In [142]:

```
# Load in basic libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [143]:

```
# Import the file
abalone = pd.read_csv('abalone.csv')
```

In [144]:

```
# Original data length
print(str(len(abalone.index))+' rows')
```

```
4177 rows
```

In [146]:

```


# View file
abalone.sample(5)
```

Out[146]:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
1012	M	0.625	0.490	0.175	1.3325	0.5705	0.2710	0.4050	10
1350	F	0.595	0.465	0.150	0.9800	0.4115	0.1960	0.2255	10
3434	I	0.370	0.280	0.090	0.2565	0.1255	0.0645	0.0645	6
2406	F	0.580	0.460	0.185	1.0170	0.3515	0.2000	0.3200	10
1784	M	0.530	0.410	0.140	0.7545	0.3495	0.1715	0.2105	8

In [147]:

```
# Confirm how many null values there are
abalone.isnull().sum()

        # No missing data to handle
```

Out[147]:

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

In [148]:

```
# Check data types
abalone.dtypes
```

Out[148]:

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

In [149]:

```
# View the unique sex categories
abalone['Sex'].unique()
```

Out[149]:

```
array(['M', 'F', 'I'], dtype=object)
```

In [161]:

```
# Encode the sex variables into binary
sex_enc = pd.get_dummies(abalone['Sex'])
sex_enc.head()
```

Out[161]:

	F	I	M
0	0	0	1
1	0	0	1
2	1	0	0
3	0	0	1
4	0	1	0

In [162]:

```
# Concatenate the data frames and confirm that the encoding seems correct
abalone_df = pd.concat([abalone, sex_enc], axis=1)
abalone_df.head() # everything looks good
```

Out[162]:

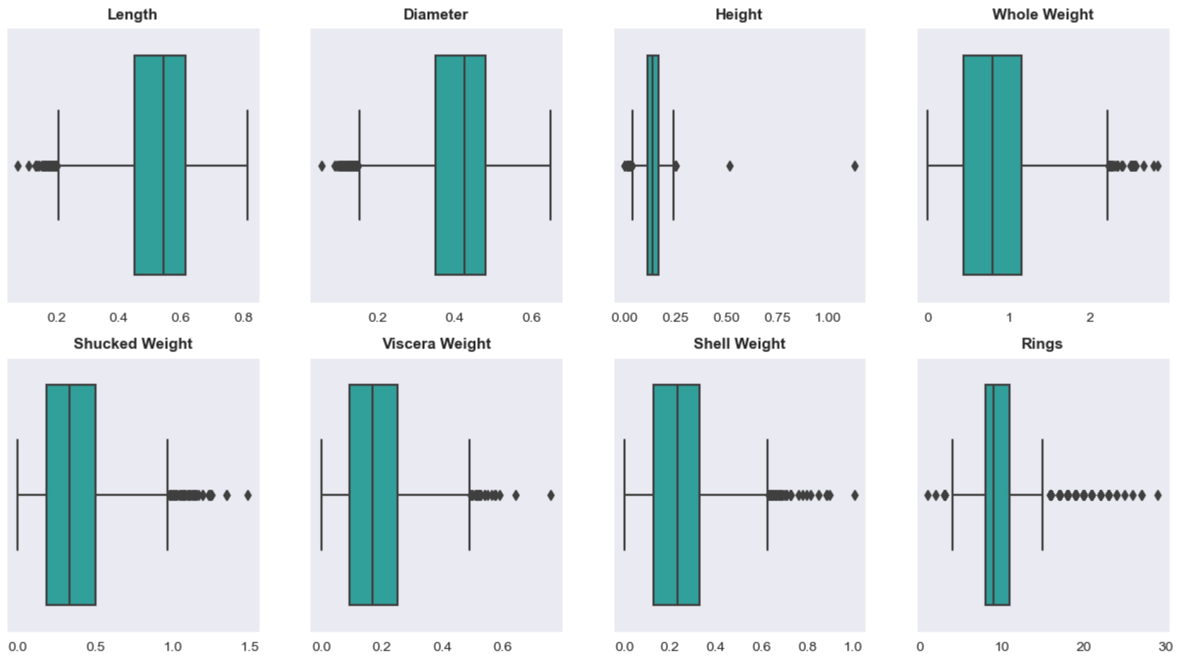
	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	F	I	M
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	0	0	1
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	0	0	1
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	1	0	0
3	M	0.440	0.365	0.125	0.5160	0.2155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

In [163]:

```
# Drop the original sex variable from data frame
abalone_df = abalone_df.drop('Sex', axis=1)
```

In [164]:

```
# Check the distribution of each variable (other than sex)

# Set up subplots
fig, axes = plt.subplots(2,4)

# Set up the box plots in each subplot
sns.boxplot(x='Length', color='lightseagreen', data=abalone_df, ax=axes[0,0])
sns.boxplot(x='Diameter', color='lightseagreen', data=abalone_df, ax=axes[0,1])
sns.boxplot(x='Height', color='lightseagreen', data=abalone_df, ax=axes[0,2])
sns.boxplot(x='Whole weight', color='lightseagreen', data=abalone_df, ax=axes[0,3])
sns.boxplot(x='Shucked weight', color='lightseagreen', data=abalone_df, ax=axes[1,0])
sns.boxplot(x='Viscera weight', color='lightseagreen', data=abalone_df, ax=axes[1,1])
sns.boxplot(x='Shell weight', color='lightseagreen', data=abalone_df, ax=axes[1,2])
sns.boxplot(x='Rings', color='lightseagreen', data=abalone_df, ax=axes[1,3])

# Format the axes
axes[0,0].set_title('Length', size=11, weight='bold')
axes[0,0].set_xlabel('')

axes[0,1].set_title('Diameter', size=11, weight='bold')
axes[0,1].set_xlabel('')

axes[0,2].set_title('Height', size=11, weight='bold')
axes[0,2].set_xlabel('')

axes[0,3].set_title('Whole Weight', size=11, weight='bold')
axes[0,3].set_xlabel('')

axes[1,0].set_title('Shucked Weight', size=11, weight='bold')
axes[1,0].set_xlabel('')

axes[1,1].set_title('Viscera Weight', size=11, weight='bold')
axes[1,1].set_xlabel('')

axes[1,2].set_title('Shell Weight', size=11, weight='bold')
axes[1,2].set_xlabel('')

axes[1,3].set_title('Rings', size=11, weight='bold')
axes[1,3].set_xlabel('')

sns.set_style('dark')

# Make plots larger
plt.rcParams.update({'figure.figsize':(15,8), 'figure.dpi':100})

# Show plot
plt.show()
```



In [165]:

```
# Remove heights greater than 0.250 (2051 & 1417)
# Length less than 0.075 (236)
# Shucked weight > 1.488 (1209)
# Viscera weight > 0.76 (1763)
# Shell weight > 1.005 (163)

abalone_df.sort_values('Height', ascending=False)
```

Out[165]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	F	I	M
2051	0.455	0.355	1.130	0.5940	0.3320	0.1160	0.1335	8	1	0	0
1417	0.705	0.565	0.515	2.2100	1.1075	0.4865	0.5120	10	0	0	1
1763	0.775	0.630	0.250	2.7795	1.3485	0.7600	0.5780	12	0	0	1
1428	0.815	0.650	0.250	2.2550	0.8905	0.4200	0.7975	14	1	0	0
2179	0.595	0.470	0.250	1.2830	0.4620	0.2475	0.4450	14	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...
1174	0.635	0.495	0.015	1.1565	0.5115	0.3080	0.2885	9	1	0	0
2169	0.165	0.115	0.015	0.0145	0.0055	0.0030	0.0050	4	0	1	0
236	0.075	0.055	0.010	0.0020	0.0010	0.0005	0.0015	1	0	1	0
1257	0.430	0.340	0.000	0.4280	0.2065	0.0860	0.1150	8	0	1	0
3996	0.315	0.230	0.000	0.1340	0.0575	0.0285	0.3505	6	0	1	0

4177 rows × 11 columns

In [166]:

```
abalone_removed = abalone_df.drop([2051, 1417, 236, 1209, 1763, 163])
abalone_removed.head()
```

Out[166]:

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

In [167]:

```
import warnings
warnings.filterwarnings('ignore')

# Create age variable
# According to the data information, the age is equal to the number of rings + 1.5
abalone_removed['Age'] = abalone_removed['Rings'] + 1.5
```

In [168]:

```
# Set the final data frame
abalone_new = abalone_removed[['Age', 'Length', 'Diameter', 'Height', 'Whole weight', 'Shuc
                                 'Viscera weight', 'Shell weight', 'M', 'F', 'I']]

# View data frame
abalone_new.head()
```

Out[168]:

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

In [169]:

```
# Create the correlation matrix
abalone_new.corr()

# Revise the above to eliminate the perfect correlations (vars with themselves)
# Also eliminate duplicate entries
# This will make our matrix easier to interpret

import warnings
warnings.filterwarnings('ignore')

# First get the correlations from our data frame
abalone_corr = abs(abalone_new.corr())

# Convert to easier format (for visualization)
abalone_corr_tri = abalone_corr.mask(np.triu(np.ones(abalone_corr.shape)).astype(np.bool))

# Now, we no longer have the duplicates, nor the perfect correlations!
```

In [170]:

```
# Drop the first row and the last column, since those are only NAs
abalone_corr_tri = abalone_corr_tri.drop(index='Age', columns='I')
abalone_corr_tri.head(10)
```

Out[170]:

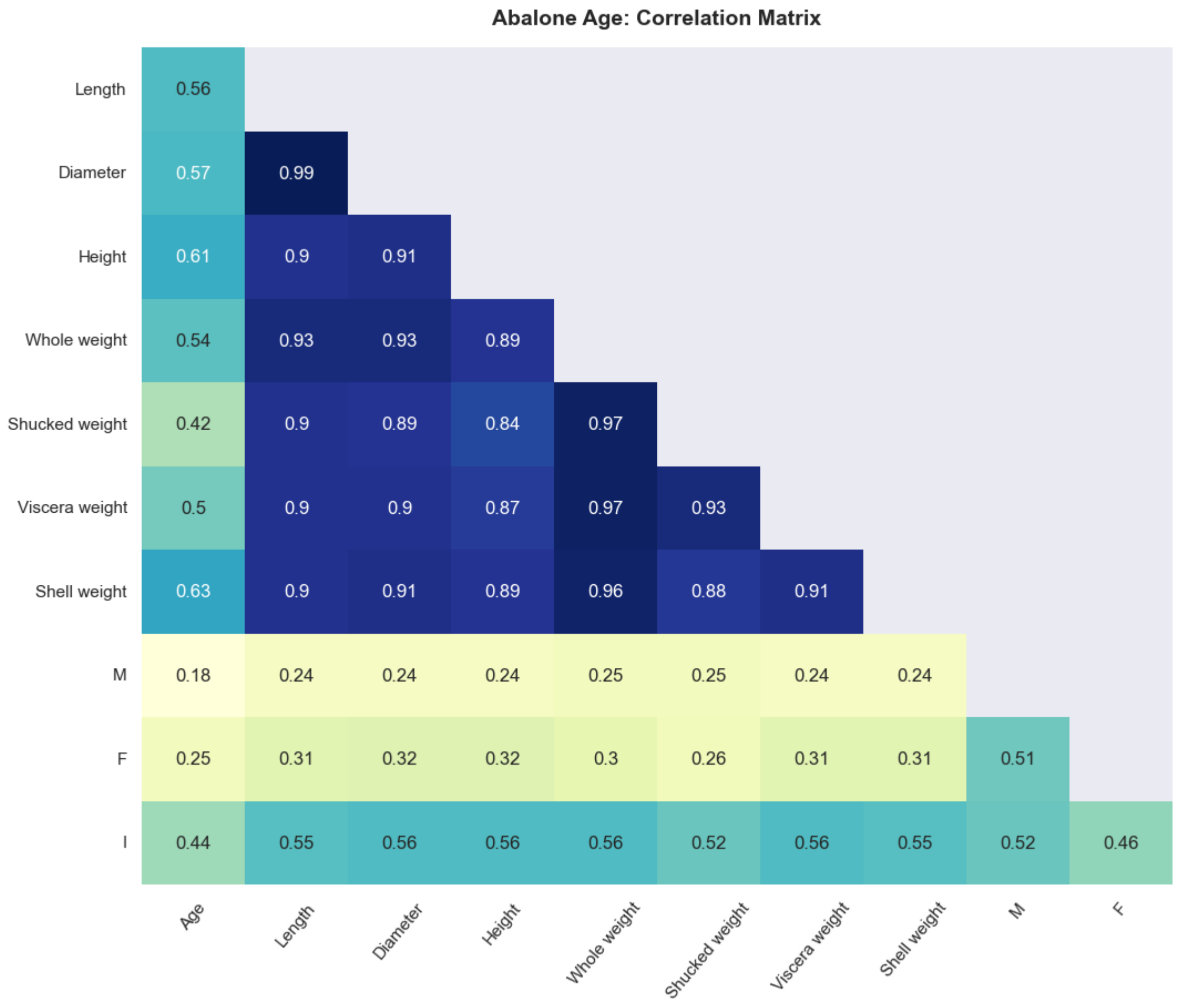
	Age	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	
Length	0.555525	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Diameter	0.573706	0.986734	NaN	NaN	NaN	NaN	NaN	NaN	
Height	0.608716	0.899299	0.905573	NaN	NaN	NaN	NaN	NaN	
Whole weight	0.540746	0.926155	0.926173	0.887753	NaN	NaN	NaN	NaN	
Shucked weight	0.421765	0.899555	0.894463	0.836762	0.969532	NaN	NaN	NaN	
Viscera weight	0.504390	0.904139	0.900592	0.865743	0.966322	0.931817	NaN	NaN	
Shell weight	0.627315	0.899077	0.906802	0.890024	0.956366	0.884768	0.909120	NaN	
M	0.181973	0.236282	0.240079	0.235900	0.252292	0.252208	0.241782	0.236588	
F	0.249827	0.309842	0.318886	0.316712	0.300476	0.264976	0.310157	0.306306	0.51
I	0.435629	0.551204	0.564097	0.557629	0.558420	0.523098	0.557190	0.548011	0.52

In [171]:

```
# Round everything to 2 decimal places so that heatmap is easier to read
abalone_corr_tri2 = abalone_corr_tri.round(2)
```

In [172]:

```
sns.set(rc={'figure.figsize':(12,10)})

# Visually display the matrix from above
# use the yellow-green-blue color map so that high correlations are more easily identifiabl
h = sns.heatmap(abalone_corr_tri2, annot=True, cmap="YlGnBu", cbar=False)

# Add plot title
h.set_title('Abalone Age: Correlation Matrix', size=14, weight='bold', pad=15)

# rotate tick marks so they are easier to read
plt.xticks(rotation=50)

# Display plot
plt.show()
```



In [173]:

```
# features to keep
abalone_final = abalone_new.drop(['Diameter', 'Length', 'Whole weight', 'M', 'F'], axis=1)
# View refined data set
abalone_final.head()
```

Out[173]:

	Age	Height	Shucked weight	Viscera weight	Shell weight	I
0	16.5	0.095	0.2245	0.1010	0.150	0
1	8.5	0.090	0.0995	0.0485	0.070	0
2	10.5	0.135	0.2565	0.1415	0.210	0
3	11.5	0.125	0.2155	0.1140	0.155	0
4	8.5	0.080	0.0895	0.0395	0.055	1

In [174]:

```
# Import necessary modules
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import confusion_matrix, classification_report, plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Predictors
X = abalone_final.drop('Age', axis=1)
# Outcome
y = abalone_final['Age']

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=33)

#Standardize
sc= StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.fit_transform(X_test)
```

In [175]:

```
# Set up the classifier for the decision tree regressor
tree_model = DecisionTreeRegressor(random_state=33)

# Create our model by fitting the classifier to our data
tree_model = tree_model.fit(X_train, y_train)
# Find my predicted values
y_pred = tree_model.predict(X_test)

