GROUP -11

ABALONE AGE PREDICTION

PV.TILAK

M.DHEERAJ

K.SHIVA KUMAR

MANIKETHAN REDDY

DESCRIPTION -

Scientific studies on abalones require knowing the age of an abalone. It involves measuring the number of layers of shell (“rings”) that make up the abalone’s shell. This is done by taking a sample of shell, staining it and counting the number of rings under the microscope.

ABALONE SHELL

OBJECTIVE -

Predicting the age of abalone from physical measurements.

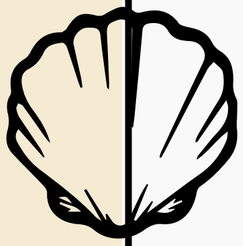
As we know, Age is a number and we have data that contains the physical measurements of abalones and their ages.

PATH -

We followed numerous ML algorithms to fit a predictive model to this data in order to determine the age of abalone.

HOW MACHINE LEARNING CAN SOLVE THIS PROBLEM?

As we know, Age is a number and we have data that contains the physical measurements of abalones and their ages. We can build a machine learning model that can predict the age of abalone given its physical measurements like weight, height, etc.



VARIABLE DISCRIPTION

THE DATA CONSISTS OF 9 FEATURES:

GENDER - (M-MALE, F-FEMALE, I-INFANT)

LENGTH -(LONGEST SHELL MEASUREMENT)

DIAMETER -(PERPENDICULAR TO LENGTH)

HEIGHT -(WITH MEAT IN SHELL)

WHOLE WEIGHT -(WHOLE ABALONE)

SHUCKED WEIGHT -(WEIGHT OF MEAT)

VISCERA WEIGHT -(GUT WEIGHT AFTER BLEEDING)

SHELL WEIGHT -(AFTER BEING DRIED)

RINGS - (+1.5 GIVES THE AGE IN YEARS)

DATA CLEANSING

CHECKING FOR NULL VALUES

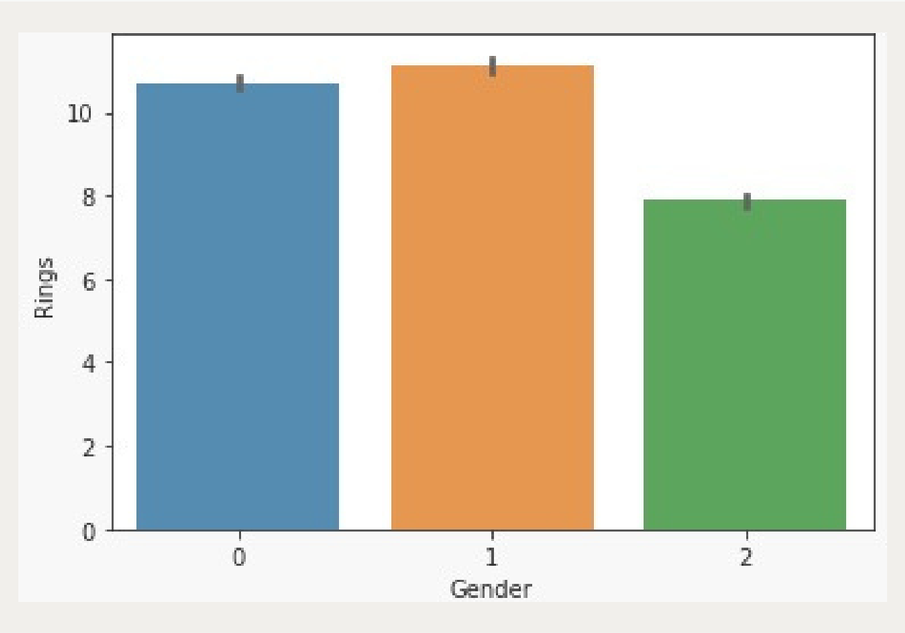
```
data.isnull().sum()

Gender            0
Length            0
Diameter          0
Height            0
Whole_weight      0
Shucked_weight    0
Viscera_weight    0
Shell_weight      0
Rings             0
Age               0
Gender_g          0
dtype: int64
```

