O que iremos aprender

Na aula de hoje iremos explorar como realizar validações com great expectations em um projeto com esquema de arquivos

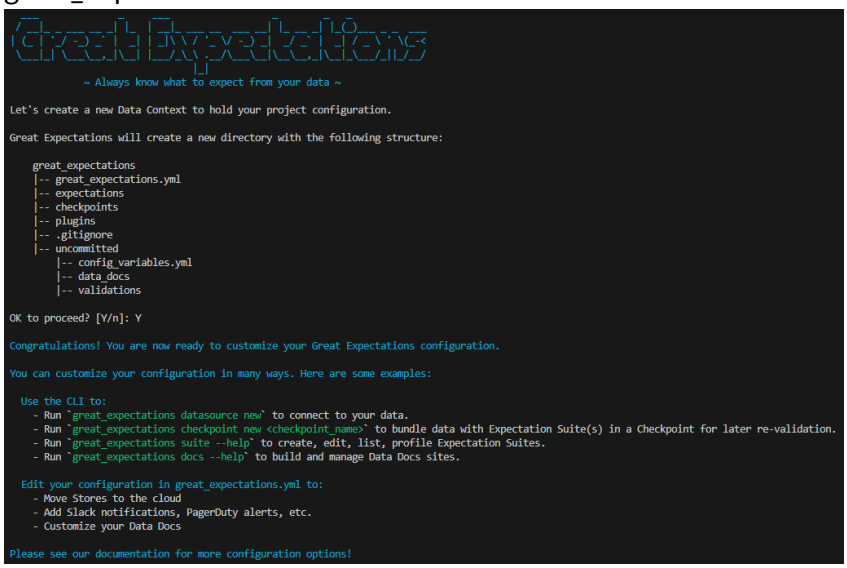
A ideia é explorarmos um cenário mais próximo do produtivo, ao invés de utilizarmos o jupyter notebook.

Hands on

Instalando o Great expectations
Bora lá então galera, o primeiro passo é se certificar que o Great Expectations está instalado em seu ambiente Python. Você pode fazer isso executando os seguintes comandos no seu terminal:
pip install great_expectations
great_expectations --version

Inicializando o projeto
Após executar esse comando, você pode executar o comando para inicializar um projeto do great expectations:

great_expectations init



A estrutura do projeto

```
great_expectations
    |-- great_expectations.yml # Este é um arquivo YAML que contém a configuração principal do projeto Great Expectations. Ele geralmente inclui informações como conexões de banco de dados, configurações de armazenamento e outras configurações de projeto.
    |-- expectations # Este é um diretório onde você pode armazenar os arquivos de especificações de expectativas. As expectativas são as regras ou condições que seus dados devem atender.
```

```
   |-- checkpoints # Este diretório pode conter checkpoints, que são conjuntos de expectativas que foram validadas em um determinado momento no tempo. Eles podem ser úteis para rastrear a qualidade dos dados ao longo do tempo.
   |-- plugins # Este diretório pode conter plugins adicionais para o Great Expectations. Os plugins podem adicionar funcionalidades extras, como suporte a diferentes tipos de fontes de dados ou integrações com outras ferramentas.
   |-- .gitignore # Este é um arquivo que especifica quais arquivos e diretórios devem ser ignorados pelo Git, um sistema de controle de versão. Isso geralmente inclui arquivos temporários, arquivos de compilação e outros arquivos que não devem ser versionados.
   |-- uncommitted # Este é um diretório onde você pode colocar arquivos que ainda não foram confirmados no controle de versão. Isso pode incluir arquivos de configuração, documentação ou resultados de validação que ainda não foram revisados ou aprovados para inclusão no repositório principal
       |-- config_variables.yml # Este arquivo YAML pode conter variáveis de configuração específicas do ambiente ou do projeto. Por exemplo, você pode definir variáveis de ambiente ou configurações de conexão de banco de dados aqui.
       |-- data_docs # Este diretório pode conter documentação gerada automaticamente sobre os dados, incluindo estatísticas, perfis de colunas e outras informações úteis para entender o conjunto de dados
       |-- validations # Este diretório pode conter resultados de validações anteriores, como logs ou relatórios de validação, que podem ser úteis para rastrear a qualidade dos dados ao longo do tempo.
```

A base que iremos extrair

Configurando o projeto

Agora que criamos o projeto podemos alterar o arquivo yaml great_expectations na raiz do projeto, adicionando o json a seguir para configurar o autos_cleansed.csv como um datasource

Execute:

great_expectations datasource new

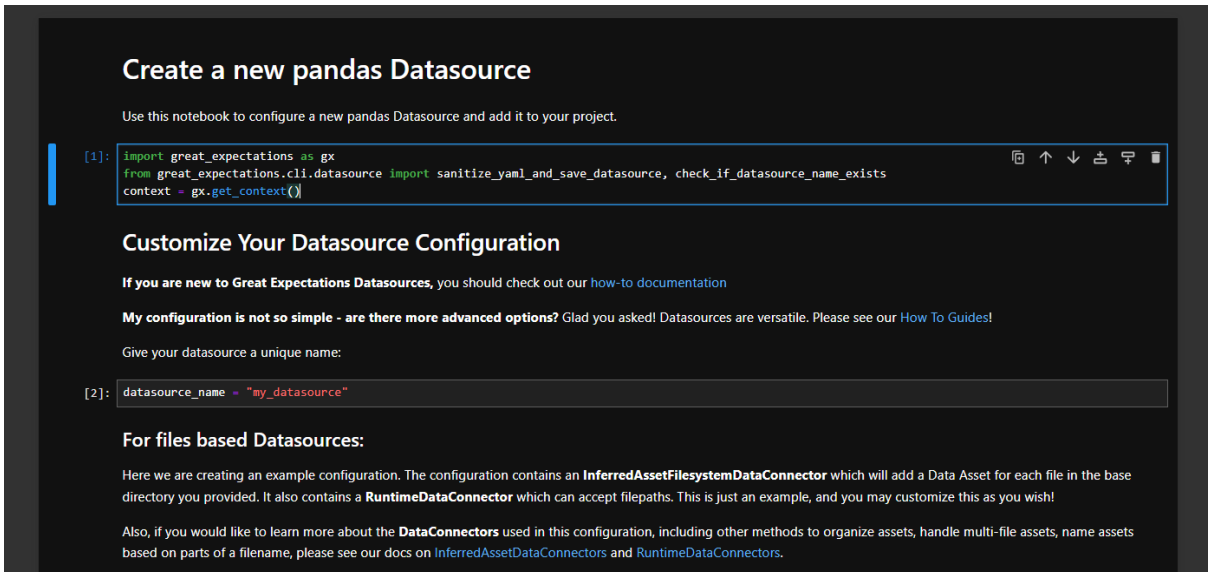
Escolha a opção 1 pandas

```
As of V0.16, the preferred method for adding a Datasource is using the fluent method of c
onfiguration, and not using the CLI.
You can read more about this here: https://greatexpectations.io/blog/the-fluent-way-to-co
nnect-to-data-sources-in-gx
If you would like to proceed anyway, press Y. [Y/n]: Y

What data would you like Great Expectations to connect to?
    1. Files on a filesystem (for processing with Pandas or Spark)
    2. Relational database (SQL)
: 1

What are you processing your files with?
1. Pandas
2. PySpark
: 1
```

