

PRODUCTION AI ENGINEERING

# DATA VOYAGE

---

Building Real AI Systems  
from Data to Deployment

ONE PROJECT · 24 CHAPTERS · RAW DATA TO DEPLOYED API

IRFAN ALI

# Data Voyage

---

Building Real AI Systems from Data to Deployment

IRFAN ALI

# Contents

Copyright

Dedication

A Note Before You Begin

Preface

Acknowledgements

Prologue: What We're Building

Part I – Foundations

Chapter 1: The Data Science Landscape

Chapter 2: The Working Engineer's Python

Chapter 3: The Statistics You Actually Need

Chapter 4: Data – Types, Sources, Ethics

Part II – Data to Insight

Chapter 5: Data Collection – Building the TalentLens Dataset

Chapter 6: Data Cleaning – Turning Raw Into Usable

Chapter 7: Exploratory Data Analysis

Chapter 8: Statistical Inference – Testing What You Noticed

Part III – Classical ML

Chapter 9: Supervised Learning – The TalentLens Role Classifier

Chapter 10: Feature Engineering and Selection

Chapter 11: Unsupervised Learning – Clustering the TalentLens Corpus

Chapter 12: Neural Networks – When Depth Pays for Itself

Part IV – Specialist AI

Chapter 13: NLP – Extracting Skills from Job Postings

Chapter 14: Time Series – Skill Trends (Methods First)

Chapter 15: Scaling Python – When You Outgrow Pandas

Part V – The GenAI Stack

Chapter 16: RAG and Vector Search – Building Intelligent Search

Chapter 17: LLM Generation – Making TalentLens Explain Itself

**Chapter 18: Agentic AI – Letting the Model Choose Its Tools**

**Part VI – Shipping**

**Chapter 19: FastAPI – Ship TalentLens as an API**

**Chapter 20: Docker + Render – Ship the API in a Container**

**Chapter 21: CI/CD – Automating Tests, Quality, and Deployment**

**Chapter 22: Packaging as a PyPI Library**

**Part VII – Career**

**Chapter 23: Real-World Case Studies**

**Chapter 24: The India Playbook – Getting Hired, Paid Well, and Building a Global Career from India**

**Glossary**

**References and further reading**

**About the Author**

# Copyright

---

## Data Voyage: Building Real AI Systems from Data to Deployment

by Irfan Ali

Published by DataCortex IQ.

Copyright © 2026 Irfan Ali. All rights reserved.

No part of the text of this book may be reproduced, stored, or transmitted in any form without written permission from the author, except for short quotations with attribution in reviews and articles.

The companion source code at [https://github.com/irfanalidv/Book-Data\\_Voyage](https://github.com/irfanalidv/Book-Data_Voyage) is released under the MIT License. The code listings printed in this book are covered by the same license, so you may copy them into your own projects.

Version 2.2.1, October 2026.

Cover photograph by Irfan Ali: a hill town in the Darjeeling hills at night, seen through the fog.

**About the data.** The TalentLens dataset used throughout the book is synthetic. Its employers are fictional, and any resemblance to real companies is coincidental. Where the book describes the real job market, including the salary bands and market segments in Chapter 24, those figures are the author’s professional judgement as of early 2026 and are labelled as such. They are not measurements, and they will change.

**Data sources.** The real-data measurements in Chapters 10 and 13 were made on job postings retrieved through The Adzuna API (Jobs by Adzuna, <https://www.adzuna.in>), used for research under Adzuna’s API terms. The book reports aggregate results only and reproduces no Adzuna listing text. All other figures come from the synthetic dataset or from the public sources credited where they appear.

**Disclaimer.** The career, salary, tax, and payment material in this book is general information, not financial, legal, or tax advice. Check current rules with a qualified professional before acting on them. The author has made every effort to ensure the accuracy of the information and code in this book, but it is provided without warranty, and the author accepts no liability for any loss or damage arising from its use.

Product and company names mentioned in this book, including those of software libraries and cloud services, may be trademarks of their respective owners. They are used for identification only.

For permissions and corrections, write to [irfan@datacortex.in](mailto:irfan@datacortex.in).

# Dedication

---

*For my mother, who believed in this work before there was any work to believe in.*

*And for every student in a small city who was told the real action was somewhere else. It isn't. It's wherever you open your laptop.*

# A Note Before You Begin

---

*Data Voyage: Building Real AI Systems from Data to Deployment*

I do not have an endorsement from a famous professor or a Silicon Valley executive. I never tried to get one. Those front-matter blurbs are usually written by people who skimmed the book and agreed as a favour. They do not tell you anything real.

What I can tell you is why I wrote this.

I have spent the last several years building AI systems: at companies, for clients, under contract, and on my own. Some of it went well. Some of it went badly in ways that were expensive and instructive. Through all of it, I kept running into the same problem: the books and courses that exist teach you concepts in isolation, on toy datasets, with none of the friction that makes real work hard.

I wanted a book that taught you to build things that work. Not a book that teaches you what gradient descent is. A book that teaches you how to know when you need gradient descent, how to implement it correctly on data you collected yourself, how to deploy the result somewhere real, and how to charge for it.

That book did not exist when I needed it. So I wrote it.

The project running through every chapter, TalentLens, a job market intelligence platform, is real in the sense that matters: the data comes from real sources, the problems it solves are problems real people have, and the code is the kind of code you would write at work. It hasn't been cleaned up for a textbook or simplified until it stops resembling the problem.

I wrote this from Siliguri, a city in West Bengal near the Nepal border. Not from Bangalore. Not from a startup hub. From a room, with a laptop, with the same internet connection and the same package documentation as everyone else. If you are reading this from somewhere people tell you is too far from the action, it is not. The work is the thing that matters. This book is about the work.

Build something real. That is the whole point.

---

*Irfan Ali Founder, DataCortex IQ Siliguri, 2026*

# Preface

---

*Data Voyage: Building Real AI Systems from Data to Deployment*

I didn't learn data science in a classroom.

I learned it at 2 AM in Siliguri, a small city pressed between the Himalayas and the Bengal plains, about as far from Silicon Valley as you can get, debugging a RAG pipeline with a client deadline in six hours and no one to ask. I learned it by shipping broken code, watching it fail in production, and figuring out why. I learned it by building things nobody asked me to build, publishing Python libraries nobody initially used, and taking contracts that stretched me into territory I had no business being in yet.

By the time I sat down to write this book, I had seven years of production AI engineering under my belt, two peer-reviewed papers, eleven libraries on PyPI, and a company I'd built from scratch. I'd fine-tuned LLMs at a Schneider Electric subsidiary, built the full AI intelligence layer for a Hong Kong startup, shipped a voice-first wellness app, and built inventory management software for a Nepal-based FMCG client. I had a Master's in Data Science and AI from IISER Tirupati. I'd negotiated equity-stake offers from US companies over a phone call from Siliguri, and I'd walked away from one of them when the contract terms didn't match what I'd signed up for.

None of that came from reading a data science textbook.

It came from building things. From the specific, sweaty experience of taking a concept from a paper or a tutorial, applying it to real data that was messy and incomplete and structured wrong, watching it fail, and fixing it. From learning that an algorithm is only as good as your ability to explain what it's doing to a product manager who doesn't care about gradients. From understanding that deployment is the chapter every book skips because it's hard to write about without actually doing it.

That's why I wrote this book the way I did.

