Hamilton

a modular open source declarative paradigm for high level modeling of dataflows

Stefan Krawczyk, Elijah ben Izzy
@ Stitch Fix

CDMS Workshop VLDB 2022

Introduction

Context:

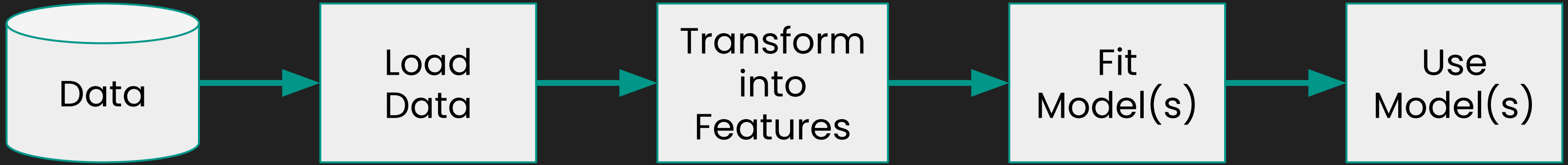
- Stitch Fix is a business where “machine learning” is core to the product
- Stitch Fix has 100+ Data Scientists (DS)
  - No hand-off; DS responsible for productionization*
  - Platform team focuses on 🛠️🧱:
    - Capabilities
    - ⬆️ Iteration speed & ⬇️ maintenance

*https://multithreaded.stitchfix.com/blog/2019/03/11/FullStackDS-Generalists/

Introduction

Context:

- DS own ETLs on top of the data lakehouse:



Problems:

- poor software eng. practices
- coupling of logic
- user migrations required due to changes in the underlying platform

Introduction

Connection with CDMS Workshop:

(1) End user experience with hooks for a platform team.

(2) Modularity by decoupling materialization from dataflow specification.

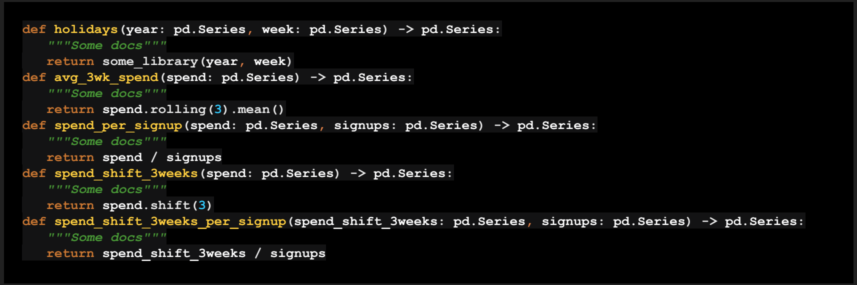
Hamilton:

Code → Dataflow → Object

Hamilton:

Code → Dataflow → Object

Code:



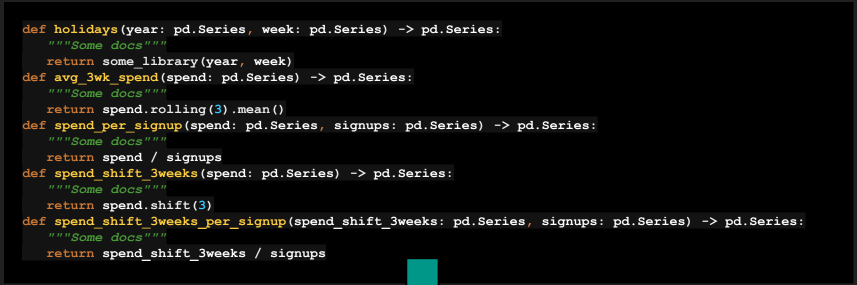
```python
def holidays(year: pd.Series, week: pd.Series) -> pd.Series:
    """Some docs"""
    return some_library(year, week)
def avg_3wk_spend(spend: pd.Series) -> pd.Series:
    """Some docs"""
    return spend.rolling(3).mean()
def spend_per_signup(spend: pd.Series, signups: pd.Series) -> pd.Series: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

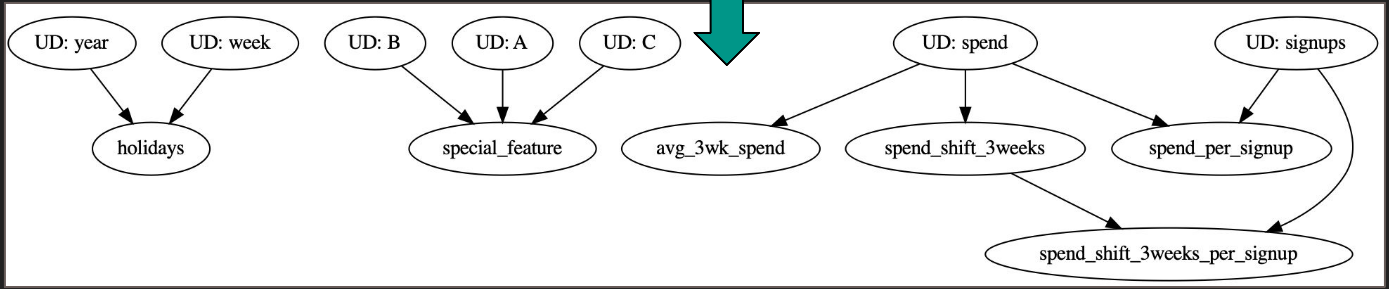
Code:



```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:

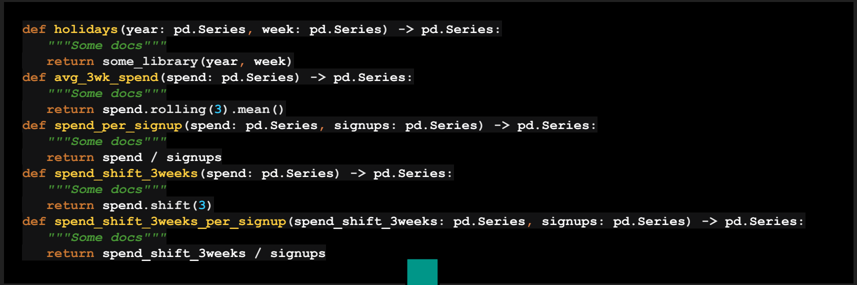


Hamilton

Hamilton:

Code → Dataflow → Object

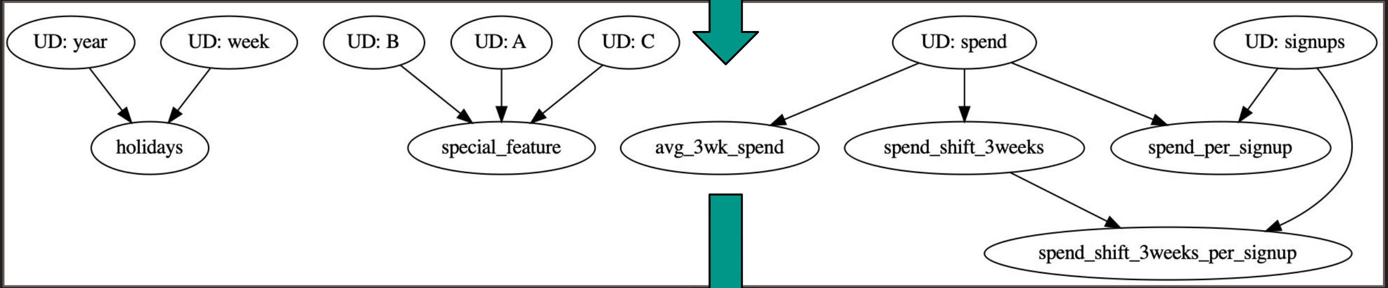
Code:



```
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
```

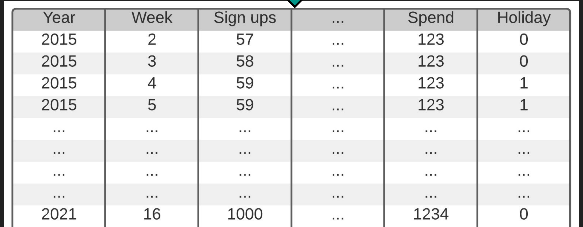
User

DAG:



Hamilton

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



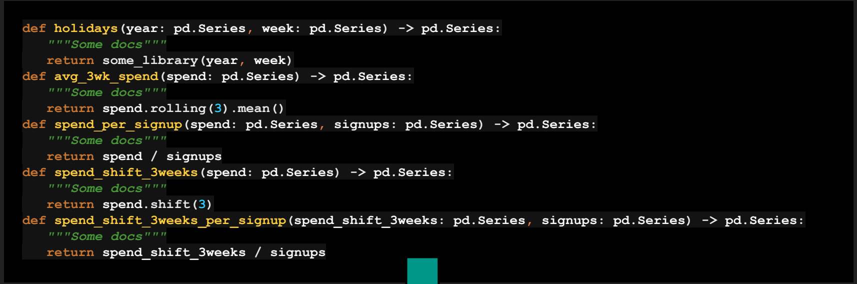
Year	Week	Sign ups	...	Spend	Holiday
2015	2	57	...	123	0
2015	3	58	...	123	0
2015	4	59	...	123	1
2015	5	59	...	123	1
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
2021	16	1000	...	1234	0

User

Hamilton:

Code → Dataflow → Object

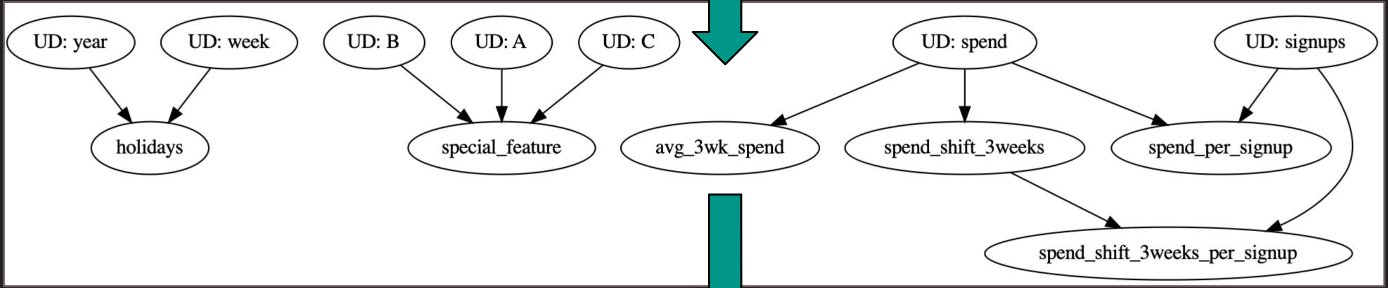
Code:



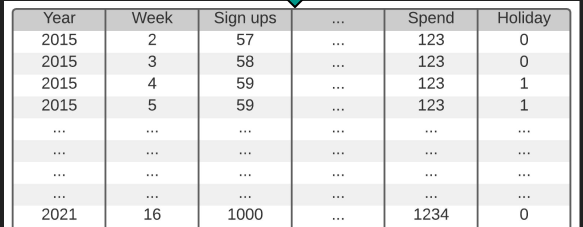
```python
def holidays(year: pd.Series, week: pd.Series) -> pd.Series:
    """Some docs"""
    return some_library(year, week)
def avg_3wk_spend(spend: pd.Series) -> pd.Series:
    """Some docs"""
    return spend.rolling(3).mean()
def spend_per_signup(spend: pd.Series, signups: pd.Series) -> pd.Series: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

