Hamilton: an open source, declarative, micro-framework for clean & robust feature transform code in Python

Stefan Krawczyk, Ex-Mgr. Model Lifecycle @ Stitch Fix

FEATURE STORE SUMMIT 2022

Organized by HOPSWORKS

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 → Dataflow → Object

Hamilton:

Code → Dataflow → 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

Hamilton:

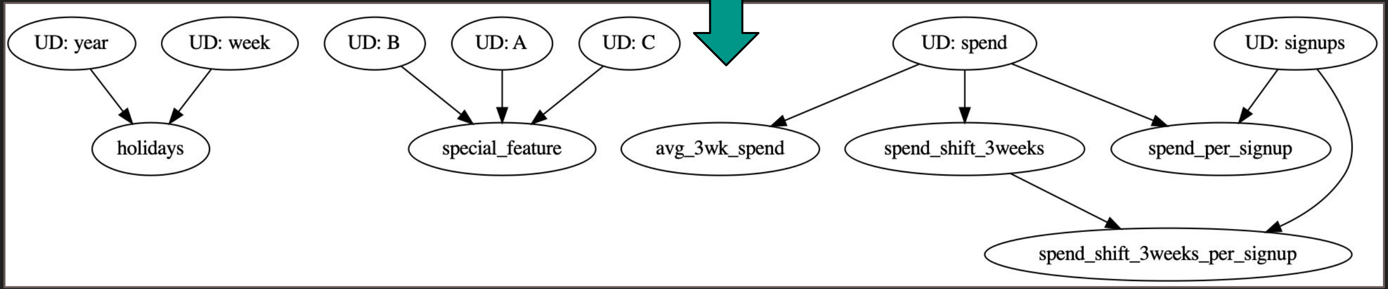
Code → Dataflow → Object

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

Hamilton:

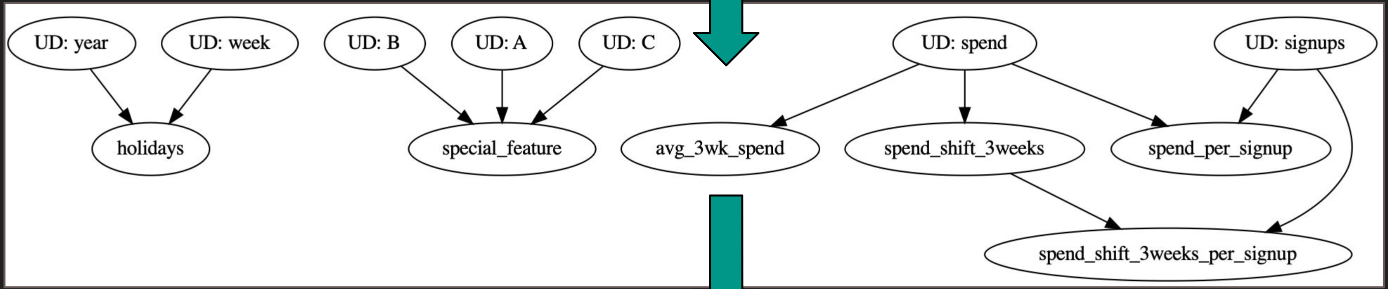
Code → Dataflow → Object

Code:

```
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(s)
(e.g. Dataframe,
ML Model):

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

Hamilton:

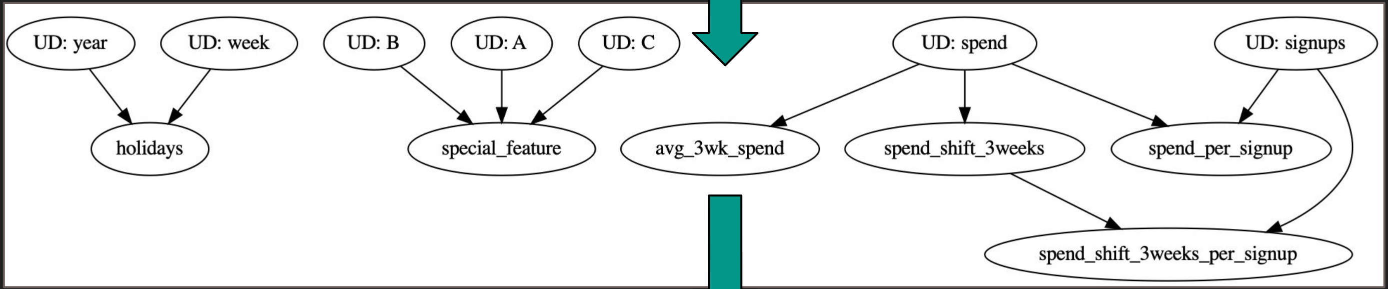
Code → Dataflow → Object

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
```

