developmentSEED

Lonboard: Fast, interactive geospatial data visualization

With a heavy dose of Apache Arrow

Kyle Barron, Development Seed

Contents

developmentSEED

My background

- Economics & math at UCLA
- Interested in data science via economic research
- Learned Stata first, then R
- Hacking in R was fun!
    - (It's an R meetup, have to burnish my R credentials…)

developmentSEED

README GPL-3.0 license

stata_kernel

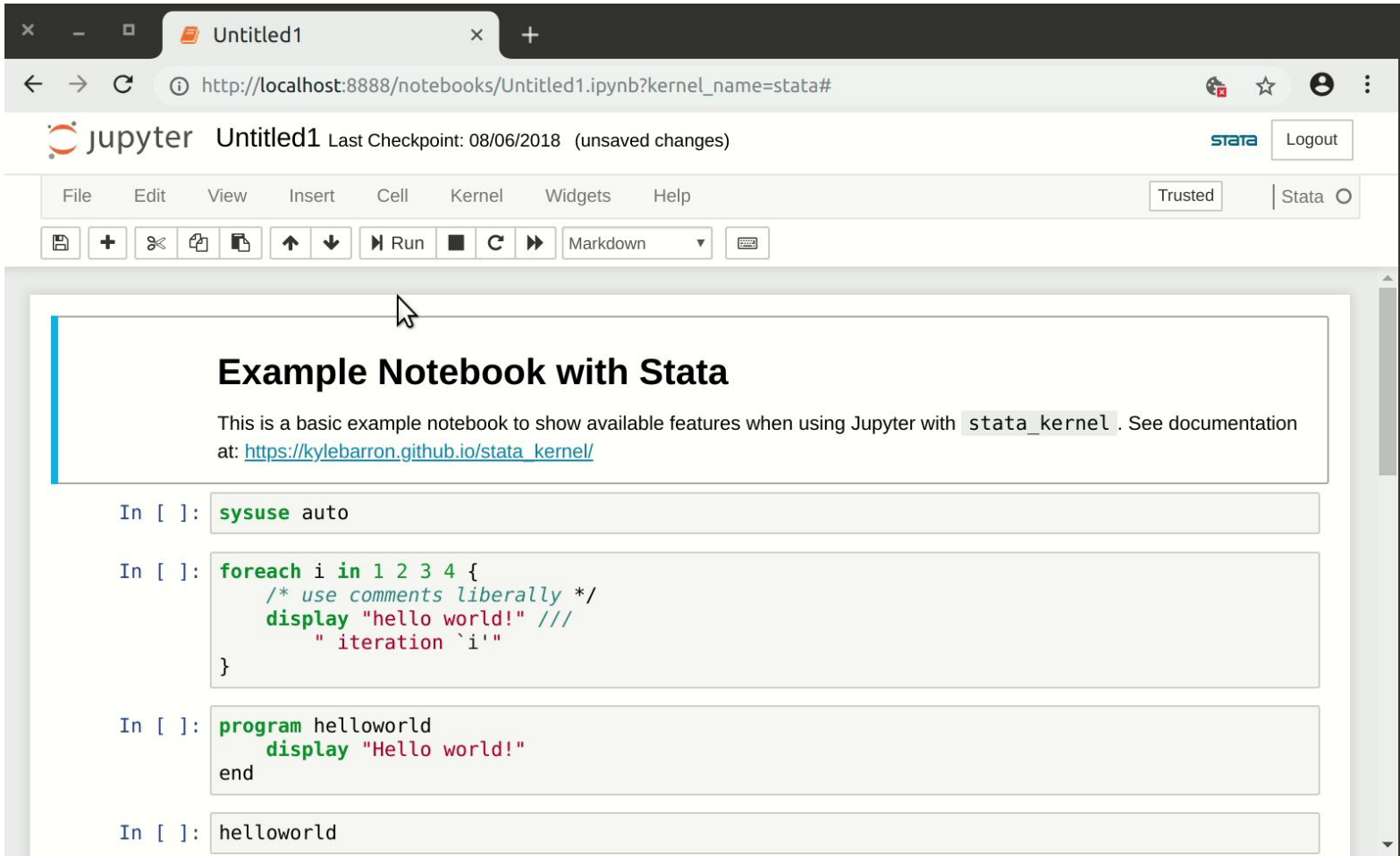
build error downloads 195k downloads/month 979

National Bureau of Economic Research

STATA

Economic Research

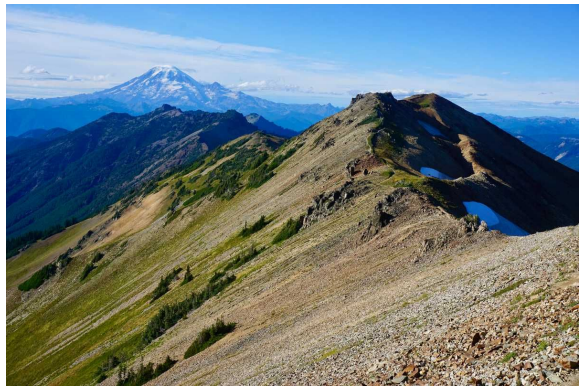
- Economics research assistant at NBER/MIT (2017-2019)
- Spent my time building tooling (in Python), slacked on the research side
- Excited about interactive programming in Jupyter & multi-language data interoperability (Stata + R + Python)
- PhD wasn't for me

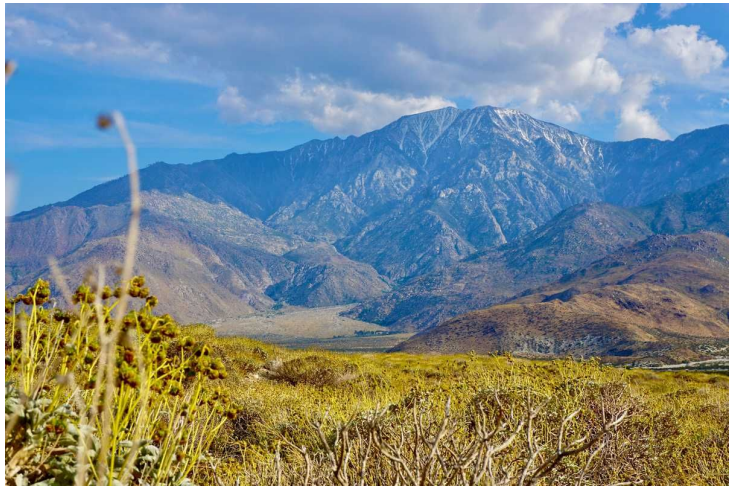


developmentSEED

Went for a hike!

- Hiked 2650-mile Pacific Crest Trail over 5 months
- Halfway through decided that I could build a better hiking app for the PCT



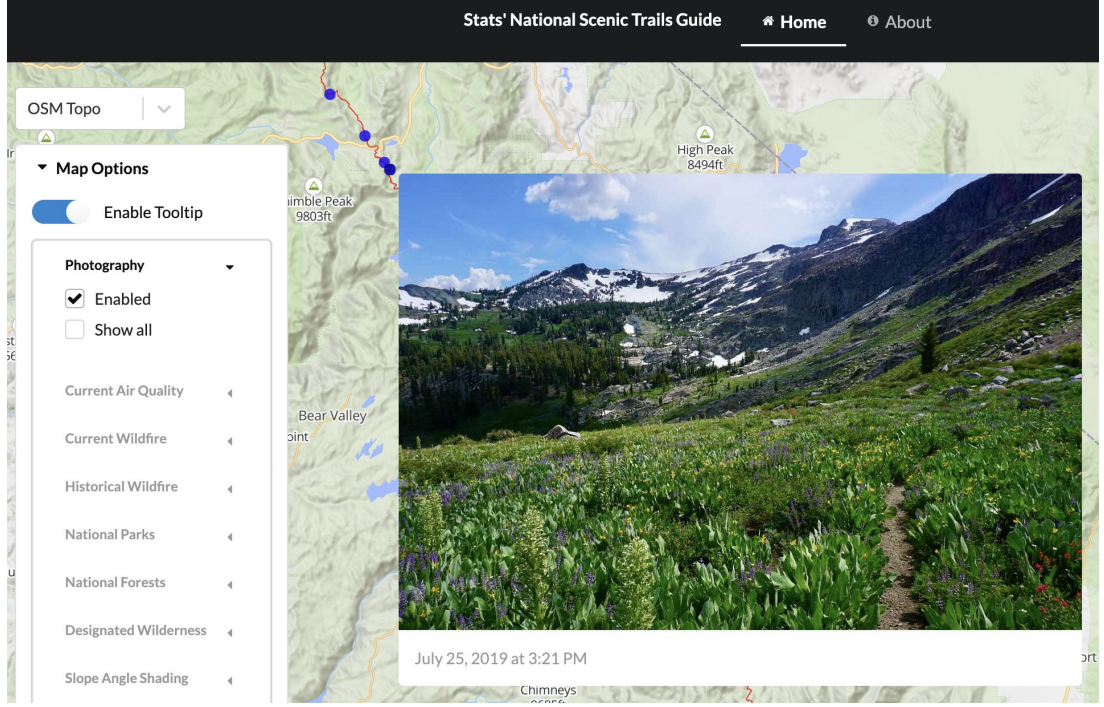


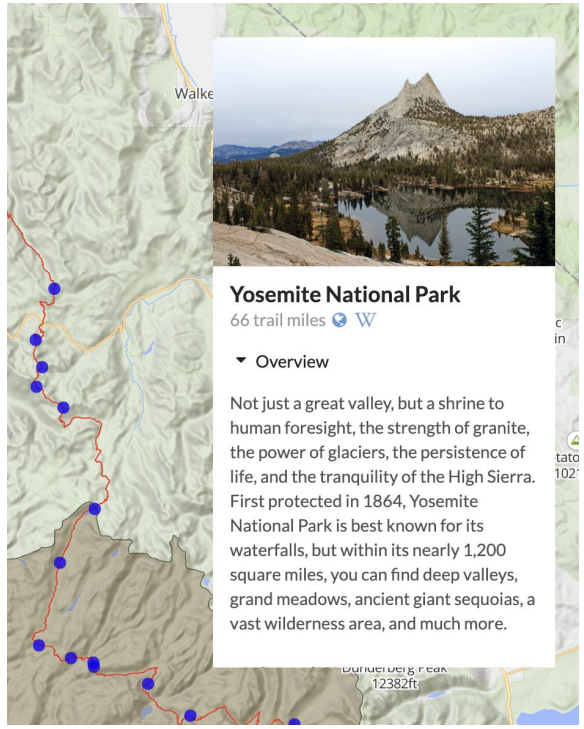
developmentSEED

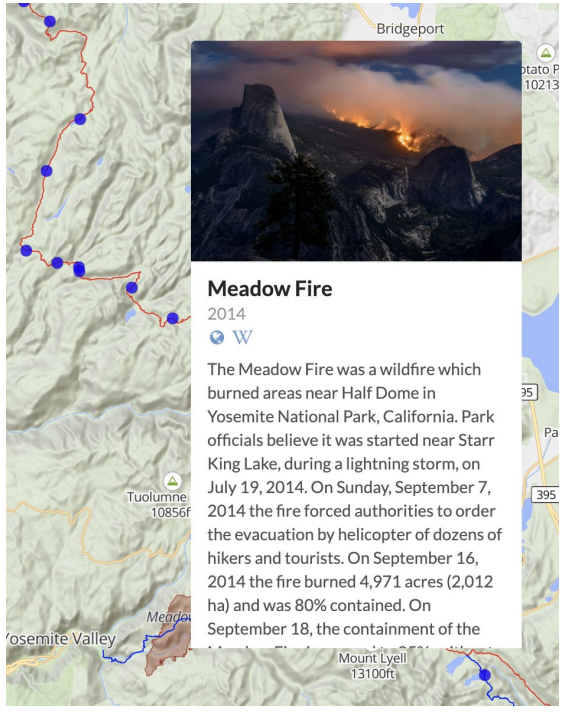
Back in the parents' nest

Built a map!

- Joined geospatial data sources with geopandas
- Learned about coordinate reference systems
- Created my own OSM vector tiles, hillshading, contour lines, raster tiles
- Learned JavaScript, React
- Used Mapbox GL, deck.gl
- Still online nst.guide and on github github.com/nst-guide





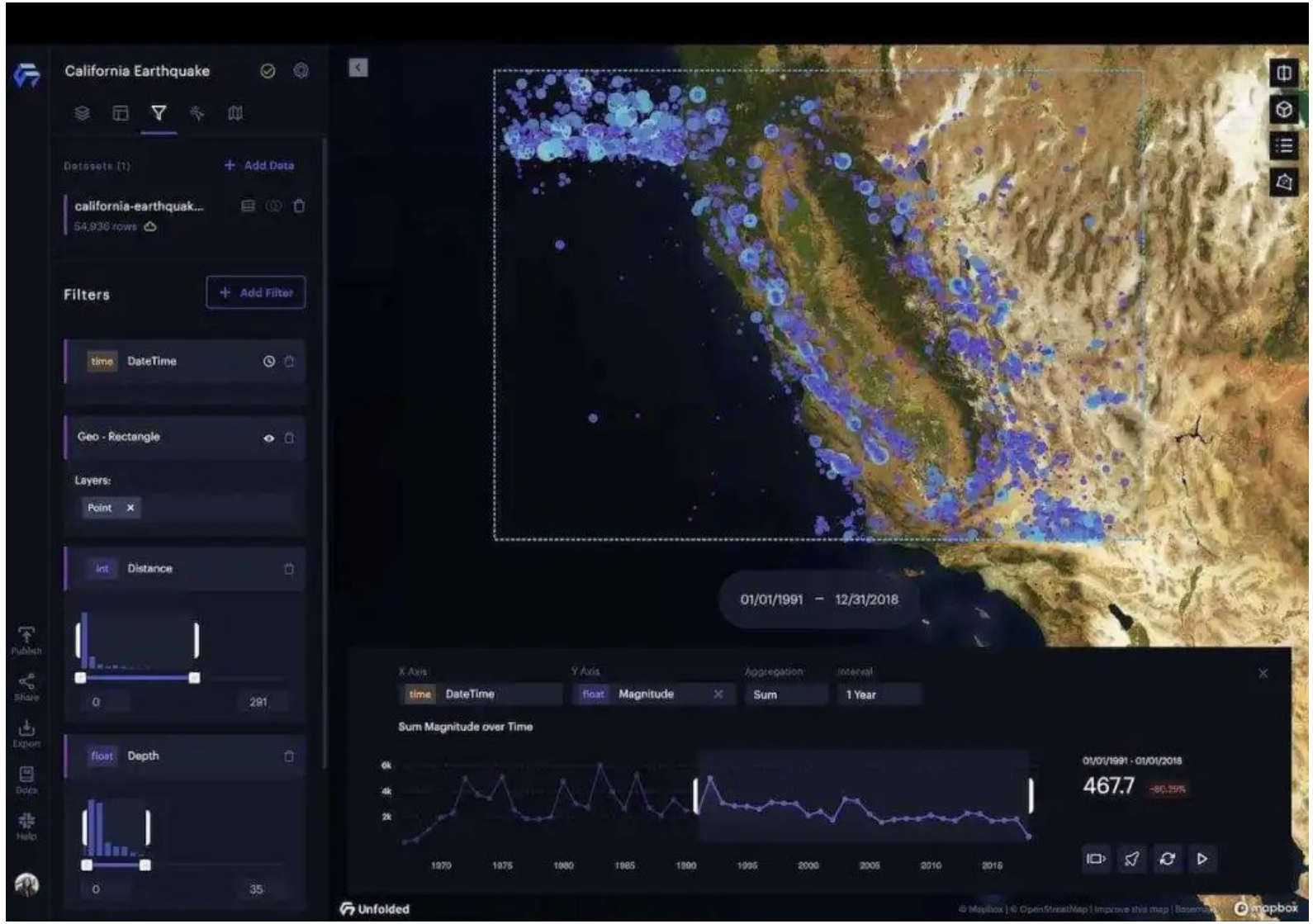


developmentSEED

Unfolded

Landed a job!

