

Keeping Spatial Scripting Sane

with Samaki and Raku



by Brian Duggan

 bduggan

FOSDEM 2026

- Brian Duggan
- Logistics Engineer at Instacart
- Dispatch/Geospatial Team
- Contributor to Raku ecosystem

- Motivation
- Samaki
- Examples
- Extending
- Raku

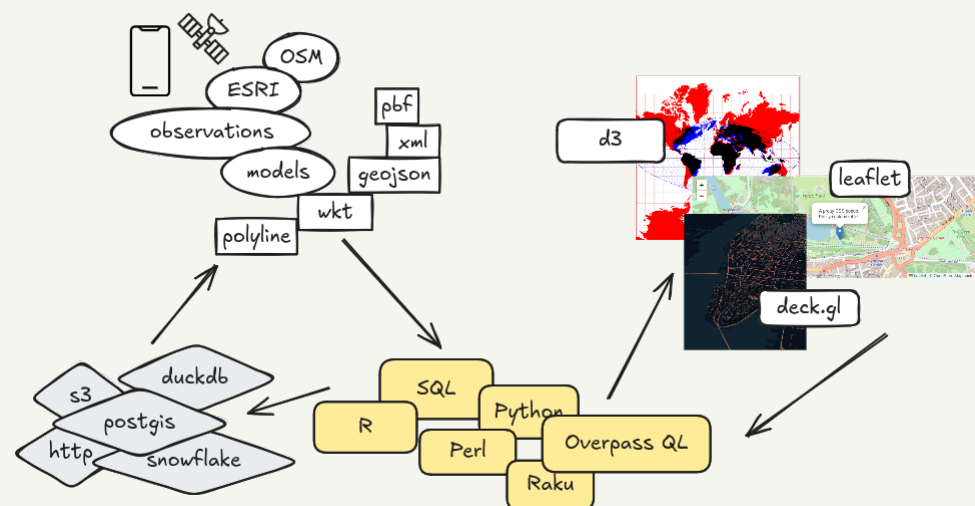
Spatial scripting!

goals

- Improve on UI-oriented workflows
- using on local text files
- that works with multiple languages
- and multiple tools
- and is easy to extend

challenges

- Visualization is part of coding
- but UIs can slow things down
- Data exploration is critical
- but volume of data can be a challenge
- Different languages for different things
- can be tricky to glue together



- Stitching together A Multitude of Kinds of Items
- Swahili for fish

design goals

- simple text file format for a script
- multiple languages at once
- easy to extend (plugins and plugouts)
- console integration; minimal UI
- some aspects of REPLs, Notebooks



M. C. Escher, Fish, Vignette, 1956

Basic concept

```
-- python
import this

-- duckdb
select st_buffer(...) as ring

-- R
plot(...)

-- llm
What latitude is the northernmost point in Belgium?
```

Terminology

- cells -- a sequence of "cells" of different types.
- plugins -- each cell type is handled by a "plugin".
- plugouts -- output files are handled by "plugouts".

Basic concept

```
-- python:location.json
import json
print(json.dumps({...})) # find lat/lon

-- duckdb:rows.csv
select st_buffer(...) as ring

-- R:plot.png
plot(...)

-- llm:response.txt
What latitude is the northernmost point in Belgium?
```

Cells can also have:

- a name ("belgium")
- an output format ("json"), determines which plugouts handle it

Share data between cells by interpolating into code

```
-- python:location.json
import json
print(json.dumps(..) ) # find lat/lon

-- duckdb:rows.csv
select st_buffer( st_makepoint( < cell("location").res<lon lat>.join(',') > ), 1) as ring

-- R:plot.png
plot(...)

-- llm:response.txt
What latitude is the northernmost point in Belgium?
```

Angle brackets contain Raku expressions.

Expressions can get output or content from other cells, after the cell is executed.

Share data between cells by interpolating into code

```
--  
my $country = 'belgium';  
  
-- python:location.json  
import json  
print(json.dumps(..) ) # find lat/lon  
  
-- duckdb:rows.csv  
select st_buffer( st_makepoint( < cell("location").res<lon lat>.join(',') > ), 1) as ring  
  
-- R:plot.png  
plot( ... )  
  
-- llm:response.txt  
What latitude is the northernmost point in < $country.uc > ?
```

You can also have Raku code outside of the cells.

Execution context is shared.

Summary

A file named `belgium.samaki` may contain a **cell** :

```
-- duckdb:circles.csv
SELECT
ST_BUFFER( .. ) as circle, *
FROM y
WHERE
    latitude > < ..raku expression.. >
    and latitude < < ..raku expression.. >
```

- **type** is duckdb
- **name** is circles
- **extension** is csv

Cells are delimited by

- lines like: `-- type:name.extension`

and will be handled by

- the first **plugin** that matches the type ("duckdb")
- the first **plugout** that match the extension ("csv")

and the contents are generated by interpolating

- `< angle brackets >` with Raku expressions
- or `<<< triple less/greater-thans >>>`

The console interface

```
1 --
2 my ($lat, $lon) = 50.8134719, 4.3812357;
3
4 -- postgres
5 select
6   st_makepoint( < "$lon,$lat" > ),
7   st_buffer(
8     st_makepoint( < "$lon,$lat" > )::geography, 100
9   )
10
```

~
~
~
~
~
~
~
~
~
~
~

circle.samaki

10,0-1

All

The console interface

```
-- circle --  
0 my ($lat, $lon) = 50.8134719, 4.3812357;  
1  
0 postgres (csv) [run] → cell-1.csv  
1 select  
2   st_makepoint( < "$lon,$lat" > ),  
3   st_buffer(  
4     st_makepoint( < "$lon,$lat" > )::geography, 100  
5   )  
0  
1  
2  
3  
4  
5  
circle/  
0 files found.
```

run `sam` for the console UI

The console interface

```
~ circle ~

0  my ($lat, $lon) = 50.8134719, 4.3812357;
1  ]
   postgres (csv)      [run] → cell-1.csv
0  select
1  st_makepoint( 4.3812357,50.8134719 ),
2  st_buffer(
3  st_makepoint( 4.3812357,50.8134719)::geography, 100
4  )
5  ]

circle/
0 files found.
```

Use [m] to toggle between eval and raw mode.

The console interface

```
~ circle ~

0  my ($lat, $lon) = 50.8134719, 4.3812357;
1  ]

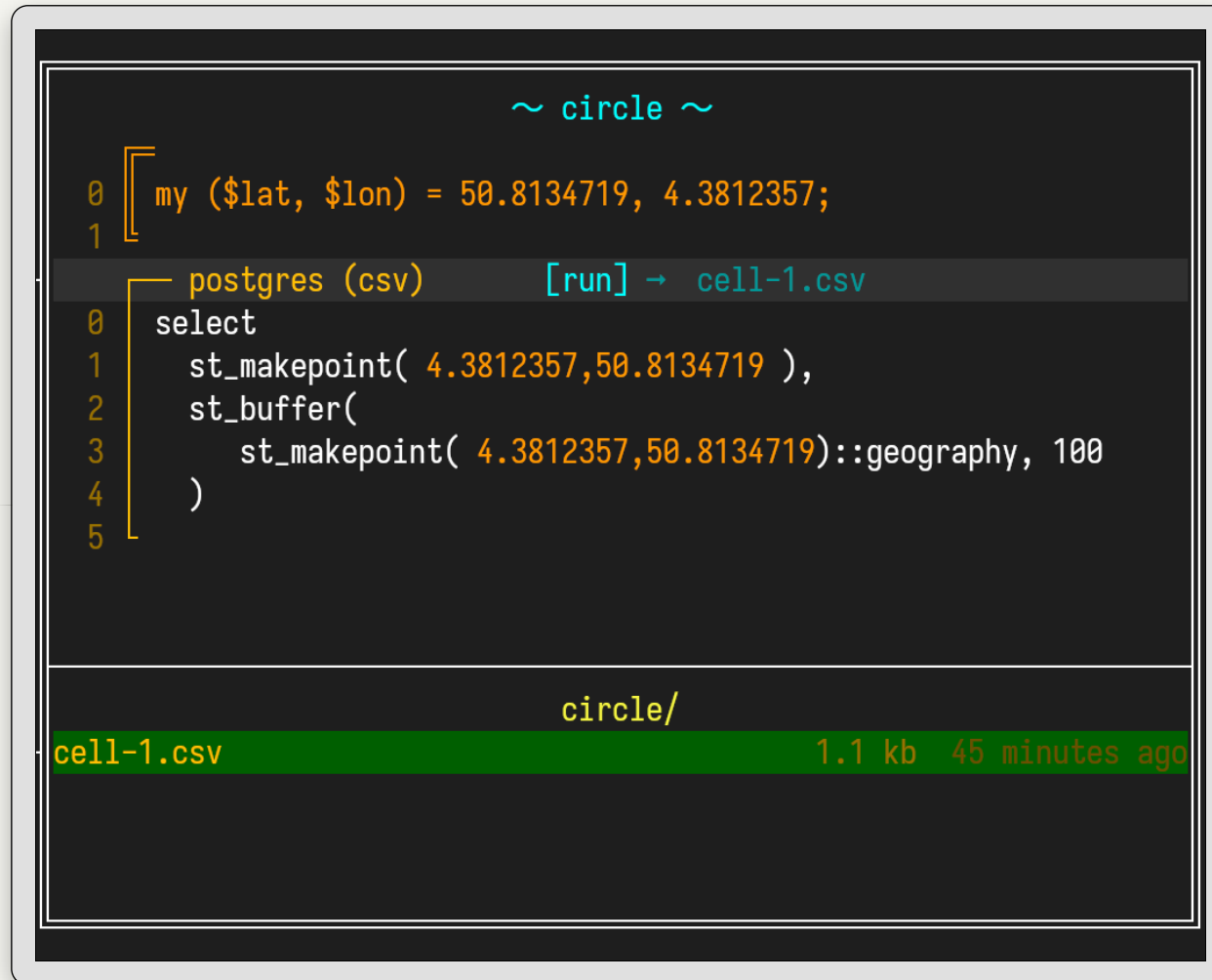
postgres (csv) [run] → cell-1.csv
0  select
1  st_makepoint( 4.3812357,50.8134719 ),
2  st_buffer(
3  st_makepoint( 4.3812357,50.8134719)::geography, 100
4  )
5  ]

1 row, 2 columns
```

st_makepoint	st_buffer
01010000004B45BEA662861140855BE3D81F684940	0103000020E61000000

Select cells to run them

The console interface



The screenshot shows the Samaki console interface. At the top, it says '~ circle ~'. Below that, there's a code editor with a SQL query. The query is: `my ($lat, $lon) = 50.8134719, 4.3812357;` on line 0, and `select` on line 1. The query continues on lines 2, 3, 4, and 5: `st_makepoint( 4.3812357,50.8134719 ),`, `st_buffer(`, `st_makepoint( 4.3812357,50.8134719)::geography, 100`, and `)`. To the right of the code editor, there's a button labeled 'postgres (csv)' and a link labeled '[run] → cell-1.csv'. Below the code editor, there's a file list showing 'circle/' and 'cell-1.csv'. The 'cell-1.csv' file is highlighted in green and shows a size of '1.1 kb' and a time of '45 minutes ago'.

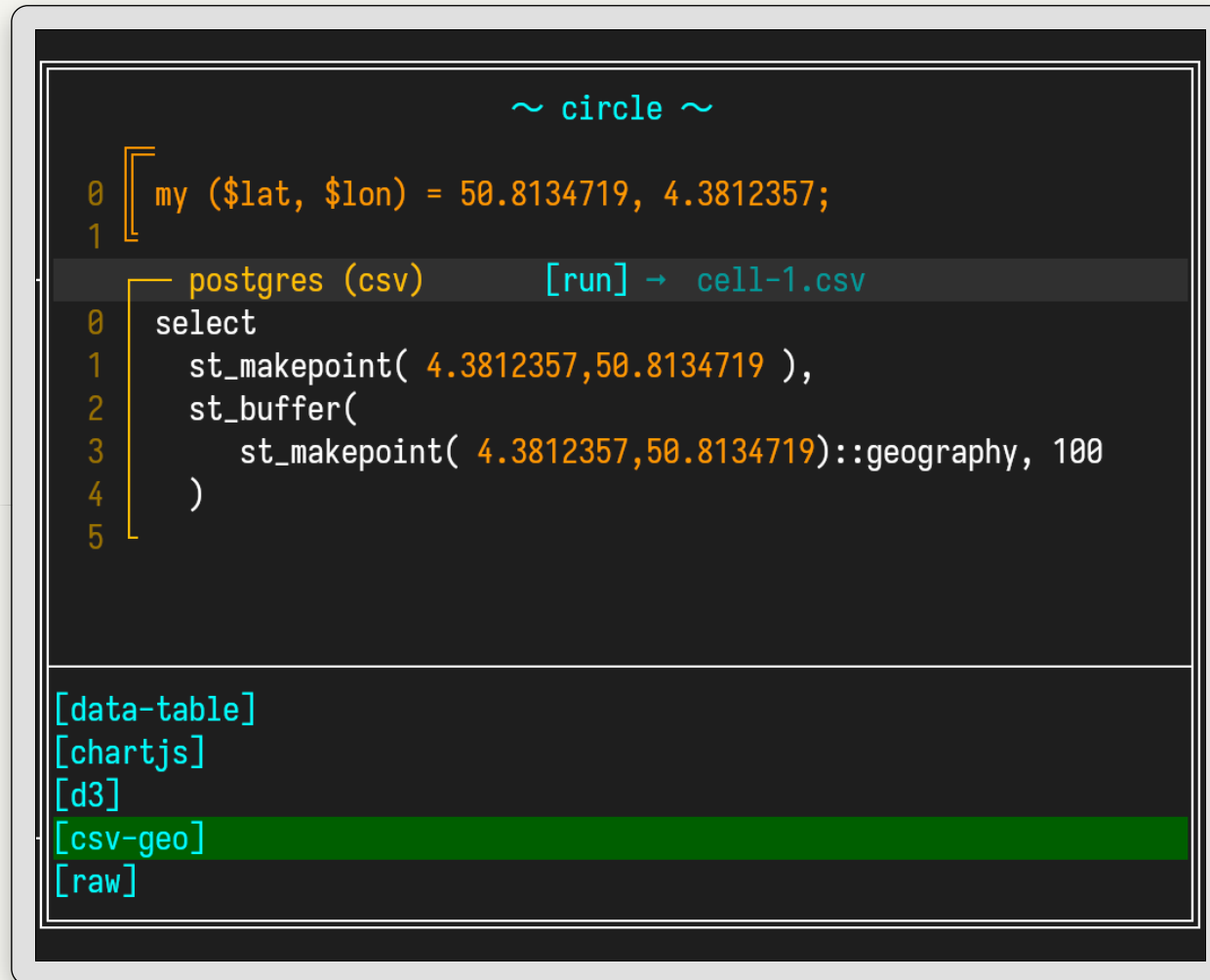
```
~ circle ~

0 my ($lat, $lon) = 50.8134719, 4.3812357;
1
postgres (csv) [run] → cell-1.csv
0 select
1   st_makepoint( 4.3812357,50.8134719 ),
2   st_buffer(
3     st_makepoint( 4.3812357,50.8134719)::geography, 100
4   )
5

circle/
cell-1.csv 1.1 kb 45 minutes ago
```

Select output files to view them

The console interface



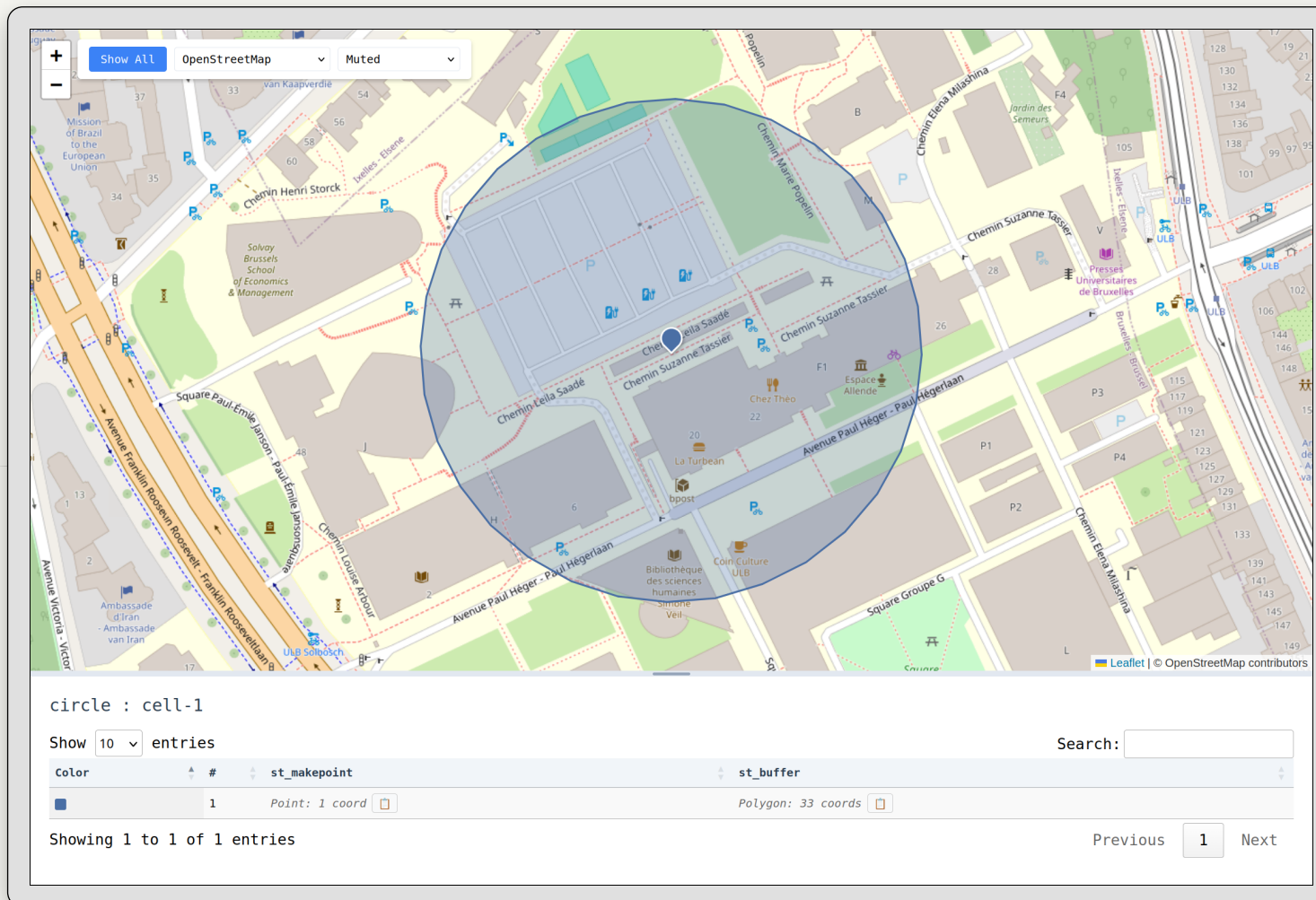
The screenshot shows the Samaki console interface. At the top, there is a header '~ circle ~'. Below it, a SQL query is displayed with line numbers 0 to 5 on the left. The query is: `my ($lat, $lon) = 50.8134719, 4.3812357;` on line 0, and `select` on line 1, `st_makepoint( 4.3812357,50.8134719 ),` on line 2, `st_buffer(` on line 3, `st_makepoint( 4.3812357,50.8134719)::geography, 100` on line 4, and `)` on line 5. To the right of the query, there is a button labeled 'postgres (csv)' and a button labeled '[run] → cell-1.csv'. Below the query, there is a list of plugouts: '[data-table]', '[chartjs]', '[d3]', '[csv-geo]', and '[raw]'. The '[csv-geo]' plugout is highlighted with a green background.

```
~ circle ~

0  my ($lat, $lon) = 50.8134719, 4.3812357;
1
2  postgres (csv)      [run] → cell-1.csv
3  select
4  st_makepoint( 4.3812357,50.8134719 ),
5  st_buffer(
   st_makepoint( 4.3812357,50.8134719)::geography, 100
 )

[data-table]
[chartjs]
[d3]
[csv-geo]
[raw]
```

All plugouts are available to view the file.



CSVGeo is a plugout reads GeoJSON, WKT, EWKB, EWKT, uses JS DataTables to view CSV content, auto detects spatial columns and has options for tiles and color schemes.

Overall flow

- Define input types (plugins)
- Define output types (plugouts)
- Write code
- Run cells
- Share data between cells
- Manage data
- Visualize data
- Write more code
- repeat!

Examples

Example 0 : Geocode with Raku + Nominatim, buffer with postgis

```
                                -- circle-too --
0  raku (csv)                    [run] → lonlat.csv
1  use Webservice::Nominatim 'nom';
2  say nom.search('University Libre de Brussels')[0]<lon lat>.join(',');

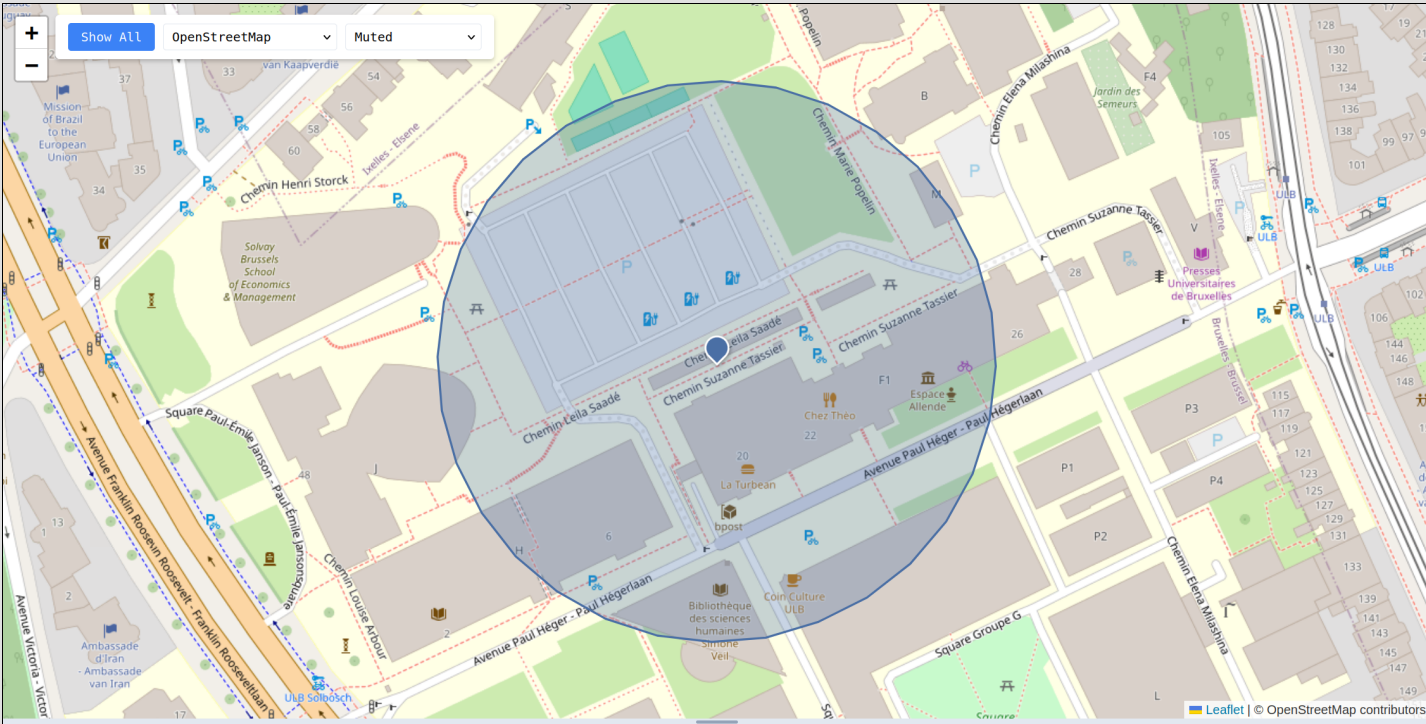
0  postgres (csv)               [run] → cell-1.csv
1  select
2  st_makepoint( < c('lonlat').out > ),
3  st_buffer(
4  st_makepoint( < c('lonlat').out > )::geography, 100
5  )

                                circle-too/
cell-1.csv                      1.1 kb      3 minutes ago
lonlat.csv                      21 b        4 minutes ago
```

- Use Nominatim (via a Raku module) to geocode ULB.
- Use `c(...).out` to get the raw output (`c` is short for cell).
- Cell names ("latlon") get other cells, so do numbers (e.g. `c(0)`)

Examples

Example 0 : Geocode with Raku + Nominatim, buffer with postgis



circle : cell-1

Show 10 entries

Search:

color	#	st_makepoint	st_buffer
	1	Point: 1 coord 	Polygon: 33 coords 

Showing 1 to 1 of 1 entries

Previous 1 Next

Examples

Example 1 : Python + R

```

-- pyr --
python3 (json) [run] → bbox.json
0 from geopy.geocoders import Nominatim
1 import json
2
3 b = Nominatim(user_agent='nom').geocode("University Libre de Bruxelles", \
4     geometry='geojson').raw['boundingbox']
5 print(json.dumps(dict(zip(['lat-min', 'lat-max', 'lon-min', 'lon-max'], b))))
6
R-repl [run] cell-1
0 library(tidyverse)
1 library(sf)
2 library(osmdata)
3
4 bbox <- c( <c('bbox').json<lon-min lat-min lon-max lat-max>.join(',') > )
5
6 opq(bbox) |>
7   add_osm_feature(key = "building") |>
8   osmdata_sf() |>

```

bbox.json pyr/ 99 b 8 minutes ago

- Type `python3` uses a python plugin to run this as a script.
- JSON is parsed and available via `c('bbox').json` .

Examples

Example 1 : Python + R

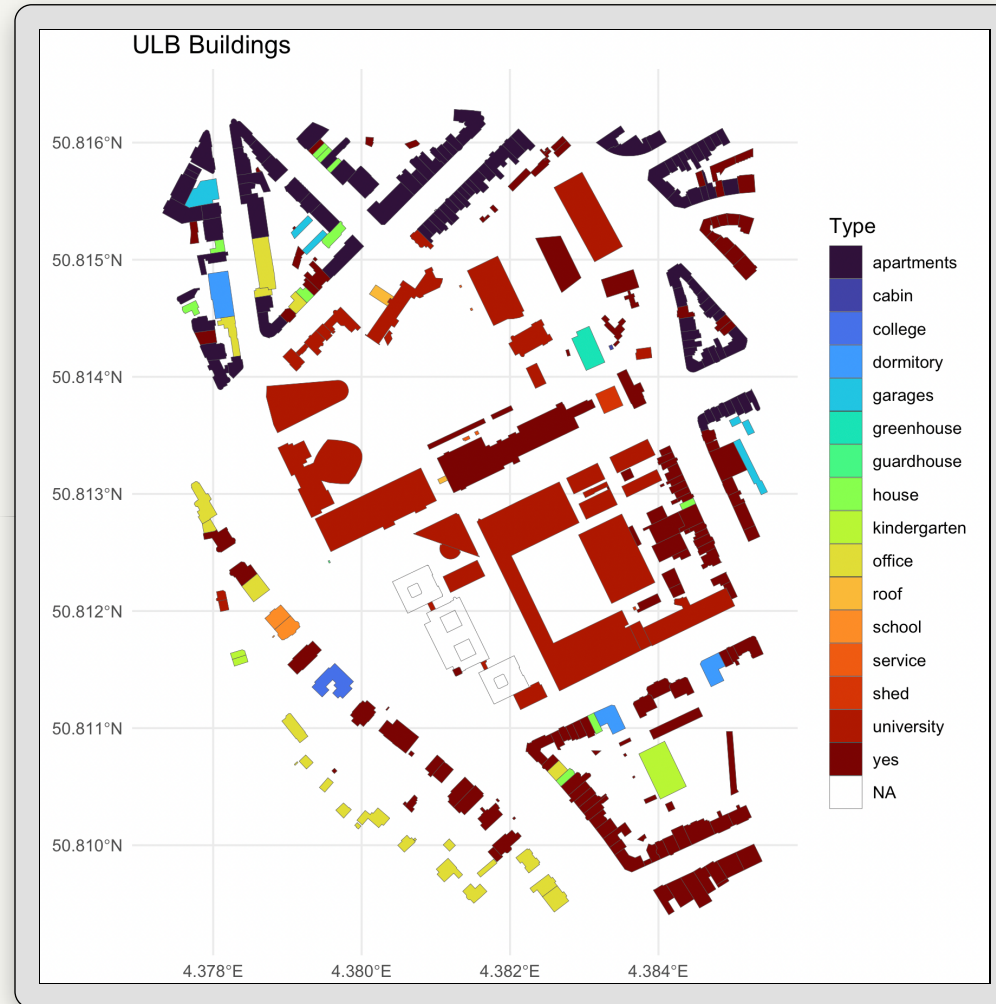
```
5 | print(json.dumps(dict(zip(['lat-min', 'lat-max', 'lon-min', 'lon-max'], b))))
6 |
R-repl [run] cell-1
0 | library(tidyverse)
1 | library(sf)
2 | library(osmdata)
3 |
4 | bbox <- c(4.3777200,50.8096525,4.3851645,50.8159744)
5 |
6 | opq(bbox) |>
7 |   add_osm_feature(key = "building") |>
8 |   osmdata_sf() |>
9 |   pluck("osm_polygons") |>
10 |  st_transform('EPSG:31370') |>
11 |  ggplot() +
12 |    geom_sf(aes(fill = building), color = "gray30", linewidth = 0.1) +
13 |    scale_fill_viridis_d(option = "turbo") +
14 |    theme_minimal() +
15 |    labs(title = "ULB Buildings", fill = "Type")

> library(sf)
library(osmdata)
Linking to GEOS 3.13.1, GDAL 3.10.3, PROJ 9.5.1; sf_use_s2() is TRUE
> library(osmdata)
Data (c) OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.org/copyright
>
> bbox <- c(4.3777200,50.8096525,4.3851645,50.8159744)
```

- R-repl uses a plugin that spawns a Pseudo TTY and interacts with the R CLI interpreter.
- The R interpreter opens windows when `ggplot` is called.

Examples

Example 1 : Python + R



Examples

Example 2 : bash + raku + duckdb

Let's find the top OSM tags in the ULB bounding box

```
raku (json) [run] → bbox.json
0 use Webservice::Nominatim 'nom';
1 use JSON::Fast;
2 with nom.search('University Libre de Bruxelles', format => 'json').head {
3   put to-json % = (<lat- lon-> X~ <min max>) Z=> .<boundingbox>.List
4 }
5

duck (csv) [run] → tags.csv
0 select
1 unnest(map_keys(tags)) as tag_name, lat, lon,
2 unnest(map_values(tags)) as tag_value
3 from ST_ReadOSM('belgium-latest.osm.pbf')
4 where lat between 50.8096525 and 50.8159744
5 and lon between 4.3777200 and 4.3851645
6

duck (csv) [run] → list.csv
0 select tag_name, count(1) from 'tags.csv' group by 1 order by 2 desc
1 limit 20
2

{
  "lat-max": "50.8159744",
  "lon-max": "4.3851645",
  "lat-min": "50.8096525",
  "lon-min": "4.3777200"
}

-- done in 2 seconds --
```

Examples

Example 2 : bash + raku + duckdb

Let's find the top OSM tags in the ULB bounding box

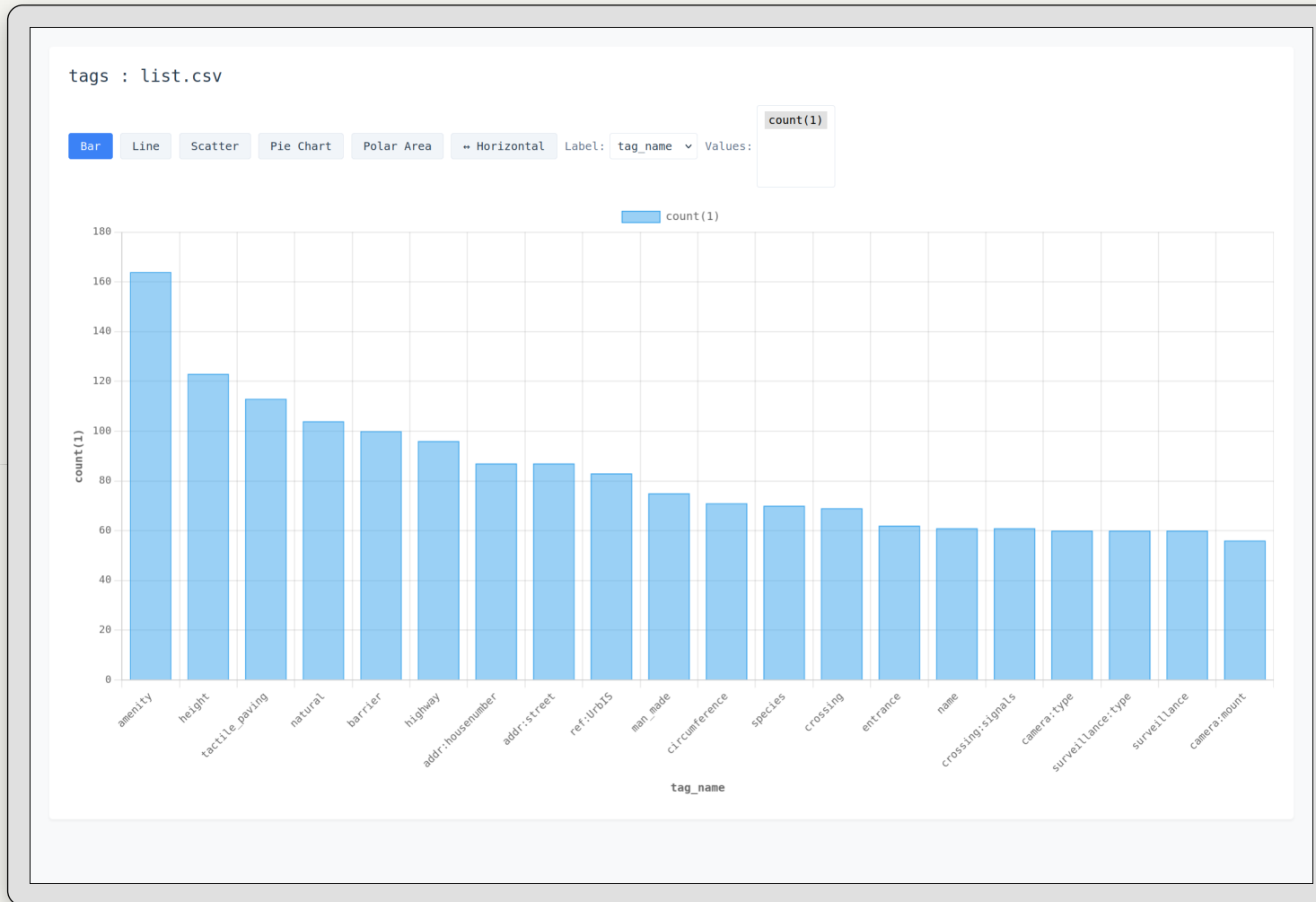
```
~ tags ~
0 raku (json) [run] → bbox.json
1 use Webservice::Nominatim 'nom';
2 use JSON::Fast;
3 with nom.search('University Libre de Bruxelles', format => 'json').head {
4   put to-json % = (<lat- lon-> X~ <min max>) Z=> .<boundingbox>.List
5 }
6
7 duck (csv) [run] → tags.csv
8 select
9   unnest(map_keys(tags)) as tag_name, lat, lon,
10  unnest(map_values(tags)) as tag_value
11 from ST_ReadOSM('belgium-latest.osm.pbf')
12 where lat between 50.8096525 and 50.8159744
13       and lon between 4.3777200 and 4.3851645
14
15 duck (csv) [run] → list.csv
16 select tag_name, count(1) from 'tags.csv' group by 1 order by 2 desc
17 limit 20

Matched plugout handler duckview for /home/bduggan/fosdem/samaki-talk/tags/list.csv
[data-table]
[chartjs]
[d3]
[csv-geo]
[raw]
-- Loading resources from /home/bduggan/.duckdbrc
```

The ChartJS plugout is a DWIM interface to Chart.js

Examples

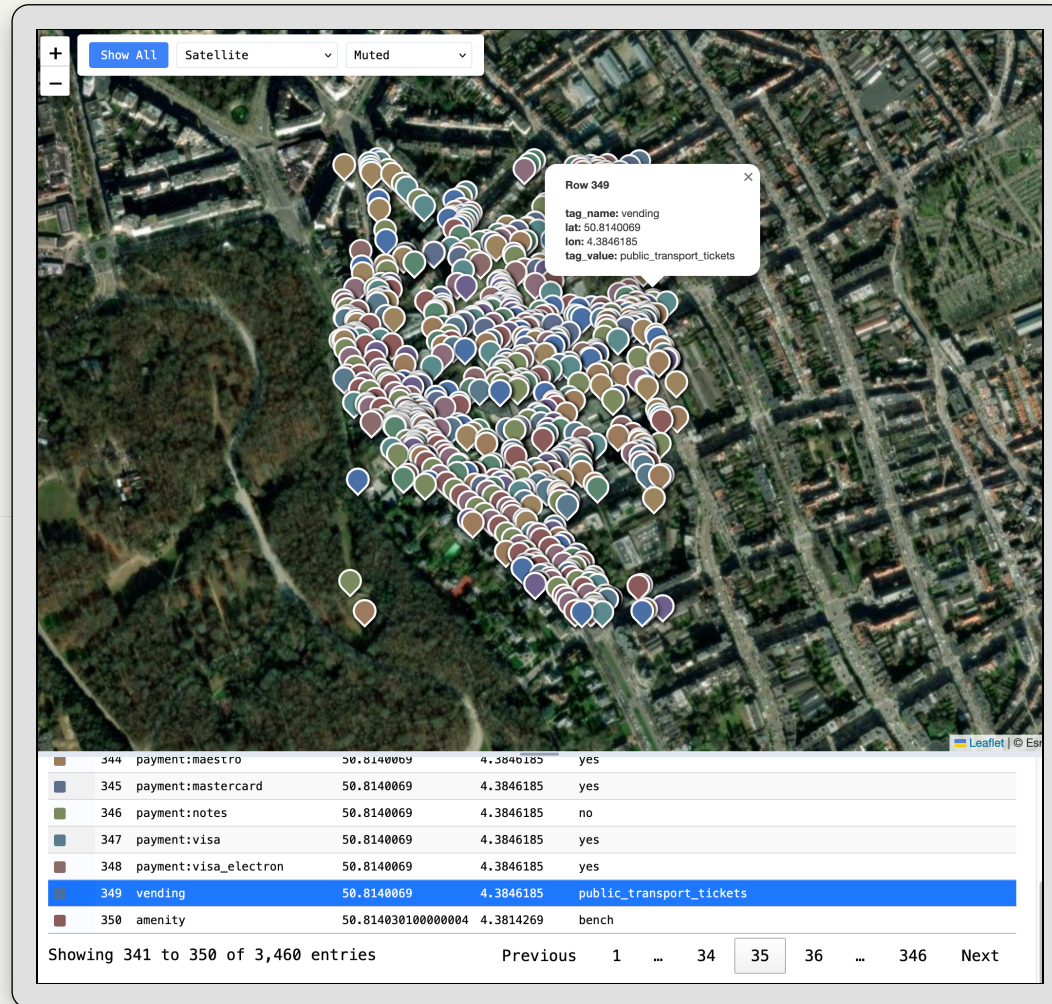
Example 2 : bash + raku + duckdb



- The ChartJS plugout picks a graph type based on heuristics about the column types and contents.
- And provides options to change chart type and column selection.

Examples

Example 2 : bash + raku + duckdb



- View another cell using the CSVGeo plugout.
- Which detects latitude + longitude columns (and geojson columns seen earlier).

Examples

Example 2 : bash + raku + duckdb

```
~ tags ~
duck (csv) [run] → bin.csv
0 load h3;
1 select
2 h3_latlng_to_cell(lat,lon,11),
3 count(1)
4 from 'tags.csv'
5 group by 1

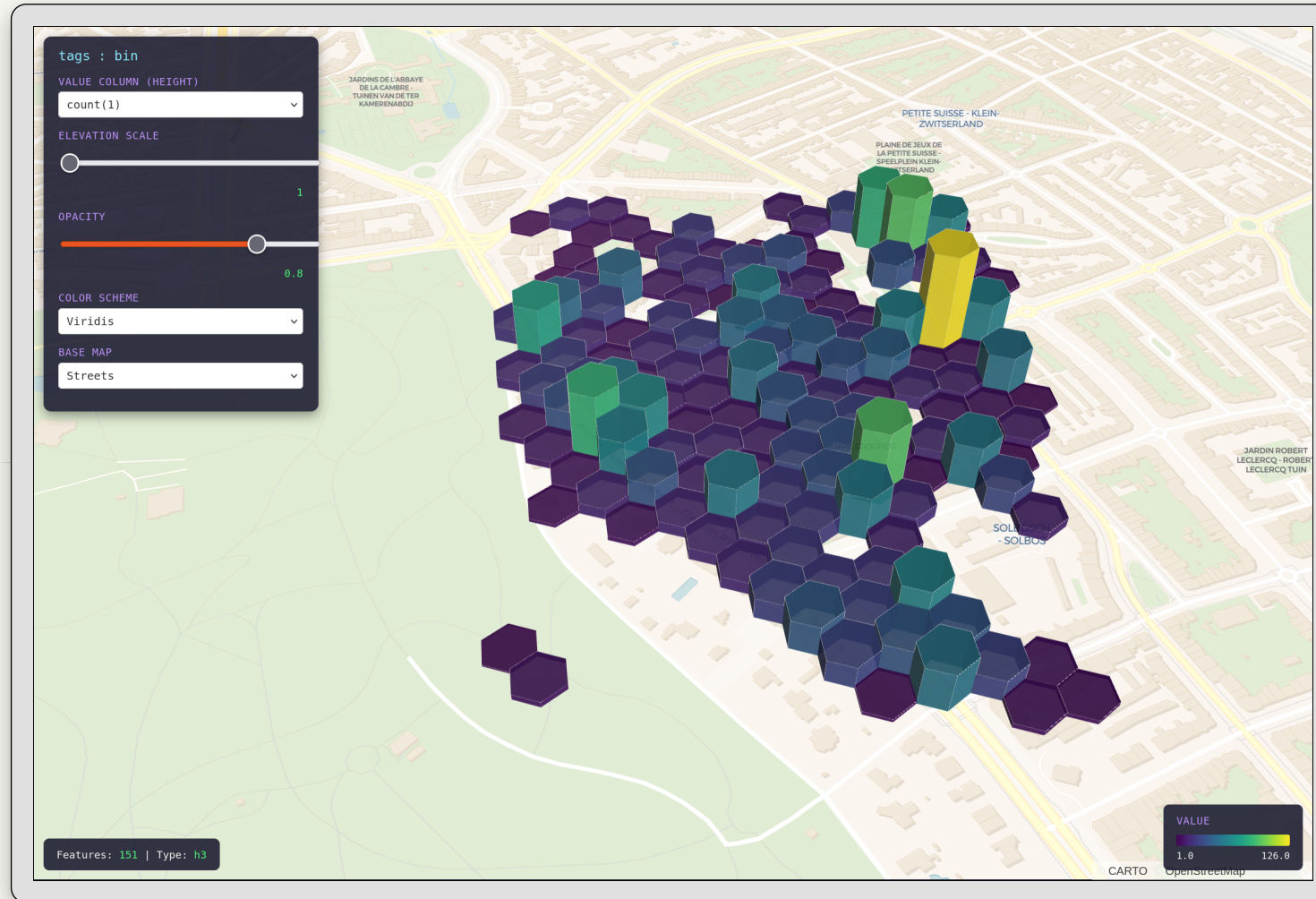
tags/
bin.csv 3.2 kb 6 minutes ago

(0:raku*) 03:36
```

- With duckdb's csv handling, it's easy to use a previously generated csv.

Examples

Example 2 : bash + raku + duckdb



- The DeckGLBin plugout supports hex or geohash binning with Deck.gl
- by detecting H3 hex ids or geohash strings.

Examples

Example 3 : LLM + Grass

```
-- llm --
claude (txt) [run] → out.txt
0 Make a sequence of grass commands that sets a high resolution region and then
1 displays a synethetic fractal landscape with shadows and shading
2 and hills and valleys.
3 Do not output anything else. Do not output comments, only commands.
4
5 [run] fractal
0 <cell('out').res.lines[1..*-2].join("\n") >
```

```

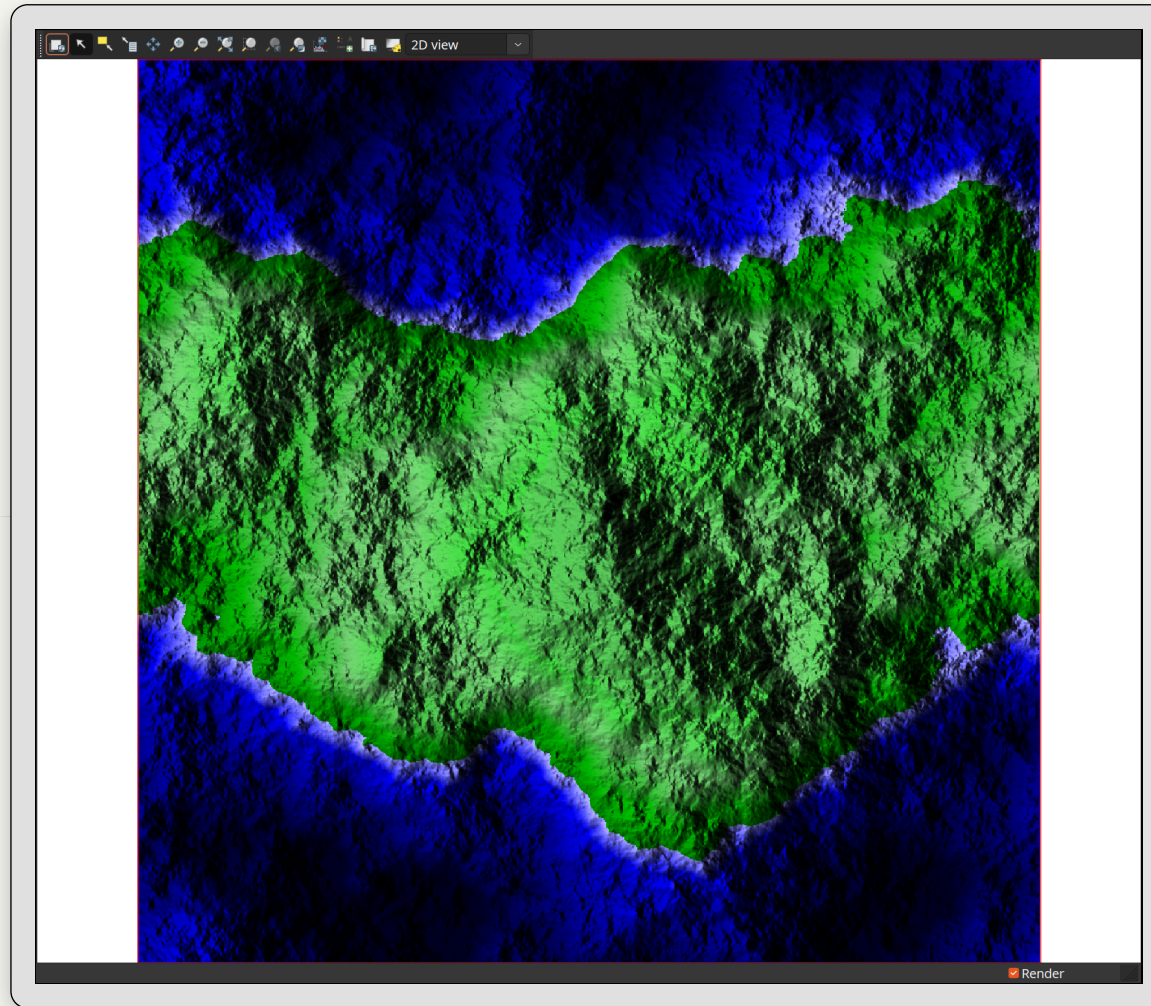
~ llm ~
1 claude (txt) [run] → out.txt
2 0 Make a sequence of grass commands that sets a high resolution region and then
3 1 displays a synthetic fractal landscape with shadows and shading
4 2 and hills and valleys.
5 3 Do not output anything else. Do not output comments, only commands.
6 4
7 grass [run] fractal
8 0 g.region n=5000 s=0 e=5000 w=0 rows=2000 cols=2000
9 1 r.surf.fractal output=fractal_terrain dimension=2.05
10 2 r.colors map=fractal_terrain color=elevation
11 3 r.relief input=fractal_terrain output=fractal_shaded
12 4 d.mon start=wx0
13 5 d.shade shade=fractal_shaded color=fractal_terrain brighten=30

```

- A `claude` plugin can take use the `claude cli` command
- to receive stdin and create stdout for quick LLM work.
- LLMs sometimes produce extra delimiters at the first and last line (``')
- Use the `raku lines` and `and` expression to trim them.
- The `d.mon grass` command opens a window.

Examples

Example 3 : LLM + Grass



Examples

Example 4 : Python + URL + XML + mapnik (bash) + png

```
-- python:bbox
...

-- url:data.osm
https://api.openstreetmap.org/api/0.6/map?bbox= <c('bbox').out >

-- bash
ogr2ogr -f GeoJSON buildings.geojson data.osm multipolygons -where "building IS NOT NULL"
ogr2ogr -f GeoJSON roads.geojson data.osm lines -where "highway IS NOT NULL"

-- raku:style.xml
use Map::Mapnik;
my $map = Mapnik::Map.new:
    background-color => '#f8f4f0', fontsets => [ ... ], styles => [ ...]
    data-sources => [
        ... buildings.geojson ...
        ... roads.geojson ...
    ];
say $map.to-xml

-- bash
mapnik-render --xml style.xml --img out.png
```

Examples

Example 4 : Python + URL + XML + mapnik (bash) + png



- An "open" plugout uses system `open` (or `xdg-open`) for image (or other files).

Configuring Plugins

Configuration file :

```
plugins => [  
  / duck /           => 'Samaki::Plugin::DuckDB',  
  / something_else / => ...plugin...  
  ...  
]
```

somewhere else

```
class Samaki::Plugin::DuckDB does Samaki::Plugin {  
  
  method execute(...) {  
    ...  
  }  
  
}
```

- A regex matches a cell **type** to a plugin class.

- Plugins have an **execute** method.

- Plugins read cell contents and
- stream output to the pane.
- write output files.
- interact with other processes.

Interacting with other processes:

Plugin classes can inherit from common ones.

```
class Samaki::Plugin::Postgres does Samaki::Plugin::P
  has $.cmd = 'psql';
}

class Samaki::Plugin::R does Samaki::Plugin::REPL {
  has $.cmd = 'R';
}
```

- Use **Samaki::Plugin::Process** for simple stdin/stdout interaction (with support for args, tempfiles).
- Use **Samaki::Plugin::Repl** for PTY interactions (pseudo TTY like `expect`).
- Also **Samaki::Plugin::Tmux** will use the tmux control protocol to run a process in a tmux window. (new!)

Extending

Make classes right in the config file easily:

```
plugins => [  
  / grass / => Samaki::Plugin::Repl[cmd => 'grass'],  
  
  / claude / => Samaki::Plugin::Process[  
    cmd => 'claude',  
    args => [ '--permission-mode', 'dontAsk',  
              :use-stdin,  
            ],
```

with built-in Raku syntax

- Parameterized roles can be used to generate a class from a role.
- This is called "punning".

Extending

```
plugouts => [  
  / html /      => 'Samaki::Plugout::HTML',  
  / csv /       => 'Samaki::Plugout::CSVGeo',  
  / csv /       => 'Samaki::Plugout::DeckGLBin',  
  / geojson /   => 'Samaki::Plugout::Geojson',  
  / .* /        => 'Samaki::Plugout::Open',  
  ...
```

Somewhere else

```
class Samaki::Plugout::Open does Samaki::Plugout {  
  
  method execute( IO::Path $path!, ... ) {  
    shell <<open $path>>;  
  }  
}
```

Plugouts also implement **execute**.

- Raku has gradual typing
- strict types enforce the plugin/
plugout interface
- at compilation time

Extending

- lots of plugins included

Plugin	Type	Description
=====	=====	=====
Bash	Process	Execute contents as a bash program
Code		Evaluate raku code in the current process
Duck	Process	Run SQL queries via duckdb executable
Duckie	inline	Run SQL queries via L<Duckie> inline driver
File		Display file metadata and info
HTML		Generate HTML from contents
LLM	inline	Send contents to LLM via LLM::DWIM
Markdown	inline	Generate HTML from markdown
Postgres	Process	Execute SQL via psql process
Raku	Process	Run raku in a separate process
Repl::Raku	Repl	Interactive raku REPL (persistent session)
Repl::Python	Repl	Interactive python REPL (persistent session)
Repl::R	Repl	Interactive R REPL (persistent session)
Text		Write contents to a text file

- and plugouts

Plugout	Description
=====	=====
ChartJS	Display CSV as interactive charts in browser (via Chart.js)
CSVGeo	Display CSV that has geojson data using a map in browser (via leaflet)
D3	Display CSV as D3.js visualizations in browser
DataTable	Display CSV in browser with sorting/pagination/search
Duckview	Show CSV summary in bottom pane (via duckdb)
Geojson	Display GeoJSON on map in browser (via leaflet)
HTML	Open HTML content in browser
JSON	Display prettified JSON in bottom pane
Plain	Display plain text in browser
Raw	Open file with system default application
TJless	View JSON in new tmux window (requires jless)

Why Raku?

- Process interaction (PTYs, streaming interaction with async primitives)
- Decoding UTF8 streams, parsing ANSI byte sequences
- Gradual typing for interface robustness
- Language and syntax extensibility
- much more

Other features

In progress

- watch for changes in a directory and auto reload (`--watch`)
- import from Jupyter notebooks (`sam import`)
- output as an HTML page (`sam export`)
- run cells without the UI (`sam run`)

Not yet implemented :

- auto run cells when other cells change
- watch REPL outputs for specific prompts
- [your feature request goes here]

 `bduggan/raku-samaki`
or search `raku.land`
or google for "raku samaki"

This presentation:

`bduggan.github.io/raku-samaki`

The end!



