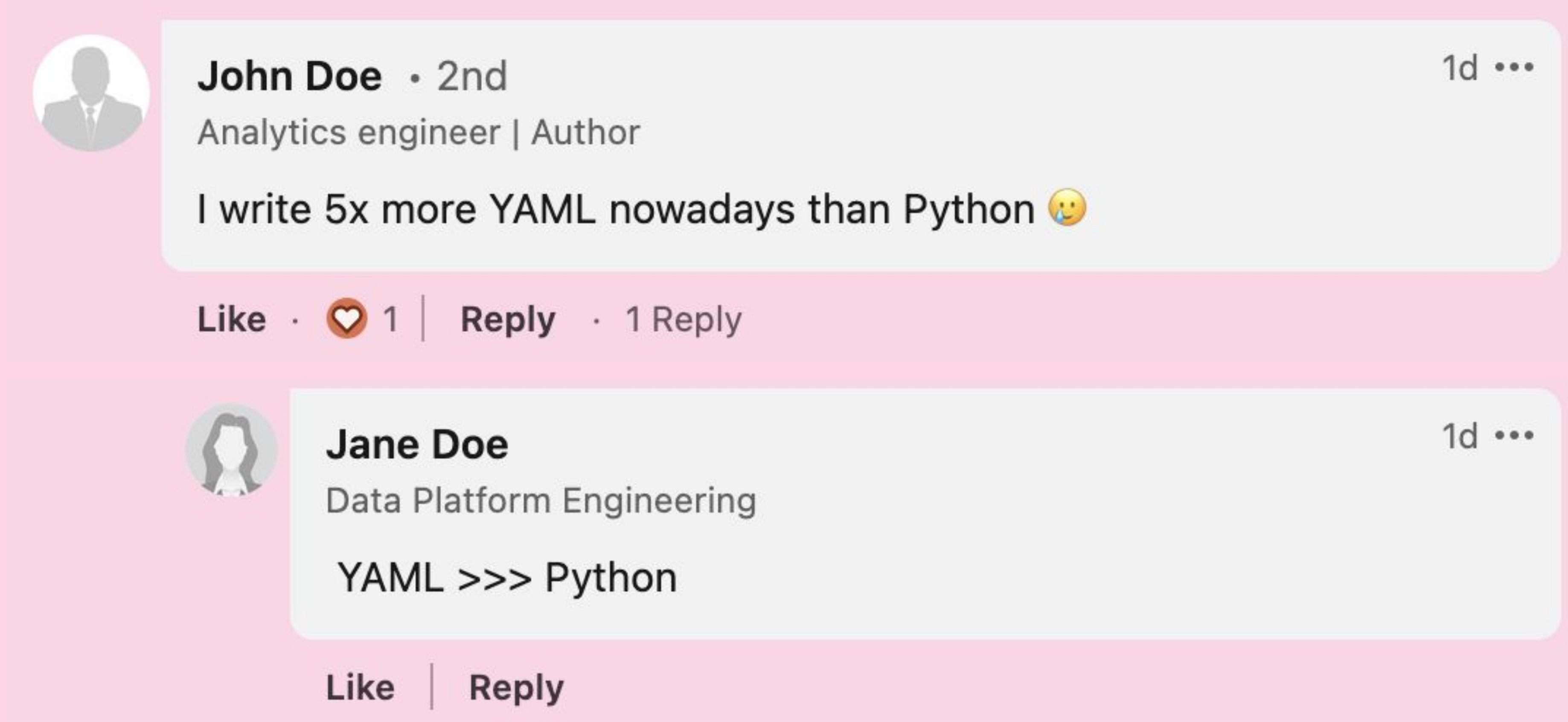

John Doe • 2nd

1d •••

Analytics engineer | Author

I write 5x more YAML nowadays than Python 🥲

Like · ❤ 1 | Reply · 1 Reply

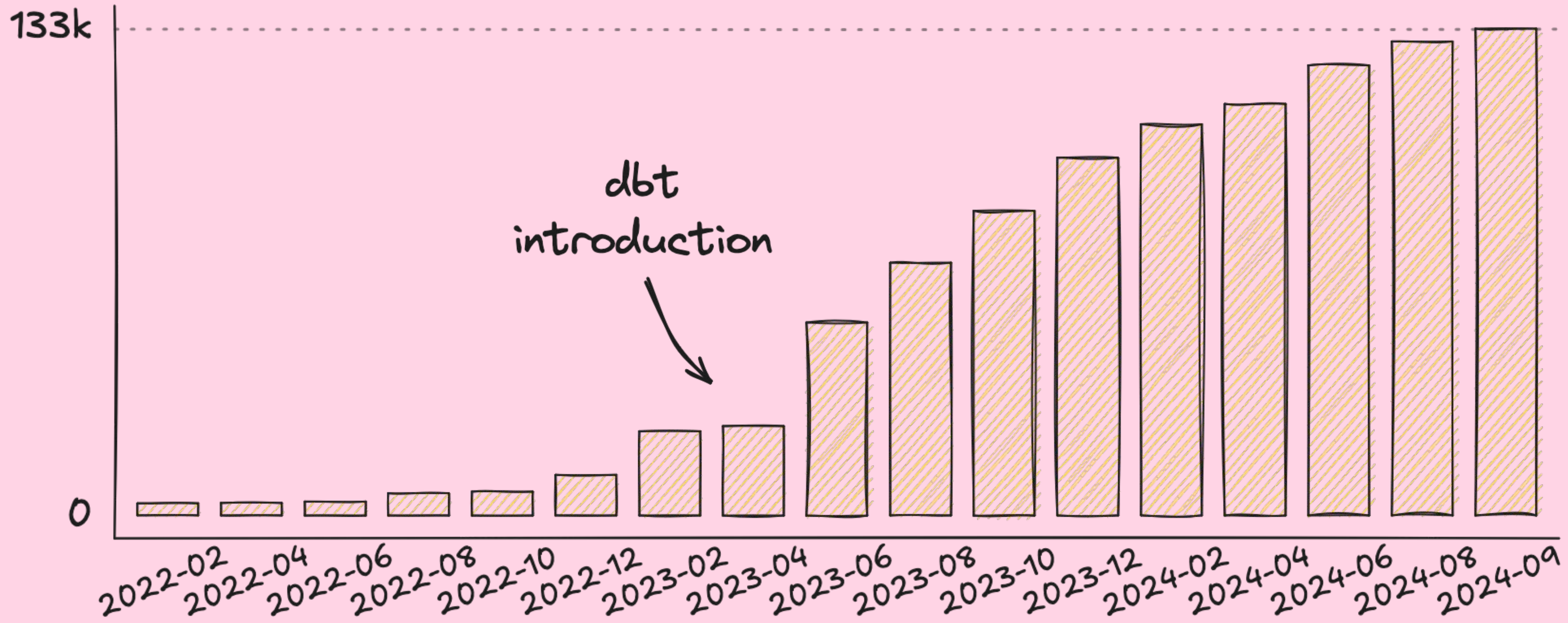
Jane Doe

1d •••

Data Platform Engineering

YAML >>> Python

Like | Reply



YAML source lines of code

Matthieu Caneill

Tech lead - Platform team, Analytics

🇫🇷 🥐

Pixel
Coding assistant

Picnic

Online supermarket
Groceries supply-chain
Lots of data

🥇 dbt-score

A linter for dbt model metadata

Takeaways

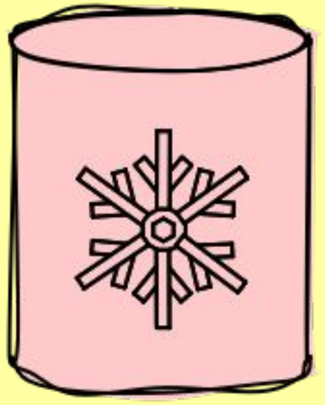
1. Use **declarative** model properties
2. Enforce **consistency** through linting
3. Allow **customizations** through simple UX

