Introduction

The primary objective of this research is to develop a Random Forest model to classify the quality of wine based on its chemical properties. By leveraging the ensemble learning capabilities of Random Forest, we aim to create an accurate and robust classifier that can assist in the quality assessment of wines, benefiting both producers and consumers.

Importing Necessary Libraries

```
#Importing required packages.
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
%matplotlib inline
```

Loading the dataset

```
#Loading dataset
wine = pd.read_csv('/content/winequality-red.csv')
```

```
#Let's check how the data is distributed
wine.head()
```

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

Next steps: Generate code with wine View recommended plots

```
print(wine.columns)
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
#Information about the data columns
wine.info()
```

```
<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
```

```
# Check for missing values
print(wine.isnull().sum())
```

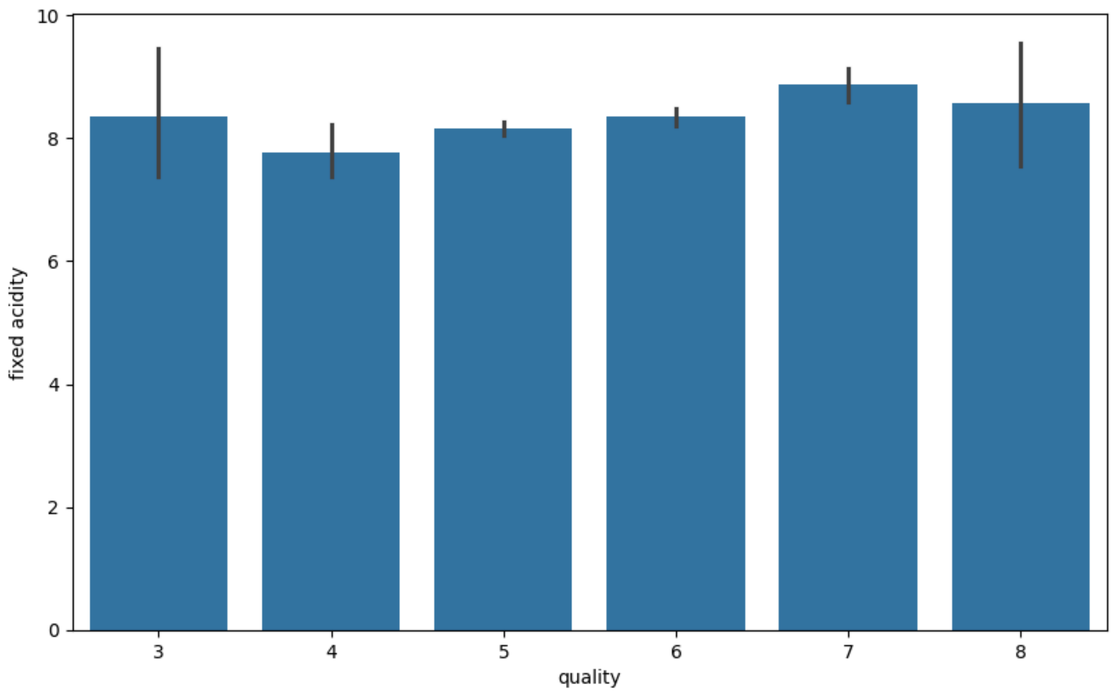
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality   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

Data Visualization

Let's do some plotting to know how the data columns are distributed in the dataset

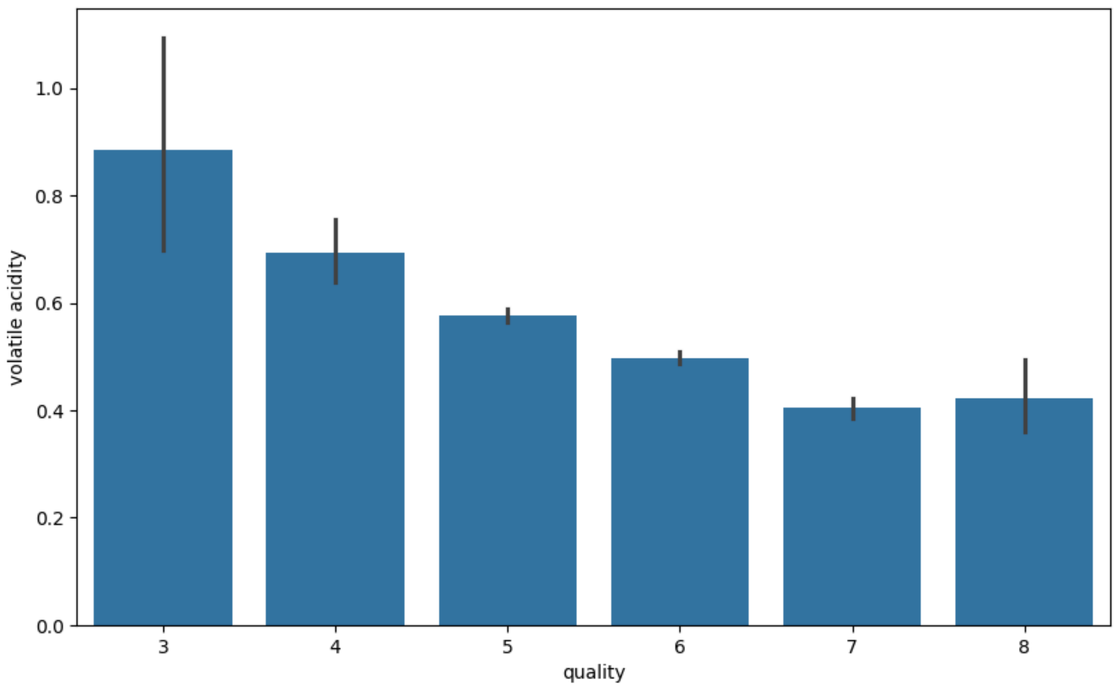
```
#Here we see that fixed acidity does not give any specification to classify the quality.
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'fixed acidity', data = wine)
```

```
<Axes: xlabel='quality', ylabel='fixed acidity'>
```

