metric:  
In dbt (data build tool), a metric refers to a measurable value or statistic that provides insights into the performance or behavior of a system or process. Metrics are used to track and evaluate the success, efficiency, and quality of various operations within a data warehouse or analytics environment.

By using dbt, businesses can easily define and calculate metrics that are essential to their operations. Metrics like revenue, user engagement, customer churn, and product adoption are some examples that businesses track to understand their performance. By using dbt, these metrics can be reliably and consistently tracked over time, enabling organizations to make data-driven decisions.

Like exposures, metrics appear as nodes in the directed acyclic graph (DAG) and can be expressed in YAML files. Defining metrics in dbt projects encodes crucial business logic in tested, version-controlled code. Further, you can expose these metrics definitions to downstream tooling, which drives consistency and precision in metric reporting.

**Defining a metric**[**​**](https://docs.getdbt.com/docs/build/metrics#defining-a-metric)

You can define metrics in .yml files nested under a metrics: key. Metric names

. Metrics are the first step towards this vision, allowing users to version control and centrally define their key business metrics in a single repo while also serving them to the entire business.

First of all the we should have a dbt model containing the table and that table is intended to be performed with several quality checks and operations.

We would have to define a model and then metric.yml and configure it according to our needs.

Table:

| **event\_date** | **order\_id** |  | **order\_country** | **order\_status** | **customer\_id** | **customer\_status** | **amount** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2022-10-01 | 1 |  | United States | completed | 19 | Healthy | 10 |
| 2022-10-01 | 2 |  | France | completed | 36 | Churn Risk | 15 |
| 2022-10-02 | 2 |  | France | returned | 36 | Churned | 15 |
| 2022-10-02 | 3 |  | Turkey | completed | 20 | Healthy | 80 |
| 2022-10-03 | 4 |  | Korea | Completed | 14 | Churn Risk | 24 |

version: 2

metrics:

  - name: total\_revenue #assign a short and unqiue name for the metric

    label: The revenue of our business # label can be a bit detail of it

    model: ref('order\_events') # it is the model of which metric is to be find.

    description: "The revenue for our business, as defined by Jerry in Finance" # bried description about the project

    calculation\_method: sum # what kind of calculation to be performed (sum,min,max,avg,median,derive)

    expression: (amount) # the column on which operation is to be performed

    timestamp: event\_date # it is the time stamp of which we will be filtering for the results

    time\_grains: [day, week, month, year, all\_time] # it is the time frame defined and we could filter it by the time defined here

    dimensions:

      - customer\_status # these are the dimensions we will be having in the final table

      - order\_country

Thoughtful curation of dimensions is essential for three main reasons:

* **Relevance:** Analysts must include the dimensions most relevant to answering the question.
* **Trust**: Curating high-quality dimensions with little to no known errors helps ensure trust in analysis results and the decisions that follow.
* **Efficiency**: Curation provides a faster path to high-quality analysis results.

To fetch it through the model:

{{config(materialized='table')}}

with order\_events as

(

select \* from {{

    metrics.calculate (

        metric(' total\_revenue '),

        grain = 'year',

        dimensions = ['Customer\_status','order\_country']

    )

}}

      )

select \* from order\_events

Available properties[​](https://docs.getdbt.com/docs/build/metrics#available-properties)

Metrics can have many declared **properties**, which define aspects of your metric. More information on [properties and configs can be found here](https://docs.getdbt.com/reference/configs-and-properties).

| **Field** | **Description** | **Example** | **Required?** | |
| --- | --- | --- | --- | --- |
| name | A unique identifier for the metric | new\_customers | | yes |
| model | The dbt model that powers this metric | dim\_customers | yes (no for derived metrics) | |
| label | A short for name / label for the metric | New Customers | yes | |
| description | Long form, human-readable description for the metric | The number of customers who.... | no | |
| calculation\_method | The method of calculation (aggregation or derived) that is applied to the expression | count\_distinct | yes | |
| expression | The expression to aggregate/calculate over | user\_id, cast(user\_id as int) | yes | |
| timestamp | The time-based component of the metric | signup\_date | no | |
| time\_grains | One or more "grains" at which the metric can be evaluated. For more information, see the "Custom Calendar" section. | [day, week, month, quarter, year] | no | |
| dimensions | A list of dimensions to group or filter the metric by | [plan, country] | no | |
| window | A dictionary for aggregating over a window of time. Used for rolling metrics such as 14 day rolling average. Acceptable periods are: [day,week,month, year, all\_time] | {count: 14, period: day} | no | |
| filters | A list of filters to apply before calculating the metric | See below | no | |
| config | [Optional configurations](https://github.com/dbt-labs/dbt_metrics#accepted-metric-configurations) for calculating this metric | {treat\_null\_values\_as\_zero: true} | no | |
| meta | Arbitrary key/value store | {team: Finance} | no | |

