Question 1. Prediction Model 1 - Linear Regression Model [50 marks]

Question 1a (5 marks)

TASK: Briefly explain why the Red Wine Quality dataset is suitable for linear regression analysis.

- Identify at least 3 characteristics that make this dataset appropriate for regression.
- Use 3 bullet points (one for each characteristics) to present your answer concisely.
- Your explanation should reflect your understanding of the linear regression model.

Q1a Answer:

1. **Continuous label and features:** Linear regression assumes a linear relationship between the features and a continuous output label. In the wine dataset, the output variable, quality, is measured on an interval scale and so is continuous. The input features such as pH are also continuous numerical values, which allows LR to model how changes in the independent variables such as citric acid affect the wine quality in a proportional way. This supports the use of linear regression analysis on the wine data set.

2. **No multicollinearity** Linear regression assumes that input features are not highly correlated, as multicollinearity can reduce the reliability of the generated coefficients. Our dataset has a diverse set of features, such as acidity and sulphates, which focus on different factors that affect wine quality and provide independent info to the model. Whilst a correlation may exist, the features are distinct for enough linear regression to model the individual effect of each feature reliably.

3. **Sufficiently sized data set:** Linear Regression works best on larger data sets as if it has more data to train with, then the model has more data to learn from which increases the generalisability and reliability of the predictions made. The wine data set has 1,599 records and therefore provides enough data for a reliable linear regression model.

Question 1b (15 marks)

TASK: Analyze the dataset using appropriate methods from the `pandas` and/or `matplotlib` libraries.

- Identify potential issues with the current dataset, specify which part(s) of the dataset are affected. Explain what could go wrong if the data is not properly pre-processed.
- Provide at least 2 short-code solutions demonstrating how you analyze these issues.
- Briefly explain how each code snippet helps evaluate data quality issues.

Q1b answer: Your answer here

Issue 1: Outliers

- **Reduced Model Accuracy**: Outliers can introduce noise into the data, which can negatively impact the LR model's ability to learn patterns accurately as not all the data is equally useful/relevant.
- **Overfitting:** Linear Regression is sensitive to noises and outliers, which increases the risk of overfitting. Outliers can be learned as patterns by the model which can reduce the strength of generalization on unseen data.
- **Bias**: Outliers can skew the distribution of the data and introduce bias, leading the model to make incorrect assumptions or predictions. An extreme outlier can also disproportionately influence the regression line, making it less representative of the majority of the data and so biased to the outlier.

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dfwine = pd.read_csv("winequality-red.csv")#creating wine dataframe
dfwine
```

```
Matplotlib is building the font cache; this may take a moment.
```

Out[1]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
0	7.4	0.700	0.00	1.9	0.076	NaN	34.0	0.99780	3
1	7.8	0.880	0.00	2.6	0.098	NaN	67.0	0.99680	3
2	7.8	0.760	0.04	2.3	0.092	NaN	54.0	0.99700	3
3	11.2	0.280	0.56	1.9	0.075	NaN	60.0	0.99800	3
4	7.4	0.700	0.00	1.9	0.076	NaN	34.0	0.99780	3
...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	NaN	3
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3

1599 rows × 12 columns

In [2]:
```
dfstats=dfwine.describe()#data frame of stats
dfstats
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free c
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1412.
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.
50%	7.900000	0.520000	0.260000	2.200000	0.079000	13.
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.

In [3]:
```
z = (((dfwine - dfwine.mean())/dfwine.std()).abs())#getting dataframe of z s
outliers = z>3 #dataframe of True/False where z score is above 3 (outlier)
outliers.sum()
```

Out[3]:

```
fixed acidity           12
volatile acidity        10
citric acid              1
residual sugar          30
chlorides               31
free sulfur dioxide     17
total sulfur dioxide    15
density                 16
pH                       8
sulphates               27
alcohol                  5
quality                  9
dtype: int64
```

