

Your First Machine Learning Model in Python with scikit-learn

Features and labels, training and test sets, fitting a decision tree, reading the rules it learned, measuring accuracy honestly, and using models responsibly.

25 min read · Content last verified 20 October 2026 · Reviewed by Shravya M

Free awareness guide. Not legal or financial advice. <https://www.shravyadigital.com/guides/python-first-ml-model>

CONTENTS

1. Before you start
2. Chapter 1: The plan
3. Chapter 2: The data: features and label
4. Chapter 3: Training and test sets
5. Chapter 4: Training a model
6. Chapter 5: Testing: is it any good?
7. Chapter 6: Using the model
8. Chapter 7: Doing it responsibly
9. Key words

Before you start

This guide is the reading for **Course 4** of the Python for AI Pathway. It takes about 25 minutes and builds on Courses 1 to 3, and on *How AI Models Work* from the AI Foundations Pathway. Open a Google Colab notebook and run each example.

You'll train a real machine learning model in a few lines of Python, test it honestly, and learn how to tell whether it's any good. No maths is needed: the library does the maths, and your job is to give it good data and judge the results.

All names and figures here are made up.

Chapter 1: The plan

We'll build a model that predicts whether a student will **pass an exam**, from two facts about them: **hours studied** per week and **attendance** percentage.

Every supervised machine learning project follows the same steps:

1. **Collect data** with the answer included (here: did each past student pass?).
2. Split it into **features** (the inputs) and a **label** (the answer to predict).
3. Split the rows into a **training set** and a **test set**.
4. **Train** (fit) a model on the training set.
5. **Test** it on the test set, which it has never seen.
6. **Use** it to predict for new cases.

We'll use **scikit-learn**, the standard Python library for classic machine learning. Colab has it installed.

Chapter 2: The data: features and label

```
import pandas as pd

data = pd.DataFrame({
    "hours": [2, 9, 4, 7, 1, 8, 5, 3, 10, 6, 2, 7, 9, 4, 8, 1, 6, 3, 10, 5],
    "attendance": [60, 95, 70, 85, 50, 90, 80, 65, 98, 75, 55, 88, 92, 72, 85, 45, 82, 68, 96, 78],
    "passed": [0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1],
})
print(data.shape)
print(data["passed"].value_counts().to_dict())
```

Output:

```
(20, 3)
{1: 12, 0: 8}
```

`passed` is 1 for pass and 0 for fail. In machine learning terms:

- **Features** (usually called `X`): the columns the model learns from, `hours` and `attendance`.
- **Label** (usually called `y`): the column it learns to predict, `passed`.

```
X = data[["hours", "attendance"]]
y = data["passed"]
```

Twenty rows is tiny, small enough to read in one go. Real projects use thousands or millions, but every step is identical.

Chapter 3: Training and test sets

If you test a model on the same rows it learned from, it can simply memorise them and look perfect. That's **overfitting**. So we hide some rows during training:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
print(len(X_train), "rows to train,", len(X_test), "rows to test")
```

Output:

```
15 rows to train, 5 rows to test
```

- `test_size=0.25` keeps 25% of rows aside for testing.
- `random_state=42` makes the random split the same every time you run it, so results are repeatable. Any number works; 42 is a programmers' joke.

The rule: Never judge a model on the data it learned from.

Chapter 4: Training a model

We'll use a **decision tree**: a model that learns a set of yes/no questions, like "Is attendance above 75%?". Training is called **fitting**:

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(max_depth=2, random_state=0)
model.fit(X_train, y_train)
print("Trained")
```

Output:

```
Trained
```

That's it: `fit` looks at the training rows and works out the questions that best separate passes from fails. `max_depth=2` limits it to two levels of questions, which keeps it simple and less likely to overfit.

