A Predictive Analysis Report on

Wine Quality Detection

Submitted to Manipal University Jaipur

Towards the partial fulfilment for the Award of the Degree of

BACHELOR OF TECHNOLOGY

In Computer Science and Engineering

2023-2027

By

Suhani Gupta (23FE10CSE00215)

Amisha Bhatia (23FE10CSE00496)

Under the Guidance of

____

Signature of Supervisor

Department of Computer Science and Engineering

School of Computer Science and Engineering

Manipal University Jaipur

Contents

1) Introduction

Assessing wine quality traditionally relies on trained sensory experts who evaluate taste, aroma, and appearance. However, sensory evaluation is time-consuming, subjective, and inconsistent across tasters. The increasing availability of food chemistry data allows wine quality to be predicted using machine learning techniques, which provide automated, fast, and repeatable evaluation based on physicochemical properties.

In this study, we apply supervised machine learning to classify red wine samples as either:

- **High Quality (1)** → if the original wine quality score ≥ 7
- **Low/Medium Quality (0)** → otherwise

Thus, the task is a **binary classification** problem.

The primary objectives of this work are:

1. To explore and analyze the wine dataset and understand the relationships between its features.
2. To train multiple classification algorithms and compare their performance.
3. To identify the most suitable model for predicting wine quality.

2) Model Selection

Three machine learning models were selected, representing different learning paradigms:

Model	Type	Reason for Selection
Logistic Regression	Linear Model	Serves as a strong and interpretable baseline classifier.
Decision Tree Classifier	Non-Linear Tree Model	Captures hierarchical decision rules and non-linear feature interactions.
Random Forest Classifier	Ensemble Learning Model	Reduces overfitting and improves accuracy by combining multiple trees.

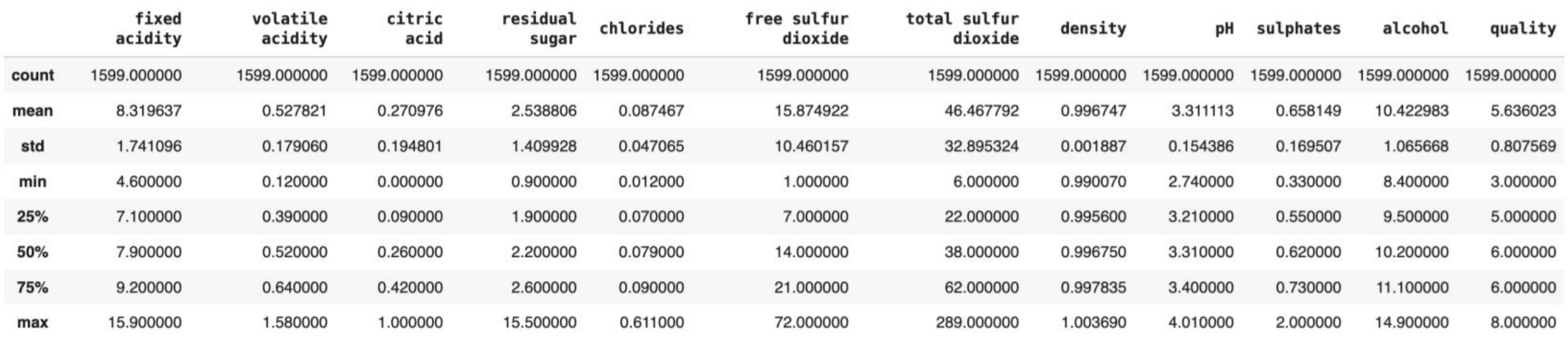
These models allow progressive improvement from baseline (Logistic Regression) to progressively more flexible and robust models.

3) Methodology

This study focuses on predicting wine quality using three supervised Machine Learning models: **Logistic Regression**, **Decision Tree Classifier**, and **Random Forest Classifier**. The methodology involves structured dataset inspection, preprocessing steps followed by model training and evaluation.

3.1 Data Description

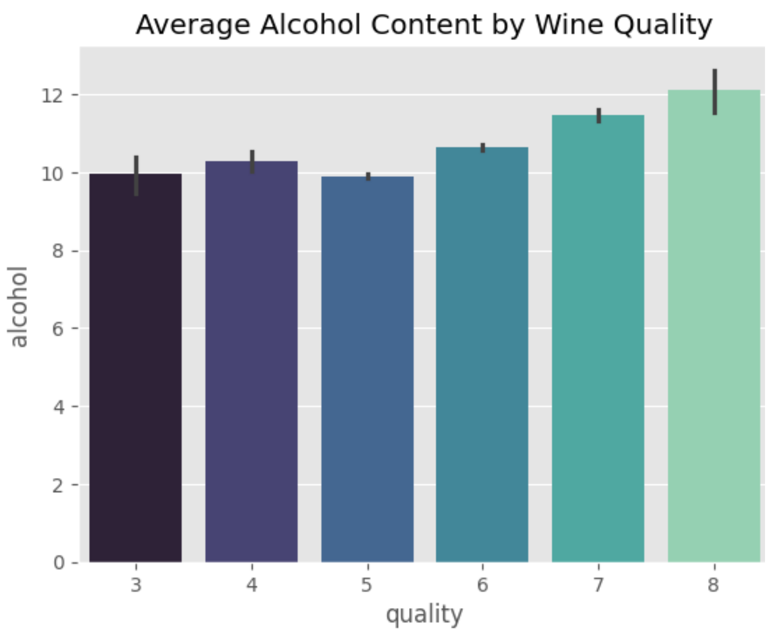
- The dataset used is the Red Wine Quality dataset from the UCI Machine Learning Repository. It consists of: 1599 samples
- 11 physicochemical attributes
- 1 quality label (integer 0–10, later converted to binary)

