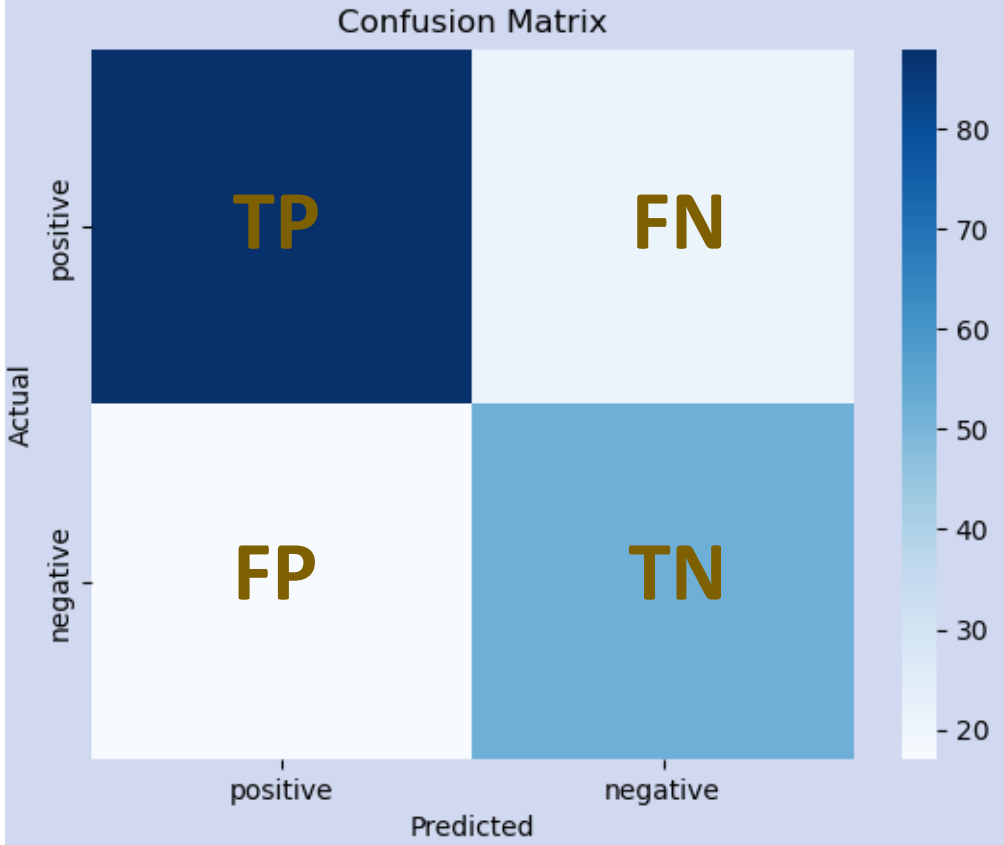
KNN

- Classification Report
- Knn for classification
- Knn for imputing

A classification_report is a summary that shows how well a classification model works. All it's values can be calculated from the confussiom matrix

```
print(classification_report(y_test_nonlinear, y_pred_nonlinear))
```

	precision	recall	f1-score	support
0	1.00	0.41	0.58	22
1	0.58	1.00	0.73	18
accuracy			0.68	40
macro avg	0.79	0.70	0.66	40
weighted avg	0.81	0.68	0.65	40



$$Precision = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

```
print(classification_report(y_test_nonlinear, y_pred_nonlinear))
```

	precision	recall	f1-score	support
0	1.00	0.41	0.58	22
1	0.58	1.00	0.73	18
accuracy			0.68	40
macro avg	0.79	0.70	0.66	40
weighted avg	0.81	0.68	0.65	40

