A Report On: -

Clustering Analysis of Wine Dataset Using K-Means

Submitted By: -

Rajan Ghimire
C0924991

Submitted To: -

Victoria Shtern

Contents

Abstract

This project aims to analyze the Wine dataset from the UCI Machine Learning Repository using the K-Means clustering algorithm. The results are visualized in both **2D** and **3D** plots to provide clear insights into the clustering patterns.

Introduction:

Clustering is a fundamental technique in machine learning used to group similar data points together based on their features. This project focuses on the Wine dataset, which contains chemical analysis results of wines grown in the same region in Italy. By applying the K-Means clustering algorithm, we aim to uncover natural groupings within the data.

Objectives:

- The project's goal is to identify the optimal number of clusters and evaluate the clustering performance through visualization.

About the Dataset:

The dataset used in this study is the Wine Quality Dataset from the UCI Machine Learning Repository. It contains 1,143 samples of red "Vinho Verde" wine, each described by 11 physicochemical properties and a quality rating. The features and their descriptions are : 'fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar', 'chlorides, 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density', 'ph', 'sulphates', 'alcohol', 'quality'.

Data Validation:

Duplicate Values:

```
print("\nNumber of duplicated rows : ", df.drop(columns=['id']).duplicated().sum(),"\n")
```

```
Number of duplicated rows :  125
```

There were **125** duplicate values in the dataset. And the total percentage of duplicate values was **10%** and has been removed in the final dataset.

Valid Data Types:

```
Incorrect df types:
None
Data is correct with following:
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
id                        int64
dtype: object
```

There were no Incorrect data, and all data was correct and was in correct format.

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
id                      0
```

There are no missing values in the dataset.

Overall, besides some duplicate values, data was valid, and we can proceed with the other steps.

Data Preparation:

Feature Selection:

Feature selection involves selecting a subset of relevant features (variables, predictors) for use in model construction. By doing so, we aim to improve the model's performance and reduce computational costs. In our case, I've selected the following features from the dataset:

```
features = ['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'ph', 'sulphates', 'alcohol']

