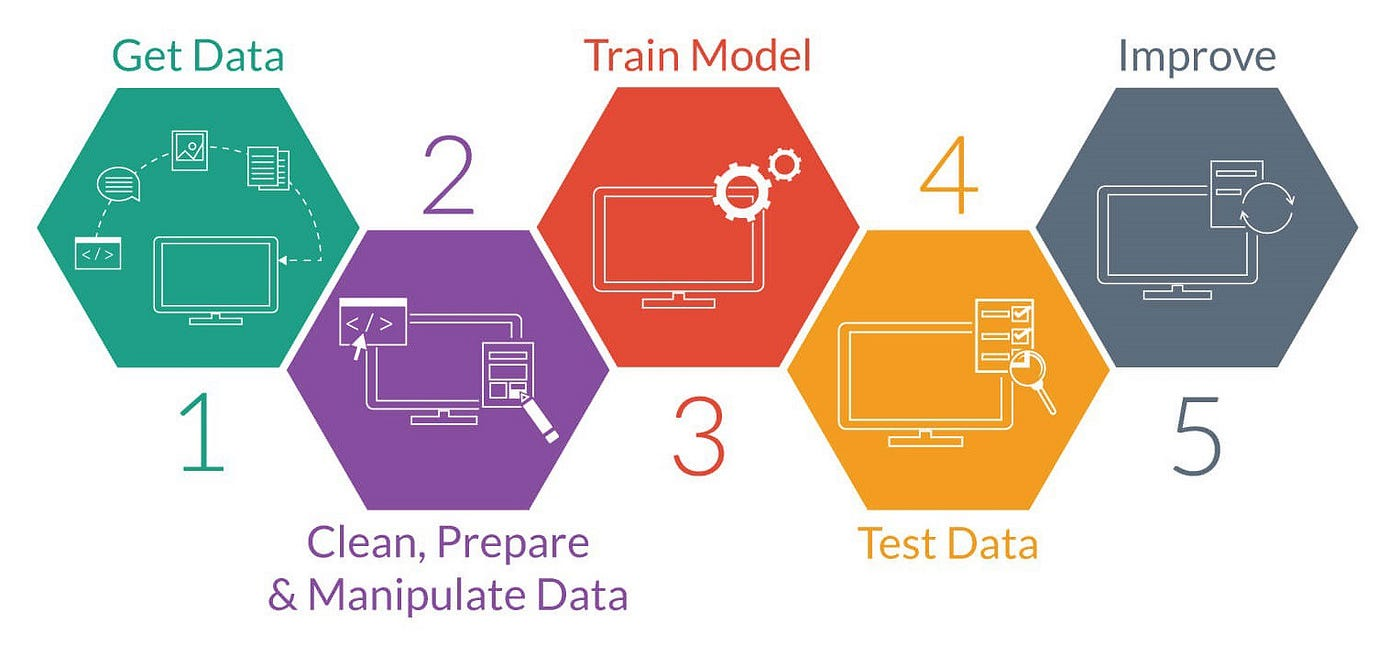
**Machine Learning Project Lifecycle**

****

1. **📚 Problem Definition**

* Understand the business problem
* Define the objective: classification, regression, clustering, etc.
* Determine success metrics

### **📥 Data Collection**

* Gather data from APIs, databases, web scraping, CSVs, etc.
* Ensure enough quantity and quality

# 03. **Know your Data**

# Understand the structure ,size and key characteristics of the data.

* **df.head(), df.tail()** – Preview data from top/bottom
* **df.shape** – Number of rows and columns
* **df.info()** – Data types, non-null counts, memory usage
* **df.describe()** – Summary statistics (numerical columns)
* **df.dtypes** – Check data types directly
* **df.columns** – View all column names
* **df.isnull().sum()** – Check for missing values
* **df.duplicated().sum()** – Check for duplicate rows
* **df.nunique()** – Number of unique values per column
* **df['column'].unique()** – Unique values in a specific column
* **df['column'].value\_counts()** – Frequency of values in a column
* **df.corr()** – Correlation between numeric features

# **04. Data Cleaning**

* **Handling Missing Values (Nulls):**
  + Drop missing rows: df.dropna()
  + Fill with mean/median/mode: **df.fillna(df['col'].mean())**, etc.
  + Use forward/backward fill: **df.ffill(), df.bfill()**
* **Handling Duplicates:**  
  + Remove exact duplicates: df.drop\_duplicates()
* **Fixing Data Types:**
  + Convert data types:

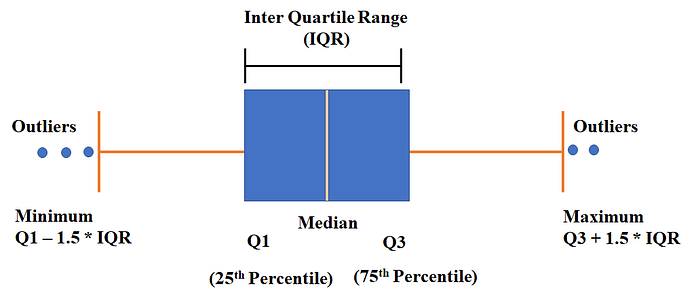
Eg:

**df['date'] = pd.to\_datetime(df['date'])**

* **Correcting Inconsistent Formatting:**  
  + **e.g**., 'Male', 'male', 'MALE' → make consistent with **.str.lower()** or **.str.title()**
* **Handling Invalid or Irrelevant Values:**  
  + **e.g**., negative ages, zero income in income column, etc.
* **Renaming Columns for Clarity (Optional):**
  + df.rename(columns={'col1': 'customer\_age'})

**05. Data Preprocessing**

**i. Identify Outliers**

****

### **Steps to Create Boxplots for Numerical Columns**

1. Select only numerical columns (to avoid errors with categorical data).  
   **df.select\_dtypes(exclude = ‘object’).columns**
2. Loop through numerical columns and create boxplots using seaborn.boxplot().
3. Visualize the outliers and understand data distribution.

### **Handling Outliers Using IQR Capping**

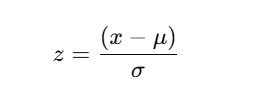
### The Interquartile Range (IQR) method is used to detect and handle outliers by capping extreme values at a threshold instead of removing them.

### **🔹 Steps to Handle Outliers Using IQR Capping**

1. Calculate Q1 (25th percentile) and Q3 (75th percentile)
2. Compute IQR = Q3 - Q1
3. Define lower and upper limits:  
   * Lower Bound = Q1 - 1.5 \* IQR
   * Upper Bound = Q3 + 1.5 \* IQR
4. Cap values:  
   * If a value is below the lower bound, replace it with lower bound
   * If a value is above the upper bound, replace it with upper bound

**🔍 Z-Score Method**

* Works best when the data is approximately normally distributed.
* The Z-score standardizes the data:



Where:

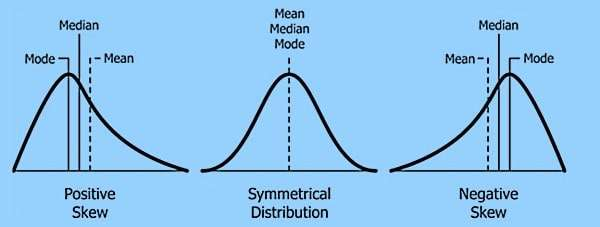
* x: data point
* μ: mean
* σ: standard deviation

### **✨ Steps to Handle Outliers Using Z-Score Method**

* Select the numerical features Identify the numeric column(s) where you want to detect outliers.
* Calculate the Z-score For each value, compute how many standard deviations it is away from the mean.
* Set a thresholdCommon threshold values are ±3. Values beyond this are **considered outliers.**
* Remove or treat the outliers
  + Remove: Drop the rows with outliers.
  + Treat: Cap them, transform them, or replace with mean/median based on context.

**ii. Skewness**

Skewness measures the asymmetry of a probability distribution of a dataset. It tells us whether the data is symmetrically distributed or leaning to one side.



### **Types of Skewness**

1. **📌 Zero Skewness (Symmetric Distribution)**
   * If skewness = 0, the distribution is **perfectly symmetrical** (like a normal distribution).
   * Example: Heights of people in a population.
2. **📈 Positive Skewness (Right-Skewed)**
   * If skewness > 0, the tail is **longer on the right side**.
   * The majority of values are **concentrated on the left**.
   * Example: **Income distribution** (most people earn less, a few earn a lot).
3. **📉 Negative Skewness (Left-Skewed)**
   * If skewness < 0, the tail is **longer on the left side**.
   * The majority of values are **concentrated on the right**.
   * Example: **Exam scores** (most students score high, few score very low).

