In [ ]:
```
# LIBRARIES

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from rich import print
from sklearn.preprocessing import PowerTransformer, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKF
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.linear_model import LogisticRegression
import pickle
from IPython.display import display
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, St
import xgboost as xgb
```

In [ ]:
```
# IMPORT DATASET

df = pd.read_csv(r'wine_data.csv')
print(df.head(3))



# HANDLE MISSING VALUES

# MISSING VALUES
print("[bold]Missing values before filling:[/bold]")
print(df.isnull().sum())
nullsum = df.isnull().sum().sum()
print("\n[bold]Total null values:[/bold]",nullsum)
print('\n\n')

# FILLING MISSING VALUES
# Fill missing values with median for all numeric columns
for col in df.select_dtypes(include='number').columns:
    if df[col].isnull().any():
        median_val = df[col].median()
        df[col] = df[col].fillna(median_val)

print("[bold]Missing values after filling:[/bold]")
print(df.isnull().sum())

nullsum = df.isnull().sum().sum()
print("\n[bold]Total null values:[/bold]", nullsum)
print('\n')
```

```
  type  fixed acidity  volatile acidity  citric acid  residual sugar  \
0  red           10.1              0.31         0.35             1.6
1  red            7.0              0.28         0.20            17.0
2  red            8.2              0.48         0.47             7.4

   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH  \
0      0.075                  9.0                 28.00    0.997  3.24
1      0.044                 47.0                 92.00    0.999  3.11
2      0.091                  2.7                149.26    0.994  3.45

   sulphates  alcohol  quality
0       0.83     11.2        7
1       0.38     10.8        5
2       0.74     10.9        8
```

Missing values before filling:

```
type                    0
fixed acidity           4
volatile acidity        1
citric acid             2
residual sugar          3
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    2
density                 2
pH                      1
sulphates               2
alcohol                 1
quality                 0
dtype: int64
```

Total null values: 20

Missing values after filling:

```
type                    0
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
```

Total null values: 0

In [ ]:

```
# DUPLICATE VALUES
```

```python
# DUPLICATES

df_duplicates = df[df.duplicated(keep="first")]
total_dup = df.duplicated().sum()
print(f"\n[bold]Total numer of duplicate rows:[/bold] {total_dup}")

# REMOVING DUPLICATES
df = df.drop_duplicates(keep="first")
print(f"[bold]Data shape after removing duplicates:[/bold] {df.shape}")
print("\n\n")
df['quality'] = df['quality'].astype(int)

# SAVING NEW CLEANED DATASET
cleaned_file_path = r'wine_cleaned.csv'
df.to_csv(cleaned_file_path, index=False)
```

Total numer of duplicate rows: 0

Data shape after removing duplicates: (7424, 13)

In [ ]:

```python
# IMPORT CLEANED WINES DATASET
df = pd.read_csv(r'wine_cleaned.csv')

# BINNING 'QUALITY' COLUMN

# 0-3: Low Quality, 4-7: Medium Quality, 8-10: High Quality
def bin_quality(val):
    if val <= 3:
        return 'Low Quality'
    elif val <= 7:
        return 'Medium Quality'
    else:
        return 'High Quality'

# Creating binned quality column
df['quality_binned'] = df['quality'].apply(bin_quality)

# Encode type and quality binned
type_map = {'red': 0, 'white': 1}
quality_map = {'Low Quality': 0, 'Medium Quality': 1, 'High Quality': 2}

df['type'] = df['type'].map(type_map)
df['quality_binned'] = df['quality_binned'].map(quality_map)

# Reverse maps for decoding later
type_map_rev = {v: k.title() for k, v in type_map.items()}  # {0: 'Red', 1: 'White'
quality_map_rev = {v: k for k, v in quality_map.items()}
```

In [ ]:

```python
# TRAIN TEST SPLIT (SCALED AND SMOTE)
X = df.drop(['quality', 'quality_binned'], axis=1)
y = df['quality_binned']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

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

print("[bold]Shape of scaled training data[/bold]")
print(X_train_scaled.shape)
print("[bold]Shape of scaled testing data[/bold]")
print(X_test_scaled.shape)
print('\n\n')

# Apply SMOTE

print("[bold]Before SMOTE:[/bold]")
print(pd.Series(y_train.map(quality_map_rev)).value_counts().sort_index())
print('\n\n')

smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

print("[bold]After SMOTE:[/bold]")
print(pd.Series(y_train_smote.map(quality_map_rev)).value_counts().sort_index())

# Map test and train labels
y_train_smote_named = y_train_smote.map(quality_map_rev)
y_test_named = y_test.map(quality_map_rev)
```

Shape of scaled training data

(5939, 12)

Shape of scaled testing data

(1485, 12)

Before SMOTE:

```
quality_binned
High Quality      1942
Low Quality       1851
Medium Quality    2146
Name: count, dtype: int64
```

After SMOTE:

```
quality_binned
High Quality      2146
Low Quality       2146
Medium Quality    2146
Name: count, dtype: int64
```

In [ ]:

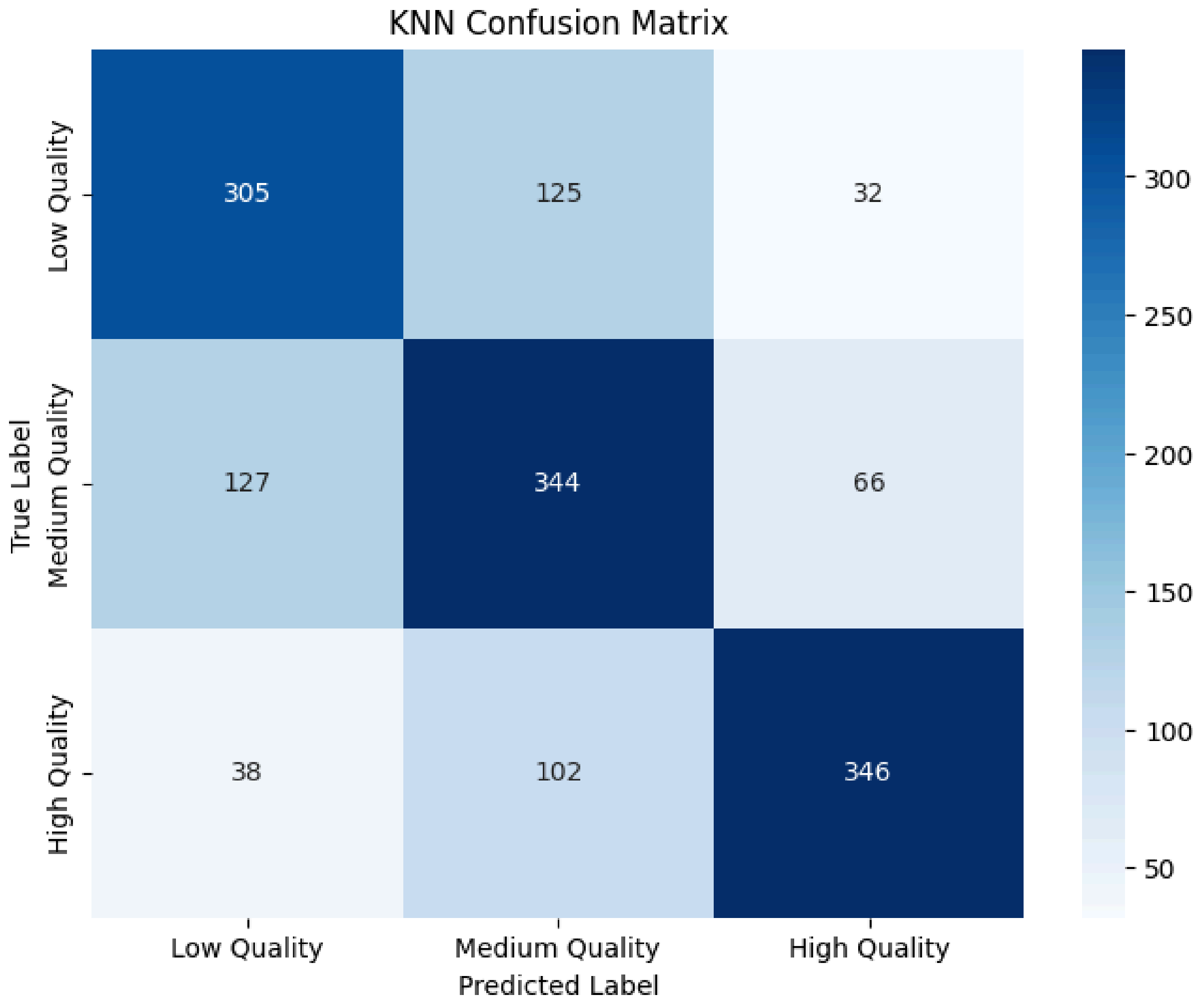
```
# KNN CLASSIFICATION

# KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
```

```python
knn.fit(X_train_smote, y_train_smote)

# Predict on test set
y_pred = knn.predict(X_test_scaled)

# Classification Report
report = classification_report(y_test, y_pred, output_dict=True)

# Convert report dict to DataFrame
report_df = pd.DataFrame(report).transpose()

# Convert metrics to percentage and round
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

# Convert support to int
report_df['support'] = report_df['support'].astype(int)

# Rename index using quality_map_rev for class labels
rename_map = {str(k): v for k, v in quality_map_rev.items()}
report_df.rename(index=rename_map, inplace=True)

print("[bold]KNN Classification Report (%):[/bold]\n")
print(report_df)
print("\n[bold]Overall Accuracy:[/bold]", f"{report_df.loc['accuracy', 'precision']

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[quality_map_rev[i] for i in sorted(y_test.unique())],
            yticklabels=[quality_map_rev[i] for i in sorted(y_test.unique())])
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# K-Fold Cross Validation (on SMOTE data)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(knn, X_train_smote, y_train_smote, cv=cv, scoring='accu

# Convert to percentage and round
cv_scores_percent = [round(score * 100, 2) for score in cv_scores]

print('\n\n')
print("[bold]K-Fold Cross Validation Scores (% Accuracy):[/bold]")
for i, score in enumerate(cv_scores_percent, 1):
    print(f"Fold {i}: {score}%")
print(f"\n[bold]Average CV Accuracy:[/bold] {round(np.mean(cv_scores_percent), 2)}%
```

KNN Classification Report (%):

	precision	recall	f1-score	support
Low Quality	64.89	66.02	65.45	462
Medium Quality	60.25	64.06	62.09	537
High Quality	77.93	71.19	74.41	486
accuracy	67.00	67.00	67.00	0
macro avg	67.69	67.09	67.32	1485
weighted avg	67.48	67.00	67.17	1485

Overall Accuracy: 67.00%



K-Fold Cross Validation Scores (% Accuracy):

