Agentic AI with Python

Book I

Easy Introduction with Twelve Easy Projects

(Agentic AI With Python Series)

By

Santanu Karmakar

Our intentions and actions

AFFECT

Self

Family

Community

Nation

Humanity

We should make responsible choices

About This Book

This book, ***Agentic AI with Python***, is written for readers who want to understand **how intelligent systems actually work in real life**, not just how to write code.
It is especially meant for teachers, students, professionals, and curious learners who may not come from a deep AI background but want to **build useful, practical systems**.

Today, many people talk about AI as if it is magic. In reality, most systems are still **rule-based workflows**—they follow instructions but do not think, decide, or adapt. This book gently explains **where automation ends and where agentic intelligence begins**, using Python as the main tool.

The goal of this book is **not to turn you into a machine-learning researcher**. The goal is to help you think like a **system designer**—someone who understands:

- how tasks flow,
- where decisions are needed,
- how memory and goals matter,
- and how multiple intelligent components can work together.

Why Python?

Python is simple, readable, and flexible.
You can think of Python as a **manager language**—it connects logic, data, APIs, and AI models smoothly. That makes it ideal for building agentic systems where coordination is more important than complex mathematics.

This book uses Python not as a coding challenge, but as a **thinking tool**.

How This Book Is Structured

The book is divided into **two clear parts**.

Part I focuses on concepts.
Here, you will learn what agentic AI really means, how it differs from

automation, and how real-world systems behave. The explanations use everyday examples—like offices, hospitals, schools, and service systems—so the ideas feel familiar and grounded.

Part II is fully practical.
It contains hands-on projects arranged in four stages:

1. Workflows without agents
2. Building simple agents
3. Workflows that include agents
4. Workflows with multiple agents working together

This progression is intentional. You will first experience the **limitations of logic-only systems**, then gradually see how agents bring flexibility, intelligence, and adaptability.

What This Book Is Not

- It is not a deep mathematical AI textbook
- It does not focus on model training or theory-heavy algorithms
- It does not assume advanced programming knowledge

Instead, it focuses on **how intelligent behavior emerges from well-designed systems**.

Who Should Read This Book

This book is suitable for:

- Educators and trainers
- Software developers and system architects
- Healthcare, education, and public-sector professionals
- Anyone curious about practical AI beyond buzzwords

If you can understand basic Python and are willing to think step by step, this book is for you.

A Practical Philosophy

Throughout this book, AI is treated as a **support system**, not a replacement for humans.
The emphasis is on responsibility, clarity, and usefulness—especially in sensitive domains like healthcare, education, and governance.

Agentic AI works best when humans and intelligent systems **collaborate**, each doing what they are best at.

This book invites you to explore that collaboration—calmly, practically, and thoughtfully.

Prerequisite Skills Expected from the Reader

This book is written to be **approachable and stress-free**.
You do **not** need to be an AI expert, a data scientist, or a mathematician to benefit from it. However, having a few basic skills will help you move through the book comfortably.

1. Basic Comfort with Computers

You should be able to:

- use a computer confidently,
- install software,
- run simple programs,
- and manage files and folders.

Think of this as being comfortable using a smartphone beyond just making calls —nothing advanced, just familiarity.

2. Basic Knowledge of Python

You are expected to understand:

- variables and data types,
- simple conditions (`if-else`),
- loops,
- functions,
- and how to run a Python script.

You do **not** need advanced Python features.
If you can read and slightly modify a simple Python program, that is enough.

3. Logical Thinking (Not Mathematics)

Agentic AI relies more on **logical flow** than on equations.

You should be comfortable thinking in terms of:

- steps,
- decisions,
- conditions,
- and outcomes.

For example:
“If this happens, what should the system do next?”

4. Willingness to Think in Systems

This book often asks you to see problems as **processes**, not isolated tasks.

You should be open to:

- breaking a problem into steps,
- identifying where decisions are needed,
- and understanding how parts of a system interact.

This is similar to how one thinks while planning a journey—routes, stops, alternatives—not just the destination.

5. Curiosity and Patience

Agentic systems evolve gradually.

You should be willing to:

- experiment,
- observe behavior,
- and refine designs step by step.

There is no rush. The book is designed to be read slowly and applied thoughtfully.

What Is *Not* Required

You do **not** need:

- machine learning theory,
- deep statistics,
- complex mathematics,
- or prior experience with AI frameworks.