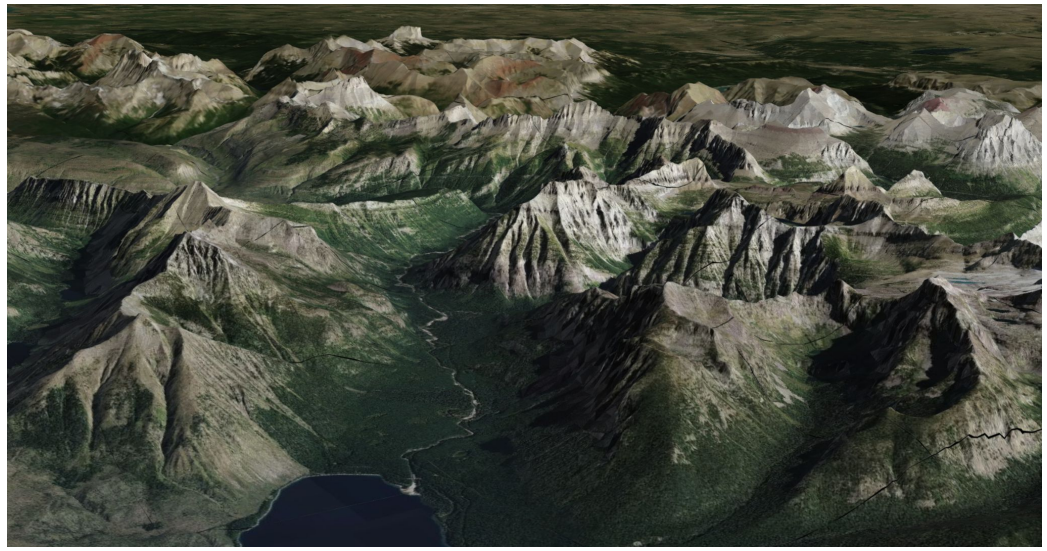
- Github networking worked!
- deck.gl maintainers had just left Uber to found a company
- Wrote low-level TypeScript parsers for various file formats (Shapefile, WKB)
- Wrote Python library to use Unfolded in Jupyter
- WebGL raster data visualization



developmentSEED

Open-source side projects

- Terrain mesh generation and visualization
- Modern data formats for geospatial data
- Rust for WebAssembly and Python
- Zero-copy data exchange with Arrow



README BSD-3-Clause license

GeoArrow Specification

This repository contains a specification for storing geospatial data in Apache Arrow and Arrow-compatible data structures and formats.

arrow-js-ffi Public

Zero-copy reading of Arrow data from WebAssembly

TypeScript 101 5

parquet-wasm Public

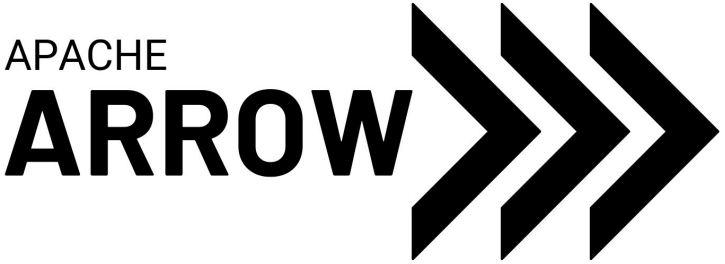
Rust-based WebAssembly bindings to read and write Apache Parquet data

Rust 485 19

developmentSEED

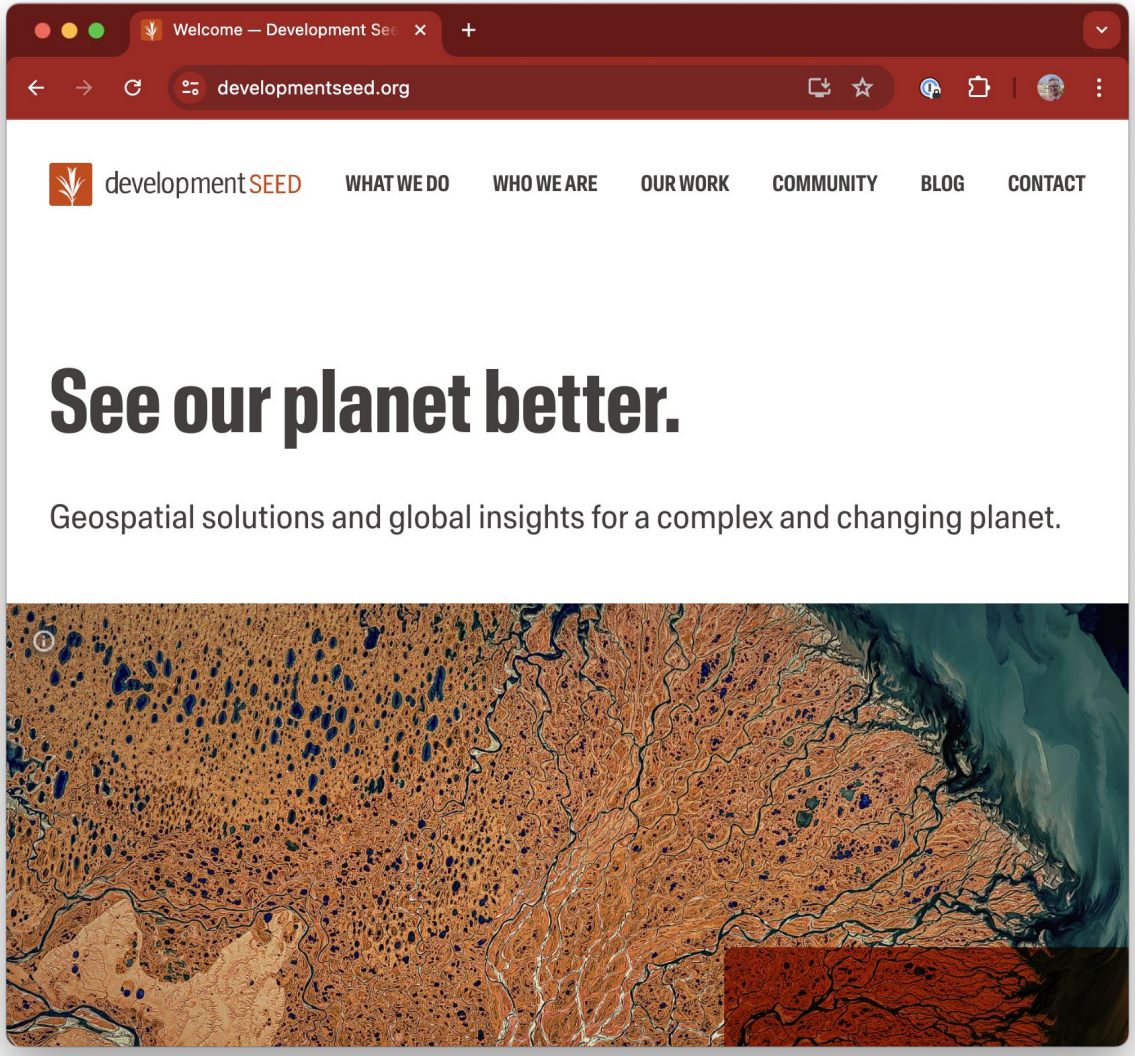
Developed my technical interests

- Geospatial data visualization (e.g. deck.gl)
- Interactive computing (e.g. Jupyter)
- Broad accessibility (e.g. browser)
- Data interoperability & multi-language workflows for data science (e.g. Parquet + Arrow)
- Low level binary data representation (Arrow)
- Fast, compiled code (Rust, Cython)



Fast forward a few years

- Joined Development Seed: open-source geospatial services
- Had some R&D time to explore…
  - Geospatial data visualization
  - Interactive computing
  - Broad accessibility
  - Data interoperability
  - Binary data representation
  - Fast, compiled code



developmentSEED

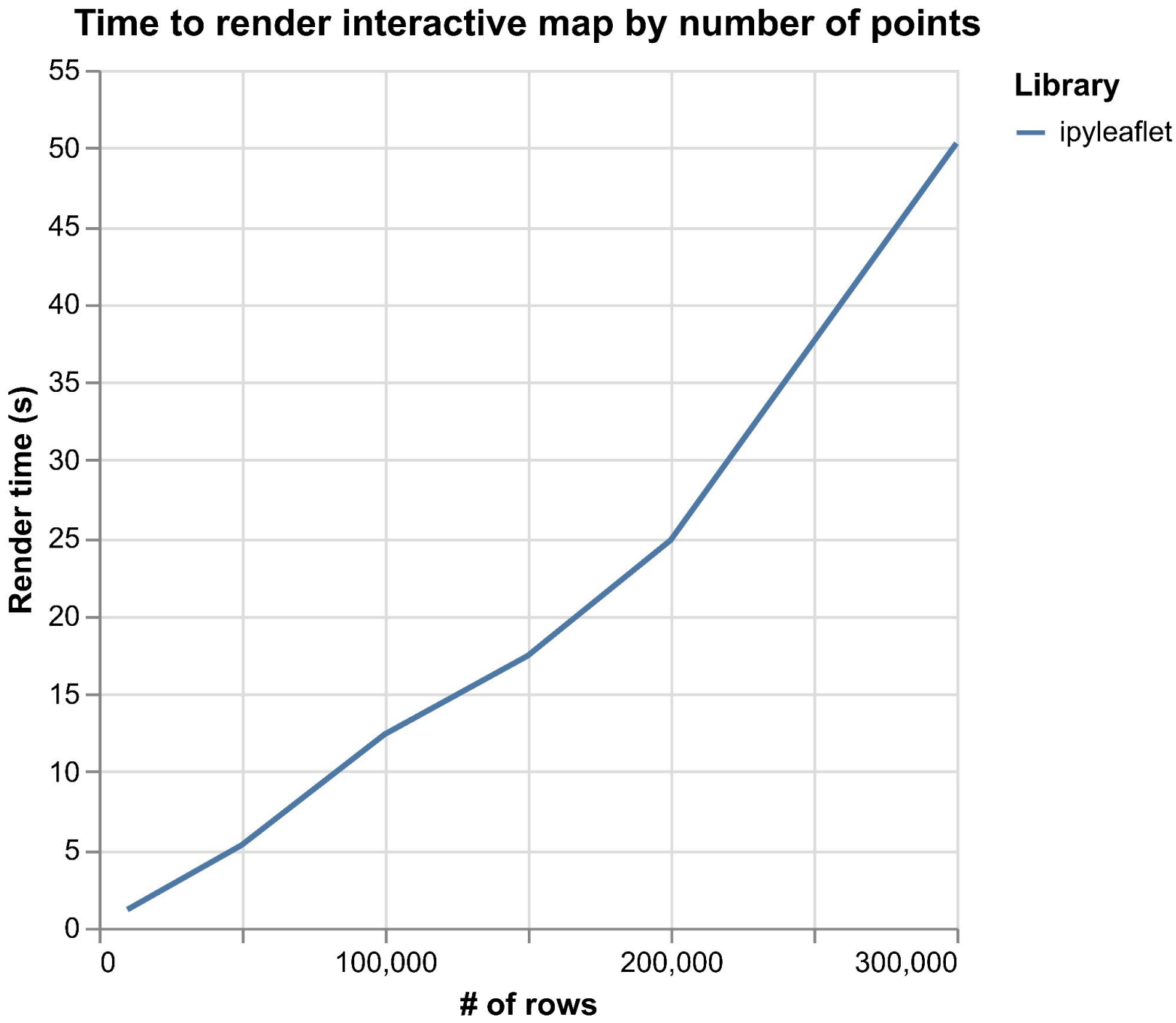
Simpler, faster rendering of large analytical vector datasets (in Python)

developmentSEED

…why not existing libraries?

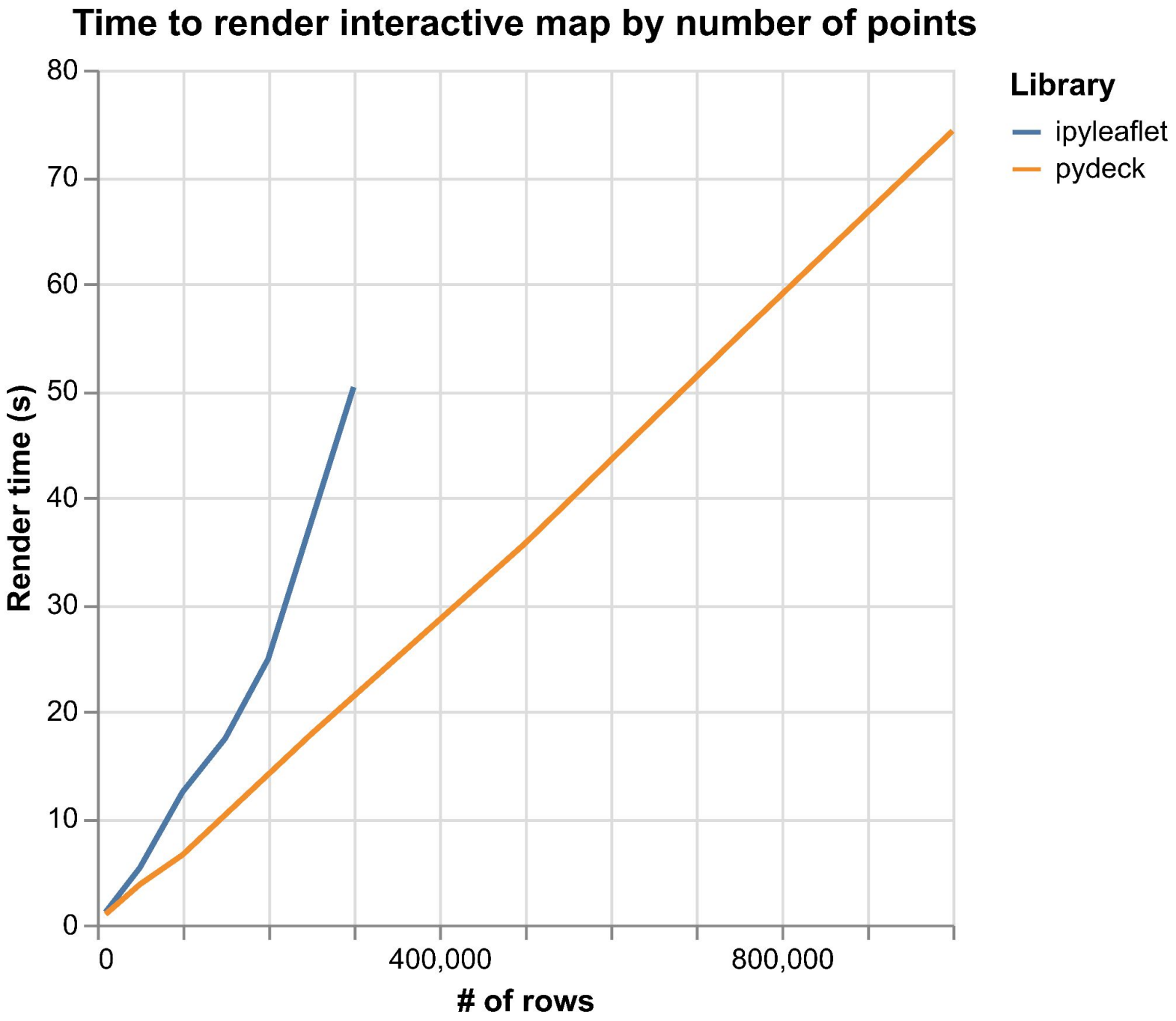
developmentSEED

ipyleaflet?