WE INTRODUCED
A NEW COLUMN
"AGE"= RINGS +1.5

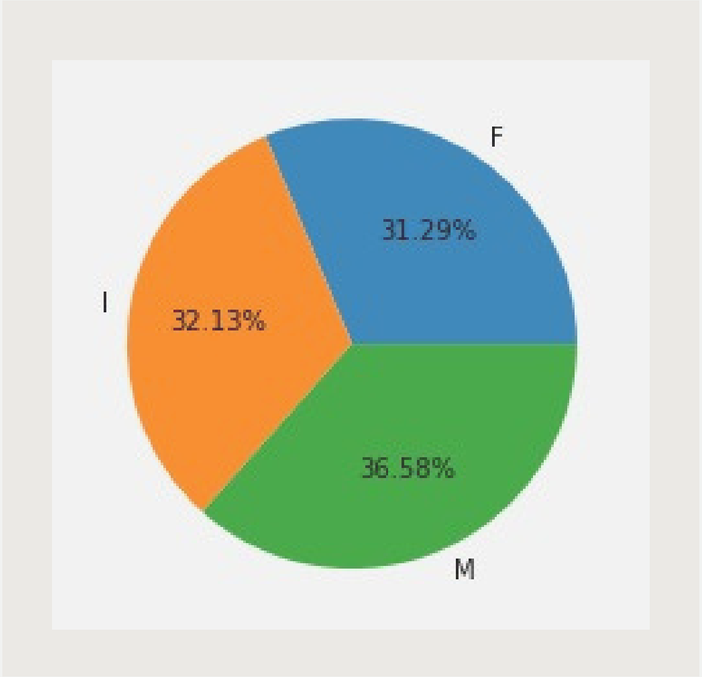
	Gender	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight	Rings	Age	Gender_g
0	0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15	16.5	M
1	0	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7	8.5	M
2	1	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9	10.5	F
3	0	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10	11.5	M
4	2	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7	8.5	I
...	...	...	...	...	...	...	...	...	...	...	...
4172	1	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11	12.5	F
4173	0	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10	11.5	M
4174	0	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9	10.5	M
4175	1	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10	11.5	F
4176	0	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12	13.5	M

