Introdução ao Dataframe e Manipulação de Dados com Pandas

Aprenda a importar, manipular e consultar conjuntos de dados utilizando uma das ferramentas mais poderosas para análise de dados em Python

Aula 04 - UC2

Objetivos da Aula

- Compreender o conceito de **Dataframe** e suas aplicações em análise de dados.
- Aprender a utilizar funcionalidades básicas do **Pandas** para importar, manipular e consultar conjuntos de dados.
- Desenvolver habilidades para **identificar padrões**, **tratar dados ausentes** e realizar operações analíticas iniciais.
- Aplicar atitudes de **colaboração**, foco em **organização de scripts**, responsabilidade e atenção aos detalhes na manipulação de dados.

Conexão com o Projeto

Após modelar o banco de dados e definir as regras éticas, o próximo passo é a **preparação dos dados** que serão armazenados no banco. Este processo é conhecido como **ETL** (Extração, Transformação e Carga).

O **Pandas** é a principal ferramenta para realizar este processo, permitindo que você trabalhe com dados de forma eficiente e prepare-os para serem inseridos no banco de dados.

Extração (Extract)

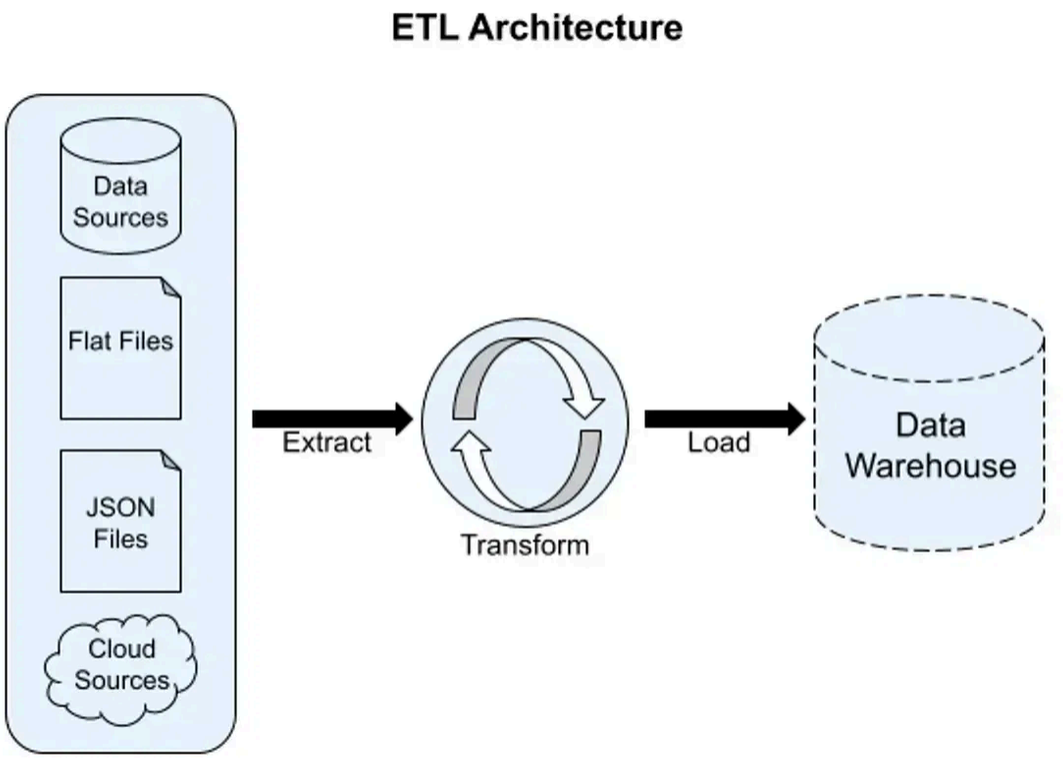
Obtenção de dados de diversas fontes como arquivos CSV, Excel, bancos de dados ou APIs.

Transformação (Transform)

Limpeza, formatação, agregação e enriquecimento dos dados para atender às necessidades do projeto.

Carga (Load)

Inserção dos dados transformados no banco de dados final para armazenamento e análise.



O Conceito de Dataframe

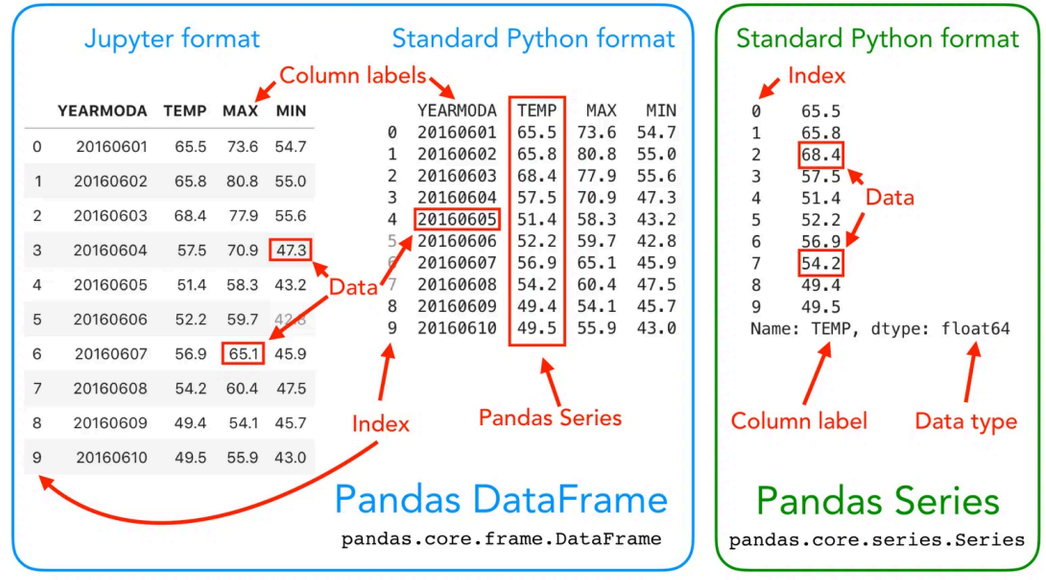
Um **Dataframe** é a estrutura central do Pandas, similar a uma tabela ou planilha, mas com recursos avançados para manipulação e análise de dados.

Componentes Principais

- **Índices (linhas)**: Identificadores únicos para cada registro, que podem ser numéricos ou baseados em rótulos.
- **Colunas (séries)**: Conjuntos de dados do mesmo tipo, cada uma representando uma variável ou atributo.
- **Células**: Interseção entre linhas e colunas, contendo os valores individuais dos dados.

Vantagens em relação a listas ou dicionários

- **Rapidez**: Operações otimizadas para grandes conjuntos de dados
- **Flexibilidade**: Manipulação intuitiva de dados heterogêneos
- **Funções prontas**: Métodos integrados para estatística e limpeza de dados
- **Integração**: Compatibilidade com bibliotecas de visualização e análise



Estrutura de um Dataframe no Pandas, mostrando índices, colunas e dados

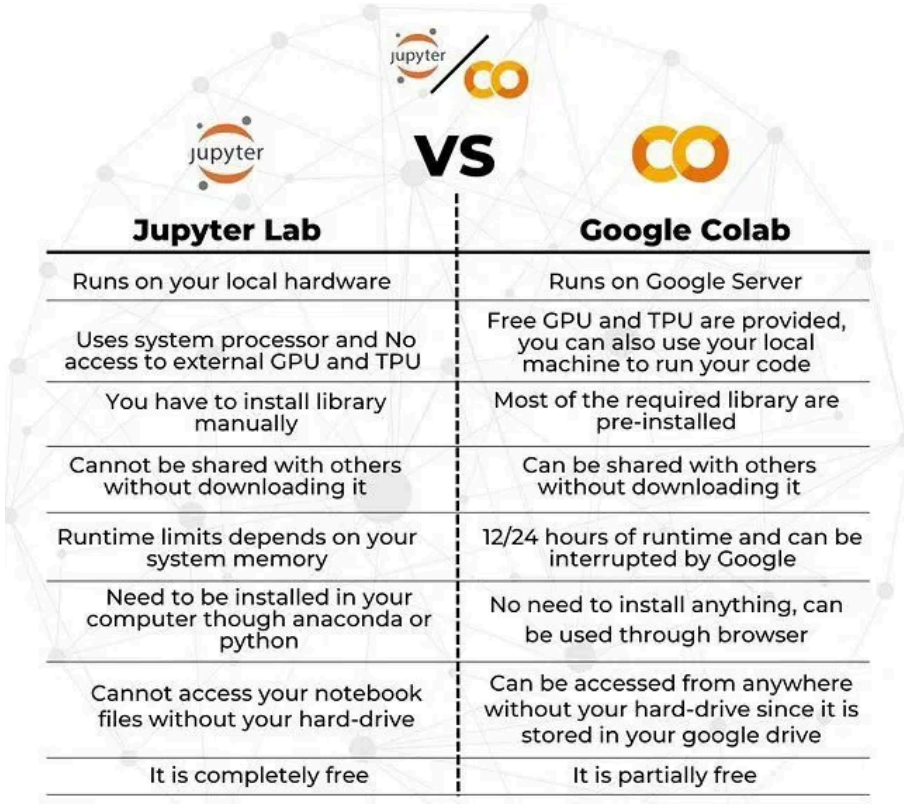
Introdução ao Pandas

Pandas é uma biblioteca Python de código aberto que fornece estruturas de dados de alto desempenho e ferramentas de análise de dados. É construída sobre a biblioteca NumPy e amplamente utilizada para manipulação e análise de dados.

Para começar a utilizar o Pandas, primeiro é necessário importá-lo:

```
import pandas as pd
```

Para trabalhar com Pandas de forma interativa, recomendamos utilizar ambientes como **Jupyter Notebook** ou **Google Colab**, que permitem a execução de código em células e visualização imediata dos resultados.



Ambientes de Trabalho Recomendados

Jupyter Notebook

- Execução local
- Integração com ambiente local

Google Colab

- Acesso via navegador
- Fácil compartilhamento

Atividade Prática 1: Exploração de Dataset Real

Objetivo: Explorar um dataset real utilizando as funcionalidades básicas do Pandas.
Material: Dataset em formato CSV (ex: dados de vendas, clima ou educacionais).

1 Importação do Dataset

```
import pandas as pd
df = pd.read_csv('nome_do_arquivo.csv')
```

2 Visualização Inicial dos Dados

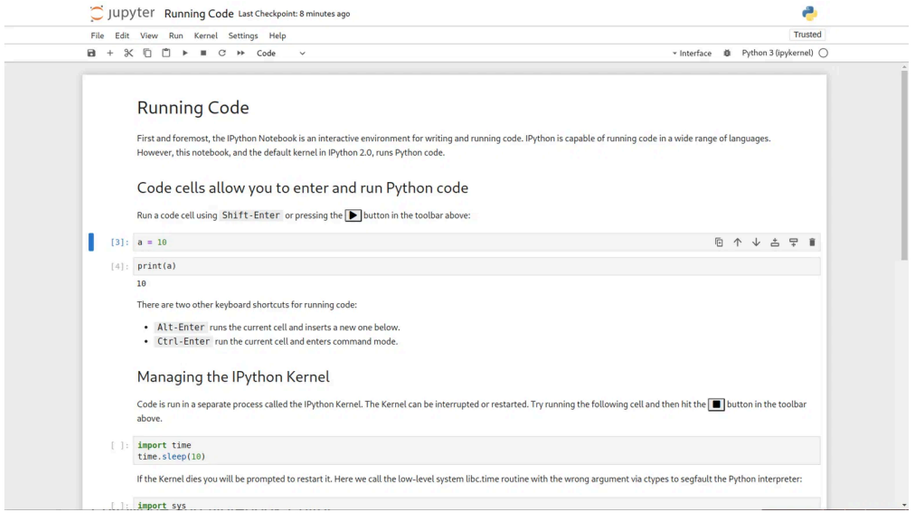
```
# Primeiras 5 linhas
df.head()

# Últimas 5 linhas
df.tail()
```

3 Verificação da Estrutura do Dataset

```
# Dimensões do dataset (linhas x colunas)
df.shape

# Informações sobre tipos de dados
df.info()
```

4 Análise Estatística Rápida

```
# Estatísticas descritivas
df.describe()
```




```
# Exemplo de saída de df.head()
   ID   Produto  Quantidade  Preço  Data_Venda
0  1    Laptop           2   1200   2023-01-15
1  2    Mouse            5     25   2023-01-16
2  3    Monitor          3    150   2023-01-16
```

Dica: Sempre comece com estas operações básicas para entender a estrutura dos dados.

Filtros e Seleção de Dados

Após importar um dataset, você precisará selecionar colunas específicas ou filtrar linhas com base em condições.

Seleção de Colunas

```
# Seleção de uma única coluna
df['nome_da_coluna']

# Seleção de múltiplas colunas
df[['coluna_A', 'coluna_B', 'coluna_C']]
```

Filtros Simples

```
# Filtrar por uma condição
df[df['idade'] > 30]

# Filtrar por múltiplas condições
df[(df['idade'] > 30) & (df['salario'] > 5000)]
```

```
Type- <class 'pandas.core.frame.DataFrame'>
```

Out[84]:

	Name	Code
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
...	...	...
244	Wallis and Futuna	WF
245	Western Sahara	EH
246	Yemen	YE
247	Zambia	ZM
248	Zimbabwe	ZW

249 rows × 2 columns

```
# Exemplo prático: Filtrar vendas acima de 1000
vendas_filtradas = df[(df['valor'] > 1000)]
resultado = vendas_filtradas[['produto', 'valor', 'data']]
```

Lembre-se: Os filtros retornam uma **cópia** do DataFrame original.

Manipulação de Dados: Identificação de Dados Ausentes

Por que tratar valores nulos é importante?

Dados ausentes (valores **NaN** ou **None**) podem inviabilizar análises ou a importação para o banco de dados.

Como identificar valores ausentes

```
# Verificar valores nulos em todo o DataFrame
df.isnull().sum()
```

Impacto dos dados ausentes

- ⚠ Resultados estatísticos incorretos
- ⚠ Modelos de ML enviesados
- ⚠ Visualizações distorcidas
- ⚠ Rejeição na importação de dados


```
# Exemplo de saída do comando df.isnull().sum()
nome             0
idade            12
salario          5
departamento     0
data_admissao    8
dtype: int64
```

💡 **Dica:** Sempre verifique valores ausentes antes de iniciar qualquer análise.

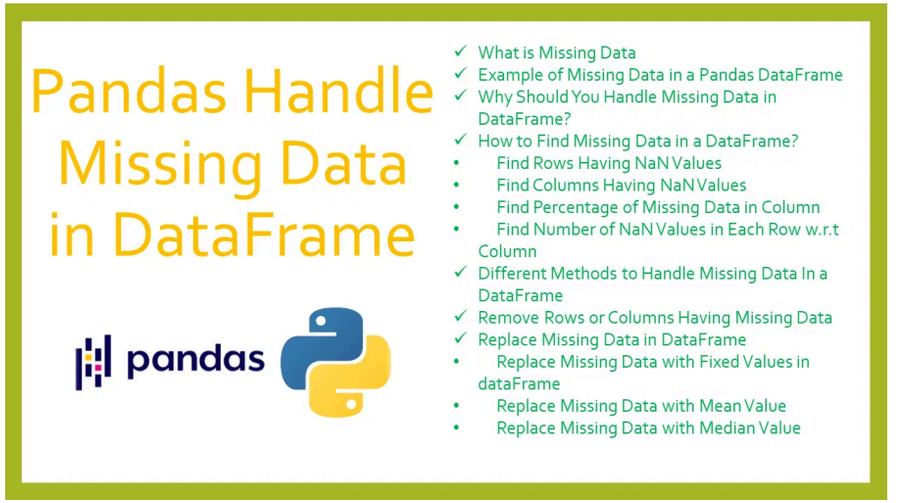
Tratamento de Ausências

Exclusão de Valores Nulos

A exclusão é uma abordagem simples, mas deve ser usada com cautela para evitar perda de informações.

```
# Remover linhas com qualquer valor nulo
df_limpo = df.dropna()

# Remover linhas onde todas as colunas são nulas
df_limpo = df.dropna(how='all')
```

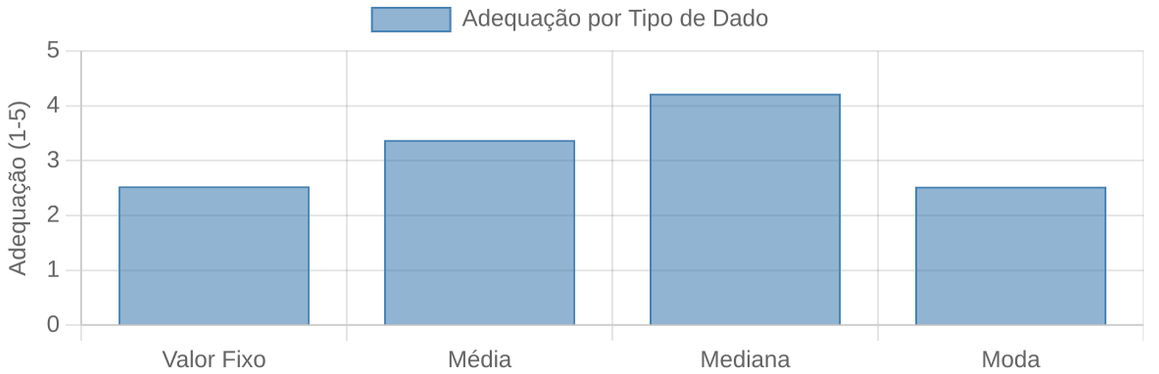
> **Atenção**
>
> Use a exclusão com cautela, especialmente em datasets pequenos.



Preenchimento de Valores Ausentes

O preenchimento permite substituir valores nulos por valores específicos ou calculados.

```
# Preencher com um valor específico
df['idade'].fillna(0)

# Preencher com a média da coluna
df['salario'].fillna(df['salario'].mean())

# Preencher com a mediana (mais robusto a outliers)
df['salario'].fillna(df['salario'].median())
```



Dica: A escolha do método deve considerar a natureza dos dados.

Modificação de Colunas

🔄 Mudança de Tipos de Dados

Frequentemente, os dados importados não estão no formato correto. O Pandas permite converter facilmente entre tipos de dados.

```
# Converter coluna para tipo inteiro
df['idade'] = df['idade'].astype(int)

# Converter coluna para tipo float
df['salario'] = df['salario'].astype(float)

# Converter coluna para tipo data
df['data_admissao'] = pd.to_datetime(df['data_admissao'])
```

➕ Criação de Colunas Derivadas

Criar novas colunas a partir de colunas existentes é uma operação comum para enriquecer o dataset.

```
# Criar coluna com operação aritmética
df['salario_anual'] = df['salario_mensal'] * 12

# Concatenar strings
df['nome_completo'] = df['nome'] + ' ' + df['sobrenome']

# Criar coluna com base em condição
df['faixa_etaria'] = np.where(df['idade'] < 30, 'Jovem',
'Adulto')
```

Exemplo Prático: Tratamento de Datas

```
# Importar dados com datas em formato string
df = pd.read_csv('vendas.csv')

# Converter para datetime
df['data_venda'] = pd.to_datetime(df['data_venda'])

# Extrair componentes da data
df['mes'] = df['data_venda'].dt.month
df['ano'] = df['data_venda'].dt.year
df['dia_semana'] = df['data_venda'].dt.day_name()
```

Desafio de Consistência

Identifique e corrija inconsistências comuns nos dados:

- Datas em formatos diferentes
- Valores nulos em campos essenciais
- Strings com espaços extras ou caracteres especiais
- Números armazenados como strings

```
# Exemplo: Remover espaços extras
df['produto'] = df['produto'].str.strip()

# Exemplo: Padronizar texto
df['categoria'] = df['categoria'].str.lower()
```

💡 **Lembre-se:** Documentar todas as transformações realizadas é uma boa prática para garantir a reprodutibilidade da análise.

Organização e Boas Práticas

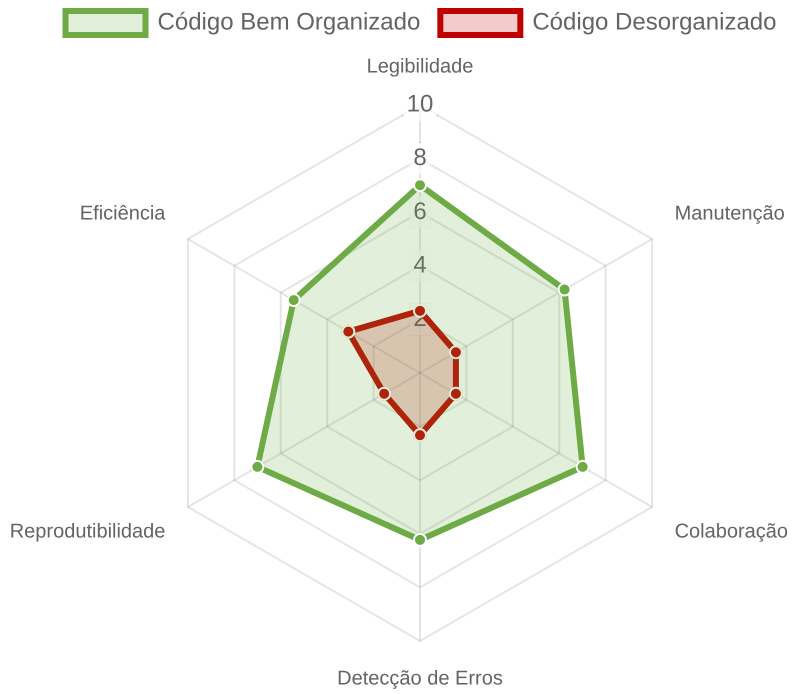
Organização de Scripts

- Use **comentários claros** para documentar o propósito de cada bloco de código.
- Adote **nomes de variáveis descritivos** que indiquem claramente seu conteúdo e propósito.
- Divida o código em **seções lógicas**: importação, limpeza, transformação e análise.

```
# Exemplo de código bem organizado

# 1. Importação de bibliotecas
import pandas as pd
import numpy as np

# 2. Carregamento de dados
df = pd.read_csv('vendas.csv')

# 3. Limpeza de dados
df_limpo = df.dropna(subset=['valor_venda'])

# 4. Transformação
df_limpo['data'] = pd.to_datetime(df_limpo['data'])

# 5. Análise
vendas_mensais =
df_limpo.groupby(df_limpo['data'].dt.month)
['valor_venda'].sum()
```

Responsabilidade na Transformação

- **Documente todas as transformações** realizadas nos dados para garantir reprodutibilidade.
- **Valide os resultados** após cada etapa de transformação para evitar propagação de erros.



Impacto dos Erros na Análise

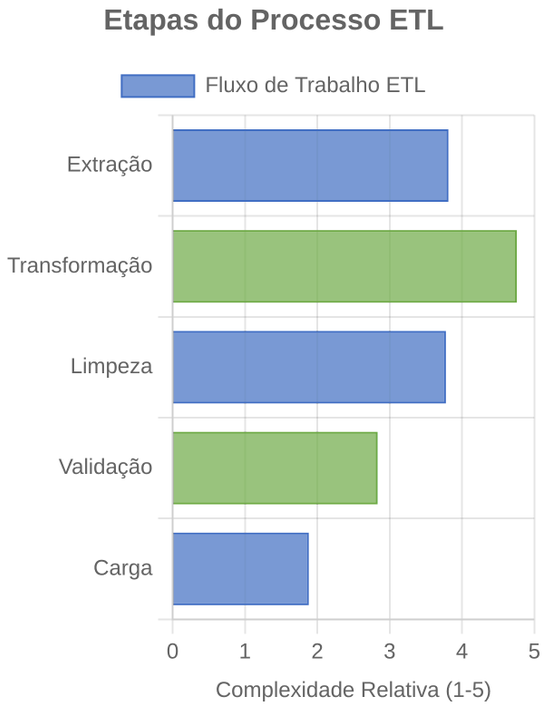
Erros na manipulação de dados podem se propagar e amplificar ao longo da análise, comprometendo decisões baseadas nos resultados. A atenção aos detalhes e a validação constante são essenciais para garantir a confiabilidade das conclusões.

Lembre-se: Um código bem organizado é mais fácil de entender, depurar e reutilizar.

Encerramento e Próximos Passos

Síntese dos Principais Conceitos

- **Dataframes** são estruturas tabulares poderosas para manipulação de dados em Python
- **Pandas** oferece funções para importação, seleção, filtragem e transformação de dados
- A **identificação e tratamento de dados ausentes** é essencial para análises confiáveis
- **Boas práticas** incluem organização de scripts, comentários claros e responsabilidade na transformação



Conexão com a Próxima Etapa do Projeto

1. Aplicar as técnicas de limpeza e transformação aos dados do projeto
2. Preparar os Dataframes limpos para carga no banco de dados modelado
3. Implementar a conexão entre Python e o banco de dados na nuvem
4. Desenvolver consultas e análises sobre os dados armazenados

Lembre-se: O processo ETL é cíclico e iterativo. À medida que novos dados chegam, o ciclo se repete para manter o banco de dados atualizado.