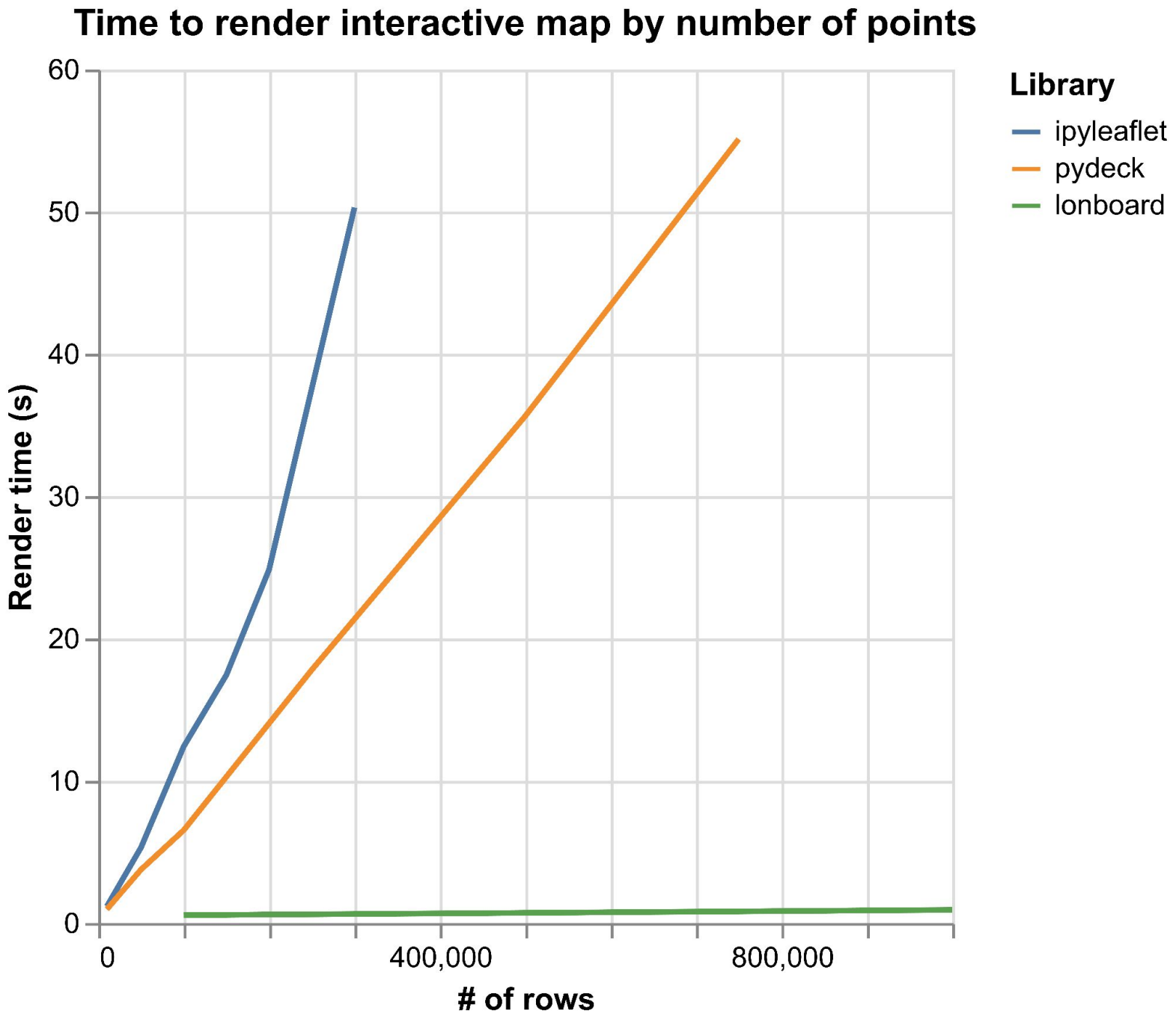
developmentSEED

pydeck?



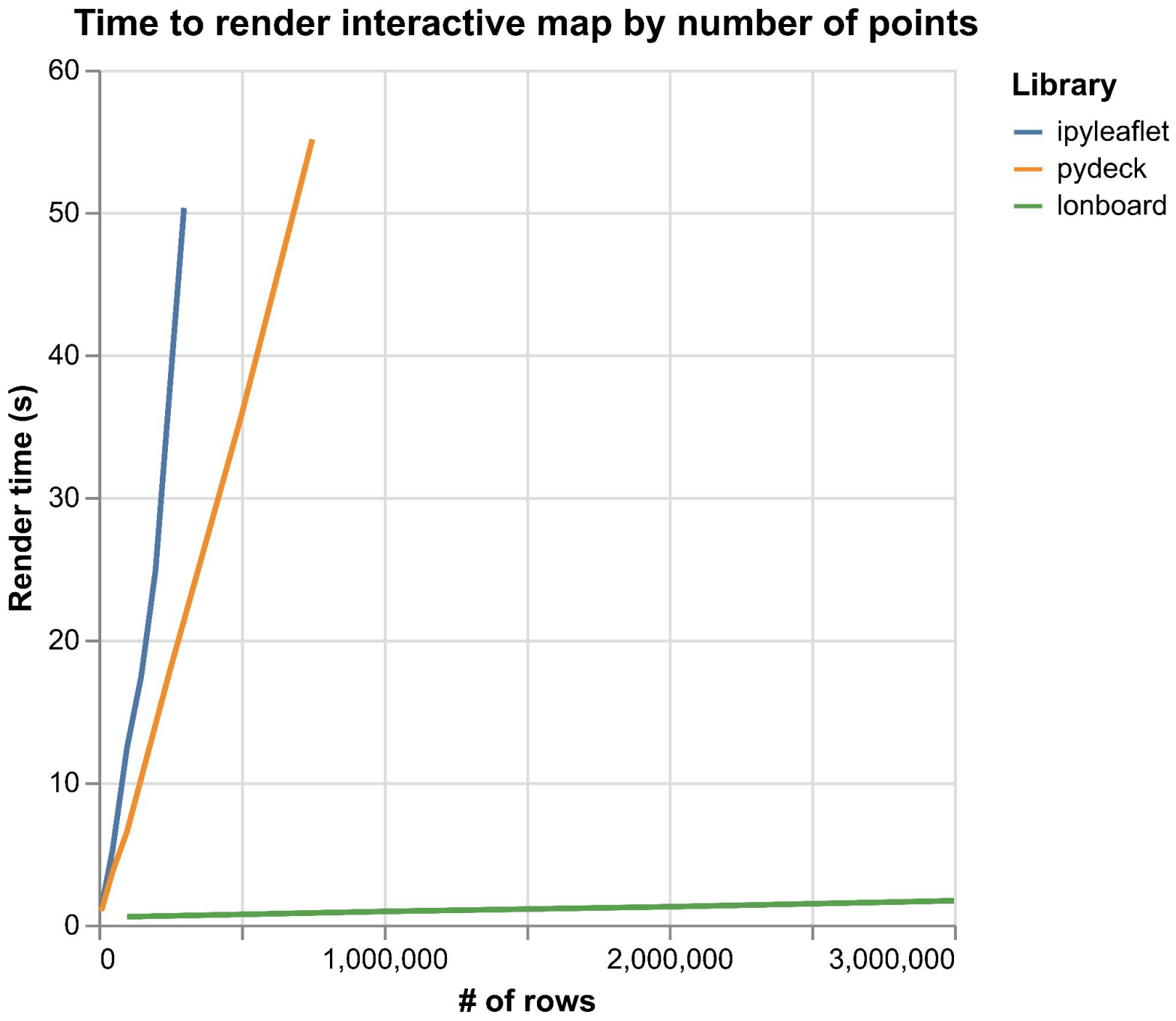
developmentSEED

lonboard!



developmentSEED

lonboard!

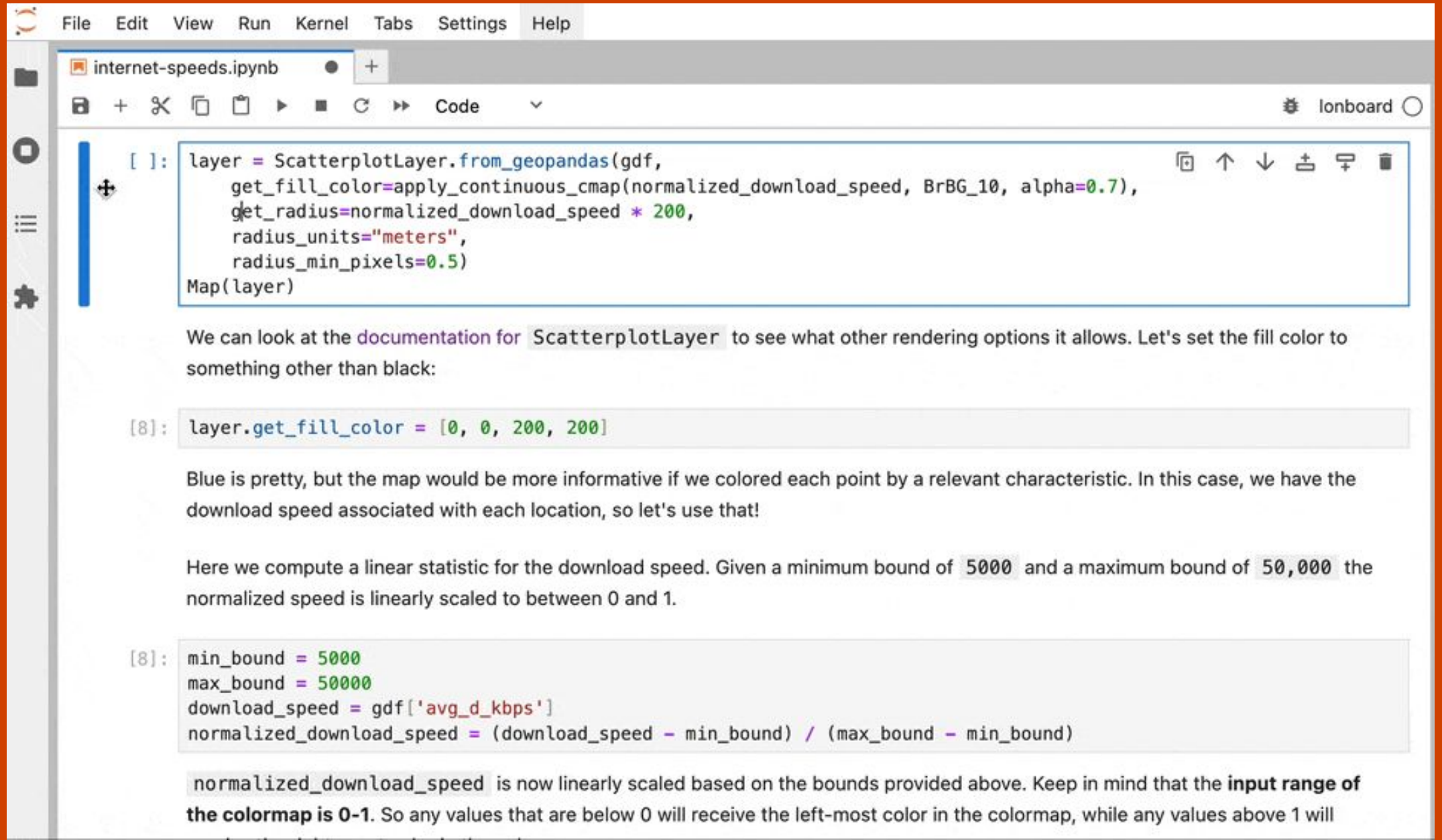


developmentSEED

Lonboard: 3 million points in 2.5 seconds

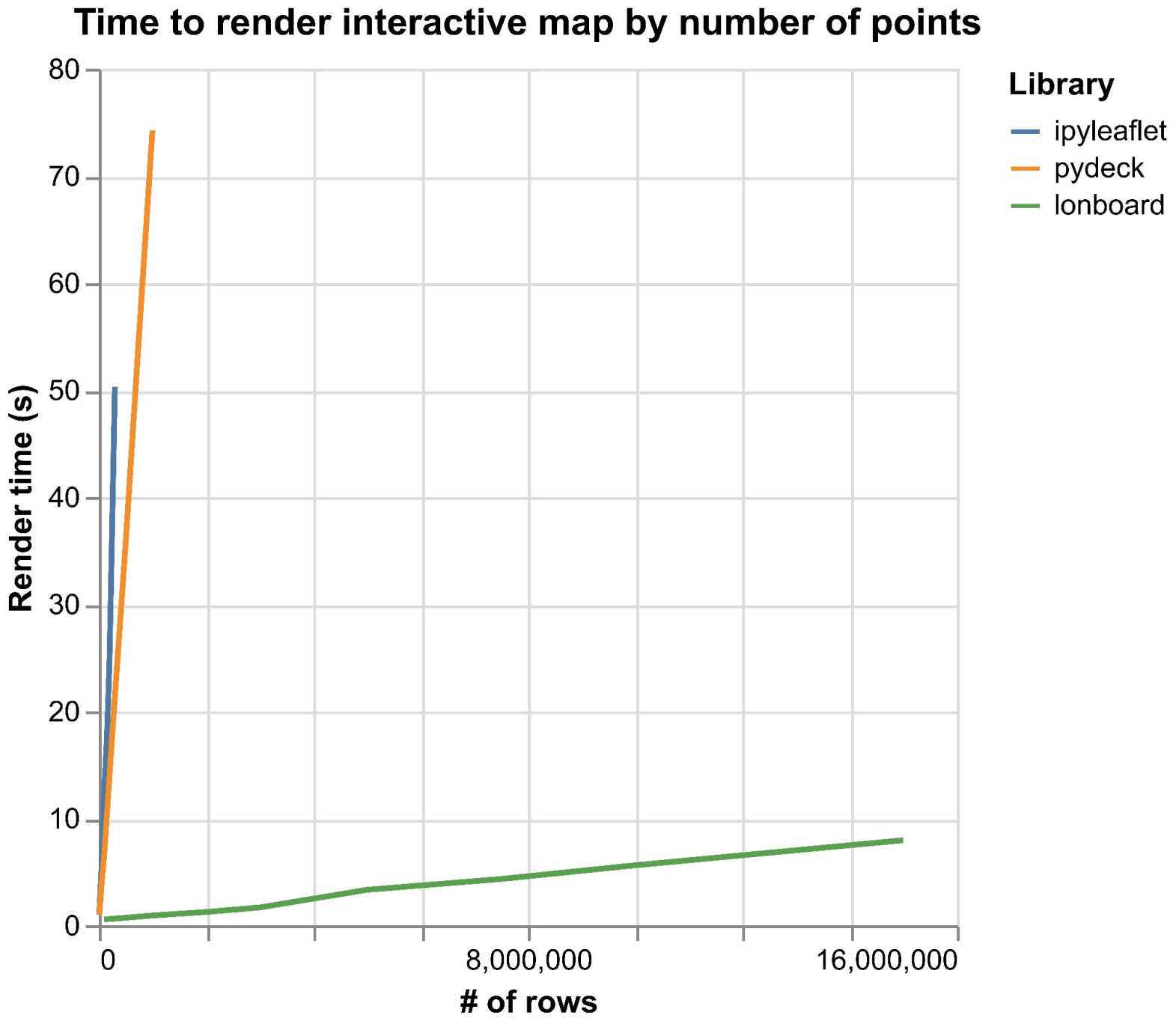
developmentSEED

Interactive Data Exploration



developmentSEED

lonboard!



developmentSEED

Laggy, but functional at 15M points

[41]:
```
layer = ScatterplotLayer.from_geopandas(
    large.iloc[:15_000_000],
    get_fill_color=apply_continuous_cmap(normalized_download_speed[:15_000_000], BrBG_10, alpha=0.7),
    get_radius=normalized_download_speed[:15_000_000] * 20,
    radius_units="meters",
    radius_min_pixels=0.5)
m = Map(layer)
m
```

[41]:

We can look at the documentation for `ScatterplotLayer` to see what other rendering options it allows. Let's set the fill color to something other than black:

[8]:
```
layer.get_fill_color = [0, 0, 200, 200]
```

Blue is pretty, but the map would be more informative if we colored each point by a relevant characteristic. In this case, we have the download speed associated with each location, so let's use that!

Here we compute a linear statistic for the download speed. Given a minimum bound of `5000` and a maximum bound of `50,000` the normalized speed is linearly scaled to between 0 and 1.

[38]:
```
min_bound = 5000
max_bound = 50000
download_speed = large['avg_d_kbps']
normalized_download_speed = (download_speed - min_bound) / (max_bound - min_bound)
```

`normalized_download_speed` is now linearly scaled based on the bounds provided above. Keep in mind that the **input range of the colormap is 0-1**. So any values that are below 0 will receive the left-most color in the colormap, while any values above 1 will receive the right-most color in the colormap.