## What this book actually is

This is not a survey of data science concepts. You don't need another one of those. There are thousands of them and they all teach the same Iris dataset.

This is a book about building things that work: in production, under pressure, with real data, in 2026.

Every concept in this book is taught through a single running project: **TalentLens**, a job market intelligence platform we build from scratch. By the time you finish, you'll have collected and cleaned job posting data, built a model that classifies roles from what a posting says, added semantic search and an LLM layer that explains how each job matches your CV, wrapped the tools in an agent, packaged the core logic as a library ready for PyPI, and shipped a FastAPI service in a Docker image with a CI/CD pipeline that deploys it to Render.

You'll also have learned every foundational concept along the way, but you'll have learned it in context, solving a real problem, with real data.

---

## Who this is for

You're learning data science or AI engineering, and you're frustrated. You've finished tutorials, you can run the sklearn examples, you might even have a Kaggle notebook or two. But you can't figure out how to go from "I understand this concept" to "I can build something with this that someone would pay for."

Or you already work in data (as an analyst, a software engineer, a researcher) and you need to move into ML/AI engineering. You need to be able to do the whole stack: data, model, deployment, and the conversation with the person who's paying for it.

Or you're in India, doing a CS or DS degree, and you want to know how to get a job. That means what FAANG interviews look like in theory, and also what AI startup interviews look like, what a remote contract with a US company looks like, how to structure your GitHub, and what salary to ask for.

This book is for all three of you.

**Primary audience:** intermediate engineers and students who can already run scikit-learn examples and want to ship real systems end to end, including CS and DS graduates in India targeting AI Engineer roles, remote contracts, and production-shaped portfolios.

**Secondary audiences:** working data analysts and software engineers upskilling into ML/AI engineering and production deployment; and senior engineers who want a single-project reference for GenAI stack choices (RAG, agents, FastAPI, CI/CD) without a survey of every framework.

---

## Prerequisites

You're ready for this book if you can:

- Run a Python script from the terminal and read its traceback without panic
- Install packages with `pip` into a virtual environment
- Write a function and a class in Python (tutorial-level fluency is enough)
- Train at least one scikit-learn model end-to-end (a course lab or Kaggle notebook counts)

You do **not** need deep math beyond comfort with means, medians, and the idea that a model can be wrong on data it has never seen. We derive what you need in context.

---

## How to read it

The book has seven parts, and each one moves the TalentLens project a step further.

**Pick your path:**

- **Brand new?** Read the Preface, Prologue, and Chapter 1, then continue sequentially through Part VII.
- **Comfortable with Python and basic ML?** Skim Parts I–II and start writing real code at Chapter 5 (data collection).

- **Already shipping ML and want the GenAI and production stack?** Jump to Part V (RAG, LLM generation, agents).
- **Want the career advice first?** Jump to Chapter 24. It is written to stand alone, though it lands harder once you have built TalentLens.

Every chapter follows the same core structure (a few reorder or merge sections where the material demands it; the case studies in Chapter 23 and the career playbook in Chapter 24 lead with their content, not with code):

1. **The problem:** a real scenario, not a textbook setup
2. **Why this method:** what alternatives exist and why we're choosing this one
3. **The algorithm:** what each parameter does, in plain English
4. **The code:** production-quality, not tutorial-quality
5. **Interpreting the output:** what the numbers mean
6. **Common mistakes:** things I got wrong so you don't have to
7. **Interview prep:** five questions with template answers
8. **What's next:** how this chapter connects to the next step in TalentLens
9. **TalentLens checkpoint:** what should now exist on your machine, and the concepts you should now own

The companion repository ([github.com/irfanalidv/Book-Data\\_Voyage](https://github.com/irfanalidv/Book-Data_Voyage)) has the full code, tests, data, and figures for every chapter. One command installs the environment, and every chapter runs offline on the bundled dataset. Every number the book measures on that dataset reproduces on your machine; the few that come from live LLM calls or from real Adzuna postings are labelled where they appear.

---

## What this book is not

This is not a survey of every algorithm in data science. It is not a math textbook, a Kaggle-competition guide, or a tour of every framework that had a launch week in 2026. We build **one** project, TalentLens, production-shaped and end to end. If you need exhaustive coverage of a single topic, use the documentation for that tool; if you need to see how the pieces connect under deadline pressure, stay on the voyage.

---

## Conventions used

- **Chapter shape:** the nine sections above, in that order, in most chapters.
  - **Signposts:** boxes marked **Reference** hold lookup tables you can skim now and return to later; **Skip ahead if...** tells experienced readers what they can safely skim; **Deeper** marks optional detail.
  - **Runnable code:** every chapter has a script you run from the repository root (`python book/chNN/...`); figures land under `book/chNN/reports/figures/`.
  - **The data:** the bundled corpus is synthetic: 576 cleaned postings from fictional employers, generated with a fixed seed and designed to behave like real postings. Chapter 5 explains how, and how to collect live data instead. Where a chapter makes a claim about the real market, it says so and says where the claim comes from.
  - **Currency:** ₹ for India-market salary and cost figures; \$ where the narrative is global or USD-denominated (remote contracts, cloud pricing).
  - **Back matter:** a glossary of every term the book defines, and references for every tool and data source it uses, are at the end of the book.
- 

## A note on honesty

I'm going to tell you when something is hard. I'm going to tell you when the “standard” approach has problems. I'm going to tell you when the benchmark number looks good but the model would fail in the real world. I'm not going to pretend data science is clean or that AI is magic or that a 94% accuracy score means your model is production-ready.

I wrote the book I needed when I was starting. Not the book that makes it look easy. The one that tells you the truth and then shows you how to handle it.

Let's build something real.

---

**Irfan Ali** Founder, DataCortex IQ Siliguri, 2026

[github.com/irfanalidv](https://github.com/irfanalidv) | [irfan@datacortex.in](mailto:irfan@datacortex.in)

# Acknowledgements

---

This book exists because of the open-source ecosystem that made the work in it possible.

## **Open-source tools this book depends on**

Every chapter uses libraries built and maintained by engineers who publish their work freely. The core ones are pandas, NumPy, scikit-learn, PyTorch, FastAPI, sentence-transformers, Pydantic, joblib, and uvicorn. The authors of these tools, among them Wes McKinney, the scikit-learn contributors, Sebastián Ramírez and Nils Reimers, built the infrastructure that the rest of us stand on.

The RAG chapter uses pgvector and the SQLite ecosystem. The CI/CD chapter runs on GitHub Actions. The deployment chapter uses Render. None of this costs anything to start. That accessibility is not accidental. It is the result of deliberate decisions by people who believed the tools should be free to use.

## **The communities that made this practical**

The Adzuna API (Jobs by Adzuna) and RemoteOK's public API, and the datasets that ship with scikit-learn, gave the book real data to work with. The Hugging Face model hub provided pretrained models that would have taken months to train from scratch. Stack Overflow answered the questions that every engineer hits at 1 AM when something breaks in a way no documentation anticipated. The India and Nepal AI/ML community, particularly the practitioners building real systems in Bangalore, Hyderabad, Pune, and Kathmandu, shaped what this book considers important.

## **The people who shaped this directly**

The clients of DataCortex IQ, particularly Sumit Pokhrel, whose real production requirements made the deployment chapters real rather than hypothetical. CA Sadiq Shariff, for handling the compliance and corporate work that lets DataCortex exist as a real business and not a side project. The IISER Tirupati faculty who taught the foundations that made the harder material approachable.