Depois passe o caminho do .csv absoluto até a pasta onde estará o arquivo

```
Enter the path of the root directory where the data files are stored. If files are on loc
al disk enter a path relative to your current working directory or an absolute path.
: C:\Users\costa\workspace\ada-tech\engenharia-de-dados-cursos\analytics-engineering\aula
_5\dados
```

Execute o jupyter notebook que o great expectations abrir, que o arquivo yml vai ser automaticamente modificado



A célula do yml é a mais importante pois será a que realizará a configuração no yml principal do great expectations de um novo data source

```
[3]: example_yaml = f"""
name: {datasource_name}
class_name: Datasource
execution_engine:
  class_name: PandasExecutionEngine
data_connectors:
  default_inferred_data_connector_name:
    class_name: InferredAssetFilesystemDataConnector
    base_directory: ..\dados
    default_regex:
      group_names:
        - data_asset_name
      pattern: (.*)
  default_runtime_data_connector_name:
    class_name: RuntimeDataConnector
    assets:
      my_runtime_asset_name:
        batch_identifiers:
          - runtime_batch_identifier_name
"""
print(example_yaml)

name: my_datasource
class_name: Datasource
execution_engine:
  class_name: PandasExecutionEngine
data_connectors:
  default_inferred_data_connector_name:
    class_name: InferredAssetFilesystemDataConnector
    base_directory: ..\dados
    default_regex:
      group_names:
        - data_asset_name
      pattern: (.*)
  default_runtime_data_connector_name:
    class_name: RuntimeDataConnector
    assets:
      my_runtime_asset_name:
        batch_identifiers:
          - runtime_batch_identifier_name
```

É possível também testar esse yml no contexto do framework

```
[4]: context.test_yaml_config(yaml_config=example_yaml)

Attempting to instantiate class from config...
        Instantiating as a Datasource, since class_name is Datasource
        Successfully instantiated Datasource


ExecutionEngine class name: PandasExecutionEngine
Data Connectors:
        default_inferred_data_connector_name : InferredAssetFilesystemDataConnector

        Available data_asset_names (3 of 4):
                .ipynb_checkpoints (1 of 1): ['.ipynb_checkpoints']
                autos.csv (1 of 1): ['autos.csv']
                autos_cleaned.csv (1 of 1): ['autos_cleaned.csv']

        Unmatched data_references (0 of 0):[]

        default_runtime_data_connector_name:RuntimeDataConnector

        default_runtime_data_connector_name : RuntimeDataConnector

        Available data_asset_names (1 of 1):
                my_runtime_asset_name (0 of 0): []

        Unmatched data_references (0 of 0):[]

[4]: <great_expectations.datasource.new_datasource.Datasource at 0x1e7c5969310>
```

E a última célula executável realiza um commit da modificação no yml

Save Your Datasource Configuration ¶

Here we will save your Datasource in your Data Context once you are satisfied with the configuration. Note that `overwrite_existing` defaults to False, but you may change it to True if you wish to overwrite. Please note that if you wish to include comments you must add them directly to your `great_expectations.yml`.

```
[5]: sanitize_yaml_and_save_datasource(context, example_yaml, overwrite_existing=False)
     context.list_datasources()
```

```
**WARNING** A Datasource named "my_datasource" already exists in this Data Context. The Datasource has *not* been saved. Please use a different name or s
et overwrite_existing=True if you want to overwrite!
```

```
[5]: [{'name': 'my_datasource',
  'class_name': 'Datasource',
  'module_name': 'great_expectations.datasource',
  'execution_engine': {'class_name': 'PandasExecutionEngine',
   'module_name': 'great_expectations.execution_engine'},
  'data_connectors': {'default_inferred_data_connector_name': {'class_name': 'InferredAssetFilesystemDataConnector',
    'module_name': 'great_expectations.datasource.data_connector',
    'base_directory': '..\\dados',
    'default_regex': {'group_names': ['data_asset_name'], 'pattern': '(.*)'}},
   'default_runtime_data_connector_name': {'class_name': 'RuntimeDataConnector',
    'module_name': 'great_expectations.datasource.data_connector',
    'assets': {'my_runtime_asset_name': {'class_name': 'Asset',
      'module_name': 'great_expectations.datasource.data_connector.asset',
      'batch_identifiers': ['runtime_batch_identifier_name']}}}}}]
```

Now you can close this notebook and delete it!

