coursera-clustering

October 9, 2024

1 IBM Unsupervised Machine Learning: Clustering

- Pedro Antonio Vázquerz González

This project was developed as a continuation of a personal challenge where I programmed the K-Means algorithm from scratch using Python. You can find the original project in this GitHub repository.

That project was part of a personal practice aimed at implementing K-Means using only the Numpy library. It served as a review of the knowledge I gained at the University of Guanajuato and from the IBM Machine Learning course. Throughout the development process, I tested my implementation with various datasets from the Scikit-Learn library to assess its performance. One of the key challenges I encountered was the difficulty of forming effective clusters when working with more than two features. To address this issue, I initially simplified the datasets by manually selecting the most relevant features based on the context and characteristics of the data. However, I always wondered whether there was a more efficient way to select features, minimizing information loss or controlling it more precisely.

Later, while taking IBM's Machine Learning: Unsupervised Learning course, I learned about techniques such as Principal Component Analysis (PCA), which can automate the feature selection process. This motivated me to create a new Jupyter Notebook where I applied PCA, with the goal of improving K-Means clustering results compared to my earlier manual method. To maintain consistency in the visualization of results, I reused the plotting functions from the original project, but moved them to a separate .py file to keep the Notebook focused on the core analysis.

1.0.1 Used libraries

In this second part, I am going to use Scikit-Learn's K-Means algorithm to streamline the workflow.

[1]:
```
import numpy as np
import pandas as pd
import helper_functions as hf  # helper functions
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.cluster import KMeans
from sklearn.datasets import load_breast_cancer, load_wine, load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```
warnings.filterwarnings('ignore')
```

1.1 Let's begin the comparison with Wine

In this code, we first load the data from the "*wine*" dataset, which contains several characteristics that describe different types of wine, such as its acidity, alcohol content, among other properties. This data is stored in the variable `X_wine`, while the names of those characteristics are stored in `wine_features`, which allows us to know what each one represents.

Datasets like Wine or Breast Cancer are excellent choices to test the effectiveness of dimensionality reduction combined with K-Means for clustering. Both datasets contain multiple features representing various measurements, and not all of them may be equally relevant for distinguishing between classes. This is where methods like PCA become particularly useful, as they allow us to reduce the number of dimensions while retaining the most important information. Additionally, by reducing the dimensions, we can better visualize the results in 2D, making it easier to interpret the formed clusters. Evaluating clustering through inertia —which measures how compact the clusters are— allows us to determine whether dimensionality reduction improves the quality of clustering.

[2]:
```
X_wine = load_wine().data
wine_features = load_wine().feature_names
```

Then, we apply a technique called "standardization", which is important to ensure that all features are on the same scale. This is critical when using algorithms such as Principal Component Analysis (PCA), because if we don't standardize them, some features with larger values (such as alcohol content) could have a disproportionate influence on the results, while others with smaller values could go unnoticed. Therefore, we transform the data with the StandardScaler() method, so that all characteristics have the same importance when performing the analysis.

