## VAR FUNCTION:

In dbt (Data Build Tool), the **var** function is used to access variables defined in your dbt project. Variables in dbt allow you to define values that can be reused across multiple models, configurations, or macros. The **var** function provides a way to reference and use these variables within your dbt code.

The **var** function takes the name of the variable as an argument and returns its value. You can use the **var** function in various dbt files, such as model files (SQL files), YAML configuration files, or macros.

Here's an example to illustrate the usage of the **var** function in dbt:

1. Defining a variable: In your dbt project, you can define variables in the **dbt\_project.yml** file or create separate variable files (e.g., **vars.yml**) in the **data** directory. Let's assume we have a variable called **my\_var** defined in the **vars.yml** file:

my\_var: 42

1. Referen**ci**ng the variable in a model file: In your model file (e.g., **my\_model.sql**), you can use the **var** function to access the value of the **my\_var** variable:

SELECT \* FROM my\_table WHERE column = {{ var('my\_var') }};

1. Referencing the variable in a YAML configuration file: You can also use the **var** function in your YAML configuration files to set dynamic values based on the defined variables. For example:

version: 2

models: - my\_model:

enabled: true

materialized: view sql: my\_model.sql tags: {{ var('my\_var') }}

In this example, the **tags** value for the **my\_model** configuration is set based on the value of the **my\_var** variable.

By utilizing the **var** function, you can easily reference and reuse variables throughout your dbt project, making your code more flexible and maintainable.

name: my\_dbt\_project  
version: 1.0.0  
  
config-version: 2  
  
vars:  
 *# The `start\_date` variable will be accessible in all resources*  
 start\_date: '2016-06-01'  
  
 *# The `platforms` variable is only accessible to resources in the my\_dbt\_project project*  
 my\_dbt\_project:  
 platforms: ['web', 'mobile']  
  
 *# The `app\_ids` variable is only accessible to resources in the snowplow package*  
 snowplow:  
 app\_ids: ['marketing', 'app', 'landing-page']  
  
models:  
 ...

## EXAMPLE:

1. Define the var anywhere in dbt\_project.yml – It is always defined in dbt\_project.yml so that it could be available to every model, test, source.

vars:

  my\_var: "THEFT"

1. now you can call it from anywhere using var function, it is same like the python variables which store any string, number or any information inside them. Now it will be used in the sql code

{{ config(tags=["crime"]) }}

with chicagocrime as (select \* from {{ source('src\_chicagocrime', 'chicagocrimedata') }})

select \*

from chicagocrime where PRIMARY\_TYPE = **'{{ var('my\_var') }}'**

so, the variable contains a text ‘theft’ and hence it will be used like to fetch all the data where the primary type of crime is theft.

When the var function is detected it directly goes to dbt\_project and see what is the variable defined and put that text in that variable where it is called or needed.

There can be many variables defined under the hood of vars and each variable must have the unique name and value assigned to it in the form of strings or integers

## **Defining variables on the command line**[**​**](https://docs.getdbt.com/docs/build/project-variables#defining-variables-on-the-command-line)

The dbt\_project.yml file is a great place to define variables that rarely change. Other types of variables, like date ranges, will change frequently. To define (or override) variables for a run of dbt, use the --vars command line option. In practice, this looks like:

$ dbt run --vars '{"key": "value"}'

The --vars argument accepts a YAML dictionary as a string on the command line. YAML is convenient because it does not require strict quoting as with [JSON](https://docs.getdbt.com/terms/json).

Both of the following are valid and equivalent:

$ dbt run --vars '{"key": "value", "date": 20180101}'  
$ dbt run --vars '{key: value, date: 20180101}'

If only one variable is being set, the brackets are optional, eg:

$ dbt run --vars 'key: value'

## **Variables Precedence (imp)**

1. Variables defined with the **--vars** command line argument: If you provide variables using the **--vars** command line argument when running dbt commands, those variables will have the highest priority. They will override any variables defined in the **dbt\_project.yml** file. These variables are globally scoped, meaning they can be accessed by all packages within your dbt project.
2. Package-scoped variable declaration in **dbt\_project.yml**: If you define variables within a specific package's configuration in the **dbt\_project.yml** file, those variables will have the second-highest priority. They are scoped to the respective package and can be accessed within that package.
3. Global variable declaration in **dbt\_project.yml**: Variables defined in the top-level scope of the **dbt\_project.yml** file (outside of any specific package's configuration) have the third-highest priority. They are accessible globally across all packages within your dbt project.
4. Variable's default argument: If a variable has a default argument defined, it will be used if no value is provided for that variable through the command line or in the **dbt\_project.yml** file. The default argument acts as a fallback value for the variable.

If dbt cannot find a definition for a variable after checking these four places in the specified order, it will raise a compilation error. This ensures that all variables used in your dbt project have valid and defined values, preventing potential issues and ensuring the reliability of your data transformations.

By understanding this order of precedence, you can control and override variables effectively based on your requirements, whether it's providing values through the command line, package-specific configurations, or global settings in the **dbt\_project.yml** file.

Problem statement:

select \* from events where event\_type = '{{ var("event\_type") }}'

***If you try to run this model without supplying an event\_type variable, you'll receive a compilation error that looks like this:***

**Encountered an error:**  
! Compilation error while compiling model package\_name.my\_model:  
! Required var 'event\_type' not found in config:  
Vars supplied to package\_name.my\_model = {  
}

Inorder to tackle this error you should have to define the var\_function or variable in the var. or the other method is:

Variable default values[​](https://docs.getdbt.com/reference/dbt-jinja-functions/var#variable-default-values)

The **var()** function takes an optional second argument, **default**. If this argument is provided, then it will be the default value for the variable if one is not explicitly defined.

*-- Use 'activation' as the event\_type if the variable is not defined.*

select \* from events where event\_type = '{{ var("event\_type", "activation") }}'