CLASS 1 · PART B

Getting Started with *Claude Code*

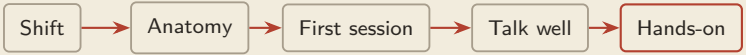
Coding basics and agentic systems — for researchers new to coding.

Jiaqi Shao · NUS Business School

Today: From Zero to Your First Data Script

You will tell the agent what to do, read its work, and check the result.

- Part I — The shift. From typing code to giving an agent a clear task.
- Part II — How it works. Model, tools, environment, and the *harness* that joins them.
- Part III — First session. Set up, work safely, and check the result.
- Part IV — Give clear instructions. Context engineering: prompts as clear briefs.
- Part V — Hands-on. Your first real analysis, on real data.



Four short quizzes along the way — think first, then check the answer.

Three Ideas That Run Through This Course

We introduce these ideas today and build on them in Classes 2 and 3.

1 — Agentic partner

Treat the AI as a *junior collaborator*

It plans, acts, and checks. Give it a task, not a request for a code snippet.

2 — Context engineering

A good prompt gives the right context

Its answer depends on what it can see. Give it the files, facts, and limits it needs.

3 — Second brain

Projects can carry *persistent memory*

A shared knowledge file is read every session. Its edit history is saved, so the project keeps what you learned.

Today: all three introduced · Class 2: memory & scale in full · Class 3A: the second brain for your *reading*.

NUS BUSINESS SCHOOL

What We Carry from This Morning

Class 1A named the risks. This afternoon builds the working answers.

Risk (Class 1A)	Plain meaning	Today's answer
Hallucination	confidently wrong — fluent even when fabricating	the *verify* habit
Sycophancy	tends to agree with whatever you suggest	ask it to *check*, not confirm
Automation bias	*you* over-trust it because it's a machine	audit outputs, always

Session 1 explained the risks. Today we practise how to manage them.

PART ONE

The Shift

- *From typing code to directing an agent.*

The Barrier Just Fell

For decades, you needed to write code to do empirical work. That is changing.

- Plain language is the new interface.
  Describe the analysis; the agent writes the code.
- Trying an idea costs minutes, not days.
  More questions asked; more data explored.
- Judgement becomes the key skill.
  Steering and verifying, not typing syntax.

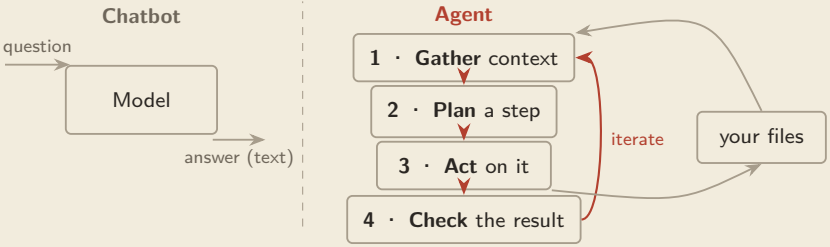
Today's promise

By the end of this session you will have run a real analysis on real data — having written *none* of the code yourself.

You no longer need to type every line. You do need to ask clearly and check the result.

What Changes When the AI Can Act?

A chatbot returns text — and leaves all the work with you. An agent closes the loop.



- Chatbot: every cycle still runs *through you* — copy, run, read the error, return.
- Agent: Gather · Plan · Act · Check — then around again until the goal is met. Gray arrows: *Act* writes to your files; their new state returns as evidence for the next *Gather*.

A chatbot returns text; an agent returns *changes in your project*.

Quiz 1 — Think First

Ten seconds on your own, then compare with your neighbour.

Three tasks are on your desk this week. For each — *vote*: hand it to the agent, or do it with a plain tool?

(a) Merge CRSP and Compustat into a firm-year panel.
(b) Rename 200 files from `.TXT` to `.txt`.
(c) Work out what a coauthor's 40-script replication package does.

Answer on the next slide — no peeking.

