PyData

Parallelize data processing

@PyDataVenice #13 #Meetup #PyData

Alessandra Bilardi

Data & Automation Specialist

- AWS User Group Venezia member
- Coderdojo member
- PyData Venezia member

alessandra.bilardi@gmail.com

@abilardi

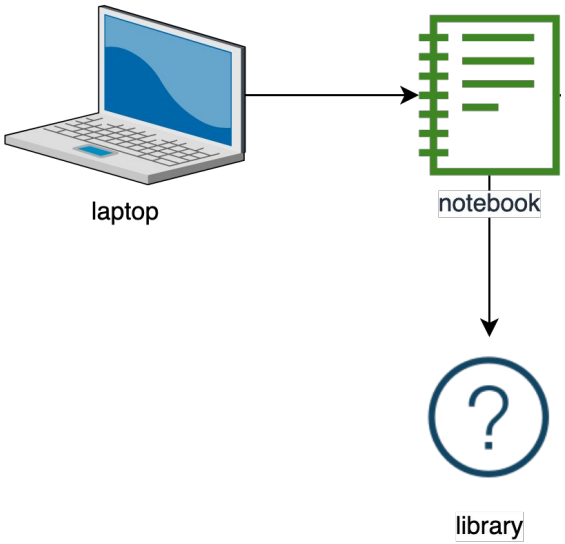
bilardi

Agenda

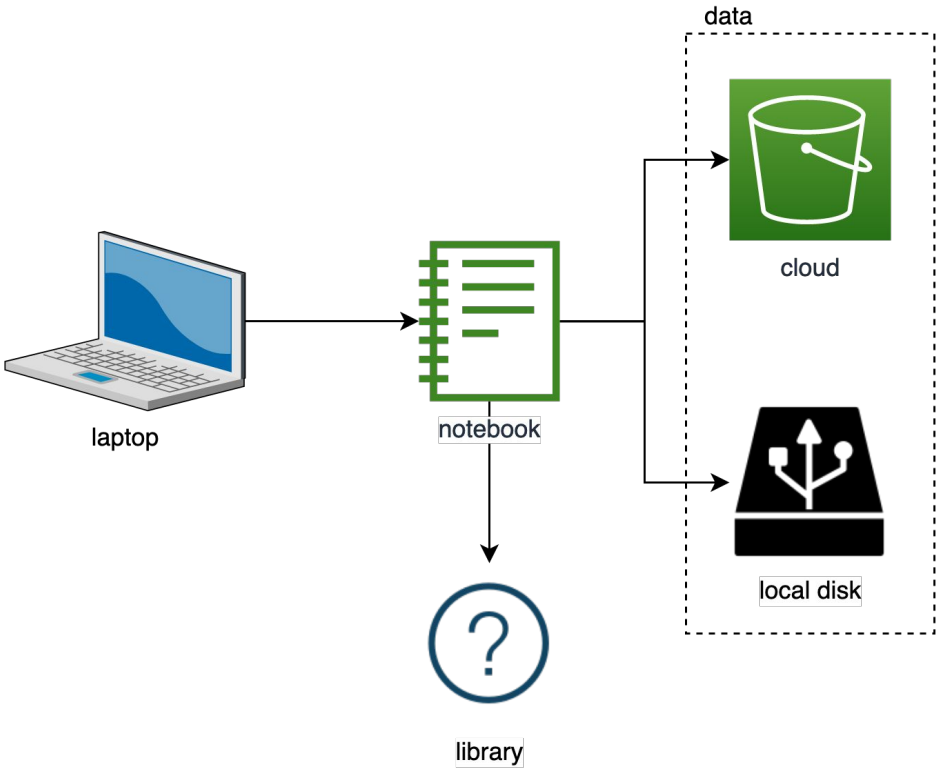
Goal
Challenge
Solution
Results
Best practices

