

Hamilton: an open source micro framework for creating dataframes

December 2021

Stefan Krawczyk

 @stefkrawczyk
 [linkedin.com/in/skrawczyk](https://www.linkedin.com/in/skrawczyk)

Try out Stitch Fix → goo.gl/Q3tCQ3

#sfhamilton #MLOps #machinelearning

STITCH FIX

What to keep in mind for the next ~15 minutes?

1. Hamilton is a new paradigm to create dataframes*.
2. Hamilton enables Data Scientists to focus on functions rather than “glue” code.
3. <https://github.com/stitchfix/hamilton>



Outline: Hamilton

> Backstory: who, what, & why

Hamilton

The Result

Future

Backstory: who

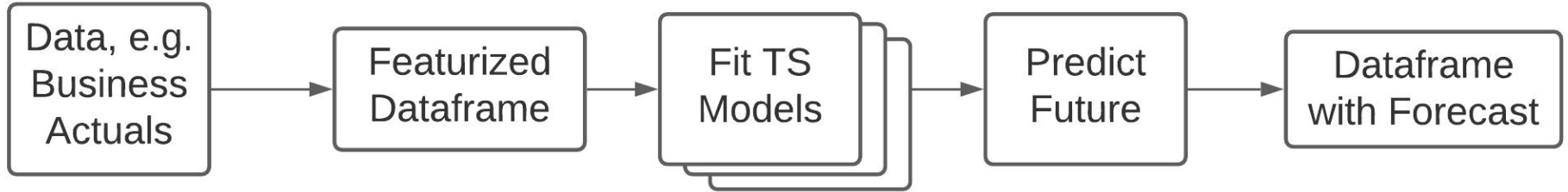
Forecasting, Estimation, & Demand (FED) Team

- Data Scientists that are responsible for forecasts that help the business make operational decisions.
 - e.g. staffing levels
- One of the oldest teams at Stitch Fix.

Backstory: what

Forecasting, Estimation, & Demand (FED) Team

FED workflow:

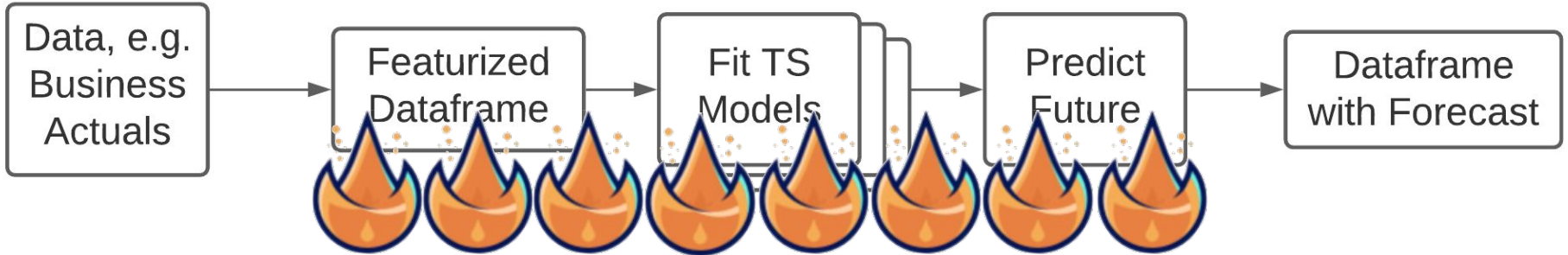


Backstory: what

Forecasting, Estimation, & Demand Team



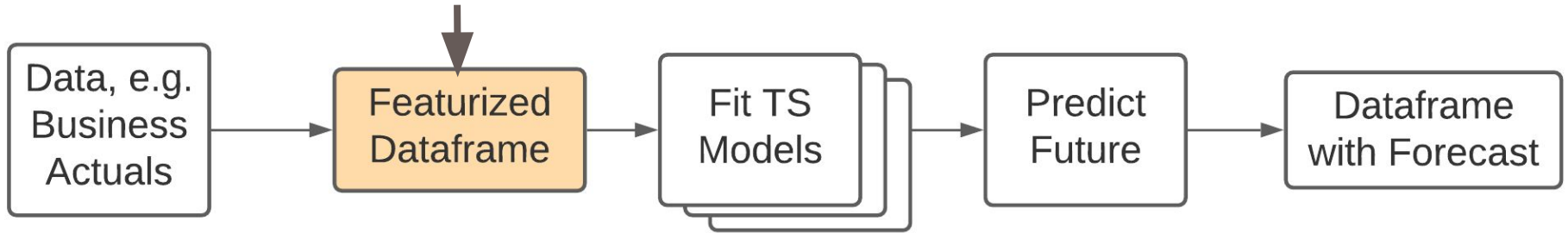
FED workflow:  +  ==



Backstory: what

Creating a dataframe for time-series modeling.

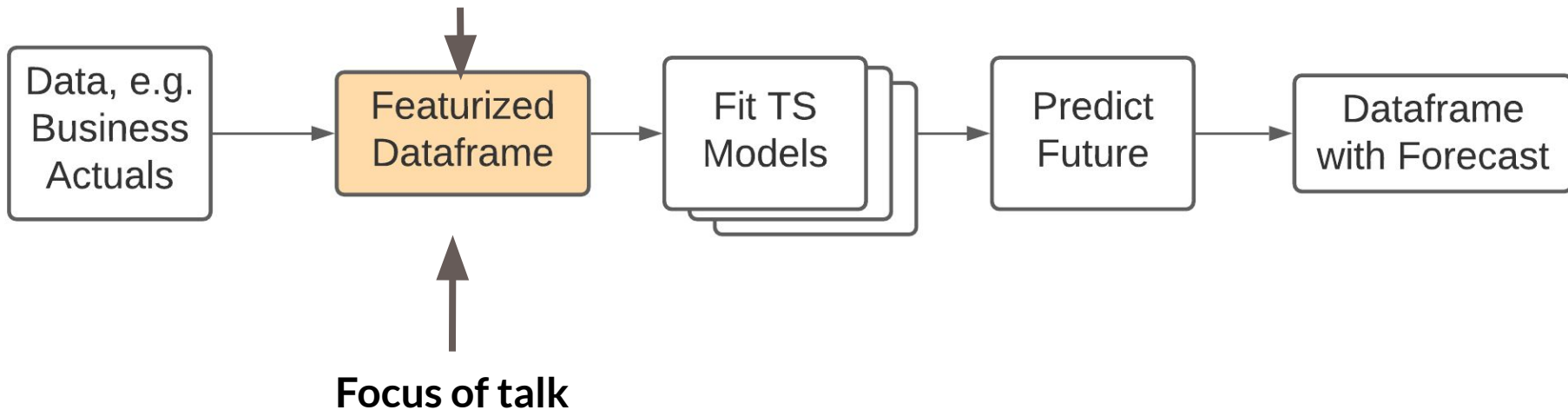
Biggest problems here



Backstory: what

Creating a dataframe for time-series modeling.

Biggest problems here



Backstory: why

What is this dataframe & why is it causing 

?

