Hamilton Global User Group
April 2024 Meetup

What is Hamilton?
Hamilton helps data scientists and engineers define testable, modular, self-documenting dataflows, that encode lineage and metadata. Runs and scales everywhere python does.

Icebreaker: Name and what you're using Hamilton for/looking for.

Agenda

1. Community Spotlight
2. The “news”
3. Deep Dive
4. Open 🎤

Community Spotlight:
🔎 "**Modular dataflows and experiment management for ML evaluation**"
by Thierry Jean.

DAGWORKS

📰 The “News”

Some Recent Features Released

FunctionInputOutputTypeChecker	
SlackNotifier	🙏 Swapnil Dewalkar
@ray_remote_options	🙏 Fran Boon
Polars Lazyframe support	🙏 Tom Barber
Polars DB I/O	🙏 Swapnil Dewalkar
Polars Spreadsheet I/O	🙏 Swapnil Dewalkar
Pandas SPSS Reader	🙏 Swapnil Dewalkar
YAML I/O	🙏 Walber Moreira
...	

Some Recent Features Released

- FunctionInputOutputTypeChecker

```
from hamilton import base, driver, lifecycle

dr = (
    driver.Builder()
    .with_config({})
    .with_modules(my_functions)
    .with_adapters(
        # this is a strict type checker for the input and output of each function.
        lifecycle.FunctionInputOutputTypeChecker(),
        # this will make execute return a pandas dataframe as a result
        base.PandasDataFrameResult(),
    )
    .build()
)
```

Some Recent Features Released

- FunctionInputOutputTypeChecker

```
from hamilton import base, driver, lifecycle

dr
                                   input and output of each function.
        lifecycle.FunctionInputOutputTypeChecker(),
        # this will make execute return a pandas dataframe as a result
        base.PandasDataFrameResult(),
    )
    .build()
)
```

```
def a(input: pd.Series) -> pd.Series:
    return input.values()

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

Some Recent Features Released

- SlackNotifier

🙏 Swapnil Dewalkar

```
from hamilton import driver
from hamilton.plugins.h_slack import SlackNotifier

import some_module

api_key = "YOUR_API_KEY"
channel = "YOUR_CHANNEL"
dr = (
    driver.Builder()
    .with_modules(some_module)
    .with_adapters(SlackNotifier(api_key=api_key, channel=channel))
    .build()
)
```

Some Recent Features Released

- @ray_remote_options 🙏 Fran Boon

```
@ray_remote_options(
      num_gpus=1,
      resources={"my_custom_resource": 1},
)
def example() -> pd.DataFrame:
      ...
```

Some Recent Features Released

Polars Lazyframe support	🙏 Tom Barber
Polars DB I/O	🙏 Swapnil Dewalkar
Polars Spreadsheet I/O	🙏 Swapnil Dewalkar
Pandas SPSS Reader	🙏 Swapnil Dewalkar
YAML I/O	🙏 Walber Moreira

