Assignment No. 5

Problem Statement: Implement the K-Means clustering algorithm using Python to analyze and visualize a given dataset.

Objective:

1. Understand and implement the K-Means clustering algorithm.
2. Apply K-Means to a dataset and analyze the results.
3. Use the Elbow Method to determine the optimal number of clusters.
4. Visualize clusters and interpret results.

Prerequisite :

1. A Python environment with essential libraries like pandas, numpy, matplotlib, seaborn, and scikit-learn.
2. Basic knowledge of Python, statistics, and machine learning principles.
3. Statistics: Understanding of mean, variance, and standard deviation.
4. Machine Learning: Basics of unsupervised learning, clustering, and K-Means.

Theory :

K-Means Clustering **is** an unsupervised machine learning algorithm used for partitioning a dataset into K distinct, non-overlapping clusters**.** It aims to group similar data points together while ensuring that different clusters are as distinct as possible.

Key Properties of K-Means:

1. **Unsupervised learning**: No labeled data is required.
2. **Centroid-based**: Each cluster is represented by its centroid (mean of the points).
3. **Iterative algorithm**: Repeatedly assigns data points and updates centroids.
4. **Works well with large datasets**: Faster than hierarchical clustering.

Working of K-Means Algorithm

Step 1: Choose K (Number of Clusters)

- Decide the number of clusters KKK manually or using methods like the **Elbow Method** or **Silhouette Score**.

Step 2: Initialize Centroids

- Randomly select **K data points** as the initial **cluster centroids**.

Step 3: Assign Each Data Point to the Nearest Centroid

- Compute the **Euclidean distance** between each data point and the centroids:

$$d(x, c) = \sqrt{\sum (x_i - c_i)^2}$$

- Assign each data point to the **closest centroid**.
- This step **forms K clusters**.

Step 4: Compute New Centroids

- For each cluster, calculate the **new centroid** by taking the **mean of all points** assigned to it:

$$C_j = \frac{1}{N_j} \sum_{i=1}^{N_j} x_i$$

where:

- Cj = New centroid of cluster j
- Nj = Number of points in cluster j
- xi = Data points in cluster j

Step 5: Repeat Until Convergence

- **Reassign points** to the new centroids.
- **Recalculate centroids**.
- Repeat **until centroids no longer change** (or changes are minimal).

3. Understanding WCSS (Within-Cluster Sum of Squares)

The **WCSS (Within-Cluster Sum of Squares)** measures how well data points fit within a cluster. It is used in the **Elbow Method** to find the optimal number of clusters.

Formula for WCSS:

$$WCSS = \sum_{j=1}^{K} \sum_{i=1}^{N_j} (x_i - C_j)^2$$

where:

- K= Number of clusters,
- Nj = Number of points in cluster j
- xi = Data points in cluster j
- Cj = Centroid of cluster

Interpreting WCSS

- **Lower WCSS** = Clusters are well-defined and compact.
- **Higher WCSS** = Clusters are too spread out (not well-defined).

4. The Elbow Method to Find Optimal K
The Elbow Method helps determine the best value of K by plotting **WCSS vs. K**.

How to Use the Elbow Method?

1. Compute **WCSS for different values of K**.
2. Plot **WCSS vs. K**.
3. Find the **"elbow point"** where WCSS stops decreasing sharply.
4. The corresponding **K is optimal**.

Why Does the Elbow Method Work?

- Adding more clusters **always decreases WCSS**.
- But after a certain **K**, **adding more clusters has little effect**.
- The **"elbow"** is where **WCSS reduction slows down significantly**.

