

project2

November 7, 2024

1 Wine Quality Prediction and Clustering Analysis

1.1 Project Overview

This project aims to analyze and predict the quality of wines based on various physicochemical properties. The dataset consists of red and white wine samples, with several features such as acidity, alcohol content, sugar levels, and pH, along with a target variable indicating the quality of each wine. The goal is to predict wine quality and identify patterns using clustering techniques.

1.2 Type of Learning/Algorithm

This project employs **unsupervised learning** for clustering analysis and **supervised learning** for regression tasks. The following algorithms are used in this analysis:

- **Clustering (Unsupervised Learning):**
 - K-Means
 - Agglomerative Clustering
- **Regression (Supervised Learning):**
 - Linear Regression

1.3 Type of Task

The task associated with this dataset involves **predicting wine quality** using the following approaches:

- **Clustering:** Unsupervised learning models are used to group wines into clusters, and then the clusters are mapped to quality labels based on their most frequent quality scores.
- **Regression:** A supervised learning approach using Linear Regression to predict continuous wine quality scores.

1.4 Data Overview

The dataset contains the following features related to the wine's characteristics: - **Fixed acidity** - **Volatile acidity** - **Citric acid** - **Residual sugar** - **Chlorides** - **Free sulfur dioxide** - **Total sulfur dioxide** - **Density** - **pH** - **Sulphates** - **Alcohol**

Additionally, there are two datasets: one for **red wine** and one for **white wine**, each containing a **quality** score that indicates the quality of the wine (integer values from 3 to 9).

1.5 Data Cleaning

- Missing values: None (the dataset is clean with no missing values).
- Duplicates: Removed any duplicate records to ensure the integrity of the data.

1.6 Citation

- UCI Machine Learning Repository. (n.d.). *Wine Quality Dataset*. Retrieved from <https://archive.ics.uci.edu/dataset/186/wine+quality>

```
[1]: # ### Import Necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import confusion_matrix, classification_report, \
    accuracy_score, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import joblib

[2]: # ### Set File Paths and Constants
RED_WINE_PATH = '../data/winequality-red.csv' # Path to the red wine quality
dataset
WHITE_WINE_PATH = '../data/winequality-white.csv' # Path to the white wine
quality dataset
TARGET_VARIABLE = 'quality' # Target variable name
FEATURES = [
    'fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
    'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
    'pH', 'sulphates', 'alcohol', 'wine_type_factor',
    # 'quality', 'wine_type'
] # Features for clustering

[3]: # ### Load the Datasets
red_wine_data = pd.read_csv(RED_WINE_PATH, sep=';')
white_wine_data = pd.read_csv(WHITE_WINE_PATH, sep=';')

# Add a column to indicate the wine type
red_wine_data['wine_type'] = 'red'
white_wine_data['wine_type'] = 'white'

# Concatenate the datasets
data = pd.concat([red_wine_data, white_wine_data], ignore_index=True)

# ### Data Cleaning
# Check for duplicates and drop if any
```

```
data.drop_duplicates(inplace=True)

# ### Data Overview
print(f"Dataset shape: {data.shape} = (examples, features + target).")
print(f"Features: {data.columns.tolist()}")
print(f"Missing values: \n{data.isnull().sum()}")
```

```
Dataset shape: (5320, 13) = (examples, features + target).
Features: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
'sulphates', 'alcohol', 'quality', 'wine_type']
Missing values:
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
wine_type             0
dtype: int64
```

```
[4]: # ### Print Data Types of Columns
print(data.dtypes)

# ### Print Unique Values of Features
print( "\nUnique Values of Features:\n", sorted(list(data[TARGET_VARIABLE].
↪unique())) )
```

```
fixed acidity          float64
volatile acidity       float64
citric acid            float64
residual sugar         float64
chlorides              float64
free sulfur dioxide    float64
total sulfur dioxide   float64
density               float64
pH                   float64
sulphates             float64
alcohol               float64
quality               int64
wine_type             object
```

