Abalone Regression and Classification

Nicholas Wong

12 May 2025

Table of Contents

Introduction and Motivation

Abalone is a type of mollusk that has multiple uses for humans. Most typically, it is found in seafood dishes, but shells can also be used in jewelry and decor. They exist in the coastal waters of almost every continent, and are thus often fished and farmed for consumption.

As with most species that are fished and farmed as commonly as the abalone, there are regulations on fishing and harvesting these species. These regulations are set to maintain ecological balance and protect the abalone as a species. For example, see this site for New Zealand's restrictions on abalone harvesting: https://www.mpi.govt.nz/fishing-aquaculture/recreational-fishing/fishing-rules-for-gear-methods-and-species/paua-rules-and-guidelines/

It is important that young abalone that have not yet reproduced be left in their natural habitat so that the abalone population is not negatively affected by excessive human interference. However, determining abalone age is time-consuming, as it involves dying the shell and counting the rings under a microscope. The abalone dataset provided by the UCI Machine Learning Repository provides features used that can help predict abalone age from measurements that are more easily gatherable.

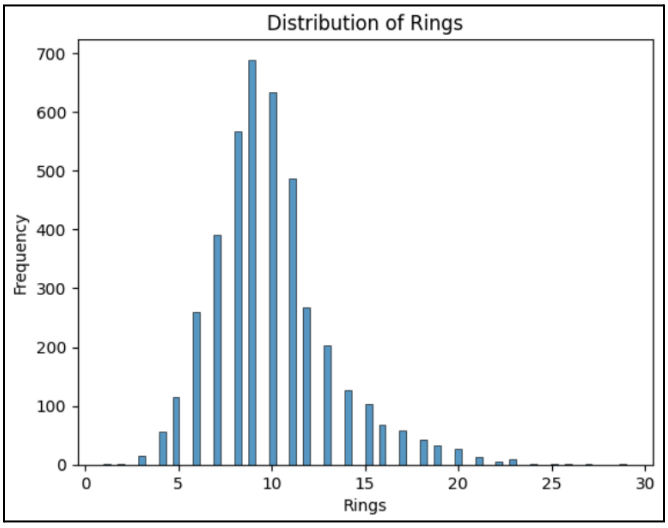
My goal with this project is to first identify the most relevant features in determining abalone age and building an accurate model. Then, using the features we identified and multiple classification methods, I will see how accurately we can distinguish young from mature abalone.

Dataset

The abalone dataset was provided in 1995. With 4177 observations, it contains eight feature variables (sex, length, diameter, width, whole weight, shucked weight, viscera weight, shell weight) and one target variable (rings). Sex is the only categorical variable (male, female, infant) and age of an abalone is given by its number of rings + 1.5. Length, diameter, and height are given in millimeters (mm), and the weight variables are given in grams (g). All of the numerical feature values are continuous and the target variable of rings is discrete.

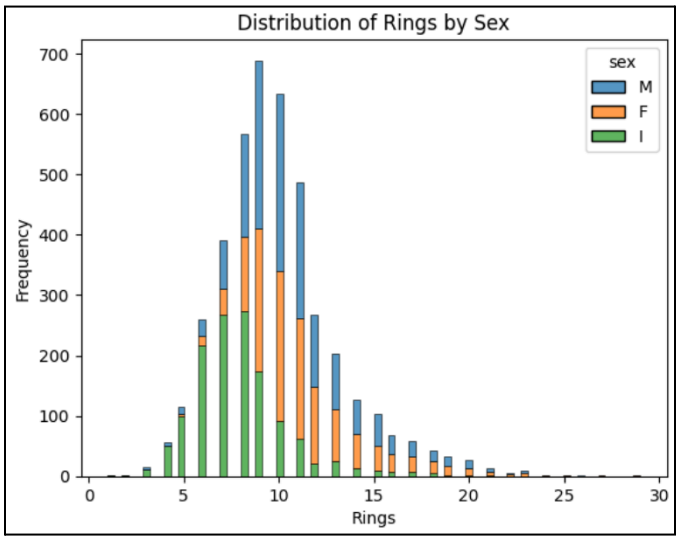
There are no missing or null values in this dataset. Some limitations may include the lack of a location variable. Though it is not necessary for this dataset as it is assumed that the abalone are from the same location, there may be a missed opportunity to collect abalone from different locations globally and find how an abalone's habitat affects its age and ring growth.

