


# Pare HOJE mesmo de NÃO TESTAR o seu CÓDIGO!

Nunca testou? Sem Pânico!

Passo a Passo para começar a usar

O Pytest ajuda você a escrever  
programas melhores!



Ser pleno ou sênior  
exige testar !

# Nunca testou? Sem Pânico!

Eu sei, **esse conceito assusta.**

Mas respire fundo e confie. Os maiores avanços acontecem quando **saímos da nossa zona de conforto.**

E você não está sozinho nessa.

Os testes foram projetados para te ajudar a escrever código melhor, **não para ser um obstáculo!**

**Bora lá?**

# Você já viu esse filme?



O dev tinha suas razões enigmáticas para evitar testes. Hoje, todos pagam o preço, dedicando horas à manutenção e resolução de tickets.

**Uma verdadeira guerra infinita!**



# Um novo filme: O TDD

Nem os Vingadores foram enfrentar o Thanos sem preparo. Pq você vai dar uma de **dev herói**?

Em vez de apenas atacar sem um plano, **vamos usar o TDD, ok?**

O TDD (Test-Driven Development) é uma técnica em que os testes são escritos antes do código.

**TDD é o preparo do Batman.**



# Verde, Vermelho, Refatore!

A abordagem verde-vermelho é essencial no TDD. Comece pelos testes **(vermelho)** e só depois desenvolva o código **(verde)** para atendê-los

TDD não é apenas sobre testes. **É sobre design de software** e garantia de que seu código atenda às **expectativas do cliente desde o início.**



# Por quê Pytest?

Porque o **Pytest** é o **Pandas** dos testes.

Assim como o Pandas torna a manipulação de dados em Python mais fácil e intuitiva, o Pytest simplifica a escrita de testes.

Para instalar é igual



```
pip install pytest # poetry add pytest
```

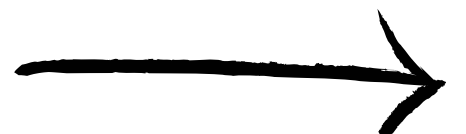


# Organização

No Pytest, é essencial uma estrutura padrão. Separe o código dos testes, e os arquivos de teste devem iniciar com o prefixo "test\_".

```
seu_projeto/
├── src/                                # Pasta principal para o seu código
│   ├── modulo1.py
│   ├── modulo2.py
│   └── ...
└── tests/                             # Pasta dedicada para testes
    ├── __init__.py                  # Não esquecer do __init__.py!
    ├── test_modulo1.py
    ├── test_modulo2.py
    └── ..
```

O primeiro comentário é o link do github do projeto, lá você pode ver em detalhe!



# Sua Task: Limpeza de Strings

Para este guia, vamos abordar um desafio comum: Você recebeu a tarefa de **criar uma função que processe strings**.

Esta função deve:

1. Converter a string para minúsculas (lowercase).
2. Remover todos os espaços no início e no fim da string.

Essa é uma operação frequentemente necessária em pré-processamento de dados, limpeza de entradas do usuário, entre outros.





# Seu planejamento: Todos os nomes

Ao adotar TDD, é fundamental começar com um bom planejamento. Seguem as etapas:

1. Nome do Módulo: Determine qual será o nome do módulo onde sua função residirá. Nosso módulo será **"data\_transform.py"**