O yml estará alterado conforme previsto

la_3 > gx > ! great_expectations.yml

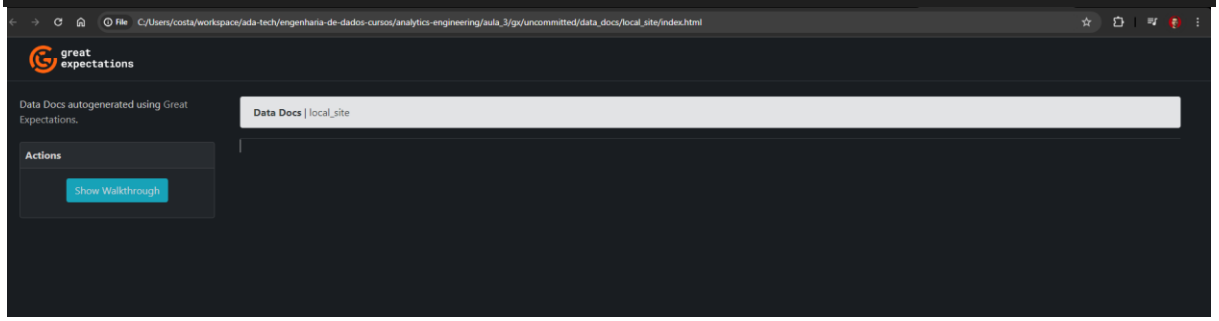
```yaml
#    - Join our slack channel: http://greatexpectations.io/slack

# config_version refers to the syntactic version of this config file, and is use
# It is auto-generated and usually does not need to be changed.
config_version: 3.0

# Datasources tell Great Expectations where your data lives and how to get it.
# Read more at https://docs.greatexpectations.io/docs/guides/connecting_to_your_
datasources:
  my_datasource:
    class_name: Datasource
    module_name: great_expectations.datasource
    execution_engine:
      class_name: PandasExecutionEngine
      module_name: great_expectations.execution_engine
    data_connectors:
      default_inferred_data_connector_name:
        class_name: InferredAssetFilesystemDataConnector
        module_name: great_expectations.datasource.data_connector
        base_directory: ..\dados
        default_regex:
          group_names:
            - data_asset_name
          pattern: (.*)
      default_runtime_data_connector_name:
        class_name: RuntimeDataConnector
        module_name: great_expectations.datasource.data_connector
        assets:
          my_runtime_asset_name:
            class_name: Asset
            module_name: great_expectations.datasource.data_connector.asset
            batch_identifiers:
              - runtime_batch_identifier_name
```

Após executar esses comandos o datasource estará criado, e então poderemos criar a nossa suite de testes

Buildando a documentação pela primeira vez

Agora que já definimos o arquivo csv como datasource, vamos configurar as nossas expectations, mas antes disso vamos iniciar o buildar nosso projeto, executando o comando abaixo:

```
great_expectations docs build
```



Podemos ver que não existe nenhuma suite de testes, ou execução criada.

Criando as expectativas

Execute o comando

great_expectations suite new

```
PS C:\Users\costa\workspace\ada-tech\engenharia-de-dados-cursos\analytics-engineering\aul
a_3\gx> great_expectations suite new

How would you like to create your Expectation Suite?
    1. Manually, without interacting with a sample Batch of data (default)
    2. Interactively, with a sample Batch of data
    3. Automatically, using a Data Assistant
: 3
```

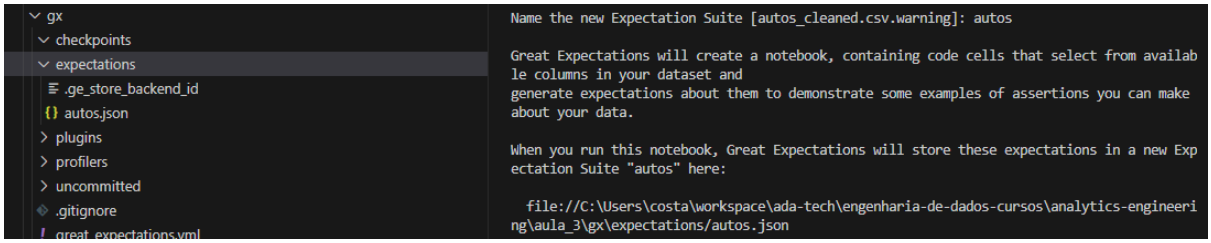
Passe o arquivo que você deseja utilizar, no nosso caso o autos_cleaned.csv

```
A batch of data is required to edit the suite - let's help you to specify it.


Which data asset (accessible by data connector "default_inferred_data_connector_name") wo
uld you like to use?
    1. .ipynb_checkpoints
    2. autos.csv
    3. autos_cleaned.csv
    4. test.csv

Type [n] to see the next page or [p] for the previous. When you're ready to select an ass
et, enter the index.
: 3
```

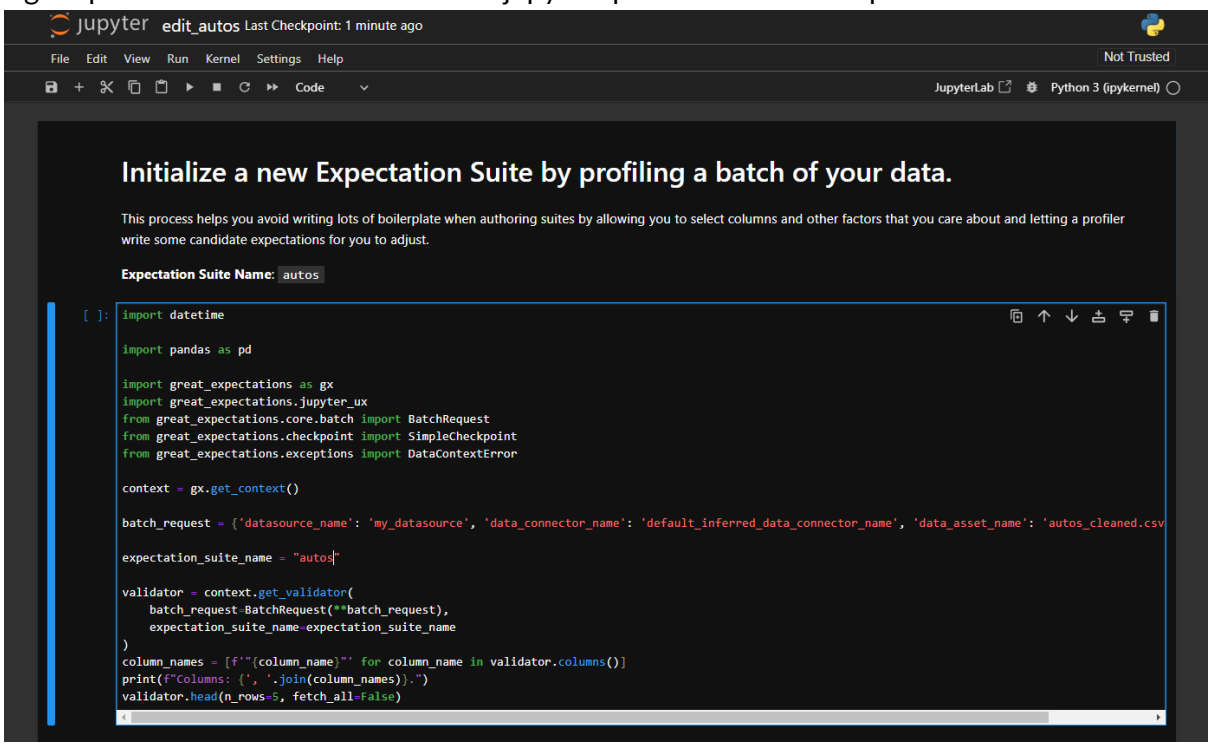
E então nomeie a suite de testes

```
Name the new Expectation Suite [autos_cleaned.csv.warning]: autos
```

