FEATURE STORE SUMMIT 2023

Feature Engineering with Hamilton

Write once / run everywhere

Elijah ben Izzy | Co-founder/CTO | DAGWORKS

Organized by HOPSWORKS

TL;DR

I want to convince you that...

1. Writing *portable* feature engineering code is hard
🌶️ SOTA approaches aren’t flexible/powerful enough
3. Hamilton can help you:
    a. Write code to run in multiple contexts
    b. Keep your code organized/clean
4. Hamilton is easy to get started with/easy to use!

DAGWORKS

The unifying layer for Data, ML, and LLM pipelines

Open Core!

>>> I’m not selling you anything in this talk! <<<

DAGWORKS

Hamilton is Open Source!!

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

Get started in <15 minutes!

Documentation

https://hamilton.readthedocs.io/

Try it out

https://www.tryhamilton.dev/

DAGWORKS

https://www.tryhamilton.dev

Hamilton

Wrangle Pandas codebases into shape.

Learn (5 mins) Github 890+ ⭐

- ☑ Write always unit testable code
- ☑ Add runtime data validation easily
- ☑ Produce readable and maintainable code
- ☑ Visualize lineage (click the run button to see)
- ☑ Run anywhere python runs: in airflow, jupyter, fastapi, etc...
- ☑ Skip the CS degree to use it

Try Hamilton right here in your browser 👇

```python
# Declare and link your transformations as functions....
import pandas as pd

def a(input: pd.Series) -> pd.Series:
    return input % 7

def b(a: pd.Series) -> pd.Series:
    return a * 2

def c(a: pd.Series, b: pd.Series) -> pd.Series:
    return a * 3 + b * 2

def d(c: pd.Series) -> pd.Series:
    return c ** 3
```

```python
# And run them!
import functions
from hamilton import driver
dr = driver.Driver({}, functions)
result = dr.execute(
   ['a', 'b', 'c', 'd'],
   inputs={'input': pd.Series([1, 2, 3, 4, 5])}
)
print(result)
dr.display_all_functions("graph.dot", {})
```

▶ Run me!

DAGWORKS

The Agenda

The problem with feature engineering
The solution: *Hamilton*
Write once, run everywhere
- ↳ **Batch**
- ↳ **Streaming**

Additional benefits of Hamilton
OS progress/updates

The Agenda

The problem with feature engineering

The solution: *Hamilton*

Write once, run everywhere

- ↳ Batch
- ↳ Streaming

Additional benefits of Hamilton

OS progress/updates

Why feature engineering is hard

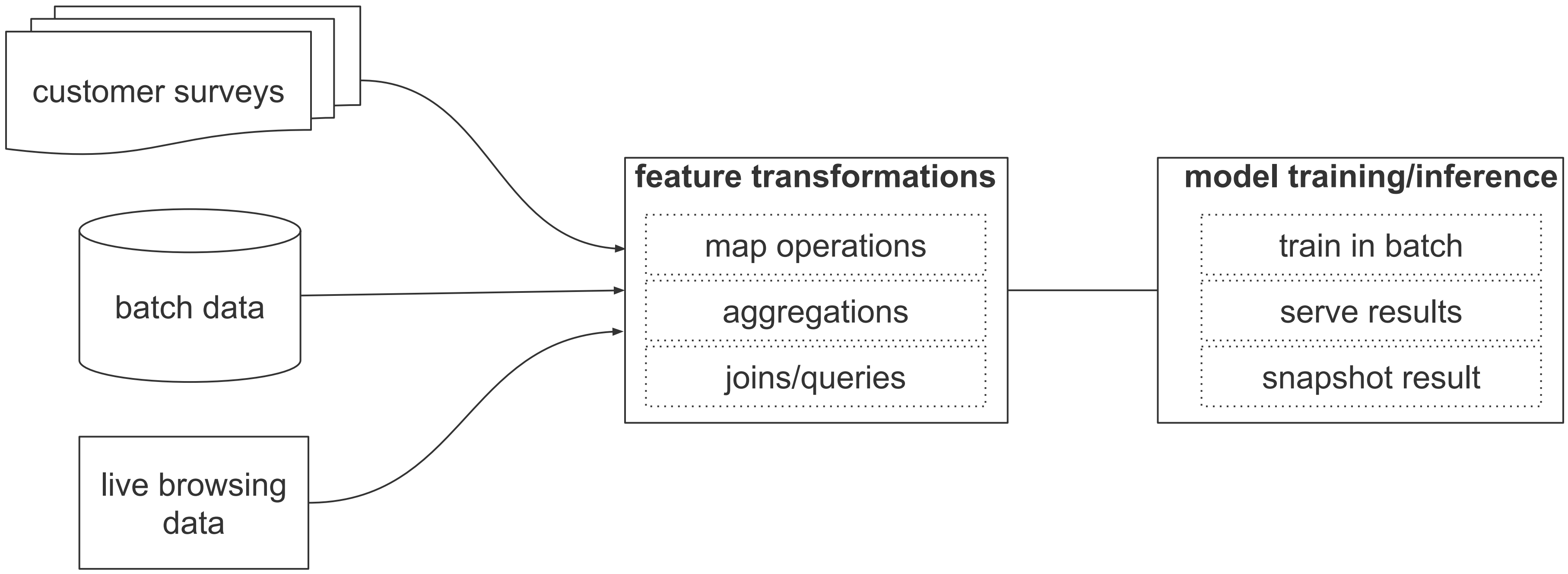
Common scenario (e-commerce)