```
All pushed some form of data saver & data loader!
```

Some Documentation & Example Updates

- Updated Parallelism documentation (guide)
- Pandas (example) 🙏 Nicolas Huray
- Ibis (example) 🙏 Thierry Jean
- dlt (example) 🙏 Thierry Jean
- AWS: (integration guide) 🙏 Konstantin Tyapochkin
    - Lambda
    - Glue
    - Sagemaker
- Document chunking for RAG (example)
- ChatGPT + DALLE telephone game (example)

Roadmap: 🚀 Hamilton UI & More!

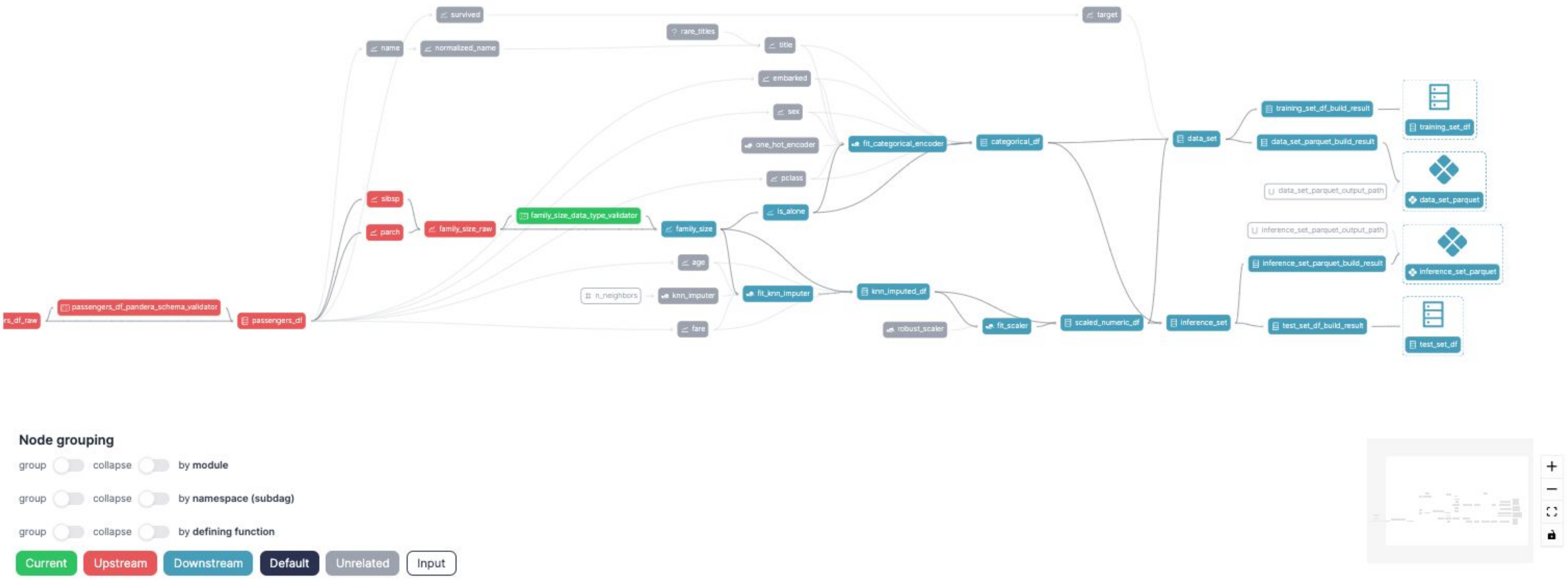
Open sourcing what we've been building with DAGWorks Inc.

Key features:

- Visualize
- Version
- Catalog
- Telemetry

Looking for a few from the community to ensure it all works!

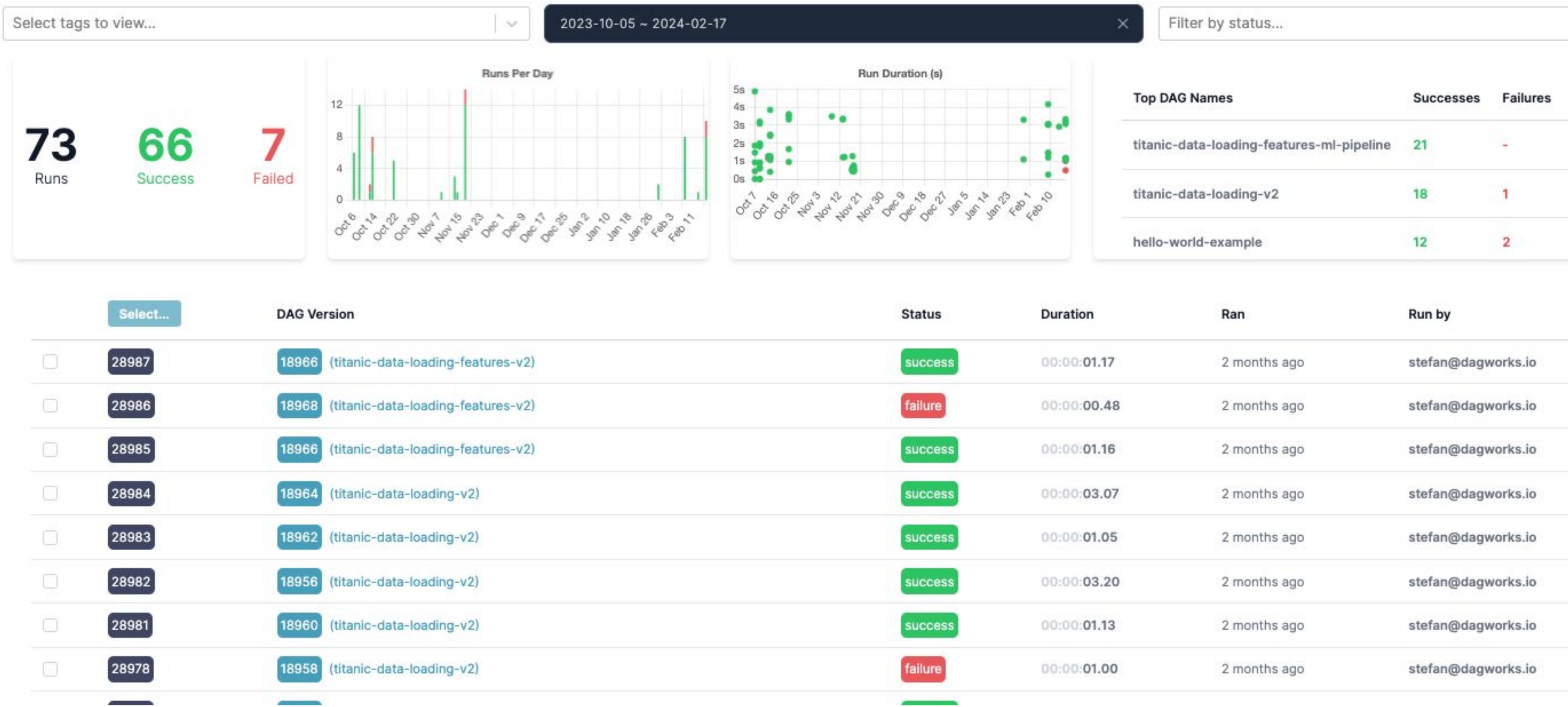
Roadmap: 🚀 Hamilton UI & More!



Roadmap: 🚀 Hamilton UI & More!

		Code	Description	Tags
transform	age	passengers_df	Function to take in a raw dataframe, check t…	module data_loader
transform	avg_3wk_spend	avg_3wk_spend	Rolling 3 day average spend.	module hw_funcs
transform	categorical_df	categorical_df	This creates the dataframe of categorical fe…	module features
input	categorical_encoder_path	fit_categorical_encoder		
data saver	cf_matrix_test			hamilton.data_saver true
data saver	cf_matrix_train			hamilton.data_saver true
transform	cm_test	cm_test		module ml_pipeline
transform	cm_test_display	cm_test_display		module ml_pipeline
transform	cm_train	cm_train		module ml_pipeline
transform	cm_train_display	cm_train_display		module ml_pipeline
transform	data_set	data_set	This function creates our dataset. Following…	module features
data saver	data_set_parquet			hamilton.data_saver true

Roadmap: 🚀 Hamilton UI & More!



28986 titanic-data-loading-features-v2 ()

28987 titanic-data-loading-features-v2 ()

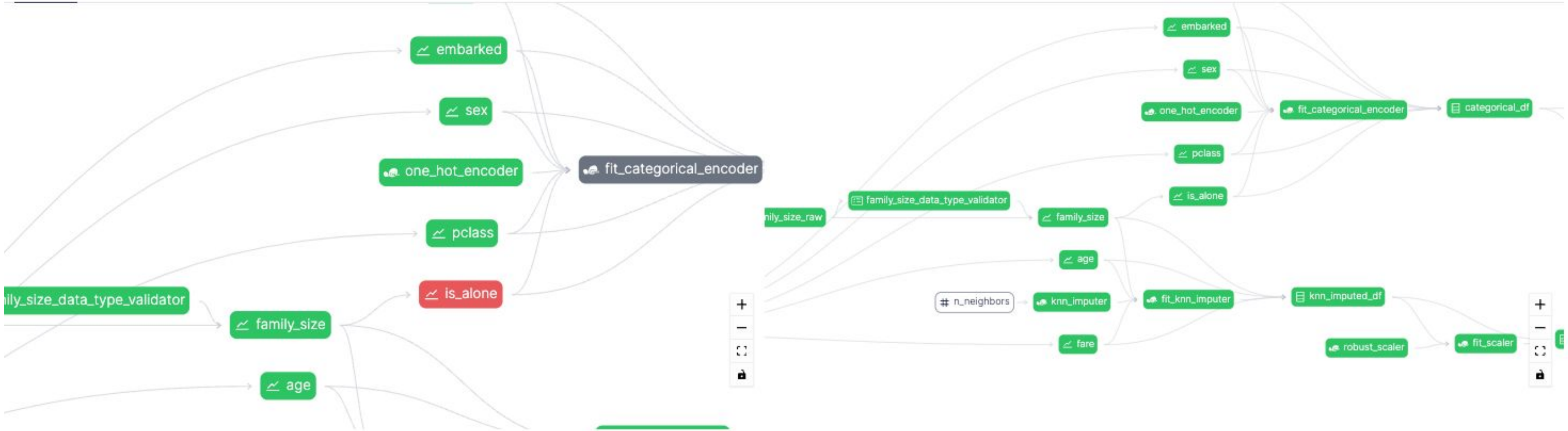
26 nodes 28986 **40** nodes 28987

0.48 seconds 28986 **1.17** seconds 28987

Tags	28986	28987
env	local	local
where	notebook	notebook

Outputs 28986 (6) 28987 (6) data_set data_set_parquet inference_set inference_set_parquet test_set_df training_set_df

DAG View Waterfall

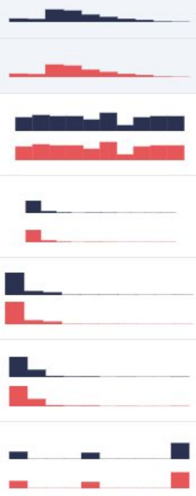


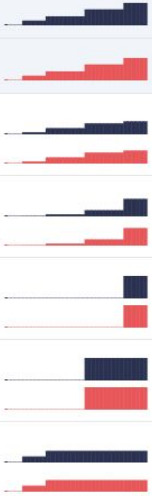
	Runs	Node	Type	Properties	State	Duration	
⌄	28986 28987	robust_scaler	sklearn.preprocessing._data.RobustScaler		success	0.001 ± 0.001s	△ diff
⌄	28986 28987	knn_imputer	sklearn.impute._knn.KNNImputer		success	0.000 ± 0.000s	△ diff
⌄	28986 28987	raw_passengers_df	pandas.core.frame.DataFrame		success	0.054 ± 0.001s	△ diff
⌄	28986 28987	passengers_df_raw	pandas.core.frame.DataFrame		success	0.032 ± 0.000s	△ diff
⌄	28986 28987	passengers_df_pandera_schema_validator	hamilton.data_quality.base.ValidationResult		success	0.028 ± 0.001s	△ diff
⌄	28986 28987	passengers_df	pandas.core.frame.DataFrame		success	0.029 ± 0.001s	△ diff

Output | Errors | Code | Upstream | Downstream | 28986 × 28987 ×

Numeric Columns

	column	runs	type	count	missing	mean ± std	range	histogram	quantiles
^	age								
		28986	float64	1307	263	29.84±14.39	[0.167, 80]		
		28987	float64	1307	263	29.84±14.39	[0.167, 80]		
v	body	28986 28987	float64	1307	1186.00		[1, 328]		
v	fare	28986 28987	float64	1307	1.00		[0, 512.329]		
v	parch	28986 28987	int64	1307	-		[0, 9]		
v	sibsp	28986 28987	int64	1307	-		[0, 8]		
v	pclass	28986 28987	int64	1307	-		[1, 3]		





Categorical Columns

	column	runs	type	count	missing	unique	top value
v	sex	28986 28987	category	1307	-		
v	embarked	28986 28987	category	1307	-		
v	survived	28986 28987	category	1307	-		

Deep Dive:
Data Savers / Loaders
a.k.a. Materializers

DAGWORKS

https://hamilton.dagworks.io/en/latest/concepts/materialization/
https://blog.dagworks.io/p/separate-data-io-from-transformation
https://blog.dagworks.io/p/enterprise-ready-data-pipelines-with
https://blog.dagworks.io/p/from-dev-to-prod-a-ml-pipeline-reference

Motivation: every dataflow reads and writes data

With Hamilton you:

1. Write functions
2. Functions are organized into modules.

But, depending on how you approach loading & saving data it can:

(a) Couple you to infrastructure/platform concerns
(b) Making your code less portable & maintainable

Data Saving & Loading from first principles

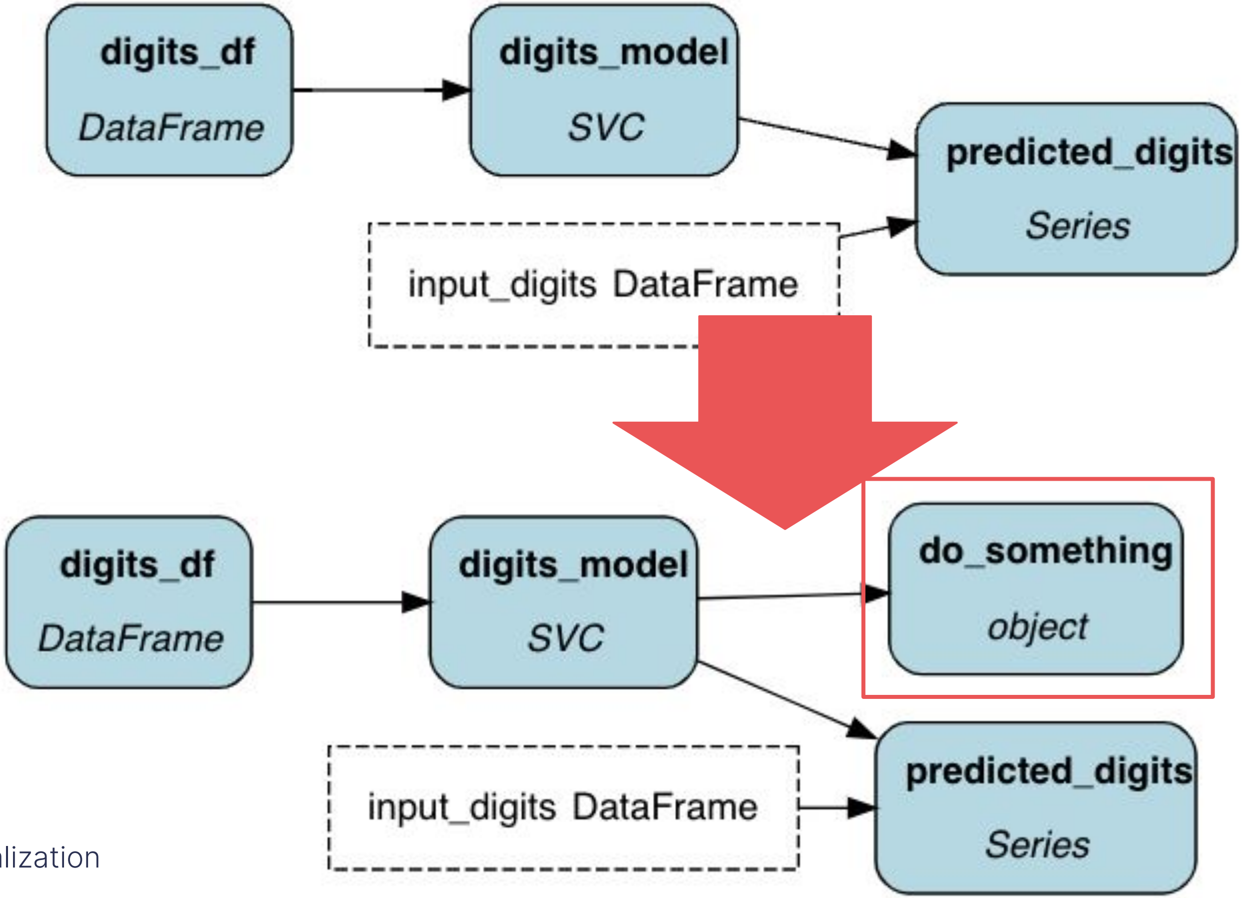
With Hamilton

Data Saving & Loading from first principles

With Hamilton

Watch for `.execute()` **versus** `.materialize()`

Mental Model: Hamilton/DAGs

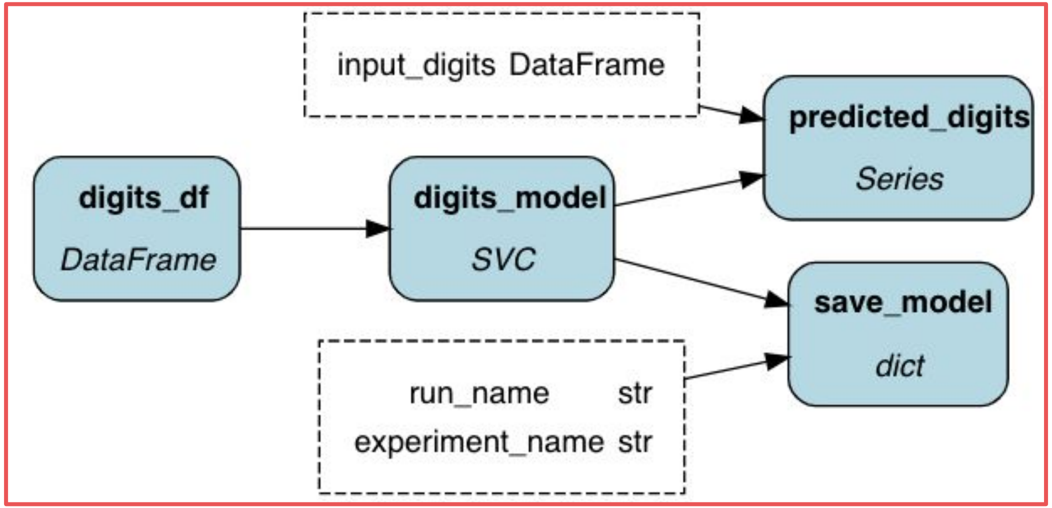


Data Saving with Hamilton

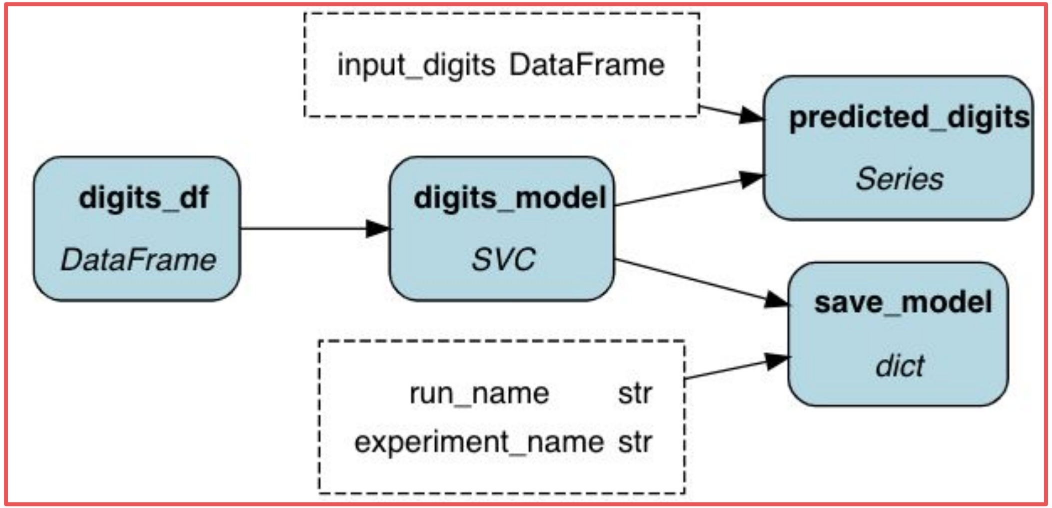
Three general approaches:

- Write a function in the DAG [`.execute()`]
- Do it outside of the DAG [`.execute()`]
- Use “materializers” and kind of do both [`.materialize()`, `@save_to + .execute()`]

Approach 1: Encode within Hamilton

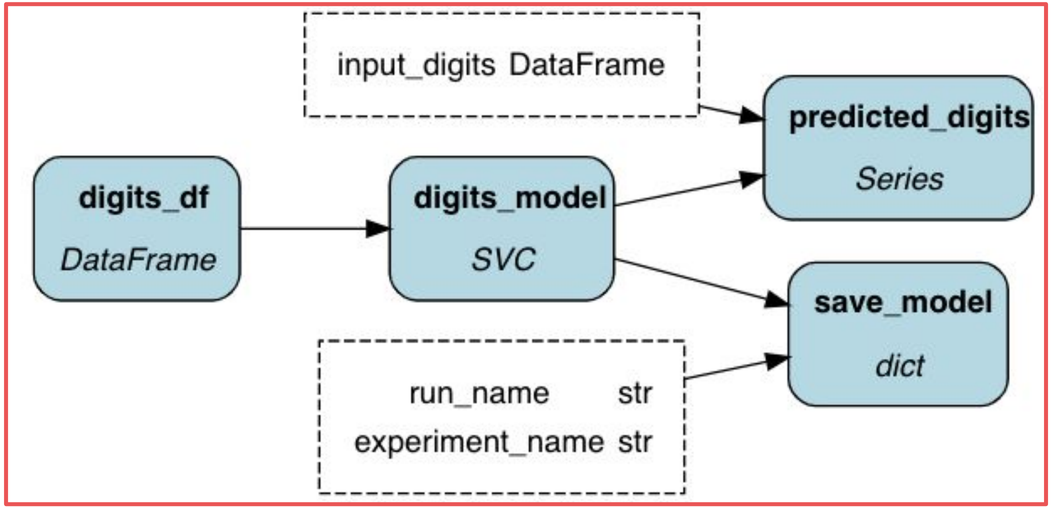


Approach 1: Encode within Hamilton




