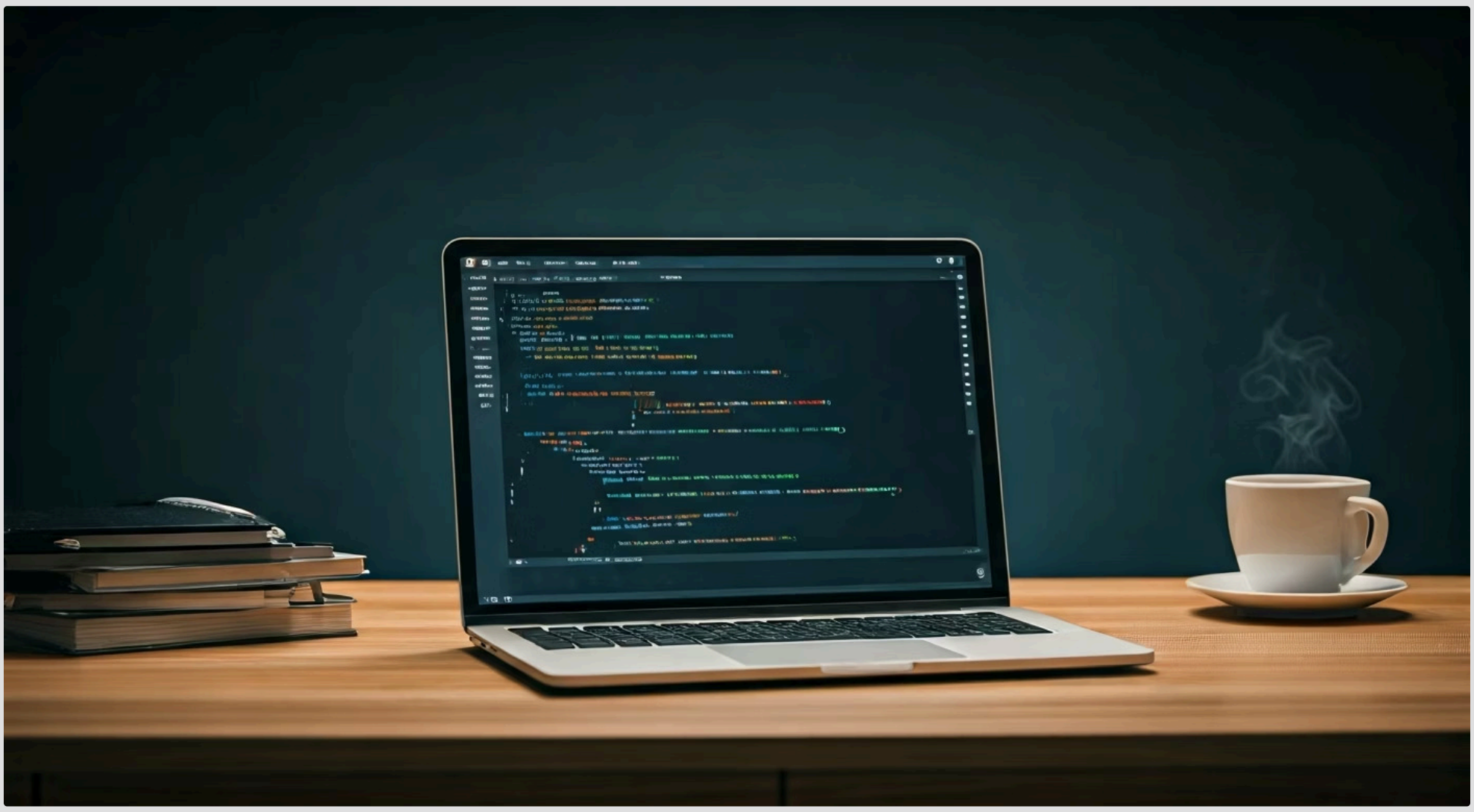
Aula 5

A Biblioteca Pandas: Estruturas de Dados Series e DataFrame



Bem-vindos à nossa jornada pelo universo da análise de dados com Python! Se você já se sentiu sobrecarregado pela quantidade de informações que precisamos processar diariamente, seja em relatórios acadêmicos, projetos de pesquisa ou na preparação para um desafio profissional, saiba que não está sozinho. A boa notícia é que existe uma ferramenta poderosa capaz de transformar esse desafio em uma tarefa muito mais intuitiva e eficiente: a biblioteca Pandas.

Nesta aula, vamos mergulhar no coração do Pandas, explorando suas duas estruturas de dados fundamentais: as Series e os DataFrames. Entender como elas funcionam é o primeiro passo para desbloquear o potencial de manipulação e análise de dados que o Python oferece, permitindo que você organize, visualize e extraia insights valiosos de conjuntos de dados complexos. Ao final, você será capaz de criar, inspecionar e compreender a essência dessas estruturas, preparando o terreno para análises mais avançadas.

Nosso objetivo é que você não apenas compreenda os conceitos, mas que também se sinta confiante para aplicá-los em cenários reais. Veremos como o Pandas se integra ao ecossistema de análise de dados, utilizando ambientes como Jupyter Notebooks e Google Colab, que são padrões da indústria. Prepare-se para dar um salto significativo em suas habilidades de ciência de dados, conectando o que você já sabe sobre programação Python com as novas ferramentas que vamos explorar.

Desvendando o Pandas: A Ferramenta Essencial para Análise de Dados

No vasto ecossistema de Python para análise de dados, o Pandas se destaca como uma das bibliotecas mais influentes e amplamente utilizadas. Mas por que ele é tão crucial? Imagine que você tem uma pilha de documentos desorganizados, com informações espalhadas em diferentes formatos e sem uma estrutura clara. Tentar extrair um padrão ou uma conclusão a partir desse caos seria uma tarefa árdua e demorada. O Pandas surge exatamente para resolver esse problema, oferecendo estruturas de dados flexíveis e de alto desempenho, projetadas para tornar a manipulação e a análise de dados tabulares uma brisa.

Por que Pandas?

- Estruturas de dados flexíveis e poderosas
- Integração perfeita com NumPy e outras bibliotecas
- Manipulação intuitiva de dados tabulares
- Alto desempenho em operações de dados
- Padrão da indústria para ciência de dados

Ele atua como a espinha dorsal para muitas operações de ciência de dados, desde a limpeza inicial e transformação dos dados até a preparação para modelagem estatística e visualização. Sua popularidade reside na sua capacidade de lidar com dados de forma intuitiva, permitindo que você trabalhe com tabelas de maneira semelhante a uma planilha eletrônica ou um banco de dados, mas com o poder e a flexibilidade da programação Python. É a ponte entre seus dados brutos e os insights que você busca.

Para quem já tem alguma familiaridade com Python e, talvez, com a biblioteca NumPy para operações numéricas, o Pandas se construirá sobre essa base, adicionando uma camada de funcionalidade que facilita enormemente o trabalho com dados rotulados e heterogêneos. Ele é o alicerce que sustenta a maioria dos projetos de análise de dados, sendo indispensável para qualquer profissional ou estudante que deseje atuar na área.

A Estrutura Series: O Bloco Fundamental com Rótulos

Array Unidimensional

Uma única coluna de dados com valores sequenciais

Índice Personalizado

Cada valor possui um rótulo único para acesso direto

Tipos Diversos

Suporta números, strings, booleanos e objetos Python

Antes de mergulharmos em tabelas complexas, precisamos entender o elemento básico do Pandas: a Series. Pense em uma Series como uma única coluna de uma planilha ou uma lista de valores, mas com uma característica especial: cada valor possui um rótulo, que chamamos de índice. Essa combinação de valores e seus respectivos rótulos torna a Series muito mais poderosa do que uma lista Python comum ou um array NumPy, pois permite um acesso e manipulação de dados mais semânticos e flexíveis.