X = df[features]
```

These features are numerical attributes of the wine that are relevant for clustering.

StandardScaler

StandardScaler is used to standardize features by removing the mean and scaling to unit variance. This is an important preprocessing step for many machine learning algorithms, especially those that rely on distance metrics like K-Means clustering.

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

Mathematically:

$$Z_i = \frac{X_i - \mu_i}{\sigma_i}$$

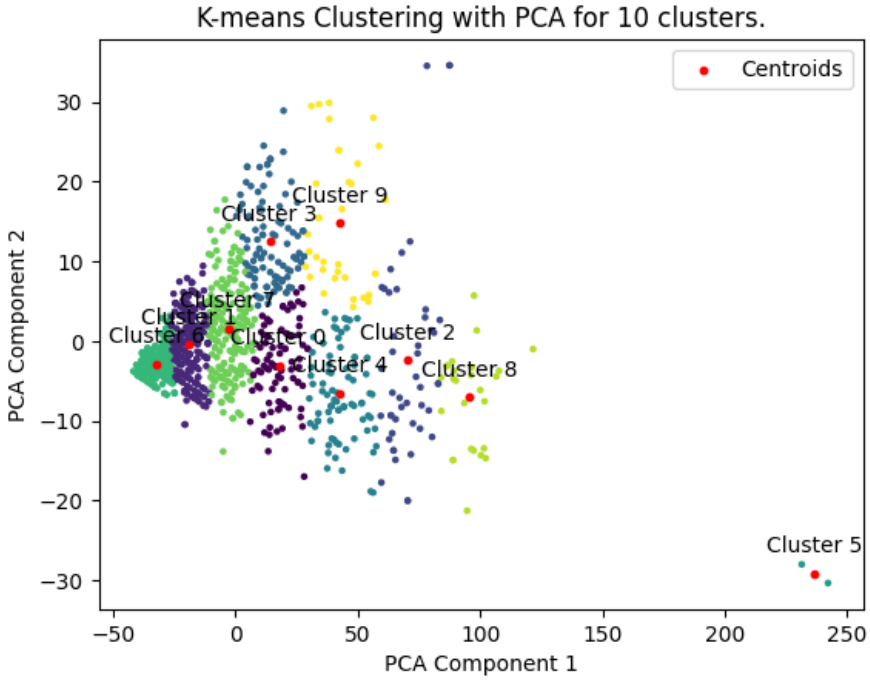
where:

- X_i is the original feature.
- μ_i is the mean of the feature X_i.
- σ_i is the standard deviation of the feature X_i.

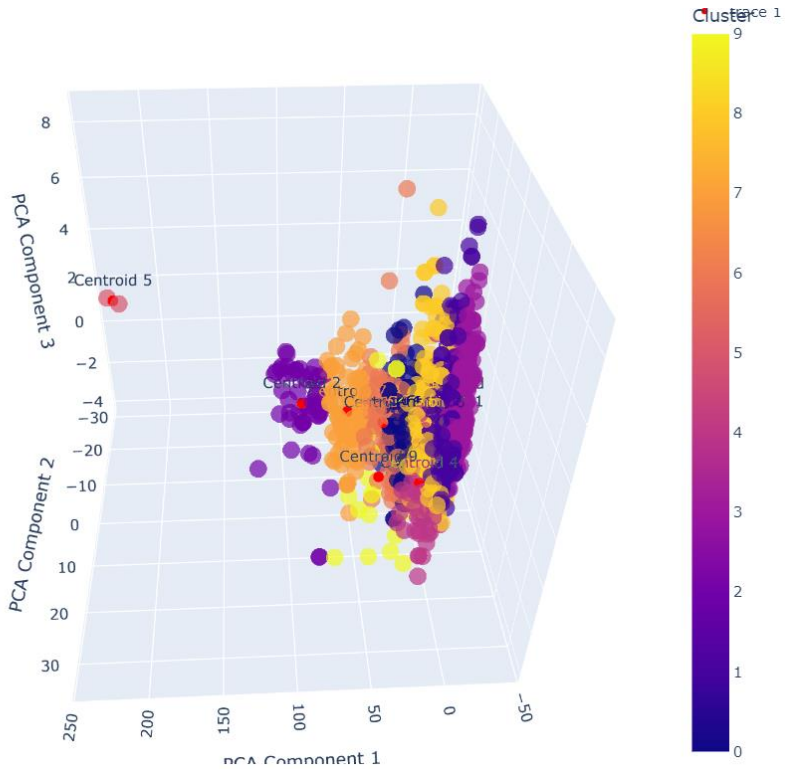
K-Means Cluster Evaluation:

For 10 cluster (K=10)

In 2D:



In 3D:

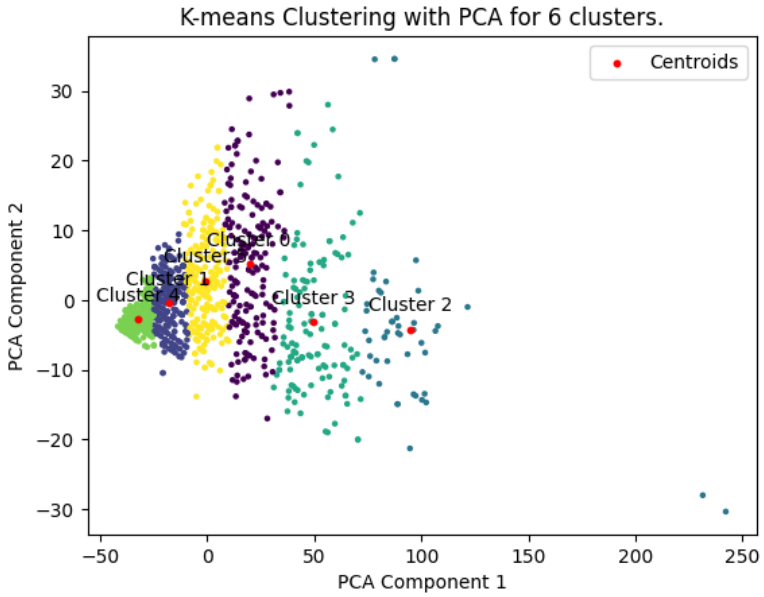


K-means with 10 Clusters:

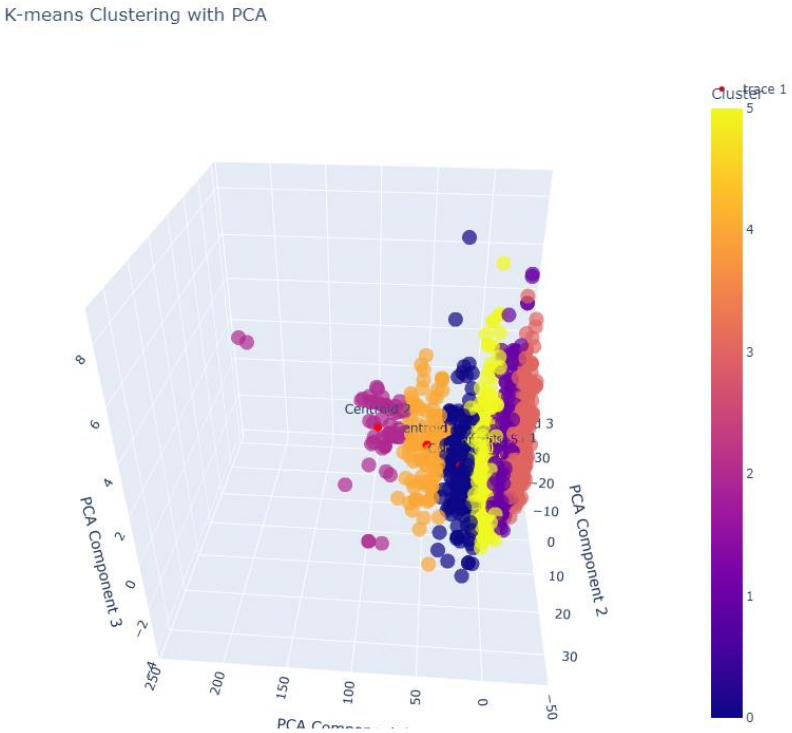
- **Shape:** The clusters are less distinct and have more irregular shapes.
- **Size:** The sizes of the clusters are smaller and more evenly distributed, but this may lead to some overfitting as some clusters appear to be too small and might not capture significant groupings.
- **Density:** The clusters are denser, but there is significant overlap, indicating possible over-segmentation.

For 6 cluster (K=6)

In 2D:



In 3D:

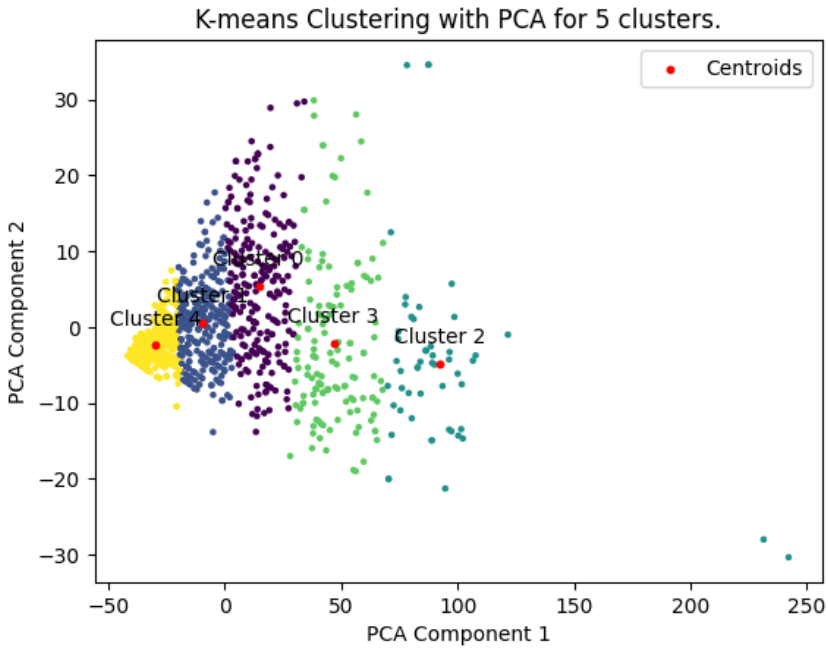


K-means with 6 Clusters:

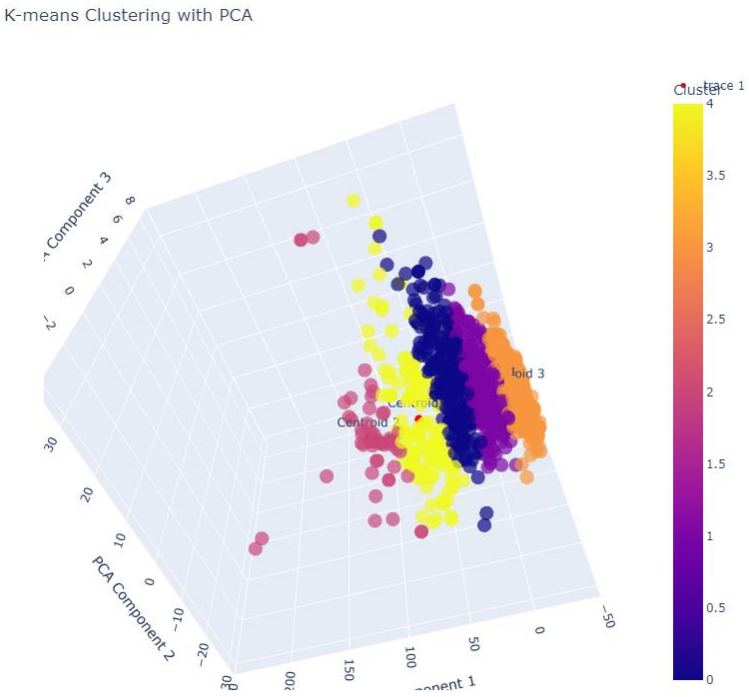
- **Shape:** The clusters are relatively well-formed with clear distinctions.
