

# Building a robust data pipeline with the “dAG” Stack: dbt, Airflow, and Great Expectations

ODSC East, April 2021

Sam Bail @spbail

# Hi, I'm Sam!

I'm a "data person" and (former) engineering director at Superconductive, the team behind Great Expectations. I'm from Germany, but currently based in NYC.\*

You can find me on Twitter @spbail  
and at sambail.com



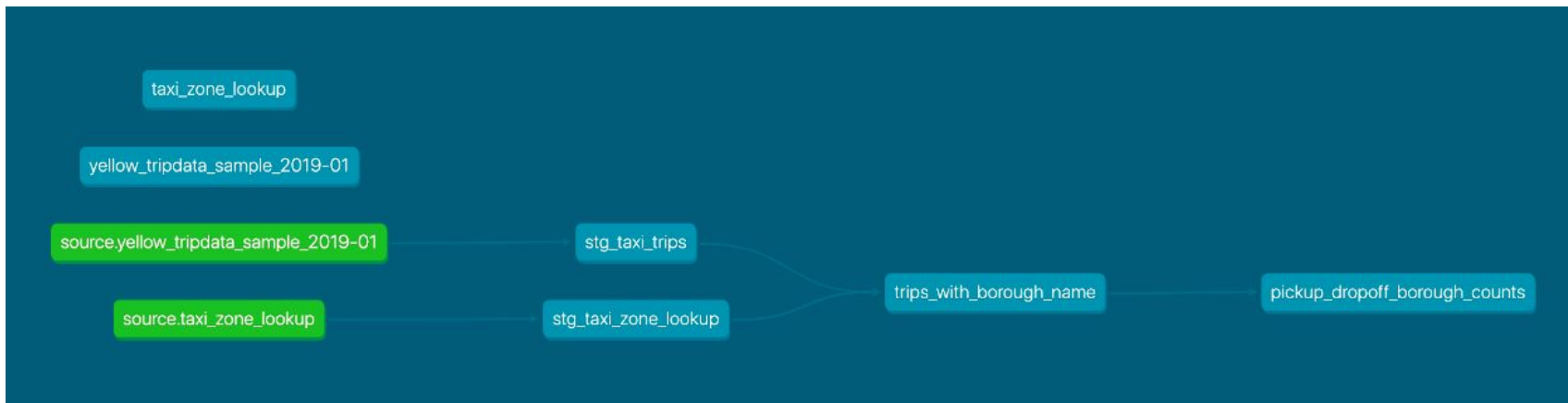
# Welcome to the "dAG" Stack: dbt, Airflow, Great Expectations

1. What are all these tools?
2. Let's talk about testing...
3. Ok, how does this all fit together?



# 1: dbt (data build tool) **dbt**

- “The T in ELT”
- Open source Python package
- Write a data pipeline as a series of templated SQL nodes



The image shows a code editor interface with a project explorer on the left and a code editor on the right. The project explorer shows a directory structure for a project named 'dag-stack'. The code editor displays the content of the file 'trips\_with\_borough\_name.sql'.

**Project Explorer:**

- ▼ **dag-stack** ~/code/dag-stack
  - > .astro
  - > .ipynb\_checkpoints
  - > dags
  - > data
  - ▼ dbt
    - > analysis
    - dbt\_modules
    - > logs
    - > macros
    - ▼ models
      - ▼ taxi\_demo
        - pickup\_dropoff\_borough\_counts.sql
        - schema.yml
        - stg\_taxi\_trips.sql
        - stg\_taxi\_zone\_lookup.sql
        - trips\_with\_borough\_name.sql**

**Code Editor (trips\_with\_borough\_name.sql):**

```
1  select
2      t.*,
3      z1.borough as pickup_borough,
4      z2.borough as dropoff_borough
5  from {{ ref('stg_taxi_trips') }} t
6  left join {{ ref('stg_taxi_zone_lookup') }} z1
7  on t.pickup_location_id = z1.locationid
8  left join {{ ref('stg_taxi_zone_lookup') }} z2
9  on t.dropoff_location_id = z2.locationid
```

Sample dbt pipeline code

```
(dag_stack) sam@DeLorean dbt % dbt run --profiles-dir /Users/sam/code/dag-stack/ --target local_dev
Running with dbt=0.19.0
Found 4 models, 0 tests, 0 snapshots, 0 analyses, 138 macros, 0 operations, 2 seed files, 2 sources, 0 exposures

16:06:33 | Concurrency: 1 threads (target='local_dev')
16:06:33 |
16:06:33 | 1 of 4 START view model public.stg_taxi_trips..... [RUN]
16:06:33 | 1 of 4 OK created view model public.stg_taxi_trips..... [CREATE VIEW in 0.24s]
16:06:33 | 2 of 4 START view model public.stg_taxi_zone_lookup..... [RUN]
16:06:33 | 2 of 4 OK created view model public.stg_taxi_zone_lookup..... [CREATE VIEW in 0.14s]
16:06:33 | 3 of 4 START view model public.trips_with_borough_name..... [RUN]
16:06:34 | 3 of 4 OK created view model public.trips_with_borough_name..... [CREATE VIEW in 0.09s]
16:06:34 | 4 of 4 START view model public.pickup_dropoff_borough_counts..... [RUN]
16:06:34 | 4 of 4 OK created view model public.pickup_dropoff_borough_counts... [CREATE VIEW in 0.10s]
16:06:34 |
16:06:34 | Finished running 4 view models in 1.78s.

Completed successfully

Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
```

Running a dbt pipeline



Search for models...