# Calculate the mean squared error
mse_tree = MSE(y_test, y_pred)
# Calculate the root mean squared error
rmse_tree = mse_tree**(1/2)
print(f' Original decision tree score: {rmse_tree}')
```

```
 Original decision tree score: 3.0313853474279115
```

In [176]:

```
# Import to search for best parameters
from sklearn.model_selection import RandomizedSearchCV

# Setup the parameters that we will test on our decision tree
param_dict = {"max_depth": [None, range(1,25)],
              #"min_samples_split":range(2,4),
              "min_samples_leaf": range(1,11),
              #"min_weight_fraction_leaf":np.arange(0,1,0.1),
              "max_features": [None, range(1,10)],
              "random_state":[33],
              "max_leaf_nodes":[None, range(1,10)],
              "min_impurity_decrease":[0.0,0.1]}

# Instantiate the classifier
trees = DecisionTreeRegressor()
```

In [177]:

```
# Run the decision tree model through the search, using the specified parameters and 10 cro
random = RandomizedSearchCV(trees,
                            n_iter = 20,
                            param_distributions = param_dict,
                            cv=10,
                            scoring = 'neg_root_mean_squared_error',
                            refit='neg_root_mean_squared_error',
                            n_jobs = -1,
                            random_state=33)

# Fit the search to our training data
random.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(random.best_params_))
print("Best decision tree score is {}".format(random.best_score_))
```

```
Tuned Decision Tree Parameters: {'random_state': 33, 'min_samples_leaf': 9,
'min_impurity_decrease': 0.0, 'max_leaf_nodes': None, 'max_features': None,
'max_depth': None}
Best decision tree score is -2.4365585492504236
```

In [178]:

```
# Import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

# Set up the model
rf = RandomForestRegressor(n_estimators=10,
                           random_state=33)

# Fit the model to our training data
rf.fit(X_train, y_train)
# Find predicted values
y_pred_rf = rf.predict(X_test)

# Calculate the mean squared error
mse_rf = MSE(y_test, y_pred_rf)
# Calculate the root mean squared error
rmse_rf = mse_rf**(1/2)
print(f'Original random forest score: {rmse_rf}')
```

```
Original random forest score: 2.277194549434554
```

In [179]:

```
# Setup the parameters that we will test on our random forest
param_dict = {"n_estimators": range(10,1000),
              "max_depth": [None, range(2,20)],
              "min_samples_leaf": range(1,20),
              "max_features": [None, range(1,20)],
              "max_leaf_nodes": [None, range(1,20)],
              "min_impurity_decrease": [0.0,0.1],
              "bootstrap": ["True", "False"],
              "oob_score": ["True", "False"],
              "random_state": [33],
              "warm_start": ["True", "False"],
              "max_samples": [None, range(1,20)]}

# Instantiate the classifier
trees = RandomForestRegressor()
```

In [180]:

```
# Run the random forest model through the search, using the specified parameters and 10 cro
random = RandomizedSearchCV(trees,
                            n_iter = 20,
                            param_distributions = param_dict,
                            cv=10,
                            scoring = 'neg_root_mean_squared_error',
                            refit='neg_root_mean_squared_error',
                            n_jobs = -1,
                            random_state=33)

# Fit the search to our training data
random.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Random Forest Parameters: {}".format(random.best_params_))
print("Best random forest score is {}".format(random.best_score_))
```

