Validando dados com Great Expectations

Antes de começarmos a discutir se os dados atendem ou não às expectativas, precisamos definir o que queremos dizer com "expectativas". Em geral, as expectativas se referem ao que acreditamos que os dados devem mostrar com base em nossas hipóteses ou intuições sobre o fenômeno que estamos investigando.

Esta verificação é fundamental para avaliar a validade das análises que realizamos e para detectar possíveis erros ou problemas em nossos dados. Neste tópico, vamos discutir alguns métodos e técnicas que podemos usar em Python para verificar se os dados atendem às nossas expectativas:

### Verificando a distribuição dos dados

Uma das primeiras coisas que podemos fazer para avaliar se os dados atendem às nossas expectativas é verificar a distribuição dos dados. Em muitos casos, esperamos que os dados sigam uma distribuição normal, especialmente se estamos trabalhando com dados contínuos.

Podemos fazer isto usando o teste de normalidade de Shapiro-Wilk, disponível na biblioteca scipy.stats, que testa a hipótese nula de que os dados seguem uma distribuição normal. Veja abaixo:

import numpy as np

from scipy.stats import shapiro

# criar um array de dados com distribuição normal

data\_normal = np.random.normal(loc=0, scale=1, size=1000)

# criar um array de dados com distribuição não-normal

data\_nao\_normal = np.random.exponential(scale=1, size=1000)

# realizar o teste de Shapiro-Wilk para os dados com distribuição normal

stat, p = shapiro(data\_normal)

print('Dados com distribuição normal:')

print('Estatística de teste:', stat)

print('Valor p:', p)

if p > 0.05:

print('Não podemos rejeitar a hipótese nula de que os dados seguem uma distribuição normal')

else:

print('Podemos rejeitar a hipótese nula de que os dados seguem uma distribuição normal')

# realizar o teste de Shapiro-Wilk para os dados com distribuição não-normal

stat, p = shapiro(data\_nao\_normal)

print('Dados com distribuição não-normal:')

print('Estatística de teste:', stat)

print('Valor p:', p)

if p > 0.05:

print('Não podemos rejeitar a hipótese nula de que os dados seguem uma distribuição normal')

else:

print('Podemos rejeitar a hipótese nula de que os dados seguem uma distribuição normal')

### Verificando a correlação entre as variáveis

Outro aspecto importante a considerar ao avaliar se os dados atendem às nossas expectativas é a correlação entre as variáveis, visto que geralmente é esperado que as variáveis relacionadas estejam, de fato, correlacionadas.

Para tanto, podemos usar a matriz de correlação, que está disponível na biblioteca Pandas e mostra a correlação entre todas as variáveis do conjunto de dados. Segue abaixo um exemplo simplificado e conciso de como esta validação pode ser feita:

import pandas as pd

# criar um dataframe com duas variáveis correlacionadas

data\_corr = pd.DataFrame({'var1': np.random.normal(loc=0, scale=1, size=1000),

'var2': np.random.normal(loc=0, scale1, size=1000) + np.random.normal(loc=0, scale=0.1, size=1000)})

calcular a matriz de correlação

corr\_matrix = data\_corr.corr()

imprimir a matriz de correlação

print('Matriz de correlação:')

print(corr\_matrix)

### Verificando a presença de valores ausentes

Como visto no tópico de Data Quality, uma outra questão que precisamos considerar ao avaliar se os dados atendem às nossas expectativas é a presença de valores ausentes. Em geral, esperamos que não haja muitos valores ausentes no conjunto de dados, pois isso pode afetar as análises e resultados obtidos. O (tão falado) Pandas possui uma função isna() que retorna um dataframe booleano indicando quais células do dataframe contêm valores ausentes:

# criar um dataframe com valores ausentes

data\_faltante = pd.DataFrame({'var1': [1, 2, np.nan, 4],

'var2': [5, np.nan, 7, 8]})

# verificar a presença de valores ausentes

print('Valores ausentes:')

print(data\_faltante.isna())

### Verificando a presença de outliers

Por fim, uma questão importante a ser considerada ao avaliar se os dados atendem às nossas expectativas é a presença de outliers. Em geral, outliers são valores extremos que diferem significativamente dos outros valores do conjunto de dados e podem afetar as análises e resultados obtidos, especialmente em modelos estatísticos.

Para verificar a presença de outliers, podemos usar o boxplot, que está disponível na biblioteca seaborn. Este recurso nos permite visualizar a distribuição dos dados com o destaque para possíveis outliers. Veja abaixo:

import seaborn as sns

# criar um array de dados com outliers

data\_outliers = np.concatenate([np.random.normal(loc=0, scale=1, size=950), [10, -10]])

# criar um gráfico boxplot

sns.boxplot(data=data\_outliers)

## Framework Great Expectations

O framework Great Expectations é uma biblioteca de código aberto no Python que nos permite definir, validar e documentar as expectativas dos nossos dados. É considerada uma ferramenta poderosa usada para ajudar a garantir a qualidade e confiabilidade dos dados que estamos usando em nossas análises e modelos.

