this+that

What to Automate First

A field guide to putting AI to work

A practical guide from **this+that** · thisandthat.chat

How to use this book

Most advice about AI at work starts in the wrong place. It hands you a list of impressive things the technology can do and leaves you to figure out which of them matter for your actual week. That is backwards, and it is why so many teams buy a tool, automate one thing in a burst of enthusiasm, and then quietly stop.

This book starts from your work instead of from the technology. It gives you a repeatable way to find the automations hiding in your own week, a way to decide which one to build first, and a way to build it so it survives contact with reality. You will not need to write code. You will need about a week of light attention and a willingness to be honest about how you actually spend your time.

The method has a shape:

- **Find** the work worth automating (the Follow-the-Work Audit).
- **Score** it, so you build the one that pays off first.
- **Place the human**, so nothing important runs without a checkpoint.
- **Build** one, measure it, and let it compound into a system.

You can read straight through, or you can start the audit today and read the rest while a real week of evidence accumulates. The second way is better.

You will see this+that turn up as a worked example in a couple of chapters, because it is the tool we know best. The method itself is tool-agnostic on purpose. Finish the book, go automate your week with something else, and it did its job.

Chapter 1: The problem isn't the work, it's the work around the work

Ask someone what they would automate if they could, and you tend to get answers pitched at the level of a job title. "Automate my email." "Automate reporting." "Automate scheduling." These feel like answers, but you cannot build any of them, because none of them is a thing you actually do. They are categories, and categories do not have triggers, steps, or outputs. They are too big to hand to a machine and too vague to hand to a person.

The work that can actually be automated is smaller and more specific, and it is mostly invisible. It is not the meeting, it is the notes and the three follow-ups the meeting generates. It is not closing the deal, it is the CRM update, the recap email, and the reminder to check back in ten days. It is not the client relationship, it is the "any update?" that arrives on a Tuesday and sends you digging through four tools to reconstruct where things stand.

Call this the work that follows the work. It is the connective tissue between the things you were actually hired to do. Nobody put it in your job description, it rarely shows up on a task list, and it quietly consumes a large share of your week precisely because it is too small and too constant to notice.

This is good news, because the work that follows the work has three properties that make it a near-perfect target for automation:

- It is **repetitive**. The same triggers produce the same chores, over and over.
- It is **low-judgment**. Looking up a project's status does not require your expertise. Deciding what to do about that status might, but the looking-up does not.
- It is **interruptive**. It arrives at random and pulls you out of the work that does need your expertise. The cost is not just the minutes, it is the context switch.

You do not have to automate your job to get most of your time back. You have to automate the drag around it. The rest of this book is about finding that drag and removing it, deliberately, one piece at a time, without breaking anything you care about.

Throughout, we will follow one running example: Maya, who runs client accounts at a small agency. She is not drowning because the work is hard. She is drowning because every client, every channel, and every open thread generates a steady stream of small follow-through, and there is no one to hand it to. She is exactly the person this method is built for, and if your week looks nothing like hers, the method still holds. Only the examples change.

Chapter 2: What AI can actually automate right now (and what it can't)

Before you go looking for automations, it helps to know what you are looking for, because the technology is genuinely good at some things and genuinely bad at others, and the gap between them is where most failed automation projects live.

Here is the map, without the marketing gloss.

AI is reliably good at:

- **Reading and sorting messages.** Pulling the signal out of a noisy inbox, spotting what is a request versus an FYI, grouping related threads. This is mature and dependable.
- **Extracting the work from a message.** Turning "can you get me the Q3 numbers before Thursday" into a concrete task with an owner and a due date. Also mature.
- **Drafting.** First-draft replies, summaries, recaps, updates. Good, sometimes very good, and always in need of a human read before anything with stakes goes out.
- **Moving information between tools.** Updating a record, posting an acknowledgement, setting a reminder, filing a note where the next person will find it. Dull, mechanical, and exactly what you want off your plate.
- **Following a defined sequence of steps**, including simple decisions along the way. "If the client is on the enterprise plan, route to the senior manager, otherwise reply with the standard update." This is where real workflows live.