All advanced ideas are introduced **only when needed** and explained in simple terms.

If you have **basic Python knowledge and an interest in understanding how intelligent systems behave**, you are well prepared to read this book.

Access to the Runnable Code Repository

All **runnable code examples** used in this book are maintained in a **public GitHub repository**.
This approach keeps the book clean and readable while ensuring that readers always have access to **working, updated code**.

Instead of printing long code listings inside the book, the focus here is on **understanding the ideas, system flow, and design thinking**. The repository contains complete, tested examples that you can directly run, explore, and modify.

What You Will Find in the Repository

The GitHub repository includes:

- fully runnable Python projects for each chapter,
- clearly named folders matching the book structure,
- step-by-step setup instructions,
- sample inputs and expected outputs,
- and simple comments explaining the logic.

Each project corresponds to a specific concept or workflow discussed in the book, making it easy to move between reading and practice.

Why the Code Is Kept Outside the Book

Keeping code in a separate repository has several advantages:

- code can be updated and improved over time,
- bug fixes can be shared instantly,
- readers can experiment freely without typing long listings,
- and the book remains focused on **conceptual clarity** rather than syntax.

You can think of the book as the **map**, and the GitHub repository as the **toolbox**.

How to Use the Code Along with the Book

A recommended approach is:

1. Read the concept in the book
2. Open the corresponding folder in the repository
3. Run the code as-is
4. Modify small parts and observe the change in behavior

This back-and-forth helps build real understanding.

Repository Link

The complete runnable code repository is available here:

GitHub Repository:

👉 *[Link will be provided here]*

(You may bookmark this page, as it will be referenced throughout the book.)

This repository is meant to support learning, experimentation, and responsible reuse.
Readers are encouraged to explore, adapt, and extend the examples for educational and personal projects.

PART I

Understanding Agentic AI

(Concepts, Not Code-Heavy)

PART I — Understanding Agentic AI (Concepts, Not Code-Heavy)

1. What Is Agentic AI?

- From simple programs to decision-making systems
- Real-life example: Assistant vs clerk vs manager
- Why “agent” is more than just automation

2. Python as the Backbone of Agentic Systems

- Why Python fits agentic thinking
- Python as a “coordinator” language
- Simple view: Python glues logic, data, and AI together

3. Workflow Thinking Before AI

- What is a workflow in plain terms?
- Steps, rules, and outcomes
- Real-life example: Hospital OPD process

4. Automation vs Intelligence

- Rule-based systems vs thinking systems
- Where automation works well
- Where it breaks down

5. What Makes a System an Agent?

- Goals
- Memory
- Decision ability

- Action
- Simple analogy: Human assistant at work

6. Agent Lifecycle

- Observe → Think → Decide → Act → Learn
- Why feedback is important
- How agents improve over time

7. Tools That Enable Agentic AI in Python

- LLMs (in very simple terms)
- APIs
- Databases
- Memory stores
  (No deep coding yet — only roles)

8. Single Agent vs Multi-Agent Systems

- One smart worker vs a team
- Coordination and hand-offs
- Real-life example: Hospital staff roles

9. Human-in-the-Loop Systems

- Why humans are still needed
- Safety, trust, and responsibility
- Agent as support, not replacement

10. Limitations, Ethics, and Practical Reality

- Where agentic AI fails
- Over-automation risks

- Responsible usage in healthcare, education, and governance

Chapter 1: What Is Agentic AI?

From Simple Programs to Decision-Making Systems

Think about how software has evolved over the past few decades. Early programs were straightforward: **input** → **process** → **output**. You gave them data, they followed a fixed set of instructions, and returned a result.

A calculator is a perfect example. You enter "5 + 3," it processes that addition, and displays "8." Every time. Predictably. The program has no ability to adapt, question, or change its approach.

Then came systems with conditional logic: **if-then statements**. Now programs could make basic choices: "If temperature < 32, warn user." Still rigid, but slightly more flexible. These systems could handle variations, but only variations their creators explicitly anticipated and programmed.

For decades, this was the ceiling of what software could do.

The Leap: From Automation to Agency

Agentic AI changes this fundamental equation.

Instead of following a predetermined sequence, agentic AI systems observe their environment, reason about what they encounter, make decisions autonomously, and take actions to reach a goal—often without explicit instruction for every step.