Below is the distribution of rings found on the abalone:



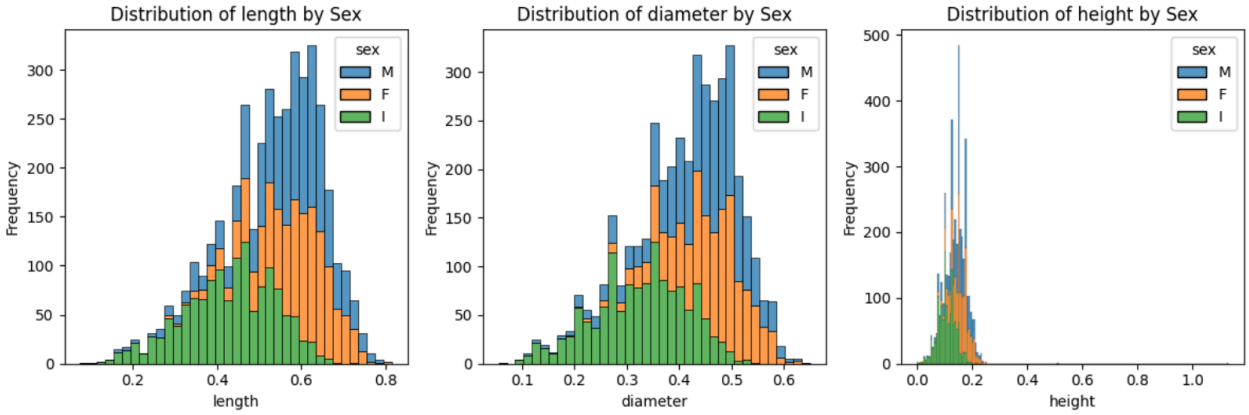
Abalone can live to be several decades old, but few exist that long before being caught, so this right-tailed distribution makes sense. The median abalone has 9 rings, with the first quartile at 8 rings, and the third quartile at 11 rings. This tells us that the majority of the abalone in our dataset sit between the short range of 8 to 11 rings (9.5 to 12.5 years old).

Then here is the distribution of abalone rings given sex:



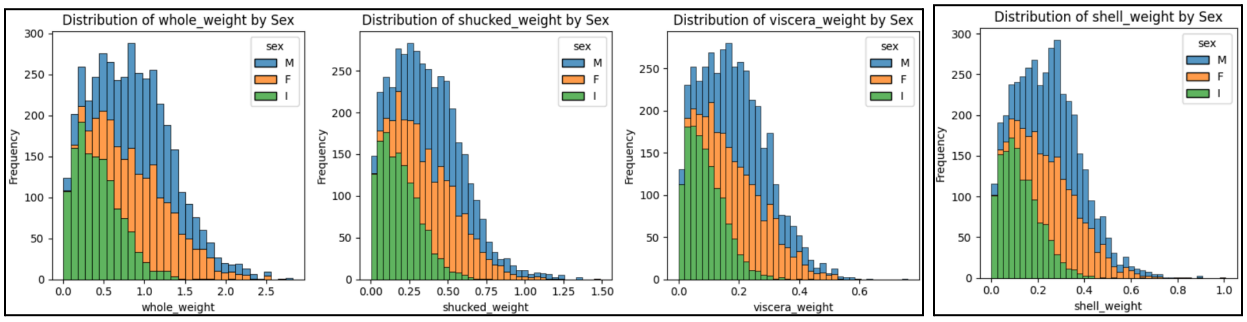
The spread of rings across male and female abalone seems to be similar. Infant abalone, as the category suggests, has a lower mean and median ring count, falling closer to 7 rings. This is useful information because it suggests that the dimensions of an abalone represent its age rather than any possible variations provided by gender.

Below are the size dimensions given sex:



The size dimensions tend to follow a similar trend to each other. Height appears to have outlying abalone observations, even suggesting possible improper measurement. But abalone, which have a generally oval or ear shape, are expected to grow their dimensions proportionally. For example, one would not expect to find a particularly long abalone if it was not also wide. Therefore, we have reason to investigate multicollinearity between the size features.