Quiz 1 — Answer

Task	Verdict	Why
(a) CRSP–Compustat merge	**agent**	multi-step, needs judgement + verification
(b) rename 200 files	**plain tool**	mechanical & deterministic — one line does it, no judgement involved
(c) decode the replication package	**agent, as reader**	*understanding* unfamiliar work is what it does best — Part II shows why

- Can't write the one-line rename? Ask the agent to *write* it once — then reuse it, free.
- The split is not "easy vs. hard" — it is mechanical-deterministic vs. judgement work.

Delegate work that needs understanding; keep mechanical work with mechanical tools — tools it can write for you.

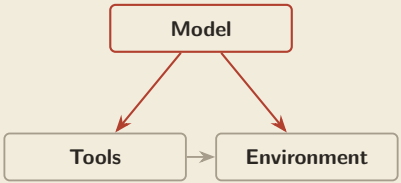
PART TWO

What It's Made Of

- *The system, the harness, the loop, and the tools.*

A Model Needs Tools and a Place to Work

An agent needs three parts. It also needs software that connects them.



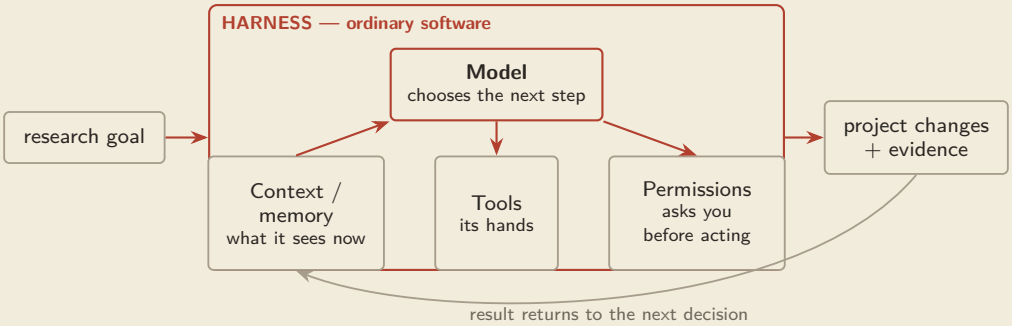
- Model — the intelligence.
  Claude Opus or Sonnet — two model tiers, deeper vs. faster; either works today.
- Tools — its hands: read, write, run.
- Environment — your project folder and terminal, where it works.
  The gray link: every read or write the tools do happens *inside* the environment.

Something must hold these together and *drive* them …

An agentic system uses a model, tools, and an environment to complete a goal.

The Harness Connects Reasoning to Action

A model can suggest a step. The harness lets it inspect files, act, and check the result.



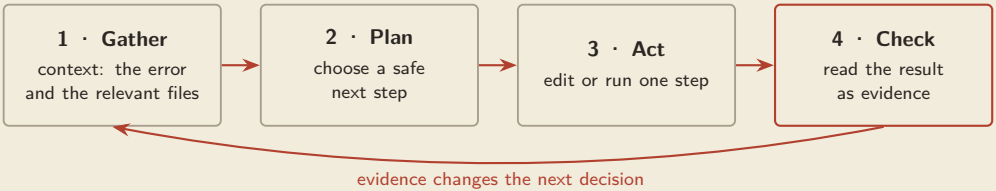
No harness, no agency. It routes context, tools, permissions, and results around the model.

Permissions — it asks before acting: the approval gate you will meet in Part III.

An Agent Works by Repeating, Not Guessing Once

Research need: the date parser failed, but the correct fix is not obvious yet.

Date parser — the code step that turns text like "2023-05" into real dates.



Why the loop matters

A failed run becomes evidence, not a dead end. The agent can revise its plan.

Where you enter

You approve consequential actions and decide whether the final evidence is convincing.

A Junior Collaborator, Not a Code Generator

Course message #1 — give it a clear brief, as you would a junior colleague.

