Workshop: Estruturando um Projeto de Dados Do Zero

eBook: Guia Passo a Passo para criar Produtos Incríveis e com Confiança!

Esse eBook faz parte do material de apoio do Workshop

Objetivo do Ebook

Projetos de dados podem ser assustadores no início, com **diversos requisitos e funcionalidades**. Entre instalação e gerenciamento de versões do Python, administração de pacotes e bibliotecas externas, testes, documentação, segurança, boas práticas e integração contínua... ufa! **Há muito a ser contemplado.**

E é exatamente por isso que criei este eBook: **para oferecer um guia prático** a ser consultado sempre que estiver construindo ou otimizando um projeto de dados.

12 Fatores

Para guiar nesse processo de estruturação de projeto de dados, **separei em 12 fatores essenciais** que visam aprimorar e solidificar a estrutura de projeto de dados.

É fundamental frisar que a minha intenção não é induzir você a implementar todos os 12 fatores de uma só vez ou gerar uma pressão desnecessária sobre o processo.

Em vez disso, o foco aqui é **cultivar o senso crítico** durante a implementação e apresentar possibilidades.

12 Fatores - Visão Geral

1) Pyenv
2) Poetry
3) Git & Github
4) Estrutura de pastas
5) Arquivos comuns
6) Pytest
7) Padrão de Código
8) Mkdocs
9) Taskipy
10) Pre-commit
11) Github Actions (CI/CD)
12) Readme

1) Pyenv

O que faz e o que é?

Pyenv é uma ferramenta para **gerenciar versões do Python**. Ele permite que você altere facilmente a versão global do Python utilizada em seu sistema, bem como a versão específica de cada projeto que for trabalhar.

Isso é útil para **garantir a compatibilidade com vários projetos** que possam exigir versões diferentes do Python, além de formalizar a versão utilizada para outros colaboradores.

Documentação

1) Pyenv

O que faz e o que é?

Pyenv é uma ferramenta para **gerenciar versões do Python**. Ele permite que você altere facilmente a versão global do Python utilizada em seu sistema, bem como a versão específica de cada projeto que for trabalhar.

Isso é útil para **garantir a compatibilidade com vários projetos** que possam exigir versões diferentes do Python, além de formalizar a versão utilizada para outros colaboradores.

Documentação

1) Pyenv

Principais comandos

Instalação de Versões do Python

- pyenv install --list: Lista todas as versões disponíveis para instalação.
- pyenv install [versão]: Instala uma versão específica do Python.

Gerenciamento de Versões

- pyenv versions: Mostra todas as versões do Python instaladas.
- pyenv global [versão]: Define a versão do Python a ser usada globalmente no sistema.
- pyenv local [versão]: Define a versão do Python a ser usada localmente em um diretório específico.

2) Poetry

O que faz e o que é?

O Poetry é uma ferramenta para gerenciamento de dependências e empacotamento em Python. Ele permite que você declare as bibliotecas das quais seu projeto depende e as gerenciará (instalará/atualizará) para você.

Além disso, ele segue a PEP 517/518, descrevendo todo o metadado do projeto através do arquivo pyproject.toml

Documentação

2) Poetry

Principais comandos

Inicialização e Configuração

- poetry init: Inicia um novo projeto ou configura um existente através de um assistente para criar um pyproject.toml.

Gerenciamento de Dependências

- poetry add [pacote]: Adiciona um novo pacote como dependência e o instala.
- poetry remove [pacote]: Remove uma dependência do projeto.
- poetry show: Mostra as dependências instaladas.

Ambiente Virtual

- poetry shell: Cria e loga em um shell dentro do ambiente virtual.
- poetry env use [versão_python]: Define a versão do Python a ser usada pelo ambiente virtual do projeto

3) Git e Github

O que faz e o que é?

O Git é uma ferramenta essencial para o **controle de versão** e colaboração em projetos de desenvolvimento de software.

GitHub é uma **plataforma de hospedagem de código-font**e e um serviço de versionamento usando o Git. Ele proporciona a desenvolvedores e equipes de software um espaço remoto.

Documentação

3) Git e Github

Principais comandos

Configuração Inicial

- **git config --global user.name "[nome]":** Define o nome que será anexado aos commits realizados.
- **git config --global user.email "[endereco-de-email]":** Define o email que será anexado aos commits realizados.