Exemplo Prático

Imagine que você está registrando as notas de uma turma. Em uma lista simples, você teria [85, 92, 78]. Mas como saber a qual aluno cada nota pertence? Com uma Series, você pode associar cada nota ao nome do aluno, como {'Alice': 85, 'Bob': 92, 'Carlos': 78}. Isso não só organiza a informação, mas também facilita a busca e a compreensão dos dados. É como ter uma lista onde cada item tem um nome próprio, além de sua posição numérica.

Diferencial da Series

A capacidade de ter índices personalizados é o que diferencia a Series e a torna tão útil. Ela é otimizada para operações rápidas, especialmente quando se trata de alinhamento de dados.

```
import pandas as pd

# Criando uma Series a partir de uma lista
notas = pd.Series([85, 92, 78, 95])
print("Series de Notas (índice padrão):\n", notas)

# Criando uma Series com índices personalizados (rótulos)
notas_alunos = pd.Series([85, 92, 78, 95],
                         index=['Alice', 'Bob', 'Carlos', 'Diana'])
print("\nSeries de Notas (índice personalizado):\n", notas_alunos)

# Acessando um valor pelo rótulo
print("\nNota da Alice:", notas_alunos['Alice'])
```

Essa capacidade de ter índices personalizados é o que diferencia a Series e a torna tão útil. Ela pode armazenar qualquer tipo de dado (números, strings, booleanos, objetos Python) e é otimizada para operações rápidas, especialmente quando se trata de alinhamento de dados. Compreender a Series é o primeiro passo para dominar a manipulação de dados no Pandas, pois ela é a base sobre a qual a estrutura mais complexa, o DataFrame, é construída.

Series na Prática: Indexação e Seleção Inteligente

Compreender a Series é um passo crucial, mas saber como extrair informações dela é onde a verdadeira mágica acontece. A indexação e seleção em uma Series são operações fundamentais que nos permitem acessar valores específicos ou subconjuntos de dados de forma eficiente. Diferente das listas Python, onde o acesso é majoritariamente numérico (pela posição), a Series oferece a flexibilidade de usar tanto índices numéricos quanto rótulos personalizados.

.loc

Location - Seleção baseada em rótulos

Use quando souber o nome/rótulo do índice

.iloc

Integer Location - Seleção baseada em posições

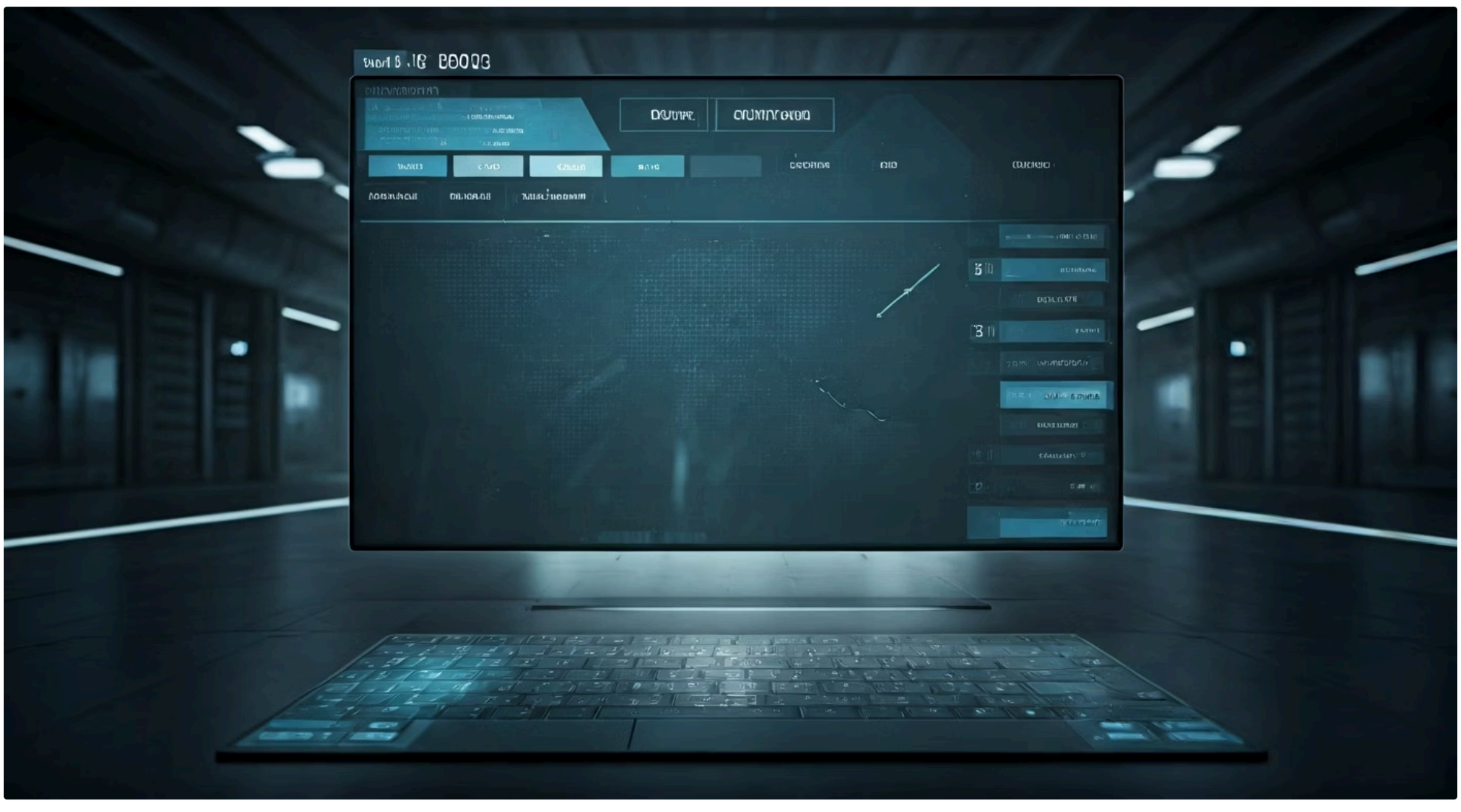
Use quando souber a posição numérica

Essa dualidade é extremamente poderosa. Você pode querer acessar a primeira nota da lista (pela posição) ou a nota de um aluno específico (pelo nome). O Pandas nos oferece métodos como .loc e .iloc para lidar com essas situações de forma clara e intencional. O .loc (location) é usado para seleção baseada em rótulos, enquanto o .iloc (integer location) é usado para seleção baseada em posições inteiras. Essa distinção é vital para evitar ambiguidades e garantir que você esteja sempre acessando os dados corretos.

