**Assignment – 3**

Python Programming

|  |  |
| --- | --- |
| Assignment Date |  |
| Student Name | NIVEDA J |
| Student Roll Number | 111519104092 |
| 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

|  |
| --- |
| **import** pandas **as** pd **import** seaborn **as** sns **import** matplotlib.pyplot **as** plt **import** numpy **as** np |

In [ ]:

# 1. Dataset has been downloaded

|  |
| --- |
| *#Name of the dataset: abalone.csv* |

In [ ]:

# 2. Load the dataset into the tool

|  |
| --- |
| data**=**pd**.**read\_csv("abalone.csv") data**.**head() |

In [ ]:

**Whole  Shucked Viscera Shell**

Out []:**Sex Length Diameter Height Rings weight weight weight weight**

1. M 0.455 0.365 0.095 0.5140 0.2245 0.1010 0.150 15
2. M 0.350 0.265 0.090 0.2255 0.0995 0.0485 0.070 7
3. F 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.210 9
4. M 0.440 0.365 0.125 0.5160 0.2155 0.1140 0.155 10
5. I 0.330 0.255 0.080 0.2050 0.0895 0.0395 0.055 7

## Let's know the shape of the data

|  |
| --- |
| data**.**shape |

In [ ]:

(4177, 9) Out[ ]:

## 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

|  |
| --- |
| Age**=**1.5**+**data**.**Rings data["Age"]**=**Age data**=**data**.**rename(columns **=** {'Whole weight':'Whole\_weight','Shucked weight': 'Sh  'Shell weight': 'Shell\_weight'})  data**=**data**.**drop(columns**=**["Rings"],axis**=**1) data**.**head() |

In [ ]:

Out[ ]: **Sex Length Diameter Height Whole\_weight Shucked\_weight Viscera\_weight Shell\_weig**

1. 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

1. F 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.2

1. M 0.440 0.365 0.125 0.5160 0.2155 0.1140 0.1
2. 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.

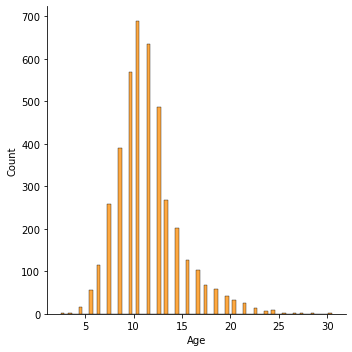
**#**

## Histogram

|  |
| --- |
| sns**.**displot(data["Age"], color**=**'darkorange') |

In [ ]:

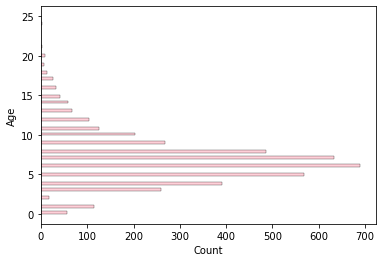
<seaborn.axisgrid.FacetGrid at 0x7fd3f837a430> Out[ ]:



|  |
| --- |
| sns**.**histplot(y**=**data**.**Age,color**=**'pink') |

In [ ]:

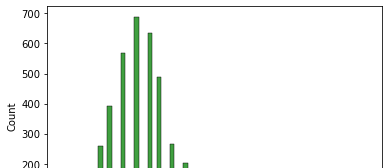
<AxesSubplot:xlabel='Count', ylabel='Age'> Out[ ]:

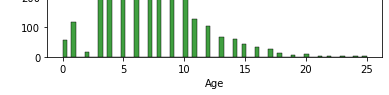


|  |
| --- |
| sns**.**histplot(x**=**data**.**Age,color**=**'green') |

In [ ]:

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



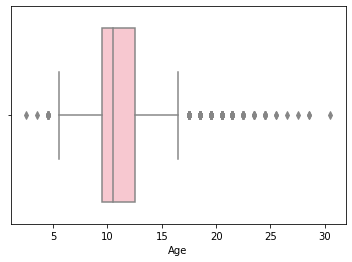


## Boxplot

|  |
| --- |
| sns**.**boxplot(x**=**data**.**Age,color**=**'pink') |

In [ ]:

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

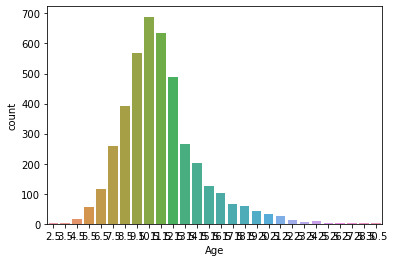


## Countplot

|  |
| --- |
| sns**.**countplot(x**=**data**.**Age) |

In [ ]:

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



# (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).

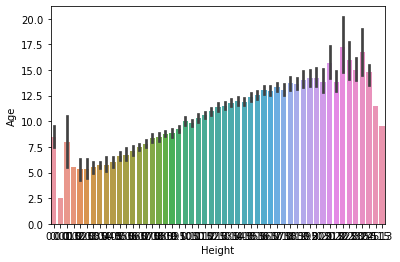
**#**

## Barplot

|  |
| --- |
| sns**.**barplot(x**=**data**.**Height,y**=**data**.**Age) |

In [ ]:

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

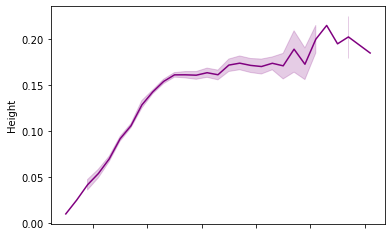


## Linearplot

|  |
| --- |
| sns**.**lineplot(x**=**data**.**Age,y**=**data**.**Height, color**=**'purple') |

In [ ]:

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



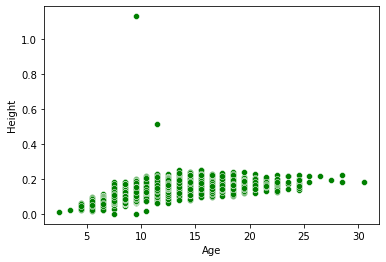


## Scatterplot

|  |
| --- |
| sns**.**scatterplot(x**=**data**.**Age,y**=**data**.**Height,color**=**'green') |

In [ ]:

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

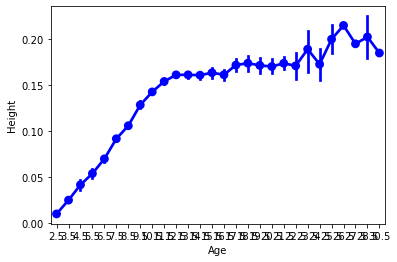


## Pointplot

|  |
| --- |
| sns**.**pointplot(x**=**data**.**Age, y**=**data**.**Height, color**=**"blue") |

In [ ]:

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

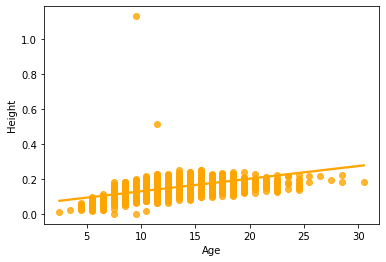


## Regplot

|  |
| --- |
| sns**.**regplot(x**=**data**.**Age,y**=**data**.**Height,color**=**'orange') |

In [ ]:

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



# (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.

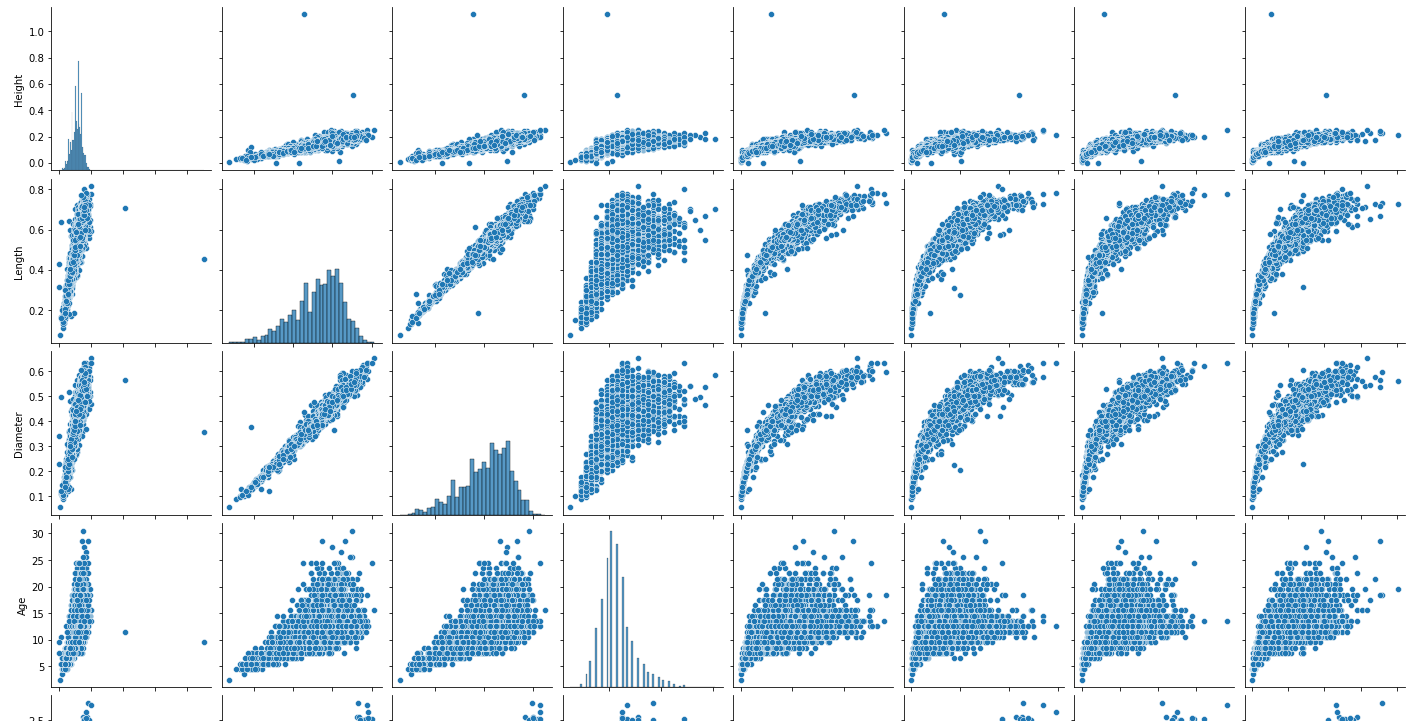
**#**

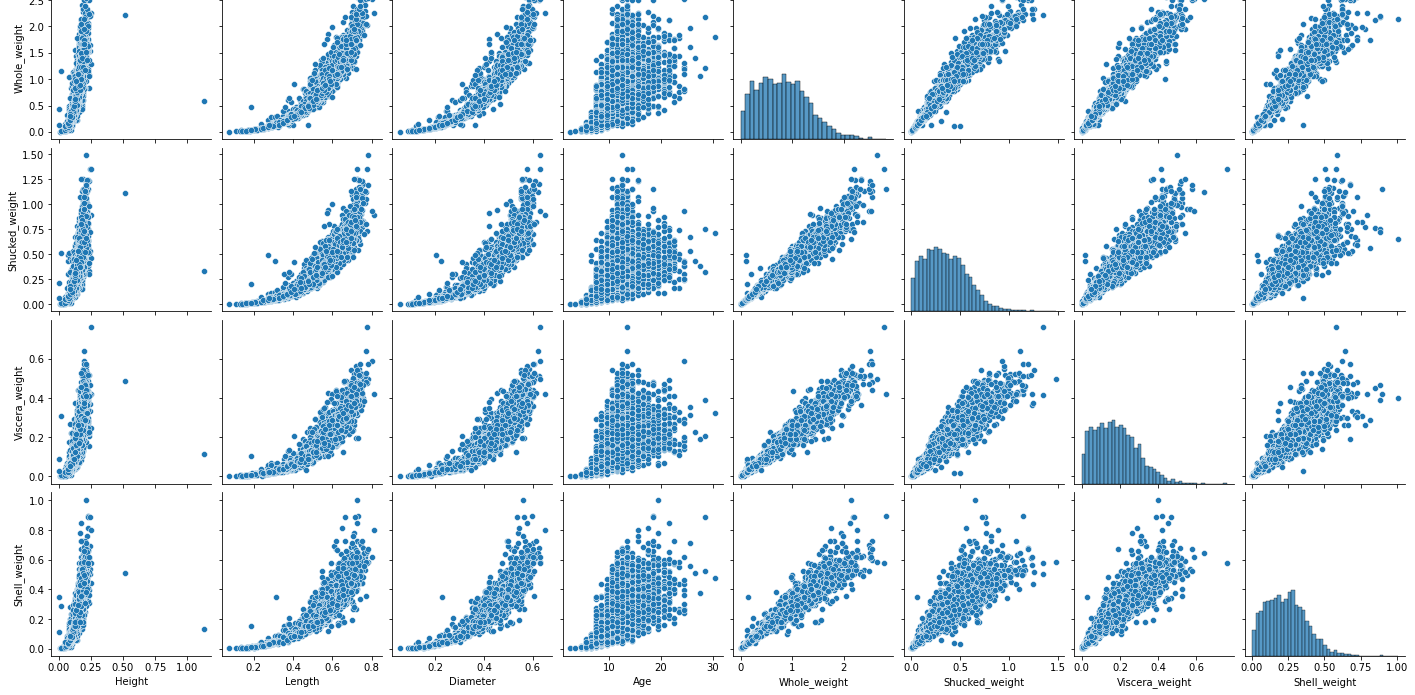
## Pairplot

|  |
| --- |
| sns**.**pairplot(data**=**data[["Height","Length","Diameter","Age","Whole\_weight","Shuc |

In [ ]:

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

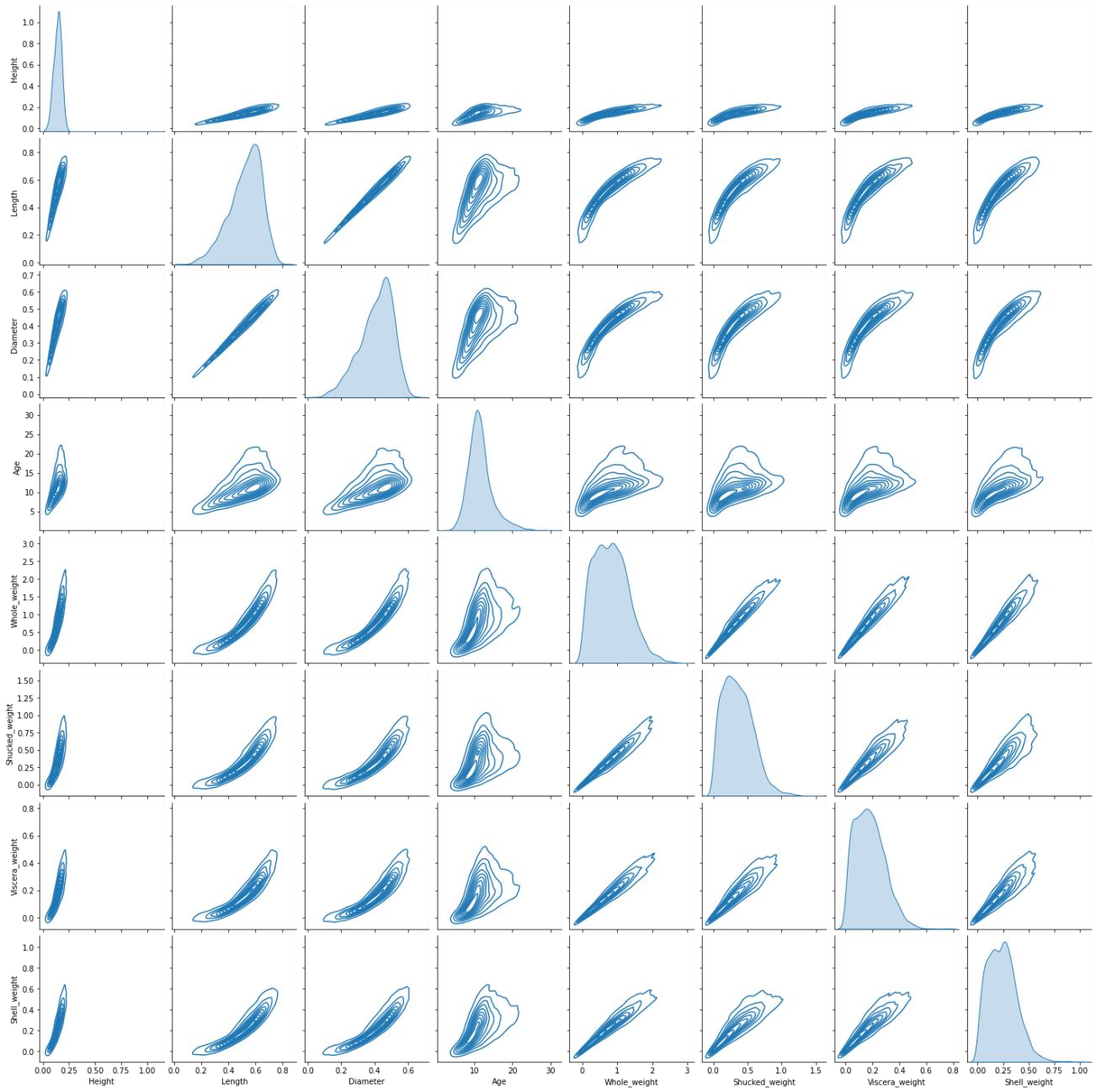




|  |
| --- |
| sns**.**pairplot(data**=**data[["Height","Length","Diameter","Age","Whole\_weight","Shuc |

In [ ]:

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



# 4. Perform descriptive statistics on the dataset

|  |
| --- |
| data**.**describe(include**=**'all') |

In [ ]:

Out[ ]: **Sex Length Diameter Height Whole\_weight Shucked\_weight Viscera\_**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **count** 4177 4177.000000 4177.000000 4177.000000 | | | | 4177.000000 | 4177.000000 | 4177. |
| 3 **unique** | NaN | NaN | NaN | NaN | NaN |  |
| **top** M | NaN | NaN | NaN | NaN | NaN |  |
| **freq** 1528 | NaN | NaN | NaN | NaN | NaN |  |
| **mean** NaN | 0.523992 | 0.407881 | 0.139516 | 0.828742 | 0.359367 | 0. |
| **std** NaN | 0.120093 | 0.099240 | 0.041827 | 0.490389 | 0.221963 | 0. |
| **min** NaN | 0.075000 | 0.055000 | 0.000000 | 0.002000 | 0.001000 | 0. |
| **25%** NaN | 0.450000 | 0.350000 | 0.115000 | 0.441500 | 0.186000 | 0. |
| **50%** NaN | 0.545000 | 0.425000 | 0.140000 | 0.799500 | 0.336000 | 0. |
| **75%** NaN | 0.615000 | 0.480000 | 0.165000 | 1.153000 | 0.502000 | 0. |
| **max** NaN | 0.815000 | 0.650000 | 1.130000 | 2.825500 | 1.488000 | 0. |

# 5. Check for Missing values and deal with them

|  |  |
| --- | --- |
| data**.**isnull()**.**sum() | |
| 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** |
