Notebook for EDA and Dense Neural Network for Analysis of red wine quality

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
```

Loading data

```
from google.colab import drive
drive.mount('/content/drive')

file_path = '/content/drive/MyDrive/CS4824_Final/winequality-red.csv'
```

```
Mounted at /content/drive
```

```
data = pd.read_csv(file_path)
```

Data Dictionary

Response variable: `quality`

- Description: Sensory score given by wine experts.
- Impact: Target variable; represents overall acceptance.

Explantory variables

`fixed acidity`:

- Description: Non-volatile acids that do not evaporate during fermentation.
- Impact: Contributes to the overall acidity and taste profile of the wine.

`volatile acidity`:

- Description: Volatile acids that can evaporate; primarily acetic acid.
- Impact: High levels may lead to an unpleasant vinegar taste.

`citric acid`:

- Description: A natural component of grapes; adds freshness and flavor.
- Impact: Enhances the sensory profile when in balanced amounts.

`residual sugar`:

- Description: Sugar remaining after fermentation; not converted to alcohol.
- Impact: Influences sweetness; higher levels make the wine sweeter.

`chlorides`:

- Description: Salt content in the wine.
- Impact: Excessive chlorides can result in a salty taste.

`free sulfur dioxide`:

- Description: SO_2 in the free form; acts as an antioxidant and antimicrobial agent.
- Impact: Protects wine from spoilage; too much can affect taste.

`total sulfur dioxide`:

- Description: Sum of free and bound sulfur dioxide.
- Impact: High levels can cause sensory defects and health concerns.

`density`:

- Description: Mass per unit volume; correlates with alcohol and sugar content.
- Impact: Used to monitor fermentation and quality control.

`pH`:

- Description: Scale of acidity (lower pH = more acidic).
- Impact: Affects color, stability, and taste.

`sulphates`:

- Description: Additive used to increase shelf life by preventing oxidation.
- Impact: Enhances flavor and longevity; excessive amounts can be detrimental.

`alcohol`:

- Description: Ethanol content in the wine.
- Impact: Affects the body, warmth, and caloric content.

Work planned for Oct 21st to Oct 31st:

focus on **exploratory data analysis (EDA)** using Python visualization libraries such as Matplotlib and Seaborn to examine the dataset through histograms, scatter plots, box plots, pair plots, and heatmaps.

Goal:

uncover key patterns, detect outliers, identify potential predictors, and gain initial insights into how the chemical properties of the wines relate to their quality ratings link text .

We will proceed once varaibles fit the assumptions of our model.

EDA - Process of examining distributions and preparing data for modeling

Examining individual distributions

Head of dataframe

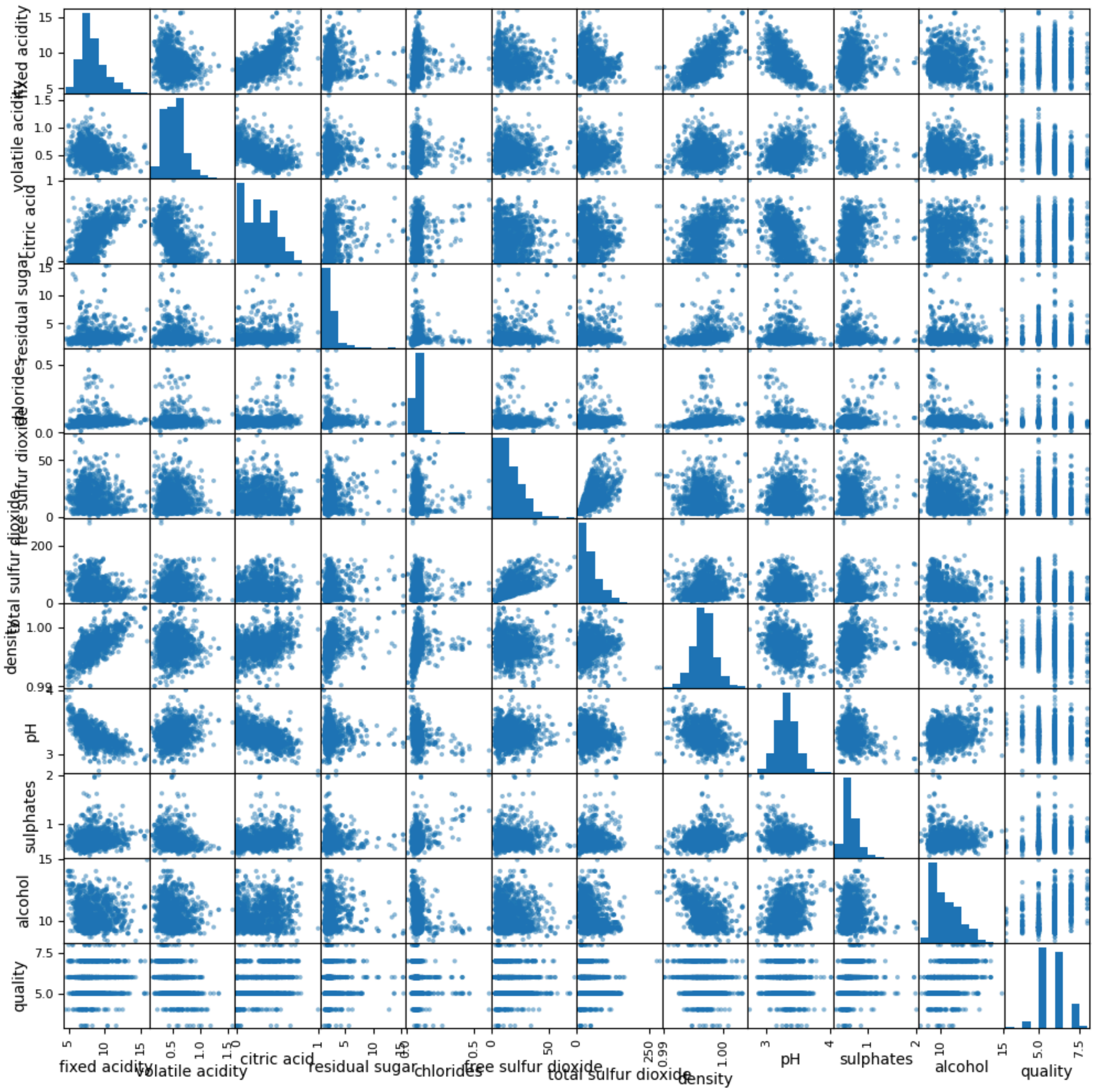
```
data[:20]
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5
10	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	0.54	9.2	5
11	7.5	0.500	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5
12	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	0.52	9.9	5
13	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	1.56	9.1	5
14	8.9	0.620	0.18	3.8	0.176	52.0	145.0	0.9986	3.16	0.88	9.2	5
15	8.9	0.620	0.19	3.9	0.170	51.0	148.0	0.9986	3.17	0.93	9.2	5
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.9969	3.30	0.75	10.5	7
17	8.1	0.560	0.28	1.7	0.368	16.0	56.0	0.9968	3.11	1.28	9.3	5
18	7.4	0.590	0.08	4.4	0.086	6.0	29.0	0.9974	3.38	0.50	9.0	4

Scatterplot matrix for all variables

```
pd.plotting.scatter_matrix(data, figsize=(12,12))[-1]
```

```
array([<Axes: xlabel='fixed acidity', ylabel='quality'>,
       <Axes: xlabel='volatile acidity', ylabel='quality'>,
       <Axes: xlabel='citric acid', ylabel='quality'>,
       <Axes: xlabel='residual sugar', ylabel='quality'>,
       <Axes: xlabel='chlorides', ylabel='quality'>,
       <Axes: xlabel='free sulfur dioxide', ylabel='quality'>,
       <Axes: xlabel='total sulfur dioxide', ylabel='quality'>,
       <Axes: xlabel='density', ylabel='quality'>,
       <Axes: xlabel='pH', ylabel='quality'>,
       <Axes: xlabel='sulphates', ylabel='quality'>,
       <Axes: xlabel='alcohol', ylabel='quality'>,
       <Axes: xlabel='quality', ylabel='quality'>], dtype=object)
```



Individual distributions

We can use a normality test from the *sci.py* package to determine if any of our data are normally distributed

```
for item in data:
  print(str(item), stats.normaltest(data[item]))
```

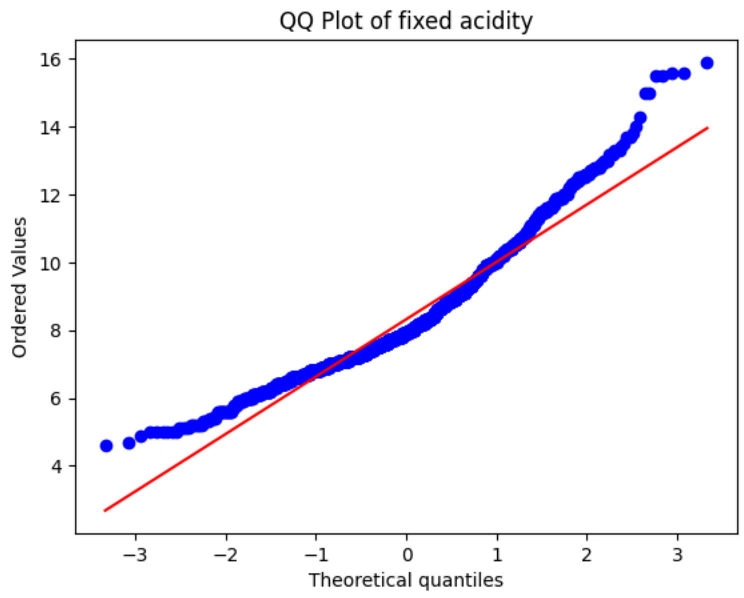
```
fixed acidity NormaltestResult(statistic=224.53087840457746, pvalue=1.7528277735470436e-49)
volatile acidity NormaltestResult(statistic=143.41934355982863, pvalue=7.192589039756591e-32)
citric acid NormaltestResult(statistic=152.039214793795, pvalue=9.662822259281018e-34)
residual sugar NormaltestResult(statistic=1520.3239698236891, pvalue=0.0)
chlorides NormaltestResult(statistic=1783.1059225626427, pvalue=0.0)
free sulfur dioxide NormaltestResult(statistic=342.2591484251237, pvalue=4.779365332171615e-75)
total sulfur dioxide NormaltestResult(statistic=487.42725648953456, pvalue=1.4338908343436201e-106)
density NormaltestResult(statistic=30.70774994095191, pvalue=2.1473202738102222e-07)
pH NormaltestResult(statistic=33.684697471483915, pvalue=4.8468645347727716e-08)
sulphates NormaltestResult(statistic=906.8944479227036, pvalue=1.1759065222978855e-197)
alcohol NormaltestResult(statistic=154.17806951912513, pvalue=3.316328847318596e-34)
quality NormaltestResult(statistic=17.262400816355534, pvalue=0.00017845030333855057)
```

p-values are all very small, meaning that none of our variables are normally distributed, the variable closest to a normal distribtion according to our test is quality, with $p = 0.00018$

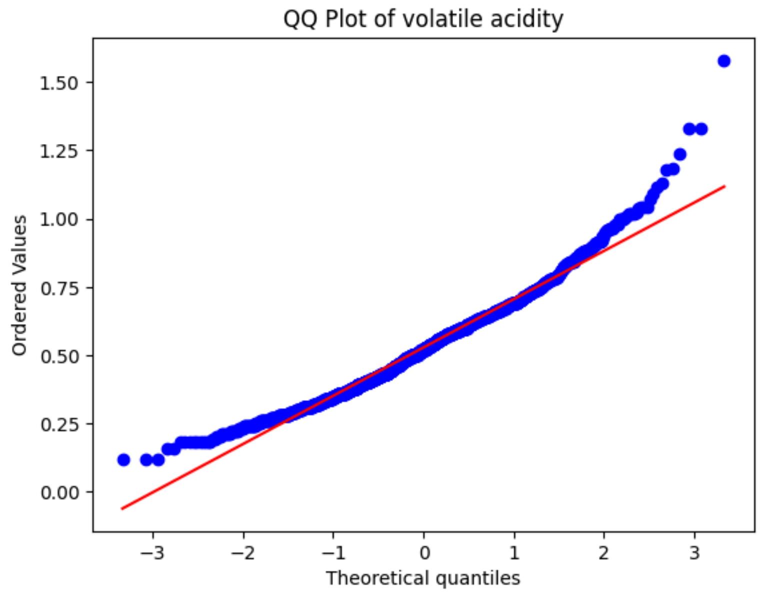
We can also use Q-Q plots to get a visual representation for normality

```
#draws a QQ-plot and perfromes shapiro normality test. Data is a list, title is a string
def test_normality(data, title):
  from scipy.stats import shapiro
  stats.probplot(x = data, dist='norm', plot = plt)
  plt.title('QQ Plot of ' + str(title))
  plt.show()
  stat, p = shapiro(data)
  print(f"Shapiro-Wilk Test Statistic for {title}: {stat}, p-value: {p}")
```

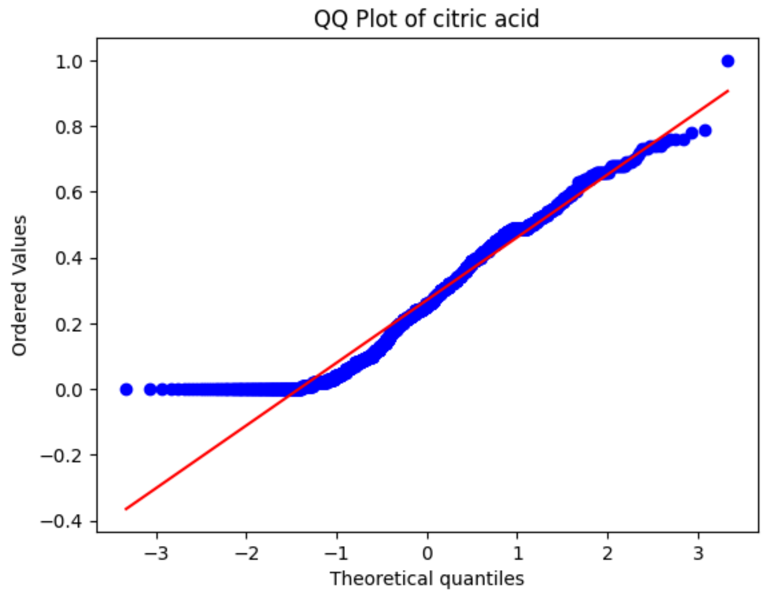
```
for item in data:
  if item != 'quality':
    test_normality(data[item], item)
```



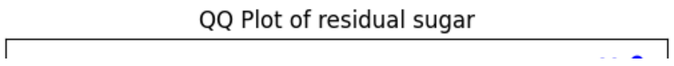
```
Shapiro-Wilk Test Statistic for fixed acidity: 0.9420297903867135, p-value: 1.525011710791387e-24
```

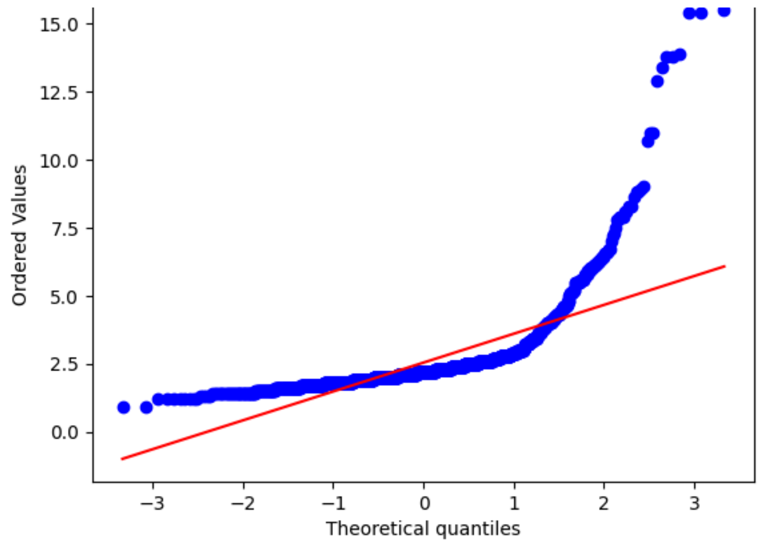


```
Shapiro-Wilk Test Statistic for volatile acidity: 0.9743368805536368, p-value: 2.692934735712727e-16
```



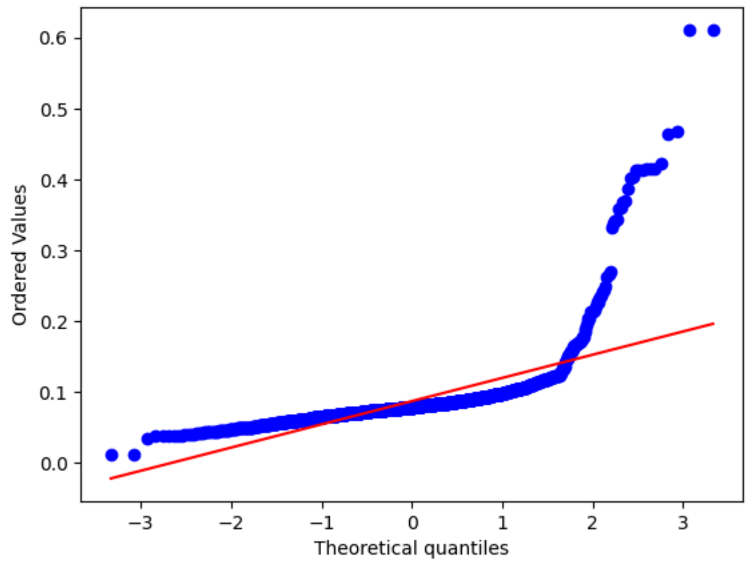
```
Shapiro-Wilk Test Statistic for citric acid: 0.9552919890668837, p-value: 1.0219317829705018e-21
```





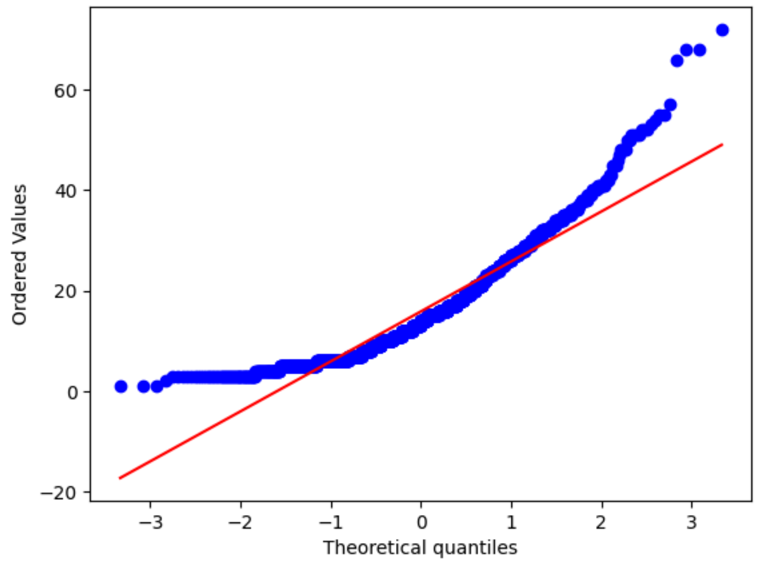
```
Shapiro-Wilk Test Statistic for residual sugar: 0.5660771057163958, p-value: 1.0201616453237868e-52
```

QQ Plot of chloprides



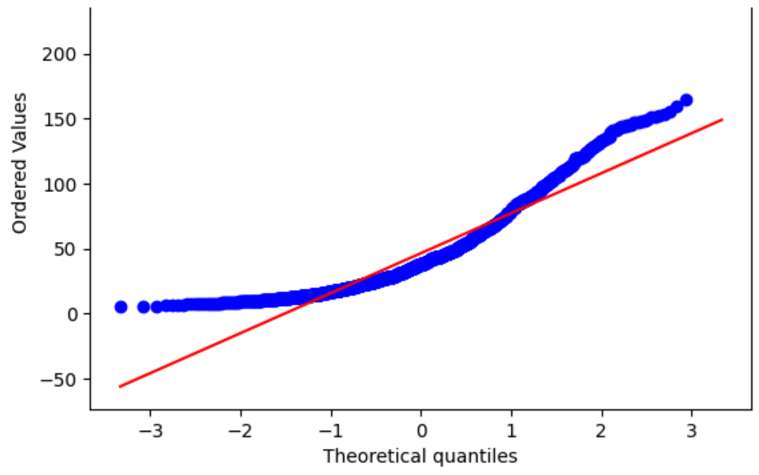
```
Shapiro-Wilk Test Statistic for chlorides: 0.48424655122518334, p-value: 1.1790556953147118e-55
```

QQ Plot of free sulfur dioxide

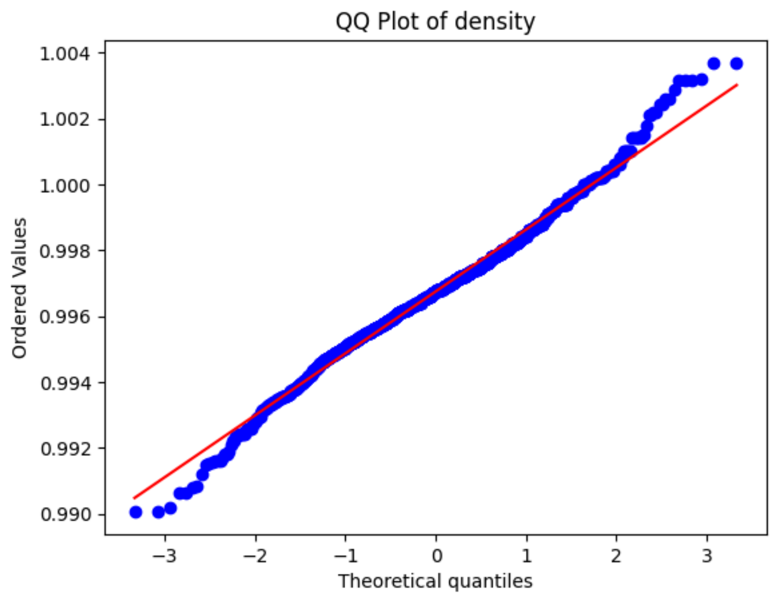


```
Shapiro-Wilk Test Statistic for free sulfur dioxide: 0.9018394916138583, p-value: 7.694596687816645e-31
```

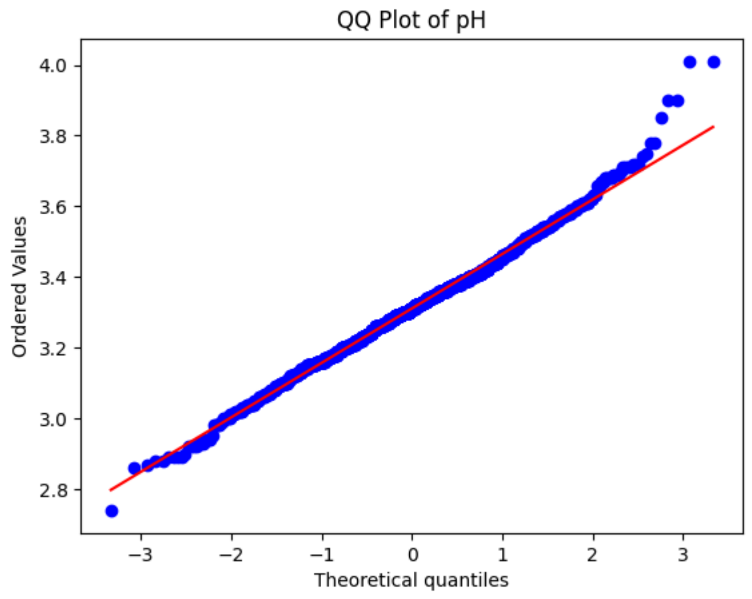
QQ Plot of total sulfur dioxide



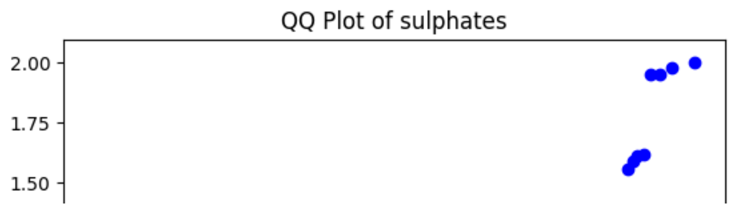
```
Shapiro-Wilk Test Statistic for total sulfur dioxide: 0.8732245604736051, p-value: 3.5734514102654424e-34
```