Aperte Y para proceder e então será aberto um novo jupyter, a ideia desse é testar algumas expectations padrão de profilling, como o do y_data_profilling que vimos em aulas anteriores, além disso o arquivo autos.json também foi criado.

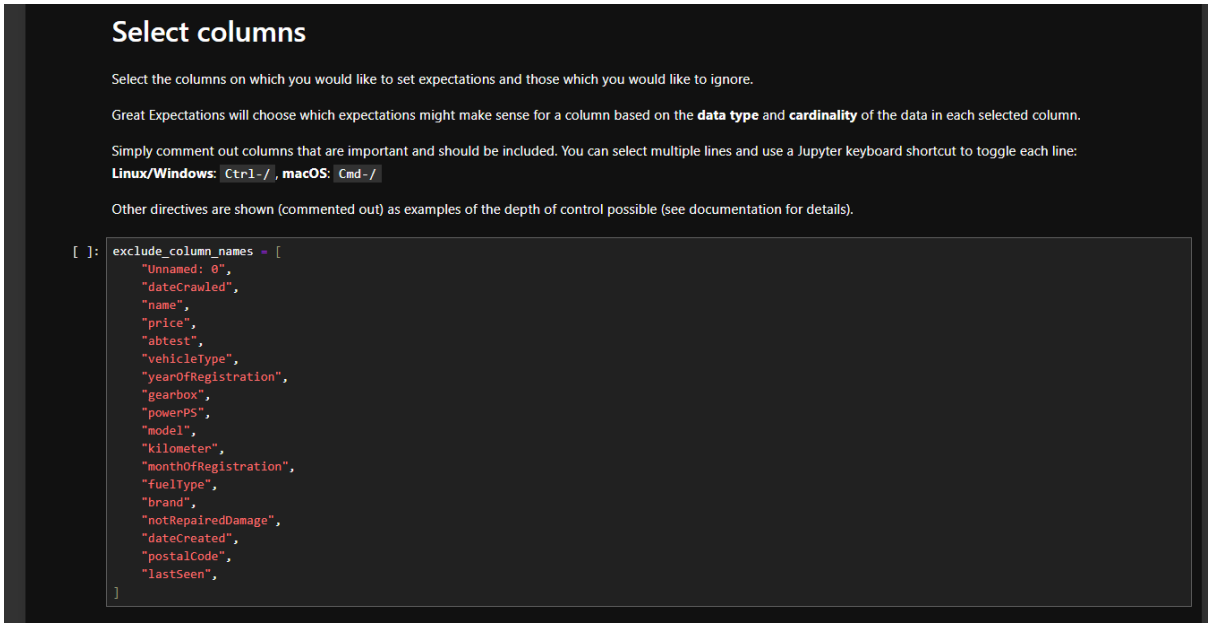


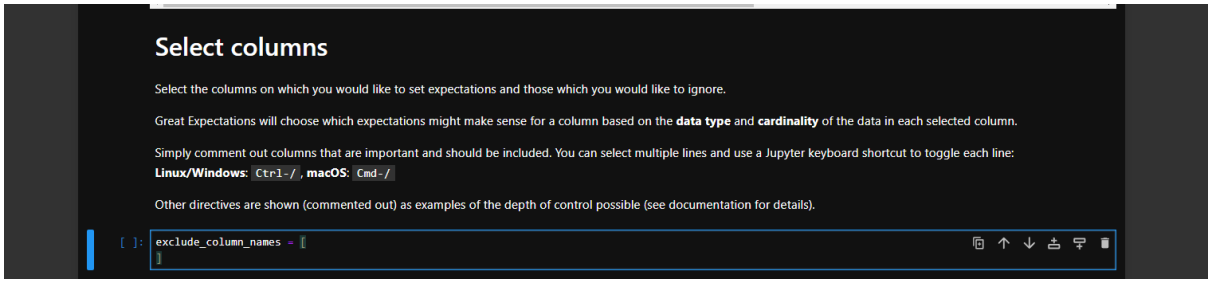
Agora podemos executar as células do jupyter que são bem auto explicativa



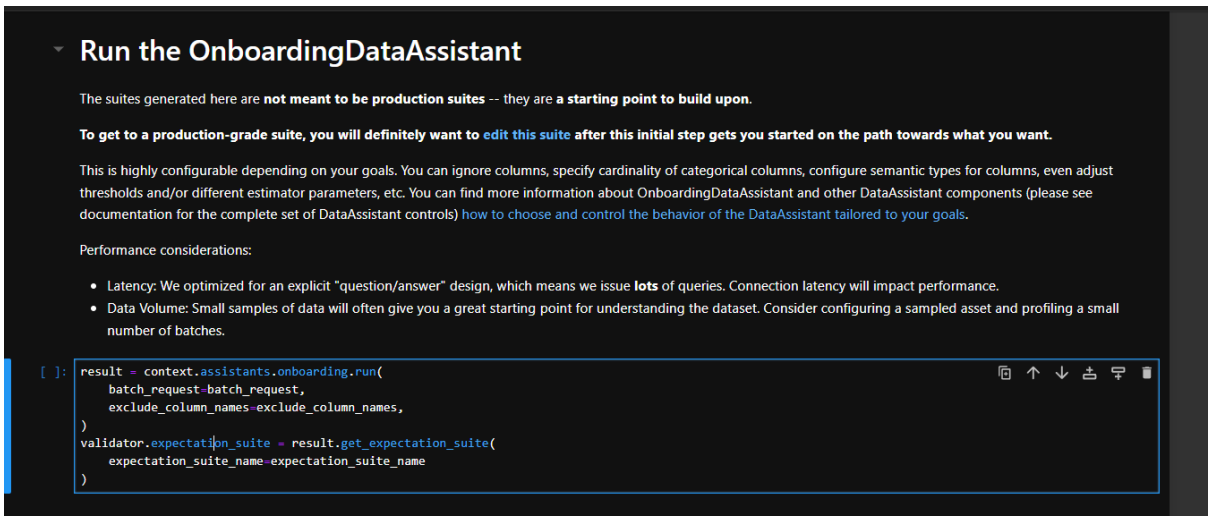
Essa primeira instancia um validador para criar a suite de fato

