Lightweight “Lineage” with Hamilton

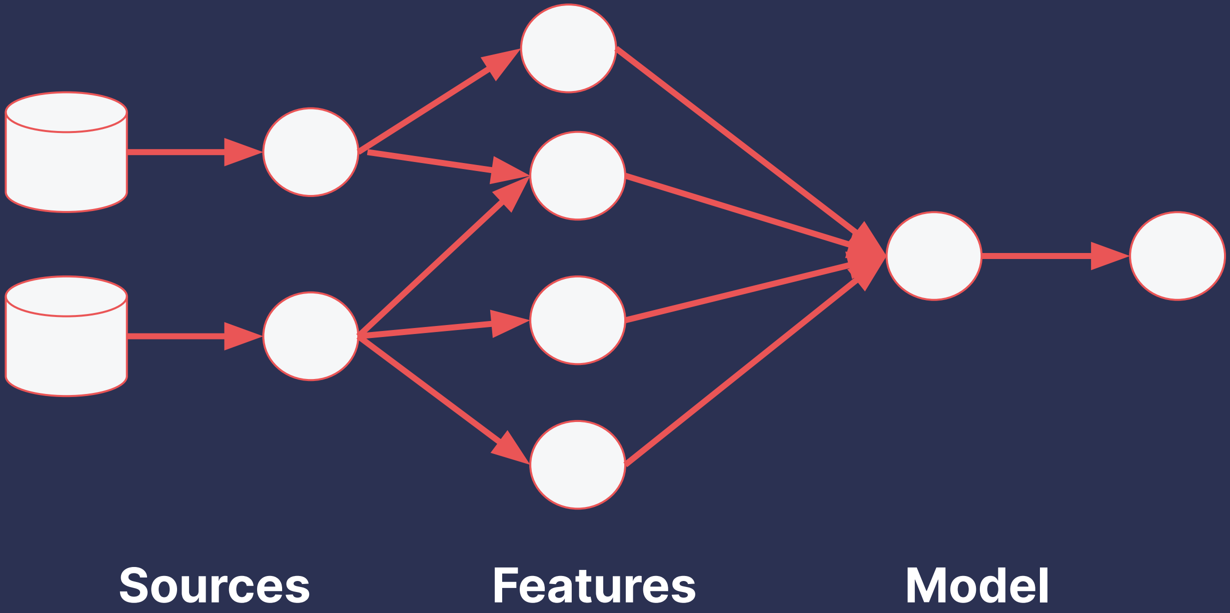
Stefan Krawczyk
CEO DAGWorks Inc.

