Module Code	: CSMAI21
Module Name	: Artificial Intelligence and Machine Learning
Convenor Name	: Dr. Yevgeniya Kovalchuk
Assignment Report Title	: Predicting Age of Abalones
Student Number	: 30841681
Student Name	: Rashmil Sinha
Date	: 07-Mar-2023

Abstract

This project aims to use regression models to predict the age of abalone shells. These shells have rings on them, the number of which can be used to predict how old or young the shells are. The dataset has been taken from UCI repository and uses two regression models and one deep learning model based on regression to predict the age using the number of rings present on each shell. The data preprocessing pipeline standardises the dataset and separates the variables into numerical and categorical variables. The results of these models are pretty similar, where Support Vector Regression is better at fitting the data and giving a higher R_2 score, and lower mean squared error. For deep learning, we used a sequential model wrapped in a regression estimator.

Background and Problem to be Addressed

Abalones are a type of shellfish that are abundantly found along the coastal waters of Australia and New Zealand, to name a few. Predicting age of abalones is a process used by researchers for the sole purpose of determining their monetary value as age of the shellfish directly relates to the price.

Abalones have rings on their shells that can be used to determine their age, but this process is highly complex and needs a specialised environment to be carried out. To further complicate matters, the inner shell of the abalone forms the rings. These rings grow gradually, almost like the human body aging, where the growth rate of a ring is one per year. The process of counting these inner rings is done by cutting down the outer rings of the abalone and using a dye to stain and polish them, making the process of counting the rings relatively easier.

Since it is not possible to completely stain all the rings, researchers decided to approximate the ring count by adding a factor of ‘1.5’ to the number of rings that could be determined to predict an approximate age. This is done to predict a reasonably close age estimate to gauge the correct price for each abalone, which helps abalone farmers to sell it at a reasonable price.

Unfortunately, this is a time-consuming process, along with being largely inaccurate due to human error. To combat this, my problem is to predict the age of abalones using machine learning models by utilising the physical characteristics such as weight, height, sex, etc. I will be

building regression models that can predict the age faster and with more accuracy than the conventional process, thereby reducing time and cost (Guney *et al.*, 2022).

Exploratory Data Analysis

Exploratory data analysis or EDA is used to decipher patterns in the dataset for further analysis and model building. A fundamental reason for performing EDA on datasets is to pick the best preprocessing methods for the data and the best analysis tools.

The Abalone dataset has been taken from the UCI repository (*UCI Machine Learning Repository: Abalone Data Set*, 2019). The data was added from a non-machine learning study conducted in Australia by marine biologist researchers for a population study of abalones. A brief description of the dataset attributes is given below –

- Number of Instances: 4177
- Number of Attributes: 8
- Attribute information: Given is the attribute name, attribute type, the measurement unit, and a brief description. The number of rings is the value to predict either as a continuous value or as a classification problem.

Name	Data Type	Measurement	Description
Sex	nominal		M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer	+1.5 years	age of abalone, determined by adding 1.5 to the value

We have performed EDA on the Abalone dataset to pick the best features for any dimensionality reduction that might be done later, to check for age distribution and also find the distribution of ages across the various attributes in the ‘Sex’ column. EDA is also performed to find any null/missing values and any duplicates present in the dataset. Figures 1-6 describe the summary and statistics of the dataset, along with printing any duplicates/missing values.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('../data/df.csv')
df['Age'] = df['Rings'] + 1.5 #add a new column for 'Age' to the existing dataset

#Print the first five rows
df.head()
```

✓ 0.1s

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Age
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	16.5
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	8.5
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	10.5
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	11.5
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	8.5

Figure 1 - Loading the dataset and printing first 5 rows

```python
# Check the shape of the dataset
df.shape
```

✓ 0.0s

```
(4177, 10)
```

Figure 2 - Shape of the dataset

```python
# Check the data types of the columns
df.dtypes
```

✓ 0.2s

```
Sex               object
Length           float64
Diameter         float64
Height           float64
Whole weight     float64
Shucked weight   float64
Viscera weight   float64
Shell weight     float64
Rings              int64
Age              float64
dtype: object
```

Figure 3 - Datatypes

```
# Check for missing values
df.isna().sum()
```

✓ 0.0s

```
Sex                0
Length             0
Diameter           0
Height             0
Whole weight       0
Shucked weight     0
Viscera weight     0
Shell weight       0
Rings              0
Age                0
dtype: int64
```

Figure 4 - Check for missing values

```
# Check for duplicates in the dataset
df.duplicated().sum()
```

✓ 0.0s

```
0
```

Figure 5 - Check for duplicates

```
# Summary statistics for the dataset
df.describe()
```

✓ 0.1s

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Age
count	4177.000000	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	11.433684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000	2.500000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000	9.500000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000	10.500000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000	12.500000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000	30.500000

Figure 6 - Statistics of the dataset

A histogram is the best way to group all the numerical columns and get their distribution. The ‘Diameter’, ‘Length’ and the ‘Age’ columns are the closest to a normal distribution among the numerical columns. Figures 7 and 8 show the two histograms.

```
# Select only the numerical columns
num_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Plot histograms of all numerical variables
df[num_cols].hist(bins=20, figsize=(15, 10))
plt.show()
```

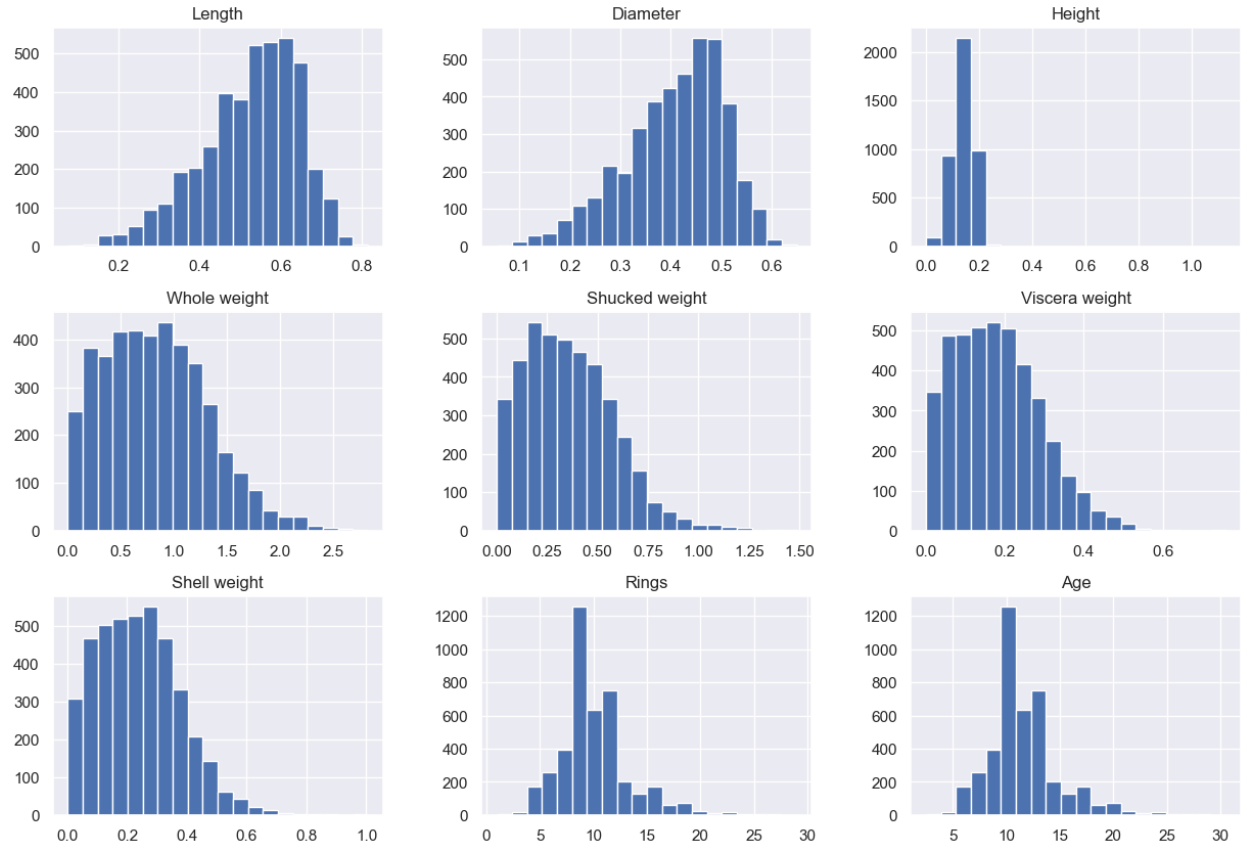

Figure 7 - Histogram of numerical columns