In [4]:

```
outliers
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
0	False	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	
1594	False	False	False	False	False	False	False	False	Fa
1595	False	False	False	False	False	False	False	False	Fa
1596	False	False	False	False	False	False	False	False	Fa
1597	False	False	False	False	False	False	False	False	Fa
1598	False	False	False	False	False	False	False	False	Fa

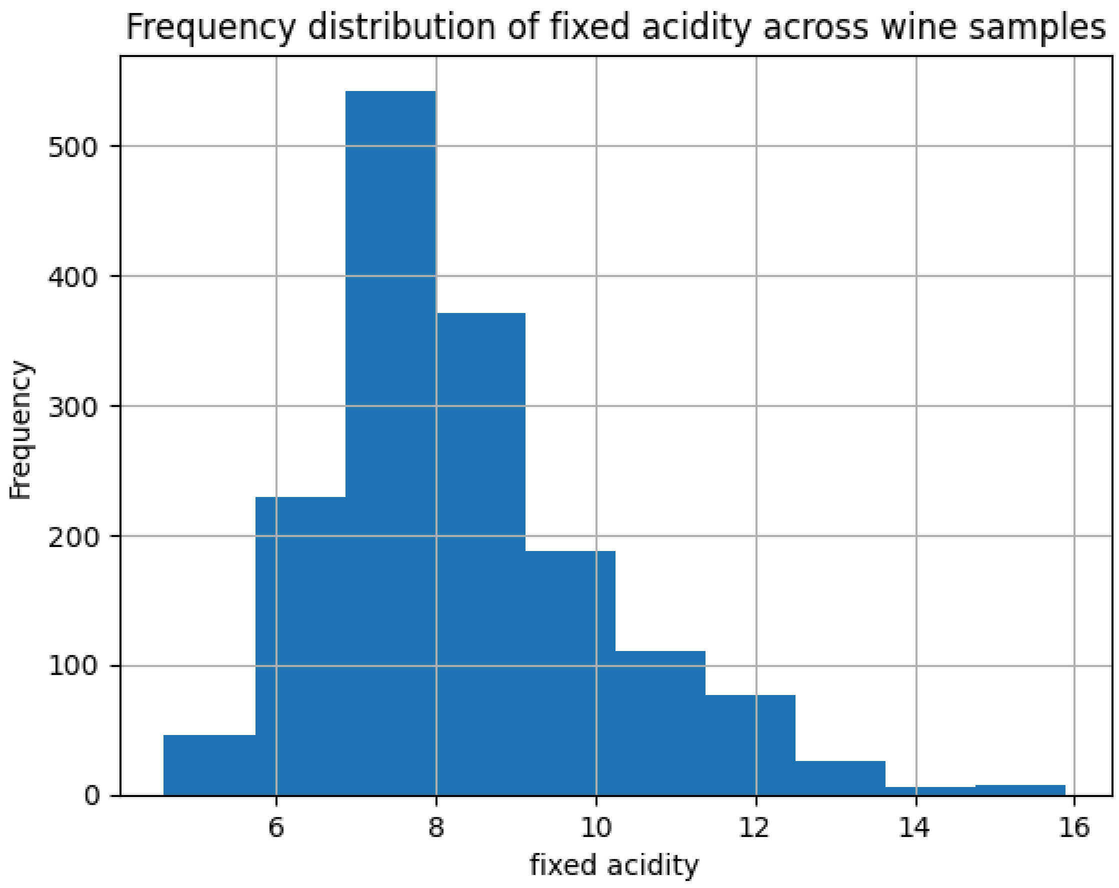
1599 rows × 12 columns

Spotting outlier values

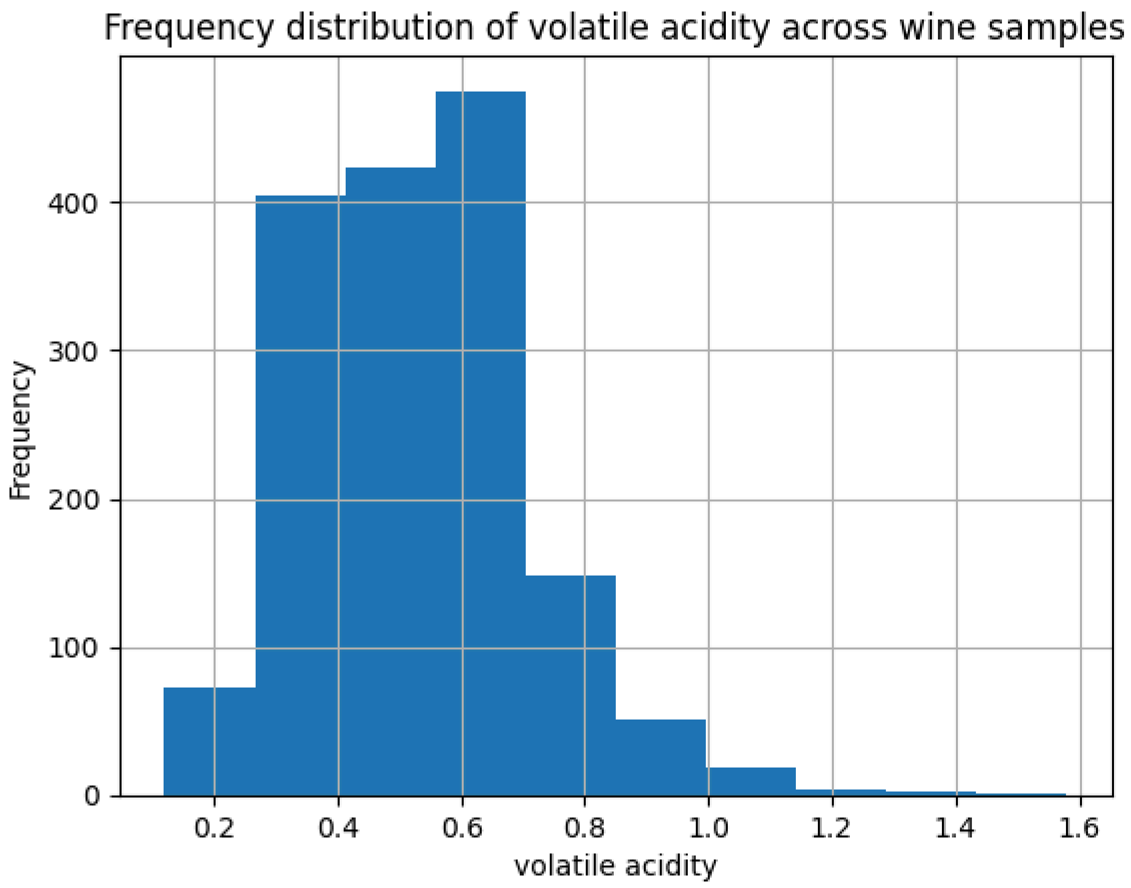
By calculating the zscore for all the separate independent variables, we are able to recognise which columns have outlier values as a zscore of above 3 or below -3 indicates that the value is an outlier. I generated a new dataframe which replaces all non outlier zscores with False and the outliers with True. I then generated how many values in each column were detected as outliers by counting how many True values were in each column. This indicated all the columns to have outliers.

In [5]:

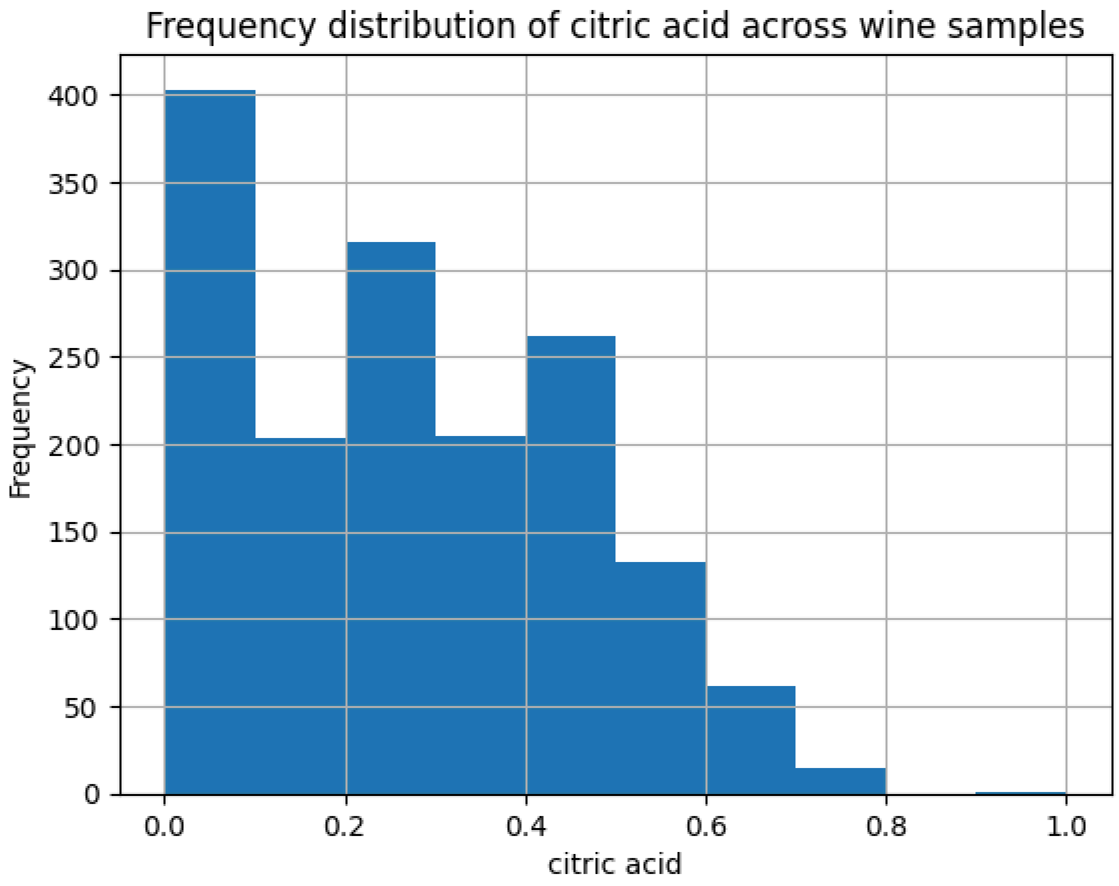
```
for column in dfwine.columns[:-1]:#for every feature not including quality
    dfwine[column].hist()#generate a histogram for that feature
    title=("Frequency distribution of "+column+" across wine samples")
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()#display histogram
    print("Skew for",column,":",dfwine[column].skew())#print the skew
    skews=dfwine.skew()
    skewdf=pd.DataFrame({'Feature':skews.index,'Skew':skews.values})
```



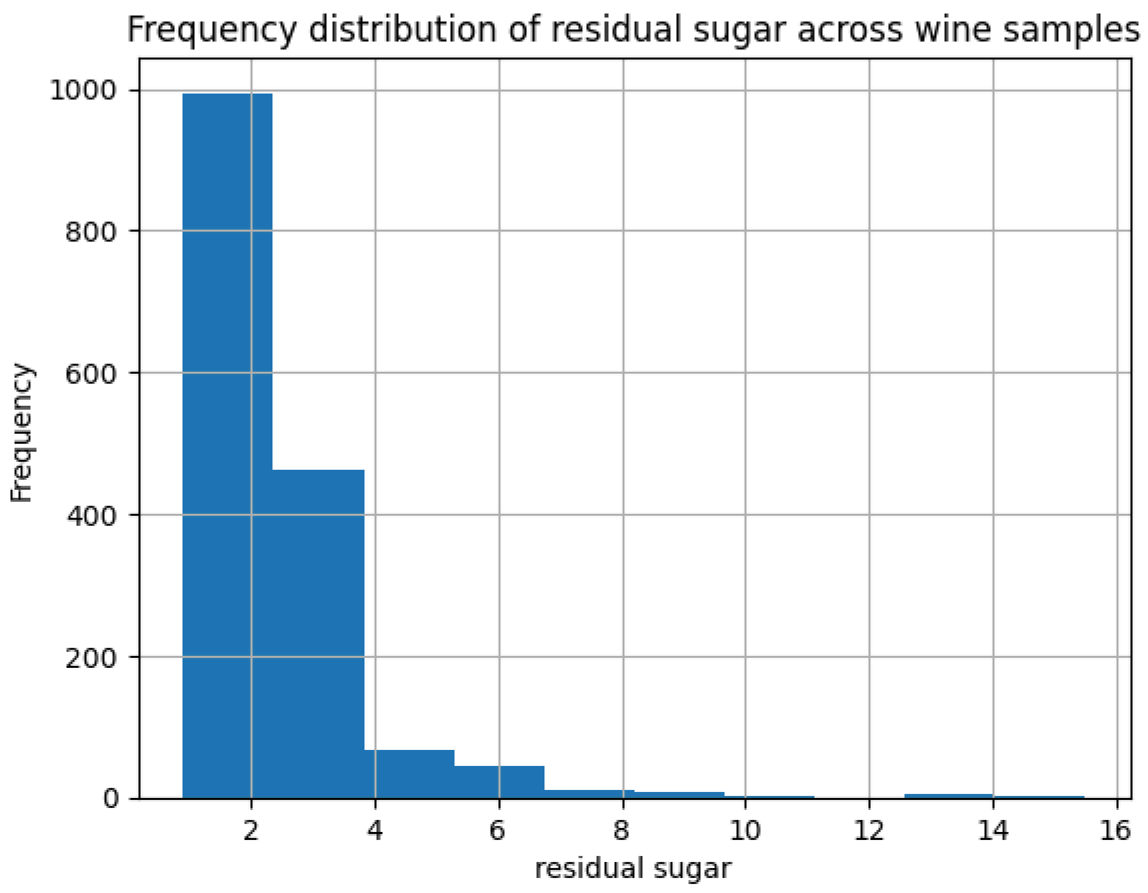
```
Skew for fixed acidity : 0.9827514413284587
```



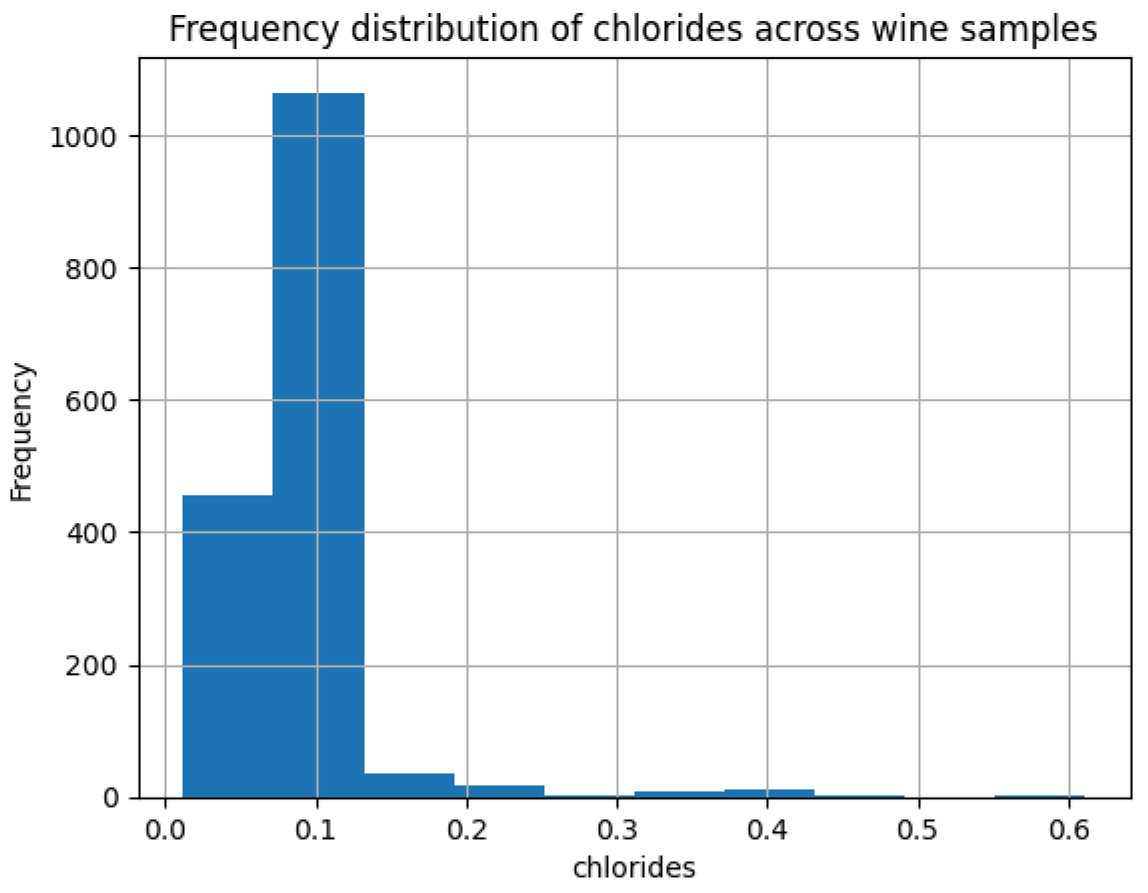
```
Skew for volatile acidity : 0.6715925723840199
```



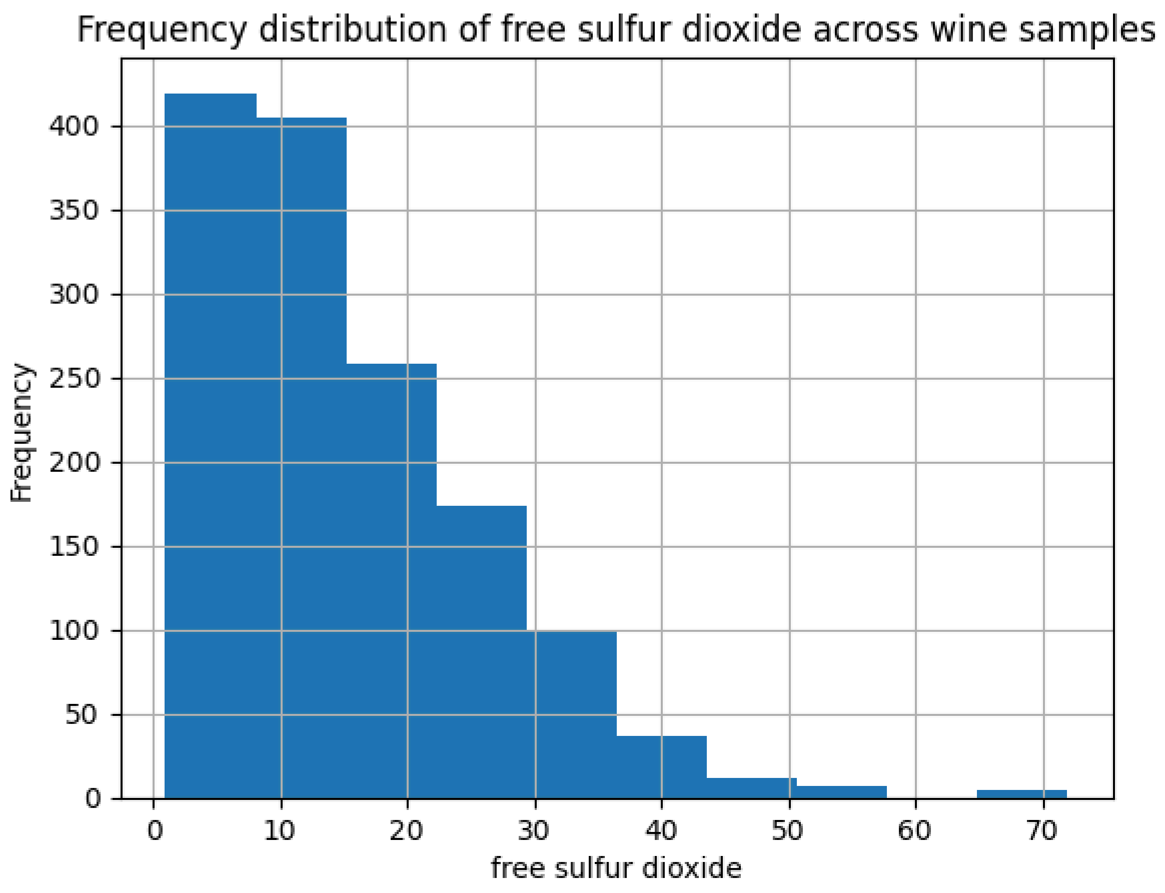
```
Skew for citric acid : 0.3183372952546368
```



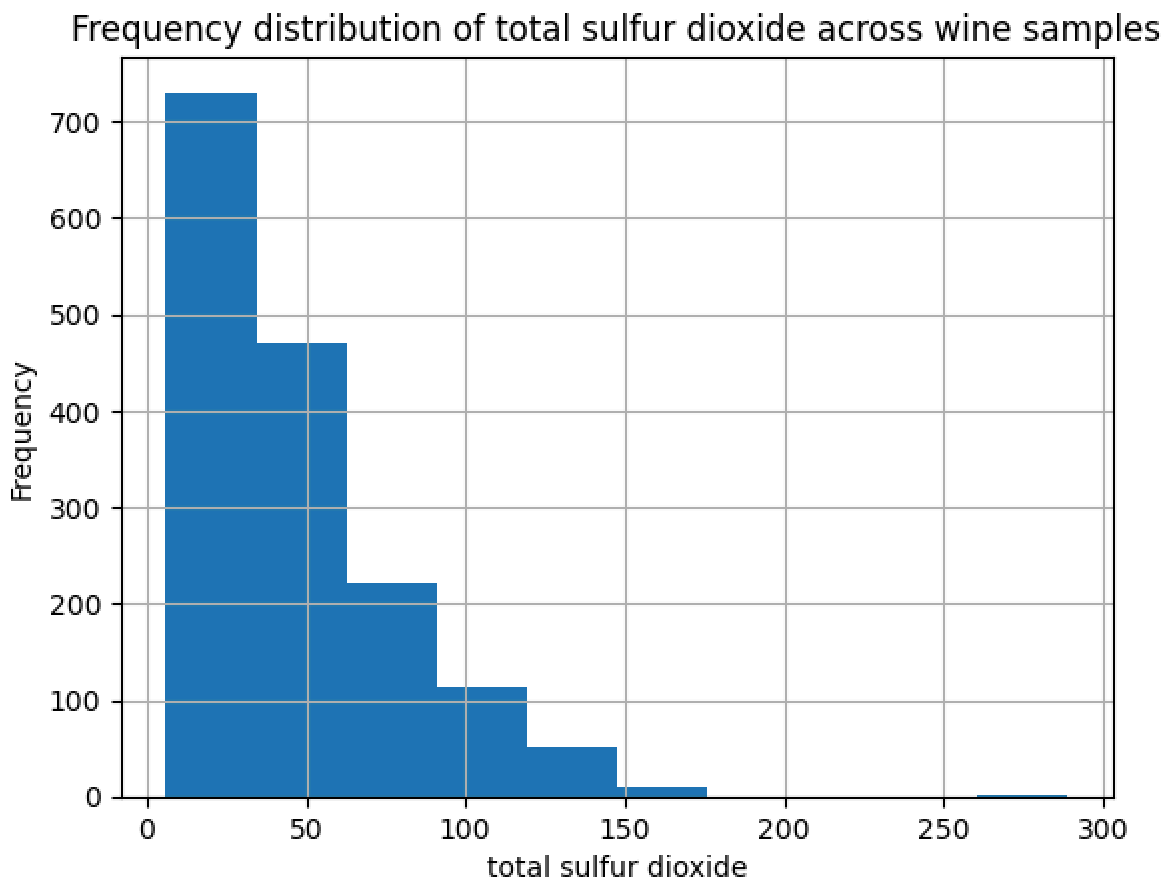
```
Skew for residual sugar : 4.54065542590319
```



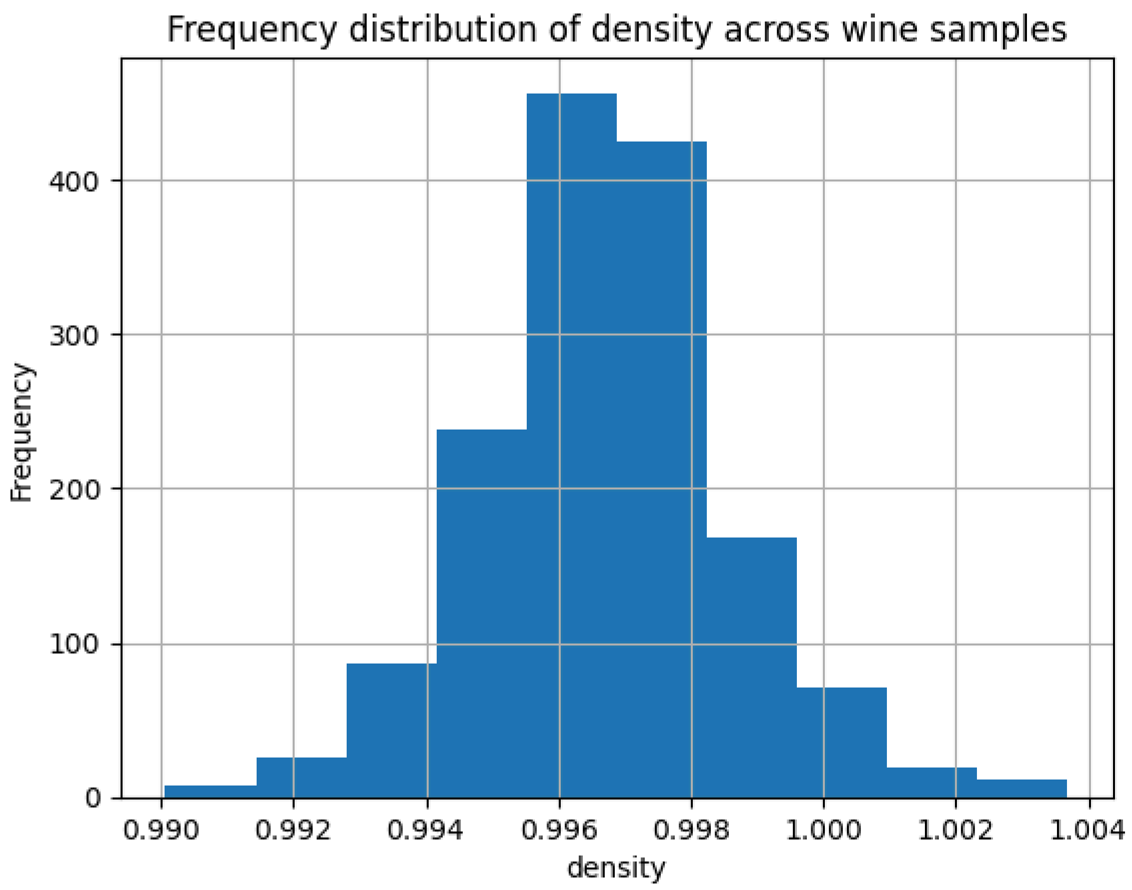
```
Skew for chlorides : 5.680346571971724
```



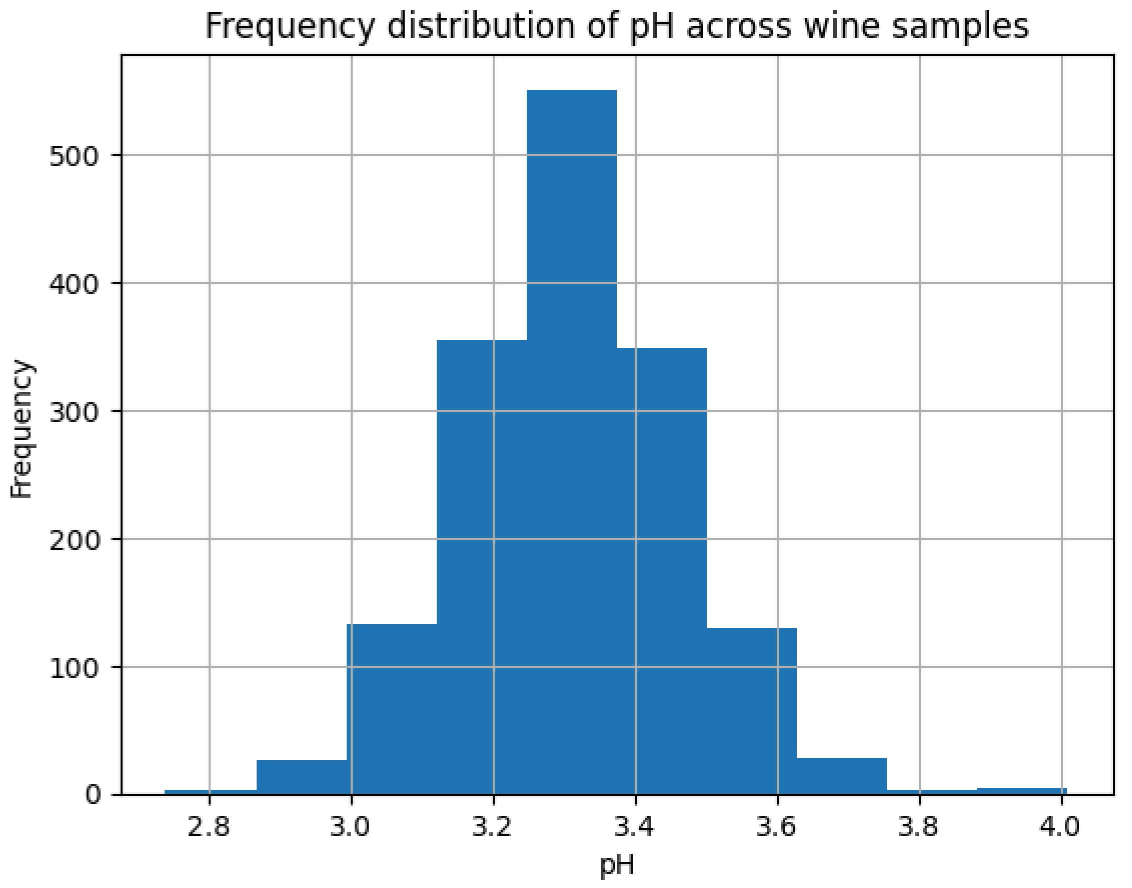
```
Skew for free sulfur dioxide : 1.2398263081961987
```



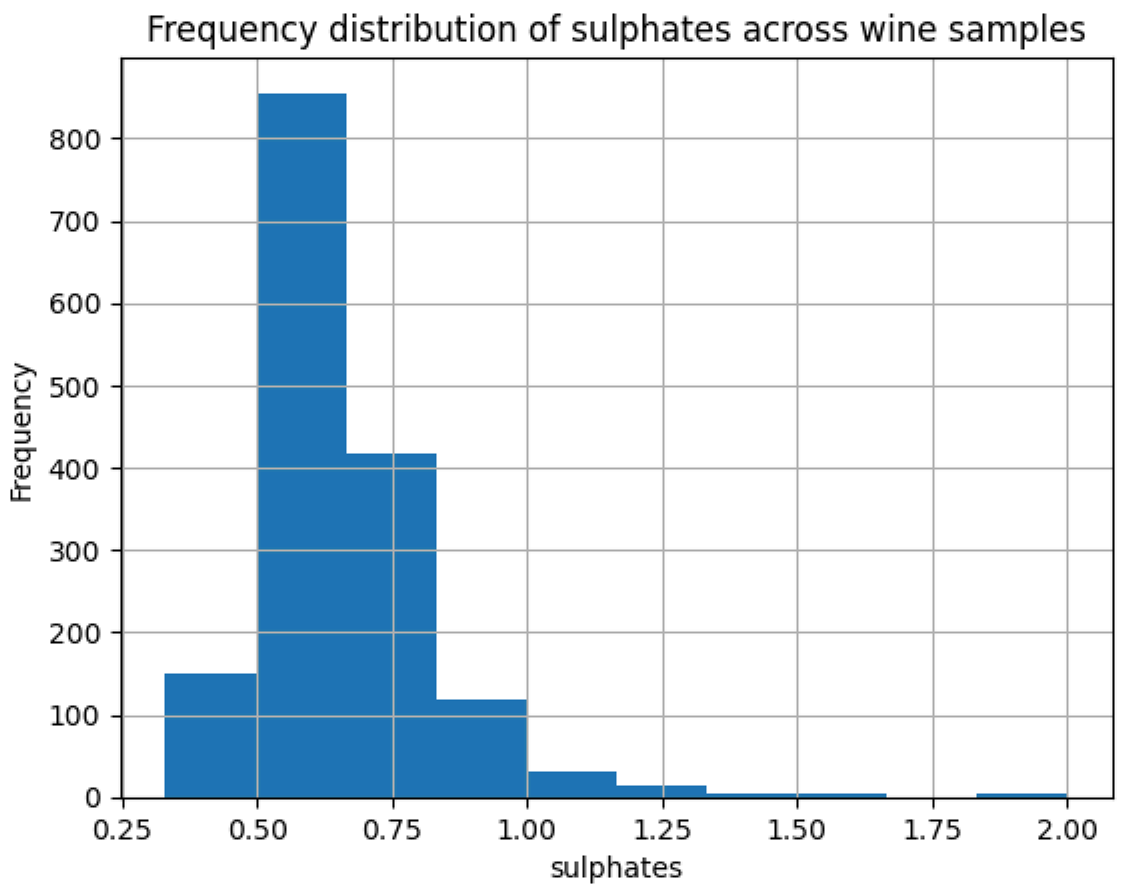
```
Skew for total sulfur dioxide : 1.515531257594554
```



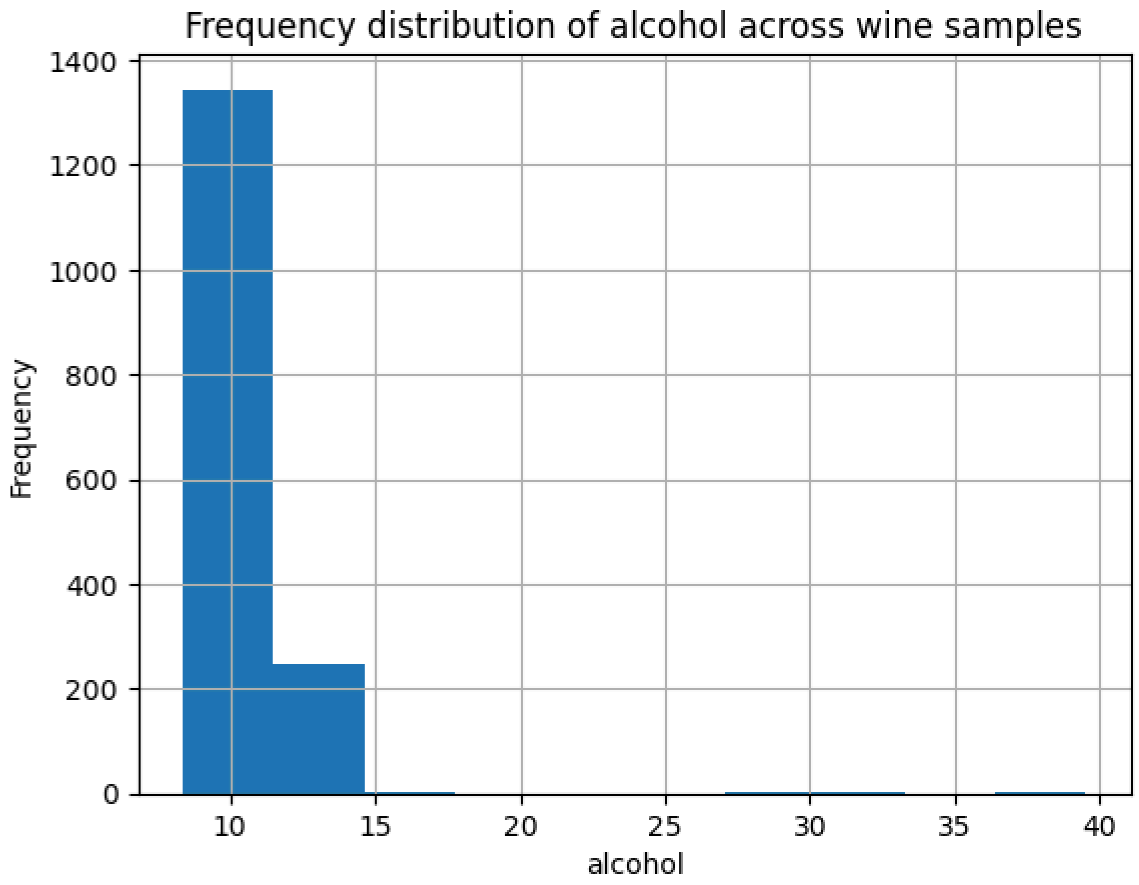
```
Skew for density : 0.08570215571193082
```



