O que iremos aprender
Na aula de hoje criaremos um projeto dbt com a base de autos_cleaned, faremos a inserção da base no postgresql, e criando camadas especializadas, assim como utilizaremos o plugin do Great Expectations para garantir a qualidade do pipeline

Durante a aula estaremos revendo todos os conceitos abordados nesse curso de forma prática em um projeto final
Então vamos lá!

Hands on
Bom, vamos executar o comando
dbt init
Dando o nome de autos_project, selecionando a opção 2 postgres

```
14:19:33  Running with dbt=1.7.11
Enter a name for your project (letters, digits, underscore): autos_project
14:19:41
Your new dbt project "autos_project" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

  https://docs.getdbt.com/docs/configure-your-profile

One more thing:

Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:

  https://community.getdbt.com/

Happy modeling!

14:19:41  Setting up your profile.
Which database would you like to use?
[1] duckdb
[2] postgres

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number: 2
```

Passando as informações abaixo para configurar a conexão no arquivo profiles.yml

```
Enter a number: 2
host (hostname for the instance): localhost
port [5432]:
user (dev username): postgres
pass (dev password):
dbname (default database that dbt will build objects in): ada
schema (default schema that dbt will build objects in): public
threads (1 or more) [1]: 1
14:20:46  Profile autos_project written to C:\Users\costa\.dbt\profiles.yml using target's profile_template
.yml and your supplied values. Run 'dbt debug' to validate the connection.
```

Vamos executar o comando dbt debug dentro da pasta autos_project para testarmos a conexão

cd autos_project
dbt debug

```
14:22:14  python path: C:\Users\costa\Python3.12\python.exe
14:22:14  os info: Windows-11-10.0.22631-SP0
14:22:14  Using profiles dir at C:\Users\costa\.dbt
14:22:14  Using profiles.yml file at C:\Users\costa\.dbt\profiles.yml
14:22:14  Using dbt_project.yml file at C:\Users\costa\workspace\ada-tech\engenharia-de-dados-cursos\anal
ics-engineering\aula_2\autos_project\dbt_project.yml
14:22:14  adapter type: postgres
14:22:14  adapter version: 1.7.11
14:22:14  Configuration:
14:22:14    profiles.yml file [OK found and valid]
14:22:14    dbt_project.yml file [OK found and valid]
14:22:14  Required dependencies:
14:22:14   - git [OK found]

14:22:14  Connection:
14:22:14    host: localhost
14:22:14    port: 5432
14:22:14    user: postgres
14:22:14    database: ada
14:22:14    schema: public
14:22:14    connect_timeout: 10
14:22:14    role: None
14:22:14    search_path: None
14:22:14    keepalives_idle: 0
14:22:14    sslmode: None
14:22:14    sslcert: None
14:22:14    sslkey: None
14:22:14    sslrootcert: None
14:22:14    application_name: dbt
14:22:14    retries: 1
14:22:14  Registered adapter: postgres=1.7.11
14:22:14    Connection test: [OK connection ok]
```

Como podemos ver, tudo funcionou corretamente,

Instalando o plugin do great expectations

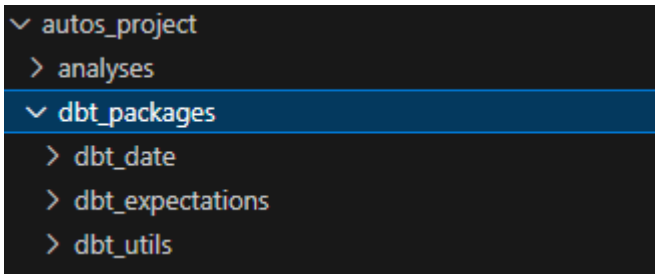
Para instalar o plugin do great expectations, vamos criar um arquivo yml chamado packages, passando o package calogica expectations

```yaml
packages:
  - package: calogica/dbt_expectations
    version: [">=0.6.0", "<0.7.0"]
```

