Wine Quality Linear Regression Analysis

Executive Summary

This technical report presents a comprehensive analysis of wine quality factors for Vinhos Verdes, a Portuguese wine company accounting for approximately 15% of all wine sold in Portugal. Through statistical modeling and exploratory data analysis, we investigate how various chemical and physical wine attributes affect overall quality ratings. Our predictive model, evaluated using Mean Square Error (MSE), provides actionable insights for optimizing wine production parameters with respect to quality. Upon completion of this report we found that, ***We advise creating red wine with 4.0 g/L fixed acidity, 1.23 g/L citric acid, 0.6 g/L residual sugar, 0.01 g/L chlorides, 140 mg/L free sulfur dioxide, and 2.0 g/L sulphates. This will give a wine quality between 6.47 and 9.8.***

Technical Report

We were tasked by Vinhos Verdes, a Portugal wine company that accounts for about 15% of all wine sold in Portugal, to investigate the way different attributions inside of the wine affect the wine's quality. Our model will be evaluated using a quadratic loss function and more specifically Mean Squared Error (MSE).

Exploratory Data Analysis

We started by importing necessary Python libraries to facilitate data analysis, manipulation, and visualization.

```
#importing require modules for our data analysis
import os
import numpy as np
import polars as pl
import pandas as pd
import seaborn.objects as so
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import seaborn as sns

#Load in our dataset
wdata = pd.read_csv('Project1_data.csv')
wdata.head().T
```

	0	1	2	3	4
fixed acidity	7.4000	7.8000	7.800	11.200	7.4000
volatile acidity	0.7000	0.8800	0.760	0.280	0.7000
citric acid	0.0000	0.0000	0.040	0.560	0.0000
residual sugar	1.9000	2.6000	2.300	1.900	1.9000
chlorides	0.0760	0.0980	0.092	0.075	0.0760
free sulfur dioxide	11.0000	25.0000	15.000	17.000	11.0000
total sulfur dioxide	34.0000	67.0000	54.000	60.000	34.0000
density	0.9978	0.9968	0.997	0.998	0.9978
pH	3.5100	3.2000	3.260	3.160	3.5100
sulphates	0.5600	0.6800	0.650	0.580	0.5600
alcohol	9.4000	9.8000	9.800	9.800	9.4000
quality	5.0000	5.0000	5.000	6.000	5.0000
is_red	1.0000	1.0000	1.000	1.000	1.0000

The provided data has several variables all that are involved in the process of creating a high quality wine. The data was created for the purpose of analyzing wine quality and has the variables for an exploratory analysis on wine quality. Vinhos Verdes gave us data they collected from 2004-2007 on different wine quality. This data was collected by the official certification entity (CVRVV), and measured via a computerized system (iLab). The data can be found at this link:

https://archive.ics.uci.edu/dataset/186/wine+quality.

The company and more information about this data can be found in these links:

https://www.vinhoverde.pt/pt/ https://www.sciencedirect.com/science/article/abs/pii/S0167923609001377?via%3Dihub

The dataset features 13 variables representing wine samples and quality ratings.Each row summarizes a wine's chemical and physical profile.

Here, we checked variable types and missing values to help evaluate analysis viability and modeling challenges.

```
#Understand variable types and check for missing values
display(wdata.info())
display(wdata.describe().T)
display(wdata.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  is_red                6497 non-null   int64  
dtypes: float64(11), int64(2)
memory usage: 660.0 KB
```

```
None
```

	count	mean	std	min	25%	50%	75%	max
fixed acidity	6497.0	7.215307	1.296434	3.80000	6.40000	7.00000	7.70000	15.90000
volatile acidity	6497.0	0.339666	0.164636	0.08000	0.23000	0.29000	0.40000	1.58000
citric acid	6497.0	0.318633	0.145318	0.00000	0.25000	0.31000	0.39000	1.66000
residual sugar	6497.0	5.443235	4.757804	0.60000	1.80000	3.00000	8.10000	65.80000
chlorides	6497.0	0.056034	0.035034	0.00900	0.03800	0.04700	0.06500	0.61100
free sulfur dioxide	6497.0	30.525319	17.749400	1.00000	17.00000	29.00000	41.00000	289.00000
total sulfur dioxide	6497.0	115.74457	56.521855	6.00000	77.00000	118.00000	156.00000	440.00000
density	6497.0	0.994697	0.002999	0.98711	0.99234	0.99489	0.99699	1.03898

	count	mean	std	min	25%	50%	75%	max
pH	6497.0	3.218501	0.160787	2.72000	3.11000	3.21000	3.32000	4.01000
sulphates	6497.0	0.531268	0.148806	0.22000	0.43000	0.51000	0.60000	2.00000
alcohol	6497.0	10.491801	1.192712	8.00000	9.50000	10.30000	11.30000	14.90000
quality	6497.0	5.818378	0.873255	3.00000	5.00000	6.00000	6.00000	9.00000
is_red	6497.0	0.246114	0.430779	0.00000	0.00000	0.00000	0.00000	1.00000

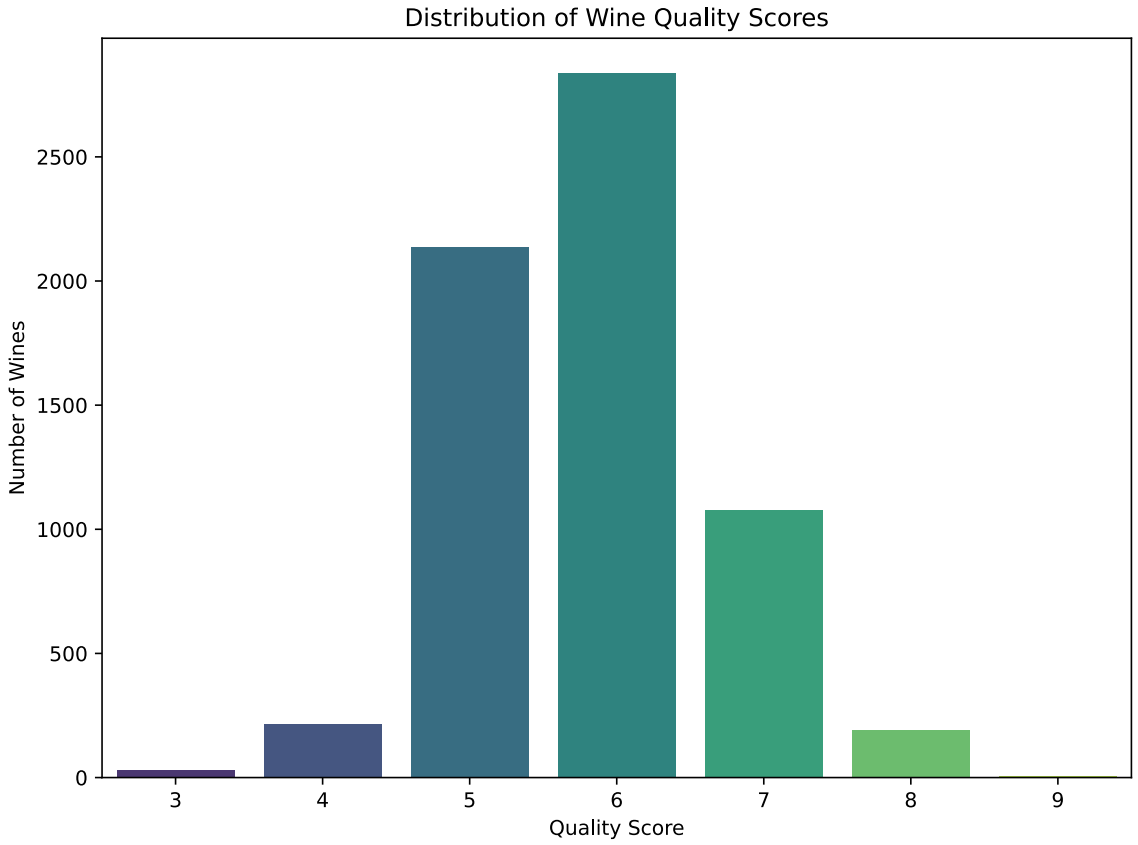
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
is_red                  0
dtype: int64
```

Below, we made some graphs to better understand the data:

```
# Create a subtable with quality levels and number of rows
quality_counts = wdata['quality'].value_counts().sort_index()

# Creates a bar chart that plots the total for each quality level
plt.figure(figsize=(8, 6))
sns.barplot(x=quality_counts.index, y=quality_counts.values,
palette='viridis')
plt.title('Distribution of Wine Quality Scores')
plt.xlabel('Quality Score')
plt.ylabel('Number of Wines')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
C:\Users\A02332124\AppData\Local\Temp\ipykernel_24000\2882911673.py:6:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=quality_counts.index, y=quality_counts.values,
palette='viridis')
```

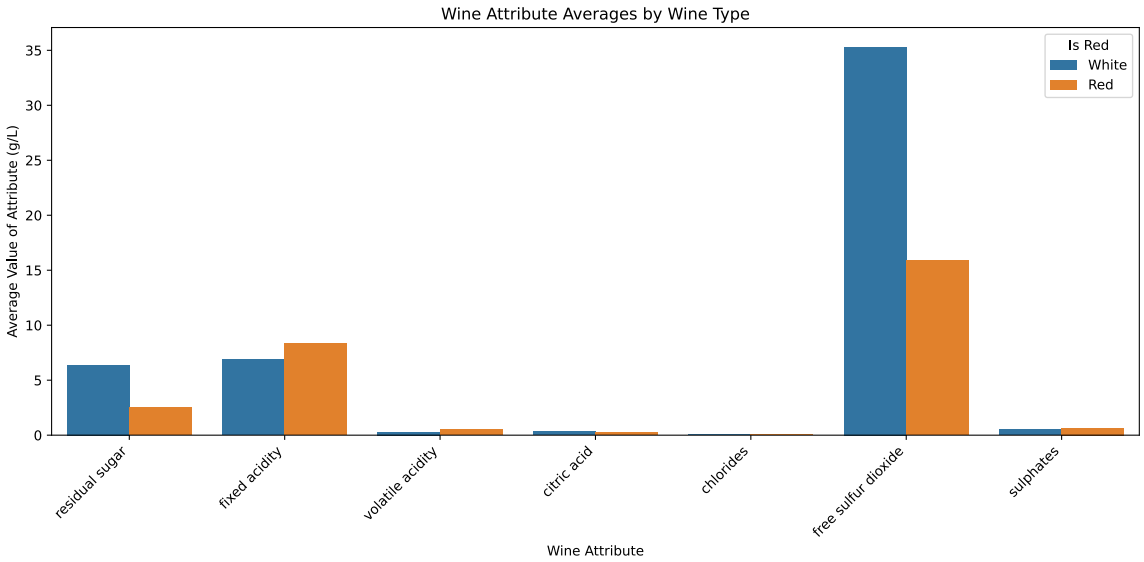


This bar chart shows the distribution of our y-variable, quality. We can see it has a roughly normal distribution, with the majority of quality scores being around 5 and 6. We also see that there are no scores for 1, 2, or 10, and very few scores for 3 and 9. Since we are wanting to create a wine in the 9-10 range, we will have to use the trends we see in lower values (particularly levels 5-8) to make our prediction.

Next, we compare red and white wines on key chemical attributes to inform modeling.