Here's the critical distinction:

Aspect	Traditional Automation	Agentic AI
Instructions	Every action pre-programmed	Goals defined; actions determined dynamically
Adaptability	Fixed logic for fixed scenarios	Learns and adjusts to new situations
Decision-Making	Rule-based (if X, then Y)	Reasoning-based (analyze, decide, act)
Environment Interaction	Passive; reacts only to expected inputs	Active; seeks information, interprets context
Problem-Solving	Applies one solution per problem type	Evaluates multiple approaches and chooses the best

Real-Life Example: Assistant vs Clerk vs Manager

Let me illustrate with a concrete scenario: **scheduling a team meeting with 12 people across three time zones.**

The Assistant (Traditional Automation)

You give an automated scheduling tool specific instructions:

"Send calendar invites for Tuesday at 2 PM ET to these 12 people."

The system executes immediately. But then:

- Three attendees are in Asia (wrong time zone—meeting is at midnight)
- Five people have conflicts (tool didn't check calendars)
- No one actually attends because the time was never viable

What happened? The assistant blindly followed your explicit instruction without understanding context, constraints, or outcomes.

The Clerk (Traditional Software with Rules)

Now imagine a slightly smarter system with pre-programmed conditional logic:

"If scheduling a meeting, check all attendees' calendars for conflicts. If conflicts exist, suggest alternative times. If time zone difference > 6 hours, flag as concern."

Better, right? The clerk now:

- Checks calendars automatically ✓
- Identifies 5 conflicts and suggests 3 alternatives ✓
- Flags the 6-hour time zone issue ✓

But the clerk still struggles:

- Suggests times that are "technically free" but still inconvenient (2 AM for Asia team)
- Doesn't understand priorities (CEO's time more valuable than intern's)
- Can't recognize that this meeting might not need all 12 people
- Doesn't proactively communicate why certain times won't work

The clerk executed the rules, but didn't reason about the problem.

The Manager (Agentic AI)

Now consider a truly agentic approach:

Goal: "Schedule a productive meeting with the core team."

The agentic AI system:

1. **Observes**: Retrieves calendar data, time zones, meeting history, and participant roles
2. **Reasons**: "If I invite all 12 people at a single time, 7 will be at inconvenient hours. Historical data shows 4 of these people rarely speak. Two attendees are in the same region and could sync separately."
3. **Decides**: "I should split this into two focused sessions: one for the core decision-makers (6 people, 9 AM ET), and a separate async briefing for others."
4. **Acts**:
- Proposes the two-meeting approach to the organizer
- Drafts calendar invites for both sessions
- Pre-writes briefing materials for the async group
5. **Adapts**: If the organizer rejects the plan, the system learns this preference and reasons differently next time

The manager understood the intent, reasoned about constraints, made a

judgment call, and took initiative.

Why "Agent" Is More Than Just Automation

The word "agent" carries specific meaning in both AI and organizational contexts—and that matters.

An agent has four essential qualities:

1. Autonomy

- Makes decisions without waiting for explicit instruction on each step
- Takes action based on interpreted goals, not just direct commands
- Example: Not "Click this button," but "Achieve this outcome"

2. Environmental Awareness

- Observes the current state (calendars, emails, files, user behavior)
- Recognizes what's changed since last interaction
- Adapts responses based on context
- Example: Understands that "Schedule a meeting" means something different at 3 PM on Friday than at 9 AM on Monday

3. Goal-Orientation

- Operates toward a defined objective, not just a sequence of tasks
- Can identify multiple paths to that goal and choose the best one
- Recognizes when the original approach won't work and pivots
- Example: Goal is "make sure the right people are informed," not "send this email to everyone"

4. Reasoning & Learning

- Can explain why it made a decision ("This time minimizes disruption for Asia team")
- Adapts based on feedback and outcomes
- Anticipates unintended consequences
- Example: Learns that scheduling meetings at lunch consistently gets low attendance, so adjusts future suggestions

* * *

The Practical Difference

Traditional software answers a question: "How do I perform Task X?"

Agentic AI solves a problem: "What needs to happen to achieve Outcome Y?"

