Introduction

This study harnesses the power of Machine Learning (ML) to predict the age of abalones, a marine snail species, by analyzing their physical characteristics. Traditionally, age estimation in abalones involves counting the rings on their shells, a method that is both time-intensive and requires expertise. The objective here is to provide a quicker, non-invasive method of age estimation by applying various ML algorithms, thereby assisting in the sustainable management of abalone populations and enhancing the efficiency of the abalone industry.

Contents

Age Estimation of Abalone Using Machine Learning

Fenglin Zhang

03/16/2024

1 Data Preprocessing

1.1 Data Loading and Initial Exploration

This code block is instrumental in loading the abalone dataset and displaying its first few entries. It leverages pandas for data manipulation, and matplotlib and seaborn for preliminary data visualization. The dataset comprises several features:

- **Sex:** M, F, and I (infant)
- **Length:** Longest shell measurement
- **Diameter:** Perpendicular to length
- **Height:** With meat in shell
- **Whole weight:** Whole abalone
- **Shucked weight:** Weight of meat
- **Viscera weight:** Gut weight (after bleeding)
- **Shell weight:** After being dried
- **Rings:** +1.5 gives the age in years (target variable)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('abalone.csv')
df.head()
```

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

1.2 Statistical Summary

```
df.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

The *df.describe*() function provides a statistical summary of the dataset, offering insights into the central tendency, dispersion, and shape of the dataset's distribution.

1.3 Data Integrity Check

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Sex             4177 non-null   object
 1   Length          4177 non-null   float64
 2   Diameter        4177 non-null   float64
 3   Height          4177 non-null   float64
 4   Whole weight    4177 non-null   float64
 5   Shucked weight  4177 non-null   float64
 6   Viscera weight  4177 non-null   float64
 7   Shell weight    4177 non-null   float64
 8   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

```
df.isna().sum()
```

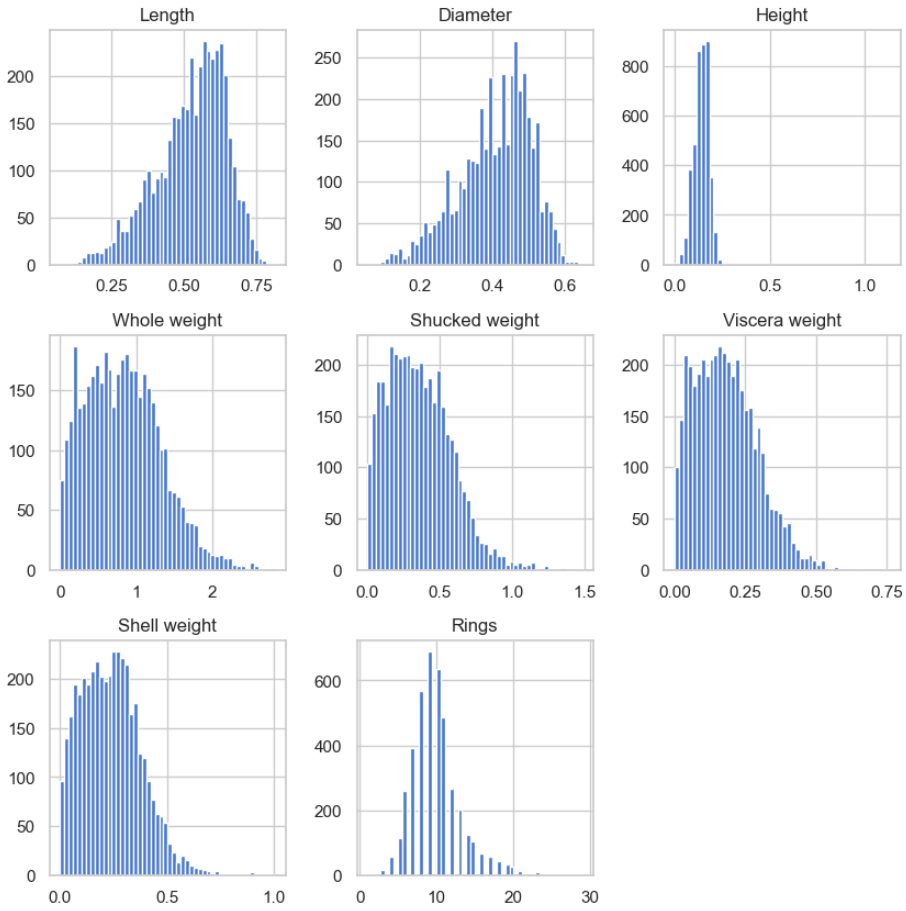
```
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked weight    0
Viscera weight    0
Shell weight      0
Rings             0
dtype: int64
```

Ensuring data integrity, $df.info()$ presents a concise summary of the dataset, while $df.isna().sum()$ checks and summarizes any missing values, ensuring the data is clean and ready for analysis.