Criando ou Clonando Repositórios

- **git init:** Inicializa um novo repositório Git.
- **git clone [url]:** Clona (faz uma cópia) de um repositório.

Trabalhando com Branches (Ramos)

- **git branch:** Lista todas as branches (ramos) no repositório.
- **git branch [nome_da_branch]:** Cria uma nova branch.

3) Git e Github

Principais comandos

Trabalhando com Branches (Ramos)

- **git checkout [nome_da_branch]**: Muda para a branch especificada.
- **git merge [nome_da_branch]**: Mescla as alterações da branch especificada na branch atual.

Manipulação de Arquivos e Commits

- **git status**: Mostra o status do diretório de trabalho.
- **git add [arquivo]**: Adiciona as alterações no arquivo especificado para a área de staging.
- **git commit -m "[mensagem de commit]"**: Realiza um commit com a mensagem especificada.
- **git rm [arquivo]:** Remove o arquivo do diretório de trabalho e da área de staging.

3) Git e Github

Principais comandos

Histórico e Diferenças

- **git log:** Exibe o histórico de commits.
- **git log --oneline:** Exibe o histórico de commits de forma simplificada.
- **git diff:** Mostra as diferenças não adicionadas na área de staging.
- **git diff --staged:** Mostra as diferenças entre os arquivos na área de staging e os commits mais recentes.

Trabalhando com Remotos

- **git remote add [apelido] [url]:** Adiciona um repositório remoto com um apelido.
- **git push:** Envia os commits para o repositório remoto na branch especificada.
- **git pull:** Busca as alterações do repositório remoto e as mescla na branch local.

3) Git e Github

Principais comandos

Desfazendo Coisas

- **git reset [arquivo]:** Remove o arquivo da área de staging, mas mantém o conteúdo.
- **git checkout -- [arquivo]:** Descarta as alterações no diretório de trabalho.
- **git revert [hash_commit]:** Cria um novo commit que desfaz as alterações do commit especificado.
- **git commit --amend -m "Nova mensagem de commit"**: Este comando permite que você modifique a mensagem do seu último commit com uma nova mensagem.

4) Estrutura de pastas

O que faz e o que são?

1. SRC/APP/[Nome do Pacóte]
Propósito: Contém o código fonte do projeto.
2. TESTS
Propósito: Armazena os testes automatizados para verificar a corretude do código em SRC.
3. DOCS
Propósito: Contém a documentação do projeto, possivelmente escrita usando ferramentas como Sphinx.
4. SCRIPTS
Propósito: Pode conter scripts úteis para automação de tarefas como instalação, compilação, execução de testes, entre outros.

5) Arquivos Comuns

O que faz e o que são?

.gitignore
Propósito: Lista de arquivos e diretórios que o Git deve ignorar ao fazer commit.
README.md
Propósito: Um arquivo markdown que fornece informações básicas sobre o projeto, como sua descrição, modo de uso e instruções de instalação.
.pre-commit-config.yaml
Propósito: Configuração para a ferramenta pre-commit, que ajuda a gerenciar e manter as pre-commits hooks.
.python-version
Propósito: Especifica a versão do Python utilizada no projeto, frequentemente usado em conjunto com gerenciadores de versão como pyenv.

5) Arquivos Comuns

O que faz e o que são?

.github/workflows
Propósito: Contém os scripts de workflow para GitHub Actions, permitindo a automação de pipelines CI/CD diretamente no GitHub.
pyproject.toml
Propósito: Um arquivo de configuração para ferramentas de construção de projetos Python, como Poetry ou PEP 517/518, que contém metadados do projeto e dependências.
requirements.txt
Propósito: Uma lista de dependências do projeto para instalação com pip, embora o pyproject.toml esteja se tornando mais comum.

6) Testes com Pytest

Tipos de Testes (uma Visão Geral)

1. Testes Unitários
Testam uma "unidade" do código isoladamente (como funções ou métodos) para garantir que esteja funcionando conforme esperado.
2. Testes de Integração
Testam a integração entre diferentes partes do código (por exemplo, o módulos de extract verifica se consegue acessar a AWS). Tem como objetivo detectar possíveis falhas na interação entre diferentes módulos.
3. Testes Funcionais
Confirmar se todo o projeto está funcionando como deveria no cenário completo.