```
# Calculate averages for each ingredient grouped by is_red
mean_data = wdata.groupby('is_red')[[
    'residual sugar', 'fixed acidity', 'volatile acidity',
    'citric acid', 'chlorides',
    'free sulfur dioxide',
    'sulphates'
```

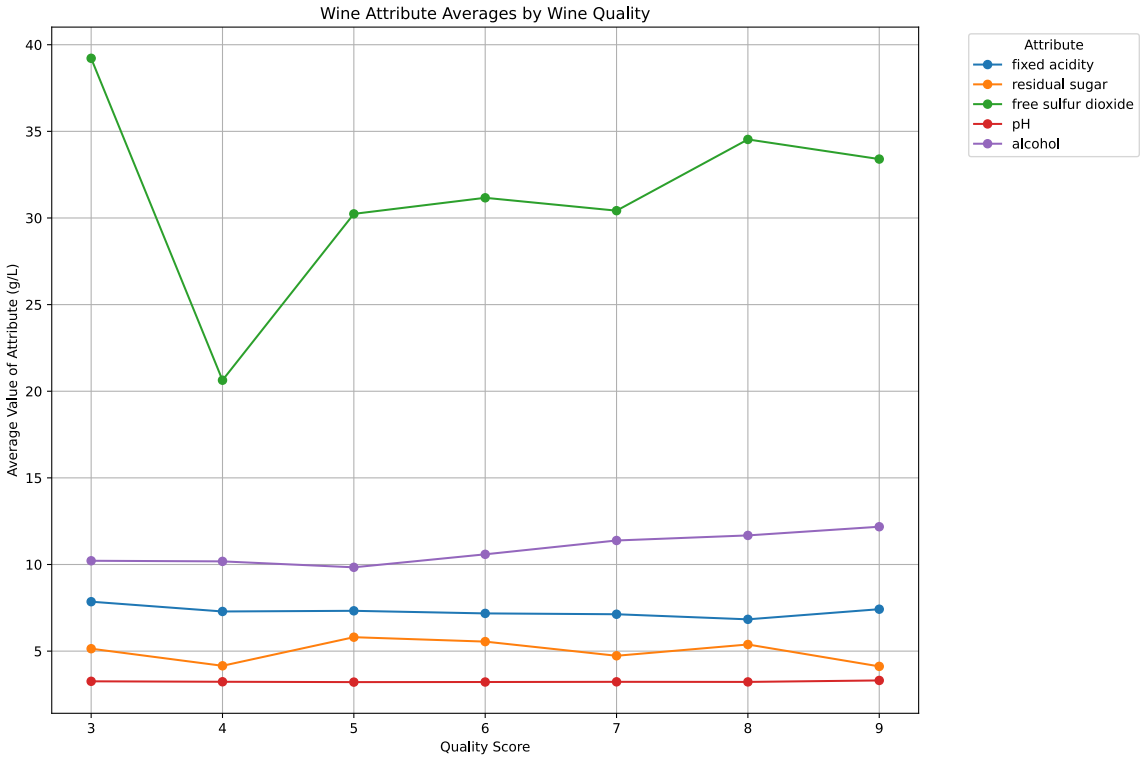
```
]].mean().reset_index()

# Melt into long format for seaborn
mean_data_melted = mean_data.melt(id_vars='is_red',
                                  var_name='attribute',
                                  value_name='mean_value')

# Change 0s and 1s to White and Red
mean_data_melted['is_red'] = mean_data_melted['is_red'].map({0: 'White', 1:
'Red'})


# Plot averages grouped by wine color
plt.figure(figsize=(12, 6))
sns.barplot(x='attribute', y='mean_value', hue='is_red',
data=mean_data_melted)
plt.title('Wine Attribute Averages by Wine Type')
plt.xlabel('Wine Attribute')
plt.ylabel('Average Value of Attribute (g/L)')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Is Red')
plt.tight_layout()
plt.show()
```



This analysis highlights higher sulfur dioxide and sugar in white wines, while red wines feature higher acidity and sulphates.

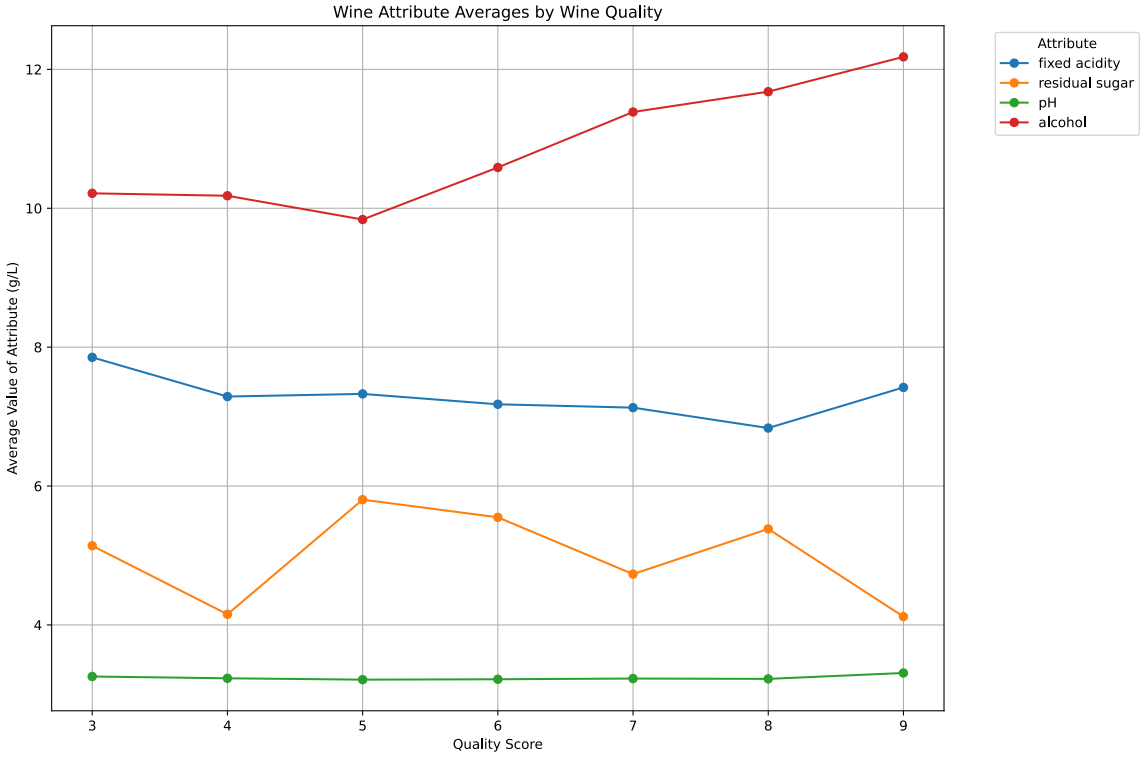
To identify which chemical attributes are most strongly associated with wine quality, we examine how average attribute levels change across different quality scores. This analysis reveals potential predictive relationships and guides our feature selection process.

```
# Quality Line Plot 1 - Average values above 1.0
# Group by quality and calculate the average of each attribute
quality_means = wdata.groupby('quality').mean()
quality_means = quality_means.drop(columns=['total sulfur dioxide', 'is_red',
'density', 'volatile acidity', 'citric acid', 'chlorides', 'sulphates'])
# Plot a line chart for each attribute
plt.figure(figsize=(12, 8))
for column in quality_means.columns:
    plt.plot(quality_means.index, quality_means[column], marker='o',
linestyle='-', label=column)
plt.title('Wine Attribute Averages by Wine Quality')
plt.xlabel('Quality Score')
plt.ylabel('Average Value of Attribute (g/L)')
plt.legend(title='Attribute', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
# Quality Line Plot 2 - Average values below 1.0
# Group by quality and calculate the average of each attribute
quality_means = wdata.groupby('quality').mean()
quality_means = quality_means.drop(columns=['total sulfur dioxide', 'is_red',
'free sulfur dioxide', 'density', 'volatile acidity', 'citric acid',
```

```
'chlorides', 'sulphates'])
# Plot a line chart for each attribute
plt.figure(figsize=(12, 8))
for column in quality_means.columns:
    plt.plot(quality_means.index, quality_means[column], marker='o',
linestyle='-', label=column)
plt.title('Wine Attribute Averages by Wine Quality')
plt.xlabel('Quality Score')
plt.ylabel('Average Value of Attribute (g/L)')
plt.legend(title='Attribute', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



These line charts show the changes in the average level of each attribute for each quality level. While some of the attributes show no consistent pattern, we see upward trends for alcohol and citric acid, meaning that an increased amount of alcohol and citric acid is correlated with a higher quality score. We also see downward trends for volatile acidity and chlorides, meaning that a decreased amount of volatile acidity and chlorides is correlated with a higher quality score.

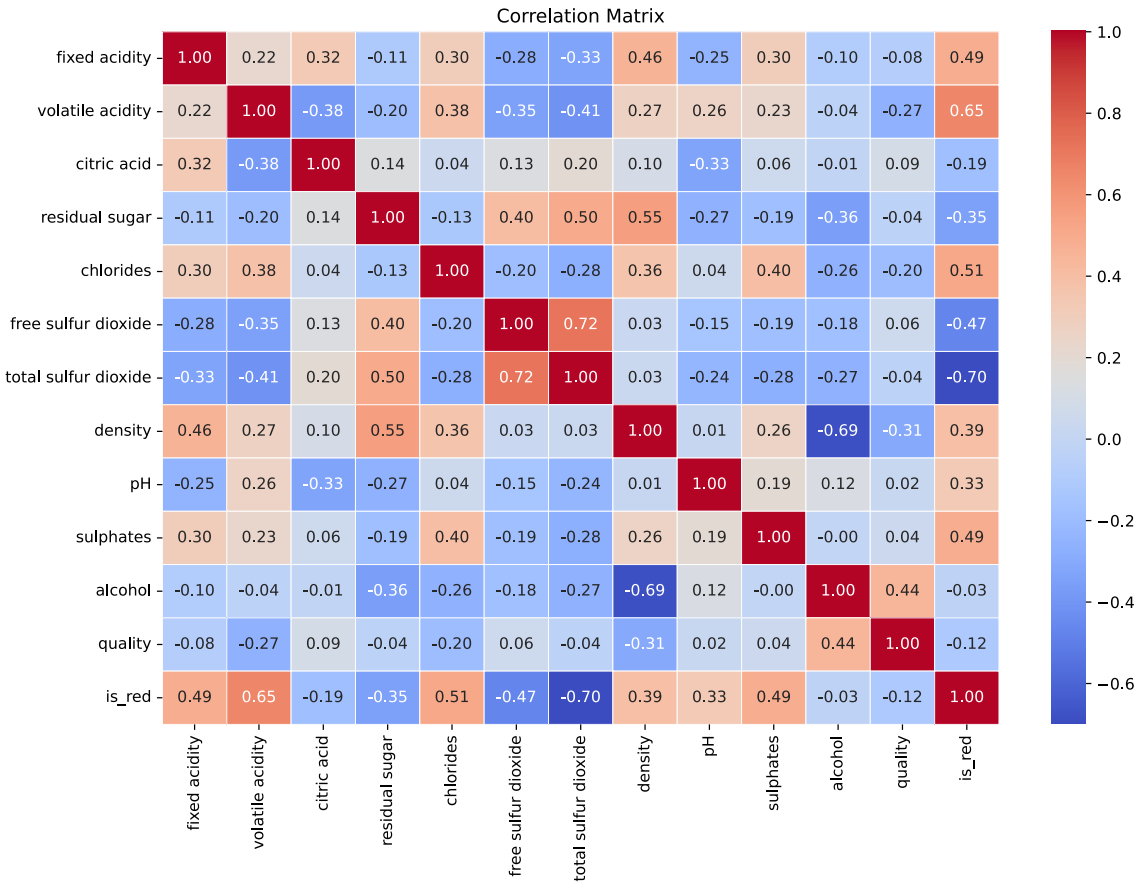
Data Preprocessing and Model Preperation

When it comes to limitations of the data, we had to cut several variables, as they had high levels of correlations with other variables in our dataset. Our dataset also lacks information to further

predict more than just wine quality. To further this project, our next steps would include gathering additional data about branding, pricing and demand in order for us to further predict wine viability in the market. Some of our data is also skewed, which could violate assumptions. We will handle these problem as we prepare to run our model.

