Introduction to Data Science: Draft Textbook

Dr Benjamin Pope and Greg Baker

November 10, 2025

Contents

1 How to Use This Book **1**

1.1 How the chapters fit together 2

1.2 Making the most of hands-on activities 2

1.3 Building a study rhythm 3

1.4 Preparing for assessments and future topics . . 4

2 Data Science Roles **5**

2.1 Mapping the Talent Pipeline 6

2.2 Owning the Model Lifecycle 10

2.3 Future-Facing Opportunities 14

3 Probability Foundations **19**

3.1 Probability Interpretations 20

3.2 Connect Probability to Inference 23

3.3 Distributions and Samples 26

3.4 Demystify Simulation Basics 28

3.5 Highlight the Central Limit Theorem 31

4 Retail Analytics Case Study **35**

4.1 Business Context and Data Assets 36

4.2 Feature Engineering 40

4.3 Clustering Workflow 45

5 Hands-on Activity 1 **55**
5.1 Prepare the Materials 55
5.2 Cluster Tokens by Hand 56
5.3 Recreate the Workflow in Orange 58
5.4 Explore a Real Dataset 59
5.5 Plan Your Next Steps 59

6 Exploratory Analysis **61**
6.1 Stage Data for Orange Workflows 62
6.2 Interactive Visuals 67
6.3 Link to Broader Workflow 71

7 Robust Regression in Orange **77**
7.1 Recognise When Baseline Lines Fail 77
7.2 Resistant Estimators 81
7.3 Test Insights with Real Housing Data 84
7.4 Document Outcomes 88

8 Hands-on Activity 2 **91**
8.1 Kit Checklist 91
8.2 Welcome and Setup 93
8.3 Fit a Line with the Theil–Sen Estimator 93
8.4 Fit a Line with RANSAC 95
8.5 Recreate the Experiment in Orange 96
8.6 Wrap Up 97

9 Logistic Regression **99**
9.1 Warn Against Overfitting 99
9.2 Ridge and Lasso 103
9.3 Regression Metrics 105
9.4 Model Log-Odds with Orange 108
9.5 Cross-Entropy and Descent 111

9.6 Teach Classification Diagnostics 114

10 Hands-on Activity 3 **119**
10.1 Kit Checklist 119
10.2 Build Chessboard Dataset 121
10.3 Stage 2: Initialise a Logistic Model 122
10.4 Stage 3: Calculate Likelihood by Hand 122
10.5 Stage 4: Greedy Coordinate Search 123
10.6 Stage 5: Evaluate the Classifier 124
10.7 Stage 6: Explore Variations 124
10.8 Rebuild in Orange 125

11 Understand and Tune k-NN **127**
11.1 k-NN Voting Intuition 127
11.2 Operational Trade-offs 133

12 NSW Land Value **137**
12.1 Understand the Valuation Data 137
12.2 Interrogate the Prediction 142

13 Red Rooster Line **147**
13.1 Frame the Geospatial Challenge 147
13.2 Interpret the Results 152

14 Model Evaluation **155**
14.1 Cross-Validation 156
14.2 Avoid Forking Paths 159
14.3 Metrics and Baselines 162

15 Hands-on Activity 5 **169**
15.1 Kit Checklist 169
15.2 Learning Goals 171
15.3 Classification with Tokens 172

15.3.1 Setup (5 minutes) 172
15.3.2 Logistic Regression by Ruler (10 minutes) 172
15.3.3 k-NN on the Same Validation Set (15 minutes) 173
15.3.4 Single Test Pass (5 minutes) 173
15.3.5 Metric Reference 174
15.4 Part B: k-NN Regression 174
15.4.1 Setup (5 minutes) 174
15.4.2 k-NN Regression (15 minutes) 175
15.5 Part C: Orange Workflow 176
15.5.1 Painted Classification (required, ∼25 minutes) 176
15.5.2 Regression in Orange (optional, 10–15 minutes) 176

16 Explainable Models **179**
16.1 Explanations or Accuracy 180
16.2 Engineer Explainable Features 183
16.3 Induce Transparent Rules with CN2 186
16.4 Explain Decisions with Trees 189

17 Metrics and Ensembles **195**
17.1 Read ROC Curves as Ranking Tools 195
17.2 Setting Thresholds 198
17.3 Stack Diverse Learners 201
17.4 Random Forests 204

18 Hands-on Activity 6 **209**
18.1 Kit Checklist 210
18.2 Learning Goals 211
18.3 Part A: Decision Trees with Paddle-Pop Sticks 211
18.3.1 Lay out the dataset 211

18.3.2 Score candidate splits 212
18.3.3 Extend the tree 213
18.4 Part B: CN2 Rule Induction Game 213
18.4.1 Key ideas recap 213
18.4.2 Weighted Relative Accuracy 214
18.4.3 Play one beam search per class 214
18.5 Part C: Orange Workflow 215
18.6 Watchband Dataset 216

19 NSW Road-Crash Data **219**
19.1 Import Crash Workbook 220
19.2 Series vs DataFrames 224
19.3 Engineer ratios 228
19.4 Bridge to the Week 8 practical 233

20 Matplotlib Foundations **235**
20.1 Adopt the figure–axes workflow 235
20.2 Scatter Plots 240
20.3 Resize and export figures 243
20.4 Leverage pandas plotting shortcuts 245
20.5 Connect to Tableau 249

21 Pipelines in scikit-learn **253**
21.1 Design column-aware pipelines 254
21.2 Select imputation strategies 257
21.3 Bundle preprocessing 263
21.4 Where to next 267

22 Text Model Tuning **269**
22.1 Structured hyperparameter search 269
22.2 Vectorise text data 272
22.3 Week 11 coordination 276

23 Recommendation Engines **279**

23.1 Implicit Feedback 280

23.2 Layered Pipelines 284

23.3 Evaluate and Monitor 289

23.4 Bridge to the Week 12 practical 292

Chapter 1

How to Use This Book

Plain-language summary

This opening chapter is a friendly orientation to the rest of the textbook. We explain how the theory chapters, hands-on activities, and supporting assets fit together so you know where to start, what to skim, and how to revisit material when assessments arrive. You will leave with a study rhythm that links the lecture-inspired explanations to the tactile workshops and digital workflows that follow.

Chapter overview

The textbook balances narrative explanations with do-it-yourself activities. This guide shows how the pieces connect: we map the chapter sequence, highlight the companion Orange workflows and datasets, and suggest ways to blend the reading with class time. We also point to support options so you can pace your learning with confidence. Keep this chapter bookmarked—you can return whenever you need a reminder

of how the book is organised or how the activities reinforce your skills.

1.1 How the chapters fit together

The book is arranged in small arcs that mirror the semester. We begin with roles and probability foundations in chapter 2 and chapter 3, then shift into clustering, regression, classification, and explainability chapters that line up with each practical. Every theory chapter opens with a plain-language summary, unpacks concepts with worked examples, and closes by pointing to its matching hands-on activity. You can read sequentially for the full story or jump directly to a chapter when a project calls for a specific technique. Each chapter stands alone but uses consistent vocabulary, so terminology carries smoothly from one context to the next.

The hands-on activities—starting with chapter 5—live alongside the theory rather than in a separate appendix. They invite you to manipulate physical kits, interpret Orange workflows, and connect those experiences back to the models described earlier in the week. When you see a reference to an asset such as a dataset, workflow, or figure, the filename appears in `asset-checklist.md`. Use that file as your packing list whenever you prepare for a lab or wish to recreate the demonstrations at home.

1.2 Making the most of hands-on activities

Each activity is designed to be tactile first, digital second. Begin by setting up the physical kit, talking through the

scenario with your peers, and writing down observations as you go. Only then move into Orange to rebuild the workflow. This order keeps the intuition alive when you translate the logic into software. The activity chapters call out common pitfalls, include metric definitions, and suggest optional extensions if you finish early or want to explore further. Treat them as a menu: complete the core steps, then pick the stretch ideas that reinforce the week's lecture focus.

When a chapter references prior knowledge, skim back to the earlier theory section or follow the cited 'For Advanced Students' callout. These sidebars extend the activity without derailing newcomers. If you need a refresher on terminology while running an activity, keep the related theory chapter open; the layout of headings, learning objectives, and summaries mirrors the activity structure so you can line concepts up quickly.

1.3 Building a study rhythm

A steady routine helps the content stick. Early in the week, read the upcoming theory chapter while noting questions or definitions to raise in class. During the lecture or tutorial, compare your notes with the explanations provided there. Soon after, work through the matching hands-on activity. Capture photos of your setups, record parameter choices, and save Orange workflows in case you need to revisit them for assessments. End the week with a short reflection: what evidence convinced you that a model worked, and what would you test next?

Lean on the support network when topics feel heavy. Super tutors, discussion forums, and drop-in help sessions exist

so you can ask for guidance before confusion builds. Share your notebook pages or workflow screenshots when seeking help—concrete artifacts make it easier for staff and peers to suggest the next step. If you miss a class, revisit the chapter summaries and activity checklists before jumping back in; they flag the essential moves so you can catch up without guesswork.

1.4 Preparing for assessments and future topics

Assessments draw from both the prose and the activities. As you revise, review the learning objectives at the start of each section and rehearse the self-check questions. Recreate Orange workflows from scratch, narrating why each widget is present and what metric confirms success. When you move into later weeks—such as the model evaluation guard rails in chapter 14 or the explainability toolkit in chapter 16—use earlier chapters as scaffolding rather than isolated notes. The habits you build in the opening weeks make ensembles, pipelines, and interpretability feel like natural extensions.

Keep exploring once you finish the course. Revisit the ‘Future Chapters and Activities‘ notes in the master schedule if you want to extend the textbook with new case studies or topics. The structure outlined here will help you integrate fresh material while keeping the reading experience cohesive for the next cohort.

Chapter 2

Data Science Roles in Modern Practice

Plain-language summary

This chapter is a people-focused tour of a data team. We describe who looks after dashboards, who designs experiments, who keeps the software reliable, and how those responsibilities build on one another. By the end you will know, in everyday terms, how a project moves from a question to an automated system without losing human oversight.

Chapter overview

Our tour of contemporary data science jobs lays the human groundwork for every technical chapter that follows. We map how teams shift from analytics to experimentation and engineering, highlight the lifecycle practices that keep production models healthy, and sketch the emerging responsibilities that now sit alongside traditional modelling. By the end of the

chapter we have a shared vocabulary for the roles who will appear throughout the book and a checklist of capabilities that the later technical chapters help us practise. Keep an eye on how the responsibilities link to upcoming uncertainty and clustering material—the probability refresher and retail case study immediately after this chapter show those roles in action and lead directly into Hands-on Activity 1.

2.1 Mapping the Talent Pipeline

This section will sketch the progression from entry-level analytics to senior leadership.

Learning objectives

After reading this section, you will be able to:

- distinguish the day-to-day responsibilities of analysts, data scientists, data engineers, and machine learning engineers;
- identify signals that justify progression from one role to the next;
- articulate how compensation and expectations scale across the ladder.

Prerequisites

Before starting, make sure you are comfortable with:

- reading descriptive statistics such as averages, medians, and growth rates;

- interpreting key performance indicators that appear on operational dashboards;
- collaborating with product and engineering stakeholders in agile rituals like stand-ups or sprint reviews.

This career ladder is easiest to understand by naming the jobs themselves. Graduates most often start as **data analysts**—early-career professionals who assemble clean datasets, interpret trends, and explain why the numbers matter for a decision. Their toolkit revolves around **dashboards**, curated displays of metrics that put charts and key performance indicators in front of stakeholders without requiring manual reporting. Organisations value that clarity, offering graduate packages in the ballpark of $80–100k for roles that keep executives informed and focused.

As responsibilities expand, analysts graduate into **data scientist** positions. These roles manage controlled experiments, analyse their outcomes, and ship **predictive models**, computational recipes that use historical features to estimate the likelihood of future events or behaviours.

Entry-level data scientists with one or two years of experience tend to see compensation closer to $90–120k, recognising that experimental design and model stewardship move measurable levers.

The pipeline continues through the infrastructure layers. **Data engineers** automate data collection, retraining, and monitoring so that freshly observed behaviour flows back into those models.

Organisations trust them with larger budgets—offers of $140–180k are not unusual when the remit includes production reliability.

At the senior end, **machine learning engineers** harden the intelligent systems themselves, tuning latency budgets, scaling serving infrastructure, and extending models into new interfaces such as chat-based assistants.

Only a few professionals leapfrog these rungs. A common story is to accept a data analyst posting, over-deliver by applying data science techniques to "boring" reports, and then slide into the modelling team when the next reorganisation arrives.

Recognising that progression helps us plan our learning: strong communication keeps the analyst seat secure, statistical rigour unlocks the data science move, and software discipline prepares us for engineering-grade expectations.

Each rung in the ladder rewards a distinct skills bundle. Analysts thrive when they can frame an ambiguous question, reshape tabular data with SQL, and present dashboards that help stakeholders see cause and effect at a glance. Data scientists add depth through inference and experiment management, designing variants that target one percent of traffic and reading the uplift with statistical care.

Data engineers bring the discipline of software teams, weaving version control, automated testing, and observability into the pipelines that retrain models overnight. Machine learning engineers extend that toolkit with systems-level thinking, managing latency budgets, failover plans, and the trust frameworks needed when an assistant drafts natural-language responses.

Building that sequence deliberately turns graduation requirements into a roadmap for career mobility.

Domain range accelerates each transition. Teams seek analysts who can relate manufacturing throughput, financial risk, or public policy objectives to the numbers on the page.

Those hybrid stories, where we pair modelling with another specialty, make it easier to propose experiments.

Whether we bring architectural insight to a built-/environment project or healthcare familiarity to a clinical workflow, combining perspectives signals readiness for the next role.

> **For Advanced Students.** Compare this ladder with career mobility theories such as boundaryless careers and tournament models. Research by Arthur, Inkson, and Pringle (1999) and Lazear and Rosen (1981) offers frameworks for how specialists accumulate human capital and compete for senior posts. Use those models to critique how incentives and organisational design either accelerate or hinder transitions between analytics, science, and engineering seats.

Section summary

- Data roles build from communicative analytics toward production-grade engineering and intelligent systems.
- Progression hinges on visibly delivering value at the current rung while developing the next role's technical depth.
- Compensation scales with the scope of systems owned and the reliability requirements imposed by the organisation.

Self-check questions

1. Which responsibilities clearly signal that someone is ready to move from a data analyst position into a data scientist role?

2. How do production reliability expectations differ between data engineers and machine learning engineers?

Self-check reflections

1. Look for analysts who already frame experiments, interpret causal uplift, and translate results into model-ready features; those habits show they can shoulder the responsibilities of a data scientist.

2. Data engineers keep data flows healthy through pipelines and monitoring, while machine learning engineers extend that remit to latency targets, failover plans, and the reliability of user-facing model interfaces.

2.2 Owning the Model Lifecycle

This section will outline operational practices that keep deployed systems reliable.

Learning objectives

After reading this section, you will be able to:

- describe the diagnose → experiment → engineer → oversee cadence for production models;

- explain why evidence gathering and monitoring must anchor each handover;

- evaluate how accountability is maintained even when automation increases.

Prerequisites

Before starting, make sure you can:

- interpret experiment summaries that report p-values, confidence intervals, or Bayesian credible intervals;
- trace how raw event data is transformed into features within an analytics warehouse;
- describe your team's incident response or escalation process for critical production issues.

The lifecycle opens with diagnosis. Analysts pull **transaction logs**[1], compare **drop-off rates**[2] with industry benchmarks, and narrate how current behaviour affects revenue or risk.

Those evidence-led stories define what needs to change and establish the baseline we will measure against.

With the problem agreed, data scientists orchestrate the interventions. They scope tightly targeted experiments—perhaps showing a revised checkout message to one percent of visitors—and analyse whether the treatment cohort behaves differently from the control group.

When the lift is real, they encode the winning changes into predictive models or decision rules that capture the lesson for future customers.

[1]Transaction logs are time-ordered records of user or system actions, such as page views or completed payments, captured to enable analysis and auditing.

[2]Drop-off rates measure the proportion of users who abandon a multi-step process before completion, helping teams pinpoint friction.

Worked example: diagnosing checkout abandonment. Consider an e-commerce platform that notices 35% of customers abandon their cart on the payment screen. Analysts triangulate evidence by inspecting the payment gateway's transaction logs, reviewing qualitative feedback, and calculating how the drop-off rate spikes during mobile sessions. They recommend testing a streamlined payment form.

Data scientists design a randomised experiment that exposes five percent of traffic to the new layout and analyse the resulting uplift. A statistically significant ten percent reduction in abandonment justifies deploying the change.

Data engineers embed the winning workflow into the production stack, add monitoring that flags if the drop-off rate rises above the new baseline, and schedule nightly backfills to keep derived features current. Product owners review automated alerts weekly and record approvals for each retrain cycle, closing the loop on accountability.

Data engineers then productionise the learning. They stream fresh interactions into **retraining pipelines**[3], watch for **context drift**[4] as customer habits evolve, and schedule model refreshes so the decisions stay aligned with the latest evidence.

Automation keeps yesterday's insight from stalling, but the engineering craft remains grounded in the experimental results that justified the model in the first place.

Oversight completes the loop. Directors and product owners stay accountable for the decisions their systems make,

[3]Retraining pipelines are automated workflows that collect new data, transform it into model-ready features, and refit models on a schedule.

[4]Context drift occurs when the external environment or user behaviour shifts enough that a model's original assumptions no longer hold, degrading performance unless addressed.

even when automated agents draft the weekly reports. Teams therefore continue to read those outputs, commission new experiments to double-check that retrained assistants behave safely, and document the approvals that keep models in service.

Owning the lifecycle is less about a one-time "handover" and more about maintaining a cadence of diagnosis, experimentation, engineering, and human review.

> **For Advanced Students.** Explore how control theory and socio-technical governance intersect in production ML. Åström and Murray's work on feedback systems (2010) offers mathematical tools for understanding monitoring thresholds, while Perrow's organisational theory (1999) helps evaluate how teams distribute responsibility when automated decisions scale.

Section summary

- Diagnosing with trustworthy evidence sets the baseline that later experiments must improve.
- Experiments capture causal impact, and engineering teams harden those lessons into durable services.
- Oversight enforces accountability by pairing automated monitoring with deliberate human review cycles.

Self-check questions

1. In the worked example, which evidence sources informed the decision to redesign the payment screen?
2. How can teams detect context drift early enough to prevent degraded customer experiences?

Self-check reflections

1. The decision drew on transaction logs, qualitative customer feedback, and the spike in mobile-session drop-off rates.

2. Teams combine automated alerts on key metrics, drift detection on feature distributions, and scheduled experiment reruns to catch context changes before they hurt customers.

2.3 Future-Facing Opportunities

This section will capture emerging responsibilities for data professionals.

Learning objectives

After reading this section, you will be able to:

- describe emerging areas of practice that extend beyond core experimentation and modelling;
- connect those opportunities to foundational skills developed earlier in the chapter;
- anticipate how upcoming hands-on work will reinforce the concepts.

Prerequisites

Before starting, ensure you:

- recall the diagnostic and experimentation cadence outlined in the previous section;

- understand basic principles of risk assessment and mitigation planning;

- can summarise findings for non-technical stakeholders in concise briefs or stand-up updates.

Looking ahead, new responsibilities continue to emerge. Evaluating AI assistants and autonomous agents now requires teams of data scientists to stress-test prompts, probe safety boundaries, and design **red-teaming protocols**[5].

These responsibilities feel adjacent to classic quality assurance, yet they rely on the experiment discipline and statistical sensitivity we practise throughout this book.

Demand is also rising for people who can design experiments at the pace of modern product iteration. Organisations want us to instrument digital experiences, hypothesise causal links, and deploy adaptive tests that respect ethical guardrails.

This skillset connects traditional inference with rapid feedback loops, positioning data scientists as partners in strategic planning rather than only technicians.

Communication remains the thread that ties every future opportunity together. Whether we are explaining why an AI assistant needs a safety review or summarising an experimentation roadmap, we succeed when we articulate uncertainty and impact in language that non-specialists understand.

Cultivating that narrative ability keeps our work inclusive and prepares us for the collaborative hands-on activity immediately after this chapter. In Hands-on Activity 1 you will

[5]Red-teaming protocols are structured exercises where specialists deliberately search for failure modes, security flaws, or harmful outputs before a system is widely deployed.

translate the role distinctions into practice by leading a cross-functional planning session for the counter-metric clustering lab, assigning analyst, scientist, and engineering responsibilities before the first counters are moved.

Future-facing teams also value breadth. Organisations repeatedly highlight their need for people who can pair technical intuition with experience in another discipline—economics, logistics, architecture, or public health—to shape analyses that resonate with decision-makers.

Portfolios that combine polished visualisations, carefully documented experiments, and domain narratives therefore become more than application material; they are evidence that we can convene cross-functional collaborators around the same table.

Section summary

- Emerging responsibilities stretch data professionals toward safety evaluation, adaptive experimentation, and strategic advisory work.
- Communication remains the enabling skill that allows technical findings to influence product, policy, and decisions.
- Upcoming practical work uses tangible counter-metric planning to rehearse the collaboration patterns described here.

Self-check questions

1. How do red-teaming protocols draw on experimentation skills learned earlier in the chapter?

2. Which communication techniques will you prioritise when coordinating the Hands-on Activity 1 planning session?

Self-check reflections

1. Red-teaming borrows the hypothesis framing, controlled testing, and measurement discipline from experimentation to probe how systems can fail before they reach customers.

2. Plan to translate technical risk into plain language, invite cross-functional input, and document decisions so everyone sees how the activity links back to the responsibilities mapped in this chapter.

Conclusion and next steps

By tracing the talent ladder, the lifecycle rhythm, and the frontier responsibilities, we have positioned ourselves to talk about methods with an appreciation for who keeps them alive in production. The themes that resonated here—clear communication, evidence-led experimentation, and reliable engineering—will resurface in the probability grounding and the retail clustering case study that follow.

When we turn the page to Hands-on Activity 1, use this chapter as a checklist: assign analyst, scientist, and engineering duties deliberately so the tabletop k-means exercise mirrors the collaborative cadence we expect in practice.

Chapter 3

Probability Foundations for Data Science

Plain-language summary

This chapter is a refresher on how we talk about uncertainty. We explain, in everyday words, how to think about chance as either “how often something happens" or “how strongly we believe it," and we show how that thinking helps us make sense of data and decisions. If probability jargon has ever felt mysterious, this chapter translates it into plain speech before the heavier chapters arrive.

Chapter overview

This chapter refreshes the probability language we rely on for the rest of the book. We revisit interpretations of probability, connect uncertainty to inference, define the vocabulary of distributions and random variables, and make simulation tools feel practical. Together those sections give us the mental

scaffolding we need before the clustering case study and hands-on activity that follow. As you read, connect each concept to the way you explain evidence in your own projects—the upcoming retail analysis will turn these ideas into decisions about depot placement and the tabletop k-means exercise in Hands-on Activity 1 will lean on the same intuition when we translate randomness into physical movement.

3.1 Contrast Interpretations of Probability

We compare frequentist and Bayesian viewpoints so our shared language for uncertainty remains consistent throughout the course.

Learning objectives

After reading this section, you will be able to:

- state the frequentist definition of probability as a long-run relative frequency;
- describe Bayesian probability as a quantified degree of belief that still obeys the same algebra;
- articulate when each interpretation provides intuition for real-world analytics problems.

Prerequisites

Before starting, make sure you can:

- compute simple relative frequencies from repeated experiments;

- read Venn diagrams or tree diagrams that represent joint and conditional events;
- explain in plain language what "uncertainty" means in the context of measurement or forecasting.

The frequentist perspective treats probability as the proportion of times an event occurs when we repeat an experiment indefinitely. When we say the probability of a checkout crash is 5%, we mean that over thousands of deployments, about one in twenty pushes fails. This long-run framing is comfortable when we can imagine physically repeating the process—rolling dice, rerunning an A/B test, or simulating a customer journey under identical conditions. The vocabulary it gives us is grounded in counts, ratios, and the arithmetic of events, which is why it appears in introductory statistics courses and operational dashboards alike.

Bayesian probability broadens that language. Instead of demanding an infinite sequence of identical experiments, it lets us encode how strongly we believe an explanation fits the evidence. A product manager might judge there is a 70% chance that a new subscription feature will reduce churn even before a single user interacts with it, based on market research and previous launches. That belief is still disciplined by the same addition and multiplication rules we already rely on; the algebra of probability does not change. What changes is what the numbers represent: not just long-run frequencies, but coherent degrees of belief we intend to update when new data arrives.

Bringing the two perspectives together is pragmatic. The frequentist viewpoint reassures us that our metrics have tangible interpretations tied to counts and rates. The

Bayesian mindset keeps inference problems—"what explains these data?"—within reach by letting us start from defensible prior knowledge. Throughout the textbook we lean on that Bayesian framing, especially when we evaluate models or weigh competing hypotheses, because it mirrors how analysts actually reason when the data we have are finite and decisions cannot wait for infinite repetition.

Section summary

- Frequentist probability expresses uncertainty through long-run proportions observed over repeated experiments.
- Bayesian probability quantifies how strongly we believe an explanation, updating those beliefs as new evidence appears.
- Both interpretations share the same algebra, letting us switch intuition without rewriting the underlying rules.

Self-check questions

1. When do repeated-trial arguments give you the clearest intuition for probability, and when do you prefer to reason in terms of belief?
2. How would you explain to a stakeholder that Bayesian probabilities still obey the same addition and multiplication laws as frequentist probabilities?

Self-check reflections

1. Repeated-trial arguments shine when you can picture running the process many times, while belief-based lan-

guage helps when you must make a call before those repetitions exist.

2. Emphasise that both views share the same arithmetic rules; Bayesian probabilities are simply beliefs that we promise to update with evidence, not a different kind of mathematics.

3.2 Connect Probability to Inference

Probabilistic reasoning lets us move from data to causes, preparing us for the modelling chapters that follow.

Learning objectives

After reading this section, you will be able to:

- distinguish between forward simulation (predicting outcomes) and inference (explaining observed data);
- use Bayes' rule conceptually to describe how new evidence revises our beliefs;
- outline how probabilistic thinking supports decision-making in analytic projects.

Prerequisites

Before starting, ensure you can:

- interpret conditional statements such as "the probability of A given B";
- summarise observational data sets with basic descriptive statistics;

- articulate a business or policy question that requires understanding why something happened, not just what will happen next.

Inference is about running probability in reverse. Instead of only forecasting what will happen if we know the data-generating process, we take the data we have and ask which underlying process is most plausible. Suppose a manufacturing line records a spike in defects. A purely forward model might simulate the number of defective units we expect tomorrow; an inferential analysis weighs competing explanations—material fatigue, operator error, or a calibration drift—and quantifies how likely each one is given the evidence collected. Probability becomes the currency of those arguments.

Bayes' rule formalises the reasoning. We start with prior beliefs about each explanation, perhaps informed by maintenance logs or earlier incidents. As new measurements arrive—sensor readings, visual inspections, or customer complaints—we assess how consistent those data are with each hypothesis. The rule tells us how to combine prior beliefs with the likelihood of the evidence to produce posterior beliefs (our after-evidence view of the world) that reflect both sources. Even when we do not plug numbers into the formula, the workflow guides our questions: What did we believe before? How surprising are the data under each candidate explanation? What should we believe now?

This inferential framing appears throughout the book. When we evaluate a clustering workflow in Orange, we are implicitly asking how credible each segmentation is given our observations. When we diagnose a regression model, we weigh whether residual patterns are more compatible with noise or with a missing variable. Working in a Bayesian frame keeps

us honest about uncertainty, encourages us to seek additional evidence when the posteriors are diffuse, and equips us to explain to stakeholders why a recommendation is justified even when certainty is impossible.

Section summary

- Inference turns observed data into statements about underlying causes, rather than only predicting future outcomes.
- Bayes' rule provides the scaffold for updating beliefs as new evidence arrives.
- A probabilistic mindset supports transparent decision-making across analytics projects.

Self-check questions

1. How would you describe the difference between simulating tomorrow's demand and inferring why today's demand fell short?
2. In a recent project, what served as your prior belief and what evidence prompted you to revise it?

Self-check reflections

1. Simulation pushes probability forward to forecast outcomes, whereas inference runs it in reverse to explain the causes behind the data you already saw.
2. Name the baseline assumption you held, then the fresh evidence—such as new metrics, interviews, or experiments—that nudged you to update that belief.

3.3 Define Distributions, Random Variables, and Samples

We cement vocabulary around distributions, events, and random variables before heavier tools appear later in the course.

Learning objectives

After reading this section, you will be able to:

- define a probability distribution and give examples from discrete and continuous settings;
- explain what makes a variable "random" in analytic practice;
- interpret histograms and sampled data as evidence of an underlying distribution.

Prerequisites

Before starting, make sure you can:

- read bar charts and histograms and describe what they show;
- distinguish categorical variables from numerical ones;
- recognise units of measurement relevant to your domain (such as dollars, minutes, or centimetres).

A **distribution** assigns probabilities to events. For a weekend football match we might describe the chances of a home win, a draw, or an away win; for customer support we might assign probabilities to response times falling within service-level agreements. Each distribution encodes the same

idea: every mutually exclusive outcome gets a probability, and the list of probabilities sums to one. In discrete cases—dice, survey responses, or product categories—we can list those outcomes individually. In continuous cases—height, rainfall, or latency—we talk about ranges because the probability of any exact real number is effectively zero.

The concept extends naturally to **random variables**. A random variable is any quantity whose value is governed by a distribution. We treat the daily number of checkouts, the temperature recorded at noon, or the proportion of positive reviews as random variables because, before observing them, each has a spread of plausible values. Naming a variable as random does not make it mysterious; it simply acknowledges that we model its uncertainty formally.

Histograms make these definitions tangible. When we sample a random variable repeatedly—collecting customer satisfaction ratings for a month or measuring the fuel efficiency of vehicles in a fleet—we can plot the counts of observations falling into bins. Smooth histograms hint at the shape of the underlying distribution, revealing whether outcomes cluster, skew, or exhibit multiple modes. Thinking in this way prepares us for later chapters where we must choose models whose assumptions about distributions align with the evidence in front of us.

Section summary

- Distributions map every possible event or range of values to a probability.
- Random variables are the measurable quantities we treat as governed by those distributions.

- Samples and histograms provide empirical glimpses of the underlying distributional shape.

Self-check questions

1. Which variables in your current project behave discretely, and which are better described with continuous distributions?
2. How does a histogram help you decide whether a modelling assumption, such as symmetry or unimodality, is reasonable?

Self-check reflections

1. Transaction counts or survey responses take discrete values, while timing, spend, or sensor readings usually fall on a continuous scale.
2. Histograms quickly show whether the data look balanced, skewed, or multi-peaked, helping you judge whether simple modelling assumptions fit or whether you need more flexible approaches.

3.4 Demystify Simulation Basics

We introduce pseudo-random generators and histogramming so the law of large numbers becomes concrete.

Learning objectives

After reading this section, you will be able to:

- explain how pseudo-random number generators approximate random draws in software;

- describe how sample size affects the stability of estimated distributions;
- interpret simulations that illustrate the law of large numbers and small-sample volatility.

Prerequisites

Before starting, ensure you can:

- run simple scripts in Python, R, or another analytics language;
- manipulate arrays or data frames to calculate summary statistics;
- read overlays of multiple histograms and compare their shapes.

Most analytics stacks ship with pseudo-random number generators. When we call `numpy.random.rand`, the library produces numbers that mimic independent draws from the uniform distribution on $[0, 1)$. Under the hood, deterministic algorithms transform an initial seed into a sequence of values that pass statistical tests for randomness. For exploration and teaching, these generators are more than adequate, letting us experiment with uncertainty without building custom hardware.

Simulations reveal how sample size shapes our understanding. Drawing one hundred values from a uniform distribution and plotting their histogram gives a noisy approximation of the underlying flat shape; repeating the exercise with ten thousand draws looks smoother, and a million draws produces a histogram that visually hugs the theoretical line. This is

the **law of large numbers** in action: as the number of independent samples increases, the sample average converges on the expected value, and sample histograms resemble the true distribution. At the same time, small-sample behaviour reminds us why caution is warranted. Tiny samples can produce streaks or gaps that disappear with more data, so we avoid over-interpreting early patterns.

A quick experiment cements the lesson. Generate uniform samples in three batches—10^2, 10^4, and 10^6 draws—and overlay the resulting histograms. Record how the variance of the sample mean shrinks and how the jagged edges smooth out. Then repeat the process with biased coins or skewed distributions. In each case we see the same trajectory: more data gives stability, and visualisations of the cumulative average settle around the true value. These mental anchors prove invaluable when we interpret Orange outputs or judge whether a practical sample size is sufficient for reliable conclusions.

Section summary

- Pseudo-random generators let us explore probabilistic ideas inside familiar programming environments.

- Large samples make empirical summaries converge to their theoretical counterparts, demonstrating the law of large numbers.

- Small-sample volatility is expected, reminding us to temper conclusions when data are scarce.

Self-check questions

1. How many samples did you need in your own simulation before the histogram looked stable, and why?

2. What safeguards can you build into analyses to avoid overreacting to artefacts from tiny samples?

Self-check reflections

1. Most simulations need thousands of draws before the shape settles; the volume smooths out random bumps that dominate tiny samples.

2. Require minimum sample sizes, report confidence intervals, and rerun simulations with new seeds to confirm that early patterns are not just noise.

