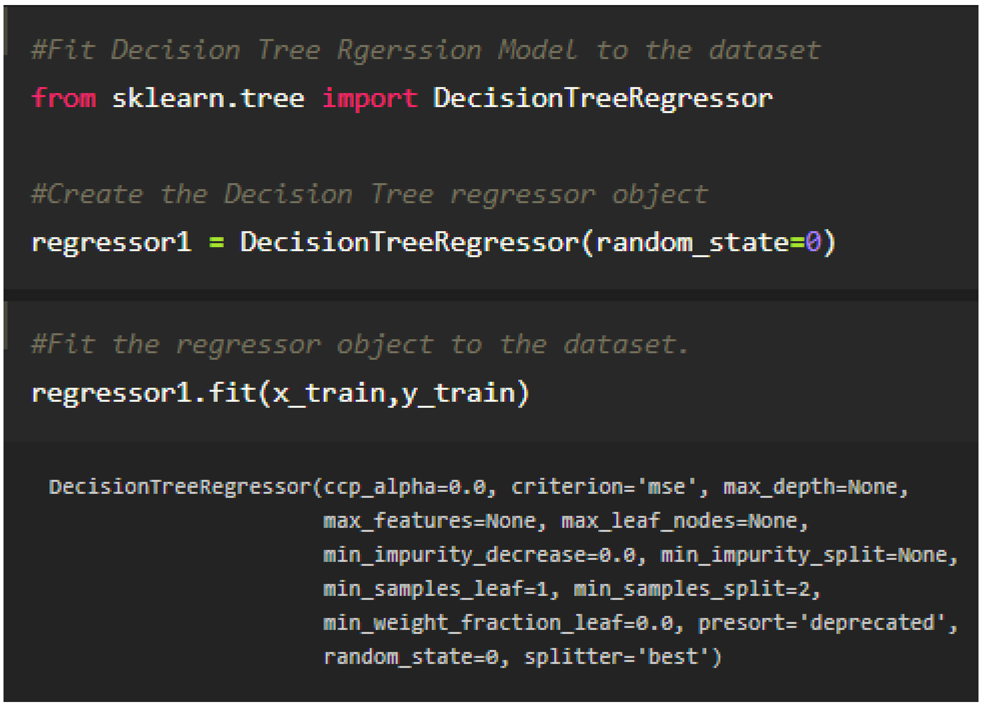
Now we apply regression algorithms on our dataset.

Decision Tree Regression / Classification: A decision tree is a supervised machine learning model used to predict a target by learning decision rules from features.

Random Forest Regression / Classification: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean/average prediction of the individual trees.

Build the model

We're going to use x_train and y_train obtained above in train_test_split section to train our regression model. We're using the fit method and passing the parameters as shown below.