Asking a tool

- "write me a loop over these files"
- you assemble the pieces yourself
- every request starts from zero

Delegating to a partner

- background · goal · what "done" looks like
- it plans, executes, and reflects
- check with me before anything irreversible

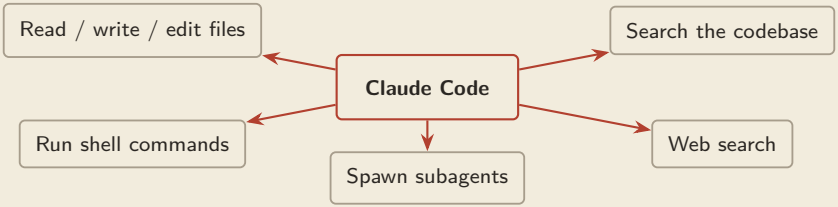
What makes it a *collaborator* rather than a text generator is the harness.

The harness gives the model tools, memory, and a review loop. It can act, check its work, and take corrections.

Class 2 opens the hood on the harness.

The Tools It Comes With

Out of the box, the agent can do what a developer does at a keyboard.



- Run shell commands *executes your analysis*; subagents take side-tasks. Extensible via MCP — the Model Context Protocol, a standard socket for plugging in new tools (Class 2).

The toolset turns a question-answerer into an investigator: it looks things up instead of guessing.

Its First Superpower Isn't Writing Code

For researchers, one of the best uses comes before writing any code.

Discovery · explanation · design

- "What's in this folder? Explain this dataset."
- "What does my coauthor's script actually do?"
- "How would you structure this analysis?"

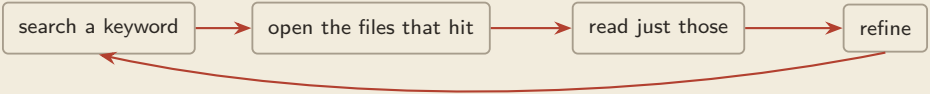
Why this matters here

The hard part is often not *producing* code. It is *understanding* code and data you didn't create.

Use it as a reader before you use it as a writer.

How Can It Find the Right File Without Reading Everything?

How does it read a project it has never ingested?



- The search runs *inside the loop*: search, inspect, refine — each finding changes what it looks for next, like a researcher following clues.
- The tool can search the local project on demand; only selected content enters the model context.
- It can inspect the current file state instead of relying only on an earlier indexed copy.

It searches your project the way you would — on demand, not from a pre-built index.

RAG and Agentic Search Solve Different Problems

Both find useful information. They differ in how they choose what to read next.

RAG — retrieve from an index

RAG = Retrieval-Augmented Generation

- finds *chunks* — small pre-cut pieces of your files — relevant to a query
- uses a persistent *index* — a pre-built lookup table (vectors or keywords)
- strong for repeated questions, one corpus
- freshness depends on index refresh

Agentic search — decide what to inspect

the pattern from the previous slide

- chooses a search, reads the result, then revises
- can use file search, databases, the web — or RAG
- strong for multi-step investigation
- quality depends on stopping and verification rules

They can work together: RAG supplies candidate evidence; an agent decides what to inspect next and when the answer is sufficiently supported.

Agentic search decides what to inspect next; it does more than one lookup.

The Honest Privacy Answer

"If I use confidential data, what leaves my machine?" Give a precise answer.

What does happen

- snippets it *opens and reads* go to the model's API

  API — the model's cloud service, reached over the internet

- treat it like any cloud AI service

What does not

- no bulk upload of your folder
- no stored cloud copy or index

Why researchers care

Data-use agreements (CRSP, Compustat, proprietary data) may forbid bulk copying to a third party. On-demand reading you can actually reason about — only what's opened leaves the machine.

Restricted data: institution policy first.

It does not upload the whole folder. But any content it opens is sent to the API.

NUS BUSINESS SCHOOL

It Can Also Remember: `CLAUDE.md`

Conversations are forgotten when a session ends — the model has no "yesterday."