3.5 Highlight the Central Limit Theorem

We unpack why sums of draws trend normal, motivating the bell curve as a default model for noisy processes.

Learning objectives

After reading this section, you will be able to:

- state the central limit theorem (CLT) in practitioner-friendly language;

- describe the conditions under which sums or averages of random variables become approximately normal;

- connect the CLT to diagnostics and modelling choices used later in the course.

Prerequisites

Before starting, make sure you can:

- compute means and standard deviations for sample data;
- explain the difference between summing and averaging observations;
- recognise normal distribution curves and their key features.

Adding independent random variables together smooths out their idiosyncrasies. Start with uniform draws between zero and one: the distribution of a single draw is flat, the sum of two looks triangular, and the sum of ten already resembles a bell curve. As we keep adding draws—or, equivalently, averaging them—the shape tends toward the normal distribution. The central limit theorem formalises that progression. It states that if we take independent draws from any distribution with a finite mean μ and variance σ^2, then the distribution of their average approaches a normal distribution with mean μ and variance σ^2/n as the sample size n grows.

This result is more than a mathematical curiosity; it justifies the approximations we rely on in modelling and diagnostics. Confidence intervals, hypothesis tests, and control limits often assume normality not because the original data are bell-shaped, but because sums or averages of many observations inherit that shape through the CLT. When we evaluate model residuals or compute feature averages in later chapters, we implicitly lean on this behaviour to argue that our summary statistics have predictable variability.

Awareness of the theorem's boundaries keeps us cautious. Heavy-tailed distributions with infinite variance, serial depen-

dence between observations, or extremely small sample sizes can all undermine the normal approximation. In those cases we reach for alternative techniques—bootstrapping, transformation, or explicit time-series models—to respect the data's structure. Most of the time, though, the CLT gives us permission to proceed with normal-based reasoning, connecting this chapter directly to the clustering counters in Hands-on Activity 1 where we average distances and rely on smooth distributions to interpret silhouette scores.

Section summary

- The central limit theorem explains why sums and averages of many independent draws often look normal, regardless of the original distribution.

- Normal approximations underpin confidence intervals, residual diagnostics, and other analytical tools used later in the course.

- Understanding the theorem's assumptions helps us recognise when to trust the approximation and when to reach for alternatives.

Self-check questions

1. Which workflow in your team averages many observations, and how does the CLT justify treating that average as approximately normal?

2. When might the CLT fail, and what alternative analysis strategies would you employ in those situations?

Self-check reflections

1. Daily revenue reports or rolling satisfaction scores average heaps of observations; the CLT tells us those averages will be bell-shaped, so confidence intervals make sense.

2. The theorem struggles with heavy tails, dependence, or tiny sample sizes. In those cases, bootstrap resampling, robust summaries, or time-series models respect the structure better than a plain normal approximation.

Conclusion and next steps

We have re-established probability as the thread that links our analytic storytelling, our model diagnostics, and our simulation toolkit. With interpretations, inference, vocabulary, and computational practice aligned, we can now read the retail clustering case study with confidence about how uncertainty shapes business framing. Keep these probability guard-rails handy as we pivot into the depot-planning narrative and then step into Hands-on Activity 1—they will help you justify why the clusters we build matter and how much trust to place in the experimental evidence we gather along the way.

Chapter 4

Retail Analytics Case Study: Mapping Store Clusters

Plain-language summary

This chapter tells a story about planning delivery depots for a supermarket chain. We look at a spreadsheet of real stores, group them into sensible regions, and decide where a depot should sit so groceries stay fresh. By breaking the work into plain steps, we show how the data, the maps, and the final recommendation all connect.

Chapter overview

Here we translate the probability mindset into a concrete retail expansion challenge. The chapter opens by grounding the business problem and data assets, walks through exploratory framing, and then demonstrates how clustering insights inform

depot planning. Each section keeps the decisions in view so that, when we move into Hands-on Activity 1, we already know why the tabletop k-means exercise matters. Treat this narrative as rehearsal for the case-work we will perform later in the semester: we are practising how to frame objectives, interrogate features, and summarise findings for stakeholders before Orange even loads.

4.1 Business Context and Data Assets

We ground the clustering exercise in a realistic expansion problem before introducing tooling decisions.

Learning objectives

After reading this section, you will be able to:

- describe the retail expansion problem that motivates the clustering workflow;
- catalogue the features supplied in the Indian supermarket dataset and how they relate to that problem;
- draft guiding questions that anchor the analysis to practical decisions.

Prerequisites

Before starting, confirm that you can:

- interpret tabular data containing geographic coordinates and sales attributes;
- read basic thematic maps or scatter plots that use colour to encode categories;

- explain the difference between exploratory questions and formal hypotheses.

Picture us as the analytics crew for a supermarket chain that wants more reach without blowing the logistics budget. The team has a chunky spreadsheet of independent grocery stores that specialise in Indian produce—perfect prospects for partnerships or acquisitions. Each row includes the address, local government area, suburb-level socio-economic indicators, and an estimate of annual turnover. A few rows also log floor space and whether a rail station sits nearby. Even before we leave our desks, those details help us reason about catchment size, spending power, and supply constraints.

This storyline lines up with the Week 6 lecture on unsupervised learning where we unpack how proximity features shape cluster geometry. It also primes us for Practical 7, in which the hands-on worksheet has you load the very same workbook and warm up with exploratory questions before jumping into Orange.

Our immediate question is simple: how many distribution depots should we run across the metro area? Depots cost real money, but marathon delivery runs spoil fresh produce and wear out store managers' patience. Clustering the stores into tidy territories lets each depot look after a compact region. A side quest is spotting neighbourhoods that feel under-served so we can pitch a new store that cuts travel time for customers chasing specialty ingredients.

International adaptation guide. Swap in local chains or community grocers if you are teaching outside Australia. Replace suburb names, socio-economic indicators, and supplier references with equivalents drawn from the region you

serve—census data, transit maps, or open retail registries work well. The analytical flow stays identical while the story feels authentic to your cohort.

Before we open Orange, we jot down a few guiding questions. Which suburbs already fall into natural demand corridors, maybe following a rail line or a major road? Where do the revenue estimates and demographic indicators hint that a premium range would actually sell? How close is each store to its nearest competitor, and does that spacing hint at unmet demand? Keeping those questions in front of us stops the clustering exercise from drifting into a purely mathematical puzzle.

We also flag the rough edges in the data straight away. The workbook is one snapshot, so we can't read seasonal swings from it. Turnover figures mix sources: some come from industry reports, others from quick-and-dirty web scraping. Calling out those caveats now keeps us alert for oddball rows during exploration and nudges us to plan a richer data feed later, perhaps by teaming up with finance to capture point-of-sale data once a pilot depot is live.

> **Advanced challenge — alternative domains.** Sketch how the same feature inventory would morph if we pivoted to urban planning or healthcare logistics. For instance, swap turnover for hospital bed utilisation and introduce travel-time buffers around emergency departments. Identifying the substitutions prepares you for the optional comparative analysis brief in Assignment 2.

Section summary

- The case study models a supermarket chain deciding how many depots to operate while maintaining customer access.

- The dataset combines geographic coordinates with revenue and demographic signals that frame the clustering task.

- Guiding questions keep the analysis tied to expansion decisions and highlight data limitations that require follow-up.

Self-check questions

1. Which dataset columns most directly influence the number of depots you would recommend, and why? *Hint: revisit the business objective before naming features.*

2. How would you validate turnover estimates before committing budget to a new distribution centre? *Hint: think about third-party data and stakeholder interviews.*

3. Draft one clarifying question for the merchandising team that would reduce uncertainty in the demographic indicators. *Hint: target the weakest assumption in the dataset.*

4. Compare two suburbs with similar turnover but different accessibility scores. What follow-up evidence would you gather? *Hint: blend quantitative data with observations from store visits.*

5. Suppose a stakeholder suggests clustering purely on geography. How would you justify the added socio-economic features? *Hint: link back to demand variation across territories.*

Self-check reflections

1. Focus on turnover, accessibility, and demographic columns because they speak directly to depot workload and demand spread.

2. Cross-check turnover with finance records, supplier invoices, or industry benchmarks, and interview store managers to confirm the figures.

3. Ask for the source and refresh cycle of each demographic field so you know whether the data reflect current neighbourhood change.

4. Pair the quantitative gap with site visits, transport surveys, or customer interviews to understand whether the accessibility score aligns with lived experience.

5. Explain that geography alone misses spending power and cultural demand; socio-economic features sharpen recommendations about stock range and service levels.

4.2 Feature Engineering and Visual Exploration

We transform the raw workbook into spatial features and visual diagnostics that inform clustering choices.

Learning objectives

After reading this section, you will be able to:

- geocode store addresses and align them with base maps for exploratory analysis;
- engineer density, proximity, and accessibility features that highlight structural patterns;
- use interactive dashboards in Orange to iterate on feature choices before modelling.

Prerequisites

Before starting, ensure you can:

- convert street addresses into latitude and longitude pairs using a trusted geocoding service;
- calculate simple distance measures between coordinate pairs;
- operate the File, Scatter Plot, and Geo Map widgets in Orange.

We start by geocoding each store address. OpenStreetMap's Nominatim API or a paid service such as Google Maps gives us latitude and longitude pairs that we drop straight into the spreadsheet. Once those coordinates exist, we load the dataset into Orange, mark the geographic columns as features, and throw the points onto a scatter plot. Longitude on the x-axis and latitude on the y-axis already sketches a map that shows chains of stores hugging train lines and main roads.

With that map scaffold in place, we craft features that explain the local market. Population density and household

income arrive at the suburb level, so we join them in and scale them so they play nicely together. We also work out the road-network distance from each store to the central business district and to its two closest competitors—handy stand-ins for delivery effort and competitive pressure. A quick flag for "near a rail station" rounds out the list because easy public transport access often lines up with steady foot traffic.

Interactive dashboards then tell us whether the engineered features are doing their job. Inside Orange we connect the Scatter Plot, Data Table, and Geo Map widgets, colouring points by potential cluster labels or by something like turnover per square metre. We drag a lasso around stores within five kilometres of a proposed depot site and compare their demographics with the outer suburbs. If a feature looks messy—say turnover fails to separate inner-city and suburban stores—we hop back to the engineering step to smooth it or swap in a better proxy.

While we poke around, we jot quick notes about interesting feature combos to revisit later. Dense, rail-adjacent corridors might deserve weights that prioritise accessibility, while isolated outer suburbs probably need the opposite. Capturing those hunches now saves time when we start tuning clustering hyperparameters.

Worked example: mapping the enrichment pipeline.

Step 1: Launch Orange and drag the `File` widget onto the canvas. Point it at the engineered workbook from Practical 7.

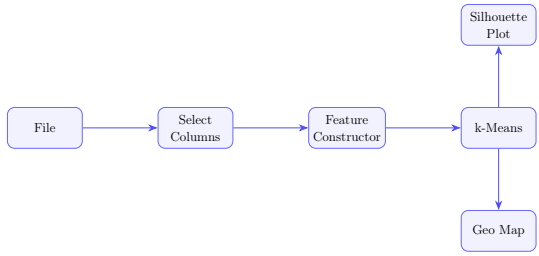
Step 2: Connect a `Select Columns` widget so we can promote latitude, longitude, and engineered features such

as competitor distance into the feature space while parking identifiers.

Step 3: Add a `Feature Constructor` widget to compute population density per square kilometre and a boolean rail-access flag. Document the formulas in the widget notes so teammates can audit the transformations.

Step 4: Branch the workflow: send one path to a `Geo Map` widget for spatial context and another to a `Scatter Plot` widget filtered by turnover quantiles. Use synchronised brushing to watch how high-turnover stores cluster in the inner suburbs.

Step 5: Save the workflow as `retail-territories.ows`. We'll reuse it when testing clustering parameters in the next section.



Orange workflow showing how engineered features feed both evaluation diagnostics and spatial sense-making.

Figure 4.1: Orange canvas showing File, Select Columns, and Feature Constructor feeding into k-Means, with outputs branching to Silhouette Plot and Geo Map for side-by-side evaluation.

This sequence mirrors the second activity on the Week 7 practical sheet, so ticking it off now leaves more time in class

for experimentation with optional widgets such as `Heat Map` or `Distance Matrix`.

Section summary

- Geocoding and map visualisation reveal the spatial structure of the supermarket network.
- Engineered features covering density, distance, and accessibility sharpen the contrasts between territories.
- Interactive Orange dashboards guide iterative refinement before we commit to a clustering configuration.

Self-check questions

1. Which engineered feature most improved your ability to spot over-served or under-served neighbourhoods? *Hint: compare maps before and after the feature was introduced.*
2. How would you test whether the proximity-to-rail indicator truly correlates with higher turnover? *Hint: design a quick statistical check inside Orange.*
3. What visual diagnostic convinced you that population density needed rescaling? *Hint: inspect colour gradients across the Geo Map.*
4. If the Geo Map renders all stores in one colour, which widget configuration will you revisit first? *Hint: start with the data role assignments in Select Columns.*
5. Describe how you would extend the workflow to handle quarterly refreshes of the dataset. *Hint: focus on reusable feature-construction steps.*

Self-check reflections

1. Density or turnover-per-store features usually make the hot spots stand out once you compare pre- and post-feature maps.

2. Run a quick correlation or train a simple regression in Orange, then check whether the relationship holds across folds.

3. The Geo Map's uneven colour stretches or a lopsided legend typically flag the need to rescale population density.

4. Return to Select Columns to ensure the cluster assignment is marked as a meta attribute and colour role; that usually fixes flat map colours.

5. Save the feature-engineering steps as a documented script or Orange workflow so you can rerun them each quarter without reinventing the pipeline.

4.3 Clustering Workflow and Decision Support

We operationalise the engineered features through k-means and translate the outputs into expansion recommendations.

Learning objectives

After reading this section, you will be able to:

- configure a k-means workflow in Orange that respects geographic scale and business constraints;

- evaluate candidate clusterings using silhouette diagnostics and scenario analysis;
- convert cluster assignments into depot placement guidance and follow-up investigations.

Prerequisites

Before starting, confirm that you can:

- explain how the k-means algorithm partitions observations around centroids;
- interpret silhouette scores as measures of cohesion and separation;
- build simple Orange workflows that chain File, k-Means, and visualisation widgets.

The workflow kicks off with a File widget pointing at the engineered dataset and a Select Columns widget that keeps latitude and longitude as features while dropping identifiers from the distance math. Inside the k-Means widget we leave normalisation off because the coordinates already share a sensible scale. If we add features with different units later, we standardise them before they meet k-means. From there we experiment with the number of clusters, starting with two depots as a bare-bones network and nudging the count upward while watching the silhouette score.

Silhouette scores range from -1 to 1 and act like a comfort meter: values near 1 mean a store sits happily inside its cluster, numbers around 0 signal a border case, and negative values warn that the store probably belongs elsewhere.

Illustrative scenarios for Week 1

Example 1 – Northern corridor depot debate.

Step 1: Launch a `Geo Map` and increase the marker size and opacity so the metropolitan layout is legible from the back of a lecture theatre.

Step 2: Insert `k-Means`, keep only latitude and longitude in the feature set, and disable normalisation to preserve the true geographic scale while we compare depot spacing.

Step 3: Branch the centroid output to a second `Geo Map` so we can contrast store clusters with suggested depot locations on side-by-side maps.

Step 4: Facilitate a quick reflection on whether a lone southern store justifies its own depot or whether we should revisit the feature mix before committing capital.

This walkthrough gives the cohort a shared visual narrative to reference when they recreate the workflow during the hands-on activity. We make the business tension explicit—balancing convenience for Engleburn against the cost of an extra warehouse—so later recommendations sound grounded rather than algorithmic.

Example 2 – Silhouette-guided network sizing.

Step 1: Switch the `k-Means` widget to suggest cluster counts, surface the silhouette scores, and narrate the cost trade-off that keeps two depots attractive even when more clusters look tempting on paper.

Step 2: Keep the `Geo Map` visible while adjusting k live so we can compare how depot coverage shifts between two, three, and seven territories.

Step 3: Add a `Silhouette Plot`, link it back to the clustering, and highlight skinny bars to show which stores sit uneasily between territories before jumping back to the map for geographic context.

Step 4: Close by demonstrating how a stray categorical variable such as postcode can distort the centroids, then filter it out so the silhouette plot settles on meaningful territories again.

Framing silhouette analysis alongside the live map prepares readers to justify their chosen k in writing. It also models the reflective loop we expect in assessments: diagnose ambiguous stores, revisit the feature set, and only then lock in a depot plan.

Silhouette diagnostics give us both numbers and pictures. Orange shows the average silhouette width for each run so we can see how cleanly separated the clusters are. We scan the plot for skinny bars—usually a hint that one store is stranded on its own. When that happens, we revisit the feature set to check for data issues or missing context. We also hook the cluster output into a Geo Map widget to colour regions by assignment and check whether the suggested centroids land in sensible spots along transport corridors rather than on top of existing stores.

Decision support appears when we mix business rules into the clustering output. For each cluster we tally total turnover, average delivery distance, and the number of stores that miss our service-level target. Maybe the two-depot setup scores

best on silhouette, but if one depot would face hour-long delivery runs, adding a third depot still makes sense. We play out quick scenarios—What if fuel prices jump 15%? What if a new competitor pops up in the western suburbs?—so we have talking points ready for capital allocation conversations and a list of extra data that would shrink the uncertainty.

To keep the workflow alive, we line up regular refreshes of the clustering with new sales data so depot assignments evolve with demand. Field teams can sense-check the suggested territories by asking store managers about delivery pain points. The hands-on activity after this chapter nudges readers to rebuild the Orange workflow, test different cluster counts, and recommend a depot plan backed by diagnostics and on-the-ground insight.

Worked example: tuning k-means with diagnostics.

Step 1: Open the saved `retail-territories.ows` workflow and insert a `k-Means` widget after the feature engineering steps. Set the number of clusters to two to establish a baseline.

Step 2: Enable the "Silhouette" output on the `k-Means` widget and connect it to the `Silhouette Plot`. Note the average score and identify any thin bars indicating unstable assignments.

Step 3: Duplicate the `k-Means` widget, change k to three, and point it at the same feature set. Compare silhouette scores and check the `Geo Map` to see whether depot territories now respect natural transport corridors.

Step 4: Add a `Data Table` widget that calculates mean deliv-

ery distance per cluster. Use this table to craft a short recommendation memo for the logistics manager.

Step 5: Capture screenshots of both the silhouette output and the tuned parameter panel. They form part of the reflective write-up for Practical 7 and provide evidence for Assignment 1's modelling section.

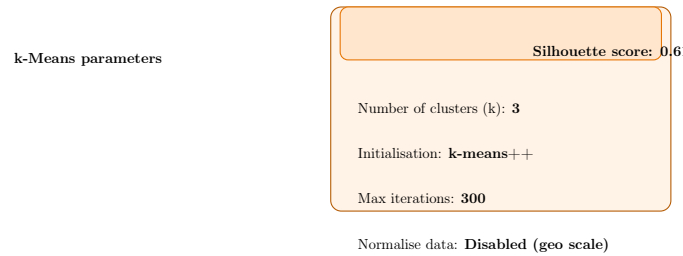


Figure 4.2: Annotated k-means configuration reflecting the tuned setup from Practical 7.

Troubleshooting sidebar.

- **"No data on input" warning:** Check that each widget has an active connection and that the `File` widget is pointed at the refreshed CSV, not a temporary preview file.
- **Silhouette plot shows negative bars:** Inspect the feature scaling. Revisit the `Feature Constructor` outputs or add a `Normalize` widget before k-means.
- **Geo Map missing tiles:** Switch the map provider to OpenStreetMap or refresh the internet connection proxy settings under Orange preferences.

- **Workflow crashes on save:** Remove absolute file paths in the `File` widget and use project-relative paths as recommended in the Week 7 practical handout.

Advanced challenge — beyond k-means. Experiment with a duplicate workflow that swaps `k-Means` for `Hierarchical Clustering` and `DBSCAN`. Compare dendrogram cut heights with silhouette scores, and reflect on how the choice of distance metric relates to the optimisation view of k-means as minimising within-cluster variance.

Theory connections for advanced readers

K-means minimises within-cluster squared error, a concrete instance of unconstrained optimisation. The objective is non-convex, so random initialisation affects which local minimum we reach; that's why k-means++ seeding and multiple restarts matter. From a statistical learning perspective, the algorithm approximates a vector quantiser that balances bias (few centroids) and variance (many centroids). Linking these ideas back to the Week 6 lecture helps frame why we monitor silhouette scores alongside business constraints.

These choices also tie directly to assessment. Assignment 1 asks for a depot recommendation supported by both diagnostics and cost reasoning. Documenting how the optimisation objective interacts with service-level targets demonstrates depth and distinguishes a Credit submission from a Distinction-level narrative.

Section summary

- Configuring k-means in Orange requires careful feature selection, scale management, and iterative tuning.
- Silhouette plots and geo-visualisations expose when a clustering produces unstable or impractical territories.
- Scenario analysis connects the technical output to depot planning decisions and follow-up fieldwork.

Self-check questions

1. Which business constraint would prompt you to accept a lower silhouette score in exchange for more clusters? *Hint: revisit service-level targets and delivery-time thresholds.*
2. How will you validate the recommended depot locations with stakeholders before committing capital expenditure? *Hint: outline a pilot or field validation plan.*
3. What does a negative silhouette bar tell you about a store's assignment, and how would you respond? *Hint: consider both feature engineering and cluster count adjustments.*
4. Describe one optimisation insight from the theory connections subsection in your own words. *Hint: link the k-means objective to a business-friendly explanation.*
5. Propose a short troubleshooting checklist to run before presenting results to leadership. *Hint: include both Orange diagnostics and data refresh checks.*

Self-check reflections

1. Service guarantees like maximum delivery times or per-depot load caps might justify extra clusters even if the silhouette drops.

2. Plan a depot pilot, gather manager feedback, and compare delivery metrics before and after to confirm the recommendation lands well.

3. A negative bar signals a store fits another cluster better; revisit the features or try a different k to see whether the store finds a clearer home.

4. Explaining that k-means balances keeping stores close to a centroid against splitting them too finely reframes the optimisation math as a trade-off between efficiency and complexity.

5. Check widget connections, confirm the latest CSV is loaded, review silhouette plots for outliers, and document any manual adjustments before presenting.

Conclusion and next steps

This case study nudges us to pair probability fluency with business framing. We now have a story, a dataset, and a decision structure that will carry into the hands-on k-means rehearsal. As you start Hands-on Activity 1, revisit the guiding questions you sketched here and use them to interpret the counter clusters you produce. The stronger your link between the physical exercise and the depot-planning narrative, the easier it will be to argue for follow-up analyses in future weeks.

Chapter 5

Hands-on Activity 1

This session is all about grouping similar things together by eye before we let software take over. We sort plastic counters into clusters, notice which tokens naturally travel together, and then rebuild the same idea inside Orange so the computers follow our reasoning. It is the hands-on bridge between the people, probability, and retail stories you just read.

We anchor this activity in the practice frameworks from chapter 2, the shared probability language in chapter 3, and the workflow guidance in chapter 4 so our tabletop clustering mirrors the theory chapters that precede it.

5.1 Prepare the Materials

Each group needs a small kit so everyone can participate in the physical clustering exercise before opening Orange. Plan on the following items per group:

- **Kadink Bulk Pack Counters (230 pieces).** Set aside roughly one pack per group so there are plenty of

purple tokens to stand in for data points. Order from Officeworks (Kadink Counters Bulk Pack 230 Piece).

- **Kadink Dinosaur Counters (90 pack).** Allocate about a handful (four to five) dinosaur counters to act as moving centroids. A single 90-pack comfortably supplies up to eight groups. Purchase from Officeworks (Kadink Dinosaur Counters Assorted 90 Pack).

- **30 cm Binder-mate style ruler.** Provide one ruler per group so they can measure distances between tokens during the activity. Pick up the Studymate recycled ruler from Officeworks (Studymate Binder Mate Recycled Ruler 30 cm).

Bring shallow tubs or trays so each group can keep their counters contained, and have a phone or camera handy to capture the final layout.

International substitutions. Swap in locally sourced counters, toys, or household objects if Officeworks or Big W are not available where you teach. Any colourful tokens, small figurines, and rulers of similar size will keep the activity identical while reflecting local retailers.

5.2 Cluster Tokens by Hand

Gather everyone around a table with their tub of purple counters, four or five dinosaur counters, and a ruler.

1. Sort the mixed counters so the purple pieces become the "data" tokens. Keep the dinosaur counters aside; they will become centroids.

2. Scatter the purple tokens across the table. Create a few tight clusters and leave some points more isolated.

3. Start with three clusters. Place three dinosaur counters (for example red, green, and blue) anywhere on the table.

4. For each purple token, decide which dinosaur is closest. Use the ruler if you need help judging distances, and put a matching coloured counter on top of the purple token to record its temporary cluster assignment.

Figure 5.1: Coloured counters arranged on a tabletop with dinosaur figurines ready to act as centroids, showing how physical tokens represent data points during clustering.

5. Move each dinosaur to the mean (average) position of the purple tokens that chose it. A grid and precise coordinates would give a perfect mean, but a careful visual estimate works for this demonstration. If a dinosaur is "lonely" (no tokens selected it), relocate that dinosaur to the most distant purple token.

6. Repeat the assign–update cycle until the dinosaurs stop moving. Take a photograph of the finished layout so you can recreate it digitally.

Try a few variations once you have a stable configuration:

- Reset the dinosaurs in different starting positions. The clusters should settle in similar places even if each dinosaur labels a different group.
- Begin with two dinosaurs very close together and observe how long it takes for them to drift apart.
- Place one dinosaur far from the data to see the "lonely centroid" scenario in action.
- Experiment with four or five dinosaurs if you have enough colours.

5.3 Recreate the Workflow in Orange

Now translate the physical intuition into a digital workflow.

1. Install Orange from `https://orangedatamining.com/download/` if it is not already available on your laptop or lab machine.
2. Open Orange and drag a **Paint Data** widget from the Data palette onto the canvas.
3. Double-click the widget and sketch the configuration from your photograph. Precision is not essential; the aim is to reflect the broad structure of your tabletop clusters.
4. Add the **k-Means** widget from the Unsupervised palette and connect the painted data to it. Fix the number of clusters to match the number of dinosaurs you used.
5. Drop a **Scatter Plot** widget onto the canvas, connect the output of k-Means, open the plot, and colour points by cluster. Compare it with your photograph.

6. Keep the scatter plot open while trying different numbers of clusters in the k-Means widget. Note how the assignments change.

7. Give k-Means a range of cluster counts so it can suggest the value that maximises the silhouette score (a number between -1 and 1 that shows how well-separated each point is from other clusters).

8. Add a **Silhouette Plot**. Connect the output of k-Means to it, and forward the silhouette scores to the scatter plot so everyone can see which points feel confidently placed (near 1) and which sit on the fence (near 0).

9. In the silhouette plot, select a point with a high score and inspect its position in the scatter plot. Repeat for a point with a low score to see whether it sits between clusters.

10. Save the workflow for later reference.

5.4 Explore a Real Dataset

Add the **Datasets** widget from the Data palette and choose the "Grades for English and Math" dataset. Feed it into k-Means and experiment with different numbers of clusters. Use the silhouette score as a guide when deciding how many clusters best summarise the dataset.

5.5 Plan Your Next Steps

Capture any observations from the hands-on run and the Orange workflow while they are fresh. Schedule a follow-up

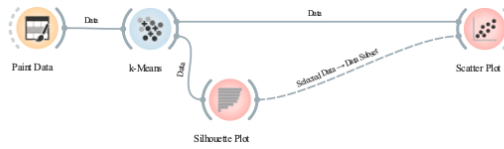


Figure 5.2: Orange workflow screenshot with Paint Data feeding into k-Means, Scatter Plot, and Silhouette Plot widgets, illustrating the digital mirror of the tabletop activity.

session to begin part A of assessment 1 so the ideas from this activity feed directly into your first submission.

Chapter 6

Exploratory Analysis with Orange

Chapter overview

This chapter shows how we prepare, explore, and share data stories inside Orange before introducing sturdier regression tools in the next chapter. We begin with staging workflows so every widget has clean inputs, progress through visual diagnostics that make outliers and subgroups visible, and close with collaboration habits that keep our canvases reproducible. The structure mirrors the Week 2 lectures, preparing us for Hands-on Activity 2 where we load the same housing data into a physical toolkit and practise the diagnostics by hand.

Beginner bridge: What is Orange? Orange is a visual data analysis studio where we connect drag-and-drop modules to build analytical stories. A **widget** is one of those modules: each widget performs a task such as loading data, plotting charts, or training a model. We place widgets on the **canvas**,

the main workspace where they become boxes we can wire together. The wires themselves form the **pipeline**, the flow of data and selections from one widget to the next. Keeping these three terms straight helps us translate between Orange's friendly interface and the statistical ideas we already know.

6.1 Stage Data for Orange Workflows

Lay out a reliable canvas by importing clean tables, confirming data types, and annotating each widget before deeper exploration begins.

Learning objectives

After reading this section, you will be able to:

- configure the **File**, **Select Columns**, and **Data Table** widgets so the entire canvas shares labelled, trustworthy features;
- profile CSV columns to spot missing values, mixed data types, and outliers before wiring analysis widgets;
- document decisions on the canvas so later modelling inherits the exploratory context.

Prerequisites

Before starting, make sure you can:

- interpret CSV headers and recognise common variable types (numeric, categorical, ordinal);
- navigate the Orange interface well enough to add widgets and connect them;

- read simple descriptive statistics such as medians and interquartile ranges.

Beginner bridge: Reading the Orange interface Orange arranges controls around the canvas so we can operate entirely with the keyboard if needed. Press `Ctrl+O` to open datasets, `Ctrl+S` to save workflows, and tap the space bar to open the widget search without reaching for the mouse. Focus moves between widgets with the arrow keys, while `Ctrl+=` and `Ctrl+-` zoom the canvas for low-vision readers. In *Options → Preferences*, the *General* tab lets us enlarge fonts and switch to the high-contrast icon set. We keep these shortcuts visible in our notes so students using screen magnifiers or switch devices can still follow along.

Staging begins by dragging a **File** widget onto a new canvas and pointing it at a structured dataset such as the Sydney housing CSV. We tick the options to detect variable types automatically and to guess special roles (class, meta, or time). The data preview lets us confirm that sale prices, lot sizes, and suburb indicators are recognised correctly before anything flows downstream. When columns misbehave—postcode encoded as text, or distance to the central business district stored as a string—we fix them immediately in **Select Columns**. The widget exposes each feature, allowing us to promote key identifiers to meta fields, drop redundant identifiers, and rename columns so later charts read cleanly.

Next we add a **Data Table** widget to inspect sampled rows. Sorting by price exposes extreme transactions that may warrant closer attention in the robust regression chapter, while the summary bar across the bottom highlights missing values. We annotate the widget (double-click its title) with

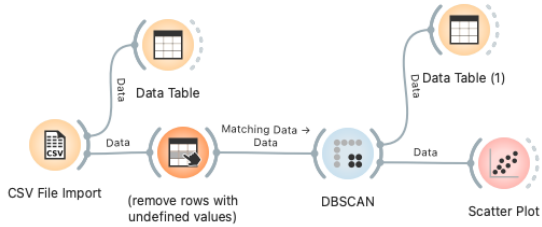


Figure 6.1: Orange staging canvas showing File, Select Columns, and Data Table widgets connected before modelling. Alt-text guidance: describe the left-to-right flow and note the annotation icons so readers using screen readers can recreate the layout.

concise notes such as "postcode promoted to meta" or "distance parsed as numeric". These annotations become the canvas-level documentation others will rely on when revisiting the project weeks later.

Step-by-step: create a reusable staging template.

1. Place **File**, **Select Columns**, **Data Table**, and **Save Data** widgets in a horizontal line.

2. Configure **File** to load the housing CSV with "First row as header" and "Auto detect types" enabled.

3. In **Select Columns**, drag geographic descriptors (suburb, postcode) into meta, retain numerical predictors (price, distance, rooms) as features, and set the target to sale price.

4. Preview the transformed table, filtering for recent sale years to ensure the date parsing succeeded.

5. Use **Save Data** to capture the cleaned subset as an ‘.ows‘ snapshot for future activities.

Try this variation. Swap in the Palmer Penguins dataset from our downloadable practice bundle (`resources/orange-practice-datasets.zip`) and repeat the staging steps. Map species to meta fields, treat flipper length as a numeric feature, and note how Orange automatically recognises the categorical island variable.

Troubleshooting. If the **File** widget shows question marks instead of preview data, check that the CSV path does not include accented characters—older Orange versions prefer ASCII file names. When columns load with the wrong type, toggle “Auto detect types” off, set the correct role manually, and add a note to the widget so collaborators know why the override was necessary.

Accessibility spotlight. Orange exposes built-in accessibility helpers: enable colour-blind-friendly palettes from the palette menu in **Scatter Plot**, turn on “Large icons” under *Preferences*, and use the canvas search (space bar) to add widgets without drag-and-drop. Reminding readers of these features keeps the workflow approachable for everyone.

Global dataset pack. The book’s build process packages the Sydney housing sample, Palmer Penguins observations, and a Singapore resale-flat summary into `orange-practice-datasets.zip`. Offer the bundle alongside the chapter so readers in different regions can substitute a local dataset without reformatting from scratch.

