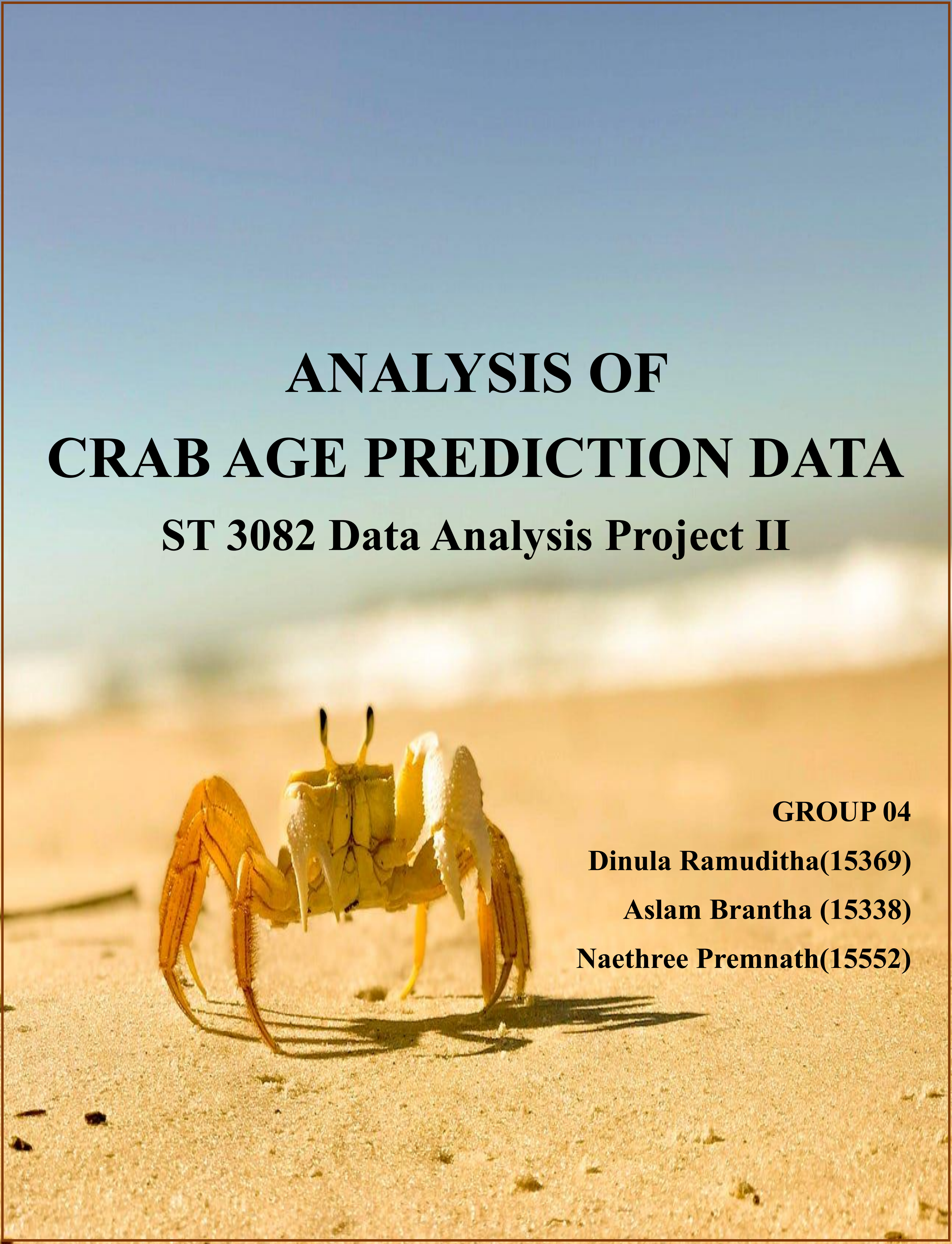

ANALYSIS OF CRAB AGE PREDICTION DATA

ST 3082 Data Analysis Project II

GROUP 04

Dinula Ramuditha(15369)

Aslam Brantha (15338)

Naethree Premnath(15552)

Abstract

This report aims to present the findings of the exploratory data analysis conducted on the Crab Age dataset obtained from Kaggle. The primary objective of this analysis is to identify the physical attributes that influence the age of crabs and to predict the age. The insights derived from this analysis contribute to a deeper understanding of the factors influencing crab age and facilitate the development of robust predictive models.

Table of Contents

List of Figures

List of Tables

1. Introduction

In the vast and vibrant world of the seafood industry, crabs stand out as both culinary delights and key players in the market. Understanding their age goes beyond mere curiosity; it's essential for sustainable harvesting, processing, and marketing.

"Capture of recently molted soft-shell crab in the Newfoundland and Labrador (NL) snow crab (Chionoecetes opilio) fishery is undesirable due to resource wastage associated with low meat yield and supposed high mortality rates upon discard." [1]

Knowing the age helps in protecting young crab populations, ensuring ecological balance, and providing consumers with the best quality. Our report delves into the relationship between a crab's physical attributes and its age, aiming to develop predictive models for age and age groups. This isn't just data crunching—it's about making informed, responsible decisions that benefit everyone from the ocean to the dinner table. By predicting a crab's age, we can harvest at the right time, ensuring top-quality seafood while supporting sustainable practices in the industry.

2. Description of the Question

For commercial crab farmers, having insight into the optimal age of crabs is crucial in determining the right moment for harvesting. As soon as crabs reach a certain age, there is minimal growth in their physical characteristics. Hence to reduce cost and increase profit, determining the age of crabs using its physical attributes is crucial.

"It is quite impossible to state the age of a crab with any degree of certainty" [2]
However, this report aims to answer the following questions considering just the data and not taking other external variables into consideration.

Q1) What are the physical attributes that affect the age of the crab and how can we predict the crab age just by looking at these physical features?

Q2) How can we predict the age group of crabs using their physical attributes so that the crabs of the optimal age category can be harvested?

3. Description of the Dataset

The Crab Age Prediction Dataset obtained from Kaggle contains 3893 observations with 9 variables. Description of each variable can be found in the table below:

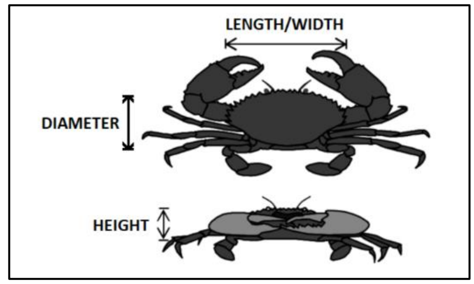

Figure 1

Variable	Description	Type
Sex	Gender of the Crab - Male, Female and Indeterminate	Categorical (nominal)
Length	Length of the Crab in feet	Quantitative
Diameter	Diameter of the Crab in feet	Quantitative
Height	Height of the Crab in feet	Quantitative
Weight	Weight of the Crab in ounces	Quantitative
Shucked Weight	Weight without the shell in ounces	Quantitative
Viscera Weight	Weight that wraps around the crab's abdominal organs in ounces	Quantitative
Shell Weight	Weight of the Shell in ounces	Quantitative
Age	Age of the Crab in months	Quantitative

Table 1

3.1. Data pre-processing

- No missing values were found in the dataset.
- No duplicate records.
- There were two zero values for the height of the crab which were removed from the dataset.
- Sex variable was transformed to a string variable and then was finally converted to a factor which will allow the Sex variable to be treated as a factor with three categories ‘Female’, ‘Male’, ‘Indeterminate’.

3.2. Feature Engineering

- A new variable ‘Age Group’ was created by using the Age variable.

  Age<10: Young

  Age>=10 & Age <=18: Adult

Age>18: Old

This feature was used as the response variable to answer the 2nd question.

- ➢ Feature scaling of all numeric variables to bring them to the same scale was performed when answering Question 1.

