Raspando dados com o GitHub Actions e analisando com Datasette

(Scraping data with GitHub Actions and analyzing with Datasette)

Simon Willison (@simonw) - simonwillison.net

Coda.Br 2021, 12th November 2021

This workshop will be presented in English with live translation.

This document: https://bit.ly/dados-datasette

Git scraping

Git scraping is the technique of writing scrapers that write scraped data to a Git repository, such that they can track changes made to that data over time.

- Git scraping: track changes over time by scraping to a Git repository
- Git scraping, the five minute lightning talk

My favourite git scraper that I've built is for the trees in San Francisco:

https://github.com/simonw/sf-tree-history

Every day it checks to see if anyone has updated the city's official CSV file of trees - over 195,000 trees! Most weekdays it turns out someone HAS updated it:

https://data.sfgov.org/City-Infrastructure/Street-Tree-List/tkzw-k3nq

Since the commit history page is broken right now, here are some recent tracked changes:

- 96528d0 Thu Nov 11 11:32:01 2021 +0000 18 trees changed, 80 trees added

- 6680fe7 Sun Nov 7 11:29:47 2021 +0000 12 trees changed, 11 trees added
- 5cc6330 Thu Nov 4 11:30:20 2021 +0000 13 trees changed, 2 trees added
- 92814f5 Sun Oct 31 11:29:53 2021 +0000 2 trees changed, 6 trees added

There are plenty more examples of Git scrapers on GitHub - over 200 now: https://github.com/topics/git-scraping - this page is even more fun if you sort by most recent updates, since you can see the Git scrapers that most recently captured data (often within the last few minutes): https://github.com/topics/git-scraping?o=desc&s=updated

The LA Times Data Desk team operates an incredible selection of scrapers for California coronavirus data. It's worth spending some time exploring their repository to see what a really sophisticated implementation of this pattern looks like, and learn from their code:

https://github.com/datadesk/california-coronavirus-scrapers

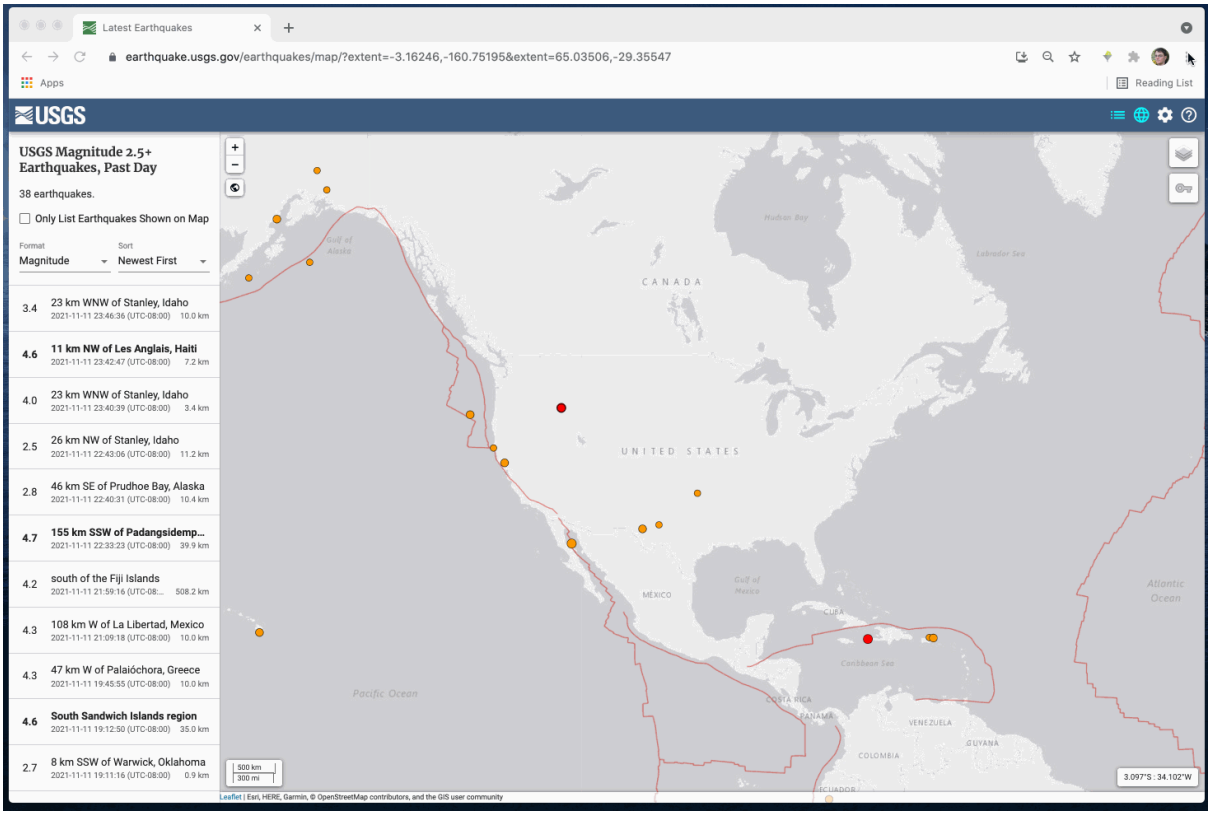
Exercise 1: Build a scraper using GitHub Actions