Analogia do Armário de Arquivos

Pense nisso como um armário de arquivos. Você pode pegar o "primeiro arquivo da segunda prateleira" (índice numérico) ou "o arquivo do cliente 'Silva'" (rótulo). Ambos os métodos são válidos e úteis, dependendo do que você precisa encontrar.

```
import pandas as pd

# Criando uma Series com índices personalizados
temperaturas = pd.Series([22, 25, 19, 28, 23],
                         index=['Seg', 'Ter', 'Qua', 'Qui', 'Sex'])
print("Temperaturas diárias:\n", temperaturas)

# Usando .loc para selecionar pelo rótulo
print("\nTemperatura de Quarta-feira (usando .loc):",
      temperaturas.loc['Qua'])

# Usando .iloc para selecionar pela posição (índice inteiro)
print("Temperatura do segundo dia (usando .iloc):",
      temperaturas.iloc[1])

# Selecionando múltiplos itens por rótulo
print("\nTemperaturas de Terça a Quinta (usando .loc):\n",
      temperaturas.loc['Ter':'Qui'])

# Selecionando múltiplos itens por posição
print("Temperaturas dos três primeiros dias (usando .iloc):\n",
      temperaturas.iloc[0:3])
```



Dominar essas técnicas de seleção é essencial para qualquer tarefa de análise de dados, pois permite que você filtre e foque nas informações mais relevantes para sua investigação.

A Estrutura DataFrame: A Tabela de Dados Bidimensional

Se a Series é uma coluna, o DataFrame é a tabela completa. Ele é a estrutura de dados mais utilizada no Pandas e representa o formato padrão para a maioria dos conjuntos de dados que encontramos no mundo real, como planilhas, tabelas de banco de dados ou arquivos CSV. Um DataFrame pode ser visualizado como uma coleção de Series, onde cada Series representa uma coluna da tabela, e todas elas compartilham o mesmo índice, que atua como os rótulos das linhas.

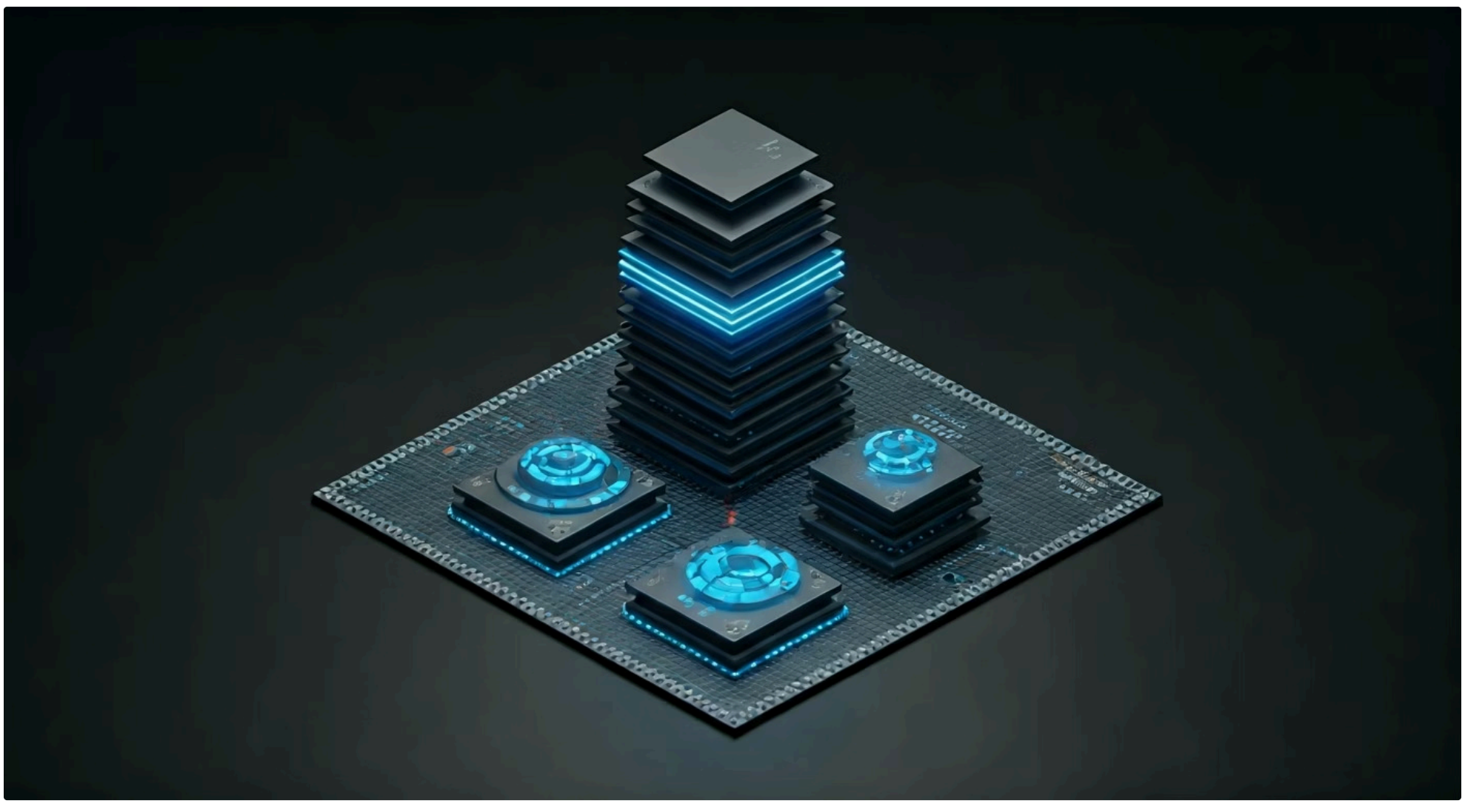
Imagine uma planilha eletrônica que você usa para organizar informações de clientes, vendas ou resultados de experimentos. Cada coluna tem um nome (como "Nome do Cliente", "Valor da Venda", "Data") e cada linha representa um registro único. O DataFrame do Pandas replica essa estrutura de forma programática, oferecendo um ambiente robusto para manipular e analisar esses dados. Ele não só armazena os dados, mas também fornece uma vasta gama de métodos para filtragem, agrupamento, agregação e muito mais.

Características do DataFrame

- Estrutura bidimensional (linhas e colunas)
- Colunas com tipos de dados heterogêneos
- Índice compartilhado entre todas as colunas
- Operações vetorizadas de alto desempenho

A beleza do DataFrame reside na sua capacidade de lidar com dados heterogêneos – ou seja, colunas podem ter diferentes tipos de dados (números, texto, datas) – e na sua otimização para operações vetorizadas, o que significa que você pode aplicar funções a colunas inteiras de uma vez, sem precisar de loops explícitos. É a ferramenta definitiva para quem busca eficiência e flexibilidade na análise de dados tabulares.

```
import pandas as pd

# Criando um DataFrame a partir de um dicionário
dados = {
    'Nome': ['Alice', 'Bob', 'Carlos', 'Diana'],
    'Idade': [25, 30, 22, 28],
    'Cidade': ['São Paulo', 'Rio de Janeiro',
               'Belo Horizonte', 'São Paulo']
}

df = pd.DataFrame(dados)
print("Exemplo de DataFrame:\n", df)
```



Criando DataFrames – Parte 1: Dicionários e Listas

Uma das primeiras tarefas ao trabalhar com o Pandas é criar um DataFrame. Embora muitas vezes importemos dados de arquivos externos, é fundamental saber como construir um DataFrame a partir de estruturas de dados Python já existentes, como dicionários e listas. Essa habilidade é particularmente útil para testes, prototipagem ou quando você tem dados menores que foram gerados diretamente no seu código.