Documentação

6) Testes com Pytest

Principais comandos

pytest

Executa todos os testes no diretório atual e subdiretórios.

pytest [nome_do_arquivo]

Executa todos os testes contidos no arquivo especificado.

pytest -v

Habilita a saída verbosa, oferecendo mais detalhes sobre os testes que estão sendo executados.

pytest --html=relatorio.html

Gera um relatório de teste em um arquivo HTML, necessita do plugin pytest-html.

pytest --cov=[diretório]

Gera um relatório de cobertura de código quando usado com o plugin pytest-cov.

7) Padrões de Código

Padrão uma visão Geral

A manutenção dos padrões de código em Python é fundamental para garantir a legibilidade, consistência e, em alguns casos, a segurança do código.

Aqui estão 12 bibliotecas e ferramentas que podem ajudá-lo a manter os padrões de código, aderindo às PEPs (Python Enhancement Proposals), verificando tipografia, docstrings e questões de segurança:

7) Padrões de Código

Principais bibliotecas

1. flake8: Linting
Descrição: Checa o código contra a PEP8 e pode ser ampliado com plugins para verificar contra mais convenções.
2. black: Formatação de Código
Descrição: Formata o código de forma consistente e adere à "The Black Code" style.
3. mypy: Checagem de Tipos
Descrição: Verifica tipagens estáticas em Python para detectar erros de tipo antes da execução do código.
4. pylint: Linting
Descrição: Ferramenta de lint que verifica a aderência ao guia de estilo e pode ser personalizado para atender às suas necessidades.

7) Padrões de Código

Principais bibliotecas

5. pydocstyle: Verificação de Docstrings
Descrição: Checa a conformidade das docstrings com a convenção PEP257.
6. bandit: Segurança
Descrição: Procura por problemas de segurança comuns através da análise estática do código.
7. isort: Organização de Imports
Descrição: Organiza as importações de maneira ordenada e agrupada.
8. blue: Formatação de Código
Descrição: Um formatador de código Python que reformatará o código para se conformar ao PEP 8, com algumas divergências.

7) Padrões de Código

Principais bibliotecas

9. pip-audit: Segurança
Uma ferramenta para verificar dependências instaladas para buscar e corrigir vulnerabilidades conhecidas
10. Prospector: Análise Estática de Código
Descrição: Uma ferramenta que agrupa várias outras (incluindo flake8 e mypy) para realizar uma análise estática do código e oferecer sugestões de melhorias.
11. SonarQube: Análise e Qualidade
O SonarQube é uma plataforma robusta utilizada para a inspeção contínua da qualidade do código para detectar bugs.
12. Radon: Análise de Complexidade
Radon é uma ferramenta de Python para obter várias métricas de código, incluindo a complexidade e um rank de manutenibilidade do código.

8) Taskipy

Padrão uma visão Geral

Taskipy é uma ferramenta de execução de tarefas para Python, permitindo a automatização de comandos e scripts através de um único arquivo de configuração: pyproject.toml.

Ele proporciona uma maneira simples e elegante de organizar e gerenciar tarefas de projeto, tais como construção, teste e deploy, de maneira fácil e repetível.

[tool.taskipy.tasks]
test = "pytest -v"
lint = "flake8 src/ tests/"
format = "blue . && isort ."

Documentação

9) Mkdocs

Padrão uma visão Geral

Definição: MkDocs é uma ferramenta rápida e simples para criar websites de documentação a partir de arquivos escritos em Markdown.

