Greenery Platform `dbt` Models

Documentation, Macros and Tests

Overview

1. Custom Project Documentation
2. Custom **`dbt`** Macros Used
    a. Written for Re-Usability
        i. Across models
        ii. Across macros
    b. Fully renders multi-granularity conversion rate models
3. Test Suite Used
    a. Purpose-built packages
    b. Stats for tests deployed by model layer
    c. Errors flagged in models by tests

Custom Project Documentation

Custom Overview Page

The dbt documentation overview page was customized for Greenery's data models with

1. explanation of model organization
2. brief note about tests and macros
3. link to source code

Overview

Welcome to the home page for the `dbt` documentation for Greenery's data models.

Project Structure

The models are organized into a project structure that follows `dbt` **Best Practices**

1. `staging` models consist of light transformations being applied to clean the source data
2. `intermediate` models are stored in separate sub-folders and are created for (a) products and (b) orders. Since `intermediate` models were created at different levels (orders and products) it was decided to create an `intermediate` folder at the root of the `models` directory. This resulted in two sub-folders (`intermediate/orders` and `intermediate/products`) within the parent `intermediate` folder. A separate nested `intermediate` sub-folder was not created within each `marts` folder for two reasons (a) three such folders would be needed, one for each `marts` model, (b) the `intermediate/orders` models are used by both the `core` and `marketing` business units. Creating a separate`intermediate` folder that is not nested within `marts` avoided repeating SQL code in those `intermediate` models.
3. `marts` models are organized into separate sub-folders for each intended business user (a) marketing team (`marketing` folder), (b) multiple teams (`core` folder, primarily the operations team) and (c) product team (`product` folder, since metrics were calculated at two levels, overall and per day, two sub-folders are created within the `product` folder)

Tests

A suite of tests was implemented using

1. `dbt`'s built-in testing utilities
2. custom-written generic tests, placed in `tests/generic`

More complex tests are implemented macros provided by two `dbt` packages

1. `dbt-expectations`, which is a package designed to bring the power of the Great Expectations framework to ensure data quality to data models in `dbt`
2. `dbt-utils`

Some of the macros that are predominantly used to test Greenery's data models are listed later.

Custom Overview Page

The dbt documentation overview page was customized for Greenery's data models with

1. explanation of model organization
2. brief note about tests and macros
3. link to source code

1. `staging` models consist of light transformations being applied to clean the source data
2. `intermediate` models are stored in separate sub-folders and are created for (a) products and (b) orders. Since `intermediate` models were created at different levels (orders and products) it was decided to create an `intermediate` folder at the root of the `models` directory. This resulted in two sub-folders (`intermediate/orders` and `intermediate/products`) within the parent `intermediate` folder. A separate nested `intermediate` sub-folder was not created within each `marts` folder for two reasons (a) three such folders would be needed, one for each `marts` model, (b) the `intermediate/orders` models are used by both the `core` and `marketing` business units. Creating a separate`intermediate` folder that is not nested within `marts` avoided repeating SQL code in those `intermediate` models.
3. `marts` models are organized into separate sub-folders for each intended business user (a) marketing team (`marketing` folder), (b) multiple teams (`core` folder, primarily the operations team) and (c) product team (`product` folder, since metrics were calculated at two levels, overall and per day, two sub-folders are created within the `product` folder)

Tests

A suite of tests was implemented using

1. `dbt`'s built-in testing utilities
2. custom-written generic tests, placed in `tests/generic`

More complex tests are implemented macros provided by two `dbt` packages

1. `dbt-expectations`, which is a package designed to bring the power of the Great Expectations framework to ensure data quality to data models in `dbt`
2. `dbt-utils`

Some of the macros that are predominantly used to test Greenery's data models are listed later.

Custom Macros

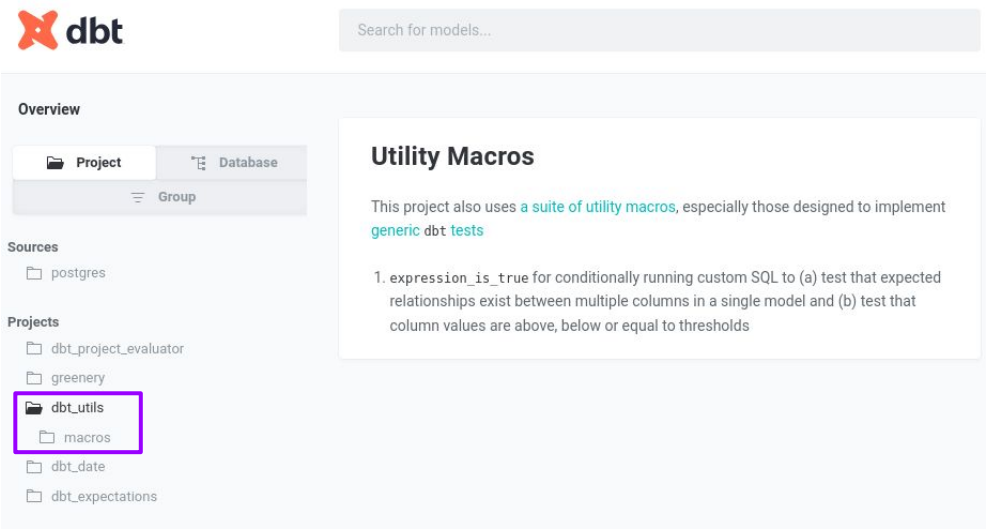
A custom set of macros has been developed to simplify and improve data model DAGs. All macros are placed in the `macros` sub-folder and are fully documented.