2. Nome da Função: O nome da função deve indicar claramente o que ela faz, o nosso será **"clean\_data\_lower"**.

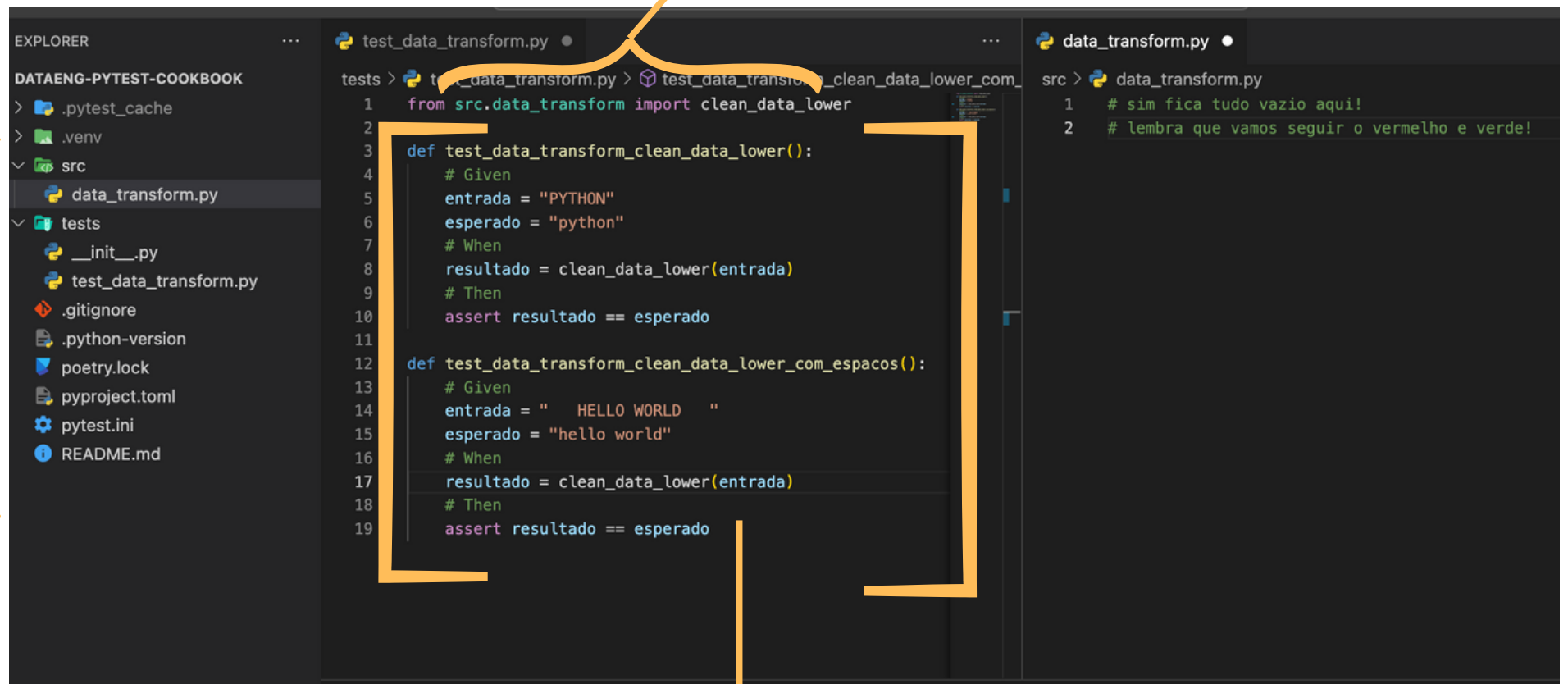
3. Definir o que irá testar. Nosso módulo **test\_data\_transform.py**, além dos nossos cenários: **"test\_clean\_data\_lower"** e **"test\_clean\_data\_lower\_com\_espacos"**



# Vamos estruturar...

Primeiro passo é criar nossos arquivos e pastas. Criei os arquivos **data\_transform.py** e o **test\_data\_transform.py**.

O segundo passo é importar o módulo e o método que eu quero testar **clean\_data\_lower**



