```
import pandas as pd   # linear algebra
import numpy as np    # data processing, CSV file I/O like (pd.read_csv)
import seaborn as sns  # for data visuvalization
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

# Avoid Warning
import warnings
warnings.filterwarnings("ignore")
```

1. Data Preprocessing and Missing Value Analysis

```
file_path = 'Fish.csv'
df = pd.read_csv(file_path)
df.head()  # Display first few rows
```

	Weight	Length	Height	Width	Price
0	200.0	50.0	3.0	10.0	150
1	200.0	50.0	3.0	10.0	150
2	200.0	50.0	3.0	10.0	150
3	200.0	50.0	3.0	10.0	150
4	200.0	50.0	3.0	10.0	150

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
print("\n=> summary information of the DataFrame\n")
df.info()
df.shape # will show the dimensions of the data set
```

```
=> summary information of the DataFrame

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Weight  156 non-null    float64
 1   Length  155 non-null    float64
 2   Height  154 non-null    float64
 3   Width   156 non-null    float64
 4   Price   159 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.3 KB
(159, 5)
```

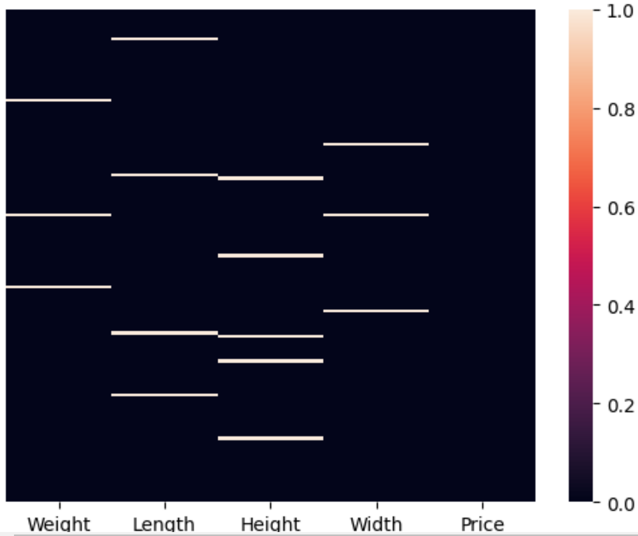
```
# Checking for NaN Values in data frame
print("\nMissing Values in Each Column:")
df.isnull().sum()
```

Missing Values in Each Column:

	0
Weight	3
Length	4
Height	5
Width	3
Price	0

dtype: int64

```
sns.heatmap(df.isnull(), yticklabels=False);
```




```
# Handiling missing values
df.fillna(df.mean(), inplace=True)  # Mean imputation
```

1. Mean Imputation

df.fillna(df.mean(), inplace=True)

Explanation: This method replaces missing values with the mean of the column. It's simple and effective for numerical data, but it can distort the distribution if the data has outliers.

2. Median Imputation

df.fillna(df.median(), inplace=True)

Explanation: Similar to mean imputation, but uses the median. This is more robust to outliers and can be a better choice for skewed distributions.

3. Mode Imputation

df.fillna(df.mode().iloc[0], inplace=True)

Explanation: This method replaces missing values with the mode (most frequent value). It's often used for categorical data.

4. Forward Fill

df.fillna(method='ffill', inplace=True)

Explanation: This method propagates the last observed value forward to fill missing values. It's useful for time series data.

5. Backward Fill

df.fillna(method='bfill', inplace=True)

Explanation: This method uses the next observed value to fill missing values. Like forward fill, it's useful for time series data.

6. Interpolation

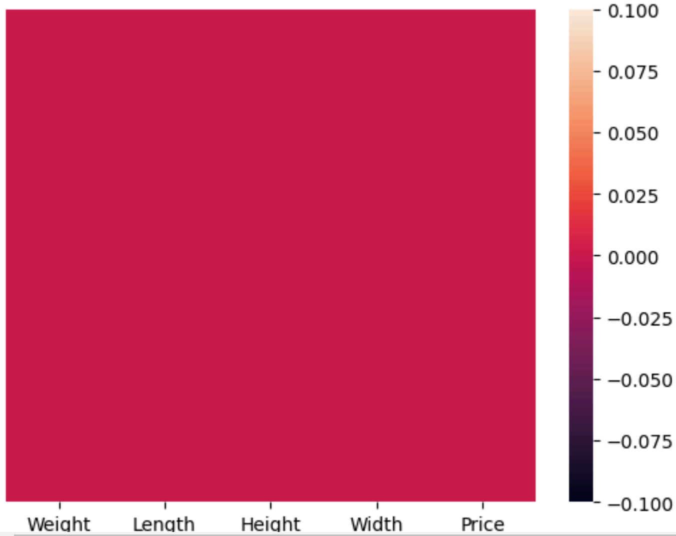
df.interpolate(method='linear', inplace=True)

Explanation: This method estimates missing values by interpolating between the known values. It's useful for numerical data and can be applied in various ways (linear, polynomial, etc.)

