HAVISH M
CONSULTING

HMC ONE
AI STRATEGY. BUILT TO SCALE.

Generative AI Playbook for Vibe Coding



NEHA JHA

HAVISH MADHVAPATY

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Generative AI Playbook for Vibe Coding

Havish Madhvapaty and Neha Jha

havishmconsulting.com/downloads

DOWNLOAD ALL OUR

FILES + HANDBOOKS + CONTENT



About Havish M Consulting

Havish M Consulting (HMC) is an award-winning analytics and AI advisory firm and a proud member of NASSCOM, recognized among Siliconindia's Top 10 Data Analytics Companies of 2025. Since its founding, HMC has delivered over 2,500 corporate workshops — including sessions for Boards of Directors and C-suite leadership — across 45+ organizations spanning Energy, CPG, BFSI, Manufacturing, Retail, and Consulting, in 10+ countries across India, Middle East, UK and North America.

At the helm is Havish Madhvapaty, Founder of HMC and a leadership coach to Fortune 500 C-suites and Boards of Directors. He serves as visiting faculty at IIM Kozhikode, SRCC, and Masters' Union, and has taught internationally at NUS Business School and S P Jain School of Global Management. His work has been featured in Businessworld, The Economic Times, Entrepreneur, and Financial Express, and he has authored professional certification content with the Economic Times Masterclass series and for National Stock Exchange NCFM Pro Series. Havish was listed on BusinessWorld 40u40 Education, Analytics Insights 40u40 Analytics, Entrepreneur India and Indian Express Educator of the Year and has received Indian Achievers Award. He has also been nominated by Microsoft as an MVP.

Working alongside him is Neha Jha, Senior AI Consultant, who leads the day-to-day management of GenAI adoption journeys for HMC's enterprise clients. Neha is the connective tissue between HMC's frameworks and their real-world implementation — coordinating readiness surveys, cohort trainings, and rollout logistics across engagements with Amber Enterprises, Bharat Petroleum, RR Kabel, JK Cement and Landmark Group, among others. Her work ensures that HMC's adoption programs don't just launch well but sustain momentum long after the workshop ends.

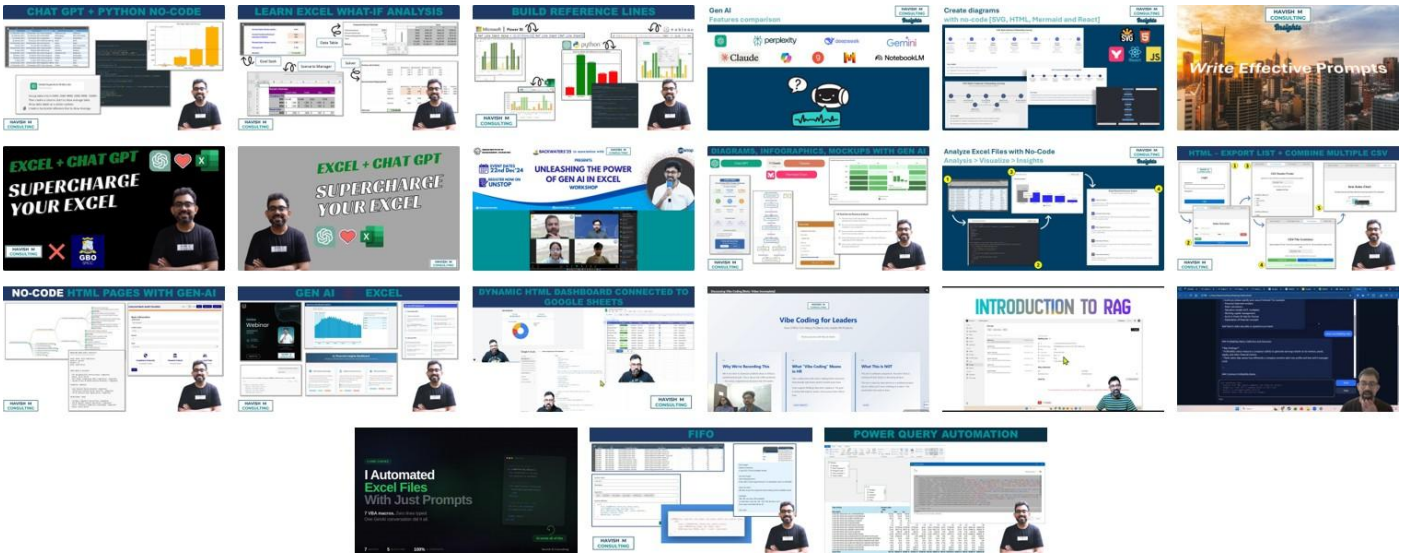
Backing this team is a strong advisory board, including Sanjay Kumar Suri (former CISO, Bharat Petroleum) and Jyotirmoy Bose (former CEO, Bennett Advisory) — bringing decades of enterprise security, governance, and strategic leadership experience to HMC's client engagements.

HMC's Generative AI adoption work sits at the heart of this book. The firm has created Generative AI labs for 8+ large organizations (USD 1 Billion +) guiding leadership teams from foundational awareness through functional training to enterprise-wide safety frameworks with measurable ROI.

YouTube

1600+ Videos covering Generative AI, Prompt Engineering, Agents, Excel Automation / Analytics / Dashboarding, Data Storytelling, No Code ML and more.

https://www.youtube.com/c/havishmadhvapaty



Key Clients


Bharat
Petroleum


amber


Welspun


LANDMARK
GROUP


JKCement


KĀBEL


pnb Housing
Finance Limited


DS GROUP

Client Testimonials



strategy&

Kushal Sinha

Partner @ Strategy&

“Havish adapts to audience requirements and enables the team to pick things up and run themselves. He covers advanced analytics techniques focusing on problem solving techniques.”



Shell

Yogesh Yadav

CFO at Shell Group Entity @ Sprng Energy

“Automation projects done helped us **streamline and accelerate** our workflows and enabled teams to complete **complex tasks faster**.”



IndusInd Bank

Siddharth Sinha

SVP and Head Analytics @ IndusInd Bank

“Wonderful session on GenAI. Really enjoyed”.



Rajeev Jain

SVP Marketing @ DS Group

“I have attended few training sessions from Havish on AI and enjoyed sessions thoroughly. I found him passionate about AI, having thorough knowledge and very sincere in approach. He understands need of trainees and customises sessions accordingly. I along with my team members learned a lot from him about applications of various AI tools.”



Rajwinder Singh

Head Equipment and Procurement LPG @ Bharat Petroleum

“Havish is an excellent trainer in Advanced Excel and Power BI. He is very competent and has the ability to respond quickly to any kind of queries. Had a great learning experience.”

GenAI Journey Snapshot across Clients

Client-anonymised view

Awareness = literacy and leadership buy-in at engagement start

Adoption = depth of embedded usage today

ROI = realised or tracked value

Industry	Sector	Awareness	Adoption	ROI
Wires & Cables	Electrical Equipment / Manufacturing	High at promoter and CHRO level; CIO governance concerns surfaced early	Growing — sanctioned enablement platform live; two flagship workstreams in sales planning and SCM	Medium-High — forecasting cycle compression, freight efficiency gains, reduced shadow-IT usage
Home Textiles, Pipes & Infrastructure	Textiles / Diversified Manufacturing	Strong promoter sponsorship across two distinct business lines	Function-deep rather than org-wide — planning and quality teams first	Medium-High — loom scheduling from days to hours; earlier defect capture; white-space led revenue pitches
Cement	Building Materials / Manufacturing	Concentrated in two business-unit heads; org-wide literacy still early	Executive-only by design — personal fluency before rollout	Medium — leadership self-sufficiency on analysis drafts; internal champions seeded ahead of scale
Retail (Fashion, Home, F&B)	Retail & Consumer	High at Gulf leadership level; category teams following	Mixed — leadership coaching alongside live pricing analytics	Medium-High — margin protection through better-timed markdowns; faster aged-stock clearance
Air Conditioners & Consumer Durables (OEM/ODM)	Consumer Durables / Manufacturing	Promoter-led from day one; baseline readiness survey segmented workforce into quadrant cohorts	Deep — enterprise licences at scale, always-on internal platform across plant, finance, commercial	High — analyst-hours reclaimed on BOM and variance reporting; board-level impact narrative funded Year 2
Steel & Power	Metals & Mining / Energy	Early-stage; discovery-led, building the	Pre-deployment — prioritised use-case pipeline, no	Early — avoided spend on low-value pilots; cross-functional

		business case before commitment	production rollout yet	consensus established
Pharmaceuticals	Healthcare / Life Sciences	Cautious but informed; governance framed the conversation before tooling	Selective — non-regulated functions only, use-case screening gate in place	Medium — faster market intelligence synthesis; reduced external research dependency
Banking	Financial Services	Function-led; strongest in Marketing, building in HR and IT	Broad but shallow — cross-functional training layer, workflow embedding next	Medium — lower agency dependency on first-draft creative; faster HR query turnaround
Trading, Retail & Real Estate	Diversified / Conglomerate	Leadership-driven; AI positioned as a strategy capability, not an IT one	Emerging — strategy and planning cell first, business units to follow	Medium — market scans in hours vs weeks; reduced first-pass consulting spend
Real Estate & Retail	Real Estate / Diversified	Engineering-function led; awareness strongest where estimation pain is felt	Targeted — design and estimation workflows in live use	Medium-High — faster bid-stage estimates; fewer revision cycles; senior engineer capacity freed
Power Transmission Infrastructure (EPC)	Power / Infrastructure	PMO-led; site teams onboarding progressively	Process-embedded — reporting and tracking layer across concurrent projects	Medium-High — multi-site status consolidation from days to hours; earlier delay flagging

Part — Basics

Article – Vibe Coding

Andrej Karpathy ✓
@karpathy

There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

3:17 PM · Feb 2, 2025 · **6.8M** Views

'Vibe coding' named word of the year by Collins Dictionary

6 November 2025

Share Save Add as preferred on Google

Laura Cress
Technology reporter



Getty Images

Article – What is HTML

Why HTML Matters for Non-Coders

When you ask AI to build a dashboard, calculator or tool, it usually gives you an HTML file. If you use vibe coding, you will work with HTML often. You do not need to write it. You need to understand it well enough to check it, fix small things and ask for the right changes.

Why AI gives you HTML

HTML is the simplest way to share a working tool.

- It opens in any browser. No software to install.
- It works on laptops and phones.
- It can be emailed, saved to a shared drive or shared as a link.
- One file can hold text, tables, charts, buttons and calculations.

The three parts of an HTML file

Most HTML files have three parts:

- HTML: what is on the page. Headings, text, tables, buttons.
- CSS: how the page looks. Colours, fonts, spacing.
- JavaScript: what the page does. Calculations, filters, clicks.

Knowing this helps you give clear instructions. If the colours are wrong, that is CSS. If the totals do not update, that is JavaScript.



Why this matters if AI writes the code

Understanding HTML helps you in three ways.

- You can check the output. Text, labels and numbers are written in plain language inside the file. You can find and verify them.
- You can ask for better changes. Clear feedback means fewer rounds with AI.
- You can make small edits yourself. Changing a title, colour or number does not need AI.

When to use HTML

- Use Excel for data entry and analysis.
- Use PowerPoint for presentations.
- Use HTML when people need to use something more than once, click on it or open it on any device.

Simple rule: if people will read it once, use a document or slides. If they will use it again and again, use HTML.

Risks to know

- Anyone who opens the file can see everything inside it. Do not put salaries, customer data or pricing in a shared file.
- There is no login or access control.
- Many copies can spread by email. Keep one file in one shared location with one owner.
- A file that looks right can still be wrong. Test calculations against known numbers before using them.

How to get started

1. Ask AI to explain the file section by section in simple words.
2. Open the file in a text editor like Notepad. Find text and numbers you recognise.
3. Make one small change yourself. Save it and reopen it in the browser.
4. When asking for changes, say whether the problem is content, look or function.

Summary

AI can write HTML for you. Your job is to know what it is, check that it works, keep data safe and ask for clear changes. That is enough to turn AI output into tools your team can use.

AI BUILDS · WHAT IS HTML · PART 1 OF 3

One HTML file runs on any device.

That's why AI gives you one.

You don't need to write HTML. **You need to read it well enough to check it, fix small things and ask for the right changes.**

WHY AI GIVES YOU HTML

Opens in any browser

Nothing to install

Laptops and phones

The same file on every screen

Email, drive or link

Share it like any other file

All in one file

Text, tables, charts, buttons and maths

THE THREE PARTS OF EVERY HTML FILE



HTML — **What is on the page**
Headings, text, tables, buttons

JS — **What the page does**
Calculations, filters, clicks

CSS — **How the page looks**
Colours, fonts, spacing

Name the part, and the fix gets faster.

Colours wrong? That's CSS. **Totals don't update?** That's JavaScript.

HMC ONE
AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE
© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · WHAT IS HTML · PART 2 OF 3

Will people use it again and again? Build it in HTML.

Knowing HTML lets you check what AI built, **and pick it only when the job needs it.**

WHY IT HELPS, EVEN WHEN AI WRITES THE CODE

01
Check the output

Text, labels and numbers sit in plain language inside the file.

02
Ask for better changes

Clear feedback means fewer rounds with AI.

03
Edit small things yourself

A title, colour or number doesn't need AI.

WHICH TOOL FOR WHICH JOB



Excel
Data entry and analysis



PowerPoint
Presentations



HTML
Tools people click, reuse and open on any device

The simple rule

Read once? Use a document or slides.

Used again and again? Use HTML.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · WHAT IS HTML · PART 3 OF 3

A file that looks right can still be wrong.

Test it, then keep it safe.

Four risks to know before you share, **and four steps to start working with HTML yourself.**

RISKS TO KNOW



Anyone can see inside

No salaries, customer data or pricing in a shared file.



There is no login

Nothing controls who opens or uses it.



Copies spread by email

Keep one file, in one shared place, with one owner.



Looks right, may be wrong

Test calculations against known numbers first.

HOW TO GET STARTED

1. Ask AI to explain the file section by section, in simple words.
2. Open it in Notepad. Find text and numbers you recognise.
3. Change one small thing. Save, then reopen it in the browser.
4. When asking for changes, say if it's content, look or function.

AI can write the HTML. Your job is the rest:

Know it. Check it. Keep data safe. Ask clearly.

That is enough to turn AI output into tools your team can use.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

Article – Safety of HTML

The business case

When you're building dashboards, reports, calculators, or internal utilities through AI-assisted ("vibe") coding, HTML is often the most practical output format — not because it's technically superior in every dimension, but because of what it means for deployment and adoption inside a business.

1. Zero-friction distribution. An HTML file opens in any browser — no installation, no license, no IT approval cycle. You can email it, put it on a shared drive, or drop it into a Teams channel, and the person on the other end just double-clicks and it works. Compare this to a Python script (needs an interpreter and dependencies) or a compiled app (needs OS compatibility, permissions, sometimes admin rights). For a consulting practice or an internal team rolling out a tool to dozens of non-technical stakeholders, this removes almost all deployment friction.

2. Self-contained and portable. A single HTML file can carry its own CSS, JavaScript, and even embedded data or charting libraries. There's nothing to "install" and nothing that breaks because a server went down or a subscription lapsed. This makes it ideal for calculators, one-off dashboards, or utilities that need to outlive a single project engagement — the client can keep using it long after your contract ends.

3. Fast iteration cycles. Because HTML/JS is quick to generate, test, and modify with AI assistance, you can go from "here's what I want" to a working prototype in the same meeting. This is a major advantage when you're co-designing a tool live with a business stakeholder — changes to layout, logic, or branding can be shown immediately.

4. No recurring cost or vendor lock-in. Unlike SaaS dashboard tools, an HTML deliverable has no subscription, no seat licensing, and no dependency on a third-party platform staying in business. For clients who want a tool they fully own, this matters commercially.

5. Branding and professionalism. HTML gives full control over look and feel — logos, color palettes, typography — so deliverables can be styled to match a client's or firm's brand identity, which matters when the output is client-facing.

On data safety: does uploading an Excel file into an HTML tool "go online"?

This depends entirely on *how* the HTML tool is built, and it's worth being precise here rather than making a blanket claim:

- **If the HTML file processes the upload entirely in the browser** (i.e., the JavaScript reads the Excel file locally using something like the SheetJS/xlsx library and does all calculations client-side), then the data never leaves the user's machine. Nothing is transmitted to a server, and there is no network call involved in reading or processing the file. In this case, it's accurate to say the data stays local and private.

If the HTML tool includes any server call — for example, it sends data to an API, calls a backend for AI processing, or logs data to an external service — then the file's contents (or parts of them) *are* going online, and this needs to be disclosed clearly to users.

AI BUILDS · THE SAFETY OF HTML

One HTML file reaches anyone. Where the data goes is the real question.

For AI-built dashboards, reports and calculators, HTML is the most practical format to ship: not because it is technically best, but because of how easily it gets adopted.

WHY HTML FOR VIBE-CODED TOOLS

01
Zero-friction
Opens in any browser. No install, licence or IT approval.

02
Self-contained
CSS, code, data and charts travel in one file.

03
Fast to iterate
Change layout or logic live, in the same meeting.

04
No lock-in
No subscription, no seats. The client owns it.

05
On brand
Full control of logo, colours and type.

DOES UPLOADING AN EXCEL FILE "GO ONLINE"? IT DEPENDS HOW THE TOOL IS BUILT.

Processed in the browser

Excel file .xlsx on your laptop → **Browser** reads and calculates → **Stays local** no network call

The JavaScript reads the file on the user's machine (e.g. with the SheetJS library) and does every calculation client-side.

Nothing is sent to a server.

Accurate to say: the data stays private

Any server call in the code

Excel file .xlsx on your laptop → **Browser** reads and calculates → **Server** API, AI backend or logging

If the tool sends data to an API, calls a backend for AI processing or logs to an external service, the file's contents, or parts of them, **go online.**

Disclose this clearly to every user

"It's HTML" is not a privacy guarantee.

Before an HTML tool goes to clients or colleagues, confirm whether any line of it sends data to a server. If none does, say the data stays on the device. If one does, say what is sent and where.

Be precise, not reassuring.

HMC ONE
AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE
© Havish M Consulting and/or affiliates. All rights reserved.

Article – Understanding JSON

Why You Need to Understand JSON When Vibe Coding HTML

Vibe coding makes it possible to build surprisingly powerful HTML dashboards, tools, and mini-apps without writing every line of code yourself. You can describe what you want to ChatGPT, Claude, Copilot, or another AI coding assistant, and get a working interface within minutes.

But once your HTML page starts working with real data, one concept becomes extremely important: **JSON**.

JSON — JavaScript Object Notation — is essentially a structured way of storing and exchanging data.

Think of HTML as the **interface**, JavaScript as the **logic**, and JSON as the **data structure connecting everything together**.

For example, instead of hardcoding employee information inside an HTML page, you might store it like this:

```
[
    {
        "name": "Rahul",
        "department": "Sales",
        "sales": 1250000
    },
    {
        "name": "Priya",
        "department": "Finance",
        "sales": 850000
    }
]
```

Your HTML dashboard can then read this JSON and automatically create tables, charts, filters, KPI cards, or summaries.

Why JSON matters in vibe coding

The biggest reason is that **most modern applications exchange data in JSON format**.

When your vibe-coded HTML application connects to an API, AI model, database, workflow tool, or another application, the information will very often arrive as JSON.

For example:

Excel / CSV → JSON → HTML Dashboard

API → JSON → JavaScript → HTML

ChatGPT / Claude API → JSON → Your Application

n8n / Power Automate → JSON → HTML Interface

Even if AI writes the code for you, understanding the JSON structure allows you to understand **what the application is actually doing with your data**.

You do not need to become a JavaScript developer. But you should be comfortable recognizing concepts such as:

- Objects: { }
- Arrays: [ ]
- Key-value pairs: "department": "Finance"
- Nested data
- Numbers, text, dates and Boolean values

This becomes especially useful when something goes wrong.

Imagine asking an AI assistant:

"Show sales by department."

The code might fail because your JSON uses "Dept" while the JavaScript is looking for "Department".

Without understanding JSON, this feels like a coding problem.

With basic JSON knowledge, you can immediately see that it is simply a **data structure mismatch**.

JSON also makes vibe-coded applications far more scalable.

Instead of writing:

Rahul - Sales - 12.5L

Priya - Finance - 8.5L

you store the data separately and let the application dynamically generate the interface.

Tomorrow, whether there are **10 records or 10,000 records**, the same HTML code can continue working.

This is the shift from building a static webpage to building a **data-driven application**.

The important mindset

Vibe coding does not mean that you need to understand every line of code generated by AI.

But you should understand the **building blocks**.

For most business users building HTML tools, dashboards, calculators, or internal applications, three concepts provide enormous leverage:

HTML tells the browser what to display.

JavaScript tells the application what to do.

JSON tells the application what data it is working with.

AI can write all three for you.

But once you understand JSON, you can give AI much better instructions, troubleshoot problems faster, connect more data sources, and build significantly more powerful applications.

That is why, for anyone learning vibe coding, **JSON is one of the highest-value technical concepts to understand.**

Article – Excel Upload Limit

If you have built a dashboard as a local HTML file, there is no standard Excel upload limit. A 200 MB workbook might load in one dashboard and fail in another. The answer depends on how the dashboard was coded, what the workbook contains, and how much memory the browser can use.

When you select a file in a local dashboard, the browser reads it from your computer. The page may have its own rule that rejects files above a certain size. If it does not, the next challenge is processing the workbook. Excel files can expand substantially in memory when the dashboard reads sheets, converts cells into data, and builds tables and charts. SheetJS notes that its standard reading functions load entire files into memory, where browser memory limits can become a constraint. (SheetJS Large Datasets)

This is different from **Excel's worksheet limit** of 1,048,576 rows and 16,384 columns. That limit describes how much a worksheet can hold; it does not set the maximum file size a local HTML dashboard can accept. (Microsoft Excel specifications)

If a 200 MB upload fails, first check whether the HTML code contains a file-size rule. If the page accepts the file but freezes or crashes, the likely issue is the work required to process and display it. Loading fewer sheets and columns, calculating summaries before drawing charts, and showing only a small preview of the rows can help. For consistently large files, preparing smaller or summarized data before opening the dashboard is often more reliable.

The practical answer: there is no universal "Excel upload limit" for a local HTML dashboard. Its working limit has to be tested with your actual file, browser, computer, and dashboard code. Converting the workbook to JSON alone does not guarantee that a large file will load.

Article – Working with Unclean Excel Files

Once you can generate a working HTML dashboard from a prompt, the bottleneck stops being the code. It becomes the file someone drags into it.

Almost every failure we see in these builds traces back to a spreadsheet that was designed to be read by a human and is now being read by a parser. Humans skip the title row, ignore the blank line, understand that the bold row at the bottom is a total. Code does none of this.

Twenty things to get right, on both sides — the file and the parsing logic.

Layout

1. Data starts at A1. No company name in row 1, no logo, no "Report as of March 2025", no blank spacer row. The parser takes the first row it finds as the header. Give it the header.

2. No merged cells anywhere. A merged cell resolves to one value and a series of empty cells. Merged headers are the worst version of this, because the columns underneath end up unnamed and everything downstream is silently misaligned.

3. One table per sheet. Two tables side by side, or stacked with a gap, will be read as one rectangle with a lot of holes. If a sheet has a summary block above the detail, split them.

4. Name the sheet you want. Parsers default to the first sheet, which in most corporate workbooks is a cover page or an instructions tab. Read sheets by name in the code, and fail loudly with a list of available sheets when the name is missing.

5. Hidden and filtered rows still come through. Filters and hidden columns are a display state, not a data state. If someone filtered to Q4 before sending the file, the code still sees all four quarters. Delete, don't hide.

Headers

6. No two columns with the same heading. Most parsers map rows to objects keyed by header. Two columns called "Amount" means one of them silently overwrites the other, and the dashboard will look completely plausible while being wrong.

7. No blank headings. An unnamed column becomes __EMPTY, __EMPTY_1, and so on. Either name it or drop it before upload.

8. One header row, not two. Stacked headers — a merged category row above a detail row — are a human convention. Flatten them into single names like Q1_Revenue, Q1_Units before the file goes anywhere near the parser.

9. Trim and normalise in the code. "Region " and "Region" are different keys, and the trailing space is invisible to whoever created the file. Strip whitespace, standardise case, and build a mapping layer rather than trusting the header string exactly as typed.

Data types

10. Numbers stored as text. The most common single cause of a chart that renders with everything at zero, or a total that concatenates instead of adding. Coerce explicitly and check what fails rather than assuming the type.

11. Currency symbols and separators live inside the cell. ₹1,20,000 is a string. So is $4.5M. Format currency in the display layer, keep the underlying cell as a raw number.

12. Percentages are ambiguous. A cell showing 45% may hold 0.45 or 45 depending on how it was formatted. Decide the convention, state it, and validate against a known total.

13. Dates are the biggest trap. Excel stores them as serial numbers, so 45231 is a date that will parse as a number. Converting through a JavaScript date without controlling for timezone routinely shifts everything by one day. And 03/04/2025 is March or April depending on who made the file. Force an unambiguous format like YYYY-MM-DD at source.

14. Leading zeros disappear. Employee IDs, PIN codes, cost centre codes, account numbers. 007 becomes 7, and your join against another table quietly returns nothing. These are identifiers, not numbers — keep them as text.

Rows and content

15. No total or subtotal rows inside the data. A grand total row will be charted as a data point roughly twice the size of everything else. Totals are something the dashboard calculates, not something it inherits.

16. No blank rows in the middle, no footnotes at the bottom. * Figures exclude the Nagpur plant is not a data row, but it will become one. Strip trailing rows that don't match the column structure.

17. Blank is not zero. An empty cell and a 0 mean different things — one is "no sales", the other is "we don't know". Averages change materially depending on how you treat them. Decide deliberately, and say which you chose on the dashboard itself.

18. Category values must be consistent. North, north, North and NORTH will produce four bars in a chart that should have one. Normalise on read, and surface the distinct values so someone can spot the duplicates.

19. Never use a pivot table or a formula-heavy sheet as the source. Pivots are output, not data. Formula errors — #REF!, #N/A, #DIV/0! — flow straight through as strings and break every numeric operation after them. Paste as values into a clean sheet.

Practical

20. Reference columns by name, never by position. row[3] works until someone inserts a column, at which point the dashboard is wrong rather than broken — which is far more dangerous. Map by header name and throw an explicit error when an expected column is missing.

Two more worth building in as habit rather than rules. Browser-side parsing struggles well before you reach enterprise data volumes, so set a row limit and tell the user what it is instead of letting the tab freeze. And be deliberate about what goes into the file at all — a browser tool is exactly the wrong place to discover that a sheet has salary data or customer names in columns nobody was looking at.

The habit that prevents most of this

One sheet. One table. Starting at A1. One row of unique headers. No merges, no totals, no footnotes, no formatting doing the work of data.

Almost every rule above is a specific instance of that. Most teams get there faster by building a clean extract sheet next to the working file than by trying to reform the working file itself — the operational spreadsheet has a human audience and should keep it.

The parser gets its own copy.

AI BUILDS · WORKING WITH UNCLEAN EXCEL FILES · PART 1 OF 2

Your dashboard is only as clean as the file

Humans skip the title row, ignore the blank line and know the bold row is a total. **A parser does none of this.**

Built for a human

A	B	C	D
Sales Report: March 2025			1
Region	Q1		Amount 2
North	03/04/25	₹1,20,000	45%
TOTAL		₹9,80,000	15
* excl. Nagpur			16

Built for a parser

A	B	C	D
Region	Date	Revenue	Margin
North	2025-03-04	120000	0.45
South	2025-03-04	98500	0.38
East	2025-03-04	131200	0.41
West	2025-03-04	87400	0.36
...			

LAYOUT

01 **Data starts at A1** no company name, logo, "as of" line or spacer row

02 **No merged cells anywhere** merged headers leave the columns beneath unnamed

03 **One table per sheet** two blocks on a sheet read as one rectangle full of holes

04 **Name the sheet you want** the first tab is usually a cover page; fail loudly if missing

05 **Delete, don't hide** hidden and filtered rows still reach the parser

HEADERS

06 **No duplicate headings** two "Amount" columns: one silently overwrites the other

07 **No blank headings** unnamed columns become __EMPTY, __EMPTY_1...

08 **One header row, not two** flatten stacked headers into Q1_Revenue, Q1_Units

09 **Trim and normalise in code** "Region " and "Region" are different keys

DATA TYPES

10 **Numbers stored as text** the top cause of zeros in charts; coerce and check failures

11 **Symbols stay out of the cell** ₹1,20,000 is a string; format in the display layer

12 **Fix the percentage convention** 45% may hold 0.45 or 45; validate against a known total

13 **Dates in YYYY-MM-DD** serial numbers, timezones and 03/04 ambiguity all bite

14 **IDs are text, not numbers** 007 becomes 7 and the join quietly returns nothing

Red numbers on the left sheet match the rules that break. The right sheet is what the parser should receive: one table, one header row, raw values.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · WORKING WITH UNCLEAN EXCEL FILES · PART 2 OF 2

Totals are calculated, not inherited

The last six rules are about what sits inside the rows, and how the code reads them. **Wrong is more dangerous than broken.**

ROWS AND CONTENT

15 **No total or subtotal rows** a grand total charts as a bar twice the size of the rest

16 **No blank rows, no footnotes** "* excludes Nagpur" will become a data row

17 **Blank is not zero** "no sales" vs "we don't know"; state your choice on the dashboard

18 **Consistent category values** North, north and NORTH make three bars; normalise on read

19 **Never a pivot or formula sheet** #REF! and #N/A flow through as text; paste as values

PRACTICAL

20 **Reference columns by name** row[3] breaks silently the day someone inserts a column

\+ **Set a row limit** tell users the cap instead of letting the browser tab freeze

\+ **Check what is in the file** salary or customer data in unwatched columns has no place here

THE HABIT THAT PREVENTS MOST OF THIS

Almost every rule is a specific case of these six.

- ✓ One sheet
- ✓ One table
- ✓ Starting at A1
- ✓ One row of unique headers
- ✓ No merges, totals or footnotes
- ✓ No formatting doing the work of data

Build a clean extract sheet next to the working file.

Don't try to reform the operational spreadsheet: it has a human audience and should keep it.
Copy the values into a separate sheet built for code.

The parser gets its own copy.

HMC ONE

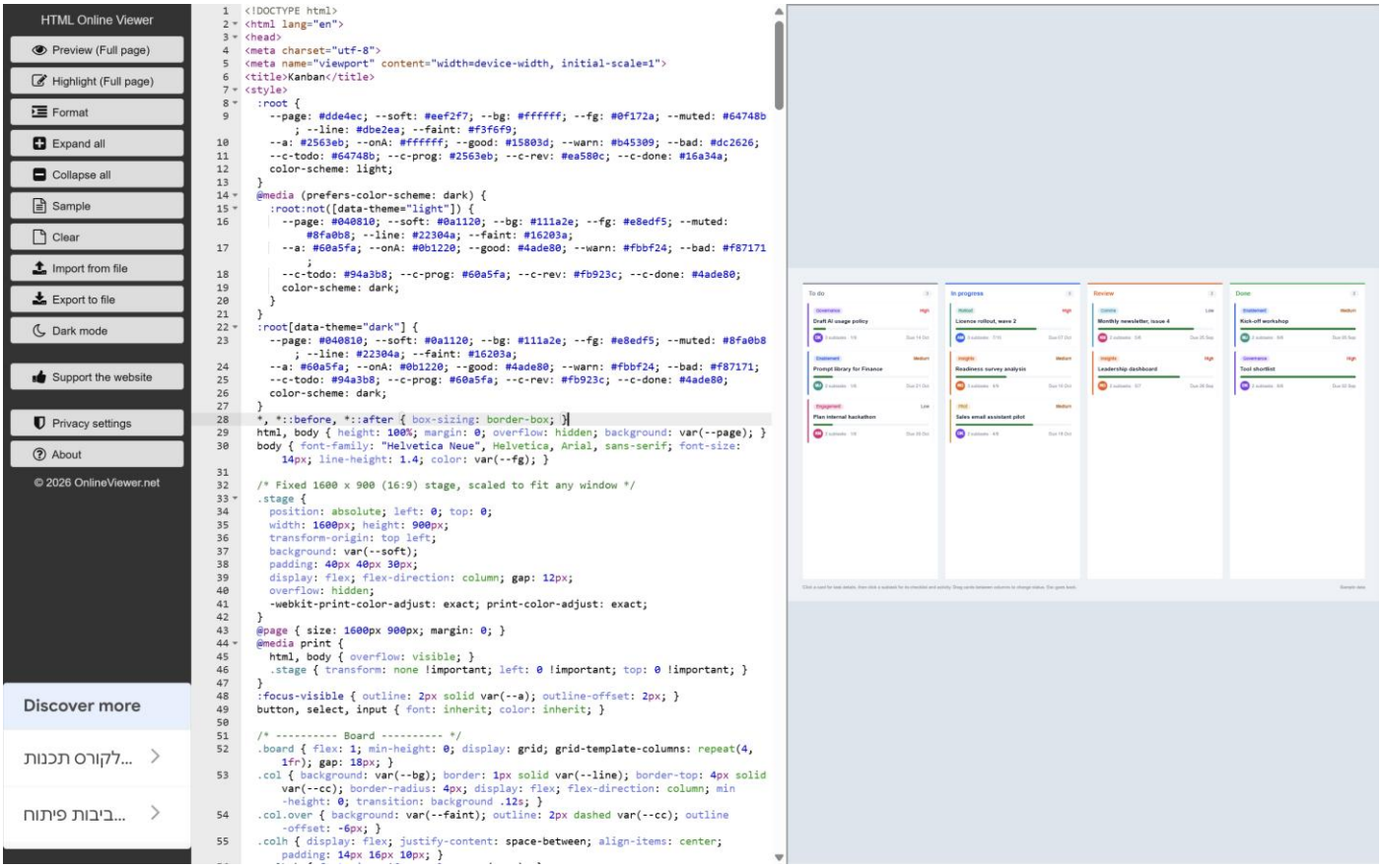
AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

Article – View HTML

https://html.onlineviewer.net/



Article – Vibe Coding Tips

Smart Tips for Editing HTML Applications Built with Vibe Coding

How to make precise changes, fix errors, and improve AI-generated HTML applications without breaking what already works.

One of the biggest advantages of vibe coding is that you can build sophisticated HTML applications, dashboards, simulators, and interactive tools without writing every line of code yourself.

But building the first version is often easier than modifying it.

You ask AI to change a button, and suddenly the entire layout changes. You fix one chart, and another stops working. You request a minor adjustment, and AI rewrites hundreds of lines of perfectly functioning code.

The solution is not necessarily better coding knowledge. It is **learning how to communicate changes more intelligently.**

Here are 12 practical techniques that can dramatically improve your workflow.

1. Share a Screenshot of the Exact Area That Needs Fixing

A screenshot is often more effective than a paragraph of explanation.

Instead of saying:

"There's something wrong with the dashboard. Please fix the layout."

Take a screenshot, crop it to the problematic section, and highlight the issue with an arrow or rectangle.

Better prompt:

"Look at the attached screenshot. The labels on the second chart overlap with the legend. Fix only this visual issue. Do not change the chart data, colours, or any other dashboard components."

Pro tip: Share two screenshots when necessary: one showing the complete page for context and another zoomed into the problematic section.

2. Tell AI What Must NOT Change

One of the most common problems with AI-assisted editing is unintended modifications.

A simple instruction can prevent considerable rework.

Example:

"Modify only the navigation menu. Preserve all existing JavaScript functionality, CSS styling, animations, charts, and other working components. Do not redesign the application."

This is particularly important for complex HTML applications containing interconnected components.

Think of every change as a surgical operation, not a complete reconstruction.

3. Describe the Expected Behaviour, Not Just the Problem

Saying "The filter isn't working" doesn't give AI enough information.

Instead, explain three things:

- What you are doing.
- What is currently happening.
- What should happen.

Better prompt:

"When I select 'North Region' from the dropdown, the revenue chart continues showing data for all regions. It should display only North Region sales. The other charts should continue responding to their existing filters."

This helps AI identify the actual functional problem rather than guessing.

4. Ask for a Diagnosis Before Requesting the Fix

AI sometimes jumps into changing code before understanding why something is broken.

A smarter approach is to separate diagnosis from implementation.

Example:

"Before modifying the HTML, identify the three most likely reasons why the table is not refreshing after importing a new Excel file. Explain which part of the JavaScript you would investigate. Do not make changes yet."

Once the issue is identified, ask AI to implement the smallest possible fix.

This technique is especially valuable when working with large applications.

5. Share Browser Console Errors

Screenshots are excellent for visual problems, but they cannot reveal every programming error.

When an HTML application fails, press **F12** in Chrome or Edge, open Developer Tools, and navigate to the Console tab.

Copy the relevant error message or share a screenshot of it.

Example:

"The Calculate button does nothing. The browser Console displays 'Uncaught TypeError: Cannot read properties of undefined'. Identify the source of the error and fix it without modifying the existing calculation methodology."

You can also check the Network tab when external files or APIs fail to load.

A console error often gives AI more actionable information than saying 'the page is broken'.

6. Make One Logical Change at a Time

Suppose you want to:

- Change dashboard colours.
- Add an Excel upload.
- Introduce a new calculation.
- Make the application mobile-friendly.

Avoid combining all four requests into one large instruction.

Instead, implement and test them separately.

This makes it easier to isolate errors, understand what changed, and reverse problematic modifications.

Pro tip: Group closely related changes together, but separate major functional, visual, and structural modifications.

7. Use Before-and-After Screenshots for Design Changes

Want your AI-generated dashboard to look like a professional enterprise application?

Show AI the current screenshot and a reference image of the desired appearance.

Example:

"Image 1 shows my existing HTML dashboard. Image 2 shows the visual style I want. Apply the typography, spacing, card styling, and colour hierarchy from Image 2 while preserving all calculations, filters, and interactive behaviour from Image 1."

This is much more precise than asking AI to "make it modern" or "make it McKinsey-style."

8. Ask AI to Identify the Relevant Code Section First

In applications with thousands of lines of HTML, CSS, and JavaScript, it may be unnecessary to modify the entire file.

Example:

"Identify the HTML element, CSS classes, and JavaScript functions responsible for the inventory reduction chart. Explain their dependencies before making any changes."

This technique helps distinguish three different types of modifications:

- HTML changes affect structure and content.
- CSS changes affect appearance and layout.
- JavaScript changes affect behaviour and calculations.

Understanding which layer needs modification can prevent unnecessary rewrites.

9. Preserve the Application's Data and Calculation Logic

This is particularly important for financial models, ROI simulators, operational dashboards, and business applications.

Suppose your HTML simulator calculates inventory savings using working capital and an 11% WACC.

You want to improve the visual presentation, but not modify the underlying financial assumptions.

Better prompt:

"Improve the visual layout of the benefits table and add tooltips explaining each metric. Do not change any formulas, financial assumptions, data dependencies, or calculation outputs. Verify that the same inputs produce the same results before and after the change."

A visually attractive application with incorrect calculations is worse than a basic application that works.

10. Save a Working Version Before Every Major Modification

AI-generated code can occasionally introduce regressions: fixing one element while breaking another.

Before making significant changes, save a working version.

For example:

- dashboard_v1_working.html
- dashboard_v2_new_filters.html
- dashboard_v3_mobile_layout.html

For more advanced projects, Git provides a more reliable way to track changes and restore earlier versions.

Pro tip: Whenever you reach a stable, fully functional version, treat it as a checkpoint.

11. Ask AI to Test Edge Cases, Not Just the Happy Path

An HTML application may work perfectly with sample data but fail when real users interact with it.

After making changes, ask AI to check scenarios such as:

- What happens when an Excel file contains blank cells?
- What if a user uploads an incorrect file format?
- What happens when a calculation involves division by zero?
- Does the dashboard remain usable on a smaller screen?
- Can dropdowns and buttons be operated using a keyboard?
- Do charts behave correctly when filters return no data?

Example:

"Review the updated application for edge cases involving missing data, invalid inputs, repeated button clicks, and empty datasets. Add appropriate validation without changing existing business logic."

Remember that AI's code review is not a substitute for actually testing the application in a browser.

12. Ask for a Change Summary and Regression Checklist

After every significant modification, request a concise summary.

Example:

"Summarise exactly which HTML elements, CSS rules, and JavaScript functions were modified. List any existing functionality that could be affected. Provide five browser tests I should perform to confirm that nothing else has broken."

This encourages controlled modifications and makes troubleshooting easier if problems appear later.

The Ultimate Prompt for Editing AI-Generated HTML

Here is a reusable prompt you can adapt for nearly any vibe-coded application.

Copy-Paste Prompt

"I have an existing HTML application that is functioning correctly except for the specific issue or enhancement described below.

Requested change: [Describe the change]

Current behaviour: [Describe what happens]

Expected behaviour: [Describe what should happen]

Visual reference: [Attach screenshot if applicable]

Please follow these instructions:

1. Inspect the relevant HTML, CSS, and JavaScript before making changes.

2. Identify the specific elements or functions responsible.
3. Make the smallest necessary modification.
4. Preserve all existing working functionality, business logic, formulas, data structures, animations, and styling unless explicitly instructed otherwise.
5. Avoid rewriting unrelated parts of the application.
6. Check potential dependencies and edge cases.
7. Summarise what changed and provide a brief regression-testing checklist.

If information is insufficient to diagnose the issue reliably, explain what additional screenshot, console error, or code section is needed rather than guessing."

Final Thought: Vibe Coding Is Easy. Controlled Iteration Is the Real Skill.

The first version of an HTML application is only the beginning.

The real challenge is making the tenth, twentieth, or fiftieth modification without compromising functionality.

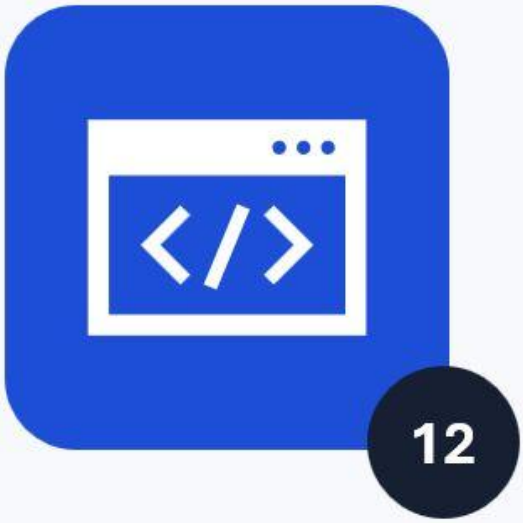
The most effective vibe coders do not simply keep asking AI to "fix this" or "make it better."

They provide precise visual references, describe expected behaviour, protect working components, manage versions, and validate each modification.

The quality of an AI-built application depends not just on how well you prompt it to build, but on how intelligently you prompt it to change.

That is what separates a quick AI-generated prototype from a reliable, reusable business application.

AI BUILDS · EDITING VIBE-CODED HTML



Vibe coding is easy. Editing it safely is the real skill.

12 techniques for changing AI-built HTML apps without breaking what already works.

12 TECHNIQUES

1 REUSABLE PROMPT

SWIPE →

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · EDITING VIBE-CODED HTML · PART 1 OF 4

Building the first version is easy.
Changing it safely is the skill.

AI-built dashboards, simulators and tools tend to break in the edit, not the build. The fix is not more coding knowledge; **it is how you ask for the change.**

WHAT GOES WRONG WHEN AI EDITS YOUR APP

The layout moves
You ask AI to change one button and the whole layout changes.

Something else breaks
You fix one chart and another one stops working.

Working code is rewritten
A minor adjustment rewrites hundreds of lines that worked.

12 TECHNIQUES, THREE HABITS

HABIT 1
Show the problem precisely

- 01 Screenshot the exact area
- 03 Describe expected behaviour
- 05 Share console errors
- 07 Use before-and-after images

HABIT 2
Scope the change

- 02 Say what must not change
- 04 Diagnose before fixing
- 06 One change at a time
- 08 Find the code section first
- 09 Lock the calculation logic

HABIT 3
Protect and verify

- 10 Save a working version
- 11 Test the edge cases
- 12 Ask for a change summary

Treat every change as surgery, not a rebuild.

Protect what works, describe behaviour precisely, and check each edit before making the next. The next three pages show how.

12 techniques and one reusable prompt.

HMC ONE
AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE
© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · EDITING VIBE-CODED HTML · PART 2 OF 4

Show AI the problem before you ask for the fix.

Most broken edits start with a vague request. Give AI the evidence, the boundaries and the expected result, **and it stops guessing.**

01 Screenshot the exact area

Crop to the problem and mark it. Add a full-page shot when context matters.

SAY THIS

"The labels on the second chart overlap the legend. Fix only this. Don't change the data, colours or other components."

02 Say what must not change

Unintended edits are the most common failure in apps with linked components.

SAY THIS

"Modify only the navigation menu. Preserve all JavaScript, CSS, animations and charts. Do not redesign the app."

03 Describe the expected behaviour

Give three things: what you did, what happens now, and what should happen.

SAY THIS

"When I select North Region, the revenue chart still shows all regions. It should show only North Region sales."

04 Diagnose before you fix

Separate diagnosis from implementation, then ask for the smallest possible fix.

SAY THIS

"List the three likeliest reasons the table doesn't refresh after an Excel import. Do not make changes yet."

05 Share the console error

Press F12, open Console and copy the error. Check Network when files or APIs fail.

SAY THIS

"Calculate does nothing. Console shows 'Uncaught TypeError'. Fix it without changing the calculation method."

06 Make one logical change at a time

New colours, an Excel upload, a calculation and a mobile layout are four requests, not one.

RULE OF THUMB

Group closely related edits. Keep visual, functional and structural changes separate, and test each one.

A console error tells AI more than "the page is broken".

Evidence first, fix second.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · EDITING VIBE-CODED HTML · PART 3 OF 4

Protect what works, then prove nothing broke.

Fixing one element while breaking another is the usual regression. Lock the logic, checkpoint the file and **test more than the happy path.**

07 Use before-and-after images

Show the current screen and a reference image of the look you want, not "make it modern".

SAY THIS

"Apply Image 2's typography, spacing and card styling. Keep every calculation, filter and interaction from Image 1."

08 Find the code section first

HTML is structure, CSS is appearance, JavaScript is behaviour. Name the layer that needs to change.

SAY THIS

"Identify the HTML, CSS and JavaScript behind the inventory chart. Explain their dependencies before changing anything."

09 Lock the calculation logic

A good-looking app with wrong numbers is worse than a plain one that works.

SAY THIS

"Restyle the benefits table and add tooltips. Don't change formulas or assumptions. Same inputs must give the same results."

10 Save a working version

Treat every stable build as a checkpoint. Use Git for larger projects.

NAME IT LIKE THIS

dashboard_v1_working.html
dashboard_v2_new_filters.html
dashboard_v3_mobile_layout.html

11 Test the edge cases

Blank cells, wrong file types, division by zero, empty filters, small screens, keyboard use.

SAY THIS

"Review for missing data, invalid inputs, repeated clicks and empty datasets. Add validation without changing business logic."

12 Ask for a change summary

Know exactly what changed and what it could affect, so the next problem is easy to trace.

SAY THIS

"List the HTML elements, CSS rules and JavaScript functions you changed, and give me five browser tests to run."

AI's code review is not a substitute for testing in a browser.

Open it, click it, break it, then ship it.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · EDITING VIBE-CODED HTML · PART 4 OF 4

One prompt for every edit you ask AI to make.

Copy it, fill in the four fields and attach the screenshot or console error. **It builds the 12 techniques into one request.**

COPY-PASTE PROMPT

I have an existing HTML application that is functioning correctly except for the specific issue or enhancement described below.