We first use a heatmap plot to check for correlation between our variables. High correlations between predictors can lead to unstable coefficient estimates and reduced model interpretability.

```
# Calculate the correlation matrix
corr_matrix = wdata.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
import seaborn as sns
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=.5)
plt.title("Correlation Matrix")
plt.show()
```



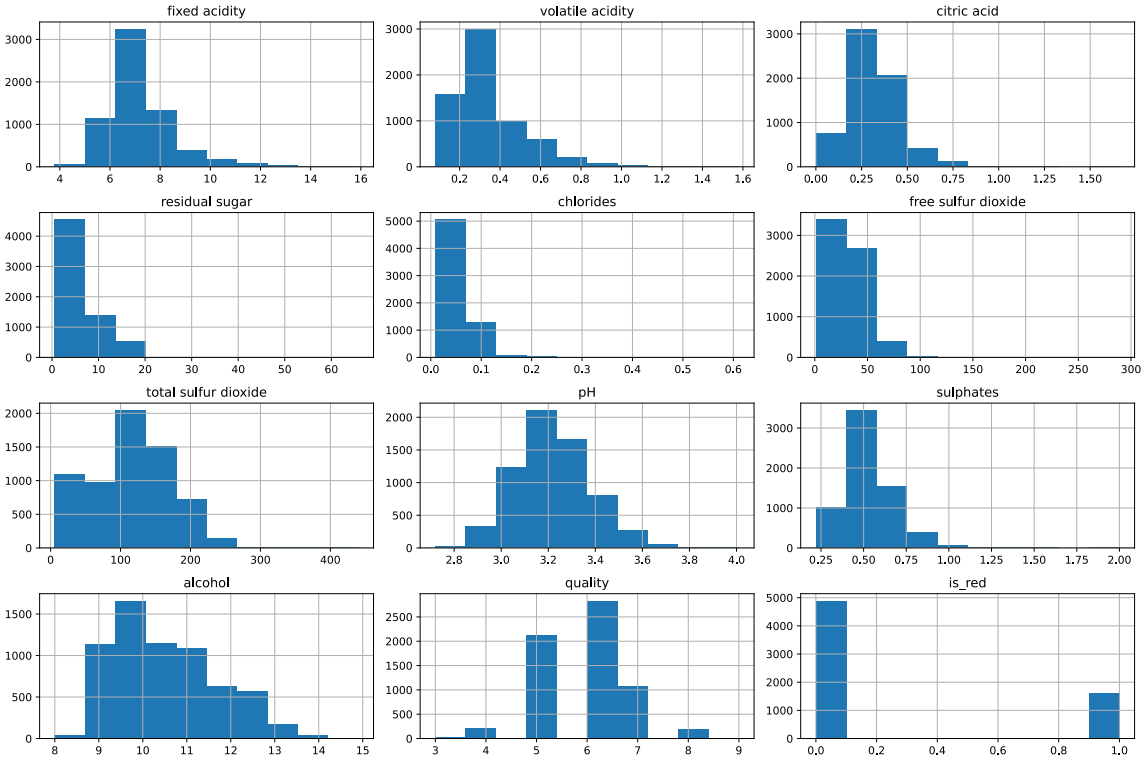
From this correlation heatmap, we notice several concerning relationships that require attention. Density shows strong relationships with multiple variables. PH relationships exhibit significant correlations with acidity measures. Sulfur dioxide connections (free and total sulfure dioxide) are highly correlated. Based on this analysis, we make our first data preprocessing decision:

```
#Remove density column
wdata_selected = wdata.drop('density', axis=1)
display(wdata_selected.head().T)
```

	0	1	2	3	4
fixed acidity	7.400	7.800	7.800	11.200	7.400
volatile acidity	0.700	0.880	0.760	0.280	0.700
citric acid	0.000	0.000	0.040	0.560	0.000
residual sugar	1.900	2.600	2.300	1.900	1.900
chlorides	0.076	0.098	0.092	0.075	0.076
free sulfur dioxide	11.000	25.000	15.000	17.000	11.000
total sulfur dioxide	34.000	67.000	54.000	60.000	34.000
pH	3.510	3.200	3.260	3.160	3.510
sulphates	0.560	0.680	0.650	0.580	0.560
alcohol	9.400	9.800	9.800	9.800	9.400
quality	5.000	5.000	5.000	6.000	5.000
is_red	1.000	1.000	1.000	1.000	1.000

Linear regression models perform optimally when predictors follow approximately normal distributions. We examine the distribution of each variable to identify those requiring transformation:

```
# Plot histograms for all numerical columns in wdata_selected
wdata_selected.hist(figsize=(15, 10))
plt.tight_layout()
plt.show()
```



We can see several of our data points are not normally distributed. This breaks an assumption of our model. We will now run some code to logarithmically transform the skewed variables so that their distributions will look more normal:

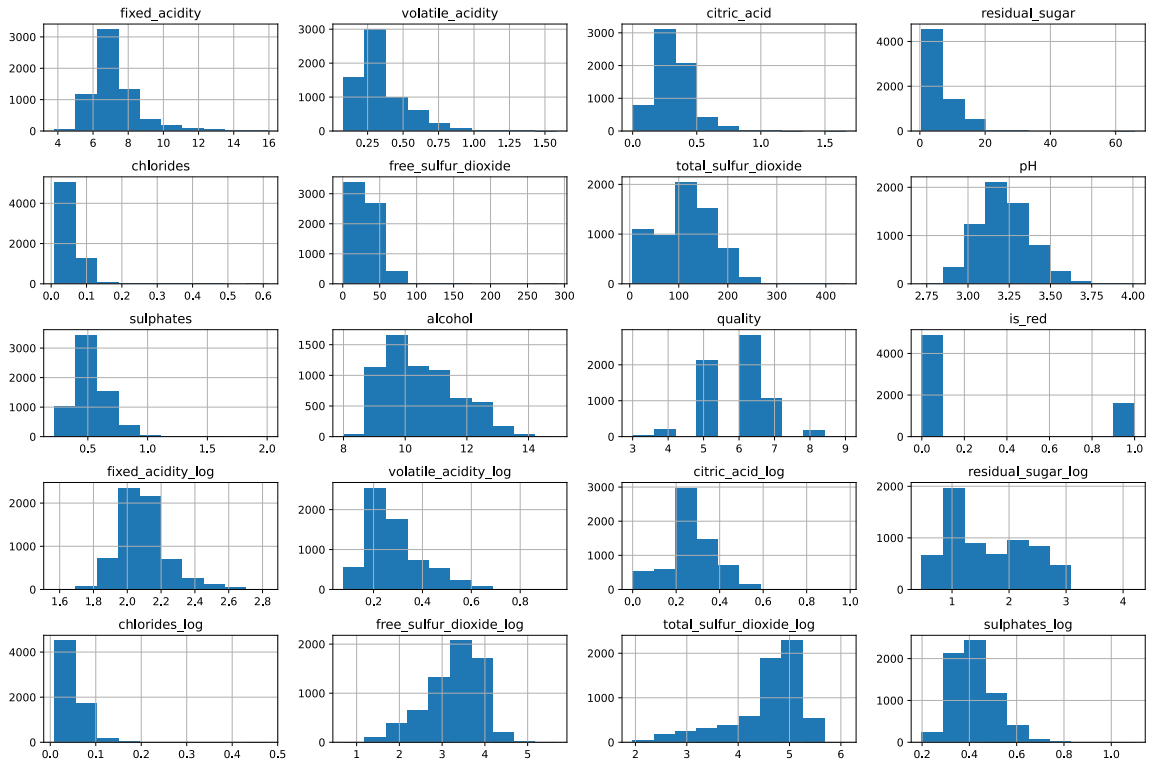
```
# Identify right-skewed variables from the histograms
# Based on visual inspection of the histograms, variables like:
# residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide,
sulphates
# appear to be right-skewed.

# Create a copy of the selected data to apply transformations
wdata_t = wdata_selected.copy()

# Apply log transformation (np.log1p) to the identified skewed variables
skewed_vars = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual
sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',
'sulphates']

for var in skewed_vars:
    # Add a small constant before log transformation if there are zero values,
    # or use np.log1p which is log(1+x) and handles zero values
    wdata_t[f"{var}_log"] = np.log1p(wdata_t[var])

# Replace variable name spaces with underscores
```

```
wdata_t.columns = [col.replace(' ', '_') for col in wdata_t.columns]

print("Data after applying log transformation to skewed variables:")
display(wdata_t.head().T)
```

```
Data after applying log transformation to skewed variables:
```

	0	1	2	3	4
fixed_acidity	7.400000	7.800000	7.800000	11.200000	7.400000
volatile_acidity	0.700000	0.880000	0.760000	0.280000	0.700000
citric_acid	0.000000	0.000000	0.040000	0.560000	0.000000
residual_sugar	1.900000	2.600000	2.300000	1.900000	1.900000
chlorides	0.076000	0.098000	0.092000	0.075000	0.076000
free_sulfur_dioxide	11.000000	25.000000	15.000000	17.000000	11.000000
total_sulfur_dioxide	34.000000	67.000000	54.000000	60.000000	34.000000
pH	3.510000	3.200000	3.260000	3.160000	3.510000
sulphates	0.560000	0.680000	0.650000	0.580000	0.560000
alcohol	9.400000	9.800000	9.800000	9.800000	9.400000
quality	5.000000	5.000000	5.000000	6.000000	5.000000
is_red	1.000000	1.000000	1.000000	1.000000	1.000000
fixed_acidity_log	2.128232	2.174752	2.174752	2.501436	2.128232
volatile_acidity_log	0.530628	0.631272	0.565314	0.246860	0.530628
citric_acid_log	0.000000	0.000000	0.039221	0.444686	0.000000
residual_sugar_log	1.064711	1.280934	1.193922	1.064711	1.064711
chlorides_log	0.073250	0.093490	0.088011	0.072321	0.073250
free_sulfur_dioxide_log	2.484907	3.258097	2.772589	2.890372	2.484907
total_sulfur_dioxide_log	3.555348	4.219508	4.007333	4.110874	3.555348
sulphates_log	0.444686	0.518794	0.500775	0.457425	0.444686

```
wdata_t.hist(figsize=(15, 10))
plt.tight_layout()
plt.show()
```



