Pandas on DuckDB With Ponder

Aditya Parameswaran

Cofounder and President, Ponder

Associate Professor, UC Berkeley

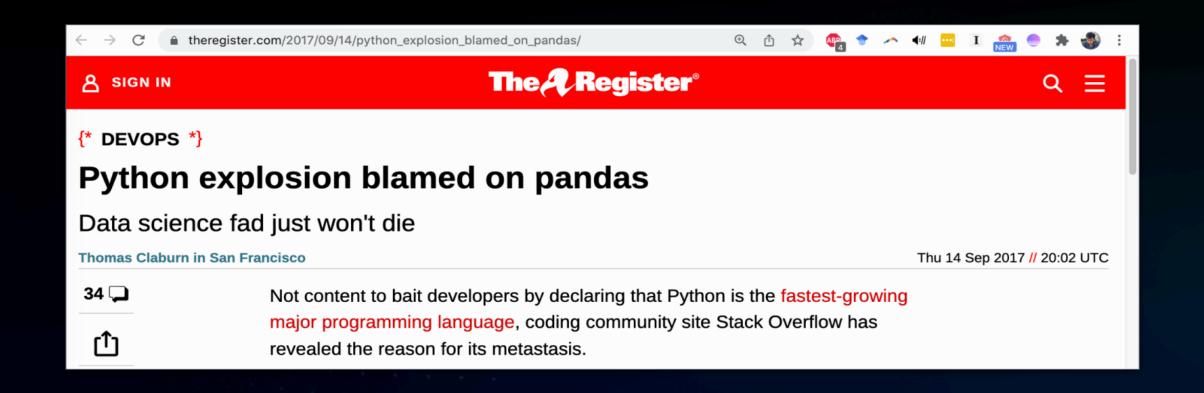








pandas The Swiss Army Knife of Data Science/ML





pandas The Swiss Army Knife of Data Science/ML



Pandas Is Now As Popular As Python Was in 2016

Peter Olson
Jul 29, 2022 • 10 min read

ARTICLES