DAGWORKS

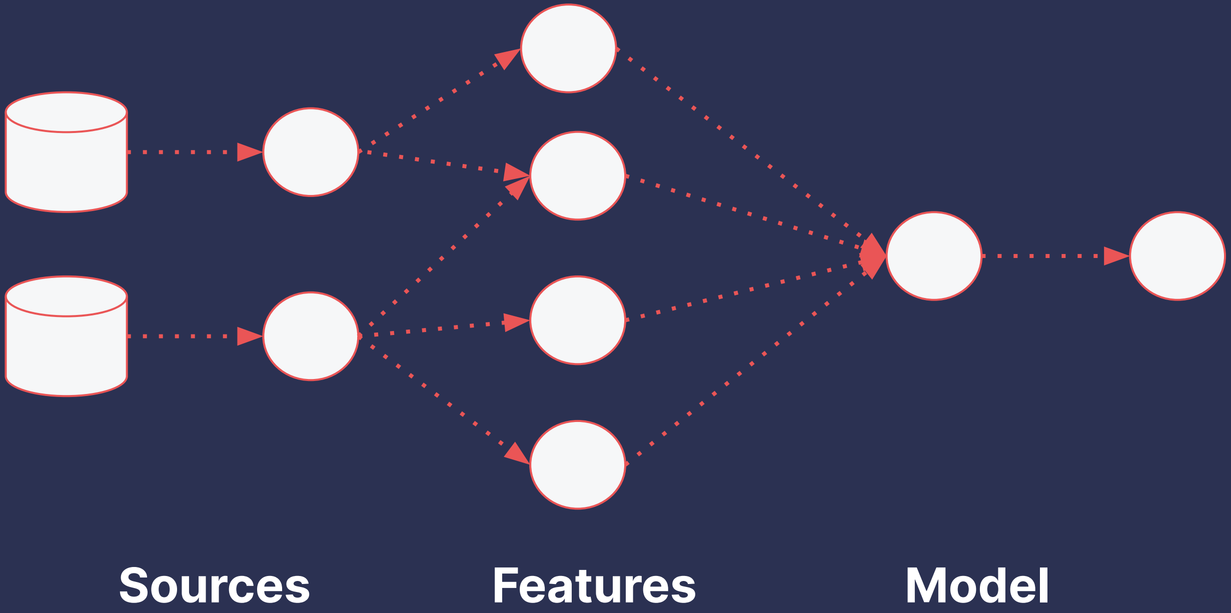
@ PyData

Seattle 2023

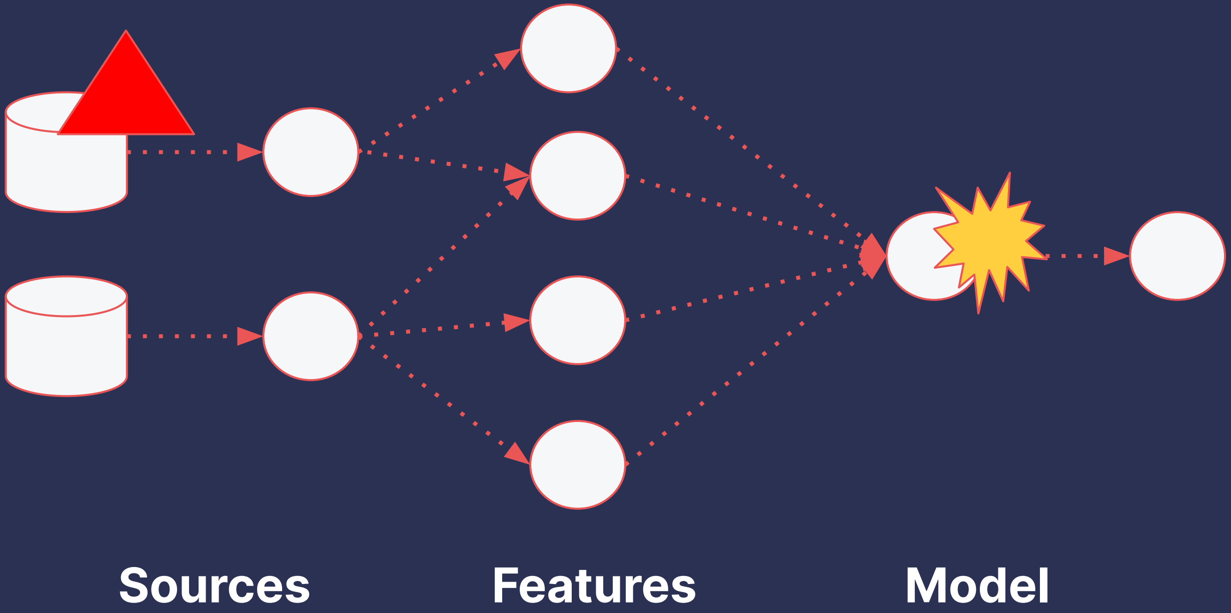
Context: Here’s a common pipeline/ETL/“data flow”