Requested change	[Describe the change]
Current behaviour	[Describe what happens]
Expected behaviour	[Describe what should happen]
Visual reference	[Attach a screenshot if applicable]

Please follow these instructions:

1. Inspect the relevant HTML, CSS and JavaScript before making changes.
2. Identify the specific elements or functions responsible.
3. Make the smallest necessary modification.
4. Preserve all working functionality, business logic, formulas, data, animations and styling.
5. Avoid rewriting unrelated parts of the application.
6. Check potential dependencies and edge cases.
7. Summarise what changed and give a short regression-testing checklist.
8. If information is insufficient, say what screenshot, console error or code is needed rather than guessing.

Vibe coding is easy. Controlled iteration is the real skill.

The first version is only the beginning. The tenth, twentieth or fiftieth change is where a quick prototype becomes a reliable business application.

Prompt the change as carefully as the build.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

Article – Reverse Engineer Prompt from HTML

You don't need to read code to own it

Every HTML file is a complete, self-describing app. The layout, the styling, the logic and the libraries it borrows all sit in one text file. That makes it the perfect thing to hand to an AI and ask: what is this, how does it work, and how do I make it mine?

This is the reverse direction of vibe coding. Instead of prompting from a blank page, you start from something that already works — a dashboard a colleague built, a calculator from a tutorial, a page you generated last month and forgot — and let the AI walk you through it before you change a single line.

Why use it

Most people stall on vibe coding not at creation but at the second step: changing something they didn't write. Uploading the file first gives you a map before you edit.

- **Reuse instead of rebuild.** A working page is 80% of the job. Adapting it is faster and safer than prompting from scratch.
- **Know what you're running.** The AI flags every external script, font and data call, so you know what loads from the internet and what stays on your machine.
- **Edit with precision.** Once you know the page has a "categories array" or a "chart config", your change prompts stop being vague ("make it better") and become surgical ("add a Travel category in purple").
- **Learn by dissection.** For non-developers, a plain-English walkthrough of a real file teaches more than a tutorial.
- **Hand-offs get easier.** Inherited a tool from a team member who left? Ten minutes with the AI replaces a knowledge-transfer meeting.

How it works

An HTML file has three layers, and the AI reads all three at once.

Layer	Where it lives	What the AI extracts
Structure (HTML)	<body> tags: headings, forms, buttons, tables	What the user sees and can click
Style (CSS)	<style> block or linked stylesheets	Colours, fonts, layout, mobile behaviour

Logic (JavaScript)	<script> blocks	What happens on click, how numbers are calculated, where data is stored
Dependencies	<script src> and <link href> pointing to URLs	Libraries (Chart.js, Tailwind), fonts, APIs the page calls

The workflow is three steps:

1. **Upload** the .html file to Claude (drag it into the chat, or paste the code).
2. **Extract** using the default prompt below. You get a structured brief of the page.
3. **Modify** with targeted follow-up prompts that reference what the brief told you.

The default extraction prompt

Copy this, attach your HTML file, and send. It works for any page, from a 30-line calculator to a 2,000-line dashboard.

Prompt:

I've attached an HTML file. Act as a senior front-end developer explaining it to a non-technical business user. Do not change any code yet.

Give me:

1. WHAT IT DOES — In 2–3 sentences, what is this page for and who would use it?

2. WHAT THE USER SEES — List each visible section, input, button and output, top to bottom.

3. HOW IT WORKS — Walk through what happens when the user interacts with it (clicks, types, submits). Name the JavaScript functions involved in plain English.

4. WHAT IT USES — List every external dependency: libraries, CDNs, fonts, APIs, and any browser storage (localStorage, cookies). For each, say what it's used for and whether the page breaks offline.

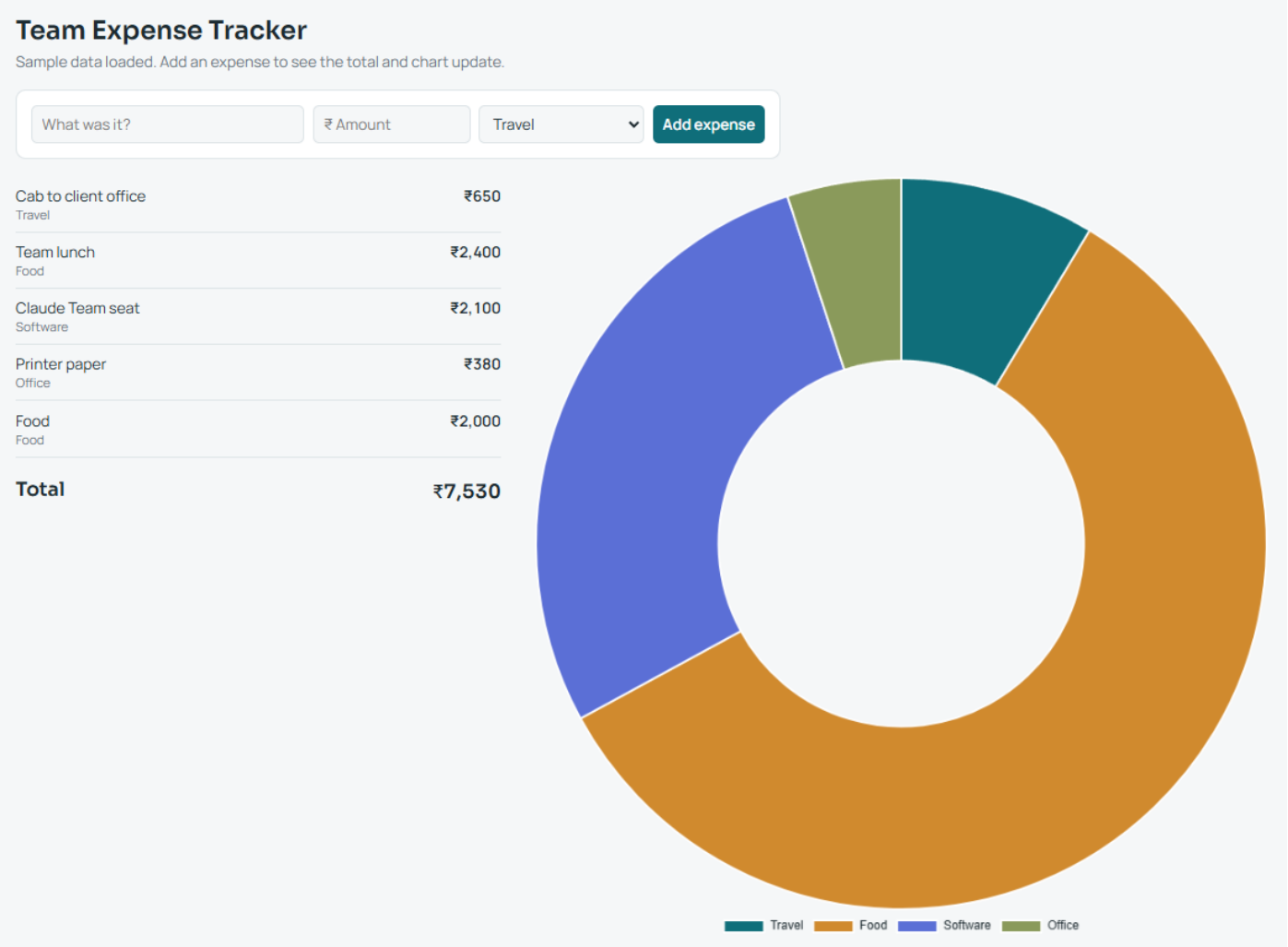
5. DATA — Where does the data come from (hard-coded, user input, file, API) and where does it go? Is anything saved?

6. CUSTOMISATION POINTS — The 5–8 easiest places to change it: labels, colours, default values, lists/arrays, formulas, chart settings. For each, quote the exact line(s) to edit.

7. RISKS & LIMITS — Anything fragile, insecure, or that won't scale (e.g., data lost on refresh, hard-coded keys, no validation).

End with 3 suggested modifications a business user might want, phrased as prompts I can send you next.

Section 6 is the one that matters most. It turns an unfamiliar file into a short list of dials you can turn.



What the AI returns (abridged)

What it does. A personal or team expense log. Users record spends, see a running total in rupees, and view the split by category as a doughnut chart.

What the user sees. A title; a form with description, amount, category dropdown and an "Add expense" button; a list of expenses with amounts; a total; a doughnut chart.

How it works. On submit, the form handler stops the page reload, adds the new row to the expenses array and calls render(). render() rebuilds the list, recomputes the total, groups amounts by category and redraws the chart.

What it uses.

Dependency	Purpose	Offline?
Chart.js 4.4.1 (cdnjs)	Draws the doughnut chart	Chart breaks offline
Google Fonts: Sora, Manrope	Heading and body type	Falls back to system font

localStorage	Saves expenses in this browser	Works offline; not shared across devices

Data. Starts from four hard-coded sample rows. New rows come from the form and are saved to the browser's localStorage. Nothing leaves the device.

Customisation points.

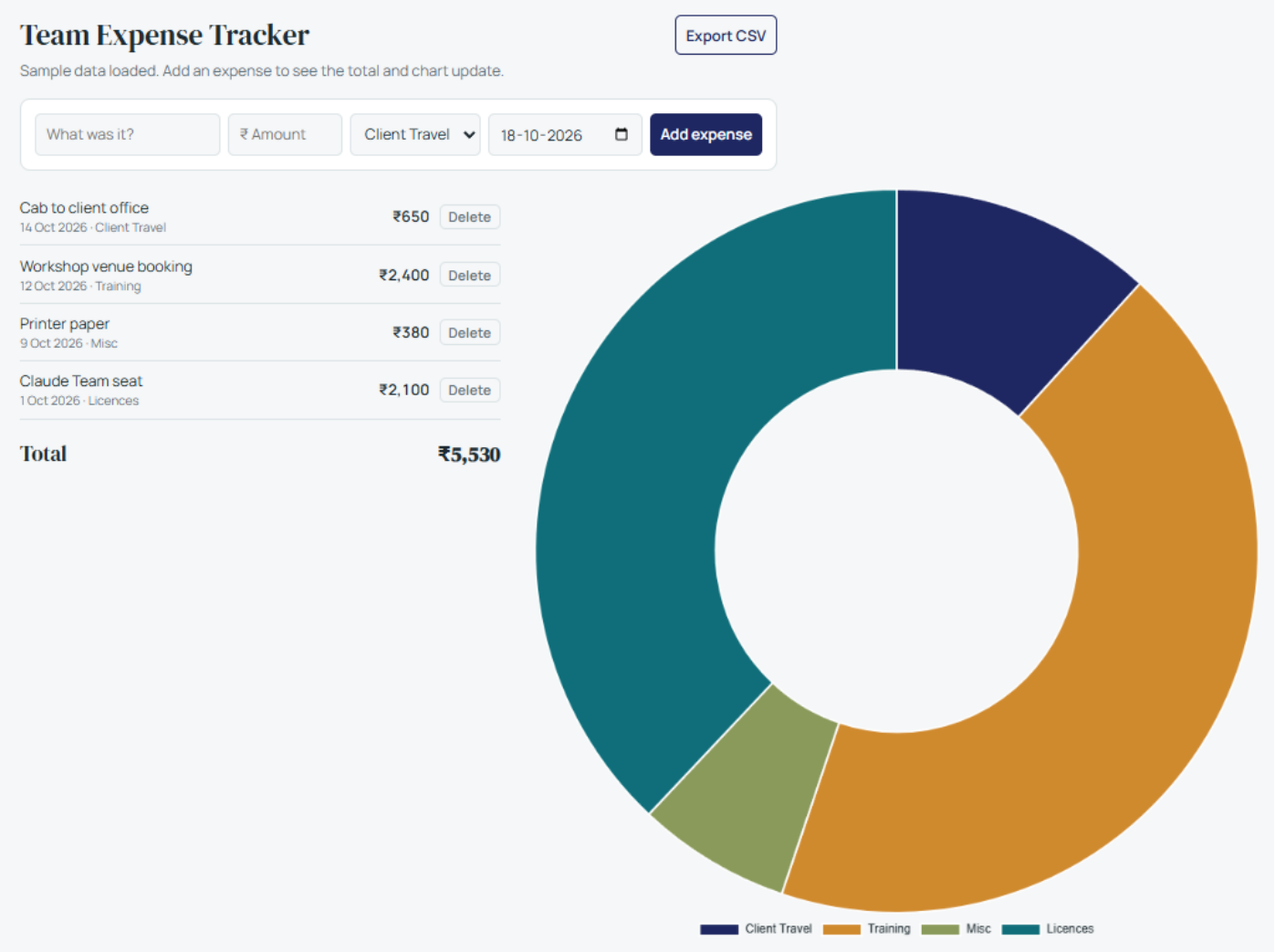
Change	Where to edit
Categories	The <option> tags in <select id="category"> and the matching keys in COLORS
Category colours	const COLORS = { Travel: "#0f6e7a", ... }
Currency	const rupee = n => "₹" + n.toLocaleString("en-IN")
Sample data	The let expenses = [...] array
Brand colour	--accent in :root
Chart type	type: "doughnut" (try "bar" or "pie")

Risks and limits. Data lives in one browser only and is lost if site data is cleared. No date field, so no monthly view. Descriptions are inserted as raw HTML, so pasted tags would render. No delete or edit button.

Modifying it to fit your needs

The extraction gives you the vocabulary. Now each change prompt can name the exact part to touch. Good modification prompts follow one pattern: **what to change + where it lives + what must stay the same.**

Goal	Prompt to send
Rebrand	"Change --accent to our navy #1E2761 and the heading font to DM Serif Display. Keep everything else."
Fit your categories	"Replace the four categories with Client Travel, Training, Licences, Misc. Update the dropdown and COLORS together."
Add a field	"Add a date input to the form, show the date in each list row, and sort the list newest first."
Add a control	"Add a delete button on each row that removes it and re-renders the total and chart."
Fix a risk	"Escape the description text before inserting it into the list, so pasted HTML shows as plain text."
Get data out	"Add an 'Export CSV' button that downloads all expenses."



Three habits keep edits safe:

- **One change per prompt.** Test the page after each. If something breaks, you know which prompt caused it.
- **Ask for the full file back.** "Return the complete updated file" avoids pasting fragments into the wrong place.
- **Re-run the extraction after big changes.** A fresh brief catches new dependencies or risks the edits introduced.

The takeaway

You don't need to understand code to adapt it. You need a reliable way to get a map of it. Upload, extract, then modify: three steps that turn any working HTML file into a starting point you own.

Article – Publish on GitHub

Publishing your HTML on GitHub

You have built something. A dashboard, a calculator, a one-page explainer. It sits in your Downloads folder as a single .html file, and it works perfectly when you double-click it.

Then someone asks you to send it, and the problem starts. Email strips it or flags it. WhatsApp won't take it. You put it on Drive and the recipient gets a download prompt and a file their browser refuses to open properly. The thing you made is trapped on your own machine.

What you need is a link. GitHub Pages gives you one, free, in about five minutes.

What GitHub Pages actually is

GitHub is where the world's code lives. Pages is a feature buried in its settings that says: take the files in this folder and serve them as a website. No server to rent, no hosting bill, no deployment pipeline. You put an HTML file in a folder, flip a switch, and it has a public address.

This works because the files you have been vibe coding are *static* — everything happens inside the visitor's browser. There is no database, nothing running on a server. That is precisely the kind of thing Pages was built to host.

The five-minute version

1. Make an account at github.com. Free is enough.

2. Create a repository. Top-right **+** → **New repository**. A repository — "repo" — is just a folder. Name it something short and lowercase, like sales-dashboard. Set it to **Public**. Tick **Add a README file**. Create.

On a free account, Pages only works on public repositories. Anyone who finds the URL can see your page, and anyone on GitHub can read the file. Paid tiers (Pro, Team) allow private repos with Pages.

3. Upload your file. On the repo page, **Add file** → **Upload files**. Drag your HTML file in. Scroll down, click **Commit changes**. No command line involved.

4. Turn Pages on. Go to the **Settings** tab → **Pages** in the left sidebar (under "Code and automation"). Under **Build and deployment**, set Source to **Deploy from a branch**, pick branch **main** and folder **/ (root)**, and Save.

5. Wait a minute, then refresh. A green banner appears with your URL. It will look like:

https://yourname.github.io/sales-dashboard/

GitHub says publishing can take up to ten minutes. It is usually under one. Send the link.

The one rule that trips everyone up

Name your file index.html.

index.html is what a web server looks for when someone visits a folder. If your file is called dashboard_final_v3.html, the bare URL shows an error or an ugly file listing, and the actual page only loads at .../dashboard_final_v3.html. Rename it before you upload and this problem disappears.

Two related gotchas: web servers are **case-sensitive** in a way Windows is not, so Index.html is a different file from index.html. And **avoid spaces** in file names — they turn into %20 in the URL.

Updating it

Upload the new version of index.html over the old one. Same URL, new content, live in about a minute.

If you don't see your change, your browser has cached the old page. Hard refresh with **Ctrl+Shift+R** (Cmd+Shift+R on Mac) before you assume something broke.

Before you click publish

A public repo is genuinely public, and "anyone with the link" includes search engines.

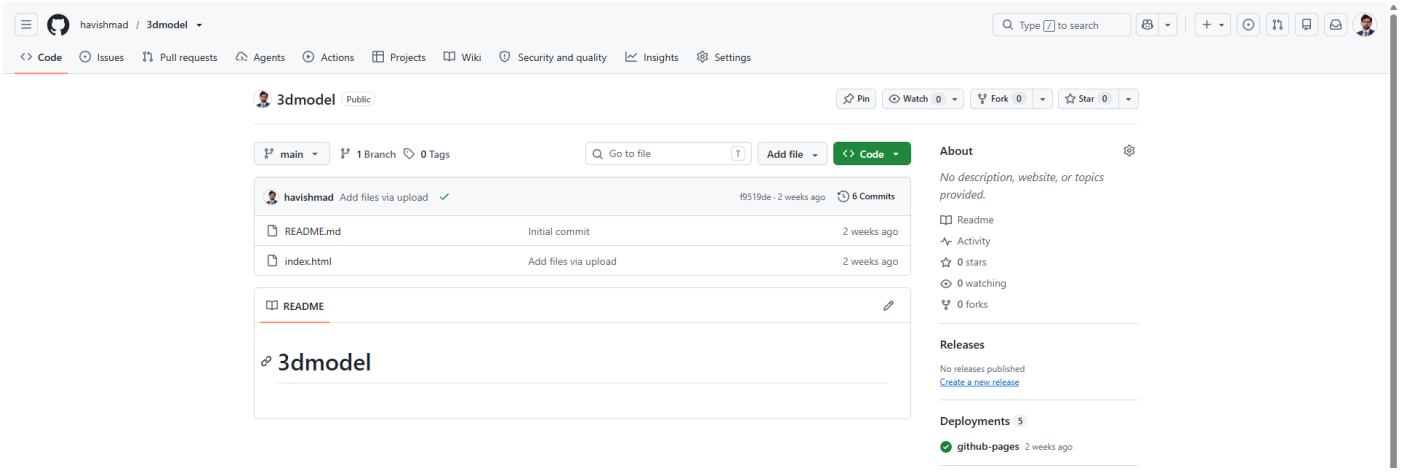
Open your HTML file and look at the data you hardcoded into it. If your demo dashboard contains a client's actual sales numbers, real customer names, or internal margins, that is now on the open internet with your name attached. Replace it with dummy data first. Same for API keys — a key pasted into client-side HTML is visible to anyone who views the source, and will be scraped within hours.

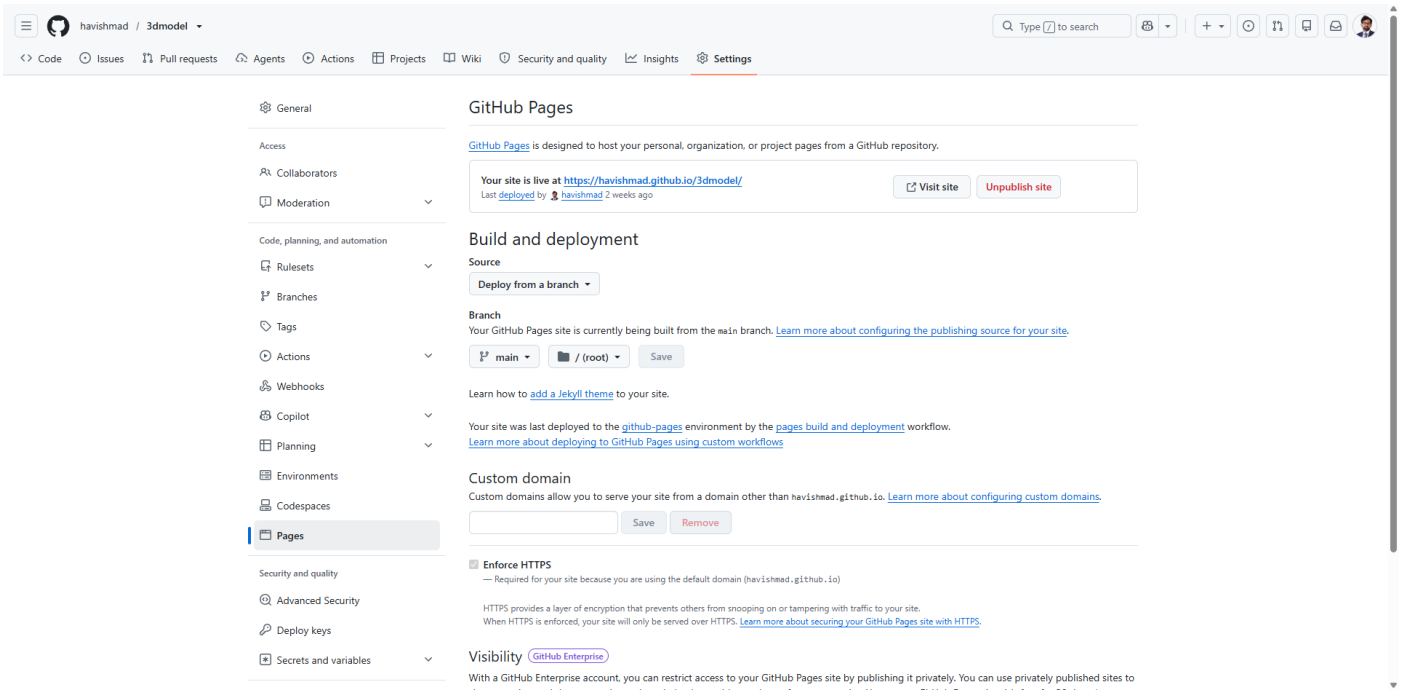
What this gives you, and what it doesn't

You get a real, permanent, shareable URL that works on any device and opens instantly — good enough for a client demo, a handbook example, a tool your team uses daily. You can point your own domain at it later if you want.

You don't get anything that needs a server: no login, no saving data between visits, no hiding anything behind a password. Those need a different class of tool.

For 90% of what you will build in this handbook, Pages is the whole answer.


havishmad / 3dmodel
Type / to search
Code
Issues
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Insights
Settings
3dmodel
Public
Pin
Watch 0
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
havishmad Add files via upload
f9519de · 2 weeks ago
6 Commits
README.md
Initial commit
2 weeks ago
index.html
Add files via upload
2 weeks ago
README
3dmodel
About
No description, website, or topics provided.
Readme
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Deployments 5
github-pages 2 weeks ago

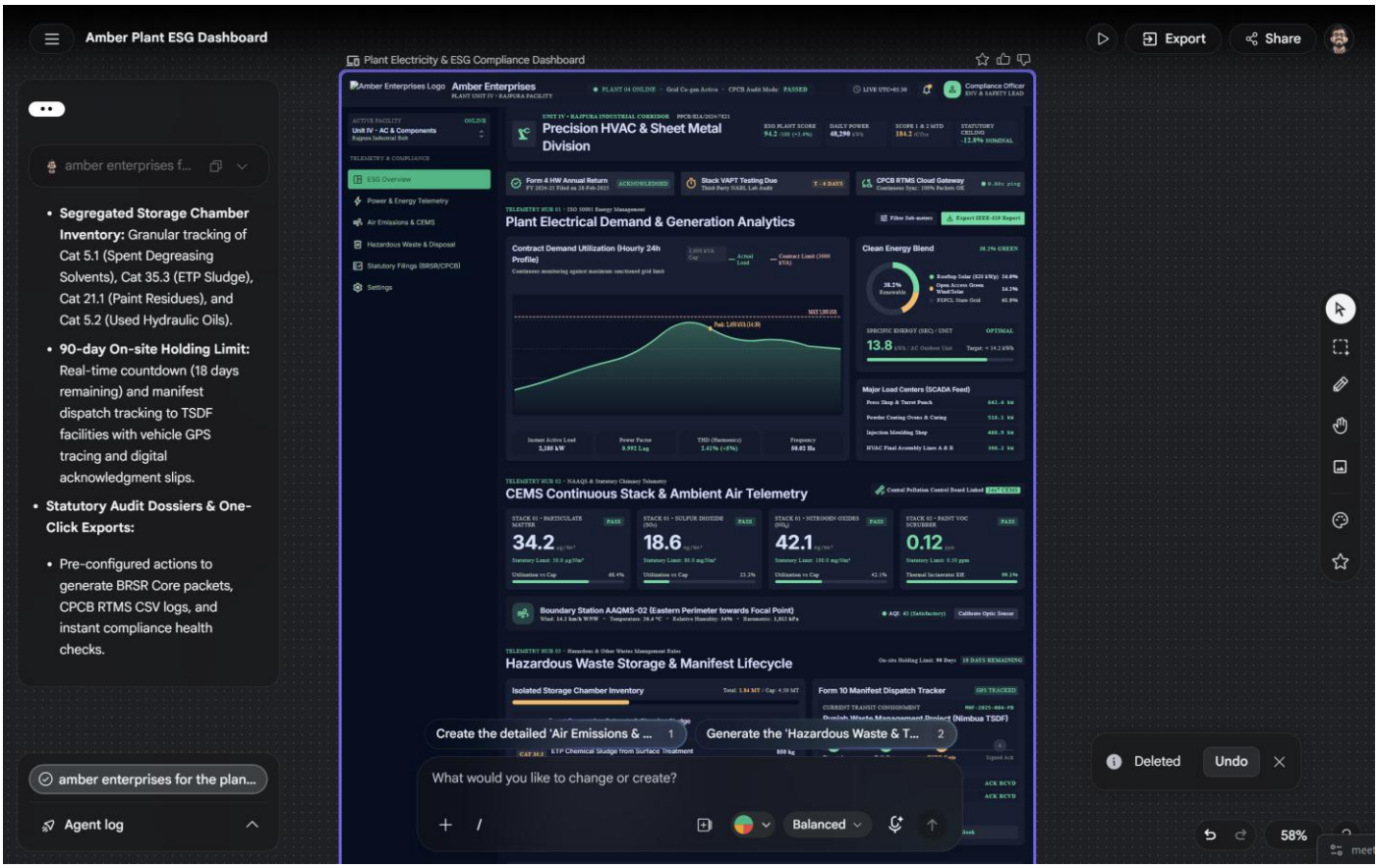

havishmad / 3dmodel
Type / to search
Code
Issues
Pull requests
Agents
Actions
Projects
Wiki
Security and quality
Insights
Settings
General
Access
Collaborators
Moderation
Code, planning, and automation
Rulesets
Branches
Tags
Actions
Webhooks
Copilot
Planning
Environments
Codespaces
Pages
Security and quality
Advanced Security
Deploy keys
Secrets and variables
GitHub Pages
GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.
Your site is live at https://havishmad.github.io/3dmodel/
Last deployed by havishmad 2 weeks ago
Visit site
Unpublish site
Build and deployment
Source
Deploy from a branch
Branch
Your GitHub Pages site is currently being built from the main branch. Learn more about configuring the publishing source for your site.
main
/ (root)
Save
Learn how to add a Jekyll theme to your site.
Your site was last deployed to the github-pages environment by the pages build and deployment workflow.
Learn more about deploying to GitHub Pages using custom workflows
Custom domain
Custom domains allow you to serve your site from a domain other than havishmad.github.io. Learn more about configuring custom domains.
Save
Remove
Enforce HTTPS
— Required for your site because you are using the default domain (havishmad.github.io)
HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site.
When HTTPS is enforced, your site will only be served over HTTPS. Learn more about securing your GitHub Pages site with HTTPS.
Visibility
GitHub Enterprise
With a GitHub Enterprise account, you can restrict access to your GitHub Pages site by publishing it privately. You can use privately published sites to

Part — Dashboard and Reports

Article – Create Prototypes

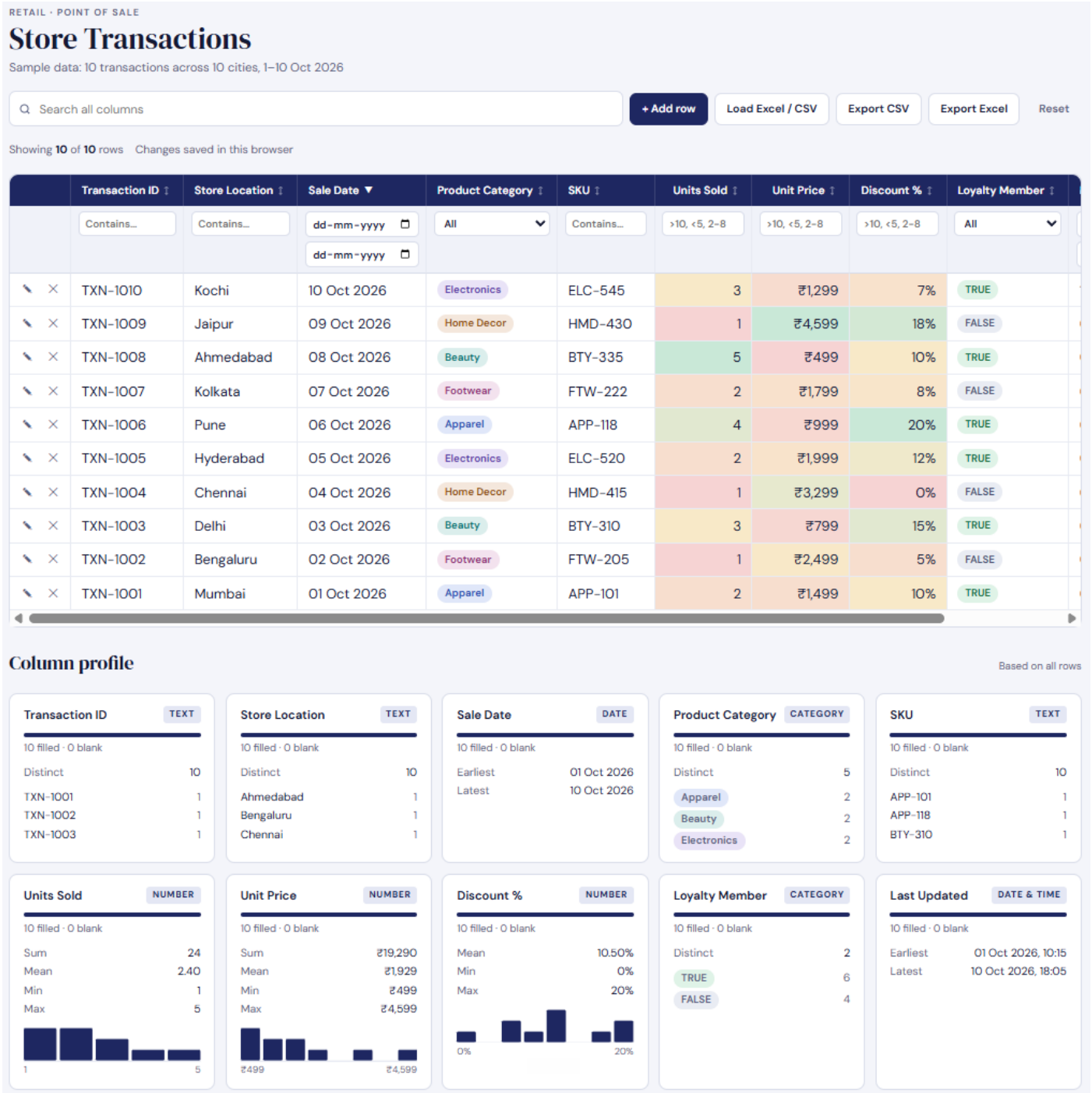
Google Stitch

https://stitch.withgoogle.com/



Article – Editable Table from Excel

Transaction ID	Store Location	Sale Date	Product Category	SKU	Units Sold	Unit Price	Discount %	Loyalty Member	Last Updated
TXN-1001	Mumbai	01-Oct-2026	Apparel	APP-101	2	₹1,499.00	10%	TRUE	01-Oct-2026 10:15
TXN-1002	Bengaluru	02-Oct-2026	Footwear	FTW-205	1	₹2,499.00	5%	FALSE	02-Oct-2026 11:30
TXN-1003	Delhi	03-Oct-2026	Beauty	BTY-310	3	₹799.00	15%	TRUE	03-Oct-2026 14:05
TXN-1004	Chennai	04-Oct-2026	Home Decor	HMD-415	1	₹3,299.00	0%	FALSE	04-Oct-2026 16:45
TXN-1005	Hyderabad	05-Oct-2026	Electronics	ELC-520	2	₹1,999.00	12%	TRUE	05-Oct-2026 09:20
TXN-1006	Pune	06-Oct-2026	Apparel	APP-118	4	₹999.00	20%	TRUE	06-Oct-2026 13:10
TXN-1007	Kolkata	07-Oct-2026	Footwear	FTW-222	2	₹1,799.00	8%	FALSE	07-Oct-2026 15:55
TXN-1008	Ahmedabad	08-Oct-2026	Beauty	BTY-335	5	₹499.00	10%	TRUE	08-Oct-2026 12:40
TXN-1009	Jaipur	09-Oct-2026	Home Decor	HMD-430	1	₹4,599.00	18%	FALSE	09-Oct-2026 17:25
TXN-1010	Kochi	10-Oct-2026	Electronics	ELC-545	3	₹1,299.00	7%	TRUE	10-Oct-2026 18:05



Prompt:

Turn the attached table (Excel file, CSV or screenshot) into a single self-contained HTML page with an interactive table.

DATA

- Read every row and column exactly as given. Keep the original column names and order. If it's a screenshot, transcribe it carefully and tell me any cell you couldn't read.
- Detect each column's type: number, date, category (6 or fewer repeating values) or text. Store dates as YYYY-MM-DD. Strip ₹, commas and % when storing numbers, but show them again on screen.
- Put the data in a JS array at the top of the script so I can swap it later.

FEATURES

1. Date picker: date columns use a calendar input when editing, and a from–to date filter.
2. Conditional colours: red-to-green tint on number columns (higher = better unless I say otherwise). Coloured pills for category columns (Paid/Done = green, Pending = amber, Overdue/Failed = red).
3. Sort: click a header to cycle ascending → descending → off. Blanks always sort last.
4. Filter:
    - a global search box
    - a filter under each header: dropdown for categories, "contains" for text, >, <, or a–b range for numbers, from–to for dates
    - "Showing X of Y rows" and a Clear filters link
5. Column profile: one card per column showing type and filled/blank counts. For numbers: sum, mean, min, max and a mini histogram. For dates: earliest and latest. For text: distinct count and top 3 values. Recalculate on the filtered rows.
6. Add, edit, delete:
    - "+ Add row" button
    - ✎ edits a row inline with inputs that match the column type (Enter saves, Esc cancels)
    - ✕ deletes a row, with an Undo toast
7. Export: "Export CSV" and "Export Excel" (.xlsx via SheetJS) of the currently filtered and sorted rows, with real numbers and dates in Excel.
8. Load file: a "Load Excel / CSV" button that replaces the data from any single-sheet file and re-detects column types.

DESIGN

- Clean, professional look: [brand colour, e.g. navy #1E2761], [heading font] for the title, [body font] for the rest. Light and dark mode.

- Sticky header. Numbers right-aligned in tabular figures. Dates shown as 04 Jul 2026. Indian number format (₹1,84,500).
- Works on a phone: the table scrolls sideways inside its card, with no page-level horizontal scroll.

TECHNICAL

- One HTML file. The only external script is SheetJS from cdnjs.
- Escape all cell text before inserting it into the page.
- Save edits in localStorage, wrapped in try/catch, with a "Reset" option.

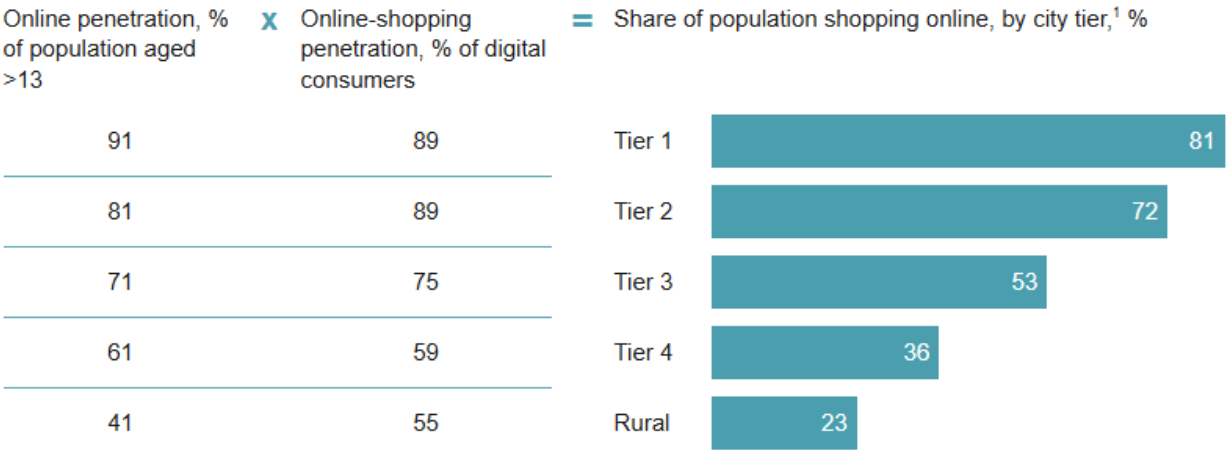
Article – Editable HTML from Image

Prompt:

Recreate this perfectly using html. Page must be editable.



In China, e-commerce penetration in top-tier cities has begun to reach saturation levels.



[1]China's cities are categorized by tiers based on GDP; tiers 1 and 2 are more urban and developed, while the lowest tiers are more rural.

McKinsey&Company

Article – Create Dashboard from own Excel

Build me a single, self-contained HTML file that works as an Excel dashboard. When I open it in a browser, I upload my Excel file, and the dashboard builds itself from the data. Nothing should be sent to a server; all processing happens in the browser.

TECH:

- One .html file with HTML, CSS and JavaScript inline
- Use SheetJS (xlsx) from cdnjs to read the Excel file
- Use Chart.js from cdnjs for charts
- No frameworks, no build step

EXCEL FILE STRUCTURE (what the uploaded file will contain):

Sheet "Raw_Data": headers in row 1, data from row 2

Header	**Type**	**Notes**
Order_ID	Text	Unique
Order_Date	Date	May arrive as an Excel serial number; convert it
Region	Text	North / South / East / West
Product_Cat	Text	6 categories
Sales_Rep	Text	~25 reps
Units	Number	
Revenue	Number (INR)	
Cost	Number (INR)	
Status	Text	Delivered / Pending / Returned

Sheet "Targets": headers in row 3, starting at column B
| Region | Month (1st of month) | Target_Revenue |

BUSINESS RULES:

- Financial year is April to March
- Margin % = (Revenue - Cost) / Revenue
- Exclude "Returned" orders from revenue, cost and margin
- Show money in Indian format (₹12,34,567) and KPIs in lakhs

UPLOAD SCREEN:

- A centred card with a drag-and-drop zone and a "Browse" button
- Accept .xlsx and .xls only
- After upload, check that both sheets and all required columns exist. If anything is missing, show a clear message naming exactly what's missing. Don't fail silently.
- Show the file name and row count once the file loads, plus a "Load another file" button

DASHBOARD LAYOUT (shown after a successful upload):

- Header bar: "Sales Performance Dashboard – FY26", navy background
- Filter bar under the header: dropdowns for Region, Product_Cat and Status (each with "All"), a date range picker, and a "Reset" button
- Row 1: four KPI cards
    1. Total Revenue (₹ lakhs)
    2. Total Units
    3. Margin %
    4. Revenue vs Target (% achieved, green if ≥100%, red if below)
- Row 2: Monthly revenue trend (line, with target as a dashed line) | Revenue by region (column, data labels on)
- Row 3: Category mix (doughnut) | Top 10 sales reps (horizontal bar, sorted descending)
- Row 4: Detail table of filtered orders, sortable by clicking headers, with a search box, 25 rows per page, and "Export to CSV"

INTERACTIVITY:

- Every KPI, chart and the table updates instantly when any filter changes
- Clicking a bar in "Revenue by region" sets the Region filter

- Charts show tooltips with formatted values

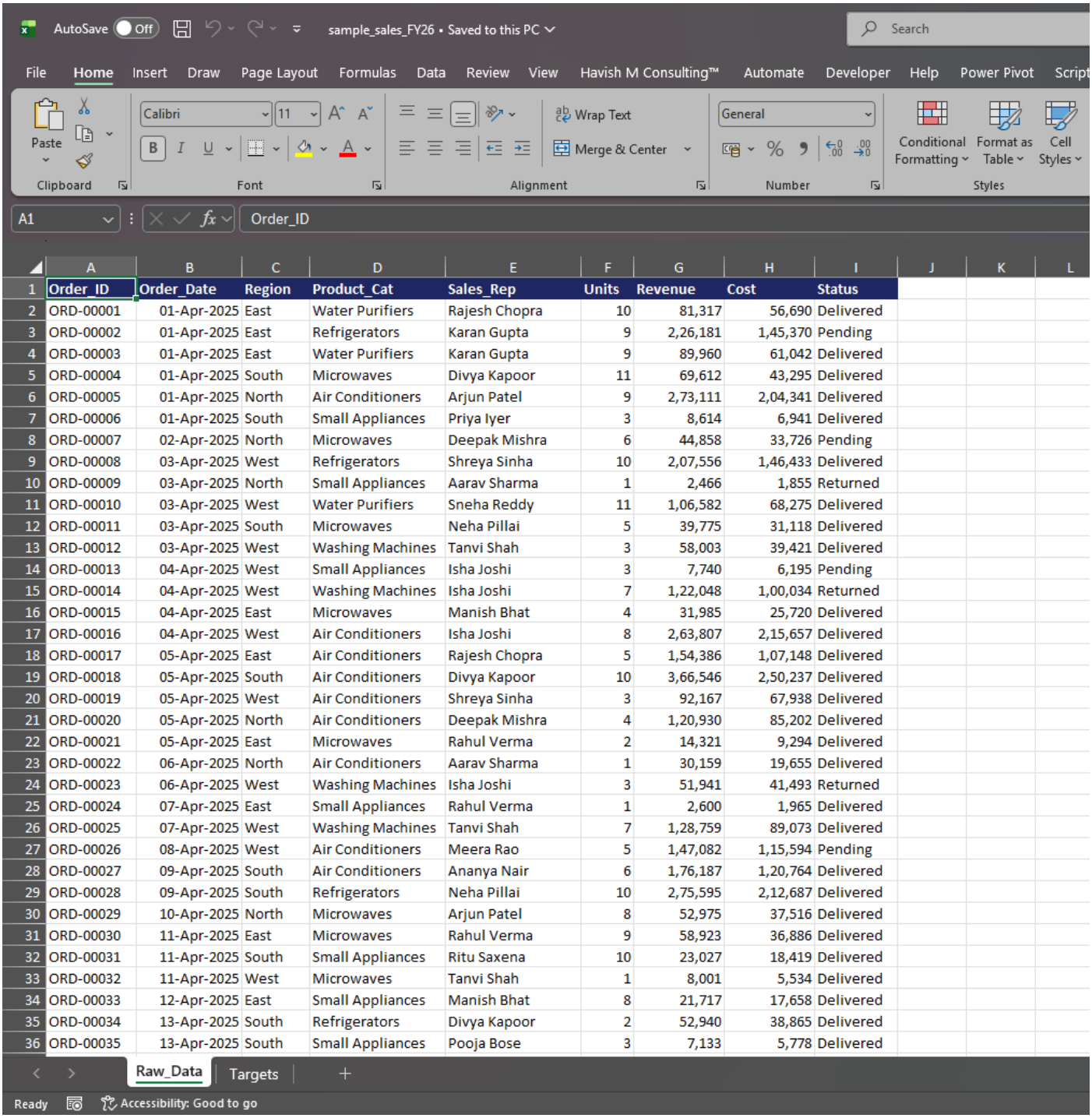
DESIGN:

- Primary #1E2761 (navy), accent #F28C28 (orange), background #F5F6FA
- Cards: white, 8px radius, soft shadow
- Doughnut colours: navy, orange, then navy tints
- Font: Inter from Google Fonts, falling back to system sans-serif
- Responsive: 2-column grid on desktop, single column on mobile
- No 3D effects, no clutter

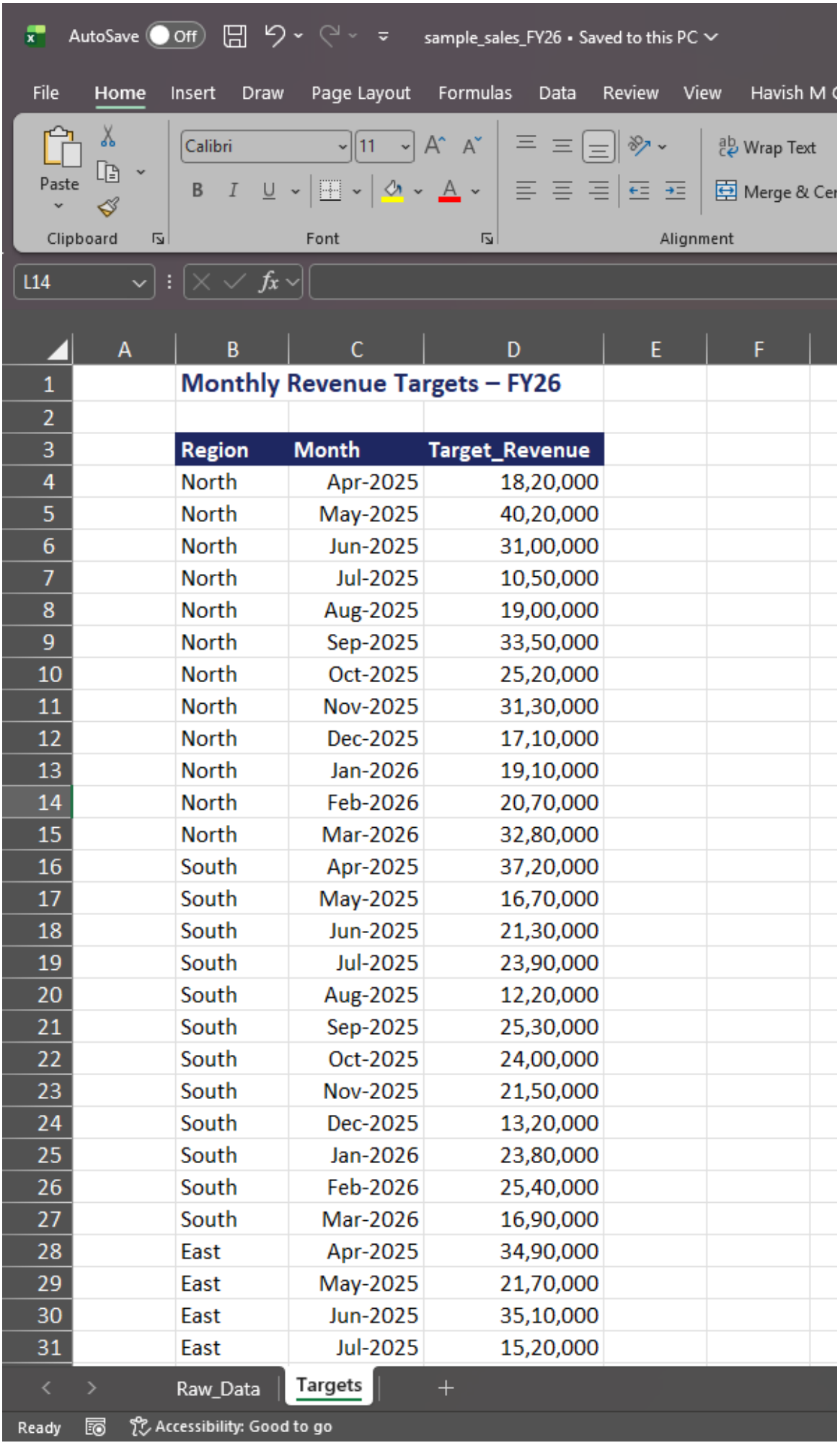
CODE QUALITY:

- Keep data loading, filtering, KPI calculation and chart rendering in separate, clearly commented functions
- Destroy and redraw charts on filter change (no memory leaks)

Before writing the code, restate the data structure and business rules in 5 lines and list any assumptions. Then give me the complete file. After the code, tell me how to test it, and generate a small sample Excel structure I can create to try it out.



