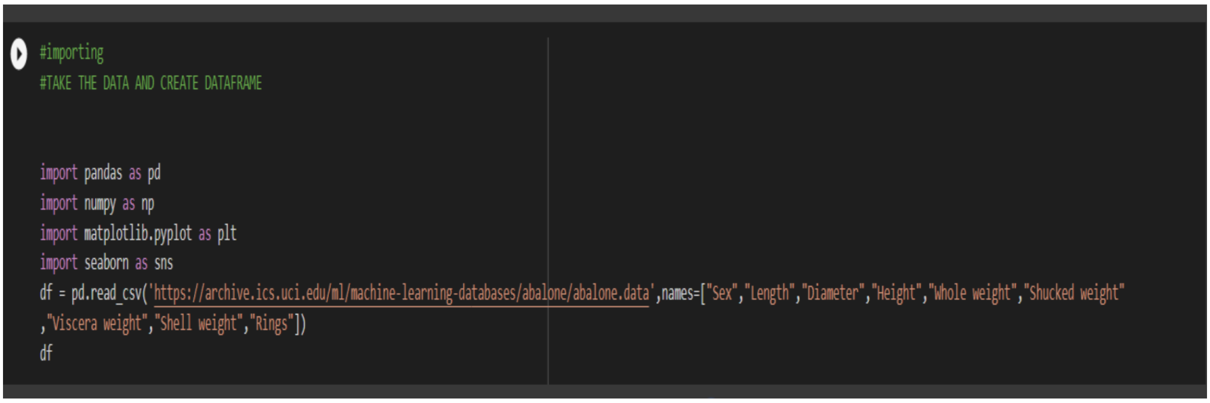
MAJOR PROJECT 1

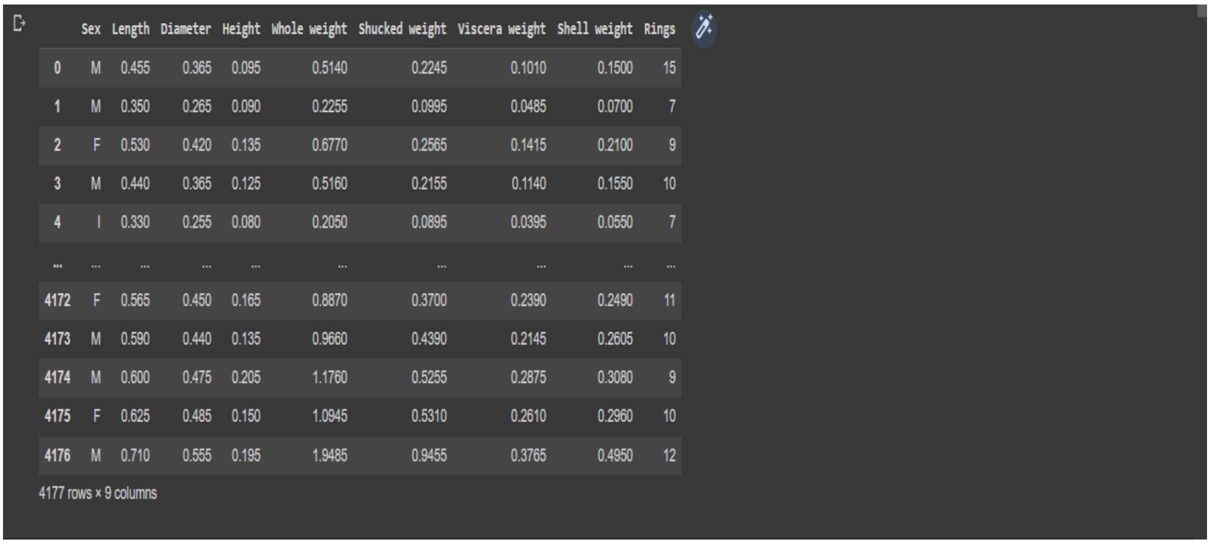
Choose any dataset of your choice and apply a suitable CLASSIFIER/REGRESSOR .