```
# Plot the distribution of the target variable 'Age'
sns.histplot(df['Age'], bins=20, kde=True)
plt.title("Distribution of Age")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()
```

✓ 2.5s

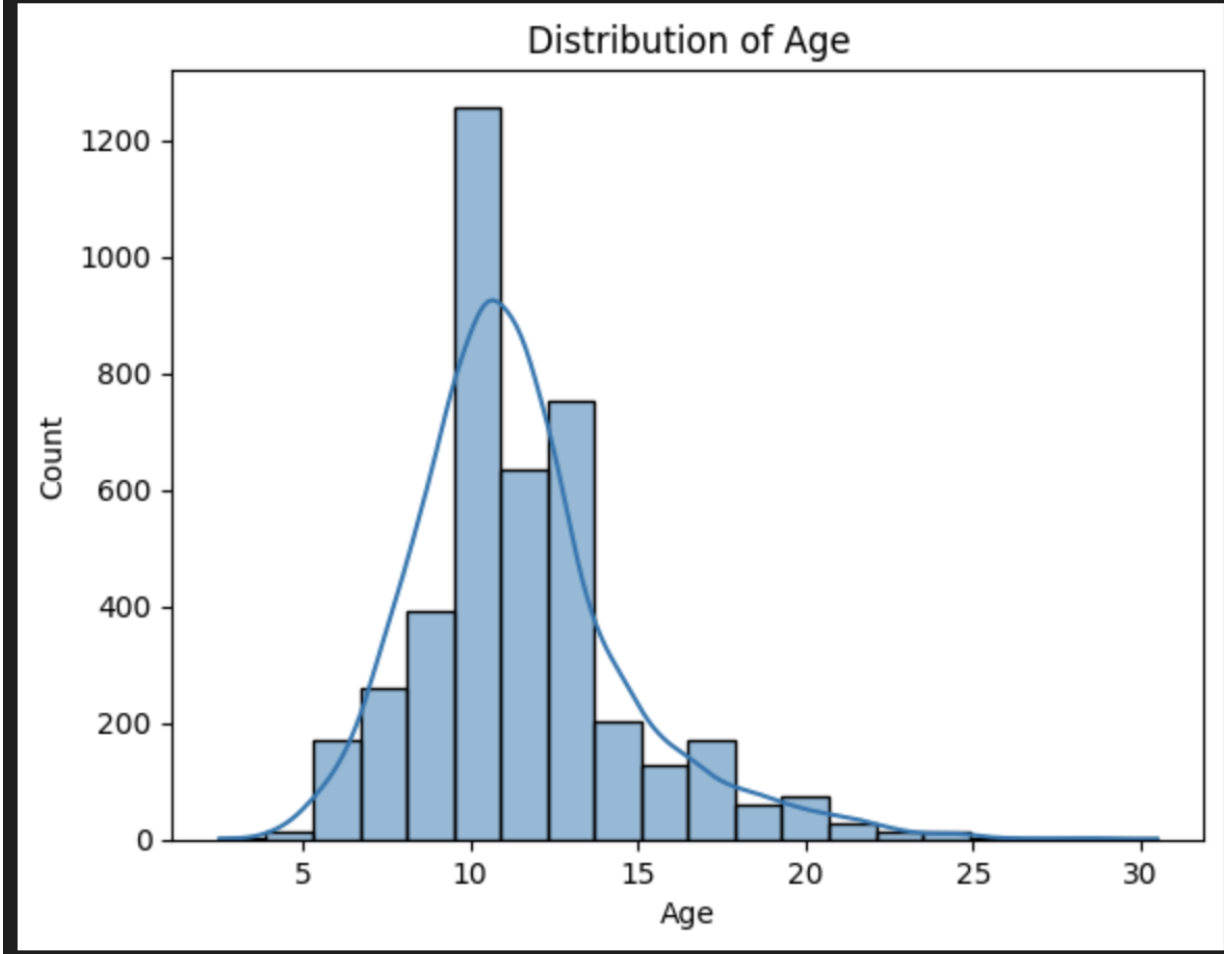

Figure 8 - Distribution of the target variable Age

In the same vein, a boxplot is good to detect any outliers that may be present in the dataset to help us handle it later on in the preprocessing section of the model building. 'Height' column seems to contain the worst outliers and is marked for preprocessing. Figures 9 - 11 are used to depict the boxplots.

```
# Plot boxplots for each feature variable
num_cols = ['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell weight', 'Age']

plt.figure(figsize=(12,8))
for i, col in enumerate(num_cols):
    plt.subplot(4, 2, i+1)
    sns.histplot(df[col], kde=False, color='red')
    plt.title(col)
plt.tight_layout()
plt.show()
```

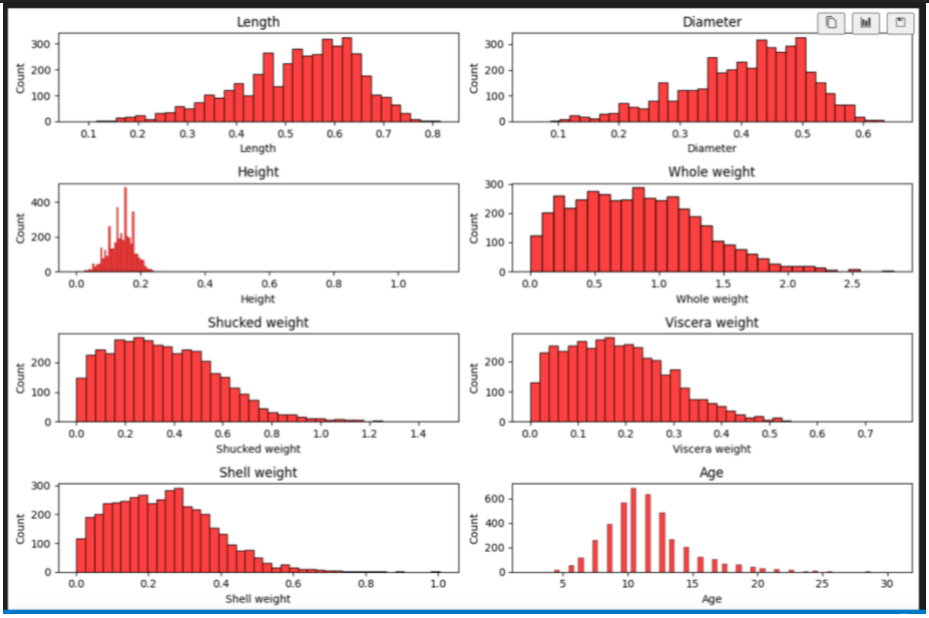

Figure 9 - Boxplots of numerical variables

```
# plot boxplots for each feature against the target variable age
fig, axes = plt.subplots(4, 2, figsize=(25,25))
sns.boxplot(ax=axes[0,0], x='Age', y='Age', data=df)
sns.boxplot(ax=axes[0,1], x='Age', y='Length', data=df)
sns.boxplot(ax=axes[1,0], x='Age', y='Diameter', data=df)
sns.boxplot(ax=axes[1,1], x='Age', y='Height', data=df)
sns.boxplot(ax=axes[2,0], x='Age', y='Whole weight', data=df)
sns.boxplot(ax=axes[2,1], x='Age', y='Shucked weight', data=df)
sns.boxplot(ax=axes[3,0], x='Age', y='Viscera weight', data=df)
sns.boxplot(ax=axes[3,1], x='Age', y='Shell weight', data=df)
plt.show()
```

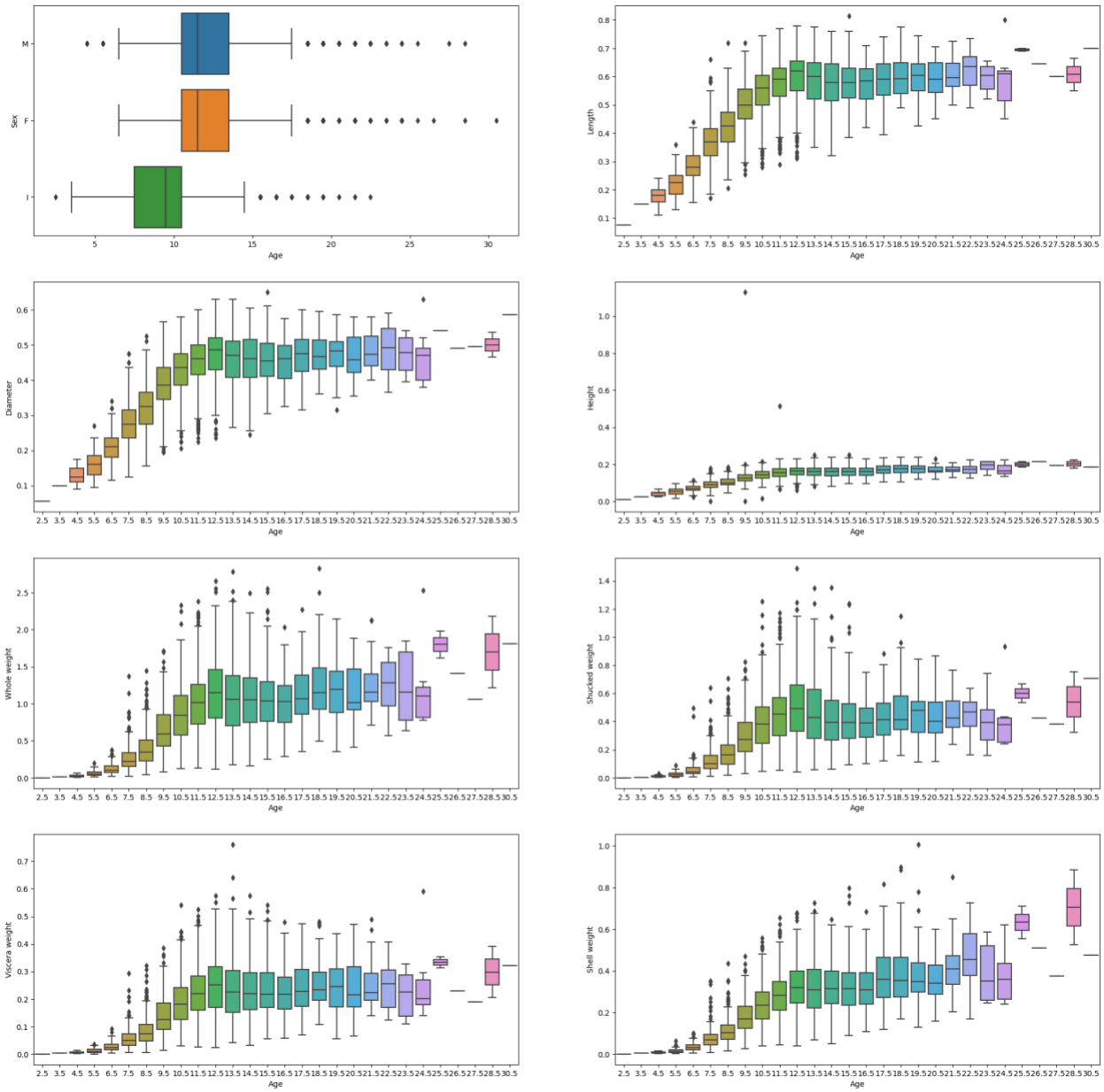

Figure 10 - Boxplots of each numerical variable against target variable

