Major Project

<u>Title--Wine Quality Analysis</u>

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are munch more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Two datasets were combined and few values were randomly removed.

Attribute Information:

Input variables (based on physicochemical tests):

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

Output variable (based on sensory data):

12 - quality (score between 0 and 10)

NOTE

TO CONVERT THE XLSX From CSV - https://cloudconvert.com/xls-to-csv

Aim:

The primary aim of the **Wine Quality Analysis** project is to build and evaluate various machine learning models to predict the quality of wine based on its physicochemical attributes. Specifically, this project seeks to achieve the following objectives:

1. **Data Understanding and Preparation**:
    - Analyze the provided wine dataset to understand its structure, attributes, and quality ratings.
    - Preprocess the data by handling any missing values, performing feature scaling, and preparing it for model training.
2. **Model Development**:
    - Implement multiple classification algorithms (e.g., Random Forest, Logistic Regression, Support Vector Machines, and k-Nearest Neighbors) to predict wine quality.
    - Evaluate the performance of each model based on various metrics such as accuracy, precision, recall, and F1-score.
3. **Performance Evaluation**:
    - Compare the performance of different models using appropriate evaluation metrics and visualizations.
    - Identify which model provides the best predictive performance for the wine quality classification task.
4. **Insights and Recommendations**:
    - Analyze the results to gain insights into which physicochemical attributes most significantly impact wine quality.
    - Provide recommendations for winemakers on how to potentially improve wine quality based on the identified features.
5. **Real-World Applications**:
    - Highlight the importance of predictive modeling in the wine industry and how it can assist in quality control and product improvement.

Algorithm:

In the **Wine Quality Analysis** project, several machine learning algorithms are implemented to predict the quality of wine based on its physicochemical properties. Below is a brief overview of each algorithm used:

1. **Random Forest Classifier**
    - **Type**: Ensemble Learning (Bagging)
    - **Description**: Random Forest is a robust and widely used ensemble learning algorithm that builds multiple decision trees during training. It merges their predictions (majority voting for classification) to improve accuracy and control overfitting.
    - **Strengths**:
        - Handles both classification and regression tasks well.
        - Robust to outliers and noise.
        - Provides feature importance scores, helping in understanding which features contribute most to predictions.

2. **Logistic Regression**
   - **Type**: Statistical Method
   - **Description**: Logistic Regression is a linear model used for binary classification problems. It estimates the probability of a binary response based on one or more predictor variables. Although primarily for binary outcomes, it can be adapted for multi-class classification using techniques like one-vs-all.
   - **Strengths**:
     - Simple and interpretable.
     - Works well when the relationship between features and target is approximately linear.
     - Outputs probabilities, which can be useful for ranking.
3. **Support Vector Machine (SVM)**
   - **Type**: Classification
   - **Description**: SVM is a supervised learning algorithm that finds the hyperplane that best separates different classes in the feature space. The algorithm can work with both linear and non-linear data by applying kernel functions.
   - **Strengths**:
     - Effective in high-dimensional spaces.
     - Robust to overfitting in high-dimensional datasets.
     - Can handle both linear and non-linear classifications effectively.
4. **k-Nearest Neighbors (k-NN)**
   - **Type**: Instance-Based Learning
   - **Description**: k-NN is a simple, non-parametric algorithm used for classification and regression. It classifies data points based on the majority class of their k-nearest neighbors in the feature space.
   - **Strengths**:
     - Simple to implement and understand.
     - Naturally handles multi-class classification.
     - No explicit training phase, as it is lazy learning.

