

Lonboard: Fast, interactive geospatial data visualization

With a heavy dose of Apache Arrow

Kyle Barron, Development Seed



Contents

1 My background

2 What is Lonboard? & Demos

3 How is Lonboard so fast?

4 Data protocols for geospatial data

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...)



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

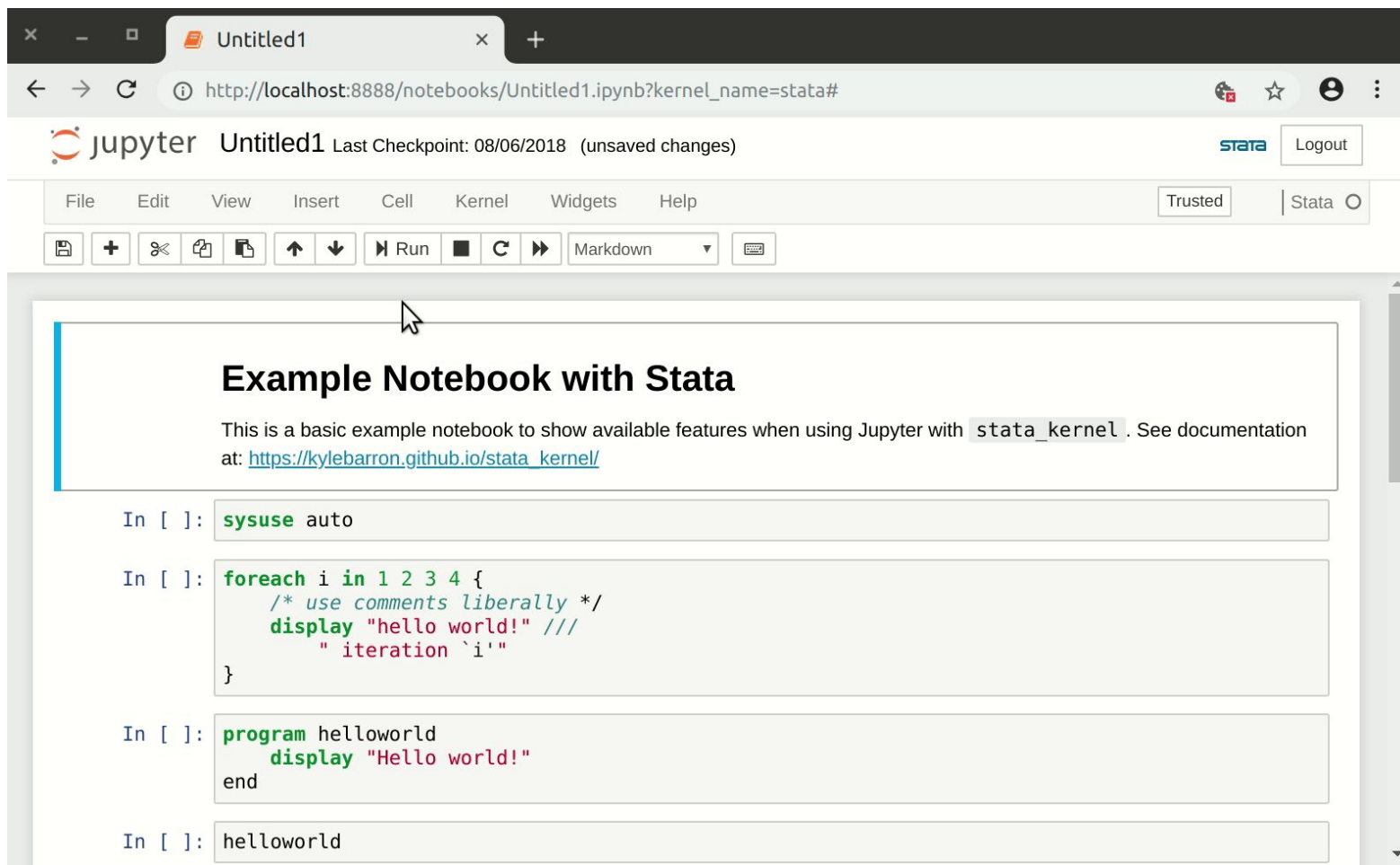
README GPL-3.0 license

stata_kernel

build error downloads 195k downloads/month 979

NBER
National Bureau of Economic Research

STATA



Untitled1

http://localhost:8888/notebooks/Untitled1.ipynb?kernel_name=stata#

jupyter Untitled1 Last Checkpoint: 08/06/2018 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted Stata

Example Notebook with Stata

This is a basic example notebook to show available features when using Jupyter with `stata_kernel`. See documentation at: https://kylebarron.github.io/stata_kernel/

```
In [ ]: sysuse auto
```

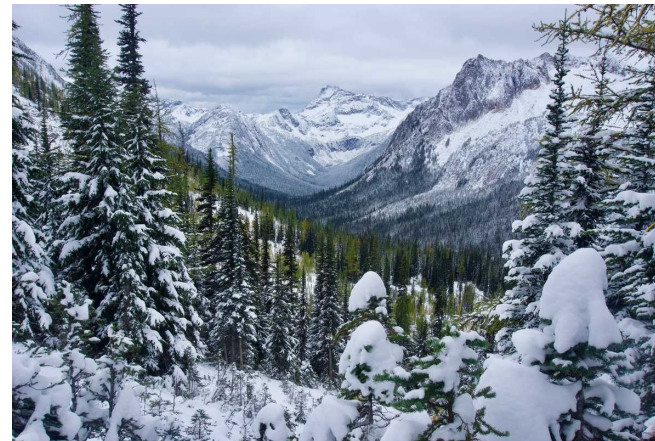
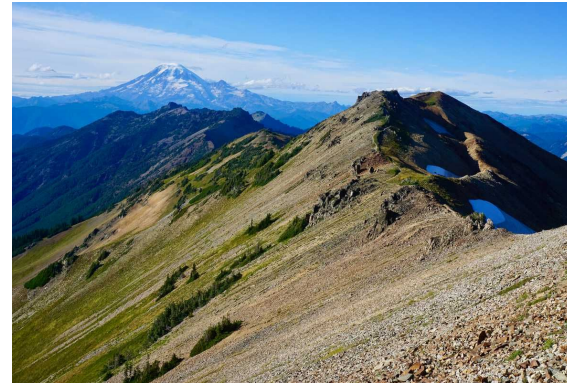
```
In [ ]: foreach i in 1 2 3 4 {
/* use comments liberally */
display "hello world!" ///
" iteration `i'"
}
```

```
In [ ]: program helloworld
display "Hello world!"
end
```

```
In [ ]: helloworld
```


Went for a hike!

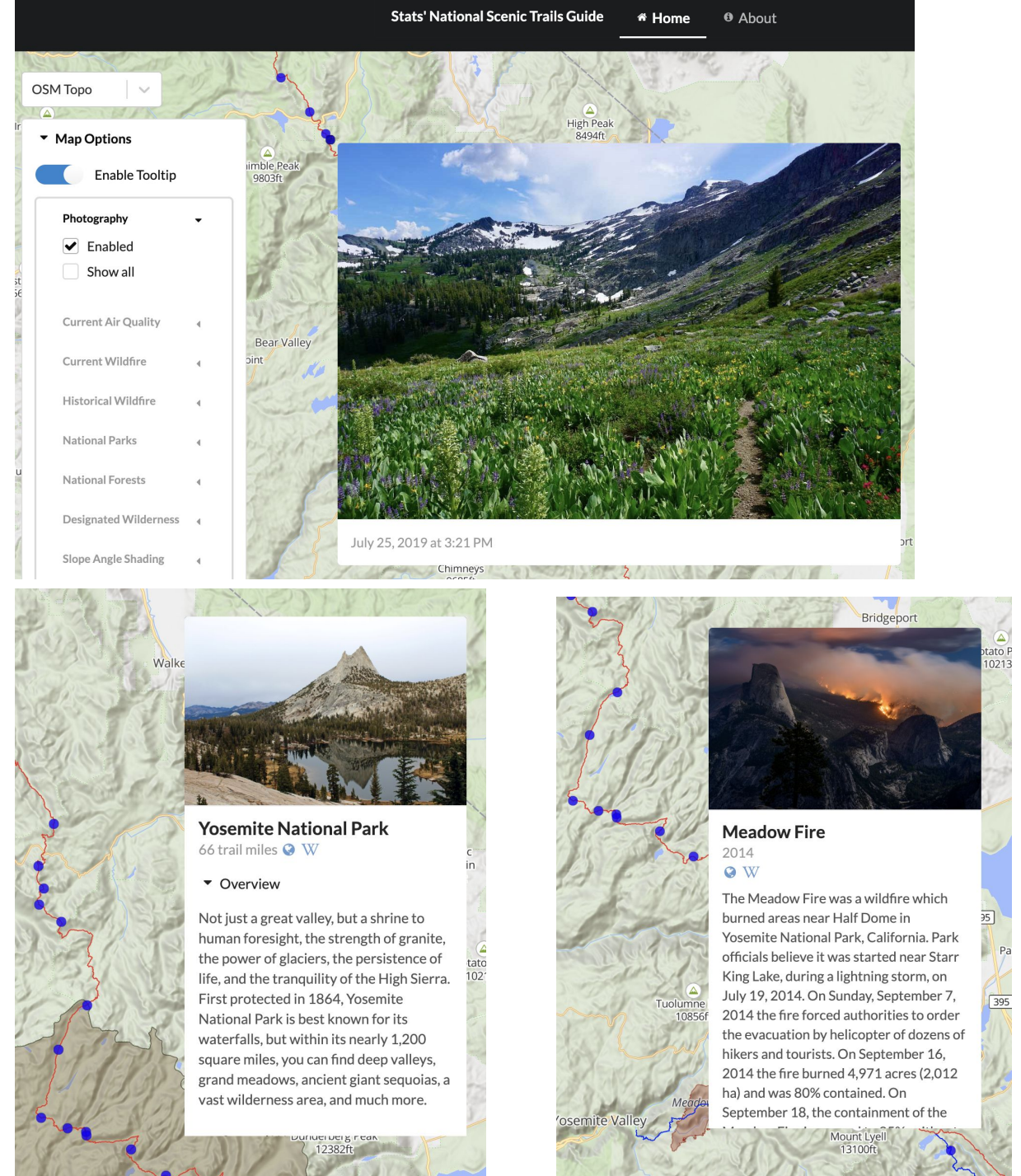
- Hiked 2650-mile Pacific Crest Trail over 5 months
- Halfway through decided that I could build a better hiking app for the PCT

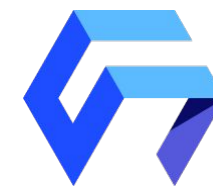


Back in the parents' nest

Built a map!

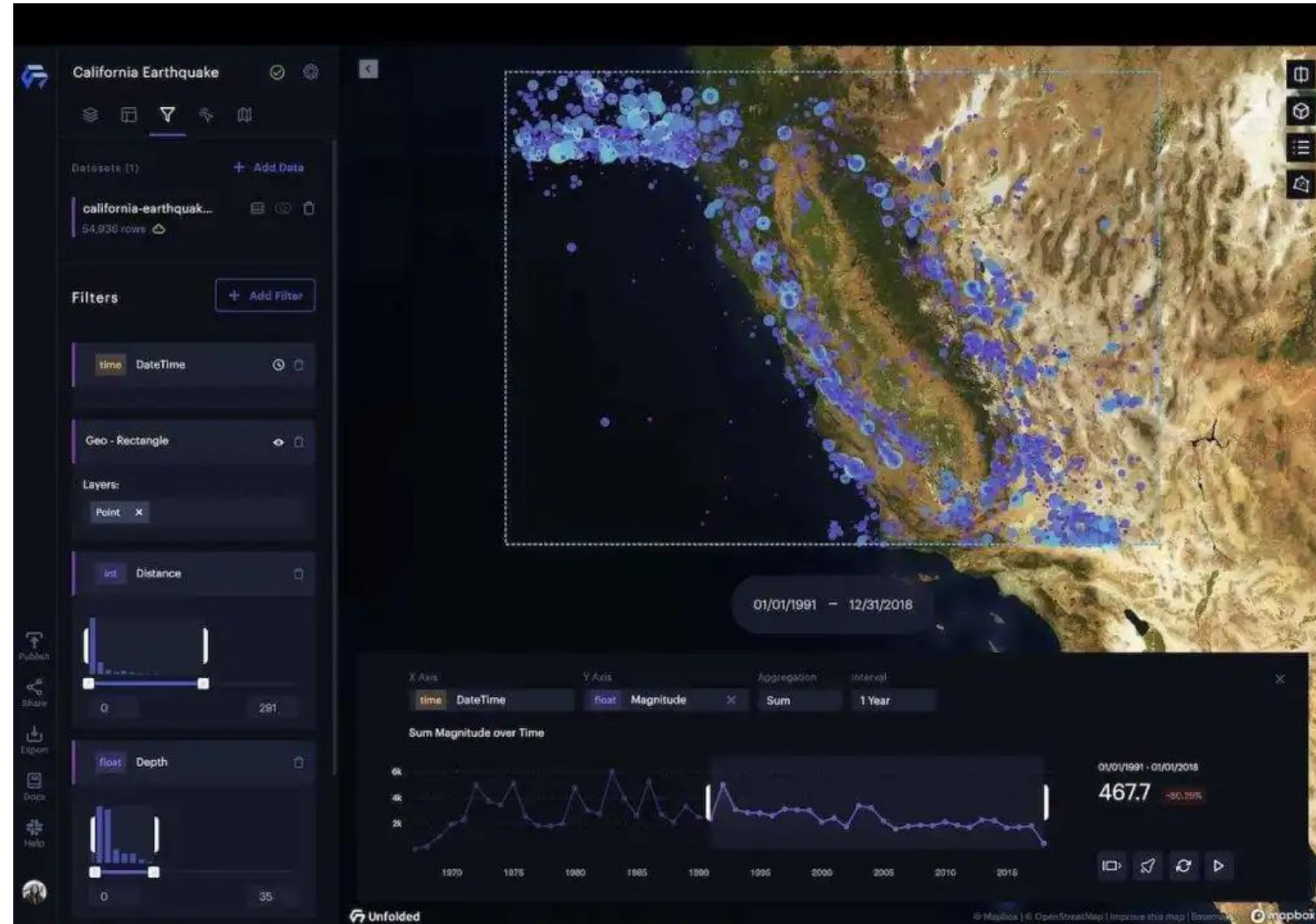
- Joined geospatial data sources with **geopandas**
- Learned about coordinate reference systems
- Created my own OSM vector tiles, hillshading, contour lines, raster tiles
- Learned JavaScript, React
- Used Mapbox GL, deck.gl
- Still online nst.guide and on github github.com/nst-guide