Below are the weight dimensions given sex:



In a similar idea to the size dimensions, the weight dimensions also follow the same distribution. Given that one of the features is total weight, it is likely that we will find multicollinearity between these weight features.

Investigating Multicollinearity for Feature Selection

Originally, I ran tests for multicollinearity in R, as it is really simple to import the "car" package and retrieve variance inflation factors (VIFs) of features. The VIFs as provided by R are as follows:

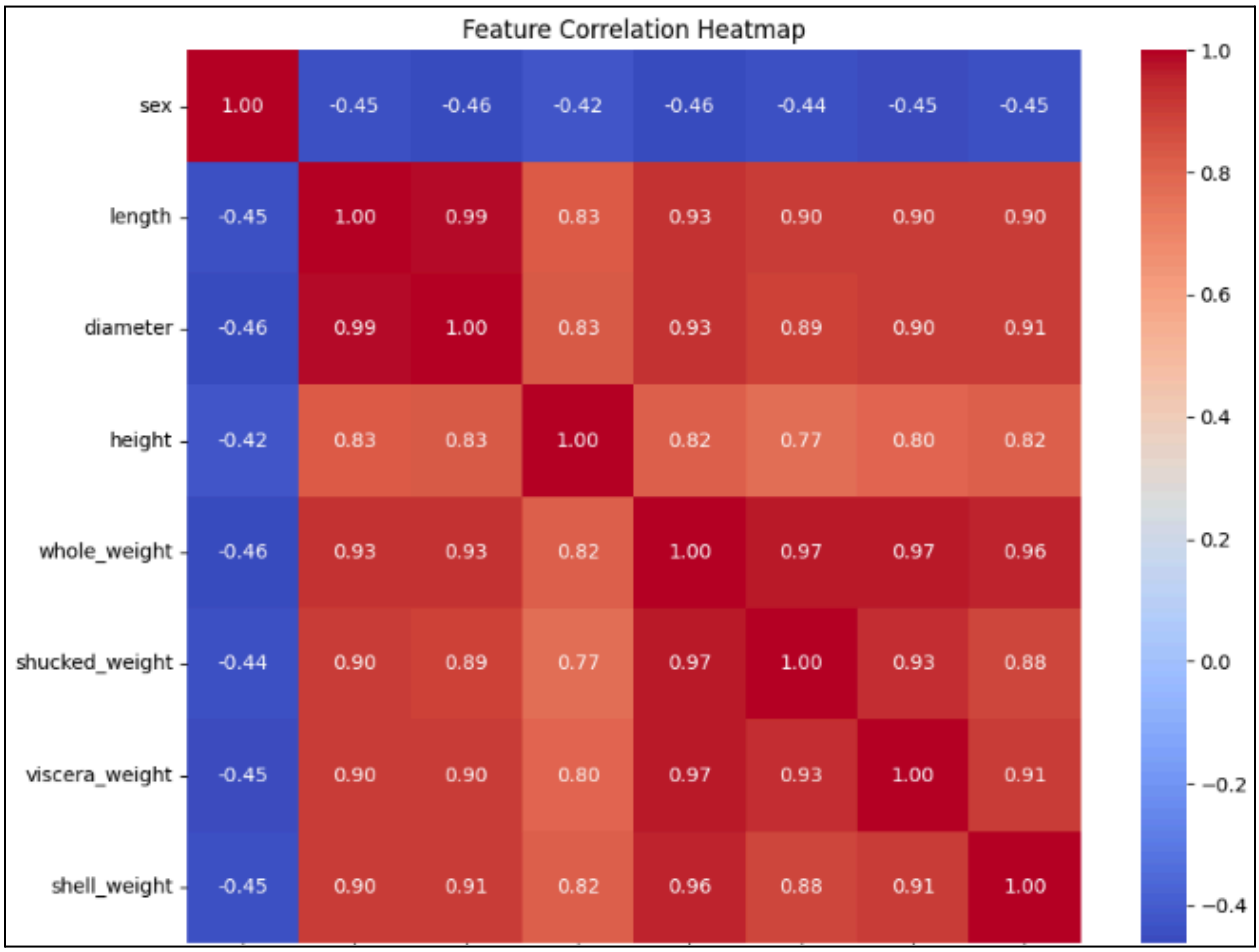
	GVIF	Df	GVIF^(1/(2*Df))
Sex	1.543449	2	1.114610
Length	40.945763	1	6.398888
Diameter	42.379841	1	6.509980
Height	3.581369	1	1.892450
Whole.weight	109.768710	1	10.477056
Shucked.weight	28.550546	1	5.343271
Viscera.weight	17.445012	1	4.176723
Shell.weight	21.263272	1	4.611212

