

## Assignment – 3

### Python Programming

Assignment Date	
Student Name	CHENNURU SAI SADVIKA REDDY
Student Roll Number	111519104018
Maximum Marks	2 Marks

## 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.

### Importing Modules

```
In [ ]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

### 1. Dataset has been downloaded

```
In [ ]: #Name of the dataset:- abalone.csv
```

### 2. Load the dataset into the tool

```
In [ ]: data=pd.read_csv("abalone.csv")
data.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

Let's know the shape of the data

```
In [ ]: data.shape
```

```
Out[ ]: (4177, 9)
```

One additional task is that, we have to add the "Age" column using "Rings" data. We just have to add '1.5' to the ring data

```
In [ ]: Age=1.5+data.Rings
data["Age"]=Age
data=data.rename(columns = {'Whole weight': 'Whole_weight', 'Shucked weight': 'Shucked_weight', 'Shell weight': 'Shell_weight'})
data=data.drop(columns=["Rings"],axis=1)
data.head()
```

```
Out[ ]:
```

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0

### 3. Perform Below Visualizations.

#### (i) Univariate Analysis #

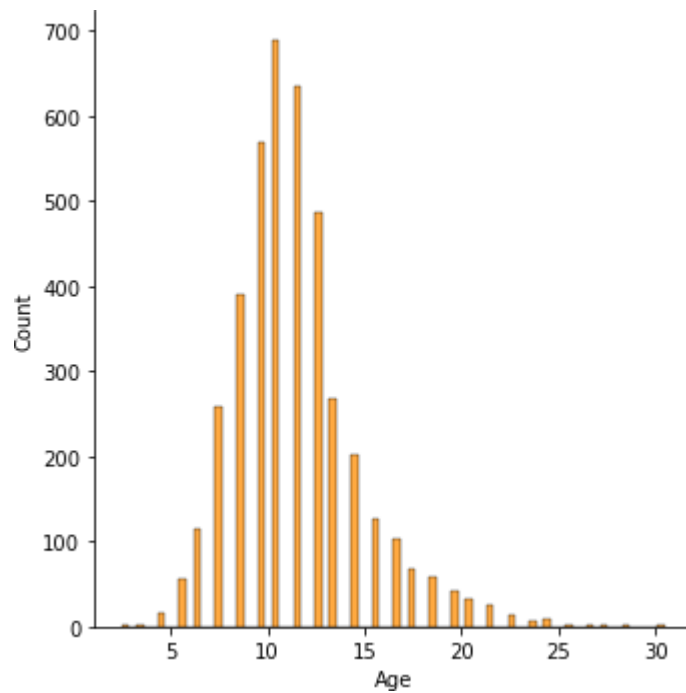
The term univariate analysis refers to the analysis of one variable. You can remember this because the prefix “uni” means “one.” There are three common ways to perform univariate analysis on one variable: 1. Summary statistics – Measures the center and spread of values.

#### #

#### Histogram

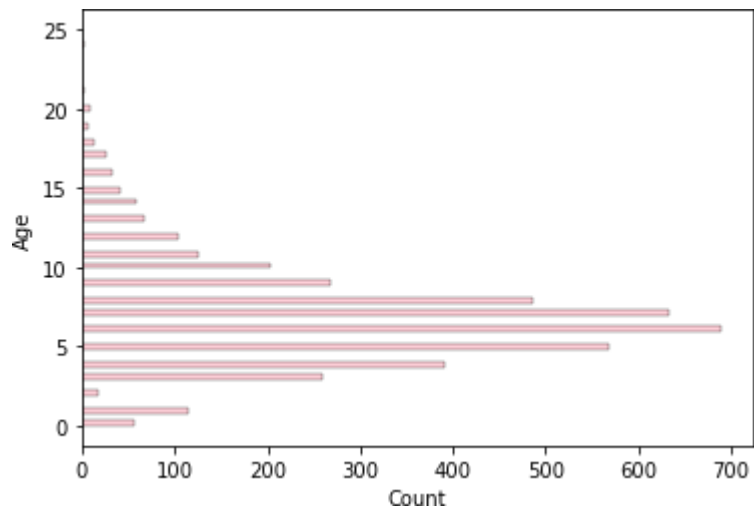
```
In [ ]: sns.displot(data["Age"], color='darkorange')
```

```
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fd3f837a430>
```



