ICT for Civic Data — Turin University 2025-26

Analysis and Storytelling

Crash Course — Day 4

civicliteraci.es

Day 4

> Today's plan

Morning:

- Finish cleaning: categorise the MainCause column
- Learn about skills: save your map process for reuse
- Build a visualisation page with charts
- Build a landing page connecting everything

Afternoon:

- Apply everything to your own angle
- Peer review: present your story and get feedback

Finishing the Cleaning

> Too many categories to visualise

The MainCause column has **~260 unique values** after cleaning: Heavy rain, Monsoonal rain, Tropical Storm Irene, Tropical Storm Harvey, etc.

A bar chart of 260 categories is unreadable. We need to group them into **7–8 broad categories**.

Category	Includes
Heavy rain	Heavy rain, Torrential rain, Brief torrential rain
Tropical storm	All named tropical storms and cyclones
Monsoonal rain	Monsoonal rain and variants
Snowmelt	Snowmelt, rain and snowmelt
Dam break	Dam break, dam release, dam failure
Ice/avalanche	Ice jams, avalanche
Other	Tsunami, storm surge, landslide, etc.

> Step 1: create a mapping sheet

In your Google Sheets FloodArchive file:

1. Create a new sheet called **"Mapping"**
2. In cell A1, type a header: `original_cause`
3. In cell A2, use: `=UNIQUE(MainSheet!E:E)` (adjust the column reference to your MainCause column)
4. This gives you every unique MainCause value in one list
5. **Sort** the list (Data → Sort) so similar values are next to each other

> Step 2: add categories

In column B of the Mapping sheet, add a header: `category`

Go through the list and type the category for each value:

- All "Heavy rain" variants → `Heavy rain`
- All "Tropical Storm …" → `Tropical storm`
- All "Monsoonal …" → `Monsoonal rain`
- etc.

This is a **human decision**. You decide what belongs where. The mapping sheet documents every decision.

> Step 3: apply the mapping with FILTER

Back in your main data sheet, add a new column: `category`

In the first data row, use:

```
=FILTER(Mapping!B:B, Mapping!A:A = E2)
```

This looks up the MainCause value in the Mapping sheet and returns the category.

Copy the formula down to all rows.

> Alternative: categorise with IFS and SEARCH

Instead of a mapping sheet, you can categorise directly with a formula:

```
=IFS(
  IFERROR(SEARCH("heavy rain",E2),0)>0, "Heavy rain",
  IFERROR(SEARCH("cyclone",E2),0)>0, "Tropical storm",
  IFERROR(SEARCH("typhoon",E2),0)>0, "Tropical storm",
  IFERROR(SEARCH("monsoon",E2),0)>0, "Monsoonal rain",
  IFERROR(SEARCH("snow",E2),0)>0, "Snowmelt",
  IFERROR(SEARCH("dam",E2),0)>0, "Dam break",
  TRUE, "Other"
)
```

`SEARCH` finds text within a cell. `IFERROR` handles cases where the text is not found. `IFS` checks conditions in order.

Both approaches work. FILTER + mapping sheet is more transparent. IFS + SEARCH is faster for large datasets.

> Step 4: keep only what you need

Your spreadsheet now has many columns. For the visualisation, you only need a few.

1. Create a **new sheet**
2. Copy only the columns you need: Country, Began, Ended, Dead, Displaced, Severity, **category**
3. Paste as **values only** (so formulas become plain data)
4. Download as CSV: File → Download → Comma Separated Values
5. Upload the CSV to your Codespace

This clean, categorised CSV is what the charts will read.

> Step 5: investigate ambiguous values

We identified this on Day 3 but haven't acted on it yet.

Look at the **OtherCountry** column. It contains a mix of:

- `0` (3,382 rows) — does this mean "no other country affected" or "unknown"?
- Blank (1,344 rows) — is this different from 0?
- Actual country names (the remaining rows) — a flood that affected multiple countries

One approach: create a helper column `multi_country` that turns this into a clean **Yes/No**.

Make a decision, apply it consistently, and write it down.

> Three ways to write the same logic

All three produce a `multi_country` column with Yes or No:

Nested IF:

```
=IF(D2="0", "No", IF(D2="", "No", "Yes"))
```

IFS (checks conditions in order, returns first match):

```
=IFS(D2="0", "No", D2="", "No", TRUE, "Yes")
```

IF with OR (most readable here):

```
=IF(OR(D2="0", D2=""), "No", "Yes")
```

There are often multiple ways to reach the same result. Sometimes they are equivalent, like here. Other times one is much easier to use than another. Ask your chatbot which formula fits your situation.

