Hamilton

A Python Micro-Framework for tidy scalable Pandas

Stefan Krawczyk
August 2022

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.

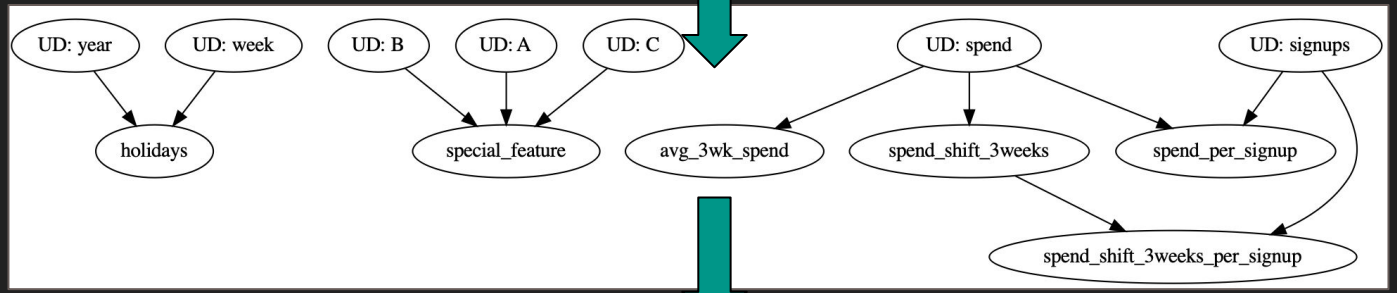
Hamilton:
Code → Directed Acyclic Graph → Object

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:

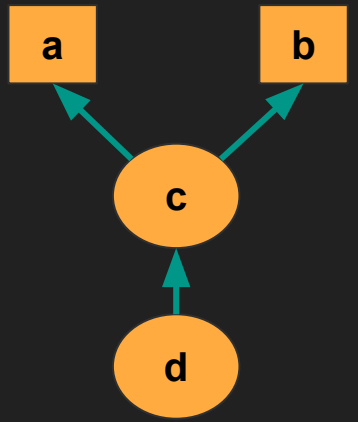
```python
# feature_logic.py
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

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.



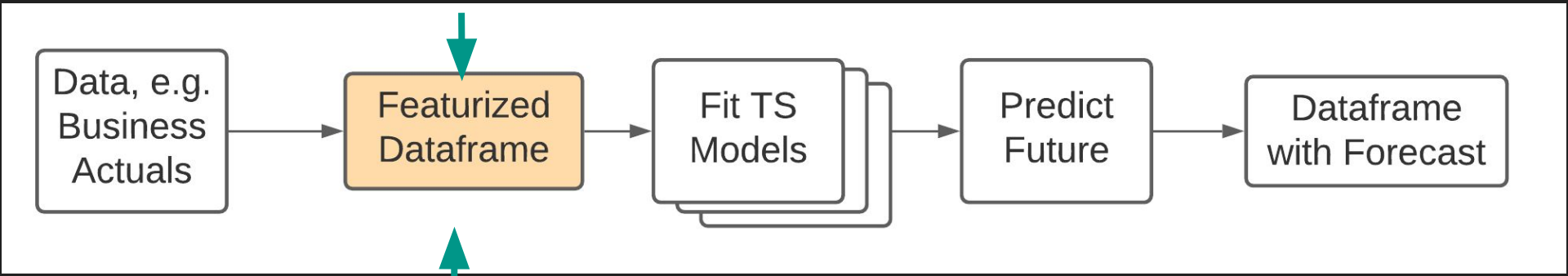

```python
# feature_logic.py
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Replaces c = a + b"""
    return a + b

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?