O(1000+) of columns →

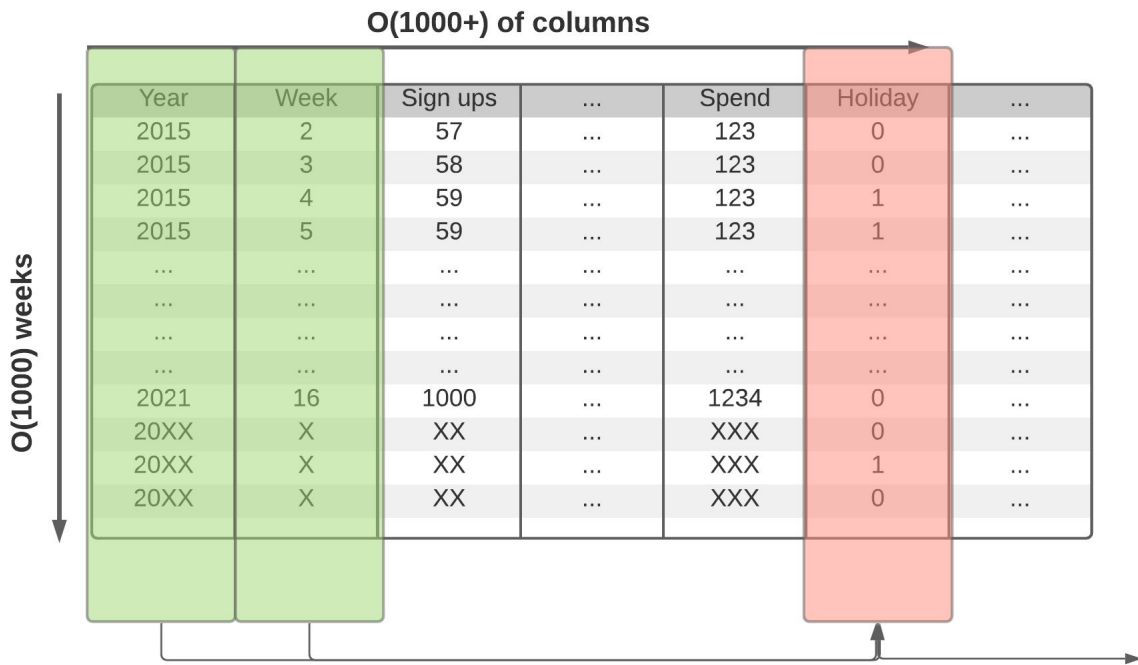
Year	Week	Sign ups	...	Spend	Holiday
2015	2	57	...	123	0
2015	3	58	...	123	0
2015	4	59	...	123	1
2015	5	59	...	123	1
...
...
...
...
2021	16	1000	...	1234	0
20XX	X	XX	...	XXX	0
20XX	X	XX	...	XXX	1
20XX	X	XX	...	XXX	0

← **O(1000) weeks**

(not big data)

Backstory: why

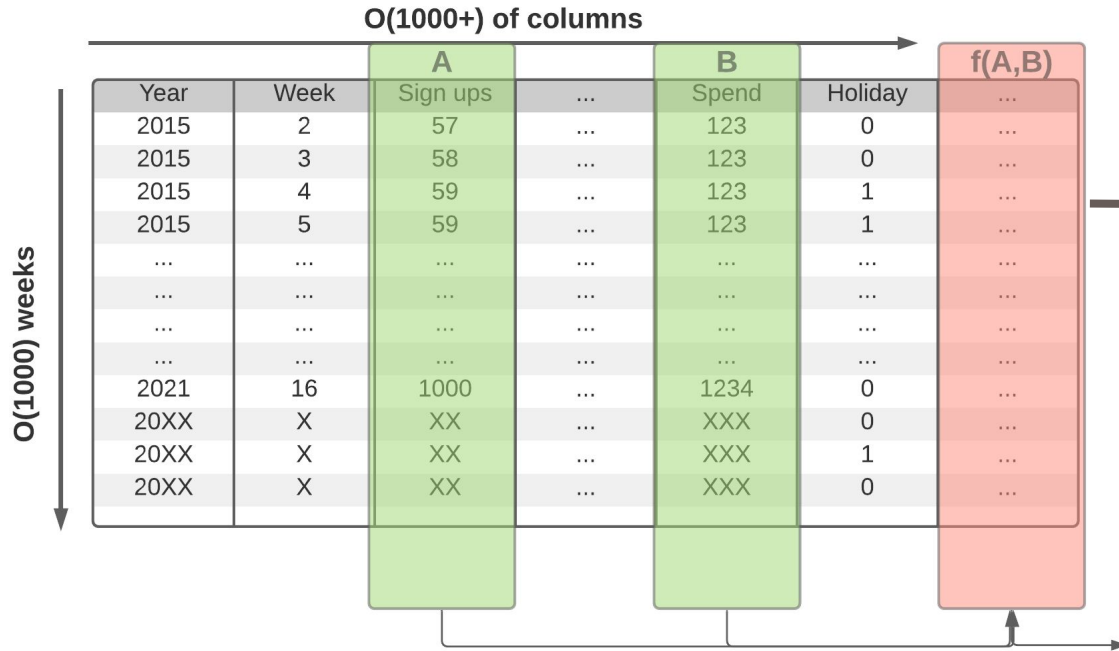
What is this dataframe & why is it causing 🔥 ?