Dicionário de Listas

Chaves = nomes das colunas

Valores = dados das colunas

Lista de Dicionários

Cada dicionário = uma linha

Chaves = nomes das colunas

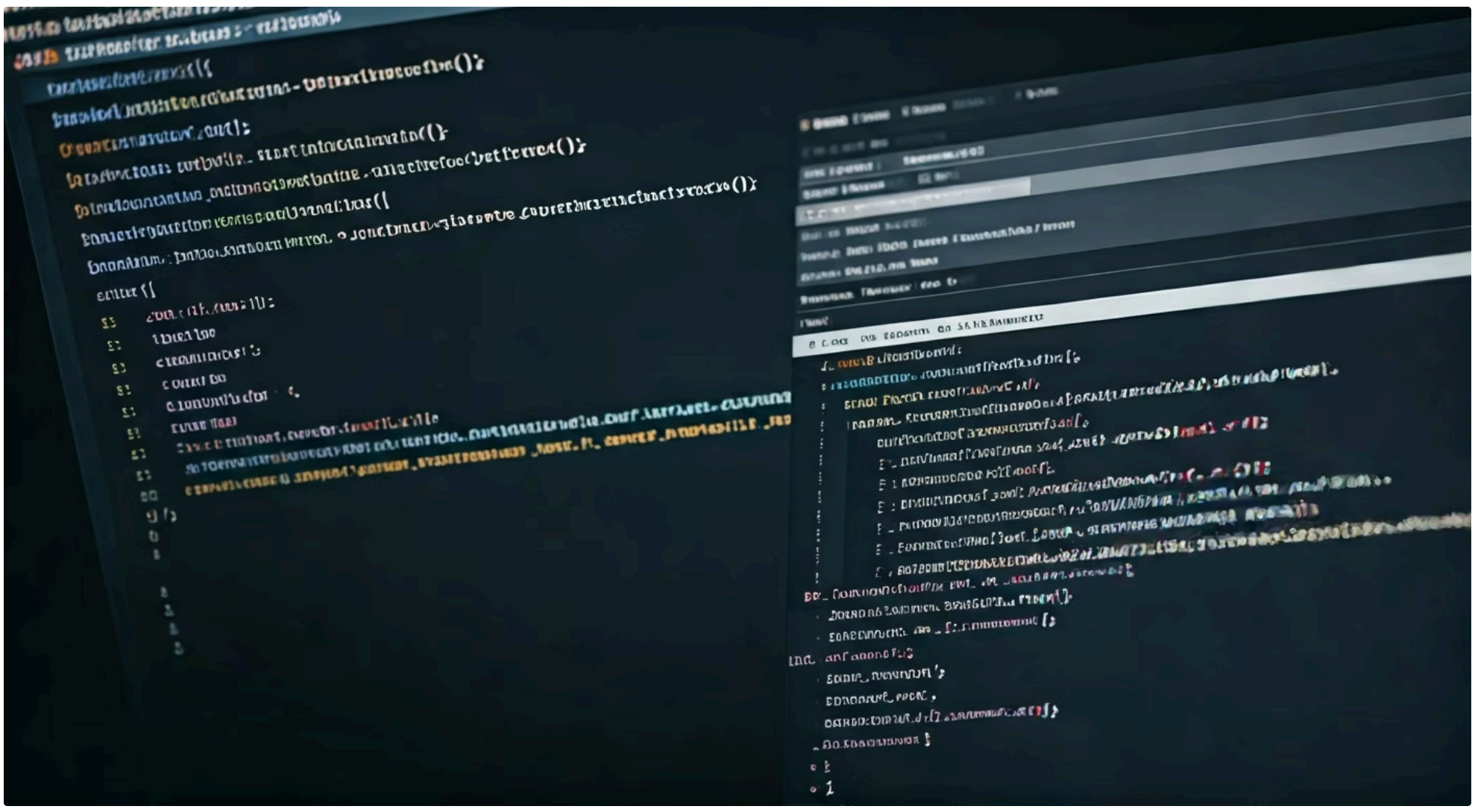
A forma mais comum e intuitiva de criar um DataFrame é a partir de um dicionário de listas. Nesse cenário, cada chave do dicionário se torna o nome de uma coluna, e a lista associada a essa chave se torna os valores dessa coluna. É como se você estivesse definindo cada coluna separadamente e depois as unindo para formar a tabela. Outra abordagem é usar uma lista de dicionários, onde cada dicionário representa uma linha do DataFrame, com as chaves sendo os nomes das colunas e os valores sendo os dados daquela linha.

Método 1: Dicionário de Listas

```
import pandas as pd

# As chaves viram nomes de colunas
# Os valores (listas) viram os dados
dados_dicionario = {
    'Produto': ['Notebook', 'Mouse',
                'Teclado', 'Monitor'],
    'Preço': [3500, 150, 280, 1200],
    'Estoque': [10, 50, 30, 5]
}

df_produtos = pd.DataFrame(dados_dicionario)
print(df_produtos)
```

Método 2: Lista de Dicionários

```
import pandas as pd

# Cada dicionário representa uma linha
dados_lista = [
    {'Nome': 'Ana', 'Idade': 28,
     'Cidade': 'Curitiba'},
    {'Nome': 'Bruno', 'Idade': 35,
     'Cidade': 'Porto Alegre'},
    {'Nome': 'Carla', 'Idade': 22,
     'Cidade': 'Florianópolis'}
]

df_clientes = pd.DataFrame(dados_lista)
print(df_clientes)
```



Esses métodos oferecem flexibilidade para estruturar seus dados antes de passá-los para o Pandas. Compreender como esses dados são mapeados para as colunas e linhas do DataFrame é crucial para garantir que sua estrutura de dados inicial esteja correta e que o DataFrame seja criado conforme o esperado. É a base para transformar dados brutos em uma estrutura organizada e pronta para análise.

Criando DataFrames – Parte 2: Arquivos Externos

No dia a dia da análise de dados, raramente começamos com dados já estruturados em dicionários ou listas dentro do nosso código. A realidade é que a maioria dos dados que vamos analisar reside em arquivos externos, como CSV (Comma Separated Values), Excel, bancos de dados ou APIs. O Pandas é excepcionalmente bom em ler esses diferentes formatos, tornando a importação de dados uma tarefa simples e direta.

CSV

Formato mais comum, leve e universal para dados tabulares

`pd.read_csv()`

Excel

Planilhas com múltiplas abas e formatação

`pd.read_excel()`

JSON

Dados estruturados em formato de objeto JavaScript

`pd.read_json()`

SQL

Consultas diretas a bancos de dados relacionais

`pd.read_sql()`

A capacidade de importar dados de arquivos é um dos pilares do Pandas, pois conecta seu ambiente de programação com o vasto mundo de dados disponíveis. Funções como pd.read_csv() e pd.read_excel() são suas portas de entrada para esses dados. Elas não apenas carregam o conteúdo do arquivo, mas também tentam inferir os tipos de dados de cada coluna e estruturá-los automaticamente em um DataFrame, economizando um tempo valioso.

