Abalone Age Prediction - Kaggle Playground Series (Season 4, Episode 4) 🐚🔮

Introduction 🌊

This competition invites you to explore the fascinating world of abalones and test your machine learning prowess. By predicting the age of abalones accurately, you'll unlock insights into these marine mollusks and their growth patterns.

Objective 🎯

Your mission is to develop robust machine learning models capable of accurately predicting the age of abalones based on their physical attributes.

Evaluation 📊

Your submissions will be evaluated using the **Root Mean Squared Logarithmic Error (RMSLE)** metric. This metric ensures your predictions are as close as possible to the true values while penalizing large errors.

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(x_i+1) - \log(y_i+1))^2}$$

Dataset 📦

The dataset for this competition includes various physical measurements of abalones, such as length, diameter, height, and weight. These features serve as valuable indicators of abalone age.

Files 📄

- **train.csv**: Training dataset containing abalone measurements and their corresponding ages.
- **test.csv**: Test dataset for which you need to predict the abalone ages.
- **sample_submission.csv**: A sample submission file demonstrating the correct format for predictions.

Installation & Usage 💻

Follow these simple steps to get started:

1. **Clone the Repository**:

   ```
   git clone https://github.com/yashksaini-coder/Abalone-age-prediction
   ```
2. **Navigate to the Repository Directory**:

   ```
   cd Abalone-age-prediction
   ```
3. **Install Dependencies**:

   ```
   pip install -r requirements.txt
   ```

Explore the dataset, develop your machine learning models, and submit your predictions through the Kaggle competition platform before the deadline.

Dependencies 🛠️

- Python 3.9
- Data manipulation libraries (e.g., Pandas, NumPy)
- Machine learning libraries (e.g., Scikit-learn, TensorFlow, PyTorch)

🐚✨ Author:-

Linkedin

GitHub

In [ ]:
```
# Importing the basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import mean_squared_log_error

import skimpy
import warnings
# Disable all warnings
warnings.filterwarnings ('ignore')
```

In [ ]:
```
train = pd.read_csv('playground-series-s4e4/train.csv')
```

In [ ]:
```
train.head()
```

Out[ ]:

	id	Sex	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight	Rings
0	0	F	0.550	0.430	0.150	0.7715	0.3285	0.1465	0.2400	11
1	1	F	0.630	0.490	0.145	1.1300	0.4580	0.2765	0.3200	11
2	2	I	0.160	0.110	0.025	0.0210	0.0055	0.0030	0.0050	6
3	3	M	0.595	0.475	0.150	0.9145	0.3755	0.2055	0.2500	10
4	4	I	0.555	0.425	0.130	0.7820	0.3695	0.1600	0.1975	9