```
a = b + c

a_prime = some_transform(a)
```

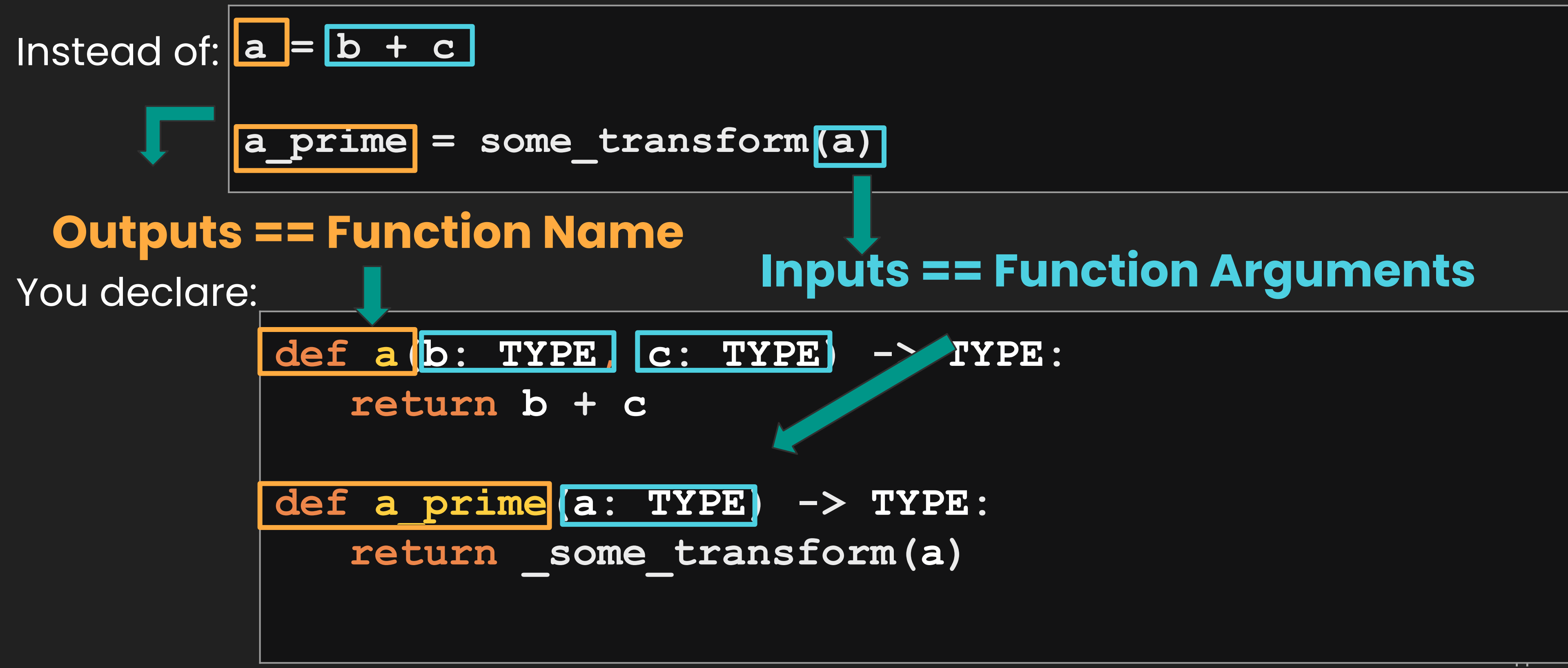
You declare:

```
def a(b: TYPE, c: TYPE) -> TYPE:
    return b + c

def a_prime(a: TYPE) -> TYPE:
    return _some_transform(a)
```

+ some driver code (not shown)

Hamilton Paradigm: declaring a dataflow



Instead of:

```
a = b + c

a_prime = some_transform(a)
```

Outputs == Function Name

Inputs == Function Arguments

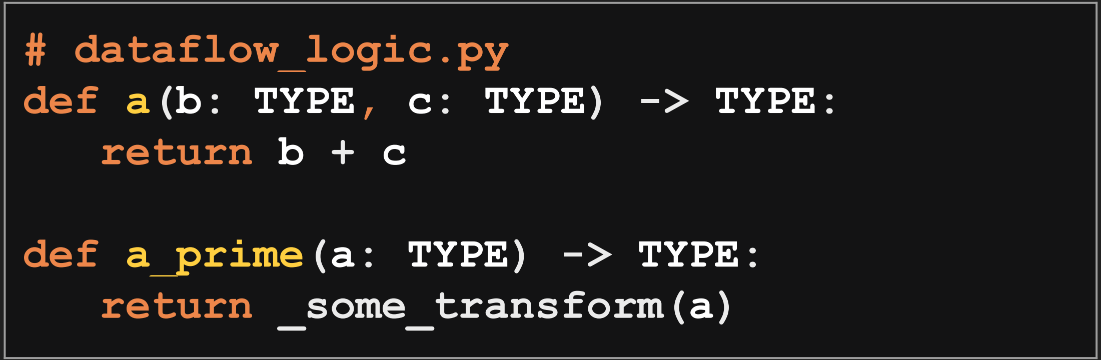
You declare:

```
def a(b: TYPE, c: TYPE) -> TYPE:
    return b + c

def a_prime(a: TYPE) -> TYPE:
    return _some_transform(a)
```

Hamilton TL;DR:

1. For each `=` statement, you write a function(s).
2. Functions define a dataflow.

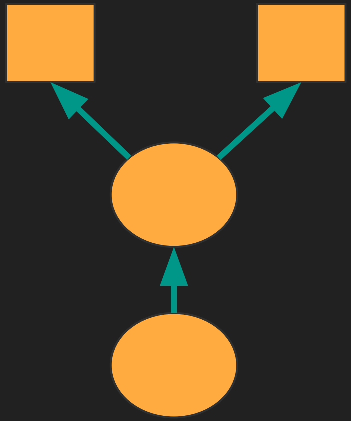