- ❑ Customers fill out survey results
- ❑ Your model makes predictions
- ❑ **Goal**: get survey results to model

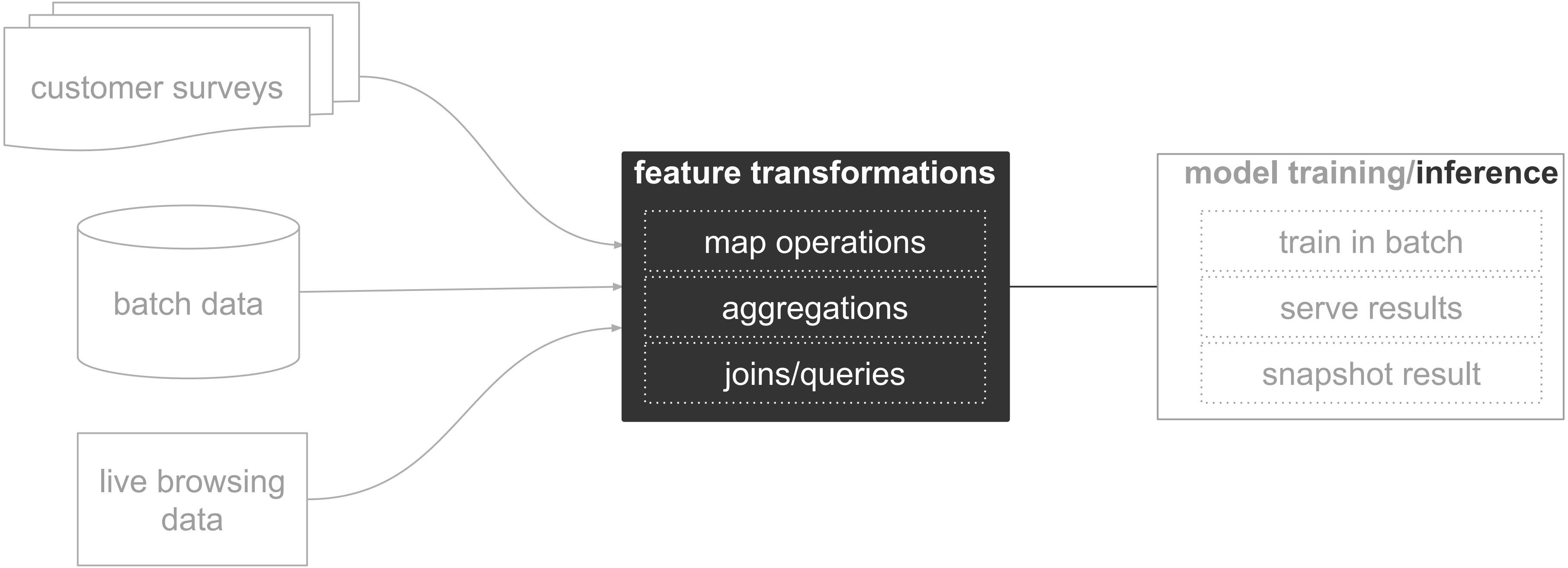
Caveats

- ❑ Survey results trickle in (streaming)
- ❑ Data comes in dumps nightly (batch)
- ❑ Multiple teams working together (features x infra x data)
- ❑ Features are derived from data (model = h(g(f(x)), …)

Why feature engineering is hard



Why feature engineering is hard



DAGWORKS

Why feature engineering is hard

Contexts

- ❏ Run on tables in your data warehouse for training data
- ❏ Run inside a streaming processor for *near-real-time*
- ❏ Transform browsing data live

Complications

- ❏ Ensuring the data is the same in all contexts:
  - ❏ How do you handle joins/alt data sources in non-batch mode?
  - ❏ How do you include aggregations in streaming mode?
- ❏ How do you track lineage, versions, etc… for different data sources?

Why feature engineering is hard

Current approaches

Context-specific execution | Feature DSL to unify

Context-specific execution:

- Cumbersome to manage
- 2 sets of tests
- 2 sets of versions
- **Do they match?**

Feature DSL to unify:

- Tougher to grok
- Limited to specific operations
- Opinionated on agg, joins

DAGWORKS

Why feature engineering is hard

Idea – can we write normal python code that is…

- DRY (**d**on't **r**epeat **y**ourself)
- Applicable in all settings
- Fully customizable:
  - You decide joins
  - You decide aggregation approach
  - You write map fns however you want
  - Bring your own infrastructure
- Self-documenting + implies structure

The Agenda

The problem with feature engineering
The solution: *Hamilton*
Write once, run everywhere
- ↳ Batch
- ↳ Streaming

Additional benefits of Hamilton
OS progress/updates

Hamilton: the “a-ha” Moment

Idea What if every feature corresponded to **exactly one** python fn?

And… what if the way that function was written tells you everything you needed to know?

*In Hamilton, the artifact (feature) is determined by the **name of the function**. The dependencies are determined by **the parameters**.*

Old way vs Hamilton way:

Instead of*

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

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
```

**Hamilton supports *all* python objects, not just dfs/series!*

DAGWORKS

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
```

**Hamilton supports *all* python objects, not just dfs/series!*

DAGWORKS

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
```

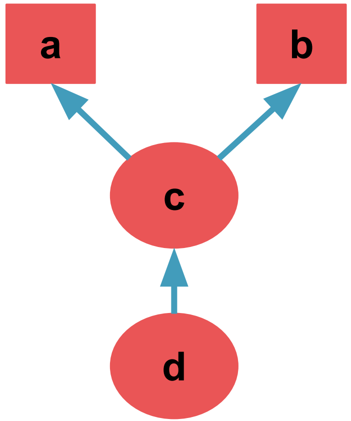
Driver says what/when to execute

```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)
```

DAGWORKS

Hamilton TL;DR

1. For each transform (=), 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)
```

Hamilton: extensions

Q: Doesn't Hamilton make your code more verbose?