From this output, we see that length, diameter, whole weight, and shucked weight, surpass the common threshold of 5 when looking for multicollinearity, and viscera weight and shell weight are close to the threshold. This confirms our suspicions of multicollinearity, but then I tried to replicate my VIF findings in Python, which returned:

	feature	VIF
0	const	68.981878
1	sex	1.289614
2	length	40.871761
3	diameter	42.085082
4	height	3.568367
5	whole_weight	109.686915
6	shucked_weight	28.360326
7	viscera_weight	17.362142
8	shell_weight	21.262056

I was unable to find out why the VIF values given by Python were much larger than those in R. In fact, they were even larger before I added a constant to the fit. However, we notice that the general trend follows the VIFs from R, so this will motivate our selection and removal of features from the whole model.

I also used a correlation heatmap to visualize the multicollinearity.



The presence of high multicollinearity indicates that some variables must be removed or changed in order to estimate the true model.

Linear Regression Process and Findings

Though it is clear that we have multicollinearity between several of our feature variables, I decided to still run a linear regression with all of the feature variables in the model to compare our future revised model(s) with.

I will strictly discuss my findings and comparisons with test results and without showing code as that can be found in the Jupyter notebook.

The full model's performance was evaluated as follows:

```
Whole model training RMSE: 2.1941215591356333
Whole model training R2: 0.5433882652503234
Whole model testing RMSE: 2.183479244868033
Whole model testing R2: 0.5130865452704279
```

Overall, we see indication of moderate correlation between our feature variables and ring count.

Considering the variables flagged for multicollinearity, I decided to revise the model to include a new feature: volume. Volume was estimated as the product of length, diameter, and height. I then removed those three features, as well as shucked weight, viscera weight, and shell weight, as those were simply parts of whole weight. This much simpler model returned the following evaluation:

```
Revised model training RMSE: 2.6534565945761157
Revised model training R2: 0.332194987413829
Revised model testing RMSE: 2.6059294159606026
Revised model testing R2: 0.30644814793782915
```

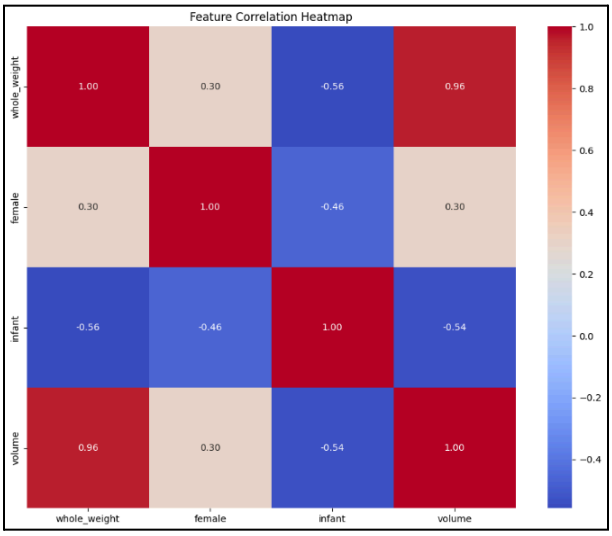
While RMSE increased and R^2 significantly decreased, I consider this simpler model to be a closer estimation of the true model than the full model, as the full model's severe multicollinearity would inflate the R^2 score by a significant amount.

Finally, I wanted to transform the variables of the revised model, including target variable rings, to account for skewness. Using a log-transformation on variables with heavy skewness (rings), and square root transformation on variables with moderate skewness (volume and whole weight), the transformed model achieved the following results:

```
Transformed model training RMSE: 0.21185052553750316
Transformed model training R2: 0.45774259221723357
Transformed model testing RMSE: 0.21118058986006397
Transformed model testing R2: 0.43358538852749007
```

The transformations significantly improved our model's ability to explain the variance.