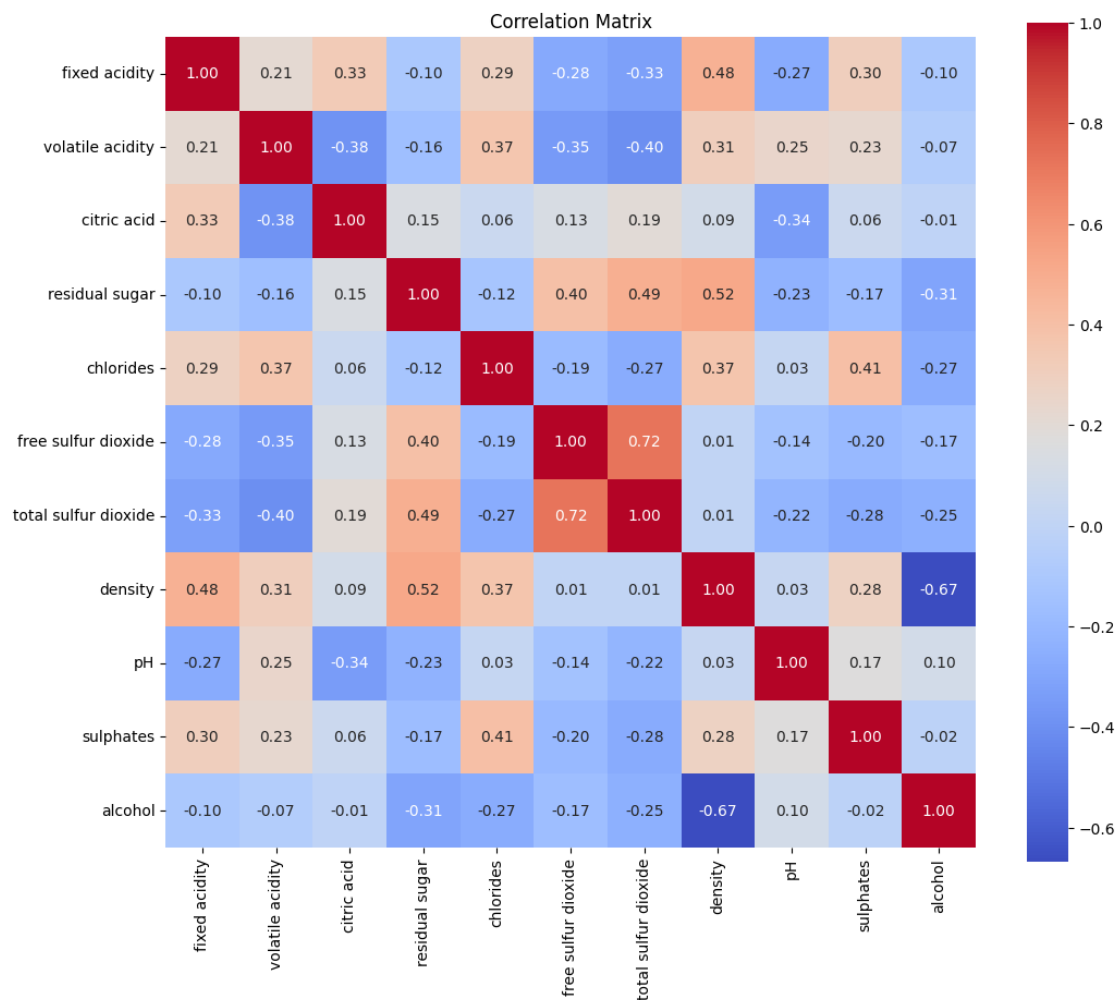
dtype: object

Unique Values of Features:

[3, 4, 5, 6, 7, 8, 9]