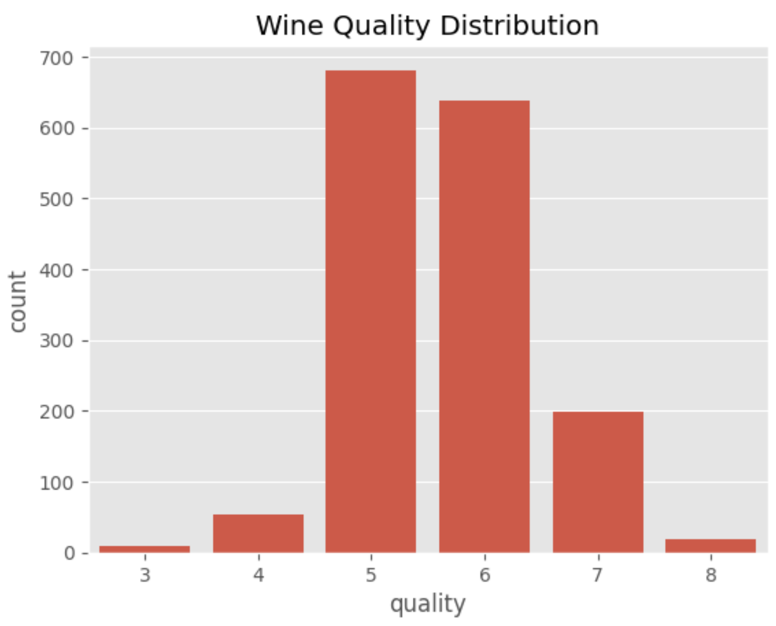


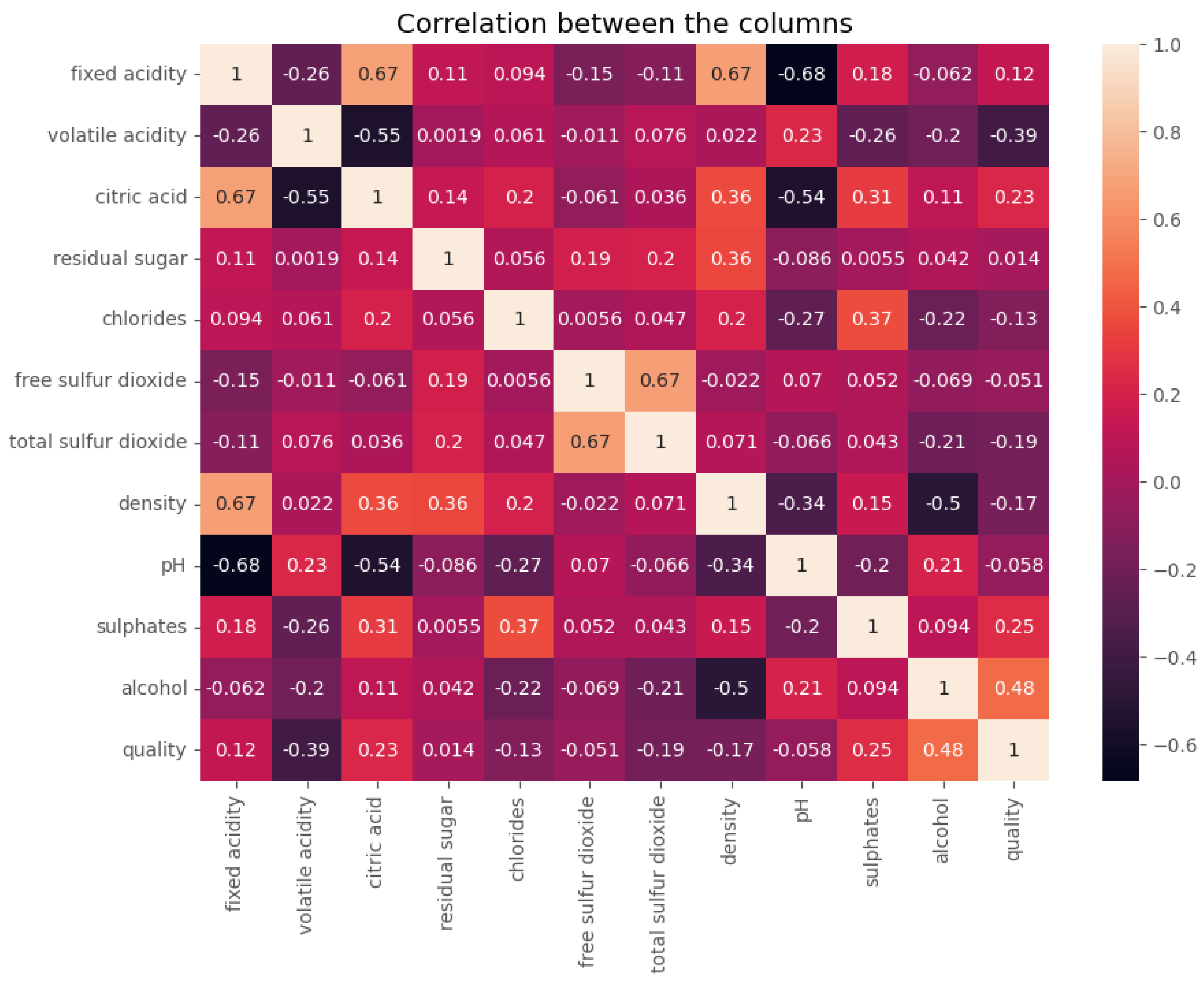
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

3.2 Feature Summary

Feature	**Description**	**Type**
fixed acidity	Non-volatile acids	Continuous
volatile acidity	Acetic acid affecting taste	Continuous
citric acid	Enhances freshness.	Continuous
residual sugar	Sugar after fermentation	Continuous
Chlorides	Salt concentration	Continuous
free sulfur dioxide	Prevents oxidation	Continuous
total sulfur dioxide	Overall sulfite level	Continuous
Density	Affected by sugar & alcohol	Continuous
pH	Acidity level	Continuous
Sulphates	Preservative agent	Continuous
Alcohol	Alcohol percentage	Continuous
quality (target)	Wine quality score (0–10)	Ordinal → converted to Binary