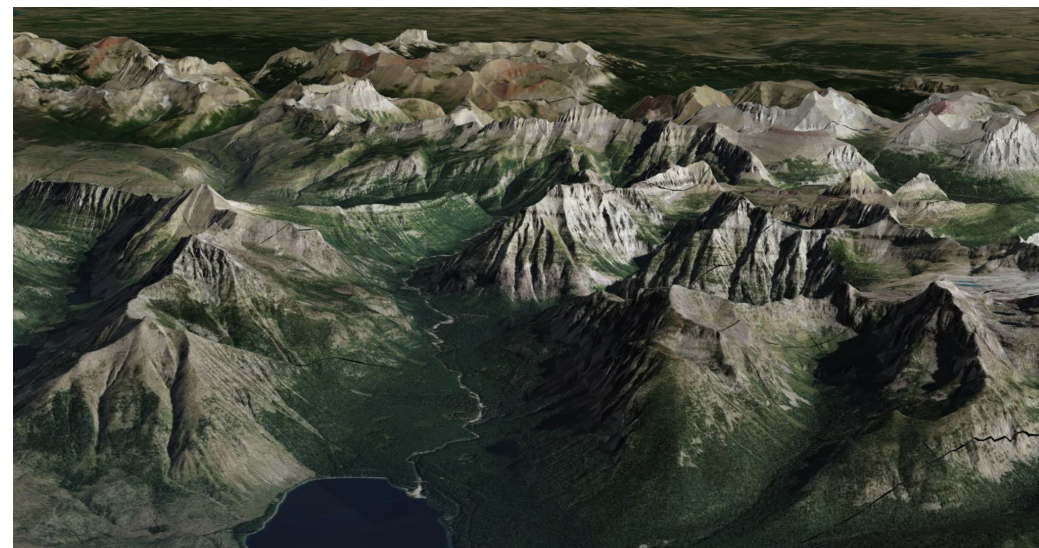
Landed a job!

- 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



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

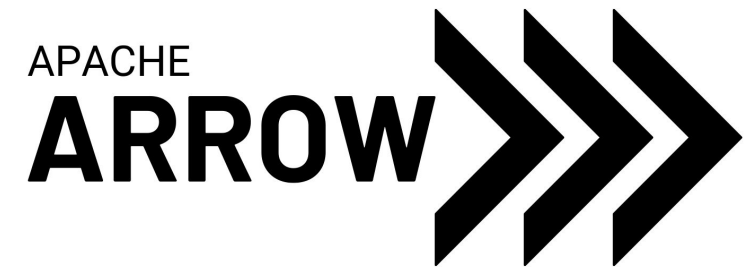


Rust-based WebAssembly bindings to read and write Apache Parquet data

 Rust 485 19

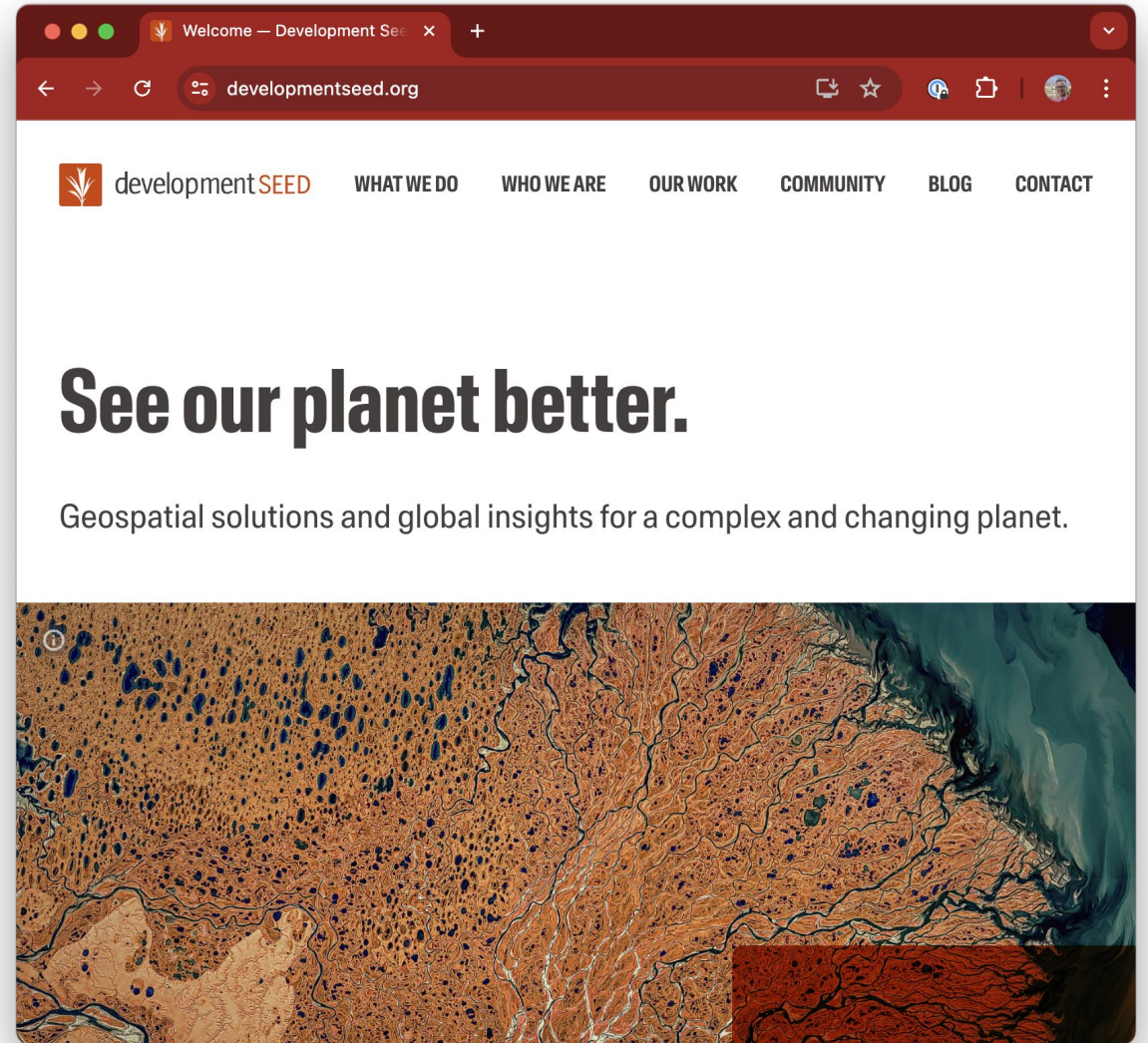
Developed my technical interests

- 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



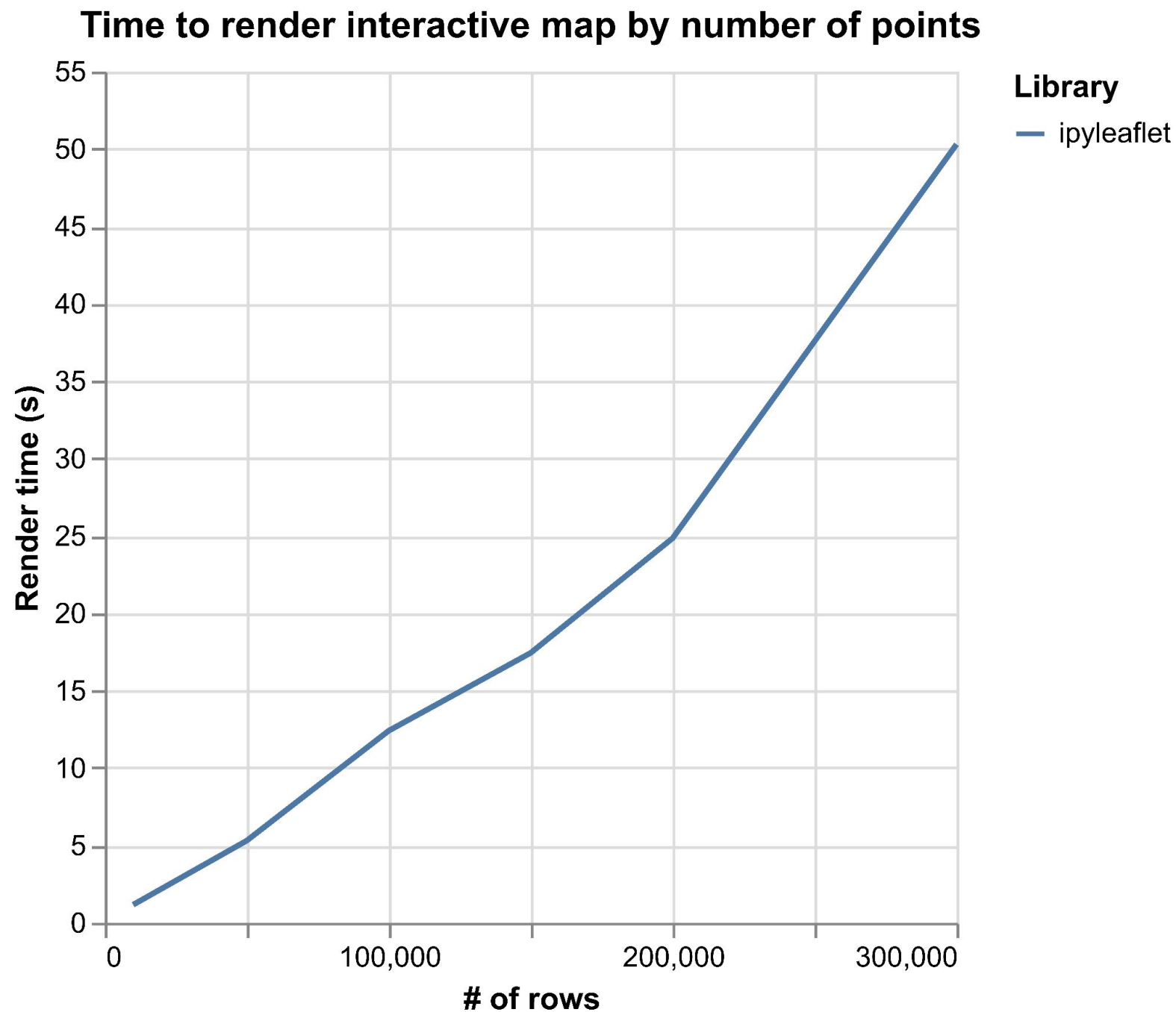


Simpler, faster rendering of large analytical vector datasets (in Python)

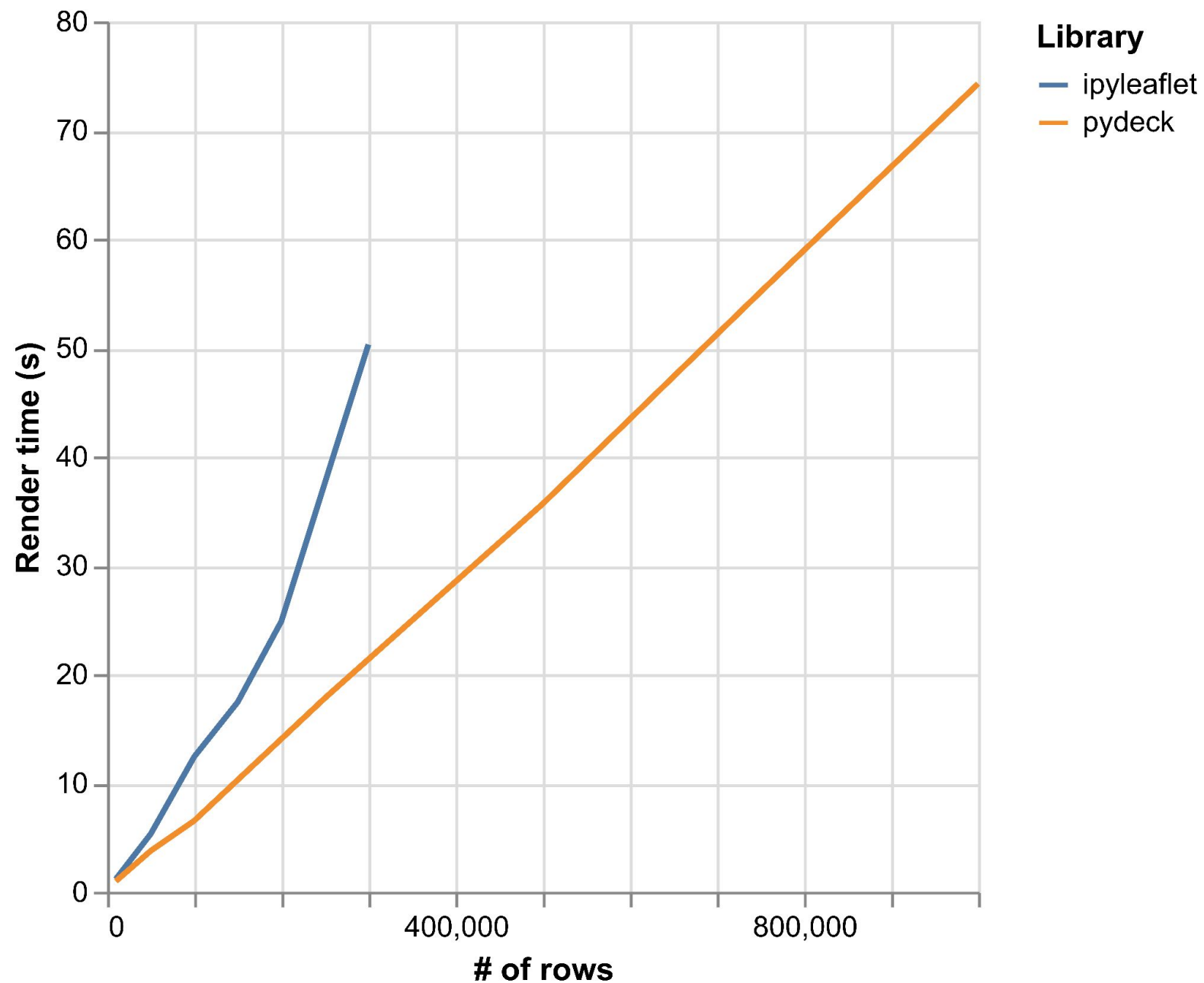


...why not existing libraries?