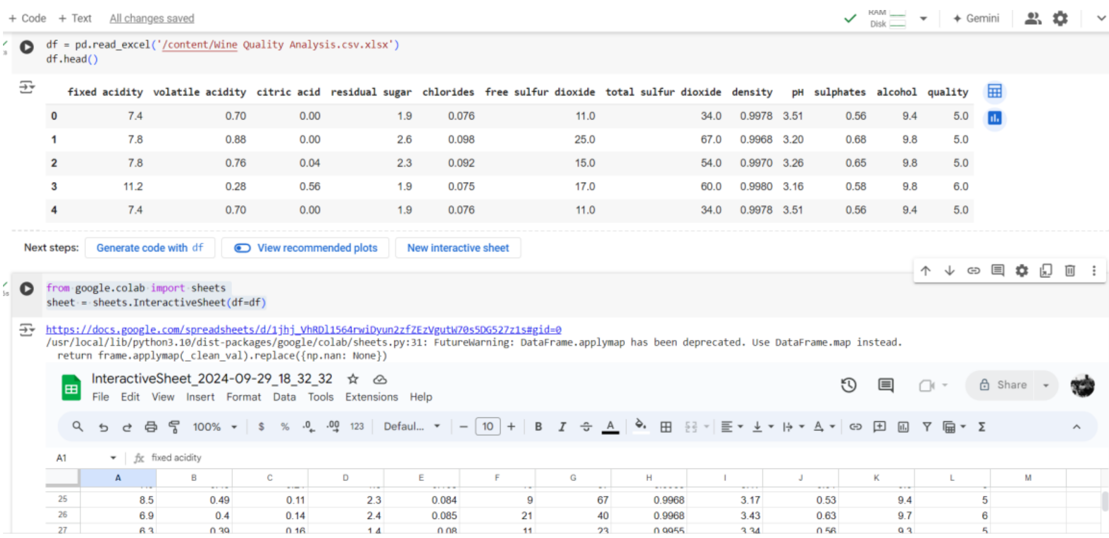
Summary of Algorithm Selection

- **Diversity**: The selection of algorithms reflects a variety of approaches to classification—ensemble methods, linear models, non-linear classifiers, and instance-based learning.
- **Performance Comparison**: Using multiple algorithms allows for a thorough comparison to determine which method is most effective for predicting wine quality.
- **Real-World Application**: Each algorithm has its own advantages and limitations, making them suitable for different types of datasets and problems, which is relevant for practical applications in the wine industry.

Implementation and Output

Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_score, recall_score, f1_score
Load the dataset
df = pd.read_excel('/content/Wine Quality Analysis.csv.xlsx')
df.head()

=>



```
df = pd.read_excel('/content/Wine Quality Analysis.csv.xlsx')
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5.0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5.0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5.0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6.0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5.0

```
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=df)
```

from google.colab import sheets
sheet = sheets.InteractiveSheet(df=df)

=>
https://docs.google.com/spreadsheets/d/1jhj_VhRDl1564rwiDyun2zfZEzVgutW70s5DG527z1s#gid=0
/usr/local/lib/python3.10/dist-packages/google/colab/sheets.py:31: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  return frame.applymap(_clean_val).replace({np.nan: None})

Exploratory Data Analysis (EDA)

Check for missing values

print(df.isnull().sum())

```
=>
fixed acidity          0
volatile acidity       0
citric acid            0
residual sugar         0
chlorides              0
free sulfur dioxide    0
total sulfur dioxide   1
density                0
pH                     1
sulphates              0
alcohol                0
quality                1
dtype: int64
```

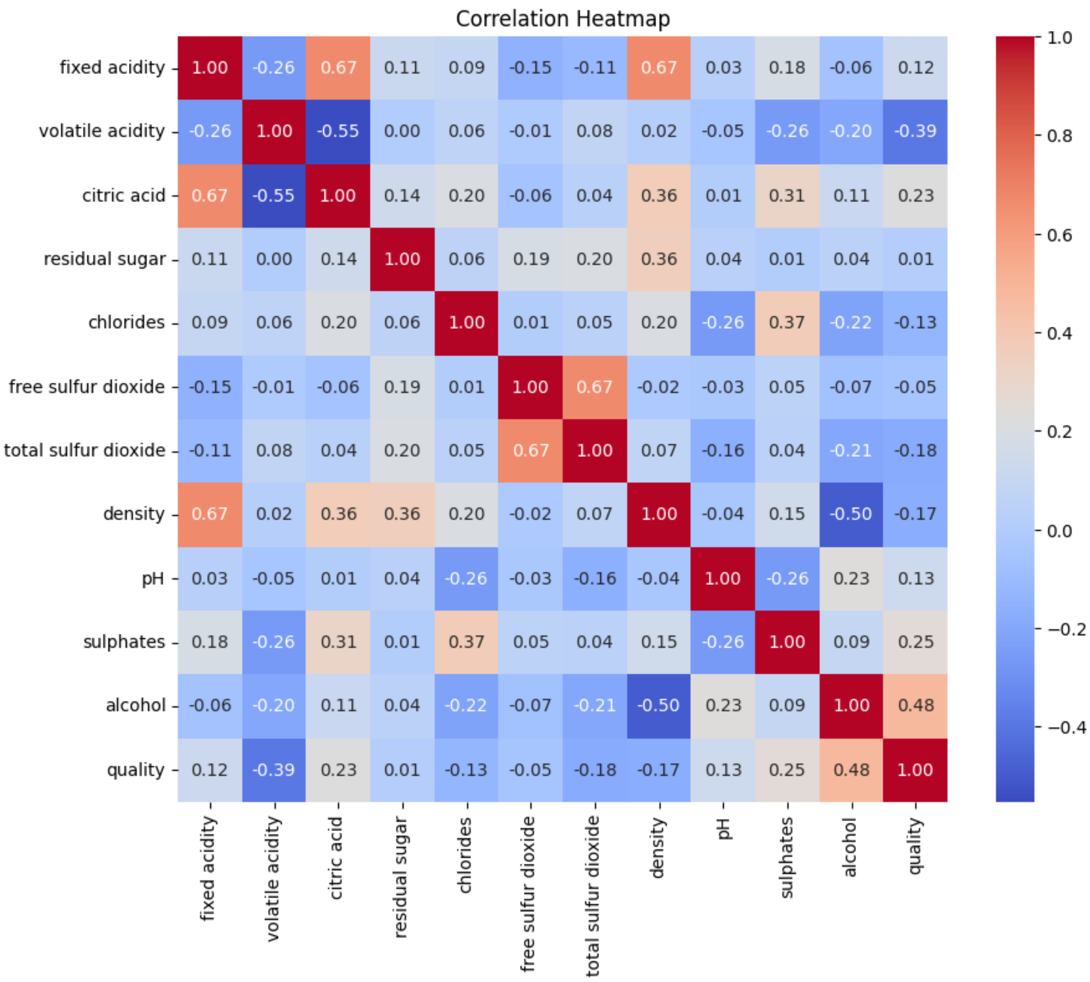
print(df.describe())