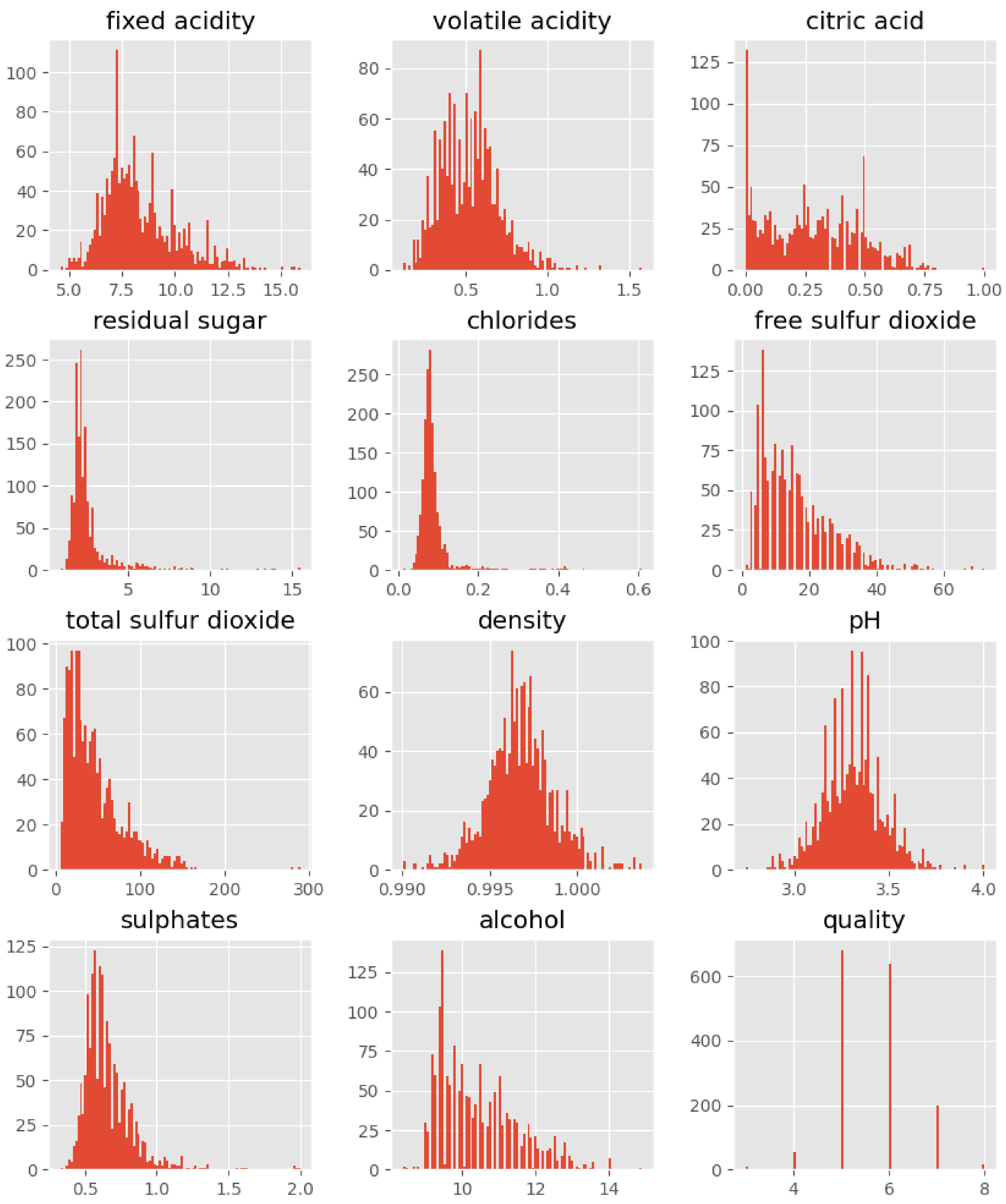






	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1	-0.26	0.67	0.11	0.094	-0.15	-0.11	0.67	-0.68	0.18	-0.062	0.12
volatile acidity	-0.26	1	-0.55	0.0019	0.061	-0.011	0.076	0.022	0.23	-0.26	-0.2	-0.39
citric acid	0.67	-0.55	1	0.14	0.2	-0.061	0.036	0.36	-0.54	0.31	0.11	0.23
residual sugar	0.11	0.0019	0.14	1	0.056	0.19	0.2	0.36	-0.086	0.0055	0.042	0.014
chlorides	0.094	0.061	0.2	0.056	1	0.0056	0.047	0.2	-0.27	0.37	-0.22	-0.13
free sulfur dioxide	-0.15	-0.011	-0.061	0.19	0.0056	1	0.67	-0.022	0.07	0.052	-0.069	-0.051
total sulfur dioxide	-0.11	0.076	0.036	0.2	0.047	0.67	1	0.071	-0.066	0.043	-0.21	-0.19
density	0.67	0.022	0.36	0.36	0.2	-0.022	0.071	1	-0.34	0.15	-0.5	-0.17
pH	-0.68	0.23	-0.54	-0.086	-0.27	0.07	-0.066	-0.34	1	-0.2	0.21	-0.058
sulphates	0.18	-0.26	0.31	0.0055	0.37	0.052	0.043	0.15	-0.2	1	0.094	0.25
alcohol	-0.062	-0.2	0.11	0.042	-0.22	-0.069	-0.21	-0.5	0.21	0.094	1	0.48
quality	0.12	-0.39	0.23	0.014	-0.13	-0.051	-0.19	-0.17	-0.058	0.25	0.48	1

Histograms for All Features

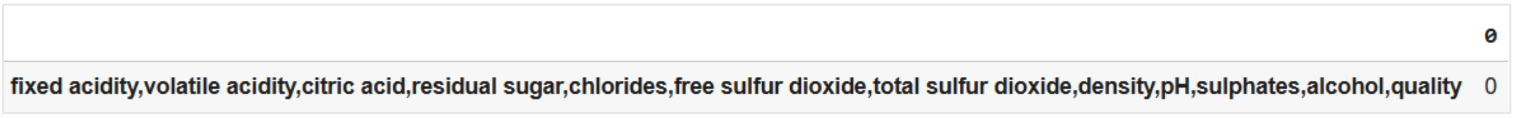


3.3 Data Preprocessing

Before training the models, the dataset was carefully examined and cleaned to improve model performance.

a) Handling Missing values

Checked missing values with **df.isnull().sum().**



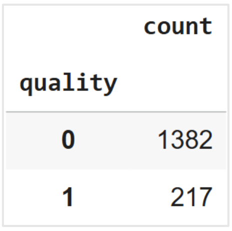
	0
fixed acidity,volatile acidity,citric acid,residual sugar,chlorides,free sulfur dioxide,total sulfur dioxide,density,pH,sulphates,alcohol,quality	0

Observation: The dataset did not contain missing values, so no imputation was required.

b) Checking Class Balance

The distribution of the *quality* target variable was analyzed. The dataset showed a skew toward medium quality (value = 5 and 6). To address this:

- Quality values were converted into two classes: Good (≥7) and Not Good (<7) for better binary classification.



	count
quality	
0	1382
1	217

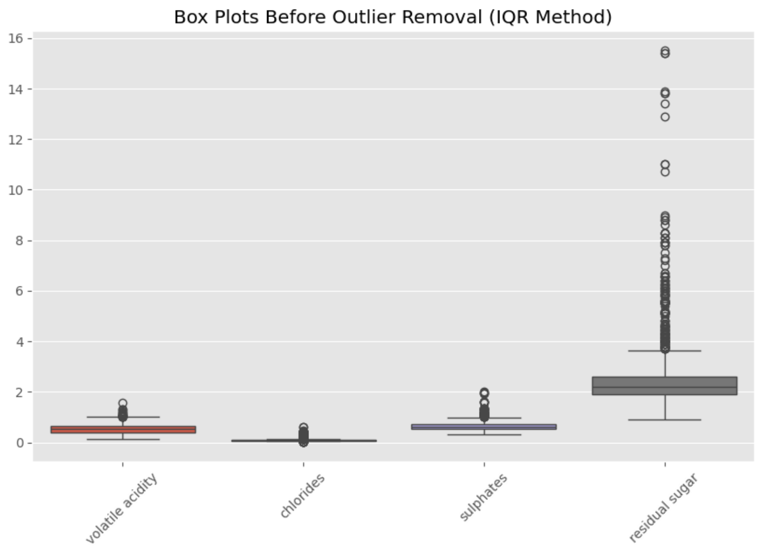
c) Outlier Detection and Handling

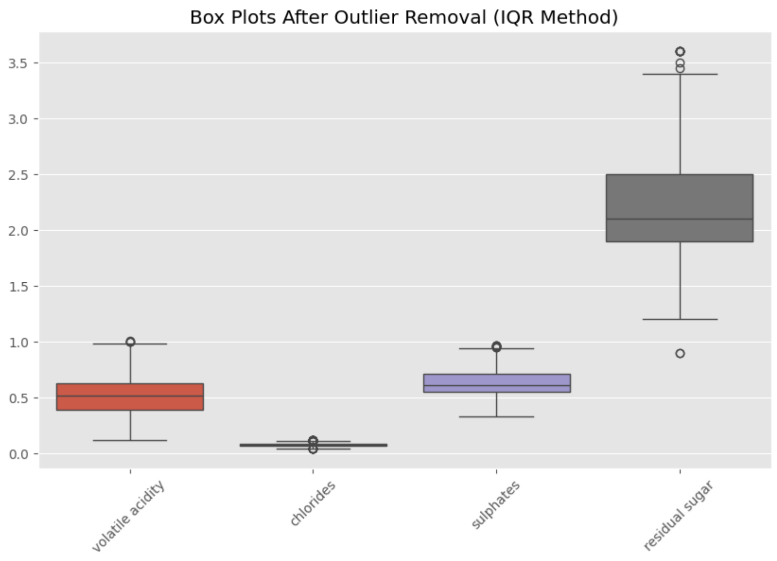
Outliers can affect model training, especially for linear models.

We used:

- Box plots and **df.quantile()** to visually and statistically identify outliers.
- Outliers beyond IQR range (Q1 - 1.5 IQR, Q3 + 1.5 IQR) for certain columns (such as *residual sugar, chlorides, sulphates*) were removed.

This ensured the model learned from stable and meaningful data.





d) Feature Scaling (Standardization)

Since the dataset contains features with different numeric ranges, we applied StandardScaler:

$$X_{scaled} = \frac{X - \mu}{\sigma}$$

where:

- μ= mean of the feature,
- σ= standard deviation.