4. Important results of the Descriptive Analysis

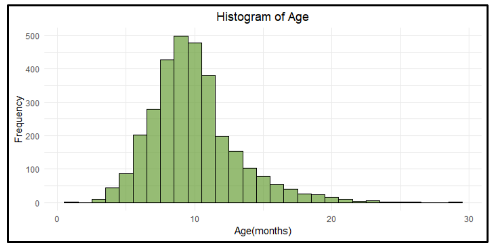

Figure 2

- ➢ All the predictor variables are skewed.
- ➢ The Age variable (response of Q1) is skewed by Figure 2 .

Although the typical lifespan of a crab is 3 to 5 years, certain species can live up to 30 years. Nevertheless, a lot depends on the type of crab. **[3]** Hence it could be understood that this dataset represents less of the old crabs.

- ➢ Outliers can be discovered through the univariate boxplots as well as Score plot by Figure 3 which was generated when doing the PLS regression.

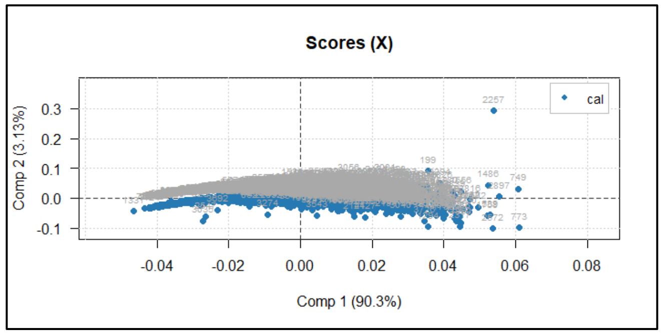

Figure 3

Variable	VIF Values
Length	38.556744
Diameter	39.388654
Height	6.155182
Weight	94.097443
Shucked Weight	25.066008
Viscera Weight	15.807106
Shell Weight	21.069303

Table 2

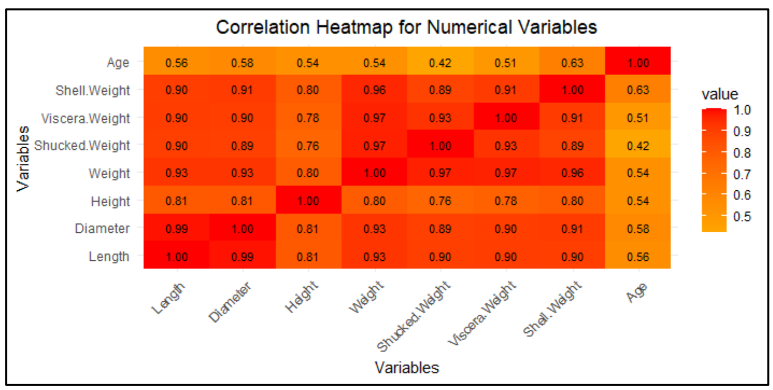

Figure 4

- ➢ Most of the numerical variables are correlated with each other as seen by Figure 4. The three components of Weight(Shucked,Viscera,Shell) were highly correlated with Weight variable. Length, Diameter, Height were highly correlated with each other. It makes sense since they are physical attributes.
- ➢ It was evident that multicollinearity exists between most of the predictor variables by Table 2 above.
- ➢ All the numerical predictors Length, Diameter, Height, Weight, Shucked Weight, Viscera Weight, Shell Weight have a positive relationship with response variable Age by the loading plot in Figure 5.

- By Figure 6, note that Female and Male crabs approximately have the similar distribution of Age. However Indeterminate crabs tend to have lower age. Hence this should be kept in mind for further analysis as Sex might not play a huge role.

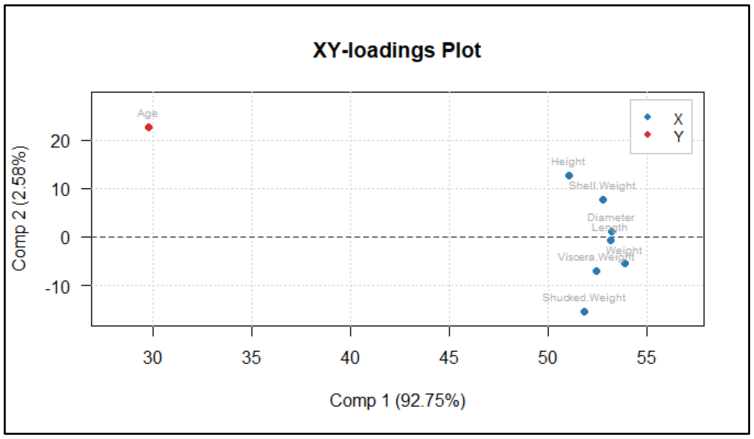

Figure 5

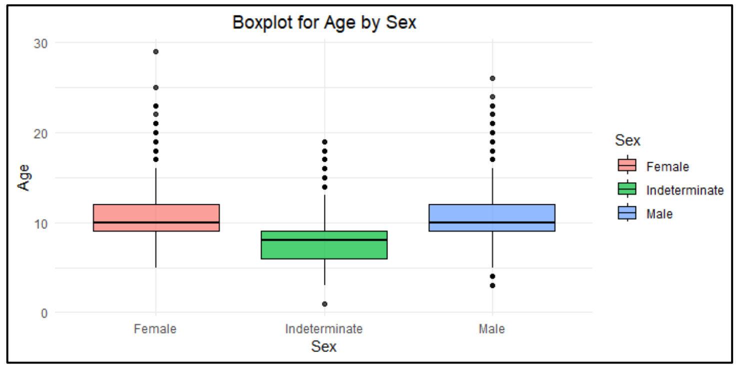

Figure 6

- From Figure 7, majority of the crabs are of the category Young and Adult. Old category is represented the least in the dataset which causes an imbalance of classes which will be dealt with in the Advanced analysis related to Q2.
- By Figure 8, bivariate analysis between Age group and numerical predictors can be observed. It is observed that old crabs have higher median length, diameter, height, weight, viscera weight and shell weight than adult and young crabs. In terms of shucked weight, adult crabs have higher median shucked weight than young and old crabs. Since Shucked Weight being the highest makes it profitable for the farmers, it could be understood that Adult class crabs seem to be the optimal age group for harvesting.

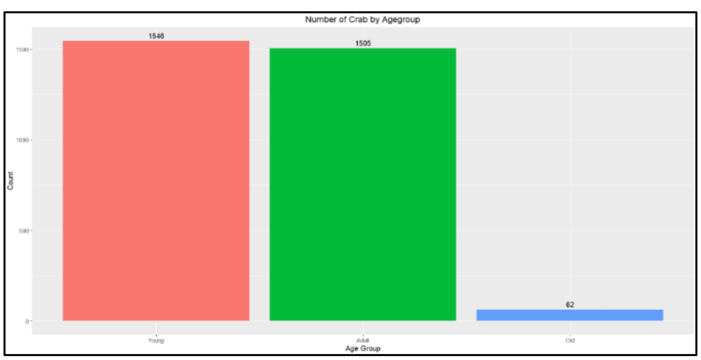

Figure 7

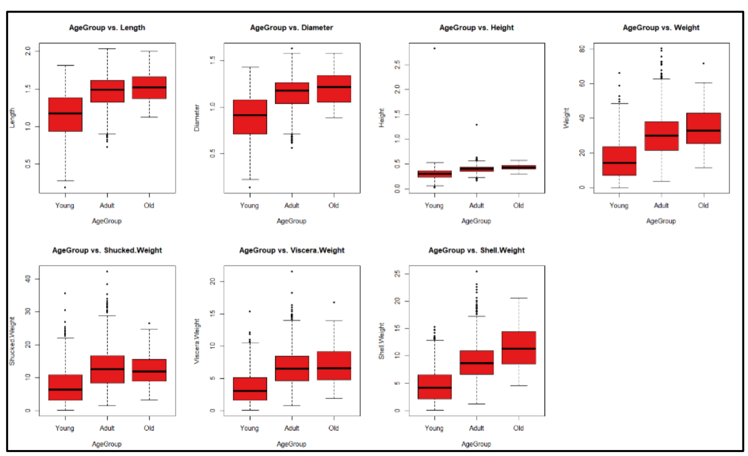

Figure 8

5. Advanced Analysis related to Q1

Dealing with Multicollinearity and duplicate information stored:

Since Weight ≈ Shucked Weight + Viscera Weight + Shell Weight, it can we understood that having the Weight variable is not needed. Hence to reduce the loss of information as well as to minimize the collinearity between variables, each of Shucked Weight, Viscera Weight, Shell Weight were converted to ratios of the Weight (Eg: Shucked Weight Ratio = Shucked Weight/Weight) and then, the Weight variable was removed. This helped reduce the multicollinearity within variables while not losing much information which helped in answering Q1.

