INFO 531: Data Warehousing and Analytics in the Cloud

Final Project Report

Name: Avikal Singh

Course: INFO 531: Data Warehousing and Analytics in the Cloud

Term: Fall 2024

Submission Week: Week 16 Final Project Report

Instructor's Name: Dr. Nayem Rahman

Date of Submission: 12/18/2024

Option-3: Based on a dataset in a CSV file you will do some data preparation for machine learning and apply some machine learning techniques. Plenty of ML data is available on the Kaggle site. During the final week you will submit a final report.

Problem Statement

This project aims to develop a predictive model to estimate the quality of wines based on their physicochemical properties. By leveraging machine learning techniques, we seek to create a tool that can assist wine producers, retailers, and enthusiasts in assessing wine quality. The model will utilize a dataset comprising various attributes of wines, with the primary challenge being to discern the intricate patterns that determine wine quality ratings.

Objectives:

1. Preprocess and clean the wine quality dataset
2. Identify significant features impacting wine quality through exploratory data analysis
3. Implement and tune machine learning models to predict wine quality accurately
4. Evaluate model performance and refine the approach to enhance prediction accuracy

1. Data Preparation

First, let's examine our dataset. Our dataset includes the following features:

Fixed Acidity: Amount of fixed acids in the wine (e.g., tartaric acid).

Volatile Acidity: Level of volatile acids, which can affect wine aroma.

Citric Acid: A component contributing to the wine's freshness.

Residual Sugar: Amount of sugar remaining after fermentation.

Chlorides: Salt content in the wine.

Free Sulfur Dioxide: SO2 in wine, which helps prevent oxidation.

Total Sulfur Dioxide: Total SO2 present.

Density: The wine's density, which relates to alcohol and sugar levels.

pH: Acidity level of the wine.

Sulphates: Wine's sulfur content, contributing to preservation.

Alcohol: The alcohol content by volume.

Quality: Wine's quality rating (0 to 10), which will serve as the target variable for classification.

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

# Setting 'Id' as index
data = data.set_index('Id')
```

[102] ✓ 0.0s

This code loads the **WineQT.csv** dataset, reads it into a DataFrame, and sets the **'Id'** column as the index for easier data referencing.

```
# Display basic information about the dataset
print("Dataset Info:")
print(data.info())
print("\nFirst 5 Rows:")
data.head()
```

[103] ✓ 0.0s Open 'data' in Data Wrangler

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
Index: 1143 entries, 0 to 1597
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1129 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 116.1 KB
```

This step provides an overview of the dataset's structure, including data types, non-null counts, and the first five rows for a quick preview.

First 5 Rows:

Id	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
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

