Final-Project

January 31, 2025

Supervised Machine Learning: Regression - Final Assignment

0.1 Instructions:

In this Assignment, you will demonstrate the data regression skills you have learned by completing this course. You are expected to leverage a wide variety of tools, but also this report should focus on present findings, insights, and next steps. You may include some visuals from your code output, but this report is intended as a summary of your findings, not as a code review.

The grading will center around 5 main points:

1. Does the report include a section describing the data?
2. Does the report include a paragraph detailing the main objective(s) of this analysis?
3. Does the report include a section with variations of linear regression models and specifies which one is the model that best suits the main objective(s) of this analysis.
4. Does the report include a clear and well-presented section with key findings related to the main objective(s) of the analysis?
5. Does the report highlight possible flaws in the model and a plan of action to revisit this analysis with additional data or different predictive modeling techniques?

[62]:
```
# Use Jupyter Black for cell formatting
import jupyter_black

jupyter_black.load()
```

0.2 Import the required libraries

[63]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.stats import boxcox
from scipy.stats.mstats import normaltest

# Importing Libraries
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import (
```

```
    KFold,
    cross_val_predict,
    GridSearchCV,
    train_test_split,
    LeaveOneOut,
)
from sklearn.linear_model import LinearRegression, RidgeCV, Lasso, Ridge
from sklearn.linear_model import LassoCV, ElasticNetCV
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.pipeline import Pipeline
```

0.3 Importing the Dataset

Before you begin, you will need to choose a data set that you feel passionate about. You can brainstorm with your peers about great public data sets using the discussion board in this module.

Once you have selected a data set, you will produce the deliverables listed below and submit them to one of your peers for review. Treat this exercise as an opportunity to produce analysis that are ready to highlight your analytical skills for a senior audience, for example, the Chief Data Officer, or the Head of Analytics at your company. Sections required in your report:

- Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation.
- Brief description of the data set you chose and a summary of its attributes.
- Brief summary of data exploration and actions taken for data cleaning and feature engineering.
- Summary of training at least three linear regression models which should be variations that cover using a simple linear regression as a baseline, adding polynomial effects, and using a regularization regression. Preferably, all use the same training and test splits, or the same cross-validation method.
- A paragraph explaining which of your regressions you recommend as a final model that best fits your needs in terms of accuracy and explainability.
- Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your linear regression model.
- Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model adding specific data features to achieve a better explanation or a better prediction.

1 1. About the Data

1.1 Introduction

This dataset will consider Vinho Verde, a unique product from the Minho (north-west) region of Portugal. Medium in alcohol, is it particularly appreciated due to its freshness (specially in the summer). This wine accounts for 15% of the total Portuguese production, and around 10% is exported, mostly white wine.

Once viewed as a luxury good, nowadays wine is increasingly enjoyed by a wider range of consumers. Portugal is a top ten wine exporting country with 3.17% of the market share in 2005. Exports of its vinho verde wine (from the northwest region) have increased by 36% from 1997 to 2007. To support its growth, the wine industry is investing in new technologies for both wine making and selling processes. Wine certification and quality assessment are key elements within this context. Certification prevents the illegal adulteration of wines (to safeguard human health) and assures quality for the wine market. Quality evaluation is often part of the certification process and can be used to improve wine making (by identifying the most influential factors) and to stratify wines such as premium brands (useful for setting prices).

Wine certification is generally assessed by physicochemical and sensory tests. Physicochemical laboratory tests routinely used to characterize wine include determination of density, alcohol or pH values, while sensory tests rely mainly on human experts. It should be stressed that taste is the least understood of the human senses, thus wine classification is a difficult task. Moreover, the relationships between the physicochemical and sensory analysis are complex and still not fully understood.

Advances in information technologies have made it possible to collect, store and process massive, often highly complex datasets. All this data hold valuable information such as trends and patterns, which can be used to improve decision making and optimize chances of success.

Source

2 2. Main Objective

In this project the objective of the analysis will be focused on prediction of the quality of the wine.

2.1 The Data

The dataset is available here or here.

Variable Name	Role	Type	Description / Comment
fixed acidity	Feature	Continuous	Fixed acidity in wine is primarily due to the presence of stable acids such as tartaric, malic, citric, and succinic. These acids contribute to the overall taste and balance of the wine.
volatile acidity	Feature	Continuous	Often referred to as VA, volatile acidity is a measure of a wine's gaseous acids. The amount of VA in wine is often considered an indicator of spoilage.
citric acid	Feature	Continuous	Citric acid is often added to wines to increase acidity, complement a specific flavour or prevent ferric hazes.
residual sugar	Feature	Continuous	Residual Sugar (or RS) is from natural grape sugars leftover in a wine after the alcoholic fermentation finishes.
chlorides	Feature	Continuous	Chlorides in wine are salts of mineral acids that can affect the taste and quality of the wine.
free sulfur dioxide	Feature	Continuous	Free sulfur dioxide (SO2) is a preservative for wine. It has both antioxidant and antimicrobial properties, making it an effective preservative
total sulfur dioxide	Feature	Continuous	Total Sulfur Dioxide (TSO2) in wine is the portion of SO2 that is free in the wine plus the portion that is bound to other chemicals in the wine such as aldehydes, pigments, or sugars.

Variable Name	Role	Type	Description / Comment
density	Feature	Continuous	Wine density is determined by the amount of sugar, alcohol, and other solutes present in the wine. Generally, the higher the sugar and alcohol content, the higher the density of the wine.
pH	Feature	Continuous	The acidity of the wine. The pH of wine typically ranges from about 2.9 to 4.0. White wine usually has a pH level of 3.0 to 3.4, while red wine is between 3.3 to 3.6
sulphates	Feature	Continuous	Sulfates, or sulfur dioxide (SO2), are naturally occurring compounds that have been used in winemaking for centuries. They are a type of preservative that can help prevent oxidation and microbial spoilage in wine.
alcohol	Feature	Continuous	Wine alcohol content varies depending on the type of wine and the amount poured. A standard serving of wine is 5 ounces and generally contains between 11-13% alcohol by volume
colour	Other	Categorical	red or white
quality	Target	Integer	score between 0 and 10

2.2 Read the Data

[64]:
```
# Read in the red wine and white wine data and concatenate together
df_red = pd.read_csv("./data/winequality-red.csv", sep=";")
df_red["colour"] = "red"

df_white = pd.read_csv("./data/winequality-white.csv", sep=";")
df_white["colour"] = "white"

df = pd.concat([df_red, df_white])
```

[65]:
```
# The shape of the dataframe
data_shape = df.shape
print(f"The data has {data_shape[0]} rows of data and {data_shape[1]} columns.")
```

```
The data has 6497 rows of data and 13 columns.
```

[66]:
```
# The first 10 rows of data
df.head(10)
```

[66]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076
5            7.4              0.66         0.00             1.8      0.075
6            7.9              0.60         0.06             1.6      0.069
7            7.3              0.65         0.00             1.2      0.065
8            7.8              0.58         0.02             2.0      0.073
```

```
9           7.5             0.50         0.36              6.1      0.071

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56
5                 13.0                  40.0   0.9978  3.51       0.56
6                 15.0                  59.0   0.9964  3.30       0.46
7                 15.0                  21.0   0.9946  3.39       0.47
8                  9.0                  18.0   0.9968  3.36       0.57
9                 17.0                 102.0   0.9978  3.35       0.80

   alcohol  quality colour
0      9.4        5    red
1      9.8        5    red
2      9.8        5    red
3      9.8        6    red
4      9.4        5    red
5      9.4        5    red
6      9.4        5    red
7     10.0        7    red
8      9.5        7    red
9     10.5        5    red
```

[67]:
```
# The dtypes of each row
df.dtypes
```

[67]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
colour                   object
dtype: object
```

[68]:
```
# Describe the data
df.describe()
```

[68]:

	fixed acidity	volatile acidity	citric acid	residual sugar
count	6497.000000	6497.000000	6497.000000	6497.000000
mean	7.215307	0.339666	0.318633	5.443235
std	1.296434	0.164636	0.145318	4.757804
min	3.800000	0.080000	0.000000	0.600000
25%	6.400000	0.230000	0.250000	1.800000
50%	7.000000	0.290000	0.310000	3.000000
75%	7.700000	0.400000	0.390000	8.100000
max	15.900000	1.580000	1.660000	65.800000

	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	6497.000000	6497.000000	6497.000000	6497.000000
mean	0.056034	30.525319	115.744574	0.994697
std	0.035034	17.749400	56.521855	0.002999
min	0.009000	1.000000	6.000000	0.987110
25%	0.038000	17.000000	77.000000	0.992340
50%	0.047000	29.000000	118.000000	0.994890
75%	0.065000	41.000000	156.000000	0.996990
max	0.611000	289.000000	440.000000	1.038980

	pH	sulphates	alcohol	quality
count	6497.000000	6497.000000	6497.000000	6497.000000
mean	3.218501	0.531268	10.491801	5.818378
std	0.160787	0.148806	1.192712	0.873255
min	2.720000	0.220000	8.000000	3.000000
25%	3.110000	0.430000	9.500000	5.000000
50%	3.210000	0.510000	10.300000	6.000000
75%	3.320000	0.600000	11.300000	6.000000
max	4.010000	2.000000	14.900000	9.000000

Looking at the spread of the data we can see that:

- Chlorides range is 0.009 to 0.611
- Total Sulphur Dioxide range is 6.0 to 440.0

[69]:

```
# Verify that there are no null values
df.isnull().sum()
```

[69]:

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
```

```
alcohol                 0
quality                 0
colour                  0
dtype: int64
```

[70]:
```
# Check if there are duplicated values
df.duplicated().sum()
```

[70]:
```
np.int64(1177)
```

There are a significant number, 1177, of duplicate values in the data. There is no indication in the documentation with the data that there is actually any duplication / up-sampling of the data so, for the purposes of this project, it will be assumed that these duplicates are actual real data examples that just happen to have the same values. Given the repeatable wine making techniques, processes and raw materials this is highly probable.