We're going to build a Git scraper, using just a GitHub account and our web browser.

We're going to scrape the data behind the USGS earthquake map at https://earthquake.usgs.gov/earthquakes/map/

I chose this because the data changes frequently, so we should see some changes even during the workshop.

We can use the browser DevTools to track down the data behind the map:



In this case it's a file at https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_day.geojson

It's also documented here: https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php

We're going to start scraping it every ten minutes, using GitHub Actions.

1. Make sure you have an account on https://github.com/ - if you don't have one yet sign up now

2. Create a new repository - https://github.com/new - I suggest calling it **usgs-scraper** but you can call it anything you like (here's mine: github.com/simonw/usgs-scraper)

3. Click "Add README" to finish creating the repository

4. Now we're going to add the scraper. Click "Add file -> Create new file" - then in "Name your file..." carefully type the following: `.github/workflows/scrape.yml`

Copy and paste the text from here into the new file: https://gist.github.com/simonw/f010059f240fddab1ffd2a60ec9eb49d (click "Raw" to make sure you copy the right text) - or you can try copying it from here:

```
name: Scrape latest data

on:
  push:
```

```
  workflow_dispatch:
  schedule:
    - cron:  '*/10 * * * *'

jobs:
  scrape:
    runs-on: ubuntu-latest
    steps:
    - name: Check out this repo
      uses: actions/checkout@v2
      with:
        fetch-depth: 0
    - name: Fetch latest data
      run: |-
        curl
"https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_day
.geojson" | jq > usgs.json
    - name: Commit and push if it changed
      run: |-
        git config user.name "Automated"
        git config user.email "actions@users.noreply.github.com"
        git add -A
        timestamp=$(date -u)
        git commit -m "Latest data: ${timestamp}" || exit 0
        git push
```

This defines what GitHub calls a "workflow". It will run every ten minutes - but it will also run any time you "push" code (edit files in your repository) or when you click a manual button - that's what "workflow_dispatch" means.

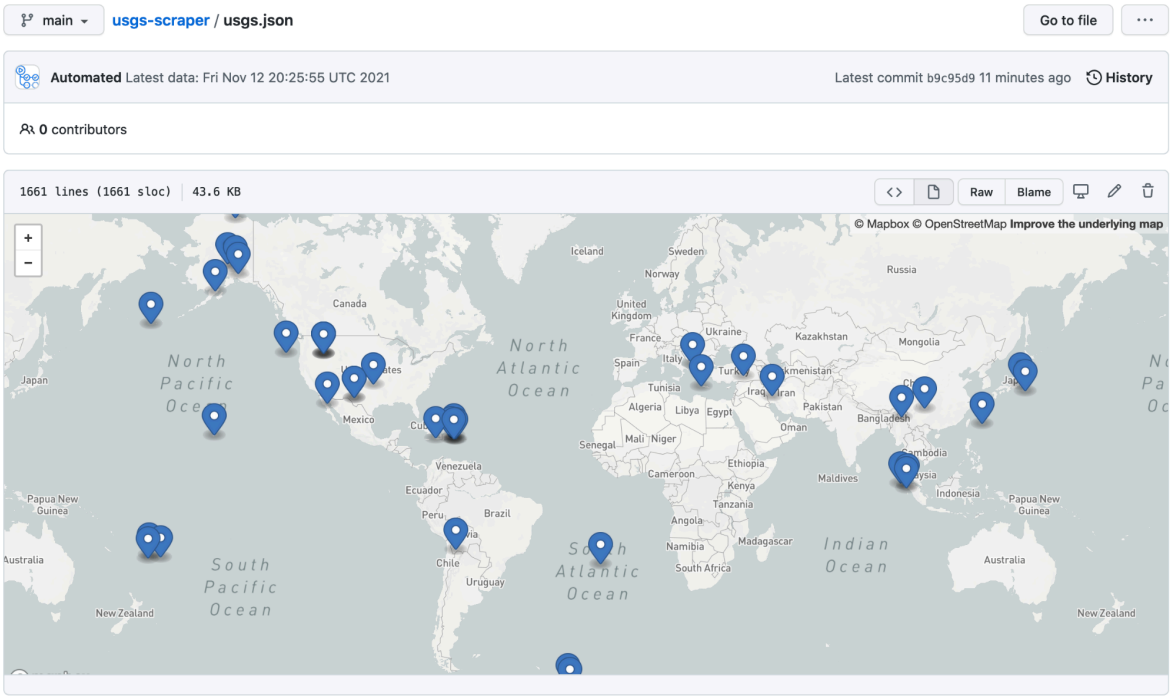
It will download that earthquake file using the `curl` tool, then pipe it through `jq` in order to pretty-print the JSON - this makes for a more useful display of file differences.

