Hamilton Global User Group February 2024 Meetup

What is Hamilton?
Hamilton helps data scientists and engineers define testable, modular, self-documenting dataflows, that encode lineage and metadata.
Runs and scales everywhere python does.

Icebreaker: Name and what you're using Hamilton for/looking for.

DAGWORKS February 2024

Agenda

1. Community Spotlight
2. Deep Dive
3. Roadmap
4. Open 🎤

Community Spotlight:

🔎 "**How we migrated our feature calculation to Hamilton**" by Arthur Andres.

DAGWORKS

Deep Dive:
Mental Models for Structuring Projects

DAGWORKS

My Mental Model for Structuring Projects

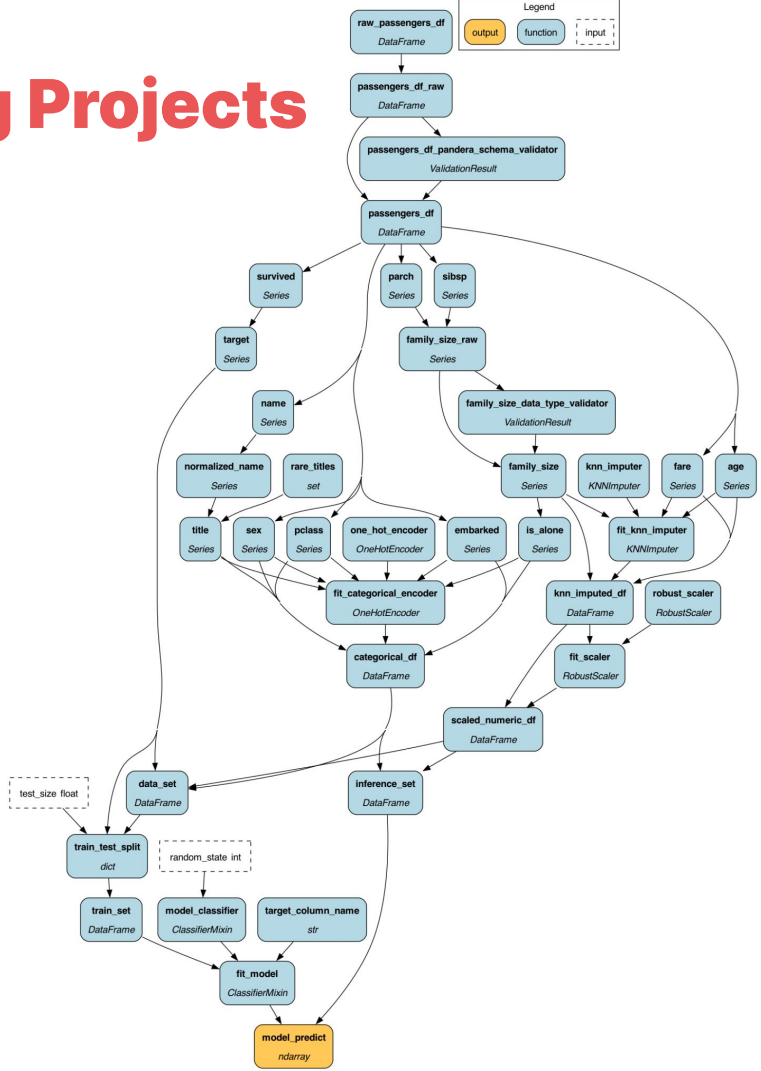
With Hamilton you:

1. Write functions
2. Functions are organized into modules.

My Mental Model for Structuring Projects

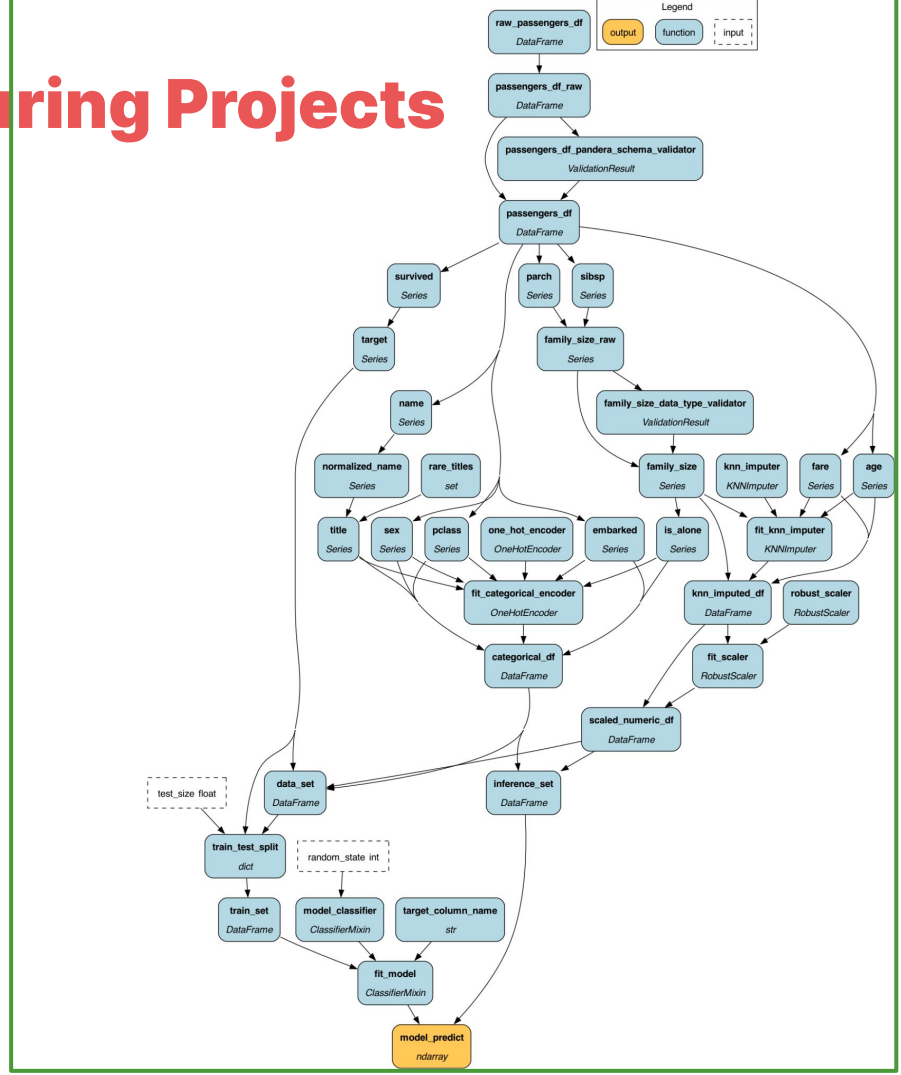
With Hamilton you:

1. Write functions
2. Functions are organized into modules.



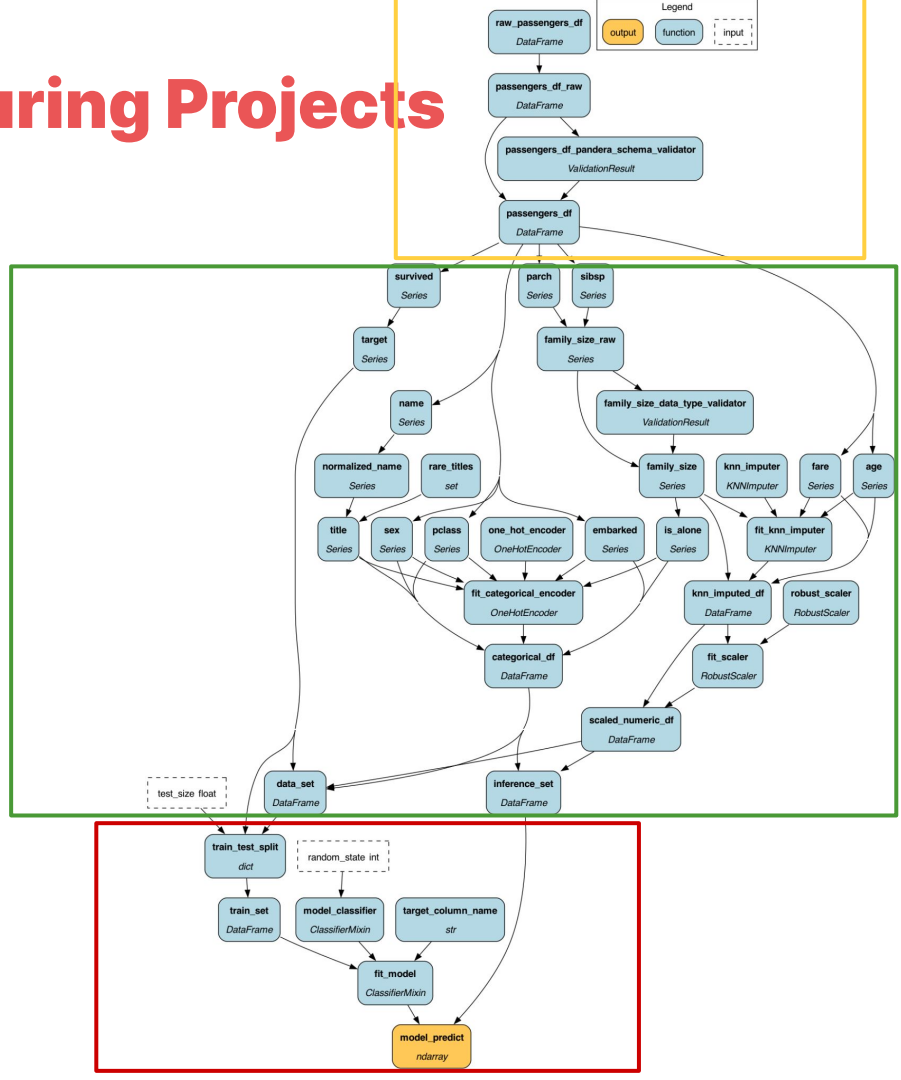
My Mental Model for Structuring Projects

Could be one module...