```
=>
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1598.000000  1599.000000
mean      0.087467            15.874922             46.433041     0.996747
std       0.047065            10.460157             32.876249     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1598.000000  1599.000000  1599.000000  1598.000000
mean      3.498586     0.658149    10.422983     5.636421
std       0.080346     0.169507     1.065668     0.807665
min       2.740000     0.330000     8.400000     3.000000
25%       3.520000     0.550000     9.500000     5.000000
50%       3.520000     0.620000    10.200000     6.000000
75%       3.520000     0.730000    11.100000     6.000000
max       3.900000     2.000000    14.900000     8.000000
```

Visualizing the correlation between features

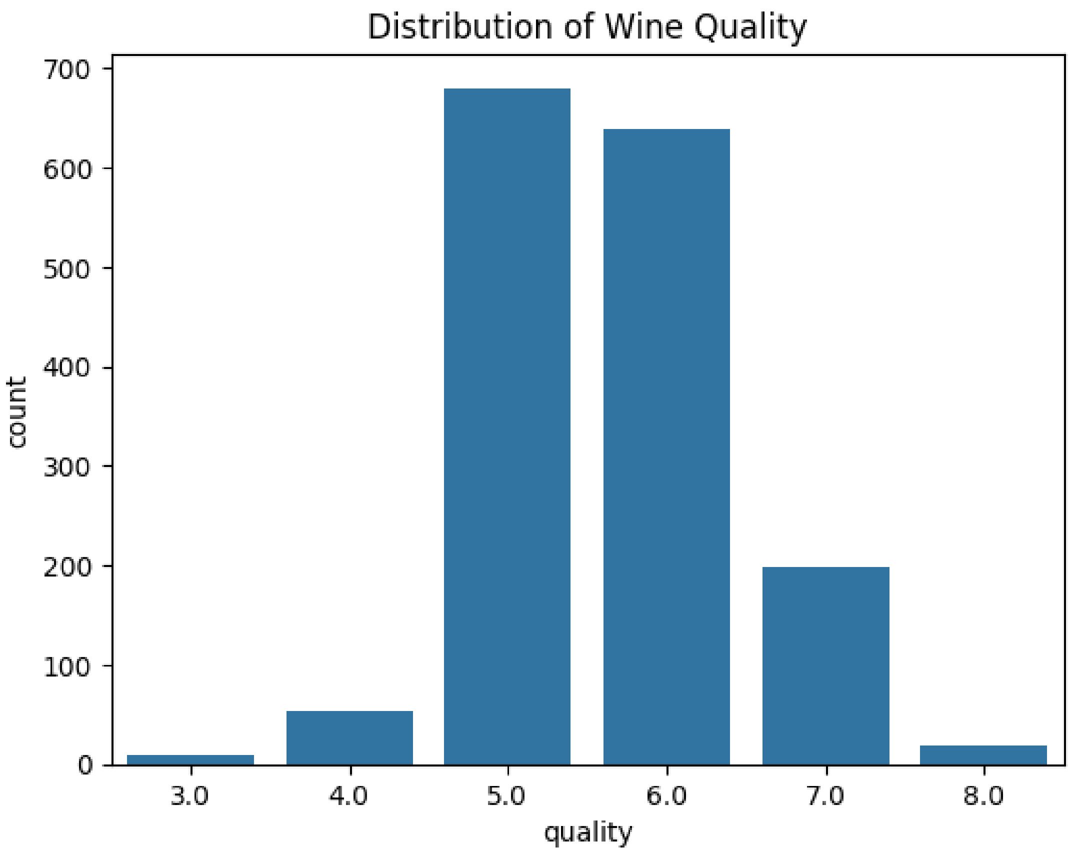
```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

