Islamic University – Gaza
Faculty of Information Technology
Department of Information
Technology

الجامعة الإسلامية – غزة
كلية تكنولوجيا المعلومات
ماجستير تكنولوجيا المعلومات

Comprehensive Evaluation of Classifiers for Wine Quality Detection

A Case Study Using the Wine Quality dataset

By:

Abeer Yousef Abu Mosameh -220232641

Supervised By:

Dr. Iyad Husni Alshami

Dec, 2024

Introduction

The field of machine learning aims to enable computers to learn from data and make decisions or predictions without explicit instructions. One of the most common tasks in machine learning is classification, where we predict a label for an input based on its features.

In this report, we will train, evaluate and compare the performance of different classifiers in predicting the quality of wine using the Wine Quality dataset. Our primary goal is to use various classifiers for predicting wine quality and identify the most accurate approach. First we combines red and white wine datasets to make one dataset, then perform required preprocessing steps in dataset, and trains four classifiers 1) Support Vector Machine (SVM), 2) XGBoost, 3) Random Forest, and 4) Multilayer Perceptron (MLP) the analyzing the accuracy, F1-Score, and ROC/AUC metrics of four different classifiers, and then identify the most effective model for wine quality detection. Finally, comparing the performance and identify the best classifier between them for identifying the wine quality.

Dataset Preparation

The Wine Quality dataset consists of wine and white wine datasets (6497 row , 13 attribute) . These datasets contain several features related to wine physicochemical properties and the quality rating.

Attributes / Feature Description

Column Name	Description	Data Type	Attribute Type
fixed_acidity	The concentration of fixed acids in wine, which affects its taste and stability. Measured in grams per liter.	float64	Numeric, Ratio-Scaled
volatile_acidity	The concentration of volatile acids, mainly acetic acid, that can affect the wine's aroma and taste. Higher levels may indicate spoilage. Measured in grams per liter.	float64	Numeric, Ratio-Scaled
citric_acid	The amount of citric acid in the wine, contributing to its freshness and flavor. Measured in grams per liter.	float64	Numeric, Ratio-Scaled

residual_sugar	The amount of sugar remaining after fermentation, impacting the sweetness of the wine. Measured in grams per liter.	float64	Numeric, Ratio-Scaled
chlorides	The concentration of chloride salts in the wine, which can affect its taste and stability. Measured in grams per liter.	float64	Numeric, Ratio-Scaled
free_sulfur_dioxide	The amount of free sulfur dioxide present, which acts as an antimicrobial and antioxidant in wine. Measured in parts per million.	float64	Numeric, Ratio-Scaled
total_sulfur_dioxide	The total amount of sulfur dioxide, combining both free and bound forms, which preserves the wine. Measured in parts per million.	float64	Numeric, Ratio-Scaled
density	The density of the wine, which can be an indicator of sugar and alcohol content. Measured in grams per cubic centimeter.	float64	Numeric, Ratio-Scaled
ph	The pH level of the wine, indicating its acidity. Lower values correspond to higher acidity	float64	Numeric, Ratio-Scaled
sulphates	The concentration of potassium sulfate in the wine, which contributes to its preservation and taste. Measured in grams per liter.	float64	Numeric, Ratio-Scaled
alcohol	The alcohol content of the wine, which affects its flavor and preservation. Measured as a percentage by volume	float64	Numeric, Ratio-Scaled
quality	The quality rating of the wine, a scale from 0 (very bad) to 10 (excellent).	Int64	Ordinal
type	The type of wine, indicating whether it is red or white.	object	Nominal

Data Integration

Load dataset and make data Integreation process

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

load winequality-red data

[4]:
```
# Load the red wine dataset
red_wine = pd.read_csv('winequality-red.csv', sep=';')

# new column type/class of wine
red_wine['type'] = 'red'

red_wine.head()
```

[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	type
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red

load winequality-white data

[6]:
```
# Load the white wine dataset
white_wine = pd.read_csv('winequality-white.csv', sep=';')

# new column type/class of wine
white_wine['type'] = 'white'

white_wine.head()
```

[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	type
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	white
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	white
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	white
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	white
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	white

data Integration "combine"

[8]:
```
# Combine the datasets
dataset = pd.concat([red_wine, white_wine], ignore_index=True)
dataset
```

[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	type
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	white
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	white
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	white
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	white
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	white

6497 rows × 13 columns

Data Exploration

Important step in the data analysis process. It involves examining the dataset to understand its structure, identify patterns, detect outliers.

1. Describing columns in dataset , take overview about numerical column (mean , min , std, quarters , count of each whice tell us no missing value in any attribute and max which indicate some outliers and so on “statistics”

Some Exploration

```
[10]: dataset.describe()  # describing columns in dataset ..  provides a summary of each feature, including mean, standard deviation, and quartiles.
```

[10]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000
mean	7.215307	0.339666	0.318633	5.443235	0.056034	30.525319	115.744574	0.994697	3.218501	0.531268	10.491801	5.818378
std	1.296434	0.164636	0.145318	4.757804	0.035034	17.749400	56.521855	0.002999	0.160787	0.148806	1.192712	0.873255
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.000000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

2. Dataset Size

```
[11]: dataset.shape #size of dataset
[11]: (6497, 13)
```

3. Information about dataset , columns name , type , count “summary of dataset”

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

Columns name has space so uniform naming

```
[13]: # rename some coulmn to make simple in use
dataset.rename(columns={
    'fixed acidity': 'fixed_acidity',
    'volatile acidity': 'volatile_acidity',
    'citric acid': 'citric_acid',
    'residual sugar': 'residual_sugar',
    'free sulfur dioxide': 'free_sulfur_dioxide',
    'total sulfur dioxide': 'total_sulfur_dioxide',
    'pH': 'ph',
}, inplace=True)

dataset.columns
```

```
[13]: Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'ph', 'sulphates', 'alcohol', 'quality', 'type'],
      dtype='object')
```

4. Values count in each class/quality “Distribution of data in quality”

```
[14]: # Check the distribution of the 'quality' column
dataset['quality'].value_counts()
```

```
[14]: quality
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: count, dtype: int64
```

5. Checking for Missing Values: Identify any missing values in the dataset to ensure data integrity *“Ensures that there are no incomplete data entries that could affect the analysis.”*

```
[16]: dataset.isnull().sum() #sum of null values
# can handel using dataset.fillna(dataset.mean(), inplace=True)
```

```
[16]: 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
type                    0
dtype: int64
```

6. Checking for Duplication: *Detects repetitive rows that could bias the model.*

```
[18]: dataset.duplicated().sum() # sum of duplicated values
```

```
[18]: 1177
```

```
[19]: dataset[dataset.duplicated()] # display the duplicated rows
```

[19]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	ph	sulphates	alcohol	quality	type
4	7.4	0.700	0.00	1.90	0.076	11.0	34.0	0.99780	3.51	0.56	9.400000	5	red
11	7.5	0.500	0.36	6.10	0.071	17.0	102.0	0.99780	3.35	0.80	10.500000	5	red
27	7.9	0.430	0.21	1.60	0.106	10.0	37.0	0.99660	3.17	0.91	9.500000	5	red
40	7.3	0.450	0.36	5.90	0.074	12.0	87.0	0.99780	3.33	0.83	10.500000	5	red
65	7.2	0.725	0.05	4.65	0.086	4.0	11.0	0.99620	3.41	0.39	10.900000	5	red
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6427	6.4	0.230	0.35	10.30	0.042	54.0	140.0	0.99670	3.23	0.47	9.200000	5	white
6449	7.0	0.360	0.35	2.50	0.048	67.0	161.0	0.99146	3.05	0.56	11.100000	6	white
6450	6.4	0.330	0.44	8.90	0.055	52.0	164.0	0.99488	3.10	0.48	9.600000	5	white
6455	7.1	0.230	0.39	13.70	0.058	26.0	172.0	0.99755	2.90	0.46	9.000000	6	white
6479	6.6	0.340	0.40	8.10	0.046	68.0	170.0	0.99494	3.15	0.50	9.533333	6	white

1177 rows × 13 columns

[20]:
```python
# shape (6497, 13) index 0 for rows , index 1 for coulmn
# Calculate the percentage of duplicated rows
percent_duplicated = (dataset.duplicated().sum() / dataset.shape[0]) * 100

print(f'The percentage of duplicated rows is: {percent_duplicated:.2f}%')
```

```
The percentage of duplicated rows is: 18.12%
```

Data Cleaning

Important step in data preprocessing, ensuring that the dataset is free from errors, inconsistencies, and irrelevant data. This step involves handling missing values, detecting and removing duplicates, and ensuring data integrity.