```python
def save_model(digits_model: svm.SVC,
               experiment_name: str,
               run_name: str) -> dict:
    """Saves a model to MLFlow"""
    mlflow.set_experiment(experiment_name)
    with mlfow.start_run(run_name=run_name) as run:
        model_info = mlflow.log_model(digits_model)
        # log more things, etc
    return {"model_info": model_info.__dict__}
```

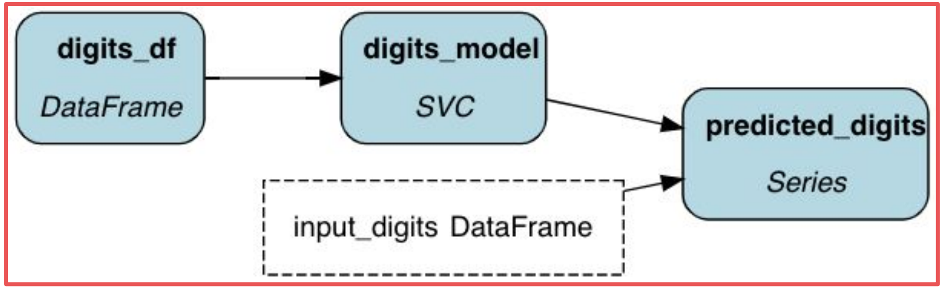
Approach 1: Encode within Hamilton