The screenshot shows the VS Code interface with three panels. The left panel (EXPLORER) shows the project structure for 'DATAENG-PYTEST-COOKBOOK', including a 'src' folder with 'data\_transform.py' and a 'tests' folder with 'test\_data\_transform.py'. The middle panel shows the code in 'test\_data\_transform.py', which imports 'clean\_data\_lower' from 'src.data\_transform' and defines two test functions: 'test\_data\_transform\_clean\_data\_lower()' and 'test\_data\_transform\_clean\_data\_lower\_com\_espacos()'. Both functions use 'assert' to verify the output of 'clean\_data\_lower'. The right panel shows the code in 'data\_transform.py', which contains two empty functions: 'clean\_data\_lower()' and 'clean\_data\_lower\_com\_espacos()'. Orange arrows point from the text above to the relevant parts of the code: one points to the file names in the Explorer, another points to the import statement, and a third points to the 'assert' statements in the test functions.

```
EXPLORER
DATAENG-PYTEST-COOKBOOK
> .pytest_cache
> .venv
✓ src
  data_transform.py
  tests
    __init__.py
    test_data_transform.py
    .gitignore
    .python-version
    poetry.lock
    pyproject.toml
    pytest.ini
    README.md

tests > test_data_transform.py
1 from src.data_transform import clean_data_lower
2
3 def test_data_transform_clean_data_lower():
4     # Given
5     entrada = "PYTHON"
6     esperado = "python"
7     # When
8     resultado = clean_data_lower(entrada)
9     # Then
10    assert resultado == esperado
11
12 def test_data_transform_clean_data_lower_com_espacos():
13     # Given
14     entrada = "  HELLO WORLD  "
15     esperado = "hello world"
16     # When
17     resultado = clean_data_lower(entrada)
18     # Then
19     assert resultado == esperado

src > data_transform.py
1 # sim fica tudo vazio aqui!
2 # lembra que vamos seguir o vermelho e verde!
```

O terceiro passo é escrever os nossos testes dentro do arquivo "**test\_data\_transform.py**". Ele sempre terá essa receita de bolo de "entrada" , "esperado" , "resultado" .

Essa aqui é a parte mais importante!

**O assert!** Ele me fala o resultado que eu espero com a entrada que eu planejei! Vamos fazer 2 testes. Um somente a string em upper e um segundo com upper e espaços.

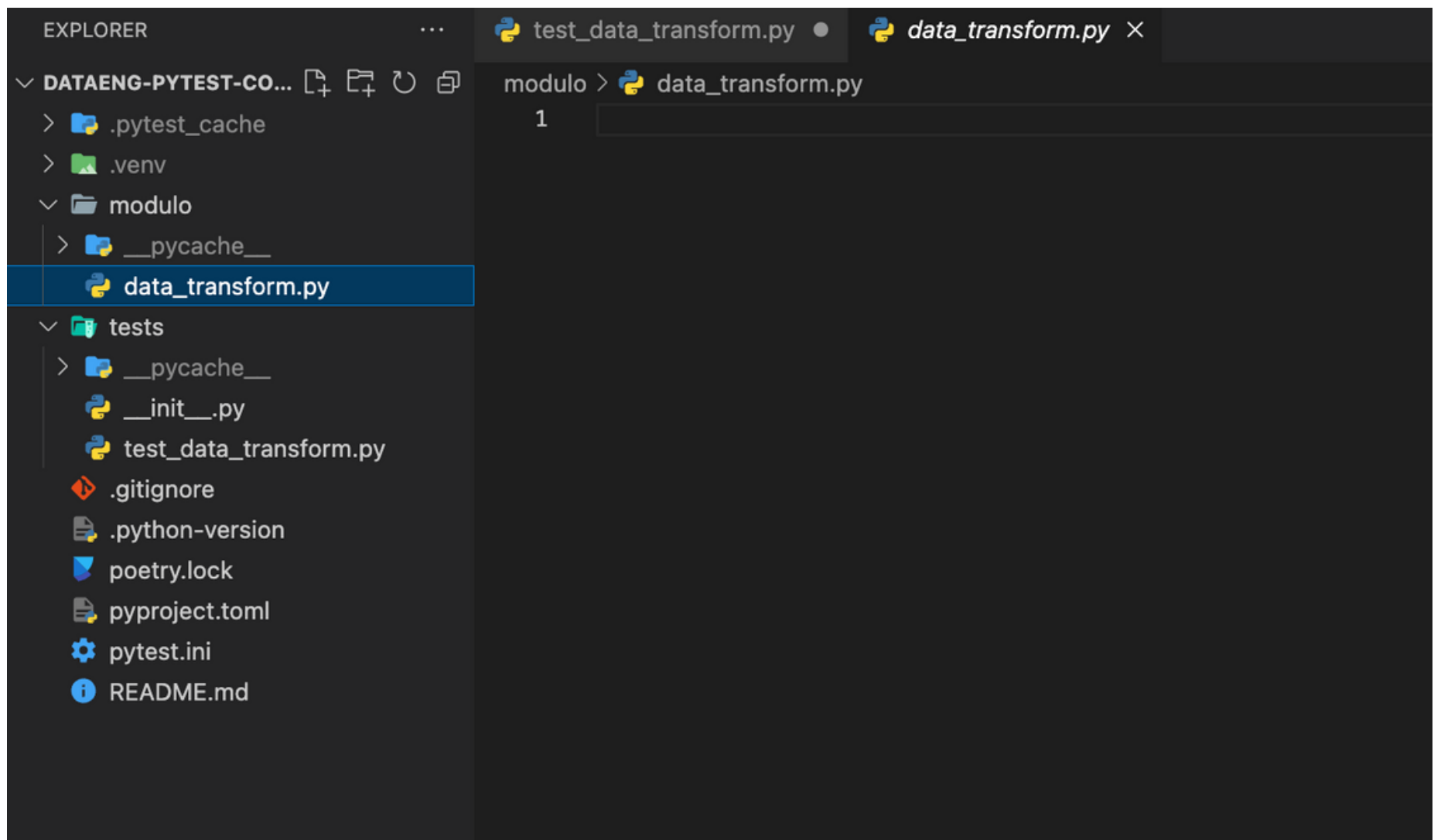
# E o código do módulo?