1. Handling missing values : No missing values

[16]:
```python
dataset.isnull().sum() #sum of null values
# can handel using dataset.fillna(dataset.mean(), inplace=True)
```

[16]:
```
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
type                    0
dtype: int64
```

2. Handling Duplicate Values: Removing duplicate rows helps maintain the quality and integrity of the dataset by *ensuring each entry contributes equally to the analysis*.

Handle Duplicates

Removing duplicates help in prevent biases in model training and evaluation "So Improved Model Performance"

[22]:
```python
dataset = dataset.drop_duplicates(keep='first')  # remove duplicates but keeping the first occurrence of row
dataset
```

[22]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	ph	sulphates	alcohol	quality	type
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	red
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.99780	3.51	0.56	9.4	5	red
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	white
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	white
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	white
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	white
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	white

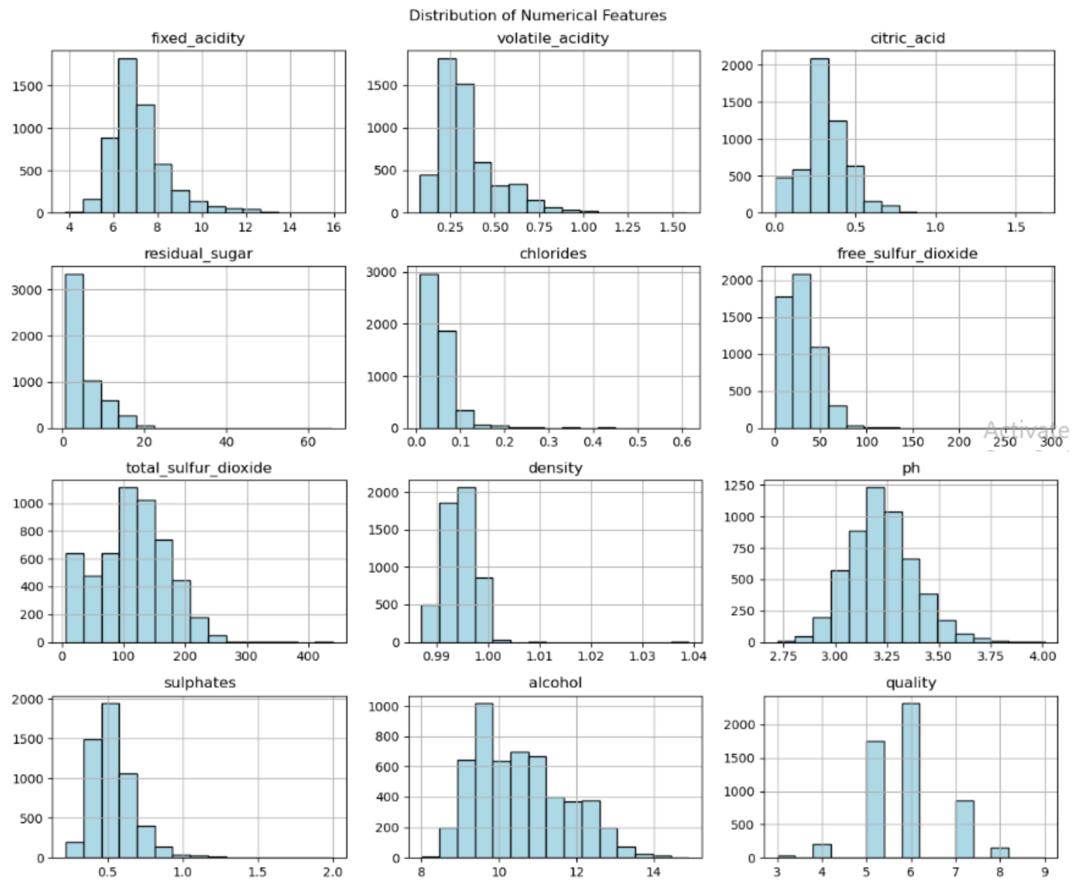
5320 rows × 13 columns

Data Visualization

Powerful step for understanding the patterns and relationships in a dataset.

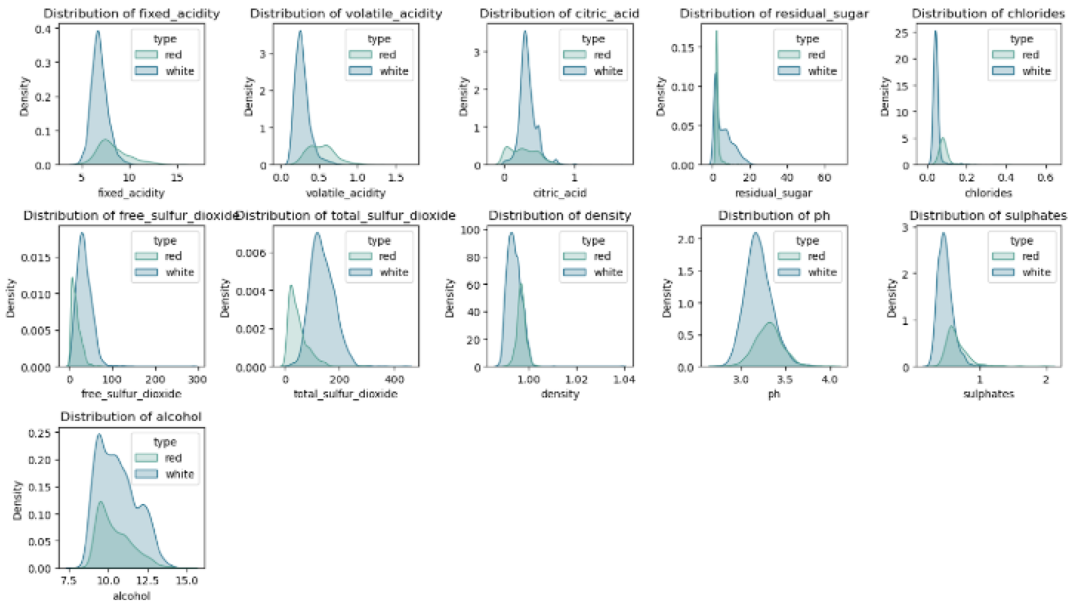
1. Distribution of Numerical Features: understand the spread and central tendency of the data.