5. Evaluation Metrics for K-Means
Silhouette Score

Measures how well a point fits within its cluster:

$$S = \frac{b - a}{\max(a, b)}$$

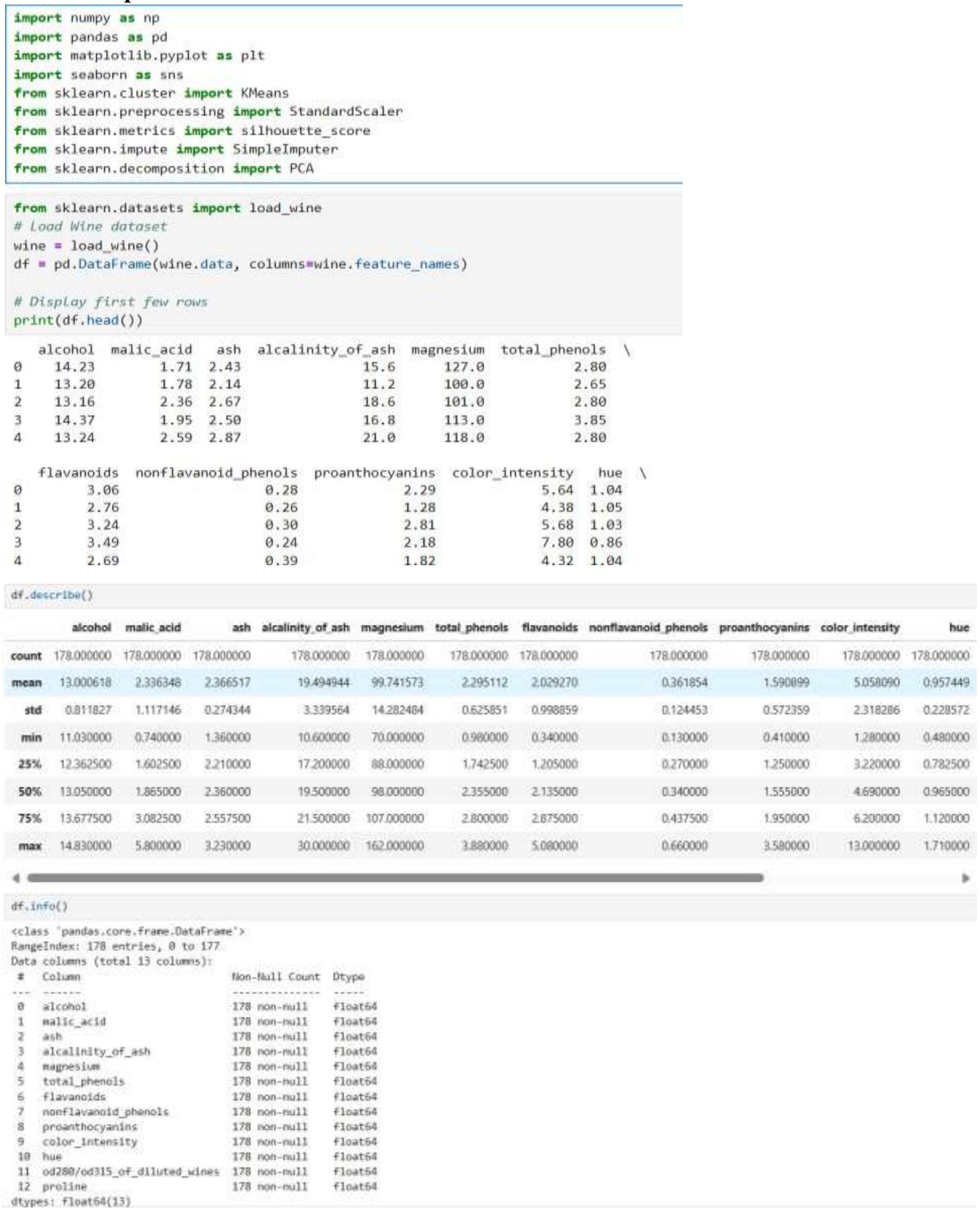
where:

- a = Average distance to other points in the same cluster.
- b = Average distance to points in the nearest cluster.