- A plain text file in your project, read automatically at the start of *every* session.
- Write a rule once — every future session starts already knowing it.
- Versioned with the project; shareable with coauthors.

```
# CLAUDE.md
- Sample period: 2010--2023
- Raw data in data/raw/ is
  read-only --- never modify
- Tables follow JF format
```

persistent · versioned · shared

The seed of the project's **second brain** (course message #3). Built properly in Class 2.

`CLAUDE.md` — the project's memory file: persistent knowledge the agent re-reads every session.

Quiz 2 — Think First

Ten seconds on your own, then compare with your neighbour.

Your supervisor asks: "If you use this on our licensed CRSP data — is that compliant?" *Vote A, B, or C:*

A. "It's fine — nothing ever leaves my machine."

B. "It uploads our whole database to the cloud, so we can't."

C. "It reads only files it opens; those snippets go to the API — no bulk copy, no stored index. So: does our license permit sending excerpts to a processor?"

Processor — data-agreement language for a third party that handles data on your behalf.

Answer on the next slide — no peeking.

Quiz 2 — Answer

C — it is the only accurate answer.

- A is wrong — content it reads *does* go to the API.
- B is also wrong — it does not copy the whole folder or keep a cloud index.
- C explains both facts and asks the right compliance question.

PART THREE

Your First Session, Safely

- *How do you let an agent act without giving up control?*

NUS BUSINESS SCHOOL

What Must Be Ready Before the First Research Task?

Start the agent inside the right project folder, so it can see the right files.

1. Install Node.js (from `nodejs.org`) — the runtime it needs. npm, the installer used in step 2, ships *with* it — nothing extra to get.
2. In the terminal (the text window where you type commands: Terminal on macOS, PowerShell on Windows), install Claude Code:

```
npm install -g @anthropic-ai/claude-code
```

3. Move into your project folder: type `cd`, a space, then *drag the folder into the terminal window* — Enter. (Same trick on macOS and Windows.)
4. Start it — and, on the *first* run only, sign in:

```
claude
```

A browser page opens — log in with your Claude account. Plan or pay-as-you-go billing: ballpark on the backup slide.

Works in a terminal, or inside the VS Code integrated terminal. Package names can change — if a command fails, check `docs.claude.com/en/docs/claude-code`.

What Happens After You Type `claude`?

After you type `claude`, here is what you will see and how to stay in control.

1. Type `claude` — a plain text prompt appears. No windows, no buttons.
2. Type a request in ordinary English. Enter.
3. Watch it narrate its own loop: searching, reading, proposing.

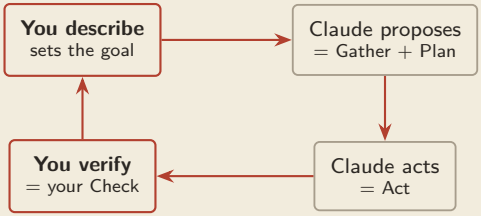
Three controls, from minute one

- `Esc` — interrupt instantly
- `/cost` — session spend so far (typical ballpark: backup slide)
- `/clear` — wipe, start fresh

You are never trapped — `Esc` is always there.

Who Is in Control During a Session?

The same Gather–Plan–Act–Check loop from Part II — now seen from your side.



Where you stand

- you *start* the loop: describe
- you *close* the loop: verify
- in between, it gathers, plans, and acts

The agent's own loop — with you inside it at two points: you start it, and you close it.

Would You Let This Command Touch the Raw Data?

The request sounds routine. The file path reveals the real risk.

```
Claude wants to run:  python analysis.py
Claude wants to edit:  data/raw/sales.csv  (+12 lines, -3 lines)
[y] allow once  [a] always allow  [n] deny
```

- Before choosing, inspect three things: target path, intended change, and whether the action is reversible.
- Here the safe response is **deny**, then steer: "Keep raw data unchanged; write cleaned output to `data/clean/`."