Python Modules

DAG:



Object(s) (e.g. Dataframe, ML Model):

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

"Driver" Code

Hamilton Paradigm: declaring a dataflow

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)

Hamilton Paradigm: declaring a dataflow

Instead of:

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

Outputs == Function Name

Inputs == Function Arguments

You declare:

```python
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)
```

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:

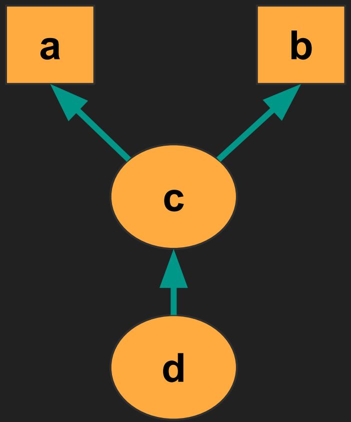
```python
# run.py
from hamilton import driver
import feature_logic
dr = driver.Driver({'a': ..., 'b': ...}, feature_logic)
df_result = dr.execute(['c', 'd'])
print(df_result)
```

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:

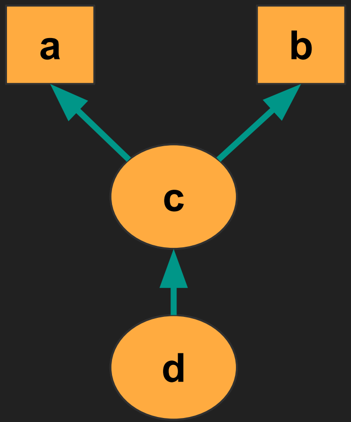
```python
# run.py
from hamilton import driver
import feature_logic
dr = driver.Driver({'a': ..., 'b': ...}, feature_logic)
df_result = dr.execute(['c', 'd'])
print(df_result)
```

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
```



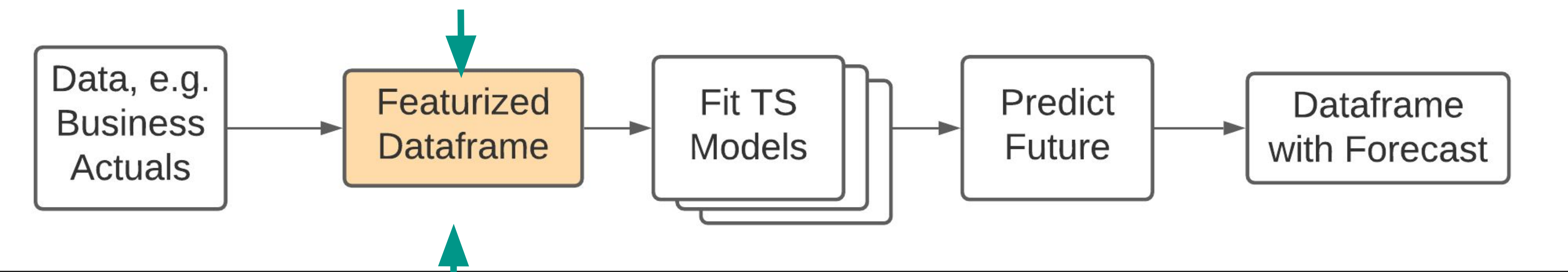
"Driver" - this actually says what and when to execute:

```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)
```

Why was Hamilton created?

Backstory: Time-series Forecasting w/FED

Biggest problems here



What Hamilton helped solve!

Backstory: TS -> Dataframe creation

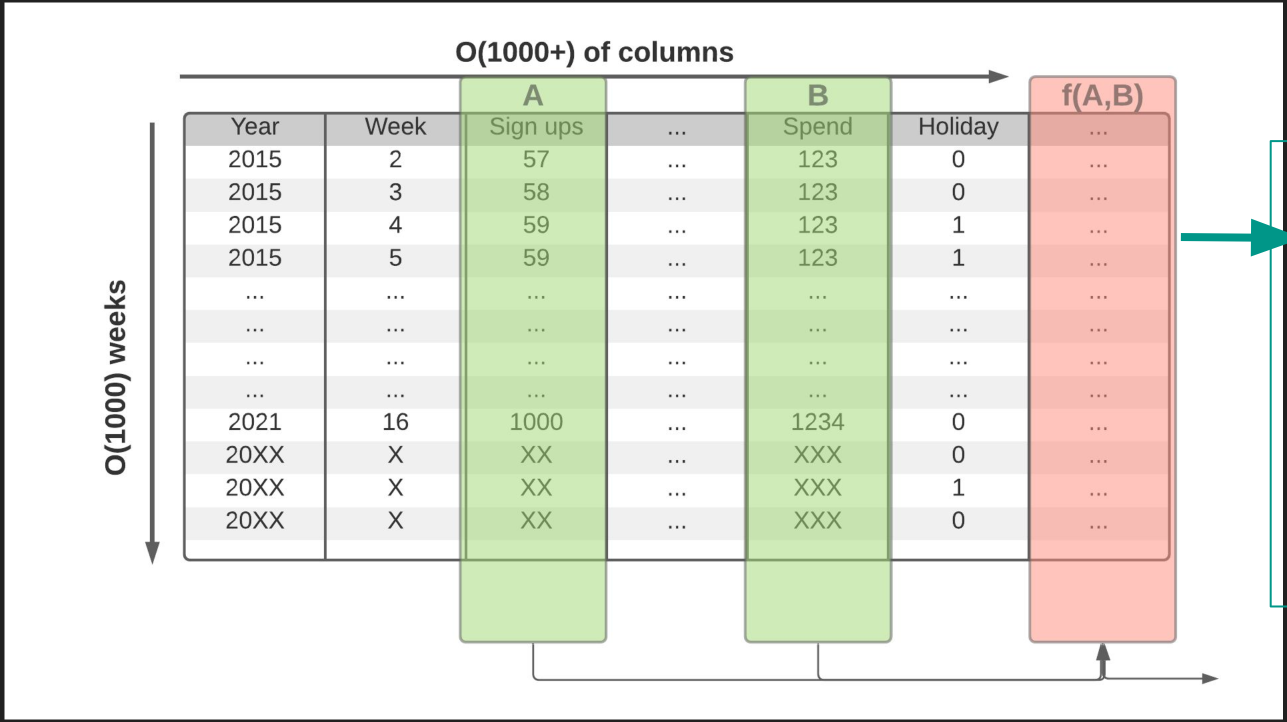
O(1000+) of columns

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



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: Example transform code

```
df = loader.load_actuals(dates) # e.g. spend, signups
```

Backstory: Example transform code

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
   df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
   df['holidays'] = is_holiday(df['year'], df['week'])
```

Backstory: Example transform code

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
    df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
    df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
```

Backstory: Example transform code

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
   df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
   df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
```

Backstory: Example transform code

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
   df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
   df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
save_df(df, "some_location")
```

Backstory: Example transform code

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
   df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
   df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
save_df(df, "some_location")
```

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

NOT CLEAN & ROBUST!

Problem: unit testing & integration testing 👎

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
    df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
    df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
save_df(df, "some_location")
```

Now scale this code to 1000+ columns & a growing team
😬
NOT CLEAN & ROBUST!

Problem: code readability & documentation 🧐

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
    df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
    df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
save_df(df, "some_location")
```

?

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

NOT CLEAN & ROBUST!

Problem: difficulty in tracing lineage 🤯

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
    df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
    df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
→df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
→df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
save_df(df, "some_location")
```

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

😬

NOT CLEAN & ROBUST!

Problem: code reuse and duplication

