# DATA TRANSFORMATION

- Data transformation involves converting raw data into a format that is more suitable for analysis.
- This can include cleaning and restructuring data, handling missing values, scaling numerical features, encoding categorical variables, and more.
- The goal is to prepare the data for modeling or visualization.

# SCALING,NOPRMALIZATION AND ENCODING

- Scaling is a broader term that encompasses both normalization and standardization. While normalization aims for a specific range (0-1), scaling adjusts the spread or variability of your data.

| Criteria | Normalization | Scaling |
|---|---|---|
| **Purpose** | Adjusts values to fit within a specific range, typically between 0 and 1. | Adjusts values to have a mean of 0 and a standard deviation of 1, without necessarily constraining them to a specific range. |
| **Range of values** | Transforms data to a common scale, preserving the shape of the original distribution. | Centers the data around 0 and scales it based on the standard deviation. |
| **Effect on outliers** | Can be sensitive to outliers since it uses the minimum and maximum values. | Less sensitive to outliers since it calculates based on mean and standard deviation. |
| **Algorithm compatibility** | Often used with algorithms that rely on distance measures, like KNN or SVM. | Suitable for algorithms that assume zero-centered data, like PCA or gradient descent-based optimization. |
| **Computation** | Requires finding the minimum and maximum values for each feature, which can be computationally expensive for large datasets. | Involves calculating the mean and standard deviation of each feature, which is computationally efficient. |
| **Distribution preservation** | Preserves the shape of the original distribution, maintaining the relative relationships between data points. | May alter the distribution slightly, particularly if the data has a non-Gaussian distribution. |
| **Data type suitability** | Suitable for features with a bounded range or when the absolute values of features are meaningful. | Suitable for features with unbounded ranges or when the mean and variance of features are meaningful. |
| **When to use** | When the scale of features varies significantly, and you want to bring them to a comparable range. Particularly useful when the algorithm doesn't make assumptions about the distribution of the data. | When features have different units or scales and you want to standardize them so that each feature contributes equally to the analysis. It's also useful when algorithms assume that features are centered around zero. |

| Criteria | Normalization | Scaling |
|---|---|---|
| **Purpose** | Adjusts values to fit within a specific range, typically between 0 and 1. | Adjusts values to have a mean of 0 and a standard deviation of 1, without necessarily constraining them to a specific range. |
| **Range of values** | Transforms data to a common scale, preserving the shape of the original distribution. | Centers the data around 0 and scales it based on the standard deviation. |
| **Effect on outliers** | Can be sensitive to outliers since it uses the minimum and maximum values. | Less sensitive to outliers since it calculates based on mean and standard deviation. |
| **Algorithm compatibility** | Often used with algorithms that rely on distance measures | Suitable for algorithms that assume zero-centered data |
| **Computation** | Requires finding the minimum and maximum values for each feature, which can be computationally expensive for large datasets. | Involves calculating the mean and standard deviation of each feature, which is computationally efficient. |
| **Distribution preservation** | Preserves the shape of the original distribution, maintaining the relative relationships between data points. | May alter the distribution slightly, particularly if the data has a non-Gaussian distribution. |
| **Data type suitability** | Suitable for features with a bounded range or when the absolute values of features are meaningful. | Suitable for features with unbounded ranges or when the mean and variance of features are meaningful. |
| **When to use** | When the scale of features varies significantly, and you want to bring them to a comparable range. Particularly useful when the algorithm doesn't make assumptions about the distribution of the | When features have different units or scales and you want to standardize them so that each feature contributes equally to the analysis. It's also useful when algorithms assume that features are centered around |

# Scikit-learn

- **Scikit-learn** is one of the most popular open-source machine learning libraries for Python. It provides simple and efficient tools for data analysis and machine learning, built on top of other scientific computing libraries like **NumPy**, **SciPy**, and **matplotlib**.

We use two type scaling techniques in data frame

1. StandardScaler(mean=0,variance=1)
2. MinMaxScaler(Scaling values between 0 and 1)

# StandardScaler(mean=0,variance=1)

- The **StandardScaler** in scikit-learn is a preprocessing tool used to standardize or scale the features of a dataset.