Vamos dividir este estudo em algumas seções cobrindo as principais funcionalidades do framework.

### Instalando o Great Expectations

Para começar, precisamos instalar o Great Expectations. Podemos fazer isso facilmente usando o pip, o gerenciador de pacotes do Python.

Para instalar a versão mais recente, podemos executar o seguinte comando em seu prompt de comando:

!pip install great\_expectations

### Definindo expectativas dos dados

Uma vez instalado, podemos começar a definir as expectativas dos nossos dados. Elas descrevem as propriedades que esperamos que nossos dados tenham. Por exemplo, podemos esperar que uma coluna de idade contenha apenas valores numéricos positivos ou que uma coluna de endereços de e-mail contenha apenas endereços de e-mail válidos.

Podemos definir as expectativas dos nossos dados usando a classe ExpectationSuite do Great Expectations. Uma ExpectationSuite é um conjunto de expectativas que se aplicam a um conjunto de dados.

Exemplo:

import great\_expectations as ge

# criar uma ExpectationSuite

suite = ge.dataset.ExpectationSuite('meus\_dados')

# definir uma expectativa para verificar se uma coluna de idade contém apenas valores numéricos positivos

suite.expect\_column\_values\_to\_be\_of\_type(column='idade', type\_='int', mostly=0.9)

# definir uma expectativa para verificar se uma coluna de endereços de e-mail contém apenas endereços de e-mail válidos

suite.expect\_column\_values\_to\_match\_regex(column='email', regex=r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$', mostly=0.9)

### Validando os dados

Depois de definir as expectativas dos nossos dados, podemos usar o Great Expectations para validá-los. A validação verifica se as expectativas definidas são atendidas pelos dados e, caso não sejam, onde e por quê.

Com a classe DataContext, que representa um ambiente Great Expectations, é possível gerenciar as configurações, as ExpectationSuites, as validações e outros recursos do Great Expectations:

# criar um objeto DataContext

context = ge.data\_context.DataContext()

# definir a fonte de dados

batch\_kwargs = {'path': 'caminho/para/meus/dados.csv', 'sep': ','}

# carregar os dados como um Batch de dados do Great Expectations

batch = context.get\_batch(batch\_kwargs, suite)

# executar a validação

result = batch.validate()

# imprimir o resultado da validação

print(result)

### Documentando os dados

Além de definir e validar as expectativas dos dados, o Great Expectations também nos permite documentar os dados. A documentação ajuda a garantir que os usuários possam entender os dados e usá-los com confiança e precisão.

Podemos documentar os dados usando a classe DataDocumentation do Great Expectations, que contém informações sobre o conjunto de dados, como a descrição do conjunto de dados, a descrição das colunas e as notas do usuário:

# criar uma DataDocumentation

doc = ge.data\_documentation.DataDocumentation()

# definir a descrição do conjunto de dados

doc.set\_key('descrição', 'Este conjunto de dados contém informações sobre clientes.')

# definir a descrição das colunas

doc.add\_column\_info('id', {'descrição': 'O ID do cliente.', 'tipo': 'int'})

doc.add\_column\_info('nome', {'descrição': 'O nome do cliente.', 'tipo': 'string'})

doc.add\_column\_info('idade', {'descrição': 'A idade do cliente.', 'tipo': 'int'})

doc.add\_column\_info('email', {'descrição': 'O endereço de e-mail do cliente.', 'tipo': 'string'})

# definir notas do usuário

doc.set\_key('notas\_do\_usuario', 'Este conjunto de dados foi coletado em janeiro de 2022.')

### Agendando validações

Uma vez que definimos e validamos nossas expectativas, podemos querer agendar a validação para ser executada regularmente para garantir que nossos dados continuem atendendo às expectativas.

Podemos agendar uma operação de validação usando a classe ValidationOperator do Great Expectations.

Exemplo:

# criar um objeto ValidationOperator

validator = ge.validator.ValidationOperator(

data\_context=context,

batches=[batch\_kwargs],

validation\_specification\_name=suite.expectation\_suite\_name,

name='validação'

)

# agendar a validação para ser executada diariamente às 10h

validator.create\_schedule(

'daily',

start\_date='2022-01-01',

interval='1d',

time='10:00'

)

### Definindo expectativas de relacionamentos

Além de definir expectativas de colunas individuais, podemos definir expectativas de relacionamentos entre colunas. Por exemplo, podemos definir que o valor de uma coluna deve ser maior que uma segunda coluna.

Exemplo:

# definir uma expectativa de soma de colunas

suite.expect\_column\_pair\_values\_a\_to\_be\_greater\_than\_b(

'coluna\_1',

'coluna\_2'

)

### Definindo expectativas de conjuntos de dados

Além de definir expectativas de colunas individuais, podemos definir expectativas para conjuntos de dados inteiros. Por exemplo, podemos definir que o conjunto de dados deve ter um número mínimo de linhas.

Exemplo:

# definir uma expectativa de número mínimo de linhas

suite.expect\_table\_row\_count\_to\_be\_between(min\_value=100, max\_value=10000)

### Validando dados em diferentes sistemas de armazenamento

O Great Expectations suporta a validação de dados em diferentes sistemas de armazenamento, como bancos de dados, arquivos CSV e Data Lakes. Para fazer isso, usamos diferentes "datasources" (fontes de dados) para carregar os dados e definir as expectativas.

Exemplo:

# carregar dados de um banco de dados usando um datasource

batch\_kwargs = {

'datasource': 'my\_db',

'table': 'meu\_tabela'

}

batch = context.get\_batch(batch\_kwargs, suite)

# definir e validar expectativas

result = validator.run(batch, run\_id='validação\_1')

### Definindo expectativas de dados distribuídos

O Great Expectations suporta a validação de dados distribuídos usando diferentes ferramentas, como Apache Spark. Para fazer isso, usamos diferentes "batch\_kinds" (tipos de lote) para carregar os dados e definir as expectativas.

Exemplo:

# carregar dados distribuídos usando Spark usando um batch\_kind

batch\_kwargs = {

'batch\_kwargs': {

'data\_asset\_name': 'meus\_dados',

'dataset': spark\_df

},

'batch\_markers': {},

'batch\_parameters': {},

'batch\_spec\_passthrough': {}

}

batch = context.get\_batch(batch\_kwargs, suite, batch\_kind='spark')

# definir e validar expectativas

result = validator.run(batch, run\_id='validação\_1')

## Técnicas mais avançadas no Great Expectations

### Customização de regras de validação

O Great Expectations oferece uma grande variedade de regras de validação padrão, que podem ser usadas para validar uma ampla gama de tipos de dados. No entanto, em alguns casos, podemos querer criar nossas próprias regras de validação personalizadas para nossos dados específicos.

Para tanto, precisamos criar uma função Python que execute a validação e retorne um objeto de resultado. Em seguida, podemos registrar essa função com o Great Expectations usando o método add\_expectation().

Exemplo:

from great\_expectations.dataset import PandasDataset

def validate\_values\_within\_range(dataset, column, min\_value, max\_value):

values = dataset[column]

num\_values\_outside\_range = len(values[(values < min\_value) | (values > max\_value)])

return {

'success': num\_values\_outside\_range == 0,

'result': {

'observed\_value': num\_values\_outside\_range,

'element\_count': len(values),

'missing\_count': values.isna().sum()

}

}

# registrar a regra de validação personalizada

PandasDataset.add\_expectation(

expectation\_type='expect\_column\_values\_within\_range',

func=validate\_values\_within\_range

)

# usar a regra de validação personalizada

my\_data = pd.read\_csv('meus\_dados.csv')

result = my\_data.expect\_column\_values\_within\_range(column='coluna\_1', min\_value=0, max\_value=10)

### Documentação de expectativas

Documentar as expectativas é uma parte importante da validação de dados, pois nos ajuda a entender o que estamos validando e por quê. O Great Expectations permite que documentemos nossas expectativas de diferentes maneiras.

Podemos usar o método add\_expectation() para adicionar uma descrição da expectativa, bem como tags e exemplos. Também podemos usar o método expect\_column\_to\_exist() para documentar se uma coluna existe ou não em um conjunto de dados.

Exemplo:

from great\_expectations.dataset import PandasDataset

# adicionar descrição, tags e exemplos para uma expectativa

PandasDataset.add\_expectation(

expectation\_type='expect\_column\_values\_to\_be\_in\_set',

column='coluna\_1',

value\_set=[1, 2, 3],

description='A coluna deve conter apenas valores de 1, 2 ou 3.',

tags=['validação', 'conjunto de dados'],

examples=[

{

'data': {'coluna\_1': 1},

'title': 'Exemplo de valor válido'

},

{

'data': {'coluna\_1': 4},

'title': 'Exemplo de valor inválido'

}

]

)

# documentar se uma coluna existe ou não em um conjunto de dados

my\_data = pd.read\_csv('meus\_dados.csv')

my\_data.expect\_column\_to\_exist('coluna\_1', column\_index=0)

### Perfil dos dados

O Great Expectations também permite criar perfis de dados, que são resumos estatísticos de um conjunto de dados. Com isto, podemos entender melhor nossos dados e a criar expectativas mais informadas.

Podemos criar um perfil de dados usando o método profile() do objeto Dataset. Este método gera um dicionário que contém informações estatísticas sobre cada coluna do conjunto de dados, incluindo valores mínimos e máximos, média, mediana, desvio padrão e percentis:

from great\_expectations.dataset import PandasDataset

my\_data = pd.read\_csv('meus\_dados.csv')

profile = my\_data.profile()

# imprimir informações do perfil de dados

print(profile['colunas']['coluna\_1'])

print(profile['colunas']['coluna\_2'])

Além disso, podemos usar o método expect\_column\_values\_to\_be\_in\_set() com o argumento value\_set='unique' para verificar se uma coluna contém valores únicos ou repetidos.

Exemplo:

from great\_expectations.dataset import PandasDataset

my\_data = pd.read\_csv('meus\_dados.csv')

# verificar se a coluna contém valores únicos

my\_data.expect\_column\_values\_to\_be\_in\_set('coluna\_1', value\_set='unique')

Essas são algumas das técnicas de customização de regras de validação, documentação e perfil de dados que podem ser realizadas usando o Great Expectations em Python. Com essas funcionalidades, podemos criar expectativas personalizadas para nossos dados e garantir que eles atendam às nossas necessidades de validação.

### Validação de dados em lote

O Great Expectations pode ser usado para validar dados em lote, o que significa que podemos definir expectativas em um conjunto de dados e aplicá-las a outros conjuntos de dados semelhantes. Isso é útil quando precisamos validar dados regularmente, como em um pipeline de processamento de dados.

Exemplo:

from great\_expectations.dataset import PandasDataset

from great\_expectations.validator.validator import Validator

# carregar o conjunto de dados de referência

reference\_data = pd.read\_csv('referencia\_dados.csv')

reference\_dataset = PandasDataset(reference\_data)

# criar um objeto Validator

validator = Validator(reference\_dataset)

# carregar o novo conjunto de dados a ser validado

new\_data = pd.read\_csv('novo\_dados.csv')

new\_dataset = PandasDataset(new\_data)

# validar o novo conjunto de dados

results = validator.validate(new\_dataset)

# imprimir resultados de validação

print(results)

### Integração com outras ferramentas de processamento de dados

O Great Expectations pode ser integrado com outras ferramentas de processamento de dados, como Apache Airflow e Databricks. Isso permite que as expectativas sejam definidas e validadas automaticamente como parte de um pipeline de processamento de dados.

Exemplo:

from great\_expectations.data\_context import DataContext

# carregar o contexto de dados

context = DataContext()

# definir expectativas para o conjunto de dados

context.add\_expectation\_suite('meu\_suite', overwrite\_existing=True)

context.add\_expectation\_to\_suite('meu\_suite', 'expect\_column\_values\_to\_be\_in\_set', 'coluna\_1', value\_set=[1, 2, 3])

# validar o conjunto de dados automaticamente em um pipeline

context.build\_data\_docs()

context.open\_data\_docs()

### Validação de dados em tempo real

O Great Expectations pode ser usado para validar dados em tempo real, o que significa que podemos definir expectativas para novos dados à medida que são recebidos. Isso é útil quando precisamos validar dados continuamente, como em um fluxo de dados.

Exemplo:

from great\_expectations.dataset import SparkDFDataset

# criar um objeto SparkDFDataset

streaming\_data = spark.readStream.format('kafka').option('kafka.bootstrap.servers', 'localhost:9092').load()

streaming\_dataset = SparkDFDataset(streaming\_data)

# definir expectativas para a coluna de valor

expectation = {

'expectation\_type': 'expect\_column\_values\_to\_be\_between',

'kwargs': {

'column': 'valor',

'min\_value': 0,

'max\_value': 100

}

}

streaming\_dataset.add\_expectation(expectation)

# validar os novos dados à medida que chegam

while True:

results = streaming\_dataset.validate()

print(results)

Essas são apenas algumas das muitas funcionalidades que o Great Expectations oferece. Com essas funcionalidades, podemos criar expectativas personalizadas para nossos dados e garantir que eles atendam às nossas necessidades de validação em uma variedade de cenários.

# Referências

* Documentação Great Expectations: <https://docs.greatexpectations.io/docs/>
* Playlist Python no canal da Ada no Youtube: <https://www.youtube.com/watch?v=cDqMbI02hRs&list=PLcmVV8telDGzH6wFY_9h_KZ3Sn7EfjT1D>
* Livro: REIS, Joe & HOUSLEY, Matt. Fundamentals of Data Engineering, O'Reilly, 2022.
* Livro: DENSMORE, James. Data Pipelines Pocket Reference: Moving and Processing Data for Analytics, 2021
* Curso digital Ada: [Git e versionamento.](https://cursos.letscode.com.br/curso-digital/37f4b5d2-dbab-4c45-ab61-aac1ba2c7d19)