# Importing all the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('winequality-red.csv')
```

In [3]:
```
#Examining the first few rows of the dataset to understand its structure
df.head()
```

Out[3]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 |
| **1** | 7.8 | 0.88 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 |
| **2** | 7.8 | 0.76 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.9970 | 3.26 | 0.65 | 9.8 |
| **3** | 11.2 | 0.28 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.9980 | 3.16 | 0.58 | 9.8 |
| **4** | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 |

In [4]:
```
#Examining the last 5 rows of the dataset.
df.tail()
```

Out[4]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alco |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1594** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | |
| **1595** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | |
| **1596** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | |
| **1597** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | |
| **1598** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | |

# Printing details about the dataset

In [5]:

```python
#Retrieve an overview of the dataset's statistics, including count, mean,
print("Basic Information of the dataset: ")
print("--------------------------------------")
df.info()
df.describe()
```

```
Basic Information of the dataset: 
--------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Out[5]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulf dioxi |
|---|---|---|---|---|---|---|---|
| **count** | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.0000 |
| **mean** | 8.319637 | 0.527821 | 0.270976 | 2.538806 | 0.087467 | 15.874922 | 46.4677 |
| **std** | 1.741096 | 0.179060 | 0.194801 | 1.409928 | 0.047065 | 10.460157 | 32.8953 |
| **min** | 4.600000 | 0.120000 | 0.000000 | 0.900000 | 0.012000 | 1.000000 | 6.0000 |
| **25%** | 7.100000 | 0.390000 | 0.090000 | 1.900000 | 0.070000 | 7.000000 | 22.0000 |
| **50%** | 7.900000 | 0.520000 | 0.260000 | 2.200000 | 0.079000 | 14.000000 | 38.0000 |
| **75%** | 9.200000 | 0.640000 | 0.420000 | 2.600000 | 0.090000 | 21.000000 | 62.0000 |
| **max** | 15.900000 | 1.580000 | 1.000000 | 15.500000 | 0.611000 | 72.000000 | 289.0000 |

In [6]:
```
#Display the data types of each column in the dataset.
df.dtypes
```

Out[6]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [7]:
```
#Count the number of unique values in each column to understand data diver
df.nunique()
```

Out[7]:
```
fixed acidity            96
volatile acidity        143
citric acid              80
residual sugar           91
chlorides               153
free sulfur dioxide      60
total sulfur dioxide    144
density                 436
pH                       89
sulphates                96
alcohol                  65
quality                   6
dtype: int64
```

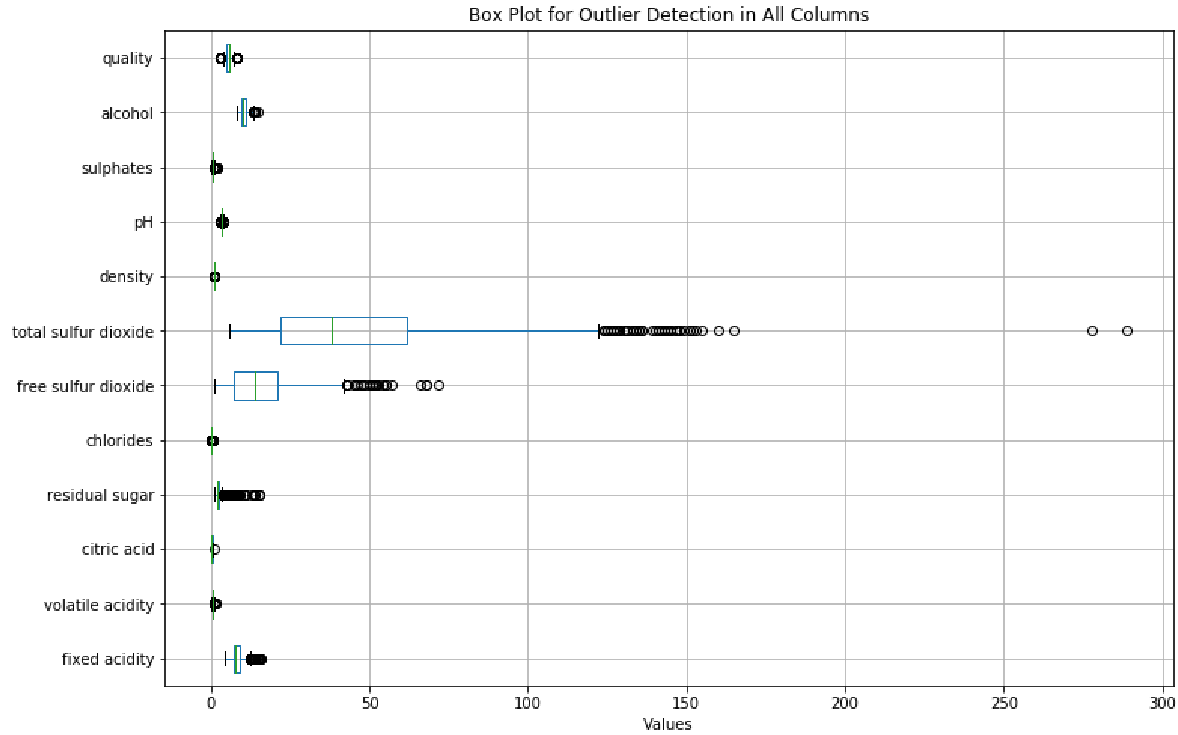
In [8]:

```
#Calculate the correlation between columns to identify relationships betwe
df.corr()
```

