Modin

Release 0.8.0

Sep 18, 2020

Installation


```
# import pandas as pd
import modin.pandas as pd
```

Modin uses Ray or Dask to provide an effortless way to speed up your pandas notebooks, scripts, and libraries. Unlike other distributed DataFrame libraries, Modin provides seamless integration and compatibility with existing pandas code. Even using the DataFrame constructor is identical.

```
import modin.pandas as pd
import numpy as np

frame_data = np.random.randint(0, 100, size=(2**10, 2**8))
df = pd.DataFrame(frame_data)
```

To use Modin, you do not need to know how many cores your system has and you do not need to specify how to distribute the data. In fact, you can continue using your previous pandas notebooks while experiencing a considerable speedup from Modin, even on a single machine. Once you've changed your import statement, you're ready to use Modin just like you would pandas.

CHAPTER 1

Installation and choosing your compute engine

Modin can be installed from PyPI:

```
pip install modin
```

If you don't have Ray or Dask installed, you will need to install Modin with one of the targets:

```
pip install modin[ray] # Install Modin dependencies and Ray to run on Ray
pip install modin[dask] # Install Modin dependencies and Dask to run on Dask
pip install modin[all] # Install all of the above
```

Modin will automatically detect which engine you have installed and use that for scheduling computation!

If you want to choose a specific compute engine to run on, you can set the environment variable `MODIN_ENGINE` and Modin will do computation with that engine:

```
export MODIN_ENGINE=ray  # Modin will use Ray
export MODIN_ENGINE=dask  # Modin will use Dask
```

This can also be done within a notebook/interpreter before you import Modin:

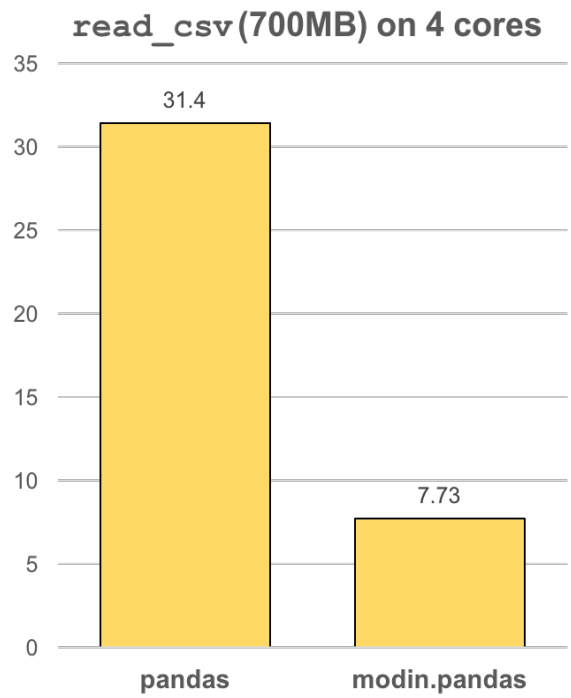
```
import os

os.environ["MODIN_ENGINE"] = "ray"  # Modin will use Ray
os.environ["MODIN_ENGINE"] = "dask"  # Modin will use Dask

import modin.pandas as pd
```

CHAPTER 2

Faster pandas, even on your laptop



The `modin.pandas` DataFrame is an extremely light-weight parallel DataFrame. Modin transparently distributes the data and computation so that all you need to do is continue using the pandas API as you were before installing Modin. Unlike other parallel DataFrame systems, Modin is an extremely light-weight, robust DataFrame. Because it is so light-weight, Modin provides speed-ups of up to 4x on a laptop with 4 physical cores.

In pandas, you are only able to use one core at a time when you are doing computation of any kind. With Modin, you are able to use all of the CPU cores on your machine. Even in `read_csv`, we see large gains by efficiently distributing the work across your entire machine.

```
import modin.pandas as pd

df = pd.read_csv("my_dataset.csv")
```

CHAPTER 3

Modin is a DataFrame for datasets from 1MB to 1TB+

We have focused heavily on bridging the solutions between DataFrames for small data (e.g. pandas) and large data. Often data scientists require different tools for doing the same thing on different sizes of data. The DataFrame solutions that exist for 1MB do not scale to 1TB+, and the overheads of the solutions for 1TB+ are too costly for datasets in the 1KB range. With Modin, because of its light-weight, robust, and scalable nature, you get a fast DataFrame at 1MB and 1TB+.

Modin is currently under active development. Requests and contributions are welcome!

3.1 Installation

There are a couple of ways to install Modin. Most users will want to install with `pip`, but some users may want to build from the master branch on the GitHub repo. The master branch has the most recent patches, but may be less stable than a release installed from `pip`.

3.1.1 Installing with pip

Stable version

Modin can be installed with pip. To install the most recent stable release run the following:

```
pip install -U modin # -U for upgrade in case you have an older version
```

If you don't have Ray or Dask installed, you will need to install Modin with one of the targets:

```
pip install modin[ray] # Install Modin dependencies and Ray to run on Ray
pip install modin[dask] # Install Modin dependencies and Dask to run on Dask
pip install modin[all] # Install all of the above
```

Modin will automatically detect which engine you have installed and use that for scheduling computation!

Release candidates

Before most major releases, we will upload a release candidate to If you would like to install a pre-release of Modin, run the following:

```
pip install --pre modin
```

These pre-releases are uploaded for dependencies and users to test their existing code to ensure that it still works. If you find something wrong, please raise an issue or email the bug reporter: bug_reports@modin.org.

Installing specific dependency sets

Modin has a number of specific dependency sets for running Modin on different backends or for different functionalities of Modin. Here is a list of dependency sets for Modin:

```
pip install "modin[dask]" # If you want to use the Dask backend
```

3.1.2 Installing from the GitHub master branch

If you'd like to try Modin using the most recent updates from the master branch, you can also use `pip`.

```
pip install git+https://github.com/modin-project/modin
```

This will install directly from the repo without you having to manually clone it! Please be aware that these changes have not made it into a release and may not be completely stable.

3.1.3 Windows

For installation on Windows, we recommend using the Dask Engine. Ray does not support Windows, so it will not be possible to install `modin[ray]` or `modin[all]`. It is possible to use Windows Subsystem For Linux (WSL), but this is generally not recommended due to the limitations and poor performance of Ray on WSL, a roughly 2-3x cost. To install with the Dask engine, run the following using `pip`:

```
pip install modin[dask]
```

You may already have a recent version of Dask installed, in which case you can simply `pip install modin`.

