

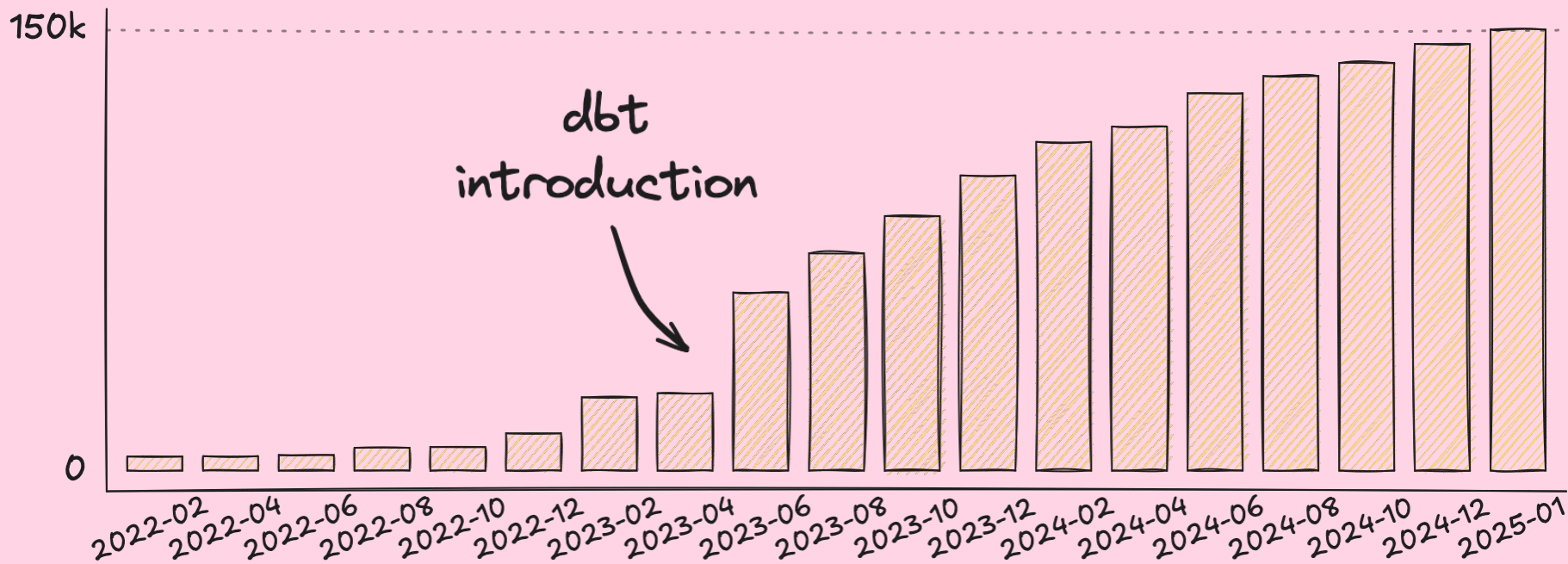


dbt-score

A linter for dbt model metadata



`dbt-score.picnic.tech`



YAML source lines of code



**Matthieu
Caneill**



Pixel & Joop
Coding assistants



Picnic
Online supermarket



**Jochem
van Dooren**





Pixel & Joop

Coding assistants

**Matthieu
Caneill**

R



Picnic
Online supermarket



**Jochem
van Dooren**





dbt-score

A linter for dbt model metadata

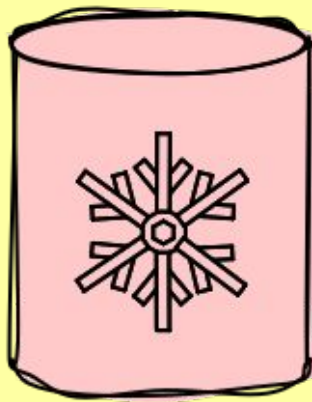
Takeaways

1. **Declarative** model properties are neat
2. Enforce their **consistency** through linting
3. **Customize** linting rules through simple **UX**

Outline

1. What is **dbt**?
2. The need for **linting**
3. **dbt-score** in the data stack

What is
dbt
?