[10]:
```
normalized_download_speed
```

developmentSEED

High-level API & integrations

Simple to use

Lonboard integrates with existing libraries

- GeoPandas `GeoDataFrame`, `GeoSeries`
- Shapely array, scalar
- DuckDB Spatial query
- GDAL (pyogrio)
- Ibis (in progress)
- Any GeoArrow Python object

developmentSEED

High-level API

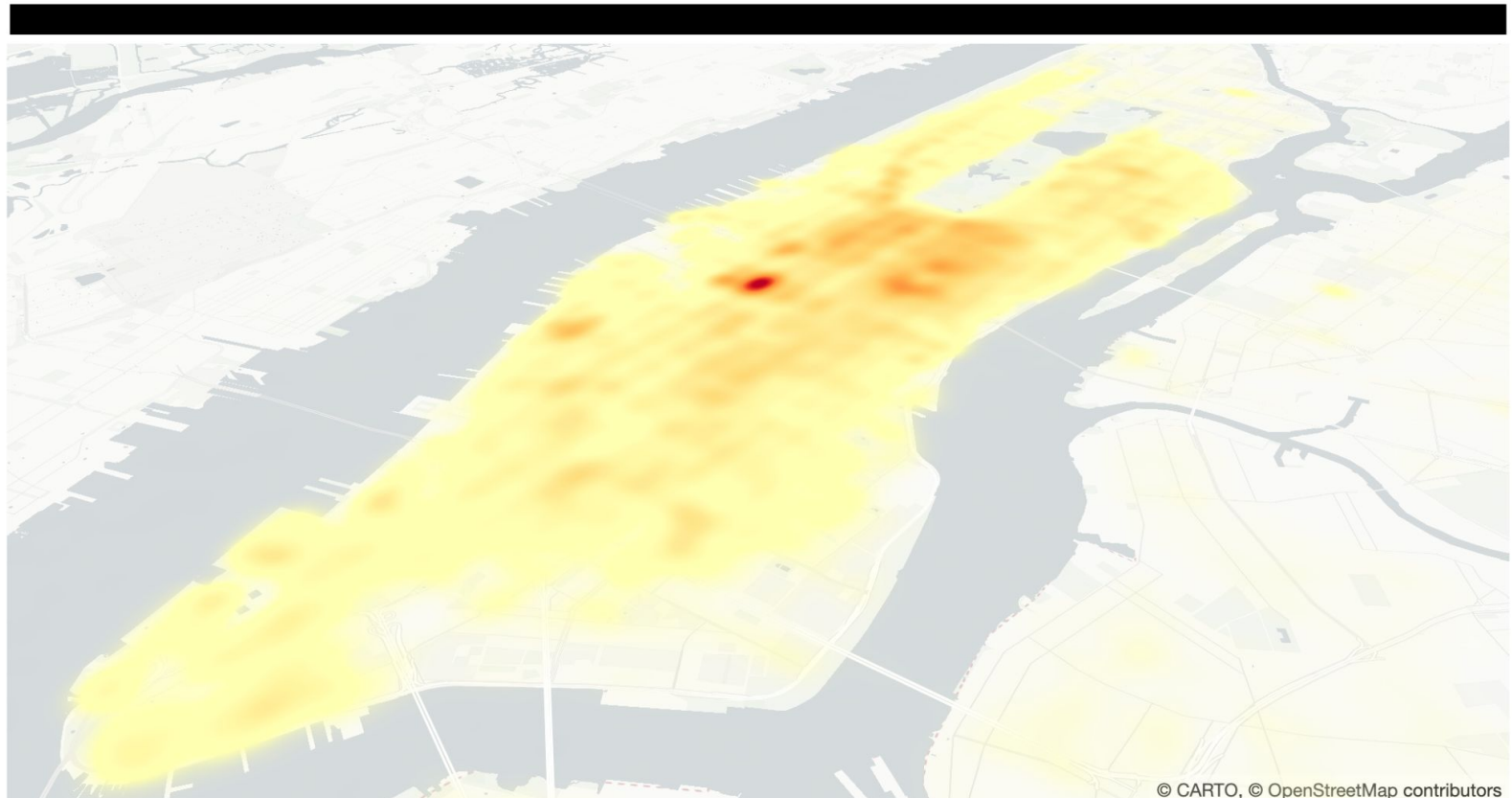
DuckDB Integration

- Pass query into `from_duckdb`

[5]:
```
sql = """
SELECT
    *,
    ST_Point(dropoff_longitude, dropoff_latitude) as geometry
FROM "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2010-03.parquet"
WHERE
    dropoff_longitude >= -74.6087 AND
    dropoff_latitude >= 40.2738 AND
    dropoff_longitude <= -73.4928 AND
    dropoff_latitude <= 41.1757 AND
    total_amount > 0
LIMIT 1_000_000;
"""
```

[6]:
```
layer = HeatmapLayer.from_duckdb(sql, con)
m = Map(layer)
m
```

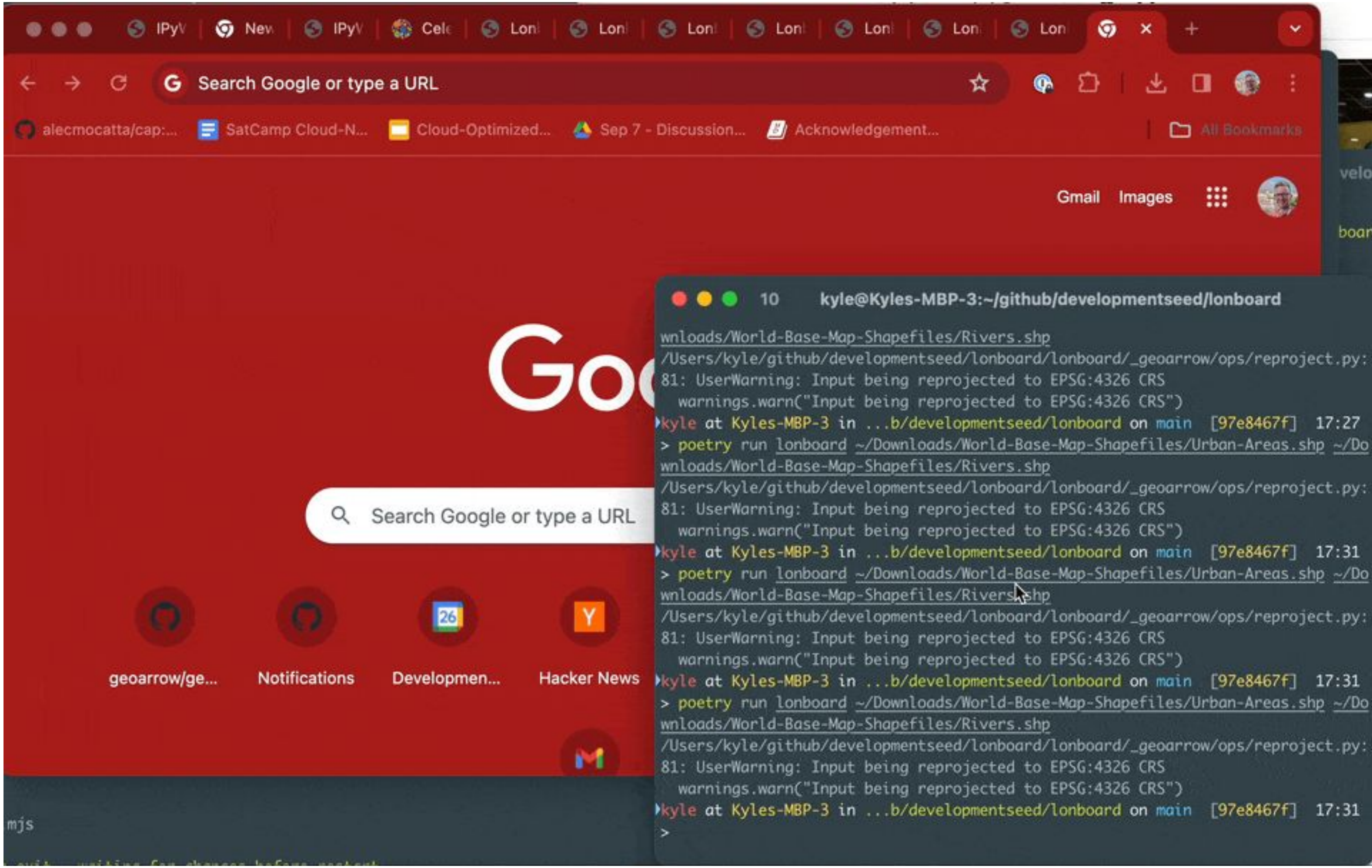
[6]:



developmentSEED

High-level API

CLI Included



developmentSEED

GPU Data Filtering

Launcher | openaq_explore.ipynb

Code

the map based on user input:

```
[16]: pm25_slider = FloatRangeSlider(
          value=[val_min, val_max],
          min=val_min,
          max=100,
          step=0.1,
          description="PM2.5: "
      )
      time_slider = FloatRangeSlider(
          value=[0, 24],
          min=0,
          max=24,
          step=0.1,
          description="Hour of day: "
      )
      slider = MultiRangeSlider([pm25_slider,
```

```
[17]: slider
```

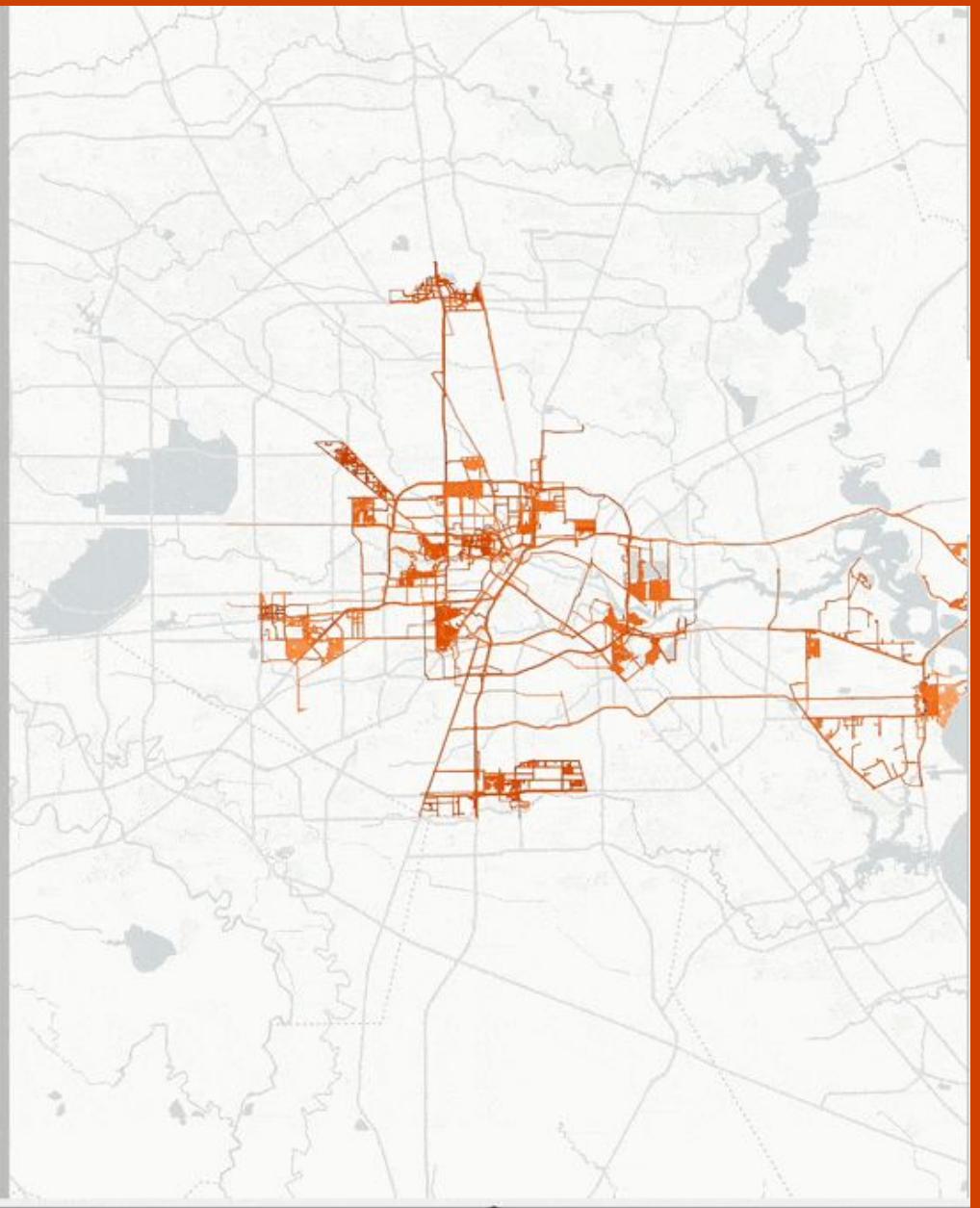
[17]: PM2.5: 0.00 – 100.0

Hour of day: 0.00 – 24.0(

Link the sliders to the visualization layer:

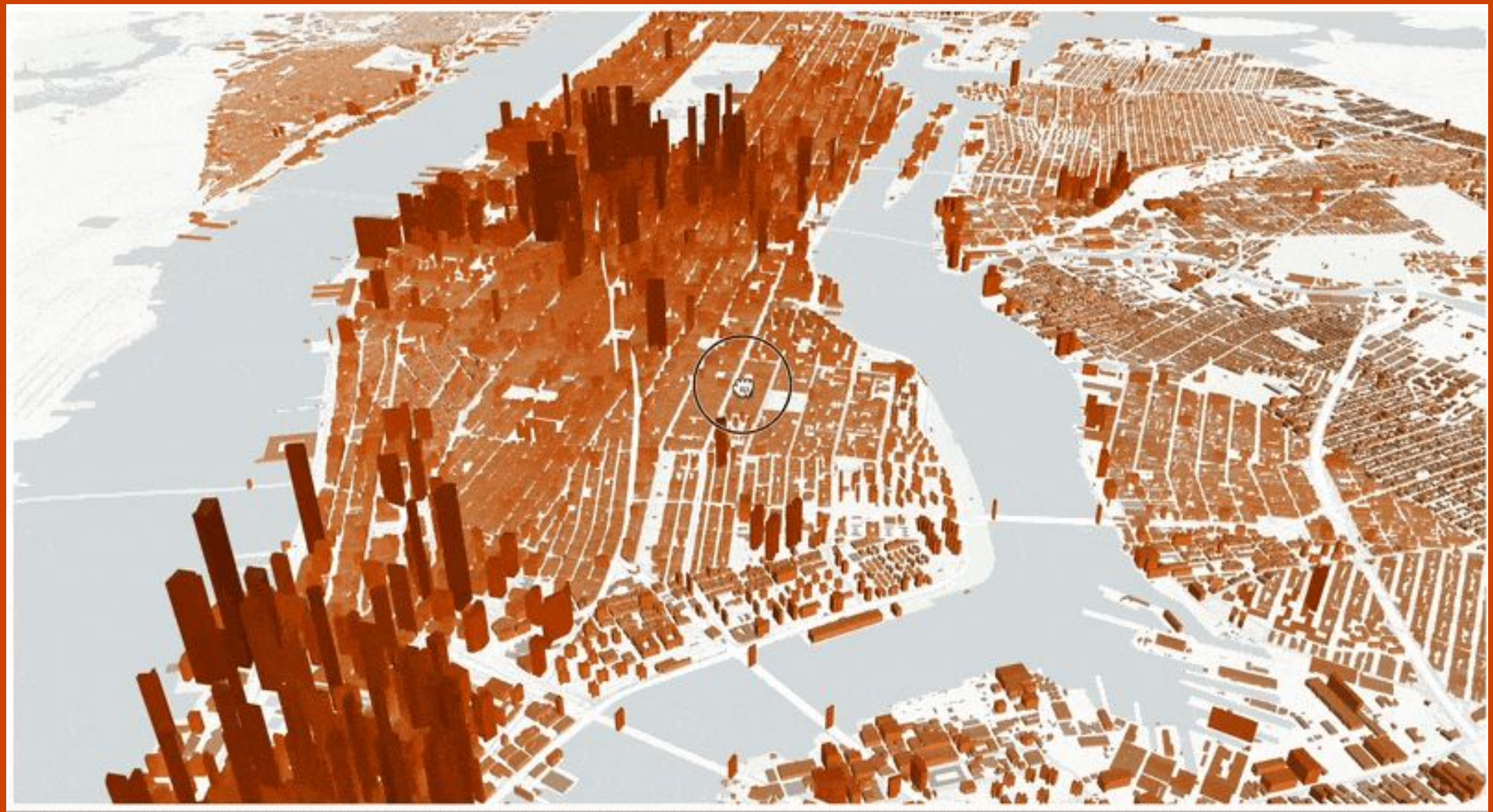
```
[18]: _ = jsdlink(
          (slider, "value"),
          (layer, "filter_range")
      )
```

```
[ ]:
```