DATASET-https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/

(Abalone dataset)

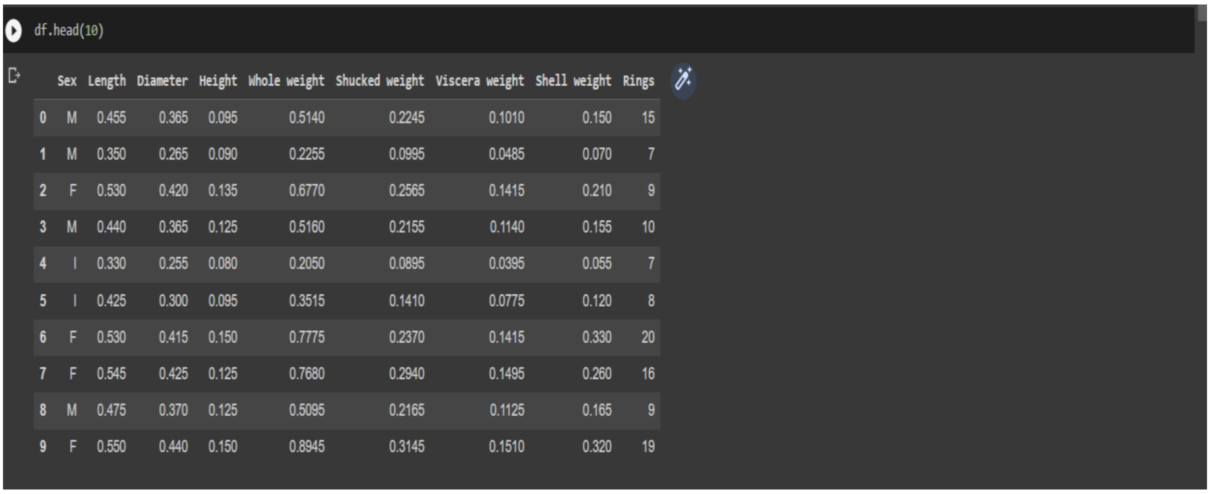


```
#importing
#TAKE THE DATA AND CREATE DATAFRAME



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data',names=["Sex","Length","Diameter","Height","Whole weight","Shucked weight"
,"Viscera weight","Shell weight","Rings"])
df
```



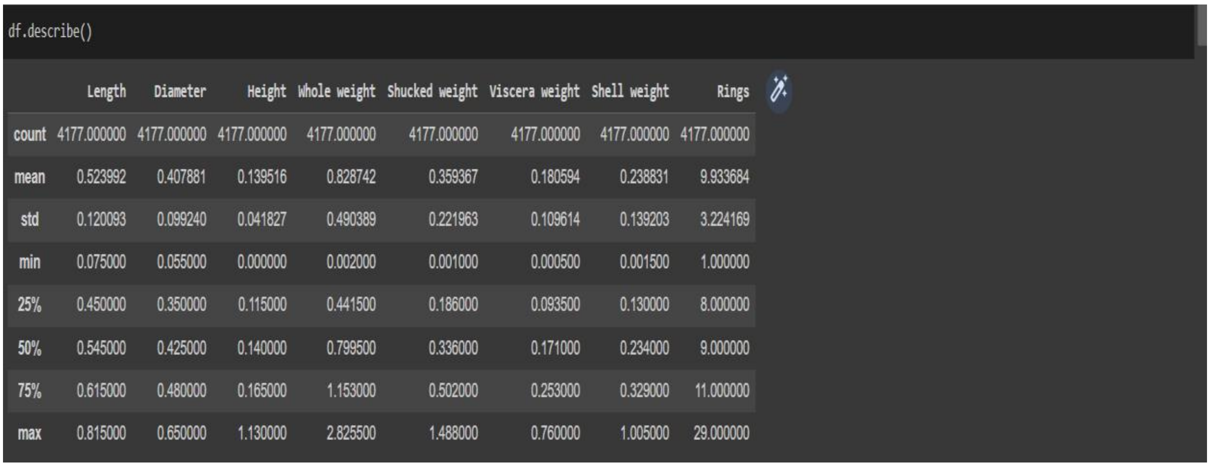
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	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