Why scaling was needed:

- Logistic Regression is sensitive to feature scales.
- Although Decision Tree and Random Forest are not scale-dependent, scaling ensures uniformity across models and avoids implicit weighting.

e) Train-Test Split

The dataset was split:

$$70\% \text{ Training Data}, 30\% \text{ Test Data}$$

This ensures generalization capability.

3.4 Model Descriptions

a) Logistic Regression

Logistic Regression is a linear model used for binary classification. It models the probability of the positive class using the sigmoid function:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}}$$

The classification boundary is determined based on whether the probability ≥ 0.5.

Advantages:

- Simple and easy to interpret.
- Works well when relationship between features and target is linear.

Disadvantages:

- Does not perform well when the data contains non-linear patterns.
- Sensitive to outliers and unscaled data (hence scaling was crucial).

b) Decision Tree Classifier

Decision Trees create a flowchart-like structure by recursively splitting the dataset based on features that maximize information gain.

Common criterion:

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{C} p_i^2$$

Advantages:

- Easy to visualize and interpret.
- Works well without scaling and handles non-linear patterns.

Disadvantages:

- Prone to overfitting.
- Small changes in data can drastically change the tree structure.

c) Random Forest Classifier

Random Forest is an **ensemble** of multiple Decision Trees. Each tree is trained on randomly selected subsets of data and features. Final prediction is based on majority voting.

$$\text{Prediction} = \text{mode}(\text{Tree}_1, \text{Tree}_2, \dots, \text{Tree}_n)$$

Advantages:

- Reduces overfitting by averaging multiple trees.
- Handles non-linear relationships effectively.
- Performs well on most real-world datasets.

Disadvantages:

- More computationally expensive than a single decision tree.
- Harder to interpret due to multiple trees (black-box model).

4) Evaluation

To assess the performance of the classification models, multiple evaluation metrics and visual tools were considered. The primary metrics used were **Accuracy**, **Error Rate**, **Confusion**

Matrix, **Precision**, **Recall**, and **F1-score**. Additionally, model behaviour during training and generalization performance were interpreted using **Learning Curves** and **Validation Curves**.

4.1 Accuracyk and Error Rate

Accuracy measures the proportion of correctly predicted samples. Error rate represents the proportion of incorrect predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

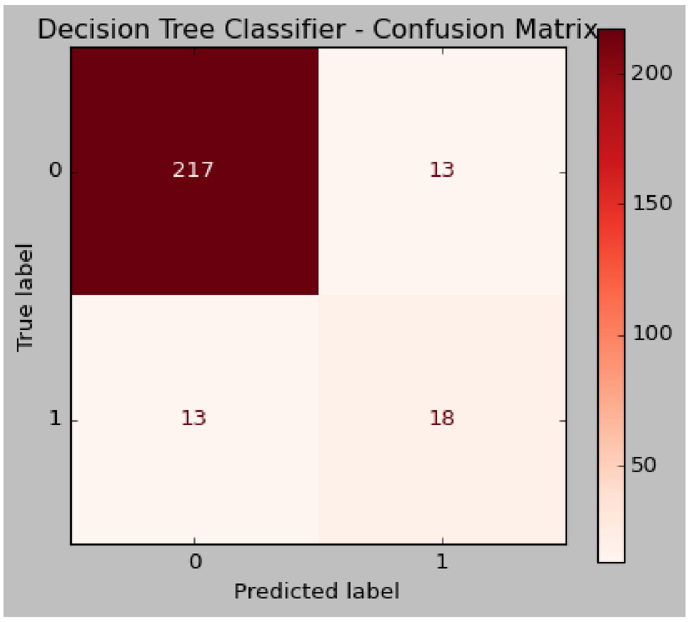
$$\text{Error Rate} = 1 - \text{Accuracy}$$

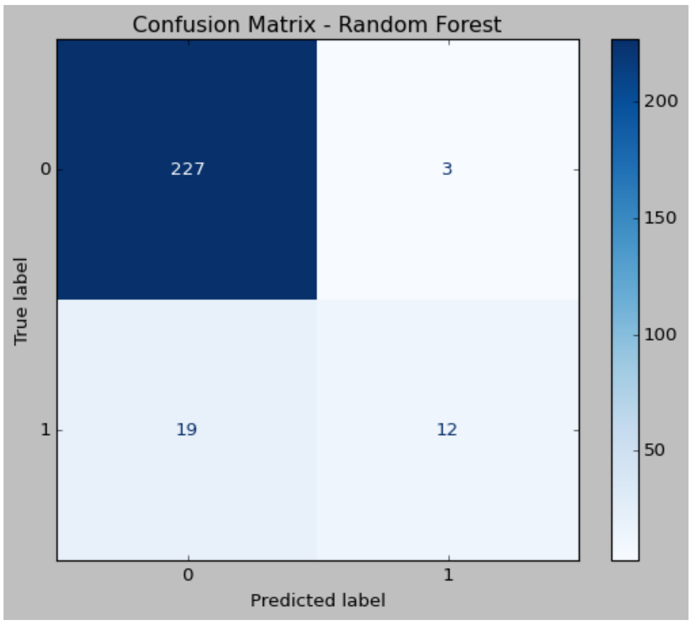
Model	**Accuracy (%)**	**Error Rate (%)**
Logistic Regression	**89.27**	10.73
Decision Tree	**90.04**	9.96
Random Forest	**93.34**	7.66

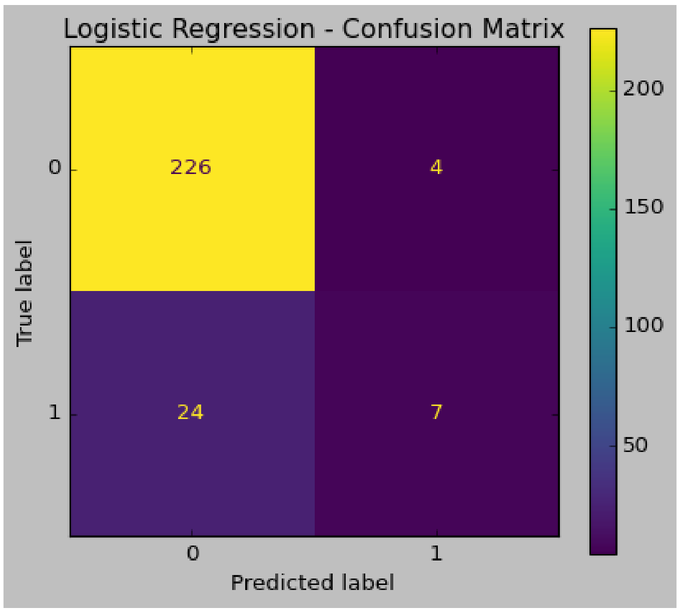
4.2 Confusion Matrix

A confusion matrix helps visualize correct and incorrect predictions across both classes (Good vs Not Good wine).
It provides insights into:

- False Positives (FP)
- False Negatives (FN)
- Bias toward one class (if any)







4.3 Precision, Recall, and F1 Score

These metrics show how well the models classify positive (good quality wine) samples.