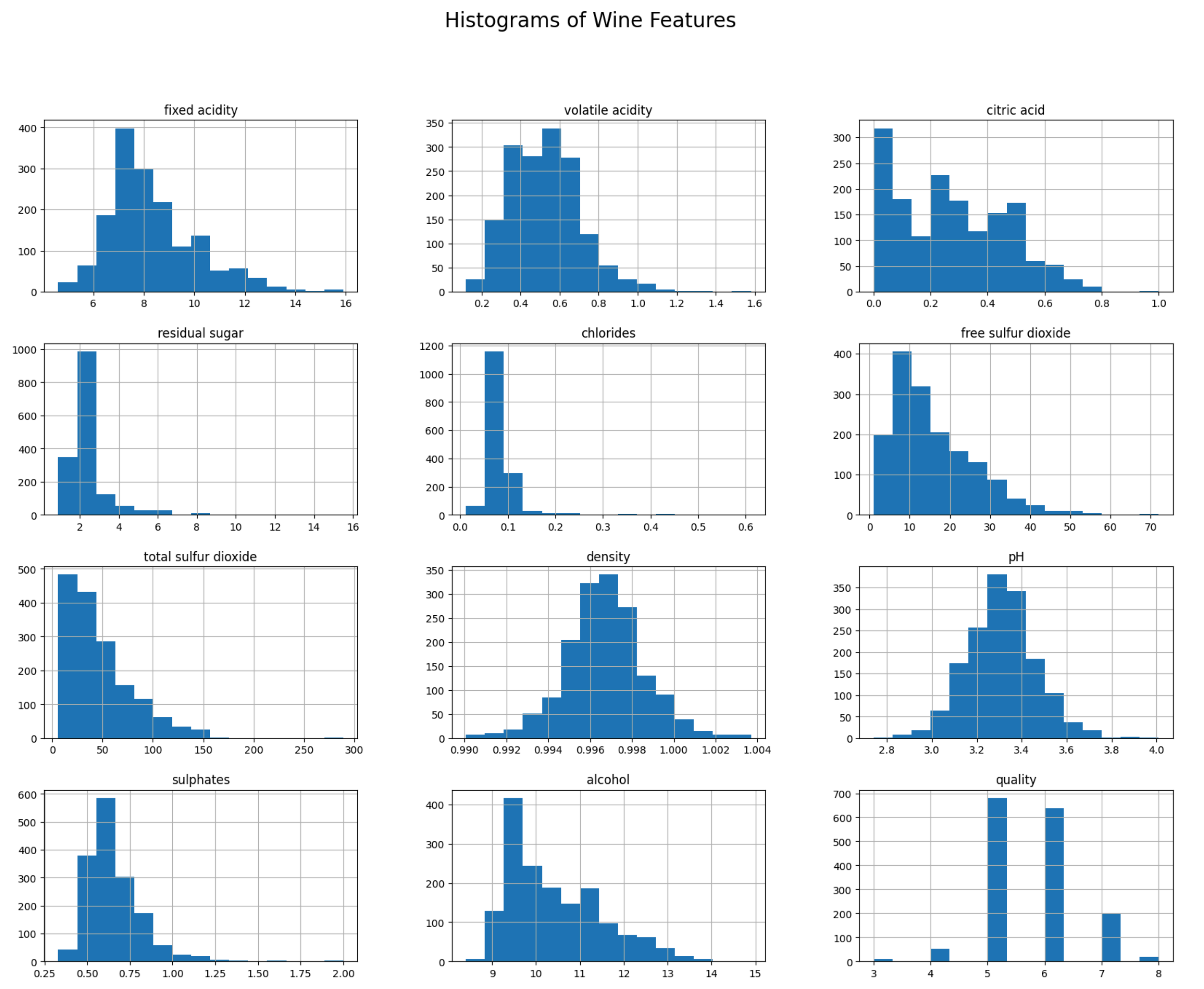


```
#Here we see that its quite a downing trend in the volatile acidity as we go higher the quality
fig = plt.figure(figsize = (10,6))
sns.barplot(x = 'quality', y = 'volatile acidity', data = wine)
```

```
<Axes: xlabel='quality', ylabel='volatile acidity'>
```