```
Tuned Random Forest Parameters: {'warm_start': 'False', 'random_state': 33,
'oob_score': 'False', 'n_estimators': 966, 'min_samples_leaf': 7, 'min_impur
ity_decrease': 0.0, 'max_samples': None, 'max_leaf_nodes': None, 'max_featur
es': None, 'max_depth': None, 'bootstrap': 'False'}
Best random forest score is -2.2066460478318604
```

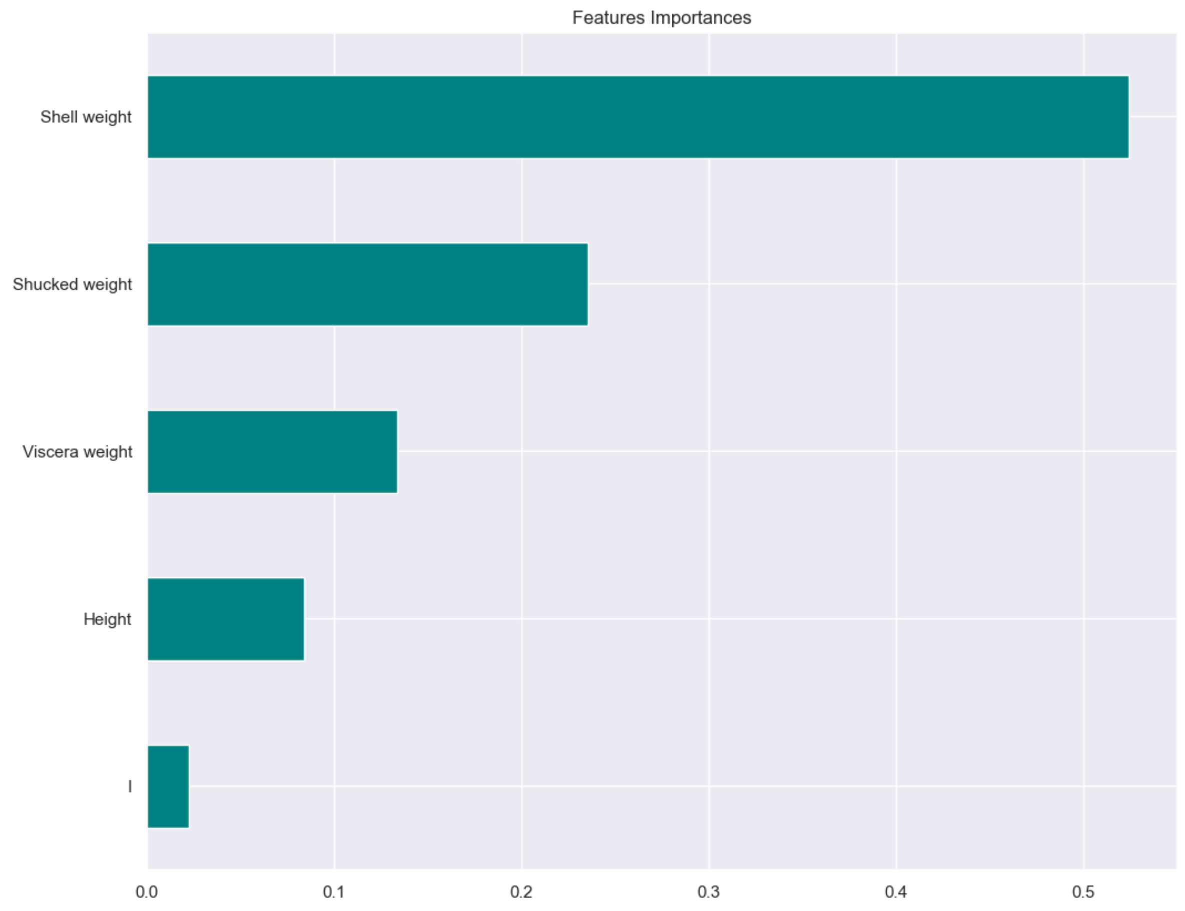
In [181]:

```
# Gives us the weights of the importance of each predictor (the feature importance)
print(tree_model.feature_importances_, X.columns)
```

```
[0.0841503  0.23524764 0.13381702 0.52410227 0.02268277] Index(['Height', 'S
hucked weight', 'Viscera weight', 'Shell weight', 'I'], dtype='object')
```

In [182]:

```
# Visualize which features are the most important in determining abalone age
import matplotlib.pyplot as plt

# Create pandas series
importances = pd.Series(data=tree_model.feature_importances_,
                        index= X.columns)

# Sort importances
importances_sorted = importances.sort_values()

# Draw a horizontal barplot of importances_sorted
importances_sorted.plot(kind='barh', color='teal')
plt.title('Features Importances')
plt.show()
```



In [ ]:

```
```