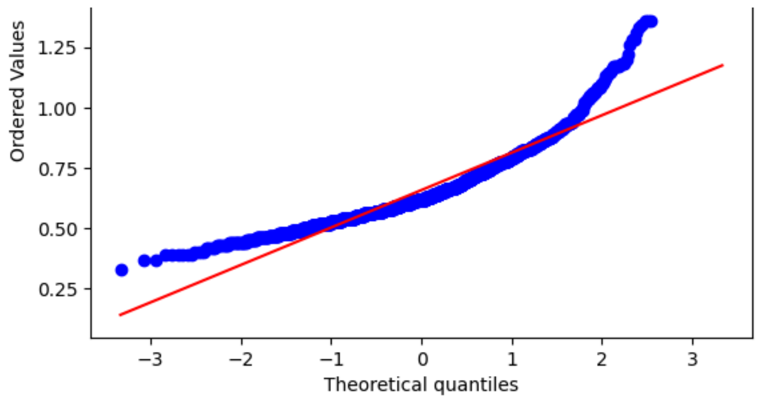


```
Shapiro-Wilk Test Statistic for density: 0.9908655166510911, p-value: 1.936052131352189e-08
```

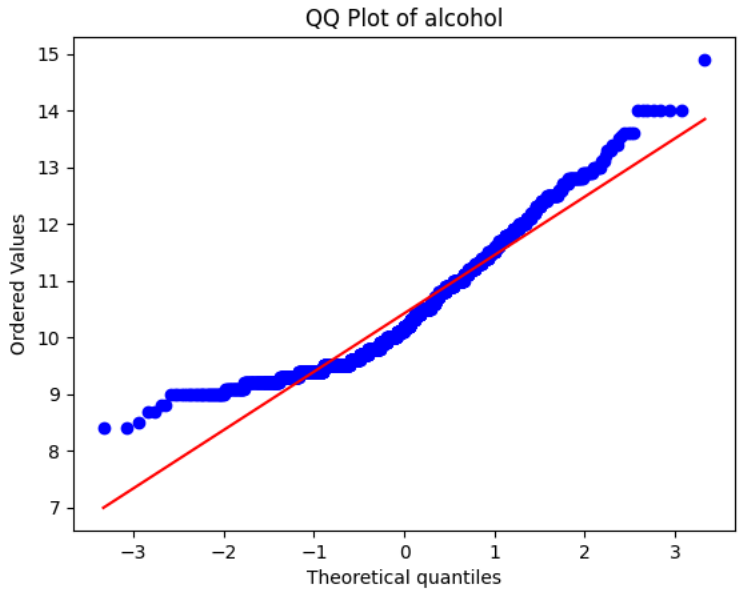


```
Shapiro-Wilk Test Statistic for pH: 0.9934862934498192, p-value: 1.7122367757609613e-06
```





```
Shapiro-Wilk Test Statistic for sulphates: 0.8330437683911954, p-value: 5.823139712583187e-38
```



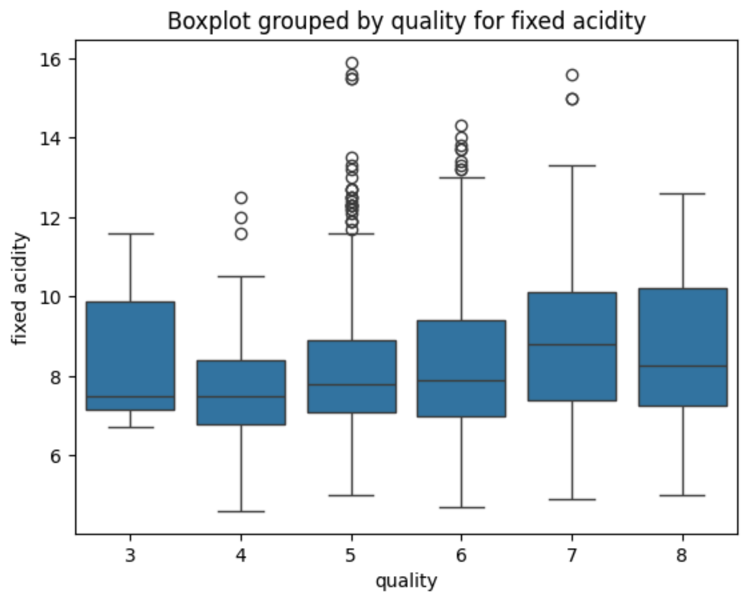
```
Shapiro-Wilk Test Statistic for alcohol: 0.9288390813054377, p-value: 6.644056905730039e-27
```

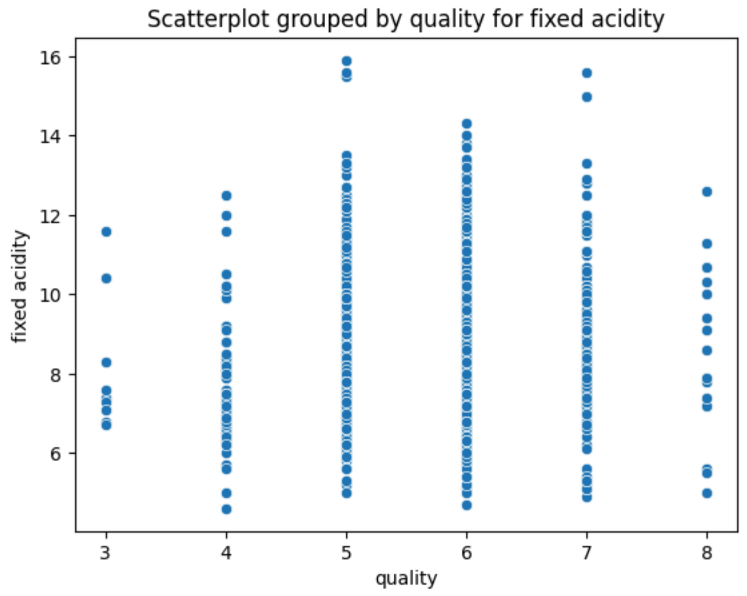
This gives us a better sense of how normally distributed our data is. It looks like most of the data are normal, and just deviate towards the tails. However, there are some variables, namely **`residual sugar`**, **`chlorides`**, **`free sulfur dioxide`**, **`total sulfur dioxide`**, **`sulfates`**, **`alcohol`** that should be modified to fit a normal distribution.

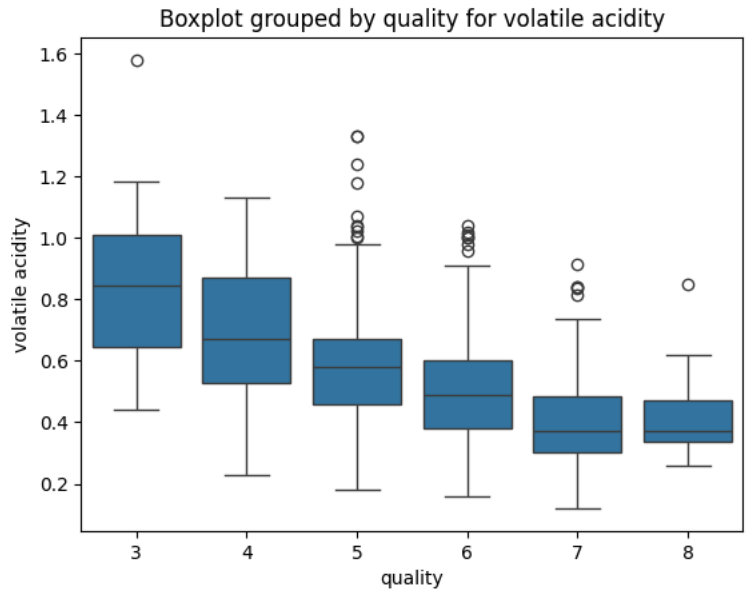
Boxplots - examining outliers

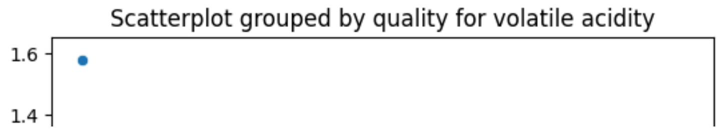
We can create boxplots for each variable that effects our response to look for outliers, done so below

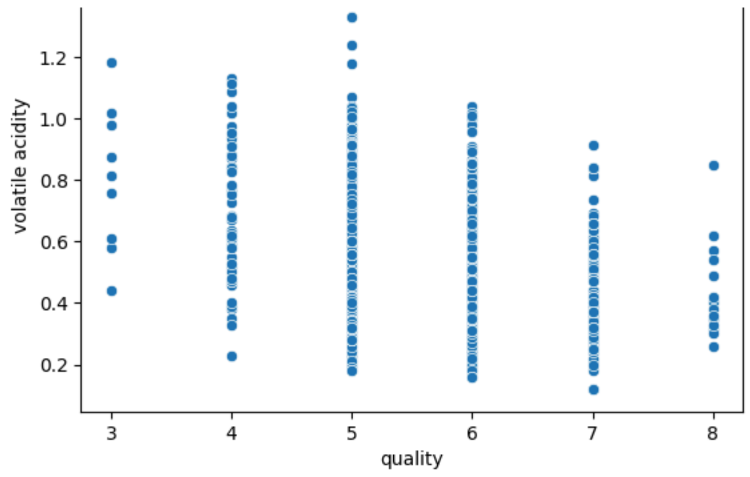
```
def draw_boxplot(df, item):
  sns.boxplot(x = df['quality'], y = df[item])
  plt.title('Boxplot grouped by quality for ' + str(item))
  plt.show()

def draw_scatterplot(df, item):
  sns.scatterplot(x = df['quality'], y = df[item])
  plt.title('Scatterplot grouped by quality for ' + str(item))
  plt.show()


for item in data:
  if item != 'quality':
    draw_boxplot(data, item)
    draw_scatterplot(data, item)
```

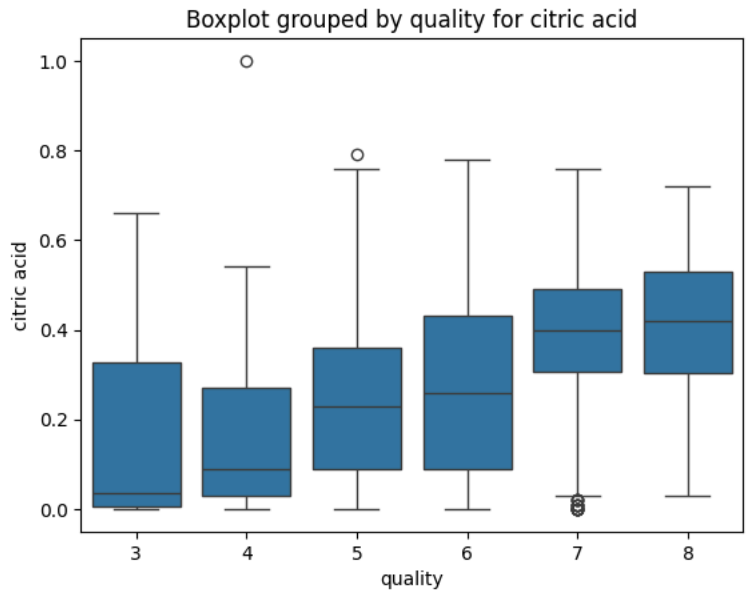

Boxplot grouped by quality for fixed acidity
fixed acidity
quality

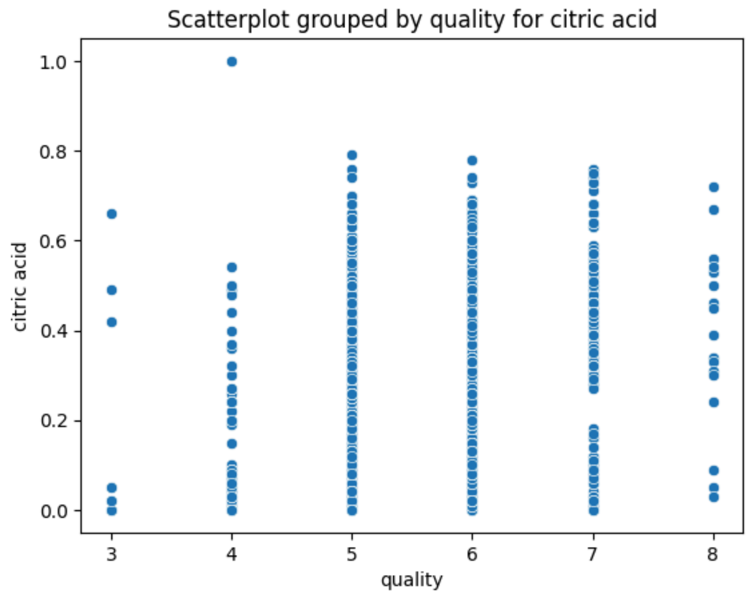

Scatterplot grouped by quality for fixed acidity
fixed acidity
quality

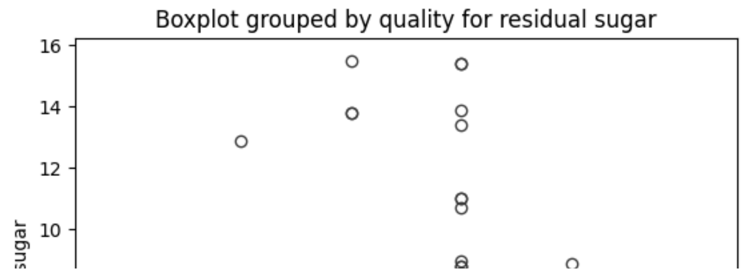

Boxplot grouped by quality for volatile acidity
volatile acidity
quality

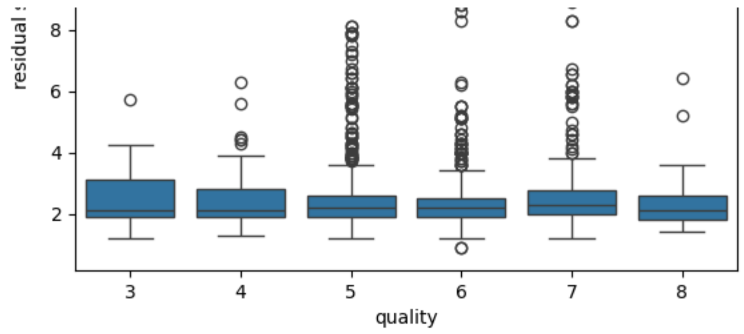

Scatterplot grouped by quality for volatile acidity


volatile acidity
quality

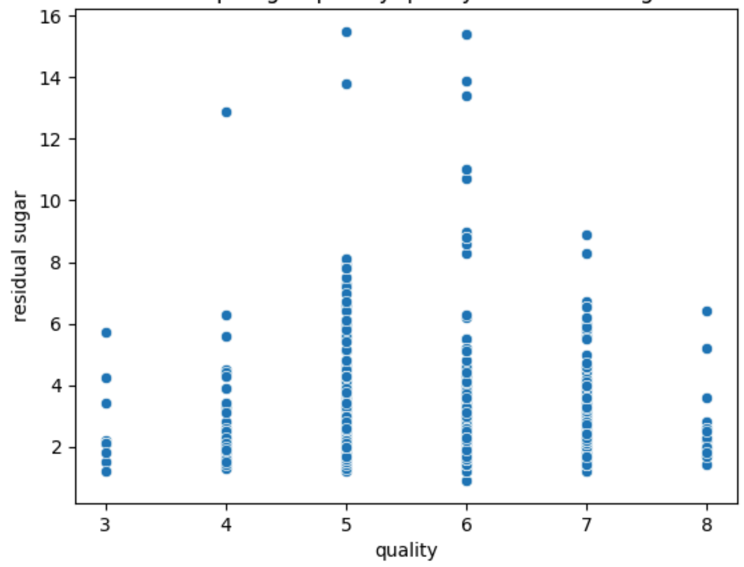

Boxplot grouped by quality for citric acid
citric acid
quality


Scatterplot grouped by quality for citric acid
citric acid
quality

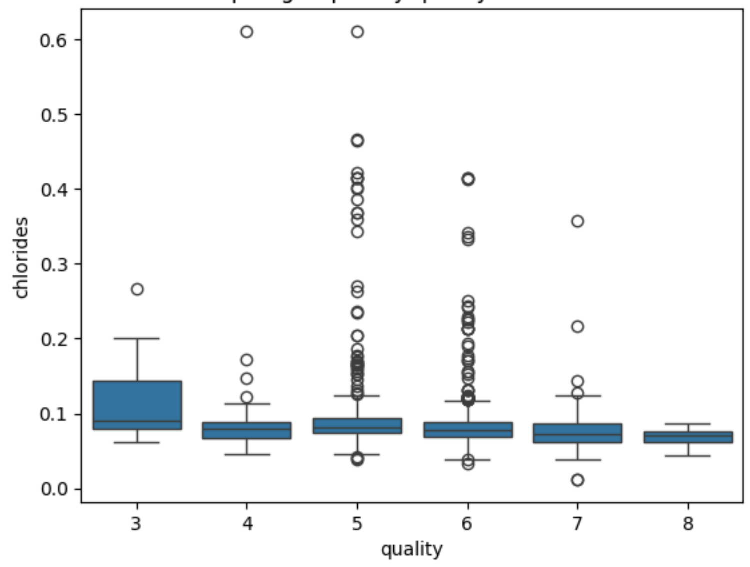

Boxplot grouped by quality for residual sugar
sugar



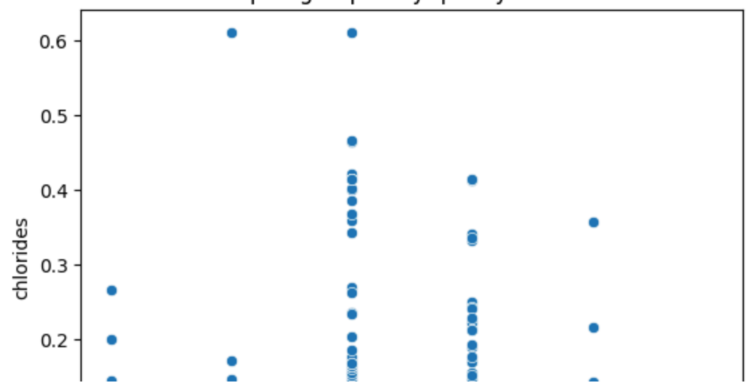
Scatterplot grouped by quality for residual sugar

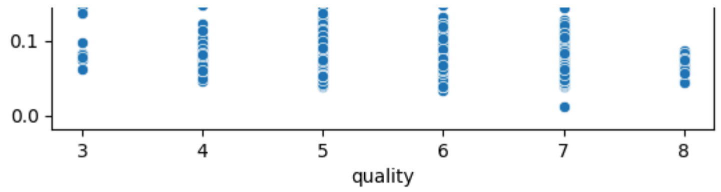


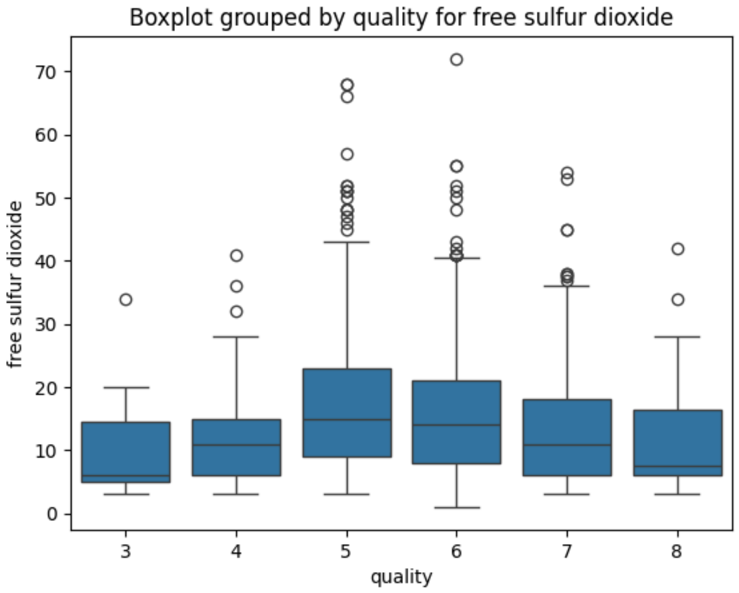
Boxplot grouped by quality for chlorides

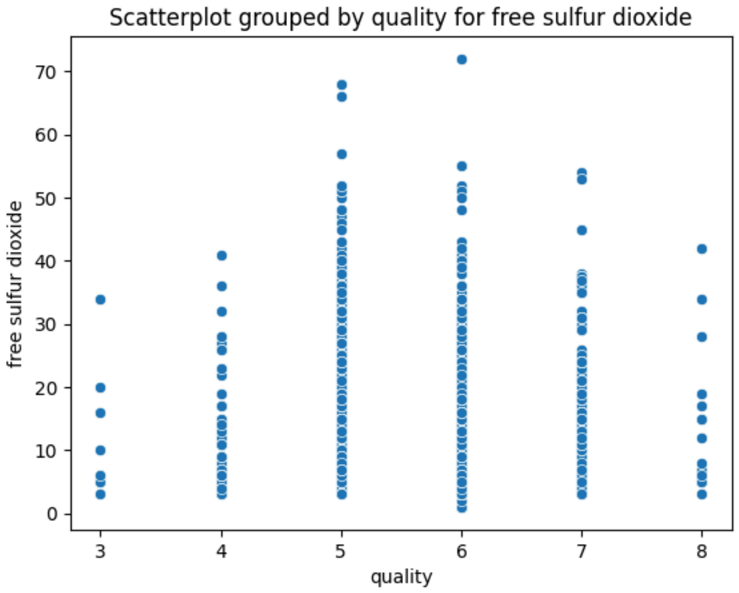


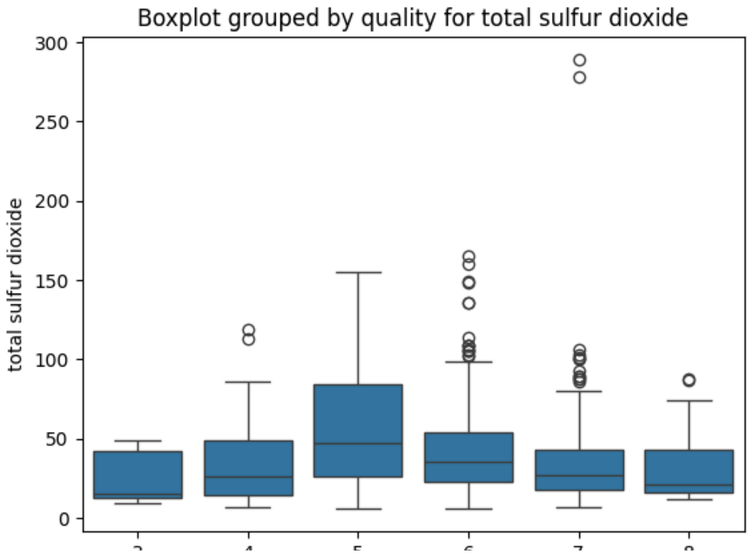
Scatterplot grouped by quality for chlorides

