Gateway Action Set

Parallelizing Open Source code

Kristoffer Nilsson

Why Gateway?

- Need *super fast* data transfer from SAS to Open Source

- Make Python/R viable for not just scripting, but for big data calculation
  - Customers and internal developers can potentially use R/Python as an alternative to DS and DS2.

- Attractive to the giant developer base

- And their cutting-edge libraries

- And preferred IDEs

How is it any different?

Gateway features

Use Python and R for Big Data

- Spawn many Python or R processes
- Run each process once

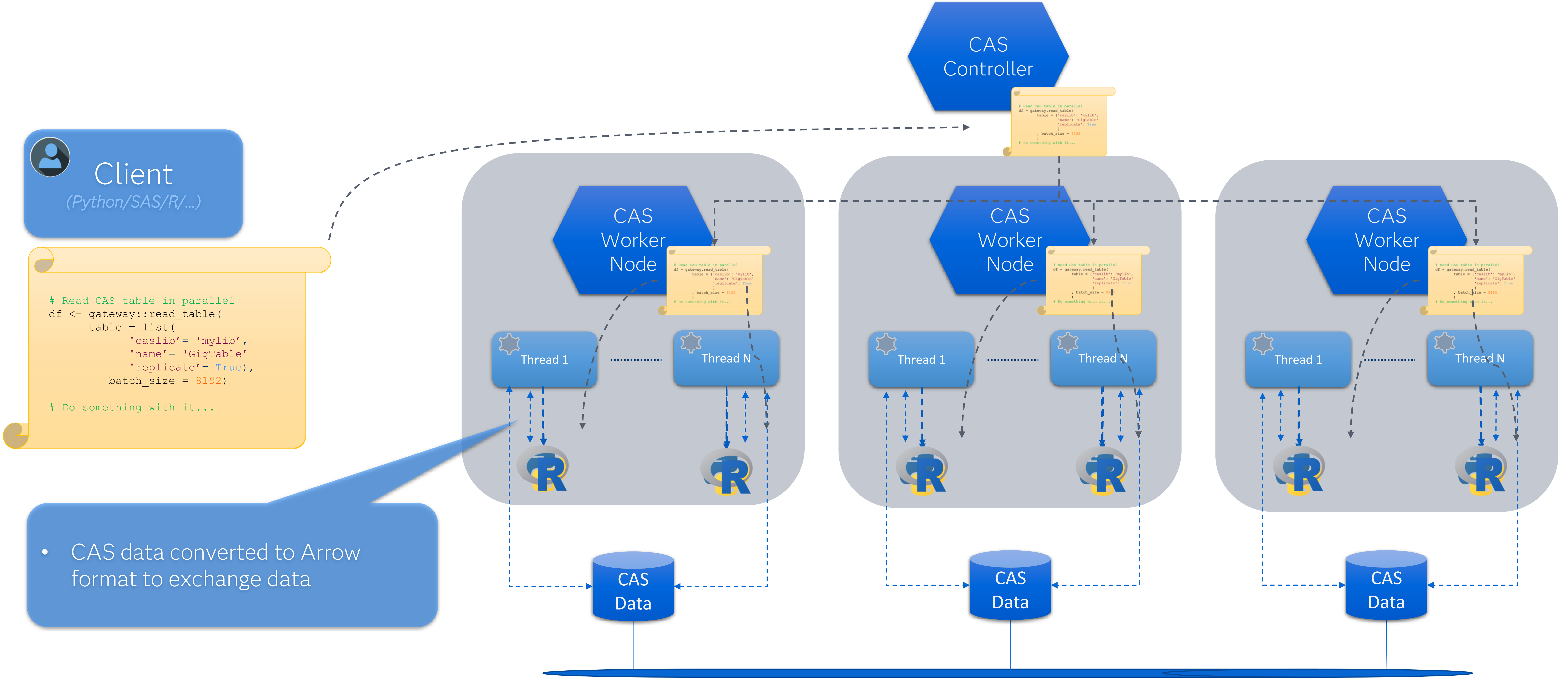
Read and Write in-memory data quickly

- Allow to process data in batches
- Use Arrow format if required
- Interacts only with CAS ⚠