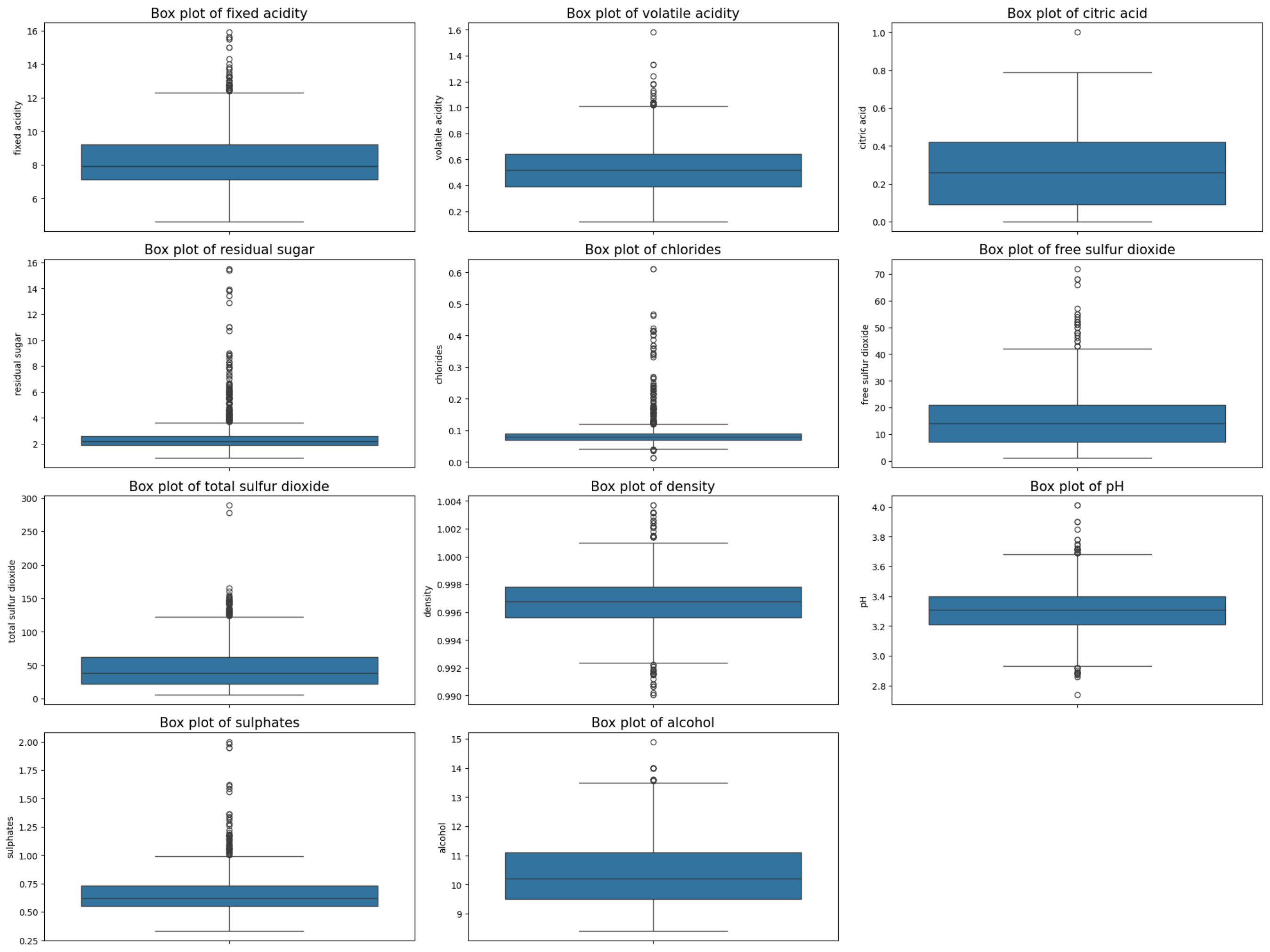


```
# Histograms for the distribution of each feature
wine.hist(bins=15, figsize=(20, 15), layout=(4, 3))
plt.suptitle('Histograms of Wine Features', size=20)
plt.show()
```