Outline

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

Py Data

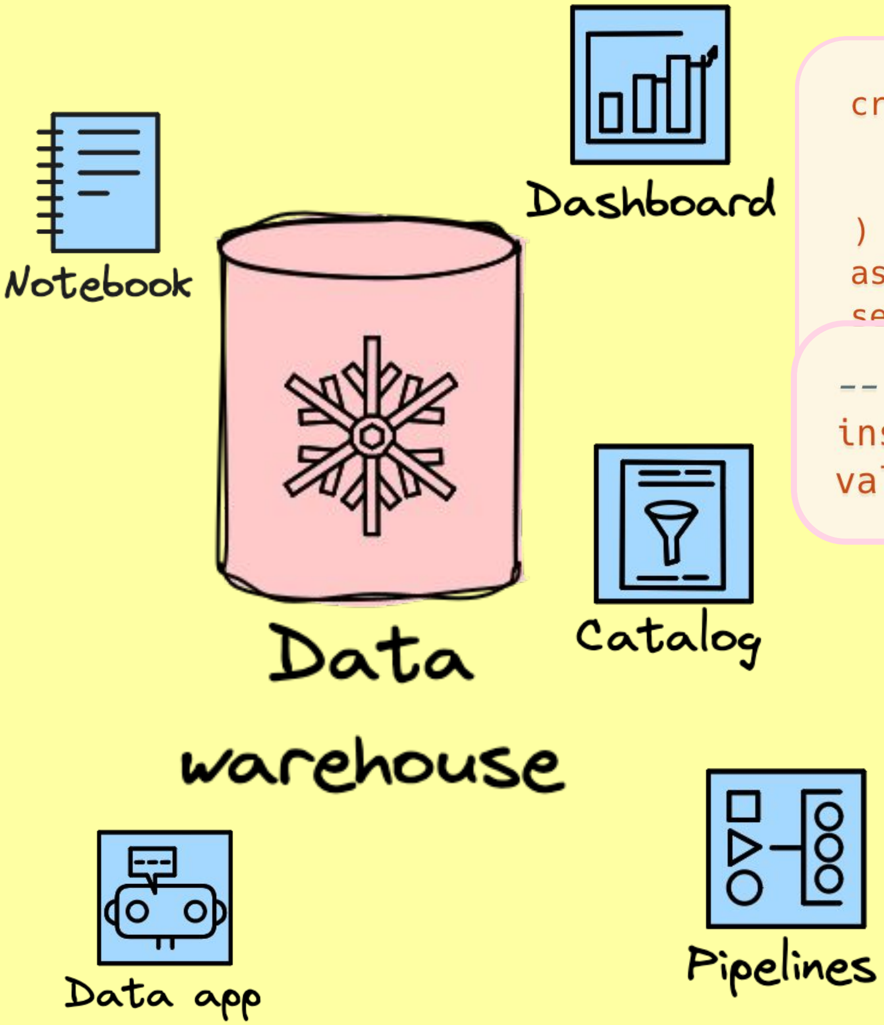
What is dbt anyway?



```
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
```



```
crea
  id
  ch
)
as
select
```

```
alter table pizzas
add column spiciness int
```

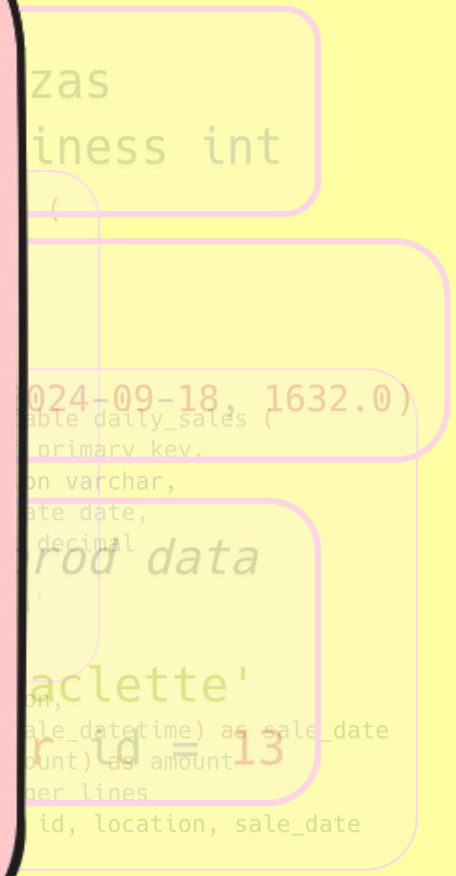
```
-- Fix 2024-09-19
insert into daily_sales
values (42, 'Amsterdam', 2024-09-18, 1632.0)
```

```
  id,
  location
```

```
location varchar,
```

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

```
_date
group by id, location, sale_date
```