```
Skew for pH : 0.1963353095208302
```



```
Skew for sulphates : 2.4286723536602945
```



```
Skew for alcohol : 9.201220065742064
```

In [6]:
```
skewdf
```

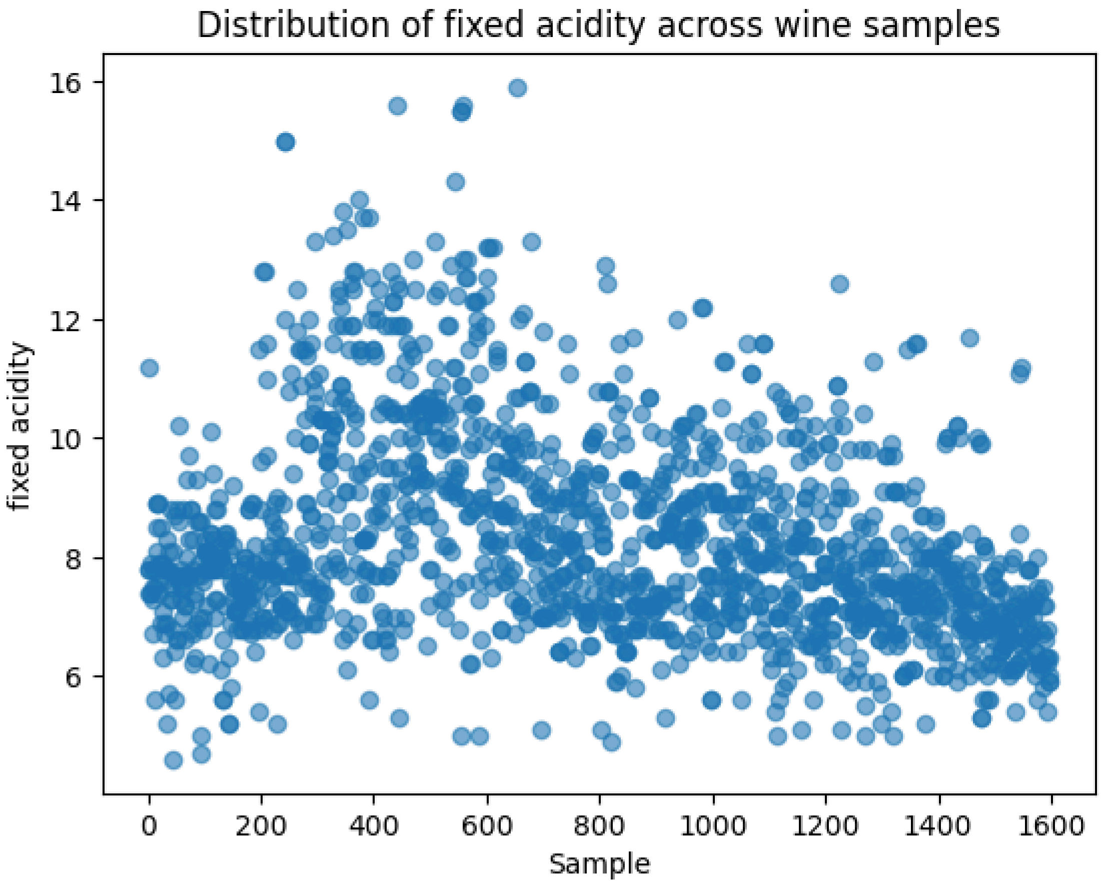
Out[6]:

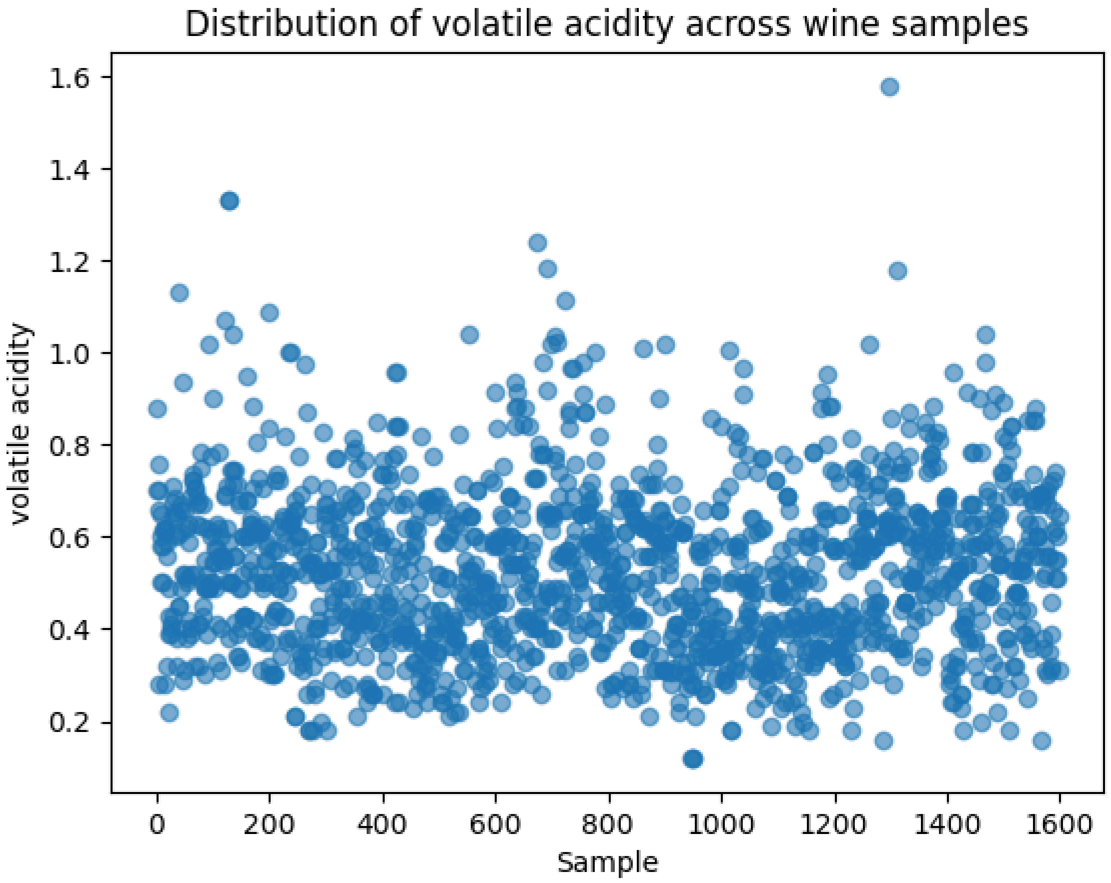
	Feature	Skew
0	fixed acidity	0.982751
1	volatile acidity	0.671593
2	citric acid	0.318337
3	residual sugar	4.540655
4	chlorides	5.680347
5	free sulfur dioxide	1.239826
6	total sulfur dioxide	1.515531
7	density	0.085702
8	pH	0.196335
9	sulphates	2.428672
10	alcohol	9.201220
11	quality	0.213963

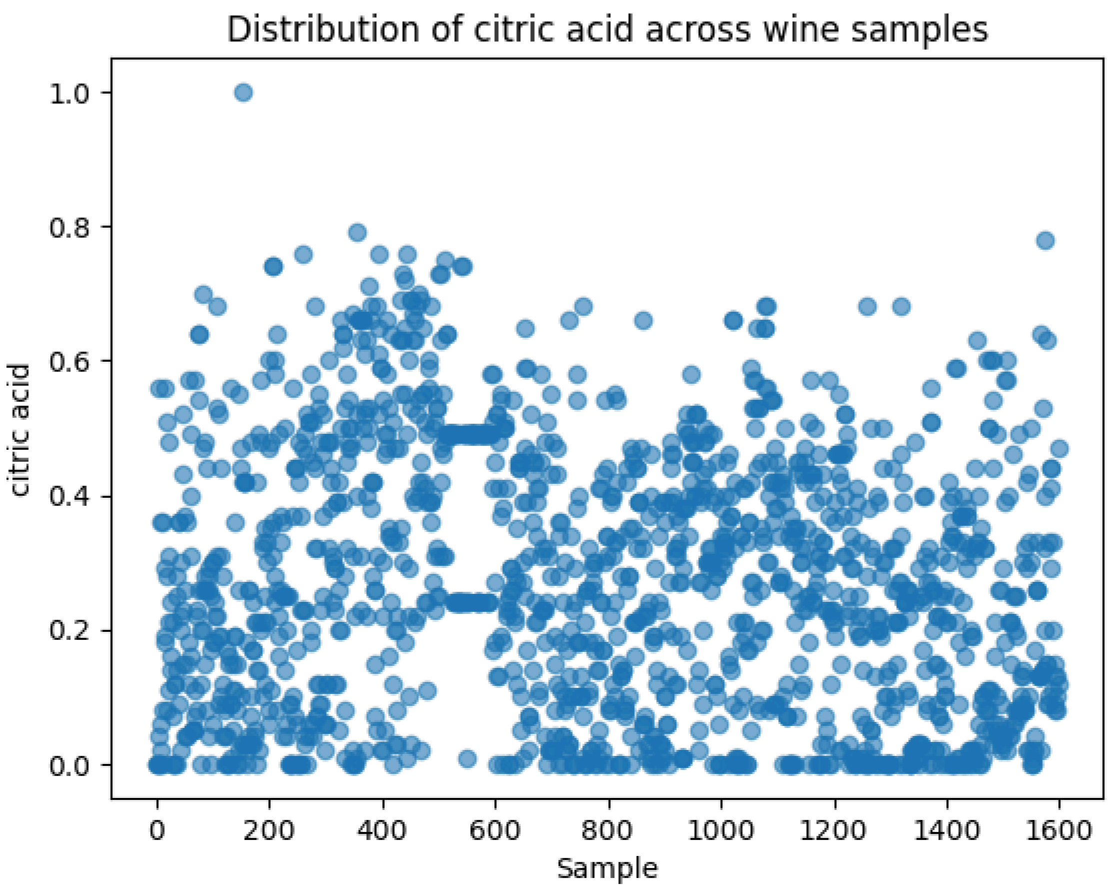
We can see from the plotted histograms that they all have a somewhat positively skewed distribution, I also displayed their skew calculation beneath each histogram.

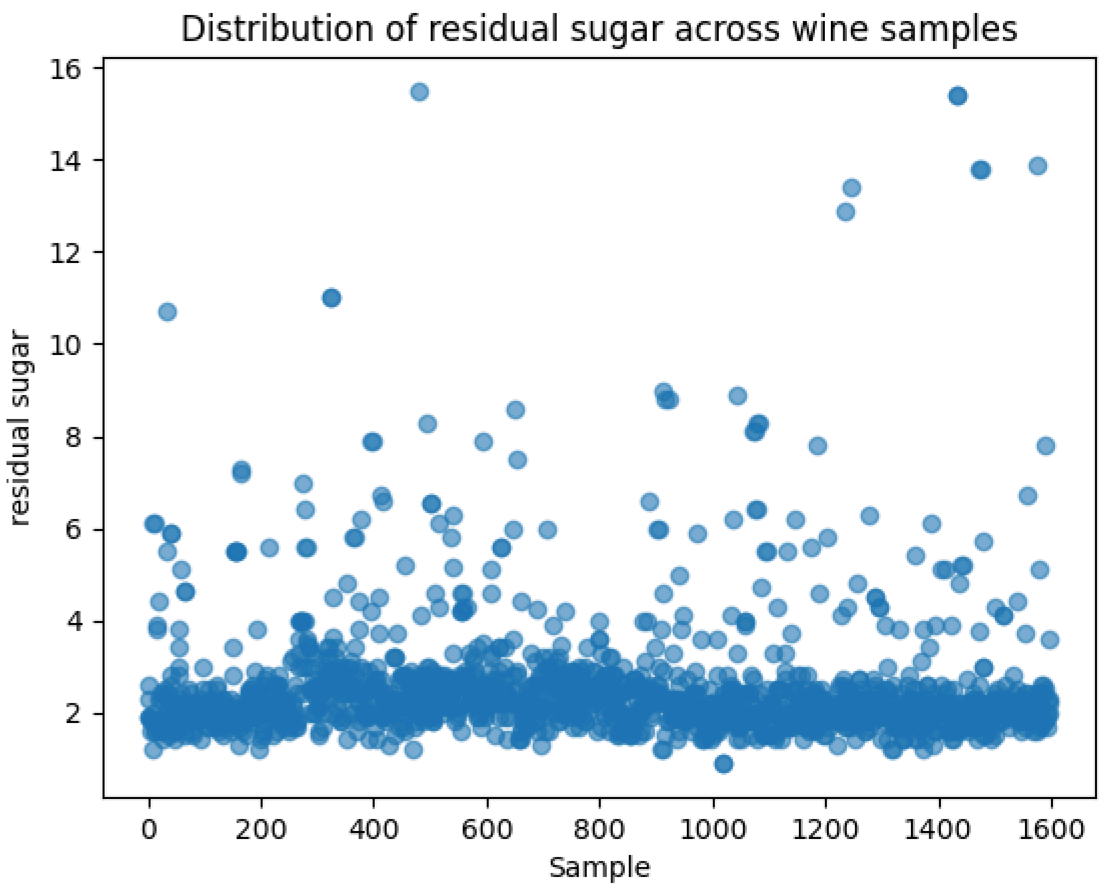
- fixed acidity and sulphates have skews greater than 0.5 but less than 1 indicating they are moderately skewed and therefore likely have some less extreme outliers
- volatile acidity, pH, density and citric acid have skews of under 0.5 and therefore are slightly skewed but not in a way too risky for LR
- residual sugar, chlorides, free sulphur dioxide, total sulphur dioxide and alcohol have a skew of over 1 and are therefore extremely skewed meaning they likely have extreme values
- residual sugars, chlorides and alcohol have very extreme skews
- alcohol has the most extreme skew of 9.201220 which is 3.520873 more than the next most extreme skew, chlorides. Alcohol only has 5 outliers, which means these must be very extreme values.

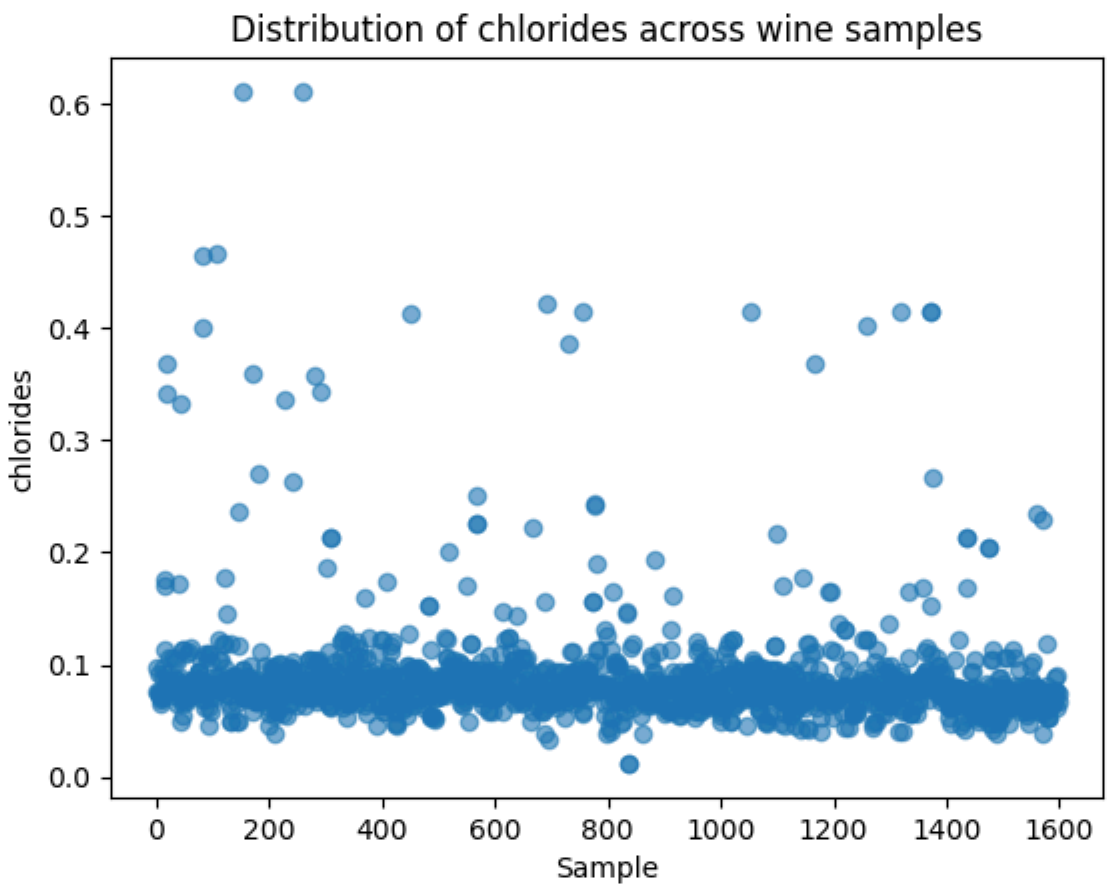
In [7]:
```
#scatter plots
for column in dfwine.columns[:-1]:#for every feature
    plt.scatter(range(len(dfwine[column])), dfwine[column], alpha=0.6)#set a
    title=("Distribution of "+column+" across wine samples")
    plt.title(title)
    plt.xlabel('Sample')
    plt.ylabel(column)
    plt.show()
```

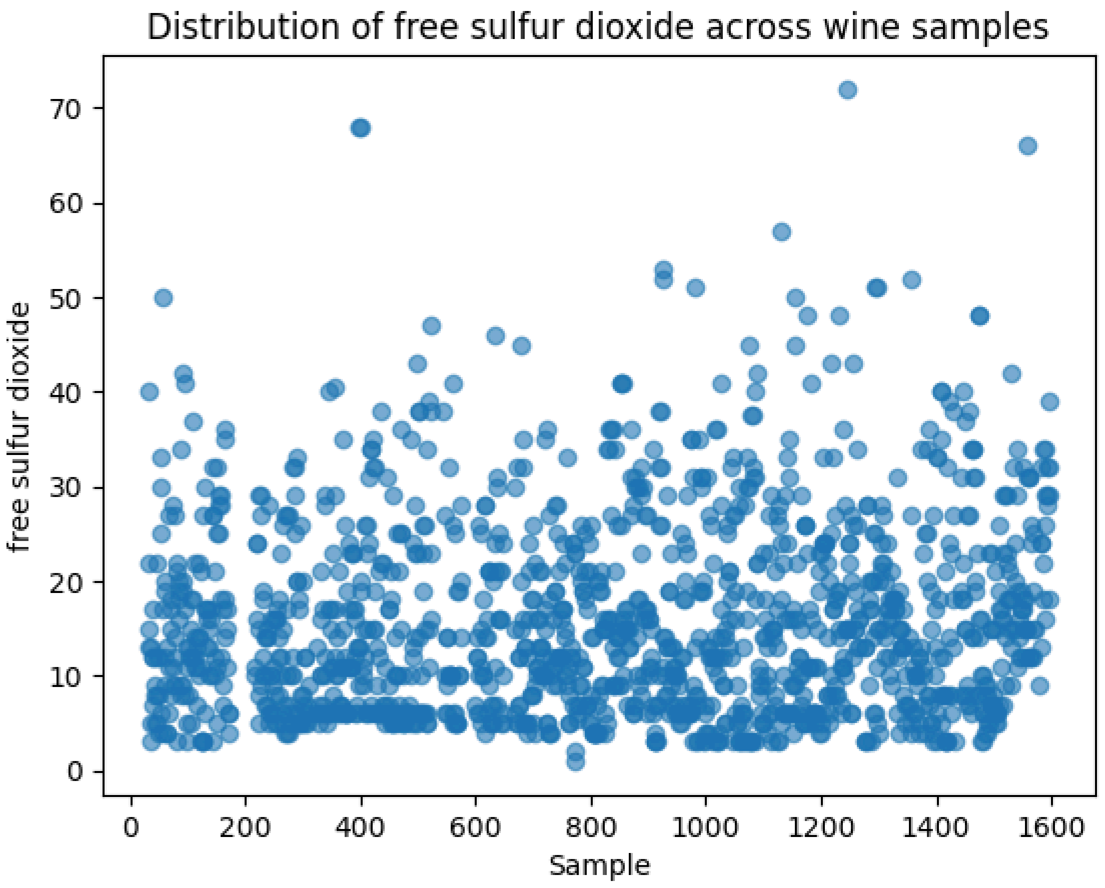

Distribution of fixed acidity across wine samples
fixed acidity
Sample
16
14
12
10
8
6
0
200
400
600
800
1000
1200
1400
1600

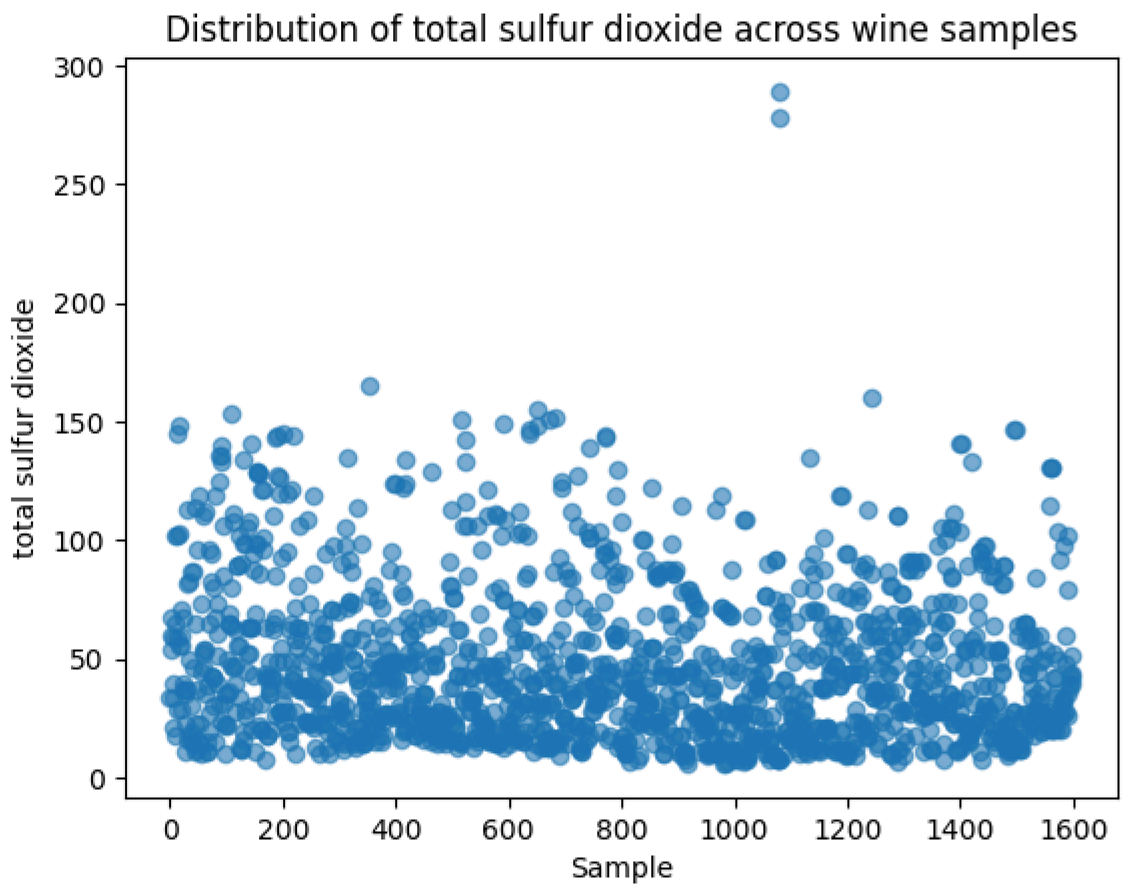

Distribution of volatile acidity across wine samples
volatile acidity
Sample
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000
1200
1400
1600

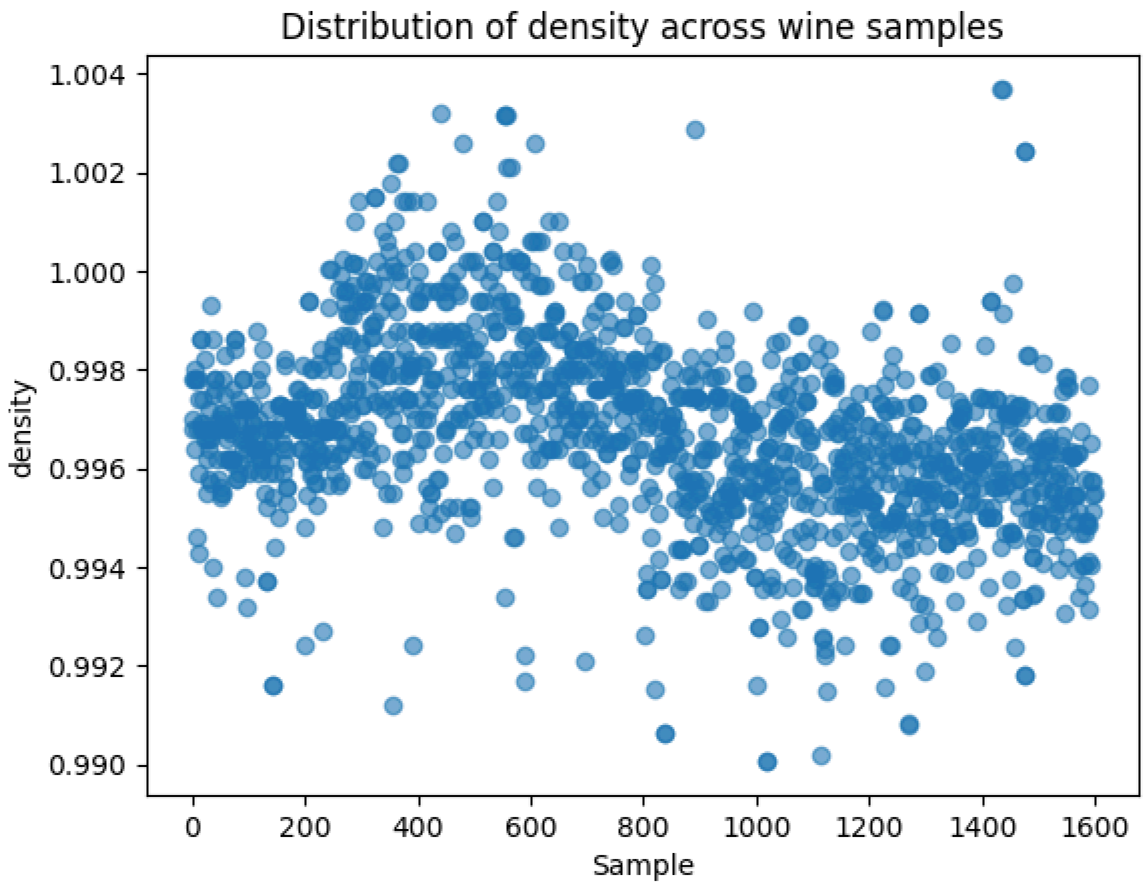

Distribution of citric acid across wine samples
citric acid
Sample
0.0
0.2
0.4
0.6
0.8
1.0
0
200
400
600
800
1000
1200
1400
1600

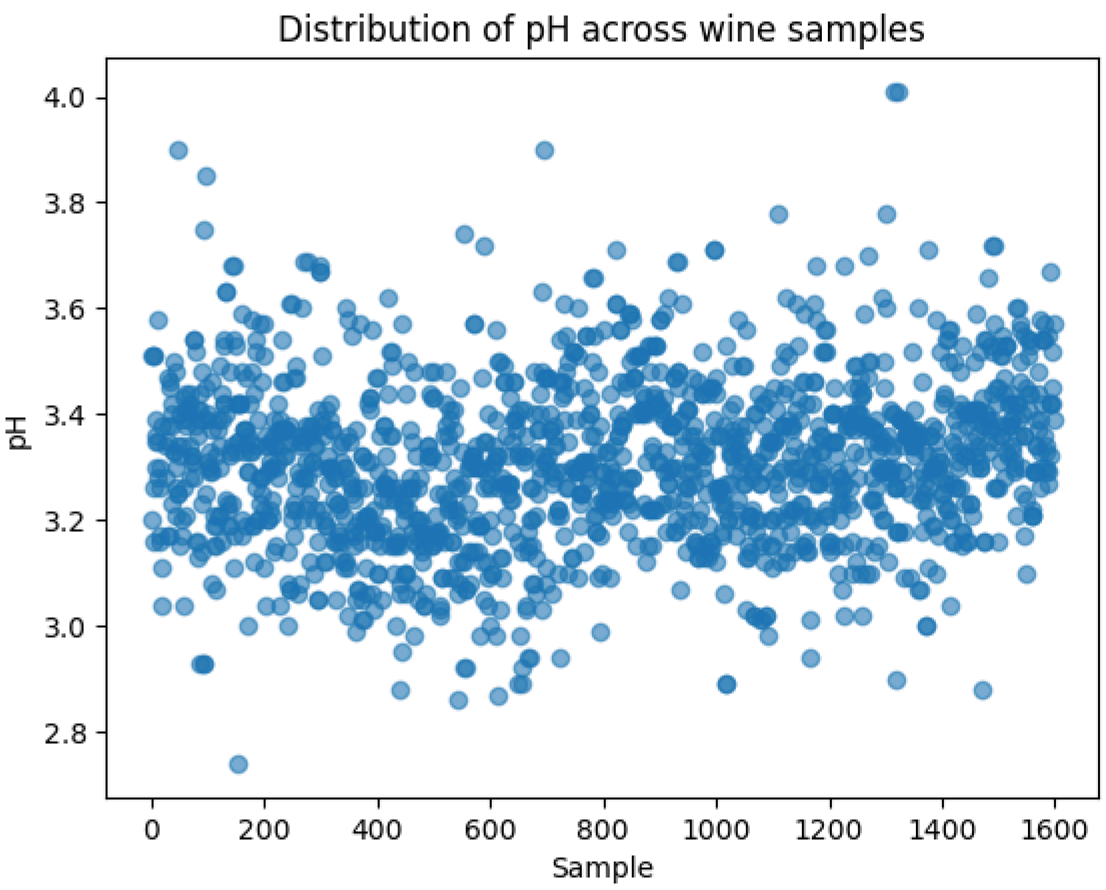

Distribution of residual sugar across wine samples
residual sugar
Sample
2
4
6
8
10
12
14
16
0
200
400
600
800
1000
1200
1400
1600

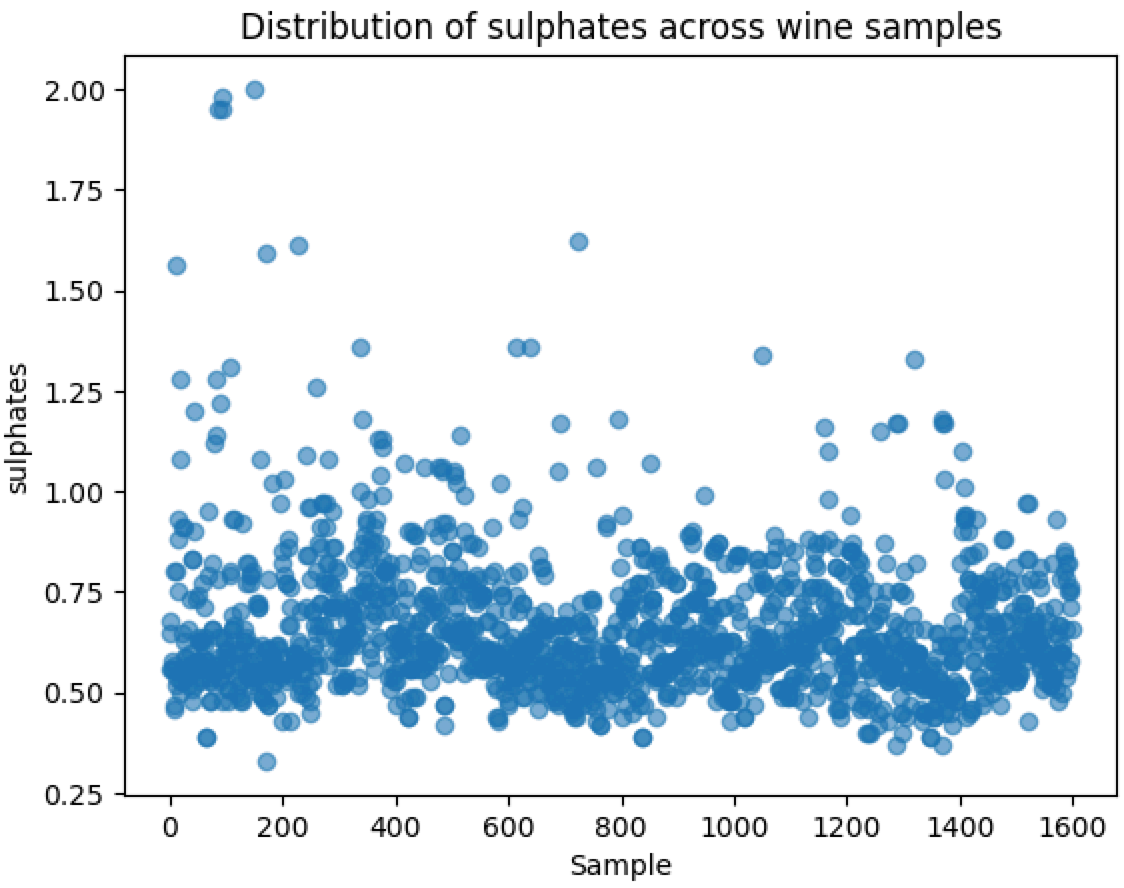

Distribution of chlorides across wine samples
chlorides
Sample
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
200
400
600
800
1000
1200
1400
1600

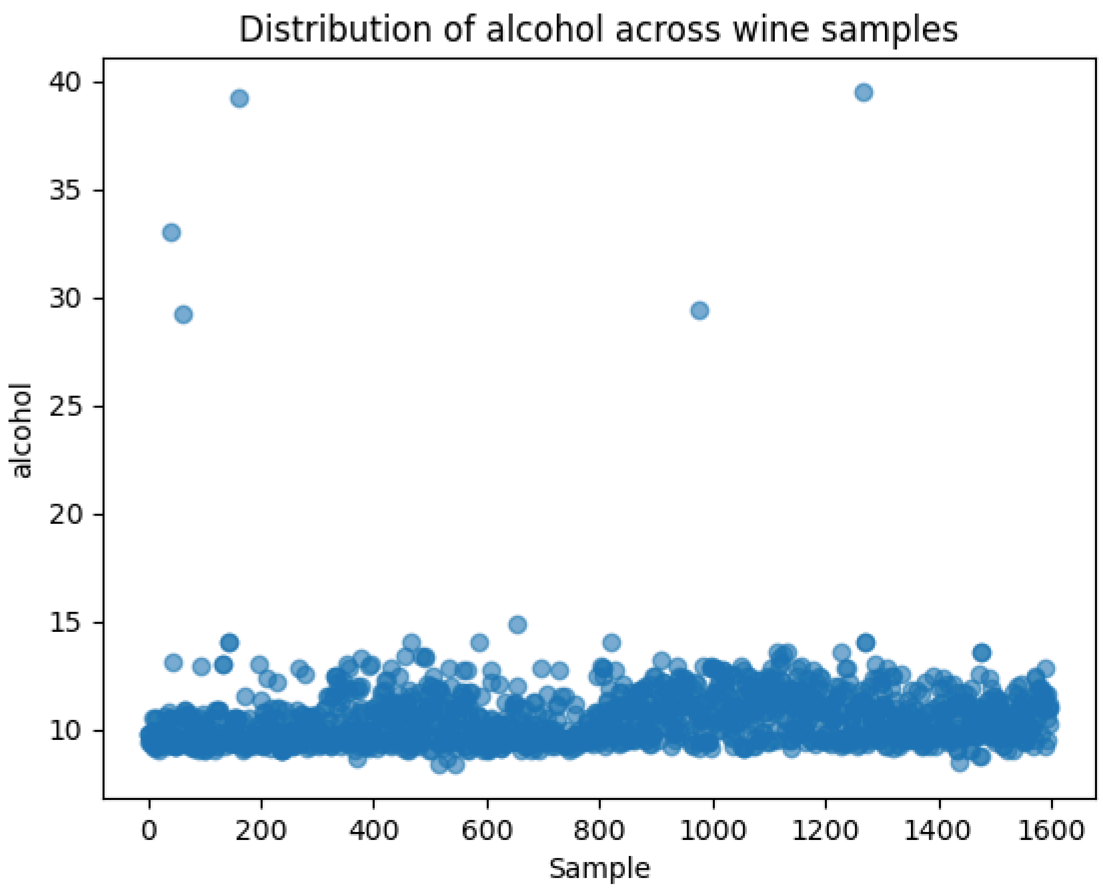

Distribution of free sulfur dioxide across wine samples
free sulfur dioxide
Sample
0
10
20
30
40
50
60
70
0
200
400
600
800
1000
1200
1400
1600


Distribution of total sulfur dioxide across wine samples
total sulfur dioxide
Sample
0
50
100
150
200
250
300
0
200
400
600
800
1000
1200
1400
1600


Distribution of density across wine samples
density
Sample
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
0
200
400
600
800
1000
1200
1400
1600


Distribution of pH across wine samples
pH
4.0
3.8
3.6
3.4
3.2
3.0
2.8
0
200
400
600
800
1000
1200
1400
1600
Sample


Distribution of sulphates across wine samples
sulphates
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
200
400
600
800
1000
1200
1400
1600
Sample



The scatter graphs show us every value and we can now visually see the extreme values/outliers. For example our most extreme skew, alcohol, clearly has 5 extreme values, with 2 being on the very top of the graph at around 40, compared to most of the values at 0 - 15. This validates our previous analysis.

Issue 2: Missing Values

Missing values make the linear regression calculation less useful as it prevents the model from learning relationships. It also may not be representative of complete data sets and so also less generalisable. The functions used to train the model like fit() also don't work with missing values so the model cannot be trained to begin with.

Spotting NULL/missing values

By using the .innull().sum function we can see how many 'non-null' values are in each column. The results show how many values of each feature are NULL values/ missing data.

