Cluster Analysis with Python

A Basic Example using Polars

Jesus LM

Apr, 2025

Table of Contents

Cluster Analysis and Clustering **5**
- Environment settings . . . 5
- Load data . . . 6
- Data preparation . . . 6
- Model selection . . . 6
  - Determing the K using the Elbow Method . . . 7
- Model Visualization . . . 12

Conclusions **13**

Reference **14**

Contact **15**

List of Figures

1 Determining the Optimal K with Elbow Method 7
2 Visualizing the clusters using annual income and spending score 12

List of Tables

1 Clustering dataset 6

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()
```

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

Table 1: Clustering dataset

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()
```

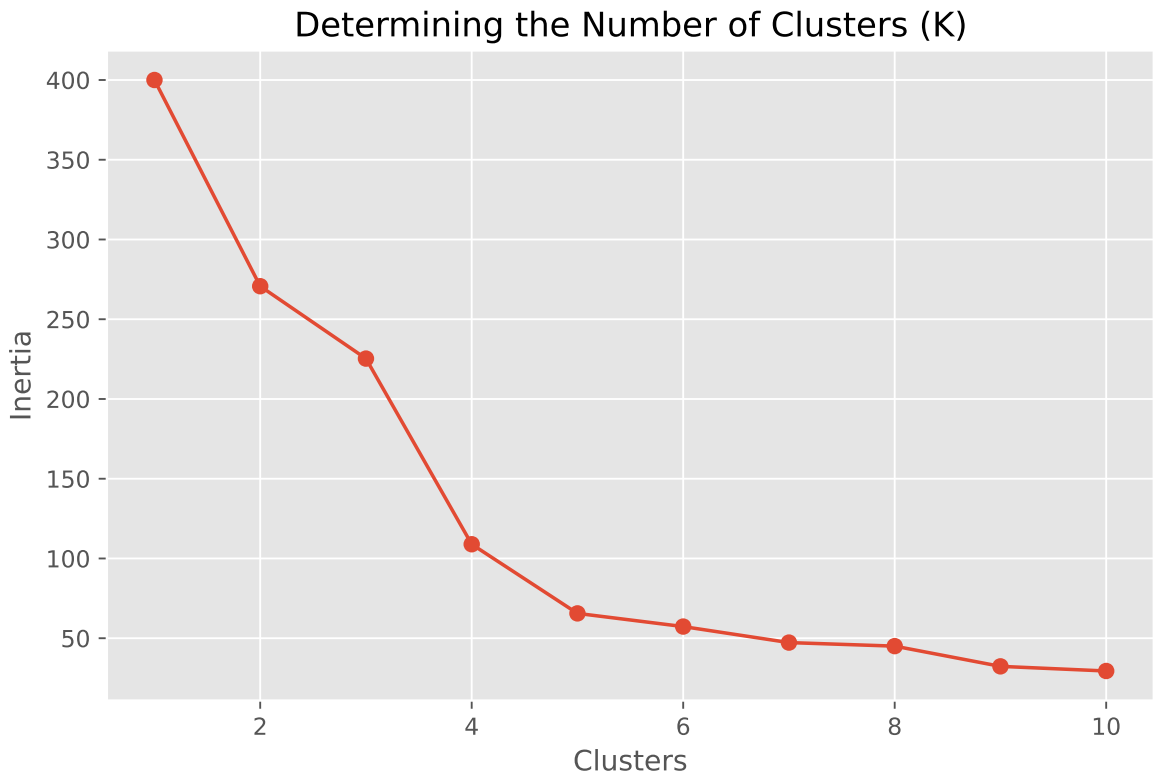

Figure 1: Determining the Optimal K with Elbow Method

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)
```

n_clusters n_clusters: int, default=8 The number of clusters to form as well as the number of centroids to generate. For an example of how to choose an optimal value for ‘n_clusters‘ refer to :ref:‘sphx_glr_auto_examples_cluster_plot_kmeans_silhouette_analysis.py‘.	5

	init init: {'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++' Method for initialization: * 'k-means++' : selects initial cluster centroids using sampling based on an empirical probability distribution of the points' contribution to the overall inertia. This technique speeds up convergence. The algorithm implemented is "greedy k-means++". It differs from the vanilla k-means++ by making several trials at each sampling step and choosing the best centroid among them. * 'random': choose ‘n_clusters‘ observations (rows) at random from data for the initial centroids. * If an array is passed, it should be of shape (n_clusters, n_features) and gives the initial centers. * If a callable is passed, it should take arguments X, n_clusters and a random state and return an initialization. For an example of how to use the different ‘init‘ strategies, see :ref:‘sphx_glr_auto_examples_cluster_plot_kmeans_digits.py‘. For an evaluation of the impact of initialization, see the example :ref:‘sphx_glr_auto_examples_cluster_plot_kmeans_stability_low_dim_dense.py‘.	'k-means++'