Goal

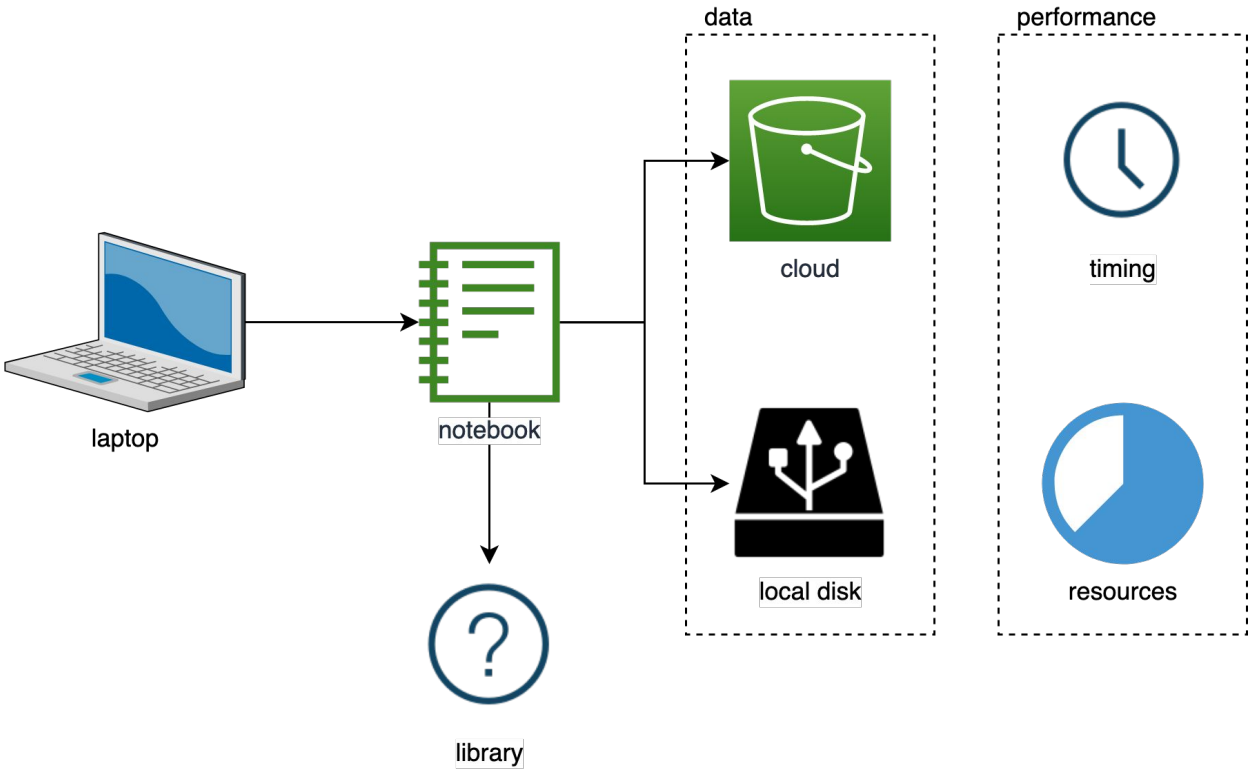
Goal



Goal



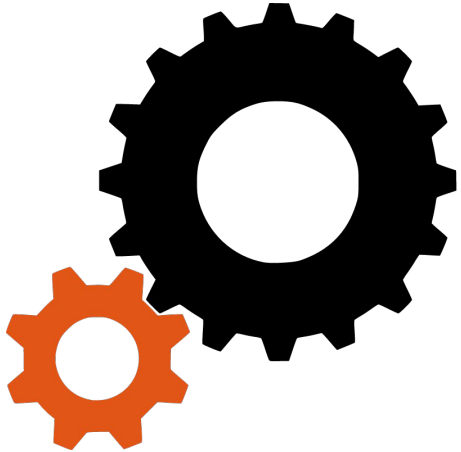
Goal

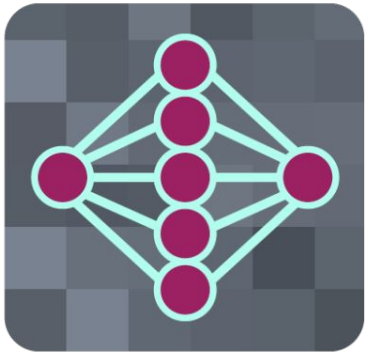

data
performance
cloud
timing
laptop
notebook
local disk
resources
library

Challenge

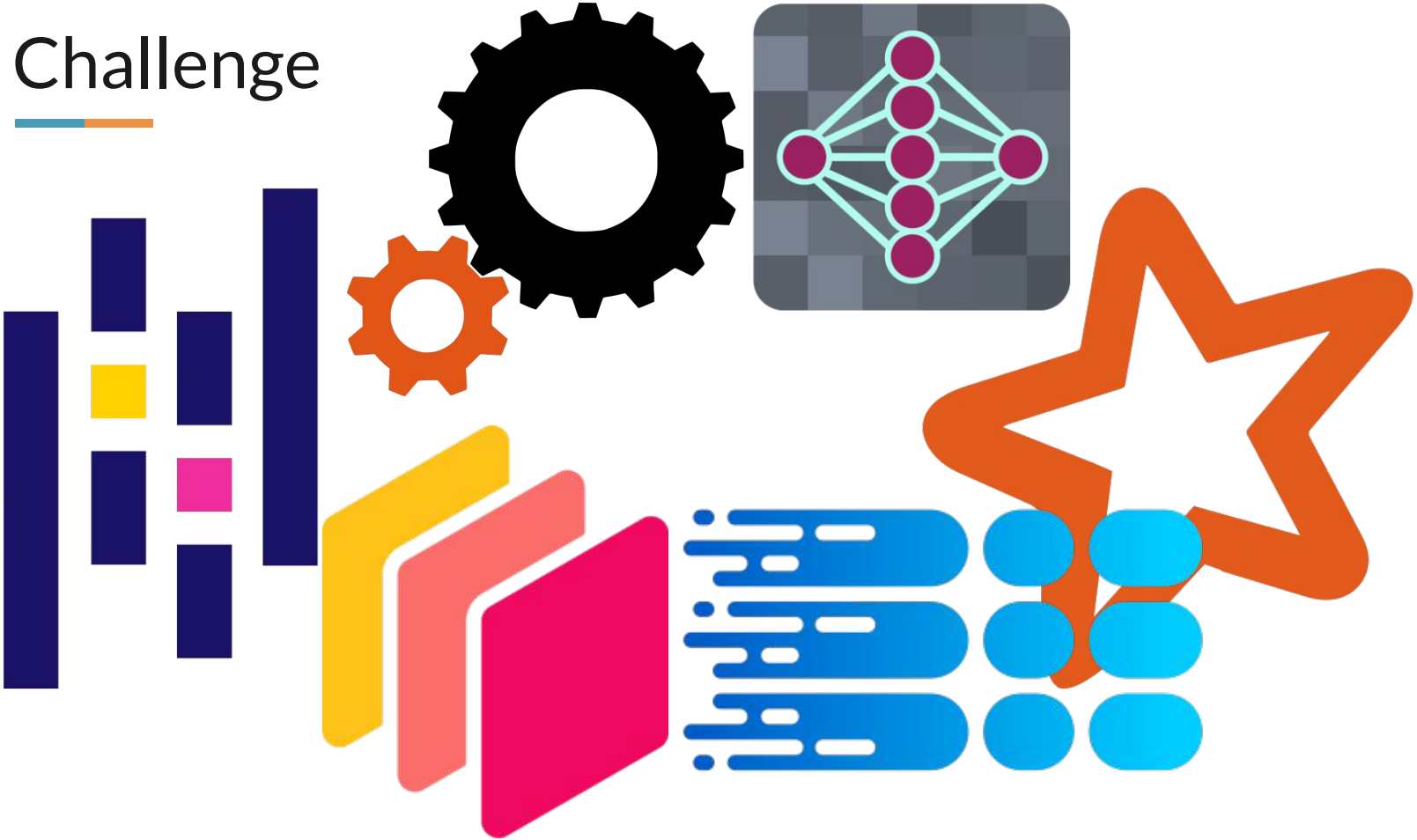
Challenge

Challenge

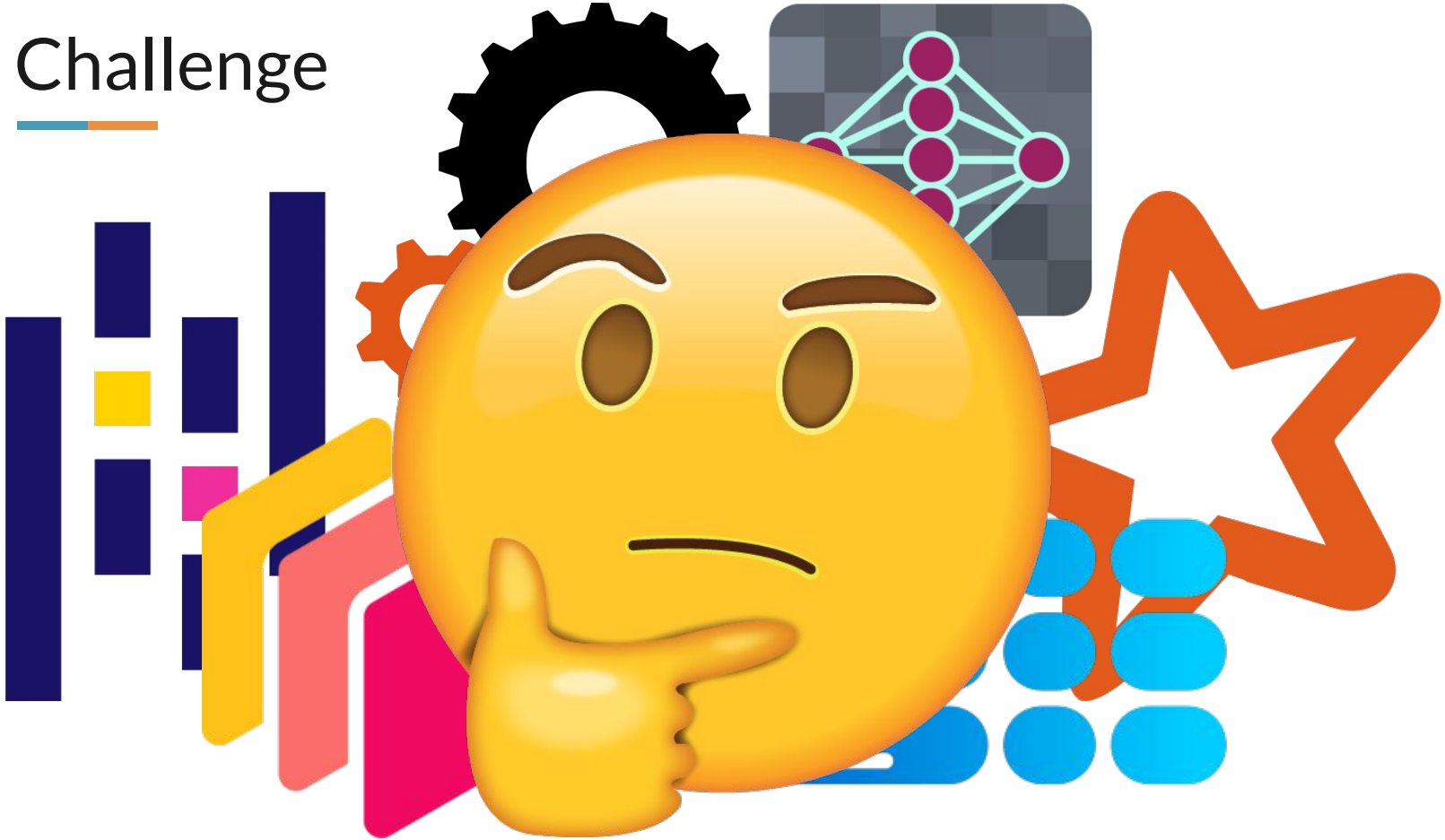




Challenge



Challenge



may

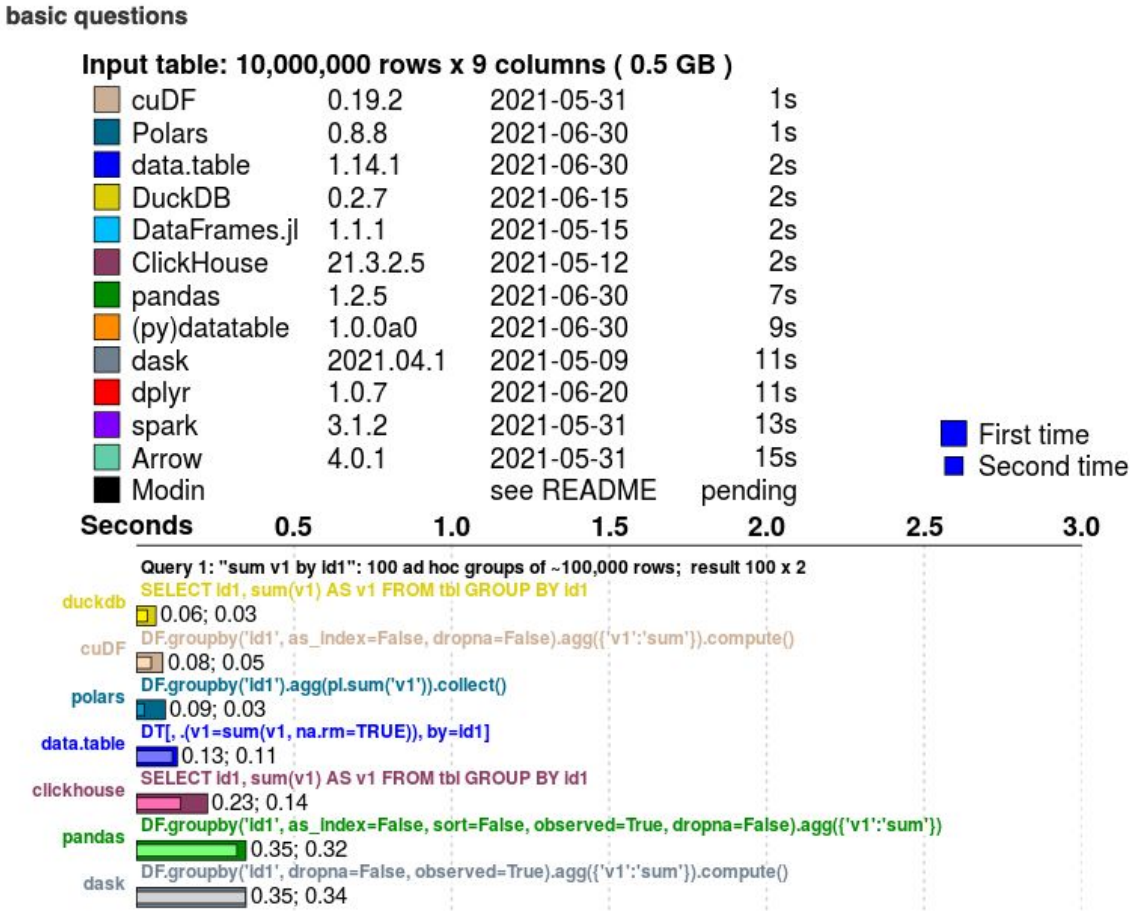
open source

with you

Challenge

PyData

Challenge



https://h2oai.github.io/db-benchmark/

Challenge

Summary

We looked at a simple CPU bound case. We then used the following

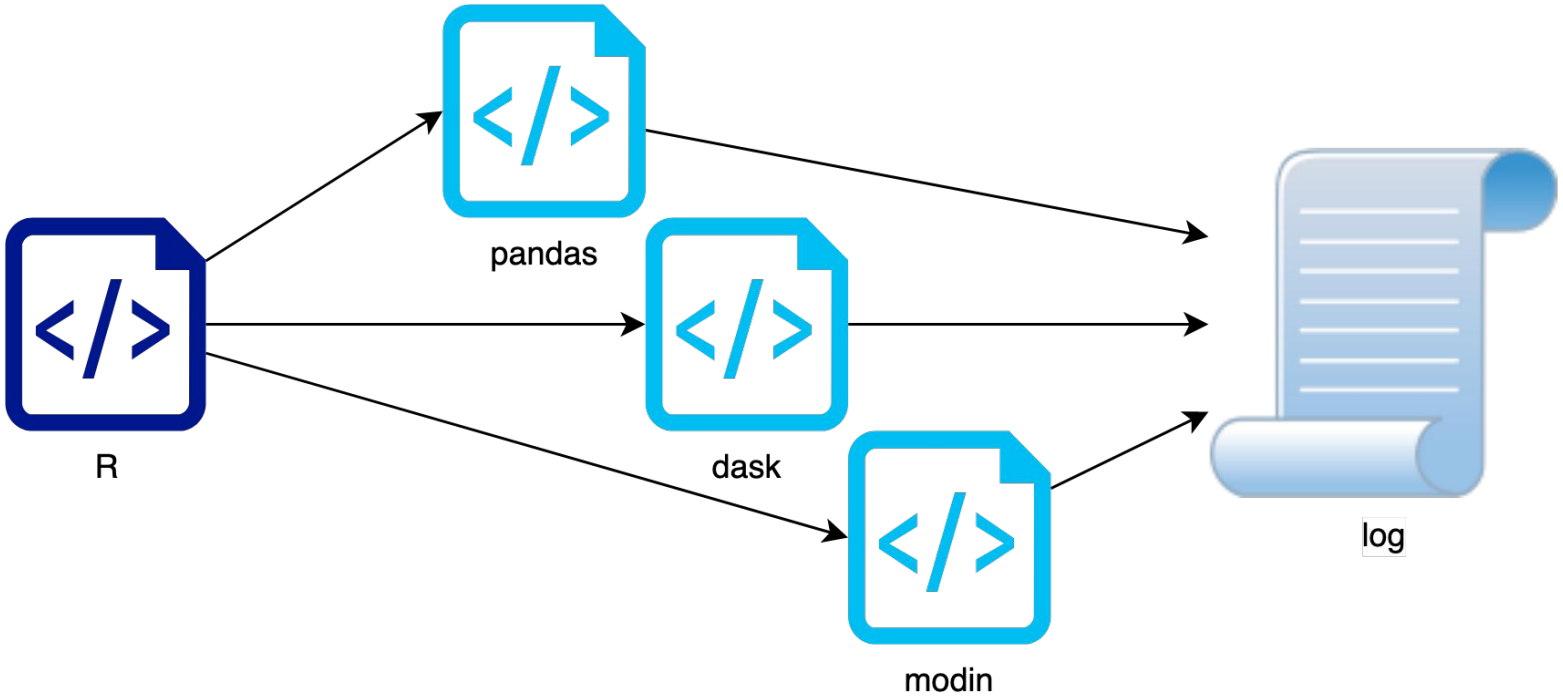
- multiprocessing
- joblib
- Dask
- Modin
- Swifter
- Pandarallel
- PySpark

```python
import math
import sys
import argparse
import multiprocessing

import numpy as np

def slow_function(start: float) -> float:
    res = 0
    for i in range(int(math.pow(start, 7))):
        res += math.atan(i) * math.atan(i)
    return res


def get_sample():
    data = {'value': np.random.random(500) + 5}
    return data
```

https://www.wrighters.io/options-to-run-pandas-dataframe-apply-in-parallel/

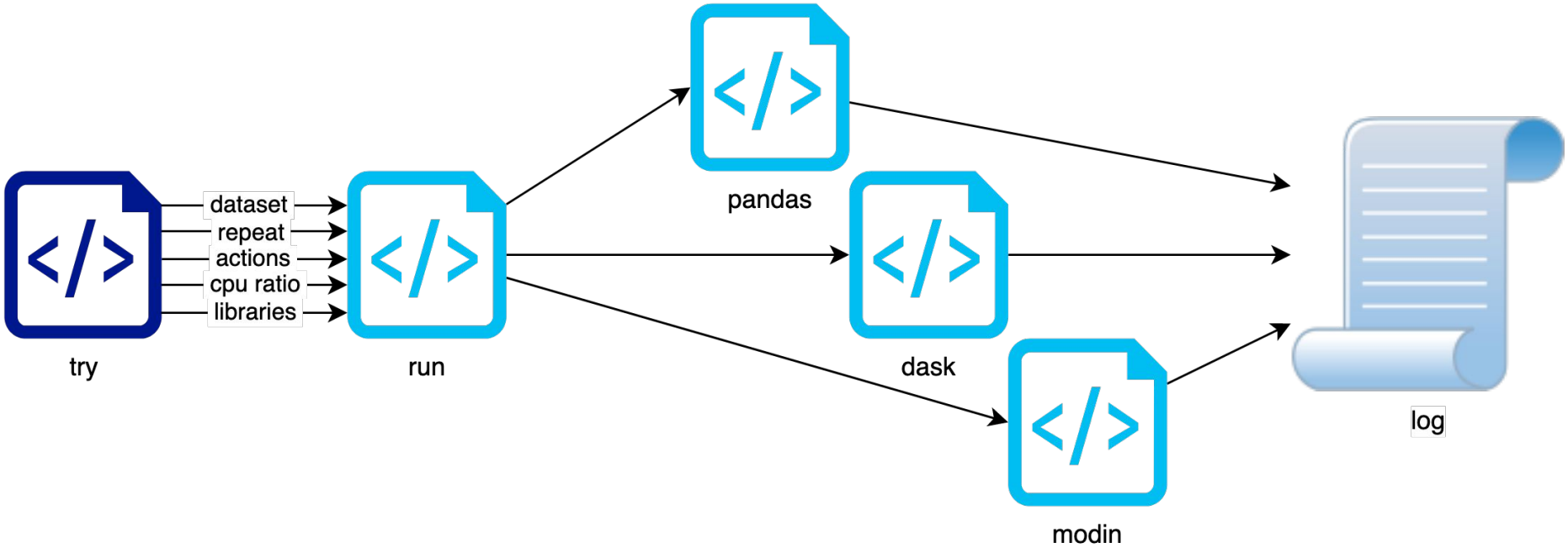
Solution

Solution



https://h2oai.github.io/db-benchmark/

Solution


dataset
repeat
actions
cpu ratio
libraries
try
run
pandas
dask
modin
log

PyData

Solution

Goal:
performance

Challenge:
on 4 CPU & 8GB RAM

- try.sh
  - for each dataset
    - run.py
- run.py
  - for each library
    - importlib
    - main()
- main()
  - for each method
    - timeit before/after
    - print timing
    - gc.collect()

PyData

Datasets - rows

Goal:

execution timing

Challenge:

resources

- 10
- 50
- 100
- 500
- 1.000
- 5.000
- 10.000
- 50.000
- 100.000
- 500.000
- 1.000.000
- 5.000.000
- 10.000.000

PyData

Libraries

Goal:
learning curve

Challenge:
the same behaviours

- concurrent
- dask
- joblib
  - joblib on dask
- modin
  - modin on dask
- multiprocessing
- multiprocesspandas
- pandarallel
- pandas
- parallelize
- pyspark
- swifter

run_libraries.py

```
if os.path.exists(args.libraries):
    files = [file for file in os.listdir(args.libraries) if os.path.isfile(f'{args.libraries}/{file}')]
    for file in files:
        if args.types == 'all' or file.startswith(f'{args.types}_'):
            library = file.split('.')[0]
            library_name = library.replace('_actions', '')
            actions = ['series', 'fast', 'slow', 'groupby']
            try:
                print_error(f'library: {library}')
                run = importlib.import_module(f'libraries.{library}')
                run.main(actions, {'RANDOM_NUMBER': RANDOM_NUMBER, 'CPU_RATIO': CPU_RATIO, 'DATASET': DATASET})
            except Exception as error:
                print_error(error)
else:
    logging.info(f'{args.libraries} not found')
    sys.exit(1)
```

run_libraries.py

```
if os.path.exists(args.libraries):
    files = [file for file in os.listdir(args.libraries) i
    for file in files:
        if args.types == 'all' or file.startswith(f'{args.
            library = file.split('.')[0]
            library_name = library.replace('_actions', '')
            actions = ['series', 'fast', 'slow', 'groupby']
            try:
                print_error(f'library: {library}')
                run = importlib.import_module(f'libraries.
                run.main(actions, {'RANDOM_NUMBER': RANDOM_
            except Exception as error:
                print_error(error)
else:
    logging.info(f'{args.libraries} not found')
    sys.exit(1)
```

```
def get_sample() -> dict:
    global RANDOM_NUMBER, DATASET
    data = {'v1': np.random.random(RANDOM_NUMBER) + 5}
    if DATASET:
        data = pd.read_csv(DATASET)
        RANDOM_NUMBER = len(data)
        data = data.to_dict()
    return data

def get_pd_sample() -> dict:
    return pd.DataFrame(get_sample())

def series_function(a: float, b: float) -> float:
    return a**2 + b**2

def slow_function(start: float) -> float:
    res = 0
    for i in range(int(math.pow(start, 7))):
        res += math.atan(i) * math.atan(i)
    return res

def get_cpu_count() -> int:
    return int(0.5 * multiprocessing.cpu_count())
```

run_libraries.py

```
if os.path.exists(args.libraries):
```

```
def get_time() -> int:
    return timeit.default_timer()

def print_info(name, action, time) -> None:
    global RANDOM_NUMBER
    print('running', name, RANDOM_NUMBER, action, time)
    print_error(f'gc: {gc.collect()}')

def print_error(message) -> None:
    print(message, file=sys.stderr)
```

```
                print_error(error)
else:
    logging.info(f'{args.libraries} not found')
    sys.exit(1)
```

```
def get_sample() -> dict:
    global RANDOM_NUMBER, DATASET
    data = {'v1': np.random.random(RANDOM_NUMBER) + 5}
    if DATASET:
        data = pd.read_csv(DATASET)
        RANDOM_NUMBER = len(data)
        data = data.to_dict()
    return data

def get_pd_sample() -> dict:
    return pd.DataFrame(get_sample())

def series_function(a: float, b: float) -> float:
    return a**2 + b**2

def slow_function(start: float) -> float:
    res = 0
    for i in range(int(math.pow(start, 7))):
        res += math.atan(i) * math.atan(i)
    return res

def get_cpu_count() -> int:
    return int(0.5 * multiprocessing.cpu_count())
```

main()

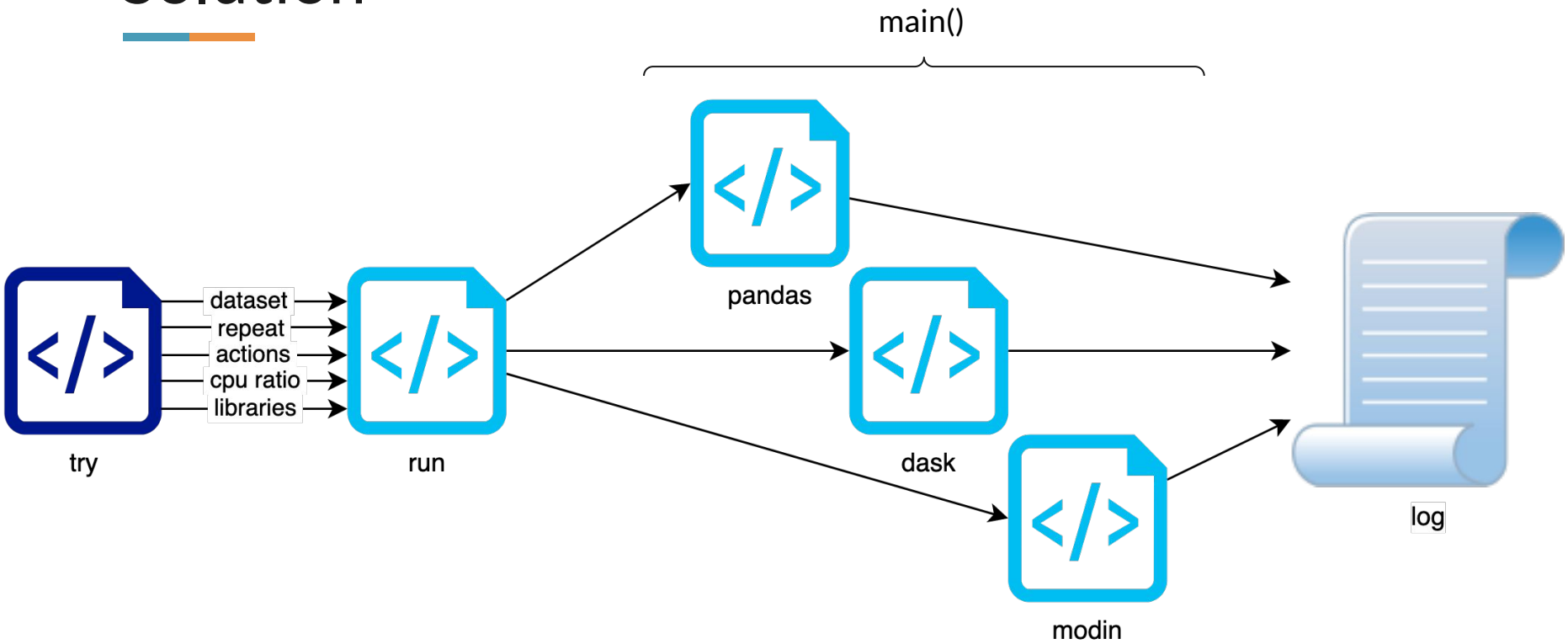
```
import os
import sys
parent_dir = os.path.dirname(os.path.realpath(__file__))
sys.path.append(f'{parent_dir}/../')
import run_libraries as hlp

def main(actions: [] = None, params: dict = None) -> None:
    hlp.set_params(params)
    name = 'pandas'
    t_start = hlp.get_time()
    sample = hlp.get_pd_sample()
    t_load = hlp.get_time() - t_start
    hlp.print_info(name, 'load', t_load)
```

main()

```
import os
import sys
parent_dir = os.path.dirname(c
sys.path.append(f'{parent_dir}
import run_libraries as hlp

def main(actions: [] = None, p
    hlp.set_params(params)
    name = 'pandas'
    t_start = hlp.get_time()
    sample = hlp.get_pd_sample
    t_load = hlp.get_time() -
    hlp.print_info(name, 'loac
```

```
    if 'series' in actions:
        t_start = hlp.get_time()
        sample['results'] = sample['v1'].apply(lambda x: hlp.series_function(x, x))
        t_apply = hlp.get_time() - t_start
        hlp.print_info(name, 'series', t_apply)

    if 'slow' in actions:
        t_start = hlp.get_time()
        sample['results'] = sample['v1'].apply(hlp.slow_function)
        t_apply = hlp.get_time() - t_start
        hlp.print_info(name, 'slow', t_apply)

    if 'groupby' in actions:
        t_start = hlp.get_time()
        groupby = sample.groupby(['id1'])
        t_apply = hlp.get_time() - t_start
        hlp.print_info(name, 'group-by-id1', t_apply)
        t_start = hlp.get_time()
        sum = [groupby['v1'].sum()]
        t_apply = hlp.get_time() - t_start
        hlp.print_info(name, 'total-sum-v1-by-group-by-id1', t_apply)
        t_start = hlp.get_time()
        groupby = sample.groupby(['id1']).agg({'v1':'sum'})
        t_apply = hlp.get_time() - t_start
        hlp.print_info(name, 'sum-v1-by-id1', t_apply)
```

Solution


main()
dataset
repeat
actions
cpu ratio
libraries
try
run
pandas
dask
modin
log

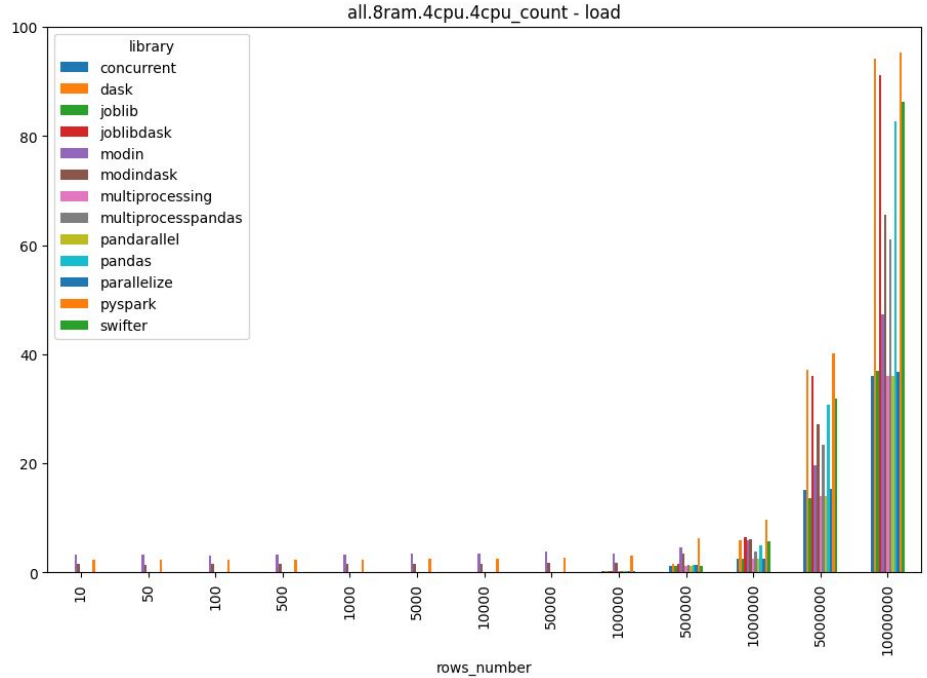
Results

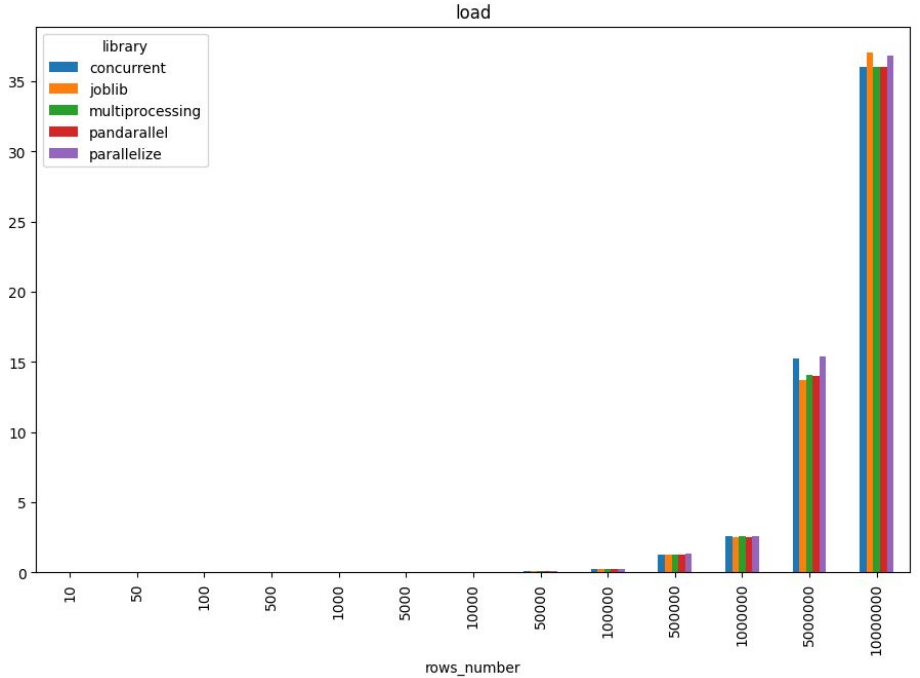
Results

```
running joblib 5 load 0.0022073750005802l4
running joblib 5 series 0.6373185830016155
running joblib 5 slow 0.141793249997l3572
running multiprocessing 5 load 0.0018739590013865381
running multiprocessing 5 series 0.0369430840000859l6
running multiprocessing 5 slow 0.9169750419969205
running pyspark 5 load 3.2806124170019757
running pyspark 5 series 1.3050238750001881
running pyspark 5 slow 0.01458825000008801
running parallelize 5 load 0.0015594170035910793
running vectorization 5 load 0.0009644999954616651
running vectorization 5 series 0.00028216600185260177
running concurrent 5 load 0.0008403749961871654
running concurrent 5 series 0.0003440829968894832
running concurrent 5 slow 0.021967125001538l8
```

PyData

Results



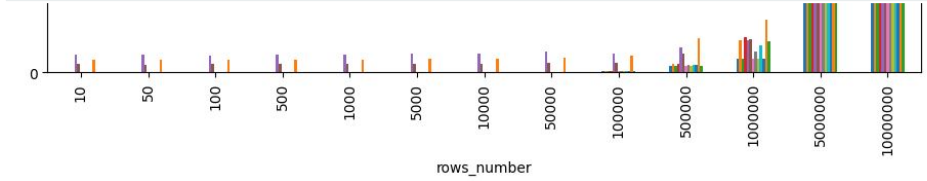


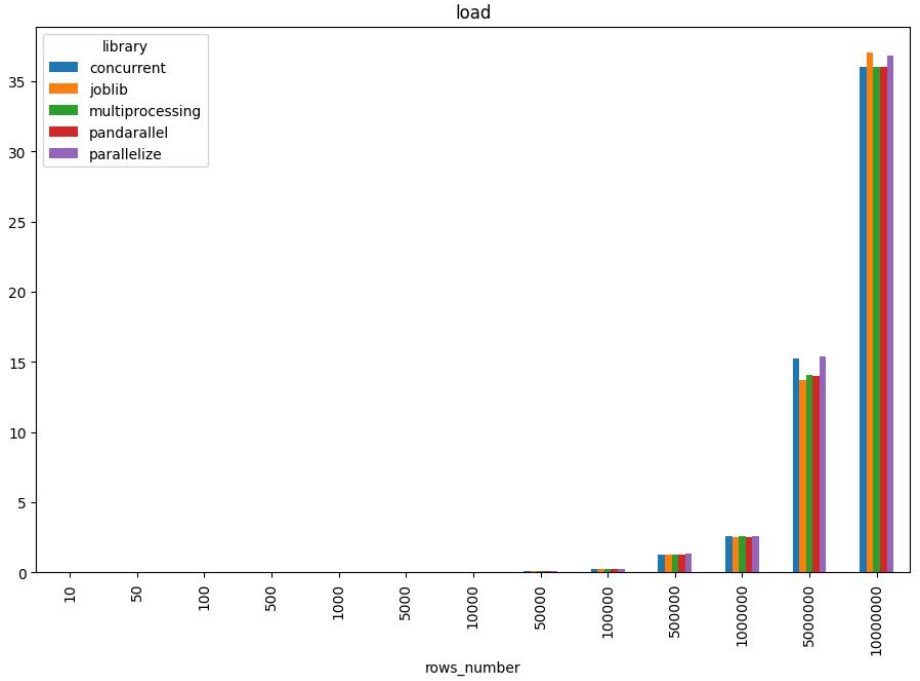
PyData

Results

$ grep "**hlp.get_pd_sample()**" *

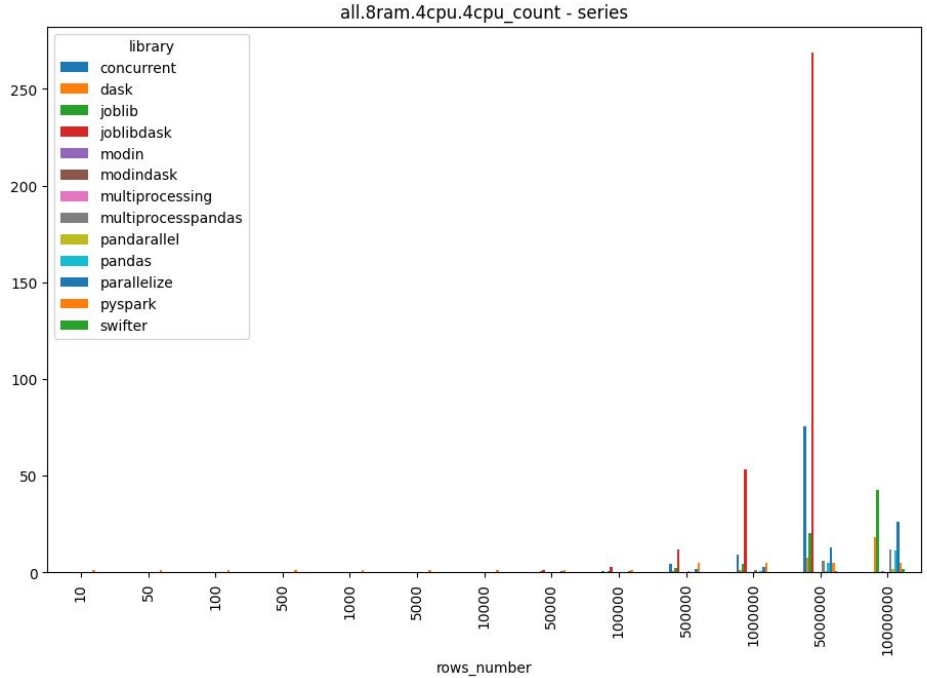
concurrent_actions.py: sample = hlp.get_pd_sample()
joblib_actions.py: sample = hlp.get_pd_sample()
joblibdask_actions.py: sample = hlp.get_pd_sample()
multiprocessing_actions.py: sample = hlp.get_pd_sample()
multiprocesspandas_actions.py: sample = hlp.get_pd_sample()
pandarallel_actions.py: sample = hlp.get_pd_sample()
pandas_actions.py: sample = hlp.get_pd_sample()
parallelize_actions.py: sample = hlp.get_pd_sample()
swifter_actions.py: sample = hlp.get_pd_sample()

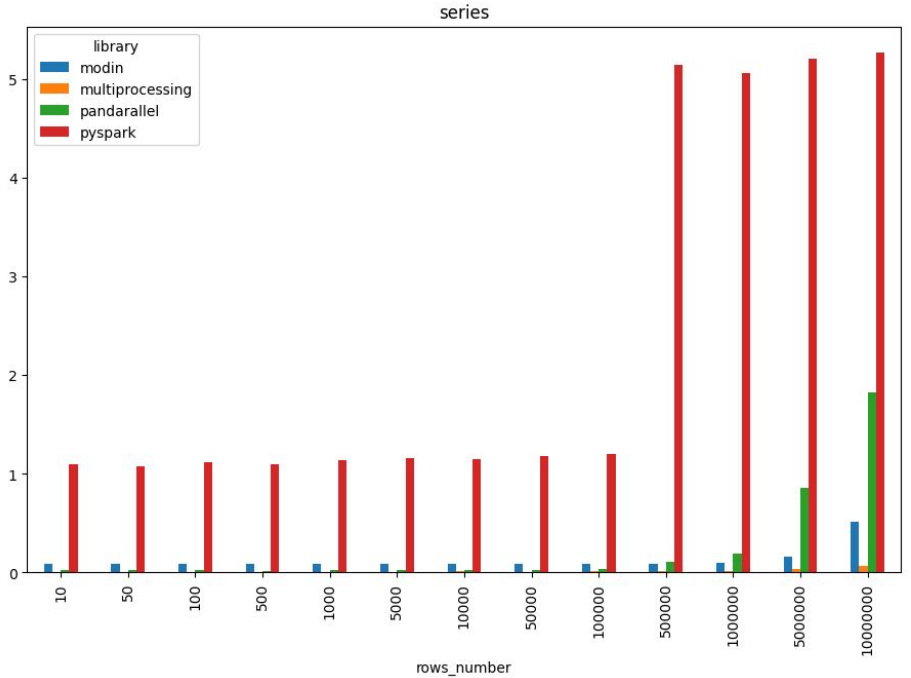




PyData

Results



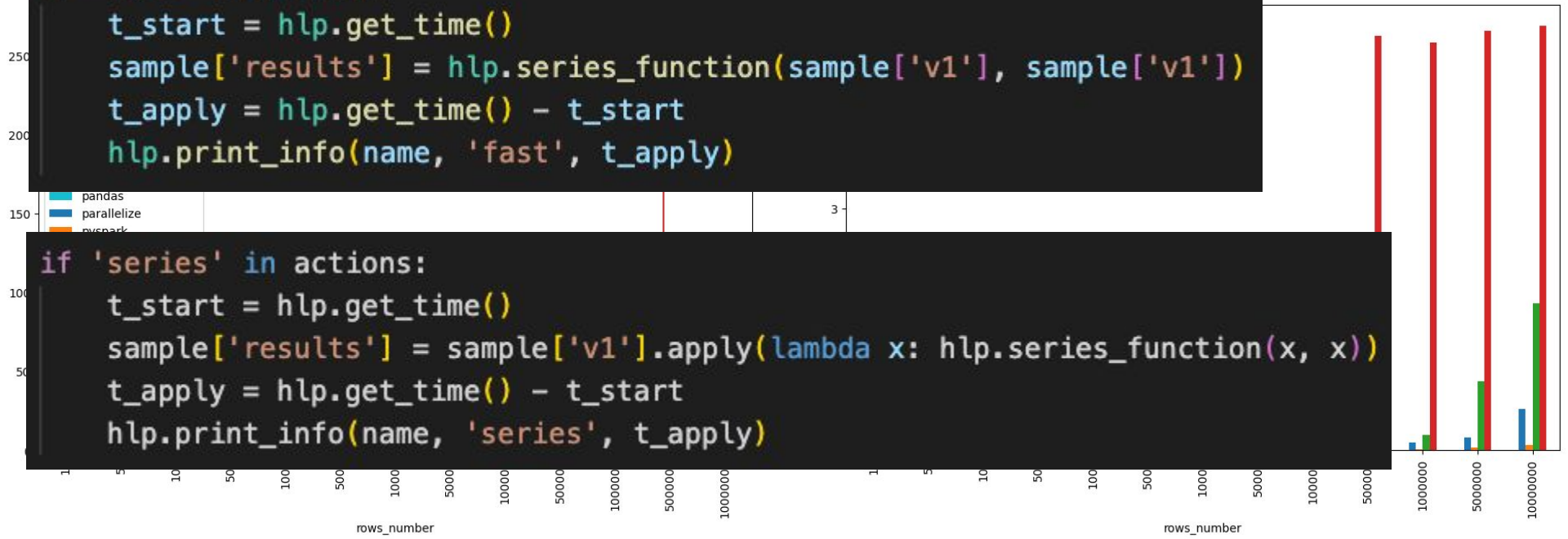


PyData

Results

```
if 'fast' in actions:
    t_start = hlp.get_time()
    sample['results'] = hlp.series_function(sample['v1'], sample['v1'])
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'fast', t_apply)
```

```
if 'series' in actions:
    t_start = hlp.get_time()
    sample['results'] = sample['v1'].apply(lambda x: hlp.series_function(x, x))
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'series', t_apply)
```



PyData

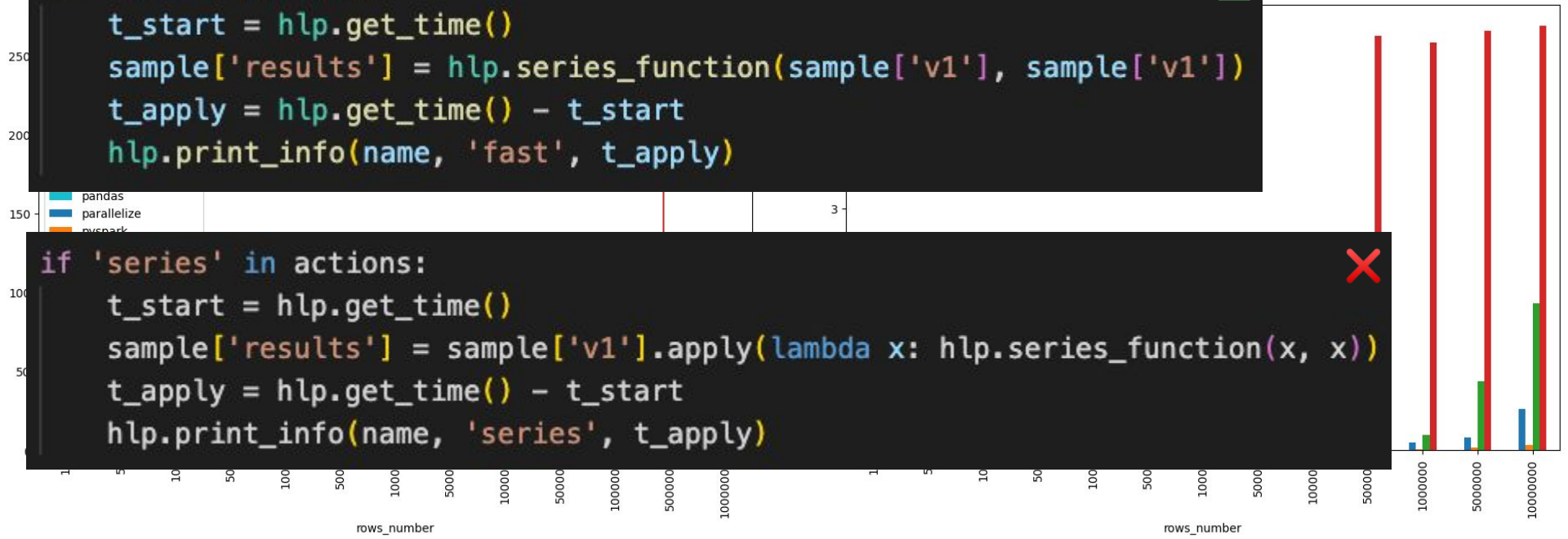
Results

```
if 'fast' in actions:
    t_start = hlp.get_time()
    sample['results'] = hlp.series_function(sample['v1'], sample['v1'])
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'fast', t_apply)
```

✅

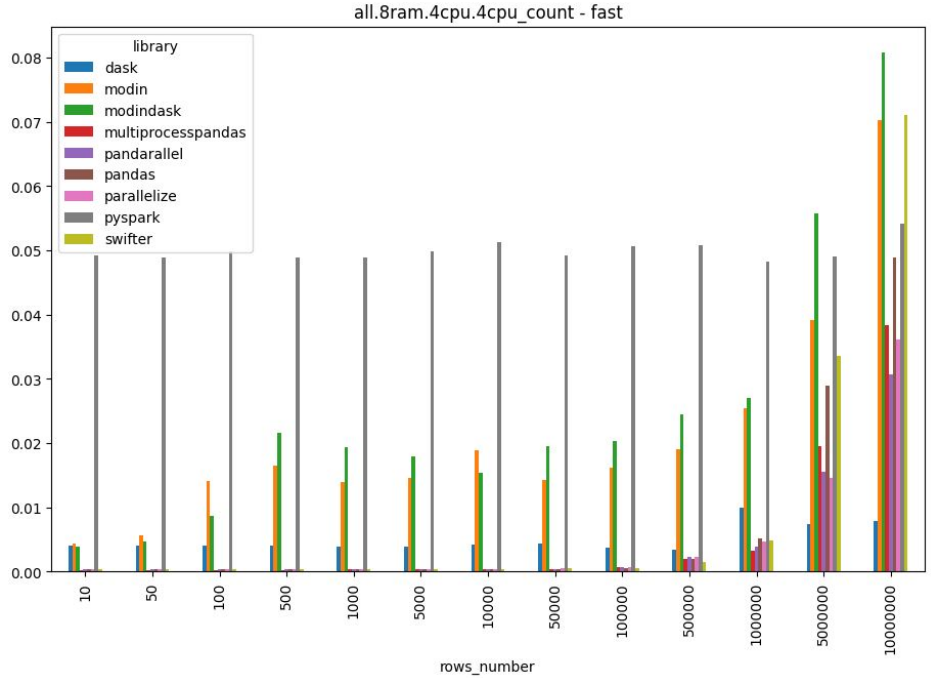
```
if 'series' in actions:
    t_start = hlp.get_time()
    sample['results'] = sample['v1'].apply(lambda x: hlp.series_function(x, x))
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'series', t_apply)
```

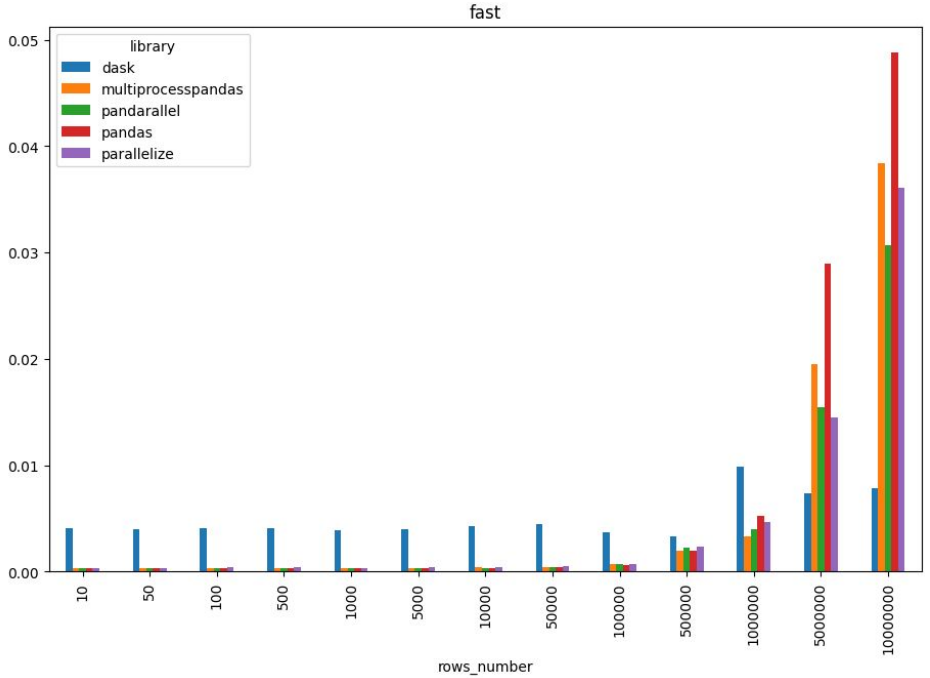
❌



PyData

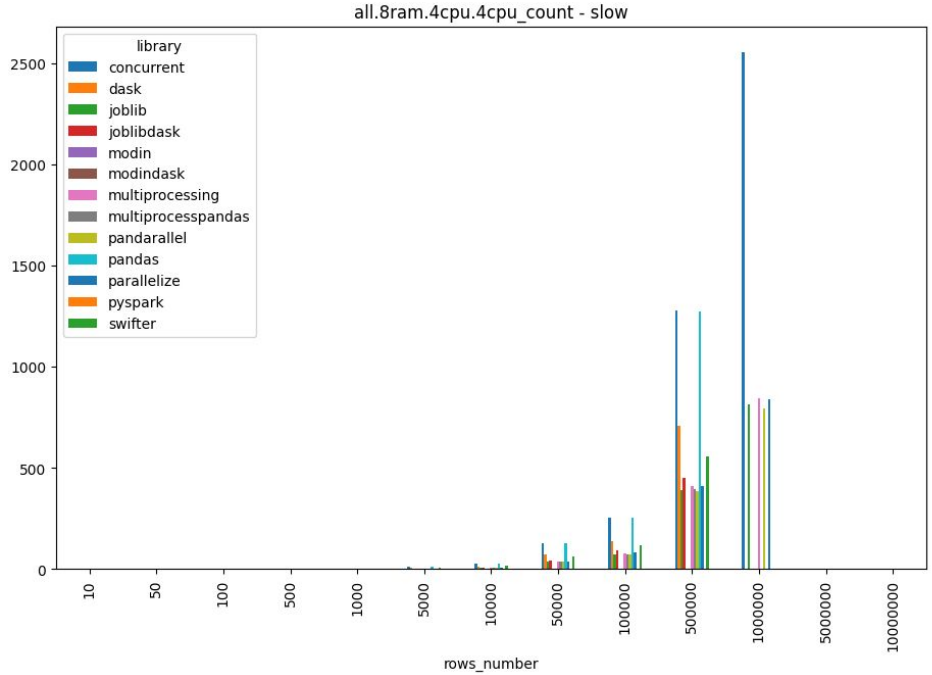
Results

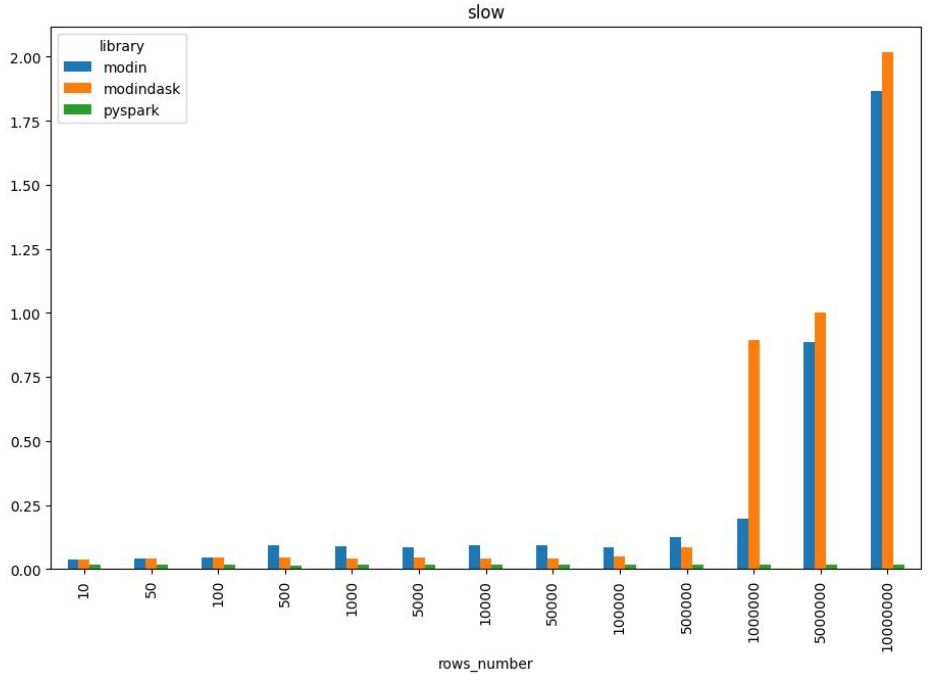




PyData

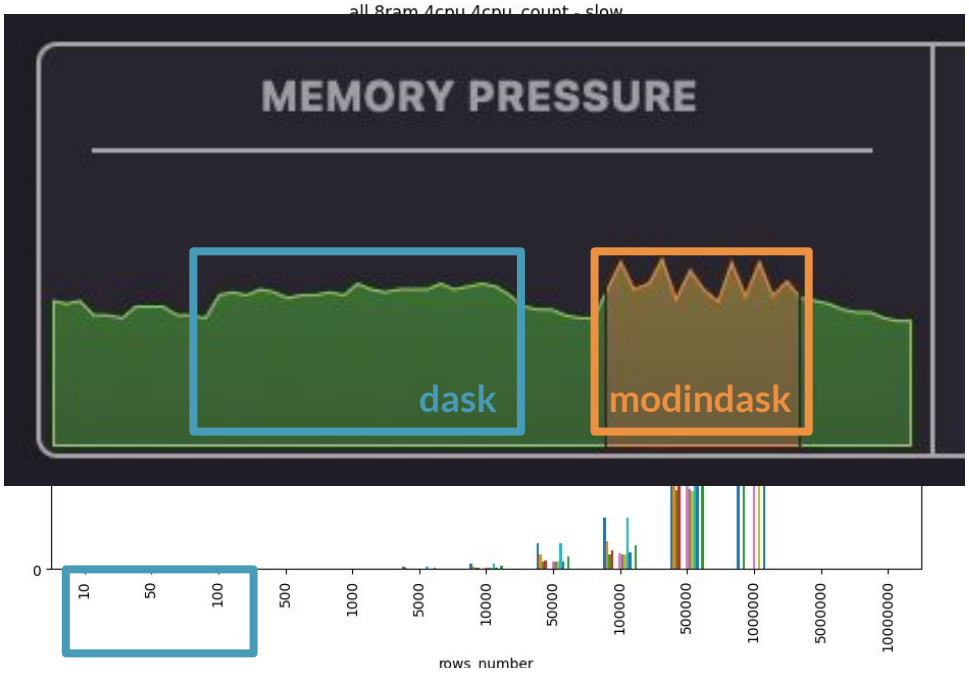
Results

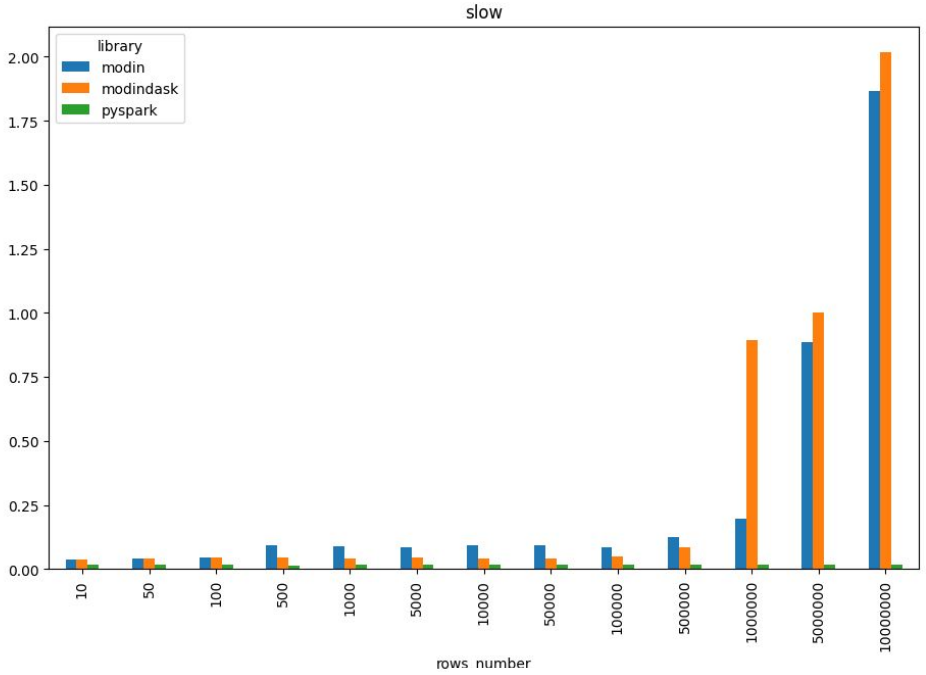




PyData

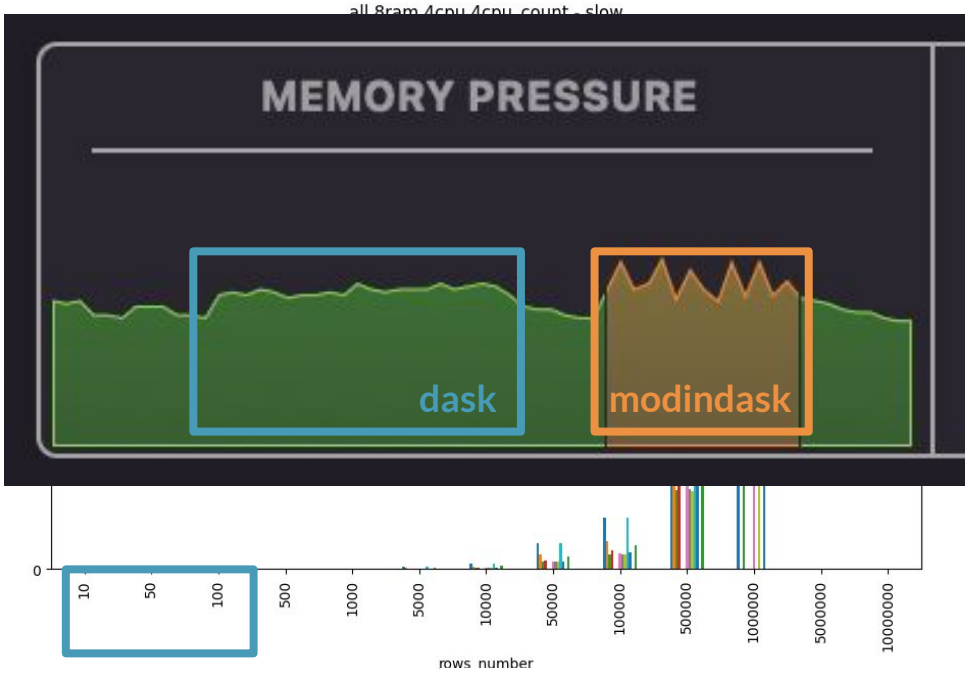
Results



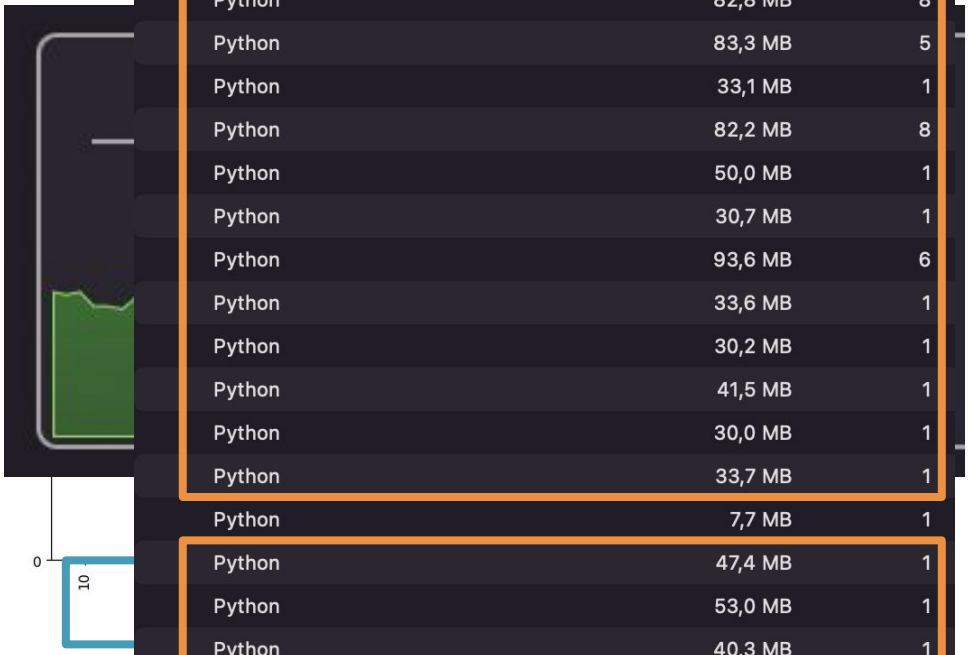


PyData

Results



Python	44,7 MB	3
Python	249,2 MB	29
Python	122,3 MB	19
Python	310,2 MB	26
Python	119,8 MB	19
Python	135,5 MB	22
Python	7,8 MB	1
Python	73,3 MB	1
Python	73,0 MB	1
Python	63,9 MB	1
Python	73,6 MB	1
Python	72,8 MB	1
Python	66,6 MB	1
Python	65,5 MB	1
Python	74,7 MB	1
Python	7,5 MB	1
powerd	4,7 MB	3
pkd	3,6 MB	2
PerfPowerServices	12,3 MB	6

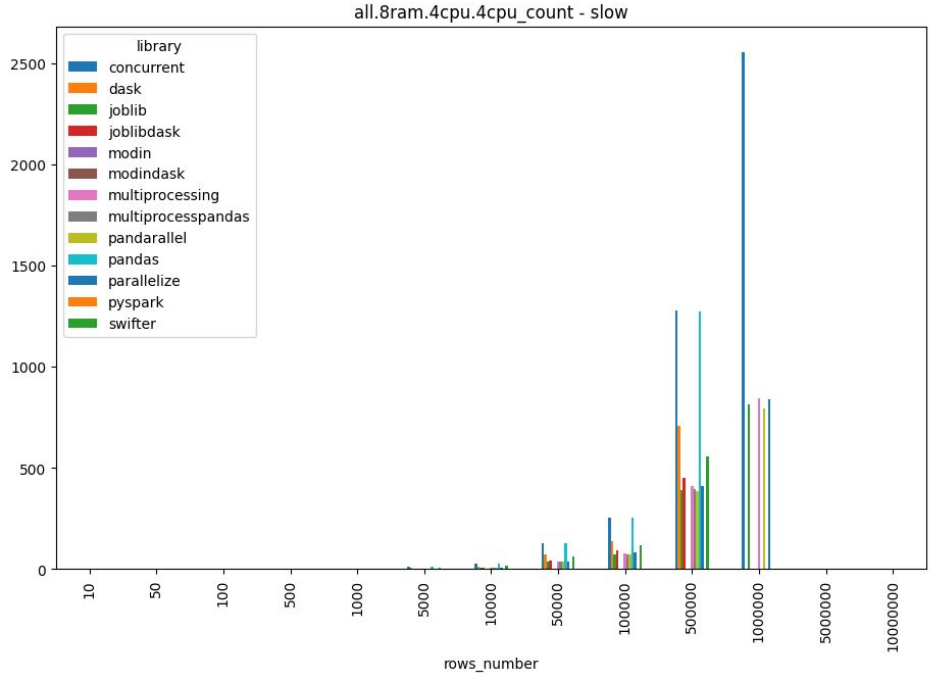


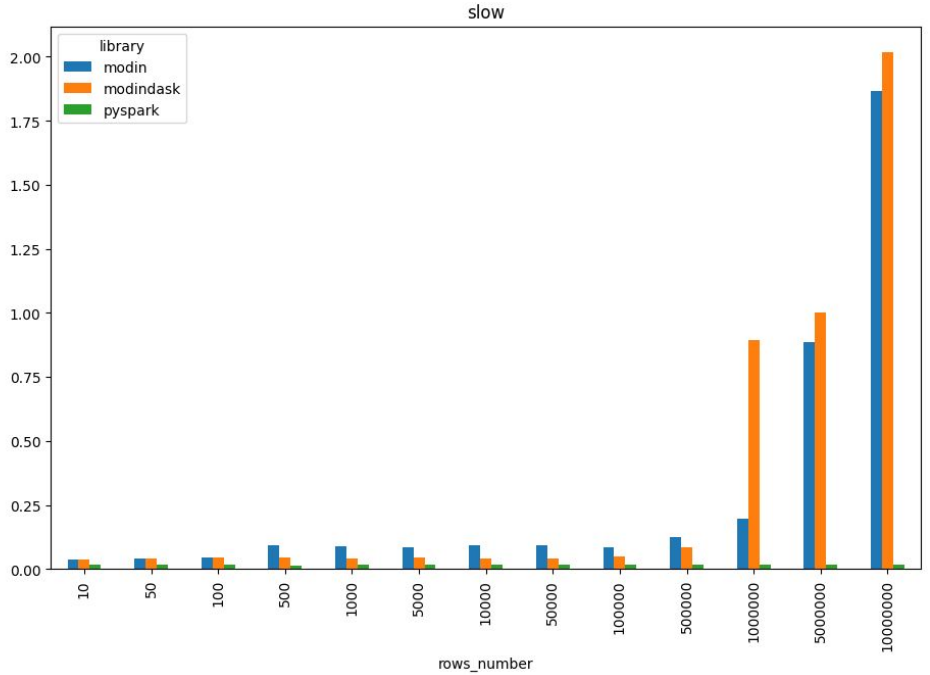
Python	122,3 MB	19
Python	135,5 MB	22
Python	310,2 MB	26
Python	249,2 MB	29
Python	44,7 MB	3
Python	119,8 MB	19
Python	92,3 MB	6
Python	82,8 MB	8
Python	83,3 MB	5
Python	33,1 MB	1
Python	82,2 MB	8
Python	50,0 MB	1
Python	30,7 MB	1
Python	93,6 MB	6
Python	33,6 MB	1
Python	30,2 MB	1
Python	41,5 MB	1
Python	30,0 MB	1
Python	33,7 MB	1
Python	7,7 MB	1
Python	47,4 MB	1
Python	53,0 MB	1
Python	40,3 MB	1
Python	53,1 MB	1
powerd	4,7 MB	3

Python	44,7 MB	3
Python	249,2 MB	29
Python	122,3 MB	19
Python	310,2 MB	26
Python	119,8 MB	19
Python	135,5 MB	22
Python	7,8 MB	1
Python	73,3 MB	1
Python	73,0 MB	1
Python	63,9 MB	1
Python	73,6 MB	1
Python	72,8 MB	1
Python	66,6 MB	1
Python	65,5 MB	1
Python	74,7 MB	1
Python	7,5 MB	1
powerd	4,7 MB	3
pkd	3,6 MB	2
PerfPowerServices	12,3 MB	6

PyData

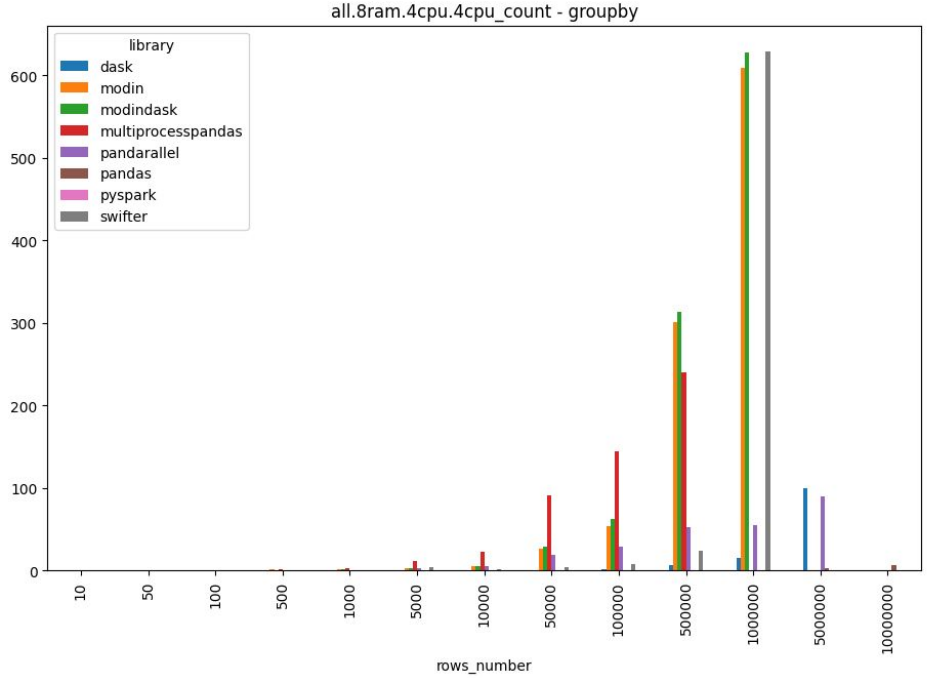
Results

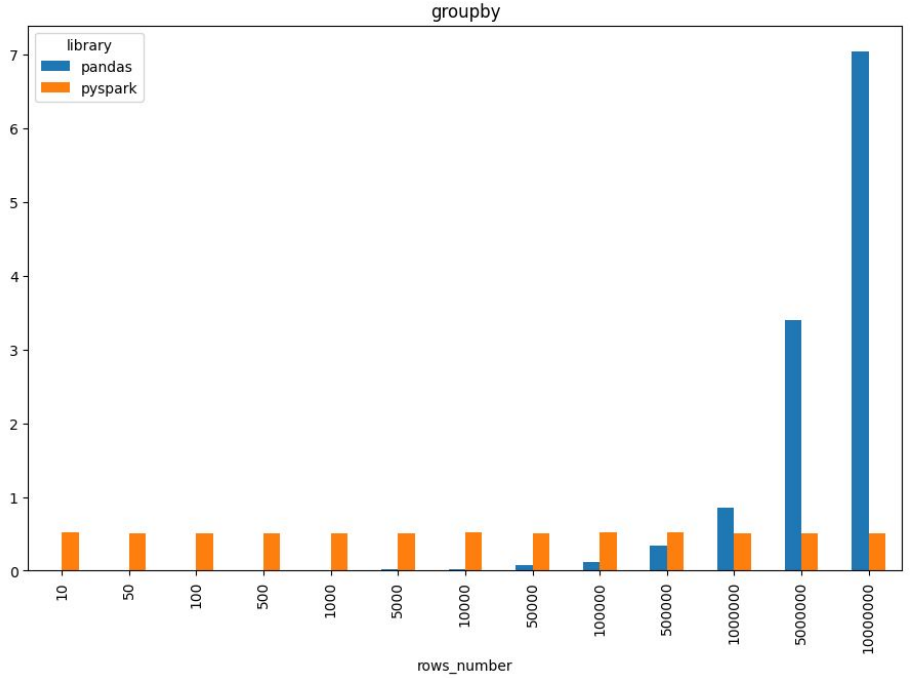




PyData

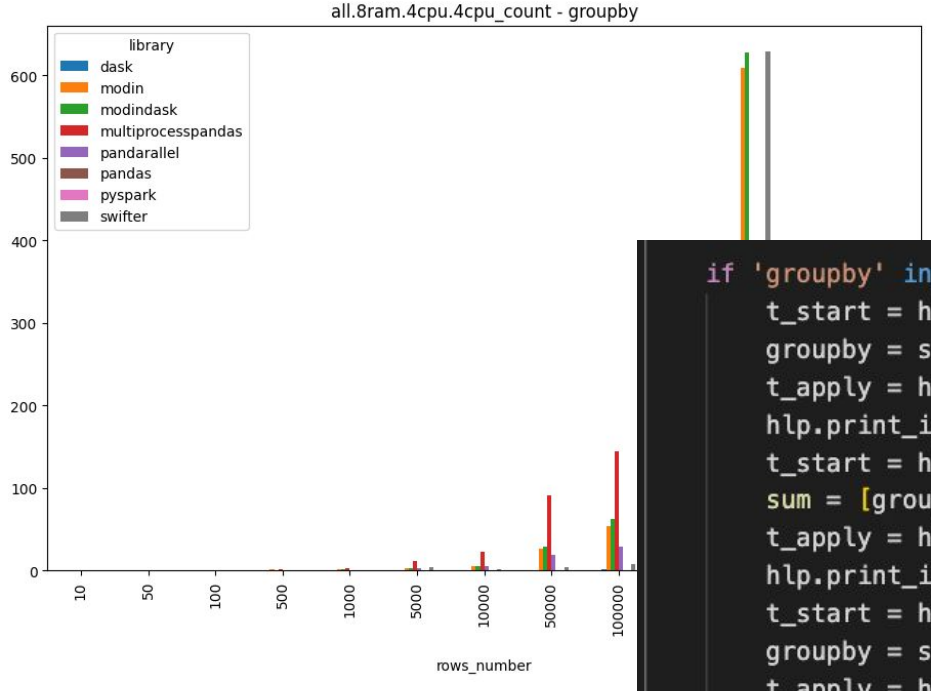
Results

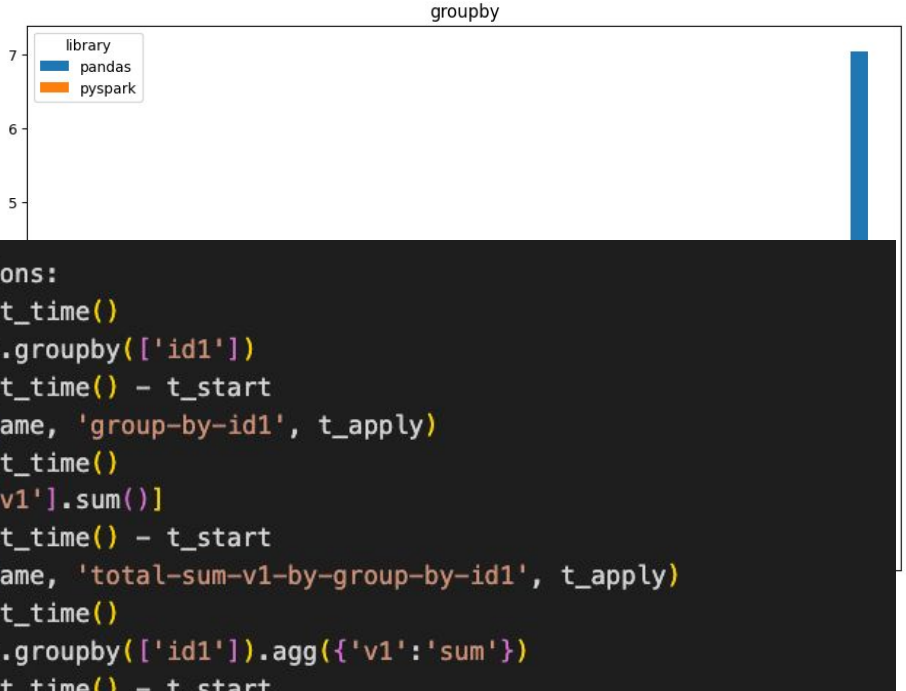




PyData

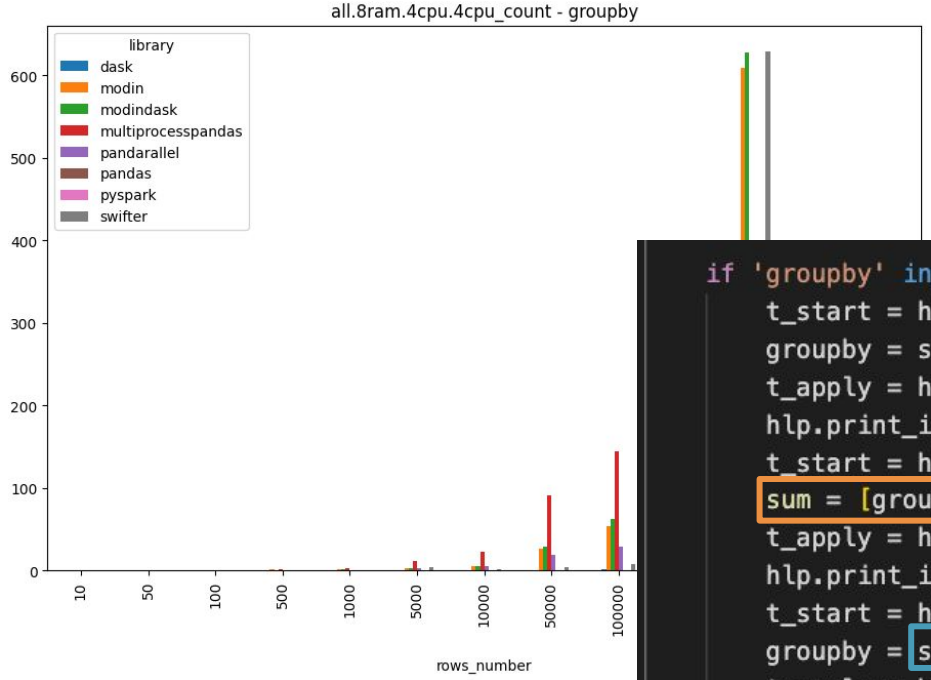
Results





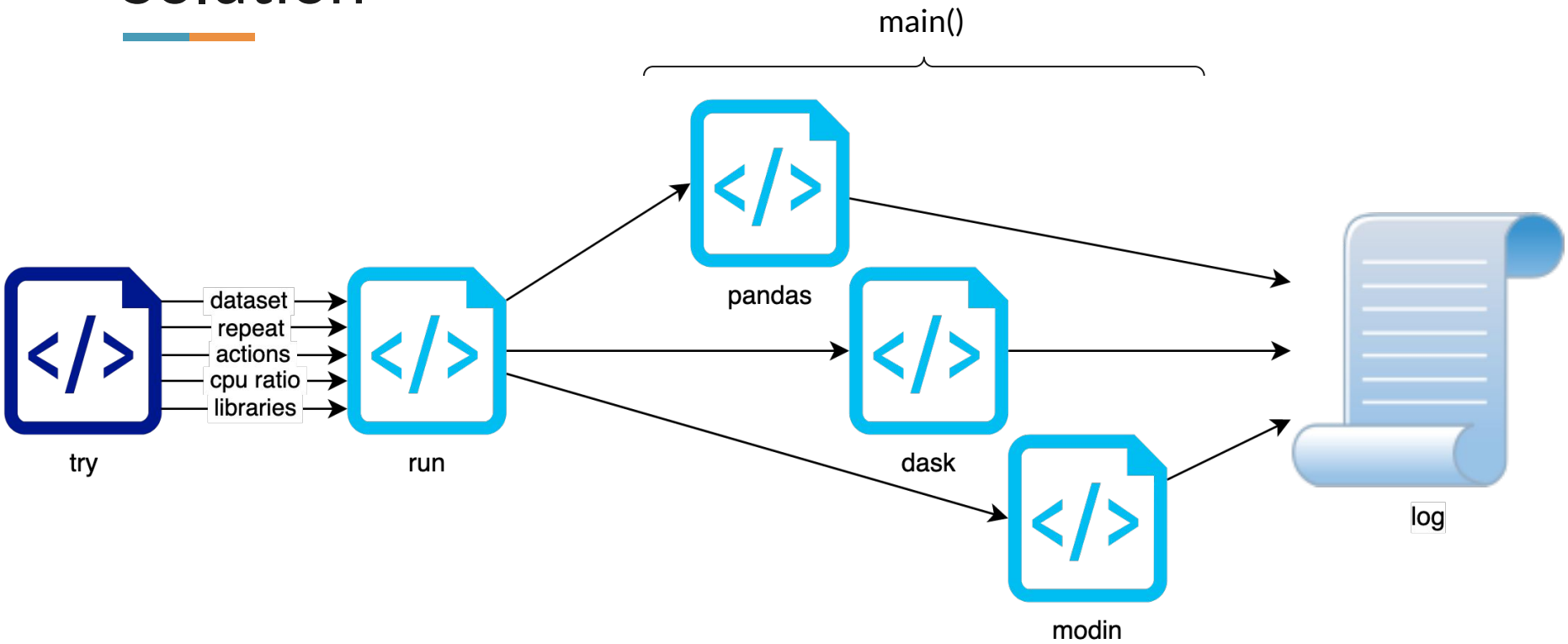

```
if 'groupby' in actions:
    t_start = hlp.get_time()
    groupby = sample.groupby(['id1'])
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'group-by-id1', t_apply)
    t_start = hlp.get_time()
    sum = [groupby['v1'].sum()]
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'total-sum-v1-by-group-by-id1', t_apply)
    t_start = hlp.get_time()
    groupby = sample.groupby(['id1']).agg({'v1':'sum'})
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'sum-v1-by-id1', t_apply)
```

Results

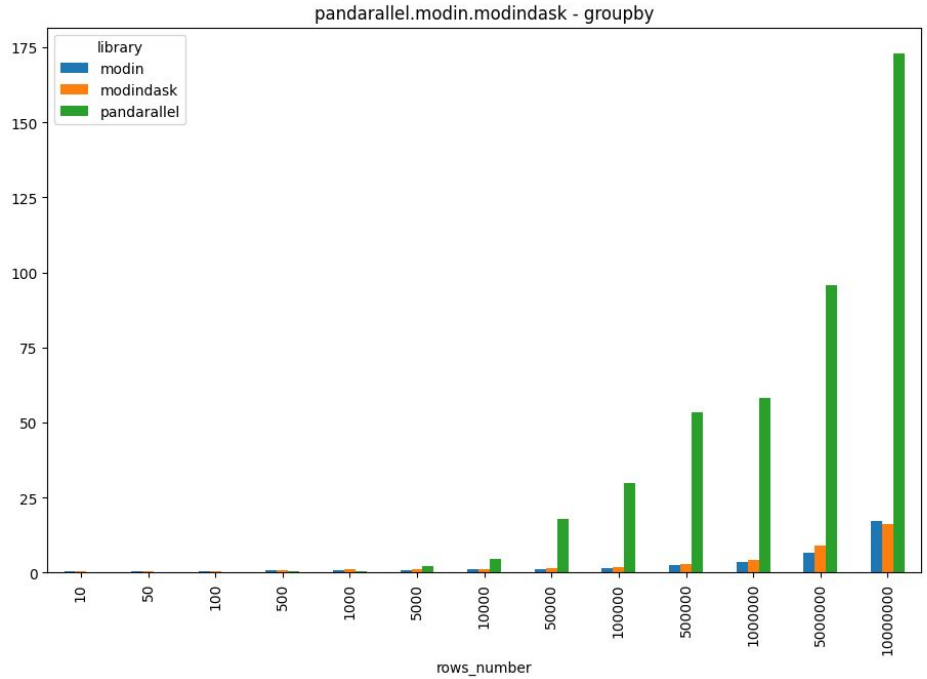


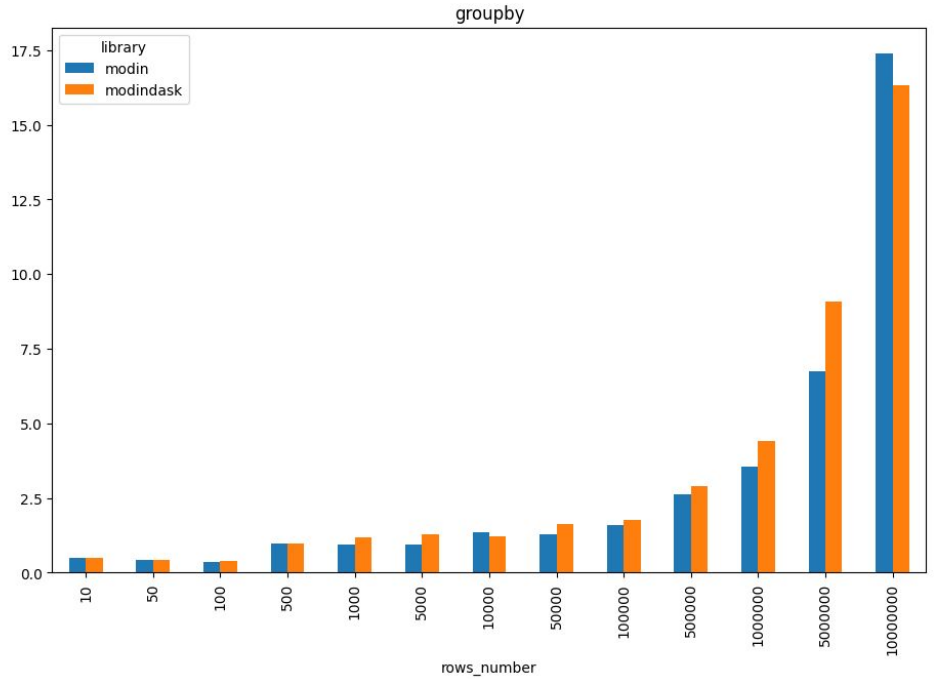

```
if 'groupby' in actions:
    t_start = hlp.get_time()
    groupby = sample.groupby(['id1'])
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'group-by-id1', t_apply)
    t_start = hlp.get_time()
    sum = [groupby['v1'].sum()]
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'total-sum-v1-by-group-by-id1', t_apply)
    t_start = hlp.get_time()
    groupby = sample.groupby(['id1']).agg({'v1':'sum'})
    t_apply = hlp.get_time() - t_start
    hlp.print_info(name, 'sum-v1-by-id1', t_apply)
```

Solution

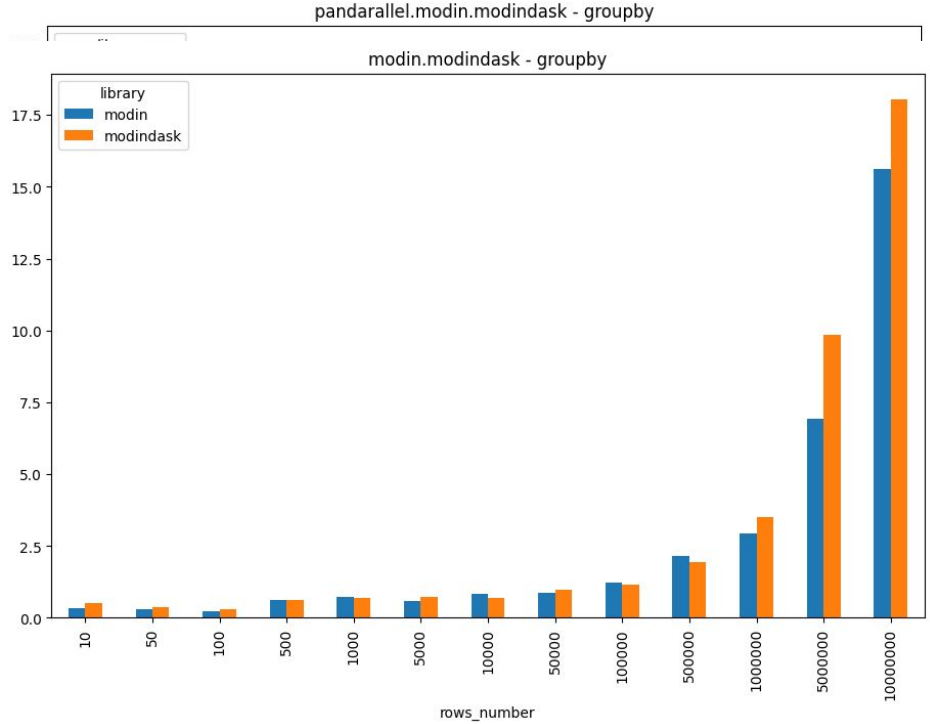

main()
dataset
repeat
actions
cpu ratio
libraries
try
run
pandas
dask
modin
log

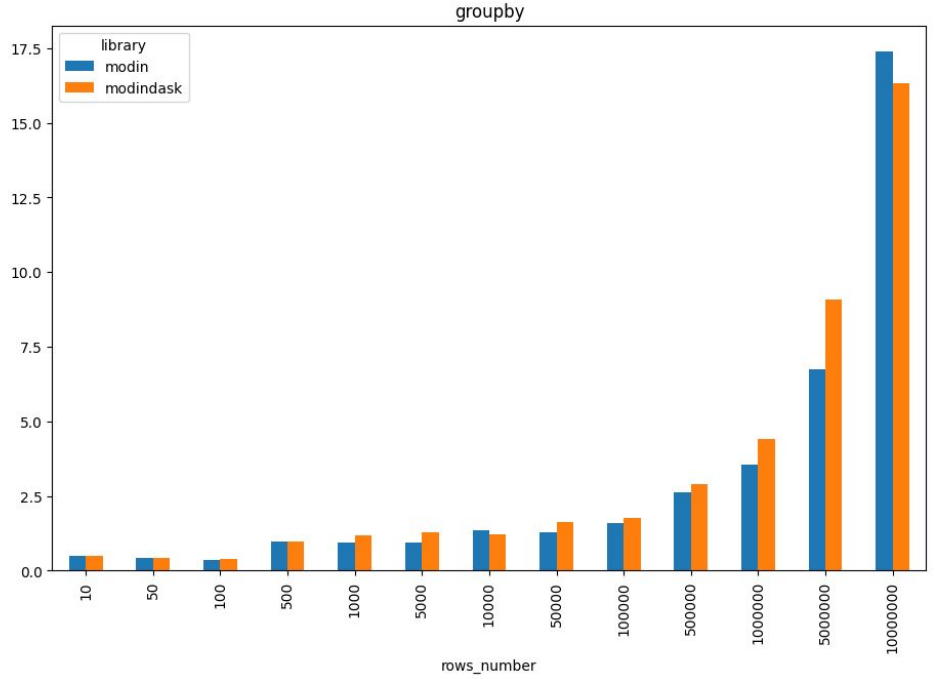
Results


pandarallel.modin.modindask - groupby
library
modin
modindask
pandarallel
175
150
125
100
75
50
25
0
10
50
100
500
1000
5000
10000
50000
100000
500000
1000000
5000000
10000000
rows_number


groupby
library
modin
modindask
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
10
50
100
500
1000
5000
10000
50000
100000
500000
1000000
5000000
10000000
rows_number

Results


pandarallel.modin.modindask - groupby
modin.modindask - groupby
library
modin
modindask
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
10
50
100
500
1000
5000
10000
50000
100000
500000
1000000
5000000
10000000
rows_number


groupby
library
modin
modindask
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
10
50
100
500
1000
5000
10000
50000
100000
500000
1000000
5000000
10000000
rows_number


Segmenti casuali di codice o è qualcosa di più ..
moviescreenshot.blogspot.com

Results

all.8ram.4cpu.4cpu_count

	library	fast	load	series	slow	groupby	to
0	concurrent		36.016				36.016
1	dask	0.008	94.16	18.537			112.705
2	joblib		37.048	42.516			79.564
3	joblibdask		91.15				91.15
4	modin	0.07	47.284	0.517	1.865		47.871
5	modindask	0.081	65.64	0.27	2.018		65.991
6	multiprocessing		36.008	0.064			36.072
7	multiprocesspandas	0.038	61.0	11.709			72.747
8	pandarallel	0.031	36.026	1.823			37.879
9	pandas	0.049	82.752	11.53		7.042	101.373
10	parallelize	0.036	36.851	26.151			63.038
11	pyspark	0.054	95.344	5.27	0.016	0.508	101.176
12	swifter	0.071	86.212	2.005			88.287

modin.modindask

	library	load	groupby	total
0	modin	50.226	15.617	65.844
1	modindask	53.744	18.043	71.787

Results

modin.modindask

	library	load	groupby	total
0	modin	50.226	15.617	65.844
1	modindask	53.744	18.043	71.787

all.8ram.4cpu.4cpu_count

	library	fast	load	series	slow	groupby	to
0	concurrent		36.016				36.016
1	dask	0.008	94.16	18.537			112.705
2	joblib		37.048	42.516			79.564
3	joblibdask		91.15				91.15
4	modin	0.07	47.284	0.517	1.865		47.871
5	modindask	0.081	65.64	0.27	2.018		65.991
6	multiprocessing		36.008	0.064			36.072
7	multiprocesspandas	0.038	61.0	11.709			72.747
8	pandarallel	0.031	36.026	1.823			37.879
9	pandas	0.049	82.752	11.53		7.042	101.373
10	parallelize	0.036	36.851	26.151			63.038
11	pyspark	0.054	95.344	5.27	0.016	0.508	101.176
12	swifter	0.071	86.212	2.005			88.287

Results

modin.modindask

	library	load	groupby	total
0	modin	50.226	15.617	65.844
1	modindask	53.744	18.043	71.787

all.8ram.4cpu.4cpu_count

	library	fast	load	series	slow	groupby	to
0	concurrent		36.016				36.016
1	dask	0.008	94.16	18.537			112.705
2	joblib		37.048	42.516			79.564
3	joblibdask		91.15				91.15
4	modin	0.07	47.284	0.517	1.865	15.617	47.871
5	modindask	0.081	65.64	0.27	2.018	18.043	65.991
6	multiprocessing		36.008	0.064			36.072
7	multiprocesspandas	0.038	61.0	11.709			72.747
8	pandarallel	0.031	36.026	1.823			37.879
9	pandas	0.049	82.752	11.53		7.042	101.373
10	parallelize	0.036	36.851	26.151			63.038
11	pyspark	0.054	95.344	5.27	0.016	0.508	101.176
12	swifter	0.071	86.212	2.005			88.287

Results

modin.modindask

	library	load	groupby	total
0	modin	50.226	15.617	65.844
1	modindask	53.744	18.043	71.787

all.8ram.4cpu.4cpu_count

	library	fast	load	series	slow	groupby	to
0	concurrent		36.016				36.016
1	dask	0.008	94.16	18.537			112.705
2	joblib		37.048	42.516			79.564
3	joblibdask		91.15				91.15
4	modin	0.07	47.284	0.517	1.865	15.617	63.488
5	modindask	0.081	65.64	0.27	2.018	18.043	84.034
6	multiprocessing		36.008	0.064			36.072
7	multiprocesspandas	0.038	61.0	11.709			72.747
8	pandarallel	0.031	36.026	1.823			37.879
9	pandas	0.049	82.752	11.53		7.042	101.373
10	parallelize	0.036	36.851	26.151			63.038
11	pyspark	0.054	95.344	5.27	0.016	0.508	101.176
12	swifter	0.071	86.212	2.005			88.287

may

data power

with you

PyData

Oddities

Goal:
performance

Challenge:
on 4 CPU & 8GB RAM

- pandas takes longer
  - than libs who use it
- pandas generate more
  - pids & threads than pyspark
- modin generate more
  - pids & threads than others
- modin works better if ..

PyData

Best practies

Goal:
performance

Challenge:
on 4 CPU & 8GB RAM

- vectorialization
  - better than apply
- multiprocessing
- pay attention
  - df.copy()
  - groupby / sum / apply

PyData

Thanks for listening.

@PyDataVenice #13 #Meetup #PyData