Preparing Data and Train/Test Split

A model must be tested on data it has never seen, or its score means nothing. Features,

Introduction

Before any model can learn, your data has to be in shape — and before you believe any score, the model must be tested on data it has never seen. These two ideas, data preparation and the train/test split, are where most beginner projects quietly go wrong.

First, two words you will use in every lesson from here on. **Features** are the inputs the model is allowed to look at — hours studied, weight of a mango, words in an email. **Labels** are the answers — pass or fail, price, spam or not spam. A dataset is a table where each row is one example, the feature columns describe it, and the label column holds the answer. Preparing data means making that table clean and honest: missing values filled or removed, numbers in a sensible form, and no column that secretly contains the answer.

Then comes the split. If you train a model on all your data and test it on the same data, you are testing memory, not learning. It is like letting a student take an exam using the exact questions they practised on — a perfect score tells you nothing. So you hold some data back: train on one part, test on the other part. The usual split is 80% for training and 20% for testing, and the test part must stay unseen until the very end.

Cleaning deserves a closer look, because it is where beginners lose the most marks without realising it. Take missing values: a student row with no "hours studied" cannot just be left blank, because most models need a number in every cell. You have two honest choices — fill it with a sensible value, such as the average hours of the other students, or drop the row entirely. Filling is gentler but invents data; dropping is cleaner but shrinks your dataset. Neither choice is free, and the choice matters more when many rows are missing.

The same care applies to categories. A model cannot read the word "morning" or "evening" as a number, so categories must be turned into numbers first — for example, morning = 0, evening = 1. And a value that looks like a number can still be wrong: a student who "studied 100 hours" in one day is almost certainly a typing error, not a genius. Real data is full of such errors, and finding them is part of the job, not a chore before the job.

(Roman Urdu: Model ko woh data par test karo jo us ne kabhi dekha hi na ho — warna score sirf yaad-dasht ka saboot hai, seekhne ka nahi.)

Key points

- **Features are inputs, labels are answers.** In a pass/fail table, "hours studied" is the feature and "pass/fail" is the label. Never mix them up.
- **Preparing data means cleaning it first.** Handle missing values, fix obvious errors, and make sure every column is in a usable form before training.

- **Split before you train, not after.** The test set must be separated before the model sees anything, or the test is already contaminated.
- **The standard split is 80/20.** Train on 80% of the rows, test on the remaining 20%. With very small datasets the test set is tiny, so treat its score as rough, not exact.
- **The test set answers one question only:** "does this model work on new data?" A high training score with a low test score means the model memorised instead of learning (lesson ML.7 explains this fully).
- **Data leakage is cheating by accident.** Leakage means information about the answer sneaks into the features — for example, a "final result" column left in the training table. The model then looks brilliant in testing and useless in real life.
- **Shuffle before splitting.** If your data is ordered (all passes first, then all fails), a plain split gives a training set of one kind and a test set of another. Shuffling mixes the rows first.
- **A validation set is a third split, used while tuning.** Train on the training set, choose settings using a validation set, and keep the test set locked away for the final check only.
- **Never use the test set to choose your model.** Peeking at test results and then changing the model turns the test set into training data, and its score stops being honest.
- **Categories must become numbers before training.** Words like "morning" or "pass" need a numeric form, and the same mapping must be used for training and for new data.

Examples

Example 1 — A tiny dataset, split by hand

Ten students, hours studied and pass/fail. Shuffle the rows, then take 8 for training and 2 for testing:

Training (8 rows): (1, Fail), (2, Fail), (3, Fail), (4, Fail), (5, Pass), (6, Pass), (7, Pass), (9, Pass). Test (2 rows): (8, Pass), (2, Fail).

Train the model on the 8 rows only. Then, and only then, ask it to predict the 2 test rows. If it gets both right, that is real evidence — it never saw those rows during training.

Example 2 — The split in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. It uses only standard scikit-learn calls, and a tiny hand-made dataset of 10 rows:

```
from sklearn.model_selection import train_test_split

hours = [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
result = [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # 0 = Fail, 1 = Pass

hours_train, hours_test, result_train, result_test = train_test_split(
    hours, result, test_size=0.2, random_state=42
)
print(len(hours_train), "training rows,", len(hours_test), "test rows")
```

What the result means: the data is split into 8 training rows and 2 test rows. `random_state=42` shuffles the rows in a fixed way so the split is the same every time you run it — handy for checking your own work. The model will be trained only on the training rows, and the test rows stay locked away until evaluation.

Example 3 — Data leakage in plain words

A hospital model predicts whether patients will be readmitted. Someone leaves a column called "discharge letter sent" in the features — but that letter is only sent to patients who are being readmitted. The model learns "letter sent = readmitted" and scores nearly perfectly in testing. In the real hospital, the letter does not exist yet at prediction time, so the model is useless. The feature contained the answer. That is leakage: always ask, "would I know this feature at the moment I need the prediction?"