Data

```
create table pizzas (  
  id int primary key,  
  cheese varchar  
)  
as  
select  
  id,  
  ingred  
from raw  
where ty
```

```
create table pizzerias (  
  id int primary key,  
  location varchar,  
  rating decimal  
)  
as  
select  
  id,  
  location,  
  avg(rating  
from ratings  
where type =  
group by id,
```

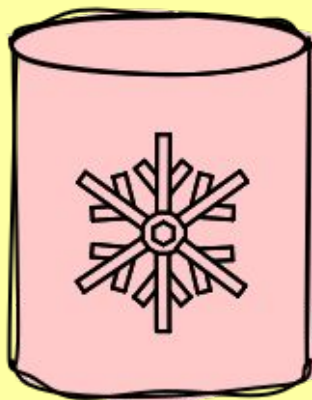
```
create table daily_sales (  
  id int primary key,  
  location varchar,  
  sale_date date,  
  amount decimal  
)  
as  
select  
  id,  
  location,  
  date(sale_datetime) as sale_date  
  sum(amount) as amount  
from ledger_lines  
group by id, location, sale_date
```



Notebook



Dashboard



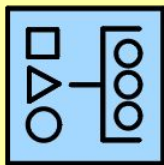
Data



Catalog



Data app



Pipelines

```

create
  ic
  ch
)
alter table pizzas
add column spiciness int
as
select

```

```

-- Fix 2024-09-19
insert into daily_sales
values (42, 'Amsterdam', 2024-09-18, 1632.0)

```

```

-- Manual fix prod data
update pizzas
set cheese = 'raclette'
where id = 12 or id = 13

```

group by id, location, sale_date

Orchestration
Out of order
Lost data
Manual input
Quality checks
PII
Performances



Notebook



Data

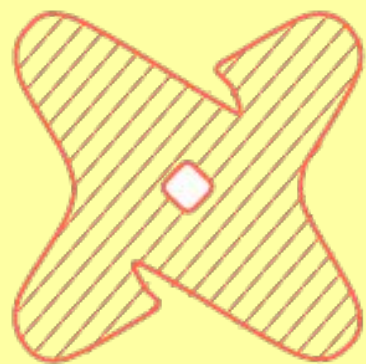


Data app

```
zas  
business int
```

```
(  
2024-09-18, 1632.0)  
table daily_sales (  
primary key,  
on varchar,  
ate date,  
decimal  
rod data
```

```
aclette'  
ale_datetime) as sale_date  
unt) as amount  
er lines  
id, location, sale_date
```



dbt

```
create table pizzas (  
  id int primary key,  
  cheese varchar  
)  
as  
select  
  id,  
  ingredient as cheese  
from raw_products  
where type = 'pizza'
```



Metadata



Logic

```
select
  id,
  ingredient as cheese
from raw_products
where type = 'pizza'
```

pizzas.sql

```
models:
  - name: pizzas
    columns:
      - name: id
        data_type: int
        constraints:
          - type: primary_key
      - name: cheese
        data_type: varchar
```

pizzas.yml

```
models:
- name: pizzas
  description: All the good pizzas
  config:
    materialized: incremental
    grants:
      select: [pizza_ceo]
  meta:
    owner: Pizzaiolo
    failure_alerts: true
    dashboard: pizzas_by_cheese
  columns:
- name: id
  description: Pizza PK
  data_type: int
  constraints:
    - type: primary_key
  data_tests:
    - unique
    - not_null
- name: cheese
  description: Main type of cheese
  data_type: varchar
  data_tests:
    - not_null
```