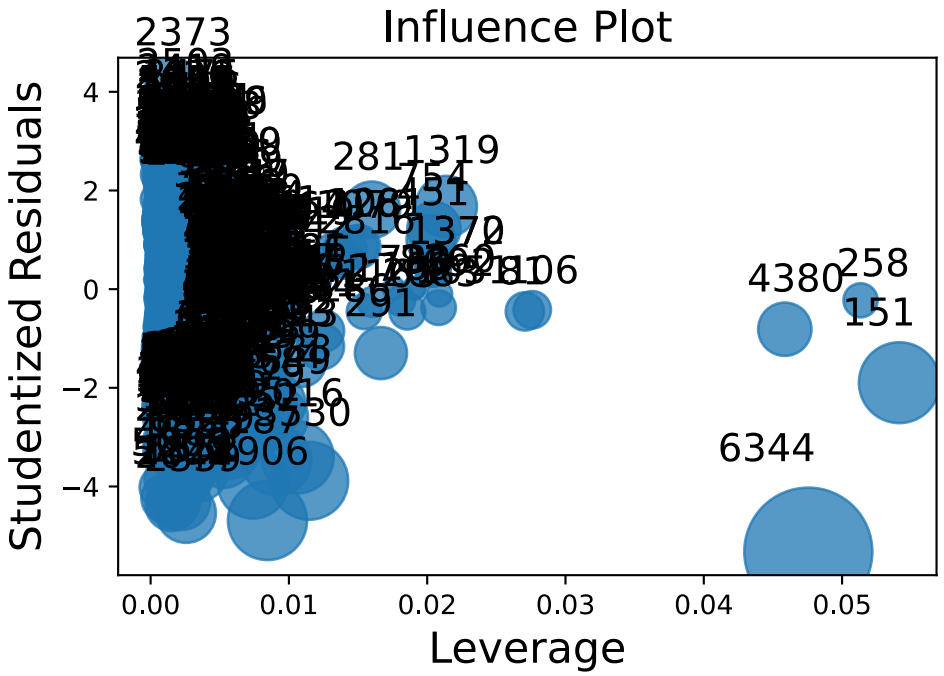
We can see that the logarithmic transformations helped the variables become more normal, although some of them still have some skew.

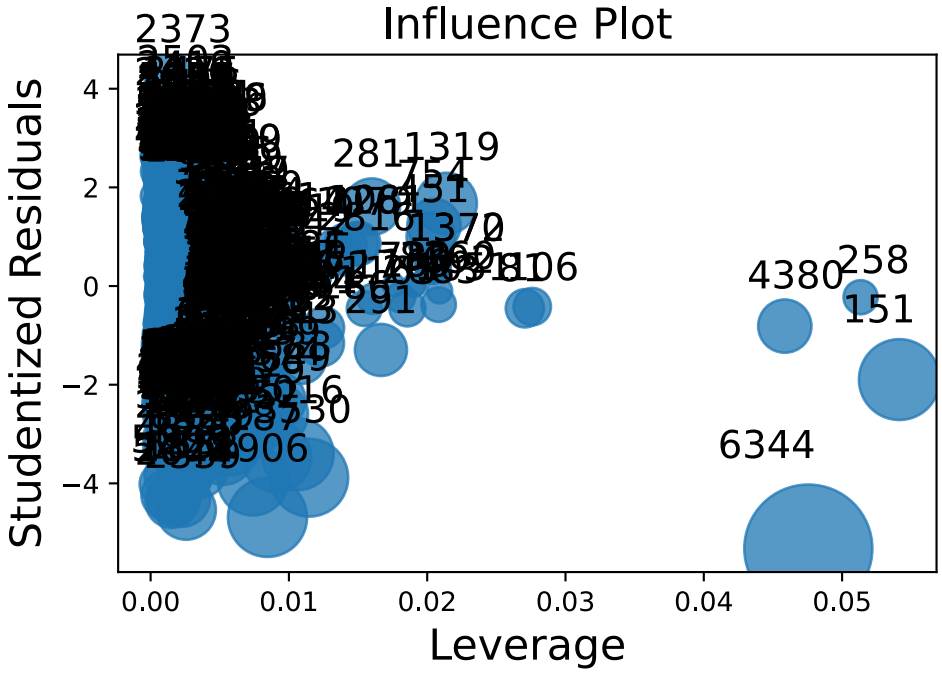
Statistical Modeling and Diagnostics

With our preprocessed data, we begin building our regression model. However, before finalizing the model, we must identify and address potentially influential observations that could skew our results.

```
# Define the independent variables (features) and the dependent variable
(target) using the transformed data
# Ensure the DataFrame used for the formula contains the 'quality' column
predictors1 = [
    'fixed_acidity_log', 'volatile_acidity_log', 'citric_acid_log',
'residual_sugar',
    'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide',
'sulphates_log',
    'alcohol', 'pH', 'is_red'
]

# Fit the OLS regression model using the wdata_t DataFrame which contains
'quality'
# Use backticks for column names that originally had spaces, although renaming
them is better.
```

```
# Since we have renamed columns, we can use the renamed names directly.
formula = 'quality ~ ' + ' + '.join(predictors1)
fit1 = smf.ols(formula, data=wdata_t).fit()
wdata_t['residuals_01'] = fit1.resid
wdata_t['fittedvalues_01'] = fit1.fittedvalues
sm.graphics.influence_plot(fit1, criterion = 'dffits')
```

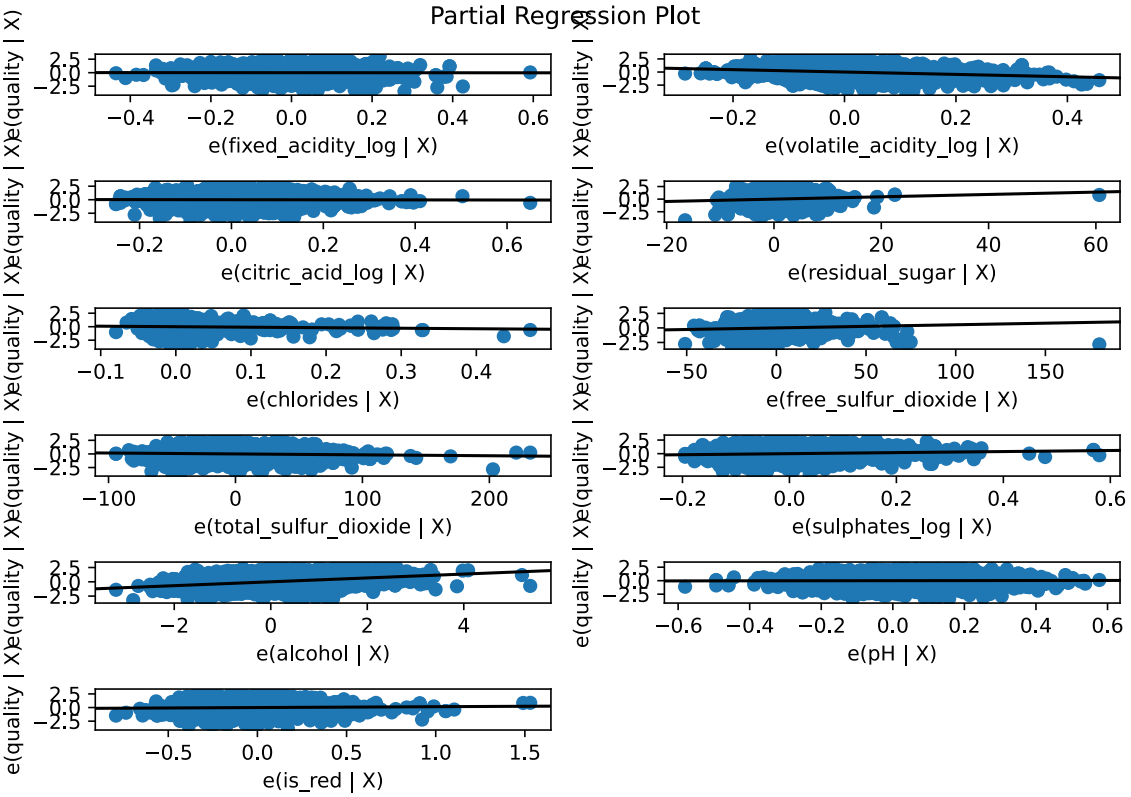




We can see some strong leverage points towards the right side of the graph. Before removing these, we will remove any multicollinear columns and check for influential points again.

We will now create partial regression plots. Partial regression plots help us understand the relationship between each predictor and the response variable while controlling for all other predictors. This analysis also reveals influential observations for individual variables.

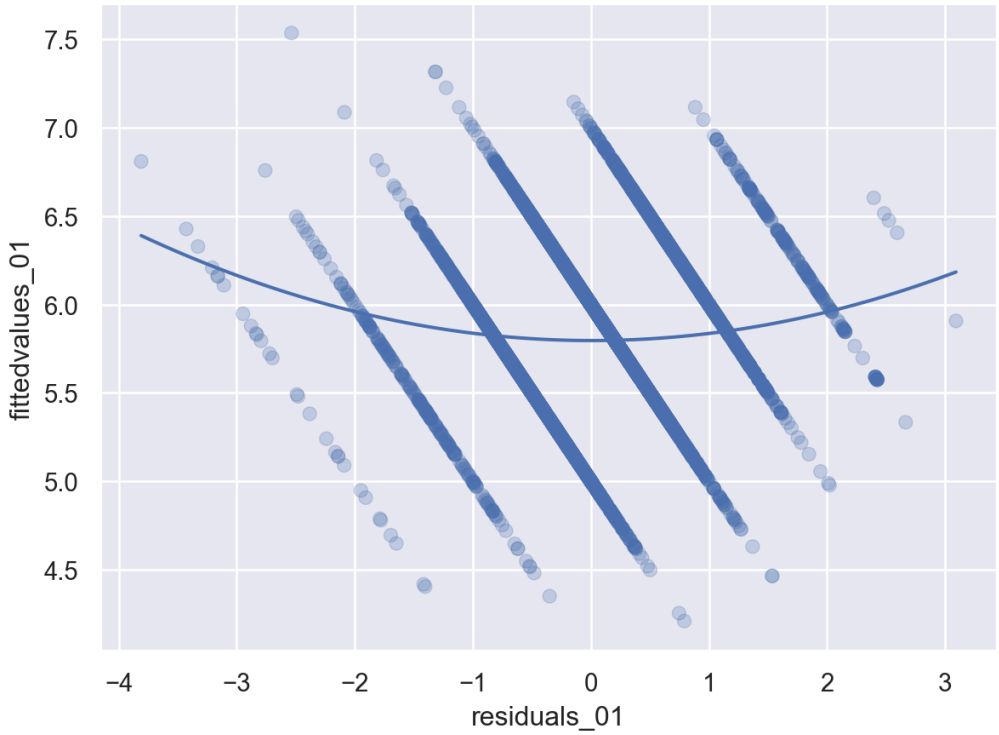
```
#plotting a partioal regression plot
Nfig = plt.figure(figsize = (8,7))
# Get the actual names of the exogenous variables from the fitted model,
excluding the intercept
exog_names = [name for name in fit1.model.exog_names if name != 'Intercept']
sm.graphics.plot_partregress_grid(
fit1,
exog_idx = exog_names,
grid = (8, 2),
fig = Nfig
)
Nfig.tight_layout()
```



Our partial regression plot also shows some influential points, particularly for residual sugar, chlorides, and total sulfur dioxide. But the dispersion of points is linear for every plot, which means our variables are all linear.

