Making Pandas Fly (live from London)

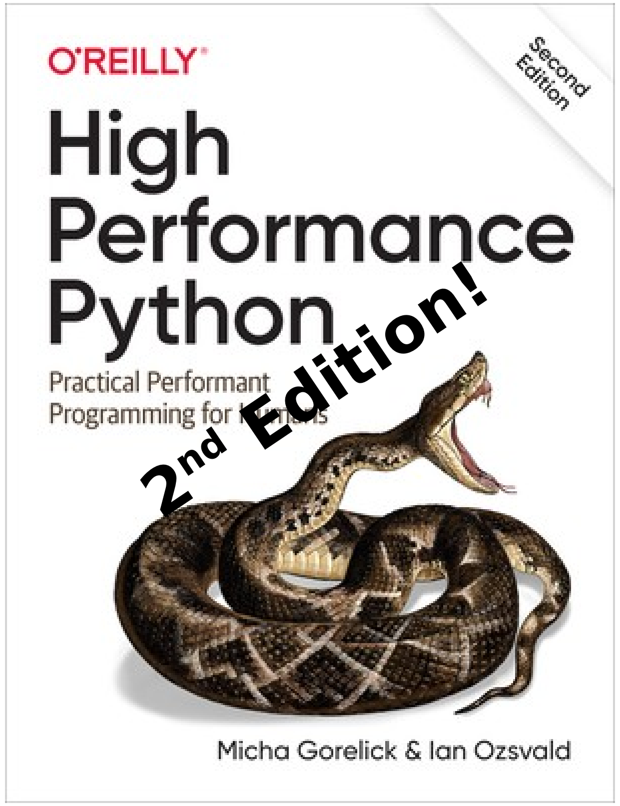
EuroPython 2020

Ian Ozsvald

@IanOzsvald – ianozsvald.com

Introductions

- Interim Chief Data Scientist
- 19+ years experience
- Team coaching & public courses
  - I'm sharing from my Higher Performance Python course



Thank the organisers!

- All volunteers – go say thank you in #lobby
- They've put in a huge amount of volunteered work for us!

Today's goal

- Pandas
  - Saving RAM to fit in more data
  - Calculating faster by dropping to Numpy
- Advice for "being highly performant"
- Has Covid 19 affected UK Company Registrations?

Strings are expensive and slow

```
display(df.CompanyCategory.value_counts()[:10])
```

```
Private Limited Company                                                                      4294231
PRI/LTD BY GUAR/NSC (Private, limited by guarantee, no share capital)                         100365
Limited Liability Partnership                                                                  51750
PRI/LBG/NSC (Private, Limited by guarantee, no share capital, use of 'Limited' exemption)      40534
Community Interest Company                                                                     20189
Other company type                                                                              8492
Public Limited Company                                                                          5935
Private Unlimited Company                                                                       4106
Registered Society                                                                               151
Industrial and Provident Society                                                                 130
Name: CompanyCategory, dtype: int64
```

```
f"{df['CompanyCategory'].memory_usage(deep=True, index=False):,} bytes"
```

```
'369,713,766 bytes'
```

Categoricals are cheap and fast!

```
df['CompanyCategory_cat'] = df.CompanyCategory.astype('category')
df['CompanyCategory_cat'].value_counts()[:10]
```

```
Private Limited Company                                                                    4294231
PRI/LTD BY GUAR/NSC (Private, limited by guarantee, no share capital)                       100365
Limited Liability Partnership                                                                 51750
PRI/LBG/NSC (Private, Limited by guarantee, no share capital, use of 'Limited' exemption)    40534
Community Interest Company                                                                    20189
Other company type                                                                             8492
Public Limited Company                                                                         5935
Private Unlimited Company                                                                      4106
Registered Society                                                                              151
Industrial and Provident Society                                                                130
Name: CompanyCategory_cat, dtype: int64
```

```
f"{df['CompanyCategory_cat'].memory_usage(deep=True, index=False):,} bytes"
```

```
'4,528,328 bytes'
```