- **Size:** The sizes of the clusters are more balanced compared to the 10-cluster solution.
- **Density:** The clusters maintain a good density and compactness, with moderate overlap.

For 5 cluster (K=5)

In 2D:



In 3D:

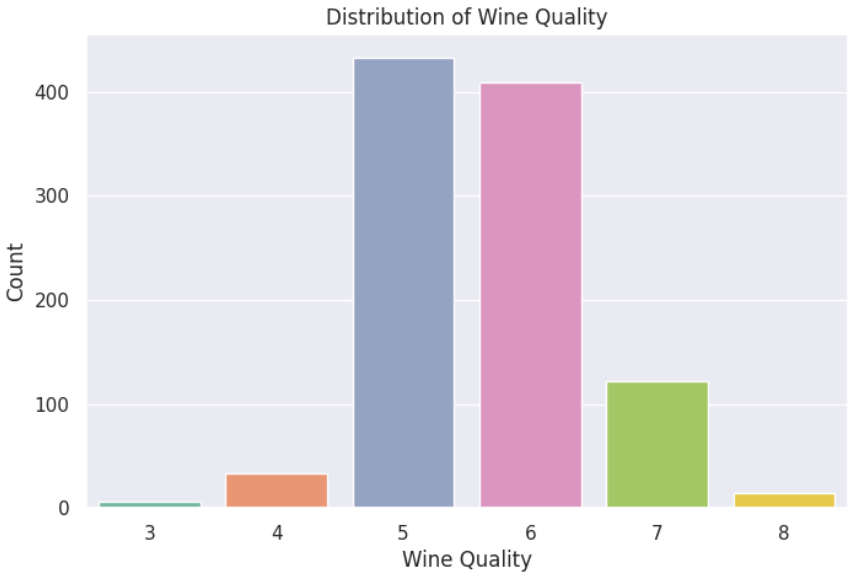


K-means with 5 Clusters:

- **Shape:** The clusters are relatively well-formed with distinct shapes.
- **Size:** The sizes of the clusters vary, but there is a reasonable distribution among the clusters.
- **Density:** The clusters are quite dense and compact, with minimal overlap.

Validity of K-Means:

If we look at the total number of classes in the given dataset:



It seems there are 6 total classes and data can be classified clustered into 6 classes. K = 6 showed the best result. This means that our clustering is valid.

Conclusion:

Based on the analysis, the most optimal number of clusters appears to be **6**. This selection balances well-formed cluster shapes, reasonably sized clusters, and good density, minimizing overlap and ensuring distinct groupings.

The 5-cluster solution also shows promise with well-formed and dense clusters, but the 6-cluster solution provides slightly better balance and less variance in cluster size, making it a more optimal choice for capturing the underlying structure of the data. The 10-cluster solution, while more segmented, leads to over-segmentation with irregular shapes and significant overlap, making it less optimal.

Appendix:

Code Used for K-Means and 2D plot:

```
CLUSTER_NUMBER = 5

kmeans = KMeans(n_clusters=CLUSTER_NUMBER, random_state=42, n_init="auto")
kmeans.fit(X)

# Get cluster labels
label = kmeans.labels_
# Add the cluster labels to the original dataframe
df['cluster'] = label
centroids = kmeans.cluster_centers_

# 2D Plot
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
centroids_pca = pca.transform(centroids)

# Plot the clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['cluster'], cmap='viridis', s=5)
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.title(f'K-means Clustering with PCA for {CLUSTER_NUMBER} clusters.')

# Annotate the clusters
for i, centroid in enumerate(centroids_pca):
    plt.annotate(f'Cluster {i}', (centroid[0], centroid[1]),
                 textcoords="offset points", xytext=(0,10), ha='center')


# Plot the centroids
plt.scatter(centroids_pca[:, 0], centroids_pca[:, 1], s=10, c='red', label='Centroids')
plt.legend()
plt.show()
```

For 3D plot:

```
# Run K-Means clustering
kmeans = KMeans(n_clusters=CLUSTER_NUMBER, random_state=42)
kmeans.fit(X_pca)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

# Create a DataFrame for Plotly
df_pca = pd.DataFrame(X_pca, columns=['PCA Component 1', 'PCA Component 2', 'PCA Component 3'])
df_pca['Cluster'] = labels

# Plot the 3D scatter plot
fig = px.scatter_3d(
    df_pca,
    x='PCA Component 1',
    y='PCA Component 2',
    z='PCA Component 3',
    color='Cluster',
    title='K-means Clustering with PCA',
    opacity=0.7
)

# Add centroids
fig.add_trace(
    go.Scatter3d(
        x=centroids[:, 0],
        y=centroids[:, 1],
        z=centroids[:, 2],
        mode='markers+text',
        marker=dict(size=5, color='red'),
        text=['Centroid {}'.format(i) for i in range(len(centroids))],
        textposition='top center'
    )
)

fig.update_layout(height=800,width=800)

fig.show()
```