```
sns.heatmap(df.isnull(), yticklabels=False);
df.isnull().sum()  # after cleaning
```

	0
Weight	0
Length	0
Height	0
Width	0
Price	0

dtype: int64




```
# Normalize the dataset
scaler = MinMaxScaler()
data_normalized = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

print("Normalized Data:\n", data_normalized.head())
```

```
Normalized Data:
    Weight  Length  Height  Width  Price
0     0.0     0.0     0.0    1.0    0.0
1     0.0     0.0     0.0    1.0    0.0
2     0.0     0.0     0.0    1.0    0.0
3     0.0     0.0     0.0    1.0    0.0
4     0.0     0.0     0.0    1.0    0.0
```

Normalization scales features to a uniform range, usually [0, 1]. This is crucial when features have different scales or units. For example, Weight might be in kilograms, while Price is in dollars. Normalization ensures that no feature dominates due to its scale.

Using MinMaxScaler to scale features to the range [0, 1]. This step is essential for ensuring all features contribute equally and that machine learning algorithms perform optimally.

we can ensure that your dataset is properly preprocessed, making it ready for accurate and effective analysis or modeling.

2. Exploratory Data Analysis (EDA)

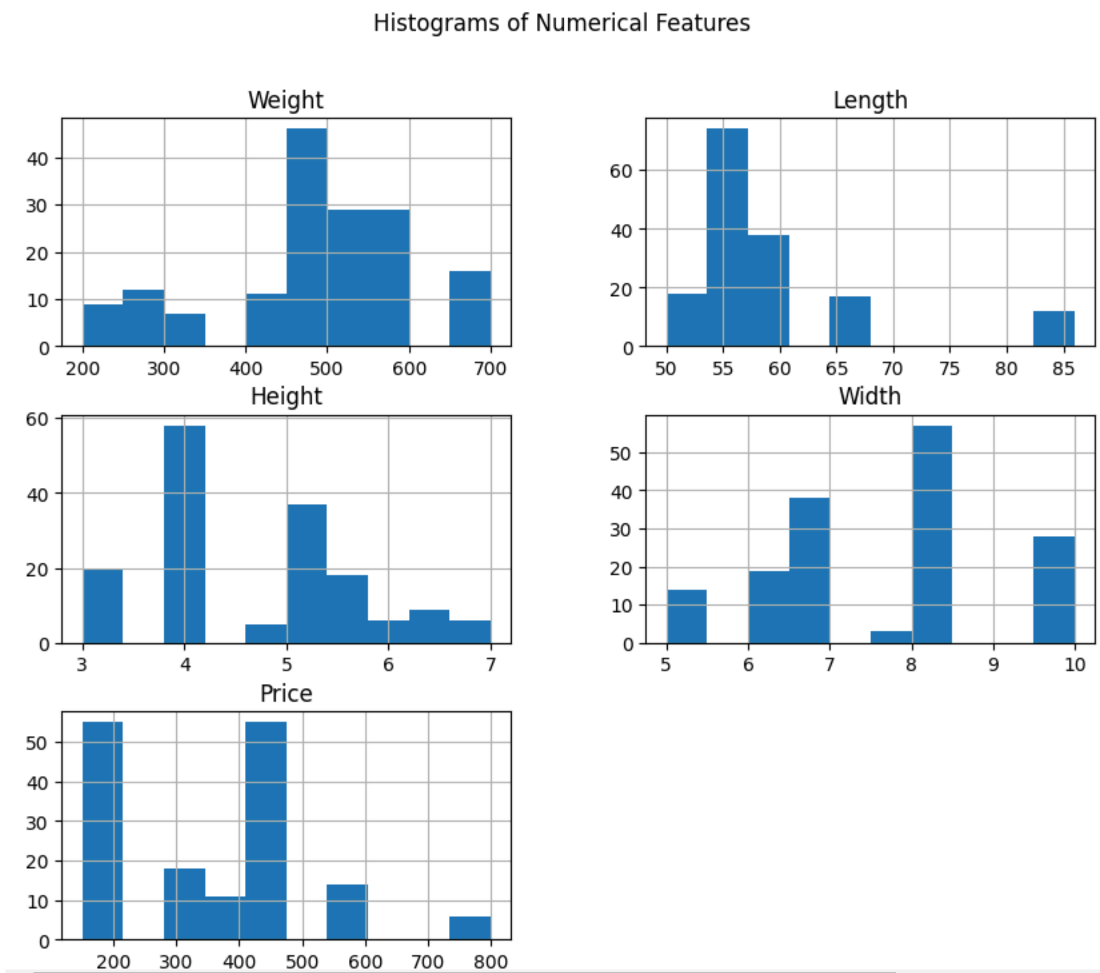
```
# Summary statistics
summary = df.describe()
print("Summary Statistics:\n", summary)