Order_ID	Order_Date	Region	Product_Cat	Sales_Rep	Units	Revenue	Cost	Status
ORD-00001	01-Apr-2025	East	Water Purifiers	Rajesh Chopra	10	81,317	56,690	Delivered
ORD-00002	01-Apr-2025	East	Refrigerators	Karan Gupta	9	2,26,181	1,45,370	Pending
ORD-00003	01-Apr-2025	East	Water Purifiers	Karan Gupta	9	89,960	61,042	Delivered
ORD-00004	01-Apr-2025	South	Microwaves	Divya Kapoor	11	69,612	43,295	Delivered
ORD-00005	01-Apr-2025	North	Air Conditioners	Arjun Patel	9	2,73,111	2,04,341	Delivered
ORD-00006	01-Apr-2025	South	Small Appliances	Priya Iyer	3	8,614	6,941	Delivered
ORD-00007	02-Apr-2025	North	Microwaves	Deepak Mishra	6	44,858	33,726	Pending
ORD-00008	03-Apr-2025	West	Refrigerators	Shreya Sinha	10	2,07,556	1,46,433	Delivered
ORD-00009	03-Apr-2025	North	Small Appliances	Aarav Sharma	1	2,466	1,855	Returned
ORD-00010	03-Apr-2025	West	Water Purifiers	Sneha Reddy	11	1,06,582	68,275	Delivered
ORD-00011	03-Apr-2025	South	Microwaves	Neha Pillai	5	39,775	31,118	Delivered
ORD-00012	03-Apr-2025	West	Washing Machines	Tanvi Shah	3	58,003	39,421	Delivered
ORD-00013	04-Apr-2025	West	Small Appliances	Isha Joshi	3	7,740	6,195	Pending
ORD-00014	04-Apr-2025	West	Washing Machines	Isha Joshi	7	1,22,048	1,00,034	Returned
ORD-00015	04-Apr-2025	East	Microwaves	Manish Bhat	4	31,985	25,720	Delivered
ORD-00016	04-Apr-2025	West	Air Conditioners	Isha Joshi	8	2,63,807	2,15,657	Delivered
ORD-00017	05-Apr-2025	East	Air Conditioners	Rajesh Chopra	5	1,54,386	1,07,148	Delivered
ORD-00018	05-Apr-2025	South	Air Conditioners	Divya Kapoor	10	3,66,546	2,50,237	Delivered
ORD-00019	05-Apr-2025	West	Air Conditioners	Shreya Sinha	3	92,167	67,938	Delivered
ORD-00020	05-Apr-2025	North	Air Conditioners	Deepak Mishra	4	1,20,930	85,202	Delivered
ORD-00021	05-Apr-2025	East	Microwaves	Rahul Verma	2	14,321	9,294	Delivered
ORD-00022	06-Apr-2025	North	Air Conditioners	Aarav Sharma	1	30,159	19,655	Delivered
ORD-00023	06-Apr-2025	West	Washing Machines	Isha Joshi	3	51,941	41,493	Returned
ORD-00024	07-Apr-2025	East	Small Appliances	Rahul Verma	1	2,600	1,965	Delivered
ORD-00025	07-Apr-2025	West	Washing Machines	Tanvi Shah	7	1,28,759	89,073	Delivered
ORD-00026	08-Apr-2025	West	Air Conditioners	Meera Rao	5	1,47,082	1,15,594	Pending
ORD-00027	09-Apr-2025	South	Air Conditioners	Ananya Nair	6	1,76,187	1,20,764	Delivered
ORD-00028	09-Apr-2025	South	Refrigerators	Neha Pillai	10	2,75,595	2,12,687	Delivered
ORD-00029	10-Apr-2025	North	Microwaves	Arjun Patel	8	52,975	37,516	Delivered
ORD-00030	11-Apr-2025	East	Microwaves	Rahul Verma	9	58,923	36,886	Delivered
ORD-00031	11-Apr-2025	South	Small Appliances	Ritu Saxena	10	23,027	18,419	Delivered
ORD-00032	11-Apr-2025	West	Microwaves	Tanvi Shah	1	8,001	5,534	Delivered
ORD-00033	12-Apr-2025	East	Small Appliances	Manish Bhat	8	21,717	17,658	Delivered
ORD-00034	13-Apr-2025	South	Refrigerators	Divya Kapoor	2	52,940	38,865	Delivered
ORD-00035	13-Apr-2025	South	Small Appliances	Pooja Bose	3	7,133	5,778	Delivered


AutoSave Off
sample_sales_FY26 • Saved to this PC
File Home Insert Draw Page Layout Formulas Data Review View Havish M
Paste
Clipboard
Calibri
11
Font
Wrap Text
Merge & Ce
Alignment
L14
Monthly Revenue Targets – FY26
Region Month Target_Revenue
North Apr-2025 18,20,000
North May-2025 40,20,000
North Jun-2025 31,00,000
North Jul-2025 10,50,000
North Aug-2025 19,00,000
North Sep-2025 33,50,000
North Oct-2025 25,20,000
North Nov-2025 31,30,000
North Dec-2025 17,10,000
North Jan-2026 19,10,000
North Feb-2026 20,70,000
North Mar-2026 32,80,000
South Apr-2025 37,20,000
South May-2025 16,70,000
South Jun-2025 21,30,000
South Jul-2025 23,90,000
South Aug-2025 12,20,000
South Sep-2025 25,30,000
South Oct-2025 24,00,000
South Nov-2025 21,50,000
South Dec-2025 13,20,000
South Jan-2026 23,80,000
South Feb-2026 25,40,000
South Mar-2026 16,90,000
East Apr-2025 34,90,000
East May-2025 21,70,000
East Jun-2025 35,10,000
East Jul-2025 15,20,000
Raw_Data
Targets
Ready
Accessibility: Good to go

Sales Performance Dashboard – FY26

Upload your sales workbook

The workbook needs two sheets: `Raw_Data` and `Targets`.

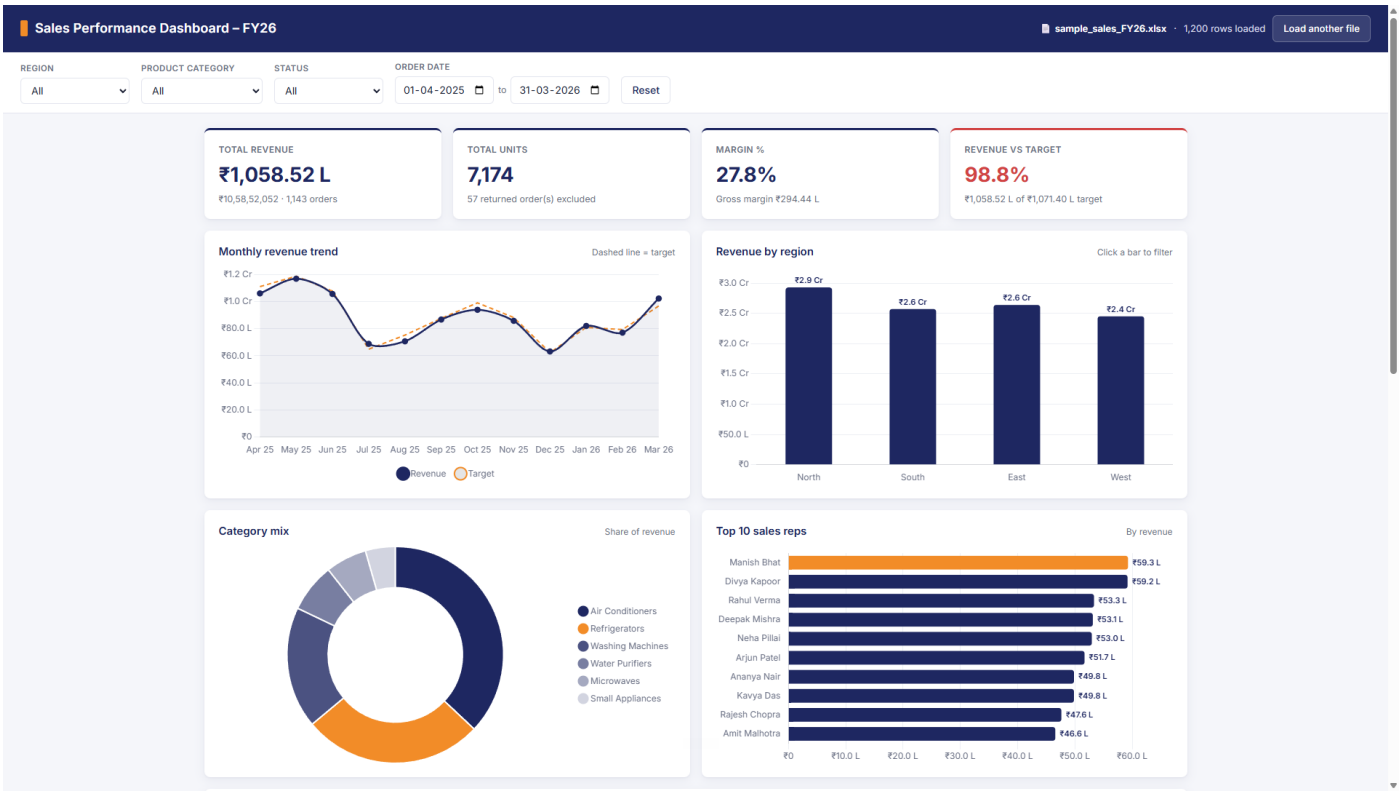
Drag & drop your Excel file here

or

Browse

.xlsx or .xls only

Your file is read and processed entirely in this browser. Nothing is uploaded to a server.



Order detail

Search order, rep, region...

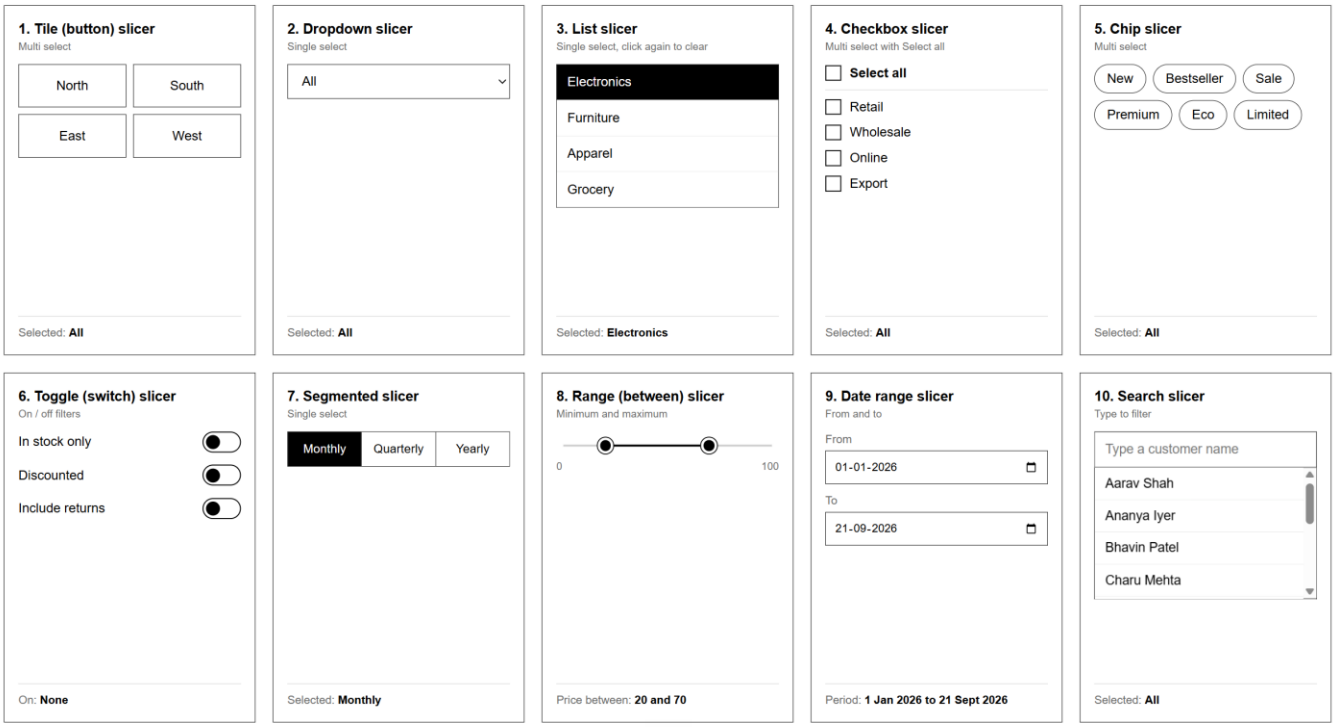
Export to CSV

ORDER ID	DATE ▼	REGION	CATEGORY	SALES REP	UNITS	REVENUE	COST	MARGIN %	STATUS
ORD-01197	31 Mar 2026	North	Microwaves	Vikram Singh	2	₹13,991	₹9,753	30.3%	Delivered
ORD-01198	31 Mar 2026	North	Washing Machines	Arjun Patel	1	₹17,692	₹14,209	19.7%	Delivered
ORD-01199	31 Mar 2026	North	Refrigerators	Suresh Agarwal	2	₹47,332	₹30,318	35.9%	Delivered
ORD-01200	31 Mar 2026	East	Refrigerators	Rahul Verma	7	₹1,47,045	₹1,04,564	28.9%	Returned
ORD-01194	30 Mar 2026	East	Washing Machines	Manish Bhat	1	₹15,801	₹10,983	30.5%	Delivered
ORD-01195	30 Mar 2026	South	Washing Machines	Pooja Bose	4	₹68,200	₹53,076	22.2%	Delivered
ORD-01196	30 Mar 2026	West	Refrigerators	Meera Rao	8	₹1,98,591	₹1,44,609	27.2%	Delivered
ORD-01193	29 Mar 2026	West	Refrigerators	Meera Rao	3	₹79,203	₹63,326	20.0%	Delivered
ORD-01190	28 Mar 2026	North	Small Appliances	Amit Malhotra	6	₹12,977	₹8,486	34.6%	Delivered
ORD-01191	28 Mar 2026	South	Small Appliances	Neha Pillai	8	₹18,156	₹13,496	25.7%	Delivered
ORD-01192	28 Mar 2026	North	Refrigerators	Suresh Agarwal	3	₹70,945	₹45,704	35.6%	Delivered
ORD-01189	27 Mar 2026	East	Water Purifiers	Karan Gupta	5	₹44,001	₹29,796	32.3%	Delivered
ORD-01188	26 Mar 2026	East	Water Purifiers	Karan Gupta	9	₹90,608	₹72,890	19.6%	Pending
ORD-01187	25 Mar 2026	North	Refrigerators	Suresh Agarwal	8	₹1,82,091	₹1,27,552	30.0%	Delivered
ORD-01185	24 Mar 2026	North	Microwaves	Vikram Singh	5	₹34,875	₹28,254	19.0%	Delivered
ORD-01186	24 Mar 2026	South	Water Purifiers	Ritu Saxena	3	₹29,749	₹19,500	34.5%	Pending
ORD-01176	23 Mar 2026	South	Washing Machines	Priya Iyer	1	₹16,746	₹11,159	33.4%	Delivered
ORD-01177	23 Mar 2026	North	Microwaves	Sanjay Menon	9	₹57,009	₹46,329	18.7%	Delivered
ORD-01178	23 Mar 2026	East	Refrigerators	Rajesh Chopra	9	₹2,23,401	₹1,59,012	28.8%	Pending
ORD-01179	23 Mar 2026	South	Small Appliances	Neha Pillai	2	₹5,161	₹3,922	24.0%	Pending
ORD-01180	23 Mar 2026	South	Air Conditioners	Divya Kapoor	2	₹65,147	₹44,574	31.6%	Returned
ORD-01181	23 Mar 2026	West	Small Appliances	Isha Joshi	11	₹28,863	₹21,277	26.3%	Pending
ORD-01182	23 Mar 2026	South	Small Appliances	Pooja Bose	5	₹11,508	₹7,389	35.8%	Delivered
ORD-01183	23 Mar 2026	South	Air Conditioners	Neha Pillai	5	₹1,54,161	₹1,19,962	22.2%	Pending
ORD-01184	23 Mar 2026	West	Washing Machines	Meera Rao	8	₹1,46,858	₹93,330	36.4%	Delivered

Showing 1–25 of 1,200 orders

‹ Prev Page 1 of 48 Next ›

Article – Slicers



Prompt:

Build a single-file HTML page that shows 10 types of dashboard slicers as a reference gallery.

Layout:

- A 5 × 2 grid of equal-height cards with a thin black border.

- Each card has a bold numbered title, a small grey subtitle describing the behaviour, the slicer itself, and a footer line "Selected: ..." that updates live as you interact.

The 10 slicers:

1. Tile (button) slicer: multi-select tiles North, South, East, West.

2. Dropdown slicer: single select, default "All".

3. List slicer: single-select list (Electronics, Furniture, Apparel, Grocery); clicking the selected item again clears it.

4. Checkbox slicer: Select all, then Retail, Wholesale, Online, Export.

5. Chip slicer: multi-select pill chips (New, Bestseller, Sale, Premium, Eco, Limited).

6. Toggle (switch) slicer: In stock only, Discounted, Include returns; the footer shows which are on.

7. Segmented slicer: Monthly / Quarterly / Yearly, single select.

8. Range (between) slicer: dual-handle slider from 0 to 100, default 20–70.

9. Date range slicer: From and To date inputs; the footer shows the period as "1 Jan 2026 to 21 Sept 2026".

10. Search slicer: a text box that filters a scrollable list of customer names.

Behaviour:

- When nothing is selected, the footer shows "All" (or "None" for the toggles).

- Selected states are inverted to a black fill with white text.

Style:

- Minimal black-and-white wireframe look, sans-serif font, light grey page background.

- No colour accents.

- Responsive, so cards wrap to fewer columns on smaller screens.

- Plain HTML, CSS and JS with no libraries.

Simplified Prompt:

Slicers and sample data:

1. Tile buttons (multi-select): North, South, East, West

2. Dropdown (single): All, Electronics, Furniture, Apparel, Grocery

3. List (single, click again to clear): Electronics, Furniture, Apparel, Grocery

4. Checkboxes with Select all: Retail, Wholesale, Online, Export

5. Chips (multi-select): New, Bestseller, Sale, Premium, Eco, Limited

6. Toggle switches: In stock only, Discounted, Include returns

7. Segmented control (single): Monthly, Quarterly, Yearly

8. Dual-handle range slider: price 0 to 100

9. Date range: From and To date pickers

10. Search box that filters a list of 10 Indian customer names

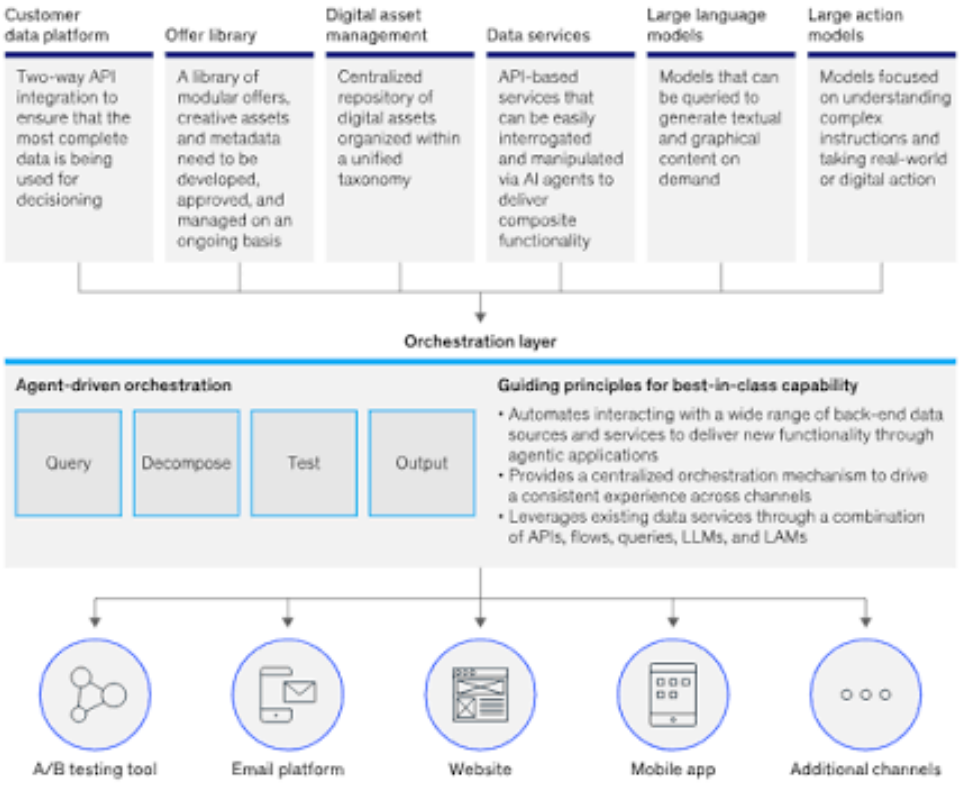
Selected items should invert (black fill, white text). No external libraries.

Article – Infographic

A	B	C	D	E	F	G	H	I	J
Order ID	Order Date	Sales Rep	Region	Dealer / Customer	Product Category	Product	Qty (Coils/Units)	Unit Price (₹)	Sales Value (₹)
RRK-1001	01-Oct-2026	Amit Sharma	North	Sharma Electricals, Delhi	House Wires	FR PVC Wire 1.5 sq mm (90m)	120	₹1,450	₹1,74,000
RRK-1002	02-Oct-2026	Priya Nair	South	Nair Traders, Kochi	House Wires	FR-LSH Wire 2.5 sq mm (90m)	85	₹2,350	₹1,99,750
RRK-1003	03-Oct-2026	Rahul Patil	West	Patil Agencies, Pune	Power Cables	LT XLPE Armoured 4C x 16 sq mm (100m)	6	₹48,500	₹2,91,000
RRK-1004	05-Oct-2026	Sneha Das	East	Das Electric Co., Kolkata	Flexible Cables	Multicore Flexible 3C x 1.5 sq mm (100m)	40	₹3,900	₹1,56,000
RRK-1005	07-Oct-2026	Vikram Singh	North	Singh Power Solutions, Jaipur	Solar Cables	DC Solar Cable 4 sq mm (100m)	55	₹4,200	₹2,31,000
RRK-1006	09-Oct-2026	Karthik Reddy	South	Reddy Enterprises, Hyderabad	House Wires	FR PVC Wire 4 sq mm (90m)	70	₹3,650	₹2,55,500
RRK-1007	12-Oct-2026	Neha Joshi	West	Joshi Hardware, Ahmedabad	Fans	Ceiling Fan 1200mm	150	₹2,450	₹3,67,500
RRK-1008	14-Oct-2026	Arjun Mehta	Central	Mehta Electricals, Indore	Switches	Modular Switch 6A (Box of 10)	200	₹420	₹84,000
RRK-1009	16-Oct-2026	Sneha Das	East	Bose Infra Projects, Bhubaneswar	Power Cables	LT Aluminium Armoured 3.5C x 95 sq mm (100m)	4	₹72,000	₹2,88,000
RRK-1010	18-Oct-2026	Amit Sharma	North	Gupta Builders, Lucknow	House Wires	FR-LSH Wire 6 sq mm (90m)	60	₹5,400	₹3,24,000

AI will drive major shifts in the martech stack including introducing an agentic orchestration layer.

How an Agentic AI orchestration layer optimizes the martech stack, illustrative



Source: McKinsey analysis

McKinsey & Company

Ready when you are.







Show insights in the format as the McKinsey one given.
Use my logo and colors.

Create an infographic.

Pro

SALES INSIGHTS

1-18 OCTOBER 2026

HMC ONE

AI STRATEGY. BUILT TO SCALE.

House wires and power cables account for 64.6% of sales value.

Sales insights from 10 supplied orders across 6 product categories and 5 regions.

₹23.71 lakh	10	₹2.37 lakh	₹3.68 lakh
TOTAL SALES VALUE	ORDERS IN THE SNAPSHOT	AVERAGE ORDER VALUE	LARGEST SINGLE ORDER

01 | PRODUCT MIX

Sales value and share of the total

House wires	Power cables	Fans	Solar cables	Flexible cables	Switches
40.2%	24.4%	15.5%	9.7%	6.6%	3.5%
₹9.53 lakh	₹5.79 lakh	₹3.68 lakh	₹2.31 lakh	₹1.56 lakh	₹0.84 lakh
4 orders	2 orders	1 order	1 order	1 order	1 order

WHAT THE SNAPSHOT SHOWS

64.6%	58.5%	21.0%
Top two categories	**North and West** combined	**Amit Sharma** Top sales rep

All shares are based on sales value.

QUESTIONS FOR SALES PLANNING

- Is stock coverage aligned with the two largest categories?
- Which North and West accounts can place repeat orders?
- Can the largest fan order be repeated with other dealers?

02 | REGIONAL FOOTPRINT

North and West contribute ₹13.88 lakh

North	West	South	East	Central
30.7%	27.8%	19.2%	18.7%	3.5%
₹7.29 lakh	₹6.59 lakh	₹4.55 lakh	₹4.44 lakh	₹0.84 lakh

Bars use a common 0-40% share scale.

LARGEST ORDER

Ceiling Fan 1200mm | ₹3,67,500 | 15.5% of total sales

RRK-1007

Source: Supplied order table, RRK-1001 to RRK-1010; HMC ONE calculations.
Scope: 10 orders dated 1-18 Oct 2026. This is a supplied snapshot, not a full-month total.
Sales are rounded to lakh; percentage totals may differ from 100% due to rounding.

HMC ONE

Prompt:

Show insights in the format as the McKinsey one given.

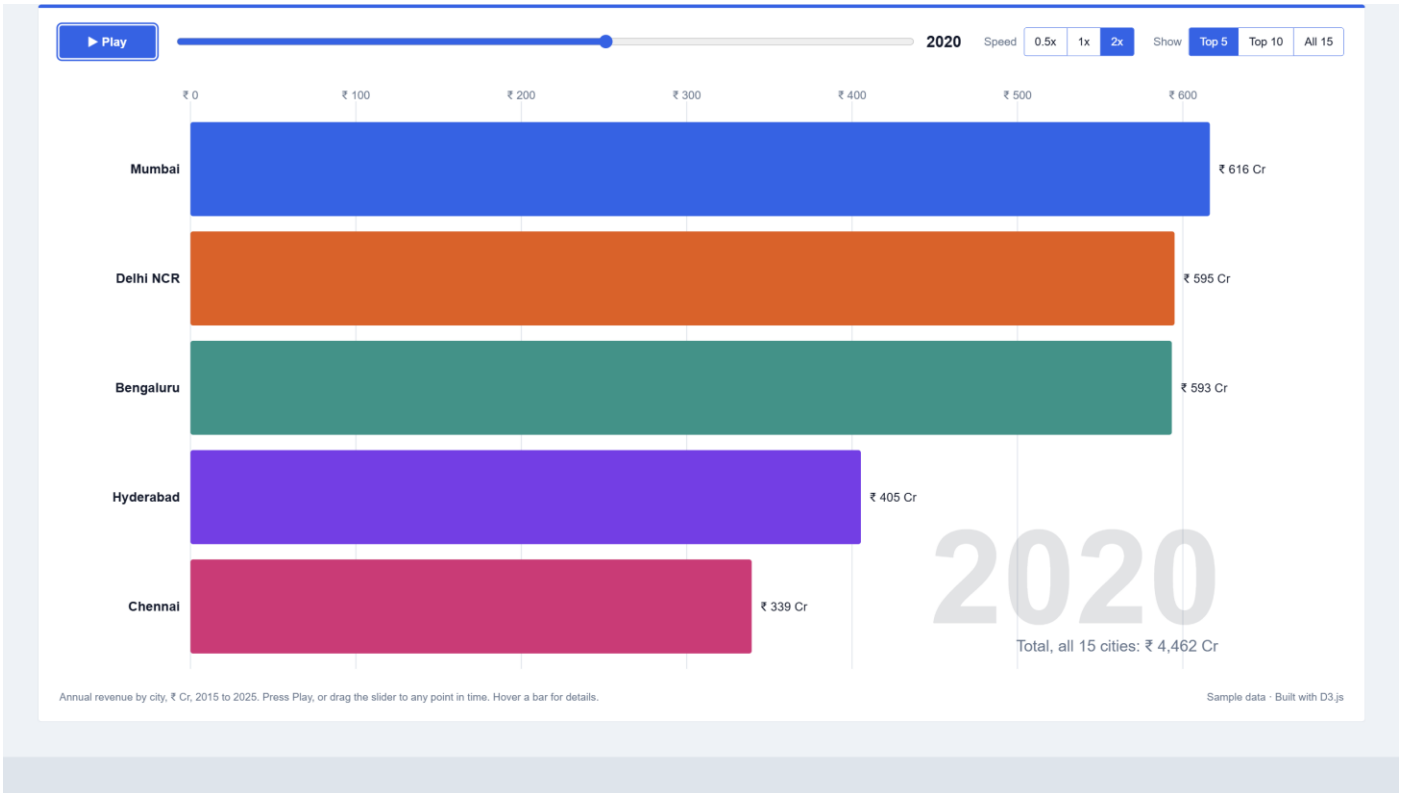
Use my logo and colors.

Create an infographic.

Article – D3js Charts

https://d3js.org/





Prompt:

Build a single-file HTML bar chart race using D3.js that shows annual revenue by Indian city from 2015 to 2026.

Data:

- Sample revenue in ₹ Crore for 15 cities: Mumbai, Delhi NCR, Bengaluru, Hyderabad, Chennai, Pune, Kolkata, Ahmedabad and others.

- Values should grow and change rank over the years so the race is interesting.

Controls bar at the top:

- A Play / Pause button.

- A year slider from 2015 to 2026, draggable to any point in time, with the current year shown beside it.

- Speed toggle: 0.5x / 1x / 2x.

- Show toggle: Top 5 / Top 10 / All 15.

Chart:

- Horizontal bars sorted largest to smallest; bars smoothly reorder, grow and shrink between years.

- Each city keeps the same distinct colour throughout.

- City name on the left of each bar and the value at the bar's end (e.g. "₹ 616 Cr").

- A ₹ axis along the top with light vertical gridlines.

- A large faded year watermark in the bottom-right, with "Total, all 15 cities: ₹ X Cr" beneath it.

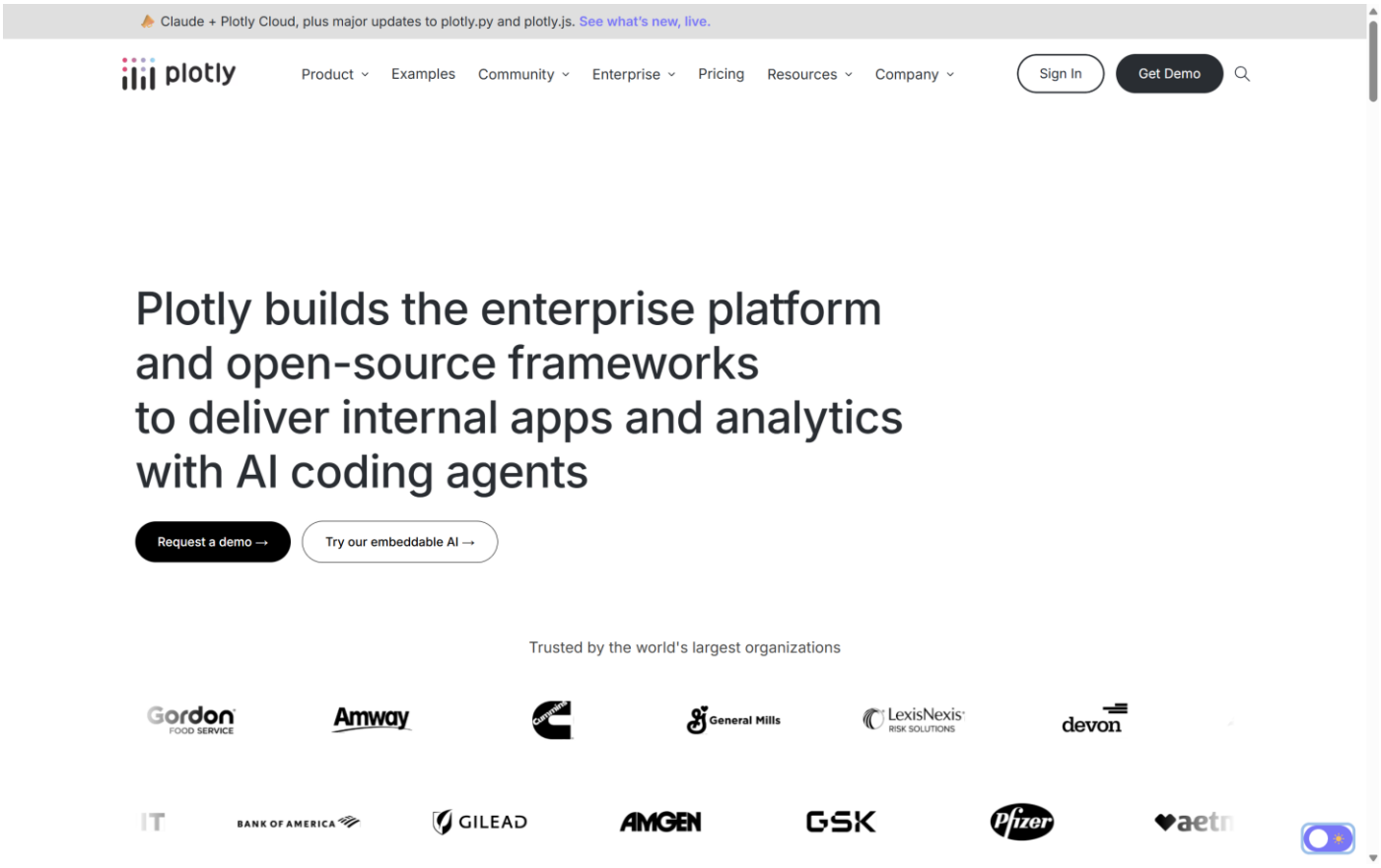
- Hovering a bar shows a tooltip with the city, year, revenue and rank.

Footer: a small caption explaining the chart on the left, and "Sample data · Built with D3.js" on the right.

Style: clean white card on a light grey background, a blue accent for the active buttons, sans-serif font, responsive width.

Article – Plotly Charts

https://plotly.com/





Prompt:

Build a single-file HTML animated waterfall chart using Plotly.js that bridges revenue to operating profit for FY 2025, in ₹ Crore.

Data (sample):

- Revenue: 1,248
- Cost of goods: −742
- Gross profit: 506 (subtotal)
- Sales & marketing: −186
- R&D: −92
- General & admin: −78
- Other income: +18
- Operating profit: 168 (total)

KPI strip at the top:

- Revenue, Gross profit, Operating profit and Operating margin (13.5%).
- Show the profit figures in green.

Controls on the top right:

- A Replay button.
- Speed toggle: 0.5x / 1x / 2x.
- Notes toggle: On / Off.

Chart:

- Plotly waterfall with dotted connector lines between bars.
- Colours: blue for revenue, subtotals and totals; red for costs; green for income.
- Value labels inside or on each bar.
- The y-axis is labelled ₹ Cr, with light gridlines.
- Bars build one at a time in sequence when the page loads and when Replay is clicked.
- Hovering a bar shows its name, value and running total.

Annotations (shown when Notes is On):

- Callout boxes with leader lines:
  - Revenue up 12.4% on FY 2024.
  - Cost of goods at 59.5% of revenue, 1.2 pts better than FY 2024.
  - Gross margin 40.5%.
  - Sales & marketing includes a ₹ 48 Cr festive campaign.
  - Operating margin 13.5%, up 2.1 pts on FY 2024.

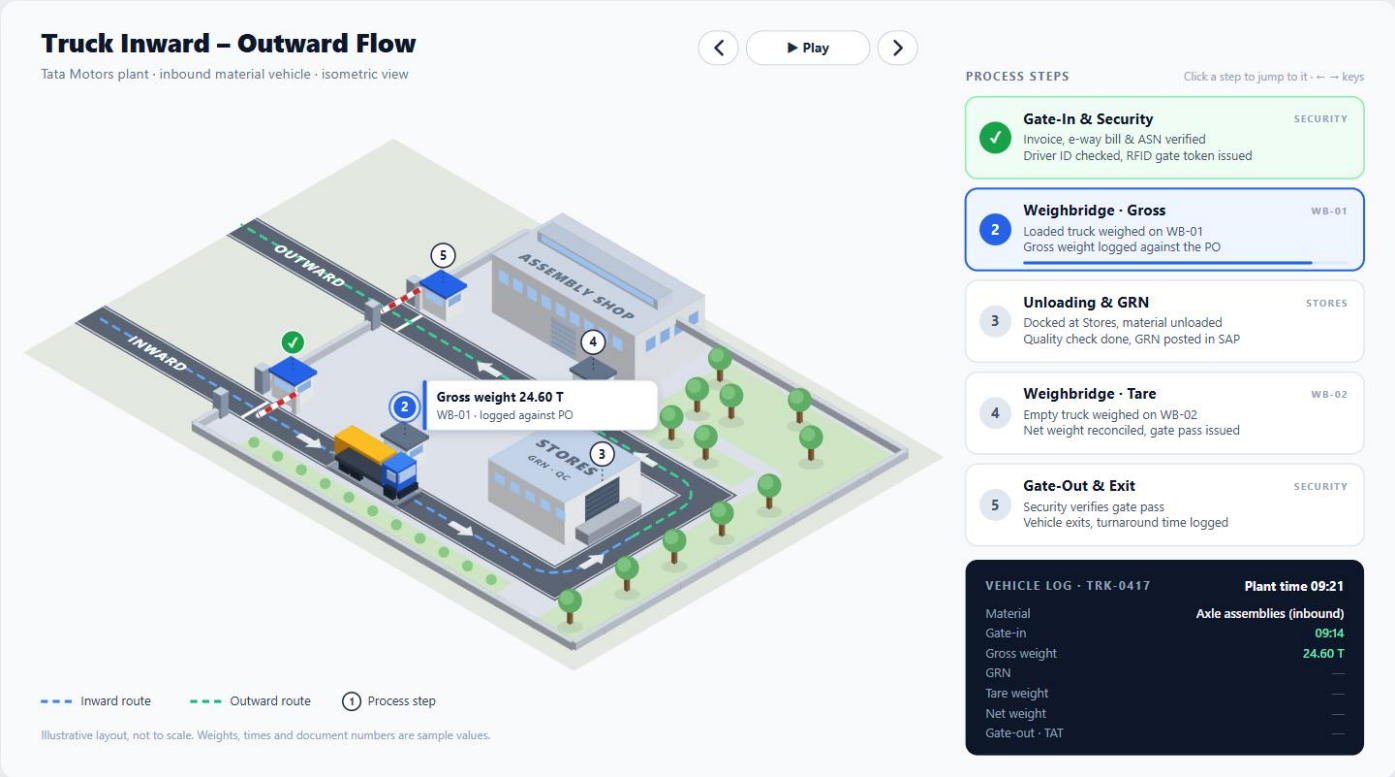
Footer: a caption on the left explaining the chart, and "Sample data · Built with Plotly.js" on the right.

Style: clean white card on a light grey background with a thin blue top border, a blue accent for the active buttons, sans-serif font, responsive width.

Article – SVG

https://www.adobe.com/in/creativecloud/file-types/image/vector/svg-file.html

The SVG file format is a popular tool for displaying two-dimensional graphics, charts and illustrations on websites. Plus, as a vector file, it can be scaled up or down without losing any of its resolution.



Prompt:

Create a detailed, animated isometric diagram of truck inward and outward movement at a Tata Motors plant, as a single self-contained HTML file with the scene drawn in SVG.

STYLE

- Isometric view, simple flat colours (blue, orange, grey, green on a light background).
- No company logo. Add a small note: "Illustrative layout, not to scale. Weights, times and document numbers are sample values."

SCENE

- Plant boundary wall, an inward road and an outward road, each with a security gate, cabin and animated boom barrier.
- Two weighbridges (WB-01 on the inward lane, WB-02 on the outward lane), each with a cabin.
- A Stores / GRN building with an unloading dock, an Assembly Shop, trees and greenery.
- Keep every stop visible: no building may hide the truck or a step marker.

- Moving dashed route lines: blue for inward, green for outward, with direction arrows.

ANIMATION (5 steps, about a 40-second loop) A loaded truck drives in, stops at each step, then exits:

1. Gate-In & Security: invoice, e-way bill, ASN and driver ID checked; barrier lifts.
2. Weighbridge (gross): WB-01 counts up to 24.60 T, logged against the PO.
3. Unloading & GRN: truck docks at Stores, cargo empties, crates appear on the dock, QC done, GRN posted.
4. Weighbridge (tare): WB-02 reads 9.80 T, net 14.80 T reconciled, gate pass issued.
5. Gate-Out & Exit: gate pass verified, barrier lifts, truck exits, turnaround time logged.

ON-SCREEN INFORMATION

- Numbered markers in the scene: they pulse when active and turn into a green tick when done.
- A callout beside the active step showing live status (e.g. "Unloading… 60%", "Gross weight 24.60 T").
- A side panel with 5 step cards (title, two-line description, progress bar) that highlight as the truck moves.
- A dark "Vehicle Log" panel with a plant clock that fills in material, gate-in time, gross weight, GRN number, tare weight, net weight, and gate-out time with TAT.
- A legend for the inward route, outward route and process step.

CONTROLS

- Pause / Play button.
- Previous / Next step buttons.
- Clicking a step card or scene marker replays the truck into that step, then holds there.
- Keyboard: ← → change step, 1–5 jump to a step, Space pauses or plays.

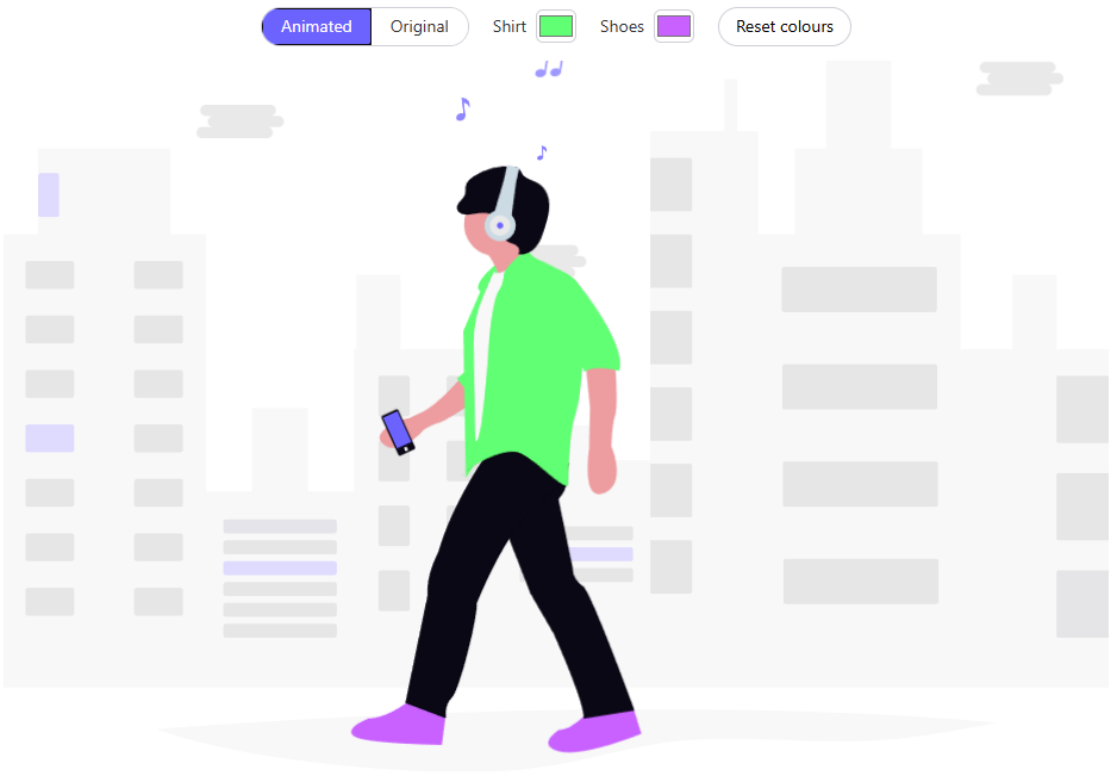
DOWNLOADS (inside the page, must work offline)

- "Download MP4 (1080p)" and "Download GIF" buttons under the scene, with a progress bar.
- Each renders the full loop in the browser and saves the file, leaving the control buttons out of the output.
- MP4: 1920 px wide, 30 fps, H.264 via the browser's WebCodecs encoder plus mp4-muxer. Fall back to a real-time MediaRecorder recording (WebM) where MP4 isn't supported, and say so.
- GIF: 1000 px wide, about 12 fps, loops forever, using gifenc with one shared palette.
- Embed both encoder libraries inside the HTML file (no CDN links), so it works without internet.

OUTPUT One .html file that runs as is in Chrome or Edge.

Article – Animate Static SVG

https://undraw.co/illustrations



Prompt:

I've attached an SVG illustration. Turn it into a single self-contained HTML file that animates it.

Rules:

- Keep the illustration exactly as it is: same shapes, colours, layout and proportions. Do not redraw, restyle or simplify anything.

- Only add motion. You may wrap existing elements in groups so they can move, but don't change their path data.

- Use only CSS and SVG animation (plus minimal vanilla JS if needed). No external libraries.

Animate it like this:

- Head: a gentle, steady nod, like someone listening to music. Hair and headphones move with the head.

- Upper body: a slow, subtle sway, pivoting at the hips. Legs stay still.

- Music player: the screen gently pulses in time with the nod.

- Music notes: add 3–4 small notes in the illustration's accent colour that float up from the headphones, drift sideways and fade out, staggered so they flow continuously.

- Clouds: drift slowly left and right, each at a different speed.

- Buildings: a few random windows softly light up in a pale tint of the accent colour, then fade back.