A segunda, nós podemos tirar as colunas ou adicionar, que queremos criar testes, eu vou tirar todas

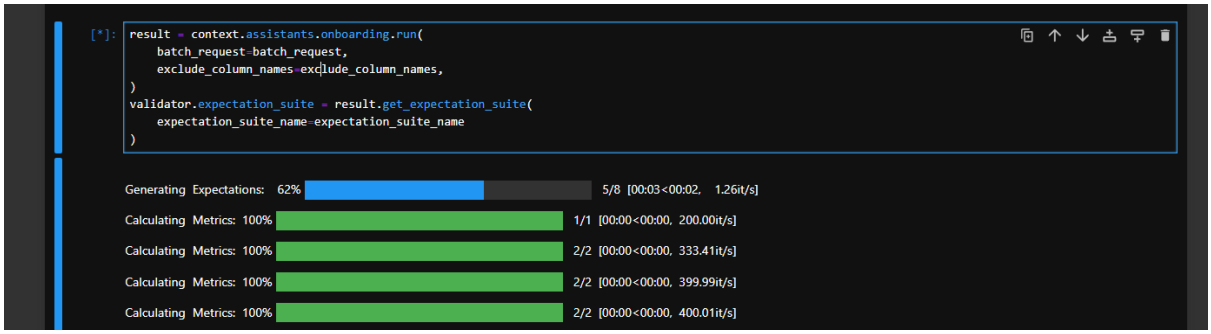




Cria a suite com as colunas que queremos, entao ele vai gerar uma serie de expectations para as colunas da tabela



Esse passo pode demorar um pouco



Agora vamos salvar o resultado da nossa suite, ele inclusive cria um checkpoint

Save & review your new Expectation Suite

Let's save the draft expectation suite as a JSON file in the `great_expectations/expectations` directory of your project and rebuild the Data Docs site to make it easy to review your new suite.

```
[ ]: print(validator.get_expectation_suite(discard_failed_expectations=False))
validator.save_expectation_suite(discard_failed_expectations=False)

checkpoint_config = {
    "class_name": "SimpleCheckpoint",
    "validations": [
        {
            "batch_request": batch_request,
            "expectation_suite_name": expectation_suite_name
        }
    ]
}
checkpoint = SimpleCheckpoint(
    f"{validator.active_batch_definition.data_asset_name}_{expectation_suite_name}",
    context,
    **checkpoint_config
)
checkpoint_result = checkpoint.run()

context.build_data_docs()

validation_result_identifier = checkpoint_result.list_validation_result_identifiers()[0]
context.open_data_docs(resource_identifier=validation_result_identifier)
```

Podemos ver que foram executadas mais de 100 expectations

```
2024-04-18T18:38:42-0300 - INFO -       110 expectation(s) included in expectation_suite.
{
  "expectation_suite_name": "autos",
  "ge_cloud_id": null,
  "expectations": [
    {
      "expectation_type": "expect_table_row_count_to_be_between",
      "kwargs": {
        "min_value": 303950,
        "max_value": 303950
      },
      "meta": {
        "profiler_details": {
          "metric_configuration": {
            "metric_name": "table.row_count",
            "domain_kwargs": {},
            "metric_value_kwargs": null
```

É possível alterar a suite com o comando

Next steps

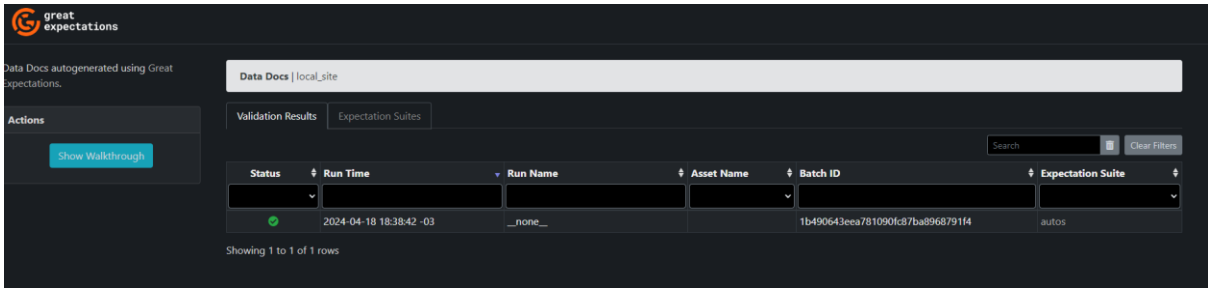
After you review this initial Expectation Suite in Data Docs you should edit this suite to make finer grained adjustments to the expectations. This can be done by running `great_expectations suite edit autos`.