A: Yes, but that's not always a bad thing. When it is, we have decorators!

- ❑ `@tag # attach metadata`
- ❑ `@parameterize # curry + repeat a function`
- ❑ `@extract_columns # one dataframe -> multiple series`
- ❑ `@check_output # data validation`
- ❑ `@config.when # conditional transforms`
- ❑ `@subdag # recursively utilize groups of nodes`
- ❑ `@... # new ones all the time`

DAGWORKS

The Agenda

The problem with feature engineering
The solution: *Hamilton*
Write once, run everywhere
- ↳ Batch
- ↳ Streaming

Additional benefits of Hamilton
OS progress/updates

DAGWORKS

Write once, run everywhere

One* feature per function

- ❑ Map operations – single versus bulk operations are equivalent
- ❑ Aggregation* – you choose (store, compute on the fly, update regularly, etc…)
- ❑ Joins* – use query instead of join

**for aggregations/joins you reimplement just the parts you need to*

Write once, run everywhere

Back to our scenario…

- ❏ Simple map operations
  - ❏ raw survey data -> [budget, gender, age]
  - ❏ *derived* features [is_high_roller, is_male, is_female]
- ❏ Joins
  - ❏ time_since_last_login = **f**(client_id, login_data)
- ❏ Aggregations
  - ❏ normalized_age = **g**(mean(age), stddev(age))

The Agenda

The problem with feature engineering
The solution: *Hamilton*
Write once, run everywhere
↳ **Batch**
↳ Streaming
Additional benefits of Hamilton
OS progress/updates

Batch feature engineering

Goal

- ❑ Compute features/infer model in batch

Context

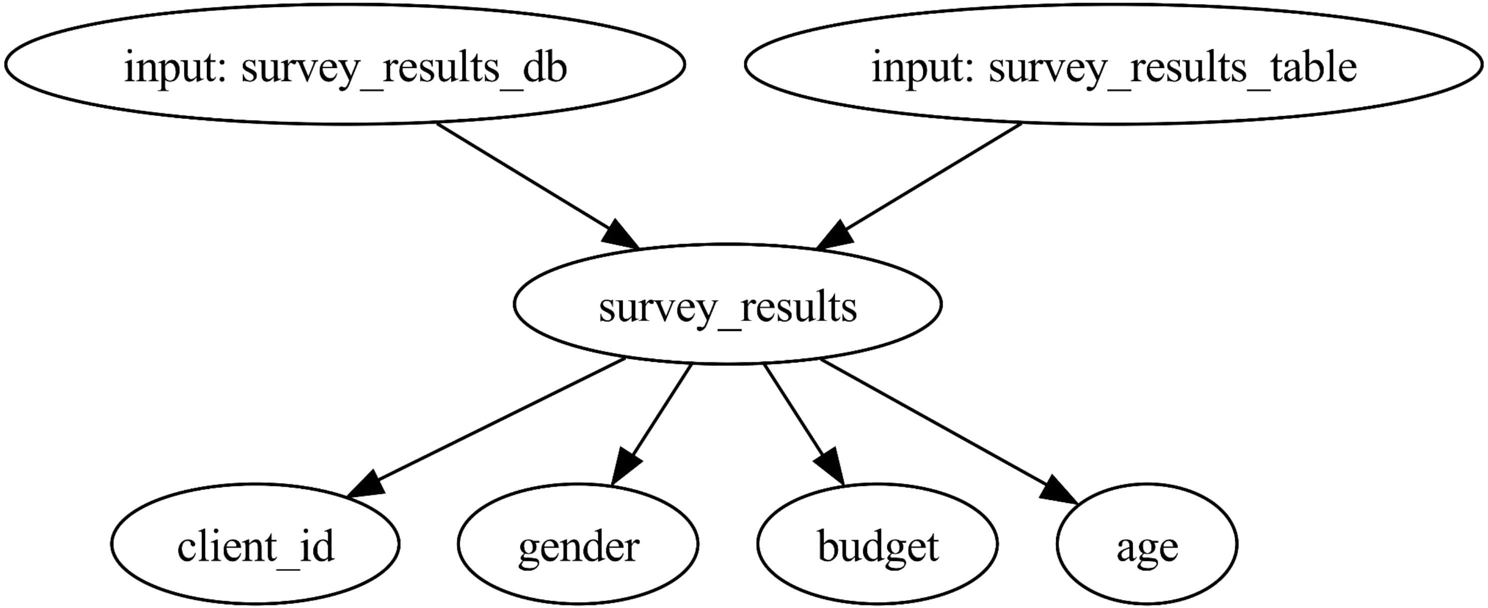
- ❑ DB with raw survey results
- ❑ DB with client login data
- ❑ Model already trained *[you can use this for training]*
- ❑ Data is reasonable size *[Hamilton can scale too]*