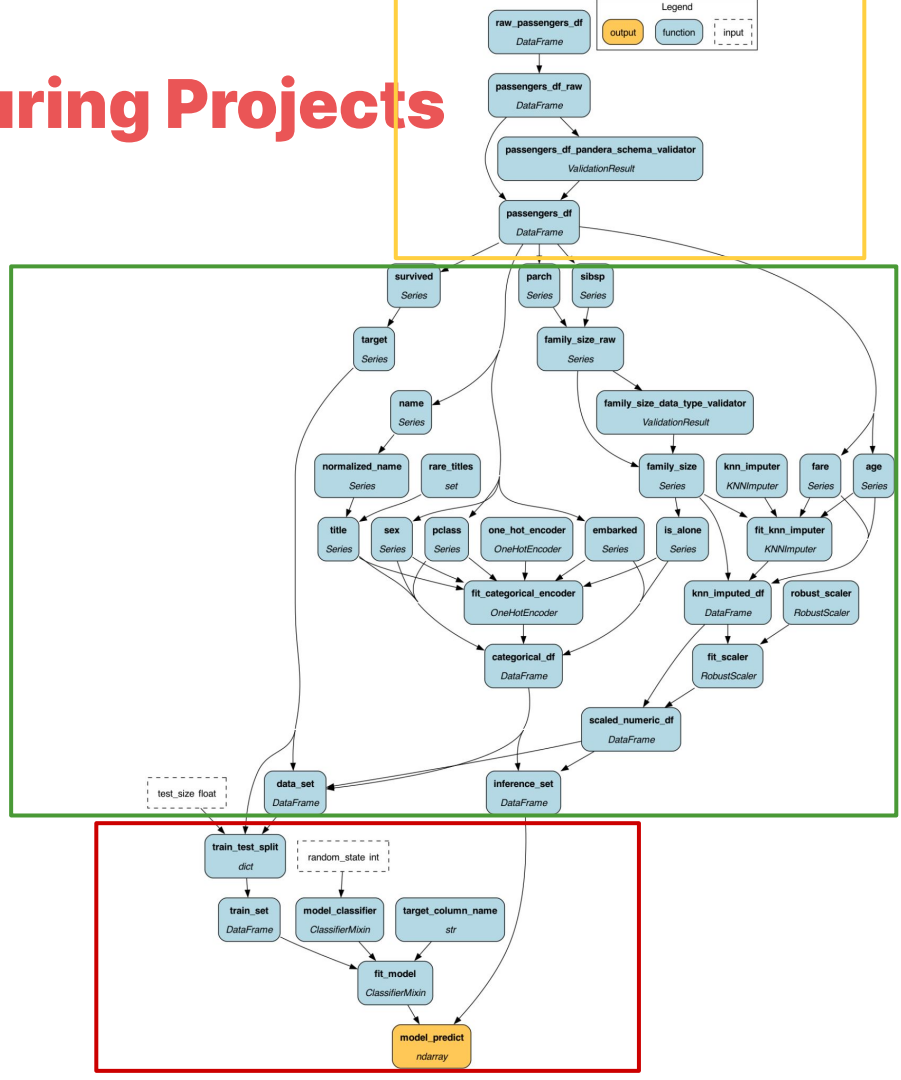
My Mental Model for Structuring Projects

Or three... or more...



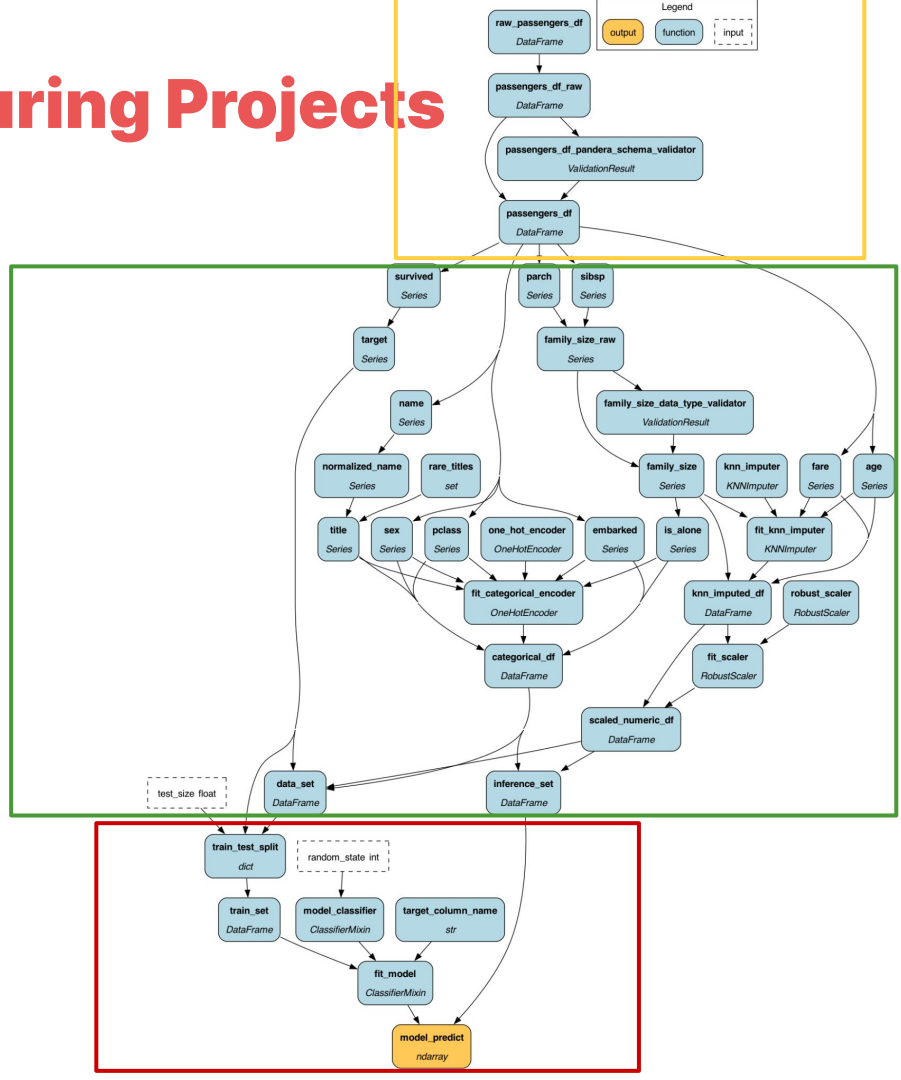
My Mental Model for Structuring Projects

```python
import data_loader, feature_transforms, model_pipeline

# DAG for training/inferring on titanic data
titanic_dag = (
  driver.Builder()
    .with_config(config),
    .with_modules(
      data_loader, feature_transforms, model_pipeline
    ).build()
)
# execute & get output
result = titanic_dag.execute(["model_predict"])
```