Source Code

Custom Pages for Packages

Brief overview pages for dbt packages were also included for

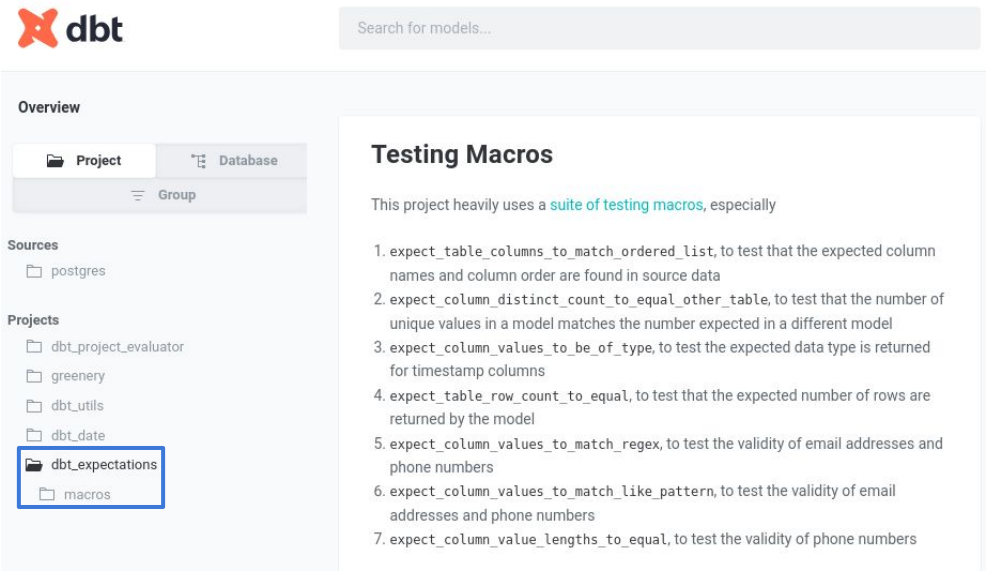
1. `dbt-utils`
2. `dbt-expectations`



Custom Pages for Packages

Brief overview pages for dbt packages were also included for

1. dbt-utils
2. dbt-expectations



Documentation was Prepared Using `dbt docs blocks` for Maintainability

EXPLORER

- course-dbt
  - .vscode
  - answers
  - greenery
    - analyses
    - dbt_packages
    - logs
    - macros
    - models
      - docs
        - intermediate
          - orders
            - .gitkeep
            - int_orders_docs.md
        - products
          - .gitkeep
        - marts
        - staging
      - intermediate
        - orders
          - _int_orders_models.yml
          - .gitkeep
          - int_orders_joined_to_addresses_promos.sql
        - products
      - marts
      - staging
      - overview.md

int_orders_docs.md

course-dbt > greenery > models > docs > intermediate > orders > int_orders_docs.md

```
{% docs int_orders_description %}

user orders combined with metadata for address and promotions

{% enddocs %}

{% docs int_orders_total_order_size %}

Total number of units of the product in this order

{% enddocs %}

{% docs int_orders_num_unique_products %}

number of uniqie items offered by Greenery that are included in
this order

{% enddocs %}

{% docs int_orders_is_on_time_delivery %}

whether product was delivered after estimated delivery time

{% enddocs %}
```

_int_orders_models.yml

course-dbt > greenery > models > intermediate > orders > _int_orders_models.yml

```
models:
  - name: int_orders_joined_to_addresses_promos
    columns:
      - name: shipping_cost
        data_tests:
        tags: [dbt_utils]
      - name: promo_id
        description: '{{ doc("promos_promo_id") }}'
        data_tests:
          - accepted_values:
              values:
                - 'instruction set'
                - Optional
                - Mandatory
                - Digitized
                - 'task-force'
                - leverage
          - dbt_expectations.expect_column_distinct_count_to_equal_othe
              compare_model: ref('stg_postgres_promos')
              compare_column_name: promo_id
              row_condition: "promo_id is not null"
        tags: [dbt_expectations]
      - name: discount
        description: '{{ doc("promos_discount") }}'
        data_tests:
          - positive_values
          - dbt_utils.expression_is_true:
              expression: ">0"
              where: "promo_id IS NOT NULL"
        tags: [dbt_utils]
      - name: order_total
        description: '{{ doc("orders_order_total") }}'
        data_tests:
          - positive_values
          - dbt_utils.expression_is_true:
              expression: "=ROUND(order_cost+shipping_cost, 2)"
              where: "promo_id IS NULL"
          - dbt_utils.expression_is_true:
              expression: "<ROUND(order_cost+shipping_cost, 2)"
              where: "promo_id IS NOT NULL"
        tags: [dbt_utils]
      - name: total_order_size
        description: '{{ doc("int_orders_total_order_size") }}'
        data_tests:
          - positive_values
      - name: num_unique_products
        description: '{{ doc("int_orders_num_unique_products") }}'
        data_tests:
          - positive_values
      - name: status
        description: '{{ doc("orders_status") }}'
        data_tests:
          - accepted_values:
              values: [delivered, shipped, preparing]
```