```
In [8]: dfwine.isnull().sum()
```

```
Out[8]: fixed acidity             0
        volatile acidity          0
        citric acid               0
        residual sugar            0
        chlorides                 0
        free sulfur dioxide     187
        total sulfur dioxide      0
        density                  91
        pH                       19
        sulphates                 0
        alcohol                   0
        quality                   0
        dtype: int64
```

Only free sulphur dioxide, density and pH have missing values.

Question 1c (20 marks)

TASK: Apply appropriate data preprocessing techniques to address the issues identified in Question 1b.

- Provide a code solution that resolves the identified data issue(s).
- Briefly explain the methods and parameters used in your solution. Ensure your explanation clearly justifies how these techniques improve data quality and suitability for analysis.

Q1c answer:

Preprocessing Missing Values

```
In [9]: #replace all null values with the median of each column
        for column in dfwine.columns:
            column_median=dfwine[column].median()#calculating median for column
            dfwine[column] = dfwine[column].fillna(column_median)#replacing missing
        dfwine.isnull().sum()
```

```
Out[9]: 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
        dtype: int64
```

There are alot of missing values for both density and and free sulfur dioxide so removing them may cause important information to be lost. However there are only 19 missing values for pH out of 1599 values which is around 1%. Therefore removing them is likely to be less detrimental. But instead I decided to replace all the NULLs with the average of their column as the other values in the row may still hold important information. This maintains data quality as it maintains the large sample size. Since all features are continious, the median is the most appropriate as all three features are somewhat skewed so the mean would be influenced by the outliers. The median preserves the overall data distribution which avoids accidentally introducing bias which could decrease the data's suitability for analysis.

Preprocessing Outlier Values

```
In [10]: #move through z df, when find an outlier, take that indices
         #replace in dfwine(in new df), that indices with median
         #outliers = z>3 #new df called outliers that only stores values where z>3 as
```

```
In [11]: #dfwine = pd.read_csv("winequality-red.csv")
         z = (((dfwine - dfwine.mean())/dfwine.std()).abs())
         outliers = z>3
         print("Outliers Before Pre-Processing \n")
         print(outliers.sum())
```

```
Outliers Before Pre-Processing 

fixed acidity           12
volatile acidity        10
citric acid              1
residual sugar          30
chlorides               31
free sulfur dioxide     19
total sulfur dioxide    15
density                 18
pH                       8
sulphates               27
alcohol                  5
quality                  9
dtype: int64
```

```
In [12]: z = (((dfwine - dfwine.mean())/dfwine.std()).abs())#recalculating z score (c
         outliers = z>3
         for column in dfwine.columns:#for every feature, replace with median
            column_median=dfwine[column].median()
            ind = dfwine[outliers[column]].index
            dfwine.loc[ind,column] = column_median
         z = (((dfwine - dfwine.mean())/dfwine.std()).abs())#recalculate now has beer
         outliers = z>3
         print("Outliers After Pre-Processing \n")
         print(outliers.sum())
```

```
Outliers After Pre-Processing

fixed acidity            5
volatile acidity         4
citric acid              0
residual sugar          54
chlorides               38
free sulfur dioxide      9
total sulfur dioxide    14
density                  7
pH                       1
sulphates               23
alcohol                  8
quality                  3
dtype: int64
```

Like the NULL values, I didn't want to remove the outlier rows completely as they may include important information, and there are also more outliers than NULL values so many rows would be removed which reduces the suitability of the data set as its size will be smaller. Due to this, I also replaced the outliers with the median as it reduces the extent to which the values pull the regression line. Since the median isn't influenced by extreme values such as the mean it helps to stabilise data distribution which increases the data's suitability for analysis. After replacing the outliers with the median, most columns appeared to decrease in the number of outliers. However, residual sugar and chlorides increased. It is impossible to fully remove all outliers, and since the number of outliers is lower than before I think this solution is adequate.

Question 1d (10 marks)

TASK: Train and evaluate a Linear Regression model using the preprocessed dataset.

- Print the model's weights.
- Print the model's accuracy.
    - Evaluate the model using at least three different metrics.
    - Briefly discuss the advantages of each metric in assessing model performance.

Q1d answer:

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
#ASSIGN FEATURES TO X(independent) y(dependent)
X = dfwine[['fixed acidity', 'volatile acidity', 'citric acid','residual sug
y = dfwine['quality']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

```
Out[13]:
▾ LinearRegression
LinearRegression()
```

```
In [14]: print(dfwine.columns)#can see features so can see corresponding coefficient
print('Coefficients/Weights:', lr_model.coef_)#gradient, influence on qualit
print('Intercept:', lr_model.intercept_)#baseline
print('\nThe linear regression function learned between X and Y is:')#relati

print(f'y = {lr_model.intercept_:.2f}', end='')
for i, coef in enumerate(lr_model.coef_):
    print(f' + {coef:.2f}*{X.columns[i]}', end='')#print out equation
print('\n')
y_pred_test_LR = lr_model.predict(X_test)#test set basis
y_pred_train_LR = lr_model.predict(X_train)#train set basis
#LINEAR REGRESSION
print('LINEAR REGRESSION MODEL \n')
print("Test\n")
mse_LR_Test = metrics.mean_squared_error(y_test, y_pred_test_LR)#how close r
print('Mean Squared Error (MSE):', mse_LR_Test)

mae_LR_Test = metrics.mean_absolute_error(y_test, y_pred_test_LR)# average a
print('Mean Absolute Error (MAE):', mae_LR_Test)

rsq_LR_Test = metrics.r2_score(y_test, y_pred_test_LR)#how well data fits re
print('Root Squared:', rsq_LR_Test)

print("\nTrain\n")
mse_LR_Train = metrics.mean_squared_error(y_train, y_pred_train_LR)
print('Mean Squared Error (MSE):', mse_LR_Train)

mae_LR_Train = metrics.mean_absolute_error(y_train, y_pred_train_LR)
print('Mean Absolute Error (MAE):', mae_LR_Train)


rsq_LR_Train = metrics.r2_score(y_train, y_pred_train_LR)
print('Root Squared:', rsq_LR_Train)
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
Coefficients/Weights: [ 3.46443557e-02 -8.82097776e-01 -3.85145596e-01  2.73
931640e-02
 -1.66396391e+00  3.97661654e-03 -3.13487817e-03 -2.25374865e+01
 -4.01130097e-01  1.38478877e+00  2.56331168e-01]
Intercept: 26.789214672085443

The linear regression function learned between X and Y is:
y = 26.79 + 0.03*fixed acidity + -0.88*volatile acidity + -0.39*citric acid
+ 0.03*residual sugar + -1.66*chlorides + 0.00*free sulfur dioxide + -0.00*t
otal sulfur dioxide + -22.54*density + -0.40*pH + 1.38*sulphates + 0.26*alco
hol

LINEAR REGRESSION MODEL

Test

Mean Squared Error (MSE): 0.4766730911152701
Mean Absolute Error (MAE): 0.5540994595944266
Root Squared: 0.2608876273573414