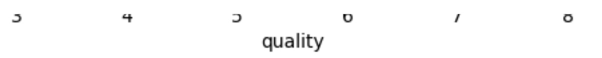


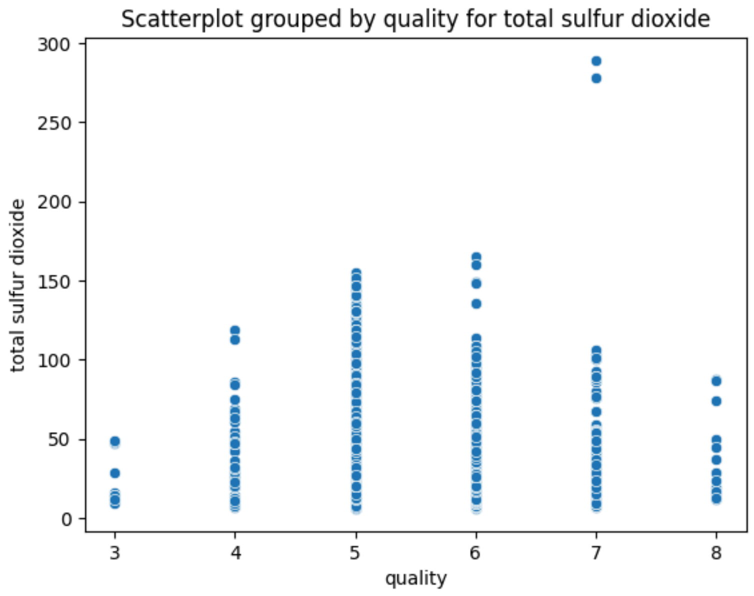

0.1
0.0
3
4
5
6
7
8
quality

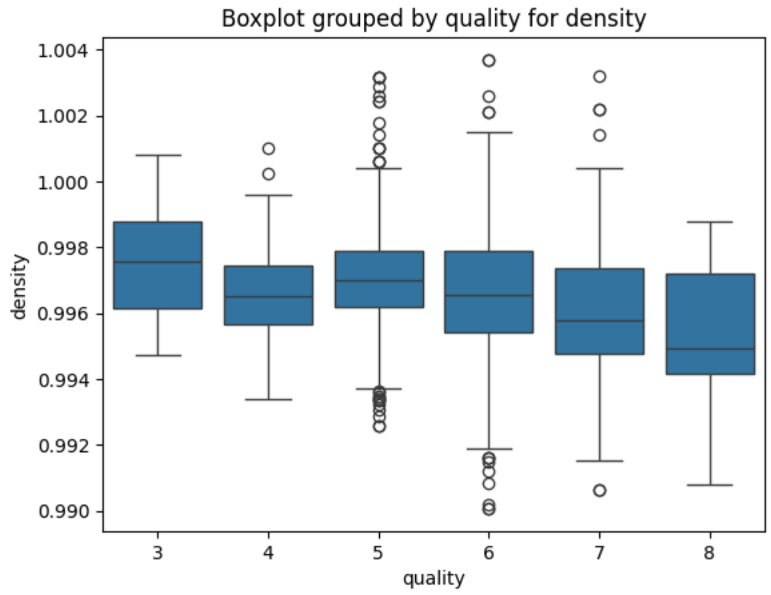

Boxplot grouped by quality for free sulfur dioxide
free sulfur dioxide
0
10
20
30
40
50
60
70
3
4
5
6
7
8
quality

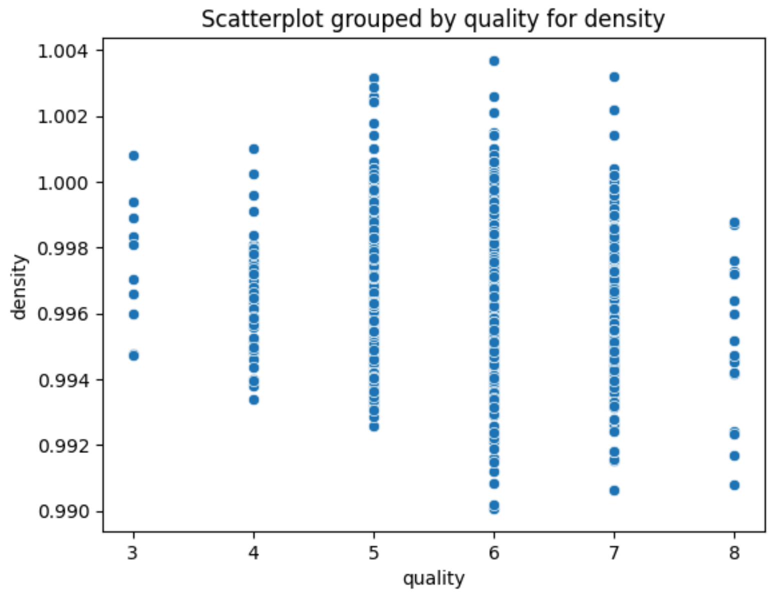

Scatterplot grouped by quality for free sulfur dioxide
free sulfur dioxide
0
10
20
30
40
50
60
70
3
4
5
6
7
8
quality

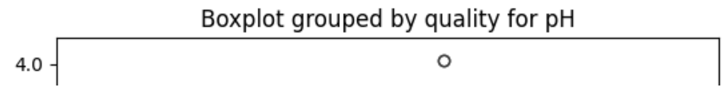

Boxplot grouped by quality for total sulfur dioxide
total sulfur dioxide
0
50
100
150
200
250
300

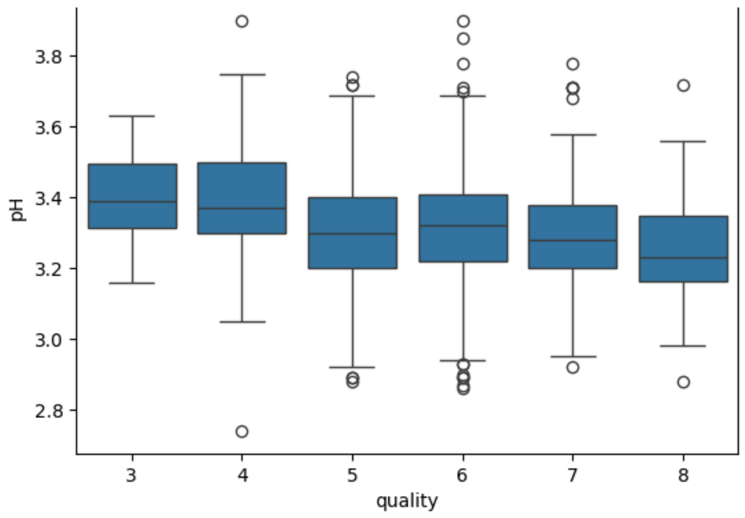

3
4
5
6
7
8
quality

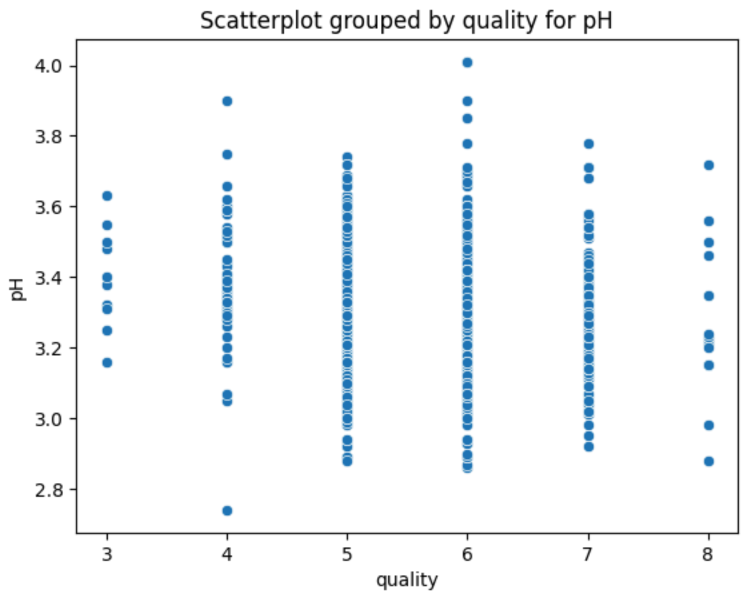

Scatterplot grouped by quality for total sulfur dioxide
total sulfur dioxide
300
250
200
150
100
50
0
3
4
5
6
7
8
quality

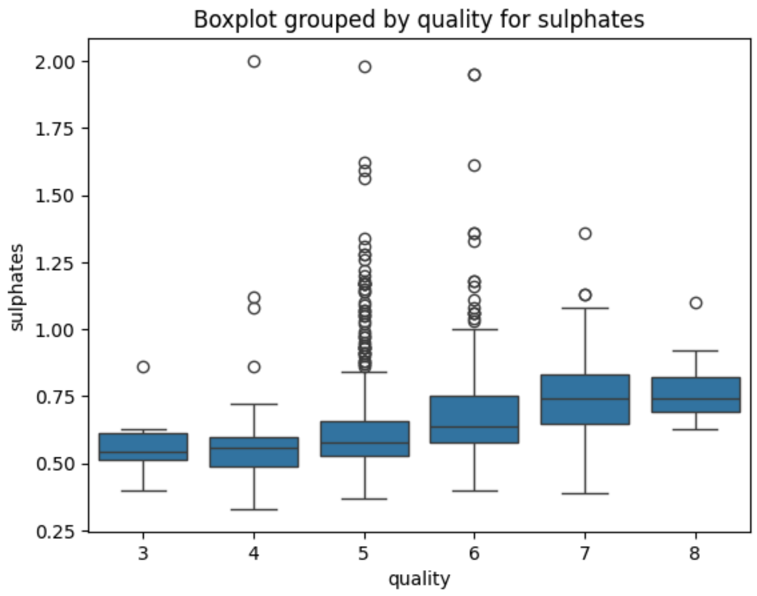

Boxplot grouped by quality for density
density
1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
3
4
5
6
7
8
quality

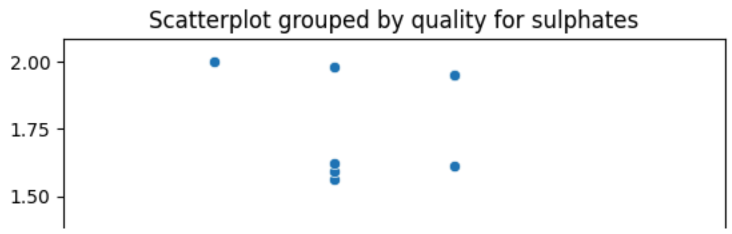

Scatterplot grouped by quality for density
density
1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
3
4
5
6
7
8
quality

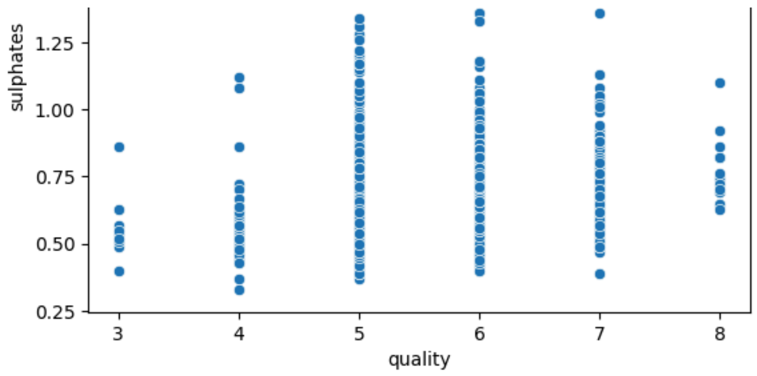

Boxplot grouped by quality for pH
4.0

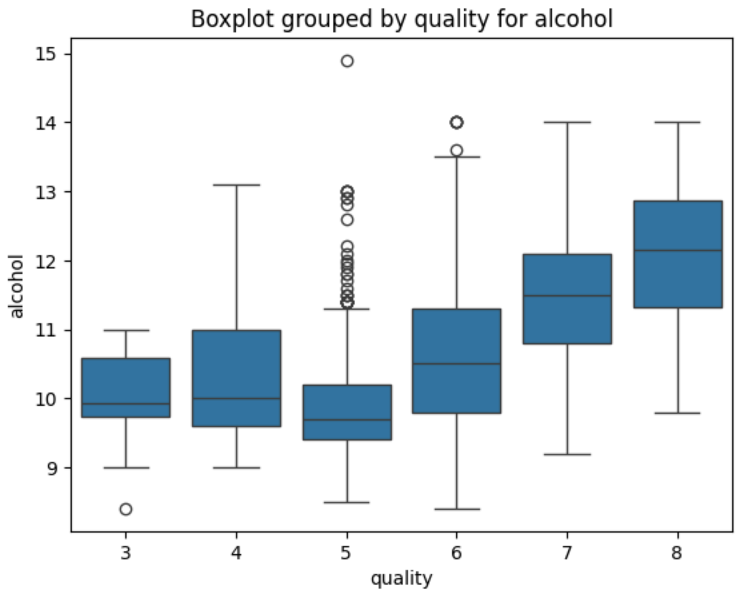

pH
quality
3.8
3.6
3.4
3.2
3.0
2.8
3
4
5
6
7
8

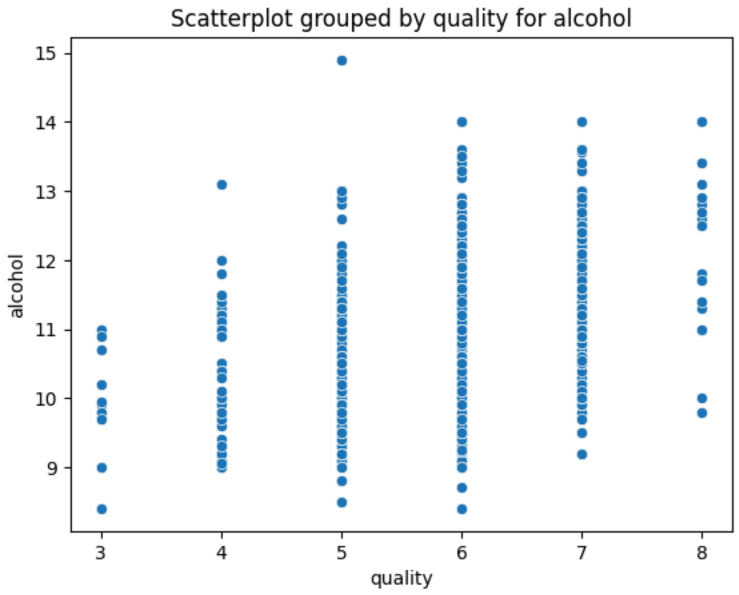

Scatterplot grouped by quality for pH
pH
quality
4.0
3.8
3.6
3.4
3.2
3.0
2.8
3
4
5
6
7
8


Boxplot grouped by quality for sulphates
sulphates
quality
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
3
4
5
6
7
8


Scatterplot grouped by quality for sulphates
2.00
1.75
1.50


sulphates
1.25
1.00
0.75
0.50
0.25
3
4
5
6
7
8
quality


Boxplot grouped by quality for alcohol
15
14
13
12
11
10
9
alcohol
3
4
5
6
7
8
quality


Scatterplot grouped by quality for alcohol
15
14
13
12
11
10
9
alcohol
3
4
5
6
7
8
quality

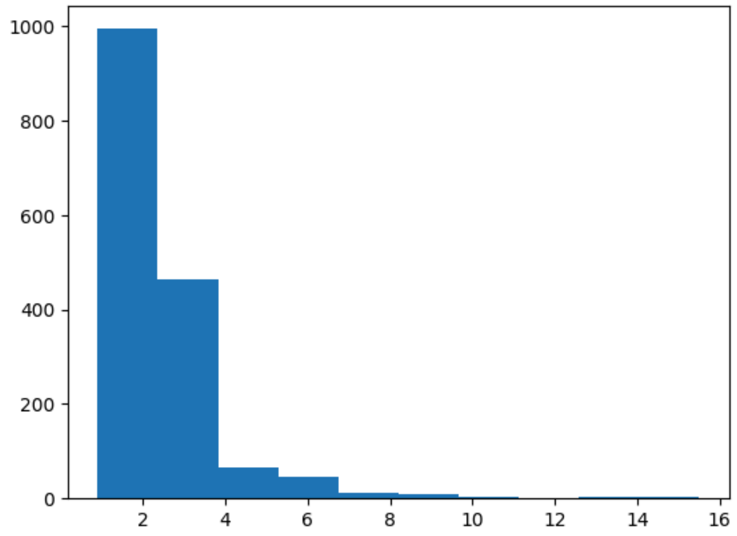
Data Transformation

Imideately, we can tell that we need to modify variables with a heavy right skew without looking at other visualizations. These variables are `residual sugar`, `chlorides`, `free sulfur dioxide`, `total sulfur dioxide`, and `alcohol`, which lines up pretty closely for what we determined should be changed when we created the Q-Q plots

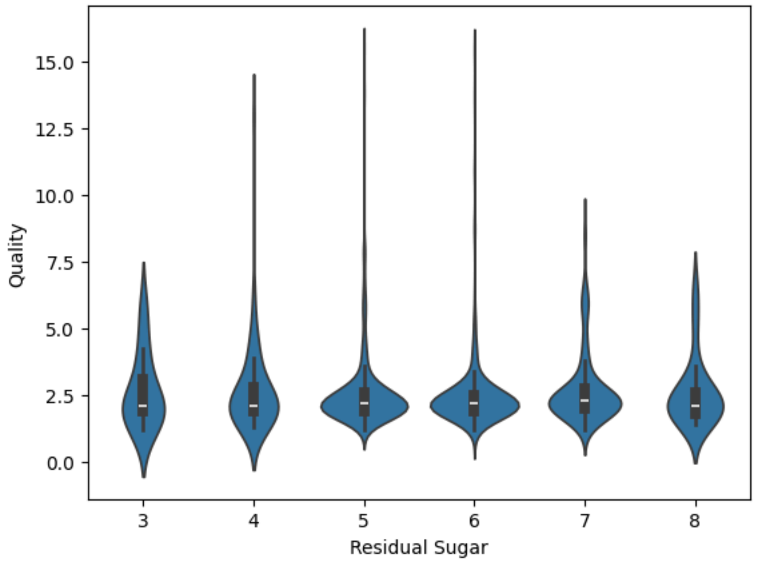
`Residual Sugar`

Below is what the distribution of `residual sugar` looks like, and a violin plot for how it relates to quality.

```
plt.hist(data['residual sugar'])
plt.show()

sns.violinplot(y = data['residual sugar'], x = 'quality', data = data)
plt.xlabel('Residual Sugar')
plt.ylabel('Quality')
```



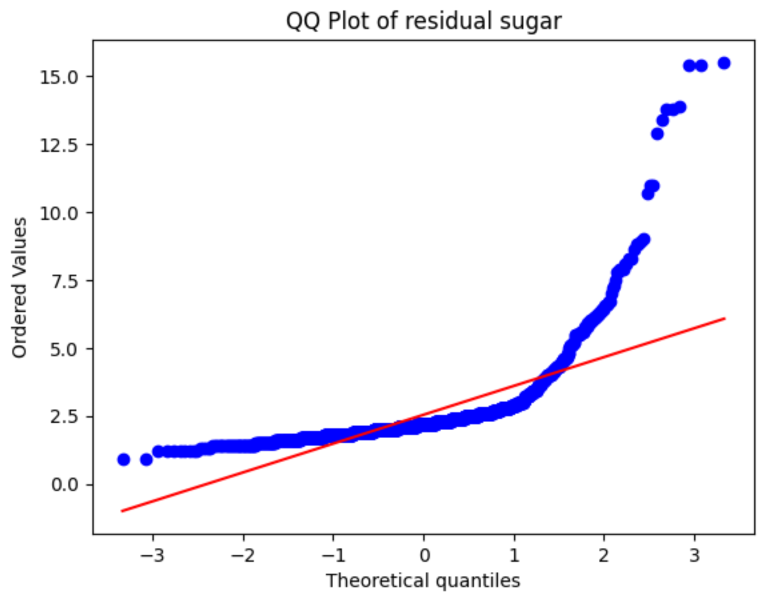
```
Text(0, 0.5, 'Quality')
```



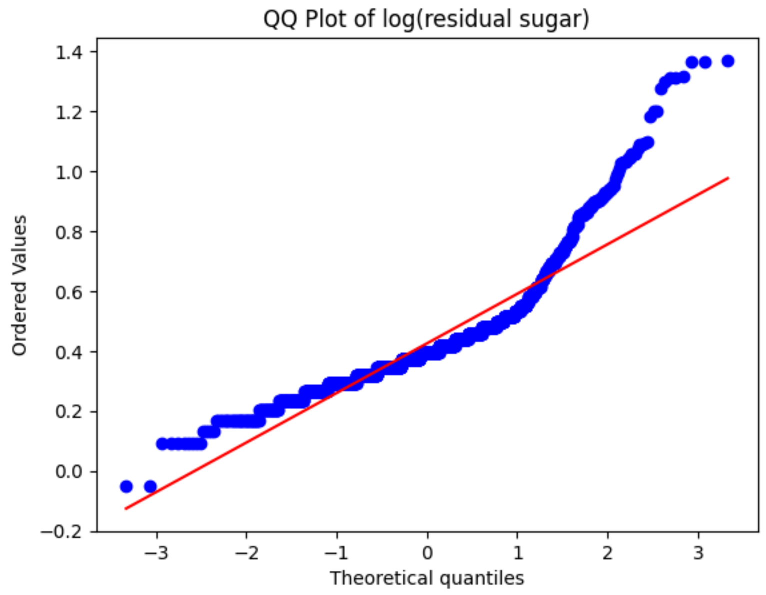
Since our data is right-skew, a log-sqrt transformation will help us shift this to a normal distribution. We can visualize the change by plotting the QQ-Plots for the original and modified data, side-by-side, where we can see we were able to wrangle in some of the right-skew-ness

```
item = 'residual sugar'

vals = np.log(np.sqrt(data[item]))
```

```
test_normality(data[item], item)
test_normality(np.log(np.sqrt(data['residual sugar'])), 'log(residual sugar)')
```



```
Shapiro-Wilk Test Statistic for residual sugar: 0.5660771057163958, p-value: 1.0201616453237868e-52
```



```
Shapiro-Wilk Test Statistic for log(residual sugar): 0.8550735645340175, p-value: 5.4710630073218684e-36
```

because we exhibit a positive change for our data's distribution with this change, we will apply the following transformation to our code in-order to train our model.