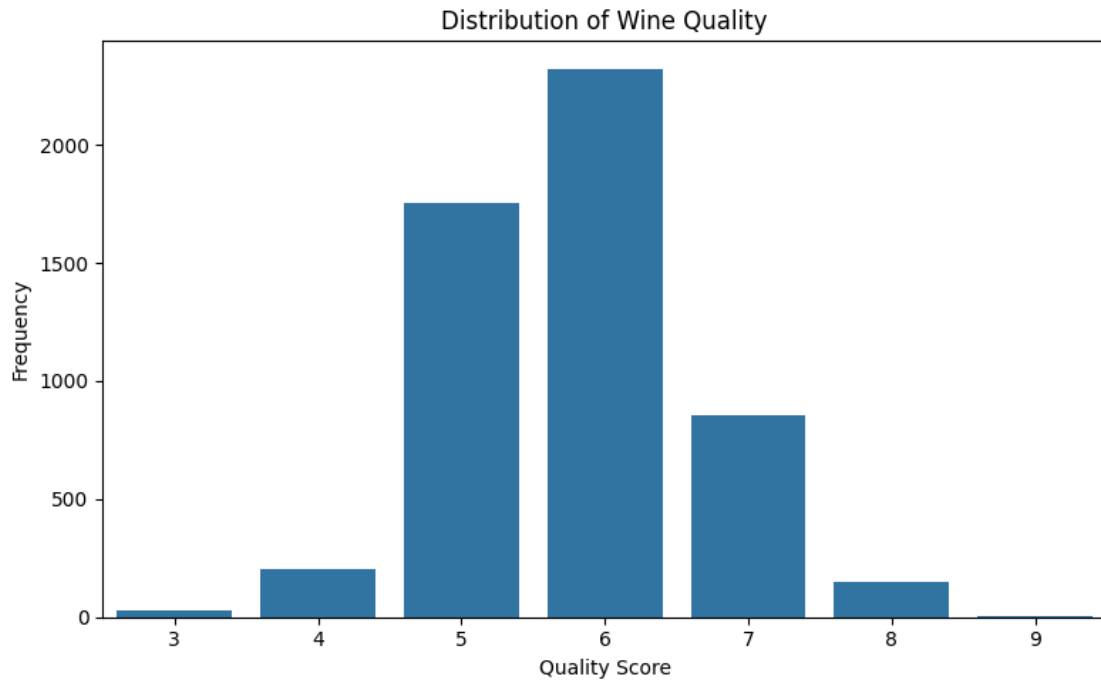
```
[5]: correlation_matrix = data[FEATURES].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm',
            square=True,
            xticklabels=FEATURES, yticklabels=FEATURES)
plt.title('Correlation Matrix')
plt.show()
```



```
[6]: # ### Exploratory Data Analysis (EDA)
# Visualize the distribution of the target variable: wine quality
```

```
plt.figure(figsize=(8, 5))
sns.countplot(x=TARGET_VARIABLE, data=data)
plt.title('Distribution of Wine Quality')
plt.xlabel('Quality Score')
plt.ylabel('Frequency')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



1.6.1 Data Overview

After loading and cleaning the datasets, we get the following summary:

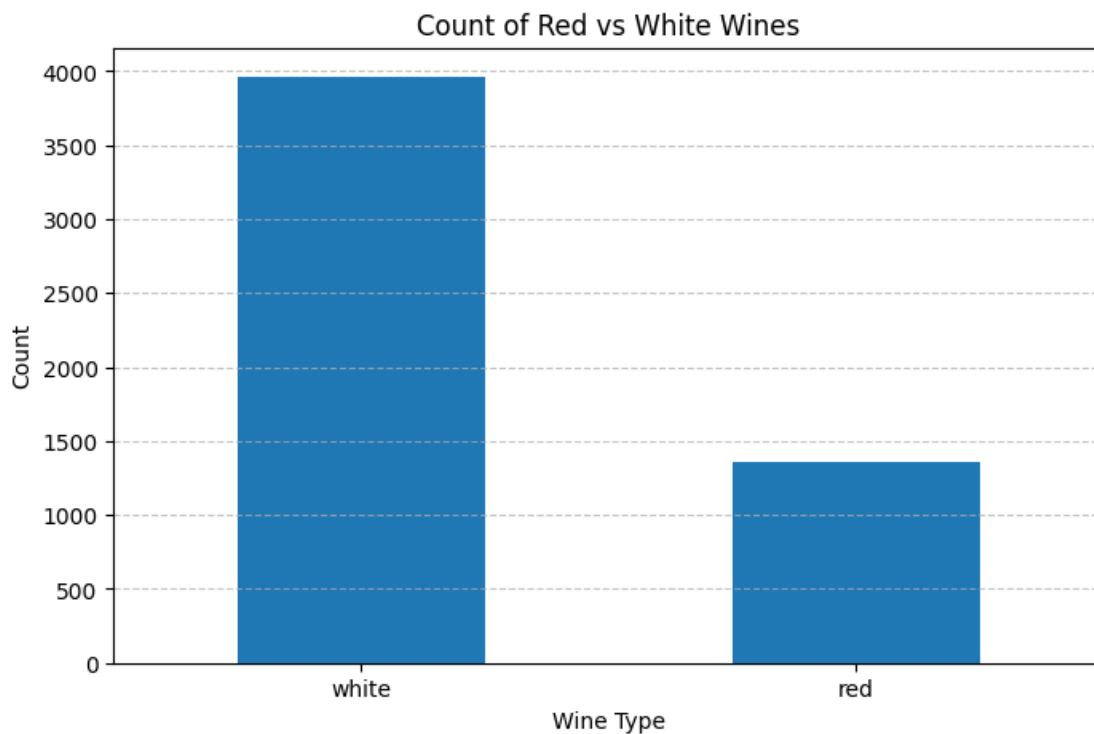
- **Dataset Shape:** The dataset contains 5320 rows (examples) and 13 columns (features including the target `quality` and the `wine_type` column).
- **Features:** The dataset contains the following features:
 - ‘fixed acidity’, ‘volatile acidity’, ‘citric acid’, ‘residual sugar’, ‘chlorides’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘density’, ‘pH’, ‘sulphates’, ‘alcohol’, ‘quality’, ‘wine_type’.
- **Missing Values:** There are no missing values in any of the columns, indicating a clean dataset ready for analysis.
- **Data Types:** The columns have the following data types:
 - All feature columns are of type `float64`, except for the `quality` column (which is `int64`) and the `wine_type` column (which is `object`).

- **Unique Values in the Target Variable (quality):** The unique values for wine quality range from 3 to 9.
 - We see that the wine quality has an almost normal distribution with mean rating of 6.