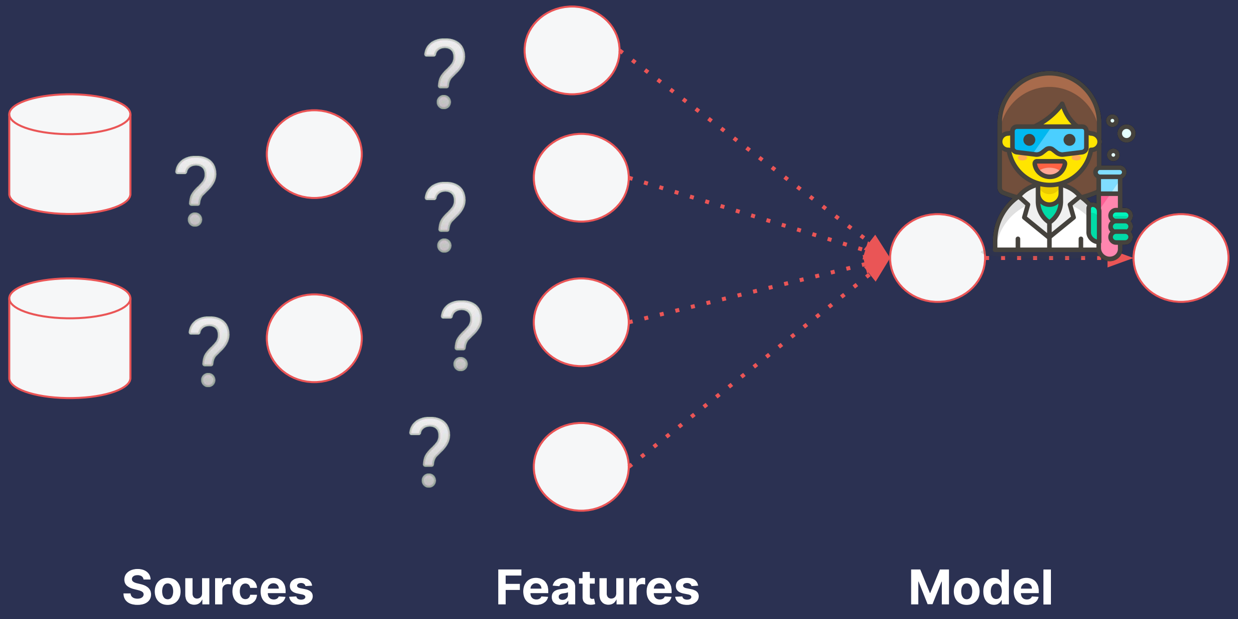
In reality it's more like



Problem: Ever had these issues?



Problem: Ever had these issues?



What is Hamilton?

micro-framework for defining dataflows

SWE best practices: ☑️ testing ☑️ documentation ☑️ modularity/reuse

```
pip install sf-hamilton [came from Stitch Fix]
```

www.tryhamilton.dev ← uses pyodide!

Hamilton: “a ha” moment

Table:

	spend	spend_zero_mean	spend_zero_mean_unit_variance
2023-01-01	10	-46	-1.173035
2023-01-02	10	-46	-1.173035
2023-01-03	20	-36	-0.918028
2023-01-04	40	-16	-0.408012
2023-01-05	40	-16	-0.408012

Idea: What if every output (column) corresponded to exactly one python fn?

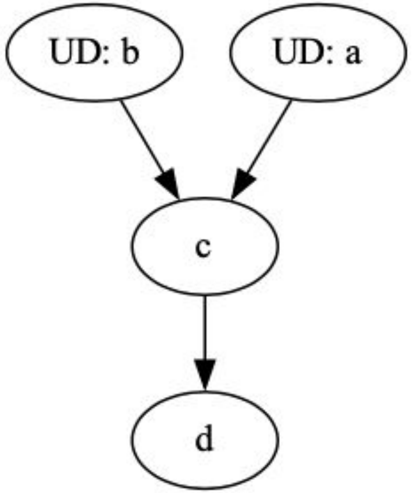
Addendum: What if you could determine the dependencies from the way that function was written?

```
def spend_zero_mean_unit_variance(
  spend_zero_mean: pd.Series, spend_std_dev: float
) -> pd.Series:
 """More docs would go here…"""
 return spend_zero_mean / spend_std_dev
```