# Count of each unique value in 'Price' column (assuming it's our species equivalent)
species_count = df['Price'].value_counts()
print("Count of Each Price:\n", species_count)
```

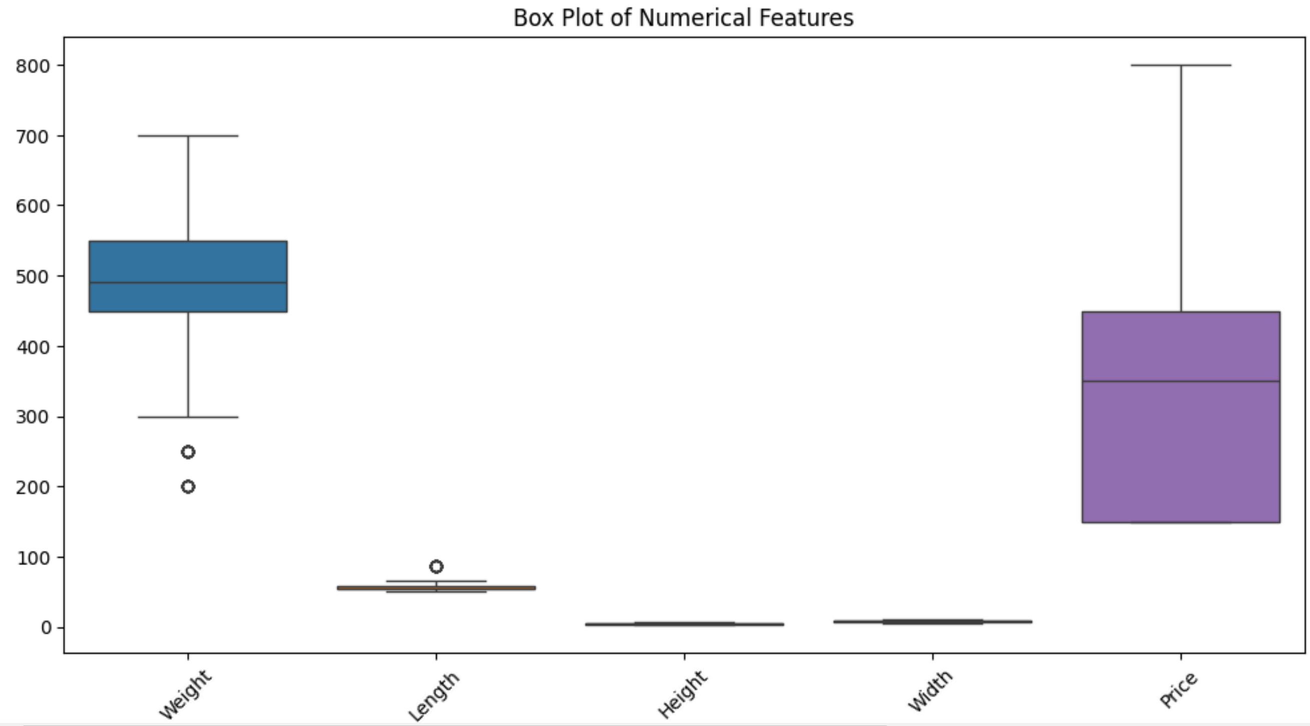
```
Summary Statistics:
             Weight       Length       Height        Width        Price
```