Out[8]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density |
|---|---|---|---|---|---|---|---|---|
| **fixed acidity** | 1.000000 | -0.256131 | 0.671703 | 0.114777 | 0.093705 | -0.153794 | -0.113181 | 0.668047 |
| **volatile acidity** | -0.256131 | 1.000000 | -0.552496 | 0.001918 | 0.061298 | -0.010504 | 0.076470 | 0.022026 |
| **citric acid** | 0.671703 | -0.552496 | 1.000000 | 0.143577 | 0.203823 | -0.060978 | 0.035533 | 0.364947 |
| **residual sugar** | 0.114777 | 0.001918 | 0.143577 | 1.000000 | 0.055610 | 0.187049 | 0.203028 | 0.355283 |
| **chlorides** | 0.093705 | 0.061298 | 0.203823 | 0.055610 | 1.000000 | 0.005562 | 0.047400 | 0.200632 |
| **free sulfur dioxide** | -0.153794 | -0.010504 | -0.060978 | 0.187049 | 0.005562 | 1.000000 | 0.667666 | -0.021946 |
| **total sulfur dioxide** | -0.113181 | 0.076470 | 0.035533 | 0.203028 | 0.047400 | 0.667666 | 1.000000 | 0.071269 |
| **density** | 0.668047 | 0.022026 | 0.364947 | 0.355283 | 0.200632 | -0.021946 | 0.071269 | 1.000000 |
| **pH** | -0.682978 | 0.234937 | -0.541904 | -0.085652 | -0.265026 | 0.070377 | -0.066495 | -0.341699 |
| **sulphates** | 0.183006 | -0.260987 | 0.312770 | 0.005527 | 0.371260 | 0.051658 | 0.042947 | 0.148506 |
| **alcohol** | -0.061668 | -0.202288 | 0.109903 | 0.042075 | -0.221141 | -0.069408 | -0.205654 | -0.496180 |
| **quality** | 0.124052 | -0.390558 | 0.226373 | 0.013732 | -0.128907 | -0.050656 | -0.185100 | -0.174919 |

In [9]:
```python
#Creating a box plot to identify potential outliers in the dataset by exam
# Create a box plot for each numerical column
plt.figure(figsize=(12, 8))
df.boxplot(column=list(df.columns[df.dtypes != 'object']), vert=False)
plt.title('Box Plot for Outlier Detection in All Columns')
plt.xlabel('Values')
plt.show()
```




Here we can see that the features with potential outliers are total sulfur dioxide, free sulfur dioxide, residual sugar, fixed acidity, etc. Basically, the plots which have datpoints away from the whiskers have outliers.

In [10]:
```python
# Calculate the sum of missing entries in the dataset for each column
missing_entries = df.isnull().sum(axis=0)
missing_entries
```

Out[10]:
```
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
```

Using the above output we can conclude that this datset contains no missing values. Hence, we do not need to drop or fill any features or columns.