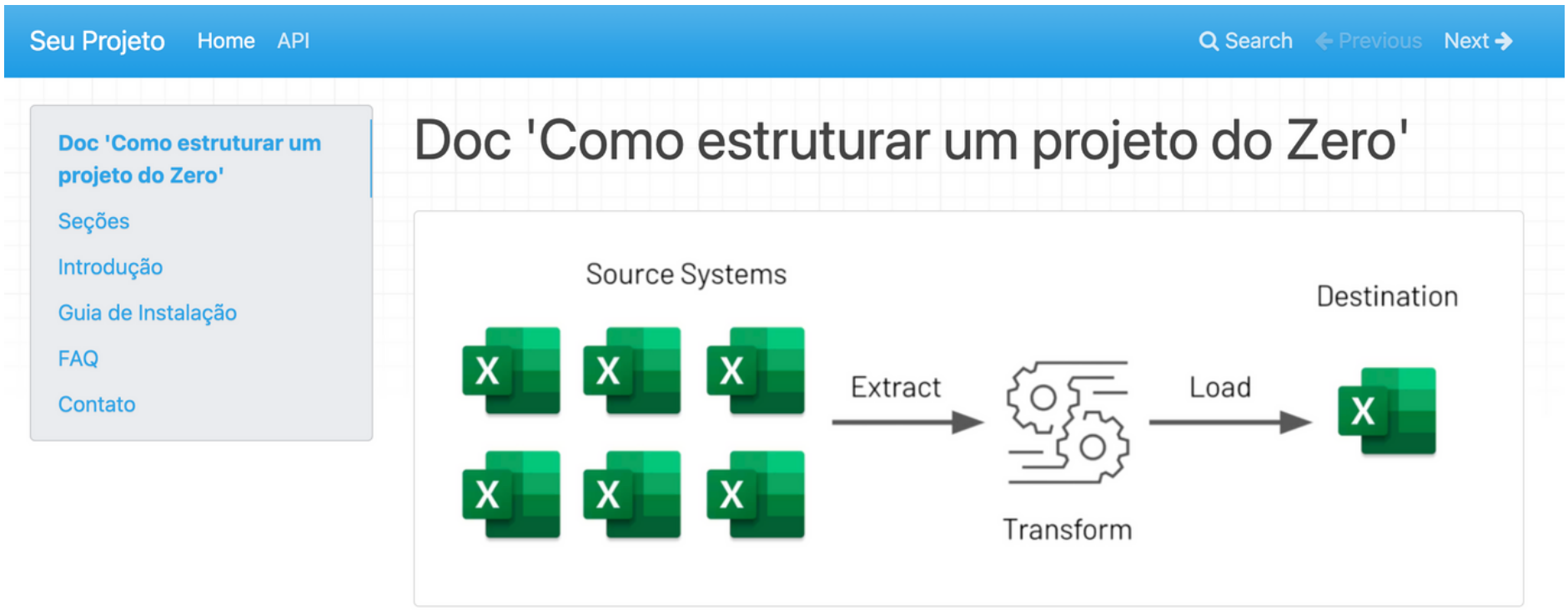
Facilitando a Documentação: Oferece uma maneira amigável e prática de documentar seu projeto, usando uma linguagem de marcação familiar e fácil de usar.

Estendendo as Possibilidades: Através de plugins e temas, o MkDocs permite uma personalização e funcionalidade ampla para atender diversos requisitos de documentação

Documentação

9) Mkdocs

Exemplo de documentação



9) Mkdocs

Primeiros passos

Instalação: Simples e direta, utilizando pip: pip install mkdocs.

Criação do Projeto: Com um simples comando `mkdocs new .`, é possível criar a estrutura básica de um projeto de documentação.

Personalização e Publicação: Modifique o arquivo mkdocs.yml para ajustar configurações e, depois, é só usar mkdocs serve para visualizar localmente ou mkdocs gh-deploy para publicar no GitHub Pages.

9) Mkdocs

Pluggins

No workshop, usamos uma série de recursos extras. Para ter todos em seu projeto instale todos esses aqui:

```
mkdocs = "^1.5.3"
mkdocstrings-python = "^1.7.2"
pygments = "^2.16.1"
mkdocs-material = "^9.4.4"
pymdown-extensions = "^10.3"
mkdocs-bootstrap386 = "^0.0.2"
```

10) Pré-Commit

Padrão uma visão Geral

Definição: O Pre-commit é uma estrutura de hooks (ganchos) de Git que **ajuda a manter a qualidade do código**, garantindo que revisões automatizadas, como formatação de código e checagens de lint, sejam executadas antes de cada commit.

Objetivo: Impedir que código que não atende a padrões específicos seja commitado.

Facilita a Vida do Desenvolvedor: Automatizando verificações e correções, assegurando que apenas código limpo e aderente às diretrizes seja submetido.
Você usa seu tempo com que gera valor, e não com formatação e coisas assim.

10) Pré-Commit

Padrão uma visão Geral

Instalação: Simples e rápida, utilizando poetry: poetry add pre-commit.

Configuração: Defina suas verificações e correções automáticas no arquivo **.pre-commit-config.yaml** em seu repositório.

