Train-Test Split

🔲 What Is It?

Train-test split is the process of dividing your dataset into **two parts**:

- **Training Set** → Used to **train** the model
- **Test Set** → Used to **evaluate** how well the model performs on **unseen data**

🎯 Why Do We Use It?

If we train and test on the **same data**, the model may **memorize** it (overfitting) instead of learning true patterns.

Test data = a simulation of real-world data.
It helps us check if the model can generalize well.

Typical Split Ratios

Training Set	Test Set
80%	20%
70%	30%
75%	25%

In some cases, we may also have a **validation set** (3-way split):

- 60% Train
- 20% Validation
- 20% Test

Why 20% Validation?

- Used for **model tuning** (e.g. choosing the best hyperparameters).
- Helps detect **overfitting** during training.
- **When making model selection**

Dataset	**Size To do**
Small	Do cross-validation (e.g. 5-fold CV)
Medium	Share Train/Validation/Test
Big	Train/Validation/Test + Cross-validation
Very big	Just Train/Test, because enough data will work as validation

Data Size: According to Row Count (Observations)

Data Type	Small Dataset	Medium Dataset	Large Dataset
General ML Data	< 1,000 rows	1,000 – 100,000 rows	> 100,000 rows
Text Data (NLP)	< 5,000 docs	5,000 – 100,000 docs	> 100,000 docs
Image Data	< 10,000 images	10,000 – 100,000	> 100,000+ images
Time Series	< 365 timestamps	365 – 10,000+	> 10,000+

What is a hyperparameter?

Hyperparameters are settings or values that you specify **before** training a model . They affect the learning process of the model.

Example (unavailable hyperparameter):

- Decision Tree → max_depth, min_samples_split
- Random Forest → n_estimators, max_features
- SVM → C, gamma
- Neural Network → learning rate, batch size, epochs, number of layers

What does Hyperparameter Tuning mean?

Hyperparameter tuning means **finding the most appropriate values** for these hyperparameters so that your model performs best on the validation data.

Summary

- ✅ Prevents **overfitting**
- ✅ Helps assess **generalization**

  (How well a trained model performs on new, unseen data.)
- ✅ Ensures **unbiased** model evaluation

Parameters of train_test_split

1)random_state: Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls.

What's the problem if it's not 42?"

Answer: No problem. You can give or even –
if you want .random_state =7,123,2025,0

from sklearn.model_selection import train_test_split

X = [1, 2, 3, 4, 5]

y = ['a', 'b', 'c', 'd', 'e']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

- X_train = [3, 1, 4]
- X_test = [5, 2]
- y_train = ['c', 'a', 'd']
- y_test = ['e', 'b']

Need to understand:

- train_test_splitXIt y **shuffles** at first , but random_state=42 it shuffles in **a predictable way .**
- Then out of the 5 items, 40%, meaning **2, go to the test set and** the remaining 60%, meaning **3, go to the train set.**
- X And y shuffle together so that the relationship 1 between a, 2 etc. **is not broken.**b
-
- **You get the same train-test split every time** you run the code
- Others can **replicate your results** exactly
-

✅**shuffle:**

Controls whether the data is shuffled before splitting.

Default is True. That’s good because datasets often have ordered labels.

If False, the model may learn from patterns in order, which is bad.

When to use shuffle=False:

In **time series data**, where order matters.

What is Time Series Data?

Time series data is data **collected over time**, usually in order — for example:

Daily stock prices 📈

Hourly weather readings 🌦

Monthly sales data 📊

If You Use shuffle=True:

- It may randomly select:
- Training: 2023-01-01, 2023-01-04, 2023-01-06
- Test: 2023-01-02, 2023-01-03, 2023-01-05
- ⇥ **Problem:** You are using **future prices to predict the past**, which breaks time logic and causes **data leakage**.
-

✅ **Stratify:**

➤ **Ensures the class distribution is the same in both train and test sets.**

- Especially important for **classification problems** with **imbalanced classes**.

What It Does:

- If class "A" is 80% and class "B" is 20% in the full dataset,
- The train and test sets will **preserve the same 80/20 ratio**.

✅ **stratify vs random_state vs shuffle**

Parameter	Purpose	Common Use Case	Key Point
random_state	Controls the randomness for reproducible splits	Ensures you get the **same split** every time	Like setting a **seed** for random number
shuffle	Decides whether to **shuffle the data before splitting**	Useful for randomizing data	❗ **Avoid in time series**
stratify	Ensures **equal class distribution** in train/test sets	Used in **classification** (imbalanced data)	Keeps same % of each class in all sets