My Mental Model for Structuring Projects

```python
import data_loader, feature_transforms, model_pipeline

# DAG for training/inferring on titanic data
titanic_dag = (
  driver.Builder()
    .with_config(config),
    .with_modules(
      data_loader, feature_transforms, model_pipeline
    ).build()
)
```



Scaling this:

1. Group into thematic modules.
2. Name + type becomes interface.
3. Modularity comes from swapping modules and/or @config.when

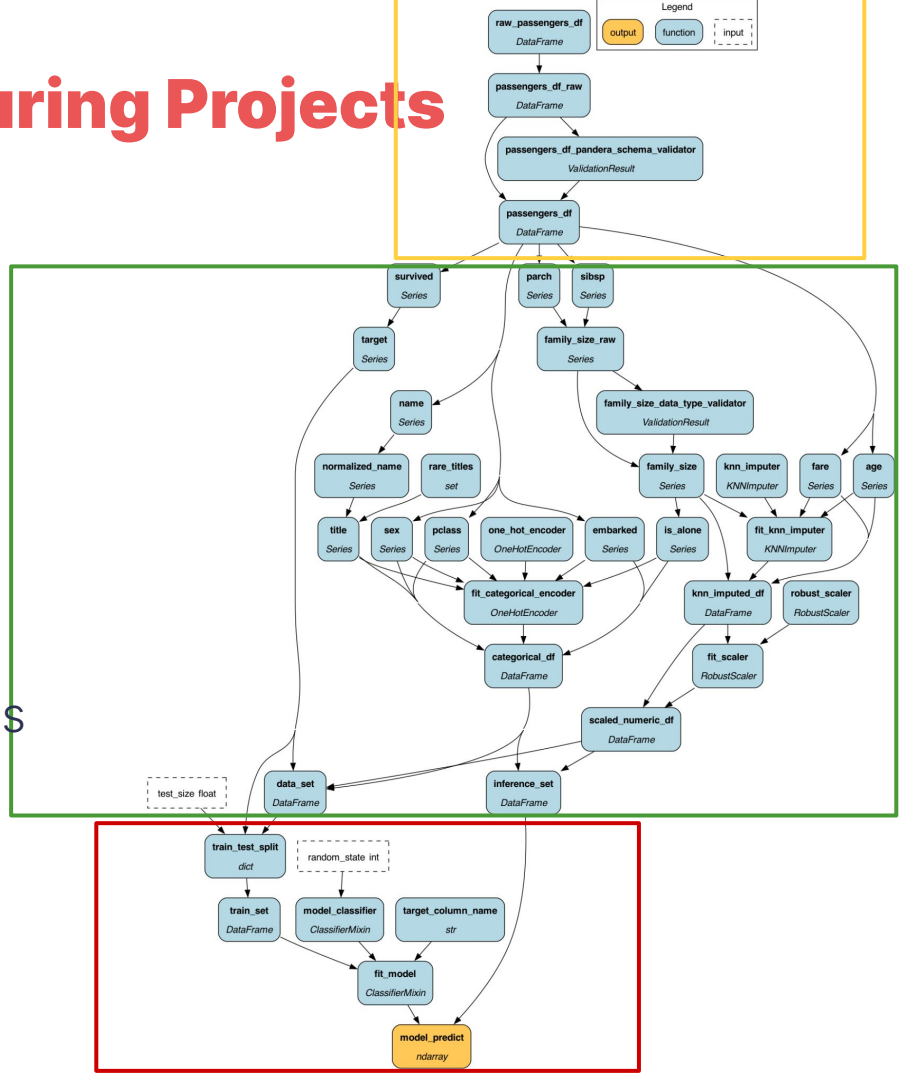
My Mental Model for Structuring Projects