**Data models
are
first-class
citizens**

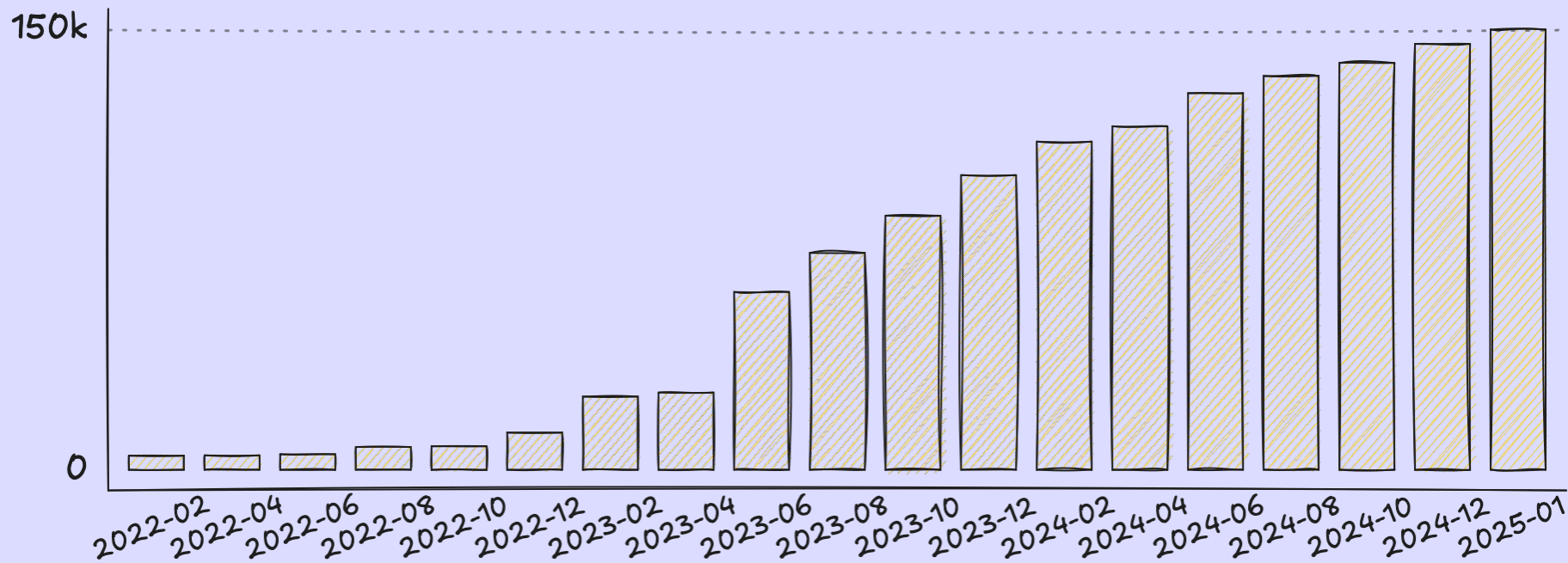
dbt allows to “program” models

- **Build and refresh data**
- **Run tests**
- **Generate catalog**
- **Automate anything based on metadata**