Next, we generate a plot of the residuals and fitted values to make sure our variance is constant:

```
#plotting residuals
(so.Plot(wdata_t, x = 'residuals_01', y = 'fittedvalues_01')
.add(so.Dot(alpha = 0.25))
.add(so.Line(), so.PolyFit())
)
```

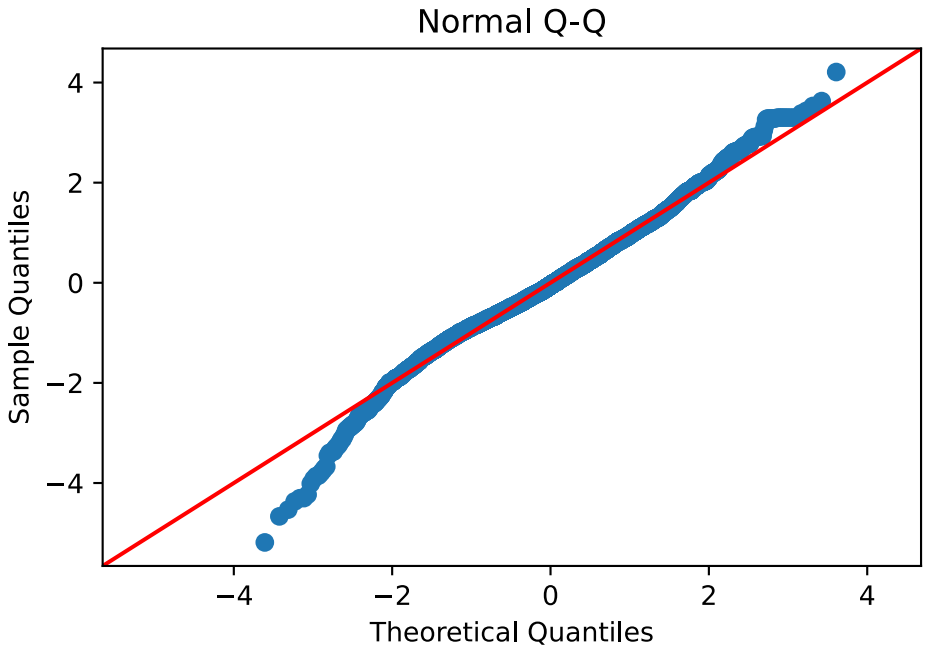


Our plot of residuals and fitted values shows a clouded shape, which means that our data is homoschedastic and that our variance is constant.

Next, to check for normality, we created a Q-Q plot:

```
#creating a q-q plot
sm.qqplot(
wdata_t['residuals_01'],
line = '45',
fit = True
)
plt.title("Normal Q-Q")
```

```
Text(0.5, 1.0, 'Normal Q-Q')
```



Our Q-Q plot also adheres to the line from −2 to 2 quantiles, although it goes below the diagnoal line past −2 quantiles. This shows that our data is roughly normal.

Finally, to check for multicollinearity, we will compute the variance inflation factors for each predictor. This will show us how much the value of each predictor is explained by another variable in our selection.

```
#creating a table to check variance inflation factors
vif_data = wdata_t[predictors1].copy()

for col in vif_data.columns:
    if vif_data[col].dtype == 'bool':
        vif_data[col] = vif_data[col].astype(int)

vif = np.zeros(len(predictors1))
for i in range(0, len(predictors1)):
    vif[i] = variance_inflation_factor(vif_data, i)

vif_results = pd.DataFrame({
    'predictors': predictors1,
    'vif': vif
})

display(vif_results)
```

	predictors	vif
0	fixed_acidity_log	182.509123
1	volatile_acidity_log	15.722883
2	citric_acid_log	11.585575
3	residual_sugar	3.365820
4	chlorides	5.540018
5	free_sulfur_dioxide	8.725275
6	total_sulfur_dioxide	20.739984
7	sulphates_log	32.445210
8	alcohol	103.682970
9	pH	208.997486
10	is_red	5.885229

We clearly have some high variance inflation factors in our dataset. We will need to remove some columns.

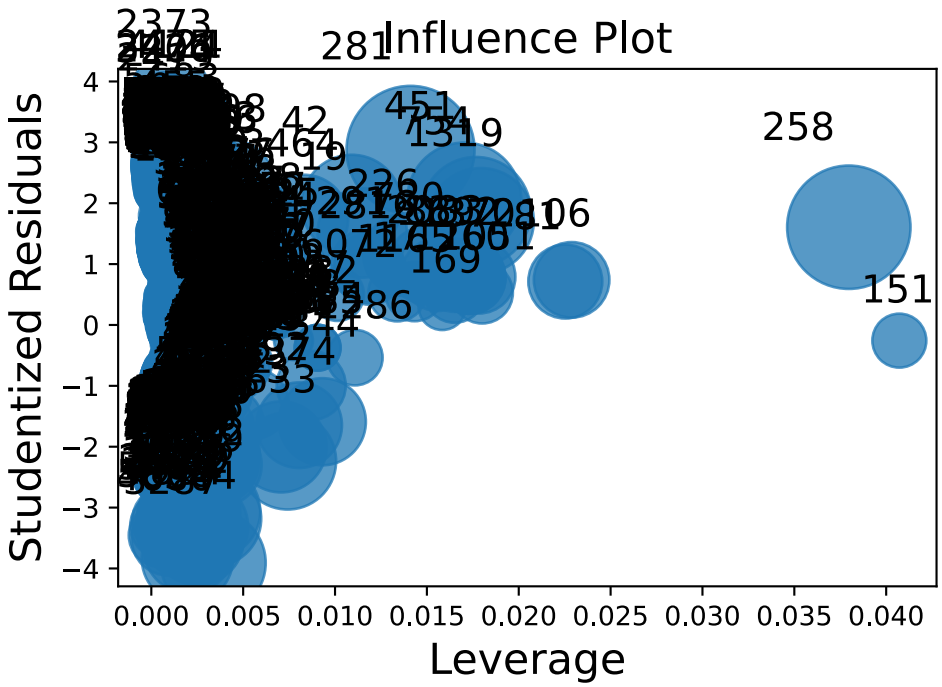
```
predictors2 = [
    'fixed_acidity_log', 'citric_acid_log', 'residual_sugar_log',
    'chlorides_log', 'free_sulfur_dioxide_log', 'sulphates_log', 'is_red'
]

formula = 'quality ~ ' + ' + '.join(predictors2)
fit2 = smf.ols(formula, data=wdata_t).fit()
wdata_t['residuals_02'] = fit2.resid
wdata_t['fittedvalues_02'] = fit2.fittedvalues

vif_data = wdata_t[predictors2].copy()

for col in vif_data.columns:
    if vif_data[col].dtype == 'bool':
        vif_data[col] = vif_data[col].astype(int)

vif = np.zeros(len(predictors2))
for i in range(0, len(predictors2)):
    vif[i] = variance_inflation_factor(vif_data, i)

vif_results = pd.DataFrame({
    'predictors': predictors2,
    'vif': vif
})

display(vif_results)
```

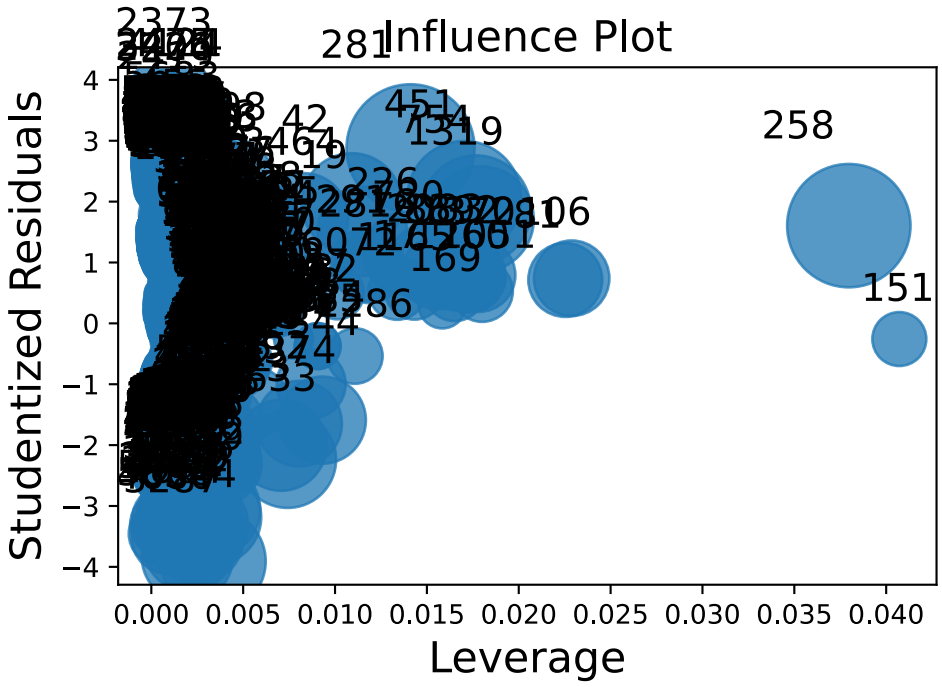
	predictors	vif
0	fixed_acidity_log	62.844861
1	citric_acid_log	8.765243
2	residual_sugar_log	8.262965
3	chlorides_log	6.016120
4	free_sulfur_dioxide_log	34.254661
5	sulphates_log	29.577923
6	is_red	3.373731

We removed pH, alcohol, total sulfur dioxide, and volatile acidity. Now, the VIF scores are still high for a few of the variables, but they are much lower than some of the high-ranking variables. We chose to keep fixed_acidity_log, free_sulfur_dioxide_log, and sulphates_log because they are important aspects of the wine that we want to include in our analysis, even if part of them is explained by other variables.