2.3 Skewness and kurtosis

2.3.1 Skewness:

Skewness is a statistical term and it is a way to estimate or measure the shape of a distribution. It is an important statistical methodology that is used to estimate the asymmetrical behavior rather than computing frequency distribution. Skewness can be two types:

1. Symmetrical: A distribution can be called symmetric if it appears the same from the left and right from the center point.
2. Asymmetrical: A distribution can be called asymmetric if it doesn't appear the same from the left and right from the center point.

Distribution on the basis of skewness value:

- Skewness $= 0$: Then normally distributed.
- Skewness > 0: Then more weight in the left tail of the distribution.
- Skewness < 0: Then more weight in the right tail of the distribution.

2.3.2 Kurtosis:

Is also a statistical term and an important characteristic of frequency distribution. It determines whether a distribution is heavy-tailed in respect of the normal distribution. It provides information about the shape of a frequency distribution.

- Kurtosis for normal distribution is equal to 3.
- For a distribution having kurtosis < 3: It is called playkurtic.
- For a distribution having kurtosis > 3, It is called leptokurtic and it signifies that it tries to produce more outliers rather than the normal distribution.

[71]:
```
# Get the Skewness and kurtosis of the numeric data
df_num = df.select_dtypes(include="number")
df_num_columns = df_num.columns
df_num_skew = df_num.skew()
df_num_kurt = df_num.kurtosis()
```

```
df_num_summary = pd.DataFrame(
    zip(df_num_columns, df_num_skew, df_num_kurt),
    columns=["Column", "Skew", "Kurtosis"],
)
```

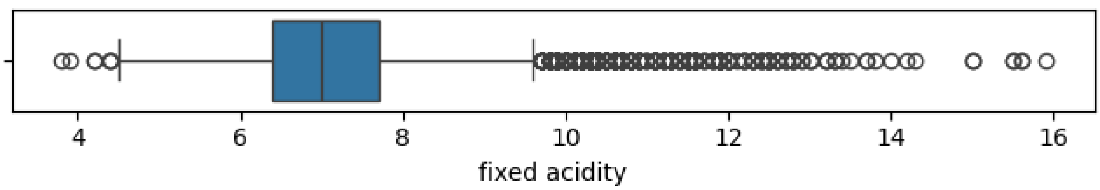
2.3.3 Analysis

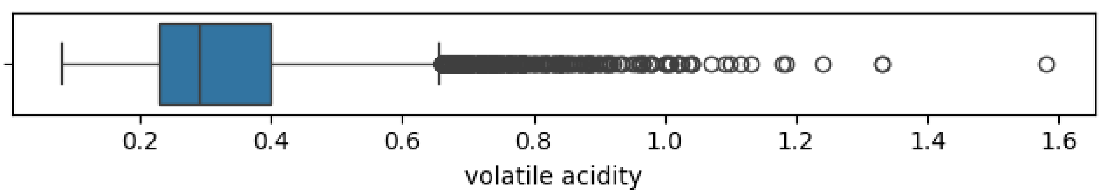
From the table above we can see that most measures are either positively or negatively skewed. Total Sulfur Dioxide appears to be the most normal distribution in restect to skew. Volitile Acidity is the attribute with the a Kurtosis value that appears to be most normal.

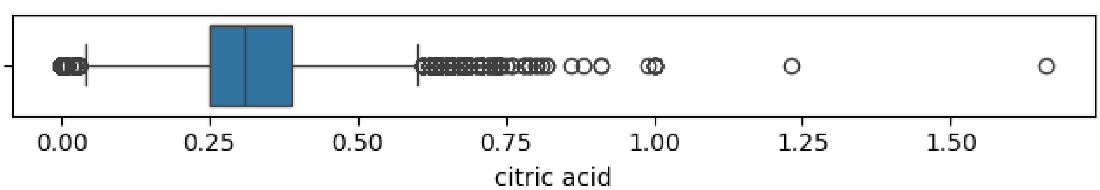
2.4 Outliers

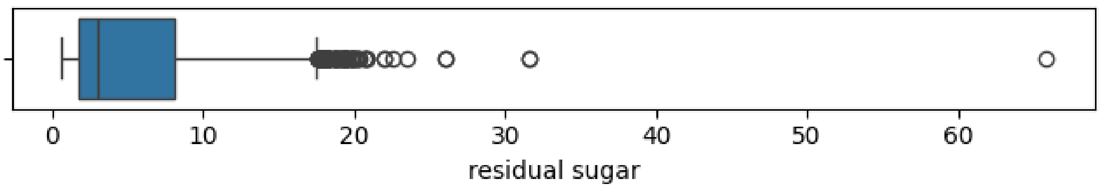
[72]:
```
# Box Plots
plt.figure(dpi=300)

for column in df_num:
    plt.figure(figsize=(8, 0.75))
    sns.boxplot(data=df_num, x=column)
```

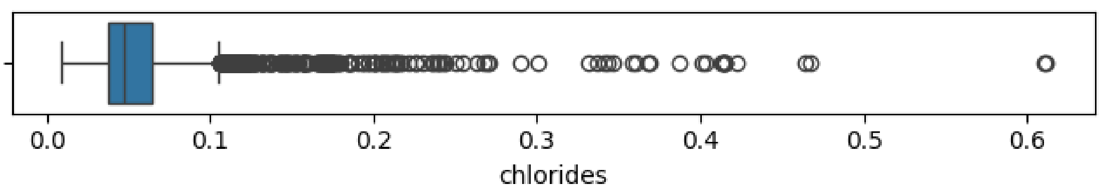
```
<Figure size 1920x1440 with 0 Axes>
```

