Hamilton

A Python Micro-Framework for Data / Feature Engineering

Stefan Krawczyk
Former Stitch Fix Model Lifecycle Platform Manager/Lead

Hamilton is Open Source Code

```
> pip install sf-hamilton
```

Get started in <15 minutes!

Documentation

https://hamilton-docs.gitbook.io/

Lots of examples:

https://github.com/stitchfix/hamilton/tree/main/examples

What is Hamilton?

What is Hamilton?

A declarative dataflow paradigm.

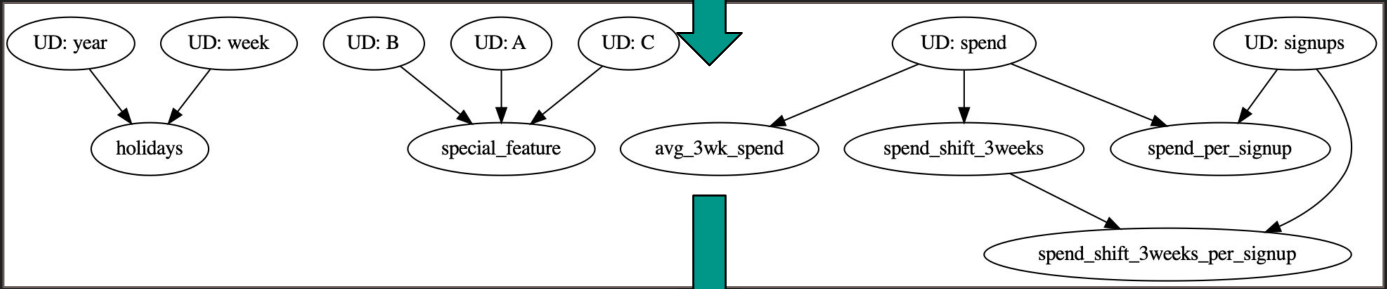
Hamilton:
Code → Directed Acyclic Graph → Object

Code:

```python
def holidays(year: pd.Series, week: pd.Series) -> pd.Series:
    """Some docs"""
    return some_library(year, week)
def avg_3wk_spend(spend: pd.Series) -> pd.Series:
    """Some docs"""
    return spend.rolling(3).mean()
def spend_per_signup(spend: pd.Series, signups: pd.Series) -> pd.Series:
    """Some docs"""
    return spend / signups
def spend_shift_3weeks(spend: pd.Series) -> pd.Series:
    """Some docs"""
    return spend.shift(3)
def spend_shift_3weeks_per_signup(spend_shift_3weeks: pd.Series, signups: pd.Series) -> pd.Series:
    """Some docs"""
    return spend_shift_3weeks / signups
```

User

DAG:



Hamilton

Object
(e.g. DataFrame):

Year	Week	Sign ups	...	Spend	Holiday
2015	2	57	...	123	0
2015	3	58	...	123	0
2015	4	59	...	123	1
2015	5	59	...	123	1
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
2021	16	1000	...	1234	0

User

Old way vs Hamilton way:

Instead of:

```
df['c'] = df['a'] + df['b']
df['d'] = transform(df['c'])
```

You declare:

```
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```

\+ some driver code (not shown)

Old way vs Hamilton way:

Instead of:

```
df['c'] = df['a'] + df['b']
df['d'] = transform(df['c'])
```

Outputs == Function Name

Inputs == Function Arguments

You declare:

```
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```

Full Hello World

Functions:

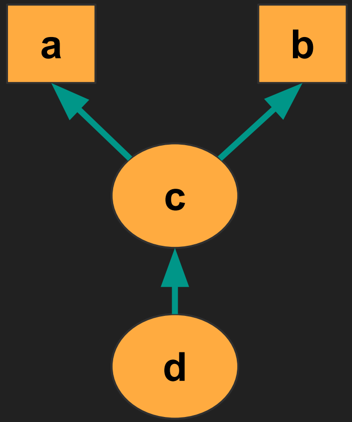
```python
# feature_logic.py
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```

“Driver” - this actually says what and when to execute:

```python
# run.py
from hamilton import driver
import feature_logic
dr = driver.Driver({'a': ..., 'b': ...}, feature_logic)
df_result = dr.execute(['c', 'd'])
print(df_result)
```

Hamilton TL;DR:

1. For each `=` statement, you write a function(s).
2. Functions declare a DAG.
3. Hamilton handles DAG execution.



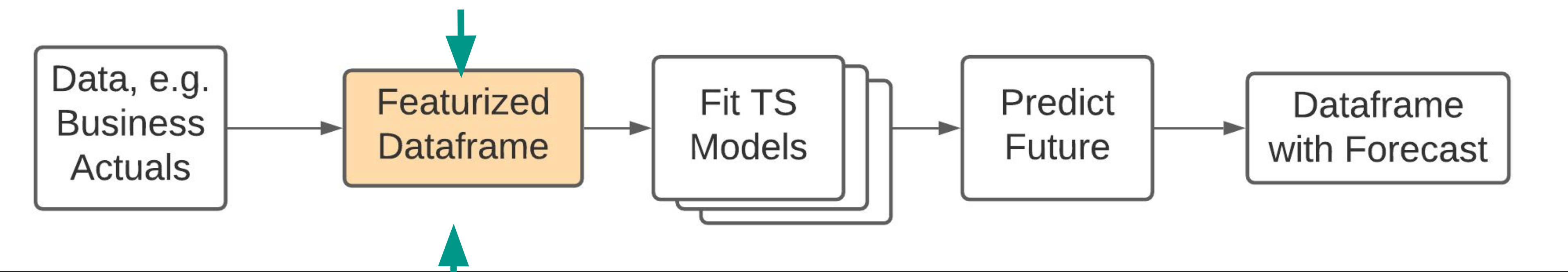

```python
# feature_logic.py
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Replaces c = a + b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Replaces d = transform(c)"""
    new_column = _transform_logic(c)
    return new_column
```

```python
# run.py
from hamilton import driver
import feature_logic
dr = driver.Driver({'a': ..., 'b': ...},
                   feature_logic)
df_result = dr.execute(['c', 'd'])
print(df_result)
```

Why was Hamilton created?

Backstory: Time-series Forecasting

Biggest problems here



What
Hamilton
helped solve!

Backstory: TS –> Dataframe creation

O(1000+) of columns

O(1000) weeks

Year	Week	Sign ups	...	Spend	Holiday	...
2015	2	57	...	123	0	...
2015	3	58	...	123	0	...
2015	4	59	...	123	1	...
2015	5	59	...	123	1	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
2021	16	1000	...	1234	0	...
20XX	X	XX	...	XXX	0	...
20XX	X	XX	...	XXX	1	...
20XX	X	XX	...	XXX	0	...

Columns are functions of other columns

Backstory: TS –> Dataframe creation

O(1000+) of columns

O(1000) weeks

		A		B		f(A,B)
Year	Week	Sign ups	...	Spend	Holiday	...
2015	2	57	...	123	0	...
2015	3	58	...	123	0	...
2015	4	59	...	123	1	...
2015	5	59	...	123	1	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
2021	16	1000	...	1234	0	...
20XX	X	XX	...	XXX	0	...
20XX	X	XX	...	XXX	1	...
20XX	X	XX	...	XXX	0	...

g(f(A,B), ...)

↓

h(g(f(A,B), ...), ...)

↓

etc 🔥

Backstory: TS -> DF -> 🍝 Code

```python
df = load_dates()  # load date ranges
df = load_actuals(df)  # load actuals, e.g. spend, signups
df['holidays'] = is_holiday(df['year'], df['week'])  # holidays
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()  # [redacted] spend
df['spend_per_signup'] = df['spend'] / df['si[redacted] signed up
df['spend_shift_3weeks'] = df.spend[redacted] shift spend because ...
df['spend_shift_3weeks [redacted]_shift_3weeks'] / df['signups']

def my_spec[redacted] pd.DataFrame) -> pd.Series:
    return ([redacted]] - df['B'] + df['C'])