Given $\vec{S}$ is the `residual sugar` for all samples, we will take

$$\vec{S} \leftarrow \log(\vec{S})$$

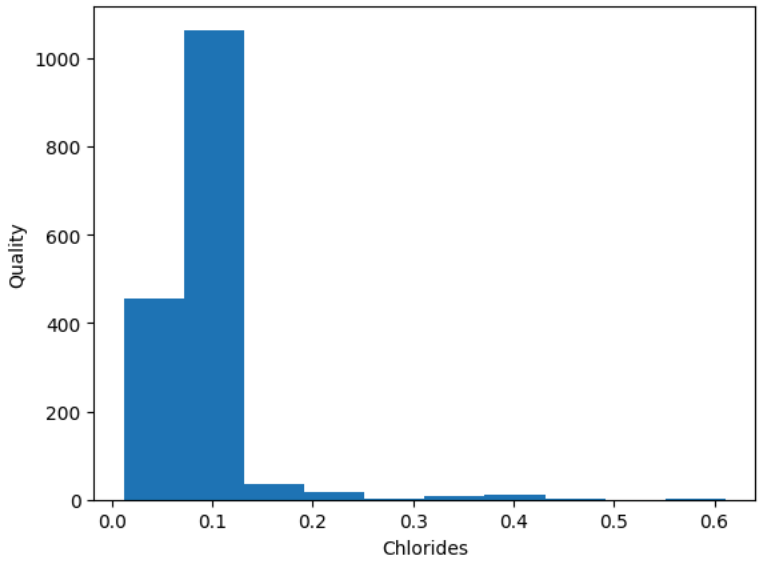
```
residual_sugar_data = data['residual sugar']
data['residual sugar'] = np.log(residual_sugar_data)
```

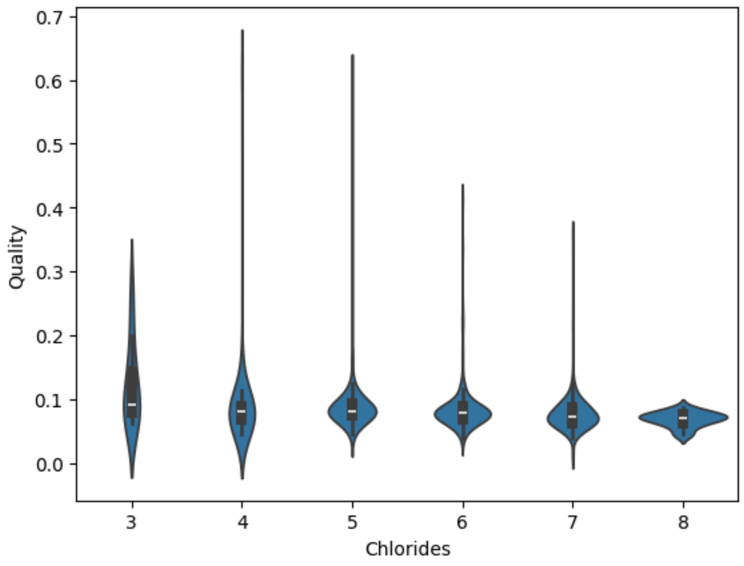
chlorides

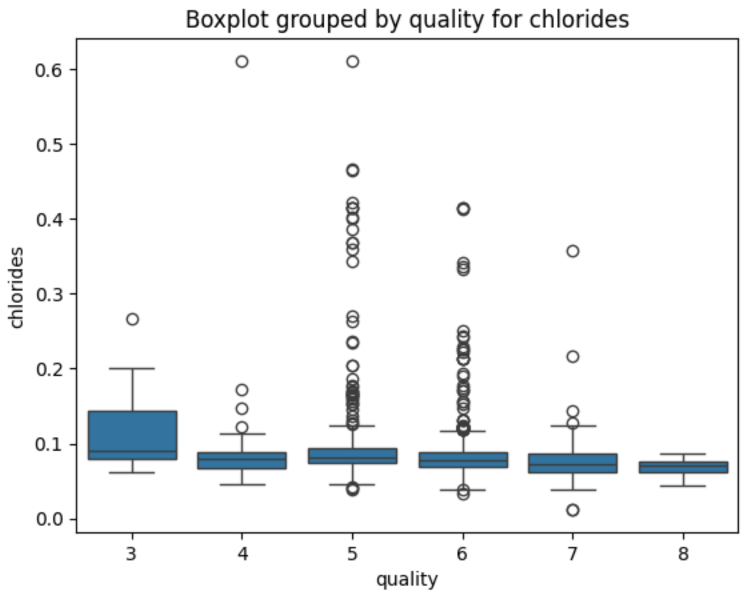
Below is what the distribution of `chlorides` looks like, and a violin plot for how it relates to quality.

Double-click (or enter) to edit

```
plt.hist(data['chlorides'])
plt.xlabel('Chlorides')
plt.ylabel('Quality')
plt.show()

sns.violinplot(y = data['chlorides'], x = 'quality', data = data)
plt.xlabel('Chlorides')
plt.ylabel('Quality')
plt.show()

draw_boxplot(data, 'chlorides')
```

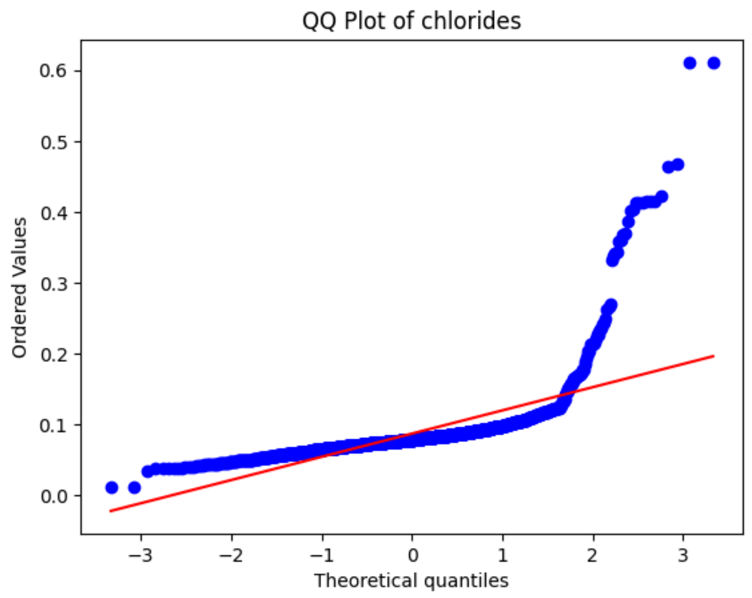




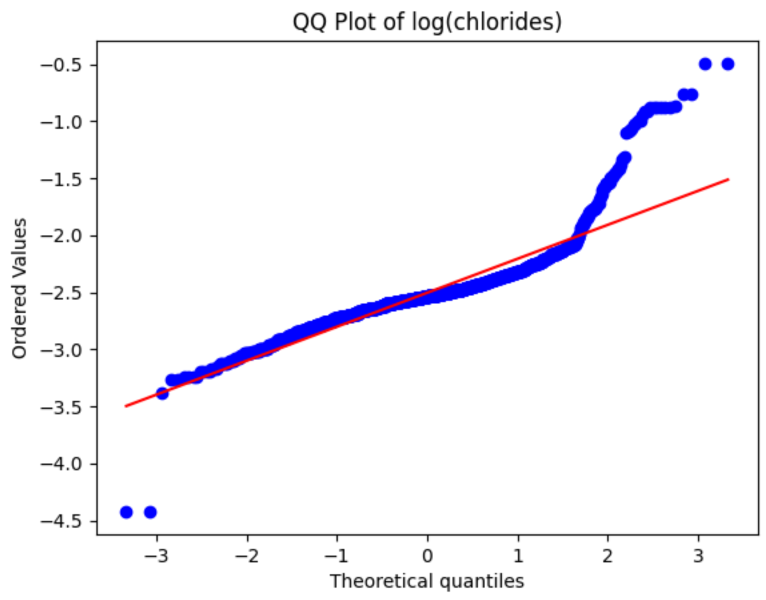


Fixed Acidity is relatively normal, with a slight right skew. Lets see how using log, sqrt, and boxcox (for observations with fixed acidity> 0) transformations change the shape of the QQ-plot and results of normality test.

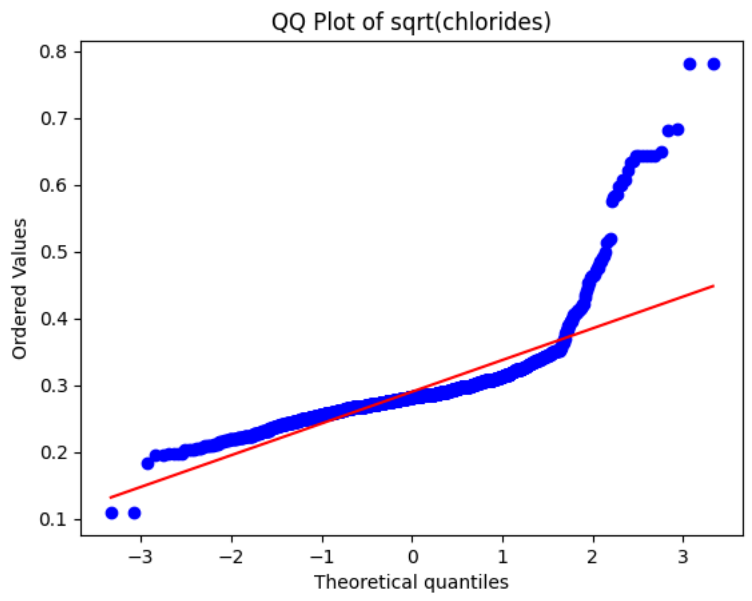
```
from scipy.stats import boxcox

#draws a QQ-plot and perfromes shapiro normality test. Data is a list, title is a string
def test_normality(data, title):
  from scipy.stats import boxcox
  from scipy.stats import shapiro
  stats.probplot(x = data, dist='norm', plot = plt)
  plt.title('QQ Plot of ' + str(title))
  plt.show()
  stat, p = shapiro(data)
  print(f"Shapiro-Wilk Test Statistic for {title}: {stat}, p-value: {p}")


chlorides_boxcox, chlorides_lambda = boxcox(data['chlorides'])


test_normality(data['chlorides'], 'chlorides')
test_normality(np.log(data['chlorides']), 'log(chlorides)')
test_normality(np.sqrt(data['chlorides']), 'sqrt(chlorides)')
test_normality(chlorides_boxcox, 'boxcox(chlorides)')
```



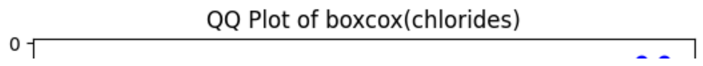
```
Shapiro-Wilk Test Statistic for chlorides: 0.48424655122518334, p-value: 1.1790556953147118e-55
```

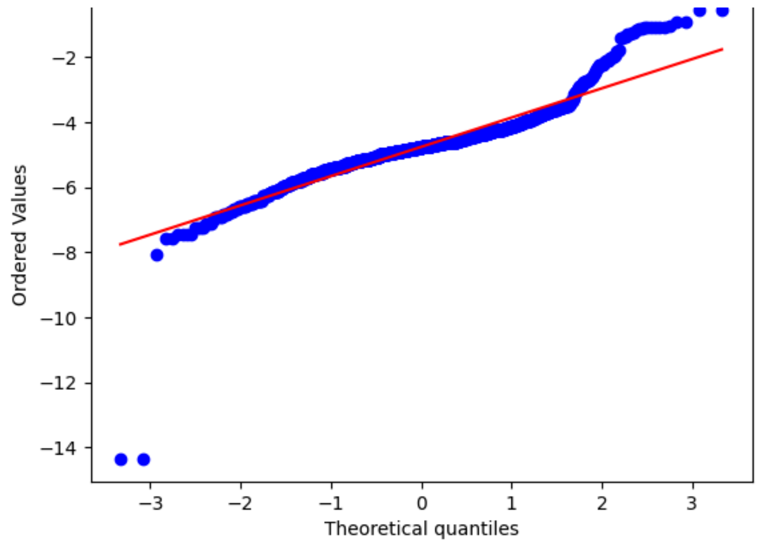


```
Shapiro-Wilk Test Statistic for log(chlorides): 0.8283642549328323, p-value: 2.3622721512330157e-38
```



```
Shapiro-Wilk Test Statistic for sqrt(chlorides): 0.6723604020744909, p-value: 4.071545937322437e-48
```





```
Shapiro-Wilk Test Statistic for boxcox(chlorides): 0.8716791986679291, p-value: 2.459048953286268e-34
```

Based on the results of our testing , it seems that boxcox did the best job of normalizing our data. We will change chlorides to boxcox(chlorides), which takes the following formula:

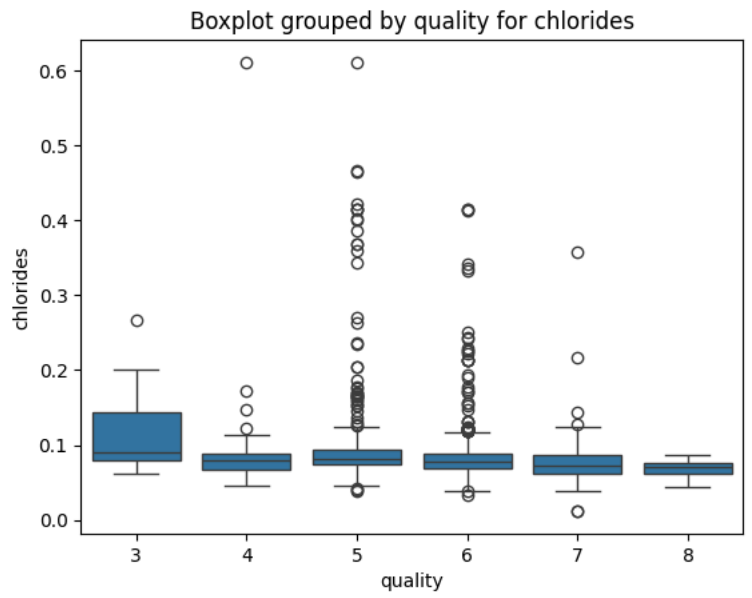
$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \\ \ln(y_i), & \text{if } \lambda = 0 \end{cases}$$

We make this change in the next cell, along with storing the original value for `data['chlorides]` in `chlorides_data` for reference.

We can also observe two extreme outliers from the QQ-plot, with values around -14, while the rest of our values fall between $(-8, 0)$. We will also remove these.

```
chlorides_data = data['chlorides']
#data['chlorides'] = chlorides_boxcox

data[(data['chlorides'] < -10)]
```

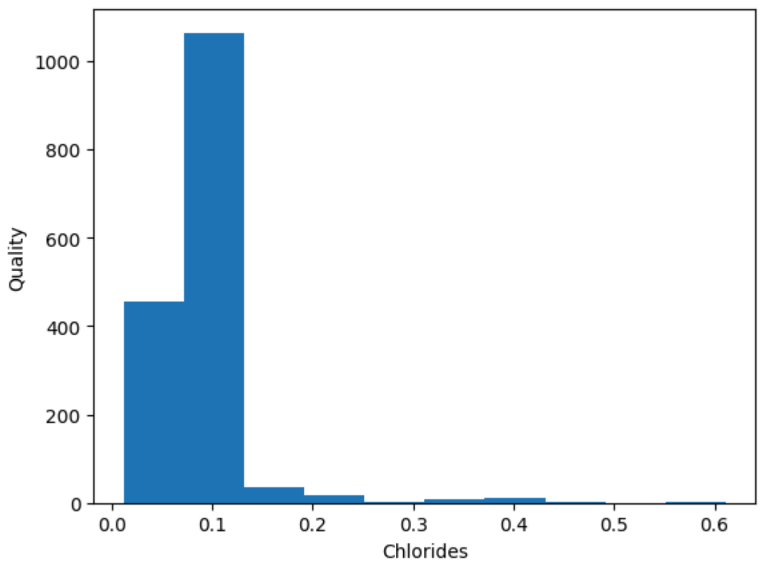
```
draw_boxplot(data, 'chlorides')
```

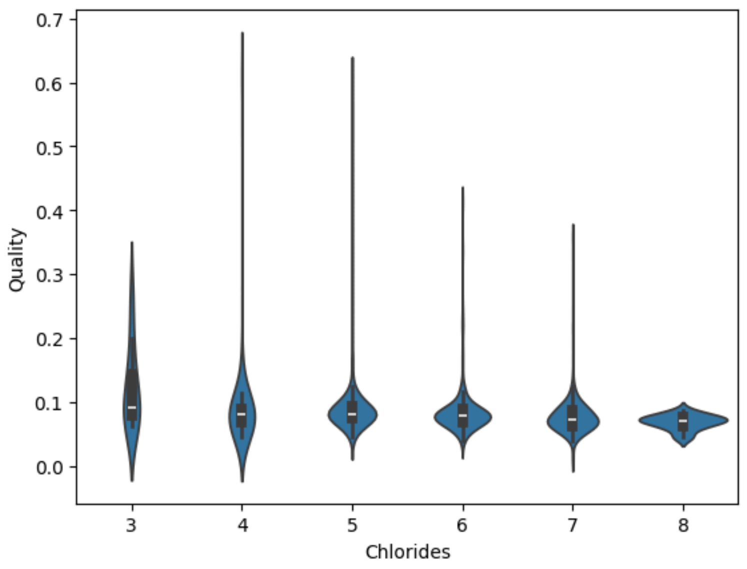


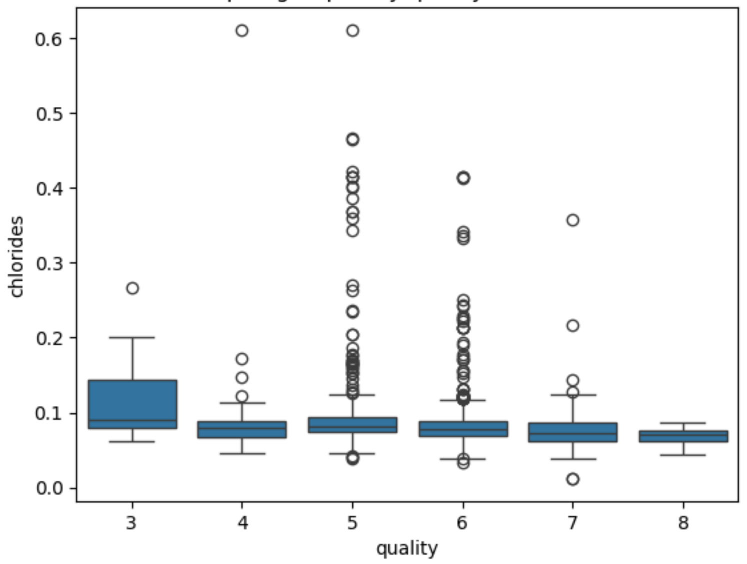
```
data = data[(data['chlorides'] > -10)]
```

Finally, `chlorides` has the following shape, presented by a histogram, boxplot, and violin plot.

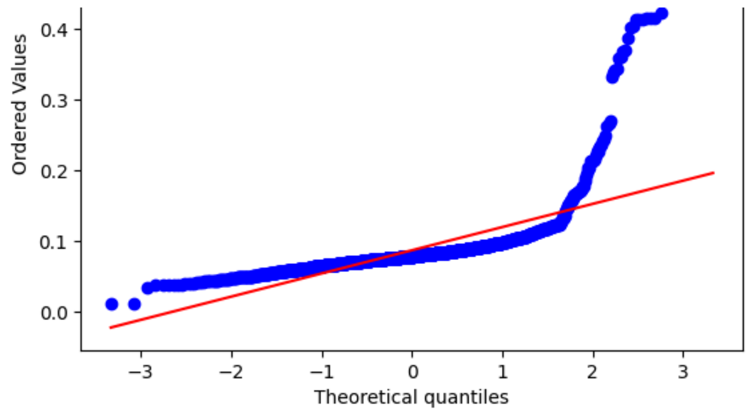
```
plt.hist(data['chlorides'])
plt.xlabel('Chlorides')
plt.ylabel('Quality')
plt.show()

sns.violinplot(y = data['chlorides'], x = 'quality', data = data)
plt.xlabel('Chlorides')
plt.ylabel('Quality')
plt.show()

draw_boxplot(data, 'chlorides')

test_normality(data['chlorides'], 'chlorides')
```





Boxplot grouped by quality for chlorides


QQ Plot of chlorides

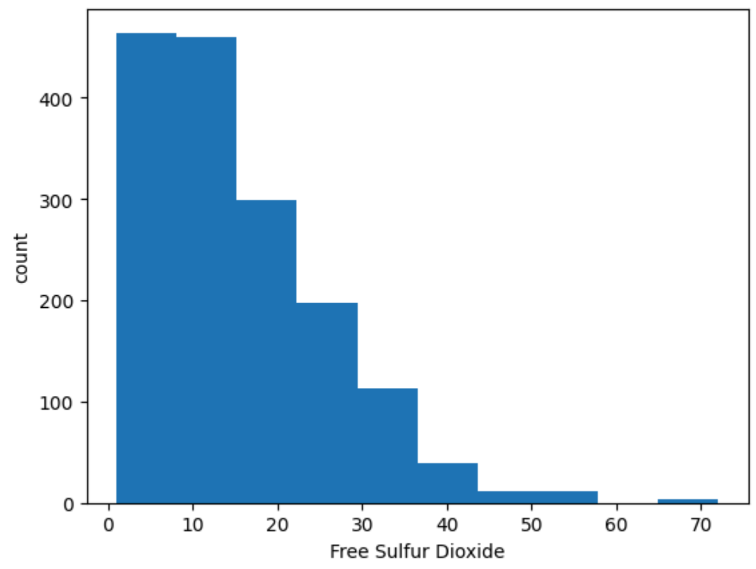


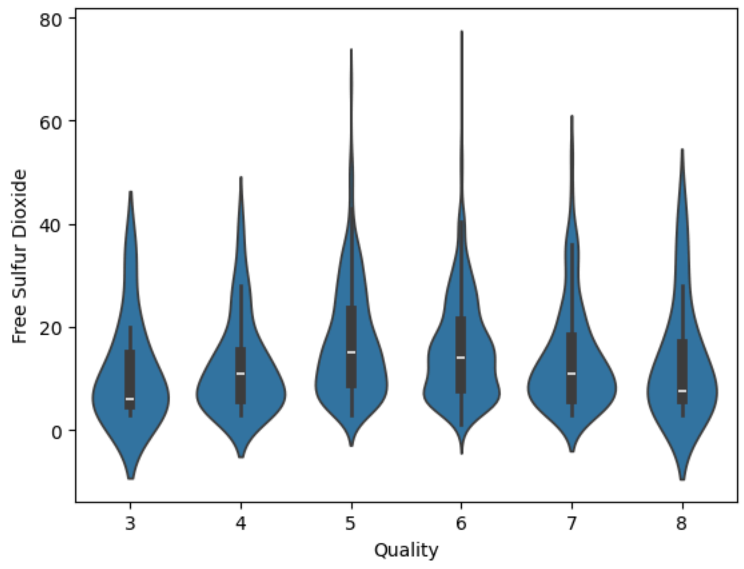
```
Shapiro-Wilk Test Statistic for chlorides: 0.48424655122518334, p-value: 1.1790556953147118e-55
```

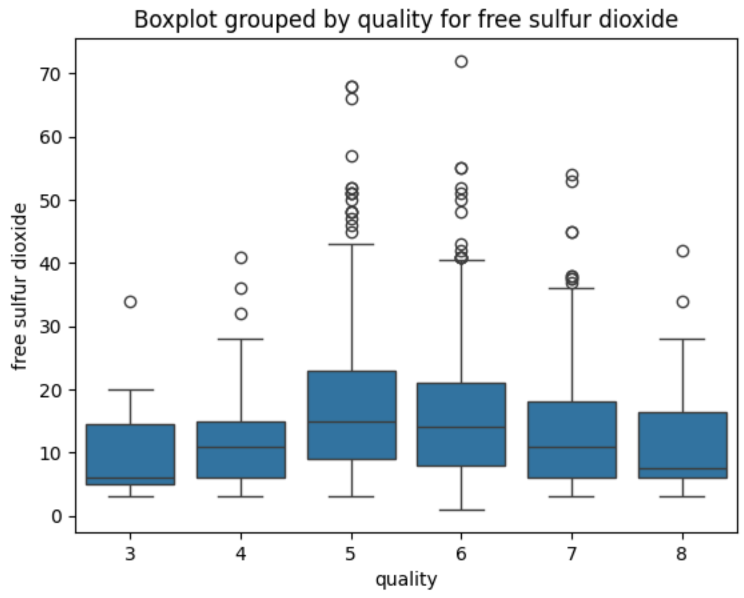
free sulfur dioxide

Below is the distribution of `free sulfur dioxide`, shown with a histogram, violin, and boxplot.

```
plt.hist(data['free sulfur dioxide'])
plt.xlabel('Free Sulfur Dioxide')
plt.ylabel('count')
plt.show()

sns.violinplot(y = data['free sulfur dioxide'], x = 'quality', data = data)
plt.ylabel('Free Sulfur Dioxide')
plt.xlabel('Quality')
plt.show()

draw_boxplot(data, 'free sulfur dioxide')
```

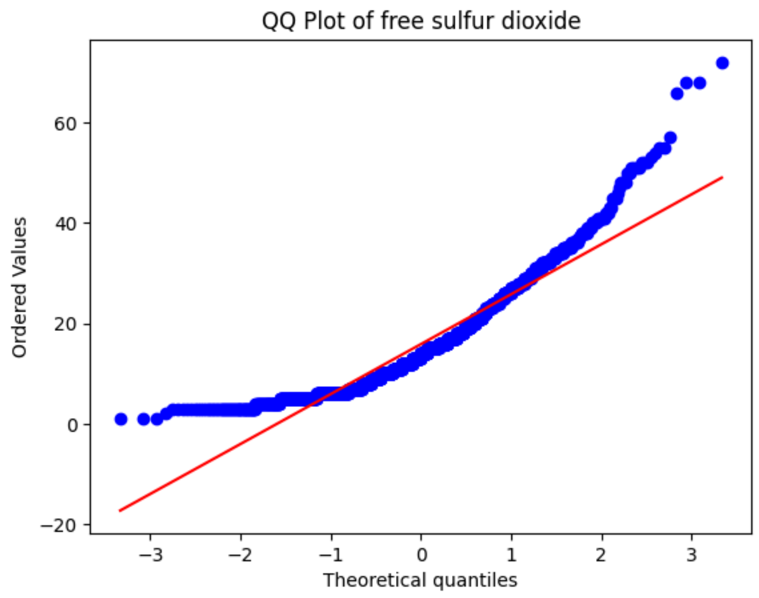




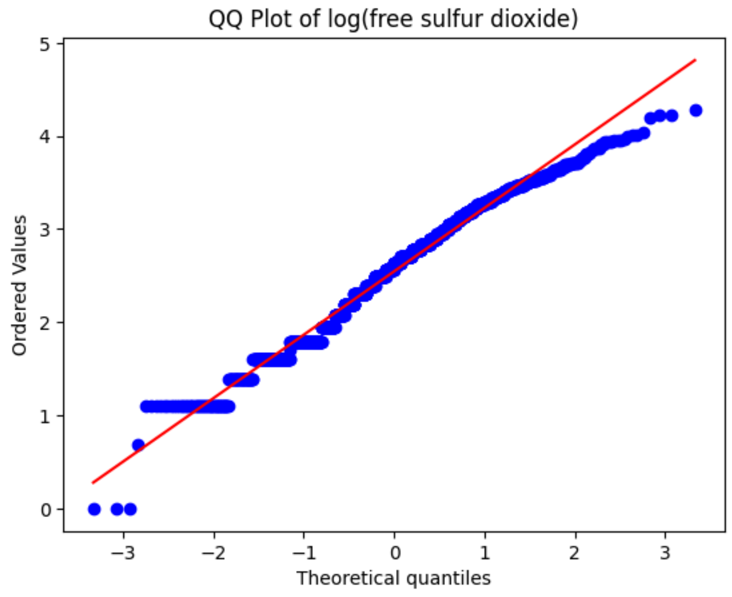


We have right skew data. We will look at the same transformations we made on `chlorides` to see if they help fix the skewness of our data. These are namely boxcox, sqrt, and log transformations.