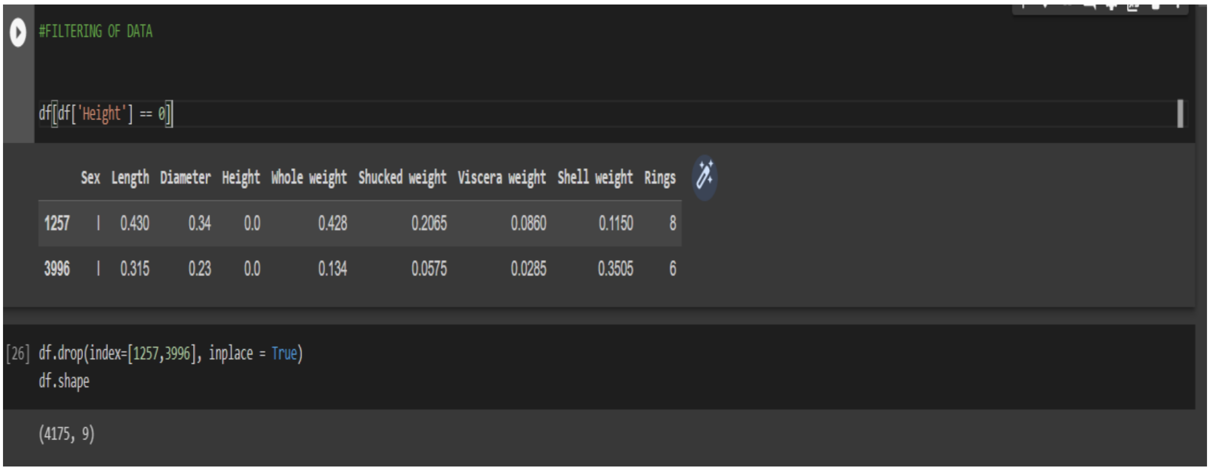
```
df.head(10)
```

	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
5	I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8
6	F	0.530	0.415	0.150	0.7775	0.2370	0.1415	0.330	20
7	F	0.545	0.425	0.125	0.7680	0.2940	0.1495	0.260	16
8	M	0.475	0.370	0.125	0.5095	0.2165	0.1125	0.165	9
9	F	0.550	0.440	0.150	0.8945	0.3145	0.1510	0.320	19



```
df.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

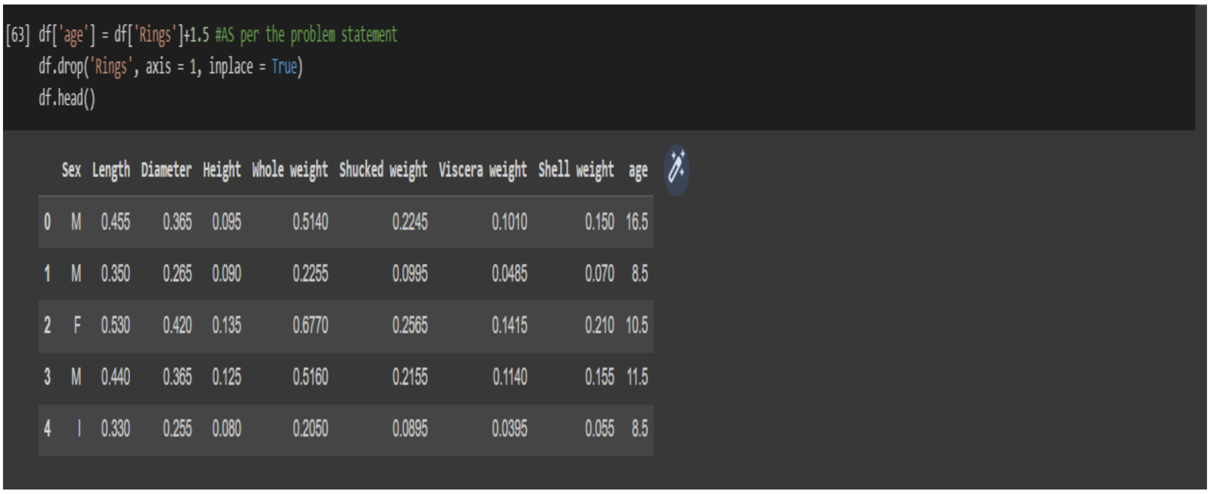