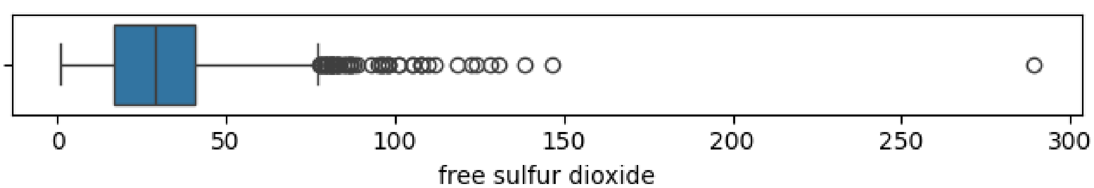


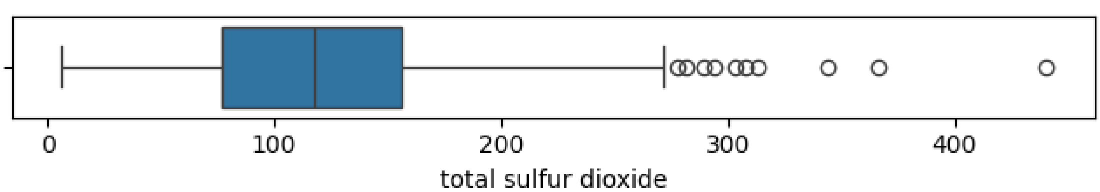


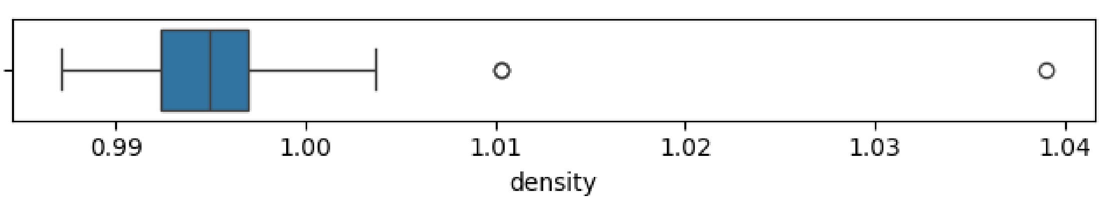


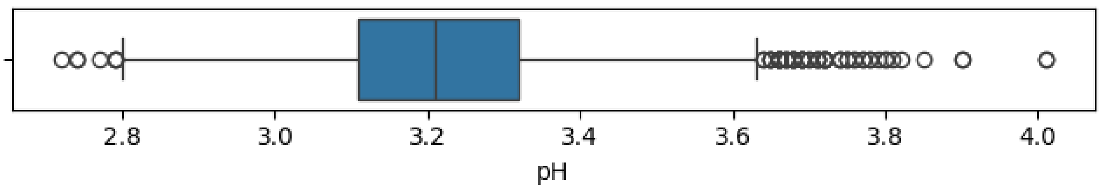

0 10 20 30 40 50 60
residual sugar

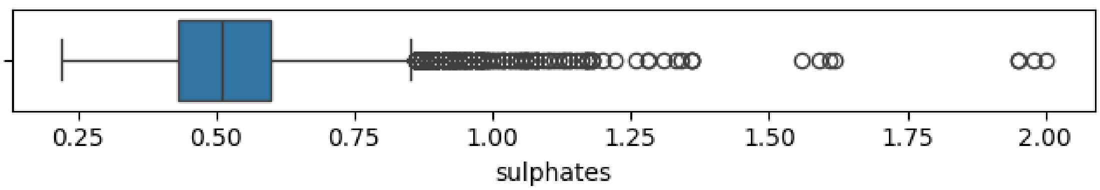

0.0 0.1 0.2 0.3 0.4 0.5 0.6
chlorides

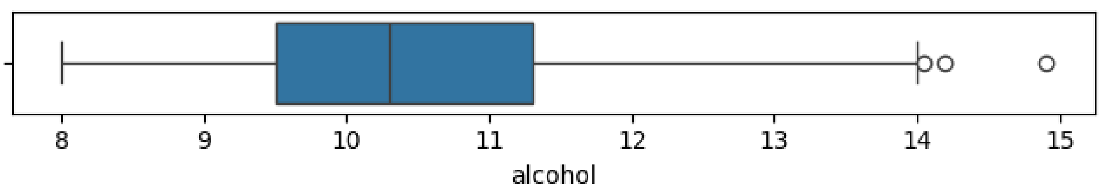

0 50 100 150 200 250 300
free sulfur dioxide

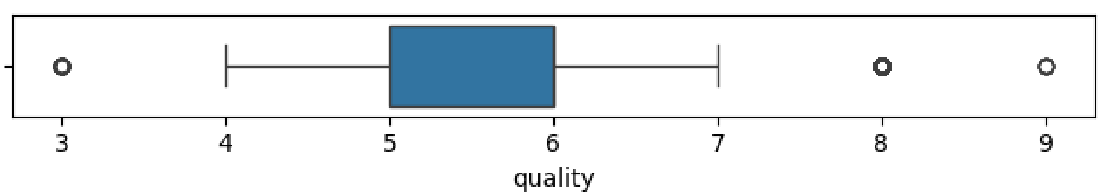

0 100 200 300 400
total sulfur dioxide


0.99 1.00 1.01 1.02 1.03 1.04
density









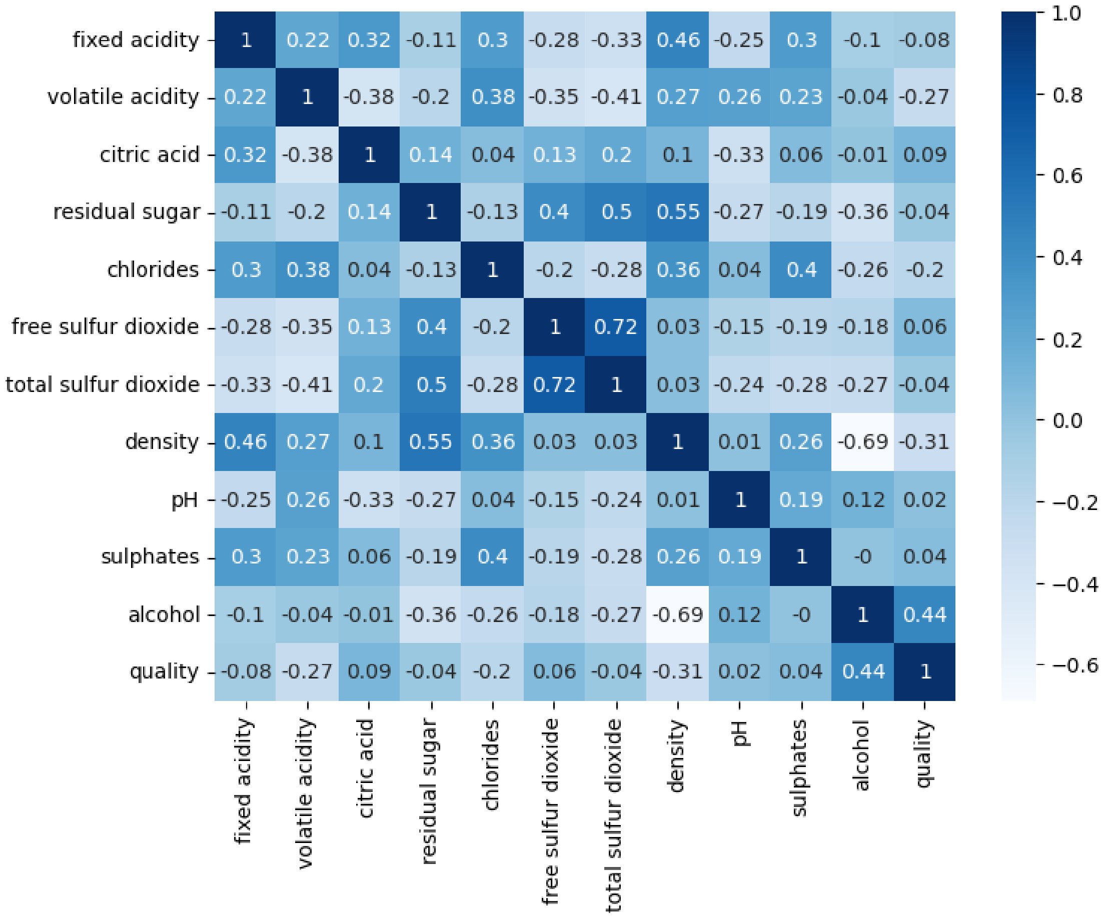
2.4.1 Analysis

Most measures have a significant number of outliers apart from **Alcohol**.

2.5 Correlation

[73]:
```
# Studying the corellations between features using Heat Map!
plt.figure(dpi=300)
plt.figure(figsize=(8, 6))
sns.heatmap(np.round(df_num.corr(), 2), annot=True, cmap="Blues")
plt.show()
```

<Figure size 1920x1440 with 0 Axes>



[74]:
```python
# Sorting features according to the strength of corretlation with Quality␣
↪feature
df_num.corr()["quality"].sort_values(ascending=False)
```

[74]:
```
quality                 1.000000
alcohol                 0.444319
citric acid             0.085532
free sulfur dioxide     0.055463
sulphates               0.038485
pH                      0.019506
residual sugar         -0.036980
total sulfur dioxide   -0.041385
fixed acidity          -0.076743
chlorides              -0.200666
volatile acidity       -0.265699
density                -0.305858
```

```
Name: quality, dtype: float64
```

2.5.1 Analysis

Alcohol is has the strongest positive correlation and **density** has the highest negative correlation.

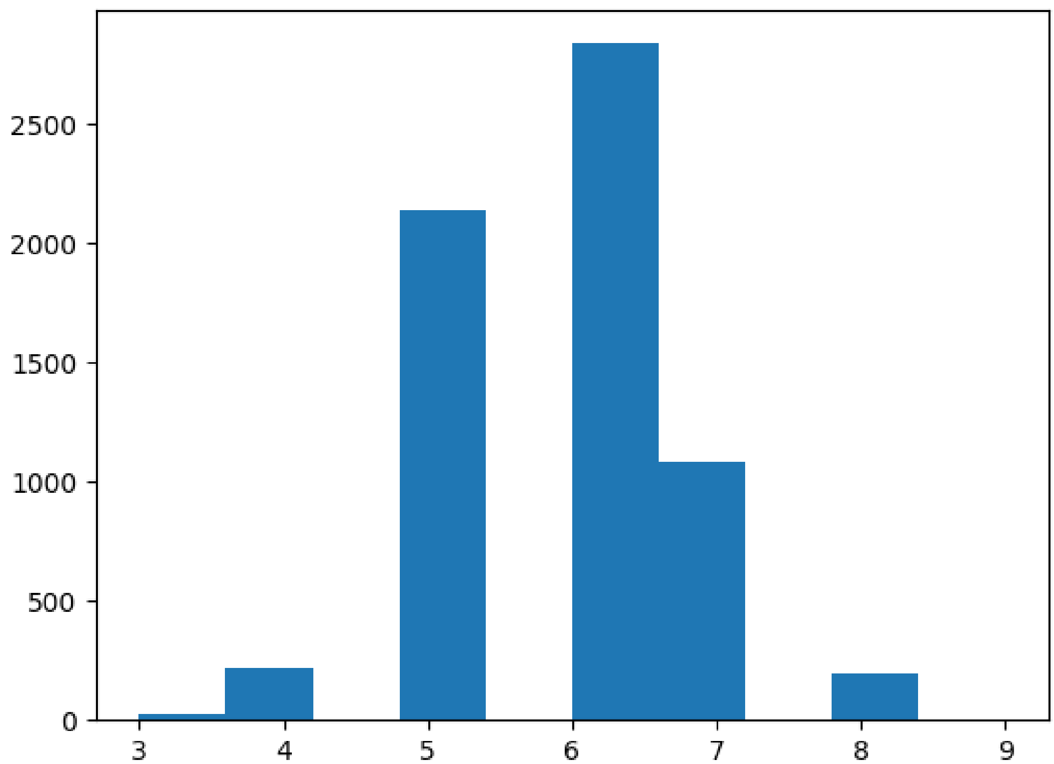
2.6 Determining Normality

A target variable that is normally distributed will often lead to better linear regression model results. If our target is not normally distributed, we can apply a transformation to it and then fit our regression to predict the transformed values.