2 Exploratory Data Analysis (EDA)

2.1 Distribution of Features

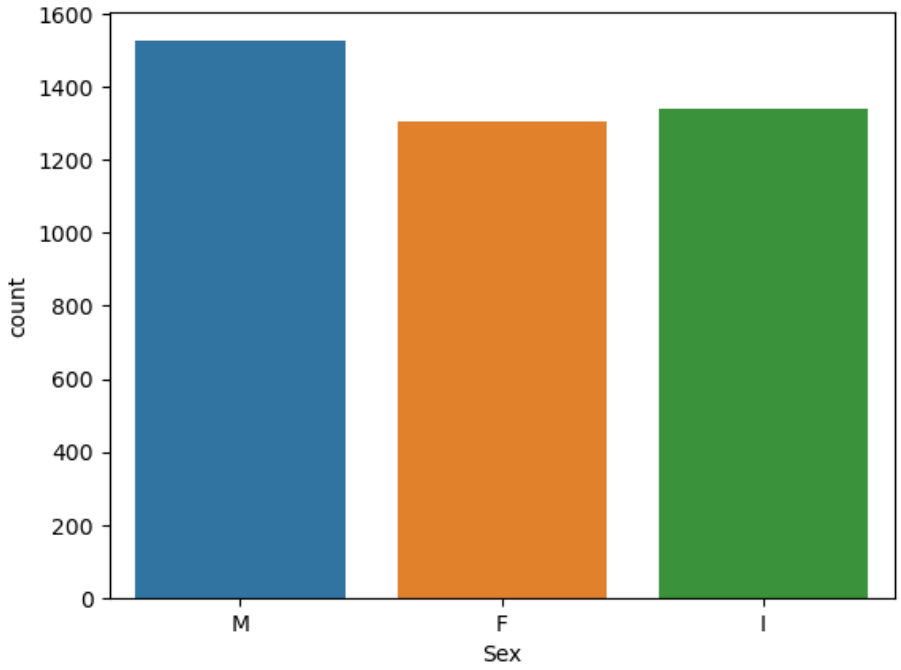
```
df.hist(bins=50, figsize=(10, 10))
plt.show()
```



This histogram plot offers a visual representation of the distribution of each numerical feature in the dataset, aiding in understanding the underlying structure of the data.

2.2 Sex Feature Analysis

```
sns.countplot(x = 'Sex', data = df)
```



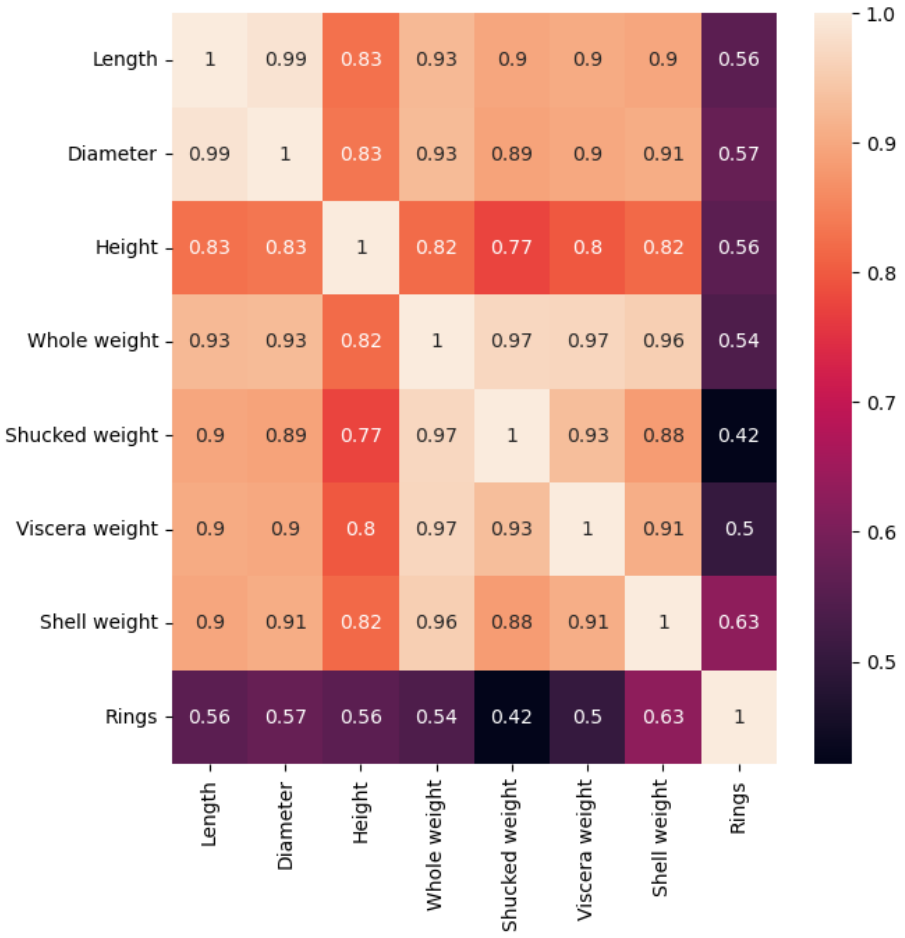
The count plot for the ‘Sex‘ feature provides a visual comparison of sample counts across different genders, integral for understanding sample distribution.

```
sns.violinplot(x='Sex', y='Rings', data=df)
plt.title('Violin Plot of Rings by Sex')
plt.xlabel('Sex')
plt.ylabel('Ringst')
```

By plotting a violin plot of the 'Rings' feature against 'Sex', this code provides a detailed view of the distribution of rings (age) across different genders.