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

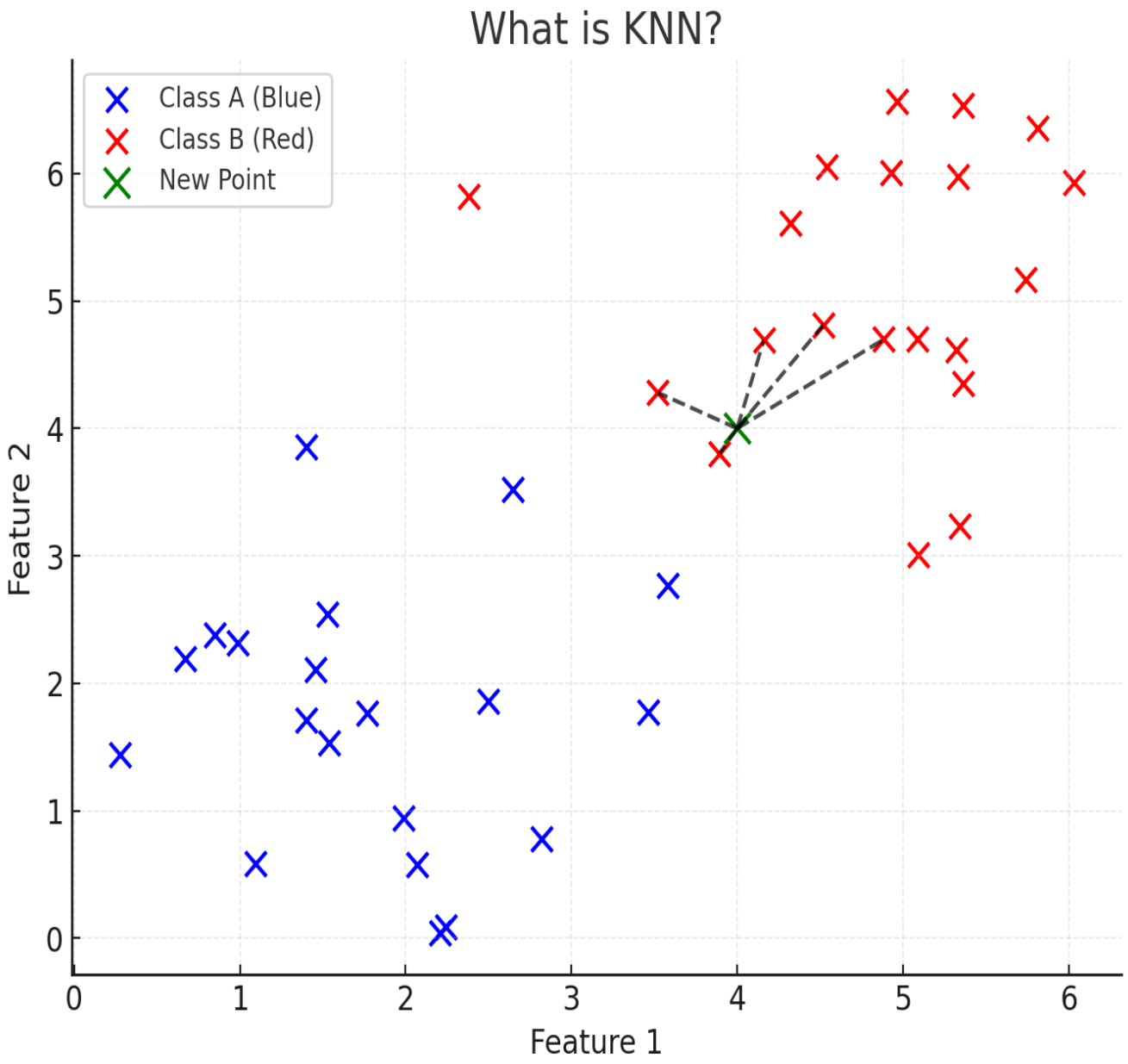
What is KNN?

•KNN, or **K-Nearest Neighbors**, is a simple, intuitive algorithm used for both **classification** and **regression**.

•It works by finding the **K nearest data points** (neighbors) to a given input and making predictions based on those neighbors.

•For classification: It assigns the class most common among the neighbors.

•For regression: It predicts the average value of the neighbors.



Why is KNN called a "Lazy Learner"?

•KNN is called a "lazy learner" because it **does not train a model in advance**.

•Instead, it stores the entire training dataset and makes predictions by calculating distances between the input and all data points during prediction.

•This makes it computationally expensive for large datasets during prediction.

KNN for Regression vs. Classification

- **Regression:** KNN predicts a continuous value by averaging the values of the K nearest neighbors.
- **Classification:** KNN assigns the most common class among the K nearest neighbors.

Why is Scaling Important for KNN?

- KNN relies on distance calculations (e.g., Euclidean distance).
- Features with larger ranges can dominate distance calculations, leading to biased predictions.
- **Scaling (e.g., StandardScaler or MinMaxScaler)** ensures all features contribute equally by standardizing their ranges.