How can we tell if our target is normally distributed? There are two ways:

1. Plotting the Histogram
2. Applying D'Agostino K^2 test to check the normality

[75]:
```
# Plot the histogram
plt.hist(df["quality"]);
```



[76]:
```
# applying D'Agostino K^2 test to check the normality!
norm = normaltest(df["quality"].values)
norm
```

[76]: NormaltestResult(statistic=np.float64(50.358972216153944),
pvalue=np.float64(1.1606148581928246e-11))

2.6.1 Analysis

The p-value is quite far away of 0.05 which indicates absense of normality! Linear Regression assumes a normally distributed residuals which can be aided by transforming the y variable (Our target).

Transformations techniques to get or approach normal distribution: 1. Square Root 2. Log Transformation 3. Box cox

[77]:
```
# Applying the transformations
sqrt_quality = np.sqrt(df["quality"])
sqrt_test_res = normaltest(sqrt_quality.values)

log_quality = np.log(df["quality"])
log_test_res = normaltest(log_quality.values)

bc_quality = boxcox(df["quality"])
boxcox_medv, lam = bc_quality
boxcox_test_res = normaltest(boxcox_medv)
```

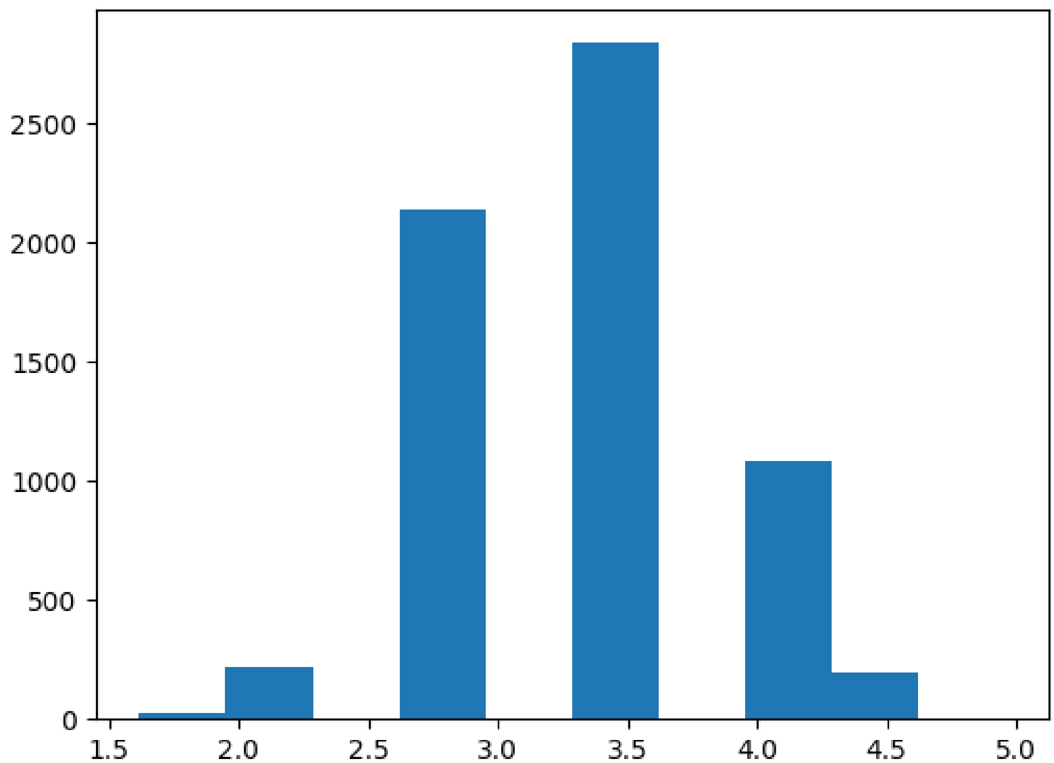
[78]:
```
# View the results
df_results = pd.DataFrame(
    {
        "Transormation": ["Original", "Square-Root", "Log", "Box Cox"],
        "P-value": [norm[1], sqrt_test_res[1], log_test_res[1],␣
 ↪boxcox_test_res[1]],
    }
)
df_results
```

[78]:

	Transormation	P-value
0	Original	1.160615e-11
1	Square-Root	3.710862e-09
2	Log	2.704157e-58
3	Box Cox	2.331304e-05

Box Cox appears to give the best result so we will use this transformation in our models.

[79]:
```
plt.hist(boxcox_medv);
```



3 3. Linear Regression Models

[80]:
```
# Helper function
def performance_evaluation(y_true, y_pred, model, data):
    metric_name = ["R2", "MSE", "MAE"]
    metric_value = [
        r2_score(y_true, y_pred),
        mean_squared_error(y_true, y_pred),
        mean_absolute_error(y_true, y_pred),
    ]
    df = pd.DataFrame(
        zip(metric_name, metric_value), columns=["metric_name", "metric_value"]
    )
    df["model"] = model
    df["data"] = data
    return df
```

[81]:
```
# Prepare the data
# Read it clean again# Data Retrieving
df_red = pd.read_csv("./data/winequality-red.csv", sep=";")
df_red["colour"] = "red"
```

```
df_white = pd.read_csv("./data/winequality-white.csv", sep=";")
df_white["colour"] = "white"

df = pd.concat([df_red, df_white])

df.head()
```

[81]:

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality colour
0      9.4        5    red
1      9.8        5    red
2      9.8        5    red
3      9.8        6    red
4      9.4        5    red
```

[82]:

```
# Select the object (string) columns
categorical_cols = df.select_dtypes(include=["object"]).columns.tolist()

le = LabelEncoder()
for category in categorical_cols:
    le.fit(df[category].drop_duplicates())
    df[category] = le.transform(df[category])

# Normalise Target
df["quality"] = boxcox(df["quality"])[0]
```

[83]:

```
# Split the data
X = df.drop("quality", axis=1)
y = df["quality"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

```
# Create folds
kf = KFold(shuffle=True, random_state=42, n_splits=3)
```

3.0.1 Vanilla Linear Regression

[84]:
```
estimator = Pipeline(
    [
        ("scaler", StandardScaler()),
        ("polynomial_features", PolynomialFeatures()),
        ("linear_regression", LinearRegression()),
    ]
)

params = {
    "polynomial_features__degree": range(3),
}

grid = GridSearchCV(estimator, params, cv=kf)
grid.fit(X_train, y_train)
grid.best_score_, grid.best_params_
```

[84]:
```
(np.float64(0.29433355642599235), {'polynomial_features__degree': 1})
```

[85]:
```
# Evaluating best model on the Training Data
lr_pipeline = Pipeline(
    [
        ("polynomial_features", PolynomialFeatures(include_bias=False,␣
 ↪degree=1)),
        ("linear_regression", LinearRegression()),
    ]
)
lr_pipeline.fit(X_train, y_train)
```

[85]:
```
Pipeline(steps=[('polynomial_features',
                 PolynomialFeatures(degree=1, include_bias=False)),
                ('linear_regression', LinearRegression())])
```

[86]:
```
# A table to hold all results
df_results = pd.DataFrame()

# Training Data Results
y_train_hat = lr_pipeline.predict(X_train)

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
```

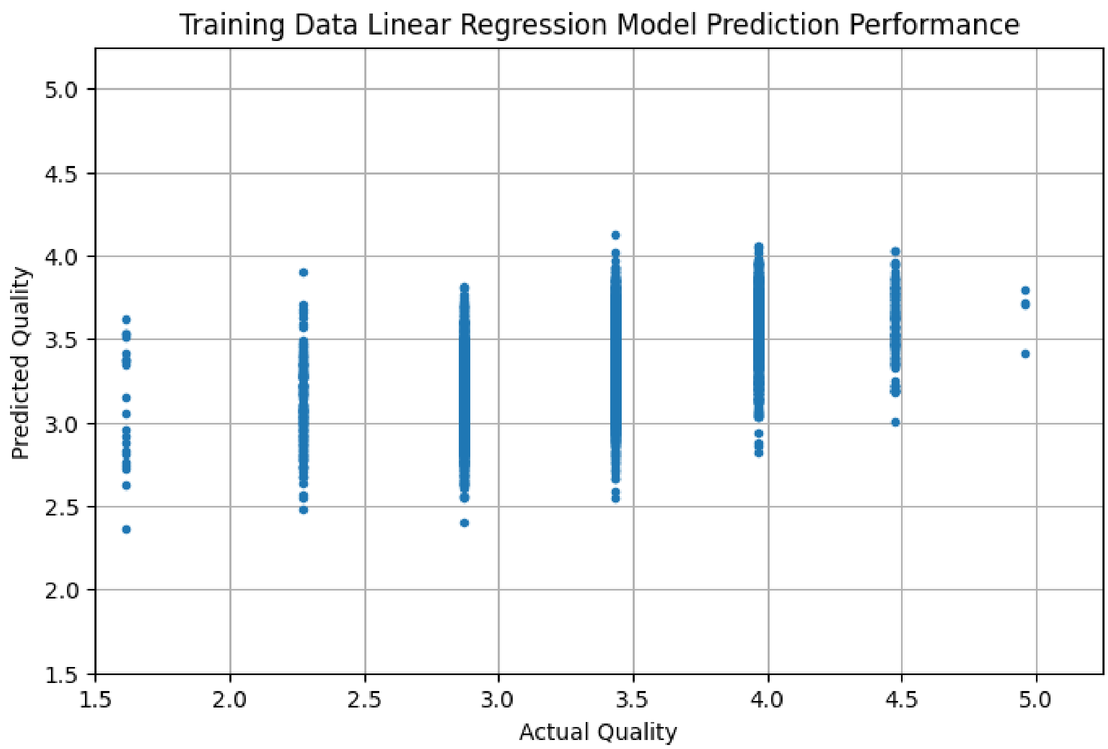
```
            y_train, y_train_hat, "Linear Regression", "Training Data"
        ),
    ]
)
df_results[
    (df_results["model"] == "Linear Regression")
    & (df_results["data"] == "Training Data")
]
```