For Advanced Students. Script the same staging steps with Orange's Python Script widget or the command-line interface to maintain versioned preprocessing pipelines alongside the GUI.

Section summary

- A staging strip anchored by **File** and **Select Columns** protects downstream widgets from messy inputs.
- Widget annotations turn the canvas into living documentation that travels with exported '.ows' files.
- Saving cleaned outputs lets us share consistent starting points across lecture examples, practicals, and assignments.

Self-check questions

1. Which Orange widgets help you detect mixed data types before modelling begins?
   *Hint: Think about the staging strip you built before adding visual analytics. Common misconception: Relying on **Scatter Plot** for type issues misses metadata problems. If you answered "Data Table only," consider how **Select Columns** lets you correct types proactively.*
2. How would you document the decision to drop a column so collaborators understand the rationale later?
   Hint: You added notes in more than one place. Common misconception: Commenting in external spreadsheets is enough. Keep the context inside Orange. If you answered "tell the team verbally," revisit the annotation and README practices described above.

6.2 See Patterns Through Interactive Visuals

Use Orange visualisations to surface structure in property data, switching encodings until relationships become obvious.

Learning objectives

After reading this section, you will be able to:

- configure **Scatter Plot**, **Box Plot**, and **Distributions** widgets to compare suburbs, dwelling types, and sale periods;
- layer colour and size channels to expose how distance from the city reshapes price bands;
- compose **Scatter Plot** and **Linear Regression** widgets into a diagnostic loop for identifying nonlinear effects.

Prerequisites

You should be comfortable with:

- reading scatter plots and pair plots;
- interpreting colour gradients and size encodings;
- distinguishing between trend, seasonality, and outliers in time series charts.

We begin by connecting a **Scatter Plot** widget to the staging pipeline. Orange automatically proposes axes; we set lot size on the horizontal axis and sale price on the vertical axis, then enable jitter to separate overlapping points. Colouring by dwelling type reveals that detached houses occupy larger

parcels, with terraces and apartments clustered toward smaller footprints. Switching the colour variable to distance from the central business district exposes a second pattern: outer-ring properties (cool colours) achieve lower price bands than inner-city dwellings (warm colours) even when lot sizes match. Size encoding adds a third dimension—setting marker size to number of bedrooms quickly surfaces the premium on multi-bedroom homes.

To compare suburbs, we drop a **Box Plot** widget and select the suburb field as the grouping variable. Sorting by median price shows which regions command a premium. Hovering over each box lists interquartile ranges and counts, helping us identify suburbs with thin sales volume that might be too noisy for modelling. When prices shift across years, the **Line Plot** widget summarises rolling averages, signalling when structural market changes (for example, the acceleration after 2019) need to be encoded as time-based features.

Visual checklist for every dataset.

1. Plot price versus size, colouring by dwelling type and distance to the city.

2. Add trend lines in **Scatter Plot** to test linear assumptions; if curvature appears, queue the robust regression methods from chapter 7.

3. Review box plots by suburb to detect location-driven variance and to spot candidate grouping variables.

4. Inspect line plots over time to isolate market shifts that may require temporal features or data filtering.

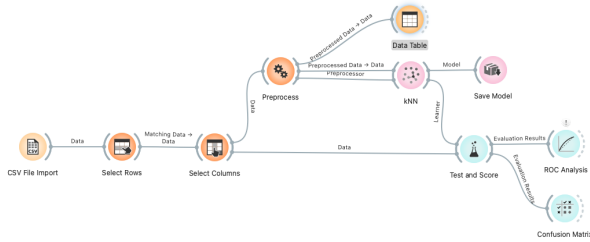


Figure 6.2: Linked visual widgets in Orange highlighting colour encodings and trend overlays for the housing dataset. Alt-text guidance: describe the scatter plot in the centre, the box plot on the right, and the highlighted selection colour so non-visual readers grasp the coordinated views.

Orange's interactivity invites exploration. Double-clicking a cluster in the scatter plot selects those instances everywhere: the **Data Table** highlights them, and the **Box Plot** filters to the same subset. We can therefore drill into "inner-west terraces under $1 million" or "outer suburban apartments post-2019" and immediately inspect their attributes. Tagging interesting selections with the **Save Selected** option lets us persist cohorts for later modelling experiments.

> **For Advanced Students.** Use the **Widget Annotations** add-on to capture hypotheses directly on the canvas—for example, "Consider log-transforming lot size; scatter plot shows diminishing returns." Those notes become the bridge to the feature engineering chapter.

Try this variation. Replace the housing data with our Singapore resale-flat sample or with the Palmer Penguins dataset and colour points by environmental indicators such

as island or conservation status. The same scatter–box plot pairing helps us surface geographic or ecological patterns in non-real-estate contexts.

Troubleshooting. If linked selections stop synchronising, confirm that "Send selections" remains enabled in each widget's control panel. For colour palettes that fail accessibility checks, open the palette menu and switch to the colour-blind-friendly option so hue differences remain legible when printed in grayscale.

Section summary

- Colour, size, and selection tools turn Orange visualisations into living dashboards for hypothesis generation.
- Box and line plots contextualise scatter plots by revealing location- and time-driven effects.
- Saved selections preserve the cohorts discovered during exploration so they can feed modelling experiments.

Self-check questions

1. How does changing the colour attribute in **Scatter Plot** alter your interpretation of price versus size?
   Hint: Compare at least two colour encodings and note what each reveals. Common misconception: Colour is only decorative; in Orange it drives linked selections. If you answered "it does not matter," try swapping to distance bands and re-reading the box plots.
2. Which widget combination would you use to isolate suburbs affected most by a post-2019 price surge?

Hint: Think about how filtering and aggregation interact. Common misconception: A single widget can reveal both time and location effects. If you answered "Line Plot alone," revisit how the box plot supports suburb-level comparisons once the line plot flags a surge.

6.3 Link Exploration to the Broader Workflow

Anchor visual discoveries within the end-to-end checklist so exploration flows naturally into modelling, evaluation, and communication.

Learning objectives

After reading this section, you will be able to:

- map each stage of the data science checklist to specific Orange widgets;
- design canvases that separate exploratory, modelling, and reporting zones for clarity;
- export evidence packs (plots, selections, cleaned data) that make later code-based work more efficient.

Prerequisites

Before continuing, ensure you:

- completed the staging and visual exploration steps earlier in the chapter;
- understand how Orange widgets pass selections and annotations downstream;

- can summarise findings for stakeholders in concise paragraphs.

The checklist begins with **preamble** and **fetch**: documented in the staging strip by noting data provenance and cleaning choices. **Visualise** and **clean** correspond to the scatter, box, and line plots we assembled, plus any feature filters or imputation widgets we add when missing data appears. When we reach **engineer**, we branch the canvas—duplication of the cleaned data line lets us experiment with feature constructors (for example, log-transforming price or calculating price per square metre) without disturbing earlier steps.

Splitting data for modelling is explicit: we chain **Select Columns** → **Training Test Split** → modelling widgets (Linear Regression, k-Means, etc.) and finish with **Test & Score**. Evaluation metrics travel into **Report** widgets, where we attach interpretations describing why one model outran another. Orange's **Rank** widget helps us justify the "choose" stage, while exports to PNG or PDF capture the evidence we will later recreate in Python.

Finally, we treat the canvas as a living notebook. Each branch is labelled with the checklist stage it serves. When new data arrives, we reopen the '.ows' file, refresh the **File** widget, and retest assumptions. Exported selections and cleaned tables feed the hands-on activities and give Python-based workflows a reliable starting point.

> **For Advanced Students.** Pair the Orange canvas with a lightweight README stored alongside the '.ows' file. Summarise the checklist decisions, include screenshots from Figure 6.2 and Figure 6.3, and link to any code notebooks that pick up the

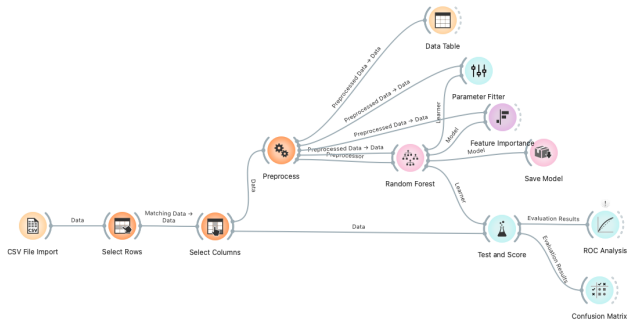


Figure 6.3: A complete Orange canvas segmented into staging, exploration, modelling, and reporting zones following the checklist. Alt-text guidance: narrate the four labelled clusters of widgets and the directional arrows so readers can imagine the branching structure.

analysis.

Global inclusivity. Invite readers to adapt the checklist to local markets by swapping the housing data for regional open datasets such as the U.K. Land Registry, New Zealand's Quotable Value extracts, or World Bank urban indicators. Encourage them to relabel branches in their native language and to note policy-relevant metrics (for example, affordability ratios or energy ratings) that matter in their context.

Try this variation. Branch the staged data into a classification pipeline—add **Logistic Regression** and **Confusion Matrix** widgets to predict whether a listing sells within 30 days, or use the penguin species label to demonstrate non-real-estate classifications. Highlight in the checklist how exploratory cohorts inform the chosen threshold.

Troubleshooting. If **Training Test Split** refuses to run, double-check that the target variable is marked as "class" in **Select Columns**. When exports omit annotations, enable "Embed annotations" in the **Report** widget options so accessibility notes survive in the PDF.

Section summary

- Every checklist stage maps cleanly onto one or more Orange widgets, keeping exploration auditable.
- Segmented canvases clarify which widgets support exploration versus modelling versus reporting.
- Exported evidence packs bridge GUI-based discovery with scripted pipelines.

Self-check questions

1. How would you label the branches of your canvas so collaborators can see which checklist stage they represent? *Hint: Think about both text boxes and colour coding. Common misconception: Saving screenshots alone provides enough context. If you answered "I would explain it during a meeting," consider the asynchronous collaborators who rely on annotations.*
2. Which exports would you bundle with the '.ows' file to help a teammate reproduce your findings in Python? *Hint: List artefacts that capture both data and interpretation. Common misconception: The '.ows' file alone contains the cleaned data. If you answered "just the workflow," revisit why CSVs, reports, and README notes travel together.*

Conclusion and next steps

Our Orange checklist now gives us a repeatable way to stage data, explore visually, and package findings for collaborators. That discipline is essential before we swap in robust regression estimators in the next chapter: the diagnostics we run there depend on well-labelled canvases and transparent exploratory notes. When you switch over to Hands-on Activity 2, use the same checklist to document your Theil–Sen and RANSAC experiments so the physical toolkit mirrors the workflows you just practised on screen.

Chapter 7

Robust Regression in Orange

Chapter overview

Building on the exploratory checklist, this chapter introduces the regression techniques that withstand messy residuals. We first diagnose when ordinary least squares falters, then experiment with resistant estimators such as Theil–Sen, Huber, and RANSAC, and finally show how to report comparisons responsibly. Each section assumes the canvases we set up in the previous chapter are in place, and each prepares us for the hands-on toolkit in Activity 2 where we test these estimators using physical materials.

7.1 Recognise When Baseline Lines Fail

Diagnose when ordinary least squares (OLS) cannot cope with messy data so that sturdier estimators get a chance to shine.

Learning objectives

After reading this section, you will be able to:

- identify leverage points and heteroscedastic variance in residual plots before trusting an OLS fit;
- explain why minimising squared error overemphasises extreme observations;
- prepare Orange canvases that juxtapose baseline and robust regressors for transparent comparison.

Prerequisites

Before diving in, ensure that you:

- understand the algebra behind OLS and can compute residuals;
- have completed the staging workflow from chapter 6;
- can interpret scatter plots with colour encodings, as introduced in the exploratory chapter.

OLS is attractive because it yields closed-form coefficients and easy-to-explain predictions. Yet the method relies on assumptions that rarely hold in open real-estate markets. Outliers—luxury penthouses, redevelopment sites, or data-entry errors—pull the fitted line toward them, distorting predictions for the rest of the data. Non-constant variance (heteroscedasticity) causes similar trouble: if price variation balloons for larger lots, the algorithm treats those noisy ranges as equally trustworthy, again skewing the slope. In Orange, we visualise these issues by pairing a **Linear Regression** widget with **Scatter Plot** and **Residual Plot** widgets. Colouring

residuals by dwelling type or year exposes whether particular subgroups consistently misbehave.

Accessibility spotlight. Enable "Large icons" under *Options → Preferences* before the diagnostics demo so the residual colour legend remains readable. Keyboard users can press `Space` to open the widget search and `Tab` through residual plot controls without a mouse.

Quick residual audit.

1. Fit a baseline model with **Linear Regression** and send predictions into **Predictions** and **Residual Plot**.
2. Colour residuals by year to check whether structural market changes drive systematic errors.
3. Switch to log(price) as the target if variance explodes for high-priced properties.
4. Flag observations whose absolute residuals exceed three standard deviations—their leverage may justify robust alternatives.

Try this variation. Load the Singapore resale-flat dataset from our practice bundle or the Canadian property tax sample from Open Data Toronto. Compare how residual patterns differ when local regulations cap prices or when apartment stock dominates detached housing.

Troubleshooting. If the residual plot appears empty, ensure "Show data" is ticked and that the **Linear Regression** widget has produced predictions. When Orange labels an outlier

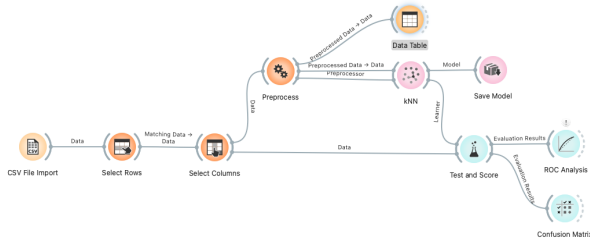


Figure 7.1: Orange canvas comparing linear regression with diagnostic plots that surface outliers and heteroscedasticity. Alt-text guidance: describe the three widget columns and the colour-coded residual plot so readers can trace the diagnostic flow.

as “NaN,” replace missing values via **Impute** before running diagnostics.

Recognising these failure modes primes us for methods that resist them. We keep the baseline line in place for context, but the remainder of the chapter focuses on estimators that reduce the influence of problematic cases rather than pretending they do not exist.

Section summary

- OLS prioritises squared errors, making it vulnerable to high-leverage outliers.
- Residual plots coloured by subgroups reveal whether assumptions break down for specific cohorts.
- Baseline diagnostics guide the decision to introduce robust estimators rather than replacing OLS blindly.

Self-check questions

1. Which visual cues in Orange reveal that variance differs dramatically across the range of prices?
   Hint: Think about both residual plots and annotations. Common misconception: Only scatter plots show variance shifts. If you answered "the main scatter plot," revisit how the residual plot highlights spread changes.

2. How would you explain to a stakeholder why an extreme sale distorts an OLS line?
   Hint: Focus on squared-error sensitivity. Common misconception: Stakeholders only need to know the sale is unusual. If you answered "because it is expensive," expand on leverage and its effect on the slope.

7.2 Adopt Resistant Estimators for Outliers

Deploy Theil–Sen and RANSAC within Orange to keep predictions trustworthy when anomalies dominate the dataset.

Learning objectives

After this section, you will be able to:

- configure **Theil–Sen** and **RANSAC** widgets to run alongside standard regression;
- tune sampling parameters so resistant estimators focus on the majority pattern without overfitting noise;
- interpret coefficient outputs and goodness-of-fit scores to communicate why the robust option wins.

Prerequisites

Make sure you can:

- read Orange's coefficient tables and evaluation metrics;
- understand median-based statistics (for Theil–Sen) and consensus sampling (for RANSAC);
- export '.ows' files and PNG plots to capture evidence.

To add Theil–Sen, branch the cleaned data stream into the **Theil–Sen Regression** widget (available through Orange's add-ons). The widget estimates the slope by taking medians of pairwise slopes, meaning single outliers cannot dominate. We then connect **Predictions** and **Test & Score** to compare root mean squared error (RMSE) with the OLS model. Theil–Sen often produces a slightly less responsive slope but dramatically lower sensitivity to anomalies.

RANSAC (Random Sample Consensus) takes a different approach: it repeatedly samples small subsets, fits a simple model, and keeps the version supported by the largest inlier set. In Orange, we add the **RANSAC Regression** widget, configure the residual threshold (for example, \$75,000) and maximum iterations, and inspect the "inlier ratio" that reports the share of points supporting the chosen line. High inlier ratios indicate the model captures the majority trend while ignoring suspicious sales.

Accessibility spotlight. When tuning parameters, use `Shift+Tab` to revisit earlier fields and the arrow keys to adjust spin boxes precisely. Encourage students to read the "Info" panel aloud so screen-reader users catch the inlier ratio updates.

Workflow recipe for robust comparisons.

1. Duplicate the cleaned data output three times, leading into **Linear Regression**, **Theil–Sen Regression**, and **RANSAC Regression**.

2. Send all three models into a shared **Test & Score** widget with cross-validation enabled.

3. Use **Rank** to order the models by MAE and RMSE, highlighting where robust methods outperform OLS.

4. Visualise predictions from each model in **Scatter Plot**, colouring by the chosen estimator to explain differences to stakeholders.

Try this variation. Swap the response variable to "price per bedroom" or "energy rating" to discuss sustainability-focused metrics, or import Kenya's open land valuation data to show how RANSAC copes with sparse rural transactions.

Troubleshooting. If Theil–Sen fails to appear, confirm that the add-on is installed via *Options* → *Add-ons*. RANSAC occasionally reports "No model found" when too few inliers exist; lower the residual threshold or increase iterations and document the change in your canvas notes.

> **For Advanced Students.** Experiment with **Huber Regression** or quantile regression (available via Python Script) to bridge the gap between least squares and fully resistant estimators when noise levels vary across regions.

Section summary

- Theil–Sen uses medians to dampen the impact of outliers, producing steadier slopes.
- RANSAC iteratively searches for a consensus line, reporting inlier ratios that explain its trustworthiness.
- Parallel evaluation in Orange clarifies when robust methods provide measurable improvements over OLS.

Self-check questions

1. How does RANSAC decide which points to treat as inliers, and how would you adjust the residual threshold in practice?
   Hint: Recall the iteration loop. Common misconception: RANSAC keeps all points by default. If you answered "it picks random points once," revisit how consensus scoring guides the threshold.

2. When might Theil–Sen underperform OLS, and how could you detect that outcome in Orange?
   Hint: Compare evaluation metrics across widgets. Common misconception: Robust estimators are always better. If you answered "never," review the Test & Score table for cases with minimal outliers.

7.3 Test Insights with Real Housing Data

Apply robust lines to the Sydney housing dataset to translate diagnostics into tangible decision support.

Learning objectives

By the end of this section, you will be able to:

- source the Kaggle housing CSV and import it with the staging template from the exploratory chapter;
- engineer features such as price per square metre and distance bands to highlight structural effects;
- narrate why a robust estimator offers more believable forecasts for everyday buyers.

Prerequisites

Confirm that you:

- saved the staged dataset (cleaned types, annotated widgets);
- can use **Feature Constructor** to add engineered variables;
- understand how to filter selections in **Data Table**.

We begin by calculating price per square metre using **Feature Constructor**: 'price / land_size'. Grouping suburbs into inner, middle, and outer rings via **Edit Domain** exposes how location influences both price and variability. Feeding these features into the three regression models yields intuitive comparisons: OLS exaggerates the price impact of land size because it chases premium outliers, while Theil–Sen and RANSAC stay closer to the majority trend. When we slice the data to first-home-buyer price brackets, robust models maintain stable predictions; OLS continues to swing wildly.

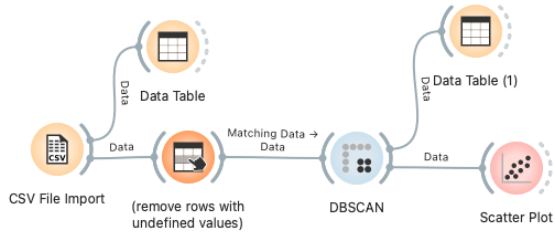


Figure 7.2: Housing workflow with engineered features feeding OLS, Theil–Sen, and RANSAC branches for side-by-side evaluation. Alt-text guidance: outline the duplicated branches and note the icons used for each estimator so collaborators can recreate the layout.

Accessibility spotlight. Use `Ctrl+Plus` to zoom into the canvas before presenting the branched workflow, and toggle "Show channel names" so connectors display textual labels that screen readers can interpret from exported PDFs.

Communicating the results. Export the **Test & Score** table to CSV and summarise the findings in a short memo: "RANSAC achieves a 12

Try this variation. Invite students to replace Sydney data with Lagos rental listings, São Paulo apartment sales, or Wellington rateable values. Discuss how currency, taxation, and housing policies change the engineered features they prioritise.

Troubleshooting. If engineered features display as zero, check for integer division by converting operands to floating-point in **Feature Constructor**. When exports drop non-

ASCII suburb names, save the CSV with UTF-8 encoding and note the step in your README.

Section summary

- Feature engineering amplifies the benefits of robust lines by isolating structural effects such as location bands.
- Robust estimators deliver steadier predictions for everyday buyers, especially when extreme sales are present.
- Exported evaluation tables and plots provide evidence that stakeholders can review without opening Orange.

Self-check questions

1. Which engineered features made the largest difference to the RANSAC fit, and why?
   Hint: Look back at the evaluation tables. Common misconception: Every engineered feature helps equally. If you answered "all of them," identify the metrics that changed the most.

2. How would you present the comparison between OLS and Theil–Sen to a buyer deciding on an offer?
   Hint: Tie the explanation to risk and stability. Common misconception: Buyers only care about the lowest error figure. If you answered "show them the RMSE," include the narrative about volatility reduction.

7.4 Document Outcomes with the Checklist

Tie robust modelling results back to the reusable checklist so the insights survive beyond the initial experiment.

Learning objectives

In this final section, you will learn to:

- capture configuration notes, parameter choices, and evaluation results alongside the '.ows' file;
- plan follow-up experiments that test sensitivity to market shifts or feature updates;
- align the robust regression chapter with the Week 2 hands-on activity.

Prerequisites

You should already:

- understand the checklist stages outlined in the exploratory chapter;
- know how to export PNGs and CSVs from Orange;
- have a shared repository or drive for storing '.ows' artefacts.

Using the checklist headings, annotate the canvas: "Visualise" near the scatter plots, "Engineer" near the feature constructors, "Train" at the regression trio, and "Evaluate" beside **Test & Score**. Add text boxes noting critical parameters such as the RANSAC residual threshold. Save the

workflow with a versioned filename (for example, ‘housing-robust-v1.ows‘) and include exported evaluation tables in the same folder. A short README summarises the results and links to the hands-on activity where students practice similar comparisons.

> **For Advanced Students.** Record parameter sweeps (different RANSAC thresholds or Theil–Sen subsample sizes) in a lightweight experiment log. Over time, the log evolves into a knowledge base for choosing defaults on new datasets.

Try this variation. Create two READMEs: one for real-estate analysts and another for sustainability officers. Compare how the same workflow supports different reporting priorities, and note which checklist items change.

Troubleshooting. If annotations disappear after saving, tick “Include annotations” in the **Save Workflow** dialog. When collaborators cannot open ‘.ows‘ files due to version differences, export a PDF of the canvas and share the dataset bundle so they can rebuild the flow manually.

Section summary

- Checklist labels ensure robust experiments remain reproducible and comprehensible to collaborators.
- Versioned ‘.ows‘ files and exported metrics create a clear audit trail.
- Linking to hands-on activities reinforces the transfer from textbook theory to practical exercises.

Self-check questions

1. What information should your README include to help a teammate rerun the robust regression comparison?
   Hint: Cover configuration, data, and interpretation. Common misconception: A single screenshot suffices. If you answered "just the '.ows' file," revisit the need for datasets and context notes.

2. Which checklist stage will you revisit first when the housing market shifts significantly?
   Hint: Think about where new assumptions enter the workflow. Common misconception: Jump straight to training. If you answered "evaluate," reconsider how staging and engineering respond to new market signals.

Conclusion and next steps

We now know when classic regression breaks, which resistant estimators to reach for, and how to package comparisons that stakeholders can read. Bring these habits into Hands-on Activity 2: the physical toolkit will have you run Theil–Sen and RANSAC by hand, and the clarity you build there depends on the diagnostics and reporting routines we just rehearsed on screen. In the next chapter we pivot to logistic regression, carrying the same discipline for evidence and communication into classification tasks.

Chapter 8

Hands-on Activity 2

We use everyday objects to show why some lines cope better with messy data than others. By stretching thread between numbered tags we can see robust regression techniques in motion before we repeat the same moves in Orange. The activity keeps the language simple while reinforcing the diagnostic habits from the companion chapters.

This activity extends the visual diagnostics from chapter 6 and the resilient line-fitting strategies in chapter 7, letting us rehearse those concepts with physical kits before we mirror them in Orange.

8.1 Kit Checklist

Each group needs a compact kit so they can build physical line-fitting demonstrations before switching to Orange. Assemble the following items per group:

- **Numbered tags (11–89).** A single pack of bee-identification tags provides enough labels for multiple groups. Allocate one strip of the numbers 11–89 to each

group. Order a bulk pack such as the one listed on Amazon Australia (Beekeeping Number Tags).

- **Pair of dice.** Purchase a mixed-colour dice assortment (for example HenMerry Dice Sets). Give each group two standard six-sided dice from the bundle.
- **Thread and scissors.** Supply one compact sewing kit that includes spools of thread plus a small pair of scissors, such as the Kmart Sewing Kit. If the sewing kit does not include scissors, pair it with a set of 130 mm scissors like the Brilliant Basics Stainless Steel Scissors 13 cm.
- **Blu Tack.** Provide a 75 g pack so groups can secure the thread to the table. One option is Bostik Blu Tack Colour 75 g.
- **Comb.** Issue one pocket comb per group, for example from the two-pack sold at Big W (Brilliant Basics Pocket Combs).
- **Ruler.** A 30 cm ruler helps students judge distances and align the thread. Any standard classroom ruler works; one option is the Studymate Binder Mate Recycled Ruler 30 cm.

Keep small resealable bags or trays on hand so each kit stays organised between classes.

International substitutions. If the Australian retailers listed above are unavailable, substitute with locally sourced tags, thread, reusable putty, and rulers. The learning hinges on contrasting sturdy and flimsy lines, not on the specific brands.

8.2 Welcome and Setup

Start with a short briefing so everyone understands the flow of the activity.

1. Reintroduce yourself, especially if the tutorial includes students who missed the first activity.

2. Highlight that “rebound” or revision drop-in sessions commence in Week 3 to recap earlier material for anyone who wants extra practice.

3. Confirm that each group has Orange installed. Make a note of any laptops that still need setup support so you can assist them while circulating.

4. Form groups of around five students. Invite anyone without a team to join an existing group.

5. Demonstrate each stage briefly at the front, then let the groups work while you circulate and answer questions.

8.3 Fit a Line with the Theil–Sen Estimator

Move to a large table so the numbered tags can stand in for data points.

1. Declare the edge of the table the x-axis and the perpendicular direction the y-axis.

2. Arrange the numbered tokens in a rough line with a few clear outliers.

3. Roll the dice twice to select two random tokens (excluding any labels in the 80s, which will serve as a test set).

4. Stretch the thread so it passes through the chosen pair and anchor it with Blu Tack. Extend the line across the dataset.

5. Repeat the random sampling seven to eleven times. Some sampled lines will obviously miss the trend—keep them anyway.

6. Compare the candidate lines and keep the "middle" one (the median slope and intercept). Discuss whether it captures the central pattern of the data.

7. Use the reserved tokens 81–89 as a test set. Measure the vertical distance from each test point to the chosen line, square the distances, sum them, then divide by ten to estimate the mean squared error.

8. Photograph the final layout for later reference.

The full Theil–Sen estimator checks every pair of points; this activity uses a stochastic approximation that captures the main idea with fewer steps.

Remind the group that the Theil–Sen estimator takes the middle (median) slope from many candidate lines, which is why it shrugs off outliers that would yank an ordinary least-squares line off course.

8.4 Fit a Line with RANSAC

Introduce the Random Sample Consensus (RANSAC) approach and reuse the same physical dataset.

1. Thread the line through the centre teeth of the comb so you can slide it along the candidate line.

2. Select a random pair of training points and rebuild the thread line through them.

3. One person guides the comb along the line while another counts how many training tokens the comb passes over (still excluding 80–89).

4. Record the inlier count on paper or a whiteboard. Keep the line if it has the best inlier count so far; otherwise discard it and sample a new pair.

5. After around ten trials you should have a strong candidate. Discuss how the inlier counts changed and when it made sense to stop searching.

6. Estimate the mean squared error on the reserved test tokens and compare it with the Theil–Sen result.

7. Capture another photograph.

Experiment with challenging configurations:

- Increase the proportion of noisy points (for example, split the dataset into two intersecting lines) to show how RANSAC copes with heavy contamination.

- Rearrange the points into a uniform grid to illustrate when Theil–Sen struggles to find a meaningful trend.

8.5 Recreate the Experiment in Orange

Translate the hands-on intuition into an Orange workflow.

1. Enter the (x, y) coordinates manually using the `Paint Data` widget (or another data-entry widget) so the scatter roughly matches the physical layout.
2. Connect the data to a `Scatter Plot` widget and enable the regression line overlay to visualise the default least-squares fit.
3. Use `Select Columns` to declare x as a feature and y as the target.
4. Attach a `Linear Regression` widget and compare its predictions with the original data in a `Predictions` widget feeding a `Data Table`.
5. Plot x versus the model predictions to see the straight-line relationship, then inspect y versus predictions and residuals to diagnose fit quality.
6. Discuss how introducing outliers in the painted data affects the regression and how that compares to the robustness of Theil–Sen and RANSAC.

RANSAC stands for **Random Sample Consensus**. It keeps sampling small point sets, asking, "Do most points agree with this line?" before it accepts a candidate. The consensus idea is what makes it resilient to stray points that do not match the main pattern.

Figure 8.1: Hands-on regression kit with numbered tags, thread, Blu Tack, and comb laid out on a table, ready to demonstrate Theil–Sen and RANSAC line fitting.

8.6 Wrap Up

Encourage students to note the key differences between least squares, Theil–Sen, and RANSAC while the activity is fresh. Prompt each group to record their photographs, Orange workflow, and error calculations in a shared folder or notebook so they can revisit the ideas before the next workshop.

Chapter 9

Logistic Regression and Model Diagnostics

Chapter overview

This chapter transitions us from robust linear modelling into classification territory. We start by showing how overfitting creeps into flexible boundaries, continue with diagnostic tools that expose where our models succeed or fail, and finish by rehearsing communication techniques that make probability-based decisions understandable. The flow mirrors the Week 3 lecture narrative and prepares us for Hands-on Activity 3, where the chessboard game brings log-odds and thresholds to life.

9.1 Warn Against Overfitting

Show how flexible curves can memorise noise so readers reach for safeguards before trusting a classification boundary.

Learning objectives

After reading this section, you will be able to:

- contrast simple logistic curves with over-parameterised polynomials to explain why interpolation between points matters;
- articulate why fitting noise leads to unreliable predictions on unseen cases;
- set up Orange canvases that keep a sensible baseline alongside experimental models.

Prerequisites

Before you begin, make sure you:

- understand polynomial regression and the concept of degrees of freedom;
- can interpret scatter plots with class labels;
- have completed the exploratory analysis chapter for staging data in Orange.

We start by plotting a binary outcome against a single feature using **Scatter Plot**. Overlaying a naive straight line (from **Linear Regression**) and a high-degree polynomial (from **Polynomial Regression**) illustrates the danger of overfitting: the polynomial snakes through every training point yet makes implausible predictions between them. By contrast, logistic regression constrains the relationship to a smooth S-curve, reflecting the reality that probabilities should move gradually rather than oscillate wildly. Keeping these three models on the canvas lets us toggle their predictions

and point to concrete failure modes when the polynomial overshoots.

Steps for a controlled comparison.

1. Connect the staged dataset to **Linear Regression**, **Polynomial Regression**, and **Logistic Regression**.
2. Use **Predictions** to display probability outputs versus class labels in a scatter plot.
3. Switch the polynomial degree from 2 to 6 and observe how the curve oscillates between points.
4. Capture screenshots of each configuration for the experiment log.

Accessibility spotlight. Toggle the "High contrast" icon theme in *Options* → *Preferences* if the colour contrast on probability plots is low, and press `Tab` to cycle through controls inside **Predictions** without a mouse. These small adjustments help screen-reader and keyboard-first users follow the demonstration.

Try this variation. Load the Palmer Penguins dataset and predict whether a penguin belongs to the Gentoo species using bill length. Keep the straight line, polynomial, and logistic widgets connected so you can compare how each model extrapolates beyond the observed measurements.

Troubleshooting. If **Polynomial Regression** refuses to plot, confirm that the target is set to "continuous" in **Select Columns** and that you enabled "Apply automatically" in the

widget settings. When the logistic curve looks flat, reduce the polynomial degree and rescale the feature using **Continuize** to avoid numerical overflow.

Section summary

- Overfitting manifests as wild swings between training points; logistic curves avoid that behaviour.
- Maintaining baselines alongside experiments keeps the canvas grounded in interpretable models.
- Captured screenshots and notes become evidence for why regularisation is necessary.