## Overview

Project

Database

## Sources

source

## Projects

taxi\_demo

..

models

taxi\_demo

pickup\_dropoff\_borough\_counts

stg\_taxi\_trips

stg\_taxi\_zone\_lookup

trips\_with\_borough\_name

## trips\_with\_borough\_name view

[Details](#) [Description](#) [Columns](#) [Referenced By](#) [Depends On](#) [SQL](#)

### Details

TAGS	OWNER	TYPE	PACKAGE	RELATION
untagged	postgres	view	taxi_demo	postgres.public.trips_with_borough_name

### Description

This model adds the borough names for the pickup and dropoff locations for each trip.

### Columns

COLUMN	DESCRIP
vendor_id	integer

Generating pipeline  
documentation with dbt

```
(dag_stack) sam@DeLorean dbt % dbt test --profiles-dir /Users/sam/code/dag-stack/ --target local_dev
Running with dbt=0.19.0
Found 4 models, 5 tests, 0 snapshots, 0 analyses, 138 macros, 0 operations, 2 seed files, 2 sources, 0 exposures

16:25:31 | Concurrency: 1 threads (target='local_dev')
16:25:31 |
16:25:31 | 1 of 5 START test accepted_values_trips_with_borough_name_vendor_id__1__2__4 [RUN]
16:25:31 | 1 of 5 PASS accepted_values_trips_with_borough_name_vendor_id__1__2__4 [PASS in 0.05s]
16:25:31 | 2 of 5 START test not_null_trips_with_borough_name_dropoff_datetime.. [RUN]
16:25:31 | 2 of 5 PASS not_null_trips_with_borough_name_dropoff_datetime..... [PASS in 0.05s]
16:25:31 | 3 of 5 START test not_null_trips_with_borough_name_passenger_count... [RUN]
16:25:31 | 3 of 5 PASS not_null_trips_with_borough_name_passenger_count..... [PASS in 0.05s]
16:25:31 | 4 of 5 START test not_null_trips_with_borough_name_pickup_datetime... [RUN]
16:25:31 | 4 of 5 PASS not_null_trips_with_borough_name_pickup_datetime..... [PASS in 0.05s]
16:25:31 | 5 of 5 START test not_null_trips_with_borough_name_vendor_id..... [RUN]
16:25:31 | 5 of 5 PASS not_null_trips_with_borough_name_vendor_id..... [PASS in 0.05s]
16:25:31 |
16:25:31 | Finished running 5 tests in 1.17s.

Completed successfully

Adding tests to a dbt pipeline

Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
```



dbt does not have any built-in scheduling  
functionality, or ability to run arbitrary  
non-SQL code

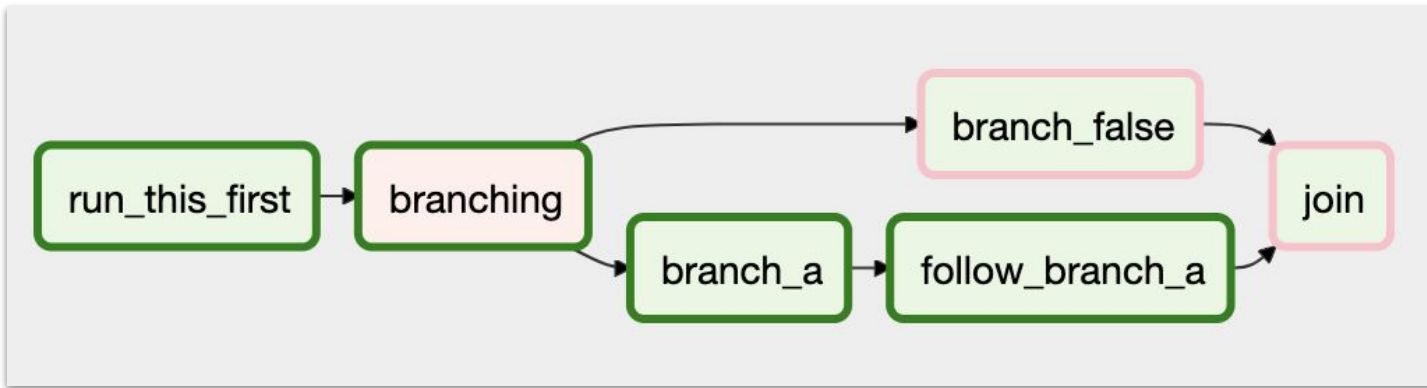
If you want to do that, you need...  
something else!