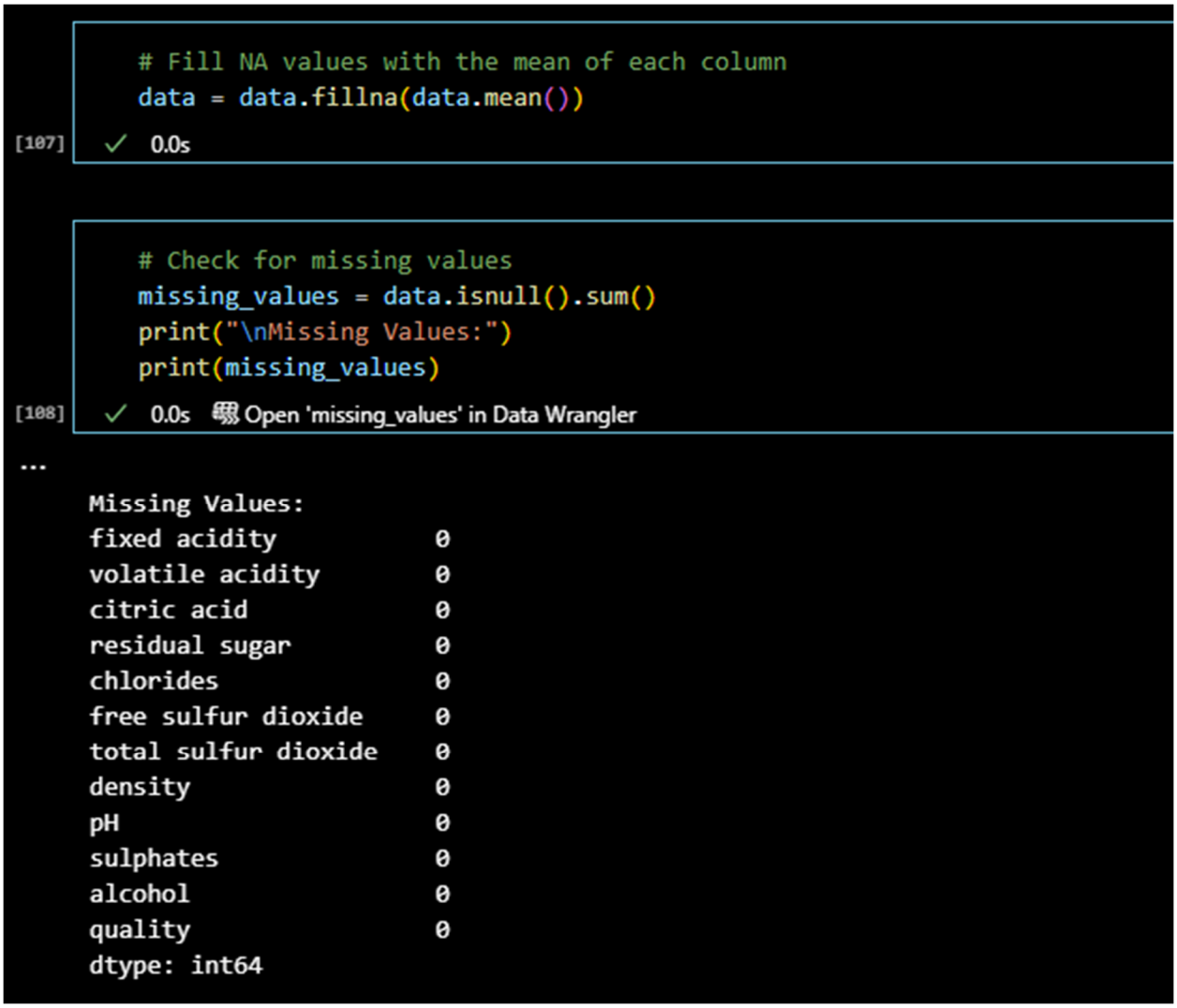
This is the overview of first five rows of the dataset.

```
# Check for missing values
missing_values = data.isnull().sum()
print("\nMissing Values:")
print(missing_values)
```