Self-check questions

1. Which Orange widgets help you demonstrate the difference between interpolation and extrapolation risk?
   *Hint: You used more than one model and a visual comparison. Common misconception: A single predictions table is enough to explain the issue. If you answered "Scatter Plot alone," revisit how **Predictions** overlays curves for side-by-side evaluation.*

2. How would you explain to a stakeholder why a perfectly fit polynomial is a liability?
   Hint: Focus on behaviour between the training points. Common misconception: Stakeholders only care about training accuracy. If you answered "show the R-squared," consider the oscillations you saw between observed cases and how to narrate them.

9.2 Introduce Ridge and Lasso Regularisation

Penalise unnecessary complexity so logistic models remain stable when predictors proliferate.

Learning objectives

After reading this section, you will be able to:

- configure Orange's **Logistic Regression** widget to apply L2 (ridge) or L1 (lasso) penalties;
- interpret coefficient shrinkage and sparsity to explain model behaviour;
- relate penalty choices to neural-network discipline as an analogy for students moving toward deep learning.

Prerequisites

Ensure you can:

- read Orange's coefficient tables and p-values;
- explain the difference between L1 and L2 norms;
- evaluate cross-validation results.

Inside the **Logistic Regression** widget, open the *Regularisation* tab. Select "L2" for ridge regression or "L1" for lasso, then adjust the C parameter (the inverse of regularisation strength). Lower C values apply stronger penalties, shrinking coefficients toward zero. With L1, many coefficients drop exactly to zero, revealing which predictors the model truly needs. Orange's coefficient table updates instantly, allowing

us to narrate how regularisation focuses the model on stable signals while preventing overfitting.

Regularisation sweep.

1. Run cross-validation with C values of 1.0, 0.5, and 0.1 for both ridge and lasso.
2. Compare accuracy, precision, and recall metrics in **Test & Score**.
3. Export the coefficient tables for each configuration and highlight variables that drop out under lasso.

Try this variation. Switch to an international dataset such as the European Credit Approval sample or the open-source South African Heart Disease data, then rerun the ridge versus lasso sweep. Discuss how cultural or regulatory differences change which predictors remain after lasso shrinkage.

Troubleshooting. If **Logistic Regression** greys out the regularisation options, ensure "Use regularisation" is checked and that the solver is set to "liblinear" or "saga". When cross-validation produces "nan" scores, standardise the inputs with **Continuize** or **Feature Scaler** so the optimisation remains stable.

Section summary

- Ridge smooths coefficient magnitudes, while lasso enforces sparsity by zeroing weaker predictors.
- Regularisation strength (C) trades off variance and bias; documenting sweeps clarifies the best compromise.

- Drawing parallels to neural networks helps readers internalise why regularisation is a universal discipline.

Self-check questions

1. How would you decide whether ridge or lasso is more appropriate for a dataset with many correlated predictors? *Hint: Consider both statistical properties and Orange diagnostics. Common misconception: Lasso always wins because it zeros coefficients. If you answered "pick lasso automatically," revisit how ridge handles grouped predictors.*

2. Which evidence from Orange would you include in a report to justify the chosen regularisation strength? *Hint: Think beyond a single metric. Common misconception: Only accuracy matters. If you answered "Test & Score table," remember to attach coefficient exports and annotation notes.*

9.3 Revisit Regression Metrics Before Classifying

Refresh evaluation habits so we detect underfitting and overfitting before moving fully into classification metrics.

Learning objectives

By completing this section, you will be able to:

- compute RMSE, MAE, and R^2 on both training and test folds;

- detect models that look strong on training data but collapse on validation sets;
- explain how regression metrics complement classification diagnostics in Orange.

Prerequisites

Before continuing, ensure you:

- know how to interpret MAE, RMSE, and R^2;
- can configure **Test & Score** for cross-validation;
- have baseline regression models on the canvas for comparison.

In Orange, we feed predictions from both OLS and regularised logistic models (treating probabilities as numeric outputs) into **Test & Score**. Reviewing RMSE and MAE across folds identifies whether models are overfitting before we even inspect classification metrics. A healthy model has similar scores on training and validation splits. If we see a large gap, we revisit regularisation strength or feature selection before moving to confusion matrices.

Metric checklist.

1. Record RMSE and MAE for each model and compare them across training and test splits.
2. Inspect R^2 to ensure the model explains a reasonable fraction of variance without going negative on test data.
3. Annotate the canvas with observations (for example, "L1 C=0.5 keeps RMSE stable across folds").

Try this variation. Use a time-based split—feed the workflow through **Time Series** → **Test on Test Data**—to show how RMSE and MAE behave when predicting future periods. This mirrors markets with structural breaks in cities beyond Sydney.

Troubleshooting. If **Test & Score** shows "Evaluation failed," confirm that the target variable remains numeric. Rerun the widget with "Replicates" set to 1 and "Random seed" fixed to stabilise comparisons when teaching live.

Section summary

- Regression metrics provide early warnings of overfitting before classification diagnostics are computed.
- Consistent performance across folds signals that regularisation choices are appropriate.
- Canvas annotations capture lessons that transfer into hands-on activities and Python notebooks.

Self-check questions

1. Which metric would you prioritise if class imbalance is severe, and why?
   Hint: Connect regression metrics to upcoming classification diagnostics. Common misconception: RMSE is always the right choice, even for imbalanced classification. If you answered "accuracy," recall why MAE can flag uneven errors before accuracy is computed.
2. How do you interpret a negative R^2 on the validation set?

Hint: Compare the model against a horizontal baseline. Common misconception: Negative R^2 means the software is broken. If you answered "it's fine," revisit why it signals worse-than-baseline performance.

9.4 Model Log-Odds with Orange

Encode categories, compute log-odds, and fit logistic curves using Orange's classification widgets.

Learning objectives

After this section, you will be able to:

- map binary outcomes to 0 and 1, creating a target suitable for logistic regression;
- interpret the logistic function that converts log-odds to probabilities;
- configure Orange's logistic widget to output both probabilities and class predictions.

Prerequisites

You should:

- understand odds ratios and their logarithms;
- know how to use **Edit Domain** to convert categorical variables;
- be familiar with Orange's **Logistic Regression** widget.

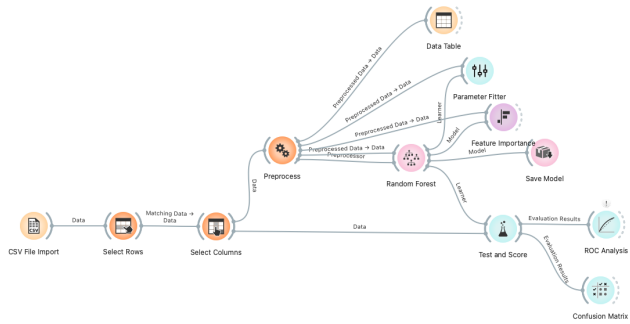


Figure 9.1: Orange classification canvas with data preprocessing, logistic regression, and evaluation widgets wired together. Alt-text guidance: describe the preprocessing widgets on the left, the logistic model in the middle, and the evaluation nodes on the right so readers can rebuild the flow.

We preprocess the data by using **Edit Domain** or **Continuize** to ensure predictors are numeric and normalised. The binary outcome (for example, “drowsy” versus “alert”) is encoded as 1 and 0. Feeding the data into **Logistic Regression**, we enable the option to produce class probabilities. Connecting a **Calibration Plot** (from the **Evaluation** add-on) shows how predicted probabilities align with observed frequencies, reinforcing the concept of log-odds.

Accessibility spotlight. When configuring **Logistic Regression**, press `Alt+Down Arrow` to open drop-down menus and `Shift+Tab` to move back through fields. Enable “Verbose output” to surface textual logs that screen readers can narrate while the chart renders.

Hands-on log-odds exploration.

1. Select a feature (such as caffeine intake) and use **Scatter Plot** to colour points by the binary outcome.

2. Connect **Linear Regression** to approximate log-odds manually, then compare with logistic predictions.

3. Export probability tables and plot them against the sigmoid function to reinforce the concept.

Try this variation. Work with an international public-health dataset, such as predicting vaccination uptake from the WHO's Immunisation data. Showcase how cultural or infrastructure variables shift the log-odds and discuss any ethical considerations before modelling.

Troubleshooting. If the **Calibration Plot** widget reports "No probabilities," double-check that "Output predicted probabilities" is enabled in **Logistic Regression**. For non-binary targets, use **Select Columns** to filter down to two classes or add **One-Hot Encoding** before fitting a multinomial variant.

Section summary

- Proper encoding of outcomes and features ensures the logistic widget behaves predictably.

- Calibration plots confirm whether probabilities align with observed frequencies.

- Visualising log-odds alongside the sigmoid solidifies conceptual understanding.

Self-check questions

1. Why is it essential to encode the binary target as 0 and 1 before fitting logistic regression?
   Hint: Think about the mathematical form of the log-odds. Common misconception: Any text label works because Orange auto-detects. If you answered "for aesthetics," recall how the optimisation relies on numeric targets.

2. How would you use Orange to demonstrate the difference between logits and probabilities?
   *Hint: Pair at least two widgets. Common misconception: Showing the sigmoid equation alone communicates the concept. If you answered "just use **Logistic Regression**," revisit why calibration plots or exported tables help.*

9.5 Explain Cross-Entropy and Gradient Descent

Demystify the optimisation routines that allow logistic models to learn from data.

Learning objectives

After working through this section, you will be able to:

- derive the cross-entropy loss for binary outcomes and explain its penalties;
- describe gradient descent and automatic differentiation in accessible language;
- relate Orange's training process to these underlying mathematical tools.

Prerequisites

Check that you:

- are comfortable with logarithms and derivatives;
- recall the chain rule from calculus;
- can interpret learning curves.

Cross-entropy measures how well our predicted probability distribution matches the true labels. When we assign a high probability to the correct class, the loss is small; confident mistakes incur large penalties. Gradient descent adjusts coefficients step by step to minimise this loss, using derivatives computed automatically. In Orange, we observe these concepts indirectly by monitoring the training log (enable "Show classification statistics") and noting how loss decreases across iterations.

Linking theory to practice.

1. Enable verbose output in the **Logistic Regression** widget to display iteration counts and loss values.
2. Plot loss versus iteration in a spreadsheet or notebook to visualise convergence.
3. Explain how adjusting the learning rate (when exposed) affects convergence speed and stability.

Try this variation. Swap Orange's logistic widget for **Neural Network** with a single hidden layer and compare the loss curves. Highlight how the optimisation story remains the same even when the model architecture changes.

Troubleshooting. If the training log shows "Did not converge," increase the maximum number of iterations or reduce the learning rate. When the log panel is hidden, press `Ctrl+Enter` to apply settings and reopen the diagnostics drawer, which keeps the workflow accessible to keyboard users.

Section summary

- Cross-entropy penalises confident errors, guiding logistic models toward calibrated probabilities.
- Gradient descent iteratively updates coefficients using derivatives of the loss function.
- Orange abstracts these mechanics but still exposes diagnostics that signal healthy training.

Self-check questions

1. What happens to the cross-entropy loss when a model assigns a probability of 0.01 to the true class?
   Hint: Consider the negative log term. Common misconception: Low probabilities always lead to small losses. If you answered "it barely changes," revisit how the logarithm amplifies confident mistakes.

2. How would you explain gradient descent to a stakeholder without using calculus terminology?
   Hint: Reach for everyday analogies. Common misconception: Stakeholders want the derivative definition. If you answered "show the formula," think about the step-by-step hill-descent story we discussed.

9.6 Teach Classification Diagnostics

Evaluate logistic models with accuracy, precision, recall, confusion matrices, and ROC curves.

Learning objectives

By the end of this section, you will be able to:

- compute and interpret accuracy, precision, recall, and F1 score in Orange;
- generate confusion matrices and ROC curves to guide threshold decisions;
- tailor diagnostic reports to different stakeholder priorities.

Prerequisites

Before proceeding, make sure you:

- configured logistic models and regularisation settings from earlier sections;
- know how to use **Confusion Matrix** and **ROC Analysis** widgets;
- can interpret stakeholder requirements (for example, false positive tolerance).

Drag the **Confusion Matrix** widget onto the canvas and connect it to **Test & Score**. Toggle between overall accuracy and per-class precision/recall to identify imbalances. Next, open **ROC Analysis** to view the curve and area under the curve (AUC). By adjusting the classification threshold

slider, we can demonstrate how prioritising recall over precision changes operational behaviour—essential for scenarios like medical screening versus fraud detection.

Accessibility spotlight. Press `Ctrl+Tab` to jump between tabs inside **Confusion Matrix**, and use the "Copy" button to grab the table as text for screen-reader-friendly reports. Orange's ROC widget also exposes a "Contrast" toggle that switches to patterns recognisable for colour-vision deficiencies.

Diagnostic reporting workflow.

1. Export the confusion matrix as a table and annotate the counts with business implications (for example, "false positives trigger manual review").

2. Capture the ROC curve image and note optimal thresholds for different stakeholder preferences.

3. Summarise metrics in a brief report that links back to regularisation choices and regression metrics from earlier sections.

Try this variation. Demonstrate the diagnostics on the Palmer Penguins workflow, classifying whether a penguin nests on Biscoe Island. Highlight how precision–recall priorities shift when the positive class represents a conservation-sensitive habitat.

Troubleshooting. If ROC curves appear as straight lines, verify that the target variable has two classes and that probability outputs are enabled. For extremely imbalanced data, switch to the **PR Curve** widget to provide a clearer story.

Section summary

- Accuracy alone can hide class imbalances; precision and recall tell a more complete story.
- ROC curves reveal threshold trade-offs and support evidence-based decision-making.
- Diagnostics should connect to business context, not just model performance numbers.

Self-check questions

1. Which metric would you emphasise for a model that flags rare but high-impact events, and why?
   Hint: Recall the relationship between prevalence and useful metrics. Common misconception: Accuracy alone captures rare event performance. If you answered "accuracy," revisit why recall or precision can better communicate risk.
2. How does changing the classification threshold affect false positive and false negative rates?
   Hint: Think about sliding the ROC threshold. Common misconception: The trade-off moves both rates in the same direction. If you answered "both decrease," reexamine how prioritising recall increases false positives.

Conclusion and next steps

We now have a disciplined approach to fitting, diagnosing, and explaining logistic regression models. Carry these habits into Hands-on Activity 3: as you move game pieces across

the chessboard, narrate which diagnostics from this chapter you are replicating physically. Looking ahead, the k-NN chapter will contrast probabilistic decision boundaries with neighbour-based reasoning, so keep your notes on thresholds and calibration close—they will help you compare the two approaches on a common footing.

Chapter 10

Hands-on Activity 3

We turn the abstract language of logistic regression into a tactile game. By placing chess pieces on a board and sliding markers along an S-shaped curve, we can feel how probabilities change before we repeat the same moves in software. The activity keeps the vocabulary friendly while showing exactly how the theory chapter's log-odds stories play out.

We carry the log-odds storytelling and diagnostic habits from chapter 9 into this activity so the chessboard and Orange workflows reinforce the same reasoning we practised in the lecture chapter.

10.1 Kit Checklist

Stock each group with the following items so they can run the logistic-regression game on a physical chessboard before moving to Orange. The prices below convert the bulk purchases to a per-group cost using the quantities ordered for the teaching labs.

- **LPG plastic magnetic chess set (20 cm).** Assign

one set to every group (total order: nine sets for the cohort), which works out to roughly $8.53 per group from Big W (LPG Plastic Magnetic Chess Set 20 cm). Keep a spare in case a piece goes missing.

- **A2 logistic S-curve poster.** Print the curve on 80 gsm bond via the Officeworks plan-printing service (Plan Printing). At $3.50 per print (ten ordered for the lab), each group can keep its own poster.

- **A2 gloss laminate.** Protect each poster with the Officeworks gloss laminate option (Laminating Service). Ten laminations covered the class set, so allow $10.10 per group.

- **Kadink Wooden People (8 pack).** These wooden figures are ideal for marking chessboard coordinates while you compute probabilities. Budget one pack per group at $2.98 from Officeworks (Kadink Wooden People 8 Pack).

- **Kadink Wooden Mini Pegs (Natural, 25 pack).** Use the pegs to pin intercept markers onto the S-curve and to track which pieces have already been scored. At $4.00 per pack (ten purchased overall) from Officeworks (Kadink Wooden Mini Pegs 25 Pack), a single pack comfortably serves one group.

- **Whiteboard or A3 pad plus markers.** Reuse existing teaching supplies so groups can record coordinates, coefficients, and loss values.

- **Optional: "ln 2" tokens and string or a ruler.** Any consistent counters help students visualise additive

changes in log-odds. Reuse tokens from other activities or cut strips of paper if you need a quick substitute.

International substitutions. If these Australian suppliers are out of reach, use any portable chessboard (or a printed grid), simple game pieces, and a hand-drawn logistic curve. The learning hinges on contrasting colours and plotting points, not on the exact brand of board or pegs.

10.2 Stage 1: Build the Chessboard Dataset

Gather each group around a table with the chess set, the laminated logistic curve, wooden people, pegs, and a whiteboard marker.

1. Treat the chessboard as a coordinate grid. Files (columns) are x and ranks (rows) are y, both ranging from 0 to 8. Place five mixed-colour pieces on distinct corners of squares; duplicates are fine as long as you track which is which.

2. Label white pieces as $y = 1$ (positive class) and black pieces as $y = 0$ (negative class). Encourage groups to include one point near $(0, 0)$ and another along the same file so the intercept and slope feel concrete.

3. Record each piece's coordinates on the whiteboard. When students move a piece, update the list so the dataset stays consistent.

4. Park the twin of every chess piece (for example the unused knight) on the logistic curve. Keep the mini pegs

handy to mark the intercept position on the curve.

10.3 Stage 2: Initialise a Logistic Model

Give each group a shared starting point so comparisons make sense.

- Initialise the intercept at 0 with both coefficients set to 0.1.
- Write the working formula $s = \beta_0 + \beta_{\text{rank}} \times \text{rank} + \beta_{\text{file}} \times$ file on the whiteboard. Remind everyone that s is in log-odds—a scale where we add numbers, then convert them to probabilities using the S-shaped logistic curve.
- Peg the intercept onto the curve and place each twin piece at the probability it implies (white pieces towards the top, black pieces towards the bottom).

Talk through the interpretation: a $+1$ step in log-odds multiplies the odds by e^1, whereas a $+\ln 2$ step doubles them. Log-odds simply means "how tilted are the chances in favour of the positive class"—positive numbers favour white pieces, negatives favour black ones. Use the optional tokens to emphasise these increments physically.

10.4 Stage 3: Calculate Likelihood by Hand

Walk the groups through the loss calculation so they can judge improvements.

1. For each chess piece, substitute its coordinates into the linear predictor to compute s.

2. Read the corresponding probability p from the logistic curve. For white pieces keep p; for black pieces use $1 - p$.

3. Multiply the five contributions together to obtain the joint likelihood that the model classifies every piece correctly.

4. Record the product on the whiteboard. Challenge groups to beat their previous best as they refine the model.

Encourage students to narrate why the probability moves the way it does: "we add on the log-odds scale, so sliding one token by $+\ln 2$ doubles the odds."

10.5 Stage 4: Greedy Coordinate Search

Let each group iterate on the coefficients to improve the likelihood.

1. Take the pieces off the logistic curve but leave them on the board.

2. Propose a small tweak to one parameter at a time (for example increase the intercept by 0.1 or adjust the rank coefficient by 0.05).

3. Recompute s and the likelihood. If the product increases, keep the change; otherwise revert and try a different direction.

4. Repeat until the improvements taper off. Record the final parameters so everyone can compare results later.

During debrief, ask groups what sequences of changes helped most and whether any coefficients surprised them.

10.6 Stage 5: Evaluate the Classifier

Once the group settles on parameters, they can translate the model back into familiar metrics.

1. Use the model to label each chess piece as white (probability > 0.5) or black (probability ≤ 0.5). Tally true positives, false positives, true negatives, and false negatives on the whiteboard.

2. Compute accuracy, precision, and recall from the tallies. Discuss how moving the threshold off 0.5 would change the counts.

3. Stretch a piece of string across the chessboard where $\beta_0 + \beta_{\text{file}} \times x + \beta_{\text{rank}} \times y = 0$. This visual decision boundary helps explain how the model separates the two colours.

10.7 Stage 6: Explore Variations

Give the class time to experiment beyond the base scenario.

- **Separable layout.** Cluster the white pieces in one quadrant and the black pieces elsewhere. Observe how the coefficients keep growing unless you cap them, highlighting why regularisation exists.

- **Ring layout.** Surround the white pieces with black pieces to demonstrate a non-linearly separable case. Prompt students to suggest extra features (for example x^2, y^2, or interaction terms) that would bend the boundary.

- **Train/test split.** Reserve a couple of pieces as a hold-out set and see how well the tuned model generalises.
- **Gradient descent in Orange.** Swap in Orange's Gradient Descent widget to watch the optimisation unfold in real time.

10.8 Stage 7: Rebuild the Workflow in Orange

Finish by connecting the physical intuition with a digital workflow.

1. Enter the chessboard coordinates into Orange using **Create Table** (from the Educational add-on) or a small CSV file loaded with a **File** widget.
2. Use **Select Columns** to designate the colour as the target and the coordinates as features. Feed the data into a **Logistic Regression** widget.
3. Inspect the coefficients, log loss, accuracy, precision, and recall in a **Data Table**. Compare them with the hand-calculated results.
4. Send the model to a **Predictions** widget, then on to a **Confusion Matrix** and **Scatter Plot**. Link the confusion matrix back to the scatter plot to highlight correctly and incorrectly classified points.
5. Optionally, replace the logistic regression widget with the **Gradient Descent** widget to observe how the parameters converge.

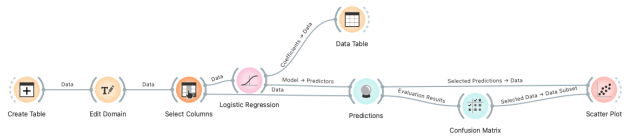


Figure 10.1: Orange workflow showing Create Table feeding Logistic Regression, Predictions, Confusion Matrix, and Scatter Plot widgets to mirror the chessboard exercise digitally.

Capture photographs of the physical setup alongside screenshots of the Orange workflow so students can revisit the reasoning before the next activity.

Chapter 11

Understand and Tune k-NN

Chapter overview

The nearest-neighbour family gives us a complementary perspective to logistic regression. We begin by anchoring the voting intuition, move through distance metrics and scaling choices, and finish by tuning hyperparameters with evaluation safeguards.

These sections set up the NSW land value and Red Rooster case studies that follow, and they foreshadow the comparisons you will make in Hands-on Activity 5 when we place neighbour and logistic boundaries side by side.

11.1 Voting Intuition and Decision Boundaries

Introduce the voting intuition, how the decision boundary responds to different k, and the practical dials we can adjust.

Learning objectives

After reading this section, you will be able to:

- describe k-NN as a majority-voting classifier that assigns new examples the label held by nearby training points;
- explain how varying k alters decision boundary smoothness and the bias–variance trade-off;
- sketch decision regions in Orange to support the land-value case study in chapter 12.

Prerequisites

Before you begin, make sure you can:

- read Orange scatter plots that colour-code class labels;
- interpret confusion matrices and class proportions;
- distinguish between training and evaluation data splits.

k-NN treats every labelled example as a voting neighbour. When a new parcel or customer arrives, we measure its distance to the existing records and tally the labels of the closest k points.

The majority outcome becomes the prediction, so the model's behaviour lives entirely in the geometry of the training set rather than in fitted coefficients.

Visualising the decision boundary—the curve that separates predicted classes—helps us see how the vote changes as we move through feature space. With $k = 1$, the boundary hugs each observed point, weaving jagged islands around every example.

Raising k makes the regions smoother by insisting that a larger neighbourhood agree before we switch classes, trading local sensitivity for stability.

The Orange workflow mirrors that logic. Connect a staged dataset to **Scatter Plot** and toggle the "Show decision boundary" option. Slide the k parameter in the linked **k Nearest Neighbours** widget between 1, 3, 5, and 15 to watch the coloured regions inflate or contract.

Capture screenshots with accessible captions (such as "Decision boundary at $k = 5$") so materials remain helpful to screen-reader users.

Accessibility spotlight. Press `Ctrl+F1` in Orange to open widget help without leaving the keyboard, and use `Tab` to step through parameter fields.

These shortcuts keep the demonstration inclusive for students who rely on keyboard navigation.

Try this variation. Load the Palmer Penguins dataset from the practice bundle (`resources/`). Plot bill length against flipper length to see how the Gentoo decision region expands as you increase k.

Troubleshooting. If the decision boundary looks noisy even for high k, confirm that the target variable is categorical in **Select Columns**. Numeric targets trigger regression mode, which draws a continuous surface instead of class regions.

Section summary

- k-NN predicts labels by letting nearby observations vote on the outcome.

- Small k values create flexible but fragile boundaries; larger k smooths away noise.
- Visual boundary checks in Orange prepare us for the land-value workflow in chapter 12.

Self-check questions

1. How would you explain the trade-off between $k = 1$ and $k = 15$ to a stakeholder?
   Hint: Focus on how much of the neighbourhood must agree before the vote flips.
   Common misconception: Higher k always improves performance.
   If you answered "pick the largest k," revisit how overly smooth boundaries lose accuracy.
2. Which Orange widgets help you visualise decision regions while adjusting k?
   Hint: Think about the combination of model and plot.
   *Common misconception: The **Scatter Plot** works alone.*
   *If you answered "just Scatter Plot," remember that the **k-NN** widget must feed it a model first.*

Learning objectives

After reading this section, you will be able to:

- configure Orange's k-NN widget for classification or regression scenarios;
- compare uniform and distance-weighted voting schemes;
- normalise features so distance metrics treat each scale fairly.

Prerequisites

Ensure you can:

- interpret Orange's **Feature Statistics** output;
- compute simple min–max and z-score transformations;
- explain the difference between Euclidean and Manhattan distance.

k-NN adapts to multiple task types. Switching the widget to "Regression" mode averages the neighbour targets instead of voting. This behaviour supports our NSW land-value estimate: the predicted price per square metre becomes the mean of the surrounding parcels, optionally weighted by distance when nearby blocks should influence the result more strongly than far-flung ones.

Distance weighting also matters for classification. Uniform weighting treats every neighbour equally, whereas inverse-distance weighting gives closer observations a louder voice, preventing fringe cases from dominating the vote.

We must also choose an appropriate distance metric. Euclidean distance measures straight-line separation, suiting continuous coordinates, while Manhattan distance sums absolute differences along each axis. Regardless of the metric, unscaled features can derail the model: a single wide-ranging attribute such as "land area" can swamp others like "zoning code." Use **Continuize** or **Normalize** to standardise the inputs before fitting. Document your choices in the experiment log so readers of chapter 13 can reproduce the comparisons.

Workflow checklist.

1. Inspect feature ranges with **Feature Statistics** and note any dominant scales.

2. Apply **Normalize** (z-score) or **Continuize** (min–max) so no feature dominates distance calculations.

3. Configure k **Nearest Neighbours** with the chosen metric, weighting, and k value.

4. Record the configuration and rationale in your Orange annotations.

Try this variation. Experiment with Manhattan distance on the Red Rooster case study to see how axis-aligned neighbourhoods reshape the decision boundary.

Troubleshooting. If Orange reports "No numeric features available," check that categorical attributes were one-hot encoded via **Continuize**. For regression, confirm that the target remains numeric; accidental type conversion will hide it from the widget.

Section summary

- k-NN can vote for classes or average regression targets depending on the widget mode.

- Distance weighting and metric selection tailor the neighbourhood to the domain.

- Normalising features keeps the distance calculation balanced across scales.

Self-check questions

1. When would you prefer inverse-distance weighting over uniform voting?
   Hint: Think about how similar the nearby cases are.
   Common misconception: Weighting only matters for regression.
   If you answered "never," revisit the geospatial parcel example where distant suburbs distorted votes.

2. How do you prepare categorical zoning codes for a k-NN model?
   Hint: Consider the encoding step.
   Common misconception: Leave them as strings.
   If you answered "k-NN can read text directly," remember that distance metrics need numbers.

11.2 Operational Trade-offs

Surface implementation realities: storage, privacy, latency, and deterministic outcomes.

Learning objectives

After reading this section, you will be able to:

- articulate the cost of storing full training sets for k-NN deployments;
- design tie-breaking policies that keep predictions reproducible;
- plan query-time optimisation strategies for latency-sensitive products.

Prerequisites

Before continuing, ensure you can:

- estimate storage requirements for medium-sized tabular datasets;
- describe basic indexing structures such as KD-trees or ball trees;
- communicate privacy obligations for customer data.

Deploying k-NN means shipping the entire training dataset, because prediction requires comparing each new query against stored examples. That obligation complicates privacy compliance: sensitive attributes remain at rest, so we must encrypt disks, manage retention policies, and justify why keeping raw records is necessary. Storage also affects latency. Without indexing, each prediction scans every row, slowing down mobile experiences. Pre-computing spatial indices or caching frequent queries helps the model respond within practical limits.

Determinism deserves equal attention. When two classes receive the same number of votes, a tie-break rule chooses the winner. Document whether you default to the majority class, the closest neighbour, or a weighted decision so stakeholders can reproduce the result. Tie policies matter in civic applications like valuation, where consistent outputs build trust.

Workflow checklist.

1. Estimate memory usage by multiplying the number of records by the feature count and storage size.

2. Choose an indexing strategy (KD-tree, ball tree) if strict latency bounds apply.

3. Define and document tie-break rules in the project README.

4. Review privacy controls with stakeholders before deploying beyond experimentation.

Troubleshooting. If Orange predictions feel slow on large datasets, reduce k temporarily or sample the training set while you evaluate feasibility. For production builds, move the workflow into Python with `KNeighborsClassifier` from `sklearn.neighbors` so you can take advantage of compiled indices.

Section summary

- k-NN stores the training set, so privacy and latency must be planned up front.

- Tie-breaking policies keep outcomes reproducible for stakeholders.

- Query-time optimisation ensures the method remains viable in real products.

Self-check questions

1. What documentation should accompany a deployed k-NN model to explain how ties are resolved?
   Hint: Consider reproducibility and stakeholder trust.
   Common misconception: Ties are so rare that no policy is needed.

If you answered "none," revisit how evenly split parcels in the land-value dataset force explicit choices.

2. How would you address a product manager's concern that storing the training set breaches privacy agreements?
   Hint: Think about anonymisation, encryption, and retention limits.
   Common misconception: Deleting identifiers is sufficient.
   If you answered "strip the IDs," remember that the attribute mix can still identify households, so governance must be broader.

Conclusion and next steps

You now have a working playbook for neighbour-based modelling: understand the voting intuition, normalise distances thoughtfully, and validate hyperparameters with care. As we turn to the NSW land value and Red Rooster case studies, keep your tuned settings close—they will shape how you interpret regional price gradients and restaurant site predictions. When Hands-on Activity 5 arrives, you will compare these neighbour models with logistic baselines, so bring your notes on bias–variance trade-offs and validation strategy to that session.

Chapter 12

Predicting NSW Land Value in Orange

Chapter overview

This case study applies the k-NN principles to a real regional dataset. We explore the valuation records, engineer features that respect spatial context, and evaluate neighbour regressors alongside baselines. Each stage echoes the lecture narrative and prepares us to compare land value predictions with the restaurant classification story and the Hands-on Activity 5 exercises that follow.

12.1 Understand the Valuation Data

Walk readers through the tax-oriented valuations, derived features, and spatial quirks of the dataset.

Learning objectives

After reading this section, you will be able to:

- summarise unimproved land value records and the variables captured by the NSW government dataset;
- derive price-per-square-metre features that make neighbouring parcels comparable;
- identify missing or unreliable valuations that require cautious interpretation.

Prerequisites

Before diving in, make sure you can:

- interpret government valuation metadata and zoning codes;
- calculate ratios and unit conversions for tabular data;
- navigate Orange's **File** and **Select Columns** widgets.

The NSW Valuer General releases "unimproved land value" assessments that capture what a parcel would sell for without considering buildings. Each record includes identifiers, geographic coordinates, lot sizes, and recent valuation amounts. We start by reading the CSV into Orange and checking the schema: lot area sits in square metres, while valuations appear as whole-dollar amounts tied to specific reporting years. Computing a price-per-square-metre feature by dividing the valuation by lot area lets us compare irregular parcels on a common footing.[1]

Bushland and recreational reserves often lack reliable valuations, either because they are exempt from rating or their usage makes open-market comparisons tricky. Our workflow

[1]Week 5 lecture slides 79–109 frame the case study as a regression problem over geolocated property data.

focuses on predicting an unseen reserve's value by leveraging nearby residential and mixed-use parcels. Highlight the "missing valuation" rows in **Data Table** and note which suburbs cluster around the gap so we can validate predictions later.

Accessibility spotlight. When sharing screenshots of the data table, include alt-text style captions that highlight the relevant columns (for example, "Rows filtered to show missing valuations with latitude and longitude retained"). This keeps the artefacts accessible when the images are embedded in chapter 11 and related activities.

Section summary

- NSW valuation data records unimproved land values alongside lot size and coordinates.
- Deriving price-per-square-metre normalises parcels of different sizes.
- Missing valuations motivate predicting a bushland reserve using nearby parcels.

Self-check questions

1. Which attributes do you need to compute price per square metre, and why is this ratio helpful?
   Hint: Think about comparing parcels of different sizes.
2. How would you document the parts of the dataset that lack valuations?
   Hint: Consider annotations in Orange and comments in your lab notes.