In [ ]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90615 entries, 0 to 90614
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              90615 non-null  int64  
 1   Sex             90615 non-null  object 
 2   Length          90615 non-null  float64
 3   Diameter        90615 non-null  float64
 4   Height          90615 non-null  float64
 5   Whole weight    90615 non-null  float64
 6   Whole weight.1  90615 non-null  float64
 7   Whole weight.2  90615 non-null  float64
 8   Shell weight    90615 non-null  float64
 9   Rings           90615 non-null  int64  
dtypes: float64(7), int64(2), object(1)
memory usage: 6.9+ MB
```

In [ ]:
```
train.isnull().sum()
```

Out[ ]:
```
id                0
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Whole weight.1    0
Whole weight.2    0
Shell weight      0
Rings             0
dtype: int64
```

EDA (Explorartory Data Analysis)

In [ ]:
```
from skimpy import skim
skim(train)
```

skimpy summary

Data Summary

dataframe	Values
Number of rows	90615
Number of columns	10

Data Types

Column Type	Count
float64	7
int32	2
string	1

number

column_name	NA	NA %	mean	sd	p0	p25
id	0	0	45000	26000	0	23000
Length	0	0	0.52	0.12	0.075	0.45
Diameter	0	0	0.4	0.098	0.055	0.34
Height	0	0	0.14	0.038	0	0.11
Whole weight	0	0	0.79	0.46	0.002	0.42
Whole weight.1	0	0	0.34	0.2	0.001	0.18
Whole weight.2	0	0	0.17	0.1	0.0005	0.086
Shell weight	0	0	0.23	0.13	0.0015	0.12
Rings	0	0	9.7	3.2	1	8

string

column_name	NA	NA %	words per row
Sex	0	0	

End

In [ ]:
```
df = train.copy(deep='True')
df = df.drop(columns='id',axis=1)
```

In [ ]:
```
df.head()
```

Out[ ]:

	Sex	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight	Rings
0	F	0.550	0.430	0.150	0.7715	0.3285	0.1465	0.2400	11
1	F	0.630	0.490	0.145	1.1300	0.4580	0.2765	0.3200	11
2	I	0.160	0.110	0.025	0.0210	0.0055	0.0030	0.0050	6
3	M	0.595	0.475	0.150	0.9145	0.3755	0.2055	0.2500	10
4	I	0.555	0.425	0.130	0.7820	0.3695	0.1600	0.1975	9

In [ ]:
```
numeric_features = df.select_dtypes(include = ['int', 'float']).columns.to_list(
categoric_features = df.select_dtypes(include = ['object', 'category']).columns.
```

In [ ]:
```
numeric_features
```

Out[ ]:
```
['Length',
 'Diameter',
 'Height',
 'Whole weight',
 'Whole weight.1',
 'Whole weight.2',
 'Shell weight',
 'Rings']
```

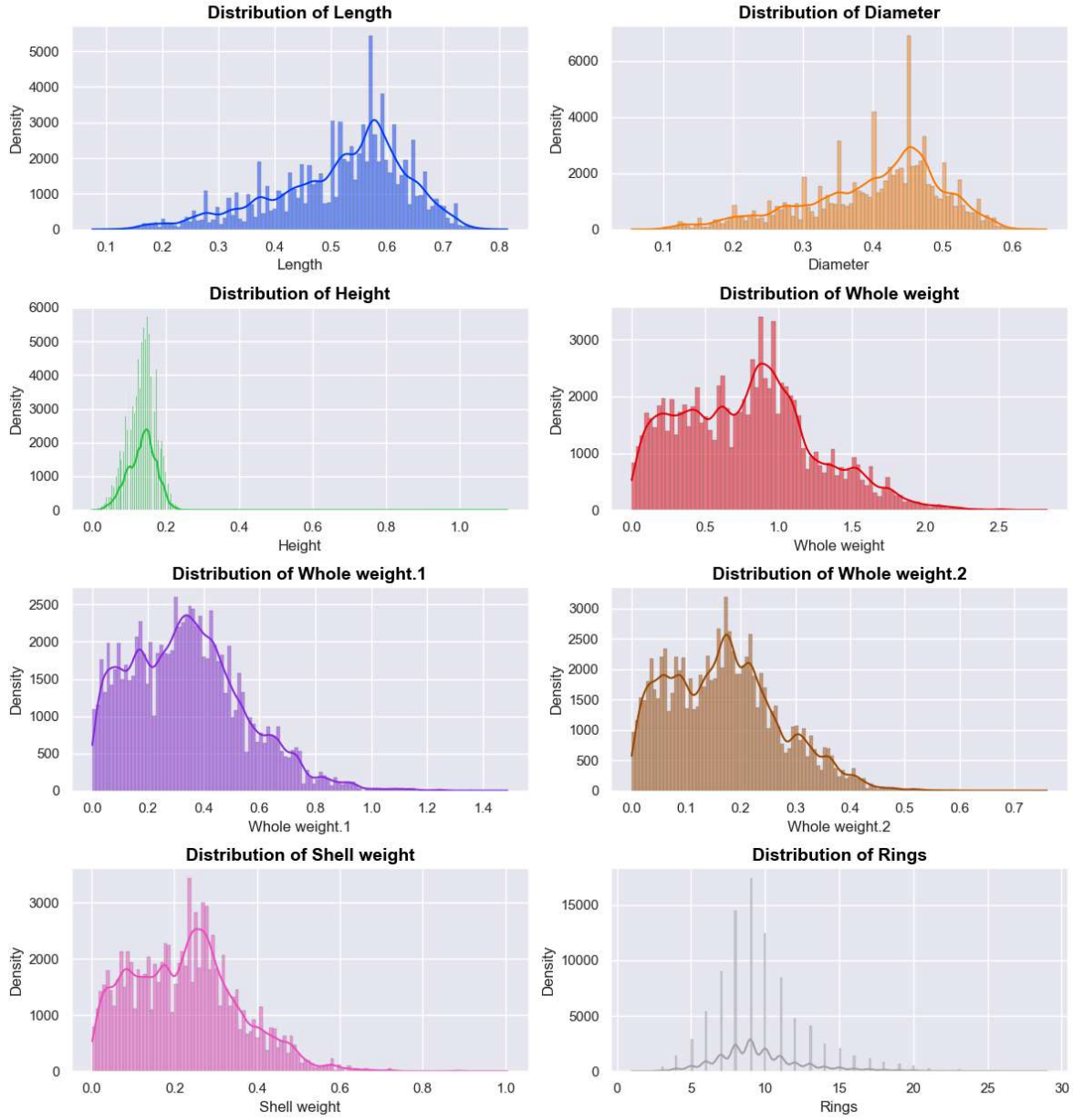
In [ ]:
```
categoric_features
```

Out[ ]:
```
['Sex']
```

Hist Plot

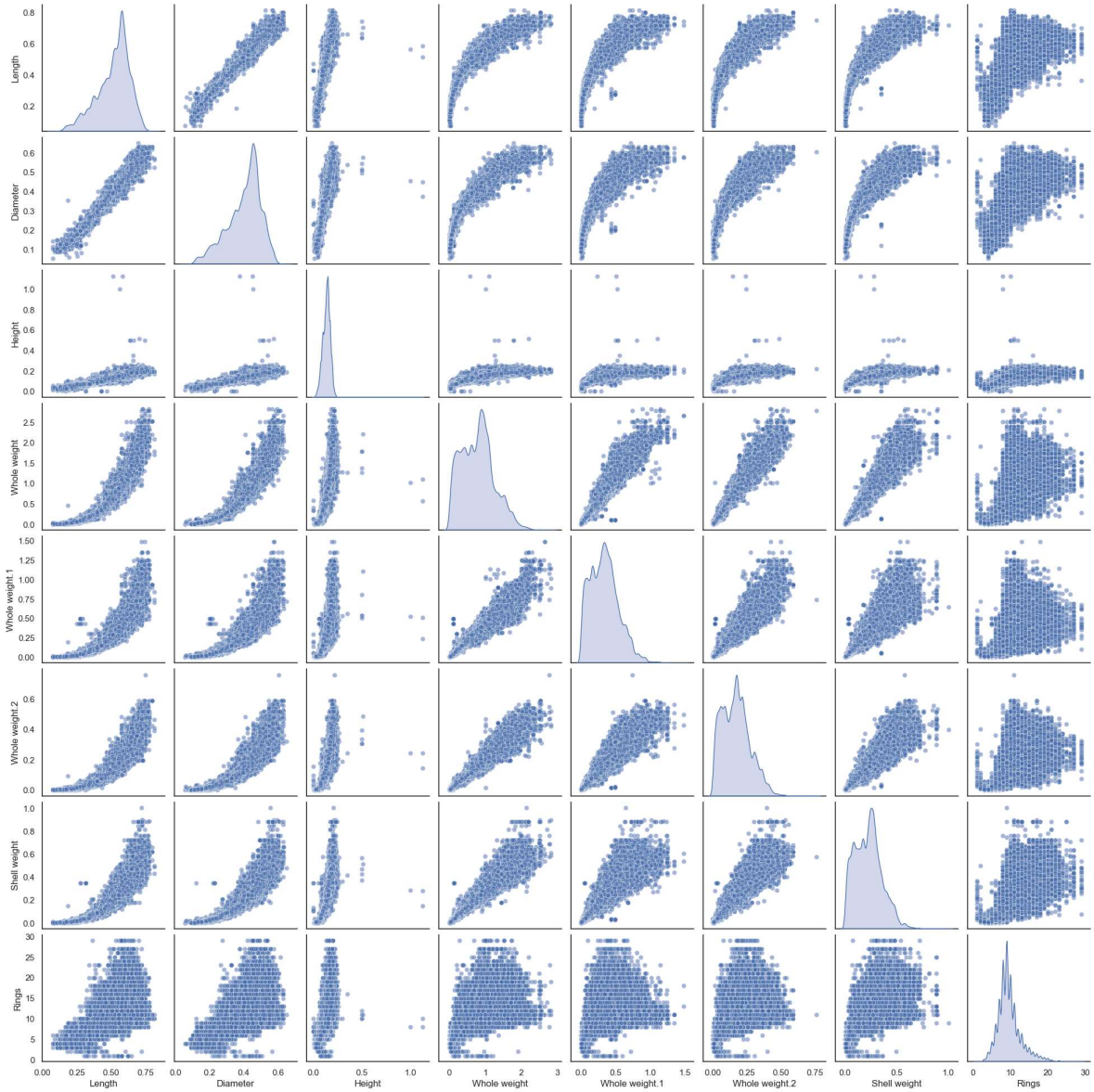
In [ ]:
```
sns.set_style(style="darkgrid")
colors = sns.color_palette(palette='bright', n_colors=len(numeric_features))
fig, axs = plt.subplots(nrows=4, ncols=2, figsize=(12, 14))
axs = axs.flat