=>



Checking the distribution of the target variable (quality)

```
sns.countplot(x='quality', data=df)
plt.title('Distribution of Wine Quality')
plt.show()
```



Handling Missing Values

```
imputer = SimpleImputer(strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

print(df_imputed.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
```

Feature Selection

```
X = df_imputed.drop('quality', axis=1)
y = df_imputed['quality']

selector = SelectKBest(score_func=f_classif, k=8)
X_new = selector.fit_transform(X, y)

selected_features = X.columns[selector.get_support()]
print(f"Selected features: {selected_features}")
```

=>

```
Selected features: Index(['fixed acidity', 'volatile acidity', 'citric acid',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol'],
      dtype='object')
```

Data Splitting and Scaling

```
X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.3,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

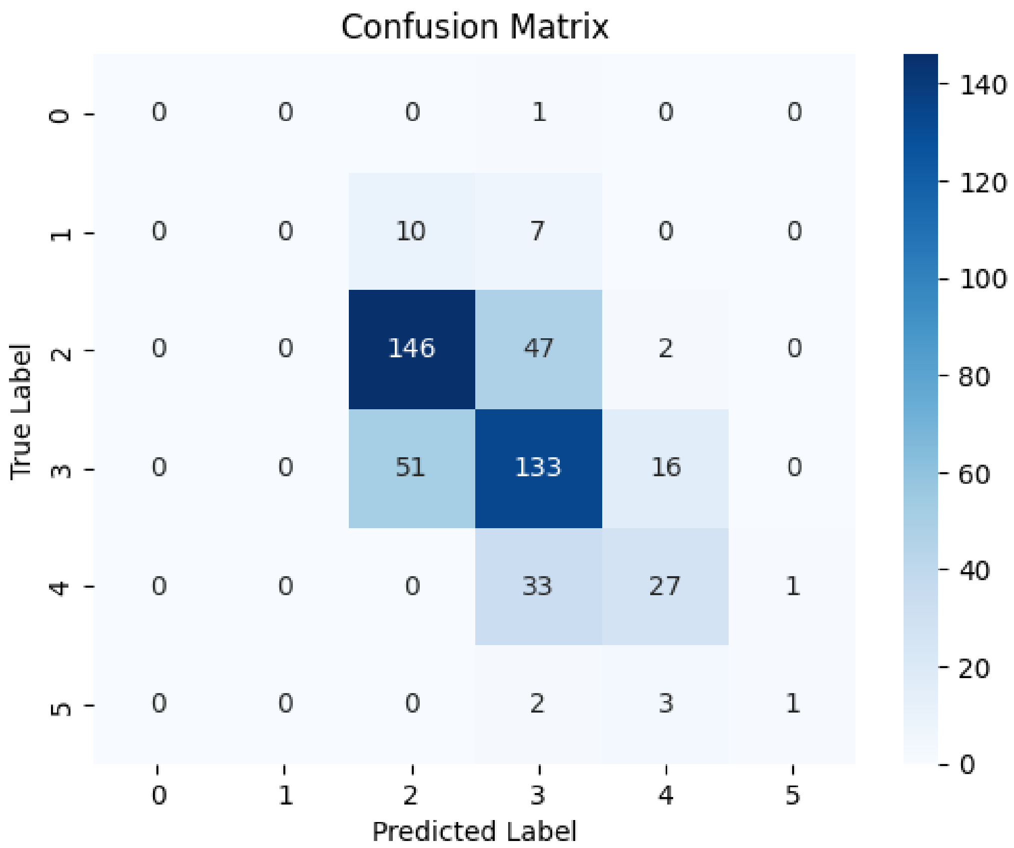
Model Building (Random Forest Classifier)