Picked "always allow" and regret it? It can be reverted — `/permissions` or the settings file; check current docs.

Before you approve, check what it will do, which files it will touch, and whether you can undo it.

Why Can a Plausible Result Still Be Wrong?

A clean chart and a confident explanation can still come from a broken date parser.

Hallucination

- The model has no feeling of "I don't know."
- Knowing or not, it answers in the *same* fluent, confident voice.
- When it doesn't know, it fabricates something *plausible*.

The one-liner

Confidence is not correctness.

This is a normal risk of the technology, not a rare glitch.

Hallucination — fluent, confident output that is fabricated; the model cannot tell you when it's guessing.

What Can You Catch Before Anything Runs?

Before anything runs — read the plan. You're reading intent, not code.

- The approval gate (or a full plan) says, in English: *"I will load this file, drop rows before 2010, compute monthly means."*
- Wrong file? Wrong years? Caught at zero cost — before the error exists.
- The habit: never wave a plan through on autopilot.

Cost–benefit

Ten seconds reading intent
beats an hour debugging results.

How Would You Notice a Silent Error?

Write down what you expect before the agent shows you a convincing result.

Expectation, stated first	Then compare
"This merge should give ∼4,500 firms."	got 212? — something's wrong
"Mean monthly return under 2%."	40%? — units or dates broke
"No missing values in the ID column."	3% missing? — investigate first

- You have something the model doesn't: you know your data — its magnitudes and shape.
- Expected-vs-actual turns that knowledge into a clear test.
- A gap tells you exactly where to investigate.

The First Output Is Evidence, Not the Answer

After the run — audit results against sense; correct with one sentence.

Audit

- sample size right? signs plausible? dates in range?
- "Walk me through what this script does, line by line, in plain English."
- ask it to write a check; ask it to test itself

Steer, don't restart

- "The dates parsed as strings — fix that."
- "Use the median, not the mean."
- small corrections; the loop does the rest

It is an excellent explainer of its own work. You audit; it explains.

"But I Can't Read Code"

You can check the result even if you cannot read every line of code.

- Reviewing an RA's work, you don't re-derive every line — you audit the *results*: sample size, signs, magnitudes, the figure that should slope up.
- Verification here is the same skill, and you already have it: domain expertise, applied at the output.
- The real danger is the opposite instinct: *"the machine wrote it, so it's probably right."* That is **automation bias** — Class 1A's third risk, live in your workflow.

You don't verify code — you verify claims about your data. Nobody is better placed than you.

Quiz 3 — Think First

Ten seconds on your own, then compare with your neighbour.

Mid-task, this approval box appears. *Vote: approve or deny — and what tipped you off?*

```
Claude wants to run: python clean_data.py
Claude wants to edit: data/raw/crsp_monthly.csv (+8 lines, -8 lines)

[y] allow once  [a] always allow  [n] deny
```

(Your project rule: raw data is read-only.)

Answer on the next slide — no peeking.

Quiz 3 — Answer

Deny. The second line edits `data/raw/` — your read-only zone.

- The *intent* ("clean the data") sounds fine — the file path is the tell. Read paths, not just intent.
- Waving it through because the plan sounded sensible? That is **automation bias**, live.
- Deny costs nothing. Then steer: *"Don't modify raw files — write cleaned output to `data/clean/` instead."*

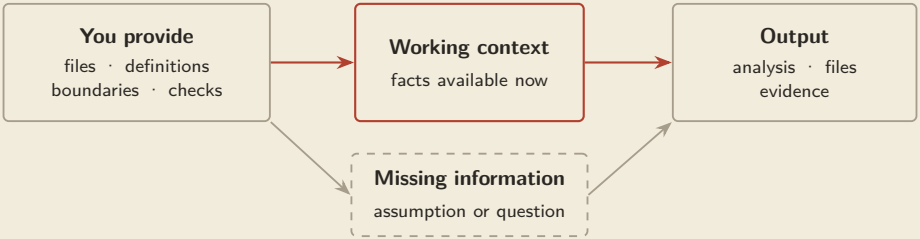
PART FOUR

Talk to It Well

- *What must the agent know before it can produce defensible work?*

What Must the Agent Know — and What Must It Guess?