## 2: Apache Airflow



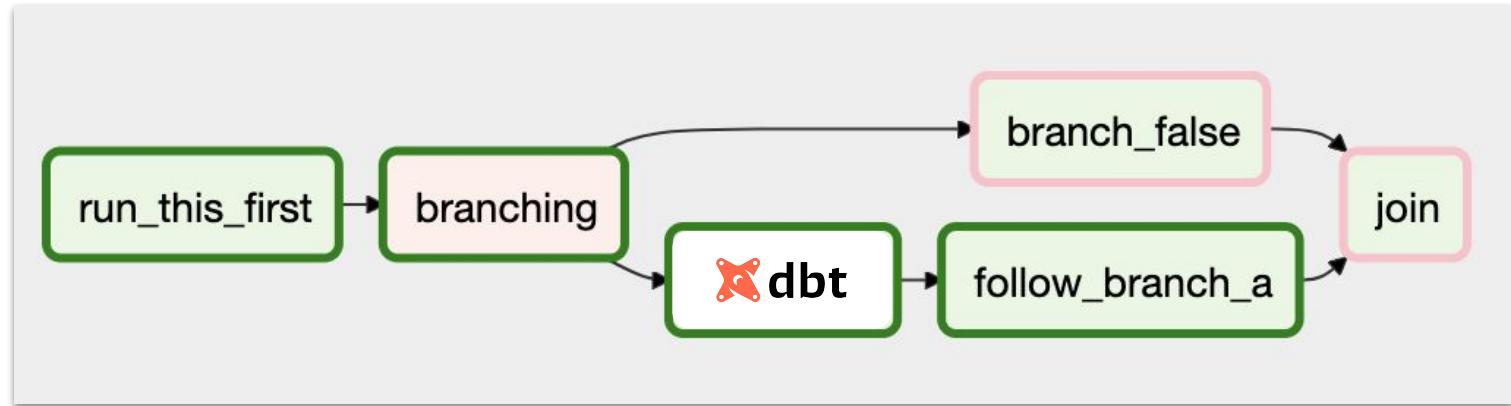
- Workflow orchestration tool
- Another open source Python package :)
- "Cron on steroids" - scheduling and more



## 2: Apache Airflow



We can run a dbt pipeline as a task in an Airflow DAG:



```
dag = DAG(
    dag_id='dag_stack_dag',
    schedule_interval=None,
    default_args=default_args
)

# This runs the transformation steps in the dbt pipeline
dbt_run = DbtRunOperator(
    task_id='dbt_run',
    dir=DBT_PROJECT_DIR,
    profiles_dir=PROJECT_HOME,
    target=DBT_TARGET,
    dag=dag
)
```

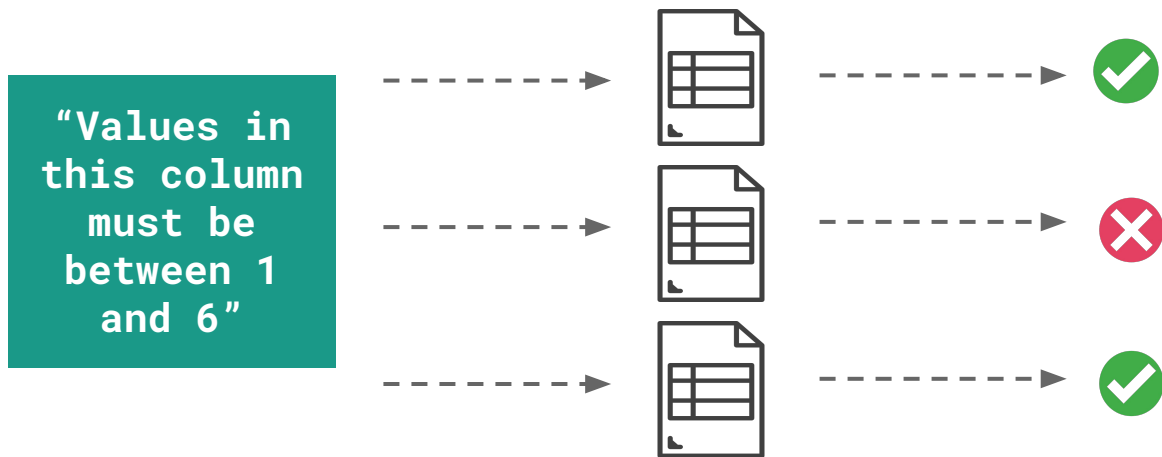
Running a dbt pipeline  
in an Airflow task\*

\* See the “Additional resources” section in the demo repo for other patterns to run dbt in Airflow

### 3: Great Expectations



- Open source data validation and documentation tool
- Let's you express what you \*expect\* from your data (ha!)