[25]:
```python
# Plot histograms for numerical features
numeric_columns = dataset.select_dtypes(include=['float64', 'int64']).columns
dataset[numeric_columns].hist(bins=15, figsize=(12, 10), color='lightblue', edgecolor='black')
plt.suptitle('Distribution of Numerical Features')
plt.tight_layout()
plt.show()
```



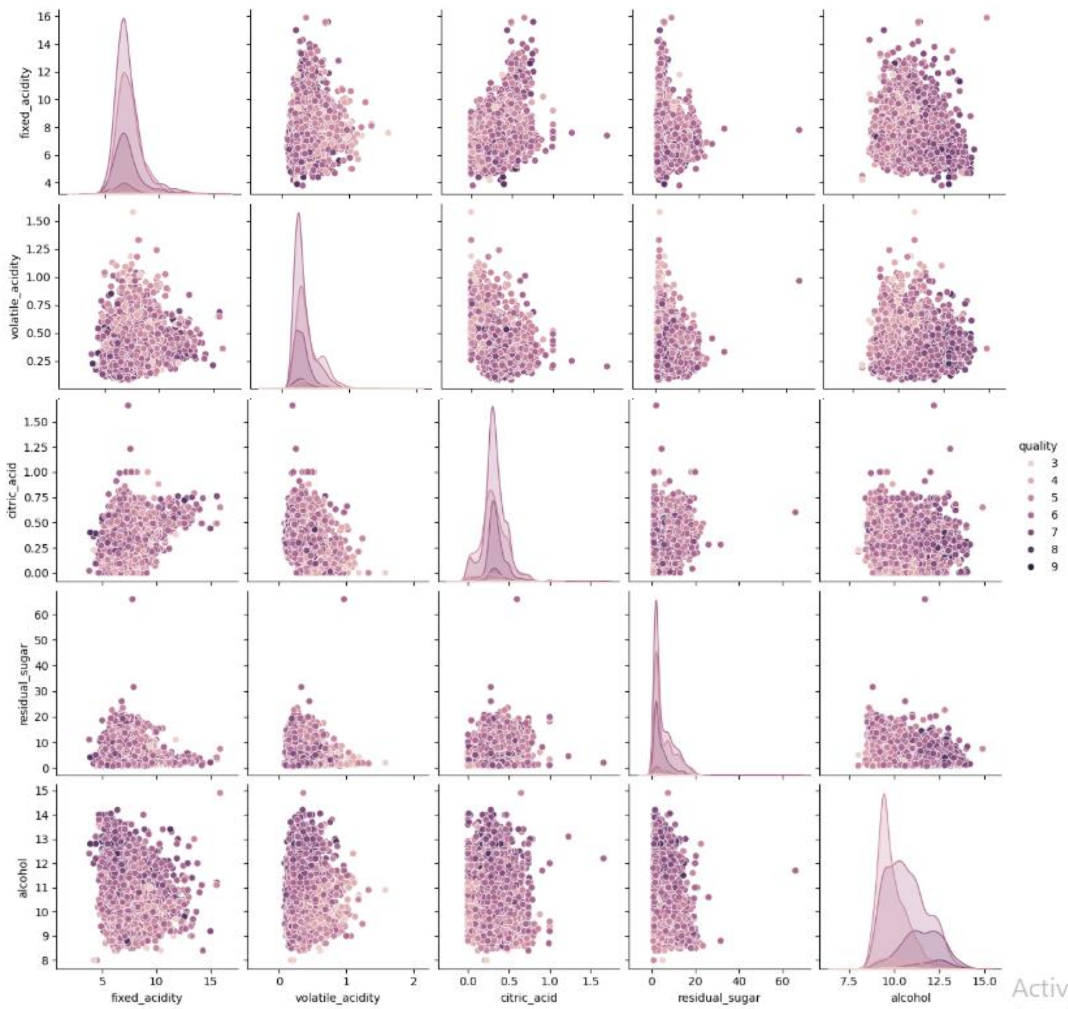
2. Different between red/white wine “comparing features”

[26]:
```python
# Comparison of features between red and white wines
plt.figure(figsize=(14, 8))
# kdeplot to all columns in the dataset except the last two columns. index (i) and column name (column).
for i, column in enumerate(dataset.columns[:-2], 1): # expet quality and type
    plt.subplot(3, 5, i) # display in 3 rows and 5 columns
    sns.kdeplot(data=dataset, x=column, hue='type', fill=True, palette='crest')
    plt.title(f'Distribution of {column}')
plt.tight_layout() #spacing between subplots
plt.show()
```



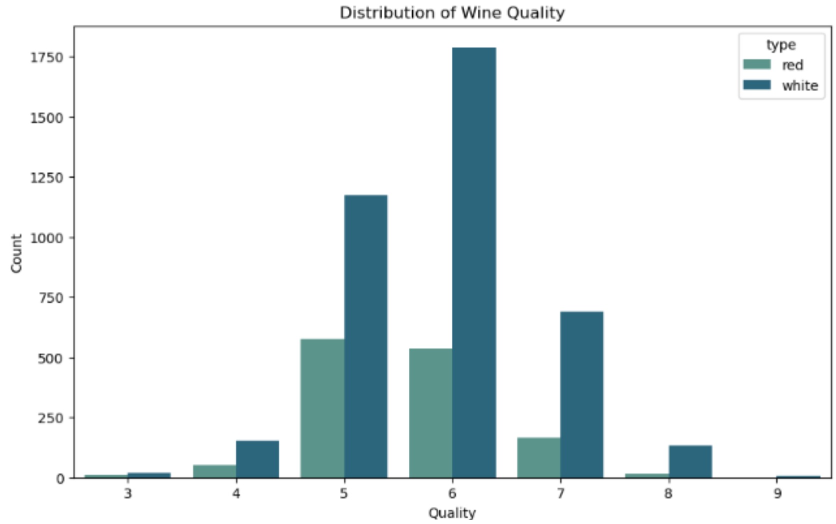
3. Pair plot: visualization of pairwise relationships between features, help *identify patterns and potential correlations*.

```
[28]: # Pair plot some features
selected_features = ['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar', 'alcohol', 'quality']
sns.pairplot(dataset[selected_features], hue='quality')
plt.show()
```

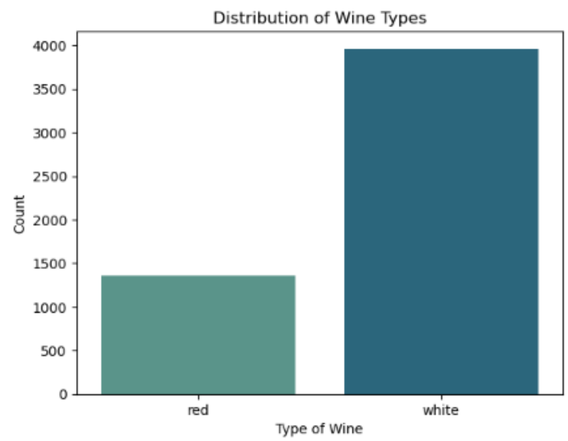


4. Distribution of wine quality *"how quality is spread across different levels, for understanding the dataset's balance and give insight sufficient samples for each quality level, which is important for training"*, type *"Understanding the balance between red and white wines helps to ensure that the dataset represents both types adequately"*

```
[24]: # Create a figure with a specific size 10 * 6
plt.figure(figsize=(10, 6))

# Create a count plot for the 'quality' column, with a hue based on the 'type' column
sns.countplot(data=dataset, x='quality', hue='type', palette="crest")

# title of the plot
plt.title('Distribution of Wine Quality')
#label for x axis
plt.xlabel('Quality')
#label for y axis
plt.ylabel('Count')

plt.show() # shows the distribution of wine rows for each quality rating.
# so understand how many wines fall into each quality category
```