AI is unreliable or inappropriate at:

- **High-stakes judgment.** Whether to fire a client, how to handle a sensitive complaint, what to promise in a negotiation. You can let AI prepare the ground. You should not let it make the call.
- **Anything where a wrong action is expensive and hard to reverse.** Sending money, deleting data, making public commitments, changing who has access to what. These stay in human hands regardless of how routine they feel.
- **Acting without current context.** AI is only as good as what it knows. An automation working from stale or missing information will produce confident, fluent, wrong output. This is the single most common cause of automation that erodes trust, and we will keep coming back to it.

The practical takeaway is a rule of thumb you will use in every chapter that follows: **automate the gathering, the moving, and the drafting. Keep the deciding, and gate anything you cannot easily undo.** Most real work is a mix of both, which is why the goal is

almost never to automate a whole task end to end. It is to strip the low-judgment drag off the front of it and leave a clean decision for a person.

Hold onto that framing. It is what keeps the rest of this practical instead of reckless.

Chapter 3: The Follow-the-Work Audit

You cannot decide what to automate by sitting in a room and brainstorming it. Every team that tries produces the same list: "answering emails," "scheduling," "reporting." Those are job categories, not automations, and they are too big and too vague to hand to anything. The work worth automating is smaller, more specific, and mostly invisible until you catch it happening.

So we are not going to brainstorm. We are going to run an audit on one real week of your own work, and let the automations surface on their own. It takes about a week of light attention and produces a backlog you can actually build from.

Step one: catch the triggers

A trigger is any message or event that reliably kicks off a bit of recurring work. A client emails "any update on this?" A bug gets reported in Slack. An invoice lands. A meeting ends and now three people are waiting on notes. A new hire starts Monday.

For one week, keep a running note, on your phone, in a doc, wherever you will actually use it, and every time you notice yourself starting a small chore because something came in, jot the trigger down. One line. You are not solving anything yet. You are just catching the moments where a message turns into work.

Do not filter. The instinct is to only write down the "important" ones, but the best automation candidates are usually the small, dull, frequent triggers you have stopped noticing precisely because they are so routine. Those are the ones quietly eating your week.

By Friday you will have a real list of triggers. That list is the raw material for everything that follows.

Step two: trace the follow-through

This is the move the whole method turns on.

For each trigger, write down the little chain of work that follows it. Not the headline task, the chain. When a client asks for a status update, what do you actually do? You go find the latest state of their project. You check what shipped since you last spoke. You look up whether that open question ever got answered. You draft a reply in their tone. You maybe update the CRM so the next person knows you responded. Then you send it.

That is six steps hiding behind "reply to the client." Most of them are not the reply. They are the looking-up, the checking, the updating, the remembering. This is the work that follows

the work, and it is where automation lives. The reply itself might need your judgment. The five steps around it usually do not.

Write the chain for every trigger. Be honest about the small stuff, the "I just quickly check X" steps you do on autopilot, because those are exactly the steps that cost you focus and never show up on any task list.

What falls out when you do this

Once you have the chains written down, something useful happens: each step sorts itself into one of three buckets, and you can see it at a glance.

Automate it. Steps that are frequent, patterned, and need little judgment. Looking up the latest project activity. Updating the CRM. Setting a reminder. Posting a standard acknowledgement so the client knows they were heard. These are pure follow-through, and a machine should do them.

Turn it into a task for a person. Steps that genuinely need judgment, the actual decision, the delicate reply, the "should we give this client the discount" call. You do not automate these away. But notice what usually makes them slow: it is not the deciding, it is the gathering you had to do before you could decide. So automate the gathering, and hand the decision to the right person as a task with everything already attached. The judgment stays human. The forty-five minutes of prep in front of it does not. A good task is a decision made easy, not just a reminder to do work.

Drop it. Some chains, once you see them written out, turn out to be work you should not be doing at all. A report nobody reads. A status meeting that a shared update would replace. The audit is worth running for these alone.