- **Precision (Positive Predictive Value):**

$$\frac{TP}{TP + FP}$$

- **Recall (Sensitivity):**

$$\frac{TP}{TP + FN}$$

- **F1-Score (Harmonic Mean of Precision and Recall):**

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Logistic Regression-

```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       230
           1       0.64      0.23      0.33        31

    accuracy                           0.89       261
   macro avg       0.77      0.60      0.64       261
weighted avg       0.87      0.89      0.87       261
```

Decision Tree-

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95       230
           1       0.80      0.39      0.52        31

    accuracy                           0.92       261
   macro avg       0.86      0.69      0.74       261
weighted avg       0.91      0.92      0.90       261
```

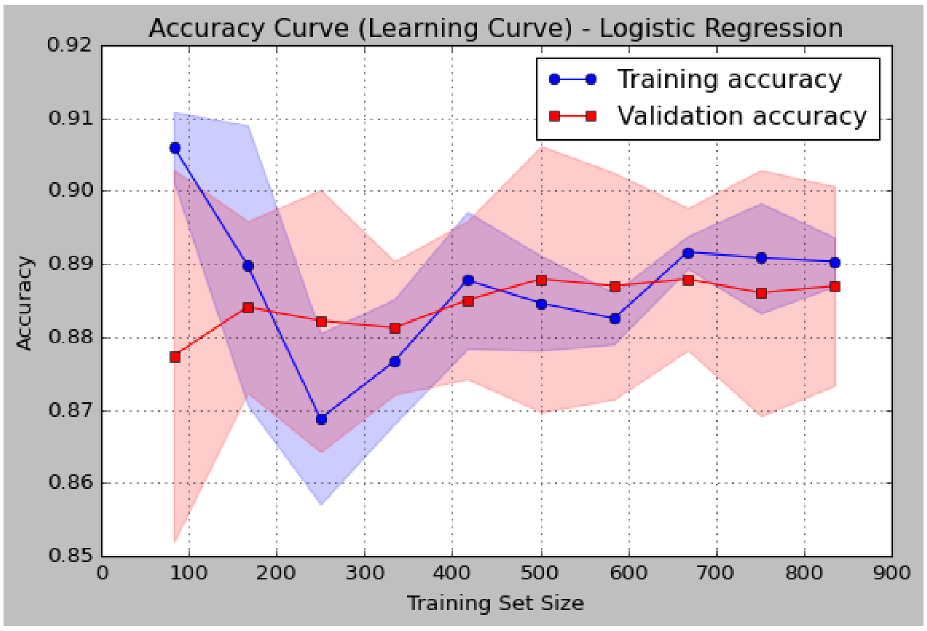
Random Forest-

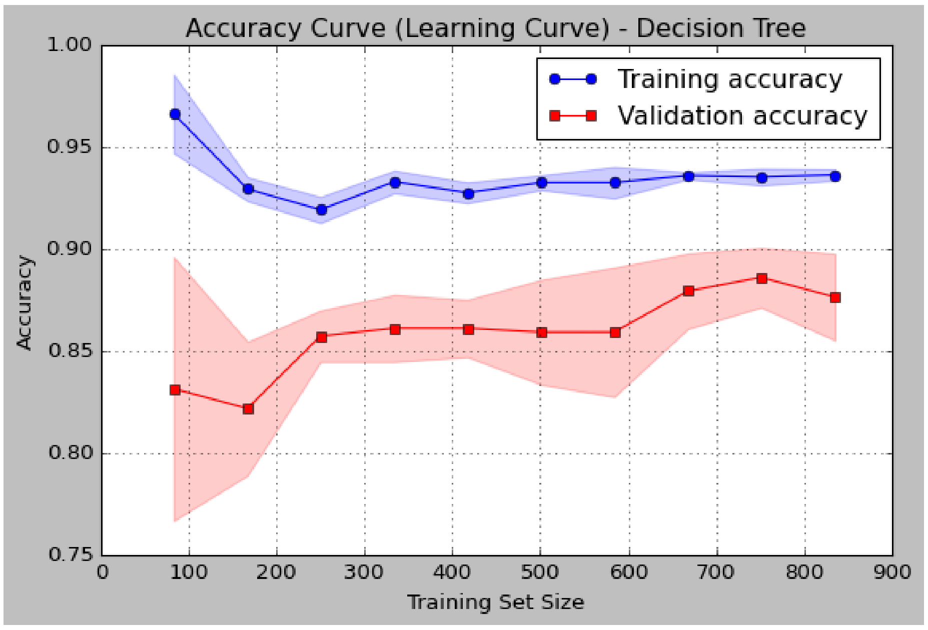
```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       230
           1       0.58      0.58      0.58        31