Data loading

```python
@extract_columns(
    'budget',
    'age',
    'gender',
    'client_id'
)
def survey_results(
    survey_results_table: str,
    survey_results_db: str) -> pd.DataFrame:
    """Map operation to explode survey results to all fields
    Data comes in JSON, we've grouped it into a series.
    """
    conn = Connection(survey_results_db)
    return pd.read_sql(conn, f"SELECT * FROM {survey_results_table}")
```

DAGWORKS

Data loading

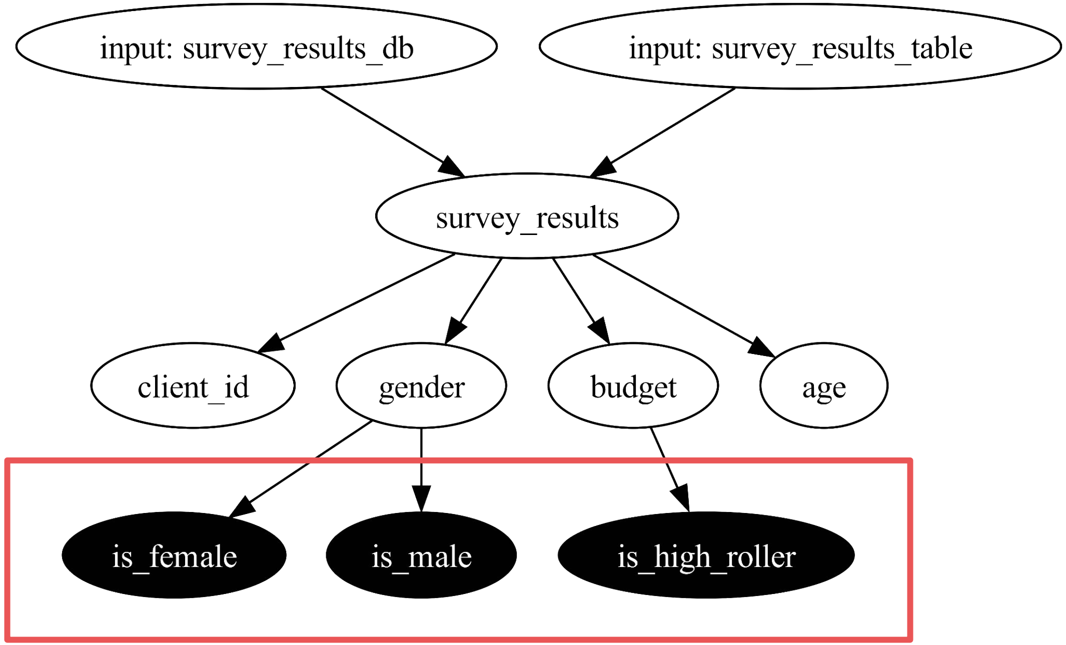


DAGWORKS

Map functions

```python
def is_male(gender: pd.Series) -> pd.Series:
    return gender == 'male'

def is_female(gender: pd.Series) -> pd.Series:
    return gender == 'female'

def is_high_roller(budget: pd.Series) -> pd.Series:
    return budget > 1000
```