pandas The Swiss Army Knife of Data Science/ML

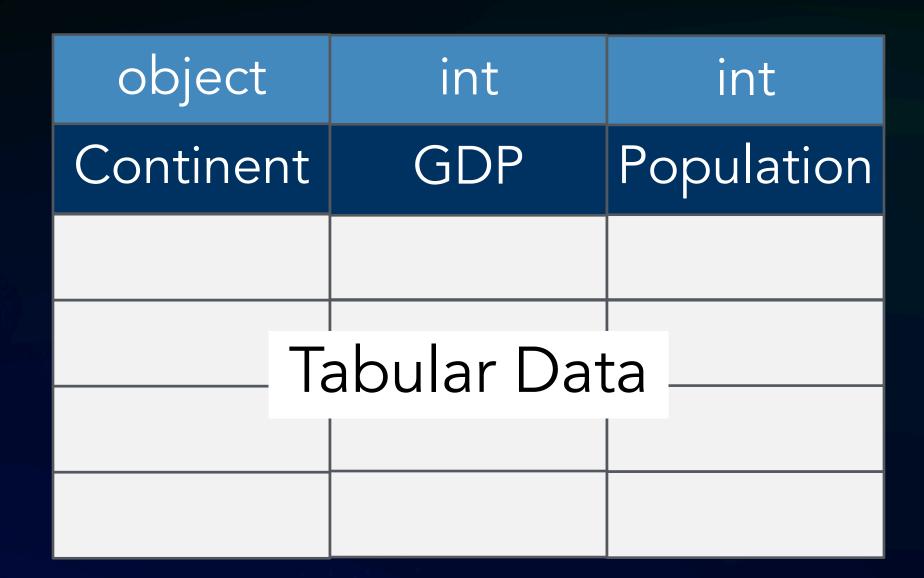


Pandas Is Now As Popular As Python Was in 2016

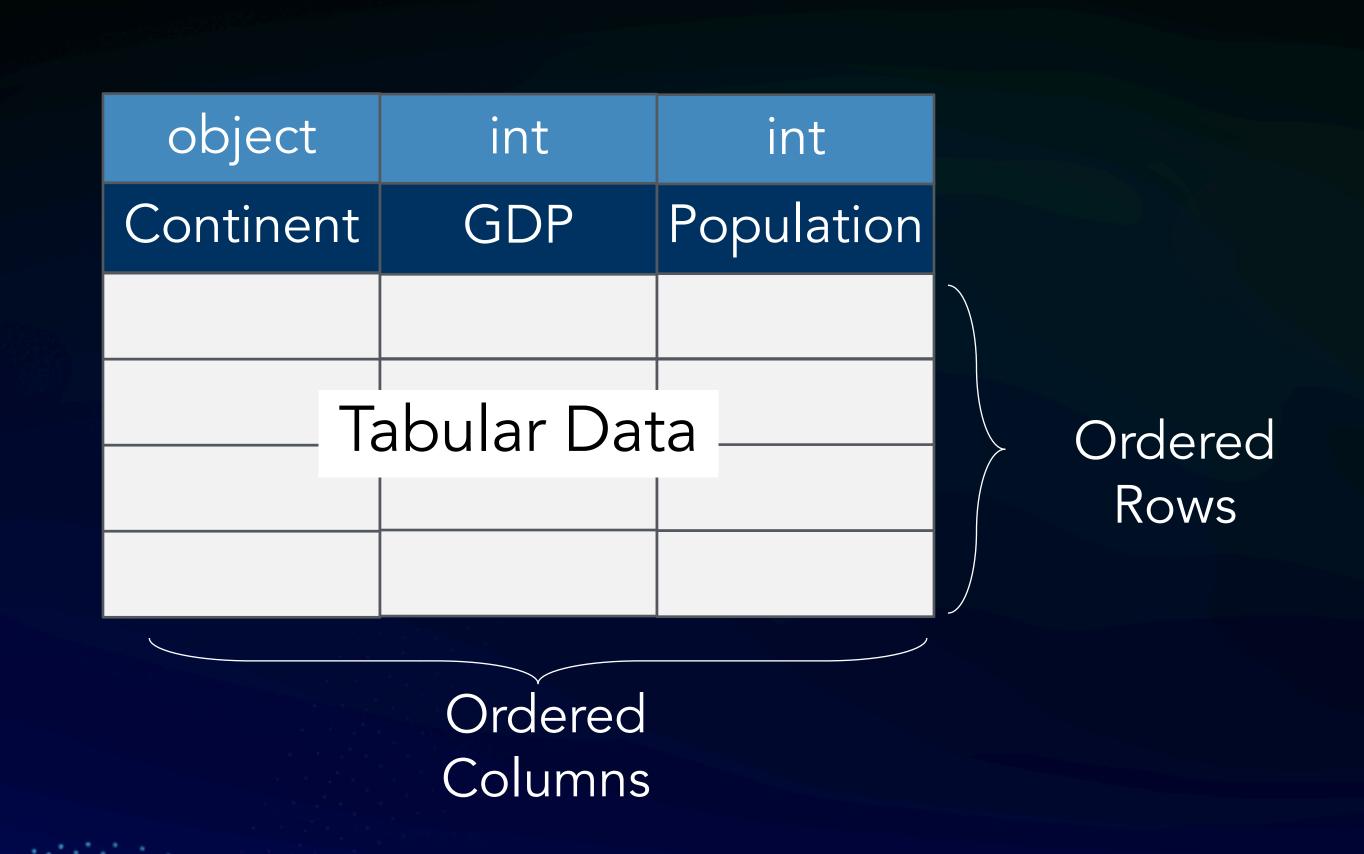
Peter Olson
Jul 29, 2022 • 10 min read



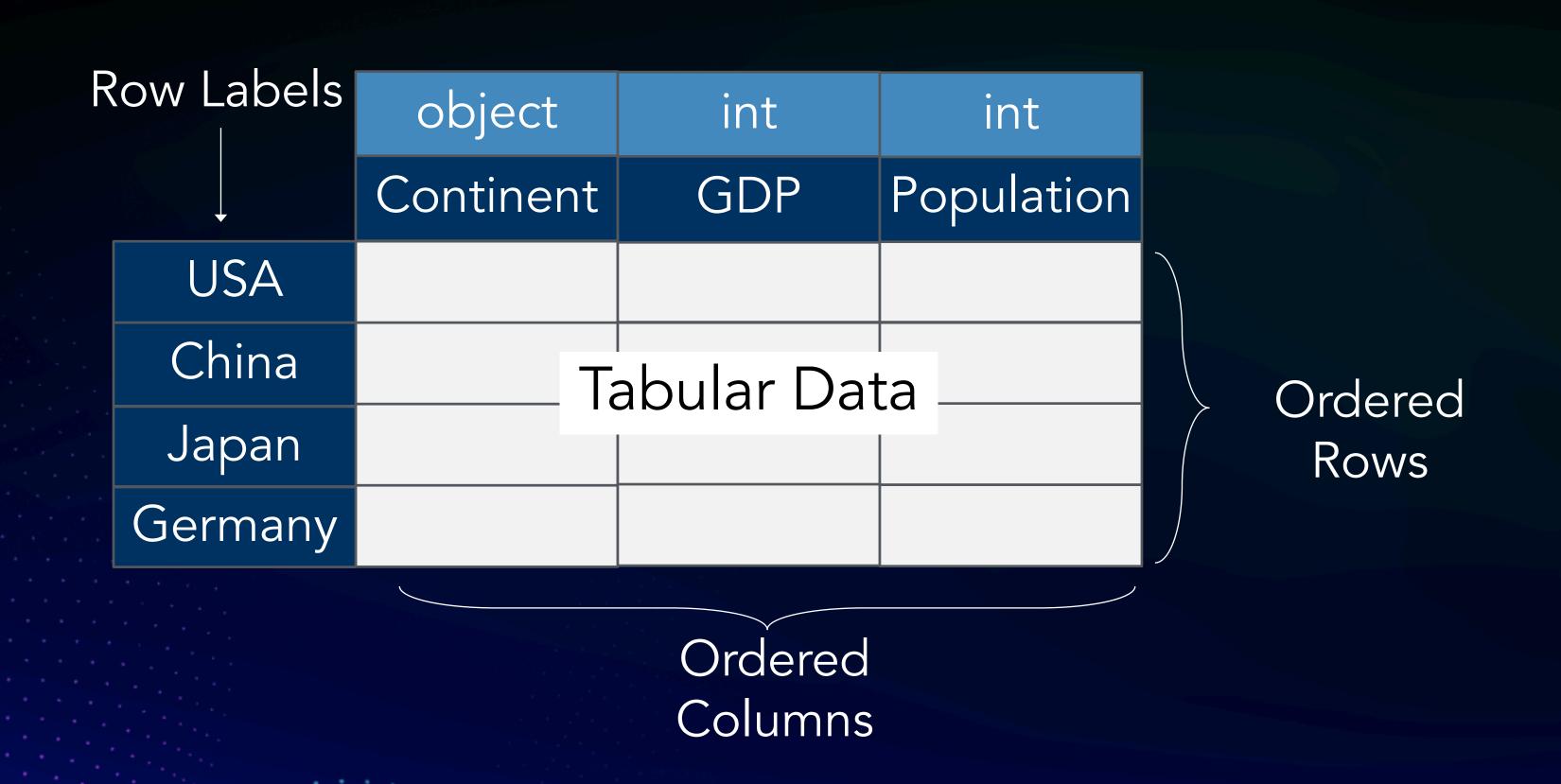




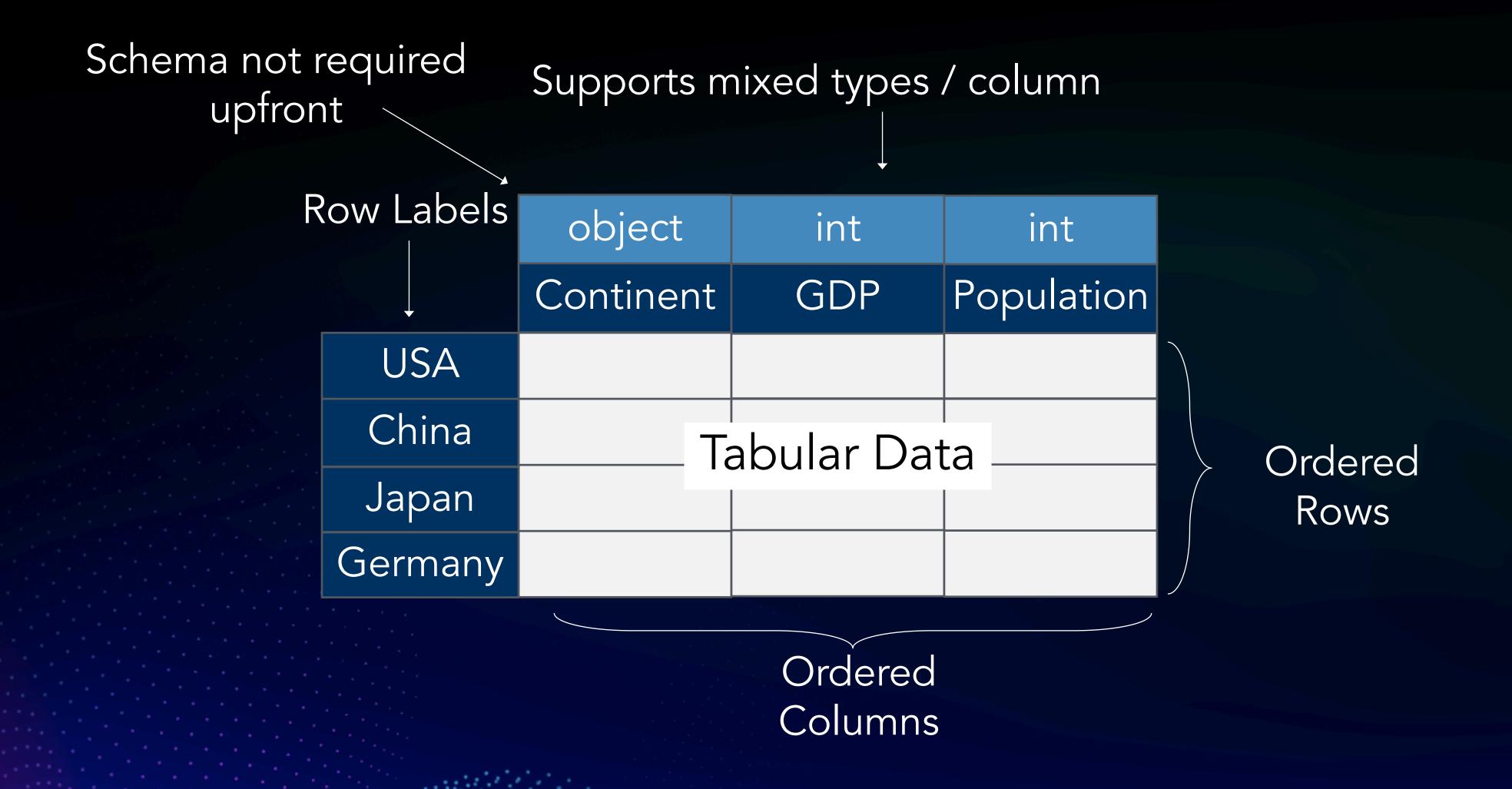












+ Hierarchical row/column labels (not shown)