```
# dataflow_logic.py
def a(b: TYPE, c: TYPE) -> TYPE:
    return b + c

def a_prime(a: TYPE) -> TYPE:
    return _some_transform(a)
```

Hamilton TL;DR:

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



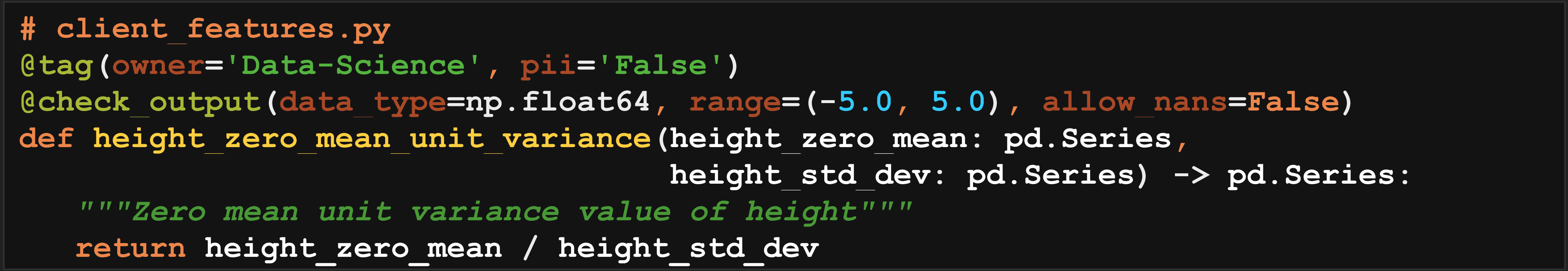
```
# dataflow_logic.py
def a(b: TYPE, c: TYPE) -> TYPE:
   return b + c

def a_prime(a: TYPE) -> TYPE:
   return _some_transform(a)
```

```
# run.py - “Driver code”
from hamilton import base, driver
import dataflow_logic
dr = driver.Driver(
  {'b': ...,}, dataflow_logic,
adapter=base.SimplePythonGraphAdapter(base
dict_result = dr.execute(['a_prime', ‘a’])
print(dict_result)
```

Hamilton Functions

Core to Hamilton – declarative 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
```

Core to Hamilton – declarative 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
```

Some benefits (see paper for more…):

- Software eng. best practices ✅ testing, docs, reuse, decoupling

Core to Hamilton – declarative 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
```