```
.execute(
 ["save_model"])
```

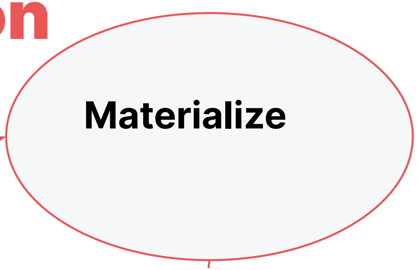
```python
def save_model(digits_model: svm.SVC,
               experiment_name: str,
               run_name: str) -> dict:
    """Saves a model to MLFlow"""
    mlflow.set_experiment(experiment_name)
    with mlfow.start_run(run_name=run_name) as run:
        model_info = mlflow.log_model(digits_model)
        # log more things, etc
    return {"model_info": model_info.__dict__}
```

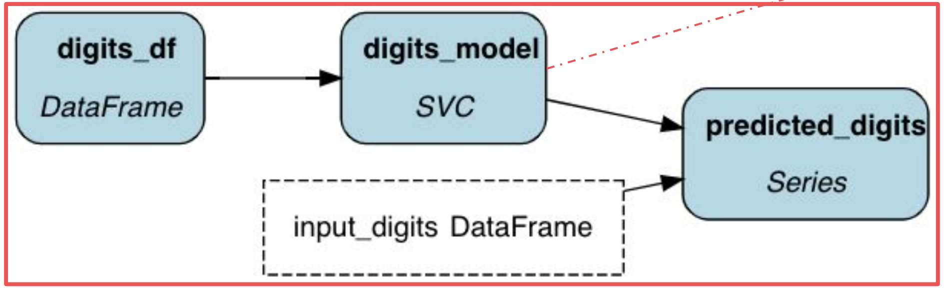
Approach 2: Outside of Hamilton