    accuracy                           0.90       261
   macro avg       0.76      0.76      0.76       261
weighted avg       0.90      0.90      0.90       261
```

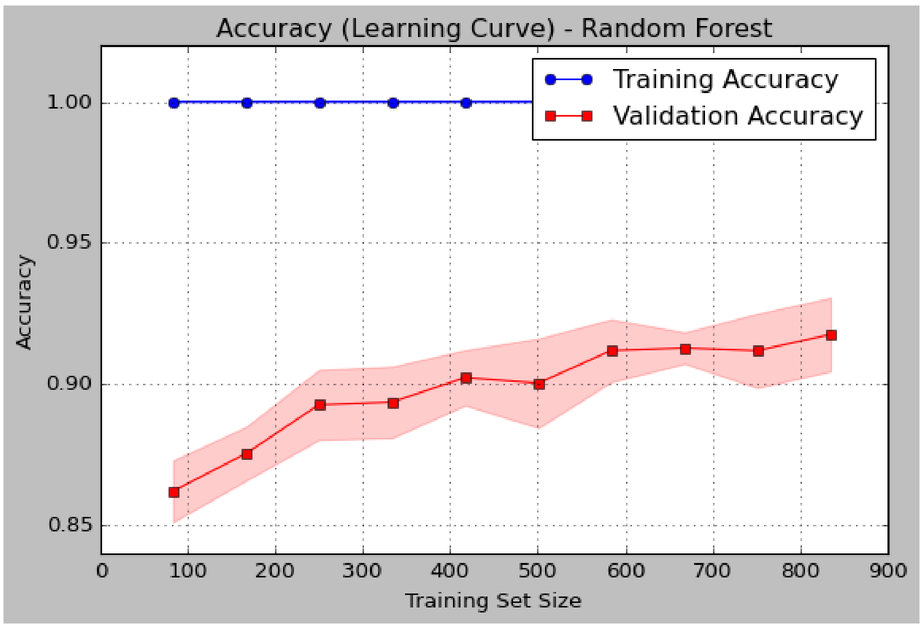
4.4 Learning Curves

Learning curves display the model's performance against the size of the training dataset.

- **Logistic Regression:** Shows stable learning and no significant overfitting.
- **Decision Tree:** Slight overfitting visible — training accuracy high, validation slightly lower.
- **Random Forest:** Good balance; curves converge smoothly—indicating stable generalization.



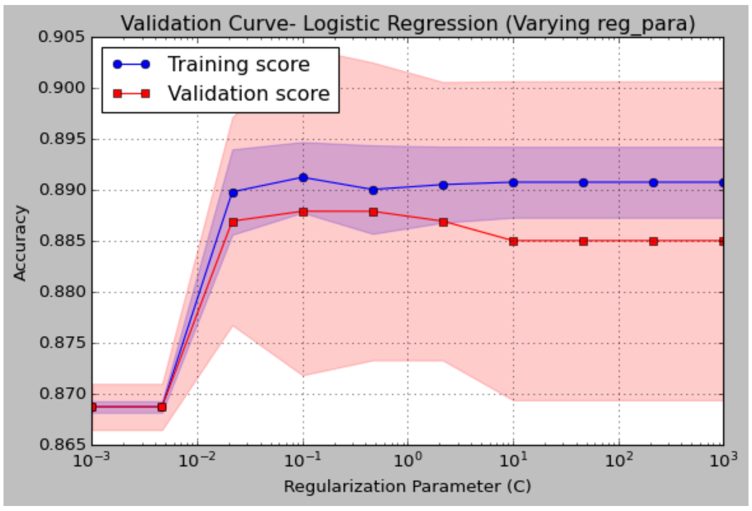


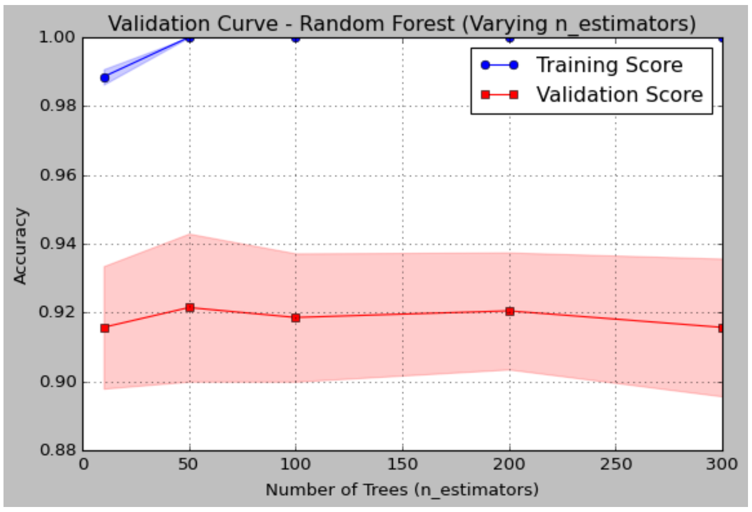


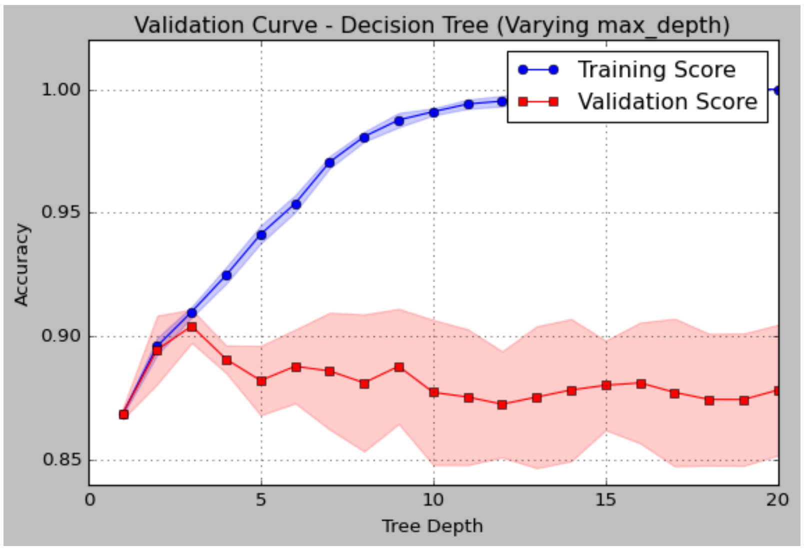
4.5 Validation Curves

Validation curves help understand how model performance changes with parameter tuning.

- **Decision Tree:** Increasing depth increases overfitting.
- **Random Forest:** Increasing number of estimators improves stability but increases computation time.



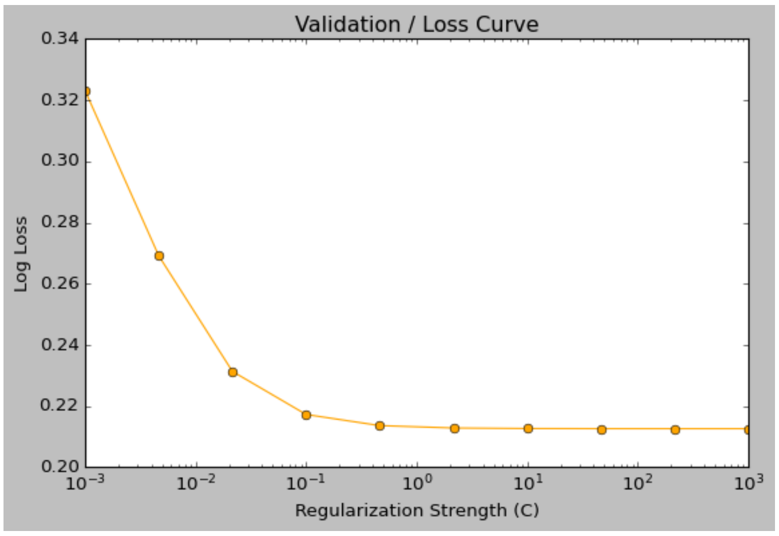


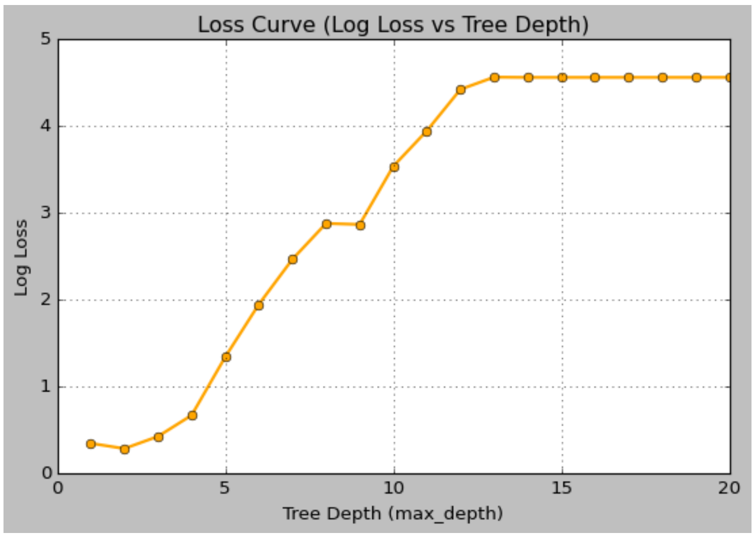


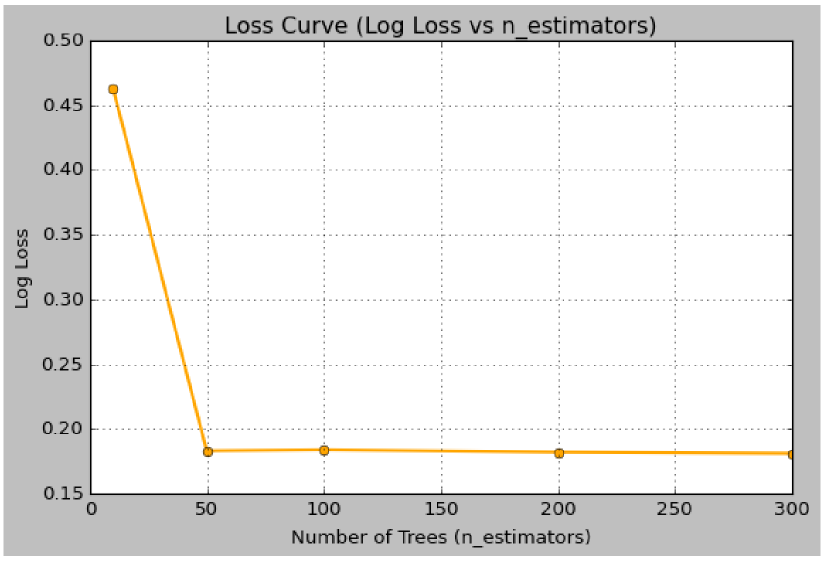
4.6 Loss Curve Analysis

Loss curves provide insight into how each model's predictive performance changes as their key hyperparameters are varied.

- **Logistic Regression:**
  Loss decreases as the regularization parameter *C* increases, showing improved model fit. The curve stabilizes at moderate regularization, indicating a good balance between bias and variance.
- **Decision Tree:**
  Loss increases as tree depth grows, meaning deeper trees tend to **overfit**. Controlling complexity (e.g., max_depth) is necessary to prevent poor generalization.
- **Random Forest:**
  Loss decreases sharply as the number of trees increases and then stabilizes. This shows that ensembles reduce overfitting and provide consistent, generalized performance after enough trees.







Key Observations

- Random Forest outperformed the other models because it aggregates multiple decision trees, reducing variance and improving robustness.
- Decision Tree showed high sensitivity to training data and slight overfitting, as seen in learning curve behaviour.
- Logistic Regression, while efficient and interpretable, struggled to capture the non-linear relationships in wine chemistry data.

5) Conclusion

This study aimed to classify wine quality using chemical composition features through three machine learning models. The preprocessing pipeline included outlier removal, label simplification, and feature scaling to ensure consistent model input. Performance evaluation demonstrated that:

- Random Forest achieved the highest accuracy (92.34%), making it the most reliable and robust model for this dataset.
- Decision Tree performed moderately well, but showed signs of overfitting due to high model complexity.
- Logistic Regression, while stable and interpretable, was limited in capturing complex feature interactions.

Overall, Random Forest is recommended for real-world deployment of wine quality prediction due to its strong predictive power and resistance to noise.

6) Code

Import Libraries

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from matplotlib import style

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, ConfusionMatrixDisplay, log_loss

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import validation_curve
from sklearn.metrics import log_loss
from sklearn.model_selection import validation_curve, learning_curve
```