Dica para Google Colab

No Google Colab, você pode fazer upload de arquivos diretamente ou montar seu Google Drive para acessar seus dados. Use o ícone de pasta na barra lateral ou o código: `from google.colab import drive; drive.mount('/content/drive')`

```
import pandas as pd

# Exemplo de leitura de um arquivo CSV
# df_csv = pd.read_csv('dados.csv')
# print("DataFrame lido de CSV:\n", df_csv.head())

# Exemplo de leitura de um arquivo Excel
# df_excel = pd.read_excel('dados.xlsx', sheet_name='Planilha1')
# print("\nDataFrame lido de Excel:\n", df_excel.head())

# Para fins de demonstração sem arquivo real
print("Para importar de arquivos, usaríamos:")
print("df = pd.read_csv('caminho/do/seu/arquivo.csv')")
print("No Google Colab, você pode fazer upload ou montar o Drive.")

# Criando um DataFrame de exemplo para continuar
dados_simulados = {
    'ID': [1, 2, 3, 4, 5],
    'Nome': ['João', 'Maria', 'Pedro', 'Ana', 'Luiza'],
    'Idade': [28, 32, 25, 30, 29],
    'Salário': [5000, 6500, 4800, 7000, 5500]
}
df_exemplo = pd.DataFrame(dados_simulados)
print("\nDataFrame de exemplo:\n", df_exemplo)
```

É importante notar que, ao importar dados, você pode encontrar desafios como dados ausentes, formatos inconsistentes ou codificações de caracteres diferentes. O Pandas oferece muitos parâmetros para essas funções de leitura que permitem lidar com essas situações já no momento da importação, o que é um tópico que exploraremos em aulas futuras. Por enquanto, o foco é entender como trazer esses dados para o seu ambiente de trabalho no Jupyter Notebook ou Google Colab, que são os ambientes interativos padrão da indústria para esse tipo de tarefa.

Inspeção Inicial de um DataFrame: .head() e .tail()

Depois de carregar seus dados em um DataFrame, a primeira coisa que você vai querer fazer é dar uma olhada rápida para ter uma ideia do que está ali. Conjuntos de dados podem ter milhares ou milhões de linhas, e imprimir o DataFrame inteiro seria impraticável e sobrecarregaria seu ambiente. É aqui que os métodos .head() e .tail() se tornam seus melhores amigos.

.head(n)

Retorna as **primeiras N linhas** do DataFrame (padrão: 5)

Perfeito para uma "espiada" inicial nos dados

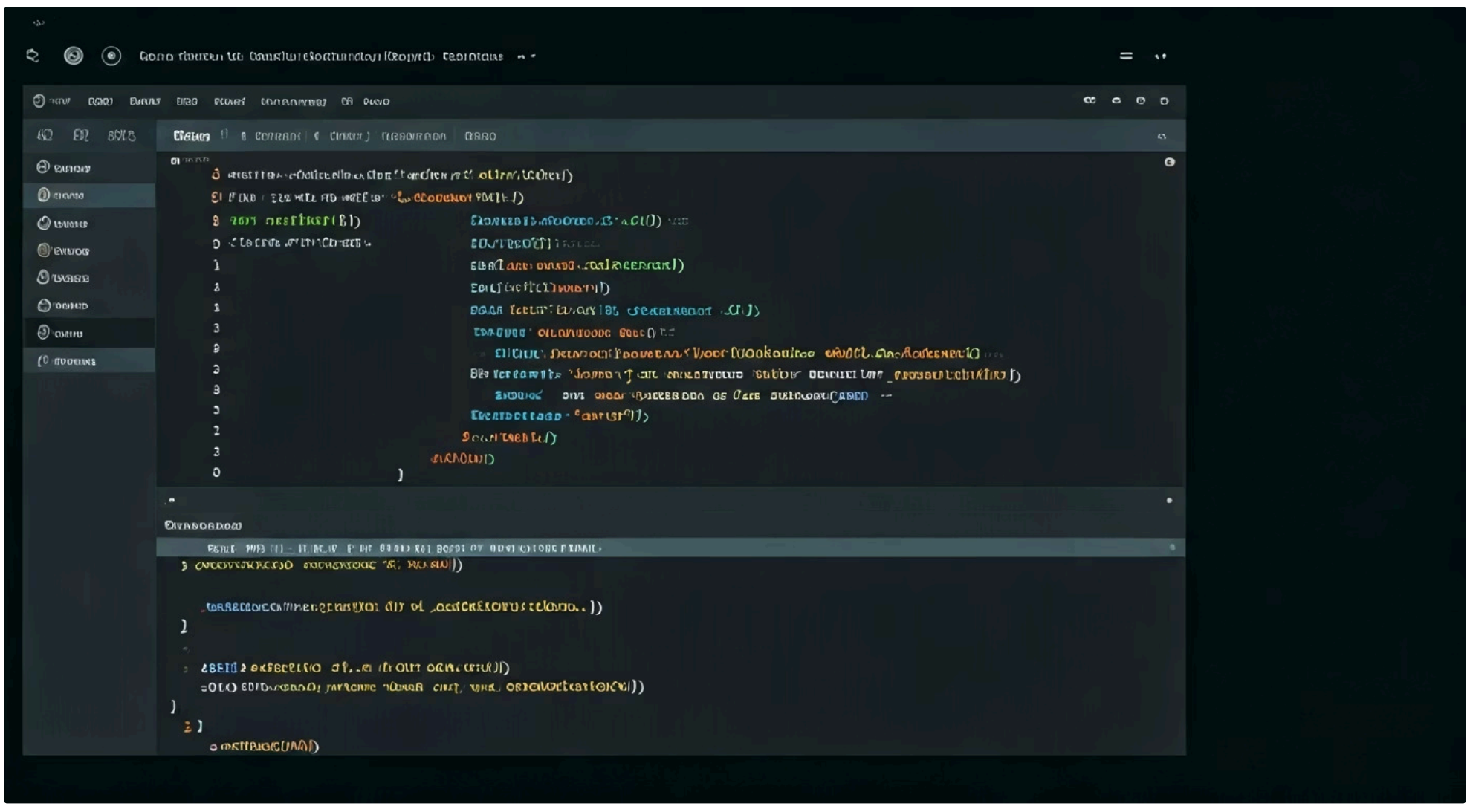
.tail(n)

Retorna as **últimas N linhas** do DataFrame (padrão: 5)

Útil para verificar o final do conjunto de dados

O método .head() retorna as primeiras N linhas do seu DataFrame (por padrão, as 5 primeiras). Ele é perfeito para uma "espiada" inicial, permitindo que você veja os nomes das colunas, os tipos de dados aparentes e a estrutura geral dos seus dados. É como abrir um livro e ler as primeiras páginas para ter uma ideia da história. Da mesma forma, .tail() retorna as últimas N linhas, o que pode ser útil para verificar se os dados foram carregados completamente ou se há algum padrão específico no final do conjunto de dados.

```
import pandas as pd

# Usando o DataFrame de exemplo
dados_simulados = {
    'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Nome': ['João', 'Maria', 'Pedro', 'Ana', 'Luiza',
             'Carlos', 'Fernanda', 'Gustavo', 'Helena', 'Igor'],
    'Idade': [28, 32, 25, 30, 29, 35, 27, 31, 26, 33],
    'Salário': [5000, 6500, 4800, 7000, 5500,
                6000, 5200, 7200, 4900, 6800]
}
df_exemplo = pd.DataFrame(dados_simulados)

print("Primeiras 3 linhas do DataFrame (df.head(3)):\n",
      df_exemplo.head(3))

print("\nÚltimas 2 linhas do DataFrame (df.tail(2)):\n",
      df_exemplo.tail(2))
```



Esses métodos são cruciais para a fase de Exploração de Dados (EDA - Exploratory Data Analysis), pois fornecem uma visão rápida sem sobrecarregar a memória ou a tela. Eles ajudam a confirmar que a importação ocorreu corretamente e que os dados estão no formato esperado, antes de você se aprofundar em análises mais complexas.