```
dr = driver.Driver({}, simple_pipeline)
result = dr.execute(["digits_model", "..."],
 inputs={"input_digits": load_some_digits().sample(5)}
)
```

Approach 2: Outside of Hamilton



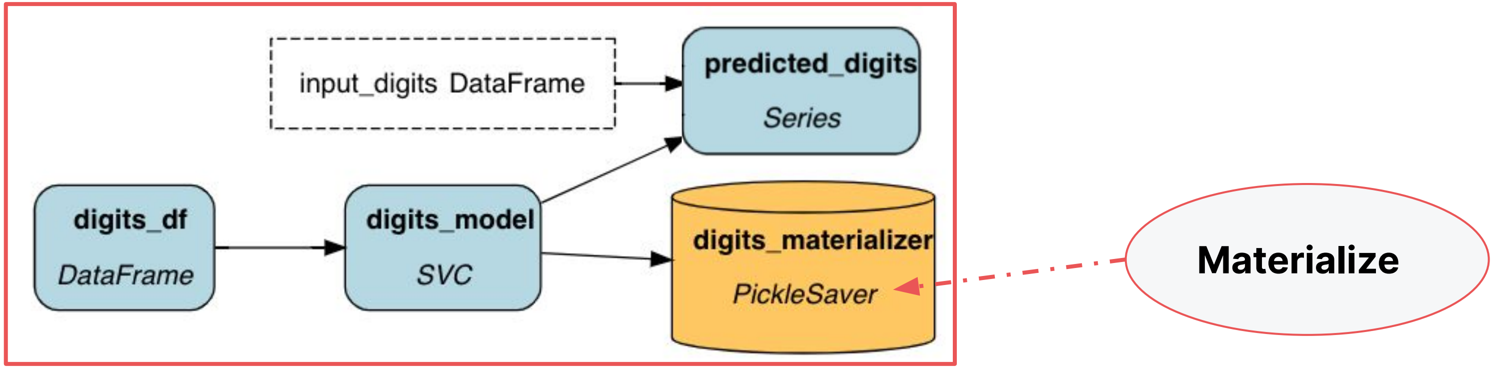


In Hamilton

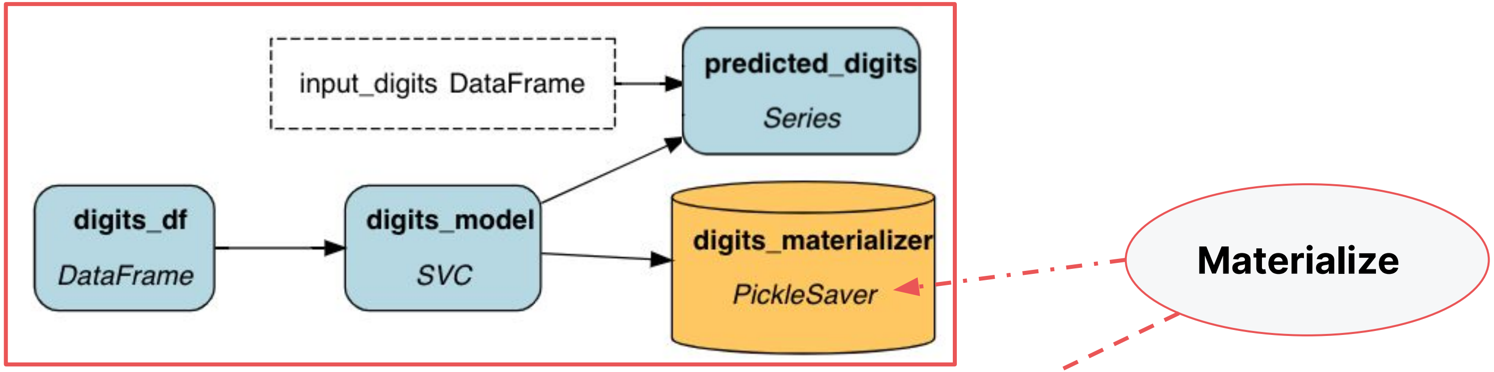
Outside Hamilton