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

dbt

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

Metadata

Logic

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

pizzas.sql

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

pizzas.yml

```yaml
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

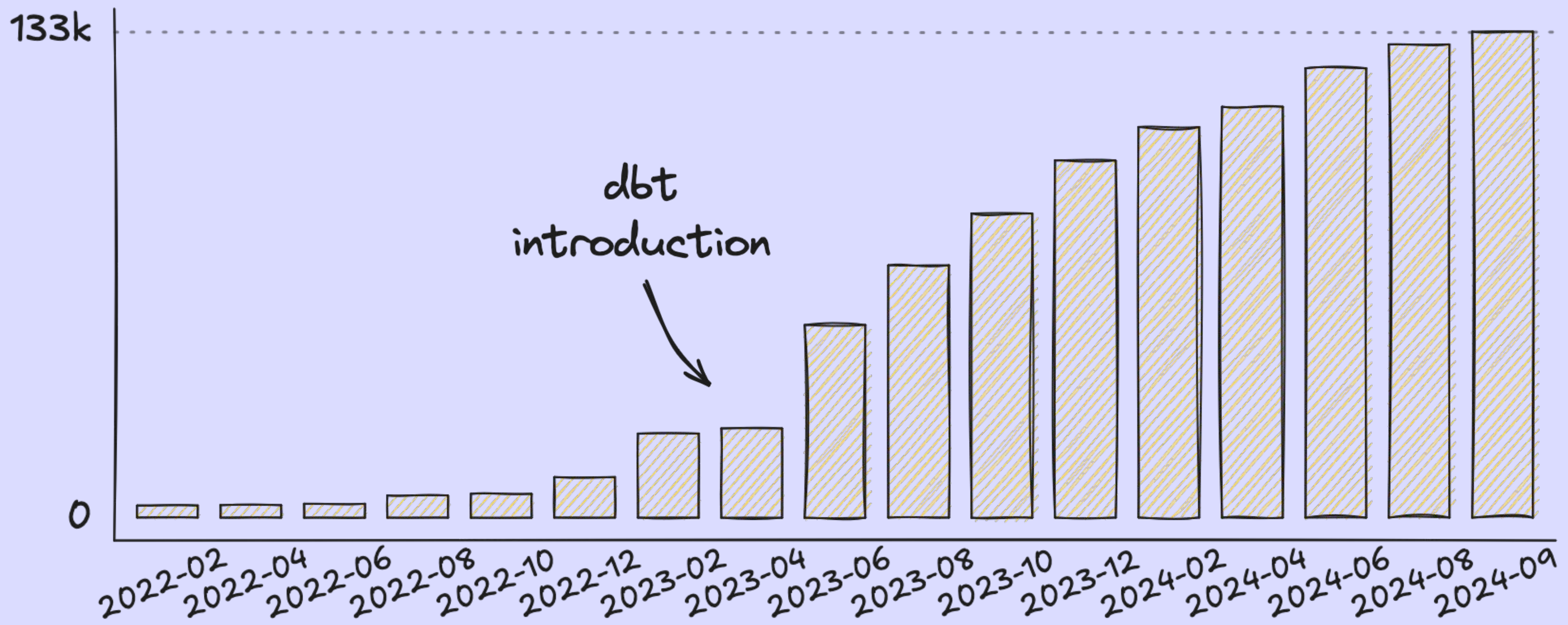
dbt allows to “program” models

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

The need for linting

```yaml
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
```

```yaml
models:                                 ⚠️ capitalization
  - name: pizzerias
    description: all the good pizzerias
    config:
      materialized: incremental
      grants:                              ⚠️ data leak
        select: [pizza_ceo, all]
    meta:
      owner: null                       ⚠️ invalid owner
      failure_alerts: true
    columns:                            ⚠️ no dashboard?
      - name: id
        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