```
count  159.000000  159.000000  159.000000  159.000000  159.00000
mean   470.512821   58.838710    4.688961    7.651282  342.45283
std    127.392520    8.716697    1.072178    1.522177  170.61585
min    200.000000   50.000000    3.000000    5.000000  150.00000
25%    450.000000   55.000000    4.000000    6.700000  150.00000
50%    490.000000   55.000000    4.688961    8.200000  350.00000
75%    550.000000   59.000000    5.300000    8.400000  450.00000
max    700.000000   86.000000    7.000000   10.000000  800.00000
Count of Each Price:
 Price
150    55
450    55
280    17
550    14
350    11
800     6
340     1
Name: count, dtype: int64
```

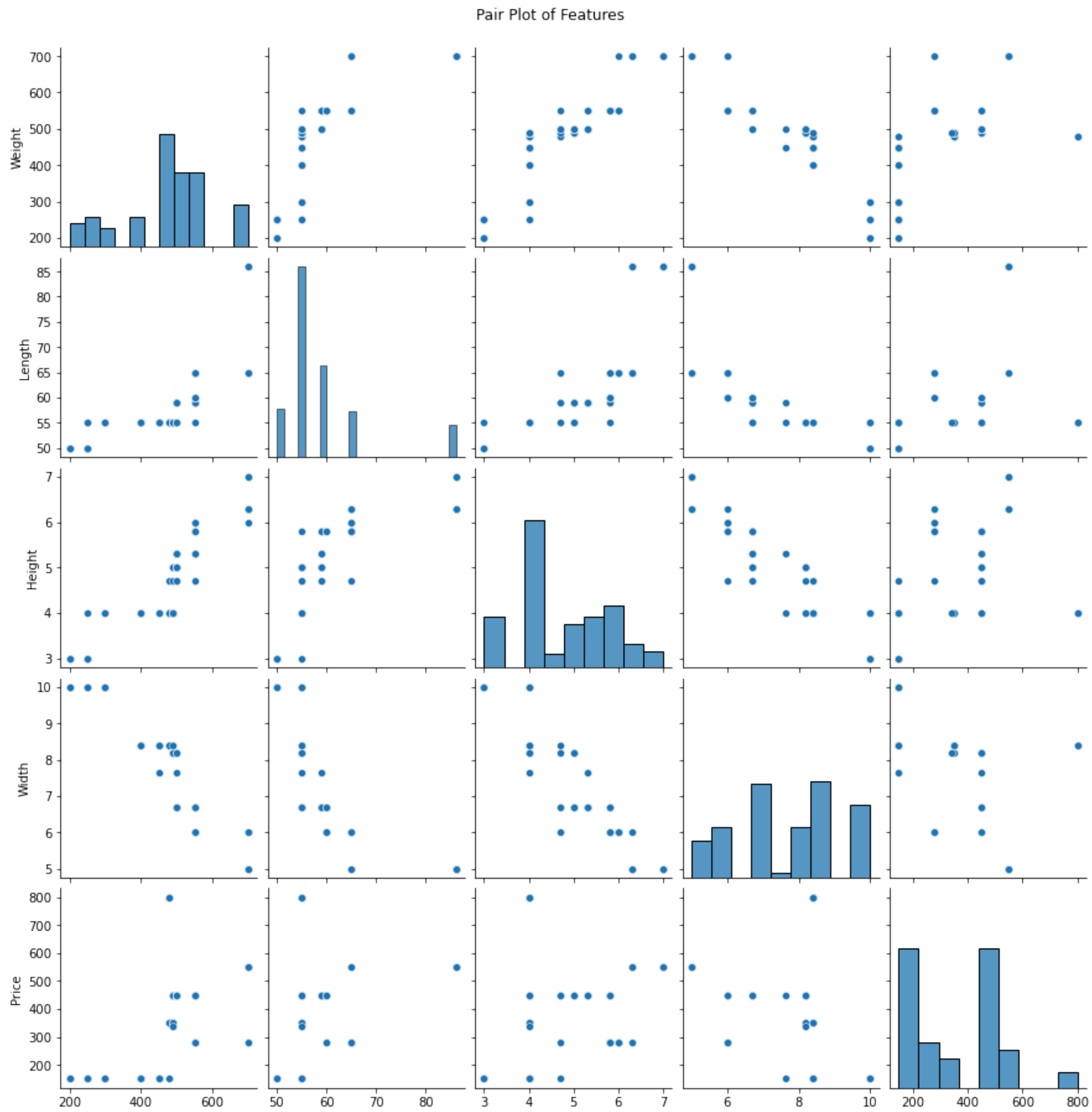
```
# Histograms for numerical features
df.hist(figsize=(10, 8))
plt.suptitle('Histograms of Numerical Features')
plt.show()
```