[106] ✓ 0.0s Open 'missing_values' in Data Wrangler

```
Missing Values:
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar         14
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

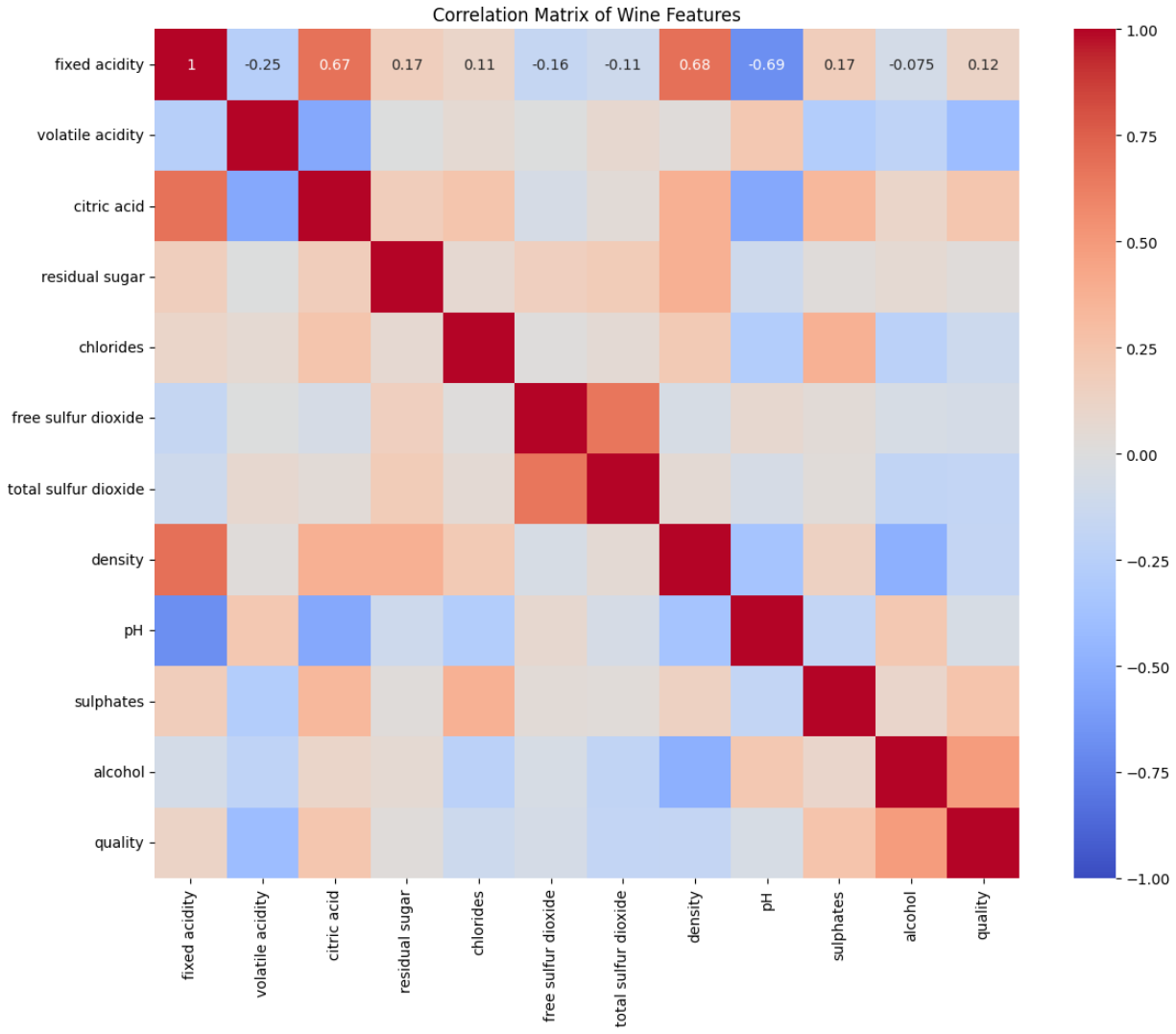
The dataset contains a total of 12 columns, with missing values present only in the **'residual sugar'** column (14 missing values). All other columns are complete, indicating minimal data cleaning is required. Addressing these missing values is essential for accurate model performance.



This step replaces missing values in the dataset with the mean of their respective columns, ensuring no data is lost while maintaining the distribution of the dataset for accurate analysis and modeling.