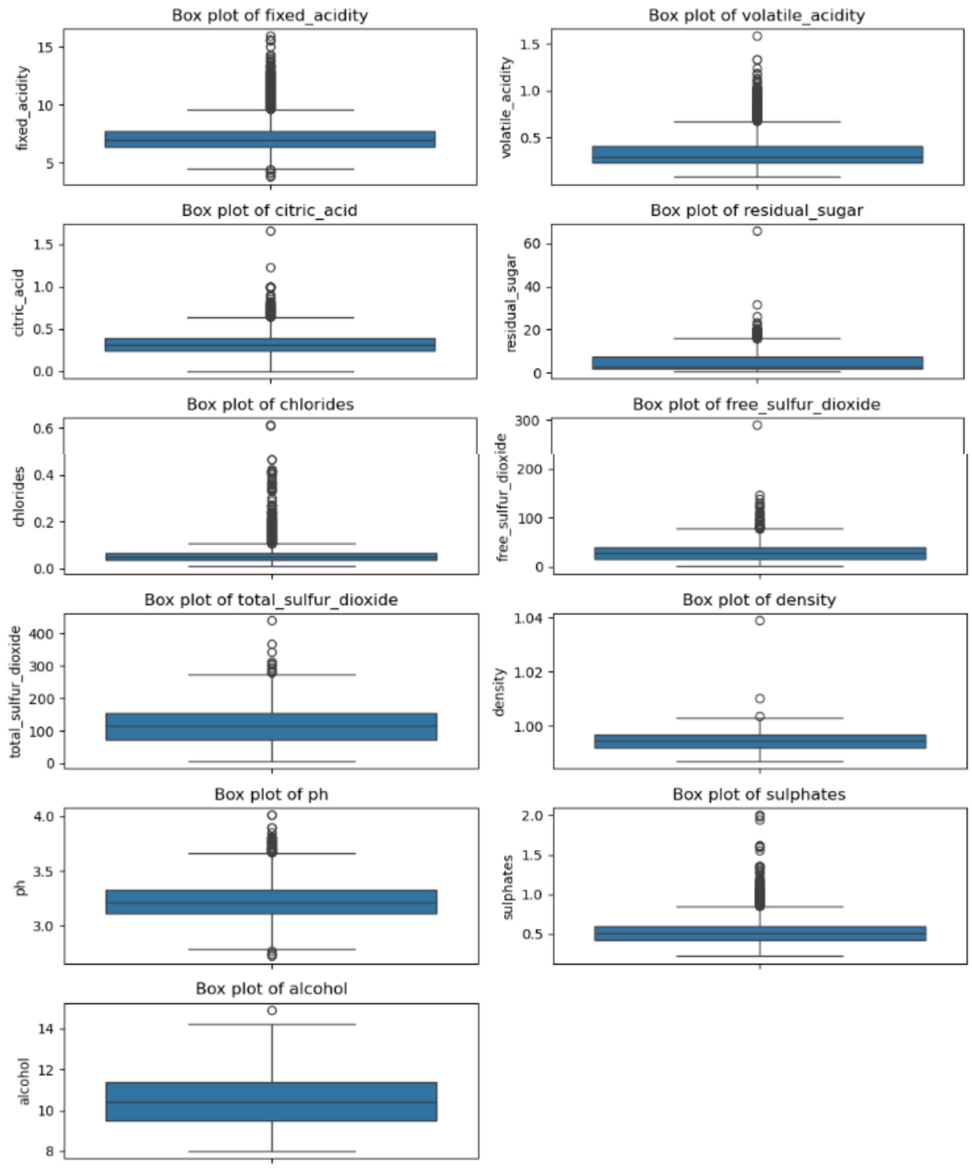
[27]:
```
# Count plot for categorical variable 'type'
sns.countplot(x='type', data=dataset, hue='type', palette="crest")
plt.title('Distribution of Wine Types')
plt.xlabel('Type of Wine')
plt.ylabel('Count')
plt.show()
```



Activ

5. Box plot to each future: tool for visualizing the distribution of data and identifying outliers *"outliers can significantly impact the performance of models"*

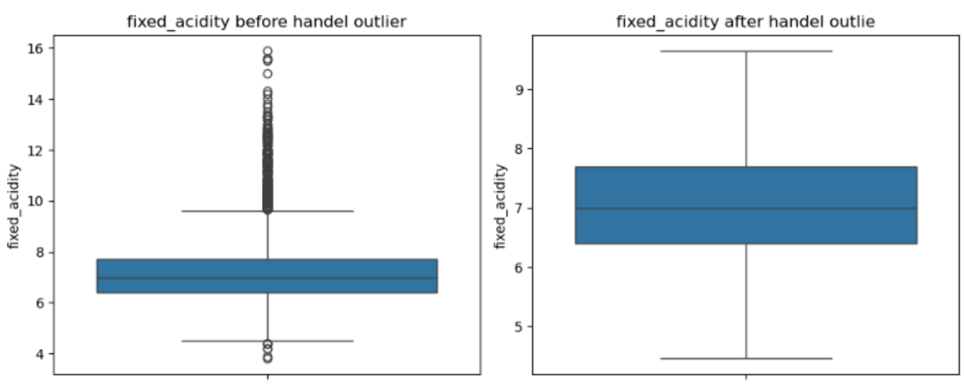
[29]:
```
plt.figure(figsize=(10,12))
# Box blot all columns in the dataset except the last two columns ('quality' and 'type')
for i, column in enumerate(dataset.columns[:-2], 1):
    plt.subplot(6,2, i) # Display in 4 rows and 3columns
    sns.boxplot(data=dataset, y=column)
    plt.title(f'Box plot of {column}') # title to each box plot
plt.tight_layout() # Adjust spacing between subplots
plt.show()
```



Data cleaning again to Handleing outliers:

1. **Removing Outliers :** clean the dataset by removing data points that are significantly different from the majority.
2. **Capping Outliers:** Determine the lower and upper bounds using 1.5 times the IQR. Cap values that fall outside these bounds to the respective lower or upper bound values. *" ensures that outliers are adjusted without completely removing them from the dataset "*

```python
[31]: # before handling outliers
fig, axs = plt.subplots(1, 2, figsize=(10,4))

sns.boxplot(data=dataset, y='fixed_acidity', ax=axs[0])
axs[0].set_title('fixed_acidity before handel outlier')

# Handle fixed_acidity column
# Step 1: Remove 2 maximum outliers -- delete max outliers
indices = dataset['fixed_acidity'].nlargest(2).index
dataset = dataset.drop(indices)

# Step 2: Calculate IQR and cap other outliers

# Calculate the IQR for restecg - interquartile range measure of  spread of the data
Q1 = dataset['fixed_acidity'].quantile(0.25)  # Q1 represents the value below which 25% of the data points fall.
Q3 = dataset['fixed_acidity'].quantile(0.75)  # Q3 represents the value below which 75% of the data points fall.
IQR = Q3 - Q1

# Lower and upper bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Cap values outside the bounds
dataset.loc[dataset['fixed_acidity'] < lower_bound, 'fixed_acidity'] = lower_bound
dataset.loc[dataset['fixed_acidity'] > upper_bound, 'fixed_acidity'] = upper_bound

# after handling outliers
sns.boxplot(data=dataset, y='fixed_acidity', ax=axs[1])
axs[1].set_title('fixed_acidity after handel outlie')

plt.tight_layout()
plt.show()
```



Repeat for all attributes has outlier

Data Encoding & Normalization

Important preprocessing step that involves scaling numerical data to a common range. In this past we ensure that the dataset is properly encoded, formatted, and normalized, making it suitable for effective machine learning model training and evaluation.