3.1.4 Building Modin from Source

If you're planning on contributing to Modin, you will need to ensure that you are building Modin from the local repository that you are working off of. Occasionally, there are issues in overlapping Modin installs from pypi and from source. To avoid these issues, we recommend uninstalling Modin before you install from source:

```
pip uninstall modin
```

To build from source, you first must clone the repo. We recommend forking the repository first through the GitHub interface, then cloning as follows:

```
git clone https://github.com/<your-github-username>/modin.git
```

Once cloned, `cd` into the `modin` directory and use `pip` to install:

```
cd modin
pip install -e .
```

3.2 Using Modin

Modin is an early stage DataFrame library that wraps pandas and transparently distributes the data and computation, accelerating your pandas workflows with one line of code change. The user does not need to know how many cores their system has, nor do they need to specify how to distribute the data. In fact, users can continue using their previous pandas notebooks while experiencing a considerable speedup from Modin, even on a single machine. Only a modification of the import statement is needed, as we demonstrate below. Once you've changed your import statement, you're ready to use Modin just like you would pandas, since the API is identical to pandas.

```
# import pandas as pd
import modin.pandas as pd
```

Currently, we have part of the pandas API implemented and are working toward full functional parity with pandas.

3.2.1 Using Modin on a Single Node

In order to use the most up-to-date version of Modin, please follow the instructions on the installation page

Once you import the library, you should see something similar to the following output:

```
>>> import modin.pandas as pd

Waiting for redis server at 127.0.0.1:14618 to respond...
Waiting for redis server at 127.0.0.1:31410 to respond...
Starting local scheduler with the following resources: {'CPU': 4, 'GPU': 0}.

======================================================================
View the web UI at http://localhost:8889/notebooks/ray_ui36796.ipynb?
↪token=ac25867d62c4ae87941bc5a0ecd5f517dbf80bd8e9b04218
======================================================================
```

Once you have executed `import modin.pandas as pd`, you're ready to begin running your pandas pipeline as you were before.

3.2.2 APIs Supported

Please note, the API is not yet complete. For some methods, you may see the following:

```
NotImplementedError: To contribute to Modin, please visit github.com/modin-project/
↪modin.
```

We have compiled a list of currently supported methods.

If you would like to request a particular method be implemented, feel free to open an issue. Before you open an issue please make sure that someone else has not already requested that functionality.

3.2.3 Using Modin on a Cluster (experimental)

Modin is able to utilize Ray's built-in autoscaled cluster. However, this usage is still under heavy development. To launch a Ray autoscaled cluster using Amazon Web Service (AWS), you can use the file *examples/cluster/aws_example.yaml* as the config file when launching an autoscaled Ray cluster. For the commands, refer to the autoscaler documentation.

We will provide a sample config file for private servers and other cloud service providers as we continue to develop and improve Modin's cluster support.

3.2.4 Advanced usage (experimental)

In some cases, it may be useful to customize your Ray environment. Below, we have listed a few ways you can solve common problems in data management with Modin by customizing your Ray environment. It is possible to use any of Ray's initialization parameters, which are all found in Ray's documentation.

```
import ray
ray.init()
import modin.pandas as pd
```

Modin will automatically connect to the Ray instance that is already running. This way, you can customize your Ray environment for use in Modin!

Exceeding memory (Out of core pandas)

Modin experimentally supports out of core operations. See more on the Out of Core documentation page.

Reducing or limiting the resources Modin can use

By default, Modin will use all of the resources available on your machine. It is possible, however, to limit the amount of resources Modin uses to free resources for another task or user. Here is how you would limit the number of CPUs Modin used in your bash environment variables:

```
export MODIN_CPUS=4
```

You can also specify this in your python script with `os.environ`. **Make sure you update the CPUS before you import Modin!**:

```
import os
os.environ["MODIN_CPUS"] = "4"
import modin.pandas as pd
```

If you're using a specific engine and want more control over the environment Modin uses, you can start Ray or Dask in your environment and Modin will connect to it. **Make sure you start the environment before you import Modin!**

```
import ray
ray.init(num_cpus=4)
import modin.pandas as pd
```

Specifying `num_cpus` limits the number of processors that Modin uses. You may also specify more processors than you have available on your machine, however this will not improve the performance (and might end up hurting the performance of the system).

3.2.5 Examples

You can find an example on our recent blog post or on the Jupyter Notebook that we used to create the blog post.

3.3 Out of Core in Modin (experimental)

If you are working with very large files or would like to exceed your memory, you may change the primary location of the DataFrame. If you would like to exceed memory, you can use your disk as an overflow for the memory. This API is experimental in the context of Modin. Please let us know what you think!

3.3.1 Install Modin out of core

Modin now comes with all the dependencies for out of core functionality by default! See the installation page for more information on installing Modin.

3.3.2 Starting Modin with out of core enabled

Out of core is detected from an environment variable set in bash.

```
export MODIN_OUT_OF_CORE=true
```

We also set up a way to tell Modin how much memory you'd like to use. Currently, this only accepts the number of bytes. This can only exceed your memory if you have enabled `MODIN_OUT_OF_CORE`.

[Optional]: Set a limit on the out of core space for Modin

Warning: Make sure you have enough space in your disk for however many bytes you request for your DataFrame

This limits the amount of memory that Modin can use.

Here is how you set `MODIN_MEMORY`:

```
export MODIN_MEMORY=200000000000 # Set the number of bytes to 200GB
```

The default for Modin is 8x the memory on the machine.

3.3.3 Running an example with out of core

Before you run this, please make sure you follow the instructions listed above.

```
import modin.pandas as pd
import numpy as np
frame_data = np.random.randint(0, 100, size=(2**20, 2**8)) # 2GB each
df = pd.DataFrame(frame_data).add_prefix("col")
big_df = pd.concat([df for _ in range(20)]) # 20x2GB frames
print(big_df)
nan_big_df = big_df.isna() # The performance here represents a simple map
print(big_df.apply(lambda col: col.sum())) # apply along an entire axis (columns in␣
↪this case)
```

This example creates a 40GB DataFrame from 20 identical 2GB DataFrames and performs various operations on them. Feel free to play around with this code and let us know what you think!