> Validating multi-country entries

For rows where OtherCountry has a real country name, do the two countries actually share a border?

Brazil + Vietnam? Suspicious. Brazil + Argentina? Plausible.

The **REST Countries API** has a `borders` field:

```
https://restcountries.com/v3.1/name/brazil?fields=borders
```

Returns: `["ARG", "BOL", "COL", "GUY", "PRY", "PER", "SUR", "URY", "VEN"]`

This is geographic knowledge that formulas cannot provide. An API can.

> Data cleaning involves ethical choices

Standardising country names is not always straightforward:

- **Scotland, UK, United Kingdom** — is Scotland its own entity or part of the UK?
- **USSR, Soviet Union, Russia** — historical entities that no longer exist
- **Palestine, Gaza** — politically contested boundaries
- **Czechoslovakia, West Germany, East Germany** — countries that split or merged

There is no universal correct answer. **It depends on what you use the data for.** Document your choices and your reasoning.

> Core vs peripheral competencies

Core (build skill here)	Peripheral (AI can handle)
> Data cleaning and validation	> Writing HTML and CSS
> Formula logic and spreadsheet work	> Memorising git commands
> Judging whether results make sense	> Producing visual polish
> Understanding what the data represents	> Code syntax
These require **your judgment**. AI cannot replace it.	These are mechanical. AI does them well.

Best AI use: ask **"what function would I use for this?"** — not "do it for me."

> Build good taste in data

If you rely on AI to do all the data work, you will never develop **intuition** for what right data looks like and what wrong data looks like.

- > Sort the column. Scroll through. Does anything stand out?
- > Check the min and max. Do they make sense?
- > Count the categories. Are there too many?

This is why we do manual spreadsheet work, even when AI could do it faster. **You are building judgment**, not just producing output.

> Principles of data cleaning (1/2)

Cleaning is not a single operation. It is a sequence of **decisions**, each documented.

- **Never modify the original file.** Work on a copy. Keep the raw data intact.
- **Never let AI modify data content directly.** AI can write scripts that transform data (filter rows, create columns), but should not edit values by itself. If the logic can't be expressed as a script or formula, do it manually.
- **Look at the data yourself.** Open it in a spreadsheet, sort, filter, scroll. You need intuition about the data before you can clean it or visualise it.

> Principles of data cleaning (2/2)

- **Create new columns rather than overwriting.** Use helper columns with formulas (TRIM, IF, FILTER) instead of modifying values directly with Find and Replace. The original stays intact, and your logic is visible in the formula.
- **One problem at a time.** Don't try to fix everything in one pass.
- **Use the right tool for the job.** Some tasks are spreadsheet work, some need OpenRefine, some need scripts.
- **Document every decision.** If someone asks "why did you change this value?", you should be able to point to a record.
- **Verify after each step.** Count rows, check totals, filter to confirm.

> Recap: the cleaning process

Step	What	Tool / technique
Explore	Sort, filter, look for problems	Google Sheets
Trim whitespace	Remove invisible spaces	Google Sheets automated suggestion + TRIM formula in a helper column
Standardise values	Fix typos, merge variants	OpenRefine clustering, or filter-and-overwrite in Google Sheets
Investigate ambiguity	Decide what 0 or blank means	IF/IFS/OR formulas to create clean helper columns
Categorise	Group many values into few	Mapping sheet (UNIQUE + FILTER) in Google Sheets
Reduce	Keep only needed columns	New sheet, paste as values only, export CSV

Choosing the Right Chart

> What is your chart trying to do?

The chart type depends on the **task**, not the data format.

Task	Chart type
Show change **over time**	Line chart
Compare categories	Bar chart
Show **parts of a whole**	Pie or donut chart (use sparingly)
Show **distribution**	Histogram
Show **relationship** between two variables	Scatter plot
Show **geographic** patterns	Map

If you are unsure, a **bar chart** is almost always a safe default.

> Storytelling vs exploration

Storytelling

For your **proposal** and presentations.

- > Simple, one message per chart
- > Easy to understand in one glance
- > Minimal interactivity
- > Title states the finding, not the topic

"Flood events have doubled since 2000" — not "Flood events over time"

Exploration

For your own **analysis** and investigation.