Many models
… with **many** properties
… written by **many** people

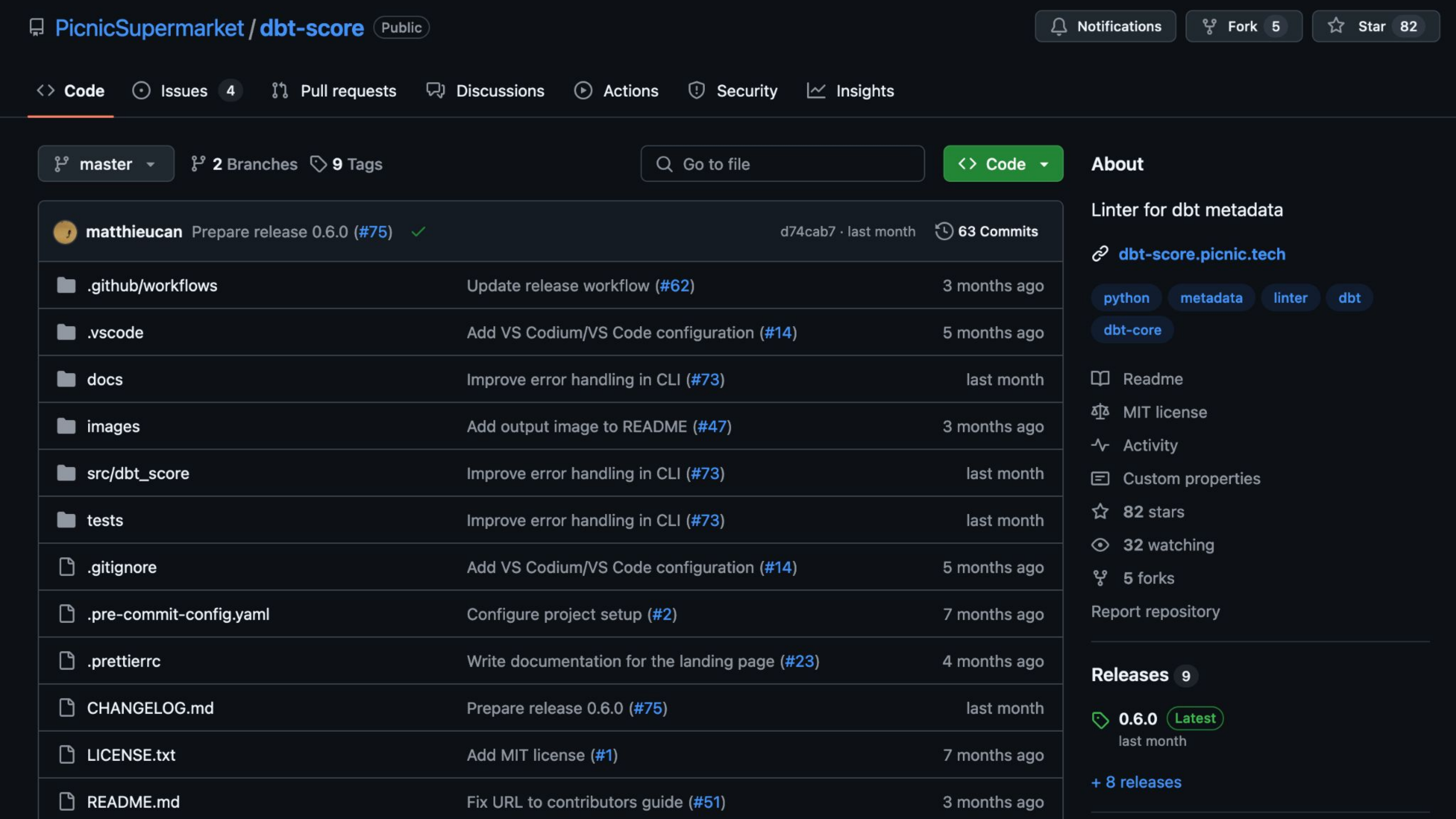
dbt offers:

- First-class citizen models
- Metadata management

But at scale, we need:

- Consistency enforcement
- Metadata quality checks
- Data security

dbt-score
in the data stack



matthieucan Prepare release 0.6.0 (#75) ✓ d74cab7 · last month 63 Commits

.github/workflows	Update release workflow (#62)	3 months ago
.vscode	Add VS Codium/VS Code configuration (#14)	5 months ago
docs	Improve error handling in CLI (#73)	last month
images	Add output image to README (#47)	3 months ago
src/dbt_score	Improve error handling in CLI (#73)	last month
tests	Improve error handling in CLI (#73)	last month
.gitignore	Add VS Codium/VS Code configuration (#14)	5 months ago
.pre-commit-config.yaml	Configure project setup (#2)	7 months ago
.prettierrc	Write documentation for the landing page (#23)	4 months ago
CHANGELOG.md	Prepare release 0.6.0 (#75)	last month
LICENSE.txt	Add MIT license (#1)	7 months ago
README.md	Fix URL to contributors guide (#51)	3 months ago

About

Linter for dbt metadata

dbt-score.picnic.tech

python metadata linter dbt dbt-core

Readme
MIT license
Activity
Custom properties
82 stars
32 watching
5 forks
Report repository

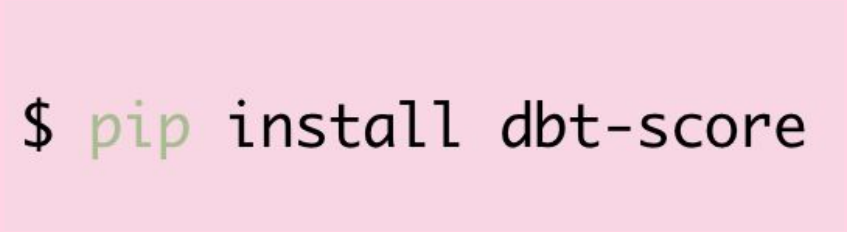
Releases 9

0.6.0 Latest
last month

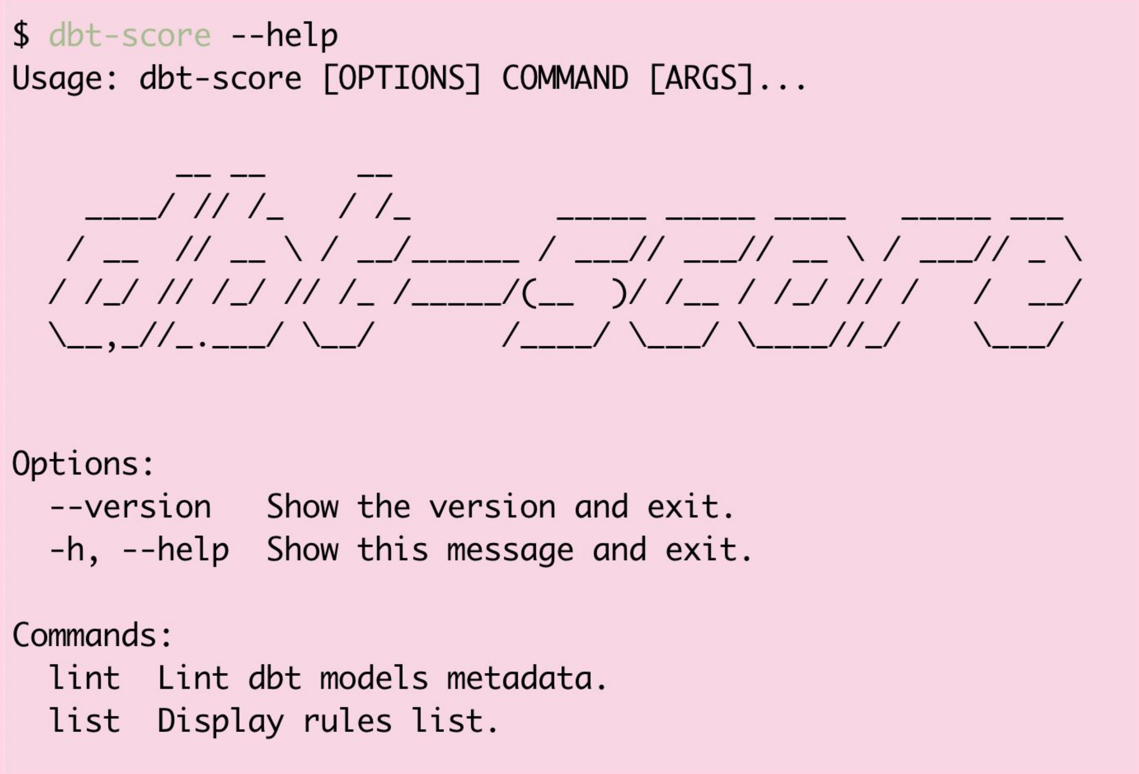
+ 8 releases

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 models metadata.
  list  Display rules list.
```

```
~/git/pizzanalytics $ dbt-score lint

🥇 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.public_model_has_example_sql
    OK   dbt_score.rules.generic.sql_has_reasonable_number_of_lines

🥉 pizzerias (score: 7.3)
    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.public_model_has_example_sql
    OK   dbt_score.rules.generic.sql_has_reasonable_number_of_lines

Project score: 8.6 🥈
```

```toml
[tool.dbt-score]
rule_namespaces = ["dbt_score.rules", "my_rules"]
disabled_rules = ["my_rules.check_foo"]
fail_project_under = 7.5
fail_any_model_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."my_rules.check_bar"]
severity = 1
some_param = "foobar"
```

```yaml
models:
  - name: pizzerias
    description: all the good pizzerias
    config:
      materialized: incremental
      grants:
        select: [pizza_ceo, all]
    meta:
      owner: null
      failure_alerts: true
    columns:
      - name: id
        data_type: int
        data_tests:
          - not_null
      - name: location
        description: Geographical location
        data_type: varchar
      - name: rating
        description: Average rating
        data_type: decimal
```

⚠️ **capitalization**

⚠️ **data leak**

⚠️ **invalid owner**

⚠️ **no dashboard?**

⚠️ **forgotten PK**

⚠️ **missing tests**

```python
@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")
```

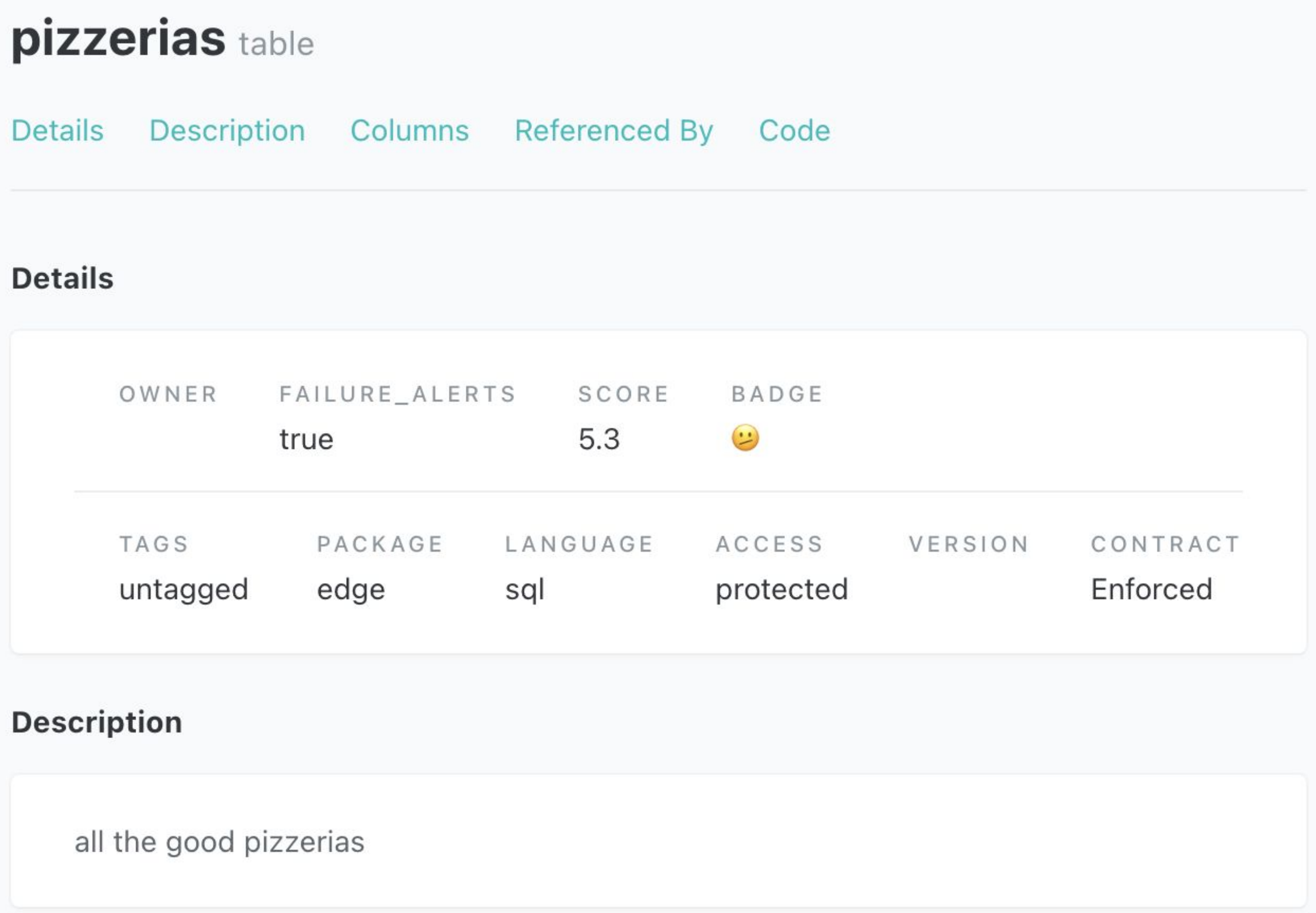
```python
@rule(severity=Severity.LOW)
def columns_have_tests(model: Model) -> RuleViolation | None:
    """Model has columns with tests."""
    missing = []
    for column in model.columns:
        if not getattr(column, "data_tests", None):
            missing += [column.name]

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

```
$ dbt-score lint -s pizzerias
😕pizzerias (score: 5.0)
    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.public_model_has_example_sql
    OK   dbt_score.rules.generic.sql_has_reasonable_number_of_lines
    WARN (medium) dbt_score_rules.pydata.columns_have_tests: Model has columns without tests: id, l
    WARN (medium) dbt_score_rules.pydata.does_not_leak: Model leaks data
    WARN (medium) dbt_score_rules.pydata.has_dashboard: Model does not have an associated dashboard
    WARN (high) dbt_score_rules.pydata.has_primary_key: Model must define a primary key
    WARN (medium) dbt_score_rules.pydata.is_not_capitalized: Description is not capitalized

Project score: 5.0 😕
```

```json
{
  "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

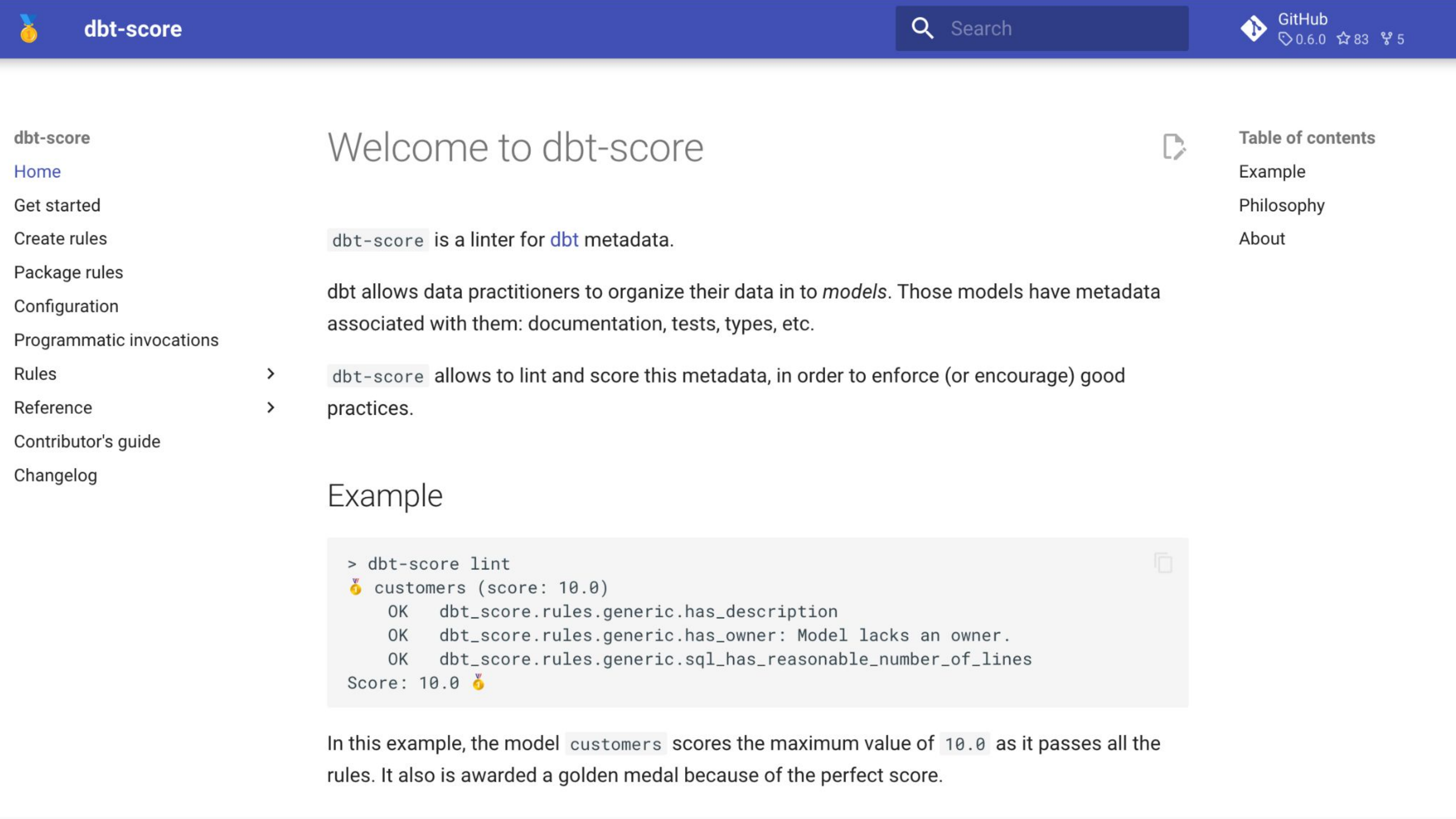
Details

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

Search

GitHub
0.6.0 83 5

Welcome to dbt-score

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

dbt allows data practitioners to organize their data in to *models*. Those models 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
🥇 customers (score: 10.0)
    OK   dbt_score.rules.generic.has_description
    OK   dbt_score.rules.generic.has_owner: Model lacks an owner.
    OK   dbt_score.rules.generic.sql_has_reasonable_number_of_lines
Score: 10.0 🥇
```

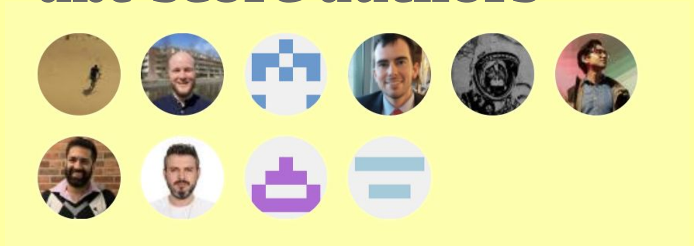
In this example, the model `customers` scores the maximum value of `10.0` as it passes all the rules. It also is awarded a golden medal because of the perfect score.

Links

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

Credits

- dbt-score authors



- Picnic - we're hiring!
  https://jobs.picnic.app

Matthieu Caneill

https://matthieu.io