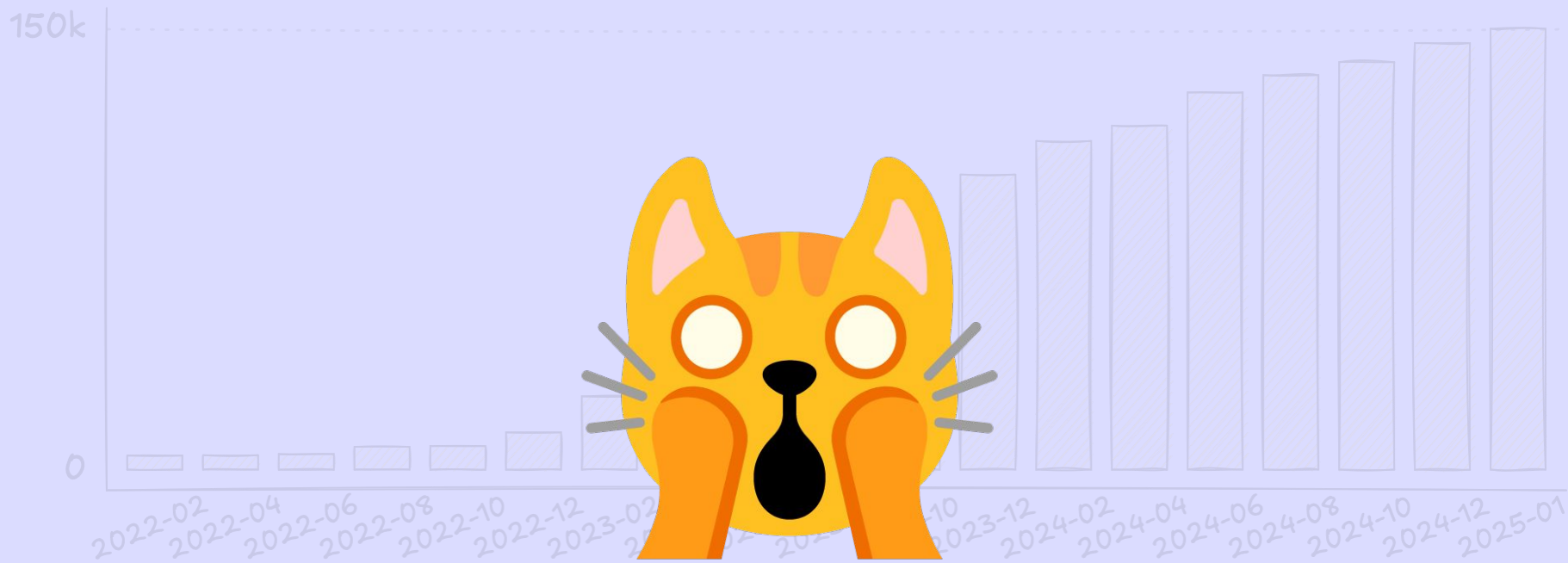
**The
need for
linting**

```
models:
- name: pizzas
  description: All the good pizzas
  config:
    materialized: incremental
    grants:
      select: [pizza_ceo]
  meta:
    owner: Pizzaiolo
    failure_alerts: true
    dashboard: pizzas_by_cheese
  columns:
- name: id
  description: Pizza PK
  data_type: int
  constraints:
    - type: primary_key
  data_tests:
    - unique
    - not_null
- name: cheese
  description: Main type of cheese
  data_type: varchar
  data_tests:
    - not_null
```

```
models:
- name: pizzerias  capitalization
  description: all the good pizzerias
  config:
    materialized: incremental
    grants:  data leak
      select: [pizza_ceo, all]
  meta:
    owner: null  invalid owner
    failure_alerts: true
  columns:
- name: id  no dashboard?
  data_type: int
  data_tests:  forgotten PK
    - not_null
- name: location
  description: Geographical location
  data_type: varchar
- name: rating  missing tests
  description: Average rating
  data_type: decimal
```

YAML source lines of code



YAML source lines of code

dbt offers:

- **First-class citizen data models**
- **Metadata management**

But at scale, we need:


- **Consistency enforcement**
- **Metadata quality checks**
- **Data security**

dbt-score
in the data
stack

dbt-score Public

📌 Edit Pins 33 Unwatch 9 Fork 116 Starred

🔗 master 4 Branches 14 Tags 🔍 Go to file Add file <> Code

 jochemvandooren Prepare release 0.10.0 (#94) ✓ 0b344ec · 41 minutes ago 77 Commits		
📁 .github/workflows	Documenting support for python 3.13 (#86)	last month
📁 .vscode	Add VS Codium/VS Code configuration (#14)	9 months ago
📁 docs	Add generic rule: models should implement uniqueness ...	53 minutes ago
📁 images	Add output image to README (#47)	7 months ago
📁 src/dbt_score	Add generic rule: models should implement uniqueness ...	53 minutes ago
📁 tests	Add generic rule: models should implement uniqueness ...	53 minutes ago
📄 .gitignore	Add VS Codium/VS Code configuration (#14)	9 months ago
📄 .pre-commit-config.yaml	Configure project setup (#2)	11 months ago
📄 .prettierrc	Write documentation for the landing page (#23)	8 months ago
📄 CHANGELOG.md	Prepare release 0.10.0 (#94)	41 minutes ago
📄 LICENSE.txt	Add MIT license (#1)	11 months ago

About

Linters for dbt metadata

dbt-score.picnic.tech
python metadata linter dbt dbt-core

- 📖 README
- 📄 MIT license
- 📈 Activity
- 📄 Custom properties
- ★ 116 stars
- 👁 33 watching
- 🔗 9 forks
- 📄 Report repository

Releases 14

📄 **0.10.0** Latest
39 minutes ago

Languages

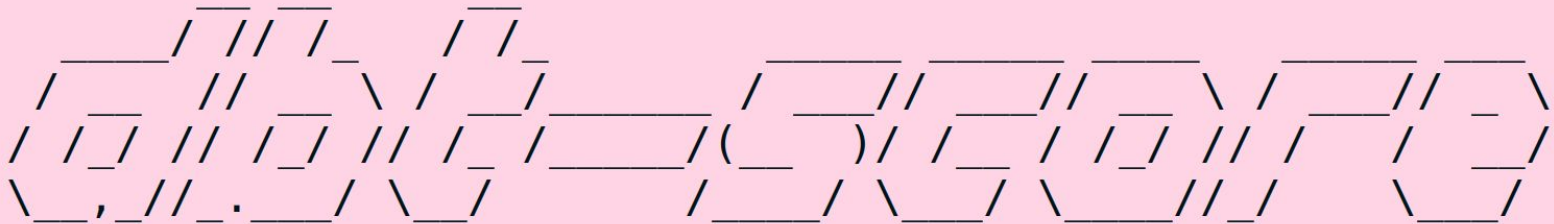


- Python 100.0%

```
> pip install dbt-score
```

```
> dbt-score --help
```

```
Usage: dbt-score [OPTIONS] COMMAND [ARGS]...
```



Options:

- version Show the version and exit.
- h, --help Show this message and exit.

Commands:

- lint Lint dbt metadata.
- list Display rules list.

```
> dbt-score lint
```

```
🔻 M: pizzerias (score: 8.0)
```

```
  WARN (medium) dbt_score.rules.generic.columns_have_description: Columns lack a description: id.
```

```
  WARN (medium) dbt_score.rules.generic.has_owner: Model lacks an owner.
```

```
Project score: 9.0 🔻
```

```
Error: evaluable score too low, fail_any_item_under = 9.5
```

```
Model pizzerias scored 8.0
```



```
> dbt-score lint --show-all
```

```
📉 M: pizzerias (score: 8.0)
```

```
WARN (medium) dbt_score.rules.generic.columns_have_description: Columns lack a description: id.
```

```
OK dbt_score.rules.generic.has_description
```

```
WARN (medium) dbt_score.rules.generic.has_owner: Model lacks an owner.
```

```
OK dbt_score.rules.generic.has_uniqueness_test
```

```
OK dbt_score.rules.generic.single_column_uniqueness_at_column_level
```

```
OK dbt_score.rules.generic.single_pk_defined_at_column_level
```

```
OK dbt_score.rules.generic.sql_has_reasonable_number_of_lines
```

```
📈 M: pizzas (score: 10.0)
```

```
OK dbt_score.rules.generic.columns_have_description
```

```
OK dbt_score.rules.generic.has_description
```

```
OK dbt_score.rules.generic.has_owner
```

```
OK dbt_score.rules.generic.has_uniqueness_test
```

```
OK dbt_score.rules.generic.single_column_uniqueness_at_column_level
```

```
OK dbt_score.rules.generic.single_pk_defined_at_column_level
```

```
OK dbt_score.rules.generic.sql_has_reasonable_number_of_lines
```

```
Project score: 9.0 📉
```

```
Error: evaluable score too low, fail_any_item_under = 9.5
```

```
Model pizzerias scored 8.0
```

```
[tool.dbt-score]
rule_namespaces = ["dbt_score.rules"]
disabled_rules = ["dbt_score.rules.foo"]
fail_project_under = 7.5
fail_any_item_under = 8.0
```

```
[tool.dbt-score]
rule_namespaces = ["dbt_score.rules"]
disabled_rules = ["dbt_score.rules.foo"]
fail_project_under = 7.5
fail_any_item_under = 8.0
```

```
[tool.dbt-score.badges]
first.threshold = 10.0
first.icon = "🥕"
second.threshold = 8.0
second.icon = "😓"
third.threshold = 6.0
third.icon = "🔨"
wip.icon = "🚧"
```

```
[tool.dbt-score]
rule_namespaces = ["dbt_score.rules"]
disabled_rules = ["dbt_score.rules.foo"]
fail_project_under = 7.5
fail_any_item_under = 8.0
```

```
[tool.dbt-score.badges]
first.threshold = 10.0
first.icon = "🥕"
second.threshold = 8.0
second.icon = "😓"
third.threshold = 6.0
third.icon = "🔪"
wip.icon = "🚧"
```

```
[tool.dbt-score.rules."foo.bar"]
severity = 1
max_lines = 300
```

```
[tool.dbt-score]
rule_namespaces = ["dbt_score.rules"]
disabled_rules = ["dbt_score.rules.foo"]
fail_project_under = 7.5
fail_any_item_under = 8.0
```

```
[tool.dbt-score.badges]
first.threshold = 10.0
first.icon = "🥕"
second.threshold = 8.0
second.icon = "😓"
third.threshold = 6.0
third.icon = "🔨"
wip.icon = "🚧"
```

```
[tool.dbt-score.rules."foo.bar"]
severity = 1
max_lines = 300
```

```
[tool.dbt-score.rules."foo.baz"]
rule_filter_names = ["dbt_score.rules.skip"]
```

```
models:
  - name: pizzerias  capitalization
    description: all the good pizzerias
    config:
      materialized: incremental
      grants:  data leak
        select: [pizza_ceo, all]
    meta:
      owner: null  invalid owner
      failure_alerts: true
    columns:
      - name: id  no dashboard?
        data_type: int
        data_tests:  forgotten PK
          - not_null
      - name: location
        description: Geographical location
        data_type: varchar
      - name: rating  missing tests
        description: Average rating
        data_type: decimal
```

```
@rule
```

```
def is_not_capitalized(model: Model) -> RuleViolation | None:  
    """Descriptions must be capitalized"""  
    if model.description and not model.description.istitle():  
        return RuleViolation("Description is not capitalized")
```

```
@rule(severity=Severity.LOW)
def columns_have_tests(model: Model) -> RuleViolation | None:
    """Model has columns with tests."""
    missing = []
    for column in model.columns:
        if not getattr(column, "data_tests", None):
            missing += [column.name]

    if missing:
        return RuleViolation(
            f"Model has columns without tests: {' ', '.join(missing)}"
        )
```



```
> dbt-score lint --show-all
```

```
📌 M: pizzerias (score: 7.0)
```

```
WARN (medium) dbt_score.rules.additional.columns_have_tests: Model has columns without tests: location, rating.
```

```
WARN (medium) dbt_score.rules.additional.is_not_capitalized: Description is not capitalized.
```

```
WARN (medium) dbt_score.rules.generic.columns_have_description: Columns lack a description: id.
```

```
OK dbt_score.rules.generic.has_description
```

```
WARN (medium) dbt_score.rules.generic.has_owner: Model lacks an owner.
```

```
OK dbt_score.rules.generic.has_uniqueness_test
```

```
OK dbt_score.rules.generic.single_column_uniqueness_at_column_level
```

```
OK dbt_score.rules.generic.single_pk_defined_at_column_level
```

```
OK dbt_score.rules.generic.sql_has_reasonable_number_of_lines
```

```
📌 M: pizzas (score: 10.0)
```

```
OK dbt_score.rules.additional.columns_have_tests
```

```
OK dbt_score.rules.additional.is_not_capitalized
```

```
OK dbt_score.rules.generic.columns_have_description
```

```
OK dbt_score.rules.generic.has_description
```

```
OK dbt_score.rules.generic.has_owner
```

```
OK dbt_score.rules.generic.has_uniqueness_test
```

```
OK dbt_score.rules.generic.single_column_uniqueness_at_column_level
```

```
OK dbt_score.rules.generic.single_pk_defined_at_column_level
```

```
OK dbt_score.rules.generic.sql_has_reasonable_number_of_lines
```

```
Project score: 8.5 📌
```

```
Error: evaluable score too low, fail_any_item_under = 9.5
```

```
Model pizzerias scored 7.0
```

```
{
  "project": {
    "score": 7.166666666666666,
    "badge": "😞",
    "pass": false
  },
  "models": {
    "pizzas": {
      "score": 9.0,
      "badge": "😞",
      "pass": true,
      "results": {
        "dbt_score.rules.generic.columns_have_description": {
          "result": "OK",
          "severity": "medium",
          "message": null
        },
        "dbt_score.rules.generic.has_description": {
          "result": "OK",
          "severity": "medium",
          "message": null
        },
        "dbt_score_rules.pydata.columns_have_tests": {
          "result": "WARN",
          "severity": "low",
          "message": "Model has columns without tests: id, cheese"
        },
        ...
      }
    }
  }
}
```