Uso Prático: Após a configuração, o pre-commit verifica e/ou corrige automaticamente os arquivos a serem commitados, conforme as regras definidas.

Implementar o Pre-commit é **um processo bastante participativo.** O ideal é que todo o time entre em um acordo sobre quais serão os padrões do projeto.

11) Github Actions

Padrão uma visão Geral

GitHub Actions é uma **plataforma de CI/CD** (Integração Contínua/Entrega Contínua) que permite automatizar, personalizar e executar seus fluxos de trabalho de software diretamente no GitHub.

Ele tem como objetivo ser um p**rotetor do seu código de produção**. Será um QA 24/7 que realiza todas as validações necessárias.

Automatize Tudo: Desde testes, construções e implantações até triagem de issues e muito mais.

Em Ação: Os workflows são construídos através de código e gerenciados diretamente nos seus repositórios GitHub.

11) Github Actions

Workflows no GitHub Actions

Workflow Files: São criados utilizando a linguagem YAML e armazenados no diretório .github/workflows do seu repositório.

Eventos Disparadores: Workflows podem ser desencadeados por uma variedade de eventos, como push, pull requests, ou até mesmo em horários programados.

No workshop realizamos somente a validação de pull requests, mas tem muito mais o que explorar.

Ações: Cada workflow pode conter uma ou mais ações, que são tarefas individuais que podem ser executadas.

