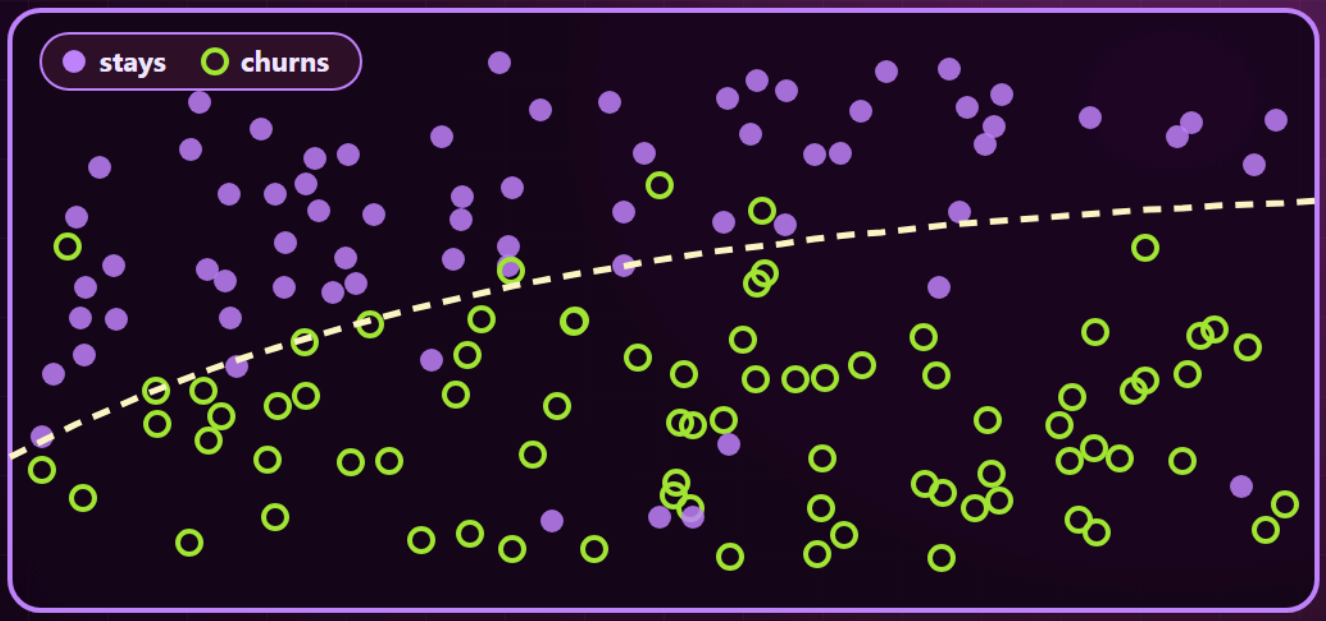
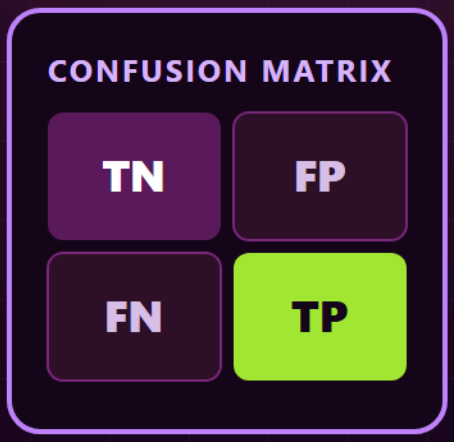


train_churn.py

```
pipe = Pipeline([
  ("prep", ColumnTransformer(...)),
  ("model", HistGradientBoosting...)
])
pipe.fit(X_train, y_train)
# random_state=42 everywhere
```



A HANDS-ON SCIKIT-LEARN GUIDE

Machine Learning with Python

Build, evaluate, explain, and deploy real models, step by step, for beginners

NumPy · pandas · Pipelines · Cross-Validation · Metrics
Random Forests · Gradient Boosting · k-Means · PCA · APIs

FOUNDATIONS · MODELS · WORKFLOWS · PRODUCTION

THENEXBOOKS

Machine Learning with Python

A Hands-On scikit-learn Guide for Beginners
From your first model to a tested, explained, deployed, and monitored prediction system — built step by step

First Edition

THENEXBOOKS

Machine Learning with Python
A Hands-On scikit-learn Guide for Beginners

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Machine learning models can be wrong; decisions that affect people should always include human judgment. Python, scikit-learn, pandas, NumPy, and the other tools discussed in this book evolve; always consult the official documentation for the versions you use.