EXPLORATORY DATA ANALYSIS



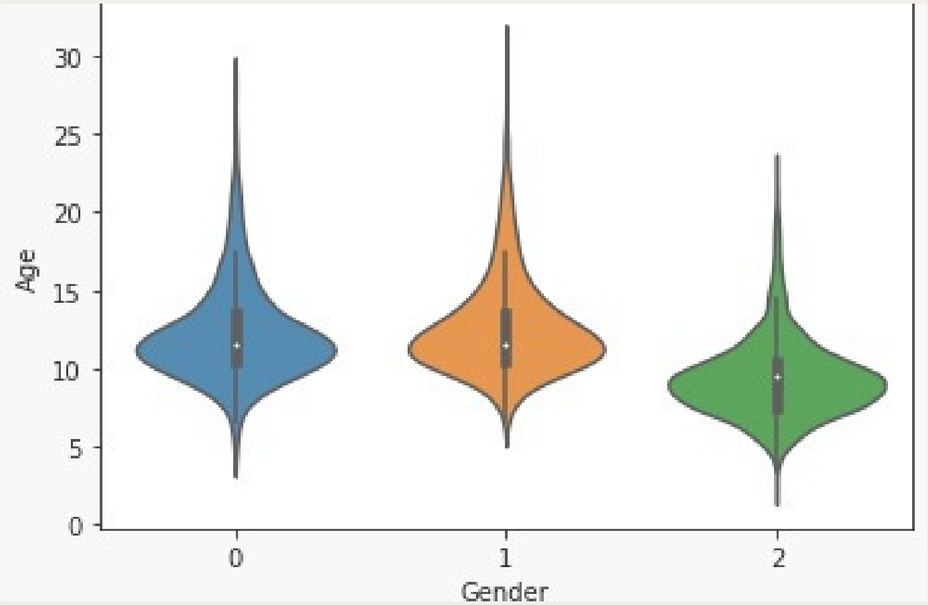
Gender and Rings

Female has more number of rings then Male and Infant



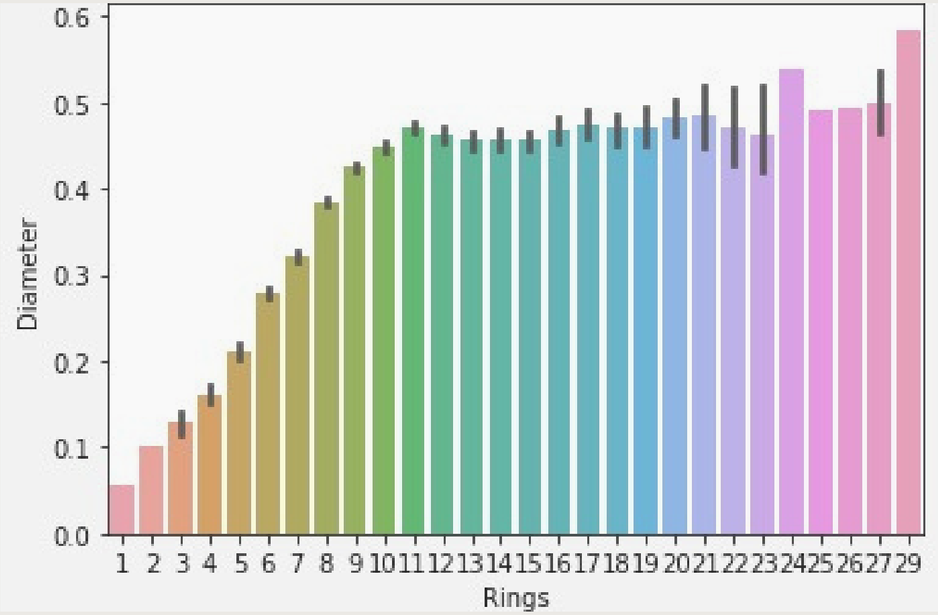
Gender Count

Male Count is more than Female and Infant



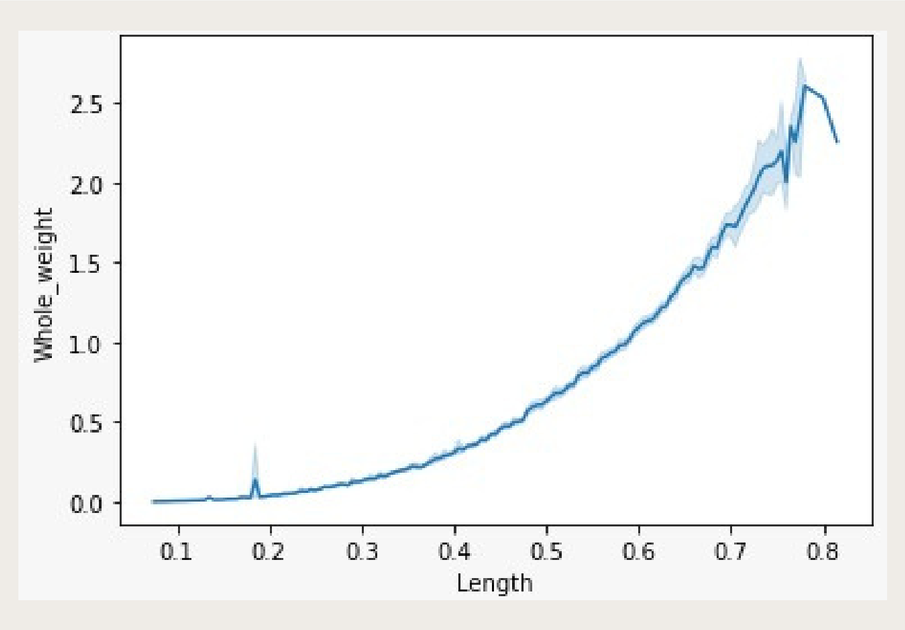
Gender and Age

Female Age majority age is more than Male and Infant



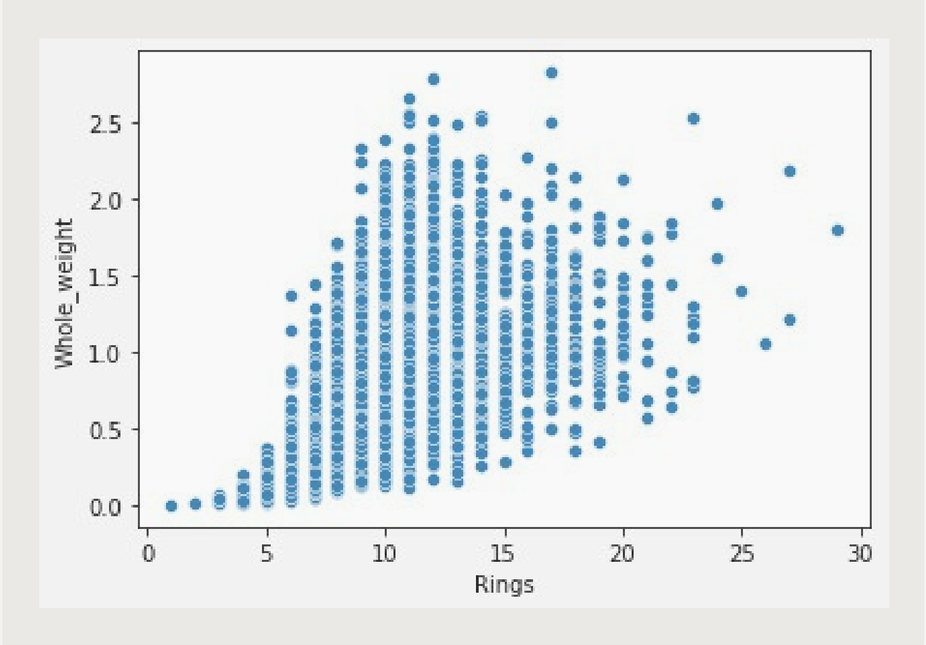
Diameter and Rings

As Diameter increases no.of Rings also increases



Whole weight and Length

As the length increases Whole weight also increases.



Whole weight and Rings

As the Whole weight increases no.of Rings increases .

LINEAR REGRESSION

Model	Train-Test Ratio	MAE
1.	60-40	1.5895
2.	75-25	1.6555
3.	70-30	1.6166
4.	80-20	1.5732

- 80-20 Is giving a less MAE value compared to other train test splits
- The least MAE we got is 1.5732

NEURAL NETWORK

Train Test Proportion	Architecture	Optimizer	Epochs	MAE
80-20	7-5--1	Adam	10	1.7864