Caveat:

1. Naming constraint.
    a. **Why?** Make it easy to go from output to code.

Depending on context:

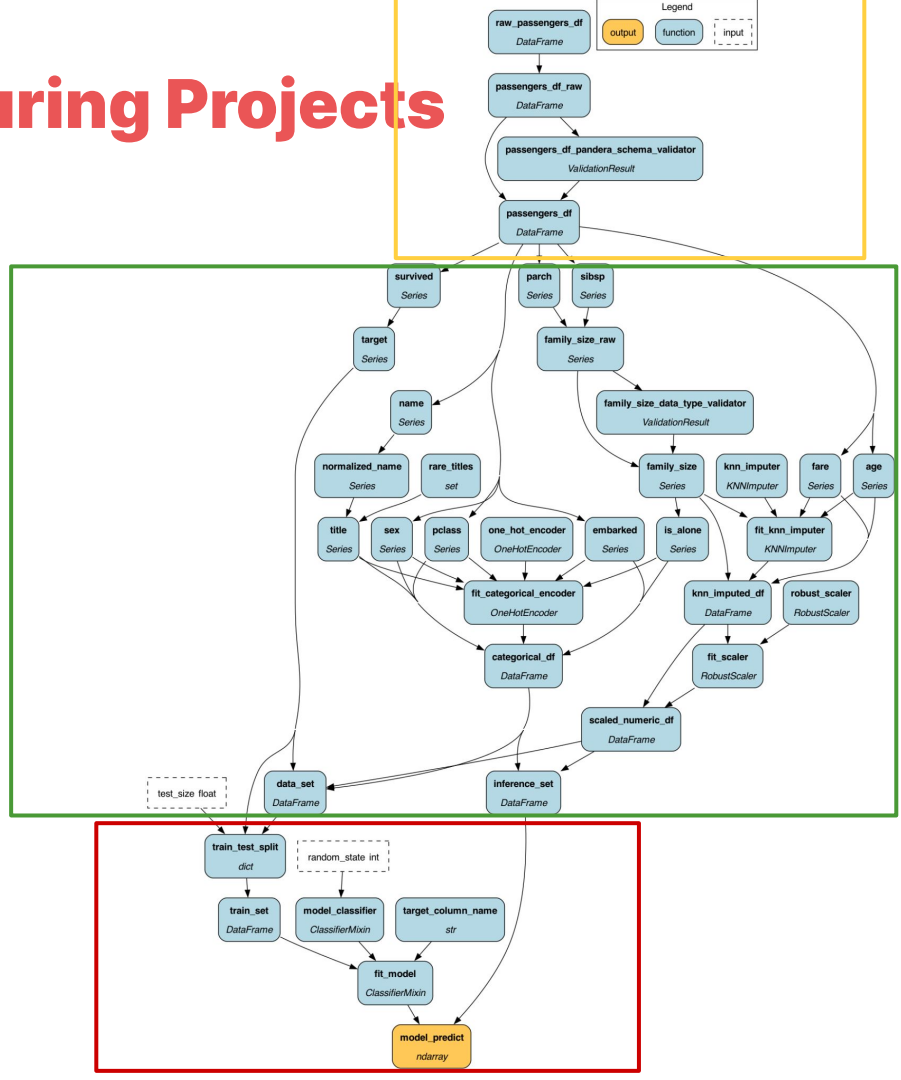
- Different modules for different contexts
- Mutation == new name
- Rename at data ingestion
- Decorators:
    - @subdag, @pipe, ...



My Mental Model for Structuring Projects

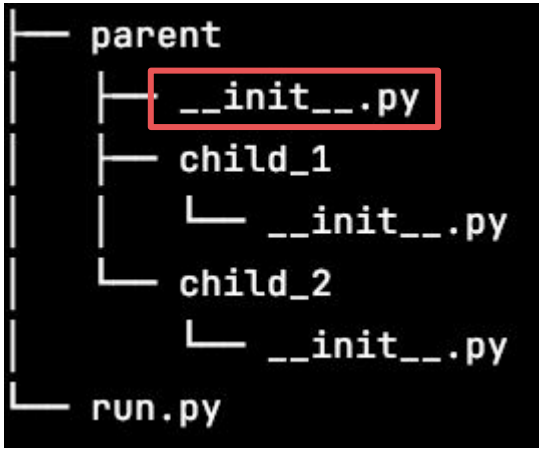
Summary:

- Modules == Lego bricks
- Lego bricks are standalone
  - But can be composed into something bigger
- Caveat:
  - Naming



Managing 4000 feature transforms at Stitch Fix

1. Single repository.
2. Naming convention for functions.
   a. E.g. `D_` is dummy variable.
3. Hierarchically group into subpackages.