```
# Box plot of the target variable 'age'
sns.boxplot(data=df, x='Age')
plt.show()
```

✓ 0.2s

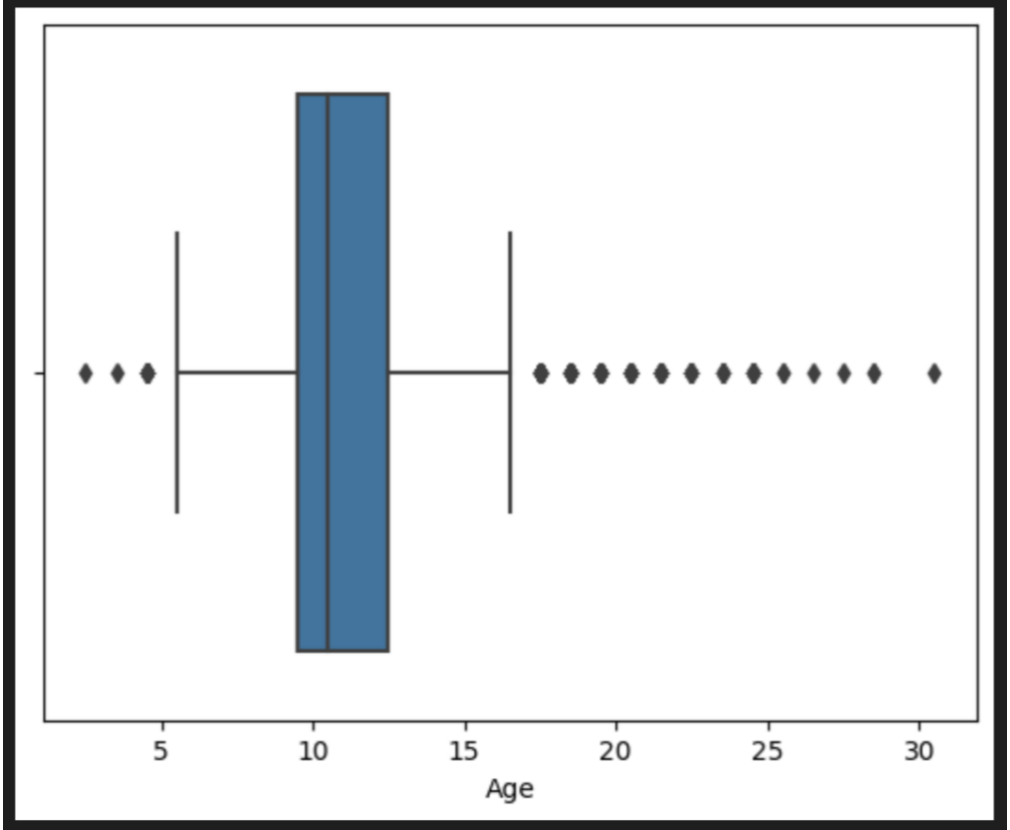

Figure 11 - Boxplot of the target

Boxplots are also a good way to visualize relationships between various features. We can use a correlation matrix to create a heatmap to find which columns are similar and can be dropped during feature selection process. Figure 11 depicts the correlation matrix.

```
# Correlation matrix plot
corr = df.corr()
sns.heatmap(corr, cmap='coolwarm', annot=True)
plt.show()
```

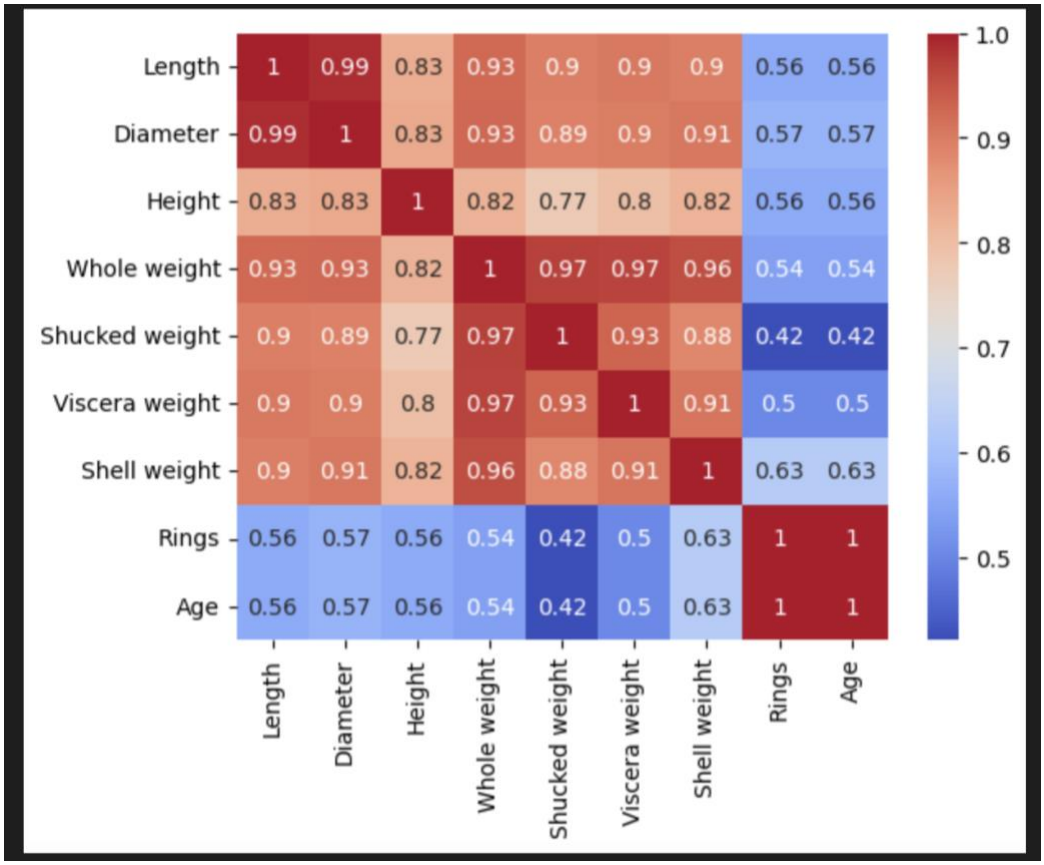

Figure 12 - Correlation Matrix

We have also made use of a pairplot to plot pairwise relationships between the various numerical columns and another for all the columns. The fundamental reason for performing this function is to become familiar with the data and its various attributes. A scatterplot is also built for a similar function. The entire reason for exploratory data analysis is to gather knowledge of the dataset to help further down the project, when performing more complex operations on the dataset. Figures 13-15 show the above plots.

```
# Scatter plot of 'length' and 'diameter' with hue='Age'
sns.scatterplot(data=df, x='Length', y='Diameter', hue='Age')
plt.show()
```

✓ 0.4s

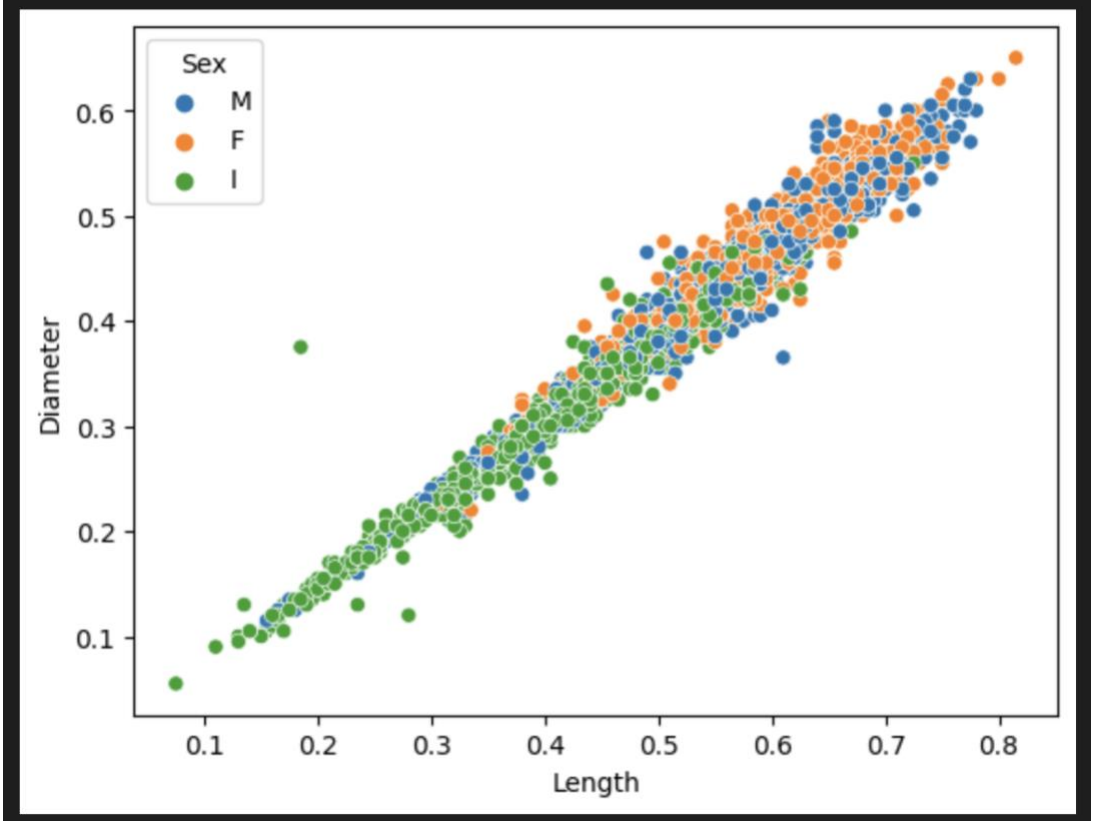

Figure 13 - Scatter plot showing distribution of length vs Diameter