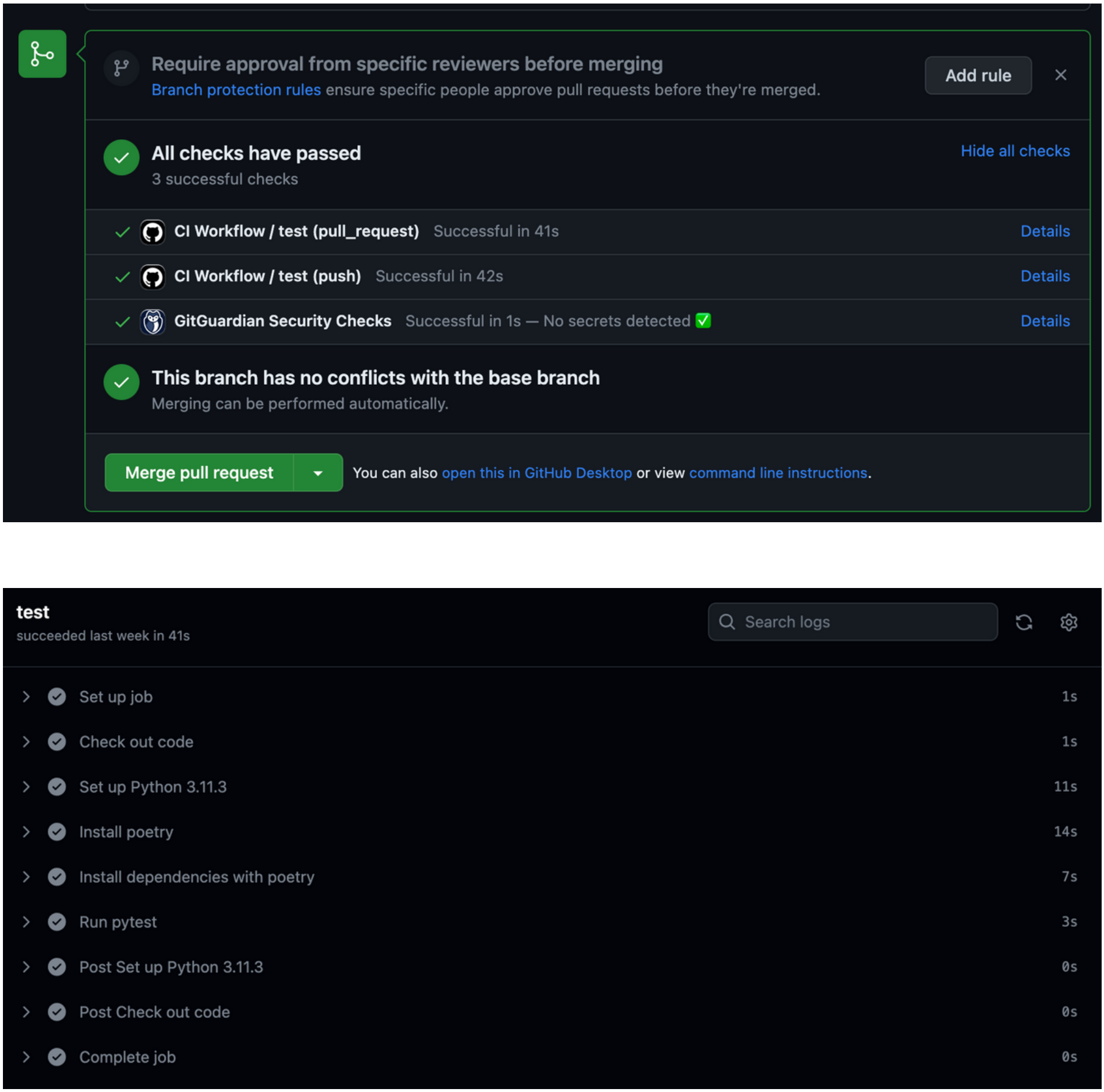
11) Github Actions

Exemplo de um CI com Poetry

```yaml
name: CI Workflow

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Check out code
        uses: actions/checkout@v2

      - name: Set up Python 3.11.3
        uses: actions/setup-python@v2
        with:
          python-version: 3.11.3

      - name: Install poetry
        run: |
          curl -sSL https://install.python-poetry.org | python3 -

      - name: Install dependencies with poetry
        run: poetry install

      - name: Run pytest
        run: poetry run pytest
```

11) Github Actions

Visão do CI rodando



12) Readme

Padrão uma visão Geral

Chegamos no último fator! **É o nosso último, mas é o primeiro arquivo que as outras pessoas tem acesso. Ou seja,** é um arquivo essencial, ele oferece uma visão inicial e geral sobre o projeto, além de fornecer as instruções para outros desenvolvedores executarem o seu projeto.

Ele é escrito em Markdown (.md), uma linguagem de marcação que é convertida facilmente para HTML.

Nos próximos slides apresento os 6 principais estilos de escrita do README

12) Readme

Principais comandos

1. Headers

Os cabeçalhos são criados usando o caractere #, seguido de um espaço. O número de # determina o nível.

```
# Cabeçalho 1
## Cabeçalho 2
### Cabeçalho 3
```

2. Ênfase

Você pode fazer texto em itálico ou negrito

```
*itálico* ou _itálico_
**negrito** ou __negrito__
```

12) Readme

Principais comandos

3. Listas

Listas podem ser ordenadas ou não ordenadas.

- Item 1
- Item 2

1. Item Ordenado 1
2. Item Ordenado 2

4. Links

Links podem ser criados da seguinte forma:
[Texto do Link](URL)

5. Imagens

A sintaxe é quase idêntica à de links.
![Texto Alternativo](URL)

12) Readme

Principais comandos

6. Blocos de Código

Você pode criar blocos de código com três crases (```)

```python
Código aqui
```

7. Listas de Tarefas

Você pode criar listas usando colchetes.

- [x] Tarefa 1
- [ ] Tarefa 2

12) Readme

O que não pode faltar

1. Nome do Projeto
2. Descrição do Projeto
3. Pré-requisitos
Informações sobre o que é necessário para usar o projeto (versão do Python, dependências, etc.)
4. Instalação
Um guia passo-a-passo de como configurar o ambiente de desenvolvimento.
5. Como usar o projeto
6. Testes
Como executar testes no projeto.
7. Documentação
Link para a documentação completa (se disponível) ou uma breve documentação no próprio README.

Conclusão

Próximos passos

Chegamos ao final deste eBook e agora **você é o protagonista** preparado para a prática, para errar, acertar e, principalmente, aprender.

Sim, projetos de dados podem ser um desafio e tanto! Há muito o que se pensar. Mas olha só, **você não está sozinho nessa** e, acredite, cada um dos 12 passos dado é uma vitória

Desenvolvi este guia como um amigo que estará sempre aí, pronto para te ajudar quando as dúvidas surgirem ou quando você precisar de um lembrete sobre os próximos projetos de dados. E sabe de uma coisa?

Conclusão

Próximos passos

Cada projeto será uma aventura única, com obstáculos diferentes, surpresas (boas e nem tão boas assim) e, claro, muitas conquistas.

Os 12 fatores que conversamos aqui não são regras rígidas, **mas sim conselhos de quem** já passou por muitos perrengues e quer te poupar de alguns.