"Analyze `sales.csv`" leaves key questions open: which measure, which dates, which output, and which checks?



Context engineering means giving the agent what it needs for important choices.

Speak in Plain Language

Permission granted: you never need jargon.

What you'd otherwise have to code	What you say
`df.groupby('month').revenue.sum()`	"Plot monthly revenue."
`df.isna().sum()`	"Which columns have missing values?"
`df.describe()`	"Give me an overview of this dataset."

The interface is your research question — not Python syntax.

A Prompt Contract Removes Five Sources of Guessing

Turn a vague request into a clear research brief that you can check.

- Context — the situation, the files
- Goal — what "done" produces
- Example — what good output looks like
- Constraints — the guardrails
- Verification — how we'll know it worked

Context: `@sales.csv` has one row per order.
Goal: compute monthly revenue and average order value.
Example: return a table of month, revenue, AOV.
Constraints: pandas only; keep 2020–2024; don't modify the raw file.
Verification: revenue positive, all 12 months present — report both.

(Verification is your expected-vs-actual, written down.)

`@sales.csv` — the `@` sign hands the agent that exact file; full tour later this part.

Always include Verification: say how you will know the result is right.

The Contract, on a Real Research Task

This brief uses plain research language and makes the task clear.

Context: `@crsp_monthly.csv` and `@compustat_annual.csv`, firm-level research project.

Goal: merge them into a firm-year panel.

Example: output columns: permno, year, return, assets.

Constraints: keep 2010–2023; never modify the raw files — write output to `merged/`; flag firm-years that fail to match.

Verification: expect roughly 4,500 firms; report the match rate.

Give the files, goal, example, limits, and checks in one clear brief.

Constraint ≠ Micromanagement

Set clear limits, but let the agent choose the steps.

Constraint — draws the boundary

> "Keep 2010–2023. Don't touch the raw files. Use only libraries already in the project. Flag unmatched firm-years."

Defines what *correct* means; leaves the path open.

Micromanagement — grabs the wheel

> "First open the CSV with pandas, then rename column 6, then call merge with how='inner', then loop over years and…"

Locks the agent into your first guess and makes it harder to adapt.

You define the right result. The agent works out how to produce it.

Why Boundaries Beat Step Lists

The reason is the loop from Part II.

- The agent's whole power is that it re-plans: discovers the misnamed column and adapts; hits a missing month and works around it.
- A step-by-step script freezes its plan at your first guess — it can't adapt without violating your instructions.
- Boundaries leave the loop *alive inside a safe region*; step lists switch it off.

You define what "correct" means; it finds the path.

You write acceptance criteria; it writes implementation
— exactly like the best RA you ever had.

NUS BUSINESS SCHOOL

Point at Your Files with @

Use @ to give the agent the exact file instead of making it guess.

> Summarise `@data.csv` and list any `@notes/codebook.md` fields it is missing.

- Type @ and the first letters — pick from the list that pops up; the agent receives that *exact* file.
- No copy-pasting, no guessing. Wrap paths that contain spaces in quotes.
- One keystroke — the cheapest context-engineering move of all.

Pointing beats pasting, and pasting beats describing.

Understand Any Project in Four Questions

Discovery-before-writing, as a recipe. Inherited a messy replication folder? Interview it.

1. "Give me an overview of this project."
2. "How is the data processed, start to finish?"
3. "Trace what happens to one observation through the pipeline."
4. "Draw a diagram of that flow."

Plain-language answers — it will genuinely sketch the architecture. Fifteen minutes instead of a lost afternoon.

Interview the project — start every unfamiliar folder this way.