Circa 1% of previous memory cost

Categoricals ".cat" accessor

```
df['CompanyCategory_cat'].cat.categories
```

```
Index(['Community Interest Company',
       'European Public Limited-Liability Company (SE)',
       'Industrial and Provident Society',
       'Investment Company with Variable Capital(Umbrella)'
       'Limited Liability Partnership', 'Limited Partnershi
       'Old Public Company', 'Other Company Type', 'Other c
       'PRI/LBG/NSC (Private, Limited by guarantee, no shar
       'PRI/LTD BY GUAR/NSC (Private, limited by guarantee,
       'PRIV LTD SECT. 30 (Private limited company, section
       'Private Limited Company', 'Private Unlimited',
       'Private Unlimited Company', 'Public Limited Company
       'Registered Society'],
      dtype='object')
```

```
df['CompanyCategory_cat'].cat.codes
```

```
CompanyNumber
08209948    12
11399177    12
11743365    12
12402527    12
12234705    12
            ..
```

Categoricals – over 10x speed up (on this data)!

```
%timeit df['CompanyCategory'].value_counts()
```

```
485 ms ± 52.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%timeit df['CompanyCategory_cat'].value_counts() # HUGE SAVING
```

```
28.6 ms ± 3.91 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Categoricals – index queries faster!

```
df2_nocat = df.set_index('CompanyCategory')
df2_cat = df.set_index('CompanyCategory_cat')
```

```
%timeit (df2_nocat.index == 'Private Limited Company')
```

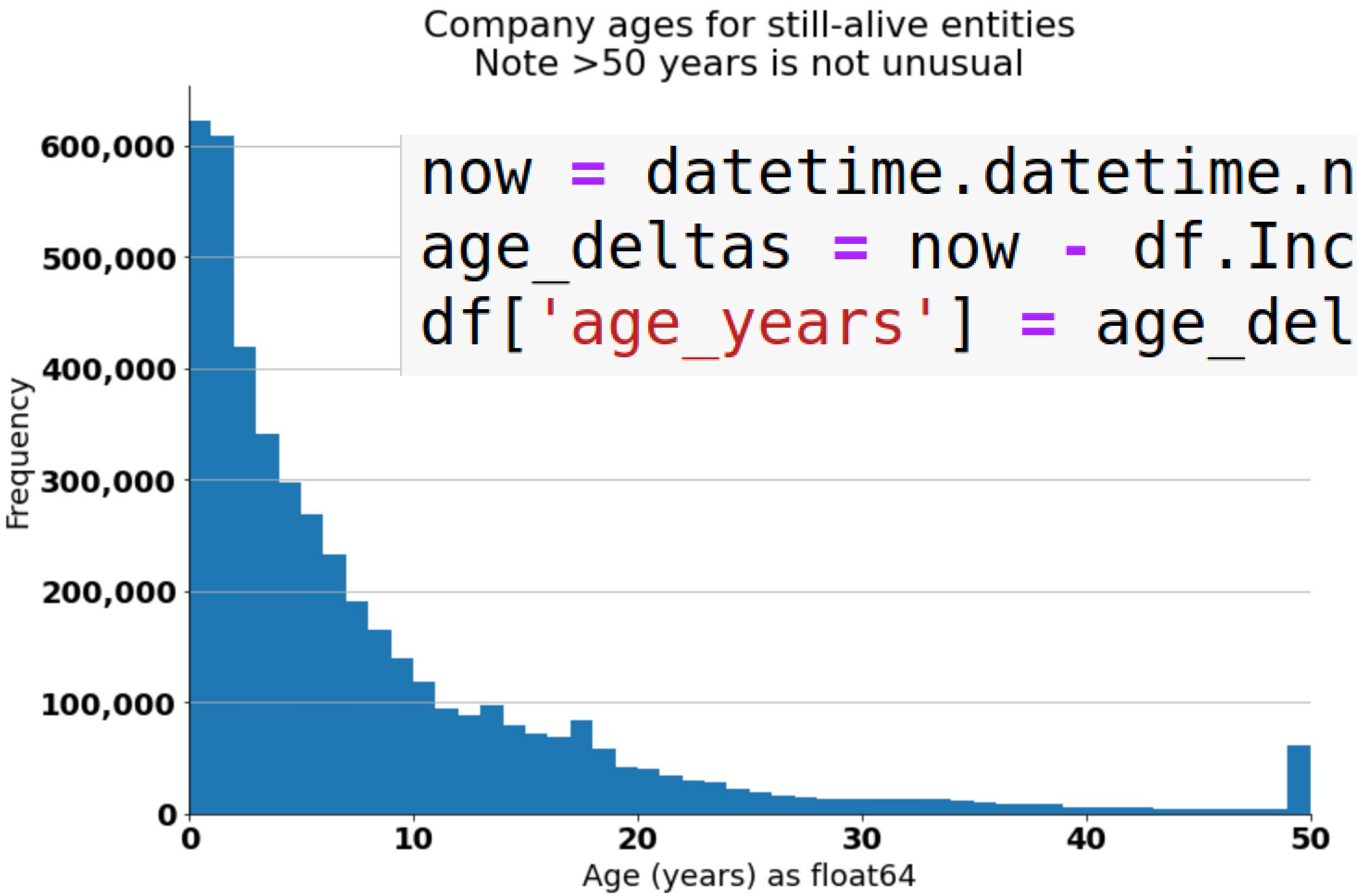
281 ms ± 3.35 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