2.3 Correlation Analysis

```
corr = df.corr(numeric_only=True)
plt.figure(figsize=(7, 7))
sns.heatmap(corr, annot=True)
```



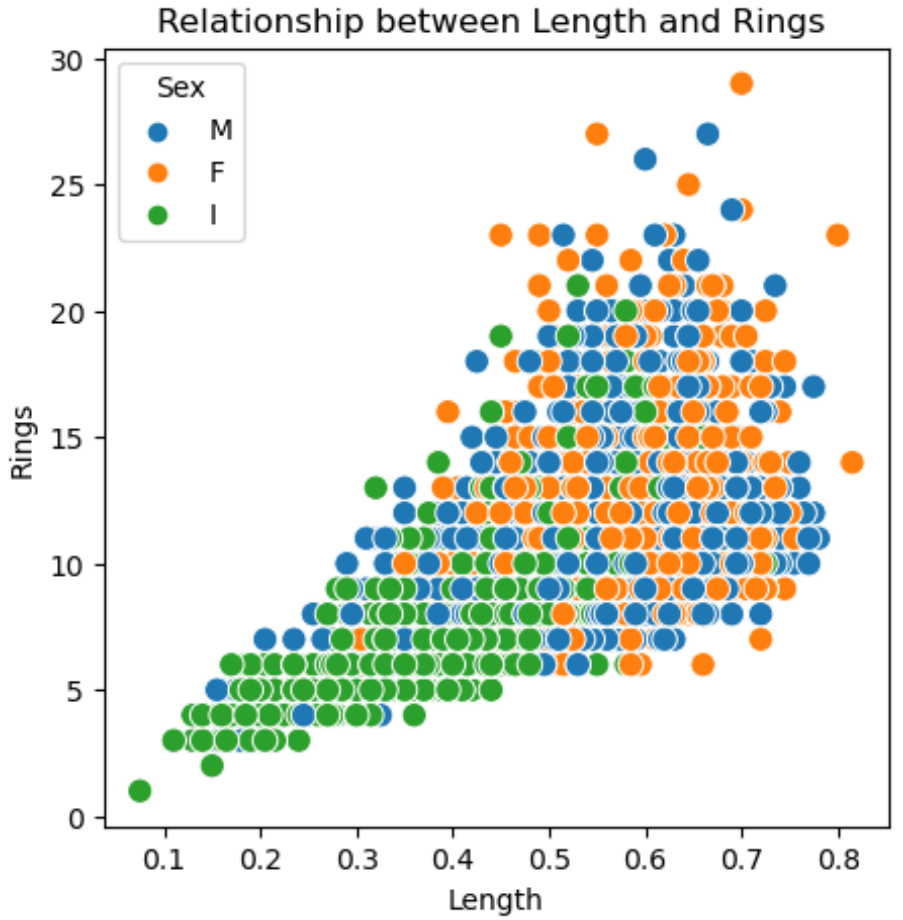
This code calculates and displays the correlation between the numerical features of the abalones, such as their physical measurements, in a heatmap. It's used to visually identify which features are most related to the abalone's age, aiding in the selection of relevant attributes for more precise age prediction models.

2.4 Relationship Between Features

The series of scatter plots is a comprehensive visual analysis of the relationship between various physical measurements of abalones (like Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, and Shell weight) and the number of Rings, which is an indicator of the abalone's age. Each plot also categorizes data points by the 'Sex' of the abalone, providing an additional layer of insight.

2.4.1 Length vs. Rings

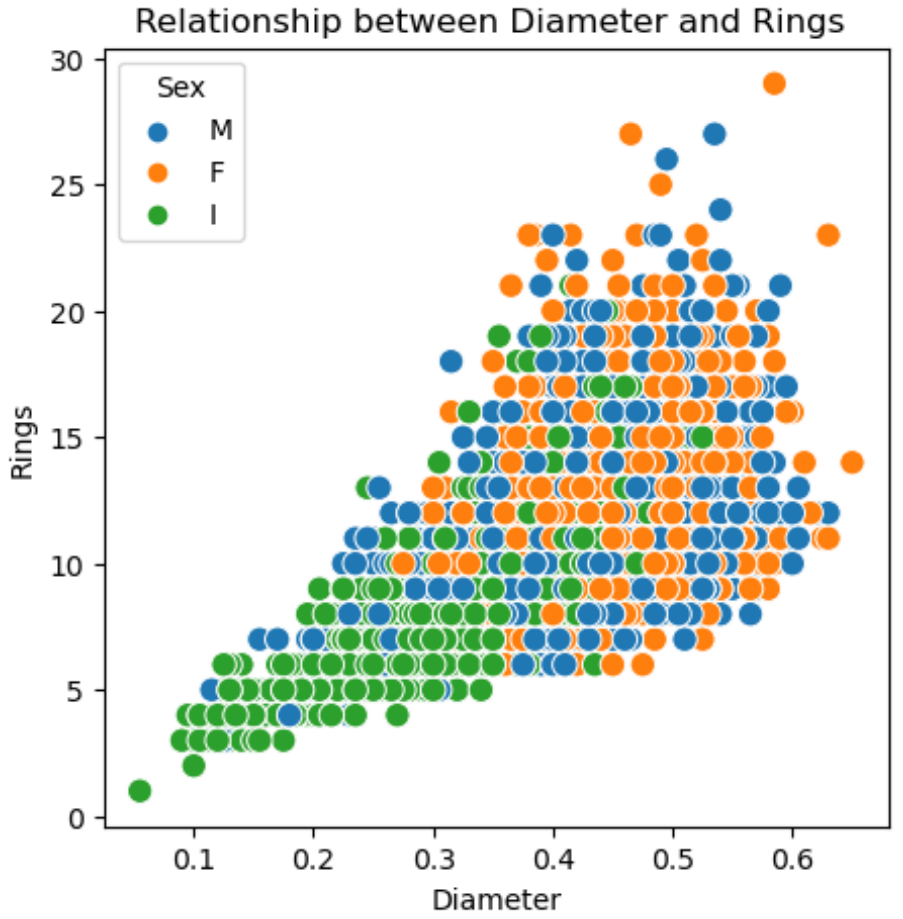
```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Length', y='Rings',
                hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Length and Rings ')
plt.xlabel('Length')
plt.ylabel('Rings')
plt.show()
```