1. Encoding Categorical Variables: Transforming the type column from categorical labels to numerical values using Label Encoder. *"Some Machine learning algorithms require numerical inputs, so categorical variables must be converted into a numerical format."*

```python
[44]: # Encode the 'type' column
encoder = LabelEncoder()
dataset['type'] = encoder.fit_transform(dataset['type'])

# Verify the data types
dataset.dtypes
```

```
[44]: 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
type                      int32
dtype: object
```

converts labels (e.g., 'red', 'white') into numerical values (e.g., 0, 1), making the data suitable for model training

2. Normalizing the Features: put numerical features in similar scale, which is particularly important for algorithms that depend on the distance between data points, such as SVM.

Normalize Features

features contribute equally to the model so improve model performence

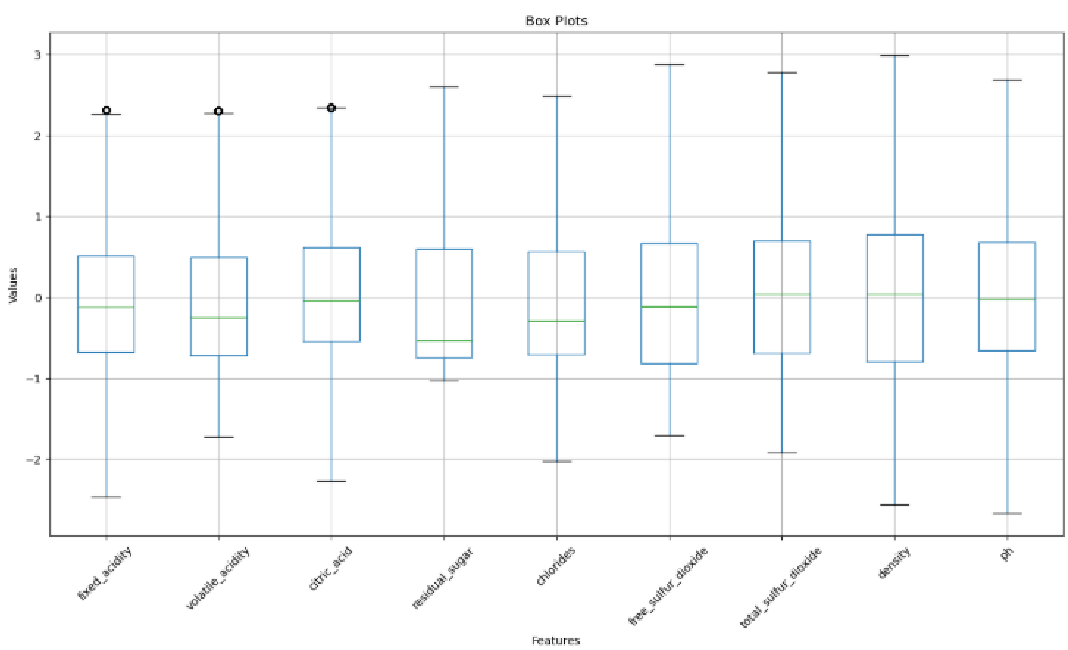
[46]:
```
# normalize the features in the dataset
scaler = StandardScaler()
# in some model like SVM depend on distance between point so performed better when the input features are on a similar scale
# so scaling/Normlization for all columns except the last 2 coulmns type and quality
dataset.iloc[:, :-2] = scaler.fit_transform(dataset.iloc[:, :-2])

dataset
```

[46]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	ph	sulphates	alcohol	quality	ty
0	0.240988	2.302867	-2.277003	-0.723823	1.023852	-1.115012	-1.420598	1.135756	1.819316	0.243606	-0.969819	5	
1	0.608494	2.302867	-2.277003	-0.559228	2.027696	-0.286986	-0.834159	0.789602	-0.152731	1.160903	-0.632021	5	
2	0.608494	2.302867	-1.988287	-0.629769	1.753920	-0.878433	-1.065181	0.858833	0.228956	0.931579	-0.632021	5	
3	2.308210	-0.385234	1.765018	-0.723823	0.978222	-0.760144	-0.958555	1.204987	-0.407189	0.396489	-0.632021	6	
5	0.240988	2.168462	-2.277003	-0.747336	0.978222	-0.996723	-1.313973	1.135756	1.819316	0.243606	-0.969819	5	
...	...	...	...	...	...	...	...	...	...	...	...	...	
6492	-0.861531	-0.855651	-0.183814	-0.794363	-0.664432	-0.346131	-0.389888	-1.169635	0.292570	-0.215043	0.550273	6	
6493	-0.494025	-0.116424	0.321439	0.710504	-0.299398	1.605645	0.960698	0.131907	-0.470803	-0.520809	-0.800920	5	
6494	-0.585901	-0.654044	-0.905603	-0.888418	-0.573173	0.008737	-0.052241	-0.685018	-1.488633	-0.520809	-0.969819	6	
6495	-1.504667	-0.318031	-0.111635	-0.911931	-1.440130	-0.582710	-0.070012	-2.017714	0.737871	-1.132340	1.901465	7	
6496	-1.045284	-0.855651	0.465797	-0.982472	-1.531388	-0.464420	-0.283262	-1.768483	0.228956	-1.590988	1.056970	6	

Show data after preprocessing steps

[47]:
```
# Select all numerical columns except the last two
columns_to_plot = dataset.select_dtypes(include=['float64']).iloc[:, :-2]

# Create a single boxplot for the selected columns
plt.figure(figsize=(16, 8))
columns_to_plot.boxplot()
plt.title('Box Plots')
plt.xticks(rotation=45)  #  better readability
plt.ylabel('Values')
plt.xlabel('Features')
plt.show()
```



Split Data for Training and Testing

Splitting the dataset into training and testing sets, this ensures that we can evaluate the performance of our models on **unseen data,** helping to avoid overfitting and ensuring that the models generalize well to new data.

Use train_test_split function to split the dataset into training and testing sets. Take 4 parameters:
1) ***features***: attributes -All columns except the target variable quality.

2) ***label***: to predict it - The quality column, which represents the wine quality ratings.”

3) ***Test_size***: Specifies that 30% of the data will be used for testing and 70% for training.

4) Random_state: Ensures reproducibility of the split by setting a random seed.

and output is:

1. `features_train`: The training set for the features, used to train the model.
2. `features_test`: The testing set for the features, used to evaluate the model.
3. `label_train`: The training set for the target labels, corresponding to the `features_train` data.
4. `label_test`: The testing set for the target labels, corresponding to the `features_test` data.

Split the Data

for training the model and evaluating its performance

```
[49]: from sklearn.model_selection import train_test_split as tts #for siplit data

# Split data into features"attribute leran model based on" and target "lable"
features = dataset.drop('quality', axis=1)
label = dataset['quality']


# Encode the target labels
label_encoded = encoder.fit_transform(label)

# Split data into training and testing sets
features_train, features_test, label_train, label_test = tts(features, label_encoded, test_size=0.3, random_state=42)
#random_state to get same results each time you run code 42 not fixed but commenly used "after reserach"
```

Classifiers' Description

To predict wine quality, we will use a different of machine learning classifiers.

- Support Vector Machine (SVM): is a supervised machine learning algorithm used for both classification and regression tasks. While it can be applied to regression problems, SVM is best suited for classification tasks. The primary objective of the SVM algorithm is to identify the optimal hyperplane in an N-dimensional space that can effectively separate data points into different classes in the feature space. The algorithm ensures that the margin between the closest points of different classes, known as support vectors, is maximized.

How work In Wine Dataset: Support Vector Machine (SVM) classifier works by finding the optimal hyperplane that best separates the wines into different quality classes. It looks at the features of each wine (e.g., acidity, sugar content, pH) and aims to maximize the margin between wines of different quality ratings, ensuring that wines with similar features are grouped together.