```
dr = driver.Driver({}, simple_pipeline)
result = dr.execute(["digits_model", "..."],
 inputs={"input_digits": load_some_digits().sample(5)}
)
with mlflow.start_run(run_name=...) as run:
    # log model and stuff here.
    run.log_model(result[...])
```

Approach 3: Inject a “materializer” into the DAG



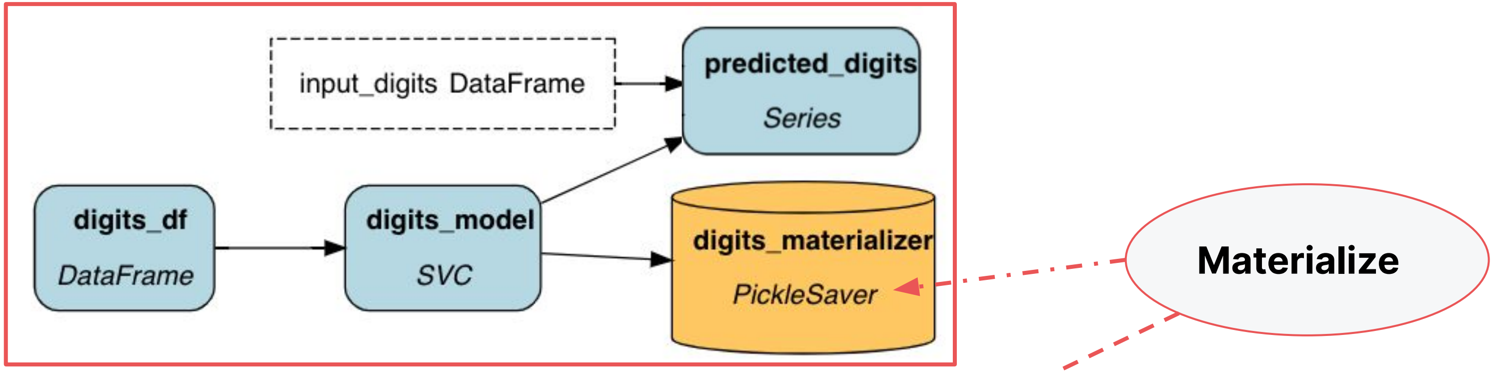
Approach 3: Inject a “materializer” into the DAG




```python
from hamilton.io.materialization import to

results = dr.materialize(
    to.pickle(
        id="digits_materializer",
        dependencies=[simple_pipeline.digits_model],
        path="model.pkl"
    ),
    inputs={
        "input_digits": load_some_digits().sample(5)
    }
)
```

Approach 3: Inject a “materializer” into the DAG



.materialize() →

```
from hamilton.io.materialization import to

results = dr.materialize(
    to.pickle(
        id="digits_materializer",
        dependencies=[simple_pipeline.digits_model],
        path="model.pkl"
    ),
    inputs={
        "input_digits": load_some_digits().sample(5)
    }
)
```

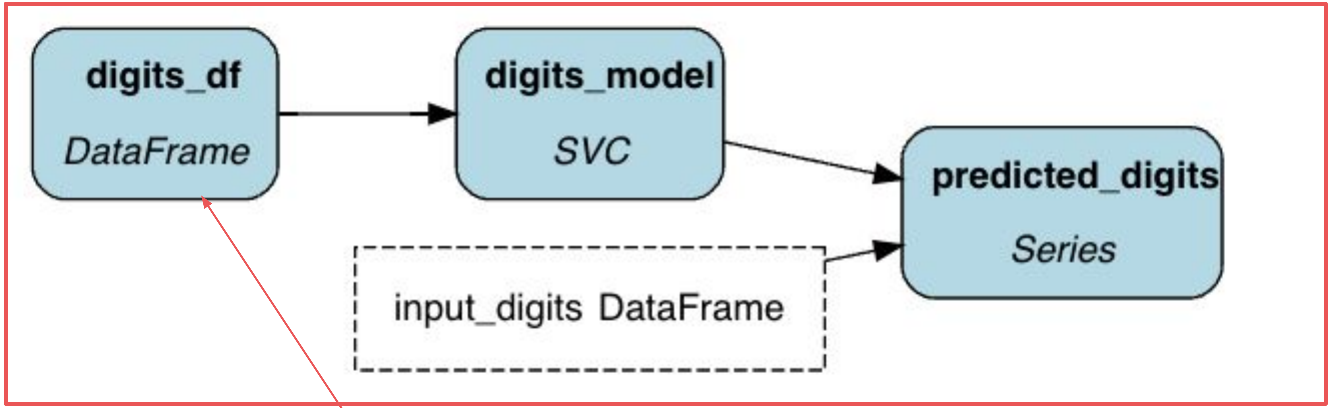
Note: @save_to.pickle decorator is equivalent.

What about *data loading?*

Three general approaches:

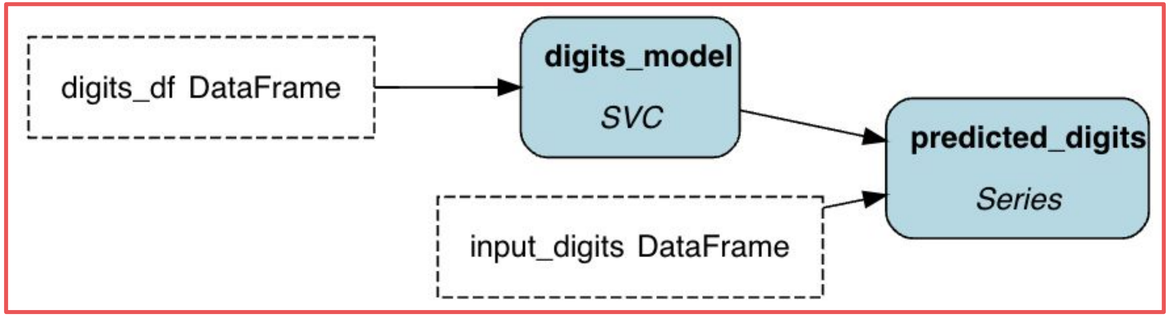
- Write a function in the DAG [`.execute()`]
- Do it outside of the DAG [`.execute(..., inputs=)`]
- Use "materializers" and kind of do both [`.materialize(), @load_from + .execute()`]

Approach 1: Encode within Hamilton