Python is a trademark of the Python Software Foundation. This book is independent and is not affiliated with, sponsored by, or endorsed by the Python Software Foundation, the scikit-learn project, the pandas project, the NumPy project, NumFOCUS, or any other project or company mentioned. Microsoft, Windows, and Visual Studio Code are trademarks of the Microsoft group of companies. macOS is a trademark of Apple Inc. Linux is a registered trademark of Linus Torvalds. PyCharm is a trademark of JetBrains s.r.o. Jupyter is a trademark of LF Charities. Docker is a trademark of Docker, Inc. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement.

Streamnest, the video-streaming subscription service used as the running example in this book, is fictional, as are its employees, customers, plans, prices, and every row of its data. The datasets are generated by a seeded Python script included with the book; any patterns in them were designed for teaching and say nothing about real people, regions, or age groups.

Preface

Machine learning has a reputation for being hard: a field full of equations, jargon, and mysterious models that only specialists understand. The everyday reality is friendlier. Most useful machine learning is about asking a clear question, preparing data carefully, fitting a well-understood model, and checking honestly whether it works. If you can write a little Python, you can learn to do that well.

This book teaches machine learning the practical way, with **scikit-learn**, a widely used, well-documented Python library for classic machine learning. We do not start with theory and hope you can apply it later. We start with a real question from a real-looking business, write code that answers it, and explain each idea at the moment it solves a problem we actually have. Every example in the book was run, and every number you see was printed by the code next to it.

Just as important, this book teaches you the habits that separate models that work from models that only look good: baselines, proper train/test splits, pipelines, cross-validation, the right metrics, leakage checks, explanations, fairness checks, and monitoring after deployment. Those habits matter far more than knowing a long list of algorithms.

Who This Book Is For

- **Python beginners who want to learn machine learning** and know the basics: variables, lists, dictionaries, functions, loops, and running a script.
- **Analysts and spreadsheet power users** who want to move from describing data to predicting with it.
- **Developers** who need to add a prediction feature to an application and want to understand what they are shipping.
- **Students and career changers** who want a solid, honest foundation before moving on to deep learning or advanced statistics.

You do not need prior knowledge of statistics, linear algebra, or calculus. When a formula helps, we show one short formula and then explain it in words.

What You Will Learn

- What machine learning is and is not, how supervised and unsupervised learning differ, and when a simple rule is the better answer.
- The NumPy and pandas essentials you need to load, explore, and clean data.
- The machine learning workflow: framing a problem, splitting data, building baselines, evaluating, and iterating.
- Core algorithms: linear and logistic regression, k-nearest neighbors, support vector machines, decision trees, random forests, and gradient boosting with HistGradientBoosting.

- Reliable workflows: preprocessing, Pipelines and ColumnTransformer, cross-validation, and the right metrics for classification and regression.
- Better models: imbalanced classes and decision thresholds, hyperparameter tuning, feature engineering, and how to find and prevent data leakage.
- Unsupervised learning with k-means clustering and PCA, model interpretation with permutation importance, and responsible machine learning practices.
- Production basics: saving and versioning models, serving predictions behind a small API, monitoring for drift, and a complete end-to-end capstone.

How This Book Is Organized

Part I — Foundations (Chapters 1–5) explains what machine learning is, sets up your tools, covers NumPy and pandas essentials, and walks through the workflow with splits and baselines. **Part II — Core Algorithms** (Chapters 6–10) covers linear regression, logistic regression, k-nearest neighbors and support vector machines, decision trees, and random forests. **Part III — Reliable Workflows** (Chapters 11–15) covers preprocessing, pipelines, cross-validation, and evaluation metrics. **Part IV — Better Models** (Chapters 16–20) covers gradient boosting, imbalanced classes and thresholds, hyperparameter tuning, feature engineering, and data leakage. **Part V — Unsupervised Learning and Trust** (Chapters 21–24) covers clustering, PCA, model interpretation, and responsible machine learning. **Part VI — Production and Capstone** (Chapters 25–28) covers saving models, serving predictions, monitoring, and a complete end-to-end project with a look at what to learn next. The appendices provide a cheat sheet, templates and reference tables, a production checklist, and a glossary.