Inspeção Inicial de um DataFrame: .info() e .describe()

Além de ver as primeiras e últimas linhas, é fundamental entender a "saúde" e a estrutura subjacente do seu DataFrame. Os métodos .info() e .describe() são ferramentas indispensáveis para essa inspeção mais aprofundada, fornecendo um panorama técnico e estatístico dos seus dados. Eles vão muito além da visualização superficial, revelando informações cruciais para as próximas etapas da análise.

.info() - Raio-X do DataFrame

- Número total de entradas (linhas)
- Número de colunas
- Tipo de dado de cada coluna (Dtype)
- Contagem de valores não nulos
- Uso de memória

Vital para identificar problemas como valores ausentes ou tipos de dados incorretos.

.describe() - Estatísticas Descritivas

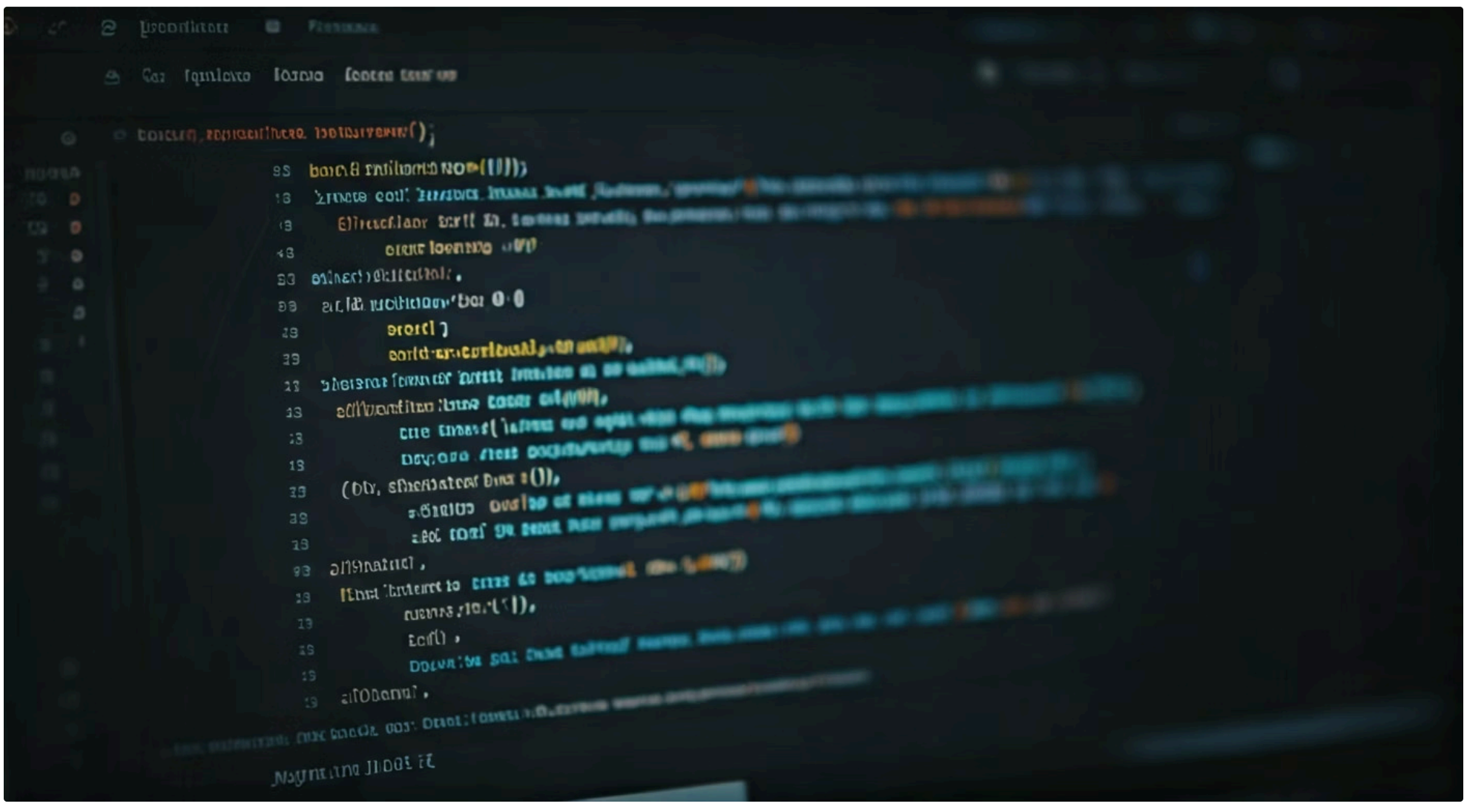
- Contagem de valores
- Média (mean)
- Desvio padrão (std)
- Valor mínimo (min)
- Quartis (25%, 50%, 75%)
- Valor máximo (max)

Poderoso para entender a distribuição e identificar outliers.

O método .info() oferece um resumo conciso do DataFrame, incluindo o número de entradas (linhas), o número de colunas, o tipo de dado de cada coluna (Dtype), a contagem de valores não nulos por coluna e o uso de memória. Essa informação é vital para identificar problemas como colunas com muitos valores ausentes (que podem precisar de tratamento) ou tipos de dados incorretos (por exemplo, números lidos como texto). É como ter um raio-X do seu conjunto de dados.

Já o .describe() gera estatísticas descritivas para as colunas numéricas do DataFrame. Ele calcula a contagem, média, desvio padrão, valor mínimo, quartis (25%, 50% - mediana, 75%) e valor máximo. Essas métricas são poderosas para entender a distribuição dos seus dados, identificar outliers ou verificar se os valores estão dentro de um intervalo esperado. Juntos, .info() e .describe() formam um par imbatível para a primeira análise de qualquer conjunto de dados.

```
import pandas as pd

dados_simulados = {
    'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Nome': ['João', 'Maria', 'Pedro', 'Ana', 'Luiza',
             'Carlos', 'Fernanda', 'Gustavo', 'Helena', 'Igor'],
    'Idade': [28, 32, 25, 30, 29, 35, 27, 31, 26, 33],
    'Salário': [5000, 6500, 4800, 7000, 5500,
                6000, 5200, 7200, 4900, 6800],
    'Ativo': [True, True, False, True, True,
              False, True, True, False, True]
}
df_exemplo = pd.DataFrame(dados_simulados)

print("Informações gerais do DataFrame (df.info()):")
df_exemplo.info()

print("\nEstatísticas descritivas (df.describe()):")
print(df_exemplo.describe())
```



A Importância do Índice (Index) no Pandas

Você já deve ter notado uma coluna à esquerda dos seus DataFrames e Series que não tem um nome explícito, geralmente começando de 0 e incrementando. Essa é a coluna de índice, e ela é muito mais do que apenas uma numeração de linhas. O índice é um dos conceitos mais poderosos e, por vezes, subestimados do Pandas, atuando como um sistema de rótulos para as linhas do seu DataFrame, similar aos rótulos que vimos nas Series.