[86]:

	metric_name	metric_value	model	data
0	R2	0.302106	Linear Regression	Training Data
1	MSE	0.164953	Linear Regression	Training Data
2	MAE	0.314530	Linear Regression	Training Data

[87]:

```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_train, y_train_hat, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Training Data Linear Regression Model Prediction Performance",
);
```



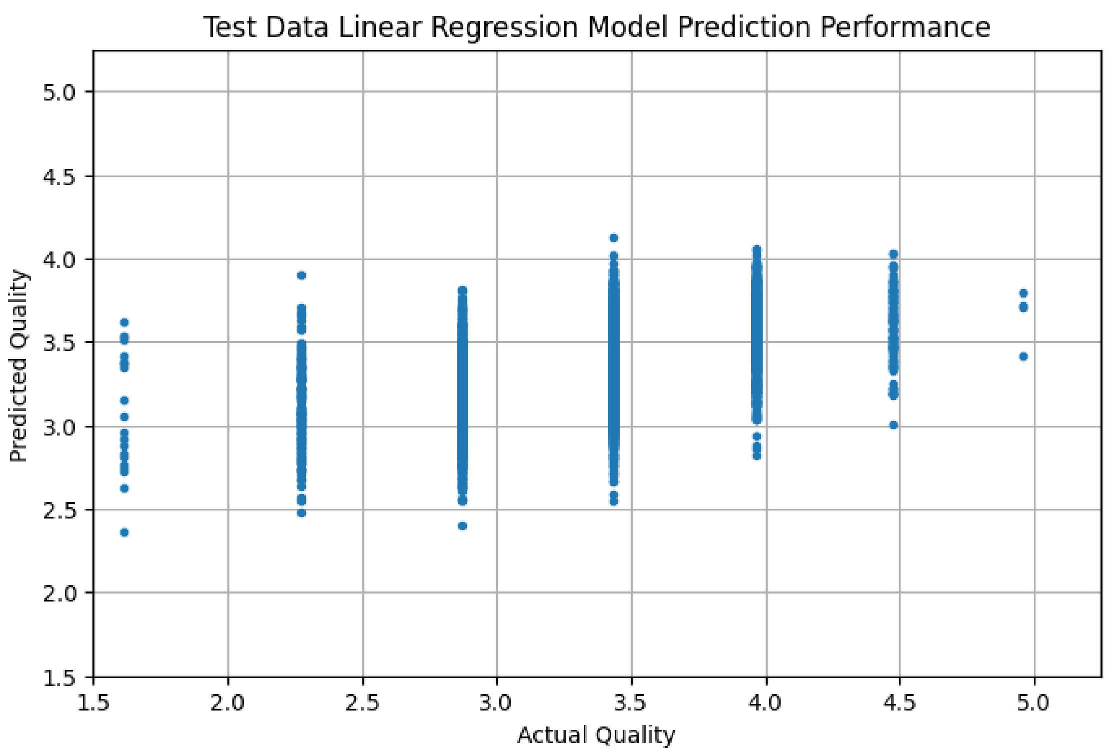
[88]:
```
# Test Data Results
y_test_hat = lr_pipeline.predict(X_test)

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(y_test, y_test_hat, "Linear Regression", "Test␣
 ↪Data"),
    ]
)
df_results[
    (df_results["model"] == "Linear Regression") & (df_results["data"] == "Test␣
 ↪Data")
]
```

[88]:

	metric_name	metric_value	model	data
0	R2	0.273143	Linear Regression	Test Data
1	MSE	0.162492	Linear Regression	Test Data
2	MAE	0.310617	Linear Regression	Test Data

[89]:
```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
```

```
plt.grid(True)
axes.plot(y_train, y_train_hat, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Test Data Linear Regression Model Prediction Performance",
);
```



3.0.2 Ridge Regression

Ridge Regression is a regression technique aimed at preventing overfitting in linear regression models when the model is too complex and fits the training data very closely, but performs poorly on new and unseen data. The algorithm adds a penalty term (referred to as the L2 regularization) to the linear regression cost function that is proportional to the square of the magnitude of the coefficients. This approach helps to reduce the magnitude of the coefficients in the model, which in turn can prevent overfitting and is especially helpful where there are many correlated predictor variables in the model. A hyperparameter alpha serving as a constant that multiplies the L2 term thereby controlling regularization strength needs to be optimized through cross-validation.

[90]:
```python
# Defining the hyperparameters for the ridge regression model
alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Defining a pipeline for the  ridge regression model
ridge_regression_pipeline = Pipeline(
    [
        ("polynomial_features", PolynomialFeatures(include_bias=False,␣
↪degree=1)),
        ("ridge_regression", RidgeCV(alphas=alphas, cv=None,␣
↪store_cv_results=True)),
    ]
)

# Fitting the ridge regression model
ridge_regression_pipeline.fit(X_train, y_train)
```

[90]:
```
Pipeline(steps=[('polynomial_features',
                 PolynomialFeatures(degree=1, include_bias=False)),
                ('ridge_regression',
                 RidgeCV(alphas=[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                                 1000],
                         store_cv_results=True))])
```

[91]:
```python
# Determining the optimal alpha
ridge_regression_pipeline["ridge_regression"].alpha_
```

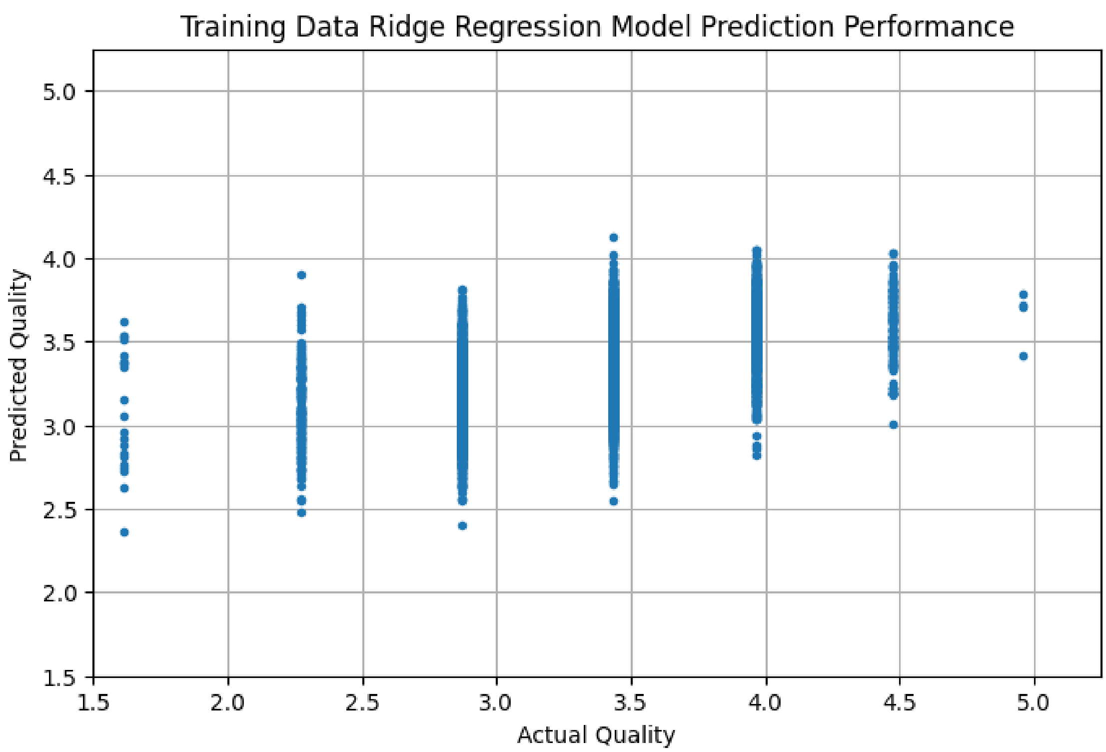
[91]:
```
np.float64(0.0001)
```

[92]:
```python
# Evaluating model on the Training Data
y_train_hat_ridge = ridge_regression_pipeline.predict(X_train)

performance_evaluation(y_train, y_train_hat_ridge, "Ridge Regression",␣
↪"Training Data")

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
            y_train, y_train_hat_ridge, "Ridge Regression", "Training Data"
        ),
    ]
)
df_results[
    (df_results["model"] == "Ridge Regression")
    & (df_results["data"] == "Training Data")
]
```