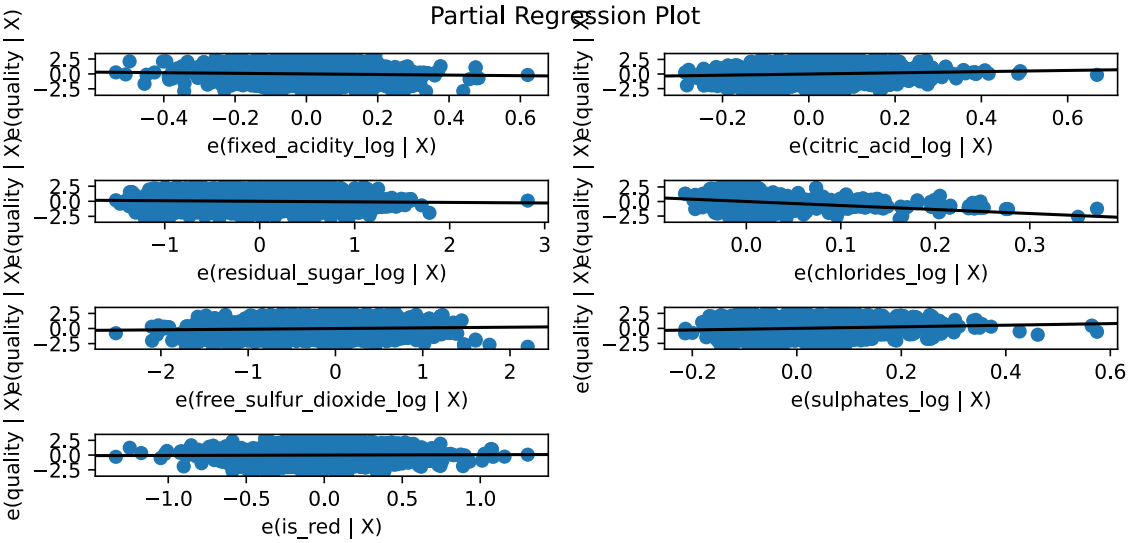
We want to make sure that our model assumptions are still met after removing these columns, so we will rerun our code for influence points, the partial regression plot, the map of residuals and fitted values, and the Q-Q plot:

```
#graphing influence plots
sm.graphics.influence_plot(fit2, criterion = 'dffits')
```

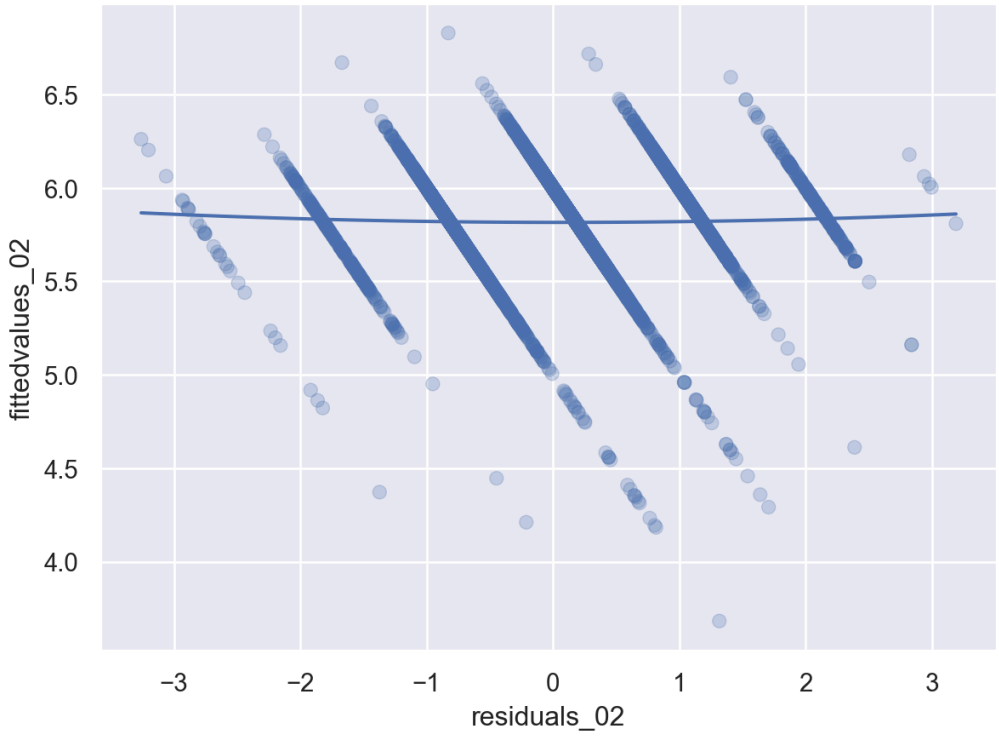





```
#once again graphing partial regression plots
fig = plt.figure(figsize = (8,7))
exog_names = [name for name in fit2.model.exog_names if name != 'Intercept']
sm.graphics.plot_partregress_grid(
fit2,
exog_idx = exog_names,
grid = (8, 2),
fig = fig
)
fig.tight_layout()
```

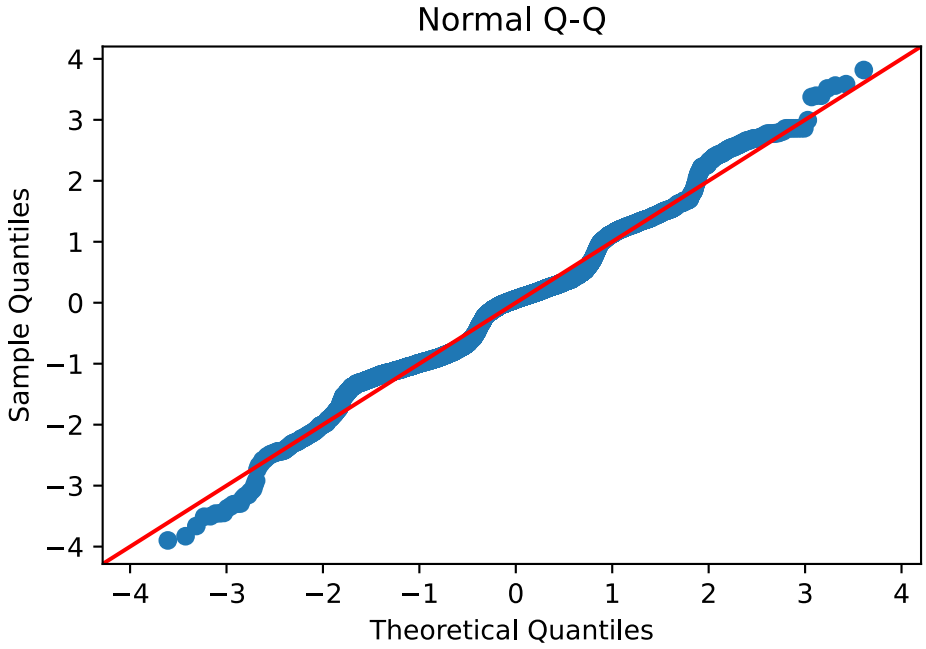



```
#checking residuals again
wdata_t['residuals_02'] = fit2.resid
wdata_t['fittedvalues_02'] = fit2.fittedvalues

(so.Plot(wdata_t, x = 'residuals_02', y = 'fittedvalues_02')
.add(so.Dot(alpha = 0.25))
.add(so.Line(), so.PolyFit())
)
```




```
#checking Q-Q plot again
sm.qqplot(
wdata_t['residuals_02'],
line = '45',
fit = True
)
plt.title("Normal Q-Q")
```

```
Text(0.5, 1.0, 'Normal Q-Q')
```



After removing those columns, we can see that the effects of influential points, shown in our influence plot and partial regression plot, are reduced. Our plot of residuals and fitted values still maintains a clouded shape, and the Q-Q plot still follows the diagonal line, although the logarithmic transform contribute to the wavy pattern.

Model Assumptions and Validation

We plan on using an Ordinary Least Squares (OLS) model to train and evaluate the data in order to make predictions about the relationship between our predictor variables and wine quality. Running an OLS model has a few assumptions that it makes about the data. These assumptions come with running a model that uses linear regression. These assumptions are addressed in the bullet points below:

Linear Regression Assumptions:

- Validity - The most important variables for assessing wine quality are in our dataset. There are things missing, like grape type, wine brand, and wine selling price. However, two of those variables might have more of a placebo effect on quality rather than a realistic effect. Some of the information grape type would capture can also be captured in wine color, as grape type determines the possible colors while production finalizes the color.
- Representativeness - The dataset comes from a Portuguese wine company and spans from 2004 to 2007. The data were collected and gathered for a research paper for a university in Portugal and made public afterwards. There were no missing values in the dataset, so we can apply our analyses without worrying about whether data was missing at random or not. We found five points that skewed the data significantly, but their impact was lessened when we dropped some of the columns they were skewing.

- Linearity - We made a partial regression plot of quality onto each variable, and all of the relationships look linear, clustered around a general area. There are still some points that are far from the mean, but performing logarithmic transformations on the variables made their effect less severe.
- Independence - There are no sequences or clusterization to this data. Each row represents a different vihno verde wine.
- Constant Variance - The plot of the residuals and the fitted values is in a clouded (circular) shape, meaning that there is homoscedasticity of errors.
- Normality - Our Q-Q plot follows the diagonal line until it gets past two quantiles on either side. After performing log transformations on the variables, the Q-Q chart extends to further quantiles in the Q-Q plot, and the distributions of individual variables appears more normal.
- Multicollinearity - We knew that some columns were going to be coordinated, such as density, alcohol, and pH. After doing a VIF analysis, we also dropped total sulfur dioxide (which is related to free sulfur dioxide) and volatile acidity (related to fixed acidity and citric acid).

Model Training and Performance Evaluation

Now that we have met all our model assumptions, we can start to fit our data to the model. First, we will split our dataset into two groups: one for training, and one for testing. We will fit the model to the training dataset and then apply it to the testing dataset to avoid overfitting to our testing data. While the OLS model is calculated using our training data, our mean squared error is calculated using our testing data.

```
#spliting data on an 80-20 split
X = wdata_t.drop('quality', axis=1)
y = wdata_t['quality']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

#dropping unused columns
X_train = X_train.drop(columns=[
    'volatile_acidity', 'total_sulfur_dioxide', 'pH',
    'alcohol', 'residuals_01', 'fittedvalues_01', 'residuals_02',
'fittedvalues_02', 'fixed_acidity', 'citric_acid', 'residual_sugar',
'chlorides', 'free_sulfur_dioxide',
    'sulphates'])

# Append quality back onto our training set for X
w_train = pd.concat([y_train.rename('quality'), X_train], axis=1)

w_train.head().T
```