```
[109]
# Check for duplicates
duplicates = data.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicates}")
✓ 0.0s

Number of duplicate rows: 124

[110]
# Remove duplicate rows
df_no_duplicates = data.drop_duplicates()

# Display the number of rows removed
rows_removed = len(data) - len(df_no_duplicates)
print(f"Number of duplicate rows removed: {rows_removed}")
✓ 0.0s

Number of duplicate rows removed: 124
```

This code checks for duplicate rows in the dataset, removes them to ensure data integrity, and displays the number of duplicates identified and removed to maintain a clean dataset for analysis.



Correlation Matrix of Wine Features

This correlation matrix visually represents the strength and direction of relationships between various wine features. Darker shades indicate a stronger correlation, while the color (blue or red) signifies the direction of the relationship (positive or negative).

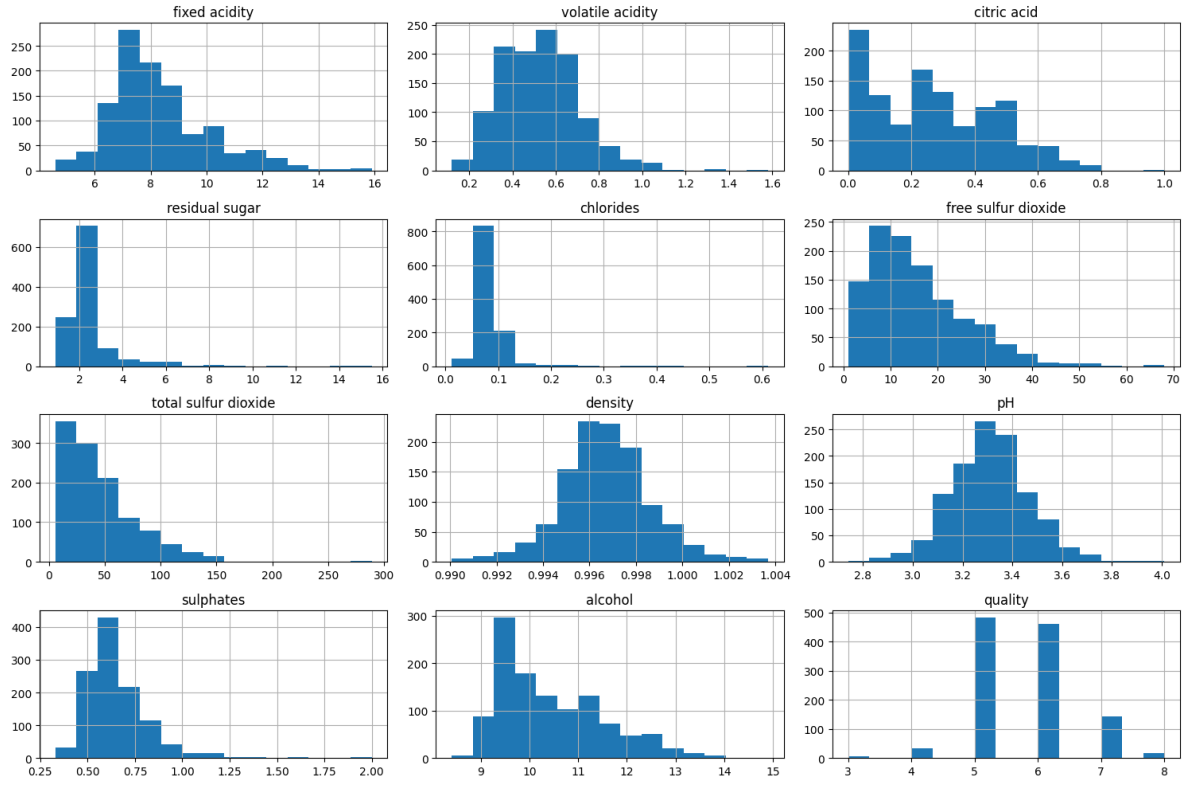
Key Observations:

- **Strong Negative Correlations:** Volatile acidity, density, and chlorides have a strong negative correlation with wine quality.
- **Strong Positive Correlations:** Alcohol content has a strong positive correlation with wine quality.
- **Moderate Correlations:** Fixed acidity and citric acid show moderate positive correlations with wine quality.

2. Playing with Predictor Variables and Response Variable.