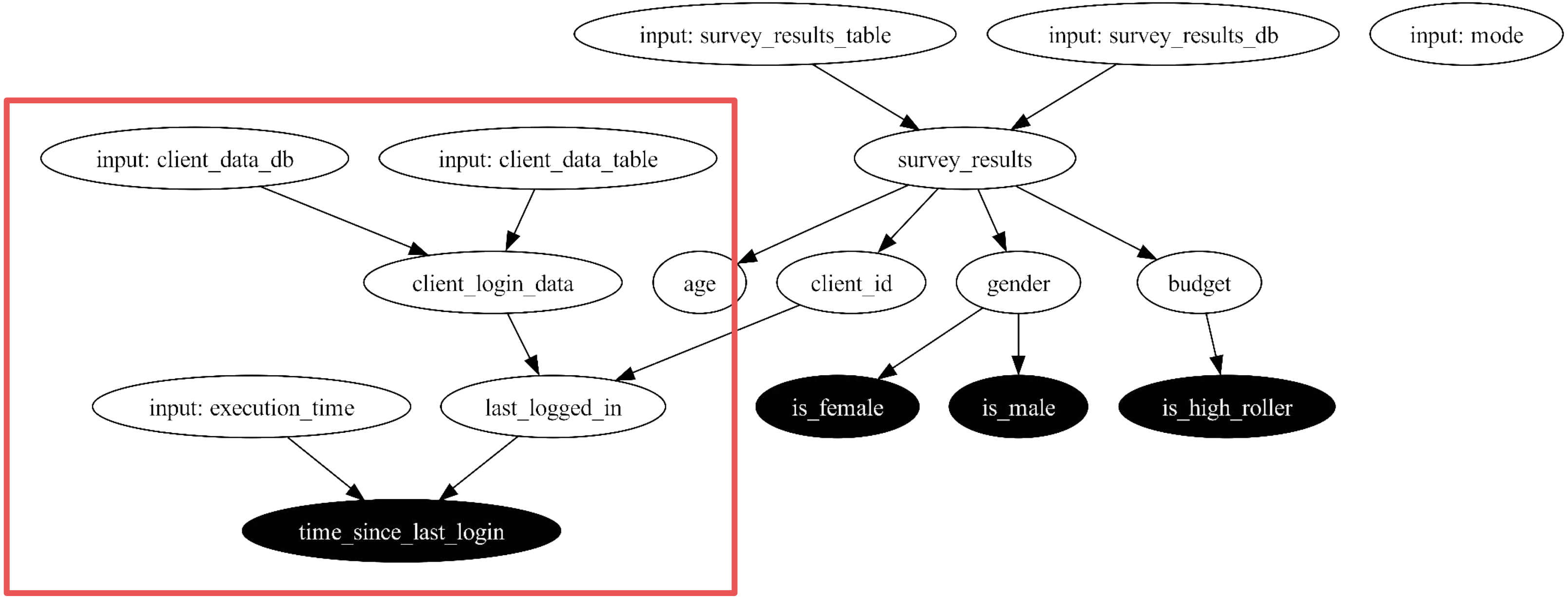
Map functions



Joins

```python
def client_login_data(table: str, db: str) -> pd.DataFrame:
    conn = create_connection(db)
    return pd.read_sql(f"SELECT * from {table}")

def last_logged_in(client_id: pd.Series, client_login_data: pd.DataFrame) -> pd.Series:
    return pd.merge(
        client_id,
        client_login_data,
        left_on='client_id',
        right_index=True)['last_logged_in']

def time_since_last_login(
    execution_time: datetime.datetime,
    last_logged_in: pd.Series) -> pd.Series:
    return execution_time - last_logged_in
```

Joins

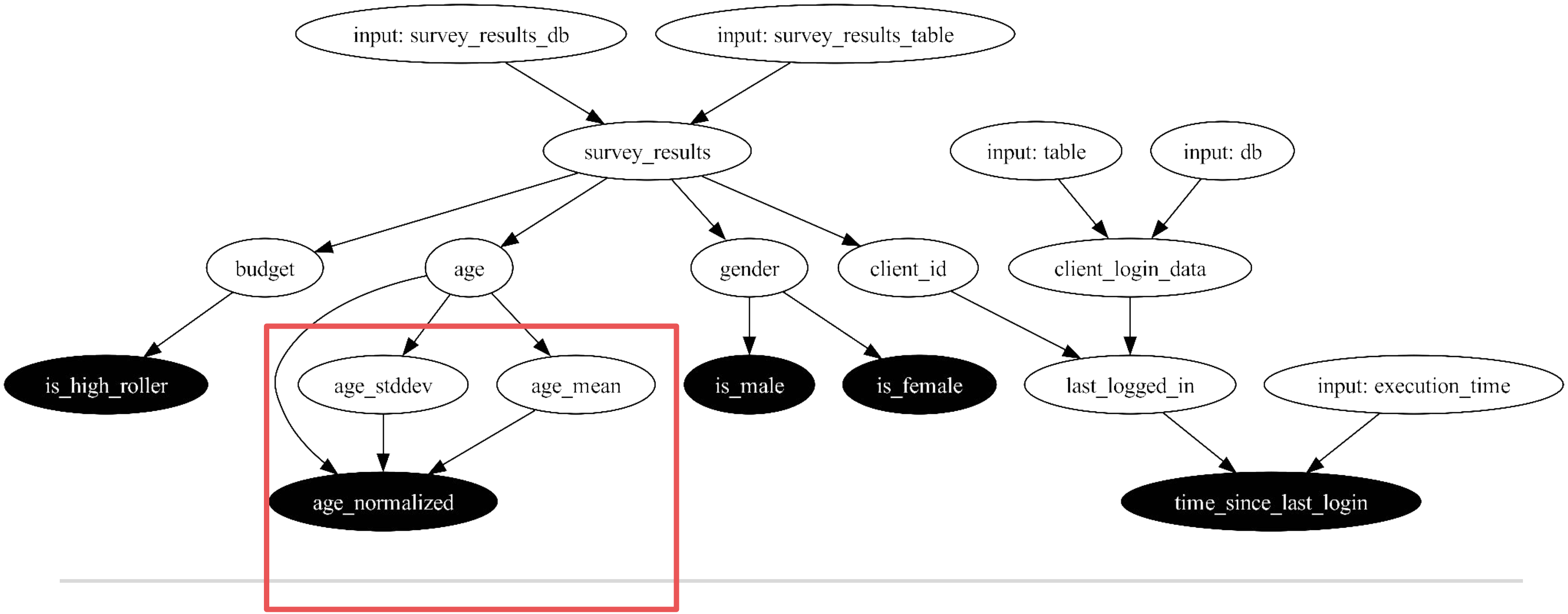


DAGWORKS

Aggregations

```python
def age_mean(age: pd.Series) -> float:
     return age.mean()

def age_stddev(age: pd.Series) -> float:
     return age.std()

def age_normalized(age: pd.Series, age_mean: float, age_stddev: float) -> pd.Series:
     return (age - age_mean)/age_stddev
```