80-20	7-5--1	Adam	50	1.5956
80-20	7-5--1	Adam	100	1.5938
80-20	7-5--1	Adam	500	1.5757
80-20	7-5--1	Adam	1000	1.5646
80-20	7-4--1	Adam	10	1.8161
80-20	7-4--1	Adam	50	1.596
80-20	7-4--1	Adam	100	1.5988
80-20	7-4--1	Adam	500	1.578
80-20	7-4--1	Adam	1000	1.5732
80-20	6-5--1	Adam	10	1.7834
80-20	6-5--1	Adam	50	1.6033
80-20	6-5--1	Adam	100	1.6017
80-20	6-5--1	Adam	500	1.5781
80-20	6-5--1	Adam	1000	1.5712
80-20	7-5--1	SGD	100	1.6055
80-20	7-5--1	SGD	500	1.5725

60-40	7-5--1	Adam	10	1.9232
60-40	7-5--1	Adam	50	1.5481
60-40	7-5--1	Adam	100	1.5486
60-40	7-5--1	Adam	500	1.5445
60-40	7-5--1	Adam	1000	1.5387
60-40	7-4--1	Adam	10	1.9022
60-40	7-4--1	Adam	50	1.5639
60-40	7-4--1	Adam	100	1.5531
60-40	7-4--1	Adam	500	1.5457
60-40	7-4--1	Adam	1000	1.5396
60-40	6-5--1	Adam	10	1.8457
60-40	6-5--1	Adam	50	1.5621
60-40	6-5--1	Adam	100	1.5507
60-40	6-5--1	Adam	500	1.5459
60-40	6-5--1	Adam	1000	1.54
60-40	7-5--1	SGD	100	1.6124
60-40	7-5--1	SGD	500	1.5441

