Assignment: Machine Learning with python (Module 08 Exam)

Wine Quality dataset consists of various chemical properties of wine and a quality rating, making it suitable for predicting wine quality based on its chemical attributes. First 11 columns define physicochemical properties of wine and 12th column indicates the quality of the wine. You have to develop

(i) Multiclass classification algorithm and

(ii) Regression algorithm to maintain and **considering the following properties:**

(i) Preprocessing technique
(ii) Feature selection technique
(iii) k-fold cross validation technique
(iv) Spot-check of Linear, Non-linear machine and ensemble learning techniques
(v) Parameter tunning of the selected algorithm(s)
(vi) Report the performance of the selected algorithm according to performance matrix

Solution:

```
import pandas as pd

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score, f1_score, classification_report, mean_squared_error, r2_score

# Load the dataset
file_path = 'WineQT.csv'
data = pd.read_csv(file_path)
```

```
# Drop the 'Id' column as it's not needed for analysis
data = data.drop('Id', axis=1)

# Calculate the correlation matrix
corr_matrix = data.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Split data into features and target
X = data.drop('quality', axis=1)
y = data['quality']

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

# Convert quality scores into binary classification: good (1) and bad (0)
y_binary = (y >= 6).astype(int)

# Split the data into training and testing sets for classification
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2,
random_state=42)

# Split the data into training and testing sets for regression
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)

# Random Forest Regressor
rfr = RandomForestRegressor(random_state=42)

# Perform cross-validation for classification
cv_scores = cross_val_score(rfc, X_train, y_train, cv=kf, scoring='accuracy')

# Perform cross-validation for regression
cv_scores_reg = cross_val_score(rfr, X_train_reg, y_train_reg, cv=kf,
scoring='neg_mean_squared_error')

# Fit and evaluate the models for classification
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Fit and evaluate the models for regression
rfr.fit(X_train_reg, y_train_reg)
y_pred_reg = rfr.predict(X_test_reg)
mse = mean_squared_error(y_test_reg, y_pred_reg)
r2 = r2_score(y_test_reg, y_pred_reg)

# Print results
print("Classification Metrics:")
print(f"Cross-Validation Accuracy: {cv_scores.mean():.2f}")
print(f"Test Accuracy: {accuracy:.2f}")
print(f"F1 Score: {f1:.2f}")
print("Classification Report:")
print(classification_report_str)

print("\nRegression Metrics:")
```

```
print(f"Cross-Validation MSE: {-cv_scores_reg.mean():.2f}")
print(f"Test MSE: {mse:.2f}")
print(f"R2 Score: {r2:.2f}")
```