Learning objectives

After reading this section, you will be able to:

- build the baseline Orange workflow that feeds staged valuations into a k-NN regressor;
- create a synthetic query instance without contaminating the training data;
- log intermediate checks so future readers can reproduce the canvas.

Prerequisites

Ensure you can:

- use **Select Columns** to mark targets and metas;
- operate Orange's **Create Instance** widget;
- interpret Orange annotations and save workflows for later reuse.

Start with **File** to load the valuation CSV, then pass it to **Select Columns**. Mark "price_per_sqm" (the derived feature) as the target, keep latitude and longitude as features, and treat identifiers as meta fields. Add **Feature Statistics** to confirm that ranges look sensible and to catch any zero-valued lots that might distort averages.

Next, connect **Create Instance**. Enter the coordinates of the bushland parcel we want to value, along with its lot size if available. Disable "Append instance to data" so the query remains separate. Finally, attach k **Nearest Neighbours** configured for regression with distance weighting, and route both the original data and the query into **Predictions**. Save

the workflow in the course repository alongside notes referencing chapter 11 so teams can revisit the configuration during the hands-on session.

Workflow checklist.

1. Load the staged CSV via **File** and inspect types in **Select Columns**.

2. Derive price per square metre (if not precomputed) using **Feature Constructor**.

3. Configure **Create Instance** with the query parcel and disable appending.

4. Connect k **Nearest Neighbours** in regression mode with inverse-distance weighting.

5. Send outputs to **Predictions** and **Data Table** for inspection.

Troubleshooting. If **Create Instance** refuses to generate a prediction, check that you supplied every feature the model expects. Missing coordinates will trigger "No data" warnings until the field is filled.

Try this variation. Duplicate the workflow and replace k-NN with **Random Forest**. Compare the predicted valuation and explain when a tree-based model might outperform distance-weighted averages.

Section summary

- The Orange canvas combines **File**, **Select Columns**, **Create Instance**, and k **Nearest Neighbours** for valuation.
- Disabling instance appending keeps the query parcel out of the training data.
- Annotated workflows document the modelling choices for future labs.

Self-check questions

1. Why do we disable "Append instance to data" when valuing the bushland parcel?
   Hint: Think about data leakage.
2. Which widget confirms that the feature ranges look sensible before modelling?
   Hint: Recall the diagnostic you inserted after ***Select Columns****.*

12.2 Interrogate the Prediction

Encourage validation on known parcels, sensitivity analysis, and sanity checks before trusting a single estimate.

Learning objectives

After reading this section, you will be able to:

- validate the model by predicting on held-out parcels with known valuations;

- assess sensitivity to different k values and distance weightings;
- document follow-up checks for geocoding accuracy and policy caps.

Prerequisites

Before continuing, make sure you can:

- perform train/test splits in Orange;
- read **Test & Score** reports for regression metrics;
- investigate geospatial artefacts such as duplicated coordinates.

Hold back a subset of parcels with known valuations and feed them through **Test & Score** alongside the query instance. Comparing root-mean-square error (RMSE) and mean absolute error (MAE) across different k values helps us understand whether the model over-smooths neighbourhood effects. If increasing k inflates RMSE, we know that our bushland prediction might be similarly dampened.

Sensitivity analysis continues the story. Duplicate the k **Nearest Neighbours** widget with $k = 3$, $k = 7$, and $k = 15$, then log how the bushland estimate changes. Large swings suggest the area lacks dense comparables, hinting that we should collect more local sales data or consult planners for context. Finally, audit the geocoding: plot the training and query points on **Geo Map** or export coordinates for a quick check in a GIS tool. Regional policy caps, such as limits on waterfront valuations, can flatten predictions; note these constraints in the commentary so stakeholders understand when manual review is required.

Troubleshooting. If **Test & Score** shows "NaN" metrics, ensure that every evaluation fold contains at least one example—sparse rural areas may require stratified sampling or manual grouping before splitting.

Try this variation. Evaluate the workflow using the Singapore resale-flat sample from the practice bundle. Discuss how the valuation features differ and which preprocessing steps must change for international datasets.

Document the evidence. Capture screenshots of the evaluation metrics, sensitivity table, and map view. Store them with descriptive filenames (for example, `nsw-valuation-k3-rmse.png`) so future cohorts can trace the reasoning behind the recommended k.

Section summary

- Validating on known parcels builds trust before predicting new valuations.
- Sensitivity checks across k values reveal how stable the bushland estimate is.
- Geocoding audits and policy notes contextualise the numerical output.

Self-check questions

1. How do you decide whether the chosen k produces an acceptable valuation error?
   Hint: Compare RMSE and MAE across multiple settings.

2. What additional evidence would you gather if the prediction changed dramatically between $k = 3$ and $k = 15$? *Hint: Think about map inspections and stakeholder input.*

Conclusion and next steps

By grounding k-NN in NSW valuation data we have practised feature engineering, spatial reasoning, and evaluation storytelling that align with real policy questions. Keep the annotated canvas handy as we turn to the Red Rooster site-selection narrative—the contrasts between regression and classification will sharpen your evaluation instincts. Those instincts matter again in Hands-on Activity 5, where you will articulate how these neighbour models stack up against logistic baselines in a token-based comparison.

Chapter 13

Red Rooster Line: Choosing the Right Classifier

Chapter overview

This chapter extends the k-NN discussion by contrasting it with logistic regression on a geospatial classification task. We set up the Red Rooster line problem, compare decision boundaries, and reflect on operational considerations when rolling out the better-performing model. The sections intentionally mirror the lecture narrative and tee up the evaluation guardrails and Hands-on Activity 5 comparison that follow.

13.1 Frame the Geospatial Challenge

Use the chicken-shop case study to contrast logistic regression with k-NN, showing how geography influences algorithm choice.

Learning objectives

After reading this section, you will be able to:

- describe the "Red Rooster line" as a supervised classification problem over latitude and longitude;
- prepare the fast-food dataset by filtering the relevant chains and constructing training/test splits;
- articulate why regional imbalance matters when comparing classifiers.

Prerequisites

Before starting, ensure you can:

- stage datasets in Orange using **File** and **Select Columns**;
- interpret class imbalance metrics such as prevalence and lift;
- explain logistic regression and k-NN basics from chapter 11.

The folklore surrounding the "Red Rooster line" suggests that one fast-food chain dominates north of Sydney while another prevails south of a notional boundary. We model this story as a binary classification problem: each store location has latitude, longitude, and a label indicating the franchise. The first step is to filter the dataset so only the chains of interest remain, preserving store counts for both classes. Record the prevalence—the proportion of stores belonging to each chain—because imbalance shapes how we judge model performance.

Split the data into training and testing subsets using **Data Sampler** or **Data Sampler (Fixed)** to keep evaluation consistent. Tag the splits in your notes so you can reproduce them when iterating on parameters. This disciplined setup paves the way for the side-by-side comparison later in the chapter.

Accessibility spotlight. When sharing the scatter plot of store locations, annotate the caption with compass directions (for example, "Map of Sydney showing Red Rooster stores north of the harbour and KFC stores in the south") to support readers who rely on screen-reader descriptions.

Section summary

- The Red Rooster line becomes a latitude/longitude classification task with two chains.
- Filtering and documenting class prevalence prepares the data for fair comparisons.
- Consistent train/test splits make subsequent evaluation trustworthy.

Self-check questions

1. How does class imbalance influence your expectations for accuracy in this case study?
   Hint: Consider what happens if one chain heavily outnumbers the other.
2. Which Orange widgets help you capture reproducible train/test splits?
   Hint: Think about sampling tools.

Learning objectives

After reading this section, you will be able to:

- build an Orange canvas that evaluates 1-NN, 3-NN, and logistic regression side by side;
- diagnose when logistic regression oversimplifies the decision boundary;
- prepare the feature space (for example by scaling longitude) before comparing metrics.

Prerequisites

Ensure you can:

- interpret confusion matrices and ROC curves;
- scale features using **Normalize** or **Continuize**;
- route models into **Test & Score** and **Confusion Matrix** widgets.

Start by normalising the coordinates. Longitude spans a larger range than latitude in this dataset, so applying z-score scaling prevents east–west variation from dominating the distance metric. Feed the scaled data into three models: k **Nearest Neighbours** with $k = 1$ and $k = 3$, plus **Logistic Regression**. Connect each to **Test & Score** and **Confusion Matrix** so you can inspect accuracy, F1, ROC AUC, and misclassification patterns.

As you explore the results, examine the scatter plot with decision regions enabled. Logistic regression draws a single linear boundary that sometimes defaults to predicting the dominant chain. When the real boundary snakes along the

harbour, k-NN retains the regional quirks by letting local neighbours steer the prediction. Capture screenshots of each model's boundary, labelling them clearly for learners reviewing chapter 11.

Try this variation. Swap the target to "Is the store within 1 km of a highway?" and compare how the models behave when the boundary becomes less linear. This encourages readers to match model flexibility to spatial structure.

Troubleshooting. If logistic regression outputs identical probabilities for every point, check that feature scaling applied correctly and that the solver converged (look for warnings in the widget). For k-NN, confirm that you reconnected the scaled data rather than the raw coordinates.

Section summary

- Scaling coordinates keeps logistic regression and k-NN comparisons fair.
- Logistic regression provides a smooth baseline, while k-NN captures local variation.
- Visualising decision regions alongside metrics grounds the model choice in evidence.

Self-check questions

1. Which metrics beyond accuracy influenced your choice between logistic regression and k-NN?
   Hint: Think about ROC AUC, F1, or MCC.

2. How does scaling longitude alter the confusion matrices? *Hint: Compare the false positive rates before and after scaling.*

13.2 Interpret the Results

Encourage metric-driven discussion of accuracy swings and the final model choice.

Learning objectives

After reading this section, you will be able to:

- evaluate variance in model accuracy using repeated sampling or cross-validation;
- communicate findings using metrics that resonate with stakeholders;
- justify the final model choice with evidence collected during the comparison.

Prerequisites

Before proceeding, make sure you can:

- run **Test & Score** with cross-validation or repeated sampling;
- interpret confidence intervals for accuracy and ROC AUC;
- present findings in meeting-ready notes or dashboards.

Use repeated cross-validation in **Test & Score** to estimate variability. Record the mean and standard deviation for accuracy and ROC AUC across folds. If logistic regression shows low variance but systematically underperforms k-NN, you can argue that its stability does not compensate for the accuracy gap.

Translate the findings into stakeholder-friendly language. Highlight metrics that connect to business questions: F1 captures balance between false positives and negatives, ROC AUC summarises ranking quality, and Matthews correlation coefficient (MCC) handles imbalance gracefully. Present the numbers with context, noting that a 3-NN model improves accuracy by a measurable margin while keeping interpretation grounded in the local geography.

Finally, recommend the model with a concise justification. For example: "We propose 3-NN because it increases ROC AUC by 0.07 relative to logistic regression and better respects the harbour boundary, as shown in Figure X." Include links to the Orange workflow and to the practice bundle created via `make dataset-bundle` so readers can replicate the analysis.

Document the decision. Add a short write-up to your lab log summarising the evaluation settings, chosen metrics, and final recommendation. This practice mirrors the reporting expectations in chapter 12 and the Week 5 hands-on activity.

Troubleshooting. If cross-validation results vary wildly between runs, ensure that the random seed in **Data Sampler (Fixed)** matches across experiments. Consistent seeding keeps comparisons fair.

Section summary

- Repeated sampling reveals whether observed accuracy differences are stable.
- Communicating metrics in plain language helps stakeholders follow the argument.
- The 3-NN model often wins because it respects geographic nuances that logistic regression flattens.

Self-check questions

1. Which evidence would you include in a one-page summary recommending 3-NN over logistic regression?
   Hint: Combine metrics, visuals, and workflow references.
2. How do you reassure stakeholders that the evaluation will hold if the dataset grows?
   Hint: Discuss cross-validation, monitoring, and periodic retraining.

Conclusion and next steps

The Red Rooster case sharpened our ability to compare classifiers in context, balancing accuracy with geographic nuance and operational constraints. As we move into the evaluation guard-rails chapter, carry forward the notes you captured on threshold choices, refresh schedules, and stakeholder messaging. They will help you build the evidence pack needed for Hands-on Activity 5, where these classifier comparisons become tangible in the token-based debate.

Chapter 14

Evaluate Models with Guard Rails and Baselines

Chapter overview

With k-NN and logistic classifiers fresh in mind, this chapter consolidates our evaluation discipline. We stabilise metrics with cross-validation, compare models against meaningful baselines, and design monitoring plans that keep deployed systems honest. These guard rails directly support the land value and Red Rooster case studies and lead into Hands-on Activity 5, where we debate model choices using the same evidence.

14.1 Stabilise Metrics with Cross-Validation

Detail validation schemes that temper noisy train/test splits and inform hyperparameter choices.

Learning objectives

After reading this section, you will be able to:

- configure Orange's **Test & Score** widget for k-fold validation with reproducible seeds;
- interpret the mean and variance reported for each metric across folds;
- justify when to reserve a sealed hold-out set in addition to cross-validation.

Prerequisites

Before starting, make sure you can:

- load datasets and connect learners within the Orange canvas;
- explain the difference between training, validation, and test data partitions;
- calculate averages and standard deviations for small tables of results.

Orange's **Test & Score** widget gives us a reproducible way to average over several train/test splits without juggling multiple canvases. Instead of trusting a single random partition, we select k-fold cross-validation, which slices the data into k

equally sized folds, rotates the hold-out fold, and reports the mean and variance of each metric across the rotations. The average smooths away unlucky splits that would otherwise produce contradictory accuracy scores, while the spread tells us when a model is inherently unstable. We keep baselines, tree ensembles, and neighbours in the same Test & Score run so we can compare them on identical folds rather than on separate experiments that might favour one model by chance.

Once we have the cross-validated summary, we still preserve a final hold-out partition for the end of the project. The hold-out data remains sealed until we commit to a modelling recipe, ensuring the reported test metrics reflect how the system will behave on new, unseen cases. This separation mirrors the staged evaluation in the logistic regression diagnostics chapter (chapter 9) and prepares us for production roll-outs where the first real-world batch effectively acts as another uncompromised test.

Worked example: auditing franchise scores. Figure 14.1 walks through a complete validation workflow for the Red Rooster franchise dataset. We connect logistic regression, k-NN, random forest, and a constant baseline to **Test & Score**, select $k = 10$, and fix the random seed so teammates can reproduce the splits. Orange reports accuracy, ROC AUC, F1, MCC, and log loss across folds. The mean ROC AUC highlights that the random forest ranks locations best, while the standard deviation warns that k-NN swings wildly across folds. We log these observations in the project journal before touching the hold-out set.

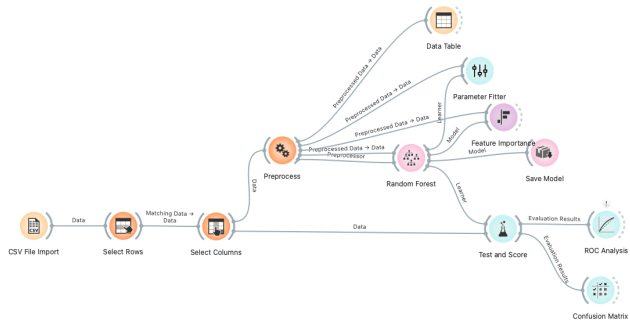


Figure 14.1: Cross-validation in Orange's **Test & Score** widget comparing baselines, neighbourhood models, and forests on the Red Rooster dataset.

Troubleshooting checklist

- If Orange refuses to run cross-validation, confirm every learner receives the same preprocessed data stream and that categorical features are encoded consistently.
- When metrics disagree across folds, inspect the **Data Table** output from each fold to catch class imbalance or duplicate records creeping into the splits.
- If results differ between team members, verify the random seeds in **Test & Score** and **Data Sampler** match the documented configuration.

  For Advanced Students. Experiment with nested cross-validation by wrapping **Test & Score** inside Orange's **Parameter Fitter**. Record how inner-loop tuning affects outer-loop estimates and compare the computational cost to a fixed validation strategy.

Section summary

- Cross-validation averages metrics over multiple folds, smoothing unlucky splits while exposing model instability.
- Reserving a sealed hold-out set protects the final evaluation from bias introduced during model selection.
- Documented seeds and shared workflows keep team members aligned when they revisit an experiment.

Self-check questions

1. Why do we still protect a hold-out set even after running k-fold cross-validation?
2. Which symptoms tell us a learner may be unstable across folds, and how should we respond?

14.2 Guard Against the Garden of Forking Paths

Warn readers about over-searching hyperparameters and reusing evaluation data.

Learning objectives

After reading this section, you will be able to:

- set boundaries on hyperparameter grids before launching a tuning sweep;
- describe logging practices that keep evaluation steps auditable;

- configure Orange's widgets so validation and testing partitions remain isolated.

Prerequisites

Before starting, ensure you:

- understand how **Data Sampler** partitions datasets;
- can export CSV logs from Orange widgets for record keeping;
- are comfortable interpreting standard error values reported by **Test & Score**.

Cross-validation is only trustworthy when we resist the temptation to rerun it indefinitely. Sweeping dozens of k values or decision-tree depths invites the "garden of forking paths": the more dials we try, the higher the chance that one combination wins by accident. To keep our search disciplined, we set expectations up front: choose a modest grid of plausible hyperparameters, log every candidate, and stop once the cross-validated difference between contenders is smaller than the reported standard error. When two models tie, we prefer the simpler configuration or whichever one is easier to explain in the accompanying hands-on activity.

Validation data should never blend with the final testing pass. Orange makes this easier by letting us allocate a separate test split in **Data Sampler** and send only the training portion through hyperparameter tuning widgets. We document the split ratios, random seeds, and selection rules in our project notes so teammates cannot accidentally reuse the test fold while iterating. This audit trail protects us when stakeholders ask how confident we are in the reported improvements, and

it gives future analysts enough context to reproduce or extend the experiment without leaking information.

Worked example: disciplined parameter sweeps. We build on the workflow from Figure 14.1 by connecting **Data Sampler** ahead of **Test & Score**. The sampler holds back 20% of records as a final test set, forwarding only the remaining data to **Parameter Fitter**. We define a three-point grid for random forest depth (4, 6, 8) and a modest range for the number of trees (50, 100, 150). Orange records each trial's cross-validated metrics. Once the standard errors overlap, we freeze the configuration, export the results table, and sign the experiment log so future teammates know why the search stopped where it did.

Troubleshooting checklist

- If Orange reports zero variance across folds, confirm that shuffling is enabled; otherwise each fold might mirror the same ordering.
- When **Parameter Fitter** produces identical winners every time, audit whether the grid is too narrow to expose trade-offs.
- If the hold-out set accidentally leaks into tuning, rebuild the canvas with explicit **Data Sampler** branches to isolate the reserved data.

  For Advanced Students. Evaluate Bayesian optimisation or adaptive grid search strategies outside Orange, then compare their efficiency against

the fixed grids executed through **Parameter Fitter**. Document when the added complexity pays off.

Section summary

- Hyperparameter discipline prevents chance configurations from masquerading as genuine gains.
- Explicit audit trails—split ratios, seeds, and exported logs—preserve evaluation integrity.
- Separate validation and test pipelines shield final metrics from tuning feedback loops.

Self-check questions

1. Which signals tell you to stop exploring additional hyperparameters?
2. How does documenting split ratios help future analysts avoid data leakage?

14.3 Choose Metrics and Baselines Wisely

Summarise scoring options for classification and regression along with necessary safety checks.

Learning objectives

After reading this section, you will be able to:

- match evaluation metrics to stakeholder goals in classification and regression projects;

- compare advanced models with explicit baseline learners inside **Test & Score**;
- interpret discrepancies between metrics to trigger diagnostic investigations.

Prerequisites

Before starting, make sure you can:

- read confusion matrices and residual plots;
- describe precision, recall, and ROC AUC in plain language;
- configure baseline learners such as Orange's **Constant** model.

Cross-validation yields a table of metrics; our job is to read the whole table instead of celebrating a single column. In the Red Rooster franchise scenario we look beyond accuracy to contrast ROC AUC, F1, and Matthews correlation coefficient (MCC). Accuracy alone rewards whichever class dominates the dataset, whereas ROC AUC judges whether the model ranks actual franchise sites above the unsuitable ones, F1 balances precision and recall for class-imbalanced campaigns, and MCC condenses the confusion matrix into a correlation-style score that penalises lopsided predictions. Reading across the columns forces us to explain why a model might lift accuracy yet fall behind on recall, a conversation that surfaces stakeholder priorities before deployment.

Regression projects need equally careful scoring. We include mean squared error (MSE) and root mean squared error (RMSE) to highlight large residuals, mean absolute error

(MAE) to focus on typical deviations, mean absolute percentage error (MAPE) when percentages matter, and R^2 to communicate variance explained. Whenever MAPE spikes or RMSE dwarfs MAE, we inspect the residual plots from chapter 7 to check for outliers or heteroskedasticity. Alongside these metrics we always score a simple baseline such as Orange's **Constant** model or a naive seasonal average. If our sophisticated ensemble cannot outperform the baseline across cross-validated folds, we pause and fix data quality or feature engineering issues before layering on more complexity. This guard rail keeps our evaluation honest and aligns directly with the rubric used in Hands-on Activity 5.

Worked example: metric review meeting. We schedule a metrics read-out using the saved results from **Test & Score**. During the meeting, the team compares ROC AUC and F1 for each model while projecting the Orange screenshot from Figure 14.1. When the logistic regression shows higher accuracy but lower recall than the forest, we debate whether marketing cares more about not missing promising sites or about minimising unnecessary follow-ups. The discussion ends with an action item to collect additional features before promising the model to stakeholders.

Troubleshooting checklist

- If metrics fluctuate dramatically between runs, verify that class stratification is enabled during cross-validation.
- When baselines outperform complex learners, inspect feature preprocessing for leakage or mis-scaled inputs.

- If percentage-based metrics explode, remove zero-valued denominators or switch to absolute error measures.

  For Advanced Students. Extend the evaluation table with cost-sensitive metrics tailored to your project. For example, compute expected monetary value for each threshold and compare it with ROC-derived operating points.

Section summary

- Reading multiple metrics together uncovers trade-offs hidden by accuracy alone.
- Baseline comparisons anchor whether additional modelling effort delivers real value.
- Metric anomalies prompt diagnostic dives into residuals, class balance, or feature engineering choices.

Self-check questions

1. How does MCC complement ROC AUC when judging franchise site classifiers?
2. Which metric would you prioritise when forecasting revenue, and why?

Chapter summary

- Structured cross-validation paired with sealed hold-out sets keeps reported performance trustworthy.

- Disciplined hyperparameter searches and audit trails prevent the garden of forking paths from eroding evidence.
- Metric literacy ensures we notice when sophisticated models fail to beat baselines or align with stakeholder goals.

Key term glossary

Cross-validation A resampling technique that rotates hold-out folds to estimate average model performance and its variability.

Standard error A measure of spread showing how much a metric varies across validation folds, helping us judge if differences are meaningful.

Garden of forking paths The risk that extensive, undocumented experimentation accidentally overfits evaluation data, producing misleading winners.

Baseline model A simple reference learner—such as a constant predictor—that anchors whether complex models add value.

Metric dashboard A curated table or visual summary of evaluation metrics shared during decision meetings to align stakeholders on trade-offs.

Extension challenges

- Recreate the workflows in this chapter using a different dataset with severe class imbalance. Document how

stratified folds affect ROC AUC.

- Implement nested cross-validation in Python or R, then compare the variance estimates with Orange's k-fold results.
- Build a cost curve for the Red Rooster classifier and present threshold recommendations in a short executive memo.

Conclusion and next steps

Our evaluation toolkit now balances rigour with practicality: cross-validation smooths variability, baselines keep improvements honest, and monitoring plans protect stakeholders after deployment. Bring these habits into Hands-on Activity 5, where you will defend a classifier choice using evidence drawn from this chapter. They also set the stage for the explainable models and ensemble discussions ahead, ensuring we judge sophisticated learners by the same disciplined standards.

Chapter 15

Hands-on Activity 5

This workshop is a tactile way to compare simple classifiers and regressors without drowning in jargon. We sort coloured tokens, measure distances with rulers, and then mirror the same logic inside Orange so the software echoes what our hands just did. Every step uses plain speech while reinforcing the neighbour-based thinking from the theory chapters.

We step into this activity after exploring the neighbour tuning playbook in chapter 11, the NSW land value regression in chapter 12, the Red Rooster classification comparison in chapter 13, and the metric guard rails in chapter 14 so the tabletop debate keeps pace with the surrounding theory chapters.

15.1 Kit Checklist

Translate the shared lab purchases into a per-group kit before students arrive. The cost estimates below assume eight working groups because eight packs of counters and rulers were ordered for the cohort. Adjust quantities if you run more or

fewer tables.

- **Kadink Bulk Pack Counters (230 pieces).** Allocate one pack to each group. At $7.99 per pack from Officeworks (Kadink counters search), the colourful discs cover the classification task.
- **Kadink Dinosaur Counters (90 pack).** Keep the pack centrally and hand a couple of dinosaurs to each group for the "test pile" marker. One $9.99 pack from Officeworks (Kadink dinosaur counters search) stretches to roughly $1.25 per group when shared across eight tables.
- **30 cm binder-mate rulers.** Provide one ruler per group to act as a linear decision boundary. The Officeworks Studymate rulers cost $1.00 each (Studymate ruler search).
- **Numbered beekeeping tags.** Each group needs a strip of sequential numbers (for example 11–16, 21–26, and so on) for the regression activity. Ten 100-piece sets were ordered from Amazon Australia for $118.09 total (Amazon beekeeping tags), so budget about $11.81 per group.
- **Optional occlusion supplies.** A compact sewing thread kit (Kmart sewing kit search) and coloured Blu Tack (Big W Blu Tack search) help students mask validation labels. One of each per group keeps everyone moving; both items were previously purchased at $2.50 and $3.75 respectively.
- **Smartphone or tablet.** Encourage each group to

photograph their layouts before tokens move so they can refer back during the Orange segment.

International substitutions. Use equivalent tokens, rulers, and markers sourced from local retailers if Officeworks, Kmart, or Big W are not part of your supply chain. The activity only relies on having clearly distinguishable colours, simple measuring tools, and small markers for test piles.

15.2 Learning Goals

By the end of the activity, students should be able to:

- Perform a clean train/validation/test split without leaking information.
- Compare a linear decision boundary (logistic regression—a straight-line probability model) against nonparametric k-NN with $k = 1$ and $k = 3$.
- Compute accuracy, precision, recall, F1, and mean absolute error (MAE) from confusion matrices or numeric predictions, remembering that precision answers "of the positive calls we made, how many were right?" while recall asks "of the real positives, how many did we catch?".
- Explain the bias–variance trade-off they observe between 1-NN and 3-NN.
- Rebuild the workflow in Orange, optionally extending it with cross-validation or hyperparameter tuning.

15.3 Part A: Classification with Coloured Tokens

15.3.1 Setup (5 minutes)

A1. Pick two or three colours to represent classes. One student scatters all the red tokens, another the greens, another the blues, so each class has its own loose cluster with some overlap. Take a quick photo.

A2. Split the dataset into approximately 60% training, 20% validation, and 20% test. Hide the validation and test labels by placing **purple** counters on top. Mark the test pile with a dinosaur so nobody peeks early.

A3. Optional stress test: split the blue tokens into two separated piles to create a non-linear class.

15.3.2 Logistic Regression by Ruler (10 minutes)

A4. Using only the **training** tokens, place a ruler to form a linear decision boundary for **red vs. not-red**. If you have spare rulers, set up additional one-vs-rest boundaries.

A5. Validate: for each **validation** token (hidden by purple), predict the class based on which side of the boundary it sits. Record the predictions without lifting the cover.

A6. Reveal the validation labels, populate a confusion matrix, and compute per-class precision, recall, F1, plus overall accuracy. Capture the layout and matrix before moving the tokens.

Remind everyone that logistic regression is simply a way to draw a straight divider and then translate distance from that divider into a probability between 0 and 1. The ruler stands in for the equation that Orange will later learn automatically.

15.3.3 k-NN on the Same Validation Set (15 minutes)

A7. Remove the rulers. With the **training** tokens, run 1-NN: for each validation token, find the single nearest training token (use the ruler as a measuring stick if needed) and predict its colour. Record predictions, reveal the labels, then score.

A8. Repeat with 3-NN: take the three closest training tokens and predict the majority colour (break ties consistently, for example with a coin flip or the nearest of the tied colours).

A9. Model selection: compare validation metrics for the ruler-based logistic boundary, 1-NN, and 3-NN. Declare the winner or note if two models are effectively tied.

15.3.4 Single Test Pass (5 minutes)

A10. Use the chosen model to classify the **test** tokens. Lift the dinosaur only after committing to predictions, then compute the test confusion matrix and metrics.

A11. Emphasise that adjusting decisions after seeing test labels constitutes data leakage; freeze the model once the test run begins.

15.3.5 Metric Reference

For each class c in a one-vs-rest breakdown:

$$\text{Accuracy} = \frac{TP + TN}{N}.$$

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}.$$

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c}.$$

$$\text{F1}_c = \frac{2\,\text{Precision}_c\,\text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}.$$

Report macro-F1 (the mean of per-class F1) or list the class-level values directly.

Common gotchas

- If blue splits into two piles, a straight boundary struggles; expect 3-NN to outperform the ruler.
- Strong class imbalance can make raw accuracy misleading. Encourage students to focus on recall and F1 instead.
- Agree on a deterministic tie-breaker for 3-NN before scoring to keep results reproducible.

15.4 Part B: One-Dimensional k-NN Regression

15.4.1 Setup (5 minutes)

B1. Treat a table edge as the x-axis. Invite students to place tags so smaller numbers sit left and larger numbers

right, with different people handling different ranges to introduce local noise.

B2. Split the tags into **training**, **validation**, and **test**. Flip or cover the validation and test values so only their positions remain visible. Use the hole in each tag to define its location precisely.

15.4.2 k-NN Regression (15 minutes)

B3. Predict each **validation** tag using 1-NN: find the nearest training tag by distance along the edge and copy its value.

B4. Repeat with 3-NN: average the three closest training values (a simple arithmetic mean works well).

B5. Compute the mean absolute error (MAE) on the validation fold,

$$\text{MAE} = \frac{1}{m} \sum_{i=1}^{m} |y_i - \hat{y}_i|.$$

Optionally compare RMSE if students want to penalise large errors more heavily.

B6. Pick the better k based on validation MAE, then run a single evaluation on the **test** tags to report test MAE.

Regression gotchas

- 1-NN clings to local noise ("low bias" meaning it hugs the training data closely, but "high variance" because it can change wildly with new points); 3-NN smooths it out (higher bias, lower variance). Discuss which behaviour matches the dataset in front of them.

- Tweaking k after you have seen the test MAE invalidates the result. Treat the test pass as a one-shot audit.

15.5 Part C: Rebuild the Workflow in Orange

15.5.1 Painted Classification (required, ∼25 minutes)

C1. Open Orange and drag `Paint Data`. Recreate the Part A layout, assigning a discrete class colour to each cluster.

C2. Add `Select Columns` to confirm the painted class is the **Target**.

C3. Add two `kNN` learners (set $k \in \{1, 3\}$) and a `Logistic Regression` learner.

C4. Feed the painted data and learners into `Test & Score`. Choose 5-fold cross-validation.

C5. Record Accuracy, macro-F1, and AUC for each learner. Open `Confusion Matrix` to inspect per-class behaviour and `Scatter Plot` to compare decision boundaries.

C6. Optional: insert `Optimize Parameters` for kNN with $k \in \{1, 3, 5, 7\}$ and macro-F1 as the metric. This mirrors the manual validation you just performed.

15.5.2 Regression in Orange (optional, 10–15 minutes)

R1. Use the Telco churn CSV (`../Practicals/Week5/WA_Fn-UseC_-Telco-Customer-Churn.csv`) with `File` →

`Select Columns` to set `TotalCharges` (or another numeric feature) as the target, then compare 1-NN and 3-NN regressors inside `Test & Score` using MAE.

R2. Alternatively, approximate the one-dimensional layout by painting a narrow strip in `Paint Data`, converting one variable into a continuous target via `Edit Domain`, and evaluating the kNN regressor.

R3. Save the workflow so students can revisit it later (`../Practicals/Week5/telco-knn.ows`).

Debrief Prompts

- When blue was split into two piles, how did logistic regression compare with k-NN? Why?
- Did 1-NN overreact to local clutter? Did 3-NN smooth it out?
- If you increased k to 5 with a small dataset, what failure modes would you expect?

Encourage students to keep photos of their physical layouts alongside screenshots of the Orange workflow so they can rehearse the reasoning between classes.

Chapter 16

Design Explainable Models with Rules and Trees

Chapter overview

We now turn evaluation discipline toward explainability. This chapter helps us decide when stakeholders need transparent models, walks through rule- and tree-based learners in Orange, and shows how to communicate their insights responsibly. The lessons lead directly into the ensemble chapter and Hands-on Activity 6, where we balance interpretability with ranking strength in a shared workspace.

16.1 Decide Whether You Need Explanations or Pure Accuracy

Start each project by clarifying whether stakeholders demand insight, automated predictions, or both.

Learning objectives

After reading this section, you will be able to:

- articulate the trade-off between interpretability and raw predictive accuracy;
- map stakeholder requirements to suitable model families in Orange;
- document decision rationales that withstand regulatory or audit scrutiny.