Additionally, this model removed most of the multicollinearity:



Classification Process and Findings

Now that I have selected the features that I believe make a closer estimation of the true model, I want to use this model for a research question that has more tangible application to the real world. I want to train the tree-based models to classify between “immature” and “mature” abalone. The goal of this is to be able to identify which abalone are ready for harvesting and which need to be left for ecological conservation. Though different states and countries have different regulations on the size/age of harvestable abalone, I will define abalone with a ring count greater than or equal to 9 as mature and 8 and below as immature. This is simply to reflect the median non-infant abalone ring count as 9 and the mean infant ring count at 7-8.

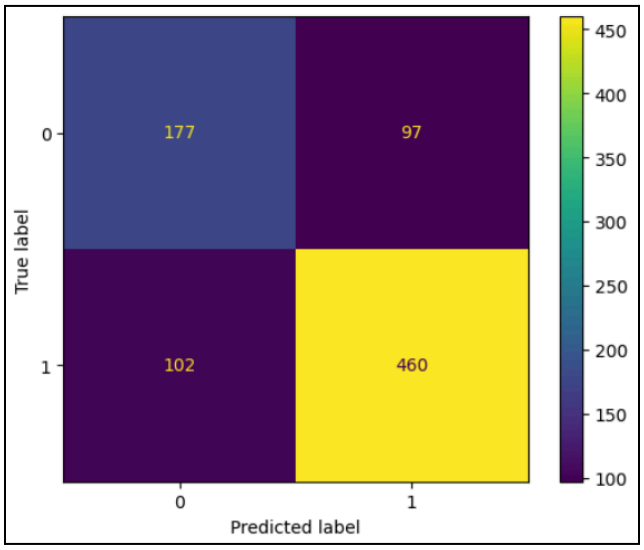
First, I trained a decision tree, random forest, and XGBoost without the use of hyperparameterization tuning. The results are as follows (for reference, the 1 in the binary classification refers to mature abalone, 0 is immature):

Decision Tree:

```
Test Accuracy: 0.7619617224880383
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.65      0.64       274
           1       0.83      0.82      0.82       562

    accuracy                           0.76       836
   macro avg       0.73      0.73      0.73       836
weighted avg       0.76      0.76      0.76       836
```

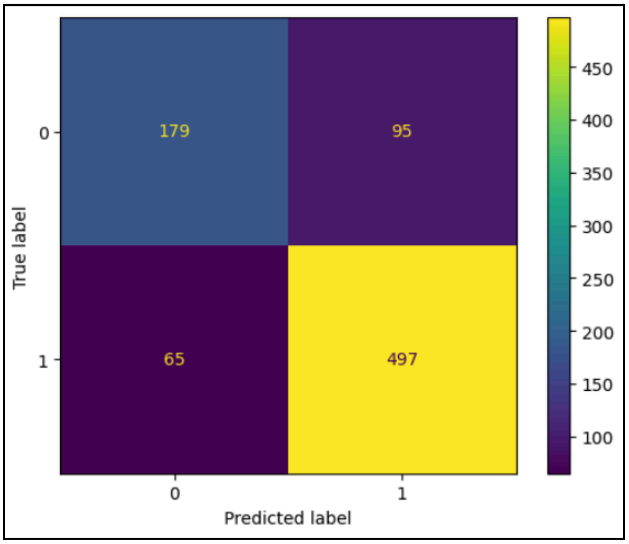


Random forest:

```
Test Accuracy: 0.8086124401913876
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.65      0.69       274
           1       0.84      0.88      0.86       562

    accuracy                           0.81       836
   macro avg       0.79      0.77      0.78       836
weighted avg       0.80      0.81      0.81       836
```