Sistema de Rótulos

Identifica unicamente cada linha do DataFrame com um valor significativo

Alinhamento de Dados

Permite combinar DataFrames de forma inteligente, mesmo em ordens diferentes

Aceleração de Buscas

Melhora o desempenho de seleção e filtragem de dados

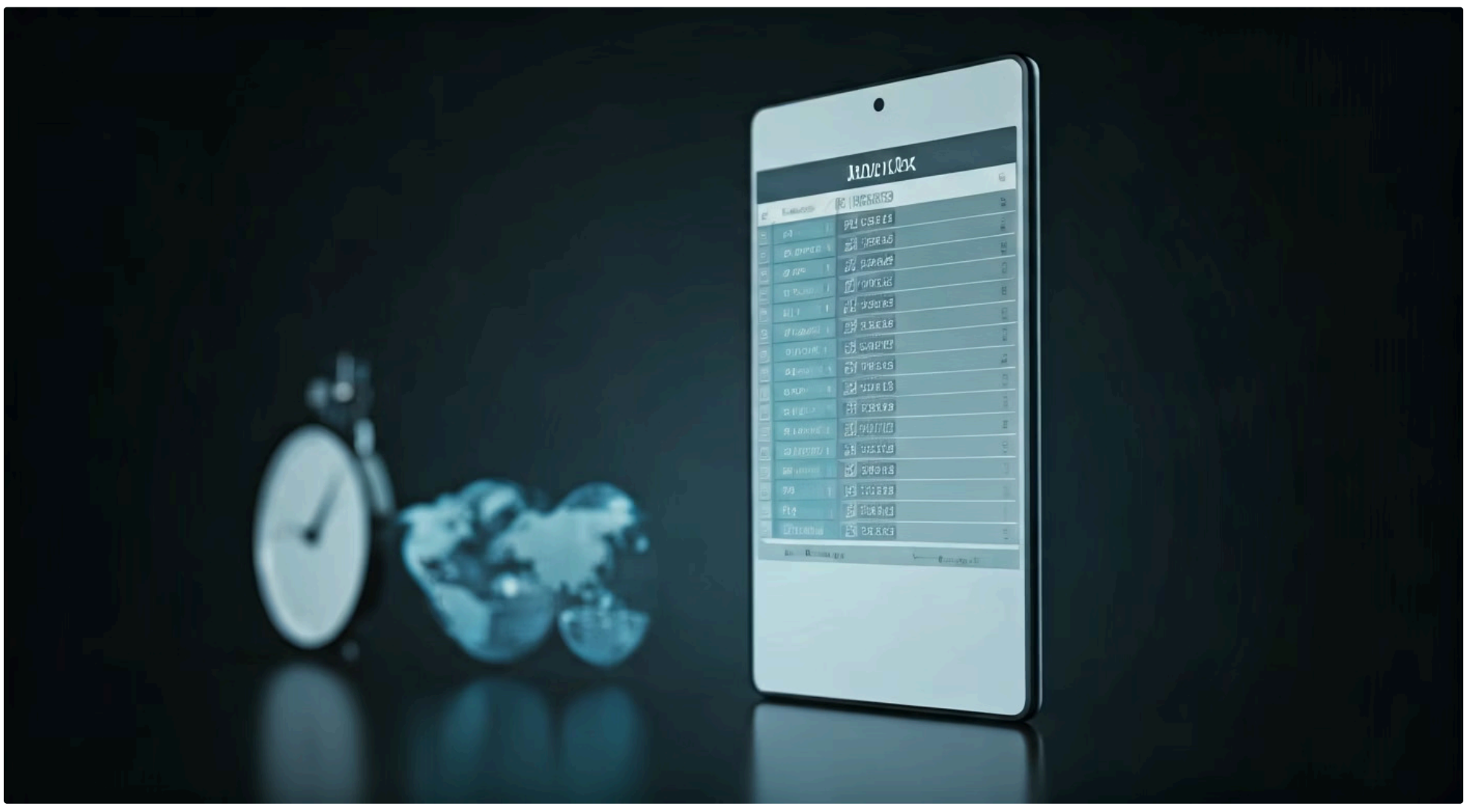
Pense no índice como a "chave primária" da sua tabela, mas com uma flexibilidade muito maior. Ele permite que o Pandas realize operações de alinhamento de dados de forma eficiente, o que é crucial quando você está combinando ou comparando diferentes DataFrames. Se você tem dois DataFrames com informações sobre os mesmos clientes, mas em ordens diferentes, o Pandas pode usar o índice (por exemplo, o ID do cliente) para garantir que as informações sejam combinadas corretamente, sem a necessidade de ordenação manual.

```
import pandas as pd

dados_vendas = {
    'ID_Venda': [101, 102, 103, 104, 105],
    'Produto': ['Celular', 'Tablet', 'Fone',
                'Smartwatch', 'Notebook'],
    'Valor': [1500, 800, 200, 600, 3000],
    'Data': ['2023-01-10', '2023-01-11',
             '2023-01-10', '2023-01-12',
             '2023-01-13']
}
df_vendas = pd.DataFrame(dados_vendas)
print("DataFrame com índice padrão:\n", df_vendas)

# Definindo 'ID_Venda' como índice
df_vendas_com_indice = df_vendas.set_index('ID_Venda')
print("\nDataFrame com 'ID_Venda' como índice:\n",
      df_vendas_com_indice)

# Acessando uma linha pelo novo índice
print("\nDetalhes da Venda 103:\n",
      df_vendas_com_indice.loc[103])
```

Quando Usar Índice Personalizado?

Use quando tiver uma coluna com valores únicos e significativos, como:

- IDs de transações
- Códigos de produtos
- Datas/timestamps
- Identificadores únicos



Além disso, um índice bem escolhido pode acelerar a seleção e filtragem de dados. Embora o Pandas atribua um índice numérico padrão (0, 1, 2...), você pode definir qualquer coluna como índice, desde que ela contenha valores únicos e significativos para o seu contexto. Por exemplo, em um DataFrame de vendas, o ID da transação ou a data da venda podem ser índices muito mais úteis do que um simples número sequencial. Compreender e utilizar o índice de forma eficaz é um diferencial para otimizar suas análises.

Operações Básicas com DataFrames:
Seleção de Colunas

Uma das operações mais frequentes e essenciais ao trabalhar com DataFrames é a seleção de colunas. Raramente precisamos de todas as colunas de um conjunto de dados para uma análise específica. Muitas vezes, queremos focar em um subconjunto de informações, como apenas os nomes dos clientes e seus e-mails, ou os valores de vendas e as datas. O Pandas oferece maneiras simples e intuitivas de fazer isso, permitindo que você extraia exatamente o que precisa.

Seleção Única

`df['Coluna']` retorna uma **Series**

Seleção Múltipla

`df[['Col1', 'Col2']]` retorna um **DataFrame**

Quando você seleciona uma única coluna de um DataFrame, o Pandas a retorna como uma Series. Isso faz sentido, pois uma coluna é, por definição, um conjunto unidimensional de dados com um rótulo (o nome da coluna) e um índice (o mesmo índice do DataFrame). Já se você precisar de várias colunas, o Pandas permite que você passe uma lista de nomes de colunas, e o resultado será um novo DataFrame contendo apenas as colunas especificadas, mantendo a estrutura tabular.

```
import pandas as pd

dados_clientes = {
    'ID': [1, 2, 3, 4, 5],
    'Nome': ['Alice', 'Bob', 'Carlos', 'Diana', 'Eduardo'],
    'Idade': [25, 30, 22, 28, 35],
    'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte',
               'São Paulo', 'Curitiba'],
    'Email': ['alice@email.com', 'bob@email.com',
              'carlos@email.com', 'diana@email.com',
              'eduardo@email.com']
}
df_clientes = pd.DataFrame(dados_clientes)
print("DataFrame original de clientes:\n", df_clientes)

# Selecionando uma única coluna (retorna uma Series)
nomes = df_clientes['Nome']
print("\nColuna 'Nome' (como Series):\n", nomes)

# Selecionando múltiplas colunas (retorna um novo DataFrame)
contatos = df_clientes[['Nome', 'Email', 'Cidade']]
print("\nColunas 'Nome', 'Email' e 'Cidade' (como DataFrame):\n",
      contatos)
```

Essa capacidade de focar em colunas específicas é como ter um filtro em uma planilha, onde você pode ocultar temporariamente as colunas que não são relevantes no momento. É uma etapa crucial na preparação dos dados, pois reduz a complexidade e o volume de informações com os quais você está trabalhando, tornando suas análises mais claras e eficientes.