Mas também é possivel alterar via yml

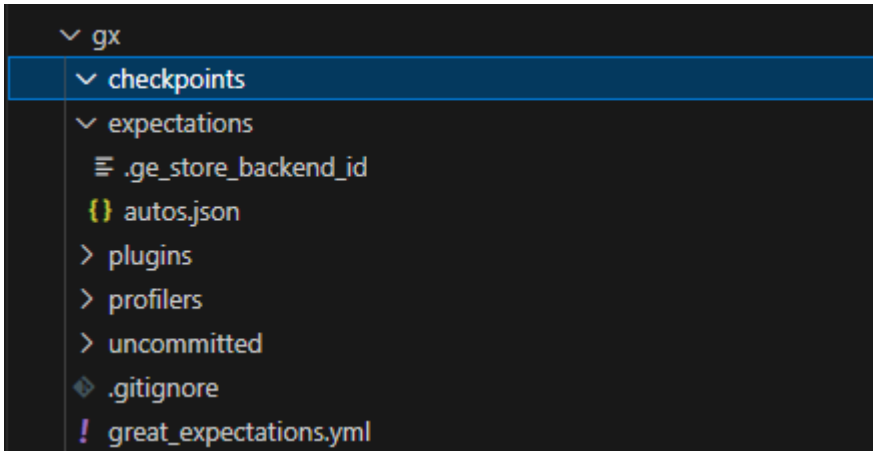
Vamos executar o comando

great_expectations docs build

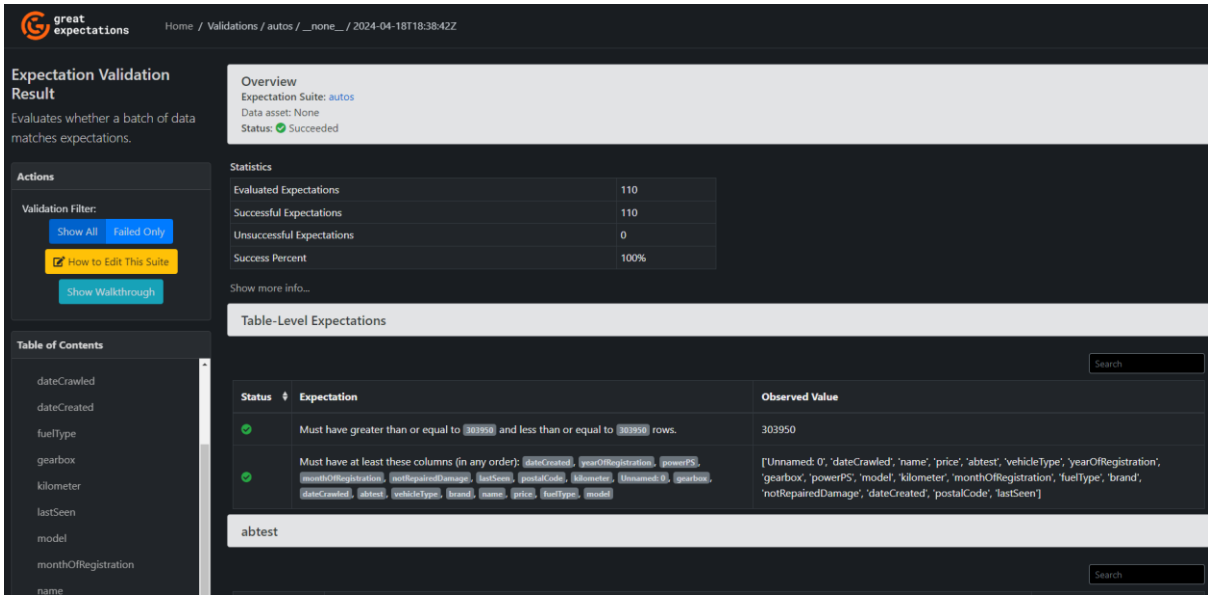
Para avaliarmos a suíte que foi executada



Podemos ver, que o checkpoint foi executado, mas nesse caso não foi criado um yml



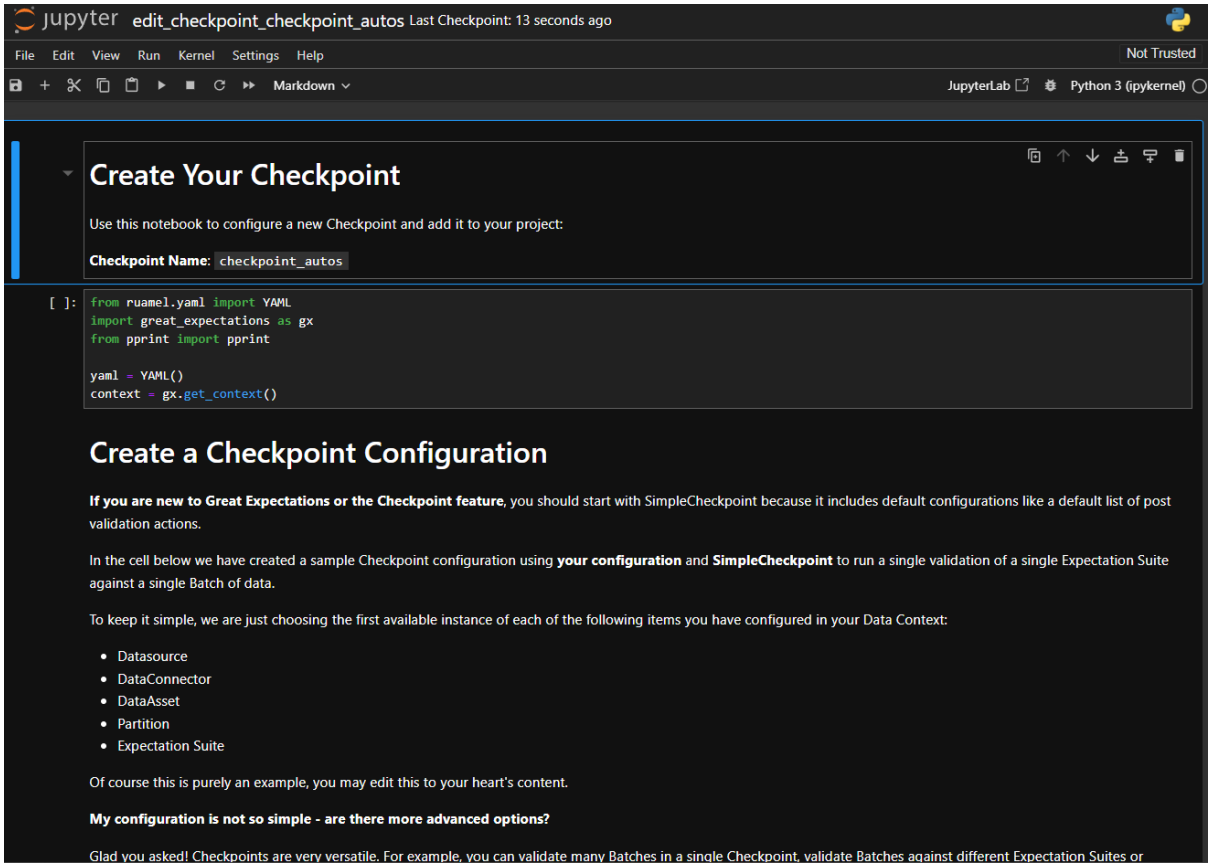
Podemos verificar que o gx conseguiu criar 110 expectativas automáticas que tiveram 100% de sucesso para todas as colunas da tabela, isso mostra o poder da ferramenta que permite uma cobertura de testes enorme com pouco esforço de configuração e de forma simples