Krishna Prasad Chitrapura, whose introduction opened doors I couldn't have opened on my own, and whose long view of the AI field shaped how I think about which problems are worth working on. Vijay Mulani, for the kind of warm-network conversations that turned this book from "something I might write someday" into something I actually finished.

The engineers in Nepal and India who reached out after seeing the open-source libraries and asked questions that revealed what the field needs from a book like this. The Robotics Association of Nepal, AIAN, and the team at MOCIT, for showing me what AI infrastructure looks like outside the usual centres of attention.

My mother, whose belief in this work made everything else possible.

## **On sources and research**

The salary benchmarks in Chapter 24 are my read of the early-2026 Indian AI/ML market, informed by seven years of offers, contracts, and hiring conversations, cross-checked against public compensation data. TalentLens is the method readers can use to verify against live postings: the chapter script measures remote-vs-on-site medians on your corpus; the five-market and negotiation bands are reference material until you have enough disclosed salaries to compute your own. Those figures will shift. Treat them as a starting point for your own research, not as fixed facts.

The technical content reflects production patterns from real systems: Reflecta (a voice-first AI wellness platform), Godam (an FMCG inventory platform serving Nepal), RAGNav and ragfallback (open-source RAG libraries), StackSift (a B2B product intelligence API), and various client projects under DataCortex IQ. When something is described as working in production, it has been running in production.

The open-source libraries cited throughout (RAGNav, ragfallback, AgentEnsemble, AgentCare, scrapeflow-py, and others) are all available on PyPI under my GitHub account. Use them, fork them, send patches.

---

*Irfan Ali Siliguri, 2026*

# Prologue: What We're Building

---

Imagine this is Monday morning.

You're a data science student in your final year, or maybe a software engineer two years into a job you've outgrown. You open LinkedIn. The first three posts in your feed are AI Engineers at companies you've never heard of, all hiring, all paying ₹35–60 lakh, all asking for things you half-recognise: RAG, fine-tuning, vector databases, agentic workflows, FastAPI, Docker, observability.

You've done the courses. You can train a logistic regression model. You once got 94% accuracy on the Titanic dataset. None of it tells you how to get from where you are to where those job posts live.

This book is the bridge.

---

We're going to build one thing together, end to end. It's called **TalentLens**, a job market intelligence platform. It collects job postings from public APIs, cleans them, classifies them by role, extracts the skills they demand, ranks them by semantic similarity to your CV, explains each match with an LLM, and serves search as an API you can deploy with one command.

When you finish this book:

- You'll have a working product built on job-market data. Not Iris. Not Titanic. The book ships a realistic synthetic corpus so everything runs offline and reproduces exactly; switch on the live collectors and the same code analyses real postings.
- You'll have a FastAPI service in a production-shaped Docker image, with health checks, rate limiting, and a CI/CD pipeline that tests every push and deploys to Render.

- You'll have a Python package, `talentlens-core`, with a release workflow that publishes to PyPI the way real maintainers do.
- You'll have a way to read the Indian AI/ML job market: which claims you can measure, which are judgement, and how to tell a real salary premium from a statistical illusion.
- You'll know how to answer the questions the career chapter asks (What's a fair salary for a Senior ML Engineer in Bangalore? Do remote roles really pay more? Which skills are growing?), and which of them your data can answer today, and which need months of history first.

Every chapter is one step. By the end of Chapter 5 you have a data collector. By Chapter 9 you have a classifier. By Chapter 16 you have semantic search. By Chapter 19 you have an API. By Chapter 21 you have a deployment pipeline that ships your code to production every time you push to main.

---

This is not a survey course. We are not going to cover every algorithm. We are going to cover the ones you need to build TalentLens, and we are going to cover them properly, with the parameters explained in plain English, the alternatives compared, and the failure modes named explicitly.

The chapters that follow are written the way I wish someone had written this material when I was starting: assuming you're smart, telling you the truth, showing you what works in production, and pointing out the places where the textbook answer and the real answer diverge.

Let's start with Chapter 1, where we look at the field we're entering: what the five roles in modern data and AI do, what they get paid, and where TalentLens fits.

# Part I – Foundations

---

*The field we're entering, the working engineer's Python, the statistics you actually need, and which data sources TalentLens is allowed to use.*

# Chapter 1: The Data Science Landscape

---

**TalentLens milestone:** Understand the problem we’re solving: why keyword search fails at job matching, what a data-science-powered platform looks like instead, and where each of the next 23 chapters sits in building one.

## The problem we’re solving

You open LinkedIn on a Monday morning. The job search bar is right there. You type “AI engineer Bangalore” and get 1,247 results. You add “remote”: 312. You add “₹40L+”: 4 results, two of which are from the same company and one of which is for a Director role.

You search again with “ML engineer” instead of “AI engineer” and get 891 results. Different jobs. Many of them are the same companies, but the postings don’t overlap because the title was different.

You try “Machine Learning Engineer” with the full spelling: 600 more results. Different again. Different listings. Same companies in many cases.

This is the failure of keyword search. The roles are similar enough that they should rank together when a candidate searches. The titles are different enough that they don’t. And no human is going to type every variation: “ML Engineer,” “Machine Learning Engineer,” “Senior ML Engineer,” “ML/AI Engineer,” “AI/ML Engineer,” “Data Scientist (ML),” “MLOps Engineer,” “Applied ML Engineer.” The companies aren’t consistent and the candidates can’t be exhaustive.

There's a second failure underneath the first. Even when the search matches, the *content* is opaque. A posting that says "5+ years Python, exposure to ML, looking for full-stack engineer" might pay ₹35L. A posting that says "Senior ML Engineer, ML fundamentals, production experience" might pay ₹14L. The titles tell you nothing. The descriptions tell you something. Nobody reads 1,247 descriptions.

**This is what TalentLens fixes.** A job market intelligence platform that reads every posting, classifies the role based on what the job actually involves (not what it's labelled), extracts the skills demanded, ranks postings by similarity to your CV, and surfaces the salary range distribution for the role-and-skill profile you're targeting. By the end of this book we will have built it end to end.

That's the running thread. Every chapter advances it. Chapter 4 picks the data sources. Chapter 5 scrapes them. Chapter 9 builds the role classifier. Chapter 13 extracts the skills. Chapter 16 powers the semantic search. Chapter 19 serves it as an API. Chapter 22 publishes the reusable core as a PyPI package. Chapter 24 uses the data the platform produces to answer the career questions you opened the chapter with.

That's the book. Let's start by understanding the field we're entering.

---

## Why now, and why this chapter

You are starting a career, or pivoting one, into a field that has changed more in the last three years than it did in the previous ten. The job titles that pay best in 2026 didn't exist in 2022. The tools that anchor modern AI engineering (vector databases, LLM frameworks, agent infrastructure) barely existed five years ago. Most data science curricula are still teaching CRISP-DM and the difference between supervised and unsupervised learning, which is fine, but the gap between "completed a data science course" and "can get hired as an AI engineer" is now wider than it has ever been.

This chapter exists to close part of that gap by giving you a map.

**What this chapter is:** the taxonomy of the field as it actually exists in mid-2026: the five roles that employers hire for, what each one does day to day, what each one pays, and where the trajectory is heading. The 24-chapter arc that follows, mapped to which roles each chapter prepares you for.