	1916	947	877	2927	6063
quality	5.000000	7.000000	6.000000	6.000000	5.000000

	1916	947	877	2927	6063
is_red	0.000000	1.000000	1.000000	0.000000	0.000000
fixed_acidity_log	2.028148	2.230014	2.163323	1.824549	2.028148
volatile_acidity_log	0.215111	0.246860	0.539413	0.314811	0.231112
citric_acid_log	0.300105	0.392042	0.009950	0.285179	0.444686
residual_sugar_log	2.163323	1.131402	1.131402	0.788457	2.797281
chlorides_log	0.030529	0.088926	0.062035	0.027615	0.051643
free_sulfur_dioxide_log	3.610918	1.945910	3.465736	2.639057	3.496508
total_sulfur_dioxide_log	4.912655	2.564949	3.784190	4.406719	4.955827
sulphates_log	0.314811	0.482426	0.451076	0.322083	0.398776

```
#doing a third fit
fit_03 = smf.ols(
'quality ~ ' + ' + '.join(predictors2),
data = wdata_t
).fit()
```

```
# Make predictions on the test data using fit_03
y_pred_03 = fit_03.predict(X_test)

# Calculate the Mean Squared Error (MSE)
mse_03 = mean_squared_error(y_test, y_pred_03)

print(f"Mean Squared Error (MSE) for fit_03: {mse_03}")
```

```
Mean Squared Error (MSE) for fit_03: 0.6712058375147787
```

```
#running a fourth fit
predictors3 = [
    'citric_acid_log', 'residual_sugar_log',
    'chlorides_log', 'free_sulfur_dioxide_log', 'sulphates_log', 'is_red'
]
fit_04 = smf.ols(
'quality ~ ' + ' + '.join(predictors3),
data = wdata_t
).fit()

# Make predictions on the test data using fit_03
y_pred_04 = fit_04.predict(X_test)
```

```
# Calculate the Mean Squared Error (MSE)
mse_04 = mean_squared_error(y_test, y_pred_04)

print(f"Mean Squared Error (MSE) for fit_03: {mse_04}")
```

```
Mean Squared Error (MSE) for fit_03: 0.6773355840673311
```

We tried a few different combinations of variables in our linear regression model, but fit_03 (with predictors 'fixed_acidity_log', 'citric_acid_log', 'residual_sugar_log', 'chlorides_log', 'free_sulfur_dioxide_log', 'sulphates_log', 'is_red') had the lowest mean squared error, so that is the model we will use.

Final Model Results and Interpretation

```
import pandas as pd
import numpy as np

# Corrected function to calculate quality based on fit_03 coefficients
def calculate_quality(fixed_acidity_log, citric_acid_log, residual_sugar_log,
chlorides_log, free_sulfur_dioxide_log, sulphates_log, is_red):
  # Pull the coefficients from fit_03
  coefficients = fit_03.params

  # Calculate predicted quality using the coefficients and input values
  predicted_quality = (coefficients['Intercept'] +
                       coefficients['fixed_acidity_log'] * fixed_acidity_log +
                       coefficients['citric_acid_log'] * citric_acid_log +
                       coefficients['residual_sugar_log'] * residual_sugar_log
+
                       coefficients['chlorides_log'] * chlorides_log +
                       coefficients['free_sulfur_dioxide_log'] *
free_sulfur_dioxide_log +
                       coefficients['sulphates_log'] * sulphates_log +
                       coefficients['is_red'] * is_red)
  return predicted_quality

# Create a DataFrame with example input values (using original scale for
skewed variables)
# Replace these values with the theoretical values you want to test
theoretical_data_original_scale = pd.DataFrame({
    'fixed_acidity': [7.0, 5.0, 3.0],
    'citric_acid': [0.3, 0.5, 0.7],
    'residual_sugar': [2.0, 1.5, 1.0],
    'chlorides': [0.05, 0.03, 0.01],
    'free_sulfur_dioxide': [15.0, 20.0, 25.0],
    'sulphates': [0.5, 1.0, 1.5],
```

```
    'is_red': [1, 1, 1] # 1 for red wine, 0 for white wine
})

# Apply log transformation to the relevant variables and update column names
theoretical_data_transformed = theoretical_data_original_scale.copy()
skewed_vars_fit03 = ['fixed_acidity', 'citric_acid', 'residual_sugar',
'chlorides', 'free_sulfur_dioxide', 'sulphates']

for var in skewed_vars_fit03:
    if var in theoretical_data_transformed.columns:
        theoretical_data_transformed[f"{var}_log"] =
np.log1p(theoretical_data_transformed[var])
        # Drop the original scale column if you only want the log-transformed
version
        # theoretical_data_transformed =
theoretical_data_transformed.drop(columns=[var])

# Now, use the transformed data with the calculate_quality function
predicted_qualities = theoretical_data_transformed.apply(
    lambda row: calculate_quality(
        fixed_acidity_log=row['fixed_acidity_log'] if 'fixed_acidity_log' in
row else row['fixed_acidity'], # Use log if exists, otherwise original
        citric_acid_log=row['citric_acid_log'] if 'citric_acid_log' in row
else row['citric_acid'],
        residual_sugar_log=row['residual_sugar_log'] if 'residual_sugar_log'
in row else row['residual_sugar'],
        chlorides_log=row['chlorides_log'] if 'chlorides_log' in row else
row['chlorides'],
        free_sulfur_dioxide_log=row['free_sulfur_dioxide_log'] if
'free_sulfur_dioxide_log' in row else row['free_sulfur_dioxide'],
        sulphates_log=row['sulphates_log'] if 'sulphates_log' in row else
row['sulphates'],
        is_red=row['is_red']
    ),
    axis=1
)


print("Predicted quality for theoretical data:")
display(predicted_qualities)
```

```
Predicted quality for theoretical data:
```

```
0    5.873863
1    6.722002
```

```
2    7.525457
dtype: float64
```

Using our model, we made three wines with different qualities and tested what our model would predict their quality to be. Following the trends seen in our model output, a higher level of sulphates, free sulfur dioxide, and citric acid is associated with greater quality, while a lower level of chlorides, residual sugar, and fixed acidity is associated with greater quality.

```
# Predictors used in fit_03:
predictors_fit_03 = [
    'fixed_acidity_log', 'citric_acid_log', 'residual_sugar_log',
    'chlorides_log', 'free_sulfur_dioxide_log', 'sulphates_log', 'is_red'
]

# Define the formula using the predictors from fit_03
formula_final = 'quality ~ ' + ' + '.join(predictors_fit_03)

# Fit the OLS regression model using the entire transformed dataset (wdata_t)
fit_final = smf.ols(formula_final, data=wdata_t).fit()

# Display the model summary
print(fit_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                quality   R-squared:                       0.083
Model:                            OLS   Adj. R-squared:                  0.082
Method:                 Least Squares   F-statistic:                     83.79
Date:                Sat, 11 Oct 2025   Prob (F-statistic):          4.27e-117
Time:                        21:24:30   Log-Likelihood:                -8056.7
No. Observations:                6497   AIC:                         1.613e+04
Df Residuals:                    6489   BIC:                         1.618e+04
Df Model:                           7
Covariance Type:            nonrobust
================================================================================================
                               coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept                    6.1830      0.204     30.375      0.000
5.784       6.582
fixed_acidity_log           -0.4989      0.094     -5.316      0.000
-0.683      -0.315
citric_acid_log              1.0186      0.113      9.050      0.000
0.798       1.239
residual_sugar_log          -0.0936      0.017     -5.512      0.000
-0.127      -0.060
```

```
chlorides_log              -6.8755      0.410    -16.789      0.000
-7.678      -6.073
free_sulfur_dioxide_log     0.1089      0.020      5.460      0.000
0.070       0.148
sulphates_log               1.3207      0.134      9.855      0.000
1.058       1.583
is_red                      0.0619      0.040      1.537      0.124
-0.017       0.141
==============================================================================
Omnibus:                       54.716   Durbin-Watson:                   1.657
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               62.126
Skew:                           0.174   Prob(JB):                     3.23e-14
Kurtosis:                       3.329   Cond. No.                         174.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

Final Model Equation: Quality = 6.183 - 0.499×fixed_acidity_log + 1.019×citric_acid_log - 0.094×residual_sugar_log - 6.876×chlorides_log + 0.109×free_sulfur_dioxide_log + 1.321×sulphates_log + 0.062×is_red

Confidence Intervals and Statistical Inference

```
#importing required library
import math

#capturing confidence intervals
frequentist_conf_int = fit_final.conf_int()

#formula for printing our the correct interpretations of the confidence
intervals
print("Frequentist interpretation of confidence intervals")
for index, row in frequentist_conf_int.iterrows():
  if index == 'Intercept':
    print(f"We are 95% confident that the {index} for our equation is between
{row[0]:.2f} and {row[1]:.2f}")
  elif index == 'is_red':

    print(f"We are 95% confident that the coefficient for {index} is between
{C1} and {C2}, in relation to white wine, holding all other variables fixed,
however this interval range crosses over zero and might not be statistically
significant.")

  else:
    C1 = row[0]
```

```
    C2 = row[1]
    C1 = round(math.exp(C1),3)
    C2 = round(math.exp(C2),3)


    print(f"We are 95% confident that the coefficient for {index} is between
{C1} and {C2}, holding all other variables fixed.")
```

