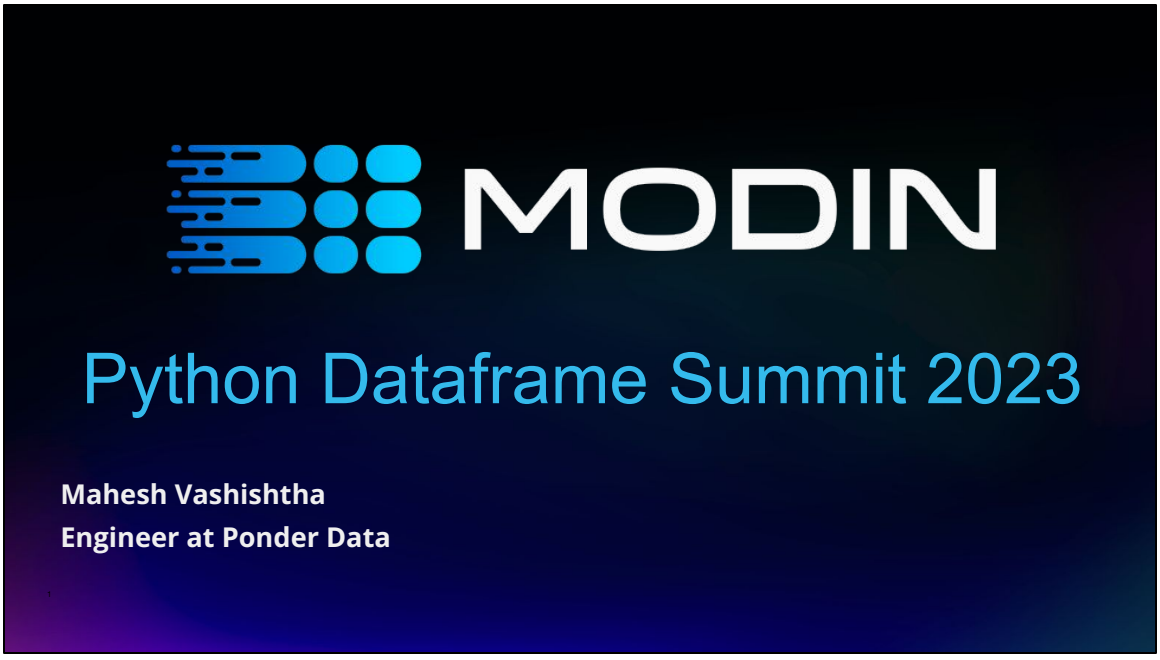
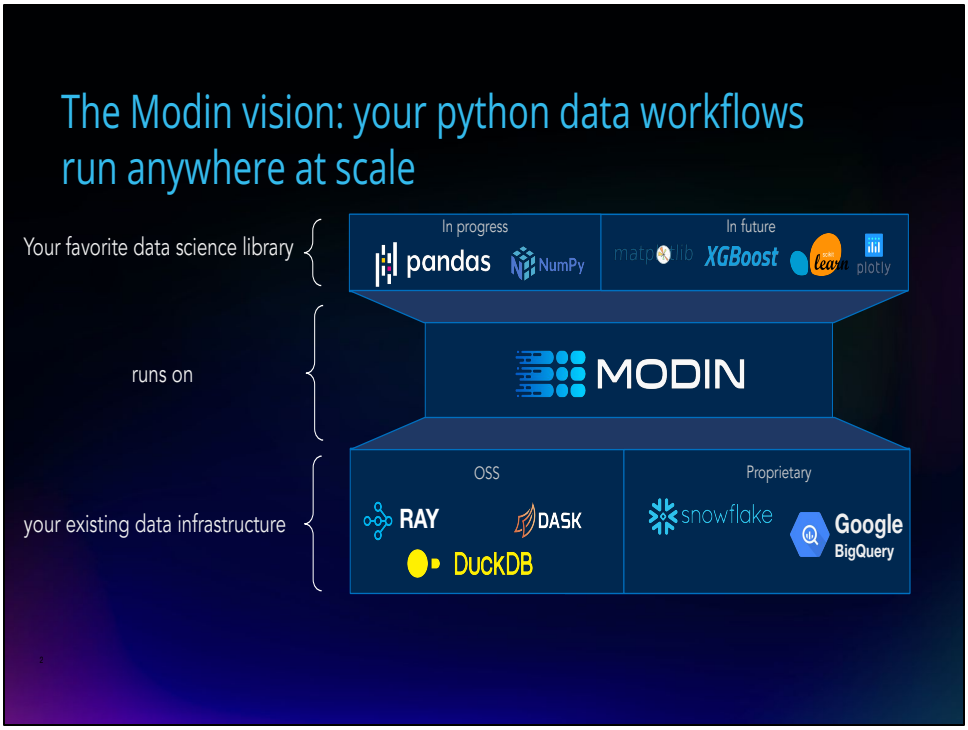