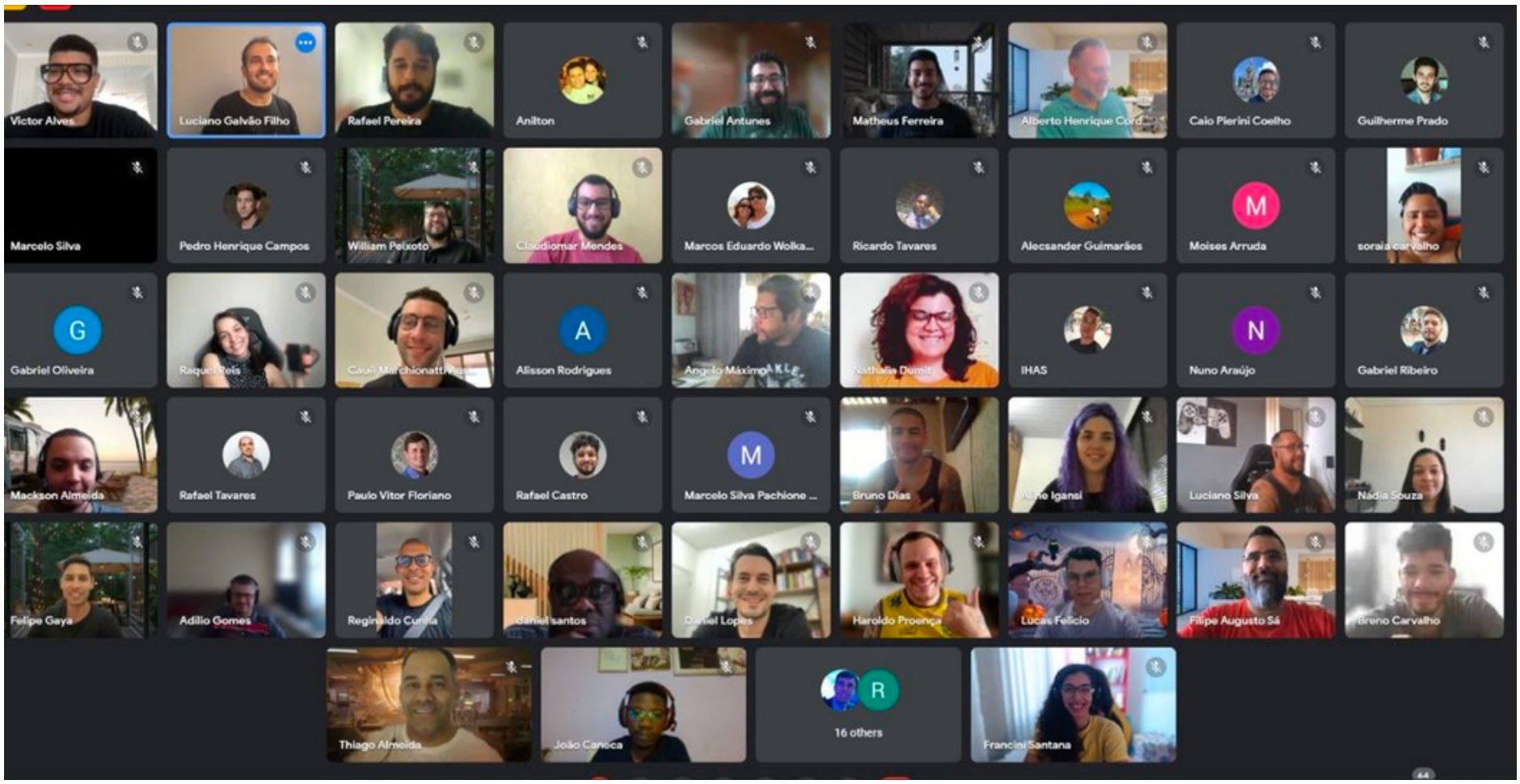
Eles estão aqui para te ajudar a navegar, mas **o capitão desse barco é você!**

Cada projeto será uma nova viagem e, sem dúvida, você descobrirá novas rotas, atalhos e talvez até mesmo novos destinos.

Obrigado!

Até breve

Boa sorte, mantenha a curiosidade acesa e vamos em frente! Que suas análises sejam incríveis, assim como vocês foram no dia 07.



Com muito carinho,
Luciano Filho