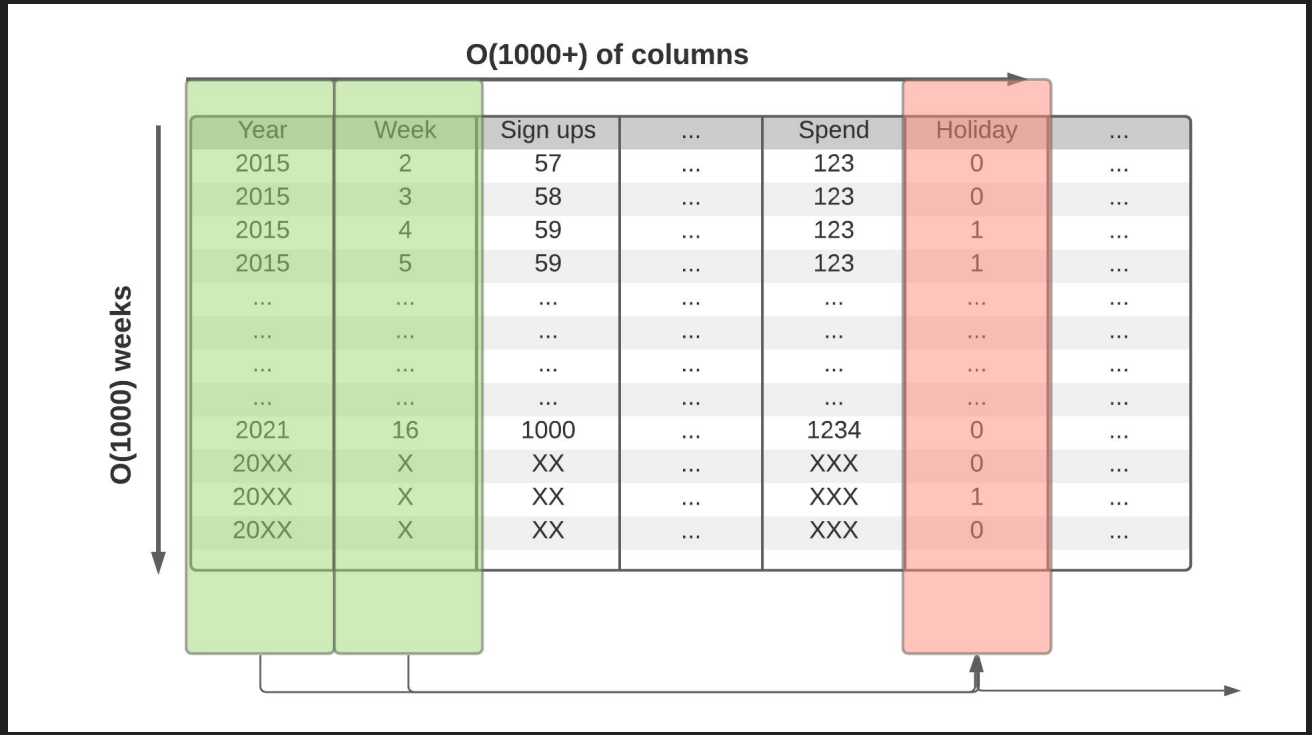
Backstory: Time-series Forecasting

Biggest problems here



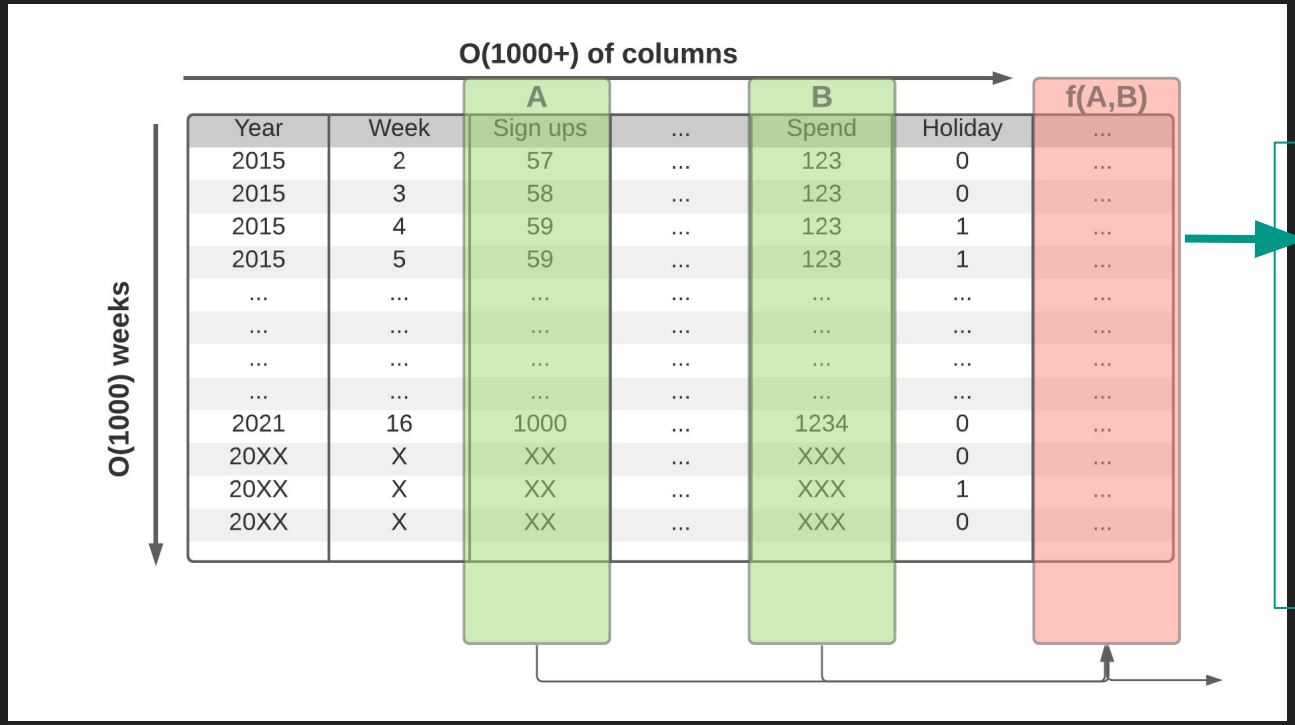
What Hamilton helped solve!

