Data Splitting

🧠 The Core Mental Model (Memorize This)

Think of ML like **studying for an exam**.

- **Training set** → What you study from
- **Test set** → A mock exam you've never seen
- **Real Kaggle test** → The final exam

You **must not peek** at the answers during the exam.

1️⃣ Start From the Raw Dataset

Your dataset has:

```
Features (X):everythingexceptPrice
Target (y):Price
```

So conceptually:

```
X = [Airline, Source, Destination, Duration, Stops, ...]
y = [Price]
```

Correct so far ✅

2️⃣ Why Do We Split the Data?

Because we want to answer this question:

> "Will my model work on new, unseen flights?"

If you train and evaluate on the same data:

- Model memorizes
- You get falsely high accuracy
- Real-world performance collapses

This is called **overfitting**.

3️⃣ What the Split Actually Means

When you split:

```
X = [Airline, Source, Destination, Duration, Stops, ...]
y = [Price]

X → X_train, X_test
y → y_train, y_test
```

You are saying:

> "I will only learn patterns from X_train & y_train
>
> and **only evaluate** on X_test & y_test"

4️⃣ What Each Piece Is Used For

Let's go one by one.

🔵 X_train (Features for learning)

This is:

- Airline
- Route
- Duration

- Stops
- etc.

But **without Price**.

The model looks at this and tries to learn:

> "How do these features relate to price?"

🔵 y_train (Correct answers)

This is:

- The actual flight prices
- Known during training

The model compares:

```
Predicted price vs Actual price
Predicted X_test vs Actual y_test
```

and adjusts itself.

👉 **This is where learning happens**

🟢 X_test (Features for evaluation)

This is:

- Same feature columns
- Flights the model has **never seen**

You give X_test to the trained model and say:

> "Predict prices for these flights"

Important:

🚫 The model has **never seen y_test yet**

🟢 y_test (Hidden answer key)

This is:

- The real prices of X_test flights
- Used **only for evaluation**

You compare:

```
Prediction(X_test) vs y_test
```

This tells you:

> "How well does my model generalize?"

5️⃣ The Flow (Very Important)

Here is the **correct mental sequence**:

```
1. Train modelon (X_train, y_train)
2. Predict pricesusing X_test
3.Compare predictionswith y_test
4. Computeerror (RMSE, MAE, etc.)
```

🚫 You NEVER:

- Train on X_test
- Tune hyperparameters using y_test repeatedly

6️⃣ Why Not Use X_test During Training?

Because that would be:

- Seeing exam questions beforehand

- Data leakage
- Unrealistically good results

Kaggle punishes this hard.

7️⃣ Where Kaggle Leaderboard Fits In

Your Kaggle dataset actually has:

Dataset	Purpose
Train.csv	You split internally
Test.csv	Kaggle's hidden exam

So:

- Your **X_test / y_test** = practice exam
- Kaggle **test.csv** = final exam

You never see Kaggle's y_test.

8️⃣ Common Beginner Confusion (You're Not Alone)

> "We compare predictions with price — where exactly?"

Answer:

- During training → compare with **y_train**
- During evaluation → compare with **y_test**
- During Kaggle submission → Kaggle compares silently

9️⃣ Interview-Grade Explanation

If asked:

> “Why do we need X_test and y_test?”

You say:

> X_test represents unseen data used to evaluate generalization. y_test is the ground truth used only to assess performance, not for training.

Error 🤧

You compute **error on both**:

- **Training set** → training error
- **Test set** → testing error

But **you use them for different purposes**.

🧠 Why Compute Training Error?

Training error answers:

> “How well did my model learn the patterns in the data it was allowed to see?”

If training error is:

- **High** → model is underfitting
- **Very low** → model is powerful enough to fit training data

But **low training error alone means nothing**.

🧠 Why Compute Testing Error?

Testing error answers:

> “How well does my model generalize to unseen data?”

This is the number you actually care about.

🧠 The Two Errors Tell Different Stories (Corrected)

Training Error	Testing Error	What It Means	Intuition
High	**High**	**Underfitting**	Model is too simple, hasn't learned patterns
Low	**High**	**Overfitting**	Model memorized training data, poor generalization
Low	**Low**	**Good fit**	Model learned true underlying patterns
Very Low	**Slightly Higher**	**Healthy generalization**	Expected behavior; small gap is normal

🔍 How to *Think* About This (Not Just Memorize)

1️⃣ High Train + High Test → Underfitting

- Linear model too simple
- Missing important features
- Poor encoding

💡 Fix:

- Better features
- More expressive model

2️⃣ Low Train + High Test → Overfitting 🚨

- Too many features
- High-cardinality OHE
- No regularization

💡 Fix:

- Regularization (L1/L2)
- Drop noisy features

- Cross-validation

3️⃣ Low Train + Low Test → Sweet Spot 🎯

- Rare but beautiful
- This is what you want

💡 Action:

- Submit to Kaggle 😄

4️⃣ Very Low Train + Slightly Higher Test → Ideal Reality

- Perfect training fit is suspicious
- Small test increase is normal

💡 Rule:

> A small gap is healthy. Zero gap is suspicious.

📌 Kaggle-Specific Wisdom

- **Local CV > single test split**
- **Private LB often aligns with CV, not public LB**
- Large train–test gap = leaderboard pain later

🧪 Practical Advice for Your Project

When you train your **baseline linear regression**, expect:

- Train RMSE lower than test RMSE
- If both are high → features need work

That's not failure — that's *information*.

🔥 Interview-Ready One-Liner

> Training error diagnoses learning capacity, while test error estimates generalization. Their gap reveals under- or overfitting.

2️⃣ Why We Look at the GAP

The **gap** between training and testing error is critical.

Example:

```
TrainRMSE=300
TestRMSE=1200
```

This screams:

> "Model memorized training data"

Kaggle leaderboard drop incoming 😄

3️⃣ What About Kaggle Submissions?

Here's a subtle but important point:

- You **do not** compute error on Kaggle test data
- Kaggle computes it for you (hidden y)

So your **local test RMSE** is your best proxy for leaderboard score.

4️⃣ Very Common Beginner Mistake (Avoid This)

❌ Tuning model until test error is minimum

❌ Reusing test set again and again

That makes your test set "not unseen" anymore.

Correct approach:

- Use cross-validation for tuning
- Use test set once as a sanity check

5️⃣ Kaggle-Grade Best Practice (Mental Model)

Think in **3 layers**:

1. **Training set**
    - Used to fit model
    - Measure training error
2. **Validation set / CV**
    - Used to tune model
    - Compare models
3. **Test set**
    - Used once
    - Final unbiased estimate

In Kaggle:

- CV ≈ leaderboard proxy
- Private LB ≈ real-world test

6️⃣ Interview-Ready Explanation

If asked:

> “Do you compute error on training and test data?”

Answer:

> Yes. Training error helps diagnose underfitting, while test error estimates generalization. The gap between them indicates overfitting.

7️⃣ Final Takeaway (Tattoo This 😄)

> Training error tells you how well you learned
>
> **Testing error tells you whether you learned the right thing**