# Loop through each numeric feature and plot its distribution
for i, num_feat in enumerate(numeric_features):
    sns.histplot(df[num_feat], kde=True, color=colors[i], ax=axs[i], edgecolor="
    axs[i].set_xlabel(num_feat, fontsize=12)
    axs[i].set_ylabel("Density", fontsize=12)
    axs[i].set_title(f"Distribution of {num_feat}", fontsize=14, fontweight='bol


for j in range(len(numeric_features), len(axs)):
    fig.delaxes(axs[j])

fig.suptitle("Distribution of numerical variables", fontsize=16, fontweight="bol
fig.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust the layout to accommodate th
plt.show()
```

Distribution of numerical variables



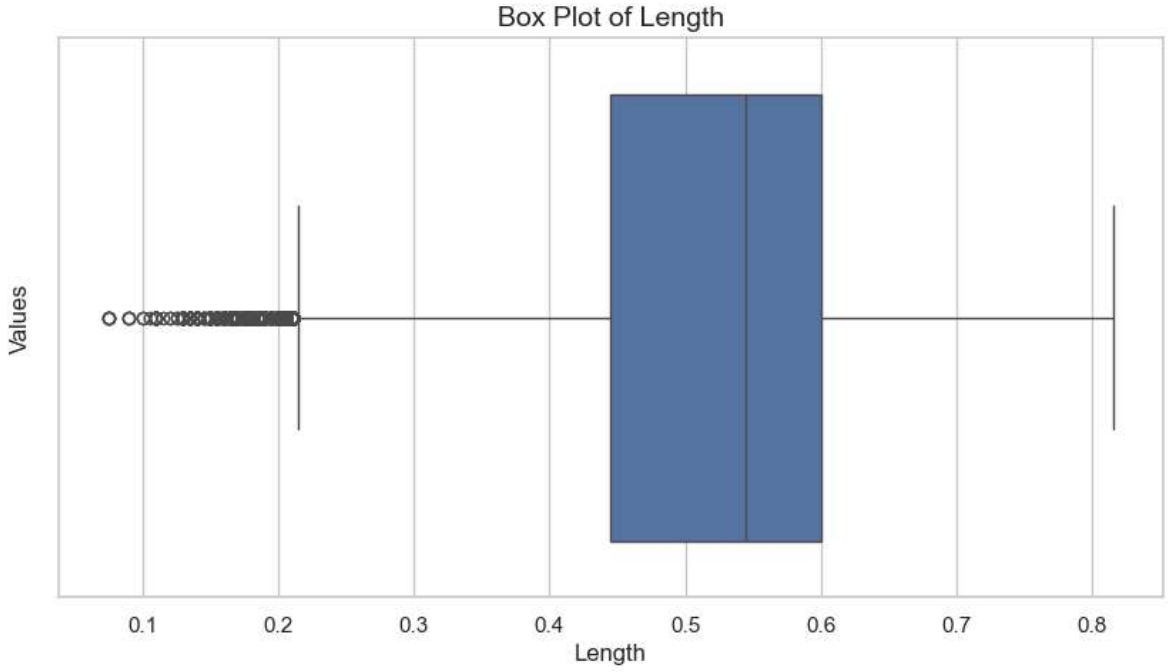
Pair Plot

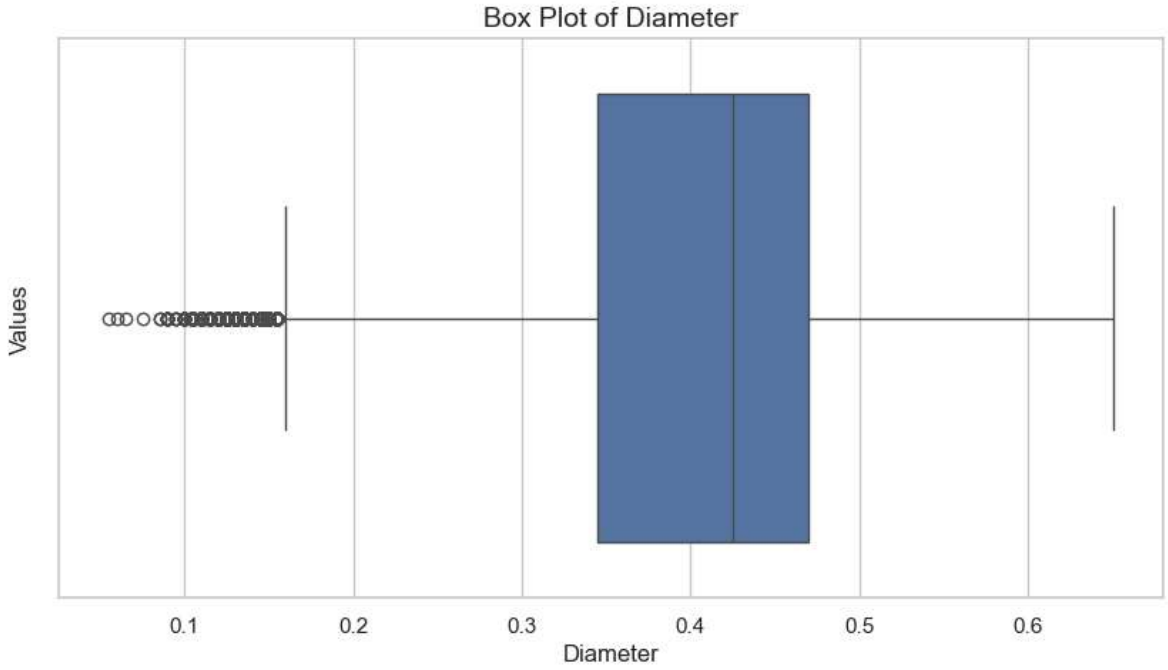
In [ ]:
```
sns.set(style="white")
sns.pairplot(df, diag_kind="kde", markers="o", plot_kws={"alpha": 0.5})
plt.show()
```

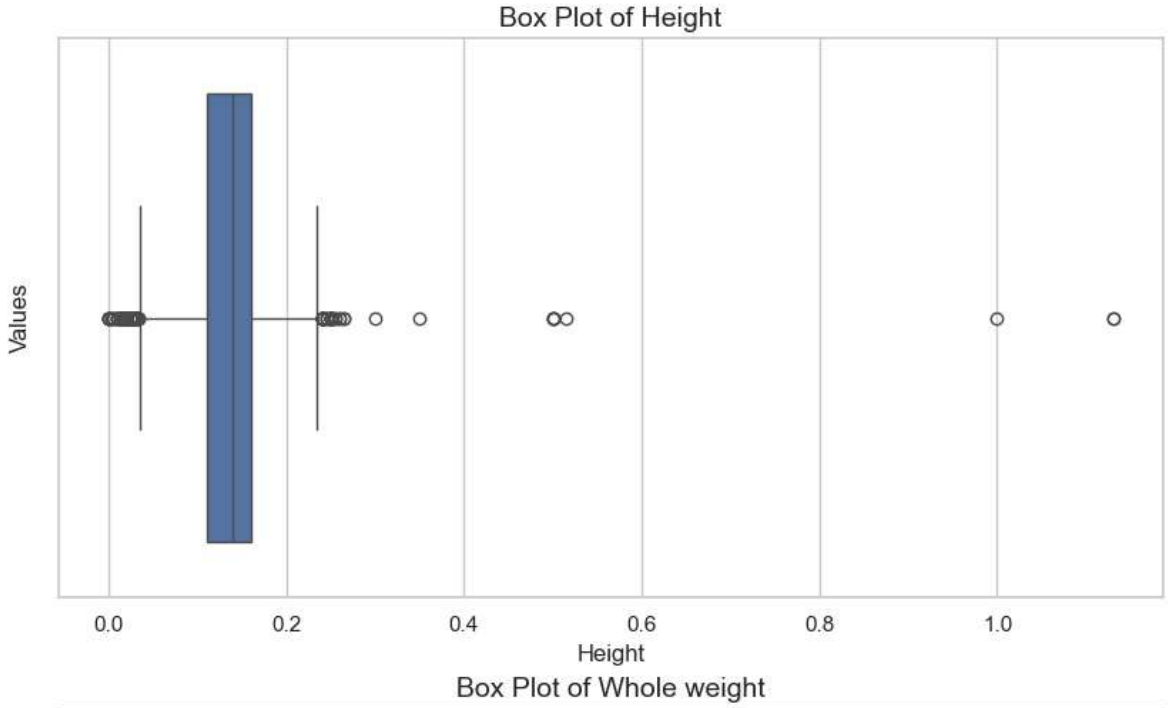


Box Plot

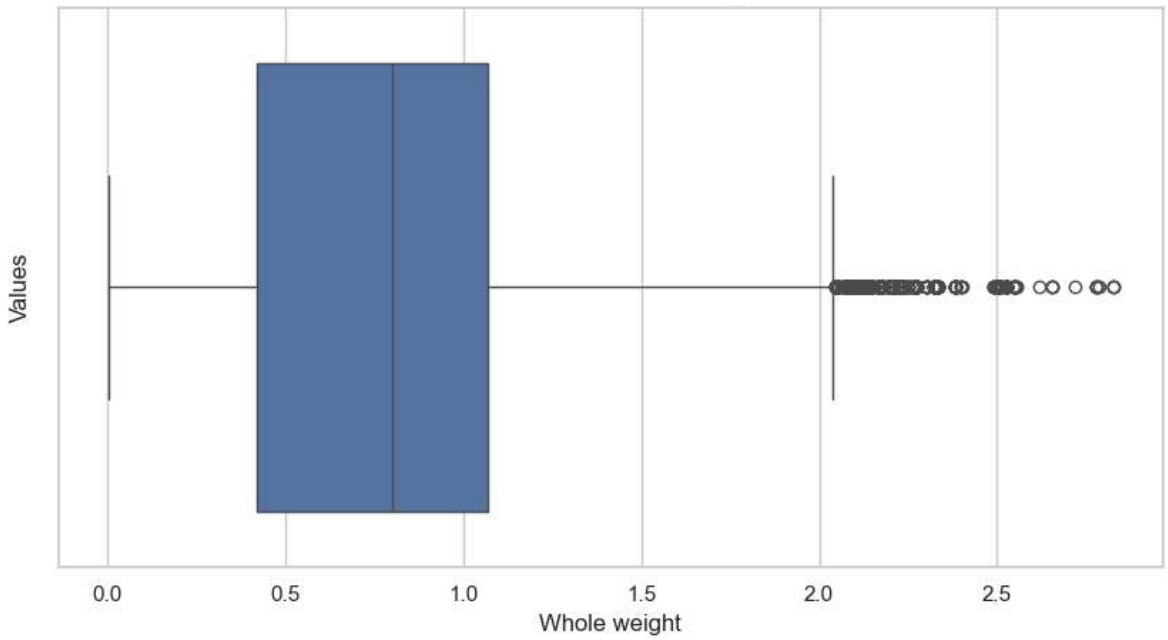
In [ ]:
```
sns.set(style="whitegrid")
fig, axs = plt.subplots(nrows=len(numeric_features), figsize=(10, 6 * len(numeri