Fold 1: 68.01%

Fold 2: 67.78%

Fold 3: 69.8%

Fold 4: 69.15%

Fold 5: 68.45%

Average CV Accuracy: 68.64%

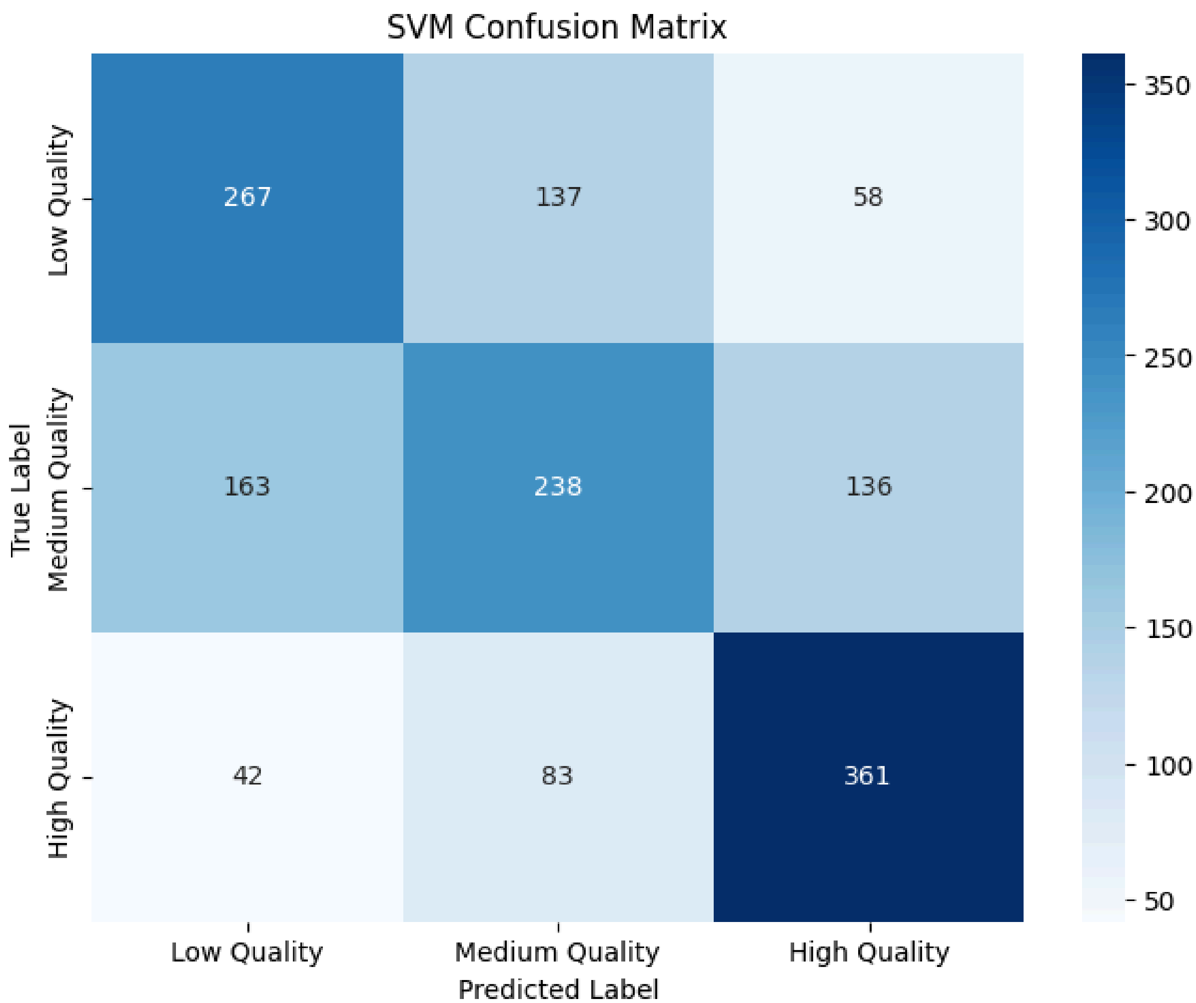
In [ ]:

```
# SVM CLASSIFICATION

# Initialize SVM
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_smote, y_train_smote)

# Predict on test set
y_pred = svm.predict(X_test_scaled)

# Classification report
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Format precision, recall, f1-score as %
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

report_df['support'] = report_df['support'].astype(int)

# Decode class labels
report_df.rename(index={str(k): v for k, v in quality_map_rev.items()}, inplace=Tru

# Reorder for display
order = list(quality_map_rev.values()) + ['macro avg', 'weighted avg', 'accuracy']
report_df = report_df.loc[order]

print("[bold]SVM Classification Report (%):[/bold]\n")
print(report_df)

print("\n[bold]Overall Accuracy:[/bold]", f"{report_df.loc['accuracy', 'precision']

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[quality_map_rev[i] for i in sorted(y_test.unique())],
            yticklabels=[quality_map_rev[i] for i in sorted(y_test.unique())])
plt.title("SVM Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# K-Fold CV on SMOTE data
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(svm, X_train_smote, y_train_smote, cv=cv, scoring='accu
cv_scores_percent = [round(score * 100, 2) for score in cv_scores]

print("\n[bold]SVM K-Fold Cross-Validation Scores (% Accuracy):[/bold]")
for i, score in enumerate(cv_scores_percent, 1):
    print(f"Fold {i}: {score}%")
print(f"\n[bold]Average CV Accuracy:[/bold] {round(np.mean(cv_scores_percent), 2)}%
```

SVM Classification Report (%):

	precision	recall	f1-score	support
Low Quality	56.57	57.79	57.17	462
Medium Quality	51.97	44.32	47.84	537
High Quality	65.05	74.28	69.36	486
macro avg	57.86	58.80	58.12	1485
weighted avg	57.68	58.32	57.79	1485
accuracy	58.32	58.32	58.32	0

Overall Accuracy: 58.32%



SVM K-Fold Cross-Validation Scores (% Accuracy):

Fold 1: 57.14%

Fold 2: 59.16%

Fold 3: 56.44%

Fold 4: 57.73%

Fold 5: 58.66%

Average CV Accuracy: 57.83%

In [ ]:

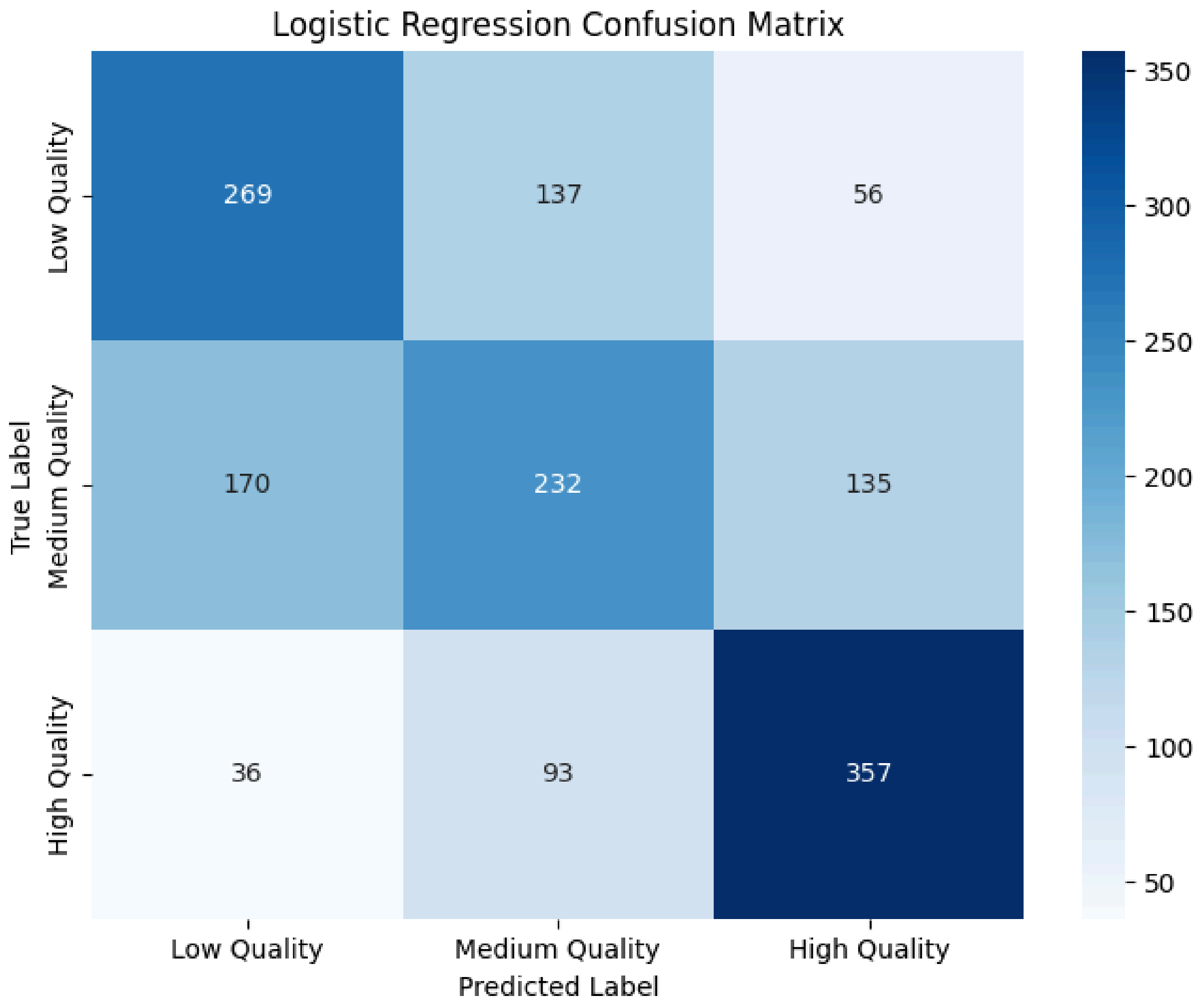
```
# LOGISTIC REGRESSION

# Train Logistic Regression
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train_smote, y_train_smote)
```

```
# Predict on test data
y_pred = logreg.predict(X_test_scaled)

# Classification Report
report = classification_report(y_test, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Format precision, recall, f1-score as %
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

# Convert support to int
report_df['support'] = report_df['support'].astype(int)

# Rename index using quality_map_rev (e.g., 0 → Low Quality)
report_df.rename(index={str(k): v for k, v in quality_map_rev.items()}, inplace=Tru

print("[bold]Logistic Regression Classification Report (%):[/bold]\n")
print(report_df)

print(f"\n[bold]Overall Accuracy:[/bold] {report_df.loc['accuracy', 'precision']:.2

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[quality_map_rev[i] for i in sorted(y_test.unique())],
            yticklabels=[quality_map_rev[i] for i in sorted(y_test.unique())])
plt.title("Logistic Regression Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# K-Fold Cross-Validation on SMOTE-applied training data
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(logreg, X_train_smote, y_train_smote, cv=cv, scoring='a
cv_scores_percent = [round(score * 100, 2) for score in cv_scores]

print("\n[bold]K-Fold Cross Validation Scores (% Accuracy):[/bold]")
for i, score in enumerate(cv_scores_percent, 1):
    print(f"Fold {i}: {score}%")
print(f"\n[bold]Average CV Accuracy:[/bold] {round(np.mean(cv_scores_percent), 2)}%
```

Logistic Regression Classification Report (%):

	precision	recall	f1-score	support
Low Quality	56.63	58.23	57.42	462
Medium Quality	50.22	43.20	46.45	537
High Quality	65.15	73.46	69.05	486
accuracy	57.78	57.78	57.78	0
macro avg	57.33	58.29	57.64	1485
weighted avg	57.10	57.78	57.26	1485

Overall Accuracy: 57.78%




```
K-Fold Cross Validation Scores (% Accuracy):
Fold 1: 56.99%
Fold 2: 58.39%
Fold 3: 57.22%
Fold 4: 57.81%
Fold 5: 58.04%
Average CV Accuracy: 57.69%
```

In [ ]:

```
# TRAIN TEST (UNSCALED)

# Features (drop target columns)
X_unscaled = df.drop(['quality', 'quality_binned'], axis=1)
y = df['quality_binned']

# Train-test split (unscaled)
X_train_unscaled, X_test_unscaled, y_train_unscaled, y_test_unscaled = train_test_s
    X_unscaled, y, test_size=0.2, random_state=42, stratify=y, shuffle=True
)
```

In [ ]:

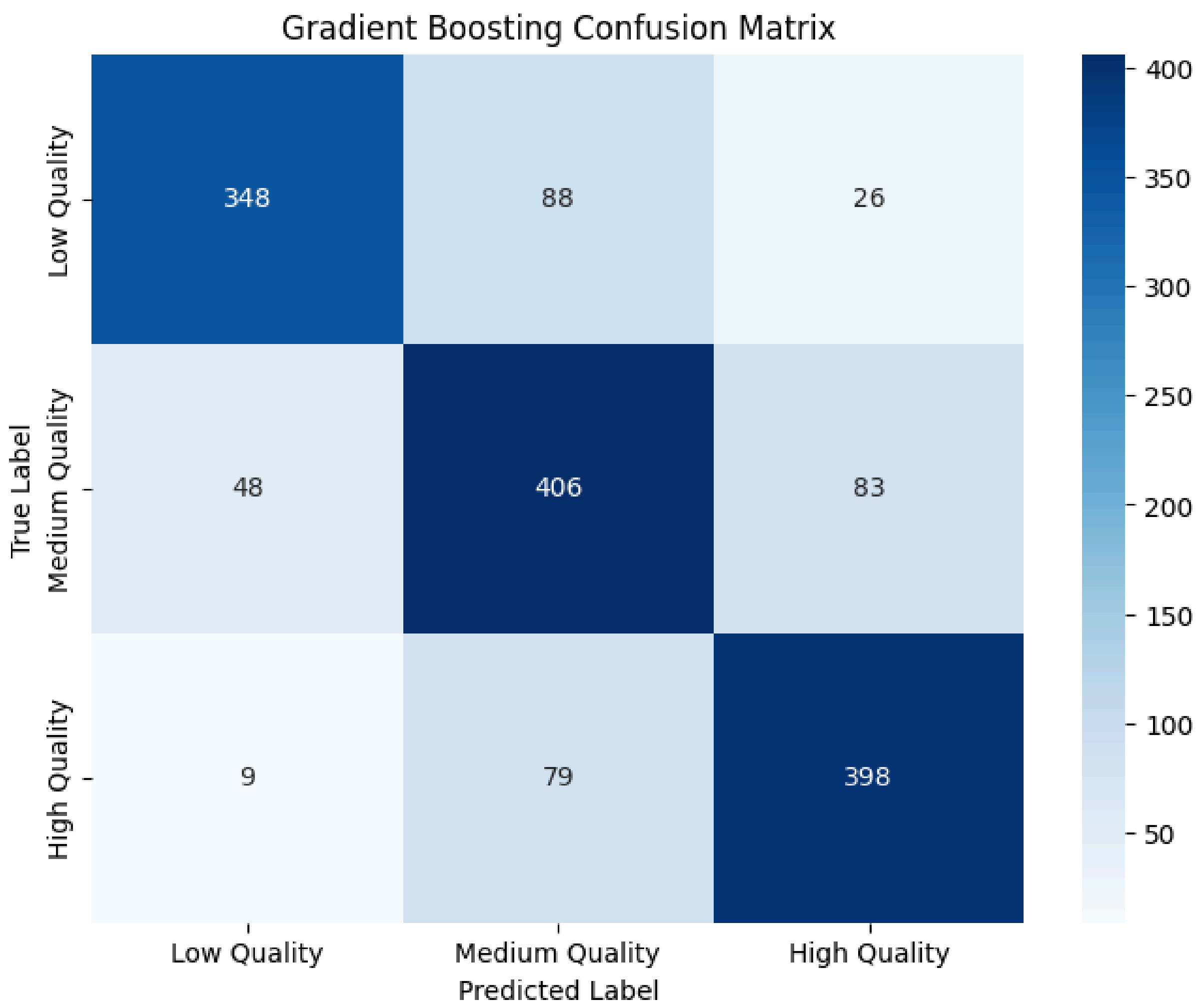
```
# GRADIENT BOOSTING

# Train Gradient Boosting
```

```
gbc = GradientBoostingClassifier(random_state=42)
gbc.fit(X_train_unscaled, y_train_unscaled)

# Predict on test set
y_pred = gbc.predict(X_test_unscaled)

# Classification report
report = classification_report(y_test_unscaled, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Format as %
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

report_df['support'] = report_df['support'].astype(int)

# Rename index for readability
report_df.rename(index={str(k): v for k, v in quality_map_rev.items()}, inplace=Tru

print("[bold]Gradient Boosting Classification Report (%):[/bold]\n")
print(report_df)
print("\n[bold]Overall Accuracy:[/bold]", f"{report_df.loc['accuracy', 'precision']


# Confusion Matrix
conf_matrix = confusion_matrix(y_test_unscaled, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=[quality_map_rev[i] for i in sorted(y_test_unscaled.unique(
            yticklabels=[quality_map_rev[i] for i in sorted(y_test_unscaled.unique(
plt.title("Gradient Boosting Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# K-Fold Cross Validation (Stratified, 5 folds)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(gbc, X_unscaled, y, cv=cv, scoring='accuracy')

cv_scores_percent = [round(score * 100, 2) for score in cv_scores]

# Print K-Fold results
print("\n[bold]K-Fold Cross Validation Scores (% Accuracy):[/bold]")
for i, score in enumerate(cv_scores_percent, 1):
    print(f"Fold {i}: {score}%")

# Print average accuracy
print(f"\n[bold]Average CV Accuracy:[/bold] {round(np.mean(cv_scores_percent), 2)}%
```

Gradient Boosting Classification Report (%):

	precision	recall	f1-score	support
Low Quality	85.93	75.32	80.28	462
Medium Quality	70.86	75.61	73.15	537
High Quality	78.50	81.89	80.16	486
accuracy	77.58	77.58	77.58	0
macro avg	78.43	77.61	77.86	1485
weighted avg	78.05	77.58	77.66	1485

Overall Accuracy: 77.58%



K-Fold Cross Validation Scores (% Accuracy):