```
# Pair plot of selected numerical columns
sns.pairplot(data=df[['Length', 'Diameter', 'Height', 'Whole weight', 'Age']])
plt.show()
```

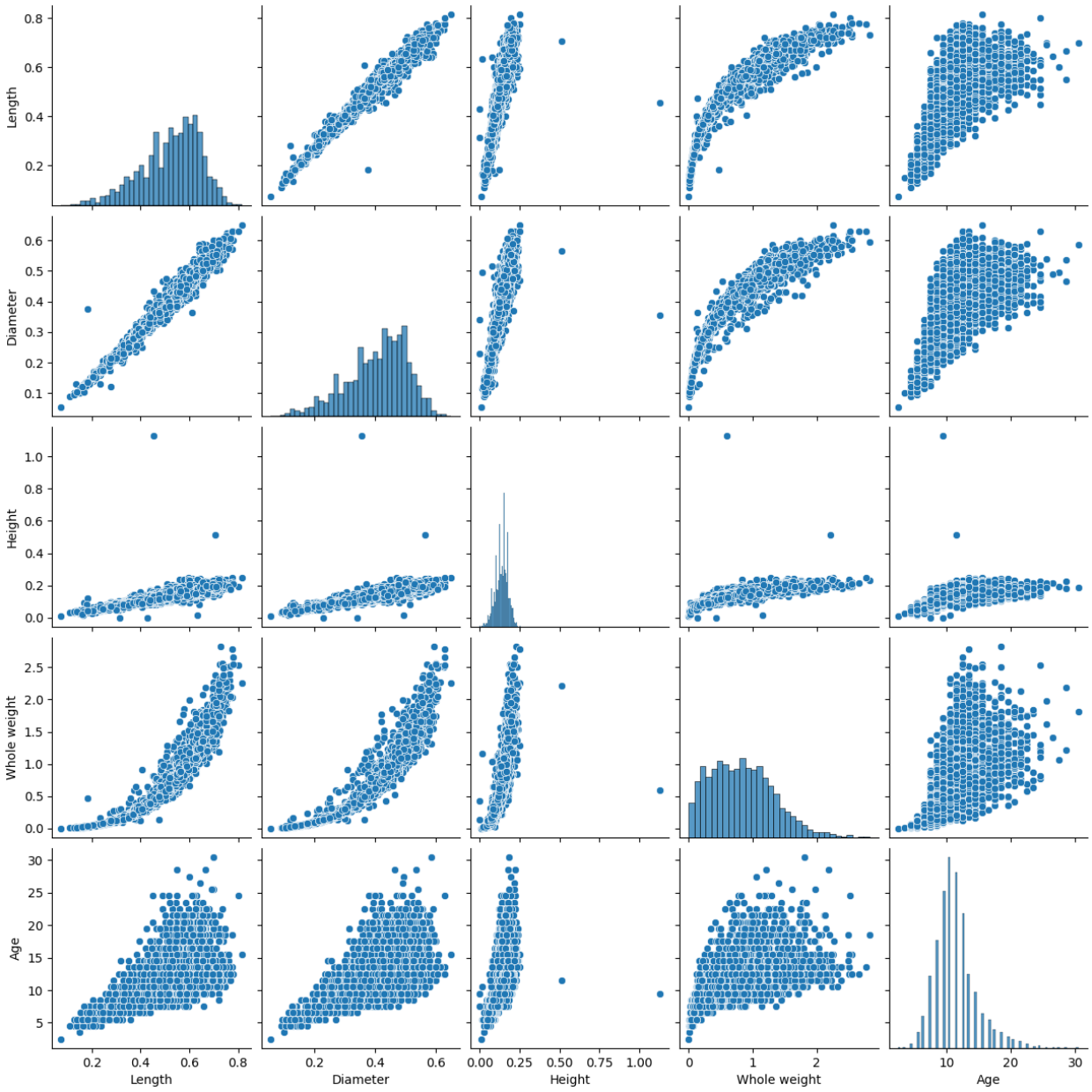

Figure 14 - Pairplot of selected numerical features

```
# Plot the scatterplot matrix
sns.pairplot(df, diag_kind='hist', hue='Sex')
plt.title("Scatterplot Matrix")
plt.show()
```

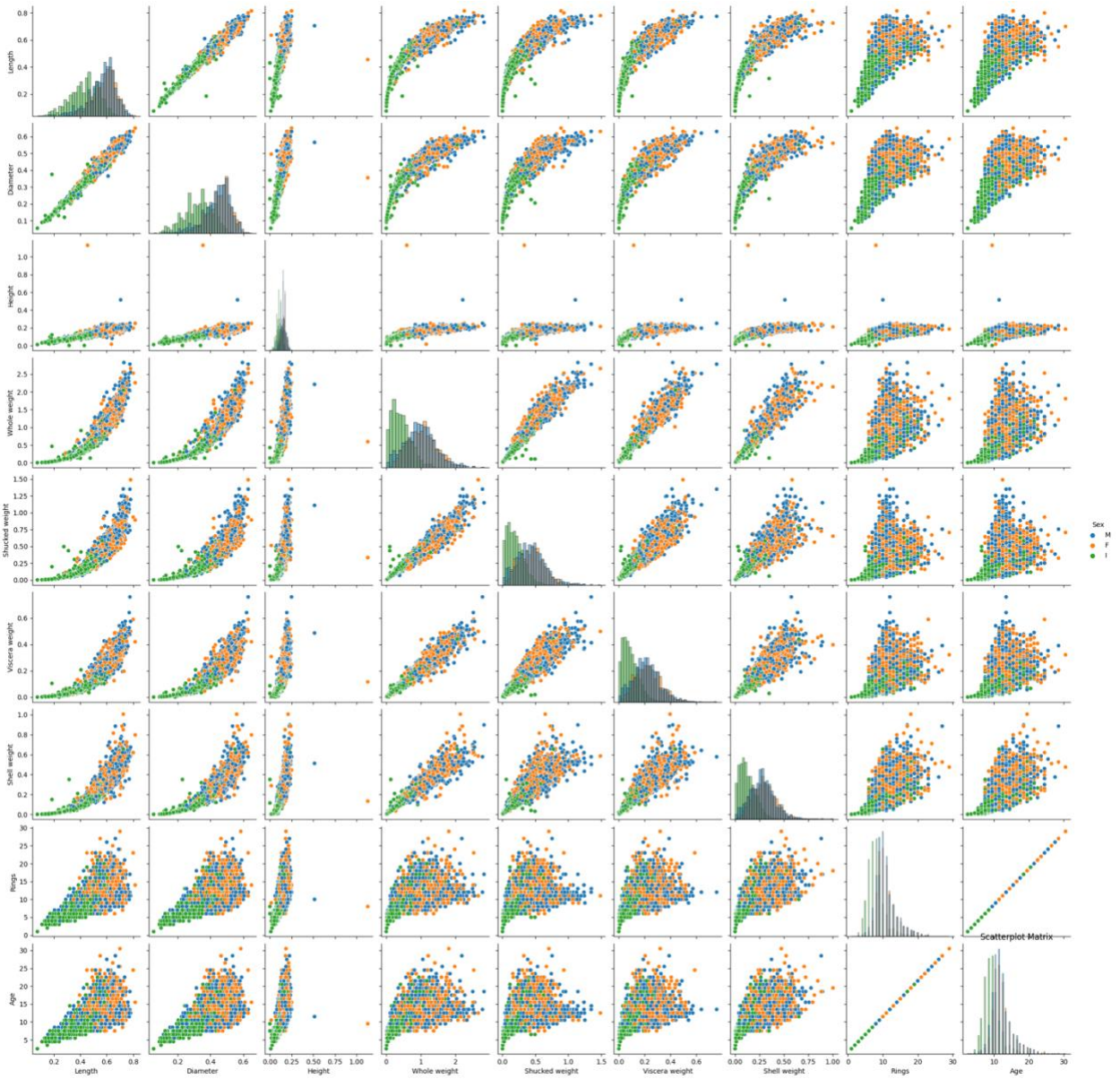

Figure 15 - Pairplot of all columns

A density plot can also be used. This is done to find the skewness of the dataset (Figure 16).

```
# Plot density plot with skewness
sns.set(style="darkgrid")
fig, ax = plt.subplots(figsize=(8,6))
sns.histplot(df['Age'], ax=ax, color='green')
plt.title('Density plot with skewness for age')
plt.xlabel('Age')
plt.ylabel('Density')
plt.show()
```

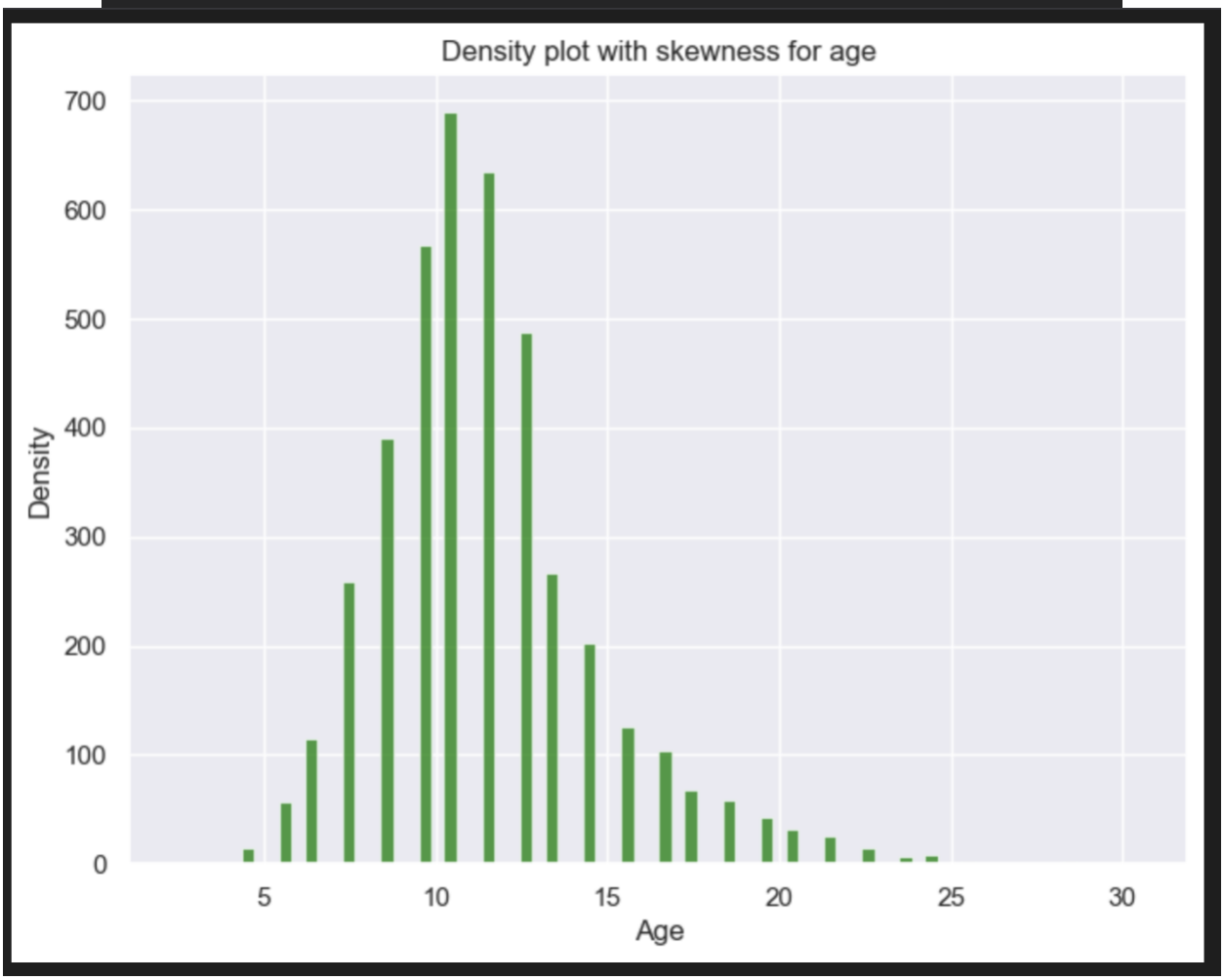

Figure 16 - Density plot

Data Pre-processing and Feature Selection

To get the data ready for regression models, we performed EDA previously to check for null and duplicated values first (Figures 4 and 5) to be handled before beginning analysis. Outliers can be handled by various methods, with the interquartile method being a common way to handle it. But after some research and experiments on the dataset, it was found that removing outliers caused

the prediction metrics to drop in performance. Since there is a categorial column present (‘Sex’), one-hot encoding was performed for better analysis.

In the case of data imbalance, it is typically not required to be handled for regression datasets/problems. Being a regression dataset, the abalone dataset contains physical measurements of the shell, and the target variable is ‘Age’, a continuous variable.

An exploratory data analysis earlier proved that the mean age of the abalone shells is at approximately 9 years old (Figure 8). It is roughly a normal distribution, slightly skewing higher. Hence, we can conclude that there is no need to perform any special data imbalance handling, owing to the fact that the data imbalance is insignificant.

It was decided to select the most relevant features to reduce dimensionality as there are very many columns that correspond to similar physical attributes, and they don’t need to be added to the training-test dataset splits. After adding feature selection in the pipeline and running the models, it was found that feature selection also reduced the performance of the models. The choice to avoid adding outlier handling and feature selection in the data preprocessing pipeline is further explored in the model selection and results section. As hyperparameters tuning is being performed, there is no requirement to create a separate validation dataset as cross validation is performed including selection of best hyperparameters. Figure 16 depicts the code snippet of the data preprocessing pipeline.