3.4 Supported APIs and Defaulting to pandas

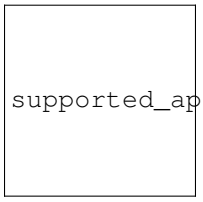
For your convenience, we have compiled a list of currently implemented APIs and methods available in Modin. This documentation is updated as new methods and APIs are merged into the master branch, and not necessarily correct as of the most recent release. In order to install the latest version of Modin, follow the directions found on the installation page.

3.4.1 Questions on implementation details

If you have a question about the implementation details or would like more information about an API or method in Modin, please contact the Modin developer mailing list.

3.4.2 Defaulting to pandas

The remaining unimplemented methods default to pandas. This allows users to continue using Modin even though their workloads contain functions not yet implemented in Modin. Here is a diagram of how we convert to pandas and perform the operation:



We first convert to a pandas DataFrame, then perform the operation. There is a performance penalty for going from a partitioned Modin DataFrame to pandas because of the communication cost and single-threaded nature of pandas. Once the pandas operation has completed, we convert the DataFrame back into a partitioned Modin DataFrame. This way, operations performed after something defaults to pandas will be optimized with Modin.

The exact methods we have implemented are listed in the respective subsections:

- DataFrame
- Series
- utilities
- I/O

We have taken a community-driven approach to implementing new methods. We did a study on pandas usage to learn what the most-used APIs are. Modin currently supports **93%** of the pandas API based on our study of pandas usage, and we are actively expanding the API.

3.5 `pd.DataFrame` supported APIs

The following table lists both implemented and not implemented methods. If you have need of an operation that is listed as not implemented, feel free to open an issue on the GitHub repository, or give a thumbs up to already created issues. Contributions are also welcome!

The following table is structured as follows: The first column contains the method name. The second column is a flag for whether or not there is an implementation in Modin for the method in the left column. `Y` stands for yes, `N` stands for no, `P` stands for partial (meaning some parameters may not be supported yet), and `D` stands for default to pandas.

DataFrame method	pandas Doc link	Implemented? (Y/N/P/D)	Notes for Current implementation
`T`	T	Y	
`abs`	abs	Y	
`add`	add	Y	Shuffles data in operations between DataFrames
`add_prefix`	add_prefix	Y	
`add_suffix`	add_suffix	Y	
`agg` / `aggregate`	agg / aggregate	P	• Dictionary `func` parameter defaults to pandas • Numpy operations default to pandas
`align`	align	D	
`all`	all	Y	
`any`	any	Y	
`append`	append	Y	
`apply`	apply	Y	See `agg`
`applymap`	applymap	Y	
`as_blocks`	as_blocks	D	Becomes a non-parallel object
`as_matrix`	as_matrix	D	Becomes a non-parallel object
`asfreq`	asfreq	D	
`asof`	asof	Y	
`assign`	assign	Y	
`astype`	astype	Y	
`at`	at	Y	
`at_time`	at_time	Y	
`axes`	axes	Y	
`between_time`	between_time	Y	
`bfill`	bfill	Y	
`blocks`	blocks	D	
`bool`	bool	Y	
`boxplot`	boxplot	D	
`clip`	clip	Y	
`clip_lower`	clip_lower	Y	
`clip_upper`	clip_upper	Y	
`combine`	combine	Y	
`combine_first`	combine_first	Y	
`compare`	**‘compare‘_**	D	
`copy`	copy	Y	
`corr`	corr	D	
`corrwith`	corrwith	D	
`count`	count	Y	

Continued on next page

Table 1 – continued from previous page

cov	cov	P	If DataFrame contains at least one NA/null value, then defaults to pandas
cummax	cummax	Y	
cummin	cummin	Y	
cumprod	cumprod	Y	
cumsum	cumsum	Y	
describe	describe	Y	
diff	diff	Y	
div	div	Y	See add
divide	divide	Y	See add
dot	dot	Y	
drop	drop	Y	
droplevel	droplevel	Y	
drop_duplicates	drop_duplicates	D	
dropna	dropna	Y	
dtypes	dtypes	Y	
duplicated	duplicated	Y	
empty	empty	Y	
eq	eq	Y	See add
equals	equals	Y	Requires shuffle, can be further optimized
eval	eval	Y	
ewm	ewm	D	
expanding	expanding	D	
explode	explode	D	
ffill	ffill	Y	
fillna	fillna	P	value parameter of type DataFrame defaults to pandas
filter	filter	Y	
first	first	Y	
first_valid_index	first_valid_index	Y	
floordiv	floordiv	Y	See add
from_dict	from_dict	D	
from_items	from_items	Y	
from_records	from_records	D	
ftypes	ftypes	Y	
ge	ge	Y	See add
get	get	Y	
groupby	groupby	Y	Not yet optimized for all operations
gt	gt	Y	See add
head	head	Y	
hist	hist	D	
iat	iat	Y	
idxmax	idxmax	Y	
idxmin	idxmin	Y	
iloc	iloc	Y	
infer_objects	infer_objects	D	

Continued on next page

Table 1 – continued from previous page

info	info	Y	
insert	insert	Y	
interpolate	interpolate	D	
isin	isin	Y	
isna	isna	Y	
isnull	isnull	Y	
items	items	Y	
iteritems	iteritems	Y	
iterrows	iterrows	Y	
itertuples	itertuples	Y	
join	join	P	When on is set to right or outer it defaults to pandas
keys	keys	Y	
kurt	kurt	Y	
kurtosis	kurtosis	Y	
last	last	Y	
last_valid_index	last_valid_index	Y	
le	le	Y	See add
loc	loc	Y	
lookup	lookup	D	
lt	lt	Y	See add
mad	mad	Y	
mask	mask	D	
max	max	Y	
mean	mean	Y	
median	median	Y	
melt	melt	Y	
memory_usage	memory_usage	Y	
merge	merge	P	Implemented the following cases: left_index=True and right_index=True, how=left and how=inner for all values of parameters except left_index=True and right_index=False or left_index=False and right_index=True. Defaults to pandas otherwise.
min	min	Y	
mod	mod	Y	
mode	mode	Y	
mul	mul	Y	See add
multiply	multiply	Y	See add