Fold 1: 75.89%

Fold 2: 75.22%

Fold 3: 76.23%

Fold 4: 74.07%

Fold 5: 75.94%

Average CV Accuracy: 75.47%

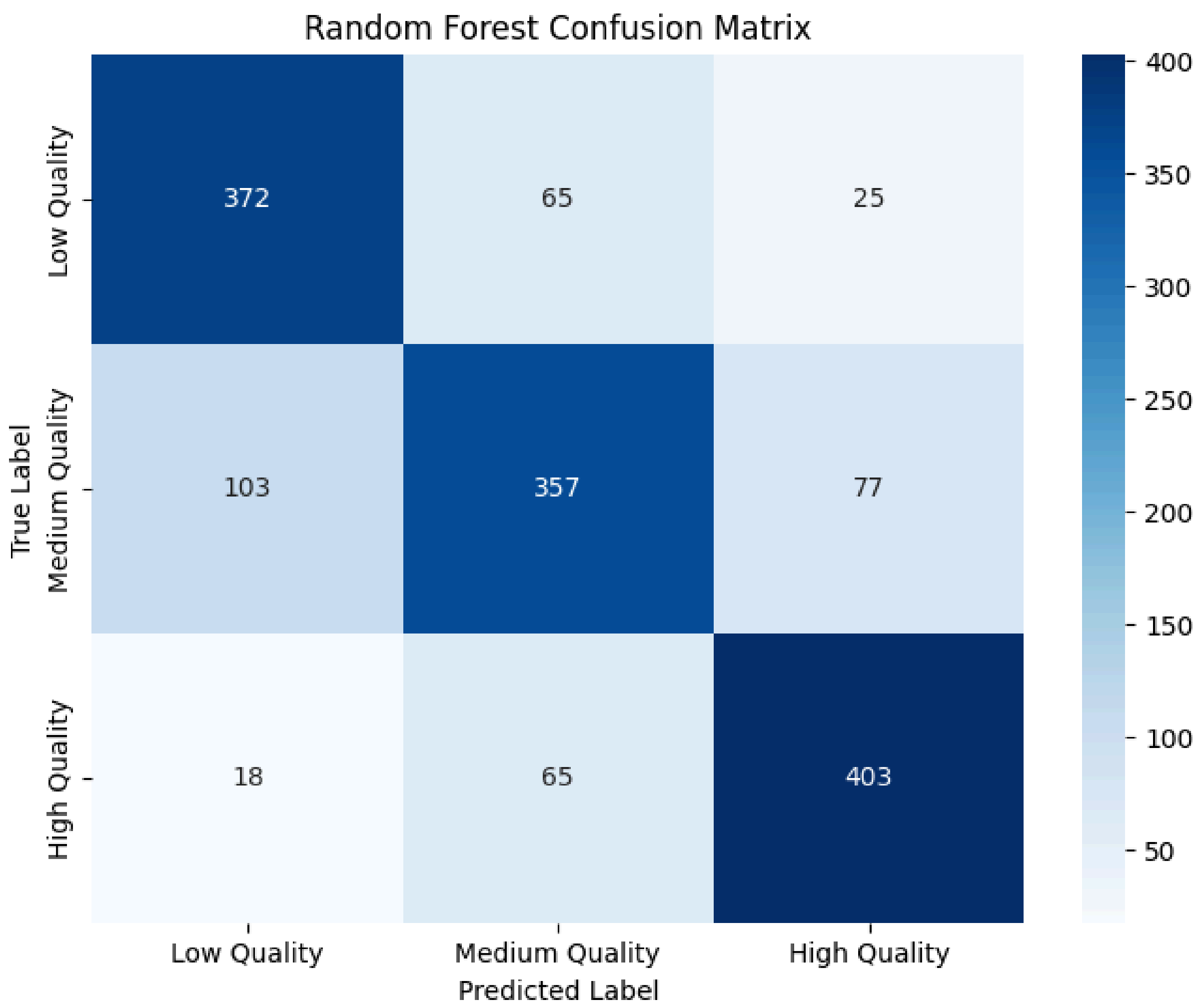
In [ ]:
```
# RANDOM FOREST CLASSIFICATION


# Initialize Random Forest
```

```
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
rf_clf.fit(X_train_unscaled, y_train_unscaled)

# Predict on test set
y_pred = rf_clf.predict(X_test_unscaled)

# Classification report
report = classification_report(y_test_unscaled, y_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()

# Format as %
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

report_df['support'] = report_df['support'].astype(int)

# Rename index for readability
report_df.rename(index={str(k): v for k, v in quality_map_rev.items()}, inplace=Tru

print("[bold]Random Forest Classification Report (%):[/bold]\n")
print(report_df)
print("\n[bold]Overall Accuracy:[/bold]", f"{report_df.loc['accuracy', 'precision']


# Confusion Matrix
cm = confusion_matrix(y_test_unscaled, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True,
            fmt='d',
            cmap='Blues',
            xticklabels=[quality_map_rev[i] for i in sorted(y_test_unscaled.unique(
            yticklabels=[quality_map_rev[i] for i in sorted(y_test_unscaled.unique(
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()


# K-Fold Cross Validation (Stratified, 5 folds)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(rf_clf, X_unscaled, y, cv=cv, scoring='accuracy')

# Print K-Fold results
print("\n[bold]K-Fold Cross Validation Scores (% Accuracy):[/bold]")
for i, score in enumerate(cv_scores, 1):
    print(f"Fold {i}: {round(score * 100, 2)}%")

# Print average accuracy
print(f"\n[bold]Average CV Accuracy:[/bold] {round(np.mean(cv_scores) * 100, 2)}%")
```

Random Forest Classification Report (%):

	precision	recall	f1-score	support
Low Quality	75.46	80.52	77.91	462
Medium Quality	73.31	66.48	69.73	537
High Quality	79.80	82.92	81.33	486
accuracy	76.23	76.23	76.23	0
macro avg	76.19	76.64	76.32	1485
weighted avg	76.10	76.23	76.07	1485

Overall Accuracy: 76.23%



K-Fold Cross Validation Scores (% Accuracy):