pizzerias table

[Details](#) [Description](#) [Columns](#) [Referenced By](#) [Code](#)

Details

OWNER	FAILURE_ALERTS	SCORE	BADGE		
	true	5.3	😬		
TAGS	PACKAGE	LANGUAGE	ACCESS	VERSION	CONTRACT
untagged	edge	sql	protected		Enforced

Description

all the good pizzerias



dbt-score

[Home](#)[Get started](#)[Create rules](#)[Package rules](#)[Configuration](#)[Programmatic invocations](#)[Rules](#)[Reference](#)[Contributor's guide](#)[Changelog](#)

Welcome to dbt-score

[Table of contents](#)[Example](#)[Philosophy](#)[About](#)

`dbt-score` is a linter for `dbt` metadata.

dbt allows data practitioners to organize their data in to *models* and *sources*. Those models and sources have metadata associated with them: documentation, tests, types, etc.

- > `dbt-score` allows to lint and score this metadata, in order to enforce (or encourage) good practices.

Example

```
> dbt-score lint --show all
🏆 M: customers (score: 10.0)
  OK dbt_score.rules.generic.has_description
  OK dbt_score.rules.generic.has_owner
  OK dbt_score.rules.generic.sql_has_reasonable_number_of_lines
Score: 10.0 🏆
```



In this example, the model `customers` scores the maximum value of `10.0` as it passes all the rules. It also is awarded a golden medal because of the perfect score. By default a passing model or source with or without rule violations will not be shown. unless we pass the `--show-`

Links

- <https://dbt-score.picnic.tech>
- <https://github.com/PicnicSupermarket/dbt-score>



**Matthieu
Caneill**

Credits

- **dbt-score authors**



- **Picnic** 



**Jochem
van
Dooren**