```
[7]: # ### Count of Red vs White
wine_counts = data['wine_type'].value_counts()
print(wine_counts)

# ### Plotting
plt.figure(figsize=(8, 5))
wine_counts.plot(kind='bar')
plt.title('Count of Red vs White Wines')
plt.xlabel('Wine Type')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
white    3961
red      1359
Name: wine_type, dtype: int64
```



1.7 Distribution white vs. red wine

- We see that there are 3961 white samples, 1359 red samples.

```
[8]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# ### Prepare Data for the Clustering Algorithm
# Split the dataset into training and testing sets
X_train, X_test = train_test_split(data, test_size=0.3, random_state=42)

# apply standard scaling to features set
scaler = StandardScaler()
X_train.loc[:, FEATURES] = scaler.fit_transform(X_train.loc[:, FEATURES])
X_test.loc[:, FEATURES] = scaler.transform(X_test.loc[:, FEATURES])

# Print the shape of the training and testing sets
print(f"Training set shape: {X_train.shape}")
print(f"Testing set shape: {X_test.shape}")

X_train.head()
# X_train[FEATURES]
```

Training set shape: (3724, 13)

Testing set shape: (1596, 13)

```
[8]:      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
4683      -0.458445        -1.039481      0.226768        -0.549262    -0.775769
2007      -0.691225        -1.218694      0.497500        -0.857809    -0.354036
6097      -0.303258        -0.860268     -0.450064         0.244143    -0.113045
5962      -0.070478        -0.203154     -0.450064         1.346094    -0.233540
494       -0.536039         0.274747     -0.585430         0.706962    -0.143169

      free sulfur dioxide  total sulfur dioxide  density      pH  \
4683          0.142201        -0.651802 -1.019137 -0.030720
2007         -0.241993         0.013879 -0.838475  0.217301
6097         -0.461532        -0.161300 -0.336638 -0.836786
5962          0.087316         0.364238  0.710529 -0.526760
494         -0.132223        -0.424069  0.232111  1.333393

      sulphates  alcohol  quality  wine_type
4683   1.342830  0.626548        6     white
2007   0.006309 -0.215335        6     white
6097   0.076652  0.205606        5     white
5962  -0.908153 -0.888841        5     white
494    0.146995  1.300053        6       red
```

```

[9]: import itertools
import numpy as np
from sklearn.metrics import accuracy_score
import heapq

def compute_cluster_options(df, cluster_col, target_col):
    """
    Compute cluster options based on the DataFrame.

    df: DataFrame containing the clusters and true labels
    cluster_col: column name for cluster labels
    target_col: column name for true labels

    Returns a dictionary where keys are cluster labels and values are lists of
    ↪ possible true labels.
    """
    cluster_options = {}

    # Group by cluster labels and collect unique target labels
    for cluster, group in df.groupby(cluster_col):
        # print(cluster, group)
        # unique_labels = group[target_col].unique()
        unique_labels, unique_counts = np.unique(group[target_col],
        ↪ return_counts=True)
        cluster_options[int(cluster)] = sorted(
            list(zip(unique_labels, unique_counts)), key=lambda x: -x[1]
        )

    return cluster_options

def label_permute_compare(y_labels, clusters, cluster_options):
    """
    y_labels: labels dataframe object
    clusters: clustering label prediction output
    cluster_options: (cluster) => list( label, count in cluster ) as seen in
    ↪ training data

    Returns the best permuted label order and its corresponding accuracy.
    Example output: ({3: 0, 4: 1, 1: 2, 2: 3}, 0.74)
    """
    y_np_labels = y_labels.to_numpy().flatten()

    # Get unique true labels and their counts
    true_labels, true_counts = np.unique(y_np_labels, return_counts=True)
    f_map = {
        value : count
        for (value, count) in zip(true_labels, true_counts)
    }

```



```

}

# Get unique predicted labels and their counts
pred_labels, pred_counts = np.unique(clusters, return_counts=True)
cluster_counts = {
    value : count
    for (value, count) in zip(pred_labels, pred_counts)
}
# print(cluster_options)
assert sum(pred_counts) == sum(true_counts)

# Generate permutations of the predicted labels
best_accuracy = 0
best_mapping = {}

# Greedy matching algorithm,
for (cluster, label_counts) in cluster_options.items():
    best_mapping[cluster] = label_counts[0][0]
    best_accuracy += label_counts[0][1]

best_accuracy /= len(y_np_labels)
# print(f_map, best_mapping, best_accuracy)
return best_mapping, best_accuracy

def evaluate_model(y_true, y_pred, model_name='', display=True):
    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    pct_error = (mae / np.mean(y_true)) * 100 if np.mean(y_true) != 0 else None
    ↪ # Avoid division by zero
    conf_matrix = confusion_matrix(y_true, y_pred)

    if display:
        # Print metrics
        print(f"Evaluation Results for: {model_name}")
        print(f"Accuracy: {accuracy:.4f}")
        print(f"Mean Absolute Error: {mae:.4f}")
        print(f"Percentage Error: {pct_error:.2f}%")