Full Hello World

Functions

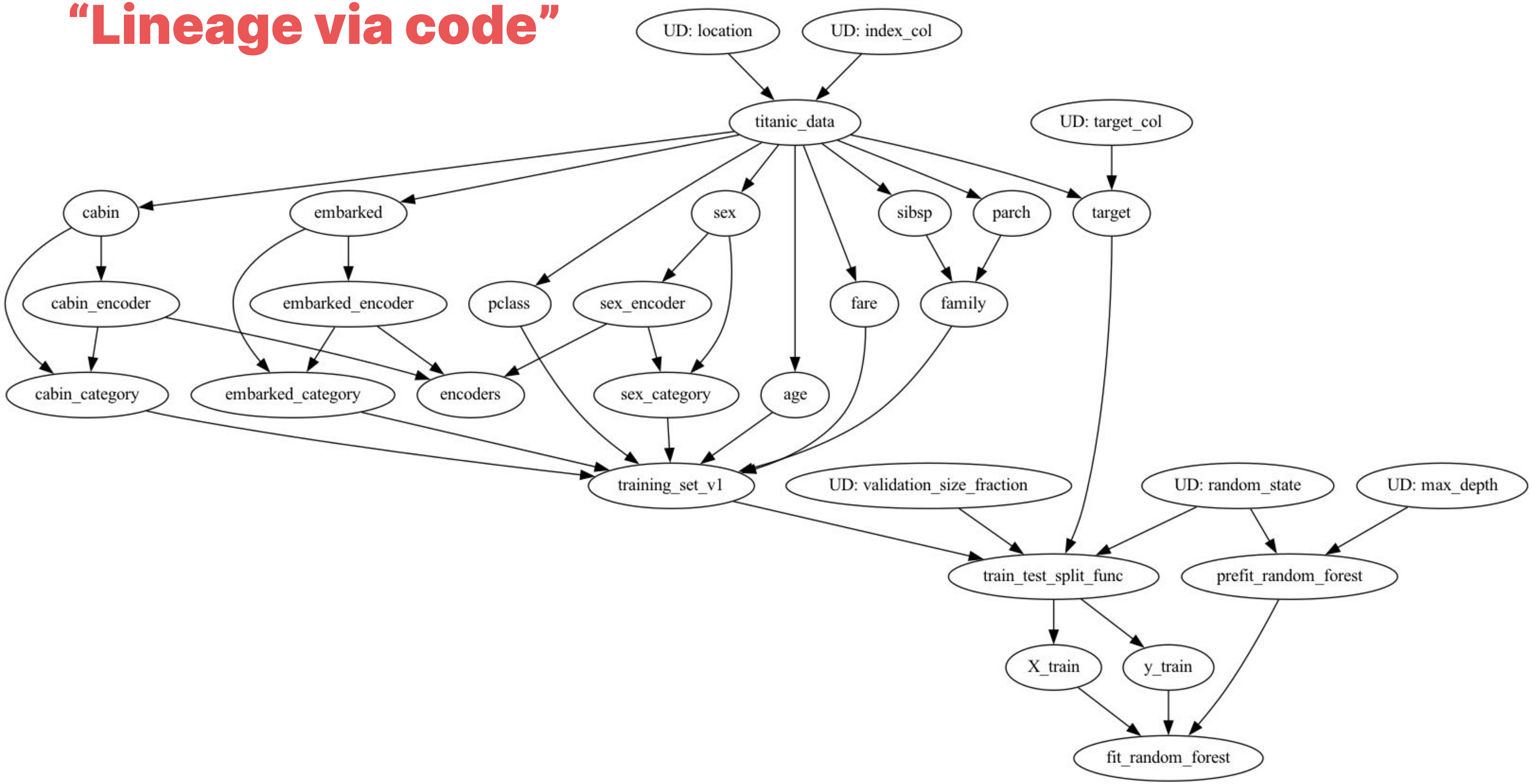
```python
# feature_logic.py
def c(a: pd.Series, b: pd.Series) -> pd.Series:
    """Sums a with b"""
    return a + b

def d(c: pd.Series) -> pd.Series:
    """Transforms C to ..."""
    new_column = _transform_logic(c)
    return new_column
```



