Name: Ranpura Divy D.   REGNO : 21BIT0176

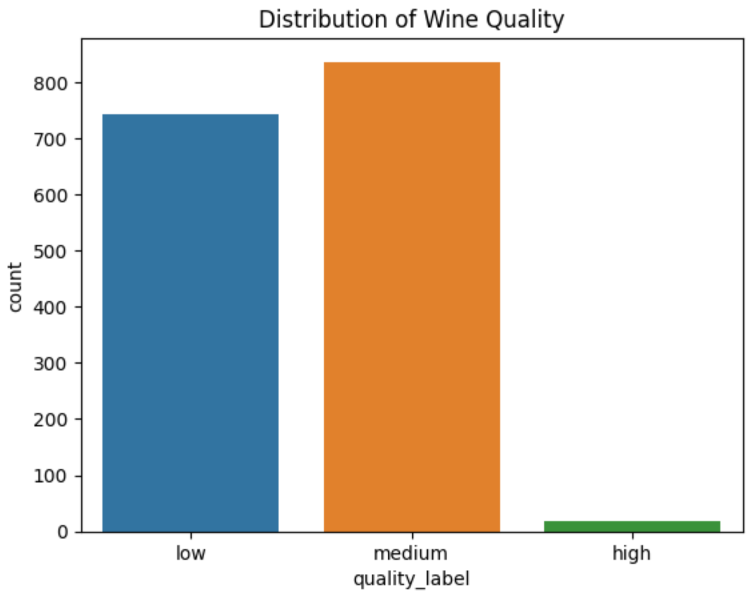
```
# Importing necessary Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt


# Loading the dataset

wine = pd.read_csv("winequality-red.csv")


# Converting the quality column into a categorical variable with three levels: low (3-5), medium (6-7), and high (8-9)

wine['quality_label'] = pd.cut(wine['quality'], bins=[2, 5, 7, 9], labels=['low', 'medium', 'high'])


# Checking the shape and info of the dataset

print(wine.shape)
print(wine.info())

# Checking the summary statistics of the numeric variables

print(wine.describe())
```

```
(1599, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
 12  quality_label         1599 non-null   category
dtypes: category(1), float64(11), int64(1)
memory usage: 151.7 KB
None
       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           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     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  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

```
# Checking the distribution of the quality label

print(wine['quality_label'].value_counts())
```

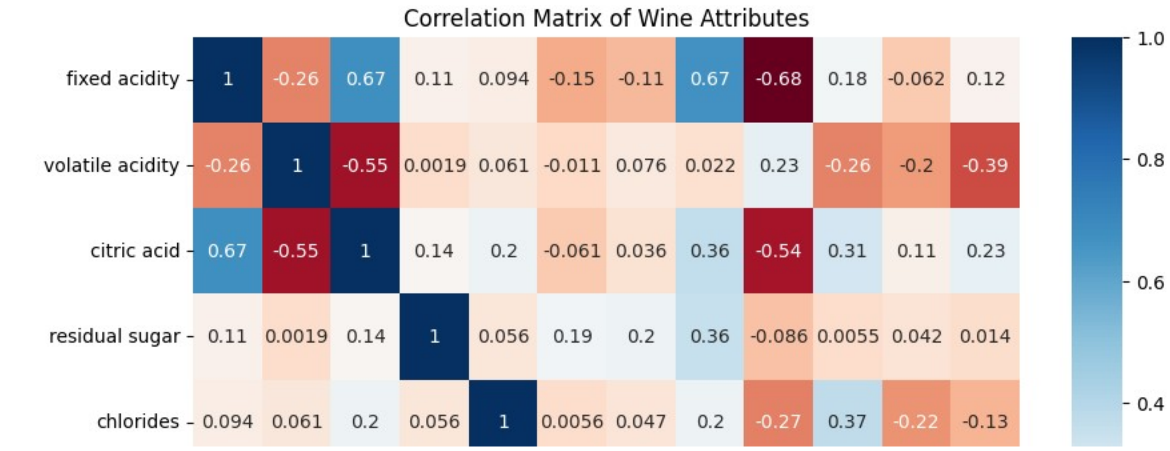
```
medium    837
low       744
high       18
Name: quality_label, dtype: int64
```