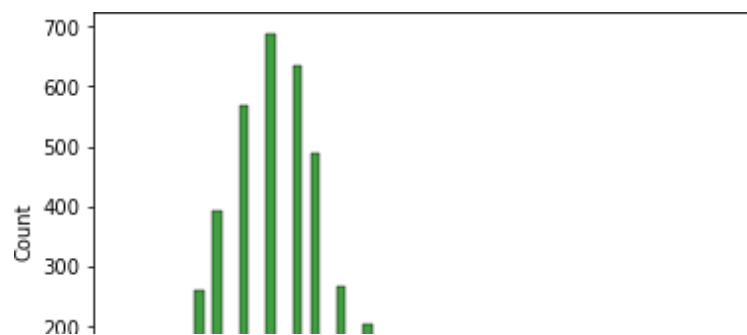
```
In [ ]: sns.histplot(y=data.Age,color='pink')
```

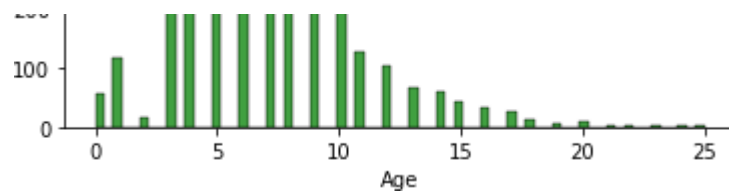
```
Out[ ]: <AxesSubplot:xlabel='Count', ylabel='Age'>
```



```
In [ ]: sns.histplot(x=data.Age,color='green')
```

```
Out[ ]: <AxesSubplot:xlabel='Age', ylabel='Count'>
```

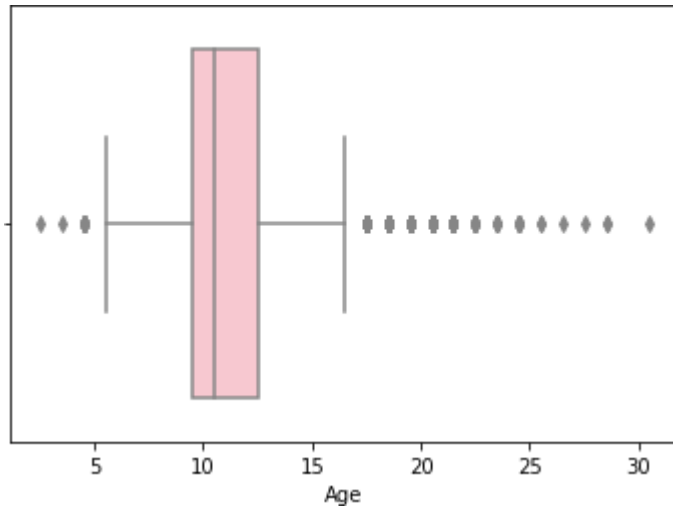




## Boxplot

```
In [ ]: sns.boxplot(x=data.Age,color='pink')
```

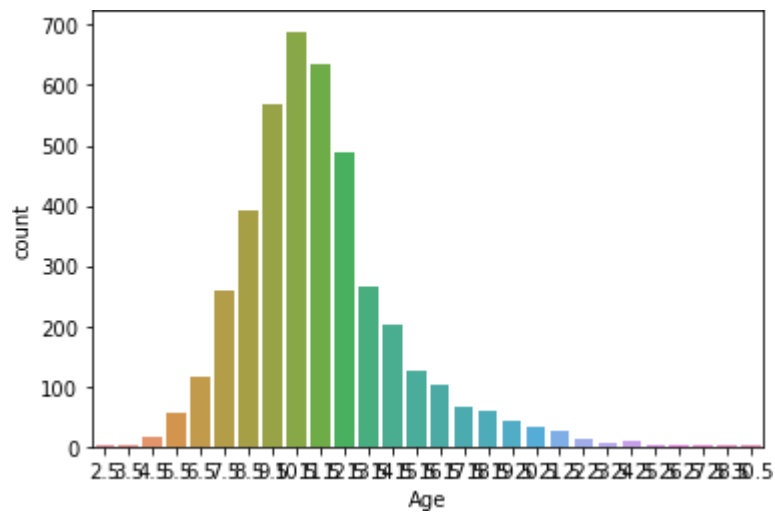
```
Out[ ]: <AxesSubplot:xlabel='Age'>
```