Aggregations

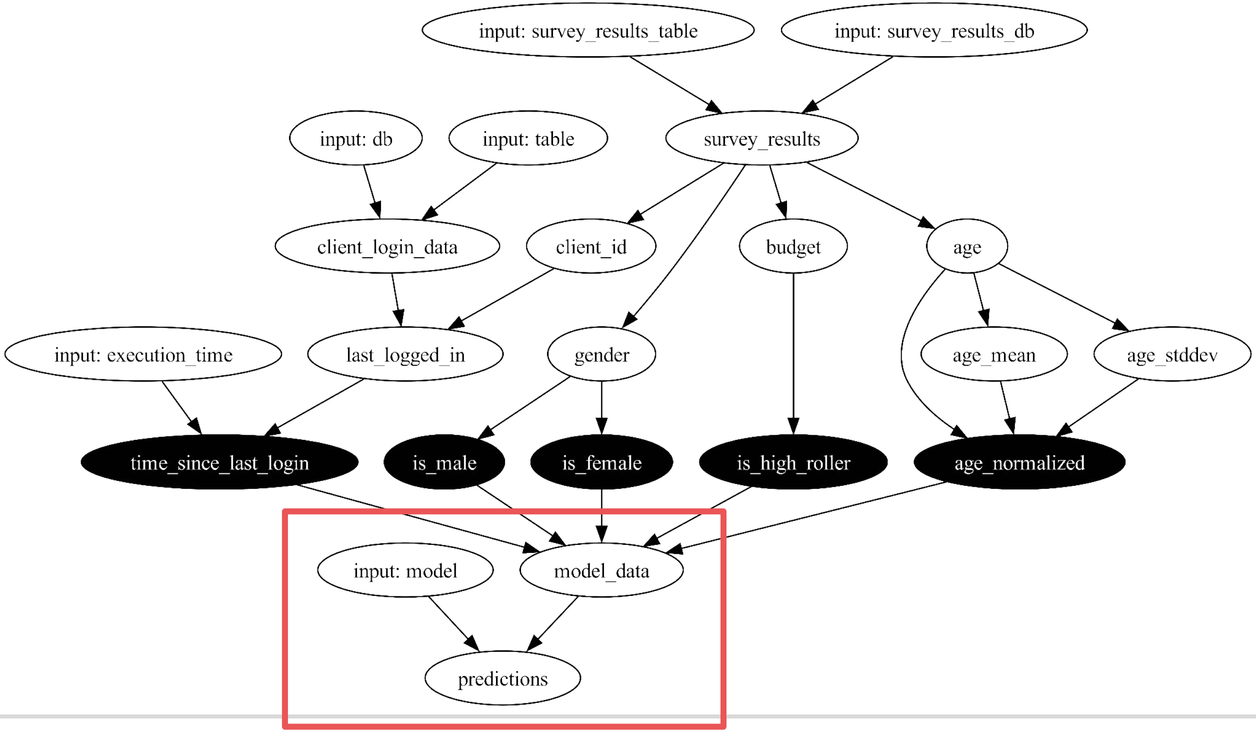


DAGWORKS

Inference

```python
def model_data(
    age_normalized: pd.Series,
    is_high_roller: pd.Series,
    is_male: pd.Series,
    is_female: pd.Series,
    time_since_last_login: pd.Series) -> pd.DataFrame:
    return pd.DataFrame(...)

def predictions(
    model: Model,
    model_data: pd.DataFrame) -> pd.Series:
    return model.predict(data)
```

Inference



DAGWORKS

Driver

```python
#etl.py

from project import load_data, map_features, join_features, agg_features, model
dr = driver.Driver(
    {},
    load_data, map_features, join_features, agg_features, model)

inputs = {
        "survey_results_table" : ...,
        "survey_results_db" : ...,
        "execution_time" : datetime.datetime.now(),
        "client_data_table" : ...,
        "client_data_db": ...,
        "model" : load_model(...)
    }
predictions = dr.execute(['predictions'], inputs=inputs)
```

The Agenda

The problem with feature engineering

The solution: *Hamilton*

Write once, run everywhere

- ↳ Batch
- ↳ **Streaming**

Additional benefits of Hamilton

OS progress/updates

DAGWORKS

Streaming features

Context

- ❏ Have service to give client login data
- ❏ Have stored aggregations from training
- ❏ Goal: “Near real time” == predict as soon as raw data is available

Changes required

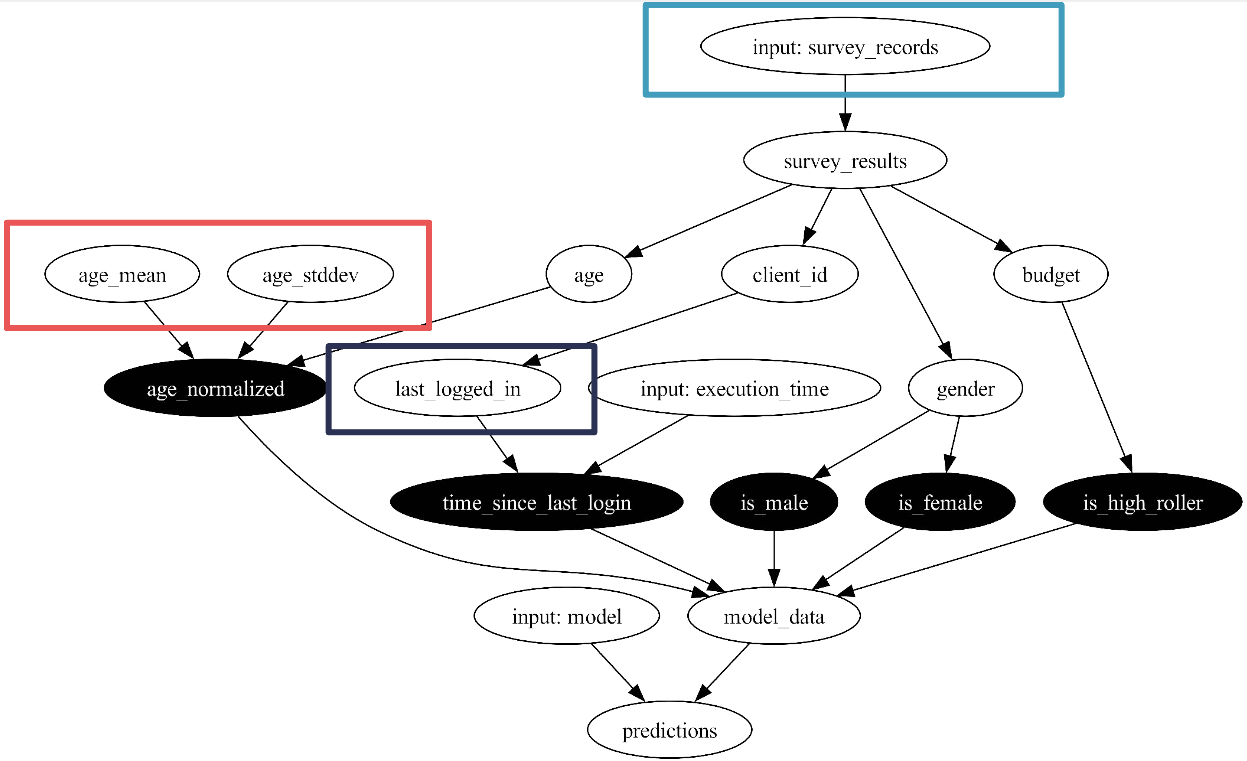
- ❏ No aggregation available
- ❏ Swap out external join with API call
- ❏ Single datums, not dataframes *[we treat them the same]*