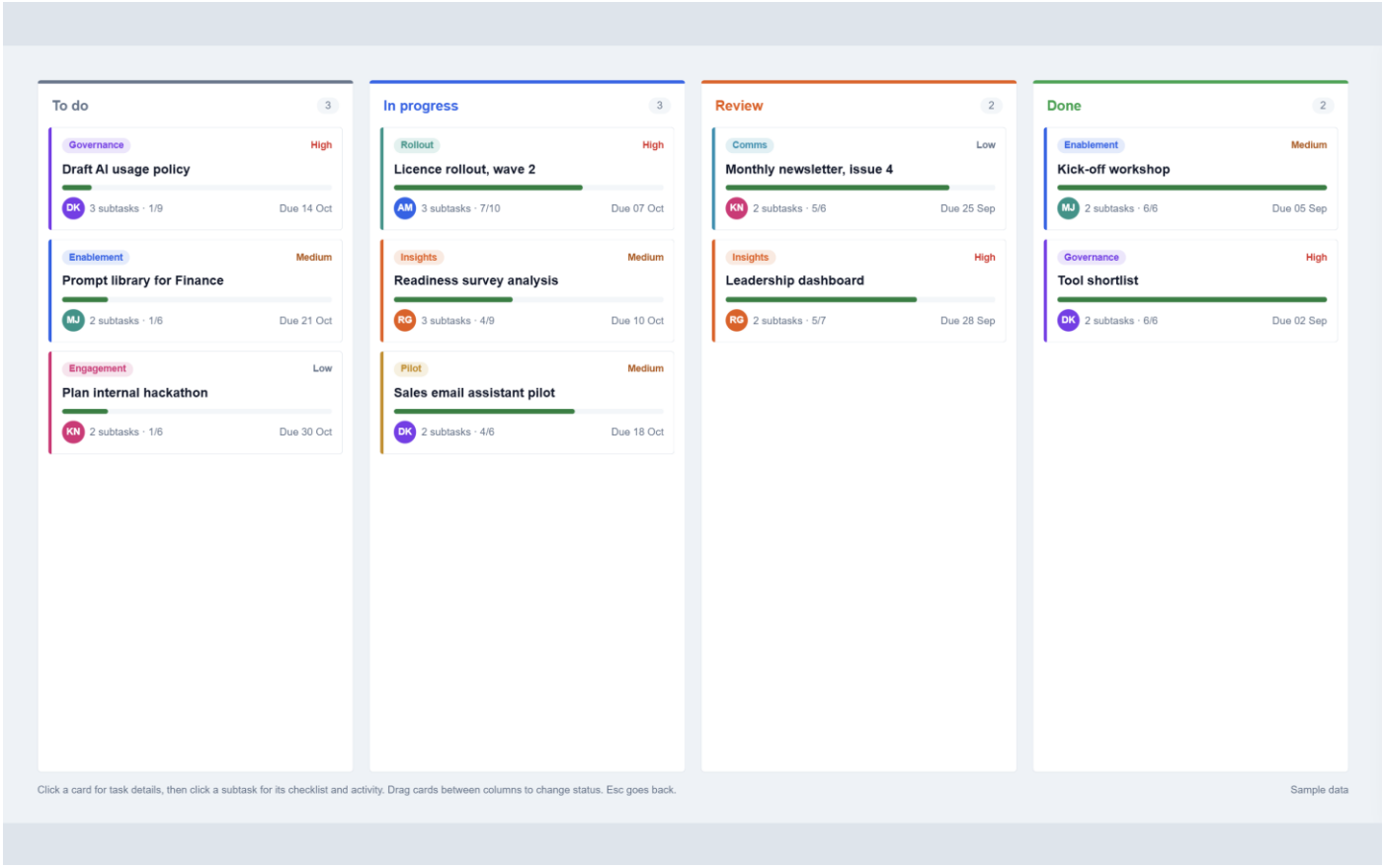
Make the motion smooth, loop seamlessly and stay subtle; it should feel calm, not busy.

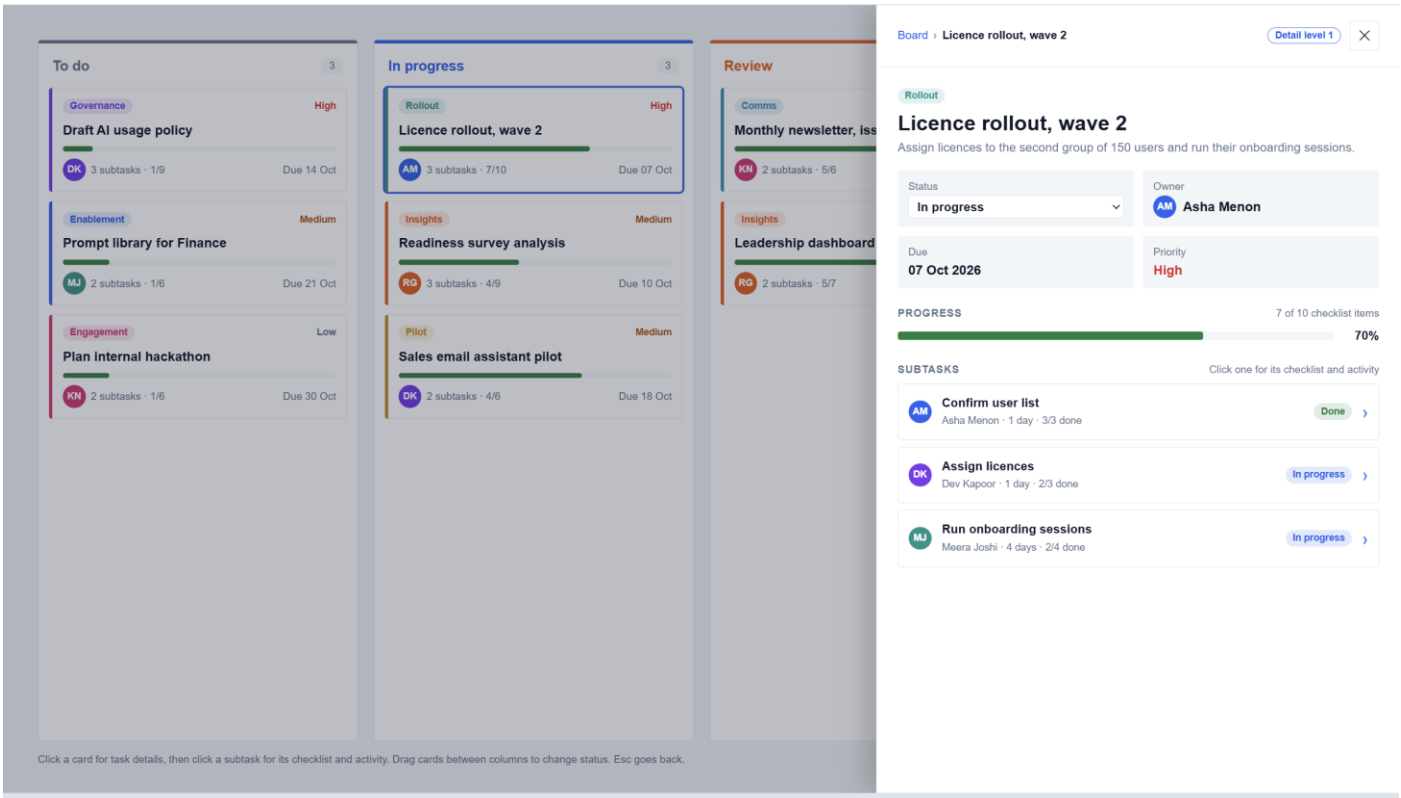
Layout:

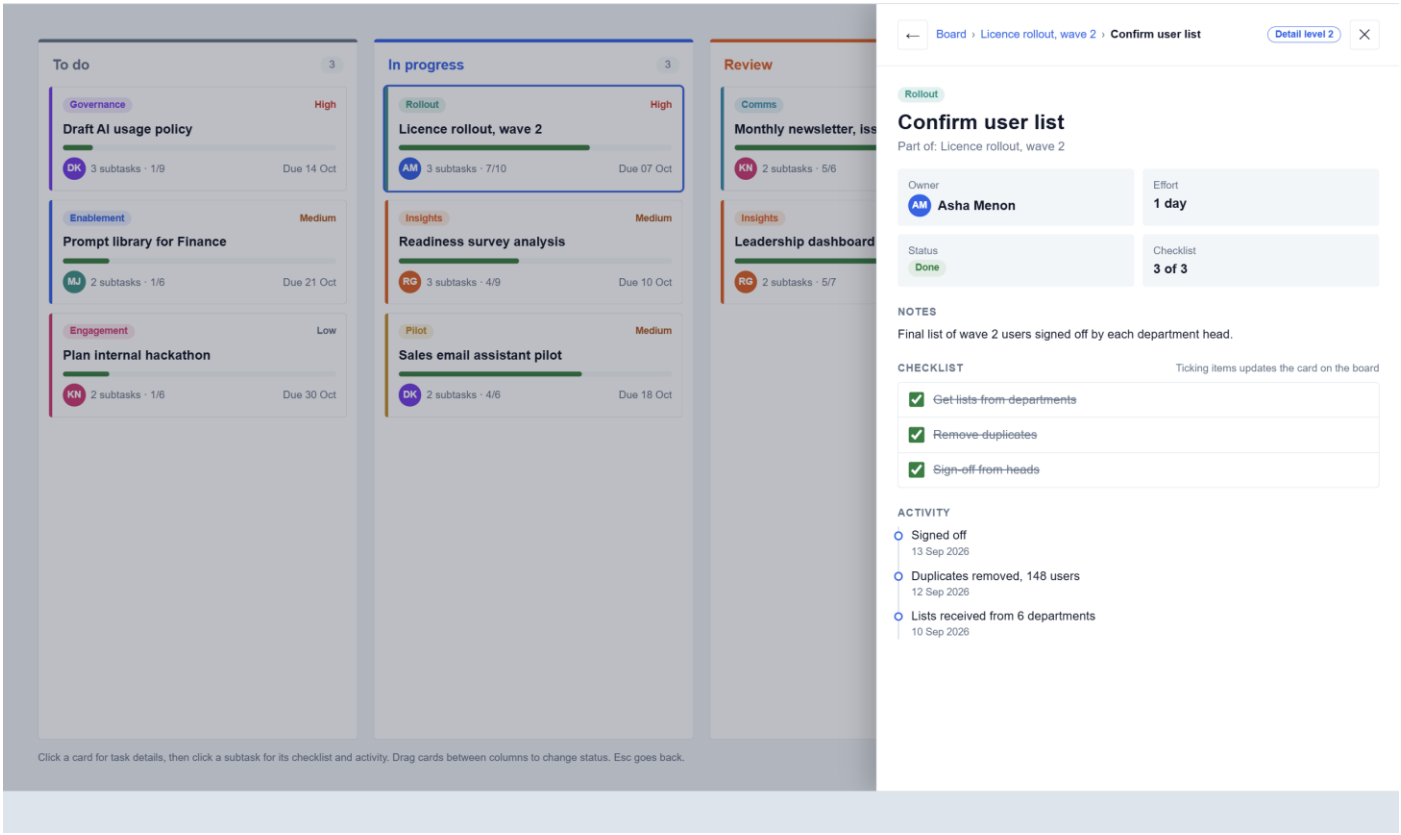
- Centre the illustration on a white page, scaling to fit any screen width, including mobile.

- Respect prefers-reduced-motion by turning the animation off for those users.

Article – Kanban







Prompt:

Build a single self-contained HTML file: an interactive Kanban board for tracking a company's AI adoption programme. No external libraries, fonts or images. All CSS and JS inline.

LAYOUT

- Fixed 1600×900 (16:9) stage, scaled with a CSS transform to fit and centre in any window (recalculate on resize). Page background slightly darker than the stage.

- Add print CSS: @page 1600×900, no margins, no transform when printing, and keep background colours.

- Helvetica/Arial, 14px base. Flat, clean style: 4px corner radius, thin borders, minimal shadows.

- Define all colours as CSS variables on :root, with a full dark theme that applies under prefers-color-scheme: dark (unless data-theme="light") and also under :root[data-theme="dark"].

BOARD

- Four equal columns: To do (slate), In progress (blue), Review (orange), Done (green). Each column has a 4px top border in its colour, its name in that colour, and a pill showing how many cards it holds. The card list scrolls inside the column.

- Each card has:

  - a left border in its tag colour

  - top row: a tag pill (tinted with the tag colour) and the priority (High red, Medium amber, Low grey)

  - a bold title

  - a thin green progress bar

  - bottom row: the owner's initials avatar (a coloured circle), "N subtasks · done/total", and "Due DD Mon"

- Cards lift slightly on hover.

- Drag and drop moves a card between columns. While dragging, the target column gets a dashed outline and the dragged card fades. A dropped card goes to the bottom of its new column.

- Cards are keyboard-focusable; Enter or Space opens them.

- Footer: a one-line usage hint on the left, "Sample data" on the right.

DETAIL DRAWER (two levels)

- Clicking a card slides in a 600px drawer from the right over a dim scrim. The selected card gets a blue ring.

- Drawer header: a back arrow (shown only at level 2), breadcrumbs (Board › Task › Subtask, each clickable), a "Detail level 1/2" pill, and a close button.

- Level 1 (task):

  - tag, large title, description

  - 2×2 facts grid: a Status dropdown that moves the card between columns, Owner with avatar, Due date with 2026 appended, Priority

  - an overall progress bar with %, calculated from all checklist items across all subtasks

  - a list of subtask buttons, each showing avatar, title, "owner · effort · x/y done", a status badge (Not started / In progress / Done) and a chevron

- Level 2 (subtask):

  - tag, title, "Part of: [task]"

  - facts grid: Owner, Effort, Status badge, Checklist x of y

  - Notes

  - a checklist with ticked items struck through. Ticking or unticking an item updates the board card live and adds "Completed: ..." or "Reopened: ..." to the activity log.

  - an activity log, newest first, drawn as a vertical timeline with dots

- The drawer body animates sliding forward or back when changing level.

- Esc goes back one level and closes the drawer from level 1. Clicking the scrim closes it.

- Escape all text before inserting it into the HTML.

SAMPLE DATA

Five people, each with a distinct avatar colour: Asha Menon, Rohan Gupta, Meera Joshi, Dev Kapoor, Kavya Nair.

Tags and colours: Governance (purple), Enablement (blue), Engagement (pink), Rollout (teal), Insights (orange), Pilot (yellow), Comms (cyan).

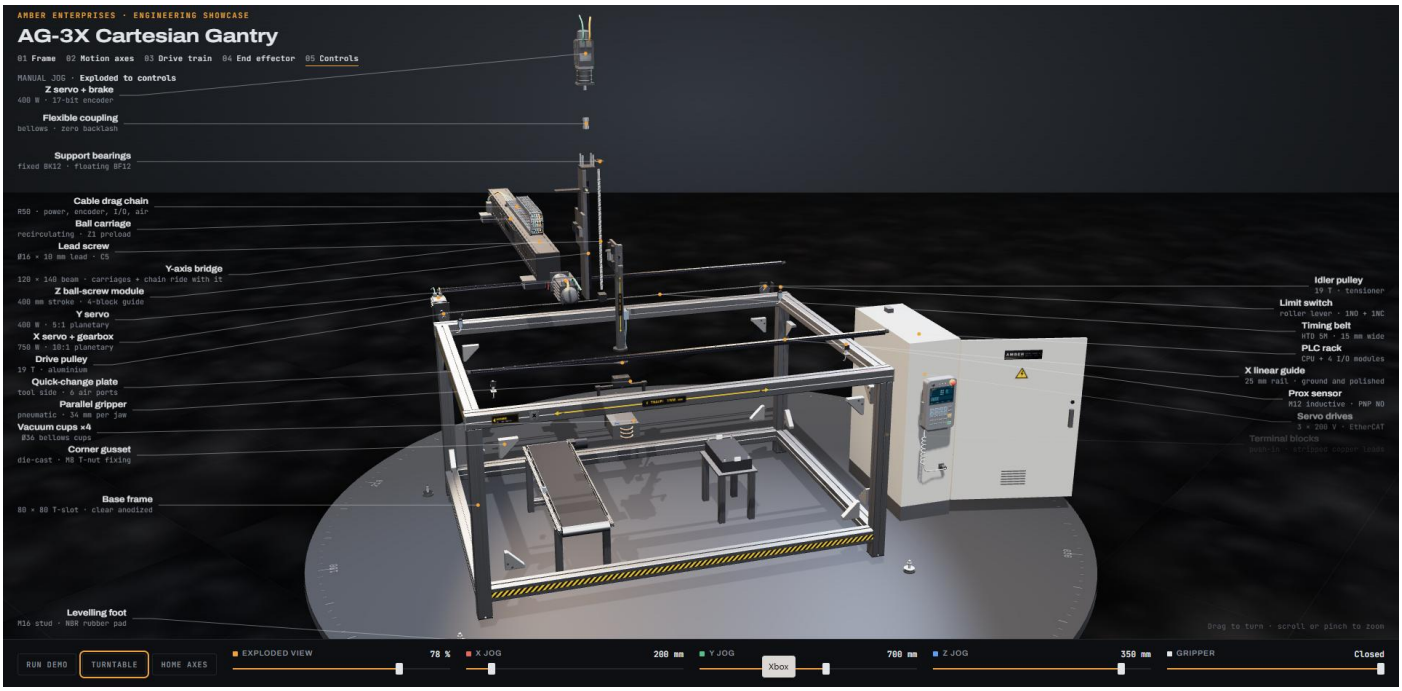
Ten tasks, each with a description and 2–3 subtasks. Each subtask has an owner, an effort estimate, notes, a 3–4 item checklist (some items already done) and dated activity entries from Aug–Sep.

- To do: Draft AI usage policy, Prompt library for Finance, Plan internal hackathon
- In progress: Licence rollout wave 2, Readiness survey analysis, Sales email assistant pilot
- Review: Monthly newsletter issue 4, Leadership dashboard
- Done: Kick-off workshop, Tool shortlist

Use realistic details, e.g. 148 users assigned licences, onboarding sessions with ~40 attendees, 386 clean survey responses, a pilot logging 146 drafts at 4.2/5 satisfaction.

Article – 3D Models

https://get.webgl.org/



Prompt:

Create an interactive 3D exploded view of an Amber Enterprises three-axis Cartesian gantry robot, presented at a three-quarter angle in a near-black industrial photography studio. Give it the precision and finish of a premium engineering showcase, with soft white strip reflections, subtle warm and cool rim highlights, a dark polished concrete floor with faint grid lines, and a softly backlit wall that separates the frame silhouette from the background. The gantry should rotate slowly on a turntable while its components separate along their own axes of motion.

Show the gantry construction in a staged teardown: anodized aluminium base frame with T-slot extrusions and corner gussets, levelling feet with rubber pads, twin X-axis linear guide rails with recirculating ball carriages, toothed timing belt with idler and drive pulleys, Y-axis bridge beam, Z-axis vertical ball-screw module with lead screw, support bearings and flexible coupling, three servo motors with planetary gearboxes and encoder caps, cable drag chain carrying colour-coded wiring, inductive proximity sensors and limit switches, pneumatic end effector with parallel gripper and four vacuum suction cups, quick-change tool plate, and a control cabinet with PLC, servo drives, terminal blocks, and a green circuit board with gold contacts. Include a wired teach pendant with a red emergency-stop button and detailed screen and keypad on both faces. Keep the Y-axis bridge, its carriages, and its drag chain together as one moving assembly. Space the components so they remain clear of one another throughout disassembly and reassembly.

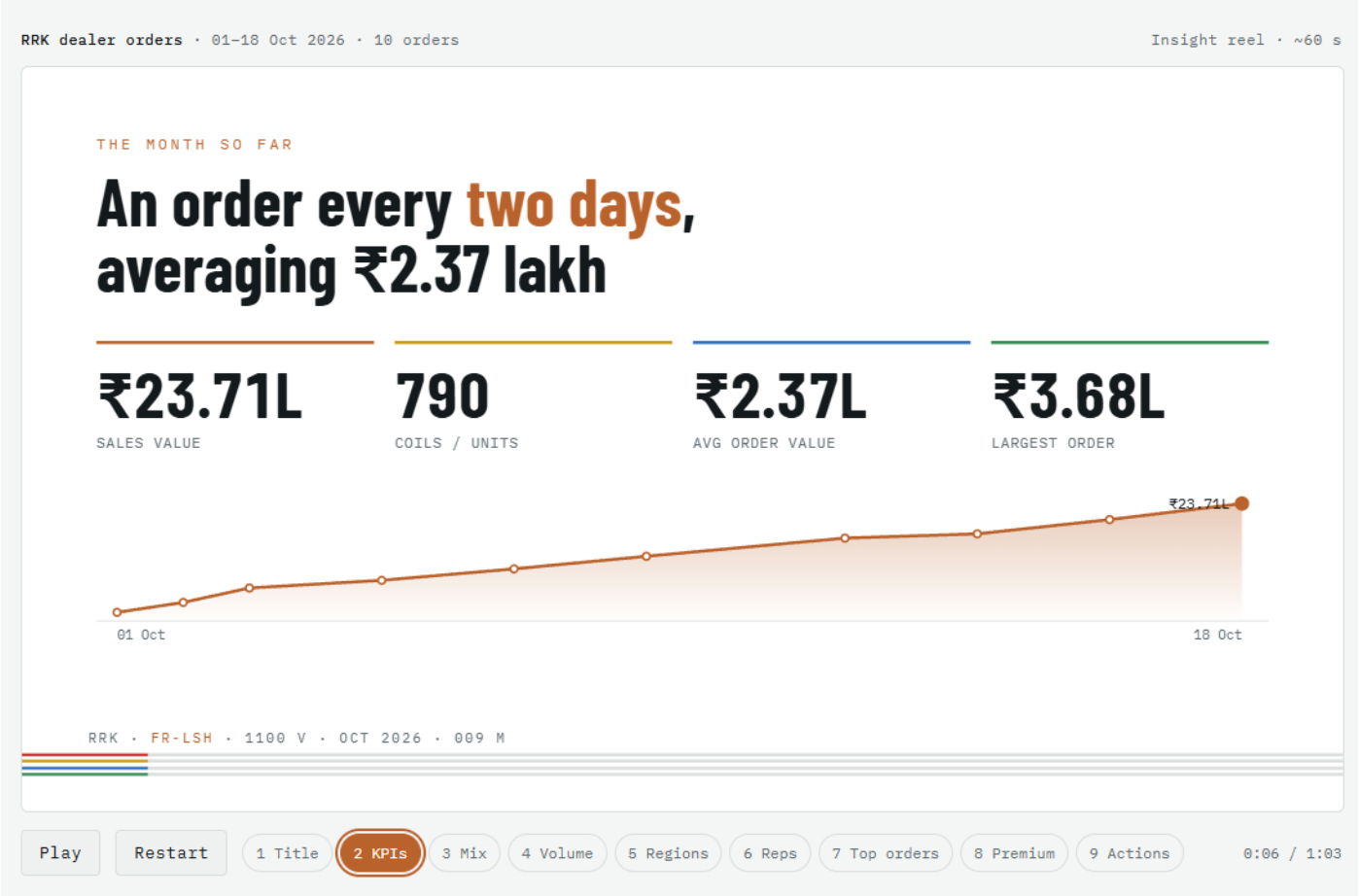
Make the materials convincing at close range: clear-anodized brushed aluminium, black powder-coated steel, ground and polished chrome rails, machined bearing blocks with fine tool marks, satin black motor housings with cooling fins, rubber timing belts with crisp tooth profiles, translucent blue pneumatic tubing, orange and grey cable insulation, copper conductors at stripped ends, and yellow-

black hazard striping on the frame edges. Give each ball screw and rail a distinct specular highlight so thin parts remain readable against the dark background. Engraved axis labels (X, Y, Z), the AMBER nameplate on the bridge beam, motor rating plates, travel-limit markings, and a millimetre scale along the X rail should remain crisp. Add direction-of-travel arrows, a home-position marker, and an orange caution label near the gripper.

Provide separate sliders for exploded-view amount, X/Y/Z jog position, and gripper open/close, plus an automatic demonstration that smoothly runs a pick-and-place cycle (lifting a copper heat-exchanger coil from an infeed conveyor and placing it on a fixture), then disassembles and reassembles the machine. Use staggered movement to reveal the frame, motion axes, drive train, end effector, and controls in sequence. Fade in fine leader-line labels as the gantry opens, and reposition them smoothly so they do not overlap. Keep the composition calm and legible, with realistic metal reflections, soft contact shadows, and controlled highlights throughout.

Article – Motion Video

A	B	C	D	E	F	G	H	I	J
Order ID	Order Date	Sales Rep	Region	Dealer / Customer	Product Category	Product	Qty (Coils/Units)	Unit Price (₹)	Sales Value (₹)
RRK-1001	01-Oct-2026	Amit Sharma	North	Sharma Electricals, Delhi	House Wires	FR PVC Wire 1.5 sq mm (90m)	120	₹1,450	₹1,74,000
RRK-1002	02-Oct-2026	Priya Nair	South	Nair Traders, Kochi	House Wires	FR-LSH Wire 2.5 sq mm (90m)	85	₹2,350	₹1,99,750
RRK-1003	03-Oct-2026	Rahul Patil	West	Patil Agencies, Pune	Power Cables	LT XLPE Armoured 4C x 16 sq mm (100m)	6	₹48,500	₹2,91,000
RRK-1004	05-Oct-2026	Sneha Das	East	Das Electric Co., Kolkata	Flexible Cables	Multicore Flexible 3C x 1.5 sq mm (100m)	40	₹3,900	₹1,56,000
RRK-1005	07-Oct-2026	Vikram Singh	North	Singh Power Solutions, Jaipur	Solar Cables	DC Solar Cable 4 sq mm (100m)	55	₹4,200	₹2,31,000
RRK-1006	09-Oct-2026	Karthik Reddy	South	Reddy Enterprises, Hyderabad	House Wires	FR PVC Wire 4 sq mm (90m)	70	₹3,650	₹2,55,500
RRK-1007	12-Oct-2026	Neha Joshi	West	Joshi Hardware, Ahmedabad	Fans	Ceiling Fan 1200mm	150	₹2,450	₹3,67,500
RRK-1008	14-Oct-2026	Arjun Mehta	Central	Mehta Electricals, Indore	Switches	Modular Switch 6A (Box of 10)	200	₹420	₹84,000
RRK-1009	16-Oct-2026	Sneha Das	East	Bose Infra Projects, Bhubaneswar	Power Cables	LT Aluminium Armoured 3.5C x 95 sq mm (100m)	4	₹72,000	₹2,88,000
RRK-1010	18-Oct-2026	Amit Sharma	North	Gupta Builders, Lucknow	House Wires	FR-LSH Wire 6 sq mm (90m)	60	₹5,400	₹3,24,000



Prompt:

You are a sharp business analyst and motion designer. Analyse the Excel data I've attached and present the insights as a short motion video built in a single self-contained HTML file.

STEP 1 — UNDERSTAND THE DATA

- Identify what each column is: dimensions (region, product, person, customer, date) and measures (quantity, price, value, cost).

- If a total/value column is missing, derive it (e.g., Qty × Unit Price). State the date range, row count and the main unit of currency.

- Flag anything odd (missing values, outliers, duplicate IDs) in one line. Don't stop for it.

STEP 2 — FIND THE STORY (compute, don't estimate)

Run the actual numbers and pick the 6–8 strongest insights, choosing from:

- Headline KPIs: total value, total volume, average per record, largest single record

- Trend over time: run-rate or cumulative line, if dates exist

- Mix: share of value by the main category dimension

- Volume vs value gaps: items that move many units but earn little, and the reverse

- Geography / segment split: who is strong, who is under-covered

- People / owner performance: top contributors and concentration (e.g., top 3 = X%)

- Concentration risk: share held by the top few records or customers

- Price or product ladders, premium vs standard mix, if visible in the data

Every insight must be a specific, quantified claim (e.g., "North and West bring 58% of value"). It must not be a description like "Sales vary by region". Percentages must add up. Double-check all arithmetic.

STEP 3 — BUILD THE MOTION VIDEO (HTML)

- A 16:9 stage (1280×720, scaled to fit the screen), auto-playing 8–10 scenes of 6–8 seconds each (about 60 seconds total).

- Scene flow: Title (headline number counts up) → KPIs + trend → 4–6 insight scenes → Top records → 3 recommended actions → close.

- Each scene: a small eyebrow label, a headline that states the insight with the key number highlighted, and one visual (animated bars, counters, line draw, donut, ranked list, or split comparison).

- Motion: staggered fade-and-rise for text, bars growing from zero, numbers counting up, lines drawing in. Keep it smooth and purposeful.

- Controls below the stage: Play/Pause, Restart, clickable scene chips, time counter. Space = pause, ← → = step scenes. Respect prefers-reduced-motion.

- Design: white background, clean professional look, one accent colour drawn from the subject's world plus 3–4 supporting colours, condensed display font for headlines, clean sans for body, mono for numbers (Google Fonts). Use tabular numbers and Indian number formatting (₹ lakh/crore) when the data is in INR.

- Charts must be drawn to scale. Labels must not overlap. Text must be readable on a phone.

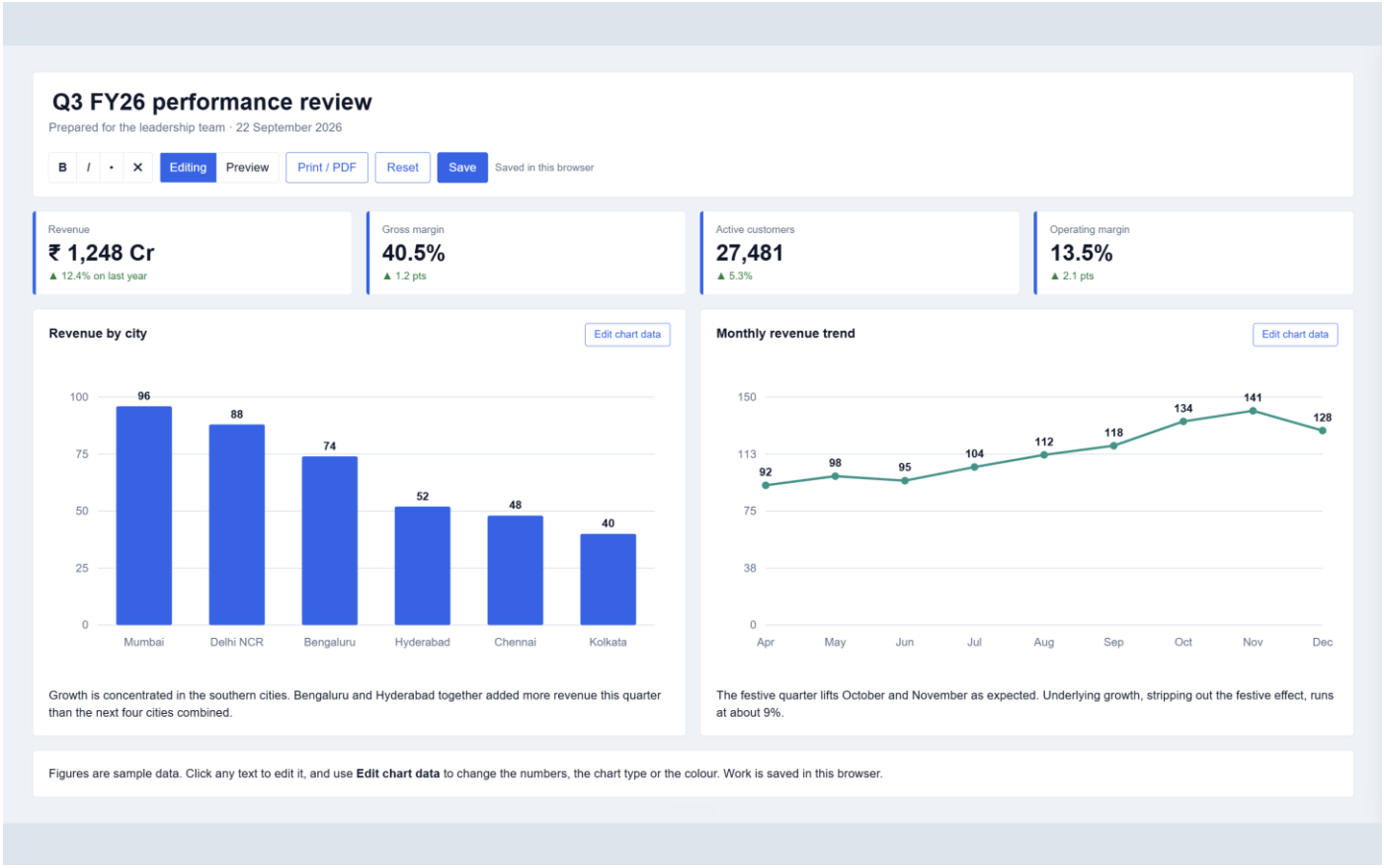
- Last scene: 3 concrete, data-backed actions (what to do, where, and the rough ₹ impact), plus a one-line source note.

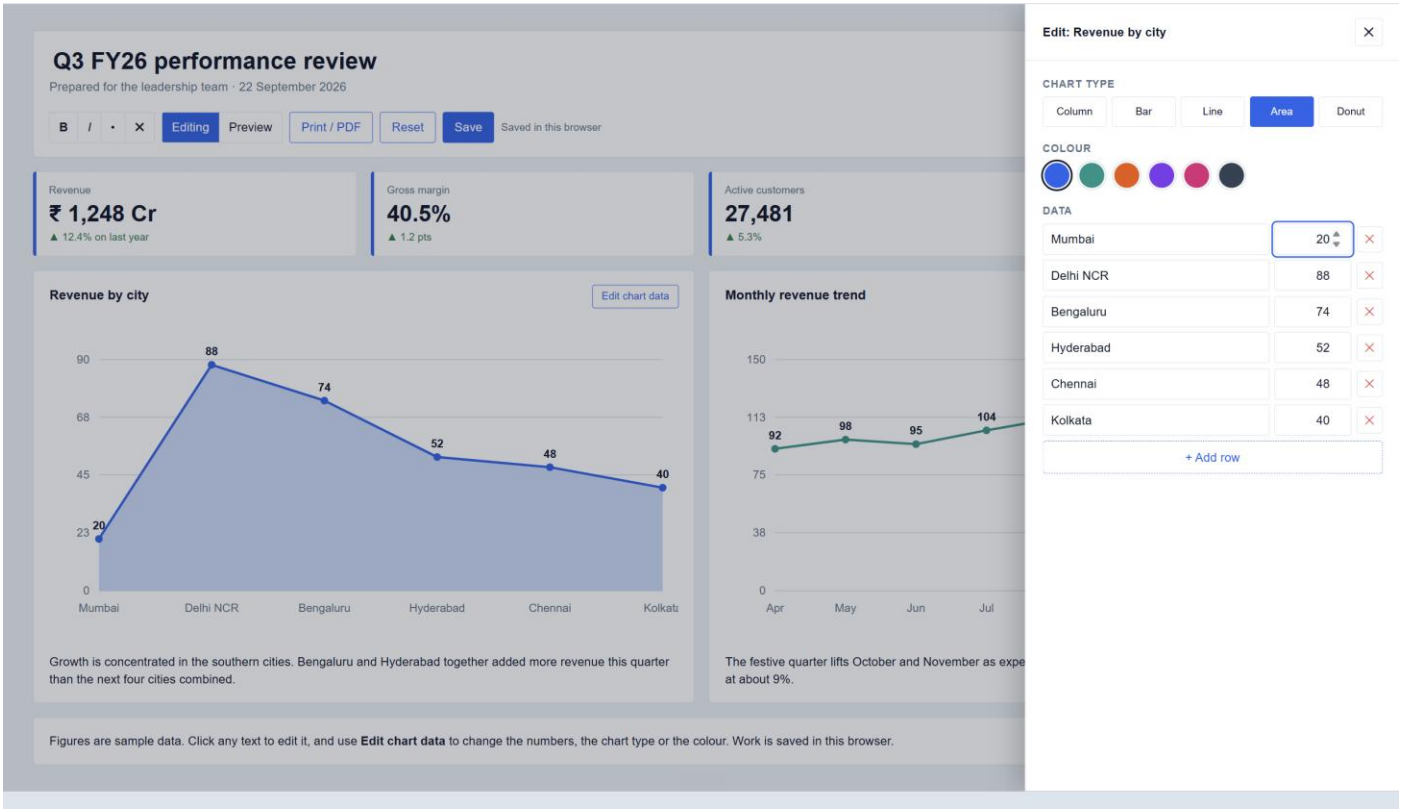
STEP 4 — DELIVER

- Publish the HTML as a shareable page.

- In chat, list the top 5 insights as one-liners. Skip any preamble.

Article – Editable Report





Prompt:

Build a single-file HTML "editable report" that fits a fixed 1600×900 (16:9) stage, scaled to fit any browser window and centred. No external libraries; plain HTML, CSS and vanilla JS only.

LAYOUT (top to bottom, inside the stage)

1. Title card: an editable report title (H1) and an editable subtitle line (period, audience, date). On the right, a toolbar:

- Formatting group: Bold, Italic, Bulleted list, Clear formatting (use execCommand; keep the text selection when clicking)

- Mode toggle: Editing / Preview

- Buttons: Print / PDF, Reset (confirm first, then restore sample data), Save (solid accent)

- A small status label ("Saved in this browser", "Editing…", "Sample report", "Loaded your last version")

2. KPI strip: 4 cards in a row, each with an editable label, big value and a small green "change" line, with a coloured left border.

3. Two chart cards side by side. Each has an editable chart title, an "Edit chart data" button (visible only in Editing mode), the chart, and an editable commentary paragraph below.

4. Footnote card: editable text for key points, method note or source.

EDITING

- All text is contenteditable in Editing mode, with dashed hover outline and accent outline on focus; placeholders shown via data-ph when empty.

- Preview mode turns editing off, hides edit buttons and closes the drawer.

- Paste as plain text only.

- Autosave the whole state (text, KPIs, charts) to localStorage with a ~700 ms debounce, save immediately on Save and on page unload. Wrap all storage calls in try/catch.

CHARTS (hand-drawn SVG, viewBox 660×300)

- Types: Column, Bar (horizontal), Line, Area, Donut.

- Gridlines with 4 steps and a "nice" rounded max; value labels on bars/points; category labels on the x-axis.

- Donut: centre total, legend on the right with values, slices as tints of the chosen colour.

- Clicking a chart (or its button) in Editing mode opens a right-side drawer (430px, slides in, with a scrim; closes on ×, scrim click or Escape) containing:

- Chart type selector (5 buttons)

- Colour swatches (6: blue #2563eb, teal #0d9488, orange #ea580c, violet #7c3aed, pink #db2777, slate #334155)

- Data rows: label + numeric value + delete button, and an "+ Add row" button

- Every change redraws the chart live and saves.

STYLE

- Clean corporate look: Helvetica/Arial, 14px, tabular numbers, white cards with 1px light borders and 4px radius on a soft blue-grey page, blue accent #2563eb.

- Colours as CSS variables on :root, with a full dark theme under prefers-color-scheme (guarded by :root:not([data-theme="light"])) and again under :root[data-theme="dark"].

- Print stylesheet: drop the scaling, hide toolbar/drawer/edit buttons, remove edit outlines, white background, keep colours (print-color-adjust: exact).

SAMPLE DATA (Indian context, ₹ Crore)

- Title: "Q3 FY26 performance review"; subtitle "Prepared for the leadership team · [date]"

- KPIs: Revenue ₹1,248 Cr (▲12.4% on last year), Gross margin 40.5% (▲1.2 pts), Active customers 27,481 (▲5.3%), Operating margin 13.5% (▲2.1 pts)

- Chart 1, column, "Revenue by city": Mumbai 96, Delhi NCR 88, Bengaluru 74, Hyderabad 52, Chennai 48, Kolkata 40, with a one-line commentary

- Chart 2, line, "Monthly revenue trend": Apr 92, May 98, Jun 95, Jul 104, Aug 112, Sep 118, Oct 134, Nov 141, Dec 128, with commentary on the festive-season lift

- Footnote explaining that figures are sample data and how to edit.

Escape all user-entered labels before injecting into SVG. Deliver as one self-contained .html file.

Simplified Prompt:

Build a single HTML file: an editable one-page business report.

Layout: fixed 1600 x 900 (16:9) canvas that scales to fit the window. Clean white cards on a light grey-blue background, blue accent, Helvetica, supports dark mode.

Sections:

1. Title bar: editable title and subtitle, plus a toolbar with Bold, Italic, Bullets, Clear formatting, an Editing / Preview toggle, Print / PDF, Reset and Save buttons, and a small save-status label.

2. Four KPI cards (label, big value, green change line), all editable. Sample: Revenue ₹1,248 Cr, Gross margin 40.5%, Active customers 27,481, Operating margin 13.5%.

3. Two charts side by side, each with an editable title and commentary below. Sample: "Revenue by city" (6 Indian cities, column chart) and "Monthly revenue trend" (Apr to Dec, line chart).

4. Editable footnote card.

Editing:

- Click any text to edit it; paste as plain text only.

- An "Edit chart data" button opens a side drawer where I can switch chart type (Column, Bar, Line, Area, Donut), pick a colour from 6 swatches, and edit, add or delete label/value rows. The chart redraws live.

- Preview mode hides all editing controls.

- Auto-save to localStorage; Reset restores the sample report.

- Print hides the toolbar and drawer.

Draw charts as inline SVG. No external libraries.

Article – Before vs After

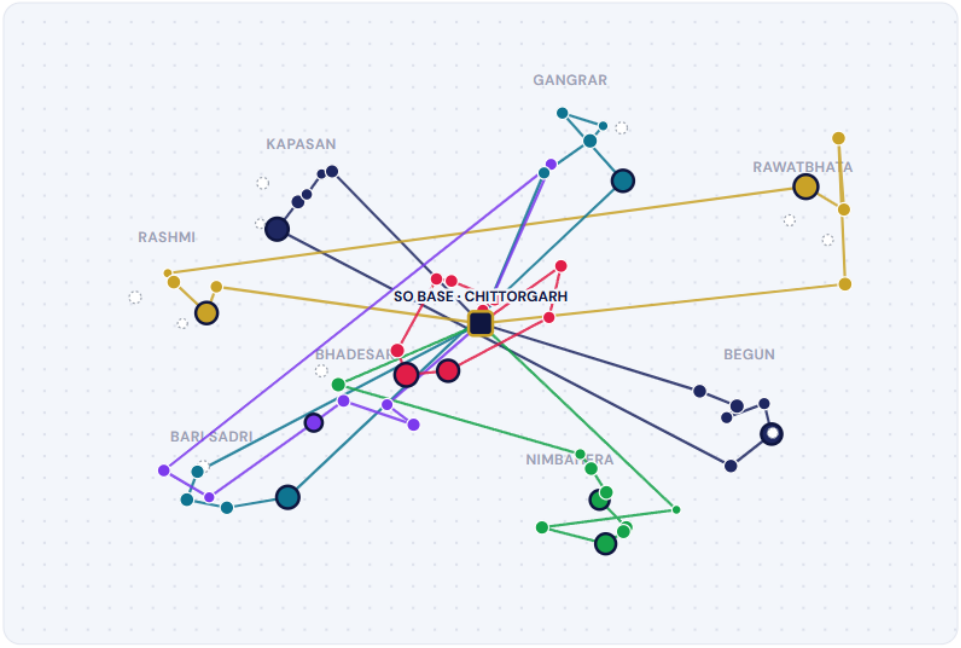
THE SOLUTION

Territory-clustered beats, sequenced to cut backtracking

One SO territory, one six-day cycle. Toggle to compare the inherited beat with the data-built one. Hover an outlet for detail; hover a day to isolate its route.

Before | After

1 SO · 64 outlets · Mon–Sat cycle · Illustrative layout



Mon · Tue · Wed · Thu · Fri · Sat · Not on a beat · Dealer (size = potential)

Avg travel per beat day
113 km
Backtracking across towns

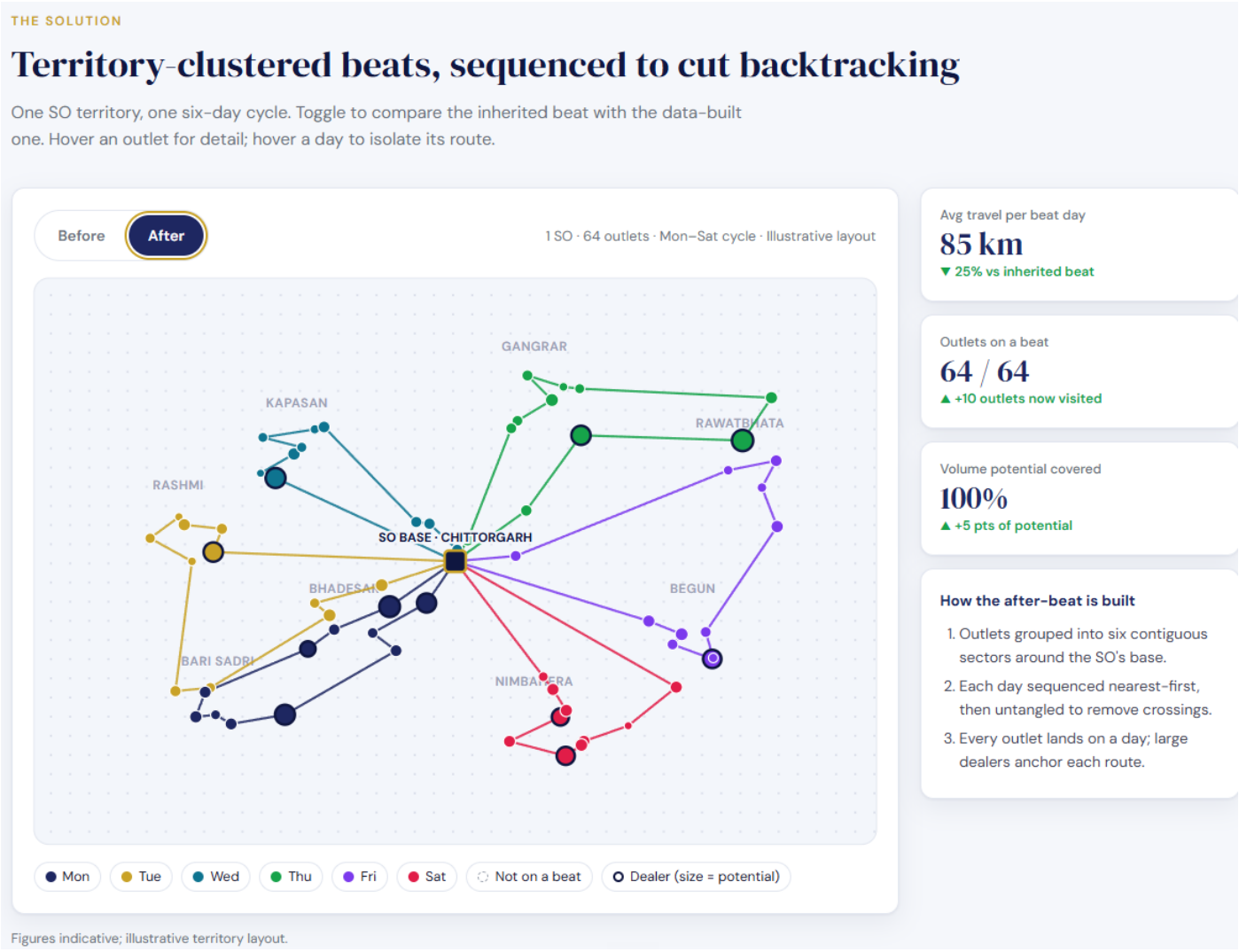
Outlets on a beat
54 / 64
10 outlets never visited

Volume potential covered
95%
Mostly C-class retailers missed

How the after-beat is built

1. Outlets grouped into six contiguous sectors around the SO's base.
2. Each day sequenced nearest-first, then untangled to remove crossings.
3. Every outlet lands on a day; large dealers anchor each route.