## Countplot

```
In [ ]: sns.countplot(x=data.Age)
```

```
Out[ ]: <AxesSubplot:xlabel='Age', ylabel='count'>
```



## (ii) Bi-Variate Analysis

#

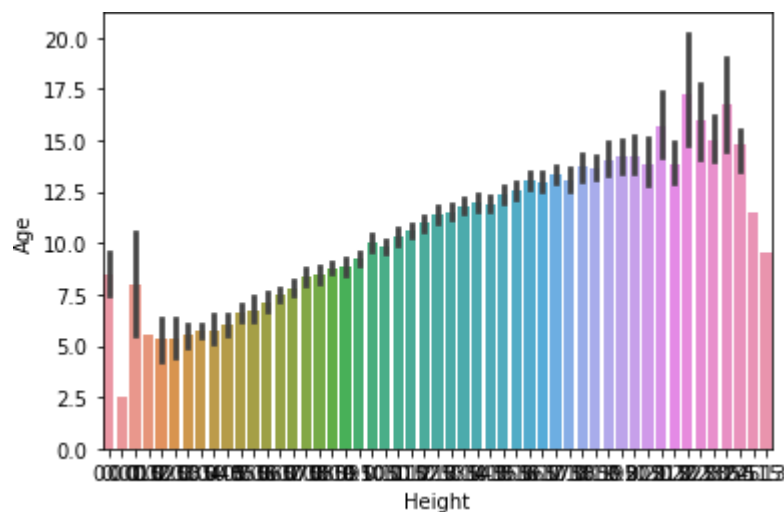
Image result for bivariate analysis in python It is a methodical statistical technique applied to a pair of variables (features/ attributes) of data to determine the empirical relationship between them. In order words, it is meant to determine any concurrent relations (usually over and above a simple correlation analysis).

#

## Barplot

```
In [ ]: sns.barplot(x=data.Height,y=data.Age)
```

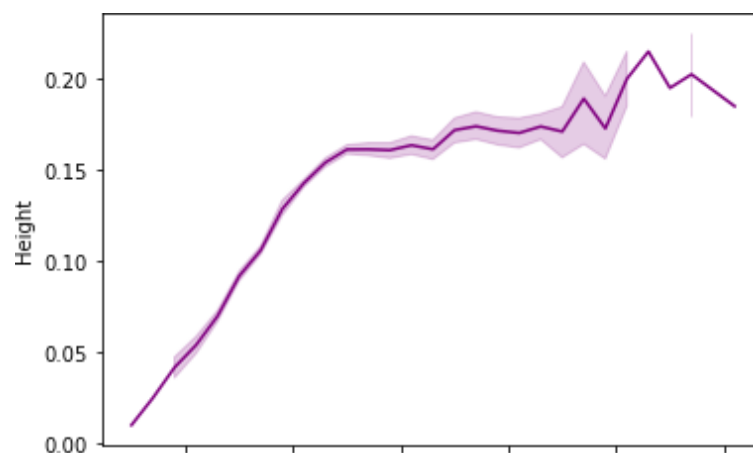
```
Out[ ]: <AxesSubplot:xlabel='Height', ylabel='Age'>
```



## Linearplot

```
In [ ]: sns.lineplot(x=data.Age,y=data.Height, color='purple')
```

```
Out[ ]: <AxesSubplot:xlabel='Age', ylabel='Height'>
```