**What this chapter is not:** a definition of data science. There are 200 books that open with that. The Wikipedia page is fine. Skip it.

**When this chapter falls short:** if you’ve been working as a senior ML or AI engineer for five years, the taxonomy here is too coarse; you already know the field. Skim it and start at Chapter 2. The audience this chapter is calibrated for is engineers and students who can read a Pandas tutorial but don’t yet have a clear picture of how the roles differ in pay, work, and trajectory.

---

## The methods

This chapter doesn’t have methods in the algorithmic sense. What it has is a taxonomy, a structured way of slicing the field that the rest of the book builds on. Treat each role definition the way you’d treat a method definition: read what it does, what it costs (in salary terms), when to “use” it (which is to say, when to target it as your career path).

*Salary bands below are indicative early-2026 India ranges from the author’s market read, not TalentLens-measured medians.*

### Role 1: Data Analyst

**What they do:** Take questions from a business stakeholder (“why did revenue drop last quarter?”), pull the relevant data from a warehouse or BI tool, produce a chart or dashboard that answers the question, present it. Most of the work is SQL and a BI tool (Tableau, Power BI, Metabase, Looker). Some Python or R for ad-hoc analysis. Rarely deploys anything.

**Primary tools:** SQL (always), Tableau or Power BI (common), Excel (still), Python or R (sometimes).

**Primary question they answer:** “What happened?” Past tense, descriptive.

**Salary band (India, 2026):** ₹5–18L for 0–5 years. ₹18–28L senior. Top of this band caps lower than the technical roles below.

**When this is the right path for you:** you like talking to humans, framing questions, communicating findings. You're patient. You enjoy stakeholder management. You'd rather influence a decision than build a system.

**Red flag, a mislabelled role:** A "Data Analyst" job at a service company that asks for "data science" skills usually means "we want you to do analyst work but the title makes us look more sophisticated to clients." Same job, lower pay than calling it what it is.

---

## Role 2: Data Scientist

**What they do:** Take a less-defined problem ("can we predict which customers will churn?"), explore the data, design and run experiments, build statistical models, communicate results. Strong in stats, hypothesis testing, A/B testing, experimental design. Builds models that may or may not be productionised. The boundary with ML Engineer is fuzzy and depends on the company.

**Primary tools:** Python (pandas, scikit-learn, statsmodels), Jupyter notebooks, occasionally R. SQL for data extraction. Some product analytics tools (Amplitude, Mixpanel).

**Primary question they answer:** "*Why did it happen?*" Causal, inferential.

**Salary band (India, 2026):** ₹8–25L for 0–5 years. ₹25–45L senior. The variance is high because the title means different things at different companies.

**When this is the right path for you:** you like maths and statistics for their own sake, not as a means to ML. You enjoy designing experiments. You're comfortable explaining a p-value to a product manager.

**Red flag, a mislabelled role:** A "Data Scientist" job that doesn't mention any statistics in the JD and asks for Spark, Airflow, and Kubernetes is actually a Data Engineer role with marketing.

---

### Role 3: Data Engineer

**What they do:** Build and maintain the infrastructure that gets data from where it lives (databases, APIs, event streams, third-party tools) to where analysts and scientists can use it. Pipelines, warehouses, schemas. Strong in distributed systems, SQL at scale, orchestration.

**Primary tools:** Spark, dbt, Airflow or Dagster, Kafka, Snowflake or BigQuery, SQL (heavy), Python.

**Primary question they answer:** *“How do we get the data?”* Infrastructure, plumbing.

**Salary band (India, 2026):** ₹10–22L for 0–5 years. ₹22–45L senior. Higher floor than analyst because the skills transfer to software engineering generally.

**When this is the right path for you:** you like systems. You’re comfortable with the dirty parts of data: schema drift, late-arriving records, broken upstream APIs. You’d rather build pipelines that run for a year than dashboards that change weekly.

**Red flag, a mislabelled role:** At smaller companies, “Data Engineer” sometimes means “the only person who knows SQL.” If the JD doesn’t mention any pipeline tooling (Airflow, dbt, Spark) and the company has fewer than 50 engineers, you’re likely being hired as a database administrator under a more fashionable title.

---

### Role 4: Machine Learning Engineer

**What they do:** Take ML models (from data scientists, from open-source releases, or from their own training work) and put them into production. Train, evaluate, deploy, monitor. Strong in the software engineering side of ML (packaging, serving, observability) and usually decent at the modelling side too.

**Primary tools:** Python (scikit-learn, PyTorch or TensorFlow, MLflow), Docker, Kubernetes, FastAPI or similar, cloud ML platforms (Vertex AI, SageMaker, Azure ML).

**Primary question they answer:** *“Can we predict it?”* And then: “Can we serve that prediction reliably at scale?”

**Salary band (India, 2026):** ₹12–28L for 0–5 years. ₹28–55L senior. Equity at product companies adds substantially.

**When this is the right path for you:** you're a software engineer at heart who likes ML. You want to ship things, not just train them. You're patient with infrastructure work.

**Red flag, a mislabelled role:** "ML Engineer" at a company with no production ML usually means "we want to do ML but haven't yet." You'll spend the first six months building infrastructure that should already exist.

---

## Role 5: AI Engineer

**What they do:** Build systems that use foundation models (LLMs, vision models, audio models) to solve product problems. RAG over documents. Agentic workflows. Voice assistants. Multi-step reasoning. Less training, more orchestration and prompting and integration. The newest of the five roles, and the one with the most rapid evolution.

**Primary tools:** Python, LLM clients (OpenAI, Anthropic, Groq), LangChain or DSPy or raw HTTP, vector databases (pgvector, Pinecone, Weaviate, Qdrant), FastAPI for serving, increasingly agent frameworks.

**Primary question they answer:** "*Can we generate it?*" Or build a system that decides and acts on its own.

**Salary band (India, 2026):** ₹15–35L for 0–5 years. ₹35–75L senior. The remote contract band is higher again: \$4–12k/month is common for experienced AI engineers working with US startups.

**When this is the right path for you:** you're comfortable with rapid change. You can read a research paper from last week and apply something from it next week. You're comfortable that the "right" architecture is going to change every six months for the foreseeable future.

**Red flag, a mislabelled role:** "AI Engineer" at a company that just discovered ChatGPT in 2025 often means "wrap GPT-4 in a Streamlit app." Real AI engineering involves retrieval, evaluation, cost management, latency, fallback behaviour, and increasingly agents. If the JD mentions only "prompt engineering" and the role pays under ₹20L, the company isn't doing AI engineering yet.

---

## The three eras of the field

The five roles above are the *snapshot*. The trajectory matters too. The field has gone through three roughly distinguishable eras in the last decade:

**Pre-2018: Classical ML era.** Logistic regression, random forests, gradient boosting. Feature engineering was the differentiator. Most production “ML” was scikit-learn on tabular data. Data Scientist was the dominant role.

**2018–2022: Deep learning era.** ResNet, BERT, the rise of PyTorch. Transformers everywhere. CV and NLP became distinct sub-fields, each producing dedicated specialists. ML Engineer emerged as a distinct role from Data Scientist as the gap between training and serving widened.

**2022–present: LLM era.** Foundation models become available via APIs. The barrier to building useful AI products drops from “PhD and a GPU cluster” to “API key and a credit card.” AI Engineer emerges as a role distinct from ML Engineer because the work (orchestration, prompting, retrieval, evaluation, agents) is different from training models. The pace of change accelerates.

