A Complete Small Project

Put it all together: a tiny dataset, a train/test split, two models trained and

Introduction

You have learned the pieces: data preparation, the train/test split, regression, classification, decision trees, overfitting and evaluation. This lesson joins them into one complete project, end to end. It is deliberately small — 12 rows, two models, one comparison — because a small project you fully understand beats a big project you copied without understanding.

The project question is simple: can study habits predict whether a student passes? The features are hours studied and class attendance; the label is pass or fail. You will split the data, train two different models, compare their test scores honestly, and then write down what you found — including what the project cannot prove. That last part is not decoration. Every real project report ends with limits, because every model has them, and naming them is what makes your work trustworthy.

By the end you will have done, in miniature, exactly what working data scientists do: question, data, split, models, comparison, findings, limits. The only difference between this and a professional project is size.

Before the first line of code, a good project also writes down what "success" means. For this project, success could be "the model beats a simple rule — predict Pass for 5+ hours — on the test set". That sentence matters because it gives you something to compare against: a model that cannot beat a one-line rule has not earned its complexity. Professionals call that simple rule a **baseline**, and every project has one. Without a baseline, any score looks impressive; with one, you know whether the model actually added value.

A second habit worth naming is **reproducibility**: someone else should be able to re-run your project and get your numbers. That is why the code fixes `random_state=42` — the same shuffle every time — and why a real report notes the data source, the split and the model settings. A result nobody can reproduce is a story, not a finding. Small habits — fixed seeds, saved code, written settings — are what turn a classroom exercise into work you can stand behind.

(Roman Urdu: Is lesson mein sab kuch aik chhote project mein jorenge — data, split, do models, muqabla, aur imandari se natija. Chhota project jo samajh aaye, bare na-samajh project se behtar hai.)

Key points

- **Every project starts with a question, not a model.** "Can study habits predict passing?" decides the data, the labels and the way you judge success.
- **The steps always run in order:** question → data → clean and split → train → test → compare → report findings and limits. Skipping the split ruins everything after it.
- **Train more than one model.** Different models make different mistakes; comparing two honest test scores teaches you more than polishing one model.

- **Compare on the test set only.** Comparing training scores is comparing memories, not skill (lesson ML.7).
- **A small test set means a rough score.** With only 3 test rows, one row flipping changes the score by a third. Say so in your findings — do not pretend the score is precise.
- **Findings say what happened; limits say what it does not prove.** Both belong in every report, even a five-line one.
- **Keep the project honest and small.** Tiny hand-made data is fine for learning, but it cannot prove anything about real students. Name that limit.
- **Always beat a baseline first.** A one-line rule (for example, "5+ hours → Pass") is the score to beat; a model that cannot beat it has not earned its complexity.
- **Reproducibility is part of the result.** Fixed random seeds, saved code and written-down settings let someone else re-run your project and get your numbers.
- **Error analysis comes after the score.** Look at which test rows the model got wrong and ask why — the wrong rows teach more than the score does.

Examples

Example 1 — The project data and split

Twelve students, with hours studied, attendance (1 = regular, 0 = irregular) and the true result:

| Hours | Attended | Result | |---|---|---| | 1 | 0 | Fail | | 2 | 0 | Fail | | 2 | 1 | Fail | | 3 | 0 | Fail | | 4 | 1 | Fail | | 5 | 1 | Pass | | 6 | 0 | Fail | | 6 | 1 | Pass | | 7 | 1 | Pass | | 8 | 0 | Pass | | 8 | 1 | Pass | | 9 | 1 | Pass |

Split: 9 rows for training, 3 held out for testing (a 75/25 split, close to the usual 80/20, adjusted for a tiny table). The test rows are chosen before training and never touched until evaluation: (6, 0, Fail), (8, 0, Pass) and (3, 0, Fail). Notice the pattern is not perfectly clean — a 6-hour irregular student failed while an 8-hour irregular student passed. Real data is messy like this.

Example 2 — Train two models and compare (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. It uses only standard calls: `train_test_split`, `LogisticRegression`, `DecisionTreeClassifier`, `.fit` and `.score`:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

X = [[1, 0], [2, 0], [2, 1], [3, 0], [4, 1], [5, 1],
     [6, 0], [6, 1], [7, 1], [8, 0], [8, 1], [9, 1]]
y = [0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1]  # 0 = Fail, 1 = Pass