```
df_imputed['quality'] = df_imputed['quality'].astype(int)
X = df_imputed.drop('quality', axis=1)
y = df_imputed['quality']

selector = SelectKBest(score_func=f_classif, k=8)
X_new = selector.fit_transform(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size=0.3,
random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf = RandomForestClassifier(n_estimators=100, random_state=42)

clf.fit(X_train_scaled, y_train)

y_pred = clf.predict(X_test_scaled)

y_pred = np.round(y_pred).astype(int)

accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy * 100:.2f}%')

print("Classification Report:")
print(classification_report(y_test, y_pred))

conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

=>

```
Accuracy: 63.96%
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00        17
           5       0.71      0.75      0.73       195
           6       0.60      0.67      0.63       200
           7       0.56      0.44      0.50        61
           8       0.50      0.17      0.25         6

    accuracy                           0.64       480
   macro avg       0.39      0.34      0.35       480
weighted avg       0.61      0.64      0.62       480

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



Function to evaluate and display results for models

```
def evaluate_model(model, model_name, metrics_dict):
    print(f"\n{model_name} Results:")

    # Train the model
    model.fit(X_train_scaled, y_train)

    # Make predictions
    y_pred = model.predict(X_test_scaled)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Save the metrics
    metrics_dict['Model'].append(model_name)
    metrics_dict['Accuracy'].append(accuracy)
    metrics_dict['Precision'].append(precision)
    metrics_dict['Recall'].append(recall)
    metrics_dict['F1-Score'].append(f1)
```

```
    print(f'Accuracy: {accuracy * 100:.2f}%')

    # Detailed classification report
    print("Classification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Plot the confusion matrix
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
    plt.title(f'{model_name} Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

# Dictionary to store the metrics for each model
metrics_dict = {'Model': [], 'Accuracy': [], 'Precision': [], 'Recall': [], 'F1-Score': []}

# Logistic Regression
log_reg = LogisticRegression(random_state=42, max_iter=1000)
evaluate_model(log_reg, "Logistic Regression", metrics_dict)

# Support Vector Machine (SVM)
svm_clf = SVC(kernel='linear', random_state=42)
evaluate_model(svm_clf, "Support Vector Machine (SVM)", metrics_dict)

# k-Nearest Neighbors (k-NN)
knn_clf = KNeighborsClassifier(n_neighbors=5)
evaluate_model(knn_clf, "k-Nearest Neighbors (k-NN)", metrics_dict)

# RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
evaluate_model(clf, "RandomForest", metrics_dict)

# Convert metrics_dict to DataFrame for easier plotting
import pandas as pd
metrics_df = pd.DataFrame(metrics_dict)

# Plot the results in a bar chart for each metric
plt.figure(figsize=(10, 6))
metrics_df.set_index('Model').plot(kind='bar', figsize=(10, 6))
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.show()
```

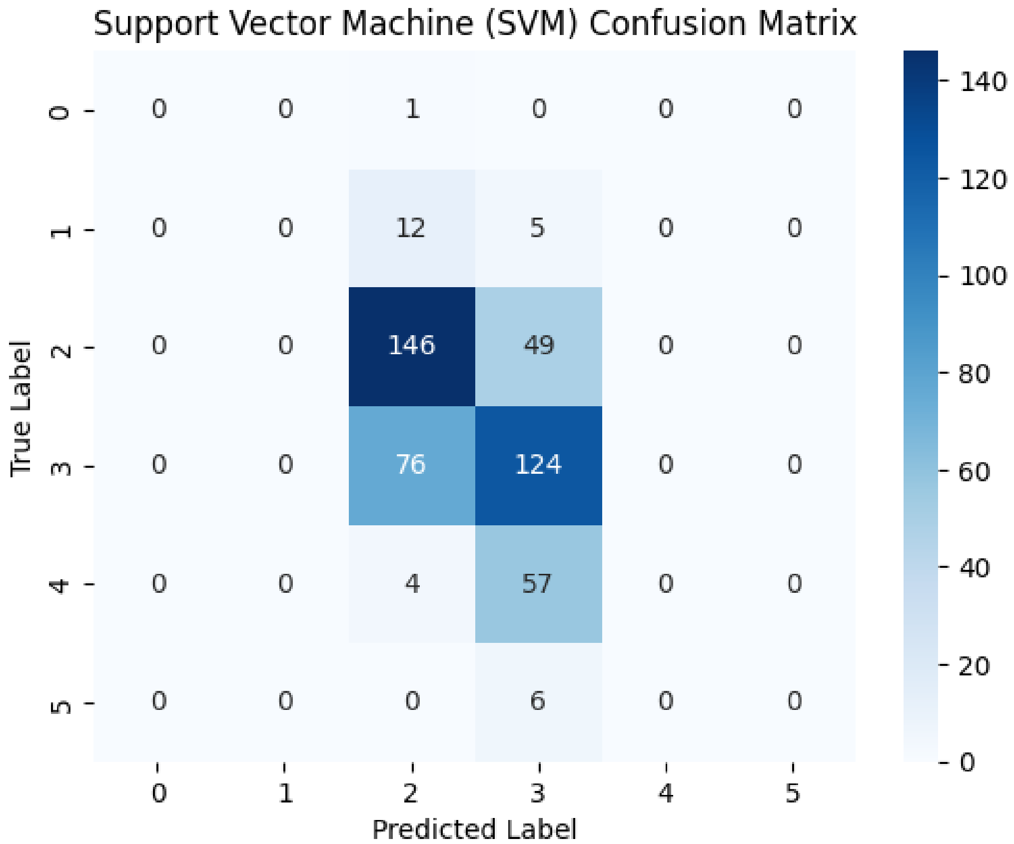
```
=>
Logistic Regression Results:
Accuracy: 54.37%
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00        17
           5       0.61      0.73      0.67       195
           6       0.51      0.53      0.52       200
           7       0.33      0.21      0.26        61
           8       0.00      0.00      0.00         6