Train

Mean Squared Error (MSE): 0.5017439481737791
Mean Absolute Error (MAE): 0.5548488596008473
Root Squared: 0.3026837973731459
```

MSE measures how far from the actual values the predicted values are, smaller the better (squares the errors)

How well we have avoided errors

- Test: An MSE of 0.4667
- Train: An MSE of 0.5017
- Indicates the model is not overfitting as both the MSE values are very similar so generalisable
- No significant large errors/outliers

MAE measures how far from the actual values the predicted values are, smaller the better (takes absolute difference)

- Test: An MAE of 0.5441
- Train: An MAE of 0.5548
- Indicates model is consistent in performance across training and test as both values are similar so similar in error.

Since MSE and MAE are close in value it suggests the errors in the model are small and evenly distributed, the LR model's predictions aren't significantly impacted by outliers

R^2 measures strength of model's explanation of variance in target variable (quality), higher the better

- Test: An R^2 of 0.2609 indicates model isn't explaning very much of the variance in quality as has a low R^2 (under 0.5)
- Train: An R^2 of 0.3027 indicates model isn't explaning very much of the variance in quality as has a low R^2 (under 0.5) but explains more of the variance in the train set than test as better R^2 than test set.

Evaluation

MSE

Benefits:

- squares each error so the larger the error, the more significant its impact is on the final value which means large errors are more emphasised so is useful at assessing how well we have avoided them

MAE

Benefits:

- MAE does not square each error and instead it uses the absolute value of the differences between the predicted values and the actual values. This means that all errors are treated equally so the impact of outliers is minimized

Root Squared

Benefits:

- Root squared is useful, especially for linear regression models, as it tells us the 'goodness' of the regression line and therefore lets us determine how well our data fits the model. Therefore a value far from 1 indicates variance in the target variable and that there is likely issues with our data.

Question 2. Prediction Model 2 [20 marks]

Question 2a (10 marks)

TASK: Build a different machine learning model for the same prediction task.

- Choose a model covered in the lectures or explain your choice of a different method. If you choose a different method, provide at at least two arguments

to justify your choice compared to the ones covered in the lectures.
- Specify which model you selected and why.
- List the key parameters of your chosen model (Model 2).
- Provide a code implementation for the selected method.

Q2a answer: Decision Tree

Why?

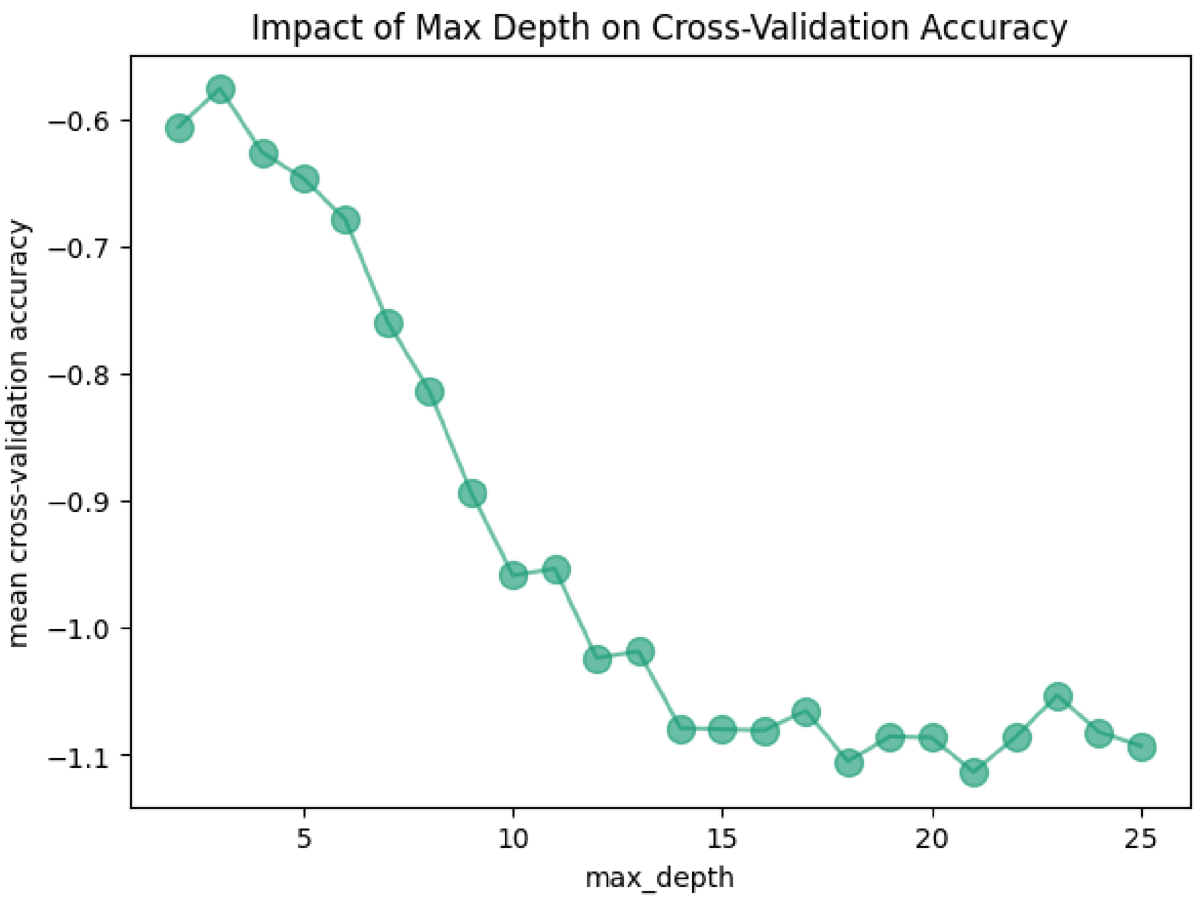
I chose a decision tree as it models non-linear patterns unlike the linear regression model which can only show linear patterns. Therefore the tree may allow me to detect and see patterns that may not be apparent in the LR model and enhance my understanding of how the different factors influence wine quality and how. Decision Trees also handle outliers better than linear regression models as they split the data up. This is especially important as there still are some outliers left in my data even after pre-processing it. A decision tree is also easy to trace and visualise which aids in understanding how we can predict the quality via which features. It also shows us which features have the most influence (features included in earlier decisions rather than later) which furthers our analysis. Also via the colour of the tree nodes we can see the class purity and so we can see how confident each prediction is.

Key Parameters

- max_depth - 3 - limit of how deep the tree can grow, set to 3 after using cross validation to see which value is best for accuracy.
- random_state - 1 - seed for random numb generator to ensure same random state used every time
- test_size - 0.3 - proportion of data set used for testing model 30:70 split

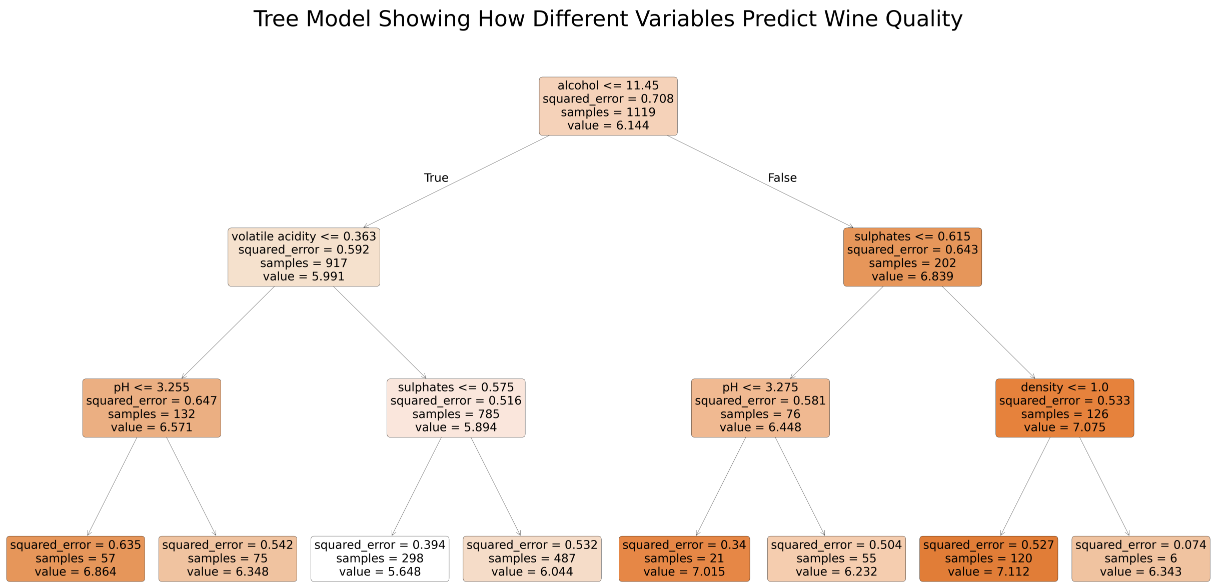
In [15]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn import tree
from sklearn import metrics
from sklearn.model_selection import cross_val_score

cv_scores = list()
depth_values = range(2,26)#testing depth up till 26
#ASSIGN FEATURES TO X(independent) y(dependent)
x = dfwine[['fixed acidity', 'volatile acidity', 'citric acid','residual sug
y = dfwine['quality']#output

for d in depth_values:#test every possible depth
    t_model_k = DecisionTreeRegressor(max_depth = d,random_state=1)
    score = cross_val_score(t_model_k, x, y, cv=3, scoring='neg_mean_squarec
    mean_score = score.mean()#calculate the average accuracy
    cv_scores.append(mean_score)
```

```
In [16]: #plot as line graph so can visualise and then see which has best accuracy
plt.figure(figsize=(7,5))
plt.plot(depth_values, cv_scores, '-o',markersize=10,alpha=0.65,color='#1b9e
plt.title('Impact of Max Depth on Cross-Validation Accuracy')
plt.xlabel('max_depth')
plt.ylabel('mean cross-validation accuracy')
plt.show()
```



Max depth of 3 seems to give best accuracy.

```
In [17]: X_train_t, X_test_t, y_train_t, y_test_t = train_test_split(x, y, test_size=
t_model = DecisionTreeRegressor(max_depth=3, random_state=1)#changed max dep

#training tree model
t_model.fit(X_train_t, y_train_t)

y_pred_test_t = t_model.predict(X_test_t)
y_pred_training_t = t_model.predict(X_train_t)

plt.figure(figsize=(100, 50))
plot_tree(t_model, feature_names=x.columns, filled=True, rounded=True)
plt.title('Tree Model Showing How Different Variables Predict Wine Quality',
plt.show()
```



Question 2b (10 marks)

TASK: Evaluate the performance of your new model and compare it to Prediction Model 1.

- Analyze whether the new model performs better or worse and explain why.
  - Base your evaluation on the same metrics used in Question 1d).
- Include one plot visually comparing the performance of both models.
- Provide a brief textual explanation interpreting the results.

Q2b answer:

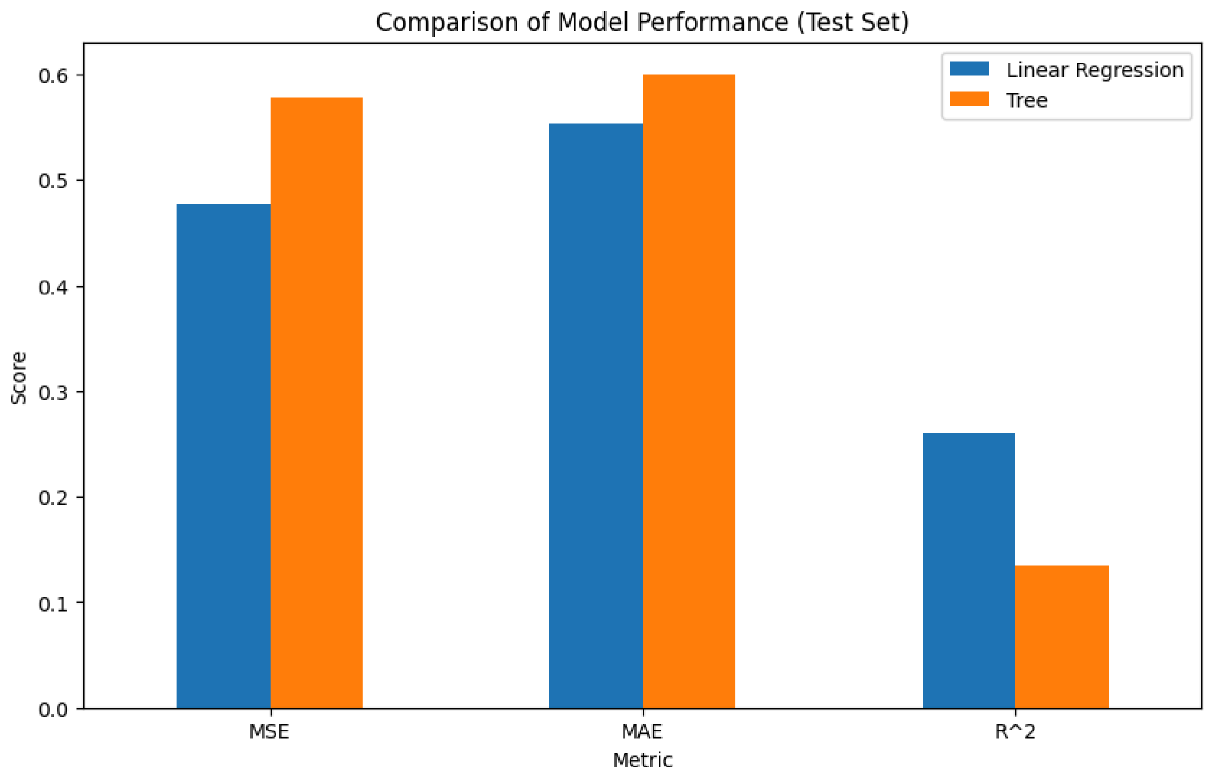
In [18]:
```
#calculate values for both
#LINEAR REGRESSION
print('LINEAR REGRESSION MODEL \n')
print("Test\n")
mse_LR_Test = metrics.mean_squared_error(y_test, y_pred_test_LR)#how close r
print('Mean Squared Error (MSE):', mse_LR_Test)

mae_LR_Test = metrics.mean_absolute_error(y_test, y_pred_test_LR)# average a
print('Mean Absolute Error (MAE):', mae_LR_Test)

rsq_LR_Test = metrics.r2_score(y_test, y_pred_test_LR)#how well data fits re
print('Root Squared:', rsq_LR_Test)

print("\nTrain\n")
mse_LR_Train = metrics.mean_squared_error(y_train, y_pred_train_LR)
print('Mean Squared Error (MSE):', mse_LR_Train)

mae_LR_Train = metrics.mean_absolute_error(y_train, y_pred_train_LR)
print('Mean Absolute Error (MAE):', mae_LR_Train)
```

```python
rsq_LR_Train = metrics.r2_score(y_train, y_pred_train_LR)
print('Root Squared:', rsq_LR_Train)

#TREE
print('\nTREE\n')
print('Test\n')
mse_Test = metrics.mean_squared_error(y_test_t, y_pred_test_t)
print('Mean Squared Error (MSE):', mse_Test)

mae_Test = metrics.mean_absolute_error(y_test_t, y_pred_test_t)
print('Mean Absolute Error (MAE):', mae_Test)

rsq_Test = metrics.r2_score(y_test_t, y_pred_test_t)
print('Root Squared:', rsq_Test)

print('\nTrain\n')
mse_Train = metrics.mean_squared_error(y_train_t, y_pred_training_t)
print('Mean Squared Error (MSE):', mse_Train)

mae_Train = metrics.mean_absolute_error(y_train_t, y_pred_training_t)
print('Mean Absolute Error (MAE):', mae_Train)


rsq_Train = metrics.r2_score(y_train_t, y_pred_training_t)
print('Root Squared:', rsq_Train)
```

```
LINEAR REGRESSION MODEL

Test

Mean Squared Error (MSE): 0.4766730911152701
Mean Absolute Error (MAE): 0.5540994595944266
Root Squared: 0.2608876273573414

Train

Mean Squared Error (MSE): 0.5017439481737791
Mean Absolute Error (MAE): 0.5548488596008473
Root Squared: 0.3026837973731459

TREE

Test

Mean Squared Error (MSE): 0.5780993842643625
Mean Absolute Error (MAE): 0.6004901631505993
Root Squared: 0.13492375217497343

Train

Mean Squared Error (MSE): 0.49319536882884885
Mean Absolute Error (MAE): 0.559158631972662
Root Squared: 0.3030003227661727
```

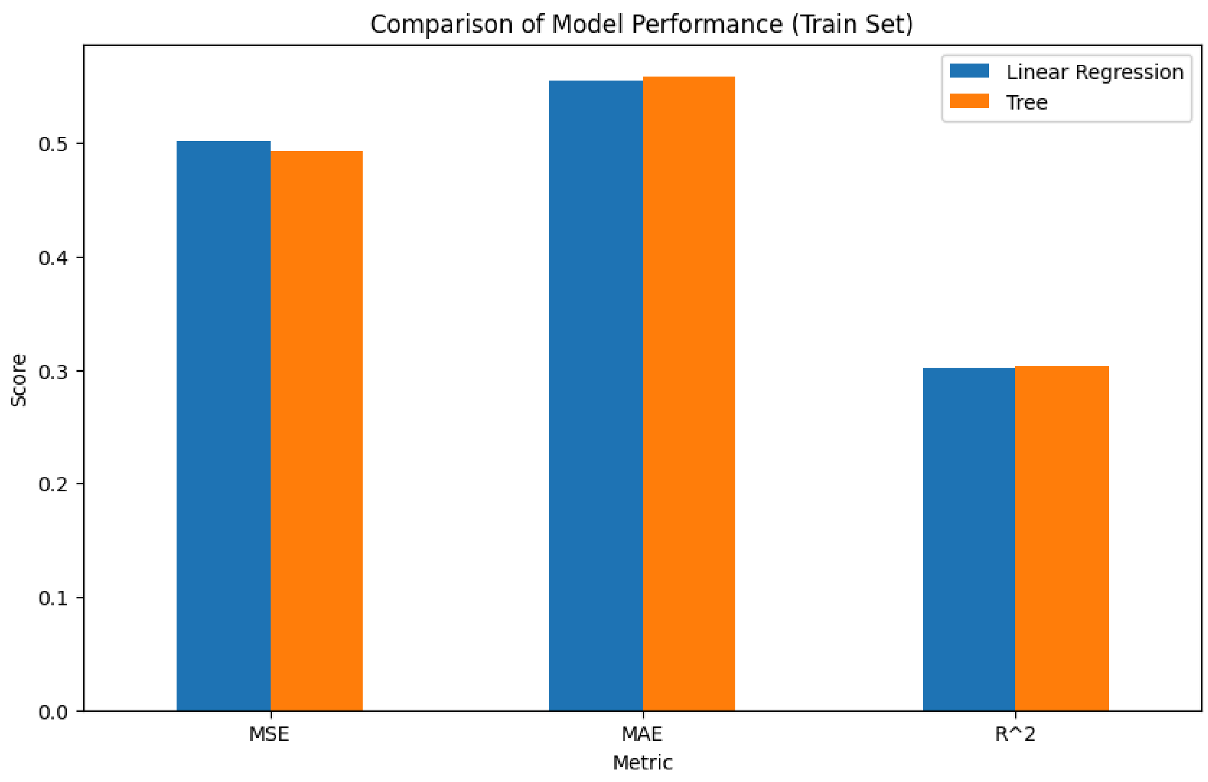
In [19]:
```python
metrics={#add metrics into one array so can be plotted
    'Metric':['MSE','MAE','R^2'],
    'Linear Regression':[mse_LR_Test, mae_LR_Test, rsq_LR_Test],
    'Tree':[mse_Test, mae_Test, rsq_Test]}
dfmetrics=pd.DataFrame(metrics)#make into dataframe

#plot bar chart
plt.figure(figsize=(10, 6))
dfmetrics.set_index("Metric").plot(kind="bar", figsize=(10, 6), rot=0)
plt.xlabel("Metric")
plt.ylabel("Score")
plt.title("Comparison of Model Performance (Test Set)")
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



In [20]:
```python
metrics_train={
    'Metric':['MSE','MAE','R^2'],
    'Linear Regression':[mse_LR_Train, mae_LR_Train, rsq_LR_Train],
    'Tree':[mse_Train, mae_Train, rsq_Train]}
dfmetrics_train=pd.DataFrame(metrics_train)

plt.figure(figsize=(10, 6))
dfmetrics_train.set_index("Metric").plot(kind="bar", figsize=(10, 6), rot=0)
plt.xlabel("Metric")
plt.ylabel("Score")
plt.title("Comparison of Model Performance (Train Set)")
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Analysis

Testing set

The LR model outperfomed the Tree on all test metrics. The LR model has a lower MSE of 0.4766 compared to the Tree model of 0.5781 indicating it makes fewer large errors. LR has a slightly lower MAE of 0.554 than the Tree at 0.6 meaning both models perform similarily at accurately predicting values but LR is slightly better so more consistent at reducing errors. LR has a higher R^2 of 0.261 than the Tree with 0.135. Since it is closer to 1 the LR has a better fit, which suggests the LR model explains more variance in the data. Therefore, for the testing set, the LR model seems to be better. The LR model generalizes better whereas the Tree has more variance suggesting it could be overfitting.

Training set

For the training set both models seemed to perform extremely similarily:

- Better MSE (lower the better): Tree with 0.493 compared to LR with 0.502 = 0.009 difference
- Better MAE (lower the better): LR with 0.555 compared to Tree with 0.559 = 0.004 difference
- Better R^2 (closer to 1 the better): Tree with 0.303 compared to LR with 0.302 = 0.001 difference
    - all the metrics are quite similar meaning whilst the Tree did technically perform better in MSE and R^2 and LR in MAE, the LR model still performed nearly the same with the biggest difference being the MSE of

0.9. Therefore both models can be claimed as equally efficient for the training set.

Since the Tree model performs slightly better than LR in training but significantly worse in the testing set it suggests the Tree model may be overfitting. The lower the MSE, the better the model performs as it indicates smaller errors. Since the Tree model has a higher MSE on test set than the train data it supports that the Tree model could be overfitting. The tree model also has a higher variance between the training set and test set with its R^2 dropping from 0.303 to 0.135. This indicates it is weaker at generalising which is potentially due to being over complex as there are many features involved which isn't optimal for Tree models as they tend to work better with less features.

Whereas the LR model is more consistent between the training and test data performance with similar values being generated for each set.This indicates the LR model generalises better to new data making it more reliable.

Conclusion

In conclusion, the Linear Regression Model seems to be the better model as it performs better for the test set, but the Tree model does still perform at a similar level and so is still a strong model.

Bar chart

The bar charts show that the models differ the most in the R^2 metric, while their MAE values remain relatively similar. For the training data, the models perform similarly with the Tree model appearing slightly better for MSE and slightly worse for MAE. This suggests both model's performance is similarily strong. However, for the test data, the Tree model's values deviate much more from the LR model values, which indicates the Tree model is weaker for generalization. The difference in their performance for the test data is greater than the difference in their performamce for the training data.

Question 3. Comparison and Improvement [30 marks]

Question 3a (15 marks)

TASK: Analyze the impact of removing the least important feature from Prediction Model 1.

- Identify and remove the least important feature.
- Retrain the Linear Regression model and evaluate its performance.

- Compare the results before and after feature removal.
- Provide a code implementation and a justification explaining the impact on model performance.

Q3a answer:

Identifying least important feature

In [21]:
```python
from sklearn.linear_model import LinearRegression
lr_model.fit(X_train, y_train)
```

Out[21]:
```
▾ LinearRegression

LinearRegression()
```

In [22]:
```python
#print coefficients again
print(dfwine.columns)
print('Coefficients:', lr_model.coef_)
print('Intercept:', lr_model.intercept_)
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
Coefficients: [ 3.46443557e-02 -8.82097776e-01 -3.85145596e-01  2.73931640e-
02
 -1.66396391e+00  3.97661654e-03 -3.13487817e-03 -2.25374865e+01
 -4.01130097e-01  1.38478877e+00  2.56331168e-01]
Intercept: 26.789214672085443
```

The smallest absolute coefficient is -3.13487817e-03. This corresponds to the feature 'total sulfur dioxide' therefore meaning it is the least important feature.