This shift is profound because:

- **Flexibility**: You don't need to anticipate every scenario
- **Scalability**: Agents handle complexity without proportional increase in programming
- **Trust**: When you understand why a decision was made, you can evaluate it more effectively
- **Capability**: Agents can tackle ambiguous, multi-step challenges that traditional automation struggles with

Where We Are (and Aren't) Today

Important context: Agentic AI is not science fiction, but it's also not yet universally deployed. Current agentic systems excel at:

- Research and information synthesis
- Customer service with real reasoning
- Code generation and debugging
- Content creation with strategic goals
- Complex scheduling and resource allocation

They still struggle with:

- Physical-world manipulation (robotics remain separate from reasoning)
- Situations requiring genuine human judgment or ethics
- Unpredictable, novel environments
- Tasks requiring real-time sensory feedback

The trajectory is clear though: Agentic capabilities are expanding rapidly, and understanding how they work—and how to work with them—is becoming essential.

* * *

Key Takeaway

An agent is not just a faster way to automate; it's a fundamentally different approach to solving problems. Instead of following instructions, agents understand intent, reason about context, and act autonomously to achieve goals.

The difference between a clerk and a manager isn't speed—it's judgment. And that judgment is what agentic AI brings to software.

* * *

PART II

Building with Python

(Hands-On Projects)

PART II — Building with Python (Hands-On Projects)

Category 1: Workflow Without Agents

(Automation, no "thinking")
These help readers **feel the limitation** of logic-only systems.

Project 1. Daily Task Automation System

- Input → rules → output
- No decisions, just steps

Project 2. Rule-Based Email / Alert System

- If this, then that
- Fixed logic

Project 3. Data Collection & Reporting Workflow

- Collect → store → display
- No interpretation

Category 2: Building Agents (Single Agent Focus)

(Now intelligence enters)

Project 4. A Simple Decision-Making Agent

- Takes input
- Evaluates options
- Chooses an action

Project 5. Goal-Oriented Agent

- Given a goal
- Breaks it into steps
- Executes one by one

Project 6. Memory-Aware Agent

- Remembers previous interactions
- Changes behavior based on past

Category 3: Workflow with Agents

(Agent placed inside a process)

Project 7. Agent-Assisted Workflow

- Workflow does the routine
- Agent handles exceptions

Project 8. Intelligent Triage System

- Workflow handles intake
- Agent decides priority

Project 9. Monitoring & Follow-Up Agent

- Watches data
- Acts only when needed

Category 4: Workflows with Multiple Agents

(Teamwork between agents)

Project 10. Manager–Worker Agent System

- One agent assigns work

- Others execute tasks

Project 11. Specialist Agents Collaboration

- Each agent has a role
- Final decision is combined

Project 12. End-to-End Multi-Agent System

- Intake agent
- Decision agent
- Action agent
- Monitoring agent
  (Very close to real-world systems)

Workflow Without Agents

Workflow Without Agents

Project 1: Daily Task Automation System

(Input → Rules → Output | No decisions, just steps)

1. Problem Statement (Real-life)

Every day we do the **same small tasks**:

- Send a daily reminder
- Generate a simple report
- Move a file from one folder to another
- Send a "Good Morning" or "Daily Summary" email

These tasks are **not difficult**, but they are **repetitive**.
Humans forget them, get tired, or do them late.

👉 This is where **simple automation** helps.

2. What We Are Building

We will build a **Daily Task Automation System** that works like a machine:

Input → Fixed Rules → Output

There is:

- ❌ No thinking
- ❌ No decision-making
- ❌ No AI

Just **clear steps executed in order**, like a checklist that runs by itself.

* * *

3. How Humans Do This Today

Let's take one example: **daily reminder email**

A human usually:

1. Checks the time
2. Opens email
3. Writes the same message
4. Sends it
5. Repeats this every day

This wastes:

- Time
- Attention
- Mental energy

And sometimes… we simply forget 😄

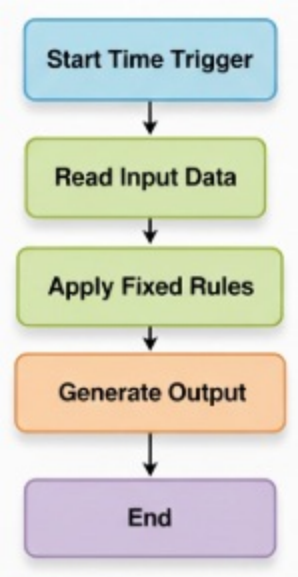
4. Agent / Workflow Design (Simple Automation)