`models/docs` sub-folders created to hold re-usable documentation and column descriptions are re-used

Documentation was Prepared Using `dbt docs blocks` for Maintainability

EXPLORER

- course-dbt
  - .vscode
  - answers
  - greenery
    - analyses
    - dbt_packages
    - logs
    - macros
    - models
      - docs
        - intermediate
          - orders
            - .gitkeep
            - int_orders_docs.md
          - products
            - .gitkeep
        - marts
        - staging
      - intermediate
        - orders
          - _int_orders_models.yml
          - .gitkeep
          - int_orders_joined_to_addresses_promos.sql
        - products
      - marts
      - staging
      - overview.md

int_orders_docs.md

course-dbt > greenery > models > docs > intermediate > orders > int_orders_docs.md

```
{% docs int_orders_description %}

user orders combined with metadata for address and promotions

{% enddocs %}

{% docs int_orders_total_order_size %}

Total number of units of the product in this order

{% enddocs %}

{% docs int_orders_num_unique_products %}

number of uniqie items offered by Greenery that are included in
this order

{% enddocs %}

{% docs int_orders_is_on_time_delivery %}

whether product was delivered after estimated delivery time

{% enddocs %}
```

models/docs/intermediate

_int_orders_models.yml

course-dbt > greenery > models > intermediate > orders > _int_orders_models.yml

```
models:
  - name: int_orders_joined_to_addresses_promos
    columns:
      - name: shipping_cost
        data_tests:
        tags: [dbt_utils]
      - name: promo_id
        description: '{{ doc("promos_promo_id") }}'
        data_tests:
          - accepted_values:
              values:
                - 'instruction set'
                - Optional
                - Mandatory
                - Digitized
                - 'task-force'
                - leverage
          - dbt_expectations.expect_column_distinct_count_to_equal_othe
              compare_model: ref('stg_postgres_promos')
              compare_column_name: promo_id
              row_condition: "promo_id is not null"
        tags: [dbt_expectations]
      - name: discount
        description: '{{ doc("promos_discount") }}'
        data_tests:
          - positive_values
          - dbt_utils.expression_is_true:
              expression: ">0"
              where: "promo_id IS NOT NULL"
        tags: [dbt_utils]
      - name: order_total
        description: '{{ doc("orders_order_total") }}'
        data_tests:
          - positive_values
          - dbt_utils.expression_is_true:
              expression: "=ROUND(order_cost+shipping_cost, 2)"
              where: "promo_id IS NULL"
          - dbt_utils.expression_is_true:
              expression: "<ROUND(order_cost+shipping_cost, 2)"
              where: "promo_id IS NOT NULL"
        tags: [dbt_utils]
      - name: total_order_size
        description: '{{ doc("int_orders_total_order_size") }}'
        data_tests:
          - positive_values
      - name: num_unique_products
        description: '{{ doc("int_orders_num_unique_products") }}'
        data_tests:
          - positive_values
```

models/intermediate

Documentation was Prepared Using `dbt docs blocks` for Maintainability

EXPLORER

- course-dbt
  - .vscode
  - answers
  - greenery
    - analyses
    - dbt_packages
    - logs
    - macros
    - models
      - docs
        - intermediate
          - orders
            - .gitkeep
            - int_orders_docs.md
        - products
        - .gitkeep
        - marts
        - staging
      - intermediate
        - orders
          - _int_orders_models.yml
          - .gitkeep
          - int_orders_joined_to_addresses_promos.sql
        - products
        - marts
        - staging
      - overview.md

int_orders_docs.md

course-dbt > greenery > models > docs > intermediate > orders > int_orders_docs.md

```
{% docs int_orders_description %}

user orders combined with metadata for address and promotions

{% enddocs %}

{% docs int_orders_total_order_size %}

Total number of units of the product in this order

{% enddocs %}

{% docs int_orders_num_unique_products %}

number of uniqie items offered by Greenery that are included in
this order

{% enddocs %}

{% docs int_orders_is_on_time_delivery %}

whether product was delivered after estimated delivery time

{% enddocs %}
```