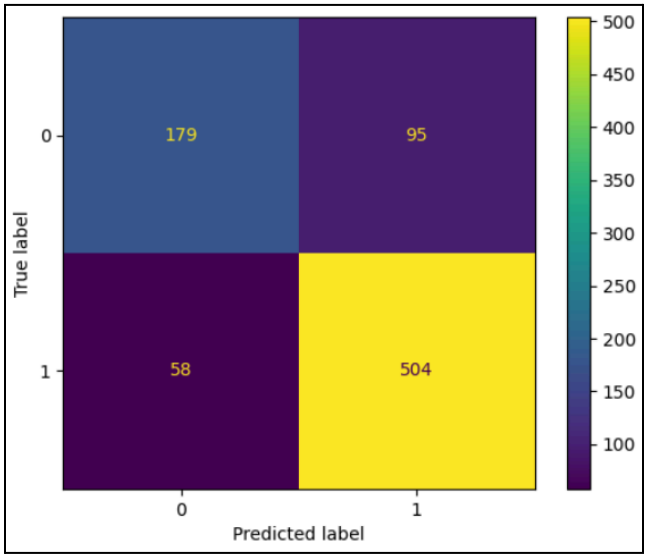


XGBoost:

```
Test Accuracy: 0.8169856459330144
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.65      0.70       274
           1       0.84      0.90      0.87       562

    accuracy                           0.82       836
   macro avg       0.80      0.78      0.78       836
weighted avg       0.81      0.82      0.81       836
```



Overall, the three classifiers perform decently well. Random forest and XGBoost perform similarly well and noticeably better than the decision tree. However, an interesting observation would be that the decision tree's classification errors are evenly spread between false positives and false negatives. In random forest and XGBoost, there are significantly more false positives than false negatives.

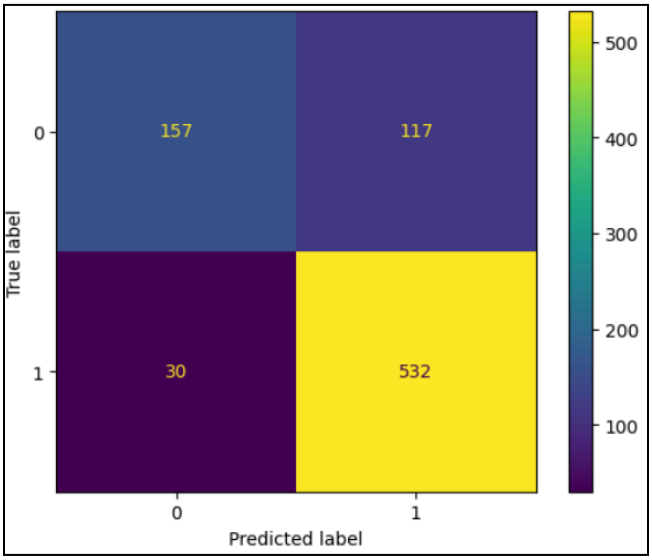
I then tried tuning the hyperparameters using grid search:

Decision tree:

```
Best parameters: {'max_depth': 3, 'min_samples_split': 2}
Test accuracy: 0.8241626794258373
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.57      0.68       274
           1       0.82      0.95      0.88       562

    accuracy                           0.82       836
   macro avg       0.83      0.76      0.78       836
weighted avg       0.83      0.82      0.81       836
```

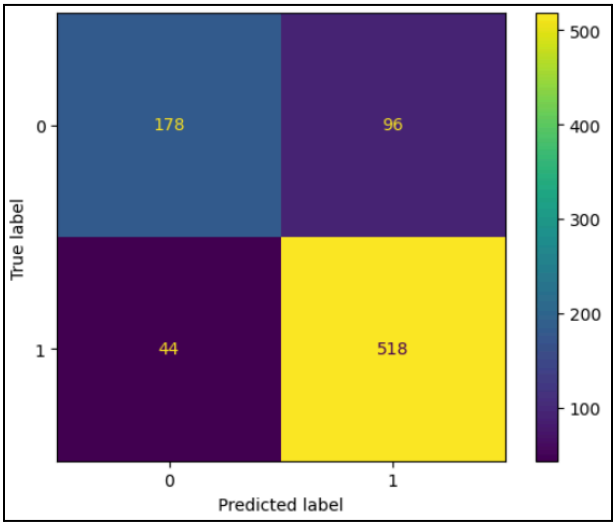


Random forest:

```
Best parameterss: {'max_depth': 5, 'n_estimators': 100}
Test accuracy: 0.8325358851674641
Classification report:
              precision    recall  f1-score   support

           0       0.80      0.65      0.72       274
           1       0.84      0.92      0.88       562

    accuracy                           0.83       836
   macro avg       0.82      0.79      0.80       836
weighted avg       0.83      0.83      0.83       836
```