Dealing with outliers:

The Mahalanobis plot showed only one outlier. However, from the Score plot(Figure 3), it was clear that there were more outliers. Since Mahalanobis works better when the data is normal, and since most of our variables are skewed, it was decided to use LOF(Local Outlier Factor) method to remove the outliers. Models were built both by removing and not removing outliers and the better models out of the two are listed below. For tree-based models, the model was fit directly without removing outliers since they are robust to outliers.

Transformation on response variable Age:

Age is skewed as seen earlier by Figure 2. To address this, a log transformation was applied to normalize the response variable, leading to improvements in the models.

- **Even though multicollinearity was reduced, there was still multicollinearity between Length, Diameter and Height variables. Hence algorithms like Ridge, Lasso, Elastic Net and PLS regression were first considered. Since PLS Regression was done to assist with EDA and deals with multicollinearity while reducing the dimensions, it was first considered as the baseline model.**

- **PLS Regression**

Best parameter for PLS: {n_components=2}

Outliers were removed after the evidence of the score plot. LOF(Local Outlier Factor) method was used to find the multivariate outliers and hence 312 outliers were removed before fitting the model.

2 components explained significant variance and hence model with 2 components was fit.

	Train	**Test**
MSE	0.0321	0.0336
RMSE	0.1792	0.1834
MAE	0.1382	0.1376
R^2	0.5814	0.5780

Table 3

- **Ridge Regression**

Best parameter for Ridge Regression: {'alpha': 1}

Ridge Regression was chosen since it deals with multicollinearity and prevents overfitting. Outliers were not removed since the model performed better with the outliers.

	Train	**Test**
MSE	0.0379	0.0338
RMSE	0.1947	0.1838
MAE	0.1468	0.1408
R^2	0.5388	0.5762

Table 4

❖ Lasso Regression

Best parameter for Lasso Regression: {'alpha': 0.1}

Outliers were not removed since the model performed better with the outliers. However since our dataset doesn't have many features, a feature selection wouldn't be necessary. Hence this model won't be chosen.

	Train	**Test**
MSE	0.0567	0.0540
RMSE	0.2381	0.2323
MAE	0.1796	0.1775
R^2	0.3105	0.3230

Table 5

❖ Elastic Net

Best parameters for ElasticNet Regression: {'alpha': 0.1, 'l1_ratio': 0.1}

Elastic Net Regression was chosen since it is a compromise between Ridge and Lasso and we were interested to see how the model performed. Outliers were removed from this model since the model performed better after removing them. LOF(Local Outlier Factor) method was used to find the multivariate outliers and hence 312 outliers were removed before fitting the ElasticNet model. However this model did not perform the best comparative to others.

	Train	**Test**
MSE	0.0429	0.0401
RMSE	0.2070	0.2003

MAE	0.1587	0.1546
R^2	0.4415	0.4968

Table 6

- **XGBoost**

Default parameters were used as tuning is difficult.

Although the train MSE is very low, it can be observed that Test MSE is comparatively large. Furthermore, from the R^2 value as well, it can be seen that fit is much better for training set than testing set. Hence it can be understood that this is an overfitted model and hence this model won't be accepted.

	Train	**Test**
MSE	0.0034	0.0327
RMSE	0.0582	0.1807
MAE	0.0427	0.1397
R^2	0.9589	0.5901

Table 7

- **Random Forest**

Best parameters for RandomForest: {'max_depth': 5, 'n_estimators': 100}

	Train	**Test**
MSE	0.0255	0.02758
RMSE	0.1600	0.1667
MAE	0.1235	0.1263
R^2	0.6886	0.6514

Table 8

6. Best Model

By considering the above Train Test MSE values, the gap between the MSE values, MAPE values and the R-squared values, it was concluded that the Random Forest Model performs best in predicting the crab age. The Table 9 shows the feature importance values obtained for the Random Forest model. Diameter, Shucked Weight Ratio are the most important variables in predicting age.

Although, this model is considered as the best model out of the ones that were studied, it is always crucial to remember ***"Growth rate is highly variable, such that individual size is not a good proxy for age".[4]*** Since Sex doesn't play a huge role in prediction according to feature importance values and also from our insights from Figure 6, it was decided to try removing Sex and fit the model. However, there were slight increases in train and test MSE and the R^2 also reduced. Hence it was decided to keep Sex in the model. Hence all the features shown in the Table 9 Was considered for the best model.

Feature	Feature Importance Value
Diameter	0.4667
Shucked_Weight_Ratio	0.2141
Height	0.1721
Length	0.0848
Sex_I (Indeterminate)	0.0268
Viscera_Weight_Ratio	0.0215
Shell_Weight_Ratio	0.0136
Sex_F (Female)	0.0003
Sex_M (Male)	0.0002

Table 9

According to the Partial Dependence Plots given by Figure 9, it is observed that as Diameter increases keeping other variables constant, the predicted log age of the crab increases, suggesting a positive partial dependency implying that crabs with higher diameter tend to be older according to the model. Similarly, positive partial dependencies are observed in Length, Height, Shell Weight Ratio suggesting higher predicted log age of crabs for increased values of above variables. On the contrary, Shucked Weight Ratio, Viscera Weight Ratio gives a negative partial dependence suggesting lower predicted log age of crabs**.** This also confirms the importance of harvesting the crabs at the optimal age since crabs tend to lose their shucked Weight when getting older, leading to reduced quality in crabs and less profits.

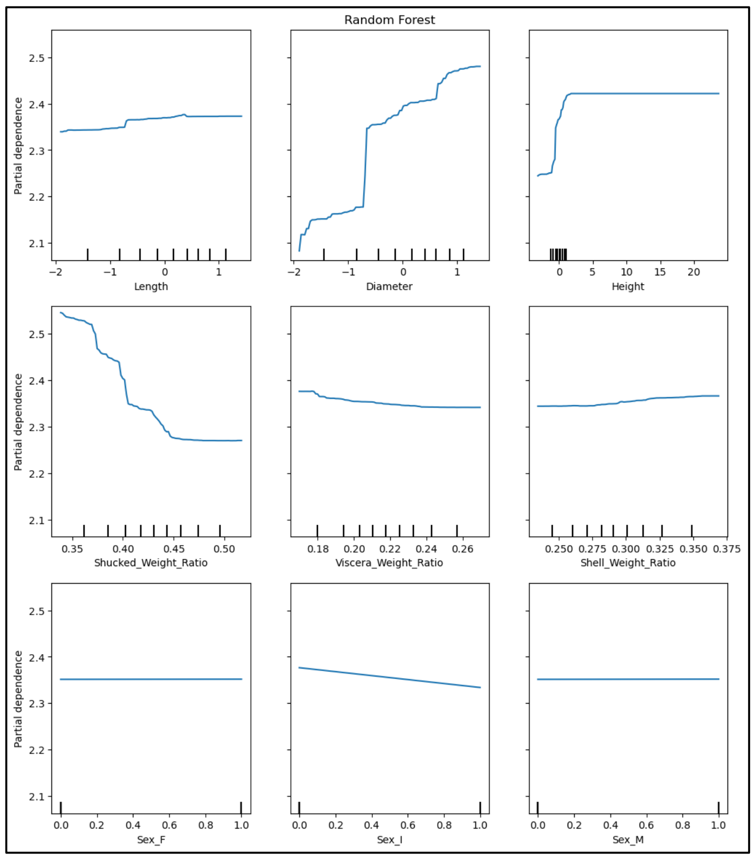

Figure 9

7. Issues encountered and proposed solutions

- Due to persistent multicollinearity even after the removal of the Weight variable and conversion of three variables to ratios, it was not feasible to fit models like Multiple Linear Regression. If we attempted to remove variables such as Diameter and Height, it would result in information loss. Removing variables from a dataset which has only a few features isn't the best solution and hence models which can be affected by Multicollinearity were not considered.

