Data Preprocessing - Transforming raw data into a format that can be effectively and efficiently used by machine learning models. Ensures that the data is clean, consistent and suitable for the specific model and algorithm you are working with.

## Steps to Preprocess the Data

### 1 - Importing the Libraries

This step involves loading necessary Python libraries that provide functions to handle data preprocessing tasks.

**Common Libraries:**
- Pandas: For data manipulation and analysis.
- NumPy: For numerical computations and handling arrays.
- Scikit-learn: For machine learning tasks, including preprocessing, model training and evaluation.
- Matplotlib: For data visualization.

### 2 - Importing the Dataset

The dataset is loaded into a data structure (like a Pandas DataFrame) from a file format (CSV, Excel, etc.).

The dataset may contain features (independent variables) and a target (dependent variable).

### 3 - Taking Care of Missing Data (if there is)

To handle any missing values in the dataset which can negatively impact model performance.

**Common Techniques:**
- Imputation: Filing missing values with the mean, median…
- Dropping: Removing rows or columns with missing values if the impact on the dataset is minimal

### 4 - Encoding Independent Variable (if needed)

Categorical variables need to be converted into numerical formats for machine learning models to process them.

**Techniques:**
- Label Encoding: Assigns a unique integer to each category.
- One-Hot Encoding: Creates binary columns for each category to avoid ordinal relationships in non-ordinal data.

### 5 - Encoding Dependent Variable (if needed)

If the dependent variable is categorical, it also needs to be encoded.

**Technique:**
- Label Encoding: Commonly used for binary or multi-class target variables.

### 6 - Splitting the Dataset Into the Training Set and Test Set (optional)

The dataset is divided into two parts: the **training set** (used to train the model) and the **test set** (used to evaluate the model).

**Why**: This allows you to evaluate how well your model generalizes to new, unseen data.

**Common Split Ratio**: 70-80% for training and 20-30% for testing.

### 7 - Feature Scaling (if needed)

Feature scaling standardizes the range of the independent variables so that they are on the same scale, especially if one feature has a much larger range than others.

**Common Techniques:**
- Standardization: Centers the data to have a mean of 0 and a standart deviation of 1. Range is between -3 and +3
- Normalization: Scales the data to a range between 0 and 1.

Feature scaling should be applied when your dataset has features with different ranges of values and when using machine learning algorithms that are sensitive to the scale of the data.

## Python Implementation

```
#importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#importing the dataset
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

#taking care of missing data (if there is)
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])

#encoding the independent variable (if needed)
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
X = np.array(ct.fit_transform(X))

#encoding the dependent variable (if needed)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)

#splitting the dataset into the training set and test set (optional)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)

#feature scaling (if there is)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```