# Loop through each numerical feature and create a box plot
for i, num_feat in enumerate(numeric_features):
    sns.boxplot(x=num_feat, data=df, ax=axs[i])
    axs[i].set_title(f"Box Plot of {num_feat}", fontsize=14)
    axs[i].set_xlabel(num_feat, fontsize=12)
    axs[i].set_ylabel("Values", fontsize=12)

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



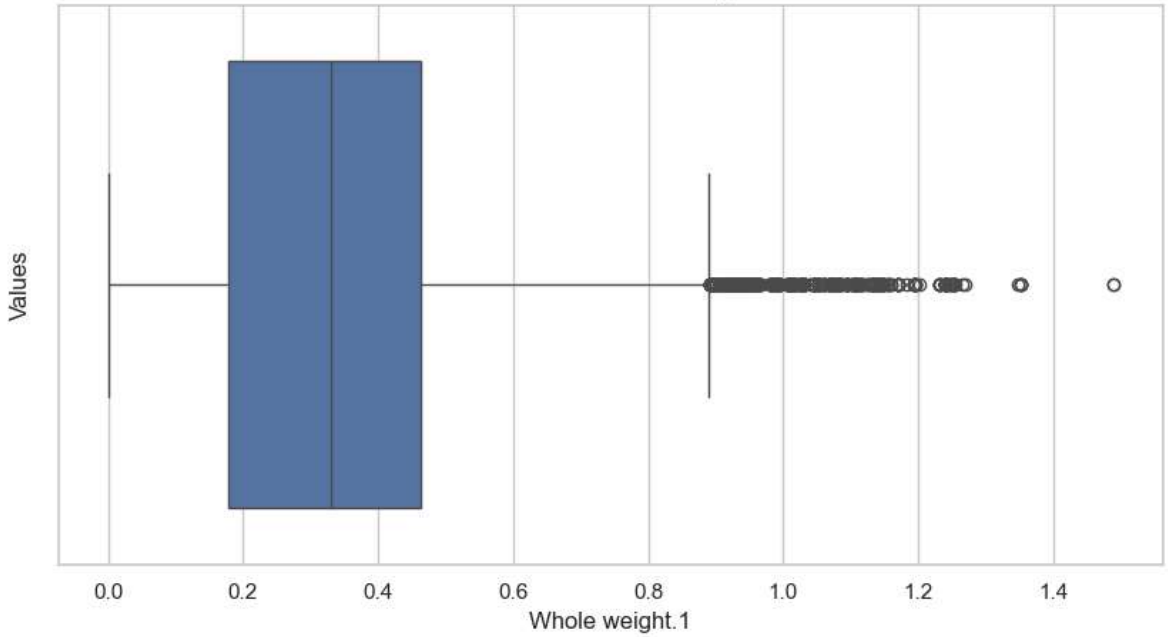




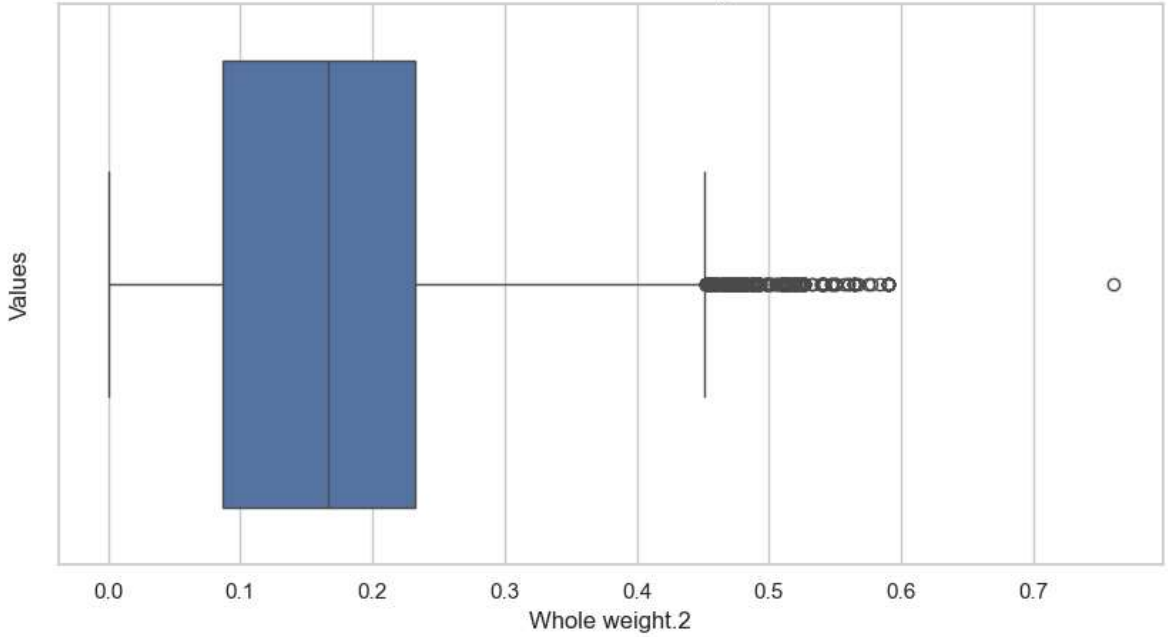
Box Plot of Whole weight



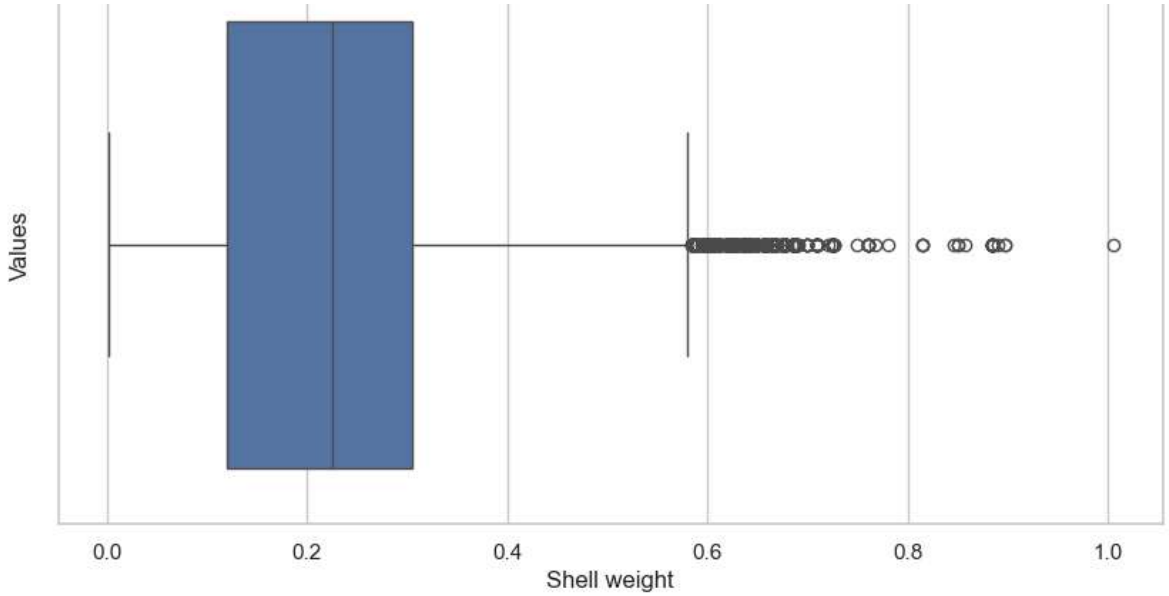
Box Plot of Whole weight.1

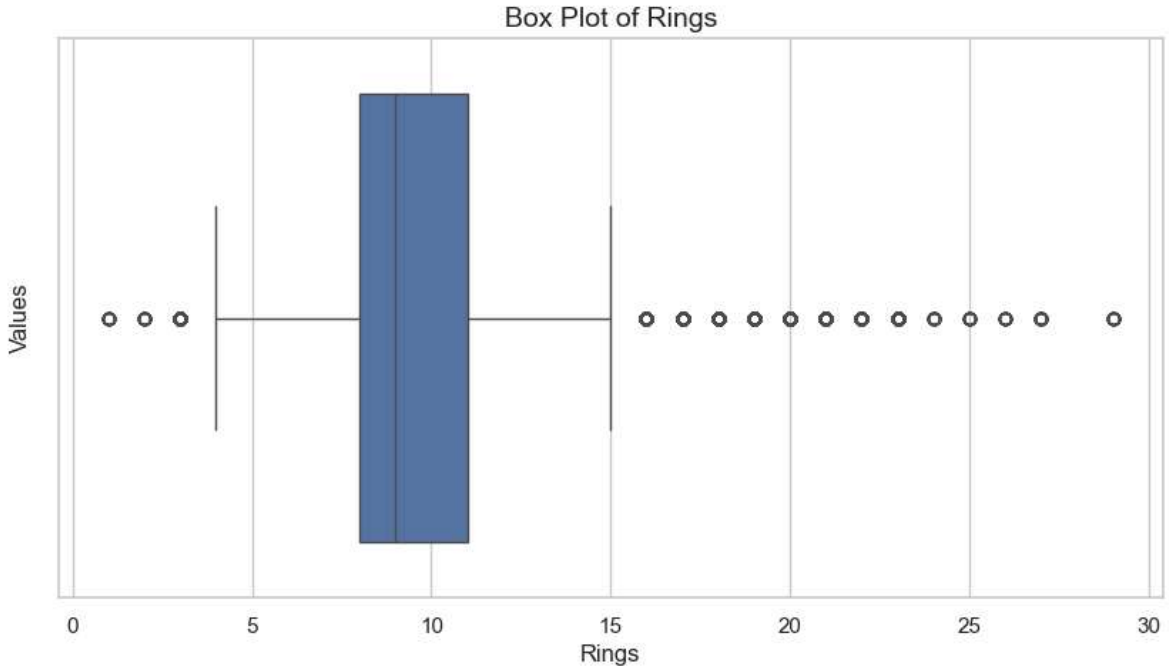


Box Plot of Whole weight.2



Box Plot of Shell weight

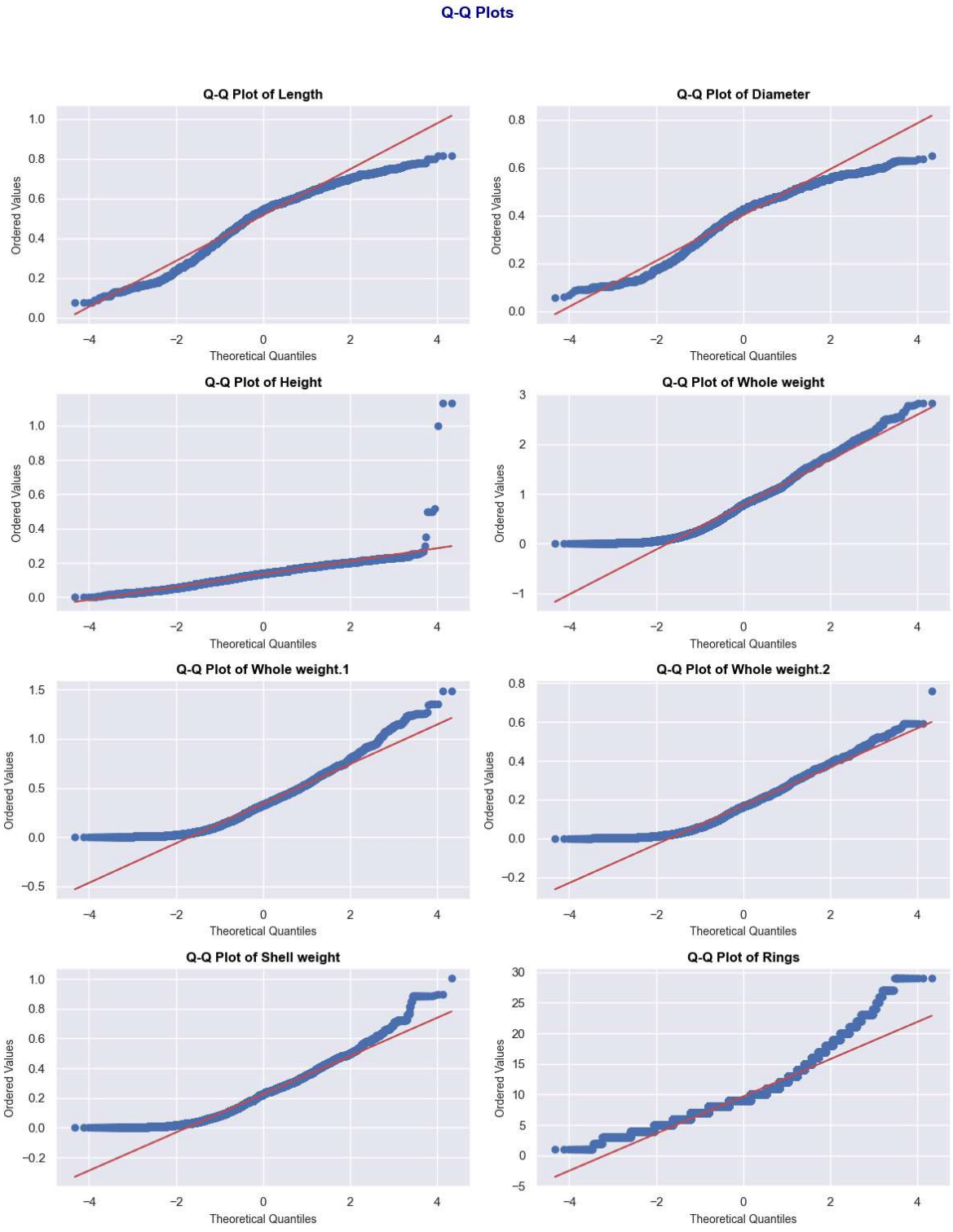




Q-Q Plot

In [ ]:
```
import scipy.stats as stats
sns.set_style(style="darkgrid")
fig, axs = plt.subplots(nrows=4, ncols=2, figsize=(12, 16))