Then it writes the result to a file called `usgs.json` and commits any updates back to the repository.

Save the file... and you're done! You've written your first scraper. If you click on the "Actions" tab you'll be able to see it running.

Since the original format of this file is GeoJSON GitHub even knows how to render it on a map for you - here's mine:

https://github.com/simonw/usgs-scraper/blob/main/usgs.json



This is the simplest possible form of Git scraper - the same trick can work for HTML pages or CSV files too if you leave out the "`| jq`" bit (which only works with JSON files).

If you start this running now it will hopefully capture some fresh earthquakes before the end of the workshop!

Exercise 2: Try out Datasette against some scraped data

Now that we know how to scrape data, what can we do with the data that we collect?

I've spent the last four years building a tool called **Datasette** (it's birthday is tomorrow!). It's a tool for exploring, visualizing and publishing data. It's a great fit for exploring data collected through Git scraping.

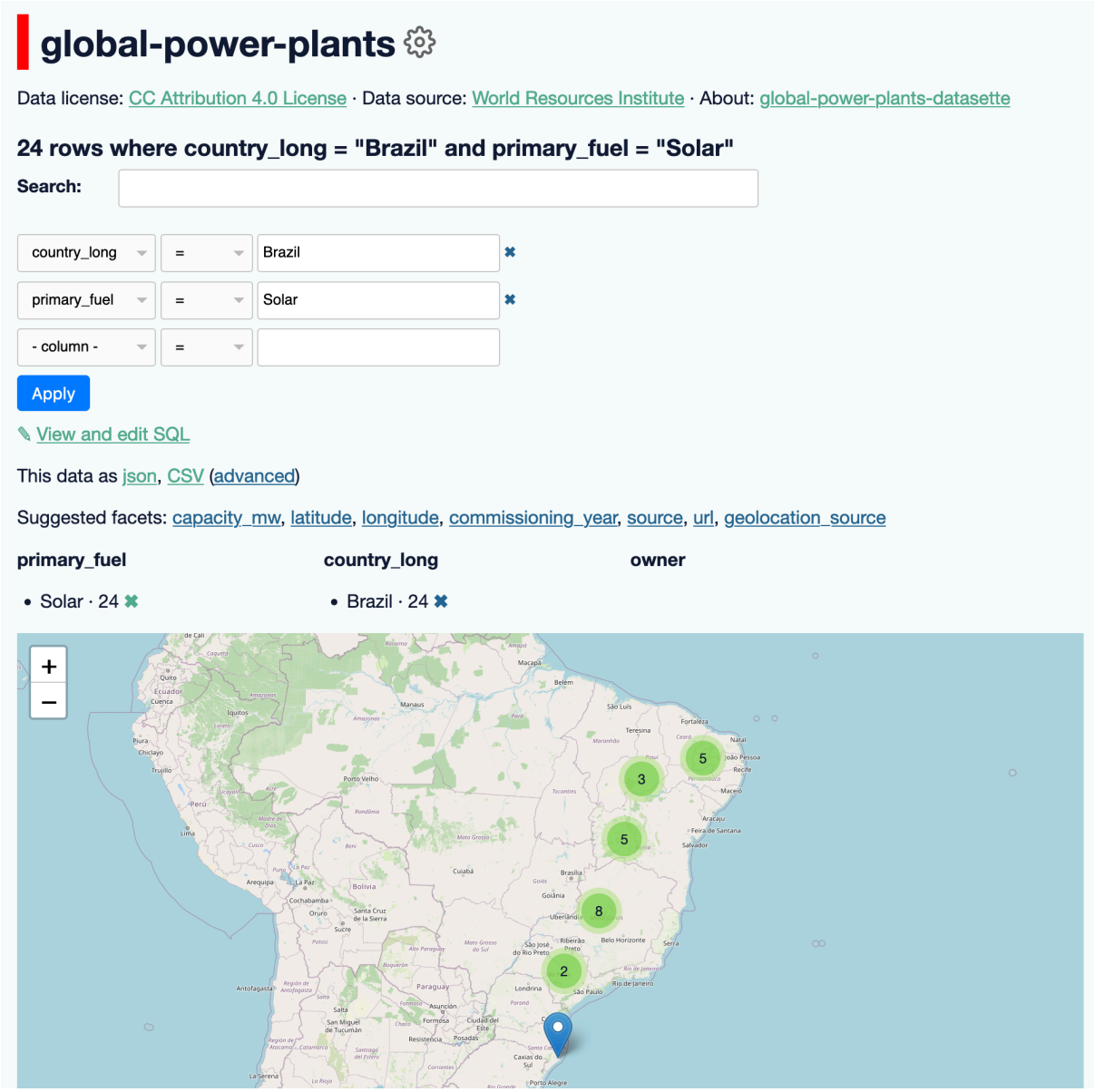
A demo: global-power-plants.datasettes.com/global-power-plants/global-power-plants

This is a Datasette instance containing 33,000 power plants from around the world, imported from a CSV file published by the World Resources Institute.

The links at the top of the page are called "facets" - they are a tool for slicing and filtering the data.

I can use them to see all of the power plants in Brazil: https://global-power-plants.datasettes.com/global-power-plants/global-power-plants?country_long=Brazil

Here's a map of all of the Solar plants in Brazil: https://global-power-plants.datasettes.com/global-power-plants/global-power-plants?country_long=Brazil&primary_fuel=Solar



The map is generated by a Datesette plugin called datasette-cluster-map - Datasette has a wide variety of plugins that add additional features like this.

I can export that data out as CSV or JSON.

If I click "View and edit SQL" I can compose my own SQL queries against the data too.

Example SQL query: https://global-power-plants.datasettes.com/global-power-plants?sql=select%0D%0A++name%2C%0D%0A++gppd_idnr%2C%0D%0A++capacity_mw%2C%0D%0A++latitude%2C%0D%0A++longitude%2C%0D%0A++primary_fuel%2C%0D%0A++commissioning_year%0D%0Afrom%0D%0A++%5Bglobal-power-plants%5D%0D%0Awhere%0D%0A++%22country_lon

g%22+%3D+%3Ap0%0D%0A++and+%22primary_fuel%22+%3D+%3Ap1%0D%0Aorder+by%0D%0A++rowid%0D%0Alimit%0D%0A++101&p0=Brazil&p1=Solar

I did another demo of my personal private Datasette, called Dogsheep. You can see more about that in Personal Data Warehouses: Reclaiming Your Data.

https://dogsheep.github.io/ - more about the Dogsheep personal analytics project.

Now let's look at a different example, created using Git scraping.

https://cdc-vaccination-history.datasette.io/ is a Datasette instance I created using data collected by my https://github.com/simonw/cdc-vaccination-history repository - which I built for this video: https://simonwillison.net/2021/Mar/5/git-scraping/

Because I've been scraping the page every day for 8 months now, I have a LOT of historical data.

I wrote some custom Python code to turn that into the database you see at https://cdc-vaccination-history.datasette.io/

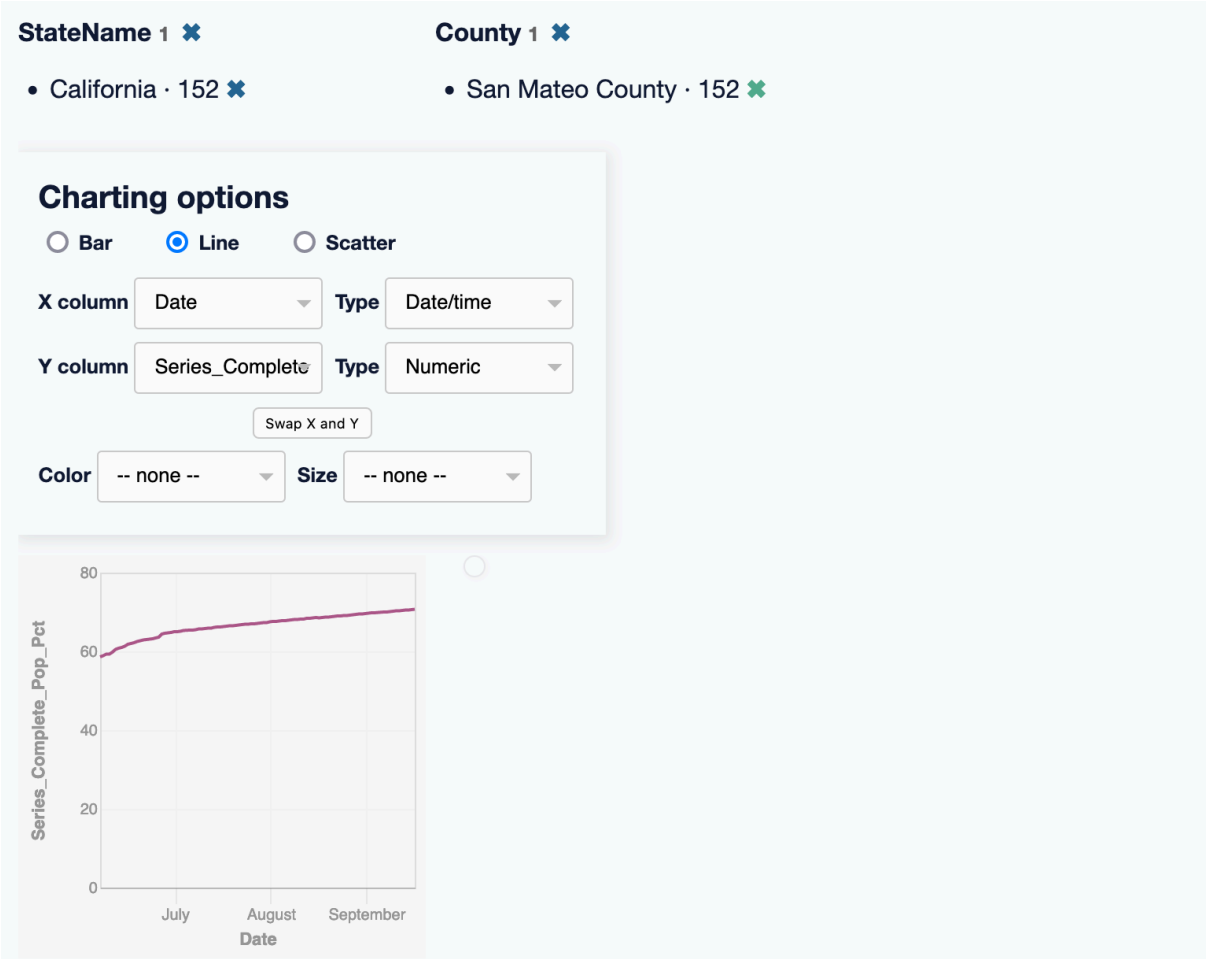
Script is https://github.com/simonw/cdc-vaccination-history/blob/main/build_database.py

Start at https://cdc-vaccination-history.datasette.io/cdc/daily_reports_counties - request StateName = California, then click the cog icon by the "County" menu and select "Facet by this".

Select "Alameda County" to see just the data collected for that county in California.

https://cdc-vaccination-history.datasette.io/cdc/daily_reports_counties?_facet=StateName&StateName=California&_facet=County&_facet_size=max&County=San+Mateo+County

Now click "Show charting options", select "line", set X column to "Date" of type "Date/time" and Y column to "Series_Complete_Pop_Pct" of type "Numeric". You'll see a line chart!



The charting feature is provided by the datasette-vega plugin.

Exercise 3: Use sqlite-utils and Datasette to build your own database

Database is built on top of SQLite: www.sqlite.org

SQLite is different from server-based database engines such as MySQL or PostgreSQL. In SQLite there is no server, and every database is a file on disk. This makes creating new SQLite databases really quick and easy - I often create dozens a day, and I have many hundreds scattered around my laptop now.

Next we're going to build our own Datasette instance.

For this, we're going to need a Python development environment.

If you have one of these on your own laptop and know how to use it (installing tools with `pip install` etc) you can do that.

If you don't have an environment on your laptop - or if you want to try something new - I recommend following along with me as I use **GitPod** as a cloud-based development environment for this next step.

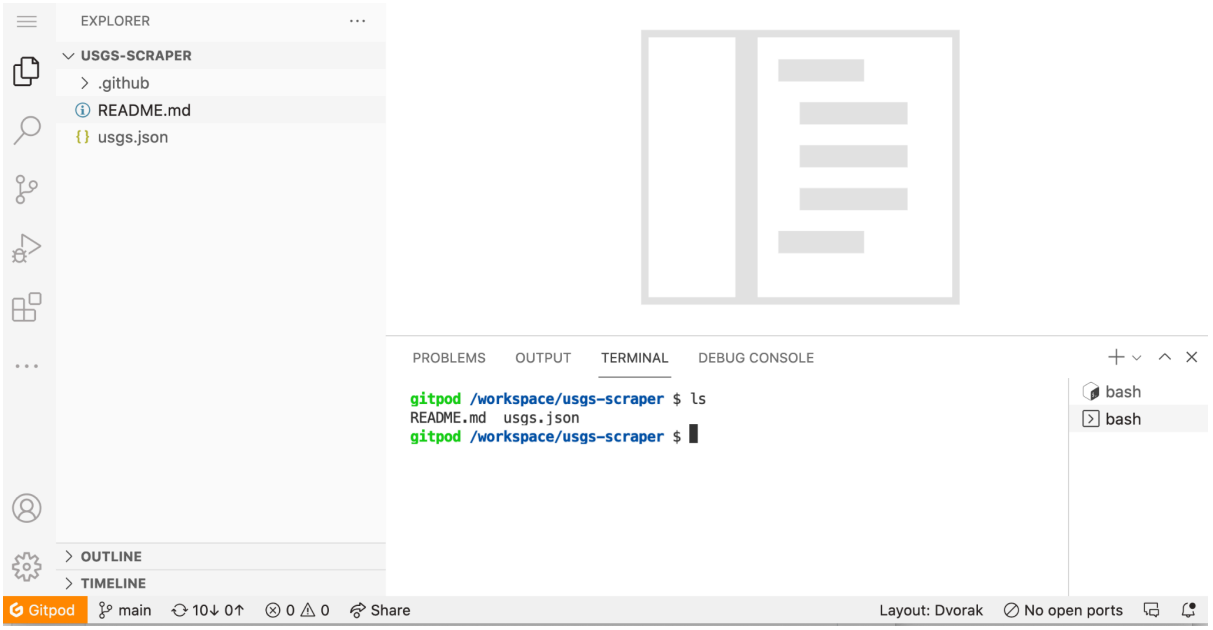
GitPod gives you a working Linux development environment with Python pre-installed and ready to go - for free, in your browser. It even includes a high quality in-browser text editor based on VS Code!

GitPod needs you to link it to a GitHub repository. We'll reuse the `usgs-scraper` repository you created earlier.

1. Visit https://gitpod.io and sign in with your GitHub account
2. Click "New Project" or visit https://gitpod.io/new and select that GitHub repository from earlier (any repository will do here actually)
3. Create and enter a Workspace for that repository.

Your Workspace gives you a full text editor running in your browser. It also runs a dedicated Linux environment for you, which you can run commands in.

Open the Terminal tab at the bottom of the page - this is where we will run commands to install and start running Datasette.



Run these commands:

```
pip install datasette
```

This installs Datasette. But we also need to install some plugins.

```
datasette install datasette-x-forwarded-host
```

This installs a plugin which is needed to ensure GitPod serves the correct links inside of Datasette.

```
datasette install datasette-cluster-map datasette-vega
```

This installs two more plugins - one for the map visualization, and one for the tool that lets you create charts.

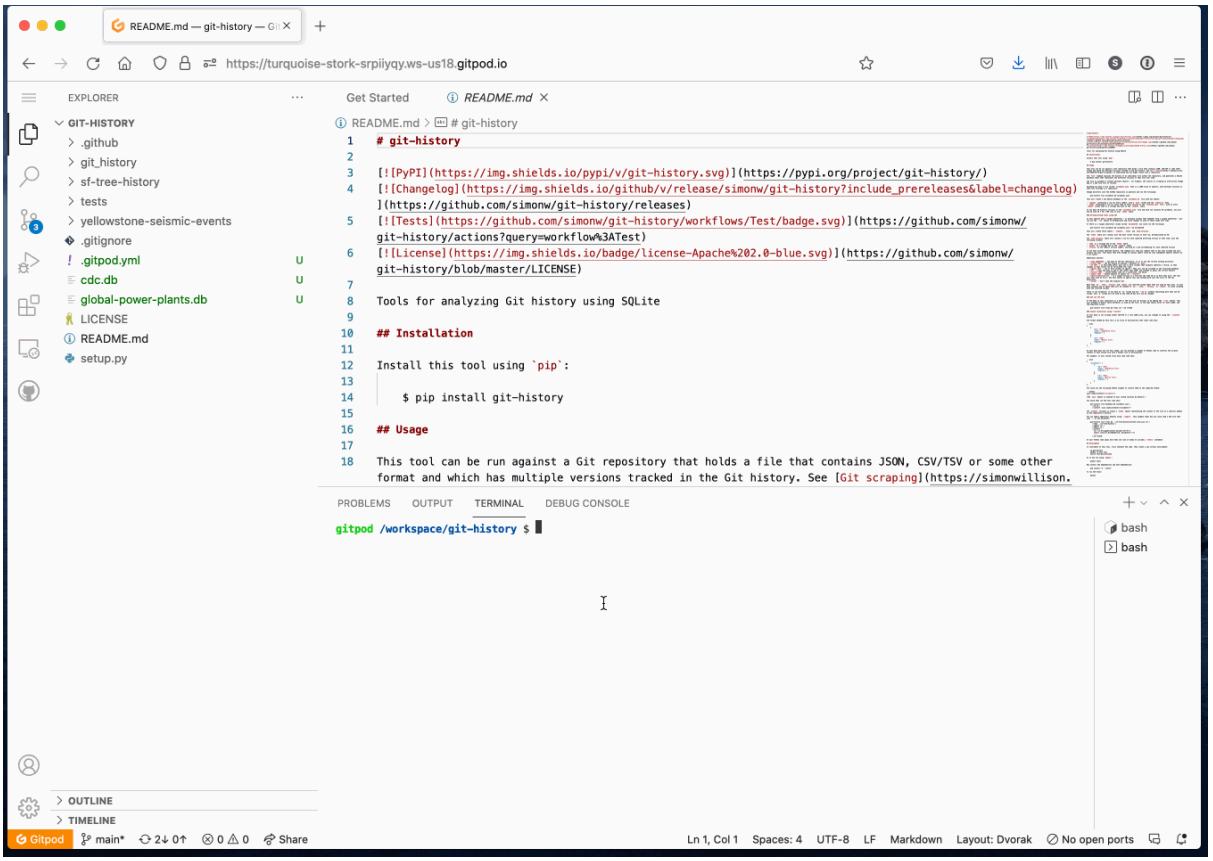
Let's grab a copy of the power plants database to try out in our own Datasette:

```
wget global-power-plants.datasettes.com/global-power-plants.db
```

Now we can start Datasette running like this:

```
datasette global-power-plants.db
```

Then click on the "Open Browser" button:



Next we're going to create our own SQLite database file. We'll use the database of trees in San Francisco that I've been creating with a Git scraper for several years now.

We can get that from https://github.com/simonw/sf-tree-history

The full link we want to the raw CSV file is:

```
https://github.com/simonw/sf-tree-history/blob/main/Str;eet_Tree_List.csv?raw=true
```

So in GitPod hit **Ctrl+C** to stop Datasette, then run the following:

```
wget "https://github.com/simonw/sf-tree-history/blob/main/Street_Tree_List.csv?raw=true"
```

```
mv Street_Tree_List.csv\?raw\=true Street_Tree_List.csv
```

That second command renames the file to something nicer - if you type `mv Stree` and then hit Tab it will complete the ugly file name for you, saving you from having to type it.

Now that we have a CSV file, we can convert it into a SQLite database. Since it's quite a large file, we're going to use a trick which loads data VERY fast into SQLite:

```
sqlite3 trees.db <<EOS
.mode csv
.import Street_Tree_List.csv trees
EOS
```

Now we can run `datasette trees.db` to see an interactive map of 195,000 trees in San Francisco!

Other things like GitPod are GitHub Codespaces (only available to beta testers) and Replit https://replit.com/

Q&A

Does Datasette allow joins?

> Yes it does! You can write custom SQL queries to join together data from two different tables.
>
> Here's an example query against my https://covid-19.datasettes.com/ Datasette which joins the New York Times county covid data against county population numbers from the US Census, using the FIPS code for the counties:
>
> ```
> select
>   ny_times_us_counties.date,
>   ny_times_us_counties.county,
>   ny_times_us_counties.state,
>   ny_times_us_counties.fips,
>   ny_times_us_counties.cases,
>   ny_times_us_counties.deaths,
>   us_census_county_populations_2019.population,
>   1000000 * ny_times_us_counties.deaths /
> us_census_county_populations_2019.population as deaths_per_million,
>   1000000 * ny_times_us_counties.cases /
> us_census_county_populations_2019.population as cases_per_million
> from
>   ny_times_us_counties
>   join us_census_county_populations_2019
>   on ny_times_us_counties.fips = us_census_county_populations_2019.fips
> where
>   "date" = (
>     select
>     max(date)
>     from
>     ny_times_us_counties
>   )
> order by
> ```

```
  deaths_per_million desc;
```

You can try that query here.

How about if you have a CSV file that uses semicolons instead of commas?

I don't (yet) know how to use the SQLite .import command for those, but my sqlite-utils tool has a `--sniff` option that tries to detect the delimiter:

```
sqlite-utils insert man.db mananciais mananciais.csv --csv --sniff
```

How do the nice commit messages in sf-tree-history work?

These use a tool I wrote called csv-diff to generate human-readable commit messages. You can see the workflow that runs that at https://github.com/simonw/sf-tree-history/blob/main/.github/workflows/update.yml

Exercise 4: potholes data, with git-history, on GitPod

We've seen how to create a database from the most recent version of a file with the San Francisco trees data.

But the point of Git scraping is to capture historical information. How can we turn that into something we can use?

Browsing the history on GitHub is not an efficient way to understand the data!

I've mainly been solving this on my own projects by writing custom Python code, but yesterday I released a new tool which can help solve this problem more generally for other projects.

It's called **git-history**: https://datasette.io/tools/git-history

We're going to try it out on some data collected using Git scraping.

When browsing through the repositories tagged with git-scraping on GitHub last night, I came across this one: https://github.com/patricktrainer/open-potholes

Patrick Trainer has been collecting data about potholes in the road in New Orleans, by scraping data from the New Orleans 311 service records on https://data.nola.gov/City-Administration/311-OPCD-Calls-2012-Present-/2jgv-pqrq/data

In GitPod, we'll start by installing the git-history tool:

```
pip install git-history
```

Next we'll clone a copy of his repository, with all of the historical pothole data:

```
git clone https://github.com/patricktrainer/open-potholes
cd open-potholes
```

In order to tell when an item is new as opposed to updated, we need to find a unique ID column we can use. Opening up `potholes.json` in the text editor shows us that `service_request` looks like a good option for this.

Now we can run this command to build ourselves a database that shows us all of those changes:

```
git-history file potholes.db potholes.json --id service_request
```

The `file` command creates a database that reflects the changes made to a single file in the repository. In this case that's the `potholes.json` file, and we want to create the database as `potholes.db`.

We can run `datasette potholes.db` to start exploring our database.

The `items` table contains just the most recent version of each pothole.

The `item_versions` table contains multiple rows for potholes that were updated multiple times.

Facet by item to see the potholes that had the most entries - then click on one and you'll see a chronological timeline of changes made to that record.

The Houston incidents demo

As a final demo, I did some work with this repository containing two years of data from the City of Houston active incidents page: https://cohweb.houstontx.gov/ActiveIncidents/Combined.aspx

The repository is at https://github.com/adolph/getIncidentsGit

It's **huge**. I ran this command over-night because it took hours to finish:

```
git-history file incidents.db /tmp/getIncidentsGit/incidents.json
--repo /tmp/getIncidentsGit \
     --id CallTimeOpened --id Address --id CrossStreet \
     --convert 'json.loads(content)["ActiveIncidentDataTable"]' \
     --ignore-duplicate-ids
```

Since this file doesn't have a unique ID column I decided to use the combination of the call time opened, the address and the cross street. This almost worked...

The `--convert` option is necessary because the stored data isn't quite in the right shape - it needs to be turned into a JSON array of objects.

`--ignore-duplicate-ids` is needed because there are a few pages in the data where the same call time / address / cross street is used for more than one record! It causes the script to take one of them at random rather than quitting with an error.

When I opened the database I found that I couldn't facet because it was too large and the faceting hit the time limit. So I restarted Datasette like this:

```
datasette incidents.db \
  --setting facet_time_limit_ms 10000 \
  --setting sql_time_limit_ms 10000
```

This bumped two of the time limits up to ten seconds, see Datasette settings.

Then I used sqlite-utils to make some modifications to the data. Full documentation on the features I used is here:

- sqlite-utils convert for converting values in columns using a Python snippet
- sqlite-utils extract for extracting columns into a separate table
- sqlite-utils transform for transforming a table to rename some columns

The full sequence of commands I ran was:

```
sqlite-utils convert incidents.db items XCoord YCoord \
  'float(value) / 1000000.0 if value else None'

sqlite-utils transform incidents.db items \
  --rename XCoord longitude \
  --rename YCoord latitude

sqlite-utils convert incidents.db items CallTimeOpened \
  'r.parsedatetime(value)'

sqlite-utils convert incidents.db items Units \
  'r.jsonsplit(value, delimiter=";")'

sqlite-utils extract incidents.db items IncidentType
```

More information

https://simonwillison.net/ will have more about `git-history` soon

https://datasette.io/ is the official website for datasette

GitHub Flat Data: https://next.github.com/projects/flat-data describes a GitHub research project along similar lines to Git scraping, which I found very reassuring as it means GitHub aren't going to decide that this is all a bad usage of GitHub and ban it!

Follow https://twitter.com/simonw for more of this kind of stuff - or subscribe to the Datasette monthly(ish) newsletter at https://datasette.substack.com/