Streaming features

@config.when swap out features you need to change:

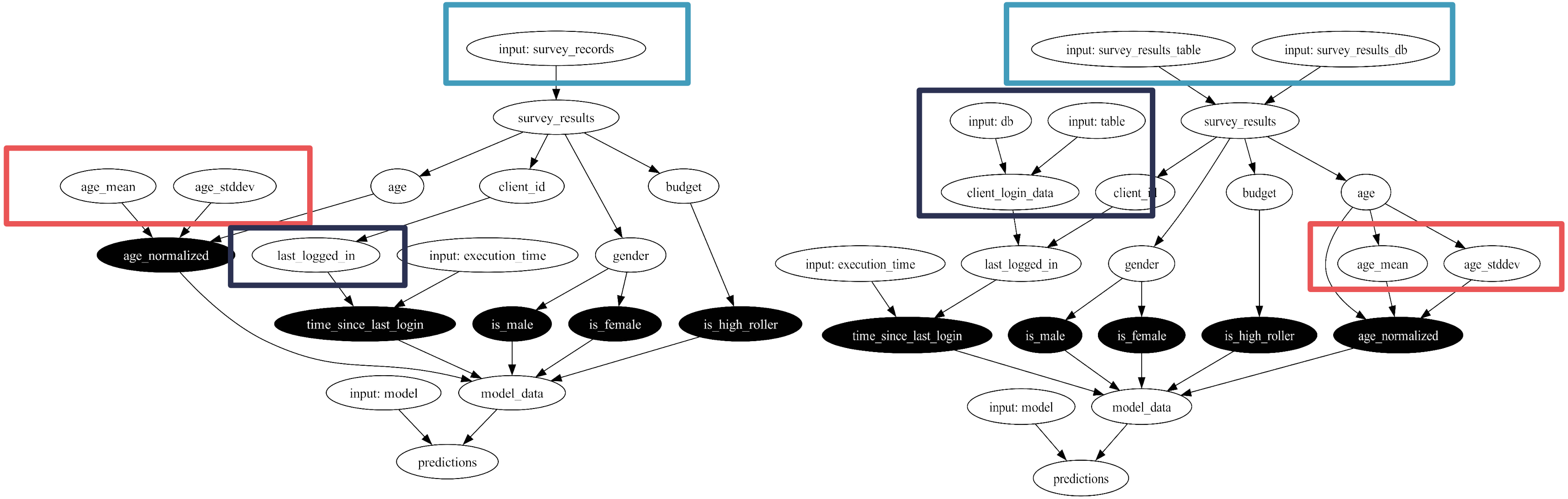
```python
@extract_columns('budget', 'age',  'gender', 'client_id')
@config.when(mode='streaming')
def survey_results__streaming(survey_records: list[dict]) -> pd.DataFrame:
    return pd.DataFrame.from_records(survey_records)

@config.when(mode='streaming')
def last_logged_in__streaming(client_id: pd.Series) -> pd.Series:
    return pd.Series(query_login_service(ids=client_id.values()))

@config.when(mode='streaming')
def age_mean__streaming() -> float:
     return query('age_mean')

@config.when(mode='streaming')
def age_stddev__streaming() -> float:
     return query('age_stddev')
```

Tying it together...



DAGWORKS

Tying it together...



Driver

```
# processor.py
from project import load_data, map_features, join_features,
    agg_features, model

config = {'mode' : 'streaming'}
dr = driver.Driver(config, load_data, map_features, join_features,
         agg_features, model)

def process_records(records: list[dict]) -> list[float]:
    inputs = {
        "records" : records,
        "execution_time" : datetime.datetime.now(),
        "model" : load_model(...)
    }
   return dr.execute(['predictions'], inputs=inputs).values
```

DAGWORKS

The Agenda

The problem with feature engineering
The solution: *Hamilton*
Write once, run everywhere
- ↳ Batch
- ↳ Streaming

Additional benefits of Hamilton
OS progress/updates

Portable FE code + ...