- Univariate boxplots weren't considered to remove outliers since considering multivariate outliers is more appropriate. However, Mahalanobis plot showed only 1 outlier maybe due to the normality assumption not being satisfied. Hence LOF outlier removal method was used to remove 312 outliers. There could be other methods which could have resulted in better models, however since our final model Random Forest Model performs well without removing outliers(since it is robust to outliers) this issue doesn't affect the final chosen model.
- Due to the lack of information of the type of crabs, the environmental factors affecting the crabs, different crabs can grow at different rates. Hence using physical attributes to predict age is not encouraged.

  "Of the 231 studies examined, 83% used length–frequency analysis, 13% used lipofuscin and 4% used growth bands"[5] . Previous references stated that although done frequently, length-frequency analysis was the least successful stating that using physical attributes to predict crab age is not very successful. Using Growth bands along with lipofuscin analysis if far more accurate in predicting age says the reports. However with the given dataset and the limited information, the models were fit and the best was chosen.

8. Discussions and Conclusions

The results obtained by Advanced Analysis can be summarized as follows:

	Train MSE	**Test MSE**	**Test - Train MSE**
PLS Regression	0.0321	0.0336	0.0015
Ridge Regression	0.0379	0.0338	-0.0041
Lasso regression	0.0567	0.0540	-0.0027
Elastic Net Regression	0.0429	0.0401	-0.0028
XG Boost	0.0034	0.0327	0.0293
Random Forest	0.0255	0.02758	0.00208

Table 10

Hence, the Random Forest Model was chosen as the best model with parameters{'max_depth': 5, 'n_estimators': 100}.

9. References:

Dataset: https://www.kaggle.com/datasets/sidhus/crab-age-prediction

[1] https://www.frontiersin.org/articles/10.3389/fmars.2021.591496/full

[2] https://www.int-res.com/articles/meps2008/353/m353p191.pdf

[3] https://www.learnaboutnature.com/invertebrates/crabs/crab-life-cycle/

[4] https://www.int-res.com/articles/meps2008/353/m353p191.pdf

[5] https://www.researchgate.net/publication/317003887_Age_determination_in_crustaceans_a_review

[6] https://www.kaggle.com/code/oscarm524/ps-s3-ep16-eda-modeling-submission

[7] https://rpubs.com/sarvinnah/1055418

[8] https://www.kaggle.com/datasets/sidhus/crab-age-prediction

[9] https://allmodelsarewrong.github.io/pls.html

[10] https://youtu.be/0xFdu5okHkw?si=vi_B44EcTdq4fYAy

[11] https://youtu.be/EHb_kuw1GNU?si=jdroGWy5WBsYV-G9

10. Appendix:

Q2:

Predict the age group of crabs based on the physical attributes with classification algorithms

- The variable "Age Group," that was created will be used. Stratification was applied to ensure consistency in the distribution of age groups between the training and testing sets, maintaining a comparable structure for accurate and representative model assessment.

Training set Age Group Percentages:

```
    Young      Adult        Old
49.614644 48.105331  2.280026
```

Testing set Age Group Percentages:

```
    Young      Adult        Old
49.678250 48.133848  2.187902
```

Advanced Analysis of Q2

- No outliers were removed.
- Logistic Regression, Support Vector Machine, Decision Trees and Random Forest were considered.

- **Logistic Regression**

  Since multicollinearity existed and it affects logistic regression, Ridge regression was applied as a pre-processing step and then Logistic Regression was fitted.

  From the figure below, the model's performance can be observed.

```
Logistic Regression Accuracy: 0.7047496790757382
              precision    recall  f1-score   support

           0       0.71      0.66      0.69       375
           1       0.20      0.61      0.30        18
           2       0.78      0.75      0.76       386

    accuracy                           0.70       779
   macro avg       0.56      0.67      0.58       779
weighted avg       0.73      0.70      0.71       779

Confusion Matrix:
[[249  43  83]
 [  7  11   0]
 [ 96   1 289]]
```

- **SVM**

  Since multicollinearity existed and it affects SVM, Ridge regression was applied as a pre-processing step and then SVM was fitted.

  From the figure below, the model's performance can be observed.

```
Support Vector Machine Accuracy: 0.7021822849807445
              precision    recall  f1-score   support

           0       0.68      0.71      0.70       375
           1       0.23      0.78      0.35        18
           2       0.82      0.69      0.75       386

    accuracy                           0.70       779
   macro avg       0.57      0.73      0.60       779
weighted avg       0.74      0.70      0.71       779

Confusion Matrix(SVM):
[[267  48  60]
 [  4  14   0]
 [120   0 266]]
```

❖ Decision Trees

Decision trees are not affected by multicollinearity since they make splits based on individual predictor variables and their thresholds without considering the relationships between predictors. Hence directly, Decision tree model was fitted.

From the figure below, the model's performance can be observed.

```
Decision Tree Accuracy: 0.6623876765083441
              precision    recall  f1-score   support

       Adult       0.65      0.66      0.66       375
         Old       0.04      0.06      0.04        18
       Young       0.72      0.69      0.71       386

    accuracy                           0.66       779
   macro avg       0.47      0.47      0.47       779
weighted avg       0.67      0.66      0.67       779

Confusion Matrix (Decision Tree):
[[248  25 102]
 [ 16   1   1]
 [117   2 267]]
```

❖ Random Forest

Similar to decision trees, random forest is also robust to multicollinearity.

From the figure below, the model's performance can be observed.

```
Random Forest Accuracy: 0.748395378690629
              precision    recall  f1-score   support

       Adult       0.70      0.83      0.76       375
         Old       0.00      0.00      0.00        18
       Young       0.81      0.71      0.76       386

    accuracy                           0.75       779
   macro avg       0.50      0.51      0.50       779
weighted avg       0.74      0.75      0.74       779

Confusion Matrix(Random Forest):
[[310   1  64]
 [ 18   0   0]
 [113   0 273]]
```

Best Model for Q2

It appears that Random Forest performs better overall, especially in terms of accuracy, precision, and recall for the 'Adult' and 'Young' classes. Even if it doesn't perform the best for the "Old" category, since Random Forest also has the advantage of being robust to outliers, it can be considered as the best model.

