Introduction to testing in dbt

INTRODUCTION TO DBT

Mike Metzger
Data Engineer

What is a test?

- Assertions / validations of dbt objects
  - Models
  - Sources, seeds, snapshots
- Used to verify data is as expected
  - Null values
  - Values in range
  - Relationships between data
  - Custom tests

Test types

- Built-in
- Singular
- Generic

Built-in tests

- `unique`
  - Verify all values are unique
- `not_null`
  - Verify all values are not null
- `accepted_values`
  - Verify all values are within a specific list
  - `values: [a, b, c, d]`
- `relationships`
  - Verifies a connection to a specific table / column
  - `to: ref('table')`
  - `field: id`

Where to apply tests?

- `models/model_properties.yml`
  - File can be named anything
  - `models/schema.yml`
- Defined in the `tests:` subheading

```
version: 2

models:
  - name: taxi_rides_raw
    columns:
      - name: tpep_pickup_datetime
        tests:
          - not_null
      - name: payment_type
        tests:
          - not_null
          - accepted_values:
              values: [1, 2, 3, 4, 5, 6]
```

Running tests

- `dbt test`
  - `dbt test --select modelname`
- Verify output passes
- If failure, check against compiled sql

```
(dbt-test) Charlene:nyc_yellow_taxi mmetzger$ dbt test
04:32:39  Running with dbt=1.4.5
04:32:39  Found 3 models, 3 tests, 0 snapshots, 0 analyses, 296 macros, 0 operations, 0 seed files, 0 sources, 0 exposures, 0 metrics
04:32:39
04:32:39  Concurrency: 1 threads (target='dev')
04:32:39
04:32:39  1 of 3 START test accepted_values_taxi_rides_raw_payment_type__1__2__3__4__5__6  [RUN]
04:32:39  1 of 3 FAIL 1 accepted_values_taxi_rides_raw_payment_type__1__2__3__4__5__6 .... [FAIL 1 in 0.09s]
04:32:39  2 of 3 START test not_null_taxi_rides_raw_payment_type ......................... [RUN]
04:32:39  2 of 3 PASS not_null_taxi_rides_raw_payment_type ............................... [PASS in 0.01s]
04:32:39  3 of 3 START test not_null_taxi_rides_raw_tpep_pickup_datetime ................. [RUN]
04:32:40  3 of 3 PASS not_null_taxi_rides_raw_tpep_pickup_datetime ....................... [PASS in 0.01s]
04:32:40
04:32:40  Finished running 3 tests in 0 hours 0 minutes and 0.15 seconds (0.15s).
```

Finding failures

1. Must find compiled SQL code
2. In the `target/compiled/projectname/models/model_properties.yml` directory
   - `target/compiled/nyc_yellow_taxi/models/model_properties.yml/`
3. Find the appropriate `.sql` file (matching the failed test)
4. Copy the contents into database client and verify where issue exists
5. Remove from data and re-run `dbt run` / `dbt test`

Let's practice!

INTRODUCTION TO DBT

Creating singular tests

INTRODUCTION TO DBT

Mike Metzger
Data Engineer

What is a singular test?

- Custom data test
- Written as an SQL query
  - Must return failing rows
- Defined as `.sql` file in `tests` directories

[1] Photo by Antoine Dautry on Unsplash

Example singular test

- Create a test to verify the `order_total` is greater than or equal to the `subtotal`

```
select *
from order
where order_total < subtotal
```

- Remember, we're looking for the rows that fail the test. If the test returns rows, then the test is marked as failed.
- Save file as `assert_order_total_gte_subtotal.sql`

ORDER	
int	order_number
string	customer_name
datetime	order_datetime
int	subtotal
int	shipping
int	tax
int	order_total

Singular test with Jinja

- We can use Jinja in our tests
  - `ref` function
  - Others as appropriate
- dbt substitutes output when test is run

```
select *
from {{ ref('order') }}
where order_total < subtotal
```

Test debugging

- Use a SQL editor to create the initial test query
- Place query in appropriate file
- Make sure to name the test uniquely
- Use the `dbt test --select <testname>` option
- Check any errors and update accordingly

Let's practice!

INTRODUCTION TO DBT

Creating custom reusable tests

INTRODUCTION TO DBT

Mike Metzger
Data Engineer

What is a reusable test?