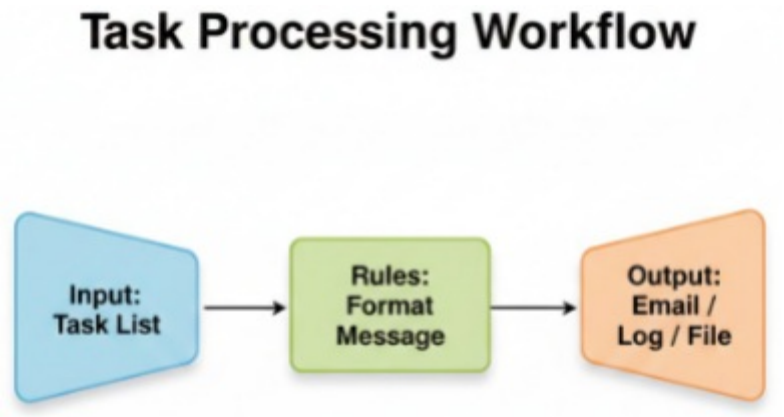
Even though this book is about *Agentic AI*, **this project has NO agent**.

Think of it like a **washing machine**:

- You give clothes (input)
- The machine follows fixed steps
- Clean clothes come out (output)

Workflow Steps





5. Python Code Snippets (Very Simple)

Example: Daily Console Reminder

```
from datetime import datetime

def daily_task():
    today = datetime.now().strftime("%d-%m-%Y")
    message = f"Daily reminder: Stay focused! Today is
{today}"
```

```
    print(message)

daily_task()
```

📌 What this does:

- Gets today's date
- Creates a message
- Prints it

No intelligence. Just steps.

Example: Writing Daily Task to a File

```
def save_task():
    with open("daily_log.txt", "a") as file:
        file.write("Daily task completed\n")

save_task()
```

This is automation at its simplest:

- Input: predefined text
- Rule: append to file
- Output: updated file

6. Prompts Explained Simply (If Applicable)

🚫 No prompts are needed here

Why?

- Prompts are for AI
- This system does not "understand" anything

- It only **executes instructions**

Like a calculator:

- You press buttons
- It gives results
- It does not think

7. Common Mistakes

1. **Overcomplicating**
- Adding AI where it is not needed
2. **Hard-coding everything**
- No flexibility for small changes
3. **No logging**
- You should know if the task ran or failed
4. **No error handling**
- Even simple systems can break

8. How to Extend This

Once this works, you can slowly grow it:

- Add a **scheduler** (run every morning)
- Send **email instead of print**
- Read tasks from a **CSV or database**
- Add a **notification** (SMS / WhatsApp)
- Later → introduce **decision logic**
- Much later → introduce an **agent**

👉 This project is the **foundation**
Without this, agentic systems become messy.

Key Takeaway

Automation comes before intelligence.

First teach the system **how to work**.
Later, you can teach it **how to think**.

This project is your **first brick** in that journey.

Project 2: Rule-Based Email / Alert System

(If this → then that | Fixed logic)

1) Problem Statement (Real-life)

In real life, we often need **alerts** when something crosses a limit.

Examples:

- Temperature is too high → send alert
- Stock is low → inform store manager
- Server is down → notify admin
- A student is absent 3 days → inform parent

Humans can do this by checking again and again, but that is tiring and error-prone.

2) What We Are Building

We will build a **Rule-Based Email / Alert System**.

It works like a strict teacher
It follows fixed rules:

IF something happens, THEN send an email/alert

No AI, no "thinking", only **predefined rules**.

3) How Humans Do This Today

Take a simple example: “low stock alert”

A human does:

1. Opens the stock sheet
2. Checks each item
3. Finds items below threshold
4. Messages the store team
5. Repeats daily

This causes:

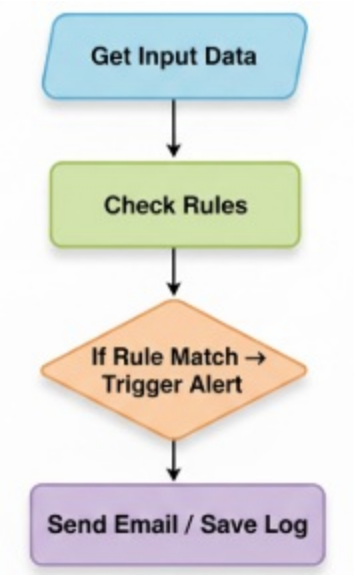
- Missed alerts
- Late action
- Unnecessary effort

4) Agent / Workflow Design (Fixed Logic)

This is still **not an agent**.
It is a simple “traffic light rule” system.