You can print the questions it learned:

```
from sklearn.tree import export_text

print(export_text(model, feature_names=["hours", "attendance"]))
```

Output:

```
|--- attendance <= 73.50
|   |--- class: 0
|--- attendance > 73.50
|   |--- class: 1
```

On this data, one question was enough: attendance above 73.5% means "pass". That's the power and the danger of a model: it finds the simplest rule that fits the examples, even if a teacher would say hours of study matter too. More, and more varied, data would teach it more.

Every scikit-learn model works the same way: create it, `fit` it, then `predict`. Swapping a decision tree for another model, such as `LogisticRegression`, changes one line.

Chapter 5: Testing: is it any good?

Now predict the test rows the model has never seen, and compare with the real answers:

```
from sklearn.metrics import accuracy_score

predictions = model.predict(X_test)
print(predictions.tolist())
print(y_test.tolist())
print("Accuracy:", accuracy_score(y_test, predictions))
```

Output:

```
[0, 0, 0, 1, 1]
[0, 0, 0, 1, 1]
Accuracy: 1.0
```

Accuracy is the share of predictions that were right: 1.0 means all of them. Be careful about reading too much into it:

- With only **5 test rows**, one lucky or unlucky row changes accuracy by 20%. Small tests are unreliable.
- Accuracy can mislead when one answer is rare. If 95% of payments are genuine, a model that always says "genuine" is 95% accurate and catches **no** fraud.
- Other measures help. **Precision** asks how many of the predicted passes really passed; **recall** asks how many of the real passes the model found.

Chapter 6: Using the model

Predict for new students:

```
new_students = pd.DataFrame({"hours": [3, 8], "attendance": [62, 91]})  
print(model.predict(new_students).tolist())  
print(model.predict_proba(new_students).round(2).tolist())
```

Output:

```
[0, 1]  
[[1.0, 0.0], [0.0, 1.0]]
```

- `predict` gives the answer: the first student is predicted to fail, the second to pass.
- `predict_proba` gives the model's **confidence** for each answer, [fail, pass].

That confidence comes from very little data here, so it's far more certain than it deserves to be. Real models are more cautious, and a good system says how sure it is.

Chapter 7: Doing it responsibly

You can now train a model in ten lines. The hard parts are about judgement:

- **Is the data fair?** If past data reflects unfair treatment, the model learns it. A model predicting who "deserves" a scholarship from old decisions can repeat old biases.
- **Is it tested properly?** Use enough test data, and check different groups separately.
- **Is a model even needed?** Sometimes a simple rule is clearer and good enough.
- **Who checks the decisions?** Predictions about people, such as exams, loans or jobs, need a human who can review and overrule them.

In Course 5 you'll use ready-made AI models, including language models, from Python.

Key words

- **scikit-learn:** the standard Python library for classic machine learning.
- **Feature / X:** an input column the model learns from.
- **Label / y:** the answer the model learns to predict.
- **Training set / test set:** rows used to learn, and rows kept back to check.
- **train_test_split:** splits data into training and test sets.
- **random_state:** fixes randomness so results repeat.
- **Decision tree:** a model that learns a series of yes/no questions.
- **fit / predict / predict_proba:** train, give answers, give confidence.
- **Accuracy:** the share of correct predictions.
- **Precision / recall:** how many predicted positives were right / how many real positives were found.
- **Overfitting:** memorising training data instead of learning general patterns.

Take the course for what you just read

Take the free 10-question assessment and pass with 8 or more. Every course you pass counts towards a Pathway Certificate issued by Shravya Digital, earned when you pass all five courses in the pathway: <https://digital.shravyaworld.org/courses/python-first-ml-model>

Sources

Checked against these primary sources on 20 October 2026. Rules change: if something here looks out of date, trust the source.

- scikit-learn: getting started (https://scikit-learn.org/stable/getting_started.html)
- scikit-learn: decision trees (<https://scikit-learn.org/stable/modules/tree.html>)
- scikit-learn: model evaluation (accuracy, precision, recall) (https://scikit-learn.org/stable/modules/model_evaluation.html)