```python
def digits_df(path: str) -> pd.DataFrame:
    """Loads a digits DF"""
    df = pd.read_csv(path)
    .. # some other transforms.
    return df
```

Approach 2: Outside of Hamilton



Outside Hamilton

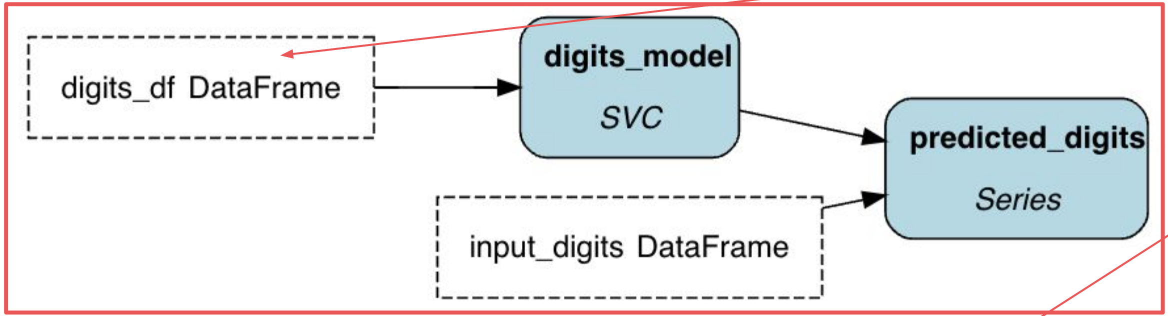
```
input_df = load_some_digits(path, ...)
```

In Hamilton

```
dr = driver.Driver({}, simple_pipeline)
result = dr.execute(["digits_model", "..."],
 inputs={"digits_df": input_df}
)
```

Approach 2: Outside of Hamilton

Materialize



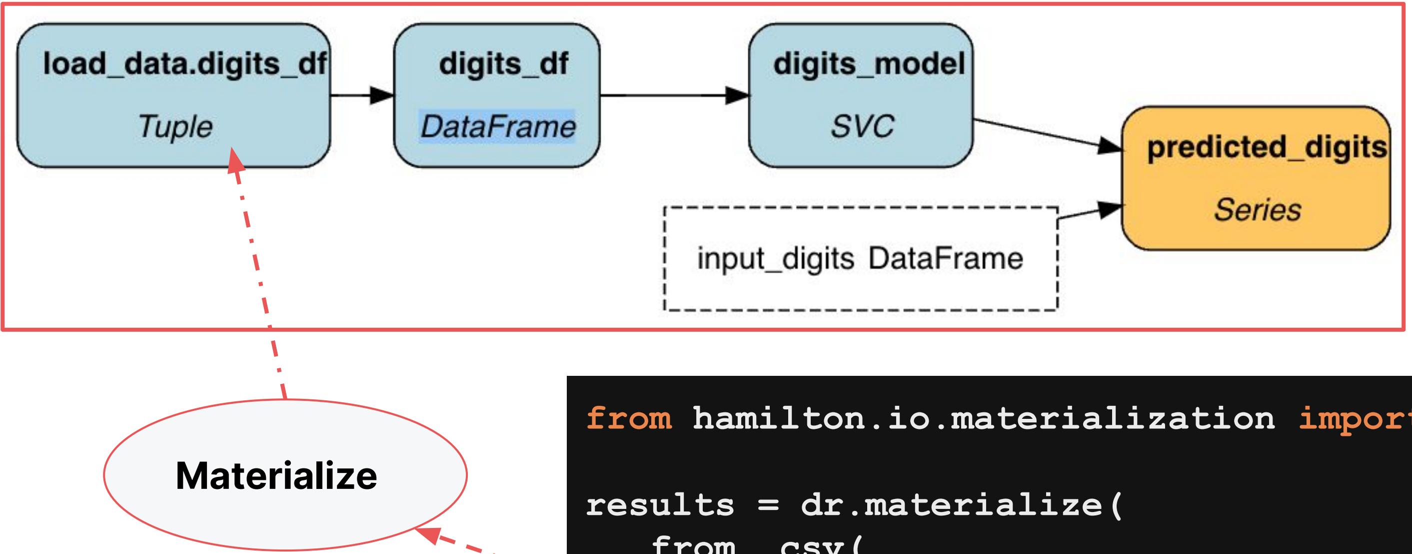
Outside Hamilton