You can run native Python/R libraries

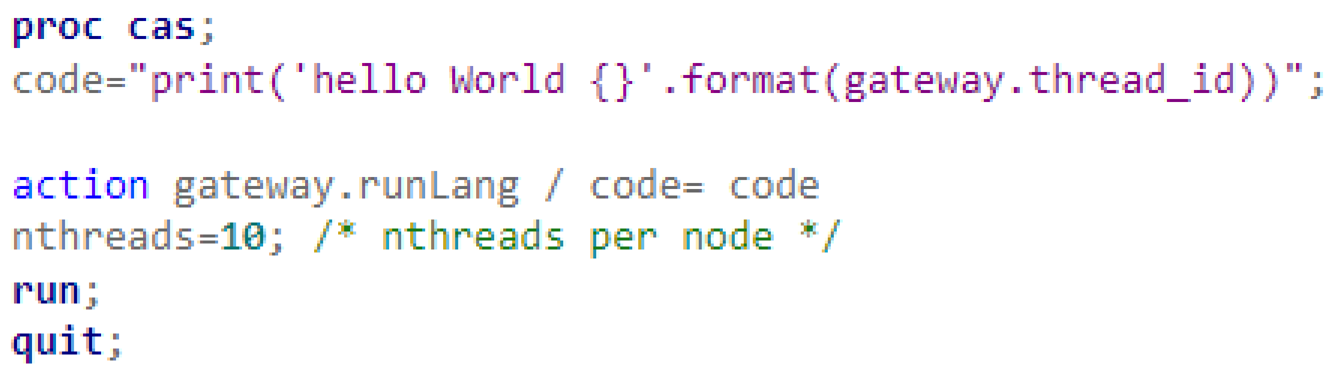
- It will not magically transform single Machine/Thread algorithms in distributed algorithms (e.g.: sklearn) ⚠
- The user must deal with distributed data processing logic

Gateway flow

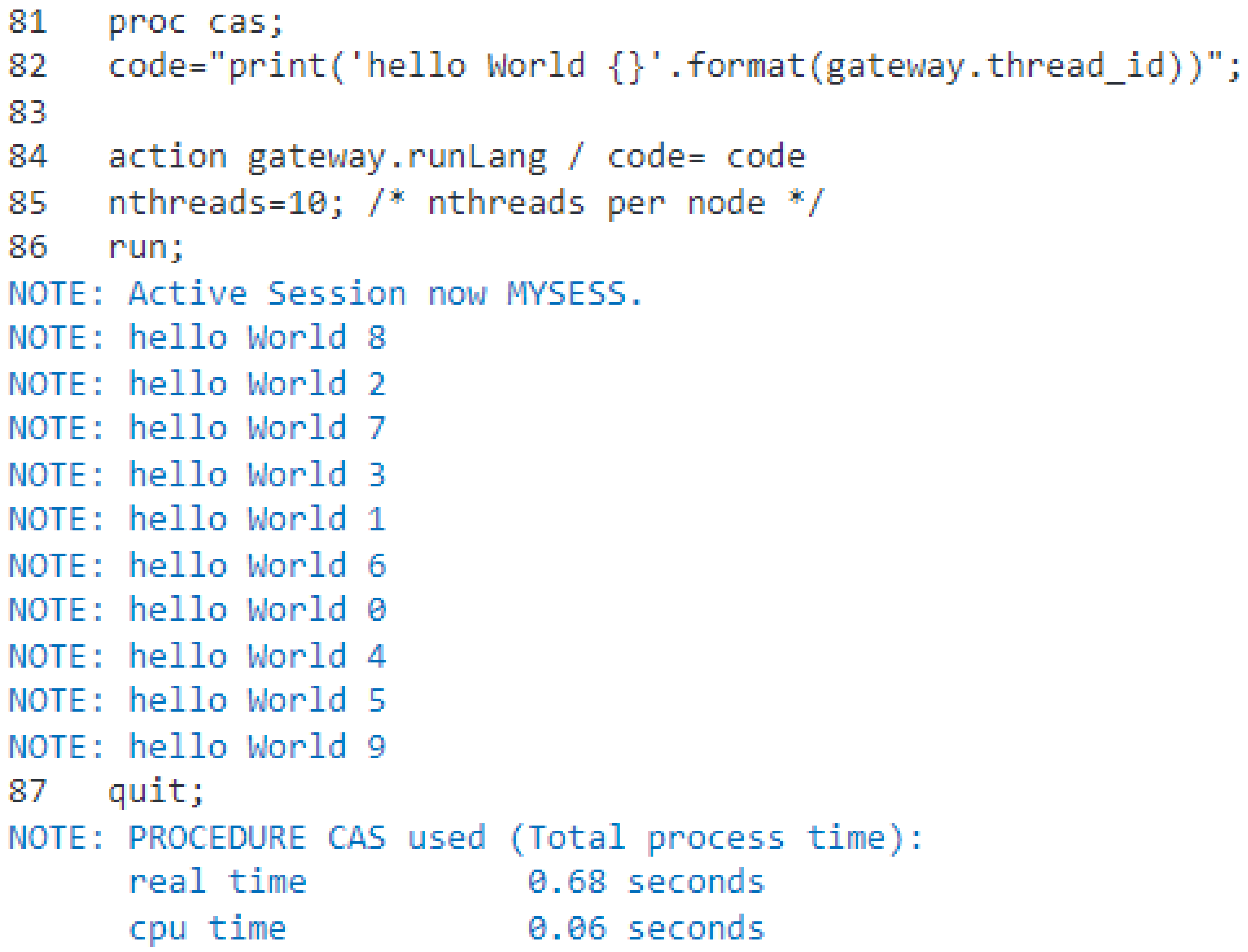


```
# Read CAS table in parallel
df <- gateway::read_table(
      table = list(
            'caslib’= 'mylib’,
            'name’= 'GigTable’
            'replicate’= True),
         batch_size = 8192)

# Do something with it...
```