- **Import Classifier using SVC**

```
from sklearn.svm import SVC # SVM classifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score # needed function to use
```

- **Model Construction:** Initialize the classifier, trained using *fit method* take feature_train and label_train as parameters, and predicted using *predict method* take features_test as parameter using the SVM classifier.

```
#  model Construction Step -  train the SVM classifierc
svm = SVC(probability=True, random_state=42)  # SVM classifier with probability
svm.fit(features_train, label_train)  # train the classifier using the training data
predict = svm.predict(features_test)  # predict the labels for the test data
```

- **Performance Metrics**: Accuracy - Measure the proportion of correctly predicted labels- , F1-Score -Mean of precision and recall- , and ROC AUC - measuring the classifier's ability to distinguish between classes using the one-vs-rest strategy- scores provide a comprehensive evaluation of the classifier's performance on the test dataset.

```
# Model Usage Step - test the SVM classifier

# Calculate accuracy "compare actual and predicted labels"
# Accuracy = (TP + TN) / (TP + TN + FP + FN) or ALL
svm_accuracy = accuracy_score(label_test, predict)

# Calculate the F1-Score
# F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN)
# Averages the scores, weighted by the number of true instances for each class
svm_f1 = f1_score(label_test, predict, average='weighted')

# Calculate the ROC AUC, one-vs-one strategy
# ROC AUC: Area Under the Receiver Operating Characteristic Curve
# Computes the ROC AUC score for each class using the One-vs-Rest strategy.
# Averages the scores across all classes.
svm_roc_auc = roc_auc_score(label_test, svm.predict_proba(features_test), multi_class='ovr')
```

- **Display Results**

```
# results
print(f'Support Vector Machine (SVM) Classifier - Accuracy: {svm_accuracy * 100:.2f}%')
print(f'Support Vector Machine (SVM) Classifier - F1-Score: {svm_f1 * 100:.2f}%')
print(f'Support Vector Machine (SVM) Classifier - ROC/AUC: {svm_roc_auc * 100:.2f}%')
```

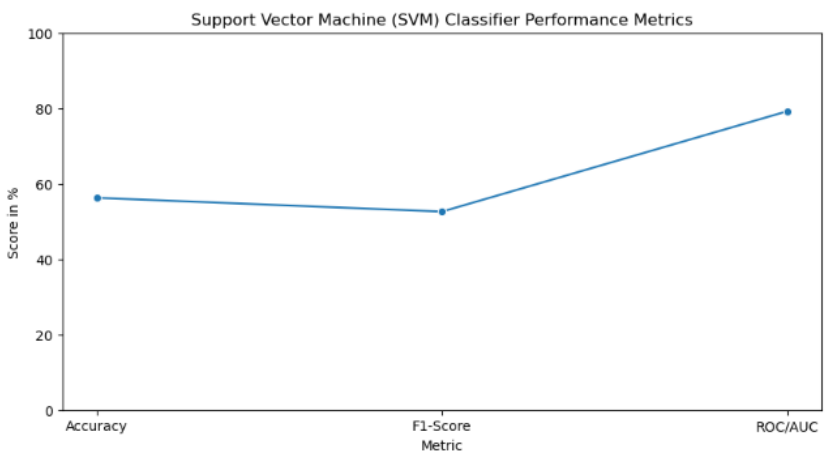
```
Support Vector Machine (SVM) Classifier - Accuracy: 56.25%
Support Vector Machine (SVM) Classifier - F1-Score: 52.59%
Support Vector Machine (SVM) Classifier - ROC/AUC: 79.19%
```

Support Vector Machine (SVM) Classifier

[51]:
```
from sklearn.svm import SVC # SVM classifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score # needed function to use

#  model Construction Step -  train the SVM classifierc
svm = SVC(probability=True, random_state=42)  # SVM classifier with probability
svm.fit(features_train, label_train)  # train the classifier using the training data
predict = svm.predict(features_test)  # predict the labels for the test data

# Model Usage Step - test the SVM classifier

# Calculate accuracy "compare actual and predicted labels"
# Accuracy = (TP + TN) / (TP + TN + FP + FN) or ALL
svm_accuracy = accuracy_score(label_test, predict)

# Calculate the F1-Score
# F1-Score = 2 * (Precision * Recall) / (Precision + Recall)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN)
# Averages the scores, weighted by the number of true instances for each class
svm_f1 = f1_score(label_test, predict, average='weighted')

# Calculate the ROC AUC, one-vs-one strategy
# ROC AUC: Area Under the Receiver Operating Characteristic Curve
# Computes the ROC AUC score for each class using the One-vs-Rest strategy.
# Averages the scores across all classes.
svm_roc_auc = roc_auc_score(label_test, svm.predict_proba(features_test), multi_class='ovr')

# results
print(f'Support Vector Machine (SVM) Classifier - Accuracy: {svm_accuracy * 100:.2f}%')
print(f'Support Vector Machine (SVM) Classifier - F1-Score: {svm_f1 * 100:.2f}%')
print(f'Support Vector Machine (SVM) Classifier - ROC/AUC: {svm_roc_auc * 100:.2f}%')
```

```
Support Vector Machine (SVM) Classifier - Accuracy: 56.25%
Support Vector Machine (SVM) Classifier - F1-Score: 52.59%
Support Vector Machine (SVM) Classifier - ROC/AUC: 79.19%
```


[52]:
```
# Create a line plot for SVM metrics
plt.figure(figsize=(10, 5))
sns.lineplot(x=['Accuracy', 'F1-Score', 'ROC/AUC'], y=[svm_accuracy * 100, svm_f1 * 100, svm_roc_auc * 100], marker='o')
plt.title('Support Vector Machine (SVM) Classifier Performance Metrics')
plt.xlabel('Metric')
plt.ylabel('Score in %')
plt.ylim(0, 100)
plt.show()
```



- XGBoost (Extreme Gradient Boosting): is an advanced machine learning algorithm based on the gradient boosting framework. It is designed for both classification and regression tasks and is known for its speed, performance, and accuracy. XGBoost builds an ensemble of weak decision trees, improving them iteratively by minimizing the loss function, and includes regularization techniques to prevent overfitting.

How it works in the Wine Dataset: The XGBoost classifier evaluates the features of each wine (e.g., acidity, sugar content, pH) and builds a series of decision trees. Each tree attempts to correct the errors made by the previous trees, resulting in a robust model that can accurately predict the wine quality ratings based on the combined input features.

- **Import Classifier using XGBClassifier**

XGBoost Classifier

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\hp\anaconda3\lib\site-packages (2.1.3)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.13.1)
```

```
import xgboost
print(xgboost.__version__)
```

```
2.1.3
```

```
from xgboost import XGBClassifier
```

- **Model Construction:** Initialized the classifier, trained using *fit method* take feature_train and label_train as parameters, and predicted using *predict method* take features_test as parameter using the XGBClassifier classifier.

```
#  model Construction Step -  train the xGboost Classifier
xgb = XGBClassifier(random_state=42, enable_categorical=True)
xgb.fit(features_train, label_train)
predict = xgb.predict(features_test)
```

- **Performance Metrics**: Accuracy, F1-Score, and ROC AUC scores provide a comprehensive evaluation of the classifier's performance on the test dataset.