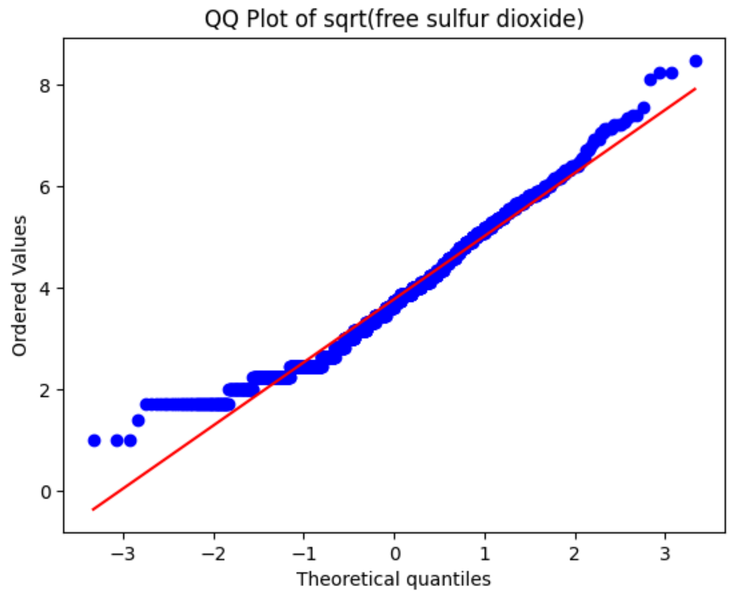
```
free_sulfur_boxcox, free_sulfur_lambda = boxcox(data['free sulfur dioxide'])


test_normality(data['free sulfur dioxide'], 'free sulfur dioxide')
test_normality(np.log(data['free sulfur dioxide']), 'log(free sulfur dioxide)')
test_normality(np.sqrt(data['free sulfur dioxide']), 'sqrt(free sulfur dioxide)')
test_normality(free_sulfur_boxcox, 'boxcox(free sulfur dioxide)')
```



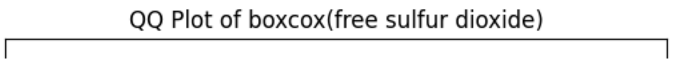
```
Shapiro-Wilk Test Statistic for free sulfur dioxide: 0.9018394916138583, p-value: 7.694596687816645e-31
```

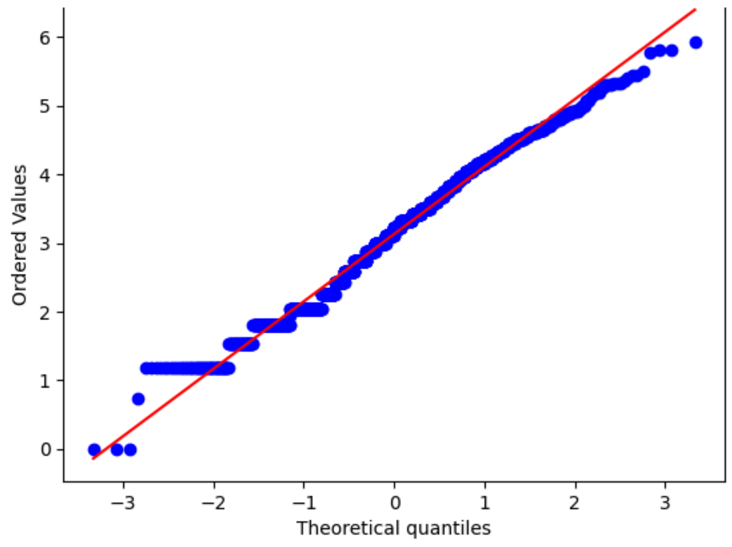


```
Shapiro-Wilk Test Statistic for log(free sulfur dioxide): 0.9833984364966966, p-value: 1.2463722155133395e-12
```



```
Shapiro-Wilk Test Statistic for sqrt(free sulfur dioxide): 0.9725749638438167, p-value: 6.699830655540897e-17
```





```
Shapiro-Wilk Test Statistic for boxcox(free sulfur dioxide): 0.9874715597874553, p-value: 1.550940154035844e-10
```

Since both the log and boxcox transformations give a similar Shapiro test statistic ($.98 < t < .99$), we will opt for a log transformation as it is slightly easier to interpret.

This means if `free sulfur dioxide` $= \vec{S}$, then our model with use

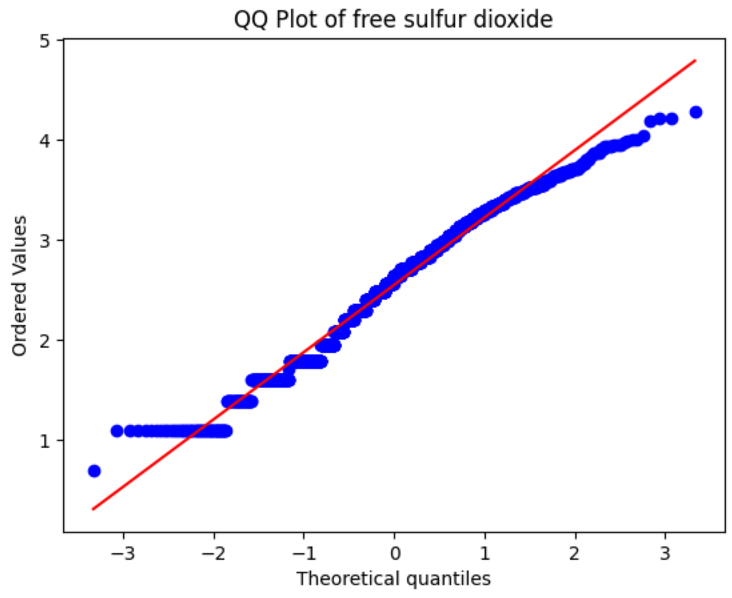
$$\vec{S} \leftarrow \log(\vec{S})$$

as the input for `free sulfur dioxide`. We make the change in the next cell

```
data['free sulfur dioxide'] = np.log(data['free sulfur dioxide'])
```

The last thing to do is re-assess for outliers, lets see how our normality and boxplots change when we remove the values that have `free sulfur dioxide` = 0.

```
test_normality(data[(data['free sulfur dioxide']) > 0.1]['free sulfur dioxide'], 'free sulfur dioxide')
```

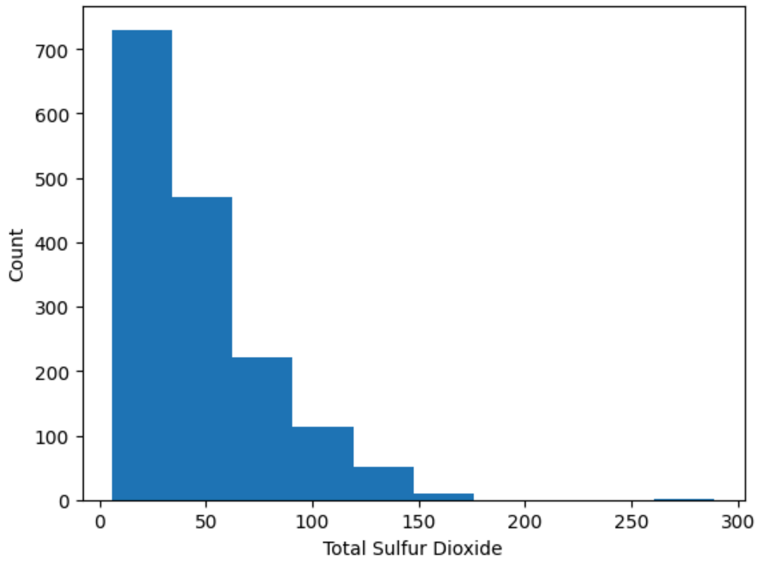


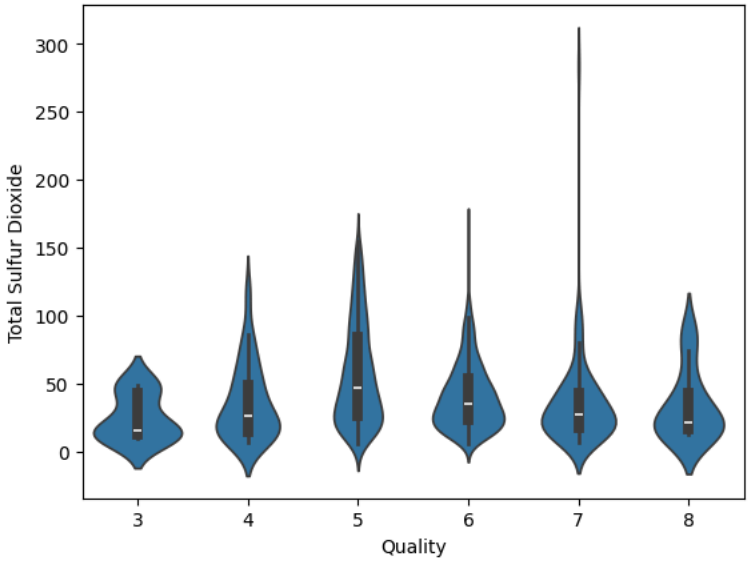
```
Shapiro-Wilk Test Statistic for free sulfur dioxide: 0.9822908798309025, p-value: 3.9812244852226943e-13
```

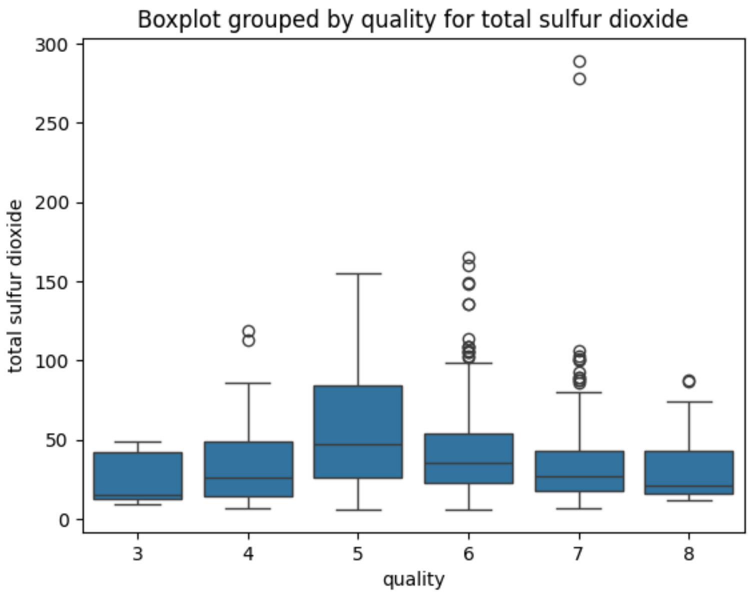
This acutally makes our data less normal, so we will leave our data as is for now.

`total sulfur dioxide`

```
plt.hist(data['total sulfur dioxide'])
plt.xlabel('Total Sulfur Dioxide')
plt.ylabel('Count')
plt.show()

sns.violinplot(y = data['total sulfur dioxide'], x = 'quality', data = data)
plt.ylabel('Total Sulfur Dioxide')
plt.xlabel('Quality')
plt.show()

draw_boxplot(data, 'total sulfur dioxide')
```





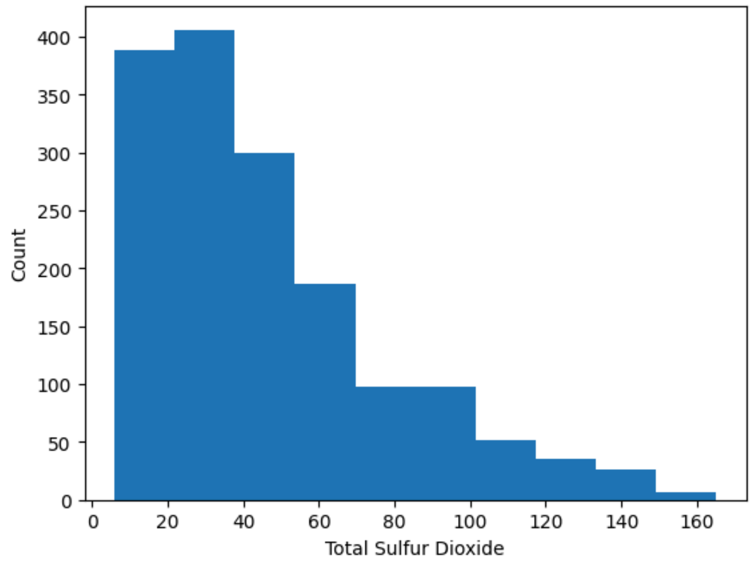


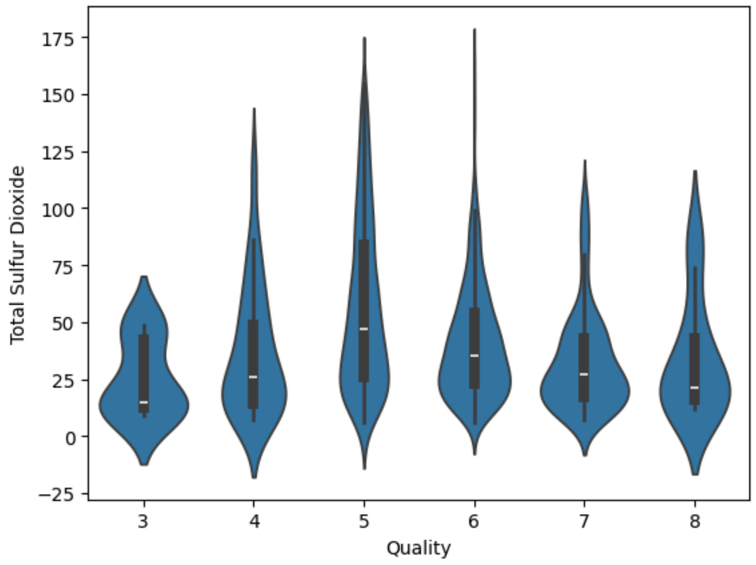
Before assessing normaility, we can see that there are two extreme outliers for total sulfur dioxide. We will define these outliers as the cases where `total sulfur dioxide` is greater than 200. We will remove these.

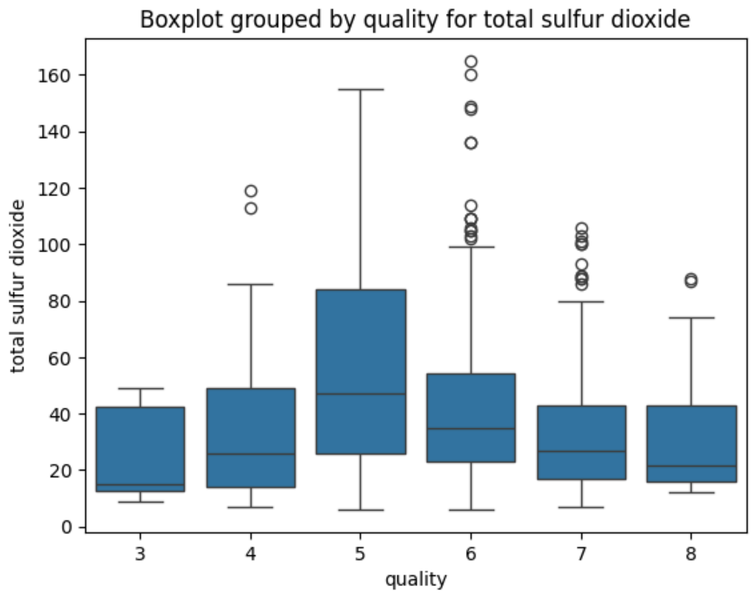
```
data = data[(data['total sulfur dioxide']) < 200] #only include values < 200
```

Let's see how this has changed the plots from earlier, and if we should still try to normalize the skew of our data.

```
#same code from above
plt.hist(data['total sulfur dioxide'])
plt.xlabel('Total Sulfur Dioxide')
plt.ylabel('Count')
plt.show()

sns.violinplot(y = data['total sulfur dioxide'], x = 'quality', data = data)
plt.ylabel('Total Sulfur Dioxide')
plt.xlabel('Quality')
plt.show()

draw_boxplot(data, 'total sulfur dioxide')
```

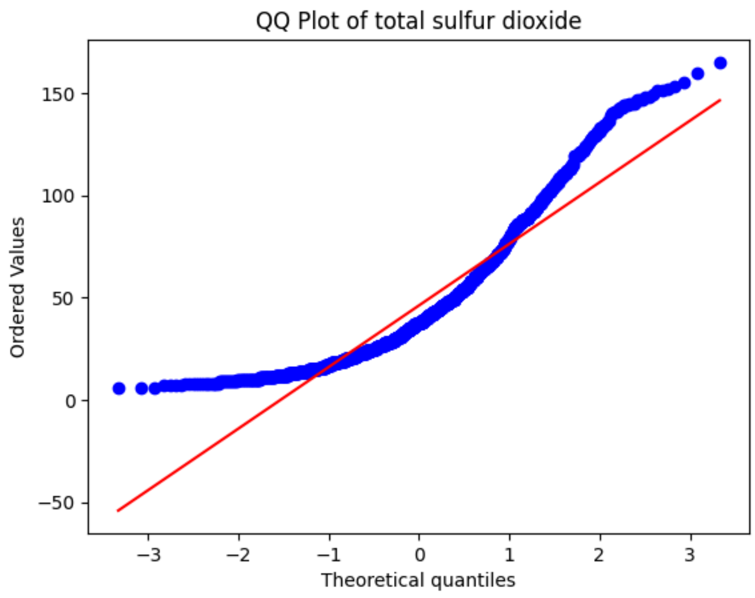




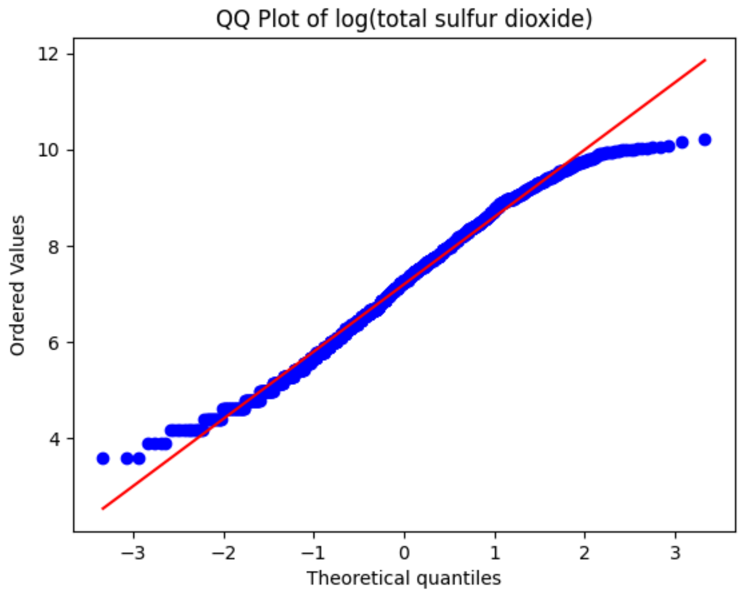


Removing the outliers this way helped the skew become less drastic, but there is still an obvious right skew. We will look at boxcox, sqrt, and log transformations to see how they effect our normaility.