```
%timeit (df2_cat.index == 'Private Limited Company') # HUGE SAVING!
```

569 µs ± 223 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

Circa 500x speed-up!

float64 is default and a bit expensive




```
now = datetime.datetime.now()
age_deltas = now - df.IncorporationDate
df['age_years'] = age_deltas.dt.days / 360
```

float32 “half-price” and a bit faster

```
%timeit ser64.std()
```

```
15.2 ms ± 8.29 µs per loop
```

```
%timeit ser32.std()
```

```
14.6 ms ± 16.4 µs per loop
```

```
ser64 = df.age_years
show_size(ser64)
ser32 = ser64.astype("float32")
show_size(ser32)
ser16 = ser64.astype("float16") # NOTE hardware _interpreted_ so SLOW
show_size(ser16)
```

```
Type float64 Range 0.0528 - 190.8972, float64, 36,208,992 bytes
Type float32 Range 0.0528 - 190.8972, float32, 18,104,496 bytes
Type float16 Range 0.0528 - 190.8750, float16, 9,052,248 bytes
```

Make choices to save RAM

```
df_original_ram = df[['CompanyCategory', 'age_years']]
df_original_ram.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4526124 entries, 08209948 to 07835383
Data columns (total 2 columns):
 #   Column           Dtype
---  ------           -----
 0   CompanyCategory  object
 1   age_years        float64
dtypes: float64(1), object(1)
memory usage: 827.7 MB
```

Including the index (previously we ignored it) we still save circa 50% RAM so you can fit in more rows of data

```
df_smaller_ram = df[['CompanyCategory_cat', 'age_years_f32']]
df_smaller_ram.info(memory_usage="deep")
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4526124 entries, 08209948 to 07835383
Data columns (total 2 columns):
 #   Column               Dtype
---  ------               -----
 0   CompanyCategory_cat  category
 1   age_years_f32        float32
dtypes: category(1), float32(1)
memory usage: 462.2 MB
```

“dtype_diet” gives you advice

```
res = dtype_diet.report_on_dataframe(df_sample).drop(columns=['col'])
res['saved_mb'] = ((res.current_nbytes - res.nbytes) / 1_000_000).astype('int')
res['shrunk_to'] = (res.nbytes / res.current_nbytes * 100).astype('int')
res.style.format({'nbytes': "{:,}", 'current_nbytes': '{:,}',
                  'saved_mb': "{}MB", 'shrunk_to': '{:}%'})
```

```
Smallest non-breaking conversion per column:
```

	column	dtype	nbr_different	nbytes	current_nbytes	saved_mb	shrunk_to
0	CompanyName	category	0	552,546,573	366,681,836	-185MB	150%
1	CompanyCategory	category	0	4,528,328	369,713,766	365MB	1%
2	Accounts.AccountRefDay	float16	0	9,052,248	36,208,992	27MB	25%
3	Accounts.AccountRefDay	float32	0	18,104,496	36,208,992	18MB	50%

ianozsvald / **dtype_diet**

Drop to NumPy if you know you can