Code

1

```python
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2
crabdata = pd.read_csv("C:/Users/User/Documents/UNI/3rd year/semester 2/ML/Project 1/CrabAgePrediction.csv")
crabdata.head()
crabdata.info()
#remove height=0
crabdata = crabdata[crabdata['Height'] > 0]
```

2

```python
# Factorize the 'Sex' column
crabdata['Sex'] = pd.factorize(crabdata['Sex'])[0]

# Change the numerical encoding to specific labels
sex_labels = {0: 'Female', 1: 'Male', 2: 'Indeterminate'}
crabdata['Sex'] = crabdata['Sex'].map(sex_labels)

# Show the first few rows of the DataFrame to verify the changes
print(crabdata.head())
crabdata.info()
```

3

```python
#data on different scales. so we standardize
from sklearn.preprocessing import StandardScaler
#split data to features and label by making a copy of each
X=crabdata[["Sex","Length","Height","Weight","Diameter","Shucked Weight", "Viscera Weight", "Shell Weight"]].copy()
X['Shucked_Weight_Ratio'] = X['Shucked Weight'] / X['Weight']
X['Shell_Weight_Ratio'] = X['Shell Weight'] / X['Weight']
X['Viscera_Weight_Ratio'] = X['Viscera Weight'] / X['Weight']

X.drop(columns=['Weight', 'Shucked Weight', 'Viscera Weight', 'Shell Weight'], inplace=True)
Y=crabdata["Age"].copy()
```

4

```python
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

categorical_cols = ["Sex"]
numerical_cols_to_scale = ["Length", "Diameter", "Height"]
numerical_cols_no_scale = ["Shucked_Weight_Ratio", "Viscera_Weight_Ratio", "Shell_Weight_Ratio"]

# Define preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num_scale', StandardScaler(), numerical_cols_to_scale),
        ('num_no_scale', 'passthrough', numerical_cols_no_scale),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Define the pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the data
X_processed = pipeline.fit_transform(X)
print(X_processed)
```

5

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train_scaled, X_test_scaled, Y_train, Y_test = train_test_split(X_processed, Y,
                                                                  train_size=0.8,
                                                                  random_state=123)

# Make log transformation
Y_train_log=np.log1p(Y_train)
Y_test_transformed = np.log1p(Y_test)
```

Q1 - Model Fitting Before Removing Outliers

6

```
#RIDGE REGRESSION
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score


param_grid_ridge = {
    'alpha': [0.1, 1, 10],
}
ridge_model_tuned = Ridge()
grid_search_ridge = GridSearchCV(ridge_model_tuned, param_grid_ridge, cv=5, scoring='neg_mean_squared_error')
grid_search_ridge.fit(X_train_scaled, Y_train_log)
best_params_ridge = grid_search_ridge.best_params_
best_score_ridge = grid_search_ridge.best_score_

print("Best parameters for Ridge Regression:", best_params_ridge)
print("Best score for Ridge Regression:", best_score_ridge)

ridge_model_best = Ridge(**best_params_ridge)
ridge_model_best.fit(X_train_scaled, Y_train_log)
```

7

```
# Predictions on the training set
ridge_train_predictions = ridge_model_best.predict(X_train_scaled)
# Training MSE
ridge_train_mse = mean_squared_error(Y_train_log, ridge_train_predictions)
# Predictions on the test set
ridge_test_predictions = ridge_model_best.predict(X_test_scaled)
# Test MSE
ridge_test_mse = mean_squared_error(Y_test_transformed, ridge_test_predictions)
# RMSE for training predictions
ridge_train_rmse = mean_squared_error(Y_train_log, ridge_train_predictions, squared=False)
# RMSE for test predictions
ridge_test_rmse = mean_squared_error(Y_test_transformed, ridge_test_predictions, squared=False)
# MAE for training predictions
ridge_train_mae = mean_absolute_error(Y_train_log, ridge_train_predictions)
# MAE for test predictions
ridge_test_mae = mean_absolute_error(Y_test_transformed, ridge_test_predictions)
# R^2 for training predictions
ridge_train_r2 = r2_score(Y_train_log, ridge_train_predictions)
# R^2 for test predictions
ridge_test_r2 = r2_score(Y_test_transformed, ridge_test_predictions)
# Difference between train and test MSE
ridge_mse_difference= ridge_train_mse - ridge_test_mse

print("Ridge Train MSE:", ridge_train_mse)
print("Ridge Test MSE:", ridge_test_mse)
print("Ridge Regression Train RMSE:", ridge_train_rmse)
print("Ridge Regression Test RMSE:", ridge_test_rmse)
print("Ridge Regression Train MAE:", ridge_train_mae)
print("Ridge Regression Test MAE:", ridge_test_mae)
print("Ridge Regression Train R^2:", ridge_train_r2)
print("Ridge Regression Test R^2:", ridge_test_r2)
print("Ridge Train-Test MSE Difference:", ridge_mse_difference)
```

8

```python
#LASSO REGRESSION
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

param_grid_lasso = {
    'alpha': [0.1, 1, 10],
}
lasso_model_tuned = Lasso()
grid_search_lasso = GridSearchCV(lasso_model_tuned, param_grid_lasso, cv=5, scoring='neg_mean_squared_error')
grid_search_lasso.fit(X_train_scaled, Y_train_log)
best_params_lasso = grid_search_lasso.best_params_
best_score_lasso = grid_search_lasso.best_score_

print("Best parameters for Lasso Regression:", best_params_lasso)
print("Best score for Lasso Regression:", best_score_lasso)
```

9

```python
lasso_model_best = Lasso(**best_params_lasso)
lasso_model_best.fit(X_train_scaled, Y_train_log)
# Predictions on the training set
lasso_train_predictions = lasso_model_best.predict(X_train_scaled)
# Training MSE
lasso_train_mse = mean_squared_error(Y_train_log, lasso_train_predictions)
# Predictions on the test set
lasso_test_predictions = lasso_model_best.predict(X_test_scaled)
# Test MSE
lasso_test_mse = mean_squared_error(Y_test_transformed, lasso_test_predictions)
# RMSE for training predictions
lasso_train_rmse = mean_squared_error(Y_train_log, lasso_train_predictions, squared=False)
# RMSE for test predictions
lasso_test_rmse = mean_squared_error(Y_test_transformed, lasso_test_predictions, squared=False)
# MAE for training predictions
lasso_train_mae = mean_absolute_error(Y_train_log, lasso_train_predictions)
# MAE for test predictions
lasso_test_mae = mean_absolute_error(Y_test_transformed, lasso_test_predictions)
# R^2 for training predictions
lasso_train_r2 = r2_score(Y_train_log, lasso_train_predictions)
# R^2 for test predictions
lasso_test_r2 = r2_score(Y_test_transformed, lasso_test_predictions)
# Difference between train and test MSE
lasso_mse_difference = lasso_train_mse - lasso_test_mse
print("Lasso Train MSE:", lasso_train_mse)
print("Lasso Test MSE:", lasso_test_mse)
print("Lasso Regression Train RMSE:", lasso_train_rmse)
print("Lasso Regression Test RMSE:", lasso_test_rmse)
print("Lasso Regression Train MAE:", lasso_train_mae)
print("Lasso Regression Test MAE:", lasso_test_mae)
print("Lasso Regression Train R^2:", lasso_train_r2)
print("Lasso Regression Test R^2:", lasso_test_r2)
print("Lasso Train-Test MSE Difference:", lasso_mse_difference)
```

10

```python
#ELASTICNET
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

param_grid_elasticnet = {
    'alpha': [0.1, 1, 10],
    'l1_ratio': [0.1, 0.5, 0.9]  # Ratio of L1 to L2 penalty
}
elasticnet_model_tuned = ElasticNet()
grid_search_elasticnet = GridSearchCV(elasticnet_model_tuned, param_grid_elasticnet, cv=5, scoring='neg_mean_squared_error')
grid_search_elasticnet.fit(X_train_scaled, Y_train_log)
best_params_elasticnet = grid_search_elasticnet.best_params_
best_score_elasticnet = grid_search_elasticnet.best_score_

print("Best parameters for Elastic Net Regression:", best_params_elasticnet)
print("Best score for Elastic Net Regression:", best_score_elasticnet)
```

11