```
# Define the preprocessing steps for numerical and categorical features
numerical_features = ['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell weight']
categorical_features = ['Sex']

numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)])

# Prepare the data
X = df.drop(['Rings', 'Age'], axis=1)
y = df['Age']

print('Preprocessing and data prep done. Splitting the data set into train-test splits.')

# Split the data into train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 17 - Data preprocessing pipeline

Another preprocessing technique that is employed further down in the model building stage of the project is to convert the train-test split that is initially a dataframe to a NumPy array. This is done because tensorflow works better on NumPy arrays. We also remove the categorical variable ‘Sex’ from this new NumPy array to further better the performance of the deep learning sequential model. Figure 17 shows the code written to perform this preprocessing technique.

```
# remove the 'Sex' column from the training set for DL model and convert the dataframe to a numpy array for ease of use
X_train = preprocessor.fit_transform(X_train, y_train)
X_test = preprocessor.transform(X_test)
X_train = np.array(X_train).astype('float32')
X_test = np.array(X_test).astype('float32')
X_train = np.delete(X_train, 3, axis=1) # remove the 'Sex' column
X_test = np.delete(X_test, 3, axis=1) # remove the 'Sex' column
```

Figure 18 - Removing categorical column and converting dataframe to NumPy array

Machine Learning Models

This section explores the various machine learning models (classical and deep learning) used for this project. It gives a brief description of each model and the tuning done and evaluation metrics used to gauge performance on the abalone dataset. The results section gives an overview of the performance metrics of the models.

- ## Random Forest Regressor

Summary of the approach

Random Forest Regressor is a specialised Decision Tree algorithm that uses multiple decision trees to provide better prediction results. This algorithm has a low computational overhead, tried to avoid model overfitting and is easy for implementation. The feature importance provided in this model is useful for understanding the influence each attribute has on the dataset (Chang *et al.*, 2023).

Random Forest Regressor algorithm can make predictions on the dataset even if outliers are present in individual features. Outliers that are present in the abalone dataset are useful for prediction purposes, hence the decision was taken to not remove them as they were found to be legitimate data points. The reason to leave the outliers is explained in the ‘Results’ section below. The way Random Forest handles outliers is by using multiple decision trees to reduce the impact of each individual outlier and this doesn’t affect the prediction.

EDA also told us that the various attributes in this dataset do not share any linear relationships and random forest regressor can handle non-linear relationships between the target and feature variables, as ‘Age’ isn’t linearly related to the categorical attribute such as ‘Sex’.

Random Forest Regressor performs marginally well on this dataset, even though the R2 score isn’t the highest although the error rate can be considered good.

Model training and evaluation

For training the Random Forest Regressor model, we made use of hyperparameter tuning using GridSearchCV instead of manually choosing parameters and tuning them until the best prediction result comes forth. GridSearchCV finds the best parameters for machine learning models by checking all the parameters inputted in the grid. This process was

done by tweaking the parameter grid multiple times to find the best possible list of range for each parameter used to build the model. GridSearchCV also performs cross-validation, so that invalidates the need for using a separate validation dataset to ensure the model performs well on any chosen dataset. We inputted ‘n_estimators’, ‘max_depth and ‘min_samples_split’ as the parameters to build a parameter grid, there are other options which can also be added or substituted to get a more robust model. This can be a future consideration, although it may possibly increase computation time and complexity. A 5-fold cross validation was performed on the dataset, to ensure better results. 10-fold increases the computation needed and tuning the model also takes longer, which isn’t always necessary for smaller datasets. Figure 19 gives the config file for Random Forest Regressor.

```yaml
model:
  RandomForestRegressor:
    n_estimators: 200
    max_depth: 10
    min_samples_split: 15

hyperparameters:
  RandomForestRegressor:
    n_estimators: [50, 100, 200]
    max_depth: [5, 10, 15]
    min_samples_split: [5, 10, 15]
```

Figure 19 - Random Forest Regressor Config

The hyperparameters we added in the parameter for tuning the model are as mentioned below:

- n_estimators: [50, 100, 200]
- max_depth: [5, 10, 15]
- min_samples_split: [5, 10, 15]

The amount of decision trees that will be constructed in the random forest is indicated by the term ‘n_estimators’. The efficiency of the model improves as the number of trees increases. If there are too many additions to the branches of the trees, this could result in model overfitting.

The smallest number of samples needed to split an internal node is represented by the variable ‘min_samples_split’. This value can be raised to avoid overfitting. If you set

min_samples_split=5, for instance, it indicates that nodes with fewer than 5 samples shouldn't be split.

‘max_depth’ displays the deepest decision tree in each random forest. By default, it is set to "None," which causes the nodes to expand until all of the leaves have fewer samples than the bare minimum needed to divide (‘min_samples_split’). Overfitting can be avoided, and the decision trees' depth can be managed by setting ‘max_depth’. Figure 20 is the code snippet of the model built for this project.

```
print('Starting Random Forest Regressor.')

# Define the random forest model pipeline
rf_model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor())
])

# Define the hyperparameters to tune
param_grid_rf = {
    'regressor__n_estimators': rf_config['hyperparameters']['RandomForestRegressor']['n_estimators'],
    'regressor__max_depth': rf_config['hyperparameters']['RandomForestRegressor']['max_depth'],
    'regressor__min_samples_split': rf_config['hyperparameters']['RandomForestRegressor']['min_samples_split']
}

# Perform grid search cross validation
rf_model_cv = GridSearchCV(rf_model_pipeline, param_grid_rf, cv=5)
rf_model_cv.fit(X_train, y_train)
print('Random Forest Model Best Parameters:', rf_model_cv.best_params_)

# Use the best model to predict on the test set
rf_model_pred = rf_model_cv.predict(X_test)
rf_model_mse = mean_squared_error(y_test, rf_model_pred)
rf_model_r2 = r2_score(y_test, rf_model_pred)
print('Random Forest Model MSE:', rf_model_mse)
print('Random Forest Model R2 Score:', rf_model_r2)

# Save the random forest model and its predictions
joblib.dump(rf_model_cv.best_estimator_, '../results/rf_model.joblib')
np.savetxt('../results/rf_pred.csv', rf_model_pred)

# Plot the random forest model predictions
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=rf_model_pred)
plt.plot([0, 30], [0, 30], color='red')
plt.xlabel('True Age')
plt.ylabel('Predicted Age')
plt.title('Random Forest Model Predictions')
plt.savefig('../results/rf_predictions.png')
plt.show(block=False)
```

Figure 20 - Random Forest Regressor

To evaluate any regression model, the measures of R2 score and Mean Squared Error (MSE) are frequently utilised, and we have done the same for evaluating our models, as well. The MSE was used to calculate the average squared difference between the predicted and real values, and the r2 score was used to calculate the percentage of the target variable's variance that was explained by the model. As this is a regression model,

it is understood that many predictions would fall either over or under the true prediction which is depicted in the plot shown in the next section.

Results and discussion

As mentioned above, Random Forest Regressor is a good model for this particular dataset as it can handle non-linearity between the target and the feature variables and also account for outliers in it, without the need to explicitly handle them. When we added feature selection and outlier handling in the data preprocessing pipeline, the R2 score and the MSE greatly reduced as the outliers contained required data points and removing them and removal of important features caused the model to underfit, signifying the model was unable to capture the variability of the data.
The hyperparameter values of n_estimators = 200, max_depth = 10, min_samples_split = 15 gives us an R2 score of 0.55 and an MSE of 4.84. Figure 21 gives a brief overview of the score and the error rate, and figure 22 shows the plot to help us understand how many predictions are close to the true predictions (the red line). As this is a regression model, it is expected that many predictions will fall over and under the line.