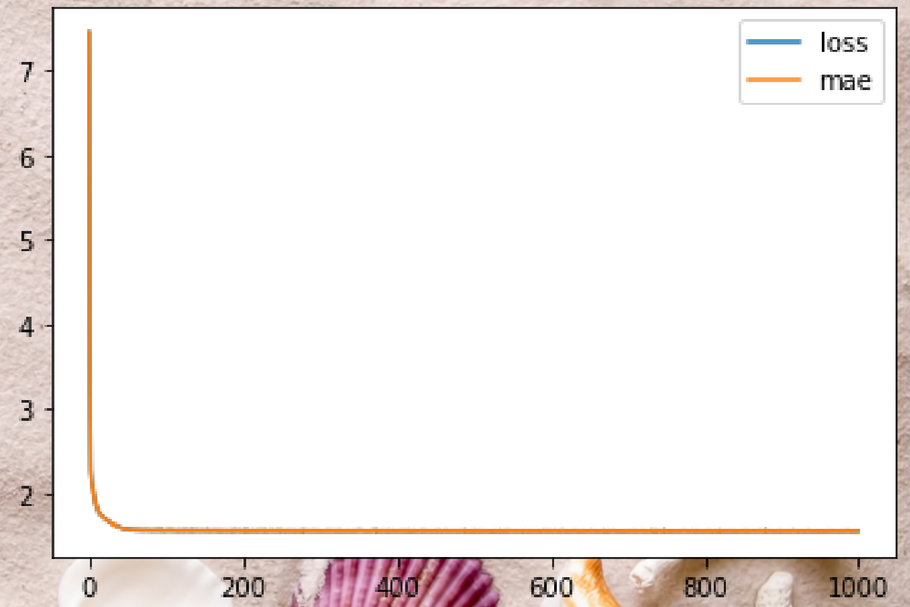
75-25	7-5--1	Adam	10	1.7285
75-25	7-5--1	Adam	50	1.5632
75-25	7-5--1	Adam	100	1.5523
75-25	7-5--1	Adam	500	1.5646
75-25	7-5--1	Adam	1000	1.536
75-25	7-4--1	Adam	10	1.7032
75-25	7-4--1	Adam	50	1.5632
75-25	7-4--1	Adam	100	1.5544
75-25	7-4--1	Adam	500	1.5645
75-25	7-4--1	Adam	1000	1.5342
75-25	6-5--1	Adam	10	1.8284
75-25	6-5--1	Adam	50	1.545
75-25	6-5--1	Adam	100	1.5442
75-25	6-5--1	Adam	500	1.5635
75-25	6-5--1	Adam	1000	1.5338
75-25	7-5--1	SGD	100	1.538
75-25	7-5--1	SGD	500	1.5871

70-30	7-5--1	Adam	10	1.9243
70-30	7-5--1	Adam	50	1.5607
70-30	7-5--1	Adam	100	1.5561
70-30	7-5--1	Adam	500	1.5448
70-30	7-5--1	Adam	1000	1.5382
70-30	7-4--1	Adam	10	1.83
70-30	7-4--1	Adam	50	1.5568
70-30	7-4--1	Adam	100	1.5505
70-30	7-4--1	Adam	500	1.5486
70-30	7-4--1	Adam	1000	1.543
70-30	6-5--1	Adam	10	1.7289
70-30	6-5--1	Adam	50	1.5591
70-30	6-5--1	Adam	100	1.5457
70-30	6-5--1	Adam	500	1.5428
70-30	6-5--1	Adam	1000	1.5389
70-30	7-5--1	SGD	100	1.5978
70-30	7-5--1	SGD	500	1.5448