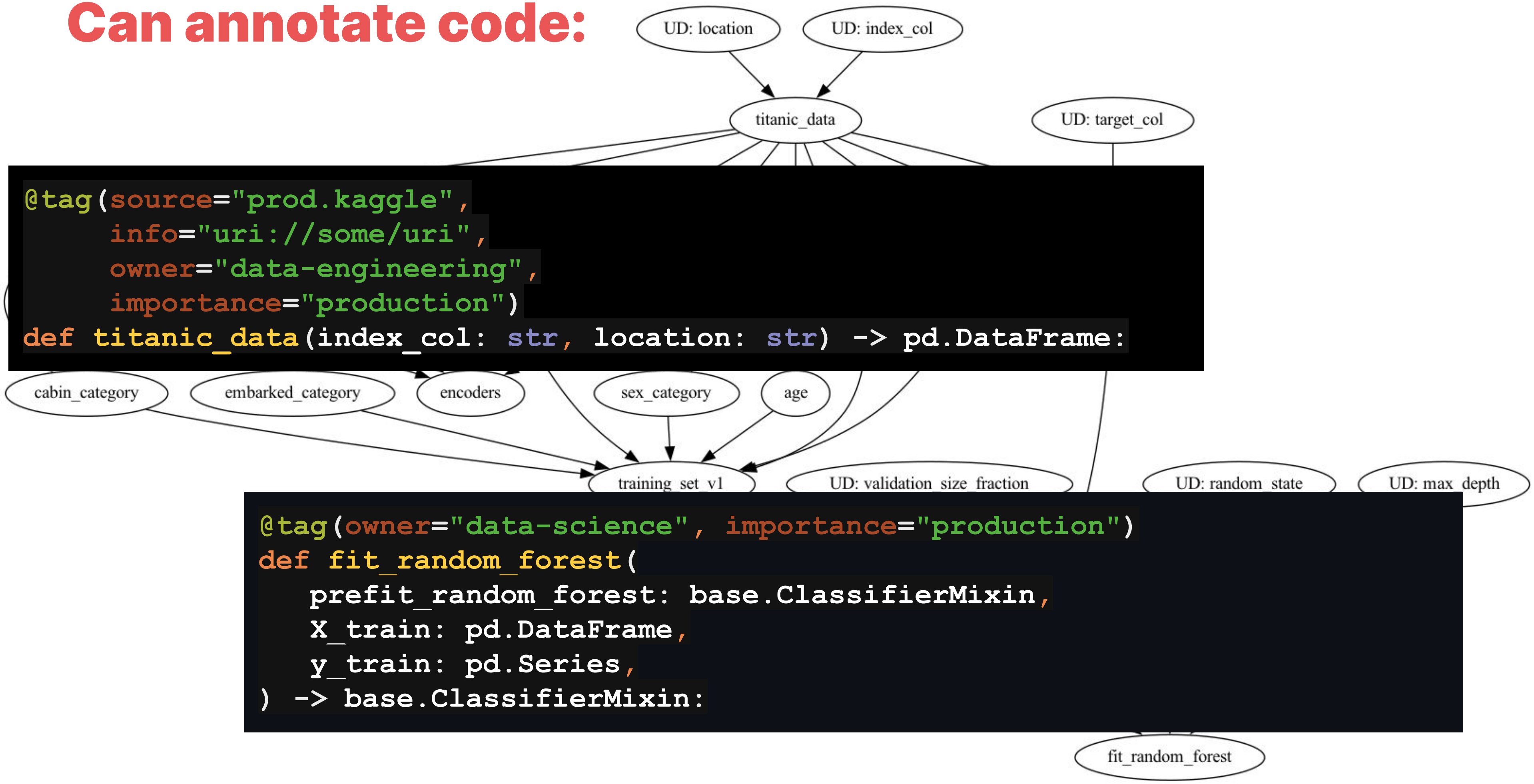
Driver says what/when to execute

```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)
```

“Lineage via code”