We’re three years into the LLM era. The roles are still settling. The work TalentLens does, semantic search over job postings, LLM-based skill extraction and CV matching with retrieval-augmented generation, wouldn’t have been feasible in 2018 and wouldn’t have been a product idea worth pursuing in 2022. In 2026 it’s a weekend project for a competent AI engineer.

That’s the world this book is teaching you to operate in.

---

## The code

The first chapter doesn’t compute anything heavy. What it does is **make sure your environment is ready** for the next 23 chapters. The companion file does three things:

1. Checks your Python version (the book targets 3.11; 3.12 also works)
2. Checks which book dependencies are installed and which are missing
3. Generates one chart: the distribution of the five roles in a small sample dataset, so you can verify matplotlib and pandas are working

Run it from the repository root:

```
python book/ch01/ch01_data_science_landscape.py
```

If everything is set up, you'll see a “ready” banner and a saved chart at [book/ch01/reports/figures/ch01\\_role\\_distribution.png](#). If something is missing, the script tells you what to install. See [book/ch01/ch01\\_data\\_science\\_landscape.py](#) for the implementation.

Three code decisions worth explaining:

*Why we check the environment in Chapter 1 rather than assuming it's set up:* Half the failures in technical books are environment failures. The reader runs the chapter code, gets a `ModuleNotFoundError`, and assumes the book is broken. By making Chapter 1's job to verify the environment, we catch those failures early and tell the reader exactly what to do.

*Why we don't yet load real TalentLens data:* Real data comes from Chapter 5's collectors. Chapter 1 uses a small synthetic sample (five role names, plausible salary distributions, a few hundred rows) because the point of the chart is to verify matplotlib works, not to do real analysis. Real analysis starts in Chapter 7.

*Why the chart is the simplest possible bar chart:* Chapter 7 introduces the book's visualisation standards in detail. Chapter 1 just needs a chart that proves the toolchain is functioning. We will not start with anything fancy.

---

## Interpreting the output

When you run `python book/ch01/ch01_data_science_landscape.py`, the output looks like this:

```
=====
Data Voyage – Chapter 1 environment check
=====

Python version:      3.11.0  ok

matplotlib:          3.9.4   ok
```

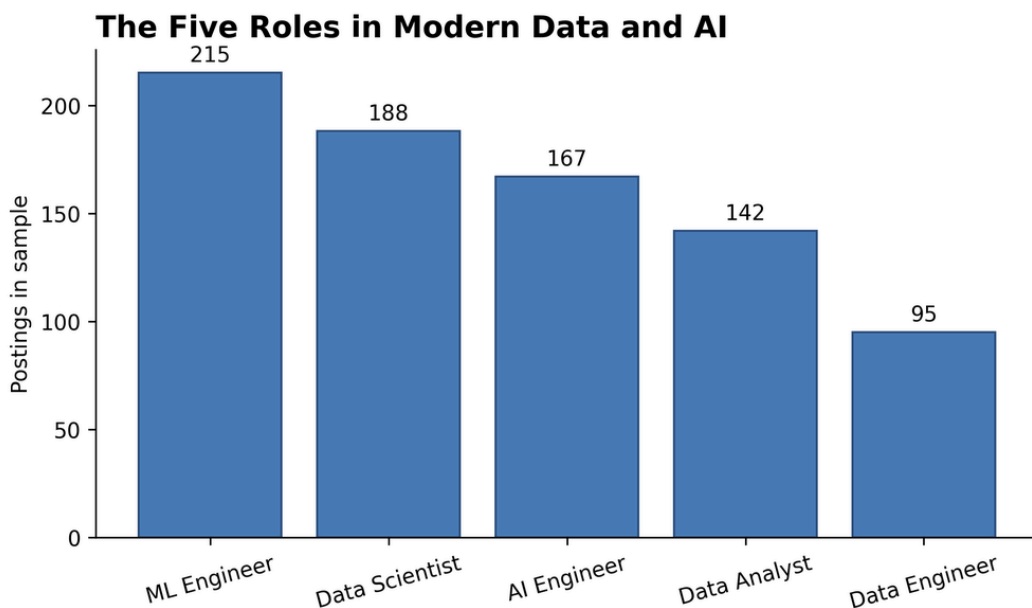
```
pandas:          2.3.3  ok
numpy:           2.1.3  ok
scikit-learn:    1.5.2  ok
sentence_transformers: 5.1.2  ok
fastapi:         0.136.1 ok
pydantic:        2.13.3  ok
openai:          1.55.3  ok
```

You are ready for Chapter 2.

Chart saved: book/ch01/reports/figures/ch01\_role\_distribution.png

**What “you are ready” means.** It means the core data stack works and every package the book pins is importable. If a line shows `not installed` instead of a tick, the script names the chapter that first needs it. `sentence_transformers`, for example, is not needed until Chapter 13.

**What the chart shows.** A simple bar chart of five role counts in the synthetic sample. It exists to confirm that matplotlib renders, that file output works (so figures get saved to the right place), and that pandas can compute a basic `value_counts()`. If the chart doesn’t appear, your visualisation toolchain is broken and Chapter 7 will fail; fix it now.



*Five-role counts in the synthetic sample: confirms matplotlib and pandas output paths work*

**What “not installed” doesn’t mean.** It doesn’t mean the package is broken; it means your environment is missing it. The lockfile install (`make install`) pulls in the whole book stack, about 1.5 GB including PyTorch, so every chapter runs without further installs. If you are on a slow connection, you can start with `pip install pandas numpy matplotlib scikit-learn` (about 200 MB), work through Chapters 1–12, and install the lockfile before Chapter 13.

---

## Common mistakes I’ve seen (and made)

**Mistake: Choosing a role based on the salary band without checking what the role does**

What happens: you see AI Engineer at ₹35L+ and decide that’s the target. You spend three months getting comfortable with LLMs and vector databases. You land an AI Engineer interview. The interviewer asks you to design a recommendation system us-

ing collaborative filtering. You realise the role is actually an ML Engineer role at a company that has rebranded its postings to attract LLM-era candidates. You weren't prepared.

How to catch it: read the job description, not the title. If the JD mentions training models, evaluation, hyperparameter tuning, and "ML system design," it's an ML Engineer role regardless of what they called it. If it mentions retrieval, prompt engineering, agents, and "LLM evaluation," it's an AI Engineer role. The title is marketing; the JD is the spec.

Fix: build your skills against the *role you want to do*, not the title that sounds impressive. The book is structured so that completing the full 24 chapters gives you AI Engineer-shaped skills. If you want to be an ML Engineer instead, that's also fine: you can stop after Chapter 22, and the resulting skill profile maps cleanly to ML Engineer roles.

---

### **Mistake: Treating the five roles as a strict hierarchy**

What happens: you assume Data Analyst → Data Scientist → ML Engineer → AI Engineer is a career progression where each role is "better" than the last. You jump roles every 18 months chasing the next title. You end up four years in with shallow experience in everything and deep experience in nothing.

How to catch it: look at the senior people you respect. The senior Data Analyst at a product company who has been there for six years is often paid more, has more influence, and does more interesting work than the AI Engineer who has been at three companies in the same period.

Fix: the five roles are *different*, not hierarchical. Salary bands overlap. Trajectories overlap. The right role for you depends on what you find interesting, not what's currently fashionable. Pick one and get deep before you switch.