Continued on next page

Table 1 – continued from previous page

ndim	ndim	Y	
ne	ne	Y	See add
nlargest	nlargest	Y	
notna	notna	Y	
notnull	notnull	Y	
nsmallest	nsmallest	Y	
nunique	nunique	Y	
pct_change	pct_change	D	
pipe	pipe	Y	
pivot	pivot	Y	
pivot_table	pivot_table	D	
plot	plot	D	
pop	pop	Y	
pow	pow	Y	See add
prod	prod	Y	
product	product	Y	
quantile	quantile	Y	
query	query	P	Local variables not yet supported
radd	radd	Y	See add
rank	rank	Y	
rdiv	rdiv	Y	See add
reindex	reindex	Y	Shuffles data
reindex_like	reindex_like	D	
rename	rename	Y	
rename_axis	rename_axis	Y	
reorder_levels	reorder_levels	Y	
replace	replace	Y	
resample	resample	Y	
reset_index	reset_index	Y	
rfloordiv	rfloordiv	Y	See add
rmod	rmod	Y	See add
rmul	rmul	Y	See add
rolling	rolling	Y	
round	round	Y	
rpow	rpow	Y	See add
rsub	rsub	Y	See add
rtruediv	rtruediv	Y	See add
sample	sample	Y	
select_dtypes	select_dtypes	Y	
sem	sem	Y	
set_axis	set_axis	Y	
set_index	set_index	Y	
shape	shape	Y	
shift	shift	Y	
size	size	Y	
skew	skew	Y	
slice_shift	slice_shift	Y	
sort_index	sort_index	Y	
sort_values	sort_values	Y	Shuffles data

Continued on next page

Table 1 – continued from previous page

sparse	sparse	N	
squeeze	squeeze	Y	
stack	stack	Y	
std	std	Y	
style	style	D	
sub	sub	Y	See add
subtract	subtract	Y	See add
sum	sum	Y	
swapaxes	swapaxes	Y	
swaplevel	swaplevel	Y	
tail	tail	Y	
take	take	Y	
to_clipboard	to_clipboard	D	
to_csv	to_csv	D	
to_dense	to_dense	D	
to_dict	to_dict	D	
to_excel	to_excel	D	
to_feather	to_feather	D	
to_gbq	to_gbq	D	
to_hdf	to_hdf	D	
to_html	to_html	D	
to_json	to_json	D	
to_latex	to_latex	D	
to_msgpack	to_msgpack	D	
to_parquet	to_parquet	D	
to_period	to_period	D	
to_pickle	to_pickle	D	
to_records	to_records	D	
to_sparse	to_sparse	D	
to_sql	to_sql	Y	
to_stata	to_stata	D	
to_string	to_string	D	
to_timestamp	to_timestamp	D	
to_xarray	to_xarray	D	
transform	transform	Y	
transpose	transpose	Y	
truediv	truediv	Y	See add
truncate	truncate	Y	
tshift	tshift	Y	
tz_convert	tz_convert	Y	
tz_localize	tz_localize	Y	
unstack	unstack	Y	
update	update	Y	
values	values	Y	
var	var	Y	
where	where	Y	

3.6 pd.Series supported APIs

The following table lists both implemented and not implemented methods. If you have need of an operation that is listed as not implemented, feel free to open an issue on the GitHub repository, or give a thumbs up to already created issues. Contributions are also welcome!

The following table is structured as follows: The first column contains the method name. The second column is a flag for whether or not there is an implementation in Modin for the method in the left column. `Y` stands for yes, `N` stands for no, `P` stands for partial (meaning some parameters may not be supported yet), and `D` stands for default to pandas. To learn more about the implementations that default to pandas, see the related section on Defaulting to pandas.

Series method	Modin Implementation? (Y/N/P/D)
abs	Y
add	Y
add_prefix	Y
add_suffix	Y
agg	Y
aggregate	Y
align	D
all	Y
any	Y
append	Y
apply	Y
argmax	Y
argmin	Y
argsort	D
array	D
as_blocks	D
as_matrix	Y
asfreq	D
asobject	D
asof	Y
astype	Y
at	Y
at_time	Y
autocorr	Y
axes	Y
base	D
between	D
between_time	Y
bfill	Y
blocks	D
bool	Y
cat	D
clip	Y
clip_lower	Y
clip_upper	Y
combine	Y
combine_first	Y
compare	D
compress	D

Continued on next page

Table 2 – continued from previous page

copy	Y
corr	Y
count	Y
cov	Y
cummax	Y
cummin	Y
cumprod	Y
cumsum	Y
data	D
describe	Y
diff	Y
div	Y
divide	Y
divmod	Y
dot	Y
drop	Y
drop_duplicates	Y
droplevel	Y
dropna	Y
dt	Y
dtype	Y
dtypes	Y
duplicated	Y
empty	Y
eq	Y
equals	Y
ewm	D
expanding	D
explode	D
factorize	D
ffill	Y
fillna	Y
filter	Y
first	Y
first_valid_index	Y
flags	D
floordiv	Y
from_array	D
ftype	Y
ftypes	Y
ge	Y
get	Y
get_dtype_counts	Y
get_ftype_counts	Y
get_value	D
get_values	D
groupby	D
gt	Y
hasnans	Y
head	Y

Continued on next page

Table 2 – continued from previous page

hist	D
iat	Y
idxmax	Y
idxmin	Y
iloc	Y
imag	D
index	Y
infer_objects	D
interpolate	D
is_monotonic	Y
is_monotonic_decreasing	Y
is_monotonic_increasing	Y
is_unique	Y
isin	Y
isna	Y
isnull	Y
item	Y
items	Y
itemsize	D
iteritems	Y
keys	Y
kurt	Y
kurtosis	Y
last	Y
last_valid_index	Y
le	Y
loc	Y
lt	Y
mad	Y
map	Y
mask	D
max	Y
mean	Y
median	Y
memory_usage	Y
min	Y
mod	Y
mode	Y
mul	Y
multiply	Y
name	Y
nbytes	D
ndim	Y
ne	Y
nlargest	Y
nonzero	Y
notna	Y
notnull	Y
nsmallest	Y
nunique	Y

Continued on next page

Table 2 – continued from previous page

pct_change	D
pipe	Y
plot	D
pop	Y
pow	Y
prod	Y
product	Y
ptp	D
put	D
quantile	Y
radd	Y
rank	Y
ravel	Y
rdiv	Y
rdivmod	Y
real	D
reindex	Y
reindex_like	Y
rename	Y
rename_axis	Y
reorder_levels	D
repeat	Y
replace	Y
resample	Y
reset_index	Y
rfloordiv	Y
rmod	Y
rmul	Y
rolling	Y
round	Y
rpow	Y
rsub	Y
rtruediv	Y
sample	Y
searchsorted	Y
sem	Y
set_axis	Y
set_value	D
shape	Y
shift	Y
size	Y
skew	Y
slice_shift	Y
sort_index	Y
sort_values	Y
sparse	Y
squeeze	Y
std	Y
str	Y
strides	D