**Ele ta em branco!**

Vamos fazer TDD se lembra?

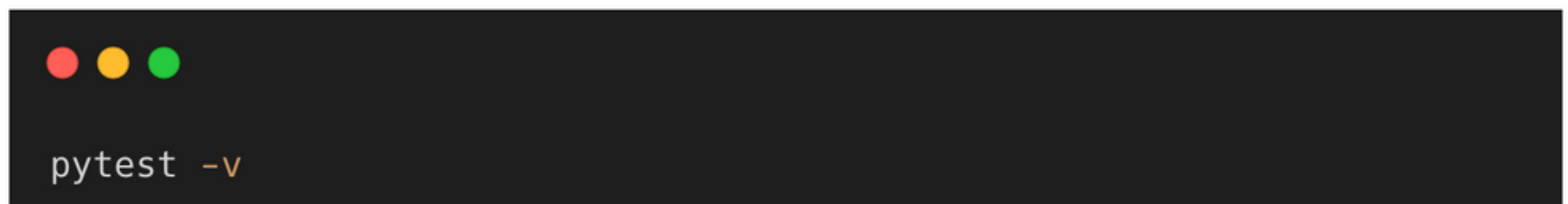
**Primeiro escrevemos os testes.**

Testamos e ai sim desenvolvemos!



# Nosso primeiro teste

Para rodar nossos testes precisamos escrever somente `pytest -v`

A terminal window with a dark background. In the top-left corner, there are three colored circles: red, yellow, and green. Below them, the text 'pytest -v' is displayed in a light gray font.

```
pytest -v
```

(esse `-v` é para falar que queremos ele verboso, dando mais detalhes)

Viu como é fácil?

E agora o que será que vai dar bom?