```python
elasticnet_model_best = ElasticNet(**best_params_elasticnet)
elasticnet_model_best.fit(X_train_scaled, Y_train_log)
# Predictions on the training set
elasticnet_train_predictions = elasticnet_model_best.predict(X_train_scaled)
# Training MSE
elasticnet_train_mse = mean_squared_error(Y_train_log, elasticnet_train_predictions)
# Predictions on the test set
elasticnet_test_predictions = elasticnet_model_best.predict(X_test_scaled)
# Test MSE
elasticnet_test_mse = mean_squared_error(Y_test_transformed, elasticnet_test_predictions)
# RMSE for training predictions
elasticnet_train_rmse = mean_squared_error(Y_train_log, elasticnet_train_predictions, squared=False)
# RMSE for test predictions
elasticnet_test_rmse = mean_squared_error(Y_test_transformed, elasticnet_test_predictions, squared=False)
# MAE for training predictions
elasticnet_train_mae = mean_absolute_error(Y_train_log, elasticnet_train_predictions)
# MAE for test predictions
elasticnet_test_mae = mean_absolute_error(Y_test_transformed, elasticnet_test_predictions)
# R^2 for training predictions
elasticnet_train_r2 = r2_score(Y_train_log, elasticnet_train_predictions)
# R^2 for test predictions
elasticnet_test_r2 = r2_score(Y_test_transformed, elasticnet_test_predictions)
# Difference between train and test MSE
elasticnet_mse_difference = elasticnet_train_mse - elasticnet_test_mse
print("Elastic Net Train MSE:", elasticnet_train_mse)
print("Elastic Net Test MSE:", elasticnet_test_mse)
print("Elastic Net Regression Train RMSE:", elasticnet_train_rmse)
print("Elastic Net Regression Test RMSE:", elasticnet_test_rmse)
print("Elastic Net Regression Train MAE:", elasticnet_train_mae)
print("Elastic Net Regression Test MAE:", elasticnet_test_mae)
print("Elastic Net Regression Train R^2:", elasticnet_train_r2)
print("Elastic Net Regression Test R^2:", elasticnet_test_r2)
print("Elastic Net Train-Test MSE Difference:", elasticnet_mse_difference)
```

12

```python
#XGBOOST(without tuning)
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score

# Create an instance of XGBRegressor with default parameters
xgb_model_default = XGBRegressor()

# Fit the model
xgb_model_default.fit(X_train_scaled, Y_train_log)

# Print the score (negative mean squared error)
score_default = xgb_model_default.score(X_train_scaled, Y_train_log)
print("Score for XGBoost (Default):", score_default)
```

13

```python
# Predictions on the training set
xgb_train_predictions = xgb_model_default.predict(X_train_scaled)
# Calculate training MSE
xgb_train_mse = mean_squared_error(Y_train_log, xgb_train_predictions)
# Predictions on the test set
xgb_test_predictions = xgb_model_default.predict(X_test_scaled)
# Test MSE
xgb_test_mse = mean_squared_error(Y_test_transformed, xgb_test_predictions)
# RMSE for training predictions
xgb_train_rmse = mean_squared_error(Y_train_log, xgb_train_predictions, squared=False)
# RMSE for test predictions
xgb_test_rmse = mean_squared_error(Y_test_transformed, xgb_test_predictions, squared=False)
# MAE for training predictions
xgb_train_mae = mean_absolute_error(Y_train_log, xgb_train_predictions)
# MAE for test predictions
xgb_test_mae = mean_absolute_error(Y_test_transformed, xgb_test_predictions)
# R^2 for training predictions
xgb_train_r2 = r2_score(Y_train_log, xgb_train_predictions)
# R^2 for test predictions
xgb_test_r2 = r2_score(Y_test_transformed, xgb_test_predictions)
# Difference between train and test MSE
xgb_mse_difference = xgb_train_mse - xgb_test_mse
# Print the results
print("XGBoost Training MSE:", xgb_train_mse)
print("XGBoost Test MSE:", xgb_test_mse)
print("XGBoost Regression Train RMSE:", xgb_train_rmse)
print("XGBoost Regression Test RMSE:", xgb_test_rmse)
print("XGBoost Regression Train MAE:", xgb_train_mae)
print("XGBoost Regression Test MAE:", xgb_test_mae)
print("XGBoost Regression Train R^2:", xgb_train_r2)
print("XGBoost Regression Test R^2:", xgb_test_r2)
print("XGBoost Train-Test MSE Difference:", xgb_mse_difference)
```

14

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score


param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5]
}
rf_model_tuned = RandomForestRegressor()
grid_search_rf = GridSearchCV(rf_model_tuned, param_grid_rf, cv=5, scoring='neg_mean_squared_error')
grid_search_rf.fit(X_train_scaled, Y_train_log)

# Access the best parameters and best score
best_params_rf = grid_search_rf.best_params_
best_score_rf = grid_search_rf.best_score_

print("Best parameters for RandomForest:", best_params_rf)
print("Best score for RandomForest:", best_score_rf)
```

15

```
rf_model_best = RandomForestRegressor(**best_params_rf)
rf_model_best.fit(X_train_scaled, Y_train_log)
# Predictions on the training set
rf_train_predictions = rf_model_best.predict(X_train_scaled)
# Training MSE
rf_train_mse = mean_squared_error(Y_train_log, rf_train_predictions)
# Predictions on the test set
rf_test_predictions = rf_model_best.predict(X_test_scaled)
# Test MSE
rf_test_mse = mean_squared_error(Y_test_transformed, rf_test_predictions)
# RMSE for training predictions
rf_train_rmse = mean_squared_error(Y_train_log, rf_train_predictions, squared=False)
# RMSE for test predictions
rf_test_rmse = mean_squared_error(Y_test_transformed, rf_test_predictions, squared=False)
# MAE for training predictions
rf_train_mae = mean_absolute_error(Y_train_log, rf_train_predictions)
# MAE for test predictions
rf_test_mae = mean_absolute_error(Y_test_transformed, rf_test_predictions)
# R^2 for training predictions
rf_train_r2 = r2_score(Y_train_log, rf_train_predictions)
# R^2 for test predictions
rf_test_r2 = r2_score(Y_test_transformed, rf_test_predictions)
# Difference between train and test MSE
rf_mse_difference = rf_train_mse - rf_test_mse
# Calculate the difference between train and test MSE
rf_mse_difference= rf_train_mse - rf_test_mse
print("Random Forest Train MSE:", rf_train_mse)
print("Random Forest Test MSE:", rf_test_mse)
print("Random Forest Regression Train RMSE:", rf_train_rmse)
print("Random Forest Regression Test RMSE:", rf_test_rmse)
print("Random Forest Regression Train MAE:", rf_train_mae)
print("Random Forest Regression Test MAE:", rf_test_mae)
print("Random Forest Regression Train R^2:", rf_train_r2)
print("Random Forest Regression Test R^2:", rf_test_r2)
print("Random Forest Train-Test MSE Difference:", rf_mse_difference)
```

Q1 - Removing Outliers

16

```
from sklearn.neighbors import LocalOutlierFactor
# Outlier detection using Local Outlier Factor (LOF)
lof_model = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
lof_outliers = lof_model.fit_predict(X_train_scaled)
lof_outliers_indices = np.where(lof_outliers == -1)[0]
# Print indices of outliers detected by LOF
print("Indices of outliers detected by LOF:", lof_outliers_indices)
# Count the number of outliers detected by LOF
num_outliers = len(lof_outliers_indices)
print("Number of outliers detected by LOF:", num_outliers)
# Remove 312 outliers from X_train_scaled
X_train_scaled_cleaned = np.delete(X_train_scaled, lof_outliers_indices, axis=0)

# Adjust Y_train
Y_train_cleaned = np.delete(Y_train, lof_outliers_indices, axis=0)
# Make log transformation
Y_train_cleaned_log=np.log1p(Y_train_cleaned)
Y_test_transformed = np.log1p(Y_test)
```

Q1 - Model Fitting After Removing Outliers

17

```
#ELASTICNET
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error

param_grid_elasticnet = {
    'alpha': [0.1, 1, 10],
    'l1_ratio': [0.1, 0.5, 0.9]
}
elasticnet_model_tuned = ElasticNet()
grid_search_elasticnet = GridSearchCV(elasticnet_model_tuned, param_grid_elasticnet, cv=5, scoring='neg_mean_squared_error')
grid_search_elasticnet.fit(X_train_scaled_cleaned, Y_train_cleaned_log)
best_params_elasticnet = grid_search_elasticnet.best_params_
best_score_elasticnet = grid_search_elasticnet.best_score_

print("Best parameters for ElasticNet Regression:", best_params_elasticnet)
print("Best score for ElasticNet Regression:", best_score_elasticnet)
```