Prerequisites

Before starting, ensure you can:

- summarise stakeholder goals in a short design brief;
- distinguish between descriptive, predictive, and prescriptive analytics tasks;
- navigate the Orange canvas to add classification learners.

Before launching a modelling sprint we meet with stakeholders to decide whether transparent reasoning outranks raw accuracy. Some projects are classic inference workloads: we must explain why service requests spike or which regional factors influence franchise success, so the organisation can intervene. Others are industrial automation problems where

millions of decisions flow through a scoring API and the only goal is a reliable yes or no. Laying out the expected workload guides our choice of modelling family and evaluation targets.

Regulations often settle the debate for us. The European Union's GDPR and China's Personal Information Protection Law both require meaningful explanations when automated decisions carry legal or significant personal effects. Even without legal pressure, explainable models build trust, accelerate debugging, and give product teams differentiators when pitching data-driven features. Documenting the rationale in a short decision memo keeps future analysts aligned with the original choice and provides evidence if auditors ask why a black-box model was rejected.

Worked example: stakeholder alignment workshop. We run a half-hour workshop before opening Orange. The operations team outlines their need to understand why certain suburbs underperform, while marketing emphasises turnaround speed for campaign approvals. Together we score priorities on a whiteboard and conclude that rule-based explanations must accompany any predictions. We then record this agreement in the project journal and tag our Orange workflow with notes explaining why CN2 and pruned trees will anchor the modelling phase. This explicit trace links stakeholder goals to modelling choices and saves time when executives revisit the decision weeks later.

Troubleshooting checklist

- If stakeholders cannot agree on priorities, draft separate evaluation dashboards for interpretability and accuracy so they can compare outcomes directly.

- When legal requirements feel ambiguous, consult compliance guidelines early and cite the relevant clauses in your modelling plan.

- If the team defaults to opaque models out of habit, prototype a transparent alternative in Orange to demonstrate what might be lost.

  For Advanced Students. Compare stakeholder alignment frameworks such as value-sensitive design or algorithmic impact assessments. Evaluate how each approach structures conversations about explainability obligations.

Section summary

- Stakeholder clarity determines whether we prioritise transparent reasoning or maximal accuracy.

- Regulatory contexts often mandate explainable pathways, especially when decisions affect individuals.

- Documenting the decision process provides future auditors with evidence that obligations were met.

Self-check questions

1. Which factors tip the balance in favour of interpretability during the workshop?

2. How would you document the rationale for selecting a black-box model when transparency is less critical?

16.2 Engineer Features that Support Human Reasoning

Augment core data with interpretable attributes so rule learners can mirror the factors humans care about.

Learning objectives

After reading this section, you will be able to:

- design feature engineering checklists that mirror stakeholder language;
- evaluate data quality after joins and imputations in Orange;
- annotate workflows so collaborators understand transformation choices.

Prerequisites

Before starting, make sure you:

- can calculate ratios and percentages from raw census attributes;
- know how to use Orange's **Feature Constructor** and **Select Columns** widgets;
- are comfortable sampling records to verify join accuracy.

Explainable modelling depends on sensible inputs. We begin with a brainstorming session to list the socio-economic signals that community managers already discuss—household income, vehicle access, cultural hubs, language communities—and note which public datasets contain them. From

there we gather complementary tables, such as census counts or transport accessibility scores, and calculate ratios that humans reason about, like average cars per household or proportion of young families per suburb. Joining these attributes carefully ensures every franchise location inherits the context we need to generate persuasive rules.

Each join deserves a quick quality check: we confirm row counts before and after merges, sample a few locations to compare against known facts, and record any imputed values so they are not mistaken for real observations later. Annotating the Orange canvas, just as we do in chapter 6, keeps those decisions close to the data. When the features align with stakeholder language, CN2 and decision trees can surface narratives that sound familiar rather than surprising jargon.

Worked example: constructing interpretable ratios. Starting from the Red Rooster feature table, we create a new Orange workflow that adds census-derived population density and cultural diversity scores. Using **Feature Constructor**, we derive "family ratio" as the proportion of households with children under ten and "mobility score" as the share of residents commuting by public transport. After each transformation we add a **Data Table** widget to inspect sample rows and confirm that values align with known suburbs. Once satisfied, we annotate the workflow with sticky notes documenting each feature's rationale so stakeholders can trace the logic when reviewing rules.

Troubleshooting checklist

- If joins drop records unexpectedly, double-check key formats and clean whitespace or casing issues before

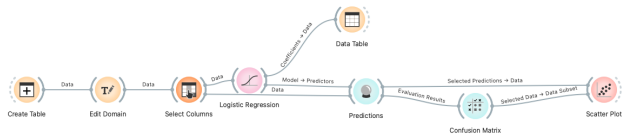


Figure 16.1: Annotated Orange workflow showing feature construction steps prior to training CN2 and decision tree models.

merging.

- When engineered ratios explode due to small denominators, add smoothing terms or cap extreme values before training.
- If collaborators struggle to follow the canvas, group widgets spatially and use colour-coded annotations to highlight data sources.

  For Advanced Students. Prototype feature importance calculations using shapley-value approximations on top of your engineered features. Compare which attributes CN2 emphasises versus a black-box gradient boosting model.

Section summary

- Feature design anchored in stakeholder vocabulary keeps explainable models persuasive.
- Routine data quality checks prevent misinterpreting imputed or mismatched records.
- Well-annotated workflows accelerate peer review and onboarding.

Self-check questions

1. Which validation steps help you trust a newly engineered ratio?

2. How do workflow annotations support future maintenance of explainable models?

16.3 Induce Transparent Rules with CN2

Use rule learners to capture geographic or demographic heuristics in concise, auditable statements.

Learning objectives

After reading this section, you will be able to:

- configure the CN2 rule inducer in Orange and interpret its outputs;

- evaluate rule quality using weighted relative accuracy and coverage;

- collaborate with domain experts to refine or challenge induced rules.

Prerequisites

Before starting, confirm you:

- have engineered interpretable features as described earlier in the chapter;

- can export Orange widget outputs to CSV for offline review;

- understand confusion matrices and class balance considerations.

The CN2 rule inducer hunts for compact descriptions that separate our target classes. It proposes mid-point splits on candidate features, tracks which combinations improve weighted relative accuracy, and locks in the best-performing clause before searching for the next. Each clause reads like "If median household income exceeds $85,000 and vehicle access is high, then franchise potential is good," mirroring how franchising teams already argue their case.

Interpreting the CN2 output means scanning both the ranked rules and the default catch-all clause. Weighted relative accuracy shows how much better the rule performs compared with random guessing, while coverage tells us how many examples it applies to. We export the rule list as a CSV or PDF so domain experts can annotate it, circle clauses that feel counterintuitive, and trace misclassifications back to the feature thresholds. These discussions often inspire extra features or prompt us to collect new context before training the next iteration.

Worked example: co-authoring rules with experts. After training CN2 on the engineered dataset, we invite the franchise operations lead to a review session. Together we open the Orange rule viewer and sort clauses by weighted relative accuracy. When we encounter a rule stating that "high mobility score and medium income" leads to poor performance, the expert recalls a specific suburb that contradicts the clause. We flag the example, trace it back through **Data Table**, and discover that recent transport upgrades have not yet been captured in the dataset. The insight triggers a follow-up task

to update mobility metrics before the next training run.

Troubleshooting checklist

- If CN2 returns overly long rules, tighten the beam width or increase the minimum covered examples to focus on meaningful segments.
- When conflicting rules appear, cluster them by shared predicates and discuss with stakeholders which clause best reflects reality.
- If coverage is too low, revisit feature engineering to ensure relevant context is available to the learner.

  For Advanced Students. Experiment with Laplace correction or entropy-based pruning parameters in CN2. Compare how these adjustments affect rule generality and stakeholder trust.

Section summary

- CN2 surfaces human-readable clauses that align with operational heuristics.
- Collaborative review ensures rules reflect up-to-date context and domain knowledge.
- Tuning rule complexity keeps explanations concise and actionable.

Self-check questions

1. How do weighted relative accuracy and coverage complement one another when ranking rules?

2. Which collaboration practices keep CN2 rules relevant as the environment changes?

16.4 Explain Decisions with Trees

Leverage decision trees when you need higher accuracy while keeping reasoning pathways accessible.

Learning objectives

After reading this section, you will be able to:

- configure tree depth, leaf sizes, and pruning options in Orange;
- narrate decision paths from root to leaf using stakeholder vocabulary;
- pair tree visualisations with evaluation metrics to justify adoption.

Prerequisites

Before starting, make sure you:

- can interpret impurity measures such as Gini and entropy;
- know how to export tree diagrams from Orange for presentation;
- understand the ensemble context provided in chapter 17.

Decision trees extend the rule-based mindset while nudging accuracy upward. Each split maximises impurity reduction—measured by Gini or entropy—so the tree quickly isolates

the most informative features. We control the bias–variance trade-off by tuning maximum depth, minimum samples per leaf, or post-pruning settings: shallow trees provide high-level policies, while deeper trees trace nuanced interactions between demographics and location characteristics.

Tree visualisations show how feature space partitions into rectangles. Moving from root to leaf, we can narrate decisions as "Start with housing density, then branch on household income, and finally check commuter ratios." When we prune the tree to a few layers and export it as a PDF, stakeholders can literally print the reasoning and keep it next to their site selection checklist. This blend of clarity and modest accuracy upgrades makes trees a natural bridge to the ensemble methods we explore in chapter 17 and the Orange workflows in Hands-on Activity 6.

Worked example: pruning for presentation. We train a decision tree on the same engineered features used for CN2, setting maximum depth to six and minimum leaf size to 20. Orange's tree viewer initially displays a complex structure, so we enable post-pruning until validation accuracy plateaus while the number of leaves drops. The resulting tree contains four levels, each tied to stakeholder-friendly splits such as income bands and mobility scores. We export the diagram, highlight the key paths in a slide deck, and rehearse how to explain each branch during the next steering committee meeting.

Troubleshooting checklist

- If the tree overfits, enable pruning or reduce maximum depth to prevent brittle leaf nodes.

- When splits rely on obscure features, revisit engineering choices or adjust feature selection to emphasise interpretable attributes.

- If visualisations become cluttered, export the tree and annotate only the most important branches for discussion.

For Advanced Students. Analyse how surrogate splits or rule extraction techniques approximate deeper tree logic. Compare their fidelity to the original model when presenting to expert audiences.

Section summary

- Decision trees offer a balance between interpretability and accuracy when tuned thoughtfully.

- Pruning and feature selection keep tree narratives focused on stakeholder-relevant signals.

- Visual storytelling turns model logic into assets for executive communication.

Self-check questions

1. How does pruning influence the balance between accuracy and interpretability?

2. Which features would you highlight when narrating a tree to decision-makers?

Chapter summary

- Stakeholder alignment determines when explainability outweighs marginal accuracy gains.
- Human-centred feature engineering primes CN2 and trees to produce persuasive narratives.
- Collaborative review and pruning practices keep rule- and tree-based explanations trustworthy over time.

Key term glossary

Explainability brief A documented agreement capturing stakeholder priorities, regulatory obligations, and desired transparency outcomes.

Weighted relative accuracy CN2's improvement score over random guessing, balancing precision with rule coverage.

Coverage The proportion of instances a rule or tree path applies to, guiding whether an explanation is broadly useful.

Post-pruning A technique that removes branches from a trained tree to improve generalisation and simplify explanations.

Workflow annotation Notes embedded within the Orange canvas to record data sources, feature logic, and modelling assumptions.

Extension challenges

- Recreate the feature engineering workflow for a healthcare or energy dataset and compare how domain context shifts the induced rules.
- Translate a pruned decision tree into a natural-language policy manual for franchise selection and gather stakeholder feedback.
- Prototype a hybrid system where a black-box model triggers alerts that must be explained using CN2-derived surrogate rules.

Conclusion and next steps

We have assembled a toolkit for pairing stakeholder requirements with explainable models, interpreting rule and tree outputs, and reporting them with care. Carry these habits into Hands-on Activity 6, where the watchband scenario will ask you to balance transparency with performance in front of peers. The next chapter on ensembles will stretch the same judgement: we will chase ranking gains without losing sight of the explanations your stakeholders now expect.

Chapter 17

Balance Ranking Metrics and Ensembles for Stronger Models

Chapter overview

We close this sequence by pairing explainability with ranking power. The chapter unpacks ROC curves as ranking tools, explores when to reach for ensembles, and shows how to communicate their benefits without overstating certainty. Each section builds on the evaluation guard rails and prepares us for Hands-on Activity 6, where we weigh tree-based transparency against ensemble strength on the watchband dataset.

17.1 Read ROC Curves as Ranking Tools

Treat AUC as evidence that your model orders cases well, not as proof that its probabilities are trustworthy.

Learning objectives

After reading this section, you will be able to:

- interpret ROC curves produced by Orange's **Test & Score** widget;
- explain why AUC reflects ranking quality rather than probability calibration;
- link ROC analysis to the broader evaluation safeguards from chapter 14.

Prerequisites

Before starting, ensure you:

- understand true positive and false positive rates;
- can configure Orange learners and evaluation widgets to output ROC curves;
- have baseline familiarity with precision–recall trade-offs.

Receiver operating characteristic (ROC) curves plot the true positive rate against the false positive rate as we slide a decision threshold from lenient to strict. Each point on the curve shows how much positive coverage we gain for the false alarms we are willing to tolerate. The area under the curve (AUC) summarises that ranking behaviour: it is the probability that a randomly chosen positive example receives a higher score than a randomly chosen negative one. AUC therefore tells us how confidently the model sorts cases, not whether the absolute probability values are calibrated.

We always pair ROC analysis with sanity checks. Accuracy and baseline comparisons from chapter 14 confirm that

a higher AUC still translates into practical gains, and calibration plots reveal whether the scores line up with observed frequencies. When two models have similar AUCs, we examine the steepness of the initial curve segment: a sharp early rise means we can capture most positives with few false alarms, which is invaluable for triaging limited human attention.

Worked example: interpreting ROC in Orange. Using the workflow from Figure 14.1, we open the ROC visualiser attached to **Test & Score**. The random forest and logistic regression curves appear on the same axes, letting us observe how quickly each model lifts true positives. The forest's curve hugs the left axis, indicating strong ranking ability, while the baseline lags along the diagonal. We export the chart, annotate the operating points under consideration, and paste the image into the evaluation report for stakeholders.

Troubleshooting checklist

- If ROC curves look identical across models, verify that class shuffling and stratification are active to avoid fold artefacts.

- When AUC appears high but precision remains low, inspect calibration plots and confusion matrices to detect poorly chosen thresholds.

- If Orange cannot generate ROC curves, ensure the learners output class probabilities rather than hard labels.

  For Advanced Students. Compare ROC analysis with precision–recall curves on imbalanced

datasets. Quantify when one metric offers clearer guidance than the other.

Section summary

- ROC curves reveal how models trade true positives for false positives as thresholds vary.
- AUC measures ranking ability but must be paired with calibration and baseline comparisons.
- Visual overlays help teams debate which model best supports limited operational capacity.

Self-check questions

1. Why does AUC capture ranking quality rather than probability accuracy?
2. When would you prefer a precision–recall curve over ROC?

17.2 Set Thresholds to Match Business Costs

Move decision cut-offs based on the relative harm of false positives and false negatives instead of defaulting to 0.5.

Learning objectives

After reading this section, you will be able to:

- translate stakeholder cost discussions into decision thresholds;

- use Orange outputs to evaluate alternative operating points;
- document threshold rationales for future audits.

Prerequisites

Before starting, make sure you:

- can interpret confusion matrices and cost tables;
- know how to export evaluation results from Orange for offline analysis;
- understand the business context driving classification decisions.

Ranking metrics are only the beginning—we still need to pick a threshold that reflects the stakes. We map out the cost of a false positive versus a false negative with stakeholders, then use ROC or cost curves to find the operating point that minimises expected loss. Sometimes we end up far from 0.5: a fraud detection system might trigger an investigation at 0.2 if missed cases are expensive, while a marketing campaign might require 0.7 to avoid spamming uninterested customers.

Once we select the threshold, we translate it into operational guidance. For probability-based models we note the exact cut-off; for ensembles that vote, we specify how many base learners must agree before we accept a positive. We capture the rationale alongside the metric dashboard so future evaluations respect the original trade-off, especially when new data drifts or leadership changes.

Worked example: choosing a franchise approval threshold. We meet with the franchise steering group to quantify costs: approving a poor site wastes $250,000 in fit-out expenses, while delaying a strong site costs $80,000 in lost revenue. Using the ROC plot from Orange, we highlight the threshold where the false positive rate remains low while recall stays above 0.75. This point corresponds to a probability cut-off of 0.63. We document the calculation, update the Orange workflow with a **Threshold** widget, and note the decision in the project log.

Troubleshooting checklist

- If stakeholders struggle to articulate costs, facilitate a scenario-planning exercise that estimates worst-case impacts.
- When cost curves are unavailable, approximate expected value manually using exported confusion matrices.
- If performance drifts after deployment, schedule quarterly reviews to revisit the threshold with fresh evidence.

  For Advanced Students. Implement adaptive thresholding that updates based on rolling cost estimates. Compare its performance against a fixed threshold in a backtesting experiment.

Section summary

- Thresholds should encode the real-world costs of false positives and negatives.
- Documented rationales keep future audits aligned with the original decision context.

- Regular reviews prevent drift from eroding carefully chosen operating points.

Self-check questions

1. How would you explain the trade-off between recall and false positives to a non-technical stakeholder?

2. Which signals indicate it is time to revisit an operational threshold?

17.3 Stack Diverse Learners for Complementary Strengths

Combine rules, trees, neighbours, and baselines when each captures different patterns in the data.

Learning objectives

After reading this section, you will be able to:

- assemble ensembles in Orange's **Voting** widget with diverse base learners;

- evaluate ensemble performance against individual models using cross-validation;

- maintain experiment logs that clarify why each learner was included.

Prerequisites

Before starting, ensure you:

- understand the strengths and weaknesses of logistic regression, k-NN, CN2, and random forests;
- can configure preprocessing steps such as scaling or feature selection;
- know how to interpret Orange's aggregated evaluation tables.

Stacking allows us to combine multiple modelling perspectives. We start with a diverse bench: logistic regression for linear structure, k-NN for local geometry, CN2 or trees for interpretable rules, and a simple baseline. Orange's **Voting** widget aggregates their predictions, either by averaging probabilities or taking a majority vote, often outperforming any individual model.

Diversity matters more than quantity. Adding near-duplicate models introduces noise without real signal gain, so we curate the ensemble deliberately. We log which base learners entered the stack, the validation scores that justified them, and any preprocessing steps (such as feature scaling) they required. This documentation keeps the ensemble maintainable and sets the stage for automating the workflow when the project outgrows manual Orange canvases.

Worked example: documenting a stacked ensemble. We extend the workflow from Figure 14.1 by introducing Orange's **Voting** widget. Logistic regression, k-NN with normalised features, CN2, and random forest feed into the ensemble. After running cross-validation, we record that the ensemble lifts ROC AUC by 0.03 compared with the best individual model. We capture these results in a shared spread-

sheet, noting preprocessing requirements and learner settings so future teammates can reproduce the stack.

Troubleshooting checklist

- If the ensemble underperforms, audit whether base learners are too similar or whether preprocessing created leakage.
- When Orange reports inconsistent improvements, check that each learner receives identical training folds.
- If collaboration becomes difficult, snapshot the workflow with versioned filenames and link them in the experiment log.

For Advanced Students. Compare Orange's majority-vote stacking with soft-voting or meta-learner stacking implemented in Python. Analyse when each approach offers the best balance of accuracy and interpretability.

Section summary

- Diverse base learners capture complementary patterns and lift ensemble performance.
- Careful documentation of preprocessing and validation settings keeps ensembles reproducible.
- Voting strategies provide flexible ways to balance interpretability with predictive strength.

Self-check questions

1. Which criteria do you use to decide whether a learner deserves a place in the ensemble?
2. How would you document ensemble settings so a teammate can rebuild the workflow?

17.4 Adopt Random Forests and Lightweight Tuning

Lean on tree ensembles as reliable baselines while tuning key hyperparameters with cross-validation.

Learning objectives

After reading this section, you will be able to:

- explain how random forests reduce variance through bootstrapping and feature subsampling;
- tune key hyperparameters using Orange's **Parameter Fitter**;
- communicate feature importance insights to stakeholders.

Prerequisites

Before starting, make sure you:

- understand decision tree mechanics and impurity measures;
- can interpret cross-validated metric tables from **Test & Score**;

- know how to export feature importance plots from Orange.

Random forests average the predictions of many decision trees that each see different feature subsets and bootstrapped samples. Randomising the split candidates reduces correlation between trees, which lowers variance and makes forests robust on tabular data. They become our go-to baseline when we need strong accuracy without extensive feature engineering.

We still tune a few high-impact parameters. Orange's **Parameter Fitter** explores the number of trees or the maximum depth, using cross-validation from chapter 14 to judge stability. Because the widget adjusts one parameter at a time, we note any interactions to revisit later with script-based tooling. Even when individual trees are opaque, feature importance plots summarise which variables drive predictions, giving us a talking point for stakeholders who need at least directional explanations. These insights close the loop with the explainable modelling chapter while preparing us to escalate towards gradient boosting in future iterations.

Worked example: tuning a random forest baseline. We start with 100 trees and a maximum depth of eight. **Parameter Fitter** evaluates tree counts of 50, 100, and 150 across ten folds. The results show diminishing returns beyond 100 trees, while increasing depth to ten introduces higher variance. We lock in 100 trees with depth eight, export the feature importance chart highlighting income, mobility, and population density, and share the summary with stakeholders alongside CN2 and tree explanations.

Troubleshooting checklist

- If training slows dramatically, reduce the number of trees or parallelise runs outside Orange.
- When feature importance seems unstable, rerun cross-validation with different seeds to confirm the signal.
- If forests dominate every comparison, re-evaluate whether simpler, more interpretable models would suffice given stakeholder needs.

  For Advanced Students. Compare Orange's random forest with gradient boosting implementations. Analyse performance gains relative to the additional tuning burden.

Section summary

- Random forests provide strong baselines through variance reduction and feature subsampling.
- Light-touch tuning balances accuracy improvements with reproducibility.
- Feature importance charts keep ensembles connected to explainability goals.

Self-check questions

1. Which hyperparameters deliver the largest performance gains for random forests in Orange?
2. How do you explain feature importance rankings to non-technical stakeholders?

Chapter summary

- ROC literacy ensures we judge ranking quality with clear eyes before locking in thresholds.
- Cost-aware thresholding, ensemble design, and random forest tuning work together to deliver reliable classifiers.
- Documentation and stakeholder dialogue transform technical metrics into actionable deployment plans.

Key term glossary

ROC curve A plot showing the trade-off between true positive and false positive rates as the decision threshold varies.

Operating point The threshold or ensemble vote that defines when a model classifies an instance as positive.

Soft voting An ensemble strategy that averages class probabilities from base learners before choosing the predicted label.

Feature importance A ranking that summarises how much each input contributes to the predictions of a model such as a random forest.

Threshold audit A documented review that revisits operating cut-offs in light of new costs or observed drift.

Extension challenges

- Recompute ROC and cost analyses for a highly imbalanced dataset and compare outcomes with precision–

recall curves.

- Prototype a meta-learner ensemble using Python scripts, then benchmark it against Orange's **Voting** widget configuration.
- Conduct a quarterly threshold audit using simulated drift scenarios and summarise recommended adjustments for leadership.

Conclusion and next steps

We now know how to read ROC curves critically, tune ensembles for ranking strength, and communicate their trade-offs with stakeholders. Take these insights into Hands-on Activity 6: when your group debates whether to favour CN2 or random forests for the watchband brief, use the evidence frameworks developed here. Looking beyond the activity, keep your monitoring and calibration plans in play—they ensure that any ensemble you deploy continues to earn trust over time.

Chapter 18

Hands-on Activity 6

We use printed customer cards and wooden sticks to see how rule-based models make decisions before trusting the same logic inside Orange. By physically drawing lines between people, we can explain every split in plain language, then confirm the computer version matches. The goal is to keep explainable AI concrete and friendly.

Our watchband investigation assumes the transparency commitments from chapter 16 and the ranking trade-offs in chapter 17, keeping the hands-on comparisons grounded in the theory chapters that frame explainable ensembles.

This activity walks students from a tangible decision-tree simulation through CN2-style rule learning and into an Orange workflow. The physical exercise uses 18 printed customer records describing watchband purchases. Age acts as the vertical axis, income as the horizontal axis, and colour is the class label.

18.1 Kit Checklist

Allocate supplies before the class so each table can run the full sequence without waiting. Twenty kits were prepared for the cohort, so the per-group quantities and costs below assume the same twenty-way split.

- **NeverTear watchband cards.** Print and trim one complete set of `watchband.csv` cards on durable stock. The Officeworks document-printing service produced twenty laminated sets for $145 total (Officeworks document printing), which works out to about $7.25 per group.

- **Zip document wallet.** Store each card deck and the evaluation notes inside a single wallet. Four five-packs of Keji A4 zip wallets cost $23 in total, so budget roughly $1.15 per group (Keji wallet search).

- **Wooden stirring sticks.** Provide around 25 stirrers per table so students can mark proposed splits. One 500-pack from Officeworks at $10.98 covers all twenty kits (Wooden stirring sticks search), or about $0.55 per group.

- **Guillotine cutting allowance.** Trimming the printed cards required six finishing services at $1 each (Guillotine cut search). Set aside roughly $0.30 per group if you are replenishing the decks.

- **Table space and camera.** Each group needs a full tabletop to lay out the cards plus a phone to photograph the final arrangement before moving to Orange.

International substitutions. Use any locally printable cardstock, resealable folders, and simple markers if Officeworks products are unavailable. The intent is to have durable cards, clear table organisation, and lightweight sticks for marking splits—the suppliers can change without affecting the learning.

18.2 Learning Goals

By the end of the session, students will be able to:

- Explain how Gini impurity scores guide split selection in decision trees.
- Run a short beam-search procedure that mirrors CN2 rule induction.
- Compare physical rules and trees with the Orange implementations of tree, CN2, and random forest learners.
- Interpret feature importance scores from an ensemble and relate them back to the physical splits.

18.3 Part A: Decision Trees with Paddle-Pop Sticks

18.3.1 Lay out the dataset

A1. Arrange the 18 cards on a clear table so income increases along the horizontal axis and age increases vertically. The exact spacing is flexible; preserve the ordering rather than the scale.

A2. Keep a generous border around the grid so you can insert stirrers without disturbing neighbouring cards.

A3. Photograph the initial layout for later comparison.

Figure 18.1: Watchband cards arranged on a table with wooden stirrers marking axes so age rises upward and income moves left to right before any splits are placed.

18.3.2 Score candidate splits

Discuss where a single vertical or horizontal stirrer could create two purer subsets. For any proposed split:

S1. Tally the orange (r), blue (b), and green (g) cards on one side of the stick and record the total $N = r + b + g$.

S2. Compute the Gini index $G = 1 - \frac{r^2+b^2+g^2}{N^2}$ for that half.

S3. Repeat for the opposite side and add the two impurity scores, weighted by each side's size, to compare candidates.

Choose the lowest-impurity split and leave the stirrer in place to mark the root decision.

Explain that a Gini index of 0 means the split side holds only one colour (perfectly pure) while values closer to 0.5 reveal a mixed group, so lower numbers are always better.

18.3.3 Extend the tree

T1. Divide the group so each person focuses on one branch. Repeat the scoring process within that subset to identify the best child split.

T2. Stop early if a branch is pure or contains two or fewer cards; otherwise continue until every leaf is pure or tiny.

T3. Sketch the resulting decision tree, labelling each node with its split rule and class distribution. Capture another photo before tidying up the sticks.

18.4 Part B: CN2 Rule Induction Game

CN2 learns *if–then* rules using a separate-and-conquer strategy. Recreate a simplified beam search on the card layout before turning to software.

18.4.1 Key ideas recap

- Hypotheses are conjunctions of simple literals such as "Age > 24" or "Income $\leq 90\,000$".
- The star operator specialises a rule by adding one additional literal, shrinking the covered rectangle on the grid.
- Maintain a beam of the top $B = 3$ candidate rules, refreshing it each round.
- Once a high-quality rule is accepted, remove the cards of the target class that it covers and search again for the remaining positives.

Make it clear that the "beam" is just a short list of the best candidates we carry forward each turn so the search stays manageable.

18.4.2 Weighted Relative Accuracy

Score rules with Weighted Relative Accuracy (WRAcc) for a target class c:

$$\text{WRAcc} = \frac{n}{N}\left(\frac{p}{n} - \frac{P}{N}\right) = \frac{p}{N} - \frac{n}{N}\cdot\frac{P}{N},$$

where N is the dataset size, P is the number of class-c cards overall, n is the number of cards covered by the rule, and p is the number of covered cards of class c.

For example, the rule `Age` $>$ `24 & Income` $\leq 90\,000 \Rightarrow$ `Green` has $(p, n, P, N) = (5, 6, 6, 18)$ and a WRAcc of $\frac{1}{6} \approx 0.167$. WRAcc compares the rule's hit rate with the baseline share of the class, so positive values mean the rule finds more target cases than guessing by overall frequency alone.

18.4.3 Play one beam search per class

C1. Choose a target class (for instance, `Blue`). Record P and $N = 18$ once.

C2. Build the initial beam from all one-literal rules that mirror the valid stirrer positions. Keep the top three by WRAcc.

C3. Specialise each kept rule by adding one more literal that still covers at least one target-class card. Re-score, keep the top three, and repeat until WRAcc no longer improves or the rule reaches accuracy ≥ 0.8 with support

$n \geq 3$.

C4. Add the best rule to your ruleset, remove the covered target-class cards from the table, and return to step **C1** for the remaining positives. Move to the next class once all positives are covered.

Compare the final rule list against the regions defined by your physical decision tree: they should describe similar rectangles.

18.5 Part C: Rebuild the Workflow in Orange

Transition to Orange to validate the manual reasoning.

O1. Load `watchband.csv` (`../Practicals/Week6/watchband.csv`) with a `File` widget and confirm the target is the colour column.

O2. Add `Tree`, `CN2 Rule Induction`, and `Random Forest` learners. Set the CN2 widget's evaluation measure to WRAcc to match the tabletop exercise.

O3. Connect all learners to `Test & Score` and choose 5-fold cross-validation.

O4. Inspect `Tree Viewer` to see if the learned splits replicate your stirrer layout, then open `CN2 Rule Viewer` to compare rules. Finally, use `Rank` or `Random Forest` feature importance to discuss why age or income dominated the decisions.

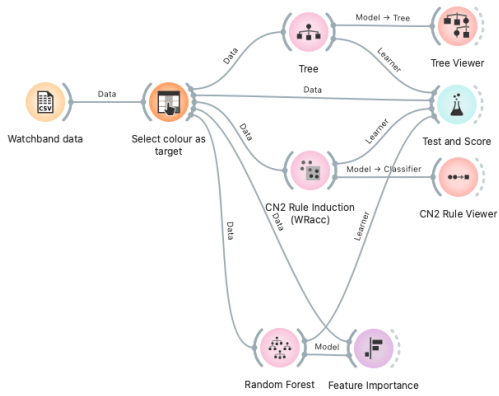


Figure 18.2: Orange workflow with File feeding Tree, CN2 Rule Induction, and Random Forest widgets into Test & Score, mirroring the physical card exercise with explainable and ensemble models.

18.6 Watchband Dataset

The practical uses the following 18-row dataset. Keep a printed copy near each table for quick reference.

#	**Age**	**Income**	**Colour**
1	12	0	Orange
2	14	10	Orange
3	15	0	Orange
4	17	500	Blue
5	18	10 000	Orange
6	19	30 000	Blue
7	21	0	Blue
8	21	20 000	Blue
9	23	30 000	Blue
10	25	40 000	Green
11	27	100 000	Orange
12	30	80 000	Green
13	40	150 000	Orange
14	41	60 000	Green
15	60	100 000	Green
16	62	40 000	Blue
17	63	60 000	Green
18	65	30 000	Green

Encourage each group to archive their physical notes and Orange results so they can revisit the decision regions when studying ensemble methods later in the course.

Chapter 19

Explore NSW Road-Crash Data with pandas

Chapter overview

We translate the Week 8 lecture walkthrough into notebook-ready code so our crash-severity modelling starts from a tidy, well-understood dataset. The chapter shows how to load the official New South Wales crash workbook, interrogate its schema, separate Series from DataFrames, and build engineered ratios that capture the difference between minor and life-threatening incidents. We finish by using Seaborn pair plots to surface correlated risk factors and point ahead to the Week 8 Python practical where we reproduce the same exploration interactively.[1]

[1] Hands-on lab: Week 8 Pandas crash analysis notebook (forthcoming).

19.1 Import and inspect the crash workbook

Use `pandas.read_excel` to load the crash records, establish a reliable index, and audit the incoming schema before modelling.

Learning objectives

After this section you will be able to:

- load large Excel workbooks directly into pandas using keyword arguments informed by the documentation;
- confirm basic structure with `.shape`, `.columns`, and `.head` so unexpected fields surface quickly;
- promote unique identifiers such as the crash number into an index for faster joins and clearer filtering.