Continued on next page

Table 2 – continued from previous page

sub	Y
subtract	Y
sum	Y
swapaxes	Y
swaplevel	Y
tail	Y
take	Y
to_clipboard	D
to_csv	D
to_dense	D
to_dict	D
to_excel	D
to_frame	Y
to_hdf	D
to_json	D
to_latex	D
to_list	D
to_msgpack	D
to_numpy	D
to_period	D
to_pickle	D
to_sparse	D
to_sql	Y
to_string	D
to_timestamp	D
to_xarray	D
tolist	D
transform	Y
transpose	Y
truediv	Y
truncate	Y
tshift	Y
tz_convert	Y
tz_localize	Y
unique	Y
unstack	Y
update	Y
valid	D
value_counts	Y
values	Y
var	Y
view	D
where	Y

3.7 pandas Utilities Supported

If you `import modin.pandas as pd` the following operations are available from `pd.<op>`, e.g. `pd.concat`. If you do not see an operation that pandas enables and would like to request it, feel free to open an issue. Make sure you tell us your primary use-case so we can make it happen faster!

The following table is structured as follows: The first column contains the method name. The second column is a flag for whether or not there is an implementation in Modin for the method in the left column. `Y` stands for yes, `N` stands for no, `P` stands for partial (meaning some parameters may not be supported yet), and `D` stands for default to pandas.

Utility method	Modin Implementation? (Y/N/P/D)	Notes for Current implementation
pd.concat	Y	
pd.eval	Y	
pd.unique	Y	
`pd.value_counts`	Y	
pd.cut	D	
pd.to_numeric	D	
pd.factorize	D	
pd.qcut	D	
`pd.match`	D	
pd.to_datetime	D	
pd.get_dummies	Y	
pd.date_range	D	
pd.bdate_range	D	
pd.to_timedelta	D	
`pd.options`	Y	
`pd.datetime`	D	

3.7.1 Other objects & structures

This list is a list of objects not currently distributed by Modin. All of these objects are compatible with the distributed components of Modin. If you are interested in contributing a distributed version of any of these objects, feel free to open a pull request.

- Panel
- Index
- MultiIndex
- CategoricalIndex
- DatetimeIndex
- Timedelta
- Timestamp
- NaT
- PeriodIndex
- Categorical
- Interval
- UInt8Dtype
- UInt16Dtype
- UInt32Dtype
- UInt64Dtype
- SparseDtype

- Int8Dtype
- Int16Dtype
- Int32Dtype
- Int64Dtype
- CategoricalDtype
- DatetimeTZDtype
- IntervalDtype
- PeriodDtype
- RangeIndex
- Int64Index
- UInt64Index
- Float64Index
- TimedeltaIndex
- IntervalIndex
- IndexSlice
- TimeGrouper
- Grouper
- array
- Period
- DateOffset
- ExcelWriter
- SparseArray
- SparseSeries
- SparseDataFrame

3.8 `pd.read_<file>` and I/O APIs

A number of IO methods default to pandas. We have parallelized `read_csv` and `read_parquet`, though many of the remaining methods can be relatively easily parallelized. Some of the operations default to the pandas implementation, meaning it will read in serially as a single, non-distributed DataFrame and distribute it. Performance will be affected by this.

The following table is structured as follows: The first column contains the method name. The second column is a flag for whether or not there is an implementation in Modin for the method in the left column. `Y` stands for yes, `N` stands for no, `P` stands for partial (meaning some parameters may not be supported yet), and `D` stands for default to pandas.

IO method	Modin Implementation? (Y/N/P/D)	Notes for Current implementation
read_csv	Y	
read_table	Y	
read_parquet	Y	
read_json	P	Implemented for `lines=True`
read_html	D	
read_clipboard	D	
read_excel	D	
read_hdf	Y	
read_feather	Y	
read_msgpack	D	
read_stata	D	
read_sas	D	
read_pickle	D	
read_sql	Y	

3.9 Ray

This section describes usage related documents for the Pandas on Ray component of Modin.

Modin uses Pandas on Ray by default, but if you wanted to be explicit, you could set the following environment variables:

```
export MODIN_ENGINE=ray
export MODIN_BACKEND=pandas
```

3.10 Pandas on Dask

The Dask engine and documentation could use your help! Consider opening a pull request or an issue to contribute or ask clarifying questions.

3.11 Pyarrow on Ray

Coming Soon!

3.12 Modin SQL API

Modin's SQL API is currently a conceptual plan, Coming Soon!

3.12.1 Plans for future development

Our plans with the SQL API for Modin are to create an interface that allows you to intermix SQL and pandas operations without copying the entire dataset into a new structure between the two. This is possible due to the architecture of Modin. Currently, Modin has a query compiler that acts as an intermediate layer between the query language (e.g. SQL, pandas) and the execution (See architecture documentation for details).

We have implemented a simple example that can be found below. Feedback welcome!

```
>>> import modin.sql as sql
>>>
>>> conn = sql.connect("db_name")
>>> c = conn.cursor()
>>> c.execute("CREATE TABLE example (col1, col2, column 3, col4)")
>>> c.execute("INSERT INTO example VALUES ('1', 2.0, 'A String of information', True)
↪")
  col1  col2                 column 3  col4
0    1   2.0  A String of information  True

>>> c.execute("INSERT INTO example VALUES ('6', 17.0, 'A String of different␣
↪information', False)")
  col1  col2                           column 3   col4
0    1   2.0            A String of information   True
1    6  17.0  A String of different information  False
```

3.13 Contributing

3.13.1 Getting Started

If you're interested in getting involved in the development of Modin, but aren't sure where start, take a look at the issues tagged Good first issue or Documentation. These are issues that would be good for getting familiar with the codebase and better understanding some of the more complex components of the architecture. There is documentation here about the architecture that you will want to review in order to get started.

Also, feel free to join the discussions on the developer mailing list.

3.13.2 LINUX KERNEL CERTIFICATE OF ORIGIN V 1.1

"By making a contribution to this project, I certify that:

1.) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or 2.) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or 3.) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it. 4.) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved."

3.13.3 How to sign-off commits

Modin requires a sign-off message in the following format appear on each commit in the pull request:

```
This is my commit message

Signed-off-by: Random J Developer <random@developer.example.org>
```

We have the same requirements for using the signed-off-by process as the Linux kernel has.