Available calculation methods[​](https://docs.getdbt.com/docs/build/metrics#available-calculation-methods)

The method of calculation (aggregation or derived) that is applied to the expression.

| **Metric Calculation Method** | **Description** |
| --- | --- |
| count | This metric type will apply the count aggregation to the specified field |
| count\_distinct | This metric type will apply the count aggregation to the specified field, with an additional distinct statement inside the aggregation |
| sum | This metric type will apply the sum aggregation to the specified field |
| average | This metric type will apply the average aggregation to the specified field |
| min | This metric type will apply the min aggregation to the specified field |
| max | This metric type will apply the max aggregation to the specified field |
| median | This metric type will apply the median aggregation to the specified field, or an alternative percentile\_cont aggregation if median is not available |
| derived | This metric type is defined as any non-aggregating calculation of 1 or more metrics |

derive:  
In the context of metrics in dbt, the term "derive" refers to the calculation or derivation of a new metric based on existing columns or metrics in your data models. It allows you to create new calculated metrics by applying mathematical operations, aggregations, or transformations to existing data.

When you define a derived metric in dbt, you specify the calculation method and expression that determines how the new metric is derived. The calculation method can be an aggregation function like sum, average, count, or a custom SQL expression that combines columns or applies transformations.

By using derived metrics, you can create meaningful indicators or measurements that provide insights into your data. For example, you might calculate metrics such as total sales, average revenue per customer, or conversion rates based on existing data points.

for derived metrics (previously named expression), which are defined as non-aggregating calculations of 1 or more metrics. An example of this would be {{metric('total\_revenue')}} / {{metric('count\_of\_customers')}}.

By defining these metrics, you are able to create metrics like:

* ratios
* subtractions
* any arbitrary calculation

As long as the two (or more) base metrics (metrics that comprise the derived metric) share the specified time\_grains and dimensions, those attributes can be used in any downstream metrics macro.

An example definition of an derived metric is:

*# models/marts/product/schema.yml*  
version: 2  
  
models:  
 - name: dim\_customers  
metrics:  
 - name: average\_revenue\_per\_customer  
 label: Average Revenue Per Customer  
 description: "The average revenue received per customer"  
  
 calculation\_method: derived  
 expression: "{{metric('total\_revenue')}} / {{metric('count\_of\_customers')}}"  
  
 timestamp: order\_date  
 time\_grains: [day, week, month, quarter, year, all\_time]  
 dimensions:  
 - had\_discount  
 - order\_country

Another example

Model defined:

Metric configuration:

version: 2

metrics:

  - name: average\_unit\_price

    model: ref('sales')

    label: average amount unit price

    description: "The average amount of unit price"

    calculation\_method: average

    expression: Sale\_amt

    timestamp: orderdate

    time\_grains: [day, week, month, quarter, year, all\_time]

    dimensions:

      - Region

      - item

      - SalesMan

    filters:

    - field: Unit\_price #column name

      operator: '>=' # operation we want to perform

      value: "500" # conditional value to which against the filteration is performed

    - field: Units

      operator: '>='

      value: '50'

  # general properties

    config:

        enabled: true # after enabling the metric config, then it will consider it as a part of the dbt project

        treat\_null\_values\_as\_zero: true # the null values are replaced with 0

**Calendar**[**​**](https://docs.getdbt.com/docs/build/metrics#calendar)

The dbt\_metrics package contains a [basic calendar table](https://github.com/dbt-labs/dbt_metrics/blob/main/models/dbt_metrics_default_calendar.sql) that is created as part of your dbt run. It contains dates between 2010-01-01 and 2029-12-31.

If you want to use a custom calendar, you can replace the default with any table which meets the following requirements:

* Contains a date\_day column.
* Contains the following columns: date\_week, date\_month, date\_quarter, date\_year, or equivalents.
* Additional date columns need to be prefixed with date\_, e.g. date\_4\_5\_4\_month for a 4-5-4 retail calendar date set. Dimensions can have any name (see following section).