Backstory: TS -> Dataframe creation



Columns are functions of other columns

Backstory: TS -> Dataframe creation



Backstory: TS -> DF -> 🍝 Code

```
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()  # m                    spend
df['spend_per_signup'] = df['spend'] / df['si                    rson signed up
df['spend_shift_3weeks'] = df.spend[                    hift spend because ...
df['spend_shift_3weeks n                    _shift_3weeks'] / df['signups']

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

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

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: General usage of Hamilton

Overview: General usage of Hamilton

1. Create functions in module(s).
2. Create drivers to drive execution of those functions.
3. Execute driver code.

Notes:

- Can model any python object creation (not just pandas)
- Batch: use Hamilton within Airflow, Dagster, Prefect, Flyte, Metaflow, Kubeflow, Jupyter notebook etc.
- Online: embed within python streaming / python webserivce

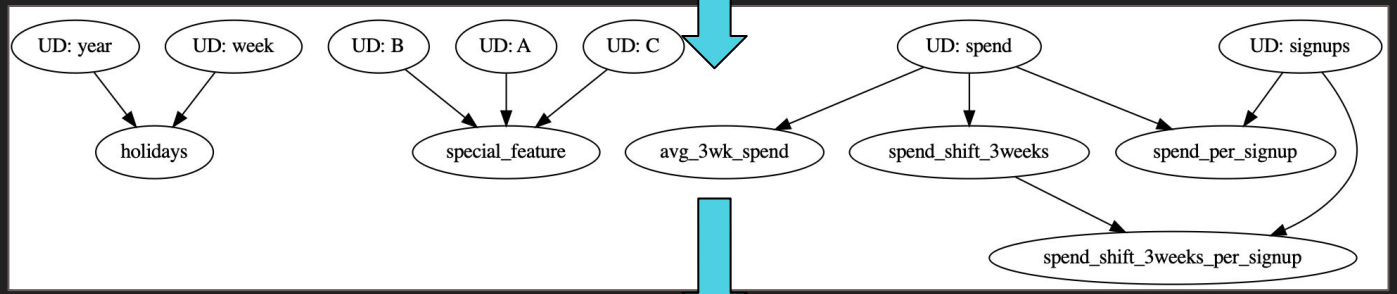
Modeling e.g. featurization

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:



run.py