What is an Unbalanced Dataset?

•In an unbalanced dataset, one class is significantly more represented than others.

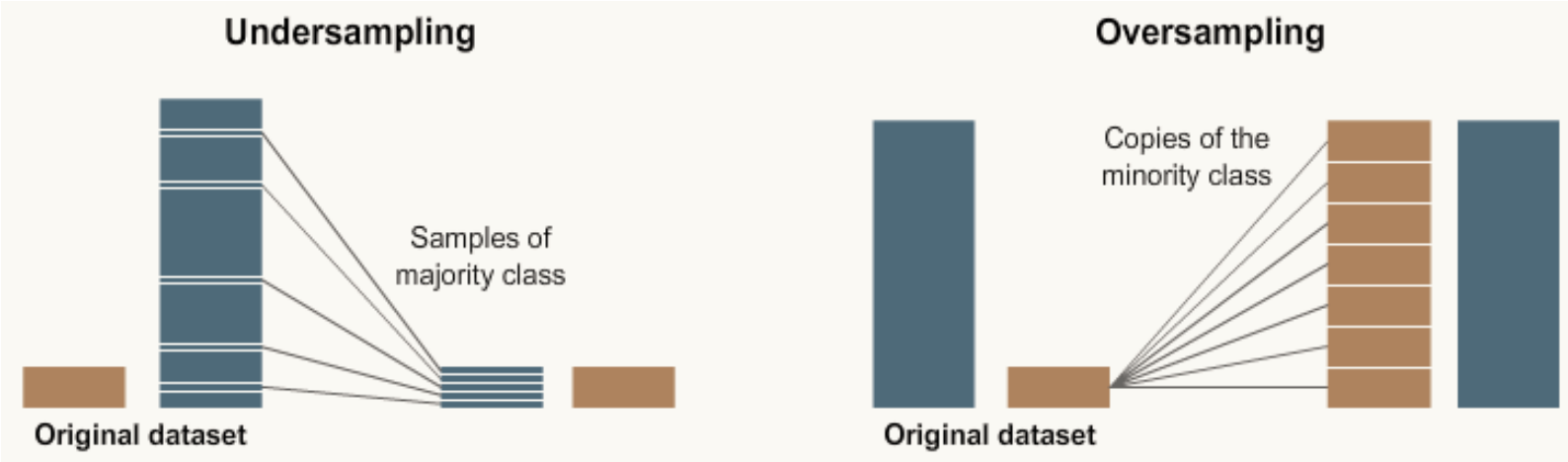
•This can cause KNN to favor the majority class, leading to poor performance on minority classes.

Solutions for Unbalanced Data:

1.Oversampling: Increase the size of the minority class (e.g., using SMOTE).

2.Undersampling: Reduce the size of the majority class.

3.Use metrics like F1-Score to evaluate performance on imbalanced data.



Key Points in KNN Syntax

Where is it imported from?

•KNN is implemented in Scikit-Learn:

```
from sklearn.neighbors import KNeighborsClassifier
```

Important Parameters in Syntax:

n_neighbors (K): The number of neighbors to consider for making predictions. A smaller K makes the model more sensitive to noise, while a larger K smoothens predictions.

weights: How neighbors influence the prediction:

uniform: All neighbors have equal weight.

distance: Closer neighbors have more influence.