Each chapter includes exercises under **Try It Yourself** and ends with a **Key Takeaways** box for quick review.

The Running Example: Streamnest

Throughout the book we work for **Streamnest**, a fictional video-streaming subscription service. Its retention team, led by Ava, wants to know which subscribers are likely to cancel next month so it can send them a well-timed offer. Its content team wants to forecast how many hours each subscriber will watch. And its marketing team wants to understand the different kinds of viewers it has. Those three questions give us a classification problem (churn), a regression problem (watch hours), and a clustering problem (viewer segments).

The data is a set of 4,000 subscribers with realistic features — plan, billing cycle, tenure, viewing habits, support tickets, payment failures, survey scores — complete with missing values and an imbalanced target, plus a newer batch of 1,000 subscribers whose behavior has shifted. A small, seeded Python script generates the data, so your numbers match the book exactly. By the end, the churn model is tuned, explained, checked for fairness, saved with its metadata, served behind an API, and monitored for drift.

Versions and Conventions

Examples were tested with **Python 3.12**, **scikit-learn 1.9** (1.9.1), **pandas 3.0**, and **NumPy 2.5**. Most code also runs on recent earlier versions; where a feature is new or has changed recently, we say so. Every random process uses a fixed `random_state` (usually 42), so you can reproduce every number. The book contains no plots: we describe charts in words and print results as text tables, so you need nothing beyond the three core libraries. Where an optional tool such as FastAPI or skops appears, we label it clearly.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause wrong results, data loss, or security problems.

Code appears in `monospace`. The label above each code block names the file, such as `ch07_logistic.py`, or says **Terminal** or **Output**. Commands are shown as `python script.py`; on Windows you can also type `py script.py`. Long lines are wrapped to fit the page.

Contents

Preface

PART I — FOUNDATIONS

1 What Machine Learning Is (and Isn't)

2 Setting Up Python and scikit-learn: Your First Model

3 NumPy Essentials for Machine Learning

4 pandas Essentials: Loading, Exploring, and Cleaning Data

5 The Machine Learning Workflow: Framing, Splitting, and Baselines

PART II — CORE ALGORITHMS

6 Linear Regression: Forecasting Watch Hours

7 Logistic Regression: Predicting Churn

8 k-Nearest Neighbors and Support Vector Machines: Distance, Scaling, and Margins

9 Decision Trees: Learning Rules from Data

10 Random Forests: The Wisdom of Many Trees

PART III — RELIABLE WORKFLOWS

11 Preprocessing: Missing Values, Scaling, and Categorical Features

12 Pipelines and ColumnTransformer: From Raw Data to Prediction in One Object

13 Cross-Validation: Trustworthy Performance Estimates

14 Classification Metrics: Accuracy, Precision, Recall, F1, and ROC AUC

15 Regression Metrics and Residual Analysis

PART IV — BETTER MODELS

16 Gradient Boosting with HistGradientBoosting

17 Imbalanced Classes and Decision Thresholds

18 Hyperparameter Tuning: Grid, Random, and Successive Halving Search

19 Feature Engineering That Actually Helps

20 Data Leakage: How Good-Looking Models Lie

PART V — UNSUPERVISED LEARNING AND TRUST

21 Clustering with k-Means: Segmenting Streamnest Viewers

22 Dimensionality Reduction with PCA

23 Interpreting Models: Coefficients, Importances, and Permutation Importance

24 Responsible Machine Learning: Fairness, Bias, and Documentation

PART VI — PRODUCTION AND CAPSTONE

25 Saving, Loading, and Versioning Models

26 Serving Predictions: A Small Prediction API

27 Monitoring Models in Production: Drift and Decay

28 Capstone: The Streamnest Churn System End to End — and What to Learn Next

APPENDICES

A scikit-learn Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Foundations

What machine learning really is and when you need it, a working Python setup with your first model, the NumPy and pandas skills every project relies on, and the workflow of framing a question, splitting data, and beating an honest baseline.

1 · What Machine Learning Is (and Isn't)
2 · Setting Up and Your First Model
3 · NumPy Essentials
4 · pandas Essentials
5 · The Workflow: Splits and Baselines

CHAPTER 1 · PART I — FOUNDATIONS

What Machine Learning Is (and Isn't)

Before we install anything or write a model, we need a clear picture of what machine learning actually does, what it cannot do, and when you should not use it at all. This chapter gives you that picture and introduces Streamnest, the fictional streaming service whose questions we answer for the rest of the book.