developmentSEED

Buildings data



developmentSEED

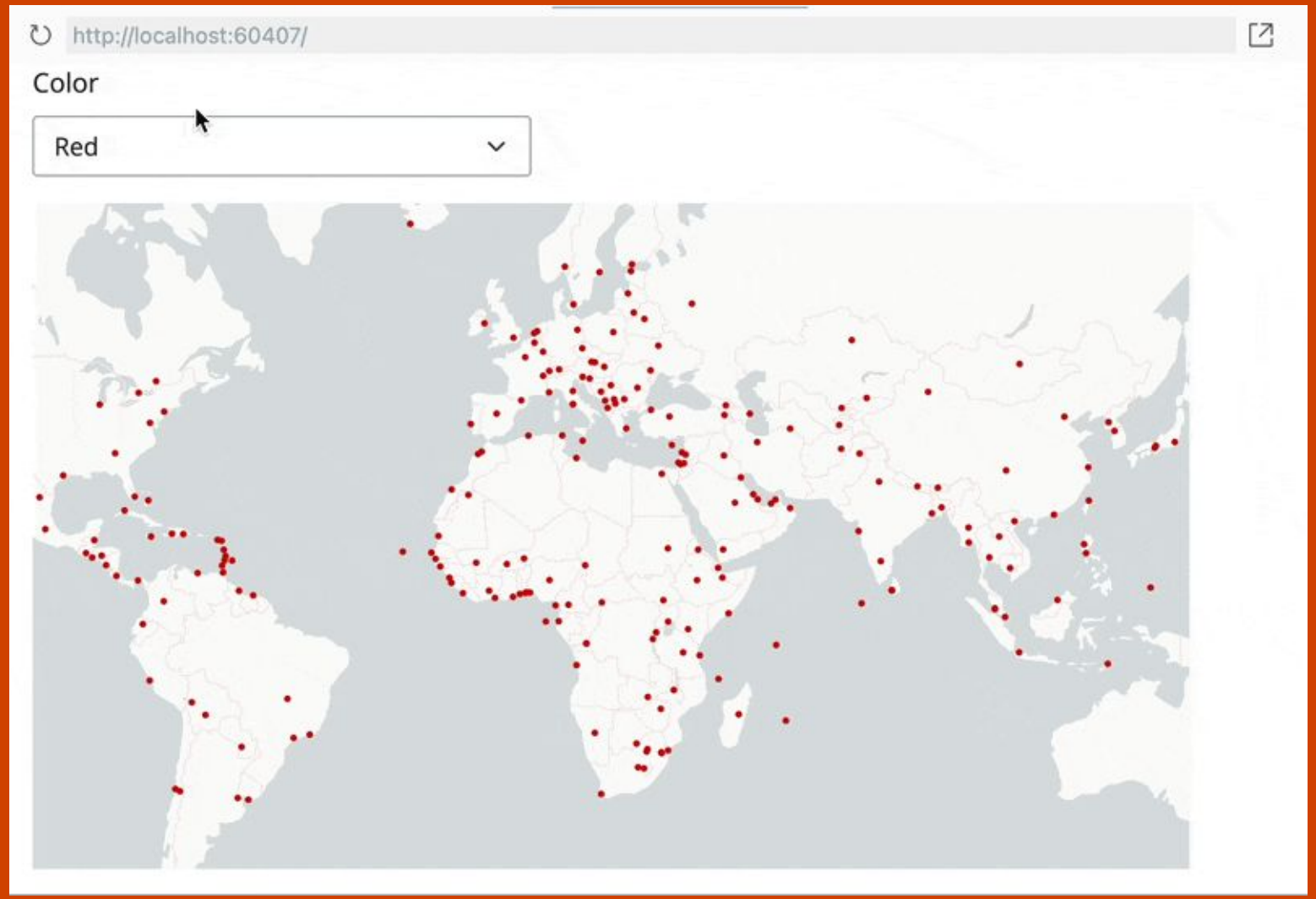
County-to-County Migration

```
[12]: map_ = Map(layers=[source_layer, target_ring_layer, arc_layer], picking_radius=10)
      map_
```

[12]:

developmentSEED

Dashboard integration
e.g. Shiny, Panel

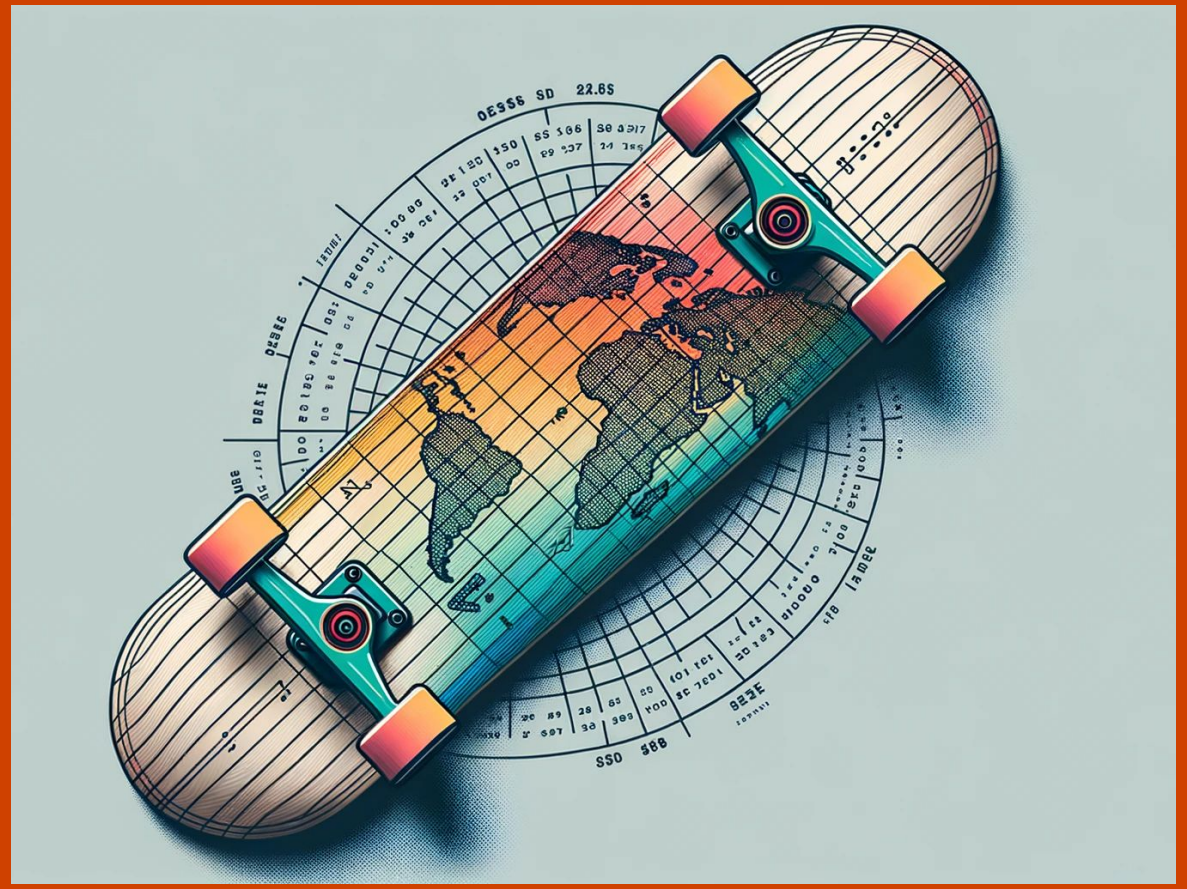


developmentSEED

What’s in a name?

A "deck" is the part of a skateboard you ride on. What's a fast, geospatial skateboard?

A *lon*board.



Credit Kiri Carini & Dall-E & internet skateboarders’ training data

developmentSEED

Demos

developmentSEED

How is it so fast?

developmentSEED

Data rendering on the web

Three parts of vector web maps

Load data to the browser

Manage data in memory

Render to screen

developmentSEED

Data rendering on the web

Three parts of vector web maps (slow)

Load data to the browser

GeoJSON

- Massive file size
- Very slow to read and write
- No partial reads

Manage data in memory

GeoJSON-like JavaScript objects

- Memory overhead
- Can't be shared with web worker or WebAssembly
- High garbage collector overhead (many small JavaScript array objects)

Render to screen

Leaflet

- Slow to iterate over input features and render to HTML canvas element

Binary data pipelines are fast

Three parts of vector web maps (fast)

Load data to the browser

GeoParquet

Fastest way to move data

- Efficient compression
- Spatial/column selection
- Very fast to read and write
- `parquet-wasm` in WebAssembly

Manage data in memory

GeoArrow

Most efficient way to represent geospatial DataFrames

- Binary buffers store entire column's data
- No serialization to share with web worker or WebAssembly
- No garbage collector overhead

Render to screen

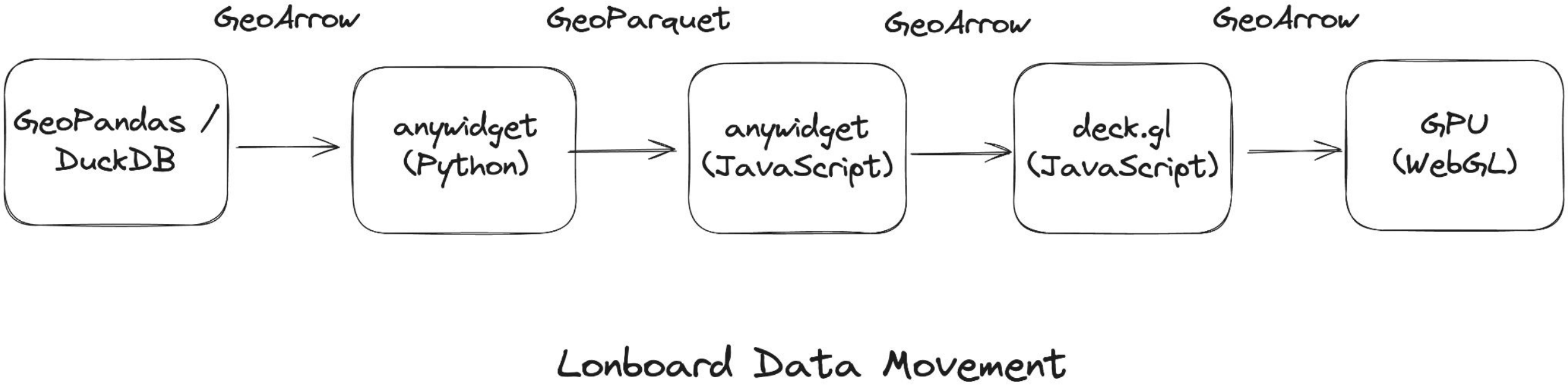
deck.gl

Fastest analytical map rendering on the web

- Copy directly from GeoArrow memory to the GPU
- Instanced (vectorized) draw calls

developmentSEED

End-to-end binary data pipeline



developmentSEED

Lonboard's *ulterior motive*

developmentSEED

Lonboard's *ulterior motive*: Establish a GeoArrow ecosystem

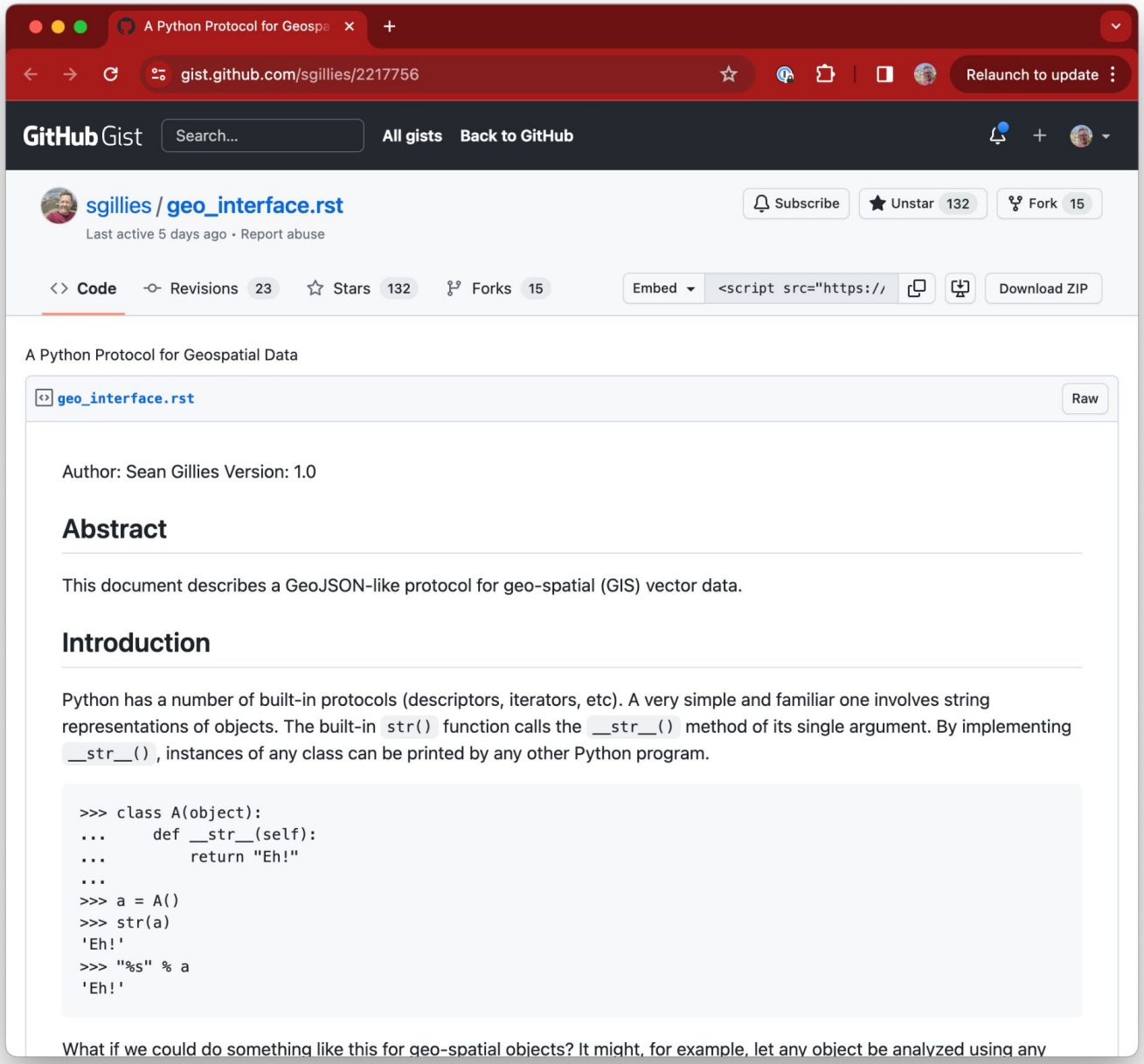
developmentSEED

A fast data protocol for geospatial data

developmentSEED

Data what?