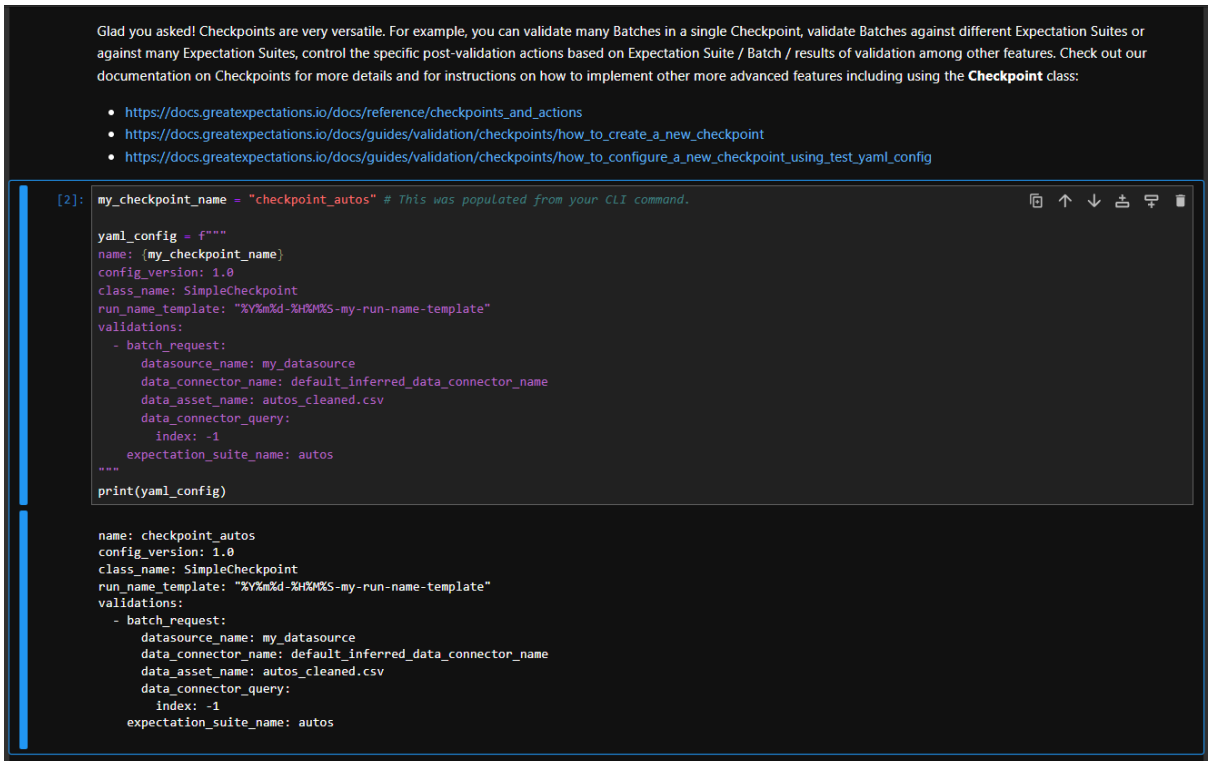
Agora vamos criar um checkpoint para deixar parametriza e reutilizável, o que utilizamos foi somente em memória via jupyter

great_expectations checkpoint new checkpoint_autos

Novamente vamos para uma interface jupyter



E vamos executar novamente as células, que gerarão nosso checkpoint final



Yaml gerado

Customize Your Configuration

The following cells show examples for listing your current configuration. You can replace values in the sample configuration with these values to customize your Checkpoint.

```
[ ]: # Run this cell to print out the names of your Datasources, Data Connectors and Data Assets
pprint(context.get_available_data_asset_names())
```

```
[ ]: context.list_expectation_suite_names()
```

Test Your Checkpoint Configuration

Here we will test your Checkpoint configuration to make sure it is valid.

This `test_yaml_config()` function is meant to enable fast dev loops. If your configuration is correct, this cell will show a message that you successfully instantiated a Checkpoint. You can continually edit your Checkpoint config yaml and re-run the cell to check until the new config is valid.

If you instead wish to use python instead of yaml to configure your Checkpoint, you can use `context.add_checkpoint()` and specify all the required parameters.

```
[ ]: my_checkpoint = context.test_yaml_config(yaml_config=yaml_config)
```

Review Your Checkpoint

You can run the following cell to print out the full yaml configuration. For example, if you used **SimpleCheckpoint** this will show you the default action list.

```
[ ]: print(my_checkpoint.get_config(mode="yaml"))
```

Add Your Checkpoint

Run the following cell to save this Checkpoint to your Checkpoint Store.

```
[ ]: context.add_checkpoint(**yaml.load(yaml_config))
```