### **✅ Steps to Find Skewness Using .skew() in a Pandas DataFrame**

1. **Inspect your data** to identify the numerical columns you want to analyze.
2. **Use the .skew() method** on the DataFrame to calculate skewness for each numerical column.
3. **Interpret the results**:  
   * A value near 0 → Symmetric distribution.
   * A positive value → Right-skewed distribution (tail on the right).
   * A negative value → Left-skewed distribution (tail on the left).
4. **Check for high skewness** (commonly above +1 or below -1) which may require transformation for certain analyses or models.

**Handling skewness with Transformations**

| Skewness Type | Description | Transformation | When to Use |
| --- | --- | --- | --- |
| **Moderate Skew** | <1 | optional | Slight asymmetry |
| **Positive Skew (>1)** | Right-tailed distribution | - Log transformation  - Square root  - Box-Cox | When values are all positive |
| **Negative Skew (<-1)** | Left-tailed distribution | - Cube  - Square  - Box-Cox | When values are all positive or transformed to positive |
| **Highly Skewed** | Extreme asymmetry or outliers present | - Winsorization(IQR)  - Clipping  - Binning  -Box-Cox/Yeo-Johnson | For improving normality in ML models |
| **Zero or Near-Zero** | Symmetric/Normal distribution | No transformation needed | Ideal for most statistical techniques |

**iii.📏Scaling**

Scaling is the process of resizing your feature values so they fall within a specific range or standard. It ensures that all features contribute equally to a model, especially when they are on different scales.

**✅** Do after transformations (if any).

## **🤔 Why is Scaling Important?**

Many machine learning models are **sensitive to the scale** of input features. Without scaling:

* Features with **large values dominate** over smaller ones.
* Algorithms that use **distance** (like KNN, SVM, K-Means) will perform poorly.
* Models may take **longer to converge** (like gradient descent in linear regression, neural nets).

**Standard Scaling:**

Standardizes features by removing the mean and scaling to unit variance (mean = 0, standard deviation = 1)

#### 🔍 **When to use:**

* Data is roughly normally distributed (mean = 0, std = 1)

**from sklearn.preprocessing import StandardScaler**

**scaler = StandardScaler()**

**scaled\_data = scaler.fit\_transform(data)**

**MinMax scaling:**

Scales features to a specified range, typically [0, 1]

🔍**When to use:**

When Data doesn't follow normal distribution; bounded scaling (0–1) needed

**from sklearn.preprocessing import MinMaxScaler**

**scaler = MinMaxScaler()**

**scaled\_data = scaler.fit\_transform(data)**

**iv. Encoding**

Encoding categorical variables is a crucial step in data preprocessing for machine learning models, as many algorithms require numerical input. The choice of encoding technique depends on the nature of the categorical data and the specific requirements of the model.

### **1. One-Hot Encoding**

**When to Use:**

* For **nominal categorical variables** (categories with no inherent order).​
* When the number of unique categories is relatively small.​

**Description:** Creates a new binary column for each category, indicating the presence (1) or absence (0) of the category.​

**Example:**

import pandas as pd

df = pd.DataFrame({'Color': ['Red', 'Blue', 'Green']})

encoded\_df = pd.get\_dummies(df, columns=['Color'])

print(encoded\_df)

**Output:**

Color\_Blue Color\_Green Color\_Red

0 0 0 1

1 1 0 0

2 0 1 0

**Note:** One-hot encoding can significantly increase the dimensionality of the dataset if the categorical variable has many unique values.

### **2. Label Encoding**

**When to Use:**

* For **ordinal categorical variables** (categories with a meaningful order).​
* When there are no inherent ordinal relationships, but the model can interpret arbitrary numerical values meaningfully (e.g., tree-based models).​

**Description:** Assigns a unique integer to each category.​

**Example:**

from sklearn.preprocessing import LabelEncoder

import pandas as pd

df = pd.DataFrame({'Animal': ['Cat', 'Dog', 'Horse']})

label\_encoder = LabelEncoder()

df['Animal\_encoded'] = label\_encoder.fit\_transform(df['Animal'])

print(df)

**Output:**

Size Animal\_encoded

0 Cat 2

1 Dog 1

2 Horse 0

**Note:** Label encoding can mislead models that assume numerical relationships between categories, as it introduces an ordinal relationship

### **3. Ordinal Encoding**

**When to Use:**

* For **ordinal categorical variables** where the categories have a clear, meaningful order.​