The scatter plot visualizes the relationship between the length of abalones and their age (Rings). It helps to investigate whether longer abalones tend to be older, and how this relationship varies across different sexes.

2.4.2 Diameter vs. Rings

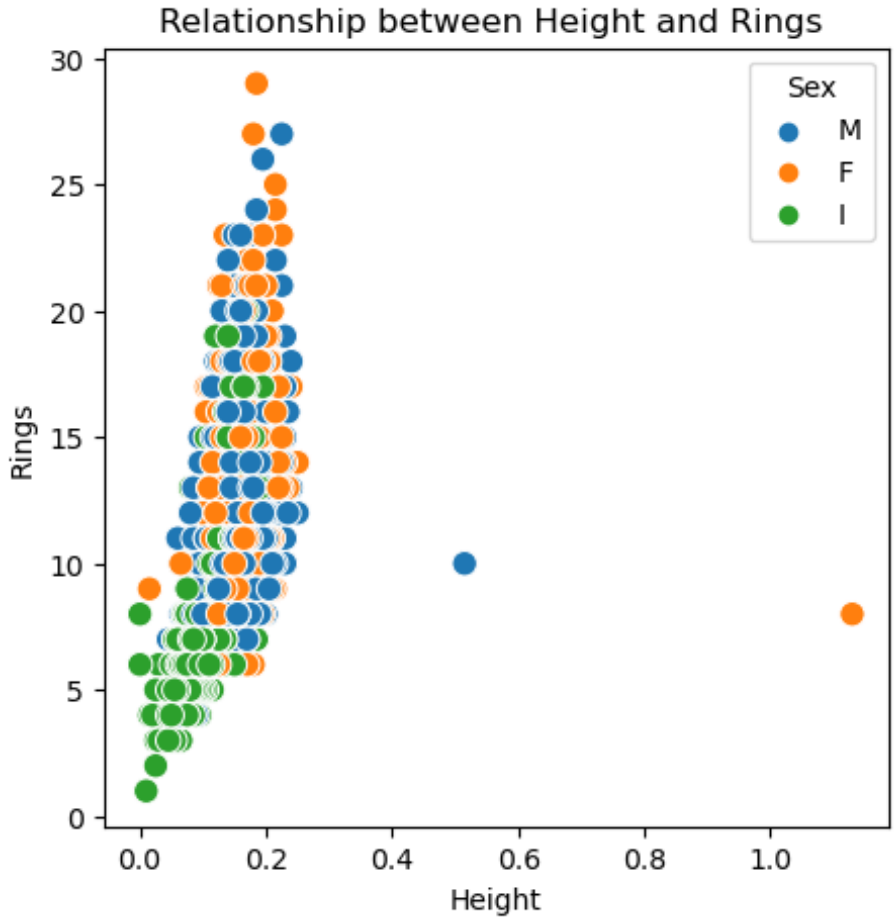
```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Diameter', y='Rings',
                hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Diameter and Rings ')
plt.xlabel('Diameter')
plt.ylabel('Rings')
plt.show()
```



This plot explores how the diameter of abalones is related to their age. Similar to length, it assesses whether a larger diameter is an indicator of an older age and how this pattern differs between male, female, and infant abalones.