```python
# Check range and distribution of each feature
print("\nFeature Ranges and Distribution:")
for column in data.columns:
    if data[column].dtype != 'object':  # Ignore non-numeric features if any
        print(f"{column}: Min = {data[column].min()}, Max = {data[column].max()}")

# Plot the distribution of features to visualize scaling needs
numeric_features = data.select_dtypes(include=[np.number]).columns.tolist()
data[numeric_features].hist(bins=15, figsize=(15, 10))
plt.tight_layout()
plt.show()

# Standardize the features if they are on different scales
scaler = StandardScaler()
scaled_data = pd.DataFrame(scaler.fit_transform(data[numeric_features]), columns=numeric_features)

# Add target column back if it's in the numeric features
if 'quality' in data.columns:  # Assuming 'quality' is the target
    scaled_data['quality'] = data['quality']

print("\nScaled Data Sample:")
print(scaled_data.head())
```

✓ 0.8s

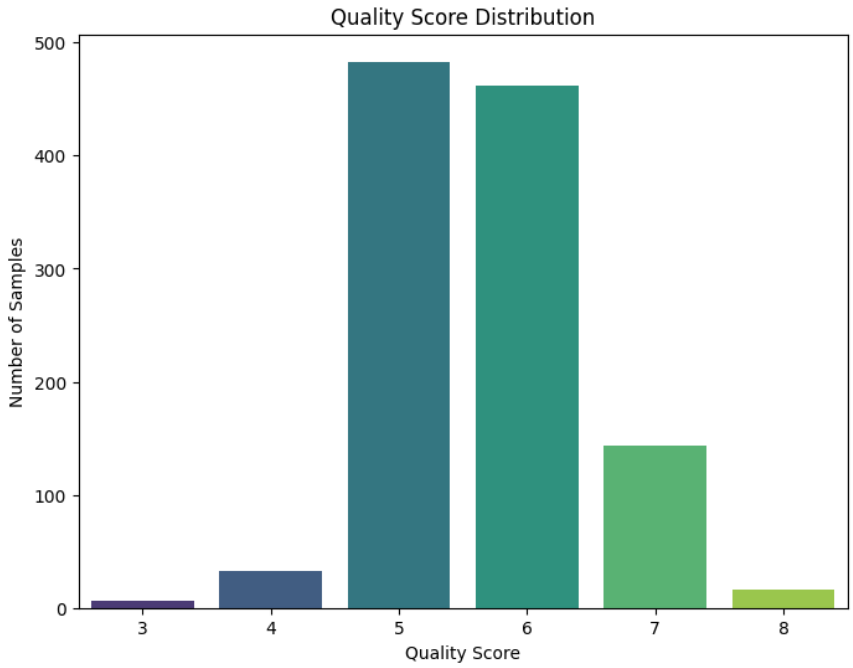


This step analyzes the range and distribution of numeric features to identify scaling requirements. Distributions are visualized using histograms, highlighting variability across features. To ensure all features are on a uniform scale, standardization is applied using

StandardScaler. The standardized data is prepared for modeling, with the target column, 'quality', re-added to the scaled dataset.

```python
# Check the distribution of the target variable (quality)
if 'quality' in data.columns:
    quality_distribution = data['quality'].value_counts()
    print("\nClass Distribution (Quality):")
    print(quality_distribution)

    # Plot the distribution
    plt.figure(figsize=(8, 6))
    sns.barplot(x=quality_distribution.index, y=quality_distribution.values, palette='viridis')
    plt.title('Quality Score Distribution')
    plt.xlabel('Quality Score')
    plt.ylabel('Number of Samples')
    plt.show()

    # Check the class imbalance ratio
    imbalance_ratio = quality_distribution.max() / quality_distribution.min()
    print(f"\nClass Imbalance Ratio (max/min): {imbalance_ratio:.2f}")