**Description:** Similar to label encoding but specifically used when the categorical variable has an inherent order.​

**Example:**

from sklearn.preprocessing import OrdinalEncoder

import pandas as pd

df = pd.DataFrame({'Size': ['Small', 'Medium', 'Large']})

size\_order = [['Small', 'Medium', 'Large']]

ordinal\_encoder = OrdinalEncoder(categories=size\_order)

df['Size\_encoded'] = ordinal\_encoder.fit\_transform(df[['Size']])

print(df)

**Output:**

Size Size\_encoded

0 Small 0.0

1 Medium 1.0

2 Large 2.0

**Note:** Ordinal encoding is appropriate when the order of categories matters, such as 'Low', 'Medium', 'High'.

### **4. Frequency (Count) Encoding**

**When to Use:**

* For categorical variables with many unique categories (high cardinality).​
* When the frequency of categories is informative.​

**Description:** Replaces each category with its frequency or count in the dataset.​

**Example:**

import pandas as pd

df = pd.DataFrame({'City': ['Paris', 'Paris', 'London', 'Tokyo', 'London', 'London']})

frequency\_encoding = df['City'].value\_counts().to\_dict()

df['City\_encoded'] = df['City'].map(frequency\_encoding)

print(df)

**Output:**

City City\_encoded

0 Paris 2

1 Paris 2

2 London 3

3 Tokyo 1

4 London 3

5 London 3

**Note:** Frequency encoding can be useful but may introduce leakage if not handled properly, especially if the target variable influences category frequencies.

### **5. Binary Encoding**

**When to Use:**

* For categorical variables with high cardinality.​
* When you want to reduce dimensionality compared to one-hot encoding.​

**Description:** Converts categories into binary digits and represents them in separate columns.​

**Example:**

import pandas as pd

import numpy as np

df = pd.DataFrame({'Category': ['A', 'B', 'C', 'D']})

binary\_encoding = {}

unique\_categories = df['Category'].unique()

for i, category in enumerate(unique\_categories):

# Convert the integer to binary and store as a list of integers

binary\_encoding[category] = [int(x) for x in np.binary\_repr(i, width=2)]

# Create new binary columns based on the encoding

for j in range(2): # 2 bits needed for 4 categories

df[f'Category\_bin\_{j}'] = df['Category'].map(lambda x: binary\_encoding[x][j])

print(df)

**Output:**

Category Category\_bin\_0 Category\_bin\_1

A 0 0

**06. Feature Engineering**

Feature Engineering is the process of creating, transforming, or selecting input variables (features) that help improve the performance of machine learning models.

### **🔷 1. Feature Creation**

Creating new features based on existing ones.

Example:

# Creating BMI from weight and height

df['BMI'] = df['weight\_kg'] / (df['height\_m'] \*\* 2)

# Creating age groups

df['age\_group'] = pd.cut(df['age'], bins=[0,18,35,60,100], labels=['Teen','Young','Adult','Senior'])

### **🔷 2. Binning / Discretization**

Turning continuous data into categorical bins.

# Binning income

df['income\_bin'] = pd.cut(df['income'], bins=3, labels=["Low", "Medium", "High"])

### **🔷 3. Date-Time Feature Extraction**

Extracting useful features from datetime columns.

df['purchase\_date'] = pd.to\_datetime(df['purchase\_date'])

df['purchase\_day'] = df['purchase\_date'].dt.day

df['purchase\_month'] = df['purchase\_date'].dt.month

df['purchase\_weekday'] = df['purchase\_date'].dt.day\_name()

Perfect! Here's a complete guide to **Feature Selection and Feature Importance** using different methods, with **code examples** and when to use them:

## **07. ✅ Feature Selection Techniques**

### **1️⃣ Univariate Feature Selection (ANOVA, Chi²)**

Select features based on statistical tests.