E executaremos o comando dbt deps para que o dbt instale esse pacote
dbt deps

```
 dbt deps
14:26:40  Running with dbt=1.7.11
14:26:42  Updating lock file in file path: C:\Users\costa\workspace\ada-tech\engenharia-de-dados-cursos\ana
lytics-engineering\aula_2\autos_project/package-lock.yml
14:26:42  Installing calogica/dbt_expectations
14:26:42  Installed from version 0.6.1
14:26:42  Updated version available: 0.10.3
14:26:42  Installing calogica/dbt_date
14:26:43  Installed from version 0.6.3
14:26:43  Updated version available: 0.10.1
14:26:43  Installing dbt-labs/dbt_utils
14:26:43  Installed from version 0.9.6
14:26:43  Updated version available: 1.1.1
14:26:43
14:26:43  Updates available for packages: ['calogica/dbt_expectations', 'calogica/dbt_date', 'dbt-labs/dbt_
utils']
```

Podemos verificar que a pasta dbt_packages foi criada para armazenas os códigos dbt das bibliotecas



Configurando o yml dbt project

Para configurarmos cada uma das pastas de modelos, o que irá habilitar a criação dos modelos de acordo com a opção que passarmos, basta alterar no yml os models da seguinte maneira

```yaml
models:
  autos_project:
    # Config indicated by + and applies to all files under models/example/
    bronze:
      +materialized: table
    silver:
      +materialized: table
    gold:
      +materialized: view
```

A ideia é que a camada bronze e prata de dados estejam como tabelas no postgres e a gold seja uma view materializada. Vamos excluir a pasta de example dentro de models, ela já não servirá para o nosso projeto.
Então executaremos o
dbt build e veremos que nenhuma mensagem de erro aparece, agora precisamos criar os modelos e os sql que serão executados

Criando os modelos de dados

Bom, vamos explorar como criar modelos de dados para o nosso projeto, vamos criar as pastas bronze, silver e gold, na pasta models

Criando a camada bronze

Para criarmos a camada bronze, que fará conexão inicial com o postgres, precisamos criar um arquivo yaml chamado sources, que terá as informações da tabela de origem de autos:

```yaml
version: 2

sources:
  - name: autos
    description: 'Fonte de dados Carros usados'
    database: ada
    schema: public
```

```
    tables:
      - name: autos_cleaned
        description: 'Dados brutos de Carros usados'
```

Bom, então vamos configurar o yaml de models para informar o modelo da tabela que será gerada a partir da source informada

```
version: 2

models:
  - name: autos_bronze
    description: "Tabela limpa de informações de automóveis."
    columns:

      - name: name
        description: "Nome do automóvel."
        tests:
          - not_null

      - name: price
        description: "Preço do automóvel."
        tests:
          - not_null

      - name: abtest
        description: "Grupo de teste A/B."
        tests:
          - not_null

      - name: gearbox
        description: "Tipo de caixa de câmbio."
        tests:
          - not_null

      - name: model
        description: "Modelo do veículo."
        tests:
          - not_null

      - name: kilometer
        description: "Quilometragem do veículo."
        tests:
          - not_null
```

```
      - name: brand
        description: "Marca do veículo."
        tests:
          - not_null
```

- not_null: Verifica se a coluna não contém valores nulos.
- greater_than: Verifica se os valores numéricos da coluna são maiores que um valor específico (por exemplo, price deve ser maior que 0).
- less_than: Verifica se os valores numéricos da coluna são menores que um valor específico (usado para yearOfRegistration para garantir que o ano esteja dentro de um intervalo realista).
- Podemos ajustar os valores nos testes greater_than e less_than conforme apropriado para seu contexto específico.
-

Lembrando que a funcionalidade de teste do dbt não é uma substituição direta para Great Expectations, mas oferece uma abordagem similar de garantir qualidade e integridade dos dados dentro do fluxo de trabalho do dbt.

Criando o SQL para a camada Bronze

Na mesma pasta que inserimos a modelagem, vamos adicionar um arquivo sql com o nome de autos_bronze para colocarmos qual será a query a ser executada com o código:

```
with source_autos as (
    select * from {{ source('autos', 'autos_cleaned') }}
),

final as (
    select * from source_autos
)

select * from final
```

Aqui a ideia é criarmos um cte, fazendo um select na origem do postgres, em cima de um cte chamado final, podemos até filtrar se quisermos, e o select * from final criará de fato a tabela nova autos_bronze no postgres

Se executarmos dbt build, o código será construido com sucesso, então podemos inclusive realizar o dbt test e dbt run para verificamos que a tabela é criada corretamente

Criando a modelagem da tabela Silver
Vamos criar um modelo Prata que agrega dados por marca e modelo do veículo, incluindo cálculos de preços médios, potência média, e quilometragem média, que é útil para análises comparativas entre diferentes marcas e modelos.