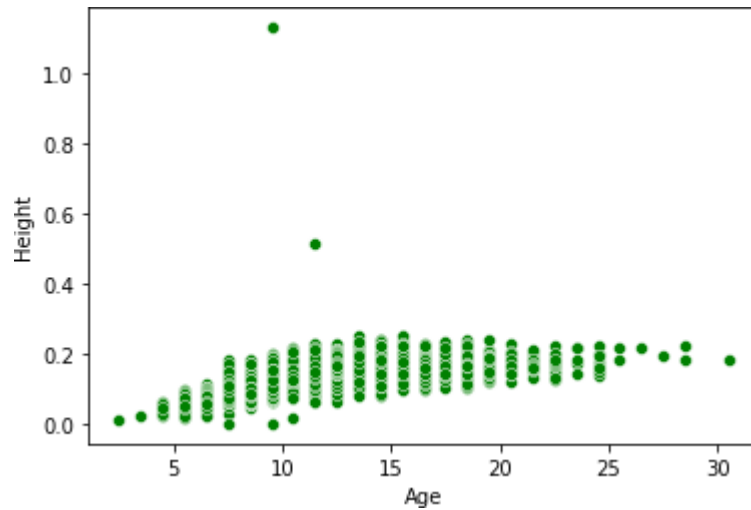


5      10      15      20      25      30  
Age

## Scatterplot

```
In [ ]: sns.scatterplot(x=data.Age, y=data.Height, color='green')
```

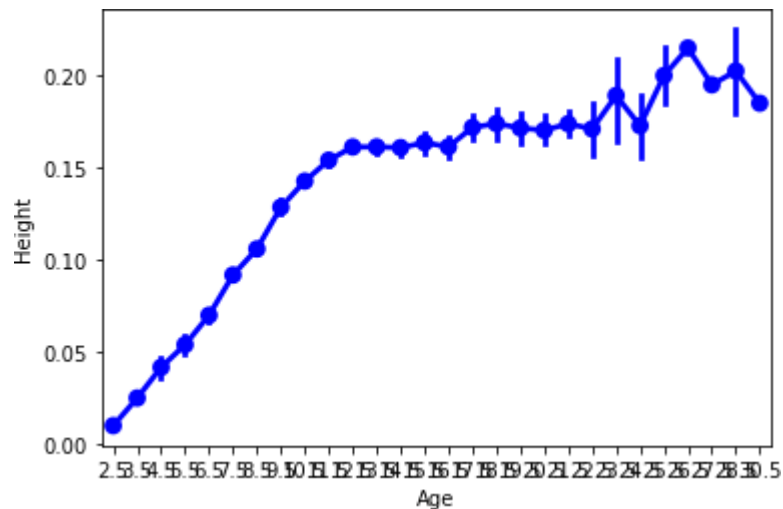
```
Out[ ]: <AxesSubplot:xlabel='Age', ylabel='Height'>
```



## Pointplot

```
In [ ]: sns.pointplot(x=data.Age, y=data.Height, color="blue")
```

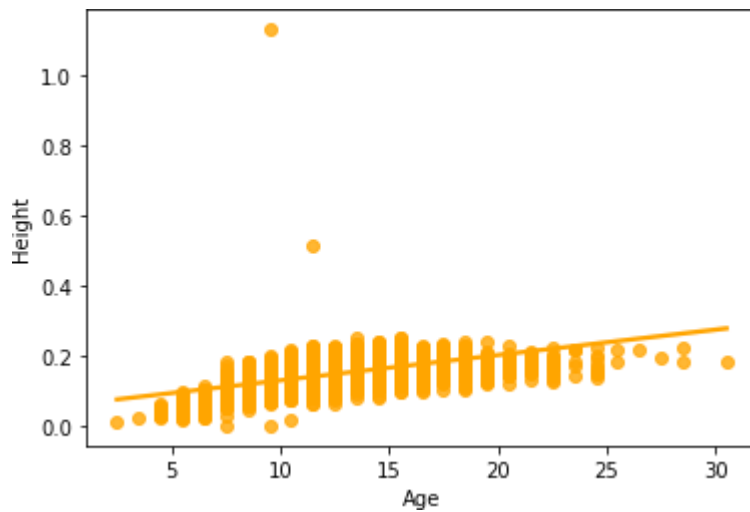
```
Out[ ]: <AxesSubplot:xlabel='Age', ylabel='Height'>
```



## Regplot

```
In [ ]: sns.regplot(x=data.Age, y=data.Height, color='orange')
```

```
Out[ ]: <AxesSubplot:xlabel='Age', ylabel='Height'>
```