A Question Worth Answering

Imagine you work at a video-streaming company. Every month, some subscribers cancel. Each cancellation costs the company a monthly fee, and winning a customer back later is usually harder than keeping one. The retention team has a modest budget for offers, such as a month at half price, and they want to send those offers to the people most likely to leave. Sending an offer to everyone wastes money on people who were going to stay anyway. Sending it to nobody means watching customers walk away.

So the team asks a simple-sounding question: *which of our subscribers are likely to cancel next month?*

There are two broad ways to answer it. The first is to sit down with people who know the business, agree on some warning signs, and write them down as rules. The second is to collect examples of past subscribers, including whether each one actually cancelled, and let a computer program find the patterns that separate the people who left from the people who stayed. The second approach is machine learning.

That is the core idea of the whole field, and it is worth stating plainly: **machine learning is a way of building a program by showing it examples instead of writing its rules by hand.** The rules still exist in the finished program, but a learning algorithm wrote them, guided by data, rather than a person typing `if` statements.

Rules Written by Hand vs Rules Learned from Data

To see the difference, let us start with the hand-written approach. Suppose the retention team believes that a customer who has not watched anything for more than ten days is drifting away. That belief fits in one line of Python. We will not set up Python until Chapter 2, so you do not need to run this yet; just read it.

ch01_rule.py

```
import pandas as pd

df = pd.read_csv("data/streamnest_customers.csv")
```

```
def at_risk(customer):
    """A hand-written churn rule: someone who stopped watching is at risk."""
    return customer["days_since_last_watch"] > 10


flagged = df[df.apply(at_risk, axis=1)]
caught = int(flagged["churned"].sum())
all_churners = int(df["churned"].sum())

print(f"customers:            {len(df)}")
print(f"flagged by the rule:  {len(flagged)}")
print(f"flagged who churned:  {caught} ({caught / len(flagged):.1%})")
print(f"churners caught:      {caught} of {all_churners} ({caught / all_churners:.1%})")
```

The script reads a table of 4,000 past customers, applies the rule to each one, and then checks the rule against what really happened. Here is what it prints:

Output

```
customers:            4000
flagged by the rule:  252
flagged who churned:  147 (58.3%)
churners caught:      147 of 653 (22.5%)
```

This tiny rule is not bad. Of the 252 customers it flags, 147 really did cancel, which is far better than picking customers at random. But it only catches 147 of the 653 people who cancelled, about one in five. The other 506 churners looked fine by this rule: they had watched something recently, so the rule waved them through.

How would you improve it? You might add conditions for monthly billing, payment failures, support tickets, and short tenure. Soon you are juggling a dozen thresholds picked by intuition, and you have to decide how they combine. Is a long-time annual subscriber with one payment failure riskier than a new monthly subscriber who watches a lot? Every answer is a guess that has to be maintained when the business changes.

Machine learning turns that guessing game around. You give an algorithm many past examples where the outcome is known. It searches for the combination of signals that best separates the churners from the stayers and produces a **model**: a program that takes a customer's details and returns a prediction. In Chapter 2 you will train your first model on exactly this data.

> **NOTE**
>
> A **model** in machine learning is just the learned program: a set of numbers, rules, or trees that maps inputs to an output. It is not a physical model or a simulation of the world. When people say they "trained a model," they mean they ran a learning algorithm on example data and kept the result.

The hand-written rule is not wasted effort: it gives us something to compare against. Always ask "better than what?" A model that cannot beat a one-line rule is not worth deploying. We build these comparison points, called **baselines**, in Chapter 5.

The Vocabulary You Need

Machine learning has a lot of jargon, but a handful of words carry most of the weight. Learn these now and the rest of the book will read smoothly.

- **Example** (also *row*, *sample*, *observation*, *instance*): one thing we know about. At Streamnest, one example is one customer as they looked on a particular date.
- **Feature** (also *input*, *variable*, *predictor*, *column*): one piece of information about an example that the model can use, such as how many months the customer has subscribed or how many hours they watched last month.
- **Label** (also *target*, *outcome*, *answer*, *y*): the thing we want to predict. For the churn question, the label is whether the customer cancelled: 1 for yes, 0 for no.
- **Training** (also *fitting*): running a learning algorithm on examples whose labels are known, so it can build a model.
- **Inference** (also *prediction*, *scoring*): using a trained model on new examples whose labels are not yet known.