    accuracy                           0.54       480
   macro avg       0.24      0.25      0.24       480
weighted avg       0.50      0.54      0.52       480

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Support Vector Machine (SVM) Results:
Accuracy: 56.25%
Classification Report:

	precision	recall	f1-score	support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	17
5	0.61	0.75	0.67	195
6	0.51	0.62	0.56	200
7	0.00	0.00	0.00	61
8	0.00	0.00	0.00	6
accuracy			0.56	480
macro avg	0.19	0.23	0.21	480
weighted avg	0.46	0.56	0.51	480

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

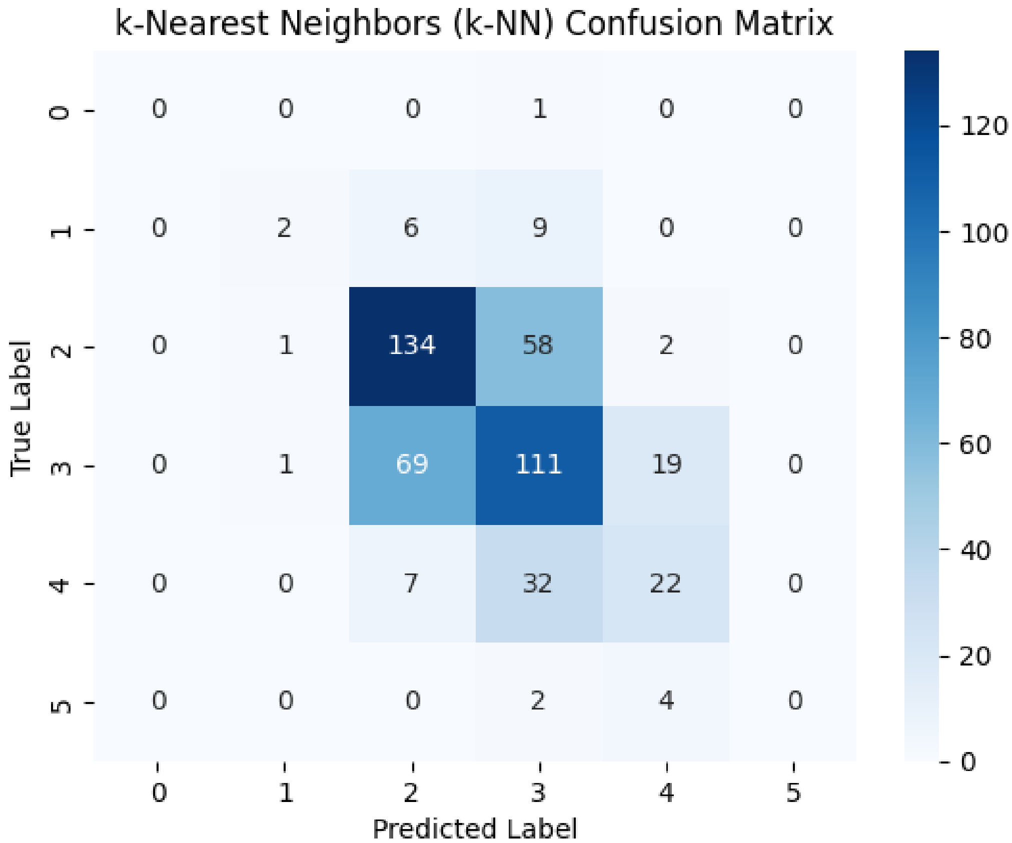
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))



k-Nearest Neighbors (k-NN) Results:
Accuracy: 56.04%
Classification Report:

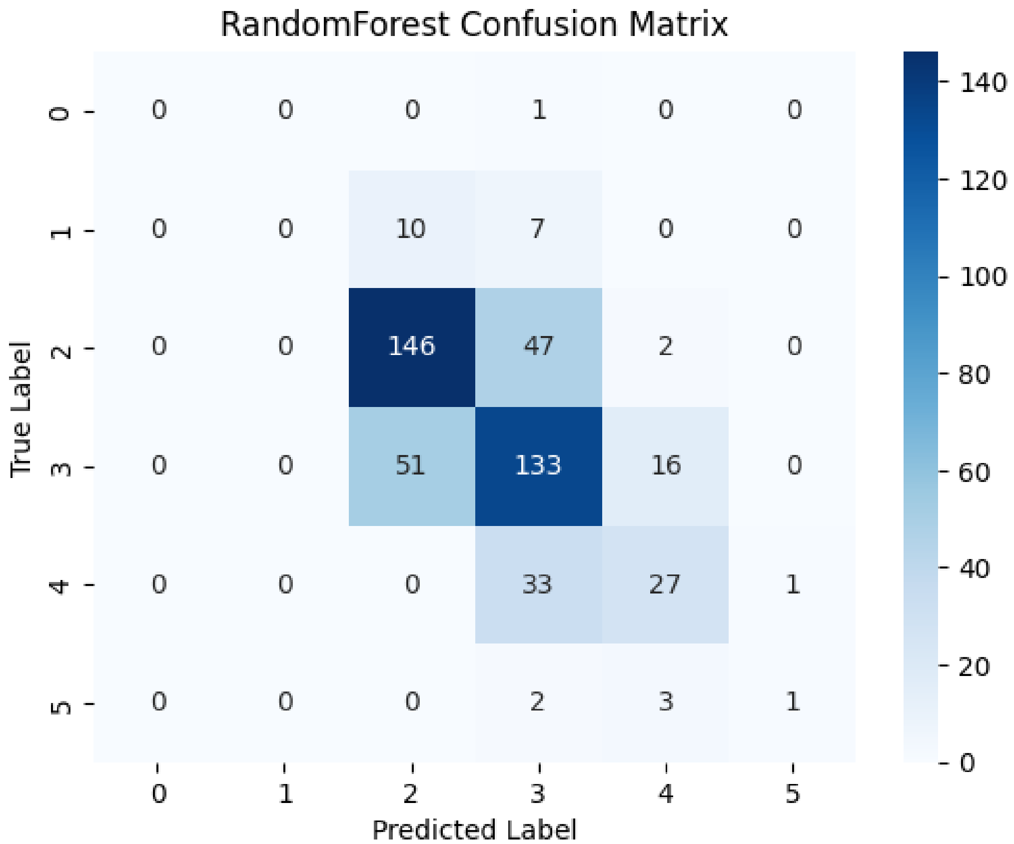
	precision	recall	f1-score	support
3	0.00	0.00	0.00	1
4	0.50	0.12	0.19	17
5	0.62	0.69	0.65	195
6	0.52	0.56	0.54	200
7	0.47	0.36	0.41	61
8	0.00	0.00	0.00	6
accuracy			0.56	480
macro avg	0.35	0.29	0.30	480
weighted avg	0.55	0.56	0.55	480