[92]:

	metric_name	metric_value	model	data
0	R2	0.302093	Ridge Regression	Training Data
1	MSE	0.164956	Ridge Regression	Training Data
2	MAE	0.314525	Ridge Regression	Training Data

[93]:
```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_train, y_train_hat_ridge, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Training Data Ridge Regression Model Prediction Performance",
);
```



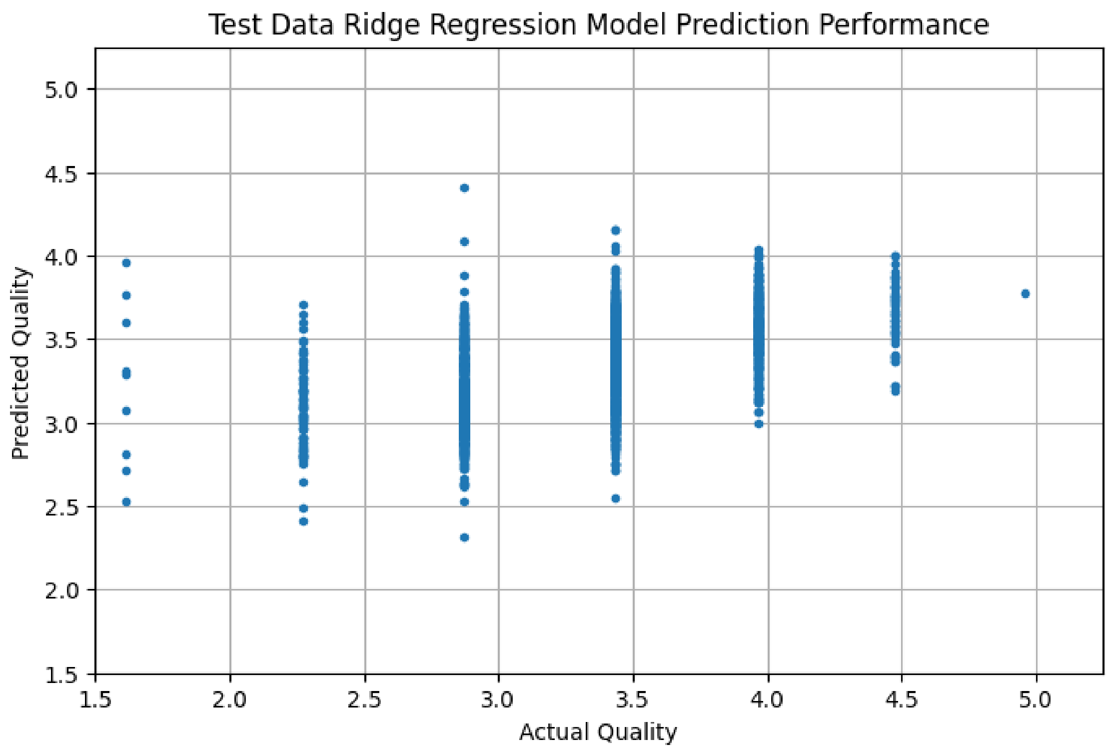
[94]:
```
# Evaluating model on the Testing Data
y_test_hat_ridge = ridge_regression_pipeline.predict(X_test)
```

```
performance_evaluation(y_test, y_test_hat_ridge, "Ridge Regression", "Test␣
↪Data")

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
            y_test, y_test_hat_ridge, "Ridge Regression", "Test Data"
        ),
    ]
)
df_results[
    (df_results["model"] == "Ridge Regression") & (df_results["data"] == "Test␣
↪Data")
]
```

[94]:

	metric_name	metric_value	model	data
0	R2	0.273128	Ridge Regression	Test Data
1	MSE	0.162495	Ridge Regression	Test Data
2	MAE	0.310633	Ridge Regression	Test Data

[95]:
```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_test, y_test_hat_ridge, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Test Data Ridge Regression Model Prediction Performance",
);
```



3.0.3 Lasso Regression

Least Absolute Shrinkage and Selection Operator (LASSO) Regression is a regression technique aimed at preventing overfitting in linear regression models when the model is too complex and fits the training data very closely, but performs poorly on new and unseen data. The algorithm adds a penalty term (referred to as the L1 regularization) to the linear regression cost function that is proportional to the absolute value of the coefficients. This approach can be useful for feature selection, as it tends to shrink the coefficients of less important predictor variables to zero, which can help simplify the model and improve its interpretability. A hyperparameter alpha serving as a constant that multiplies the L1 term thereby controlling regularization strength needs to be optimized through cross-validation.

[96]:
```
# Defining the hyperparameters for the ridge regression model
alphas_lasso = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

# Defining a pipeline for the  lasso regression model
lasso_regression_pipeline = Pipeline(
    [
        ("polynomial_features", PolynomialFeatures(include_bias=False,␣
 ↪degree=1)),
        ("lasso_regression", LassoCV(alphas=alphas, cv=LeaveOneOut())),
    ]
)
```

```
# Fitting the ridge regression model
lasso_regression_pipeline.fit(X_train, y_train)
```

[96]:
```
Pipeline(steps=[('polynomial_features',
                 PolynomialFeatures(degree=1, include_bias=False)),
                ('lasso_regression',
                 LassoCV(alphas=[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                                 1000],
                         cv=LeaveOneOut()))])
```

[97]:
```
# Determining the optimal alpha
lasso_regression_pipeline["lasso_regression"].alpha_
```

[97]:
```
np.float64(1e-05)
```

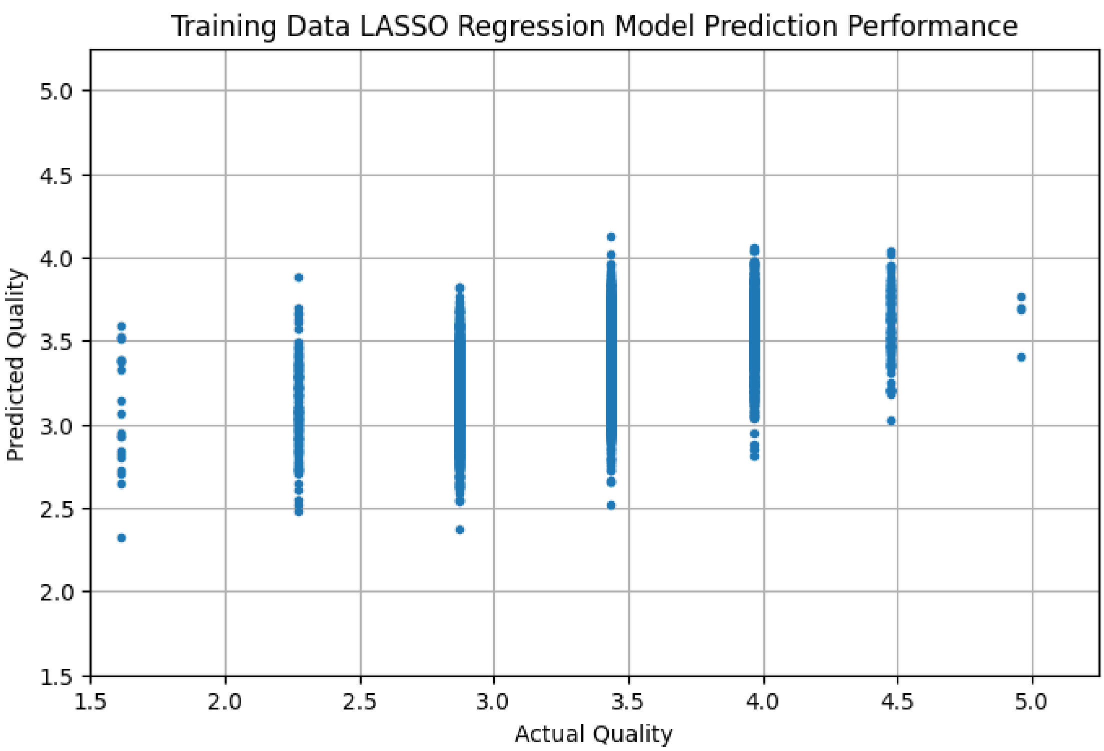
[98]:
```
# Evaluating model on the Training Data
y_train_hat_lasso = lasso_regression_pipeline.predict(X_train)

performance_evaluation(y_train, y_train_hat_lasso, "LASSO Regression",␣
 ↪"Training Data")

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
            y_train, y_train_hat_lasso, "LASSO Regression", "Training Data"
        ),
    ]
)
df_results[
    (df_results["model"] == "LASSO Regression")
    & (df_results["data"] == "Training Data")
]
```

[98]:

	metric_name	metric_value	model	data
0	R2	0.301092	LASSO Regression	Training Data
1	MSE	0.165193	LASSO Regression	Training Data
2	MAE	0.314665	LASSO Regression	Training Data