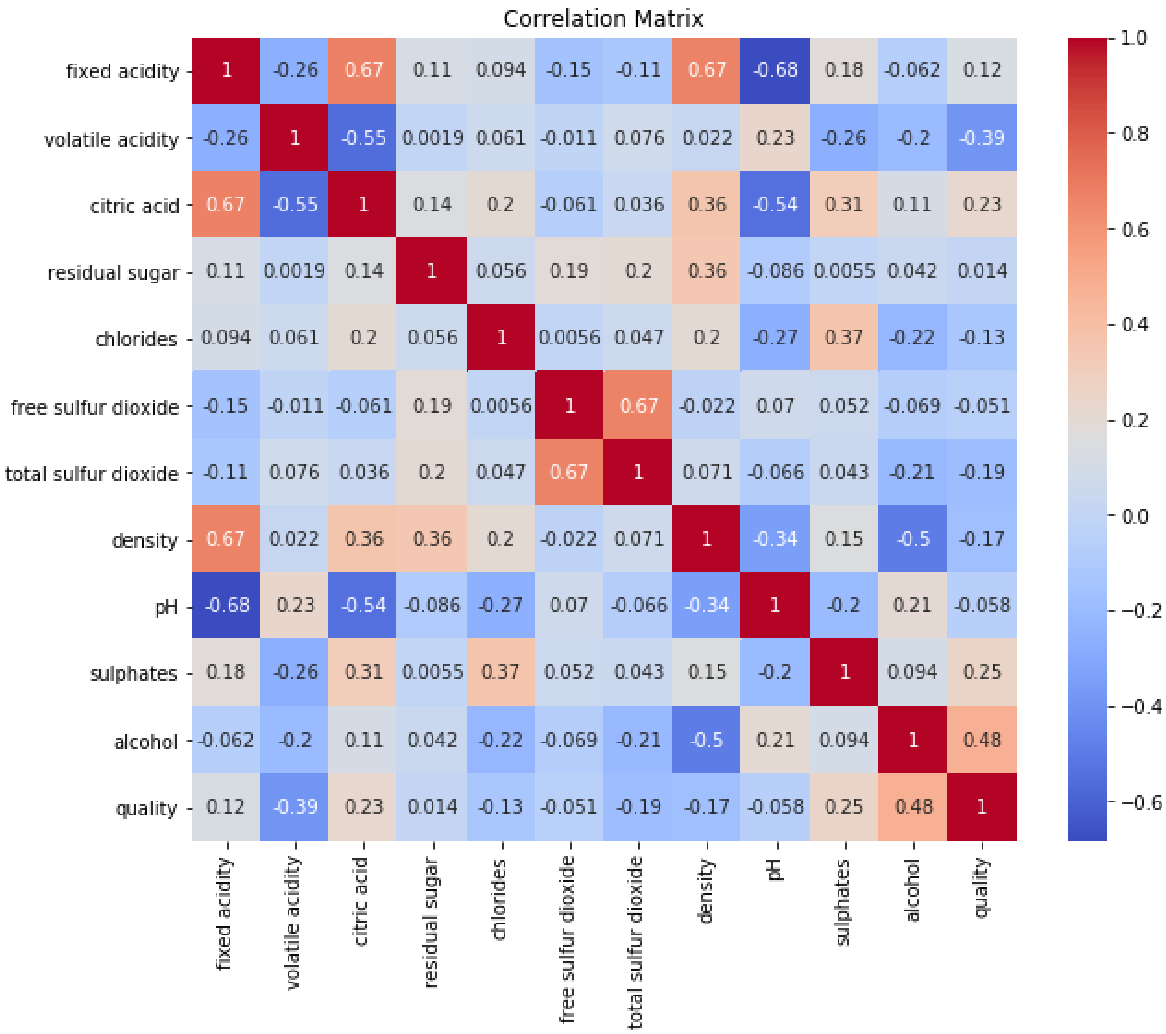
In [11]:
```
# Identify the columns with a data type of string (object)
string_columns = df.select_dtypes(include=['object']).columns
string_columns
```

Out[11]:
```
Index([], dtype='object')
```

From the above result we can conclude that the wine dataset does not contain any feature or column with string datatype.