80-20	7-3--1	Adam	100	1.5908
80-20	7-3--1	Adam	500	1.5709
80-20	7-3--1	Adam	1000	1.572
80-20	7-5-3--1	Adam	100	1.6023
80-20	7-5-3--1	Adam	500	1.579
80-20	7-5-3--1	Adam	1000	1.5726
75-25	7-3--1	Adam	100	1.5452
75-25	7-3--1	Adam	500	1.5676
75-25	7-3--1	Adam	1000	1.5348
75-25	7-5-3--1	Adam	100	1.553
75-25	7-5-3--1	Adam	500	1.5672
75-25	7-5-3--1	Adam	1000	1.5339
75-25	7-5-3-2-1	Adam	100	1.5427
75-25	7-5-3-2-1	Adam	500	1.5692
75-25	7-5-3-2-1	Adam	1000	1.5292
75-25	7-5-3-2-1	SGD	1000	1.5306

As per the tables, we can see that all the MAE values are mostly in the range of 1.5 to 1.7

Adam as optimizer is giving the best result than SGD optimizer.



BAGGING

Train-Test Ratio	MAE
60-40	2.4843
70-30	2.4842
75-25	2.5726
80-20	2.5371

- 70-30 Is giving a less MAE value compared to other train test splits
- The least MAE we got is 2.4842

ADABOOSTING

Train-Test Ratio	MAE
60-40	2.5080
70-30	2.4091
75-25	2.4804
80-20	2.6769

- 70-30 Is giving a less MAE value compared to other train test splits
- The least MAE we got is 2.4091

GRADIENT BOOSTING

Train-Test Ratio	MAE
60-40	1.6569
70-30	1.6479
75-25	1.6181
80-20	1.6755

- 75-25 Is giving a less MAE value compared to other train test splits
- The least MAE we got is 1.6181

EXTREME GRADIENT BOOSTING

Train-Test Ratio	MAE
60-40	1.5493
70-30	1.5416
75-25	1.5396
80-20	1.5702

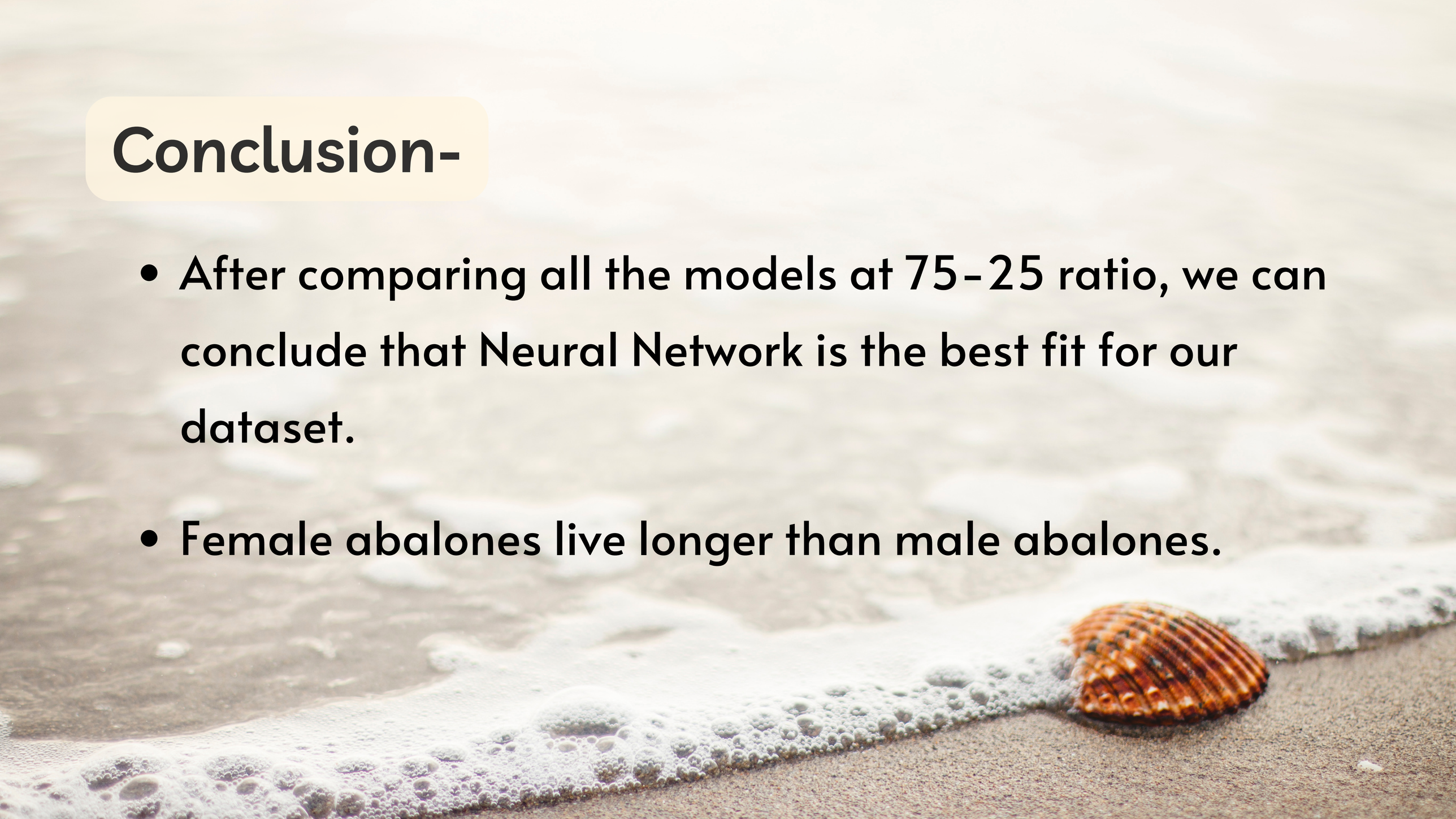
- 75-25 Is giving a less MAE value compared to other train test splits
- The least MAE we got is 1.5396