To do this, set the value of the dbt\_metrics\_calendar\_model variable in your dbt\_project.yml file:

*#dbt\_project.yml*  
config-version: 2  
[...]  
vars:  
 dbt\_metrics\_calendar\_model: my\_custom\_calendar

**Dimensions from calendar tables**[**​**](https://docs.getdbt.com/docs/build/metrics#dimensions-from-calendar-tables)

You may want to aggregate metrics by a dimension in your custom calendar table, for example is\_weekend. You can include this within the list of dimensions in the macro call without it needing to be defined in the metric definition.

To do so, set a list variable at the project level called custom\_calendar\_dimension\_list, as shown in the example below.

*#dbt\_project.yml*  
vars:  
 custom\_calendar\_dimension\_list: ["is\_weekend"]

Let me provide a step-by-step explanation of how to create and use a custom calendar in dbt:

1. **Create a custom calendar table:**

You need to create a table in your database that represents your custom calendar. This table should have the necessary columns mentioned earlier: ***date\_day****,* ***date\_week****,* ***date\_month****,* ***date\_quarter****,* ***date\_year****,* and any additional date columns prefixed with ***date\_***. Populate this table with the relevant dates and corresponding time periods according to your business calendar.

1. **Define the custom calendar table in dbt**:

In your dbt project, specify the name of your custom calendar table by setting the **dbt\_metrics\_calendar\_model** variable in the **dbt\_project.yml** file:

**dbt\_project.yml**

config-version: 2

[...]

**vars:**

dbt\_metrics\_calendar\_model: **my\_custom\_calendar**

*Make sure to replace* ***my\_custom\_calendar*** *with the name of your model in which you have defined the table in warehouse where the custom calendar has been*

Add custom dimensions (optional):

If you have additional dimensions in your custom calendar table that you want to use for aggregating metrics, you can define them in the **custom\_calendar\_dimension\_list** variable. For example, if you have an **is\_weekend** column in your custom calendar table, you can include it like this:

**dbt\_project.yml**

vars:

**custom\_calendar\_dimension\_list:** ["is\_weekend"]

1. **Configure metrics:** In your dbt project, define the metrics you want to calculate and configure their settings in a metric YAML file (e.g., **models/metrics.yml**). Specify the custom calendar-related settings within the **config** section of each metric. For example:

**version: 2**

**metrics:**

**- name: example\_metric**

**label: Example Metric**

**model: ref(' my\_custom\_calendar’)**

**calculation\_method: count**

**timestamp: date\_field**

**time\_grains: [day, week, month]**

**config:** enabled: true

In this example, **my\_custom\_calendar** is the name of the model containing the data used for the metric calculation. The **timestamp** parameter refers to the column in **my\_custom\_calendar** that represents the date. The **time\_grains** parameter specifies the desired time periods for aggregating the metric (e.g., day, week, month). The **config** section allows you to include additional configurations for the metric, such as enabling or disabling it.

1. Run dbt: Once you have configured your custom calendar and metrics, you can run dbt using the **dbt run** command. This will trigger the execution of your models and metrics, utilizing the custom calendar table and its associated dimensions for aggregating the metrics according to your business calendar.

By following these steps, you will be able to create and use a custom calendar in dbt for performing time-based analysis and aggregating metrics based on your specific business calendar requirements.

*Therefore, even though the metrics configuration code example didn't explicitly mention the custom calendar, once you have set the* ***dbt\_metrics\_calendar\_model*** *variable in the* ***dbt\_project.yml*** *file to your custom calendar table, dbt will automatically recognize and use that custom calendar for metrics calculations when you run* ***dbt run****.`*

You will have to create a table in warehouse for custom calendar and put all the columns with names ***date\_day****,* ***date\_week****,* ***date\_month****,* ***date\_quarter****,* ***date\_year****,* or anyother dimension or column that consists of date you want to add like is\_weekend. Then you will have to create a model and call that database in the model and save it,

Conclusion of Custom Calendar

After that insert some configurations in dbt\_project.yml and define a variable dbt\_metrics\_calendar\_model and put its value the name of the model as it is, **vars:** dbt\_metrics\_calendar\_model: **my\_custom\_calendar** it should be written as it is, so when the dbt is run the metric package will dbt will automatically recognize and use that custom calendar for metrics calculations when you run **dbt run**.`

Configuration[​](https://docs.getdbt.com/docs/build/metrics#configuration)

Metric nodes now accept config dictionaries like other dbt resources. Specify Metric configs in the metric yml itself, or for groups of metrics in the dbt\_project.yml file.

* Metric yml
* dbt\_project.yml

dbt\_project.yml

metrics:   
 your\_project\_name:   
 +enabled: true

.

https://docs.getdbt.com/docs/build/metrics