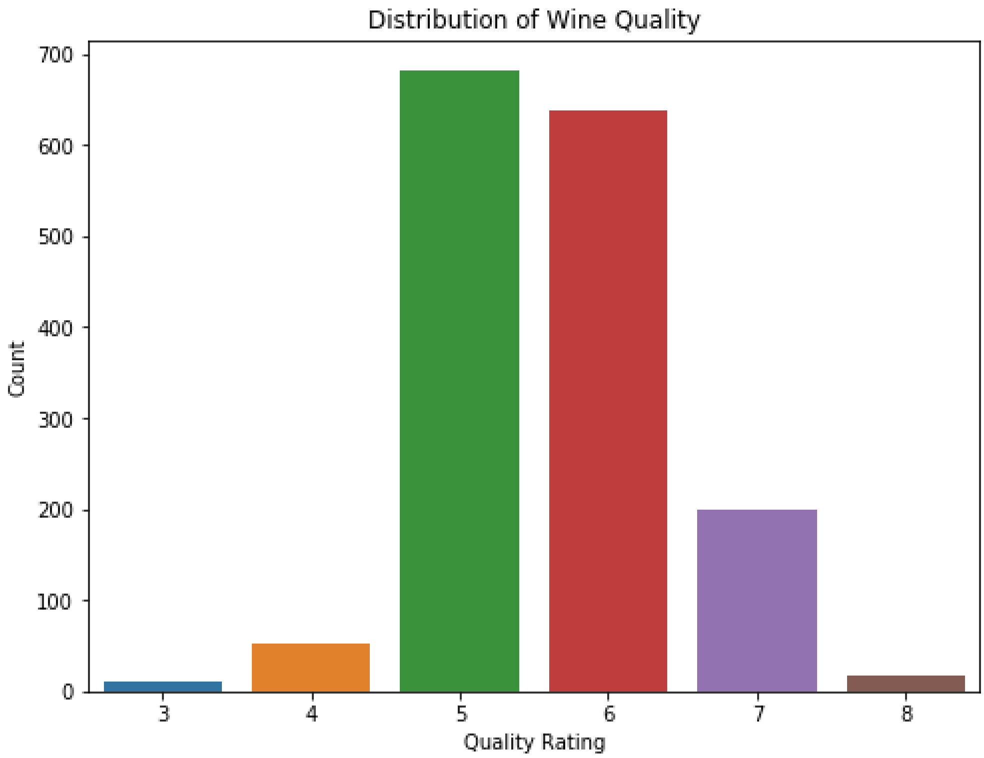
## Visualization of the data

In [12]:
```
#Visualize the correlation between numerical features using a heatmap. Thi
correlation_matrix = df.corr()

# Create a heatmap to visualize the correlation
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



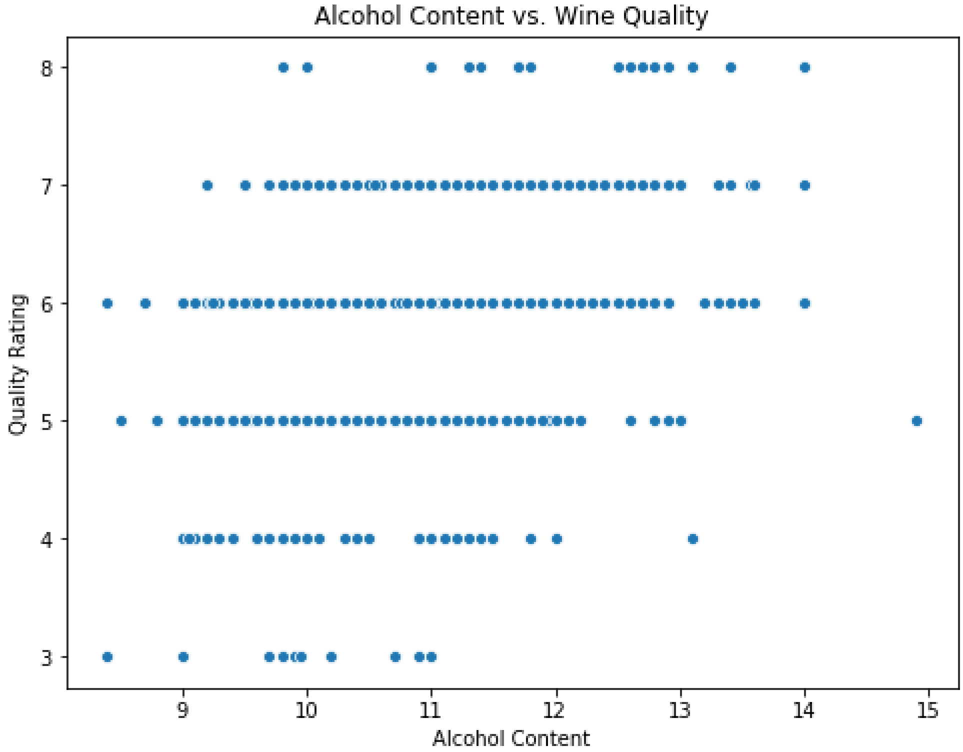
In [13]:

```
# Create a count plot for wine quality
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='quality')
plt.title("Distribution of Wine Quality")
plt.xlabel("Quality Rating")
plt.ylabel("Count")
plt.show()
```



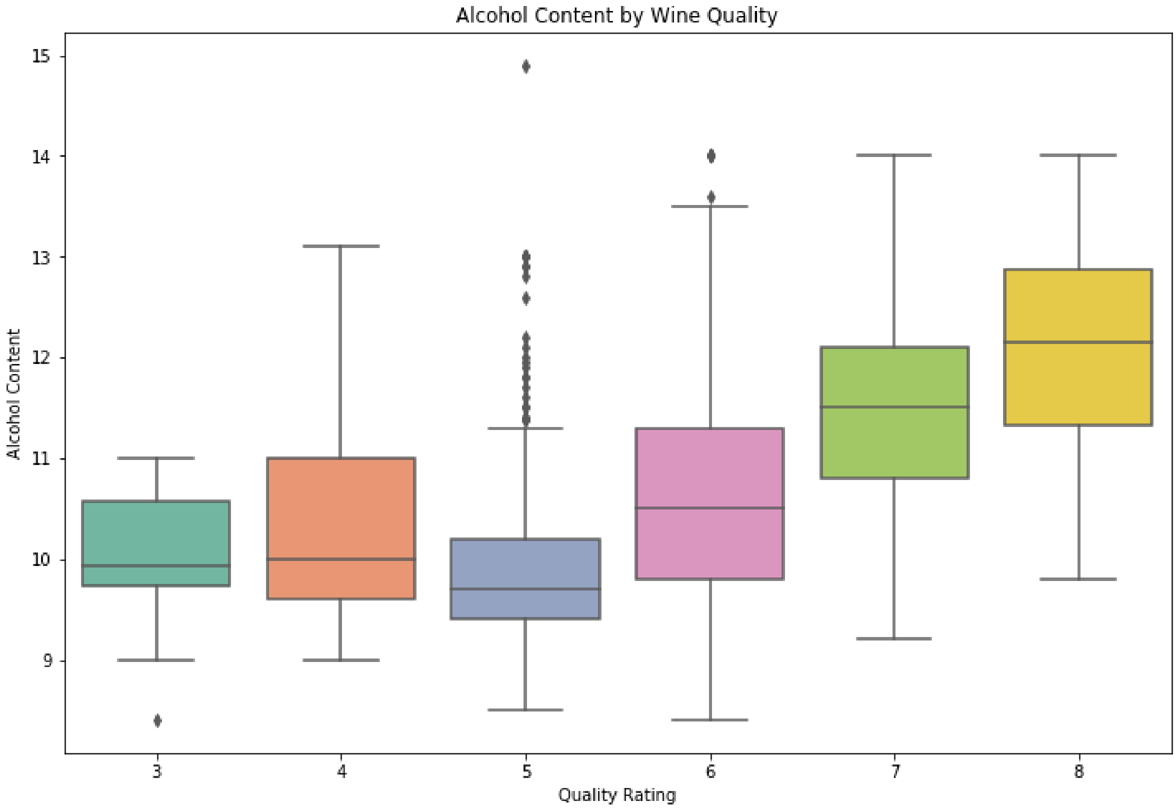
In [14]:

```
# Create a scatter plot for alcohol content vs. wine quality
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='alcohol', y='quality')
plt.title("Alcohol Content vs. Wine Quality")
plt.xlabel("Alcohol Content")
plt.ylabel("Quality Rating")
plt.show()
```

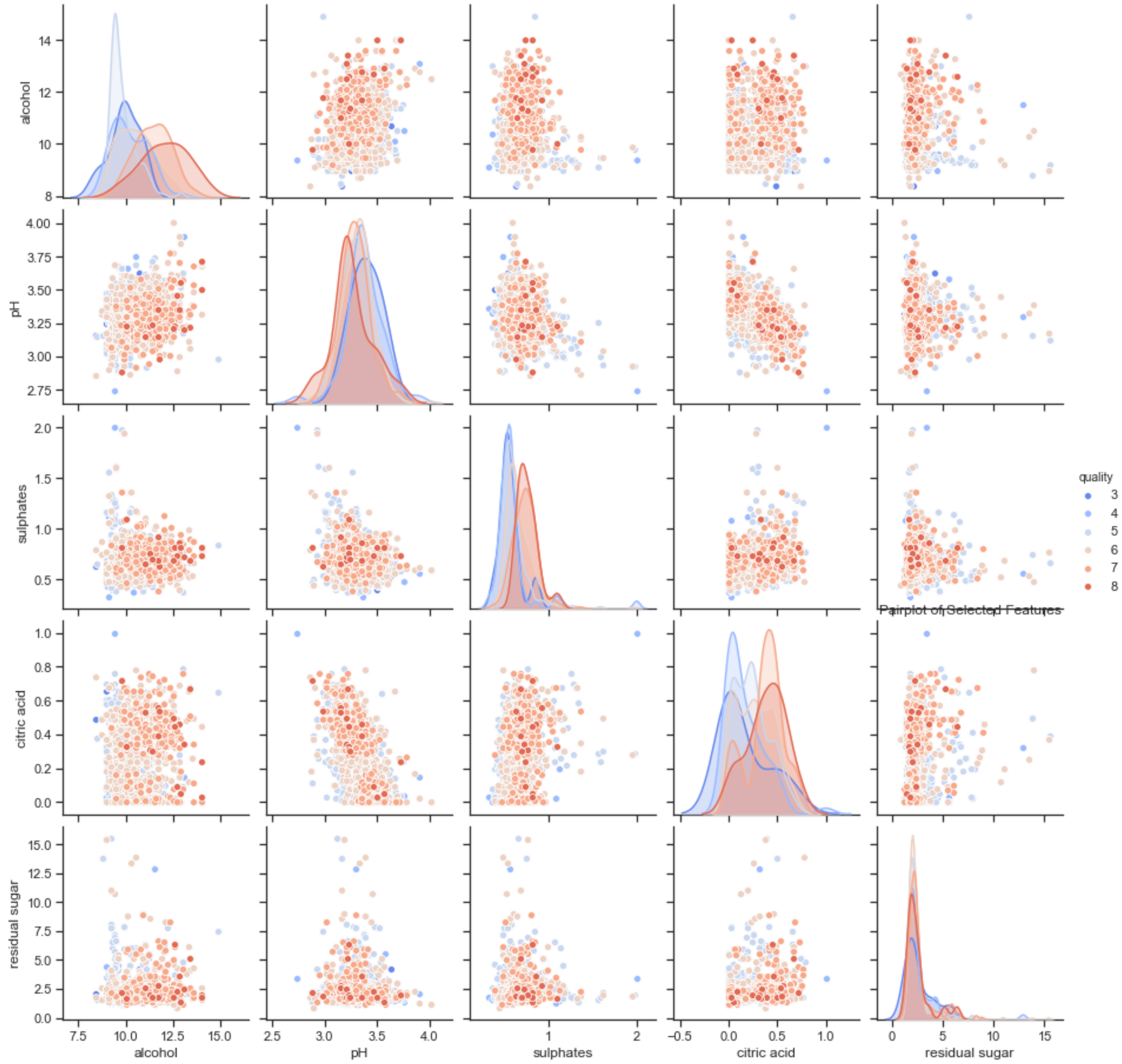


In [15]:

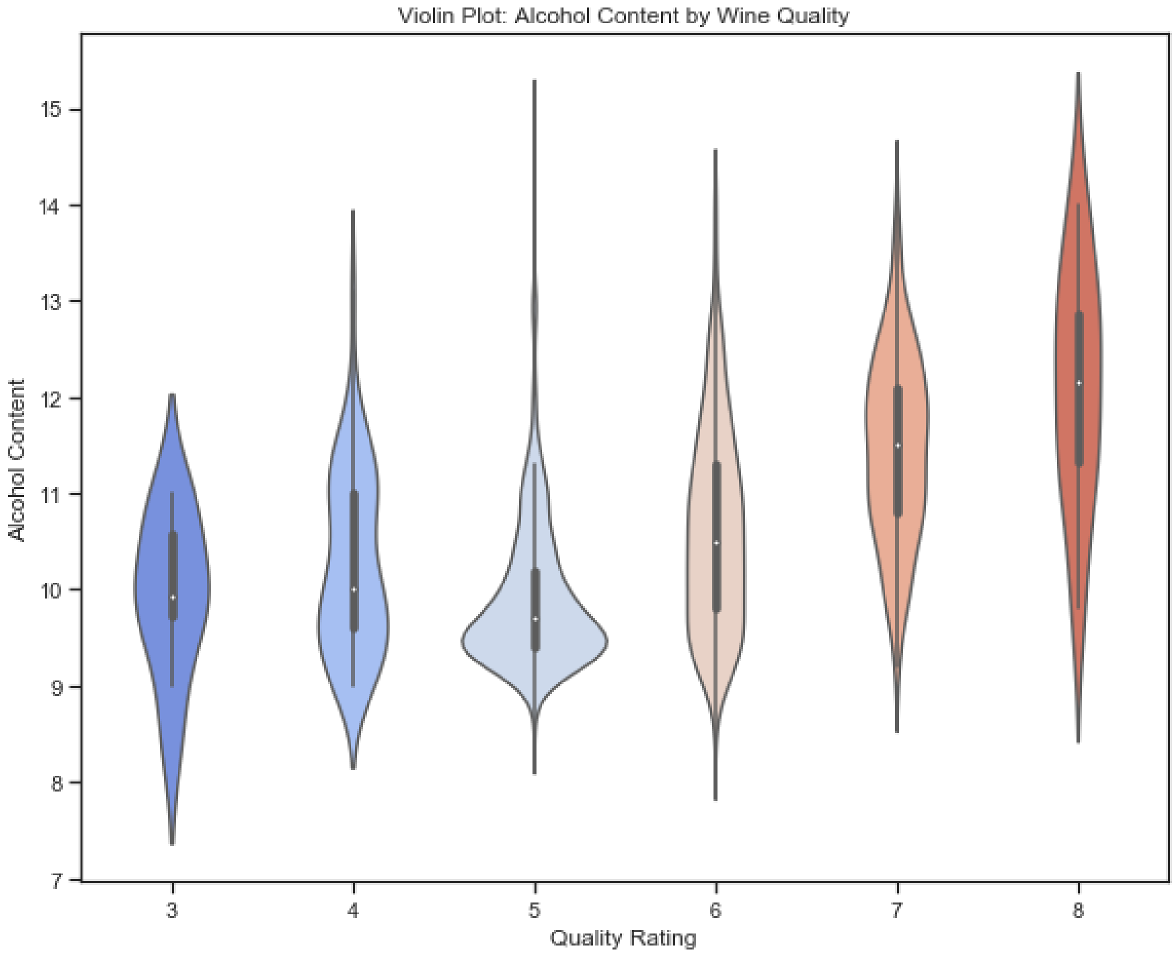
```python
# Create box plots for select features
plt.figure(figsize=(12, 8))
sns.boxplot(data=df, x='quality', y='alcohol', palette='Set2')
plt.title("Alcohol Content by Wine Quality")
plt.xlabel("Quality Rating")
plt.ylabel("Alcohol Content")
plt.show()
```



In [16]:
```python
features = ['alcohol', 'pH', 'sulphates', 'citric acid', 'residual sugar']