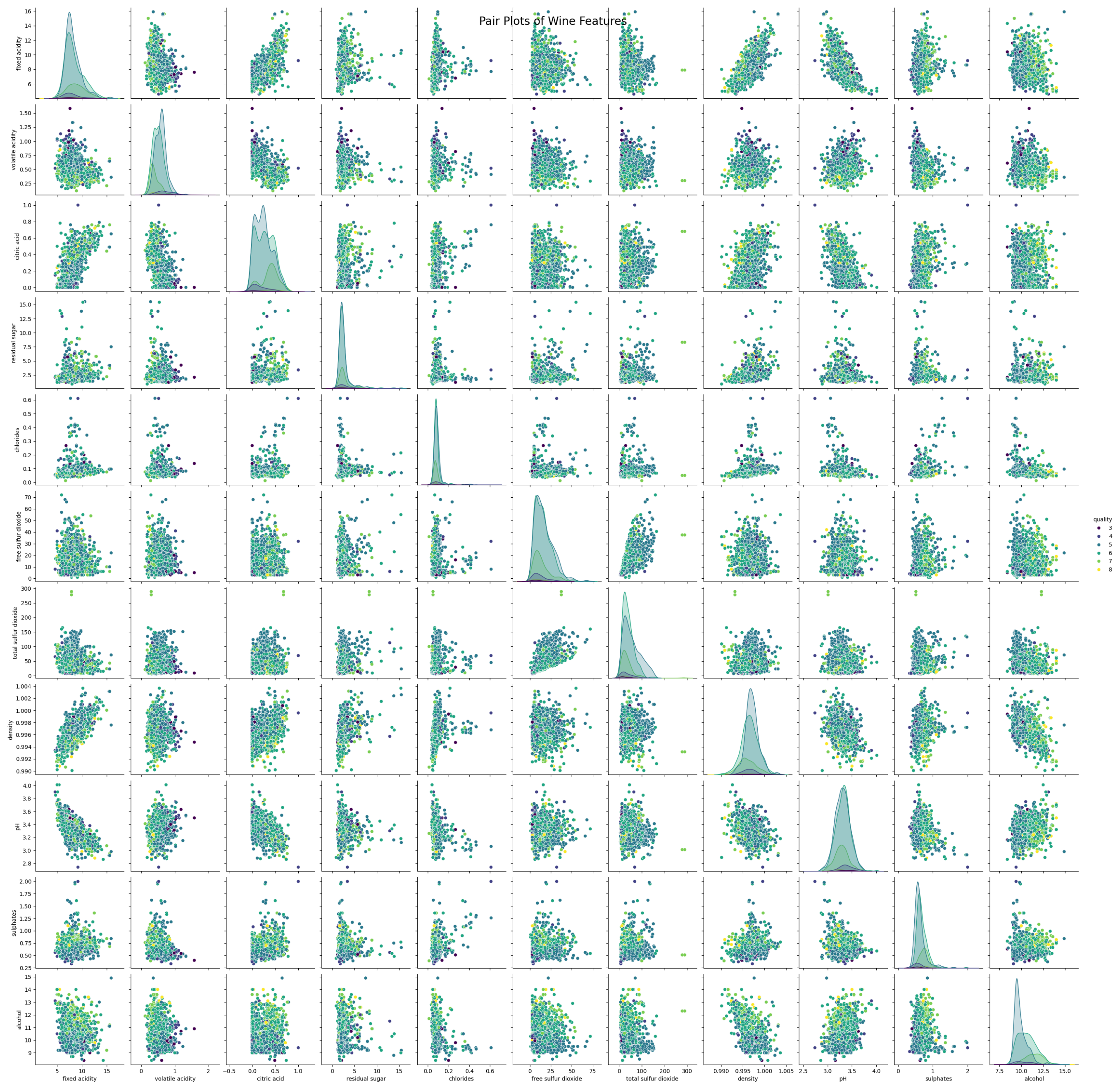
Understand Distribution: Histograms provide a visual representation of the frequency distribution of a dataset. This helps to understand the shape of the data distribution for each feature, including the central tendency, variability, and skewness. Identify Outliers: They help to identify outliers and the presence of any extreme values in the dataset. Detect Data Imbalance: Histograms can reveal if there is any imbalance in the distribution of data points across different ranges.

```
# Box plots for each feature to visualize spread and outliers
plt.figure(figsize=(20, 15))
for i, column in enumerate(wine.columns[:-1], 1):
    plt.subplot(4, 3, i)
    sns.boxplot(data=wine, y=column)
    plt.title(f'Box plot of {column}', size=15)
plt.tight_layout()
plt.show()
```