[3]:
```
X_wine_std = StandardScaler().fit_transform(X_wine)
```

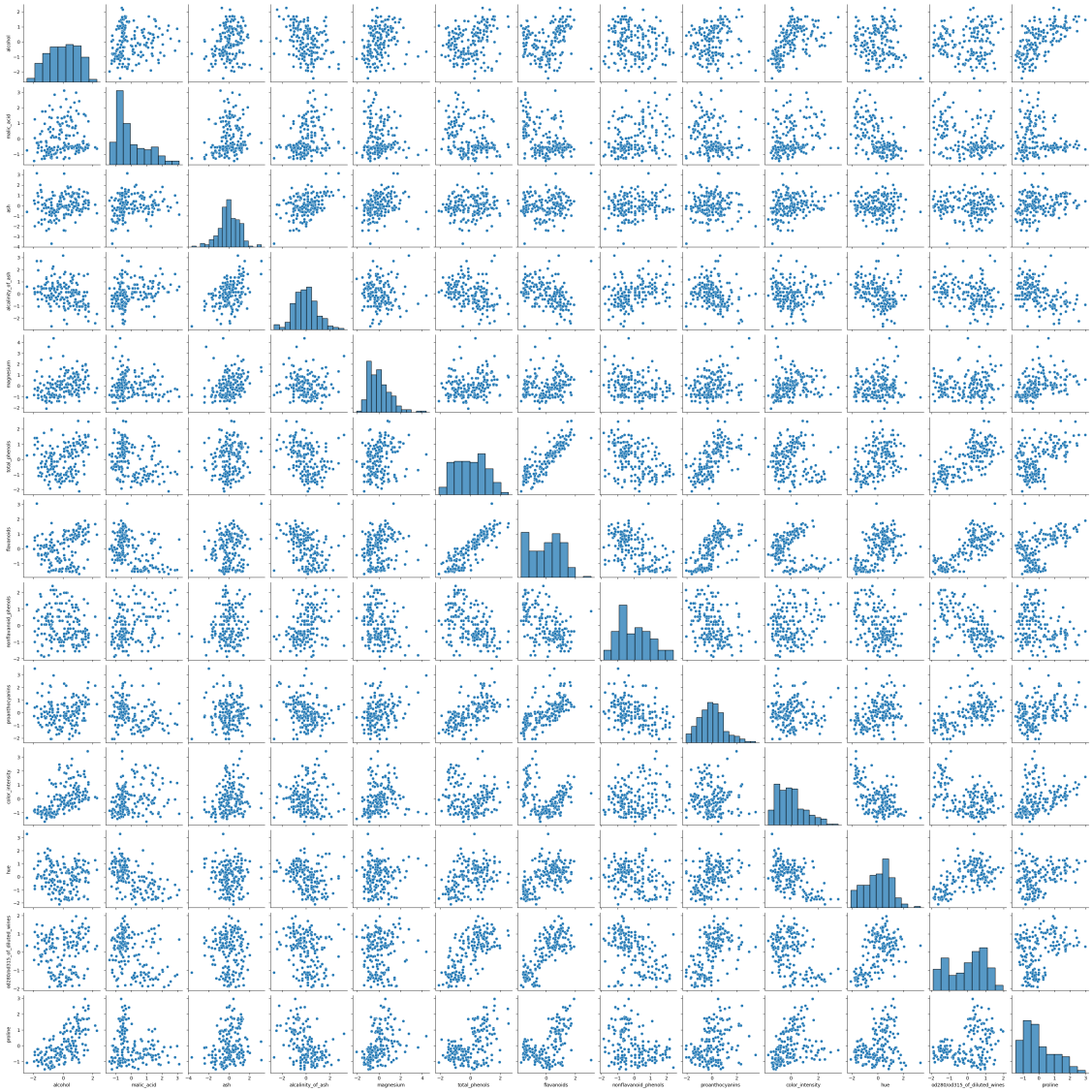
1.1.1 Original Wine Plotting 2D

We observe that we have an unfavorable distribution to perform 2-D clustering using the original dataset characteristics. This is where we test the PCA to determine how favorable it is to simplify the information and allow us to take the most relevant information without losing information and improve its visualization.

[4]:
```
df_wine_std = pd.DataFrame(X_wine_std, columns=wine_features)
```

[5]:
```
sns.pairplot(df_wine_std)
```

[5]:
```
<seaborn.axisgrid.PairGrid at 0x168a89fd0>
```



1.2 Aplying PCA

PCA is a technique that reduces the number of variables or features in the data while preserving as much original information as possible. This is achieved by identifying the principal components, which are combinations of the original features that retain the majority of the variance in the data. By reducing dimensions, we simplify analysis and processing without sacrificing significant amounts of information.

The following functions are adaptations of the code seen in the course in order to visualize the effects of the PCA on datasets and to be able to visualize it easily.

[6]:
```
def explain_variance(X, n_components, features_df):
    pca_list = []
```

```
    feature_weight_list = []

    for n in range(1, n_components + 1):
        PCA_model = PCA(n_components=n)
        PCA_model.fit(X)

        pca_list.append(
            pd.Series(
                {
                    "n": n,
                    "model": PCA_model,
                    "var": PCA_model.explained_variance_ratio_.sum(),
                }
            )
        )

        abs_feature_values = np.abs(PCA_model.components_).sum(axis=0)
        feature_weight_list.append(
            pd.DataFrame(
                {
                    "n": n,
                    "features": features_df,
                    "values": abs_feature_values / abs_feature_values.sum(),
                }
            )
        )

    pca_df = pd.concat(pca_list, axis=1).T.set_index("n")

    features_df = pd.concat(feature_weight_list).pivot(
        index="n", columns="features", values="values"
    )

    return pca_df, features_df
```

[7]:
```
def plot_variance(pca_df, ds_name=None):
    sns.set_theme(context="notebook", style="whitegrid")
    ax = pca_df['var'].plot(kind='bar', colormap='viridis')
    ax.set(xlabel='Number of components',
        ylabel='Percent explained variance',
        title=f'Explained Variance vs Dimensions {ds_name}');
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

def plot_features(features_df, ds_name=None):
    sns.set_theme(context="notebook", style="whitegrid")
```

```
    ax = features_df.plot(kind='bar', colormap='viridis', figsize=(13, 8))
    ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1))
    ax.set(xlabel='Number of components',
        ylabel='Relative importance',
        title=f'Feature importance vs Dimensions {ds_name}');
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()
```

[8]:
```
pca_df, features_df = explain_variance(X_wine_std, 5, wine_features)
```

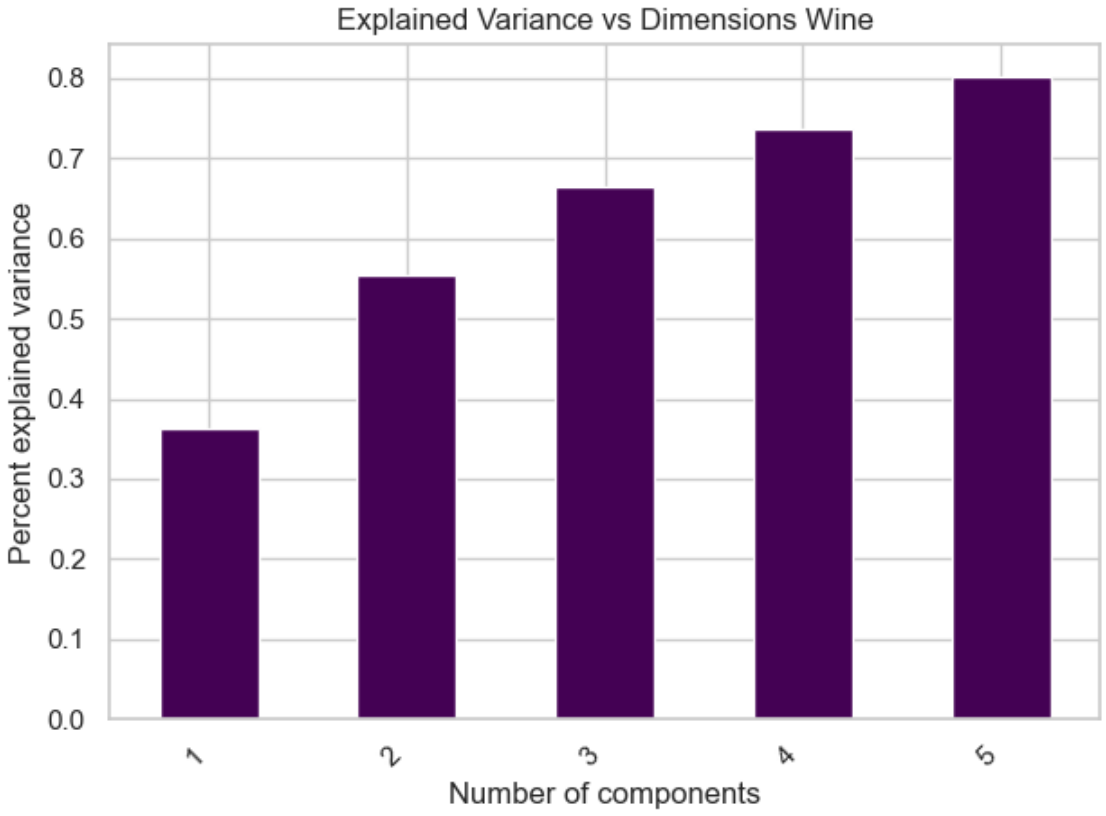
[9]:
```
features_df
```

[9]:
```
features  alcalinity_of_ash   alcohol       ash  color_intensity  flavanoids  \
n
1                  0.073621  0.044400  0.000631         0.027261    0.130106
2                  0.041235  0.103616  0.052489         0.102070    0.070338
3                  0.097311  0.094305  0.106608         0.085336    0.065135
4                  0.077536  0.071685  0.097336         0.069049    0.061272
5                  0.067996  0.076929  0.089488         0.061762    0.057637

features       hue  magnesium  malic_acid  nonflavanoid_phenols  \
n
1         0.091277   0.043681    0.075426              0.091837
2         0.095031   0.072867    0.077569              0.054006
3         0.074640   0.064617    0.063121              0.056184
4         0.091490   0.077647    0.092085              0.058895
5         0.086808   0.113531    0.077778              0.082622

features  od280/od315_of_diluted_wines  proanthocyanins   proline  \
n
1                             0.115719         0.096419  0.088213
2                             0.089208         0.058200  0.107522
3                             0.079777         0.056692  0.087875
4                             0.074842         0.075720  0.084897
5                             0.068202         0.071375  0.080330

features  total_phenols
n
1              0.121408
2              0.075850
3              0.068398
4              0.067546
5              0.065542
```

The Explained Variance graph shows how much information (or variance) is retained as we reduce the dimensions. As we increase the number of principal components, we can see how many we need to capture most of the relevant information from the dataset. This is crucial for determining how

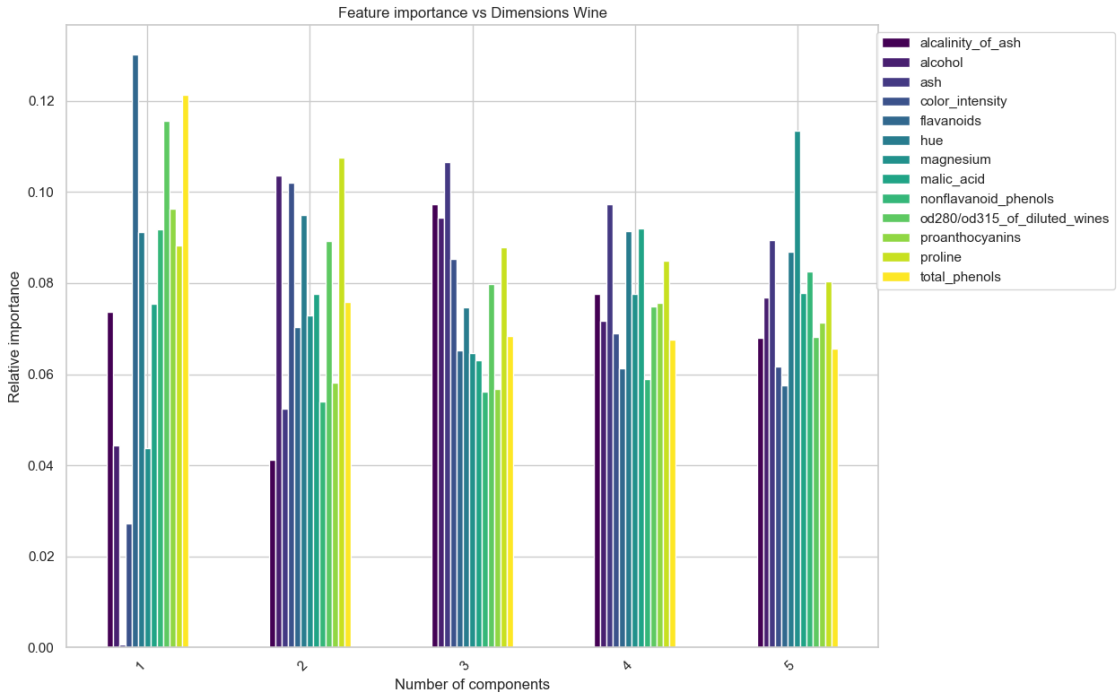
many dimensions to keep without losing too much information, thus optimizing the performance of the K-Means clustering.

[10]:
```
plot_variance(pca_df, 'Wine')
```



On the other hand, the Feature Importance graph illustrates how the importance of each original feature is redistributed across the principal components. This visualization is key to identifying which features have the most impact on the formation of clusters. By understanding how the features are weighted across the principal components, we can better evaluate which variables are crucial for the K-Means clustering process.

[11]:
```
plot_features(features_df, 'Wine')
```

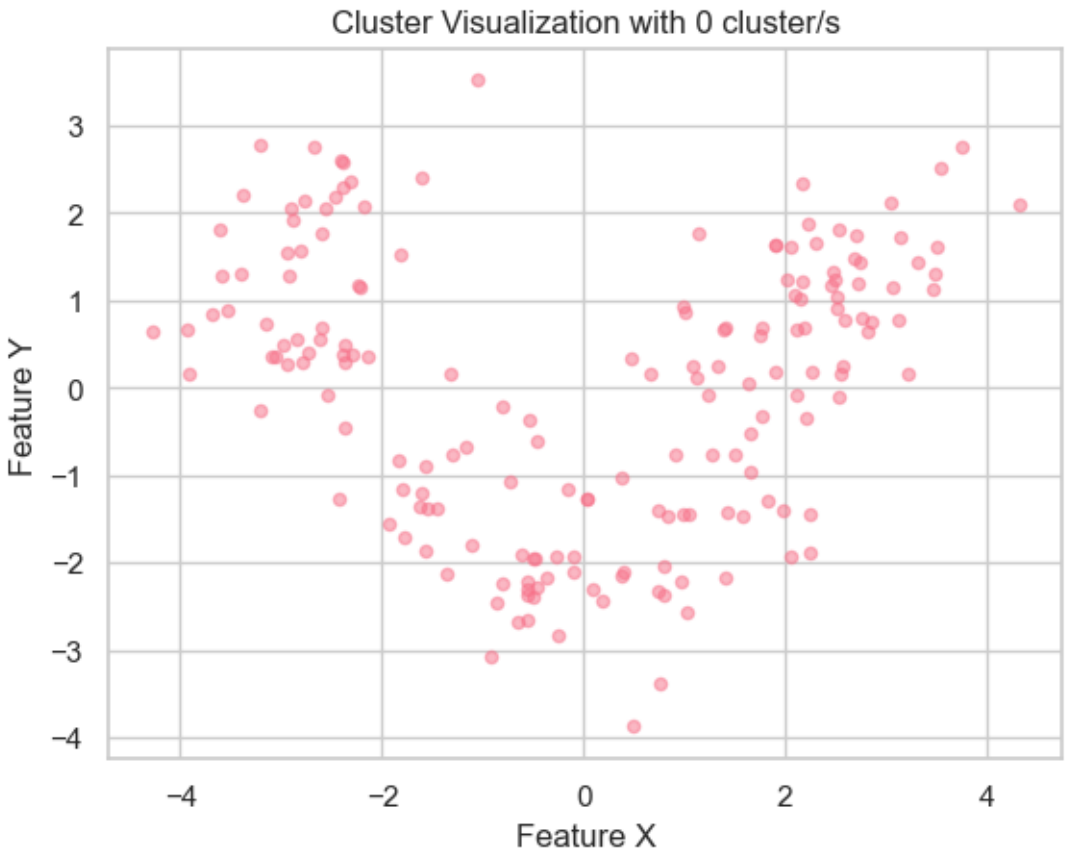


[12]:
```
# Aplicar PCA para reducir a 2 dimensiones
pca = PCA(n_components=2)
X_wine_pca = pca.fit_transform(X_wine_std)

print("Proportion of variance explained by each component:", pca.
  ↪explained_variance_ratio_)
```

```
Proportion of variance explained by each component: [0.36198848 0.1920749 ]
```

This graph provides a visual representation of how the number of principal components explains different proportions of the total variance in the dataset. As we add more components, the amount of variance they explain decreases, making it crucial to select the right number of components that best summarize the original dataset. In this case, the first two principal components explain 36.2% and 19.2% of the variance, respectively, meaning that together they capture about 55.4% of the total variance in the data.

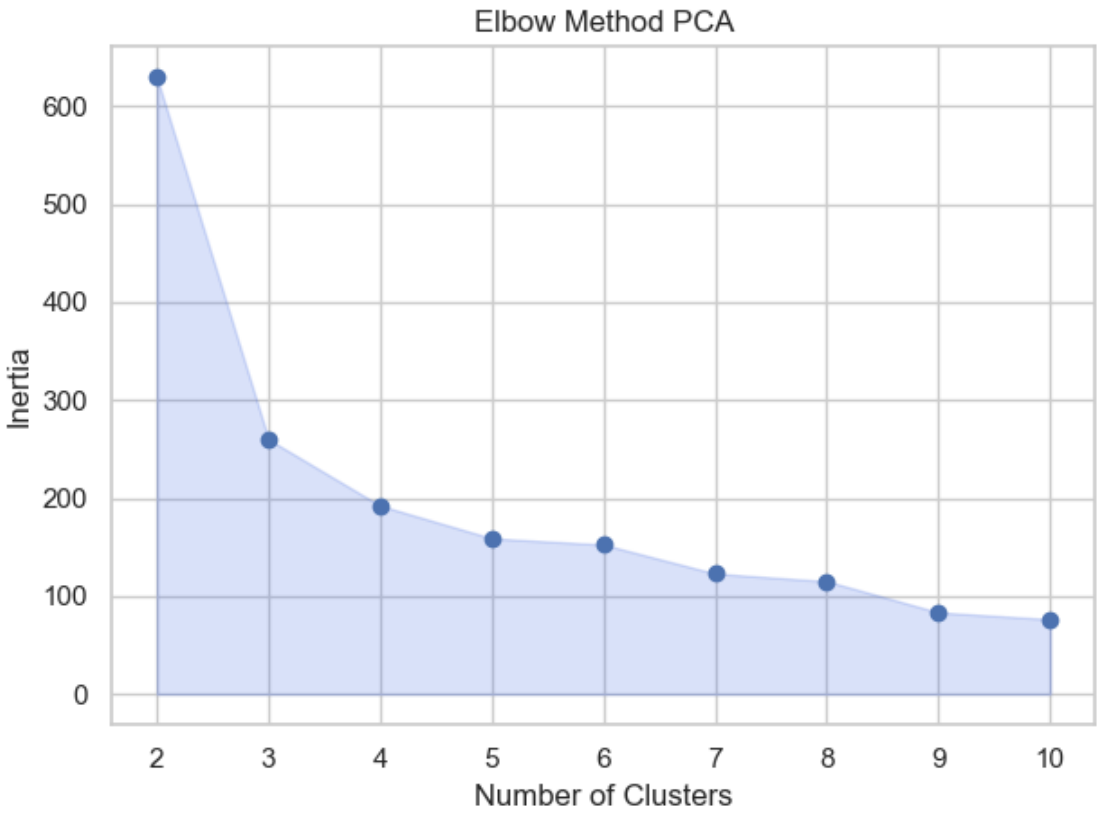
[13]:
```
hf.display_cluster(X_wine_pca)
```



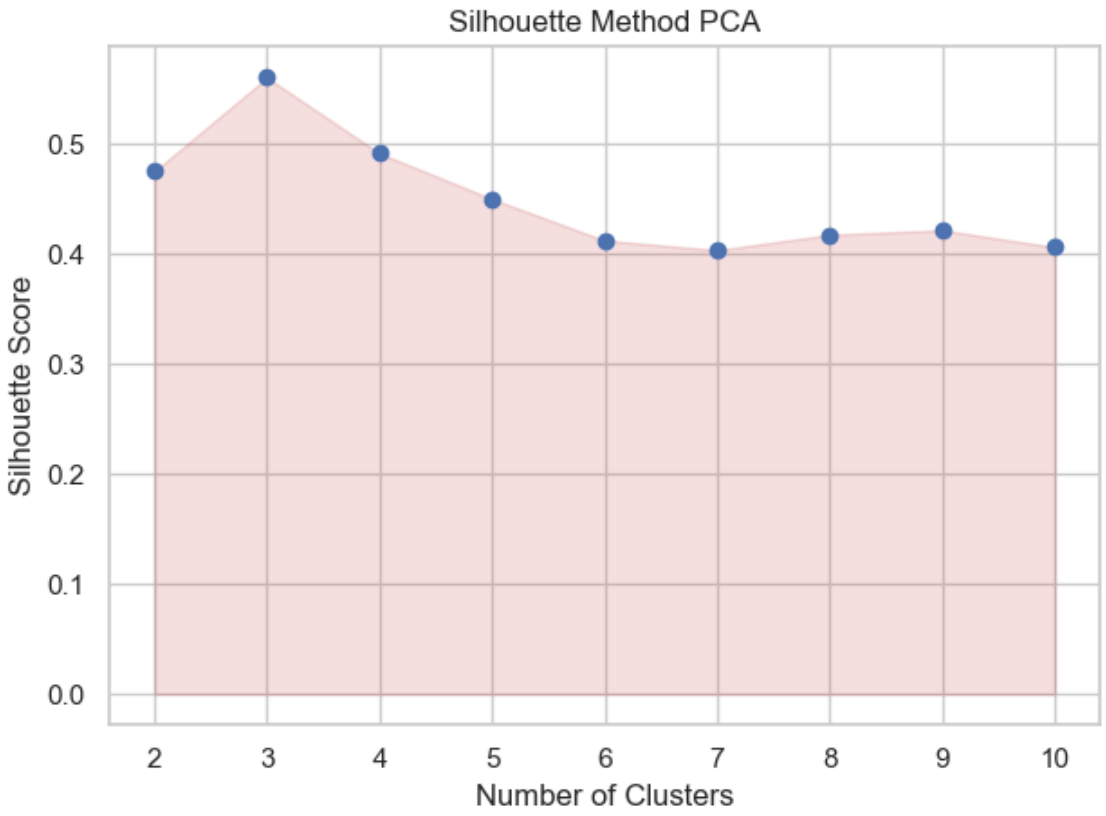
1.3 Clustering with K-Means for Wine PCA Dataset

We are conducting the elbow method and silhouette analysis to assess the effectiveness of the K-Means algorithm on the dataset reduced to 2 principal components using PCA. The goal is to determine the optimal number of clusters and evaluate how well the data is grouped after dimensionality reduction. The elbow method helps us identify how many clusters are sufficient to describe the data pattern, while the silhouette analysis measures the quality of these clusters, ensuring that each group is well-defined and separated from the others.

[14]:
```
hf.elbow_method(X=X_wine_pca, title="PCA")
```



[15]:
```
hf.silhouette_analysis(X_wine_pca, title="PCA")
```



For the Wine dataset, both the elbow method and silhouette analysis have correctly identified that 3 is the optimal number of clusters. This result matches the official information from the Wine dataset, which categorizes the wines into three different types based on their chemical properties. The alignment between the clustering results and the original classification confirms that the dimensionality reduction with PCA and the use of K-Means were effective in identifying clear patterns in the data.

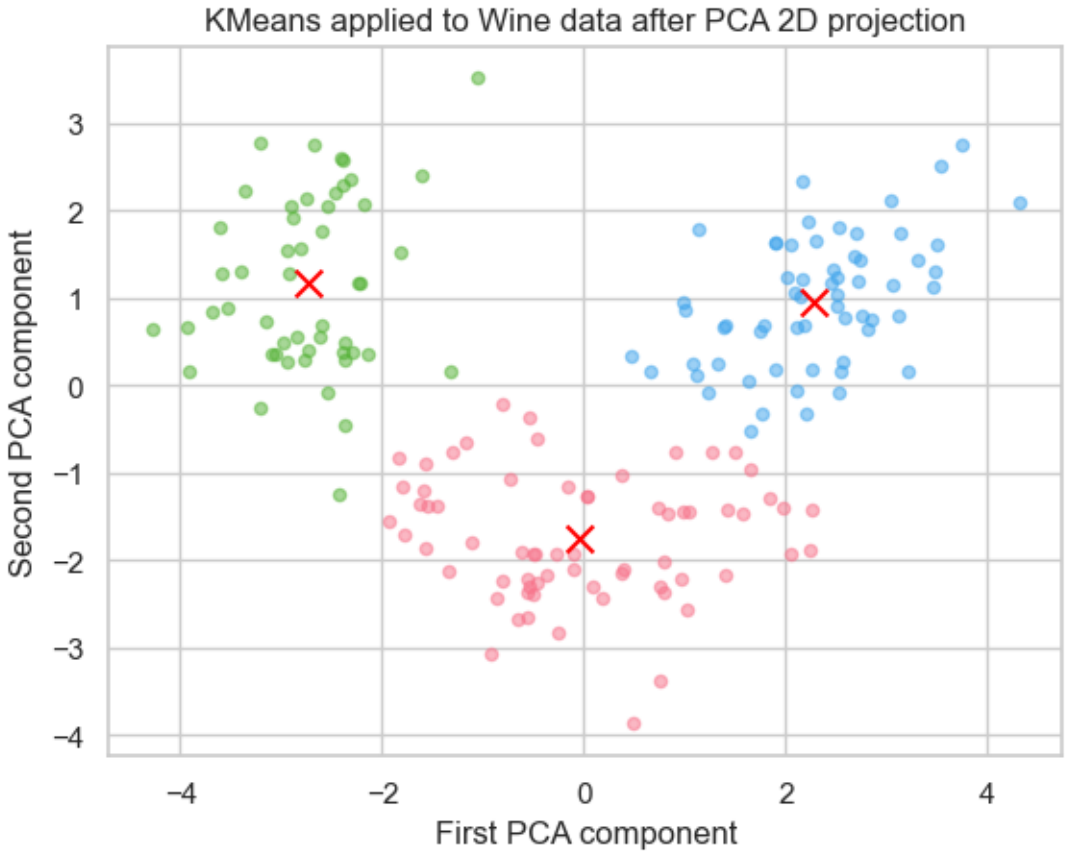
Compared to your version of K-Means and the tests performed in the first part of my lab. We can see that there is clearly a differentiation in the two graphs presented. For example, the inertia values are higher in both cases for the wine PCA dataset in the silhouette analysis, showing that the clusters are better differentiated.

1.3.1 Applying K-Means to PCA wine dataset

[16]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
predictions = kmeans.fit_predict(X_wine_pca)
```

[17]:
```
hf.display_cluster(
    X_wine_pca,
    model=kmeans,
    num_clusters=3,
```

```
    title="KMeans applied to Wine data after PCA 2D projection",
    feature_x="First PCA component",
    feature_y="Second PCA component",
)
```



1.4 Conclusion

The application of K-Means on the Wine dataset reduced with PCA confirms that it is much simpler and visually clearer to form 3 clusters using the principal components. By reducing the dimensions to just two principal components, we can clearly see how the data naturally groups into three distinct categories, which aligns with the official classification of the dataset. This improvement is evident when looking at the inertia and the silhouette analysis results, where the highest value, close to 0.6, reinforces that these two components are sufficient to effectively define the clusters.