Quiz 4 — Think First

Ten seconds on your own, then compare with your neighbour.

The merge you asked for comes back:

```
firms: 212  mean monthly return: 38.4%  sample: 2010--2035
```

You expected ∼4,500 firms, returns well under 2%, sample ending 2023.

Name the *three* checks you'd run — then write the *one steering sentence* you'd type.

Answer on the next slide — no peeking.

NUS BUSINESS SCHOOL

Quiz 4 — Answer

Anomaly	Check
212 firms, not 4,500	merge keys and join type — where did the rows go?
38.4% mean return	units — percent vs. decimal, or dates parsed wrong
sample runs to 2035	date parsing / the 2010–2023 filter never applied

One steering sentence: "I expected ∼4,500 firms, mean returns under 0.02, and a sample ending 2023 — show the row count after each merge step and diagnose all three gaps."

"Under 0.02" *is* "under 2%" written as a decimal — stating the units removes the exact trap from check two.

Compare expected and actual results. You can find problems without reading the code.

PART FIVE

Your First Script

- *Can you turn an unfamiliar CSV into a result you can defend?*

Can You Trust the First Plot from an Unfamiliar CSV?

Research need: understand the data, create one trend figure, and leave visible checks. ∼20 minutes.

```
Inspect @data.csv and report its shape and missingness.        [file + verification]
Plot the first date-like column against the first non-ID numeric column.   [goal]
If either choice is ambiguous, stop and ask me.                [verification]
Save the figure as trend.png.                                  [output]
```

A mini-contract — every clause tagged: file, goal, output, verification.

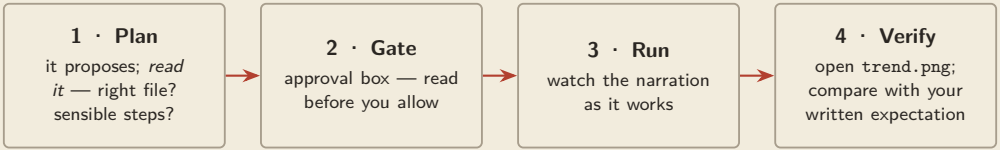
Shape = rows × columns · missingness = empty cells per column.

[Setup] Insert the actual `data.csv` path (course folder or chat link) before class — or use any CSV students brought.

- First, write down one *expectation* — "about 500 rows," "sales peak in December."
- Then: terminal → cd to the folder (type cd, space, drag the folder in, Enter) → `claude` → go. Stuck? Hand up.

A Trustworthy Result Passes Four Milestones

The four milestones — this slide stays up while you work.



At milestone 4? Check your neighbour's plot too — fresh eyes catch what owners miss.

Verify, Then Steer

The first result is a draft — read it and correct course.

Good signs

- runs cleanly, no errors
- statistics match your *written* expectation
- the plot is labelled and readable

If it's off — state the fix, re-ask

- "The dates aren't parsed as dates — fix that."
- "Use the median, not the mean."
- "Label the axes and add a title."

Satisfied? Save a snapshot — a *commit* in Git, the standard version-history tool:
type `git init` once in the folder, then simply ask the agent to "commit this result." Reproducible from here.

Small corrections beat starting over — that's the loop doing its job.

When Things Go Wrong (They Will)

Three classic failures, three one-line fixes. Notice what's not on the list: panic.

Failure	Fix
An error you don't understand	paste it back: "this failed — fix it"
A number that looks wrong	your expected-vs-actual fired: say what you expected, let it reconcile
It went off in a strange direction	`Esc`, then `/clear`, restate with a tighter contract

No panic, no starting over, no reading Python — failures are steering opportunities. That is the workflow.

When *Not* to Reach for the Agent

Judgement cuts both ways — match the tool to the task.