```
input_df = load_some_digits(path, ...)
```

In Hamilton

```
dr = driver.Driver({}, simple_pipeline)
result = dr.execute(["digits_model", "..."],
 inputs={"digits_df": input_df}
)
```

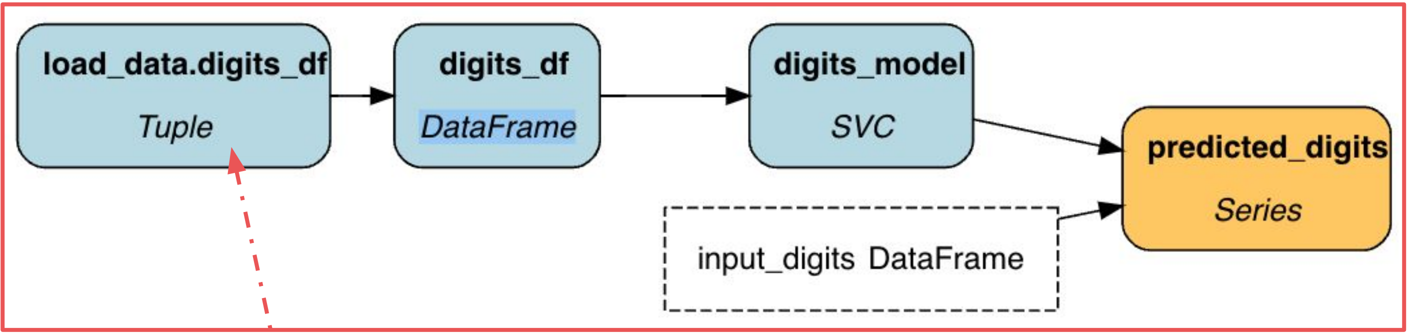
Approach 3: Inject a “materializer” into the DAG




```python
from hamilton.io.materialization import from_

results = dr.materialize(
    from_.csv(
        target="digits_df",
        path="digits_csv_path.csv"
    ),
    additional_vars=[...],
    inputs={
        "input_digits": load_some_digits().sample(5)
    }
)
```

Approach 3: Inject a “materializer” into the DAG

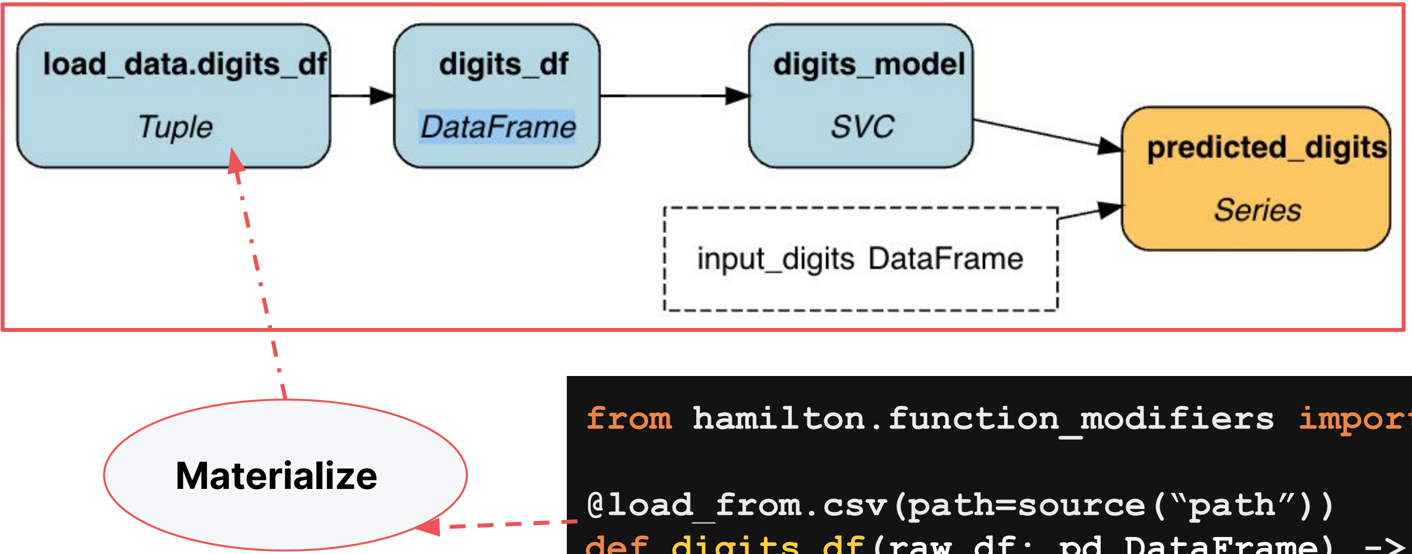


.materialize() →

```
from hamilton.io.materialization import from_

results = dr.materialize(
    from_.csv(
        target="digits_df",
        path="digits_csv_path.csv"
    ),
    additional_vars=[...],
    inputs={
        "input_digits": load_some_digits().sample(5)
    }
)
```

Approach 3: Inject a "materializer" into the DAG




```python
from hamilton.function_modifiers import load_from, source

@load_from.csv(path=source("path"))
def digits_df(raw_df: pd.DataFrame) -> pd.DataFrame:
    """Loads a digits DF"""
    df = pd.read_csv(path)
    .. # some other transforms.
    return df
```

Alternative:

Could use `.execute()` here

Let's look at some code

Let's look at some code:

- April Meet-up
- Pandas one
- MLFlow one

Recap: Data saving & loading with Hamilton

Hamilton:

- Three main approaches:
  - Embed within Hamilton
  - Embed outside Hamilton
  - Inject with `from_.*` & `to.*` or `@load_from` & `@save_to`

Benefits:

- All approaches allow you to swap out implementations.
- You have your choice of “coupling” and you can mix & match
- Side-by-side comparison: https://hamilton.dagworks.io/en/latest/concepts/materialization/
- Read more here: https://blog.dagworks.io/p/separate-data-io-from-transformation
  https://blog.dagworks.io/p/enterprise-ready-data-pipelines-with
  https://blog.dagworks.io/p/from-dev-to-prod-a-ml-pipeline-reference

Why use Hamilton's I/O abstraction?

Benefits of the data savers / loaders (i.e. materializers):

- "*Platform*" approach to encapsulating & centralizing I/O
  - Less technical debt & simpler migrations
    - E.g. can dual write for migrations
- Logging of artifacts:
  - Track & store lineage, metadata on artifacts, & provenance
  - can build CI/CD, reporting/alerting functionality

🗓️ Next month - May ??:
Looking for community spotlight!

DAGWORKS

🎤 Open Mic.

DAGWORKS

FIN. Thanks for coming!

DAGWORKS