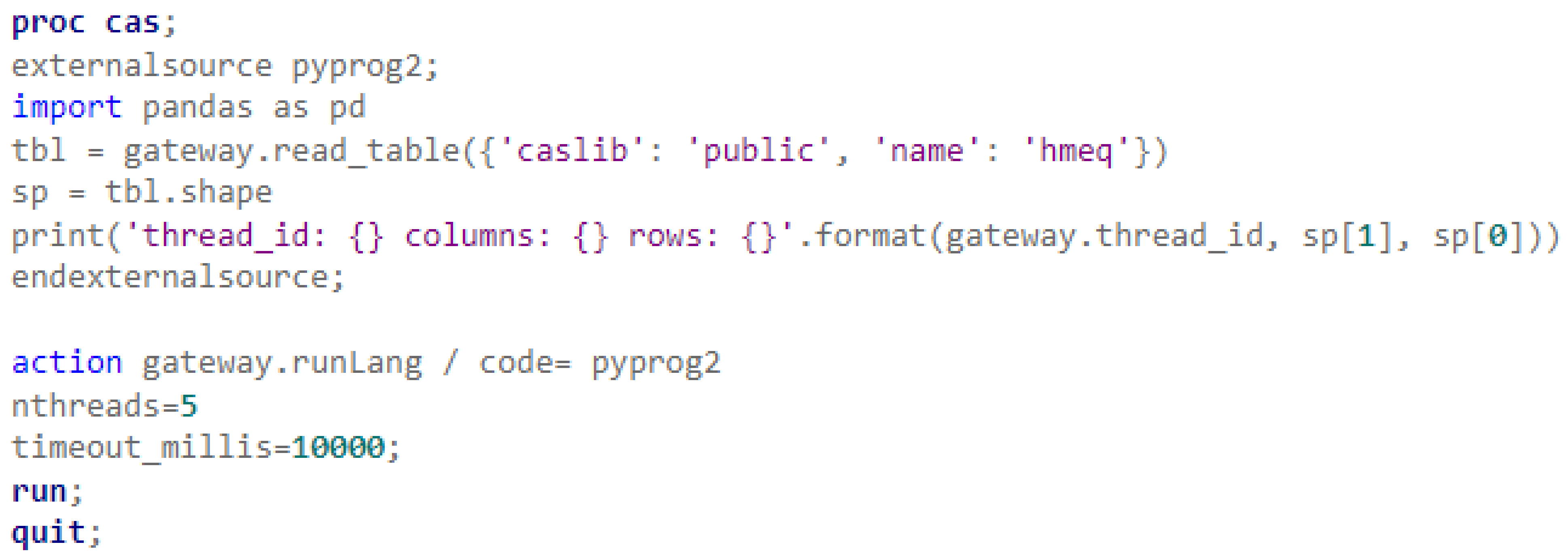
- CAS data converted to Arrow format to exchange data