Most real chains are a mix. The client-update example is five automate-able steps wrapped around one human sentence. That is the normal shape, and it is good news: you rarely have to automate a whole job to remove most of its drag.

The thing every chain quietly depends on

Trace a few chains and you will notice how often a step is really "know something." Know this client's history. Know what we shipped last sprint. Know the refund policy. Know who owns this account now. Every automate step and every hand-off-to-a-person task is only as good as the context behind it. An automated client update built on stale information is worse than no update. A task routed to a colleague without the background just moves your confusion onto their desk.

So as you audit, mark the knowledge each chain needs. You are not building a wiki, and you should resist the urge to, because hand-maintained knowledge bases go stale the week after

you build them, which is why the ones you already have are half-wrong. What you are looking for is the context that would need to be true and current for this work to run without you: the project state, the account facts, the policy, the recent decisions. The best version of this keeps itself current from the same stream of messages you are already auditing, rather than from someone remembering to update a doc. Hold that thought. It is what separates automations that survive contact with reality from the brittle ones that break the first time the facts move, and we will build on it in Chapter 7.

For now, just tag it. Next to each chain, note what it needs to know.

What you have at the end of the week

Not a vague sense that "we should use AI more." A concrete backlog:

- a list of real triggers from your actual week,
- the follow-through chain behind each one,
- every step tagged automate, task, or drop,
- and the context each one depends on to be any good.

That is an automation backlog grounded in evidence instead of imagination, and it is worth more than any list of trendy use cases, because it is yours. In the next chapter we score it, so you build the one that pays off first instead of the one that happened to be top of the list.

> **Do this now:** start the note today. One line every time a message turns into work. Do not plan, do not filter, do not build anything yet. Just catch a week. The audit only works if the raw material is real. There is a ready-made spreadsheet for this, with these columns and the scoring built in, if you would rather not start from a blank page.

Chapter 4: Scoring what's worth automating

You now have a backlog of work-chains. If you try to automate all of them, you will automate none of them well. The point of scoring is to find the one that pays off first, build it, and earn the confidence to build the next.

Score each chain on three axes, each from 1 to 5, set up so that 5 always means a stronger case to automate this one first. Keep it rough. This is a triage tool, not accounting.

Frequency: how often does this trigger fire? 5 is many times a day; 1 is once or twice a quarter. The more often a chore fires, the more a small saving compounds, so frequent ones score high. Automating something rare is a common trap: it feels satisfying and saves almost nothing.

Drag: how tedious and how spread out is the follow-through? 5 is a chain that spans several tools and eats fifteen minutes of clicking and switching; 1 is a single quick action in one place. Drag is where the felt pain lives, and it counts the cost of breaking your focus, not just the minutes. A chore can be frequent but low-drag (a one-line reply) or rare but high-drag (a painful monthly reconciliation), and this axis is what separates them.

Automatability: how much of the chain can a machine actually do? 5 means a machine can do essentially all of it with little or no human judgment; 1 means it hinges on a human decision. This axis does double duty. It feeds the score, because a fully automatable chain is an easier, lower-risk win, and it tells you the shape: a 5 you automate end to end, a 1 or 2 you automate the prep for and route the decision to a person as a task.

Priority = Frequency + Drag + Automatability. Add the three; the highest total is what you build first. Here is Maya's backlog, scored:

Work-chain	Frequency	Drag	Automatability	Priority	What to build
Client asks "any update?"	5	4	4	13	Auto-gather the project state and draft the update, approve and send
Log a call outcome in the CRM	4	3	5	12	Fully automate from the call notes
Weekly status report nobody reads	1	5	5	11	Drop it; replace with a shared live view
Onboard a new client	1	5	3	9	Worth a workflow eventually, but not first
Decide whether to escalate a complaint	2	2	1	5	Automate the prep, route the decision as a task

The winner is clear once it is on the page. The "any update?" chain tops the list: it fires constantly, it is a real slog, and a machine can do almost all of it, leaving Maya one sentence to approve. The CRM logging is a close second and even simpler, low enough on judgment to run fully hands-off.