```
# model Usage Step - test the XGboost Classifier
xgboost_accuracy = accuracy_score(label_test, predict)
xgboost_f1 = f1_score(label_test, predict, average='weighted')
xgboost_roc_auc = roc_auc_score(label_test, xgb.predict_proba(features_test), multi_class='ovr')
```

- **Display Results**

```
# results
print(f'XGBoost Classifier - Accuracy: {xgboost_accuracy * 100:.2f}%')
print(f'XGBoost Classifier - F1-Score: {xgboost_f1 * 100:.2f}%')
print(f'XGBoost Classifier - ROC/AUC: {xgboost_roc_auc * 100:.2f}%')
```

```
XGBoost Classifier - Accuracy: 54.05%
XGBoost Classifier - F1-Score: 52.51%
XGBoost Classifier - ROC/AUC: 73.88%
```

- Random Forest Classifier: is an ensemble learning method used for both classification and regression tasks. It constructs multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. This method reduces overfitting by averaging the results of multiple trees, making it robust and reliable.

How it works in the Wine Dataset: The Random Forest classifier evaluates the features of each wine (e.g., acidity, sugar content, pH) and builds multiple decision trees. Each tree is trained on a random subset of the data, and the final prediction is made by averaging the results (for regression) or taking the majority vote (for classification) of these trees. This ensemble approach enhances the model's accuracy and robustness.

1. **Import Classifier using RandomForestClassifier**

```
from sklearn.ensemble import RandomForestClassifier
```

2. **Model Construction:** Initialize the classifier, trained using *fit method* take feature_train and label_train as parameters, and predicted using *predict method* take features_test as parameter using the SVM classifier.

```
#  model Construction Step -  train the Random Forest classifier
rf = RandomForestClassifier(n_estimators=50, random_state=42)
rf.fit(features_train, label_train)
predict = rf.predict(features_test)
```

3. **Performance Metrics**: Accuracy - Measure the proportion of correctly predicted labels- , F1-Score -Mean of precision and recall- , and ROC AUC - measuring the classifier's ability to distinguish between classes using the one-vs-rest strategy- scores provide a comprehensive evaluation of the classifier's performance on the test dataset.

```
# model Usage Step - test the Random Forest classifier
rforest_accuracy = accuracy_score(label_test, predict)
rforest_f1 = f1_score(label_test, predict, average='weighted')
rforest_roc_auc= roc_auc_score(label_test, rf.predict_proba(features_test),multi_class='ovr')
```

4. **Display Results**

```
# results
print(f'Random Forest Classifier - Accuracy: {rforest_accuracy * 100:.2f}%')
print(f'Random Forest Classifier - F1-Score: {rforest_f1 * 100:.2f}%')
print(f'Random Forest Classifier - ROC/AUC: {rforest_roc_auc * 100:.2f}%')
```

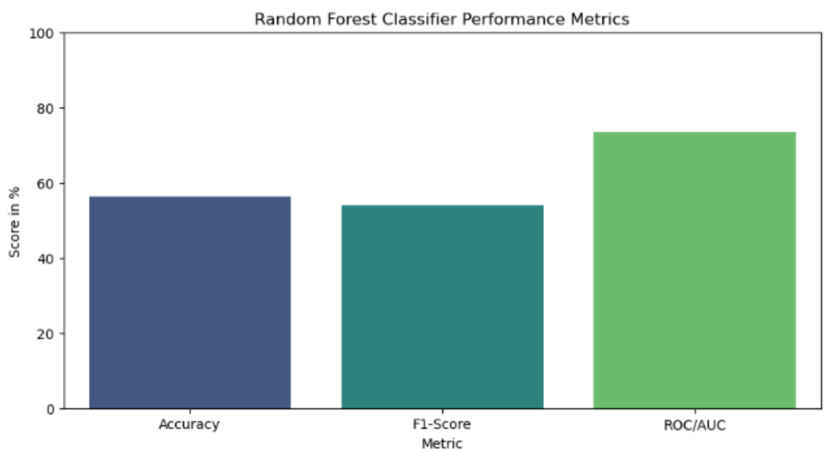
```
Random Forest Classifier - Accuracy: 56.18%
Random Forest Classifier - F1-Score: 53.96%
Random Forest Classifier - ROC/AUC: 73.44%
```

Random Forest Classifier (with 50 estimators)

[58]:
```python
from sklearn.ensemble import RandomForestClassifier

#  model Construction Step -  train the Random Forest classifier
rf = RandomForestClassifier(n_estimators=50, random_state=42)
rf.fit(features_train, label_train)
predict = rf.predict(features_test)

# model Usage Step - test the Random Forest classifier
rforest_accuracy = accuracy_score(label_test, predict)
rforest_f1 = f1_score(label_test, predict, average='weighted')
rforest_roc_auc= roc_auc_score(label_test, rf.predict_proba(features_test),multi_class='ovr')

# results
print(f'Random Forest Classifier - Accuracy: {rforest_accuracy * 100:.2f}%')
print(f'Random Forest Classifier - F1-Score: {rforest_f1 * 100:.2f}%')
print(f'Random Forest Classifier - ROC/AUC: {rforest_roc_auc * 100:.2f}%')
```

```
Random Forest Classifier - Accuracy: 56.18%
Random Forest Classifier - F1-Score: 53.96%
Random Forest Classifier - ROC/AUC: 73.44%
```

[59]:
```python
metrics = ['Accuracy', 'F1-Score', 'ROC/AUC']

# Create a bar plot for Random Forest Classifier metrics
plt.figure(figsize=(10, 5))
sns.barplot(x=metrics, y= [rforest_accuracy * 100, rforest_f1 * 100, rforest_roc_auc * 100], hue=metrics, palette='viridis')
plt.title('Random Forest Classifier Performance Metrics')
plt.xlabel('Metric')
plt.ylabel('Score in %')
plt.ylim(0, 100)
plt.show()
```



- Multilayer Perceptron (MLP): type of artificial neural network that consists of multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer. MLPs are capable of learning complex non-linear relationships in the data by adjusting the weights of the connections between neurons through the process of backpropagation.

How it works in the Wine Dataset: The MLP classifier processes the features of each wine (e.g., acidity, sugar content, pH) through multiple layers of neurons. Each layer applies a transformation using weights and activation functions, allowing the network to learn intricate patterns and relationships between the features. The final output layer predicts the wine quality rating based on the learned patterns.

Multilayer Perceptron (MLP) Classifier

[61]:
```python
from sklearn.neural_network import MLPClassifier

#  model Construction Step -  train the MLP classifier with 22 hidden neurons in one layer
mlp = MLPClassifier(hidden_layer_sizes=(22,), max_iter=1000, random_state=42)
# max_iter=1000, algorithm will run for up to 1000 iterations
#hidden_layer_sizes=(22,) means that the neural network has one hidden layer with 22 neurons
# using a single hidden layer with 22 neurons make a balance between simplicity and complexity
# making it a suitable choice for many machine learning tasks

mlp.fit(features_train, label_train)
predict = mlp.predict(features_test)
# model Usage Step - test the MLP classifier
mlp_accuracy = accuracy_score(label_test, predict)
mlp_f1 = f1_score(label_test, predict , average='weighted')
mlp_roc_auc = roc_auc_score(label_test, mlp.predict_proba(features_test), multi_class='ovr')

# results
print(f'Multilayer Perceptron (MLP) Classifie - Accuracy: {mlp_accuracy * 100:.2f}%')
print(f'Multilayer Perceptron (MLP) Classifie - F1-Score: {mlp_f1 * 100:.2f}%')
print(f'Multilayer Perceptron (MLP) Classifie - ROC/AUC: {mlp_roc_auc * 100:.2f}%')
```