# What is an Expectation?

```
expect_column_values_to_be_between(  
    column='passenger_count',  
    min_value=1,  
    max_value=6  
)
```

A statement about what we expect from our data, that can be expressed in code

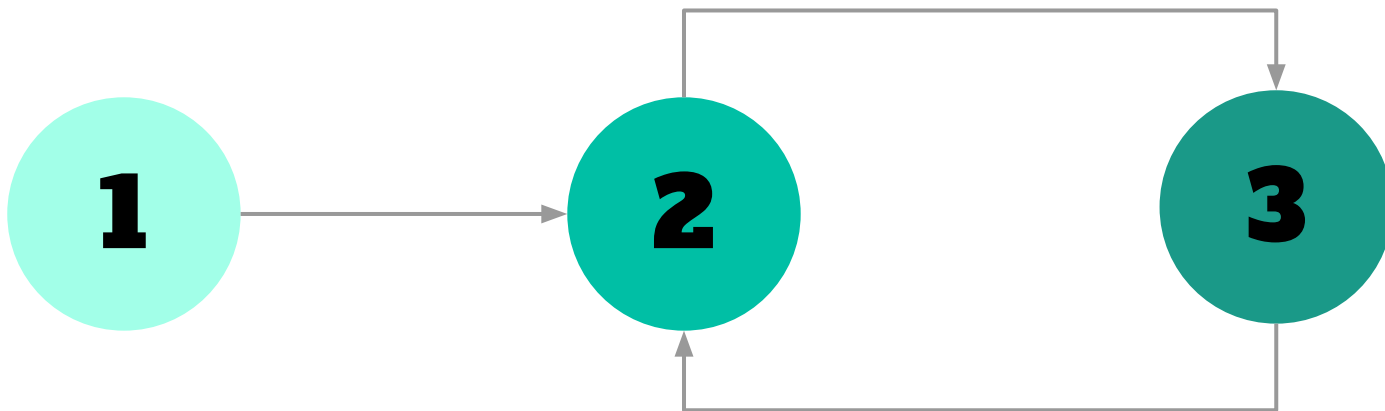
```
{  
  "expectation_type": "expect_column_values_to_be_between",  
  "kwargs": {  
    "column": "passenger_count",  
    "min_value": 1,  
    "max_value": 6  
  },  
}
```