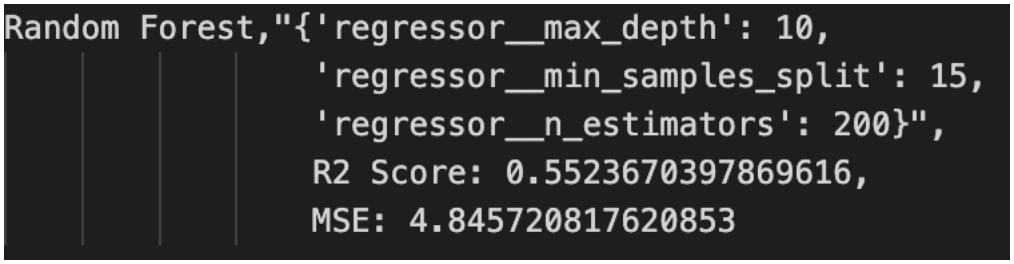

Figure 21 - Overview of the best hyperparameters and R2 score and MSE

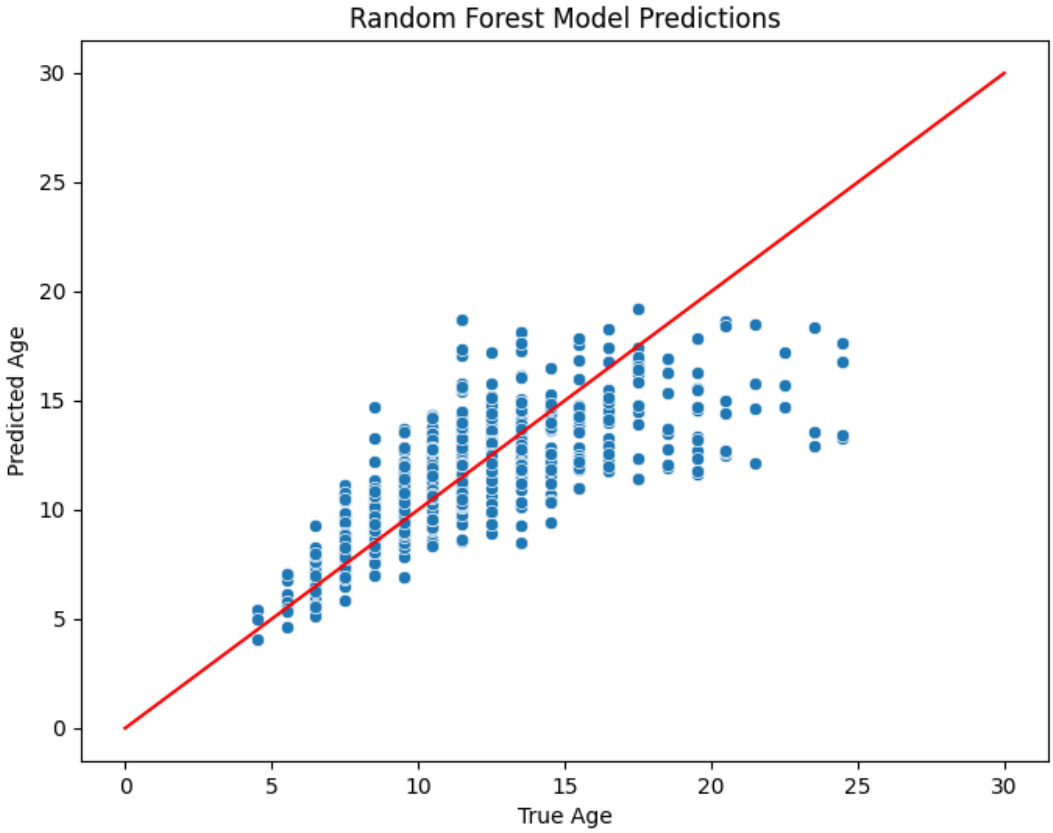

Figure 22 - Random Forest Regressor predicted age vs true age plot

The R2 score indicates Random Forest Regressor could explain a variance of 54% in the test dataset whereas the error rate is signifying the model is off predicting the age of the abalone by 4.84 years on average. It is considered a moderate fit for the models, but that's usually the case in regression problems. The low R2 score is expected for a dataset that is not too complex as we only did 5-fold cross validation, using a 10-fold cv might increase the variance of the performance measures, thereby skewing the metrics. The error rate is kept fairly low which tells us that the Random Forest Regressor is a good model for predicting the age of abalone, in spite of a lower R2 score.

- ## Support Vector Regression

Summary of the approach

Support Vector Regression is also a supervised machine learning model for regression problems. It is also efficient in handling high dimensional data and performs well. The use of a 'kernel' as one of the parameters in the SVR is beneficial as it can classify data linearly using a set of training samples which are called support vectors. This is good for regression problems which output continuous data (Zhang and O'Donnell, 2020).

The handling of non-linear relationships between the target and the feature variables is important for choosing the model and Support Vector Regression does that well. Both continuous and categorical features are present in the Abalone dataset, and some of these features have a nonlinear association with the target variable, ‘Age’.

Through the use of kernel functions, SVR is able to transform the input features into a higher dimensional space where a hyperplane can better separate them, allowing it to record such nonlinearities. SVR is a versatile model that can be adjusted to enhance efficiency. It has a number of hyperparameters, including the regularization parameter, epsilon-insensitive loss function, kernel type, and kernel parameter, which can be adjusted using techniques like GridSearchCV to get the optimal model performance.

SVR can also manage big datasets and is robust to outliers. Over 4000 examples make up the moderately sized Abalone dataset, and some of its features contain outliers. SVR can deal with these situations by reducing the impact of outliers on the final model.

For the abalone dataset, the manual method of predicting the age of the shellfish is tedious and already has lower accuracy with a higher error rate. We decided to choose the SVR model because even though it is data dependent and high dimensionality matters, the error rate is fairly low, and the model performs well on this unpredictable dataset.

Model training and evaluation

As used above in Random Forest, for training the Support Vector Regression model, we made use of hyperparameter tuning using GridSearchCV instead of manually choosing parameters and tuning them until the best prediction result comes forth. GridSearchCV finds the best parameters for machine learning models by checking all the parameters inputted in the grid. This process was done by tweaking the parameter grid multiple times to find the best possible list of range for each parameter used to build the model. GridSearchCV also performs cross-validation, so that invalidates the need for using a separate validation dataset to ensure the model performs well on any chosen dataset. We inputted ‘C’, ‘gamma’ and ‘kernel’ as the parameters to build a parameter grid. These hyperparameters were picked because research has indicated that they have a big effect on performance. Again, a 5-fold cross validation was performed during model tuning for giving the best possible results. Figure 23 is the config file for the SVR model.

```yaml
model:
  SVR:
    kernel: rbf
    C: 10.0
    gamma: 0.1

hyperparameters:
  SVR:
    C: [0.1, 1.0, 10.0]
    kernel: ['linear', 'rbf', 'poly']
    gamma: [0.01, 0.1, 1]
```

Figure 23 - SVR config

The hyperparameters used are described below:

- C: [0.1, 1.0, 10.0]
- kernel: ['linear', 'rbf', 'poly']
- gamma: [0.01, 0.1, 1]

The flexibility of the model can be achieved by regulating ‘C’, which is the regularisation parameter that managed the trade-off between obtaining a low training error and a low testing error. A lesser ‘C’ value results in a wider margin and allows some misclassifications, whereas a larger ‘C’ value produces a tighter margin and reduces misclassifications. In essence, ‘C’ determines the ideal overfitting to underfitting ratio.

The decision boundary's shape is determined by ‘gamma’. A larger ‘gamma’ value produces a more complex and tighter curve whereas a smaller ‘gamma’ value results in a smoother and wider curve.

The ‘kernel’ of the algorithm determines what kind of kernel function should be used. The kernel function maps the input data into a higher dimensional feature space with a linear decision boundary. Radial Basis Function (RBF), sigmoid, linear, and polynomial are widely used kernel functions.

As we can see from figure 23, the config file has a section to perform SVR without hyperparameter tuning and a section using GridSearchCV. The config file was updated to add the former section after hyperparameter tuning found the best parameters that gave the optimal result. Figure 24 below is a snippet of the model building.

```
print('Starting SVR.')

# Define the SVR model pipeline
svr_model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', SVR())
])

# Define the hyperparameters to tune
param_grid_svr = {
    'regressor__C': svr_config['hyperparameters']['SVR']['C'],
    'regressor__kernel': svr_config['hyperparameters']['SVR']['kernel'],
    'regressor__gamma': svr_config['hyperparameters']['SVR']['gamma']
}

# Perform grid search cross validation
svr_model_cv = GridSearchCV(svr_model_pipeline, param_grid_svr, cv=5, n_jobs=-1)
svr_model_cv.fit(X_train, y_train)
print('SVR Model Best Parameters:', svr_model_cv.best_params_)

# Use the best model to predict on the test set
svr_model_pred = svr_model_cv.predict(X_test)
svr_model_mse = mean_squared_error(y_test, svr_model_pred)
svr_model_r2 = r2_score(y_test, svr_model_pred)
print('SVR Model MSE:', svr_model_mse)
print('SVR Model R2 Score:', svr_model_r2)

# Save the SVR model and its predictions
joblib.dump(svr_model_cv.best_estimator_, '../results/svr_model.joblib')
np.savetxt('../results/svr_pred.csv', svr_model_pred)

# Plot the SVR model predictions
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=svr_model_pred)
plt.plot([0, 30], [0, 30], color='red')
plt.xlabel('True Age')
plt.ylabel('Predicted Age')
plt.title('SVR Model Predictions')
plt.savefig('../results/svr_predictions.png')
plt.show(block=False)
```

Figure 24 - SVR model building code

To assess the effectiveness of the model, we used the R2 score and mean squared error. The MSE was used to calculate the average squared difference between the predicted and real values, and the r2 score was used to calculate the percentage of the target variable's variance that was explained by the model. The figures in the next section give a brief overview of the score and the error rate, and the plot also helps us understand how many predictions are close to the true predictions (the red line). As this is a regression model, it is expected that many predictions will fall over and under the line.

Results and discussion

The SVR model with hyperparameters of C=10, gamma=0.1, and kernel=rbf achieved an R2 score of 0.56 and an MSE of 4.71. This suggests that the SVR model was able to explain 56% of the variance in the target variable, which is a moderate fit for the model. The MSE of 4.71 indicates that the model's predictions were off by about 4.71 years on average. Figure 25 gives the evaluated metrics and figure 26 shows a plot of the model.

```
SVR,"{'regressor__C': 10.0,
      'regressor__gamma': 0.1,
      'regressor__kernel': 'rbf'}",
      R2 Score: 0.5646443047630817,
      MSE: 4.712816845469386
```

Figure 25 - SVR metrics and hyperparameters

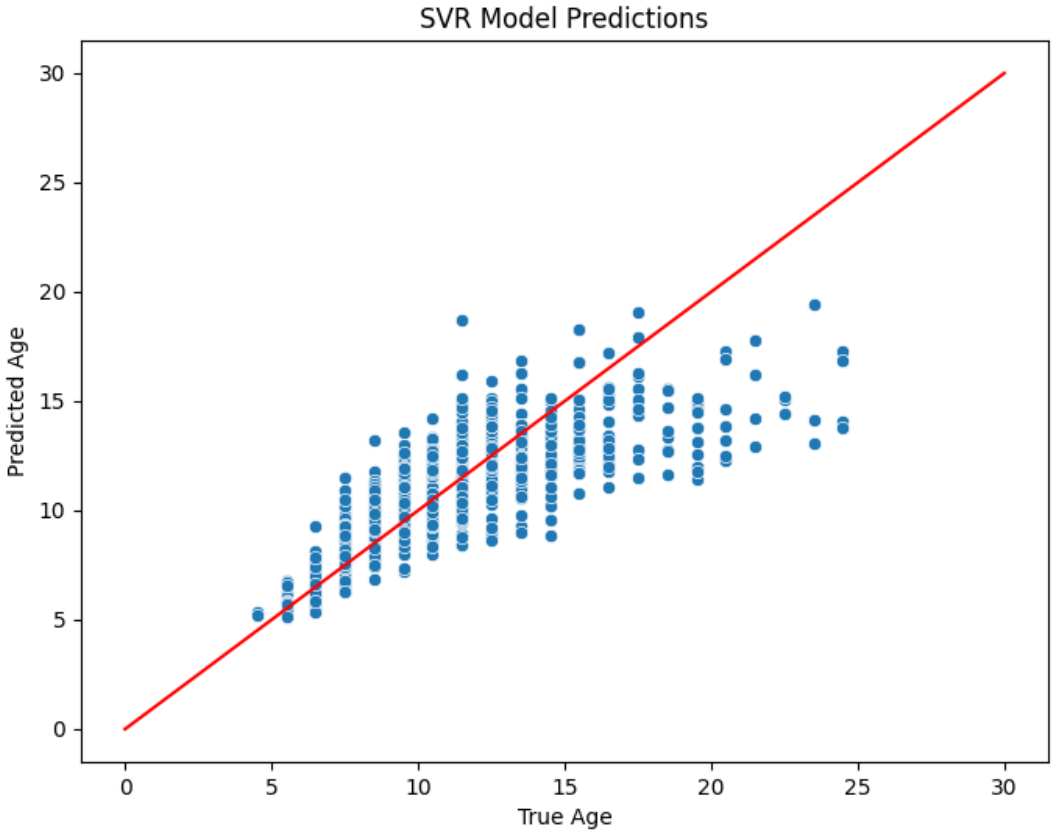

Figure 26 - SVR predicted age vs true age

The R2 score of the SVR model is slightly better than Random Forest Regressor model, suggesting SVR performs better on the abalone dataset. This is possible because SVR is a