- It transforms the features such that they have a **mean of 0** and a **variance of 1,** which is particularly useful when dealing with machine learning algorithms that are sensitive to the scale of the data.

**How StandardScaler Works:**

The **StandardScaler** standardizes the features by applying the following transformation to each feature xxx:

$$x_{scaled} = x - \mu / \sigma$$

Where:

- μ\muμ is the **mean** of the feature (calculated across all samples).
- σ\sigmaσ is the **standard deviation** (square root of variance) of the feature.

This ensures that:

- The **mean** of each feature is 0.
- The **variance** (or standard deviation) of each feature is 1.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Example DataFrame with numeric columns
data = { 'Age': [25, 30, 35, 40, 45],
    'Salary': [50000, 60000, 70000, 80000, 90000],
    'Experience': [1, 2, 3, 4, 5]}
df = pd.DataFrame(data)
# Initialize StandardScaler
scaler = StandardScaler()
# Apply StandardScaler to numeric columns
numeric_columns = ['Age', 'Salary', 'Experience']
 # Columns to scale
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
# Print the scaled DataFrame
print(df)
# Check the mean and variance of the transformed columns
print("\nMean of each column after scaling:")
print(df.mean())

print("\nVariance of each column after scaling:")
print(df.var())
```

| | Age | Salary | Experience |
|---|---|---|---|
| 0 | -1.414214 | -1.414214 | -1.414214 |
| 1 | -0.707107 | -0.707107 | -0.707107 |
| 2 | 0.000000 | 0.000000 | 0.000000 |
| 3 | 0.707107 | 0.707107 | 0.707107 |
| 4 | 1.414214 | 1.414214 | 1.414214 |

Mean of each column after scaling:
Age 0.0
Salary 0.0
Experience 0.0
dtype: float64

Variance of each column after scaling:
Age 1.0
Salary 1.0
Experience 1.0
dtype: float64

- The fit_transform() method first calculates the **mean** and **standard deviation** of each column and then scales the data accordingly.

- You can apply scaling to only specific columns of a DataFrame (as shown in the numeric_columns list).

# MinMaxScaler(Scaling values between 0 abd 1)

The MinMaxScaler is another preprocessing tool provided by scikit-learn to scale features to a specific range, typically between 0 and 1. Unlike the StandardScaler, which standardizes the data to have a mean of 0 and a standard deviation of 1, the MinMaxScaler scales the features by transforming them based on the minimum and maximum values of each feature.

# How MinMaxScaler Works:

The MinMaxScaler scales each feature according to the following

formula:

$$x\text{scaled} = x - \min(x) / \max(x) - \min(x)$$

Where: min($x$) is the minimum value of the feature

max($x$) is the maximum value of the feature.
x is the individual data point.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Example DataFrame with numeric columns
data = {
    'Age': [25, 30, 35, 40, 45],
    'Salary': [50000, 60000, 70000, 80000, 90000],
    'Experience': [1, 2, 3, 4, 5]
}
df = pd.DataFrame(data)
# Initialize MinMaxScaler
scaler = MinMaxScaler()
# Apply MinMaxScaler to numeric columns
numeric_columns = ['Age', 'Salary', 'Experience']
# Columns to scale
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
# Print the scaled DataFrame
print(df)
# Check the min and max of the transformed columns
print("\nMinimum value of each column after scaling:")
print(df.min())
print("\nMaximum value of each column after scaling:")
print(df.max())
```

| | Age | Salary | Experience |
|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 |
| 1 | 0.25 | 0.25 | 0.25 |
| 2 | 0.50 | 0.50 | 0.50 |
| 3 | 0.75 | 0.75 | 0.75 |
| 4 | 1.00 | 1.00 | 1.00 |

Minimum value of each column after scaling:
Age 0.0
Salary 0.0
Experience 0.0
dtype: float64

Maximum value of each column after scaling:
Age 1.0
Salary 1.0
Experience 1.0
dtype: float64