Removing least important feature and retraining

In [23]:
```python
#removing feature from dfwine
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
#ASSIGN FEATURES TO X(independent) y(dependent)
#assign IV to only consist of the features excluding total sulfur dioxide
X2 = dfwine[['fixed acidity', 'volatile acidity', 'citric acid','residual su
y2 = dfwine['quality']

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.
lr_model2 = LinearRegression()
lr_model2.fit(X_train2, y_train2)
```

Out[23]:

```
▾ LinearRegression
LinearRegression()
```

In [24]:

```python
print("!!!!                                         LINEAR REGRESSION WITHOUT
print(X2.columns)
print('Coefficients:', lr_model2.coef_)
print('Intercept:', lr_model2.intercept_)
print('\nThe linear regression function learned between X and Y is:')
#equation
print(f'y = {lr_model2.intercept_:.2f}', end='')
for i, coef in enumerate(lr_model2.coef_):
    print(f' + {coef:.2f}*{X2.columns[i]}', end='')
print('\n')
y_pred_test_LR2 = lr_model2.predict(X_test2)
y_pred_train_LR2 = lr_model2.predict(X_train2)
#LINEAR REGRESSION
print('LINEAR REGRESSION MODEL \n')
print("Test\n")
mse_LR_Test2 = metrics.mean_squared_error(y_test2, y_pred_test_LR2)#how clos
print('Mean Squared Error (MSE):', mse_LR_Test2)

mae_LR_Test2 = metrics.mean_absolute_error(y_test2, y_pred_test_LR2)# averag
print('Mean Absolute Error (MAE):', mae_LR_Test2)

rsq_LR_Test2 = metrics.r2_score(y_test2, y_pred_test_LR2)#how well data fits
print('Root Squared:', rsq_LR_Test2)

print("\nTrain\n")
mse_LR_Train2 = metrics.mean_squared_error(y_train2, y_pred_train_LR2)
print('Mean Squared Error (MSE):', mse_LR_Train2)

mae_LR_Train2 = metrics.mean_absolute_error(y_train2, y_pred_train_LR2)
print('Mean Absolute Error (MAE):', mae_LR_Train2)


rsq_LR_Train2 = metrics.r2_score(y_train2, y_pred_train_LR2)
print('Root Squared:', rsq_LR_Train2)
```

```
!!!!                                        LINEAR REGRESSION WITHOUT TOTAL S
ULFUR DIOXIDE                                !!!!

Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'density', 'pH', 'sulphates',
       'alcohol'],
      dtype='object')
Coefficients: [ 5.69419606e-02 -9.44073853e-01 -5.08249037e-01  1.82247382e-
02
 -1.79556230e+00 -2.19828013e-03 -2.99630044e+01 -2.91858392e-01
  1.41920536e+00  2.70752769e-01]
Intercept: 33.519445904816095

The linear regression function learned between X and Y is:
y = 33.52 + 0.06*fixed acidity + -0.94*volatile acidity + -0.51*citric acid
+ 0.02*residual sugar + -1.80*chlorides + -0.00*free sulfur dioxide + -29.96
*density + -0.29*pH + 1.42*sulphates + 0.27*alcohol

LINEAR REGRESSION MODEL

Test

Mean Squared Error (MSE): 0.4824471914084439
Mean Absolute Error (MAE): 0.5618481523689014
Root Squared: 0.2519345124299198

Train

Mean Squared Error (MSE): 0.5069770848960334
Mean Absolute Error (MAE): 0.558768923340991
Root Squared: 0.29541086256193105
```

In [25]:

```
#WITH total sulfur dioxide included
print("!!!!                                        LINEAR REGRESSION WITH TO

#ASSIGN FEATURES TO X(independent) y(dependent)
X = dfwine[['fixed acidity', 'volatile acidity', 'citric acid','residual sug
y = dfwine['quality']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
print(dfwine.columns)
print('Coefficients:', lr_model.coef_)
print('Intercept:', lr_model.intercept_)
print('\nThe linear regression function learned between X and Y is:')
#equation
print(f'y = {lr_model.intercept_:.2f}', end='')
for i, coef in enumerate(lr_model.coef_):
    print(f' + {coef:.2f}*{X.columns[i]}', end='')
print('\n')
y_pred_test_LR = lr_model.predict(X_test)
y_pred_train_LR = lr_model.predict(X_train)
#LINEAR REGRESSION
print('LINEAR REGRESSION MODEL \n')
print("Test\n")
```

```
mse_LR_Test = metrics.mean_squared_error(y_test, y_pred_test_LR)#how close r
print('Mean Squared Error (MSE):', mse_LR_Test)

mae_LR_Test = metrics.mean_absolute_error(y_test, y_pred_test_LR)# average a
print('Mean Absolute Error (MAE):', mae_LR_Test)

rsq_LR_Test = metrics.r2_score(y_test, y_pred_test_LR)#how well data fits re
print('Root Squared:', rsq_LR_Test)

print("\nTrain\n")
mse_LR_Train = metrics.mean_squared_error(y_train, y_pred_train_LR)
print('Mean Squared Error (MSE):', mse_LR_Train)

mae_LR_Train = metrics.mean_absolute_error(y_train, y_pred_train_LR)
print('Mean Absolute Error (MAE):', mae_LR_Train)


rsq_LR_Train = metrics.r2_score(y_train, y_pred_train_LR)
print('Root Squared:', rsq_LR_Train)
```

```
!!!!                                       LINEAR REGRESSION WITH TOTAL SUL
FUR DIOXIDE                                  !!!!

Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'densit
y',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
Coefficients: [ 3.46443557e-02 -8.82097776e-01 -3.85145596e-01  2.73931640e-
02
 -1.66396391e+00  3.97661654e-03 -3.13487817e-03 -2.25374865e+01
 -4.01130097e-01  1.38478877e+00  2.56331168e-01]
Intercept: 26.789214672085443

The linear regression function learned between X and Y is:
y = 26.79 + 0.03*fixed acidity + -0.88*volatile acidity + -0.39*citric acid
+ 0.03*residual sugar + -1.66*chlorides + 0.00*free sulfur dioxide + -0.00*t
otal sulfur dioxide + -22.54*density + -0.40*pH + 1.38*sulphates + 0.26*alco
hol

LINEAR REGRESSION MODEL

Test

Mean Squared Error (MSE): 0.4766730911152701
Mean Absolute Error (MAE): 0.5540994595944266
Root Squared: 0.2608876273573414

Train

Mean Squared Error (MSE): 0.5017439481737791
Mean Absolute Error (MAE): 0.5548488596008473
Root Squared: 0.3026837973731459
```

```
In [26]: #create data set of old coefficients and separate array for new coefficients
oldLR=np.array([
    3.51305672e-02, -8.87785818e-01, -3.91228843e-01, 2.64907075e-02,
    -1.71169576e+00, 2.60918640e-03, -2.87306709e-03, -2.25336868e+01,
    -3.93353193e-01, 1.39097837e+00, 2.56420005e-01
])

newLR=np.array([
    5.71811749e-02, -9.44756578e-01, -5.09376095e-01, 1.81890667e-02,
    -1.80282713e+00, -2.40704708e-03, -3.01040397e+01, -2.86843732e-01,
    1.42045164e+00, 2.71012601e-01
])

features=['fixed acidity', 'volatile acidity', 'citric acid', 'residual suga
metrics={
    "Test MSE": [0.4766295816959547, 0.4832764195380482],
    "Test MAE": [0.554322693853113, 0.562607579337854],
    "Test R^2": [0.26095509151829765, 0.25064874073070365],
    "Train MSE": [0.5021661912222658, 0.5068872294104745],
    "Train MAE": [0.5551165880246985, 0.5586936509604156],
    "Train R²": [0.3020969703267451, 0.2955357423660685]
}

metrics_df=pd.DataFrame(metrics, index=["Old LR Model", "New LR Model"])
coefdiff=pd.DataFrame({
    "Feature":features,
    "Old Coefficients":oldLR[:-1],#not include total sulfur dioxide
    "New Coefficients":newLR,
    "change":newLR-oldLR[:-1]
})
```

In [27]:
```
metrics_df
```

Out[27]:

	Test MSE	Test MAE	Test R^2	Train MSE	Train MAE	Train R²
Old LR Model	0.476630	0.554323	0.260955	0.502166	0.555117	0.302097
New LR Model	0.483276	0.562608	0.250649	0.506887	0.558694	0.295536

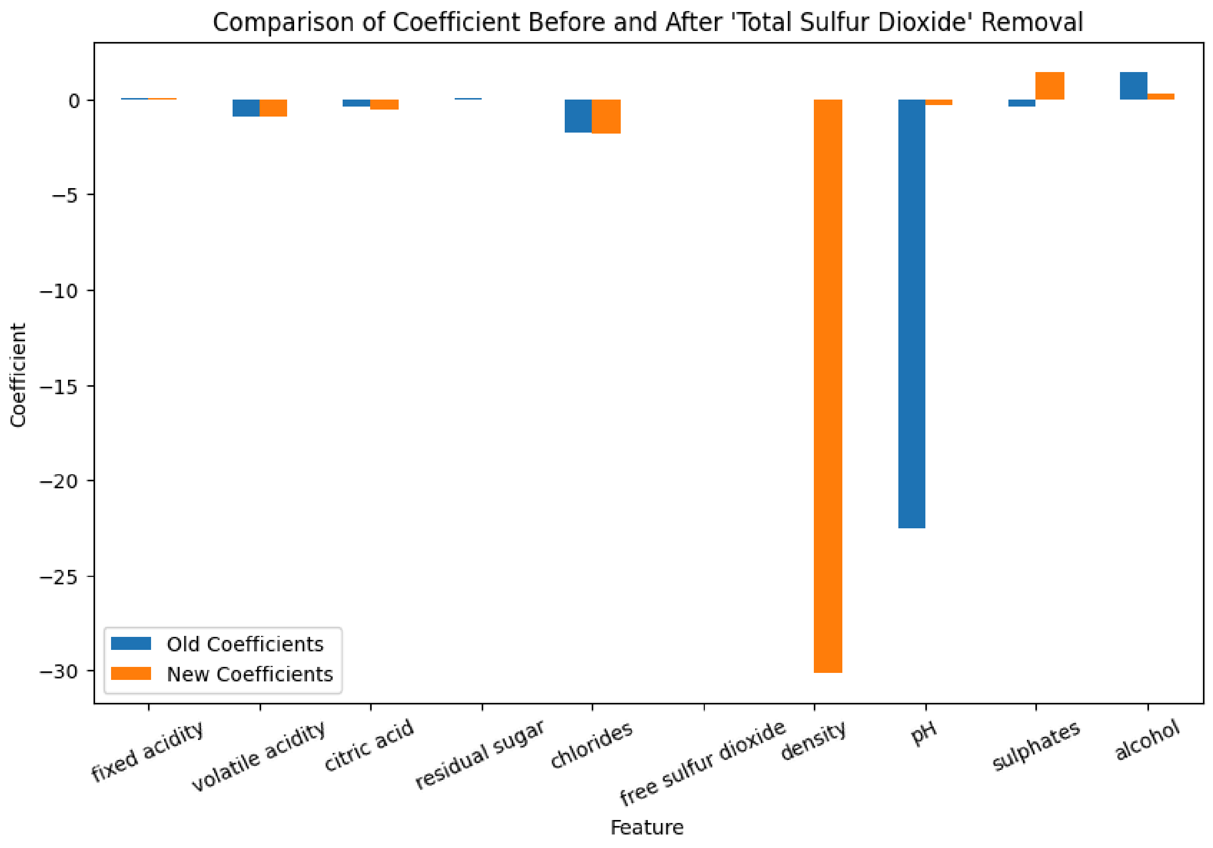
In [28]:
```
coefdiff
```

Out[28]:

	Feature	Old Coefficients	New Coefficients	change
0	fixed acidity	0.035131	0.057181	0.022051
1	volatile acidity	-0.887786	-0.944757	-0.056971
2	citric acid	-0.391229	-0.509376	-0.118147
3	residual sugar	0.026491	0.018189	-0.008302
4	chlorides	-1.711696	-1.802827	-0.091131
5	free sulfur dioxide	0.002609	-0.002407	-0.005016
6	density	-0.002873	-30.104040	-30.101167
7	pH	-22.533687	-0.286844	22.246843
8	sulphates	-0.393353	1.420452	1.813805
9	alcohol	1.390978	0.271013	-1.119966

In [29]:
```
plt.figure(figsize=(10, 6))
coefdiff.set_index("Feature").drop(columns=['change']).plot(kind="bar", figs
plt.xlabel("Feature")
plt.ylabel("Coefficient")
plt.title("Comparison of Coefficient Before and After 'Total Sulfur Dioxide'
plt.xticks(rotation=25)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Impact

Coefficients:

By removing the least important feature we can see from the bar chart that most features were barely affected however:

- Density decreased significantly. Of all features, it decreased the most by -30.101167 from -0.002873 to -30.10 meaning it has more influence on the wine quality prediction after removing total sulfur dioxide
- Chlorides also decreased slightly from -1.71 to -1.8 meaning it also has more influence, but only slightly
- Citric Acid decreased slightly from -0.39 to -0.51 also meaning more influence
- pH increased the most, by 22 meaning it has less influence on the wine quality prediction

Metrics:

For testing: The MSE and MAE increased slightly meaning the model became slightly worse after. However this is extremely minimal with the difference for the MSE being only 0.0067. So overall the model's performance measured by MSE and MAE stayed nearly the same. The same can be said for the R^2 value which decreased slightly from 0.2609 to 0.2506.

For training: Again the MSE and MAE increased slightly and the R^2 decreased slightly.

Overall removing total sulfur dioxide doesn't seem to significantly improve or weaken the LR model's performance but it does reduce it very slightly. This suggests that whilst total sulfur dioxide wasn't the most impactful feature, it is still somewhat useful to include.

Question 3b (15 marks)

TASK: Based on your observations, suggest strategies for improving future models when predicting on new data.

- Discuss potential improvements.

Hint: based on relevant analysis, feature selection, feature scaling and data processing (e.g. resolve imbalanced samples, errors and outliers, etc.) could all potentially improve the model by reducing training time, fixing overfitting and improving interpretability, etc. You can also explore external resources for other potential approaches or techniques.

Note: Coding is optional here, but your answers should be supported by relevant analysis or justifications.

Q3b answer:

Feature selection

We saw that removing even the least important feature, sulfur dioxide had a slight impact on our LR model, so we could look at the other features as well incase they have an impact on our model's accuracy.

- Each feature could be tested by removing it to see its impact on the model like we did for total sulfur dioxide.
- Then the features who seem to weaken the model can be removed to improve interpretability and reduce overfitting.
- Or features which don't seem to impact the model and are therefore just wasted training time.
- This could also reduce training time as there are less features to process as well as simplify the model making it more generalisable.

Feature interactions

Instead of each feature directly influencing wine quality, there may instead be a combined influence of multiple features interacting which affect wine quality.

- For example by plotting the features against quality we could identify seemingly insignificant relationships. (See scatter grams below)
- Like quality and pH where the correlation doesn't seem to be very strong with no clear trend.
- Whilst this could mean there may not be a strong linear relationship between pH and wine quality, it could also mean two or more of the features may influence wine quality together e.g. the pH's influence may be impacted by the acidity, an interaction between the two.
- So to improve the model we could combine interactions between existing features into a new feature and then include this new feature when training the model to predict based off of these interactions as well.
- This means the model can better include complex relationships which can enhance our understanding of how and which features determine the output allowing our model to make better predictions
- It also helps better our feature selection

Cross-validation can be used to check generalisation on unseen data

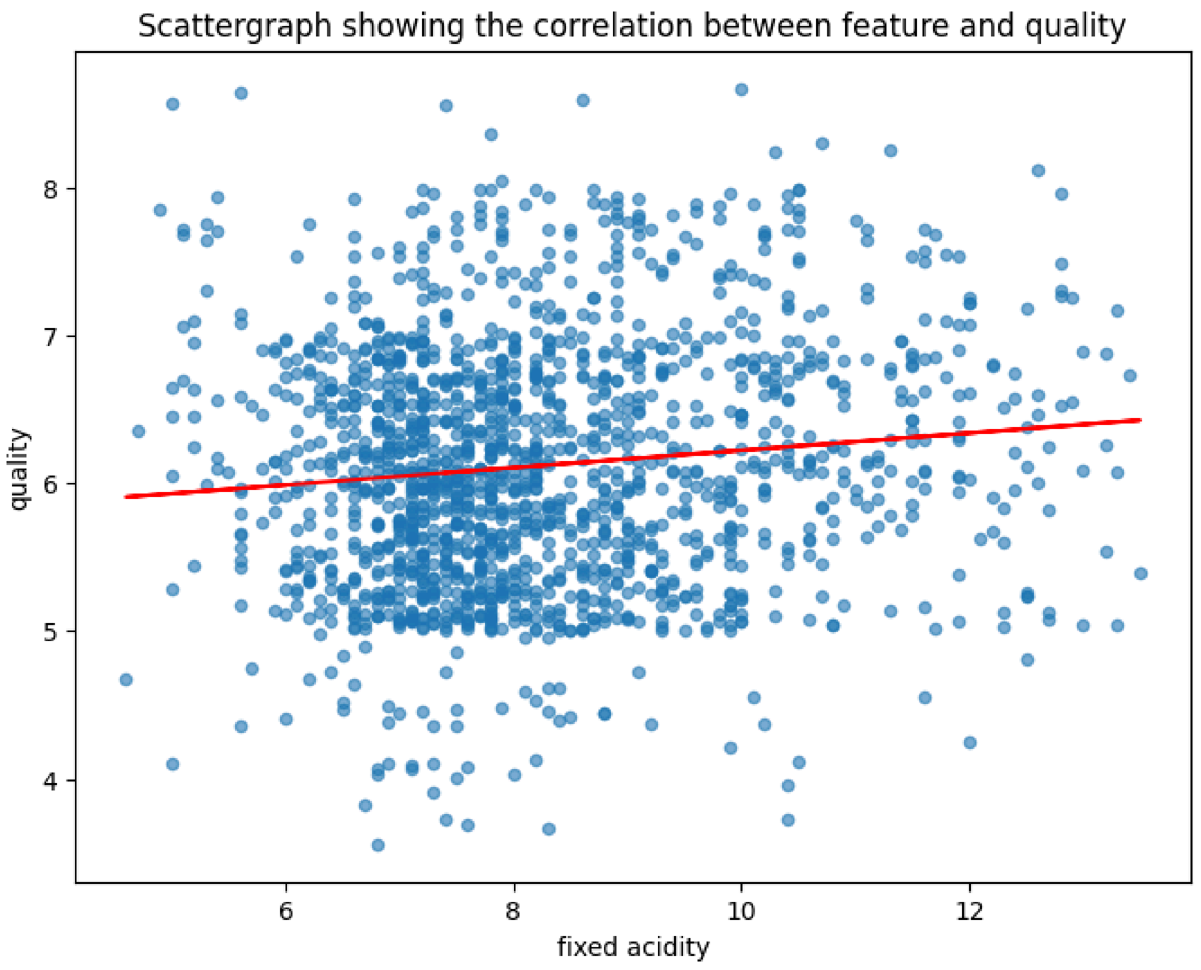
- This checks the data sets used for the training and test data weren't flukes and so increases the reliability and generalisabilty as we can be sure of the performance of our model beyond our test and training data.

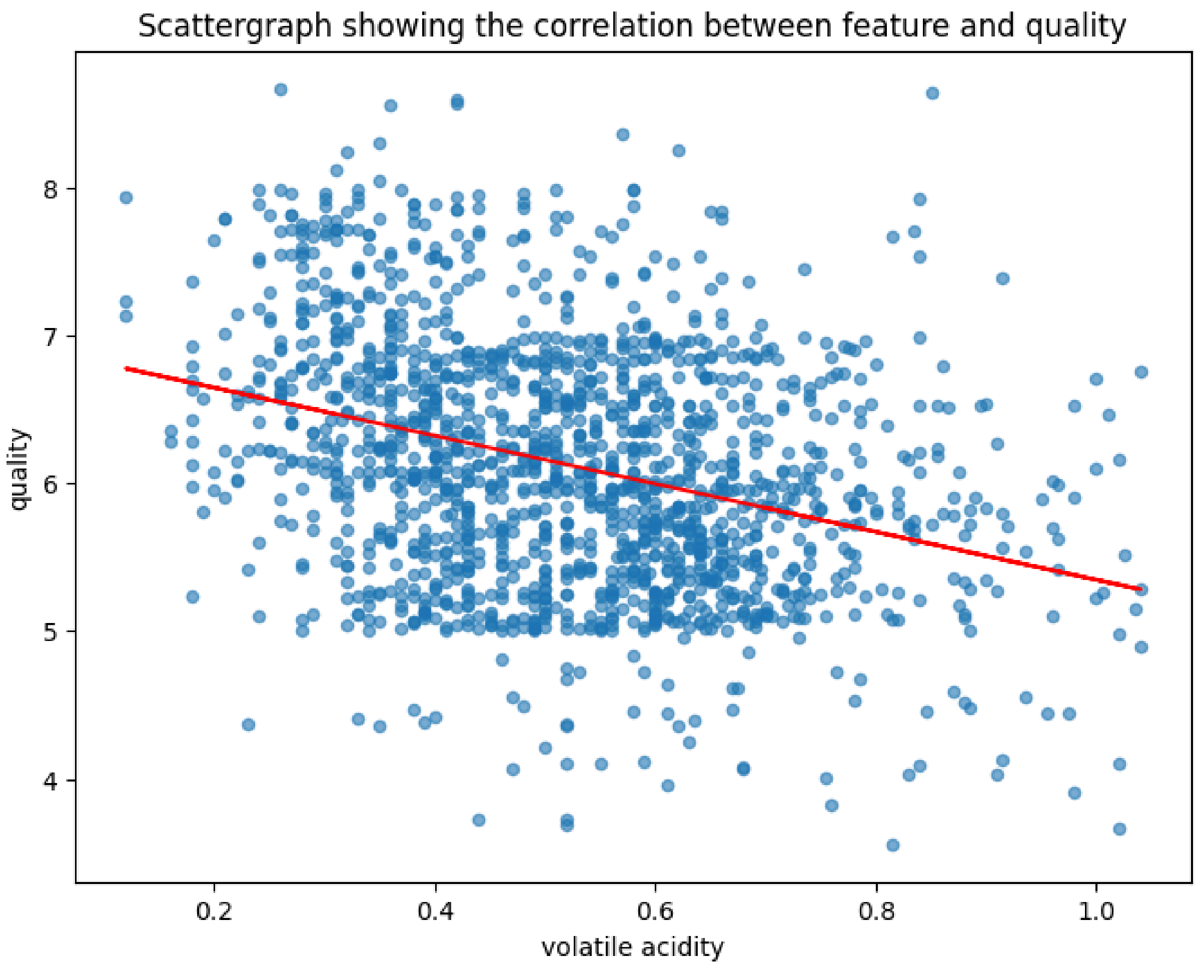
Non-linear relationships/ polynomial regression

Polynomial regression extends linear regression by adding higher degree terms which allows a model to fit more complex data terms. If the relationships between features and the output are not linear, introducing polynomial regression can reveal hidden patterns missed by linear regression. Polynomial regression captures curved relationships unlike linear which only captures straight-line relationships. This lets us see if some features may instead have an optimal range rather than a direct linear influence. This would increase the accuracy of predictions as less obvious relationships can be identified and included.

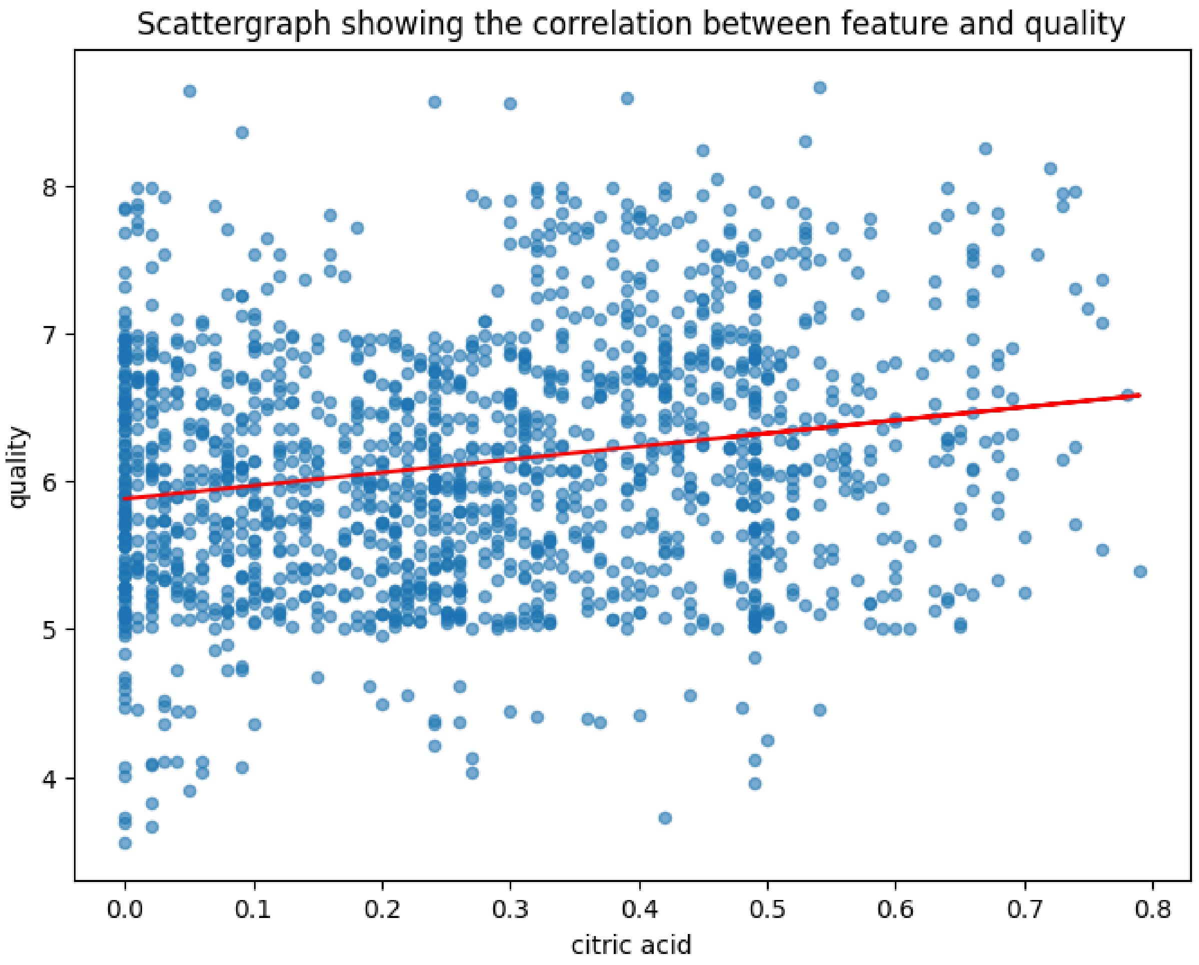
- For example by plotting the features against quality below we could identify potential non linear relationships.
- Like quality and pH where the correlation doesn't seem to be very strong with no clear trend. Which suggests there may be a non linear (or potentially no relationship) between quality and pH.
- pH may instead have an optimal range.