# ImportError



Caraca, Quanto vermelho!  
Quebrou tudo! Mas calma!

```
TERMINAL
> ✓ TERMINAL
(dataeng-pytest-cookbook-py3.11) → dataeng-pytest-cookbook git:(main) x pytest
===== test session starts =====
platform darwin -- Python 3.11.3, pytest-7.4.0, pluggy-1.2.0
rootdir: /Users/lucianogalvao/github/dataeng-pytest-cookbook
configfile: pytest.ini
plugins: anyio-3.7.1
collected 0 items / 1 error

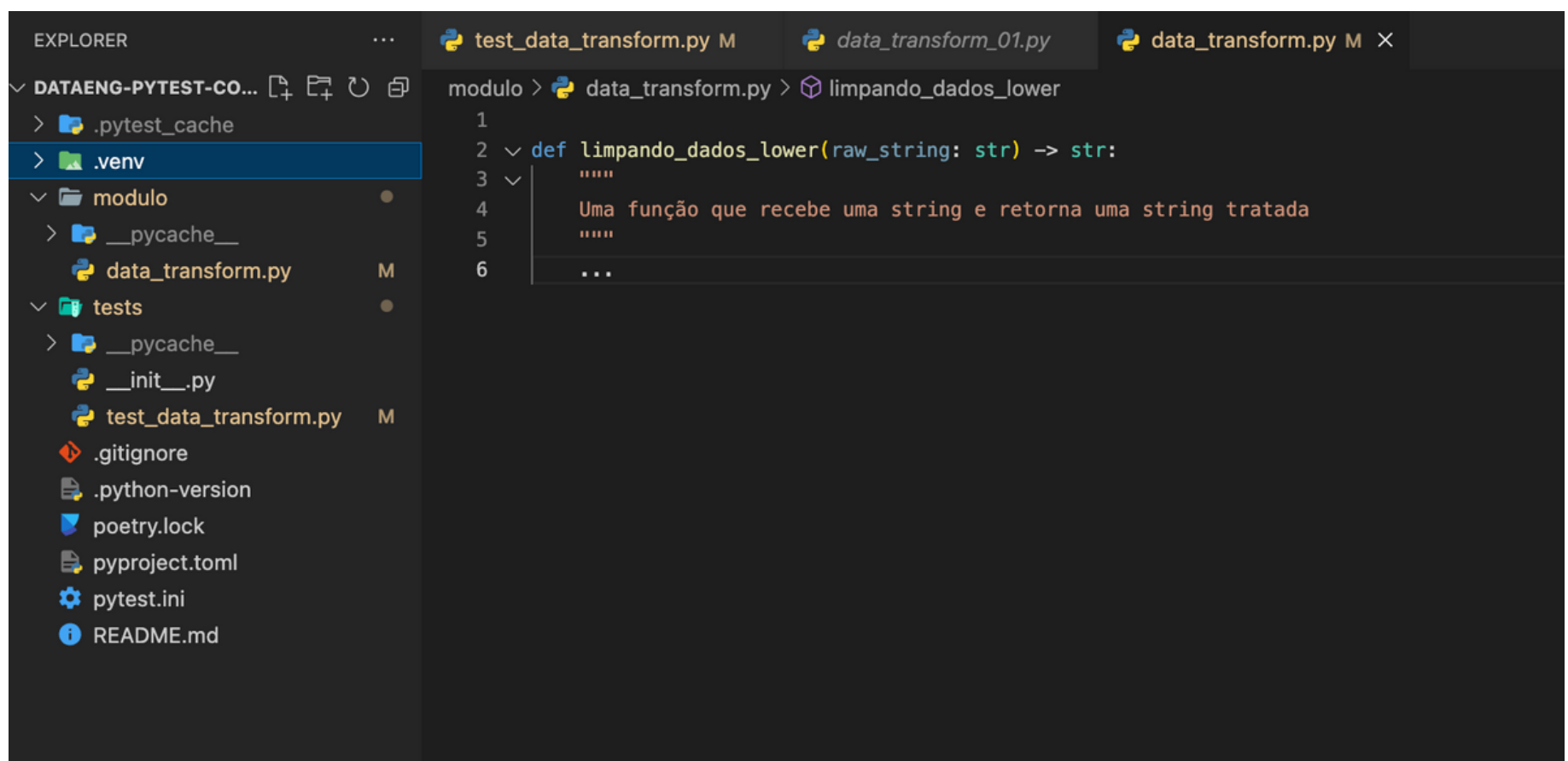
===== ERRORS =====
ERROR collecting tests/test_data_transform.py
ImportError while importing test module '/Users/lucianogalvao/github/dataeng-pytest-cookbook/tests/test_data_transform.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
../.pyenv/versions/3.11.3/lib/python3.11/importlib/__init__.py:126: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
tests/test_data_transform.py:1: in <module>
    from modulo.data_transform import limpando_dados_lower
E   ImportError: cannot import name 'limpando_dados_lower' from 'modulo.data_transform' (/Users/lucianogalvao/github/dataeng-pytest-cookbook/modulo/data_transform.py)
===== short test summary info =====
ERROR tests/test_data_transform.py
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
===== 1 error in 0.07s =====
(dataeng-pytest-cookbook-py3.11) → dataeng-pytest-cookbook git:(main) x
```

Vamos ler a mensagem de erro! É um erro de **ImportError**, ou seja, ele tentou importar a função, e como não existe, deu erro! O que vamos fazer agora?