2.4.3 Height vs. Rings

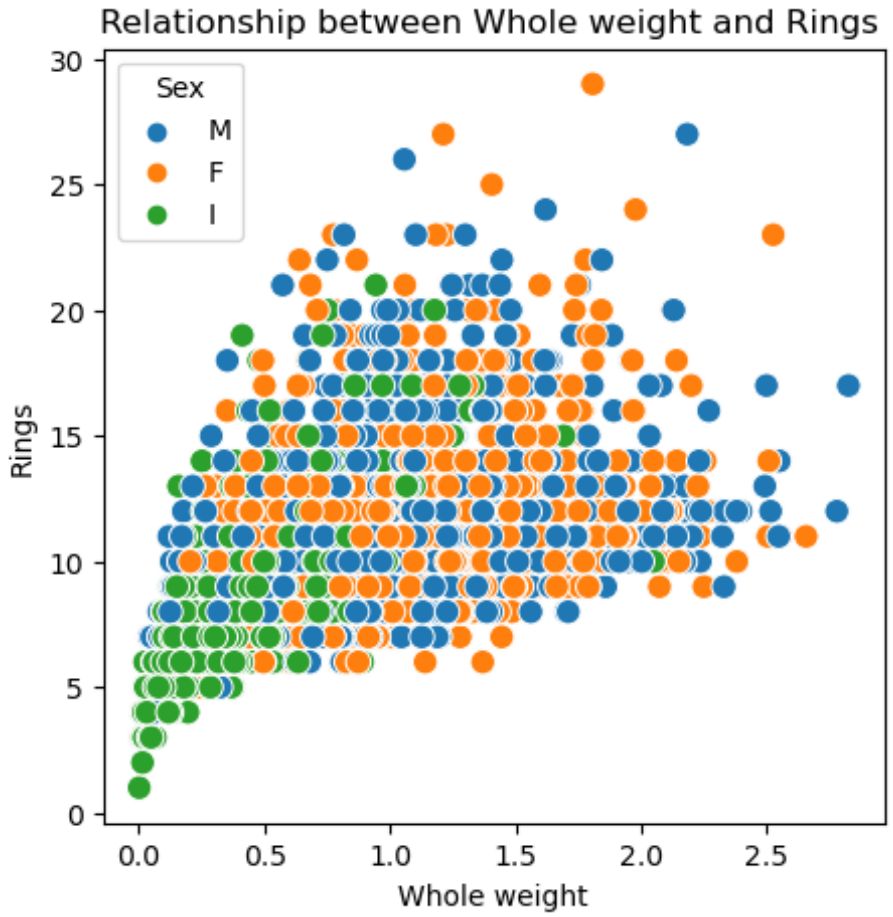
```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Height', y='Rings',
                hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Height and Rings ')
plt.xlabel('Height')
plt.ylabel('Rings')
plt.show()
```



By analyzing height against rings, this plot aims to understand the correlation between the vertical size of abalones and their age. It's useful for seeing if taller abalones are typically older and if this trend is consistent across different sexes.

2.4.4 Whole weight vs. Rings

```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Whole weight', y='Rings',
                hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Whole weight and Rings ')
plt.xlabel('Whole weight')
plt.ylabel('Rings')
plt.show()
```

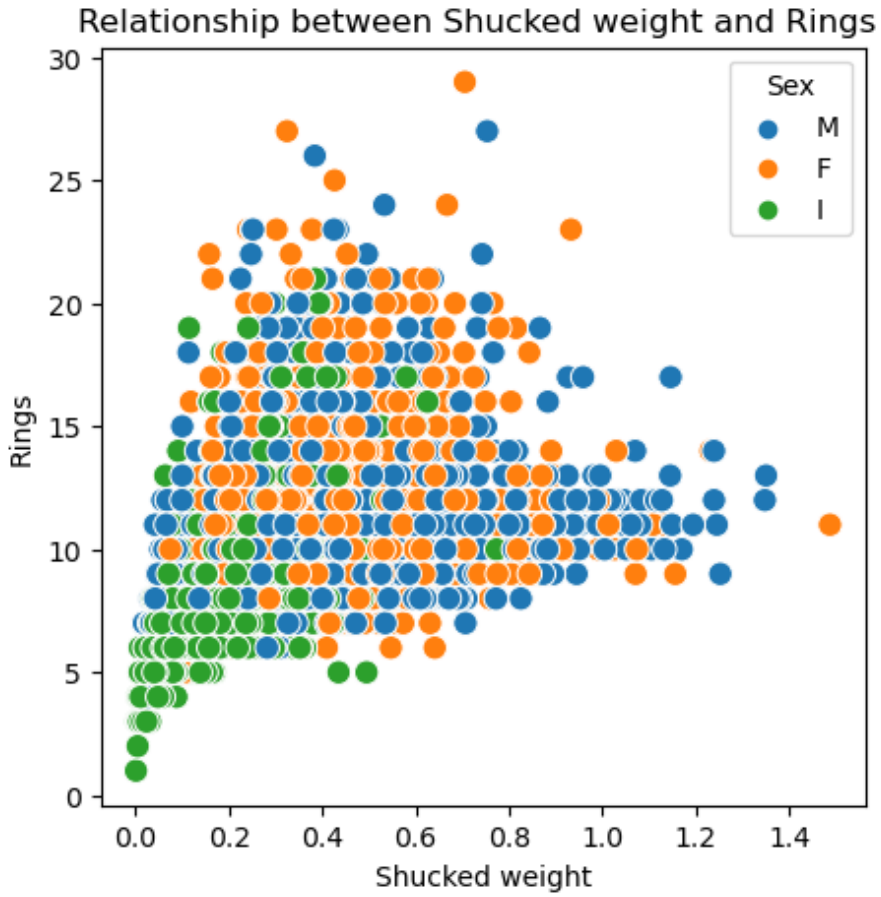


This scatter plot examines the relationship between the overall weight of abalones and the number of rings. It determines if heavier abalones are likely

to be older and how this relationship manifests in male, female, and infant abalones.