```
Multilayer Perceptron (MLP) Classifie - Accuracy: 54.99%
Multilayer Perceptron (MLP) Classifie - F1-Score: 53.21%
Multilayer Perceptron (MLP) Classifie - ROC/AUC: 75.78%
```

Results and Discussion

We will evaluate the performance of each classifier Then, we will compare the results visually to summarize the performance of the classifiers.

1. Support Vector Machine (SVM)

- Support Vector Machine (SVM) Classifier - Accuracy: 56.25%
- Support Vector Machine (SVM) Classifier - F1-Score: 52.59%
- Support Vector Machine (SVM) Classifier - ROC/AUC: 79.19%

The SVM classifier achieved a 56.25 **accuracy**, indicating it correctly predicted the wine quality in around 56.25% of cases. The **F1-Score** is slightly lower than the accuracy, suggesting that the classifier might have some imbalance between precision and recall. The ROC/AUC score is relatively high, indicating that the classifier has a good ability to distinguish between different quality classes.

2. XGBoost

- XGBoost Classifier - Accuracy: 54.05%
- XGBoost Classifier - F1-Score: 52.51%
- XGBoost Classifier - ROC/AUC: 73.88%

XGBoost achieved an accuracy of 54.05%, which is slightly lower compared to SVM and Random Forest. The **F1-Score** is close to the accuracy, indicating a balanced model in terms of precision and recall. The ROC/AUC score indicates a decent ability to distinguish between quality classes but is lower compared to SVM.

3. Random Forest

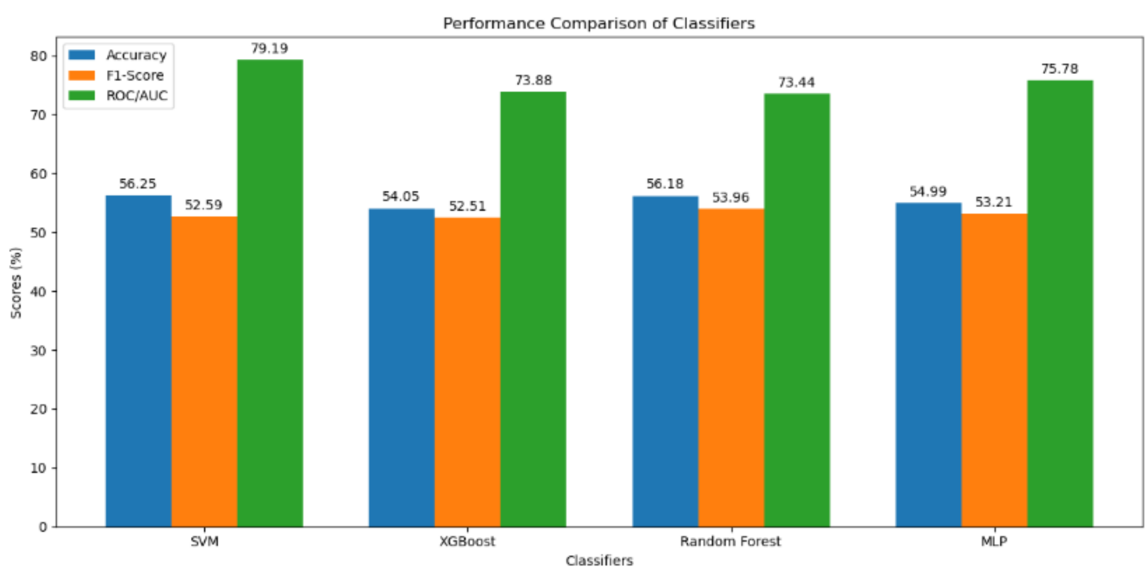
- Random Forest Classifier - Accuracy: 56.18%
- Random Forest Classifier - F1-Score: 53.96%
- Random Forest Classifier - ROC/AUC: 73.44%

Random Forest achieved a solid accuracy of 56.18%, performing similarly to SVM.The F1-Score is slightly higher than SVM, suggesting a better balance between precision and recall. The ROC/AUC score is similar to XGBoost, indicating a reasonable performance in distinguishing between quality classes.

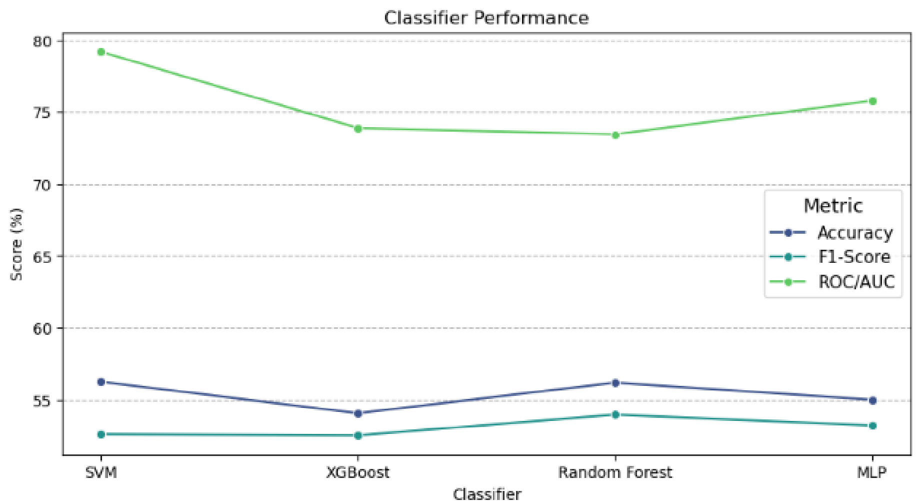
4. Multilayer Perceptron (MLP)

- Multilayer Perceptron (MLP) Classifie - Accuracy: 54.99%
- Multilayer Perceptron (MLP) Classifie - F1-Score: 53.21%
- Multilayer Perceptron (MLP) Classifie - ROC/AUC: 75.78%

MLP achieved an accuracy of 54.99%, indicating effective performance in predicting wine quality. The F1-Score is slightly lower than the accuracy, suggesting the model is well-balanced. The ROC/AUC score indicates that MLP has a strong ability to distinguish between quality classes, higher than Random Forest and XGBoost.



[65]:
```
# Plot the results
plt.figure(figsize=(10, 5))
sns.lineplot(x='Classifier', y='Score', hue='Metric', data=results_melted, marker='o', palette='viridis')
plt.title('Classifier Performance')
plt.xlabel('Classifier')
plt.ylabel('Score (%)')
plt.legend(title='Metric', title_fontsize='13', fontsize='11')
plt.grid(axis='y', linestyle='--')
plt.show()
```



Conclusion

In this report we explored different machine learning techniques to predict wine quality based on its features/attributes like Fixed Acidity, Volatile Acidity, Citric Acid, residual_sugar. Our goal focus on train and evaluate four different classifiers: Support Vector Machine (SVM), XGBoost, Random Forest, and Multilayer Perceptron (MLP). We Evaluate each classifier based on key performance metrics, including accuracy, F1-Score, and ROC/AUC scores.

Support Vector Machine (SVM) achieved moderate performance, with an accuracy of 56.25%, an F1-Score of 52.59%, and a high ROC/AUC score of 79.19%. XGBoost demonstrated reasonable performance, with an accuracy of 54.05%, an F1-Score of 52.51%, and an ROC/AUC score of 73.88%. Random Forest provided robust results, achieving an accuracy of 56.18%, an F1-Score of 53.96%, and an ROC/AUC score of 73.44%. Multilayer Perceptron (MLP) showed effective performance with an accuracy of 54.99%, an F1-Score of 53.21%, and a strong ROC/AUC score of 75.78%.

Overall, each classifier show its strengths in predicting wine quality, with SVM and Random Forest providing the best accuracy and balanced F1-Scores. The visual comparison of performance metrics highlighted these differences, so help us in selecting the most appropriate model for this classification task. By these machine learning techniques, we can make predictions about wine quality, enhancing quality control and decision-making in the wine industry.