- A test that can be reused in multiple situations
- Much like a built-in dbt test, but can check any condition
- Uses Jinja templating
- Saved as a `.sql` file in the `tests/generic` project folder
- Must add test to the `model_properties.yml` for each model that uses it

[1] Photo by Sigmund on Unsplash

Creating a reusable test

- Add `{% test testname(model, column_name) %}`
- Add SQL query, with `{{ object }}` substitutions
- End the file with `{% endtest %}`

Reusable test example

```
{% test check_gt_0(model, column_name) %}


select *
from {{ model }}
where {{ column_name }} > 0


{% endtest %}
```

Applying reusable test to model

- Add to `model_properties.yml`
- Define the objects as necessary
- The models: name value is the model argument
- The columns: name argument is the column_name argument

```
version: 2


models:
  - name: taxi_rides_raw
    columns:
      - name: tpep_pickup_datetime
        tests:
          - not_null
      - name: total_fare
        tests:
          - check_gt_0
```

Extra parameters

- Can add extra parameters to the test
- Similar to `accepted_values` and `relationships`
- Add as arguments to the Jinja header

```
{% test check_columns_unequal(model, column_name, column_name2) %}

select * from {{ model }}
where {{ column_name }} = {{ column_name2 }}

{% endtest %}
```

Applying extra parameters tests

- Define like other tests
- Add the extra arguments below the test details

```
models:
  - name: order
    columns:
      - name: order_time
        tests:
          - check_columns_unequal:
              column_name2: shipped_time
```

Let's practice!

INTRODUCTION TO DBT

Creating and generating dbt documentation

INTRODUCTION TO DBT

Mike Metzger
Data Engineer

datacamp

Why document?

- Sharing data details with other consumers
- Centralize sources of documentation
- Providing details for updates / changes / etc
- Creating examples, suggestions for use, SLA details

Creating documentation in dbt

- Can provide documentation with model definitions
- Can add documentation about columns within models
- Automatically show data lineage / DAG
- Document any test / validations
- View generated warehouse information
  - Column data types
  - Data sizes

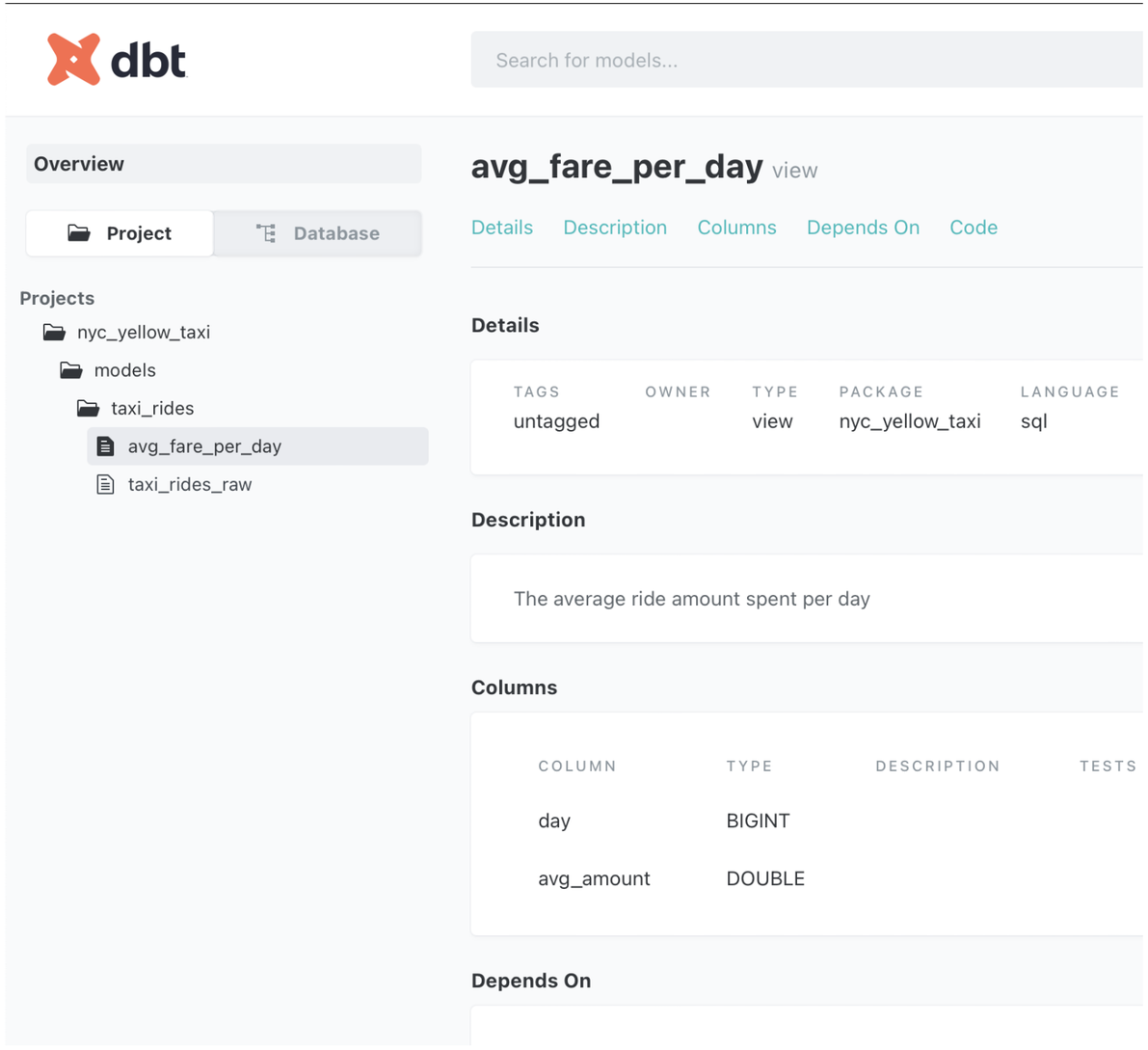
```
version: 2

models:
  - name: taxi_rides_raw
    description: Yellow Taxi raw data
    access: public
  - name: avg_fare_per_day
    description: Average ride per day
    access: public
```

