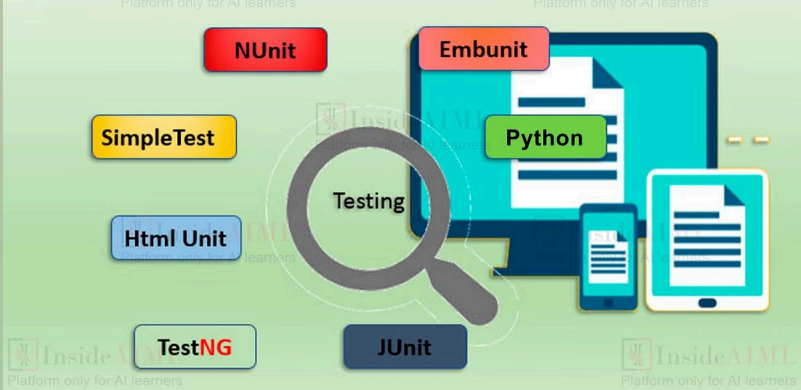


Unit Testing in Python



Pytest: Biblioteca de testes em Python

O Pytest é uma poderosa biblioteca de testes para a linguagem Python, conhecida por sua simplicidade e flexibilidade. Ele é amplamente utilizado para a escrita de testes unitários e de integração, graças à sua facilidade de configuração, organização e execução dos testes.

Introdução ao Pytest: simplicidade e flexibilidade

1

Sintaxe Simples

Escrever testes com código limpo e intuitivo, sem a necessidade de classes ou métodos complexos.

2

Asserções Nativas

Utiliza as asserções do próprio Python, tornando o código dos testes mais direto e legível.

3

Fixtures Poderosas

Recursos configuráveis que ajudam a preparar o ambiente ideal para os testes.

Configurando e organizando os testes com Pytest

Estrutura de Diretórios

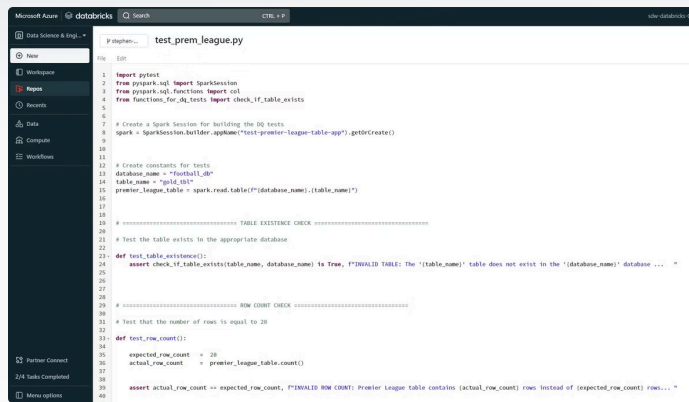
O Pytest segue uma convenção de estruturação de diretórios que facilita a organização dos testes.

Configuração com Fixtures

As fixtures permitem configurar e limpar o ambiente de teste antes e depois da execução.

Plugins Úteis

Pytest oferece uma ampla gama de plugins para estender suas funcionalidades.



```
1 import pytest
2 from pytest import SparkSession
3 from pytest import function, report_col
4 from functions_for_testing import check_if_table_exists
5
6
7 # Create a Spark Session for building the DB tests
8 spark = SparkSession.builder.appName("test-premier-league-table-app").getOrCreate()
9
10
11 # Create constants for tests
12 database_name = "premier_league"
13 table_name = "premier_league"
14
15 # Create the table
16 premier_league_table = spark.read.table(f"{database_name}.{table_name}")
17
18
19 # ===== TABLE EXISTENCE CHECK =====
20
21 # Test the table exists in the appropriate database
22
23 def test_table_exists():
24     assert check_if_table_exists(table_name, database_name) is True, f"INVALID TABLE: The '{table_name}' table does not exist in the '{database_name}' database ..."
25
26
27 # ===== ROW COUNT CHECK =====
28
29 # Test that the number of rows is equal to 20
30
31 def test_row_count():
32     expected_row_count = 20
33     actual_row_count = premier_league_table.count()
34
35     assert actual_row_count == expected_row_count, f"INVALID ROW COUNT: Premier League table contains {actual_row_count} rows instead of {expected_row_count} rows..."
```

Escrevendo testes unitários e de integração

Testes Unitários

Foco em testar pequenas unidades de código isoladamente.

Parametrização

Pytest permite executar o mesmo teste com múltiplas entradas, garantindo cobertura em diferentes cenários.

1

2

3

Testes de Integração

Verificam o funcionamento do sistema como um todo, incluindo interações entre módulos.

Benefícios do Pytest

Pytest Code

```
import pytest

def test_first_assertion():
    assert 1 == 1
```

unittest code

```
import unittest

class FirstTest(unittest.TestCase):

    def test_first_assertion(self):
        self.assertEqual(1,1)
```

Sintaxe Simples

Permite escrever testes com código limpo e intuitivo.

Asserções Nativas

Utiliza as asserções do próprio Python, tornando o código mais legível.

Fixtures Poderosas

Recursos configuráveis que ajudam a preparar o ambiente ideal para os testes.

Extensível com Plugins

Pytest possui uma grande coleção de plugins que expandem suas funcionalidades.

Conclusão: por que adotar o Pytest?



Simplicidade

Pytest é fácil de aprender e usar, com uma sintaxe intuitiva.



Flexibilidade

Pytest é poderoso o suficiente para lidar com cenários de teste complexos.



Comunidade Ativa

Pytest possui uma grande comunidade que contribui com plugins e soluções.

Pytest Framework



TURING