```
In [30]: for column in dfwine.columns[:-1]:#for every feature
             dfwine.plot.scatter(x=column, y='quality', title=('Scattergraph showing
             m,b = np.polyfit(dfwine[column], dfwine['quality'], 1)
             plt.plot(dfwine[column], m * dfwine[column] + b, color='red', label='Lir
```

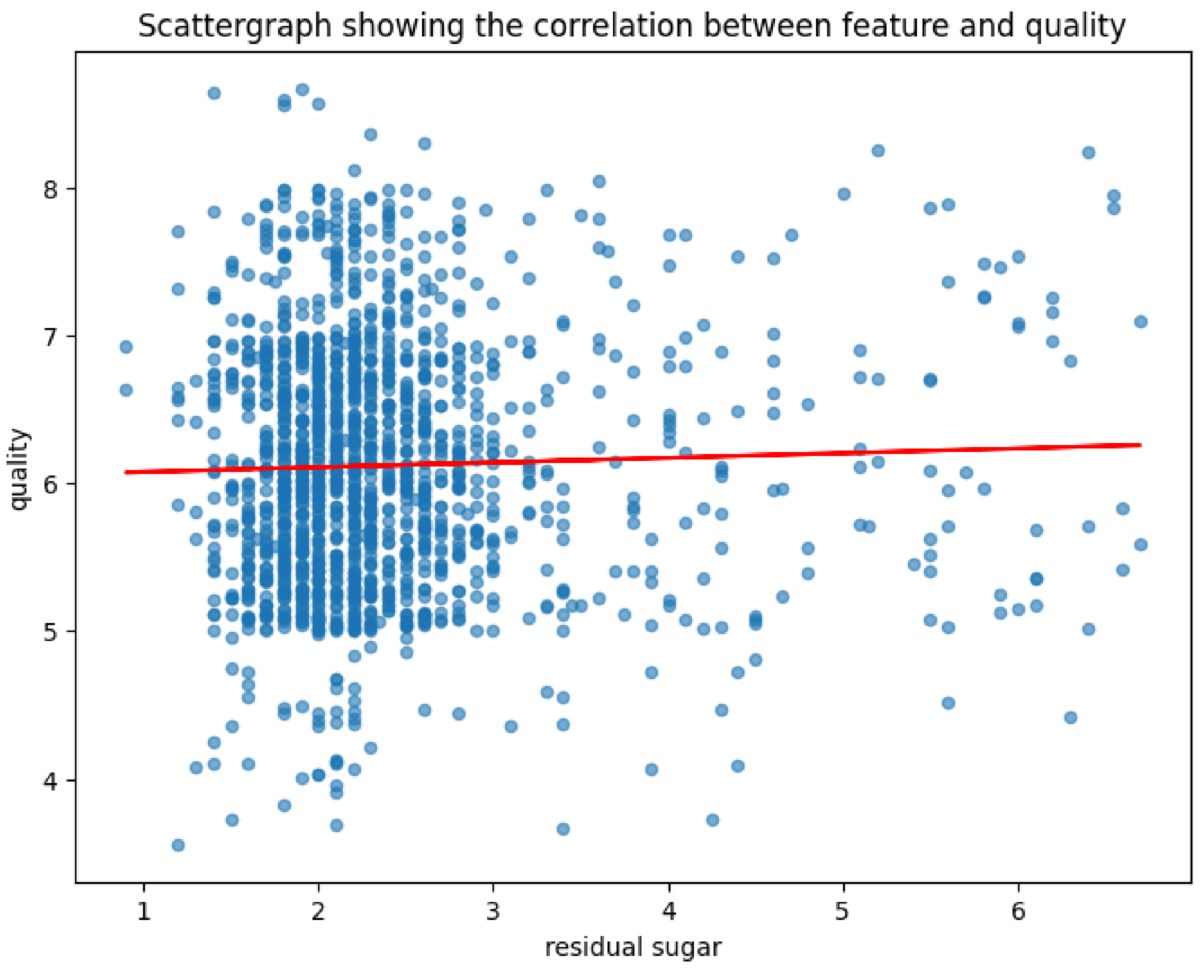

Scattergraph showing the correlation between feature and quality
quality
fixed acidity
4
5
6
7
8
6
8
10
12

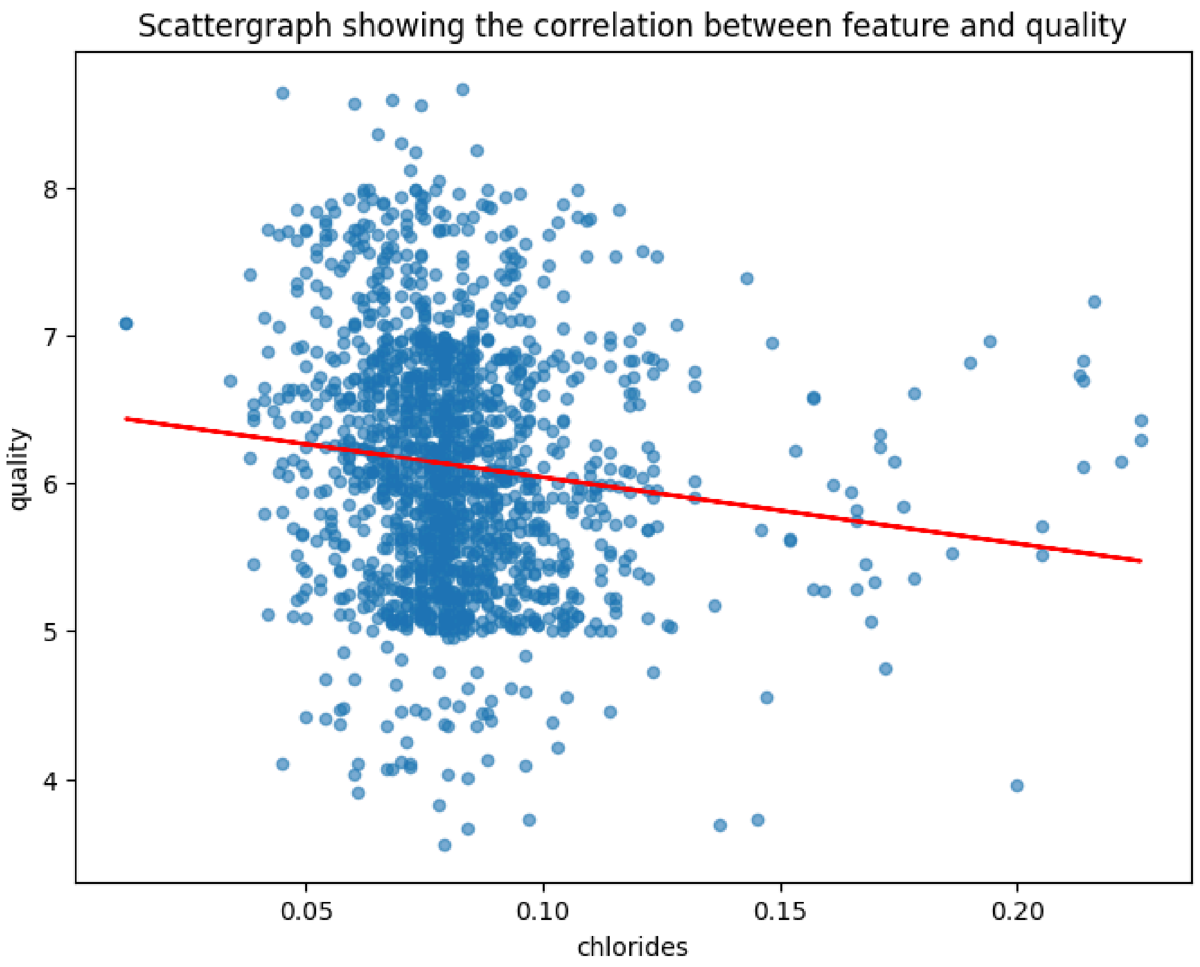


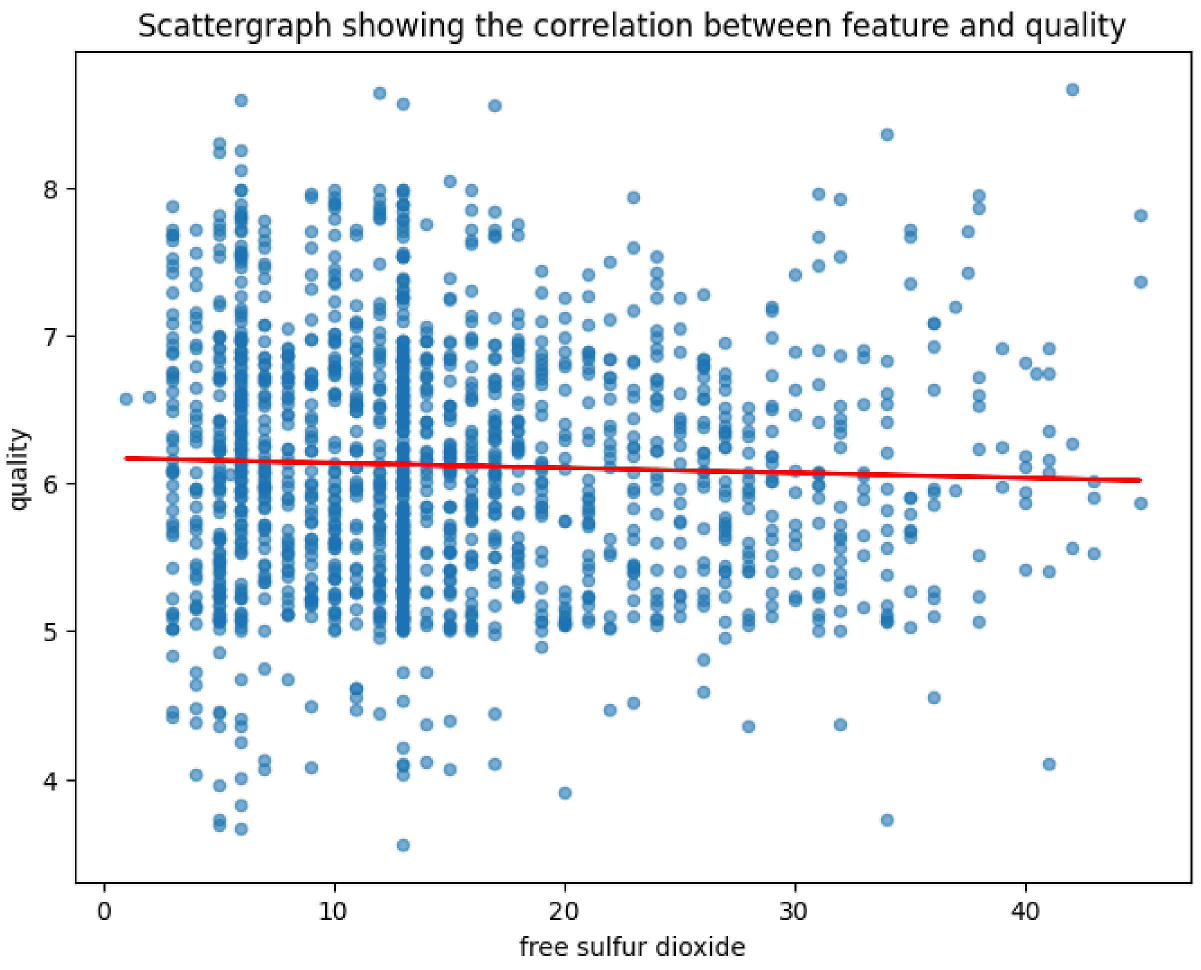
Scattergraph showing the correlation between feature and quality
quality
volatile acidity
8
7
6
5
4
0.2
0.4
0.6
0.8
1.0



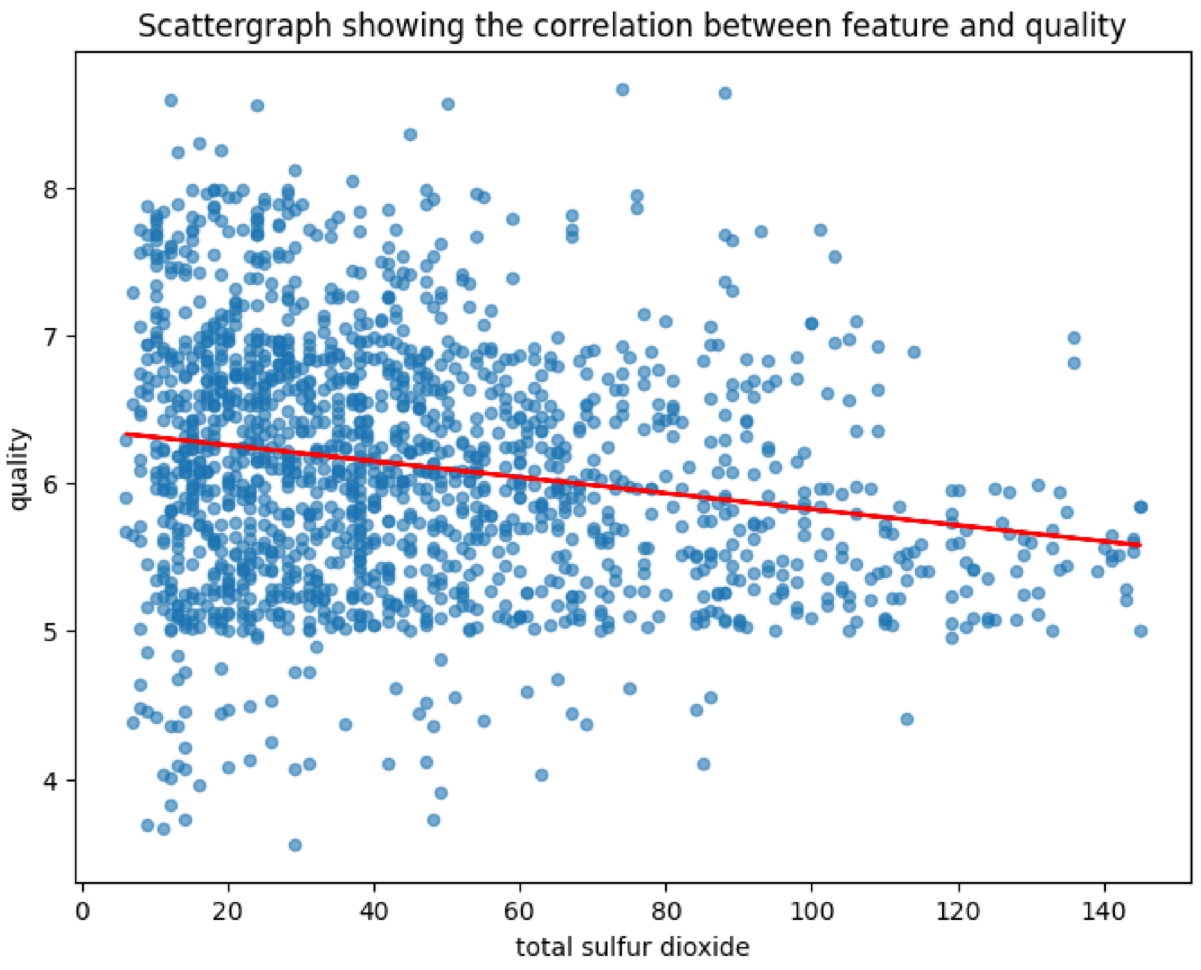
Scattergraph showing the correlation between feature and quality
quality
citric acid
8
7
6
5
4
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

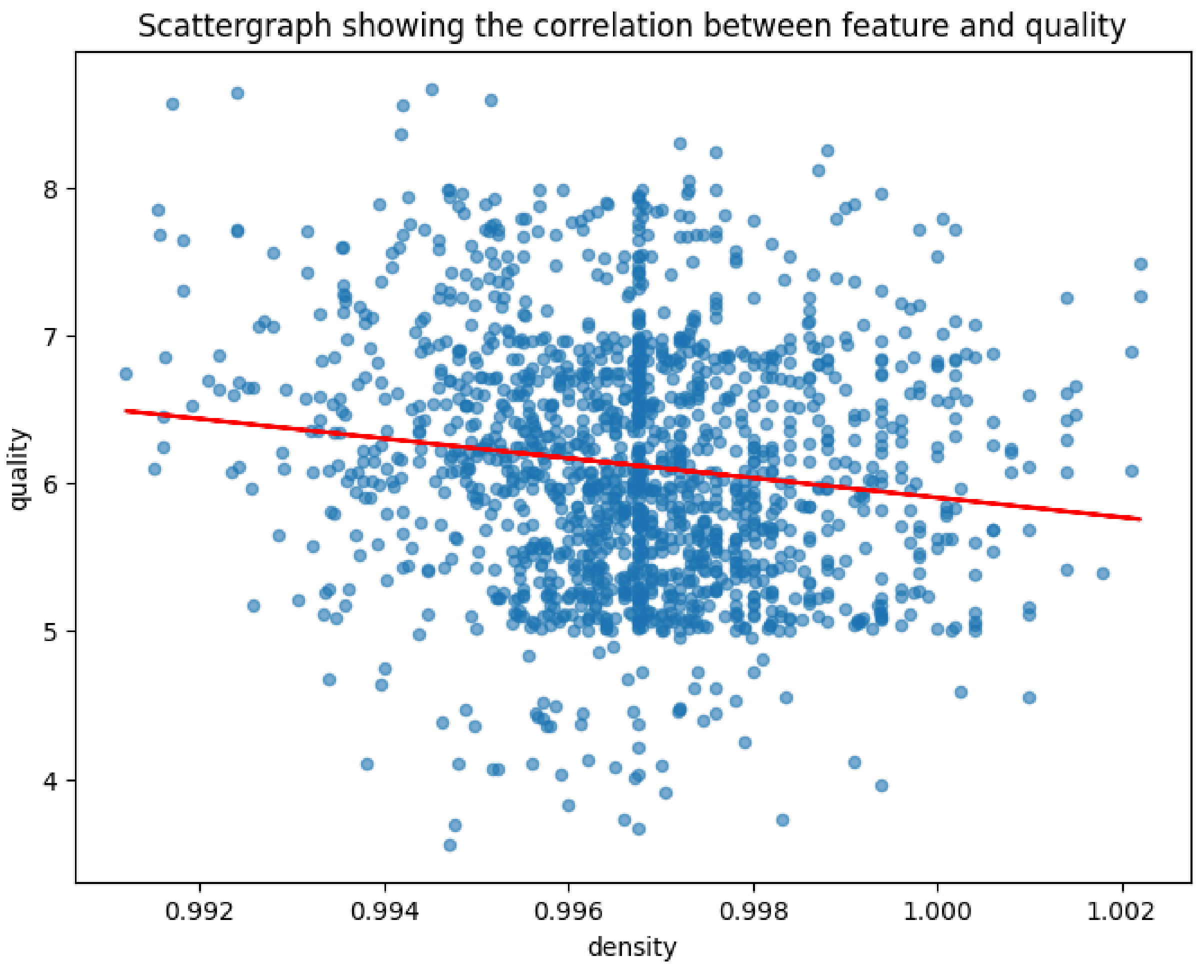

Scattergraph showing the correlation between feature and quality
quality
residual sugar
8
7
6
5
4
1
2
3
4
5
6

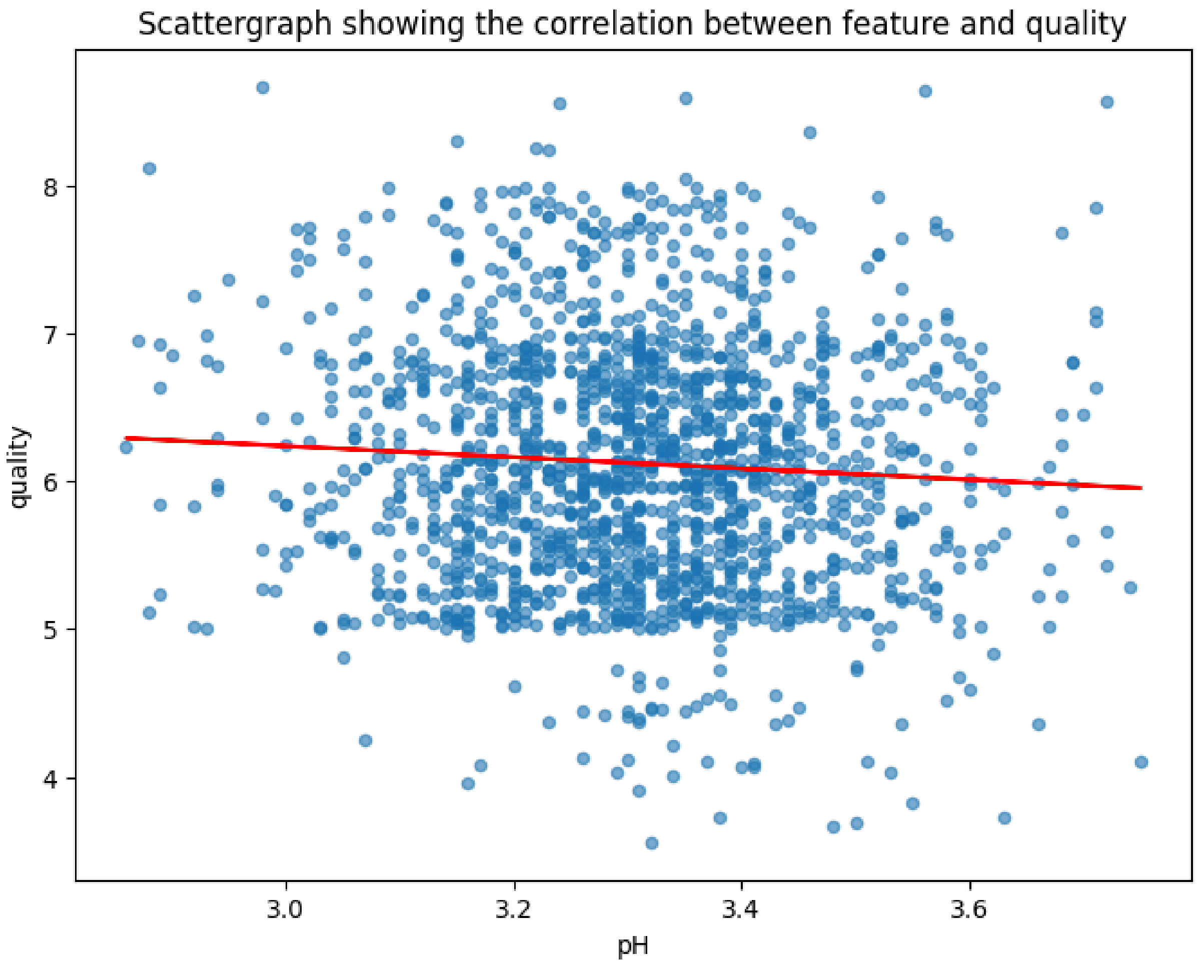

Scattergraph showing the correlation between feature and quality
quality
chlorides
8
7
6
5
4
0.05
0.10
0.15
0.20

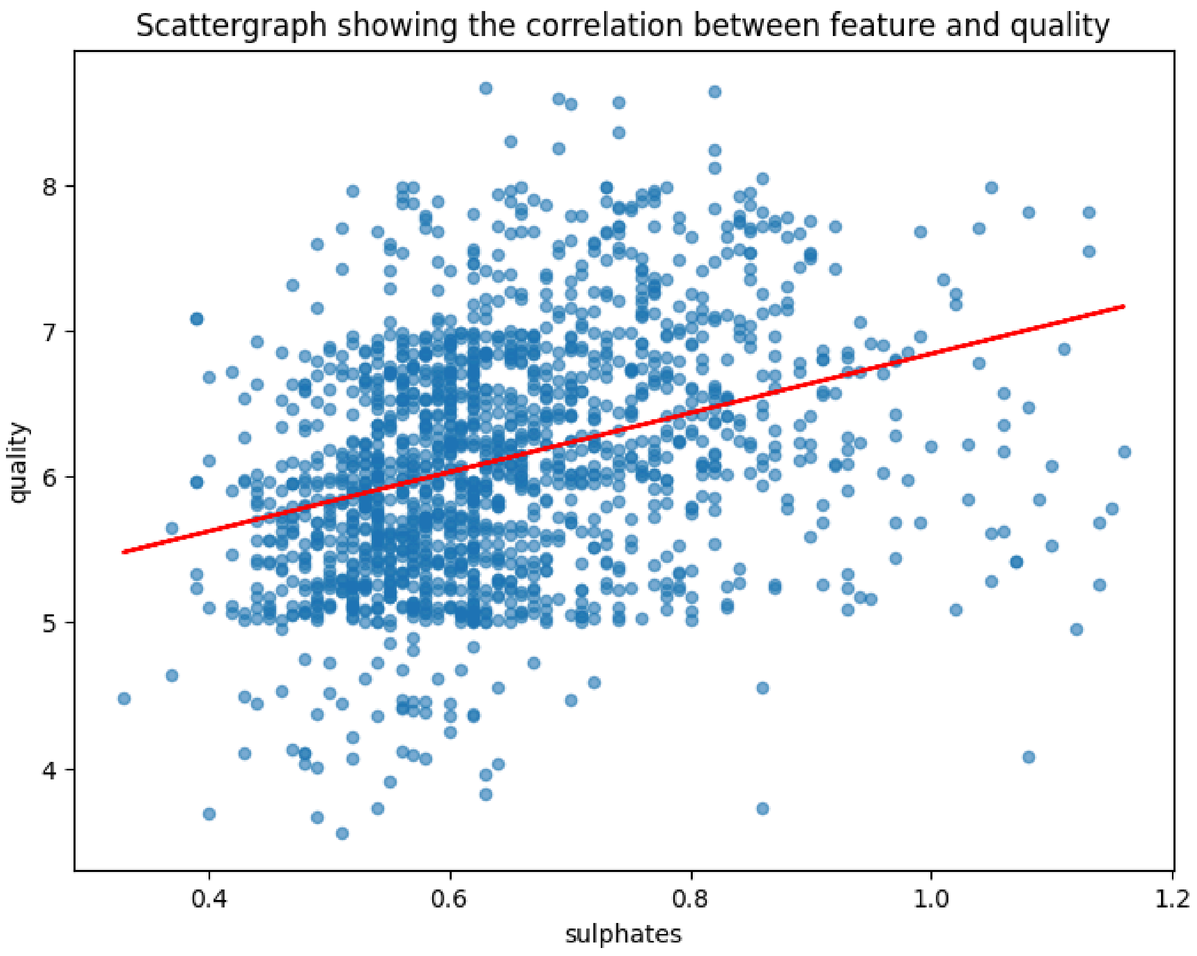


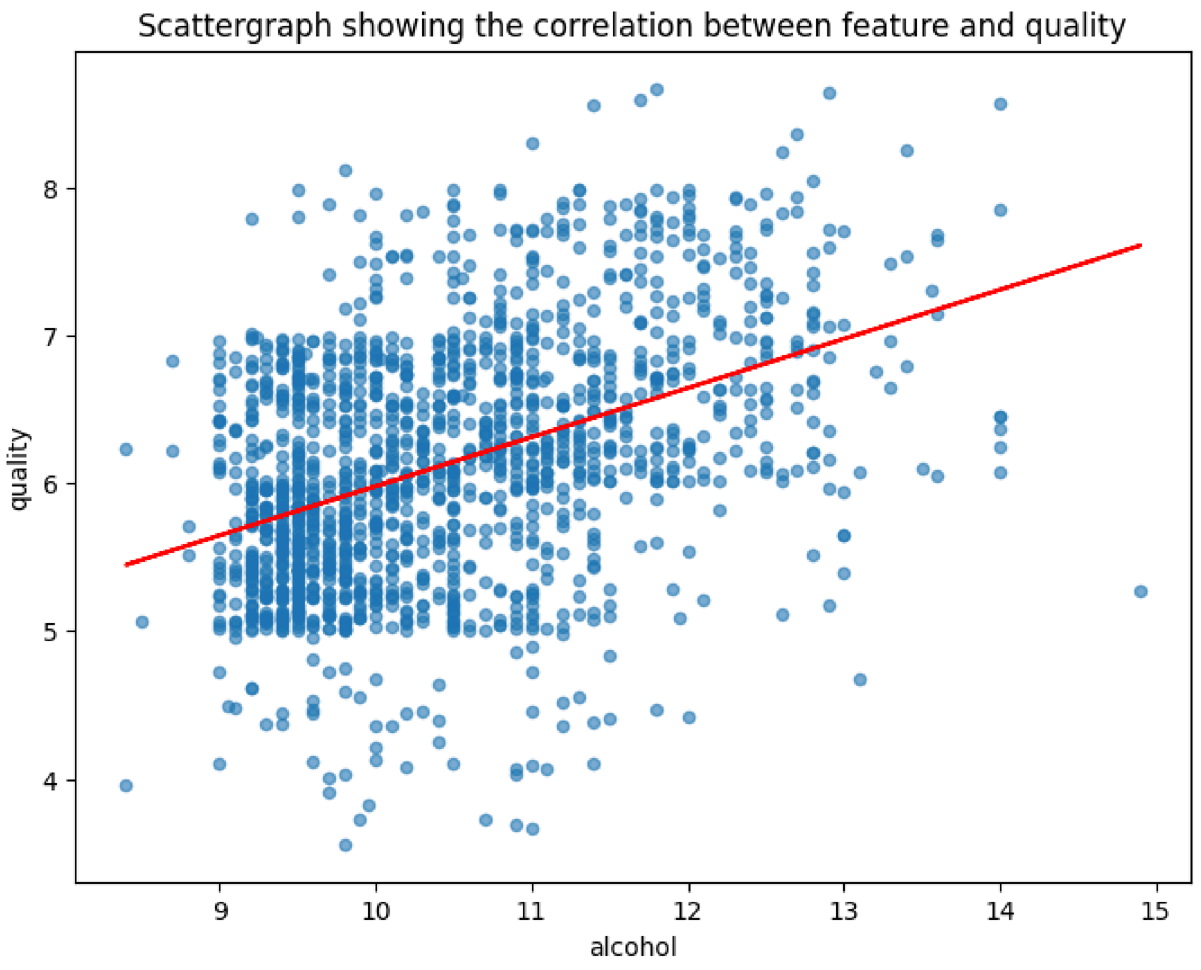
Scattergraph showing the correlation between feature and quality
quality
free sulfur dioxide
0
10
20
30
40
4
5
6
7
8


Scattergraph showing the correlation between feature and quality
quality
total sulfur dioxide
4
5
6
7
8
0
20
40
60
80
100
120
140


Scattergraph showing the correlation between feature and quality
quality
density
8
7
6
5
4
0.992
0.994
0.996
0.998
1.000
1.002


Scattergraph showing the correlation between feature and quality
quality
pH
8
7
6
5
4
3.0
3.2
3.4
3.6

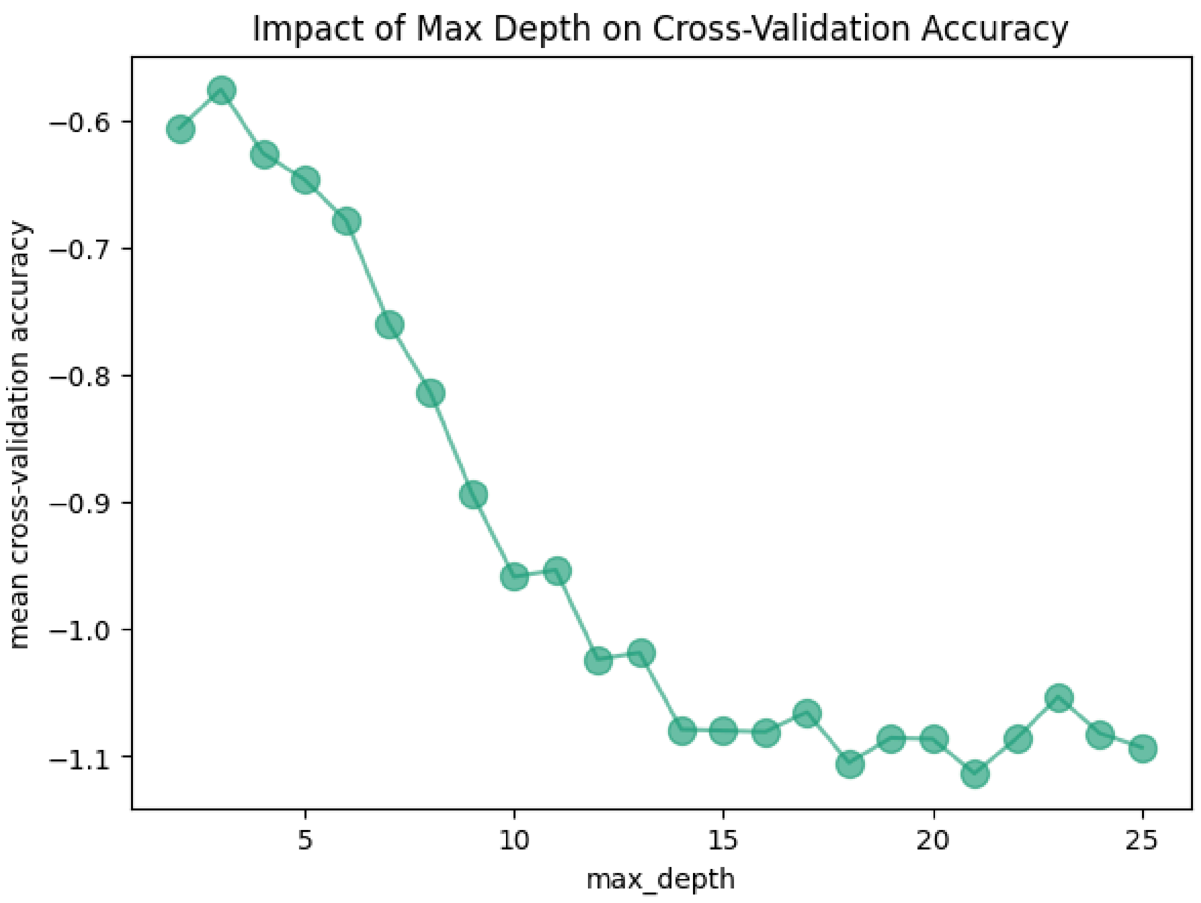

Scattergraph showing the correlation between feature and quality
quality
sulphates
4
5
6
7
8
0.4
0.6
0.8
1.0
1.2



Use hyperparameters

- Every model is likely to have hyperparameters that can be finetuned and optimised to aid in accuracy. For example for my Tree model I adjusted the max_depth to be the most optimal value by using cross validation which (as seen below) showed me that even just the difference of 5 can significantly reduce the model accuracy. There are other hyperparameters though e.g. Tree models can adjust the min_samples_split to force the tree to only make meaningful splits which helps to reduce overfitting. By finetuning hyperparameters for future models, we can optimise and have more accurate predictions.