Interpretation:

- **S ≈ 1:** Well-clustered
- **S ≈ 0:** Overlapping clusters
- **S ≈ -1:** Wrong clustering

Code & Output



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
```

```python
from sklearn.datasets import load_wine
# Load Wine dataset
wine = load_wine()
df = pd.DataFrame(wine.data, columns=wine.feature_names)

# Display first few rows
print(df.head())
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80   
1    13.20        1.78  2.14               11.2      100.0           2.65   
2    13.16        2.36  2.67               18.6      101.0           2.80   
3    14.37        1.95  2.50               16.8      113.0           3.85   
4    13.24        2.59  2.87               21.0      118.0           2.80   

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04   
1        2.76                  0.26             1.28             4.38  1.05   
2        3.24                  0.30             2.81             5.68  1.03   
3        3.49                  0.24             2.18             7.80  0.86   
4        2.69                  0.39             1.82             4.32  1.04   
```

```python
df.describe()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
dtypes: float64(13)
```

```python
df.isnull().sum()
```

```
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
dtype: int64
```

```python
#dataset has no missing values
imputer = SimpleImputer(strategy='mean')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

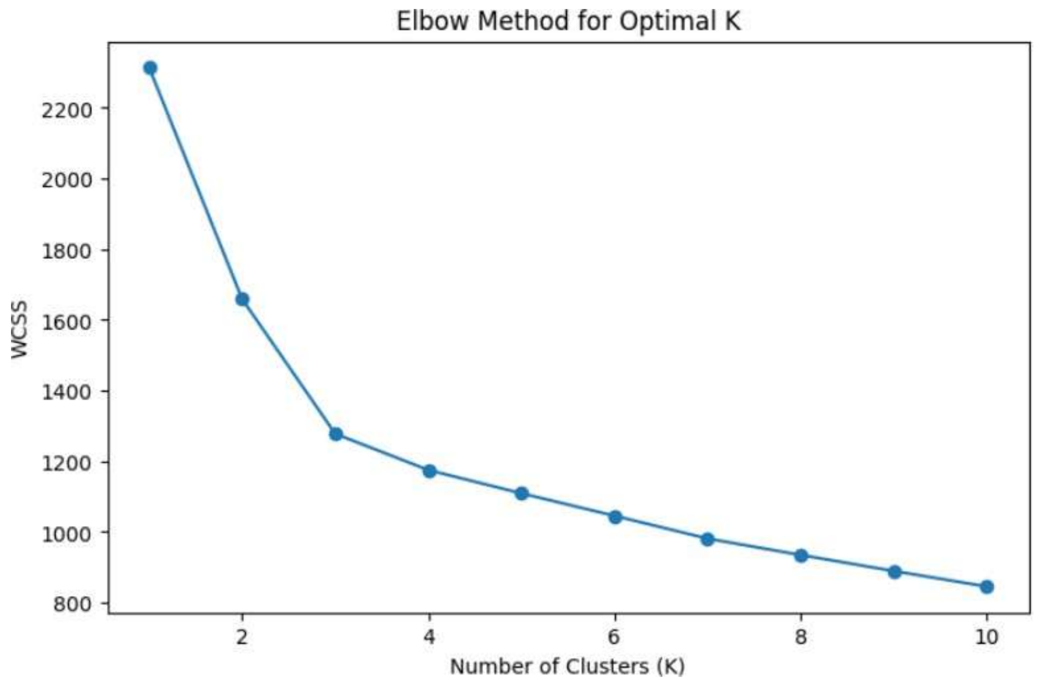
```python
# K-Means is sensitive to different feature scales, so we use StandardScaler
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

```python
# To visualize clusters in 2D, we reduce dimensions using PCA
pca = PCA(n_components=2)
df_pca = pd.DataFrame(pca.fit_transform(df_scaled), columns=['PC1', 'PC2'])
```

```python
# Determine Optimal K Using the Elbow Method
wcss = []  # Within-cluster sum of squares
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(K_range, wcss, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal K')
plt.show()
```



