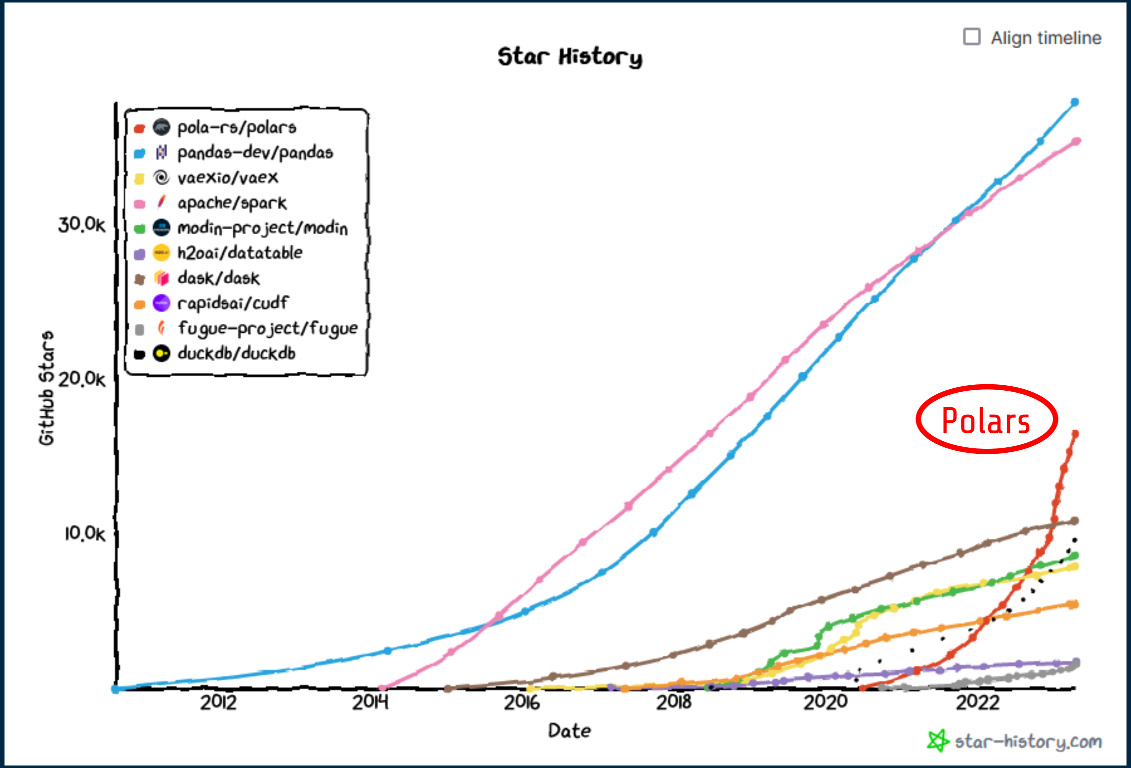
LIGHTNING FAST DATAFRAMES WITH POLARS

Going beyond Pandas
Overview and performance

Alberto Danese

One question: why all the hype?



https://star-history.com/#pola-rs/polars&pandas-dev/pandas&vaexio/vaex&apache/spark&modin-project/modin&h2oai/datatable&dask/dask&rapidsai/cudf&fugue-project/fugue&duckdb/duckdb&Date

About me

Alberto Danese
Head of Data Science

nexi

www.linkedin.com/in/albertodanese

Computer Engineer (Politecnico di Milano)
15+ years in data & tech, mostly in financial services

Competitions Grandmaster on kaggle

I write regularly on:
allaboutdata.substack.com
A²D

Speaker at AWS Re:Invent, Google, Codemotion, Kaggle and other data & tech events

eBook and paperback

Dataframes

- Working with data in Python used to be an **easy choice**!

- Does the data fit in your machine RAM? Pandas!
- It doesn't? (py)Spark

The good

- A **large ecosystem** – a pandas dataframe is what most libraries in the data and ML field expect
- A **huge community** – with 1000s of contributors with code, documentation, guide, tutorials
- A relatively **stable API** – as many projects depend on it

- All of this have to be expected: it's the de facto standard for Python dataframes, developed since 2008

- It would be too long to list all of Spark's benefits, as it's much more than a DF library, but when it comes to handling data, it provides:

- **Horizontal scaling** – you can add computation at need
- A set of tools to deal with data, starting from SparkSQL to a proper adoption of the pandas API (*koalas* has been integrated in the pySpark codebase since 3.2)

The not-so good

pandas

- **Limited scaling** – begin designed as single threaded severely limits performances
- **Questionable syntax** – you may like it or not, but it easily gets messy

APACHE PySpark

- It's **complicated**! (this is also the reason why many love it)
- Sometimes you'd just avoid the **complexity** of handling a cluster unless it's really needed

Most of the time, we are somehow **in the middle**: the data is **not big enough for Spark, but too big for Pandas**

Memo: 96 vCPU and 768GB of ram cost just 8$/hour on major Cloud providers

Dataframes world in 2023

polars

RAY

DuckDB

DASK

datatable

MODIN

fugue

vaex

cuDF (RAPIDS)

THE DATAFRAME LIBRARY FOR GPU DATA SCIENCE

Dataframes world in 2023

RAY

DASK

Same field of Spark:
computational frameworks that allow horizontal scaling and distributing the workloads across a cluster of machines

Dataframes world in 2023

Memory mapping alternative (not to load a df in memory), apparently not developed since December 2022

vaex

MODIN

Wannabe drop-in replacement of Pandas with a single line of code, providing parallelism

Dataframes world in 2023

Porting of the R library by H2O.ai team, very concise and fast… once it was the fastest around

datatable

Fast dataframe library, if you have a GPU and the GPU ram is enough

cuDF (RAPIDS)

THE DATAFRAME LIBRARY FOR GPU DATA SCIENCE

Dataframes world in 2023

DuckDB

Fast and intuitive in-process SQL-based OLAP DBMS, for Python and more

fugue

Semantic layer providing abstractions to distribute pandas, plain sql, polars workload on different kind of clusters: spark, ray, dask

Dataframes world in 2023

polars

- Designed from **scratch** (from early 2020), initially to provide a dataframe library to the **Rust** ecosystem
- Built on top of **Arrow** for efficiency
- Written in Rust, but available with bindings for **Python** as well

- Personal project of Ritchie Vink that got a bit out of hand: 16.000+ stars on Github, 6.000+ commits (still 70% by the original author) in just 3 years!

Why Polars?

SPEED

Often an **order of magnitude** (or more) **faster** than Pandas, plus **lazy evaluation** and **larger-than-memory** data support

SYNTAX

Pure **pythonic** syntax, just **intuitive** and **expressive**

The ex-H2O.ai db benchmark

- In Mid-april 2023, DuckDB forked the original H2O.ai **db benchmark** (stuck in 2021) and ran several analytical workloads on 10 libraries, with different data size (0.5GB, 5GB, 50GB) and families of operations (mainly **groupby** and **join**)

- The code is open, here: https://duckdblabs.github.io/db-benchmark/

Some results

Task

groupby | join | groupby2014

0.5 GB | 5 GB | 50 GB

Polars 13x faster than Pandas 2.0 (with arrow)

basic questions

Input table: 100,000,000 rows x 9 columns (5 GB)

duckdb-latest	0.8.0	2023-04-13	6s
DuckDB	0.7.1	2023-04-05	8s
Polars	0.16.18	2023-04-05	9s
ClickHouse	22.12.1.1752	2023-03-24	13s
data.table	1.14.9	2023-03-24	16s
spark	3.3.2	2023-03-24	37s
Arrow	11.0.0.3	2023-04-12	37s
(py)datatable	1.1.0a0	2023-03-24	77s
pandas	2.0.0	2023-04-07	116s
dask	2023.3.2	2023-04-07	124s
dplyr	1.0.10	2022-12-30	257s
Modin		see README	pending

Task

groupby | join | groupby2014

0.5 GB | 5 GB | 50 GB

Polars 16x faster than Pandas 2.0 (with arrow)

basic questions

Input table: 100,000,000 rows x NA columns (NA GB)

duckdb-latest	0.8.0	2023-04-12	22s
DuckDB	0.7.1	2023-04-05	23s
Polars	0.16.18	2023-04-05	42s
data.table	1.14.9	2023-03-24	112s
ClickHouse	22.12.1.1752	2023-03-24	187s
spark	3.3.2	2023-03-24	469s
dplyr	1.0.10	2022-12-30	498s
pandas	2.0.0	2023-04-07	666s
(py)datatable	1.1.0a0	2023-03-24	10903s
dask	2023.3.2	2023-04-07	internal error
Arrow	11.0.0.3	2023-04-12	out of memory
Modin		see README	pending

Actually Polars 0.17.x was released just a few days after this benchmark

Key features: eager vs. lazy

Eager evaluation

- What we are used to (in pandas aswell): each command gets **executed right away**, line-by-line
- Nothing else: as simple as that!

Lazy evaluation

- You can *pipe* as many operations as you like in lazy way: nothing actually happens until you call a **collect()**
- This leaves room for optimizing an appropriate query plan and much more

```
df = pl.read_csv('ghtorrent-2019-02-04.csv')
```

→

```
df = pl.scan_csv('ghtorrent-2019-02-04.csv')
```

Key features of lazy evaluation

Optimizations

According to the actual needs of the process to be collected, the query planner takes care of:

- Predicate pushdown: filter data as early as possible
- Projection pushdown: select columns that are really needed
- Join ordering: to minimize memory usage
- Various tricks to optimize groupby strategy
- And much (much!) more

https://pola-rs.github.io/polars-book/user-guide/optimizations/intro.html

Larger-than-memory dataframes

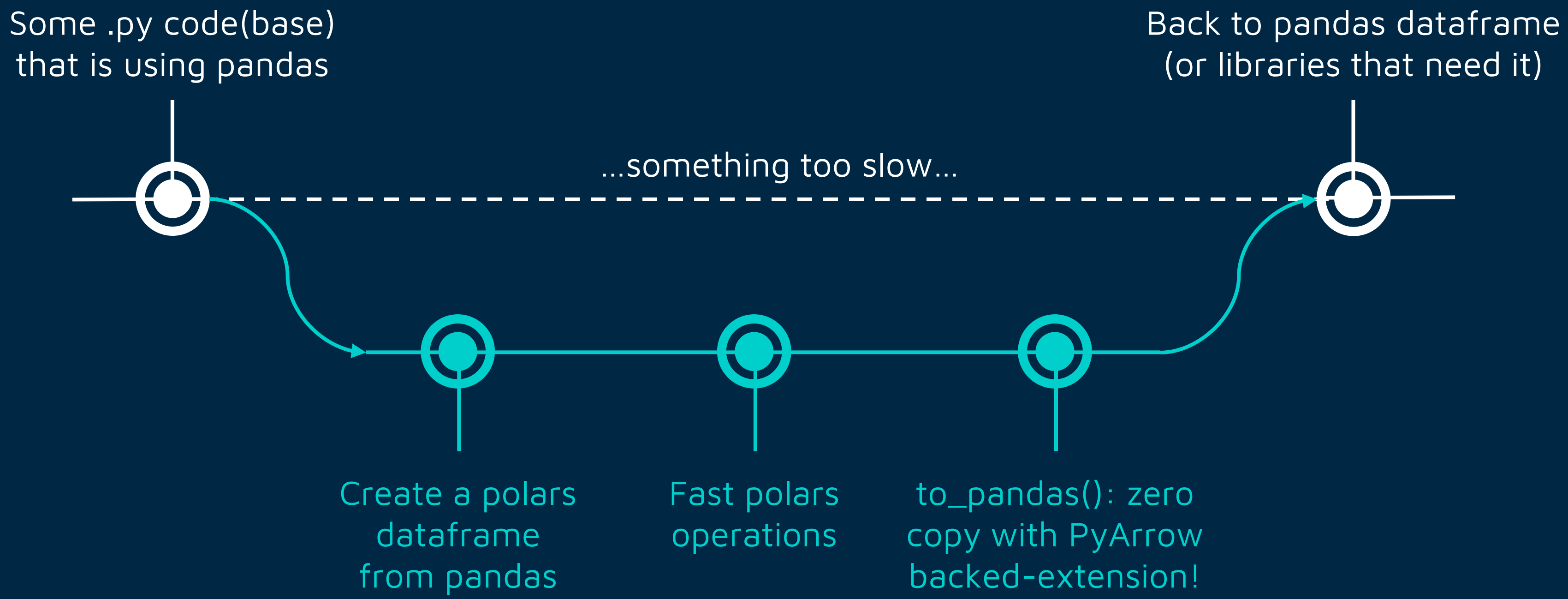
- Remember reading data in chunks to avoid *out of memory* errors? Polars takes care of this under the hood
- How: collect -> collect(streaming=True)

- Not all operations are supported in streaming mode (but most are)
- The final dataset has to fit in memory… unless you sink it directly to a parquet file on disk

https://pola-rs.github.io/polars-book/user-guide/lazy-api/streaming.html

PYCON(IT) 23

Integration in a pandas codebase?



My own benchmarks (1/3)

STEPHAN GARLAND · UPDATED 2 YEARS AGO

7 New Notebook Download (42 GB)

GitHub Pull Requests

A sampling of GitHub pull requests with metadata, sourced from GHTorrent

Data Card Code (1) Discussion (0)

About Dataset

Context

GHTorrent is an amazing project, without which this would be impossible. This was for a research paper on Pull Requests, and I hadn't found anything like it available.

Content

I combined the publicly available tables in BigQuery to get five days of data, each from a different non-holiday Monday from Jan-May of 2019. I'll append the SQL query at the bottom in case anyone would like to get different days worth of data.

Acknowledgements

Georgios Gousios for creating the GHTorrent dataset.

Usability
5.88

License
CC0: Public Domain

Expected update frequency
Not specified

A 20GB csv ☺

~90M rows x 11 columns

https://www.kaggle.com/datasets/stephangarland/ghtorrent-pull-requests

shape: (5, 11)

actor_login	actor_id	comment_id	comment	repo	language	author_login	author_id	pr_id	c_id	commit_date
str	i64	i64	str	str	str	str	i64	i64	i64	datetime[μs]
"calbach"	87112	273585366	"nit: slightly ...	"workbench"	null	"dolbeew"	44115666	59573116	1358455907	2019-02-04 21:10:24
"njohner"	9058655	194786075	"done"	"opengever.core...	"Python"	"deiferni"	971629	40421364	1038953673	2019-02-04 13:09:03
"calbach"	87112	279063421	"rm"	"workbench"	null	"dolbeew"	44115666	60894156	1375725662	2019-02-04 21:10:24
"michaelvidal24...	34641709	270884365	"Should we be c...	"Fabric.Identit...	"C#"	"hckenmiller"	44348190	59127871	1348009853	2019-02-04 20:23:57
"jasiekmiko"	3470241	245735843	"Ah great spot ...	"beis-mspsds"	null	"DWRendell"	7504910	52750526	1258777659	2019-02-04 17:18:14

My own benchmarks (2/3)

A serious benchmark is already DuckDB's one (formerly H2O)… but let's first-hand try something not-so-fancy (group by's, datetime operations, counts, lists of uniques)

Tested on a 2016 desktop PC with 32GB of RAM

```
gby_lazy = (
    df
    .groupby('actor_login')
    .agg(
        [
            pl.count(),
            pl.col('repo').unique().alias('unique_repos'),
            pl.col('repo').n_unique().alias('unique_repos_count'),
            pl.min('commit_date').alias('first_commit'),
            pl.max('commit_date').alias('last_commit'),
            (pl.max('commit_date') - pl.min('commit_date')).alias('delta_time')
        ]
    )
    .sort('count', descending=True)
    .collect()
    .limit(5)
)
gby_lazy
```

My own benchmarks (3/3)

	Polars 0.17.9 Lazy eval	Polars 0.17.9 Eager eval	Pandas 2.0.1 Pyarrow backend	Pandas 2.0.1 Numpy backend
Full dataset read	0s*	∞	∞	∞
Full dataset query	34.9s	∞	∞	∞
First 10M rows read	0s*	3.2s	9.5s**	29.1s**
First 10M rows query	6.1s	1.6s	26.5s	28.3s

* By definition of lazy, not a proper *read*
** Not including casting time for dates

If it's not enough... approx

	Polars 0.17.9 Eager eval approx_unique()	Polars 0.17.9 Eager eval n_unique()	Pandas 2.0.1 Pyarrow backend nunique()	
Result	**92.038**	91.599	91.599	*This is the number of distinct logins (the real one is indeed 91.599)*
Execution time	**0.1s**	0.7s	0.8s	

```
df.select(pl.n_unique('actor_login'))
df.select(pl.approx_unique('actor_login'))
df['actor_login'].nunique()
```

Approximate (i.e. wrong) result, but may be good enough in some cases and takes a fraction of time

Drill-down tomorrow with Luca Baggi @ 14.30

"I came for the **speed**, but I stayed for the **syntax**"

Many Polars users*

* But this precise sentence is taken from this nice article: https://benfeifke.com/posts/the-3-reasons-why-i-switched-from-pandas-to-polars-20230328/

Sneak peek on syntax

```
gby_lazy = (
   df
   .groupby('actor_login')
   .agg(
      [
         pl.count(),
         pl.col('repo').unique().alias('unique_repos'),
         pl.col('repo').n_unique().alias('unique_repos_count'),
         pl.min('commit_date').alias('first_commit'),
         pl.max('commit_date').alias('last_commit'),
         (pl.max('commit_date') - pl.min('commit_date'))      ]
   )
   .sort('count', descending=True)
   .collect()
   .limit(5)
)
gby_lazy
```

My point of view on Polars' syntax:

- **Pythonic and easy to read** even for newbies
- Very **expressive**
- Typically **not as concise** as Pandas

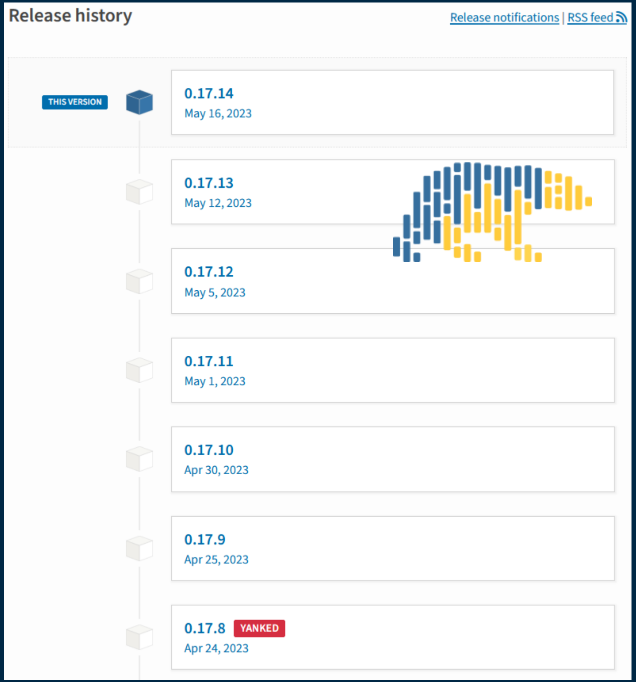
So what is missing?

- There's a **strong reason** why everybody is talking about Polars (and you'll enjoy syntax as much as performance)

- Yet there are many things that are missing (so far)

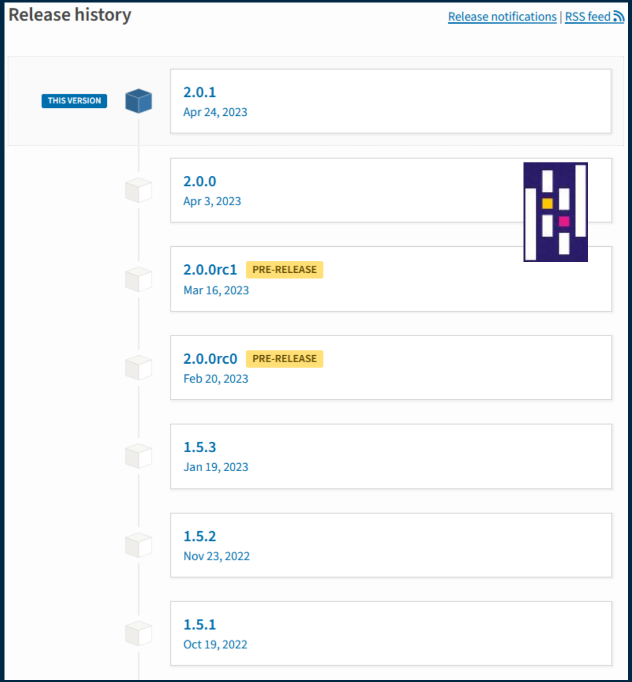
1. **Stability**: close to daily releases, frequent breaking changes
2. **Ecosystem**: first projects based on top of polars starts to show (e.g. ultibi), but most libraries (e.g. ML ones) do require a pandas dataframe – pandas native support for pyarrow (and consequently zero-copy from polars to pandas) may be a game-changer!
3. **Community**: documentation, user guide, tutorials are all getting old very quickly

What I mean with frequent releases


Release history
Release notifications | RSS feed

THIS VERSION 0.17.14 May 16, 2023
0.17.13 May 12, 2023
0.17.12 May 5, 2023
0.17.11 May 1, 2023
0.17.10 Apr 30, 2023
0.17.9 Apr 25, 2023
0.17.8 YANKED Apr 24, 2023

https://pypi.org/project/polars/#history


Release history
Release notifications | RSS feed

THIS VERSION 2.0.1 Apr 24, 2023
2.0.0 Apr 3, 2023
2.0.0rc1 PRE-RELEASE Mar 16, 2023
2.0.0rc0 PRE-RELEASE Feb 20, 2023
1.5.3 Jan 19, 2023
1.5.2 Nov 23, 2022
1.5.1 Oct 19, 2022

https://pypi.org/project/pandas/#history

My take on Polars vs. Pandas vs. rest

- For those who do not like Pandas syntax and/or speed, or have data that is big but not huge, there's a **valid alternative**!

- Built on top of SOTA technologies, with eager/lazy support, a growing community, intuitive syntax and frequent releases, **Polars is here to stay** – the other competitors of pandas have lost momentum
- And if you thought Pandas 2.0 with support for pyarrow could dramatically change the landscape… think again!

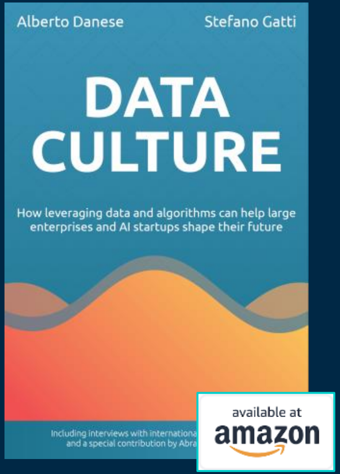
- **Adoption is key**: check out the (free and beautiful) course over at Calmcode.com* and **give Polars a try**!

* https://calmcode.io/polars/calm.html

Alberto Danese

Head of Data Science

www.linkedin.com/in/albertodanese



eBook and paperback

THANKS!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik

https://slidesgo.com/theme/data-science-consulting