        # Plot confusion matrix
        plt.figure(figsize=(8, 6))
        sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
                    xticklabels=np.unique(y_true),
                    yticklabels=np.unique(y_true))
        plt.title(f'Confusion Matrix: {model_name}')
        plt.xlabel('Predicted Quality')

```

```

plt.ylabel('Actual Quality')
plt.show()

return {
    "accuracy" : accuracy,
    "mae" : mae,
    "pct_error" : pct_error,
}

```

1.7.1 Clustering Evaluation and Model Comparison

This code defines three functions for evaluating clustering models:

1. **compute_cluster_options**: Calculates the most frequent true labels for each cluster.
2. **label_permute_compare**: Finds the best mapping between predicted clusters and true labels, and computes the accuracy of this mapping.
3. **evaluate_model**: Evaluates the model's performance using accuracy, Mean Absolute Error (MAE), and percentage error, and displays a confusion matrix.

These functions help assess how well clustering algorithms predict wine quality by comparing predicted labels to true wine quality labels.

```

[10]: # ### Define the WineQualityClustering Class
class WineQualityClustering:
    def __init__(self, n_clusters, features, target,
                 metric='euclidean', linkage='ward'):
        self.features = features
        self.target = target
        # self.model = AgglomerativeClustering(n_clusters=n_clusters,
        ↪ affinity=metric, linkage=linkage)
        self.model = KMeans(n_clusters=n_clusters, n_init=10, random_state=42)
        self.mapping = {}

    def fit(self, df):
        """ Fit the clustering model. """
        df['cluster'] = self.model.fit_predict( df[self.features])
        # print(df['cluster'].isna().sum())

        cluster_options = compute_cluster_options(df, 'cluster', self.target)
        # print(cluster_options)

        best_mapping, best_accuracy = label_permute_compare(df[self.target],
        ↪ df['cluster'], cluster_options)
        self.mapping = best_mapping
        # print(best_accuracy)

        return df['cluster'].map(self.mapping)

```

```

def predict(self, df):
    clusters = pd.Series( self.model.predict( df[self.features] ) )
    return clusters.map(self.mapping)

def evaluate(self, df):
    """ Evaluate the clustering performance using confusion matrix and
    ↪classification report. """
    y_true = df[self.target]
    y_pred = self.predict(df)
    return evaluate_model(y_true, y_pred, display=False)

```

1.7.2 WineQualityClustering Class

This class performs wine quality clustering using K-Means:

1. **__init__**: Initializes the clustering model with the number of clusters, features, and target variable.
2. **fit**: Trains the clustering model, assigns clusters, and maps clusters to the most frequent wine quality labels.
3. **predict**: Predicts the cluster for new data.
4. **evaluate**: Evaluates the clustering model's performance using accuracy and a confusion matrix.

```

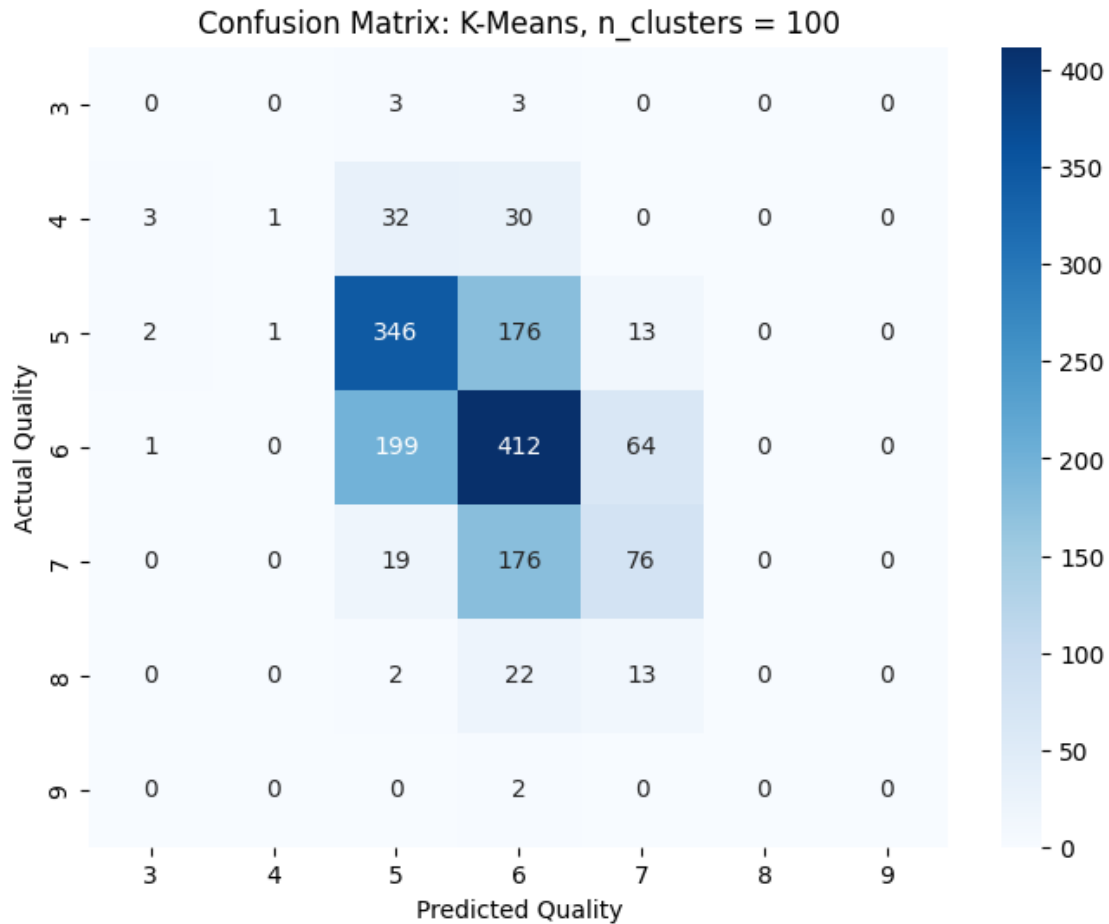
[21]: # Do sample execution on 100 clusters
n_clusters = 100
w = WineQualityClustering(n_clusters, FEATURES, TARGET_VARIABLE)