Vamos criar o arquivo yml models com o seguinte formato

```yaml
version: 2

models:
  - name: autos_silver
    columns:
      - name: avg_price
        tests:
          - not_null
          - dbt_expectations.expect_column_values_to_be_of_type:
              column_type: numeric
          - dbt_expectations.expect_column_values_to_be_between:
              min_value: 0
              max_value: 500000
      - name: avg_power_ps
        tests:
          - not_null
      - name: avg_kilometer
        tests:
          - not_null
```

A ideia aqui é realizarmos a validação de 3 colunas sumarizadas principais
E então podemos declarar o sql final que realizará a criação de uma base com a marca, o modelo, ano de registros, preço para aquele modelo etc.
Algo que conseguimos fazer, é utilizar o great expectations para extrair métricas de `expect_column_values_to_be_of_type e expect_column_values_to_be_between` para coluna avg_prince, o que facilita muito os nossos testes

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

SELECT
    brand,
    model,
    "yearOfRegistration" as year_of_registration,
    ROUND(AVG(price)::numeric, 2) AS avg_price,
    ROUND(AVG("powerPS")::numeric, 2) AS avg_power_ps,
    ROUND(AVG(kilometer)::numeric, 2) AS avg_kilometer,
```

```
    COUNT(*) AS total_listings
FROM {{ ref('autos_bronze') }}
GROUP BY brand, model, "yearOfRegistration"
```

Nesse caso, nós não precisaremos trazer o source yml, porque já estabelecemos a conexão com o source da camada bronze, o arquivo profiles yml, e o dbt package

Criando a Camada Gold

Para a camada Gold, vamos criar um modelo que foca em insights de alto nível, como o desempenho do mercado de automóveis ao longo do tempo. Este modelo agregará dados por ano de registro, destacando a evolução dos preços médios e a distribuição de tipos de veículos.

```
version: 2

models:
  - name: autos_gold
    columns:
      - name: avg_price
        tests:
          - not_null
```

Executaremos esse modelo com o sql

```
{{ config(materialized='view') }}

SELECT
    year_of_registration,
    ROUND(AVG(avg_price)::numeric, 2) AS avg_price,  -- Arredonda o preço
médio para duas casas decimais
    SUM(total_listings) AS total_vehicles
FROM {{ ref('autos_silver') }}

GROUP BY year_of_registration
ORDER BY year_of_registration desc
```

Executando o projeto dbt

Executaremos o comando dbt build, ou o dbt test e dbt run, para testar e executarmos o nosso projeto, se tudo tiver dado certo, teremos todos os testes e execuções como PASS