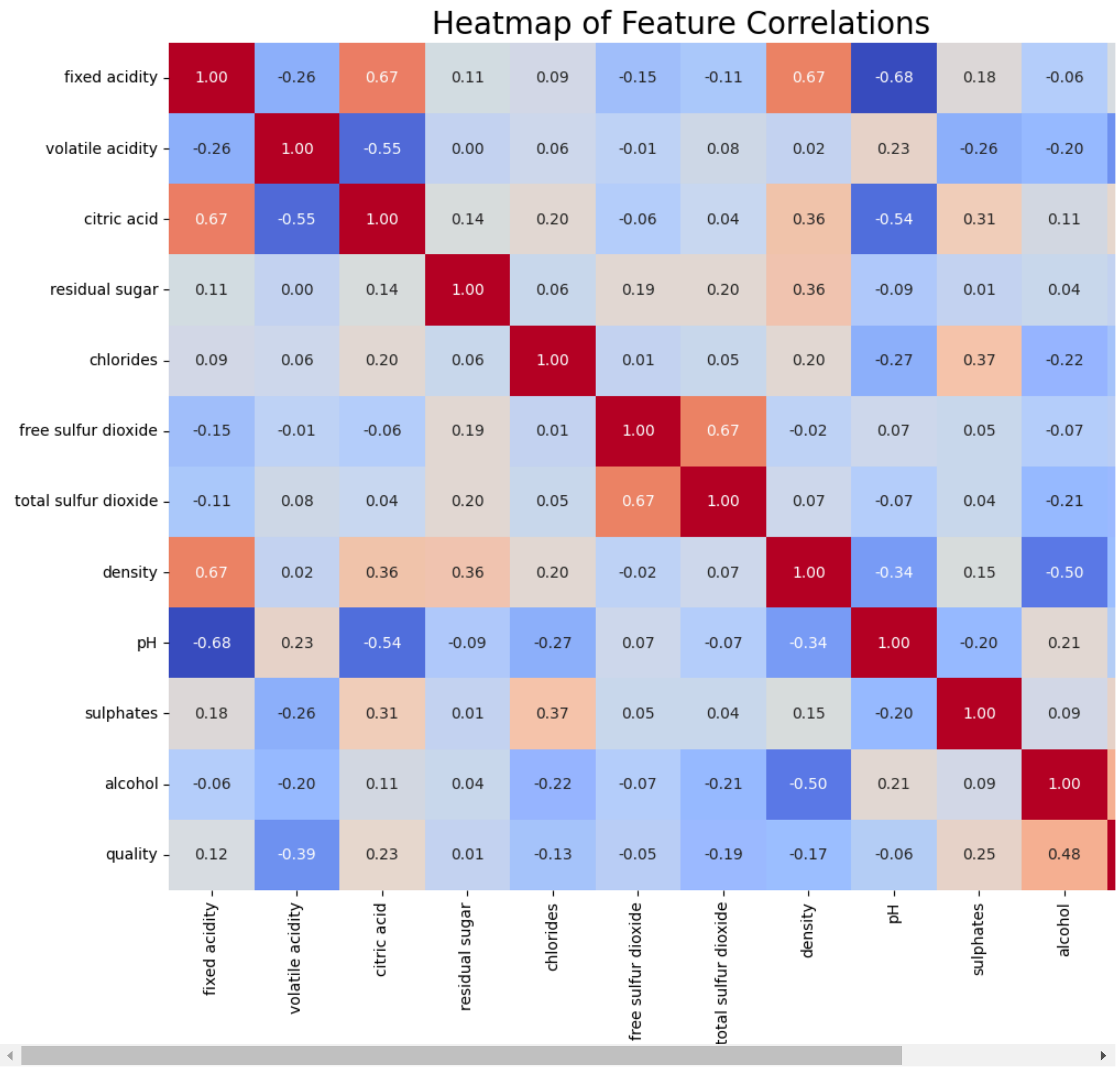
Summary Statistics: Box plots show the median, quartiles, and potential outliers in the data, providing a summary of the distribution. Detect Outliers: They are effective in identifying outliers and understanding the range and distribution of the data. Compare Distributions: Box plots allow for easy comparison of distributions across different features.

```
# Pair plots to see pairwise relationships and distributions
sns.pairplot(wine, hue='quality', palette='viridis')
plt.suptitle('Pair Plots of Wine Features', size=20)
plt.show()
```