    # Determine if class balancing is needed
    if imbalance_ratio > 2.0:  # Threshold for significant imbalance
        print("\nClass imbalance detected.")
    else:
        print("\nClass distribution is balanced.")
```

✓ 0.0s Python



This step examines the distribution of the target variable, 'quality', to identify potential class imbalance issues. The distribution is visualized using a bar plot, and the class imbalance ratio (max/min) is calculated. The ratio exceeds a threshold of 2.0 which shows a significant imbalance is flagged, indicating the need for class balancing techniques to ensure fair model training.

3. Training and Testing Data Processing

```python
# Separate predictors and target
X = scaled_data.drop(columns=['quality'], axis=1)
y = data['quality']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

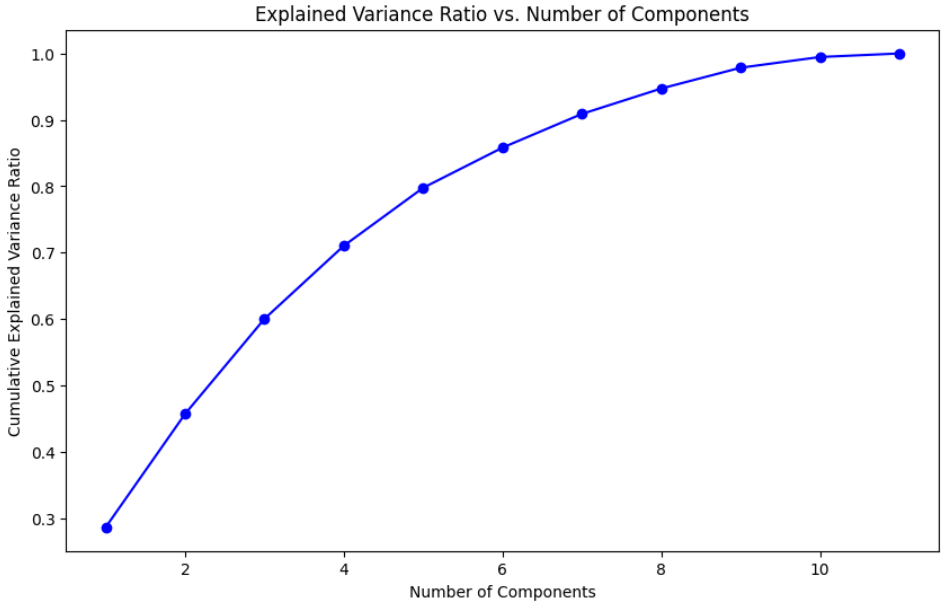
✓ 0.0s Python

We separated predictors and target variable, followed by splitting the dataset into training and testing sets with stratification to preserve class distribution.

```python
# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X)

# Calculate the cumulative explained variance ratio
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Plot the cumulative explained variance ratio
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance_ratio) + 1), cumulative_variance_ratio, 'bo-')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Components')
```

✓ 0.2s Python

We are applying PCA, calculating the cumulative explained variance ratio, and plotting it to determine the optimal number of components for dimensionality reduction.



This plot shows the cumulative explained variance ratio as a function of the number of principal components, indicating the amount of variance captured by each additional component.

```
# Select the number of components that explain 90% of the variance
n_components = np.argmax(cumulative_variance_ratio >= 0.90) + 1

print(f"Number of components selected: {n_components}")
```

✓ 0.0s Python

```
Number of components selected: 7
```

```
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

✓ 0.0s Python

This is the code for selecting the 7 PCA components explaining 90% variance and transforming the training and testing data using the selected components.

4. ML Techniques

4.1 Regression Models

a. Linear Regression

```
# Linear Regression Model
linear_model = LinearRegression()
linear_model.fit(X_train_pca, y_train)

# Predictions and evaluation for Linear Regression
y_pred_linear = linear_model.predict(X_test_pca)
print("\nLinear Regression:")
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred_linear):.2f}")
print(f"R^2 Score: {r2_score(y_test, y_pred_linear):.2f}")
```

✓ 0.0s