_int_orders_models.yml

course-dbt > greenery > models > intermediate > orders > _int_orders_models.yml

```
models:
  - name: int_orders_joined_to_addresses_promos
    columns:
      - name: shipping_c
        data_tests:
          tags: [dbt_utils
      - name: promo_id
        description: '{{ doc("promos_promo_id") }}'
        data_tests:
          - accepted_values:
              values:
                - 'instruction set'
                - Optional
                - Mandatory
                - Digitized
                - 'task-force'
                - leverage
          - dbt_expectations.expect_column_distinct_count_to_equal_othe
              compare_model: ref('stg_postgres_promos')
              compare_column_name: promo_id
              row_condition: "promo_id is not null"
        tags: [dbt_expectations]
      - name: discount
        description: '{{ doc("promos_discount") }}'
        data_tests:
          - positive_values
          - dbt_utils.expression_is_true:
              expression: ">0"
              where: "promo_id IS NOT NULL"
        tags: [dbt_utils]
      - name: order_total
        description: '{{ doc("orders_order_total") }}'
        data_tests:
          - positive_values
          - dbt_utils.expression_is_true:
              expression: "=ROUND(order_cost+shipping_cost, 2)"
              where: "promo_id IS NULL"
          - dbt_utils.expression_is_true:
              expression: "<ROUND(order_cost+shipping_cost, 2)"
              where: "promo_id IS NOT NULL"
        tags: [dbt_utils]
      - name: total_order_size
        description: '{{ doc("int_orders_total_order_size") }}'
        data_tests:
          - positive_values
      - name: num_unique_products
        description: '{{ doc("int_orders_num_unique_products") }}'
        data_tests:
          - positive_values
```

this was defined in `models/docs/staging/stg_postgres_events.md` and is re-used here

`models/intermediate`

Custom-Written Macros

Re-usability across **`dbt`** models (to partially render te model)

Re-Usable Macros - Part 1/2

Macros were written to be re-used across multiple models and other macros.

Two macros shown here were each used by two `dbt` models

1. `add_columns`
2. `get_conversion_rates`

Description and Arguments are documented

greenery.add_columns macro

Description Arguments Referenced By Code

Description

calculate the row-wise sum of two columns

Arguments

ARGUMENT

column_1

column_2

col_name_new

Referenced By

Models

int_events_sessions_aggregated_to_product
fct_products

Models using `add_columns` macro

greenery.get_conversion_rates macro

Description Arguments Referenced By Code

Description

get conversion rate by product or overall

Arguments

ARGUMENT

agg_type

Referenced By

Models

fct_conversion_rate
fct_products_conversion_rates

Models using `get_conversion_rate` macro

Custom-Written Macros

Re-usablity across **`dbt`** macros

Re-Usable Macros - Part 2/2

Macros were written to be re-used across multiple models and other macros.

Two macros shown here were each used by two `dbt` models

1. `add_columns`
2. `get_conversion_rates`

the `label_status` macro invokes another macro

greenery.label_status macro

Description | Arguments | Referenced By | Depends On | Code

ship_status	string

Referenced By

Models

fct_user_orders

Depends On

Macros

case_when

Code

Source

```
{%- macro label_status(ship_status) %}
{%- if ship_status == 'delivered' %}
{%- set outcome='delivered' -%}
{%- else %}
{%- set outcome='shipping' -%}
{%- endif -%}

{%- set session_col_name="num_orders_{}".format(outcome) -%}
{%- set status_filter="status='{}'".format(ship_status) -%}

SUM({{ case_when(status_filter, 1, 0, '') }}) AS {{ session_col_name }}
{%- endmacro -%}
```

Custom-Written Macros

Re-usability within same **`dbt`** model, with different arguments

```
9  products_purchase_sessions AS (
10     SELECT session_id,
11            product_id,
12            event_type
13     FROM dev_db.dbt_elsdes3gmailcom.int_product_purchases_filtered
14     -- (ADDED) get events showing the ID of the purchased product
15     WHERE product_id IS NOT NULL
16 ),
17 /* count number of sessions not ending in a purchase in which product page was
18 viewed */
19 product_non_purchase_page_views AS (
20     SELECT product_id,
21            COUNT(*) AS num_non_purchase_page_views,
22            COUNT(DISTINCT(session_id)) AS num_non_purchase_page_view_sessions
23     FROM products_non_purchase_sessions
24     -- get page view event in product non-purchase sessions
25     WHERE event_type = 'page_view'
26     GROUP BY product_id
27 ),
28 /* count number of sessions ending in a purchase in which product page was
29 viewed */
30 product_purchase_page_views AS (
31     SELECT product_id,
32            COUNT(*) AS num_purchase_page_views,
33            COUNT(DISTINCT(session_id)) AS num_purchase_page_view_sessions
34     FROM products_purchase_sessions
35     -- get page view event in product purchase sessions
36     WHERE event_type = 'page_view'
37     GROUP BY product_id
38 ),
39 /* count purchases */
40 product_purchases AS (
41     SELECT product_id,
42            COUNT(DISTINCT(session_id)) AS num_purchases
43     FROM products_purchase_sessions
44     -- get add-to-cart events since only products in a cart can be purchased
45     WHERE event_type = 'add_to_cart'
46     GROUP BY product_id
```