---

### **Mistake: Believing the LLM era will replace all the others**

What happens: you skip the chapters on classical ML and statistics because “GPT-4 can do all that now.” You go to an interview for an AI Engineer role. The interviewer asks you to debug why a sentence-transformer is returning high similarity for opposite-meaning sentences. You don’t know what cosine similarity is doing. You don’t know what an embedding actually represents. You can’t reason about it. (We define both properly in Chapter 16; for now: an **embedding** is a numeric vector that encodes meaning, and **cosine similarity** measures how close two meanings are in that space.)

How to catch it: the interview is the catch. The job will be the catch. Foundations always matter.

Fix: every era builds on the previous one. Modern AI Engineers who can’t reason about a confusion matrix are limited. Modern Data Scientists who can’t use an LLM API are limited. The book covers all three eras for a reason; don’t skip the parts that feel “old.”

---

## Interview questions

### Q1: What’s the difference between a Data Scientist and an ML Engineer?

Template answer: “The distinction varies by company, but the cleanest way to think about it is along the production axis. A Data Scientist explores data, designs experiments, builds and evaluates models, and communicates findings. They might productionise their models, but usually someone else does. An ML Engineer takes models, their own or someone else’s, and puts them into production, including the serving infrastructure, monitoring, evaluation pipelines, and reliability work. At small companies the same person does both. At large companies the roles are distinct. If you ask in an interview whether the company expects this role to ship models to production with end-to-end ownership, the answer tells you which kind of role it actually is.”

### Q2: Why has ‘AI Engineer’ emerged as a role distinct from ‘ML Engineer’ in the last few years?

Template answer: “The work is different. ML Engineering is about training, evaluating, and serving models. AI Engineering, in the LLM era, is mostly about orchestrating foundation models: retrieval-augmented generation, prompt engineering, agent de-

sign, evaluation of generated content, cost and latency management, fallback handling. You can be a strong AI Engineer without having trained a model in two years; you can be a strong ML Engineer without having ever called an OpenAI API. The skill overlap exists but the day-to-day work and the depth of expertise required are different enough that companies now hire for them separately.”

**Q3: Walk me through how you’d choose between the five roles if you were starting today.**

Template answer: “I’d look at three things. First, what kind of work I find interesting. Talking to humans and framing questions favours analyst; designing experiments favours scientist; building infrastructure favours data engineer; shipping production systems favours ML or AI engineer. Second, what the market is paying for the trajectory I want. AI Engineer has the strongest current pay growth but also the most uncertainty about what the role will look like in three years. Third, where my existing strengths actually compound. Someone with five years of backend Python and a math degree compounds faster into ML Engineer than into Data Analyst, even if Data Analyst feels lower-risk on day one. Career choice should be a long compounding bet, not a salary maximisation in year one.”

**Q4: What’s the most overhyped role in the current market?**

Template answer: “Probably ‘Prompt Engineer’ as a standalone title. Companies hired for it in 2023; most of those roles have either been absorbed into broader AI Engineer roles or eliminated entirely. The skill of designing effective prompts and evaluating outputs is real and valuable, but as a full-time job it didn’t survive contact with production. The lesson is broader: chasing a single hot title is fragile. Building a portfolio of skills that map to a role’s *underlying work* is more durable than betting on which title happens to be in fashion.”

**Q5: What’s the most undervalued skill across all five roles?**

Template answer: “Communication. The data analyst who can frame findings for an executive, the data scientist who can explain why their model isn’t ready to ship, the ML engineer who can write a clear post-mortem after an incident, the AI engineer who can document a system another engineer will inherit: these are the people who get promoted regardless of role. Technical skill at the median level for the role is

enough to get hired; communication is what separates the people who plateau at senior from the people who become staff and beyond. Every interview process I've seen tests for this, even when the interviewers don't realise they're testing for it."

---

## What's next

In Chapter 2, we set up the working environment properly. Not Python syntax basics, but the project structure, dependency management, type hints, logging, and configuration patterns that every chapter from here on assumes. By the end of Chapter 2 you have a real project scaffold that the rest of the book builds on, including a `pyproject.toml`, a `Makefile` for common tasks, and the directory layout TalentLens grows into.

The road is long, but it's mapped. You now know the field. The next 23 chapters teach you to operate in it, building one real platform end to end. The first concrete code you write, outside of the environment check, happens in Chapter 5, when we start collecting real job postings. Everything between here and there is preparation.

Start Chapter 2.

---

## TalentLens checkpoint

At the end of this chapter, your environment should have:

- ❑ Python 3.11 or 3.12 installed and active
- ❑ A virtual environment created at `.venv` in the repository root
- ❑ Dependencies installed from the lockfile: `make install` (or `pip install -r requirements-lock.txt` then `pip install -e .`)
- ❑ `book/ch01/ch01_data_science_landscape.py` runs without errors
- ❑ One chart saved to `book/ch01/reports/figures/ch01_role_distribution.png`
- ❑ You can articulate, in your own words, which of the five roles you're targeting

**Concepts you own:**

- The five roles are different career paths, not a strict hierarchy; salary bands overlap and trajectories compound
- Title inflation vs job description reality: read the JD, not the marketing label
- Three eras of the field (classical ML → deep learning → GenAI orchestration) and which chapters map to which era

Reproduce locally from the repository root:

```
python -m venv .venv
source .venv/bin/activate          # Windows: .venv\Scripts\activate
make install                       # lockfile + editable talentlens + spaCy
    model
python book/ch01/ch01_data_science_landscape.py
```

If the environment check passes, you're ready for Chapter 2.

# Chapter 2: The Working Engineer's Python

---

**TalentLens milestone:** Set up project structure, virtual environment, `pyproject.toml`, and the config + logging patterns every later chapter imports from `talentlens/`.

## The problem we're solving

You finished Chapter 1 ten minutes ago. The terminal still shows the last line from the environment check:

```
You are ready for Chapter 2.
```

Python works. Matplotlib drew the five-role bar chart. You have a `.venv` activated. And you are about to open a new file called `scrape_jobs.py` in the repository root because that feels like the fastest way to start scraping for TalentLens.

Your teammate Priya clones the repo on Tuesday. She runs your script. It dies immediately:

```
FileNotFoundError: /Users/you/Book-Data_Voyage/data/jobs_raw.csv
```

Your path was hardcoded to your laptop. Hers is `/home/priya/projects/data-voyage`. The script never had a project. It had a folder of files that happened to run on one machine.

**This chapter fixes that.** We turn the book repository into an installable project: `pyproject.toml` at the root, a `talentlens/` package for shared code, pydantic-backed configuration, structured logging, a `Makefile` as the command interface, and pre-commit hooks so the next twenty chapters do not accumulate silent style drift. When you finish, `pip install -e .` works, `make test-ch02` passes, and `python book/ch02/ch02_python_setup.py` prints resolved paths instead of a traceback.

---

## Why this practice, and why now

Chapter 1 answered *what field am I entering?* Chapter 2 answers *how do working engineers structure the code that field runs on?*

We teach this immediately after the environment check because every chapter from here assumes the same imports:

```
from talentlens import get_settings, REPO_ROOT, DATA_DIR
```

If we waited until Chapter 5 (data collection), you would have already written three chapter scripts with duplicated path logic and three different conventions for reading API keys.