- **Data protocols “unlink” the producer and consumer of data**
  - No knowledge of each other needed
- Python: Geo Interface protocol
- A `__geo_interface__` attribute gives you GeoJSON.
- v1.0 in March 2012



Data protocols enable innovation

- **A `__geo_interface__` attribute gives you GeoJSON.**
- *Many* more implementations. (unknown number!)
- All of these libraries *just work* together.
- A new library can connect to a full ecosystem by implementing a single interface

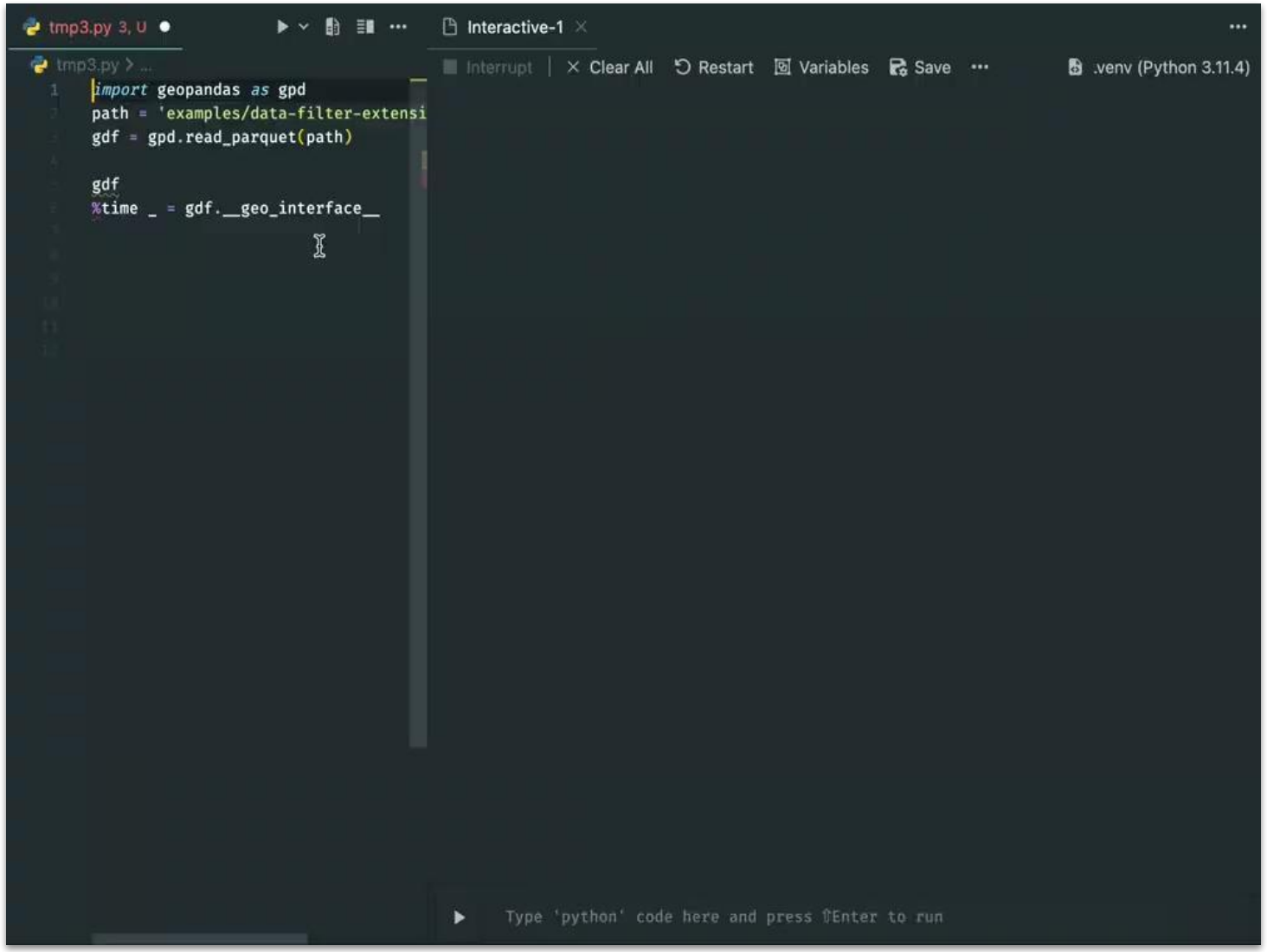
GeoPandas	traffic	fastkml	ArcPy
pyproj	geojson	mapnik	descartes
Shapely	Fiona	PySAL	geojson
PySHP	gdal	lonboard	ezdxf

Selected libraries implementing `__geo_interface__`

developmentSEED

Problem Solved…?

- **Format of the data protocol very different than internal representation**
- Calling `__geo_interface__` triggers *data transformation*
- Overhead for both producer & consumer
- GeoJSON not efficient to operate on, so libraries don't store it directly



1 minute overhead on 3 million point dataset

developmentSEED

Fast data protocols enable innovation

Enabling zero-copy data sharing for scientific Python libraries

Python Buffer Protocol (2006)

- **Problem**: Expensive to copy & convert array data between Python libraries
- ***Binary protocol for array data***
- Enabled widespread innovation on top of NumPy (v1.0 in 2006)

"This cannot be emphasized enough: **it is fundamentally the Buffer Protocol and related NumPy functionality that make Python useful as a scientific computing platform.**"

– JAKE VANDERPLAS (An Introduction to the Python Buffer Protocol)

https://jakevdp.github.io/blog/2014/05/05/introduction-to-the-python-buffer-protocol/

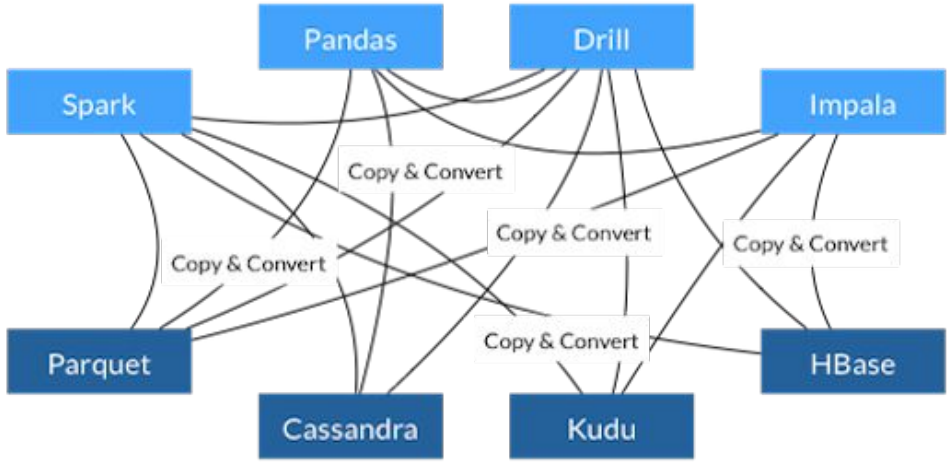
developmentSEED

Binary data protocol for tabular data

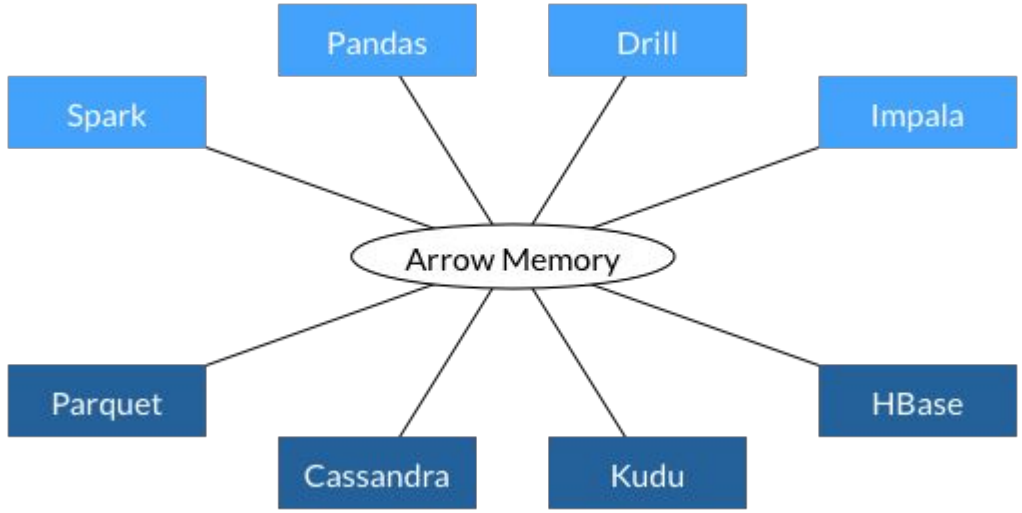
Apache Arrow (2015)

- **Problem**: Expensive to copy & convert *tabular* data between libraries (e.g. Pandas) and languages (e.g. Python - R)
- ***Binary data protocol for tabular data.***
- Language-independent, zero-copy data sharing

Before Arrow



With Arrow

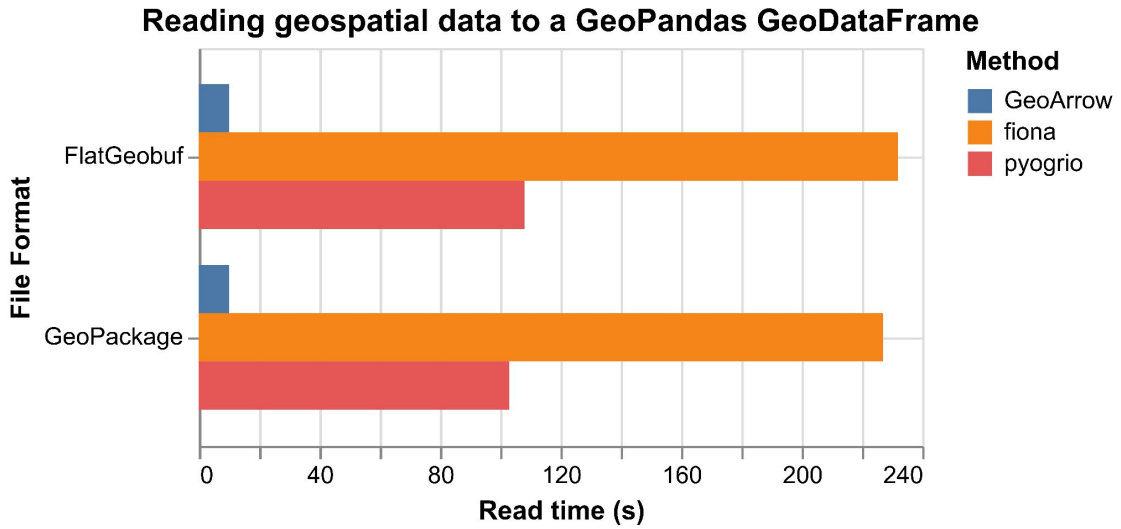


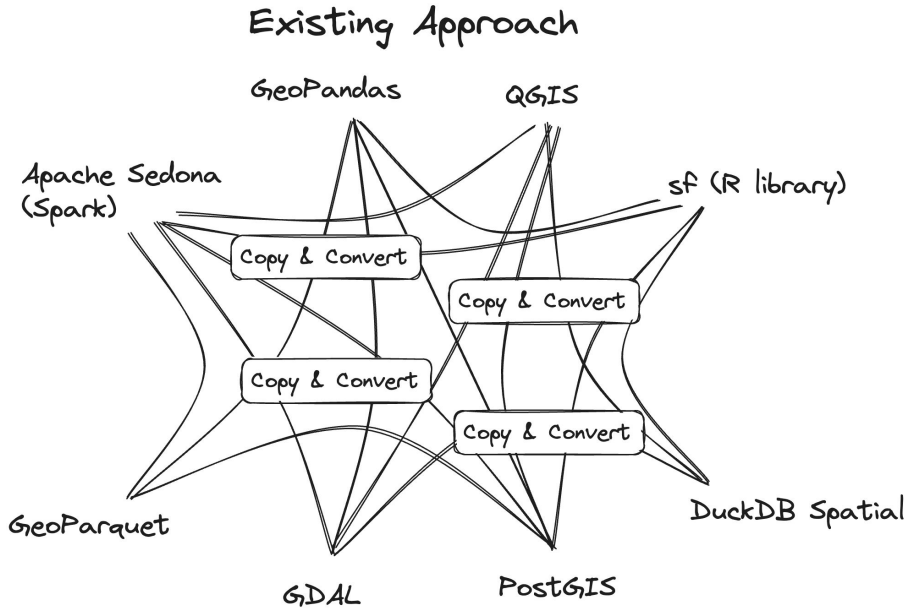
developmentSEED

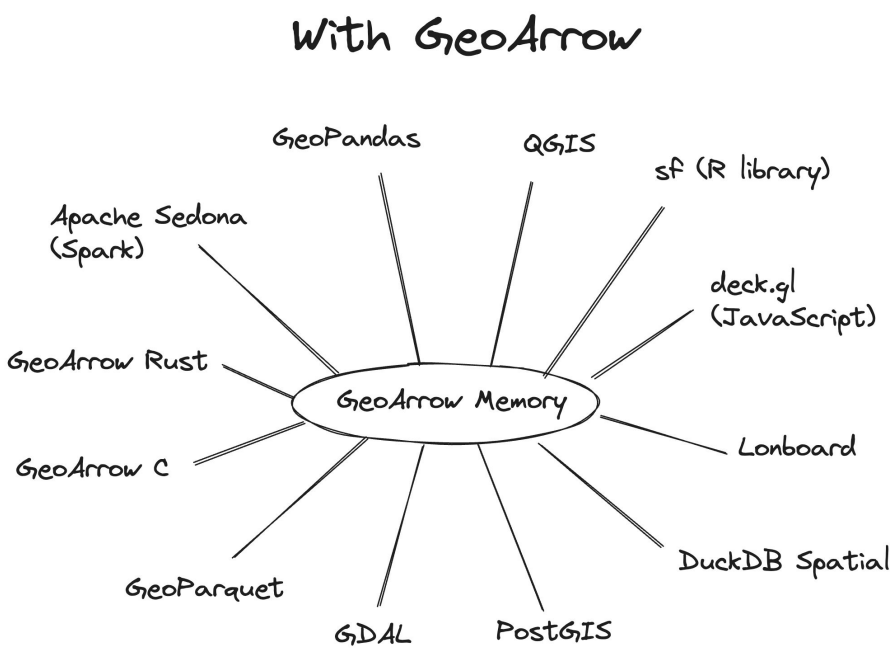
Enabling zero-copy data sharing for geospatial

GeoArrow (2020)

- **Problem**: Expensive to copy & convert *geospatial* tabular data between libraries and languages
- ***Binary data protocol for geospatial tabular data.***
- Language-independent, zero-copy data sharing



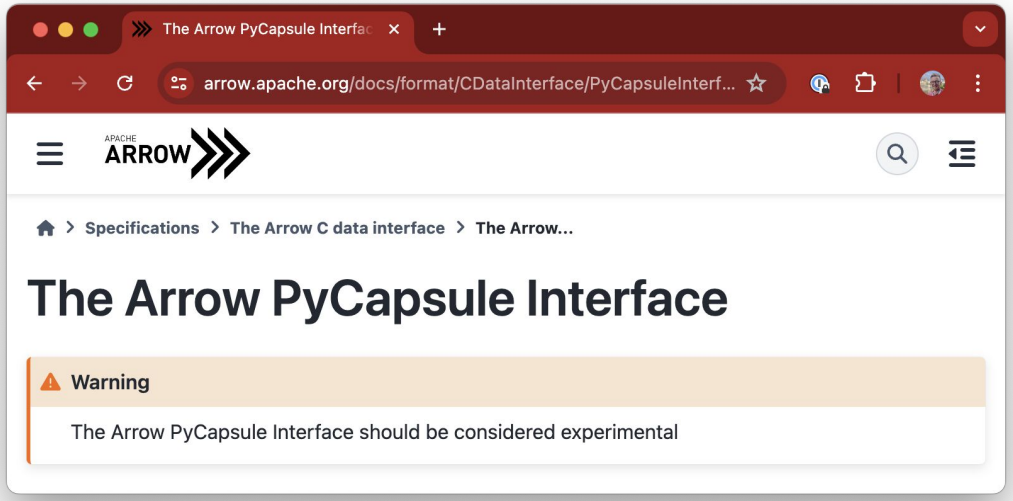




developmentSEED

Enabling zero-copy data sharing across Python libraries

Arrow PyCapsule Interface (2023)



- **Python protocol** for zero-copy sharing of Arrow data
- Arrow itself is just a format; this interface describes how Python packages should expose/ingest that format
- Works for **Arrow and GeoArrow** out of the box*
- Lonboard supports every such library (as long as they provide data with geospatial metadata)