n_init n_init: 'auto' or int, default='auto' Number of times the k-means algorithm is run with different centroid seeds. The final results is the best output of ‘n_init‘ consecutive runs in terms of inertia. Several runs are recommended for sparse high-dimensional problems (see :ref:‘kmeans_sparse_high_dim‘). When ‘n_init='auto'‘, the number of runs depends on the value of init: 10 if using ‘init='random'‘ or ‘init‘ is a callable; 1 if using ‘init='k-means++'‘ or ‘init‘ is an array-like. .. versionadded:: 1.2 Added 'auto' option for ‘n_init‘. .. versionchanged:: 1.4 Default value for ‘n_init‘ changed to ‘'auto'‘.	'auto'
max_iter max_iter: int, default=300 Maximum number of iterations of the k-means algorithm for a single run.	300
tol tol: float, default=1e-4 Relative tolerance with regards to Frobenius norm of the difference in the cluster centers of two consecutive iterations to declare convergence.	0.0001
verbose verbose: int, default=0 Verbosity mode.	0
random_state random_state: int, RandomState instance or None, default=None Determines random number generation for centroid initialization. Use an int to make the randomness deterministic. See :term:‘Glossary ‘.	626

	copy_x copy_x: bool, default=True When pre-computing distances it is more numerically accurate to center the data first. If copy_x is True (default), then the original data is not modified. If False, the original data is modified, and put back before the function returns, but small numerical differences may be introduced by subtracting and then adding the data mean. Note that if the original data is not C-contiguous, a copy will be made even if copy_x is False. If the original data is sparse, but not in CSR format, a copy will be made even if copy_x is False.	True
	algorithm algorithm: {"lloyd", "elkan"}, default="lloyd" K-means algorithm to use. The classical EM-style algorithm is ‘"lloyd"‘. The ‘"elkan"‘ variation can be more efficient on some datasets with well-defined clusters, by using the triangle inequality. However it's more memory intensive due to the allocation of an extra array of shape ‘(n_samples, n_clusters)‘. .. versionchanged:: 0.18 Added Elkan algorithm .. versionchanged:: 1.1 Renamed "full" to "lloyd", and deprecated "auto" and "full". Changed "auto" to use "lloyd" instead of "elkan".	'lloyd'

```
# 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()
```

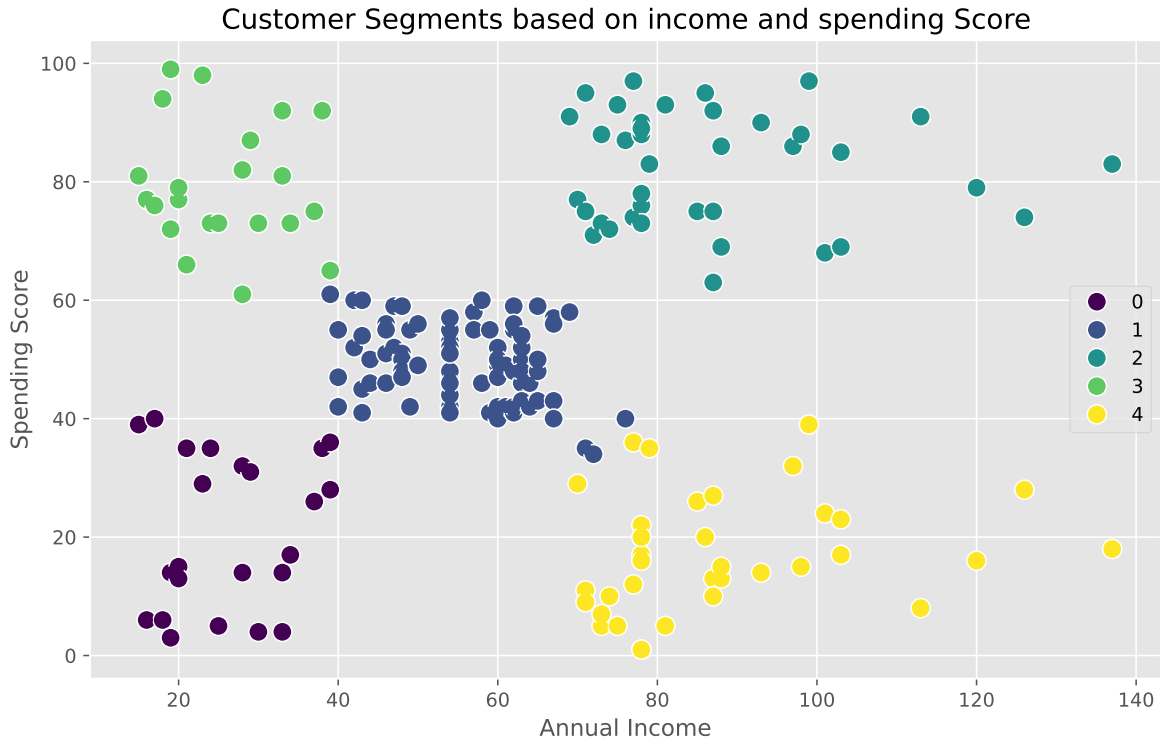

Figure 2: Visualizing the clusters using annual income and spending score

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 LM
Economist & Data Scientist

Linkedin | Medium | Twitter