Prerequisites

Before diving in, make sure you can:

- navigate the Week 8 data bundle and upload files to Google Colab or your local Jupyter environment;
- interpret pandas `Series` and `DataFrame` outputs at a glance;
- recognise obvious missing-value sentinels (for example, impossible distances like 999,000 km).

Pandas fundamentals refresher

Before opening the workbook, pause to recall how pandas organises data. A `DataFrame` behaves like a spreadsheet table:

rows represent records and columns hold variables, each with a shared index that labels the rows. A `Series` is the one-dimensional companion that stores a single column (or row) with the same index, making it ideal for targets or quick summaries. When we talk about the *index*, we mean that labelled axis—it can be the default integer range, a unique identifier such as the crash number, or a time stamp.

Pandas defaults sometimes hide this structure. Selecting a single column with square brackets drops the column dimension and returns a Series; wrapping the column name in a list preserves the DataFrame shape. Grouping, joins, and plotting all depend on keeping track of which form we are working with, so we will call it out each time we switch between them.

Because pandas leans on vectorised operations, we rarely write explicit loops. Instead we apply methods that operate on whole Series or DataFrames at once, which keeps analysis concise and performant. That design also means error messages often mention alignment, broadcasting, or data types; in the troubleshooting callout later in this chapter we decode the ones that show up most frequently in the crash workbook workflow.

Start by importing pandas alongside the plotting stack we will use later:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.options.display.max_columns = 0
```

When you forget an argument, lean on the notebook help shorthand. Typing `pd.read_excel?` in Colab opens the doc-

string panel without executing the cell, mirroring the live demo from the lecture. With the path handy, load the crash sheet and promote the crash identifier to the index so each row mirrors a real-world incident:

```
df = (
    pd.read_excel(
        "NSW_road_crash_2019_2023.xlsx",
        sheet_name="crash"
    )
    .set_index("Crash ID")
)
```

Immediately check the shape and skim the first records. The crash workbook contains roughly 95,000 rows and around 50 columns, so a quick `df.shape` sanity-check reassures us the upload succeeded. Follow with `df.head()` and `df.columns` to confirm columns like `Degree of crash`, `Number of vehicles`, `Speed limit`, and the reporting timestamps appear as expected. Using `df.describe()` on numeric fields highlights skew: some officers left distances blank, producing zeros, while others used a high sentinel such as 999,000 km. Flag those anomalies in your notebook so the team agrees they represent missing data rather than real physics.

Pair `df.head()` with `df.info()` and `df.dtypes` to confirm data types before you engineer features; mismatched types often cause downstream joins to fail.

Accessibility spotlight. Share a textual summary alongside screenshots when briefing stakeholders: list the headline column counts, noteworthy null markers, and key categories

so readers who rely on screen readers do not miss the context hidden in tables.

Section summary

- Use the inline help in notebooks to recall loader arguments on demand.
- Promote unique identifiers to the index immediately after loading; it simplifies joins later.
- Inspect shape, column names, and descriptive statistics before proceeding to ensure nothing silently dropped during import.

Self-check questions

1. Which pandas method reveals whether unexpected sheets exist in the workbook before you call `read_excel`?
   Hint: explore `pd.ExcelFile` to list sheet names.

2. How would you document the decision to treat 999,000 km as missing data so future analysts maintain the same cleaning rule?
   Hint: think about a project README or an inline comment near the imputation code.

Solutions and discussion

1. Use `pd.ExcelFile("NSW_road_crash_2019_2023.xlsx")` then `.sheet_names` to preview sheets. This avoids unexpected tabs derailing imports.

2. Record the assumption in two places: add a short note in the project README that explains the sentinel value and place an inline comment near the cleaning code (for example, where you replace 999,000 with `NaN`). Future collaborators can then confirm the rationale before they tweak the pipeline.

19.2 Separate Series from DataFrames when selecting features

Distinguish one-dimensional Series operations from multi-column DataFrame workflows so selections, summaries, and filters stay predictable.

Learning objectives

After this section you will be able to:

- split the crash dataset into feature and target frames using column lists or drop-based exclusions;
- call Series-specific helpers such as `value_counts()` (which summarises frequencies, optionally as proportions when `normalize=True` instructs pandas to show ratios instead of raw counts) to understand class balance;
- craft boolean masks (filter conditions that evaluate to `True` or `False` for each row) that filter DataFrames without losing the index alignment between inputs and targets.

Prerequisites

Ensure you can:

- interpret the difference between `df["column"]` (Series) and `df[["column"]]` (DataFrame);
- use Python lists to reference multiple column names reliably;
- explain why leaking outcome columns (for example, `Number killed`) into the feature set compromises evaluation.

Begin by isolating the target outcome. The `Degree of crash` column encapsulates whether an incident was fatal, caused injuries, or only damaged property. Selecting it with single brackets returns a Series, which makes downstream methods such as `value_counts()` available:

```
y = df["Degree of crash"]
y.value_counts(normalize=True)
```

That quick tally mirrors the lecture finding: only a sliver of rows are fatal, most involve injuries, and a substantial minority are tow-aways without casualties. Logging the proportions early helps justify baseline models (for example, a "predict injury every time" dummy) before we celebrate more sophisticated scores.

For the feature matrix, pick the context columns that plausibly influence severity. One option mirrors the live coding example:

```
feature_columns = [
    "Street type",
    "Speed limit",
    "Type of location",
```

```
    "Weather",
    "KSU type",
    "First impact type",
]
X = df[feature_columns]
```

Alternatively, drop leakage columns by name to keep the rest of the schema:

```
X = df.drop(
    columns=["Degree of crash", "Number killed"],
    axis=1
)
```

Both strategies retain a DataFrame with the same index as `y`, preserving row alignment for modelling. Use boolean masks to focus on specific scenarios without copying data:

```
fatal = df[df["Degree of crash"] == "Fatal"]
night_rain = df[
    (df["Weather"] == "Rain") &
    (df["Crash time"].dt.hour >= 18)
]
```

Because pandas maintains index alignment, you can join summaries (for example, average speed limit by region) back onto `X` without resorting to merges. Remember to use `.copy()` when creating derived slices you intend to mutate; this avoids the "SettingWithCopy" warnings that surface when chained indexing mutates a view.

> **Troubleshooting clinic.** Keep these fixes nearby when selections misbehave:

- *Empty mask.* Print `df["Weather"].unique()` to check spacing or casing, then normalise with `.str.strip()`and `.str.casefold()`.
- *SettingWithCopy warnings.* Call `.copy()` on any slice you plan to mutate so pandas works on a fresh DataFrame instead of a view.
- *Unexpected type errors.* Inspect `df.dtypes` to confirm categorical columns have consistent types. Mixed integers and strings often appear when spreadsheets store placeholders such as "N/A".
- *Misaligned joins.* Reset or set the index explicitly on both frames before joining to ensure the alignment columns match; mismatched indexes introduce `NaN`s.

Section summary

- Treat targets as Series so you can leverage Series-specific utilities such as `value_counts()` and `map()`.
- Build feature DataFrames either by explicit inclusion lists or by dropping leakage columns; both patterns maintain index alignment.
- Compose boolean masks carefully and normalise categorical text before comparison to avoid silent mismatches.

Self-check questions

1. Why does `df[["Degree of crash"]]` return a DataFrame while `df["Degree of crash"]` returns a Series, and when would you prefer each form?

Hint: think about integration with scikit-learn versus descriptive statistics.

2. How can you guard against accidental target leakage when you maintain a long drop-column list?
   Hint: consider centralising the list of forbidden columns in a shared constant or config file.

Solutions and discussion

1. Single brackets return a Series because pandas assumes you want the column without its table structure; double brackets keep the DataFrame shape. Reach for the Series when you need Series-only helpers such as `value_counts()` or `map()`, and keep the DataFrame when libraries like scikit-learn expect two-dimensional input.

2. Store the forbidden columns in a constant (for example, `PROTECTED_COLUMNS`) and reuse it wherever you build features. Centralising the list keeps the drop logic consistent and makes code review easier when new target-like fields appear.

19.3 Engineer ratios and visualise numeric relationships

Create interpretable features from the crash counts and use Seaborn pair plots to surface correlated risk factors.

Learning objectives

After this section you will be able to:

- derive ratio-style features (for example, people injured per vehicle) that distinguish high-severity crashes;
- handle division safely when counts include zeros or missing values;
- generate and interpret Seaborn pair plots on filtered subsets without overwhelming your notebook session.

Prerequisites

Confirm you can:

- use `.assign()` to add new columns to a DataFrame without mutating the original;
- recognise when to replace zeros with `NaN` before computing ratios;
- read scatterplot matrices and histograms to comment on monotonic or clustered patterns.

Crash reports bundle raw counts: number of vehicles, number of people injured, counts of serious versus minor harm, and so on. Ratios help us reason about severity. For example, a two-vehicle incident with multiple serious injuries deserves prioritised investigation compared with a single-vehicle tow-away. Build those derived signals defensibly:

```
# Clip vehicle counts to avoid div by zero
vehicles = df["Number of vehicles"].clip(lower=1)
```

```
# Fill missing injury counts with zero
injured = df["Number of people injured"].fillna(0)
serious = df["Number seriously injured"].fillna(0)

# Compute injury ratios
inj_per_veh = injured / vehicles
serious_rate = serious / vehicles

# Add the new features to main dataframe
df["injuries_per_vehicle"] = inj_per_veh
df["serious_injury_rate"] = serious_rate
```

Clipping the denominator at one (`clip` limits values to the range you supply) avoids division-by-zero errors when tow-away records log zero vehicles, and filling missing counts with zero keeps the ratios grounded. Document the assumptions (for example, "missing injuries imply none reported") in your lab notes so reviewers can challenge them if local reporting rules differ.

With engineered features in place, explore numeric relationships using Seaborn. Pair plots reveal which ratios cluster by severity, but rendering the full 95,000-row dataset is slow. Sample first, just as we throttled the demo during the lecture:

```
sample = df.sample(n=5_000, random_state=8)
numeric_view = sample[[
    "Number of vehicles",
    "Number of people injured",
    "Number seriously injured",
    "injuries_per_vehicle",
    "serious_injury_rate",
]]
```

Run `uv run Lectures/Week8/generate_crash_pairplot.py` to regenerate. The script samples the crash workbook and writes both `pairplot.png` (slides) and `crash-pairplot.png` (textbook).

Figure 19.1: Seaborn pair plot of crash injury counts and engineered ratios. Alt-text: "Scatter and histogram matrix showing most crashes clustered near zero injuries, with a thin diagonal line where injury counts and ratios rise together."

```
sns.pairplot(
    numeric_view, diag_kind="hist", corner=True)
plt.suptitle("Injury counts and ratios", y=1.02)
plt.show()
```

> **For CS Students.** Estimate the cost of these transformations. Counting operations like `value_counts()` run in roughly $O(n)$ time because they touch each row once, while wide joins and pair plots scale closer to $O(n^2)$ as the visualisation compares every point with every other. When datasets grow past a few hundred thousand rows, budget time for chunked loading or downsampling so notebooks stay responsive.

Expect to see tight clusters near zero (no injuries) and narrow bands where serious-injury rates climb with vehicle counts. Highlight those findings in margin notes so the Week 8 lab can pick up the story when fitting classifiers.

Try this variation. Run the same pair plot separately for urban versus rural locations (use the `Urban classification` column) to check whether the injury ratios diverge. The lecture audience spotted hints of this split; plotting it explicitly informs targeted road-safety recommendations.

Section summary

- Ratios derived from raw counts often express severity better than absolute numbers; clip denominators and fill missing values deliberately.
- Sampling before plotting keeps Seaborn responsive while preserving the overall pattern of correlations.
- Use pair plots to validate intuitive relationships before investing time in more complex models.

Self-check questions

1. How would you adapt the ratio pipeline for crashes involving pedestrians, where the number of vehicles may legitimately be zero?
   Hint: consider a separate denominator that clips at one for vehicles but allows a non-zero count of people involved.

2. Which additional numeric columns (for example, `Speed limit`) would you add to the pair plot to test the "speed increases fatal risk" hypothesis from the lecture?
   Hint: think about scaling or encoding categorical speed limits into integers first.

Solutions and discussion

1. Replace the denominator with a blended measure: keep `vehicles_clipped` for vehicle-focused incidents but add a `people_involved` denominator that clips at one for pedestrian counts. Switch to that denominator when `Number of vehicles == 0` so you avoid inflating ratios while still capturing serious harm.

2. Enrich the pair plot with `Speed limit` converted to numeric form (for example, map categorical labels to integers) and consider including `Average age of drivers` or `Time of day`. These variables provide context for whether higher speeds or night-time conditions align with more severe outcomes.

19.4 Bridge to the Week 8 practical

Carry the cleaned, feature-rich DataFrame into the upcoming lab so modelling can focus on evaluation rather than wrangling.

Spend a moment documenting the pipeline you assembled: list the import commands, the index promotion, the engineered ratios, and any filtering rules you applied (such as sampling or text normalisation). Save the notebook with those notes so your Week 8 practical team can reproduce the setup without replaying the entire lecture. During the lab we will reuse `X`, `y`, and the engineered ratio columns as inputs to scikit-learn pipelines, closing the loop between exploratory pandas work and the deployment-focused pipelines in chapter 21.

Section summary

- Share reproducible notebooks that encapsulate both the code and the rationale for each cleaning step.
- Keep the engineered ratios alongside the raw counts; the lab may compare model performance with and without them.
- Cross-reference the pipeline chapter so readers see how exploratory ratios feed into production-ready preprocessing.

Self-check prompt. Before the practical, rehearse explaining the wrangling steps aloud: how did you load the Excel file, which anomalies did you flag, how did you separate Series and DataFrames, and which ratios told the most compelling story about crash severity? Being able to narrate the workflow builds confidence when classmates ask for help.

Chapter 20

Matplotlib Foundations for Exploratory Plots

Chapter overview

We turn the Week 9 lecture demo into a reliable Matplotlib recipe that keeps our exploratory charts tidy and reproducible. The chapter covers the figure–axes mental model, scatter plot conventions, layout adjustments, and the shortcuts available through pandas. We finish with export patterns so every chart can move from the notebook into reports without pixelation.[1]

20.1 Adopt the figure–axes workflow

Create the figure and axes explicitly so every plot call has a predictable canvas.

[1] Hands-on lab: Week 9 Matplotlib activity (forthcoming).

Learning objectives

After this section we will be able to:

- import `matplotlib.pyplot` using the common `plt` alias alongside pandas and NumPy;
- instantiate a figure and axes pair with `plt.subplots()` and reason about the tuple return signature;
- control the canvas size with the `figsize` argument and explain how on-screen scaling differs from print layouts.

Prerequisites

Make sure we can:

- read tabular data into a pandas `DataFrame`;
- run notebooks in Google Colab or a local Jupyter environment;
- interpret Python functions that return multiple values.

Matplotlib thinks in terms of a *figure*—the whole image—and one or more *axes* objects that host individual charts. Calling `fig, ax = plt.subplots(figsize=(6, 4))` returns both components so we can customise them directly. The tuple unpacking mirrors earlier tools such as `train_test_split`, grounding students who have seen multi-value returns before.

The `figsize` argument uses inches; Matplotlib rescales the display to fit our notebook window but retains the specified proportions when we save to disk. On a laptop, a 20 inch width still shrinks to the available viewport, yet the exported PNG respects those physical dimensions. We can confirm the types with `type(fig)` and `type(ax)` if the tuple feels abstract

at first; Matplotlib reports a `Figure` and `AxesSubplot` respectively, reassuring us that we have the expected handles for later methods.

To make the handles tangible, build two axes at once: one for an environmental snapshot and another for a workplace survey. The code below labels each axis according to the story it tells so we can reference them later when styling.

Example: paired scatter layout.

```
import pandas as pd
import matplotlib.pyplot as plt

air_quality = pd.DataFrame({
    "pm25": [11.2, 13.4, 9.8, 15.1, 12.0],
    "tree_canopy": [28, 24, 35, 19, 31],
    "city": [
        "Harbour City", "River Junction",
        "Coastal Ridge", "Lakeside", "Sunset Hills"
    ],
})

commute = pd.DataFrame({
    "remote_days": [0, 1, 2, 3, 4],
    "median_commute": [62, 58, 54, 51, 49],
})

fig, (ax_air, ax_commute) = plt.subplots(
    1, 2, figsize=(10, 4)
)

ax_air.scatter(
    air_quality["pm25"],
```

```
    air_quality["tree_canopy"],
    s=air_quality["tree_canopy"] * 8,
    c=air_quality["pm25"],
    cmap="viridis",
    alpha=0.85,
)
ax_air.set(
    xlabel="PM2.5 concentration (µg/m³)",
    ylabel="Tree canopy coverage (%)",
    title="Air quality versus canopy",
)

ax_commute.scatter(
    commute["remote_days"],
    commute["median_commute"],
    color="#34495e",
    s=160,
)
ax_commute.set(
    xlabel="Remote work days per week",
    ylabel="Median commute time (minutes)",
    title="Remote flexibility and commute",
)

fig.suptitle("Environment and commute")
fig.tight_layout()
```

Running the snippet creates a figure with two clearly labelled axes, shown in Figure 20.1. Naming the axes variables `ax_air` and `ax_commute` makes subsequent styling reads like prose, which is especially helpful when sharing notebooks with collaborators.

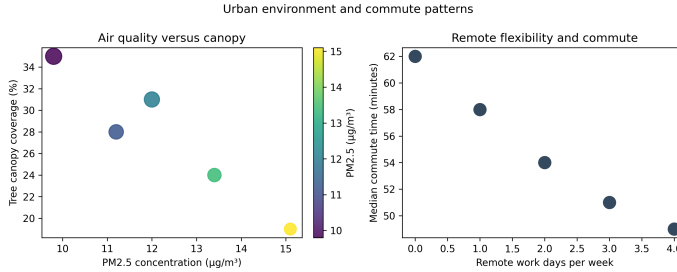


Figure 20.1: Two scatter plots generated from the same figure: the left axis compares particulate matter with tree canopy coverage across five cities, while the right axis charts remote-work days against median commute times. Marker size encodes canopy coverage, and all labels come from explicit axis methods.

Accessibility spotlight. When explaining the figure–axes split verbally, accompany the description with a caption that summarises how each object contributes to the final chart. That narration helps readers who rely on screen readers to picture the relationship before we dive into code.

Section summary

- Import pyplot as `plt` and unpack `plt.subplots()` into figure and axes variables every time.

- Treat `figsize` as a print-setting: it locks proportions for exports even if the notebook preview scales.

- Lean on Python's tuple unpacking intuition from earlier data-splitting utilities.

Self-check questions

1. Why does `plt.subplots()` return two objects, and how does that compare with `train_test_split`?
   Hint: think about keeping handles for later method calls.

2. How would you explain the difference between on-screen scaling and the saved figure size to a teammate preparing print assets?
   Hint: relate pixels in the browser to the inch-based `figsize` setting.

20.2 Build scatter plots with axis methods

Use the axes handle to plot, label, and adjust markers with confidence.

Learning objectives

After this section we will be able to:

- call `ax.scatter(x, y, s=...)` to create dot plots anchored to our axes;
- control marker sizing with the `s` argument and reason about how extreme values affect readability;
- label axes and titles with `set_xlabel`, `set_ylabel`, and `set_title`;
- tighten layouts with `fig.tight_layout()` to avoid clipped text when multiple plots share a figure.

Prerequisites

Before proceeding we should:

- assemble a tidy DataFrame containing the numeric columns we intend to plot;
- recall Python dictionary-to-DataFrame construction for quick examples;
- understand basic scatter-plot interpretation (cluster spread, trends).

Start with the air-quality axis from Figure 20.1. Because the DataFrame tracks particulate matter, tree canopy, and city names, we can encode canopy coverage as marker size and colour each dot by its PM2.5 reading. That single call to `ax_air.scatter` communicates three variables at once without overwhelming readers. Rotate to the commute dataset on the right-hand axis to illustrate a completely different use case: smaller, uniformly coloured markers that highlight the downward trend between remote-work flexibility and travel time. Having two contexts in the same figure reinforces that the axes workflow adapts to environmental monitoring, workforce analytics, or any other domain we bring to class discussion.

Once `ax.scatter` draws the points, experiment with `s`: small values such as 1 produce hairline dots suited to dense datasets, while large values like 10 000 swallow nearby observations. Label axes immediately after plotting; otherwise presentations inherit default column names that may confuse stakeholders. Call `ax.legend()` whenever multiple series share an axis so readers can distinguish the markers. The same pattern applies to `ax.plot` for line charts: the axes handle keeps styling consistent whether we draw dots, connected

trends, or overlays. A concise title reinforces the narrative focus (for example, *Remote flexibility and commute*). Conclude with `fig.tight_layout()` to ask Matplotlib to resize subplots and padding automatically—a guard rail when we add more axes later.

Troubleshooting clinic. If column names contain spaces or apostrophes, copy them carefully or reference them via `df["Tree canopy"]` to avoid syntax errors. A `KeyError` often signals a stray capital letter, so compare `df.columns` against the strings passed into `x` and `y`. When labels overlap despite `tight_layout`, rotate tick labels with `ax.tick_params(axis="x", labelrotation=45)` so values remain legible. If a `TypeError` complains about non-numeric data, convert the columns with `pd.to_numeric(errors="raise")` so unexpected strings surface early.

Performance note. Large geospatial or telemetry datasets can overwhelm scatter plots. Consider plotting a stratified sample, aggregating to hourly or daily summaries, or enabling rasterisation with `ax.scatter(..., rasterized=True)` so exports stay responsive. Combining `alpha=0.5` with smaller marker sizes reveals dense clusters without freezing the notebook.

Alt-text practice. Summarise each scatter in plain language (for example, "Dark-blue circles trend upward as ages increase, showing taller students in older cohorts") so any exported figure includes context for screen-reader users.

Section summary

- Plot through the axes handle to keep all styling in one place.
- Moderate `s` values so dots stay visible without smothering neighbours.
- Always apply layout tightening before exporting or sharing the notebook.

Self-check questions

1. How would you justify a chosen `s` value when communicating with a non-technical audience?
   Hint: tie marker size to the story you want the trend to tell.

2. When should you choose axis method calls over letting pandas infer labels for you?
   Hint: consider how much control you need over titles, colours, and later annotations.

20.3 Resize figures and export publication-ready images

Treat figure size, DPI, and file formats as part of the modelling workflow.

Learning objectives

After this section we will be able to:

- resize figures deliberately for notebooks versus print media;
- save figures via `fig.savefig()` with `dpi` and `bbox_inches` to preserve layout;
- choose between PNG and PDF outputs based on downstream usage;
- use `pathlib.Path` to build cross-platform export paths.

Prerequisites

We should already know how to:

- navigate the Colab or local file browser;
- recognise common raster versus vector file formats;
- manage Python imports from the standard library.

Doubling `figsize` values (for example, `figsize=(12, 8)`) creates more breathing room for annotations, yet the notebook preview may still shrink the chart to fit the browser pane. Shrinking `figsize` has the opposite effect: fonts appear enormous because the same pixel budget represents a physically smaller canvas. Discuss these trade-offs with collaborators before finalising dimensions for slides or print.

When the layout looks right, export through `fig.savefig` instead of right-clicking the inline image. Use `pathlib.Path` to build the output path, set `dpi=300` for sharp printouts, and use `bbox_inches="tight"` to trim excess whitespace. Swap the extension for `.pdf` when vector artwork suits the destination (for example, design software or LaTeX). In hosted environments such as Colab, remind teams that files land in the session storage; download them before the runtime resets.

Section summary

- Match `figsize` and `dpi` to the medium where the figure will live.
- Prefer `fig.savefig` over manual downloads to guarantee repeatable exports.
- Build file paths with `pathlib` so the script runs cleanly on any operating system.

Self-check questions

1. Which parameters would you tweak when preparing a figure for print in a journal versus a slide deck?
   Hint: focus on `figsize`, `dpi`, and `bbox_inches`.

2. How can `pathlib` simplify collaboration across Windows, macOS, and Linux machines?
   Hint: consider how it abstracts away backslash versus forward-slash differences.

20.4 Leverage pandas plotting shortcuts

Choose between axes methods and DataFrame helpers depending on how much automation you need.

Learning objectives

After this section we will be able to:

- call `df.plot.scatter(x=..., y=..., ax=ax)` as a concise alternative;

- explain how pandas infers axis labels from column names and when to override them;
- weigh the trade-offs between brevity and explicit control when choosing a plotting API.

Prerequisites

Before adopting the shortcut we should:

- understand keyword arguments in pandas plotting methods;
- be comfortable passing Matplotlib axes into helper functions;
- recognise when automatic labelling may expose unfriendly column names.

Example: power-grid planning with pandas.

```
import pandas as pd
import matplotlib.pyplot as plt

energy = pd.DataFrame({
    "solar_share": [0.18, 0.25, 0.32, 0.41, 0.29],
    "evening_demand": [420, 390, 370, 355, 400],
    "region": [
        "Coastal Belt", "Highland", "River Plains",
        "Sun Corridor", "Northern Range"
    ],
})

fig, ax = plt.subplots(figsize=(6, 4))
energy.plot.scatter(
```

```
    x="solar_share",
    y="evening_demand",
    s=energy["evening_demand"],
    color="#d35400",
    ax=ax,
)

for _, row in energy.iterrows():
    ax.annotate(
        row["region"],
        (row["solar_share"],
         row["evening_demand"]),
        textcoords="offset points",
        xytext=(6, -10),
    )

ax.set_title("Solar vs evening peak demand")
ax.set_xlabel("Share from solar")
ax.set_ylabel("Evening demand (MW)")
fig.tight_layout()
```

Pandas delegates to Matplotlib under the hood, so `df.plot.scatter` reaches the same visual outcome with fewer lines of code. Because the DataFrame already knows about its columns, we only supply the `x` and `y` column names plus the target axes. Labels default to the column headers, saving time during rapid exploration. When presenting findings, we still favour explicit `set_xlabel` calls to translate shorthand like `solar_share` into audience-ready phrasing.

Choosing between approaches hinges on intent. The axes-first pattern centralises styling decisions and keeps the Matplotlib API visible to newcomers. The pandas shortcut pri-

oritises velocity when exploring dozens of combinations in a notebook. Encourage teams to start with the explicit approach, then switch to the concise form once they internalise how keyword arguments map to Matplotlib behaviour.

Collaboration tip. Commit both versions of the plotting helper to version control and note the expected arguments in a docstring. That way, teammates reviewing a pull request can swap between the explicit axes workflow and the pandas shortcut without rewriting surrounding analysis cells.

Section summary

- Use pandas plotting helpers for quick experimentation, but keep axes handles for consistent styling.
- Override inferred labels before sharing outputs beyond the notebook.
- Remember that both APIs rely on Matplotlib, so knowledge transfers between them.

Self-check questions

1. When does the pandas shortcut save you time, and when might it hide important styling options?
   Hint: balance exploratory speed against presentation polish.

2. How could you refactor a notebook so teammates can choose either plotting style without rewriting code?
   Hint: wrap plotting logic in helper functions that accept an `ax` *argument.*

20.5 Connect Matplotlib workflows to Tableau dashboards

Spot the similarities so the lecture's Tableau demo feels familiar rather than foreign.

Learning objectives

After this section we will be able to:

- recognise how Matplotlib's figure–axes model maps to Tableau worksheets and dashboards;
- describe when to start in Matplotlib for programmable control versus Tableau for drag-and-drop exploration;
- prepare exports with Tableau in mind so colleagues can blend notebook and BI assets.

Prerequisites

Before making the jump we should:

- feel comfortable exporting figures from Matplotlib;
- know the basics of launching Tableau and connecting to a CSV file;
- understand the audience expectations for an interactive dashboard versus a static report.

Tableau's worksheet is conceptually close to a single Matplotlib axes: each hosts one visual that we configure through marks, encodings, and annotations. A Tableau dashboard mirrors a multi-axes Matplotlib figure where we compose several plots into a single view. Translating a scatter from

Matplotlib to Tableau becomes easier when we articulate the encoding choices explicitly—what data fields drive the x- and y-positions, how colour groups categories, and which tooltips we rely on for context. Explaining those ingredients up front means a teammate can rebuild the insight in Tableau with confidence.

Deciding where to begin depends on our constraints. Matplotlib shines when we need version-controlled scripts, automated report generation, or statistical overlays that go beyond Tableau's default calculations. Tableau, on the other hand, accelerates collaborative exploration and interactive filtering during stakeholder workshops. Many teams prototype in Tableau to surface interesting slices, then translate the final story into Matplotlib for publication alongside the code that sourced the data.

When handing work between tools, plan the exports deliberately. Saving a high-DPI PNG or vector PDF from Matplotlib provides assets we can drop into Tableau dashboards as static reference panels. Likewise, exporting Tableau visuals as images or embedding them via Tableau Public can accompany Matplotlib plots in a shared report. Keeping filenames and captions consistent across both ecosystems helps readers track the same narrative thread regardless of the platform.

Section summary

- Think of Tableau worksheets as cousins of Matplotlib axes, and dashboards as composed figures.
- Choose Matplotlib for scripted control and Tableau for interactive exploration depending on stakeholder needs.
- Coordinate exports so figures remain consistent when

they move between notebooks and dashboards.

Self-check questions

1. How would you brief a teammate to rebuild your Matplotlib scatter in Tableau without losing key encodings? *Hint: spell out the roles of position, colour, and annotations.*

2. Which scenarios demand Tableau's interactivity, and when does Matplotlib's scripting pay dividends instead? *Hint: contrast collaborative workshops against automated reporting pipelines.*

Chapter wrap-up

We now have a dependable template for Matplotlib work: import pyplot with confidence, create figures and axes explicitly, build scatter plots with thoughtful sizing, and export the results through code. With the pandas helper in our back pocket, we can alternate between deliberate chart crafting and quick exploratory passes. The Tableau bridge keeps the lecture's BI demo within reach, positioning us to combine scripted visuals with interactive dashboards. The forthcoming Week 9 activity will extend these foundations into multi-plot layouts and side-by-side comparisons that deepen the Matplotlib–Tableau dialogue.

Chapter 21

Build Reliable Pipelines in scikit-learn

Chapter overview

We now translate the tidy workflows we perfected in Orange into Python code. This chapter explains how scikit-learn pipelines help us keep preprocessing, imputation, and modelling in lock-step so data never drifts between steps. We begin with a design checklist for mapping each feature to its transformation, then demonstrate imputation strategies for numeric and categorical columns, and finish with evaluation practices that prevent leakage. The narrative builds on the guard rails in chapter 14 and prepares us for the Week 10 laboratory where we recreate these pipelines in a notebook.

21.1 Design column-aware pipelines before writing code

Plan transformations around the dataset schema so every column receives the right treatment and nothing slips through unprocessed.

Learning objectives

After working through this section, you will be able to:

- sketch a preprocessing plan that separates numeric, categorical, and derived features;[1]
- choose the appropriate scikit-learn transformer for each feature type before opening a notebook;[2]
- explain why pipelines reduce data leakage by bundling preprocessing with model training.

Prerequisites

Confirm that you can:

- read schema documentation or inspect a DataFrame to identify column types;
- interpret summary statistics to spot skew, missingness, and outliers;
- describe the purpose of the transformations we used in Orange (for example, normalisation, one-hot encoding, feature selection).

[1] Week 10 lecture slides, frames 12–20, introduce the column mapping diagrams that anchor this plan.

[2] See the same slides for the mapping between Orange widgets and scikit-learn classes.

> Export the Week 10 column-planning slide to `Lectures/Week10/column-plan-overview.png` so the textbook can show the annotated schema blueprint referenced throughout this section.

Figure 21.1: Column-planning slide from Week 10 Lecture 10a. Each colour-coded band in the slide maps to a branch in the pipeline diagram, reinforcing why we sketch the schema before coding.

The scikit-learn pipeline API mirrors the Orange canvas metaphor: we draw the sequence before we click the buttons. Start by listing every column under three headings—numeric, categorical, and flags that should be dropped. Annotate the list with any bespoke steps such as log transforms for skewed revenue fields or domain-specific encodings. Once the plan looks complete, translate it into a `ColumnTransformer` definition. The snippet below uses list comprehensions to keep the schema declarative.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import (
    OneHotEncoder,
    StandardScaler
)

numeric_features = [
    "bedrooms",
    "bathrooms",
    "land_area"
]
categorical_features = ["suburb", "zoning"]
```

```
preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(),
         numeric_features),
        ("cat", OneHotEncoder(
            handle_unknown="ignore"),
         categorical_features),
    ],
    remainder="drop",
)
```

Document each choice in the lab journal so teammates understand why certain fields were dropped or grouped. That transparency keeps the upcoming imputation logic defensible.

Accessibility spotlight. When drafting the column plan, share it in a text-based format (for example, a table in the lab wiki) rather than an image so screen-reader users can review and edit the schema.

Section summary

- Treat the pipeline diagram as a design artefact before coding.
- Use `ColumnTransformer` to keep column mappings explicit and reproducible.
- Record transformation decisions so future readers can audit the workflow.

Self-check questions

1. What risks do we invite if we start coding without a column plan?
   Hint: Think about inconsistent scaling and dropped columns.
   Common misconception: Pipelines automatically infer the correct schema.

2. Why does setting `handle_unknown="ignore"` on `OneHotEncoder` matter in production?
   Hint: Consider new categories appearing after deployment.
   Common misconception: The model will crash only when retrained, not during inference.

21.2 Select imputation strategies that match the data story

Choose imputers that respect the distribution of each feature and document when synthetic values enter the dataset.