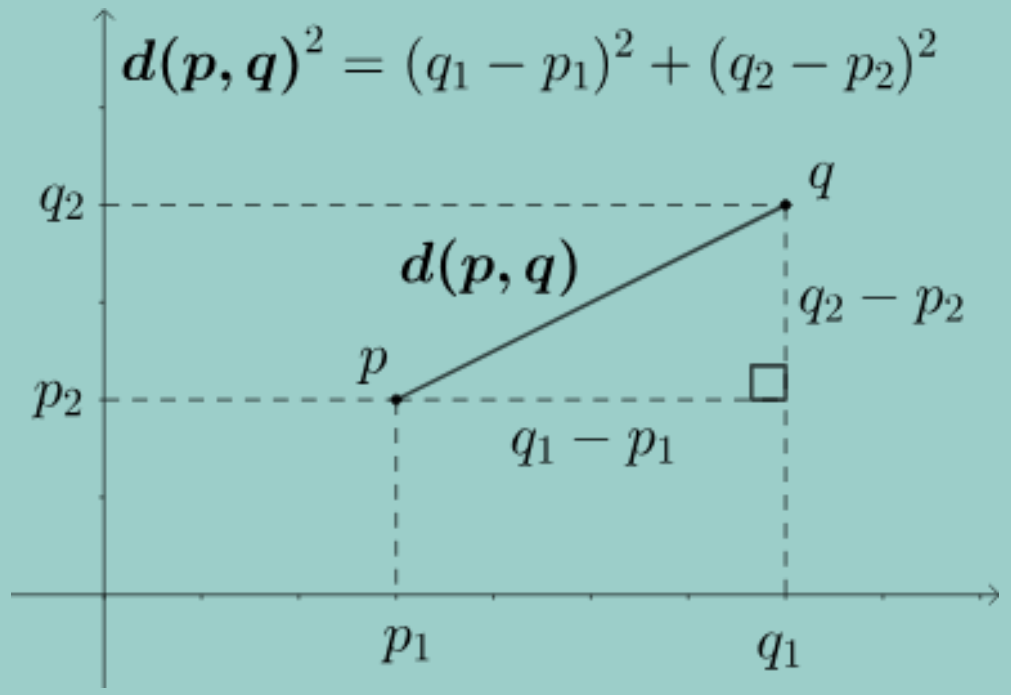
```
knn = KNeighborsClassifier(n_neighbors= 7,p=2, weights='distance' )
```

Important Parameters in Syntax:

p: The power parameter for the Minkowski distance:

```
knn = KNeighborsClassifier(n_neighbors= 7,p=2, weights='distance' )
```

p=2: Euclidean distance



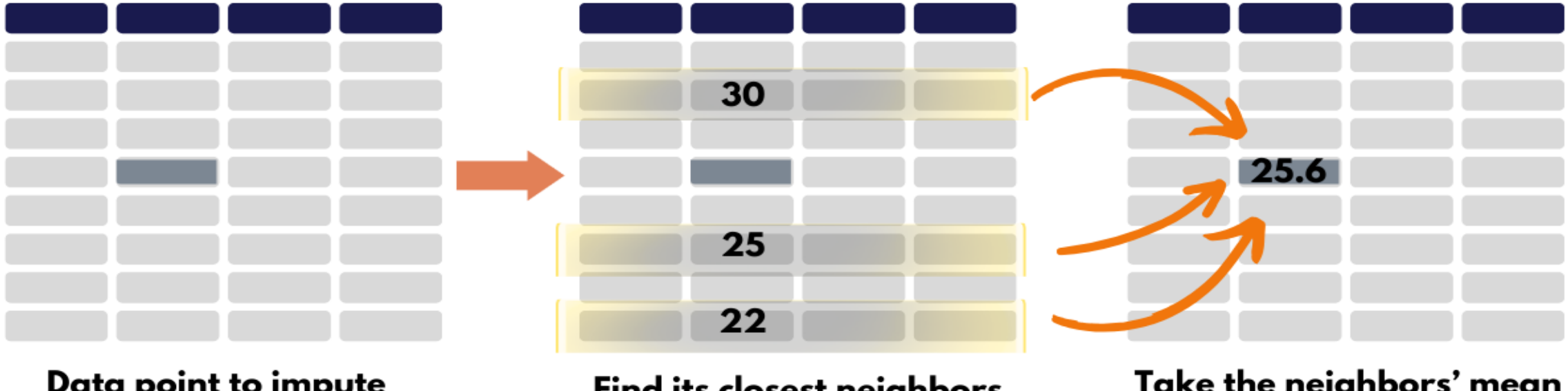
p=1: Manhattan distance.

What is KNN Imputer?

•**KNN Imputer** is used for filling in missing values in datasets by using the K-nearest neighbors algorithm.

•It works by:

- Finding the K nearest neighbors of a data point with missing values.
- Filling the missing value with the average (or weighted average) of the corresponding values from its neighbors.



How to implement

```
from sklearn.impute import KNNImputer

# Create the imputer
imputer = KNNImputer(n_neighbors=5, weights='distance')

# Fit and transform the data
imputed_data = imputer.fit_transform(data)
```

Summary

- KNN is a simple yet powerful algorithm for classification and regression.
- It's easy to implement, and intuitive but computationally expensive for large datasets.
- Scaling and handling imbalanced data are critical for improving performance.
- The KNN Imputer extends KNN's functionality to handle missing data effectively.