Visualize Relationships: Pair plots allow for the visualization of relationships between pairs of features. This can help to identify potential correlations and interactions between features. Multi-dimensional Analysis: They provide a way to look at the distribution of each feature as well as their relationships in a multi-dimensional context. Class Separation: By adding hue based on the target variable (quality), pair plots can show how different classes are distributed in the feature space.

```
# Heatmap to visualize correlations between features
plt.figure(figsize=(15, 10))
corr_matrix = wine.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Heatmap of Feature Correlations', size=20)
plt.show()
```

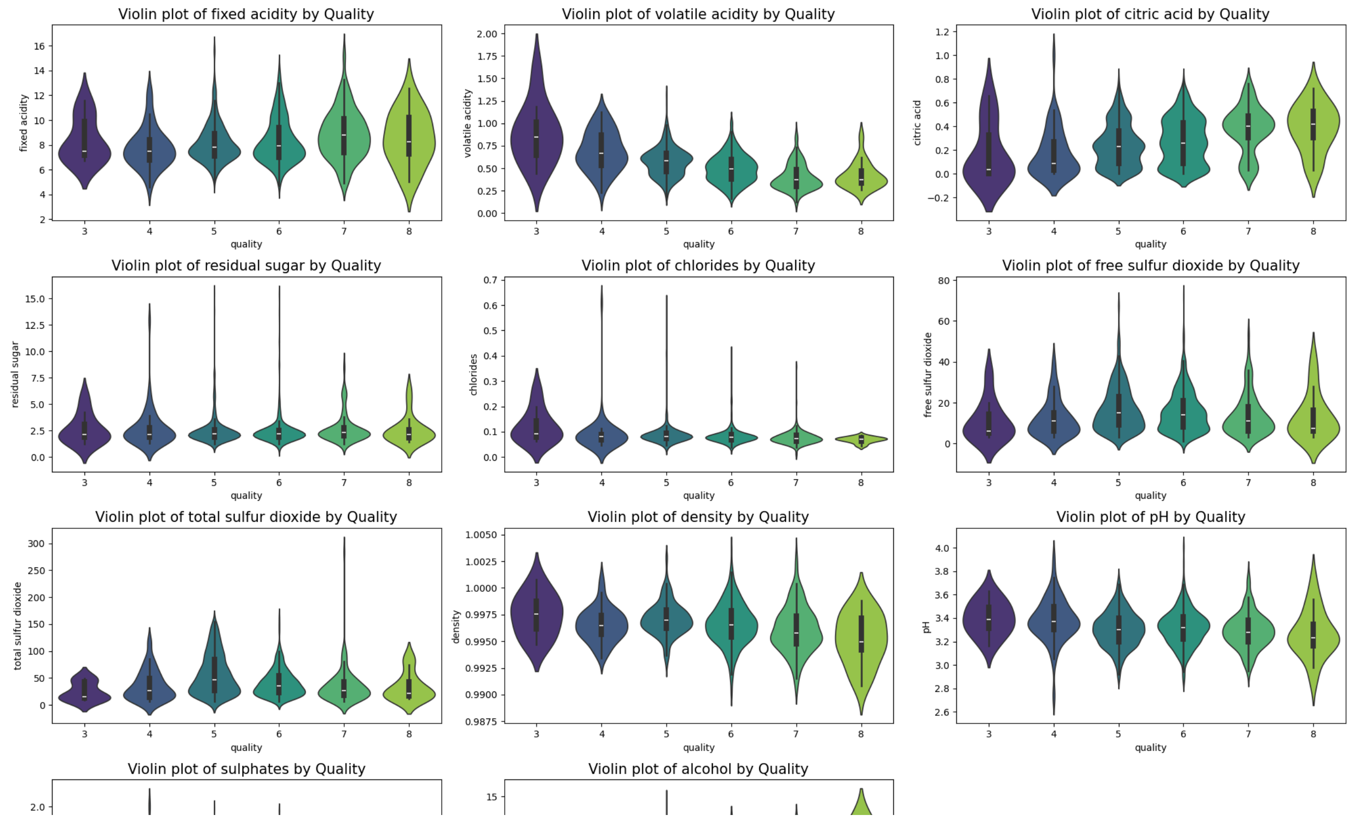


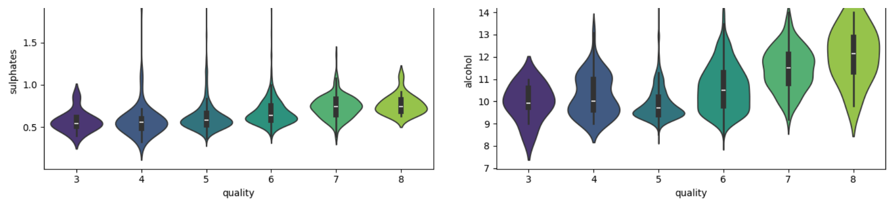
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity	1.00	-0.26	0.67	0.11	0.09	-0.15	-0.11	0.67	-0.68	0.18	-0.06
volatile acidity	-0.26	1.00	-0.55	0.00	0.06	-0.01	0.08	0.02	0.23	-0.26	-0.20
citric acid	0.67	-0.55	1.00	0.14	0.20	-0.06	0.04	0.36	-0.54	0.31	0.11
residual sugar	0.11	0.00	0.14	1.00	0.06	0.19	0.20	0.36	-0.09	0.01	0.04
chlorides	0.09	0.06	0.20	0.06	1.00	0.01	0.05	0.20	-0.27	0.37	-0.22
free sulfur dioxide	-0.15	-0.01	-0.06	0.19	0.01	1.00	0.67	-0.02	0.07	0.05	-0.07
total sulfur dioxide	-0.11	0.08	0.04	0.20	0.05	0.67	1.00	0.07	-0.07	0.04	-0.21
density	0.67	0.02	0.36	0.36	0.20	-0.02	0.07	1.00	-0.34	0.15	-0.50
pH	-0.68	0.23	-0.54	-0.09	-0.27	0.07	-0.07	-0.34	1.00	-0.20	0.21
sulphates	0.18	-0.26	0.31	0.01	0.37	0.05	0.04	0.15	-0.20	1.00	0.09
alcohol	-0.06	-0.20	0.11	0.04	-0.22	-0.07	-0.21	-0.50	0.21	0.09	1.00
quality	0.12	-0.39	0.23	0.01	-0.13	-0.05	-0.19	-0.17	-0.06	0.25	0.48

Correlation Analysis: Heatmaps display the correlation matrix, which shows the strength and direction of relationships between pairs of features. Feature Selection: Identifying highly correlated features can inform decisions about feature selection and engineering. Pattern Recognition: They help to recognize patterns and relationships that might not be immediately obvious.

```
# Violin plots to show the distribution of the data across different quality levels
plt.figure(figsize=(20, 15))
for i, column in enumerate(wine.columns[:-1], 1):
    plt.subplot(4, 3, i)
    sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
    plt.title(f'Violin plot of {column} by Quality', size=15)
plt.tight_layout()
plt.show()
```

```
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
<ipython-input-15-abf5486b5155>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.violinplot(data=wine, x='quality', y=column, palette='viridis')
```

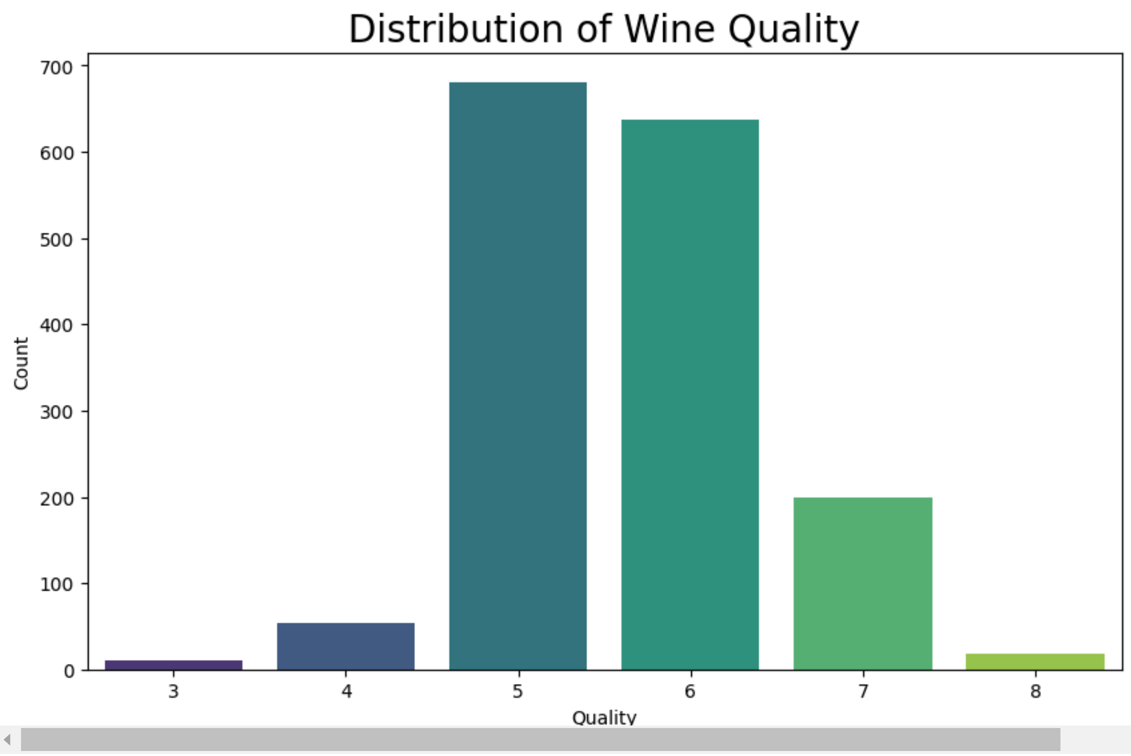




Distribution Analysis: Violin plots combine aspects of box plots and density plots, showing the distribution shape, central tendency, and variability. Class Comparison: They allow for the comparison of the distribution of a feature across different categories of the target variable (quality). Detailed Insight: Violin plots provide more detail about the distribution compared to box plots, including the density of the data points at different values.

```
# Bar plot for the distribution of the target variable (quality)
plt.figure(figsize=(10, 6))
sns.countplot(x='quality', data=wine, palette='viridis')
plt.title('Distribution of Wine Quality', size=20)
plt.xlabel('Quality')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-16-1f06f8baba98>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(x='quality', data=wine, palette='viridis')
```



Class Distribution: Bar plots provide a clear visualization of the frequency distribution of categorical data, in this case, the quality of the wine. Detect Imbalance: They help in detecting any imbalance in the target classes, which is important for model training and evaluation. Simple and Effective: Bar plots are straightforward and effective for comparing the sizes of different categories.

Preprocessing Data for performing Machine learning algorithms

```
#Making binary classificaion for the response variable.
#Dividing wine as good and bad by giving the limit for the quality
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group_names)
```

```
#Now lets assign a labels to our quality variable
label_quality = LabelEncoder()
```

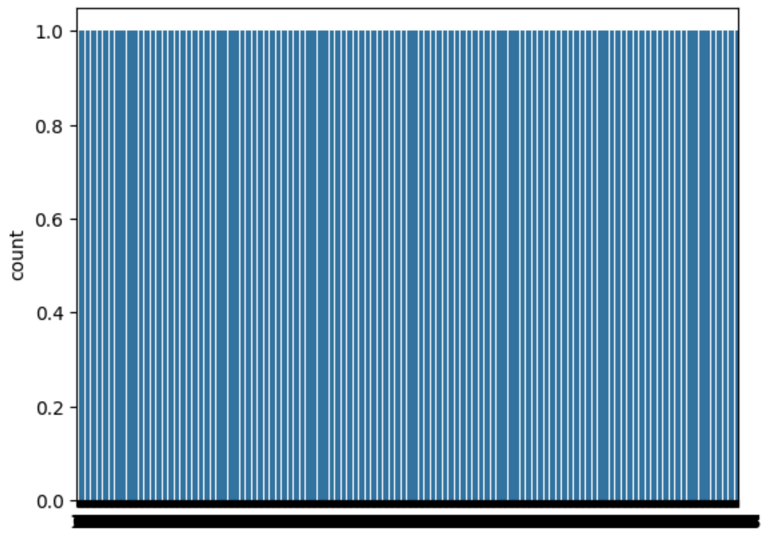
```
#Bad becomes 0 and good becomes 1
wine['quality'] = label_quality.fit_transform(wine['quality'])
```

```
wine['quality'].value_counts()
```

```
quality
0    1382
1     217
Name: count, dtype: int64
```

```
sns.countplot(wine['quality'])
```

```
<Axes: ylabel='count'>