### (iii) Multi-Variate Analysis

#

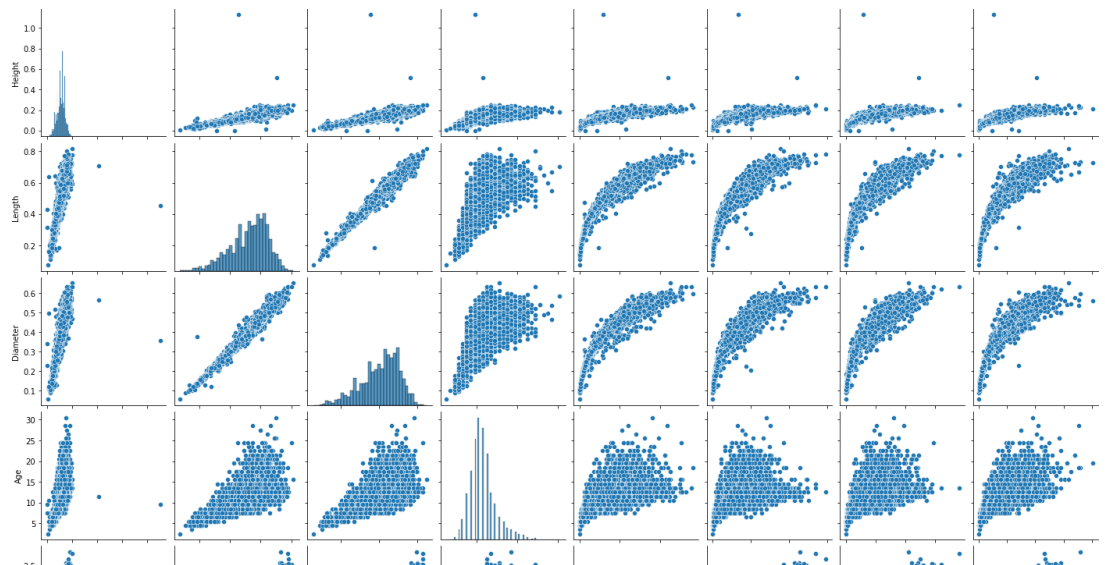
Multivariate analysis is based in observation and analysis of more than one statistical outcome variable at a time. In design and analysis, the technique is used to perform trade studies across multiple dimensions while taking into account the effects of all variables on the responses of interest.

#

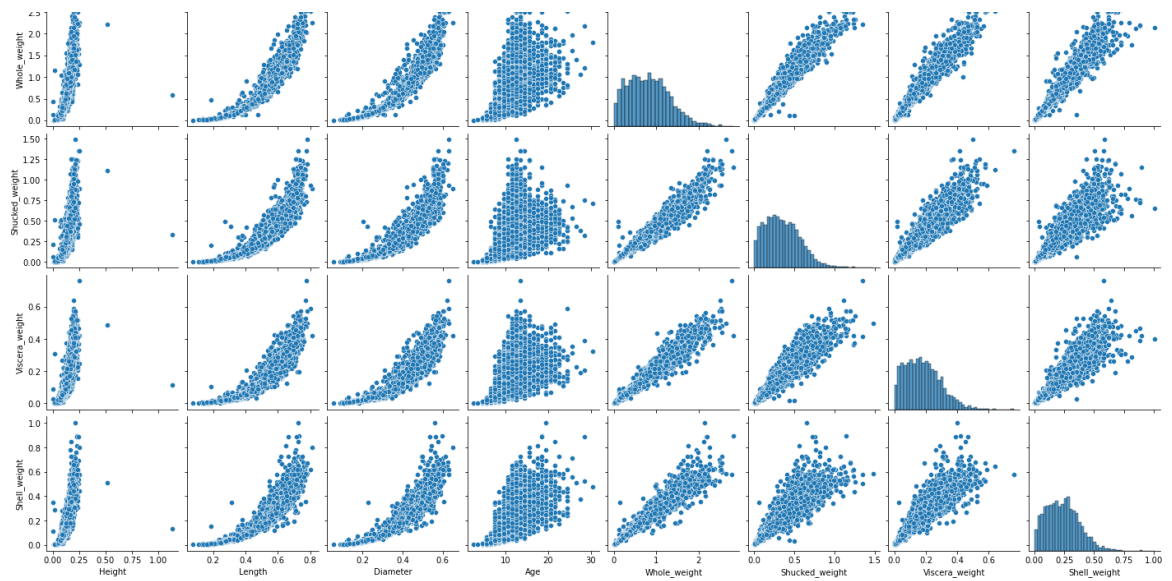
### Pairplot

```
In [ ]: sns.pairplot(data=data[["Height", "Length", "Diameter", "Age", "Whole_weight", "Shuc
```

```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x7fd3d93e1040>
```