SUMMARY

Linear Regression	1.6555
Neural Network	1.5292
ADA Boost	2.4804
Gradient Boost	1.6181
XG Boost	1.5396
Bagging	2.5726

Conclusion-

- After comparing all the models at 75-25 ratio, we can conclude that Neural Network is the best fit for our dataset.
- Female abalones live longer than male abalones.




Thank you...

APPENDIX

LINEAR REGRESSION

```
[ ] LR6 = smf.ols(formula='Rings~Diameter+Height+Whole_weight+Shucked_weight+Viscera_weight+Shell_weight+Gender', data=data).fit()
    LR6.params
```

```
Intercept         3.714369
Diameter         11.054384
Height           11.183837
Whole_weight      9.074323
Shucked_weight  -20.135833
Viscera_weight  -10.209289
Shell_weight      8.717142
Gender           -0.389536
dtype: float64
```

```
[ ] LR6.summary()
```

OLS Regression Results

Dep. Variable:	Rings	**R-squared:**	0.535
Model:	OLS	**Adj. R-squared:**	0.535
Method:	Least Squares	**F-statistic:**	686.1
Date:	Tue, 13 Dec 2022	**Prob (F-statistic):**	0.00
Time:	14:28:58	**Log-Likelihood:**	-9215.7
No. Observations:	4177	**AIC:**	1.845e+04
Df Residuals:	4169	**BIC:**	1.850e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3.7144	0.265	14.017	0.000	3.195	4.234
Diameter	11.0544	0.986	11.208	0.000	9.121	12.988
Height	11.1838	1.537	7.278	0.000	8.171	14.196
Whole_weight	9.0743	0.727	12.483	0.000	7.649	10.499
Shucked_weight	-20.1358	0.814	-24.734	0.000	-21.732	-18.540
Viscera_weight	-10.2093	1.288	-7.924	0.000	-12.735	-7.683
Shell_weight	8.7171	1.127	7.735	0.000	6.508	10.927
Gender	-0.3895	0.047	-8.352	0.000	-0.481	-0.298