```
#FILTERING OF DATA


df[df['Height'] == 0]
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
1257	I	0.430	0.34	0.0	0.428	0.2065	0.0860	0.1150	8
3996	I	0.315	0.23	0.0	0.134	0.0575	0.0285	0.3505	6

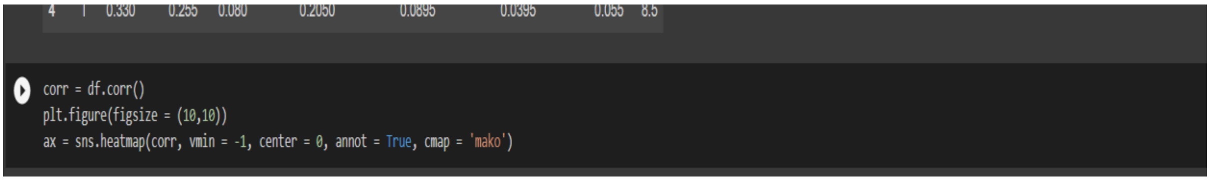
```
[26] df.drop(index=[1257,3996], inplace = True)
     df.shape
```

```
(4175, 9)
```



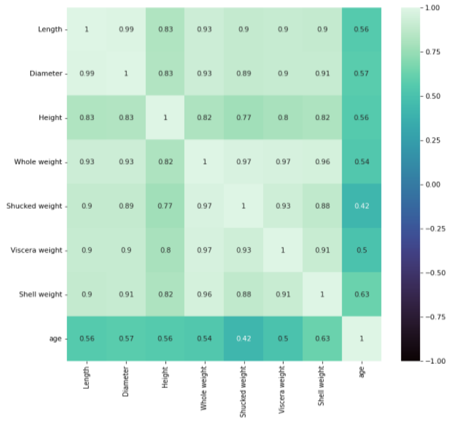
```
[63] df['age'] = df['Rings']+1.5 #AS per the problem statement
     df.drop('Rings', axis = 1, inplace = True)
     df.head()
```

```
   Sex  Length  Diameter  Height  Whole weight  Shucked weight  Viscera weight  Shell weight   age
0    M   0.455     0.365   0.095        0.5140          0.2245          0.1010         0.150  16.5
1    M   0.350     0.265   0.090        0.2255          0.0995          0.0485         0.070   8.5
2    F   0.530     0.420   0.135        0.6770          0.2565          0.1415         0.210  10.5
3    M   0.440     0.365   0.125        0.5160          0.2155          0.1140         0.155  11.5
4    I   0.330     0.255   0.080        0.2050          0.0895          0.0395         0.055   8.5
```