In short, you need to include a signed-off-by tag in every patch:

Signed-off-by is a developer's certification that he or she has the right to submit the patch for inclusion into the project. It is an agreement to the Developer's Certificate of Origin (above). Code without a proper signoff cannot be merged into the master branch. Note: You must use your real name (sorry, no pseudonyms or anonymous contributions.)

The text can either be manually added to your commit body, or you can add either `-s` or `--signoff` to your usual `git commit` commands:

```
git commit --signoff
git commit -s
```

This will use your default git configuration which is found in .git/config. To change this, you can use the following commands:

```
git config --global user.name "FIRST_NAME LAST_NAME"
git config --global user.email "MY_NAME@example.com"
```

If you have authored a commit that is missing the signed-off-by line, you can amend your commits and push them to GitHub.

```
git commit --amend --signoff
```

If you've pushed your changes to GitHub already you'll need to force push your branch after this with `git push -f`.

3.13.4 Commit Message formatting

To ensure that all commit messages in the master branch follow a specific format, we enforce that all commit messages must follow the following format:

```
FEAT-#9999: Add `DataFrame.rolling` functionality, to enable rolling window operations
```

The `FEAT` component represents the type of commit. This component of the commit message can be one of the following:

- FEAT: A new feature that is added
- DOCS: Documentation improvements or updates
- FIX: A bugfix contribution
- REFACTOR: Moving or removing code without change in functionality
- TEST: Test updates or improvements

The `#9999` component of the commit message should be the issue number in the Modin GitHub issue tracker: https://github.com/modin-project/modin/issues. This is important because it links commits to their issues.

The commit message should follow a colon (:) and be descriptive and succinct.

3.13.5 Development Dependencies

We recommend doing development in a virtualenv, though this decision is ultimately yours. You will want to run the following in order to install all of the required dependencies for running the tests and formatting the code:

```
pip install -r requirements.txt
```

3.13.6 Code Formatting and Lint

We use black for code formatting. Before you submit a pull request, please make sure that you run the following from the project root:

```
black modin/
```

We also use flake8 to check linting errors. Running the following from the project root will ensure that it passes the lint checks on Travis:

```
flake8 .
```

We test that this has been run on our Travis CI test suite. If you do this and find that the tests are still failing, try updating your version of black and flake8.

3.13.7 Adding a test

If you find yourself fixing a bug or adding a new feature, don't forget to add a test to the test suite to verify its correctness! More on testing and the layout of the tests can be found in our testing documentation. We ask that you follow the existing structure of the tests for ease of maintenance.

3.13.8 Running the tests

To run the entire test suite, run the following from the project root:

```
pytest modin/pandas/test
```

The test suite is very large, and may take a long time if you run every test. If you've only modified a small amount of code, it may be sufficient to run a single test or some subset of the test suite. In order to run a specific test run:

```
pytest modin/pandas/test::test_new_functionality
```

The entire test suite is automatically run for each pull request.

3.13.9 Contributing a new execution framework or in-memory format

If you are interested in contributing support for a new execution framework or in-memory format, please make sure you understand the architecture of Modin.

The best place to start the discussion for adding a new execution framework or in-memory format is the developer mailing list.

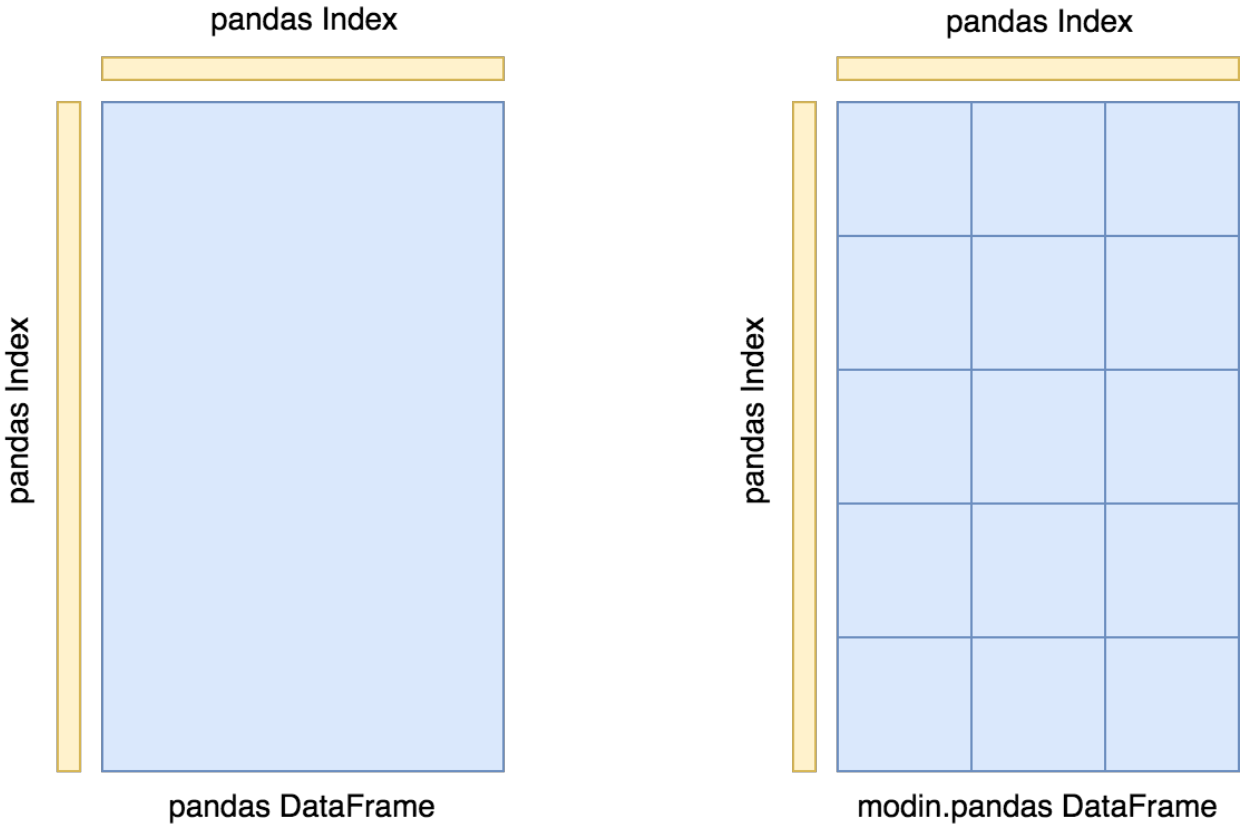
More docs on this coming soon...