Fold 1: 73.6%

Fold 2: 75.02%

Fold 3: 75.22%

Fold 4: 73.94%

Fold 5: 76.75%

Average CV Accuracy: 74.91%

In [ ]:
```
# XG BOOST


# Initialize the XGBoost classifier
```

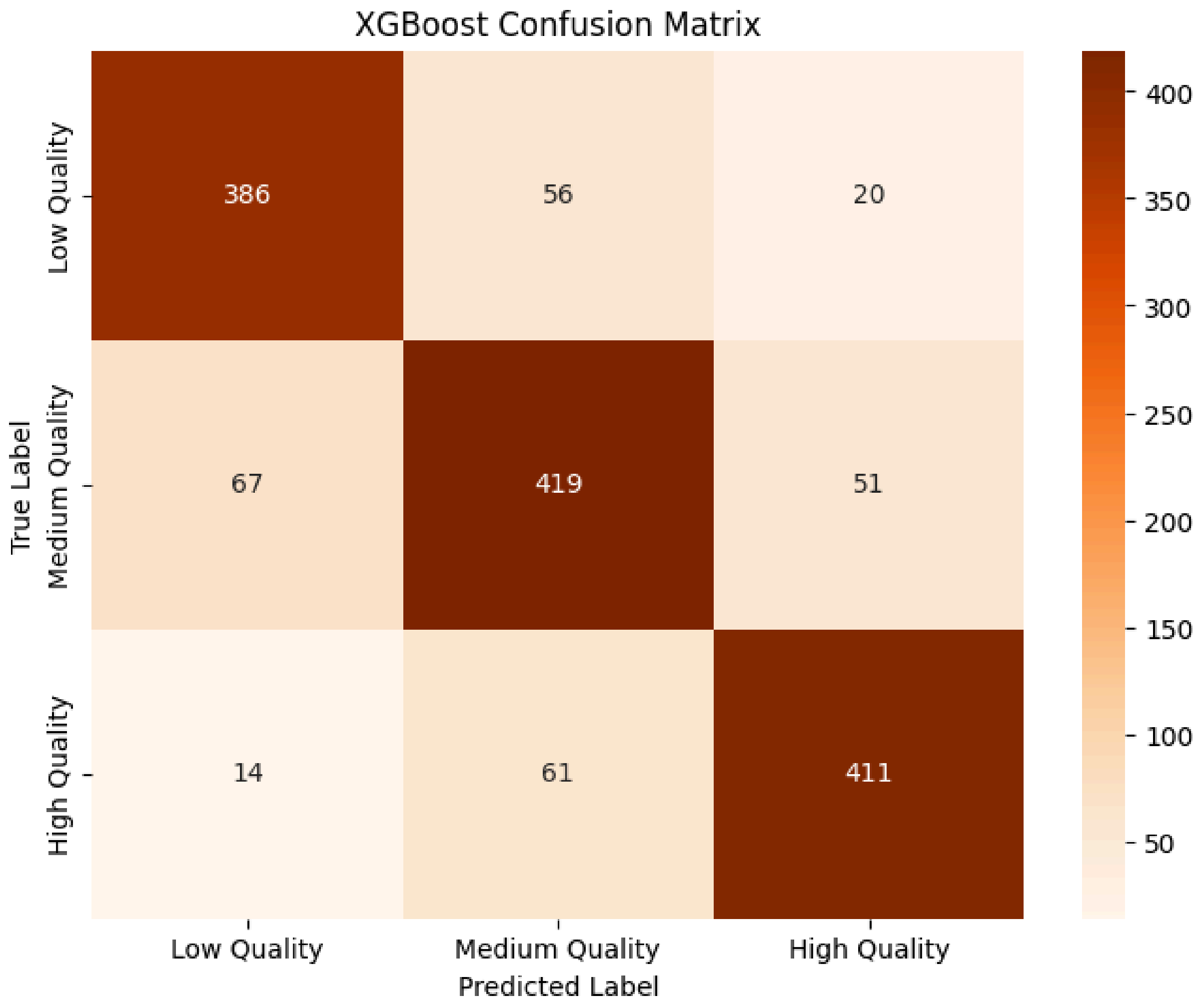
```
xgb_clf = xgb.XGBClassifier(
    objective='multi:softmax',  # For multi-class classification
    num_class=3,                # Number of classes
    eval_metric='mlogloss',
    random_state=42
)

# Fit the model
xgb_clf.fit(X_train_unscaled, y_train_unscaled)

# Predict on test set
y_pred = xgb_clf.predict(X_test_unscaled)

# Classification report
report_dict = classification_report(
    y_test_unscaled,
    y_pred,
    target_names=[quality_map_rev[i] for i in sorted(quality_map_rev.keys())],
    output_dict=True
)

report_df = pd.DataFrame(report_dict).transpose()

# Format as % for precision, recall, f1-score (except 'support')
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

# Convert support to integer
report_df['support'] = report_df['support'].astype(int)

print("[bold]XGBoost Classification Report:[/bold]\n")
print(report_df)
print("\n[bold]Overall Accuracy:[/bold]", f"{report_df.loc['accuracy', 'precision']


# Confusion matrix plot
conf_matrix = confusion_matrix(y_test_unscaled, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Oranges',
            xticklabels=[quality_map_rev[i] for i in sorted(quality_map_rev.keys())
            yticklabels=[quality_map_rev[i] for i in sorted(quality_map_rev.keys())
plt.title("XGBoost Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# K-Fold Cross Validation (Stratified, 5 folds)
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = cross_val_score(xgb_clf, X_unscaled, y, cv=kf, scoring='accuracy')

# Convert K-Fold cross-validation scores to percentages
cv_scores_percent = [round(score * 100, 2) for score in cv_scores]

# Print K-Fold results in desired format
print("\n[bold]K-Fold Cross Validation Scores (% Accuracy):[/bold]")
```

```python
for i, score in enumerate(cv_scores_percent, 1):
    print(f"Fold {i}: {score}%")

# Print average accuracy
print(f"\n[bold]Average CV Accuracy:[/bold] {round(np.mean(cv_scores_percent), 2)}%
print('\n\n')

# Get feature importances (gain, weight, cover - default is 'weight')
xgb_importances = xgb_clf.feature_importances_

# DataFrame and plot as above
feat_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': xgb_importances
}).sort_values(by='Importance', ascending=False)

# Round importance values to 2 decimal places
feat_importance_df['Importance'] = feat_importance_df['Importance'].round(2)

print("[bold]Feature Importance Ranking[\bold]")
print(feat_importance_df)

# Plotting Feature importance
plt.figure(figsize=(9,6))
sns.barplot(data=feat_importance_df, x='Importance', y='Feature',hue=None)
plt.title('Feature Importance Ranking - XGBoost')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```

XGBoost Classification Report:

	precision	recall	f1-score	support
Low Quality	82.66	83.55	83.10	462
Medium Quality	78.17	78.03	78.10	537
High Quality	85.27	84.57	84.92	486
accuracy	81.89	81.89	81.89	0
macro avg	82.03	82.05	82.04	1485
weighted avg	81.89	81.89	81.89	1485

Overall Accuracy: 81.89%



K-Fold Cross Validation Scores (% Accuracy):

Fold 1: 78.79%

Fold 2: 80.34%

Fold 3: 79.53%

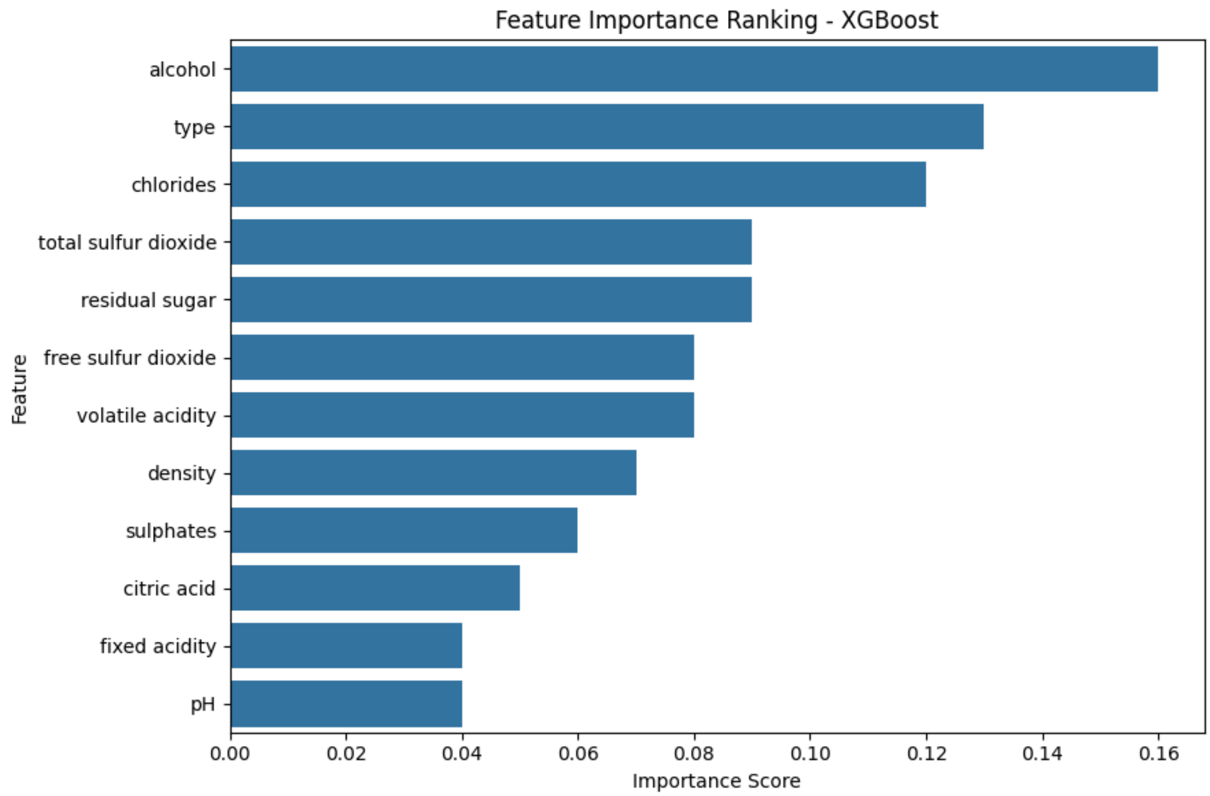
Fold 4: 80.13%

Fold 5: 81.0%

Average CV Accuracy: 79.96%

Feature Importance Ranking[old]