Omnibus:	959.512	**Durbin-Watson:**	1.422
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2768.195
Skew:	1.193	**Prob(JB):**	0.00
Kurtosis:	6.196	**Cond. No.**	86.6

```
[ ] X = data[["Gender","Diameter","Whole_weight","Shucked_weight","Viscera_weight","Shell_weight","Length"]]
    y = data.Rings
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    lm1 = LinearRegression()
    lm1.fit(X_train, y_train)
    y_pred1 = lm1.predict(X_test)
    print(np.sqrt(metrics.mean_squared_error(y_test, y_pred1)))
```

```
2.183963129575756
```

```
[ ] mae = metrics.mean_absolute_error(y_test, y_pred1)
    mse = metrics.mean_squared_error(y_test , y_pred1)
    rmse = np.sqrt(mse)
    r2 = metrics.r2_score(y_test, y_pred1)
    print(mae)
    print(mse)
    print(rmse)
    print(r2)
    print(100-(1.96*rmse))
```

```
1.5732914568323197
4.769699495134633
2.183963129575756
0.558886036005825
95.71943226603152
```

NEURAL NETWORK

```
[ ]  tf.random.set_seed(42)

     # STEP1: Creating the model

     model= tf.keras.Sequential([
                                        tf.keras.layers.Dense(7),
                                        tf.keras.layers.Dense(3),
                                        tf.keras.layers.Dense(1)
     ])

     # STEP2: Compiling the model   # optimizer casn be SGD, Adam

     model.compile(loss= tf.keras.losses.mae,
                   optimizer= tf.keras.optimizers.SGD(),
                   metrics= ["mae"])

     # STEP3: Fit the model

     history= model.fit(x_train, y_train, epochs= 1250, verbose=0)
```

```
[ ]  model.evaluate(x_test, y_test)
```

```
33/33 [==============================] - 0s 803us/step - loss: 1.5351 - mae: 1.5351
[1.5350645780563354, 1.5350645780563354]
```

BAGGING

```
[ ]  from sklearn import datasets
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score
     from sklearn.model_selection import cross_val_score
     from sklearn.ensemble import BaggingRegressor
```

```
[ ]  x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.25, random_state=42)
```

```
[ ]  bgr = BaggingRegressor(n_estimators=90, random_state=1)
     bgr.fit(x_train, y_train)
```

```
BaggingRegressor(n_estimators=90, random_state=1)
```

```
[ ]  bgr.score(x_test, y_test)
```

```
0.5339855247464264
```

```
[ ]  mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
     print(mae)
```

```
2.572611431215284
```

BOOSTING

ADAPTIVE BOOSTING

```
[ ] from sklearn.ensemble import AdaBoostRegressor
    from sklearn import datasets
    # Import train_test_split function
    from sklearn.model_selection import train_test_split
    #Import scikit-learn metrics module for accuracy calculation
    from sklearn import metrics
```

```
[ ] x_train, x_test, y_train, y_test= train_test_split(x, y, test_size=0.3, random_state=42)
```

```
[ ] # Create adaboost classifer object
    abc = AdaBoostRegressor(n_estimators=100,
                            learning_rate=0.6)
    # Train Adaboost Classifer
    model = abc.fit(x_train, y_train)
    model.score(x_test, y_test)
```

```
0.13083920047434439
```

```
[ ] mae = metrics.mean_absolute_error(y_test, model.predict(x_test))
    print(mae)
```

```
2.572611431215284
```

GRADIENT BOOSTING

```
[ ] from sklearn.ensemble import GradientBoostingRegressor
```

```
[ ] reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.6,max_depth=1,
      random_state=0).fit(x_train, y_train)
    reg.score(x_test, y_test)
```

```
0.5097606736538218
```

```
[ ] mae = metrics.mean_absolute_error(y_test, reg.predict(x_test))
    print(mae)
```

```
1.6258262419458662
```

XG BOOSTING

```
[ ] from xgboost import XGBRegressor
```

```
[ ] xgbr = XGBRegressor()
    xgbr.fit(x_train, y_train)
```

```
[17:17:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor()
```

```
[ ] mae = metrics.mean_absolute_error(y_test, xgbr.predict(x_test))
    print(mae)
```

```
1.5412572972131877
```

```
[ ] xgbr.score(x_test, y_test)
```

```
0.5315482870861954
```