Predictive Models of the Age of Abalones Based on Physical Characteristics

Summary

In this report, we aim to create an effective model which can predict age of Abalone accurately based on individual physical characteristics. We perform exploratory data analysis and visualization, test both linear and non-linear models on our data, then compare models through regression metrics.

Authors: Serene Zha, Mehmet Imga, Claudia Liauw, Wendy Frankel

Introduction

Abalone are marine mollusks that are commercially important in fisheries and aquaculture, particularly in regions such as Tasmania. Estimating the age structure of abalone populations is essential for setting sustainable harvest limits and monitoring stock health. However, the standard method for determining age requires cutting the shell through the cone, staining it, and counting growth rings under a microscope—a destructive, time-consuming, and labor-intensive procedure[1]. Because of this, methods that can infer age from simple, non-destructive measurements of the animal are of practical interest to biologists, fisheries managers, and growers.

Here, we ask whether we can use a machine learning model to predict the age of an abalone from basic physical measurements. Specifically, we will explore linear regression models in Python to relate age to various attributes, including sex, shell length, diameter, height, and several weight measurements. To investigate this question, we use the UCI Abalone dataset, which contains 4,177 abalones with eight predictor variables and a target variable, “Rings.” Each row corresponds to one abalone, and the recorded features include sex (male, female, infant), three shell size measurements (length, diameter, height), and four weight measurements (whole, shucked, viscera, and shell weight). The number of rings serves as a proxy for age, with age in years given approximately by Rings + 1.5 [2], to account for any rings missed in counting (they can be difficult to see). By building and evaluating linear regression models on this dataset, we aim to understand how well these readily obtained physical measurements can predict abalone age and what this implies for practical, non-destructive age estimation.

Methods and Results

1. Load Necessary Packages

In [1]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
import pandera.pandas as pa
import altair as alt
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
```

2. Load Data

In [2]:
```
import requests
import zipfile

url = "https://archive.ics.uci.edu/static/public/1/abalone.zip"

request = requests.get(url)
with open("../data/raw/abalone.zip", 'wb') as f:
    f.write(request.content)

with zipfile.ZipFile("../data/raw/abalone.zip", 'r') as zip_ref:
    zip_ref.extractall("../data/raw")
```

In [3]:
```
 # fetch dataset
abalone = fetch_ucirepo(id=1)

# Extract features and targets
X = abalone.data.features
y = abalone.data.targets
```

3. Data Wrangling and Cleaning

Split Data

In [4]:
```
# Split Data (Same random_state as baseline for comparison)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=522
)

train_df = pd.concat([X_train, y_train], axis=1)
test_df = pd.concat([X_test, y_test], axis=1)
```

```
# ravel y for sklearn
y_train = y_train.values.ravel()
y_test = y_test.values.ravel()

print("Data loaded and split.")

train_df
```

```
Data loaded and split.
```

Out[4]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

Save Data

In [5]:
```
train_df.to_csv('../data/processed/abalone_train.csv', index=False)
test_df.to_csv('../data/processed/abalone_test.csv', index=False)
```

4. Data Validation

Correct Data File Format

In [6]:
```
assert isinstance(train_df, pd.DataFrame), "Expected 'train_df' to be a Pandas Data
```

Data Validation Using Panderas

In [7]:
```
#This schema checks column types, and that there are no NULL values in the feature

schema = pa.DataFrameSchema({
    "Sex": pa.Column(str, nullable=False),
    "Length": pa.Column(float, nullable=False),
    "Diameter": pa.Column(float, nullable=False),
    "Height": pa.Column(float, nullable=False),
    "Whole_weight": pa.Column(float, nullable=False),
    "Shucked_weight": pa.Column(float, nullable=False),
    "Viscera_weight": pa.Column(float, nullable=False),
    "Shell_weight": pa.Column(float, nullable=False),
    "Rings": pa.Column(int)
}
)

schema.validate(train_df, lazy=True)
```

Out[7]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

In [8]:
```
#This checks that there are not more than 5% missing values in the target column

schema = pa.DataFrameSchema(
    {
        "Rings": pa.Column(int,
                                pa.Check(lambda s: s.isna().mean() <= 0.05,
                                    element_wise=False,
                                    error="Too many null values in 'Rings' column."
                                nullable=True)
    }
)

schema.validate(train_df, lazy=True)
```

Out[8]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

In [9]:
```
# checking that numeric features are within range; no extreme outliers

schema = pa.DataFrameSchema(
    {
        "Length": pa.Column(float, pa.Check.between(0, 1)),
        "Diameter": pa.Column(float, pa.Check.between(0, 1)),
        "Height": pa.Column(float, pa.Check.between(0, 1)),
        "Whole_weight": pa.Column(float, pa.Check.between(0, 3)),
        "Shucked_weight": pa.Column(float, pa.Check.between(0, 2)),
        "Viscera_weight": pa.Column(float, pa.Check.between(0, 1)),
        "Shell_weight": pa.Column(float, pa.Check.between(0, 1.10)),
        "Rings": pa.Column(int, pa.Check.between(0, 30))
    }
)

schema.validate(train_df, lazy=True)
```

Out[9]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

In [10]:

```
#checking that the 'sex' column only has the values M, F, or I

schema = pa.DataFrameSchema(
    {
        "Sex": pa.Column(str, pa.Check.isin(["M", "F", "I"]), nullable = False)
    }
)

schema.validate(train_df, lazy=True)
```

Out[10]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

In [11]:

```
#checking for duplicates

schema = pa.DataFrameSchema(
    checks=[
        pa.Check(lambda train_df: ~train_df.duplicated().any(), error="Duplicate ro
    ]
)

schema.validate(train_df, lazy=True)
```

Out[11]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

In [12]:

```
#checking for empty observations

schema = pa.DataFrameSchema(
    checks=[
        pa.Check(lambda train_df: ~train_df.duplicated().any(), error="Duplicate ro
        pa.Check(lambda train_df: ~(train_df.isna().all(axis=1)).any(), error="Empt
    ]
)

schema.validate(train_df, lazy=True)
```

Out[12]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	She
2194	I	0.430	0.325	0.110	0.3675	0.1355	0.0935	
3996	I	0.315	0.230	0.000	0.1340	0.0575	0.0285	
3329	F	0.545	0.435	0.150	0.6855	0.2905	0.1450	
492	F	0.655	0.510	0.155	1.2895	0.5345	0.2855	
241	I	0.270	0.200	0.070	0.1000	0.0340	0.0245	
...	...	...	...	...	...	...	...	
3956	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	
154	F	0.565	0.450	0.135	0.9885	0.3870	0.1495	
3360	F	0.580	0.440	0.175	1.0730	0.4005	0.2345	
1899	M	0.575	0.450	0.130	0.7850	0.3180	0.1930	
3988	M	0.665	0.515	0.165	1.3855	0.6210	0.3020	

3341 rows × 9 columns

Distribution of Target

In [13]:
```
sns.boxplot(data=train_df, x='Rings')
```

Out[13]:
```
<Axes: xlabel='Rings'>
```

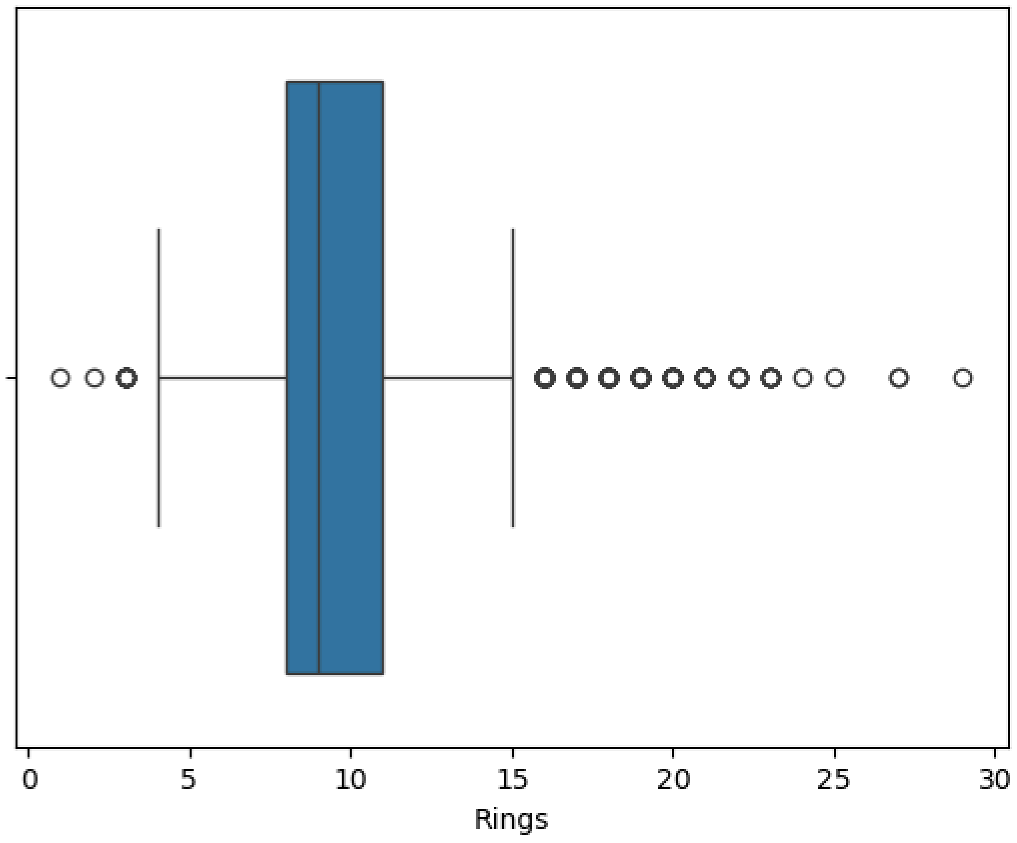


Figure 1: Boxplot showing the outliers of target variable Rings.

Observation: There are outliers, but no anomalous values below 0 or very high.

```
In [14]: sns.histplot(data=train_df, x='Rings', binwidth=1)
```

```
Out[14]: <Axes: xlabel='Rings', ylabel='Count'>
```

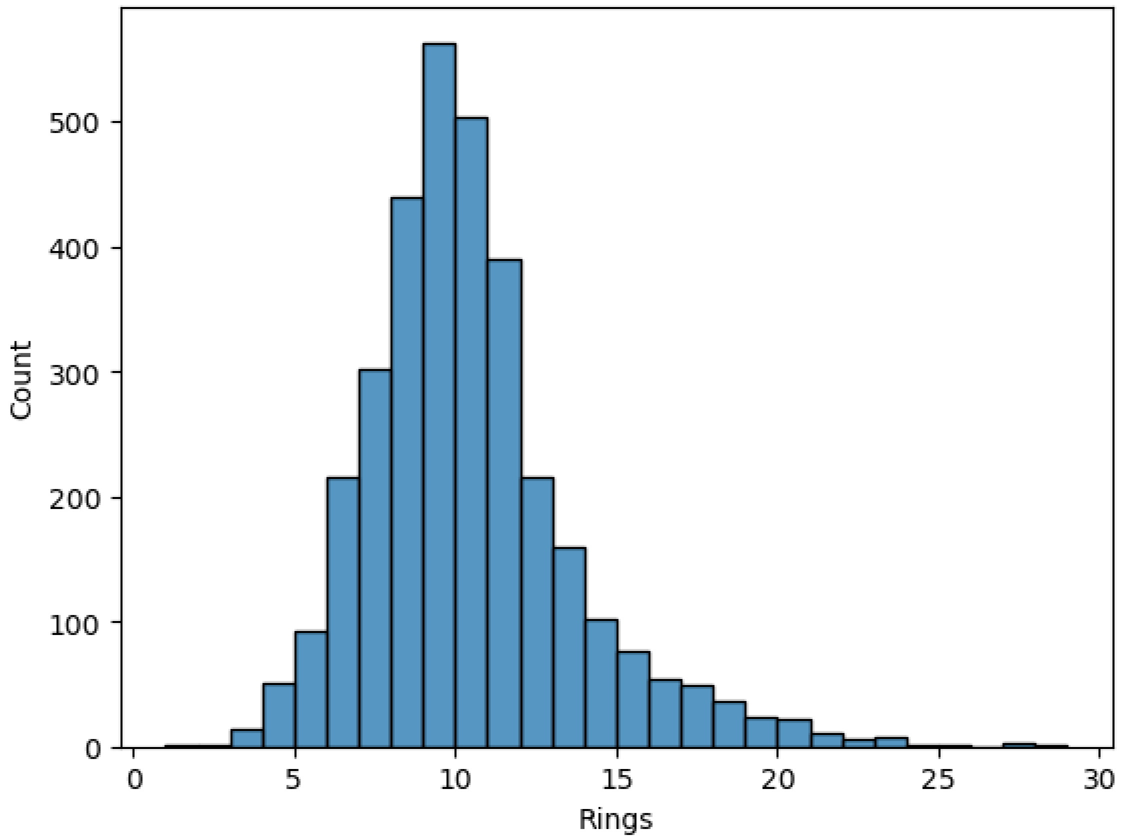


Figure 2: Boxplot showing the outliers of target variable Rings.

Observation: There is a bit of right skew.

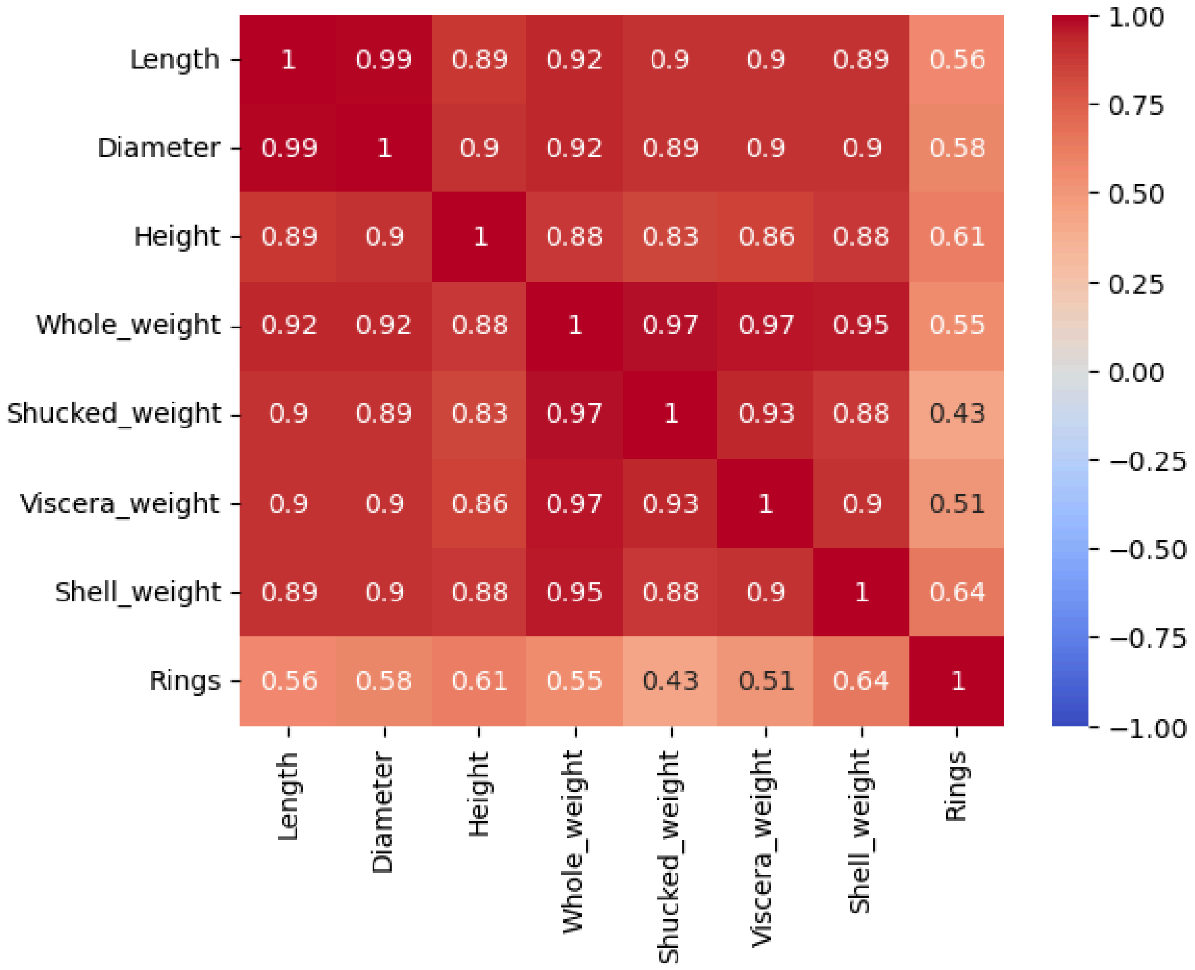
```
In [15]: from scipy.stats import shapiro
         normal_pvalue = shapiro(train_df.Rings).pvalue
         try:
             assert normal_pvalue > 0.05
         except AssertionError:
             print(f"Target variable is not normal! Shapiro p-value: {normal_pvalue}")
```

```
Target variable is not normal! Shapiro p-value: 1.7121391423967805e-36
```

Correlations

```
In [16]: corr_matrix = train_df.select_dtypes(include=['float64', 'int64']).corr()
         sns.heatmap(corr_matrix, annot=True, vmin=-1, cmap='coolwarm')
```

```
Out[16]: <Axes: >
```



	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight	Rings
Length	1	0.99	0.89	0.92	0.9	0.9	0.89	0.56
Diameter	0.99	1	0.9	0.92	0.89	0.9	0.9	0.58
Height	0.89	0.9	1	0.88	0.83	0.86	0.88	0.61
Whole_weight	0.92	0.92	0.88	1	0.97	0.97	0.95	0.55
Shucked_weight	0.9	0.89	0.83	0.97	1	0.93	0.88	0.43
Viscera_weight	0.9	0.9	0.86	0.97	0.93	1	0.9	0.51
Shell_weight	0.89	0.9	0.88	0.95	0.88	0.9	1	0.64
Rings	0.56	0.58	0.61	0.55	0.43	0.51	0.64	1

Figure 3: Heatmap showing the correlation between numerical features and the target variable Rings, and within numerical features.

Observation: There are no anomalous correlations between Rings and features, but there are high correlations between most features. This can possibly be accounted for through L2 regularisation.

In [17]:
```
threshold = 0.9
exceeds_threshold = [(corr_matrix.index[i], corr_matrix.columns[j], corr_matrix.ilo
 for i, j in np.argwhere(corr_matrix > threshold)]
try:
    assert len(exceeds_threshold) == 0
except AssertionError:
    print(f"Anomalous correlations above {threshold}!")
    print(exceeds_threshold)
```

```
Anomalous correlations above 0.9!
[('Length', 'Length', np.float64(1.0)), ('Length', 'Diameter', np.float64(0.9867)),
('Length', 'Whole_weight', np.float64(0.9237)), ('Length', 'Viscera_weight', np.floa
t64(0.9009)), ('Diameter', 'Length', np.float64(0.9867)), ('Diameter', 'Diameter', n
p.float64(1.0)), ('Diameter', 'Whole_weight', np.float64(0.9238)), ('Diameter', 'She
ll_weight', np.float64(0.9024)), ('Height', 'Height', np.float64(1.0)), ('Whole_weig
ht', 'Length', np.float64(0.9237)), ('Whole_weight', 'Diameter', np.float64(0.923
8)), ('Whole_weight', 'Whole_weight', np.float64(1.0)), ('Whole_weight', 'Shucked_we
ight', np.float64(0.9696)), ('Whole_weight', 'Viscera_weight', np.float64(0.9662)),
('Whole_weight', 'Shell_weight', np.float64(0.9544)), ('Shucked_weight', 'Whole_weig
ht', np.float64(0.9696)), ('Shucked_weight', 'Shucked_weight', np.float64(1.0)), ('S
hucked_weight', 'Viscera_weight', np.float64(0.9336)), ('Viscera_weight', 'Length',
np.float64(0.9009)), ('Viscera_weight', 'Whole_weight', np.float64(0.9662)), ('Visce
ra_weight', 'Shucked_weight', np.float64(0.9336)), ('Viscera_weight', 'Viscera_weigh
t', np.float64(1.0)), ('Viscera_weight', 'Shell_weight', np.float64(0.9045)), ('Shel
l_weight', 'Diameter', np.float64(0.9024)), ('Shell_weight', 'Whole_weight', np.floa
t64(0.9544)), ('Shell_weight', 'Viscera_weight', np.float64(0.9045)), ('Shell_weigh
t', 'Shell_weight', np.float64(1.0)), ('Rings', 'Rings', np.float64(1.0))]
```

5. EDA

Summary

- Mostly numerical variables except sex.
- No missing values.
- Target (Rings) ranges from 1 to 29. Mostly normal, slight right skew.
- Sex needs to be one-hot encoded, the rest should be scaled.
- Numeric variables are moderately positively correlated with target.

In [18]:
```python
missing_values = train_df.isnull().sum()
print("Missing values per column:", missing_values[missing_values > 0])
```

```
Missing values per column: Series([], dtype: int64)
```

In [19]:
```python
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3341 entries, 2194 to 3988
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Sex             3341 non-null   object 
 1   Length          3341 non-null   float64
 2   Diameter        3341 non-null   float64
 3   Height          3341 non-null   float64
 4   Whole_weight    3341 non-null   float64
 5   Shucked_weight  3341 non-null   float64
 6   Viscera_weight  3341 non-null   float64
 7   Shell_weight    3341 non-null   float64
 8   Rings           3341 non-null   int64  
dtypes: float64(7), int64(1), object(1)
memory usage: 261.0+ KB
```

In [20]:
```python
train_df.describe().round(2)
```

Out[20]:

	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_w
count	3341.00	3341.00	3341.00	3341.00	3341.00	3341.00	33
mean	0.52	0.41	0.14	0.82	0.36	0.18	
std	0.12	0.10	0.04	0.49	0.22	0.11	
min	0.08	0.06	0.00	0.00	0.00	0.00	
25%	0.45	0.35	0.12	0.44	0.18	0.09	
50%	0.55	0.42	0.14	0.80	0.33	0.17	
75%	0.62	0.48	0.16	1.14	0.50	0.25	
max	0.82	0.65	0.52	2.83	1.49	0.76	

Visualisation

In [21]:
```python
from ydata_profiling import ProfileReport
ProfileReport(train_df)
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
100%|██████████| 9/9 [00:00<00:00, 1198.33it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

YData Profiling Report

Overview

Overview | Alerts 8 | Reproduction

Dataset statistics

Number of variables	9
Number of observations	3341
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	261.0 KiB
Average record size in memory	80.0 B

Variable types

Categorical	1
Numeric	8

Variables

Select Columns

Out[21]:

Below, we investigate the possible relationship between sex of adults (M/F), Infants, and number of rings, as the relationship may differ between those categories.

In [22]:
```
base = alt.Chart(train_df).mark_circle(opacity=0.3).encode(
    x='Shell_weight',
    y='Rings',
    color='Sex'
)

lines = base.transform_regression(
    'Shell_weight', 'Rings', groupby=['Sex']
).mark_line().encode(
    color='Sex'
)

(base + lines).properties(
    title="Rings vs Shell Weight by Sex (with Regression Lines)",
    width=500
)
```

Out[22]:

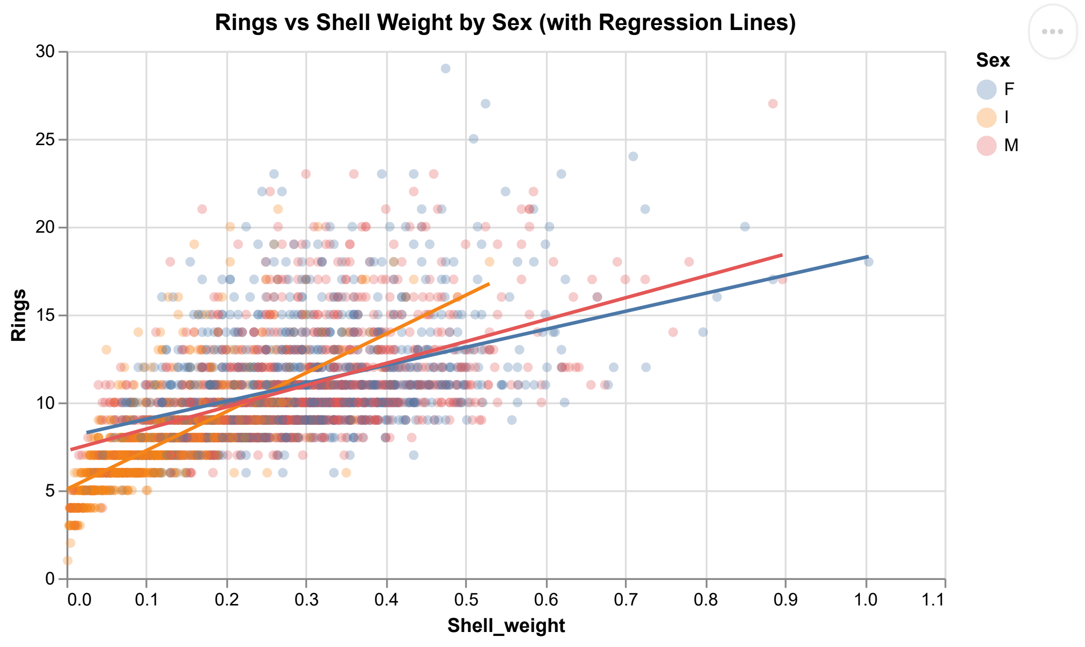


Figure 4. Number of rings (target variable) by shell weight (grams), grouped into Male (M), Female (F), and Infant (I). Linear regression lines for each group are shown.

Observation: The slopes appear slightly different, particularly for Infants (I) compared to adults (M/F). Infants seem to have a steeper growth curve in this dimension.

6. Modeling Interactions

We will use a Linear Regression model to predict the number of rings. We construct a pipeline that:

1. One-hot encodes the categorical Sex feature.
2. Scales the numerical features.
3. Applies Linear Regression.

In [23]:

```
# Define features
numeric_features = ['Length', 'Diameter', 'Height', 'Whole_weight', 'Shucked_weight
categorical_features = ['Sex']
```

```
# Create preprocessor
preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    (OneHotEncoder(drop='if_binary'), categorical_features)
)

# Create and fit pipeline
lr_model = make_pipeline(preprocessor, LinearRegression())
lr_model.fit(X_train, y_train)

# Make predictions
lr_y_pred = lr_model.predict(X_test)
```

7. Visual Evaluation of the model:

We evaluate the model by plotting the Predicted vs. Actual values.

In [24]:
```
# Calculate metrics
lr_rmse = np.sqrt(mean_squared_error(y_test, lr_y_pred))
lr_r2 = r2_score(y_test, lr_y_pred)

print(f"Root Mean Squared Error (RMSE): {lr_rmse:.4f}")
print(f"R-squared (R2): {lr_r2:.4f}")

# Visualization
results_df = pd.DataFrame({
    'Actual': y_test.flatten(),
    'Predicted': lr_y_pred.flatten()
})

pred_chart = alt.Chart(results_df).mark_circle(opacity=0.5).encode(
    x=alt.X('Actual', title='Actual Rings'),
    y=alt.Y('Predicted', title='Predicted Rings')
).properties(
    title=f'Actual vs Predicted Rings (R2 = {lr_r2:.2f})',
    width=500,
    height=500
)

line = alt.Chart(pd.DataFrame({'x': [0, 30], 'y': [0, 30]})).mark_line(color='red',
    x='x',
    y='y'
)

pred_chart + line
```

```
Root Mean Squared Error (RMSE): 2.3419
R-squared (R2): 0.4427
```

Out[24]:

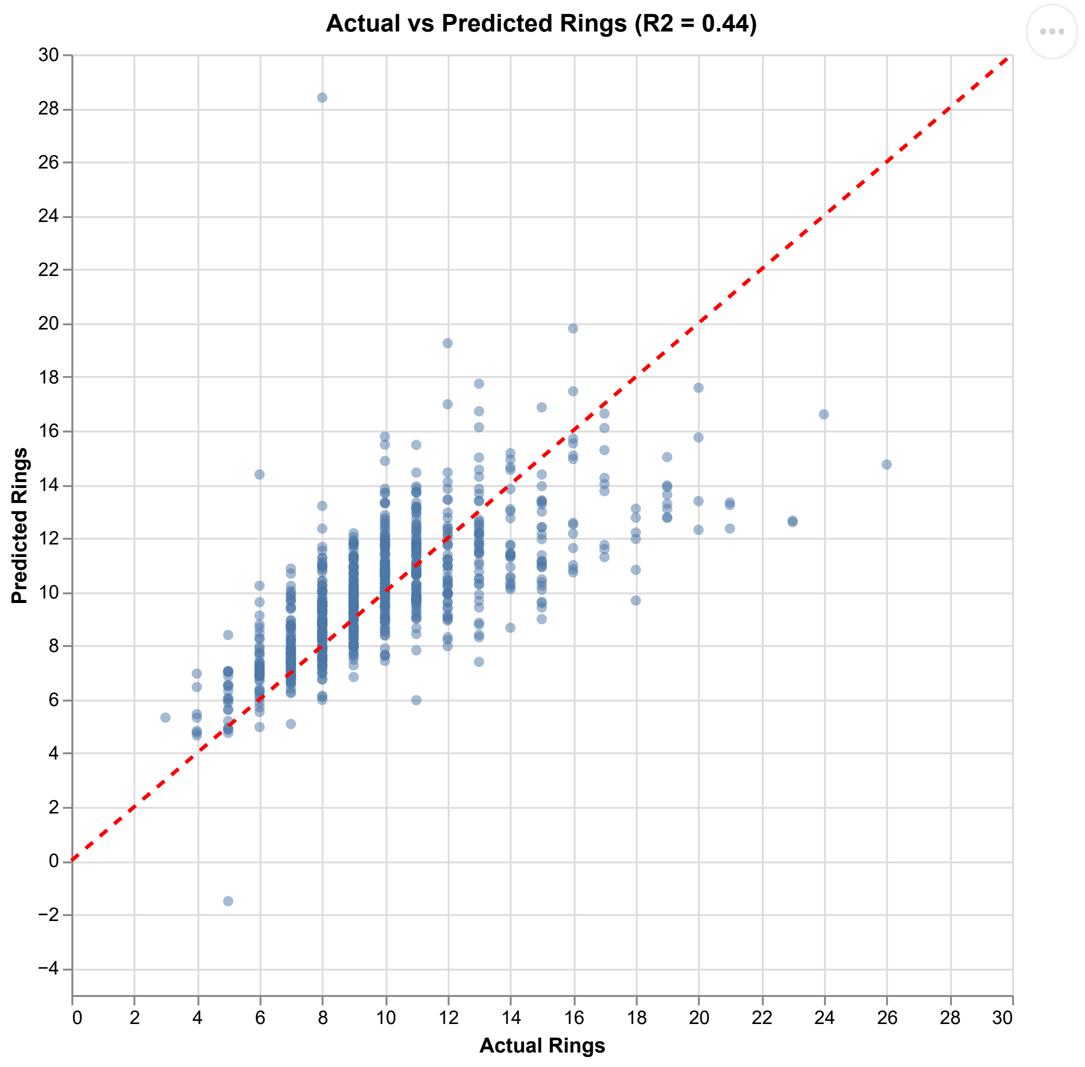


Figure 5: Actual Rings (x-axis) vs Predicted Rings (y-axis). The red dashed line represents the linear regression line. Points below the line indicate over-prediction, while points above indicate under-prediction.

In [25]:
```
# Extracting feature names from the preprocessor
ohe = lr_model.named_steps['columntransformer'].named_transformers_['onehotencoder'
ohe_features = list(ohe.get_feature_names_out(categorical_features))

# combining to get all features
all_features = numeric_features + ohe_features

# Extract coefficients from linear regression
coefficients = lr_model.named_steps['linearregression'].coef_

coef_df = pd.DataFrame({
    "Feature": all_features,
    "Coefficient": coefficients
}).sort_values("Coefficient", ascending=False)

coef_df
```

Out[25]:

	Feature	Coefficient
3	Whole_weight	4.483371
1	Diameter	1.153338
6	Shell_weight	1.020152
2	Height	0.808758
9	Sex_M	0.306958
7	Sex_F	0.208326
0	Length	-0.265891
8	Sex_I	-0.515284
5	Viscera_weight	-1.184490
4	Shucked_weight	-4.397524

Table 1. Linear Regression coefficient values showing the estimated contribution of each feature to the predicted target after preprocessing and one-hot encoding.

8. Analysis 2: Non-Linear Models

Model A: Random Forest Regressor

In [26]:
```
rf_preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features), # Scaling isn't strictly necessary for RF
    (OneHotEncoder(drop='if_binary'), categorical_features)
)

rf_model = make_pipeline(rf_preprocessor, RandomForestRegressor(n_estimators=100, r
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - RMSE: {rmse_rf:.4f}, R2: {r2_rf:.4f}")
```

```
Random Forest - RMSE: 2.1817, R2: 0.5163
```

Model B: Support Vector Regression (SVR)

In [27]:
```
svr_preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features), # Scaling is CRITICAL for SVR
    (OneHotEncoder(drop='if_binary'), categorical_features)
)

svr_model = make_pipeline(svr_preprocessor, SVR(kernel='rbf', C=1.0, epsilon=0.1))
svr_model.fit(X_train, y_train)
y_pred_svr = svr_model.predict(X_test)
```

```
rmse_svr = np.sqrt(mean_squared_error(y_test, y_pred_svr))
r2_svr = r2_score(y_test, y_pred_svr)

print(f"SVR (RBF Kernel) - RMSE: {rmse_svr:.4f}, R2: {r2_svr:.4f}")
```

```
SVR (RBF Kernel) - RMSE: 2.1673, R2: 0.5227
```

9. Comparison of Models

In [28]:
```
results = pd.DataFrame({
    'Model': ['Baseline (Linear)', 'Random Forest', 'SVR (RBF)'],
    'RMSE': [lr_rmse, rmse_rf, rmse_svr],
    'R2 Score': [lr_r2, r2_rf, r2_svr]
})

print(results.round(4))

# Visualize Comparison
base = alt.Chart(results).encode(x=alt.X('Model', sort='-y'))

bar_r2 = base.mark_bar().encode(
    y=alt.Y('R2 Score', title='R2 Score'),
    color=alt.Color('Model', legend=None)
).properties(title='Model Performance Comparison (R2)')

bar_rmse = (
    alt.Chart(results)
        .encode(
            x=alt.X('Model', sort='-y'),
            y=alt.Y('RMSE', title='RMSE'),
            color=alt.Color('Model', legend=None)
        )
        .mark_bar()
        .properties(title='Model Performance Comparison (RMSE)')
)

bar_rmse | bar_r2
```

```
               Model    RMSE  R2 Score
0  Baseline (Linear)  2.3419    0.4427
1      Random Forest  2.1817    0.5163
2          SVR (RBF)  2.1673    0.5227
```

Out[28]:

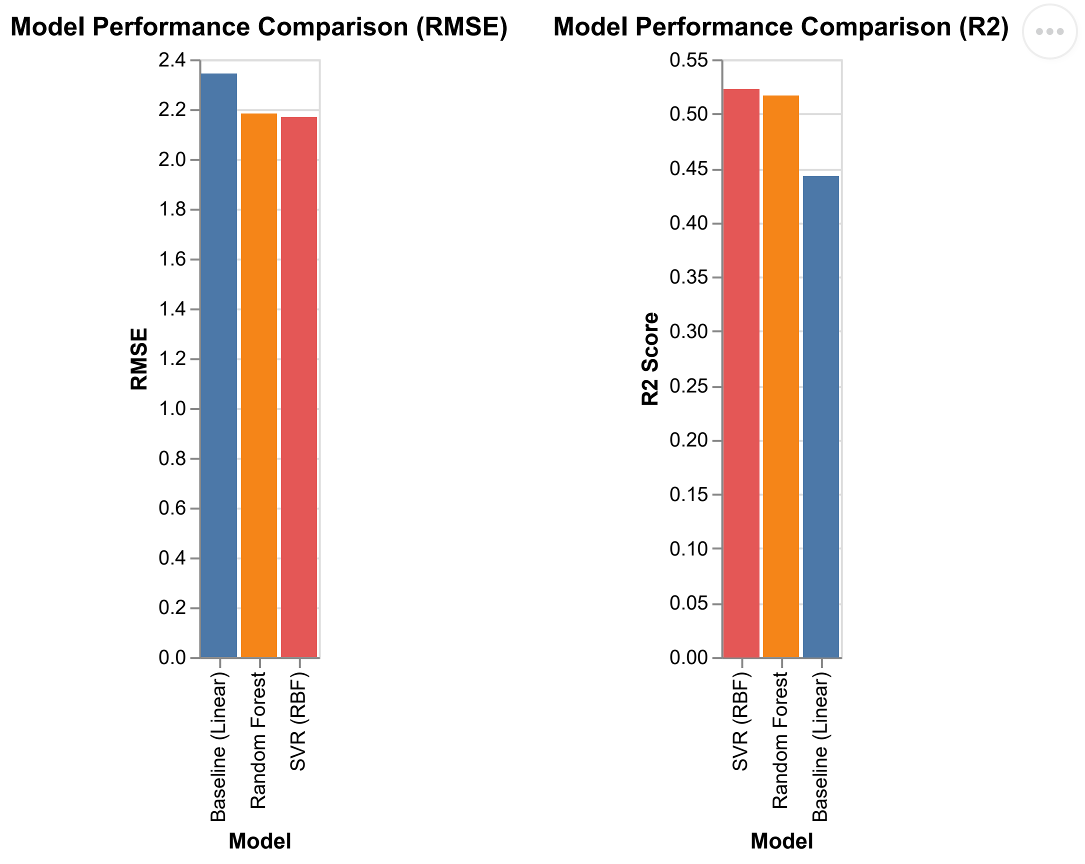


Figure 6. RMSE and R^2 of each model tested.

Discussion

In this report, we tested 3 different models to find which model was best at predicting number of rings (as a proxy for age) of Abalone molluscs. The baseline linear model explains about half of the variance in number of rings (R^2 = 0.44) using size and weight; errors are moderate and roughly symmetric, with a root mean squared error (RMSE) of approximately 5.48. The other two models tested were non-linear models. The Random Forest model generally achieves higher R^2 (0.52) and lower RMSE (4.76) than linear models, showing that abalone growth is not purely linear in the features. Finally, the support vector regression with an RBF kernel model is also an improvement over the baseline (RMSE of 4.70 and R^2 of 0.52), especially in mid-age ranges, but can be more sensitive to scaling and hyperparameters than Random Forest. The non-linear models (especially random forest) provided the best predictive performance, suggesting that future work should focus on flexible models and possibly add environmental features (e.g., location, temperature).

Although the linear model did not perform well relative to the non-linear models, extracting the coefficients for each feature reveals important patterns in our data. Whole weight had the strongest positive coefficient at (4.48), while shucked weight had the lowest coefficient (-4.40). These data indicate two things: that larger, heavier abalone (higher whole weight) had a strong positive relationship with age, and that there is likely collinearity in these features. We can assume possibility of collinearity based on the fact that shucked weight has a strong negative coefficient, despite the fact that, in context, shucked weight and whole weight should be highly associated. In the future, models which account for this collinearity, and potentially others, should be utilized. That whole weight is a strong positive predictor of age is expected - these organisms continue to grow larger with age throughout their lives [2].

Within the context of biological organisms, that a linear regression model did not perform well is unsurprising. Growth rate, which impacts features such as those in our dataset, can be highly variable depending on biotic and abiotic factors, and so it is likely that a linear regression model cannot account for enough of the variance present in the dataset. In a literature review by Guney et al. (2022), the authors review six different predictive models of Abalone age based on physical characteristics, and found that the BPFFNN model had the highest test accuracy, followed by a random forest model [3]. This agrees with our results on which model performed best.

Estimating Abalone age in a non-destructive way is important for growers and consumers of this mollusc. Therefore, a model which can predict age using easy-to-obtain physical measurements would be of practical importance. Our results suggest that the model should be non-linear, and that weight is likely to be an important predictor of age. However, there are still many improvements to be made on models predicting Abalone age, and questions to be answered. Future work could improve on models by gathering information on other physical characteristics, investigating collinearity in features, and testing other types of models.

References

[1] Dua, D., & Graff, C. (2019). UCI Machine Learning Repository: Abalone Data Set. University of California, Irvine, School of Information and Computer Science. Retrieved from the UCI Machine Learning Repository.

[2] Nash, W. J., Sellers, T. L., Talbot, S. R., Cawthorn, A. J., & Ford, W. B. (1994). The population biology of abalone (Haliotis species) in Tasmania. I. Blacklip abalone (H. rubra) from the north coast and islands of Bass Strait. Sea Fisheries Division Technical Report No. 48.

[3] Guney, S., Kilinc, I., Hameed, A.A., Jamil, A. (2022). Abalone Age Prediction Using Machine Learning. In: Djeddi, C., Siddiqi, I., Jamil, A., Ali Hameed, A., Kucuk, İ. (eds) Pattern Recognition and Artificial Intelligence. MedPRAI 2021. Communications in Computer and Information Science, vol 1543. Springer, Cham. https://doi.org/10.1007/978-3-031-04112-9_25

[4] Python Core Team. Python: A dynamic, open source programming language. Python Software Foundation, 2019. Python version 2.7. URL: https://www.python.org/.