Columns are functions of other columns

Backstory: why

What is this dataframe & why is it causing  ?



Columns are functions of other columns:

$g(f(A,B), \dots)$

$h(g(f(A,B), \dots), \dots)$

etc 

Backstory: why

Featurization: some example 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() # moving average of spend
df['spend_per_signup'] = df['spend'] / df['signups'] # spend per person signed up
df['spend_shift_3weeks'] = df.spend['spend'].shift(3) # shift spend because ...
df['spend_shift_3weeks_per_signup'] = df['spend_shift_3weeks'] / df['signups']

def my_special_feature(df: pd.DataFrame) -> pd.Series:
    return (df['A'] - df['B'] + df['C']) * weights

df['special_feature'] = my_special_feature(df)
# ...
```

Backstory: why

Featurization: some example 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() # moving average of spend
df['spend_per_signup'] = df['spend'] / df['signups'] # spend per person signed up
df['spend_shift_3weeks'] = df['spend'].shift(3) # spend 3 weeks ago
df['spend_shift_3weeks_normalized'] = df['spend_shift_3weeks'] / df['signups']

def my_special_feature(df: pd.DataFrame) -> pd.Series:
    return (df['A'] - df['B'] + df['C']) * weights

df['special_feature'] = my_special_feature(df)
# ...
```

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

Backstory: why



```
df = load_dates() # load date ranges
df = load_actuals()
df['holidays'] = ...
df['avg_3wk_spend'] = ...
df['spend_per_shift'] = ...
df['spend_shift'] = ...

def my_special_feature(df):
    return (df['spend_per_shift'] * df['spend_shift'])

df['special_feature'] = my_special_feature(df)
# ...
```

Scaling this type of code results in the following:


- lots of heterogeneity in function definitions & behaviors
- inline dataframe manipulations
- code ordering is super important