Learning objectives

After this section you will be able to:

- distinguish between mean, median, constant, and learned imputations and justify each choice;[3]
- encode missingness explicitly when it carries signal (for example, "unknown" categories or zero-inventory flags);

[3]Week 10 lecture slides, frames 24–31, compare these strategies with their Orange equivalents.

- audit the volume of imputed entries to set expectations with stakeholders.

Prerequisites

Before proceeding, ensure you can:

- interpret missing value heatmaps or percentage summaries;
- explain when median imputation outperforms the mean (for skewed numeric fields);
- describe how we used imputation widgets in Orange during Week 2 and Week 5 labs.

Missing data is not a single problem. Sometimes the gap results from a genuine absence (for example, optional survey questions); other times it signals a broken integration. For numeric columns with symmetric distributions, a mean imputer keeps the totals balanced. Skewed variables, such as property prices, benefit from the median or from domain-aware defaults (e.g. replacing missing bedrooms with the mode). Categorical columns often deserve an explicit "Unknown" category so the model can learn whether absence correlates with outcomes.

Worked example: diagnosing churn risk

To see the planning mindset in action, let us swap the housing sample for the Telco Customer Churn dataset.[4] Begin by splitting the schema into three lists:

[4]We revisit this dataset during the Week 10 lab; it ships with features such as contract type, tenure in months, monthly charges, and a churn label.

- numeric behavioural signals like `tenure`, `monthly_charges`, and `total_charges`;
- categorical descriptors such as `contract`, `internet_service`, and `payment_method`;
- high-cardinality identifiers (for example, customer IDs) that we will drop rather than encode.

Stakeholders from customer success flagged that missing `total_charges` values usually occur when a service is brand new, so we decided to impute using the median but also keep an indicator flag. The code below mirrors the agreed blueprint and adds a memory-friendly sparse output for one-hot encoding so the 40-plus contract permutations do not bloat RAM in production notebooks.

We combine these ideas by adding imputers inside the column pipeline. Scikit-learn lets us chain steps within each branch so we never forget to scale after filling values.

```
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

num_pipeline = Pipeline([
    ("impute", SimpleImputer(
        strategy="median")),
    ("scale", StandardScaler()),
])

cat_pipeline = Pipeline([
    ("impute", SimpleImputer(
        strategy="most_frequent")),
    ("encode", OneHotEncoder(
```

```
        handle_unknown="ignore")),
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", num_pipeline, numeric_features),
        ("cat", cat_pipeline, categorical_features),
    ]
)
```

After fitting, inspect the `SimpleImputer.statistics_` arrays to confirm the replacement values match expectations. Summarise the number of imputed records per feature in your project notes; stakeholders deserve to know when a quarter of the dataset relies on synthetic entries.

Try this variation. If missingness correlates strongly with the target, add a boolean "was_missing" feature through the `add_indicator=True` option on `SimpleImputer`. This mirrors the indicator columns we toggled in Orange and gives the downstream model an explicit flag to reason about.

Stakeholder conversation starter. Before finalising the imputers, share a quick summary with domain experts: "We imputed 6% of `total_charges` using the median because new contracts often lack a first invoice. We added a `total_charges_missing` flag so finance can quantify synthetic revenue." Conversations like this make synthetic data transparent and build trust with teams who rely on the churn dashboard.

Troubleshooting clinic.

Strategy	Best suited for	Example rationale
Mean	Approximately symmetric data	Hospital bed-occupancy with stable seasonal averages.
Median	Skewed or heavy-tailed data	Property prices where extremes drag mean off centre.
Constant	Known safe defaults	Power-usage with "estimated consumption" flag.
Most frequent	Low-cardinality categoricals	Device platform; missing means common option.
Learned / model-based	Complex interactions	Triage where iterative imputation uses labs.

Table 21.1: Choosing an imputation strategy depends on both the distribution and the story you need to tell stakeholders. Use the table to explain to domain experts why you favoured a particular replacement.

- **Shape mismatches:** A "Found array with dim 3" error usually means a transformer returned a dense matrix while the model expected sparse input. Set `sparse_output=False` on `OneHotEncoder` or wrap downstream estimators that only handle dense arrays.
- **Unknown categories:** If production data introduces new payment methods, ensure `handle_unknown="ignore"`. For legacy models, patch them with `OneHotEncoder` set to `categories="auto"` and `handle_unknown="ignore"`. Refit and redeploy before the log fills with failed predictions.
- **Memory pressure:** When pipelines balloon past a few hundred thousand rows, switch to histogram-based gradient boosting classifiers or regressors (they accept sparse matrices) and stream evaluation with `partial_fit` on compatible estimators. Profiling with `memory_profiler` pinpoints transformers that silently densify arrays.

Advanced imputation options. For projects where the signal lives in subtle correlations, graduate beyond `SimpleImputer`. `IterativeImputer` cycles through regressors to estimate each missing column, while `KNNImputer` searches the nearest neighbours in feature space. Reserve these heavier tools for datasets with enough rows (think hospital records or credit-risk ledgers) and budget extra compute time for cross-validation.

Section summary

- Match imputation strategies to the distribution and business context of each column.
- Chain imputers and scalers inside pipelines so filled values flow through the same transformations as observed ones.
- Record replacement statistics and share them with the project team.

Self-check questions

1. Why might a constant value of zero be misleading when imputing missing stock counts?
   Hint: Consider the difference between "zero items remaining" and "information not recorded."
2. How would you convince a stakeholder that median imputation is safer than the mean for house prices?
   Hint: Reference the long right tail we observed in the Week 5 land value case study.

Progressive exercises

Tackle these in order to cement the workflow before lab time:

1. **Column audit:** Take a fresh dataset—for example, the open ICU mortality benchmark—and sketch the numeric/categorical/drop lists with one sentence explaining each decision.

2. **Imputer rationale:** For the same dataset, note which fields deserve median, constant, or learned imputers and justify them using Table 21.1.

3. **Pipeline evaluation:** Implement the plan in a notebook, add `add_indicator=True` to the most critical missing fields, and run `cross_validate`. Capture both accuracy and memory usage so you can report trade-offs to stakeholders.

21.3 Bundle preprocessing and modelling for leak-free evaluation

Train models through a single pipeline so cross-validation repeats every transformation consistently and we can deploy the exact same recipe.

Learning objectives

By the end of this section, you will be able to:

- wrap preprocessing and estimators in one `Pipeline` object for training and prediction;[5]

[5]Week 10 lecture slides, frames 36–44, demonstrate this pattern with logistic regression.

- evaluate the pipeline with cross-validation and interpret variance across folds;[6]
- export the fitted pipeline for deployment without rerunning manual preprocessing steps.

Prerequisites

Make sure you can:

- explain the difference between training, validation, and test splits;
- compute accuracy, precision, recall, and RMSE depending on the task;
- use scikit-learn's `cross_val_score` or `cross_validate` helpers.

Wrapping everything into a pipeline means we call `fit` once and trust that scikit-learn handles the sequence. The code below extends the preprocessing we built earlier with a gradient boosting classifier, but the same pattern works for regression or linear models.

```
from sklearn.ensemble import (
    HistGradientBoostingClassifier
)
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_validate

model = Pipeline([
```

[6] Cross-validation guard rails from chapter 14 still apply—this section shows them in Python.

```
    ("preprocess", preprocess),
    ("clf", HistGradientBoostingClassifier(
        random_state=42
    )),
])

cv_results = cross_validate(
    model,
    X_train,
    y_train,
    cv=5,
    scoring=["accuracy", "roc_auc"],
    return_train_score=False,
)

print(
    cv_results["test_accuracy"].mean(),
    cv_results["test_accuracy"].std()
)
```

Because the pipeline owns the transformations, every fold sees only training data during imputation and scaling. That discipline guards against the accidental leakage that chapter 14 warned us about. Review the mean and standard deviation for each metric to judge stability, and plot them over time as you tweak hyperparameters so the team can spot regressions quickly.

Reporting tip. Log the exact pipeline object with `joblib.dump` and include the file hash in your project documentation. Reproducibility beats screenshots when auditors ask how the model was trained.

Run `uv run Lectures/Week10/peek_inside.py` to regenerate `resources/figures/sms-token-weights.png`.

Figure 21.2: Token-weight visual from the Week 10 notebook. The figure shows how the pipeline's text preprocessing redistributes feature importance, reinforcing why we evaluate the end-to-end pipeline rather than isolated steps.

Computational headroom. Pipelines are only as fast as their slowest transformer. Scaling wide one-hot matrices or training boosted trees on dense features can chew through gigabytes of RAM. Profile the preprocessing step with `memory_profiler` to spot bottlenecks, then consider:

- caching intermediate results with `Pipeline` using `memory=joblib.Memory(...)` when repeated cross-validation reuses the same imputations;
- swapping to `SparseNormalizer` or hashing encoders for ultra-wide categorical spaces;
- chunking evaluation with `HalvingGridSearchCV` once we enter Week 11's hyperparameter sweeps.

Document the runtime alongside metric tables so product managers understand the compute budget they are signing up for.

Section summary

- Pipelines align preprocessing and modelling so evaluation never reuses information improperly.
- Cross-validation on the pipeline yields honest performance estimates and highlights unstable folds.

- Serialising the fitted pipeline simplifies deployment and future audits.

Self-check questions

1. What does it mean if one cross-validation fold scores dramatically lower than the others?
   Hint: Investigate class imbalance or unhandled categories in that split.

2. Why do we prefer exporting the entire pipeline over saving individual transformers?
   Hint: Think about guaranteeing the same preprocessing logic during inference.

21.4 Where to next

The pipeline discipline here underpins the bag-of-words text classifier we build in Week 10 and the hyperparameter sweeps in Week 11. Bring the column plan, imputation log, and serialised pipeline to the upcoming Python lab so you can compare results with classmates and extend the workflow into feature unions or model stacks without redoing the groundwork.

Chapter 22

Tune Text Models with Hyperparameter Search

Chapter overview

Week 11 introduces our first sizeable Python lab after migrating Orange workflows into scikit-learn. This chapter shows how hyperparameter search keeps models honest while a count vectoriser turns raw text into features we can tune. We explain how cross-validation underpins any search routine, then build a reusable pipeline that marries tokenisation, vocabulary management, and linear models. The closing section maps the strategy to the Week 11 notebook so readers arrive ready to compare settings and document their findings.

22.1 Treat hyperparameter search as a structured experiment

Plan search grids around defensible questions and evaluate each candidate with cross-validation rather than a single lucky

split.

Learning objectives

After this section you will be able to:

- describe how cross-validation estimates generalisation performance for each candidate configuration;
- design a search space that balances model flexibility with computation time;
- explain the guard rails that prevent leakage between the validation fold and the test set.

Prerequisites

Confirm that you can:

- interpret cross-validation score summaries (mean, standard deviation, minimum);
- trace how scikit-learn pipelines bundle preprocessing and modelling work from chapter 21;
- document experimental settings in a lab journal or issue tracker.

Hyperparameters control how expressive a model may become before it memorises training quirks. Grid search is the simplest approach: define a dictionary of candidate values, evaluate each with cross-validation, and keep the configuration that delivers the best validation score without inflating variance. Randomised search samples the space instead of enumerating it, which is practical when the grid would otherwise explode combinatorially.

Regardless of the method, treat the search like any other experiment. Start by naming the question: are we exploring regularisation strengths, n-gram widths, or both? Sketch a table that lists each hyperparameter, the rationale for its range, and any computational limits. For example, the Week 11 lab narrows the logistic regression penalty to a short list of inverse regularisation strengths (C values) because lecture demonstrations showed diminishing returns past four trees in a random forest search.

Once the grid is locked, specify a scorer that matches the business objective. Accuracy suffices for balanced sentiment datasets; F_1 or area under the ROC curve better reflect imbalanced complaint logs. When results return, chart both the mean score and its spread. A configuration that wins by a fraction of a percent but doubles variance rarely improves the deployed workflow. Keep the untouched test set for a single, final confirmation run after the search concludes.

Stakeholder check-in. Share the proposed search grid with collaborators before running it. Doing so avoids situations where a teammate expects character 5-grams while you only trial unigrams and bigrams, and it surfaces runtime constraints that might dictate a random search instead of an exhaustive grid.

Section summary

- Formulate the search question first, then encode it as a grid or sampling distribution.
- Cross-validate each candidate so the reported score reflects unseen data, not a single split.

- Present both the mean and spread of validation scores to guide the final choice.

Self-check questions

1. How does increasing the number of cross-validation folds influence the stability of your search results?
   Hint: balance statistical reliability against compute time.
   Common misconception: more folds always improve accuracy without side-effects.

2. When might a randomised search outperform a grid search even on a small dataset?
   Hint: consider interactions across multiple hyperparameters.
   Common misconception: grids always cover the "important" combinations.

22.2 Vectorise text data with reproducible preprocessing

Transform raw documents into structured features while keeping the vocabulary manageable and explainable.

Learning objectives

After this section you will be able to:

- outline the stages of a bag-of-words pipeline (tokenisation, normalisation, vocabulary restriction);

- configure scikit-learn's `CountVectorizer` to match project requirements;

- integrate text features into a pipeline that feeds a linear classifier alongside tabular context.

Prerequisites

Before diving in, ensure you can:

- explain basic natural language concepts such as tokens, stop words, and n-grams;
- load and inspect text datasets in pandas;
- interpret logistic regression coefficients.

The bag-of-words model counts how often each token appears in a document, ignoring order but preserving frequency patterns. We start with `CountVectorizer` because it mirrors the Week 7 uncertainty discussion about reproducibility: fixing the random state and vocabulary keeps the pipeline deterministic when we rerun it or share it with teammates.

Configure the vectoriser with a clear story about your audience. Customer-support tickets might prioritise lowercasing and moderate stop-word removal, while triage notes from a community health clinic may demand a wider n-gram window to capture symptom phrases and abbreviations. In both cases, cap the maximum feature count so the model stays interpretable for reviewers. Pair the vectoriser with a `TfidfTransformer` if relative term importance outweighs raw counts. Keep an eye on memory usage: each additional n-gram expands the sparse matrix, so restrict `max_features` or `max_df` to prune ubiquitous filler words.

When combining text with tabular features, wrap the vectoriser inside a pipeline branch and feed it into a `ColumnTransformer` alongside numeric preprocessing. The

scaffold below illustrates a minimal setup for classifying triage notes while retaining vital-sign summaries from chapter 21.

```
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import (
    CountVectorizer
)
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

text_features = "triage_note"
numeric_features = [
    "heart_rate",
    "respiratory_rate",
    "acuity_score"
]

model = Pipeline([
    ("features", ColumnTransformer([
        ("text", CountVectorizer(
            lowercase=True,       # normalise
            ngram_range=(1, 2),   # phrases
            max_features=5000,    # cap
            min_df=5,             # drop rare
        ), text_features),
        ("numeric", Pipeline([
            ("scale", StandardScaler()),
        ]), numeric_features),
    ])),
    ("clf", LogisticRegression()),
```

```
])
```

Export the fitted vocabulary with `get_feature_names_out()` and save it in version control. This explains which terms the model recognised and eases audits.

Accessibility spotlight. Provide alternate text summaries for any word clouds or heatmaps derived from the vectorised features so readers who rely on screen readers receive the same analytical insight.

Section summary

- Align vectoriser options with the communication needs of your audience and the constraints of the dataset.
- Combine text and tabular preprocessing through `ColumnTransformer` to keep the workflow reproducible.
- Version the learned vocabulary to support audits and fairness reviews.

Self-check questions

1. Why might you increase `min_df` before raising `max_features`?
   Hint: think about rare tokens that add noise to the classifier.
   Common misconception: more features always improve accuracy.
2. How could you adapt the pipeline when new jargon enters the ticket queue mid-semester?

Hint: consider scheduled retraining and vocabulary diffing.
Common misconception: vectorisers automatically absorb unseen words without refitting.

22.3 Coordinate search routines with the Week 11 notebook

Translate the design decisions above into concrete workflow steps for the upcoming lab.

Week 11's notebook guides us through setting up a `GridSearchCV` over the count vectoriser and logistic regression hyperparameters. Prepare by sketching three candidate grids:

- an "express" grid for live demos with two C values and unigram-only features that finishes in roughly 30–45 seconds on an 8-core laptop;
- a "balanced" grid for the main lab, trialling unigrams and bigrams alongside three C values that completes in about 3–4 minutes on the same hardware;
- an "exploratory" grid for homework that tests adding trigram features or swapping in a linear support vector machine, which can stretch to 8–10 minutes on shared lab machines.

Set expectations early by listing the runtime budget and recommended hardware (a quad-core CPU with 8 GB of RAM suffices for the first two grids) so students can choose the right tier. Annotate the notebook with markdown cells that remind students when to downgrade the search if the environment slows down. Close the lab with a comparison table that lists

the best parameters, validation scores, and the top weighted tokens. That summary creates a clean bridge into Week 12's recommendation workshop, where we will revisit ranking metrics such as precision@k and mean reciprocal rank alongside personalised text features.

Beyond grids. Mention Bayesian optimisation libraries such as Optuna or Hyperopt for curious readers. These tools model the objective surface and adaptively sample promising regions, which can trim compute budgets once students outgrow fixed grids.

Connection to practice. Encourage readers to log every search run, even "failed" ones, in the shared lab journal. Hyperparameter work is iterative; leaving a paper trail saves classmates from repeating an unhelpful configuration and demonstrates professional rigor.

Chapter 23

Designing Recommendation Engines

Chapter overview

Week 12 caps the unit with a recommender systems sprint that unites ranking metrics, sparse-matrix tooling, and the governance themes we have revisited all semester. This chapter reframes the problem around top-N recommendations on implicit feedback, assembles a layered pipeline that blends popularity baselines with collaborative and content-based models, and closes with evaluation guard rails and ethics checklists. The final section links directly to the Week 12 practical so we approach the lab ready to ship a trustworthy baseline before exploring advanced models.

23.1 Frame recommender problems around implicit feedback

Surface the right items by modelling top-N rankings, not just average ratings, while respecting catalogue constraints and cold-start realities.

Learning objectives

After this section you will be able to:

- distinguish rating prediction from top-N recommendation and justify why the latter dominates production settings;
- describe the structure of an implicit feedback log and the metadata that supports cold-start mitigation;
- outline the core product constraints — business rules, availability, and regulatory limits — that shape any recommender brief.

Prerequisites

Ensure you can:

- read contingency tables and pivot interactions as reviewed in chapter 6;
- interpret model evaluation metrics from chapter 14;
- discuss how data governance policies influence modelling scope as introduced in chapter 2.

If you are new to the space, think of implicit feedback as the digital equivalent of noticing what someone lingers on in

a shop. We rarely ask shoppers to score every product, yet we can still tell a story from their behaviour: they clicked on a raincoat, watched a how-to video all the way through, and added gardening gloves to the basket but never checked out. Those actions are soft signals rather than direct ratings, so we treat them as clues about interest rather than declarations of love or dislike.

In practice, explicit ratings remain rare. When customers purchase an item, they seldom leave a numerical score; if a venue asks patrons to write a review, perhaps one response arrives for every hundred visits. Instead, our logs contain binary observations: the customer chose chocolate yogurt but ignored the strawberry option. This binary choice reveals a preference ordering without exposing the underlying rating scale the customer might have imagined. We infer only that one item ranked higher than another, which suffices to power collaborative filtering and ranking models.

Recommender systems collect sequences of (u, i, t, c) tuples: a user u, an item i, a timestamp t, and optional context c such as device, locale, or intent. Most entries record implicit signals like clicks, plays, or basket additions instead of explicit star ratings. Because zero entries represent "unknown" rather than dislike, we reshape the problem into ranking the most promising items for each user. Production teams align this to top-N lists: the ten videos to autoplay next, the five articles for a homepage shelf, or the bundle of elective subjects we want to surface in a portal.

Two structural challenges appear immediately. Cold-start users and items lack histories, so we borrow side information. Cohort details, declared interests, or onboarding questionnaire responses enrich user profiles, while item metadata — tags,

textual blurbs, price points, and recency — helps the system reason about fresh catalogue entries. At the same time, long-tail behaviour means most catalogue items sit in sparse rows or columns. Rather than chasing full coverage with one algorithm, we layer signals: a global popularity fallback for coverage, similarity models to personalise mid-tail options, and content features for the newest additions.

Throughout, we respect non-technical rules. Merchandising teams may blacklist items awaiting clearance, policy teams prohibit certain combinations, and operations staff enforce stock limits. Document these constraints alongside the modelling objective before training. Doing so keeps later evaluation honest: a high hit rate on banned items is not a win.

Section summary

- Treat recommender tasks as ranking problems over implicit interactions rather than pure rating prediction.
- Enrich sparse interaction logs with user and item metadata to offset cold-start gaps.
- Encode business and compliance rules as first-class requirements before modelling.

Self-check questions

1. Which contextual attributes would you prioritise collecting to help a new streaming catalogue survive its launch month?
   Hint: balance demographic insight with privacy obligations.

2. How would you explain the difference between "no interaction" and "negative feedback" to a product manager planning KPIs?
   Hint: emphasise that implicit datasets rarely capture true dislikes.

3. A school careers portal wants to nudge students toward internships they might overlook. Which implicit signals could you log ethically, and how would you reassure stakeholders about data use?
   Hint: focus on opt-in browsing history, bookmarking, and transparent messaging.

Worked example: computing item similarity from implicit clicks

To demystify the mechanics, consider a toy implicit-feedback table with five learners browsing a skills platform. Each row records whether the learner viewed a resource in the last fortnight. The platform tracks ten resources: three videos (V1–V3), four practice sets (P1–P4), and three project briefs (B1–B3). We begin by pivoting the log into a binary user–item matrix X where $X_{ui} = 1$ if learner u interacted with item i.

Learner	V1	V2	V3	P1	P2	P3	P4	B1	B2	B3
L1	1	1	0	1	0	0	0	1	0	0
L2	0	1	1	0	1	0	0	0	1	0
L3	1	0	0	1	1	0	0	0	0	1
L4	0	1	1	0	0	1	0	0	1	0
L5	1	0	0	0	1	0	1	0	0	1

We now compute cosine similarity between two items — say V2 and B2 — by taking the dot product of their columns

and dividing by the product of their norms. The column vectors are $v_{V2} = (1, 1, 0, 1, 0)$ and $v_{B2} = (0, 1, 0, 1, 0)$. Their dot product is $1 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 = 2$. Each vector has norm $\sqrt{1^2 + 1^2 + 0^2 + 1^2 + 0^2} = \sqrt{3}$, so the cosine similarity becomes $\frac{2}{\sqrt{3} \times \sqrt{2}} \approx 0.82$. A high score tells us that learners who watch V2 frequently explore B2 soon after.

Repeating this computation for every pair of items gives an item–item similarity matrix. When generating recommendations for learner L3, we look at the items they have already enjoyed (V1, P1, P2, B3) and pull the top neighbours from the matrix. Suppose B2 and P3 emerge with cosine similarities above 0.7 relative to the consumed items. We would recommend those next, while explaining to the stakeholder that the suggestions come from observing which resources tend to co-occur in similar study sessions. This exact workflow scales to thousands of items once we represent X as a sparse matrix, yet the core arithmetic mirrors this small example.

23.2 Build layered recommendation pipelines

Start with interpretable baselines and gradually add collaborative, content-based, and ranking models to improve personalisation.

Learning objectives

After this section you will be able to:

- implement popularity and item–item co-occurrence baselines as dependable first passes;

- compare collaborative filtering, matrix factorisation, and content-based strategies for candidate generation;
- describe how two-stage architectures combine recall and ranking while enforcing business constraints.

Prerequisites

Before diving in, confirm that you can:

- compute cosine and Jaccard similarity scores as practised in chapter 6;
- manipulate sparse matrices or long-form interaction tables in pandas or NumPy;
- explain at a high level how gradient-based optimisation works from chapter 22.

Popularity and trending lists. Count interactions per item over a recent window to create a global popularity baseline. Segment the counts by cohort, geography, or device when different audiences behave differently. Apply a shrinkage factor (a weighted average that gently pulls noisy proportions toward a reliable global mean) $\hat{p}_i = \frac{n_i + m\bar{p}}{n_i + m}$ so rarely observed items do not leap to the top due to a single lucky click. This baseline is not glamorous, yet it sets expectations, offers an immediate fallback when personalisation fails, and highlights coverage gaps.

Item–item co-occurrence. Convert the interaction log into a binary user–item matrix X where $X_{ui} = 1$ when user u touched item i. Compute similarities between item vectors with cosine or Jaccard metrics and score a target item i for user

u by aggregating similarities to the items already consumed. Stakeholders appreciate the resulting "people who liked this also liked that" narratives, and the method thrives on implicit signals. Remind readers that very new items need help from metadata because no interactions exist yet.

Association rules for basket data. For checkout logs or short session histories, frequent-pattern mining provides high-confidence rules of the form $A \Rightarrow B$. Support, confidence, and lift mirror the Orange workflows from chapter 15. When we translate those metrics into recommendation scores, we keep the storytelling power of market-basket analysis while integrating it into the wider engine.

Two-stage architectures. Once baselines behave, combine them into a pipeline with distinct stages. Stage 1 recalls a tractable set of candidates using fast heuristics: popularity, co-occurrence neighbours, matrix factorisation factors, content-based similarity, or random walks on the user–item graph. Stage 2 ranks those candidates with richer features: personalised factors, recency scores, diversity penalties, or outputs from a gradient-boosted tree. A final post-processing pass enforces business rules, ensures a minimum share of long-tail items, and blocks banned content. Think of the pipeline as a conversation between shortlists and rerankers: the first ensures coverage, the second refines relevance.

Matrix factorisation for implicit data. Weighted regularised matrix factorisation (often delivered by alternating least squares) factorises the interaction matrix into user and item embeddings. We treat observed interactions as confidence-

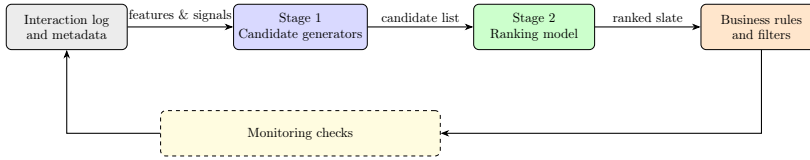


Figure 23.1: Two-stage recommendation flow.

weighted positives and downweight the unobserved majority. The latent factors then serve two roles: they power a personalised dot-product score and supply features to downstream ranking models. Tune the regularisation strength (a penalty that discourages the factors from growing too large and overfitting) and the number of latent factors so the model balances expressiveness with generalisation.

Content-based and hybrid approaches. Vectorise item metadata — textual descriptions, tags, categories, or numeric attributes — and compute similarity directly in the feature space. This channel keeps recommendations flowing when catalogue additions arrive faster than user feedback accrues. Blend content and collaborative signals with weighted sums, stacking models, or learning-to-rank frameworks so the system honours both behavioural and descriptive evidence.

Pairwise ranking losses and graph walks. Techniques such as Bayesian Personalised Ranking optimise the relative order of preferred versus unobserved items, offering better top-of-list control than pure squared-error objectives. Personalised PageRank on a user–item graph, meanwhile, reuses network ideas from chapter 4 by running random walks with restart to prioritise items connected to the user's neighbourhood. These methods broaden the candidate toolkit once baselines and

factor models are stable.

Section summary

- Ship popularity and co-occurrence baselines first to establish trust and provide fallback behaviour.
- Combine multiple candidate generators inside a two-stage architecture before introducing complex rankers.
- Use matrix factors, content vectors, and pairwise objectives to personalise beyond simple heuristics while respecting operational rules.

Self-check questions

1. Your catalogue team launches 500 new items overnight. Which blend of candidate generators keeps them discoverable by the next morning?
   Hint: think about content vectors plus a popularity safety net.

2. How would you explain the benefit of a two-stage recommender to a stakeholder worried about compute cost?
   Hint: emphasise that the first stage narrows the search space before heavier ranking happens.

3. You notice that association-rule recommendations overindex on bundled accessories. How could you combine rules with similarity scores or diversification to balance the slate?
   Hint: revisit how post-processing and diversity penalties reshape final lists.

23.3 Evaluate, monitor, and govern recommendation engines

Match offline evaluation to real-world exposure, then track bias, coverage, and compliance risks once the model deploys.

Learning objectives

After this section you will be able to:

- construct temporal holdouts that respect the causal order of implicit interactions;
- select ranking metrics that align with product funnels and catalogue health;
- articulate bias and governance considerations, including feedback loops and do-not-recommend policies.

Prerequisites

You should already be comfortable with:

- creating train/validation/test splits without leakage as covered in chapter 14;
- interpreting ROC-style metrics and coverage statistics from chapter 17;
- collaborating with stakeholders to define acceptable trade-offs as practised in chapter 2.

Implicit logs capture behaviour over time, so random splits leak the future into the past. Instead, choose a cutoff timestamp T, train on interactions up to T, validate on the next slice, and reserve the most recent window for last-mile checks.

This temporal discipline aligns the offline experiment with the deployment scenario: we always predict tomorrow using yesterday's knowledge. Track catalogue coverage alongside accuracy so we notice when only the headliners receive exposure.

Naïve random splits produce misleading validation scores for recommenders. Consider a user who has watched seasons one, two, three, five, and six of a series. If season four lands in the test set by chance, the model will trivially predict it belongs on the user's shortlist, inflating the metric without demonstrating genuine discovery or preference modelling. Moreover, consumption behaviour shifts: after completing several seasons, viewers often seek thematically related content rather than the next episode of the same show. A temporal split captures that evolving preference; a random split conflates initial interest with sustained engagement.

Choosing the right objective. Before selecting ranking metrics, clarify what the recommender should optimise. Streaming-subscription platforms and advertising-driven services pursue fundamentally different goals. A subscription service that charges a flat monthly fee benefits from member retention and satisfaction; whether a viewer watches ten minutes a day or five hours, the revenue stays constant. In contrast, platforms that monetise through advertisements earn more when users remain engaged longer, because every additional minute watched translates to more ad impressions. The latter optimisation can create filter bubbles that show only similar content, reducing serendipity, or drive rabbit-hole effects in which progressively more extreme recommendations maximise the probability of continued viewing.

Ethical recommender design therefore pairs the business

metric with safeguards: inject diversity, limit consecutive recommendations in the same narrow topic cluster, and rotate long-tail content to prevent over-concentration on popular items. Stakeholders should recognise that when a service reaches hundreds of millions or billions of users, the recommendation code shapes global media consumption patterns and carries responsibility for preventing harmful feedback loops.

Ranking metrics. Precision@k and Recall@k link naturally to top-N shelves; normalised discounted cumulative gain (NDCG) rewards correctly ordering relevant items. Beyond accuracy, monitor novelty via the mean negative log popularity and assess diversity by measuring the average dissimilarity between recommended items. When offline metrics improve, corroborate them with online experiments such as A/B tests or interleaving so the product team trusts the gains.

Bias and governance deserve equal space in the evaluation plan. Popularity bias can snowball: showing the same hits over and over suppresses the long tail. Feedback loops mean offline logs reflect past policies rather than counterfactual behaviour. Mitigation strategies include injecting exploration traffic, estimating counterfactual propensity weights, or rotating long-tail content into the shortlist. Keep a register of do-not-recommend rules covering safety, compliance, and stock availability, and ensure the post-processing stage enforces them. Finally, co-design explanation strings with stakeholders so end users understand why an item surfaced, which supports accountability conversations.

Section summary

- Temporal splits mirror deployment reality and prevent future information leaking into training.
- Combine precision/recall with coverage, novelty, and diversity metrics to balance accuracy with catalogue health.
- Monitor bias, feedback loops, and policy constraints as integral parts of the recommender lifecycle.

Self-check questions

1. How would you convince a teammate to adopt a temporal validation split when a random split reports higher accuracy?
   Hint: highlight the risk of leaking future interactions and overestimating performance.
2. Which monitoring dashboard widgets would reveal a recommender drifting toward popularity bias over time?
   Hint: think about long-tail coverage and novelty trends.
3. Your compliance lead asks for proof that banned items never surface. Which offline tests and live alerts would you set up to provide that assurance?
   Hint: combine post-processing unit tests with production incident logging.

23.4 Bridge to the Week 12 practical

Arrive at the lab with a baseline, a shortlist of datasets, and an evaluation script so you can focus on iterative improvement.

The Week 12 practical follows a two-track structure. Analytics-focused students pivot implicit logs into user–item matrices with pandas or spreadsheets, compute item–item similarities, and report Precision@10 alongside catalogue coverage. Programming-focused students implement sparse k-nearest neighbours, train an implicit alternating-least-squares model, and compare NDCG@10 across a temporal split. Both tracks share the same rhythm: establish a popularity baseline, validate a personalised model, and document trade-offs in a lab journal.

Prepare by selecting a dataset that fits your hardware budget. MovieLens 100K offers quick iteration; the RetailRocket e-commerce sample highlights purchase funnels; small open-education logs, such as OULAD course clicks, complement association-rule storytelling. Sketch a notebook or script template that loads the data, constructs the temporal split, computes baseline metrics, and leaves placeholders for candidate generators you want to test. Bring a checklist of business rules that could apply in your chosen context so you can rehearse how to encode them in post-processing.

Next steps. Before class, rehearse presenting a baseline dashboard to a stakeholder. Include the metric summary, a coverage snapshot, and a paragraph on policy compliance. That preparation makes the in-lab reflection smoother and mirrors the reporting expectations for the final assignment.