| outliers**=**data**.**quantile(q**=**(0.25,0.75)) outliers | |

In [ ]:

Out[ ]:

In [ ]:

Out[ ]: **Length Diameter Height Whole\_weight Shucked\_weight Viscera\_weight Shell\_weight**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0.25** 0.450 0.35 0.115 | 0.4415 | 0.186 | 0.0935 | 0.130 |
| **0.75** 0.615 0.48 0.165 1.1530 0.502 0.2530 0.329 | | | | |

|  |
| --- |
| 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**) |

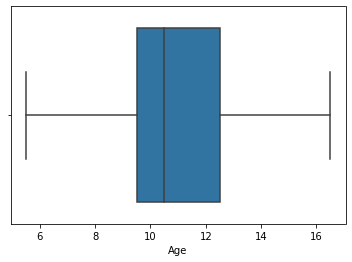
In [ ]:

|  |  |
| --- | --- |
| Length Out[ ]:  Diameter  Height  Whole\_weight  Shucked\_weight  Viscera\_weight  Shell\_weight Age  dtype: float64 | 0.5450  0.4250 0.1400 0.7995 0.3360 0.1710  0.2340  10.5000 |

|  |
| --- |
| data['Age'] **=** np**.**where(data['Age'] **<** lower\_limit, 7, data['Age']) sns**.**boxplot(x**=**data**.**Age,showfliers **= False**) |

In [ ]:

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



# 7. Check for Categorical columns and perform encoding

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ]: ]: | data**.**head() | |  |  |  |  |  |  |
| **Sex** | **Length** | **Diameter** | **Height** | **Whole\_weight** | **Shucked\_weight** | **Viscera\_weight** | **Shell\_weig** |
|  | 1. M 2. M 3. F 4. M 5. I | 0.455  0.350  0.530  0.440  0.330 | 0.365  0.265  0.420  0.365  0.255 | 0.095  0.090  0.135  0.125  0.080 | 0.5140  0.2255  0.6770  0.5160  0.2050 | 0.2245  0.0995  0.2565  0.2155  0.0895 | 0.1010  0.0485  0.1415  0.1140  0.0395 | 0.1  0.0  0.2  0.1  0.0 |