### What we are not doing:

- **setup.py as the packaging entry point:** deprecated in favour of `pyproject.toml` (PEP 621). New projects should not add `setup.py` unless they are maintaining legacy packaging.
- **Raw `os.environ.get()` for configuration:** no validation, no types, secrets scattered through call sites. We use pydantic-settings (see below).
- **`print()` for diagnostics:** fine for a one-off script; useless when you need log levels, filtering, or production log aggregation.
- **Teaching Python syntax:** you already know loops and comprehensions. This chapter is practice, not language basics.

**What Chapter 22 will do with today's work:** the `talentlens/` package you create here is the *development* copy inside the monorepo. In Chapter 22 we extract `talentlens_core` to PyPI as `talentlens-core`, the installable library consumers `pip install` without cloning the book. Same modules, different packaging boundary. Building the dev package correctly now means the extraction is a move, not a rewrite.

## The methods

**Skip ahead if...** you already run pyproject-based projects with a Makefile and pydantic-settings. Skim to the TalentLens-specific decisions ( `talentlens/paths.py` , `get_settings()` , and the checkpoint at the end).

### pyproject.toml : the project contract

`pyproject.toml` is the single file that tells tooling who you are and how to build you.

#### Reference: pyproject.toml sections

| Section                                      | Purpose                                                                                                                    |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| <code>[build-system]</code>                  | Which backend builds the package ( <code>setuptools</code> here).                                                          |
| <code>[project]</code>                       | Name, version, authors, <code>requires-python</code> , runtime <code>dependencies</code> .                                 |
| <code>[project.optional-dependencies]</code> | Feature groups ( <code>dev</code> , <code>ml</code> , <code>llm</code> ) so readers install heavy stacks only when needed. |
| <code>[tool.ruff]</code>                     | Linters + formatters config: one tool instead of <code>flake8</code> + <code>isort</code> + <code>autopep8</code> .        |
| <code>[tool.mypy]</code>                     | Static types; strict on <code>talentlens/</code> , relaxed on <code>book/ch*/</code> teaching scripts.                     |
| <code>[tool.pytest.ini_options]</code>       | Where tests live ( <code>tests/</code> at repo root for chapter integration tests).                                        |

Why not only `requirements.txt`? `requirements.txt` pins versions for reproducible installs. `pyproject.toml` makes the tree *installable* with `pip install -e .` so `import talentlens` resolves from anywhere in the repo. You need both: pins in requirements for the full book stack, metadata in pyproject for the package boundary.

## Virtual environments: pick one default

### Reference: Virtual environment tools

| Tool                              | When it wins                            | Why we do not default to it here                                      |
|-----------------------------------|-----------------------------------------|-----------------------------------------------------------------------|
| <code>python -m venv .venv</code> | Standard library, no extra install      | <b>Our default</b> ; matches CI and most teams                        |
| Poetry                            | Lockfiles + publish workflow in one CLI | Extra abstraction; book already pins in <code>requirements.txt</code> |
| pipenv                            | Pipfile lock                            | Less common in ML repos in 2026                                       |
| Conda                             | Scientific stack with binary deps       | Heavy; use when CUDA/system libs require it                           |

**Recommendation:** `python -m venv .venv`, activate, `make install`. Chapter 1 already walked you through this.

## Type hints and ruff

Type hints are documentation the interpreter can check. We type every public function in `talentlens/` and chapter entrypoints. Ruff enforces import order, unused imports, and many bug classes flake8 used to catch, without a second formatter pass for imports.

Run quality checks the way CI will:

```
make lint          # same checks as CI (ruff E,F,W,I + black on book/, tests/, talentlens/)
make lint-all     # full pyproject ruff rules - stricter, optional locally
ruff format book/ch02/ talentlens/ # direct formatter when you want ruff only
```

The repo's `make format` target invokes `black`, because CI checks formatting with `black`. `ruff format` produces near-identical output; use whichever your team standardises on, and make CI check the same one.

## Configuration: `pydantic-settings` vs `os.environ`

The brittle pattern (do not ship this):

```
import os

LOG_LEVEL = os.environ.get("LOG_LEVEL", "INFO")
DATA_DIR = os.environ.get("DATA_DIR", "./data")
OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY") # None or str – no
                                                    validation

if LOG_LEVEL not in ("DEBUG", "INFO", "WARNING", "ERROR"):
    raise ValueError("bad log level") # you write this by hand, every project
```

Problems: defaults are strings until you coerce them; paths stay strings; invalid `LOG_LEVEL` fails at runtime in random places; optional secrets are indistinguishable from empty strings; nothing documents which variables exist.

The pattern we use (pydantic v2 + `pydantic-settings`):

```
from functools import lru_cache
from pathlib import Path

from pydantic import SecretStr
from pydantic_settings import BaseSettings, SettingsConfigDict

from talentlens.paths import REPO_ROOT

class Settings(BaseSettings):
    model_config = SettingsConfigDict(
        env_file=".env",
        env_file_encoding="utf-8",
        extra="ignore",
    )

    log_level: str = "INFO"
    data_dir: Path = REPO_ROOT / "data"
    openai_api_key: SecretStr | None = None
    groq_api_key: SecretStr | None = None
```

```
@lru_cache
def get_settings() -> Settings:
    return Settings()
```

Wins: `data_dir` is a `Path`; missing `.env` is fine; wrong types fail at startup with a clear pydantic error; `SecretStr` keeps keys out of accidental logs; `get_settings()` is cached so you do not re-parse the environment on every import.

**Import rule for 2026:** use `from pydantic_settings import BaseSettings`, not `from pydantic import BaseSettings` (removed in Pydantic v2). The book pins `pydantic==2.13.3`.

## Logging: `logging.getLogger(__name__)` vs `print()`

```
import logging

logger = logging.getLogger(__name__)

logger.debug("Only when LOG_LEVEL=DEBUG")
logger.info("Normal diagnostic – filterable in production")
```

`print()` always goes to stdout with no level. Logging lets you turn verbosity up for debugging and down for demos without editing code. Chapter 2’s script uses logging for diagnostics and a short `print()` banner for the final “ready” message, the same pattern as Chapter 1’s environment check.

## Makefile as the command interface

Every repeated command becomes `make <verb>`:

### Reference: Makefile targets

| Target                        | What it does                                                                                                  |
|-------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>make install</code>     | <code>pip install -r requirements-lock.txt</code> , then <code>pip install -e .</code> , then the spaCy model |
| <code>make install-dev</code> | Runtime install plus lint and test tools (ruff, black, pytest)                                                |

| Target                      | What it does                                                          |
|-----------------------------|-----------------------------------------------------------------------|
| <code>make test-ch01</code> | Chapter 1 tests (environment check)                                   |
| <code>make test-ch02</code> | Chapter 2 tests (config + paths)                                      |
| <code>make test</code>      | Full suite under <code>tests/</code>                                  |
| <code>make lint</code>      | ruff check (the same rules CI enforces) plus a black formatting check |
| <code>make clean</code>     | Remove generated artefacts                                            |

When Priya joins, she does not need your shell history. She runs `make help`.

### pre-commit hooks

`.pre-commit-config.yaml` runs ruff and file hygiene checks before each commit. Hooks catch trailing whitespace and accidental large files; they do not replace `make test`. Install once:

```
pip install pre-commit
pre-commit install
```

## The code

Runnable entrypoint:

```
python book/ch02/ch02_python_setup.py
```

**Design decision 1: import from `talentlens`, not from a relative path.** The script starts with `from talentlens import get_settings, REPO_ROOT`. That only works after `pip install -e .`, which is intentional: if you skip the install step, you find out before Chapter 5's collector fails mysteriously.