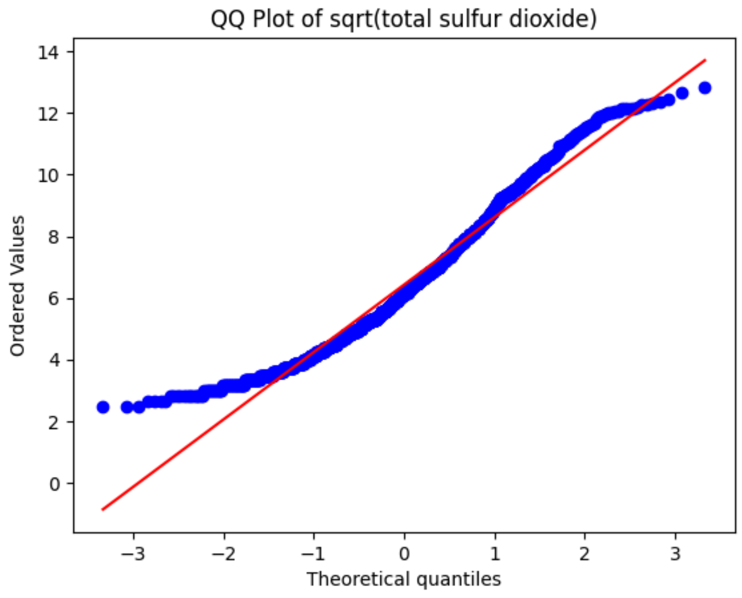
```
total_sulfur_boxcox, total_sulfur_lambda = boxcox(data['total sulfur dioxide'])


test_normality(data['total sulfur dioxide'], 'total sulfur dioxide')
test_normality(np.log(data['total sulfur dioxide']**2), 'log(total sulfur dioxide)')
test_normality(np.sqrt(data['total sulfur dioxide']), 'sqrt(total sulfur dioxide)')
test_normality(total_sulfur_boxcox, 'boxcox(total sulfur dioxide)')
```



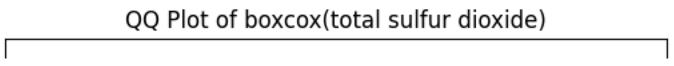
```
Shapiro-Wilk Test Statistic for total sulfur dioxide: 0.8901005442661531, p-value: 2.845716503915968e-32
```

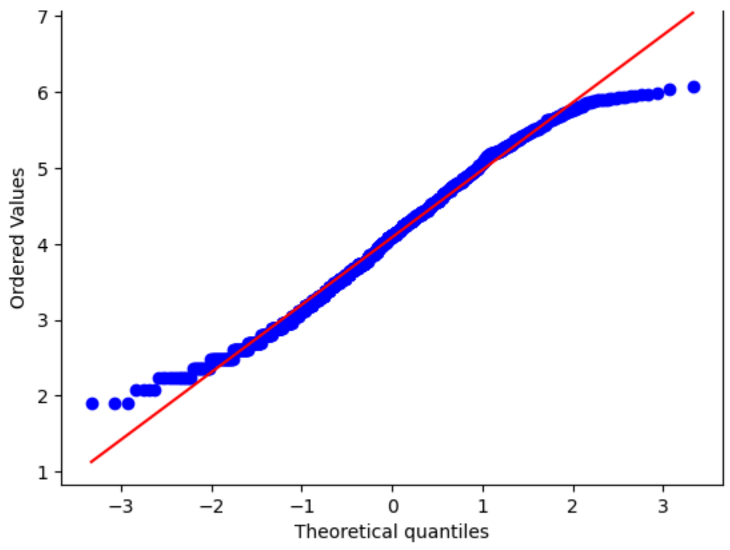


```
Shapiro-Wilk Test Statistic for log(total sulfur dioxide): 0.987955192091651, p-value: 2.986940444373711e-10
```



```
Shapiro-Wilk Test Statistic for sqrt(total sulfur dioxide): 0.9643677622789827, p-value: 2.1870819867468594e-19
```





```
Shapiro-Wilk Test Statistic for boxcox(total sulfur dioxide): 0.9885075208251186, p-value: 6.288487548570056e-10
```

We will choose to use a log transformation for our data, as it does nearly as well as boxcox while being more easily interpretable.

Given $\vec{S}$ is the `total sulfur dioxide` for our samples, we will take

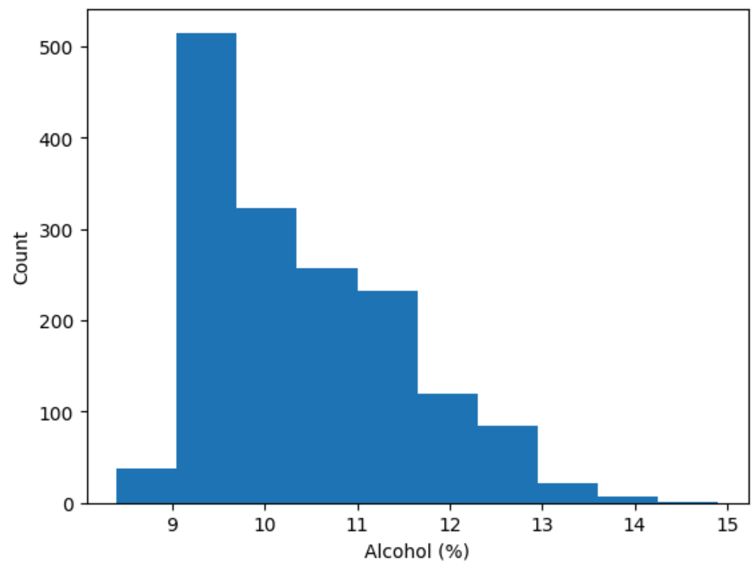
$$\vec{S} \leftarrow \log(\vec{S})$$

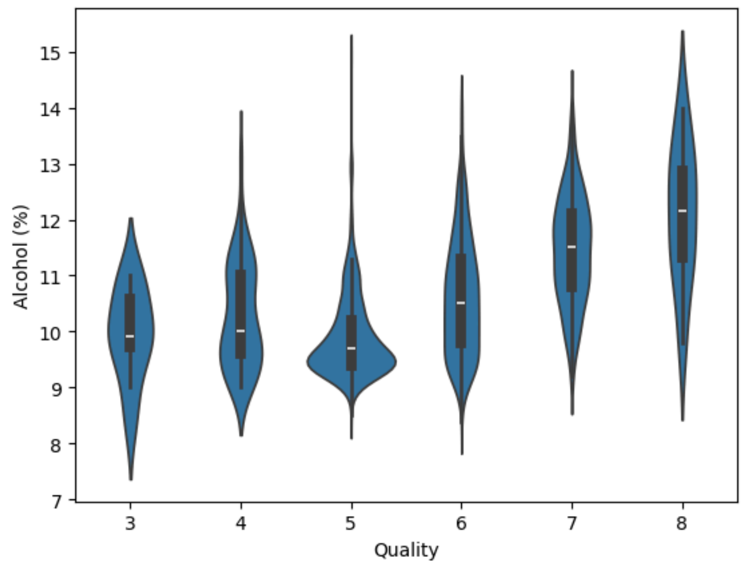
```
data['total sulfur dioxide'] = np.log(data['total sulfur dioxide'])
```

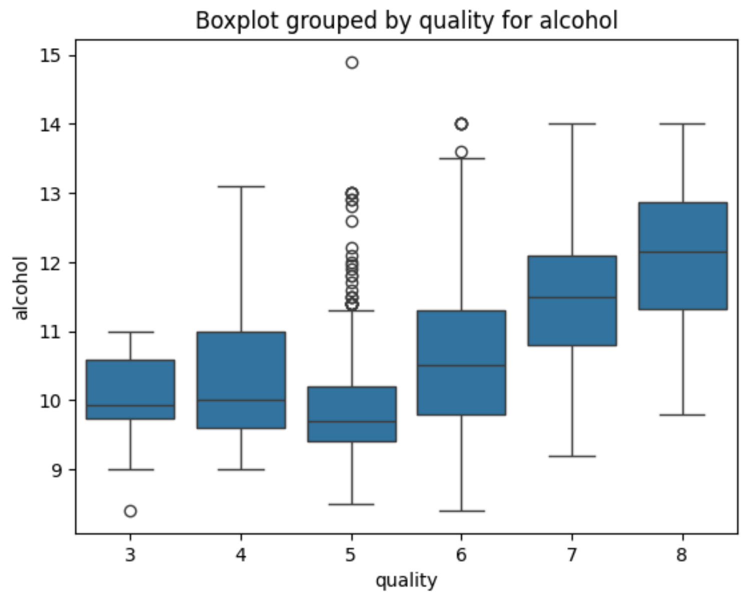
alcohol

The last variable we will look at to change the distribution, based on our initial glace, will be `alcohol`.

```
#same code from above
plt.hist(data['alcohol'])
plt.xlabel('Alcohol (%)')
plt.ylabel('Count')
plt.show()

sns.violinplot(y = data['alcohol'], x = 'quality', data = data)
plt.ylabel('Alcohol (%)')
plt.xlabel('Quality')
plt.show()

draw_boxplot(data, 'alcohol')
```



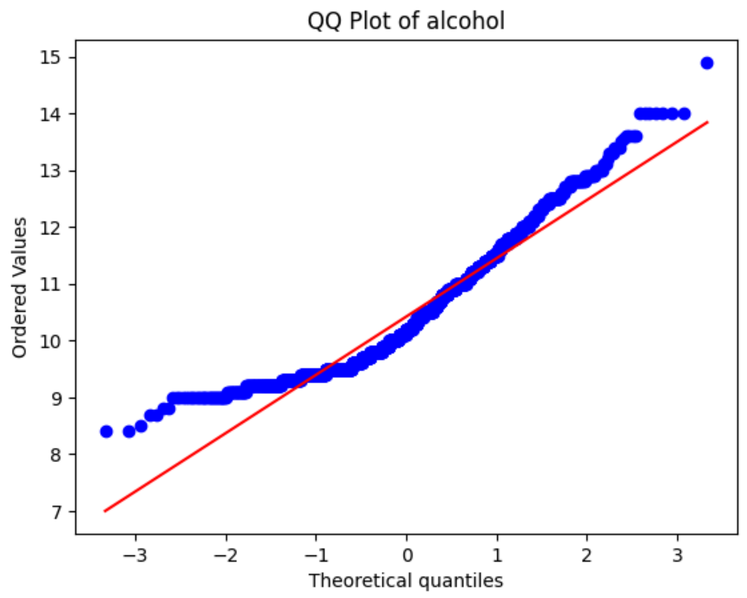




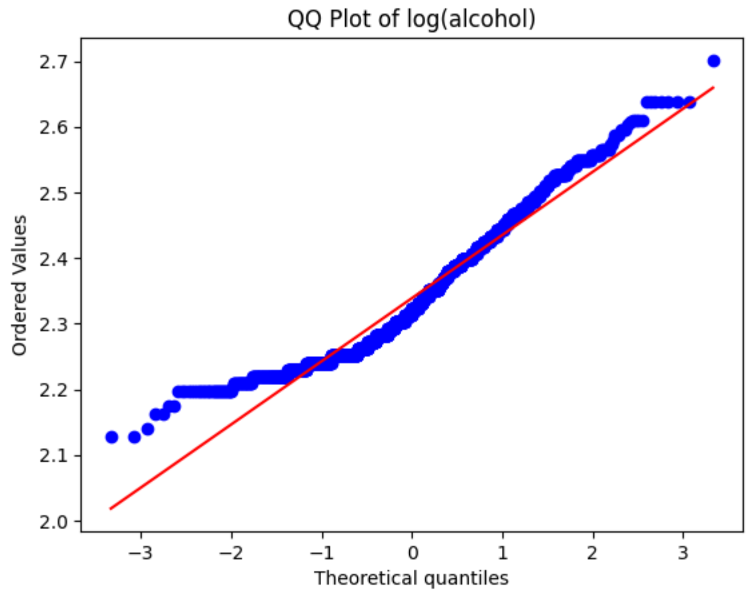
Check how boxcox, log, and sqrt transformations effect outliers and normality.

```
alcohol_boxcox, alcohol_lambda = boxcox(data['alcohol'])
```

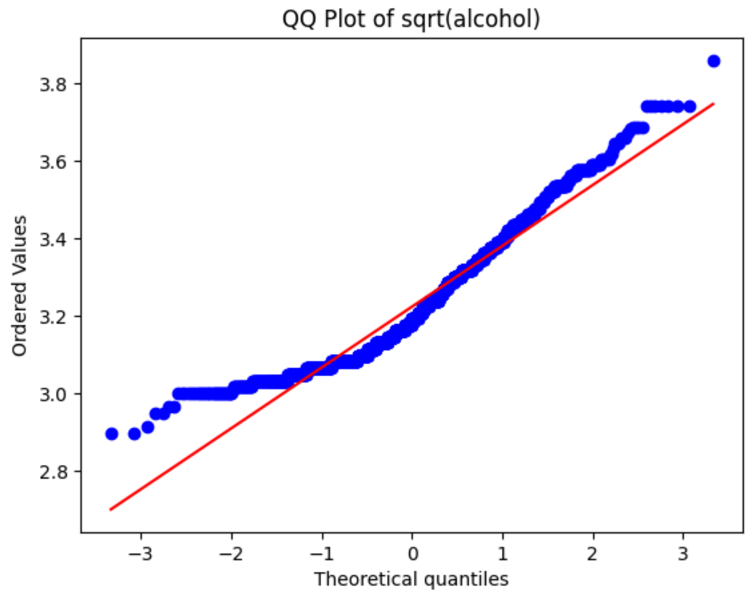
```
test_normality(data['alcohol'], 'alcohol')

test_normality(np.log(data['alcohol']), 'log(alcohol)')


test_normality(np.sqrt(data['alcohol']), 'sqrt(alcohol)')


test_normality(total_sulfur_boxcox, 'boxcox(alcohol)')
```



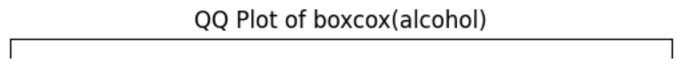
```
Shapiro-Wilk Test Statistic for alcohol: 0.9286791253859746, p-value: 6.482928724457048e-27
```

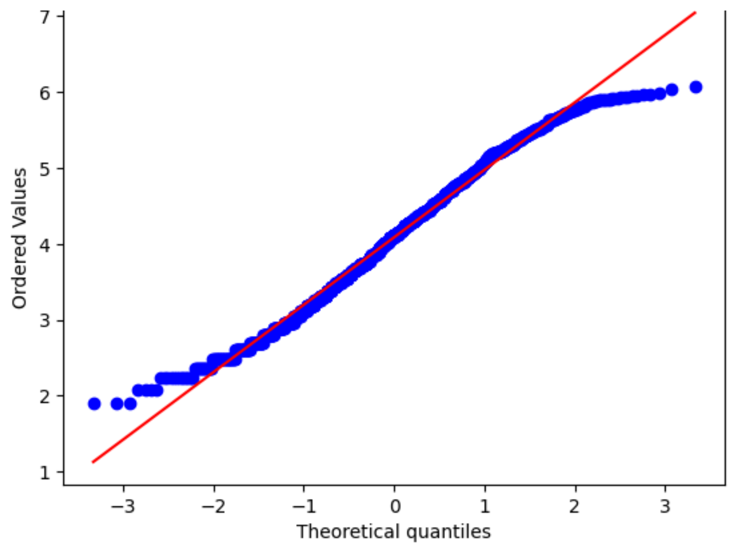


```
Shapiro-Wilk Test Statistic for log(alcohol): 0.9463831504798743, p-value: 1.1657602285248229e-23
```



```
Shapiro-Wilk Test Statistic for sqrt(alcohol): 0.938173968431395, p-value: 2.946880346198002e-25
```





```
Shapiro-Wilk Test Statistic for boxcox(alcohol): 0.9885075208251186, p-value: 6.288487548570056e-10
```

Based on the results of our testing , it seems that boxcox did the best job of normalizing our data. We will change `alcohol` to `boxcox(alcohol)`, which takes the following formula:

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \\ \ln(y_i), & \text{if } \lambda = 0 \end{cases}$$

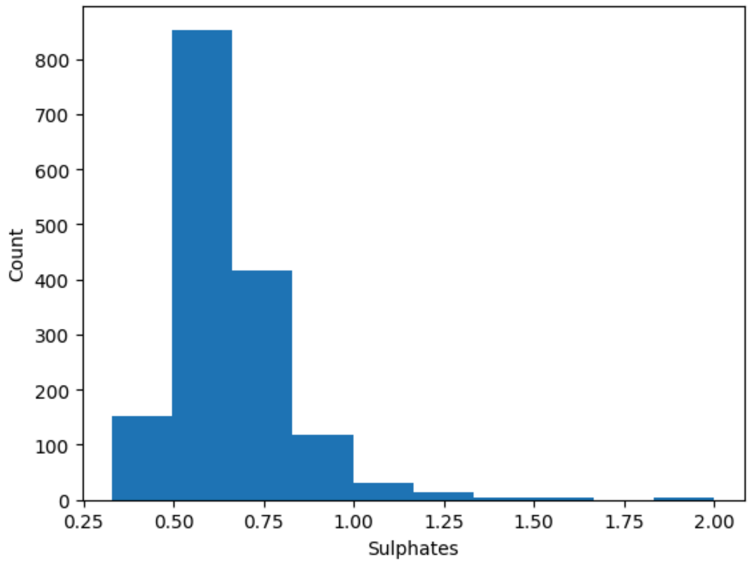
We make this change in the next cell, along with storing the original value for `data['alcohol']` in `alcohol_data` for reference.

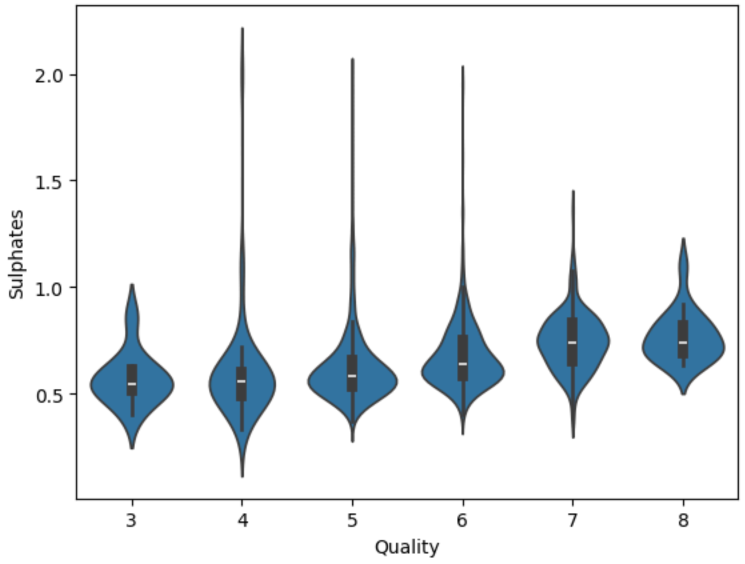
```
alcohol_data = data['alcohol']
data['alcohol'] = alcohol_boxcox
```

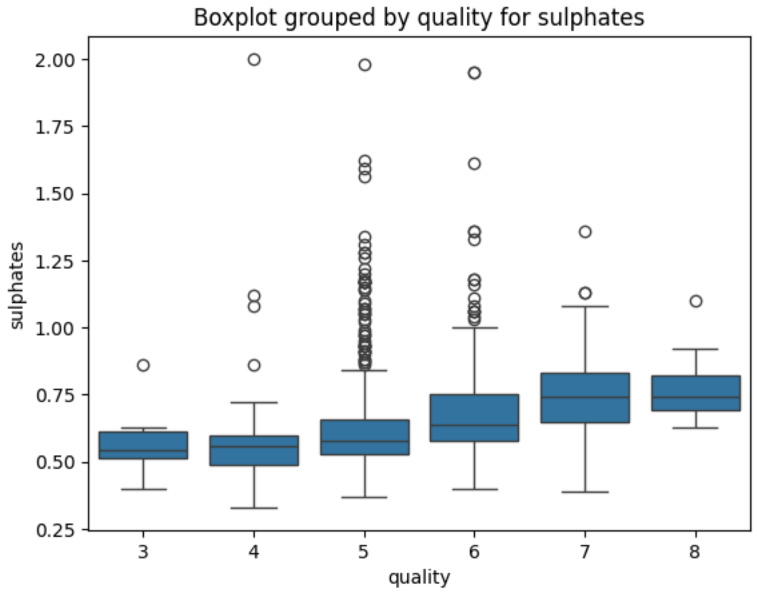
sulfates

lets look at the shape of sulfates to see what changes should be made.

```
#same code from above
plt.hist(data['sulphates'])
plt.xlabel('Sulphates')
plt.ylabel('Count')
plt.show()

sns.violinplot(y = data['sulphates'], x = 'quality', data = data)
plt.ylabel('Sulphates')
plt.xlabel('Quality')
plt.show()

draw_boxplot(data, 'sulphates')
```

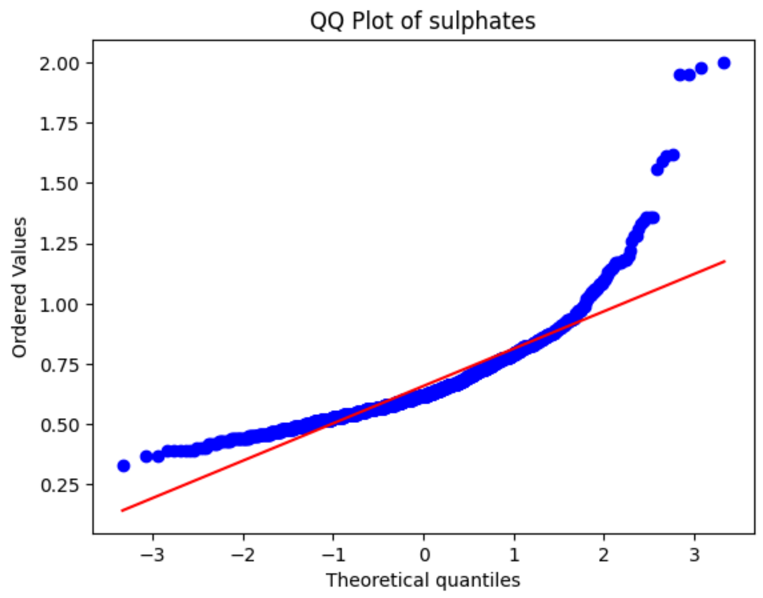




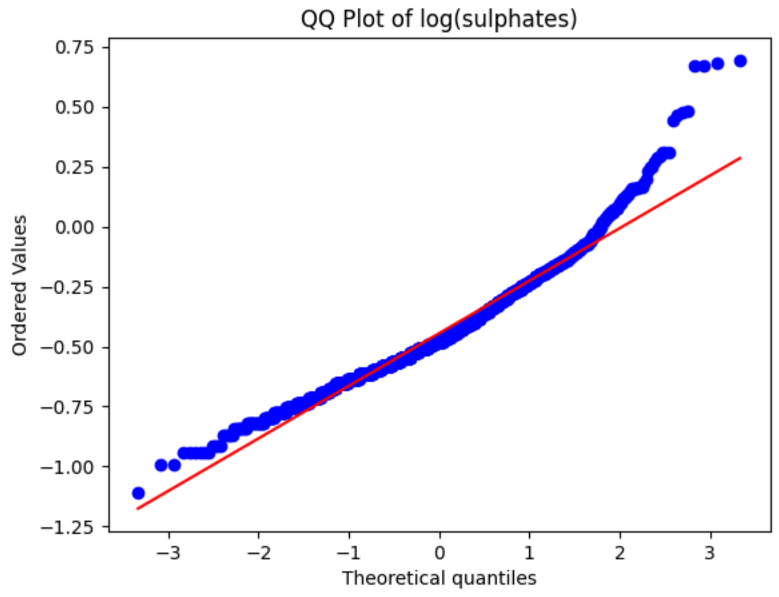


We will try our typical transformations, along with an inverse transformation to try and normalize the distribution of our data.