- Unit testing
- Documentation
- Onboarding 
- Debugging 



Backstory - Summary

1. Code for featurization == 🤖.
2. Data all fits in memory.



Outline: Hamilton
Backstory: who, what, & why
> Hamilton
The Result
Future

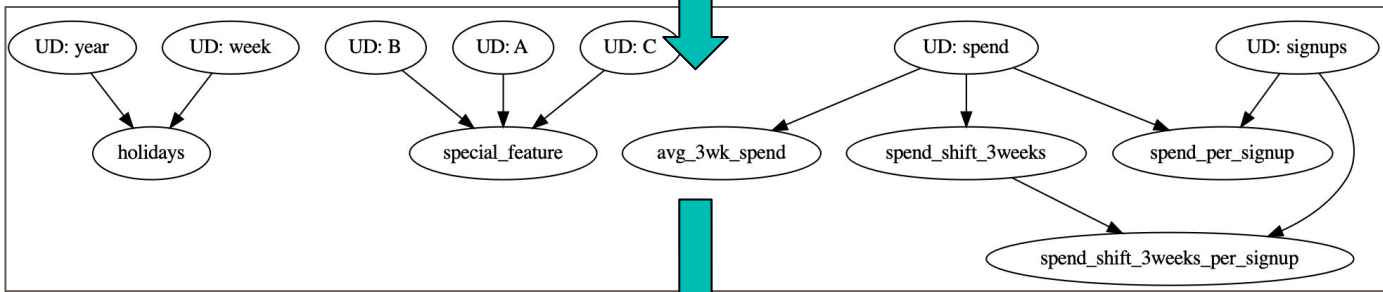
Hamilton: Code → DAG → DF

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
```

DS

DAG:



Platform

DF:

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

DS

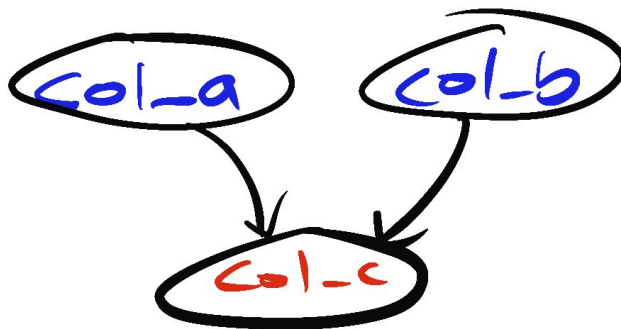
Hamilton: a new paradigm

1. Write functions!
2. **Function name**
== output column
3. **Function inputs**
== input columns

Code: $df["col_c"] = df["col_a"] + df["col_b"]$

```
def col_c(col_a: pd.Series, col_b: pd.Series) -> pd.Series:  
    "documentation goes here"  
    return col_a + col_b
```

DAG:




Hamilton: a new paradigm

4. Use **type hints** for typing checking.
5. Documentation is easy and natural.

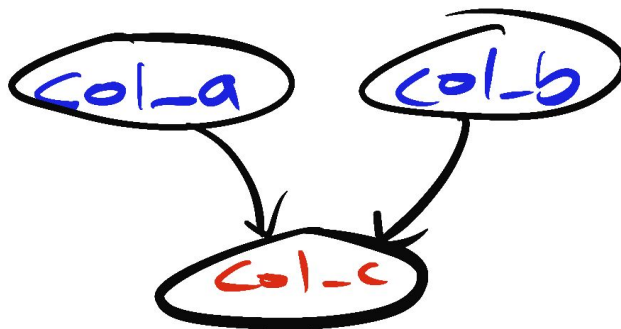
Code:

```
df["col_c"] = df["col_a"] + df["col_b"]
```



```
def col_c(col_a: pd.Series, col_b: pd.Series) -> pd.Series:  
    "documentation goes here"  
    return col_a + col_b
```

DAG:



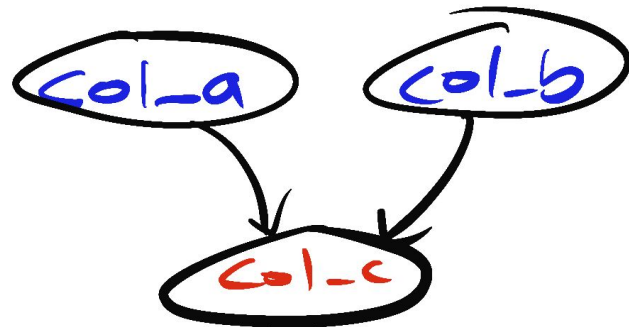
Hamilton: code to DAG - how?

1. Inspect module.
2. Nodes & edges + graph theory 101.

Code:

```
df["col_c"] = df["col_a"] + df["col_b"]  
  