Two things the scoring protects you from. The onboarding workflow is tempting because it is elaborate and visibly valuable, but it fires rarely and is a large build, so it waits. And the weekly status report, less obviously, scores high yet should be killed rather than automated: a high score means the chain deserves attention, not that a machine should run it. Automatability tells you a machine could do it, never that it should exist. Score to decide what deserves attention, then decide whether the answer is automate, route, or delete.

Pick one. Just one. The next three chapters are about building it well.

Chapter 5: Where the human belongs

The fastest way to lose trust in automation, your own and your team's, is to let it do something wrong that you cannot take back. Trust is the whole game. An automation people trust gets used and expands. An automation people do not trust gets switched off, and it poisons the well for the next one. So before you build anything, you decide where the human sits.

There are three checkpoints, and every automation uses one of them:

Full-auto. The automation acts on its own, no human in the loop. Reserve this for chains where the cost of a wrong action is low and easily reversed. Logging a call in the CRM. Posting an acknowledgement. Filing a note. If it goes wrong, you fix it in a moment and nobody was harmed.

Approve-before-send. The automation does all the work and then waits for a person to say go. This is the workhorse checkpoint, and it is where almost everything with any stakes should start. The client update gets fully drafted, with the context gathered and the reply written, and Maya reads it and hits send. She saves the fifteen minutes of prep and keeps the one moment of judgment that matters.

Draft-for-human. The automation prepares, but a person does the final act deliberately, often because the act itself needs care or a personal touch. The escalated complaint, where the automation attaches the full history and a suggested response, but a senior person writes and sends the actual reply.

The guiding question is not "can this be automated?" It is "what does it cost if this is wrong, and can I undo it?" Cheap and reversible, let it run. Expensive or permanent, gate it, every time, no matter how routine it feels. Sending money and changing who has access to things stay human even when they happen daily.

And then there is the trust ramp, which is the part most people miss. You do not have to pick a checkpoint forever. Start new automations one notch more cautious than you think you need, usually at approve-before-send. Watch them. When an automation has drafted forty client updates and you have approved thirty-nine of them without a change, you have earned the right to let that one run closer to auto, or to auto entirely for the low-stakes slice of it. Trust is built by evidence, and the approve step is how you gather it. It is not bureaucracy. It is the on-ramp.

Chapter 6: Build your first automation

Time to build the thing. We will walk Maya's top-scored chain, the "any update?" request, end to end, in plain terms that map onto whatever tool you use. An automation, stripped of jargon, is five parts: a trigger, the steps, the context it draws on, a checkpoint, and a record.

1. Define the trigger precisely. Vague triggers produce chaos. Not "when a client emails," but "when a message from a client contact asks for a status or progress update." Precision here is what keeps the automation from firing on the wrong things. Most tools let you match on sender, channel, and intent, and the intent match is where modern AI earns its place, because it can tell "any update?" from "please update my billing address."

2. Lay out the steps. For Maya's update, the steps are: gather what has happened on this client's project since the last contact, check whether any open question from the last thread got resolved, and assemble that into a short, plain update in her voice. Each step is something AI does well: retrieve, check, draft.

3. Wire in the context. This is the step people skip, and it is why their automations feel generic and wrong. The draft is only as good as what it knows about this client. It needs the project's current state, the recent decisions, the tone this client expects. In Chapter 7 we make this a proper system. For your first build, at minimum point the automation at the real source of project truth rather than letting it guess.

4. Set the checkpoint. Per Chapter 5, this one starts at approve-before-send. The automation produces a finished, ready-to-send update, and Maya reads it and sends it. She is doing five seconds of judgment instead of fifteen minutes of prep.

5. Leave a record. After it sends, the automation logs that the client was updated, so the next person to touch the account can see it, and so you can measure whether the thing is working.

That last point matters more than it looks. **Measure your first automation, or you will not know if it is helping.** Two numbers are enough: how many times it ran this week, and how often you changed its output before approving. Frequent runs and rare changes mean it is working and earning the right to run closer to auto. Rare runs mean you automated something too infrequent, go build the next candidate instead. Frequent changes mean the context is wrong, which is the subject of the next chapter.