df['special_feature'] = my_special_featu[redacted]
# ...
```

Now scale this code to 1000+ columns & a growing team 😬

Human scaling 😔:

- Testing / Unit testing 👎
- Documentation 👎
- Code Reviews 👎
- Onboarding 📈 👎
- Debugging 📈 👎

Underrated problem!

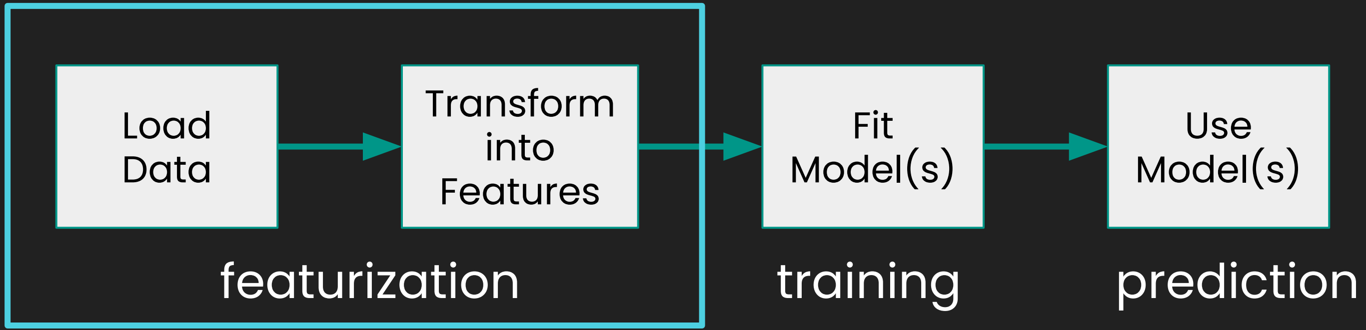
Hamilton @ Stitch Fix

Hamilton @ Stitch Fix

- Running in production for 2.5+ years
- Manages 4000+ feature definitions
- All feature definitions are:
  - Unit testable
  - Documentation friendly
  - Centrally curated, stored, and versioned in git.
- Data Science team ❤️s it:
  - Enabled a monthly task to be completed 4x faster
  - Easy to onboard new team members
  - Code reviews are simpler

Overview:
Feature Engineering
with Hamilton

Hamilton + Feature Engineering: Overview



- Can model this all in Hamilton (if you wanted to)
- We'll just focus on featurization
  - FYI: Hamilton works for any object type.
    - Here we'll assume pandas for simplicity.
  - E.g. use Hamilton within Airflow, Dagster, Prefect, Flyte, Metaflow, Kubeflow, etc.

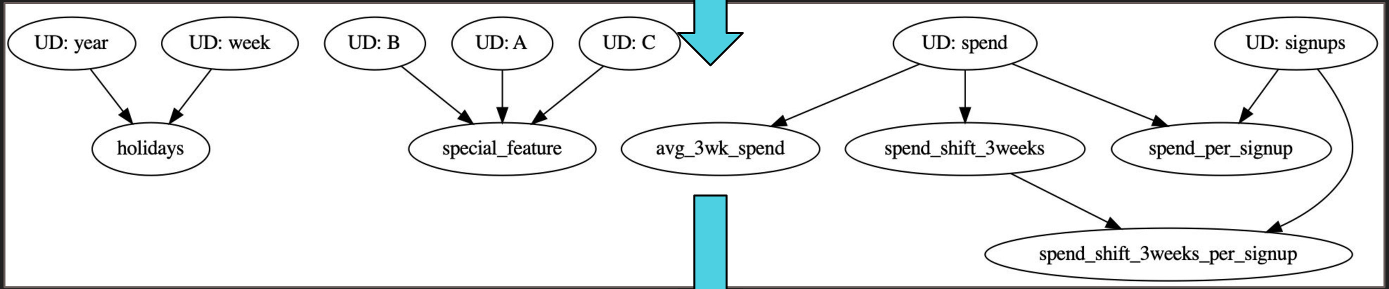
Modeling featurization

Data loading &
Feature code:

```python
def holidays(year: pd.Series, week: pd.Series) -> pd.Series:
   """Some docs"""
   return some_library(year, week)
def avg_3wk_spend(spend: pd.Series) -> pd.Series:
   """Some docs"""
   return spend.rolling(3).mean()
def spend_per_signup(spend: pd.Series, signups: pd.Series) -> pd.Series:
   """Some docs"""
   return spend / signups
def spend_shift_3weeks(spend: pd.Series) -> pd.Series:
   """Some docs"""
   return spend.shift(3)
def spend_shift_3weeks_per_signup(spend_shift_3weeks: pd.Series, signups: pd.Series) -> pd.Series:
   """Some docs"""
   return spend_shift_3weeks / signups
```

features.py

Via
Driver:



Feature
Dataframe:

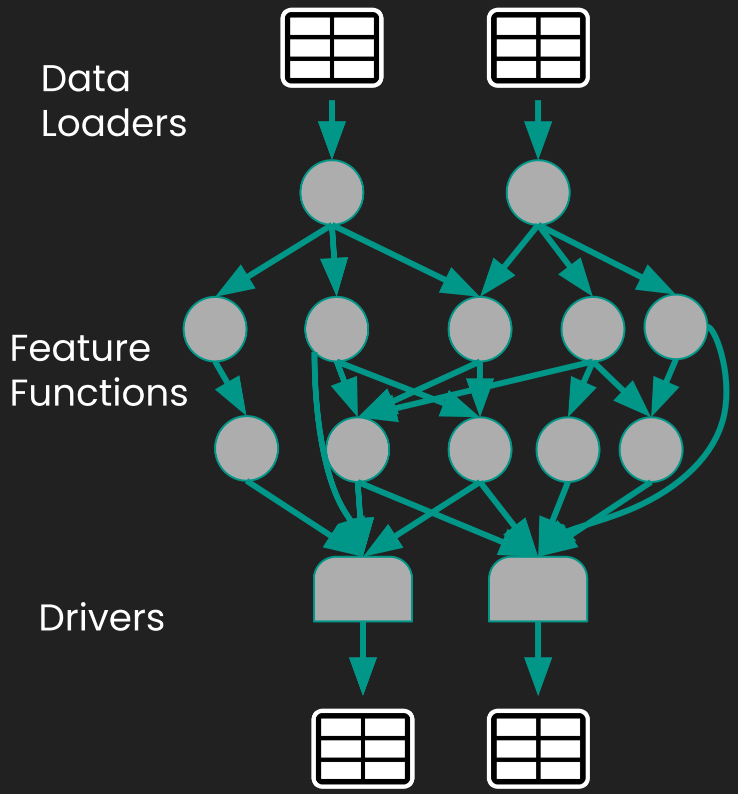
Year	Week	Sign ups	...	Spend	Holiday
2015	2	57	...	123	0
2015	3	58	...	123	0
2015	4	59	...	123	1
2015	5	59	...	123	1
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
2021	16	1000	...	1234	0

run.py

Modeling featurization

Code that needs to be written:

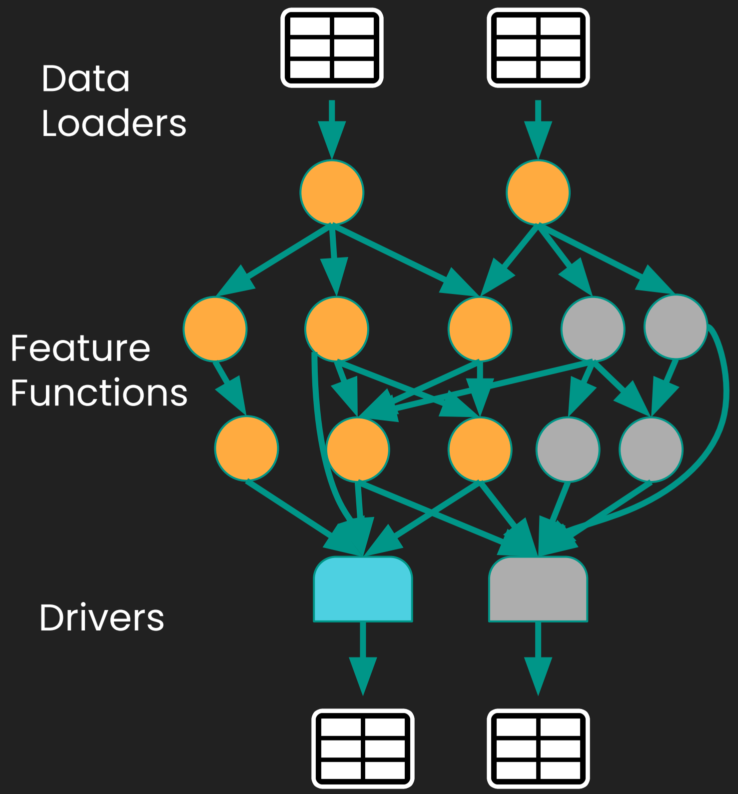
1. Functions to load data
    a. normalize/create common index to join on
2. Feature functions
    a. Optional: model functions.
3. Drivers materialize data
    a. DAG is walked for only what's needed.



Modeling featurization

Code that needs to be written:

1. Functions to load data
   a. normalize/create common index to join on
2. Feature functions
   a. Optional: model functions.
3. Drivers materialize data
   a. DAG is walked for only what's needed.



Problems with Feature Engineering

Problems with Feature Engineering

> Human/Team:

- Highly coupled code
- In ability to reuse/understand work
- Broken/unhealthy production pipelines

Hamilton helps here!

> Machines:

- Data is too big to fit in memory
- Cannot easily parallelize computation

Hamilton has integrations here!

Scaling Humans/Teams

How Hamilton helps with Human/Team Scaling

Highly coupled code	Decouples “functions” from use (driver code).

How Hamilton helps with Human/Team Scaling

Highly coupled code	Decouples “functions” from use (driver code).
In ability to reuse/understand work	Functions are curated into modules. Everything is unit testable. Documentation is natural. Forced to align on naming.

How Hamilton helps with Human/Team Scaling

Highly coupled code	Decouples "functions" from use (driver code).
In ability to reuse/understand work	Functions are curated into modules. Everything is unit testable. Documentation is natural. Forced to align on naming.
Broken/unhealthy production pipelines	Debugging is straightforward. Easy to version features via git/packaging. Runtime data quality checks.

Scaling Humans/Teams

Hamilton Functions:

```python
# client_features.py
@tag(owner='Data-Science', pii='False')
@check_output(data_type=np.float64, range=(-5.0, 5.0), allow_nans=False)
def height_zero_mean_unit_variance(height_zero_mean: pd.Series,
                                   height_std_dev: pd.Series) -> pd.Series:
    """Zero mean unit variance value of height"""
    return height_zero_mean / height_std_dev
```

Hamilton Features:

- Unit testing ✅ always possible
- Documentation ✅ tags, visualization, function doc
- Modularity/reuse ✅ module curation & drivers
- Central feature definition store ✅ naming, curation, versioning
- Data quality ✅ runtime checks

Scaling Humans/Teams

Code base implications:

1. Functions are always in modules
2. Driver script, i.e execution script, is decoupled from functions.

> Code reuse from day one!
> Low maintenance to support many driver scripts

Scaling Compute/Data

Scaling Compute/Data with Hamilton

Hamilton has the following integrations out of the box:

- Ray
  - Single process -> Multiprocessing -> Cluster
- Dask
  - Single process -> Multiprocessing -> Cluster
- Pandas on Spark
  - Uses enables using Pandas Spark API with your Pandas code easily
- Switching to run on Ray/Dask/Pandas on Spark requires:

  › **Only changing driver.py code***

  › Pandas on Spark also needs changing how data is loaded.

Hamilton + Ray/Dask/Spark: Driver only change

```
# run.py
from hamilton import driver
import data_loaders
import date_features
import spend_features
config = {...} # config, e.g. data_location
dr = driver.Driver(config,
                   data_loaders,
                   date_features,
                   spend_features)
features_wanted = [...] # choose subset wanted
feature_df = dr.execute(features_wanted)
save(feature_df, 'prod.features')
```

Hamilton + Ray: Driver only change

```
# run_on_ray.py
…
from hamilton import base, driver
from hamilton.experimental import h_ray
…
ray.init()
config = {...}
rga = h_ray.RayGraphAdapter(
    result_builder=base.PandasDataFrameResult())
dr = driver.Driver(config,
                   data_loaders, date_features, spend_features,
                   adapter=rga)
features_wanted = [...]  # choose subset wanted
feature_df = dr.execute(features_wanted,
                        inputs=date_features)
save(feature_df, 'prod.features')
ray.shutdown()
```

Hamilton + Dask: Driver only change