Excerpt from `intermediate/products/overall/int_events_sessions_aggregated_to_product` model

Without Macro

OBJECTIVE

Count page view sessions ending and not ending in a purchase, and number of purchases per product

Line Numbers 18-26, 29-37, 39-46
= 25 lines

With Macro

OBJECTIVE

Count page view sessions ending and not ending in a purchase, and number of purchases per product

```
9   products_purchase_sessions AS (
10      SELECT session_id,
11             product_id,
12             event_type
13      FROM {{ ref('int_product_purchases_filtered') }}
14      -- (ADDED) get events showing the ID of the purchased product
15      WHERE product_id IS NOT NULL
16  ),
17  /* count number of sessions not ending in a purchase in which product page was
18  viewed */
19  product_non_purchase_page_views AS (
20      {{ count_purchases_views_by_product(
21          'page_view', 'products_non_purchase_sessions', 'product_id'
22      ) }}
23  ),
24  /* count number of sessions ending in a purchase in which product page was
25  viewed */
26  product_purchase_page_views AS (
27      {{ count_purchases_views_by_product(
28          'page_view', 'products_purchase_sessions', 'product_id'
29      ) }}
30  ),
31  /* count purchases */
32  product_purchases AS (
33      {{ count_purchases_views_by_product(
34          'add_to_cart', 'products_purchase_sessions', 'product_id'
35      ) }}
    ),
```

macro is re-used, but with different argument

Line Numbers 18-22, 25-29, 31-35
= 15 lines
(40% reduction with macro)

Re-Use of same macro to calculate all three metrics

Excerpt from `intermediate/products/overall/int_events_sessions_aggregated_to_product` model

Custom-Written Macros

Re-usability across **`dbt`** models (to fully render the model)

By Product

Calculate conversion rate by product using `get_conversion_rate` macro

fct_products_conversion_rates table

Details Description Columns Referenced By Depends On Code

```
        ),
        /* get conversion rate */
        product_conversion_rates AS (
            SELECT product_id,
                   (
                       num_non_purchase_page_view_sessions
                       +num_purchase_page_view_sessions
                   ) AS total_num_page_view_sessions,
                   num_purchases AS num_purchase_sessions,
                   (
                       100*num_purchase_sessions/total_num_page_view_sessions
                   ) AS conversion_rate
            FROM dev_db.[redacted].int_events_sessions_aggregated_to_product
        ),
        /* get first and last event timestamp */
        event_timestamp_bounds AS (
            SELECT
                   product_id,
                   MIN(created_at) AS first_event,
                   MAX(created_at) AS last_event
            FROM dev_db.[redacted].stg_postgres_events
            GROUP BY product_id
        ),
        /* combine conversion rates and event timestamp bounds */
        conversion_rates_timestamp_bounds AS (
            SELECT p.product_name,
                   c.total_num_page_view_sessions,
                   c.num_purchase_sessions,
                   c.conversion_rate,
                   b.first_event,
                   b.last_event
            FROM product_conversion_rates c
            INNER JOIN event_timestamp_bounds b USING (product_id)
            INNER JOIN products p USING (product_id)
            ORDER BY conversion_rate DESC
        )
        SELECT *
        FROM conversion_rates_timestamp_bounds
    )
)
SELECT *
FROM conversion_rates
```

Without Macro (49 lines)

implements calculation by product

Depends On

Models Macros

stg_postgres_products
int_events_sessions_aggregated_to_product
stg_postgres_events

Code

Source Compiled

```
WITH conversion_rates AS (
    SELECT *
    FROM (
        {{ get_conversion_rates('product') }}
    )
)
SELECT *
FROM conversion_rates
```

With Macro (8 lines, 84% reduction)