`pyarrow`	`nanoarrow`	`arro3`
`pyogrio`	`gdal`	`polars`
`quak`	`pandas`	`vegafusion`

Selected libraries implementing PyCapsule Interface

`fastexcel`	*`daft`*	*`lance`*
`datafusion`	*`duckdb`*	*`lightgbm`*

Libraries with open issue for PyCapsule Interface

developmentSEED

Enabling zero-copy data sharing across Python libraries

Arrow PyCapsule Interface (2023)

- **Python protocol** for zero-copy sharing of Arrow data
- Arrow itself is just a format; this interface describes how Python packages should expose/ingest that format
- Works for **Arrow and GeoArrow** out of the box*
- Lonboard supports every such library (as long as they provide data with geospatial metadata)

```
[1]: %load_ext quak

[2]: from arro3.io import read_parquet

[3]: table = read_parquet("athletes.parquet")

[4]: table
```

[4]:

	id int64	name utf8	nationality utf8	sex utf8	date_of_birth date32[day]	heigl floate
	0 – 1G	unique	207 categories	male / female 2 categories		Ø1.2
1	532,037,425	A Lam Shin	KOR	female	1986-09-23	
0	736,041,664	A Jesus Garcia	ESP	male	1969-10-17	
1	532,037,425	A Lam Shin	KOR	female	1986-09-23	
2	435,962,603	Aaron Brown	CAN	male	1992-05-27	
3	521,041,435	Aaron Cook	MDA	male	1991-01-02	
4	33,922,579	Aaron Gate	NZL	male	1990-11-26	
5	173,071,782	Aaron Royle	AUS	male	1990-01-26	

quak: a scalable data profiler

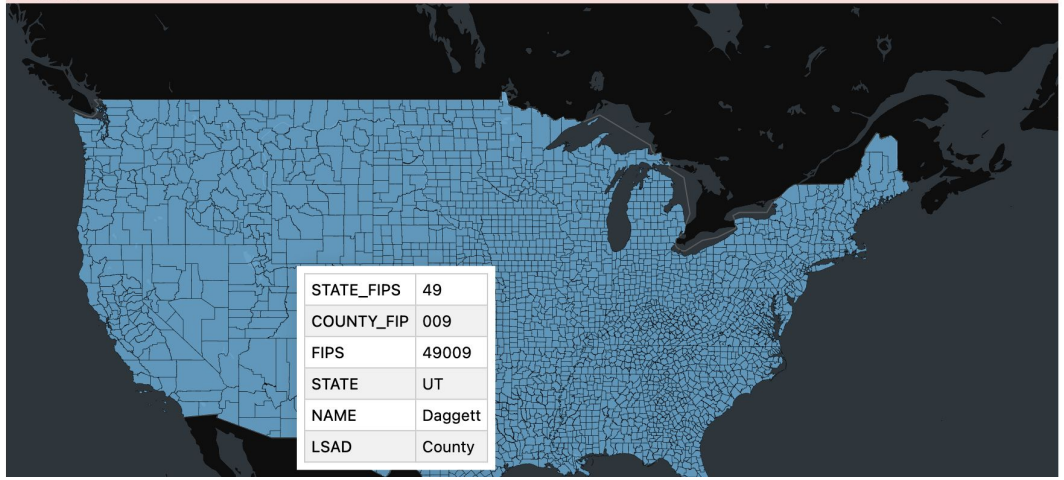
```
[1]: from geoarrow.rust.core import read_flatgeobuf
     from lonboard import viz

[2]: path = "/Users/kyle/Downloads/UScounties.fgb"

[3]: viz(read_flatgeobuf(path))
```

```
/Users/kyle/tmp/lonboard-geoarrow-rs/env/lib/python3.11/site-packages/lonboard/_geoarrow/ops/reproject.py:23:
UserWarning: No CRS exists on data. If no data is shown on the map, double check that your CRS is WGS84.
  warn(
```

[3]:



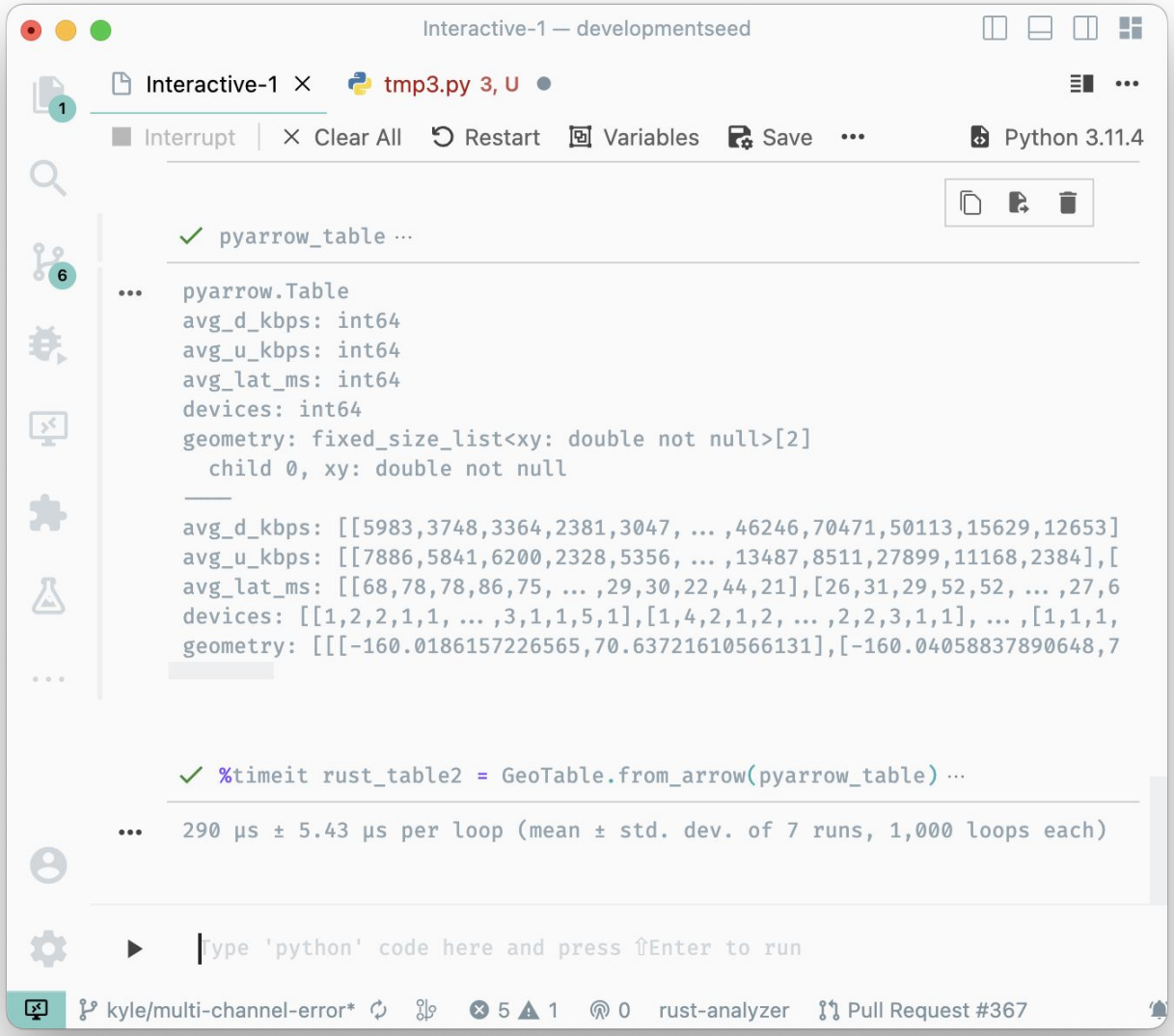
Lonboard: Pushing towards a GeoArrow ecosystem

- Lonboard's *end goal*: to jumpstart an Arrow-native ecosystem for geospatial data
- Ecosystem network effects
  - If Lonboard is widely used, then more libraries will interface with Arrow
  - The more libraries built around Arrow, the higher Lonboard's value

The commoditization of geospatial databases

GeoArrow in Python

- Recall the original geo interface example that took 60s
- 290 *microseconds* to move data from Python -> Rust
- >185,000x faster than `__geo_interface__` for this dataset. Even including consuming

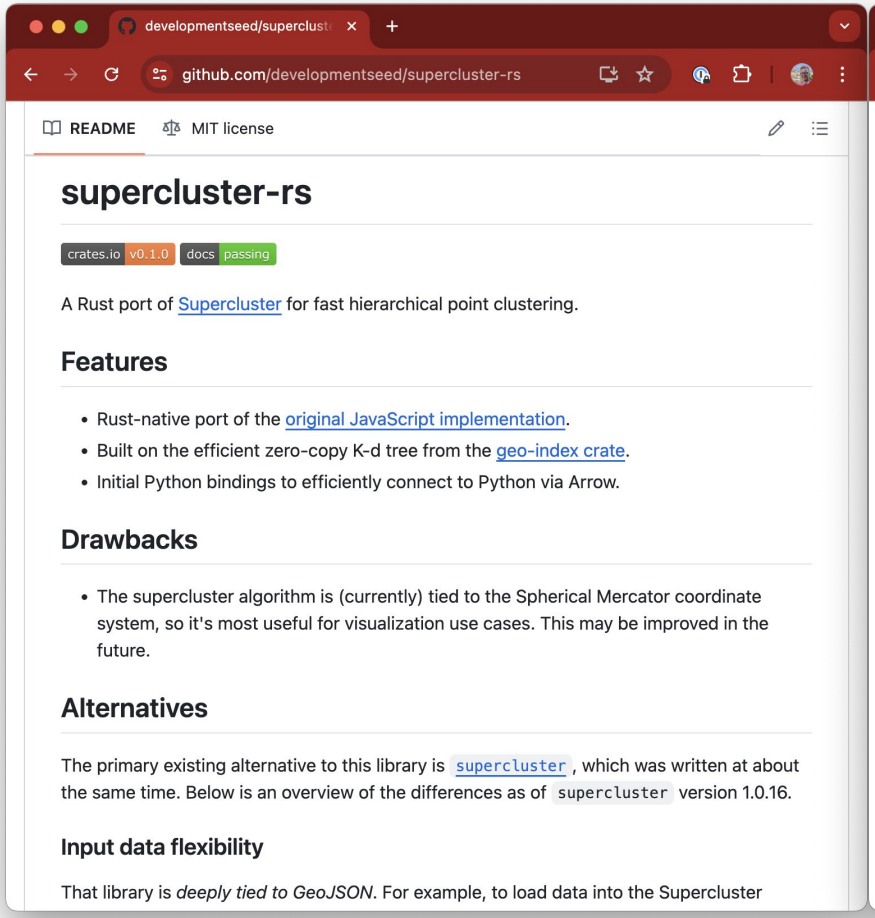


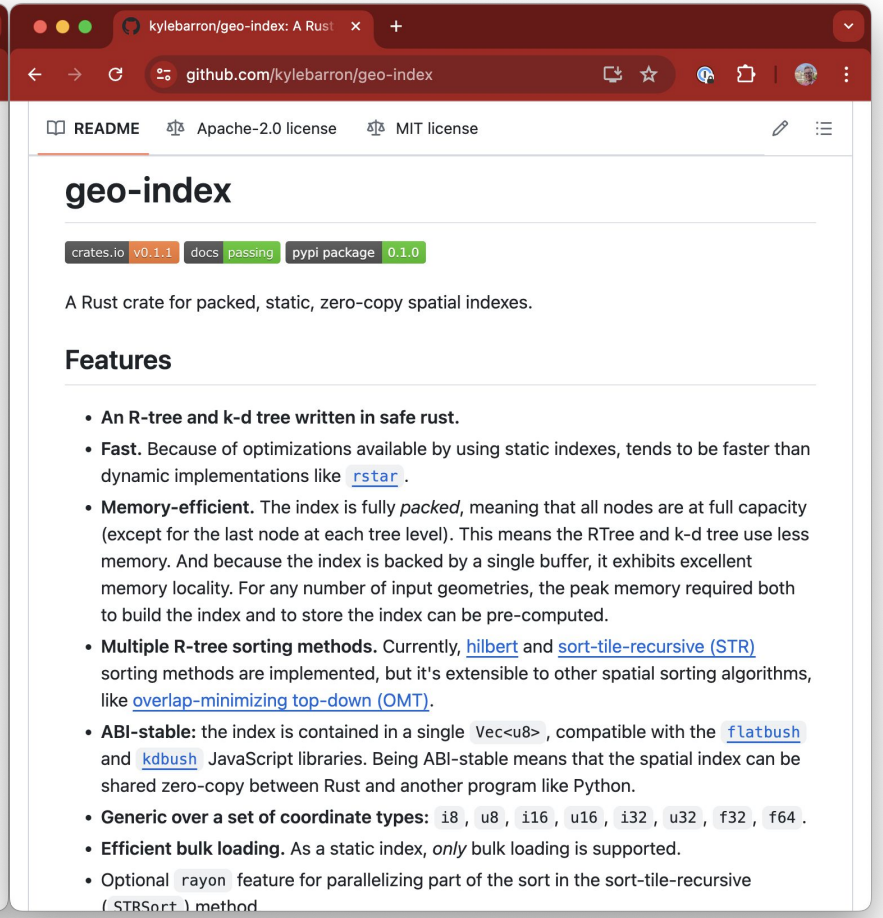
developmentSEED

Future work

Scalable rendering

- Today *all user input* is sent to browser
- In the future:
  - Smarter relationship between Python and JavaScript
  - Efficient spatial indexes in Rust/Python
  - Transfer small portions of input, depending on map's view





developmentSEED

Future Work: Trajectory data

ds.io/lonboard

Code

Notebook

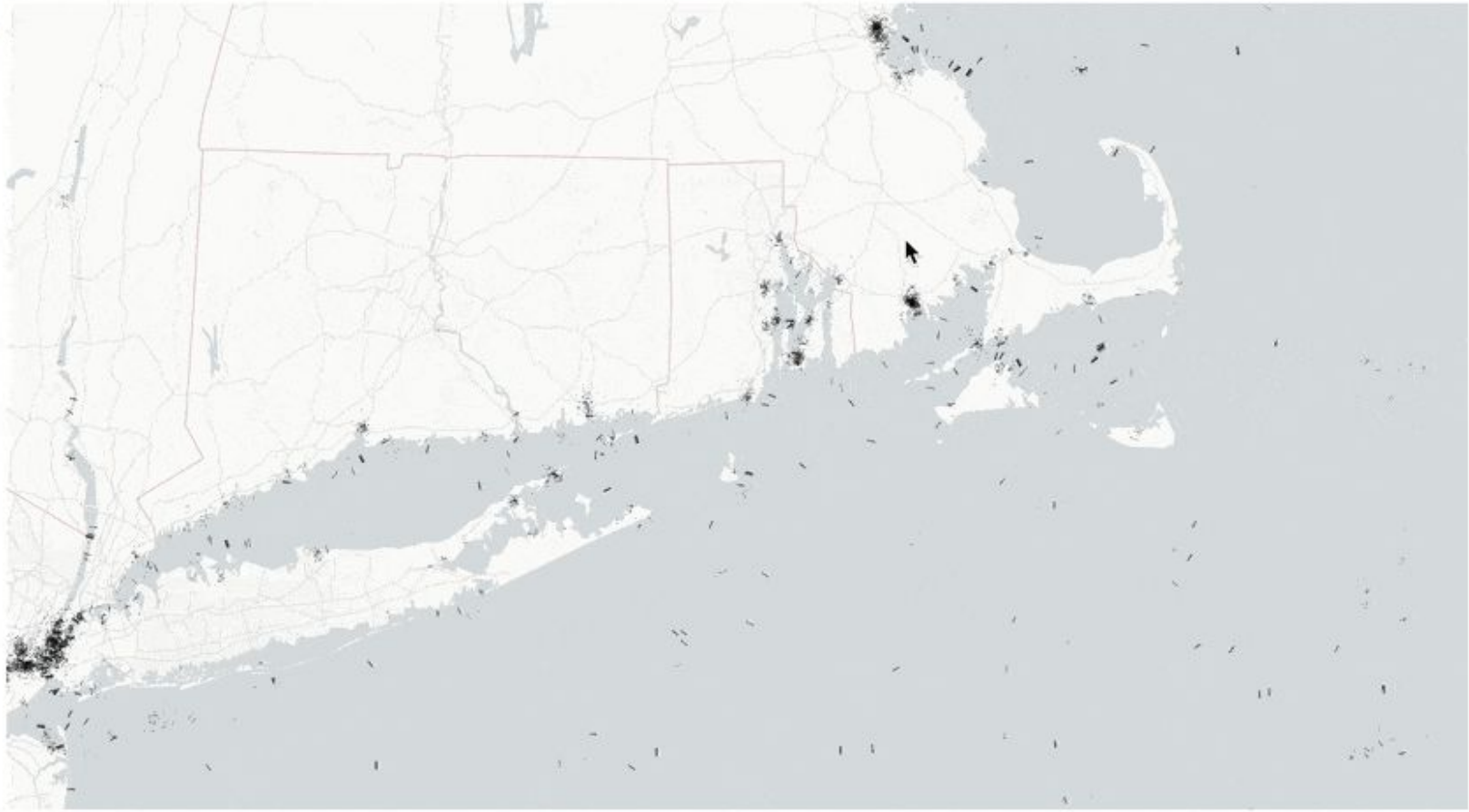
lonboard

```
re parsing is consistent and as-expected, please specify a format.
  df[t] = to_datetime(df[t])
```