# Fit model
w.fit(X_train)

# Evaluate model
_ = evaluate_model(X_test[TARGET_VARIABLE], w.predict(X_test),
    ↪model_name=f'K-Means, n_clusters = {n_clusters}')

```

Evaluation Results for: K-Means, n_clusters = 100
 Accuracy: 0.5232
 Mean Absolute Error: 0.5426
 Percentage Error: 9.37%



```
[20]: # Initialize lists to hold results
train_accuracy_results = []
train_mae_results = []
test_accuracy_results = []
test_mae_results = []

# cluster_range = range(10, 200 + 1, 10)
cluster_range = range(100, 500 + 1, 50)

# Loop over the range of clusters, fit the model, and evaluate
for n_clusters in cluster_range:
    # Initialize the WineQualityClustering model
    w = WineQualityClustering(n_clusters, FEATURES, TARGET_VARIABLE)

    # Fit the model on the training data
    w.fit(X_train)

    # Evaluate on the training set
```

```

train_eval_res = w.evaluate(X_train)
train_accuracy_results.append(train_eval_res["accuracy"])
train_mae_results.append(train_eval_res["mae"])

# Evaluate on the test set
test_eval_res = w.evaluate(X_test)
test_accuracy_results.append(test_eval_res["accuracy"])
test_mae_results.append(test_eval_res["mae"])

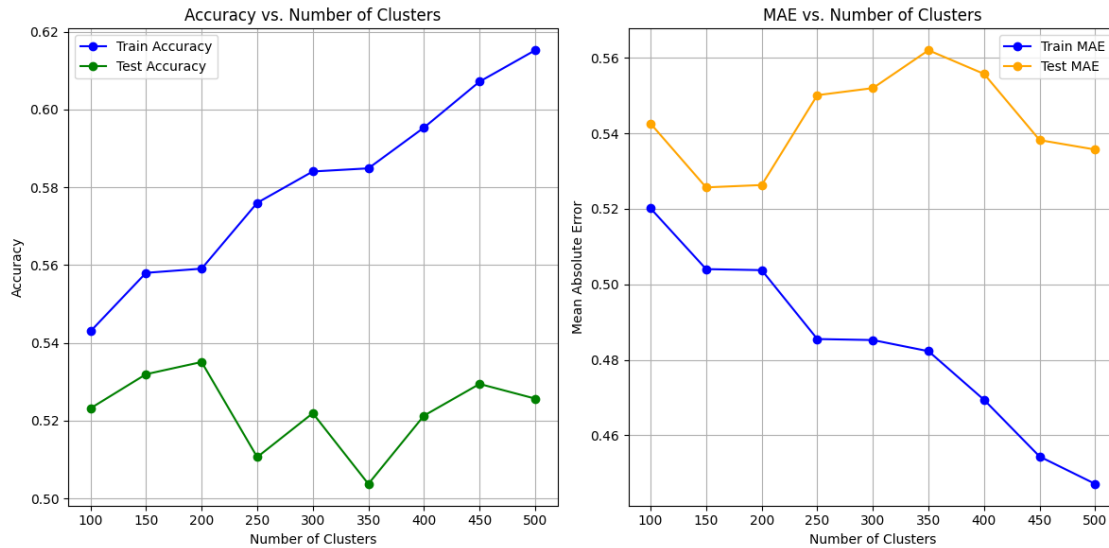
# Plot the results
plt.figure(figsize=(12, 6))