2.4.5 Shucked weight vs. Rings

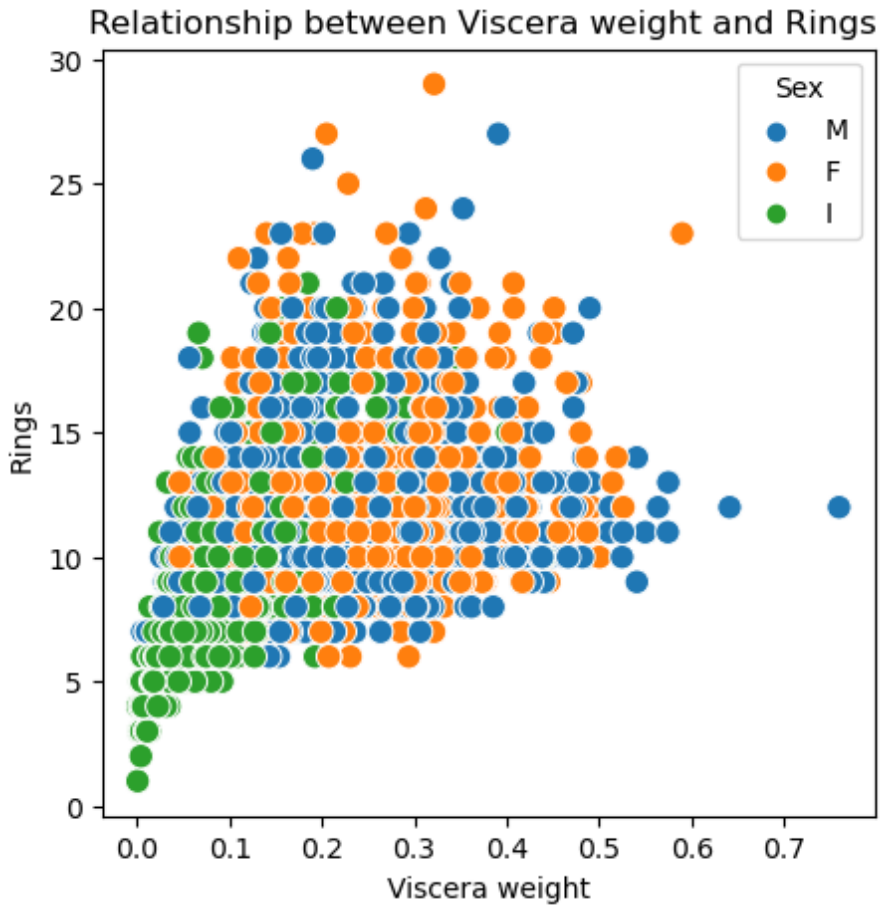
```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Shucked weight',
                y='Rings', hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Shucked weight and Rings')
plt.xlabel('Shucked weight')
plt.ylabel('Rings')
plt.show()
```



The plot compares the weight of abalone meat (after being shucked) with the number of rings. It's particularly interesting for understanding if the weight of the meat correlates with the age, and if this correlation varies based on the sex of the abalone.

2.4.6 Viscera weight vs. Rings

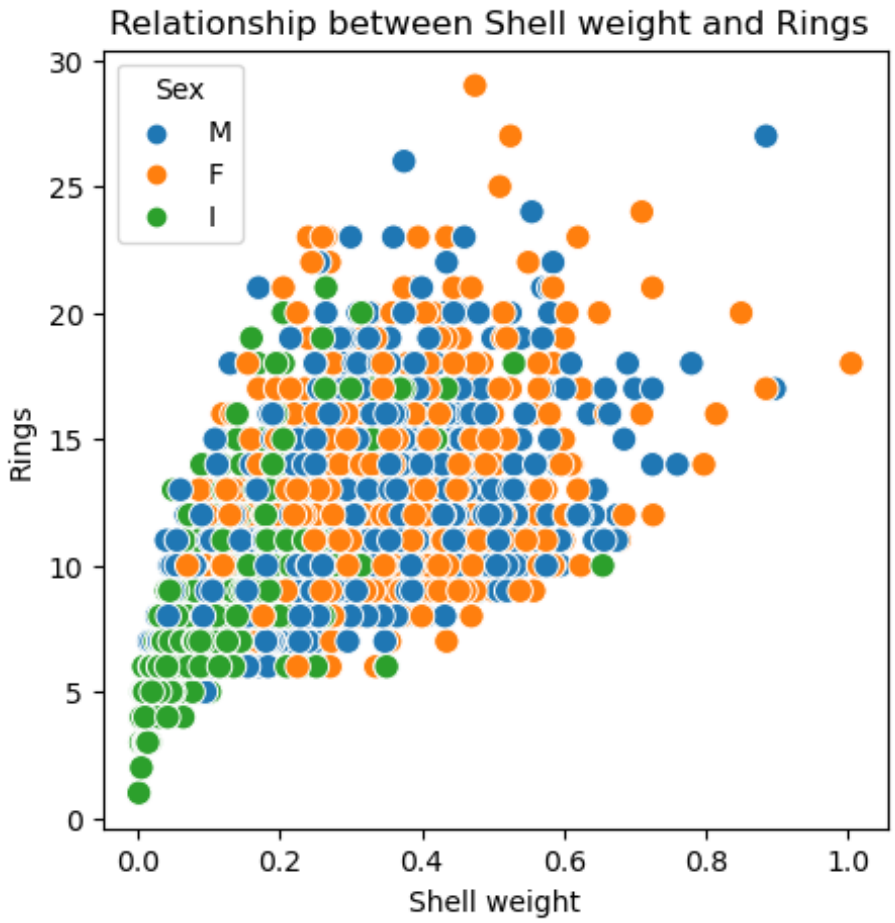
```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Viscera weight',
                y='Rings', hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Viscera weight and Rings')
plt.xlabel('Viscera weight')
plt.ylabel('Rings')
plt.show()
```



By plotting the weight of the abalone's viscera against the number of rings, this plot aims to explore the relationship between the gut weight and the age of the abalone. It helps to determine if abalones with heavier viscera are generally older, and how this trend differs among sexes.