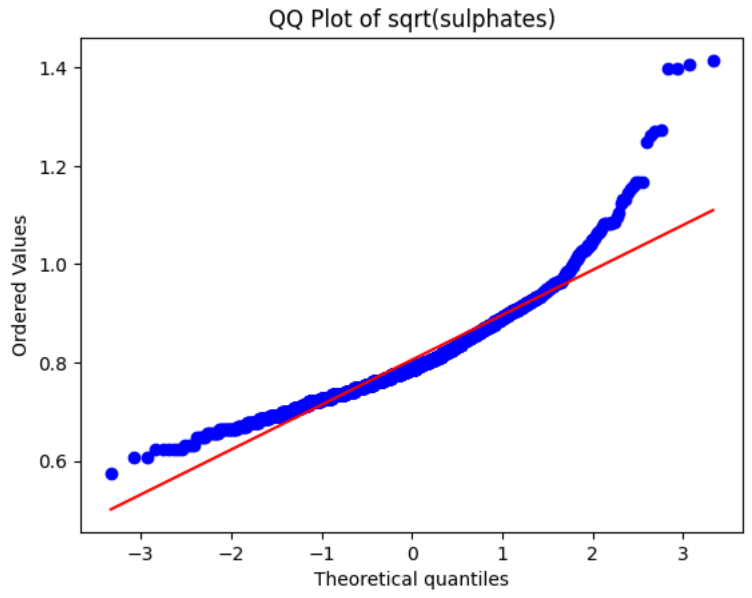
```
def test_varied_normality(data, title): #uses test_normality with boxcox, log, and sqrt transformations and the base case
  test_normality(data, title)
  test_normality(np.log(data), 'log(' + title + ')')
  test_normality(np.sqrt(data), 'sqrt(' + title + ')')
  test_normality(boxcox(data)[0], 'boxcox(' + title + ')')

test_varied_normality(data['sulphates'], 'sulphates')
test_normality(1/data['sulphates'], '1/sulphates')
```



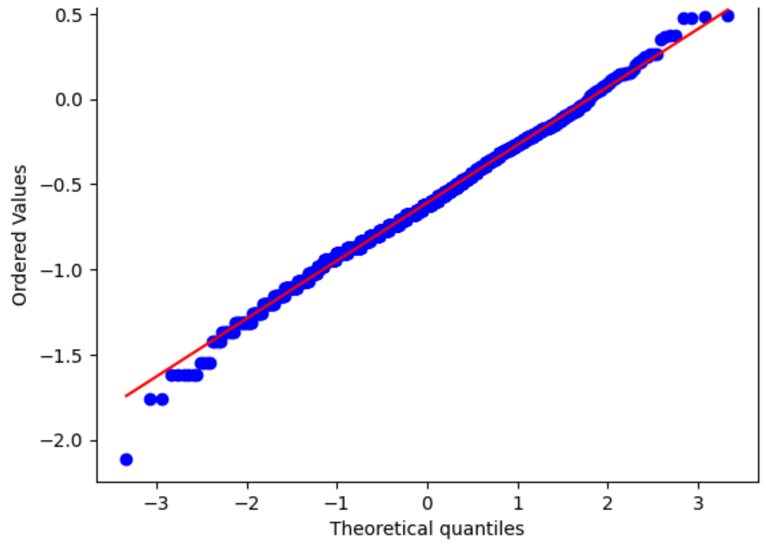
```
Shapiro-Wilk Test Statistic for sulphates: 0.8331331995132054, p-value: 6.198155100264859e-38
```



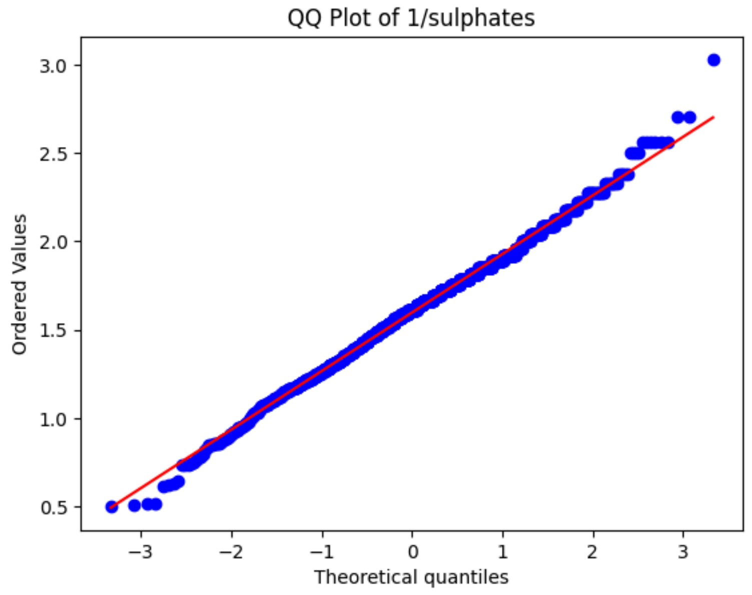
```
Shapiro-Wilk Test Statistic for log(sulphates): 0.9589699699532543, p-value: 8.247531812929651e-21
```



```
Shapiro-Wilk Test Statistic for sqrt(sulphates): 0.9087192223058499, p-value: 6.562777046500386e-30
```



```
Shapiro-Wilk Test Statistic for boxcox(sulphates): 0.9964508719419674, p-value: 0.0009392978353782306
```



```
Shapiro-Wilk Test Statistic for 1/sulphates: 0.9963963086800092, p-value: 0.0008240181117632388
```

Boxcox and Inverse perform similarly, so we will opt for the inverse transformation. Where if $\vec{S}$ is `sulphates`, then our updated value becomes

$$\vec{S} = \frac{1}{\vec{S}}$$

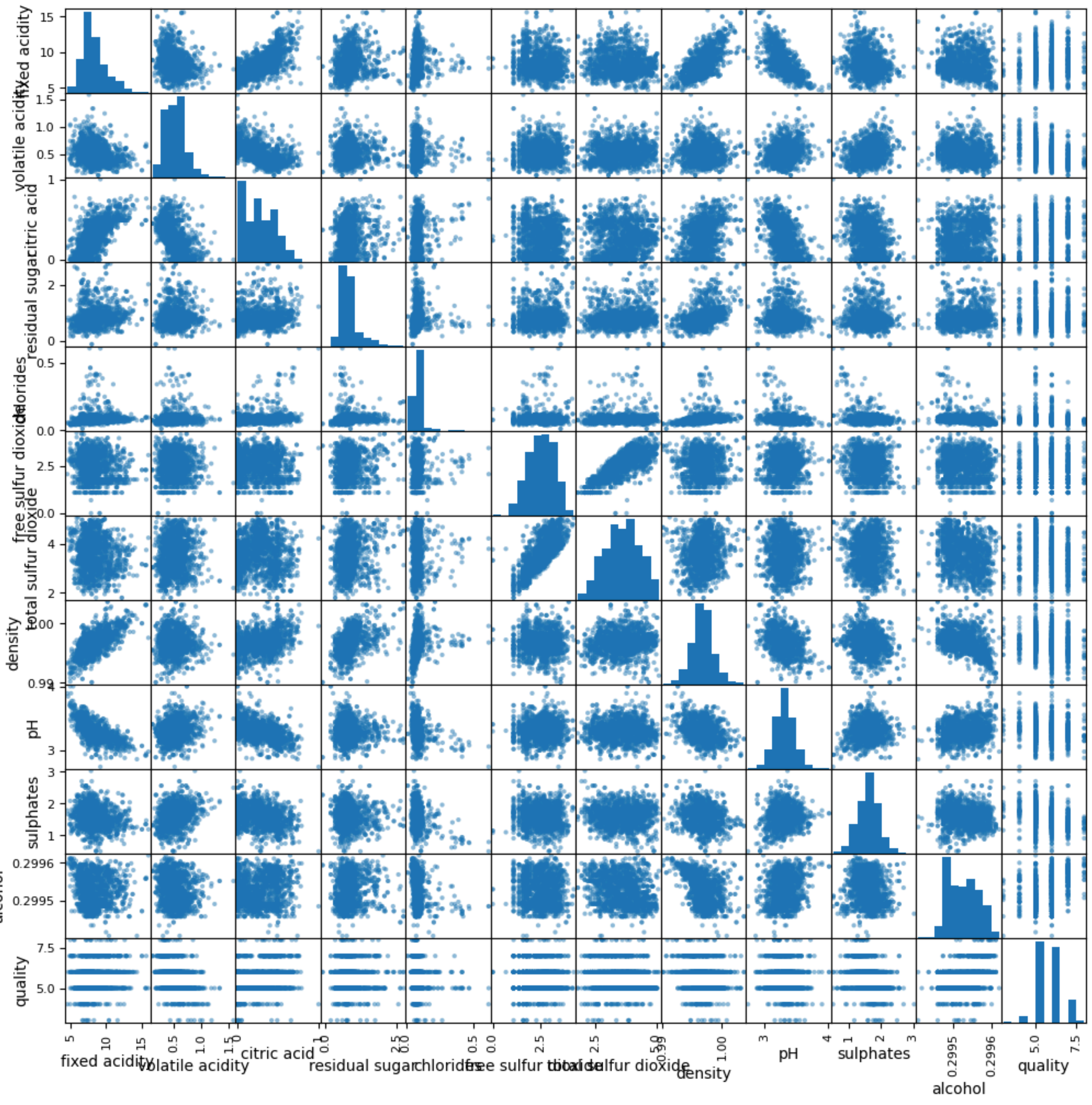
```
data['sulphates'] = 1/data['sulphates']
```

Verfiying data transformation

lets re-assess the scatter matrix and see if we have done enough with modifying distributions

Scatterplot Matrix

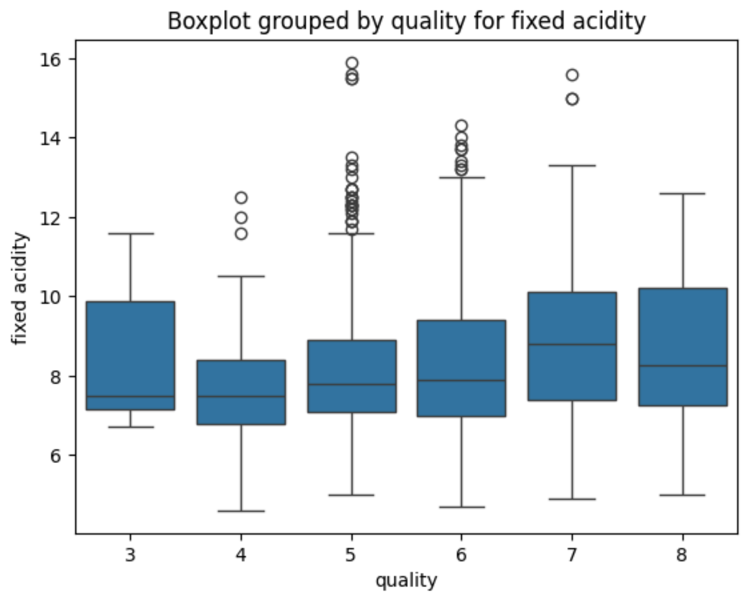
```
pd.plotting.scatter_matrix(data, figsize=(12,12))[-1]
plt.show()
```

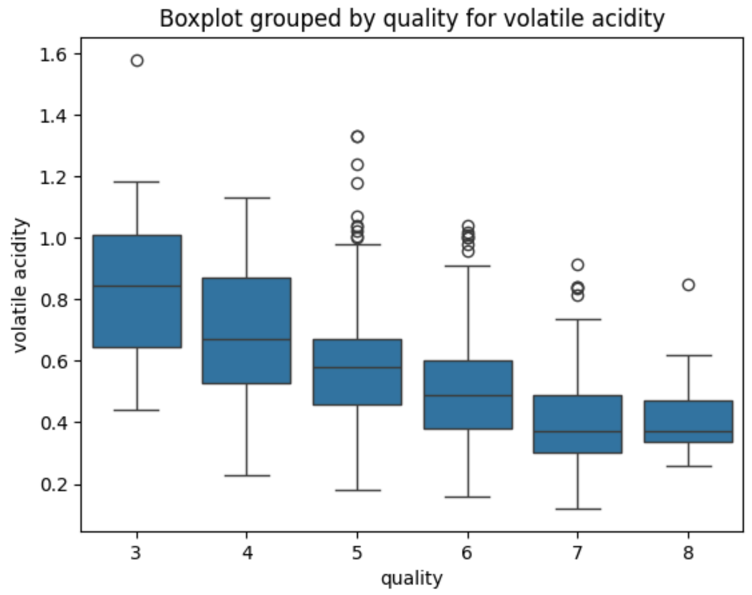


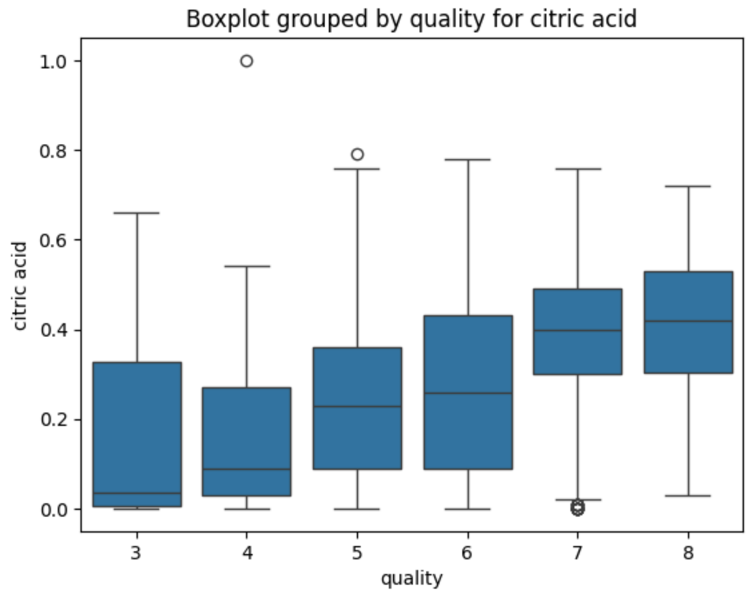
Boxplots for Outliers

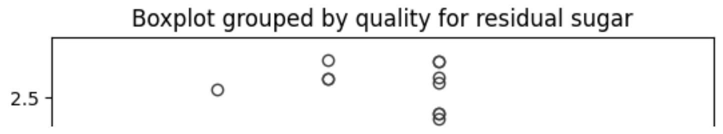
Double-click (or enter) to edit

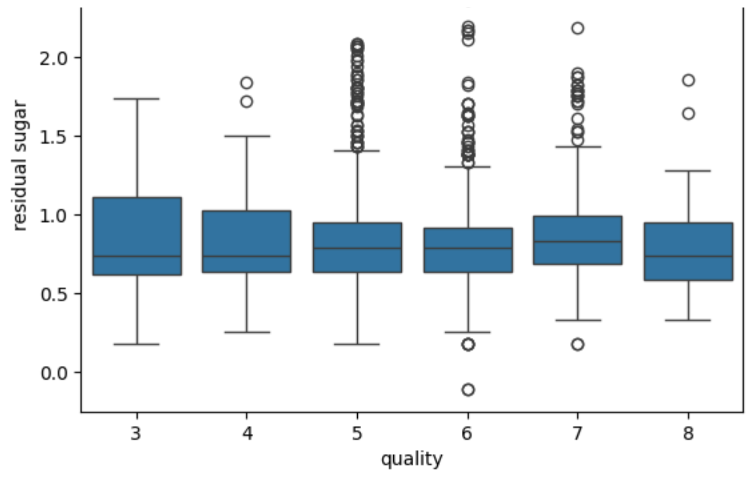
```
for item in data:
  if item != 'quality':
    draw_boxplot(data, item)
```


Boxplot grouped by quality for fixed acidity
fixed acidity
quality
16
14
12
10
8
6
3
4
5
6
7
8

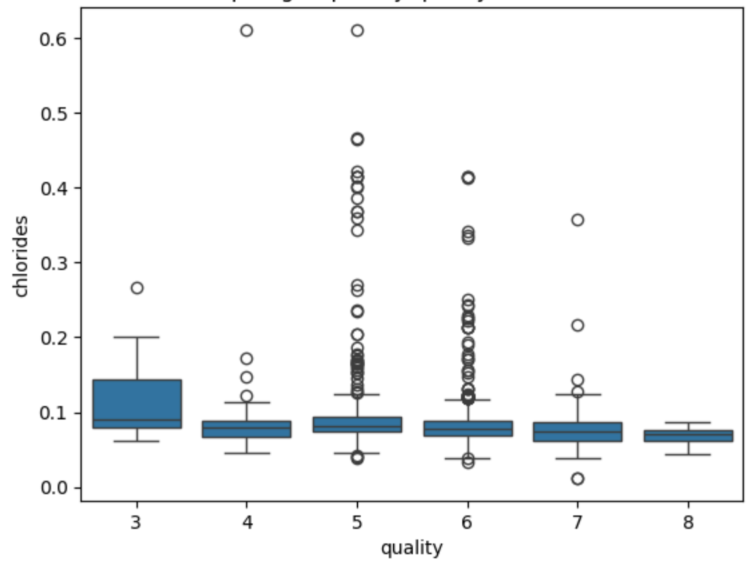

Boxplot grouped by quality for volatile acidity
volatile acidity
quality
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
3
4
5
6
7
8


Boxplot grouped by quality for citric acid
citric acid
quality
1.0
0.8
0.6
0.4
0.2
0.0
3
4
5
6
7
8

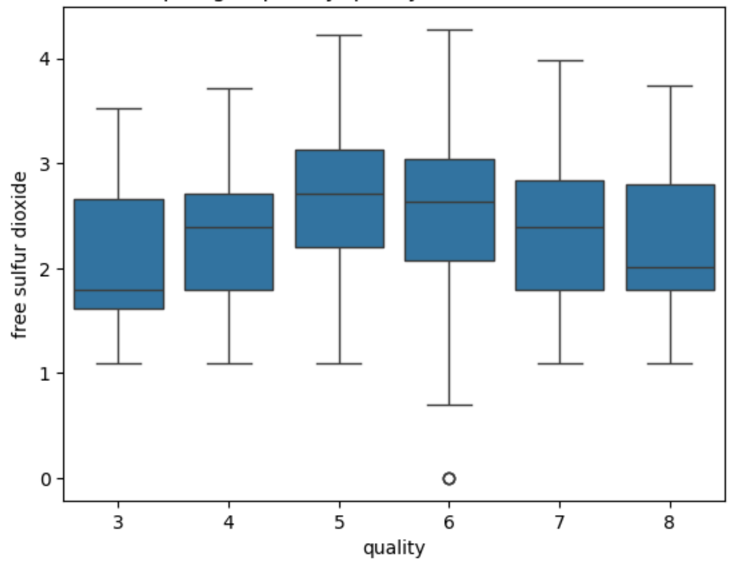

Boxplot grouped by quality for residual sugar
2.5



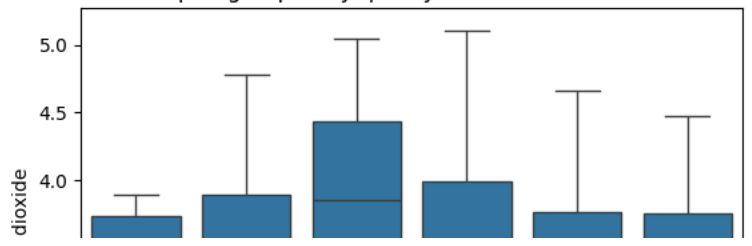
Boxplot grouped by quality for chlorides

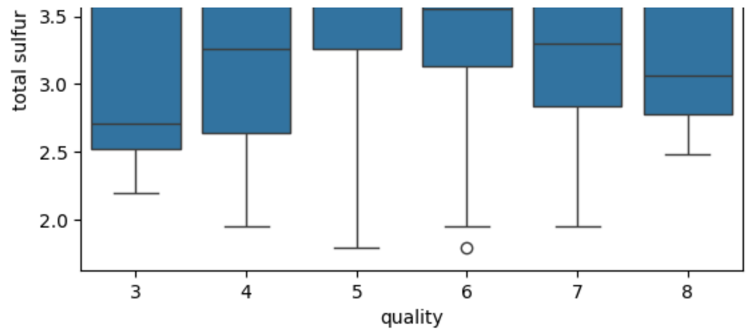


Boxplot grouped by quality for free sulfur dioxide

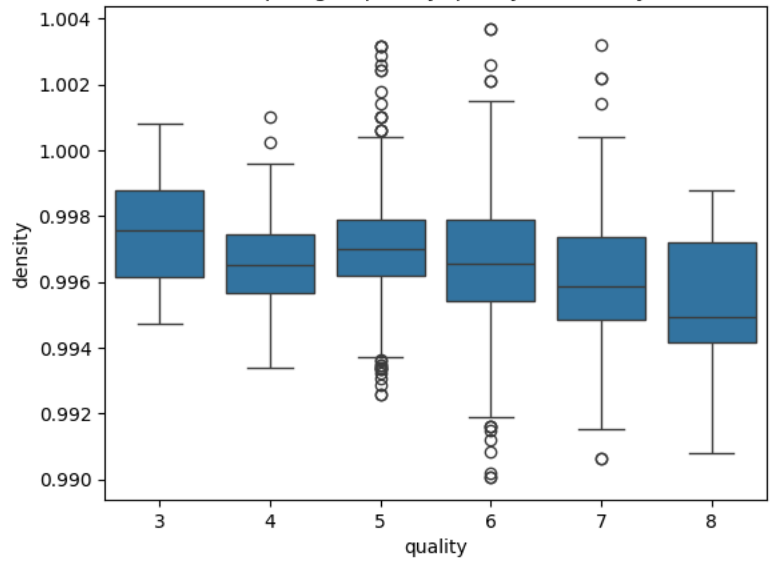


Boxplot grouped by quality for total sulfur dioxide

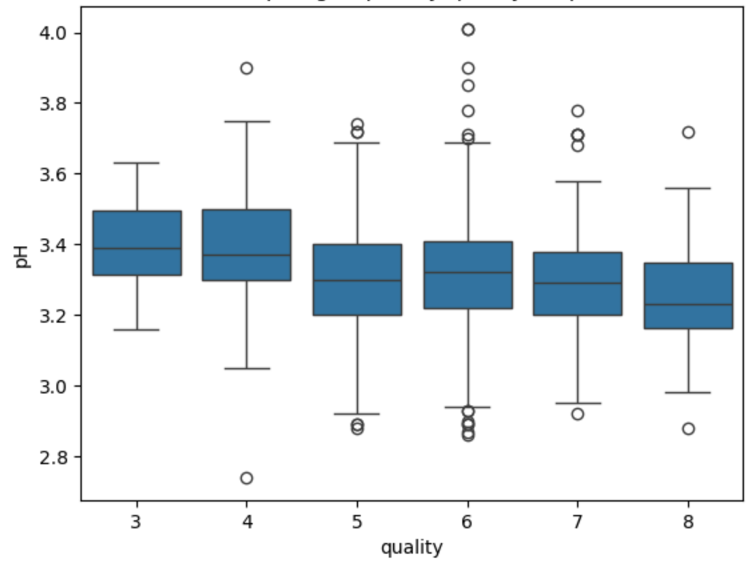




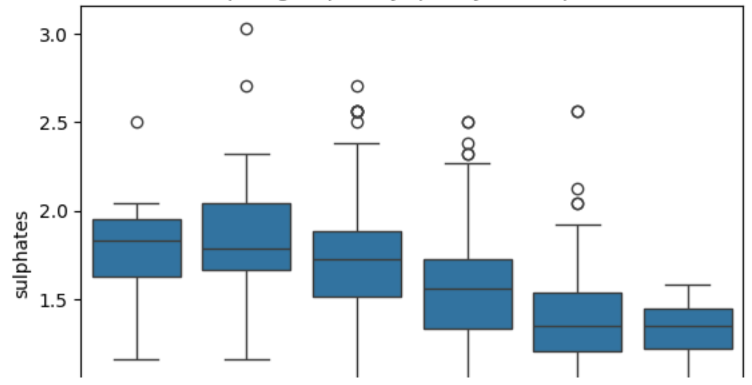
Boxplot grouped by quality for density

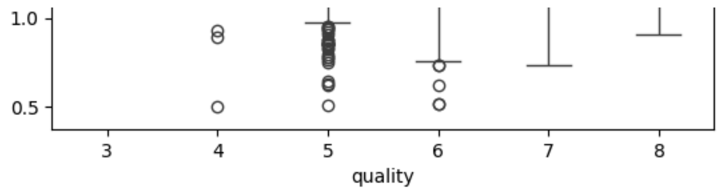


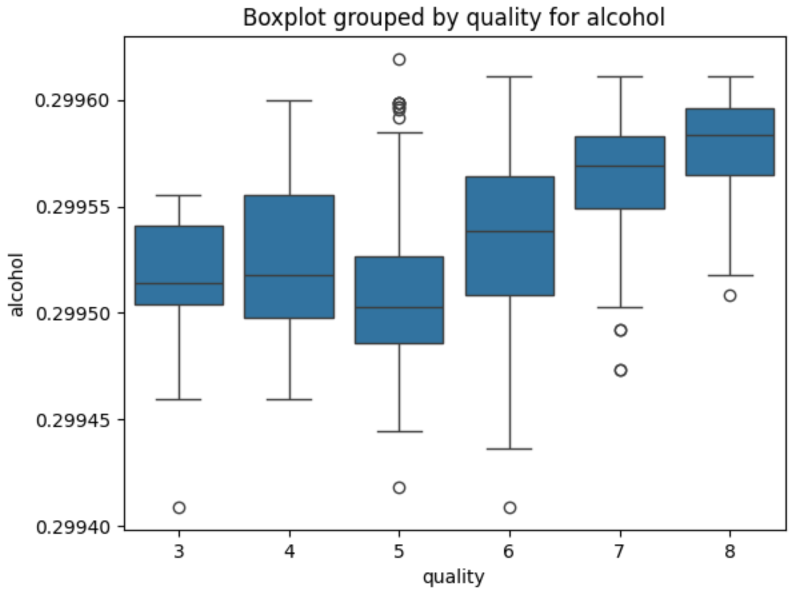
Boxplot grouped by quality for pH



Boxplot grouped by quality for sulphates




1.0
0.5
3
4
5
6
7
8
quality


Boxplot grouped by quality for alcohol
0.29960
0.29955
0.29950
0.29945
0.29940
alcohol
3
4
5
6
7
8
quality