18

```
elasticnet_model_best = ElasticNet(**best_params_elasticnet)
elasticnet_model_best.fit(X_train_scaled_cleaned, Y_train_cleaned_log)
# Predictions on the training set
elasticnet_train_predictions = elasticnet_model_best.predict(X_train_scaled_cleaned)
# Training MSE
elasticnet_train_mse = mean_squared_error(Y_train_cleaned_log, elasticnet_train_predictions)
# Predictions on the test set
elasticnet_test_predictions = elasticnet_model_best.predict(X_test_scaled)
# Test MSE
elasticnet_test_mse = mean_squared_error(Y_test_transformed, elasticnet_test_predictions)
# RMSE for training predictions
elasticnet_train_rmse = mean_squared_error(Y_train_cleaned_log, elasticnet_train_predictions, squared=False)
# RMSE for test predictions
elasticnet_test_rmse = mean_squared_error(Y_test_transformed, elasticnet_test_predictions, squared=False)
# MAE for training predictions
elasticnet_train_mae = mean_absolute_error(Y_train_cleaned_log, elasticnet_train_predictions)
# MAE for test predictions
elasticnet_test_mae = mean_absolute_error(Y_test_transformed, elasticnet_test_predictions)
# R^2 for training predictions
elasticnet_train_r2 = r2_score(Y_train_cleaned_log, elasticnet_train_predictions)
# R^2 for test predictions
elasticnet_test_r2 = r2_score(Y_test_transformed, elasticnet_test_predictions)
# Difference between training and test MSE
elasticnet_mse_difference = elasticnet_train_mse - elasticnet_test_mse
print("ElasticNet Train MSE:", elasticnet_train_mse)
print("ElasticNet Test MSE:", elasticnet_test_mse)
print("ElasticNet Regression Train RMSE:", elasticnet_train_rmse)
print("ElasticNet Regression Test RMSE:", elasticnet_test_rmse)
print("ElasticNet Regression Train MAE:", elasticnet_train_mae)
print("ElasticNet Regression Test MAE:", elasticnet_test_mae)
print("ElasticNet Regression Train R^2:", elasticnet_train_r2)
print("ElasticNet Regression Test R^2:", elasticnet_test_r2)
print("ElasticNet Train-Test MSE Difference:", elasticnet_mse_difference)
```

19

```
# PLS
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt

# Define PLS Regression with varying number of components
pls_model_cv = PLSRegression()

components_range = range(1, 10)
cv_mse_scores = []  # Store MSE scores

for n_components in components_range:
    pls_model_cv.n_components = n_components
    # Perform cross-validation with MSE scoring
    scores = -1 * cross_val_score(pls_model_cv, X_train_scaled_cleaned,
                                  Y_train_cleaned_log, cv=5, scoring='neg_mean_squared_error')
    cv_mse_scores.append(scores.mean())

# Plot the cross-validation MSE scores
plt.plot(components_range, cv_mse_scores)
plt.xlabel('Number of Components')
plt.ylabel('Cross-Validation Mean Squared Error (MSE)')
plt.title('Cross-Validation Mean Squared Error (MSE) vs. Number of Components')
plt.grid(True)
plt.show()
```

20

```python
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score
# Instantiate PLS Regression(chose 2 components from above graph)
pls_model = PLSRegression(n_components=2)
# Fit PLS Regression
pls_model.fit(X_train_scaled_cleaned, Y_train_cleaned_log)
# Predict on training data
pls_train_predictions = pls_model.predict(X_train_scaled_cleaned)
# Predict on test data
pls_test_predictions = pls_model.predict(X_test_scaled)
# MSE for training predictions
pls_train_mse = mean_squared_error(Y_train_cleaned_log, pls_train_predictions)
# MSE for test predictions
pls_test_mse = mean_squared_error(Y_test_transformed, pls_test_predictions)
# RMSE for training predictions
pls_train_rmse = mean_squared_error(Y_train_cleaned_log, pls_train_predictions, squared=False)
# RMSE for test predictions
pls_test_rmse = mean_squared_error(Y_test_transformed, pls_test_predictions, squared=False)
# MAE for training predictions
pls_train_mae = mean_absolute_error(Y_train_cleaned_log, pls_train_predictions)
# MAE for test predictions
pls_test_mae = mean_absolute_error(Y_test_transformed, pls_test_predictions)
# R^2 for training predictions
pls_train_r2 = r2_score(Y_train_cleaned_log, pls_train_predictions)
# R^2 for test predictions
pls_test_r2 = r2_score(Y_test_transformed, pls_test_predictions)
# Difference between training and test MSE
pls_mse_difference = pls_train_mse - pls_test_mse
print("PLS Regression Train MSE:", pls_train_mse)
print("PLS Regression Test MSE:", pls_test_mse)
print("PLS Regression Train RMSE:", pls_train_rmse)
print("PLS Regression Test RMSE:", pls_test_rmse)
print("PLS Regression Train MAE:", pls_train_mae)
print("PLS Regression Test MAE:", pls_test_mae)
print("PLS Regression Train R^2:", pls_train_r2)
print("PLS Regression Test R^2:", pls_test_r2)
print("PLS Regression Train-Test MSE Difference:", pls_mse_difference)
```

Q1 - Remove variables causing before fitting SVR

21

```python
#data on different scales. so we standardize
from sklearn.preprocessing import StandardScaler
#split data to features and label by making a copy of each
X=crabdata[["Sex","Length","Height","Weight","Diameter","Shucked Weight", "Viscera Weight", "Shell Weight"]].copy()
X['Shucked_Weight_Ratio'] = X['Shucked Weight'] / X['Weight']
X['Shell_Weight_Ratio'] = X['Shell Weight'] / X['Weight']
X['Viscera_Weight_Ratio'] = X['Viscera Weight'] / X['Weight']

X.drop(columns=['Weight', 'Shucked Weight', 'Viscera Weight', 'Shell Weight','Diameter','Height'], inplace=True)
Y=crabdata["Age"].copy()
```

22

```python
#SVR
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

param_grid_svr = {
    'C': [0.1, 1, 10],
    'gamma': [0.01, 0.1, 1],
    'kernel': ['linear', 'rbf']
}
svr_model_tuned = SVR()
grid_search_svr = GridSearchCV(svr_model_tuned, param_grid_svr, cv=5, scoring='neg_mean_squared_error')
grid_search_svr.fit(X_train_scaled_cleaned, Y_train_cleaned_log)
best_params_svr = grid_search_svr.best_params_
best_score_svr = grid_search_svr.best_score_

print("Best parameters for SVR:", best_params_svr)
print("Best score for SVR:", best_score_svr)
```