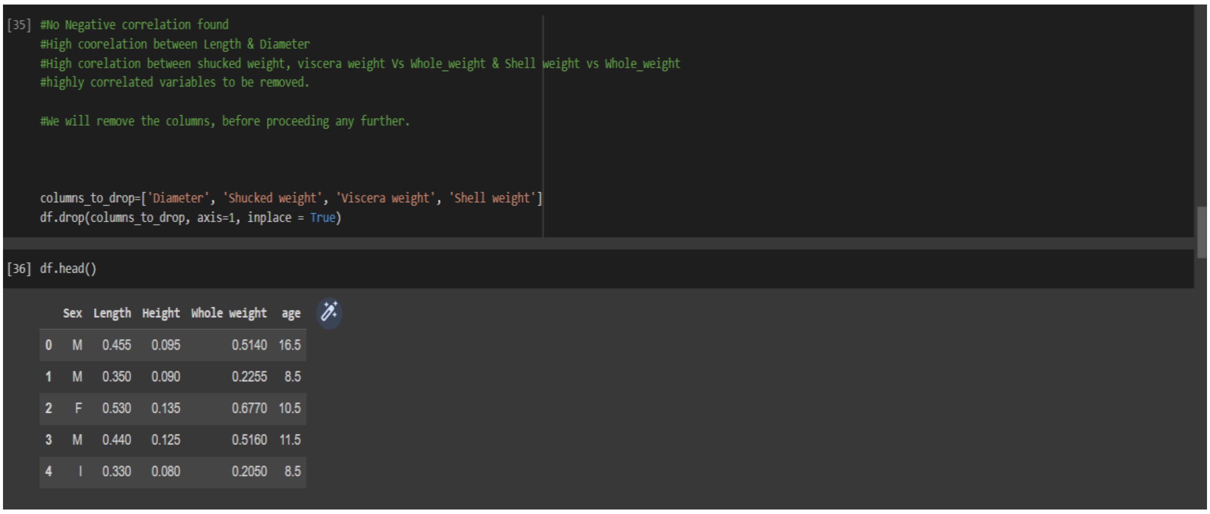


```
4    I   0.330     0.255   0.080        0.2050          0.0895          0.0395         0.055   8.5
```

```
corr = df.corr()
plt.figure(figsize = (10,10))
ax = sns.heatmap(corr, vmin = -1, center = 0, annot = True, cmap = 'mako')
```



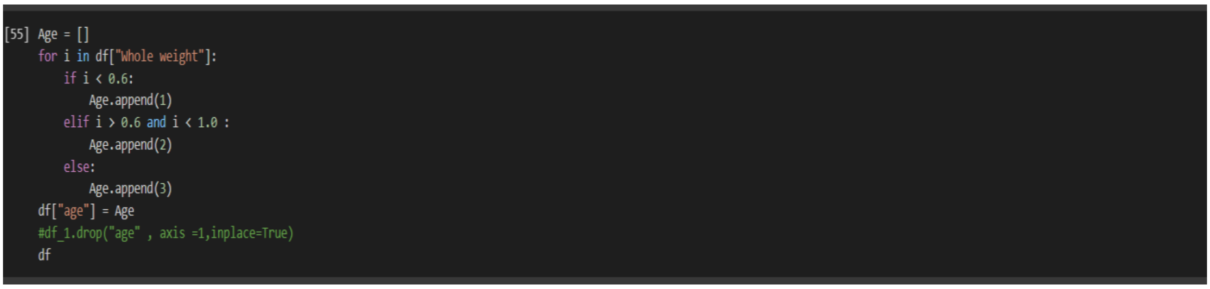
	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	age
Length	1	0.99	0.83	0.93	0.9	0.9	0.9	0.56
Diameter	0.99	1	0.83	0.93	0.89	0.9	0.91	0.57
Height	0.83	0.83	1	0.82	0.77	0.8	0.82	0.56
Whole weight	0.93	0.93	0.82	1	0.97	0.97	0.96	0.54
Shucked weight	0.9	0.89	0.77	0.97	1	0.93	0.88	0.42
Viscera weight	0.9	0.9	0.8	0.97	0.93	1	0.91	0.5
Shell weight	0.9	0.91	0.82	0.96	0.88	0.91	1	0.63
age	0.56	0.57	0.56	0.54	0.42	0.5	0.63	1