```



```
#Now seperate the dataset as response variable and feature variabes
X = wine.drop('quality', axis = 1)
y = wine['quality']
```

```
#Train and Test splitting of data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
#Applying Standard scaling to get optimized result
sc = StandardScaler()
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Our training and testing data is ready now to perform machine learning algorithm

Random Forest Classifier

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
                        RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from matplotlib.colors import ListedColormap

# Load the dataset
wine = pd.read_csv('/content/winequality-red.csv')

# Select two features for visualization purposes
features = ['alcohol', 'volatile acidity']
X = wine[features]
y = wine['quality']

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Train the Random Forest Classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train, y_train)

# Function to visualize decision boundaries
def plot_decision_boundaries(X, y, classifier, features):
    X_set, y_set = X, y
    X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1, stop=X_set[:, 0].max() + 1, step=0.01),
                         np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1, step=0.01))
    plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
                 alpha=0.75, cmap=ListedColormap(('red', 'green', 'blue', 'purple', 'orange', 'brown')))
    plt.xlim(X1.min(), X1.max())
    plt.ylim(X2.min(), X2.max())
    for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                    c=ListedColormap(('red', 'green', 'blue', 'purple', 'orange', 'brown'))(i), label=j)
    plt.title('Random Forest (Training set)')
    plt.xlabel(features[0])
    plt.ylabel(features[1])
    plt.legend()
    plt.show()

# Plot decision boundaries for the training set
plot_decision_boundaries(X_train, y_train, classifier, features)
```