Use it to aggregate

#parent/__init__.py

```python
from .child_1 import *
from .child_2 import *

def func_e(func_d: int, func_b: int) -> int:
    return func_d + func_b
```

```python
from hamilton import driver
import parent

if __name__ == '__main__':
    dr = driver.Builder().with_modules(parent).build()
    dr.display_all_functions("dag.png")
```

Managing 4000 feature transforms at Stitch Fix

1. Single repository.
2. Naming convention for functions.
    a. E.g. `D_` is dummy variable.
3. Hierarchically group into subpackages.
4. At driver construction time, pick right subset of packages
    a. Can use importlib to script:

```
from hamilton import driver

import importlib

if __name__ == '__main__':
    modules = [
        importlib.import_module(module_name)
        for module_name in [...]
    ]
    dr = driver.Builder().with_modules(*modules).build()
```

Managing 4000 feature transforms at Stitch Fix

1. Single repository.
2. Naming convention for functions.
    a. E.g. `D_` is dummy variable.
3. Hierarchically group into subpackages.
4. At driver construction time, pick right subset of packages
    a. Can use importlib to script
5. Can change overtime easily:
    a. Renaming is just find & replace (largely)
    b. Move functions around into other modules

Deep Dive:
Explaining **@subdag** in two minutes

DAGWORKS

@subdag in two minutes

```python
def feature_engineering(source_path: str) -> pd.DataFrame:
    '''You could recursively use Hamilton within itself.'''
    dr = driver.Driver({}, feature_modules)
    df = dr.execute(["feature_df"], inputs={"path": source_path})
    return df
```

← Can use Hamilton within Hamilton

← But lose “visibility”

```python
@subdag(
    feature_modules,
    inputs={"path": source("source_path")},
    config={}
)
def feature_engineering(feature_df: pd.DataFrame) -> pd.DataFrame:
    '''We've hidden the driver and requested feature_df from it.'''
    return feature_df
```

← Make it visible in a single DAG

← Subdag is “name spaced”.

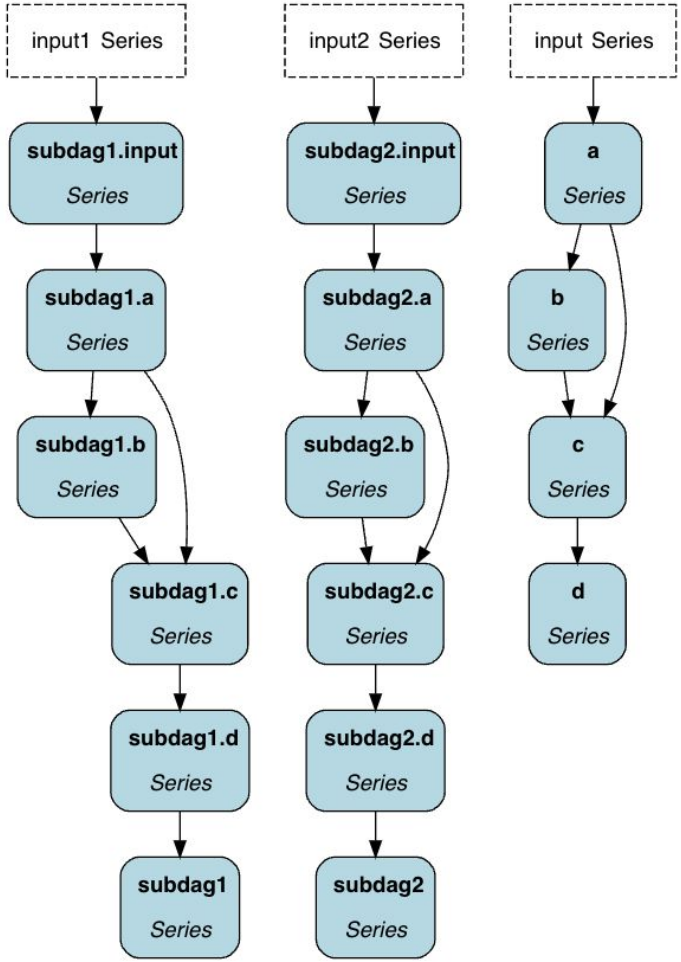
@subdag in two minutes

```python
# functions.py - declare and link your transformations as functions....
import pandas as pd
from hamilton.function_modifiers import subdag, source

def a(input: pd.Series) -> pd.Series:
    return input % 7

def b(a: pd.Series) -> pd.Series:
    return a * 2

def c(a: pd.Series, b: pd.Series) -> pd.Series:
    return a * 3 + b * 2

def d(c: pd.Series) -> pd.Series:
    return c ** 3

@subdag(
  a, b, c, d,
  inputs={"input": source("input1")},
  config={}
)
def subdag1(d: pd.Series) -> pd.Series:
    return d * 2

@subdag(
  a, b, c, d,
  inputs={"input": source("input2")},
  config={}
)
def subdag2(d: pd.Series) -> pd.Series:
    return d
```

```python
# And run them!
import functions
from hamilton import driver
dr = driver.Driver({}, functions)
result = dr.execute(
  ['subdag2', 'subdag1'],
  inputs={'input2': pd.Series([1, 2, 3, 4, 5]),
          'input1': pd.Series([1, 2, 3, 4, 5])}
)
print(result)
dr.display_all_functions(
   "graph.dot", orient="TB", show_legend=False)
```