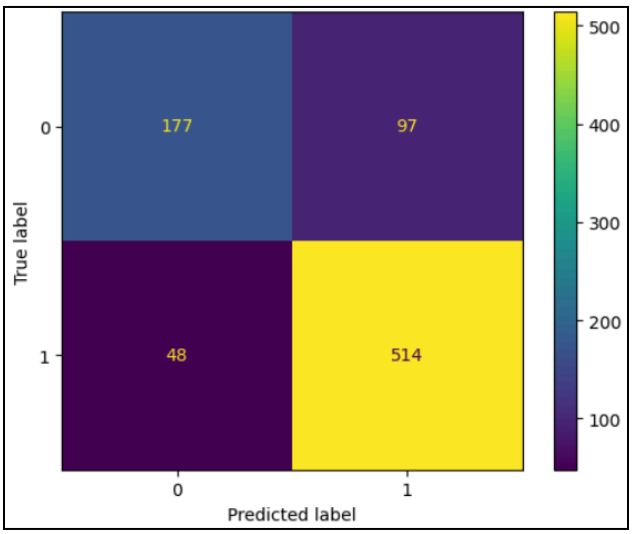


XGBoost:

```
Best parameters: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 50}
Test accuracy: 0.8265550239234449
Classification report:
              precision    recall  f1-score   support

           0       0.79      0.65      0.71       274
           1       0.84      0.91      0.88       562

    accuracy                           0.83       836
   macro avg       0.81      0.78      0.79       836
weighted avg       0.82      0.83      0.82       836
```



The grid search hyperparameter tuning improved all classifiers, though the difference seems negligible for random forest and XGBoost. The hyperparameters tested were chosen simply to cover the general range from simplicity (ex. testing max depth = 2) to complexity (depth = 10).

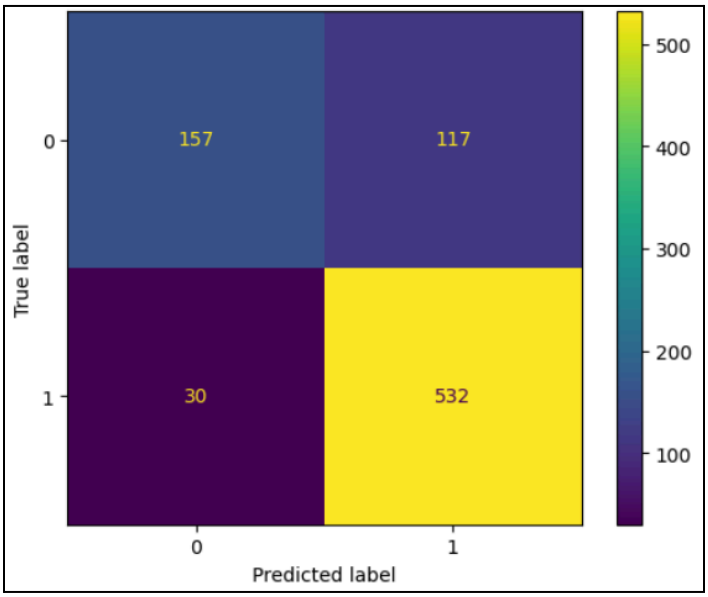
Finally, I tried hyperparameter tuning with random search:

Decision Tree:

```
Best parameters: {'min_samples_split': 2, 'max_depth': 3}
Test accuracy: 0.8241626794258373
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.57      0.68       274
           1       0.82      0.95      0.88       562

    accuracy                           0.82       836
   macro avg       0.83      0.76      0.78       836
weighted avg       0.83      0.82      0.81       836
```