```
In [31]: plt.figure(figsize=(7,5))#plot line graph
         plt.plot(depth_values, cv_scores, '-o',markersize=10, alpha=0.65,color='#1b9
         plt.title('Impact of Max Depth on Cross-Validation Accuracy')
         plt.xlabel('max_depth')
         plt.ylabel('mean cross-validation accuracy')
         plt.show()
```



Feature Scaling

```
In [32]: dfwine.describe()
```

Out[32]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free c
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.
mean	8.267855	0.523412	0.270513	2.392276	0.081935	14.
std	1.646141	0.169903	0.193945	0.880843	0.022761	8.
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.
25%	7.100000	0.390000	0.090000	1.900000	0.070000	8.
50%	7.900000	0.520000	0.260000	2.200000	0.079000	13.
75%	9.150000	0.635000	0.420000	2.600000	0.089000	19.
max	13.500000	1.040000	0.790000	6.700000	0.226000	45.

We can see the ranges 'min' and 'max' of the features are much larger than others for example total sulfur dioxide with a min of 6 and a max of 145 compared to citric acid with a min of 0 and max of 0.79. This can cause issues if we use models sensitive to feature scaling such as linear regression or K Means as it can introduce bias such as biased clusters(for K means). In our previous LR model it may explain why density with a much smaller range of around 0.01 seemed to have a much lower coefficient of -0.002873 compared to citric acid which has a larger range and a coefficient of -0.391229, therefore having more influence on the model. We can use the MinMaxScaler method to scale the wine features into the same range so they all have the same max and min/ same range. This can then reduce imbalance, increasing model interpretability and performance.

Use IQR to eliminate outliers

- Replacing the outliers with the median isn't always the most helpful, for example whilst this method decreased most of our columns' outliers, it increased the number of residual sugar and chloride outliers. Therefore by using the IQR we can remove or cap outliers which could produce cleaner data that maintains the data distribution better than the median method.

Appendix. Coursework Instructions

Coursework Support:

- COMP1008 computing tutorials and exercises on data processing and machine learning models on different example problems
- Example code building and analysing machine learning models in COMP1008 lectures slides on 'Machine learning'
- In the computing sessions, Q&A support for developing .ipynb projects
- In Teams channel 'COMP1008 2024/25 / Questions': support of common questions

Marks: in total 100 marks (count for 25% in COMP1008), awarded on the basis of:

- knowledge and understanding on the theories covered in lectures when answering the questions in the Jupyter Notebook report
- how informative and well presented your code, visualisations and results are (e.g. necessary labels in plots)
- self-learning ability making use of tutorial materials and online resources
- problem solving skills to obtain the answers and results for the specific dataset
- concise report with key details, e.g. parameters, data, etc. for others to repeat your methods and obtain the same results.

For more information of COMP1008 assessment please refer to the coursework issue in Moodle ('Course Content / Assessment').

Format:

- One single .ipynb file named 202425_COMP1008_cw_XXX.ipynb, where XXX is your username (e.g. psxyz)
- The .ipynb file should include your code and answers, using this given .ipynb template (please add cells as needed)
- You could use additional Python libraries as you wish, in addition to the ones demonstrated in the computing sessions
- There are multiple ways using different methods to complete the tasks. These are fine as long as all answers and analysis are supported by the code implemented in Jupyter Notebook, not by using other means (e.g. operations in Excel, or by using other languages, etc.).

Submission:

- Deadline: **24 March, 3pm**.
- Late submission leads to a 5% deduction of the coursework on each weekday. Work submitted one week late will receive a 0 for the coursework.
- Method: in Moodle submit a single .ipynb file named 202425_COMP1008_cw_XXX.ipynb
- If you can’t submit your coursework on time due to ECs, please contact Student Services and your personal tutor ASAP

Note: Plagiarism vs. Group Discussions

As you should know, plagiarism is completely unacceptable and will be dealt with according to University's standard policies.
Students are encouraged to have only general discussions on the theory (not the specific questions) when completing the coursework.
It is important that when you actually do your coursework and write the answers, you do it individually.
Do NOT, under any circumstances, share your report, code or figures, etc. with anyone else.