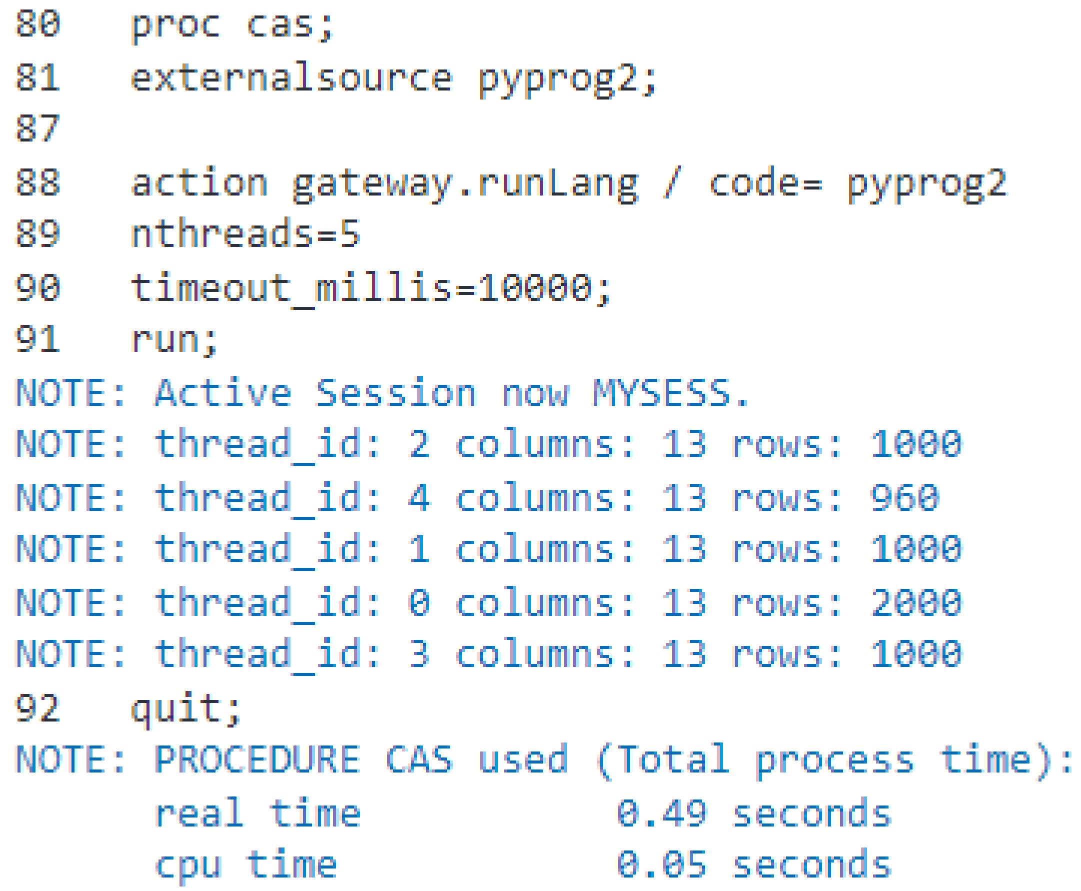
```
proc cas;
code="print('hello World {}'.format(gateway.thread_id))";

action gateway.runLang / code= code
nthreads=10; /* nthreads per node */
run;
quit;
```



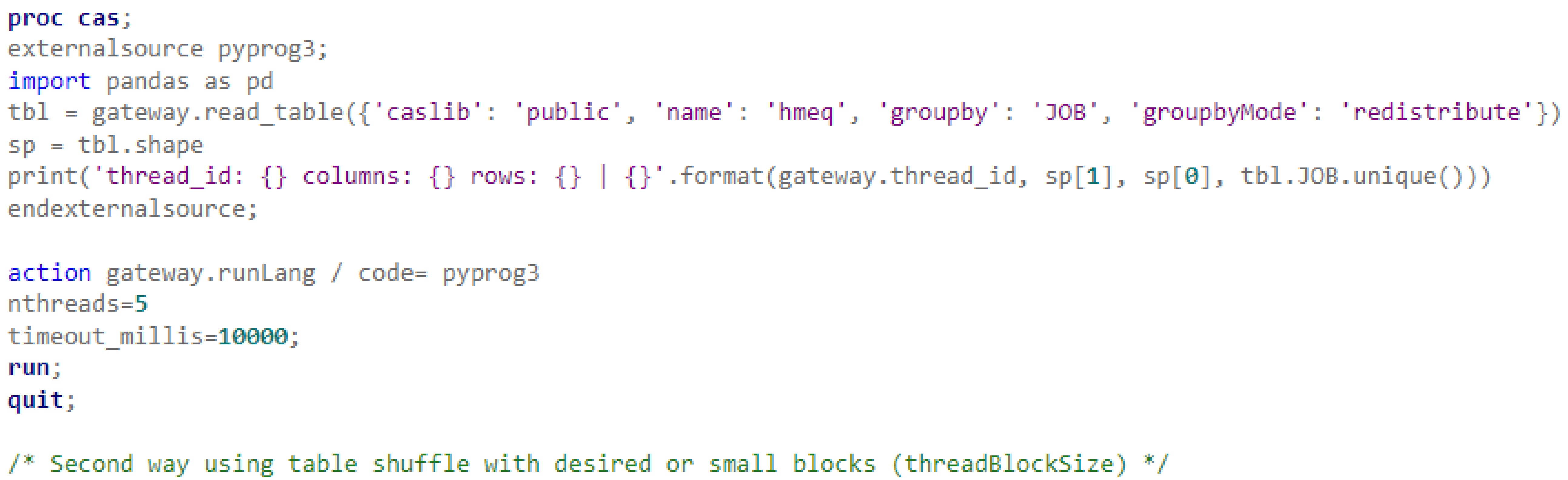
```
proc cas;
code="print('hello World {}'.format(gateway.thread_id))";

action gateway.runLang / code= code
nthreads=10; /* nthreads per node */
run;
NOTE: Active Session now MYSESS.
NOTE: hello World 8
NOTE: hello World 2
NOTE: hello World 7
NOTE: hello World 3
NOTE: hello World 1
NOTE: hello World 6
NOTE: hello World 0
NOTE: hello World 4
NOTE: hello World 5
NOTE: hello World 9
quit;
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.68 seconds
      cpu time            0.06 seconds
```



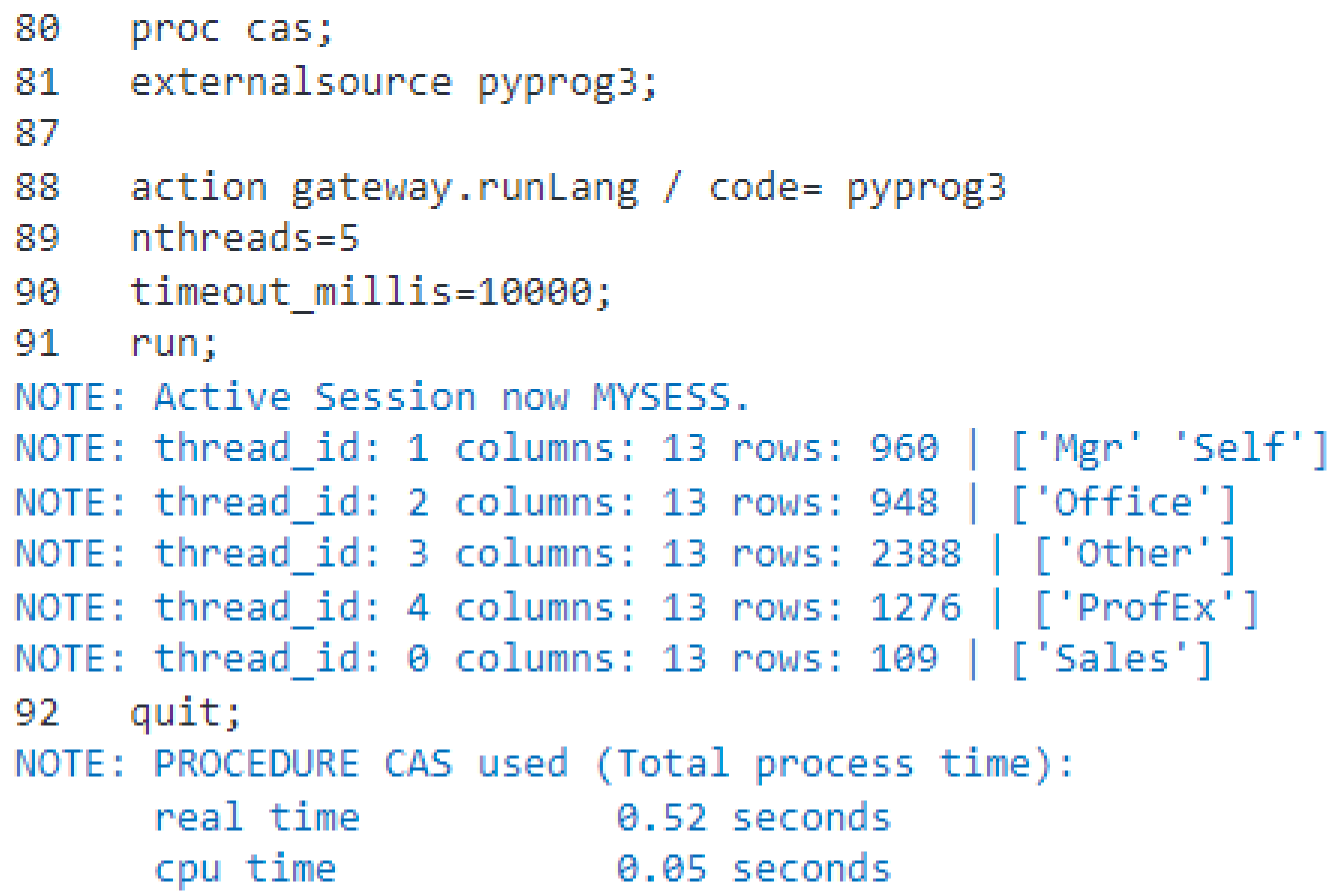
```
proc cas;
externalsource pyprog2;
import pandas as pd
tbl = gateway.read_table({'caslib': 'public', 'name': 'hmeq'})
sp = tbl.shape
print('thread_id: {} columns: {} rows: {}'.format(gateway.thread_id, sp[1], sp[0]))
endexternalsource;

action gateway.runLang / code= pyprog2
nthreads=5
timeout_millis=10000;
run;
quit;
```



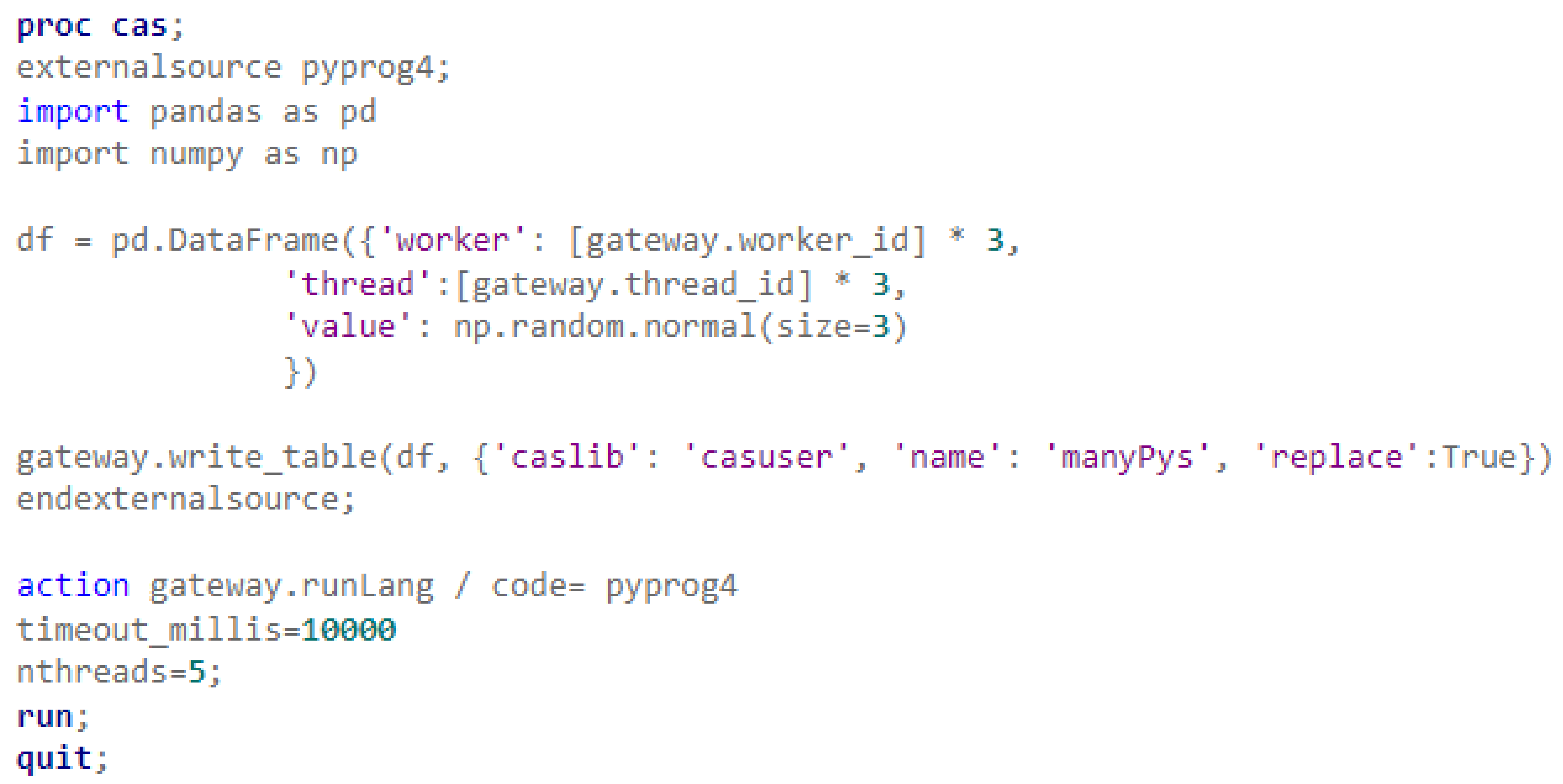
```
80    proc cas;
81    externalsource pyprog2;
87
88    action gateway.runLang / code= pyprog2
89    nthreads=5
90    timeout_millis=10000;
91    run;
NOTE: Active Session now MYSESS.
NOTE: thread_id: 2 columns: 13 rows: 1000
NOTE: thread_id: 4 columns: 13 rows: 960
NOTE: thread_id: 1 columns: 13 rows: 1000
NOTE: thread_id: 0 columns: 13 rows: 2000
NOTE: thread_id: 3 columns: 13 rows: 1000
92    quit;
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.49 seconds
      cpu time            0.05 seconds
```



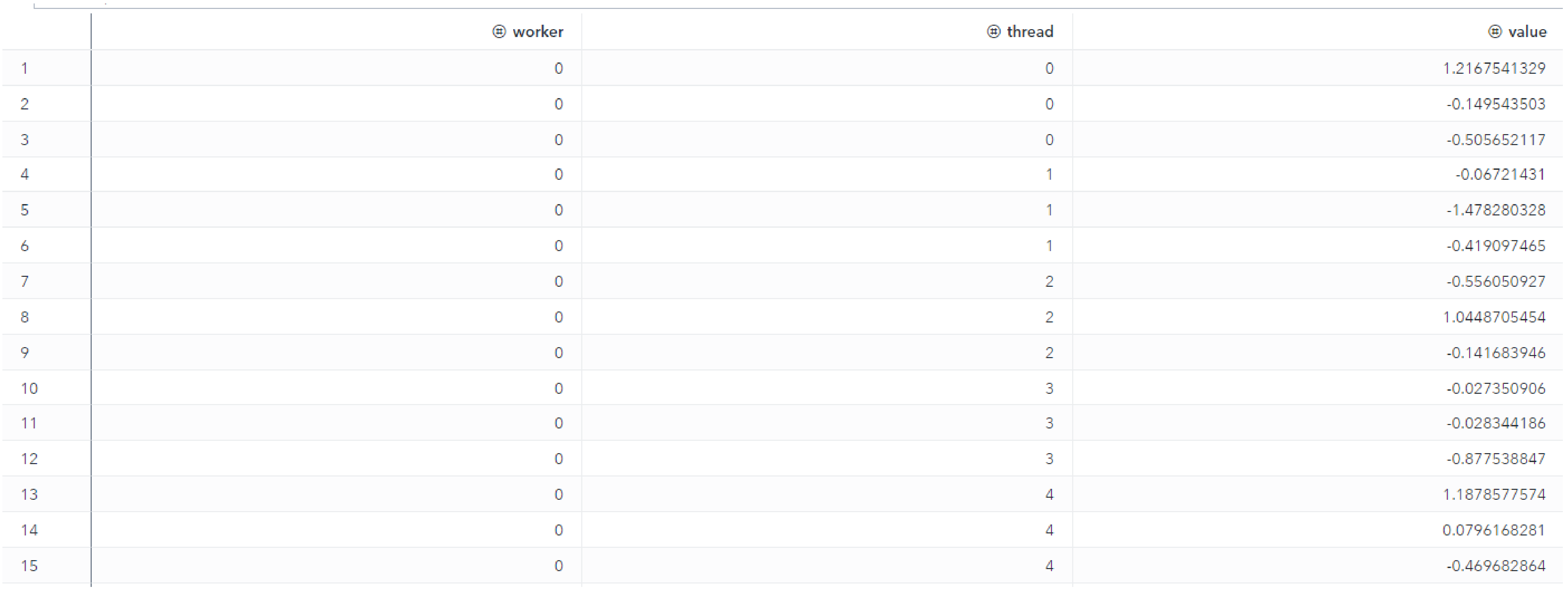
```
proc cas;
externalsource pyprog3;
import pandas as pd
tbl = gateway.read_table({'caslib': 'public', 'name': 'hmeq', 'groupby': 'JOB', 'groupbyMode': 'redistribute'})
sp = tbl.shape
print('thread_id: {} columns: {} rows: {} | {}'.format(gateway.thread_id, sp[1], sp[0], tbl.JOB.unique()))
endexternalsource;

action gateway.runLang / code= pyprog3
nthreads=5
timeout_millis=10000;
run;
quit;

/* Second way using table shuffle with desired or small blocks (threadBlockSize) */
```



```
80    proc cas;
81    externalsource pyprog3;
87
88    action gateway.runLang / code= pyprog3
89    nthreads=5
90    timeout_millis=10000;
91    run;
NOTE: Active Session now MYSESS.
NOTE: thread_id: 1 columns: 13 rows: 960 | ['Mgr' 'Self']
NOTE: thread_id: 2 columns: 13 rows: 948 | ['Office']
NOTE: thread_id: 3 columns: 13 rows: 2388 | ['Other']
NOTE: thread_id: 4 columns: 13 rows: 1276 | ['ProfEx']
NOTE: thread_id: 0 columns: 13 rows: 109 | ['Sales']
92    quit;
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.52 seconds
      cpu time            0.05 seconds
```



```
proc cas;
externalsource pyprog4;
import pandas as pd
import numpy as np

df = pd.DataFrame({'worker': [gateway.worker_id] * 3,
              'thread':[gateway.thread_id] * 3,
              'value': np.random.normal(size=3)
              })

gateway.write_table(df, {'caslib': 'casuser', 'name': 'manyPys', 'replace':True})
endexternalsource;

action gateway.runLang / code= pyprog4
timeout_millis=10000
nthreads=5;
run;
quit;
```



	worker	thread	value
1	0	0	1.2167541329
2	0	0	-0.149543503
3	0	0	-0.505652117
4	0	1	-0.06721431
5	0	1	-1.478280328
6	0	1	-0.419097465
7	0	2	-0.556050927
8	0	2	1.0448705454
9	0	2	-0.141683946
10	0	3	-0.027350906
11	0	3	-0.028344186
12	0	3	-0.877538847
13	0	4	1.1878577574
14	0	4	0.0796168281
15	0	4	-0.469682864

Tankar

Exempel på Use case

- Scoring av open source-modeler
- Oberoende beräkningar/simuleringar
- OCR (t ex tesseract)
- Riskberäkningar (ECL, Stresstestning etc)

Takeaways

- Man kan nu parallellisera “vanlig” pythonkod för visa use case
- Dataåtkomsten I CAS är riktigt snabb

Tack!