MODIN
Python Dataframe Summit 2023
Mahesh Vashishtha
Engineer at Ponder Data




Our vision expands beyond pandas to *any* data science library running at scale on *any* infrastructure backend that you already have. Our goal is to make data teams more productive without changing the way they work.

That means meeting users where they are using pandas or any other library in the Python data processing ecosystem for ML, visualization, and beyond, leveraging whatever data infrastructure they already have, whether that's a database or warehouse like Snowflake or BigQuery; a distributed computing cluster like Ray or Dask; or anywhere else where their data lives.

## Current API coverage

- Coverage of pandas API for “distributed pandas” execution with ray, dask, and MPI:
  - pandas.DataFrame - 90% (>93% based on usage)
  - pandas.Series - 88% (>86% based on usage)
  - pandas.read_* - >42% (>90% based on usage)
- Slightly less coverage for execution on HDK, a backend based on the relational HeavyDB
- Coverage via ponder, a closed-source Modin extension, for relational databases: ~40%
- Preliminary numpy API coverage in modin.numpy

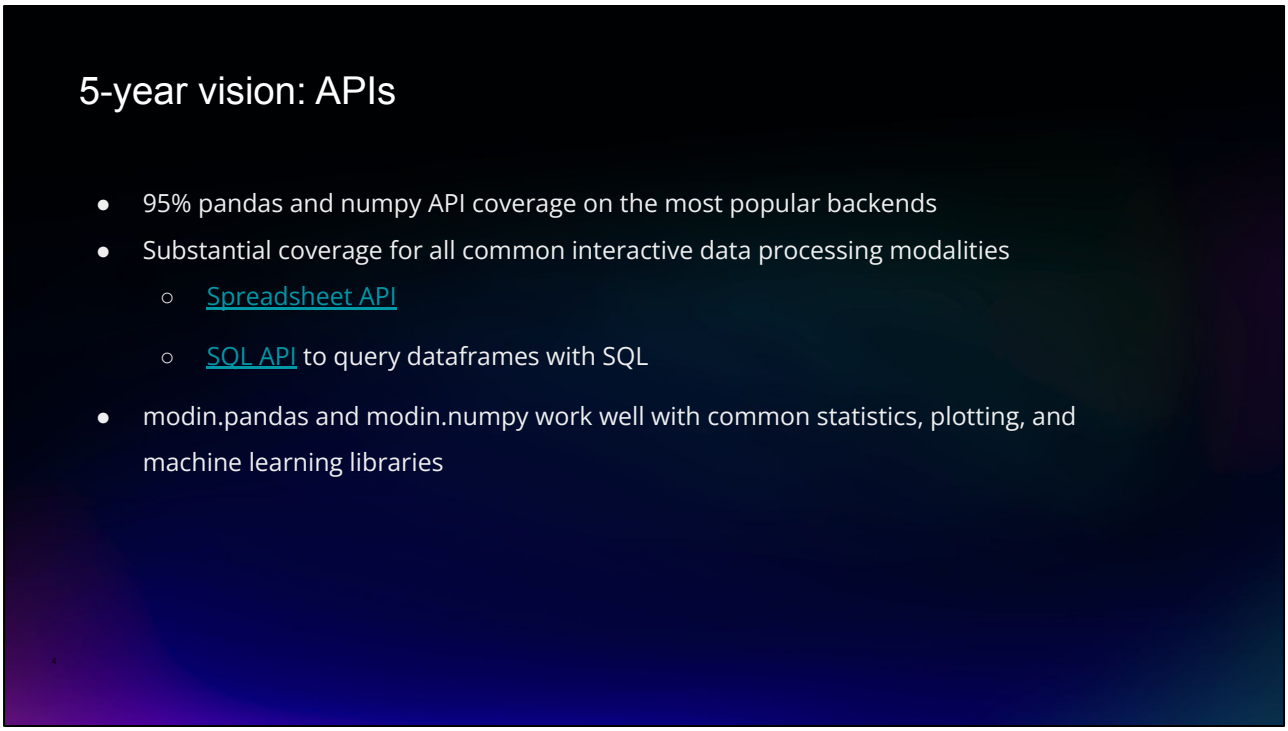
Each backend supports each API to a different extent.