# Criar a função

Vamos abrir nosso módulo e criar a função. Só isso!



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane displays the project structure for 'DATAENG-PYTEST-CO...'. The 'modulo' folder is expanded, showing 'data\_transform.py' (marked with an 'M' for modified). The main editor pane shows the 'data\_transform.py' file with the following code:

```
modulo > data_transform.py > limpando_dados_lower
1
2 def limpando_dados_lower(raw_string: str) -> str:
3     """
4     Uma função que recebe uma string e retorna uma string tratada
5     """
6     ...
```

Vamos escrever todo o código dela?  
Não, vamos testar novamente!



A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The command 'pytest -v' is entered in the terminal.





# Assertion Error



```

✓ TERMINAL
~/github/dataeng-pytest-cookbook/modulo/
data_transform.py · Modified  dataeng-pytest-cookbook git:(main) x pytest -v
===== test session starts =====
platform darwin -- Python 3.11.3, pytest-7.4.0, pluggy-1.2.0 -- /Users/lucianogalvao/.pyenv/versions/3.11.3/bin/python3.11
cachedir: .pytest_cache
rootdir: /Users/lucianogalvao/github/dataeng-pytest-cookbook
configfile: pytest.ini
plugins: anyio-3.7.1
collected 1 item

tests/test_data_transform.py::test_data_transform_clean_data_lower FAILED

===== FAILURES =====
_____ test_data_transform_clean_data_lower _____

def test_data_transform_clean_data_lower():
    # Given
    entrada = "PYTHON"
    esperado = "python"

    # When
    resultado = limpando_dados_lower(entrada)

    # Then
    > assert resultado == esperado
E   AssertionError: assert None == 'python'

tests/test_data_transform.py:19: AssertionError
===== short test summary info =====
FAILED tests/test_data_transform.py::test_data_transform_clean_data_lower - AssertionError: assert None == 'python'
===== 1 failed in 0.07s =====
(dataeng-pytest-cookbook-py3.11) → dataeng-pytest-cookbook git:(main) x
```

Já melhorou! Ta menos vermelho, kkk  
Agora Já é outro tipo de erro.

O **Assertion Error** mostra que é um erro do resultado do teste. Ou seja, já fizemos toda a configuração. **O erro foi somente do resultado.**



# Evoluir a função



Vamos abrir nosso módulo e melhorar nossa função!

The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' sidebar displays the project structure for 'DATAENG-PYTEST-COOKBOOK'. It includes folders like '.pytest\_cache', '.venv', and 'src', along with a 'tests' folder. The 'src' folder is expanded, showing 'data\_transform.py' (marked with a green dot and 'M') and a 'tests' folder containing '\_\_init\_\_.py', 'test\_data\_transform.py' (marked with a green dot and 'M'), and other files like '.gitignore', '.python-version', 'poetry.lock', 'pyproject.toml', 'pytest.ini', and 'README.md'. The main editor area shows the code for 'data\_transform.py'. The function 'clean\_data\_lower' is defined, taking 'raw\_string: str' as an argument and returning a 'str'. The function body includes a docstring, a call to 'raw\_string.lower()', and a return statement. The code is as follows:

```
src > data_transform.py > clean_data_lower
1 def clean_data_lower(raw_string: str) -> str:
2     """ uma função que recebe uma raw_string e retorna uma tratada """
3     result = raw_string.lower()
4     return result
```

E agora?  
Testar novamente!

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The command 'pytest -v' is entered in the terminal.

```
pytest -v
```





**E ai?**



# 50%



```
(dataeng-pytest-cookbook-py3.11) → dataeng-pytest-cookbook git:(main) x pytest -v
===== test session starts =====
platform darwin -- Python 3.11.3, pytest-7.4.0, pluggy-1.2.0 -- /Users/lucianogalvao/.pyenv/versions/3.11.3/bin/python3.11
cachedir: .pytest_cache
rootdir: /Users/lucianogalvao/github/dataeng-pytest-cookbook
configfile: pytest.ini
plugins: anyio-3.7.1
collected 2 items

tests/test_data_transform.py::test_data_transform_clean_data_lower PASSED [ 50%]
tests/test_data_transform.py::test_data_transform_clean_data_lower_com_espacos FAILED [100%]

===== FAILURES =====
test_data_transform_clean_data_lower_com_espacos

def test_data_transform_clean_data_lower_com_espacos():
    # Given
    entrada = "  HELLO WORLD  "
    esperado = "hello world"
    # When
    resultado = clean_data_lower(entrada)
    # Then
    > assert resultado == esperado
E   AssertionError: assert '  hello world  ' == 'hello world'
E       - hello world
E       +   hello world
E       ? +++          +++

tests/test_data_transform.py:19: AssertionError

===== short test summary info =====
FAILED tests/test_data_transform.py::test_data_transform_clean_data_lower_com_espacos - AssertionError: assert '  hello world  ' == 'hello w
orld'

===== 1 failed, 1 passed in 0.07s =====
(dataeng-pytest-cookbook-py3.11) → dataeng-pytest-cookbook git:(main) x
```

Na trave!

O primeiro teste foi **PASSED**, já o segundo **FAILED**. Olha que ele mostra em detalhe:

" hello world " !=! "hello world".

Cara, é lindo o retorno do pytest. Olha com calma todas as linhas.



# Evoluir a função



Sinal vermelho! Vamos continuar melhorando a função. Vamos colocar agora o método `strip()` que deve da conta do recado.

```
EXPLORER
DATAENG-PYTEST-COOKBOOK
> .pytest_cache
> .venv
src
  > __pycache__
  data_transform.py M
tests
  > __pycache__
  __init__.py
  test_data_transform.py M
  .gitignore
  .python-version
  poetry.lock
  pyproject.toml
  pytest.ini
  README.md

src > data_transform.py > clean_data_lower
1 def clean_data_lower(raw_string: str) -> str:
2     """ uma função que recebe uma raw_string e retorna uma tratada"""
3     result = raw_string.strip().lower()
4     return result
```

E agora?

Testar novamente!

```
pytest -v
```



**E ai?**



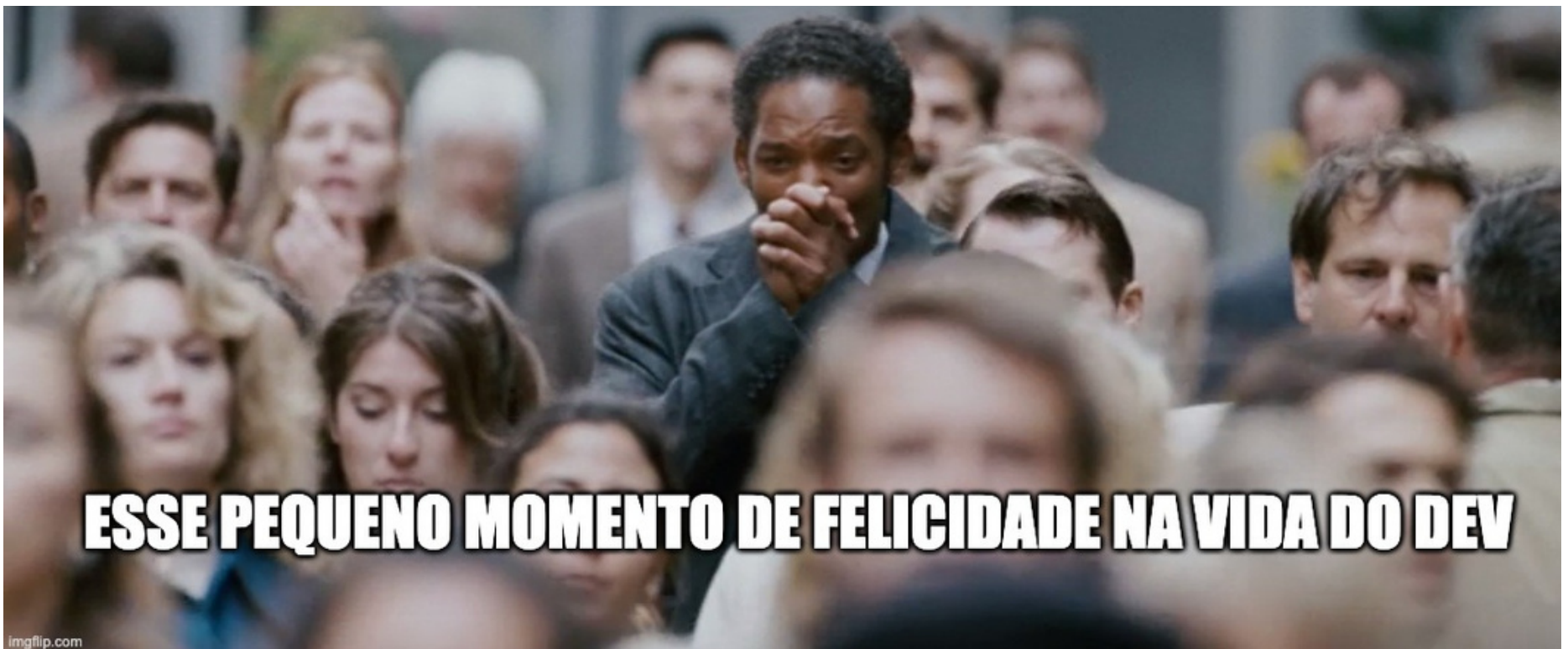
# Vitória!



```
(dataeng-pytest-cookbook-py3.11) → dataeng-pytest-cookbook git:(main) x pytest -v
===== test session starts =====
platform darwin -- Python 3.11.3, pytest-7.4.0, pluggy-1.2.0 -- /Users/lucianogalvao/.pyenv/versions/3.11.3/bin/python3.11
cachedir: .pytest_cache
rootdir: /Users/lucianogalvao/github/dataeng-pytest-cookbook
configfile: pytest.ini
plugins: anyio-3.7.1
collected 2 items

tests/test_data_transform.py::test_data_transform_clean_data_lower PASSED [ 50%]
tests/test_data_transform.py::test_data_transform_clean_data_lower_com_espacos PASSED [100%]

===== 2 passed in 0.01s =====
```



Os 2 testes deram **PASSED!**

**Conseguimos desenvolver o código o  
necessário para passar nos testes!  
Fizemos nosso primeiro TDD!**



# E Agora?

# Os Testes Passaram!

Após os testes passarem, continue:

1. **Feedback:** Comunique-se com a equipe, solicite opiniões e crie mais testes.
2. **Refatoração:** Aprimore o código mantendo a funcionalidade. O TDD sinaliza erros rapidamente.
3. **Documentação:** Atualize conforme alterações.
4. **Integração Contínua:** Use um sistema CI/CD.
5. **Novas features:** Crie testes e adicione funcionalidades.



# A Segurança dos Testes em Dados

Os dados são o coração de muitos negócios modernos. Proteja-os com **uma estratégia sólida de testes!**

A implementação de testes desde o início do desenvolvimento - em um approach de TDD - pode reduzir significativamente a necessidade de retrabalho e assegurar que as transformações de dados sejam consistentes e confiáveis!

**Paz de Espírito: Durma tranquilo sabendo que tudo está bem!**



# Obrigado!

Se essa postagem te ajudou, curte e comente! **Sua interação pode ajudar outras pessoas** a se beneficiarem desse conhecimento.

Siga-me para receber dicas, insights e tutoriais de problemas que já passei muita noite virado para resolver!

(e espero que você não passe!)