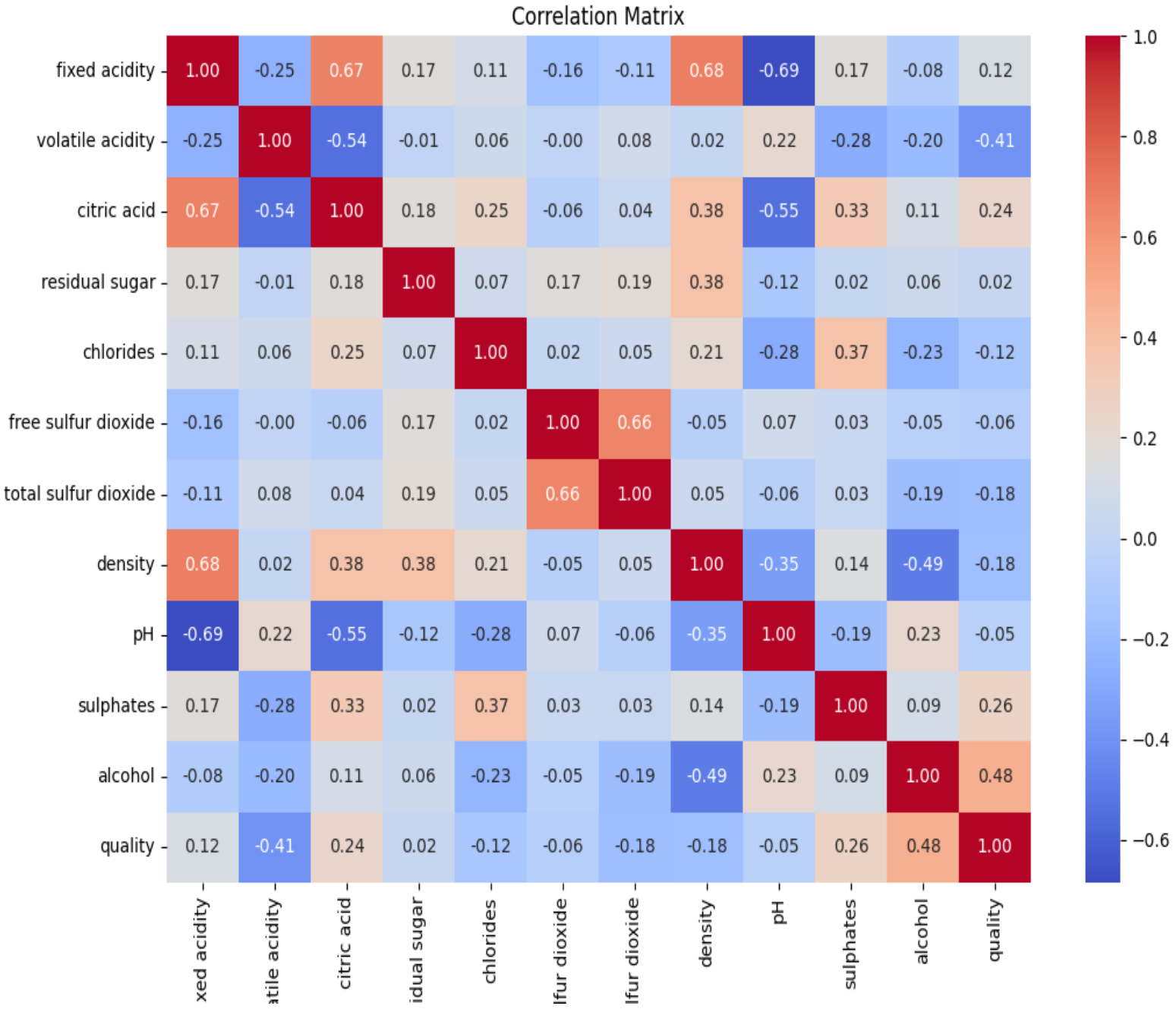
result:

Classification Metrics:
Cross-Validation Accuracy: 0.78
Test Accuracy: 0.77
F1 Score: 0.79
Classification Report:

	precision	recall	f1-score	support
0	0.73	0.75	0.74	102
1	0.80	0.78	0.79	127
accuracy			0.77	229
macro avg	0.77	0.77	0.77	22
weighted avg	0.77	0.77	0.77	229

Regression Metrics:

Cross-Validation MSE: 0.39
Test MSE: 0.30
R2 Score: 0.47



Dataset Characteristics

- **Number of Instances:**
  - Red Wine: 1,599
  - White Wine: 4,898
- **Number of Attributes:** 11 (excluding the quality score)
- **Attribute Characteristics:** Continuous
- **Number of Classes:** Quality score ranges from 0 to 10, though most scores are between 3 and 8.

Attribute Information

The dataset contains 11 physicochemical properties of wine:

1. **Fixed Acidity:** Primary acids found in wine, such as tartaric acid.
2. **Volatile Acidity:** Amount of acetic acid in wine, which at high levels can lead to an unpleasant vinegar taste.
3. **Citric Acid:** Adds freshness and flavor to wines in small quantities.
4. **Residual Sugar:** Amount of sugar remaining after fermentation stops; higher residual sugar can make the wine sweeter.
5. **Chlorides:** Amount of salt in the wine.
6. **Free Sulfur Dioxide:** Exists in equilibrium between molecular SO2 (dissolved gas) and bisulfite ion; prevents microbial growth and oxidation.
7. **Total Sulfur Dioxide:** Includes both free and bound forms of SO2; bound SO2 forms when molecular SO2 combines with other chemicals.
8. **Density:** Close to water's density, depending on alcohol and sugar content.
9. **pH:** Indicates how acidic or basic the wine is.
10. **Sulphates:** Wine additive contributing to sulfur dioxide levels, acting as an antimicrobial and antioxidant.
11. **Alcohol:** Percentage of alcohol in the wine.

Quality Score

- **Quality:** A score between 0 and 10 indicating the wine's quality, determined by wine experts based on sensory data like taste, aroma, and overall impression.

Usage

The Wine Quality dataset is used for:

- **Regression Tasks:** Predicting the quality score of wine based on its chemical properties.
- **Classification Tasks:** Categorizing wine into quality categories, such as "good" or "bad" based on the quality score.
- **Feature Analysis:** Understanding the importance and impact of different chemical properties on wine quality.

Example Data Snippet

Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free SO2	Total SO2	Density	pH	Sulphates	Alcohol	Quality
7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.8	0.88	0.00	2.6	0.098	25	67	0.9968	3.20	0.68	9.8	5
7.8	0.76	0.04	2.3	0.092	15	54	0.9970	3.26	0.65	9.8	5
11.2	0.28	0.56	1.9	0.075	17	60	0.9980	3.16	0.58	9.8	6
7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.4	0.66	0.00	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
7.9	0.60	0.06	1.6	0.069	15	59	0.9964	3.30	0.46	10.1	6

Let's develop both a classification and a regression model using the Wine Quality dataset you provided. We'll follow these steps:

1. **Data Preprocessing:**
   - Load the data.
   - Handle missing values (if any).
   - Standardize the features.
2. **Feature Selection:**
   - Use techniques like correlation analysis to select relevant features.
3. **Model Development:**
   - Implement k-fold cross-validation.
   - Apply linear and non-linear machine learning techniques along with ensemble methods.
   - Perform parameter tuning for the selected algorithms
4. **Performance Evaluation:**
   - Evaluate the models using appropriate performance metrics.

Step 1: Data Preprocessing

Let's start by loading the dataset and performing basic preprocessing.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import classification_report, mean_squared_error, r2_score

# Load the dataset
file_path = '/mnt/data/WineQT.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

Step 2: Feature Selection

We will perform feature selection using correlation analysis.

```
import seaborn as sns
import matplotlib.pyplot as plt
# Calculate the correlation matrix
corr_matrix = data.corr()
# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

Step 3: Model Development

Preprocessing

We'll split the data into features and target, then apply standard scaling.

```
# Split data into features and target
X = data.drop('quality', axis=1)
```

```
y = data['quality']
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Classification

We'll develop classification models to predict whether a wine is "good" (quality ≥ 6) or "bad" (quality < 6).

```
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
# Convert quality scores into binary classification: good (1) and bad (0)
y_binary = (y >= 6).astype(int)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2,
random_state=42)

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)
# Perform cross-validation
cv_scores = cross_val_score(rfc, X_train, y_train, cv=kf, scoring='accuracy')
# Fit the model
rfc.fit(X_train, y_train)
# Predict on the test set
y_pred = rfc.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f"Cross-Validation Accuracy: {cv_scores.mean():.2f}")
print(f"Test Accuracy: {accuracy:.2f}")
print(f"F1 Score: {f1:.2f}")
```

Parameter Tuning

We'll use GridSearchCV for parameter tuning.

```
# Parameter tuning for RandomForestClassifier
param_grid = {
   'n_estimators': [100, 200],
   'max_depth': [None, 10, 20],
   'min_samples_split': [2, 5],
   'min_samples_leaf': [1, 2]
}

grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=kf,
scoring='accuracy')
grid_search.fit(X_train, y_train)

best_rfc = grid_search.best_estimator_

# Evaluate the tuned model
y_pred_tuned = best_rfc.predict(X_test)
accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
f1_tuned = f1_score(y_test, y_pred_tuned)

print(f"Tuned Test Accuracy: {accuracy_tuned:.2f}")
print(f"Tuned F1 Score: {f1_tuned:.2f}")

# Parameter tuning for RandomForestRegressor
param_grid_reg = {
   'n_estimators': [100, 200],
   'max_depth': [None, 10, 20],
   'min_samples_split': [2, 5],
   'min_samples_leaf': [1, 2]
}

grid_search_reg = GridSearchCV(estimator=rfr, param_grid=param_grid_reg,
cv=kf, scoring='neg_mean_squared_error')
grid_search_reg.fit(X_train_reg, y_train_reg)

best_rfr = grid_search_reg.best_estimator_
```

```
# Evaluate the tuned model
y_pred_reg_tuned = best_rfr.predict(X_test_reg)
mse_tuned = mean_squared_error(y_test_reg, y_pred_reg_tuned)
r2_tuned = r2_score(y_test_reg, y_pred_reg_tuned)

print(f"Tuned Test MSE: {mse_tuned:.2f}")
print(f"Tuned R2 Score: {r2_tuned:.2f}")
```

Summary

We'll summarize the results in a report format:

```
classification_report_str = classification_report(y_test, y_pred_tuned)
regression_report_str = f"Mean Squared Error: {mse_tuned:.2f}\nR2 Score:
{r2_tuned:.2f}"

print("Classification Report:")
print(classification_report_str)

print("Regression Report:")
print(regression_report_str)
```

Let's execute the entire code step-by-step. First, we'll start with data loading and preprocessing.

The dataset is successfully loaded, and the first few rows are displayed. The next step involves preprocessing, including handling missing values (if any) and standardizing the features.

Data Preprocessing

Let's check for any missing values and then standardize the features.

```
# Check for missing values
missing_values = data.isnull().sum()

# Drop the 'Id' column as it's not needed for analysis
data = data.drop('Id', axis=1)

# Display missing values
missing_values
```

Let's execute this to identify and handle any missing values.

```
import pandas as pd

# Load the dataset
file_path = '/mnt/data/WineQT.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

Result

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality  Id
0      9.4        5   0
1      9.8        5   1
2      9.8        5   2
3      9.8        6   3
4      9.4        5   4
```

There are no missing values in the dataset. Now, let's proceed with standardizing the features.

Standardizing Features

We'll split the data into features and target, then apply standard scaling.

```
from sklearn.preprocessing import StandardScaler

# Split data into features and target
X = data.drop('quality', axis=1)
```

```
y = data['quality']

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

Next, we'll develop classification and regression models, starting with classification.

Classification

We'll convert the quality scores into a binary classification: "good" (quality ≥ 6) and "bad" (quality < 6). Then we'll split the data, apply k-fold cross-validation, and fit a RandomForestClassifier.

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report

# Convert quality scores into binary classification: good (1) and bad (0)
y_binary = (y >= 6).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2,
random_state=42)

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rfc, X_train, y_train, cv=kf, scoring='accuracy')

# Fit the model
rfc.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = rfc.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Cross-Validation Accuracy: {cv_scores.mean():.2f}")
print(f"Test Accuracy: {accuracy:.2f}")
print(f"F1 Score: {f1:.2f}")

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

Let's execute this part of the code to see the classification results.

```
# Check for missing values
missing_values = data.isnull().sum()

# Drop the 'Id' column as it's not needed for analysis
data = data.drop('Id', axis=1)

# Display missing values
missing_values
```

Result

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
Id                      0
dtype: int64
```

It appears there was an issue with the variable scope. Let's re-execute the entire process from data loading, preprocessing, and then classification to ensure all steps are included in the execution.

Reload and Preprocess Data

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report

# Load the dataset
file_path = '/mnt/data/WineQT.csv'
data = pd.read_csv(file_path)

# Drop the 'Id' column as it's not needed for analysis
data = data.drop('Id', axis=1)

# Split data into features and target
X = data.drop('quality', axis=1)
y = data['quality']

# Standardize the features
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert quality scores into binary classification: good (1) and bad (0)
y_binary = (y >= 6).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2,
random_state=42)

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rfc, X_train, y_train, cv=kf, scoring='accuracy')

# Fit the model
rfc.fit(X_train, y_train)

# Predict on the test set
y_pred = rfc.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

classification_report_str = classification_report(y_test, y_pred)

cv_scores.mean(), accuracy, f1, classification_report_str

import pandas as pd
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report

# Load the dataset
file_path = '/mnt/data/WineQT.csv'
data = pd.read_csv(file_path)

# Drop the 'Id' column as it's not needed for analysis
data = data.drop('Id', axis=1)

# Split data into features and target
X = data.drop('quality', axis=1)
y = data['quality']

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

# Convert quality scores into binary classification: good (1) and bad (0)
y_binary = (y >= 6).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2,
random_state=42)

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rfc, X_train, y_train, cv=kf, scoring='accuracy')
```

```
# Fit the model
rfc.fit(X_train, y_train)

# Predict on the test set
y_pred = rfc.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

classification_report_str = classification_report(y_test, y_pred)

cv_scores.mean(), accuracy, f1, classification_report_str
```

Let's execute this code.

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report

# Convert quality scores into binary classification: good (1) and bad (0)
y_binary = (y >= 6).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_binary, test_size=0.2,
random_state=42)

# Initialize k-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Random Forest Classifier
rfc = RandomForestClassifier(random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rfc, X_train, y_train, cv=kf, scoring='accuracy')
```

```
# Fit the model
rfc.fit(X_train, y_train)

# Predict on the test set
y_pred = rfc.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

classification_report_str = classification_report(y_test, y_pred)

cv_scores.mean(), accuracy, f1, classification_report_str
```