3.14 Architecture

In this documentation page, we will lay out the overall architecture for Modin, as well as go into detail about the implementation and other important details. This document also contains important reference information for those interested in contributing new functionality, bugfixes, and enhancements.

3.14.1 DataFrame Partitioning

The Modin DataFrame architecture follows in the footsteps of modern architectures for database and high performance matrix systems. We chose a partitioning schema that partitions along both columns and rows because it gives Modin flexibility and scalability in both the number of columns and the number of rows supported. The following figure illustrates this concept.



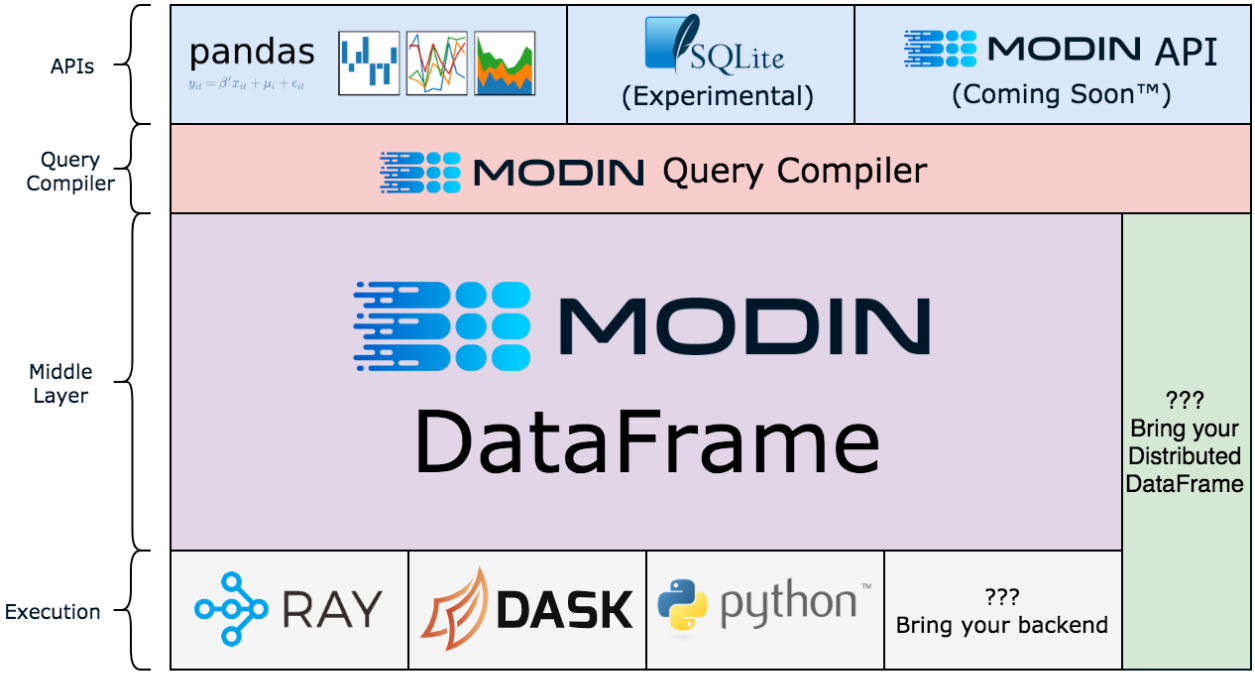
Currently, each partition's memory format is a pandas DataFrame. In the future, we will support additional in-memory formats for the backend, namely Arrow tables.

Index

We currently use the `pandas.Index` object for both indexing columns and rows. In the future, we will implement a distributed, pandas-compatible Index object in order remove this scaling limitation from the system. It does not start to become a problem until you are operating on more than 10's of billions of columns or rows, so most workloads will not be affected by this scalability limit. **Important note**: If you are using the default index (`pandas.RangeIndex`) there is a fixed memory overhead (~200 bytes) and there will be no scalability issues with the index.

3.14.2 System Architecture

The figure below outlines the general architecture for the implementation of Modin.



Modin is logically separated into different layers that represent the hierarchy of a typical Database Management System. Abstracting out each component allows us to individually optimize and swap out components without affecting the rest of the system. We can implement, for example, new compute kernels that are optimized for a certain type of data and can simply plug it in to the existing infrastructure by implementing a small interface. It can still be distributed by our choice of compute engine with the logic internally.

API

The API is the outer-most layer that faces users. The majority of our current effort is spent implementing the components of the pandas API. We have implemented a toy example for a sqlite API as a proof of concept, but this isn't ready for usage/testing. There are also plans to expose the Modin DataFrame API as a reduced API set that encompasses the entire pandas/dataframe API.

Query Compiler

The Query Compiler receives queries from the pandas API layer. The API layer's responsibility is to ensure clean input to the Query Compiler. The Query Compiler must have knowledge of the compute kernels/in-memory format of the data in order to efficiently compile the queries.

The Query Compiler is responsible for sending the compiled query to the Modin DataFrame. In this design, the Query Compiler does not have information about where or when the query will be executed, and gives the control of the partition layout to the Modin DataFrame.

In the interest of reducing the pandas API, the Query Compiler layer closely follows the pandas API, but cuts out a large majority of the repetition.

Modin DataFrame

At this layer, operations can be performed lazily. Currently, Modin executes most operations eagerly in an attempt to behave as pandas does. Some operations, e.g. `transpose` are expensive and create full copies of the data in-memory. In these cases, we can wait until another operation triggers computation. In the future, we plan to add additional query planning and laziness to Modin to ensure that queries are performed efficiently.

The structure of the Modin DataFrame is extensible, such that any operation that could be better optimized for a given backend can be overridden and optimized in that way.