Overall

Calculate overall conversion rate using `get_conversion_rate` macro

fct_conversion_rate table

Details | Description | Columns | Referenced By | Depends On | Code

```
4       WITH products AS (
5           SELECT 1 AS product_id,
6                  'all' AS product_name
7       ),
8       /* get conversion rate */
9       overall_conversion_rates AS (
10          SELECT 1 AS product_id,
11                 SUM(
12                     num_non_purchase_page_view_sessions
13                     +num_purchase_page_view_sessions
14                 ) AS total_num_page_view_sessions,
15                 SUM(num_purchases) AS num_purchase_sessions,
16                 (
17                     100*num_purchase_sessions/total_num_page_view_sessions
18                 ) AS conversion_rate
19          FROM dev_db.[redacted]int_events_sessions_aggregated_to_product
20      ),
21      /* get first and last event timestamp */
22      event_timestamp_bounds AS (
23          SELECT 1 AS product_id,
24                 MIN(created_at) AS first_event,
25                 MAX(created_at) AS last_event
26          FROM dev_db[redacted]stg_postgres_events
27
28      ),
29      /* combine conversion rates and event timestamp bounds */
30      conversion_rates_timestamp_bounds AS (
31          SELECT p.product_name,
32                 c.total_num_page_view_sessions,
33                 c.num_purchase_sessions,
34                 c.conversion_rate,
35                 b.first_event,
36                 b.last_event
37          FROM overall_conversion_rates c
38          INNER JOIN event_timestamp_bounds b USING (product_id)
39          INNER JOIN products p USING (product_id)
40          ORDER BY conversion_rate DESC
41      )
42      SELECT *
43      FROM conversion_rates_timestamp_bounds
44  )
45  )
46  SELECT *
47  FROM conversion_rates
```

placeholder

overall calculation without grouping over product

Depends On

Models | Macros

stg_postgres_events
int_events_sessions_aggregated_to_product

Code

Source | Compiled

```
1  WITH conversion_rates AS (
2      SELECT *
3      FROM (
4          {{ get_conversion_rates('overall') }}
5      )
6  )
7  SELECT *
8  FROM conversion_rates
```

Without Macro (46 lines)

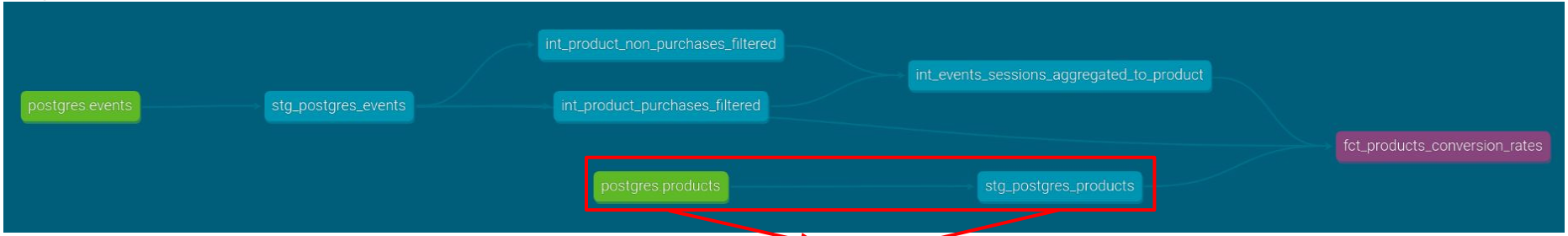
With Macro (8 lines, 82% reduction)

Impact of Macro on Conversion Rate Model DAGs

The use of a single macro to render both data models keeps their DAGs identical to each other, with only a single branch controlling the granularity of the calculation of the conversion rate

- Since the same metric is being calculated, it is intuitive that the DAG should only differ in the level at which the metric must be reported; all other aspects of the DAG can be the same
- The re-use of this single macro greatly helped to **simplify these two model DAGs**

By Product



Single branch adds grouping level at which conversion rate is reported

Overall

Summary: Aggregated Impact of Macro on Models

1. 9 `dbt` macros were written and they reduced a total of 171 lines of SQL to 64 lines
2. Macros were used in 5 of 7 `marts` models and in 5 of 7 `intermediate` models
3. 7 of 9 macros reduced the number of lines of SQL, 2 of 9 macros increased the number of lines
4. 8 of 9 macros were used in data models and 1 of the 9 macros was called by another macro
5. 2 of 9 macros were each used in 2 different data models

`dbt` Tests

Testing Infrastructure

Built-in, Custom Tests and External Packages

Built-in and Custom-Written Tests

To start testing data models, built-in and custom (`generic`) `dbt` tests were used

Built-in tests

1. `not_null` to ensure unexpected missing values were not found in a column
2. `unique` to ensure specific columns contained only unique values

Custom tests (see next page for code and models using these tests)

1. `order_before_delivery` to test that one timestamp column was chronologically earlier than another
2. `positive_values` to ensure values in a column were positive (> 0)

greenery.test_order_before_delivery macro

used in 8 tests

Description Arguments Referenced By Code

This macro is not currently documented

Arguments

Details are not available for this macro. This may be due to the fact that this macro doesn't have any arguments or t

Referenced By

Data Tests

order_before_delivery_fct_user_orders_first_order_date__last_order_date
order_before_delivery_stg_postgres_orders_created_at__delivered_at
order_before_delivery_stg_postgres_orders_created_at__estimated_delivery_at
order_before_delivery_stg_postgres_users_created_at__updated_at
order_before_delivery_int_orders_joined_to_addresses_promos_created_at__delivered_at
order_before_delivery_int_orders_joined_to_addresses_promos_created_at__estimated_delivery_at
order_before_delivery_fct_orders_created_at__delivered_at
order_before_delivery_fct_orders_created_at__estimated_delivery_at