Example 4 — Worked example: cleaning a messy table, step by step

Six students, as recorded — with problems:

| Student | Hours | Result | |---|---|---| | A | 5 | Pass | | B | (blank) | Fail | | C | 100 | Pass | | D | 5 | Pass | | E | 3 | Fail | | F | 3 | Fail |

Step 1: student C's 100 hours is an error (nobody studies 100 hours in a day), so fix it or drop the row — here, drop it, because guessing the true value would invent data. Step 2: student B's missing hours can be filled with the average of the remaining hours (5 + 5 + 3 + 3) ÷ 4 = 4, or B can be dropped; either is defensible, but you must record which you chose. Step 3: check for duplicates — none here, but a student recorded twice would count twice in training and tilt the model. The cleaned table (dropping C, filling B with 4) is: (5, Pass), (4, Fail), (5, Pass), (3, Fail), (3, Fail). Only now is it ready to split and train on.

Example 5 — Worked example: a stratified split, in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. The plain split in Example 2 shuffles rows, but with unbalanced data it can by bad luck put almost all of one class in the test set. A stratified split keeps the class balance the same in both sets:

```
from sklearn.model_selection import train_test_split

hours = [[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]]
result = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1]  # 0 = Fail, 1 = Pass

hours_train, hours_test, result_train, result_test = train_test_split(
    hours, result, test_size=0.3, stratify=result, random_state=42
)
print("Train labels:", result_train)
print("Test labels:", result_test)
```

Run it and check: the test set keeps roughly the same share of passes as the whole data, instead of depending on luck. With small or unbalanced data, `stratify=` is a cheap habit that makes your test score much more trustworthy. Without it, one unlucky shuffle can make a good model look bad.

Real-world case — Cleaning data for a real online shop

A small online shop wants to predict which visitors will buy. Its raw data is messy: some visitors have no age recorded, some accounts appear twice (the same person signed up with two emails), and a few orders show a price of zero because of a discount bug. The shop's data scientist spends most of her first week just on this cleaning — filling or dropping missing ages, merging duplicate accounts, removing zero-price orders — before any model is trained. Beginners often think this week was wasted. In real projects, it is usually the most valuable week: a model trained on the raw, dirty table would have learned that "price zero" means "will buy", which is exactly the kind of false pattern that fails in real life.

Quick recap

- Features are inputs; labels are answers. Keep them separate and clean.
- Split your data before training: usually 80% train, 20% test.
- The test set must be unseen data, or the score proves nothing.
- Shuffle before splitting so both sets are mixed fairly.
- Data leakage = the answer hiding inside a feature. Remove it.

Common mistakes

1. **Testing on the training data** — a perfect score that means nothing. Fix: always hold out a test set before training, and report the test score, not the training score.
2. **Splitting after peeking at the whole dataset** — choosing features or cleaning using all rows leaks test information into training. Fix: split first, then do your preparation using the training set only.
3. **Leaving an answer-like column in the features** — the model cheats without you noticing. Fix: for every feature, ask "would I know this before I need the prediction?"

1. **Filling missing values using the whole dataset, test rows included** — the average you fill with then carries test information into training. Fix: compute fill values from the training set only, then apply them to the test set.
2. **Dropping every row with any missing value** — with many columns, you can lose most of your data this way. Fix: drop a row only when the missing value matters, and fill sensible values otherwise.

Practice

1. Take any 10-row table from an earlier lesson, shuffle the rows by hand, and mark 8 as train and 2 as test.
2. Write down, for the pass/fail example, one feature that would be leakage (a column that gives the answer away).
3. Run the code in Example 2 on your own computer (after `pip install scikit-learn`) and change `test_size` to 0.3. How many test rows do you get now?

1. Take the cleaned table in Example 4 and split it into train and test by hand. With only 5 rows, what split would you choose, and why is any score from such a tiny test set only rough?

2. Write down two features for a "will this student pass?" model: one that is safe to use, and one that would be leakage. Explain the leakage one in a single sentence.
3. Run the code in Example 5 without `stratify=` and with three different `random_state` values. Does the test-set balance stay the same? Write one line about what you notice.

Practice quiz

1. In ML, what are features and labels?

- Features are answers, labels are questions
- Features are the inputs, the label is the answer to predict
- Both are the same
- Features are charts

Answer: Features are the inputs, the label is the answer to predict

2. Why must you test on data the model has never seen?

- To make testing harder
- To prove it learned the pattern instead of memorising the training data
- Unseen data is cleaner
- It is a tradition

Answer: To prove it learned the pattern instead of memorising the training data

3. What is data leakage?

- Data spilling from a file
- Test answers sneaking into training, so the score looks better than it really is
- A slow download
- Missing values

Answer: Test answers sneaking into training, so the score looks better than it really is

---

Watch the video lesson: https://youtu.be/THefPM30FhQ

Read online: https://scopetreasure.com/notes/ml-data-prep-train-test · More free tools and notes at https://scopetreasure.com