This layer has a significantly reduced API from the QueryCompiler and the user-facing API. Each of these APIs represents a single way of performing a given operation or behavior. Some of these are expanded for convenience/understanding. The API abstractions are as follows:

Modin DataFrame API

- `mask`: Indexing/masking/selecting on the data (by label or by integer index).
- `copy`: Create a copy of the data.
- `mapreduce`: Reduce the dimension of the data.
- `foldreduce`: Reduce the dimension of the data, but entire column/row information is needed.
- `map`: Perform a map.
- `fold`: Perform a fold.
- `apply_<type>`: Apply a function that may or may not change the shape of the data.
  - `full_axis`: Apply a function requires knowledge of the entire axis.
  - `full_axis_select_indices`: Apply a function performed on a subset of the data that requires knowledge of the entire axis.
  - `select_indices`: Apply a function to a subset of the data. This is mainly used for indexing.
- `binary_op`: Perform a function between two dataframes.
- `concat`: Append one or more dataframes to either axis of this dataframe.
- `transpose`: Swap the axes (columns become rows, rows become columns).
- `groupby`:
  - `groupby_reduce`: Perform a reduction on each group.
  - `groupby_apply`: Apply a function to each group.
- **take functions**
  - `head`: Take the first `n` rows.
  - `tail`: Take the last `n` rows.
  - `front`: Take the first `n` columns.
  - `back`: Take the last `n` columns.
- **import/export functions**
  - `from_pandas`: Convert a pandas dataframe to a Modin dataframe.
  - `to_pandas`: Convert a Modin dataframe to a pandas dataframe.
  - `to_numpy`: Convert a Modin dataframe to a NumPy array.

More documentation can be found internally in the code. This API is not complete, but represents an overwhelming majority of operations and behaviors.

This API can be implemented by other distributed/parallel DataFrame libraries and plugged in to Modin as well. Create an issue or discuss on our Discourse for more information!

The Modin DataFrame is responsible for the data layout and shuffling, partitioning, and serializing the tasks that get sent to each partition. Other implementations of the Modin DataFrame interface will have to handle these as well.

Execution Engine/Framework

This layer is what Modin uses to perform computation on a partition of the data. The Modin DataFrame is designed to work with task parallel frameworks, but with some effort, a data parallel framework is possible.

Internal abstractions

These abstractions are not included in the above architecture, but are important to the internals of Modin.

Partition Manager

The Partition Manager can change the size and shape of the partitions based on the type of operation. For example, certain operations are complex and require access to an entire column or row. The Partition Manager can convert the block partitions to row partitions or column partitions. This gives Modin the flexibility to perform operations that are difficult in row-only or column-only partitioning schemas.

Another important component of the Partition Manager is the serialization and shipment of compiled queries to the Partitions. It maintains metadata for the length and width of each partition, so when operations only need to operate on or extract a subset of the data, it can ship those queries directly to the correct partition. This is particularly important for some operations in pandas which can accept different arguments and operations for different columns, e.g. `fillna` with a dictionary.

This abstraction separates the actual data movement and function application from the DataFrame layer to keep the DataFrame API small and separately optimize the data movement and metadata management.

Partition

Partitions are responsible for managing a subset of the DataFrame. As is mentioned above, the DataFrame is partitioned both row and column-wise. This gives Modin scalability in both directions and flexibility in data layout. There are a number of optimizations in Modin that are implemented in the partitions. Partitions are specific to the execution framework and in-memory format of the data. This allows Modin to exploit potential optimizations across both of these. These optimizations are explained further on the pages specific to the execution framework.

Supported Execution Frameworks and Memory Formats

This is the list of execution frameworks and memory formats supported in Modin. If you would like to contribute a new execution framework or memory format, please see the documentation page on Contributing.

- **Pandas on Ray**
  - Uses the Ray execution framework.
  - The compute kernel/in-memory format is a pandas DataFrame.
- **Pandas on Dask**

  - Uses the Dask Futures execution framework.
  - The compute kernel/in-memory format is a pandas DataFrame.

- **Pyarrow on Ray (experimental)**
  - Uses the Ray execution framework.
  - The compute kernel/in-memory format is a pyarrow Table.

3.15 Troubleshooting

We hope your experience with Modin is bug-free, but there are some quirks about Modin that may require troubleshooting.

3.15.1 Frequently encountered issues

This is a list of the most frequently encountered issues when using Modin. Some of these are working as intended, while others are known bugs that are being actively worked on.

Error During execution: `ArrowIOError: Broken Pipe`

One of the more frequently encountered issues is an `ArrowIOError: Broken Pipe`. This error can happen in a couple of different ways. One of the most common ways this is encountered is from pressing **CTRL + C** sending a `KeyboardInterrupt` to Modin. In Ray, when a `KeyboardInterrupt` is sent, Ray will shutdown. This causes the `ArrowIOError: Broken Pipe` because there is no longer an available plasma store for working on remote tasks. This is working as intended, as it is not yet possible in Ray to kill a task that has already started computation.

The other common way this `Error` is encountered is to let your computer go to sleep. As an optimization, Ray will shutdown whenever the computer goes to sleep. This will result in the same issue as above, because there is no longer a running instance of the plasma store.

Solution

Restart your interpreter or notebook kernel.

Avoiding this Error

Avoid using `KeyboardInterrupt` and keeping your notebook or terminal running while your machine is asleep. If you do `KeyboardInterrupt`, you must restart the kernel or interpreter.

Error during execution: `ArrowInvalid: Maximum size exceeded (2GB)`

Encountering this issue means that the limits of the Arrow plasma store have been exceeded by the partitions of your data. This can be encountered during shuffling data or operations that require multiple datasets. This will only affect extremely large DataFrames, and can potentially be worked around by setting the number of partitions. This error is being actively worked on and should be resolved in a future release.

Solution

```
import modin.pandas as pd
pd.DEFAULT_NPARTITIONS = 2 * pd.DEFAULT_NPARTITIONS
```

This will set the number of partitions to a higher count, and reduce the size in each. If this does not work for you, please open an issue.

Hanging on `import modin.pandas as pd`

This can happen when Ray fails to start. It will keep retrying, but often it is faster to just restart the notebook or interpreter. Generally, this should not happen. Most commonly this is encountered when starting multiple notebooks or interpreters in quick succession.

Solution

Restart your interpreter or notebook kernel.

Avoiding this Error

Avoid starting many Modin notebooks or interpreters in quick succession. Wait 2-3 seconds before starting the next one.

3.16 Contact

3.16.1 Mailing List

https://groups.google.com/forum/#!forum/modin-dev

General questions, potential contributors, and ideas should be directed to the developer mailing list. It is an open Google Group, so feel free to join anytime! If you are unsure about where to ask or post something, the mailing list is a good place to ask as well.

3.16.2 Issues

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

Bug reports and feature requests should be directed to the issues page of the Modin GitHub repo.