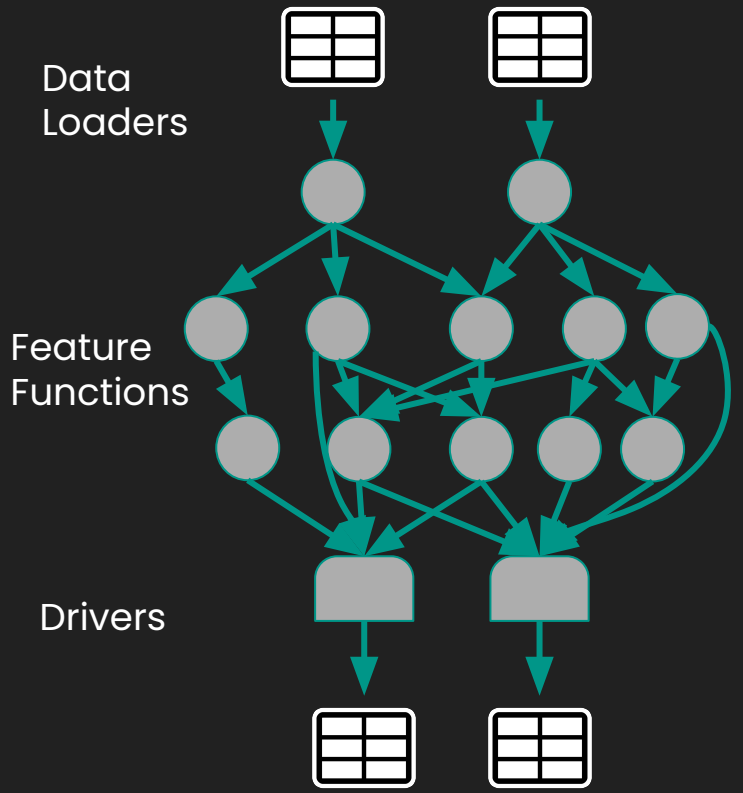
Feature Dataframe:

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

Modeling e.g. featurization

Code that needs to be written:

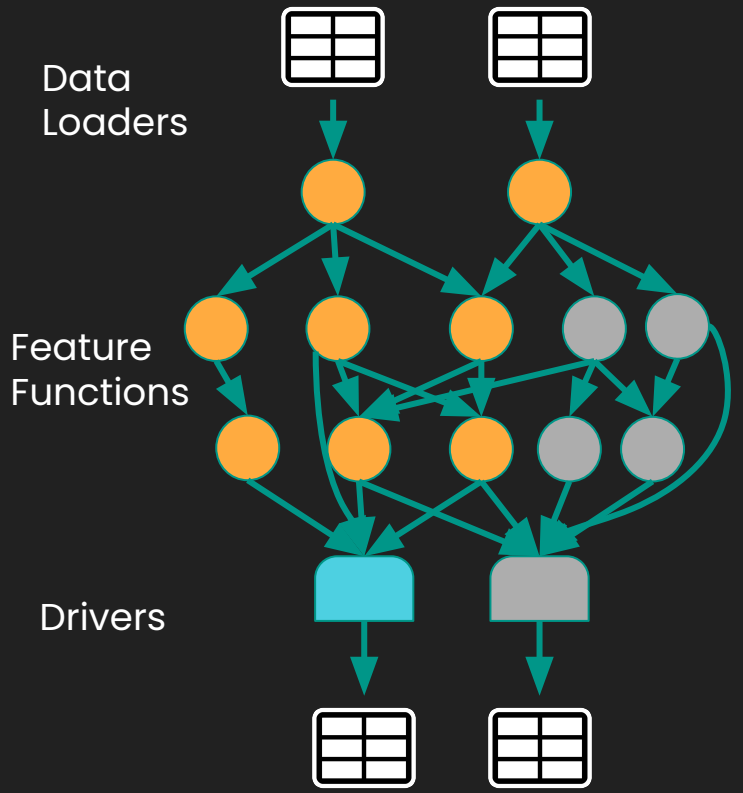
1. Functions to load data
   a. normalize/create common index to join on
2. Feature functions
   a. Optional: model functions.
3. Drivers materialize data
   a. DAG is walked for only what's needed.



Modeling e.g. 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 Pandas Code

Problems with Pandas Code

> 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! (will skip this part)

Scaling Humans/Teams

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
- Documentation
- Modularity/reuse

- Central feature definition store
- Data quality

✅ always possible
✅ tags, visualization, function doc
✅ module curation & decoupled drivers
✅ naming, curation, versioning
✅ 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

Summary

Summary: Hamilton for tidy pandas

- 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** paired with a system (e.g. modin, ray, etc) 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)

☑️ 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