```
                   Feature  Importance
11                 alcohol        0.16
0                     type        0.13
5                chlorides        0.12
7     total sulfur dioxide        0.09
4           residual sugar        0.09
6      free sulfur dioxide        0.08
2         volatile acidity        0.08
8                  density        0.07
10               sulphates        0.06
3              citric acid        0.05
1            fixed acidity        0.04
9                       pH        0.04
```



In [ ]:
```
# STACKING

# Define base models
estimators = [
    ('rf', RandomForestClassifier(random_state=42)),
    ('gb', GradientBoostingClassifier(random_state=42)),
    ('xgb', xgb.XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', rand
]

# Meta-model (final estimator)
final_estimator = LogisticRegression(max_iter=1000, random_state=42)

# Define stacking classifier
stacking_clf = StackingClassifier(
    estimators=estimators,
    final_estimator=final_estimator,
    cv=5,
    n_jobs=-1,
```

```
    passthrough=False
)

# Fit on training data (use unscaled)
stacking_clf.fit(X_train_unscaled, y_train_unscaled)

# Predict on test set
y_pred = stacking_clf.predict(X_test_unscaled)

# Classification Report
report = classification_report(
    y_test_unscaled,
    y_pred,
    target_names=[quality_map_rev[i] for i in sorted(quality_map_rev.keys())],
    output_dict=True
)

report_df = pd.DataFrame(report).transpose()

# Format precision, recall, f1-score as percentages
metrics = ['precision', 'recall', 'f1-score']
for metric in metrics:
    report_df.loc[report_df.index != 'support', metric] = report_df.loc[report_df.i

# Convert support to integer
report_df['support'] = report_df['support'].astype(int)

# Print final report
print("\nStacking Classifier Classification Report (%):\n")
print(report_df)

print("\n[bold]Overall Accuracy:[/bold]", f"{report_df.loc['accuracy', 'precision']

# Confusion Matrix
conf_matrix = confusion_matrix(y_test_unscaled, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Purples',
            xticklabels=[quality_map_rev[i] for i in sorted(quality_map_rev.keys())
            yticklabels=[quality_map_rev[i] for i in sorted(quality_map_rev.keys())
plt.title("Stacking Classifier Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Stacking Classifier Classification Report (%):

	precision	recall	f1-score	support
Low Quality	83.99	82.90	83.44	462
Medium Quality	77.11	78.40	77.75	537
High Quality	84.68	84.16	84.42	486
accuracy	81.68	81.68	81.68	0
macro avg	81.93	81.82	81.87	1485
weighted avg	81.73	81.68	81.70	1485

Overall Accuracy: 81.68%

In [ ]:

```
# SAVING TRAINED STACKED MODEL (ONLY ENSEMBLE TECHNIQUES)

# Save the trained stacked model
with open(r'stacked_model.pkl', 'wb') as f:
    pickle.dump(stacking_clf, f)
```