Keeping Spatial Scripting Sane

with Samaki and Raku

by Brian Duggan

bduggan

FOSDEM 2026

about me

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

Outline

- Motivation
- Samaki
- Examples
- Extending
- Raku

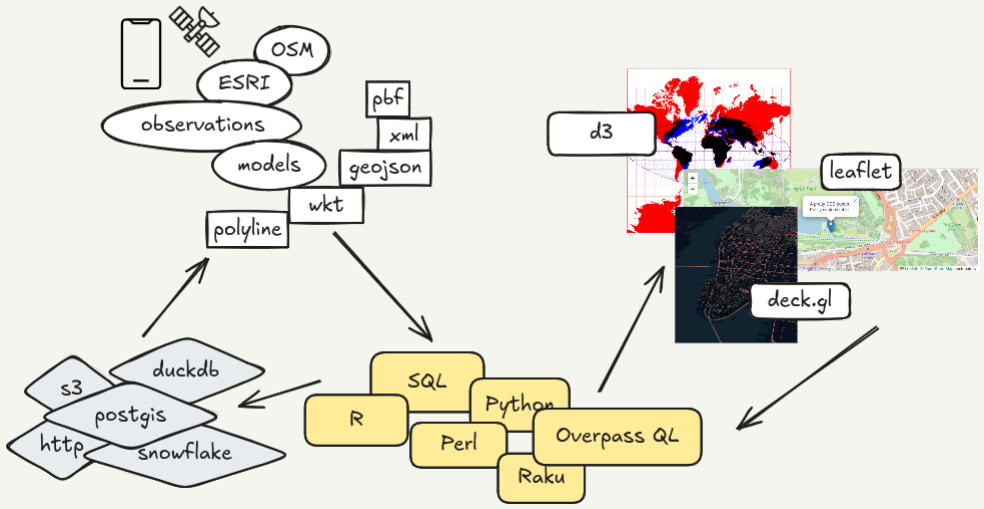
Spatial scripting!

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



Samaki

- 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

Samaki

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".

Samaki

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

Samaki

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.

Samaki

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

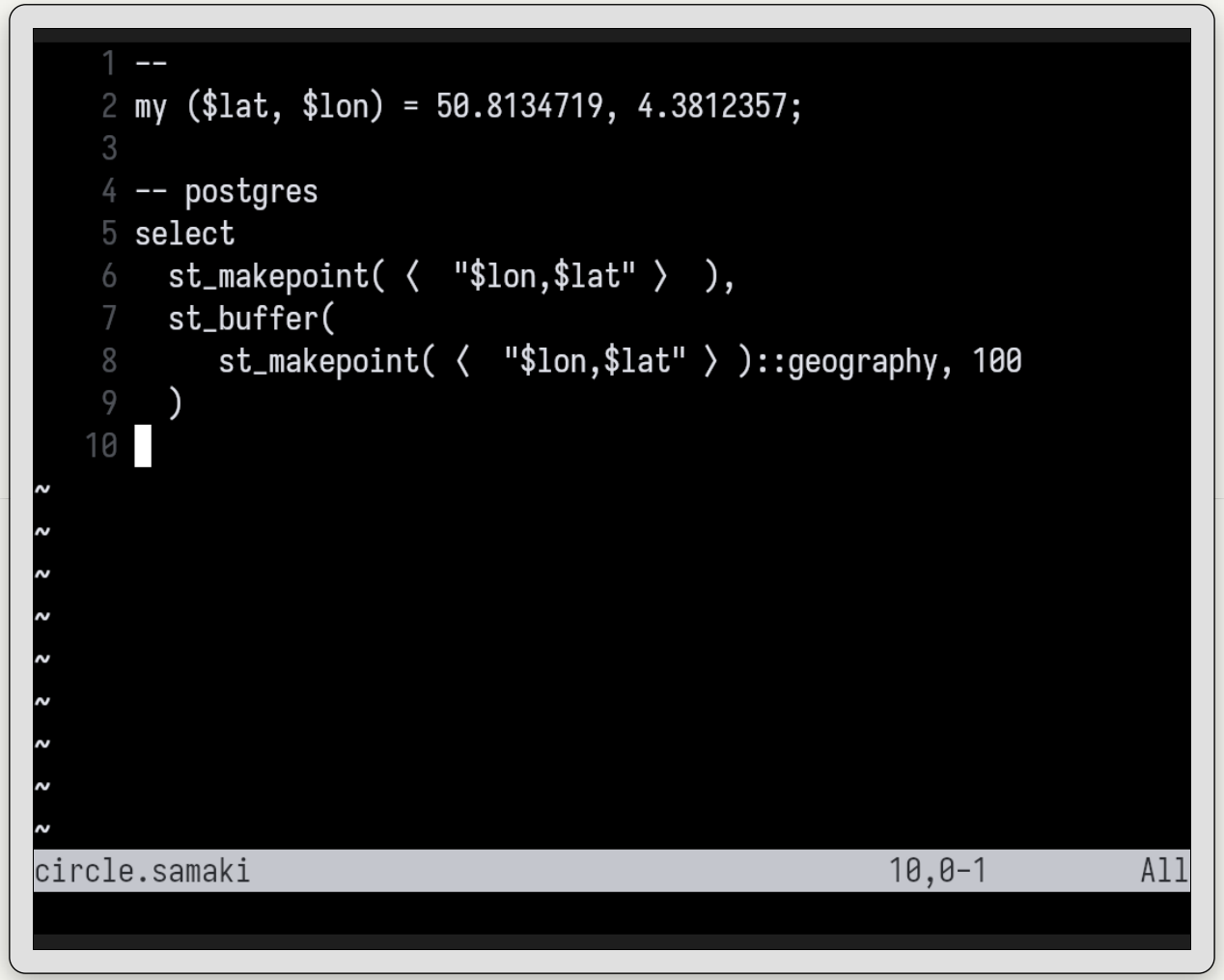
and will be handled by

- 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 >>>

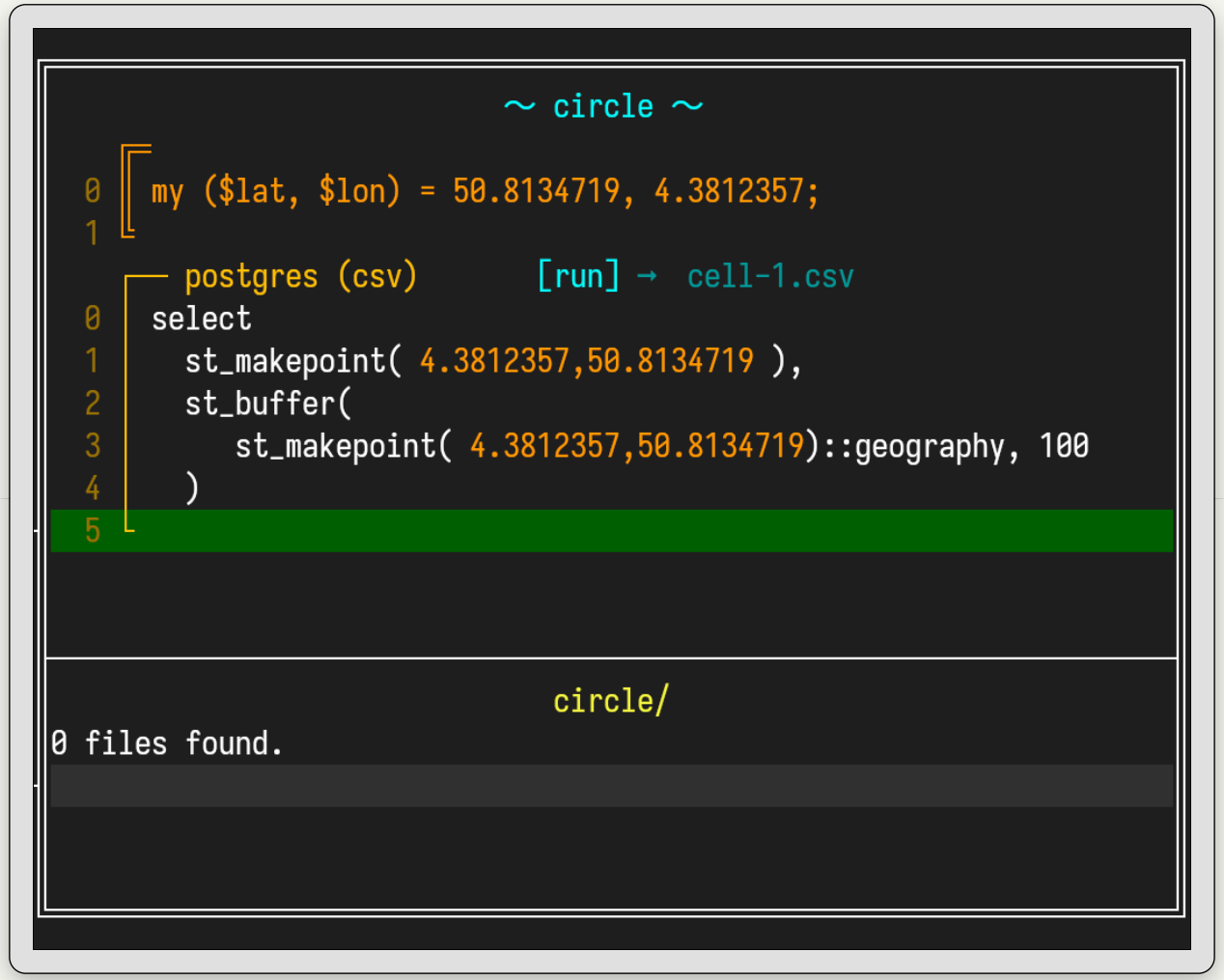
The console interface



The console interface

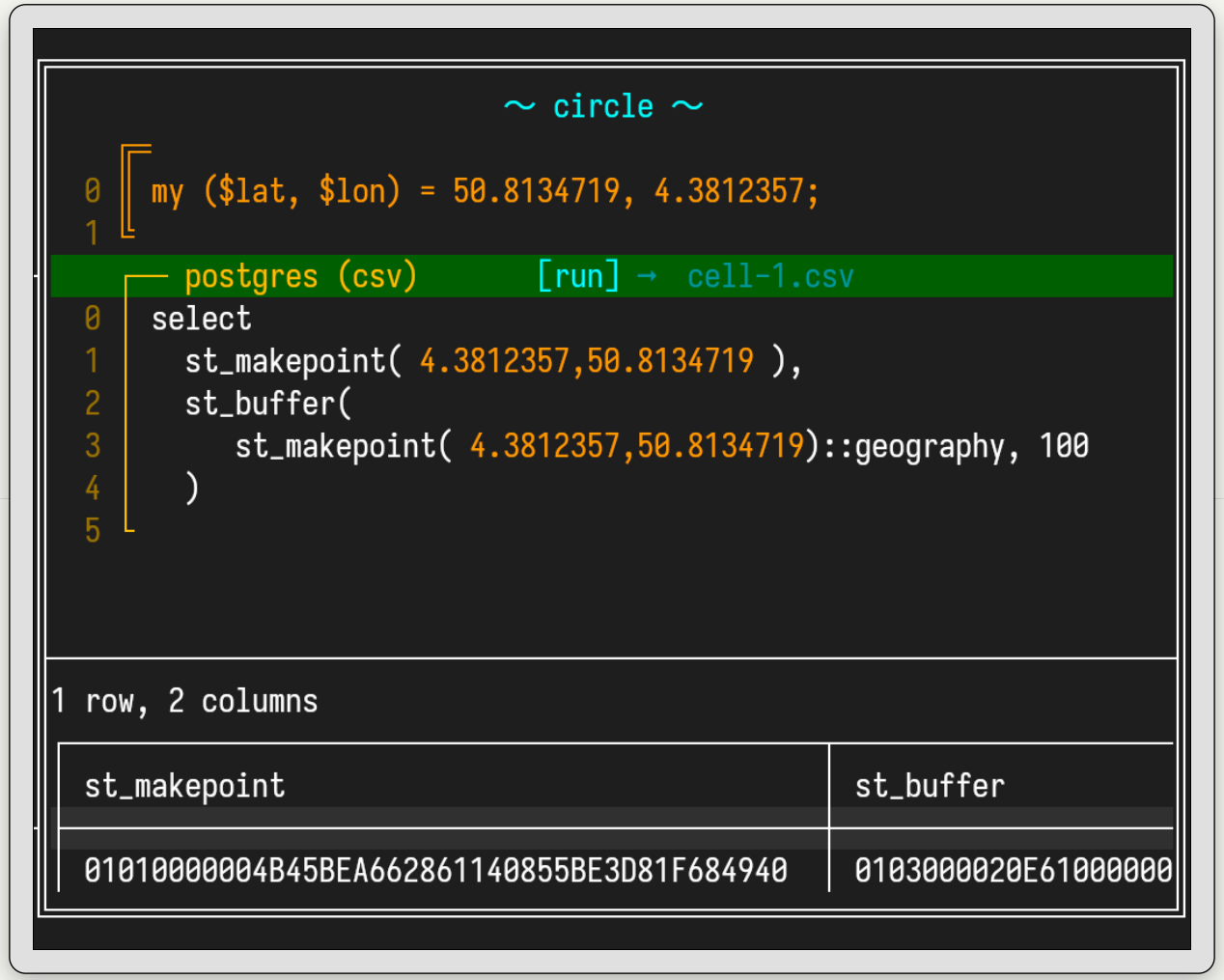
run `sam` for the console UI

The console interface



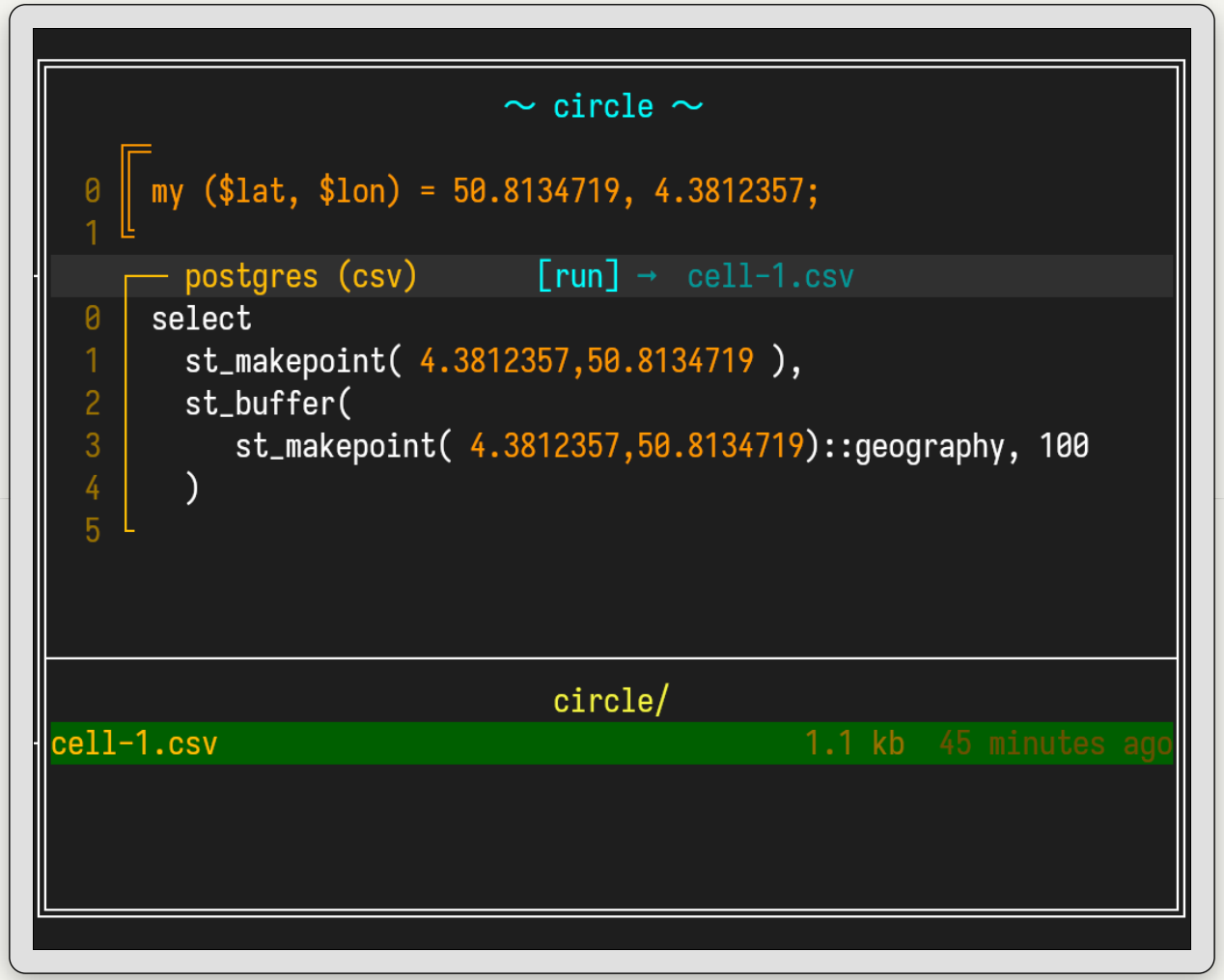
Use [m] to toggle betwen eval and raw mode.

The console interface



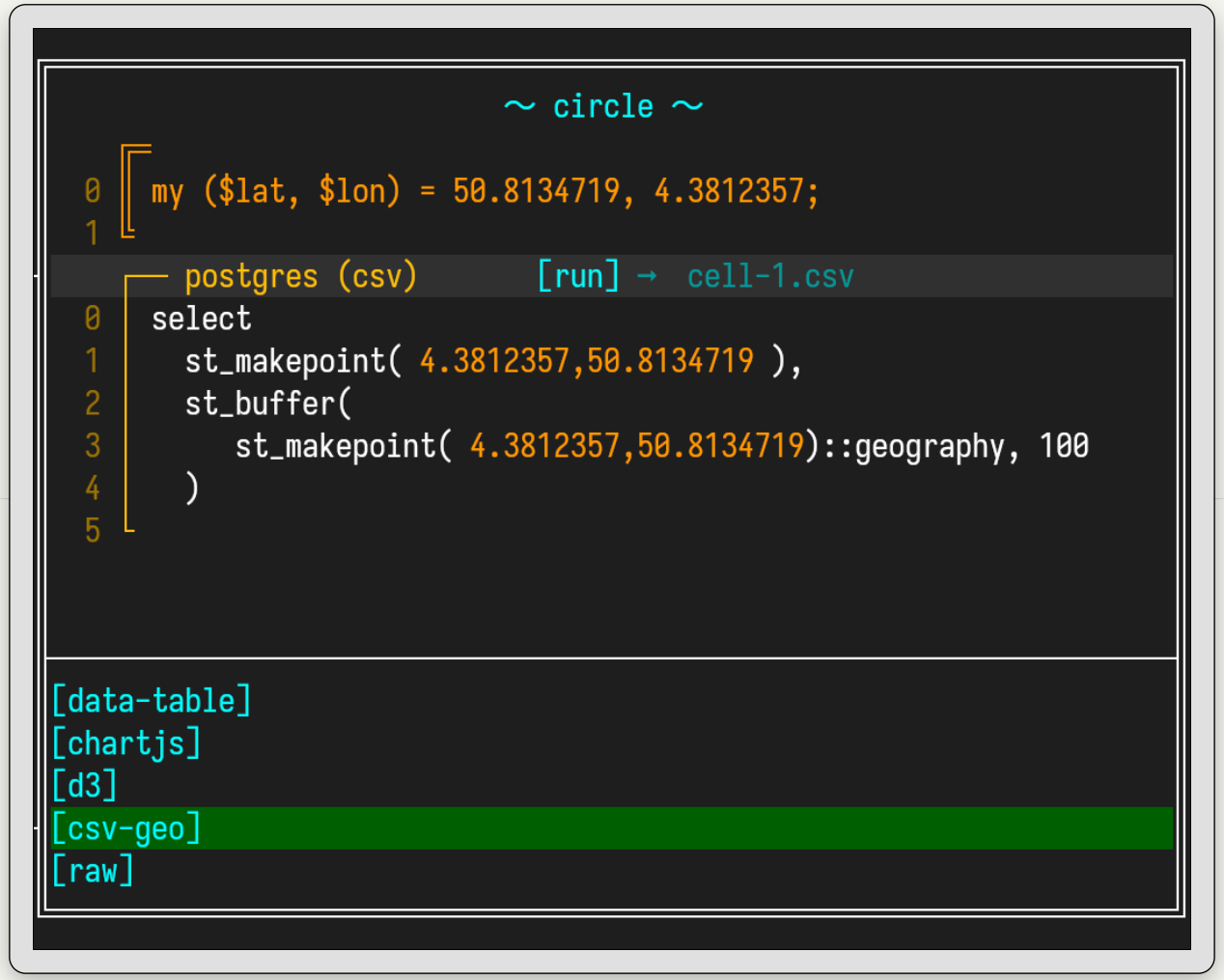
Select cells to run them

The console interface

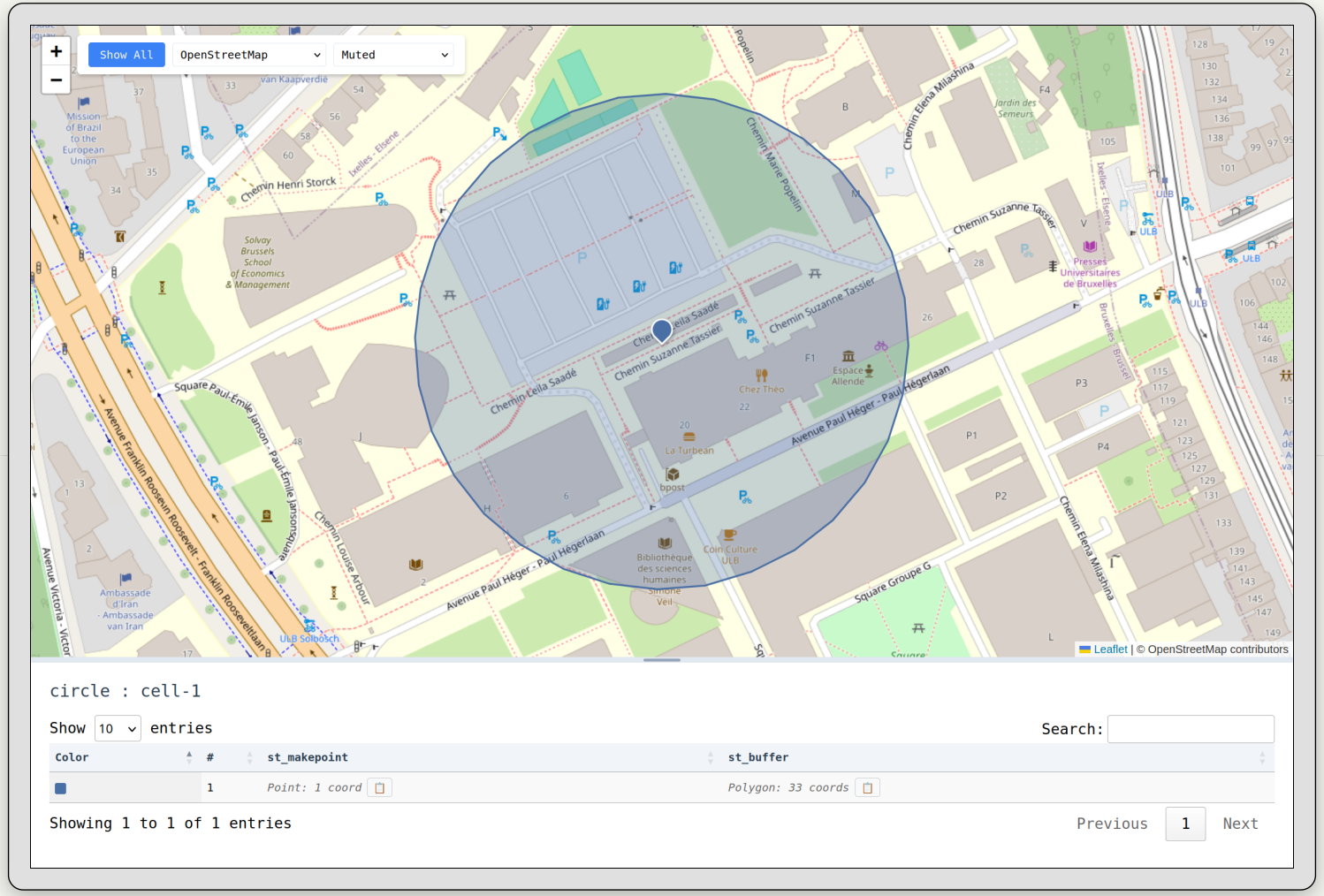


Select output files to view them

The console interface



All plugouts are available to view the file.



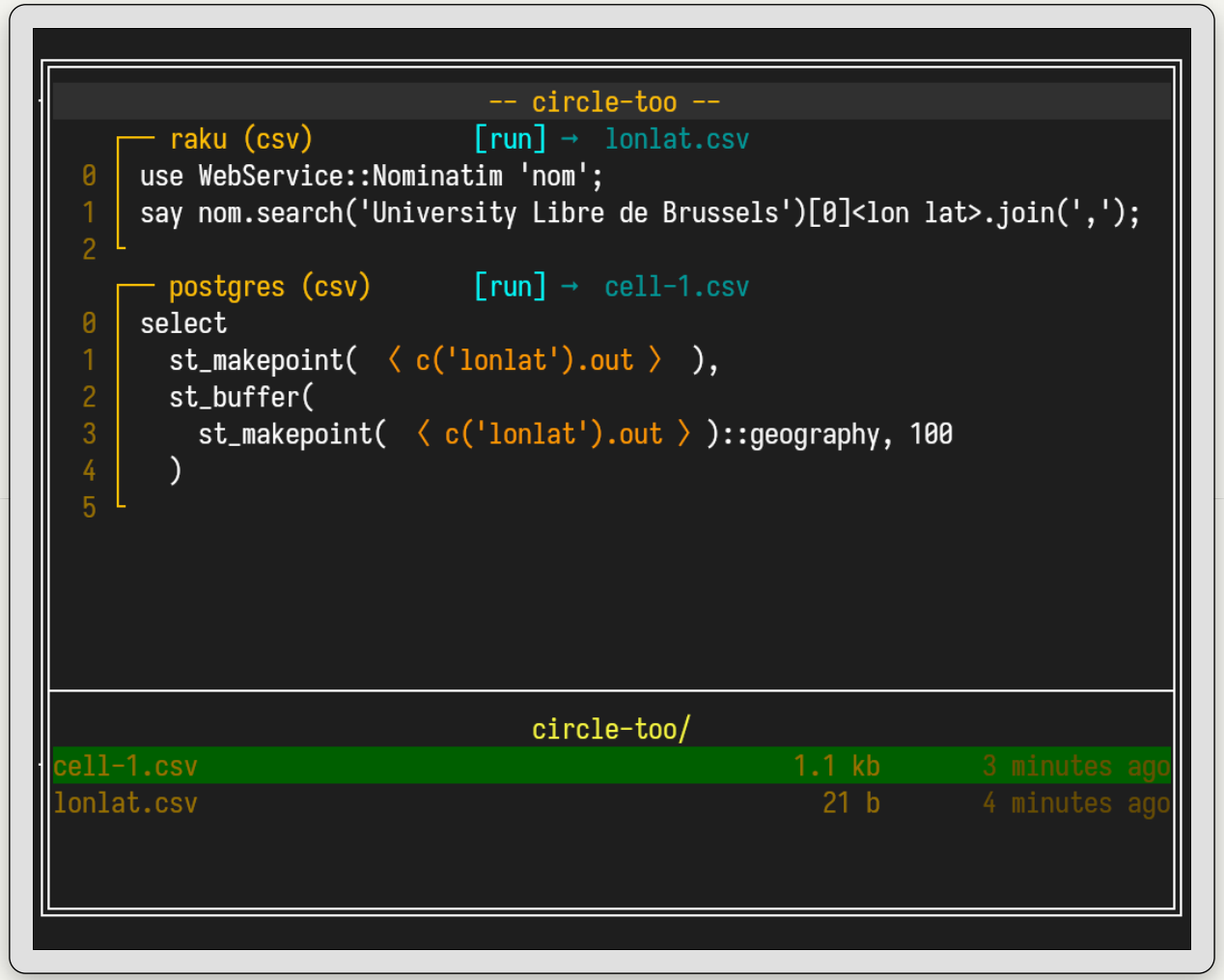
CSVGeo is a plugout reads GeoJSON, WKT, EWKB, EWKT, uses JS DataTables to view CSV content, auto detects spatial columns and has options for tiles and color schemes.

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!

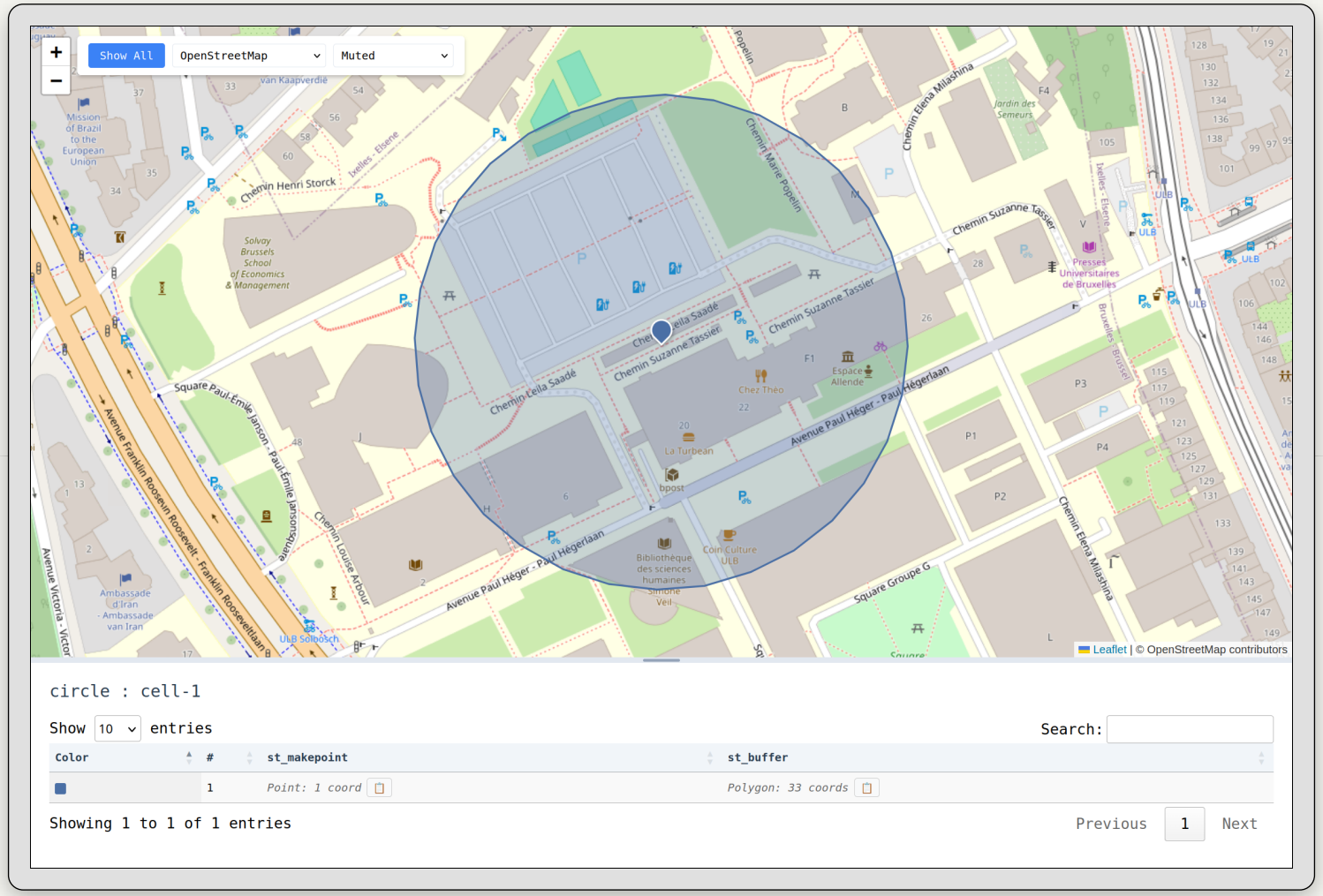
Examples

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



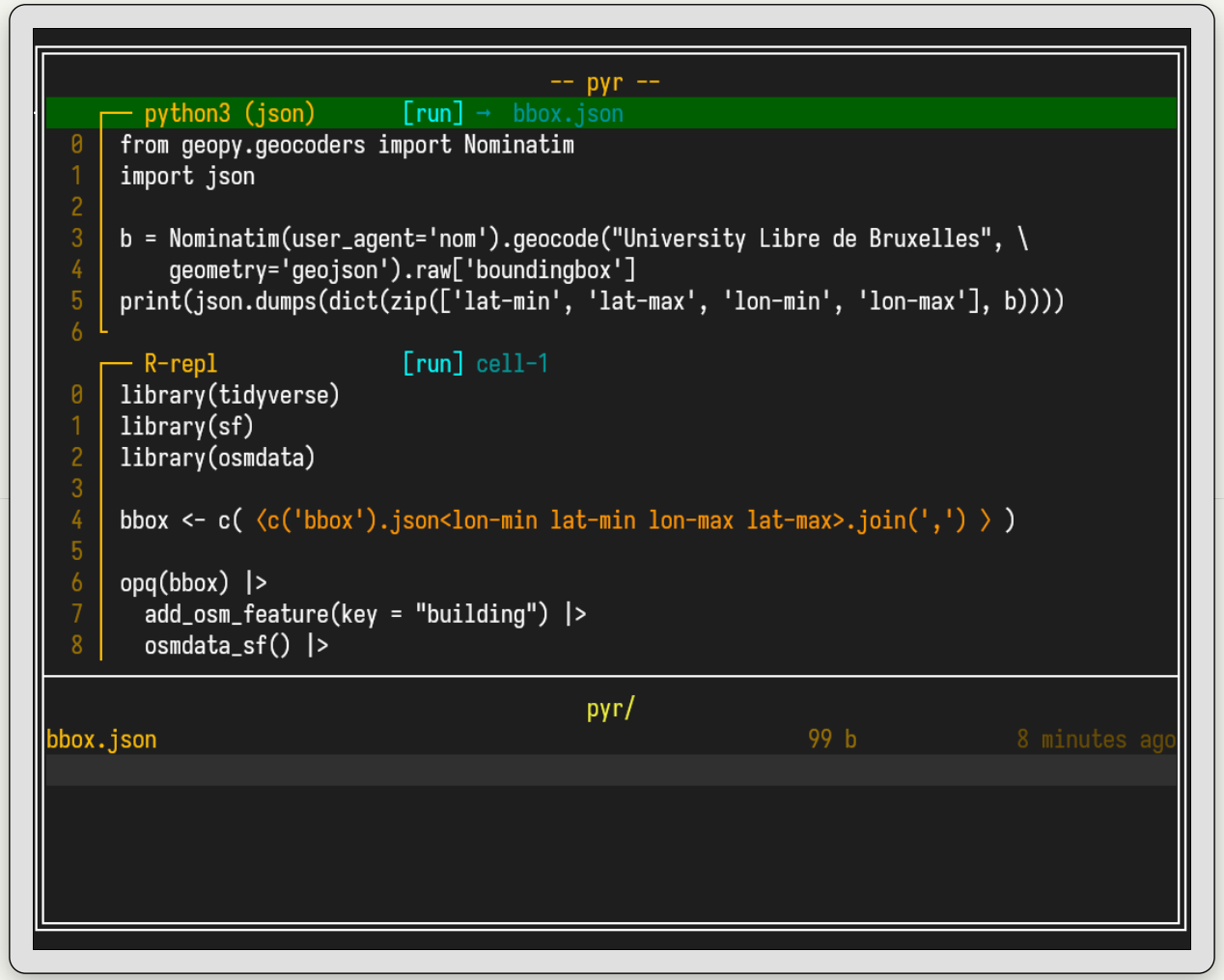
- Use Nominatim (via a Raku module) to geocode ULB.
- Use `c(...).out` to get the raw output (c is short for cell).
- Cell names ("latlon") get other cells, so do numbers (e.g. `c(0)`)

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



Examples

Example 1 : Python + R



- 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

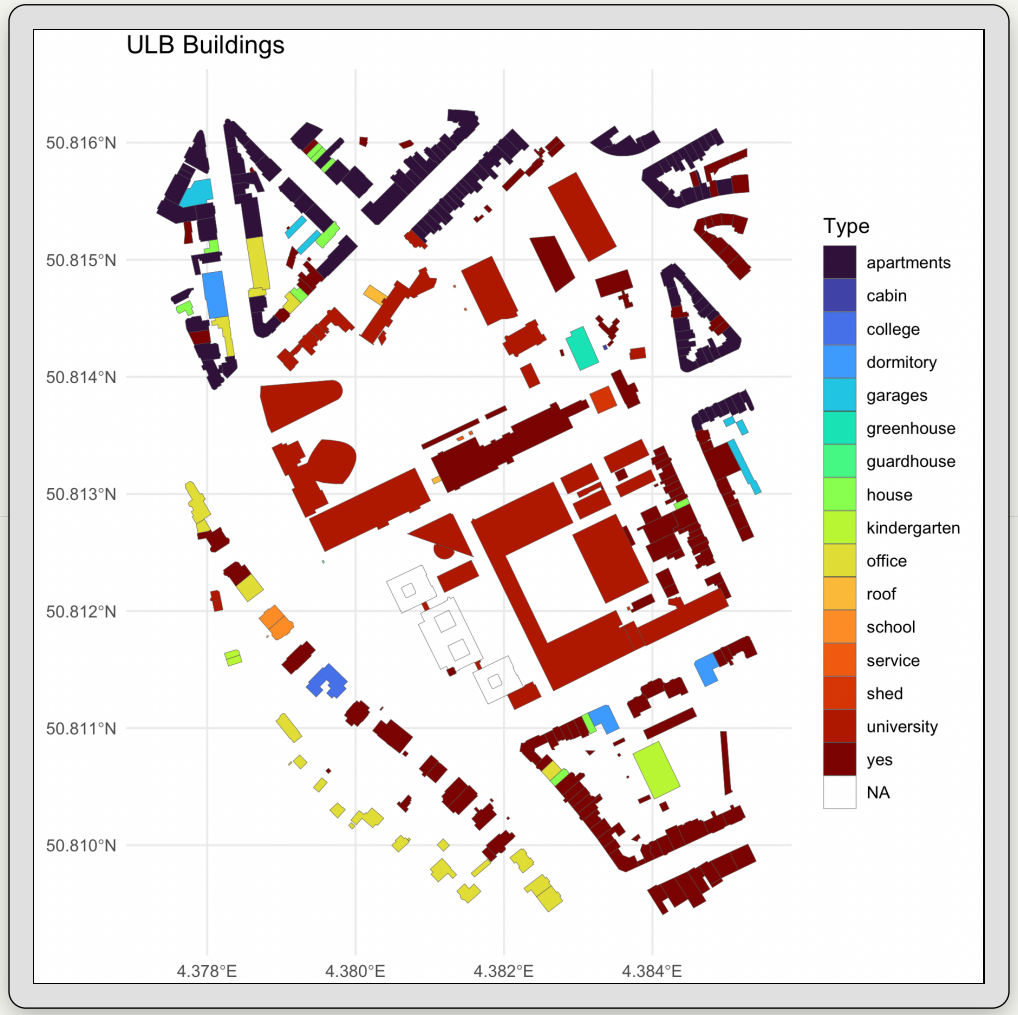
```
print(json.dumps(dict(zip(['lat-min', 'lat-max', 'lon-min', 'lon-max'], b))))

── R-repl                [run] cell-1
library(tidyverse)
library(sf)
library(osmdata)

bbox <- c(4.3777200,50.8096525,4.3851645,50.8159744)

opq(bbox) |>
  add_osm_feature(key = "building") |>
  osmdata_sf() |>
  pluck("osm_polygons") |>
  st_transform('EPSG:31370') |>
  ggplot() +
  geom_sf(aes(fill = building), color = "gray30", linewidth = 0.1) +
  scale_fill_viridis_d(option = "turbo") +
  theme_minimal() +
  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

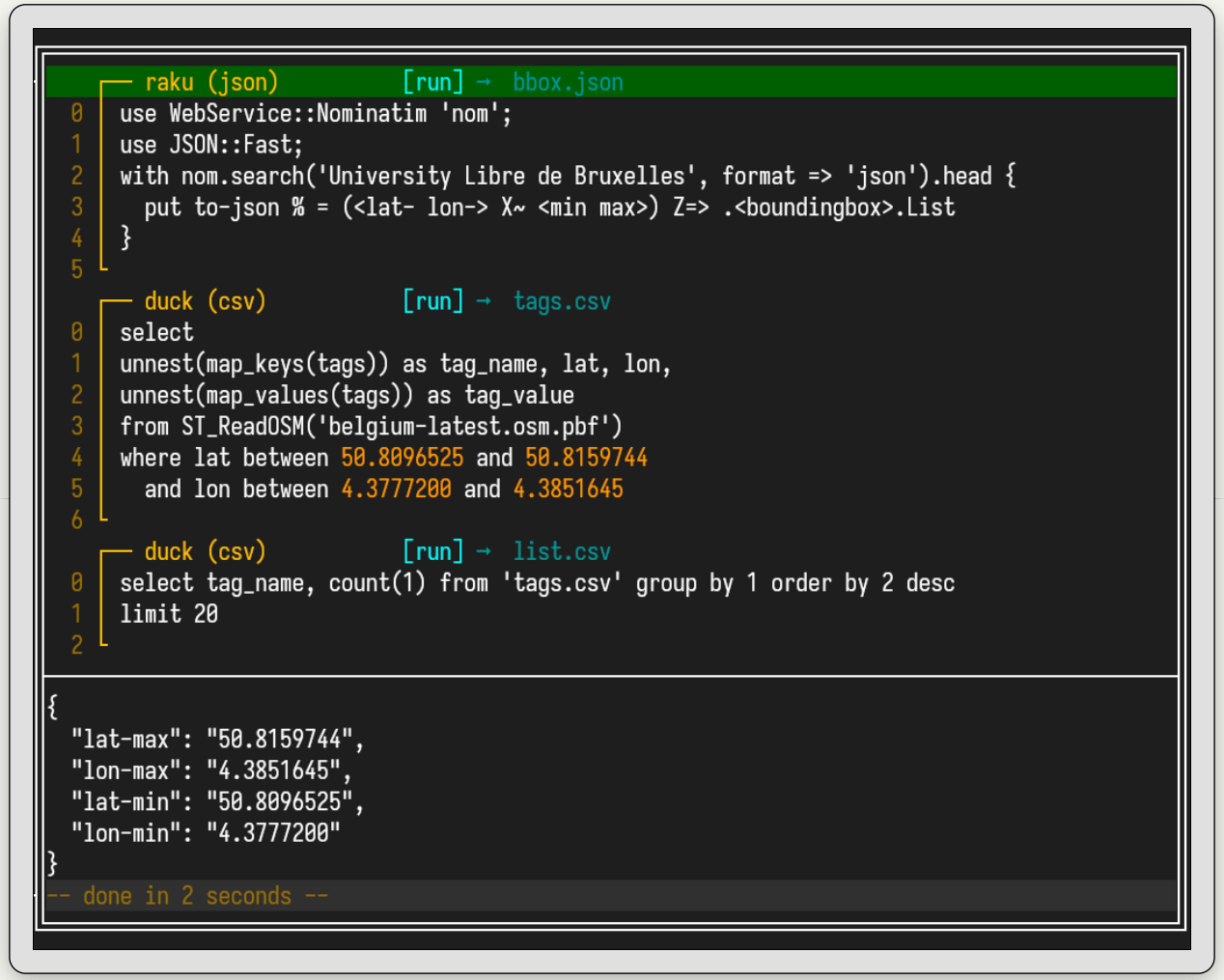
Example 1 : Python + R



Examples

Example 2 : bash + raku + duckdb

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



Examples

Example 2 : bash + raku + duckdb

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

~ tags ~

raku (json) [run] → bbox.json

```
use WebService::Nominatim 'nom';
use JSON::Fast;
with nom.search('University Libre de Bruxelles', format => 'json').head {
  put to-json % = (<lat- lon-> X~ <min max>) Z=> .<boundingbox>.List
}
```

duck (csv) [run] → tags.csv

```
select
unnest(map_keys(tags)) as tag_name, lat, lon,
unnest(map_values(tags)) as tag_value
from ST_ReadOSM('belgium-latest.osm.pbf')
where lat between 50.8096525 and 50.8159744
  and lon between 4.3777200 and 4.3851645
```

duck (csv) [run] → list.csv

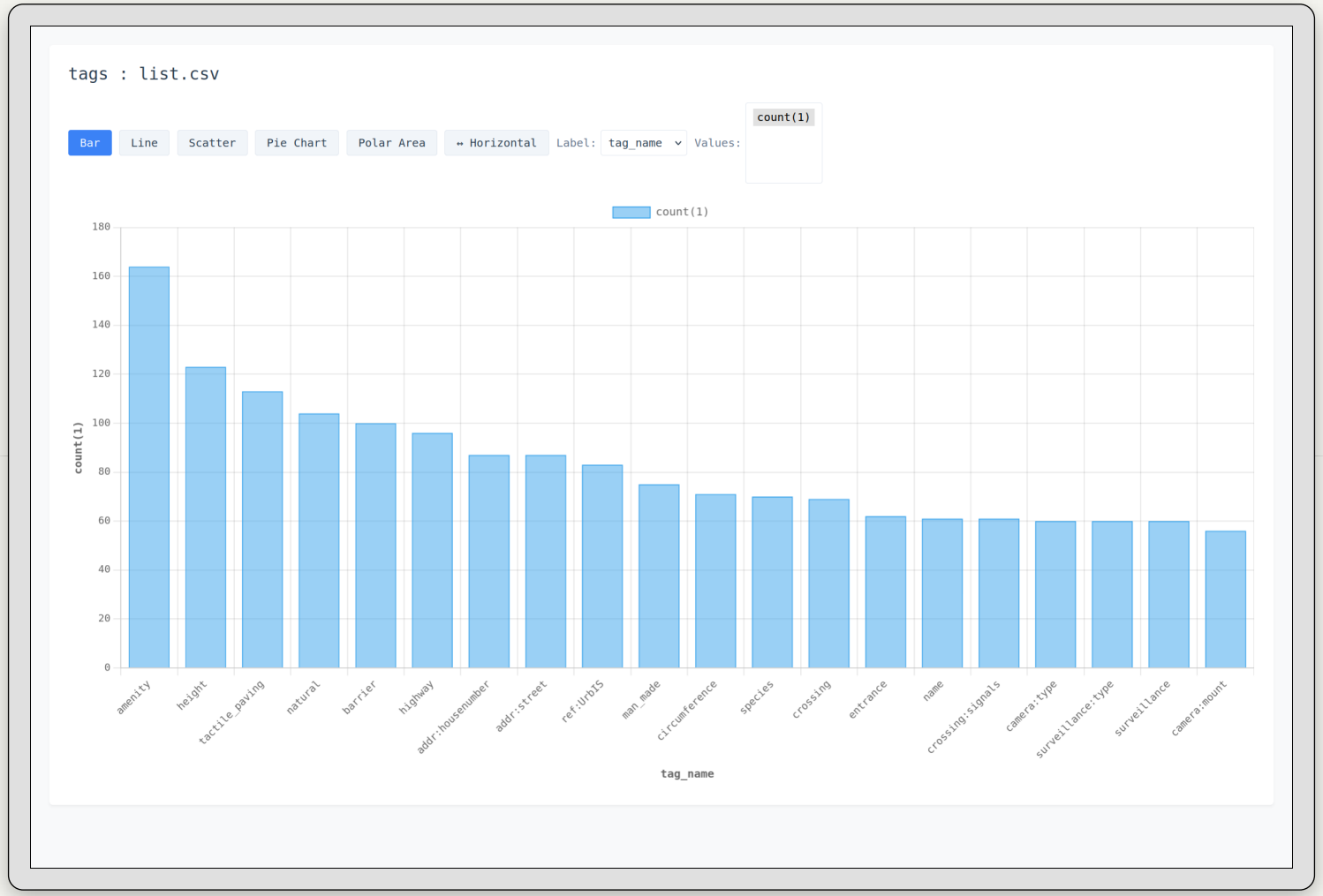
```
select tag_name, count(1) from 'tags.csv' group by 1 order by 2 desc
limit 20
```

```
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

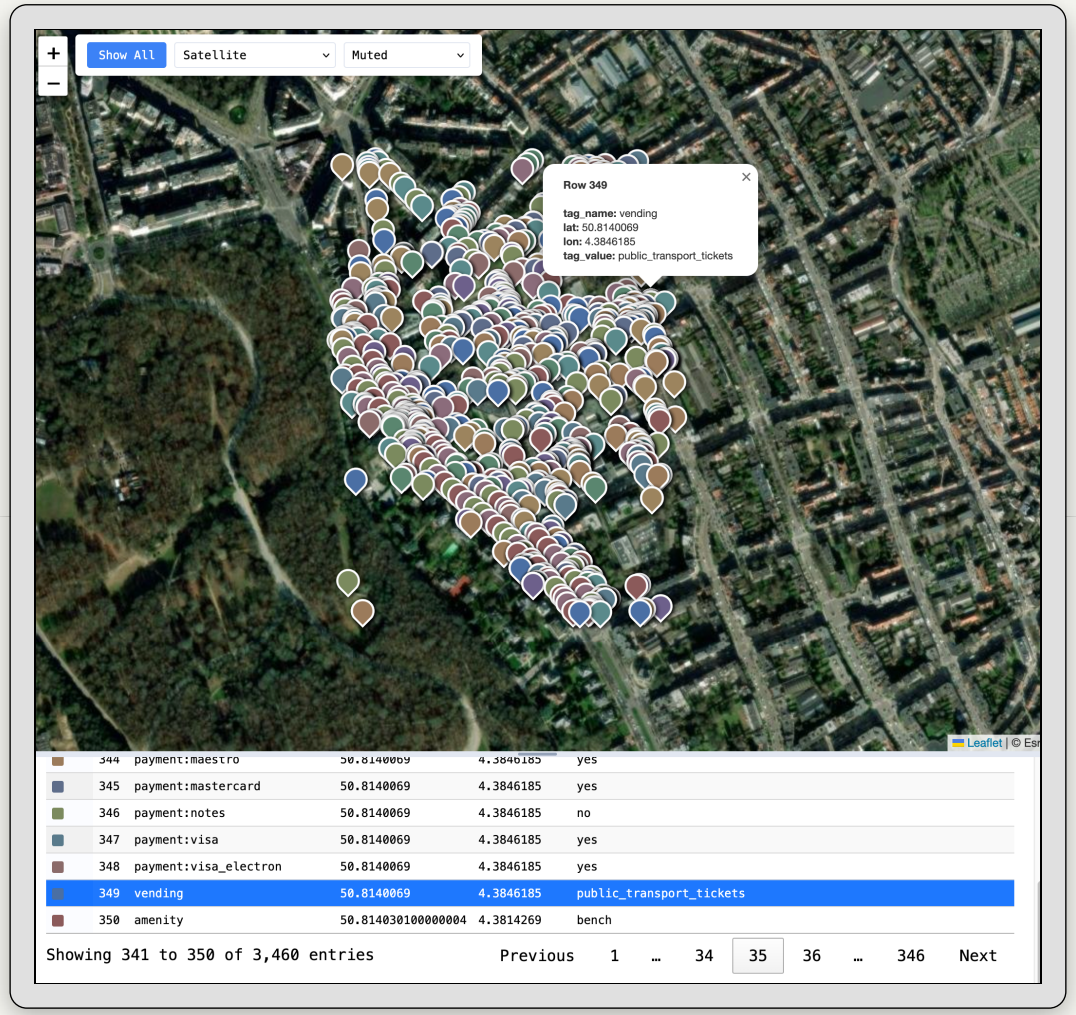
Example 2 : bash + raku + duckdb



- 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

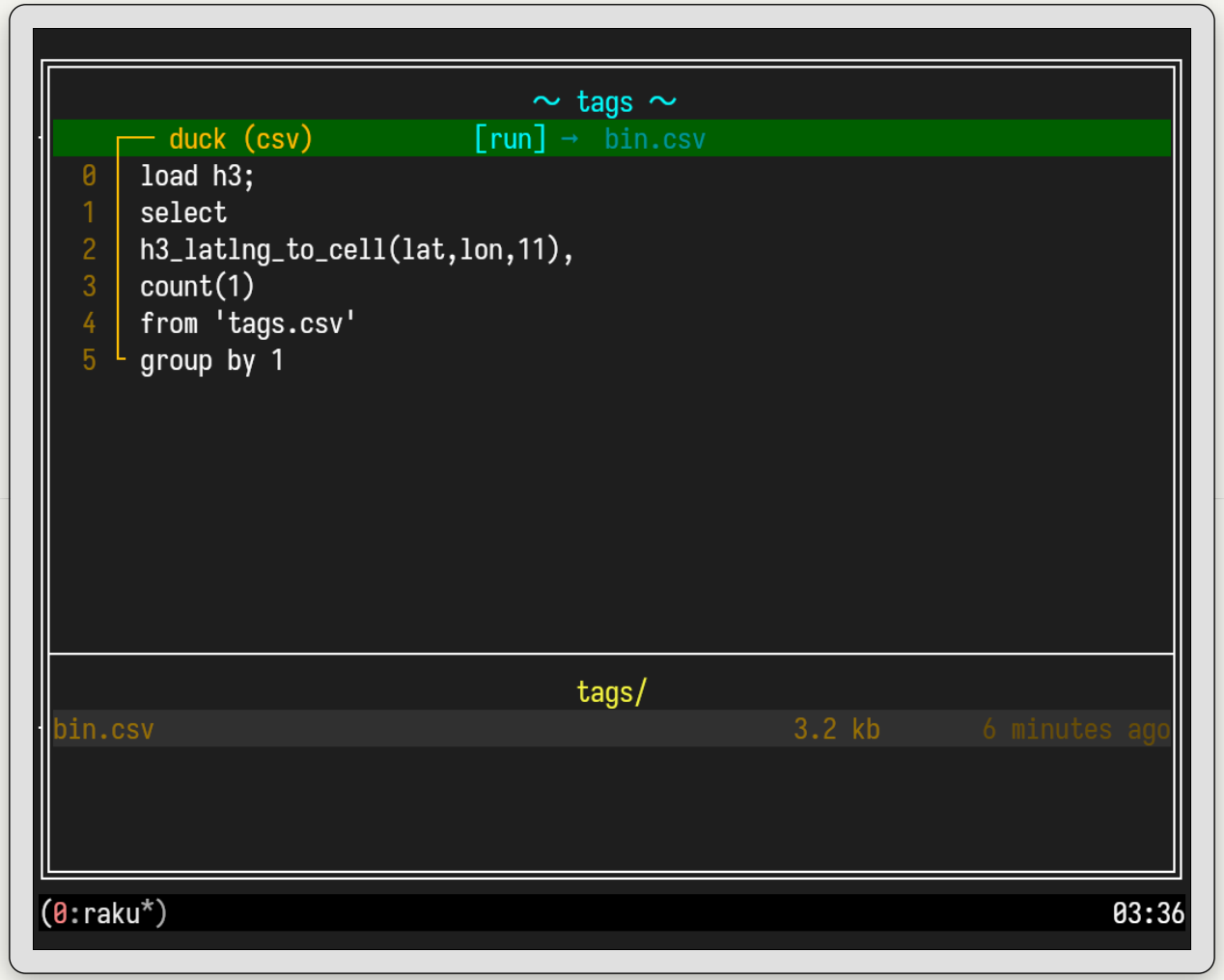
Example 2 : bash + raku + duckdb



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

Examples

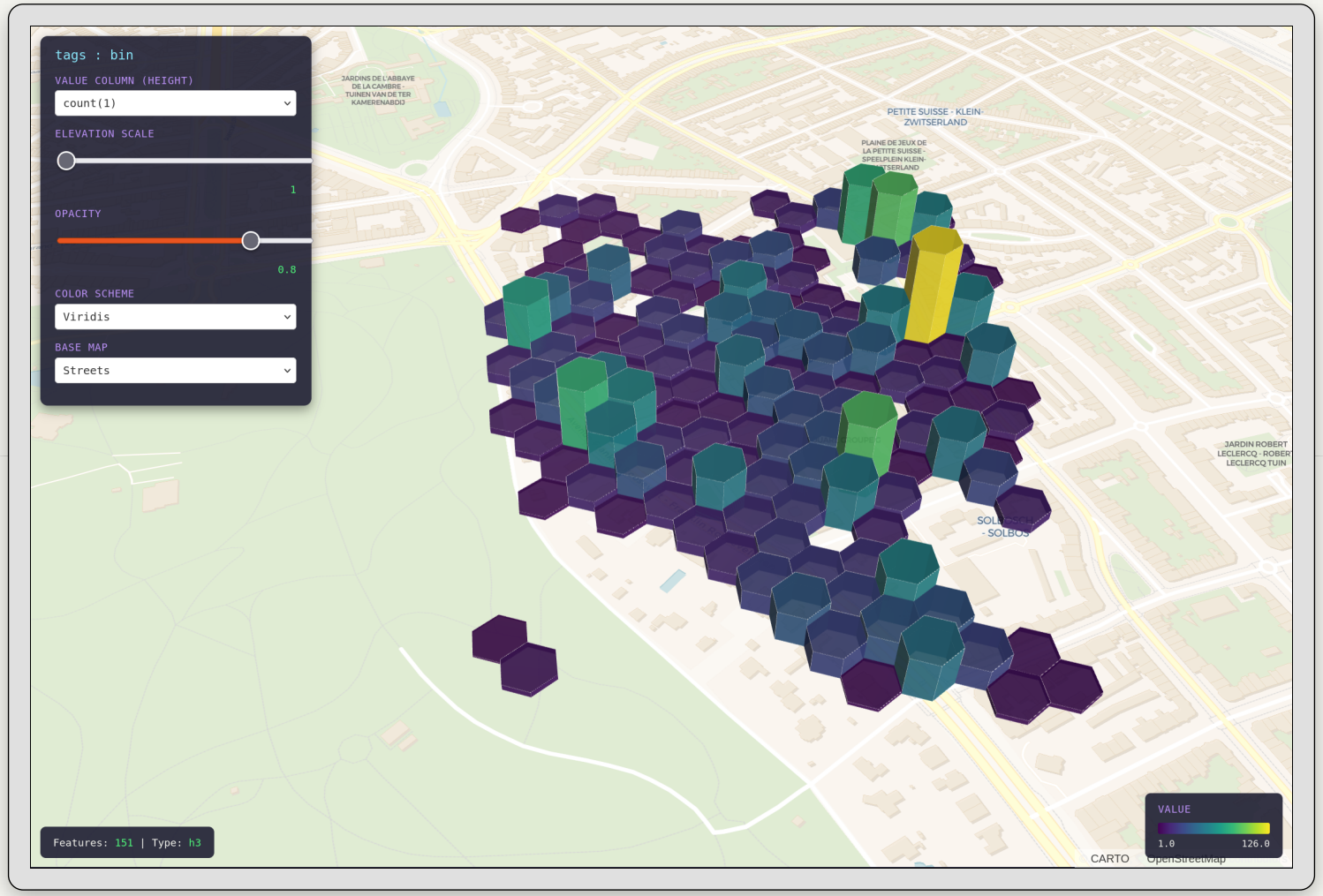
Example 2 : bash + raku + duckdb



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

Examples

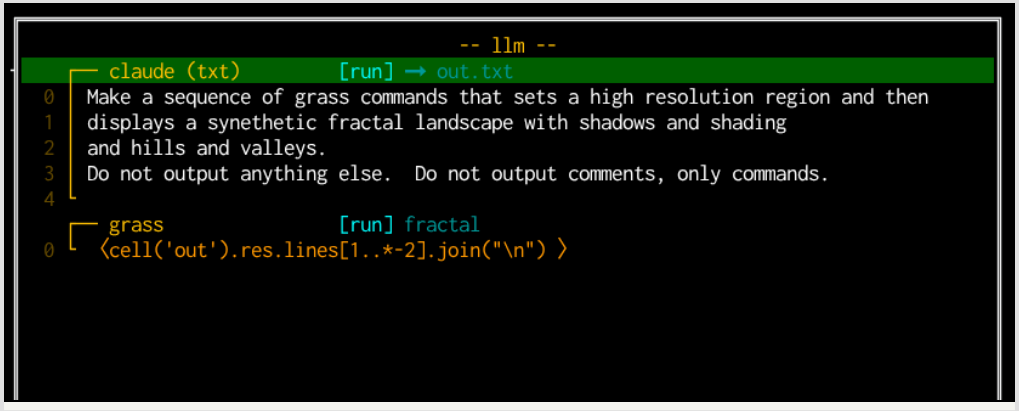
Example 2 : bash + raku + duckdb

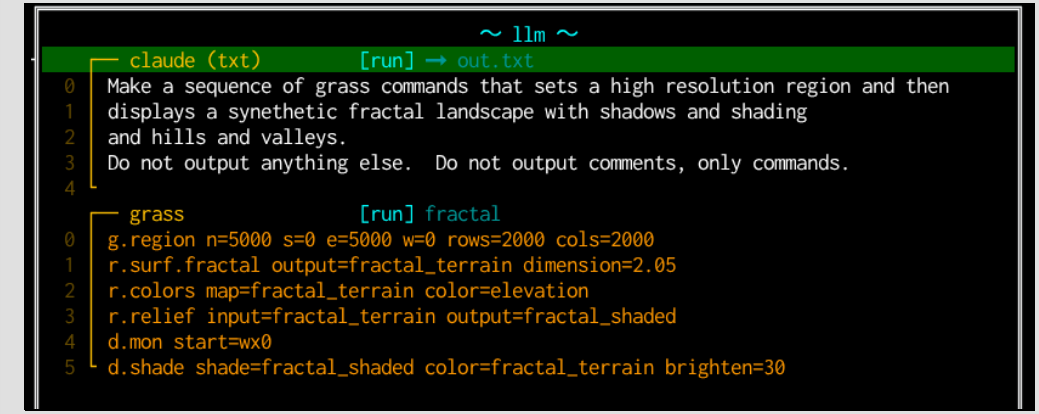


- The DeckGLBin plugout supports hex or geohash binning with Deck.gl
- by detecting H3 hex ids or geohash strings.

Examples

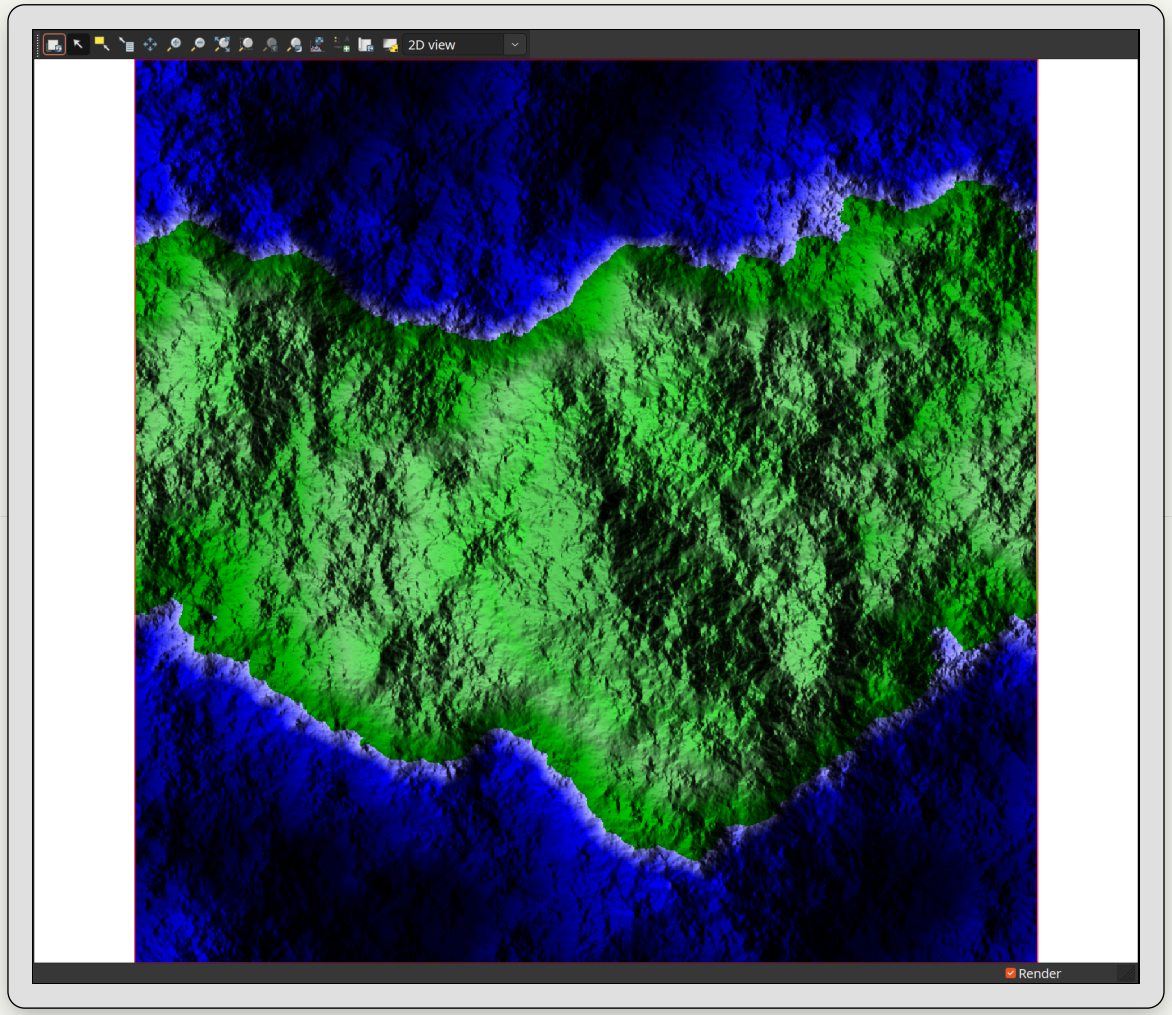
Example 3 : LLM + Grass





- 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.

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.

Extending

Interacting with other processes:

Plugin classes can inherit from common ones.

```
class Samaki::Plugin::Postgres does Samaki::Plugin::Pr
  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)
```

Raku

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 Jupytyer 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]

Conclusion

bduggan/raku-samaki

or search raku.land

or google for "raku samaki"

This presentation:

bduggan.github.io/raku-samaki

The end!