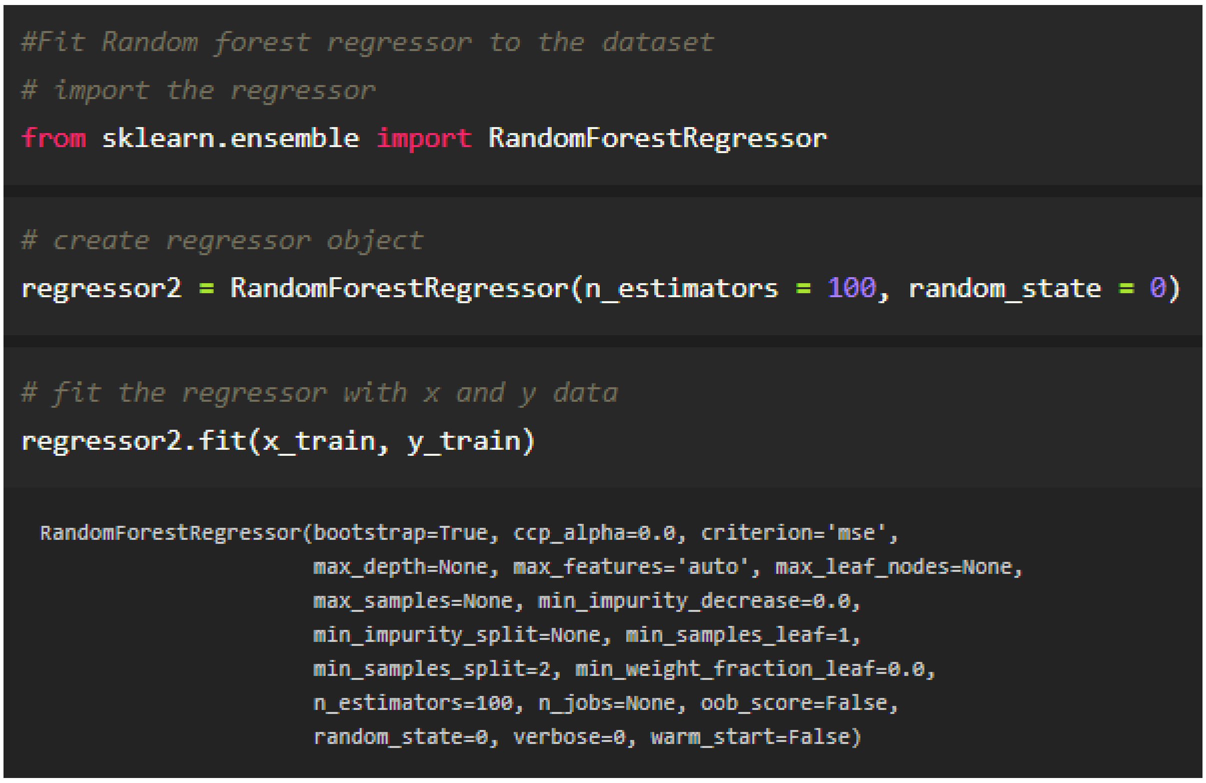


```
#Fit Decision Tree Rgerssion Model to the dataset
from sklearn.tree import DecisionTreeRegressor

#Create the Decision Tree regressor object
regressor1 = DecisionTreeRegressor(random_state=0)
```

```
#Fit the regressor object to the dataset.
regressor1.fit(x_train,y_train)
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
```



```
#Fit Random forest regressor to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor
```

```
# create regressor object
regressor2 = RandomForestRegressor(n_estimators = 100, random_state = 0)
```

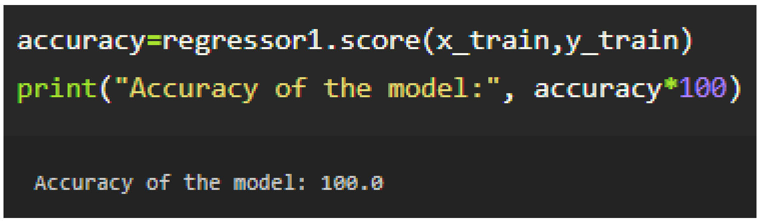
```
# fit the regressor with x and y data
regressor2.fit(x_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=0, verbose=0, warm_start=False)
```

Finally, we need to check to see how well our model is performing on the test data.

Evaluation Metrics:

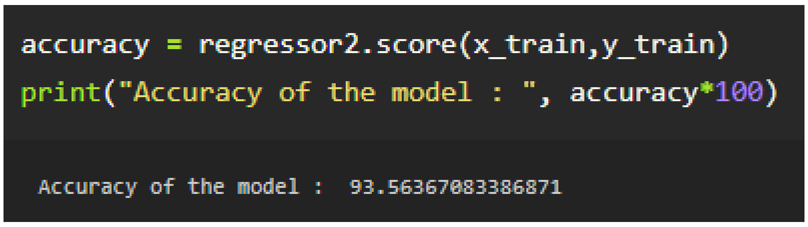
Accuracy score of Decision Tree Regressor is



```
accuracy=regressor1.score(x_train,y_train)
print("Accuracy of the model:", accuracy*100)
```

```
Accuracy of the model: 100.0
```

Accuracy score of Random Forest Regressor is



```
accuracy = regressor2.score(x_train,y_train)
print("Accuracy of the model : ", accuracy*100)
```

```
Accuracy of the model :  93.56367083386871
```

As we can see that the accuracy of the Decision Tree Regressor model is greater than the accuracy of the Random Forest Regressor model, we are proceeding with the Decision Tree Regressor model.

Save the Model

After building the model we have to save the model.

Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later on, this character stream can then be retrieved and de-serialized back to a Python object.