powerful regression model when it comes to non-linear relationships and how it's handled by the parameters. It should also be mentioned that SVR's robust handling of outliers is better in comparison with other classical models. The outliers present in the abalone dataset affect the prediction, so proper handling of it will increase any model's variability and reduce their error rate, as we have seen in the case of SVR.

Another advantage of using SVR is that it is easier to understand because being an ensemble model, random forest regressor contains multiple decision trees, each having their own parameter grid.

- ## KerasRegressor (Regression based Deep Learning Model)

Summary of the approach

Deep Neural Network-based predictive models are popular for solving both classification and regression problems. This particular dataset is a regression problem and to solve it, a deep neural network sequential model is built using a Keras wrapper for regression. This wrapper is called a KerasRegressor, and it is a regression estimator (Manjula and Vijaya, 2020).

The KerasRegressor deep learning sequential model is good for the abalone dataset because it has the ability to learn complex non-linear relationships between the input features and the target variable. The dataset chosen has a large number of input features, which can be complicated for traditional linear regression models to accurately capture the underlying relationships between the features and target variable. It uses multiple layers of interconnected nodes to learn these relationships in a hierarchical fashion. Each layer transforms the input data into a new representation that is more abstract than the previous layer. The final layer outputs the predicted target value. This hierarchical approach allows the model to learn complex non-linear relationships between the input features and target variable, which is necessary to accurately predict the target variable.

Additionally, the KerasRegressor model allows for easy customisation of the number of layers, the number of nodes per layer, and the activation functions used in each layer. This flexibility allows for the model to be easily tuned to find the optimal architecture for the specific dataset being used. The easy customisation of building the model was a major factor in choosing this particular sequential model.

Model training and evaluation

We chose to perform GridSearchCV for hyperparameter tuning, as was done with the previous two models, to identify the best combination of parameters for better prediction accuracy.

The grid search was conducted with 5-fold cross validation, and the R-squared score and mean squared error (MSE) were used as evaluation metrics. The following hyperparameters were tested:

- Optimiser: Adam and SGD (Stochastic Gradient Descent)
- Number of hidden layers: 1 and 2
- Number of neurons: 32, 64, and 128

The config file is shown below (Figure 27).

```yaml
model:
  KerasRegressor:
    epochs: 100
    batch_size: 10
    verbose: 1
    optimizer: Adam
    hidden_layers: 2
    neurons: 32

hyperparameters:
  KerasRegressor:
    optimizer:
      - Adam
      - SGD
    hidden_layers: [1, 2]
    neurons: [32, 64, 128]
```

Figure 27- KerasRegressor config

The neural network's optimiser parameter determines the optimisation strategy to be utilised during training. Stochastic Gradient Descent (SGD), Adam, RMSprop, Adagrad, etc. are a few examples of optimisers. To enhance model performance, each optimiser has a unique collection of hyperparameters that can be adjusted.

The number of hidden layers in the neural network is specified by this parameter. The intricacy of the model and its capacity to identify intricate correlations in the data depend on how many hidden layers are there. Deeper networks, in general, have the ability to learn more complicated characteristics but may also be more susceptible to overfitting.

The ‘neurons’ parameter indicates the count of total neurons present in each hidden layer of the neural network. The number of learnable features in each layer is based on the number of neurons in that layer. In general, adding more neurons to a layer can aid in the model learning more intricate features, but it also raises the possibility of overfitting. In many cases, a hyperparameter that can be adjusted during model selection is the number of neurons in each layer.

A neural network's training process in Keras is described by the term ‘epoch’, which is defined as a complete iteration over the entire training dataset.

‘batch_size’ refers to the volume of samples the model will analyse at once. Instead of doing so after processing the full dataset, the model weights are modified during training after each batch to give more flexibility to model building.

Figures 28 and 29 show the model building code for Keras sequential model.

```
print('Starting Keras Regressor.')

# define a function to build the Keras regressor model, adding an input_shape that corresponds to the changes in the preprocessed
# data, including the removal of 'Sex' column to ensure the code runs smoothly
def build_regressor_model(optimizer=Adam(), hidden_layers=1, neurons=64, input_shape=None):
    model = Sequential()
    model.add(Dense(units=neurons, input_shape=input_shape, activation='relu'))
    for _ in range(hidden_layers - 1):
        model.add(Dense(units=neurons, activation='relu'))
    model.add(Dense(units=1, activation='linear'))
    model.compile(loss='mse', optimizer=optimizer)
    return model

# create the KerasRegressor object
regressor = KerasRegressor(build_fn=build_regressor_model, epochs=100, batch_size=10, input_shape=(X_train.shape[1],))

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

# define the parameter grid for the grid search
param_grid = {
    'optimizer': dl_config['hyperparameters']['KerasRegressor']['optimizer'],
    'hidden_layers': dl_config['hyperparameters']['KerasRegressor']['hidden_layers'],
    'neurons': dl_config['hyperparameters']['KerasRegressor']['neurons']
}

# create the GridSearchCV object and perform GridSearch, print the best hyperparameters
dl_model_cv = GridSearchCV(estimator=regressor,
                           param_grid=param_grid,
                           cv=5,
                           verbose=2,
                           n_jobs=-1)
dl_model_cv.fit(X_train, y_train)
print("Best Hyperparameters: ", dl_model_cv.best_params_)
```

Figure 28 - Defining the Keras model

```
# Use the best model to predict on the test set
dl_model_pred = dl_model_cv.predict(X_test)
dl_model_mse = mean_squared_error(y_test, dl_model_pred)
dl_model_r2 = r2_score(y_test, dl_model_pred)
print('Keras Regressor Model MSE:', dl_model_mse)
print('Keras Regressor Model R2 Score:', dl_model_r2)

# Save the DL model and its predictions
dl_model_cv.best_estimator_.model.save('../results/dl_model.h5')
np.savetxt('../results/dl_pred.csv', dl_model_pred)

# Plot the dl model predictions
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=dl_model_pred)
plt.plot([0, 30], [0, 30], color='red')
plt.xlabel('True Age')
plt.ylabel('Predicted Age')
plt.title('DL Model Predictions')
plt.savefig('../results/dl_predictions.png')
plt.show(block=False)

# Fit the model and get the history object
history = pipeline['regressor'].fit(X_train, y_train, validation_split=0.2, verbose=0)

# Plot the training and validation loss
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Epoch vs Loss')
plt.savefig('../results/dl_val.png')
plt.show(block=False)
```

Figure 29 - Evaluation metrics and plots

It is worth noting that the KerasRegressor model had a significantly longer training time compared to other models tested in this study. However, similar performance of the model justifies the additional computational cost, and it can be postulated that by fine tuning the model more by adding additional parameters to the grid, this model can perform better than the classical machine learning models.

Overall, the KerasRegressor model with the identified hyperparameters appears to be a promising approach for predicting the age of abalone based on physical measurements.

Results and discussion

The best combination of hyperparameters for the KerasRegressor sequential model was found to be Adam optimizer, 1 hidden layer, and 128 neurons. We decided to add a

batch_size of 10 and train the model for 100 epochs to update the weight after each batch is run. The model achieved a R2 score of 0.54 and an MSE of 4.94 on the test set.

Figure 30 gives the evaluation metrics and figures 31 and 32 depict the various plots.

```
KerasRegressor,"{'hidden_layers': 1,
                'neurons': 128,
                'optimizer': 'Adam'}",
               R2 Score: 0.5434642880507332,
               MSE: 4.9420949751489776
```

