1. What is Data Preprocessing?

Data Preprocessing means **cleaning and preparing raw data** so that it can be used by a Machine Learning model.

Raw data collected from real life is **not perfect**. It usually has problems like:

- **Incomplete data** (missing values)
- **Noisy data** (wrong or random values)
- **Inconsistent data** (different formats, duplicates)

If we directly give such data to a Machine Learning model:

- The model may give **wrong predictions**
- Accuracy becomes **very low**

That is why data preprocessing is a **mandatory step** before training any ML model.

---

2. Why Data Preprocessing is Necessary?

Machine Learning models have limitations:

- They **cannot handle missing values** (NaN) properly
- They **cannot understand text data** like “Male”, “Female”
- They get confused when features have **different scales**

For example:

- Age ranges from 20 to 60
- Salary ranges from 20,000 to 100,000

The model gives more importance to salary because numbers are larger.

So data preprocessing is required to:

- Improve model accuracy
- Avoid wrong predictions
- Convert data into a usable numeric form

---

3. Common Data Preprocessing Steps

In most Machine Learning projects, preprocessing follows this order:

1. Handle Missing Values

2. Encode Categorical (text) Data
3. Feature Scaling
4. Train–Test Split

This pipeline prepares data for training.

---

4. Handling Missing Values

Missing values mean **empty cells or NaN values** in the dataset.

Example:

Age	Salary
25	50000
NaN	60000

Here, the age value is missing.

Method 1: Remove Missing Data

```
df.dropna()
```

This removes rows that contain missing values.

This method is used when:

- Missing values are very few
- Removing them does not affect the dataset much

Method 2: Replace with Mean (Most Common Method)

```
df['Age'].fillna(df['Age'].mean(), inplace=True)
```

Here:

- The missing value is replaced with the **average (mean)** of the column
- This keeps the dataset size unchanged

This method is used when:

- The data is numerical
- Missing values are not too many

---

5. Encoding Categorical Data

Machine Learning models **only understand numbers**, not text.

So text data must be converted into numbers.

Example:

Gender
Male
Female

Label Encoding

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
```

After encoding:

- Male → 1
- Female → 0

Now the model can understand the gender feature.

---

6. Feature Scaling

Feature scaling means **bringing all features to a similar range**.

Why is it needed?

Some algorithms like:

- KNN
- SVM

depend on **distance calculations**.

Example problem without scaling:

- Age: 20–60
- Salary: 20,000–100,000

Salary values dominate age values, causing biased predictions.

Standardization (Most Common Method)

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

This converts data so that:

- Mean = 0
- Standard deviation = 1

Now all features are on the **same scale**.

---

7. Train–Test Split

The dataset is divided into:

- **Training data** → used to train the model
- **Testing data** → used to evaluate the model

```
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=42
)
```

Here:

- 80% data is used for training
- 20% data is used for testing

This helps check how well the model performs on **new data**.

---

8. Model Evaluation Metrics

Accuracy alone does not always give a complete picture.

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

A confusion matrix shows:

- Correct predictions
- Incorrect predictions
- True positives, false positives, etc.

Precision, Recall, F1-score

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

These metrics help understand:

- How precise the model is
- How many correct predictions it makes
- Overall balance of performance

---

9. Complete End-to-End ML Code Explanation

Step 1: Import Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

These libraries are used for:

- Data handling
- Preprocessing
- Model training
- Evaluation

Step 2: Create Dataset

```
data = {
    'Age': [25, 30, 35, None, 40],
    'Gender': ['Male', 'Female', 'Female', 'Male', 'Female'],
    'Salary': [50000, 60000, 65000, 70000, 80000],
    'Purchased': [0, 1, 1, 0, 1]
}
```

- Age, Gender, Salary → input features
- Purchased → output (label)

Step 3: Handle Missing Values

```
df['Age'].fillna(df['Age'].mean(), inplace=True)
```
Missing age is replaced with average age.

Step 4: Encode Gender Column

```
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
```

Gender is converted into numbers.

Step 5: Separate Features and Label

```
X = df[['Age', 'Gender', 'Salary']]
y = df['Purchased']
```
X → input features
y → output label

Step 6: Feature Scaling

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```
All input features are scaled.

Step 7: Train–Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2
```

)

Data is split for training and testing.

Step 8: Train the Model

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

The model learns patterns from training data.

Step 9: Prediction

```
y_pred = model.predict(X_test)
```

The model predicts results for test data.

Step 10: Accuracy

```
print("Accuracy:", accuracy_score(y_test, y_pred))
```

This prints how accurate the model is.