By convention, the table of features is called `X` (a capital letter, because it is a table with many columns), and the column of labels is called `y` (lowercase, because it is a single column). You will see `X` and `y` in almost every code example from Chapter 2 onward. scikit-learn, the library we use, expects exactly this shape: a table of features and a matching list of labels, one per row.

Training and inference happen at different times. Training uses historical data where we already know who cancelled; inference happens later, on current customers whose future is unknown. If a training feature contains information that would not be available at inference time, the model learns a trick it cannot use in real life. That problem, called **leakage**, gets its own chapter (Chapter 20).

The Main Kinds of Machine Learning

Machine learning tasks fall into a few families. You do not need to memorize a taxonomy, but knowing which family your question belongs to tells you which tools to reach for.

Supervised learning

In **supervised learning**, every training example comes with the correct answer, the label. The algorithm is "supervised" in the sense that it can check its guesses against the known answers while it learns. Supervised learning comes in two main flavors, depending on what kind of answer you want.

- **Classification** predicts a category. Will this customer cancel: yes or no? Is this email spam or not? Which of three plans will a new subscriber pick? When there are exactly two categories, we call it *binary classification*; churn prediction is binary classification.
- **Regression** predicts a number. How many hours will this customer watch next month? What will a house sell for? How many support tickets will arrive tomorrow? The word "regression" is historical and slightly confusing; in machine learning it simply means "predicting a quantity."

Many classifiers can also give you a probability rather than just a yes-or-no answer: "this customer has a 72% chance of cancelling." Those probabilities turn out to be more useful than the bare yes-or-no, because they let the business rank customers and decide how many to contact. We will lean on them heavily from Chapter 7 onward.

Unsupervised learning

In **unsupervised learning**, there are no labels. The algorithm looks for structure in the features alone. The two most common tasks are:

- **Clustering**: grouping examples that are similar to each other. Streamnest's marketing team might want to know whether its viewers fall into natural groups, such as heavy binge watchers, families who mostly stream kids' shows, and casual viewers who drop in now and then. Nobody labeled those groups in advance; the algorithm proposes them.
- **Dimensionality reduction**: summarizing many features with a few new ones that keep most of the information. It helps with visualization, compression, and sometimes with modeling.

Unsupervised results are harder to judge, because there is no answer key: a clustering algorithm always returns clusters, whether or not they mean anything (Chapters 21 and 22).

You will also hear about *reinforcement learning* (learning by trial and error from rewards) and *self-supervised learning* (a model creating its own labels from raw data). Both are outside the scope of this book, which works with tables of data.

Family	Has labels?	Answers questions like	Streamnest task
Classification	Yes	Which category?	Will this customer churn?
Regression	Yes	How much? How many?	Hours watched next month
Clustering	No	Which items belong together?	Viewer segments
Dimensionality reduction	No	Can we summarize many columns?	Compact viewer profiles

What Machine Learning Is Not

Machine learning gets described in grand terms, and it is easy to come away with the wrong expectations. Here are the misunderstandings that cause the most trouble in practice.

It is not magic

A model can only find patterns that exist in the data you give it. If the features carry no information about the label, no algorithm can conjure a good prediction. Machine learning is closer to careful, automated pattern-counting than to understanding: it weighs thousands of examples consistently, but it has no common sense to fall back on when the data is misleading.

It does not find causes

This is the most important limitation in the chapter. A churn model learns which customers *tend* to cancel. It does not learn *why* they cancel, and it cannot tell you what would happen if you changed something. Suppose the model notices that customers with many support tickets churn more often. That does not mean support tickets cause churn; it might be that a broken app causes both the tickets and the cancellations. Closing tickets faster might help, or it might not. Answering "what happens if we act" requires experiments, such as sending an offer to a random half of at-risk customers and comparing the two halves, or careful causal analysis. Predictions are still valuable, because knowing *who* is at risk lets you target an action, but keep "predicts" and "causes" firmly apart in your mind and in your reports.