This is roughly the shape of what our own product does, for what it's worth: this+that reads the client's message, pulls the relevant project activity from the connected tools, drafts the update, and holds it for approval until you trust it. But there is nothing proprietary about the shape. Any capable automation tool can be arranged this way, and the discipline, precise

trigger, context wired in, a checkpoint, a record, and a measurement, matters far more than which tool you pick.

Ship it. Run it for a week. Then come back, because one automation is a time-saver, and a system is something else entirely.

Chapter 7: From one automation to an operating system

One automation saves you a chore. The reason to keep going is that automations compound, but only if you build them as a system rather than as a drawer full of disconnected gadgets. This chapter is about the difference, and it turns on the thing you have been tagging since Chapter 3: knowledge.

Here is the failure mode to avoid. Most people, once they get a taste for it, build automation after automation as isolated point-to-point wires. This trigger updates that tool. That message pings this channel. Each one works in a demo. Then the world changes: an account changes hands, a policy updates, a project pivots, and every automation that hard-coded the old facts starts producing confident nonsense. You end up with a pile of brittle wires that need constant babysitting, which is worse than no automation at all, because now the babysitting is the job.

The thing that makes automations durable instead of brittle is a shared layer of current context that all of them draw on. Call it a knowledge layer. When Maya's update automation, her onboarding workflow, and her escalation task all read the client's state from one living source of truth, three things happen. They stay consistent with each other. They keep working when the facts change, because they read the facts fresh instead of remembering an old copy. And each new automation gets cheaper to build, because the context it needs is already there.

Now, the real problem with knowledge layers: they go stale. Every team has a wiki that was accurate the month it was written and has been quietly rotting ever since, which is why nobody trusts it. A knowledge layer that a human has to maintain by hand does not solve the brittleness problem, it relocates it. The maintenance becomes the new drag.

The version that actually works keeps itself current from the same stream of work the automations are already watching. The messages, the updates, the decisions flowing through your channels are a live feed of how the business is changing. A knowledge layer fed from that feed stays close to true without anyone stopping to update a document. This is the hardest and most valuable piece of the whole picture, and it is genuinely early technology, the automatic upkeep is emerging rather than solved, so test it before you rely on it. But the direction is right, and it is worth designing toward even while the automation is partial and some of the upkeep is still yours.

This is the operating system view: a stream of triggers on one side, a set of automations and human tasks on the other, and a living knowledge layer in the middle that grounds everything so it acts with current context. We built this+that around exactly this shape, an inbox that

catches the triggers, automations and tasks that handle the follow-through, and a knowledge layer, the Brain, that grounds them, but the shape is the lesson, not the product. Build toward the system, not the gadget drawer, and your second year of automation is worth more than your first instead of collapsing under its own maintenance.

Chapter 8: Common failure modes

Almost every automation that fails does so in one of a handful of predictable ways. Knowing them in advance is cheaper than learning them one disaster at a time.

Automating a judgment call. The most damaging mistake. Something needs a person's discretion and you handed it to a machine because it happened often. Frequency is not the test. Judgment is. Automate the prep, route the decision.

No checkpoint on something expensive. You let an automation take an action that was costly and hard to reverse, and one day it took the wrong one. Anything you cannot easily undo gets a human gate, forever, regardless of how routine it feels.

Stale context. The automation acts on information that used to be true. This is the quiet killer, because the output looks fine, fluent and confident and wrong, so you do not catch it until a client does. It is the reason Chapter 7 exists.

Paving the cowpath. You automated a broken process instead of fixing it, and now the broken process runs faster. Before you automate a chain, ask whether it should exist at all. Sometimes the right automation is deletion.

No measurement. You built it, you assume it is helping, and you never checked. Without the two numbers from Chapter 6, run count and change rate, you cannot tell a working automation from a decorative one, and you cannot know when it has earned more autonomy.

Automating the rare. It was fun to build and it fires once a quarter. All that effort for something that saves almost nothing. Score before you build.

The brittle-wire pile. Covered in Chapter 7. Point-to-point automations with no shared context that break the moment the facts change. The cure is a knowledge layer, not more wires.

