Cluster Analysis with Python

A Basic Example using Polars

Jesus L. Monroy

Apr, 2025

Table of Contents

List of Figures

List of Tables

Cluster Analysis and Clustering

Cluster analysis encompasses the tools, algorithms, and methods used to uncover hidden groupings within a dataset based on similarity.

This process includes not only **clustering** —*the application of algorithms like k-means, DBSCAN, or hierarchical clustering to group data*— but also the subsequent analysis of the identified groups' characteristics and properties, ultimately informing decision-making in domains such as marketing (*customer and product segmentation*) and finance (*fraud detection*).

While clustering is a key step in cluster analysis, the terms are not synonymous. cluster analysis is the broader framework that includes interpretation and action based on the discovered groups.

This example is inspired in Machine Learning Mastery blog (Carrascosa 2024).

Carrascosa, Ivan Palomares. 2024. "Performing Cluster Analysis in Python: A Step-by-Step Tutorial." Posted on Statology. https://www.statology.org/performing-cluster-analysis-in-python-a-step-by-step-tutorial.

Environment settings

We import necessary libraries.

```
# import libraries
import numpy as np
import polars as pl
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

We specify the url link where the dataset is hosted.

```
url = 'https://tinyurl.com/5n8k4bku'
```

We save the dataset in a polars dataframe.

```
df = pl.read_csv(url)
```

Load data

In the following table, we can see an extract of the dataset used for this project.

```
# Show dataframe
df.head()
```

Table 1: Clustering dataset

CustomerID i64	Gender str	Age i64	Annual Income (k$) i64	Spending Score (1-100) i64
1	”Male”	19	15	39
2	”Male”	21	15	81
3	”Female”	20	16	6
4	”Female”	23	16	77
5	”Female”	31	17	40

Data preparation

The dataset needs no furhter cleaning, so we just have to do the next steps:

- Feature selection: select relevant numerical attributes for clustering.
- Normalization: scale the values of attributes. This will be helpful for the effectiveness of the clustering algorithm.

```
# Select relevant features for clustering
X= df.select('Annual Income (k$)', 'Spending Score (1-100)').to_numpy()

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Model selection

Many clustering algorithms require setting configuration parameters, often termed *hyperparameters* in machine learning.

Specifically, the *k-means algorithm*, requires that we predetermine the number of *clusters* (K) to be identified.

In some cases, domain expertise or existing knowledge about the problem and data can provide an approximate value for K. However, when such guidance is lacking, the **Elbow Method** offers a more systematic alternative to trial and error.

This method involves repeatedly applying the k-means algorithm with a range of increasing K values and then plotting the inertia for each result. Inertia serves as a metric for cluster quality, with lower values signifying more well-defined clusters. The goal is to find a clustering solution with both low inertia and a reasonable number of clusters.

Consequently, when the inertia values for various K are visualized as a curve, the point on the curve closest to the origin (0,0), representing the trade-off between low inertia and low K, often suggests a good choice for the number of clusters.

Determing the K using the Elbow Method

```
# Determine optimal number of clusters (K)
inertia = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=626)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow method to decide on the best 'K'
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Determining the Number of Clusters (K)')
plt.xlabel('Clusters')
plt.ylabel('Inertia')
plt.show()
```

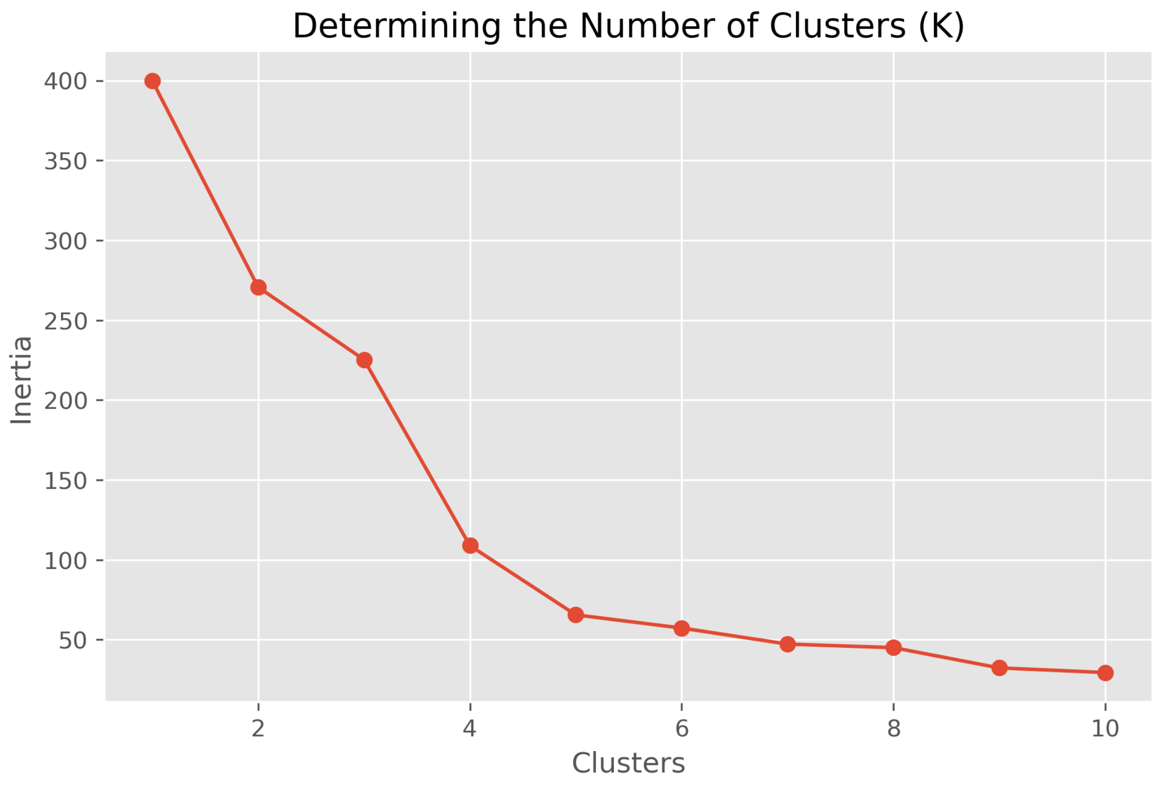

Figure 1: Determining the Optimal K with Elbow Method

Now that we have determined the optimal k, we can proceed to create the kmeans model using $k = 5$.

```
# Apply K-means with K=5
kmeans = KMeans(n_clusters=5, random_state=626)
kmeans.fit(X_scaled)
```

```
KMeans(n_clusters=5, random_state=626)
```

```
# Add the cluster identifiers as a new field in the dataset
df = df.with_columns(
    pl.Series(name='Cluster', values=kmeans.labels_)
)
```

Model Visualization

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['Annual Income (k$)'],
                y=df['Spending Score (1-100)'],
                hue=df['Cluster'],
                palette='viridis',
                s=100)
plt.scatter(X[:, 0], X[:, 1], c=df['Cluster'], cmap='viridis')
plt.title('Customer Segments based on income and spending Score')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.show()
```

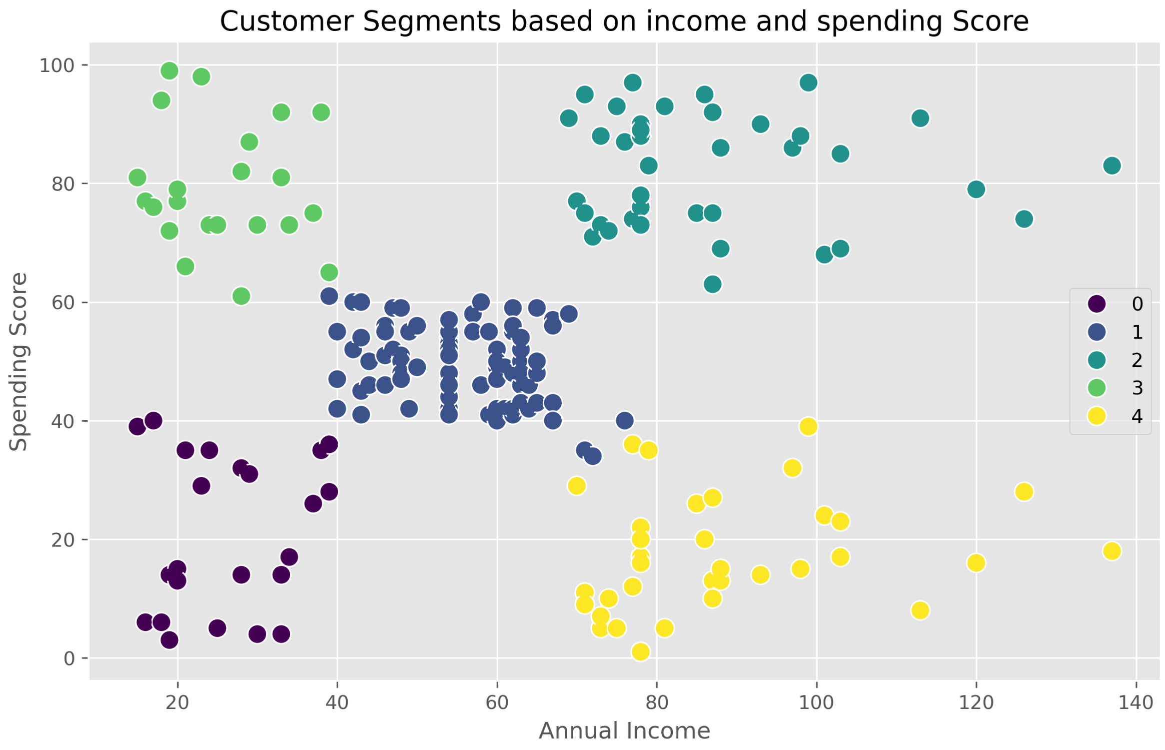

Figure 2: Visualizing the clusters using annual income and spending score

Conclusions

This document demonstrated a basic application of K-Means clustering in Python.

We hope this provides a clear understanding of the fundamental steps involved in using this powerful unsupervised learning technique.

Experimentation is key in clustering as we can see when exploring with different values of k, and visualize results to gain deeper insights from data.

Clustering is a valuable tool for data exploration and discovery. By uncovering hidden structures in data, clustering can lead to valuable insights and inform decision-making in various domains.

Reference

- Carrascosa, Ivan Palomares. 2025. "The Beginner's Guide to Clustering with Python." Practical Machine Learning.

Contact

Jesus L. Monroy
Economist & Data Scientist

Portfolio | Medium | Linkedin | Twitter