```
df = loader.load_actuals(dates) # e.g. spend, signups
if config['region'] == 'UK':
    df['holidays'] = is_uk_holiday(df['year'], df[' week'])
else:
    df['holidays'] = is_holiday(df['year'], df['week'])
df['avg_3wk_spend'] = df['spend'].rolling(3).mean()
df['acquisition_cost'] = df['spend'] / df['signups']
df['spend_shift_3weeks'] = df['spend'].shift(3)
df['special_feature1'] = compute_bespoke_feature(df)
df['spend_b'] = multiply_columns(df['acquisition_cost'], df['B'])
save_df(df, "some_location")
```

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

NOT CLEAN & ROBUST!

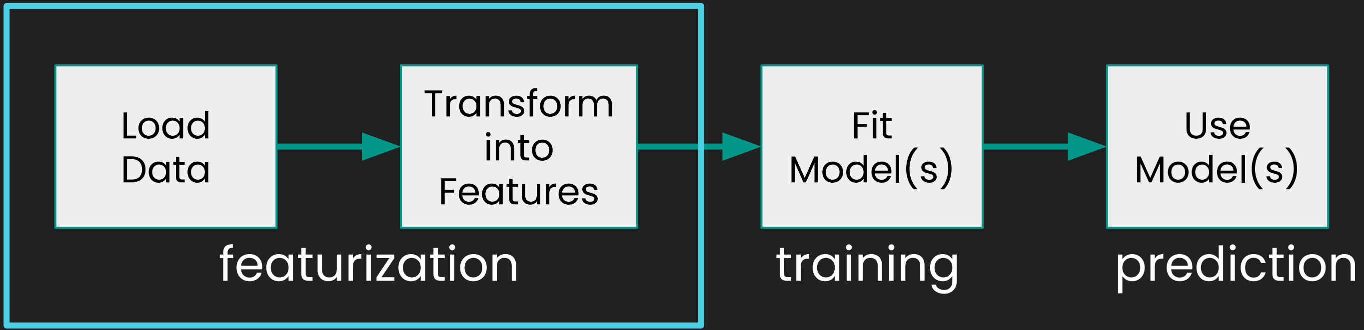
Hamilton @ Stitch Fix

Hamilton @ Stitch Fix

- Running in production for 2.5+ years
- FED team manages 4000+ feature definitions
  - All feature definitions are:
    - Unit testable
    - Documentation friendly
    - Centrally curated, stored, and versioned in git.
- Data Science teams ❤️ it:
  - Best adoption from active time-series forecasting teams
    - Most willing to pay migration cost.
  - Enabled a monthly feature update & model fitting task to be completed 4x faster

Overview:
Feature/data Engineering
with Hamilton

Hamilton + Feature/data 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.
    - **Batch**: use within an orchestration system (e.g. Airflow), Jupyter notebook, in front of Feast, etc.
    - **Online**: embed within python streaming / python web serivce

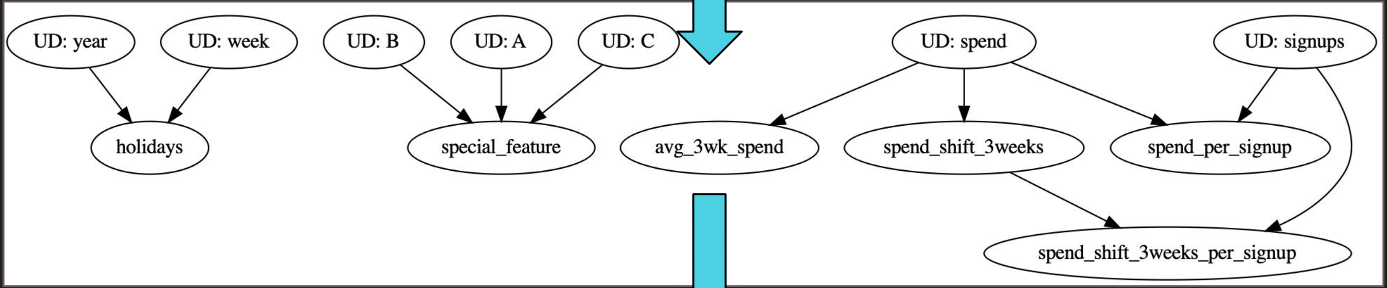
Modeling featurization

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:

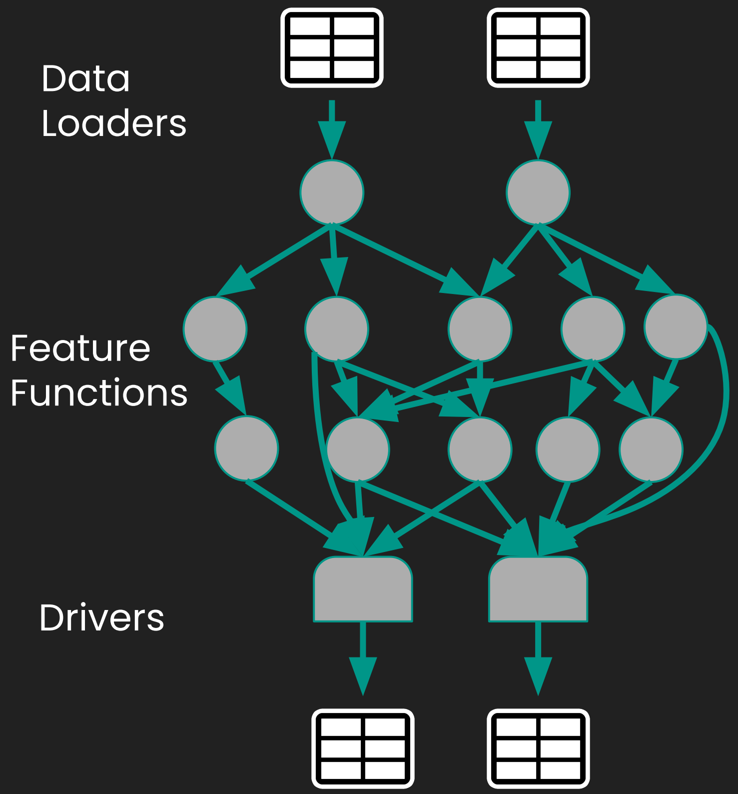
Year	Week	Sign ups	...	Spend	Holiday
2015	2	57	...	123	0
2015	3	58	...	123	0
2015	4	59	...	123	1
2015	5	59	...	123	1
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
2021	16	1000	...	1234	0

run.py

Modeling featurization