Read this list before every build. Most of these are obvious in hindsight and invisible in the moment, which is exactly why a checklist beats good intentions.

Chapter 9: Your 30-day starter plan

Everything in this book fits into one month of light, deliberate effort. Here is the sequence.

Week 1: Audit. Run the Follow-the-Work Audit from Chapter 3. Keep the running note. One line every time a message turns into work. Do not build anything. By Friday you have a backlog of real triggers and their follow-through chains.

Week 2: Score and design. Score your backlog on frequency, drag, and automatability (Chapter 4). Pick the single highest-priority chain. Decide its checkpoint (Chapter 5). Sketch its five parts: trigger, steps, context, checkpoint, record (Chapter 6). Resist the urge to pick three. One.

Week 3: Build and run. Build the one automation. Start it at approve-before-send. Let it run for the week and watch it. Approve its output, and notice every time you change something before you do, because that is your signal about whether the context is right.

Week 4: Measure, graduate, expand. Check your two numbers. If it ran often and you rarely changed its output, it is working, and you can let the low-stakes part of it run closer to auto. Now build your second automation from the backlog, and this time, wire both of them to read from one shared source of context, so you are building a system and not a second gadget. Then start designing where a living knowledge layer fits, because from here the compounding begins.

At the end of the month you will have two working automations, a scored backlog of the next ten, a checkpoint discipline that keeps you safe, and the beginnings of a system. More than that, you will have the skill the whole book was really about: the ability to look at your own week and see the work that follows the work, which is a thing you can now do for the rest of your career, with any tool, as the technology keeps getting better.

That skill does not go stale. The tools will change. The habit of finding the drag and removing it, carefully, with a human where it counts, is the durable part.

Appendix: worksheets and a starter library

The audit worksheet

For each trigger you catch during Week 1, fill one row:

Trigger (what came in)	Follow-through chain (the steps you do)	Each step: automate / task / drop	Context it needs to know

The scoring sheet

For each chain, before you build:

Chain	Frequency (1-5)	Drag (1-5)	Automatability (1-5)	Priority	What to build

Priority = Frequency + Drag + Automatability; build the highest total first. All three run 1 to 5, with 5 meaning a stronger case to automate. Automatability also tells you the shape: a 5 you automate end to end, a 1 or 2 you automate the prep for and route the decision as a task.

Both worksheets above are available as a ready-to-use spreadsheet, with the Priority scoring built in and the example already filled in, so you can start from a working page instead of a blank one.

A starter library, by role

Candidate automations most people in these roles find in their audit. Use them as prompts, not prescriptions. Your real backlog comes from your own week.

Founder / owner

- Incoming intro or inbound request: gather who they are and draft a triage reply, you approve.
- Investor or board update: assemble the numbers and recent milestones into a draft, you finalize.
- "Can we meet?" from anyone important: check your real availability and propose times.

Operations

- New hire starts: fire the onboarding checklist across the tools that need accounts and access requests (access changes routed to a human, never auto).

- Recurring vendor invoice: match it, file it, flag anything off-pattern as a task.
- Weekly metrics: gather them into a shared live view, and kill the report nobody reads.

Agency / account management (Maya)

- "Any update?": gather project state, draft the update, you approve and send.
- Call or meeting ends: log the outcome and create the follow-up tasks from the notes.
- Client goes quiet: detect the dropped thread and surface it before it becomes a problem.

Sales

- Lead replies: update the CRM stage and draft the next touch, you approve.
- Deal stalls past a threshold: raise it as a task with the full history attached.
- Post-demo: send the recap and set the follow-up reminder automatically.

Customer success

- Support signal that hints at churn risk: route it as a task with the account context, do not auto-reply.
- Renewal approaching: assemble the account's health and usage into a prep pack for the human conversation.
- Recurring how-to question: draft the answer from your existing docs, you approve.

Written by the team at this+that. The method is yours to use with any tool. If it helped you find and remove some of the work that follows the work, it did its job. Learn more at thisandthat.chat.