```
:16:03
:16:03  1 of 16 START sql table model public.autos_bronze ..............................[RUN]
:16:04  1 of 16 OK created sql table model public.autos_bronze .........................[SELECT 303950 i
0.70s]
:16:04  2 of 16 START test not_null_autos_bronze_abtest .................................[RUN]
:16:04  2 of 16 PASS not_null_autos_bronze_abtest .......................................[PASS in 0.12s]
:16:04  3 of 16 START test not_null_autos_bronze_brand ..................................[RUN]
:16:04  3 of 16 PASS not_null_autos_bronze_brand ........................................[PASS in 0.09s]
:16:04  4 of 16 START test not_null_autos_bronze_gearbox ................................[RUN]
:16:05  4 of 16 PASS not_null_autos_bronze_gearbox ......................................[PASS in 0.09s]
:16:05  5 of 16 START test not_null_autos_bronze_kilometer ..............................[RUN]
:16:05  5 of 16 PASS not_null_autos_bronze_kilometer ....................................[PASS in 0.14s]
:16:05  6 of 16 START test not_null_autos_bronze_model ..................................[RUN]
:16:05  6 of 16 PASS not_null_autos_bronze_model ........................................[PASS in 0.09s]
:16:05  7 of 16 START test not_null_autos_bronze_name ...................................[RUN]
:16:05  7 of 16 PASS not_null_autos_bronze_name .........................................[PASS in 0.08s]
:16:05  8 of 16 START test not_null_autos_bronze_price ..................................[RUN]
:16:05  8 of 16 PASS not_null_autos_bronze_price ........................................[PASS in 0.08s]
:16:05  9 of 16 START sql table model public.autos_silver ...............................[RUN]
:16:05  9 of 16 OK created sql table model public.autos_silver ..........................[SELECT 8661 in
22s]
:16:05  10 of 16 START test dbt_expectations_expect_column_values_to_be_between_autos_silver_avg_price__5
000__0  [RUN]
:16:05  10 of 16 PASS dbt_expectations_expect_column_values_to_be_between_autos_silver_avg_price__500000_
  [PASS in 0.04s]
:16:05  11 of 16 START test dbt_expectations_expect_column_values_to_be_of_type_autos_silver_avg_price__n
eric  [RUN]
:16:05  11 of 16 PASS dbt_expectations_expect_column_values_to_be_of_type_autos_silver_avg_price__numeric
[PASS in 0.07s]
:16:05  12 of 16 START test not_null_autos_silver_avg_kilometer .........................[RUN]
:16:05  12 of 16 PASS not_null_autos_silver_avg_kilometer ...............................[PASS in 0.04s]
:16:05  13 of 16 START test not_null_autos_silver_avg_power_ps ..........................[RUN]
:16:05  13 of 16 PASS not_null_autos_silver_avg_power_ps .................................[PASS in 0.04s]
:16:05  14 of 16 START test not_null_autos_silver_avg_price .............................[RUN]
:16:05  14 of 16 PASS not_null_autos_silver_avg_price ...................................[PASS in 0.04s]
:16:05  15 of 16 START sql view model public.autos_gold .................................[RUN]
:16:05  15 of 16 OK created sql view model public.autos_gold ...........................[CREATE VIEW in
07s]
:16:05  16 of 16 START test not_null_autos_gold_avg_price ................................[RUN]
:16:05  16 of 16 PASS not_null_autos_gold_avg_price ......................................[PASS in 0.04s]
:16:06
:16:06  Finished running 2 table models, 13 tests, 1 view model in 0 hours 0 minutes and 2.14 seconds (2.
```

Gerando a documentação final

No dbt (Data Build Tool), a geração de documentação é uma funcionalidade integrada que permite criar uma documentação interativa do seu projeto dbt, incluindo descrições de modelos, colunas, testes, e o lineage de dados (a linhagem que mostra como os dados fluem de um modelo para outro). Aqui está como você pode gerar e visualizar a documentação do seu projeto dbt:

Adicionar Descrições no seu projeto dbt

Antes de gerar a documentação, é uma boa prática adicionar descrições aos seus modelos e colunas nos arquivos .yml associados aos seus modelos. Isso torna a documentação mais útil e informativa.

Por exemplo, para a camada prata podemos adicionar a seguintes descrições

version: 2

models:
  - name: autos_silver
    description: "Tabela agregada dos dados de veículos, incluindo médias de preço e quilometragem."