We have spent the most effort on execution with ray, dask’s distributed, and MPI, all using pandas dataframes to store partitions of a dataframe.
These backends support something like 90% of the pandas API if we weigh coverage by usage (because many of the unimplemented functions are rarely used.)

We also have some support for execution on HDK, a high-performance, SQL-based, relational, columnar database engine.

Meanwhile, Ponder Data is developing a proprietary extension to Modin that uses relational databases for execution and storage. Right now ponder covers ~40% of the pandas API on bigquery, snowflake, and duckdb

Some of the backends support a few numpy methods as well.




Here is our 5-year vision for modin APIs.

We would like to cover 95% of the pandas and numpy APIs on the most popular backends. These would probably include the ray, dask, and MPI; HDK; and the relational databases that ponder supports.

We also want to have good coverage for other common data processing modalities, including spreadsheets and the modin SQL API.

We want modin.pandas and modin.numpy to work as well as pandas and numpy with common statistics, plotting, and machine learning libraries

## 5-year vision: API execution

- Performance always at least as good as pandas, and better for many methods
- On distributed pandas engines: minimize data movement between workers
- On HDK: GPU integration and heterogenous execution (e.g. execute on CPUs as well as GPUs)
- Query planning and optimization on distributed pandas engines
  - Requires more lazy execution
  - Possibly: choosing to execute queries asynchronously while the user is thinking
- "serverless" execution can scale up and down as the user needs (e.g. keep everything on one node until data reaches a certain size)

5

Here is our 5-year vision for executing the APIs that we support.

First, our primary concern is performance. We want modin to always perform at least as well as pandas, and it should do better for many methods.

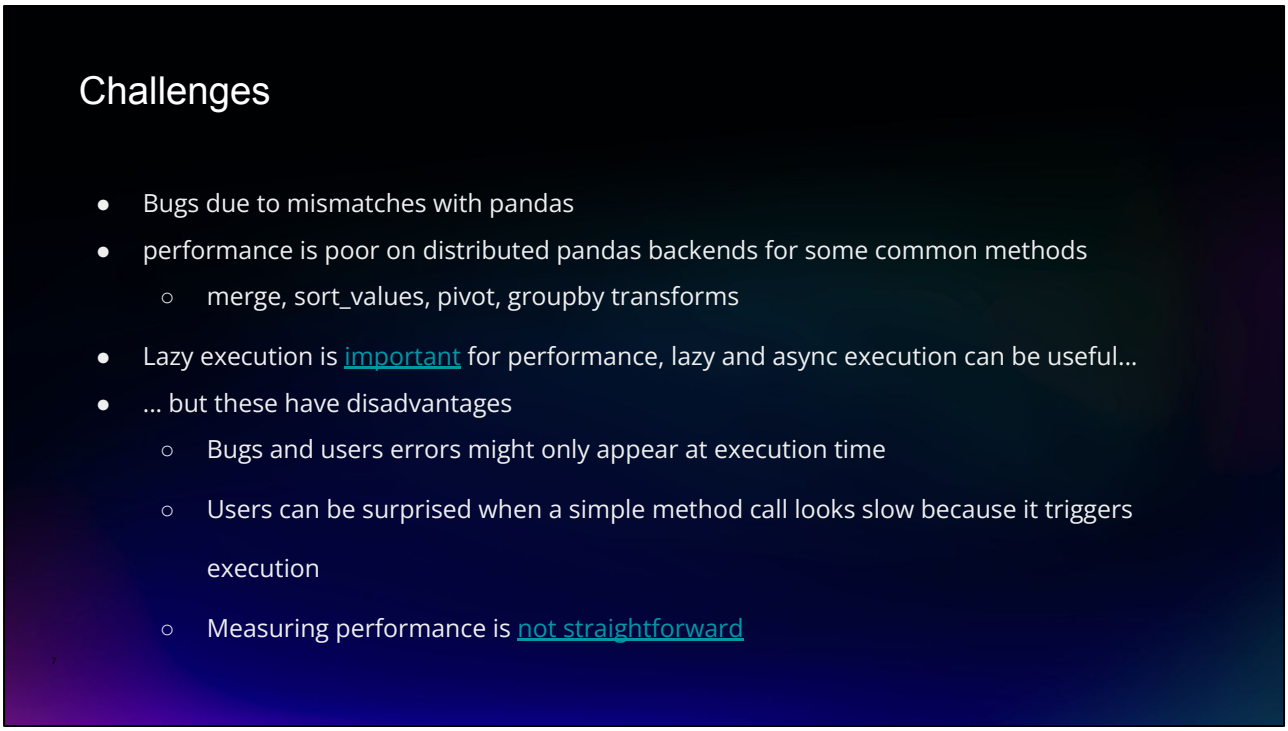
On distributed pandas backends, we need to continue minimizing data movement between workers. Data movement between workers is usually the main cause of slowness.