```
# Box plot to check for outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.title('Box Plot of Numerical Features')
plt.xticks(rotation=45)
plt.show()
```




```
# Scatter plot matrix to explore relationships
sns.pairplot(df)
plt.suptitle('Pair Plot of Features', y=1.02)
plt.show()
```



Patterns or trends observed from the visualizations

Skewness: Both the histograms and box plots reveal right-skewed distributions for Weight and Price, indicating more lower values with fewer higher values.

Outliers: The box plot highlights outliers in the Weight feature, which could be important for further analysis.

Correlations: The pair plot shows positive correlations between Age and Length, and distinct clusters in the Rings feature, suggesting subgroups.

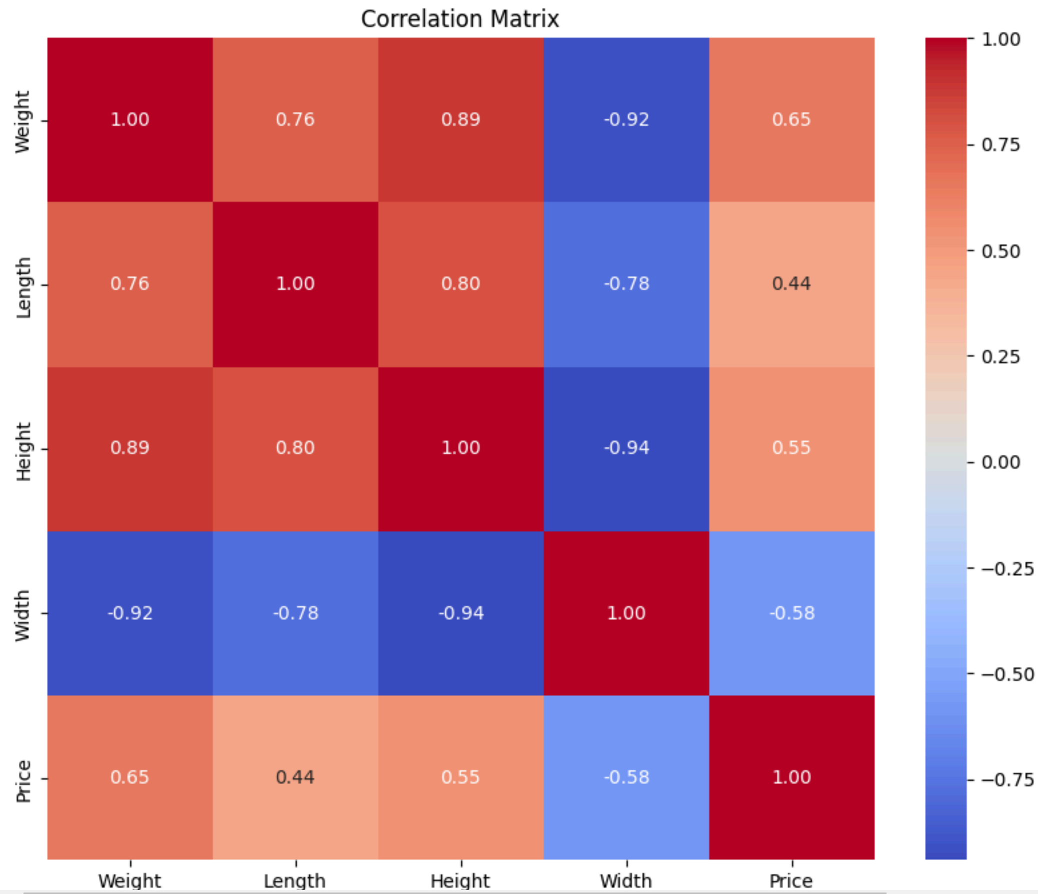
3. Feature Selection and Engineering

```
# Calculate the correlation matrix
correlation_matrix = df.corr()

print("Correlation Matrix:\n", correlation_matrix)

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

```
Correlation Matrix:
            Weight    Length    Height     Width     Price
Weight  1.000000  0.763484  0.890118 -0.922655  0.647659
Length  0.763484  1.000000  0.801108 -0.782303  0.435523
Height  0.890118  0.801108  1.000000 -0.940418  0.550545
Width  -0.922655 -0.782303 -0.940418  1.000000 -0.576734
Price   0.647659  0.435523  0.550545 -0.576734  1.000000
```



