

Dashboard Structure Derivation

Hand the whole dataset to the smartest model you have and make it argue for a page structure from the data instead of guessing a layout.

01 When to use this

Use this before you design a single screen of a dashboard. It is the step people skip, and skipping it is why most internal dashboards are five pages nobody opens. Reach for it whenever the data lives in more than two places.

02 The prompt

I am building a marketing dashboard. Do not design anything yet and do not write any styling.

Here is the full dataset: `<PATHS_OR_CONNECTORS_TO EVERY TABLE>`. Read all of it, not a sample.

First, tell me what you actually see:

1. Which tables logically join, on which keys, and how confident you are in each join.
2. Where the coverage gaps are. Which fields or sources exist for only some of the records, and how many.
3. What the natural primary axis of this data is. By client, by channel, by time, by rep, or something else I have not thought of.

Second, propose `<NUMBER>` candidate page structures. For each one give me:

- The page list and what lives on each page.
- The argument for it, made from the data you just read, not from dashboard convention.
- What it makes hard to see.

Third, tell me which one you would pick and why, in plain language. Argue against the others.

Do not average the options into a compromise. I want distinct structures I can choose between.

03 What it is actually doing

Most dashboards are laid out the way dashboards are usually laid out. Overview, then acquisition, then traffic, then site. It looks organized and it is often wrong, because the layout came from convention rather than from what is in the data.

So you flip it. You give the model everything and you ask it what shape the data actually is. Which tables join. Where the holes are. What the story is usually about. Then you

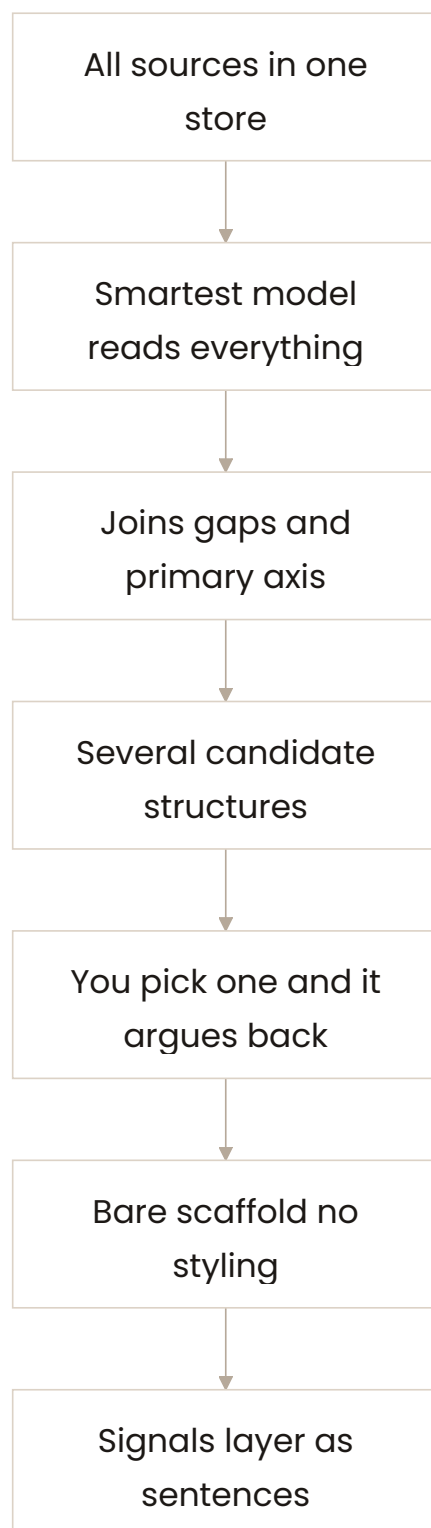
make it propose several structures and argue for each one, including what each structure hides.

This is the one place in the course where the model tier visibly mattered. A mid-tier model proposed the standard funnel layout. The top-tier model rejected that layout with an argument taken from the data itself: the anomalies were client-scoped, so a funnel layout scatters each client's story across five pages, while a client-centric layout keeps each story on one screen. The coverage gaps backed it up, because ecommerce data existed for only 4 of 12 clients, which punishes any page built to show all clients at once.

The other half of a dashboard worth reading is the signals layer. A chart makes you find the problem. A signal tells you the problem in a sentence. Something like client 10, week of March 9, 3 leads against a 58 lead baseline, critical. That is what people actually read.

Build the pages bare and data-first with no styling. Design comes after the structure is right.

04 How it flows



05 Step by step

01 Get every source into one place

Ad spend, sessions, revenue, CRM, email. The reason internal dashboards are bad is that the data lives in six systems and nobody joins it.

02 Use the best model available

This step rewards intelligence more than any other in the build. Pay for the top tier here and economize somewhere else.

03 Give it the whole dataset

Not a schema, not a sample. Coverage gaps only show up when the model can count how many records actually carry a field.

04 Ask for joins and gaps first

Make it report what it sees before it proposes anything. The proposals are only as good as its read of the data.

05 Demand distinct candidates

Ask for several structures and forbid a compromise blend. Distinct options let you compare tradeoffs. A blend hides them.

06 Make it argue against the others

The reasoning is the deliverable. A structure you can defend from the data survives the first client meeting. A guessed one does not.

07 Add the signals layer

Specify threshold detection over a rolling window that surfaces anomalies as sentences, not charts. This is what makes the dashboard worth opening.

08

Scaffold bare, then design

Build the pages with real data and no styling at all. That scaffold is the shared starting point for the parallel design pass.

06 **Words explained****Structure derivation**

Working out the right page layout from what the data actually contains, instead of copying a standard dashboard shape.

Primary axis

The main thing every page is organized around. By client, by channel, by time. Pick wrong and every screen fights you.

Coverage gap

A field or source that exists for only some records. Ecommerce data for 4 of 12 clients is a coverage gap, and it kills global pages.

Signals layer

Automatic checks that watch numbers over a rolling window and write out anything unusual as a plain sentence.

Rolling window

A moving stretch of recent time, like the last eight weeks, used as the baseline for what is normal right now.

Scaffold

The unstyled first build. Real data, real pages, no design. It exists so structure and looks are judged separately.

07 Watch out for

- A mid-tier model gives you a plausible funnel layout with no argument behind it. This is the step to spend on.
- Feeding a schema instead of the real data hides the coverage gaps, and coverage gaps are what decide the layout.
- Asking for one recommendation gets you one confident answer. Ask for several so you can see the tradeoffs.
- Styling the scaffold early makes you defend a layout because it looks nice. Keep it ugly until the structure is settled.
- Cost reference: full derivation plus scaffold over 50000 to 100000 rows ran about 13 to 18 US dollars.

08 How it connects

Parallel Design Variants

The bare scaffold this note produces is the exact input the parallel design pass forks from.

Self-Healing and Error Logging Clauses

Once the dashboard refreshes on a schedule it becomes an unattended routine, so it needs both clauses to fail loudly.

Marketing Dashboard Build Pipeline

The framework note recording the full chain from ingestion to hosting, plus the model tier comparison.

Automate vs Keep Human Filter

The signals layer generates bad news as well as good. Do not auto-send the bad half to a client.

Credit. Framework and approach by Nick Saraev, from the dashboard structure derivation segment of his Zero to One course on Claude Code for marketing. Written up for the Kape vault by BrewedOps.