Core Flow

* * *



Example Rules (simple)

- If `temperature > 40` → alert
- If `absent_days >= 3` → alert
- If `balance < 500` → alert

5) Python Code Snippets

A) Simple Rule Check (No email yet)

```
def check_temperature(temp_c: float) -> str:
    if temp_c > 40:
        return "ALERT: Temperature too high!"
    else:
        return "OK: Temperature normal."

print(check_temperature(42))
```

This is the heart of rule-based automation:

- Input: `temp_c`

- Rule: `> 40`
- Output: message

B) Send Email Alert (Basic SMTP Example)

You can use Gmail SMTP (works for many people), but you may need an **App Password** if 2-step verification is enabled.

```
import smtplib
from email.mime.text import MIMEText

def send_email_alert(subject: str, body: str, to_email: str):
    from_email = "YOUR_EMAIL@gmail.com"
    app_password = "YOUR_APP_PASSWORD"  # safer than your
normal password

    msg = MIMEText(body)
    msg["Subject"] = subject
    msg["From"] = from_email
    msg["To"] = to_email

    with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
        server.login(from_email, app_password)
        server.send_message(msg)

def temperature_monitor(temp_c: float):
    if temp_c > 40:
        send_email_alert(
            subject="High Temperature Alert",
            body=f"Temperature is {temp_c}°C. Please take
action.",
            to_email="RECIPIENT_EMAIL@gmail.com"
        )
        print("Alert sent!")
    else:
        print("No alert needed.")

temperature_monitor(42)
```

* * *

☑ This is “If this, then that” in real life.

6) Prompts Explained Simply (If Applicable)

⃠ No prompts needed.

Because:

- The system is not generating smart text
- It is sending a fixed alert message
- The rules are written by you

Think of it like a **thermostat**:

- If too hot → switch ON the fan
  No AI needed.

7) Common Mistakes

1. **Too many rules in one place**

- Hard to read and maintain

2. **No “else” handling**

- System becomes confusing when rule does not match

3. **Alert spam**

- If you check every minute, you may send 100 emails
  ☑ Fix: add “cooldown time” or “send once per day”

4. **Hard-coded thresholds**

- Better to store thresholds in a config file later

5. **No logging**

- Always store “what happened” somewhere

* * *

8) How to Extend This

Easy upgrades (still rule-based):

- Read input from **CSV / Google Sheet**
- Write alerts to a **log file**
- Add **cooldown**: don't send same alert repeatedly
- Add **multiple rules** using a list of rules
- Add **multiple channels**: email + SMS + WhatsApp (later)

Next-level upgrade (later in the book):

- Replace fixed thresholds with **dynamic thresholds**
- Or use an agent to **summarize alerts** in human-friendly language

Key Takeaway

This system does not "understand".
It only obeys rules.

But if your rules are good, it becomes **very useful** in real life.

About the Author

Santanu Karmakar is a teacher, writer, and technology professional with a long-standing interest in **education, healthcare systems, and practical uses of AI**. Over the years, he has worked closely with students, professionals, and real-world systems, which has shaped his belief that **technology should simplify life, not complicate it**.

Rather than focusing only on theory, Santanu prefers to understand **how systems work on the ground**—how people interact with them, where processes fail, and how small, well-designed technological steps can make a meaningful difference. This thinking naturally led him to explore **agentic AI**, where software systems do more than follow instructions and instead assist humans in thinking, deciding, and managing complexity.

Santanu writes mainly for **non-technical and semi-technical readers**. His style avoids heavy jargon and complex mathematics. He believes that even advanced ideas can be explained clearly if they are connected to **everyday examples** such as offices, hospitals, classrooms, and service workflows.

He is the author of several fiction and non-fiction works and actively uses AI as a **thinking partner**—to organize ideas, test system designs, and improve clarity —while keeping human judgment and responsibility at the center.

Through his books, Santanu aims to:

- make emerging technologies understandable,
- encourage responsible and human-centered AI use,
- and help readers think like **system designers**, not just tool users.

When not writing, he spends his time teaching, learning, and quietly observing how people, systems, and technologies interact in real life—an observation that strongly influences his work.