Delegate the task itself	Have it *write* you a one-line tool
semantic understanding of text	a simple search-and-replace
multi-step tasks across files	counting word frequencies
unfamiliar code or data	a trivial rename

- The line is mechanical-deterministic vs. judgement: one right answer, no interpretation → a plain tool wins on speed, cost, and reliability.
- You never need to know these tools: ask the agent *once* for the one-liner — a rename command, a *regex* (a one-line text-matching pattern) — then reuse it yourself.

Mechanical and deterministic? Have it *write* the one-line tool — don't burn a reasoning session running it.

NUS BUSINESS SCHOOL

Three Things to Remember

One sentence: direct it well, understand what it is, check what it does.

1

Direct, don't type.

Describe outcomes in plain language — like briefing a junior collaborator.

2

The harness makes it act.

Model + tools in a loop — which is why boundaries beat step lists.

3

Always verify.

Hallucination makes checking non-negotiable; your domain judgement makes you the best checker alive.

After Today

Three pointers before we close.

- Study notes. Everything today — including quiz answers and every prompt on these slides — is written down: `Slides_Jiaqi/notes/study_notes_A.md`. Don't transcribe; it's done.
- Practice this week. One real task on your own data — twenty minutes, use the contract, verify the output.
- Next — Class 2: research scale. Plan Mode, the `CLAUDE.md` second brain in full, test-driven debugging, and running several agents in parallel. Bring the same data.

References

E. Schoppik. *Claude Code: A Highly Agentic Coding Assistant*. DeepLearning.AI short course, 2025. `deeplearning.ai/short-courses`.

Anthropic. *Claude Code Documentation*. `docs.claude.com/en/docs/claude-code`.

Anthropic. *Claude Code: Best Practices for Agentic Coding*. Engineering blog, 2025.

Risk vocabulary (hallucination, sycophancy, automation bias): Class 1A slides and references.

CLASS 1 · PART B

Thank you.

Ninety minutes ago, a terminal was foreign territory.
You've now directed an agent through a real analysis — and verified it.

Next: Class 2, research scale.

Backup Slides

Reference material for questions.

Backup: Setup Troubleshooting

- VS Code: type `claude` in the integrated terminal; the extension auto-installs. If it fails, ensure the `code` command is on your `PATH`.
- Restricted network: a proxy may be needed to reach the service; check the official setup page for your region.
- Check your install: run `claude` in any folder — a session should start.

Backup: Cost & Plans

- Subscription — Pro or Max plan; a Pro plan is enough for this course.
- Pay-as-you-go — an API key billed per token.
- Track spend — `/cost` shows the current session's usage.
- Ballpark — an exercise like today's typically costs well under a few dollars on pay-as-you-go; a Pro subscription covers the course's sessions.

Plan names and limits change — confirm current details before class.

NUS BUSINESS SCHOOL

Backup: A Pocket Glossary

Term	Plain meaning
Terminal	the text window where you type commands
cd	"change directory": type cd, space, drag the folder in, Enter
npm	Node's package installer — ships with Node.js
Script	a file of instructions the computer runs
CSV	a spreadsheet saved as plain text
Function	a reusable named step
Commit	a labelled snapshot of your work in Git
@	in a prompt: hands the agent that exact file

Term	Plain meaning
Context window	the model's working memory — its "scratch paper"
Hallucination	fluent, confident output that is fabricated
Harness	the frame that gives the model tools and a loop
API	a program's door to a service — here, the model's cloud
MCP	Model Context Protocol — a socket for new tools
RAG	answering from a pre-built index of documents
Regex	a one-line text-match pattern
Processor	a third party handling data for you

Backup: The Class 1A Risk Map, One Slide

For when a risk question goes deep — the morning's frame, compressed.

Layer	Failure mode	Today's counter
Output / epistemic	hallucination; sycophancy	verify moves 1–3
Information-set	stale or partial context	context engineering; `@` files
Delegation / alignment	acting beyond intent; automation bias	approval gate; read plans; audit outputs

Full treatment: Class 1A slides (Shirley). Vocabulary used consistently across all sessions.