ICT for Civic Data — Crash Course 2026

Finding and Getting Data

Self-Paced Review — Section C

Civic Literacy Initiative

civicliteraci.es

Why This Matters

> Build the evidence base

Find data sources, retrieve data, and verify it. These are the **Find**, **Get**, and initial **Verify** steps of the pipeline.

The goal: collect the data that will back your proposal's arguments. Every dataset you retrieve is a potential piece of evidence. The Find step discovers what exists. The Get step brings it into your workspace. The Verify step confirms it is what you think it is.

Before You Start

> Where data lives

Location	What you find	How to get it
Open data portal	Downloadable files (CSV, Excel, GeoJSON)	Browse and download
API	Structured data you can query	Send a request, get a response
Web page	Tables, lists, text	Scraping (manual or AI-assisted)
PDF or report	Tables, figures, text	Extraction (AI, Tabula, OCR)
Physical archive	Paper, photos	Scan + OCR or manual entry
"Nowhere"	Data that doesn't exist yet	FOI requests, crowdsourcing, field collection

"Nowhere" has strategies: Freedom of Information requests (e.g., UK knife crime data from 43 police forces), media monitoring and scraping (e.g., Turkish worker deaths), crowdsourcing (e.g., La Nacion Vozdata), collaborative research networks (e.g., Land Conflict Watch India).

> Common data formats

Format	What it is	Good for
CSV	Text with commas separating values	Tabular data. Opens in any spreadsheet.
GeoJSON	JSON with coordinates	Map data. Goes straight onto Leaflet.
Shapefile	Legacy GIS format (3-6 files)	Geographic boundaries. Needs conversion for web.
JSON	Nested, structured data	API responses. Needs flattening for tables.
Excel (.xlsx)	Spreadsheet with sheets and formulas	Common in government data. May need cleaning.

CSVs can use different separators: commas (standard), semicolons (common in Europe where commas are decimal separators), or tabs (.tsv). Google Sheets asks which separator to use during import.

> What is an API?

An API is a way to ask a database for specific data. Instead of downloading a huge file, you send a **request** and get back only what you asked for.

Example: the REST Countries API

```
https://restcountries.com/v3.1/name/kenya
```

Returns: population, area, region, capital, coordinates, borders, for Kenya only.

APIs are useful because your AI tool can **write the request for you** and **process the response**. Without knowing an API exists, you might instruct AI to scrape a website page by page, when the API could give you exactly the data you need.

> Chatbot for thinking, CLI for building

Web chatbot

- Free, no token limits
- Good for **questions and discovery**
- Cannot touch your files
- Each session is isolated

Use for: "What data sources exist for flood events?"

CLI agent (Gemini, Claude Code)

- Token limits (reset periodically)
- Works **on your files** directly
- Writes and runs scripts
- Sessions persist (`gemini --resume`)

Use for: "Write a script to download this data and build a map."

Do your thinking in the chatbot. Do your building in the CLI. This saves tokens and keeps the CLI context focused.

> Precise prompts produce better results

Think of AI as an **assistant**:

- Two **beginner assistants** asked a vague question will do their best, but their answers will be imprecise, sometimes irrelevant, and very different from each other
- Two **expert assistants** asked a precise question will give relevant, useful answers, and their answers will be much more similar

AI is **non-deterministic**: same prompt, different results every time. But **precision narrows the variation** dramatically.

Vague: "Find me data about disasters."

Precise: "I need open datasets of historical flood events with geographic coordinates. What sources exist? For each, tell me what it covers, what format the data is in, and how to access it."

> Separate data from interface to protect privacy

When working with sensitive data:

1. **Separate data from interface** — build the interface first with anonymised sample data (first 5 rows, randomised coordinates). Connect to real data afterwards.
2. **AI only needs the shape** — data models need column names and types, not actual values. Provide a minimal sample.
3. **Self-contained HTML** processes data entirely in the browser. Nothing leaves your device, even when published.
4. **Don't put sensitive data in free tools** — paid versions may contractually commit not to train on your data. Free tools make no such guarantee.

How to Do It

> Separate Find, Get, and Verify

Do them as **distinct steps**, not all at once:

1. **Find** — ask the AI what sources exist. Save the list to `sources.md` in your repo. This is documentation.
2. **Get** — pick a source. Ask the AI to write a script that retrieves the data. Read the script. Run it.
3. **Verify** — put the data on a map or in a spreadsheet. Does it look right?

If Verify reveals a problem, go back to Find or Get. Do not try to patch bad data.

> Prompt: discovering data sources

« I need open datasets of historical flood events with geographic coordinates. What sources exist? For each, tell me what it covers, what format the data is in, and how to access it. I'm a beginner, please explain in simple terms. »

Objective Steps Output shape Reasoning

Use the **chatbot** for this, not the CLI. Review what comes back. Then decide which source to try first.

> Prompt: extracting data with a script

« Write a script that reads FloodArchive_clustered.csv and keeps only the rows where Country is [your country]. Save the result as a new CSV file. I'm a beginner, please explain what you're doing in simple terms. »

Objective Steps Output shape Reasoning

Read the script before running it. A script is a reproducible artifact. If someone asks how you got this data, you show them the script.

> Prompt: building a simple Leaflet map

« Create a simple Leaflet map that can be published on GitHub Pages. Display the flood events from [your CSV] as circles. Add popups with date, cause, and severity. Center the map on [your country]. Publish it on GitHub Pages from the main branch. I'm a beginner, please explain what you're doing in simple terms. »

Objective | Steps | Output shape | Reasoning

Always say **"simple Leaflet map"**. Without "simple," the agent will build something complex and waste tokens.

> Prompt: querying the Overpass API

« Query the Overpass API for all hospitals and clinics in [your country]. Save the result as a GeoJSON file. I'm a beginner, please explain what you're doing in simple terms. »

Objective Steps Output shape Reasoning

The Overpass API queries OpenStreetMap data. No registration, no API key. Queries can take 30+ seconds for large countries; that is normal. Rate limits are generous: under 10,000 queries/day is safe.

> Disaster data sources

Source	Data	Format	Access
GDACS	Real-time disaster alerts (floods, earthquakes, cyclones)	GeoJSON	Free API, no key
FloodArchive	Historical flood events 1985–2024	CSV, Shapefile	Download from RIMES CKAN
EM-DAT	All disasters 1900–present, casualties, damage	Excel/CSV	Free, requires registration
healthsites.io	Health facility locations worldwide	JSON	API, requires OSM account
OSM / Overpass	Any mapped infrastructure	GeoJSON	Free API, no key
WorldPop	Population density grids	GeoTIFF	Free download
Meta HRSL	ML-derived 30m population density	GeoTIFF	HDX download
HDX	Thousands of humanitarian datasets	Mixed	Free, browse by country
REST Countries	Country-level data (population, borders)	JSON	Free API, no key

Behind the Approach

> Tasks stay within pipeline steps

If you ask AI to "find and map data" in one prompt, it does three things at once:

- Finds a source (but which one? did it hallucinate?)
- Downloads the data (but the right data? the right format?)
- Renders a map (but are the coordinates correct?)

If the map shows wrong data, you cannot tell which step failed. **Separating steps = separating failure modes.**

This is why we always do Find, then Get, then Verify, each as its own prompt, with its own verifiable output.

> More automation means more verification

As your extraction power goes up, the burden on **Verify** increases. AI makes getting data frictionless, but it does not make verifying data easier.

The pipeline keeps you honest: every step has a checkpoint where you inspect the result before proceeding.

AI concentrates at Get and Clean. The heatmap from Lecture 2 shows where AI helps most. But Define, Verify, and Analyse remain human-dominated because they require judgment about meaning.

FAQ

> What do I do when the AI agent isn't working?

- **Agent thinking too long?** Press **Escape** (may need multiple presses), then ask it why it is taking so long
- **Agent builds something too complex?** Ask for a **"simple Leaflet map"** or **"simple HTML page"**
- **Map not updating?** Hard refresh: **Ctrl+Shift+R** (Windows) or **Cmd+Shift+R** (Mac). This clears the browser cache. It is safe.
- **Agent fails to download a file?** Download it yourself and drag-and-drop into the Codespace
- **Agent publishes to wrong branch?** Tell it "publish on the main branch, not gh-pages" or change Settings → Pages to use the gh-pages branch

> How do I manage Gemini CLI sessions?

- `gemini --resume` reopens your latest session
- Different folders = different conversation histories
- Use separate terminal tabs for separate tasks
- Your Codespace preserves files but restarts apps. Gemini CLI needs to be relaunched, but `--resume` recovers the session

> Git and GitHub Pages essentials

Concept	What it does
Clone	Copy repo to your machine (Codespace does this automatically)
Branch	Parallel version for experiments (like duplicating a spreadsheet)
Commit	Save a change with a message
Push	Send committed changes to GitHub
Pull	Get changes from GitHub to your machine

GitHub Pages: enable in Settings → Pages. Set source branch to `main`. Your site appears at `username.github.io/repo-name/`.