That is stored in JSON

**"Values in this column must be between 1 and 6"**

And can be translated into a human-readable format

# Typical Great Expectations workflow



## Create an Expectation Suite

- ``suite new`` (from scratch)
- ``suite scaffold`` (profiled)
- Save the suite

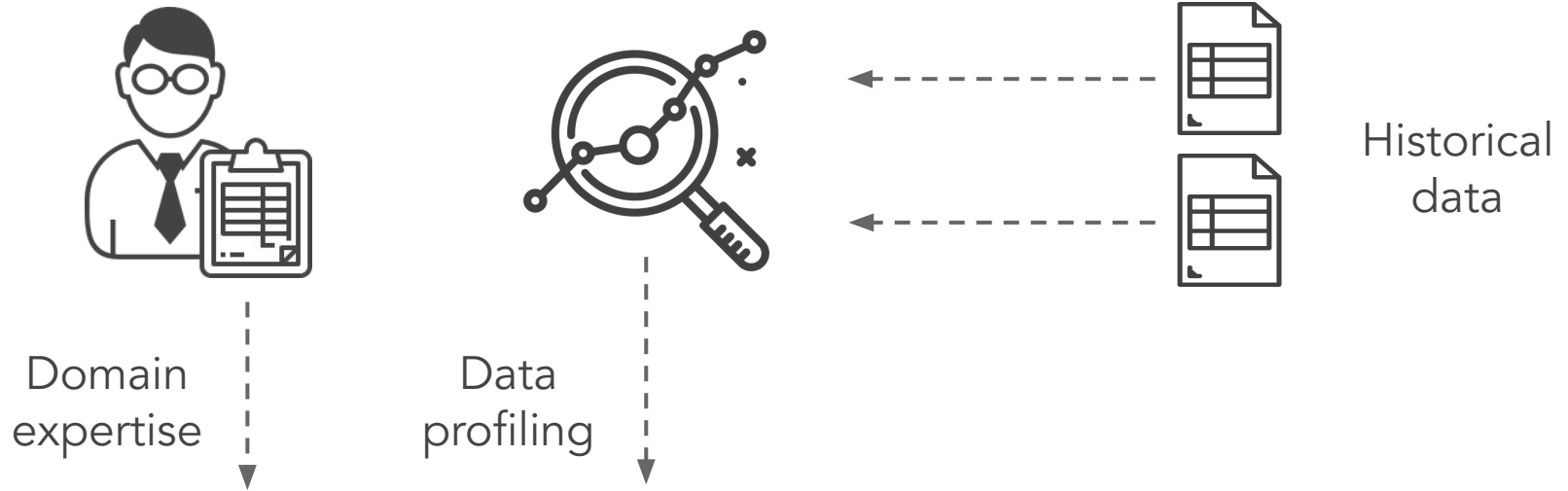
## Validate incoming data

- Load new data
- Load the suite
- Run validation

## Respond to validation results

- Stop pipeline
- Send alerts
- Build & inspect Data Docs

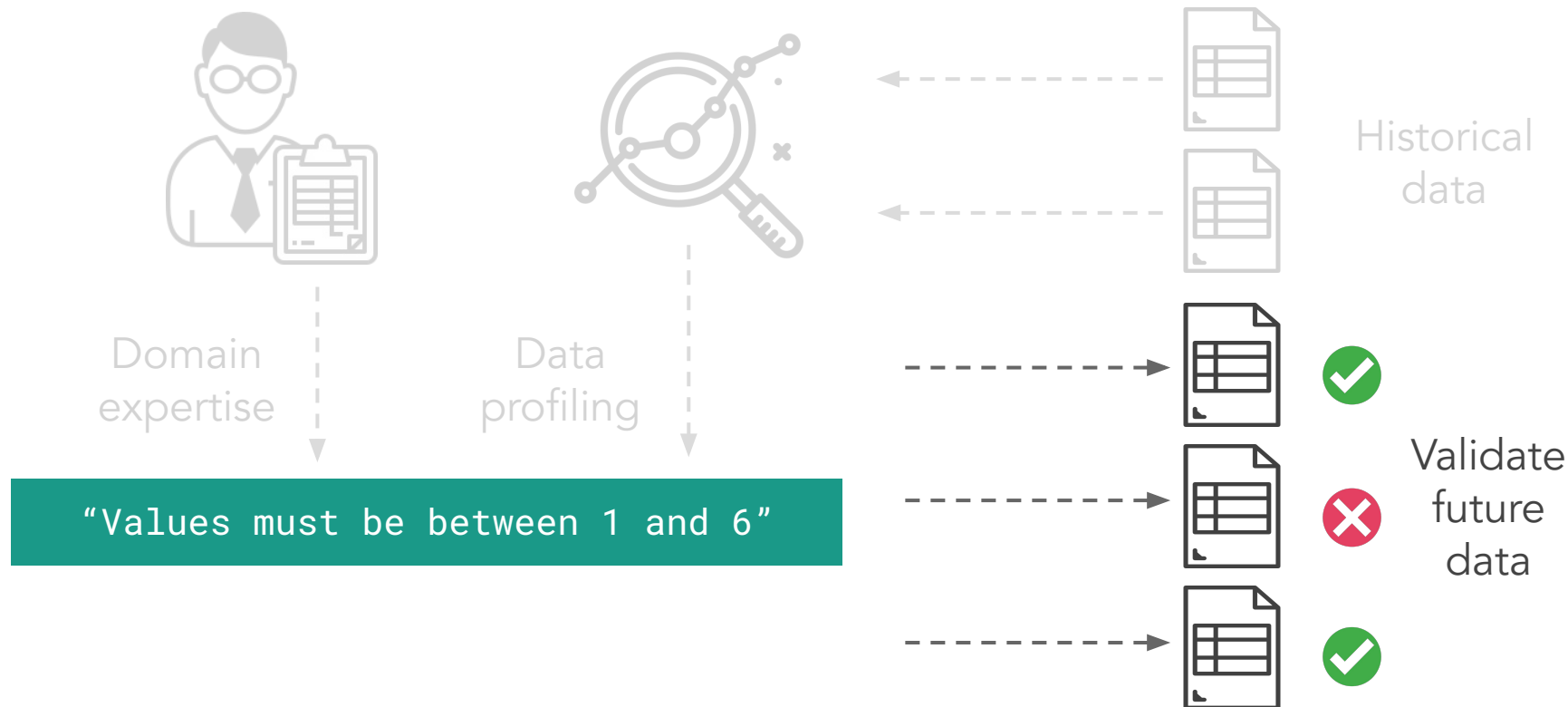
# Automated profiling to “scaffold” Expectations



"Values must be between 1 and 6"



# Validating your data





## Actions

Validation Filter:

[Show All](#) [Failed Only](#)[✎ How to Edit This Suite](#)[Show Walkthrough](#)

## Table of Contents

[Overview](#)[dropoff\\_datetime](#)[fare\\_amount](#)[passenger\\_count](#)[pickup\\_datetime](#)[trip\\_distance](#)

## passenger\_count

# Data Docs renders validation results into "reports"

Status	Expectation	Observed Value
✖	<p>values must always be between 1 and 6.</p> <p>1579 unexpected values found. ≈15.79% of 10000 total rows.</p> <div><b>Sampled Unexpected Values</b></div> <div>0.0</div>	≈15.79% un

## pickup\_datetime

Status	Expectation	Observed Value
✔	values must never be null.	100% not null

```
validate_source_data = GreatExpectationsOperator(  
    task_id='validate_source_data',  
    assets_to_validate = [  
        {  
            'batch_kwargs': {  
                'path': os.path.join(PROJECT_HOME, 'data', 'taxi_zone_lookup.csv'),  
                'datasource': 'data_dir'  
            },  
            'expectation_suite_name': 'taxi_zone.source'  
        },  
        {  
            'batch_kwargs': {  
                'path': os.path.join(PROJECT_HOME, 'data', 'yellow_tripdata_sample_2019-01.csv'),  
                'datasource': 'data_dir'  
            },  
            'expectation_suite_name': 'taxi_trips.source'  
        },  
    ],  
    data_context_root_dir=GE_ROOT_DIR,  
    dag=dag  
)
```

Running validation with Great  
Expectations in an Airflow task

Ok, let's talk about testing!



# 1: You should test your data.

(No, really.)

Don't believe me?