É possivel verificar como ficara o checkpoint final

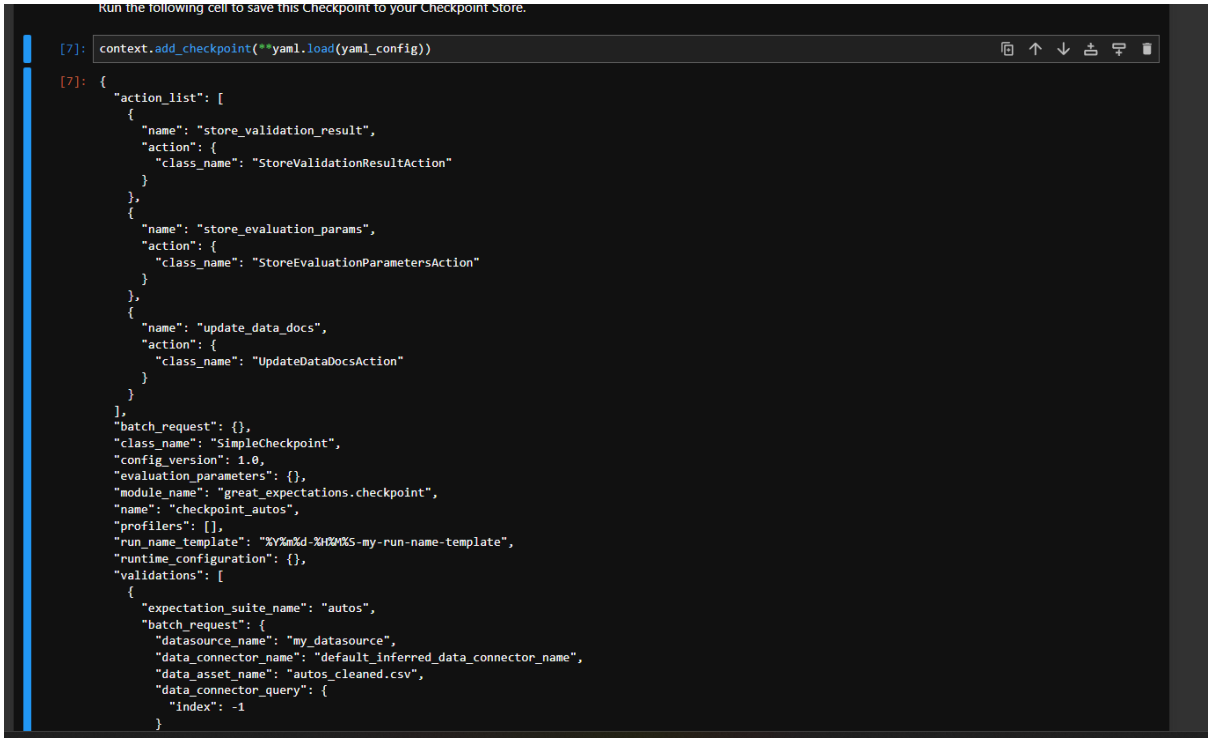
Review Your Checkpoint

You can run the following cell to print out the full yaml configuration. For example, if you used **SimpleCheckpoint** this will show you the default action list.

```
[6]: print(my_checkpoint.get_config(mode="yaml"))
```

```yaml
name: checkpoint_autos
config_version: 1.0
template_name:
module_name: great_expectations.checkpoint
class_name: SimpleCheckpoint
run_name_template: '%Y%m%d-%H%M%S-my-run-name-template'
expectation_suite_name:
batch_request: {}
action_list:
  - name: store_validation_result
    action:
      class_name: StoreValidationResultAction
  - name: store_evaluation_params
    action:
      class_name: StoreEvaluationParametersAction
  - name: update_data_docs
    action:
      class_name: UpdateDataDocsAction
evaluation_parameters: {}
runtime_configuration: {}
validations:
  - batch_request:
      datasource_name: my_datasource
      data_connector_name: default_inferred_data_connector_name
      data_asset_name: autos_cleaned.csv
      data_connector_query:
        index: -1
    expectation_suite_name: autos
profilers: []
ge_cloud_id:
expectation_suite_ge_cloud_id:
```

E o último comando, realiza o add checkpoint, e materializa o arquivo yml



Como podemos ver, o yml está criado corretamente na pasta checkpoints do projeto

```yaml
name: checkpoint_autos
config_version: 1.0
template_name:
module_name: great_expectations.checkpoint
class_name: SimpleCheckpoint
run_name_template: '%Y%m%d-%H%M%S-my-run-name-template'
expectation_suite_name:
batch_request: {}
action_list:
  - name: store_validation_result
    action:
      class_name: StoreValidationResultAction
  - name: store_evaluation_params
    action:
      class_name: StoreEvaluationParametersAction
  - name: update_data_docs
    action:
      class_name: UpdateDataDocsAction
evaluation_parameters: {}
runtime_configuration: {}
validations:
  - batch_request:
      datasource_name: my_datasource
      data_connector_name: default_inferred_data_connector_name
      data_asset_name: autos_cleaned.csv
      data_connector_query:
        index: -1
    expectation_suite_name: autos
profilers: []
ge_cloud_id:
expectation_suite_ge_cloud_id:

```

Gerando o relatório uma última vez

Ao gerar o data docs novamente, podemos verificar que uma nova execução foi pontuada pelo great expectations, e agora conseguimos compartilhar esse arquivo yaml de checkpoint e trabalhar a partir dele!

Recapitulação

Na aula de hoje, exploramos o Great Expectations em um cenário mais próximo do ambiente produtivo, utilizando uma estrutura de arquivos para organizar o projeto. Começamos instalando o Great Expectations e inicializando um novo projeto. Em seguida, configuramos o datasource para um arquivo CSV específico e criamos expectativas sobre os dados.

Ao construir as expectativas, utilizamos uma interface Jupyter para testar algumas expectativas padrão de profiling e gerar um conjunto de expectativas para as colunas da tabela. Essas expectativas foram automaticamente executadas e verificadas, mostrando o poder da ferramenta para uma cobertura de testes extensa com pouco esforço de configuração.

Após criar as expectativas, construímos a documentação do projeto, permitindo uma visualização clara das execuções e resultados das validações. Geramos um checkpoint para tornar as execuções reutilizáveis e parametrizadas, o que facilita a validação contínua dos dados.

Em resumo, aprendemos a instalar o Great Expectations, inicializar um projeto, configurar o datasource, criar expectativas, construir a documentação e criar checkpoints para validações contínuas dos dados. Essa abordagem oferece uma maneira poderosa e simples de garantir a qualidade dos dados em um ambiente de análise de dados, facilitando a manutenção da confiabilidade dos dados ao longo do tempo.

Nos vemos na próxima aula!