Based on the boxplots, there are only two glaring outliers. There is one case where volitile acicity is ~ 1.6, where the rest of the values fall between $(0.1, 1.4)$. Similarly, `citric acid` falls between $(0, 0.8)$, and there is an obersvation with `citric acid` $= 1$. These are the last changes that will be made before we apply standard scaler.

```
data = data[(data['volatile acidity'] < 1.5)] # removing outliers for volatile acidity
data = data[(data['citric acid'] < 0.95)]
```

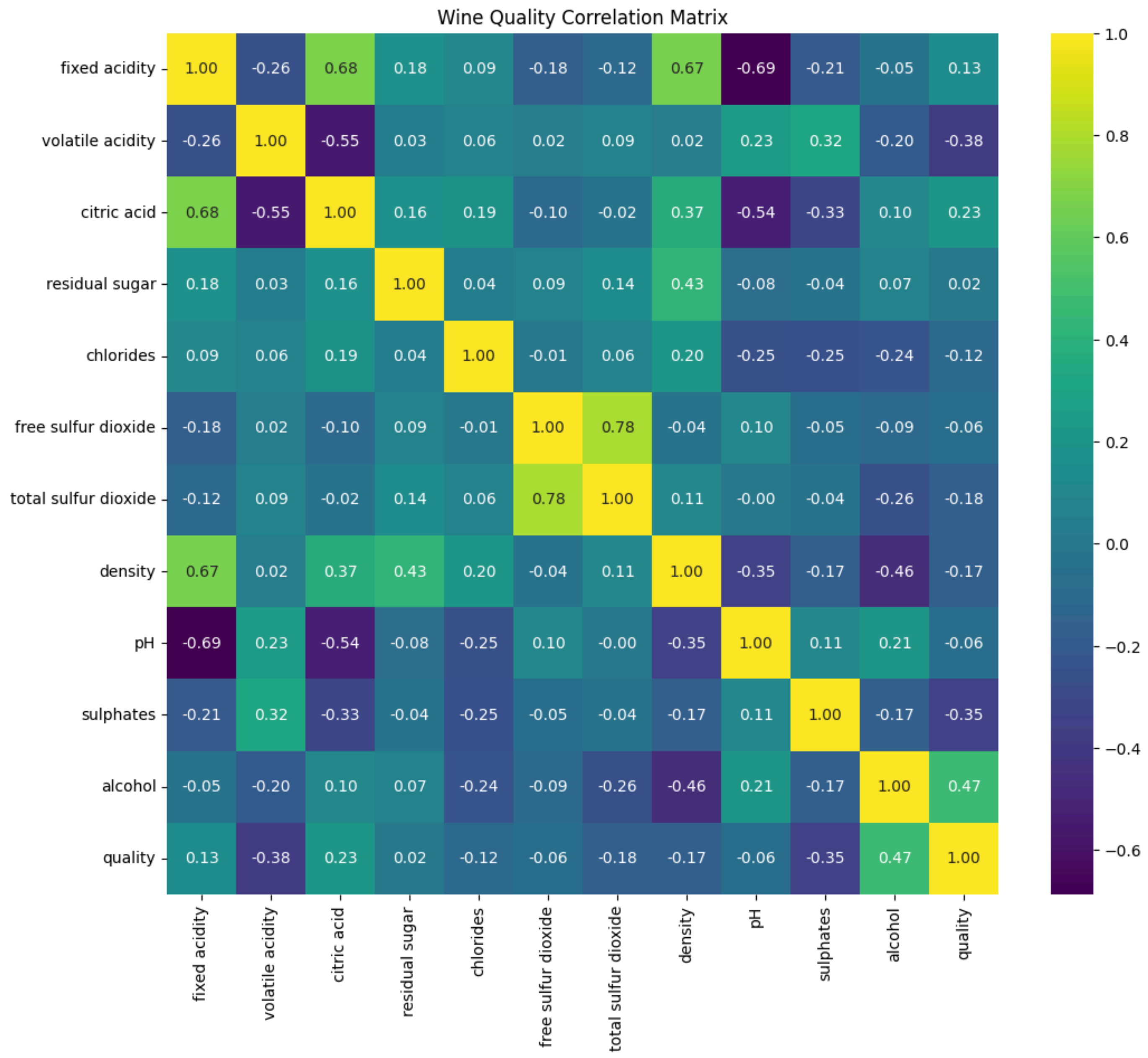
Work Planned for November 1st to November 7th

Project will proceed with data processing. We will **apply the StandardScaler** function from TensorFlow to normalize the features and carry out feature engineering to create new variables that may enhance the model's ability to predict wine quality. This step is *essential for preparing the data* for neural network training by ensuring all features are appropriately scaled and engineered.

Correlation Matrix

It's important to check the covariance matrix for confounding variables. If two vars are highly corellated, it might be wise to only include one of them in the model, or create a new feature all together.

```
plt.figure(figsize=(12, 10))
sns.heatmap(data.corr(), annot=True, fmt=".2f", cmap='viridis')
plt.title('Wine Quality Correlation Matrix')
plt.show()
```



Since $corr($ `free sulfur dioxide`, `total sulfur dioxide` $) = 0.78$, we will remove free sulfur dioxide from the dataset for traing the model, since it will be partly included with `total sulfur dioxide`.

Train-Test split

```
from sklearn.model_selection import train_test_split

X = data.drop('quality', axis=1)
y = data['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Below is duplicated code with unqiue naming, that will be used at the end for validation

```
X = data.drop('quality', axis=1)
y = data['quality']
X_training, X_testing, y_training, y_testing = train_test_split(X, y, test_size=0.2, random_state=1)
```

Applying StandardScaler

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
# Transform test data
X_test_scaled = scaler.transform(X_test)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_test_scaled.head()
```

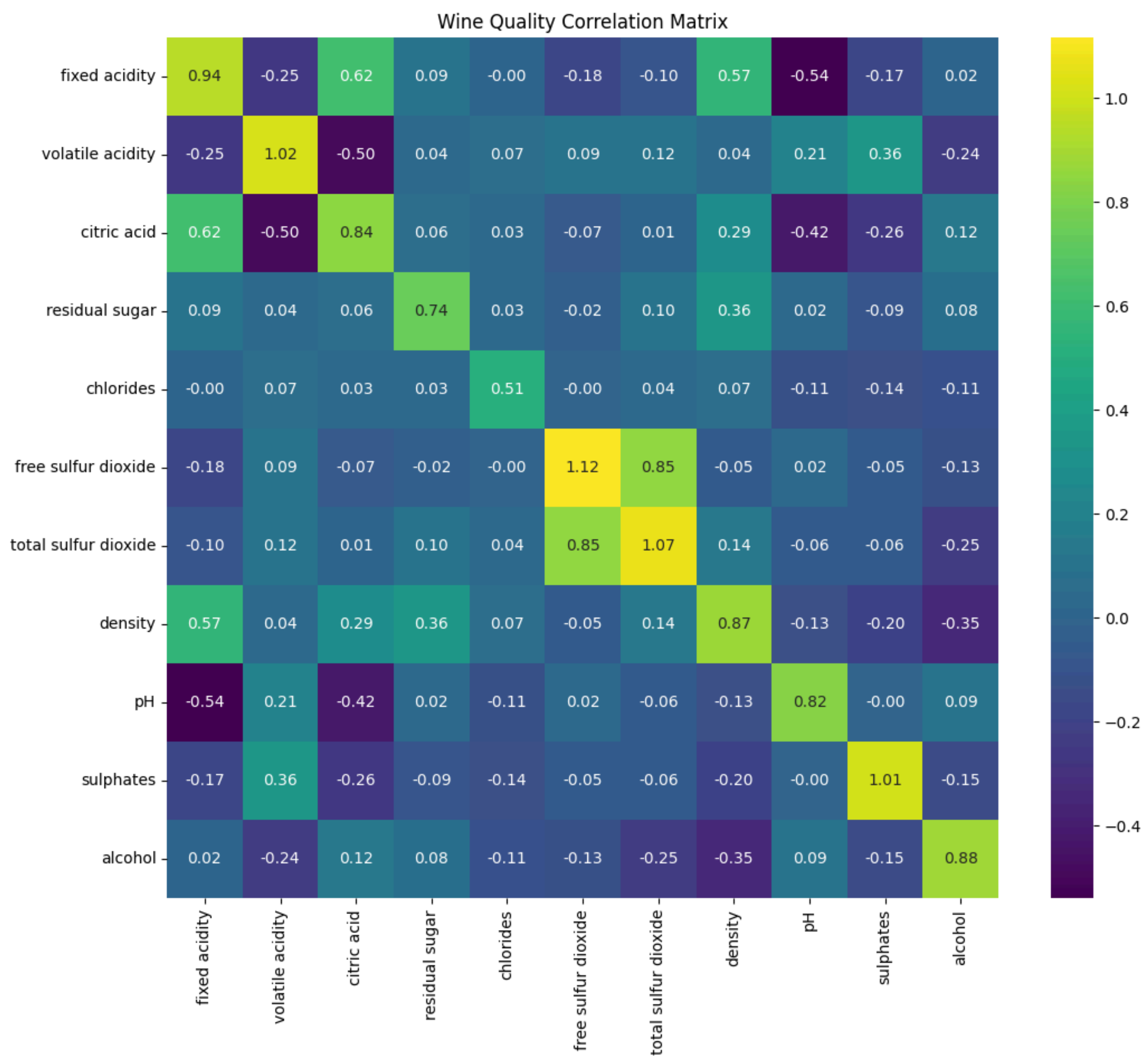
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.270467	-0.662167	1.877624	-0.187404	0.107817	-0.517499	0.196093	0.974621	1.465310	-0.226912	0.264711
1	1.982070	-0.831797	1.419809	-0.315140	-0.354194	-1.386497	-0.943020	0.974621	-1.286937	0.070212	0.537108
2	1.582696	-0.605624	1.012862	0.323600	-0.060187	-0.517499	-0.107356	0.324679	-0.454862	-0.884830	1.409417
3	-1.555242	0.072895	-1.174477	-0.895358	-0.816205	-1.386497	-1.487984	-1.331624	2.617414	0.408319	0.993326
4	0.441627	-1.284143	0.351574	-0.449109	-0.501197	-0.092183	-0.548917	-0.828443	-0.326851	-1.213789	1.184903

Next steps: Generate code with X_test_scaled   View recommended plots   New interactive sheet

Updated Covariance Matrix

Since our values are now scaled, our covariance matrix will tell us more about the data and look much closer to the correlation matrix on the non-scaled data

```
plt.figure(figsize=(12, 10))
sns.heatmap(X_test_scaled.cov(), annot=True, fmt=".2f", cmap='viridis')
plt.title('Wine Quality Correlation Matrix')
plt.show()
```

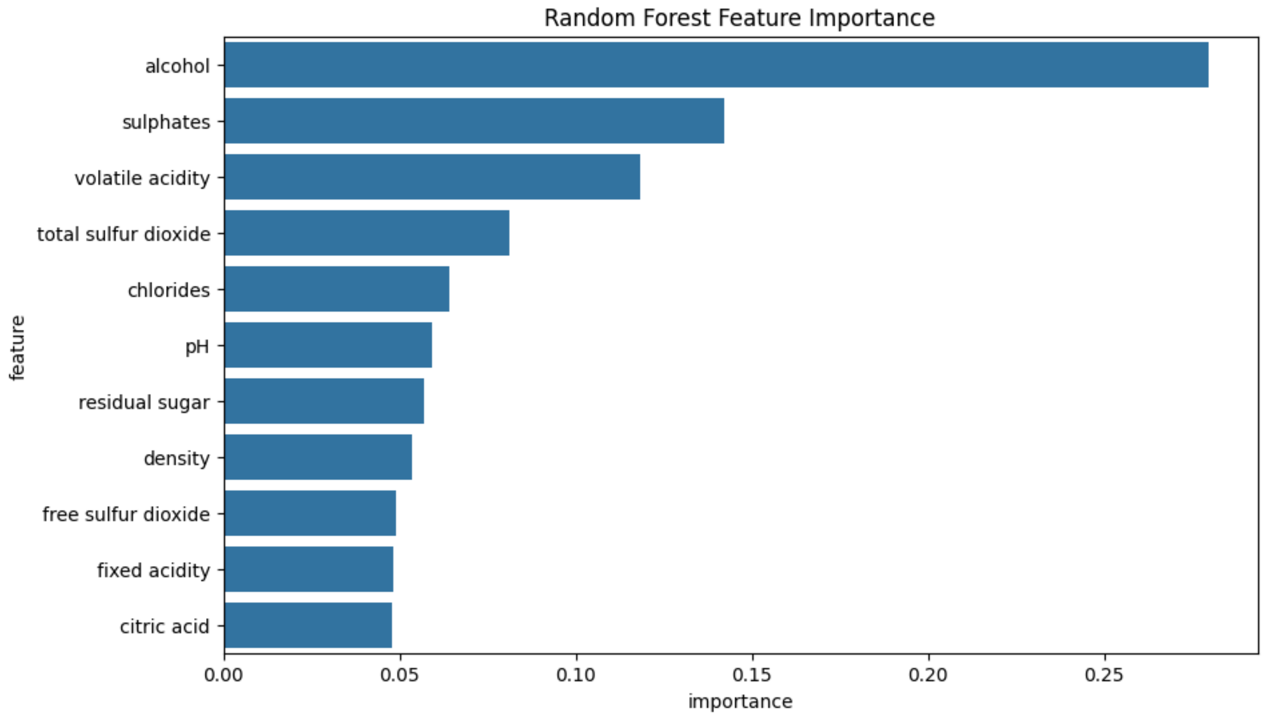


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity	0.94	-0.25	0.62	0.09	-0.00	-0.18	-0.10	0.57	-0.54	-0.17	0.02
volatile acidity	-0.25	1.02	-0.50	0.04	0.07	0.09	0.12	0.04	0.21	0.36	-0.24
citric acid	0.62	-0.50	0.84	0.06	0.03	-0.07	0.01	0.29	-0.42	-0.26	0.12
residual sugar	0.09	0.04	0.06	0.74	0.03	-0.02	0.10	0.36	0.02	-0.09	0.08
chlorides	-0.00	0.07	0.03	0.03	0.51	-0.00	0.04	0.07	-0.11	-0.14	-0.11
free sulfur dioxide	-0.18	0.09	-0.07	-0.02	-0.00	1.12	0.85	-0.05	0.02	-0.05	-0.13
total sulfur dioxide	-0.10	0.12	0.01	0.10	0.04	0.85	1.07	0.14	-0.06	-0.06	-0.25
density	0.57	0.04	0.29	0.36	0.07	-0.05	0.14	0.87	-0.13	-0.20	-0.35
pH	-0.54	0.21	-0.42	0.02	-0.11	0.02	-0.06	-0.13	0.82	-0.00	0.09
sulphates	-0.17	0.36	-0.26	-0.09	-0.14	-0.05	-0.06	-0.20	-0.00	1.01	-0.15
alcohol	0.02	-0.24	0.12	0.08	-0.11	-0.13	-0.25	-0.35	0.09	-0.15	0.88

Random Forest for feature importance

```
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
model = RandomForestRegressor(n_estimators=100, random_state=1)
# If classification, use:
# model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
model.fit(X, y)

# Get feature importances
importances = model.feature_importances_
feature_names = X.columns
feature_importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': importances
}).sort_values('importance', ascending=False)

print(feature_importance_df)
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance_df)
plt.title('Random Forest Feature Importance')
plt.show()
```

```
                feature  importance
10              alcohol    0.279491
9             sulphates    0.142195
1      volatile acidity    0.118382
6  total sulfur dioxide    0.081100
4             chlorides    0.064307
8                    pH    0.059135
3        residual sugar    0.056924
7               density    0.053345
5   free sulfur dioxide    0.049042
0         fixed acidity    0.048168
2           citric acid    0.047911
```



All of our features seem to be at least somewhat important to predicting `quality`, so we will leave our features as is for now, and focus on feature engineering more later on.

Work Planned for November 8th to November 15th

The project focus will shift to model selection and preliminary testing, where we will develop an initial deep neural network model using TensorFlow's Keras API. This stage will involve defining the network architecture, selecting activation functions, and running early tests to evaluate the model's initial performance on the dataset. We will validate the dataset for compatibility with the neural network and assess any core issues or necessary adjustments.

We also need to perform one-hot encoding on our `quality` classes, since they go from 3 to 8, instead of starting at zero.

Initial Model selection, compilation, training, and sharing accuracy

We will begin with a simple DNN model trained on one layer, and potentially add as we go. We will use ReLU activation at first, but we will experiment with different activation functions.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

#n_classes = y_train.shape[1] #currently not used
n_features = X_train.shape[1]

#initializing the model design
model = keras.Sequential([
    layers.Input(shape=(n_features,)),
    # Hidden Layer
    layers.Dense(64, activation='relu'),
    layers.Dense(1)  # Output layer for regression (linear by default), n_classes, activation = 'softmax' for classification
])

'''
Compile the model for training. Since this is a regression
problem, we use the 'mse' loss function and 'mae' as
the desired performance metric.
'''
opt = 'adam'
model.compile(loss='mse',
              optimizer=opt,
              metrics=['mae'])
```

Initial Model Training

Before training the model, we need to partition our training data in validation and training.

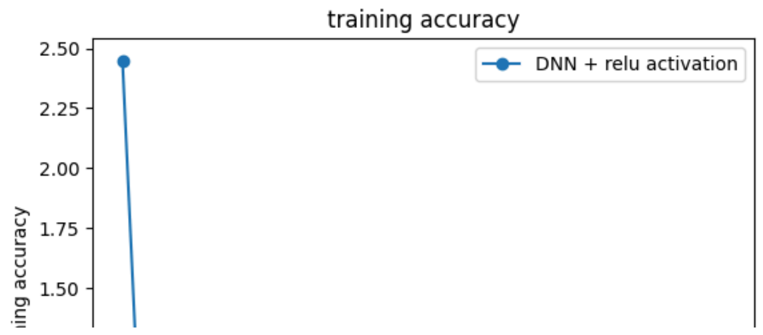
```
# partition training set into training and validation set
X_validate = X_train[1000:]
X_train = X_train[:1000]
y_validate = y_train[1000:]
y_train = y_train[:1000]

print(f"X_train has shape {X_train.shape}")
print(f"y_train has shape {y_train.shape}")
print(f"X_validate has shape {X_validate.shape}")
print(f"y_validate has shape {y_validate.shape}")
```

```
X_train has shape (1000, 11)
y_train has shape (1000,)
X_validate has shape (276, 11)
y_validate has shape (276,)
```

```
#change the values from going
```

```
history_init_model = model.fit(
    X_train, y_train,
    epochs=35,
    validation_data=(X_validate, y_validate),
    batch_size=32,
    verbose=1
)

# Plot training accuracy
plt.plot(history_init_model.history['mae'], 'o-', label='DNN + {} activation'.format('relu'))
plt.title('training accuracy')
plt.ylabel('training accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()

#plot validation accuracy
plt.plot(history_init_model.history['val_mae'], 'o-', label='DNN + {} activation'.format('relu'))
plt.title('validation accuracy')
plt.ylabel('validation accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()
```

```
Epoch 1/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - loss: 12.8512 - mae: 3.3354 - val_loss: 1.0198 - val_mae: 0.8070
Epoch 2/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 1.1617 - mae: 0.8653 - val_loss: 0.9545 - val_mae: 0.7814
Epoch 3/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.0596 - mae: 0.8267 - val_loss: 0.9000 - val_mae: 0.7603
Epoch 4/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9662 - mae: 0.7943 - val_loss: 0.8797 - val_mae: 0.7520
Epoch 5/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.9722 - mae: 0.8025 - val_loss: 0.8793 - val_mae: 0.7523
Epoch 6/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.9597 - mae: 0.7913 - val_loss: 0.8427 - val_mae: 0.7373
Epoch 7/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.9449 - mae: 0.7887 - val_loss: 0.8222 - val_mae: 0.7296
Epoch 8/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.8980 - mae: 0.7706 - val_loss: 0.8055 - val_mae: 0.7240
Epoch 9/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.8965 - mae: 0.7674 - val_loss: 0.8081 - val_mae: 0.7256
Epoch 10/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.9013 - mae: 0.7582 - val_loss: 0.7707 - val_mae: 0.7108
Epoch 11/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7868 - mae: 0.7269 - val_loss: 0.7543 - val_mae: 0.7039
Epoch 12/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7515 - mae: 0.7143 - val_loss: 0.7402 - val_mae: 0.6972
Epoch 13/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.8512 - mae: 0.7380 - val_loss: 0.7296 - val_mae: 0.6949
Epoch 14/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7693 - mae: 0.7143 - val_loss: 0.7290 - val_mae: 0.6946
Epoch 15/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7329 - mae: 0.7015 - val_loss: 0.6901 - val_mae: 0.6796
Epoch 16/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7125 - mae: 0.6943 - val_loss: 0.6720 - val_mae: 0.6722
Epoch 17/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7155 - mae: 0.6860 - val_loss: 0.6562 - val_mae: 0.6656
Epoch 18/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.7022 - mae: 0.6828 - val_loss: 0.6532 - val_mae: 0.6642
Epoch 19/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6501 - mae: 0.6632 - val_loss: 0.6284 - val_mae: 0.6528
Epoch 20/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6928 - mae: 0.6865 - val_loss: 0.6341 - val_mae: 0.6527
Epoch 21/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.6411 - mae: 0.6618 - val_loss: 0.6112 - val_mae: 0.6421
Epoch 22/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.5566 - mae: 0.6148 - val_loss: 0.5933 - val_mae: 0.6308
Epoch 23/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.5870 - mae: 0.6356 - val_loss: 0.5916 - val_mae: 0.6221
Epoch 24/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6032 - mae: 0.6432 - val_loss: 0.5806 - val_mae: 0.6208
Epoch 25/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5834 - mae: 0.6288 - val_loss: 0.5706 - val_mae: 0.6126
Epoch 26/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6172 - mae: 0.6277 - val_loss: 0.6486 - val_mae: 0.6454
Epoch 27/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.5792 - mae: 0.6018 - val_loss: 0.5595 - val_mae: 0.6028
Epoch 28/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5567 - mae: 0.6078 - val_loss: 0.5530 - val_mae: 0.5964
Epoch 29/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5160 - mae: 0.5861 - val_loss: 0.5760 - val_mae: 0.6081
Epoch 30/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5843 - mae: 0.6246 - val_loss: 0.5572 - val_mae: 0.5974
Epoch 31/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5387 - mae: 0.5859 - val_loss: 0.5465 - val_mae: 0.5891
Epoch 32/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6015 - mae: 0.6031 - val_loss: 0.5609 - val_mae: 0.5975
Epoch 33/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5186 - mae: 0.5643 - val_loss: 0.5613 - val_mae: 0.5963
Epoch 34/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5151 - mae: 0.5769 - val_loss: 0.5553 - val_mae: 0.5859
Epoch 35/35
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5678 - mae: 0.5920 - val_loss: 0.5480 - val_mae: 0.5878
```


training accuracy
DNN + relu activation
ing accuracy