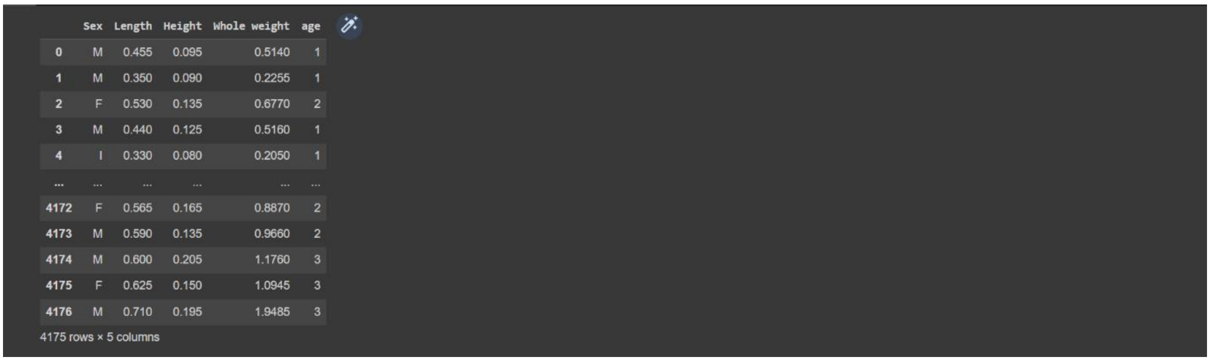
```
[35] #No Negative correlation found
     #High coorelation between Length & Diameter
     #High corelation between shucked weight, viscera weight Vs Whole_weight & Shell weight vs Whole_weight
     #highly correlated variables to be removed.

     #We will remove the columns, before proceeding any further.



     columns_to_drop=['Diameter', 'Shucked weight', 'Viscera weight', 'Shell weight']
     df.drop(columns_to_drop, axis=1, inplace = True)
```

```
[36] df.head()
```

```
   Sex  Length  Height  Whole weight   age
0    M   0.455   0.095        0.5140  16.5
1    M   0.350   0.090        0.2255   8.5
2    F   0.530   0.135        0.6770  10.5
3    M   0.440   0.125        0.5160  11.5
4    I   0.330   0.080        0.2050   8.5
```

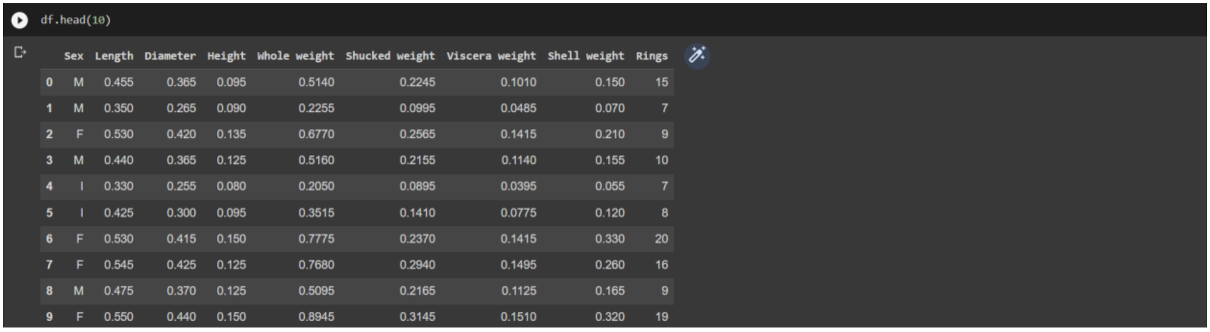


```
[55] Age = []
     for i in df["Whole weight"]:
         if i < 0.6:
             Age.append(1)
         elif i > 0.6 and i < 1.0 :
             Age.append(2)
         else:
             Age.append(3)
     df["age"] = Age
     #df_1.drop("age" , axis =1,inplace=True)
     df
```



