Data Processing

Data preprocessing is the process of evaluating, filtering, manipulating, and encoding data so that a machine learning algorithm can understand it and use the resulting output. The major goal of data preprocessing is to eliminate data issues such as missing values, improve data quality, and make the data useful for machine learning purposes.

Why is Data Preprocessing Important?

Improves Data Quality → Fixes missing, duplicate, or incorrect data.

Prepares Data for Algorithms → Converts data into formats ML models can understand (e.g., encoding, scaling).

Boosts Model Performance → Enhances accuracy, speed, and generalization.

Enables Feature Engineering → Creates new, meaningful features to improve learning.

Supports Fairness & Privacy → Helps remove bias, rebalance data, and anonymize sensitive info.

Ensures Consistency Across Data → Standardizes formats and ensures uniformity.

Reduces Overfitting/Underfitting → By cleaning noise and focusing on relevant feature

Import necessary libraries

In [15]:
```
import pandas as pd
import numpy as np
```

LOAD DATASET

In [16]:
```
# Load data
data = pd.DataFrame({
'age': [25, np.nan, 30, 45, np.nan],
'salary': [50000, 60000, np.nan, 65000, 70000],
'city': ['New York', 'Los Angeles', 'New York', 'San Francisco', np.nan],
'target': [1, 0, 1, 0, 1]
})
```

In [17]:
```
# Display the original data
print("Original Data:")
print(data)
```

```
Original Data:
    age   salary           city  target
0  25.0  50000.0       New York       1
1   NaN  60000.0    Los Angeles       0
2  30.0      NaN       New York       1
3  45.0  65000.0  San Francisco       0
4   NaN  70000.0            NaN       1
```

Identifying missing values

In [18]:
```
# Display missing values
print("Missing values:")
print(data.isnull().sum())
```

```
Missing values:
age       2
salary    1
city      1
target    0
dtype: int64
```

In [19]:
```
# Percentage of missing values
print("percentage of Missing values:")
print(data.isnull().mean() * 100)
```

```
percentage of Missing values:
age       40.0
salary    20.0
city      20.0
target     0.0
dtype: float64
```

Handling missing values

1. Deleting the rows of missing values

In [20]:
```
# removing all  rows with null values
data1 = data # creating a copy of the data
data1 = data1.dropna()
print('after removing rows')
print(data1)
```

```
after removing rows
    age   salary           city  target
0  25.0  50000.0       New York       1
3  45.0  65000.0  San Francisco       0
```

In [21]:
```
# removing all columns with null values
data1 = data1.dropna(axis=1)
print('after removing columns')
print(data1)
```

```
after removing columns
    age   salary           city  target
0  25.0  50000.0       New York       1
3  45.0  65000.0  San Francisco       0
```

2. Replacing the missing values of numerical attributes with mean

In [ ]:
```
from sklearn.impute import SimpleImputer
# For numerical features, use mean imputation
num_features = ['age', 'salary'] # Creating a list of numerical features
imputer_num = SimpleImputer(strategy='mean') # creating an instance of simple imput
data[num_features] = imputer_num.fit_transform(data[num_features]) # fitting to the
data
```

Out[ ]:

	age	salary	city	target
0	25.000000	50000.0	New York	1
1	33.333333	60000.0	Los Angeles	0
2	30.000000	61250.0	New York	1
3	45.000000	65000.0	San Francisco	0
4	33.333333	70000.0	NaN	1

3. Replacing the missing values of categorical attributes with the most frequent values

In [ ]:
```
# For categorical features, use the most frequent value imputation
cat_features = ['city'] #creating a list of categorical attributes
imputer_cat = SimpleImputer(strategy='most_frequent') # creating a instance of simp
data[cat_features] = imputer_cat.fit_transform(data[cat_features]) # fitting and tr
data
```

Out[ ]:

	age	salary	city	target
0	25.000000	50000.0	New York	1
1	33.333333	60000.0	Los Angeles	0
2	30.000000	61250.0	New York	1
3	45.000000	65000.0	San Francisco	0
4	33.333333	70000.0	New York	1

4. Label encoding of categorical values

Label encoding assigns a unique integer to each distinct category in a feature.

In [24]:
```
from sklearn.preprocessing import StandardScaler, LabelEncoder
# Encoding categorical features
cat_features = ['city']
label_encoders = {}
for col in cat_features:
    le = LabelEncoder() # creating an instance of labelencoder
    data[col] = le.fit_transform(data[col])
```

```
    label_encoders[col] = le
```

In [25]:
```
data
```

Out[25]:

	age	salary	city	target
0	25.000000	50000.0	1	1
1	33.333333	60000.0	0	0
2	30.000000	61250.0	1	1
3	45.000000	65000.0	2	0
4	33.333333	70000.0	1	1

5. Scaling the numerical attribute values using standard scalar

The StandardScaler is a preprocessing tool provided by Scikit-Learn's sklearn.preprocessing module. It standardizes features by removing the mean and scaling to unit variance, effectively transforming the data to have a mean of 0 and a standard deviation of 1. This process is also known as Z-score normalization.

How StandardScaler Works

For each feature in your dataset, StandardScaler computes the mean and standard deviation on the training set. It then transforms the data using the formula:

$$x' = \frac{x - \mu}{\sigma}$$

x is the original feature value,

μ is the mean of the feature values in the training set,

σ is the standard deviation of the feature values in the training set

In [26]:
```
#Feature scaling
scaler = StandardScaler() #creating a instance of standard scalar
num_features = ['age','salary']
data[num_features] = scaler.fit_transform(data[num_features]) # Fitting the data
data
```

Out[26]:

	age	salary	city	target
0	-1.265924	-1.700840	1	1
1	0.000000	-0.188982	0	0
2	-0.506370	0.000000	1	1
3	1.772294	0.566947	2	0
4	0.000000	1.322876	1	1