Upload File

```
from google.colab import files
uploaded=files.upload()
```

Load Dataset and Display Head

```
wine_df = pd.read_csv('winequality-red.csv', sep=';')
wine_df.head()
```

Shape of Dataset

```
wine_df.shape
```

Dataset Information

```
wine_df.info()
```

Check Missing Values

```
wine_df.isnull().sum()
```

Statistical Summary

```
wine_df.describe()
```

Verify Column Names

```
import pandas as pd
wine_df = pd.read_csv('winequality-red.csv', sep=',')
```

```
print(wine_df.columns)
```

Value Counts of Quality Feature

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

Inspect Quality Values

```
print(wine_df['quality'].head())
print(wine_df['quality'].unique())
print(wine_df['quality'].dtype)
```

Plot Quality Distribution

```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
sns.countplot(x='quality', data=wine_df)
plt.title('Wine Quality Distribution')
plt.show()
```

Plot Histograms for All Features

```
wine_df.hist(bins=100, figsize=(10,12))
plt.show()
```

Heatmap of Feature Correlations

```
plt.figure(figsize=(10,7))
sns.heatmap(wine_df.corr(), annot=True)
plt.title('Correlation between the columns')
plt.show()
```

Display Correlation with Quality

```
wine_df.corr()['quality'].sort_values()
```

Alcohol vs Quality Barplot

```
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import style
style.use('ggplot')
sns.barplot(x='quality', y='alcohol', data=wine_df, palette='mako')
plt.title('Average Alcohol Content by Wine Quality')
plt.show()
```

Convert Quality to Binary (Good vs Bad)

```
wine_df['quality'] = wine_df.quality.apply(lambda x:1 if x>=7 else 0)
```

Value Counts After Binarization

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

Box Plot of Selected Features Before Outlier Removal

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,6))
sns.boxplot(data=wine_df[['volatile acidity', 'chlorides', 'sulphates', 'residual sugar']])
plt.title("Box Plots Before Outlier Removal (IQR Method)")
plt.xticks(rotation=45)
plt.show()
```

Removing outliers using IQR method

```
def remove_outliers(wine_df, column):
    Q1 = wine_df[column].quantile(0.25)
    Q3 = wine_df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return wine_df[(wine_df[column] >= lower_bound) & (wine_df[column] <= upper_bound)]
```

List of features to remove outliers from

```
cols_to_filter = ['volatile acidity', 'chlorides', 'sulphates', 'residual sugar']
df_clean = wine_df.copy()
for col in cols_to_filter:
    df_clean = remove_outliers(df_clean, col)
print("Original dataset shape:", wine_df.shape)
print("After removing outliers:", df_clean.shape)
```

Box Plot of Selected Features After Outlier Removal

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,6))
sns.boxplot(data=df_clean[['volatile acidity', 'chlorides', 'sulphates', 'residual sugar']])
plt.title("Box Plots After Outlier Removal (IQR Method)")
plt.xticks(rotation=45)
plt.show()
```

Split Dataset into Train and Test

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
X = df_clean.drop('quality', axis=1)
y = df_clean['quality']
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

Logistic Regression- Confusion Matrix (with Metrics Table)

```
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
logreg_pred = logreg.predict(X_test)
logreg_accuracy = accuracy_score(y_test, logreg_pred)
logreg_precision = precision_score(y_test, logreg_pred, average='weighted')
logreg_recall = recall_score(y_test, logreg_pred, average='weighted')
logreg_f1 = f1_score(y_test, logreg_pred, average='weighted')
print("Test Accuracy: {:.2f}%".format(logreg_accuracy * 100))
# Performance Table
logreg_metrics_table = pd.DataFrame({
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],
    'Score': [logreg_accuracy, logreg_precision, logreg_recall, logreg_f1]
})
print("\nPerformance Metrics:")
display(logreg_metrics_table)
# Confusion Matrix
cm = confusion_matrix(y_test, logreg_pred, labels=logreg.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)
disp.plot()
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

print("\nConfusion Matrix Breakdown:")
print("TN:", cm[0][0])
print("FP:", cm[0][1])
print("FN:", cm[1][0])
print("TP:", cm[1][1])
```

Logistic Regression Learning Curve

```
log_reg = LogisticRegression(max_iter=1000)
train_sizes, train_scores, test_scores = learning_curve(
    log_reg, X_train, y_train, cv=5, scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10)
)
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)
plt.figure(figsize=(8,5))
#Training accuracy curve
plt.plot(train_sizes, train_mean, label="Training accuracy", color='blue', marker='o')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='blue',
alpha=0.2)
#Validation accuracy curve
plt.plot(train_sizes, test_mean, label="Validation accuracy", color='red', marker='s')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color='red', alpha=0.2)
plt.title("Learning Curve - Logistic Regression")
plt.xlabel("Training Set Size")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

Logistic Regression Loss Curve

```
loss_values = []
```

```
for c in C_values:
    model = LogisticRegression(C=c, max_iter=1000)
    model.fit(X_train, y_train)
    y_prob = model.predict_proba(X_test)
    loss_values.append(log_loss(y_test, y_prob))
plt.figure(figsize=(8, 5))
plt.plot(C_values, loss_values, color='orange', marker='o')
plt.xscale('log')
plt.xlabel('Regularization Strength (C)')
plt.ylabel('Log Loss')
plt.title('Validation / Loss Curve')
plt.show()
```

Logistic Regression Validation Curve