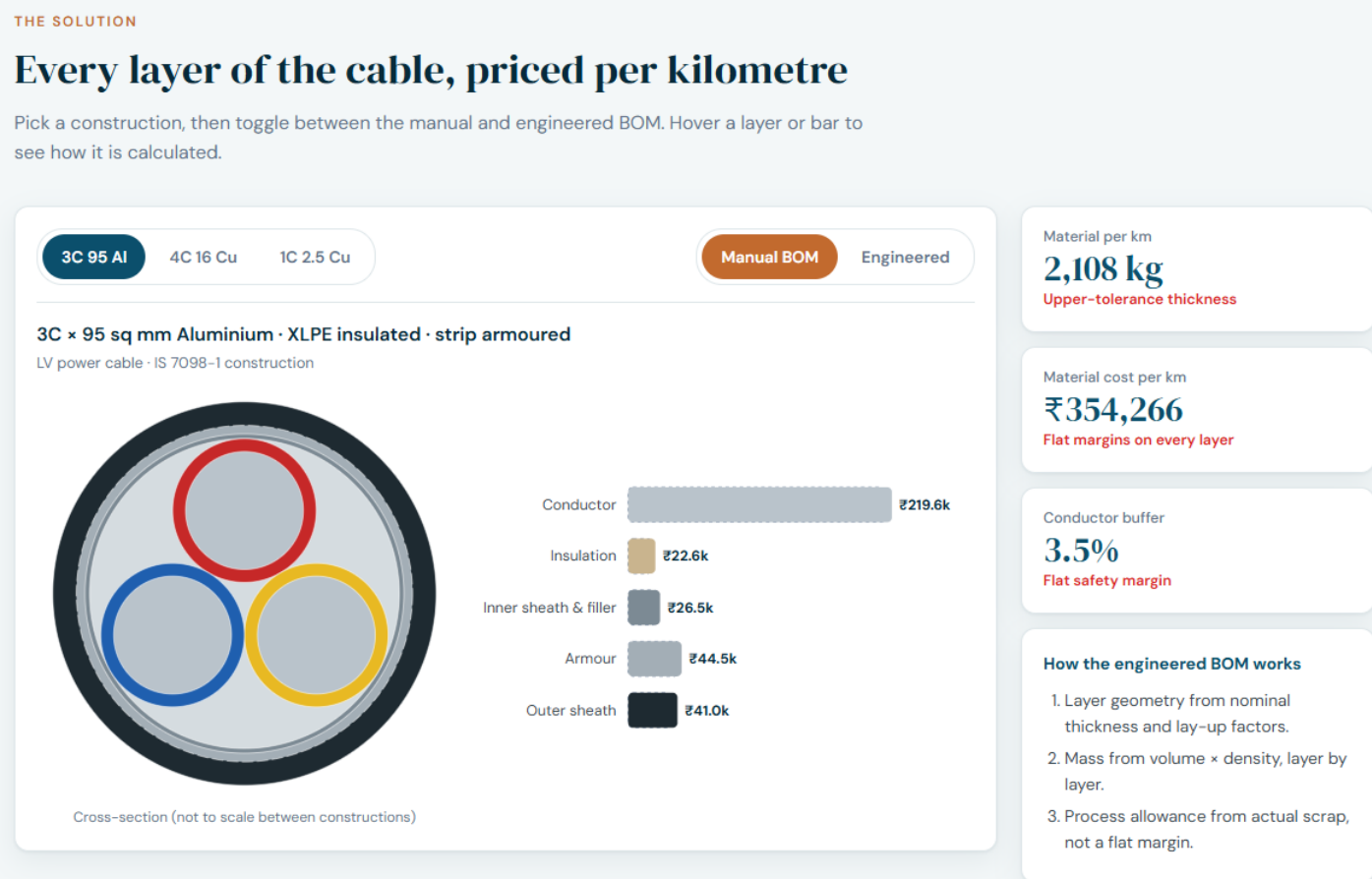
Figures indicative; illustrative territory layout.



Prompt:

Build a single-file HTML page with D3 showing a sales officer's territory map for a cement company: about 64 outlets (dealers and retailers, sized by potential) clustered around 9 towns, with routes from a central base. Add a Before/After toggle. Before shows messy inherited routes that skip some outlets. After groups the outlets into 6 day-wise sectors (Mon–Sat), each with an optimised route. Beside the map, show stat cards for average km per day, outlets covered and potential covered, all updating with the toggle. Add hover tooltips and a day legend that isolates each day's route. Style: navy and gold, DM Serif Display and DM Sans.

Article – Interactive Simulation for Visuals



Prompt:

Build a single-file interactive HTML page that shows a cable's engineered Bill of Materials (BOM).

Content:

- Section header: eyebrow "The Solution", title "Every layer of the cable, priced per kilometre", one-line instruction underneath.

- Three cable constructions to toggle between: 3C 95 sq mm Aluminium XLPE strip armoured, 4C 16 sq mm Copper PVC wire armoured, and 1C 2.5 sq mm Copper FR-PVC house wire.

- A second toggle: Manual BOM vs Engineered BOM.

Visuals:

- Left: an SVG cross-section of the cable, with concentric layers (conductor, insulation, inner sheath, armour, outer sheath) and colour-coded cores.

- Right: horizontal bars showing ₹/km cost per layer, with a dashed ghost bar for the other BOM mode so the difference is visible.

- Side cards: Material per km (kg), Material cost per km (₹), Conductor buffer (%), plus a short "How the engineered BOM works" list.

Logic:

- Calculate each layer's mass from geometry × density, using simple illustrative material prices.

- Manual BOM: upper-tolerance thickness and flat buffers (3.5% on the conductor).

- Engineered BOM: nominal thickness and lower allowances (1% on the conductor).

- In Engineered mode, show the savings in green.

- Hover tooltips on layers and bars show kg/km, ₹/km, how the layer is calculated, and the allowance.

- Animate transitions with D3.

Style: clean consulting look, DM Serif Display + DM Sans, deep teal primary (#0B4F6C), copper accent (#C26A2E), white cards on a light grey-blue background, responsive down to mobile. Add a footnote that the figures are indicative.

Article – Dynamic Dashboard connected to Google Sheets

Prompt:

Create code for google sheets

Sheet url is:

https://docs.google.com/spreadsheets/d/1wJMXo8_jN1BHdFBzTqi5bcNhosShjP9FPzsRUndbbWc/edit?gid=846480193#gid=846480193

It should make an html dashboard showing:

Column chart for manager wise average rating, KPIs for Sum of Sales, Average Rating and Average Sales

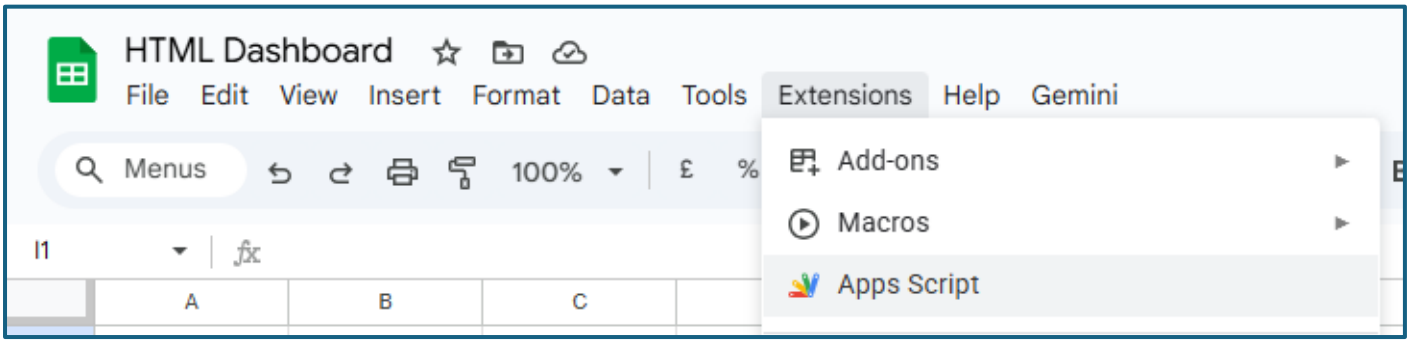
User

	A	B	C	D	E	F
1	**Region**	**Item**	**Sales**	**Manager**	**Date**	**Rating**
2	East	Laptop	$12000	Sarah Lee	01/10/2025	4.5
3	West	Smartphone	$8500	John Doe	02/10/2025	4.2
4	Central	Tablet	$5500	Alex Kim	03/10/2025	3.9
5	South	Monitor	$3200	Maria Garcia	04/10/2025	4.8
6	East	Keyboard	$800	Sarah Lee	05/10/2025	4

create code for google sheets
sheet url is:
https://docs.google.com/spreadsheets/d/1wJMXo8_jN1BHdFBzTqi5bcNhosShjP9FPzsRUndbbWc/edit?gid=846480193#gid=846480193

it should make an html dashboard showing:
column chart for manager wise average rating
KPIs for Sum of Sales, Average Rating and Average Sales

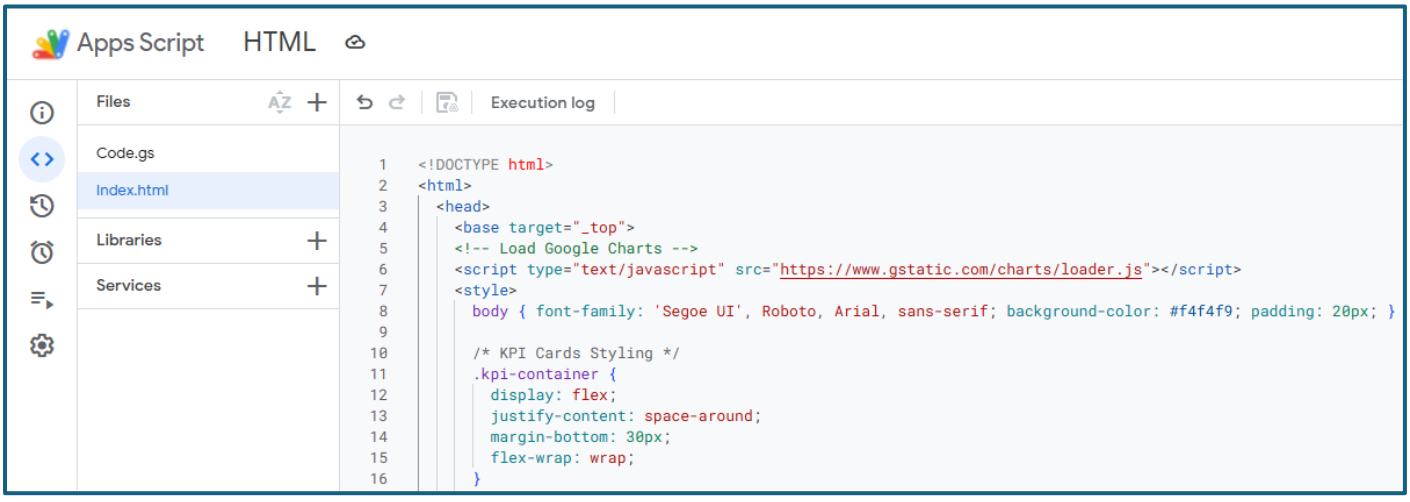
Extensions > Apps Script



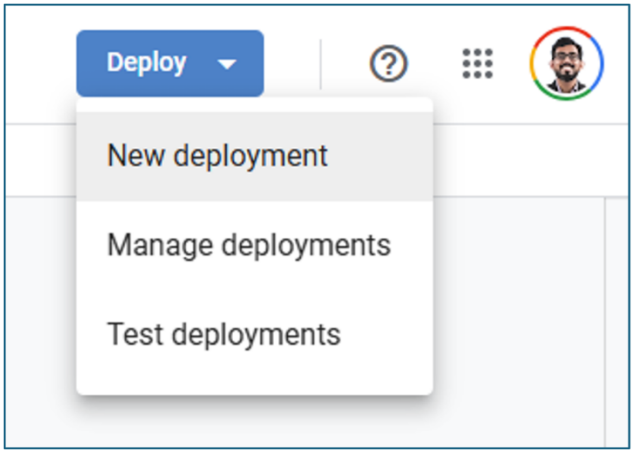
Code.gs

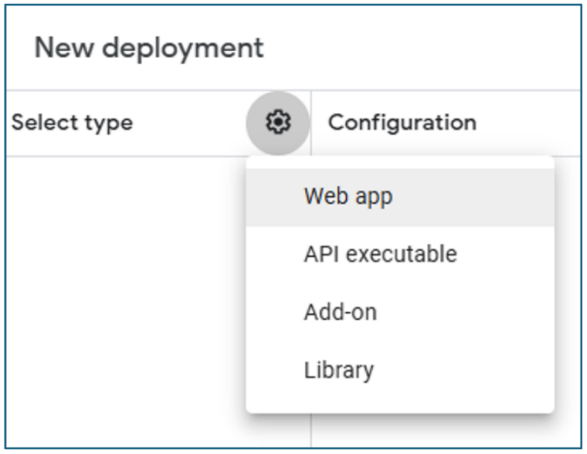


Index.html

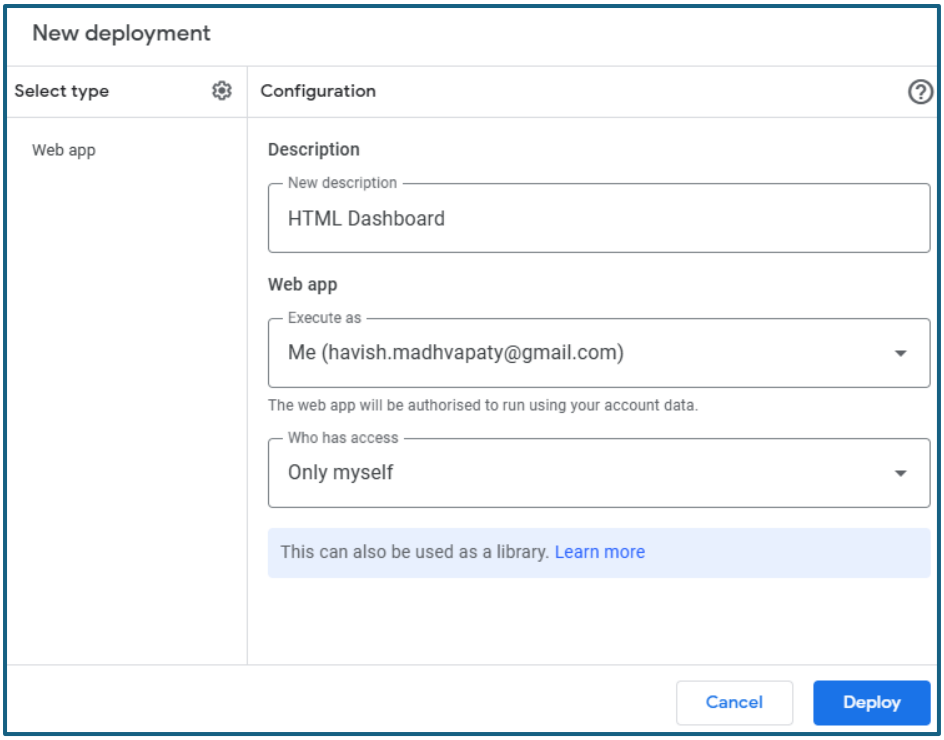


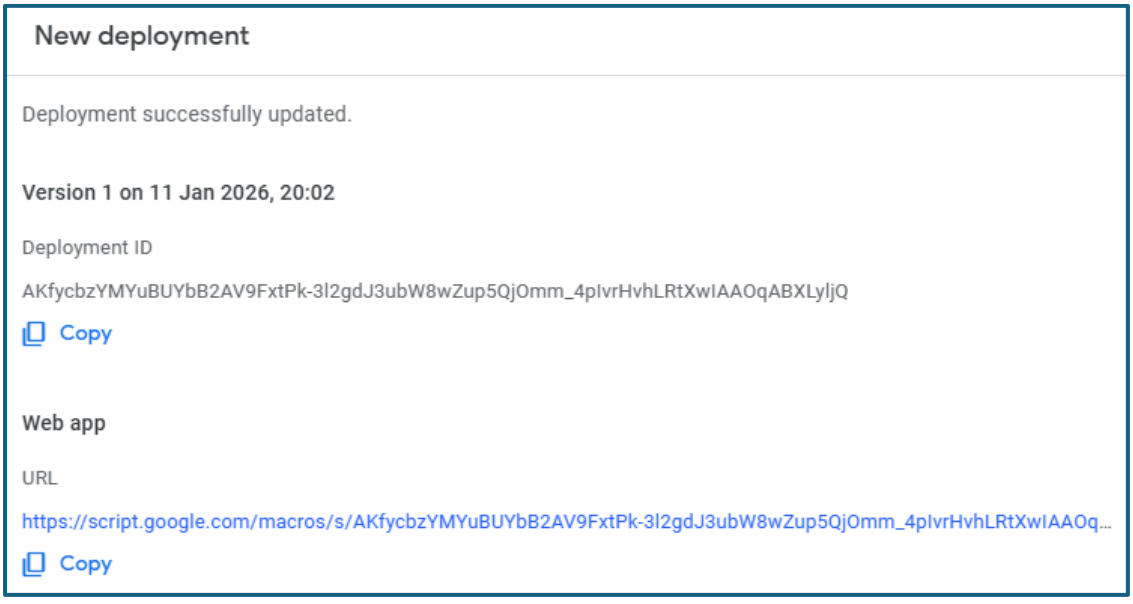
Deploy > New deployment > Web app



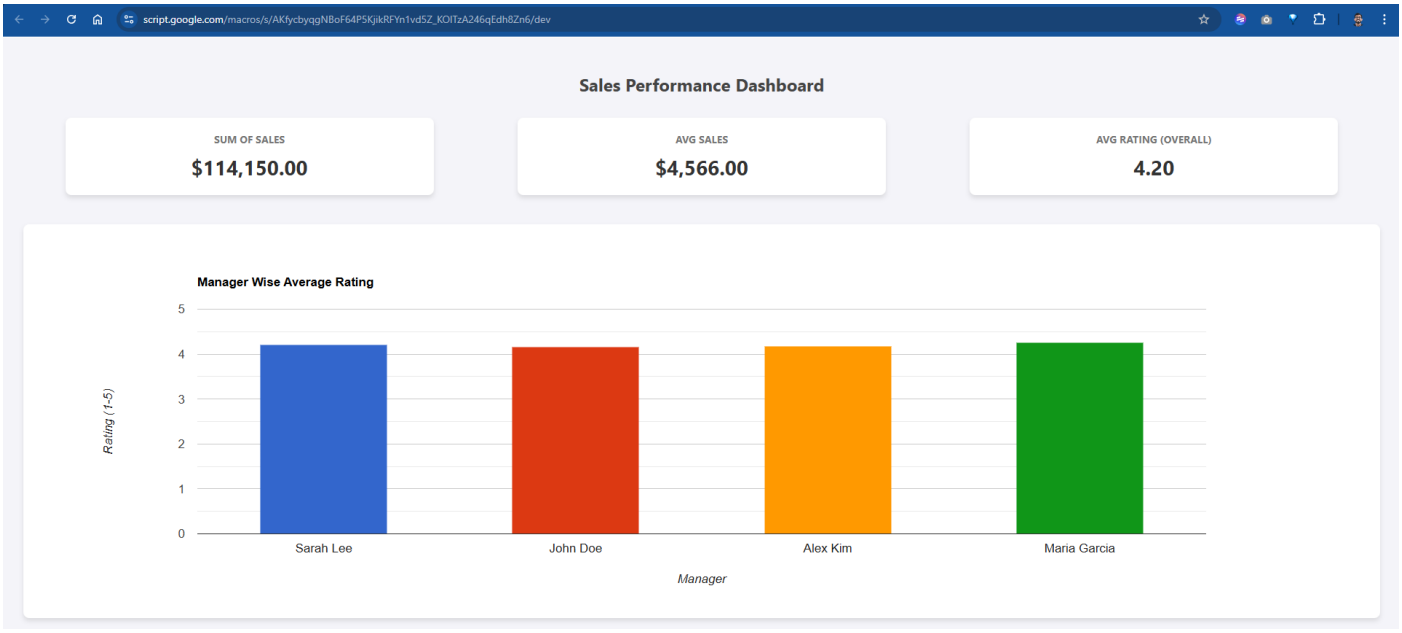


New deployment





Final Dashboard

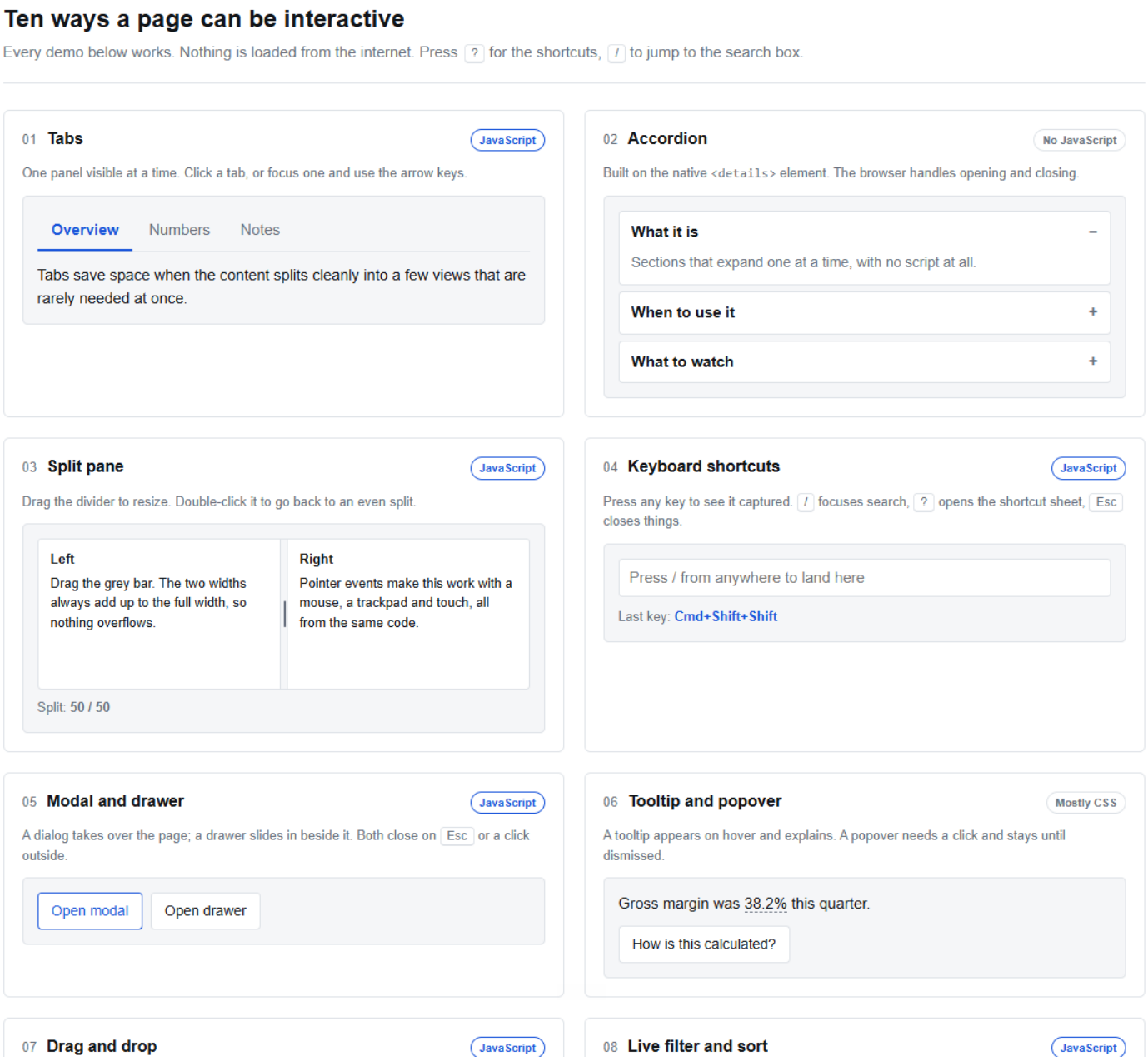


https://docs.google.com/spreadsheets/d/1wJMXo8_jN1BHdFBzTqi5bcNhosShjP9FPzsRUndbbWc/edit?gid=846480193#gid=846480193

https://script.google.com/macros/s/AKfycbyqgNBoF64P5KjikRFYn1vd5Z_KOlTzA246qEdh8Zn6/dev

Part — Content and Interaction

Article – Interact



Prompt:

Build a single-file HTML showcase page titled "Ten ways a page can be interactive". Plain HTML, CSS and vanilla JS only: no frameworks, no libraries, nothing loaded from the internet. Every demo must actually work.

PAGE

- Centred container, max-width ~1180px. Header with H1, a muted intro line ("Every demo below works. Nothing is loaded from the internet.") and kbd hints for ? and /.

- A 2-column grid of cards (1 column under 860px). Each card has: a muted number (01–10), an H2, a pill tag on the right ("JavaScript" in accent colour, or "No JavaScript" / "Mostly CSS" in muted grey), a one-line muted note explaining how to use it, and a soft grey demo box.

- Footer noting that 02 and 06 need little or no script and the rest are a few lines each.

THE TEN DEMOS

01 Tabs: three tabs (Overview, Numbers, Notes) with role="tablist", aria-selected, one panel visible at a time, Left/Right arrow keys move between tabs.

02 Accordion: native <details name="acc"> with three sections, only one open at a time; custom +/− marker; a small JS fallback for browsers that don't support the name attribute.

03 Split pane: two panes with a draggable divider using pointer events (mouse, trackpad, touch), clamped 15–85%, double-click resets to 50/50, arrow keys nudge it when focused, live "Split: 50 / 50" readout.

04 Keyboard shortcuts: a search box; "/" focuses it from anywhere (with a toast), "?" toggles a shortcut-sheet overlay, Esc closes any open overlay; show the last key pressed (with Ctrl/Cmd/Shift prefixes). Ignore shortcuts while typing in inputs.

05 Modal and drawer: "Open modal" uses the native <dialog> with showModal (backdrop click closes); "Open drawer" slides a 320px panel in from the right with a scrim. Both close on Esc or a click outside.

06 Tooltip and popover: a CSS-only hover/focus tooltip on "38.2%" explaining gross margin; a click-to-open popover button ("How is this calculated?") that closes on any outside click.

07 Drag and drop: a 4-item task list (Collect requirements, Draft the model, Review with finance, Publish the dashboard) reorderable with native drag-and-drop, a blue top edge showing the drop position, and a live "Order: 1. … · 2. …" readout.

08 Live filter and sort: a table of Indian cities (City, Region, Units, Revenue): Mumbai/West/18,240/96, Delhi NCR/North/16,980/88, Bengaluru/South/14,120/74, Hyderabad/South/10,450/52, Chennai/South/9,380/48, Kolkata/East/7,920/40, Pune/West/7,140/36. Type-to-filter on city or region, click headers to sort (click again to reverse) with ↑/↓ arrows and aria-sort, an empty-state row, and a "X of 7 rows, sorted by …" line. Default sort: revenue descending.

09 Inline edit: label/value pairs (Owner: Asha Menon, Target: 1,600 Cr, Review date: 24 October). Click a value to turn it into an input; Enter or blur saves (with a toast), Esc cancels; empty input reverts.

10 Form validation: work email and password fields with live messages (green/red), a 5-step password strength meter that changes colour, and a submit button disabled until both are valid (email regex; password ≥8 chars plus enough variety). On submit, prevent sending and show "Submitted, nothing left this page." plus a toast.

SHARED

- A bottom-centre toast that fades in and out.

- Escape all user text before inserting into HTML.

STYLE

- Clean, neutral: Helvetica/Arial 15px, white cards, 1px light borders, 4–6px radius, accent blue #1a56db, green for success, red for errors, styled <kbd> keys, tabular numbers in the table.

- Colours as CSS variables on :root, with a full dark theme under prefers-color-scheme (guarded by :root:not([data-theme="light"])) and again under :root[data-theme="dark"].

- Visible :focus-visible outlines everywhere; good keyboard support throughout.

Deliver as one self-contained .html file.

Simplified Prompt:

1. Tabs (with arrow-key support)

2. Accordion (native <details>, one open at a time)

3. Resizable split pane (drag the divider)

4. Keyboard shortcuts (/ focuses search, ? opens a shortcut list, Esc closes things)

5. Modal and slide-in side drawer

6. Hover tooltip and click popover

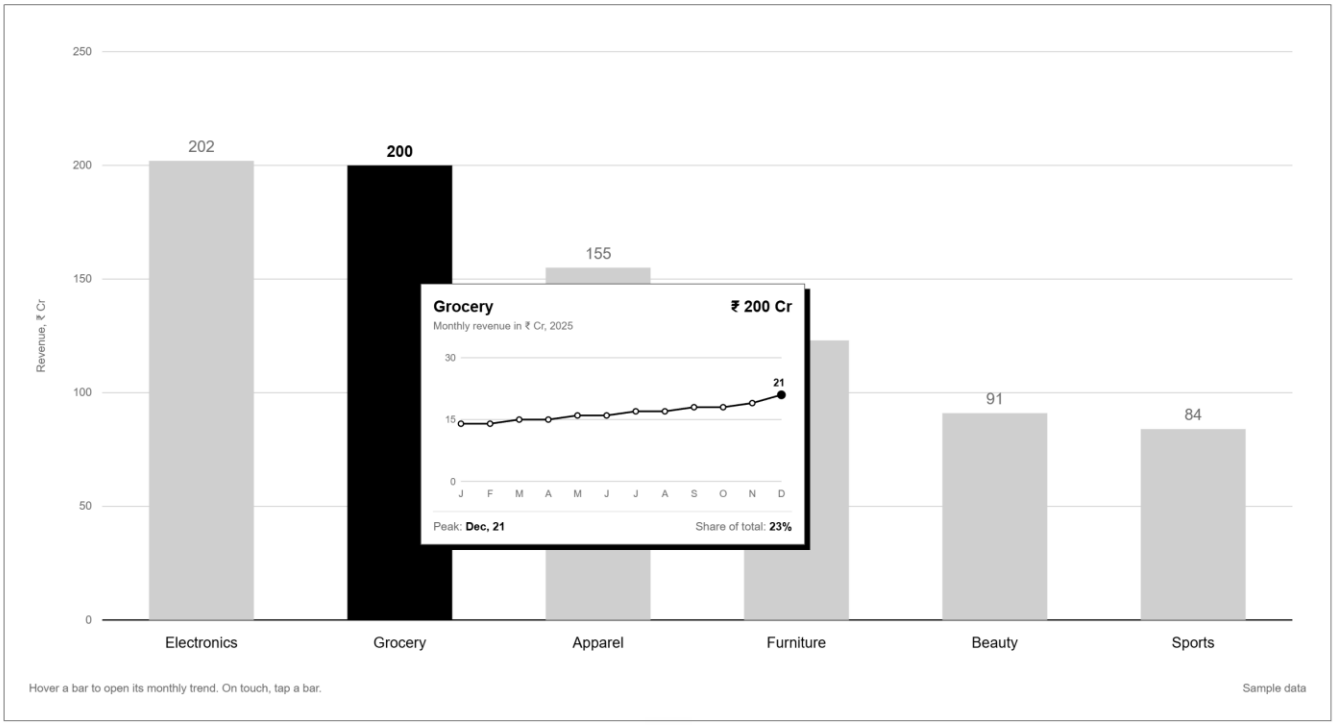
7. Drag-and-drop list to reorder tasks

8. Table of Indian cities with live filter and click-to-sort columns

9. Click-to-edit fields (Enter saves, Esc cancels)

10. Signup form with live validation and a password strength meter

Article – Tooltip



Prompt:

Build a single HTML file: a column chart where hovering a bar shows a tooltip containing a second, smaller chart (a drill-through tooltip).

Look and layout

- Fixed 1600 x 900 (16:9) canvas that scales to fit any window, centred.

- Black and white only: black bars, thin black borders, light grey gridlines, Helvetica. Supports dark mode (colours invert).

- One full-size card holding the chart, with a small footer: "Hover a bar to open its monthly trend. On touch, tap a bar." on the left and "Sample data" on the right.

Data (sample, ₹ Cr, 2025)

- Six categories: Electronics, Furniture, Apparel, Grocery, Beauty, Sports.

- Each has 12 monthly values (Jan to Dec) with a festive bump in Oct to Dec.

- Each bar's total = sum of its 12 months. Sort bars highest to lowest.

Main chart

- Vertical column chart drawn as inline SVG.
- Y axis in steps of 50, labelled "Revenue, ₹ Cr".
- Category name under each bar, total in bold above each bar.
- When a bar is active, it stays black and the others fade to light grey.

Tooltip

- Opens next to the cursor and follows it; flips left if it would run off the right edge.
- White box, black border, solid black offset shadow.
- Top line: category name on the left, "₹ [total] Cr" on the right.
- Subtitle: "Monthly revenue in ₹ Cr, 2025".
- Small line chart of the 12 months with dots, month initials below, 3 gridlines.
- Peak month gets a filled dot with its value labelled above.
- Footer: "Peak: [Month], [value]" and "Share of total: [x]%".

Interaction

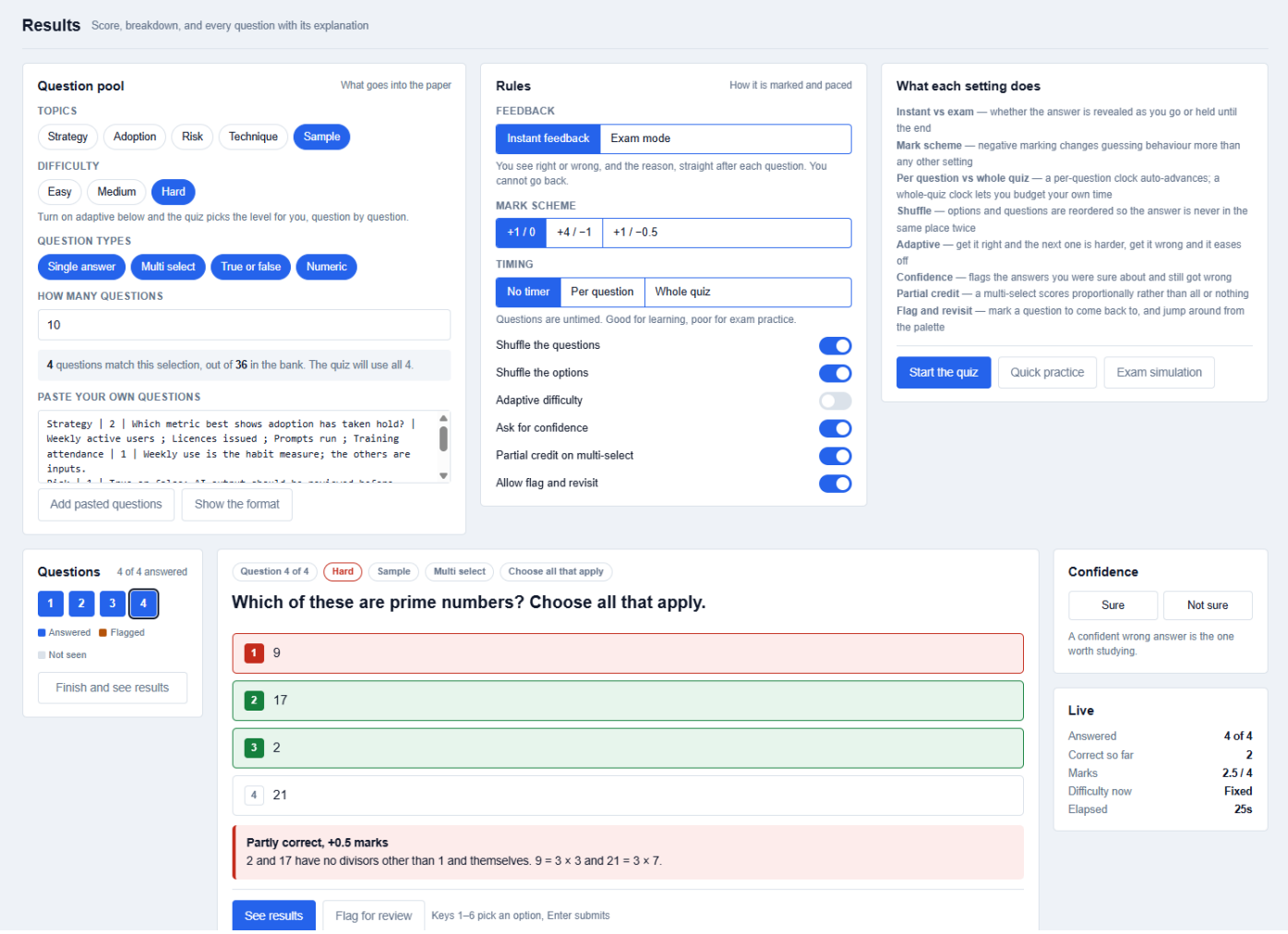
- Mouse: hover to show, move away to hide.
- Touch: tap a bar to pin the tooltip, tap elsewhere to close.
- Keyboard: bars are tabbable; focusing one opens its tooltip.

No external libraries.

Simplified Prompt:

Build a single HTML file: a column chart of revenue by category (Electronics, Furniture, Apparel, Grocery, Beauty, Sports, sample ₹ Cr data). Hovering a bar shows a tooltip with a small line chart of that category's 12-month trend, the peak month and its share of total. Fixed 16:9 canvas that scales to fit, black and white style, dark mode, no external libraries.

Article – MCQs



Prompt:

Build a single HTML page that works as a multiple-choice quiz engine, using only HTML, CSS and plain JavaScript (no libraries, nothing loaded from the internet). It has three screens:

1. SETUP

- Choose topics, difficulty (Easy / Medium / Hard), question types (single answer, multi-select, true/false, numeric) and number of questions; show how many questions match.

- Rules: instant feedback or exam mode, mark scheme (+1/0, +4/−1, +1/−0.5), timer (none, per question, or whole quiz).

- On/off switches: shuffle questions, shuffle options, adaptive difficulty, ask for confidence (Sure / Not sure), partial credit on multi-select, flag and revisit.

- Two presets: "Quick practice" and "Exam simulation".

- A box to paste extra questions in a simple one-line-per-question format.

2. QUIZ

- Left: numbered question palette showing answered, flagged and current.

- Centre: question with difficulty/topic/type tags, options as large clickable buttons (keys 1–6 to pick, Enter to submit), an explanation after answering in instant mode, and Previous / Next / Skip / Flag buttons.

- Right: countdown timer with a progress bar, confidence buttons, and a live score panel.

3. RESULTS

- Score ring and percentage, marks, time taken.

- Accuracy bars by topic, difficulty and question type, plus a warning for answers marked "Sure" that were wrong.

- Full review of every question with your answer, correct answer and explanation; filter to wrong only.

- Buttons: Retry the wrong ones, New quiz, Download CSV, Copy summary.

Keep the question bank as a plain editable list at the top of the script. Fill it with about 20 questions on "GenAI for leaders" (strategy, adoption, risk, prompting technique) and a few easy general-knowledge sample questions, including one that reads a small bar chart.

Clean, minimal look with a blue accent, green/red for right/wrong, dark mode support, works on mobile. Deliver as one .html file.

Article – Maps

https://leafletjs.com/

Prompt:

Build a single HTML page with an interactive map of India using Leaflet (load it from cdnjs), fitted to a 16:9 frame that scales to any window.

- Plot 8 cities as circles sized by revenue (₹ Cr, sample data): Mumbai 96, Delhi NCR 88, Bengaluru 74, Hyderabad 52, Chennai 48, Ahmedabad 44, Kolkata 40, Pune 36.

- Hovering a circle shows a tooltip with the city name and "₹ X Cr".

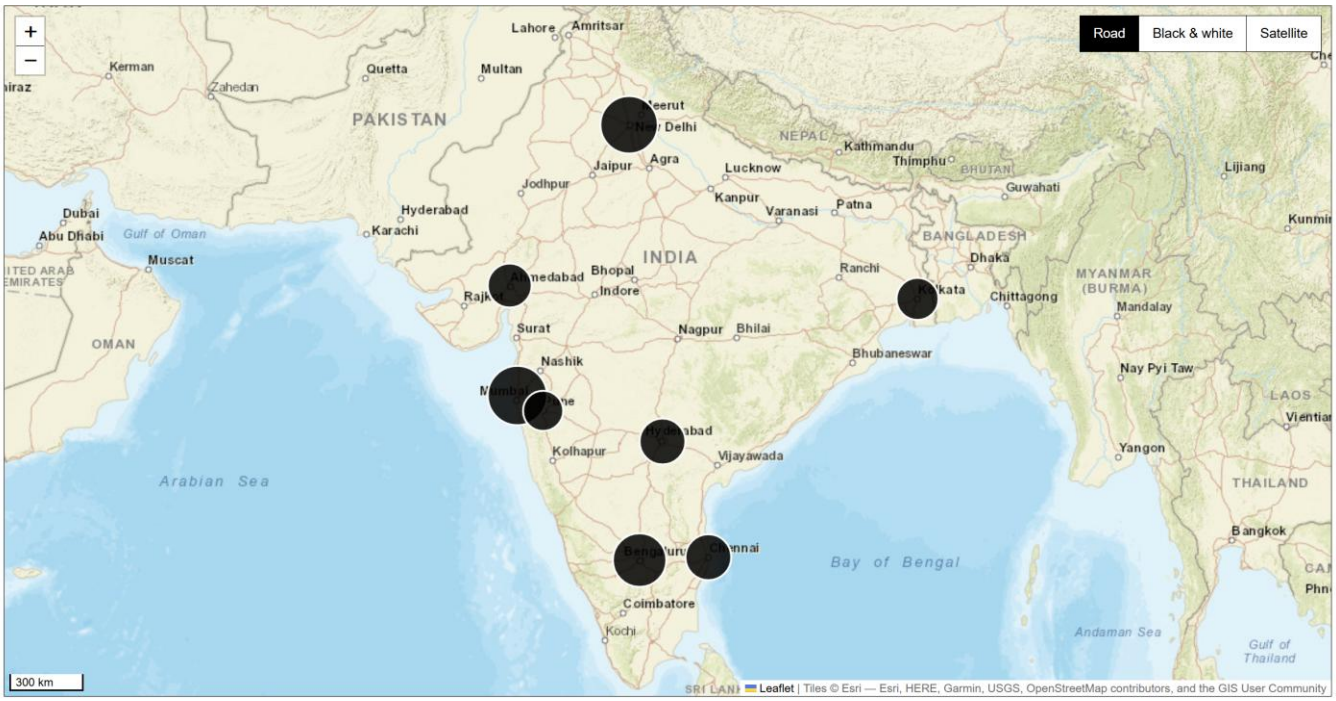
- Add a switch on the map for three map types: Road, Black & white (default), and Satellite with place labels. Use free Esri tiles, no API key.

- Zoom buttons, a metric scale bar, and auto-zoom to fit all cities.

- A short footer line: "Hover a circle to see revenue by city. Circle size shows revenue." and "Sample data" on the right.

- Show a friendly message if the map fails to load.

Style: stark black-and-white look (black circles, square tooltip with a hard shadow, black-bordered controls), with dark mode support. Deliver as one .html file.



Hover a circle to see revenue by city. Circle size shows revenue. Use the switch on the map to change the map type.

Sample data

Article – Globe

Prompt:

Build a single HTML page with an animated dotted globe showing where Havish M Consulting (HMC) works. Plain HTML, CSS and JavaScript on a <canvas>; no 3D libraries. Font: DM Sans from Google Fonts. Transparent background so it can be embedded anywhere.

- Draw the Earth as a globe made of small blue dots for land, with a soft white-to-light-blue shading and a faint glow around the edge.

- Mark four locations with pulsing pins and a small label: India (the hub), UAE, United Kingdom, United States. Draw curved arcs from India to the other three, with a small dot travelling along each arc.

- The globe slowly spins on its own and tours each country in turn. It can be dragged to rotate, and clicking a pin focuses that country.

- Below the globe, a row of pill buttons for the four countries; the active one is filled navy.

- Clicking India zooms smoothly into India, shows a more detailed dot map, and reveals a second row of pills for 15 states (Delhi, Haryana, Punjab, Rajasthan, Uttar Pradesh, Madhya Pradesh, Gujarat, Maharashtra, Karnataka, Kerala, Tamil Nadu, Telangana, Odisha, West Bengal, Meghalaya). States pop in one by one, auto-cycle with a highlight, and can be hovered or clicked. A "Back to world view" button zooms back out.

- Respect reduced-motion settings, keep it sharp on high-resolution screens, and resize to fit its container.

Colours: HMC blues (#1a5fa8 main, #0f3460 deep navy, #d2e3f5 light).

Deliver as one .html file.



India | UAE | United Kingdom | United States

Article – Motion Video

A	B	C	D	E	F	G	H	I	J
Order ID	Order Date	Sales Rep	Region	Dealer / Customer	Product Category	Product	Qty (Coils/Units)	Unit Price (₹)	Sales Value (₹)
RRK-1001	01-Oct-2026	Amit Sharma	North	Sharma Electricals, Delhi	House Wires	FR PVC Wire 1.5 sq mm (90m)	120	₹1,450	₹1,74,000
RRK-1002	02-Oct-2026	Priya Nair	South	Nair Traders, Kochi	House Wires	FR-LSH Wire 2.5 sq mm (90m)	85	₹2,350	₹1,99,750
RRK-1003	03-Oct-2026	Rahul Patil	West	Patil Agencies, Pune	Power Cables	LT XLPE Armoured 4C x 16 sq mm (100m)	6	₹48,500	₹2,91,000
RRK-1004	05-Oct-2026	Sneha Das	East	Das Electric Co., Kolkata	Flexible Cables	Multicore Flexible 3C x 1.5 sq mm (100m)	40	₹3,900	₹1,56,000
RRK-1005	07-Oct-2026	Vikram Singh	North	Singh Power Solutions, Jaipur	Solar Cables	DC Solar Cable 4 sq mm (100m)	55	₹4,200	₹2,31,000
RRK-1006	09-Oct-2026	Karthik Reddy	South	Reddy Enterprises, Hyderabad	House Wires	FR PVC Wire 4 sq mm (90m)	70	₹3,650	₹2,55,500
RRK-1007	12-Oct-2026	Neha Joshi	West	Joshi Hardware, Ahmedabad	Fans	Ceiling Fan 1200mm	150	₹2,450	₹3,67,500
RRK-1008	14-Oct-2026	Arjun Mehta	Central	Mehta Electricals, Indore	Switches	Modular Switch 6A (Box of 10)	200	₹420	₹84,000
RRK-1009	16-Oct-2026	Sneha Das	East	Bose Infra Projects, Bhubaneswar	Power Cables	LT Aluminium Armoured 3.5C x 95 sq mm (100m)	4	₹72,000	₹2,88,000
RRK-1010	18-Oct-2026	Amit Sharma	North	Gupta Builders, Lucknow	House Wires	FR-LSH Wire 6 sq mm (90m)	60	₹5,400	₹3,24,000



Prompt:

I will share an Excel/sales data file. Analyse it and turn the key insights into a 15-second motion video.

STEP 1 – Understand the data

- Identify the columns (dates, regions, products/categories, people, quantities, values) and the period the data covers.
- Validate totals (e.g. Qty × Price = Value) and flag any errors or blanks before building anything.

STEP 2 – Find the story

- Headline metric: total value for the period.
- 2–3 KPIs: e.g. number of transactions, average value, largest single item.
- Breakdowns: by the most meaningful dimensions available (region, category, salesperson, customer, time).
- One clear takeaway: the biggest driver and the biggest gap or risk.
- Use only what the data supports; do not invent figures.

STEP 3 – Build the video (15 seconds, 16:9)
Suggested structure (adapt to what the data shows):
0–2s Title + period, headline total counts up
2–5s KPI cards pop in
5–8s Breakdown 1 (animated bar chart)
8–11s Breakdown 2 (animated donut/pie)
11–13.5s Top performers (ranked list or podium)
13.5–15s One-line key takeaway, fade out

STYLE

- Clean corporate look; pick a colour palette that matches the brand or header colours in the file, otherwise use navy, white and one accent.
- Bold sans-serif numbers, smooth count-up and grow animations.
- Number format to match the data's currency and locale (e.g. ₹ in Lakhs for Indian data).
- On-screen text only, no voiceover. Every number must stay readable for at least 1 second. Max 6 scenes.