```
Linear Regression:
Mean Squared Error: 0.40
R^2 Score: 0.39
```

Training a Linear Regression model on PCA-transformed data to predict target values. The linear regression model has a Mean Squared Error (MSE) of 0.40, indicating the average squared difference between predicted and actual values. The R^2 score of 0.39 suggests that the model explains about 39% of the variance in the data, indicating a relatively weak fit.

b. Random Forest Regressor

```
# Random Forest Regressor Model
rf_regressor = RandomForestRegressor(random_state=42)
rf_regressor.fit(X_train_pca, y_train)

# Predictions and evaluation for Random Forest Regressor
y_pred_rf = rf_regressor.predict(X_test_pca)
print("\nRandom Forest Regressor:")
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred_rf):.2f}")
print(f"R^2 Score: {r2_score(y_test, y_pred_rf):.2f}")
```

✓ 0.5s

```
Random Forest Regressor:
Mean Squared Error: 0.36
R^2 Score: 0.44
```

Training a Random Forest Regressor on PCA-transformed data to predict target values. The Random Forest Regressor model has a Mean Squared Error (MSE) of 0.36, indicating better prediction accuracy compared to linear regression. The R^2 score of 0.44 shows that the model explains 44% of the variance, indicating a moderate fit to the data.

```
bins = [0, 4, 6, 10]  # Define bins for low, medium, high quality
labels = ['low', 'medium', 'high']
y_binned = pd.cut(data['quality'], bins=bins, labels=labels)

# Update the target variable for classification
y_train_cls, y_test_cls = train_test_split(y_binned, test_size=0.2, random_state=42, stratify=y_binned)
```

✓ 0.0s Python

We are binning the target variable 'quality' into low, medium, and high categories, followed by splitting the binned data into training and testing sets for classification.

4.2 Classification Models

a. Logistic Regression

```
# Logistic Regression Model
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train_pca, y_train_cls)

# Predictions and evaluation for Logistic Regression
y_pred_logistic = logistic_model.predict(X_test_pca)
print("\nLogistic Regression Classification:")
print(classification_report(y_test_cls, y_pred_logistic))
print(f"Accuracy: {accuracy_score(y_test_cls, y_pred_logistic):.2f}")
```

✓ 0.0s

```
Logistic Regression Classification:
              precision    recall  f1-score   support

        high       0.00      0.00      0.00        32
         low       0.00      0.00      0.00         8
      medium       0.82      0.98      0.90       189

    accuracy                           0.81       229
   macro avg       0.27      0.33      0.30       229
weighted avg       0.68      0.81      0.74       229

Accuracy: 0.81
```

We train a logistic regression model on PCA-transformed training data and evaluates its performance using a classification report and accuracy score. The model achieves an overall accuracy of 81%, but performs poorly on the 'high' and 'low' classes, with strong performance on the 'medium' class.

b. Support Vector Machine

```
# Support Vector Machine Model
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train_pca, y_train_cls)

# Predictions and evaluation for SVM
y_pred_svm = svm_model.predict(X_test_pca)
print("\nSupport Vector Machine Classification:")
print(classification_report(y_test_cls, y_pred_svm))
print(f"Accuracy: {accuracy_score(y_test_cls, y_pred_svm):.2f}")
```

✓ 0.0s

```
Support Vector Machine Classification:
              precision    recall  f1-score   support

        high       0.00      0.00      0.00        32
         low       0.00      0.00      0.00         8
      medium       0.82      0.99      0.90       189

    accuracy                           0.82       229
   macro avg       0.27      0.33      0.30       229
weighted avg       0.68      0.82      0.74       229

Accuracy: 0.82
```

We train a Support Vector Machine (SVM) model with an RBF kernel on PCA-transformed data and evaluates its performance. The model achieves an overall accuracy of 82%, with strong performance on the 'medium' class, but struggles with the 'high' and 'low' classes, similar to the logistic regression model.