def col_c(col_a: pd.Series, col_b: pd.Series) -> pd.Series:  
    "documentation goes here"  
    return col_a + col_b
```

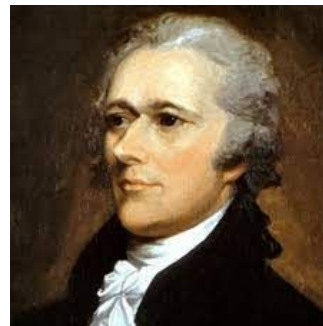
DAG:



Hamilton: why is it called Hamilton?

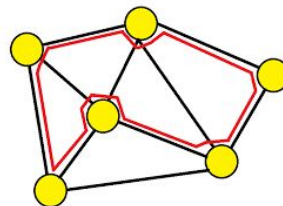
Naming things is hard...

1. Associations with “FED”:
 - a. Alexander Hamilton is the father of the Fed.
 - b. FED models business mechanics.
2. We’re doing some basic graph theory.



$$H_{operator} = \frac{-\hbar^2}{2m} \frac{\partial^2}{\partial x^2} + V(x)$$

Operator associated with kinetic energy Potential energy



apropos Hamilton



Example Hamilton Code

So you can get a feel for this paradigm...

Basic code - defining “Hamilton” functions

my_functions.py

```
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
```

Basic code - defining “Hamilton” functions

my_functions.py

```
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
```

Output Column

Input Column

Driver code - how do you get the Dataframe?

```
from hamilton import driver
initial_columns = { # load from actuals / initial data
    'A': ...,
    'B': ...,
    'C': 3,
    'signups': ...,
    'spend': ...,
    'week': ...,
    'year': ...
}
module_name = 'my_functions' # e.g. my_functions.py
module = importlib.import_module(module_name) # The python file to crawl
dr = driver.Driver(initial_columns, module) # can pass in multiple modules

output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']

df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```

Driver code - how do you get the Dataframe?

```
from hamilton import driver

initial_columns = { # load from actuals / initial data
    'A': ...,
    'B': ...,
    'C': 3,
    'signups': ...,
    'spend': ...,
    'week': ...,
    'year': ...
}

module_name = 'my_functions' # e.g. my_functions.py
module = importlib.import_module(module_name) # The python file to crawl

dr = driver.Driver(initial_columns, module) # can pass in multiple modules

output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']

df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```

Driver code - how do you get the Dataframe?

```
from hamilton import driver
initial_columns = { # load from actuals / initial data
    'A': ...,
    'B': ...,
    'C': 3,
    'signups': ...,
    'spend': ...,
    'week': ...,
    'year': ...
}

module_name = 'my_functions' # e.g. my_functions.py
module = importlib.import_module(module_name) # The python file to crawl

dr = driver.Driver(initial_columns, module) # can pass in multiple modules

output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']

df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```

Driver code - how do you get the Dataframe?

```
from hamilton import driver
initial_columns = { # load from actuals / initial data
    'A': ...,
    'B': ...,
    'C': 3,
    'signups': ...,
    'spend': ...,
    'week': ...,
    'year': ...
}
module_name = 'my_functions' # e.g. my_functions.py
module = importlib.import_module(module_name) # The python file to crawl
dr = driver.Driver(initial_columns, module) # can pass in multiple modules
output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']
df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```

Driver code - how do you get the Dataframe?

```
from hamilton import driver
initial_columns = { # load from actuals / initial data
    'A': ...,
    'B': ...,
    'C': 3,
    'signups': ...,
    'spend': ...,
    'week': ...,
    'year': ...
}
module_name = 'my_functions' # e.g. my_functions.py
module = importlib.import_module(module_name) # The python file to crawl
dr = driver.Driver(initial_columns, module) # can pass in multiple modules
output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']
df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```

Driver code - how do you get the Dataframe?