```
train_scores, valid_scores = validation_curve(
    LogisticRegression(max_iter=1000),
    X_train, y_train,
    param_name="C",
    param_range=C_values,
    cv=5,
    scoring="accuracy"
)
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)
plt.figure(figsize=(8, 5))
plt.semilogx(C_values, train_mean, label="Training score", color="blue", marker="o")
plt.fill_between(C_values, train_mean - train_std, train_mean + train_std, color="blue",
alpha=0.2)
plt.semilogx(C_values, valid_mean, label="Validation score", color="red", marker="s")
plt.fill_between(C_values, valid_mean - valid_std, valid_mean + valid_std, color="red", alpha=0.2)
plt.title("Validation Curve for Logistic Regression")
plt.xlabel("Regularization Parameter (C)")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid(True)
plt.show()
```

Logistic Regression Error Rate

```
error_rate = 1 - logreg_accuracy
print("Error Rate: {:.2f}%".format(error_rate * 100))
```

Confusion Matrix - Decision Tree (with Metrics Table)

```
dtree = DecisionTreeClassifier(max_depth=6, min_samples_split=10, min_samples_leaf=5)
dtree.fit(X_train, y_train)
dtree_pred = dtree.predict(X_test)
#Metrics
dtree_accuracy = accuracy_score(y_test, dtree_pred)
dtree_precision = precision_score(y_test, dtree_pred, average='weighted')
dtree_recall = recall_score(y_test, dtree_pred, average='weighted')
dtree_f1 = f1_score(y_test, dtree_pred, average='weighted')
print("Test Accuracy: {:.2f}%".format(dtree_accuracy * 100))
#Performance Table
dtree_metrics_table = pd.DataFrame({
   'Metric': ['Accuracy', 'Precision', 'Recall', 'F1-Score'],
   'Score': [dtree_accuracy, dtree_precision, dtree_recall, dtree_f1]
})
print("\nPerformance Metrics:")
display(dtree_metrics_table)
#Confusion Matrix
```

```
cm = confusion_matrix(y_test, dtree_pred, labels=dtree.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dtree.classes_)
disp.plot(cmap='Reds')   # Red color theme
plt.title("Confusion Matrix - Decision Tree")
plt.show()
print("\nConfusion Matrix Breakdown:")
print("TN:", cm[0][0])
print("FP:", cm[0][1])
print("FN:", cm[1][0])
print("TP:", cm[1][1])
```

Decision Tree Learning Curve

```
train_sizes, train_scores, test_scores = learning_curve(
    dtree, X_train, y_train, cv=5, scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10)
)
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)
plt.figure(figsize=(8,5))
plt.plot(train_sizes, train_mean, label="Training accuracy", color='blue', marker='o')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='blue',
alpha=0.2)
plt.plot(train_sizes, test_mean, label="Validation accuracy", color='red', marker='s')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color='red', alpha=0.2)
plt.title("Learning Curve - Decision Tree")
plt.xlabel("Training Set Size")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

Decision Tree Validation Curve

```
param_range = np.arange(1, 21)
train_scores, valid_scores = validation_curve(
    DecisionTreeClassifier(),
    X_train, y_train,
    param_name="max_depth",
    param_range=param_range,
    cv=5,
    scoring="accuracy"
)
train_mean = np.mean(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
train_std = np.std(train_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)
plt.figure(figsize=(8,5))
plt.plot(param_range, train_mean, label="Training Score", color='blue', marker='o')
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, color='blue', alpha=0.2)
plt.plot(param_range, valid_mean, label="Validation Score", color='red', marker='s')
plt.fill_between(param_range, valid_mean - valid_std, valid_mean + valid_std, color='red', alpha=0.2)
plt.title("Validation Curve - Decision Tree (Varying max_depth)")
plt.xlabel("Tree Depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

Decision Tree Loss Curve

```
loss_values = []
depth_range = range(1, 21)
```

```
for depth in depth_range:
    dt_temp = DecisionTreeClassifier(max_depth=depth, random_state=42)
    dt_temp.fit(X_train, y_train)
    y_pred_prob = dt_temp.predict_proba(X_test)
    loss_values.append(log_loss(y_test, y_pred_prob))
plt.figure(figsize=(8,5))
plt.plot(depth_range, loss_values, color='orange', marker='o', linewidth=2)
plt.title("Loss Curve- Decison Tree")
plt.xlabel("Tree Depth (max_depth)")
plt.ylabel("Log Loss")
plt.grid(True)
plt.show()
```

Decision Tree Error Rate

```
error_rate = np.mean(dtree_pred != y_test)
print("Error Rate: {:.2f}%".format(error_rate * 100))
```

Train Random Forest Classifier

```
rforest = RandomForestClassifier()
rforest.fit(X_train, y_train)
rforest_pred = rforest.predict(X_test)
rforest_acc = accuracy_score(rforest_pred, y_test)
print("Test accuracy: {:.2f}%".format(rforest_acc*100))
print(classification_report(y_test, rforest_pred))
cm = confusion_matrix(y_test, rforest_pred, labels=rforest.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rforest.classes_)
disp.plot(cmap='Blues')
plt.title("Confusion Matrix - Random Forest")
plt.show()
print("TN:", cm[0][0])
print("FP:", cm[0][1])
print("FN:", cm[1][0])
print("TP:", cm[1][1])
```

Random Forest Learning Curve

```
train_sizes, train_scores, test_scores = learning_curve(
    rforest, X_train, y_train, cv=5, scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10)
)
train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_std = np.std(test_scores, axis=1)
plt.figure(figsize=(8,5))
plt.plot(train_sizes, train_mean, label='Training Accuracy', color='blue', marker='o')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, color='blue',
alpha=0.2)
plt.plot(train_sizes, test_mean, label='Validation Accuracy', color='red', marker='s')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, color='red', alpha=0.2)
plt.title("Learning Curve - Random Forest")
plt.xlabel("Training Set Size")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

Random Forest Validation Curve (n_estimators)

```
param_range = [10, 50, 100, 200, 300]
train_scores, valid_scores = validation_curve(
    RandomForestClassifier(random_state=42),
    X_train, y_train,
    param_name="n_estimators",
    param_range=param_range,
    cv=5,
    scoring="accuracy"
)
train_mean = np.mean(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
train_std = np.std(train_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)
plt.figure(figsize=(8,5))
plt.plot(param_range, train_mean, label='Training Score', color='blue', marker='o')
plt.fill_between(param_range, train_mean - train_std, train_mean + train_std, color='blue', alpha=0.2)
plt.plot(param_range, valid_mean, label='Validation Score', color='red', marker='s')
plt.fill_between(param_range, valid_mean - valid_std, valid_mean + valid_std, color='red', alpha=0.2)
plt.title("Validation Curve - Random Forest (Varying n_estimators)")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()
```

Random Forest Loss Curve

```
loss_values = []
for n in param_range:
    rf_temp = RandomForestClassifier(n_estimators=n, random_state=42)
    rf_temp.fit(X_train, y_train)
    y_prob = rf_temp.predict_proba(X_test)
    loss_values.append(log_loss(y_test, y_prob))
plt.figure(figsize=(8,5))
plt.plot(param_range, loss_values, color='orange', marker='o', linewidth=2)
plt.title("Loss Curve- Random Forest")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Log Loss")
plt.grid(True)
plt.show()
```

Random Forest Error Rate

```
error_rate = np.mean(rforest_pred != y_test)
print("Error Rate: {:.2f}%".format(error_rate * 100))
```