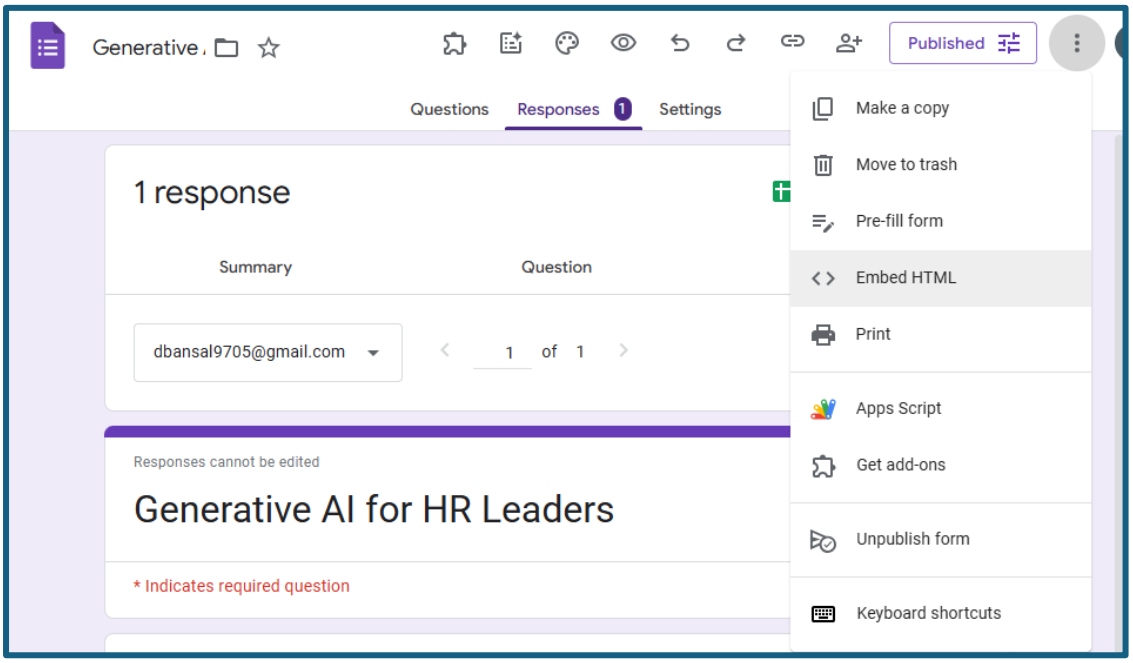
Before building, show me a one-line summary of the insights you plan to use for each scene so I can confirm.

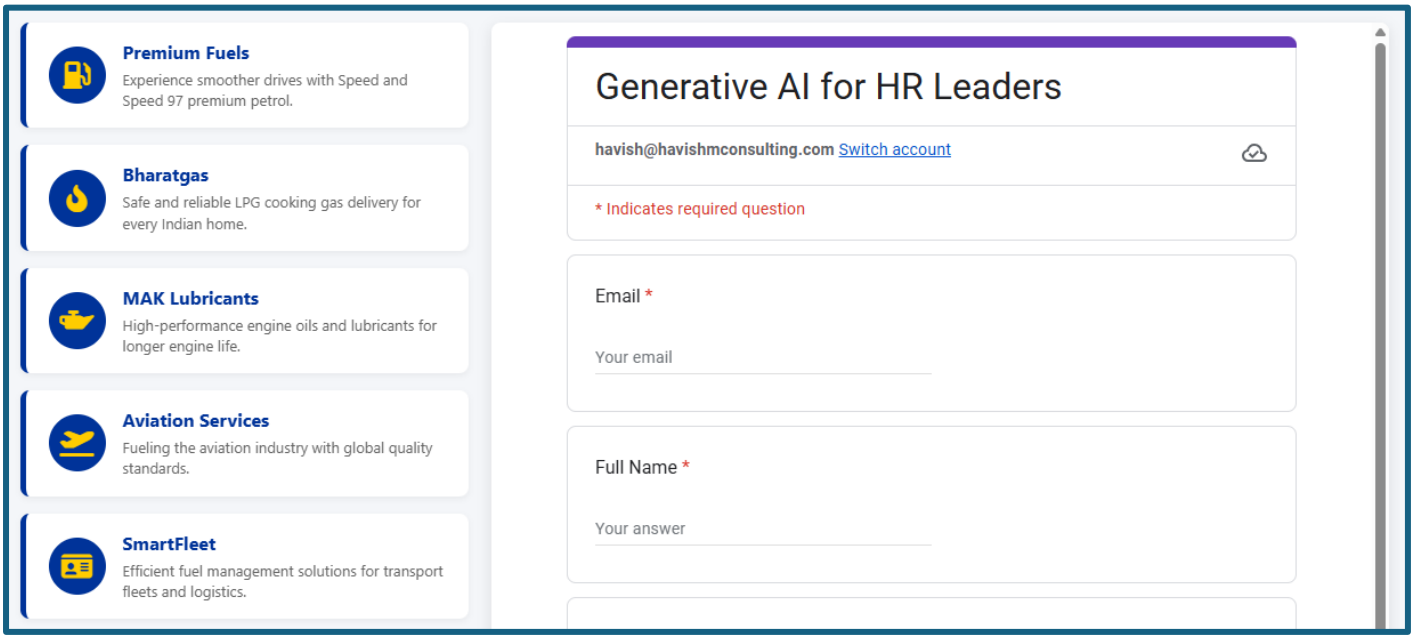
Article – Embed HTML

Prompt:

Embed this in an HTML page

<iframe src="https://docs.google.com/forms/d/e/1FAIpQLSdH0WfZzrbAXCh8xuR7JVz-2NaJk9K4OZRaOy63xqrH3XDyKQ/viewform?embedded=true" width="640" height="1229" frameborder="0" marginheight="0" marginwidth="0">Loading...</iframe>





Article – Bookmarks

Prompt:

Build a single HTML sales dashboard for FY 2025–26 (Apr 2025 to Mar 2026) that shows off Power BI-style bookmarks and a timeline slicer. Plain HTML, CSS and JavaScript only; no libraries.

DATA

Generate realistic sample data in the script: 4 regions (North, South, East, West) × 4 categories (Beverages, Snacks, Dairy, Personal care) × 3 channels (Modern trade, General trade, Ecommerce) × 12 months, with sales in ₹ Cr and units. Add some seasonality, faster growth in West, and a mid-year dip in East.

LAYOUT (top to bottom)

1. Bookmarks card: clickable pill chips for saved views, plus a text box and "Save current view" button. Include four built-in bookmarks: "Full year, everything", "West · H2", "Festive quarter" (Sep–Dec), "Ecommerce · Q4". User-saved bookmarks get a small × to delete. Show the active bookmark highlighted and a line describing what it shows; if filters are changed by hand, say so and prompt to save.

2. Filters card: dropdowns for Region, Category and Channel, a "Clear filters" button, and a draggable two-handle timeline slider across the 12 months (snaps to months, arrow keys work, shows the selected range and month count).

3. Four KPI cards: Net sales, Units sold, Average realisation per unit, Best month.

4. Two charts side by side: monthly sales column chart (for the selected months) and a horizontal bar chart by category.

5. Region table: net sales, units, average realisation and share, with a total row.

Everything updates instantly when filters, the timeline or a bookmark change. Show a small toast for actions like "Saved" or "Filters cleared".

Clean card-based look with rounded corners, a blue accent, Indian number formatting, dark mode support, and a layout that stacks on mobile. Deliver as one .html file.

Sales dashboard FY 2025–26

BOOKMARKS

Full year, everything | West · H2 | Festive quarter | Ecommerce · Q4 | CEO ×

Name this view | Save current view

Showing "CEO" — South · Snacks · Modern trade · Jun 2025 – Oct 2025

FILTERS

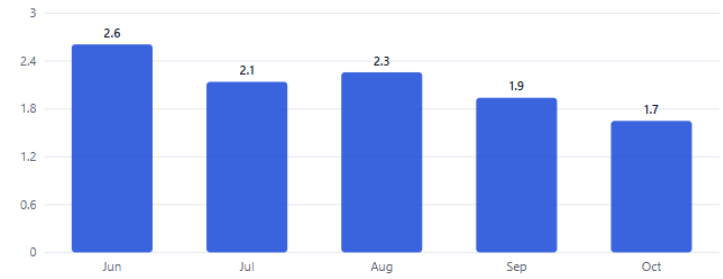
REGION: South | CATEGORY: Snacks | CHANNEL: Modern trade | Clear filters

Jun 2025 – Oct 2025 5 months

Apr May **Jun** **Jul** **Aug** **Sep** **Oct** Nov Dec Jan Feb Mar

NET SALES	UNITS SOLD	AVG REALISATION	BEST MONTH
₹10.6 Cr	0.3 Cr	₹40.0	Jun 2025
5 months in view	Snacks	per unit	₹2.6 Cr

MONTHLY SALES



BY CATEGORY



REGION DETAIL

REGION	NET SALES	UNITS	AVG REALISATION	SHARE

Article – PDF Reader and Editor

Prompt:

Build a single HTML page called "PDF reader and editor" that works entirely in the browser. Nothing is uploaded. Use pdf.js to show pages and pdf-lib to edit them, with both libraries embedded in the file so it works offline.

TOP BAR

- An "Open a PDF" button, an "Undo" button, a "Save PDF" button, and a line that shows the file name and page count.

- Two tabs: Read and Edit pages.

READ TAB

- A strip of page thumbnails on the left. Click one to jump to that page.

- The current page shown large in the middle, kept sharp on high-resolution screens.

- A bottom bar with First, Previous, a page-number box ("of N"), Next and Last. Then zoom − and +, the zoom %, Fit width and Fit page.

- Left/Right arrow keys and PageUp/PageDown move between pages.

EDIT PAGES TAB

- A grid of page thumbnails. Click a card or its checkbox to select it. Add Select all and Select none, plus a "3 selected" count.

- Drag a card to reorder pages, or use the ← → buttons on each card.

- Delete selected pages, Extract selected pages (downloads them as a new PDF), and Merge another PDF (adds its pages at the end).

- Undo reverses any edit. Save PDF downloads the edited file.

EXTRAS

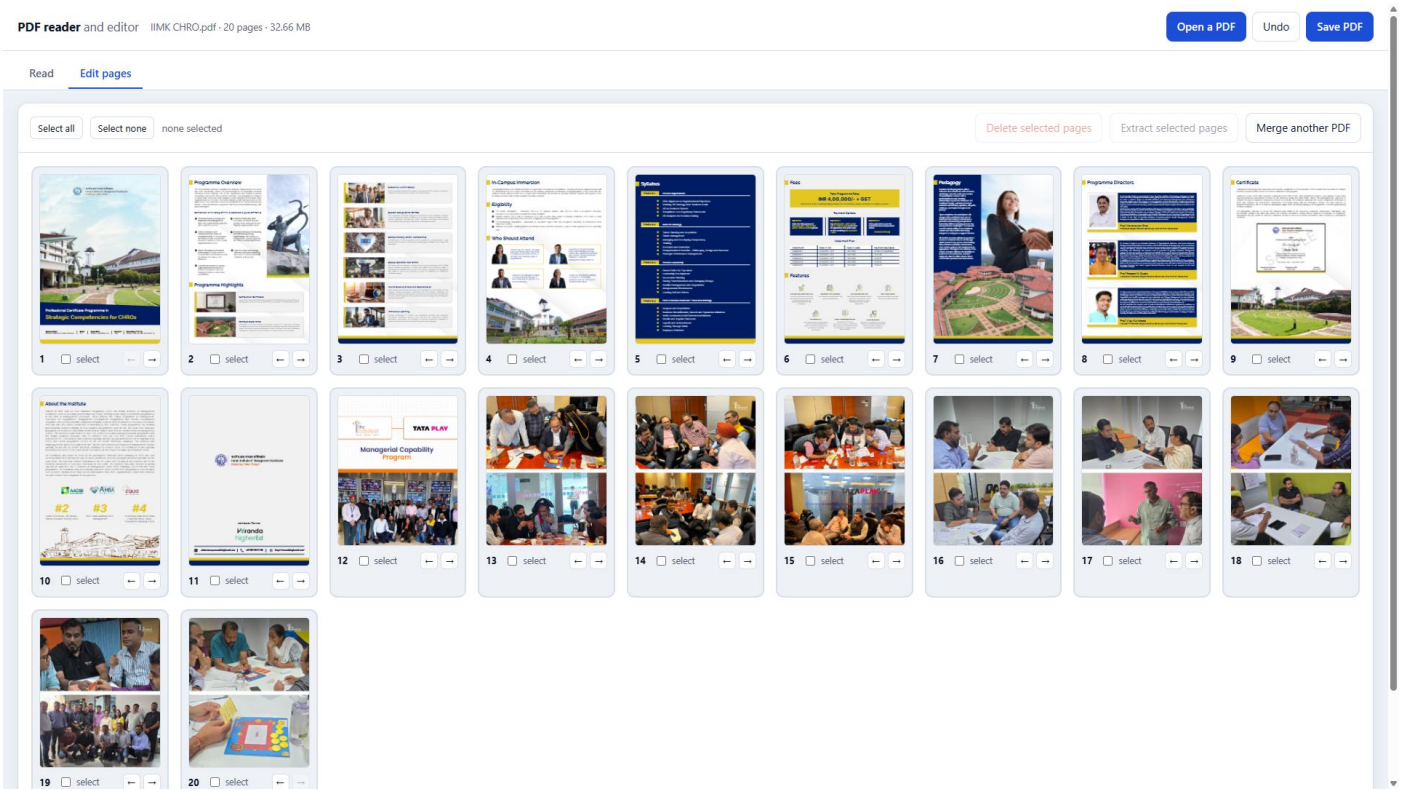
- A "Working..." overlay during slow steps.

- A small toast for feedback, such as "Opened report.pdf, 12 pages" or "Page 3 is now page 5".

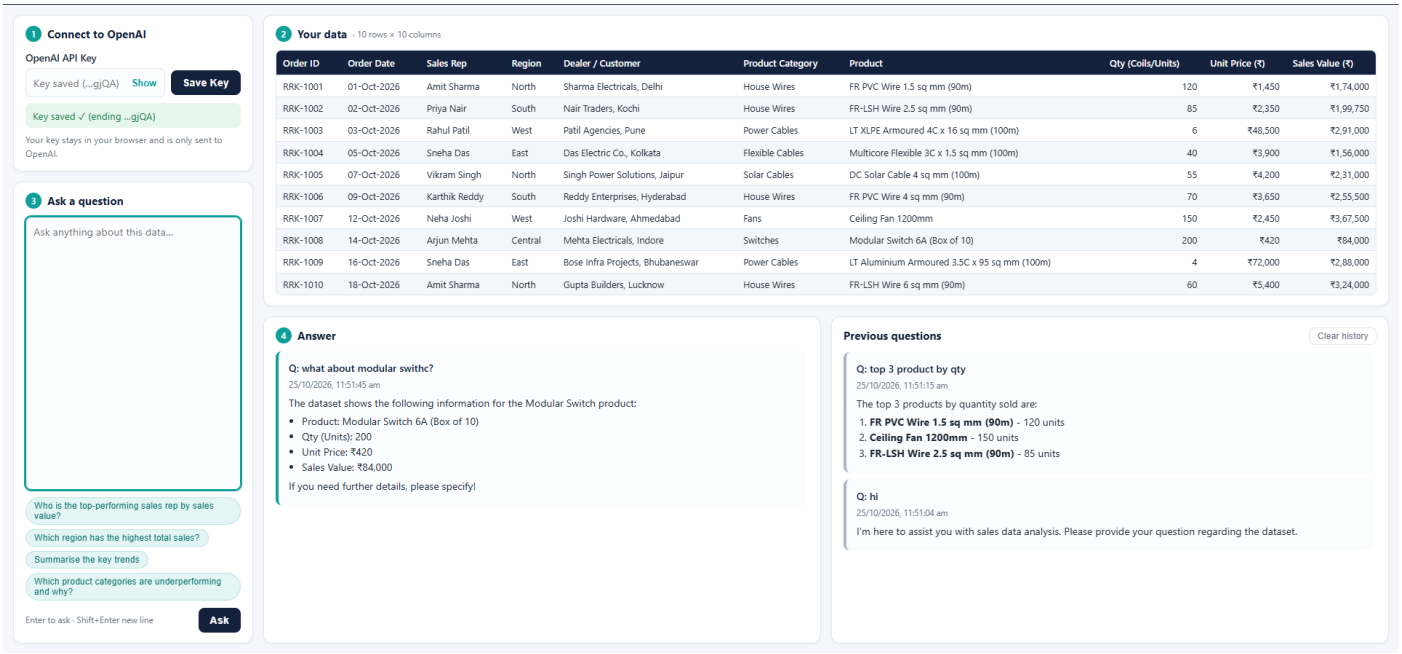
- Friendly messages for password-protected or broken files.

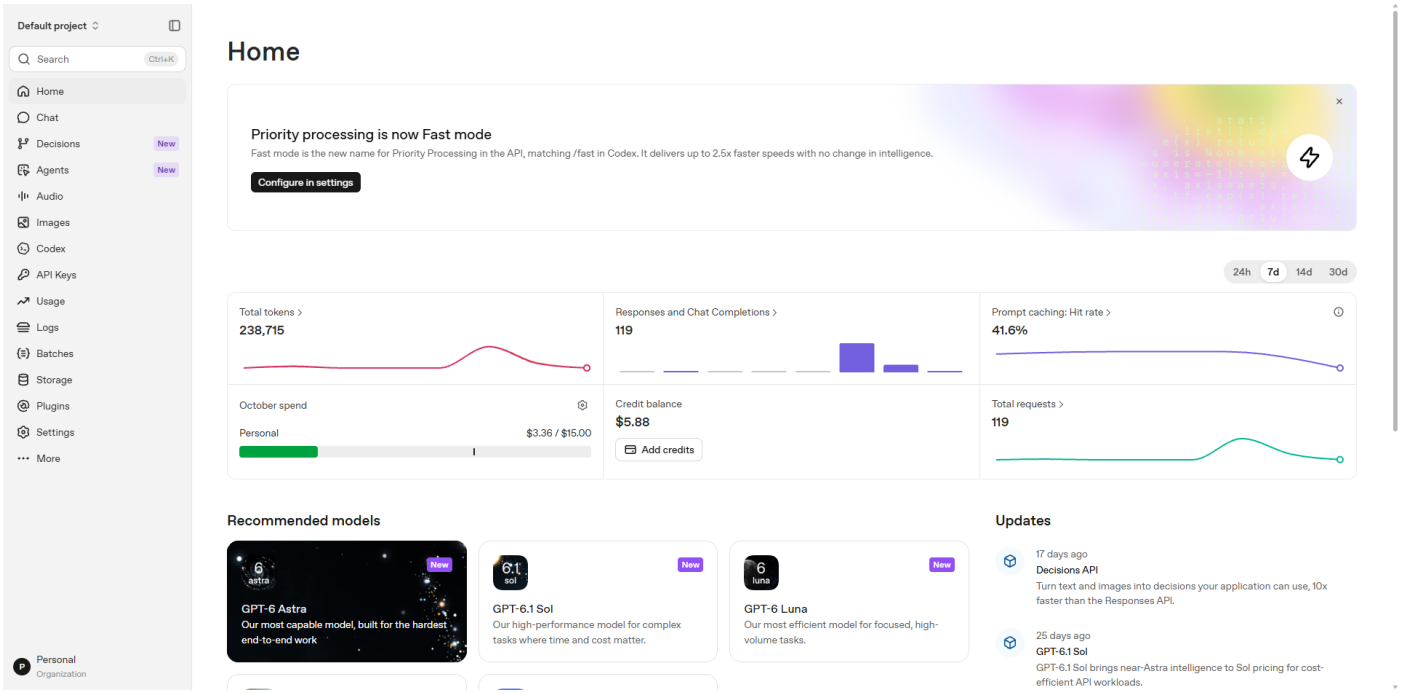
- An empty state that says "Nothing open yet. The file is read inside this browser and never uploaded."

Keep the look clean and minimal, with a blue accent and a red Delete button. Support dark mode. Deliver as one .html file.



Article – API





Prompt:

Build me a single, self-contained HTML file (HTML, CSS and JavaScript all in one file) that I can open by double-clicking it in my browser. I am not a coder, so keep everything simple and make the page work without installing anything.

PURPOSE

A small "Ask Your Data" tool. I paste my OpenAI API key, see my employee performance dataset on screen, type any question about it in plain English, and get an answer from OpenAI.

THE DATASET

I have already provided an employee performance tracking dataset (10 rows, 10 columns), pasted below / attached. Use it exactly as given. Do not invent, modify or add any data. Embed it inside the HTML file so the page works on its own.

PAGE LAYOUT (top to bottom)

1. Header: title "Employee Performance – Ask Your Data" and a one-line subtitle explaining what the page does.
2. API Key section:
    - A password-style input box labelled "OpenAI API Key" (characters hidden, with a small Show/Hide toggle).
    - A "Save Key" button. Show a green "Key saved" message once entered.
    - Keep the key only in the browser's memory for this session. Never display it in full or send it anywhere except OpenAI.
    - A short note: "Your key stays in your browser and is only sent to OpenAI."
3. Dataset section:
    - Show the full dataset as a clean, readable table with column headers.
    - Alternate row shading, header row highlighted, table scrolls sideways on small screens.
4. Question section:
    - A large text box: "Ask anything about this data..."
    - An "Ask" button (pressing Enter should also work).
    - 3–4 clickable example question chips that fill the box when clicked, e.g. "Who is the top performer?", "Which department has the highest average rating?", "Summarise the key trends", "Who might need a performance improvement plan and why?"
5. Answer section:
    - Show a "Thinking..." loading indicator while waiting.
    - Display the answer in a neat box with proper line breaks, bullet points and bold text where relevant.
    - Keep a running history of previous questions and answers below (newest first), with a "Clear history" button.

HOW THE ANSWER IS GENERATED

- Call the OpenAI Chat Completions API directly from the browser using the saved key.
- Use the model "gpt-4o-mini" (put the model name in one clearly marked place near the top of the script so I can change it later).
- Send the full dataset with every question, along with these instructions to the AI:
  "You are an HR data analyst. Answer only using the dataset provided. Show the numbers behind your answer. If the question cannot be answered from the data, say so clearly. Keep answers concise and business-friendly."
- Do the analysis through the API only; do not hard-code any answers.

ERROR HANDLING (in plain, friendly language)

- No key entered → "Please enter your OpenAI API key first."
- Empty question → "Please type a question."
- Invalid key, no credit or network problem → show a simple explanation of what went wrong and what to check.

DESIGN

- Clean, modern, professional look suited to a business audience.
- Calm colour palette (navy and white with one accent colour), clear fonts, generous spacing.
- Works well on both laptop and mobile screens.

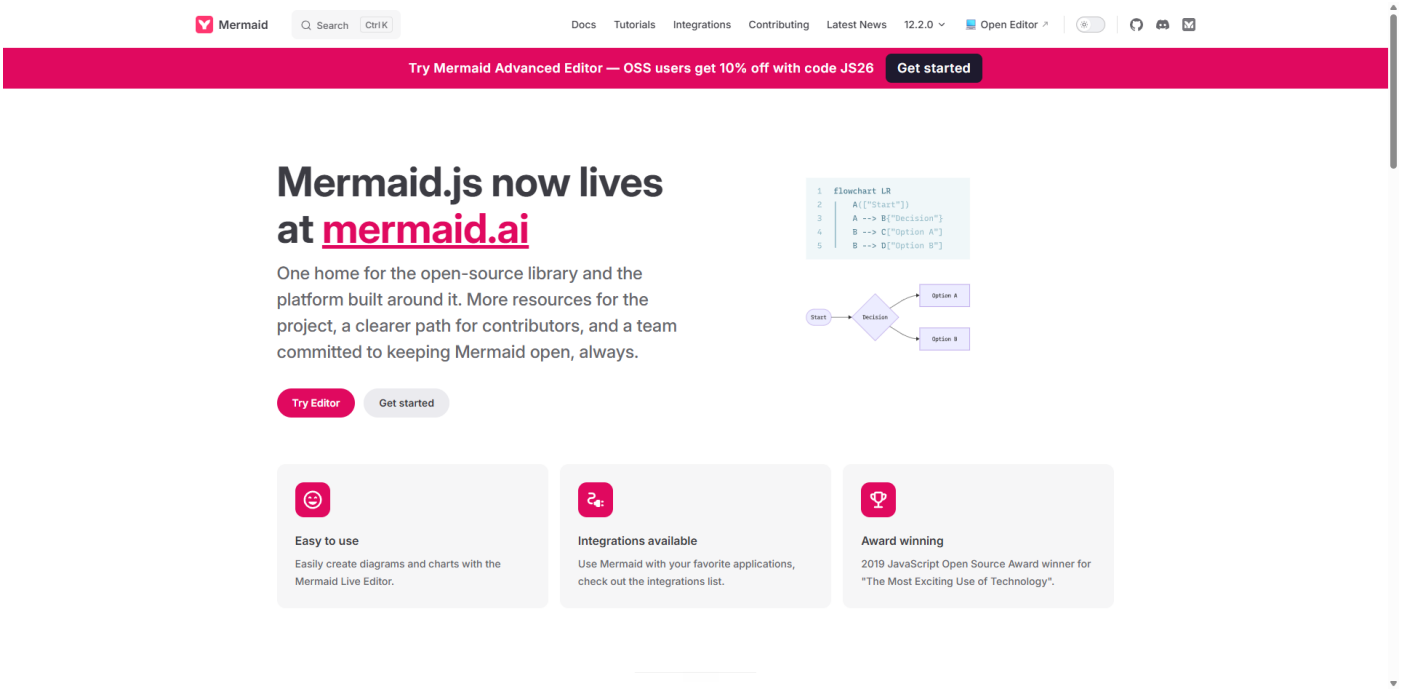
FINAL OUTPUT

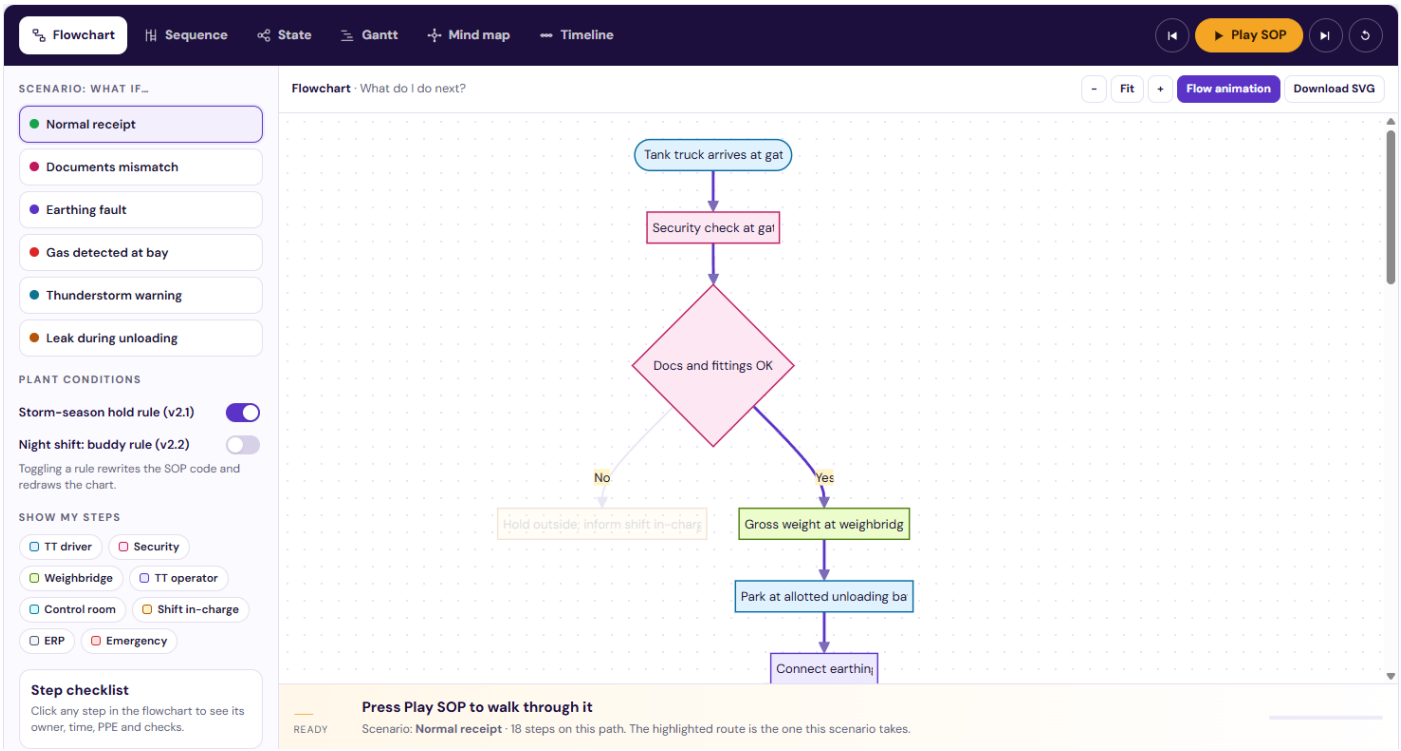
- Give me the complete HTML file in one block, ready to copy into a file named "ask-your-data.html".
- After the code, add 4–5 short steps telling a non-coder how to save the file, open it, get an OpenAI API key, and use the page.
- Add a brief note that this is for learning/demo use, and that in a real company the API key should be handled by a secure backend rather than typed into a web page.

Part — Charts & Data Visualization

Article – Mermaid Code

https://mermaid.js.org/





Flowchart | Sequence | State | Gantt | Mind map | Timeline

Play SOP

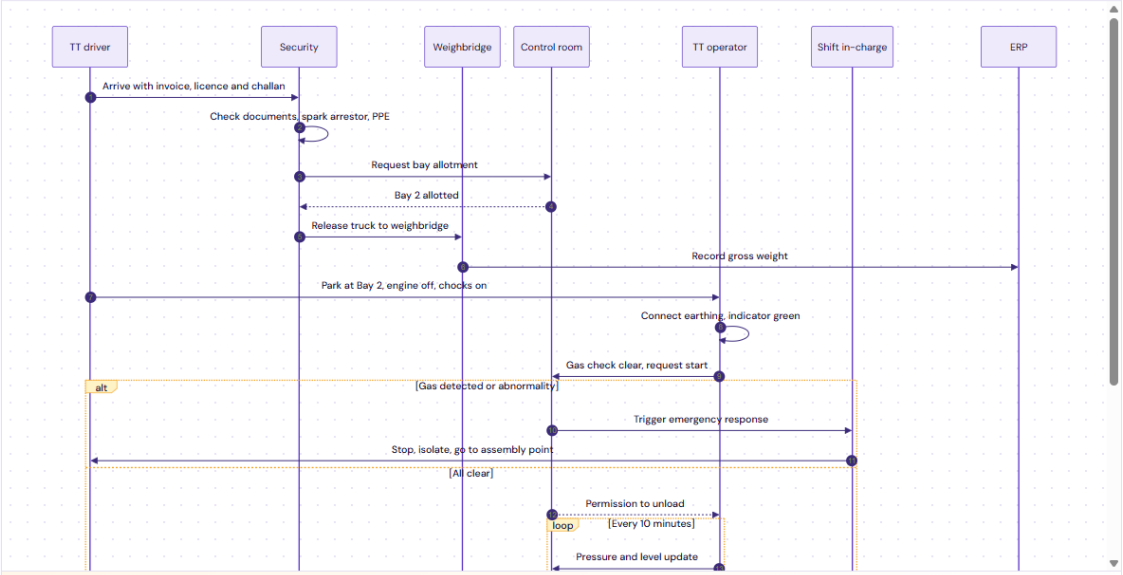
ABOUT THIS VIEW

Shows the hand-offs between seven roles, including the emergency branch.

Play it to step through each message in order. The loop shows the 10-minute pressure log.

Sequence · Who tells whom, and when?

- | Fit | + | Flow animation | Download SVG



READY

Press Play SOP to walk through it

19 messages to step through.

Flowchart | Sequence | State | Gantt | Mind map | Timeline

Play SOP

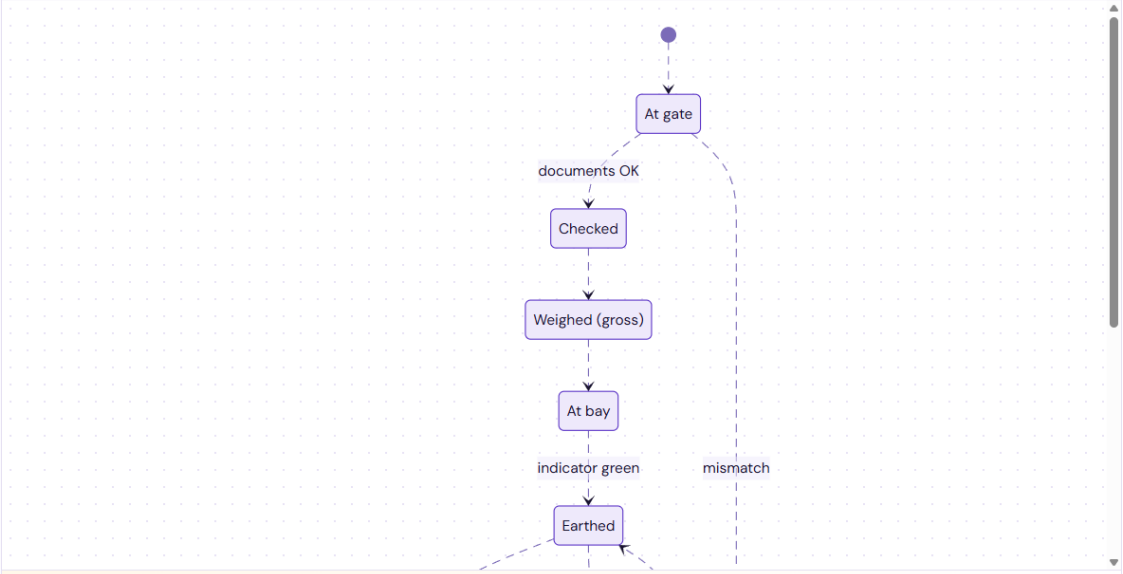
ABOUT THIS VIEW

Every tanker is always in exactly one state. The control-room screen shows each truck on this map.

Hold and Emergency are the only ways off the normal path.

State · Where is this tanker now?

- | Fit | + | Flow animation | Download SVG



READY

Press Play SOP to walk through it

13 items to step through.

Flowchart | Sequence | State | Gantt | Mind map | Timeline | Play SOP

SHUTDOWN WHAT-IF

Pipeline survey 4 days

Parallel repair crews

Tasks chain with "after" links, so a longer survey pushes every later task automatically. Orange bars are the critical path.

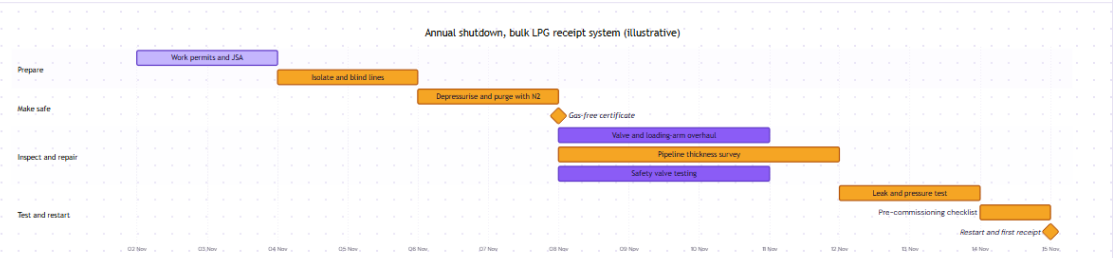
Plant back online

Sun 15 Nov

Repair window 4 days, crews in parallel

Gantt · When will the shutdown finish? · finish **15 Nov**

Fit | Flow animation | Download SVG



READY

Press Play SOP to walk through it

10 tasks to step through.

Flowchart | Sequence | State | Gantt | Mind map | Timeline | Play SOP

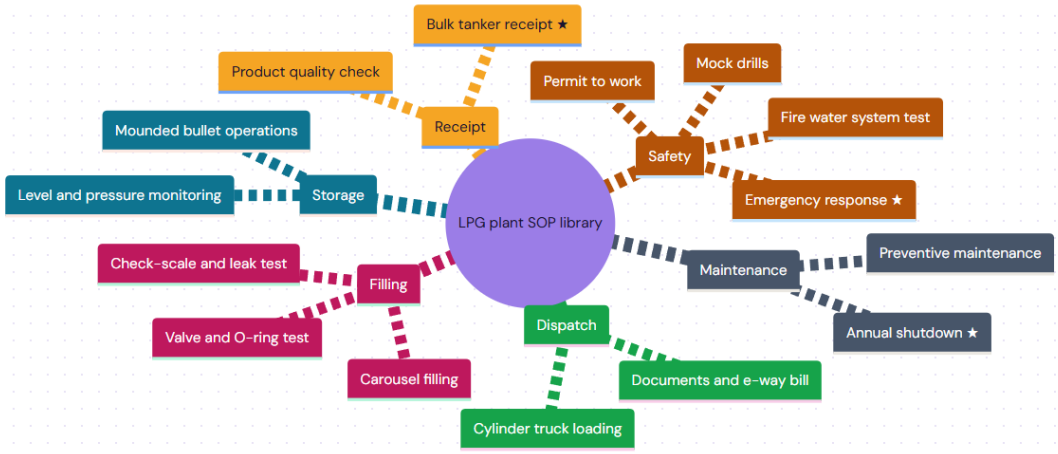
ABOUT THIS VIEW

The plant's SOP library at a glance. ★ marks SOPs already live as dynamic SOPs.

Click a ★ SOP to open it: Bulk tanker receipt, Emergency response or Annual shutdown.

Mind map · Which SOP do I need?

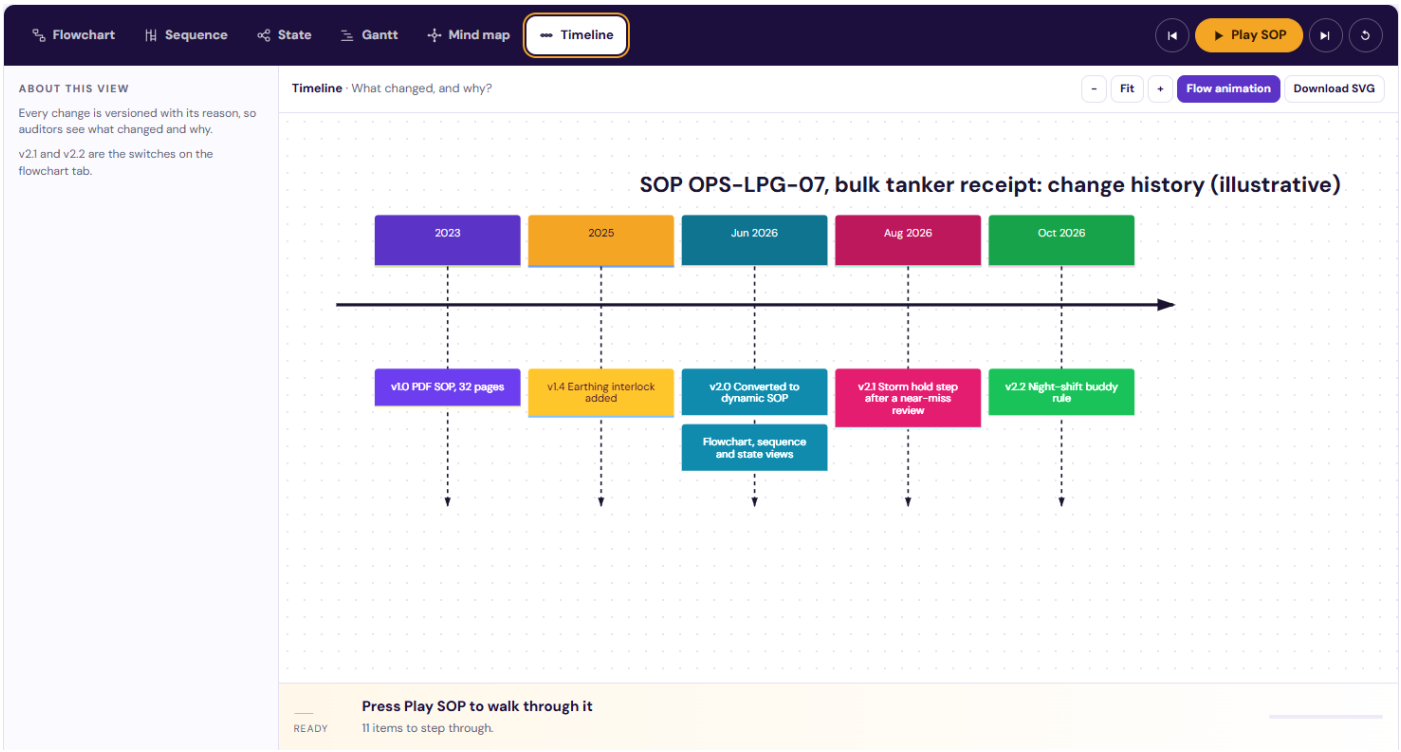
Fit | Flow animation | Download SVG



READY

Press Play SOP to walk through it

22 items to step through.



Prompt:

Create a professional, interactive, standalone HTML application that transforms a Standard Operating Procedure (SOP) into a dynamic visual experience using Mermaid.js.

Input: I will provide an SOP document, PDF, Word file, or written process description.

Objective: Convert the SOP into an interactive, visually engaging, and operationally useful web application.

1. Diagram views

Create six switchable Mermaid.js visualizations:

- **Flowchart:** Complete process with decision branches, exceptions, and outcomes.
- **Sequence diagram:** Communication and handoffs between people, teams, and systems.
- **State diagram:** Different process states and transitions.
- **Gantt chart:** Activity schedules, dependencies, and critical paths.
- **Mind map:** Hierarchical overview of SOP sections.
- **Timeline:** Procedure milestones and revision history.

2. Interactive capabilities

- Play, pause, reset, previous, and next controls for step-by-step animation.

- Highlight the active step and completed steps.
- Scenario selector to simulate normal and exception conditions.
- Interactive decision branches that update the highlighted process route.
- Role-based filters to show responsibilities for selected stakeholders.
- Clickable diagram nodes displaying step details, ownership, duration, PPE, requirements, and validation checklists.
- Checklist completion tracking.
- Zoom in, zoom out, and fit-to-screen.
- Animated process connections.

3. Editable Mermaid source

- Display the underlying Mermaid code for each diagram.
- Allow users to edit the code and automatically redraw the visualization.
- Provide Copy Code and Reset buttons.
- Enable SVG downloads.
- Show understandable error messages for invalid Mermaid syntax.

4. UI/UX design

Use a premium enterprise dashboard style inspired by McKinsey consulting reports.

Layout:

- Top navigation with six diagram tabs.
- Left sidebar for scenarios, roles, conditions, and checklists.
- Large central interactive diagram canvas.
- Playback controls and progress indicator.
- Collapsible Mermaid code editor below the diagram.

Make the application responsive, visually polished, and suitable for live demonstrations to senior management.

5. SOP processing

Extract the actual process steps from the supplied document, including decision points, responsibilities, control checks, exceptions, and source references.

Preserve the source procedure's meaning and terminology. Do not invent safety-critical instructions, timings, regulatory references, or approval requirements. Clearly flag missing information and distinguish assumptions from verified SOP content.

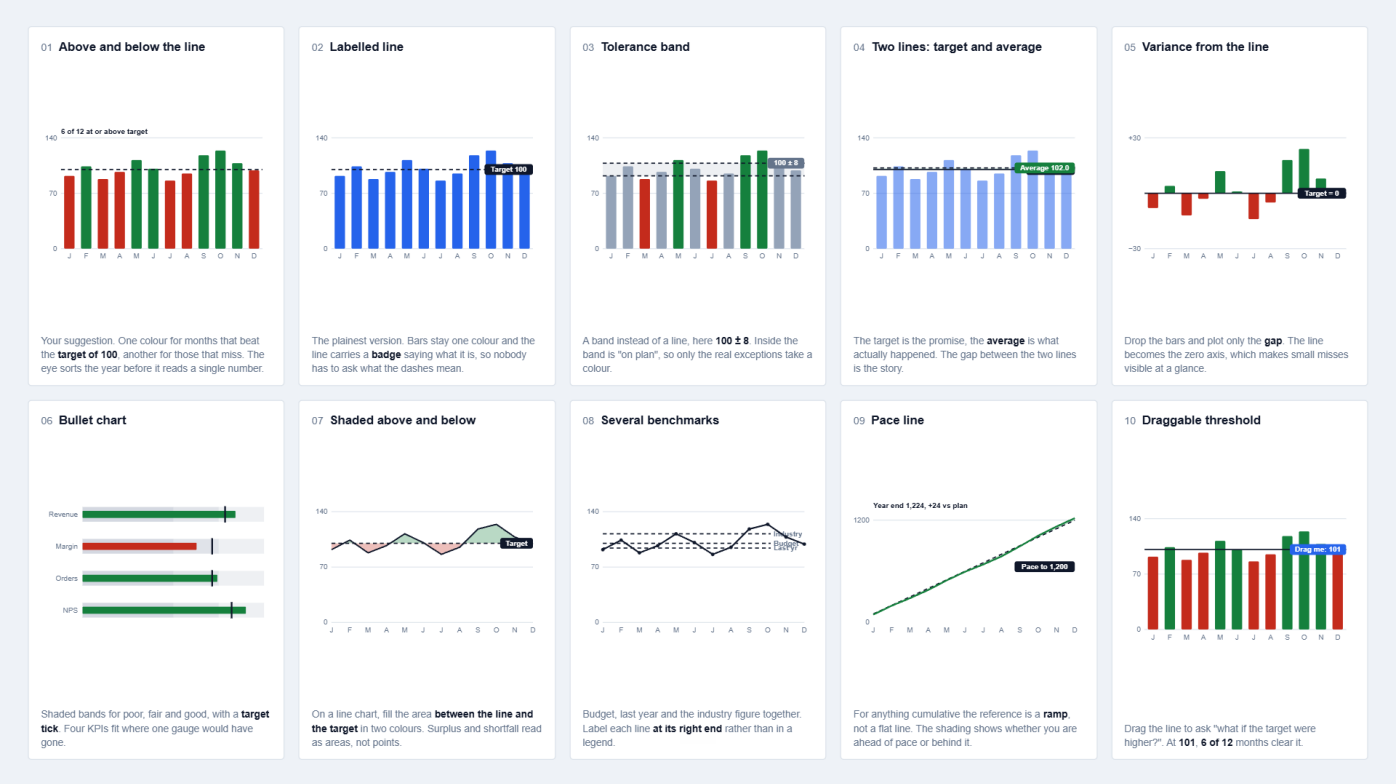
6. Technical requirements

- Deliver one complete, functional HTML file with embedded CSS and JavaScript.
- Use Mermaid.js for diagram rendering.
- Use D3.js only where necessary for additional interactivity.
- Do not require a backend, database, or server.
- Ensure every button and interactive feature works.
- Use responsive layouts suitable for desktops and tablets.
- Include appropriate error handling.
- Make the file suitable for local browser use and GitHub Pages deployment.

Output: Provide a downloadable standalone HTML file containing the interactive SOP Studio, without unrelated case study sections or marketing material.

Use the uploaded SOP as the authoritative content source. Any generated procedure must be explicitly labeled as a draft requiring approval by the responsible subject-matter experts before operational use.

Article – Reference Lines



Prompt:

1. Above and below the line

Build a 12-month column chart with a dashed target line at 100, colouring bars green when they meet or beat it and red when they miss.

Add a small heading that counts how many months were at or above target.

2. Labelled line

Build a 12-month column chart in a single colour with a dashed target line at 100.

Put a small "Target 100" badge on the right end of the line so it explains itself without a legend.

3. Tolerance band

Build a 12-month column chart with a shaded band from 92 to 108 around a target of 100, labelled "100 ± 8".

Colour bars grey inside the band, green above it and red below it, so only real exceptions stand out.

4. Target and average

Build a 12-month column chart with two reference lines: a solid target line at 100 and a dashed line at the actual average, each with its own labelled badge.

Shade the gap between the two lines to show how far actual performance landed from the promise.

5. Variance from the line

Build a 12-month chart of each month's gap from a target of 100 (actual minus target), with the target as the zero axis.

Draw bars up in green for months above target and down in red for months below, on a ±30 scale.