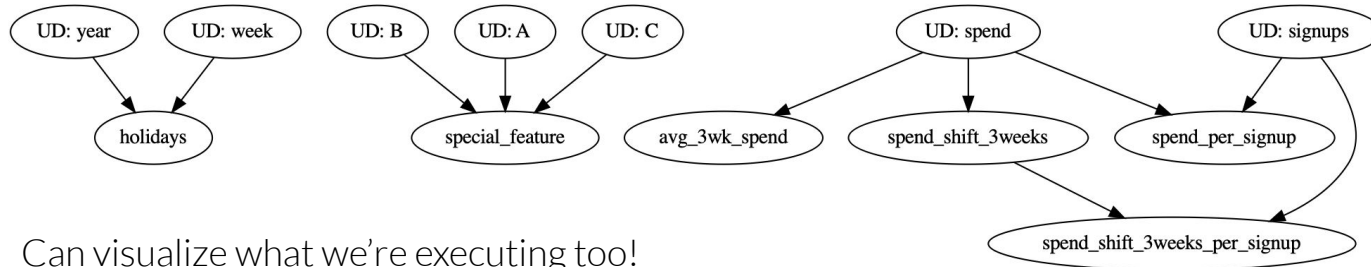
```
from hamilton import driver
initial_columns = { # load from actuals / initial data
    'A': ...,
    'B': ...,
    'C': 3,
    'signups': ...,
    'spend': ...,
    'week': ...,
    'year': ...
}
module_name = 'my_functions' # e.g. my_functions.py
module = importlib.import_module(module_name) # The python file to crawl
dr = driver.Driver(initial_columns, module) # can pass in multiple modules

output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']

df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```

Driver code - how do you get the Dataframe?

```
from hamilton import driver
initial_columns = { # load from actuals / initial data
```




```
module = importlib.import_module(module_name) # The python file to crawl
```

```
dr = driver.Driver(initial_columns, module) # can pass in multiple modules
```

```
output_columns = ['year', 'week', ..., 'spend_shift_3weeks_per_signup', 'special_feature']
```

```
df = dr.execute(output_columns, display_graph=True) # only walk DAG for what is needed
```



Outline: Hamilton
Backstory: who, what, & why
Hamilton
> The Result
Future

The Result - after 1.5+ years in production



The Result:

```
def col_c(col_a: pd.Series, col_b: pd.Series) -> pd.Series:  
    """documentation goes here"""  
    return col_a + col_b
```

- Unit testing (found bugs during migration)
- Documentation
- Visualization
- Onboarding
- Debugging
- Syntactic sugar to keep things DRY
- DS focus on what matters
- Platform owns “glue” code



The Result - after 1.5+ years in production

DS Quote:

I've previously onboarded at an organization that dealt with multiple layers of information dependency. The "data product" was a result of years of multiple authors adding layers without systematic examination for potholes like circular references. Because of that, the product was a terribly fragile product and knowledge transfers took place through a series of ad-hoc trial and error by a new member; they run into an issue, ask their supervisor for clarification, then they are handed an opaque explanation and workaround which then gets propagated through a game of telephone to the next person (the mechanics of solution is propagated, but the reason behind the solution is not). In my own experience, the ah-ha moment did not occur until I doggedly followed the thread of information and built a dag on my own.

Having had that experience, onboarding a data product that already had a graph structure to embody the complex dependency was a delight; chief among its many benefits is that the product is amenable to other generic analysis approaches for graphs. Moreover, because of the abstraction that separates dataframe structure from quantitative specification, it helps a new person to process the information without having to have a priori domain knowledge, since dependencies are clearly specified and functions are simple and concise.

The Result - after 1.5+ years in production

DS Quote: TL;DR:

Before/Previous:

- *understanding code base is hard.*
- *it's not obvious where things are used.*
- *easy to break things.*

With Hamilton:

- *no prior domain knowledge required to ramp up.*
- *dependencies are clearly specified.*
- *functions are simple and concise.*
- *hard to break things.*

The Result: try it for yourself!

> **pip install sf-hamilton**

Get started in < 15 minutes!

Documentation - see README -

<https://github.com/stitchfix/hamilton#hamilton-in-15-minutes>

Example

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

The Result: try it for yourself!


> **pip install sf-hamilton**

<https://github.com/stitchfix/hamilton>

 on github

 create issues on github

 join us on [discord](#)



Outline: Hamilton
Backstory: who, what, & why
Hamilton
The Result
> Future

Future

Ideas:

- distributed processing
 - we have a DAG, why not farm it out?
- why pandas?
 - no reason we couldn't support other data structures
 - extend to anything that can be “unioned/merged” together
- general featurization
 - can we take this more mainstream?