# Plot Accuracy vs. Number of Clusters (for both train and test sets)
plt.subplot(1, 2, 1)
plt.plot(cluster_range, train_accuracy_results, marker='o', label='Train_
↳Accuracy', color='blue')
plt.plot(cluster_range, test_accuracy_results, marker='o', label='Test_
↳Accuracy', color='green')
plt.title('Accuracy vs. Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Accuracy')
plt.legend()
plt.xticks(cluster_range)
plt.grid()

# Plot MAE vs. Number of Clusters (for both train and test sets)
plt.subplot(1, 2, 2)
plt.plot(cluster_range, train_mae_results, marker='o', label='Train MAE',
↳color='blue')
plt.plot(cluster_range, test_mae_results, marker='o', label='Test MAE',
↳color='orange')
plt.title('MAE vs. Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.xticks(cluster_range)
plt.grid()

plt.tight_layout()
plt.show()

```



1.7.3 Cluster Evaluation

This code evaluates the performance of the **WineQualityClustering** model for different numbers of clusters:

- It loops over a range of cluster values (100 to 500, with a step size of 50) and fits the model on the training data (**X_train**).
- For each number of clusters, the model's **accuracy** and **mean absolute error (MAE)** are calculated for both the training dataset (**X_train**) and the test dataset (**X_test**).
- Four key metrics are evaluated for each cluster count:
 - **Train Accuracy**: Measures the model's accuracy on the training data.
 - **Train MAE**: Measures the model's Mean Absolute Error on the training data.
 - **Test Accuracy**: Measures the model's accuracy on the test data.
 - **Test MAE**: Measures the model's Mean Absolute Error on the test data.

Two plots are generated to visualize the results: - **Accuracy vs. Number of Clusters**: Shows how the accuracy changes with the number of clusters for both the training and test sets. - **MAE vs. Number of Clusters**: Displays the relationship between the Mean Absolute Error and the number of clusters for both the training and test sets.

Observation: As expected, model accuracy and MAE improve on training data as number of clusters increases. However, the accuracy on the test data set does not significantly improve as the number of clusters increases. This indicates that adding more clusters may not improve the model's performance on the test data.

```
[13]: # ### Linear Regression Approach
      # We will also try to predict quality using Linear Regression as a comparison.

      # Initialize and fit the Linear Regression model
      lr_model = LinearRegression()
```

```

lr_model.fit(X_train[FEATURES], X_train[TARGET_VARIABLE])

# Make predictions
y_test_lr = X_test[TARGET_VARIABLE]
y_pred_lr = np.round(
    lr_model.predict(X_test[FEATURES])
).astype(int) # Round predictions to nearest integer quality score

# Evaluate Linear Regression model
# Evaluate model
_ = evaluate_model(y_test_lr, y_pred_lr, model_name=f'Linear Regression')