```
# Plotting a histogram of the quality label

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

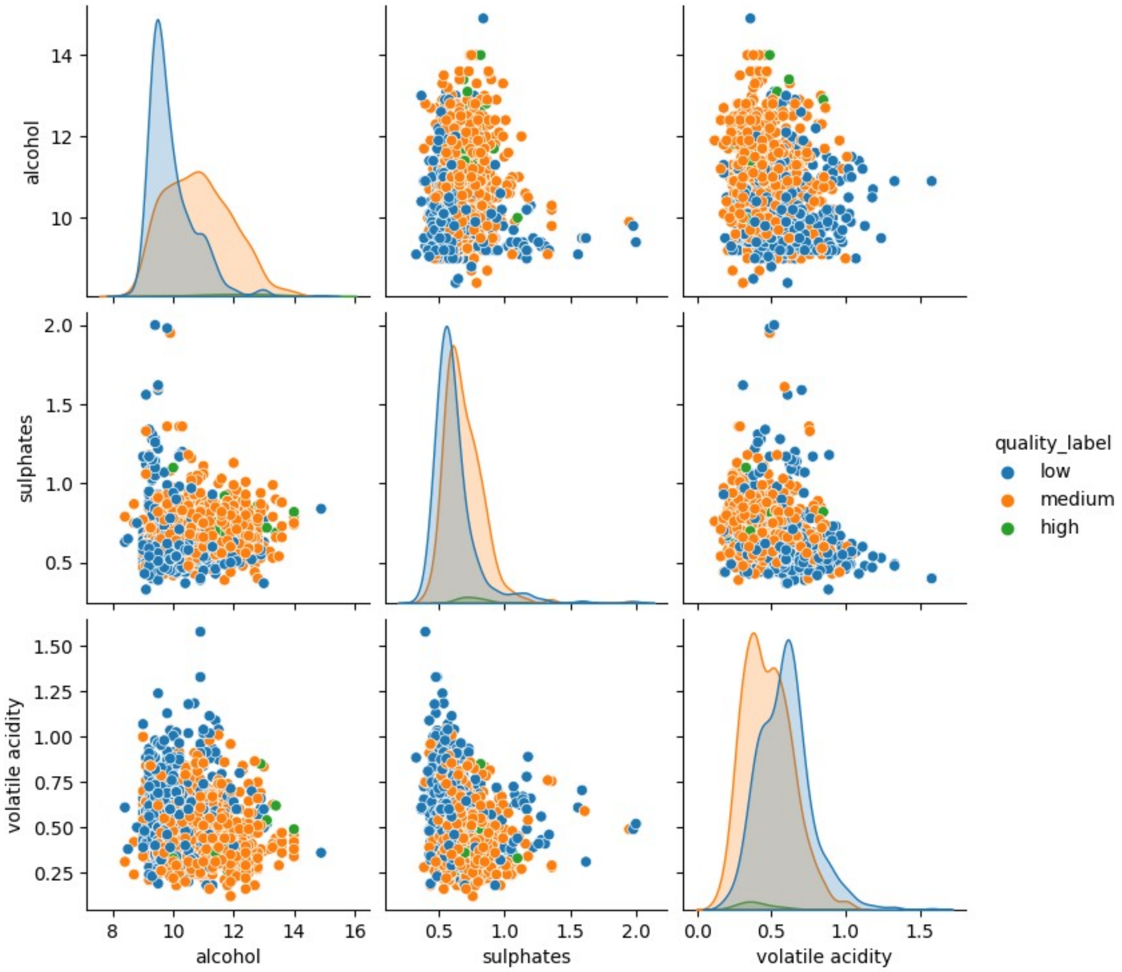



```
# Plotting a correlation matrix of the numeric variables

plt.figure(figsize=(10, 10))
sns.heatmap(wine.corr(), annot=True, cmap='RdBu')
plt.title('Correlation Matrix of Wine Attributes')
plt.show()
```

```
<ipython-input-47-020fce734050>:4: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a
  sns.heatmap(wine.corr(), annot=True, cmap='RdBu')
```