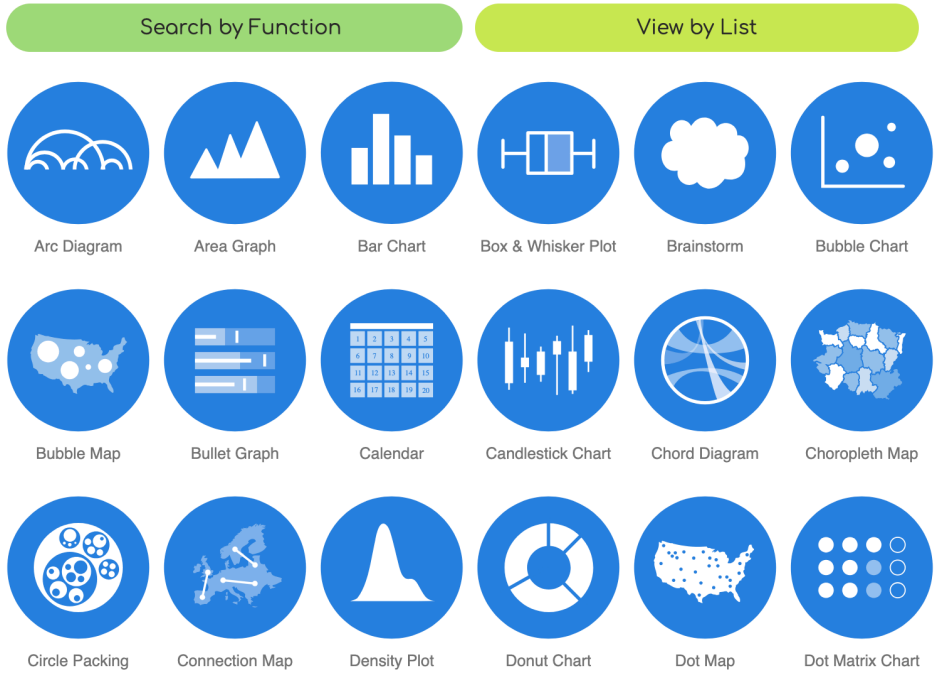
- > More complex, multiple dimensions
- > May need legends, tooltips, filters
- > Interactive (zoom, filter, hover)
- > Title describes the data, not the conclusion

"Flood events by country and year" — inviting the viewer to explore

For the proposal: **storytelling**. For your data portal: **exploration** is fine.

> A useful reference

datavizcatalogue.com — browse by function to find the right chart for your task.



Building the Visualisation Page

> What we are building

A **single HTML page** with multiple charts showing different views of the FloodArchive data:

- Flood events by **country** (top 20)
- Flood events **per year** (trend)
- Breakdown by **category** (the 7 groups you just created)
- Total displaced by **severity**

Presented as **cards**: each chart in its own box, arranged on the page. Cards are a good default for presenting multiple pieces of information with structure.

> Being precise with your prompt

Just like asking for a "simple Leaflet map," be specific about visualisations:

- Name the **library**: "use Chart.js" (works in a single HTML file, no installation needed)
- Name the **chart types**: "a bar chart for countries, a line chart for years"
- Name the **layout**: "present each chart as a card on the page"
- Name the **data source**: "read the CSV from my repo"

If you say "make me some visualisations," the agent will guess and you will waste tokens.

> Prompt for the visualisation page

« Build a simple HTML page using Chart.js that reads my cleaned FloodArchive CSV. Create four charts presented as cards on the page: (1) a bar chart of flood events by country, top 20, (2) a line chart of flood events per year, (3) a bar chart of events by category, (4) a bar chart of total displaced by Severity. When you're done, publish it on GitHub Pages at a different URL than the map, from the main branch. I'm a beginner, please explain what you're doing in simple terms. »

Objective | Steps | Output shape | Reasoning

> Exercise: build and publish the charts page

Individual work — 30 minutes

1. Make sure your cleaned, categorised CSV is in the Codespace
2. Ask Gemini to build the Chart.js page (use the prompt above)
3. Push and check the live page at `your-repo/charts.html`
4. Refine: adjust colours, labels, chart types if needed

If the charts look wrong, check the data: are categories consistent? Is the CSV loading correctly?

Better Design with v0

> v0 by Vercel

v0.dev is an AI-powered webpage creator that generates polished web pages from a description.

- Describe what you want → get a professional-looking page
- Good for **landing pages, dashboards, layouts**
- Much better at visual design than Gemini CLI

The key trick: always ask for **"pure HTML, CSS, and JavaScript in a single file."** By default v0 produces overly complex code that won't work on GitHub Pages. The single-file constraint keeps it simple and compatible.

> Using v0 for this project

Two uses:

1. **Landing page design**: describe your portal layout to v0, get a polished single-file HTML page, upload it to your Codespace and ask Gemini to use it as a starting point
2. **Map page redesign**: upload your current map HTML to v0 and ask it to improve the design

In both cases: generate the design in v0, download the HTML, upload to your Codespace, then use Gemini to connect it to your data.

The Landing Page

> A mini open data portal

Your landing page is the **entry point** to your project. It lists:

- **Project title and description** — what is this about?
- **Datasets** with metadata: what data, where from, what format, when retrieved
- **Navigation** — links to the map and charts pages

Think of it as a one-page open data portal: someone landing on this URL should understand what the project is, what data it uses, and where to explore further.

> Prompt for the landing page

« I want a landing page for my project that is the main page when someone visits my GitHub Pages site. It should have: a title and short description of the project, a section listing the datasets I used with metadata (name, source, format, date retrieved), and a top navigation menu linking to the map page and the charts page. Use a clean, card-based layout. When you're done, publish it on GitHub Pages from the main branch. Explain how you chose to present the information. »

Objective Steps Output shape Reasoning

> Exercise: build and publish the landing page

Individual work — 20 minutes

1. Ask Gemini to rename and create the landing page
2. Fill in the dataset metadata (check your sources.md)
3. Push and check all three URLs work:
    4. `your-repo/` → landing page
    5. `your-repo/map.html` → map
    6. `your-repo/charts.html` → charts
7. Click through the navigation to verify links

> Data privacy: separate the data from the interface

When working with sensitive data, **never send the real data to AI tools**.

Instead:

1. **Build the interface first** with anonymised sample data (first 5 rows, randomised coordinates)
2. Share only the sample with the AI
3. Connect the interface to the real data afterwards

Self-contained HTML pages process data **entirely in the browser**. Your data never leaves your device, even when the page is published.

Rule of thumb: any tool you use for free, don't put sensitive data in it.

> Clean up your repository

Before we move on, let's tidy up. Ask Gemini:

> "Clean up the repository so that files have consistent, self-explanatory names and are organised tidily."

A clean repo is easier for you, for collaborators, and for the AI. If the AI can understand your file structure at a glance, it works better.

Skills:
Saving Your Process

> What is a skill?

A **skill** is a saved procedure that tells the AI how to do a specific task. A guide and a shortcut.

Instead of explaining every detail each time, you save the process once. Next time, you say "use the map skill" and the AI follows the saved procedure.

A skill is a **markdown file** with:

- A **name** and **description** (so the AI knows when to use it)
- The **steps** to follow
- Any **scripts or templates** (so it doesn't have to recreate them)

> Skill scope

Workspace scope

Available **only in this project folder**.

Good for project-specific procedures: "how we build maps for this disaster preparedness project."

Global scope

Available **across all projects**.

Good for general procedures: "how I build any Leaflet map" or "how I clean CSV files."

Be selective. The more skills the AI has loaded, the more confused it can get. Keep skills focused and relevant.

> Exercise: save your map process as a skill

Resume your Gemini session:

```
gemini --resume
```

Then ask:

> "Can you save the process we used to create the Leaflet map as a skill? Include the steps for getting data, creating the map, improving the design, and publishing to GitHub Pages. I want you to follow that process every time I ask you to create a map."

Gemini will create a skill file. Next time you need a map, just say "use the map skill" and it will follow the full process, including the design quality.

> What we built this morning

You now have a **mini data portal** with three pages:

- A **landing page** listing your datasets and linking to everything
- An **interactive map** with flood events, health facilities, and a risk filter
- A **visualisation page** with charts showing patterns in the data

All published on GitHub Pages. All based on data you found, cleaned, and categorised yourself.

> From tools to storytelling

You now have the technical skills. The challenge shifts.

Don't be too drunk on your work. The person consuming your content doesn't care that you have a lot of data or that your website is beautiful.

They care that you are **telling a clear, substantive story**.

This afternoon: apply everything to your own angle, tell the story, and get peer feedback on whether it lands.

> Q&A: OpenRefine from the terminal?

Some of you asked about running OpenRefine through Gemini CLI. OpenRefine itself is a GUI tool, but:

orcli is a command-line interface for OpenRefine. It lets you script OpenRefine operations (import, cluster, apply operations, export) from the terminal.

In practice, for the kind of cleaning we did this week, having Gemini write a Python script that does fingerprint clustering achieves the same result without installing OpenRefine. But if you work with OpenRefine regularly, orcli lets you automate recurring cleaning workflows.