In [

Out[

|  |
| --- |
| **from** sklearn.preprocessing **import** LabelEncoder  lab **=** LabelEncoder()  data**.**Sex **=** lab**.**fit\_transform(data**.**Sex)  data**.**head() |

In [ ]:

Out[ ]: **Sex Length Diameter Height Whole\_weight Shucked\_weight Viscera\_weight Shell\_weig**

1. 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

1. 0 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.2

1. 2 0.440 0.365 0.125 0.5160 0.2155 0.1140 0.1
2. 1 0.330 0.255 0.080 0.2050 0.0895 0.0395 0.0

# 8. Split the data into dependent and independent variables

|  |
| --- |
| y **=** data["Sex"] y**.**head() |

In [ ]:

1. 2

Out[ ]:

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**

1. 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

1. 0.530 0.420 0.135 0.6770 0.2565 0.1415 0.210

1. 0.440 0.365 0.125 0.5160 0.2155 0.1140 0.155
2. 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\_weigh**

1. -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

1. 0.050033 0.122130 -0.107991 -0.309469 -0.463500 -0.356690 -0.20713

1. -0.699476 -0.432149 -0.347099 -0.637819 -0.648238 -0.607600 -0.60229

1. -1.615544 -1.540707 -1.423087 -1.272086 -1.215968 -1.287337 -1.32075

# 10. Split the data into training and testing

|  |
| --- |
| **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 |

In [ ]:

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

|  |
| --- |
| Y\_Train**.**shape,Y\_Test**.**shape |

In [ ]:

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X\_Test**.**head() |  |  |  |  |  |
| **Length Diameter** | **Height** | **Whole\_weight** | **Shucked\_weight** | **Viscera\_weight** | **Shell\_w** |

In [ ]:

Out[ ]:

**668** 0.216591 0.172519 0.370226 0.181016 -0.368878 0.569396 0.69

**1580** -0.199803 -0.079426 -0.466653 -0.433875 -0.443224 -0.343004 -0.32

**3784** 0.799543 0.726798 0.370226 0.870348 0.755318 1.764639 0.56

**463** -2.531611 -2.447709 -2.020857 -1.579022 -1.522362 -1.538247 -1.57

**2615** 1.007740 0.928354 0.848442 1.390405 1.415417 1.778325 0.99

|  |
| --- |
| Y\_Train**.**head() |

In [ ]:

3141 1

Out[ ]:

3521 1

883 2

3627 2

2106 2

Name: Sex, dtype: int64

|  |
| --- |
| Y\_Test**.**head() |

In [ ]:

668 2

Out[ ]:

1580 1

3784 2

463 1

2615 2

Name: Sex, dtype: int64

# 11. Build the Model

|  |
| --- |
| **from** sklearn.ensemble **import** RandomForestClassifier  model **=** RandomForestClassifier(n\_estimators**=**10,criterion**=**'entropy') |

In [ ]:

|  |
| --- |
| model**.**fit(X\_Train,Y\_Train) |

In [ ]:

RandomForestClassifier(criterion='entropy', n\_estimators=10) Out[ ]:

|  |
| --- |
| y\_predict **=** model**.**predict(X\_Test) |

In [ ]:

|  |
| --- |
| y\_predict\_train **=** model**.**predict(X\_Train) |

In [ ]:

# 12. Train the Model

|  |
| --- |
| **from** sklearn.metrics **import** accuracy\_score,confusion\_matrix,classification\_repo |

In [ ]:

|  |
| --- |
| print('Training accuracy: ',accuracy\_score(Y\_Train,y\_predict\_train)) |

In [ ]:

Training accuracy: 0.9787488775815624

# 13. Test the Model

|  |
| --- |
| print('Testing accuracy: ',accuracy\_score(Y\_Test,y\_predict)) |

In [ ]:

Testing accuracy: 0.5526315789473685

# 14. Measure the performance using Metrics

|  |
| --- |
| pd**.**crosstab(Y\_Test,y\_predict) |

In [ ]:

Out[ ]: **col\_0 0 1 2**

**Sex**

1. 122 29 98

1. 37 217 37
2. 120 53 123

|  |
| --- |
| print(classification\_report(Y\_Test,y\_predict)) |

In [ ]:

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 |