Code

Source

test that two columns are in chronological order

```
{% test order_before_delivery(model, column_name, field) %}

    select *
    from {{ model }}
    where {{ field }} < {{ column_name }}


{% endtest %}
```

greenery.test_positive_values macro

used in 53 tests

Description Arguments Referenced By Code

positive_values_fct_products_rank_purchases
positive_values_fct_products_avg_time_on_page_seconds
positive_values_int_products_purchase_abandoned_cart_sessions_summed_num_carts
positive_values_int_events_sessions_aggregated_to_product_num_non_purchase_page_view_sessions
positive_values_int_events_sessions_aggregated_to_product_num_purchase_page_view_sessions
positive_values_int_events_sessions_aggregated_to_product_num_purchases
positive_values_int_events_sessions_aggregated_to_product_num_page_views
positive_values_int_products_page_viewing_time_averaged_avg_time_on_page_seconds
positive_values_int_products_page_viewing_time_averaged_std_time_on_page_seconds
positive_values_int_orders_joined_to_addresses_promos_order_cost
positive_values_int_orders_joined_to_addresses_promos_shipping_cost
positive_values_int_orders_joined_to_addresses_promos_discount
positive_values_int_orders_joined_to_addresses_promos_order_total
positive_values_int_orders_joined_to_addresses_promos_total_order_size
positive_values_int_orders_joined_to_addresses_promos_num_unique_products
positive_values_int_products_daily_totals_num_page_views
positive_values_int_products_daily_totals_num_sessions
positive_values_int_products_daily_totals_num_orders
positive_values_fct_orders_order_cost
positive_values_fct_orders_discount
positive_values_fct_orders_order_total
positive_values_fct_orders_total_order_size
positive_values_fct_orders_num_unique_products
positive_values_fct_orders_delivery_time_seconds
positive_values_fct_orders_delivery_delay_seconds

partial list of tests

Code

Source

test that column values are positive

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

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


{% endtest %}
```

External Package - `dbt-expectations`

External `dbt` packages were used to perform more complex tests on data models

`dbt-expectations` was the main package used for testing data models

The following macros from this package were used

1. `expect_table_columns_to_match_ordered_list`
2. `expect_column_distinct_count_to_equal_other_table`
3. `expect_column_values_to_be_of_type`
4. `expect_table_row_count_to_equal`
5. `expect_column_values_to_match_regex`
6. `expect_column_values_to_match_like_pattern`
7. `expect_column_value_lengths_to_equal`

External Package - `dbt-utils`

`dbt-utils` was another `dbt` package

It was used to run custom SQL to conditionally test the existence of expected relationships between multiple columns in the same model

The following macro from this package were used

1. `exppression_is_true`

Test Suite Usage

Aggregated totals per layer and per model

Aggregated Usage of `dbt` Model Tests

Below are the total number of tests run on each type of model (`staging`, `intermediate` and `marts`)

Type of Test	`staging`	`intermediate`	`marts`	total
`built-in and custom`	53	61	87	201
`dbt-expectations`	37	22	35	94
`dbt-utils`	5	5	19	29

Aggregated Impact of Tests on `staging` Models (89% tested)

Below are the total number of tested and untested columns for **staging** models

Name	Untested Columns	Tested Columns
stg_postgres_users	1	8
stg_postgres_addresses	0	5
stg_postgres_promos	0	3
stg_postgres_products	0	4
stg_postgres_orders	3	13
stg_postgres_order_items	0	3
stg_postgres_events	1	8

Aggregated Impact of Tests on `intermediate` Models (100% tested)

Below are the total number of tested and untested columns for **intermediate** models

Name	Untested Columns	Tested Columns
`int_product_purchases_filtered`	0	7
`int_product_non_purchases_filtered`	0	7
`int_events_sessions_aggregated_to_product`	0	5
`int_products_page_viewing_time_averaged`	0	3
`int_products_purchase_abandoned_cart_sessions_summed`	0	2
`int_products_daily_total`	0	2
`int_ordres_joined_to_addresses_promos`	0	15

Aggregated Impact of Tests on `marts` Models (100% tested)

Below are the total number of tested and untested columns for **marts** models

Name	Untested Columns	Tested Columns
fct_orders	0	15
fct_conversion_rates	0	6
fct_user_orders	0	12
fct_promo_orders	0	8
fct_products	0	10
fct_products_conversion_rates	0	6
fct_products_daily	0	4

Summary: Aggregated Impact of Tests on Models

1. 100% of intermediate and marts model columns were tested
2. ~89% of staging model columns were tested
3. Total of 324 tests were invoked, across all models
4. ~200 built-in or custom tests were invoked
5. 95 dbt-expectations tests were invoked
6. the most invoked test was a custom-written dbt test `positive_values`, which was used to ensure positive values existed across a column

Impact of Tests on Model Accuracy

Model errors flagged by writing tests

Column in `fct_orders` Created to Calculate Delay Seconds