*“Our stakeholders would notice data issues before we did... which really **eroded trust** in the data and our team.”*

*(A Great Expectations user)*

*“Re-running our pipelines after finding a data quality issue would incur **actual costs** for the compute environment.”*

*(A Great Expectations user)*



*“Remember that one Thanksgiving where we **worked all weekend** to fix those data issues we only noticed at the last minute? Never again.”*

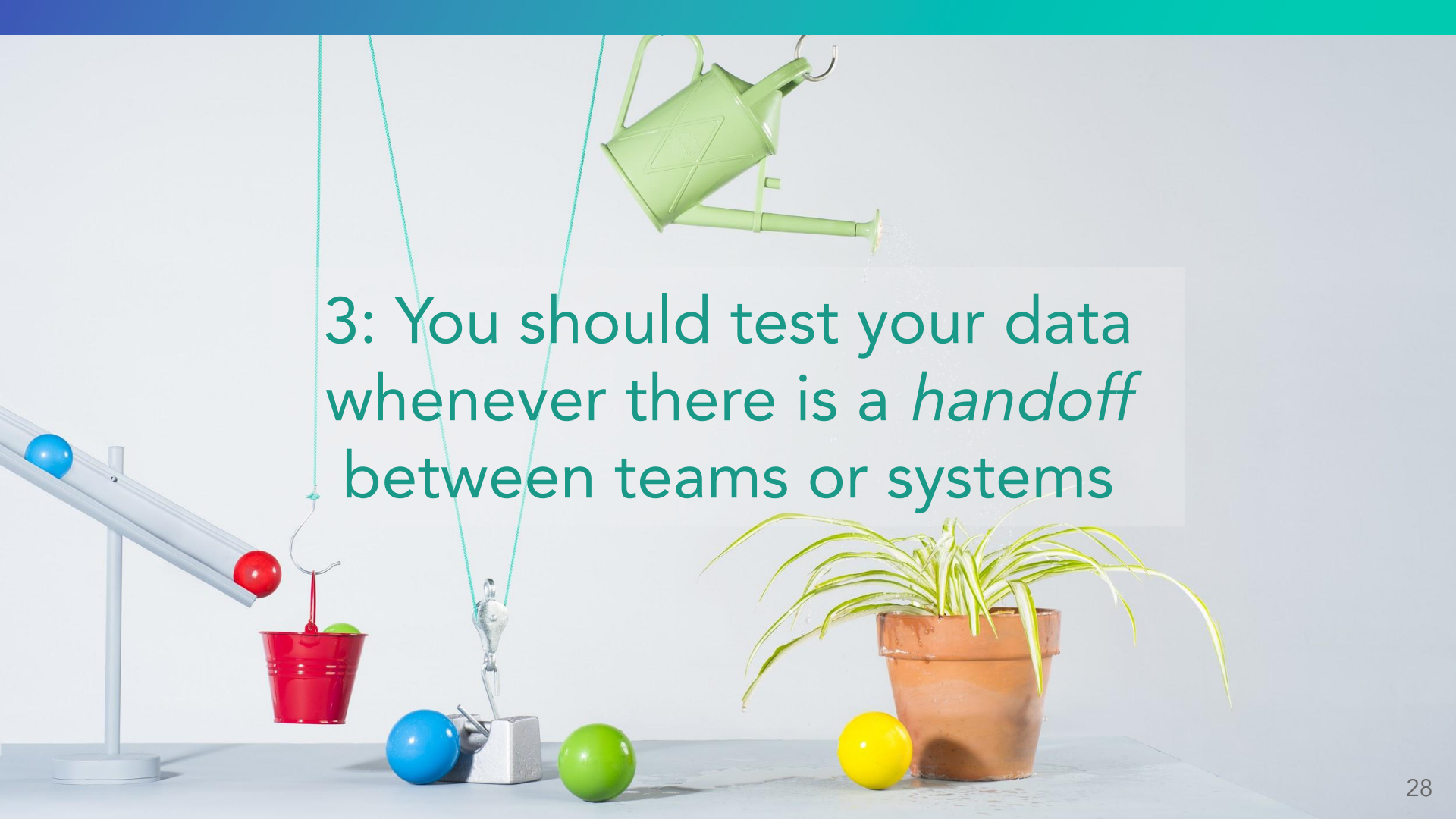
*(That was me. True #datahorrorstory.)*

# But... where do we start?

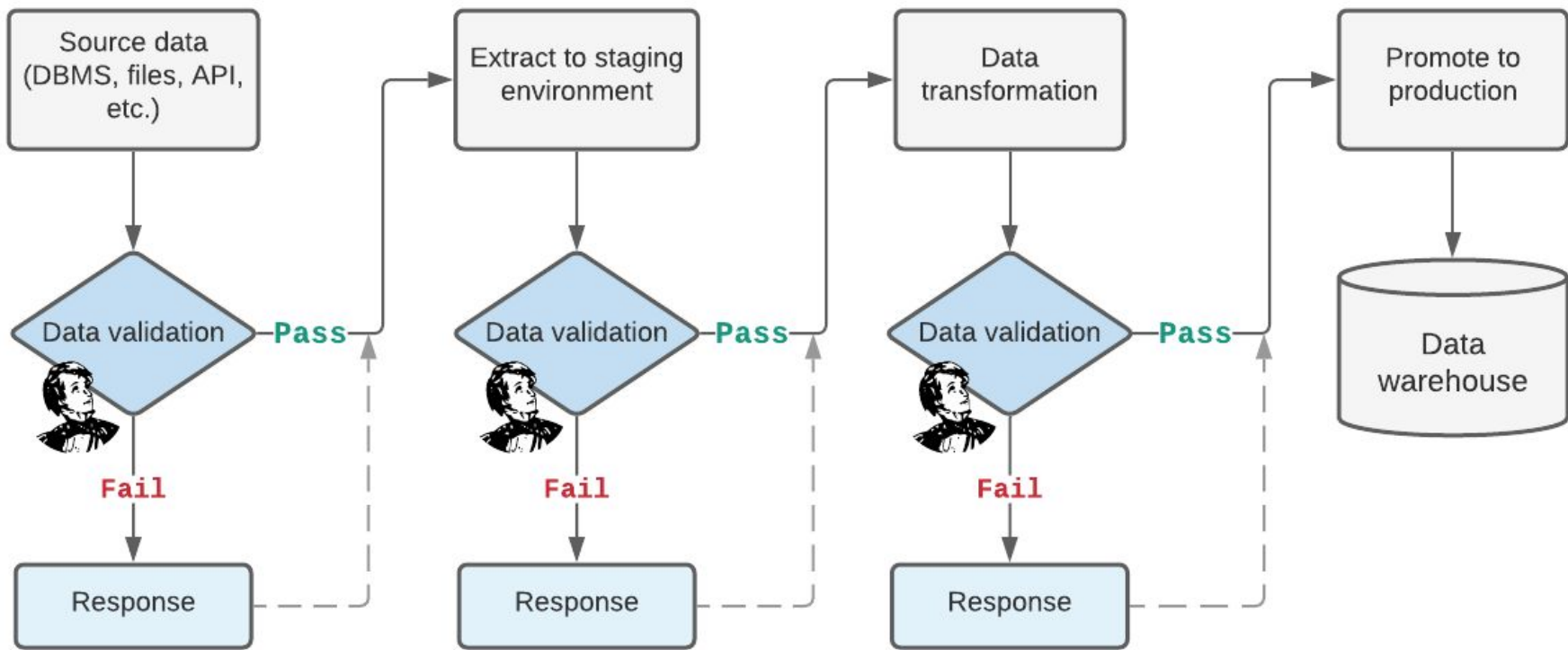
(Which tool should we use? How do we know we're testing the right thing? What do we do when tests fail? Who owns this? How do we keep them up to date? How do our stakeholders find out about the state of the data?)

## 2: Data testing is kinda hard.

(But I can show you how to get started...)

A Rube Goldberg-style contraption is shown against a light blue background. At the top, a green watering can is suspended by a string and is pouring water. Below it, a red bucket is suspended by a string and is catching water. To the left, a ramp with a blue ball and a red ball is shown. In the center, a pulley system is visible. On the right, a potted plant is shown. The text "3: You should test your data whenever there is a *handoff* between teams or systems" is overlaid on the image.

3: You should test your data  
whenever there is a *handoff*  
between teams or systems



Example pipeline with tests at every relevant "handoff" point