#### **🔹 ANOVA F-test (for numerical features → classification target)**

**from sklearn.feature\_selection import SelectKBest, f\_classif**

**selector = SelectKBest(score\_func=f\_classif, k=5)**

**X\_new = selector.fit\_transform(X, y)**

**selected\_features = X.columns[selector.get\_support()]**

**print(selected\_features)**

#### **🔹 Chi-Square Test (for categorical features → classification target)**

**from sklearn.feature\_selection import SelectKBest, chi2**

**# Ensure X is non-negative for chi2**

**X\_cat = X.select\_dtypes(include='int') # or apply encoding before**

**selector = SelectKBest(score\_func=chi2, k=5)**

**X\_new = selector.fit\_transform(X\_cat, y)**

**selected\_features = X\_cat.columns[selector.get\_support()]**

**print(selected\_features)**

### **2️⃣ Model-Based Feature Importance (Tree-Based Models)**

#### **🔹 Random Forest**

**from sklearn.ensemble import RandomForestClassifier**

**import pandas as pd**

**model = RandomForestClassifier()**

**model.fit(X, y)**

**importances=pd.Series(model.feature\_importances\_, index=X.columns)**

**importances.sort\_values(ascending=False).plot(kind='barh')**

#### **🔹 ExtraTreesClassifier (alternative tree ensemble)**

**from sklearn.ensemble import ExtraTreesClassifier**

**model = ExtraTreesClassifier()**

**model.fit(X, y)**

**importances = pd.Series(model.feature\_importances\_, index=X.columns)**

**importances.nlargest(10).plot(kind='barh')**

### **3️⃣ Recursive Feature Elimination (RFE)**

Select features recursively by eliminating least important ones.

**from sklearn.feature\_selection import RFE**

**from sklearn.linear\_model import LogisticRegression**

**model = LogisticRegression()**

**rfe = RFE(estimator=model, n\_features\_to\_select=5)**

**rfe.fit(X, y)**

**selected\_features = X.columns[rfe.support\_]**

**print(selected\_features)**

### 

### 

### **4️⃣ XGBoost**

**from xgboost import XGBClassifier**

**import pandas as pd**

**model = XGBClassifier()**

**model.fit(X, y)**

**importances = pd.Series(model.feature\_importances\_, index=X.columns)**

**importances.sort\_values(ascending=False).plot(kind='barh')**

## **✅ Quick Summary: When to Use What?**

| **Method** | **Use Case** | **Model Type** |
| --- | --- | --- |
| **ANOVA (f\_classif)** | Numeric features & classification | Classification |
| **Chi² Test** | Numeric features & classification | Classification |
| **RFE** | General feature ranking | Classification/Reg |
| **Random Forest / XGB** | Model-based, works on all features | Classification/Reg |
| **SelectKBest** | Quick filter-based approach | Classification |

**07. Check for Class Imbalance (For classification)**

Class imbalance occurs when some classes have significantly more samples than others. This can lead to:

* Biased models that favor the majority class.
* Poor precision/recall for minority classes

**🚨 What to Do If There’s Imbalance?**

| **Method** | **Description** | **What to use** |
| --- | --- | --- |
| Oversampling | Add copies of minority class | SMOTE, RandomOverSampler,SMOTENC |
| Undersampling | Remove samples from majority class | RandomUnderSampler |

#### **📌 Example Using SMOTE**

**from imblearn.over\_sampling import SMOTE**

**smote = SMOTE()**

**X\_resampled, y\_resampled = smote.fit\_resample(X, y)**

#### **📌 Example Using SMOTENC**

### 1. **Identify Categorical Columns**

You need to pass column indices (not names) of categorical features.

**categorical\_cols = ['gender', 'job\_type'] # example categorical columns**

**categorical\_indices = [X.columns.get\_loc(col) for col in categorical\_cols]**

### **2. Apply SMOTENC**

**from imblearn.over\_sampling import SMOTENC**

**smote\_nc = SMOTENC(categorical\_features=categorical\_indices, random\_state=42)**

**X\_resampled, y\_resampled = smote\_nc.fit\_resample(X, y)**

**08. Split Dataset**

Split your dataset into features (X) and target (y).

**X = df\_resampled.drop('target\_column\_name', axis=1)**

**y = df\_resampled['target\_column\_name']**

**09. Train-Test Split**

Train-test split is a technique used to divide your dataset into two parts:

* Training - Used to train the machine learning model
* Testing - Used to evaluate how well the model performs on unseen data