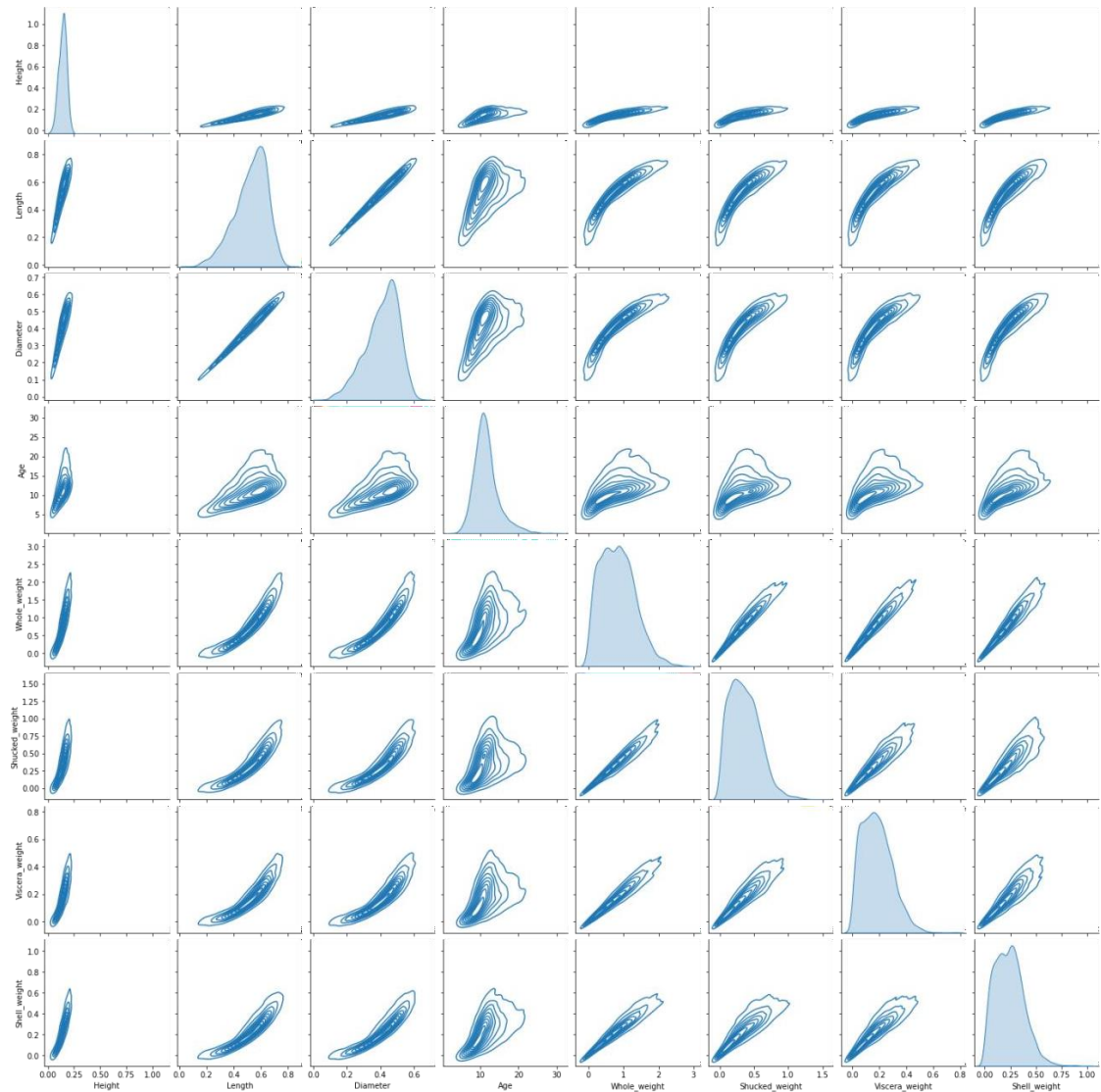






In [ ]: `sns.pairplot(data=data[["Height", "Length", "Diameter", "Age", "Whole_weight", "Shuc`

Out[ ]: `<seaborn.axisgrid.PairGrid at 0x7fd39840c790>`



## 4. Perform descriptive statistics on the dataset

```
In [ ]: data.describe(include='all')
```

```
Out[ ]:
```

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_
<b>count</b>	4177	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.
<b>unique</b>	3	NaN	NaN	NaN	NaN	NaN	
<b>top</b>	M	NaN	NaN	NaN	NaN	NaN	
<b>freq</b>	1528	NaN	NaN	NaN	NaN	NaN	
<b>mean</b>	NaN	0.523992	0.407881	0.139516	0.828742	0.359367	0.
<b>std</b>	NaN	0.120093	0.099240	0.041827	0.490389	0.221963	0.
<b>min</b>	NaN	0.075000	0.055000	0.000000	0.002000	0.001000	0.
<b>25%</b>	NaN	0.450000	0.350000	0.115000	0.441500	0.186000	0.
<b>50%</b>	NaN	0.545000	0.425000	0.140000	0.799500	0.336000	0.
<b>75%</b>	NaN	0.615000	0.480000	0.165000	1.153000	0.502000	0.
<b>max</b>	NaN	0.815000	0.650000	1.130000	2.825500	1.488000	0.

## 5. Check for Missing values and deal with them

```
In [ ]: data.isnull().sum()
```

```
Out[ ]:
```

Sex	0
Length	0
Diameter	0
Height	0
Whole_weight	0
Shucked_weight	0
Viscera_weight	0
Shell_weight	0
Age	0

dtype: int64

## 6. Find the outliers and replace them outliers

```
In [ ]: outliers=data.quantile(q=(0.25,0.75))
outliers
```

```
Out[ ]:
```

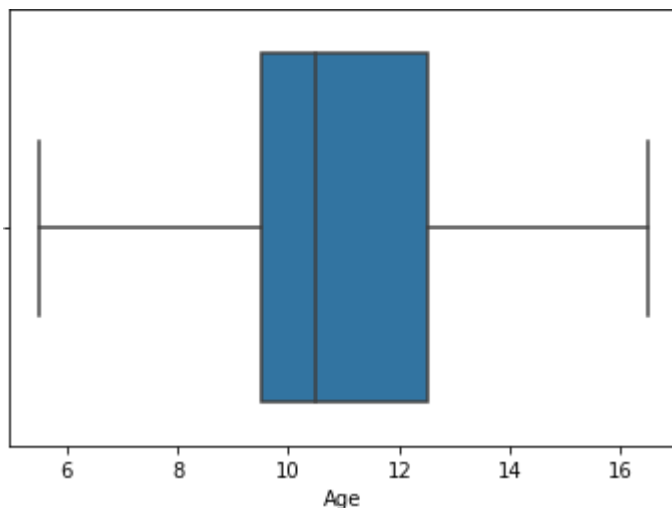
	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
<b>0.25</b>	0.450	0.35	0.115	0.4415	0.186	0.0935	0.130
<b>0.75</b>	0.615	0.48	0.165	1.1530	0.502	0.2530	0.329

```
In [ ]: a = data.Age.quantile(0.25)
b = data.Age.quantile(0.75)
c = b - a
lower_limit = a - 1.5 * c
data.median(numeric_only=True)
```

```
Out[ ]: 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
Age         10.5000
dtype: float64
```

```
In [ ]: data['Age'] = np.where(data['Age'] < lower_limit, 7, data['Age'])
sns.boxplot(x=data.Age, showfliers = False)
```

```
Out[ ]: <AxesSubplot:xlabel='Age'>
```



## 7. Check for Categorical columns and perform encoding

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

```
Out[ ]:
```

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weig
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

lab = LabelEncoder()
data.Sex = lab.fit_transform(data.Sex)

data.head()
```

Out[ ]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
0	2	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1
1	2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2
3	2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1
4	1	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0

## 8. Split the data into dependent and independent variables

In [ ]:

```
y = data["Sex"]
y.head()
```

Out[ ]:

```
0    2
1    2
2    0
3    2
4    1
Name: Sex, dtype: int64
```

In [ ]:

```
x=data.drop(columns=["Sex"],axis=1)
x.head()
```

Out[ ]:

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight	A
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	
4	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	

## 9. Scale the independent variables

In [ ]:

```
from sklearn.preprocessing import scale
X_Scaled = pd.DataFrame(scale(x), columns=x.columns)
X_Scaled.head()
```

```
Out[ ]:
```

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
0	-0.574558	-0.432149	-1.064424	-0.641898	-0.607685	-0.726212	-0.63821
1	-1.448986	-1.439929	-1.183978	-1.230277	-1.170910	-1.205221	-1.21298
2	0.050033	0.122130	-0.107991	-0.309469	-0.463500	-0.356690	-0.20713
3	-0.699476	-0.432149	-0.347099	-0.637819	-0.648238	-0.607600	-0.60229
4	-1.615544	-1.540707	-1.423087	-1.272086	-1.215968	-1.287337	-1.32075

## 10. Split the data into training and testing

```
In [ ]: from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X_Scaled, y, test_size=0.2,
```

```
In [ ]: X_Train.shape, X_Test.shape
```

```
Out[ ]: ((3341, 8), (836, 8))
```

```
In [ ]: Y_Train.shape, Y_Test.shape
```

```
Out[ ]: ((3341,), (836,))
```

```
In [ ]: X_Train.head()
```

```
Out[ ]:
```

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_w
3141	-2.864726	-2.750043	-1.423087	-1.622870	-1.553902	-1.583867	-1.64
3521	-2.573250	-2.598876	-2.020857	-1.606554	-1.551650	-1.565619	-1.62
883	1.132658	1.230689	0.728888	1.145672	1.041436	0.286552	1.53
3627	1.590691	1.180300	1.446213	2.164373	2.661269	2.330326	1.37
2106	0.591345	0.474853	0.370226	0.432887	0.255175	0.272866	0.90

```
In [ ]: X_Test.head()
```

```
Out[ ]:
```

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_w
668	0.216591	0.172519	0.370226	0.181016	-0.368878	0.569396	0.6
1580	-0.199803	-0.079426	-0.466653	-0.433875	-0.443224	-0.343004	-0.3

<b>3784</b>	0.799543	0.726798	0.370226	0.870348	0.755318	1.764639	0.56
<b>463</b>	-2.531611	-2.447709	-2.020857	-1.579022	-1.522362	-1.538247	-1.57
<b>2615</b>	1.007740	0.928354	0.848442	1.390405	1.415417	1.778325	0.99

```
In [ ]: Y_Train.head()
```

```
Out[ ]: 3141    1
        3521    1
        883     2
        3627    2
        2106    2
        Name: Sex, dtype: int64
```

```
In [ ]: Y_Test.head()
```

```
Out[ ]: 668     2
        1580    1
        3784    2
        463     1
        2615    2
        Name: Sex, dtype: int64
```

## 11. Build the Model

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
        model = RandomForestClassifier(n_estimators=10, criterion='entropy')
```

```
In [ ]: model.fit(X_Train, Y_Train)
```

```
Out[ ]: RandomForestClassifier(criterion='entropy', n_estimators=10)
```

```
In [ ]: y_predict = model.predict(X_Test)
```

```
In [ ]: y_predict_train = model.predict(X_Train)
```

## 12. Train the Model

```
In [ ]: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
In [ ]: print('Training accuracy: ', accuracy_score(Y_Train, y_predict_train))
```

Training accuracy: 0.9787488775815624

## 13. Test the Model

```
In [ ]: print('Testing accuracy: ',accuracy_score(Y_Test,y_predict))
```

Testing accuracy: 0.5526315789473685

## 14. Measure the performance using Metrics

```
In [ ]: pd.crosstab(Y_Test,y_predict)
```

```
Out[ ]: col_0    0    1    2
        Sex
0  122   29   98
1   37  217   37
2  120   53  123
```

```
In [ ]: print(classification_report(Y_Test,y_predict))
```

	precision	recall	f1-score	support
0	0.44	0.49	0.46	249
1	0.73	0.75	0.74	291
2	0.48	0.42	0.44	296
accuracy			0.55	836
macro avg	0.55	0.55	0.55	836
weighted avg	0.55	0.55	0.55	836