# DATA SCIENCE 2 - DATA & A.I.

## 3 V MACHINE LEARNING

### 4-7 ML TECHNIQUES

Karel de Grote Hogeschool

# DATA SCIENCE 2
# DATA & A.I. 3

I

## PYTHON BASICS

Python for data science

II

## WORKING WITH ARRAYS

Numpy

III

## DATA ENGINEERING

pandas

IV

## DATA VISUALISATION

Matplotlib

V

## MACHINE LEARNING

Automatically find patterns

V

# MACHINE LEARNING

scikit-learn

# MACHINE LEARNING

01 WHAT IS MACHINE LEARNING
Automatically find patterns

02 INTRODUCING SCIKIT-LEARN
Machine learning with Python

03 HYPERPARAMETERS AND CROSS VALIDATION
Holdout samples and cross-validation

04 REGRESSION
Best fitting line

05 DECISION TREES
Best separating lines

06 K-MEANS CLUSTERING
Object grouping

07 ASSOCIATION RULES
Frequent itemsets

08 ARTIFICIAL NEURAL NETWORK
Imitate the human brain

# UNSUPERVISED LEARNING: CLUSTERING

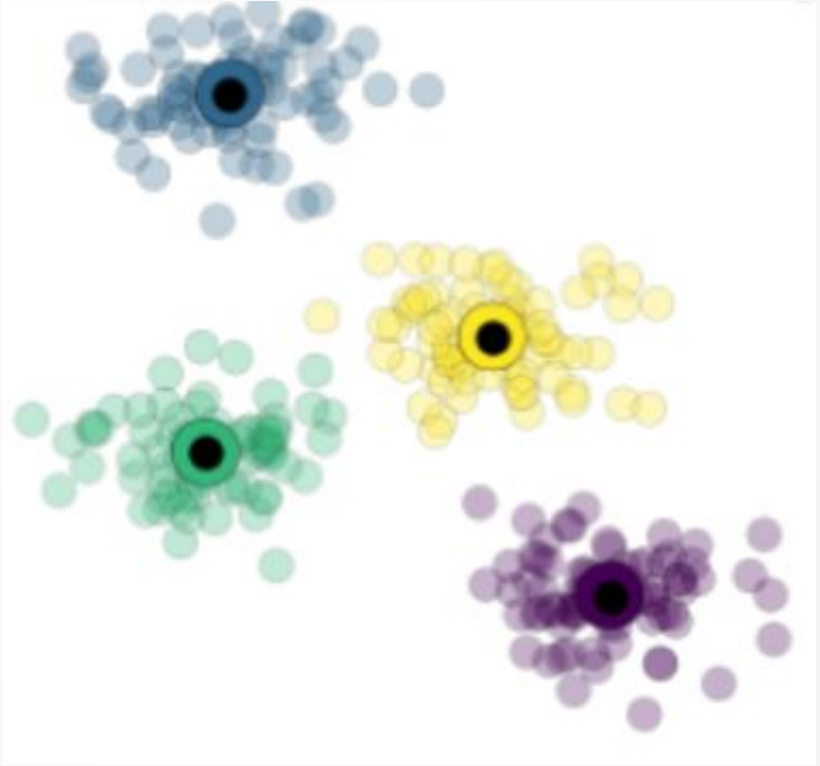
K-Means

# K-MEANS CLUSTERING

**K-Means Clustering:**

An unsupervised machine learning algorithm used to group similar data points into a predefined number of clusters based on their feature similarities.

**See Data & A.I. 2**



# K-MEANS CLUSTERING WITH SCIKIT-LEARN

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=4)
```

## Hyperparameters:

**n_clusters**

- **Type**: int, default=8
- **Description**: The number of clusters to form, and the number of centroids to generate.
- **Usage**: Set this to specify how many groups you want the data divided into.

**init**

- **Type**: {'k-means++', 'random'}, default='k-means++'
- **Description**: Method for initializing the cluster centroids. k-means++ chooses centroids to speed up convergence, while random selects random points.
- **Usage**: k-means++ is typically preferred, but random can be used for experimentation.

**max_iter**

- **Type**: int, default=300
- **Description**: The maximum number of iterations allowed for the algorithm to run until it converges.
- **Usage**: Increase this if your model is not converging, but usually the default works well.

# K-MEANS CLUSTERING WITH SCIKIT-LEARN

## Hyperparameters:

**tol**
- **Type**: float, default=1e-4
- **Description**: The tolerance for the convergence criterion. The algorithm stops when the difference in the cluster centers falls below this threshold.
- **Usage**: Decrease this value to make the algorithm more precise at the cost of computation time.

**n_init**
- **Type**: int, default=10
- **Description**: The number of times the algorithm will be run with different centroid seeds. The final result will be the best output of these runs.
- **Usage**: Increase this to get a more stable result by running the algorithm multiple times with different initializations.

**algorithm**
- **Type**: {'auto', 'full', 'elkan'}, default='llkan'
- **Description**: The algorithm to use for clustering. elkan is faster with sparse data, while full uses the standard EM-style algorithm.
- **Usage**: elkan is recommended for efficiency, especially when handling large datasets.

**random_state**
- **Type**: int, default=None
- **Description**: The seed used by the random number generator. Setting this ensures the same results each time the algorithm is run.
- **Usage**: Useful when you want reproducible results.

# K-MEANS CLUSTERING WITH SCIKIT-LEARN

```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt

# Create or load some example data
# For illustration, we generate a synthetic dataset
from sklearn.datasets import make_blobs

# Generate synthetic data with 4 clusters
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=42)

# Initialize and fit the KMeans model
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, random_state=42)
kmeans.fit(X)

# Get the cluster centers and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
```

# K-MEANS CLUSTERING WITH SCIKIT-LEARN

```
# Get the cluster centers and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(8, 6))

# Plot the points and color by cluster
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette="viridis", s=100, alpha=0.6,
edgecolor="k")

# Plot the centroids
plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red', label='Centroids')
plt.title('KMeans Clustering')
plt.legend()
plt.show()

# Predicting new points
new_points = np.array([[0, 0], [5, 5], [-5, -5]])
predictions = kmeans.predict(new_points)
print(f"Cluster labels for new points: {predictions}")
```

# K-MEANS CLUSTERING WITH SCIKIT-LEARN EXAMPLE