axs = axs.flat

# Loop through each numerical feature and create a Q-Q plot
for i, num_feat in enumerate(numeric_features):
    # Generate Q-Q plot
    stats.probplot(df[num_feat], dist="norm", plot=axs[i])

    # Add reference line
    axs[i].plot(axs[i].get_lines()[1].get_xdata(), axs[i].get_lines()[1].get_yda
    axs[i].set_title(f"Q-Q Plot of {num_feat}", fontsize=12, fontweight='bold',
    axs[i].set_xlabel("Theoretical Quantiles", fontsize=10)
    axs[i].set_ylabel("Ordered Values", fontsize=10)

fig.suptitle("Q-Q Plots", fontsize=14, fontweight="bold", color="darkblue")
```

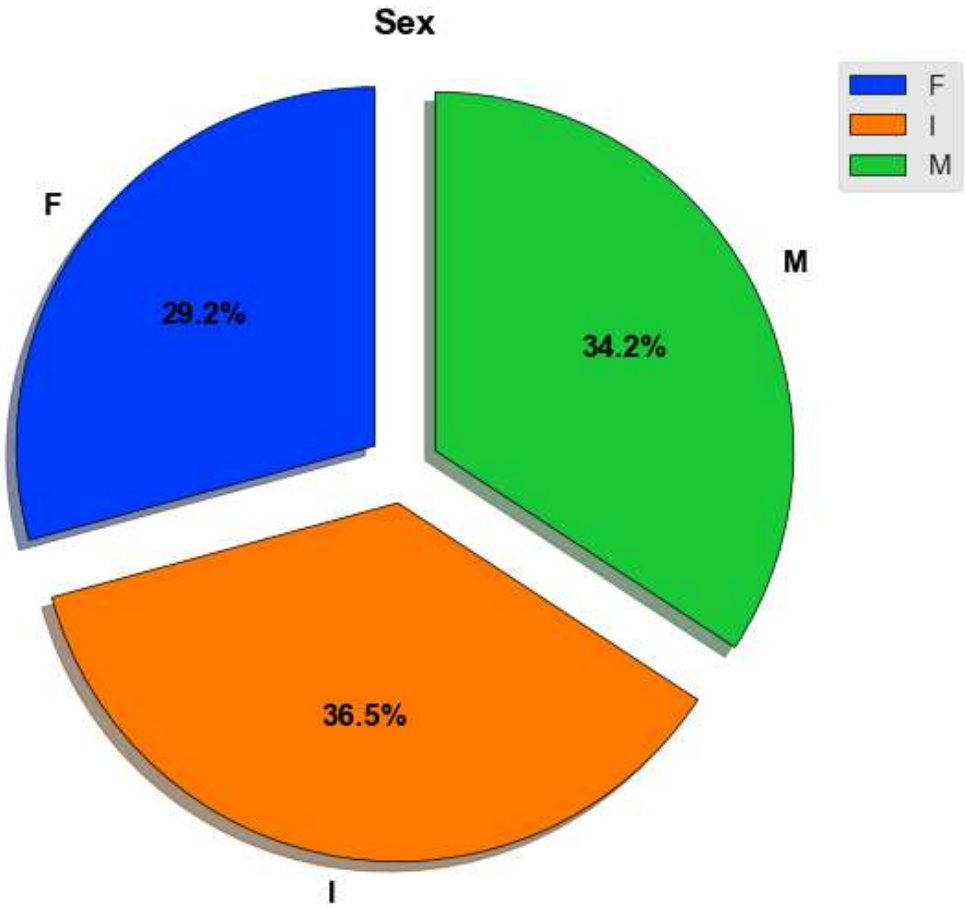
```
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Pie Plot

In [ ]:
```
from collections import Counter
plt.style.use("ggplot")
category_data = df[categoric_features[0]]
category_counts = Counter(category_data)

plt.figure(figsize=(8, 6))
plt.pie(x=list(category_counts.values()),
        labels=list(category_counts.keys()),
```

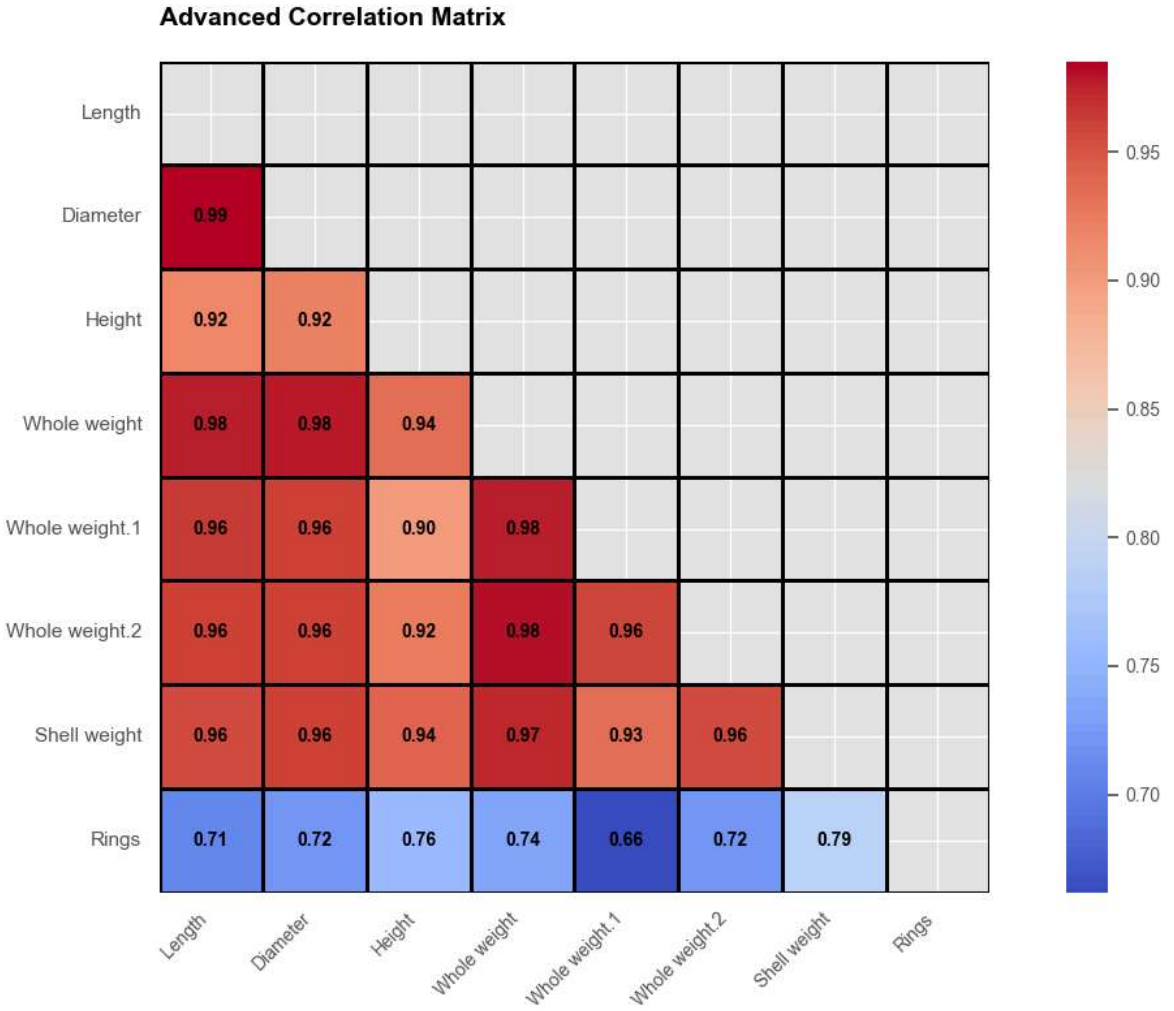
```
        colors=[color for color in sns.color_palette(palette='bright', n_colors=
        shadow=True,
        wedgeprops={'edgecolor': 'black'},
        textprops={'fontsize': 12, 'fontweight': 'bold', 'color': 'black'},
        autopct="%.1f%%",
        startangle=90,
        explode=[0.1] * len(category_counts))

plt.title(categoric_features[0], fontsize=14, fontweight='bold', color='black')
plt.legend(loc="upper right")
plt.axis("equal")
plt.show()
```



Correlation Matrix

In [ ]:
```
corr_matrix = df[numeric_features].corr(method="spearman")
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
fig, ax = plt.subplots(figsize=(12, 8))

sns.heatmap(corr_matrix,
            cmap="coolwarm",
            annot=True,
            fmt=".2f",
            annot_kws={'fontsize': 10, 'fontweight': 'bold', 'color': 'black'},
            linewidths=1,
            linecolor='black',
            square=True,
            mask=mask,
            ax=ax)
```

```
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right
ax.set_title("Advanced Correlation Matrix", fontsize=14, fontweight="bold", colo
cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=10)
for i in range(len(corr_matrix)):
    ax.axhline(i, color='black', lw=1)
    ax.axvline(i, color='black', lw=1)

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



Feature Processing

In [ ]:
```
test = pd.read_csv('playground-series-s4e4/test.csv')
test.head()
```

Out[ ]:

	id	Sex	Length	Diameter	Height	Whole weight	Whole weight.1	Whole weight.2	Shell weight
0	90615	M	0.645	0.475	0.155	1.2380	0.6185	0.3125	0.3005
1	90616	M	0.580	0.460	0.160	0.9830	0.4785	0.2195	0.2750
2	90617	M	0.560	0.420	0.140	0.8395	0.3525	0.1845	0.2405
3	90618	M	0.570	0.490	0.145	0.8740	0.3525	0.1865	0.2350
4	90619	I	0.415	0.325	0.110	0.3580	0.1575	0.0670	0.1050

In [ ]:
```
ordinal = OrdinalEncoder(dtype='float')
```

In [ ]:
```
train[categoric_features] = ordinal.fit_transform(train[categoric_features])
test[categoric_features] = ordinal.transform(test[categoric_features])
```