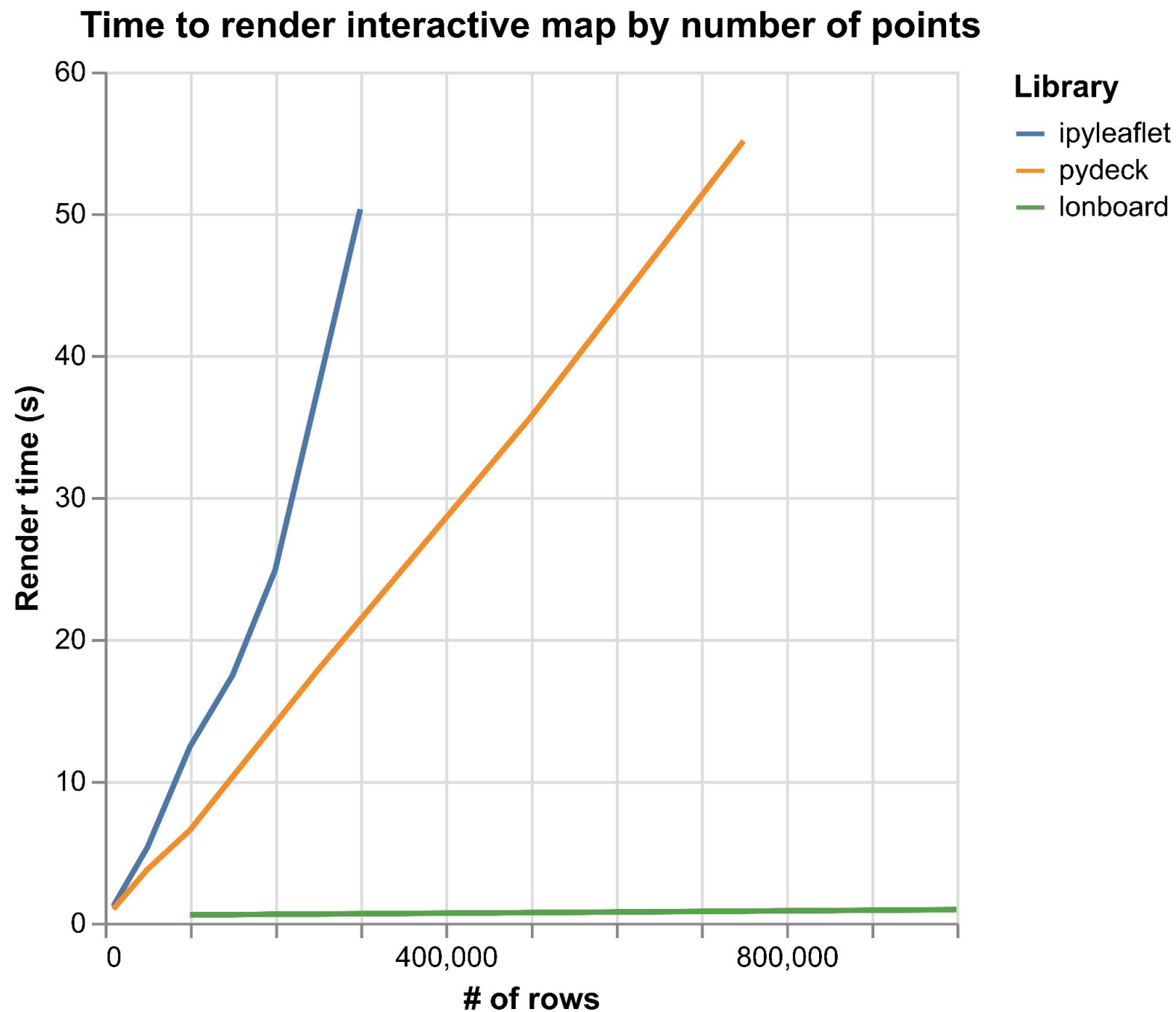
ipyleaflet?



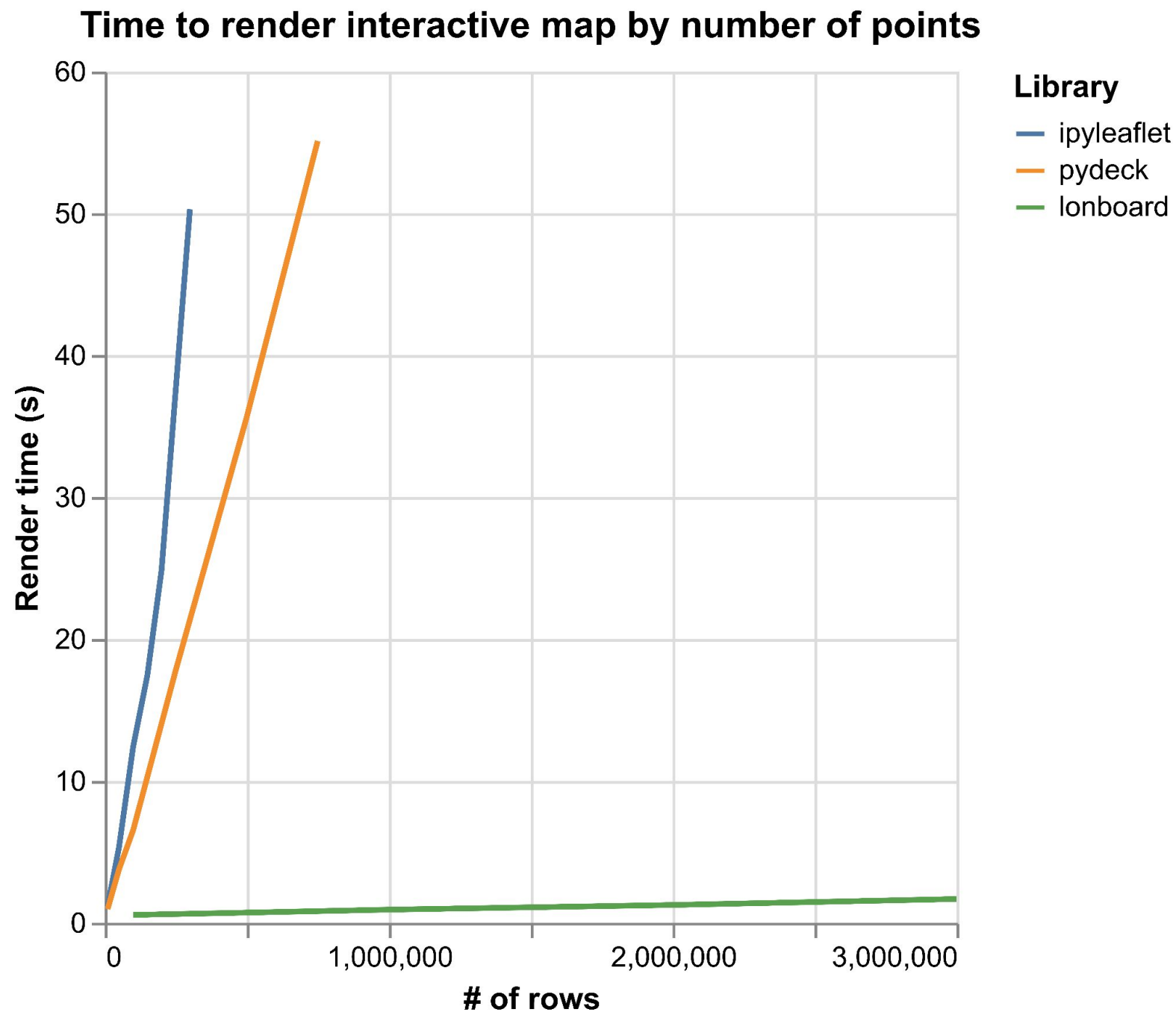
Time to render interactive map by number of points



lonboard!



lonboard!





Lonboard: 3 million points in 2.5 seconds

```
[ ]: layer = ScatterplotLayer.from_geopandas(  
    gdf[["geometry"]],  
    get_fill_color=apply_continuous_cmap(normalized_download_speed, BrBG_10, alpha=0.3),  
    get_radius = np.array(normalized_download_speed) * 200,  
    radius_units = "meters",  
    radius_min_pixels = 0.5  
)  
layer  
  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:  
[ ]:
```

Simple 0 4 lonboard | Idle Mode: Command Ln 8, Col 6 internet-speeds.ipynb 0



Interactive Data Exploration

The screenshot shows a Jupyter Notebook window titled "internet-speeds.ipynb". The interface includes a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar with icons for saving, adding, deleting, and running code. The notebook content consists of three code cells and two text blocks.

```
[ ]: layer = ScatterplotLayer.from_geopandas(gdf,
      get_fill_color=apply_continuous_cmap(normalized_download_speed, BrBG_10, alpha=0.7),
      get_radius=normalized_download_speed * 200,
      radius_units="meters",
      radius_min_pixels=0.5)
Map(layer)
```

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]
```

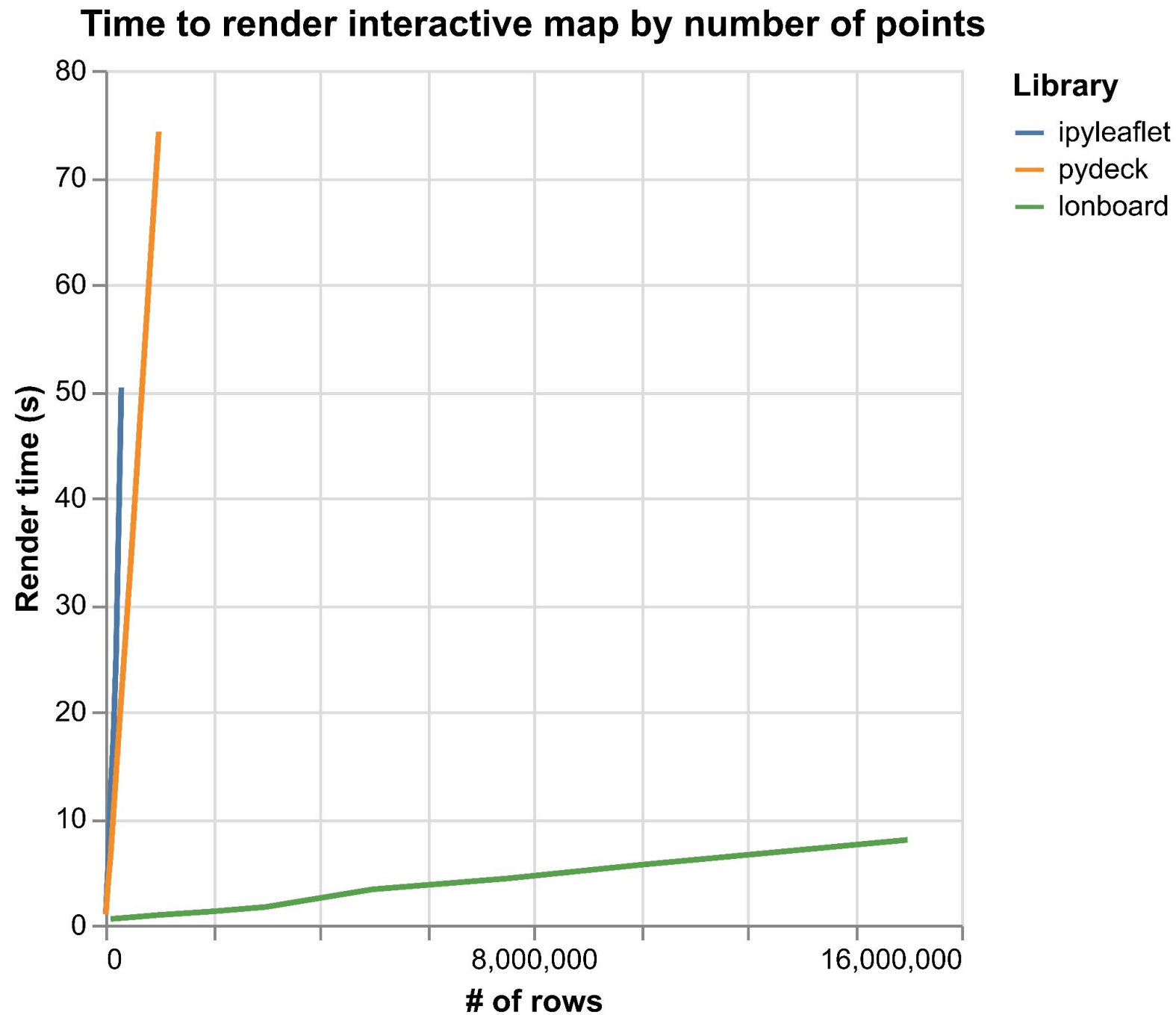
Blue is pretty, but the map would be more informative if we colored each point by a relevant characteristic. In this case, we have the download speed associated with each location, so let's use that!

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.

```
[8]: min_bound = 5000
max_bound = 50000
download_speed = gdf['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

lonboard!





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
```



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

High-level API

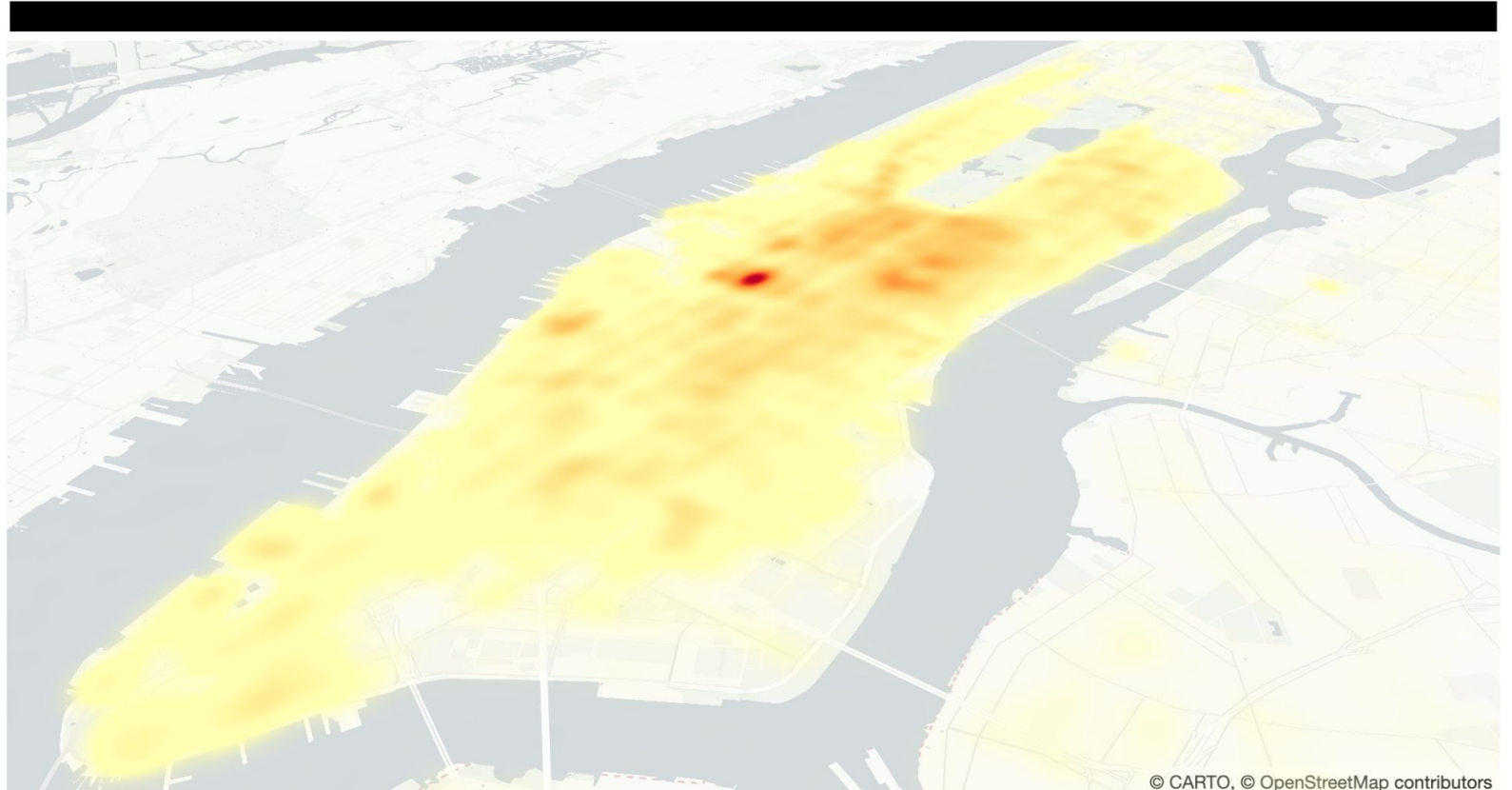
DuckDB Integration

- 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;
"""
```

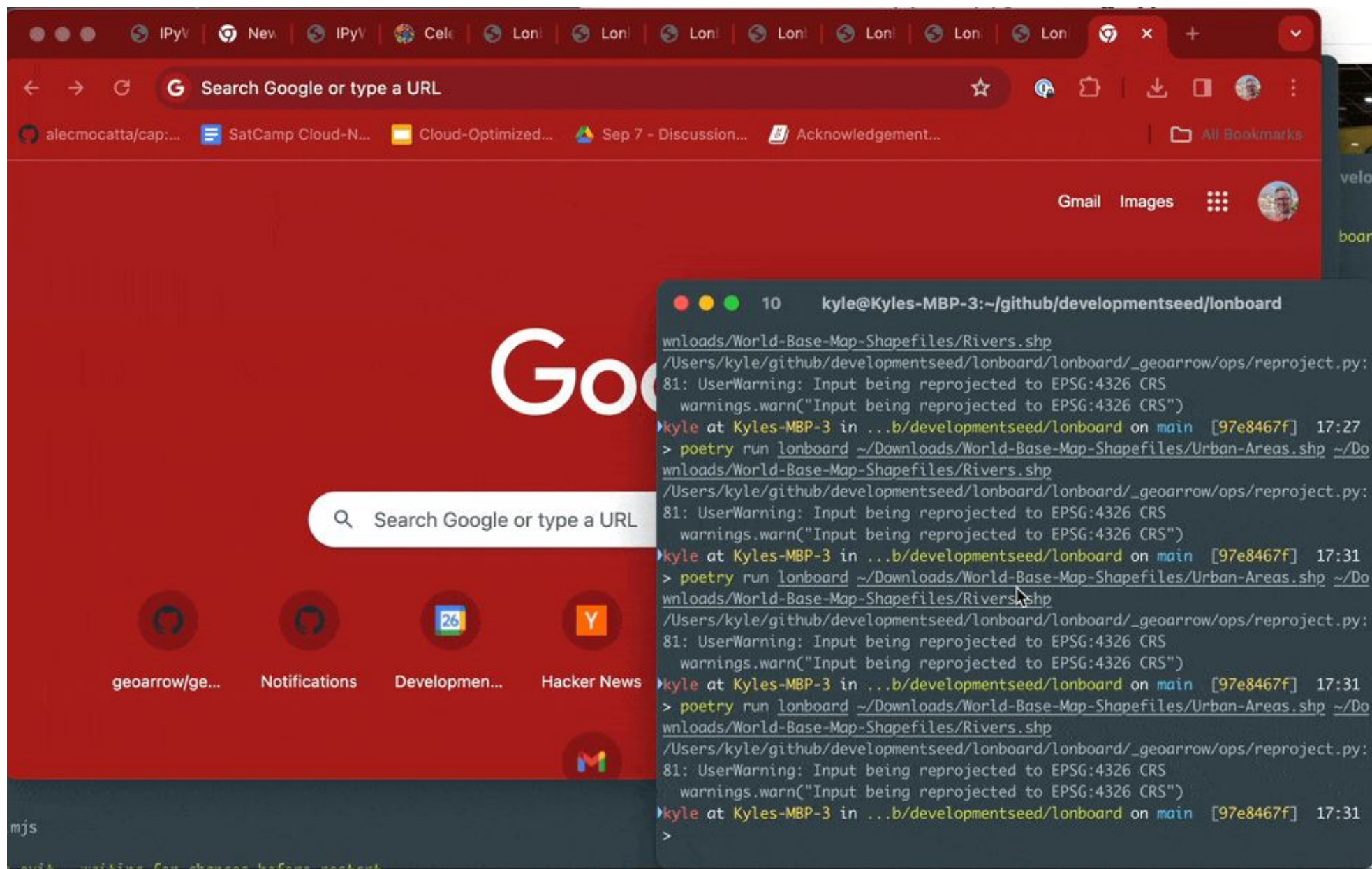
```
[6]: layer = HeatmapLayer.from_duckdb(sql, con)
m = Map(layer)
m
```

[6]:

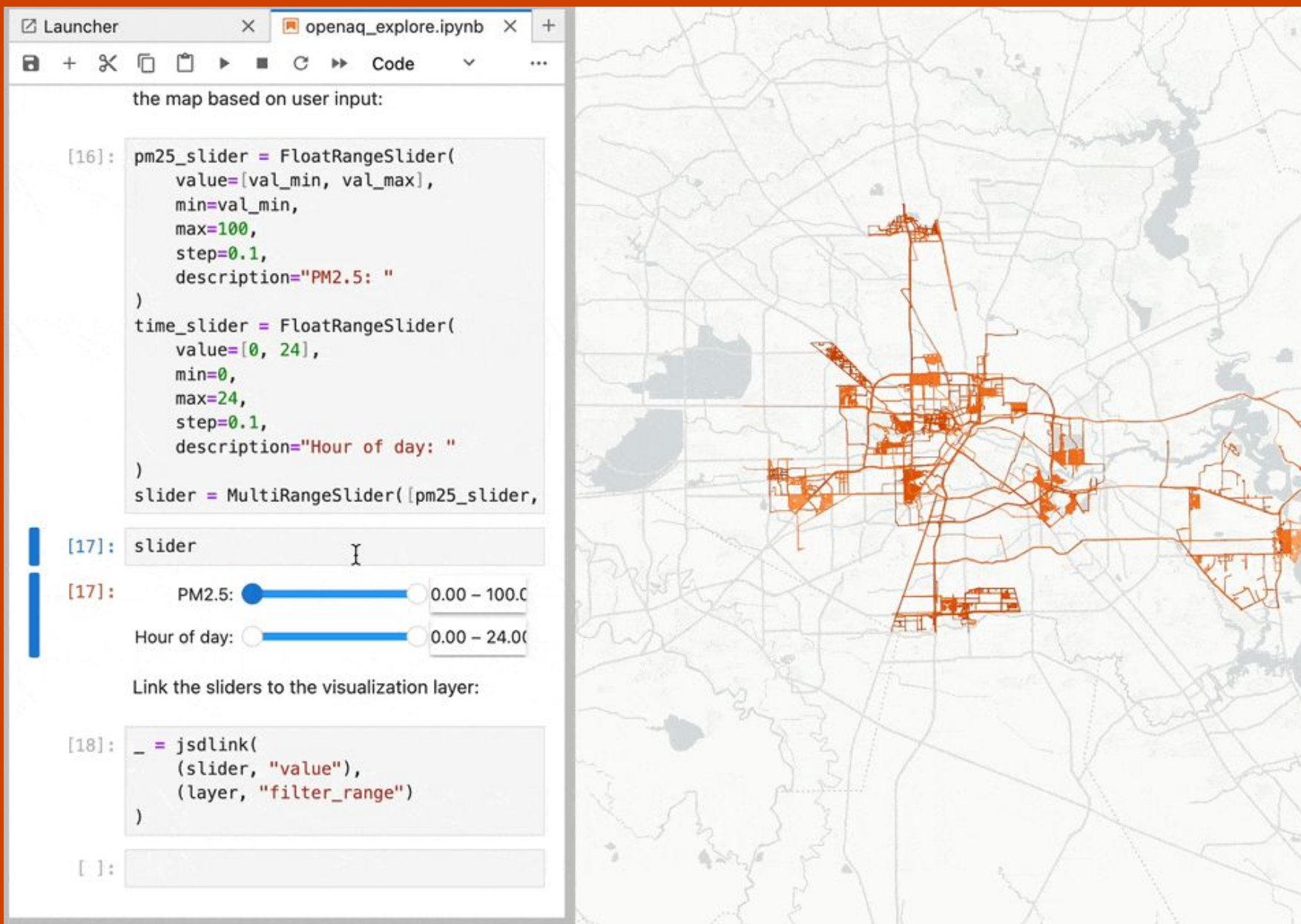


High-level API

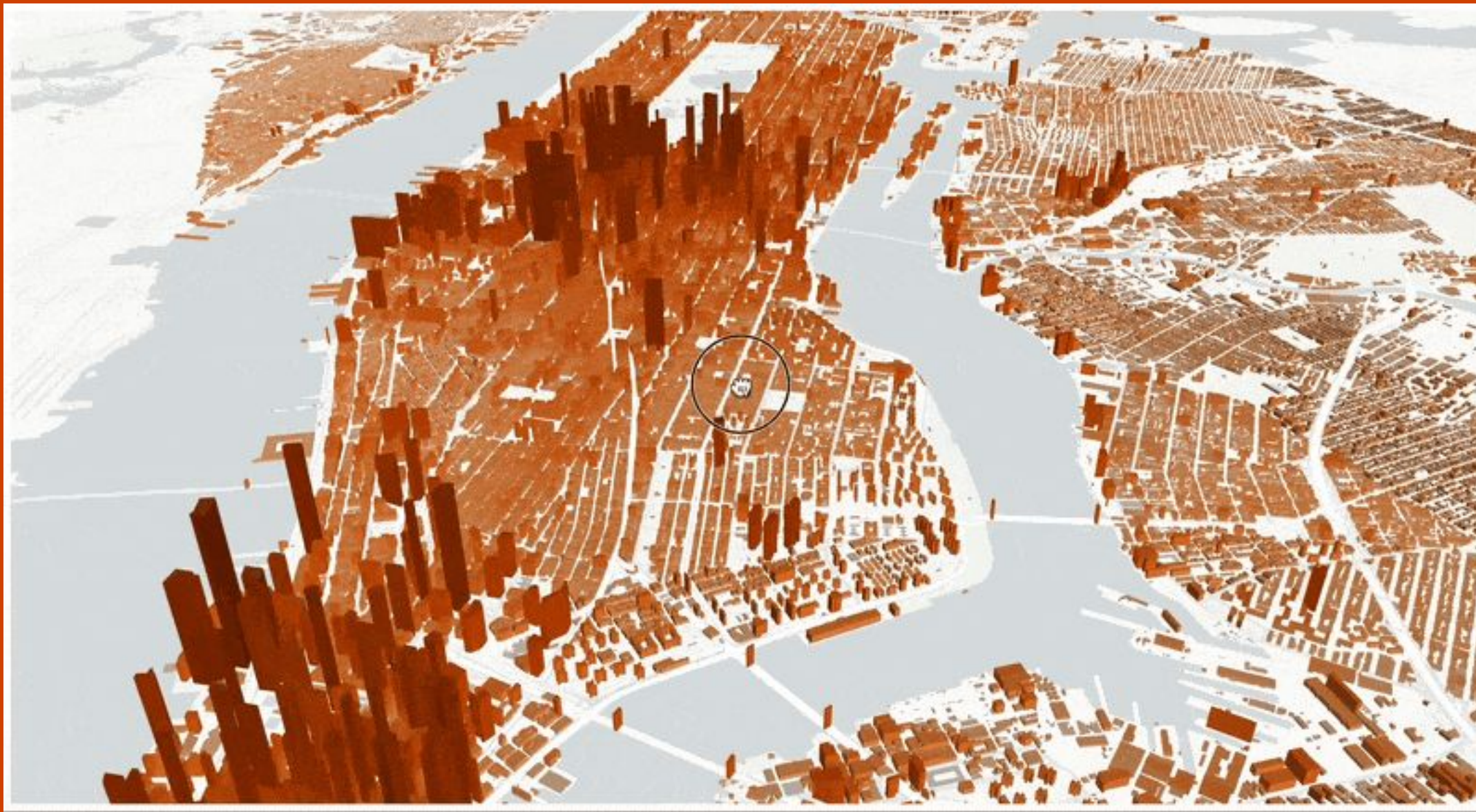
CLI Included



GPU Data Filtering



Buildings data





County-to-County Migration

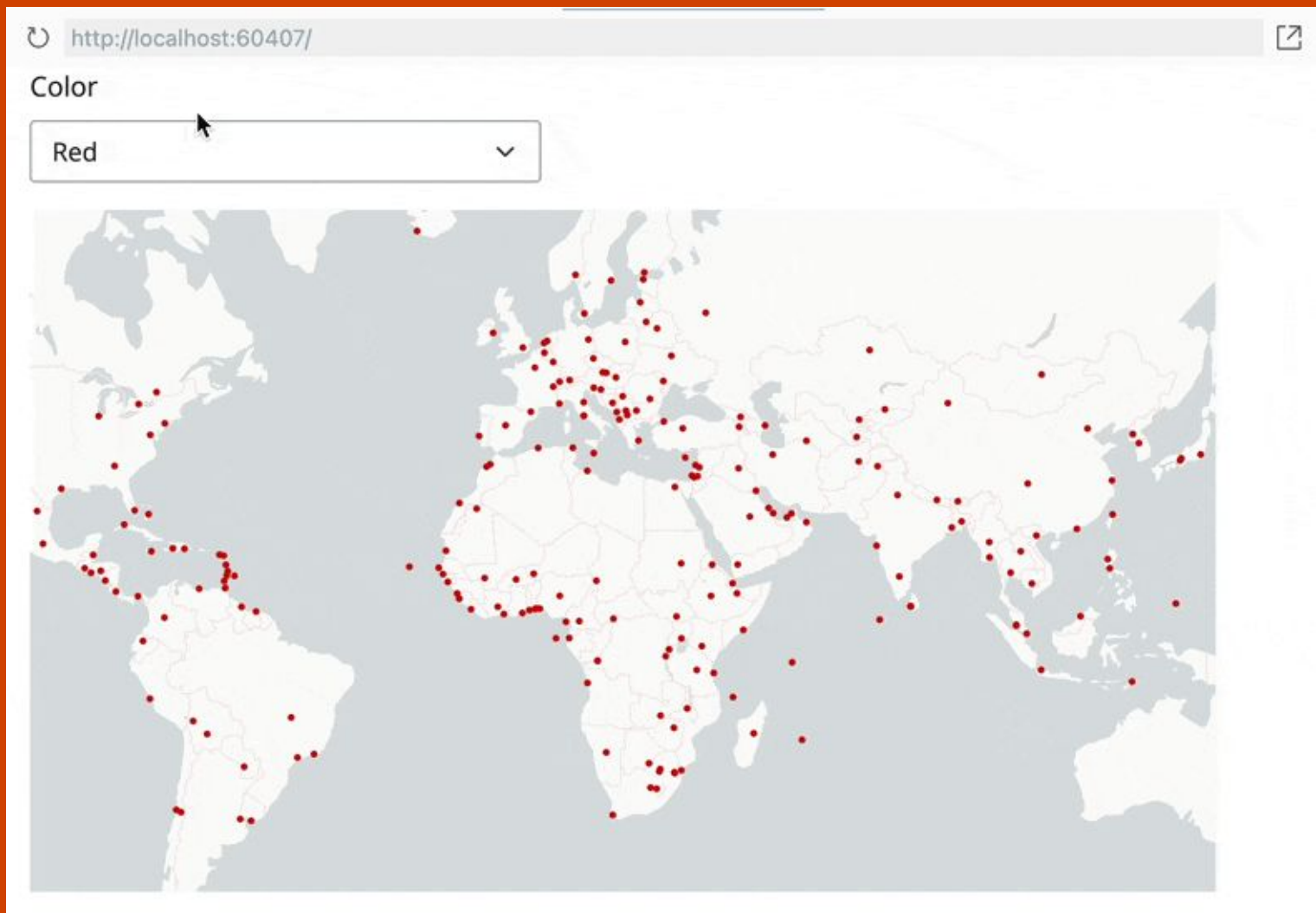
```
[12]: map_ = Map(layers=[source_layer, target_ring_layer, arc_layer], picking_radius=10)  
map_
```

```
[12]:
```



Dashboard integration

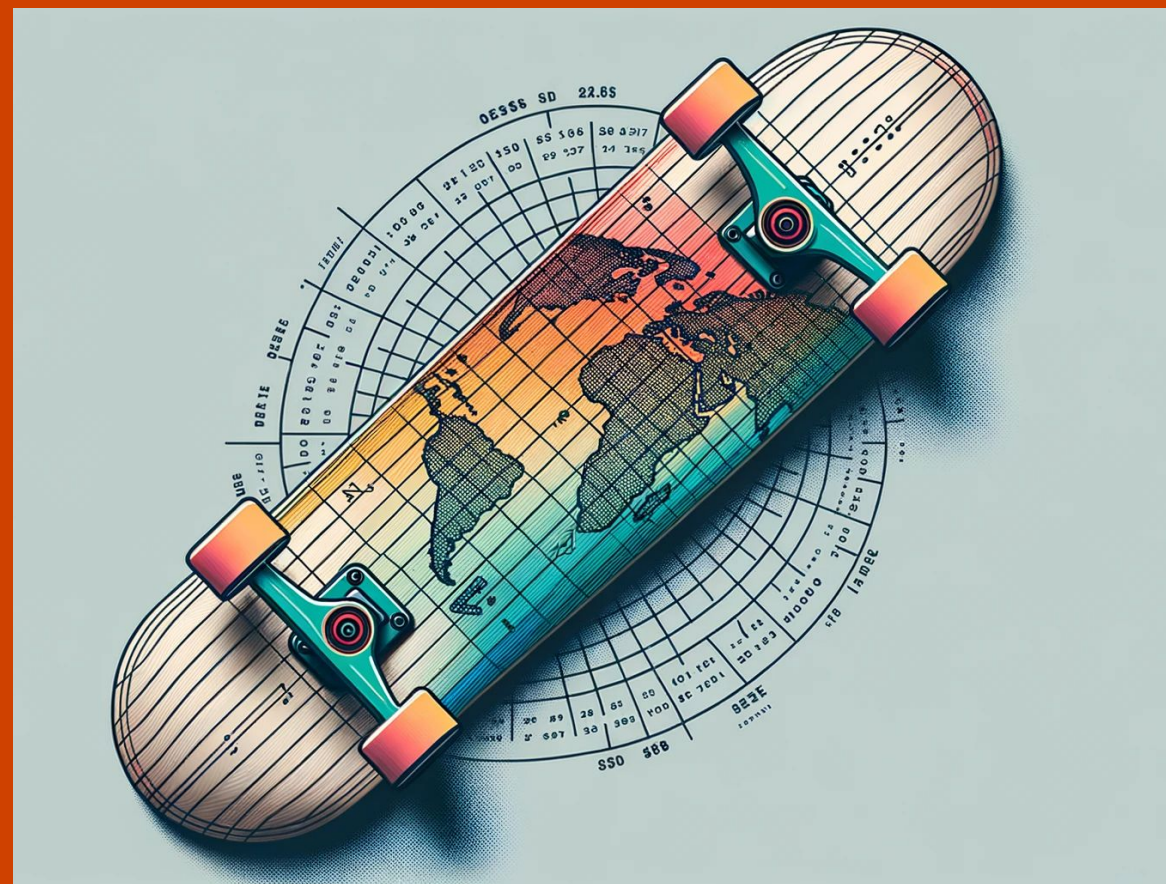
e.g. Shiny, Panel



What's in a name?

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

A *lonboard*.





Demos



How is it so fast?

Data rendering on the web

Three parts of vector web maps

Load data to the browser

Manage data in memory

Render to screen



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

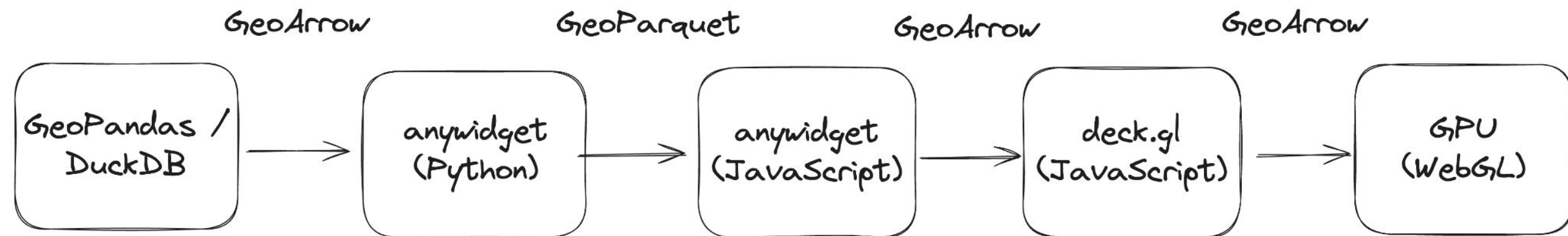
Render to screen

deck.gl

Fastest analytical map rendering on the web

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

End-to-end binary data pipeline



Lonboard Data Movement



Lonboard's *ulterior motive*



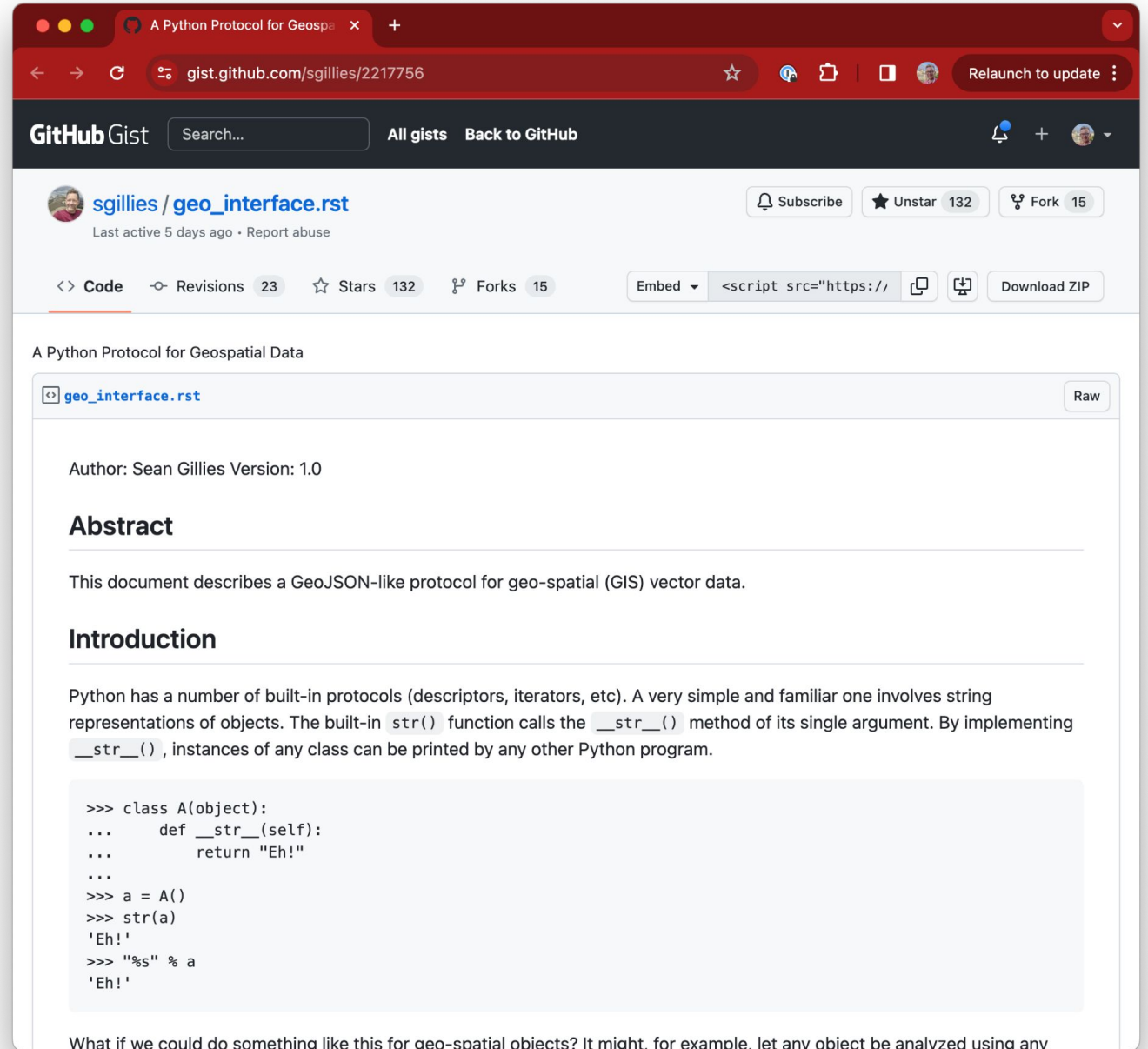
**Lonboard's *ulterior motive*:
Establish a GeoArrow ecosystem**



A fast data protocol for geospatial data

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



The screenshot shows a web browser displaying a GitHub Gist page for a file named `geo_interface.rst` by user `sgillies`. The page title is "A Python Protocol for Geospatial Data". The gist has 132 stars and 15 forks. The content of the file is displayed in a code editor view. It starts with "Author: Sean Gillies Version: 1.0", followed by an "Abstract" section stating it describes a GeoJSON-like protocol for geo-spatial (GIS) vector data. The "Introduction" section explains that Python has built-in protocols and that implementing `__str__()` allows objects to be printed. A code block shows a simple class `A` with a `__str__` method that returns "Eh!".

Author: Sean Gillies Version: 1.0

Abstract

This document describes a GeoJSON-like protocol for geo-spatial (GIS) vector data.

Introduction

Python has a number of built-in protocols (descriptors, iterators, etc). A very simple and familiar one involves string representations of objects. The built-in `str()` function calls the `__str__()` method of its single argument. By implementing `__str__()`, instances of any class can be printed by any other Python program.

```
>>> class A(object):
...     def __str__(self):
...         return "Eh!"
...
>>> a = A()
>>> str(a)
'Eh!'
>>> "%s" % a
'Eh!'
```

What if we could do something like this for geo-spatial objects? It might, for example, let any object be analyzed using any

Data protocols enable innovation

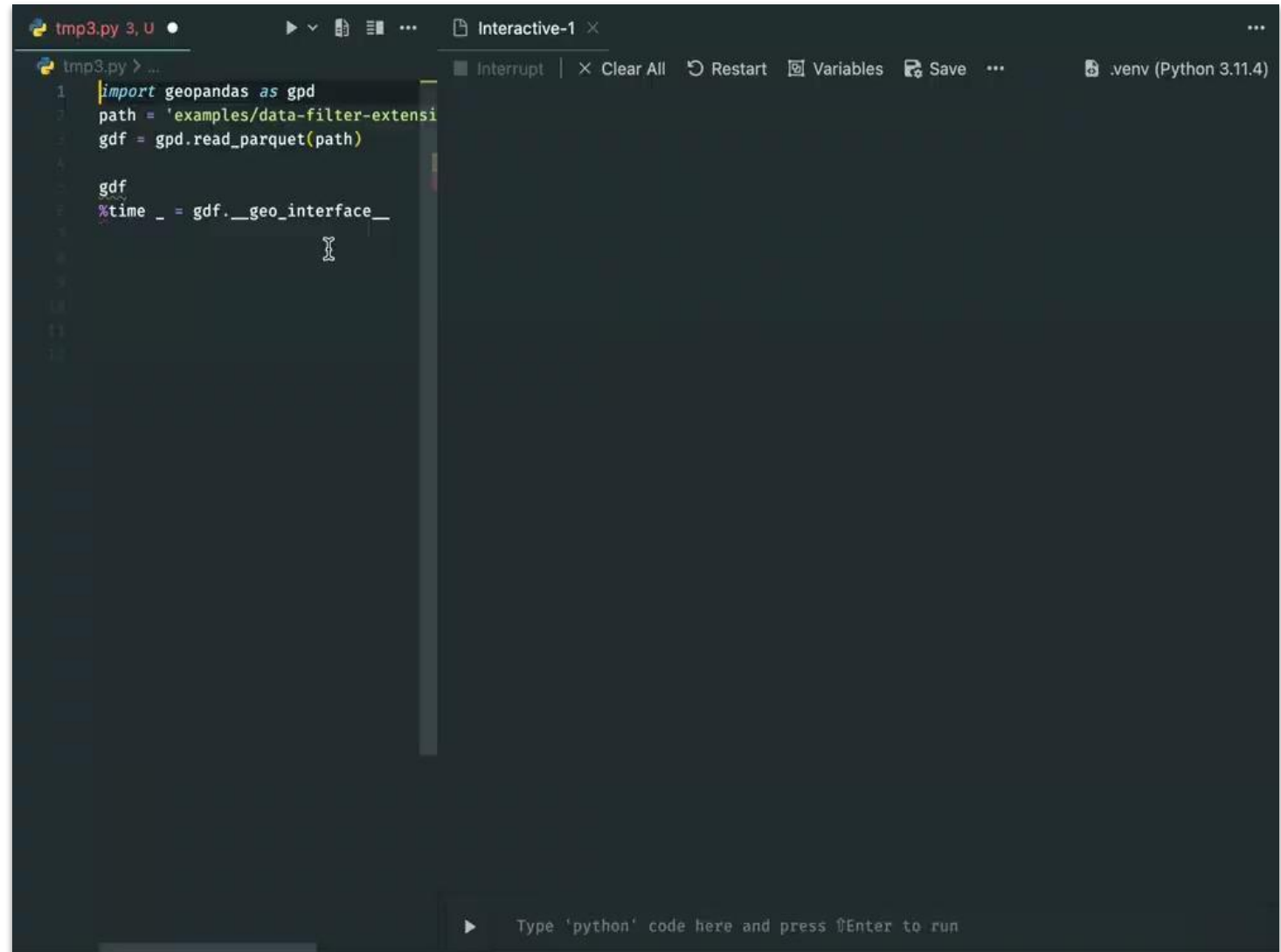
- A **`__geo_interface__`** attribute gives you **GeoJSON**.
- *Many* more implementations. (unknown number!)
- All of these libraries *just work* together.
- A new library can connect to a full ecosystem by implementing a single interface

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

Selected libraries implementing `__geo_interface__`

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



```
tmp3.py 3, U • Interactive-1 ×
tmp3.py > ...
1 import geopandas as gpd
2 path = 'examples/data-filter-extensi
3 gdf = gpd.read_parquet(path)
4
5 gdf
6 %time _ = gdf.__geo_interface__
7
8
9
10
11
12
```

Interrupt | × Clear All | ↺ Restart | 📄 Variables | 💾 Save | ... | .venv (Python 3.11.4)

▶ Type 'python' code here and press ⏎Enter to run

1 minute overhead on 3 million point dataset



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)

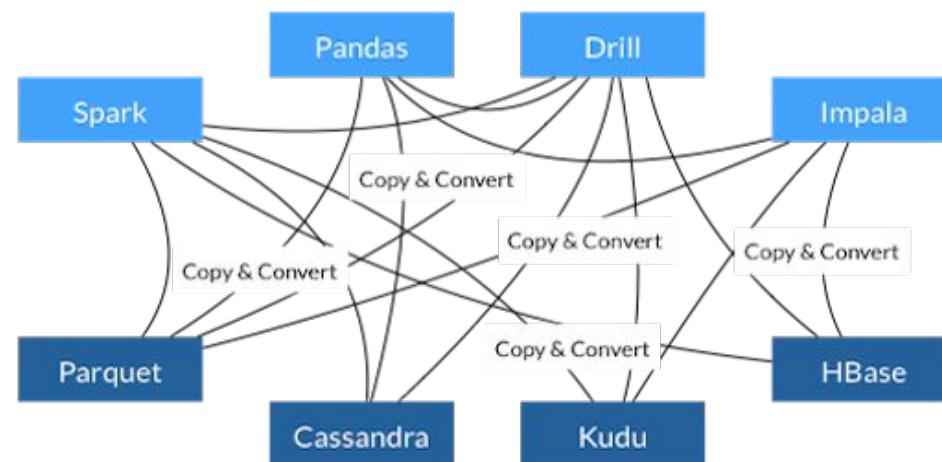
<https://jakevdp.github.io/blog/2014/05/05/introduction-to-the-python-buffer-protocol/>

Binary data protocol for tabular data

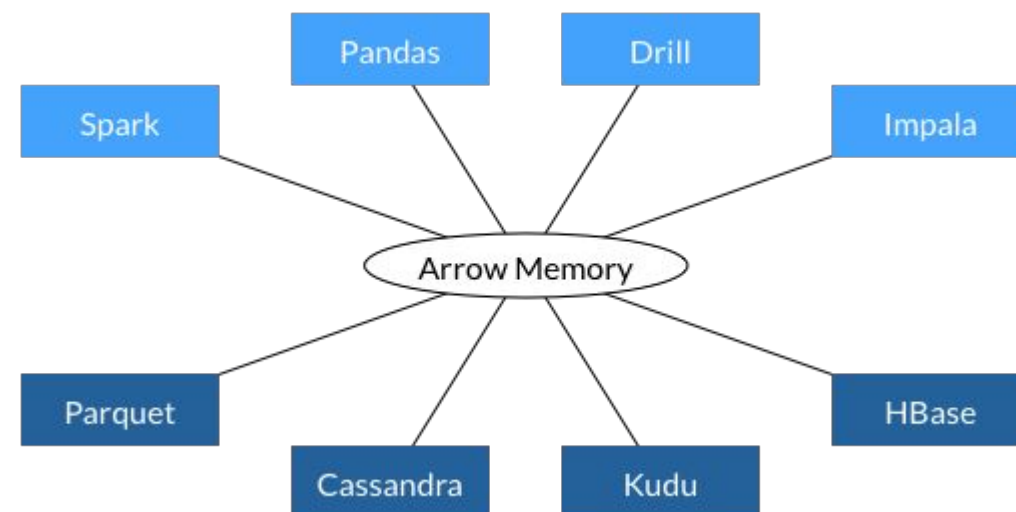
Apache Arrow (2015)

- **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



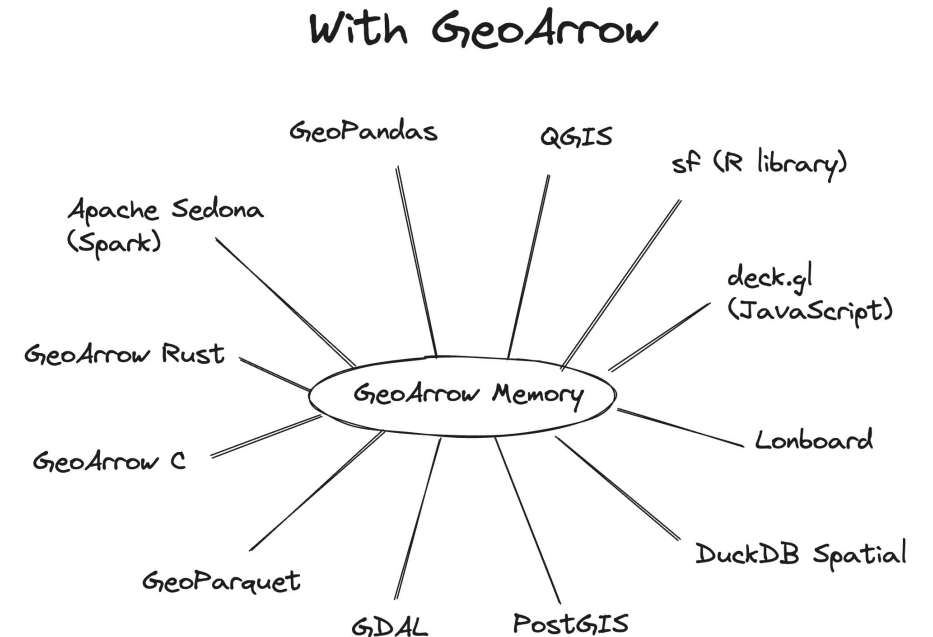
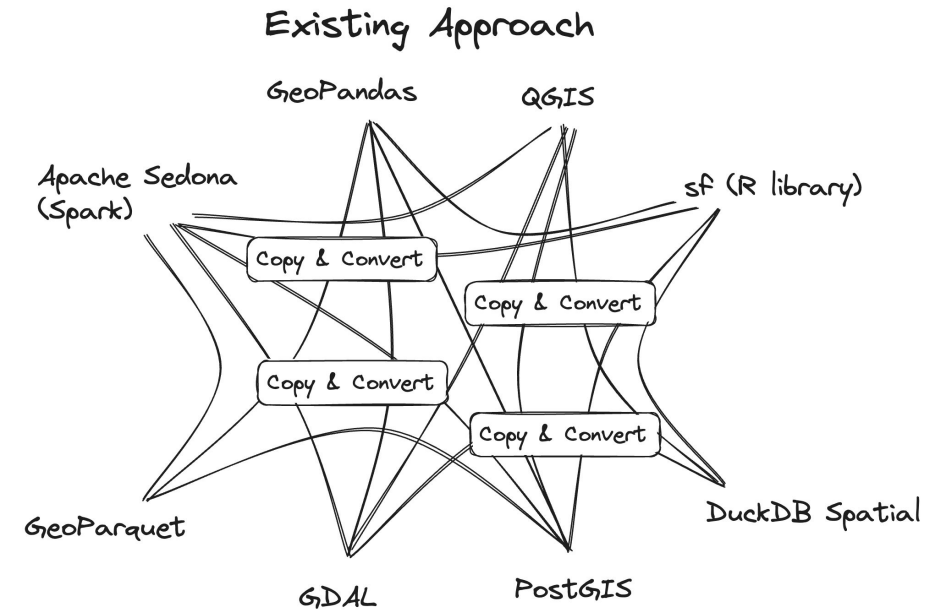
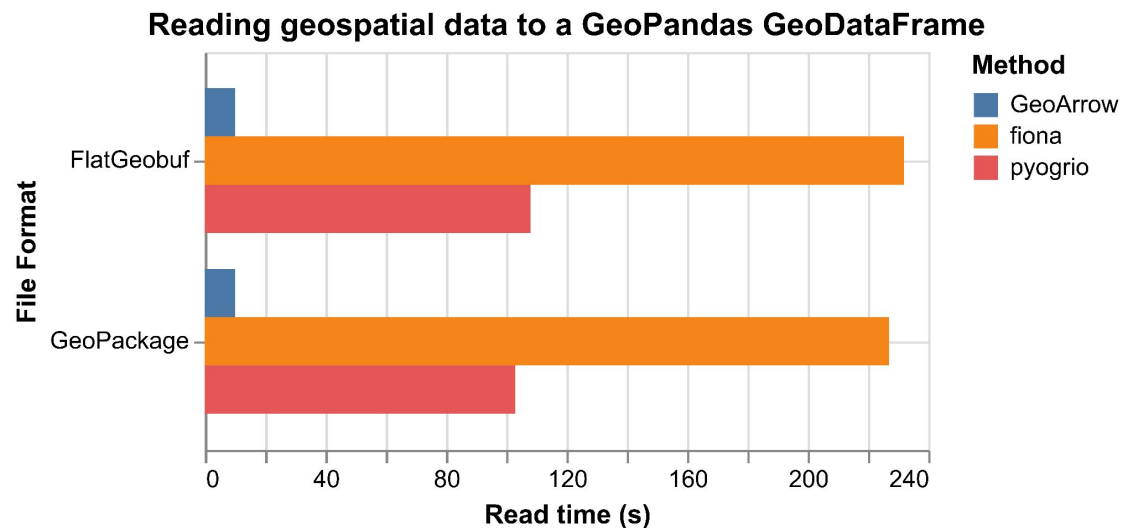
With Arrow



Enabling zero-copy data sharing for geospatial

GeoArrow (2020)

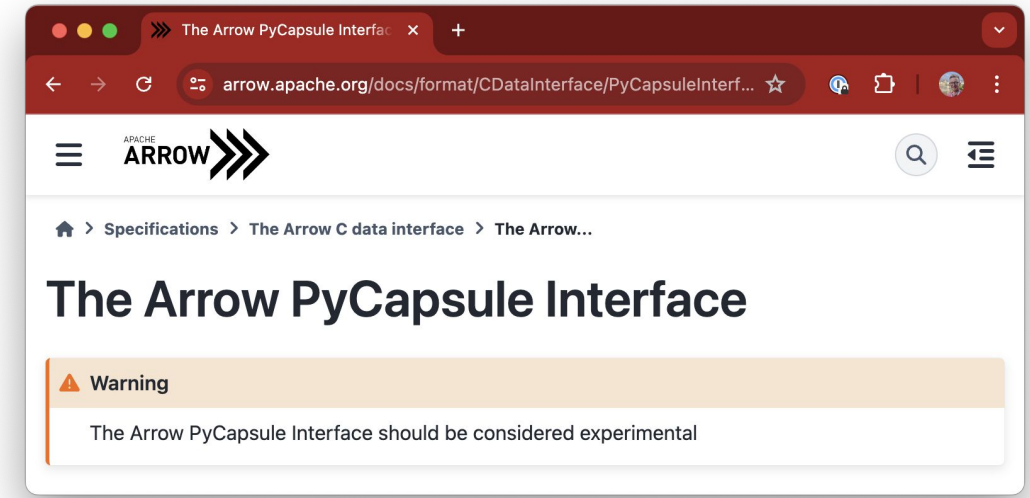
- **Problem:** Expensive to copy & convert *geospatial* tabular data between libraries and languages
- **Binary data protocol for geospatial tabular data.**
- Language-independent, zero-copy data sharing



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

<i>fastexcel</i>	<i>daft</i>	<i>lance</i>
<i>datafusion</i>	<i>duckdb</i>	<i>lightgbm</i>

Libraries with open issue for PyCapsule Interface

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
```

	id	name	nationality	sex	date_of_birth	height
	int64	utf8	utf8	utf8	date32[day]	float64
	0	22 categories	207 categories	2 categories		01.2
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	

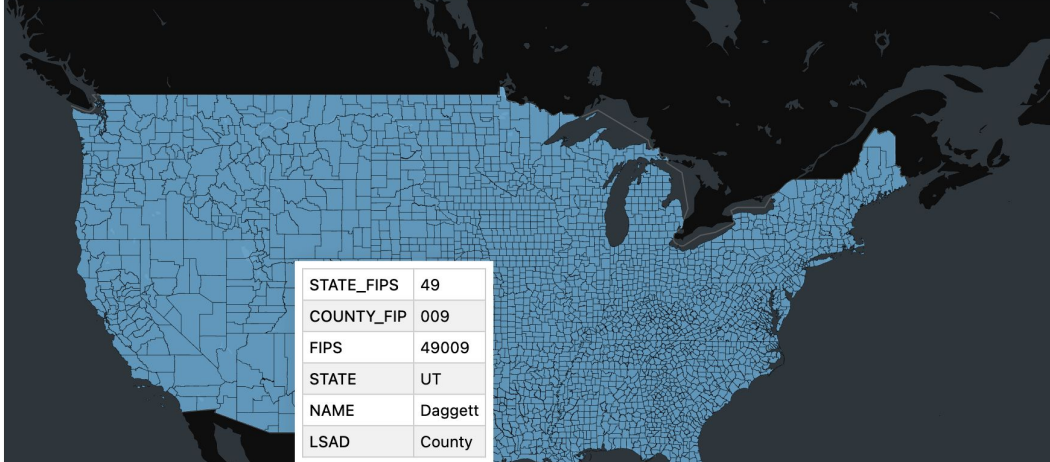
quak: a scalable data profiler

```
[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.



STATE_FIPS	49
COUNTY_FIP	009
FIPS	49009
STATE	UT
NAME	Daggett
LSAD	County

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

- 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

Interactive-1 — developmentseed

Interactive-1 × tmp3.py 3, U

Interrupt | Clear All | Restart | Variables | Save | Python 3.11.4

✓ pyarrow_table ...

...
pyarrow.Table
avg_d_kbps: int64
avg_u_kbps: int64
avg_lat_ms: int64
devices: int64
geometry: fixed_size_list<xy: double not null>[2]
 child 0, xy: double not null

avg_d_kbps: [[5983,3748,3364,2381,3047, ... ,46246,70471,50113,15629,12653]
avg_u_kbps: [[7886,5841,6200,2328,5356, ... ,13487,8511,27899,11168,2384],[
avg_lat_ms: [[68,78,78,86,75, ... ,29,30,22,44,21],[26,31,29,52,52, ... ,27,6
devices: [[1,2,2,1,1, ... ,3,1,1,5,1],[1,4,2,1,2, ... ,2,2,3,1,1], ... ,[1,1,1,
geometry: [[[-160.0186157226565,70.63721610566131],[-160.04058837890648,7

✓ %timeit rust_table2 = GeoTable.from_arrow(pyarrow_table) ...

... 290 µs ± 5.43 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

Type 'python' code here and press Enter to run

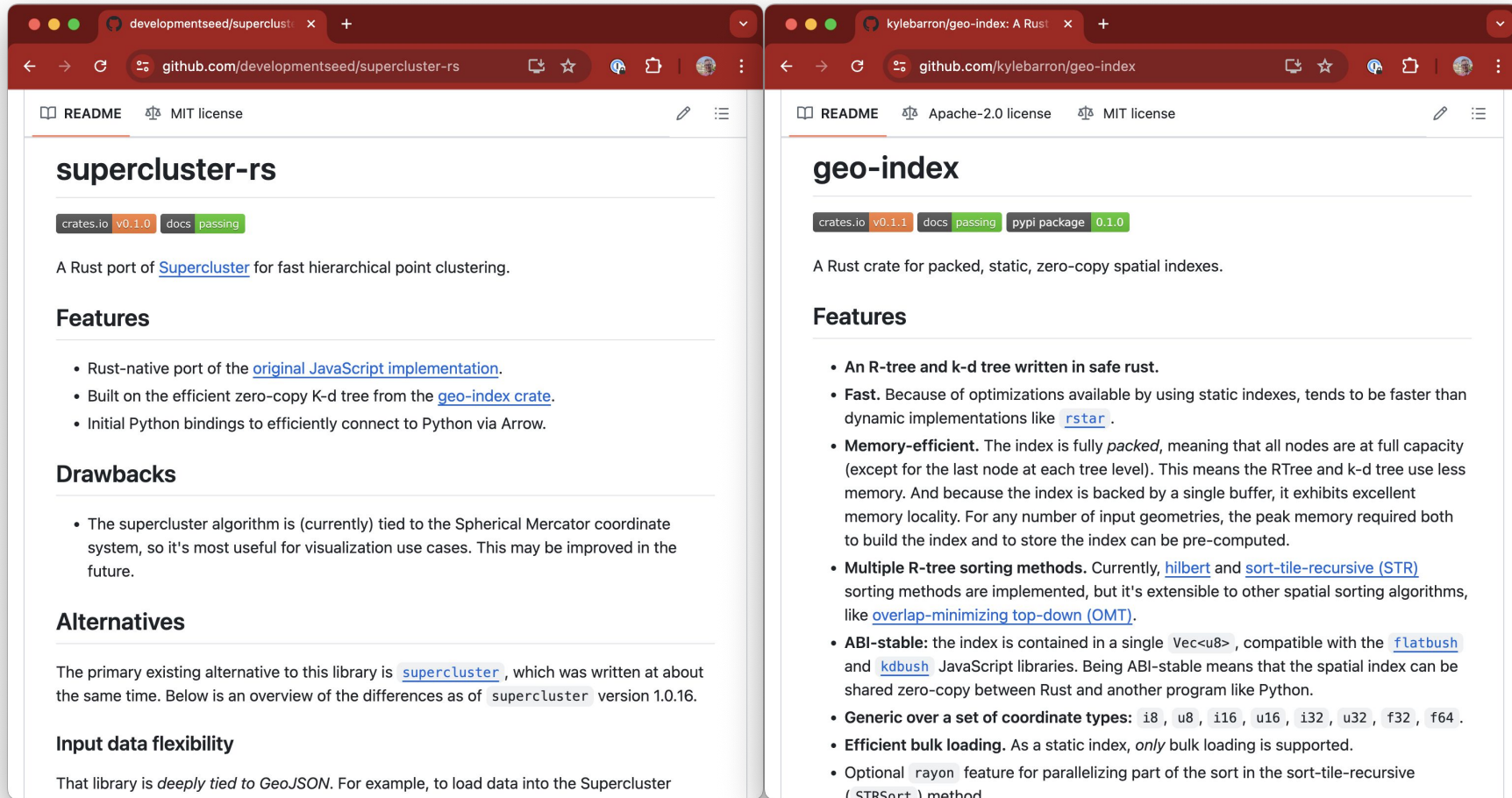
kyle/multi-channel-error* 5 1 0 rust-analyzer Pull Request #367



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



The image shows two browser windows side-by-side, displaying the GitHub pages for two Rust crates: **supercluster-rs** and **geo-index**.

Left Window: developmentseed/supercluster-rs

- URL: `github.com/developmentseed/supercluster-rs`
- MIT license
- Crates.io: `v0.1.0`, docs: `passing`
- Description: A Rust port of [Supercluster](#) for fast hierarchical point clustering.
- Features**
 - Rust-native port of the [original JavaScript implementation](#).
 - Built on the efficient zero-copy K-d tree from the [geo-index crate](#).
 - Initial Python bindings to efficiently connect to Python via Arrow.
- Drawbacks**
 - The supercluster algorithm is (currently) tied to the Spherical Mercator coordinate system, so it's most useful for visualization use cases. This may be improved in the future.
- Alternatives**

The primary existing alternative to this library is [supercluster](#), which was written at about the same time. Below is an overview of the differences as of `supercluster` version 1.0.16.
- Input data flexibility**

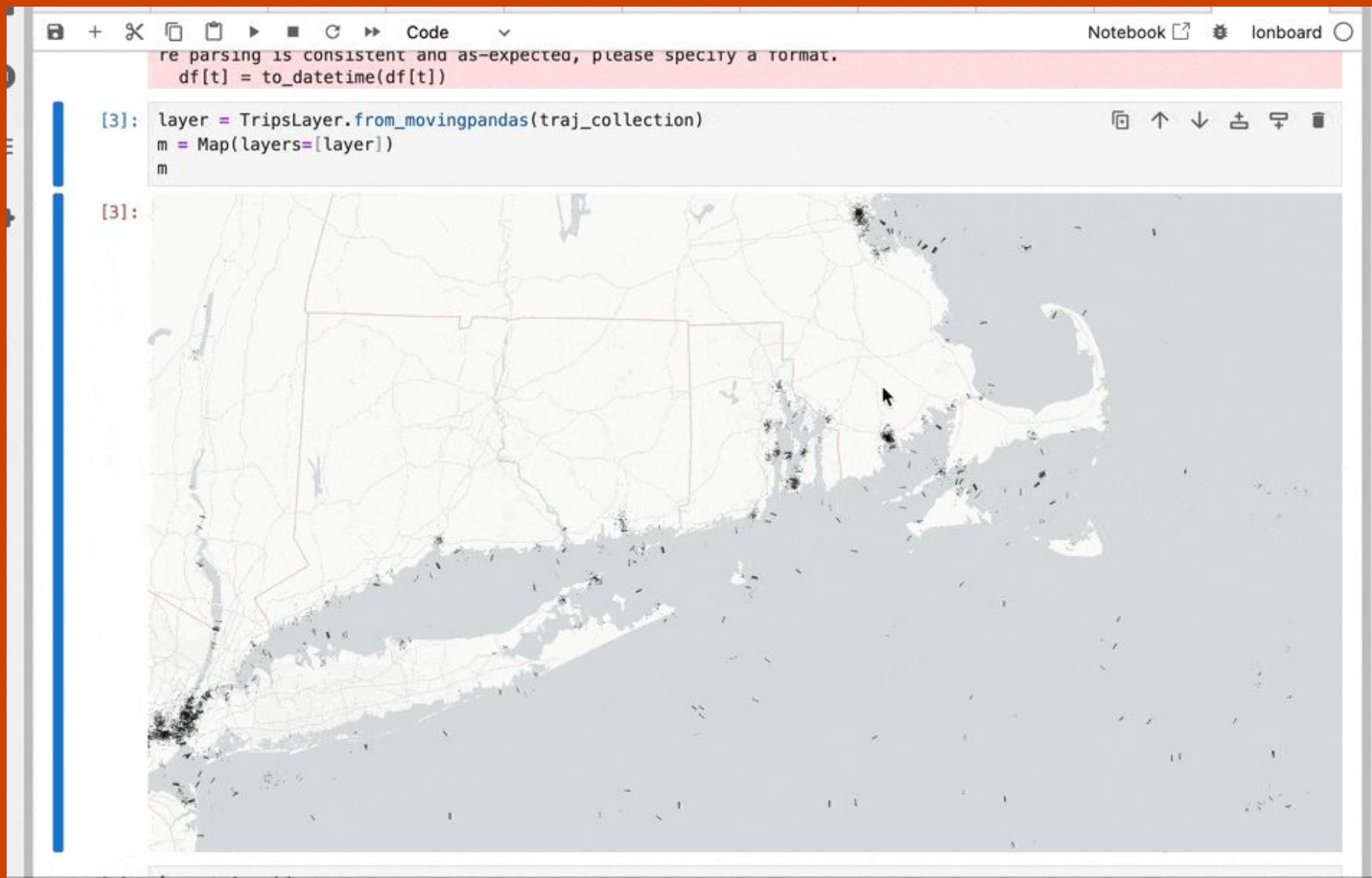
That library is *deeply tied to GeoJSON*. For example, to load data into the Supercluster

Right Window: kylebarron/geo-index: A Rust

- URL: `github.com/kylebarron/geo-index`
- Apache-2.0 license, MIT license
- Crates.io: `v0.1.1`, docs: `passing`, pypi package: `0.1.0`
- Description: A Rust crate for packed, static, zero-copy spatial indexes.
- Features**
 - An R-tree and k-d tree written in safe rust.
 - Fast.** Because of optimizations available by using static indexes, tends to be faster than dynamic implementations like [rstar](#).
 - Memory-efficient.** The index is fully *packed*, meaning that all nodes are at full capacity (except for the last node at each tree level). This means the RTree and k-d tree use less memory. And because the index is backed by a single buffer, it exhibits excellent memory locality. For any number of input geometries, the peak memory required both to build the index and to store the index can be pre-computed.
 - Multiple R-tree sorting methods.** Currently, [hilbert](#) and [sort-tilde-recursive \(STR\)](#) sorting methods are implemented, but it's extensible to other spatial sorting algorithms, like [overlap-minimizing top-down \(OMT\)](#).
 - ABI-stable:** the index is contained in a single `Vec<u8>`, compatible with the [flatbush](#) and [kdbush](#) JavaScript libraries. Being ABI-stable means that the spatial index can be shared zero-copy between Rust and another program like Python.
 - Generic over a set of coordinate types:** `i8`, `u8`, `i16`, `u16`, `i32`, `u32`, `f32`, `f64`.
 - Efficient bulk loading.** As a static index, *only* bulk loading is supported.
 - Optional `rayon` feature for parallelizing part of the sort in the sort-tilde-recursive (`STRSort`) method.

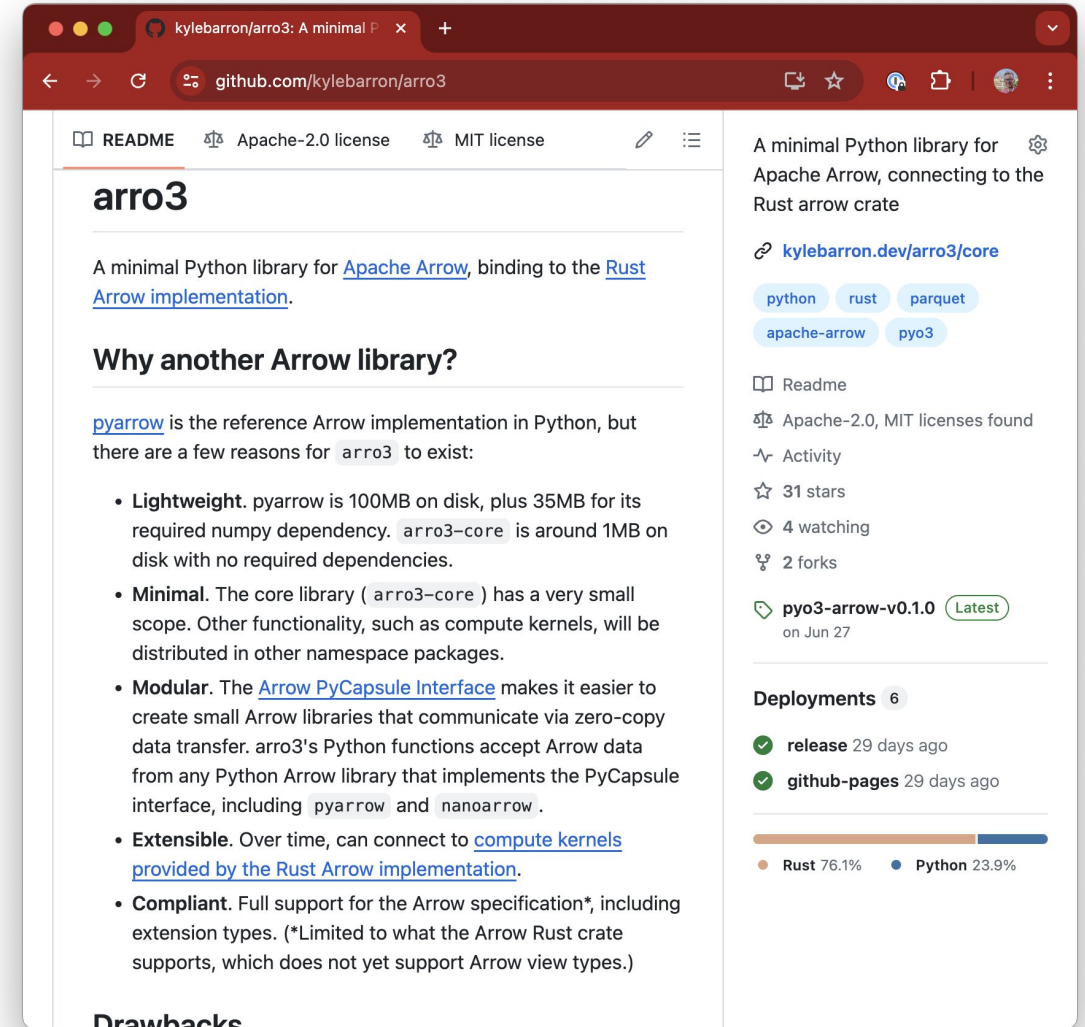


Future Work: Trajectory data



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.



The screenshot shows the GitHub repository page for `kylebarron/arro3`. The repository is described as "A minimal Python library for Apache Arrow, connecting to the Rust arrow crate". It includes a README, Apache-2.0 license, and MIT license. The repository has 31 stars, 4 watchers, and 2 forks. The latest release is `pyo3-arrow-v0.1.0` on Jun 27. The repository also has 6 deployments, including `release` and `github-pages`, both 29 days ago. A progress bar shows the distribution of the library: Rust 76.1% and Python 23.9%.

arro3

A minimal Python library for [Apache Arrow](#), binding to the [Rust Arrow implementation](#).

Why another Arrow library?

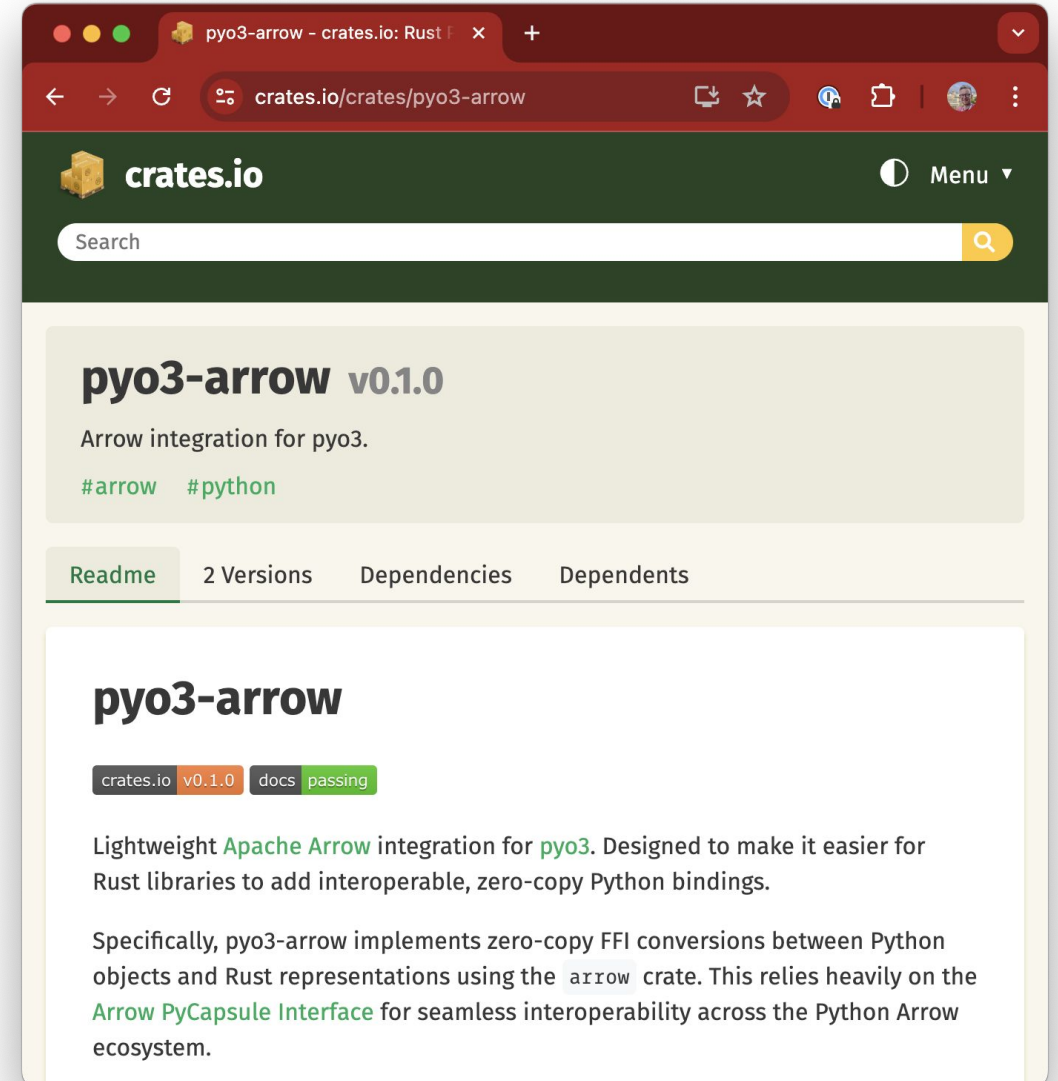
[pyarrow](#) is the reference Arrow implementation in Python, but there are a few reasons for `arro3` to exist:

- **Lightweight.** `pyarrow` is 100MB on disk, plus 35MB for its required numpy dependency. `arro3-core` is around 1MB on disk with no required dependencies.
- **Minimal.** The core library (`arro3-core`) has a very small scope. Other functionality, such as compute kernels, will be distributed in other namespace packages.
- **Modular.** The [Arrow PyCapsule Interface](#) makes it easier to create small Arrow libraries that communicate via zero-copy data transfer. `arro3`'s Python functions accept Arrow data from any Python Arrow library that implements the PyCapsule interface, including `pyarrow` and `nanoarrow`.
- **Extensible.** Over time, can connect to [compute kernels provided by the Rust Arrow implementation](#).
- **Compliant.** Full support for the Arrow specification*, including extension types. (*Limited to what the Arrow Rust crate supports, which does not yet support Arrow view types.)

Drawbacks

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`

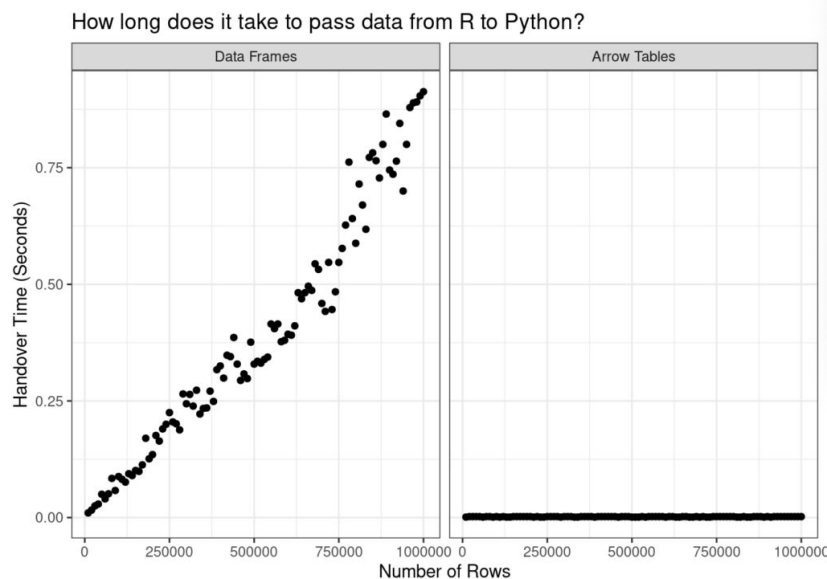


An exercise for the reader

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](#))



geoarrow/geoarrow-r: Extens...

github.com/geoarrow/geoarrow-r

README Apache-2.0 license License

geoarrow

codecov 95%

The goal of geoarrow is to leverage the features of the [arrow](#) package and larger [Apache Arrow](#) ecosystem for geospatial data. The geoarrow package provides an R implementation of the [GeoParquet](#) file format of and the draft [geoarrow data specification](#), defining extension array types for vector geospatial data.

Installation

You can install the released version of geoarrow from [CRAN](#) with:

```
install.packages("geoarrow")
```

You can install the development version of geoarrow from [GitHub](#) with:

```
# install.packages("pak")
pak::pak("geoarrow/geoarrow-r")
```

Example

The geoarrow package implements conversions to/from various geospatial types (e.g., sf, sfc, s2, wk) with various Arrow representations (e.g., arrow, nanoarrow). The most useful conversions are between the [arrow](#) and [sf](#) packages, which in most cases allow sf objects to be passed to [arrow](#) functions directly after `library(geoarrow)` or `requireNamespace("geoarrow")` has been called.

library(geoarrow)

Extension types for geospatial data for use with 'Arrow'

[geoarrow.org/geoarrow-r/](#)

Readme

Apache-2.0, Unknown licenses found

Activity

Custom properties

147 stars

6 watching

6 forks

geoarrow 0.2.1 Latest
on Jun 14

Contributors 3

[paleolimbot](#) Dewey Dunnington

[mikemahoney218](#) Michael Mahoney

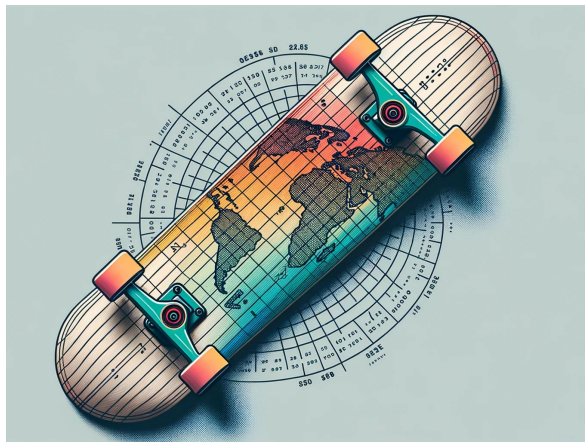
[olivroy](#)

Deployments 62

[github-pages](#) last month

R 74.1% C 15.0% C++ 10.4% Shell 0.5%

Report repository



Learn More

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