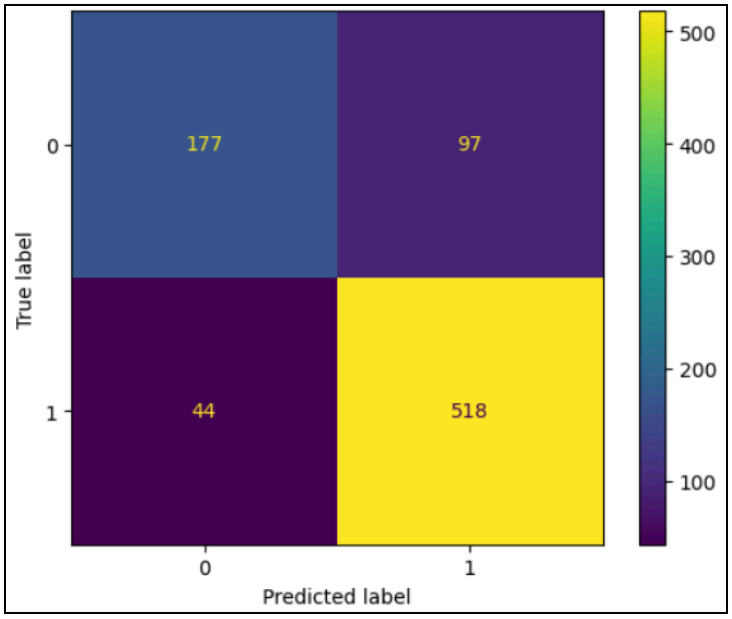


Random forest:

```
Best parameters: {'max_depth': 5, 'n_estimators': 118}
Test accuracy: 0.8313397129186603
Classification report:
              precision    recall  f1-score   support

           0       0.80      0.65      0.72       274
           1       0.84      0.92      0.88       562

    accuracy                           0.83       836
   macro avg       0.82      0.78      0.80       836
weighted avg       0.83      0.83      0.83       836
```

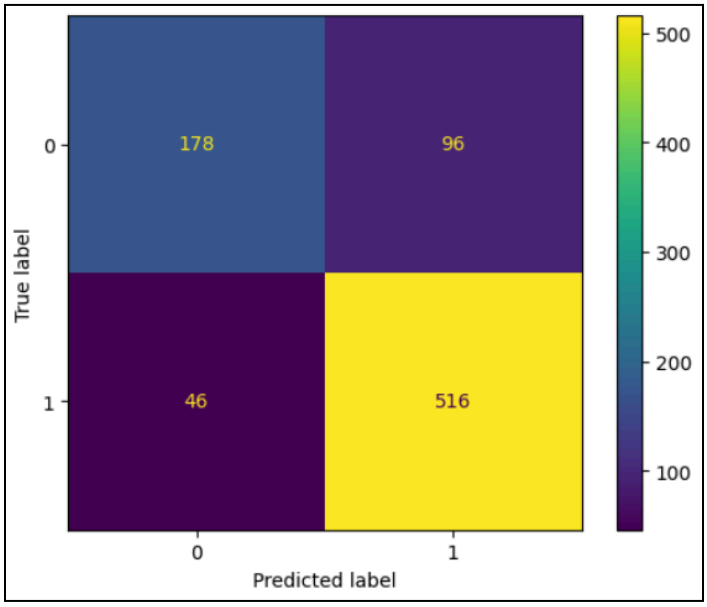


XGBoost:

```
Best parameters: {'learning_rate': np.float64(0.25874407221014245), 'max_depth': 3, 'n_estimators': 63}
Test accuracy: 0.8301435406698564
Classification report:
              precision    recall  f1-score   support

           0       0.79      0.65      0.71       274
           1       0.84      0.92      0.88       562

    accuracy                           0.83       836
   macro avg       0.82      0.78      0.80       836
weighted avg       0.83      0.83      0.83       836
```



For the random search, I used 30 random iterations for each classifier as a good mix of deliberate search and a reasonably fast response time. Overall, the random search tuning performed very similarly to the grid search tuning.

Conclusions and Takeaways

To conclude, I was able to estimate a linear regression model that could explain the variance of the dataset to a moderate degree using the variables given. I then trained tree-based classifiers to distinguish between abalone that were immature or mature, imitating the real-world problem of fishing regulations on species that are too young or small.

To apply my findings to this real-world scenario, I would say that given the size and weight dimensions of a given abalone, we can predict its ring count, and therefore age. In doing so, local governments and ecological conservation departments can establish regulations on the general dimensions of abalone that are harvestable without needing to go through the old tedious process of physically counting rings under a microscope. This allows auditors to make sure that fishing companies and even individuals with fishing licenses are compliant with the regulations, as well as the inverse, that fishers know the general dimensions of acceptable abalone before returning with their harvest.

As for the process itself, I struggled in areas that were about replicating statistical findings in Python. Since I am more experienced in R, I initially used that to find patterns and guide my work in Python. However, I wanted to replicate these findings in Python, but ultimately failed to do so. A good example of this is the VIFs, which I left in this documentation for this very reason. I still do not understand where the difference in VIF values comes from, but I still tried to replicate the findings I had in R.