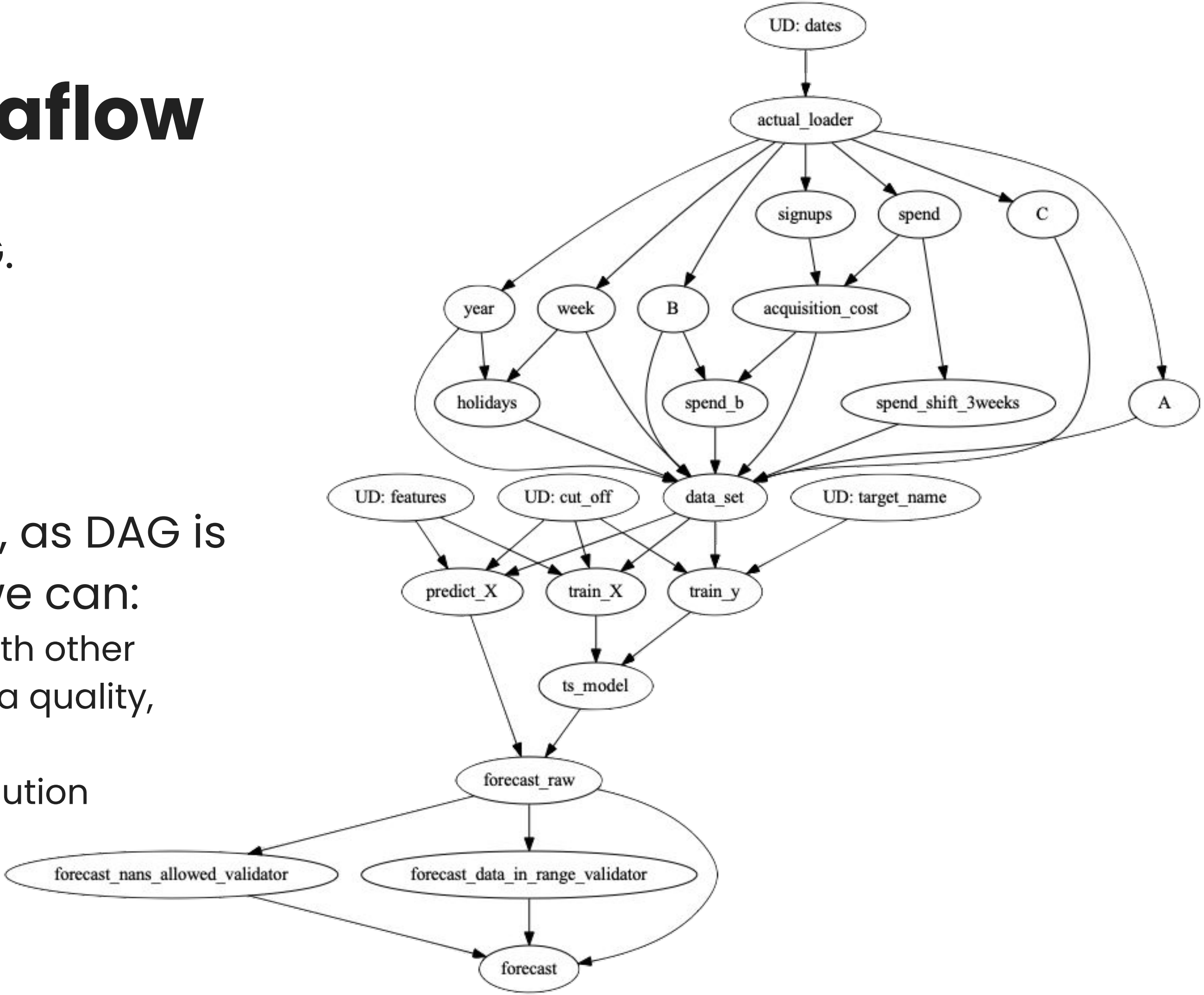
Some benefits (see paper for more…):

- Software eng. best practices ✅ testing, docs, reuse, decoupling
- Lineage ✅ “shift left”, DAG, versioning, `@tag`
- Modularity/composability ✅ stable UX, hide platform details, add capabilities e.g. data quality (`@check_output`), e.g. Ray/Dask

Example Dataflow

Example Dataflow

- Single logical DAG.
- Can materialize in multiple ways:
    - One step
    - Multiple steps
- Without UX clutter, as DAG is created/walked we can:
    - Wrap functions with other concerns, e.g. data quality, profiling, etc.
    - E.g. delegate execution



Evaluation

Hamilton @ Stitch Fix

Adoption:

- Running in production for 2.5+ years
- One DS team manages 4000+ feature definitions
- Best adoption from active time-series forecasting teams
  - Most willing to pay migration cost.
- Open source still early

Impact:

- Data Science teams ❤️ it:
  - Enabled a monthly feature update & model fitting task to be completed 4x faster

Hamilton @ Stitch Fix

Adoption:

- Running in production for 2.5+ years
- One DS team manages 4000+ feature definitions
- Best adoption from active time-series forecasting teams
  - Most willing to pay migration cost.
- Open source still early

WORK IN PROGRESS

Impact:

- Data Science teams ❤️ it:
  - Enabled a monthly feature update & model fitting task to be completed 4x faster

Summary & Future Work

Summary: *Hamilton - a modular O.S. declarative paradigm for high level modeling of dataflows*

- A declarative dataflow paradigm in python.
  - Functions, via naming, encode a logical dataflow.
  - Source code captures dataflow & can encode extra metadata.
- Modularity & composability:
  - Functions are the interface for UX and platform.
  - Decoupling of transform logic from materialization.

Future Work:

- Data governance & policy integrations
- Compiling to an orchestration framework
- Logically modeling your data warehouse
- HPC

Hamilton is Open Source Code

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

Get started in <15 minutes!

Star ⭐ on github:

https://github.com/stitchfix/hamilton

Documentation

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

Various examples:

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

Thank you.

Questions?

https://twitter.com/stefkrawczyk

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

https://github.com/stitchfix/hamilton