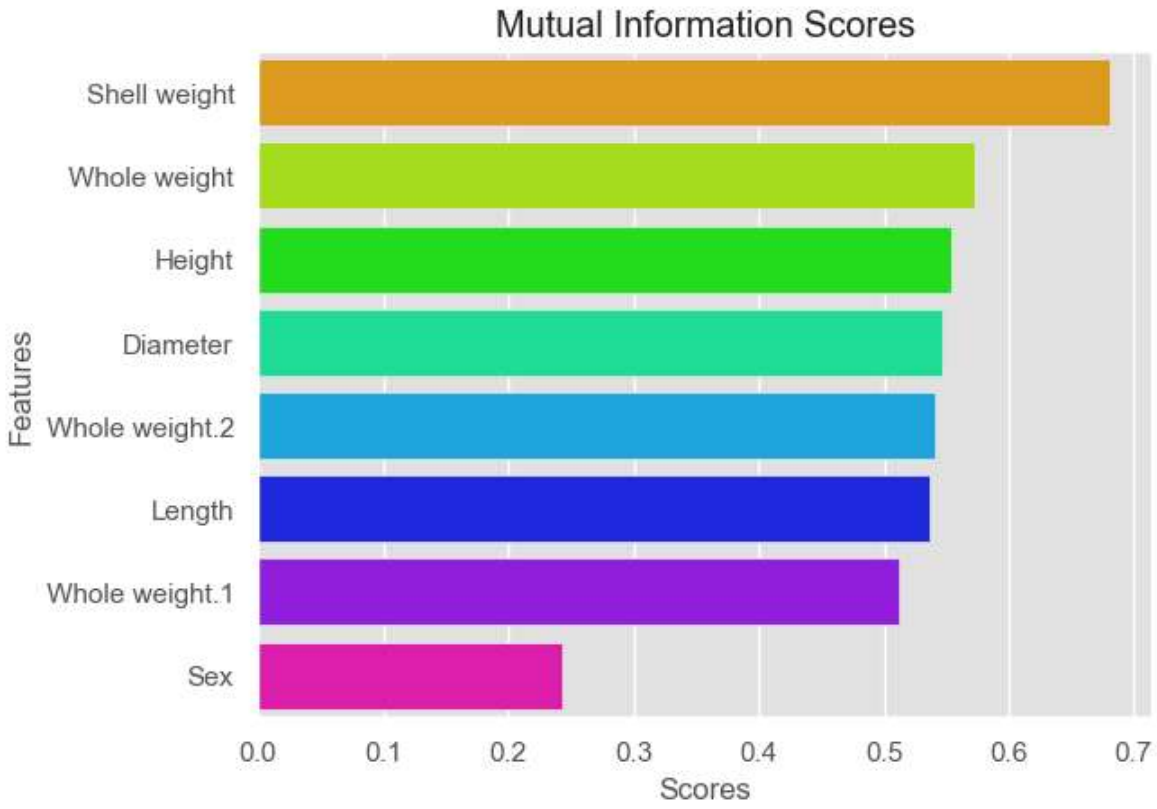
Model Training

In [ ]:
```
test = test.drop(columns=['id'], axis=1)
train = train.drop(columns=['id'], axis=1)


y = train.Rings
X = train.drop(['Rings'], axis=1)
```

In [ ]:
```
from sklearn.feature_selection import mutual_info_regression

discrete_features = X.dtypes == int

def mi_score_maker(X,y,discrete_features):
    scores = mutual_info_regression(X,y,discrete_features=discrete_features)
    df = pd.DataFrame({
        'Features':X.columns,
        'Scores':scores
    })
    df = df.sort_values(['Scores'],ascending=False).reset_index(drop=True)
    return df
```

In [ ]:
```
mi_scores = mi_score_maker(X,y.astype('float64'),discrete_features)
```

In [ ]:
```
colors = sns.color_palette('hsv', len(mi_scores))
sns.barplot(x='Scores',y='Features',data=mi_scores, palette=colors)
plt.title("Mutual Information Scores")
plt.show()
```



In [ ]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_st
```

In [ ]:
```
xgb_params = {
    'verbosity': 0,
    "early_stopping_rounds":25,
    'n_estimators': 1500,
    'eval_metric': 'rmse',
    'random_state':1234,
    'max_depth': 8,
    'subsample': 0.79,
    'objective': 'reg:squarederror',
    'learning_rate': 0.013,
    'colsample_bytree': 0.56,
    "min_child_weight": 10,
}

xgb_model = XGBRegressor(**xgb_params)
```

In [ ]:
```
xgb_model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=False)
```

Out[ ]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.56, device=None, early_stopping_roun
ds=25,
             enable_categorical=False, eval_metric='rmse', feature_t
ypes=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.013, max_
bin=None,
```

Predictions

In [ ]:
```
pred = xgb_model.predict(X_val)
```

In [ ]:
```
pred
```

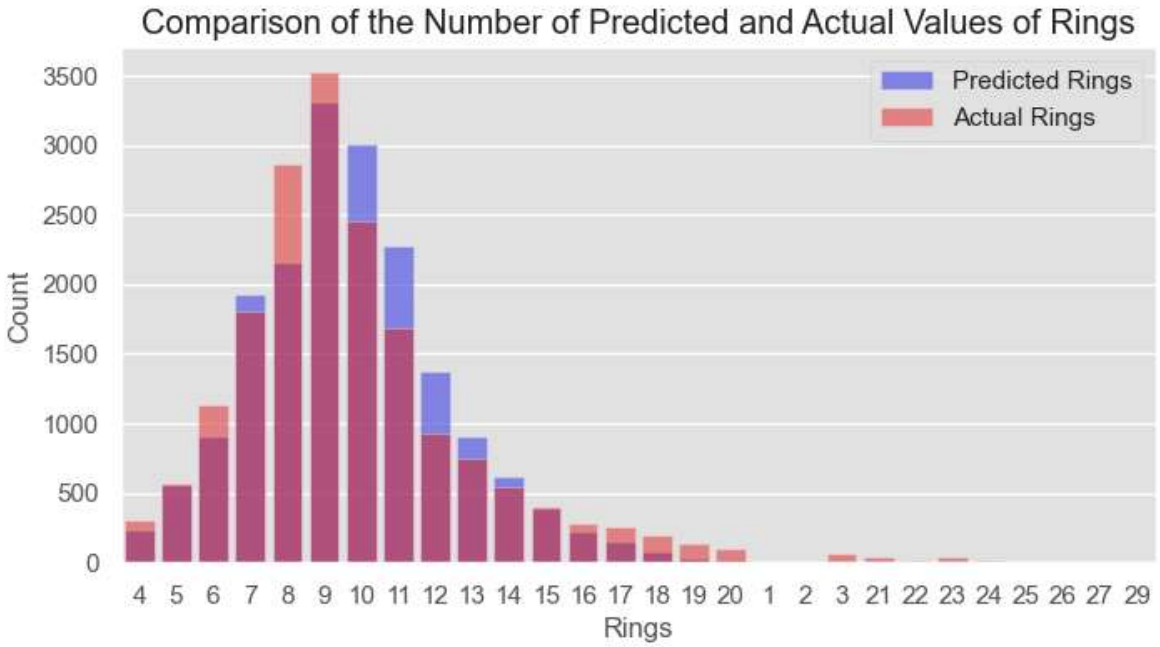
Out[ ]:
```
array([ 8.729883,  9.5491  ,  7.203663, ...,  9.503734,  9.675576,
       11.117143], dtype=float32)
```

In [ ]:
```
print("The RMSLE score is: ", mean_squared_log_error(pred, y_val)**0.5)
```

```
The RMSLE score is:  0.14970180113973386
```

In [ ]:
```
predictions = xgb_model.predict(test)
```

In [ ]:
```
%%time

df_pred = pd.DataFrame({'Rings': pred.round().astype(int)})
df_vald = pd.DataFrame({'Rings': y_val})
plt.figure(figsize=(8, 4))
sns.countplot(x='Rings', data=df_pred, color='blue', alpha=0.5, label='Predicted
sns.countplot(x='Rings', data=df_vald, color='red', alpha=0.5, label='Actual Rin
plt.title("Comparison of the Number of Predicted and Actual Values of Rings")
plt.xlabel('Rings')
plt.ylabel('Count')
plt.legend()
plt.show()
```




```
CPU times: total: 281 ms
Wall time: 655 ms
```

Submission

V1

In [ ]:
```
# submission = pd.read_csv('playground-series-s4e4/sample_submission.csv')

# submission['Rings'] = predictions
# submission.to_csv('submission.csv', index=False)
```

V2

In [ ]:
```
submission = pd.read_csv('playground-series-s4e4/sample_submission.csv')

submission['Rings'] = predictions
submission.to_csv('submission2.csv', index=False)
```