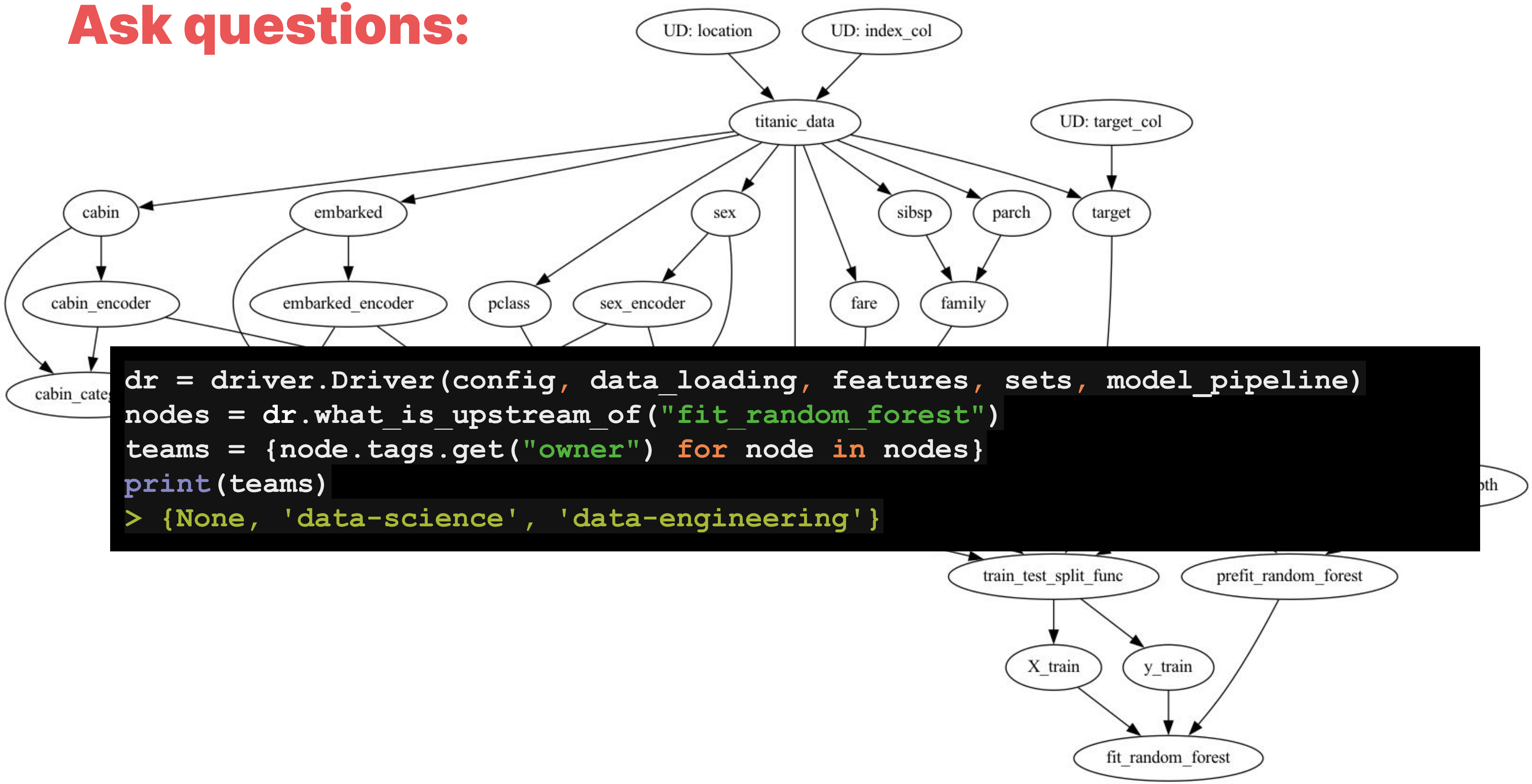
Can annotate code:



```python
@tag(source="prod.kaggle",
     info="uri://some/uri",
     owner="data-engineering",
     importance="production")
def titanic_data(index_col: str, location: str) -> pd.DataFrame:
```

```python
@tag(owner="data-science", importance="production")
def fit_random_forest(
   prefit_random_forest: base.ClassifierMixin,
   X_train: pd.DataFrame,
   y_train: pd.Series,
) -> base.ClassifierMixin:
```

Ask questions:



```
dr = driver.Driver(config, data_loading, features, sets, model_pipeline)
nodes = dr.what_is_upstream_of("fit_random_forest")
teams = {node.tags.get("owner") for node in nodes}
print(teams)
> {None, 'data-science', 'data-engineering'}
```

Recipe for a lightweight “lineage” store:

Lightweight lineage store just requires:

1. Define data flow in code.
2. Add metadata to code.
3. Commit to version control system.

When you execute to create an artifact:

1. Store commit hash.
2. And what you requested to be run.

Can now answer questions:

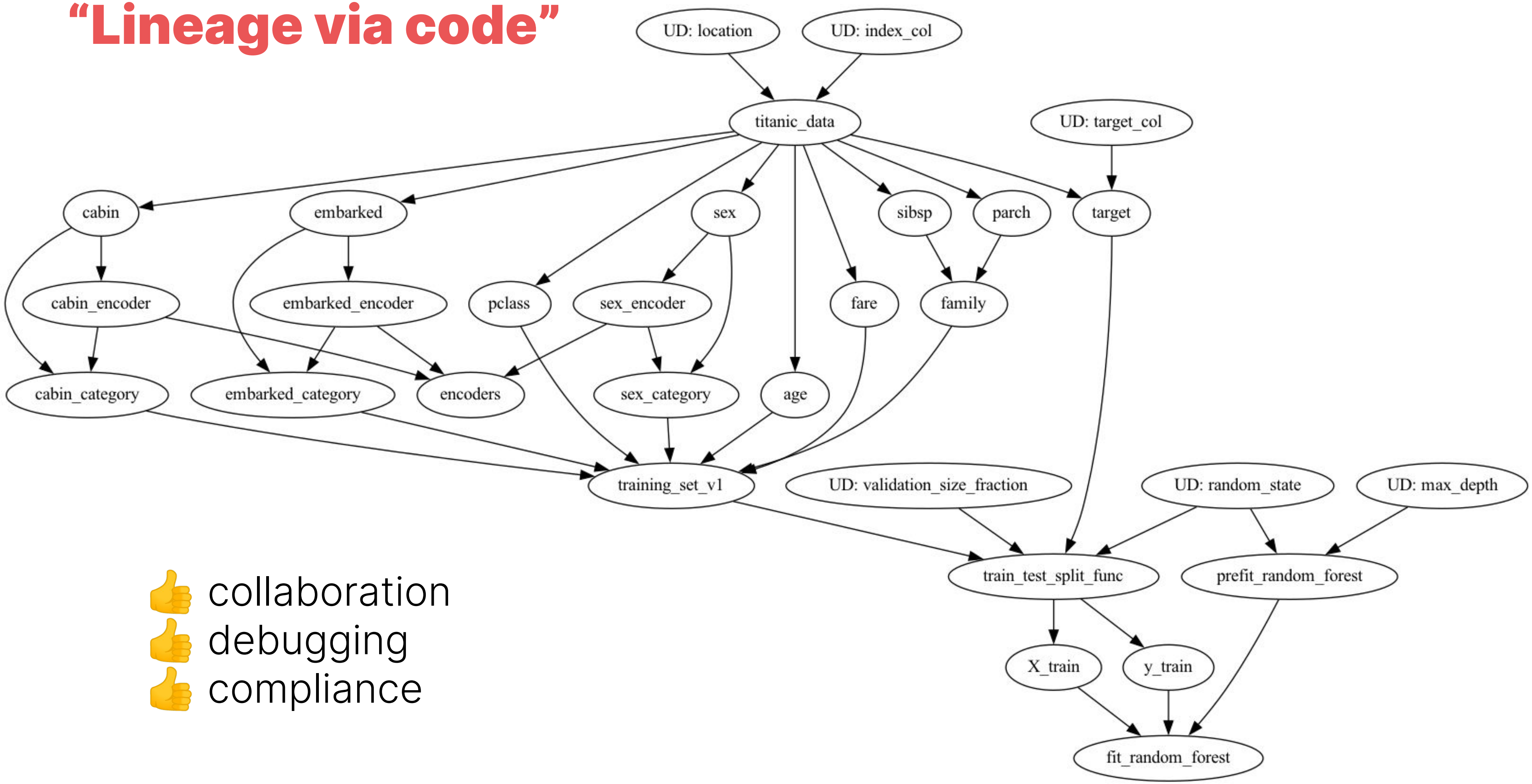
Using **e.g. with git sha** can:

1. Diff code to understand differences (obviously)
2. Go back in time to recreate the world.

Using the DAG:

1. If I change this “function” who/what will I impact?
2. What are my “production” features?
3. Where is birth date used?
4. etc.

“Lineage via code”



👍 collaboration
👍 debugging
👍 compliance

Hamilton: Lightweight “Lineage”

TL;DR:

1. Write functions - get lineage as code.
2. Add annotations - build something you can query!

Star Hamilton - ⭐ https://github.com/dagworks-inc/hamilton 👈

(this example will be written up this soon)

Thanks! Q&A

Hamilton:

www.tryhamilton.dev

Hamilton (@hamilton_os) / Twitter

⭐ https://github.com/dagworks-inc/hamilton 👈

📚 https://hamilton.readthedocs.org

Me: stefan@dagworks.io

https://twitter.com/stefkrawczyk

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

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 @ SF models business mechanics.
2. We’re doing some basic graph theory.

$$H_{operator} = \underbrace{\frac{-\hbar^2}{2m}\frac{\partial^2}{\partial x^2}}_{\text{Operator associated with kinetic energy}} + \underbrace{V(x)}_{\text{Potential energy}}$$

apropos Hamilton