[99]:
```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_train, y_train_hat_lasso, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
```

```
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Training Data LASSO Regression Model Prediction Performance",
);
```



[100]:

```
# Evaluating model on the Testing Data
y_test_hat_lasso = lasso_regression_pipeline.predict(X_test)

performance_evaluation(y_test, y_test_hat_lasso, "LASSO Regression", "Test␣
↪Data")

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
            y_test, y_test_hat_lasso, "LASSO Regression", "Test Data"
        ),
    ]
)
df_results[
```

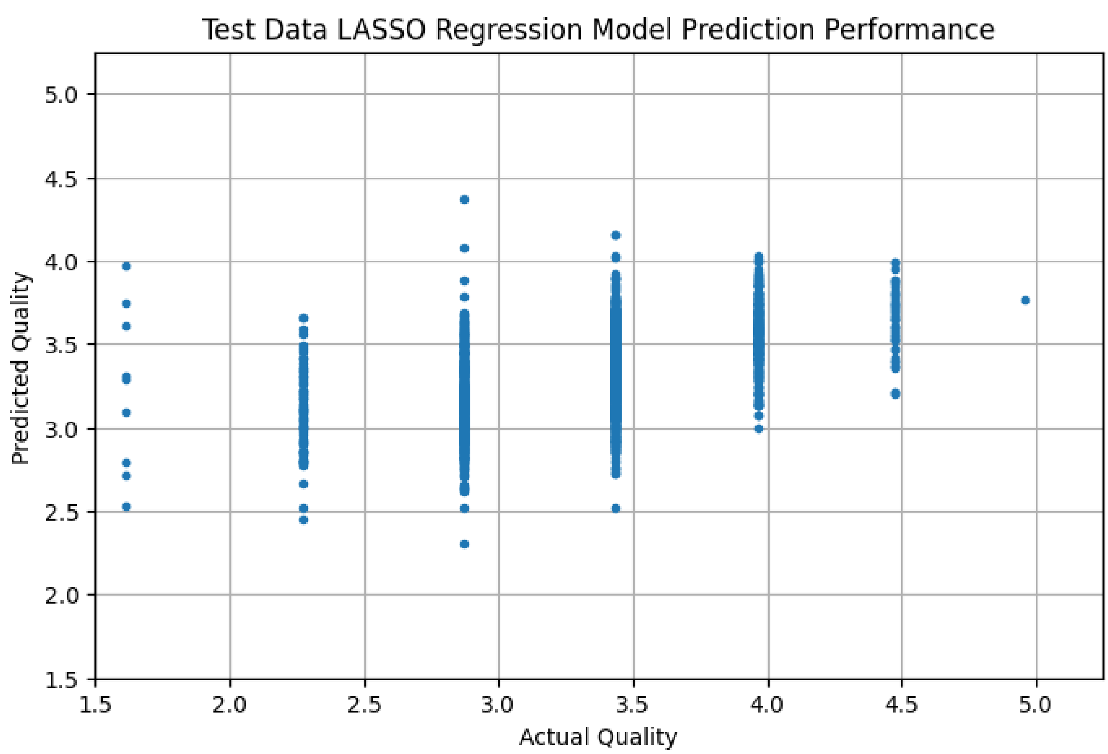
```
    (df_results["model"] == "LASSO Regression") & (df_results["data"] == "Test␣
↪Data")
]
```

[100]:

	metric_name	metric_value	model	data
0	R2	0.272135	LASSO Regression	Test Data
1	MSE	0.162717	LASSO Regression	Test Data
2	MAE	0.310927	LASSO Regression	Test Data

[101]:

```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_test, y_test_hat_lasso, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Test Data LASSO Regression Model Prediction Performance",
);
```



3.0.4 Elastic Net Regression

Elastic Net Regression is a regression technique aimed at preventing overfitting in linear regression models when the model is too complex and fits the training data very closely, but performs poorly on new and unseen data. The algorithm is a combination of the ridge and LASSO regression methods, by adding both L1 and L2 regularization terms in the cost function. This approach can be useful when there are many predictor variables that are correlated with the response variable, but only a subset of them are truly important for predicting the response. The L1 regularization term can help to select the important variables, while the L2 regularization term can help to reduce the magnitude of the coefficients. Hyperparameters alpha which serves as the constant that multiplies the penalty terms, and l1_ratio that serves as the mixing parameter that penalizes as a combination of L1 and L2 regularization - need to be optimized through cross-validation.

[102]:
```python
# Defining the hyperparameters for the elastic-net regression model
l1_ratios = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
alphas_en = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Defining a pipeline for the elastic-net regression model
elasticnet_regression_pipeline = Pipeline(
    [
        ("polynomial_features", PolynomialFeatures(include_bias=False,␣
 ↪degree=1)),
        (
            "elasticnet_regression",
            ElasticNetCV(alphas=alphas_en, l1_ratio=l1_ratios,␣
 ↪cv=LeaveOneOut()),
        ),
    ]
)

# Fitting an elastic-net regression model
elasticnet_regression_pipeline.fit(X_train, y_train)
```

[102]:
```
Pipeline(steps=[('polynomial_features',
                 PolynomialFeatures(degree=1, include_bias=False)),
                ('elasticnet_regression',
                 ElasticNetCV(alphas=[0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                                      1000],
                              cv=LeaveOneOut(),
                              l1_ratio=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
                                        0.9]))])
```

[103]:
```python
# Determining the optimal alpha
elasticnet_regression_pipeline["elasticnet_regression"].alpha_
```

[103]:
```
np.float64(0.0001)
```

[104]:
```
# Determining the optimal l1_ratio
elasticnet_regression_pipeline["elasticnet_regression"].l1_ratio_
```

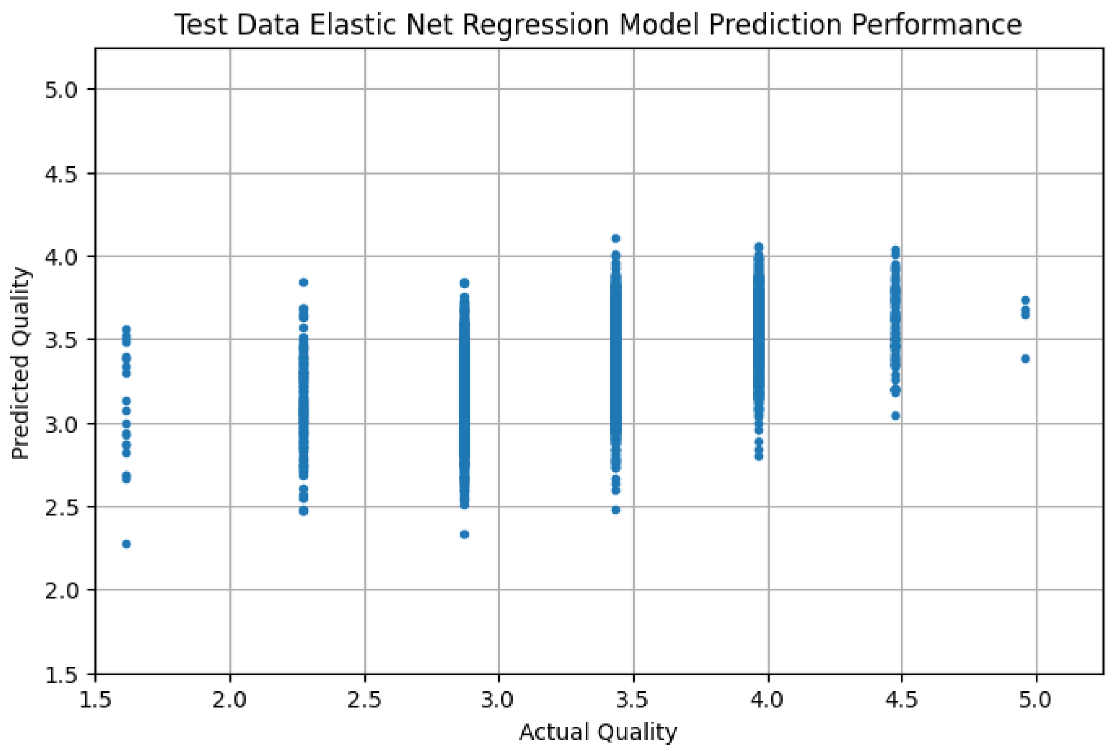
[104]: np.float64(0.1)

[105]:
```
# Evaluating model on the Training Data
y_train_hat_en = elasticnet_regression_pipeline.predict(X_train)

performance_evaluation(
    y_train, y_train_hat_en, "Elastic Net Regression", "Training Data"
)

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
            y_train, y_train_hat_en, "Elastic Net Regression", "Training Data"
        ),
    ]
)
df_results[
    (df_results["model"] == "Elastic Net Regression")
    & (df_results["data"] == "Training Data")
]
```

[105]:

	metric_name	metric_value	model	data
0	R2	0.296640	Elastic Net Regression	Training Data
1	MSE	0.166245	Elastic Net Regression	Training Data
2	MAE	0.315592	Elastic Net Regression	Training Data

[106]:
```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_train, y_train_hat_en, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Test Data Elastic Net Regression Model Prediction Performance",
);
```



[107]:

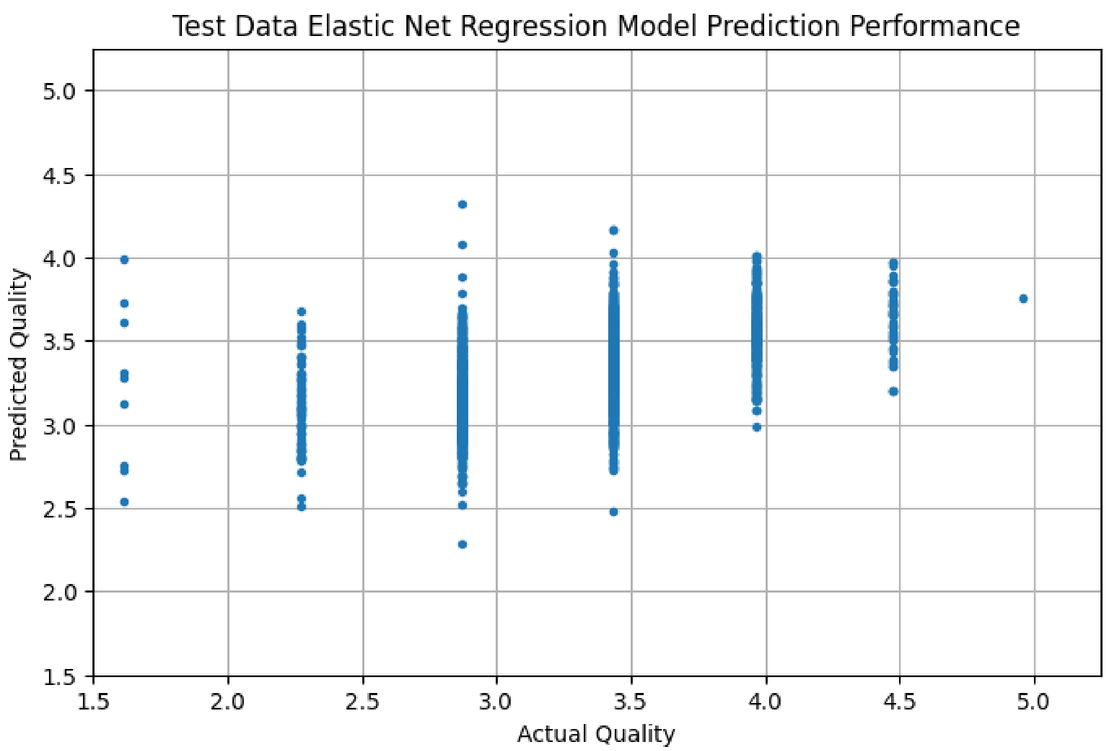
```
# Evaluating model on the Testing Data
y_test_hat_en = elasticnet_regression_pipeline.predict(X_test)

performance_evaluation(y_test, y_test_hat_en, "Elastic Net Regression", "Test␣
↪Data")

df_results = pd.concat(
    [
        df_results,
        performance_evaluation(
            y_test, y_test_hat_en, "Elastic Net Regression", "Test Data"
        ),
    ]
)
df_results[
    (df_results["model"] == "Elastic Net Regression")
    & (df_results["data"] == "Test Data")
]
```

[107]:

	metric_name	metric_value	model	data
0	R2	0.267927	Elastic Net Regression	Test Data
1	MSE	0.163658	Elastic Net Regression	Test Data
2	MAE	0.311906	Elastic Net Regression	Test Data

[108]:

```
# Plot actual vs predicted
figure = plt.figure(figsize=(8, 5))
axes = plt.axes()
plt.grid(True)
axes.plot(y_test, y_test_hat_en, marker="o", ls="", ms=3.0)
lim = (1.5, 5.25)
axes.set(
    xlabel="Actual Quality",
    ylabel="Predicted Quality",
    xlim=lim,
    ylim=lim,
    title="Test Data Elastic Net Regression Model Prediction Performance",
);
```



4 4. Insights and key findings

The linear regression model was the worst performing test model among the candidate models.

- $R^2 = 0.273143$
- Mean Squared Error = 0.162492
- Mean Absolute Error = 0.310617

Among the penalized models, the optimal elastic net regression model demonstrated the best independent test model performance with the tuned hyperparameter leaning towards a lower L1 regularization effect of 0.1.

- $R^2 = 0.267927$
- Mean Squared Error = 0.163658
- Mean Absolute Error = 0.311906

The optimal lasso regression model using an L1 regularization term demonstrated a high independent test model performance.

- $R^2 = 0.272135$
- Mean Squared Error = 0.162717
- Mean Absolute Error = 0.310927

The optimal ridge regression model using an L2 regularization term equally demonstrated good independent test model performance.

- $R^2 = 0.273128$
- Mean Squared Error = 0.162495
- Mean Absolute Error = 0.310633

In all instance the models generalised well and had a slightly lower R^2 value on the Test Data.

The computed r-squared metrics for the formulated models were all very close - only ranging from 0.267 to 0.273, which could be further improved by: * Considering more informative predictors * Considering more complex models other than linear regression and its variants

[109]:
```
df_results[df_results["data"] == "Test Data"]
```

[109]:

	metric_name	metric_value	model	data
0	R2	0.273143	Linear Regression	Test Data
1	MSE	0.162492	Linear Regression	Test Data
2	MAE	0.310617	Linear Regression	Test Data
0	R2	0.273128	Ridge Regression	Test Data
1	MSE	0.162495	Ridge Regression	Test Data
2	MAE	0.310633	Ridge Regression	Test Data
0	R2	0.272135	LASSO Regression	Test Data
1	MSE	0.162717	LASSO Regression	Test Data
2	MAE	0.310927	LASSO Regression	Test Data
0	R2	0.267927	Elastic Net Regression	Test Data
1	MSE	0.163658	Elastic Net Regression	Test Data
2	MAE	0.311906	Elastic Net Regression	Test Data

[110]:
```
df_results[df_results["metric_name"] == "R2"]
```

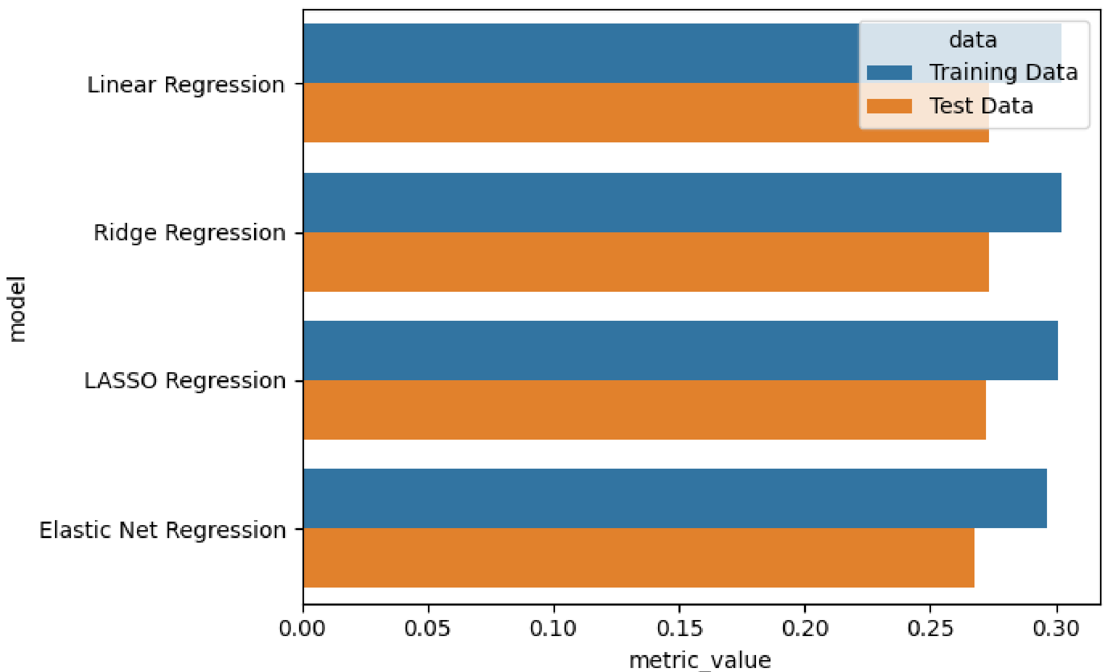
[110]:

	metric_name	metric_value	model	data
0	R2	0.302106	Linear Regression	Training Data
0	R2	0.273143	Linear Regression	Test Data
0	R2	0.302093	Ridge Regression	Training Data
0	R2	0.273128	Ridge Regression	Test Data
0	R2	0.301092	LASSO Regression	Training Data

```
0          R2      0.272135        LASSO Regression      Test Data
0          R2      0.296640  Elastic Net Regression  Training Data
0          R2      0.267927  Elastic Net Regression      Test Data
```

[111]:
```
sns.barplot(
    x="metric_value",
    y="model",
    hue="data",
    orient="h",
    data=df_results[df_results["metric_name"] == "R2"],
)
```

[111]: <Axes: xlabel='metric_value', ylabel='model'>



5 5. Next Steps

The next steps would be to look at more sophisticated models e.g. SVM or to look at this challlenge as a classification problem i.e. could we predict the colour of the wine.

##