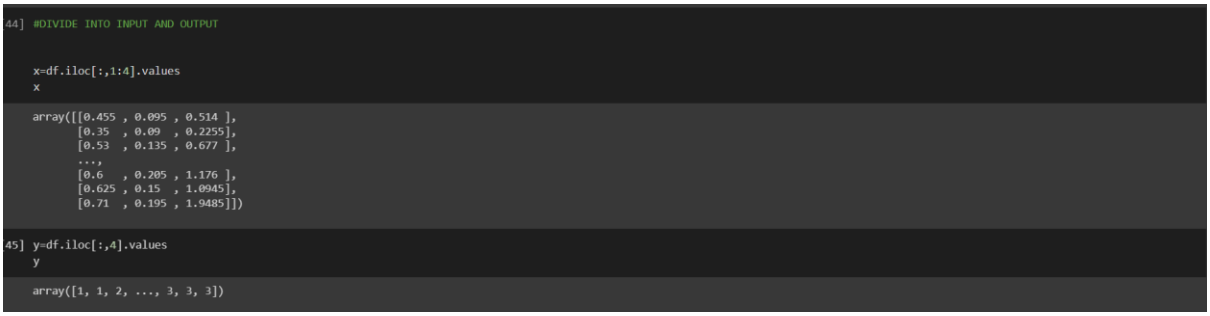
	Sex	Length	Height	Whole weight	age
0	M	0.455	0.095	0.5140	1
1	M	0.350	0.090	0.2255	1
2	F	0.530	0.135	0.6770	2
3	M	0.440	0.125	0.5160	1
4	I	0.330	0.080	0.2050	1
...	...	...	...	...	...
4172	F	0.565	0.165	0.8870	2
4173	M	0.590	0.135	0.9660	2
4174	M	0.600	0.205	1.1760	3
4175	F	0.625	0.150	1.0945	3
4176	M	0.710	0.195	1.9485	3

4175 rows × 5 columns

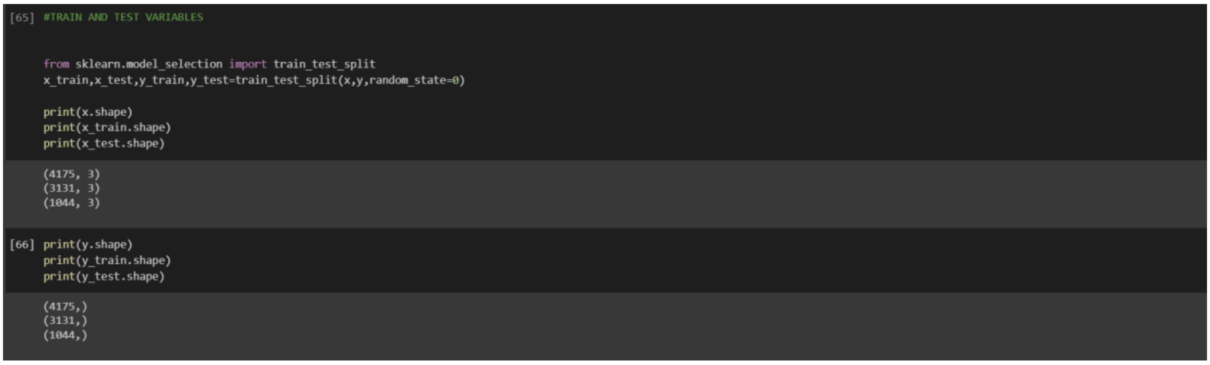


```
df.head(10)
```

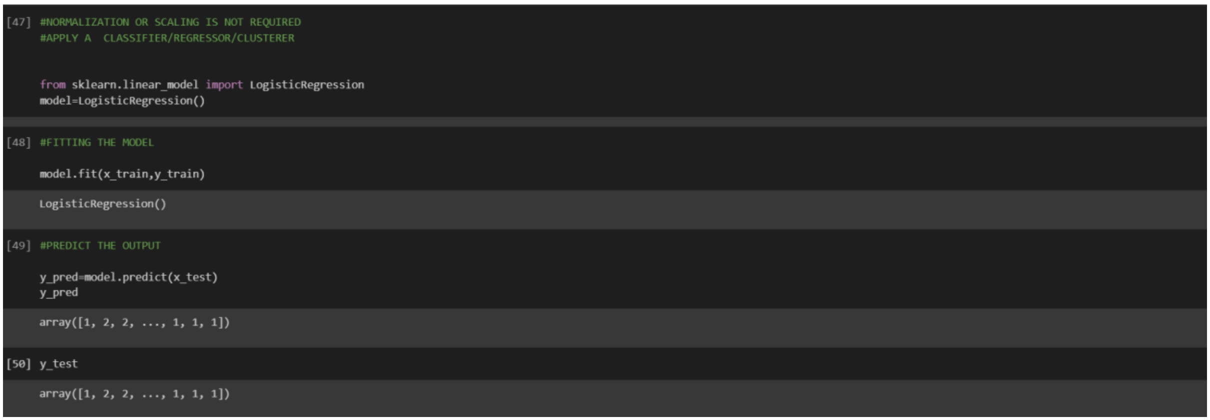
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7
5	I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8
6	F	0.530	0.415	0.150	0.7775	0.2370	0.1415	0.330	20
7	F	0.545	0.425	0.125	0.7680	0.2940	0.1495	0.260	16
8	M	0.475	0.370	0.125	0.5095	0.2165	0.1125	0.165	9
9	F	0.550	0.440	0.150	0.8945	0.3145	0.1510	0.320	19