```python
# run_on_dask.py
…
from hamilton import base, driver
from hamilton.experimental import h_dask
…
client = Client(Cluster(...)) # dask cluster/client
config = {...}
dga = h_dask.DaskGraphAdapter(client,
    result_builder=base.PandasDataFrameResult())
dr = driver.Driver(config,
                   data_loaders, date_features, spend_features,
                   adapter=dga)
features_wanted = [...]  # choose subset wanted
feature_df = dr.execute(features_wanted,
                        inputs=date_features)
save(feature_df, 'prod.features')
client.shutdown()
```

Hamilton + Spark: Driver change + loader

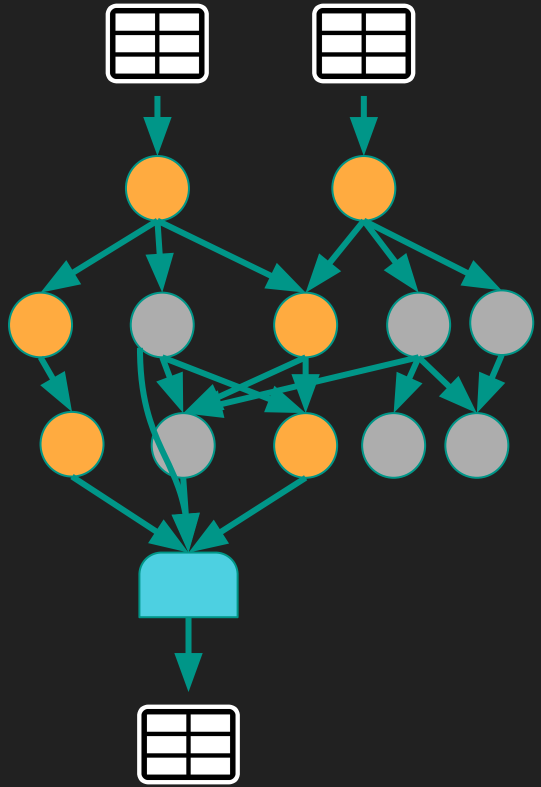
```
# run_on_pandas_on_spark.py
…
import pyspark.pandas as ps
from hamilton import base, driver
from hamilton.experimental import h_spark
…
spark = SparkSession.builder.getOrCreate()
ps.set_option(...)
config = {...}
skga = h_dask.SparkKoalasGraphAdapter(spark, spine='COLUMN_NAME',
    result_builder=base.PandasDataFrameResult())
dr = driver.Driver(config,
                   spark_data_loaders, date_features,spend_features,
                   adapter=skga)
features_wanted = [...]  # choose subset wanted
feature_df = dr.execute(features_wanted,
                        inputs=date_features)
save(feature_df, 'prod.features')
spark.stop()
```

Hamilton + Ray/Dask: How does it work?

```python
# FUNCTIONS
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```

```python
# DRIVER
from hamilton import base, driver
from hamilton.experimental import h_ray
…
ray.init()
config = {...}
rga = h_ray.RayGraphAdapter(
    result_builder=base.PandasDataFrameResult())
dr = driver.Driver(config,
                   data_loaders,
                   date_features,
                   spend_features,
                   adapter=rga)
features_wanted = [...]  # choose subset wanted
feature_df = dr.execute(features_wanted,
                        inputs=date_features)
save(feature_df, 'prod.features')
ray.shutdown()
```

DAG

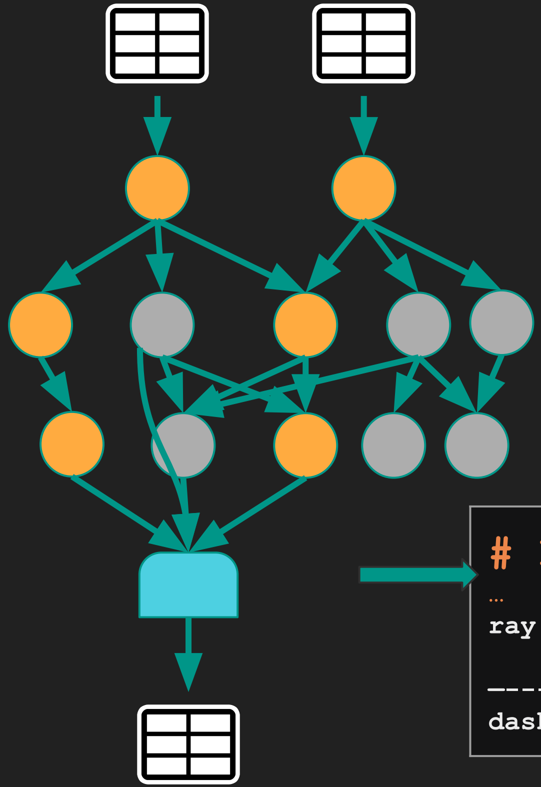


Hamilton + Ray/Dask: How does it work?

```
# FUNCTIONS
def c(a: pd.Series, b: pd.Series) -> pd.Series:
   """Sums a with b"""
   return a + b

def d(c: pd.Series) -> pd.Series:
   """Transforms C to ..."""
   new_column = _transform_logic(c)
   return new_column
```

```
# DRIVER
from hamilton import base, driver
from hamilton.experimental import h_ray
…
ray.init()
config = {...}
rga = h_ray.RayGraphAdapter(
    result_builder=base.PandasDataFrameResult())
dr = driver.Driver(config,
                   data_loaders,
                   date_features,
                   spend_features,
                   adapter=rga)
features_wanted = [...]  # choose subset wanted
feature_df = dr.execute(features_wanted,
                        inputs=date_features)
save(feature_df, 'prod.features')
ray.shutdown()
```