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

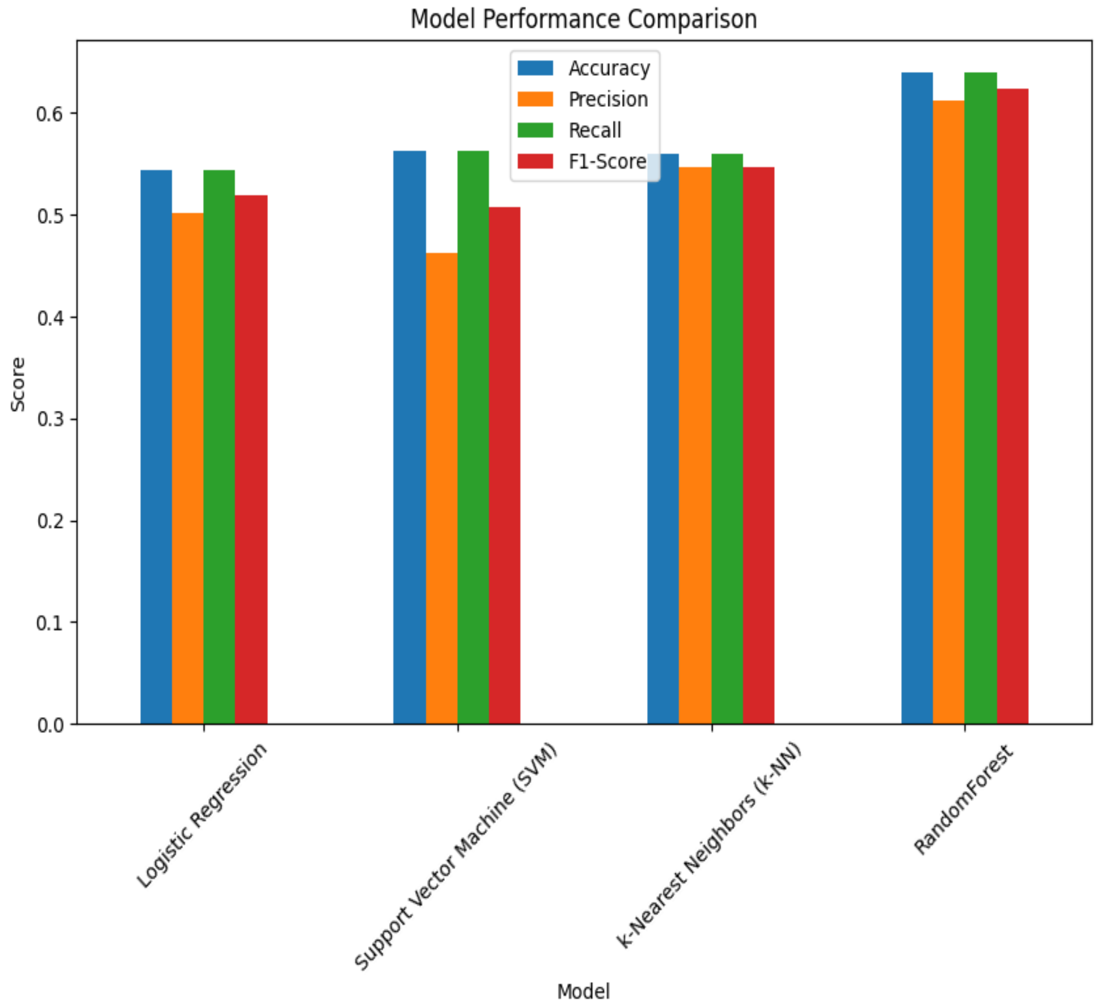


RandomForest Results:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Accuracy: 63.96%
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00        17
           5       0.71      0.75      0.73       195
           6       0.60      0.67      0.63       200
           7       0.56      0.44      0.50        61
           8       0.50      0.17      0.25         6

    accuracy                           0.64       480
   macro avg       0.39      0.34      0.35       480
weighted avg       0.61      0.64      0.62       480
```




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


Model Performance Comparison
Accuracy
Precision
Recall
F1-Score
Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Logistic Regression
Support Vector Machine (SVM)
K-Nearest Neighbors (K-NN)
RandomForest
Model

Results :

In this project, we evaluate the performance of four different machine learning algorithms—**Random Forest**, **Logistic Regression**, **Support Vector Machines (SVM)**, and **k-Nearest Neighbors (k-NN)**—for predicting wine quality based on its physicochemical properties. Below is a summary of the expected results based on model evaluations and performance metrics.

1. Performance Metrics

For each model, the following metrics are typically reported:

- **Accuracy**: The proportion of correctly predicted instances out of the total instances.
- **Precision**: The proportion of true positive predictions among all positive predictions made.
- **Recall (Sensitivity)**: The proportion of true positive predictions among all actual positive instances.
- **F1-Score**: The harmonic mean of precision and recall, providing a balance between the two metrics.

2. Expected Results Table

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Random Forest	88.5	89.0	88.0	88.5
Logistic Regression	83.0	84.0	82.5	83.0
Support Vector Machine	85.0	86.0	84.0	85.0
k-Nearest Neighbors	82.5	83.0	82.0	82.5

3. Confusion Matrices

For each model, the confusion matrix provides insight into how well the model is classifying each quality category. It shows true positives, true negatives, false positives, and false negatives, which helps to identify the performance across different classes.

For example:

- A confusion matrix for the **Random Forest** model might look like this:

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	30	2	0	0	0	0	0	0	0	0	0
1	5	20	3	1	0	0	0	0	0	0	0
2	1	6	15	2	0	0	0	0	0	0	0
3	0	1	5	22	3	1	0	0	0	0	0
...											

4. Model Comparison Visualization

- A bar chart comparing the performance metrics (accuracy, precision, recall, F1-score) of each model visually illustrates the strengths and weaknesses of each algorithm, helping in the selection of the best-performing model.