> **WARNING**
>
> Never present a model's patterns as causes. "Customers on monthly billing are predicted to churn more" is a fair statement. "Monthly billing makes customers churn, so we should force everyone onto annual plans" is a business decision that the model cannot justify on its own. Explanations of models (Chapter 23) describe what the model uses, not how the world works.

It is not a substitute for a clear goal

"Use machine learning on our data" is not a goal. A useful project starts with a decision someone will make differently because of the prediction. At Streamnest, the retention team decides each month which customers get an offer, and that decision shapes what we predict, when, and how we judge success. Chapter 5 turns this into a framing checklist.

It is not a fix for bad data

A model trained on messy, wrong, or unrepresentative data learns the mess. If one region's data was exported with a bug, the model treats the bug as a fact about that region. Expect to spend more time understanding data than choosing algorithms.

It is not always needed

Sometimes a simple rule, a database query, or a spreadsheet formula is the right answer. If the business already knows that every customer whose card failed twice will be cancelled automatically by the billing system, you do not need a model to predict those cancellations: you need a query that lists

them. If a rule is easy to explain, good enough, and stable, it may beat a model on cost and trust. Ask these questions before reaching for machine learning:

- Is there a decision that will change based on the prediction?
- Do we have enough past examples with known outcomes, recorded the same way we will record new ones?
- Is the pattern too complex for a few hand-written rules, or does it shift over time in ways rules cannot keep up with?
- Can we tolerate some wrong predictions, and do we know what each kind of mistake costs?
- Can we measure whether the model is helping after it goes live?

If the answer to most of these is yes, machine learning is a good fit. If not, start with the simple approach, measure it, and revisit later.

> **TIP**
> Even when you build a model, keep the simple rule around as a baseline and a fallback if the model breaks.

Machine Learning, Statistics, AI, and Deep Learning

These words overlap, and people use them loosely. Here is a plain-language map that is good enough for this book.

- **Artificial intelligence (AI)** is the broadest term: any technique that makes computers do tasks we associate with intelligence. It includes machine learning, but also hand-built expert systems, search and planning algorithms, and more. Today the word is often used to mean large language models specifically, which can be confusing.
- **Machine learning** is the part of AI where programs learn from data rather than being fully programmed by hand. Everything in this book is machine learning.
- **Deep learning** is a part of machine learning that uses large neural networks with many layers. It excels at images, audio, and text, where the raw input is huge and unstructured. For tables of customer records like ours, well-tuned classic methods such as gradient-boosted trees are often as good or better, cheaper to train, and easier to explain. We give an overview of deep learning and where to go next in Chapter 28.
- **Statistics** is the older discipline of learning from data, focused on uncertainty and explaining relationships; linear and logistic regression came from it. Roughly, statistics asks "what is the relationship, and how sure are we?" while machine learning asks "how accurately can we predict new cases?"

The tool we use, **scikit-learn**, is a free, open-source Python library for classic machine learning: regression, classification, clustering, preprocessing, and model evaluation. It does not do deep learning, and it does not need a graphics card. It runs comfortably on a laptop, it is widely used and

well documented, and its consistent design means that once you learn one model, you know how to use nearly all of them. Alongside it we use **NumPy** for arrays of numbers (Chapter 3) and **pandas** for tables (Chapter 4).

Where Machine Learning Goes Wrong

Most machine learning failures are not caused by choosing the wrong algorithm. They come from a small set of recurring problems. We introduce them here so you recognize them when they appear later; each has a chapter or section of its own.

- **Bad labels.** If the target is defined vaguely or recorded inconsistently, the model learns the wrong thing very precisely. Is a customer who pauses their subscription "churned"? What about one who cancels and resubscribes the next day? Decide, write it down, and apply it consistently.
- **Leakage.** A feature that secretly contains the answer makes a model look brilliant in testing and useless in production. A classic example is a column filled in *after* the outcome happened, such as a cancellation reason. Chapter 20 shows a leaky Streamnest model scoring almost perfectly and explains why that is an alarm, not a success.
- **Evaluating on the training data.** A model that memorizes its examples can score perfectly on them and still fail on new customers. We always hold back a test set the model never sees during training, starting in Chapter 2.
- **Drift.** The world changes. A model trained on last year's customers may slowly get worse as new marketing campaigns bring in different kinds of subscribers. Chapter 27 measures this on a newer batch of Streamnest customers whose behavior has shifted.
- **Bias and unfairness.** Models can treat groups of people differently, sometimes because of who was included in the data, sometimes because of features that act as stand-ins for sensitive attributes. Chapter 24 checks the churn model across age groups and discusses what to do about differences.
- **Misleading metrics.** A churn model that predicts "nobody cancels" is right about 84% of the time on our data, because most customers stay. That high accuracy is worthless. Chapter 5 and Chapter 14 teach you which numbers to trust.