6. Bullet chart

Build bullet charts for 4 KPIs (Revenue, Margin, Orders, NPS), each with shaded poor/fair/good bands, an actual-value bar and a target tick.

Colour each bar green if it meets its target and red if it doesn't.

7. Shaded above and below

Build a 12-month line chart with a dashed target line at 100.

Fill the area between the line and the target green where it is above and red where it is below, so surplus and shortfall read as areas.

8. Several benchmarks

Build a 12-month line chart with three dashed reference lines: Budget (100), Last year (94) and Industry (112), each in its own colour.

Label each line at its right end instead of using a legend.

9. Pace line

Build a cumulative year-to-date line chart against a dashed straight-line "Pace to 1,200" ramp from zero to the annual goal.

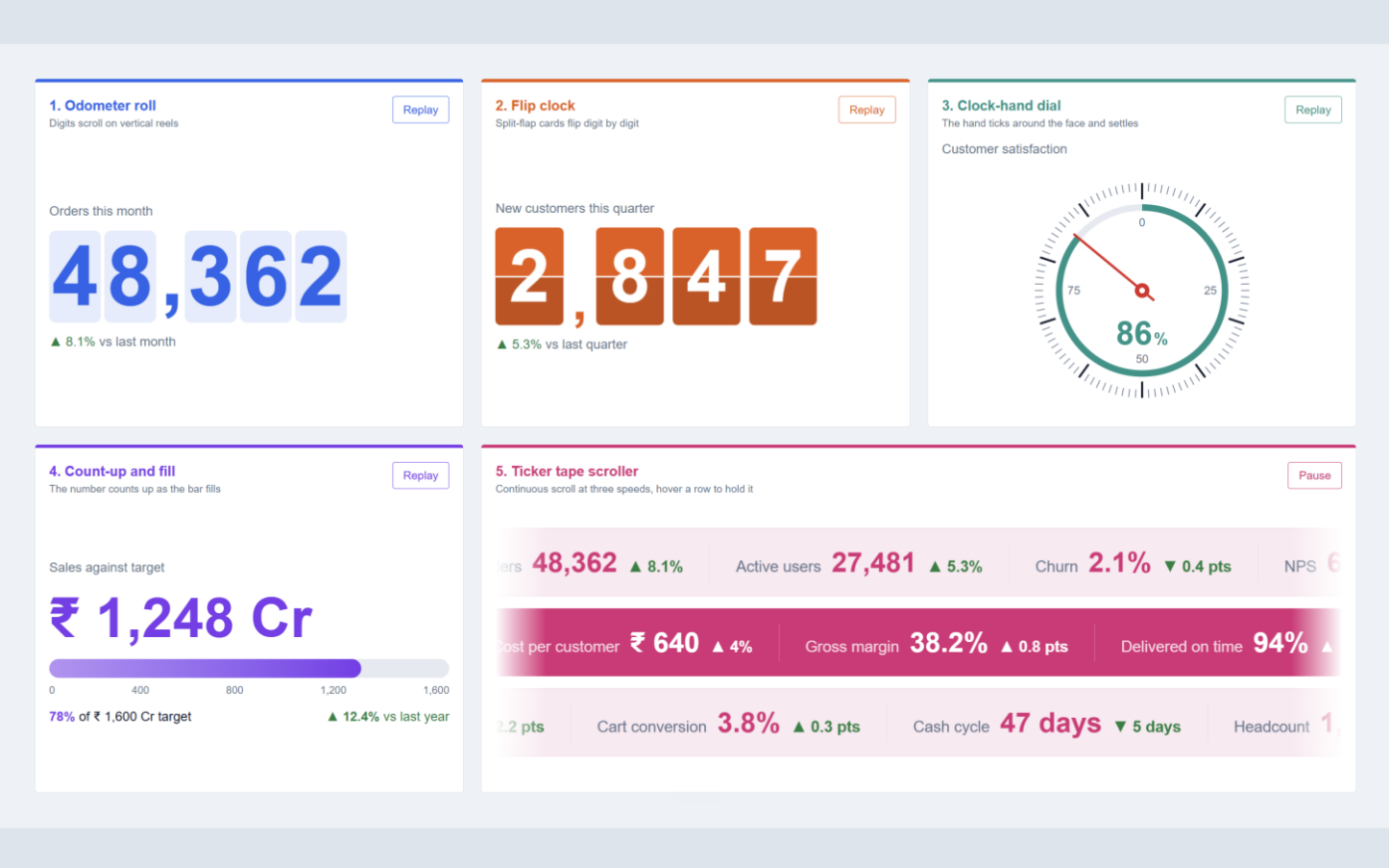
Shade the gap between actual and pace, and show a heading with the year-end total and how far it is over or under plan.

10. Draggable threshold

Build a 12-month column chart with a target line the user can drag up and down.

Recolour bars green or red live as the line moves, and update a note saying "At X, N of 12 months clear it".

Article – Animated KPIs



Prompt:

1. Odometer roll

Build a KPI card showing "Orders this month: 48,362" where each digit rolls up on its own vertical reel, like a car odometer, with each reel settling a little later than the one before.

Add "▲ 8.1% vs last month" below it and a Replay button.

2. Flip clock

Build a KPI card showing "New customers this quarter: 2,847" as split-flap cards that flip digit by digit until they land on the final number, like an airport departures board.

Add "▲ 5.3% vs last quarter" below it and a Replay button.

3. Clock-hand dial

Build a "Customer satisfaction" dial with 100 tick marks, where a red hand sweeps round to 86% while a coloured arc fills behind it, then wobbles slightly as it settles.

Show the percentage in the centre of the dial and add a Replay button.

4. Count-up and fill

Build a "Sales against target" card where ₹ 1,248 Cr counts up from zero while a progress bar fills to 78% of a ₹ 1,600 Cr target.

Show the % of target and "▲ 12.4% vs last year" counting up in step, and add a Replay button.

5. Ticker tape scroller

Build a wide card with three rows of KPIs (label, value, green or red change) scrolling continuously like a stock ticker, each row at a different speed and the middle row running the other way.

Hovering a row pauses it, and a Pause/Play button controls all three rows.

Article – Heatmap



Prompt:

Build a single-file HTML heatmap of monthly revenue (₹ Cr, 2025) for 15 Indian cities by 12 months. It sits on a 16:9 page that scales to fit any window. Use plain HTML, CSS and JavaScript, no libraries.

LEFT CONTROL PANEL

- Show cities: All / Top N / Bottom N, with a slider for N.

- Rank by: total of the shown months, or any single month.

- Colour: 6 swatches (blue, teal, orange, purple, pink, grey). The heatmap uses light-to-dark shades of the chosen colour.

- A two-handle slider to hide values outside a range. Hidden cells show a diagonal hatch pattern.

- A switch to show or hide the numbers in the cells.

- Clickable chips to hide individual cities or months, each set with a "Show all" link.

- A Reset all button.

RIGHT: THE HEATMAP

- Each row has a rank (#1, #2…) and a city name, then one coloured cell per month, then a Total column with a small bar.

- The column being ranked by is highlighted.

- A summary line at the top, such as "Showing top 5 of 15 cities, ranked by total, across 12 months. 3 cells hidden by value."

- A colour legend with min and max values plus a "Hidden" swatch. The colour scale refits to the cells on screen.

- Hovering a cell shows a tooltip with the city, month, ₹ value, and whether it is above or below that city's monthly average.

- Show a friendly empty message if every city or every month is hidden.

DATA AND STYLE

- Sample data from a fixed seed so it's the same every time, with a festive-season lift in Oct to Dec. Use Mumbai, Delhi NCR, Bengaluru, Hyderabad, Chennai, Ahmedabad, Kolkata, Pune, Surat, Jaipur, Lucknow, Vadodara, Indore, Kochi and Chandigarh.

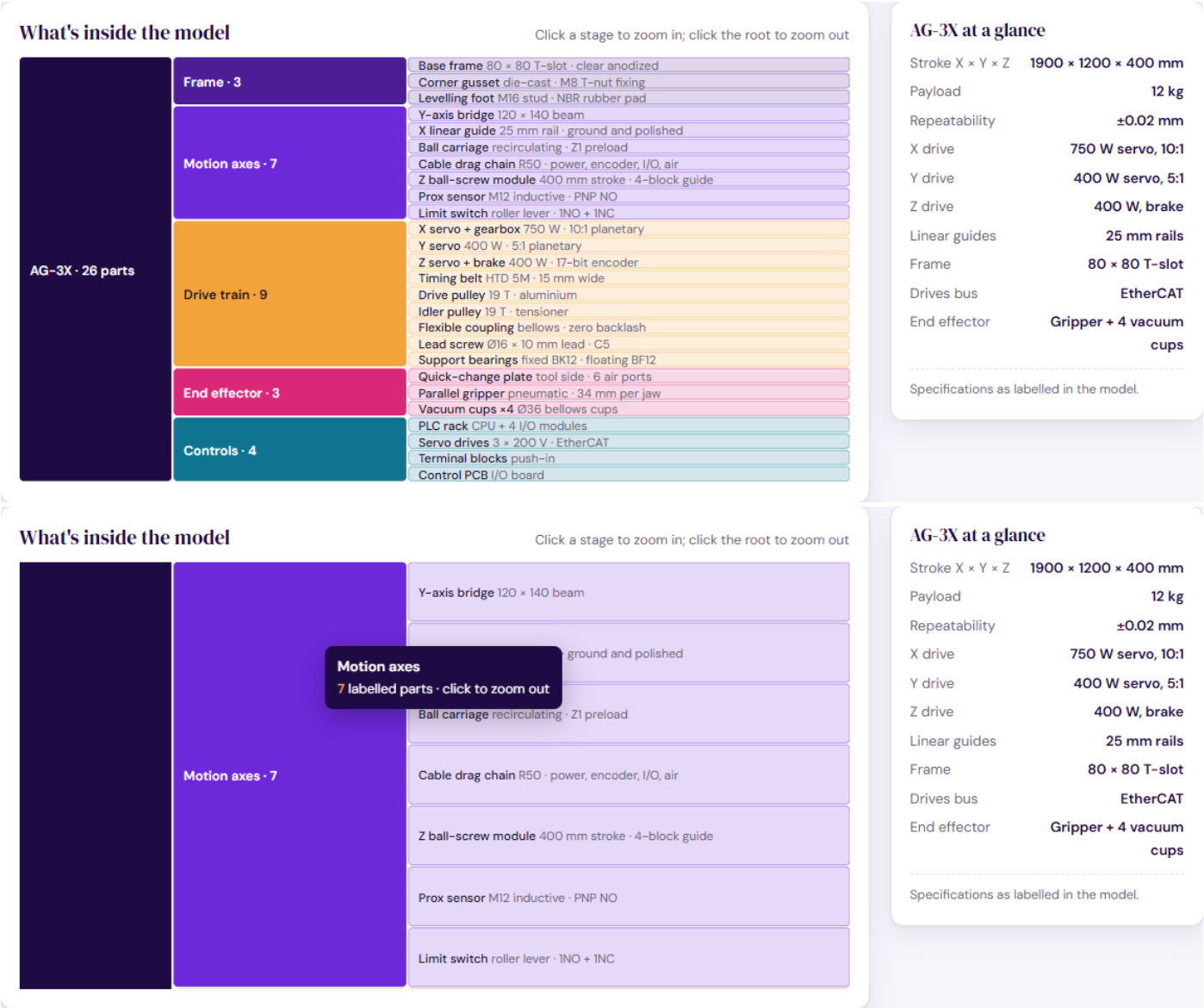
- Clean white cards, Helvetica, an accent colour that follows the chosen swatch, dark mode support, and print-ready. Deliver as one .html file.

Simplified Prompt:

Build a single-file 16:9 HTML heatmap of monthly revenue for 15 Indian cities × 12 months, with a total column and hover tooltips.

Add a side panel for Top/Bottom N, rank by month or total, colour choice, a value-range filter, a numbers on/off switch, chips to hide cities or months, and Reset all.

Article – Zoomable Table



Prompt:

Build a premium, interactive **Zoomable Hierarchical Table (D3.js Icicle / Partition Chart)** based on the attached reference HTML file.

Objective: Transform complex hierarchical information into a visually engaging, interactive, three-column visualization that allows users to explore categories and subcategories through animated zooming.

1. Visualization structure

Create a three-level hierarchical chart:

- **Column 1 – Root:** Main product, process, organization, or system.
- **Column 2 – Categories:** Major components, departments, stages, or groups.

- **Column 3 – Details:** Individual components, activities, or elements, along with their specifications.

Use a D3.js partition layout to automatically position elements based on the underlying hierarchy.

Each category must have a distinct color, and its child elements should use lighter shades of the same color.

2. Zoom and navigation

Replicate the interaction in the reference file:

- Clicking a category smoothly zooms into that category.
- All three columns remain aligned as the visualization changes.
- The selected category expands vertically, making its child elements easier to read.
- Other categories move outside the visible focus.
- Clicking the selected category again zooms out to the full view.
- Clicking the root also resets the visualization.
- Use smooth, professional animations lasting approximately 850 milliseconds.

Important: This must be a genuinely interactive hierarchical visualization, not a static table or a collection of expandable cards.

3. Hover tooltips

Display a floating tooltip when hovering over any element.

For categories, show the category name and number of child elements.

For individual elements, show their name, parent category, and technical or descriptive specifications.

Position tooltips near the cursor without letting them overflow the browser viewport.

4. Information panel

Add a separate information card on the right, similar to the reference HTML.

The card should contain relevant technical specifications, KPIs, product details, or business information.

Use a clean, two-column label-and-value layout.

5. Design language

Follow a premium McKinsey-style consulting dashboard aesthetic:

- White cards with subtle shadows.
- Light grey background.
- Deep purple primary color (#4C1D95).

- Dark purple headings (#1E0B44).
- Orange accents (#F2A33A).
- DM Sans typography.
- Rounded corners and generous whitespace.
- Precise alignment and readable labels.
- Minimal visual clutter.

Make the design suitable for corporate presentations, executive dashboards, and client demonstrations.

6. Technology

- Use D3.js v7.
- Build using HTML5, CSS3, SVG, and vanilla JavaScript.
- Use d3.hierarchy() and d3.partition().
- Animate transitions using d3.transition().
- Use responsive SVG viewBox dimensions.
- Ensure smooth zooming and accurate positioning.
- Support desktop and tablet layouts.
- Respect reduced-motion accessibility preferences.

7. Dynamic data

Store the entire hierarchy in a clearly structured JavaScript object.

Allow the content to be replaced easily with other hierarchical datasets without modifying the visualization engine.

Automatically calculate category counts and the overall number of leaf-level elements.

Do not invent factual specifications. Use the data I supply and mark missing information clearly.

8. Final deliverable

Produce a **single, standalone HTML file** containing the complete interactive visualization.

The HTML should open directly in a browser and work without a backend. External CDN loading for D3.js and fonts is acceptable.

Critical requirements:

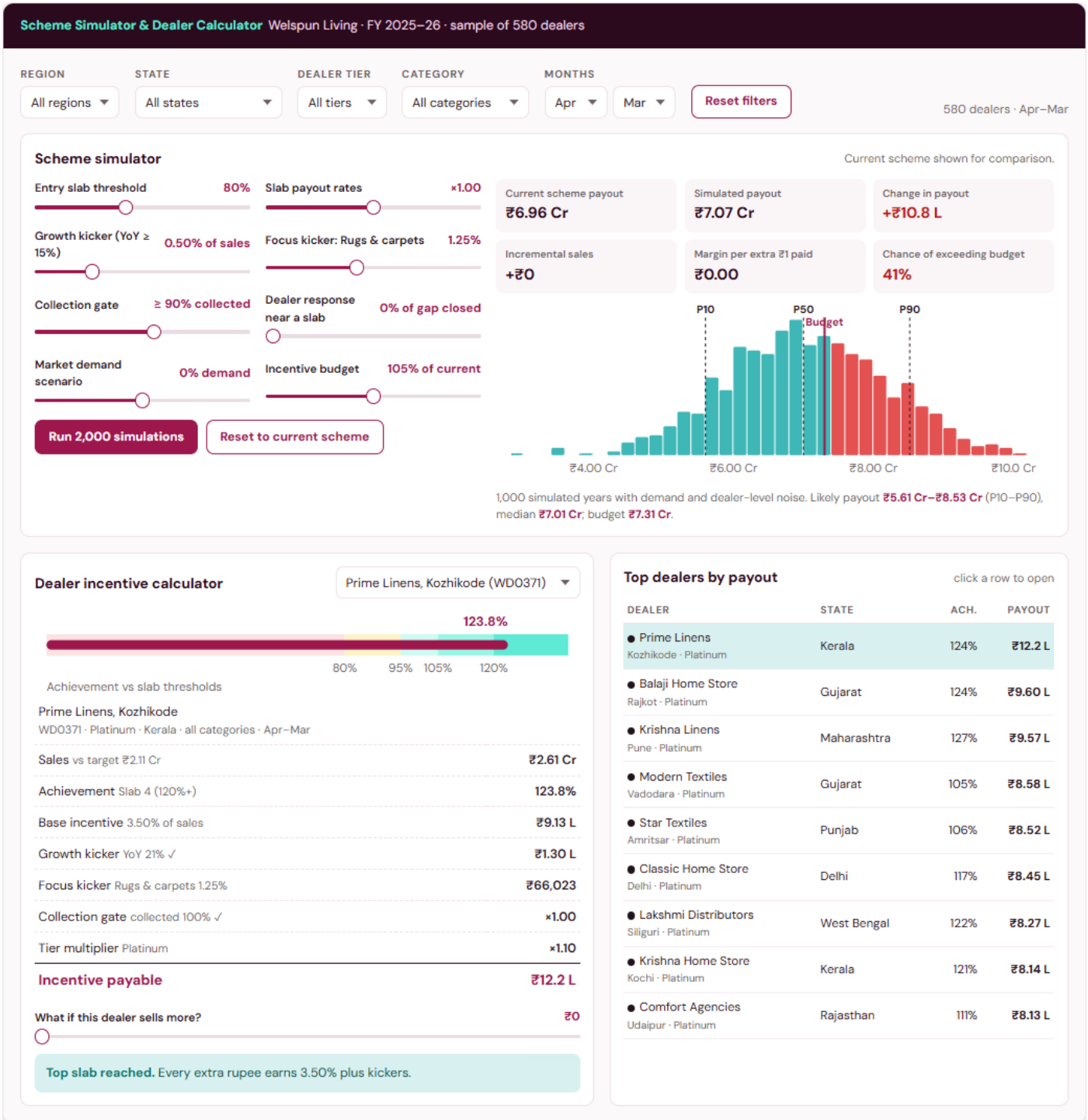
1. Reproduce the reference file's three-column zoomable partition layout.
2. Preserve smooth animated drill-down and zoom-out behavior.
3. Maintain category color consistency.
4. Keep the right-side specification panel.

5. Include functional hover tooltips.
6. Ensure text remains readable when zoomed.
7. Deliver working HTML, not a screenshot, mockup, or static diagram.

Use the attached **Zoomable Table.html** as the primary reference for layout, animation behavior, visual styling, and interaction design.

Adapt the underlying hierarchy and specifications to the new content I provide.

Article – Interactive Simulation for Charts



Prompt:

Build a single self-contained HTML page: an interactive "Dealer Incentive Scheme Simulator & Dealer Calculator" for a home-textiles brand (Welspun Living) that sells through ~580 dealers across India.

DATA (synthetic, deterministic)

- Use a seeded random generator (mulberry32, seed 2610) so numbers are identical on every load.

- Create ~580 dealers across Indian states, grouped into 6 regions (North, West, South, East, Central, North East). Weight dealer counts by state size (e.g. Maharashtra 60, UP 50, Tamil Nadu 45 … Sikkim 1), each with 2–4 real cities.

- Each dealer has: code (WD0001…), name ("Shree Furnishings, Pune" style), state, region, city, tier (Platinum 14% / Gold 36% / Silver 50%), annual target in ₹ lakh (Platinum 90–220, Gold 30–85, Silver 8–30), monthly target split by FY seasonality (Apr–Mar, festive peak Oct–Nov), monthly actuals = target × dealer performance (~0.55–1.45) with noise, a category mix across Bath linen / Bed linen / Rugs & carpets / Flooring, YoY growth %, and collection % (~72–100%).

SCHEME ENGINE

- Achievement = sales ÷ target for the selected months and category.

- Slabs: 80% / 95% / 105% / 120% paying 1% / 2% / 3% / 3.5% of sales; nothing below entry slab.

- Growth kicker: +0.5% of sales if YoY growth ≥ 15%.

- Focus kicker: +1% on Rugs & carpets sales.

- Collection gate: if collections < 90%, payout halved.

- Tier multiplier: Platinum ×1.1, Gold ×1.0, Silver ×0.9.

- Keep a frozen "current scheme" and an editable "simulated scheme" so every number can be compared.

LAYOUT

1. Filter row: Region, State (filtered by region), Tier, Category, From/To month, Reset. Show "N dealers · Apr–Mar" on the right.

2. Scheme simulator panel (two columns):

  - Left: 8 sliders with live value labels — entry slab threshold (70–94%), slab rate multiplier (×0.5–1.5), growth kicker (0–2%), focus kicker (0–3%), collection gate (80–98%), dealer response near a slab (0–100% of gap closed when within 7 pts of next slab), market demand scenario (−15% to +15%), incentive budget (70–140% of current). Buttons: "Run 2,000 simulations", "Reset to current scheme".

  - Right: 6 output tiles — current payout, simulated payout, change (red if up, green if down), incremental sales, margin per extra ₹1 paid (assume 30% margin on incremental sales), chance of exceeding budget.

  - Below tiles: Monte Carlo histogram (D3) of total payout across simulated years with demand shock and dealer-level noise; bars teal below budget, red above; dashed markers for P10/P50/P90, solid marker for Budget; labels must not overlap. One-line summary underneath.

  - Re-run 1,000 simulations automatically (debounced) when any slider moves.

3. Bottom row (two columns):

- Dealer incentive calculator: dealer dropdown; a horizontal gauge showing achievement against coloured slab bands; a line-by-line breakdown table (sales vs target, achievement + slab, base incentive, growth kicker, focus kicker, collection gate, tier multiplier, incentive payable); a "What if this dealer sells more?" slider; a teal callout: "₹X more sales takes [dealer] to the next slab, lifting payout to ₹Y (+₹Z)".

- Top dealers by payout: table of top 9 (name with tier dot, city, state, achievement, payout); clicking a row opens that dealer in the calculator and highlights the row.

FORMATTING

- Values are stored in ₹ lakh. Display adaptively: ≥ ₹100 L as "₹x.xx Cr", ≥ ₹1 L as "₹x.x L", below that as full rupees with Indian digit grouping. Never show "₹0.0 Cr" for small filtered views.

- Tabular numbers everywhere.

DESIGN

- DM Serif Display for headings, DM Sans for UI (Google Fonts). D3 v7 from cdnjs.

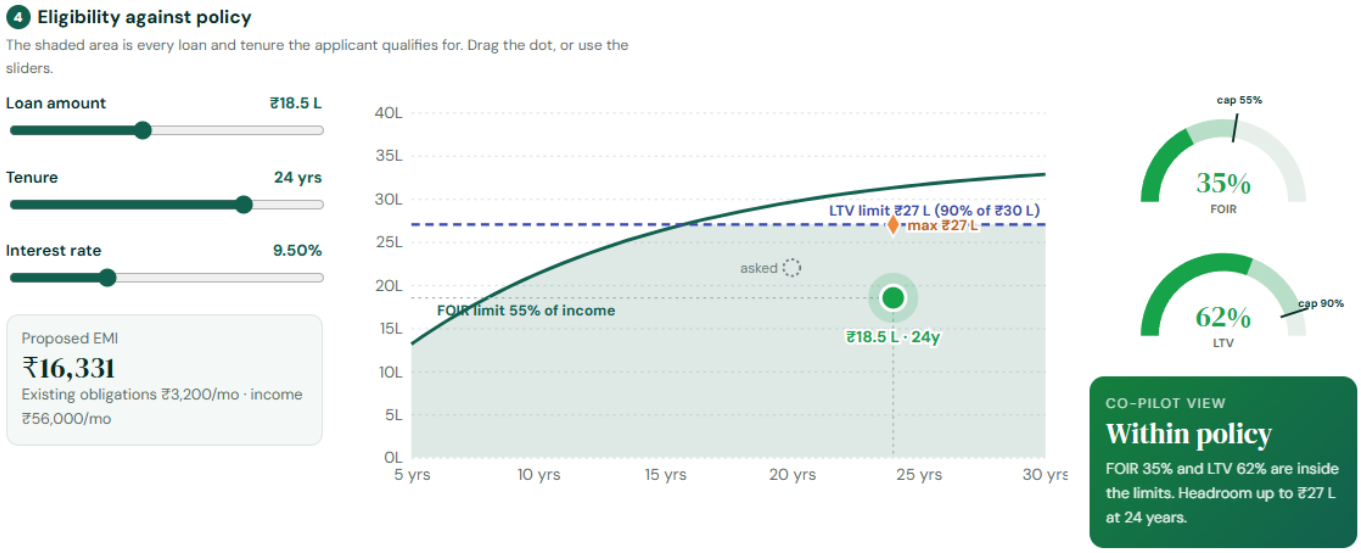
- Palette: primary #9D174D, deep #2B0A1E, accent teal #0EA5A4, background #FAF6F8, text #22141C, muted #6E5F67, lines #EFE3E9.

- Dark header bar with title, white cards with soft shadow, 10–12px radius, custom-styled range sliders filled in primary colour.

- Responsive: two-column panels stack below 1000px; sliders single-column and tiles 2-up on mobile. Respect prefers-reduced-motion.

- Footer note: "Illustrative sample; names, figures and scheme values are synthetic. © Havish M Consulting".

Everything in one HTML file, no build step.



Prompt:

Build a single self-contained HTML page: an interactive "Eligibility against policy" panel for an affordable home-loan credit co-pilot (PNB Housing Finance). It shows every loan amount and tenure an applicant qualifies for, and checks a proposed loan against credit policy.

APPLICANTS (illustrative, switchable via a segmented control top-right)

- Priya S. · Salaried · IT support executive, Pune · income ₹56,000/mo · obligations ₹3,200/mo · FOIR cap 55% · property value ₹30 L · asked ₹22 L, 20 yrs, 9.50%

- Ramesh K. · Self-employed · Kirana store owner, Indore · income ₹32,712/mo · obligations ₹2,500/mo · FOIR cap 50% · property ₹25 L · asked ₹18 L, 20 yrs, 10.25%

- Imran A. · Self-employed · Cab driver-owner, Lucknow · income ₹31,271/mo · obligations ₹14,500/mo (vehicle loan with 5 EMIs left) · FOIR cap 50% · property ₹19 L · asked ₹12 L, 20 yrs, 10.75%

Show a one-line summary of the selected applicant under the header. Switching applicant resets the sliders to their ask.

POLICY MATHS

- EMI = loan × r / (1 − (1+r)^−n), r = annual rate/1200, n = years × 12.

- FOIR = (EMI + existing obligations) ÷ income; must be ≤ the profile's cap.

- LTV cap by loan size: 90% up to ₹30 L, 80% up to ₹75 L, 75% above. LTV = loan ÷ property value.

- Max loan by FOIR at tenure t = (FOIR cap × income − obligations) ÷ EMI factor(t). Max eligible = min(FOIR max, LTV max).

- Toggle (only for applicants with a loan closing within 6 EMIs): "Exclude loans closing within 6 EMIs (with closure proof)" sets obligations to 0.

LAYOUT (three columns)

1. Left: sliders for loan amount (₹3–40 L, step 0.5), tenure (5–30 yrs), interest rate (8–13%, step 0.05), each with a live value; the exclusion checkbox; an EMI box showing proposed EMI large, plus obligations and income.

2. Centre: D3 eligibility frontier chart. X = tenure (5–30 yrs), Y = loan (0–40 L).

- Shaded area = all eligible loan/tenure combinations.

- Solid green curve = FOIR limit, labelled "FOIR limit 55% of income".

- Dashed indigo horizontal line = LTV limit, labelled "LTV limit ₹27 L (90% of ₹30 L)".

- Dashed grey ring marking the applicant's original ask, labelled "asked".

- A draggable dot for the current proposal (green if within policy, red if not) with a halo, a label like "₹17 L · 23y", and dotted guide lines to both axes. Dragging or clicking anywhere on the chart snaps loan to ₹0.5 L and tenure to whole years, and updates the sliders.

- An orange diamond at the current tenure showing "max ₹X L"; hide it when the dot is already at the max.

3. Right: two semicircle gauges (FOIR and LTV) showing the value in a big serif number, a light-green arc up to the cap, a tick and "cap 55%" label at the cap, and an animated value arc that turns red above the cap. Below them, a verdict card:

- Within policy (green gradient): "FOIR 33% and LTV 57% are inside the limits. Headroom up to ₹27 L at 23 years."

- Outside policy (orange gradient): headline "Reduce to ₹X L" (or "Not eligible as filed"), reasons listed, plus a suggestion such as "Or keep ₹17 L with a 25-year tenure", or "Excluding the closing vehicle loan may help".

FORMATTING

- ₹ amounts in lakh as "₹17 L" / "₹4.25 L"; EMI and income in full rupees with Indian digit grouping.

DESIGN

- DM Serif Display for numbers and headlines, DM Sans for UI (Google Fonts). D3 v7 from cdnjs.

- Palette: primary #12614F, deep #072B24, accent #F0883E, indigo #3B4BA8, background #F4F8F6, text #10231E, muted #5A6E68, lines #DCE7E2.

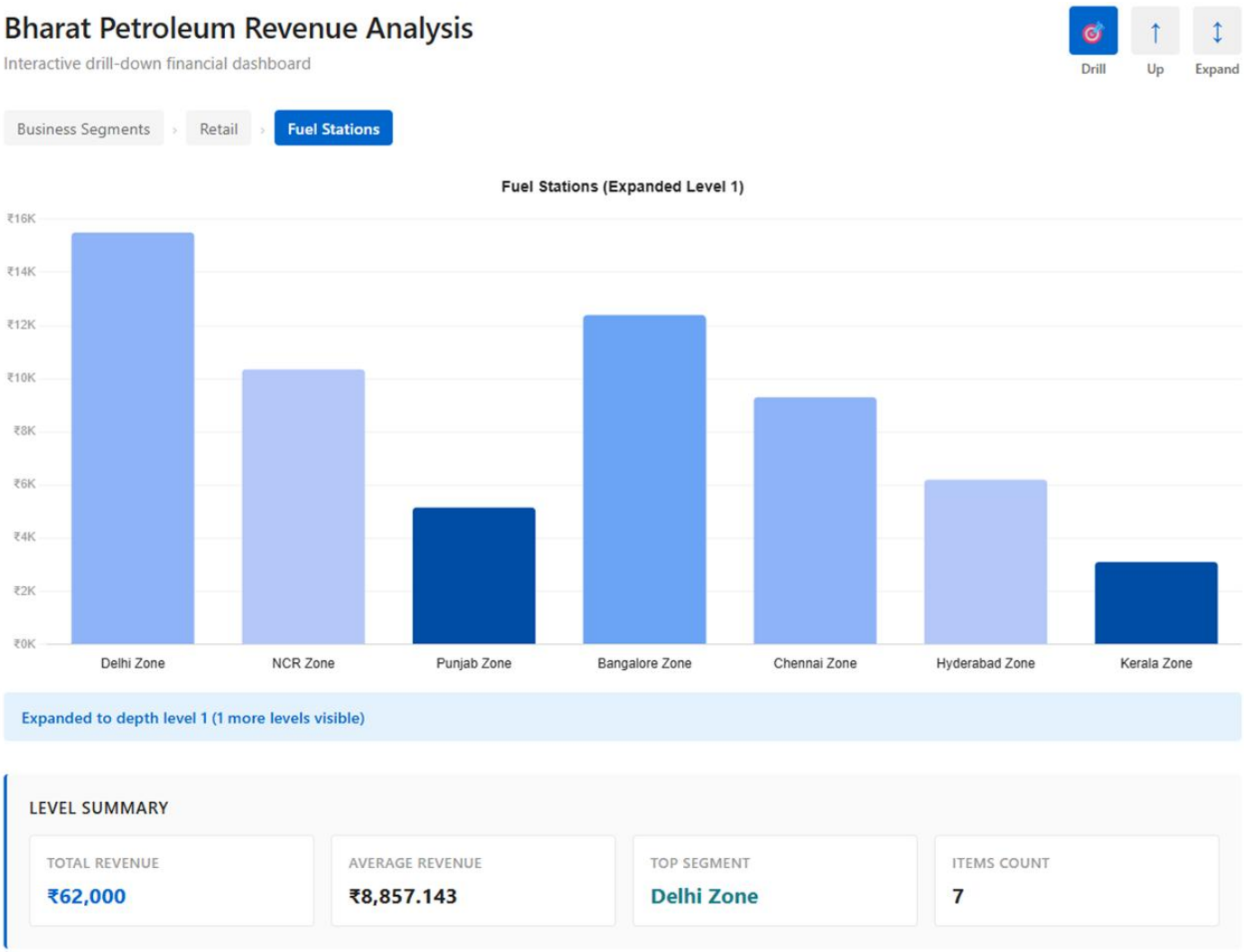
- White card with soft shadow and 12px radius; header "④ Eligibility against policy" with a one-line explanation underneath.

- Responsive: the chart goes full-width above the controls below 1060px; everything stacks on mobile, and the chart scrolls horizontally rather than squashing. Respect prefers-reduced-motion.

- Footnote: "Applicants and policy limits are illustrative and do not represent any real customer. © Havish M Consulting".

Everything in one HTML file, no build step.

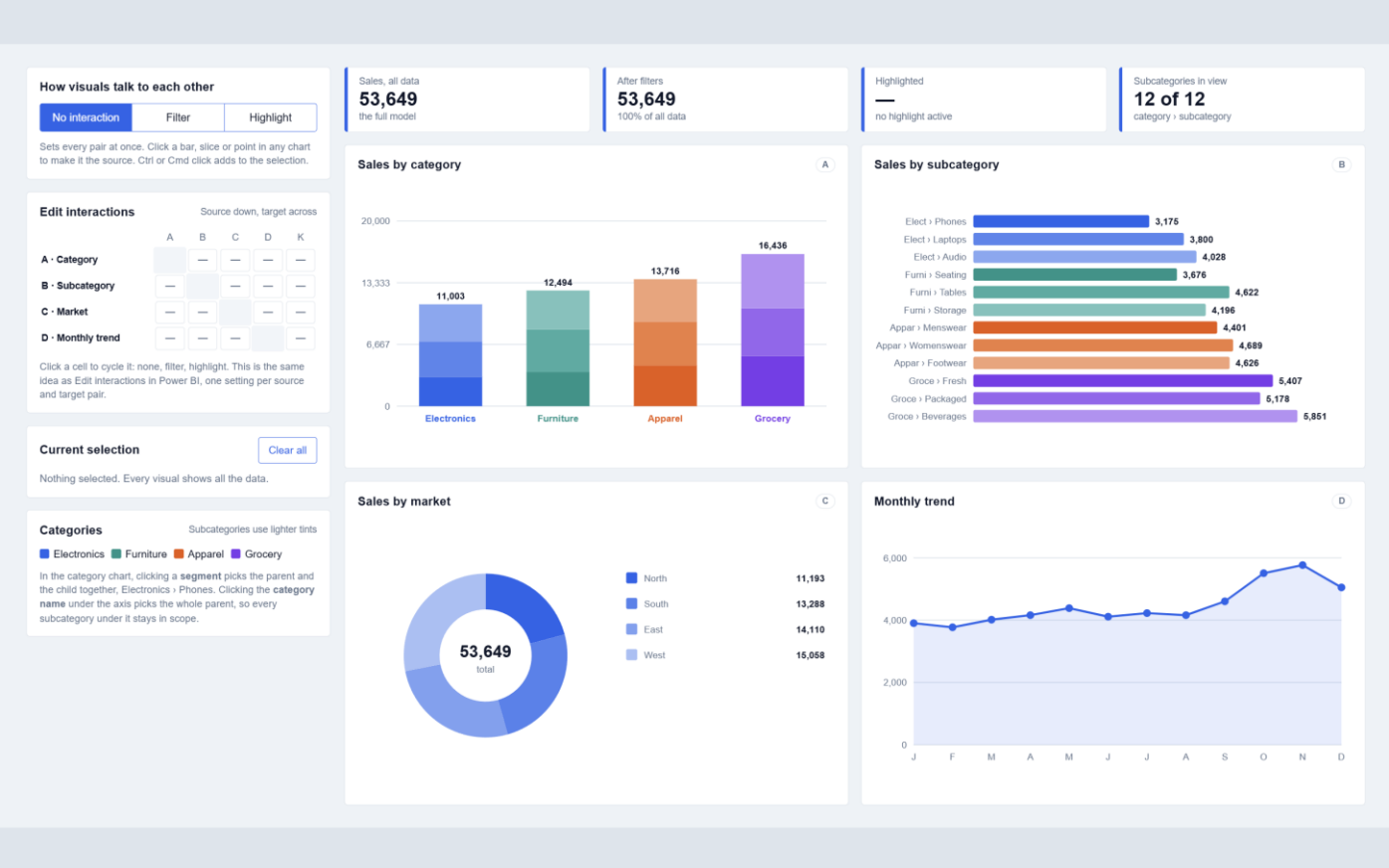
Article – Drill



Prompt:

Create an interactive HTML + JavaScript column chart that supports drill-down (click a bar to expand into sub-categories) and drill-up (Back button). Include expand/collapse controls to show/hide deeper levels, with animated transitions and breadcrumb navigation.

Article – Filter and Highlight



Prompt:

Build a single-file HTML page (16:9, scales to fit any window) that demonstrates Power BI-style cross-filtering and cross-highlighting between charts. Use plain HTML, CSS and JavaScript, no libraries.

DATA

- Sample sales data from a fixed seed with 4 categories, each with 3 subcategories: Electronics (Phones, Laptops, Audio), Furniture (Seating, Tables, Storage), Apparel (Menswear, Womenswear, Footwear) and Grocery (Fresh, Packaged, Beverages).

- Cover 4 markets (North, South, East, West) and 12 months, with a festive-season lift.

RIGHT SIDE: THE REPORT

- 4 KPI cards: Sales (all data), After filters, Highlighted (% of what's shown), and Subcategories in view.

- 4 charts in a 2×2 grid, labelled A to D:

A. Sales by category as stacked columns, with each segment a lighter tint of its category's colour.

B. Sales by subcategory as horizontal bars grouped under their parent.

C. Sales by market as a donut with a legend.

D. Monthly trend as a line chart.

- Clicking any bar, segment, slice or point selects it. Ctrl/Cmd-click adds to the selection, and clicking empty space clears it.

- When a chart is the source of a selection, outline it and tag it "source".

LEFT SIDE: CONTROLS

- A switch to set every interaction at once: No interaction / Filter / Highlight.

- An "Edit interactions" grid (source charts down, target charts and KPIs across). Click a cell to cycle it through none, filter (F) and highlight (H).

- A "Current selection" panel listing what's selected in each chart as chips, with a Clear all button.

- A category colour legend and a short note explaining that clicking a segment selects parent and child, while clicking the category name selects the whole parent.

BEHAVIOUR

- Filter removes non-matching data from the target.

- Highlight keeps the full totals faint and draws the matching part solid on top: the lower part of a column, the left part of a bar, the inner band of a donut slice, or a second line.

- Hover tooltips show the name, the value, and the highlighted amount and %.

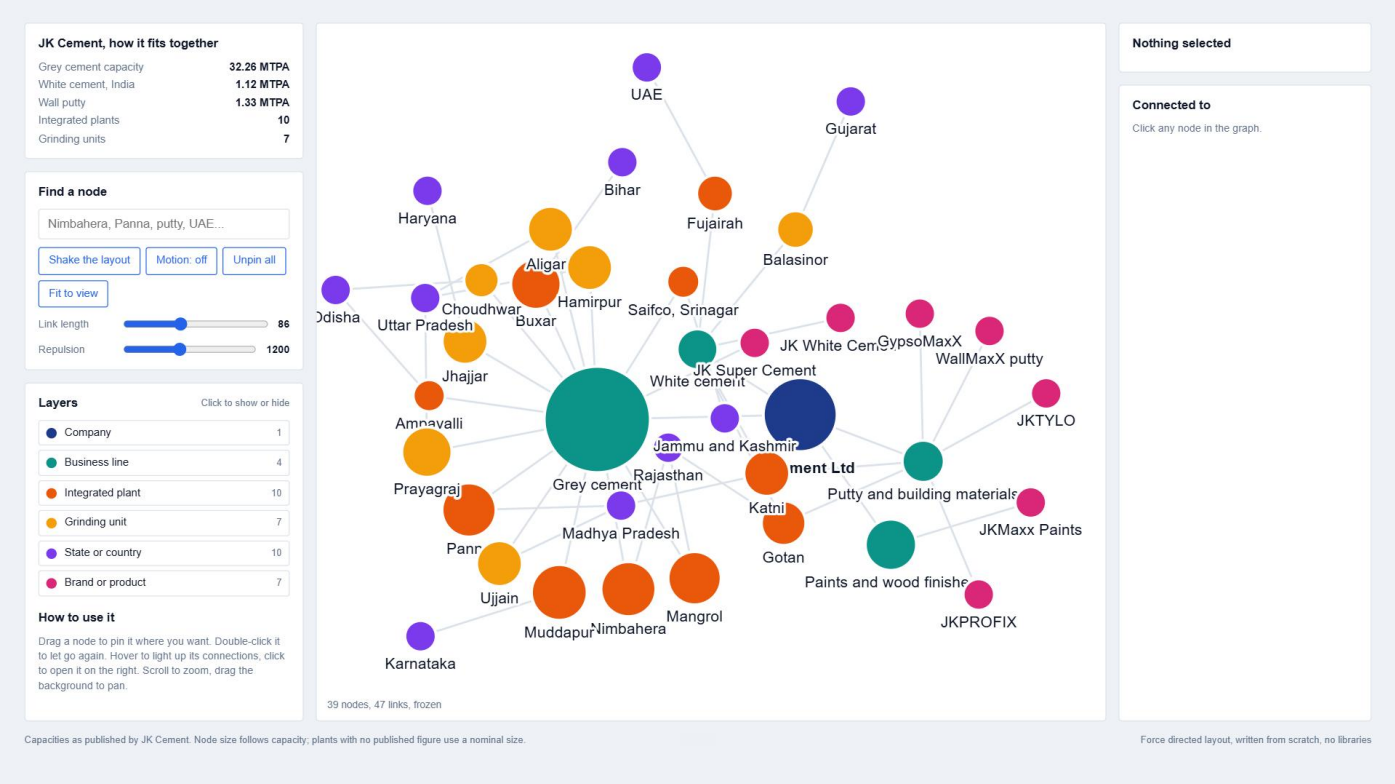
Clean white cards, blue accent, dark mode support. Deliver as one .html file.

Simplified Prompt:

Build a single-file 16:9 HTML report with 4 KPI cards and 4 linked charts (category stacked columns, subcategory bars, market donut, monthly line). Clicking any element filters or highlights the others, Power BI style.

Add a side panel with a None/Filter/Highlight switch, an "Edit interactions" grid to set each source-target pair, and a current-selection list with Clear all.

Article – Force Directed Graph



Prompt:

Build a single-file HTML force-directed network graph of how JK Cement fits together. Fit it to a 16:9 page that scales to any window. Write the force layout from scratch in plain JavaScript and SVG, with no libraries.

THE NETWORK (about 40 nodes in 6 colour-coded layers)

- Company: JK Cement Ltd at the centre.

- Business lines: Grey cement (32.26 MTPA), White cement (1.12 MTPA), Putty and building materials (1.33 MTPA), Paints and wood finishes.

- Integrated plants: Nimbahera, Mangrol, Muddapur, Panna, Buxar, Ampavalli, Saifco (Srinagar), Gotan, Katni, Fujairah.

- Grinding units: Jhajjar, Aligarh, Hamirpur, Prayagraj, Ujjain, Balasinor, Choudhwar.

- States or countries: link each plant to its state, or to the UAE for Fujairah.

- Brands: JK Super Cement, JK White Cement, WallMaxX, GypsoMaxX, JKTYLO, JKPROFIX, JKMaxx Paints.

- Node size follows capacity in MTPA. Nodes without a published figure get a nominal size.

LAYOUT (3 columns)

- Left panel:
  - Key company specs.
  - A search box that jumps to a matching node.
  - Buttons: Shake the layout, Motion on/off, Unpin all, Fit to view.
  - Sliders for link length and repulsion.
  - A layer legend where clicking a layer shows or hides it, with node counts.
  - Short how-to-use notes.
- Centre: the graph.
  - Hovering a node lights up its connections, dims the rest and shows a tooltip.
  - Dragging a node pins it, and double-clicking releases it.
  - Scroll to zoom and drag the background to pan.
  - A small status line shows the node and link count and whether the layout is settling or settled.
- Right panel: details for the selected node (layer, capacity, connections, note) and a clickable list of connected nodes for hopping around the network.

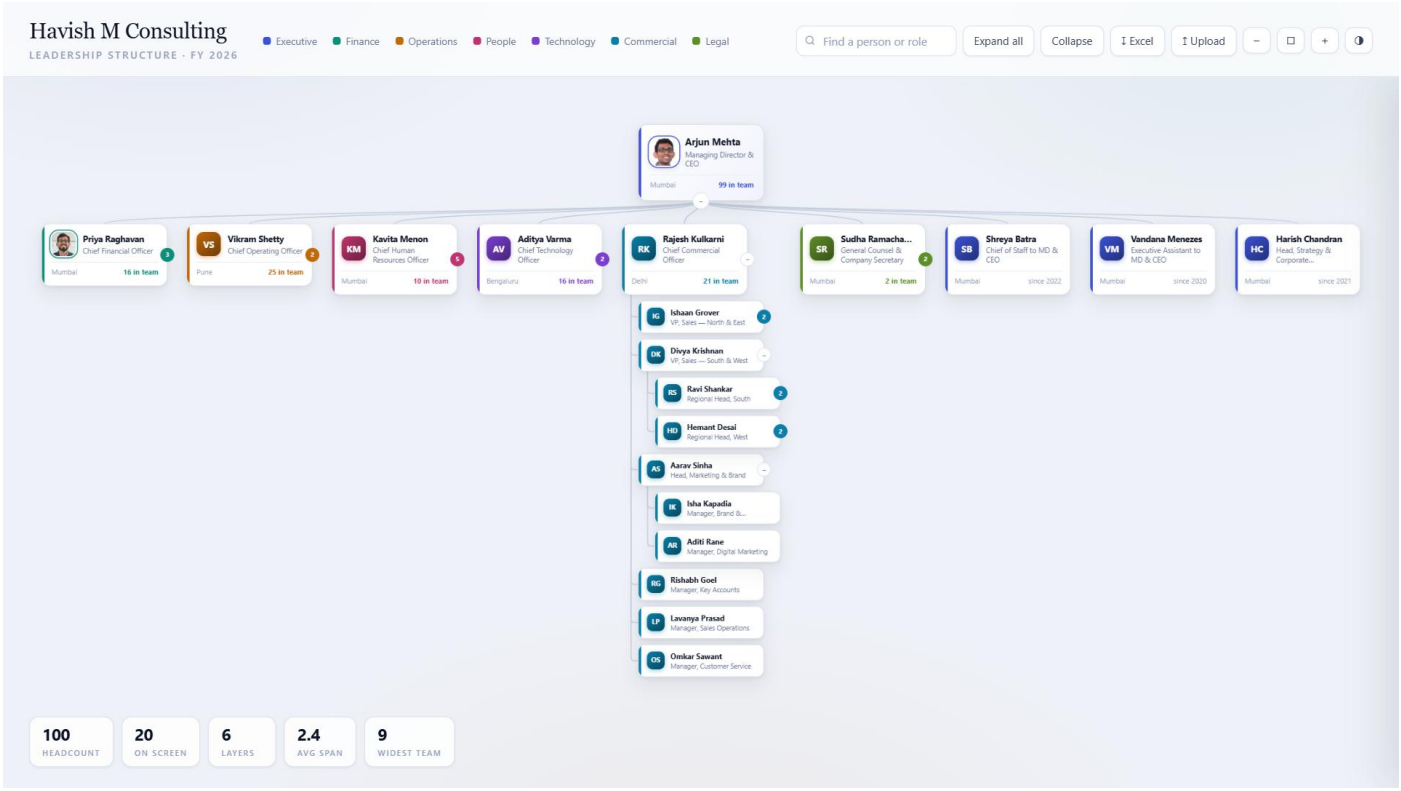
Clean white cards, a distinct colour per layer, dark mode support, and a footer noting the capacities are as published. Deliver as one .html file.