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)
log = LogisticRegression().fit(X_train, y_train)
tree = DecisionTreeClassifier(max_depth=2, random_state=42).fit(X_train, y_train)
print("Logistic regression test score:", log.score(X_test, y_test))
print("Decision tree test score:", tree.score(X_test, y_test))
```

What the result means: each model trains on the same 9 rows and is judged on the same 3 unseen rows, so the comparison is fair. Whichever scores higher on those 3 rows wins *this* comparison — but with only 3 test rows, a one-row difference swings the score a lot, so treat the winner as a hint, not a verdict. The decision tree's predictions can also be read back as a flowchart (lesson ML.6), which is a second reason to like it even if its score ties.

Example 3 — Findings and limits in five lines

1. Both models predicted most test students correctly, and the tree's rules were easier to explain.
2. Hours studied was the strongest signal; attendance mattered only for borderline students.
3. The messy rows (6 hours, irregular, failed) show the pattern is not a clean rule.
4. Limit: only 12 hand-made rows, so nothing here proves anything about real students.
5. Limit: with 3 test rows, the scores are rough — a bigger, real dataset is needed before trusting either model.

Five lines, no exaggeration, limits included. That is what a good project report looks like at any size.

Example 4 — Worked example: error analysis on the project

Suppose the decision tree from Example 2 predicts the three test rows like this:

| Test student | Hours | Attended | True result | Tree predicts | Right? | |---|---|---|---|---|---| | A | 6 | 0 | Fail | Pass | Wrong | | B | 8 | 0 | Pass | Pass | Right | | C | 3 | 0 | Fail | Fail | Right |

The score is 2 out of 3, about 67%. But the interesting row is A: a 6-hour irregular student who failed, predicted as Pass. Why? Because in the training rows, most 6+ hour students passed, and the tree's rule "6+ hours → Pass" overrode the attendance signal. The training data itself contains the clue: the only other 6-hour irregular student in the full table also failed. A good report does not just say "67%"; it says "the model missed the irregular 6-hour student, because attendance matters more at 6 hours than the tree's rule allows". That

one sentence is worth more than the score, because it points at the next fix: more training rows around 6 hours with mixed attendance.

Example 5 — Worked example: predicting new students, in code (scikit-learn)

This example needs scikit-learn installed first: `pip install scikit-learn`. It retrains both models from Example 2 and asks them about two brand new students, so you can see the comparison turn into actual decisions:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

X = [[1, 0], [2, 0], [2, 1], [3, 0], [4, 1], [5, 1],
     [6, 0], [6, 1], [7, 1], [8, 0], [8, 1], [9, 1]]
y = [0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1]  # 0 = Fail, 1 = Pass

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)
log = LogisticRegression().fit(X_train, y_train)
tree = DecisionTreeClassifier(max_depth=2, random_state=42).fit(X_train, y_train)
new_students = [[5, 0], [7, 0]]  # 5 hours irregular, 7 hours irregular
print("Logistic regression:", log.predict(new_students))
print("Decision tree:", tree.predict(new_students))
```

Run it and compare the two rows. If the models disagree on a student, that disagreement is information: it marks a case where the data is thin or the pattern is genuinely borderline. Do not hide disagreements in a report — name them, and say which model you would trust there and why (for example, the tree, because you can read its reasons). A report that shows a disagreement honestly is more trustworthy than one that quietly picks the prettier score.

Real-world case — A small shop's real first project

A small shop owner asks a student: "Can you predict which first-time visitors will come back?" The student copies this lesson's structure exactly. Question first, then data: 60 past visitors with two features (minutes spent on the site, pages viewed) and one label (returned or not). Split 45/15, train logistic regression and a depth-2 tree, compare on the same 15 test visitors. The tree wins narrowly, 11 out of 15 against 10 out of 15 — but the student reports it honestly: with 15 test rows, one visitor flipping changes the score by nearly 7 points, so the win is a hint, not proof. The shop still uses the tree's rule ("visitors who viewed 5+ pages usually return") as a rough guide for who gets a follow-up message. That is a real project: small, honest about its limits, and still useful the next morning.

Quick recap

- A complete project runs: question → data → split → train → test → compare → findings + limits.

- Train at least two models and compare them on the same test set.
- Test scores on tiny test sets are rough — say so, do not hide it.
- Findings report what happened; limits report what it does not prove.
- Small and understood beats big and copied.

Common mistakes

1. **Comparing models on training scores** — you compare memories, not skill. Fix: compare on the held-out test set, always.
2. **Writing findings without limits** — a report that claims too much loses trust. Fix: end every report with at least one honest limit.
3. **Picking the winner from a tiny test set and stopping** — one flipped row can change the winner. Fix: report the scores as rough, and plan a bigger test before deciding.

1. **Tuning the model until the test score peaks, then reporting that peak** — you have turned the test set into a tuning set. Fix: tune on validation data, report the test score once.
2. **Writing a report nobody else can reproduce** — no seed, no split, no settings recorded. Fix: write the data, split, seed and model settings into the report itself, not just into memory.

Practice

1. Run the code in Example 2 on your own computer. Write down both test scores and, in one line, which model you would pick and why.
2. Change `random_state` in the split to a different number and run it again. Did the winner change? Write one line about what that tells you.
3. Write your own five-line findings + limits for a project question of your choice (for example, predicting exam marks from sleep hours). Keep it honest.

1. Re-run the code in Example 5 and add a third new student: 9 hours, irregular attendance. Do both models agree on that student? Write one sentence about why (or why not).
2. Write a baseline rule for this project by hand (one sentence, no model) and work out its score on the three test rows from Example 1. Does either model beat it?
3. Pick your own project question (for example, predicting whether a friend will like a film). Write the full pipeline in six lines: question, data, split, models, comparison, limits — and name your baseline.

Practice quiz

1. What is the correct order of a small ML project?

- Train, collect, report
- Data, split, train, compare on test scores, write findings
- Report, train, split

- Compare, collect, train

Answer: Data, split, train, compare on test scores, write findings

2. How should you compare two models honestly?

- Pick the one you like
- Compare their scores on the same unseen test data
- Pick the faster one
- Pick the one with more features

Answer: Compare their scores on the same unseen test data

3. What must honest project findings include?

- Only the best score
- The findings plus the limits — what the result does and does not prove
- The full code
- A chart

Answer: The findings plus the limits — what the result does and does not prove

---

Watch the video lesson: https://youtu.be/7joiHUFpOHg

Read online: https://scopetreasure.com/notes/ml-complete-project · More free tools and notes at https://scopetreasure.com