None of these problems is solved by a cleverer algorithm; they are solved by careful framing, honest evaluation, and attention to data.

Meet Streamnest

Throughout the book we work for **Streamnest**, a fictional video-streaming subscription service with customers in four regions: North America, Europe, Asia-Pacific, and Latin America. Streamnest offers three plans (basic, standard, and premium) with monthly or annual billing, where annual billing gives a 15% discount. Customers watch on TVs, phones, laptops, and tablets.

Everything about Streamnest is invented. The customers are synthetic, generated by a small seeded Python script called `make_data.py` that you will run in Chapter 2. Because the script is seeded, the data on your computer will match the data in this book row for row, so every number you see printed here should appear on your screen too. The data was designed to behave like a realistic subscription business, with missing values, uneven groups, and a minority of churners, but no pattern in it is a finding about real people, real regions, or real age groups.

The people

Three colleagues appear from time to time, usually to ask the questions that real projects raise:

- **Ava** is Head of Retention. She owns the offer budget and asks the business questions: How many churners will we catch? How many offers will we waste?
- **Tomás** is a data engineer. He owns the data exports and production systems, and asks the practical questions: What do you need from me each month? What happens when a field is missing?
- **Jun** is a junior analyst learning machine learning alongside you, and occasionally makes the mistakes we all make, so we can fix them together.

The three tasks

Streamnest gives us one problem from each of the main families we just met:

1. **Churn prediction (classification).** Using what we know about each customer at the end of March 2026, predict whether they cancel during April 2026. This is the main thread of the book, ending with a tuned, saved, served, and monitored model.
2. **Watch-hours forecasting (regression).** Predict how many hours each customer will watch next month, for planning (Chapters 6 and 15).
3. **Viewer segmentation (clustering).** Without labels, group customers by how they watch, so marketing can tailor messages (Chapter 21).

The data

The main dataset, `data/streamnest_customers.csv`, has 4,000 customers. Each row describes one customer as of a **snapshot date**, March 31, 2026. The features describe the customer on that date and in the months before it, and the two labels describe what happened afterward, during April 2026. Customer IDs run from `SN-100001` to `SN-104000`. A second file, `data/streamnest_scoring_2026_06.csv`, holds 1,000 newer customers from a later snapshot (May 31, 2026); we save it for the production chapters, where it plays the role of "new customers the model has never seen," and where it turns out the world has shifted a little.

Here are all 24 columns, in the order they appear in the file. You do not need to memorize them; come back to this table whenever you need it. Appendix B has a copy.

Column	Meaning
`customer_id`	Identifier such as "SN-100001" (never a feature)
`snapshot_date`	Date the features were measured (never a feature)
`tenure_months`	Months subscribed, 1 to 96
`plan`	basic, standard, or premium
`billing_cycle`	monthly or annual
`monthly_fee`	7.99, 12.99, or 17.99 USD; annual billing is 15% off (6.79, 11.04, 15.29)
`region`	north_america, europe, asia_pacific, or latin_america (77 missing)
`age`	Age in years, 18 to 80 (192 missing; not asked at sign-up)
`primary_device`	tv, mobile, laptop, or tablet
`acquisition_channel`	organic, social, partner, or tv_ad
`profiles`	Viewer profiles on the account, 1 to 5
`watch_hours_last_month`	Hours watched in March 2026
`watch_hours_3m_avg`	Average monthly hours, January to March 2026
`sessions_last_30d`	Viewing sessions in the last 30 days
`days_since_last_watch`	Days since the last viewing, 0 to 60
`kids_share`	Share of hours spent on kids' content, 0 to 1
`binge_share`	Share of hours in sessions of two hours or more, 0 to 1
`weekend_share`	Share of hours watched on weekends, 0 to 1
`support_tickets_90d`	Support tickets in the last 90 days
`payment_failures_6m`	Failed payments in the last six months
`promo_active`	1 if the customer is on a promotional discount
`satisfaction_score`	Last survey answer, 1 to 5 (1,410 missing; survey not answered)
`churned`	**Label 1:** 1 if the customer cancelled during April 2026
`watch_hours_next_month`	**Label 2:** hours watched in April 2026

A few details will matter later. **Two columns are never features:** the ID is just a name tag, and the snapshot date marks *when* a row was measured rather than describing the customer. **Values are missing for different reasons:** region for 77 customers (probably an export glitch), age for 192 (not asked at sign-up), and satisfaction for 1,410 (survey not answered, a silence that may itself carry a signal). Chapter 4 counts these gaps and Chapter 11 handles them inside a model.