Let's put this all together now.



dag-stack > dags > dag\_stack\_dag.py

Project

- dag-stack ~/code/dag-stack
  - .astro
  - .ipynb\_checkpoints
  - dags
    - dag\_stack\_dag.py
  - data
  - dbt
    - analysis
    - dbt\_modules
  - logs
  - macros
  - models
  - snapshots
  - target
  - tests
  - .gitignore
  - dbt\_project.yml
  - README.md
- great\_expectations
  - checkpoints
  - expectations
  - notebooks
  - plugins
  - uncommitted
  - .gitignore
  - great\_expectations.yml

Commit

Pull Requests

```
35 GE_ROOT_DIR = os.path.join(PROJECT_HOME, 'include', 'great_expectations_root',
36
37 dag = DAG(
38     dag_id='dag_stack_dag',
39     schedule_interval=None,
40     default_args=default_args
41 )
42
43 ...
44
45 validate_source_data = GreatExpectationsOperator(
46     task_id='validate_source_data',
47     assets_to_validate=[...],
48     data_context_root_dir=GE_ROOT_DIR,
49     dag=dag
50 )
51
52 # The dbt seed command loads files in the data directory to the database
53 dbt_seed = DbtSeedOperator(
54     task_id='dbt_seed',
55     dir=DBT_PROJECT_DIR,
56     profiles_dir=PROJECT_HOME,
57     target=DBT_TARGET,
58     dag=dag
59 )
60
```

Demo repo live on

[github.com/spbail/dag-stack](https://github.com/spbail/dag-stack)

# DAG: dag\_stack\_dag

success schedule: None

Tree View Graph View Task Duration Task Tries Landing Times Gantt Details Code



2021-03-31T01:28:39Z

Runs

25



Run

manual\_\_2021-03-31T01:28:38.003284+00:00



Layout

Left > Right



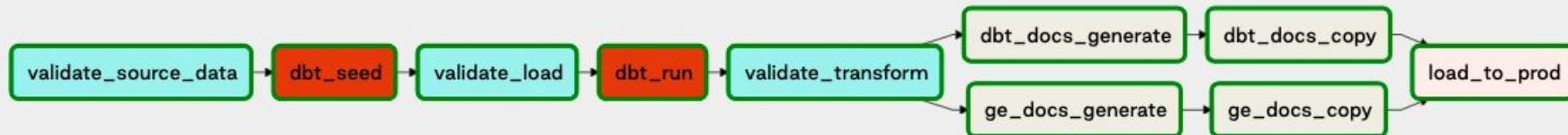
Update

Find Task...