Generating documentation in dbt

- `dbt docs`
  - `dbt docs -h`
  - `dbt docs generate`
- Creates the documentation website based on project
- Should be run after `dbt run`

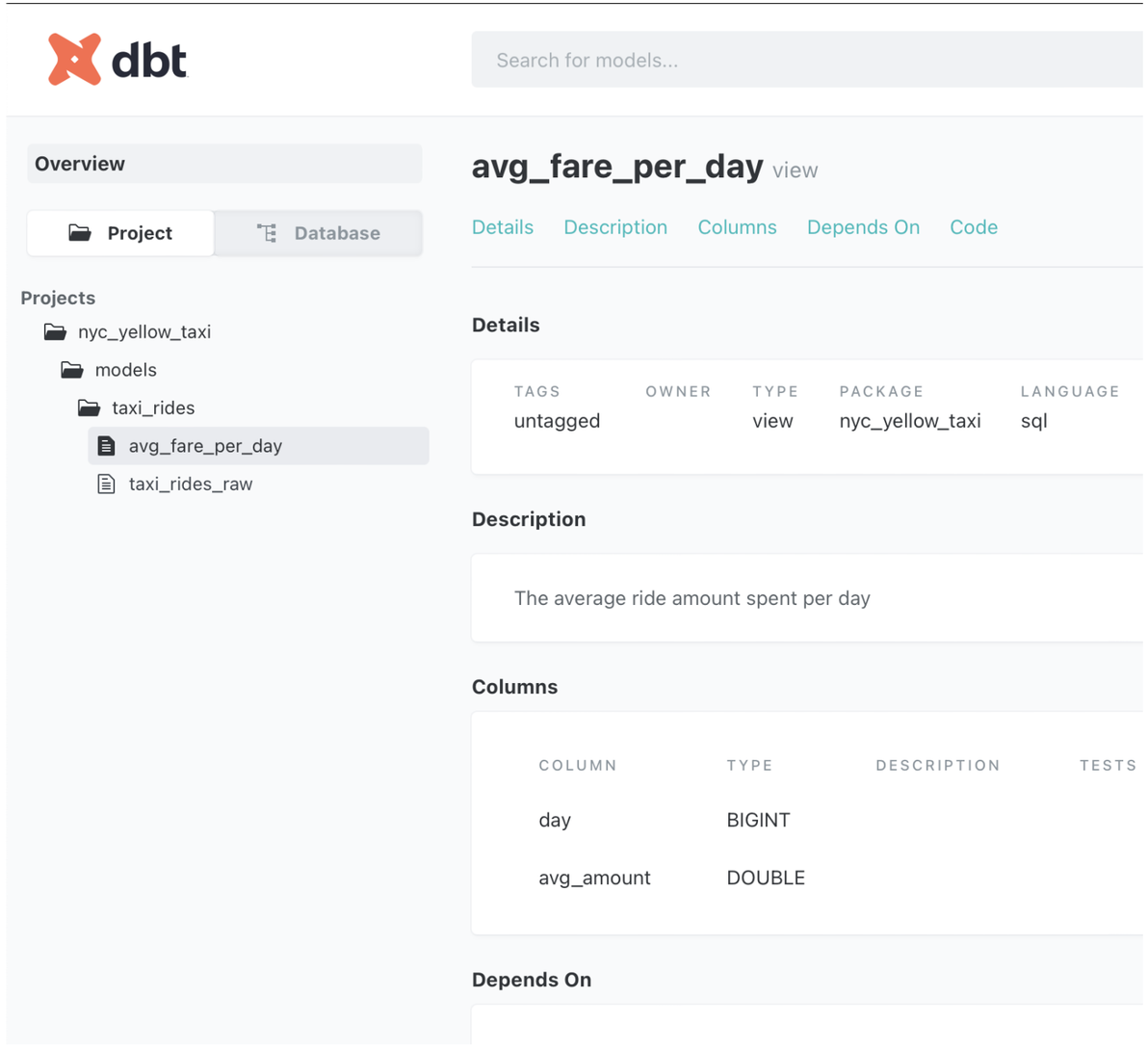
Accessing documentation

- Web browser
- `dbt docs serve`
  - Should only be used locally / for development
- Copy content to other hosting service
  - dbt cloud
  - Amazon S3
  - Nginx / Apache / etc



Documentation example

- View
  - Models
  - Description information
  - Column details
  - Lineage graphs



Let's practice!

INTRODUCTION TO DBT