2.4.7 Viscera weight vs. Rings

```
plt.figure(figsize=(5, 5))

sns.scatterplot(data=df, x='Shell weight',
                y='Rings', hue='Sex',s=80, edgecolor='w')
plt.title('Relationship between Shell weight and Rings')
plt.xlabel('Shell weight')
plt.ylabel('Rings')
plt.show()
```



This scatter plot focuses on the relationship between the weight of the abalone's shell and its age. It's valuable for assessing if the shell weight is a good predictor of age and how this relationship is distributed across different sexes.

3 Model Training and Evaluation

3.1 Feature Encoding and Data Splitting

Before training the models, it's crucial to convert categorical data into numerical format, as ML algorithms typically require numerical input. Here, the 'Sex' feature is categorical and represented by 'M', 'F', and 'I'. It's encoded into numerical values where 'M' is 0, 'F' is 1, and 'I' is 2.

```
df['Sex'] =
    df['Sex'].replace({"M":0,"F":1,"I":2}).astype(float)
```

This encoding allows the algorithms to process the 'Sex' feature. After encoding, the dataset is split into features (X) and the target variable (y, the 'Rings' column). The 'train_test_split' function is used to divide the dataset into a training set and a testing set.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

X = df.drop("Rings", axis="columns")
y = df["Rings"]
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3, random_state=40)
```

3.2 Model Training

The Linear Regression, K-Nearest Neighbors, and Random Forest Regressor models are trained to predict the age of abalones based on their physical attributes.

```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

knn_reg = KNeighborsRegressor()
knn_reg.fit(X_train, y_train)

random_forest_reg = RandomForestRegressor()
random_forest_reg.fit(X_train, y_train)
```

3.3 Predicting Test Data

The trained models make predictions on the test set, which are then used for evaluation.

```
linear_pred = linear_reg.predict(X_test)
random_forest_pred = random_forest_reg.predict(X_test)
knn_pred = knn_reg.predict(X_test)
```

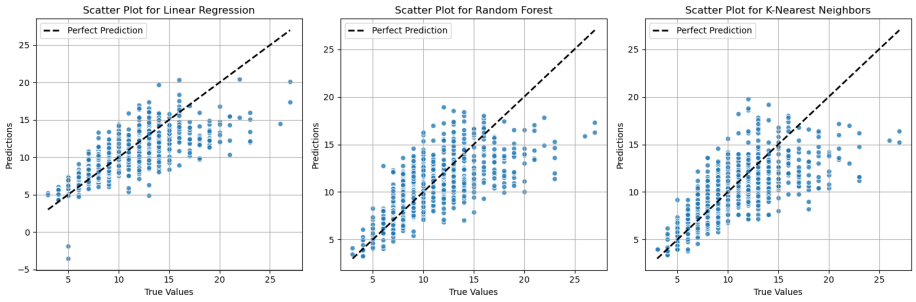
3.4 Visualization of Model Predictions

The scatter, residual, and error distribution plots visually represent the performance of the models, showing the relationship between the true values and the model's predictions.

3.4.1 Scatter Plot

These plots compare the true values of the target variable with the predictions made by the models. A perfect model would result in a straight line where the true values exactly match the predictions.

```
def scatter_plot(y_test, y_pred, model_name):
    plt.scatter(y_test, y_pred, alpha=0.8,edgecolor="w")
    plt.title(f'Scatter Plot for {model_name}')
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.plot([y_test.min(), y_test.max()],
        [y_test.min(), y_test.max()], linestyle='—',
        color='black', label='Perfect Prediction', lw=2)
    plt.legend()

model_predictions = {'Linear Regression': linear_pred, '
                     Random Forest': random_forest_pred,
                     'K-Nearest Neighbors': knn_pred}

plt.figure(figsize=(15, 5))

for index, key in enumerate(model_predictions.keys()):
    plt.subplot(1, 3, index + 1)
    scatter_plot(y_test, model_predictions[key], key)
    plt.grid(True)