DAG



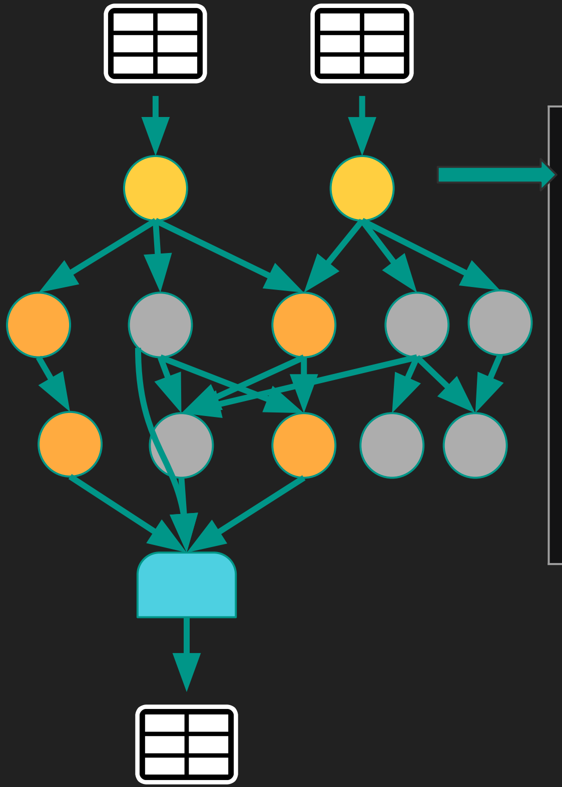
```
# Delegate to Ray/Dask
…
ray.remote(
    node.callable).remote(**kwargs)
--------------------
dask.delayed(node.callable)(**kwargs)
```

Hamilton + Spark: How does it work?

```python
# FUNCTIONS
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```

```python
# DRIVER
from hamilton import base, driver
from hamilton.experimental import h_ray
…
ray.init()
config = {...}
rga = h_ray.RayGraphAdapter(
    result_builder=base.PandasDataFrameResult())
dr = driver.Driver(config,
                   data_loaders,
                   date_features,
                   spend_features,
                   adapter=rga)
features_wanted = [...]  # choose subset wanted
feature_df = dr.execute(features_wanted,
                        inputs=date_features)
save(feature_df, 'prod.features')
ray.shutdown()
```

DAG



```
# With Spark
…
Change these to load
Spark "Pandas"
equivalent object
instead.

Spark will take care
of the rest.
```

Hamilton + Ray/Dask/Pandas on Spark: Caveats

Things to think about:

1. Serialization:
    a. Hamilton defaults to serialization methodology of these frameworks.
2. Memory:
    a. Defaults should work. But fine tuning memory on a “function” basis is not exposed.
3. Python dependencies:
    a. You need to manage them.
4. Looking to graduate these APIs from experimental status

>> Looking for contributions here to extend support in Hamilton! <<

Summary

Summary: Hamilton for feature/data engineering

- Hamilton is a declarative paradigm to describe data/feature transformations
  - Embeddable anywhere that runs python.
- It grew out of a need to tame a feature code base
  - it'll make yours better too!
- The Hamilton paradigm scales humans/teams through software engineering best practices.
- **Hamilton + Ray/Dask/Pandas on Spark** enables one to:

scale humans/teams **and** *scale data/compute*.

Give Hamilton a Try!
We'd love your Feedback

```
> pip install sf-hamilton
```

⭐ on [github](https://github.com/stitchfix/hamilton) (https://github.com/stitchfix/hamilton)

☑️ create & vote on issues on github

📣 join us on on [Slack](https://join.slack.com/t/hamilton-opensource/shared_invite/zt-1bjs72asx-wcUTgH7q7QX1igiQ5bbdcg)
(https://join.slack.com/t/hamilton-opensource/shared_invite/zt-1bjs72asx-wcUTgH7q7QX1igiQ5bbdcg)

Thank you.

Questions?

https://twitter.com/stefkrawczyk

https://www.linkedin.com/in/skrawczyk/

https://github.com/stitchfix/hamilton