Figure 30 - Keras evaluation metrics and hyperparameters

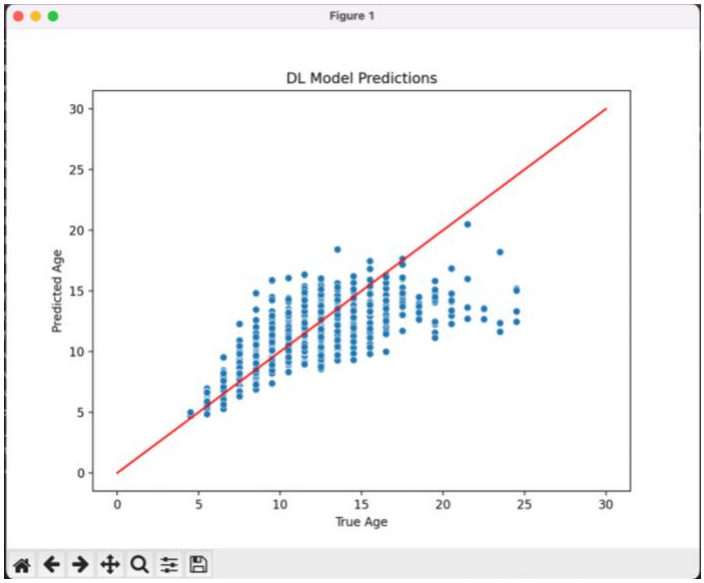

Figure 31 - Keras predicted age vs true age plot

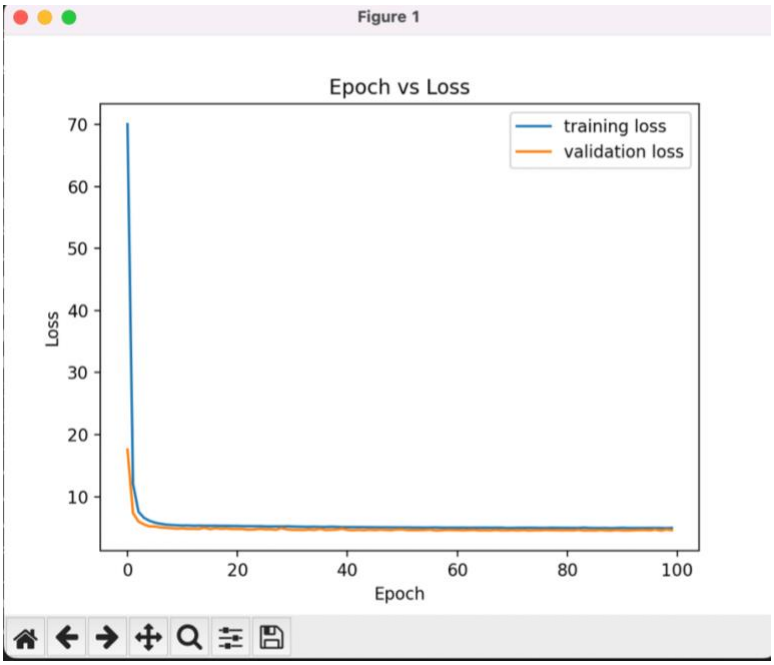

Figure 32 - Keras training loss vs validation loss

This suggests that the KerasRegressor model was able to explain 54% of the variance in the target variable, which is also a moderate fit for the model. The MSE of 4.93 indicates that the model's predictions were off by about 4.93 years on average.

It is important to note that the KerasRegressor model is a deep learning model, which needs to be tuned frequently and requires the use of more computational resources than traditional machine learning models.

Results comparison across the models built

We chose three models - Random Forest Regressor, SVR, and KerasRegressor - to predict the age of abalone using regression. For all the models, we chose not to add a random seed for reproducibility as there was a slight possibility for that to affect the prediction metrics but adding a random seed might be useful when conducting more experiments with more models.

The Random Forest Regressor model achieved an R2 score of 0.55 and an MSE of 4.84 with hyperparameters n_estimators=200, max_depth=10, and min_samples_split=15.

The SVR model achieved a slightly better R2 score of 0.56 and a slightly lower MSE of 4.71 with hyperparameters C=10, gamma=0.1, and kernel=rbf.

The KerasRegressor model achieved an R2 score of 0.54 and an MSE of 4.94 with hyperparameters optimizer=Adam, hidden_layers=1, and neurons=128.

Figure 33 shows a code snippet to calculate and plot a comparison of the three models and their true error rates. The difference is minimal amongst all three models. Figure 34 shows a comparison of all three models and their true error rates.

```
# Load the true ages and the predicted ages from each model
true_ages = y_test
rf_pred = np.loadtxt('../results/rf_pred.csv')
svr_pred = np.loadtxt('../results/svr_pred.csv')
dl_pred = np.loadtxt('../results/dl_pred.csv')

# Compute the errors for each model
rf_error = true_ages - rf_pred
svr_error = true_ages - svr_pred
dl_error = true_ages - dl_pred

# Create a boxplot to compare the errors of each model
plt.figure(figsize=(10,6))
sns.boxplot(data=[rf_error, svr_error, dl_error], showmeans=True)
plt.xticks(np.arange(3), ['Random Forest', 'SVR', 'Keras Regressor'])
plt.xlabel('Model')
plt.ylabel('Error')
plt.title('Error Comparison')
plt.savefig('../results/model_comparison.png')
plt.show(block=False)
```

Figure 33 - Model Comparison code

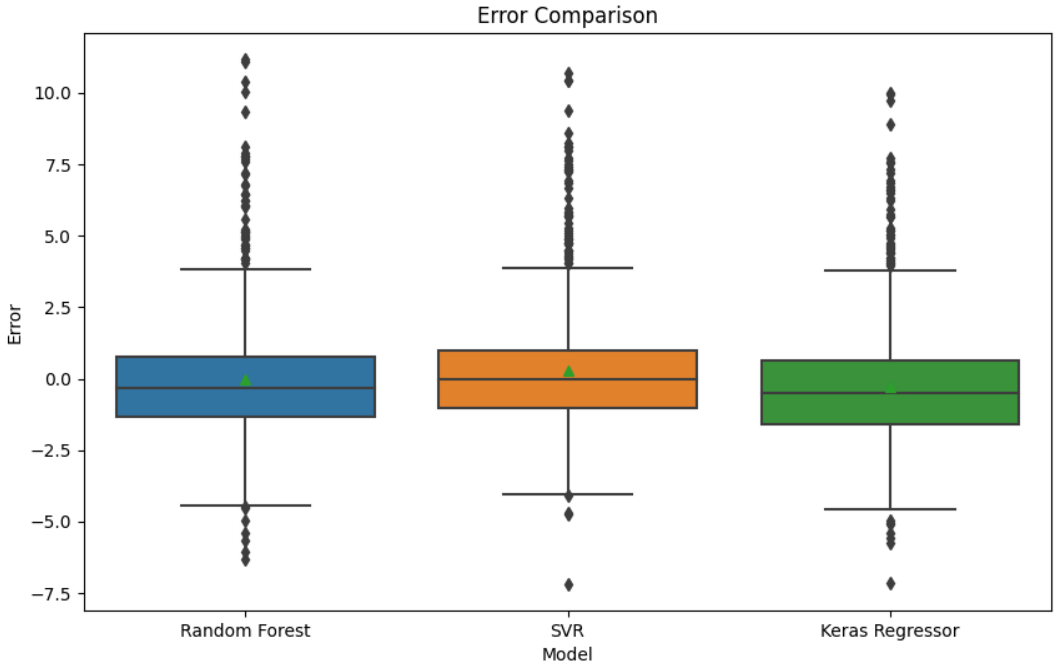

Figure 34 - Model comparison plot

It was found that each model had its own strengths and weaknesses. The Random Forest Regressor was able to handle non-linearity between the target and feature variables and account for outliers, but its performance was limited by the dataset's low complexity. On the other hand, the KerasRegressor model using deep learning techniques was able to capture the nonlinearities in the data, but it was more prone to overfitting and had a slightly lower R2 score compared to the Random Forest Regressor and Support Vector Regression.

The Support Vector Regression, with its various hyperparameters, was able to achieve the best performance among the three models with an R2 score of 0.56 and a mean squared error of 4.71. The use of the radial basis function kernel allowed for the model to capture the nonlinear relationships between the target and feature variables, while the parameter tuning with grid search allowed for an optimal combination of hyperparameters to be found. Furthermore, the Support Vector Regressor was easier to understand compared to the other models due to its mathematical simplicity and clear interpretation of the hyperparameters.

However, the differences in performance were relatively small, with all three models achieving similar results.

It is worth noting that the choice of model ultimately depends on the specific problem and the data being analyzed. In this case, all three models performed reasonably well and the choice between them could depend on factors such as interpretability, computational efficiency, and ease of implementation.

Conclusions, Recommendations and Future Work

This report presents the results of a project on predicting the age of abalone using three different machine learning models. The Support Vector Regression (SVR) model performed the best, with an R2 score of 0.56 and MSE of 4.71. The Random Forest Regressor and KerasRegressor models had slightly lower R2 scores and higher MSE values. We recommend using caution when performing data preprocessing steps that may harm the model's performance and suggest exploring other models and hyperparameters to improve performance in future work.

The project also highlights the importance of interpretability and computational efficiency when selecting a machine learning model. The SVR model was found to be the simplest and easiest to understand, while the KerasRegressor model was the most computationally expensive.

As recommended above, we want to highlight the impact of feature selection and outlier removal on model performance. The Random Forest Regressor model was found to be particularly sensitive to feature selection and outlier removal. Therefore, we reiterate using caution when performing data preprocessing steps to avoid harming the model's performance.

In conclusion, we recommend using the SVR model for predicting the age of abalone in future studies, especially when dealing with small to medium-sized datasets with a relatively low number of features.

Furthermore, we recommend investigating the relationships between the physical attributes of abalones and their growth patterns to gain more insights into the biology of these organisms. The findings of the project can be applied not only to the abalone dataset but also to other similar datasets.

Overall, the project provides valuable insights into the use of machine learning models in predicting the age of abalone and their potential applications in various industries. It is also recommended to ensure careful selection of model evaluation metrics, and hyperparameter tuning in achieving the best performance from the models.

Moreover, we suggest exploring the use of more advanced machine learning algorithms, such as deep neural networks, for predicting the age of abalone. This could provide further insights into the relationships between the physical attributes of abalones and their growth patterns. We also recommend validating the models on independent datasets to assess their generalisability and applicability in real-world scenarios. This would ensure that the models can be used effectively in various industries and provide valuable insights.

In summary, the study provides a comprehensive analysis of the use of machine learning models in predicting the age of abalone. The findings can be used to guide future research and provide a framework for selecting the best machine learning model for a given problem. The study highlights the importance of model evaluation metrics, hyperparameter tuning, and careful data preprocessing in achieving the best performance from the models.

References

Uci.edu. (2019). *UCI Machine Learning Repository: Abalone Data Set*. [online] Available at: https://archive.ics.uci.edu/ml/datasets/abalone

Guney, S., Kilinc, I., Hameed, A.A. and Jamil, A. (2022). Abalone Age Prediction Using Machine Learning. *Pattern Recognition and Artificial Intelligence*, pp.329–338. doi: 10.1007/978-3-031-04112-9_25.

Chang, X., Xing, Y., Gong, W., Yang, C., Guo, Z., Wang, D., Wang, J., Yang, H., Xue, G. and Yang, S. (2023). Evaluating gross primary productivity over 9 ChinaFlux sites based on random forest regression models, remote sensing, and eddy covariance data. *Science of The Total Environment*, [online] 875, p.162601. doi: 10.1016/j.scitotenv.2023.162601.

Zhang, F. and O'Donnell, L.J. (2020). Chapter 7 - Support vector regression. In: Mechelli, A. and Vieira, S., eds., *Machine learning: Methods and applications to brain disorders*. [online] Academic Press. Available at: https://doi.org/10.1016/B978-0-12-815739-8.00007-9.

Manjula, R. and Vijaya, M.S. (2020). Deep Neural Network for Evaluating Web Content Credibility Using Keras Sequential Model. *Lecture Notes in Electrical Engineering*, pp.11–19. doi:10.1007/978-981-15-5558-9_2.