Featured

Getting started

Hello, world

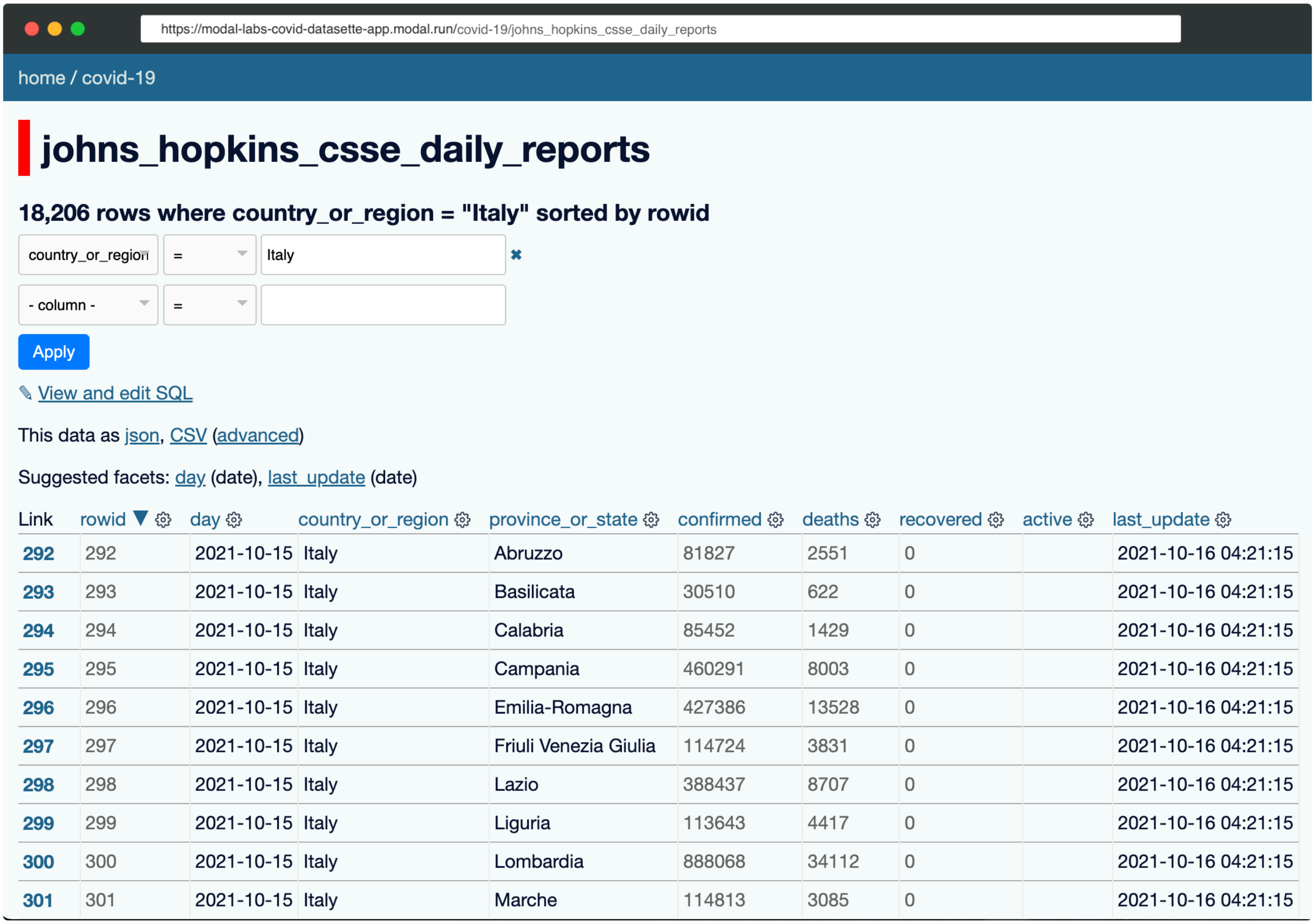
Simple web scraper

Large language models (LLMs)

Featured: Serverless TensorRT-LLM

Publish interactive datasets with Datasette

View on GitHub



https://modal-labs-covid-datasette-app.modal.run/covid-19/johns_hopkins_csse_daily_reports

home / covid-19

johns_hopkins_csse_daily_reports

18,206 rows where country_or_region = "Italy" sorted by rowid

✎ View and edit SQL

This data as json, CSV (advanced)

Suggested facets: day (date), last_update (date)

Link	rowid ▼	day	country_or_region	province_or_state	confirmed	deaths	recovered	active	last_update
292	292	2021-10-15	Italy	Abruzzo	81827	2551	0		2021-10-16 04:21:15
293	293	2021-10-15	Italy	Basilicata	30510	622	0		2021-10-16 04:21:15
294	294	2021-10-15	Italy	Calabria	85452	1429	0		2021-10-16 04:21:15
295	295	2021-10-15	Italy	Campania	460291	8003	0		2021-10-16 04:21:15
296	296	2021-10-15	Italy	Emilia-Romagna	427386	13528	0		2021-10-16 04:21:15
297	297	2021-10-15	Italy	Friuli Venezia Giulia	114724	3831	0		2021-10-16 04:21:15
298	298	2021-10-15	Italy	Lazio	388437	8707	0		2021-10-16 04:21:15
299	299	2021-10-15	Italy	Liguria	113643	4417	0		2021-10-16 04:21:15
300	300	2021-10-15	Italy	Lombardia	888068	34112	0		2021-10-16 04:21:15
301	301	2021-10-15	Italy	Marche	114813	3085	0		2021-10-16 04:21:15

This example shows how to serve a Datasette application on Modal. The published dataset is COVID-19 case data from Johns Hopkins University which is refreshed daily. Try it out for yourself at modal-labs-example-covid-datasette-app.modal.run/covid-19.

Some Modal features it uses:

- Volumes: a persisted volume lets us store and grow the published dataset over time.
- Scheduled functions: the underlying dataset is refreshed daily, so we schedule a function to run daily.
- Web endpoints: exposes the Datasette application for web browser interaction and API requests.

Basic setup

Let's get started writing code. For the Modal container image we need a few Python packages, including `GitPython` , which we'll use to download the dataset.

```
import asyncio
import pathlib
import shutil
import subprocess
from datetime import datetime
from urllib.request import urlretrieve

from modal import App, Image, Period, Volume, asgi_app

app = App(
    "example-covid-datasette"
)  # Note: prior to April 2024, "app" was called "stub"
datasette_image = (
    Image.debian_slim()
    .pip_install("datasette~=0.63.2", "sqlite-utils")
    .apt_install("unzip")
)
```

Persistent dataset storage

To separate database creation and maintenance from serving, we'll need the underlying database file to be stored persistently. To achieve this we use a `Volume` .

```
volume = Volume.from_name(
    "example-covid-datasette-cache-vol", create_if_missing=True
)
```

```
VOLUME_DIR = "/cache-vol"
REPORTS_DIR = pathlib.Path(VOLUME_DIR, "COVID-19")
DB_PATH = pathlib.Path(VOLUME_DIR, "covid-19.db")
```

Getting a dataset

Johns Hopkins has been publishing up-to-date COVID-19 pandemic data on GitHub since early February 2020, and as of late September 2022 daily reporting is still rolling in. Their dataset is what this example will use to show off Modal and Datasette's capabilities.

The full git repository size for the dataset is over 6GB, but we only need to shallow clone around 300MB.

```
@app.function(
    image=datasette_image,
    volumes={VOLUME_DIR: volume},
    retries=2,
)
def download_dataset(cache=True):
    if REPORTS_DIR.exists() and cache:
        print(f"Dataset already present and {cache=}. Skipping download.")
        return
    elif REPORTS_DIR.exists():
        print("Cleaning dataset before re-downloading...")
        shutil.rmtree(REPORTS_DIR)

    print("Downloading dataset...")
    urlretrieve(
        "https://github.com/CSSEGISandData/COVID-19/archive/refs/heads/master.zip",
        "/tmp/covid-19.zip",
    )

    print("Unpacking archive...")
    prefix = "COVID-19-master/csse_covid_19_data/csse_covid_19_daily_reports"
    subprocess.run(
        f"unzip /tmp/covid-19.zip {prefix}/* -d {REPORTS_DIR}", shell=True
    )
    subprocess.run(f"mv {REPORTS_DIR / prefix}/* {REPORTS_DIR}", shell=True)

    print("Committing the volume...")
    volume.commit()

    print("Finished downloading dataset.")
```

Data munging

This dataset is no swamp, but a bit of data cleaning is still in order. The following two functions read a handful of `.csv` files and clean the data, before inserting it into SQLite.

```python
def load_daily_reports():
    daily_reports = list(REPORTS_DIR.glob("*.csv"))
    if not daily_reports:
        raise RuntimeError(
            f"Could not find any daily reports in {REPORTS_DIR}."
        )
    for filepath in daily_reports:
        yield from load_report(filepath)


def load_report(filepath):
    import csv

    mm, dd, yyyy = filepath.stem.split("-")
    with filepath.open() as fp:
        for row in csv.DictReader(fp):
            province_or_state = (
                row.get("\ufeffProvince/State")
                or row.get("Province/State")
                or row.get("Province_State")
                or None
            )
            country_or_region = row.get("Country_Region") or row.get(
                "Country/Region"
            )
            yield {
                "day": f"{yyyy}-{mm}-{dd}",
                "country_or_region": (
                    country_or_region.strip() if country_or_region else None
                ),
                "province_or_state": (
                    province_or_state.strip() if province_or_state else None
                ),
                "confirmed": int(float(row["Confirmed"] or 0)),
                "deaths": int(float(row["Deaths"] or 0)),
                "recovered": int(float(row["Recovered"] or 0)),
                "active": int(row["Active"]) if row.get("Active") else None,
                "last_update": row.get("Last Update")
                or row.get("Last_Update")
                or None,
            }
```

Inserting into SQLite

With the CSV processing out of the way, we're ready to create an SQLite DB and feed data into it. Importantly, the `prep_db` function mounts the same volume used by `download_dataset()`, and rows are batch inserted with progress logged after each batch, as the full COVID-19 has millions of rows and does take some time to be fully inserted.

A more sophisticated implementation would only load new data instead of performing a full refresh, but we're keeping things simple for this example!

```
def chunks(it, size):
    import itertools

    return iter(lambda: tuple(itertools.islice(it, size)), ())


@app.function(
    image=datasette_image,
    volumes={VOLUME_DIR: volume},
    timeout=900,
)
def prep_db():
    import sqlite_utils

    volume.reload()
    print("Loading daily reports...")
    records = load_daily_reports()

    DB_PATH.parent.mkdir(parents=True, exist_ok=True)
    db = sqlite_utils.Database(DB_PATH)
    table = db["johns_hopkins_csse_daily_reports"]

    batch_size = 100_000
    for i, batch in enumerate(chunks(records, size=batch_size)):
        truncate = True if i == 0 else False
        table.insert_all(batch, batch_size=batch_size, truncate=truncate)
        print(f"Inserted {len(batch)} rows into DB.")

    table.create_index(["day"], if_not_exists=True)
    table.create_index(["province_or_state"], if_not_exists=True)
    table.create_index(["country_or_region"], if_not_exists=True)

    print("Syncing DB with volume.")
    volume.commit()
    db.close()
```

Keep it fresh

Johns Hopkins commits new data to the dataset repository every day, so we set up a scheduled function to automatically refresh the database every 24 hours.

```
@app.function(schedule=Period(hours=24), timeout=1000)
def refresh_db():
    print(f"Running scheduled refresh at {datetime.now()}")
    download_dataset.remote(cache=False)
    prep_db.remote()
    volume.commit()
    print("Volume changes committed.")
```

Web endpoint

Hooking up the SQLite database to a Modal webhook is as simple as it gets. The Modal `@asgi_app` decorator wraps a few lines of code: one `import` and a few lines to instantiate the `Datasette` instance and return its app server.

```
@app.function(
    image=datasette_image,
    volumes={VOLUME_DIR: volume},
    allow_concurrent_inputs=16,
)
@asgi_app()
def ui():
    from datasette.app import Datasette

    ds = Datasette(files=[DB_PATH], settings={"sql_time_limit_ms": 10000})
    asyncio.run(ds.invoke_startup())
    return ds.app()
```

Publishing to the web

Run this script using `modal run covid_datasette.py` and it will create the database.

You can then use `modal serve covid_datasette.py` to create a short-lived web URL that exists until you terminate the script.

When publishing the interactive Datasette app you'll want to create a persistent URL. Just run `modal deploy covid_datasette.py`.

```
@app.local_entrypoint()
def run():
    print("Downloading COVID-19 dataset...")
    download_dataset.remote()
    print("Prepping SQLite DB...")
    prep_db.remote()
```

You can explore the data at the deployed web endpoint.