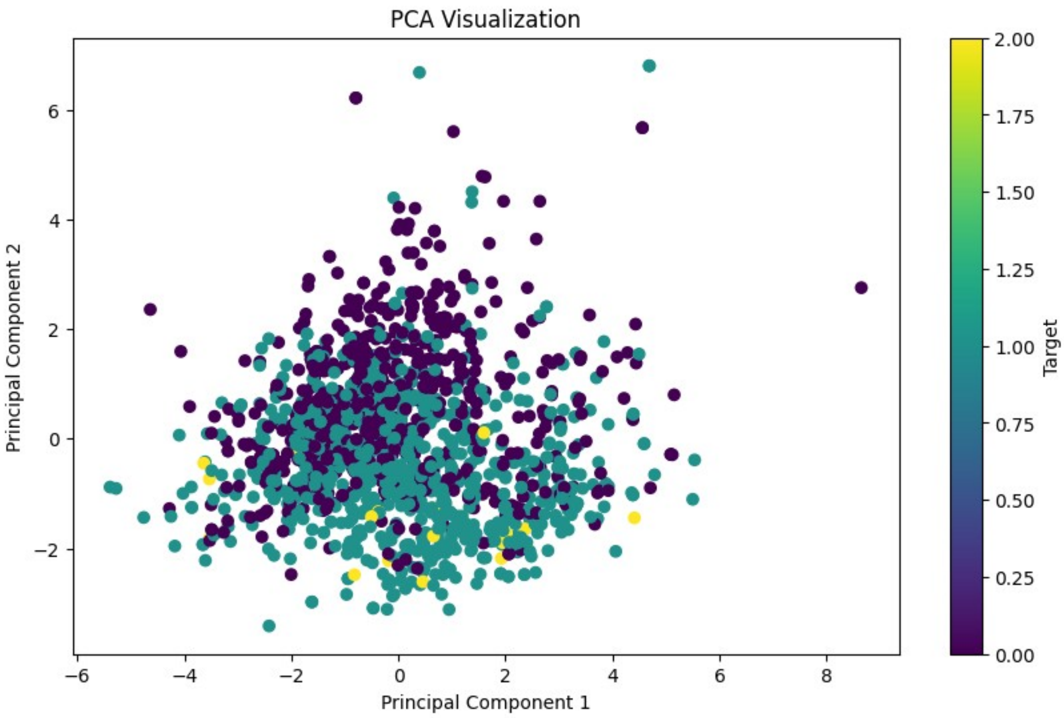

```
# Plotting a pairplot of some selected variables by quality label

sns.pairplot(wine[['alcohol', 'sulphates', 'volatile acidity', 'quality_label']], hue='quality_label')
plt.show()
```




```
# Splitting the dataset into features (X) and target (y)

X = wine.drop(['quality', 'quality_label'], axis=1)
y = wine['quality_label']

# Splitting the data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Data Preprocessing

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

```
X_scaled = scaler.fit_transform(X)


# Define the number of components you want to keep (e.g., 2 for visualization)
n_components = 2

# Create a PCA instance
pca = PCA(n_components=n_components)

# Fit and transform the data to reduce dimensionality
X_pca = pca.fit_transform(X_scaled)

# Calculate the explained variance ratio
explained_variance = pca.explained_variance_ratio_
print("Explained Variance Ratios:", explained_variance)

# Create a DataFrame for the reduced data
df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])  # Adjust column names as needed

# Visualize PCA Results with Colors for Class Labels
plt.figure(figsize=(10, 6))
# Replace 'y' with numeric labels if needed
plt.scatter(df_pca['PC1'], df_pca['PC2'], c=y.cat.codes, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Visualization')
plt.colorbar(label='Target')
plt.show()
```

```
Explained Variance Ratios: [0.28173931 0.1750827 ]
```




```
# Addressing class imbalance using SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)


# Defining the parameter grid for hyperparameter tuning

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Creating a Random Forest Classifier

rf_classifier = RandomForestClassifier(random_state=42)

# Performing GridSearchCV for hyperparameter tuning
```

```
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)

# Getting the best hyperparameters

best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Training the model with the best hyperparameters

best_rf_classifier = RandomForestClassifier(random_state=42, **best_params)
best_rf_classifier.fit(X_train_resampled, y_train_resampled)

# Making predictions on the test set

y_pred = best_rf_classifier.predict(X_test)

# Evaluating the model's performance

accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Classification Report:\n{classification_rep}")
print(f"Confusion Matrix:\n{confusion_mat}")
```

```
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 300}
Accuracy: 0.7875
Classification Report:
              precision    recall  f1-score   support

        high       0.17      0.20      0.18         5
         low       0.78      0.80      0.79       141
      medium       0.81      0.79      0.80       174

    accuracy                           0.79       320
   macro avg       0.59      0.60      0.59       320
weighted avg       0.79      0.79      0.79       320

Confusion Matrix:
[[  1   0   4]
 [  0 113  28]
 [  5  31 138]]
```

```
# Testing with a random observation

sample = np.array([7.4, 0.7, 0.0, 1.9, 0.076,
                   11.0, 34.0, 0.9978,
                   3.51,
                   0.56,
                   9.4]).reshape(1,-1)
sample_pred = logreg.predict(sample)
print('The predicted quality label for this sample is:', sample_pred[0])
```

```
The predicted quality label for this sample is: low
```

+ Code + Text