Simplified Prompt:

Build a single-file 16:9 HTML force-directed graph (no libraries) of JK Cement: company, business lines, plants, grinding units, states and brands as colour-coded nodes sized by capacity.

Add drag-to-pin, hover highlighting, zoom and pan, search, layer toggles, link-length and repulsion sliders, and a detail panel listing each node's connections.

Article – Org Chart



Prompt:

Build a single-file HTML interactive organisation chart for "Meridian Industries · Leadership structure · FY 2026". Fit it to a 16:9 page that scales to any window, using HTML, CSS and JavaScript.

THE ORG (sample data, about 35 people, Indian names)

- The CEO sits at the top. Five CXOs report to the CEO (CFO, COO, CHRO, CTO, CCO), each with VPs and Heads below them.

- Each person has a name, role, department, location, year joined and manager.

- There are 6 colour-coded departments: Executive, Finance, Operations, People, Technology, Commercial.

LAYOUT

- The top two levels run across the page as cards connected by elbow lines. Everyone below stacks as an indented list under their manager, so a wide org stays readable on one screen.

- Each card shows an initials avatar (or a photo), name, role, a department colour strip and a count of people below them. Click a card's toggle to expand or collapse its team, with smooth animation.

- The header has the title, a department legend (click a department to fade it out), a search box that finds a person or role and flies to them, Expand all, Collapse, zoom − / fit / +, and a light/dark theme toggle.

- Drag to pan and scroll to zoom. Hovering a card highlights its reporting line up to the CEO.

- Clicking a card opens a side panel with the person's details (role, department, location, since, reports to, team size) and a clickable list of their direct reports.

- A small stats box shows headcount, levels and spans, and a hint line reads "Click a card for detail · drag to pan · scroll to zoom".

EXCEL ROUND-TRIP

- "↓ Excel" downloads everyone as an editable spreadsheet with ID, Name, Role, Department, Reports To, Location, Since and Photo columns.

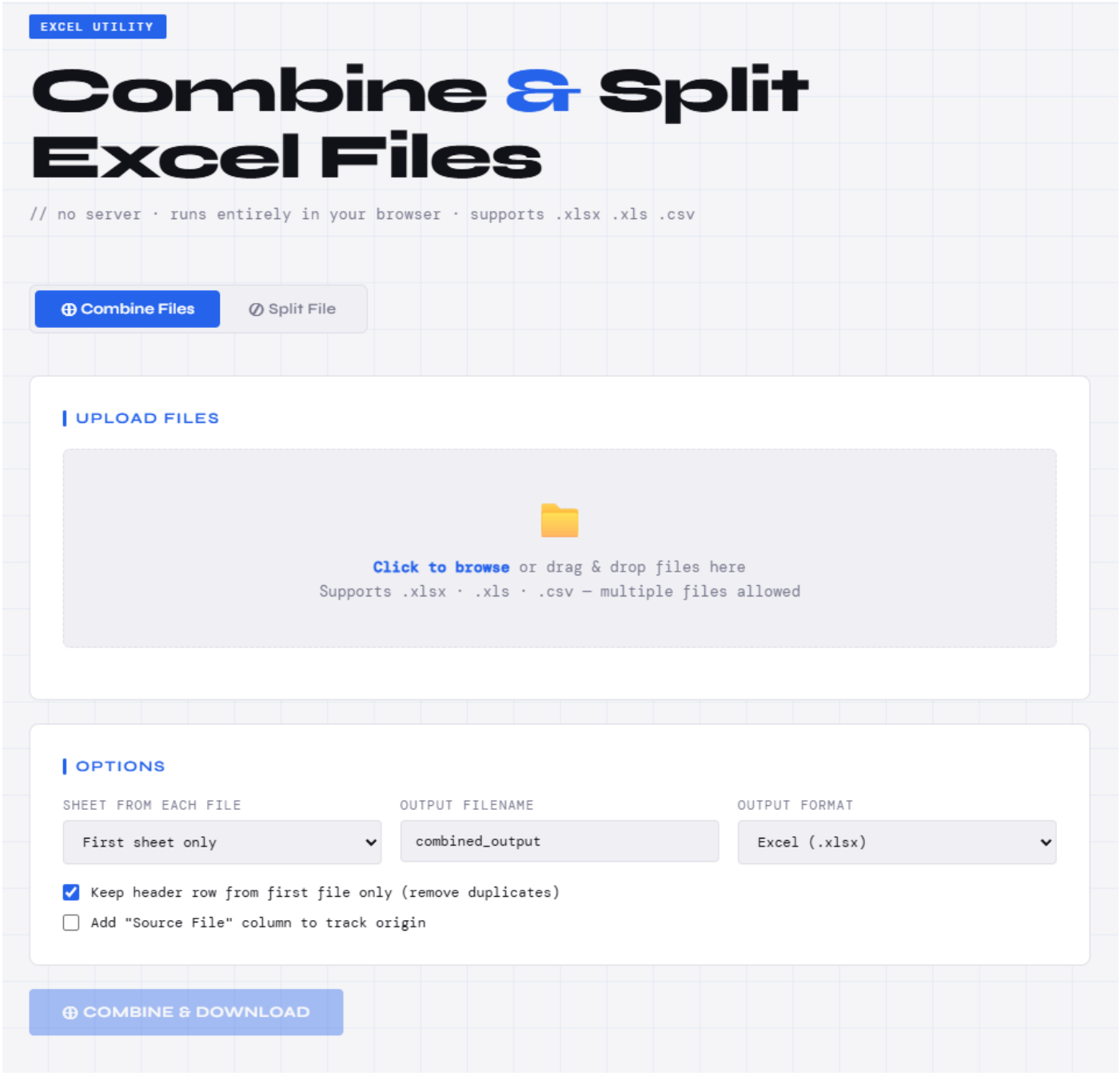
- "↑ Upload" rebuilds the chart from the user's own Excel or CSV. Match column headers loosely (e.g. "Manager" or "Reports to"). Check for missing managers, loops or multiple top people, and show a clear dialog listing any problems. Add new departments automatically.

- Use SheetJS (embedded or from cdnjs) to read and write the files.

Clean white cards, soft shadows, a toast for feedback, and dark mode. Deliver as one .html file.

Part — Media and Export

Article – Combine and Split



Article – Adding Images

Four Ways to Put an Image in an HTML Page

An HTML page can show an image in four main ways. The right choice depends on whether the page must travel as a single file. When you hand Claude your logo or a photo and ask for a web page, it picks one of these methods. Knowing them helps you ask for the right one, and explains why a page breaks when you move it.

1. Keep the image in a folder and reference it

The image sits as a normal file next to the HTML page, and the page points to it by name.

html

<img src="images/logo.png">

This is how most websites work. The HTML stays small and the image is easy to swap. The catch is that the page and the folder must always travel together. If you email only the HTML file, the image disappears and you get a broken-image icon.

2. Embed the image as code inside the HTML

The image file is converted into a long string of text, called base64, and placed directly inside the HTML.

html

<img src="...">

The browser turns that text back into the exact same picture, pixel for pixel. The image now lives inside the page, so one file carries everything. It works offline and survives email. This is what Claude usually does when you attach a logo. The cost is that the file grows by about a third, so this suits logos and a few small images, not a photo gallery.

3. Link to an image URL

The image is hosted somewhere online, and the page loads it from that web address.

html

<img src="https://yourwebsite.com/logo.png">

The HTML stays tiny, and the image can be shared across many pages. But the page needs internet, and if the link moves or is deleted, the image breaks. Many share links, such as Google Drive links, open a viewer page instead of the raw image, so they often fail here.

4. Inline SVG for logos and icons

Some images are not made of pixels but of drawing instructions, like “a navy circle here, a line there.” This format is called SVG, and most logos exist in it. Its code can be pasted straight into the HTML.

html

<svg viewBox="0 0 100 100"><circle cx="50" cy="50" r="40" fill="#1E2761"/></svg>

An SVG stays sharp at any size, is usually very small, and its colours can be changed with code. If your designer can give you an SVG version of your logo, this is the best option.

Which one to choose

Method	**Single file?**	**Works offline?**	**File size**	**Best for**
Folder reference	No	Yes	Small	Websites and projects with many images
Embedded (base64)	Yes	Yes	About 33% larger	Reports and pages you email or share as one file
Image URL	Yes	No	Small	Images already hosted online
Inline SVG	Yes	Yes	Very small	Logos, icons and simple graphics

A simple rule: if the page must travel as one file, embed the image or use SVG. If it lives on a website, use a folder or a URL.

When you don’t need an image at all

Simple shapes, badges and charts can be drawn with code instead, and standard icons can come from free icon libraries. Never ask AI to redraw your real logo this way, because it can only approximate it. AI looks at your image to understand it, but the exact copy comes from moving the original file, not from redrawing it.

AI BUILDS · ADDING IMAGES · PART 1 OF 2

Four ways to put an image in an HTML page.

When you hand Claude a logo or photo, it picks one of these. **The right choice depends on whether the page must travel as one file.**

METHOD 1

Folder reference

```
<img src="images/logo.png">
```

Good Small HTML; the image is easy to swap.

Watch Page and folder must travel together. Email the HTML alone and the image breaks.

METHOD 2 — CLAUDE'S DEFAULT

Embedded (base64)

```
<img src="…">
```

Good One file carries everything; works offline and survives email.

Watch File grows by about a third. Suits logos, not a photo gallery.

METHOD 3

Image URL

```
<img src="https://yourwebsite.com/logo.png">
```

Good Tiny HTML; one image shared across many pages.

Watch Needs internet. Drive share links open a viewer, not the image.

METHOD 4 — BEST FOR LOGOS

Inline SVG

```
<svg viewBox="0 0 100 100"><circle …/></svg>
```

Good Sharp at any size, very small, recolour with code.

Watch Needs an SVG version of your logo from your designer.

When you attach a logo, Claude usually embeds it.

One file carries everything, so the page still works after you email or move it.

HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

AI BUILDS · ADDING IMAGES · PART 2 OF 2

Must the page travel as one file?

Embed the image or use SVG.

If the page lives on a website, **use a folder or a URL instead.**

YES

It travels as one file

Reports, pages you email or share

Embedded (base64) **Inline SVG**

NO

It lives on a website

Projects with many pages and images

Folder reference **Image URL**

HOW THE FOUR METHODS COMPARE

Method	Single file?	Offline?	File size	Best for
Folder reference	No	Yes	Small	Websites with many images
Embedded (base64)	Yes	Yes	About 33% larger	Pages you email or share
Image URL	Yes	No	Small	Images already hosted online
Inline SVG	Yes	Yes	Very small	Logos, icons, simple graphics

When you don't need an image at all

Draw simple shapes, badges and charts with code, and take standard icons from free icon libraries.

Never ask AI to redraw your real logo. It can only approximate it. The exact copy comes from moving the original file.

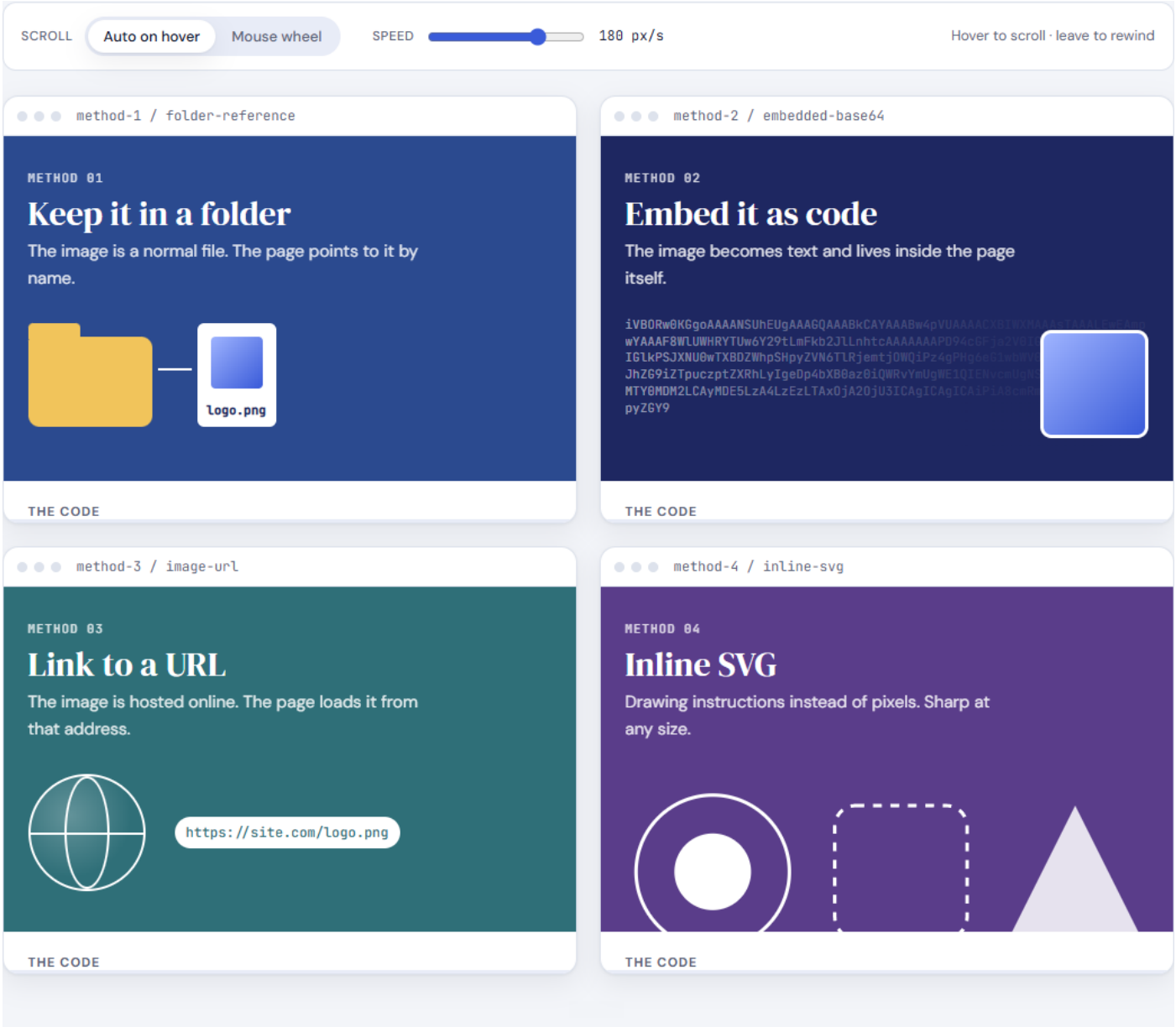
HMC ONE

AI STRATEGY. BUILT TO SCALE.

Source: HMC ONE

© Havish M Consulting and/or affiliates. All rights reserved.

Article – Hover vs Mouse Wheel Scroll



Prompt:

Build a single, self-contained HTML page: an interactive gallery of scrollable preview cards.

CONTENT

- Topic: [TOPIC, e.g. "4 ways to put an image in HTML"]

- Number of cards: [3–6]

- Each card is a tall mini-page about one item: [LIST THE ITEMS]

- Each mini-page has: a coloured hero (label, title, one-line summary, a small CSS/SVG illustration), then 3–4 short sections such as [e.g. code snippet / 3 quick facts / pros & cons / "best for"].

- Keep content limited and uncluttered: short sentences, plenty of white space, no filler.

- If I don't give content, write it yourself from the topic.

LAYOUT

- Title and one-line intro at the top.

- A control bar below it.

- Cards in a responsive grid (2 columns on desktop, 1 on mobile).

- Each card looks like a small browser window (three dots + a fake address) with a fixed-height frame (~340px) and a thin progress bar at the bottom showing how far it has scrolled.

INTERACTIONS

1. Hover to scroll: hovering a card scrolls its content downward smoothly; moving the mouse away rewinds it to the top.

2. Each card scrolls independently of the others.

3. Click to peek: clicking a card zooms it up from its position into a large centred pop-up (FLIP-style animation), with a dimmed backdrop. The pop-up content is larger and scrolls with the mouse wheel. Close with ×, Esc, or a backdrop click (it shrinks back into the card).

4. Speed slider: sets auto-scroll speed in px/second (range 20–240, default 80), with the live value shown.

5. Mode toggle: "Auto on hover" vs "Mouse wheel". In wheel mode, cards scroll manually and the slider is disabled. Show a small hint that changes with the mode.

STYLE

- [BRAND / COLOURS / FONTS, or "choose a clean, professional look"]

- Distinct hero colour per card, consistent everything else.

- Works in light and dark mode, keyboard accessible (Enter opens a card, visible focus), respects reduced motion.

- No external libraries; plain HTML, CSS and JavaScript in one file.

Article – Export Options

Most people who build something with AI stop one step too early. They get a working page — a dashboard, a tracker, a calculator — and then send a screenshot of it to their colleague. The page is good. The screenshot is not. It cannot be sorted, filtered, filed or forwarded, and the moment the underlying numbers change it becomes a lie.

The gap is not a technical one. A single HTML file, with nothing behind it, can hand the user an Excel workbook, a Word document, a PDF, a PowerPoint deck or a zip of all four. Very few people building with AI ask for this, because they do not know it is possible. It is worth knowing, because the export button is usually what turns a demo into a tool people actually use.

Why exports decide whether your tool survives

Every business tool sits inside a workflow that already exists. The finance analyst still has to paste the numbers into the month-end pack. The consultant still has to attach something to the email. The team lead still has to file the report.

If your page cannot produce the artefact that workflow expects, one of two things happens. Either the user retypes the output by hand, which they will resent and eventually stop doing, or they abandon the tool and go back to the spreadsheet. Neither failure looks like a technical failure. It looks like "we tried that AI thing and it did not stick".

An export button is therefore not a nice finishing touch. It is the join between your tool and the organisation around it.

The two tiers of export

Everything a browser can produce falls into one of two tiers, and the difference matters when you are specifying what you want.

Tier one needs nothing at all. Plain browser code can write a file and hand it to the user. This covers CSV, TSV, JSON, Markdown, HTML, images in PNG, JPEG or SVG, and a rich clipboard copy that pastes into Word or Excel with its formatting intact. It also covers PDF, through the browser's own print dialog and its Save as PDF option, which produces the sharpest text of any method available.

Exhibit: Formats a browser can produce on its own

Format	What it gives you	Worth knowing
CSV	The data, ready for any other system	Add a byte order mark or Excel mangles ₹ and accented names
TSV	Tab-separated data	Pastes into a spreadsheet cleanly

JSON	Structured data for another tool or script	For handing to a developer, not to a colleague
Markdown	A table for notes, wikis and documentation	Plain text, survives anywhere
HTML	A standalone page anyone can open	No software needed at the other end
Excel (.xls)	A formatted sheet, via an HTML table saved with an Excel file type	Excel shows a one-time format warning. Fine internally, not for clients
Word (.doc)	Headings and tables, via the same trick	Same warning, same rule
Selectable-text PDF	The browser's print dialog, Save as PDF	Sharpest text of any method, and free
SVG	A chart as true vector	Opens in Illustrator, Figma and PowerPoint, stays sharp at any size
PNG / JPEG	A chart as an image	Render at 2x so it holds up on a high-resolution screen
Clipboard, rich text	A table that pastes into Word or Excel with formatting intact	The most-used export in practice, and the most overlooked
Clipboard, image	A chart that pastes straight into a slide	Saves the save-then-insert round trip
.ics / .vcf	Calendar invite, contact card	Same technique, different text format

There are two useful tricks in this tier. A table written as HTML but saved with an Excel file type opens in Excel as a formatted sheet, and the same trick with a Word file type opens in Word with headings and tables intact. Both show a one-time warning that the file format does not match the extension. For a quick internal deliverable that warning is a fair trade. For anything going to a client, it is not.

Tier two needs a library. A genuine .xlsx with multiple sheets and no warning, a genuine .docx, a PDF you lay out yourself, an editable PowerPoint deck, a zip bundle of several files at once. These need a code library, typically half a megabyte to a megabyte each.

Exhibit: Formats that need a library embedded in the file

Format	**Library**	**Approximate size**	**What it adds over the free version**
Excel (.xlsx)	SheetJS	900 KB	A genuine workbook, multiple sheets, no warning dialog

Excel with full styling	ExceIJS	1 MB	Colours, borders, column widths, frozen panes
Word (.docx)	docx.js	700 KB	A genuine Word document with proper headings, tables and page breaks
PDF, vector	jsPDF with AutoTable	400 KB	You control the layout, text stays selectable, long tables split across pages
PDF, screenshot	html2canvas with jsPDF	600 KB	Pixel-identical to the screen, but the text becomes an image
PowerPoint (.pptx)	PptxGenJS	500 KB	Editable slides with real tables, not pictures of tables
Zip bundle	JSZip	100 KB	One download instead of five

The important point is that the library can be embedded inside the HTML file itself. Ask for that explicitly, because the default behaviour is to load libraries from the internet, and the day someone opens your file on a locked-down corporate laptop or on a plane, the page breaks. A self-contained file is larger and completely reliable. In a business setting, take the trade.

Which format for which job

The instinct is to offer every format. Resist it. Each format carries an implicit instruction about what the recipient should do next, and offering all of them signals that you did not think about it.

Exhibit: Matching the format to the situation

The situation	**The format**	**Because**
The data is going into another system or model	CSV	The most useful format on the list, and the most commonly forgotten
The recipient will re-sort, extend or recut the numbers	Excel	Put the summary on its own sheet
The deliverable is an argument, and someone will edit the wording	Word	Structure matters more than the data
The content is final and should not change	Selectable-text PDF	The default for anything leaving the organisation
The material will be presented	PowerPoint	An editable table beats a picture of a table
One chart needs to travel into someone else's deck	SVG, or PNG at 2x	The vector version stays sharp when resized

The recipient needs several of the above	Zip	One click instead of five
The output is going into an email	Rich-text clipboard copy	The most common action of all, and the biggest time saver

Two limits worth being clear about

The first is deliberate: everything described here happens on the user's own machine. No data is uploaded, nothing passes through a server. When the content is client data or salary figures or anything under an NDA, that is not a limitation, it is the main argument in favour of the approach.

The second is real: a browser file cannot save to a shared drive, post to a team channel or send an email on its own. Those require credentials, a connection and a system to connect to. The honest workaround is to export the file and let the person put it where it belongs. If a tool genuinely needs to write into a live system, it has outgrown a single HTML file, and that is a different conversation.

How to ask for it

The pattern is short, and it works whichever AI tool you use:

> Add an export bar to this page. Excel with the data on one sheet and the summary on another, a PDF with the chart and the table, and a CSV. Embed the libraries in the file so it works with no internet. Show the filename after each download.

Three things in that instruction do the work. Naming the formats, rather than asking for exports in general. Saying what goes into each one, so the Excel file is not just the screen dumped into a grid. And requiring that it works offline, which is the single request most people leave out and most often regret.

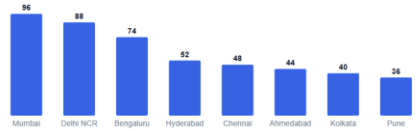
The takeaway

The value of a tool is not the analysis it performs. It is the artefact somebody else can act on. A page that produces a clean Excel file and a clean PDF will be used long after a more sophisticated page that only produces a view on screen.

Build the export before you polish the interface. It is the part that determines whether the work leaves your laptop.

Q3 FY26 revenue by city

₹ Cr · sample data



CITY	REGION	UNITS	REVENUE	MARGIN %	GROWTH %
Mumbai	West	18,240	96	38.2	+12.4
Delhi NCR	North	16,980	88	36.5	+9.8
Bengaluru	South	14,120	74	41	+18.6
Hyderabad	South	10,450	52	39.4	+15.2
Chennai	South	9,380	48	35.1	+6.4
Ahmedabad	West	8,610	44	34.8	+11.1
Kolkata	East	7,920	40	32.6	-2.3
Pune	West	7,140	36	37.9	+8.7
Total		**92,840**	**478**		

No library needed

Plain JavaScript

- **CSV** — **CSV file** — Comma separated, with a BOM so Excel reads ₹ correctly
- **TSV** — **Tab separated** — Pastes straight into a spreadsheet
- **JSON** — **JSON data** — For feeding another tool or script
- **MD** — **Markdown table** — For notes, wikis and README files
- **HTML** — **HTML file** — A standalone page anyone can open
- **XLS** — **Excel, legacy trick** — HTML table with an Excel MIME type. Excel warns once on open
- **DOC** — **Word, legacy trick** — Same idea for Word. Keeps headings and tables
- **PDF** — **PDF via print dialog** — Choose Save as PDF. Sharpest text, no library
- **SVG** — **Chart as SVG** — True vector, opens in Illustrator or PowerPoint
- **PNG** — **Chart as PNG** — Rendered at 2x through a canvas
- **JPEG** — **Chart as JPEG** — Smaller file, no transparency
- **COPY** — **Copy table as rich text** — Paste into Word or Excel with formatting intact
- **COPY** — **Copy chart as image** — Paste straight into a slide

Library embedded in this file

Still works offline

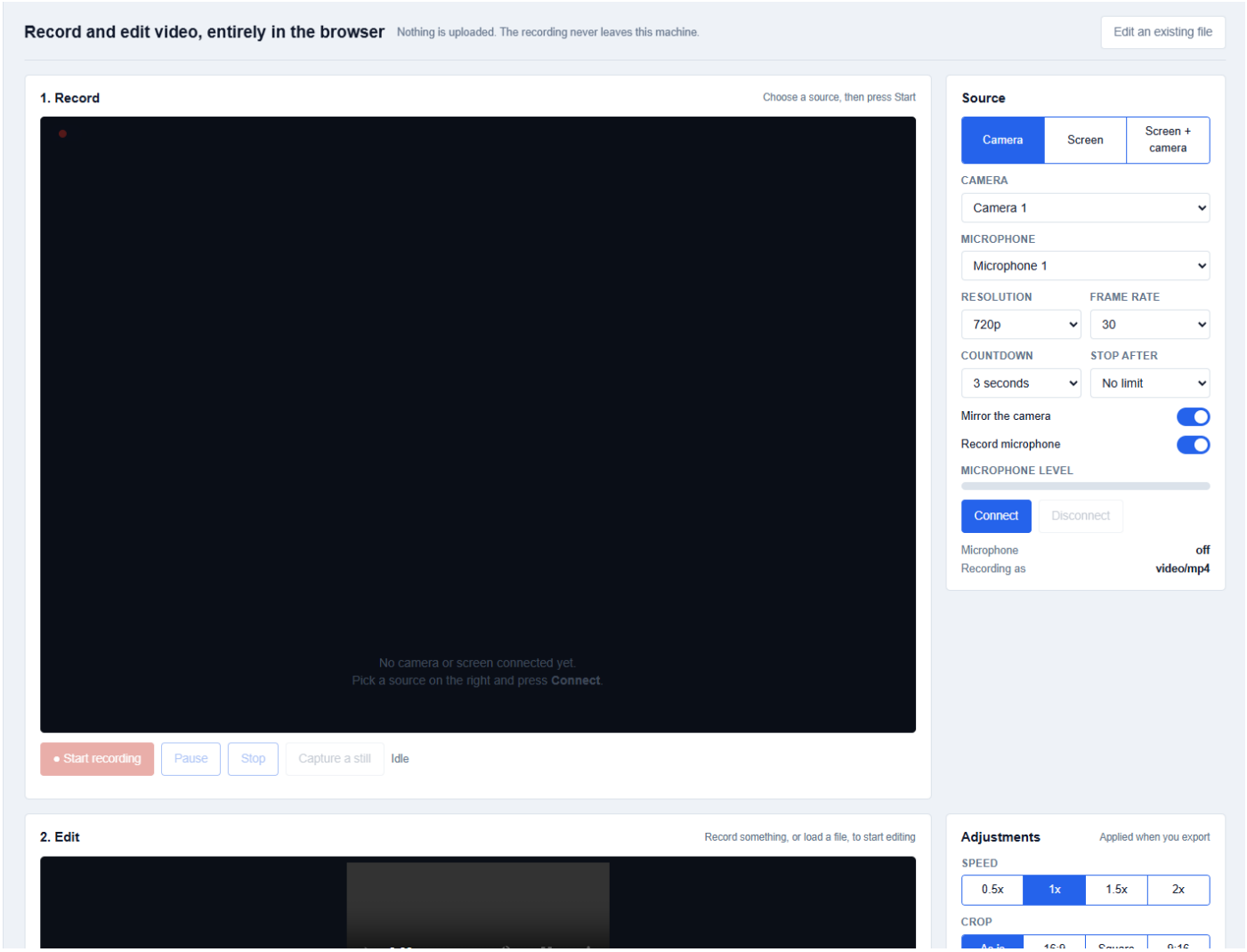
- **XLSX** — **Real Excel workbook** — SheetJS · two sheets, no warning dialog
- **DOCX** — **Real Word document** — docx.js · headings and a proper table
- **PDF** — **PDF, vector** — jsPDF and autoTable · text stays selectable
- **PDF** — **PDF, screenshot of the page** — html2canvas · pixel perfect, text becomes an image
- **PPTX** — **PowerPoint deck** — PptxGenJS · editable table plus the chart
- **ZIP** — **Everything in one zip** — JSZip · CSV, JSON, chart PNG and the PDF

Exports from this session

2 files

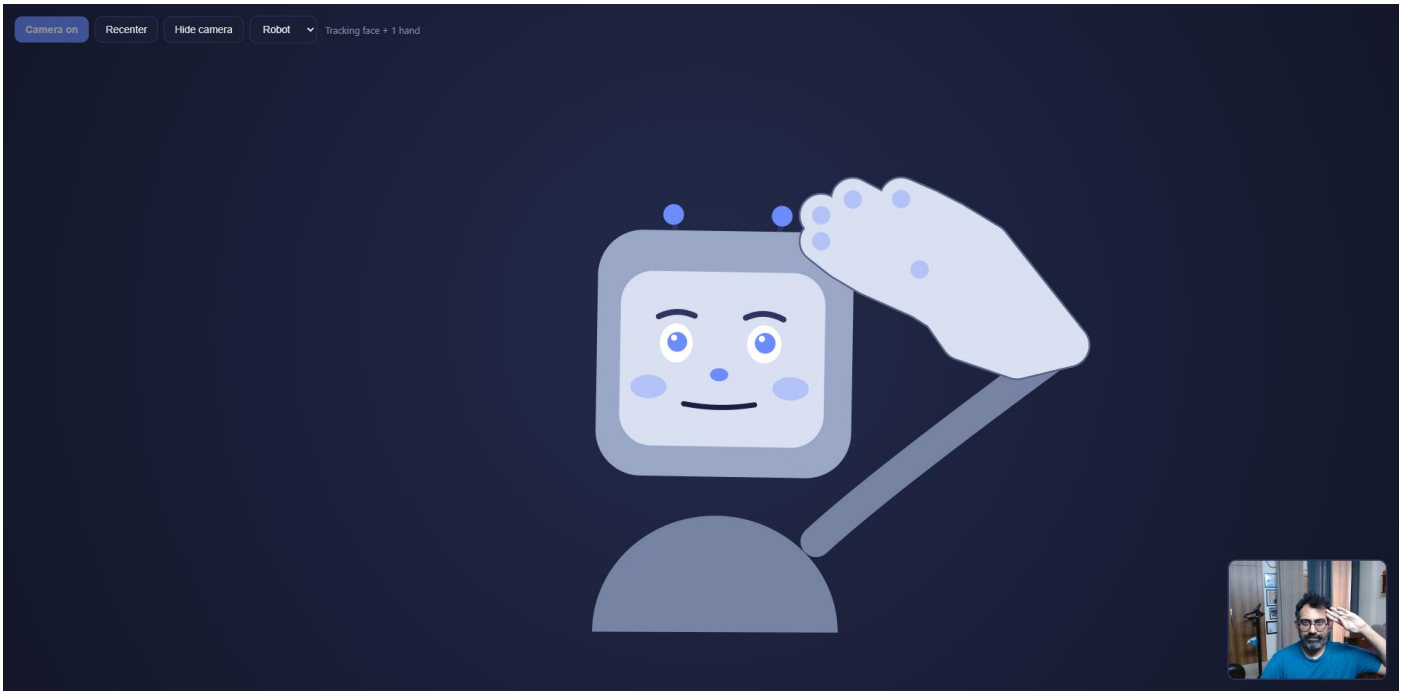
revenue.pdf	13.4 KB
revenue.pptx	176.4 KB

Article – Record Video



Article – Facial Avatar





Article – Browser Cache

What the Browser Keeps: Cache and Storage in a Single HTML File

There is a moment that happens to almost everyone who starts building with AI. You change something in your file, save it, refresh the browser, and nothing changes. The old version is still on screen. You change it again. Still nothing. After ten minutes of this you conclude the AI broke something.

It did not. The browser is showing you a copy it kept from earlier. Understanding what the browser keeps, and why, is one of those small pieces of knowledge that saves hours of confusion and quietly makes your tools better.

There are two separate things at work, and they get confused constantly. The first is the cache, which is how the browser stores the file itself. The second is storage, which is how your page remembers what the user did. Different mechanisms, different rules, different problems.

Part one: the cache, or why your changes do not appear

When a browser loads anything — a page, an image, a code library — it keeps a copy on disk. The next time that same thing is needed it uses the copy rather than fetching it again. This is why the second visit to a website feels instant.

The rule is that the browser decides how long to trust its copy, based on instructions from the server. For a file you are editing on your own machine, that logic still applies, and it is usually the reason your edits seem to vanish.

The fix takes two seconds once you know it: a hard refresh. **Ctrl+Shift+R** on Windows, **Cmd+Shift+R** on a Mac. That tells the browser to discard what it has and fetch everything again. If you are going to be iterating on a file for an hour, open the developer tools with F12 and tick "Disable cache" in the Network tab, which keeps the browser honest for as long as the tools are open.

Worth knowing: this is also why a page that loads a library from the internet works fine all week and then breaks on a plane. The library was never in your file. It was in the browser's cache, and the cache expired.

Part two: storage, or how your page remembers things

The cache is about the file. Storage is about the contents — the numbers someone typed, the tab they had open, the draft they had not finished. A single HTML file has four places to keep them, and choosing the wrong one is the most common mistake in this area.

Exhibit: The four places a page can keep data

Where	Survives a reload	Survives closing the tab	Shared between tabs	Rough limit	Use it for
A variable in the code	No	No	No	Whatever memory allows	Anything you are happy to lose
sessionStorage	Yes	No	No	About 5 MB	One sitting's work: the open tab, a filter, a step in a form
localStorage	Yes	Yes	Yes	About 5 MB	Preferences, drafts, the last version of a report
IndexedDB	Yes	Yes	Yes	Hundreds of MB	Large datasets, files, images, anything past a few MB

Two of these deserve a comment.

localStorage is the default choice and the one you will use ninety per cent of the time. It holds text, it is simple, and it survives a browser restart. Its limit is around five megabytes, which is far more than it sounds — a few thousand rows of a table fit comfortably.

IndexedDB is what you move to when you outgrow that: a year of daily records, uploaded images, a cached dataset. It is more work to use, which is why it is worth naming explicitly when you ask for it, rather than hoping it appears.

The boundary nobody explains

Storage is tied to the origin, which means roughly the website address the page came from. Data saved by a page on one address cannot be read by a page on another. This has three consequences that surprise people:

- Data does not travel with the file. If you email your report tool to a colleague, they receive the tool, not your saved work. Their browser starts empty.
- The same file in two places is two different stores. Open it from your Downloads folder, then open it from a shared drive, and the second copy will not see what the first one saved.
- Clearing browsing data clears it. If the user tidies up their browser, or opens the page in a private window, everything the page remembered is gone.

None of this makes storage a bad idea. It makes it the wrong place for anything that matters. The rule is simple: **storage is a convenience, not a record.** Anything the user would be upset to lose must be

exportable to a file. This is why the save button and the export button are two different things and you need both.

What this means in practice

Exhibit: Matching the mechanism to the need

What you want	Use	Why
Remember the user's preferences and filters	localStorage	Small, simple, survives a restart
Keep a draft while they work	localStorage, saved as they type	Costs almost nothing, prevents the worst outcome
Remember which tab or step they were on	sessionStorage	Should not follow them into tomorrow
Hold a large dataset the page loaded	IndexedDB	Past five megabytes, localStorage fails
Hold anything the user cannot afford to lose	An exported file	Storage is not a record and never was
Make your page work offline for good	Embed the libraries in the file	The cache is not a guarantee, only a convenience

How to ask for it

The instruction is short:

> Save the user's edits to localStorage as they type, and reload them when the page opens. Add a Reset button. Tell the user plainly that the data is saved in this browser only and add an export to file.

The last clause is the one people leave out, and it is the one that prevents the complaint that always follows: *I lost everything*.

The takeaway

The cache is the browser being efficient with the file. Storage is your page being useful with the data. Both are working exactly as designed, and both look like bugs the first time they surprise you.

Two habits cover almost every case. Hard refresh while you build, so you are always looking at what you actually wrote. And treat anything the browser remembers as a courtesy to the user, never as the place their work lives.

This page was served
Local file **read from disk** file:// pages skip the network cache

Navigation type
First load reload, first load or history

Load time
not reported from request to load event

Page size
15.9 KB after decoding

Address
file: storage is tied to this origin

Reload page | How to hard refresh | Open in a new tab | Clear everything

Variable in the code

The simplest store, and the one people use by accident

Survives a reload	**No**
Survives closing the tab	**No**
Shared between tabs	**No**
Rough limit	**Memory**

0 page loads recorded here

Empty. Nothing survived, or nothing was saved yet.

Type something to save

Save | Read | Clear

sessionStorage

One sitting's work, in this tab only

Survives a reload	**Yes**
Survives closing the tab	**No**
Shared between tabs	**No**
Rough limit	**~5 MB**

0 page loads recorded here

Empty. Nothing survived, or nothing was saved yet.

Type something to save

Save | Read | Clear

localStorage

The default choice for preferences and drafts

Survives a reload	**Yes**
Survives closing the tab	**Yes**
Shared between tabs	**Yes**
Rough limit	**~5 MB**

7 page loads recorded here

hello
saved 23/09/2026, 23:04:43
74 B of text

Type something to save

Save | Read | Clear

IndexedDB

For datasets and files too big for the rest

Survives a reload	**Yes**
Survives closing the tab	**Yes**
Shared between tabs	**Yes**
Rough limit	**Hundreds of MB**

7 page loads recorded here

hello
saved 23/09/2026, 23:04:45
74 B of text

Type something to save

Save | Read | Clear

What happened

Newest first

23:10:31 This origin may store up to about **10240.00 MB** in total, with 5.1 KB used.
23:10:31 IndexedDB: **survived**, 7 loads recorded.
23:10:31 localStorage: **survived**, 7 loads recorded.
23:10:31 Page opened. Opened as a local file.

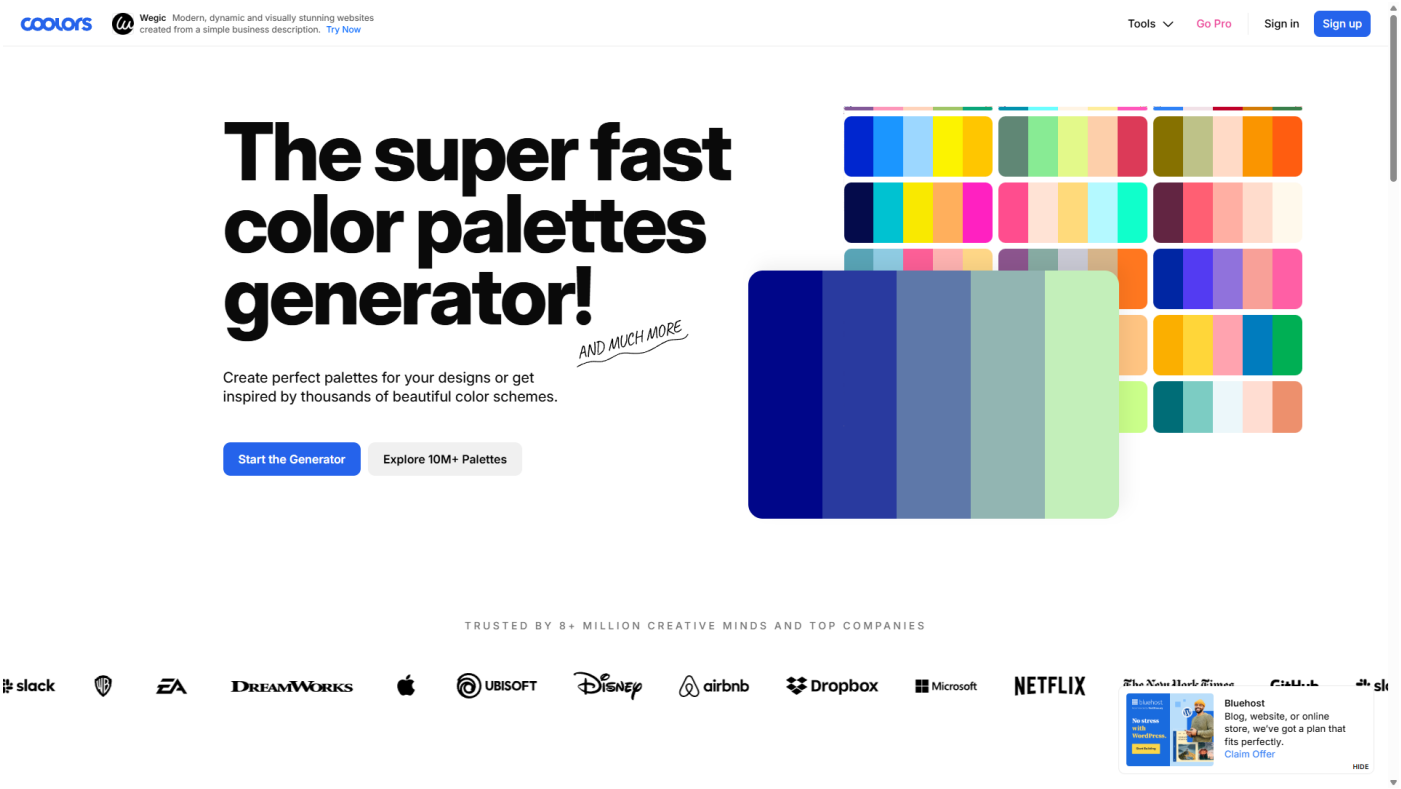
Two habits worth keeping

- **While you build:** hard refresh with `Ctrl` + `Shift` + `R`, or `Cmd` + `Shift` + `R` on a Mac, so you see the file you actually saved.
- **When you ship:** treat everything above as a convenience. Storage belongs to this browser on this machine, it does not travel with the file, and a private window or a cleared history wipes it. Anything the user cannot afford to lose needs an export to a file.

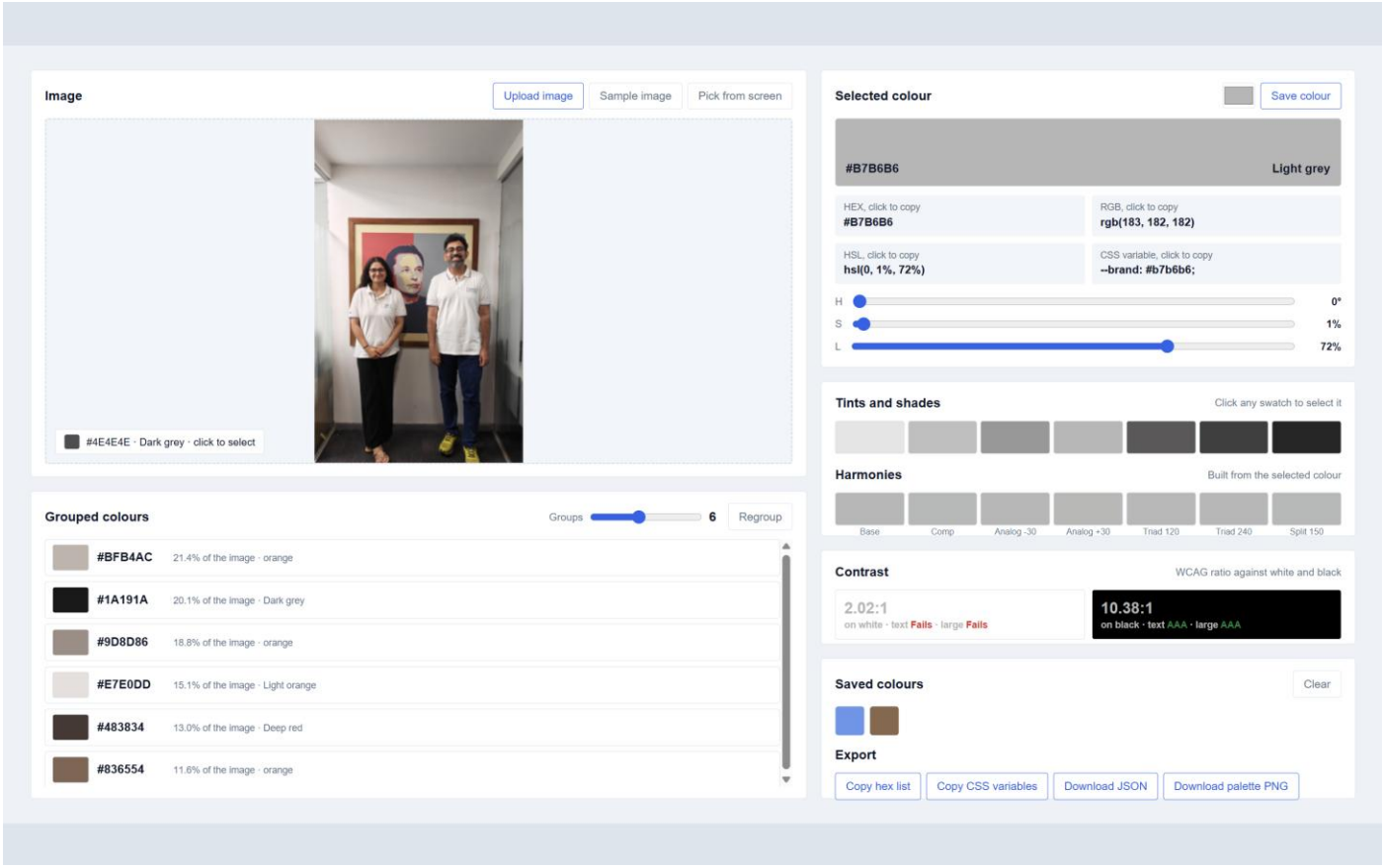
Part — Branding and Visual Style

Article – Brand Colors

https://coolors.co/



Article – Color Picker



Part — Links

Article – Vibe Coding Platforms

https://replit.com

https://lovable.dev

https://bolt.new

https://app.emergent.sh

Gallery



IIM KOZHIKODE



NUS BUSINESS SCHOOL - SINGAPORE



SP JAIN DUBAI



MASTERS' UNION





BHARAT PETROLEUM – KOCHI REFINERY



NUVOCO CEMENT





JINDAL INDIA





BAHRI MAZROEI TRADING COMPANY – DUBAI





LANDMARK GROUP – DUBAI





AMBER ENTERPRISES



youtube.com/c/havishmadhvapaty

havishmconsulting.com