plt.tight_layout()
plt.show()
```

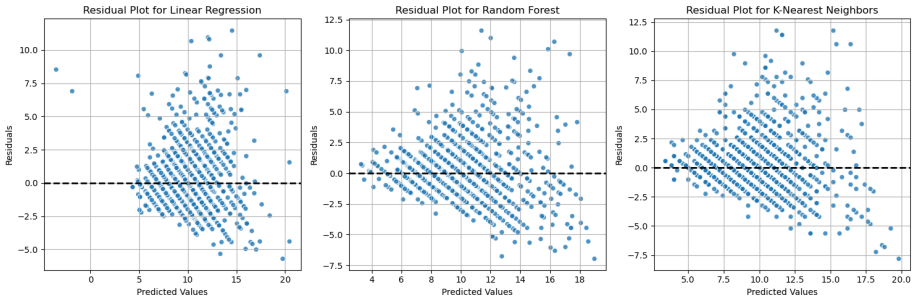


3.4.2 Residual Plot

Residual plots display the difference between the actual values and the predicted values (residuals) against the predicted values. A well-fitted model will

show residuals randomly scattered around zero, indicating that the model's predictions are unbiased.

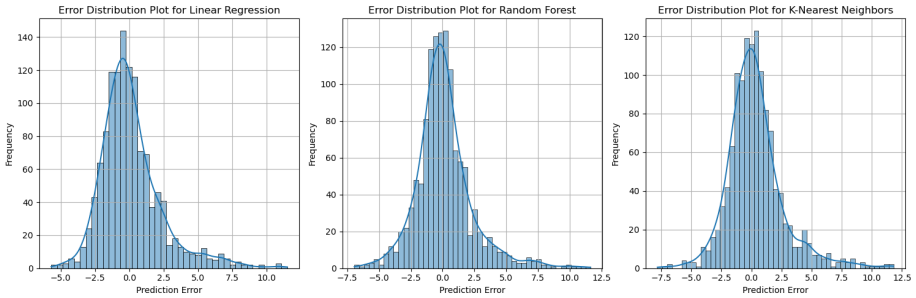
```
import matplotlib.pyplot as plt
import numpy as np

def residual_plot(y_test, y_pred, model_name):
    residuals = y_test - y_pred
    plt.scatter(y_pred, residuals,
                alpha=0.8,edgecolor="w")
    plt.title(f'Residual Plot for {model_name}')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.axhline(y=0, color='black', linestyle='--', lw=2)
    plt.grid(True)

model_predictions = {'Linear Regression': linear_pred,
                     'Random Forest': random_forest_pred,
                     'K-Nearest Neighbors': knn_pred}

plt.figure(figsize=(15, 5))

for index, key in enumerate(model_predictions.keys()):
    plt.subplot(1, 3, index + 1)
    residual_plot(y_test, model_predictions[key], key)

plt.tight_layout()
plt.show()
```



3.4.3 Error Distribution Plot

These plots show the distribution of the prediction errors. Ideally, the errors should be normally distributed, which is a common assumption for many statistical techniques.

```
import matplotlib.pyplot as plt
import numpy as np

def errors_distribution_plot(y_test, y_pred, model_name):
    residuals = y_test - y_pred
    sns.histplot(y_test - y_pred, kde=True)
    plt.title(f'Error Distribution Plot for{model_name}')
    plt.xlabel('Prediction Error')
    plt.ylabel('Frequency')
    plt.grid(True)

model_predictions = {'Linear Regression': linear_pred,
                     'Random Forest': random_forest_pred,
                     'K-Nearest Neighbors': knn_pred}

plt.figure(figsize=(15, 5))

for index, key in enumerate(model_predictions.keys()):
    plt.subplot(1, 3, index + 1)
    errors_distribution_plot(y_test,
                             model_predictions[key], key)

plt.tight_layout()
plt.show()
```



3.5 Model Evaluation

The performance of the models is evaluated using the following metrics, each providing insights into different aspects of the model's predictive power:

1. **Root Mean Squared Error (RMSE):**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

RMSE is the square root of MSE and provides error metrics in the same units as the target variable. It gives a relatively high weight to large errors, guiding models to avoid them.

2. **Mean Absolute Error (MAE):**

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

MAE calculates the average of the absolute errors between the predicted values and the actual values. It's a straightforward measure of prediction accuracy that, unlike MSE or RMSE, isn't overly sensitive to outliers, providing a more robust error metric in the presence of anomalies.

3. **R-squared (R^2):**

$$R = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}$$

R^2 measures the proportion of variance in the dependent variable that can be predicted from the independent variables. It indicates the goodness of fit, with values closer to 1 signifying better model performance.

```
from sklearn.metrics import mean_squared_error,
                            mean_absolute_error, r2_score


def evaluate_model(y_true, y_pred, model_name):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{model_name} - RMSE: {rmse:.2f},
                           MAE: {mae:.2f}, R^2: {r2:.2f}")


for index, key in enumerate(model_predictions.keys()):
    evaluate_model(y_test, model_predictions[key], key)
```

```
Linear Regression - RMSE: 2.15, MAE: 1.55, R^2: 0.54
Random Forest - RMSE: 2.19, MAE: 1.59, R^2: 0.53
K-Nearest Neighbors - RMSE: 2.26, MAE: 1.60, R^2: 0.50
```

In a comparison of model performance:

- Linear Regression outperforms the others, showing the most accurate predictions with the lowest errors (RMSE of 2.15, MAE of 1.55) and explaining about 54% of the data's variance (R^2 of 0.54).
- Random Forest is a close second, with slightly higher errors and a comparable ability to explain the data's variance (R^2 of 0.53).
- K-Nearest Neighbors trails with the least accurate predictions, highest errors, and a moderate explanation of variance (R^2 of 0.50).

4 References

1. Seaborn documentation: `https://seaborn.pydata.org/`
2. Matplotlib documentation: `https://matplotlib.org/`
3. Pandas documentation: `https://pandas.pydata.org/`
4. Scikit-learn documentation: `https://scikit-learn.org/stable/`
5. Abalone Age Prediction Regression: `https://www.kaggle.com/code/anmolbajpai/abalone-age-prediction-regression#Exploratory-Data-Analysis`