```
# The elbow point helps determine the optimal number of clusters.
```

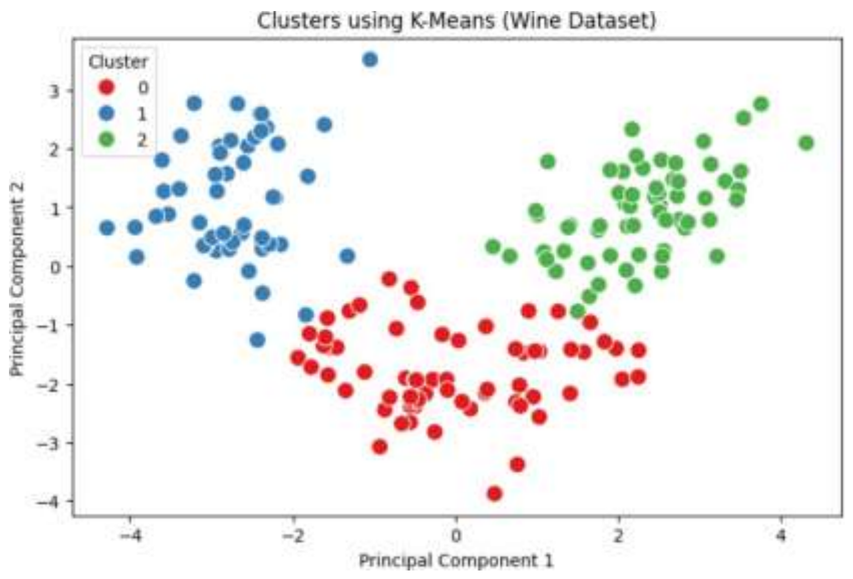
```
# Choosing optimal K (3 based on Elbow Method)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(df_scaled)

# Compute Silhouette Score
sil_score = silhouette_score(df_scaled, df['Cluster'])
print(f'Silhouette Score: {sil_score:.2f}')
```

```
Silhouette Score: 0.28
```

```
# silhouette Score > 0.5 means the clustering is good.
```

```
# Scatter plot of clusters after PCA
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df_pca['PC1'], y=df_pca['PC2'],
                hue=df['Cluster'], palette='Set1', s=100)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Clusters using K-Means (Wine Dataset)')
plt.legend(title='Cluster')
plt.show()
```



```
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score

# Compute clustering evaluation metrics
sil_score = silhouette_score(df_scaled.drop(columns=['Cluster']), df_scaled['Cluster'])

print("\nK-Means Clustering Evaluation:")
print(f"Silhouette Score: {sil_score:.3f} (Higher is better)")

# ------------------------- KNN CLASSIFICATION EVALUATION -------------------------
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns=['target']), df['target'], test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=5)  # Choosing k=5
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Compute classification evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)

print("\nK-Nearest Neighbors (KNN) Evaluation:")
print(f"Accuracy: {accuracy:.3f} (Higher is better)")
print(f"Precision: {precision:.3f} (Higher is better)")
print(f"Recall: {recall:.3f} (Higher is better)")
print(f"F1 Score: {f1:.3f} (Higher is better)")
print("Confusion Matrix:\n", cm)
```

```
K-Means Clustering Evaluation:
Silhouette Score: 0.285 (Higher is better)

K-Nearest Neighbors (KNN) Evaluation:
Accuracy: 0.722 (Higher is better)
Precision: 0.722 (Higher is better)
Recall: 0.722 (Higher is better)
F1 Score: 0.722 (Higher is better)
Confusion Matrix:
 [[12  0  2]
 [ 0 11  3]
 [ 2  3  3]]
```

Github: https://github.com/Pratik-Gadekar123/ML

Conclusion:
In this assignment, we implemented **K-Means clustering** and **KNN classification** on the **Wine dataset**. The **K-Means model** resulted in a **Silhouette Score of 0.285**, indicating weak cluster separation. The **KNN classifier achieved 72.2% accuracy**, with balanced precision, recall, and F1-score. However, the confusion matrix showed misclassifications, particularly in class 2.