**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, # 20% for testing**

**random\_state=42, # ensures reproducibility**

**stratify=y # useful for classification problems**

**)**

**10. Model Building**

### **🔵 Regression using LinearRegression**

**from sklearn.linear\_model import LinearRegression**

**from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score**

**model = LinearRegression()**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

**Evaluation Metrics:**

**print("MAE:", mean\_absolute\_error(y\_test, y\_pred))**

**print("MSE:", mean\_squared\_error(y\_test, y\_pred))**

**print("MSE:", mean\_squared\_error(y\_test, y\_pred))**

**print("R² Score:", r2\_score(y\_test, y\_pred))**

### **🔴 Classification using RandomForestClassifier**

**from sklearn.ensemble import RandomForestClassifier**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score**

**model = RandomForestClassifier()**

**model.fit(X\_train, y\_train)**

**y\_pred = model.predict(X\_test)**

**Evaluation Metrics:**

**print("Accuracy:", accuracy\_score(y\_test, y\_pred))**

**print("Precision:", precision\_score(y\_test, y\_pred))**

**print("Recall:", recall\_score(y\_test, y\_pred))**

**print("F1 Score:", f1\_score(y\_test, y\_pred))**

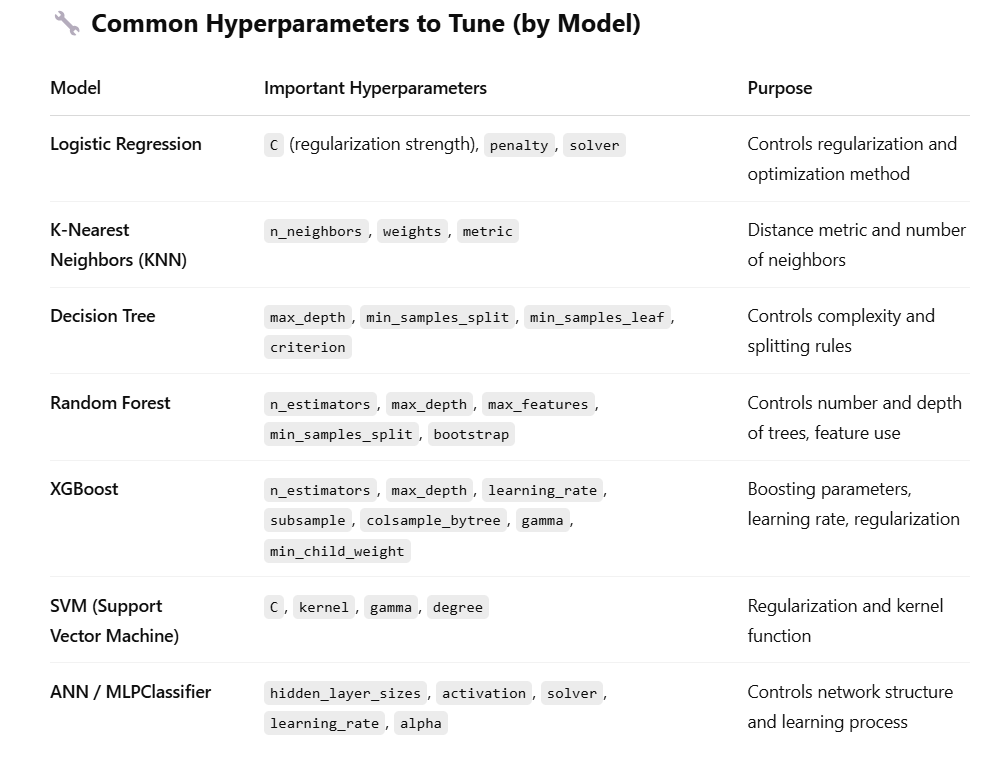
**11. HyperParameter Tuning**

### **✅ Why Hyperparameter Tuning?**

* Goal: To find the best set of parameters that improves model performance (accuracy, F1 score, etc.)
* Hyperparameters are not learned during training; they are set before training.
* Helps avoid underfitting or overfitting, improves generalization on unseen data.

### **🛠️ How to Perform Hyperparameter Tuning**

* **Grid Search: Tries all combinations  
   from sklearn.model\_selection import GridSearchCV**
* **Random Search: Tries a random subset  
   from sklearn.model\_selection import RandomizedSearchCV**



|  |
| --- |

**Example code:**

**from xgboost import XGBClassifier**

**param\_grid = {**

**'n\_estimators': [100, 200],**

**'max\_depth': [3, 6, 10],**

**'learning\_rate': [0.01, 0.1],**

**'subsample': [0.8, 1],**

**'colsample\_bytree': [0.8, 1]**

**}**

**grid = GridSearchCV(XGBClassifier(), param\_grid, cv=5, scoring='accuracy')**

**grid.fit(X\_train, y\_train)**

**print("Best Parameters:", grid.best\_params\_)**

**12. Deployment**

You can deploy your trained machine learning model using frameworks like Streamlit, Flask, or Django for web applications, or use cloud platforms like AWS, Azure, or Google Cloud for scalable and production-level deployment.