Hamilton lets you write transforms in python functions

These python functions provide everything you need:

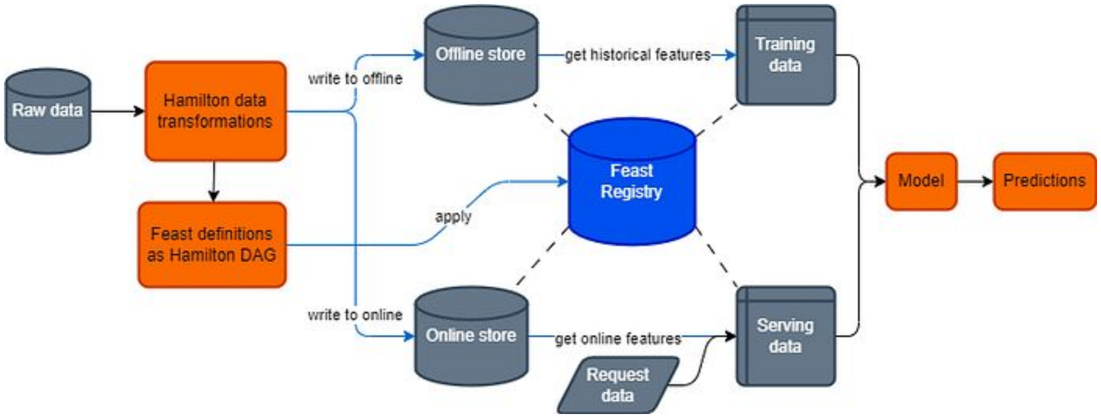
- ❏ **Unit testing**: *simple – plain python functions!*
- ❏ **Documentation**: *use the docstring*
- ❏ **Modularity**: *small pieces -> by definition*
- ❏ **Data catalogue**: *code = central feature definition store*
- ❏ **Debugging**: *execute functions individually + breakpoints*
- ❏ **Trustworthy data**: *validation included out of the box*

Integration with feature stores

Hamilton = transform layer

FS (Hopsworks, Feast, Tecton) = storage layer

[Feast Integration]



DAGWORKS

Broader Applications/Overall Stack

Hamilton improves code whenever python + data are involved

- ❑ LLM pipelines (RAG/fine-tuning)
- ❑ ML training pipelines
- ❑ DE pipelines (pandas, pyspark, polars, etc…)
- ❑ Complimentary with existing infrastructure

Airflow | dbt | prefect | etc…	Complementary
Langchain	Replaces
PySpark	Complementary
Kedro	Complementary + Replaces
Ray & Dask	Complementary
SWE Skills	Uplevels

DAGWORKS

The Agenda

The problem with feature engineering
The solution: *Hamilton*
Write once, run everywhere
- ↳ Batch
- ↳ Streaming
- ↳ Online

Additional benefits of Hamilton
OS progress/updates

DAGWORKS

OS Progress

Thriving community (110k+ downloads)

- ❏ Myriad of production users ->
- ❏ Growing set of core contributors
- ❏ Full company dedicated to building it!

Looking for

- ❏ Contributors (hacktoberfest!)
- ❏ Bug hunters
- ❏ User feedback

In Progress

Expressive APIs

- ❑ Flexible loading/materialization
- ❑ New high-power decorators
- ❑ <Your idea here!>

Execution

- ❑ Hamilton compile -> orchestration
- ❑ Snowpark integration
- ❑ <Your idea here!>

jmarvin90 commented 7 days ago

> I guess I'd be really curious to hear about your workflow with/without it. Not sure if you can share code or not, but I'd love to know what it would take to do your workflow with it, what it would take without it, and what the value of it would be to your day-to-day.

If we consider an example in which we are extracting a large number (dozens) of features from a dataset and then passing those features into a node which requires them all as input (e.g. to fit a model/make a prediction), we have a couple of friction points:

1. The function definition for the 'process' node gets a bit unwieldy when we have to specify a really large number of inputs as parameters;
2. We have to create a new feature extraction node **and** update the processing node definition for each new feature we add

e.g.

```
def extract_feature_1(col_a:pd.Series) -> pd.Series:
    return helpers._do_something(col_a)


# Dozens more feature node definitions
...


def extract_feature_100(col_zzz:pd.Series) -> pd.Series:
    return helpers._do_something_else(col_zzz)


# Really long definition!
def process_all_features(
    extract_feature_1:pd.Series,
    extract_feature_2:pd.Series,
    # Dozens more feature node references
    ...
    extract_feature_100:pd.Series
    ) -> object:
    return ml_model.do_some_maths(
        extract_feature_1:pd.Series,
        extract_feature_2:pd.Series,
        # Dozens more feature node references
        ...
        extract_feature_1000:pd.Series
    )
```

mattharrison commented 12 days ago

Hey there,

I'm curious to try out Hamilton on a multi-step pandas transform. I'm stuck though because the input CSV has spaces in the column names and I can't find any documentation for dealing with that. I figured there might be a decorator that helps with this but I can't see one.

For example, my input column is named "Lot Frontage" and I want the output to be named "lot_frontage".

mattharrison added the triage label 12 days ago

DAGWORKS

Give Hamilton a Try! We'd Love Your Feedback.

www.tryhamilton.dev

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

⭐ on github (https://github.com/dagworks-inc/hamilton)

☑️ create & vote on issues on github

📣 join us on on Slack

Blog post on feature engineering

Code to play with

Thank you!

Questions?

- https://twitter.com/elijahbenizzy
- https://www.linkedin.com/in/elijahbenizzy/
- https://github.com/dagworks-inc/hamilton
- elijah@dagworks.io
- linktr.ee/elijahbenizzy