[3]:

```
layer = TripsLayer.from_movingpandas(traj_collection)
m = Map(layers=[layer])
m
```

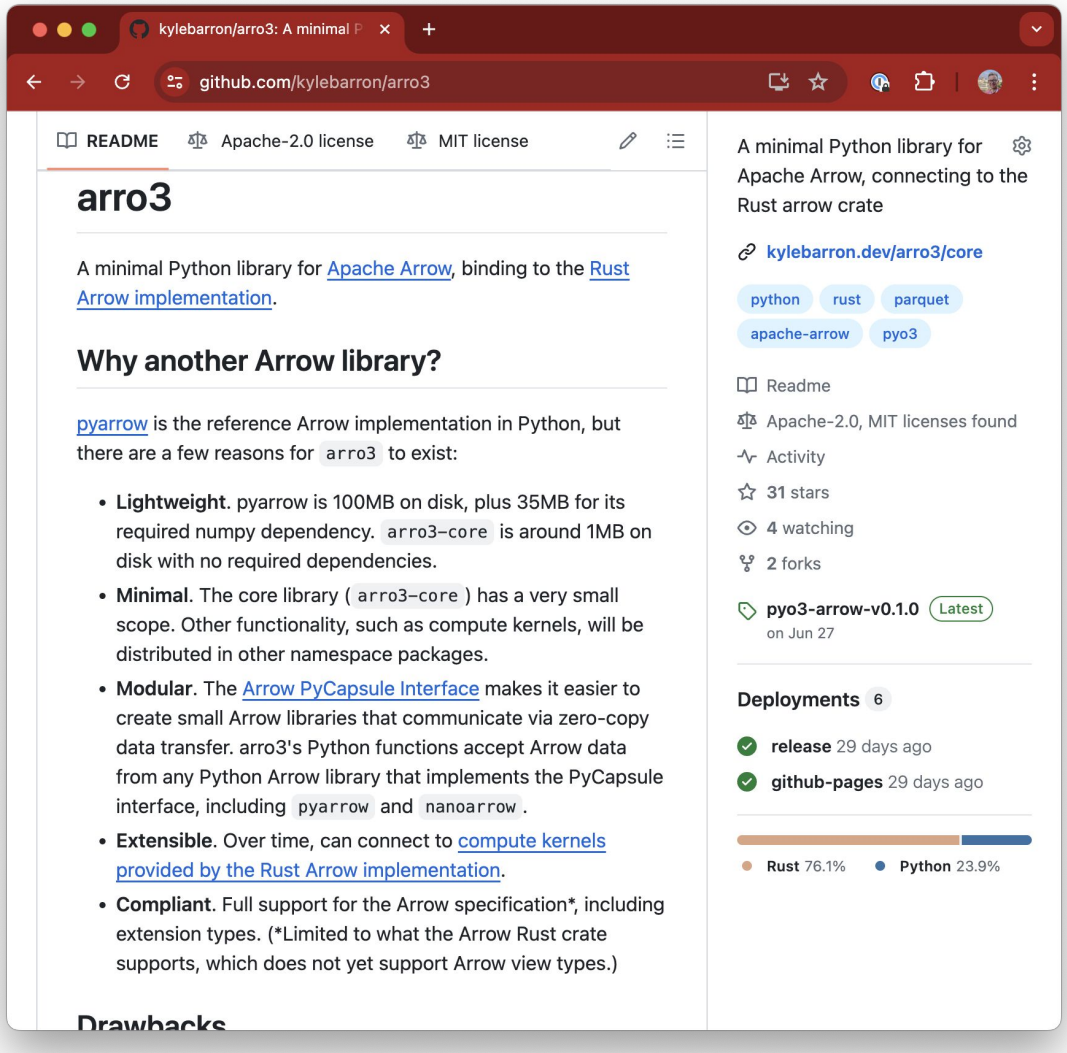
[3]:



developmentSEED

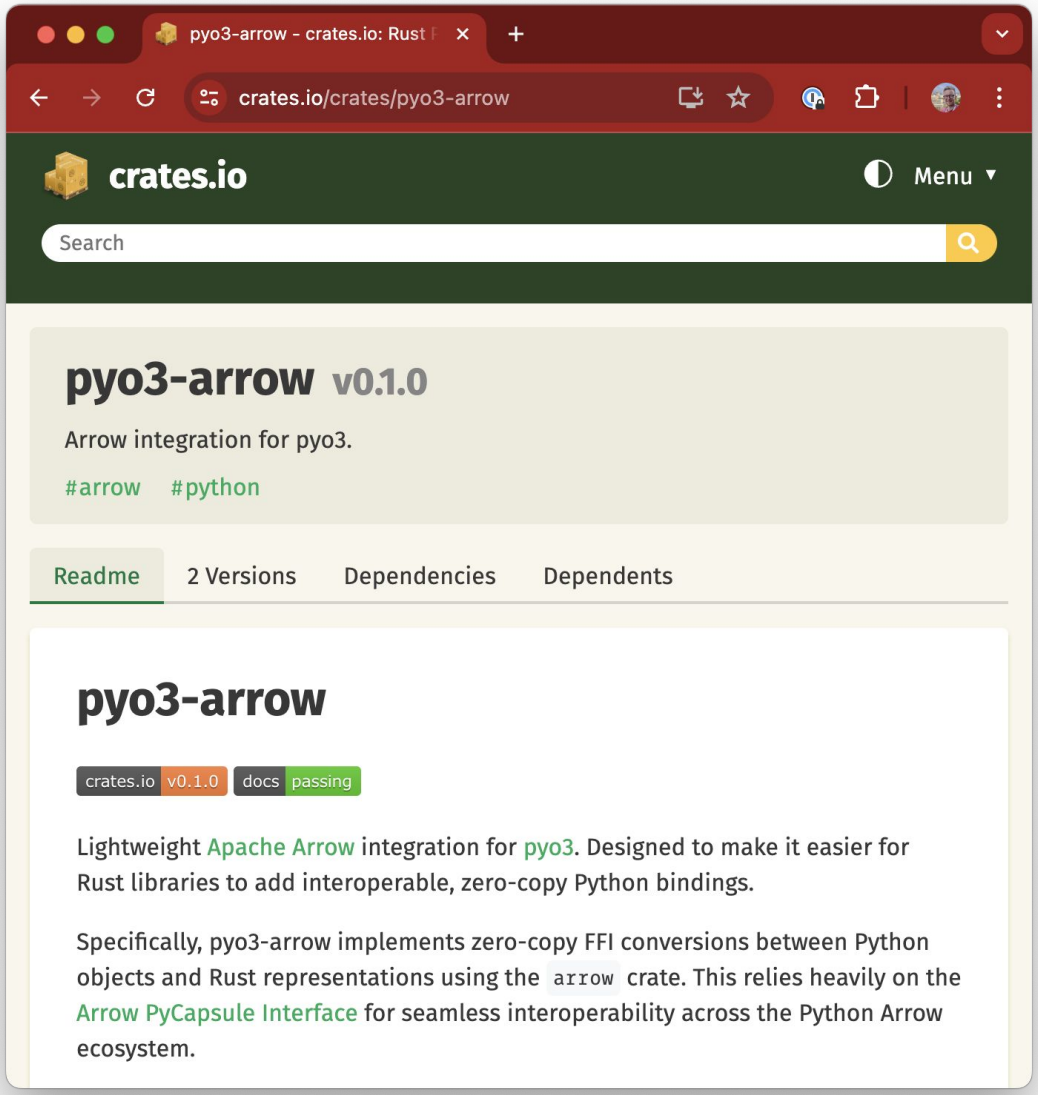
Lonboard: Slim down dependencies

- No required dependencies on Pandas, GeoPandas, Shapely, pyarrow
- Swap out `pyarrow` for arro3
  - Enables Lonboard in Pyodide in WebAssembly (fully client side, no server)
- Treat Arrow as the first-class citizen; everything else is a connector.



Grow Arrow/GeoArrow ecosystems

- Advocate for/implement PyCapsule interface in more libraries
- Make it easier for developers to add Arrow
  - `pyo3-arrow` simplifies passing Arrow data to/from Rust and Python
- Push for geospatial support and/or extensions:
  - geoarrow-rust
  - GeoPolars

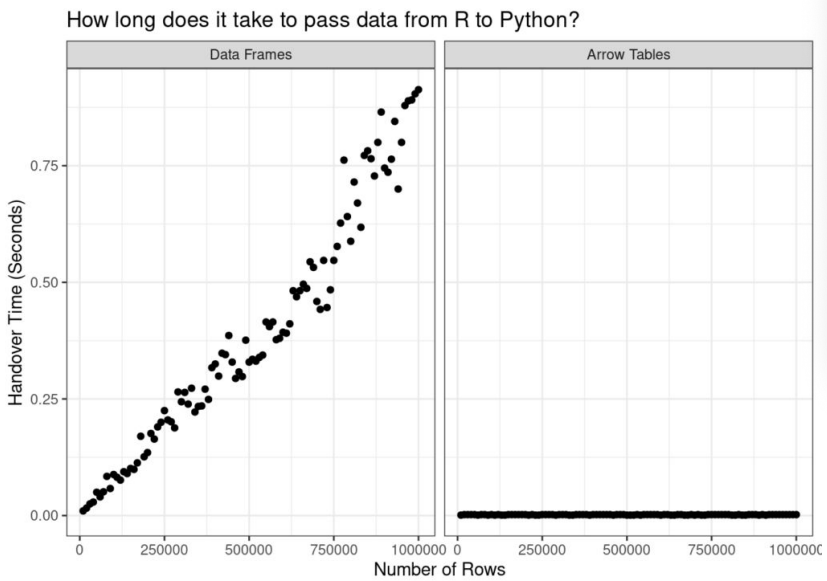


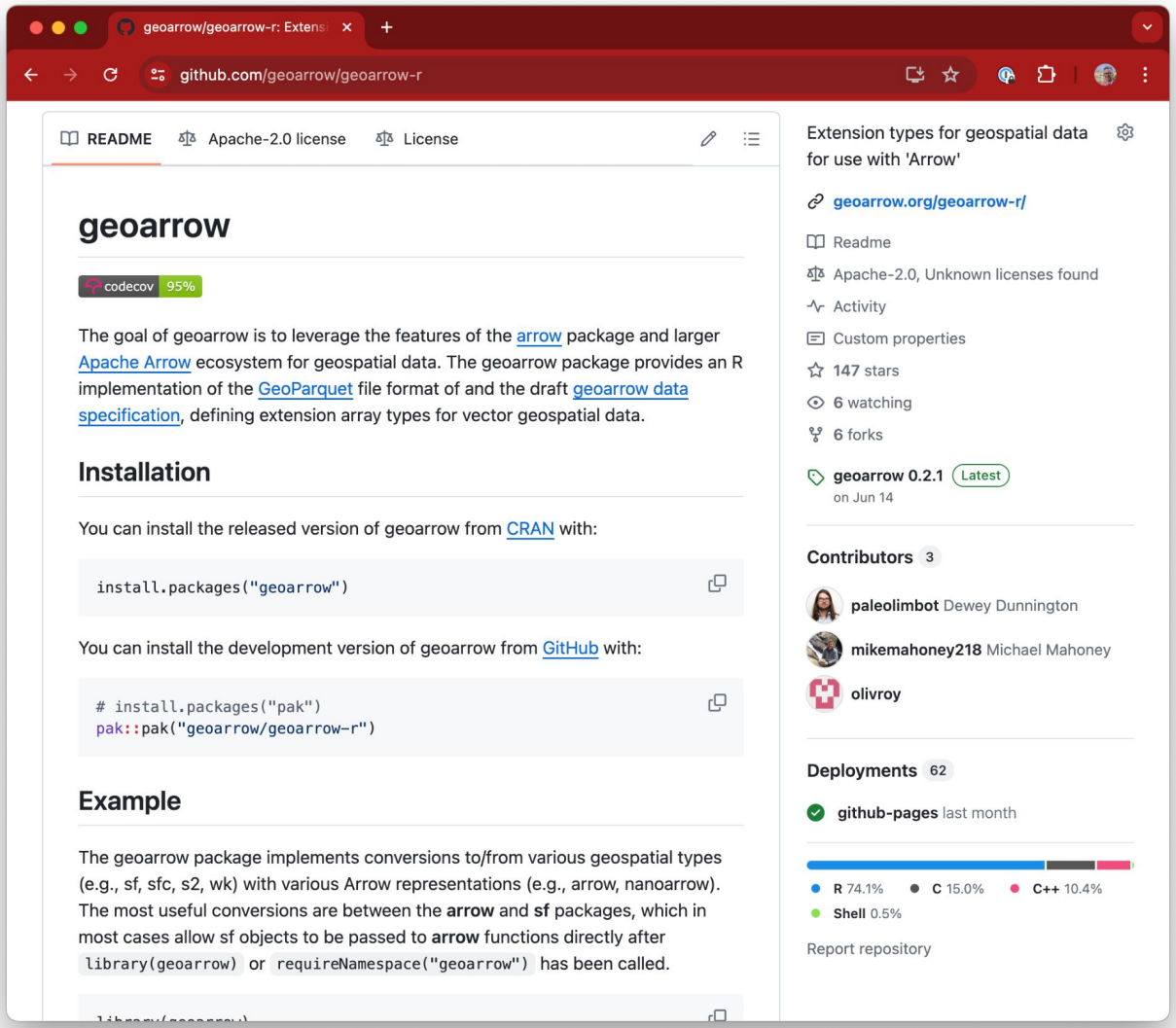
An exercise for the reader

Lonboard + Reticulate

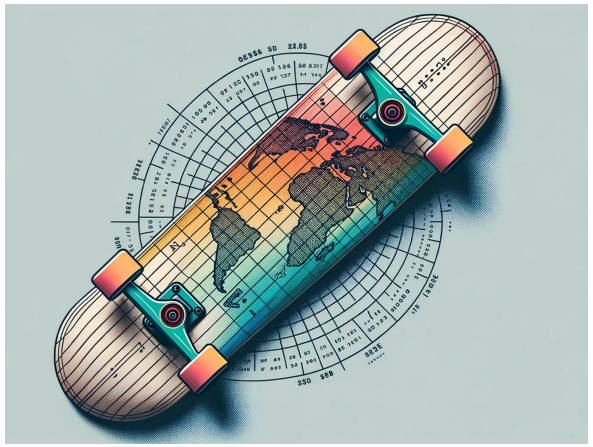
- Should be possible to connect Lonboard - Reticulate - geoarrow-r
- PyCapsule interface should work with reticulate
- Zero-copy data sharing between R and Python 👉

Danielle Navarro (Passing Arrow data between R and Python with reticulate)





developmentSEED



Learn More

- ds.io/blog
- ds.io/Lonboard
- geoarrow.org
- anywidget
- Podcasts:
  - Mapscaping (GeoParquet)
  - Matt Forrest (Lonboard & GeoArrow)
  - Browsertech (JavaScript/WebAssembly)