On HDK, we want GPU integration and heterogenous execution, meaning we can execute a query on both CPUs and GPUs.

To improve performance, we would like to do some amount of query planning and optimization. We already do this for lazy execution with HDK,
but we don't attempt it on distributed pandas backends like Ray. Query planning requires lazy evaluation so that we can understand queries

holistically– Modin on distributed backends is sometimes lazy, but we would have to make it more lazy.

Finally, we would like at least one backend to behave in a kind of "serverless" way, meaning that the resources can scale up and down as the user needs. For example,
while the data is small, it may be best to keep everything on one compute node, and only add another node, when the data is sufficiently large.




Here are some of the problems we face right now.

First, there are quite a number of open bugs due to mismatches with pandas. Some come from subtleties of trying to decompose a single pandas method in a distributed
way. Sometimes a method that makes sense for the whole frame only makes sense in one partition. e..g here. Sometimes, the metadata for one partition will depend on results of methods
in other partitions, e.g. here.

Users are also not satisfied with the performance of some common methods. We have partially implemented some kinds of merge, and there's a lot left to improve their performance. We
have made some progress with a new shuffling approach to methods that move data across all the partitions like sort_values and groupby transforms. There is more work to be done for
those, though.

To get good performance for scripts that chain methods, we need more

lazy execution. However, lazy and async execution change the user experience in ways that are unexpected or hard to understand. There are some bugs that we can't detect until execution, e.g. if a user-defined function raises an error. Also, currently users often think an operations was really fast because it's happening asynchronously or not happening at all.

## Opportunities for collaboration at this summit

- Users commonly want to use modin with other libraries for statistics, machine learning, and visualization that already work with pandas and numpy
  - Some do not work at all, some work partially
  - Some work inefficiently, e.g. might require materializing all data or iterating through all of it
  - Will the dataframe interchange protocol help?
- We could use modin to execute a different API, perhaps one that is explicitly for lazy execution, like polars...
- ... or we could use another dataframe library as a backend for modin.pandas
- Understand what the ideal lazy dataframe experience is for users
  - polars, ibis, dask, pyspark.pandas all have some version of this
- Do we think users often care about row order in dataframes?
  - Maintaining order with a relational backend can be complex and slow
  - ibis tables are not ordered, pyspark dataframes not ordered by default for performance

These are some topics that we could benefit from discussing in this group.

Often, users want to use modin with other libraries for statistics, machine learning, and visualization that already work with pandas and numpy. These libraries sometimes
don't work well with modin dataframes, and sometimes they won't work at all. There can be explicit type checks like isinstance(x, pandas.DataFrame), but the libraries
can break in other ways as well. The user can convert their modin object to pandas with df._to_pandas(), but that might be expensive.

We can use the existing modin backends to execute an API other than pandas, e.g. one like polars that is explicitly meant for building a query without executing it.

We could also use another dataframe library as a backend for modin.pandas.

A few different libraries represented here have some API for lazy execution. I know polars, ibis, dask, and pyspark.pandas all do that. I would like to get people's thoughts
on doing lazy execution with the pandas API exactly as it is. Should lazy execution be the default for Modin? Is it a pain point that every time someone switches between
libraries, they have to learn a new way of writing queries?

I also wonder what we think as a group about order in dataframes. Maintaining order when using a relational backend can be complex and slow. As far as I know, ibis dataframes
are not ordered, pyspark dataframes are not ordered by default, and I think the rest of the libraries here are ordered. Order matters for things like point updates (e.g.
df.iloc[0, 4] = 5) and if users care about determinism, e.g. head() should always print the same rows. Do we think users care about those?

# Thank you!

Mahesh Vashishtha
mahesh@ponder.io