```sql
SELECT delivered_at,
       estimated_delivery_at,
       DATEDIFF(
           second, created_at, estimated_delivery_at
       ) AS estimated_delivery_time_seconds,
       datediff(second, created_at, delivered_at) AS delivery_time_seconds,
       (
           CASE
               WHEN delivered_at > estimated_delivery_at
               THEN ABS(
                   DATEDIFF(second, delivered_at, estimated_delivery_at)
               )
               ELSE NULL
           END
       ) AS delivery_delay_seconds
FROM stg_postgres_orders
```

Expected Values for `delivery_delay_seconds`

If an order is delayed, then the calculated `delivery_delay_seconds` is non-`NULL` and correctly contains the delivery delay in seconds

If an order is on-time, then the calculated value is `NULL`

Tests Implemented for Expected Values in `delivery_delay_seconds`

Below are the three `dbt` tests that were implemented to ensure that values in the delivery delay column were expected

```yaml
- name: delivery_delay_seconds
  - dbt_utils.expression_is_true:
      expression: "IS NULL"
      where: "delivered_at < estimated_delivery_at"
  - dbt_utils.expression_is_true:
      expression: "IS NULL"
      where: "delivery_time_seconds < estimated_delivery_time_seconds"
  - dbt_utils.expression_is_true:
      expression: "IS NULL"
      where: "delivery_time_seconds IS NOT NULL"
```

Sample Data to Assess Test Outcomes

```sql
SELECT ...
FROM stg_postgres_orders
WHERE delivered_at IN ('2021-02-17 23:30:34', '2021-02-13 15:13:09')
OR estimated_delivery_at IN ('2021-02-14 23:35:14', '2021-02-16 07:08:04')
OR order_id = '8385cfcd-2b3f-443a-a676-9756f7eb5404'
```

delivered_at	estimted_delivery_at	estimated_delivery_time_seconds	delivery_time_seconds	delivery_delay_seconds
2021-02-17 23:30:34	2021-02-12	86400	518400	432000
2021-02-13 15:13:09	2021-02-16 15:13:09	432000	172800	NULL
NULL	2021-02-12 10:15:26	53078169	NULL	NULL
2021-02-15 23:35:14	2021-02-12 23:35:14	259200	345600	86400
NULL	2021-02-12 07:08:04	432000	NULL	NULL

If an order is not yet delivered…

If an order is not yet delivered, then the `delivered_at` column is `NULL` and **the first test passes** since the < operator only compares non-NULL values **but** this `NULL` does not mean the order was delivered on time (as expected from page 37)

delivered_at	estimted_delivery_at	estimated_delivery_time_seconds	delivery_time_seconds	delivery_delay_seconds
2021-02-17 23:30:34	2021-02-12	86400	518400	432000
2021-02-13 15:13:09	2021-02-16 15:13:09	432000	172800	NULL
NULL	2021-02-12 10:15:26	53078169	NULL	NULL
2021-02-15 23:35:14	2021-02-12 23:35:14	259200	345600	86400
NULL	2021-02-12 07:08:04	432000	NULL	NULL

If an order is not yet delivered…

`delivery_delay_seconds` column is `NULL` and **the second test passes** as expected **but** this `NULL` also does not mean the order was delivered on time (as expected from page 37)

delivered_at	estimted_delivery_at	estimated_delivery_time_seconds	delivery_time_seconds	delivery_delay_seconds
2021-02-17 23:30:34	2021-02-12	86400	518400	432000
2021-02-13 15:13:09	2021-02-16 15:13:09	432000	172800	NULL
NULL	2021-02-12 10:15:26	53078169	NULL	NULL
2021-02-15 23:35:14	2021-02-12 23:35:14	259200	345600	86400
NULL	2021-02-12 07:08:04	432000	NULL	NULL

If an order is not yet delivered…

If an order is not yet delivered, then the `delivery_time_seconds` column is `NULL` and **the third test fails** since the CASE WHEN logic

1. expects this column to only be NULL for on-time deliveries
2. does not expect this column to be NULL in other cases (such as this one)

delivered_at	estimted_delivery_at	estimated_delivery_time_seconds	delivery_time_seconds	delivery_delay_seconds
2021-02-17 23:30:34	2021-02-12	86400	518400	432000
2021-02-13 15:13:09	2021-02-16 15:13:09	432000	172800	NULL
NULL	2021-02-12 10:15:26	53078169	NULL	NULL
2021-02-15 23:35:14	2021-02-12 23:35:14	259200	345600	86400
NULL	2021-02-12 07:08:04	432000	NULL	NULL

Summary: Impact of Tests on Model Accuracy

NULLs are occurring in `delivery_delay_seconds` under the wrong scenarios

The implementation of tests using the `dbt-expectations` package have identified a flaw in the implementation of determining the order delivery delay

Future work should focus on fixing this error or excluding this column from the `fct_orders` model