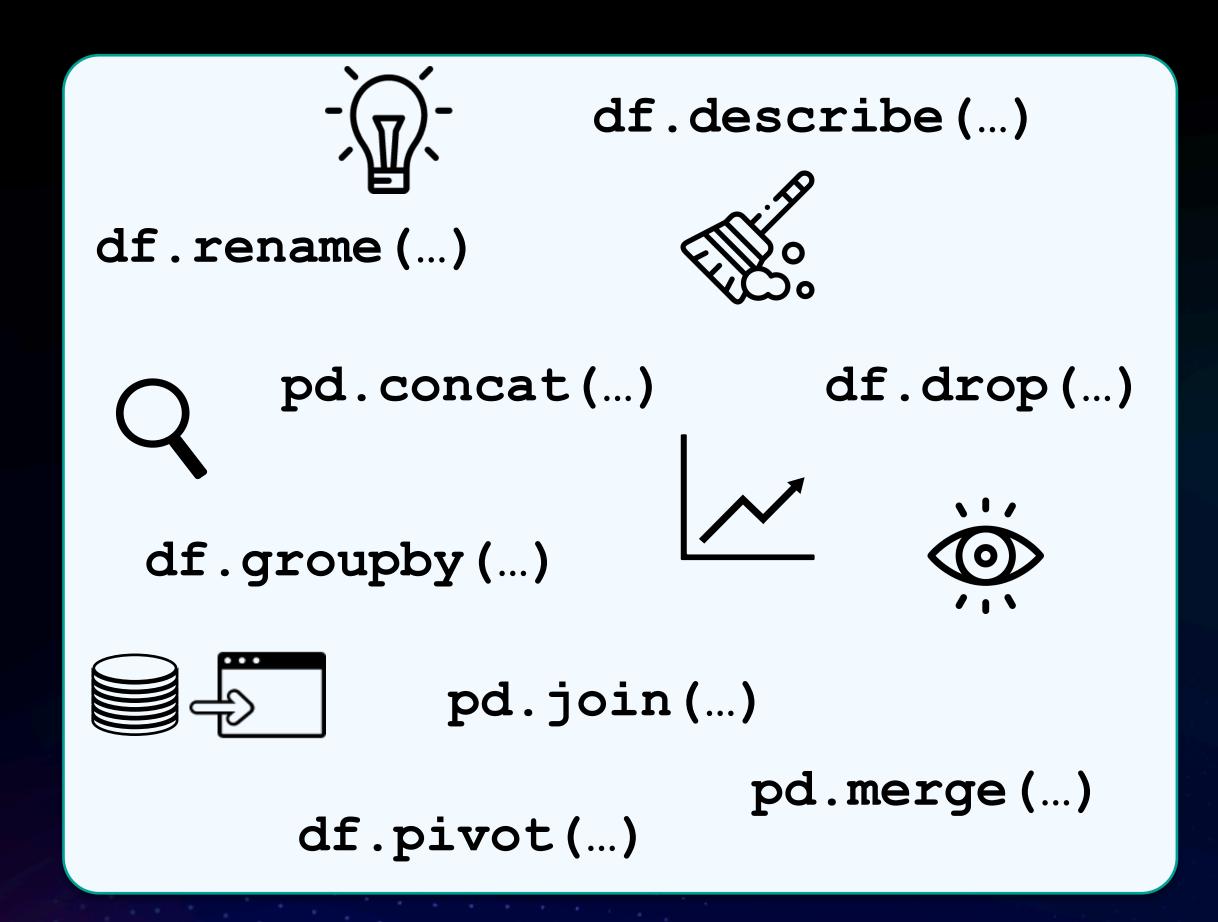


Why? (II) An Expressive & Concise Data Science API

```
df.describe(...)
df.rename(...)
                            df.drop(...)
       pd.concat(...)
 df.groupby(...)
             pd.join(...)
                         pd.merge(...)
       df.pivot(...)
```

600+ functions to clean, featurize, explore, and summarize data spanning rel., linear, & spreadsheet algebra

Why? (II) An Expressive & Concise Data Science API



600+ functions to clean, featurize, explore, and summarize data spanning rel., linear, & spreadsheet algebra

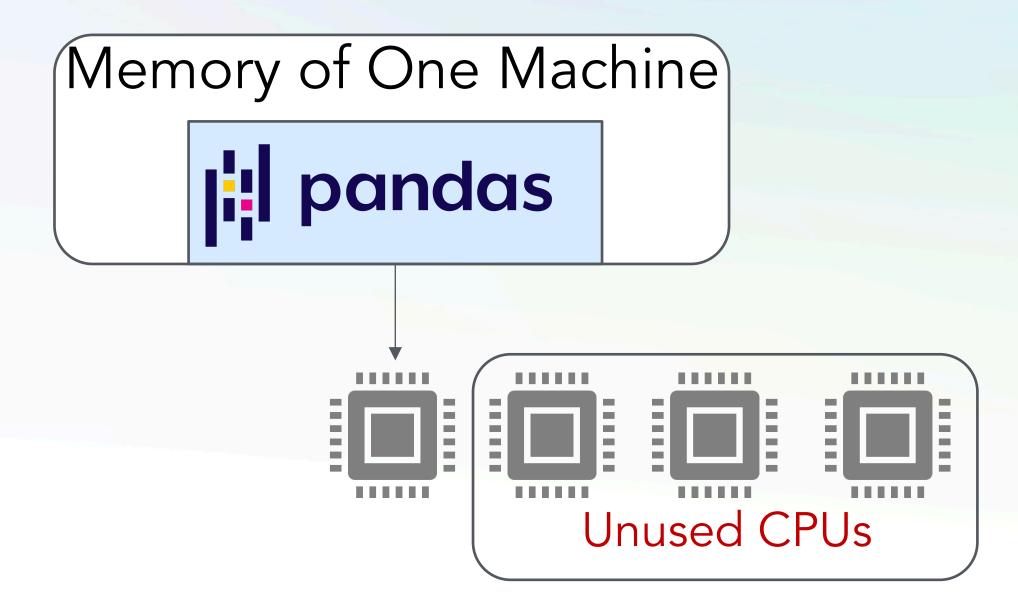
With operations such as:

- Drop columns with null values
- Transpose
- One hot encoding
- Positional updates
- Elementwise matrix ops

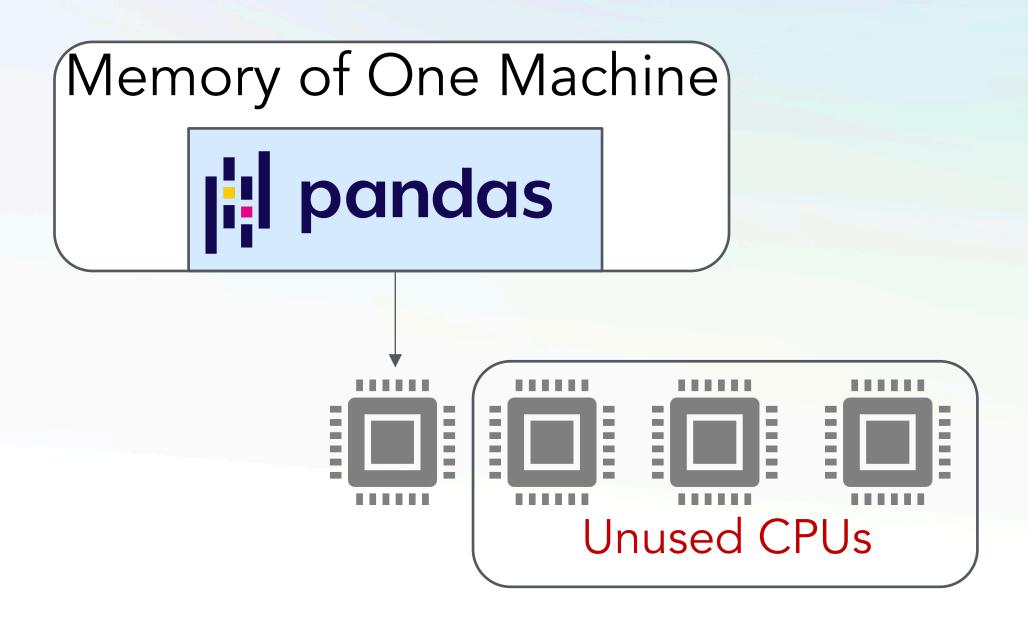
•

Single threaded

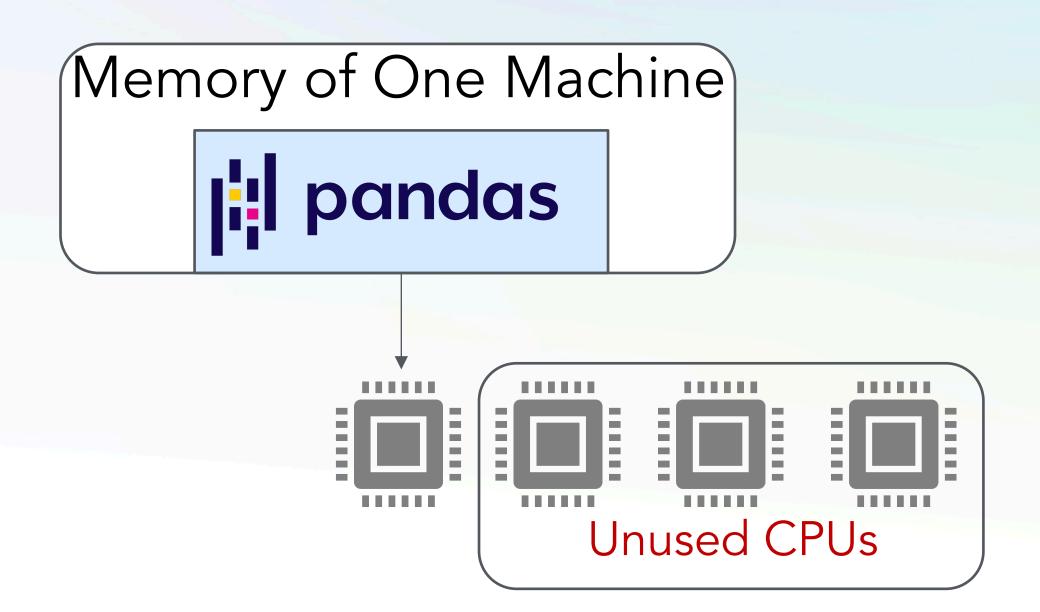
- Single threaded
- Dataset size restricted to memory of a machine



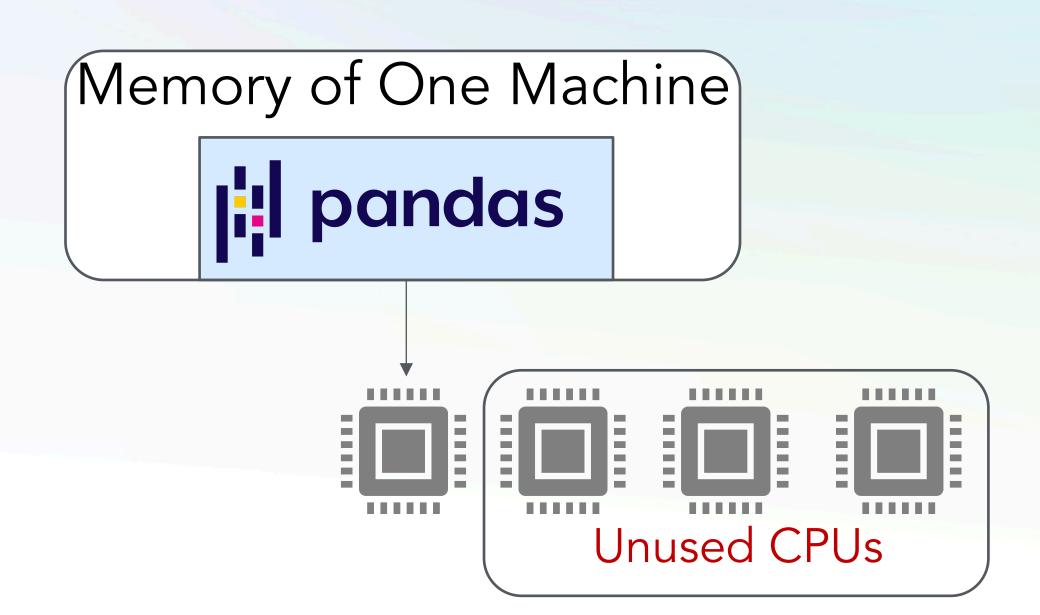
- Single threaded
- Dataset size restricted to memory of a machine
- Inefficient with memory

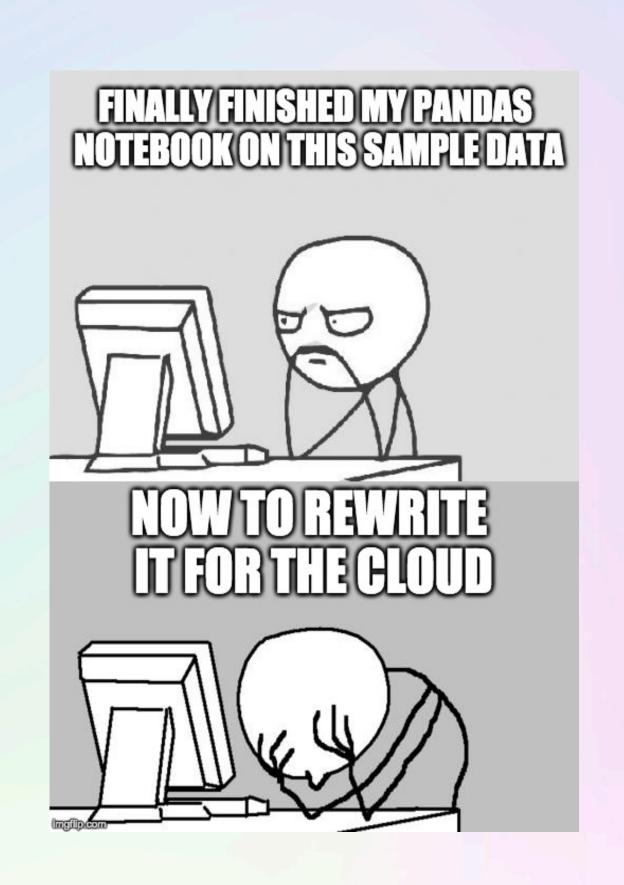


- Single threaded
- Dataset size restricted to memory of a machine
- Inefficient with memory
- No real optimization



- Single threaded
- Dataset size restricted to memory of a machine
- Inefficient with memory
- No real optimization



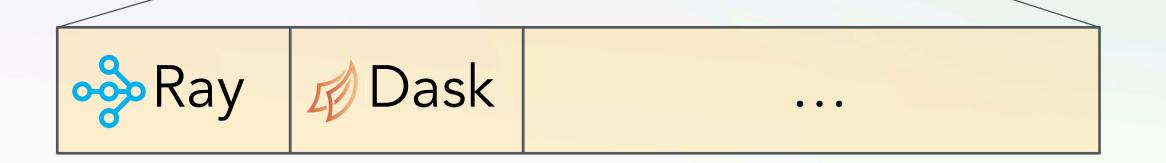


import pandas as pd import modin.pandas as pd

import pandas as pd
import modin.pandas as pd

pandas API





https://github.com/modin-project/modin

import pandas as pd import modin.pandas as pd

pandas API







• • •

https://github.com/modin-project/modin

- 4 10+ Million Downloads (1M+/month)
- > 100+ contributors
- Used by 40+ companies & orgs
- Part of AWS, Intel distros

import pandas as pd import modin.pandas as pd

pandas API







• • •

https://github.com/modin-project/modin



10+ Million Downloads (1M+/month)



> 100+ contributors



Used by 40+ companies & orgs



Part of AWS, Intel distros

Towards Scalable Dataframe Systems

Devin Petersohn, William Ma, Doris Lee, Stephen Macke, Doris Xin, Xiangxi Mo Joseph E. Gonzalez, Anthony D. Joseph, Joseph M. Hellerstein, Aditya Parameswaran

ABSTRACT

structure, clean, and analyze data during exploratory data analysis Despite the success of dataframe libraries in R and Python (pandatasets. In this vision paper, we take the first steps towards formally defining dataframes, characterizing their properties, and outlining a research agenda towards making dataframes more interactive at scale. We draw on tools and techniques from the database community, and describe ways they may be adapted to serve dataframe systems, as well as the new challenges therein. We also describe our current progress toward a scalable dataframe system, MODIN, which is already up to 30× faster than pandas in preliminary case studies, while enabling unmodified pandas code to run as-is. In its first 18 months, MODIN is already used by over 60 downstrear projects, has over 250 forks, and 3,900 stars on GitHub, indicating the pressing need for pursuing this agenda.

1 INTRODUCTION

Characteristics such as these have helped dataframes becon credibly popular for EDA; for instance, the dataframe abstra provided by pandas within Python (pandas.pydata.org), has of 2019, been downloaded over 200 million times, served dency for over 160,000 repositories in GitHub, and star on GitHub more 22,000 times. Python's own popularity has b attributed to the success of pandas for data exploration and data :

ence [7,9]. Due to its ubiquity, we focus on pandas for concrete Pandas has been developed from the ground up via open-sour contributions; pandas DataFrame API operators and their imple mentations have been provided by dozens of contributors to satisfy immediate or ad-hoc needs, spanning capabilities that mimic rela tional algebra, linear algebra, and spreadsheet formula computation. To date, the pandas DataFrame API has ballooned to over 200 distinct operators [14]. R, which is both more mature and more carefully curated than pandas, has only 70 operators-but this is

still far more than, say, relational and linear algebra combined [15]. While this rich API is sometimes cited as a reason for panda attractiveness, the set of operators has significant redundancies, or



Flexible Rule-Based Decomposition and Metadata Independence in Modin: A Parallel Dataframe System

Devin Petersohn*, Dixin Tang*, Rehan Durrani, Areg Melik-Adamyan[†], Joseph E. Gonzalez, Anthony D. Joseph, Aditya G. Parameswaran

UC Berkeley | † Intel $\{devin.petersohn, to temtang, rdurrani, jegonzal, adj, adityagp\} @berkeley.edu, areg.melik-adamyan @intel.com, and all adityagp and adityagp and all adityagp and adityagp and all adityagp and all adityagp and adityagp adityagp and adityagp and adityagp and adityagp and adityagp and$

in various stages of structure, and manipulate it using a rich of operators—thereby becoming an essential tool in the data ntists' toolbox. However, dataframe systems, such as pandas, cale poorly—and are non-interactive on moderate to large datasets We discuss our experiences developing MODIN, our first cut at a parallel dataframe system, which already has users across several industries and over 1M downloads. MODIN translates pandas func tions into a core set of operators that are individually parallelized via columnar, row-wise, or cell-wise decomposition rules that we formalize in this paper. We also introduce metadata independence to allow metadata—such as order and type—to be decoupled from the physical representation and maintained lazily. Using rule-based omposition and metadata independence, along with careful engineering, MODIN is able to support pandas operations across both

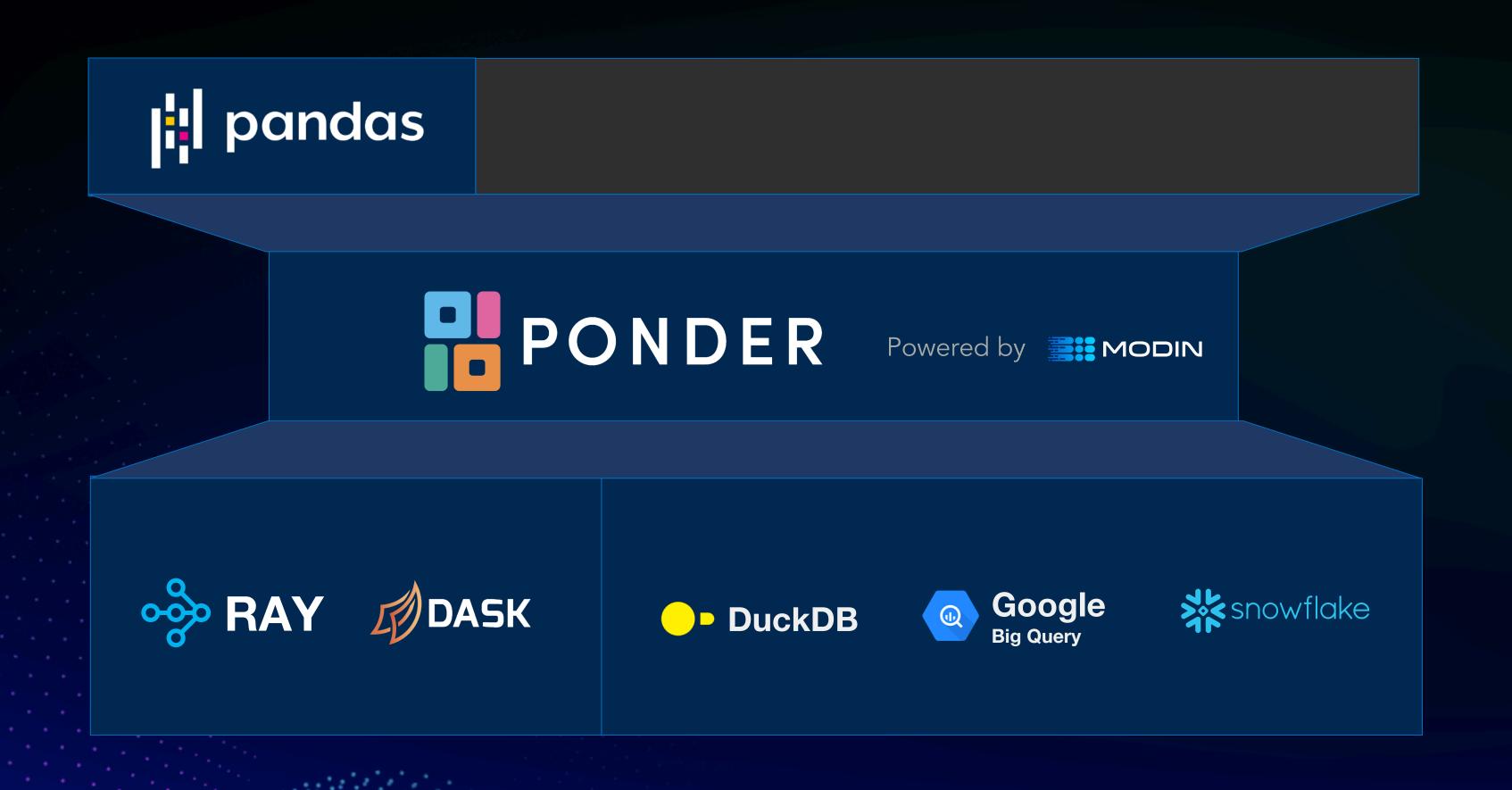
quickly get started on analysis without having to declare a schema of resolve type issues, and can employ non-relational operations useful in data analysis (such as transpose). From the operator standpoin dataframe systems provide a rich and varied set tailored to data science, allowing users to operate equivalently across both rows and columns; pandas supports over 600 such functions. For example fillna allows data scientists to clean data by filling in NULL value without having to write custom code.

At the same time, it is well-known that dataframe systems lik pandas are non-interactive on moderate-to-large datasets, and break down completely when operating on datasets beyond main mem ory [2, 6, 32-34, 42, 45]. These issues represent significant cha lenges for users who are unwilling or unable to switch to other, more scalable tools, such as relational databases. To address these shortcomings, we have been developing MODIN (https://github.com modin-project/modin), a parallel dataframe system, acting as a drop nent for pandas. MODIN is already being used

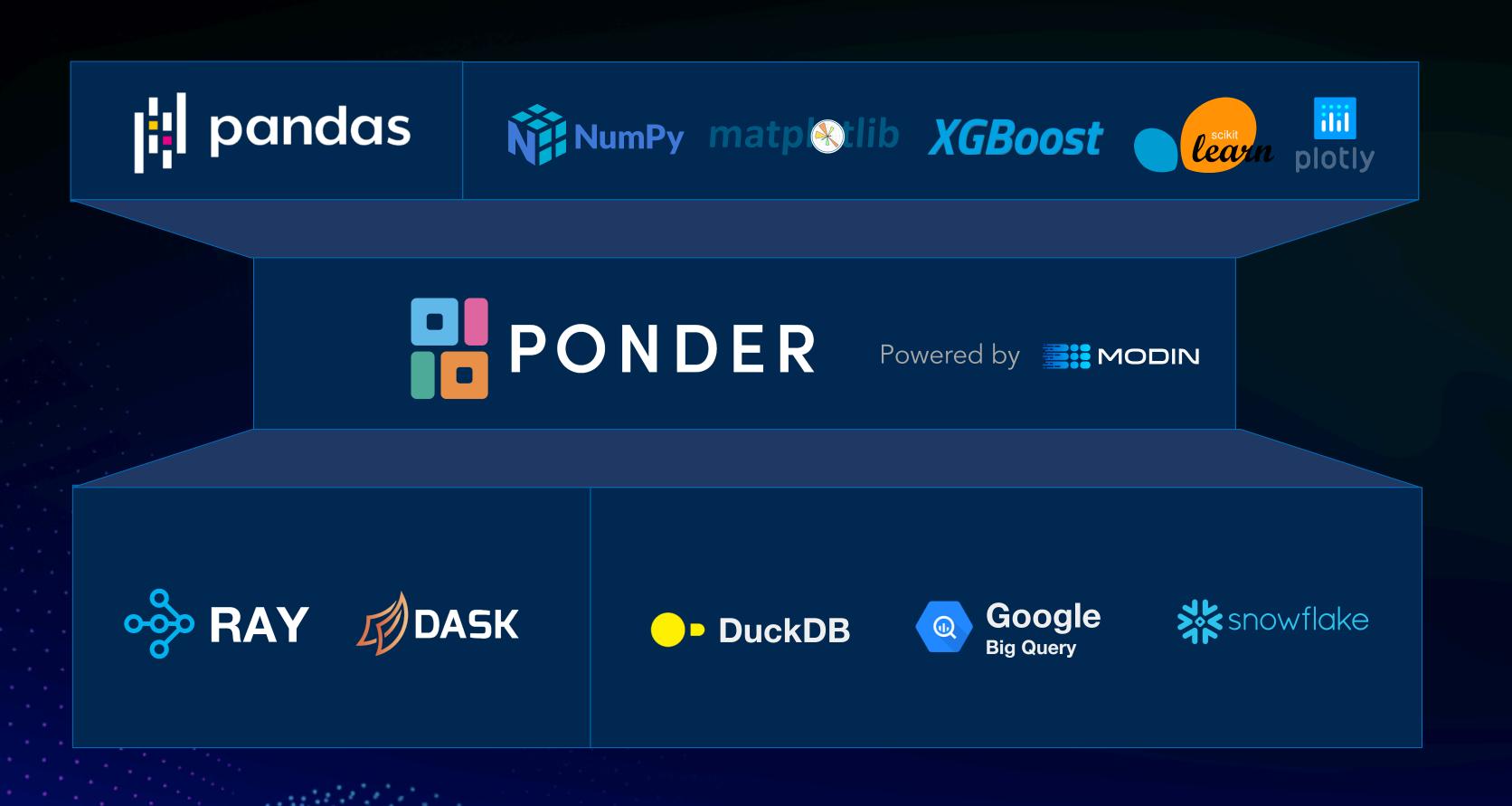
"Core" Dataframe Algebra

Parallelization Framework

The Next Generation with Ponder: Data Science at scale now powered by your database



The Next Generation with Ponder: Data Science at scale now powered by your database



The Magic of Ponder



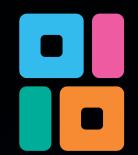
Min-Max normalization

$$x_{scaled} = rac{x - \min(x)}{\max(x) - \min(x)}$$

One-line in pandas

```
df - df.min()) / (df.max() - df.min())
```

The Magic of Ponder



Min-Max normalization

$$x_{scaled} = rac{x - \min(x)}{\max(x) - \min(x)}$$

One-line in pandas

```
(df - df.min()) / (df.max() - df.min())
```

Equivalent SQL is 300+ lines!!

```
1 SELECT
     _PONDER_ROW_LABELS_,
     C_CUSTKEY,
     C_NATIONKEY,
    C_ACCTBAL
       SELECT
       FROM
10
11 -
12
           SELECT
13
              _PONDER_ROW_NUMBER_,
14
              _PONDER_ROW_LABELS_,
15
             C_CUSTKEY / C_CUSTKEY_ponder_right AS C_CUSTKEY,
16
             C_NATIONKEY / C_NATIONKEY_ponder_right AS C_NATIONKEY,
17
             C_ACCTBAL / C_ACCTBAL_ponder_right AS C_ACCTBAL
18
           FROM
19 -
               SELECT
20
21
                 _PONDER_ROW_NUMBER_,
22
                 _PONDER_ROW_LABELS_,
23
                 C_CUSTKEY - C_CUSTKEY_ponder_right AS C_CUSTKEY,
                 C_NATIONKEY - C_NATIONKEY_ponder_right AS C_NATIONKEY,
24
                 C_ACCTBAL - C_ACCTBAL_ponder_right AS C_ACCTBAL
25
               FROM
26
27 -
28
                   SELECT
29
                      _PONDER_ROW_NUMBER_,
30
                      _PONDER_ROW_LABELS_,
31
                      C_CUSTKEY,
32
33
                     C_NATIONKEY,
                     C_ACCTBAL
34
                   FROM
```

Easy to Get Started

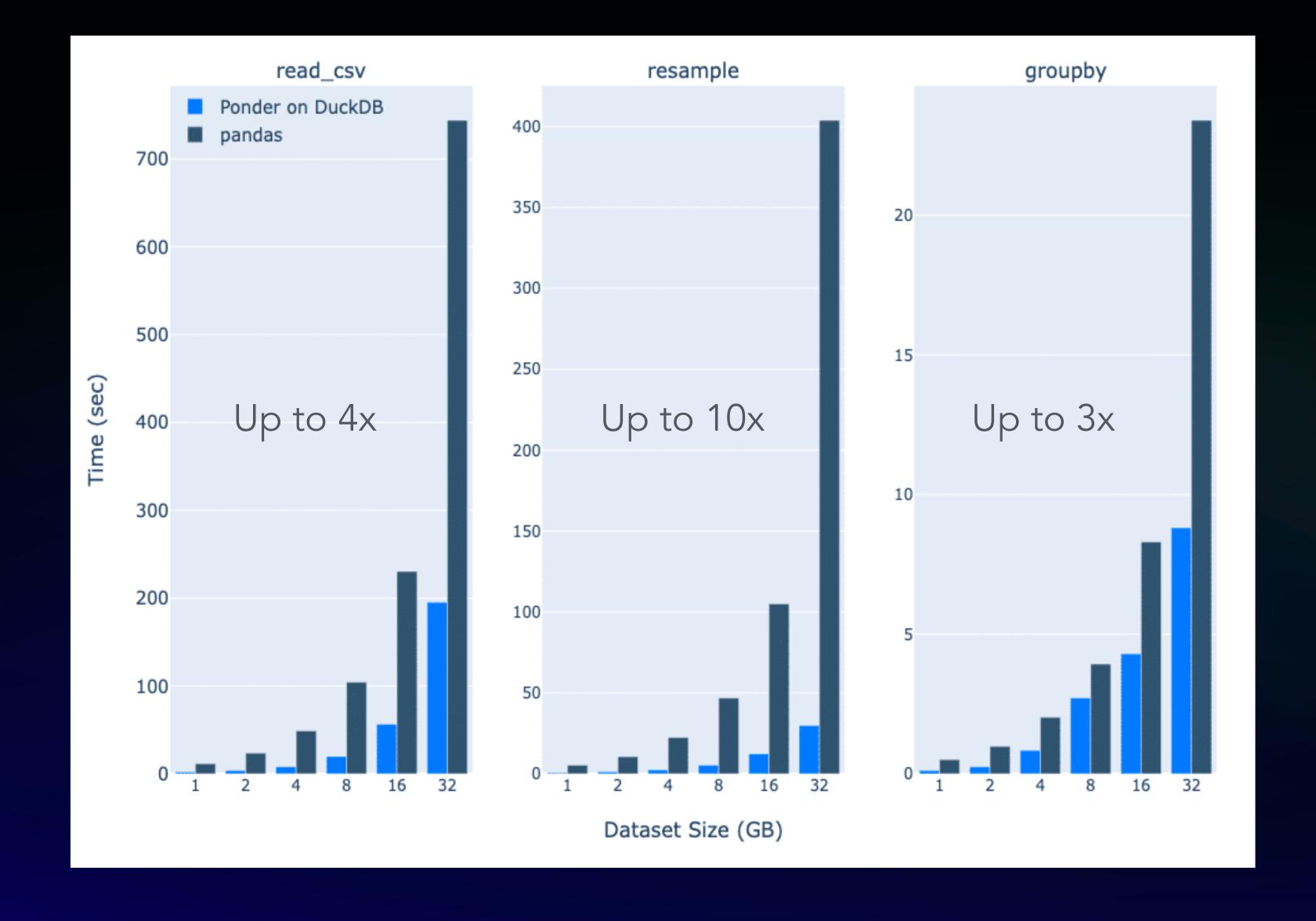
Create a database connection

```
import duckdb
db_con = duckdb.connect()
```

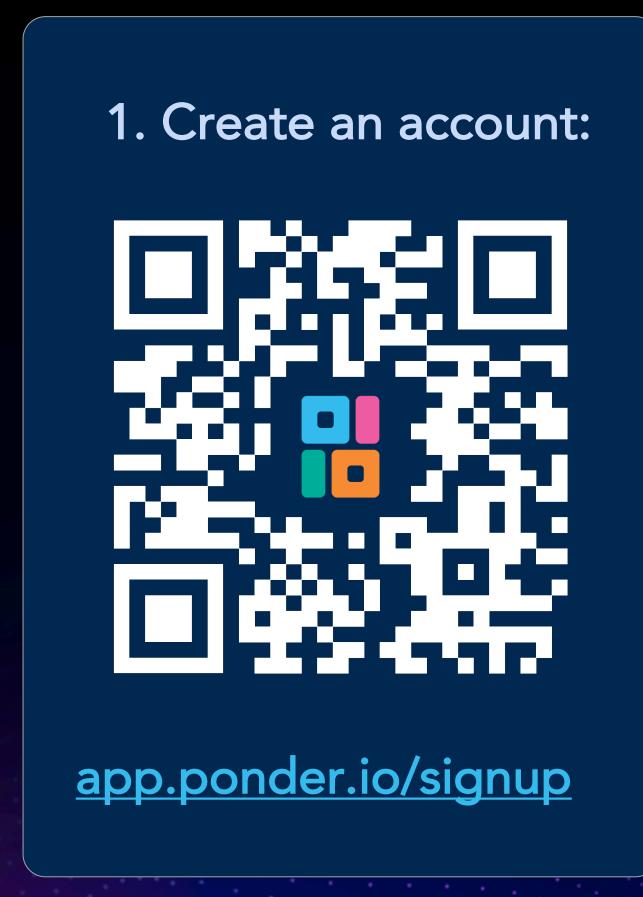


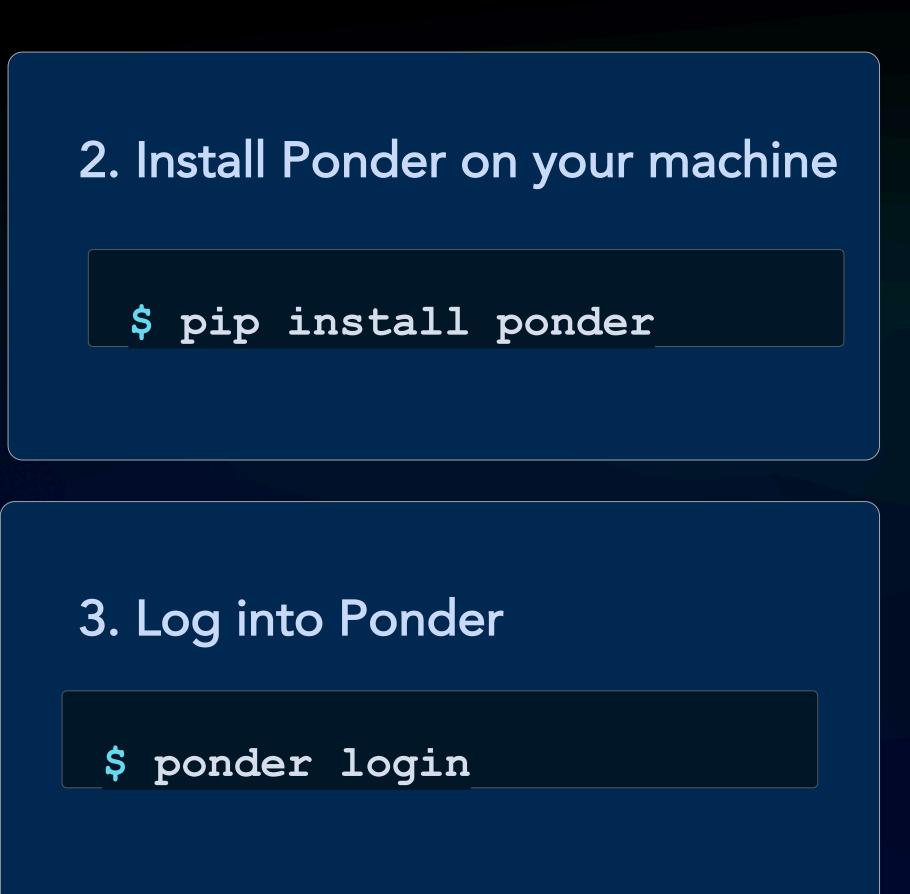
Start running pandas directly in your database

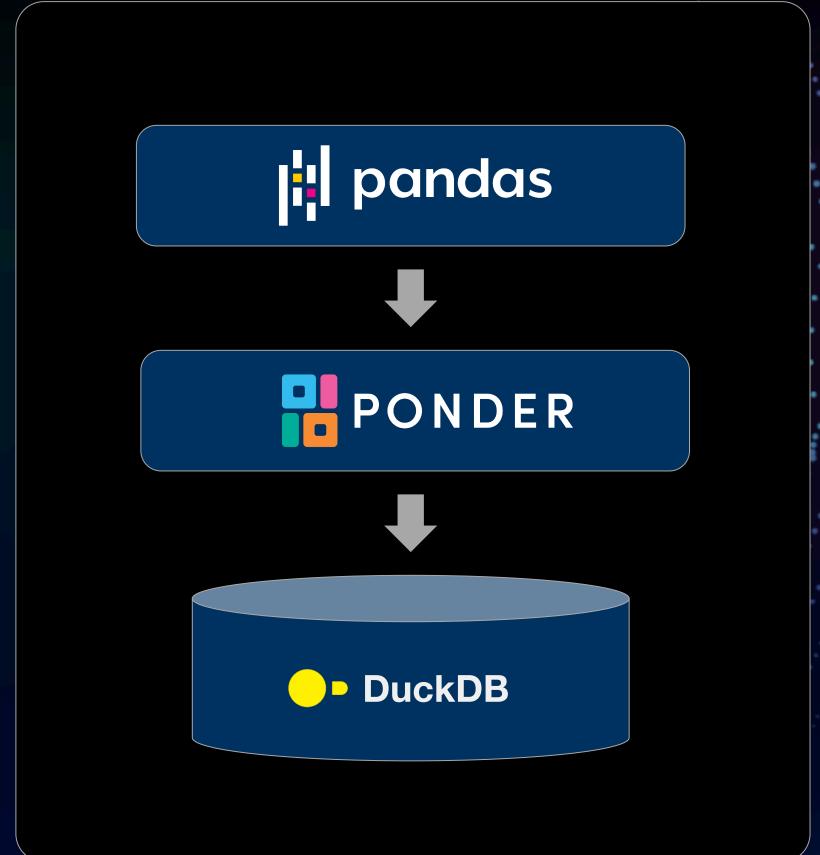
API-level Performance – Ponder on DuckDB vs. Pandas



Two Minutes to Scalable Pandas with DuckDB!

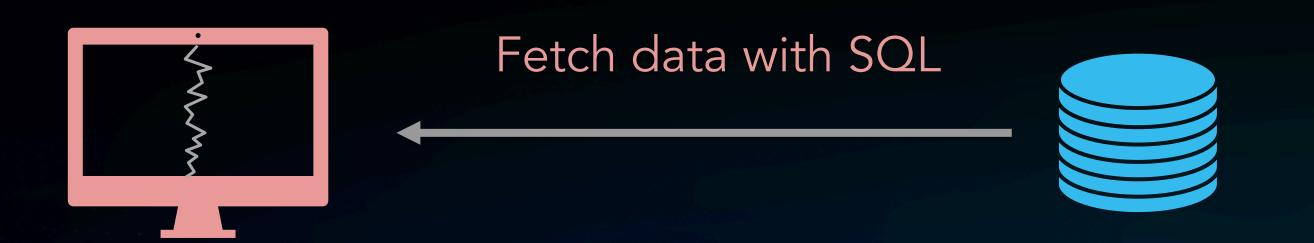






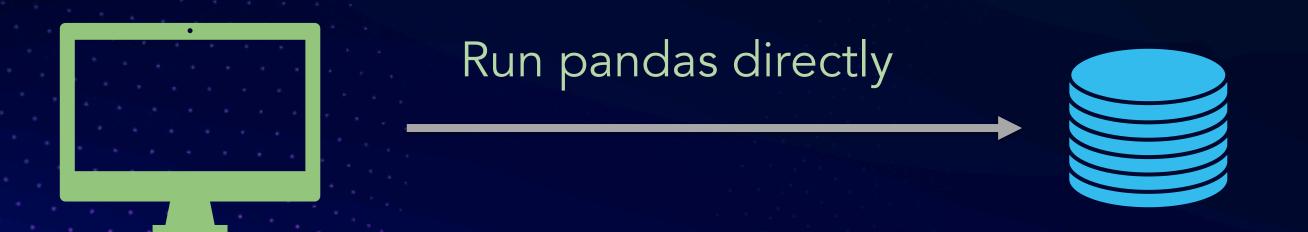
Backup Slides

With pandas



- X Limited samples
- X Out-of-memory errors
- X Expensive I/O costs
- X Un-secure access

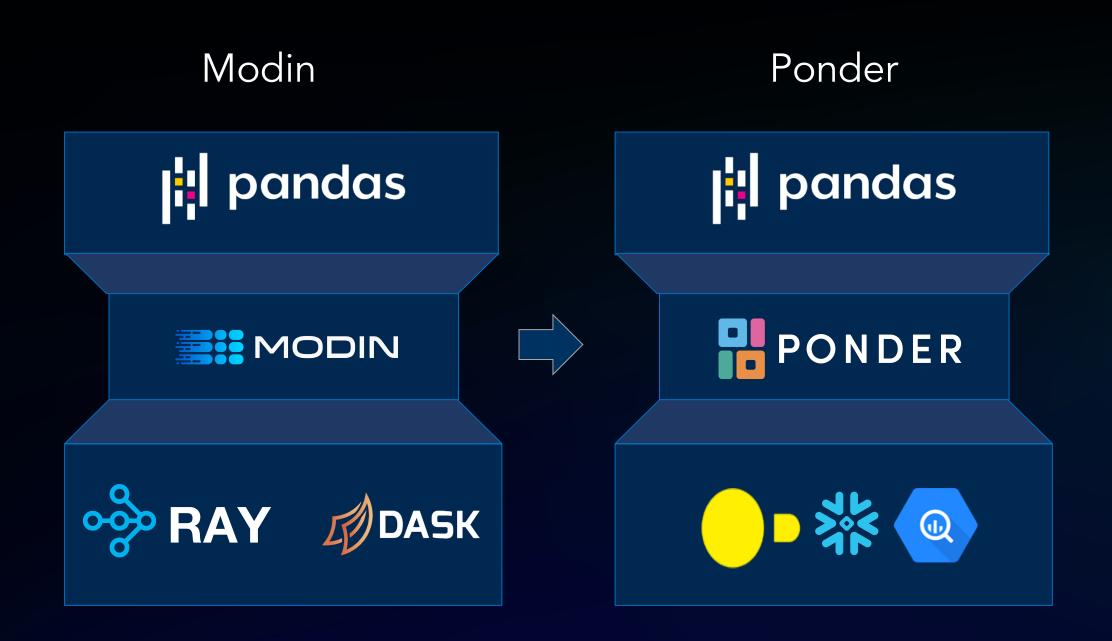




- Ready for production
- Leverage database optimizations
- ✓ No infrastructure setup required
- Securely access data where it lives

Why was this Difficult?





It is difficult to map Pandas to SQL, while supporting

- Order Semantics
- Metadata (mixed types, data/metadata equivalence)
- Incremental query construction
- Coverage of 600+ APIs
 (incl. ops not possible in SQL systems)

Modin instead maps to distrib. compute engines (execute anything they're told).