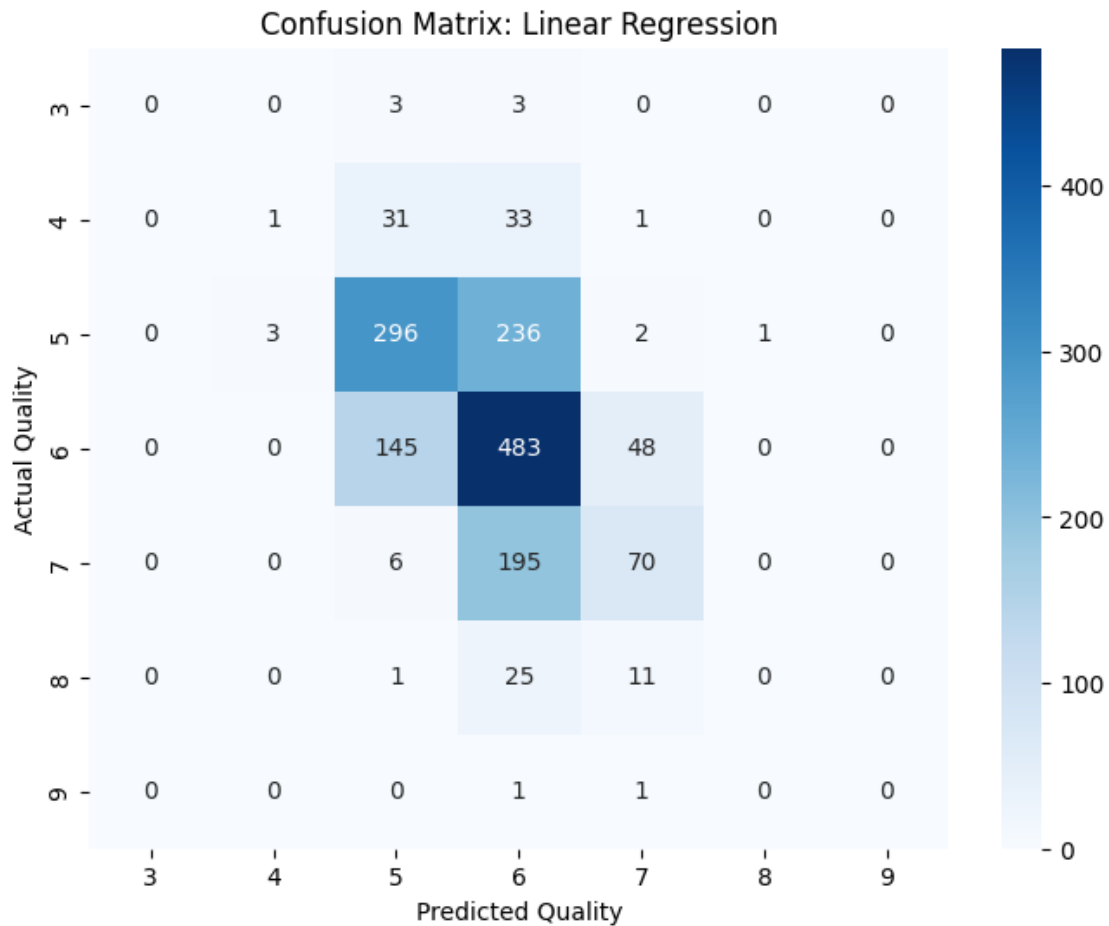
```

Evaluation Results for: Linear Regression

Accuracy: 0.5326

Mean Absolute Error: 0.5201

Percentage Error: 8.98%



1.7.4 Linear Regression vs K-Means Clustering

This section compares the performance of **Linear Regression** and **K-Means Clustering** for predicting wine quality.

- **Linear Regression:**
 - Accuracy: 53.26%
 - Mean Absolute Error (MAE): 0.5201
 - Percentage Error: 8.98%
- **K-Means Clustering (n_clusters = 100):**
 - Accuracy: 52.32%
 - Mean Absolute Error (MAE): 0.5426
 - Percentage Error: 9.37%

Observation: Linear Regression outperforms K-Means clustering in terms of accuracy and lower mean absolute error, indicating it is a better approach for predicting wine quality in this case.

```
[14]: # ### Save the Model if Needed (Optional)
# joblib.dump(wine_cluster_model, 'wine_quality_clustering_model.pkl')
# joblib.dump(lr_model, 'wine_quality_linear_regression_model.pkl')
```

1.7.5 Conclusion

In this experiment, we evaluated two approaches—**K-Means Clustering** and **Linear Regression**—for predicting wine quality.

- **K-Means Clustering:** Despite being an unsupervised learning technique, K-Means struggled to provide a meaningful prediction of wine quality. The performance did not significantly improve as the number of clusters increased, with an accuracy of 52.32% and an MAE of 0.5426. Clustering is less effective in this context because wine quality is a continuous variable, making it challenging to map clusters directly to the quality labels.
- **Linear Regression:** Linear Regression outperformed K-Means with an accuracy of 53.26% and a lower Mean Absolute Error (MAE) of 0.5201. As a supervised learning method, it directly models the relationship between the features and the target variable, providing better results for predicting continuous values like wine quality.

1.7.6 Data Preprocessing

The dataset was relatively clean, with no missing values and no significant outliers. We performed standard scaling of features to ensure they were on a similar scale, which is particularly important for both clustering and regression models. Data preprocessing steps such as scaling and removing duplicates ensured that the models received high-quality inputs for training.

However, further improvements in data preprocessing could involve:

- **Feature Engineering:** Exploring additional feature creation, such as interaction terms or domain-specific transformations, might help improve model performance.
- **Handling Imbalanced Classes:** The target variable (wine quality) appeared to have a normal distribution, and very low number of samples of low quality and high quality wines.

1.7.7 Further Work

To enhance the predictive performance of the models, future work could include: - **Exploring Other Models:** For this dataset a deep neural network would be a good place to start to consider complex interactions. - **Adding Features:** For this dataset, perhaps there are additional attributes of wine which could help predict its quality. - **Hyperparameter Tuning:** For both the K-Means and Linear Regression models, further optimization (e.g., tuning the number of clusters for K-Means or number of iterations) could lead to improved accuracy.

In summary, the low performance of **K-Means Clustering** implies that similar feature vectors maybe mapped to multiple quality scores. This could possibly indicate that the quality ratings may depend on the person(s) determining the quality ratings. Moreover, if the given dataset was a complete parametric description of wine, then we can confirm the prior statement.

[]: