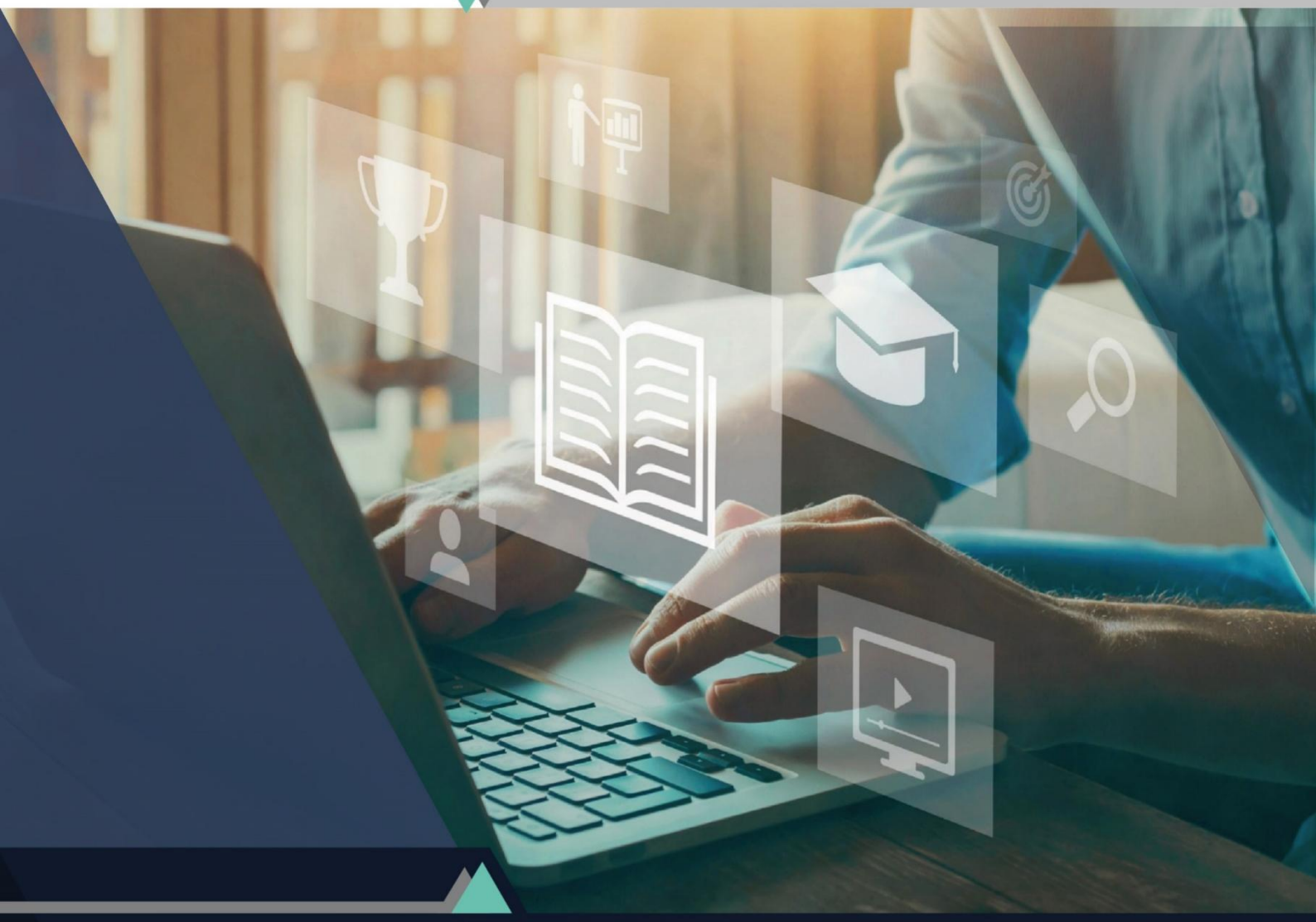


Engenharia de Dados na Prática

Autoria: Jean Martins

Disciplina: Engenharia de Dados na Prática

Título do tema: **A disciplina aborda os conceitos e ferramentas essenciais para a atuação na área de Engenharia de Dados, com foco na aplicação prática de Git, GitHub, Python e SQL. O curso busca capacitar os alunos a gerenciar versionamento de código, desenvolver scripts em Python, manipular bancos de dados com SQL e integrar essas tecnologias em projetos práticos. A disciplina combina teoria com atividades práticas, preparando os alunos para desafios do dia a dia de um engenheiro de dados.**

Autoria: **Jean Felipe Martins da Costa**

Leitura crítica: **leitor, insira aqui seu nome completo**

Objetivos

- Conhecer as linguagens SQL e Python para análise de dados.
    - Utilizar comandos git para interagir com repositórios github.
- Aplicar SQL, Python e Git para simular um ambiente de dados.

1. Preparando o Ambiente de Desenvolvimento Local

Antes de iniciar qualquer projeto prático de Engenharia de Dados, é fundamental preparar o ambiente de desenvolvimento. Ter as ferramentas certas instaladas e configuradas garante maior produtividade e evita problemas futuros. Neste capítulo, abordaremos a instalação e configuração das principais ferramentas necessárias: Python, Visual Studio Code (VS Code) e Git.

Instalando o Python

O Python é uma das linguagens mais utilizadas em Engenharia de Dados devido à sua simplicidade e vasta gama de bibliotecas.

Passo 1: Baixar o Python

- Acesse o site oficial: https://www.python.org/downloads/
- Baixe a versão mais recente recomendada para seu sistema operacional (Windows, macOS ou Linux).

Passo 2: Instalar o Python

- Execute o instalador baixado.
- Importante: Marque a opção "Add Python to PATH" antes de clicar em Install Now.
- Finalize a instalação.

Passo 3: Verificar a Instalação

Abra o terminal ou prompt de comando e digite:

```
python --version
```

Se retornar a versão do Python, a instalação foi concluída com sucesso.

Instalando o Visual Studio Code (VS Code)

O Visual Studio Code é um editor de código leve e altamente extensível, ideal para projetos de Engenharia de Dados.

Passo 1: Baixar o VS Code

- Acesse o site oficial: https://code.visualstudio.com/
- Baixe a versão correspondente ao seu sistema operacional.

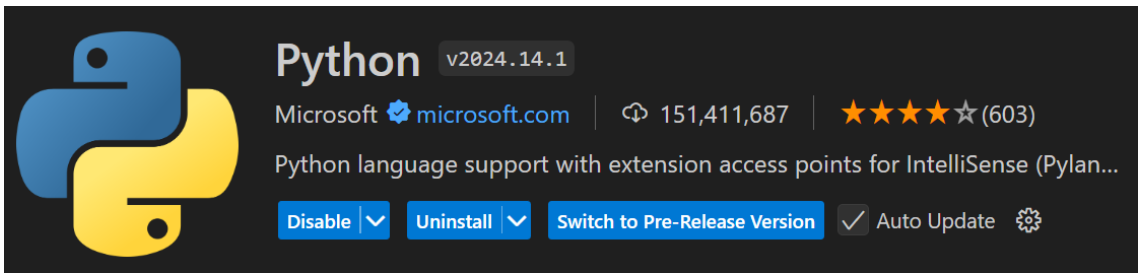
Passo 2: Instalar o VS Code

- Execute o instalador e siga as instruções.
- Marque a opção "Add to PATH" durante a instalação.

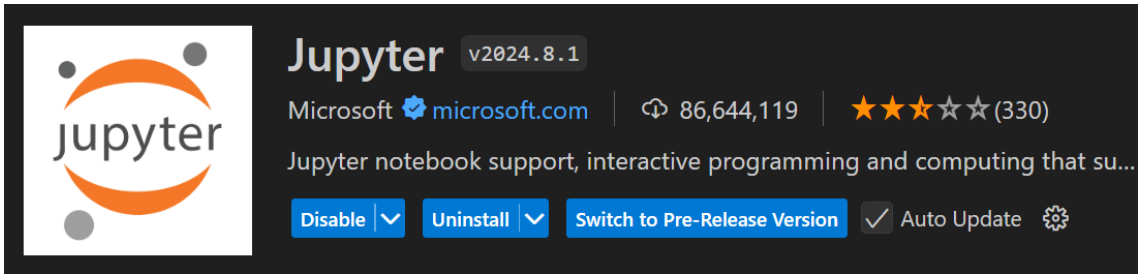
Passo 3: Instalar Extensões Essenciais

Abra o VS Code e instale as seguintes extensões:

- **Python (Microsoft):** Suporte ao desenvolvimento em Python.



- **Jupyter:** Para notebooks interativos.



Instalando o Git

O Git é essencial para controle de versão e colaboração em projetos de Engenharia de Dados.

Passo 1: Baixar o Git

- Acesse o site oficial: https://git-scm.com/downloads
- Baixe a versão adequada ao seu sistema operacional.

Passo 2: Instalar o Git

- Execute o instalador e siga os passos recomendados.
- Mantenha as configurações padrão.

Passo 3: Configurar o Git

Após a instalação, configure seu nome de usuário e e-mail:

```
git config --global user.name "Seu Nome"
git config --global user.email "seuemail@example.com"
```

Passo 3: Configurar o Git

Para verificar se o Git foi instalado corretamente, execute:

```
git --version
```

Se o comando retornar a versão do Git, a instalação foi concluída.

Conclusão

Com Python, VS Code e Git instalados e configurados, seu ambiente de desenvolvimento está pronto para projetos práticos de Engenharia de Dados. No próximo capítulo, exploraremos como criar e gerenciar ambientes virtuais com o venv e como organizar seu projeto de forma eficiente.

2. Git e seus Conceitos Fundamentais

O Git é uma das ferramentas mais importantes para qualquer desenvolvedor, especialmente na área de Engenharia de Dados. Trata-se de um sistema de controle de versão distribuído que permite gerenciar alterações no código-fonte de maneira eficiente, segura e colaborativa. Neste capítulo, abordaremos em detalhes o funcionamento do Git, seus conceitos fundamentais e os principais comandos para iniciar um projeto.

O que é Git?

O Git é um sistema de controle de versão distribuído criado por Linus Torvalds em 2005. Ele foi desenvolvido para ser rápido, eficiente e capaz de lidar com projetos de grande escala. Sua principal função é rastrear alterações em arquivos e permitir a coordenação do trabalho entre diversas pessoas.

Vantagens do Git:

- Rastreamento de mudanças: Histórico completo de alterações.
- Trabalho em equipe: Integração e colaboração facilitada.
- Flexibilidade: Suporte a múltiplos fluxos de trabalho.
- Desempenho: Operações locais rápidas.

Conceitos Fundamentais do Git

O Git é um sistema distribuído, o que significa que cada desenvolvedor tem uma cópia completa do repositório (incluindo histórico de alterações) em sua máquina local. Isso oferece mais segurança e independência, pois não há dependência constante de um servidor central.

Repositórios Locais e Remotos:

- Repositório Local: Está na máquina do desenvolvedor, onde o código é editado, testado e versionado.
- Repositório Remoto: Normalmente hospedado em servidores (GitHub, GitLab, Bitbucket) e usado para compartilhar o projeto com outros desenvolvedores.

Área de Staging

A área de staging é um espaço temporário onde as mudanças são organizadas antes de serem efetivamente salvas no histórico do repositório. Isso permite que o desenvolvedor selecione quais alterações serão incluídas no próximo commit.

Commit

Um commit representa um snapshot do projeto em determinado momento. Ele registra todas as alterações feitas e armazenadas na área de staging.

Branches (Ramificações)

Permite trabalhar em diferentes versões de um projeto simultaneamente. A branch principal geralmente se chama main ou master, mas novas branches podem ser criadas para desenvolver funcionalidades isoladamente.

Fluxo de Trabalho no Git

1. Editar arquivos: Desenvolver novas funcionalidades ou corrigir bugs.
2. Adicionar ao Staging: Preparar as alterações para o commit.
3. Fazer commit: Salvar um snapshot do projeto.
4. Enviar ao remoto: Compartilhar alterações com o time.

Exemplo básico:

1. Inicializa um repositório local
2. Adiciona o arquivo ao staging
3. Cria um commit com os arquivos do staging
4. Conecta ao repositório remoto
5. Envia as alterações remotas

```
git init
git add arquivo.py
git commit -m "Adiciona novo script"
git remote add origin <url-do-repositorio>
git push origin main
```

Principais Comandos do Git:

Inicializar um novo repositório

```
git init
```

Clonar um repositório existente

```
git clone <url-do-repositorio>
```

Verificar o status das alterações

```
git status
```

Adicionar arquivos ao staging

```
git add <arquivo>
git add .    # Adiciona todas as mudanças
```

Realizar um commit

```
git commit -m "Mensagem descritiva"
```

Ver histórico de commits

```
git log
```

Criar e alternar entre branches

```
git branch nova-branch    # Cria uma nova branch
git checkout nova-branch    # Alterna para a nova branch
```

Atualizar o repositório local com mudanças do remoto

```
git pull origin main
```

Enviar alterações para o repositório remoto

```
git push origin main
```

Boas Práticas com Git

- Commits pequenos e frequentes: Facilita o rastreamento de alterações.
- Mensagens de commit claras: Descreva o que foi feito de forma objetiva.
- Uso de branches: Organize o fluxo de trabalho com branches para novas funcionalidades, correções e releases.
- Pull requests: Utilize para revisar e aprovar mudanças antes de integrá-las ao projeto principal.

Conclusão

O Git é uma ferramenta poderosa e essencial para qualquer projeto de Engenharia de Dados. Compreender seus conceitos fundamentais e dominar seus comandos permite gerenciar projetos de maneira organizada, colaborativa e eficiente. Nos próximos capítulos, aprofundaremos o uso de Git em ambientes colaborativos e integração com plataformas como GitHub.

3. GitHub e seus Conceitos Fundamentais

O GitHub é uma plataforma de hospedagem de código-fonte que utiliza o Git como sistema de controle de versão. Ele facilita a colaboração entre desenvolvedores e equipes, oferecendo recursos como repositórios remotos, gerenciamento de projetos, controle de versões e integração com ferramentas de automação. Neste capítulo, exploraremos o funcionamento do GitHub, seus conceitos fundamentais e como utilizá-lo de forma eficiente em projetos de Engenharia de Dados.

O que é o GitHub?

O GitHub é uma plataforma online criada para armazenar e gerenciar projetos de software que utilizam o Git como sistema de controle de versão. Fundado em 2008 e adquirido pela Microsoft em 2018, o GitHub permite que desenvolvedores compartilhem código, colaborem em projetos e automatizem fluxos de trabalho.

Vantagens do GitHub:

- Hospedagem de Repositórios: Armazena projetos e facilita o acesso remoto.
- Colaboração: Permite o trabalho em equipe por meio de Pull Requests e Issues.
- Controle de Versão: Integração com o Git para rastreamento de alterações.
- Automação: Suporte a workflows com GitHub Actions.
- Documentação: Uso de arquivos README e Wikis.

Conceitos Fundamentais

Repositórios

Um repositório é onde os arquivos do projeto são armazenados. Pode ser público (visível para todos) ou privado (acesso restrito).

- Local: Repositório na máquina do desenvolvedor.
- Remoto: Repositório hospedado no GitHub.

Fork

Um fork cria uma cópia de um repositório em sua conta do GitHub, permitindo que você trabalhe em alterações sem afetar o projeto original.

Branches

Branches permitem desenvolver novas funcionalidades de forma isolada. A branch principal geralmente se chama main.

Pull Request (PR)

Pull Requests são solicitações para integrar mudanças de uma branch (geralmente de um fork ou branch secundária) ao repositório principal. São essenciais para revisão de código e colaboração.

Fluxo de um Pull Request:

1. Criação de uma nova branch.
2. Desenvolvimento de funcionalidades.
3. Envio do PR para revisão.
4. Discussão e ajustes.
5. Merge da branch após aprovação.

Markdown para os arquivos README

O Markdown é uma linguagem de marcação leve, criada por John Gruber em 2004, com o objetivo de ser simples de escrever e fácil de converter para HTML. Amplamente utilizado em documentação de projetos, blogs e arquivos README no GitHub, o Markdown permite criar textos formatados de forma rápida e intuitiva. Sua sintaxe direta permite criar documentos bem estruturados com títulos, listas, links, imagens, tabelas e blocos de código.

Vantagens do Markdown:

- Simplicidade: Fácil de aprender e usar.
- Portabilidade: Compatível com várias plataformas.

- Flexibilidade: Permite converter para HTML, PDF e outros formatos.
- Integração: Amplamente usado em plataformas como GitHub, GitLab, Jupyter Notebooks e blogs.

Título

Use o símbolo # para definir títulos. Quanto mais #, menor o nível do título.

```
# Título 1
## Título 2
### Título 3
#### Título 4
##### Título 5
###### Título 6
```

Ênfase de Texto

- Negrito: Use ** ou __ para negrito.
- Itálico: Use * ou _ para itálico.
- Tachado: Use ~~ para tachado.

```
**Texto em negrito**
*Texto em itálico*
~~Texto tachado~~
```

Listas

- Listas não ordenadas: Use -, * ou +.
- Listas ordenadas: Use números seguidos de ponto.

```
- Item 1
- Item 2
  - Subitem 2.1
  - Subitem 2.2

1. Primeiro item
2. Segundo item
3. Terceiro item
```

Links e Imagens

- Links: [Texto](URL)
- Imagens: ![Texto alternativo](URL-da-imagem)

```
[Visite o GitHub](https://github.com)

![Logo do GitHub](https://github.githubassets.com/images/modules/logos_page/GitHub-Mark.png)
```

Linhas Horizontais

Use ---, *** ou ___ para inserir uma linha horizontal.

```
---
```

Blocos de Código

Para destacar trechos de código, use crases simples (`) para inline ou três crases (```) para blocos de código.

````
`print("Olá, mundo!")`


```python
# Exemplo em Python
print("Olá, mundo!")
```
````

Tabelas

Para criar tabelas, utilize | para colunas e - para separar o cabeçalho.

```
| Nome       | Idade | Cidade            |
|------------|-------|-------------------|
| Ana        | 25    | São Paulo         |
| João       | 30    | Rio de Janeiro    |
```

Boas Práticas no Uso de Markdown:

- Organização: Estruture o documento com títulos claros.
- Consistência: Use a mesma sintaxe para formatações semelhantes.
- Legibilidade: Prefira clareza ao excesso de formatação.
- Comentários: Use HTML <!-- comentário --> para notas invisíveis.

4. Introdução ao SQL

Structured Query Language (SQL) é a linguagem padrão para gerenciamento de bancos de dados relacionais. É amplamente utilizada para criar, modificar, consultar e manipular dados em sistemas de gerenciamento de banco de dados (SGBD) como MySQL, PostgreSQL, SQL Server e SQLite. Dominar SQL é essencial para profissionais de Engenharia de Dados, pois permite a organização e análise eficiente de grandes volumes de dados.

Conceitos Fundamentais de SQL

SQL é uma linguagem declarativa usada para interagir com bancos de dados relacionais. Seu principal objetivo é facilitar a manipulação de dados por meio de comandos específicos.

Vantagens do SQL:

- Padronização: É a linguagem padrão para bancos de dados relacionais.
- Flexibilidade: Permite consultas simples e complexas.
- Escalabilidade: Funciona em bancos pequenos e grandes.
- Integração: Compatível com diversas linguagens de programação.

Componentes do SQL:

- DDL (Data Definition Language): Define a estrutura do banco de dados.
- DML (Data Manipulation Language): Manipula os dados armazenados.
- DCL (Data Control Language): Controla permissões de acesso.
- DQL (Data Query Language): Realiza consultas de recuperação de dados.

Principais Comandos SQL

Comandos DDL

CREATE: Cria estruturas no banco de dados.

```
CREATE TABLE clientes (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(100),
    idade INT
);
```

ALTER: Modifica estruturas existentes.

```
ALTER TABLE clientes ADD telefone VARCHAR(15);
```

DROP: Exclui tabelas ou bancos de dados.

```
DROP TABLE clientes;
```

Comandos DML

INSERT: Insere dados.

```
INSERT INTO clientes (id, nome, email, idade) VALUES (1, 'Ana',
'ana@email.com', 25);
```

UPDATE: Atualiza dados.

```
UPDATE clientes SET idade = 26 WHERE id = 1;
```

DELETE: Remove dados.

```
DELETE FROM clientes WHERE id = 1;
```

Comandos DQL

SELECT: Consulta dados.

```
SELECT nome, email FROM clientes;
```

Comandos DCL

GRANT: Concede permissões.

```
GRANT SELECT, INSERT ON clientes TO usuario;
```

REVOKE: Remove permissões.

```
REVOKE INSERT ON clientes FROM usuario;
```

Consultas Avançadas

Cláusula WHERE

Filtra registros com base em condições.

```
SELECT * FROM clientes WHERE idade > 18;
```

Ordenação de Dados (ORDER BY)

```
SELECT * FROM clientes ORDER BY idade DESC;
```

Junções (JOIN)

Combina dados de múltiplas tabelas.

```
SELECT clientes.nome, pedidos.valor
FROM clientes
JOIN pedidos ON clientes.id = pedidos.cliente_id;
```

Agrupamento de Dados (GROUP BY)

```
SELECT idade, COUNT(*)
FROM clientes
GROUP BY idade;
```

Funções de Agregação

- COUNT: Conta registros.
- SUM: Soma valores.
- AVG: Calcula média.
- MAX/MIN: Valor máximo/mínimo.

```
SELECT AVG(idade) FROM clientes;
```

Boas Práticas em SQL

- Nomes Descritivos: Utilize nomes claros para tabelas e colunas.
- Normalização: Organize dados para evitar redundâncias.
- Índices: Crie índices para otimizar consultas.
- Segurança: Defina permissões adequadas.

Conclusão

SQL é uma linguagem essencial para manipulação e análise de dados. Seus comandos permitem estruturar, consultar e gerenciar grandes volumes de informação de forma eficiente. Nos próximos capítulos, exploraremos a modelagem de dados e técnicas avançadas de otimização de consultas SQL.

5. Introdução ao Python

Python é uma linguagem de programação de alto nível, amplamente utilizada em diversos campos como desenvolvimento web, ciência de dados, automação, inteligência artificial e muito mais. Sua sintaxe simples e clara facilita o aprendizado e torna o desenvolvimento mais ágil e eficiente.

Comandos Úteis em Python

Exibir Mensagens no Console

```
print("Olá, mundo!")
```

Trabalhando com Listas

```
lista = [1, 2, 3, 4, 5]
lista.append(6)    # Adiciona um elemento
print(lista[0])    # Acessa o primeiro elemento
```

Estruturas Condicionais

```
idade = 18
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")
```

Laços de Repetição

```
for i in range(5):
    print(i)
```

Definindo Funções

```
def saudacao(nome):
    return f"Olá, {nome}!"
print(saudacao("Ana"))
```

Manipulação de Arquivos

```
with open("arquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```

Boas Práticas em Python

- Ambientes Virtuais: Utilize venv para gerenciar dependências de projetos.
- Nomes Descritivos: Use nomes claros para variáveis e funções.
- Organização: Estruture o código com módulos e pacotes.
- Documentação: Inclua comentários e docstrings.

6. Biblioteca Pandas

A biblioteca Pandas é uma das ferramentas mais poderosas e populares do Python para manipulação e análise de dados. Projetada para facilitar o trabalho com grandes volumes de dados, ela oferece estruturas de dados rápidas, flexíveis e expressivas, como Series e DataFrame, além de diversas funcionalidades para leitura, escrita e transformação de dados.

Instalando o Pandas

Para começar a usar o Pandas, é necessário instalá-lo utilizando o gerenciador de pacotes pip.

Comando de Instalação

```
pip install pandas
```

Para verificar se a instalação foi concluída com sucesso, execute:

```
import pandas as pd
print(pd.__version__)
```

Estruturas de Dados do Pandas

Series

Uma Series é uma estrutura unidimensional semelhante a uma lista ou array.

```
import pandas as pd
dados = [10, 20, 30, 40]
serie = pd.Series(dados)
print(serie)
```

DataFrame

O DataFrame é uma tabela bidimensional com rótulos nas linhas e colunas, similar a uma planilha do Excel.

```
import pandas as pd
dados = {
    "Nome": ["Ana", "João", "Carlos"],
    "Idade": [28, 35, 42],
    "Cidade": ["São Paulo", "Rio de Janeiro", "Belo Horizonte"]
}
df = pd.DataFrame(dados)
print(df)
```

Interagindo com Arquivos:

Lendo CSV:

```
df = pd.read_csv("dados.csv")
```

Lendo Excel:

```
df = pd.read_excel("dados.xlsx")
```

Escrevendo CSV:

```
df.to_csv("saida.csv", index=False)
```

Manipulando Dados

Selecionando uma coluna:

```
print(df["Nome"])
```

Selecionando múltiplas colunas:

```
print(df[["Nome", "Idade"]])
```

Selecionando uma linha pelo índice:

```
print(df.loc[0])
```

Filtrando dados com condição:

```
print(df[df["Idade"] > 30])
```

Ordenando pelo nome:

```
print(df.sort_values(by="Nome"))
```

Renomar colunas :

```
df.rename(columns={‘nome_antigo’: ‘novo_nome’})
```

Deletar colunas :

```
df.drop(columns=['Idade'])
```

Boas Práticas com Pandas

- Evite loops: Use métodos vetorizados.
- Gerencie memória: Use tipos de dados adequados.
- Documente o processo: Explique transformações complexas.

Conclusão

A biblioteca Pandas é essencial para análise e manipulação de dados em Python. Dominar suas principais funcionalidades permite trabalhar de forma mais eficiente e produtiva. Nos próximos capítulos, exploraremos integrações do Pandas com outras ferramentas de análise de dados e técnicas avançadas de manipulação.

Referências

Taylor, A. G. SQL para Leigos. 8. ed. Rio de Janeiro: Alta Books, 2019.

Chacon, S.; Straub, B. Pro Git. 2. ed. Berkeley: Apress, 2014.

Ramalho, L. Python Fluente. 2. ed. São Paulo: Novatec Editora, 2020.