Code that needs to be written:

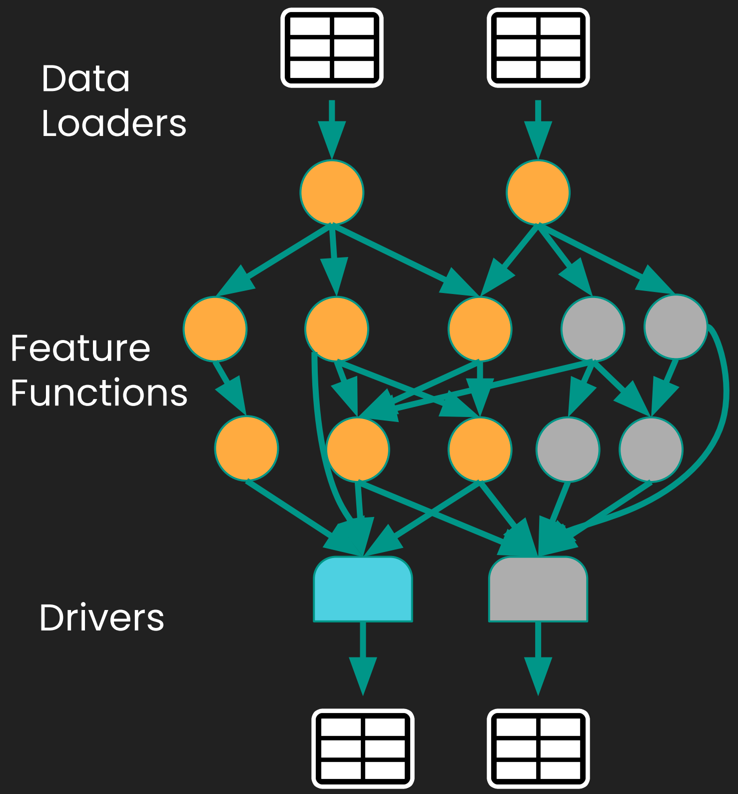
1. Functions to load data
   a. normalize/create common index to join on
2. Feature functions
   a. Optional: model functions.
3. Drivers materialize data
   a. DAG is walked for only what's needed.



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.



General Problems with Feature Engineering

General 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, e.g. Ray & Dask!

General Problems with Feature Engineering

Focus for rest of talk

> 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, e.g. Ray & Dask!

Making Feature Engineering Clean & Robust

Clean & Robust Feature Engineering

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

Clean & Robust Feature Engineering

Issue	Hamilton
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.

Clean & Robust Feature Engineering

Issue	Hamilton
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.

Clean & Robust Feature Engineering

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

Clean & Robust Feature Engineering

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.
> Code base ends up well structured.

Summary

Summary: Hamilton -
Clean & Robust Feature 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!
- **Hamilton** paradigm enables one to:

    Write clean & robust feature transforms
    via software engineering best practices
    without you thinking about it!

Anyone who is doing feature engineering in python should know about it!

Give Hamilton a Try!
We'd love your Feedback

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

⭐ on github (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)

Thank you.

Questions?

https://twitter.com/stefkrawczyk

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

https://github.com/stitchfix/hamilton