# Create a pairplot
sns.set(style='ticks')
sns.pairplot(df, vars=features, hue='quality', palette='coolwarm')
plt.title("Pairplot of Selected Features")
plt.show()
```



In [17]:
```
#A violin plot combines a box plot and a kernel density estimation. It sho
plt.figure(figsize=(10, 8))
sns.violinplot(data=df, x='quality', y='alcohol', palette='coolwarm')
plt.title("Violin Plot: Alcohol Content by Wine Quality")
plt.xlabel("Quality Rating")
plt.ylabel("Alcohol Content")
plt.show()
```




## Model building

In [27]:
```
# Assuming you have other features in the dataset, and 'pH' is the target
X = df.drop('pH', axis=1)  # Remove the 'pH' column from the features
y = df['pH']  # 'pH' column becomes the target variable
```

# Why we choose 'quality' as our target variable?

# Winemakers, sommeliers, and wine experts often assess wine quality based on various characteristics like taste, aroma, acidity, sweetness, and more. Using quality as the target variable allows you to model and predict wine quality, which can be of great interest to experts in the field.

In [28]:

```
# Verify the shapes of X and y
print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

```
Shape of X: (1599, 11)
Shape of y: (1599,)
```

In [29]:

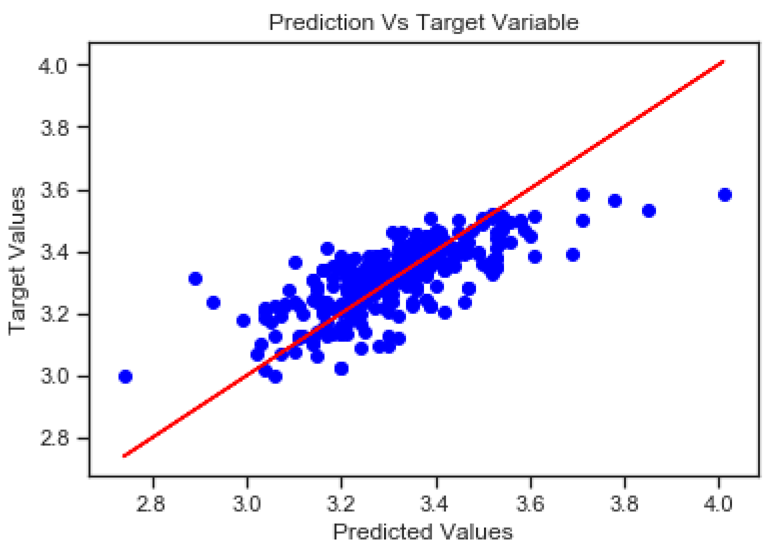
```
# Set a random seed for reproducibility
np.random.seed(42)

# Calculate the number of data points for training (80%) and testing (20%)
total_samples = len(df)
train_size = int(0.8 * total_samples)

# Shuffle the dataset
shuffled_indices = np.random.permutation(total_samples)

# Split the indices into training and testing sets
train_indices = shuffled_indices[:train_size]
test_indices = shuffled_indices[train_size:]

# Create the training and testing sets
X_train, y_train = X.iloc[train_indices], y.iloc[train_indices]
X_test, y_test = X.iloc[test_indices], y.iloc[test_indices]

# Verify the shapes of the training and testing sets
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (1279, 11)
Shape of y_train: (1279,)
Shape of X_test: (320, 11)
Shape of y_test: (320,)
```

In [36]:
```python
# Define a function to calculate weights for Linear Regression
def weights_calculation(x_train, y_train):
    XTX = np.dot(x_train.T, x_train)
    XTY = np.dot(x_train.T, y_train)
    weights = np.linalg.solve(XTX, XTY)
    return weights

# Calculate the weights using the training data
weights = weights_calculation(X_train, y_train)

# Calculate the predicted values on both training and test data
y_train_pred = np.dot(X_train, weights)
y_test_pred = np.dot(X_test, weights)

# Calculate Mean Squared Error (MSE) for training and testing
mse_train = np.mean((y_train - y_train_pred) ** 2)
mse_test = np.mean((y_test - y_test_pred) ** 2)

# Print the MSE for training and testing
print("MSE Loss (Training):", mse_train)
print("MSE Loss (Testing):", mse_test)

# Print the final weight vector
print("Final Weight Vector (Coefficients):")
print(weights)
```

```
MSE Loss (Training): 0.010137170255481082
MSE Loss (Testing): 0.010530904960440162
Final Weight Vector (Coefficients):
[-5.61467033e-02  8.16871135e-02 -3.55857662e-02 -4.82114024e-04
 -5.79500321e-01  1.60117311e-03 -9.09249781e-04  3.63428464e+00
  3.67348383e-02  2.57918582e-02 -1.79626780e-02]
```

In [31]:
```python
# Create a scatter plot to visualize the predictions
plt.scatter(y_test, y_pred, color='blue')
plt.plot(y_test, y_test, color='red')
plt.title("Prediction Vs Target Variable")
plt.xlabel("Predicted Values")
plt.ylabel("Target Values")
plt.show()
```



In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```