BashOperator DbtRunOperator DbtSeedOperator GreatExpectationsOperator PythonOperator

queued running success failed up\_for\_retry up\_for\_reschedule upstream\_failed skipped scheduled no\_status

Auto-refresh





## DAG: dag\_stack\_dag

success schedule: None

Tree View Graph View Task Duration Task Tries Landing Times Gantt Details Code

2021-03-31T01:28:39Z Runs 25 Run manual\_\_2021-03-31T01:28:38.003284+00:00 Layout Left > Right Update Find Task...

BashOperator DbtRunOperator DbtSeedOperator GreatExpectationsOperator PythonOperator

queued running success failed up\_for\_retry up\_for\_reschedule upstream\_failed skipped scheduled no\_status

Auto-refresh



Test that source data (in this example: CSV files) matches expected format, e.g. correct number of columns, data types, row count "similar" to last month's, etc.

## DAG: dag\_stack\_dag

success schedule: None

Tree View Graph View Task Duration Task Tries Landing Times Gantt Details Code

2021-03-31T01:28:39Z Runs 25 Run manual\_\_2021-03-31T01:28:38.003284+00:00 Layout Left > Right Update Find Task...

BashOperator DbtRunOperator DbtSeedOperator GreatExpectationsOperator PythonOperator

queued running success failed up\_for\_retry up\_for\_reschedule upstream\_failed skipped scheduled no\_status

Auto-refresh



Test that source data load was successful, e.g. no rows lost compared to source

## DAG: dag\_stack\_dag

success schedule: None

Tree View Graph View Task Duration Task Tries Landing Times Gantt Details Code



2021-03-31T01:28:39Z

Runs

25



Run

manual\_\_2021-03-31T01:28:38.003284+00:00



Layout

Left > Right



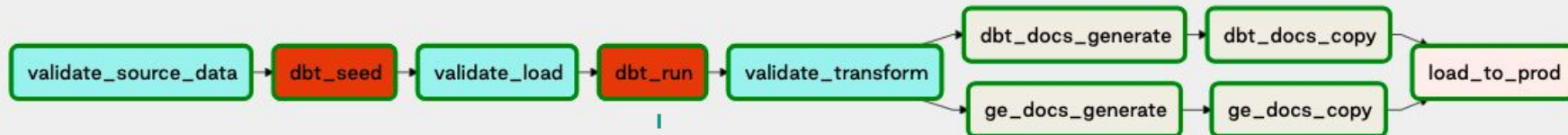
Update

Find Task...

BashOperator DbtRunOperator DbtSeedOperator GreatExpectationsOperator PythonOperator

queued running success failed up\_for\_retry up\_for\_reschedule upstream\_failed skipped scheduled no\_status

Auto-refresh



❌ **dbt** Run tests during DAG *development* to check for integrity of transformations

## DAG: dag\_stack\_dag

success schedule: None

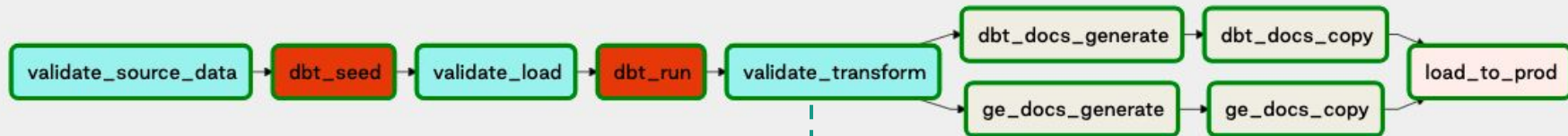
Tree View Graph View Task Duration Task Tries Landing Times Gantt Details Code

2021-03-31T01:28:39Z Runs 25 Run manual\_\_2021-03-31T01:28:38.003284+00:00 Layout Left > Right Update Find Task...

BashOperator DbtRunOperator DbtSeedOperator GreatExpectationsOperator PythonOperator

queued running success failed up\_for\_retry up\_for\_reschedule upstream\_failed skipped scheduled no\_status

Auto-refresh



❌ **dbt** Test integrity of transformations, e.g. no fan-out joins, no NULL columns, etc.



Use off-the-shelf methods for complex tests, e.g. distributions of values

# Wrap-up!

- Airflow, dbt, and Great Expectations = a modern data pipeline stack with built-in testing capabilities
- Test your data whenever there is a handoff between teams or systems
- Use dbt or Great Expectations depending on what data you test and the complexity of your tests



# Thank you!

Ping me - Sam Bail, @spbail  
Repo: [github.com/spbail/dag-stack](https://github.com/spbail/dag-stack)