```
    columns:
      - name: brand
        description: "Marca do veículo."
      - name: model
        description: "Modelo do veículo."
      - name: year_of_registration
        description: "Ano em que o veículo foi registrado."
      - name: avg_price
        description: "Preço médio do veículo."
      - name: avg_power_ps
        description: "Potência média do motor em PS."
      - name: avg_kilometer
        description: "Quilometragem média percorrida pelo veículo."
      - name: total_listings
        description: "Número total de listagens agregadas."
```

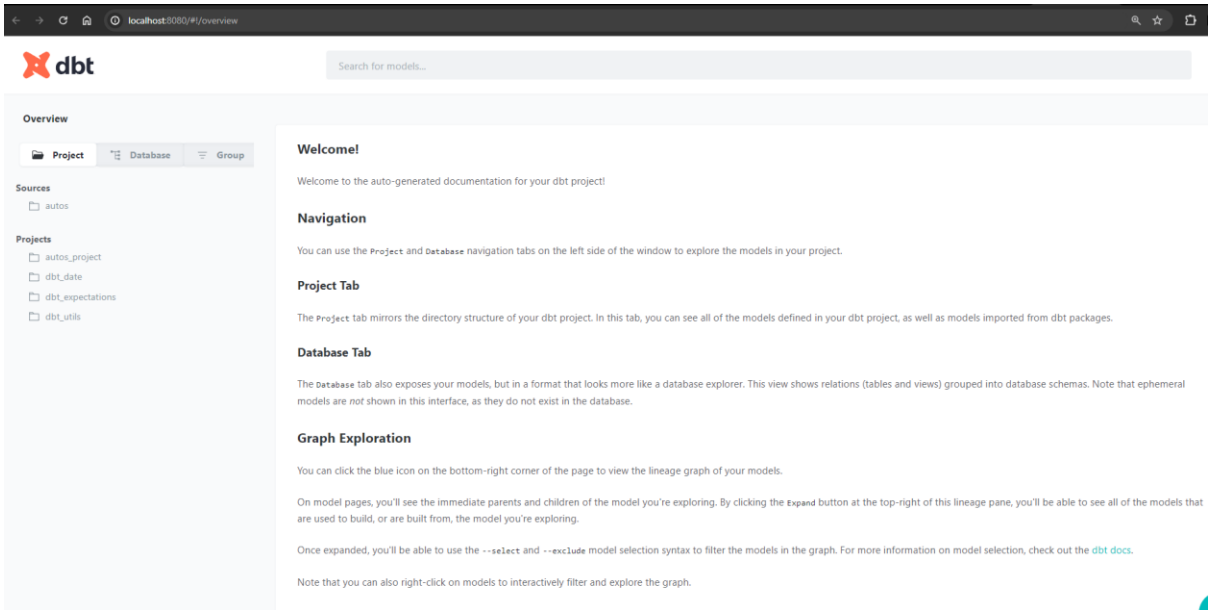
Após adicionar descrições relevantes, você pode gerar a documentação usando o seguinte comando no terminal:

dbt docs generate

```
dbt docs generate
16:29:22  Running with dbt=1.7.11
16:29:22  Registered adapter: postgres=1.7.11
16:29:22  Unable to do partial parsing because saved manifest not found. Starting full parse.
16:29:24  Found 3 models, 13 tests, 1 source, 0 exposures, 0 metrics, 782 macros, 0 groups, 0 semantic mode
ls
16:29:24
16:29:24  Concurrency: 1 threads (target='dev')
16:29:24
16:29:24  Building catalog
16:29:24  Catalog written to C:\Users\costa\workspace\ada-tech\engenharia-de-dados-cursos\analytics-enginee
ring\aula_2\autos_project\target\catalog.json
```

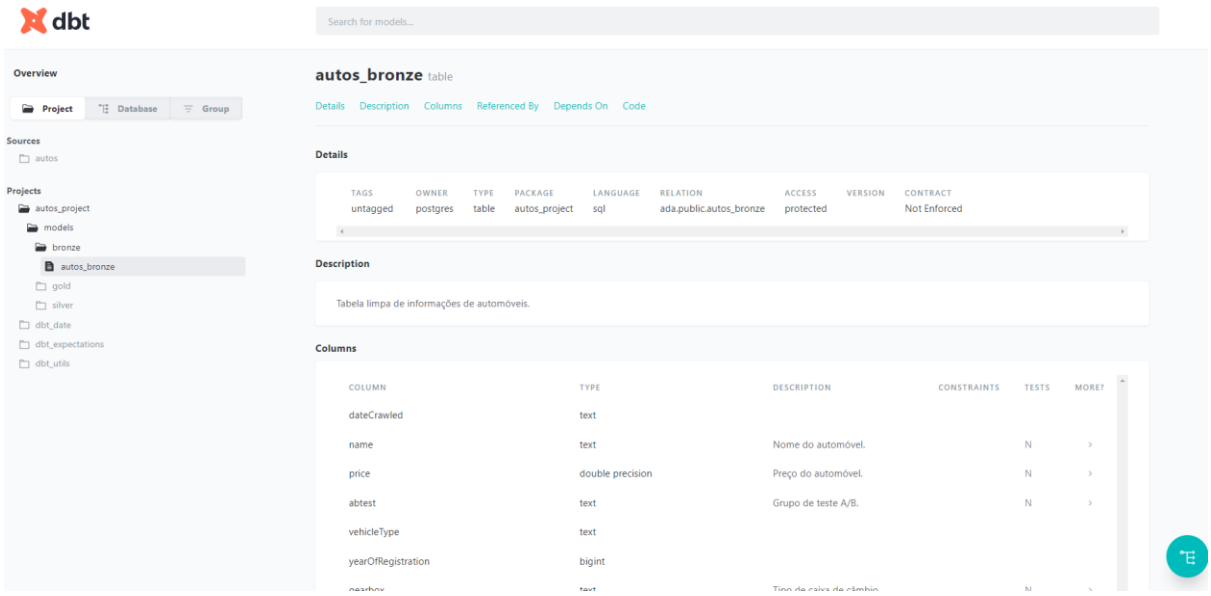
Uma vez que a documentação é gerada, você pode visualizá-la localmente usando o comando:

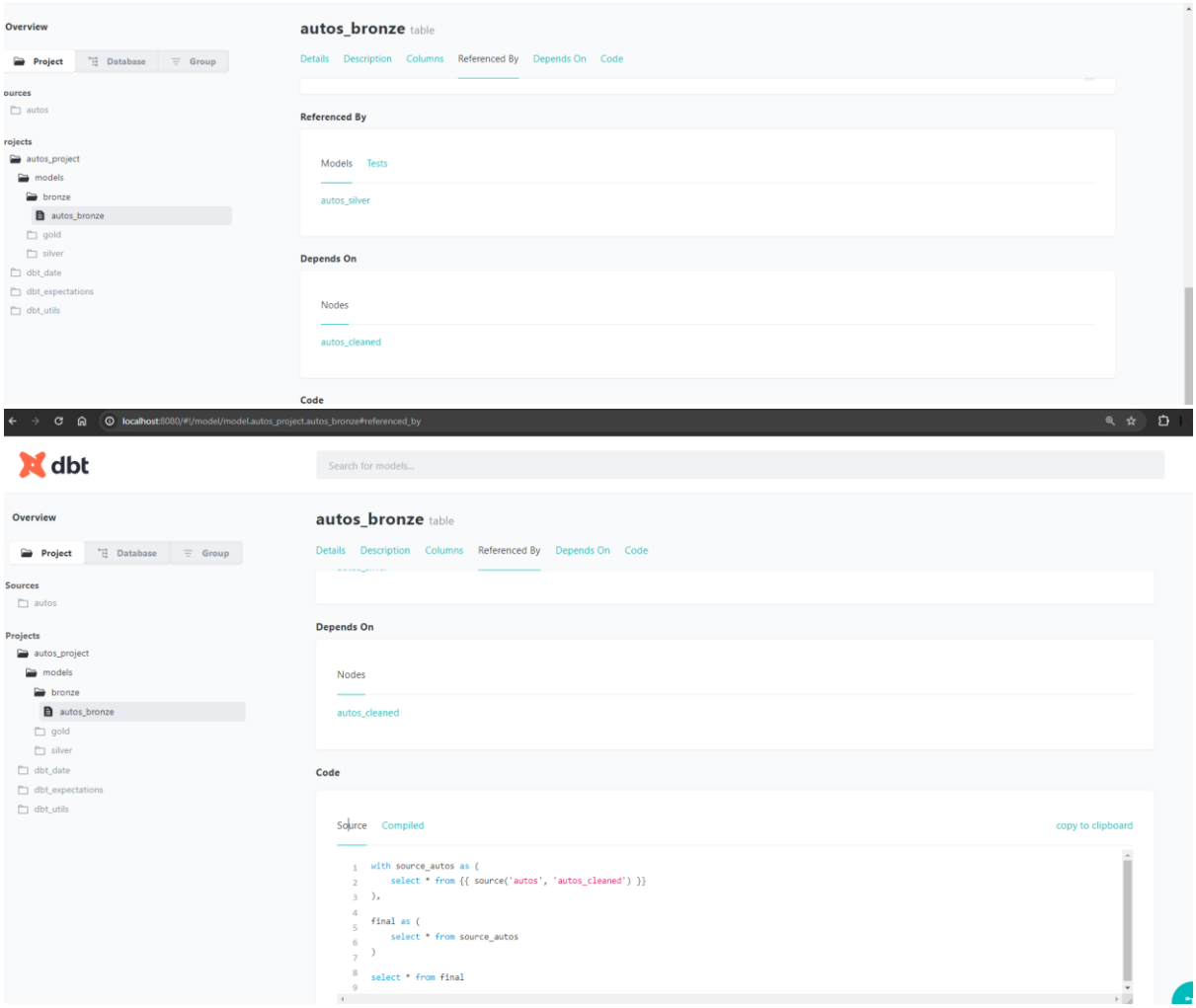
dbt docs serve



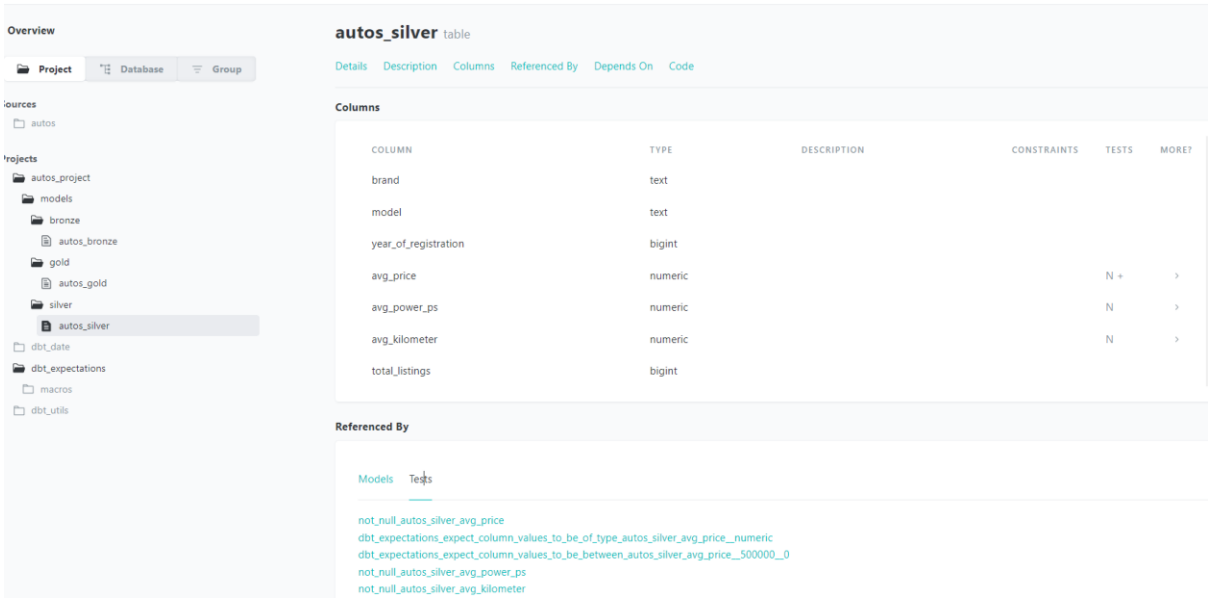
Na documentação interativa, você pode:

- Explorar modelos e fontes: Clique nos modelos para ver detalhes, incluindo descrições, colunas, tipos de dados e SQL gerado.
- Visualizar a linhagem de dados: Navegue pelo gráfico que mostra como os dados fluem de uma fonte para os modelos finais.
- Ver testes e resultados: Confira os testes aplicados aos modelos e suas últimas execuções.





Podemos ver que os testes funcionaram



Se desejar compartilhar a documentação com colegas ou integrá-la a sistemas de documentação maiores, você pode copiar os arquivos estáticos gerados (localizados geralmente em target/docs) para qualquer servidor web ou serviço de hospedagem de documentos estáticos.

Esses passos ajudarão você a criar uma documentação robusta e informativa para o seu projeto dbt, tornando mais fácil para as equipes entenderem e colaborarem no seu pipeline de dados.

Então podemos consultar a base Silver e gold no pgadmin, ou via pandas voce escolhe, o resultado esperado é
Camada Silver

	brand text	model text	year_of_registration bigint	avg_price numeric	avg_power_ps numeric	avg_kilometer numeric	total_listings bigint
1	peugeot	3_reihe	1991	637.25	75.50	120000.00	4
2	citroen	c1	2014	7375.00	68.00	20000.00	2
3	volkswagen	jetta	1995	1100.00	86.25	143750.00	4
4	fiat	500	1994	849.50	42.50	150000.00	2
5	volkswagen	jetta	1996	1373.17	80.00	133333.33	6
6	opel	bus	2010	6000.00	110.00	125000.00	1
7	skoda	fabia	2002	1968.70	75.92	138455.28	123
8	peugeot	4_reihe	1968	1850.00	76.00	20000.00	1
9	fiat	500	1970	6352.00	18.50	53500.00	10
10	mitsubishi	limousine	2006	1850.00	95.00	150000.00	1
11	alfa_romeo	spider	1990	9287.50	120.00	126250.00	4
12	opel	vectra	1996	917.38	106.54	139137.06	197
13	ford	galaxy	2010	15489.00	161.53	120666.67	15
14	opel	astra	2013	13763.80	137.69	54833.33	90
15	mercedes_benz	c_klasse	2014	23752.51	161.03	39487.18	39
16	suzuki	kleinwagen	2012	6700.00	68.00	40000.00	1
17	audi	a4	2016	4407.78	136.64	143441.56	77
18	honda	andere	1967	19000.00	67.00	100000.00	1
19	sonstige_autos	andere	1956	1300.00	34.00	10000.00	1
20	volkswagen	kombi	1997	1205.75	106.00	139642.86	28
21	chevrolet	andere	2016	2683.33	158.67	145833.33	6
22	skoda	octavia	2016	5023.64	117.57	96071.43	14
23	volkswagen	eos	2012	17493.73	145.97	53833.33	30
24	fiat	coupe	2000	6250.00	250.00	150000.00	1
25	volvo	c_reihe	2003	6572.25	183.50	143750.00	4
26	volkswagen	no_info	1996	1080.00	45.00	150000.00	1
27	daihatsu	charade	1900	500.00	89.00	90000.00	1

Camada Gold

	year_of_registration bigint	avg_price numeric	total_vehicles numeric
1	2016	4469.23	5999
2	2015	16269.82	1723
3	2014	16573.50	3578
4	2013	15246.24	5137
5	2012	13623.87	8139
6	2011	12623.22	10753
7	2010	10863.08	11146
8	2009	9288.68	14210
9	2008	8335.79	14374
10	2007	7199.55	15709
11	2006	6248.80	18098
12	2005	4913.37	18213
13	2004	4256.02	17345
14	2003	3576.35	17512
15	2002	3134.89	16809
16	2001	2629.02	17450
17	2000	2280.96	17932
18	1999	2007.96	18824
19	1998	2104.40	14478
20	1997	1874.56	11604
21	1996	1743.99	8514
22	1995	1878.13	6575
23	1994	2219.98	3768
24	1993	2569.74	2667
25	1992	2765.58	2392
26	1991	3014.10	2189
27	1990	3092.36	1774

Recaptulação

Hoje, na conclusão do nosso curso, aplicamos de forma prática e abrangente os conceitos que estudamos, focando na integração do DBT com o Great Expectations. Conseguimos estabelecer e configurar um projeto DBT chamado autos_project, que integramos ao PostgreSQL. Demonstramos a instalação e configuração do plugin dbt_expectations, adicionando uma camada robusta de validação que assegura a qualidade das nossas transformações de dados.

Desenvolvemos modelos para as camadas Bronze, Silver e Gold, cada uma refletindo um nível de refinamento e agregação dos dados, o que facilita a realização de análises

específicas e proporciona insights detalhados sobre o mercado de automóveis usados. Implementamos testes e validações essenciais, como a não nulidade e limites numéricos, utilizando tanto funcionalidades nativas do DBT quanto do plugin Great Expectations.

Além disso, geramos e utilizamos a documentação interativa do projeto, essencial para manter a equipe informada sobre a estrutura e lógica dos modelos. Finalizamos com uma análise das camadas Silver e Gold, evidenciando a transformação e apresentação das informações.

Encorajo todos a continuar explorando e criando seus próprios modelos e testes no DBT, utilizando diferentes fontes de dados e integrando novas ferramentas como o Great Expectations. Para aprofundar-se mais, recomendo estudar sobre práticas avançadas de CI/CD em pipelines de dados e participar ativamente da comunidade DBT e Great Expectations para trocar ideias e manter-se atualizado.

Nos vemos na próxima aula!