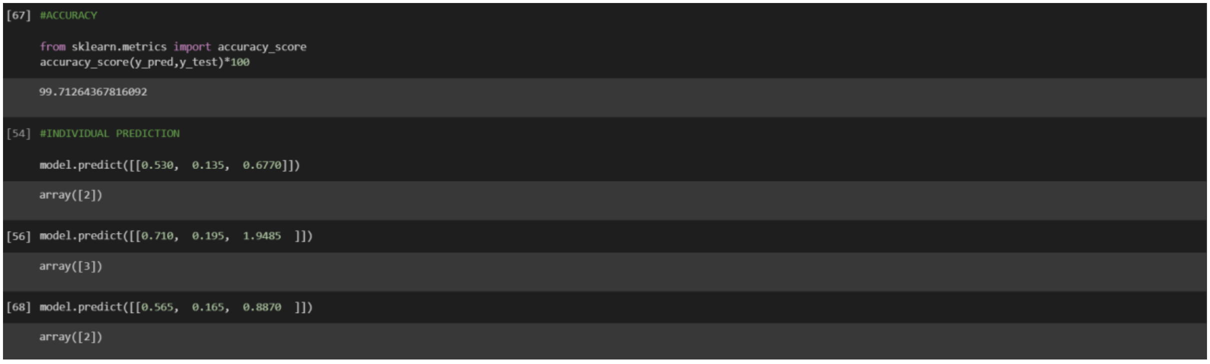
```
44] #DIVIDE INTO INPUT AND OUTPUT


    x=df.iloc[:,1:4].values
    x

    array([[0.455 , 0.095 , 0.514 ],
           [0.35  , 0.09  , 0.2255],
           [0.53  , 0.135 , 0.677 ],
           ...,
           [0.6   , 0.205 , 1.176 ],
           [0.625 , 0.15  , 1.0945],
           [0.71  , 0.195 , 1.9485]])

45] y=df.iloc[:,4].values
    y

    array([1, 1, 2, ..., 3, 3, 3])
```



```
[65] #TRAIN AND TEST VARIABLES


     from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0)

     print(x.shape)
     print(x_train.shape)
     print(x_test.shape)

     (4175, 3)
     (3131, 3)
     (1044, 3)

[66] print(y.shape)
     print(y_train.shape)
     print(y_test.shape)

     (4175,)
     (3131,)
     (1044,)
```



```
[47] #NORMALIZATION OR SCALING IS NOT REQUIRED
     #APPLY A  CLASSIFIER/REGRESSOR/CLUSTERER


     from sklearn.linear_model import LogisticRegression
     model=LogisticRegression()

[48] #FITTING THE MODEL

     model.fit(x_train,y_train)

     LogisticRegression()

[49] #PREDICT THE OUTPUT

     y_pred=model.predict(x_test)
     y_pred

     array([1, 2, 2, ..., 1, 1, 1])

[50] y_test

     array([1, 2, 2, ..., 1, 1, 1])
```



```
[67] #ACCURACY

     from sklearn.metrics import accuracy_score
     accuracy_score(y_pred,y_test)*100

     99.71264367816092

[54] #INDIVIDUAL PREDICTION

     model.predict([[0.530,  0.135,  0.6770]])

     array([2])

[56] model.predict([[0.710,  0.195,  1.9485  ]])

     array([3])

[68] model.predict([[0.565,  0.165,  0.8870  ]])

     array([2])
```