4. Linear Regression Model

```
from sklearn.model_selection import train_test_split

# Define features and target variable
X = df.drop('Price', axis=1)
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"\nTraining set size: {X_train.shape}")
print(f"Testing set size: {X_test.shape}")
```

```

Training set size: (127, 4)
Testing set size: (32, 4)
```

Initialize and Train the Model

```
from sklearn.linear_model import LinearRegression

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Evaluate the Model

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Train the model on the training set
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("\nEvaluation Metrics:")
print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")

# Display the coefficients
print("\nModel Coefficients:")
print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
```

```
Evaluation Metrics:
Mean Absolute Error: 110.06999732821613
Mean Squared Error: 27247.738487100283
Root Mean Squared Error: 165.06889012500292
R-squared: 0.34448924438055173

Model Coefficients:
Intercept: 25.80208606854484
Coefficients: [ 0.81671667 -3.3692484  26.21581243 -0.21287798]
```

Make Predections

```
# Predicting new data
# Create new fish data
new_fish_data = pd.DataFrame({
    'Length': [30],
    'Weight': [500],
    'Species_X': [1],  # Assuming species X was one-hot encoded
    'Species_Y': [0]   # Assuming species Y was one-hot encoded
})

# Ensure the new data matches the training data columns
# Add missing columns with default values if necessary
for column in X.columns:
    if column not in new_fish_data.columns:
        new_fish_data[column] = 0

# Reorder columns to match training data
new_fish_data = new_fish_data[X.columns]

# Predict the price of the new fish
predicted_price = model.predict(new_fish_data)
print("\nPredicted Price for New Fish:")
print(predicted_price)
```

```
Predicted Price for New Fish:
[333.08296788]
```

5. Discussion and Conclusion

Simple Python Function for Model Evaluation

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
import numpy as np

def evaluate_model(X, y):
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Initialize the model
    model = LinearRegression()

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    # Print evaluation metrics
    print("\nEvaluation Metrics:")
    print(f"Mean Absolute Error: {mae}")
    print(f"Mean Squared Error: {mse}")
    print(f"Root Mean Squared Error: {rmse}")
    print(f"R-squared: {r2}")

    # Print model coefficients
    print("\nModel Coefficients:")
    print(f"Intercept: {model.intercept_}")
```

```
print(f"Coefficients: {model.coef_}")
```

Discussion on Model Performance

Performance of the Linear Regression Model

- **Mean Absolute Error (MAE)**: Indicates the average magnitude of errors in predictions, without considering their direction.
- **Mean Squared Error (MSE)**: Measures the average of the squares of the errors, penalizing larger errors more.
- **Root Mean Squared Error (RMSE)**: The square root of MSE, providing an error metric in the same units as the target variable.
- **R-squared (R^2)**: Represents the proportion of the variance in the dependent variable that is predictable from the independent variables. Higher values indicate better model performance.

How Well Does the Model Predict Fish Price?

- The performance metrics (MAE, MSE, RMSE, R^2) will give us a quantitative measure of how well the model predicts fish prices. Generally, lower MAE, MSE, and RMSE values indicate better performance, while a higher R^2 value (closer to 1) indicates a better fit.

Limitations of the Model

- **Assumption of Linearity**: Linear regression assumes a linear relationship between the features and the target variable, which may not always be the case.
- **Outliers**: The model can be sensitive to outliers, which can skew the results.
- **Multicollinearity**: High correlation between independent variables can affect the stability of the model coefficients.
- **Overfitting/Underfitting**: The model might overfit or underfit the data, depending on the complexity of the relationship between features and the target variable.

Possible Improvements

- **Feature Engineering**: Creating new features or transforming existing ones to better capture the underlying patterns.
- **Regularization**: Techniques like Ridge or Lasso regression can help in dealing with multicollinearity and overfitting.
- **Outlier Treatment**: Identifying and treating outliers can improve model performance.
- **Non-linear Models**: Exploring non-linear models like decision trees, random forests, or gradient boosting machines if the relationship between features and target is non-linear.

Reflection on Workflow and Challenges

- **Data Preprocessing**: Handling missing values, scaling features, and encoding categorical variables are crucial steps.
- **Model Selection**: Choosing the right model and tuning its hyperparameters significantly impact in the performance.
- **Evaluation**: Using appropriate metrics to evaluate model performance and ensuring it generalizes well to unseen data.
- **Challenges**: Common challenges include dealing with imbalanced data, high dimensionality, and ensuring reproducibility of results.