```
Frequentist interpretation of confidence intervals
We are 95% confident that the Intercept for our equation is between 5.78 and
6.58
We are 95% confident that the coefficient for fixed_acidity_log is between
0.505 and 0.73, holding all other variables fixed.
We are 95% confident that the coefficient for citric_acid_log is between 2.221
and 3.453, holding all other variables fixed.
We are 95% confident that the coefficient for residual_sugar_log is between
0.881 and 0.941, holding all other variables fixed.
We are 95% confident that the coefficient for chlorides_log is between 0.0 and
0.002, holding all other variables fixed.
We are 95% confident that the coefficient for free_sulfur_dioxide_log is
between 1.072 and 1.16, holding all other variables fixed.
We are 95% confident that the coefficient for sulphates_log is between 2.881
and 4.872, holding all other variables fixed.
We are 95% confident that the coefficient for is_red is between 2.881 and
4.872, in relation to white wine, holding all other variables fixed, however
this interval range crosses over zero and might not be statistically
significant.
```

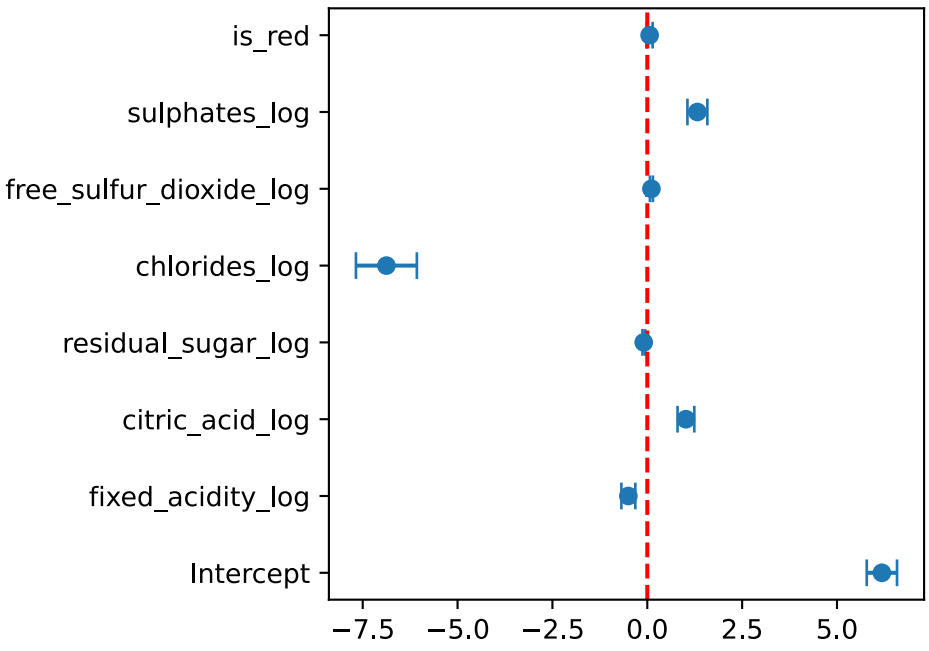
From this report we can extract our confidence intervals. Here are the confidence intervals.

Frequentist interpretation of confidence intervals * We are 95% confident that the Intercept for our equation is between 5.78 and 6.58, holding all other variables fixed.

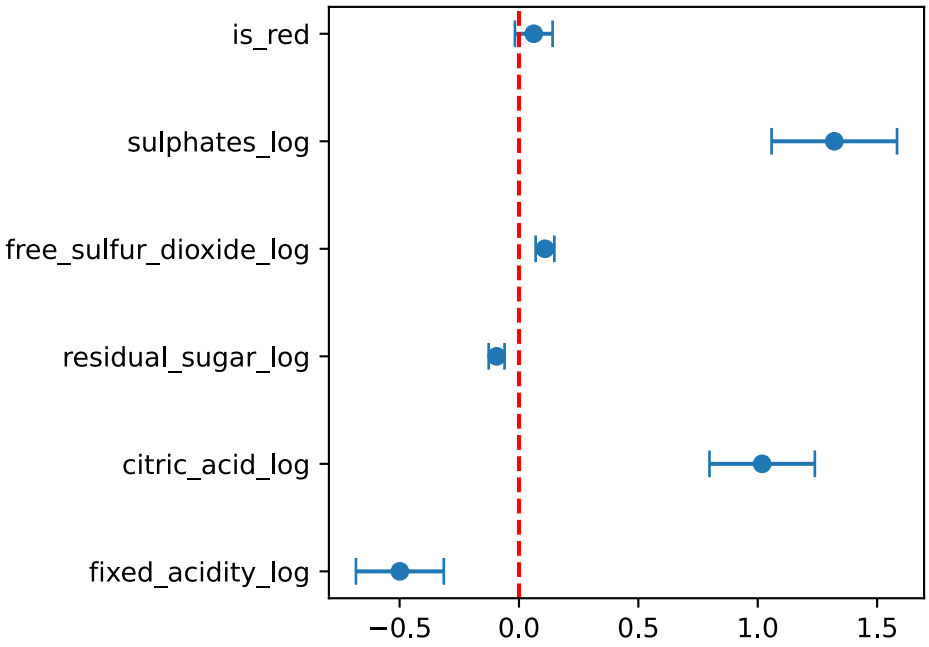
- We are 95% confident that the coefficient for fixed_acidity is between 0.505 and 0.73, holding all other variables fixed.
- We are 95% confident that the coefficient for citric_acid is between 2.221 and 3.453, holding all other variables fixed.
- We are 95% confident that the coefficient for residual_sugar is between 0.881 and 0.941, holding all other variables fixed.
- We are 95% confident that the coefficient for chlorides is between 0.0 and 0.002, holding all other variables fixed.
- We are 95% confident that the coefficient for free_sulfur_dioxide is between 1.072 and 1.16, holding all other variables fixed.

- We are 95% confident that the coefficient for sulphates is between 2.881 and 4.872, holding all other variables fixed.
- We are 95% confident that the coefficient for is_red is between −0.02 and 0.14, holding all other variables fixed, in relation to white wine, however this interval range crosses over zero and might not be statistically significant.

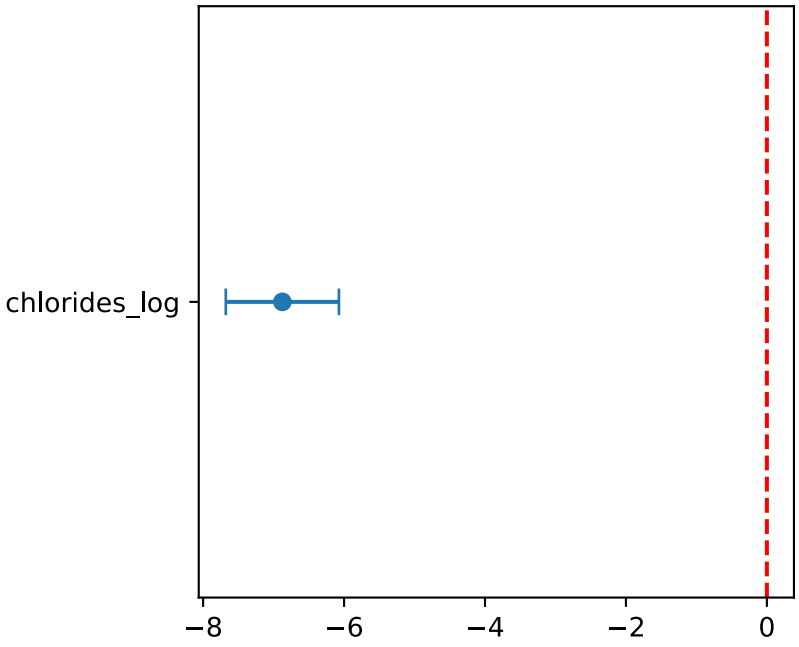
```
# Construct confidence interval data frame
df_01 = pl.DataFrame({
    'term': fit_final.conf_int().index.tolist(),
    'coef': fit_final.params.tolist(),
    'conf_low': fit_final.conf_int().loc[:, 0].tolist(),
    'conf_high': fit_final.conf_int().loc[:, 1].tolist()
})

# Plotting the confidence interval
df = df_01
plt.figure(figsize=(4, 4))
plt.errorbar(df['coef'], df['term'],
    xerr=[df['coef'] - df['conf_low'], df['conf_high'] - df['coef']],
    fmt='o',
    capsize=5,
    label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')
```



```
# Selecting just slopes
df_02 = df_01.filter(pl.col('term') != 'Intercept')
df_02 = df_02.filter(pl.col('term') != 'chlorides_log')

# Plotting the confidence interval
df = df_02
plt.figure(figsize=(4, 4))
plt.errorbar(df['coef'], df['term'],
    xerr=[df['coef'] - df['conf_low'], df['conf_high'] - df['coef']],
    fmt='o',
    capsize=5,
    label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')
```




```
# Selecting just slopes
df_03 = df_01.filter(pl.col('term') == 'chlorides_log')

# Plotting the confidence interval
df = df_03
plt.figure(figsize=(4, 4))
plt.errorbar(df['coef'], df['term'],
    xerr=[df['coef'] - df['conf_low'], df['conf_high'] - df['coef']],
    fmt='o',
    capsize=5,
    label='Estimates')
plt.axvline(0, color='red', linestyle='--', label='y=0')
```



These three graphs show our confidence intervals visually. Increased sulphates and citric acid are associated with greater quality, while lower fixed acidity and chlorides are associated with lower quality.

Optimal Wine Composition Predictions

Using our validated model, we explore different wine compositions to identify formulations that maximize predicted quality scores.

```
# Create a DataFrame with new theoretical input values
# Remember to apply log transformations to the relevant variables if your
input is on the original scale
x_new = pd.DataFrame({
    'fixed_acidity_log': [6.0],
    'citric_acid_log': [1.23],
    'residual_sugar_log': [.6],
    'chlorides_log': [0.04],
    'free_sulfur_dioxide_log': [40.0],
    'sulphates_log': [2],
    'is_red': [1]
})

# Apply log transformation to relevant columns in x_new
skewed_vars_fit_final = ['fixed_acidity_log', 'citric_acid_log',
'residual_sugar_log', 'chlorides_log', 'free_sulfur_dioxide_log',
'sulphates_log']
```

```
for var in skewed_vars_fit_final:
    if var in x_new.columns:
        x_new[var] = np.log1p(x_new[var])

# Predict the quality using the fit_final model
predicted_quality_new = fit_final.predict(x_new)

print("Predicted quality for the new theoretical data:")
display(predicted_quality_new)

# Get prediction intervals (confidence interval for the prediction of a single
new observation)
prediction_intervals =
fit_final.get_prediction(x_new).summary_frame(alpha=0.05)

print("\nPrediction intervals for the new theoretical data:")
display(prediction_intervals)
```

```
Predicted quality for the new theoretical data:
```

```
0    7.632697
dtype: float64
```

```

Prediction intervals for the new theoretical data:
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	7.632697	0.113185	7.410817	7.854577	5.977491	9.287904

```
# Create a DataFrame with new theoretical input values
# Remember to apply log transformations to the relevant variables if your
input is on the original scale
x_new = pd.DataFrame({
    'fixed_acidity_log': [4.0],
    'citric_acid_log': [1.23],
    'residual_sugar_log': [.6],
    'chlorides_log': [0.01],
    'free_sulfur_dioxide_log': [140.0],
    'sulphates_log': [2],
    'is_red': [1]
})

# Apply log transformation to relevant columns in x_new
```

```
skewed_vars_fit_final = ['fixed_acidity_log', 'citric_acid_log',
'residual_sugar_log', 'chlorides_log', 'free_sulfur_dioxide_log',
'sulphates_log']

for var in skewed_vars_fit_final:
    if var in x_new.columns:
        x_new[var] = np.log1p(x_new[var])

# Predict the quality using the fit_final model
predicted_quality_new = fit_final.predict(x_new)

print("Predicted quality for the new theoretical data:")
display(predicted_quality_new)

# Get prediction intervals (confidence interval for the prediction of a single
new observation)
prediction_intervals =
fit_final.get_prediction(x_new).summary_frame(alpha=0.05)

print("\nPrediction intervals for the new theoretical data:")
display(prediction_intervals)
```

```
Predicted quality for the new theoretical data:
```

```
0    8.136363
dtype: float64
```

```

Prediction intervals for the new theoretical data:
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	8.136363	0.138071	7.865698	8.407028	6.473913	9.798813

Above we have some predictions using our loss function. The code above applies our quadratic loss function and allows us to make predictions based on the variables inputted. We tried a bunch of combinations using the minimum and maximum of each variable's data points as constraints. We left one previous example and our best prediction in the code, as most of it was just replacing a variable with a new number and running the prediction again. The combination on the bottom was the highest prediction for quality we achieved. This code's prediction can be interpreted as:

We advise creating red wine with 4.0 g/L fixed acidity, 1.23 g/L citric acid, 0.6 g/L residual sugar, 0.01 g/L chlorides, 140 mg/L free sulfur dioxide, and 2.0 g/L sulphates. This will give a wine quality between 6.47 and 9.8.

Conclusion and Recommendations

Overall, we found that fixed acidity, citric acid, chlorides, and sulphates have the greatest impact on wine quality. These variables need to be properly monitored and controlled if wine quality is to be improved.

We advise creating red wine with 4.0 g/L fixed acidity, 1.23 g/L citric acid, 0.6 g/L residual sugar, 0.01 g/L chlorides, 140 mg/L free sulfur dioxide, and 2.0 g/L sulphates. This will give a wine quality between 6.47 and 9.8.