This is done by the below code

```
import pickle
pickle.dump(regressor1, open('abalone.pkl','wb'))
```

Here, regressor1 is our Decision Tree regression class, saving as abalone.pkl file. Wb is the write binary in bytes.

Predicting the output using the model

Let us predict the age of Abalone by giving input to the model build.

First, load the saved model. The input is given to a model to predict the output. Age of abalone is Output+1.5 years.

```
model=pickle.load(open('abalone.pkl','rb'))
input=[[2, 0.455 , 0.365 , 0.095 , 0.514 , 0.2254, 0.101 , 0.15]]
pred=model.predict(input)
pred
```

```
array([15.])
```

In this case, Age of Abalone is 15(Predicted output) +1.5 =16.5 years.

Application Building

Application Building involves following steps

1. Create an HTML file
2. Build a Python Code
3. Run the app

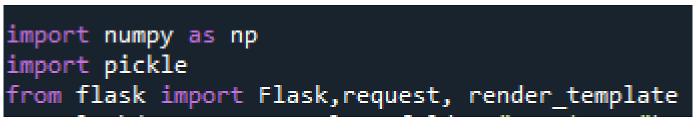
Create an HTML File

- We use HTML to create the front-end part of the web page.
- Here, we created 2 html pages- home.html, upload.html.
- home.html displays the home page.
- upload.html accepts the values from the user and displays the prediction.
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

Build Python Code

- Let us build a flask file ‘app.py’ which is a web framework written in python for server-side scripting. Let’s see the step by step procedure for building the backend application.
- App starts running when the “__name__” constructor is called in main.
- render_template is used to return an html file.
- “GET” method is used to take input from the user.
- “POST” method is used to display the output to the user.

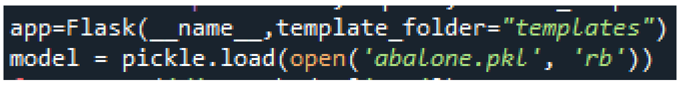
Importing Libraries



```python
import numpy as np
import pickle
from flask import Flask,request, render_template
```

Libraries required for the app to run are to be imported.

Creating our flask app and loading the model

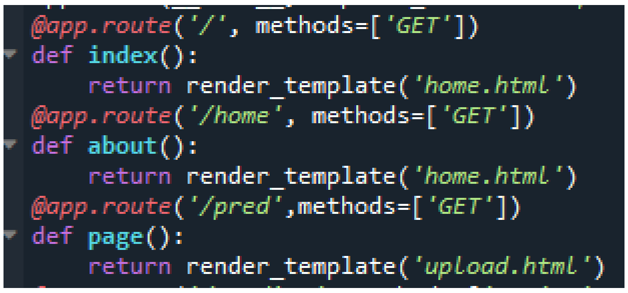


```python
app=Flask(__name__,template_folder="templates")
model = pickle.load(open('abalone.pkl', 'rb'))
```

Now after all the libraries are imported, we will be creating our flask app. and the load our model into our flask app.

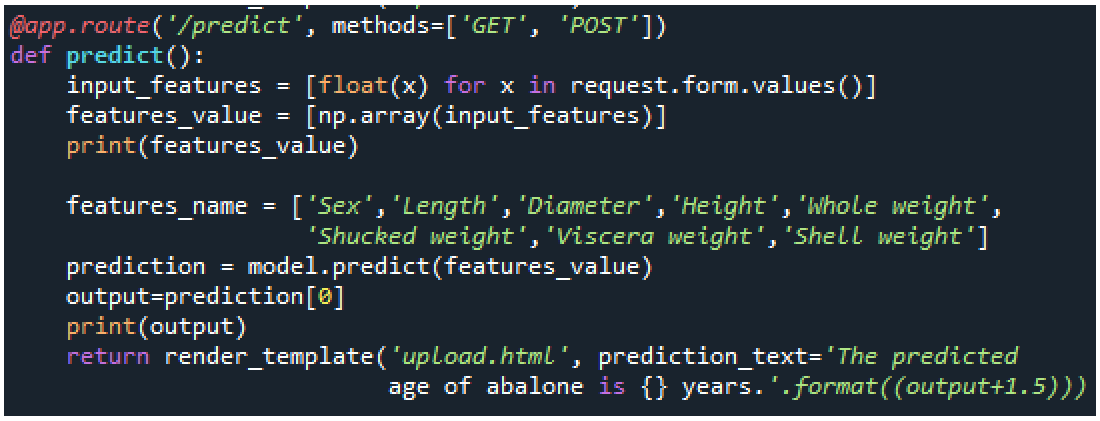
Routing to the html Page:

@app.route is used to route the application where it should route to.
‘/’ URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page is rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.
Here, “home.html” is rendered when the home button is clicked on the UI and “upload.html” is rendered when the predict button is clicked.



```
@app.route('/', methods=['GET'])
def index():
    return render_template('home.html')
@app.route('/home', methods=['GET'])
def about():
    return render_template('home.html')
@app.route('/pred',methods=['GET'])
def page():
    return render_template('upload.html')
```

Firstly, we are rendering the home.html template and from there we are navigating to our prediction page that is upload.html. We enter input values here and these values are sent to the loaded model and the resultant output is displayed on upload.html.

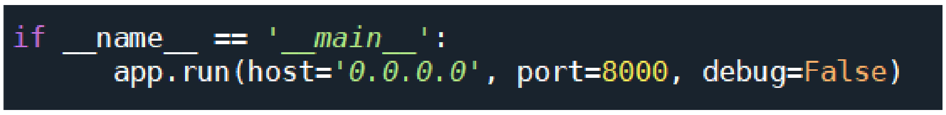


```
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    input_features = [float(x) for x in request.form.values()]
    features_value = [np.array(input_features)]
    print(features_value)

    features_name = ['Sex','Length','Diameter','Height','Whole weight',
                     'Shucked weight','Viscera weight','Shell weight']
    prediction = model.predict(features_value)
    output=prediction[0]
    print(output)
    return render_template('upload.html', prediction_text='The predicted
                           age of abalone is {} years.'.format((output+1.5)))
```

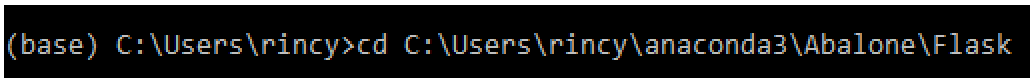
Main Function

This is used to run the application in a local host.



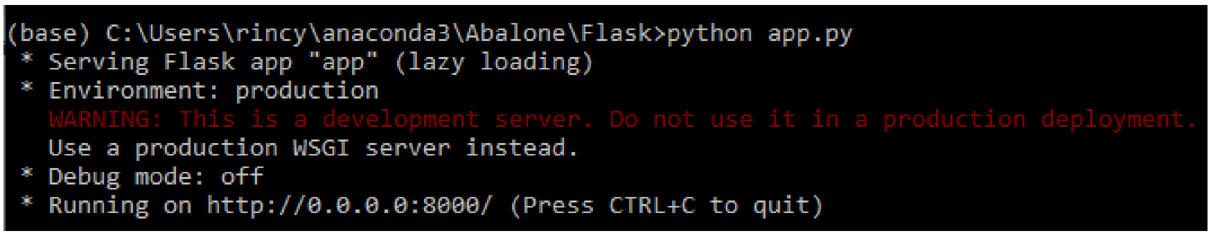
```
if __name__ == '__main__':
      app.run(host='0.0.0.0', port=8000, debug=False)
```

The local host runs on port number 8000.



```
(base) C:\Users\rincy>cd C:\Users\rincy\anaconda3\Abalone\Flask
```

This is used to run the application in a localhost.



```
(base) C:\Users\rincy\anaconda3\Abalone\Flask>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
```

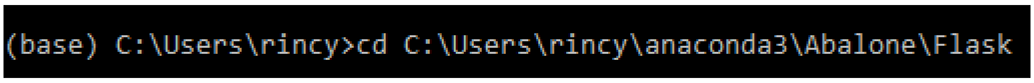
The local host runs on port number 8000.

Run the App

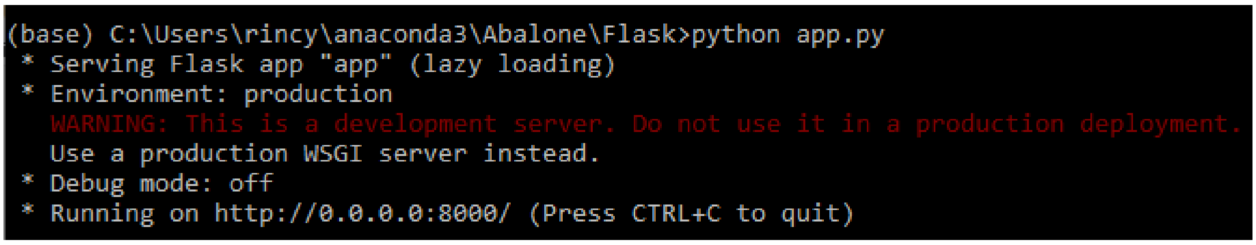
Run the application from anaconda prompt

- Open new anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command

- It will show the local host where your app is running on **http://127.0.0.1.8000/**
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page



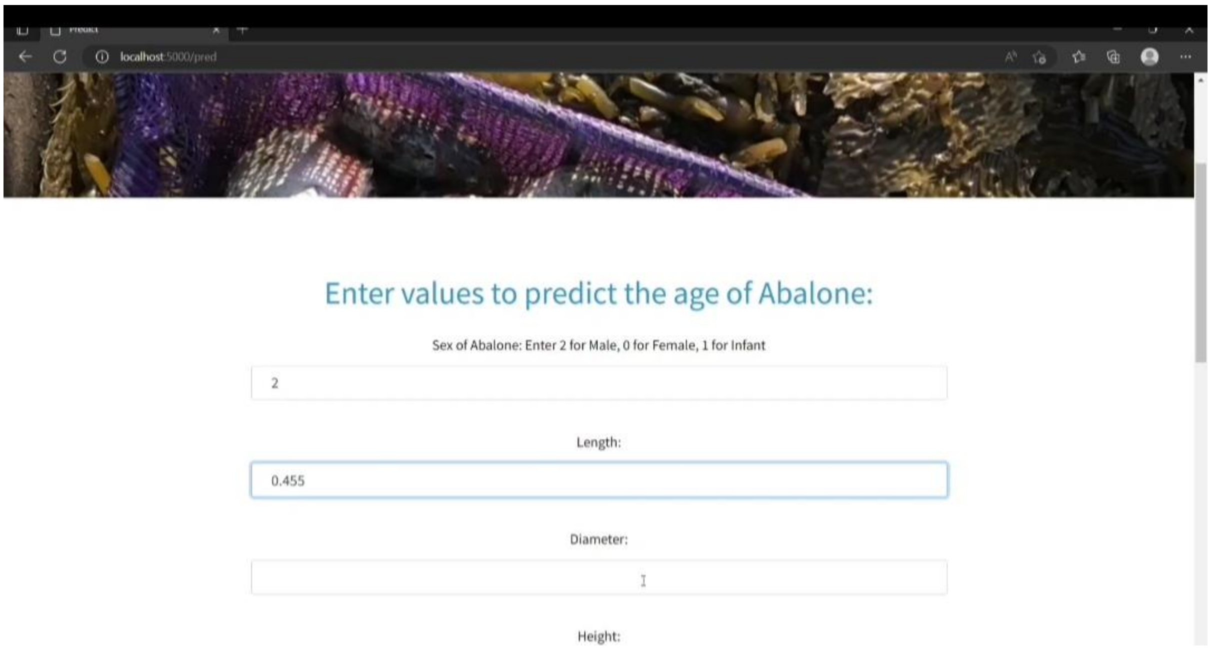
```
(base) C:\Users\rincy>cd C:\Users\rincy\anaconda3\Abalone\Flask
```



```
(base) C:\Users\rincy\anaconda3\Abalone\Flask>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8000/ (Press CTRL+C to quit)
```

- Showcasing the output on UI

Home page is displayed when home button is clicked. Predict page is displayed when predict button is clicked. In predict page, enter input values to predict the age of Abalone. Finally, the prediction for the given input features is shown.



Enter values to predict the age of Abalone:

Sex of Abalone: Enter 2 for Male, 0 for Female, 1 for Infant

2

Length:

0.455

Diameter:

Height: