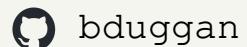


# **Keeping Spatial Scripting Sane**

**with Samaki and Raku**



by Brian Duggan



FOSDEM 2026

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

- Motivation
- Samaki
- Examples
- Extending
- Raku

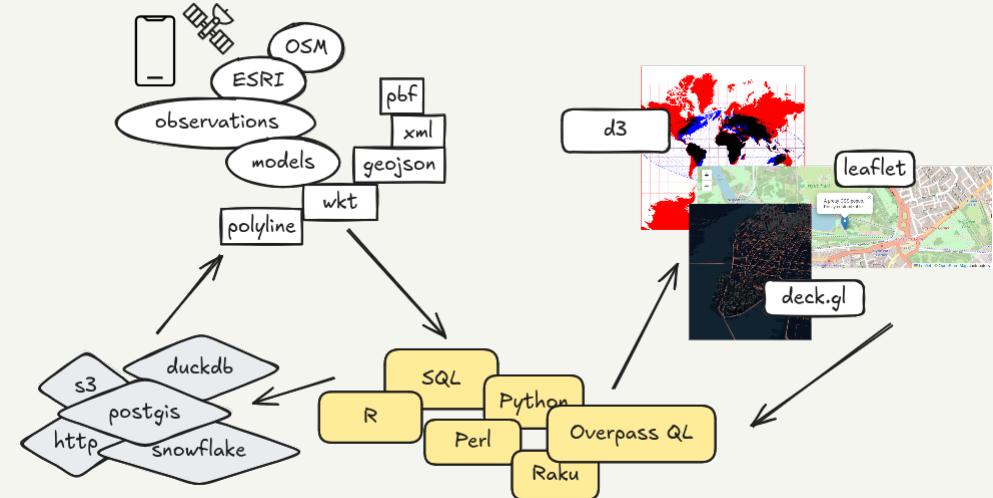
# Spatial scripting!

### 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, $lon) = 50.8134719, 4.3812357;
3
4 -- postgres
5 select
6   st_makewkt( < "$lon,$lat" > ),
7   st_buffer(
8     st_makewkt( < "$lon,$lat" > )::geography, 100
9   )
10 ~
~  
~  
~  
~  
~  
~  
~  
~  
~  
circle.samaki          10,0-1      All
```

## The console interface

A screenshot of a terminal window titled "Samaki". The window has a dark background with white text. It displays a PostgreSQL command to create a circle geometry and a file listing.

```
-- circle --
0 my ($lat, $lon) = 50.8134719, 4.3812357;
1
2   postgres (csv)      [run] → cell-1.csv
3
4 select
5   st_makeline( < "$lon,$lat" > ),
6   st_buffer(
7     st_makeline( < "$lon,$lat" > )::geography, 100
8   )
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
7010
7011
7012
7013
7014
7015
7016
7017
7018
7019
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7030
7031
7032
7033
7034
7035
7036
7037
7038
7039
7030
7031
7032
7033
7034
7035
7036
7037
7038
7039
7040
7041
7042
7043
7044
7045
7046
7047
7048
7049
7040
7041
7042
7043
7044
7045
7046
7047
7048
7049
7050
7051
7052
7053
7054
7055
7056
7057
7058
7059
7050
7051
7052
7053
7054
7055
7056
7057
7058
7059
7060
7061
7062
7063
7064
7065
7066
7067
7068
7069
7060
7061
7062
7063
7064
7065
7066
7067
7068
7069
7070
7071
7072
7073
7074
7075
7076
7077
7078
7079
7070
7071
7072
7073
7074
7075
7076
7077
7078
7079
7080
7081
7082
7083
7084
7085
7086
7087
7088
7089
7080
7081
7082
7083
7084
7085
7086
7087
7088
7089
7090
7091
7092
7093
7094
7095
7096
7097
7098
7099
7090
7091
7092
7093
7094
7095
7096
7097
7098
7099
7100
7101
7102
7103
7104
7105
7106
7107
7108
7109
7100
7101
7102
7103
7104
7105
7106
7107
7108
7109
7110
7111
7112
7113
7114
7115
7116
7117
7118
7119
7110
7111
7112
7113
7114
7115
7116
7117
7118
7119
7120
7121
7122
7123
7124
7125
7126
7127
7128
7129
7120
7121
7122
7123
7124
7125
7126
7127
7128
7129
7130
7131
7132
7133
7134
7135
7136
7137
7138
7139
7130
7131
7132
7133
7134
7135
7136
7137
7138
7139
7140
7141
7142
7143
7144
7145
7146
7147
7148
7149
7140
7141
7142
7143
7144
7145
7146
7147
7148
7149
7150
7151
7152
7153
7154
7155
7156
7157
7158
7159
7150
7151
7152
7153
7154
7155
7156
7157
7158
7159
7160
7161
7162
7163
7164
7165
7166
7167
7168
7169
7160
7161
7162
7163
7164
7165
7166
7167
7168
7169
7170
7171
7172
7173
7174
7175
7176
7177
7178
7179
7170
7171
7172
7173
7174
7175
7176
7177
7178
7179
7180
7181
7182
7183
7184
7185
7186
7187
7188
7189
7180
7181
7182
7183
7184
7185
7186
7187
7188
7189
7190
7191
7192
7193
7194
7195
7196
7197
7198
7199
7190
7191
7192
7193
7194
7195
7196
7197
7198
7199
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239
7240
7241
7242
7243
7244
7245
7246
7247
7248
7249
7240
7241
7242
7243
7244
7245
7246
7247
7248
7249
7250
7251
7252
7253
7254
7255
7256
7257
7258
7259
7250
7251
7252
7253
7254
7255
7256
7257
7258
7259
7260
7261
7262
7263
7264
7265
7266
7267
7268
7269
7260
7261
7262
7263
7264
7265
7266
7267
7268
7269
7270
7271
7272
7273
7274
7275
7276
7277
7278
7279
7270
7271
7272
7273
7274
7275
7276
7277
7278
7279
7280
7281
7282
7283
7284
7285
7286
7287
7288
7289
7280
7281
7282
7283
7284
7285
7286
7287
7288
7289
7290
7291
7292
7293
7294
7295
7296
7297
7298
7299
7290
7291
7292
7293
7294
7295
7296
7297
7298
7299
7300
7301
7302
7303
7304
7305
7306
7307
7308
7309
7300
7301
7302
7303
7304
7305
7306
7307
7308
7309
7310
7311
7312
7313
7314
7315
7316
7317
7318
7319
7310
7311
7312
7313
7314
7315
7316
7317
7318
7319
7320
7321
7322
7323
7324
7325
7326
7327
7328
7329
7320
7321
7322
7323
7324
7325
7326
7327
7328
7329
7330
7331
7332
7333
7334
7335
7336
7337
7338
7339
7330
7331
7332
7333
7334
7335
7336
7337
7338
7339
7340
7341
7342
7343
7344
7345
7346
7347
7348
7349
7340
7341
7342
7343
7344
7345
7346
7347
7348
7349
7350
7351
7352
7353
7354
7355
7356
7357
7358
7359
7350
7351
7352
7353
7354
7355
7356
7357
7358
7359
7360
7361
7362
7363
7364
7365
7366
7367
7368
7369
7360
7361
7362
7363
7364
7365
7366
7367
7368
7369
7370
7371
7372
7373
7374
7375
7376
7377
7378
7379
7370
7371
7372
7373
7374
7375
7376
7377
7378
7379
7380
7381
7382
7383
7384
7385
7386
7387
7388
7389
7380
7381
7382
7383
7384
7385
7386
7387
7388
7389
7390
7391
7392
7393
7394
7395
7396
7397
7398
7399
7390
7391
7392
7393
7394
7395
7396
7397
7398
7399
7400
7401
7402
7403
7404
7405
7406
7407
7408
7409
7400
7401
7402
7403
7404
7405
7406
7407
7408
7409
7410
7411
7412
7413
7414
7415
7416
7417
7418
7419
7410
7411
7412
7413
7414
7415
7416
7417
7418
7419
7420
7421
7422
7423
7424
7425
7426
7427
7428
7429
7420
7421
7422
7423
7424
7425
7426
7427
7428
7429
7430
7431
7432
7433
7434
7435
7436
7437
7438
7439
7430
7431
7432
7433
7434
7435
7436
7437
7438
7439
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7470
7471
7472
7473
7474
7475
7476
7477
7478
7479
7470
7471
7472
7473
7474
7475
7476
7477
7478
7479
7480
7481
7482
7483
7484
7485
7486
7487
7488
7489
7480
7481
7482
7483
7484
7485
7486
7487
7488
7489
7490
7491
7492
7493
7494
7495
7496
7497
7498
7499
7490
7491
7492
7493
7494
7495
7496
7497
7498
7499
7500
7501
7502
7503
7504
7505
7506
7507
7508
7509
7500
7501
7502
7503
7504
7505
7506
7507
7508
7509
7510
7511
7512
7513
7514
7515
7516
7517
7518
7519
7510
7511
7512
7513
7514
7515
7516
7517
7518
7519
7520
7521
7522
7523
7524
7525
7526
7527
7528
7529
7520
7521
7522
7523
7524
7525
7526
7527
7528
7529
7530
7531
7532
7533
7534
7535
7536
7537
7538
7539
7530
7531
7532
7533
7534
7535
7536
7537
7538
7539
7540
7541
7542
7543
7544
7545
7546
7547
7548
7549
7540
7541
7542
7543
7544
7545
7546
7547
7548
7549
7550
7551
7552
7553
7554
7555
7556
7557
7558
7559
7550
7551
7552
7553
7554
7555
7556
7557
7558
7559
7560
7561
7562
7563
7564
7565
7566
7567
7568
7569
7560
7561
7562
7563
7564
7565
7566
7567
7568
7569
7570
7571
7572
7573
7574
7575
7576
7577
7578
7579
7570
7571
7572
7573
7574
7575
7576
7577
7578
7579
7580
7581
7582
7583
7584
7585
7586
7587
7588
7589
7580
7581
7582
7583
7584
7585
7586
7587
7588
7589
7590
7591
7592
7593
7594
7595
7596
7597
7598
7599
7590
7591
7592
7593
7594
7595
7596
7597
7598
7599
7600
7601
7602
7603
7604
7605
7606
7607
7608
7609
7600
7601
7602
7603
7604
7605
7606
7607
7608
7609
7610
7611
7612
7613
7614
7615
7616
7617
7618
7619
7610
7611
7612
7613
7614
7615
7616
7617
7618
7619
7620
7621
7622
7623
7624
7625
7626
7627
7628
7629
7620
7621
7622
7623
7624
7625
7626
7627
7628
7629
7630
7631
7632
7633
7634
7635
7636
7637
7638
7639
7630
7631
7632
7633
7634
7635
7636
7637
7638
7639
7640
7641
7642
7643
7644
7645
7646
7647
7648
7649
7640
7641
7642
7643
7644
7645
7646
7647
7648
7649
7650
7651
7652
7653
7654
7655
7656
7657
7658
7659
7650
7651
7652
7653
7654
7655
7656
7657
7658
7659
7660
7661
7662
7663
7664
7665
7666
7667
7668
7669
7660
7661
7662
7663
7664
7665
7666
7667
7668
7669
7670
7671
7672
7673
7674
7675
7676
7677
7678
7679
7670
7671
7672
7673
7674
7675
7676
7677
7678
7679
7680
7681
7682
7683
7684
7685
7686
7687
7688
7689
7680
7681
7682
7683
7684
7685
7686
7687
7688
7689
7690
7691
7692
7693
7694
7695
7696
7697
7698
7699
7690
7691
7692
7693
7694
7695
7696
7697
7698
7699
7700
7701
7702
7703
7704
7705
7706
7707
7708
7709
7700
7701
7702
7703
7704
7705
7706
7707
7708
7709
7710
7711
7712
7713
7714
7715
7716
7717
7718
7719
7710
7711
7712
7713
7714
7715
7716
7717
7718
7719
7720
7721
7722
7723
7724
7725
7726
7727
7728
7729
7720
7721
7722
7723
7724
7725
7726
7727
7728
7729
7730
7731
7732
7733
7734
7735
7736
7737
7738
7739
7730
7731
7732
7733
7734
7735
7736
7737
7738
7739
7740
7741
7742
7743
7744
7745
7746
7747
7748
7749
7740
7741
7742
7743
7744
7745
7746
7747
7748
7749
7750
7751
7752
7753
7754
7755
7756
7757
7758
7759
7750
7751
7752
7753
7754
7755
7756
7757
7758
7759
7760
7761
7762
7763
7764
7765
7766
7767
7768
7769
7760
7761
7762
7763
7764
7765
7766
7767
7768
7769
7770
7771
7772
7773
7774
7775
7776
7777
7778
7779
7770
7771
7772
7773
7774
7775
7776
7777
7778
7779
7780
7781
7782
7783
7784
7785
7786
7787
7788
7789
7780
7781
7782
7783
7784
7785
7786
7787
7788
7789
7790
7791
7792
7793
7794
7795
7796
7797
7798
7799
7790
7791
7792
7793
7794
7795
7796
7797
7798
7799
7800
7801
7802
7803
7804
7805
7806
7807
7808
7809
7800
7801
7802
7803
7804
7805
7806
7807
7808
7809
7810
7811
7812
7813
7814
7815
7816
7817
7818
7819
7810
7811
7812
7813
7814
7815
7816
7817
7818
7819
7820
7821
7822
7823
7824
7825
7826
7827
7828
7829
7820
7821
7822
7823
7824
7825
7826
7827
7828
7829
7830
7831
7832
7833
7834
7835
7836
7837

```

## The console interface

The screenshot shows a terminal window with a dark background. At the top, it says `~ circle ~`. Below that, a PostgreSQL command is entered:

```
0 [my ($lat, $lon) = 50.8134719, 4.3812357;
1
2   postgres (csv)      [run] → cell-1.csv
3   select
4     st_makeline( 4.3812357,50.8134719 ),
5     st_buffer(
6       st_makeline( 4.3812357,50.8134719)::geography, 100
7     )
8 ]
```

The command creates a circle geometry and saves it to a CSV file named `cell-1.csv` in a directory named `circle/`. The output shows:

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

1 row, 2 columns

st_makewkt          st_buffer
01010000004B45BEA662861140855BE3D81F684940 010300020E61000000
```

Select cells to run them

**The console interface**

~ circle ~

```
0 [ my ($lat, $lon) = 50.8134719, 4.3812357;
1
2   postgres (csv)      [run] → cell-1.csv
3
4   select
5     st_makepoint( 4.3812357,50.8134719 ),
6     st_buffer(
7       st_makepoint( 4.3812357,50.8134719)::geography, 100
8     )
9
```

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

Select output files to view them

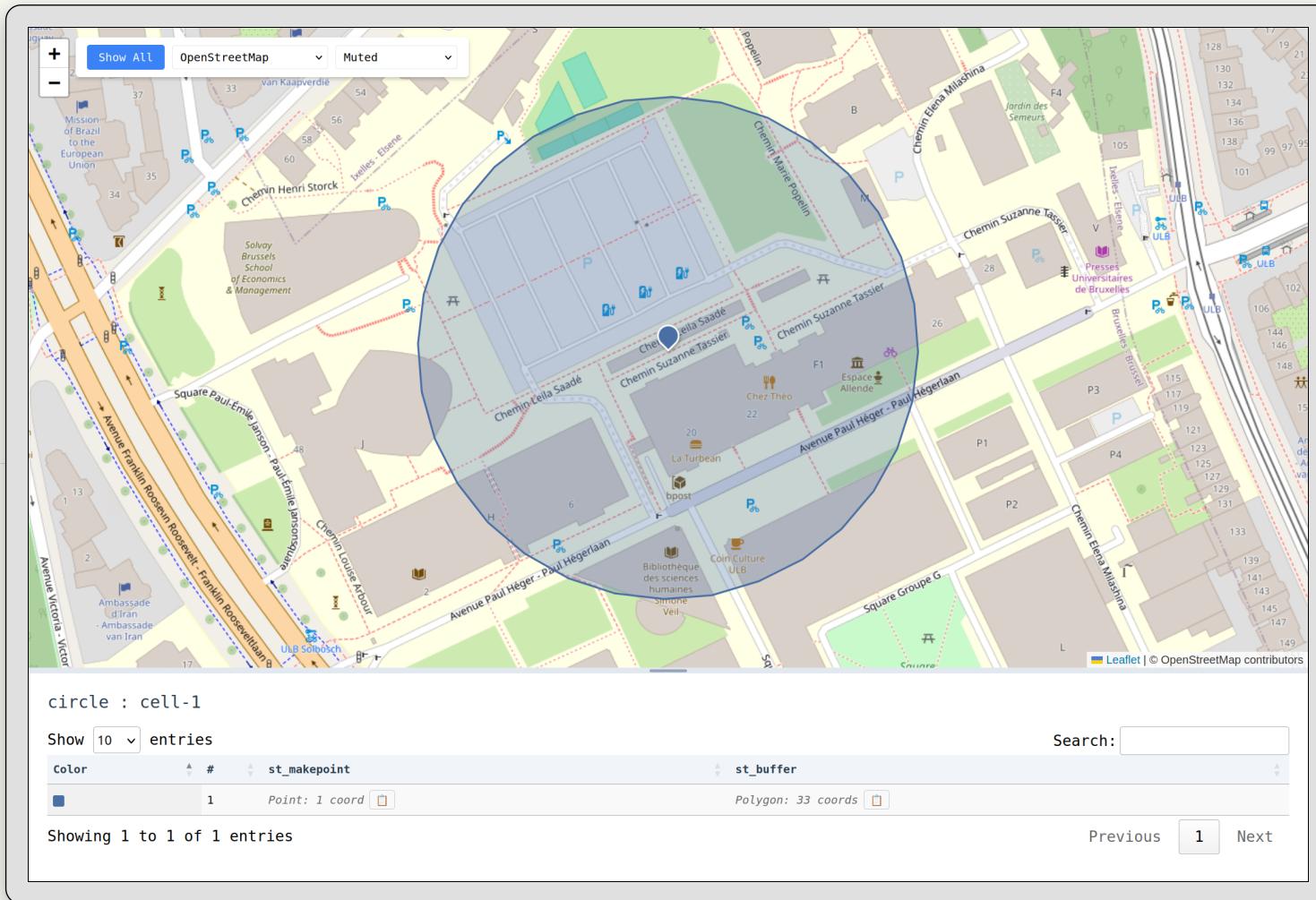
**The console interface**

```
~ circle ~

0 [ my ($lat, $lon) = 50.8134719, 4.3812357;
1
2   postgres (csv)      [run] → cell-1.csv
3
4   select
5     st_makeline( 4.3812357,50.8134719 ),
6     st_buffer(
7       st_makeline( 4.3812357,50.8134719)::geography, 100
8     )
9

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

All plugouts are available to view the file.



CSVGeo is a plugin that 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

The screenshot shows a terminal window with two code snippets. The top snippet is Raku code:

```
-- circle-too --
raku (csv)      [run] → lonlat.csv
0 use WebService::Nominatim 'nom';
1 say nom.search('University Libre de Brussels')[0]<lon lat>.join(',');
2
postgres (csv)   [run] → cell-1.csv
0 select
1   st_makepoint( < c('lonlat').out > ),
2   st_buffer(
3     st_makepoint( < c('lonlat').out > )::geography, 100
4   )
5
```

The bottom part of the terminal shows a file listing:

	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, `s9/42` numbers ( e.g. `c(0)` )

## Examples

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

The screenshot shows a map of a residential area in Brussels, Belgium. A blue circle, representing a buffer, is centered on a point labeled "cell-1". The map includes street names like Avenue Franklin Roosevelt, Avenue Paul Héger - Paul Hégerlaan, and Chemin Suzanne Tassier. Buildings are represented by grey shapes, and green areas indicate parks or gardens. A legend at the bottom left shows a blue square next to the text "circle : cell-1". Below the map, there is a search bar and a table with one entry:

Color	#	st_makepoint	st_buffer
■	1	Point: 1 coord	Polygon: 33 coords

Below the table, it says "Showing 1 to 1 of 1 entries". At the bottom right, there are buttons for "Previous", "1", and "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() |>
pyr/
bbox.json          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

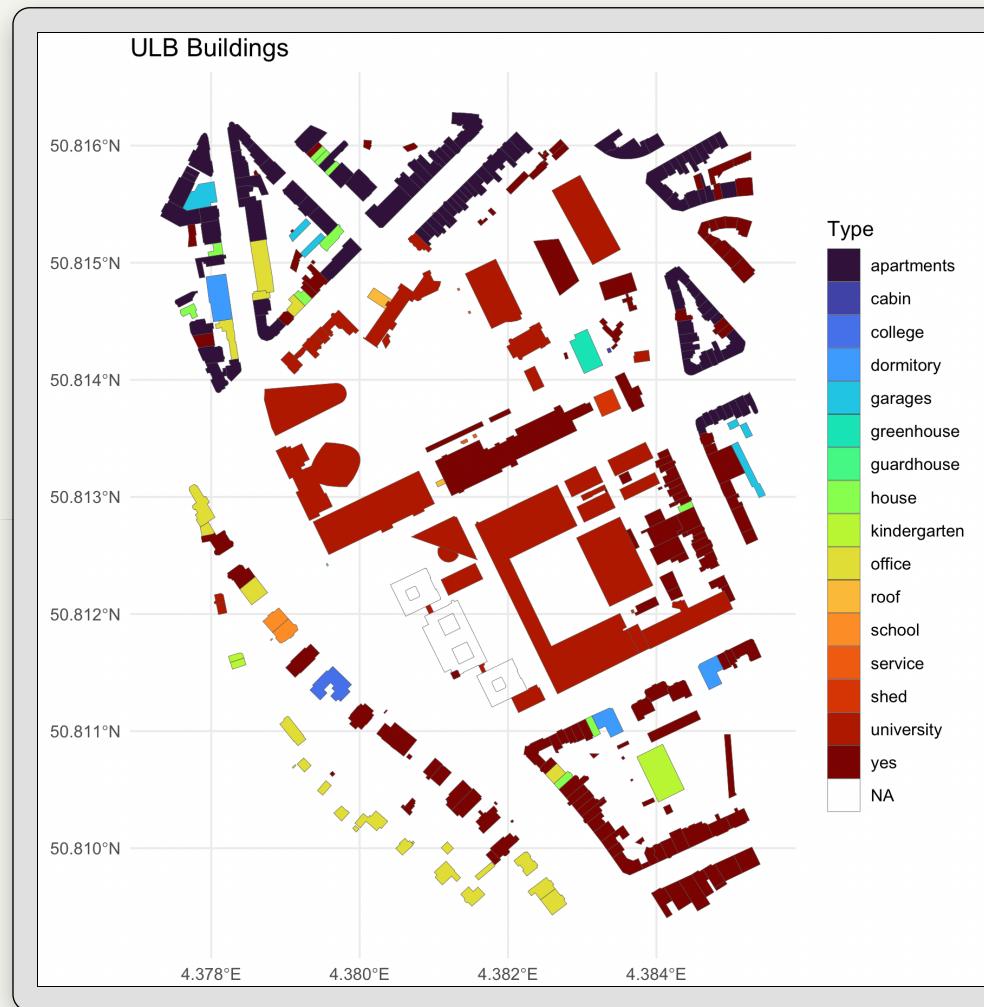
```
5 | print(json.dumps(dict(zip(['lat-min', 'lat-max', 'lon-min', 'lon-max'], b))))
6 |
R-repl      [run] cell-1
0 | library(tidyverse)
1 | library(sf)
2 | library(osmdata)
3 |
4 | bbox <- c(4.3777200,50.8096525,4.3851645,50.8159744)
5 |
6 | opq(bbox) |>
7 |   add_osm_feature(key = "building") |>
8 |   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

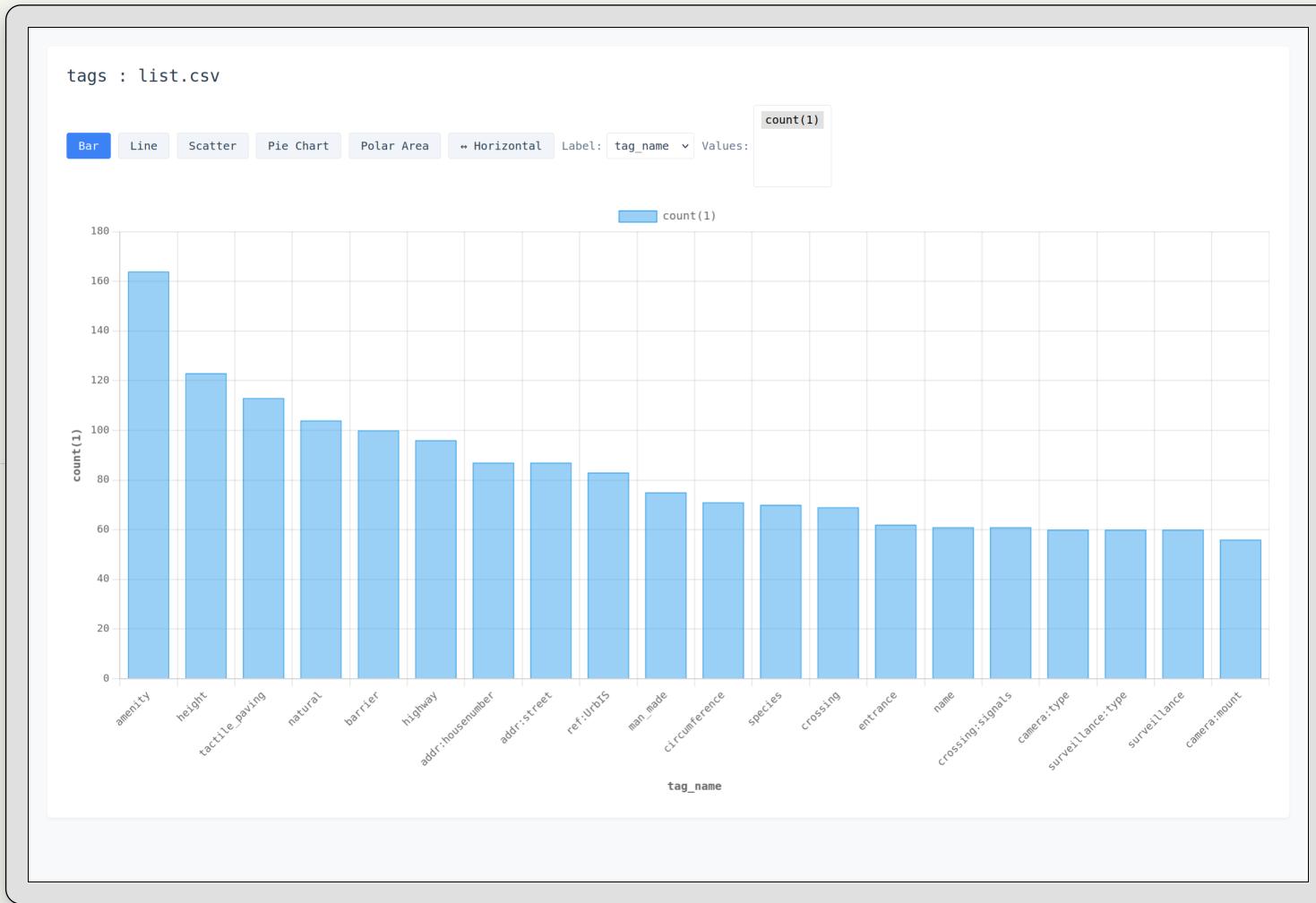
```
~ tags ~
0 [raku (json)] [run] → bbox.json
1 use WebService::Nominatim 'nom';
2 use JSON::Fast;
3 with nom.search('University Libre de Bruxelles', format => 'json').head {
4   put to-json % = (<lat- lon-> X~ <min max>) Z=> .<boundingbox>.List
5 }
6
0 [duck (csv)] [run] → tags.csv
1 select
2 unnest(map_keys(tags))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

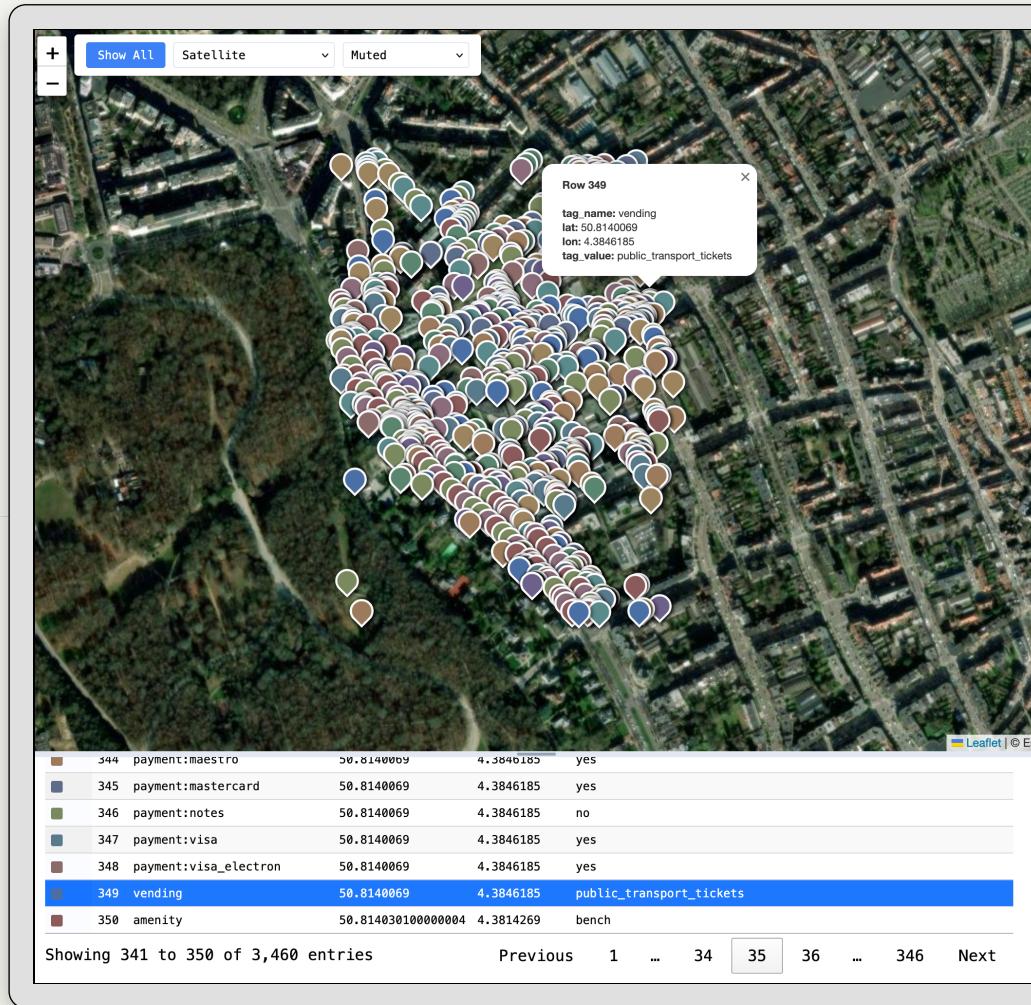
### Example 2 : bash + raku + duckdb



- 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

The terminal window shows the following session:

```
~ 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

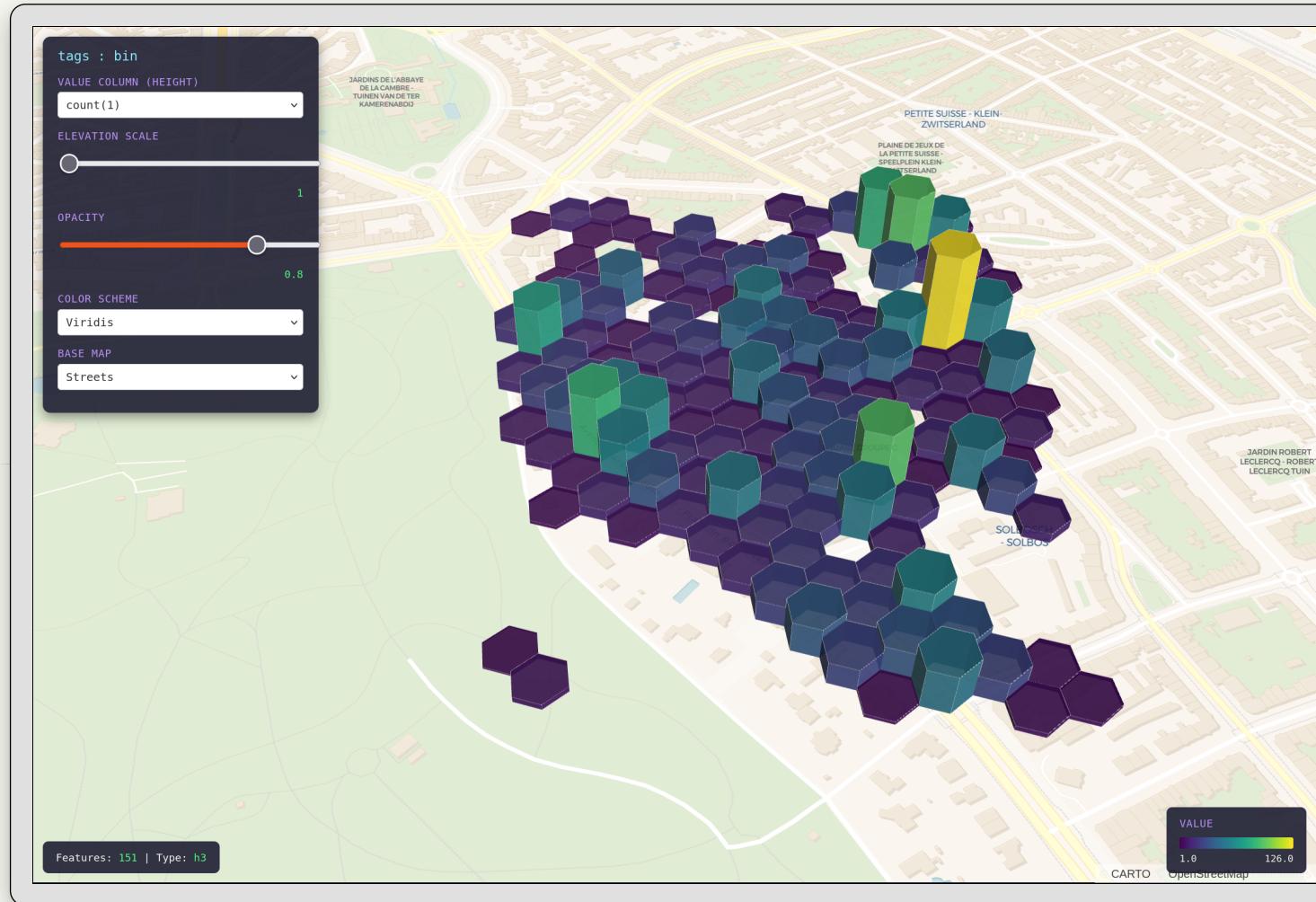
tags/
bin.csv          3.2 kb    6 minutes ago
(0:raku*)        03:36
```

The Raku code performs a query on a DuckDB database named 'duck (csv)'. The query loads the 'h3' module, selects data from 'tags.csv', and groups it by location. The resulting CSV file, 'bin.csv', is generated and saved in the current directory.

- 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

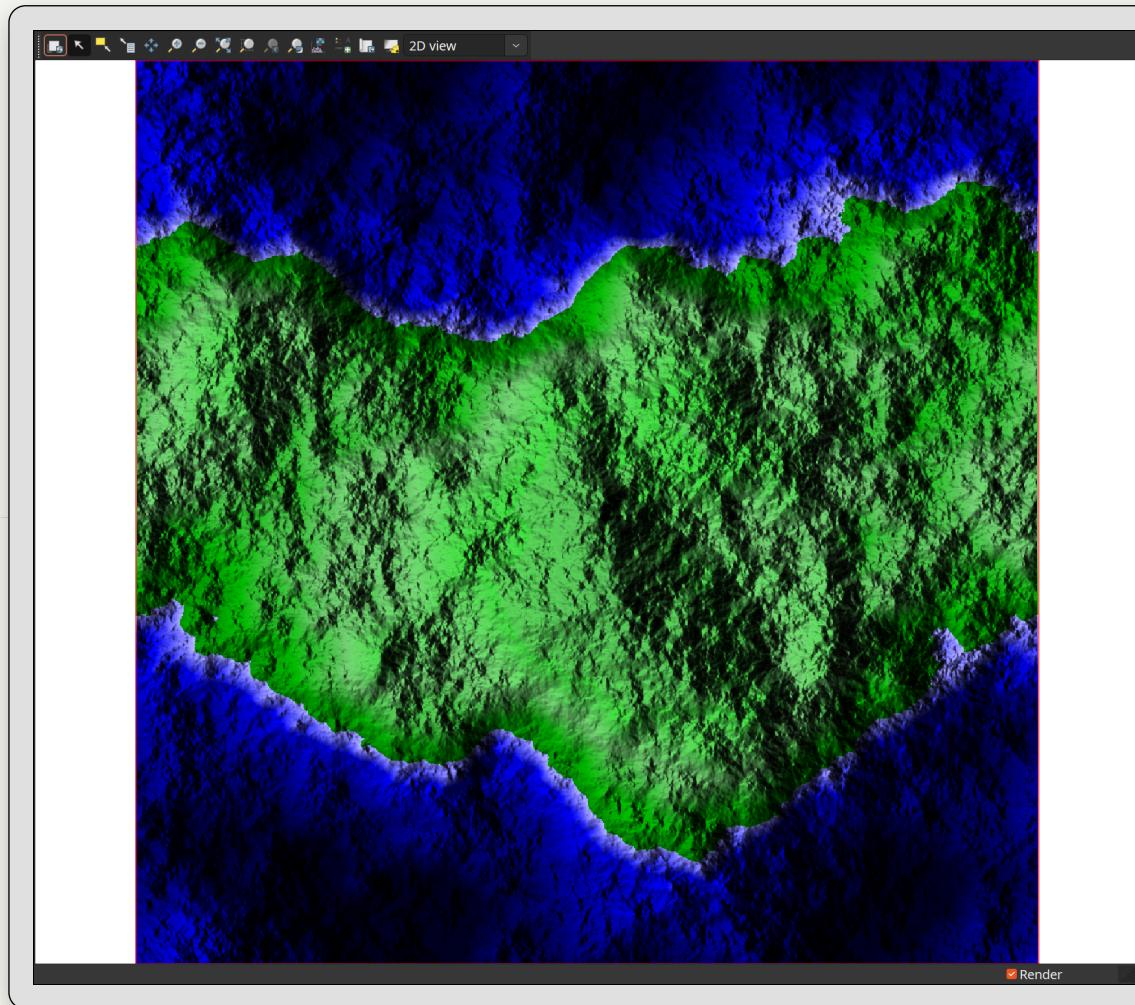
```
-- llm --
claude (txt) [run] → out.txt
0 Make a sequence of grass commands that sets a high resolution region and then
1 displays a synthetic fractal landscape with shadows and shading
2 and hills and valleys.
3 Do not output anything else. Do not output comments, only commands.
4
grass [run] fractal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

```
~ llm ~
claude (txt) [run] → out.txt
0 Make a sequence of grass commands that sets a high resolution region and then
1 displays a synthetic fractal landscape with shadows and shading
2 and hills and valleys.
3 Do not output anything else. Do not output comments, only commands.
4
grass [run] fractal
0 g.region n=5000 s=0 e=5000 w=0 rows=2000 cols=2000
1 r.surf.fractal output=fractal_terrain dimension=2.05
2 r.colors map=fractal_terrain color=elevation
3 r.relief input=fractal_terrain output=fractal_shaded
4 d.mon start=wx0
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::Process {
    has $.cmd = 'psql';
}

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

- Use **Samaki::Plugin::Process** for simple stdio/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](https://bduggan.github.io/raku-samaki)

The end!