23

```python
svr_model_best = SVR(**best_params_svr)
svr_model_best.fit(X_train_scaled_cleaned, Y_train_cleaned_log)
# Predictions on the training set
svr_train_predictions = svr_model_best.predict(X_train_scaled_cleaned)
# Training MSE
svr_train_mse = mean_squared_error(Y_train_cleaned_log, svr_train_predictions)
# Predictions on the test set
svr_test_predictions = svr_model_best.predict(X_test_scaled)
# Test MSE
svr_test_mse = mean_squared_error(Y_test_transformed, svr_test_predictions)
# RMSE for training predictions
svr_train_rmse = mean_squared_error(Y_train_cleaned_log, svr_train_predictions, squared=False)
# RMSE for test predictions
svr_test_rmse = mean_squared_error(Y_test_transformed, svr_test_predictions, squared=False)
# MAE for training predictions
svr_train_mae = mean_absolute_error(Y_train_cleaned_log, svr_train_predictions)
# MAE for test predictions
svr_test_mae = mean_absolute_error(Y_test_transformed, svr_test_predictions)
# R^2 for training predictions
svr_train_r2 = r2_score(Y_train_cleaned_log, svr_train_predictions)
# R^2 for test predictions
svr_test_r2 = r2_score(Y_test_transformed, svr_test_predictions)
# Difference between training and test MSE
svr_mse_difference = svr_train_mse - svr_test_mse
# Difference between train and test MSE
svr_mse_difference= svr_train_mse - svr_test_mse
print("SVR Train MSE:", svr_train_mse)
print("SVR Test MSE:", svr_test_mse)
print("SVR Train RMSE:", svr_train_rmse)
print("SVR Test RMSE:", svr_test_rmse)
print("SVR Train MAE:", svr_train_mae)
print("SVR Test MAE:", svr_test_mae)
print("SVR Train R^2:", svr_train_r2)
print("SVR Test R^2:", svr_test_r2)
print("SVR Train-Test MSE Difference:", svr_mse_difference)
```

Best Model:Random Forest : Feature Importance Values

24

```python
# Train the RandomForestRegressor model with the best parameters
rf_model_best = RandomForestRegressor(**best_params_rf)
rf_model_best.fit(X_train_scaled, Y_train_log)
feature_names = ['Length', 'Diameter', 'Height', 'Shucked_Weight_Ratio',
                 'Viscera_Weight_Ratio', 'Shell_Weight_Ratio', 'Sex_F', 'Sex_I', 'Sex_M']

# Get feature importances from the model
feature_importances = rf_model_best.feature_importances_
# Combine feature names with their respective importances
feature_importances_dict = dict(zip(feature_names, feature_importances))

# Print feature importances
for feature, importance in feature_importances_dict.items():
    print(f"Feature: {feature}, Importance: {importance}")
```

PDP plot for Random Forest Model

25

```python
#Partial Dependence Plot for Random Forest Model(Best Model)
from sklearn.inspection import PartialDependenceDisplay
features=[0,1,2,3,4,5,6,7,8]
fig,ax=plt.subplots(figsize=(12,14))
ax.set_title("Random Forest")
PartialDependenceDisplay.from_estimator(rf_model_best,X_train_scaled,
                                        features,target=Y_train_log,ax=ax,feature_names=feature_names)
```

Q2

26

```python
# Define conditions for age groups
conditions = [
    (crabdata['Age'] < 10),
    (crabdata['Age'] >= 10) & (crabdata['Age'] <= 18),
    (crabdata['Age'] > 18)
]

# Define corresponding age group labels
age_labels = ['Young', 'Adult', 'Old']

crabdata['AgeGroup'] = np.select(conditions, age_labels, default=np.nan)

# Convert AgeGroup to categorical type with specified levels
crabdata['AgeGroup'] = pd.Categorical(crabdata['AgeGroup'], categories=age_labels, ordered=True)

# Print the updated DataFrame
print(crabdata)

crabdata.info()
```

27

```python
#data on different scales. so we standardize
from sklearn.preprocessing import StandardScaler
#split data to features and label by making a copy of each
X=crabdata[["Sex","Length","Diameter","Height","Weight","Shucked Weight", "Viscera Weight","Shell Weight"]].copy()
Y=crabdata["AgeGroup"].copy()
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

categorical_cols = ["Sex"]
numerical_cols = ["Length", "Diameter", "Height", "Weight", "Shucked Weight", "Viscera Weight", "Shell Weight"]

# Define preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Define the pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Fit and transform the data
X_processed = pipeline.fit_transform(X)
print(X_processed)
```

28

```python
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets with balanced classes
# Use stratify=Y to ensure that the class distribution is maintained in both sets
X_train_scaled, X_test_scaled, Y_train, Y_test = train_test_split(X_processed, Y,
                                                    train_size=0.8,
                                                    random_state=123,
                                                    stratify=Y)

# Check the splits
print(f"Train size: {round(len(X_train_scaled) / len(X_processed) * 100)}
%, Test size: {round(len(X_test_scaled) / len(X_processed) * 100)}%")

# Calculate the percentages of each category in the train set
train_percentages = Y_train.value_counts(normalize=True) * 100

# Calculate the percentages of each category in the test set
test_percentages = Y_test.value_counts(normalize=True) * 100

print("Train Set:")
print(train_percentages)
print("\nTest Set:")
print(test_percentages)

# Define class weights so that imbalance is addressed
class_weights = {'Young': 1, 'Adult': 1, 'Old': 9}
```

Q2 – Model Fitting

29

```python
#DECISION TREE
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Decision Tree classifier
dt_classifier = DecisionTreeClassifier(class_weight=class_weights)

# Train the classifier
dt_classifier.fit(X_train_scaled, Y_train)

# Evaluate the classifier
dt_pred = dt_classifier.predict(X_test_scaled)

# Accuracy
dt_accuracy = accuracy_score(Y_test, dt_pred)

print("Decision Tree Accuracy:", dt_accuracy)

# Classification report
print(classification_report(Y_test, dt_pred))

# Confusion matrix
print("Confusion Matrix (Decision Tree):")
print(confusion_matrix(Y_test, dt_pred))
```

30

```python
#RANDOM FOREST
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Random Forest with class weights
random_forest = RandomForestClassifier(class_weight=class_weights)

# Train the classifiers
random_forest.fit(X_train_scaled, Y_train)

# Evaluate the classifiers
random_forest_pred = random_forest.predict(X_test_scaled)

# Accuracy
random_forest_accuracy = accuracy_score(Y_test, random_forest_pred)

print("Random Forest Accuracy:", random_forest_accuracy)

# Classification report
print(classification_report(Y_test, random_forest_pred))

# Confusion matrix
print("Confusion Matrix(Random Forest):")
print(confusion_matrix(Y_test, random_forest_pred))
```

Applying Ridge regression before fitting Logistic Regression & SVM

31

```python
# Define preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Define Ridge regression as a preprocessing step
ridge_preprocessor = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('ridge', Ridge(alpha=1.0))
])

# Fit the Ridge preprocessor on the training data
ridge_preprocessor.fit(X_train, Y_train)

# Transform the training and testing data
X_train_processed = ridge_preprocessor.named_steps['preprocessor'].transform(X_train)
X_test_processed = ridge_preprocessor.named_steps['preprocessor'].transform(X_test)


# Check the splits
print(f"Train size: {round(len(X_train_processed) / len(X) * 100)}
%, Test size: {round(len(X_test_processed) / len(X) * 100)}%")
```

32

```python
from sklearn.linear_model import LogisticRegression

# Define class weights
class_weights = {0:1, 1:9, 2:1}

# Logistic Regression with class weights
logistic_regression = LogisticRegression(class_weight=class_weights)

# Fit Logistic Regression to the preprocessed training data
logistic_regression.fit(X_train_processed, Y_train)

# Predict on the testing data
logistic_pred = logistic_regression.predict(X_test_processed)
# Evaluate the model performance
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Accuracy
accuracy = accuracy_score(Y_test, logistic_pred)
print("Logistic Regression Accuracy:", accuracy)

# Classification report
print(classification_report(Y_test, logistic_pred))

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(Y_test, logistic_pred))
```

33

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Support Vector Machine with class weights
svm_classifier = SVC(class_weight=class_weights)

# Train the classifiers
svm_classifier.fit(X_train_processed, Y_train)

# Evaluate the classifiers
svm_pred = svm_classifier.predict(X_test_processed)

# Accuracy
svm_accuracy = accuracy_score(Y_test, svm_pred)

print("Support Vector Machine Accuracy:", svm_accuracy)

# Classification report
print(classification_report(Y_test, svm_pred))

# Confusion matrix
print("Confusion Matrix(SVM):")
print(confusion_matrix(Y_test, svm_pred))
```