```
%%timeit
df['age_years'].sum()
```

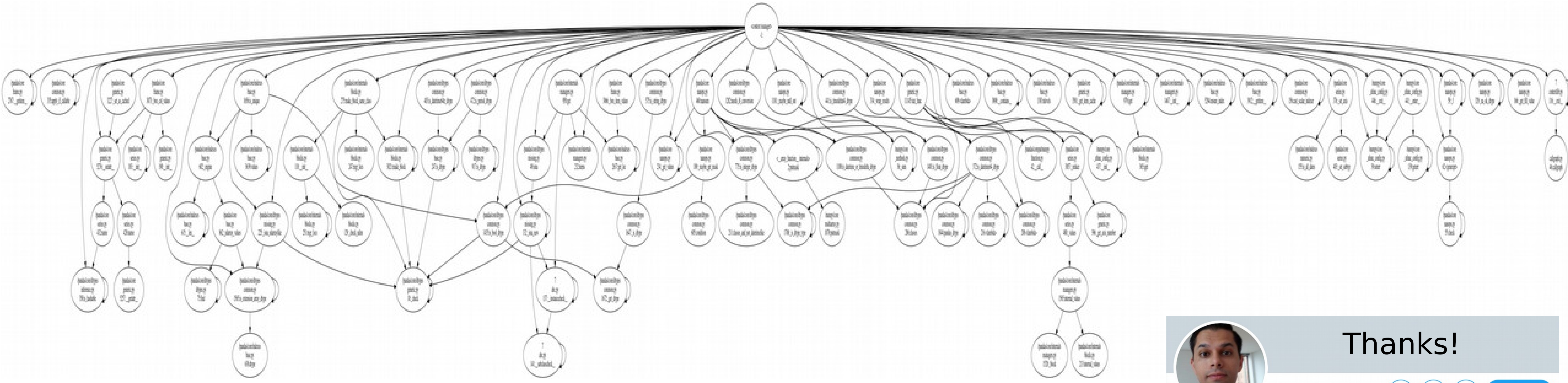
```
19.1 ms ± 1.58 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%%timeit
df['age_years'].values.sum()
# 10X SAVING DROPPING - IF YOU _KNOW_ YOU CAN (see slides for some details)
```

```
2.44 ms ± 68.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

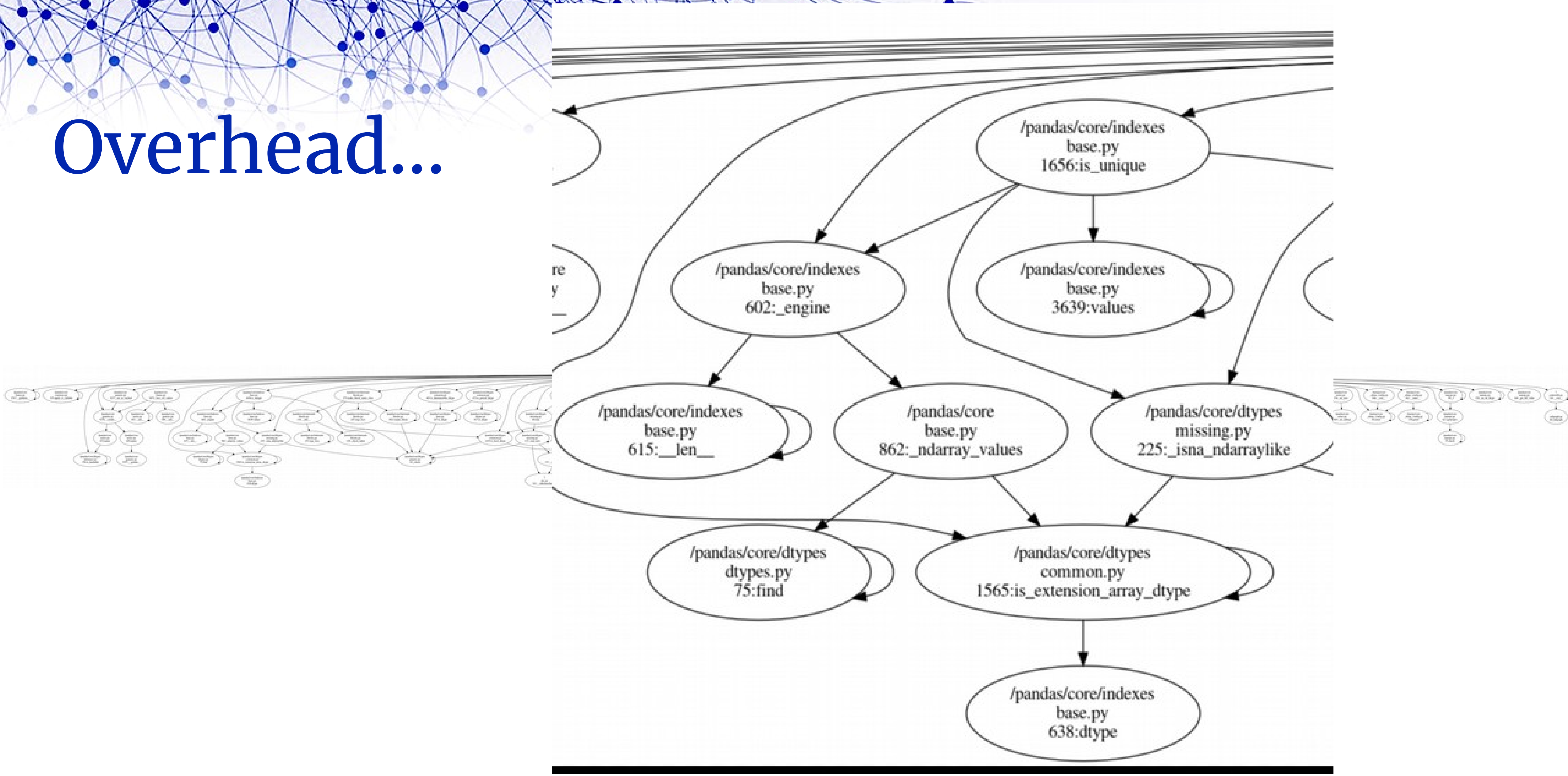
Caveat – Pandas mean is not np mean, the fair comparison is to np nanmean which is slower – see my blog or PyDataAmsterdam 2020 talk for details

NumPy vs Pandas overhead (ser.sum())

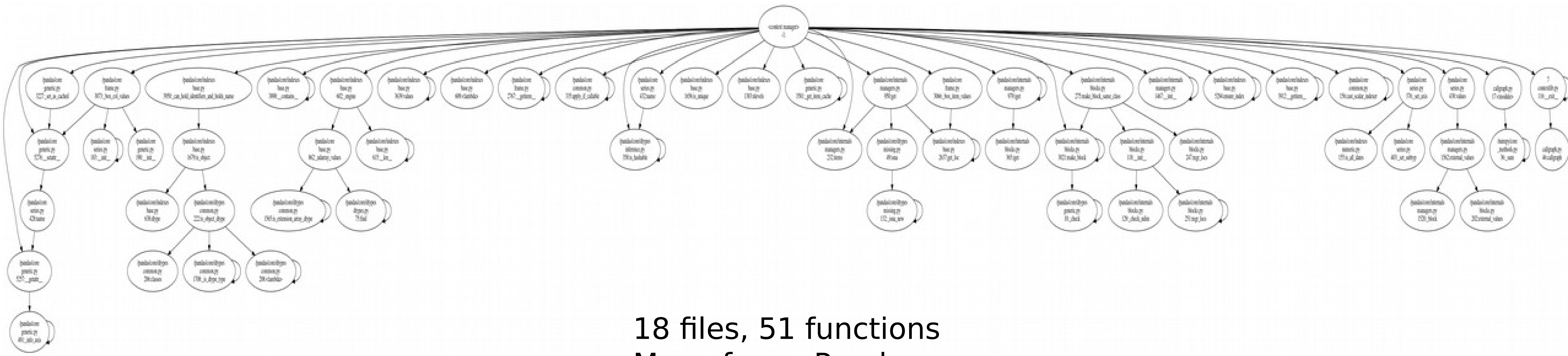


25 files, 83 functions
Very few NumPy
calls!

Overhead...


/pandas/core/indexes
base.py
1656:is_unique
/pandas/core/indexes
base.py
602:_engine
/pandas/core/indexes
base.py
3639:values
/pandas/core/indexes
base.py
615:__len__
/pandas/core
base.py
862:_ndarray_values
/pandas/core/dtypes
missing.py
225:_isna_ndarraylike
/pandas/core/dtypes
dtypes.py
75:find
/pandas/core/dtypes
common.py
1565:is_extension_array_dtype
/pandas/core/indexes
base.py
638:dtype

Overhead with ser.values.sum()



18 files, 51 functions
Many fewer Pandas
calls (but still a lot!)

Is Pandas unnecessarily slow – NO!

```
In [15]: pd.options.compute.use_bottleneck=False

In [16]: %timeit ser.mean()
54.4 ms ± 228 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [17]: pd.options.compute.use_bottleneck=True

In [18]: %timeit ser.mean()
16.7 ms ± 30.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

In [19]: %timeit ser.values.mean()
5.05 ms ± 75.9 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

https://github.com/pandas-dev/pandas/issues/34773 -
the truth is a bit complicated!

Being highly performant

- Install optional (but great!) Pandas dependencies
  - bottleneck
  - numexpr

https://pandas.pydata.org/pandas-docs/stable/user_guide/enhancingperf.html

- Investigate https://github.com/ianozsvald/dtype_diet
- Investigate my ipython_memory_usage (PyPI/Conda)

Pure Python is “slow” and expressive

Deliberately poor function – pretend this is clever but slow!

```
%timeit df['age_years'].apply(won_huge_project)
```

```
13.2 s ± 21.6 ms per loop (mean ± std. dev. of
```

```
def won_huge_project(age):
    """Older companies have a higher chance
       of winning a low-probability big project"""
    flag = False
    for n in range(int(age)):
        if random.uniform(0, 1) > 0.9:
            flag = True
    return flag
df_sample = df.sample(5)
ser = df_sample['age_years'].apply(won_huge_project) # example output
```

	IncorporationDate	age_years	won_huge_prj
CompanyNumber			
08148215	2012-07-18	8.119444	True
10596616	2017-02-02	3.508333	False

Compile to Numba judiciously

```
%timeit df['age_years'].apply(won_huge_project_numba)
1.5 s ± 2.05 ms per loop (mean ± std. dev. of 7 runs,
```

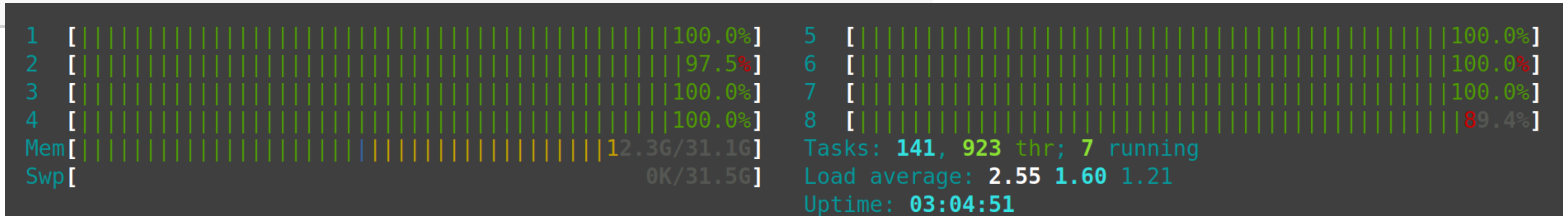
Near 10x speed-up!

```
from numba import njit
@njit()
def won_huge_project_numba(age):
    """Older companies have a higher chance
       of winning a low-probability big project"""
    flag = False
    for n in range(int(age)):
        if random.uniform(0, 1) > 0.9:
            flag=True
    return flag
```

Parallelise with Dask for multi-core

```python
import dask.dataframe as dd
cols = ['age_years']
# NOTE the reset_index on text column!
ddf = dd.from_pandas(df[cols].reset_index(drop=True),
                     npartitions=8, sort=False)
t1 = time.time()
result = ddf.apply(won_huge_project, axis=1,
                   raw=True, meta=(None, 'bool')) \
  .compute(scheduler='processes')
print(f"Took {time.time() - t1}")
```

```
Took 5.533645868301392
```



- Make plain-Python code multi-core
- Note I had to drop text index column due to speed-hit
- Data copy cost can overwhelm any benefits so (always) profile & time

Being highly performant

- Mistakes slow us down (PAY ATTENTION!)
  - Try **nullable** Int64 & boolean, forthcoming Float64
  - Write tests (unit & end-to-end)
  - Lots more material & my newsletter on my blog IanOzsvald.com
  - Time saving docs: ianozsvald / **notes_to_self**

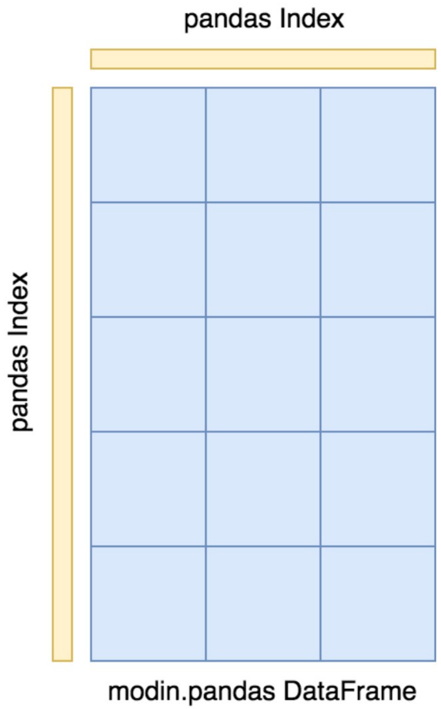
Vaex / Modin

- Memory mapped & lazy computation
  - New string dtype (RAM efficient)
- Modin sits on Pandas, new “algebra” for dfs
  - Drop in replacement, easy to try

ML impossible: Train 1 billion samples in 5 minutes on your laptop using Vaex and Scikit-Learn

Make your laptop feel like a supercomputer.

Jovan Veljanoski Follow
Jan 22 · 15 min read

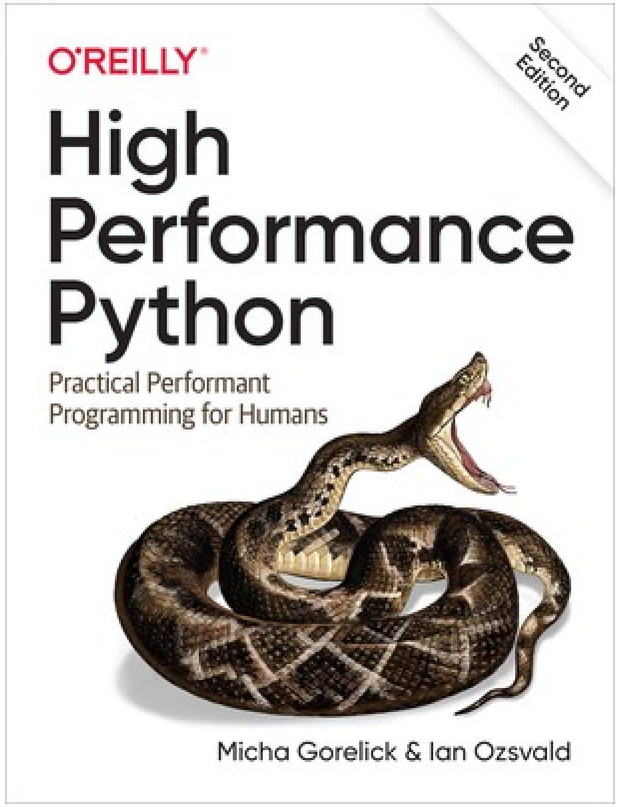


See talks on my blog: “Flying Pandas” and “Making Pandas Fly” – virtual talks this weekend on faster data processing with Pandas, Modin, Dask and Vaex

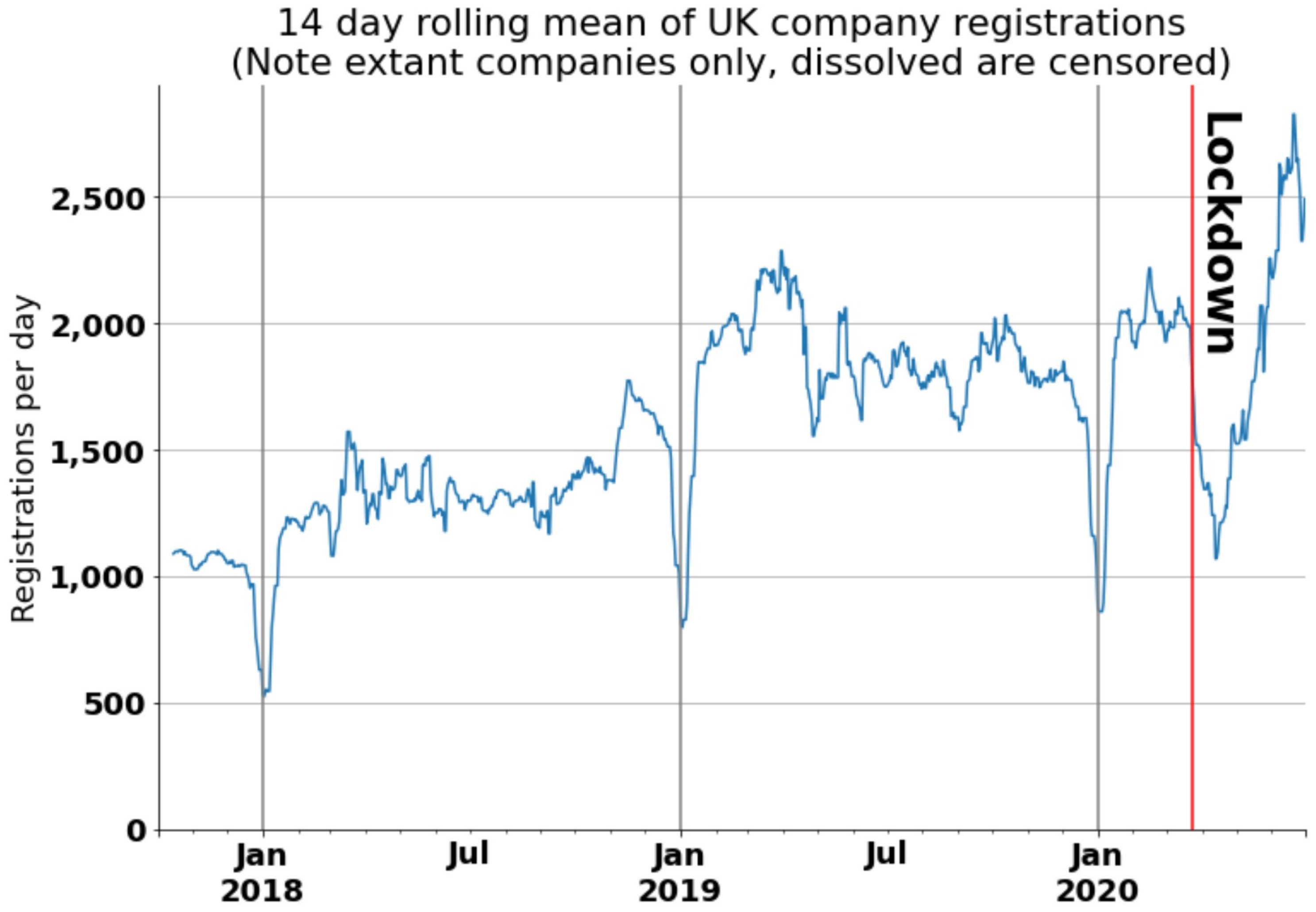
High Performance Python Book, pydata, Python | April 27, 2020

Summary

- Make it right then make it fast
- Think about *being* performant
- See blog for my classes
- I’d love a postcard if you learned something new!



Covid 19's effect on UK Economy?



Sharp decline in corporate registration after Lockdown – then apparent surge (perhaps just backed-up paperwork?). Will the recovery "last"? All **open data**, you can do similar things!