**Design decision 2: `get_settings()` is cached.** `functools.lru_cache` on the factory means environment parsing happens once per process. Chapter 19's FastAPI app will call the same function inside dependencies.

**Design decision 3: secrets masked in output.** `Settings.mask_secrets()` redacts API keys before printing. Never log raw `SecretStr` values.

---

## Interpreting the output

After a successful run you should see something like:

```
INFO __main__: Info line – this is how chapters log diagnostics
=====
Data Voyage – Chapter 2 project scaffold
=====
talentlens version: 0.1.0
REPO_ROOT:          /path/to/Book-Data_Voyage
DATA_DIR:           /path/to/Book-Data_Voyage/data
BOOK_DIR:           /path/to/Book-Data_Voyage/book
Settings (secrets masked):
  log_level: INFO
  data_dir: /path/to/Book-Data_Voyage/data
  openai_api_key: None
  groq_api_key: None

Project scaffold is ready. Continue with Chapter 3.
```

If you have filled in `.env`, the two key lines read `***set***` instead of `None`; the value itself never prints.

**REPO\_ROOT**: directory containing `pyproject.toml`. Every path builder in later chapters should anchor here, not to `__file__` three levels deep in a chapter folder.

**DATA\_DIR**: where TalentLens datasets land (`data/clean/jobs_clean.csv` after Chapters 5–6). If this points somewhere unexpected, check `DATA_DIR` in `.env`.

**Verify the install:**

```
python -c "import talentlens; print(talentlens.__file__)"
```

You should see a path inside this repository (`.../Book-Data_Voyage/talentlens/__init__.py`). That is what an editable install means: Python imports the package from your working tree, so edits take effect without reinstalling.

**make help** : lists every target with a one-line description. If `test-ch02` is missing, the Makefile patch from this chapter did not land.

---

## Common mistakes I've seen (and made)

### Mistake: Hardcoded paths that work on the author's laptop

What happens: `pd.read_csv("/Users/irfan/Book-Data_Voyage/data/jobs.csv")` runs in your notebook. A reader clones on Linux, paths break, they blame pandas.

How to catch it: grep for `/Users/` or `C:\\` in the repo. Replace with `DATA_DIR / "jobs.csv"`.

---

### Mistake: `requirements.txt` without `pyproject.toml`

What happens: teammates `pip install -r requirements.txt` and can run scripts, but `import talentlens` fails because nothing declares a package. Six months later nobody can publish or test the tree as a unit.

How to catch it: fresh venv, only `pip install -r requirements.txt`, try `import talentlens`. If it fails, add `pip install -e .` to your onboarding docs (we add it to `make install` here).

---

### Mistake: `print()` instead of `logging.getLogger(__name__)`

What happens: debugging an ingestion pipeline at 2 AM, you add twenty `print()` calls. In production there is no way to silence them or route errors to your log aggregator.

How to catch it: code review for `print()` in library modules under `talentlens/`. Prints in `if __name__ == "__main__"` banners are fine.

---

### Mistake: Secrets committed because `.env` was not in `.gitignore`

What happens: `GROQ_API_KEY=gsk_...` lands in git history. You rotate the key and still get security scanner emails forever.

How to catch it: only `.env.example` belongs in git; real `.env` stays local. pre-commit's large-file hook catches some accidents, not all.

---

### **Mistake: Everyone runs different commands**

What happens: you run `pytest tests/`, Priya runs `python -m pytest book/ch02/tests`, CI runs `make test-ch09` only. Failures reproduce on one machine.

How to catch it: if a task appears twice in README, Makefile, and a GitHub workflow, pick one canonical entry (Makefile) and link to it.

---

## **Interview questions**

### **Q1: What belongs in a good `pyproject.toml` for a data/ML project?**

Template answer: “At minimum: `[build-system]`, `[project]` with name, version, `requires-python`, and dependencies, plus tool sections for the linter and test runner. Optional dependency groups keep heavy ML stacks out of the default install. The file should make `pip install -e .` work so internal packages import cleanly. Version pins for reproducibility often stay in `requirements.txt` or a lockfile; pyproject declares what the package is, not every transitive pin.”

### **Q2: Why pydantic Settings instead of `os.environ.get()`?**

Template answer: “Environment variables are untyped strings. Settings classes coerce types, validate values at startup, document allowed fields, and support `.env` files through `SettingsConfigDict`. Secrets can use `SecretStr` so they do not leak into logs. A cached `get_settings()` factory gives one validated instance per process, the same pattern FastAPI dependency injection uses later.”

### **Q3: How do you structure a Python project for production?**

Template answer: “Separate installable library code from entrypoints and notebooks. Root `pyproject.toml`, `src` or top-level package (`talentlens/`), `tests/` mirroring modules, config via settings not globals, logging module per package, Makefile or task runner for repeatable commands, pre-commit for style, CI running the same targets. Data lives outside the package under `data/` with paths resolved from repo root.”

#### **Q4: What does ruff replace and why use it?**

Template answer: “Ruff replaces flake8, isort, pyupgrade, and much of the autopep8/black workflow for linting and formatting in one fast binary. Fewer tools means fewer version conflicts in CI. We still pin versions in pre-commit so local and CI match.”

#### **Q5: How do you handle config across dev, staging, and prod?**

Template answer: “Same settings class, different `.env` files or environment variables injected by the host. Never branch on `if os.getenv('ENV') == 'prod'` scattered through code. One Settings object, different values. Secrets from the environment or secret manager, never committed. Defaults safe for local dev (e.g. `LOG_LEVEL=INFO`, data dir under repo root).”

---

## **What’s next**

Chapter 3 applies statistics to TalentLens salary data: means versus medians on right-skewed pay, confidence intervals, and the questions EDA in Chapter 7 will visualise. You already have paths and config; Chapter 3 adds the analytical vocabulary.

Chapter 4 maps data sources (Adzuna, RemoteOK, archives) against what TalentLens can legally and practically ingest. Chapter 5 runs the collectors into `DATA_DIR`.

---

## TalentLens checkpoint

You should have:

- `pyproject.toml` with project metadata and tool config
- `talentlens/` with `config.py`, `paths.py`, `get_settings()`
- `pip install -e .` succeeding (`make install`)
- `python book/ch02/ch02_python_setup.py` exiting 0
- `make test-ch02` passing
- `.env` copied from `.env.example` (local only)
- `pre-commit install` run once

### Concepts you own:

- `REPO_ROOT`-anchored paths: no hardcoded laptop directories in shared code
- `pyproject.toml` declares the installable package; `requirements.txt` pins the full book stack
- pydantic Settings vs raw `os.environ`: validate once at startup, not at every call site

```
make install
python book/ch02/ch02_python_setup.py
make test-ch02
```

# End of the free sample

---

The complete book has 24 chapters in seven parts: data collection and cleaning, classical machine learning, NLP and time series, vector search, LLM generation, agents, FastAPI, Docker, CI/CD, packaging, four case studies from shipped products, and a career playbook for the Indian AI and ML market.

**Get the full PDF and EPUB at [leanpub.com/datavoyage](https://leanpub.com/datavoyage)**

Companion code: [github.com/irfanalidv/Book-Data Voyage](https://github.com/irfanalidv/Book-Data-Voyage)