▶ Run me!

```
subdag2  subdag1
```



Roadmap Thoughts

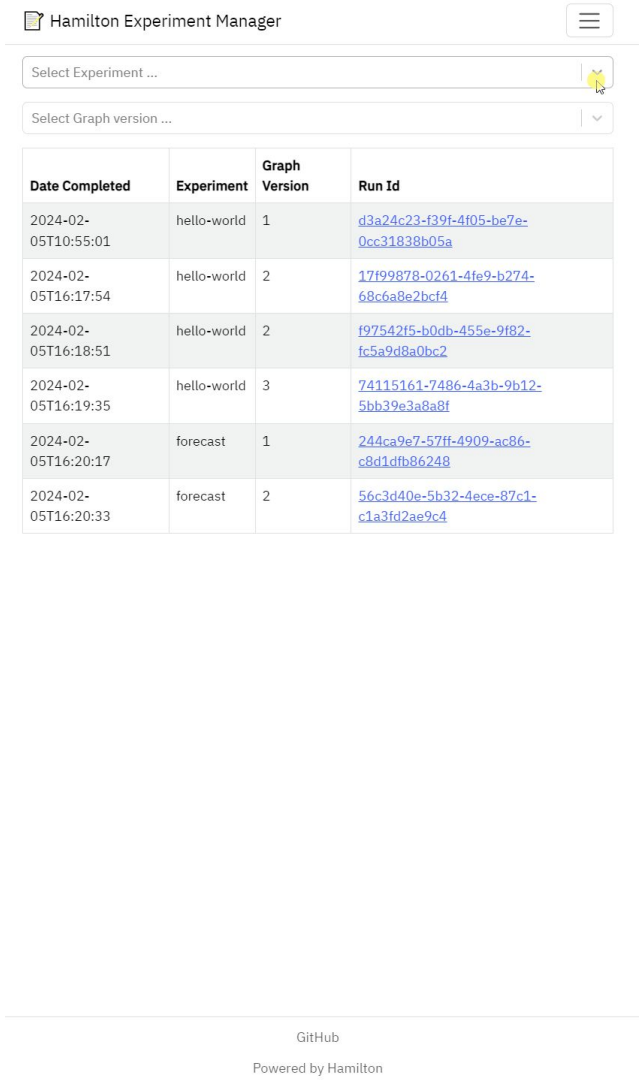
DAGWORKS

Roadmap: Some Recent Releases

- @tag now supports list of strings
- Custom styling for graphviz
- JSON export

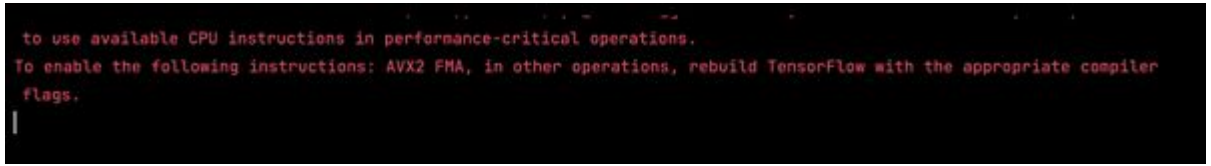
Roadmap: Some Recent Releases

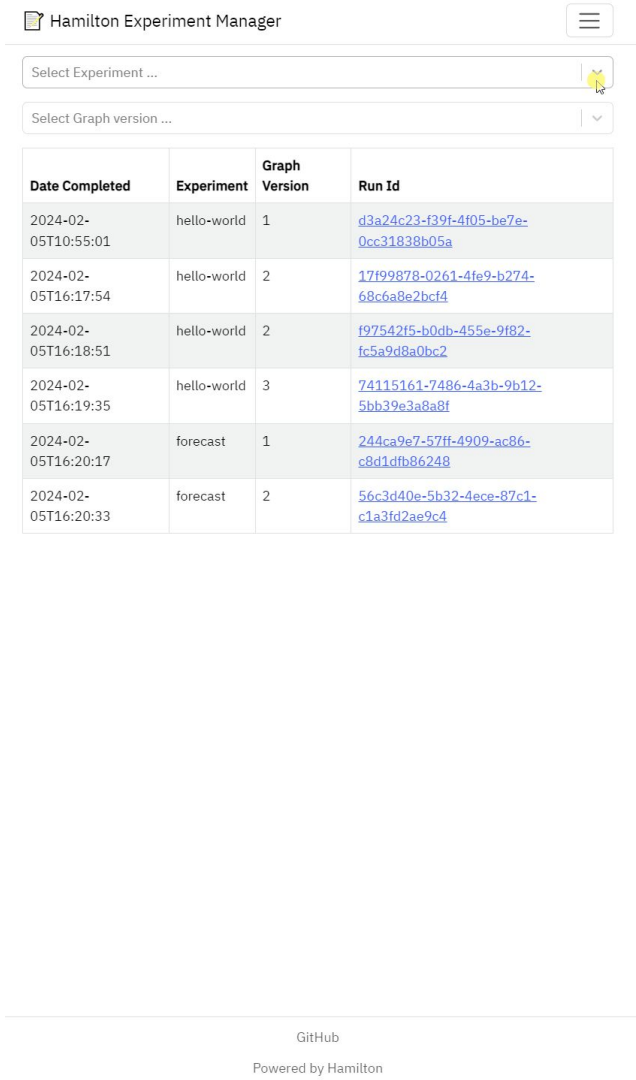
- @tag now supports list of strings
- Custom styling for graphviz
- JSON export
- Lightweight experiment tracker →



Roadmap: Some Recent Releases

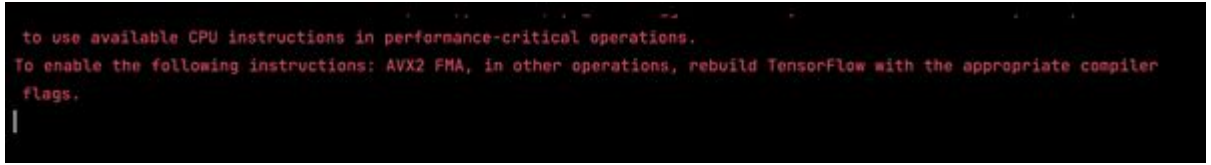
- @tag now supports list of strings
- Custom styling for graphviz
- JSON export
- Lightweight experiment tracker →
- https://hub.dagworks.io/ additions
- Documentation refactoring
- Lifecycle API & additions, e.g. TQDM, Datadog



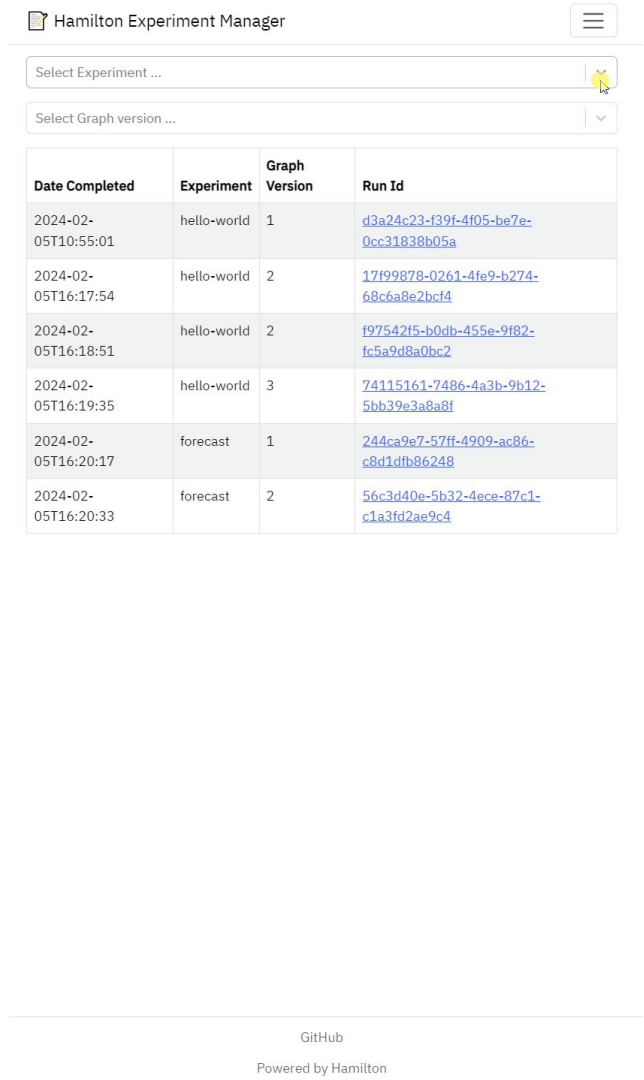


Roadmap: Some Recent Releases

- @tag now supports list of strings
- Custom styling for graphviz
- JSON export
- Lightweight experiment tracker →
- https://hub.dagworks.io/ additions
- Documentation refactoring
- Lifecycle API & additions, e.g. TQDM, Datadog



- Caching: `.with_adapters(h_diskcache.CacheHook())`
- Blogs - e.g. dev to prod ML pipelines.



Roadmap: DAGWorks callout

1. We're working with design partners to more tightly build out integrations that shapes open source.
    a. Ping us/put your hand up and we can do a 1-1 session if interested.

2. FYI: www.dagworks.io has a free tier we'd love feedback on:
    a. One line addition to Hamilton code.
    b. Get catalog, telemetry, observability, version capture...

Roadmap: What's on the horizon

- Documentation - code comparisons, integration guides...
- hub.dagworks.io - more examples

Needs prioritization - some ideas:

- **Your input/ideas here!**
  - **Do you have pains? Monitoring? What are you building/integrating with? Etc.**
  - **Testimonials page - we'd love some blog posts/quotes for social proof.**
- Ray/Dask: node grouping
- Snowpark integration
- Data quality integrations/pluggability
- Unit test generator
- Generating Airflow DAGs, AWS lambda integrations, etc.

📅 Next month - March 19th: Roel Bertens

“How to use Hamilton to share (feature) logic across multiple development teams.”

DAGWORKS

Open Mic.

DAGWORKS

FIN. Thanks for coming!