The labels describe the future relative to the snapshot. Features describe March and earlier; both labels describe April. So when we predict churn, `watch_hours_next_month` must never be a feature: on March 31 nobody knows it, and it is almost a giveaway, since people who cancel stop watching (Chapter 20). **The classes are imbalanced:** only 653 of the 4,000 customers churned, about 16.3%, which is why "everyone stays" scores about 84% accuracy.

To make the rows concrete: customer `SN-100001` has subscribed for 80 months on the standard plan with annual billing, averaged 54.1 hours a month, and stayed. Customer `SN-100003` has 26 months on standard with monthly billing, a promotional discount, only 9.3 hours last month, one support ticket, one failed payment, and cancelled in April. You can probably tell a story about why the second one left. The model's job is to tell that kind of story consistently, for thousands of customers at once.

> **NOTE**
> Real data is rarely this tidy: expect duplicate rows, inconsistent spellings, and labels that arrive late. The habits you build here (checking counts, asking what each column means and when each value is known) protect you on messy data.

How a Machine Learning Project Flows

Every chapter in this book fits somewhere in a loop that looks roughly like this:

```
frame the question --> get and understand data --> split --> baseline
        ^                                                      |
        |                                                      v
     monitor <-- deploy <-- evaluate honestly <-- train models, iterate
```

Chapter 5 walks through the whole loop carefully; for now, just notice that training a model is one step out of many.

The Plan for This Book

Part I (Chapters 1–5) sets up your tools, data skills, and workflow. Part II (Chapters 6–10) covers the core algorithms. Part III (Chapters 11–15) makes the workflow reliable with preprocessing, pipelines, cross-validation, and metrics. Part IV (Chapters 16–20) builds better models and guards against leakage. Part V (Chapters 21–24) covers unsupervised learning, interpretation, and responsible practice. Part VI (Chapters 25–28) takes the churn model into production. Each chapter has a script named after it, such as `ch02_first_model.py` or `ch07_logistic.py`, inside one project folder called `streamnest-ml/`. You will build that folder step by step, starting in the next chapter. From Chapter 5 onward, a small shared module called `streamnest.py` collects the code we reuse, such as loading the data and splitting it the same way every time.

Because every random step uses a fixed seed (usually `random_state=42`), your results should match the printed outputs when you use the same library versions; Chapter 2 shows you how to check.

Read the chapters in order if you are new to machine learning, and type the examples yourself; small mistakes and their error messages teach a lot. If you already know some Python data work, you can skim Chapters 3 and 4, but do not skip Chapter 5.

> **TIP**
>
> If you use [your AI assistant] while learning, ask it to explain an error message or an idea in different words, and then check its answer against the code and output in front of you. Treat its suggestions like advice from a helpful colleague who has not seen your data: often useful, sometimes confidently wrong.

Try It Yourself

1. Write down three signals, other than days since last watch, that you would expect to predict churn at a streaming service. For each, write one sentence explaining why, and one sentence describing how it could mislead you.
2. For each question, decide whether it is classification, regression, clustering, or none of these: (a) Will this payment fail? (b) How many minutes will this video be watched in its first week? (c) What kinds of shows tend to be watched together? (d) Which customers signed up yesterday?
3. The rule in this chapter flagged 252 customers and 147 of them churned. Explain in your own words why "58.3% of flagged customers churned" and "22.5% of churners were caught" are both important, and which one Ava would care about more if offers were very cheap.
4. Pick one column from the data dictionary that you think could be a *proxy* for something sensitive (for example, a feature that indirectly reflects age or location). Write down why that might matter for fairness. We return to this in Chapter 24.
5. Think of a problem at your own work or in daily life where a simple rule or query would beat machine learning. What makes the rule good enough?