Operações Básicas com DataFrames:
Filtragem de Linhas

Depois de selecionar as colunas que interessam, o próximo passo lógico é filtrar as linhas. A filtragem permite que você selecione apenas os registros que atendem a uma ou mais condições específicas, eliminando o ruído e focando nos dados que são realmente relevantes para sua análise. Seja para encontrar todos os clientes de uma determinada cidade, todas as vendas acima de um certo valor, ou todos os produtos com estoque baixo, a filtragem é uma habilidade indispensável.

Condição Simples

`df[df['Coluna'] > valor]`

Múltiplas Condições (E)

`df[(cond1) & (cond2)]`

Múltiplas Condições (OU)

`df[(cond1) | (cond2)]`

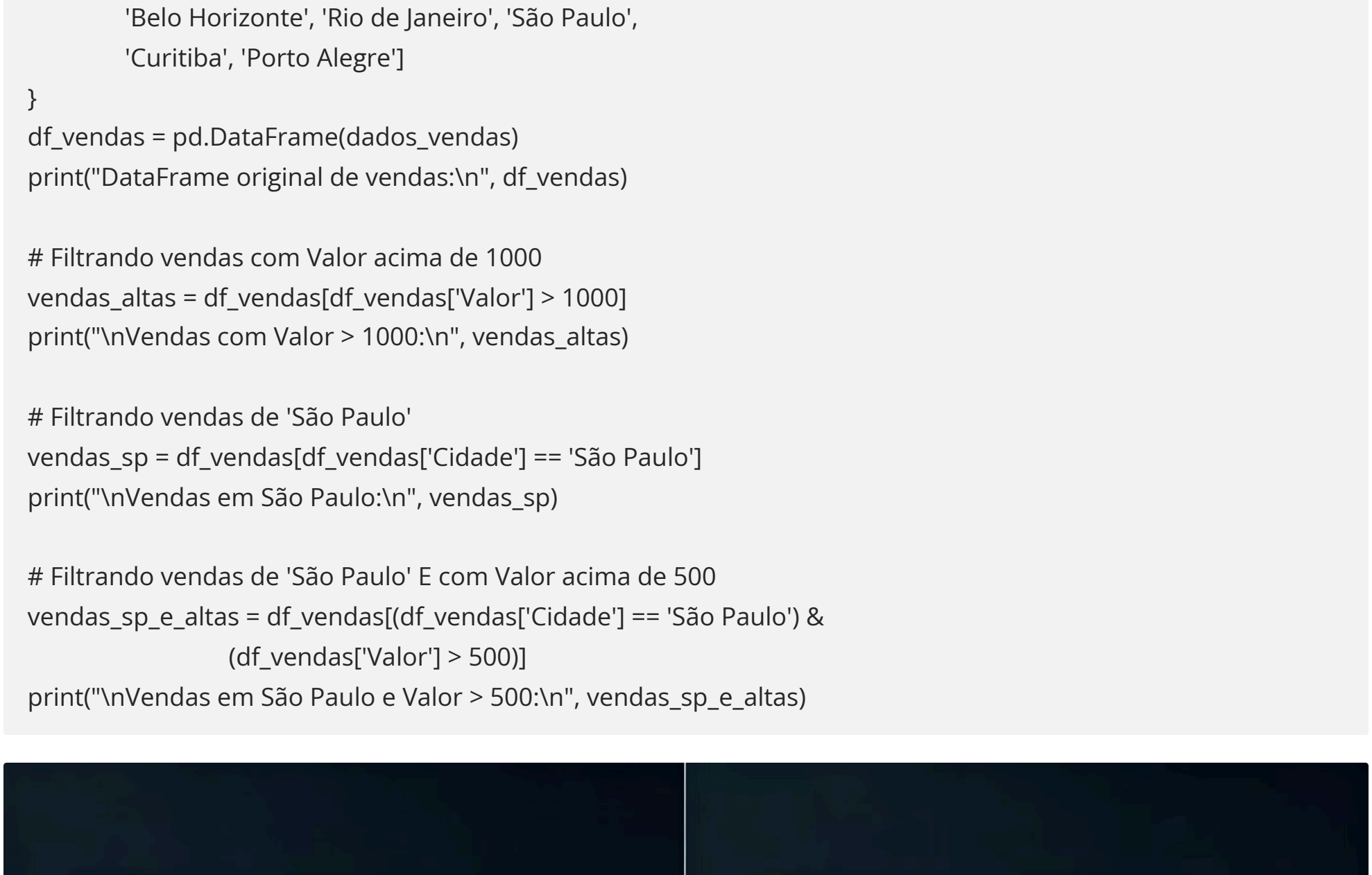
O Pandas torna a filtragem de linhas incrivelmente intuitiva, utilizando o que chamamos de "indexação booleana". Você cria uma condição que, quando aplicada a uma coluna (ou a múltiplas colunas), retorna uma Series de valores True ou False. Essa Series booleana é então usada para "mascarar" o DataFrame, selecionando apenas as linhas onde a condição é True. É como usar um filtro de café: você passa todos os dados, mas retém apenas o que atende ao seu critério.

Operadores Lógicos

- `&` - E lógico (ambas condições devem ser True)
- `|` - OU lógico (pelo menos uma condição deve ser True)
- `~` - NÃO lógico (inverte a condição)

Importante: Use parênteses ao combinar condições!

```
import pandas as pd

dados_vendas = {
    'ID_Venda': [101, 102, 103, 104, 105, 106, 107, 108],
    'Produto': ['Celular', 'Tablet', 'Fone', 'Smartwatch',
                'Notebook', 'Câmera', 'Monitor', 'Teclado'],
    'Valor': [1500, 800, 200, 600, 3000, 1200, 900, 250],
    'Quantidade': [1, 2, 5, 1, 1, 1, 2, 3],
    'Cidade': ['São Paulo', 'Rio de Janeiro', 'São Paulo',
               'Belo Horizonte', 'Rio de Janeiro', 'São Paulo',
               'Curitiba', 'Porto Alegre']
}
df_vendas = pd.DataFrame(dados_vendas)
print("DataFrame original de vendas:\n", df_vendas)

# Filtrando vendas com Valor acima de 1000
vendas_altas = df_vendas[df_vendas['Valor'] > 1000]
print("\nVendas com Valor > 1000:\n", vendas_altas)

# Filtrando vendas de 'São Paulo'
vendas_sp = df_vendas[df_vendas['Cidade'] == 'São Paulo']
print("\nVendas em São Paulo:\n", vendas_sp)

# Filtrando vendas de 'São Paulo' E com Valor acima de 500
vendas_sp_e_altas = df_vendas[(df_vendas['Cidade'] == 'São Paulo') &
                              (df_vendas['Valor'] > 500)]
print("\nVendas em São Paulo e Valor > 500:\n", vendas_sp_e_altas)
```



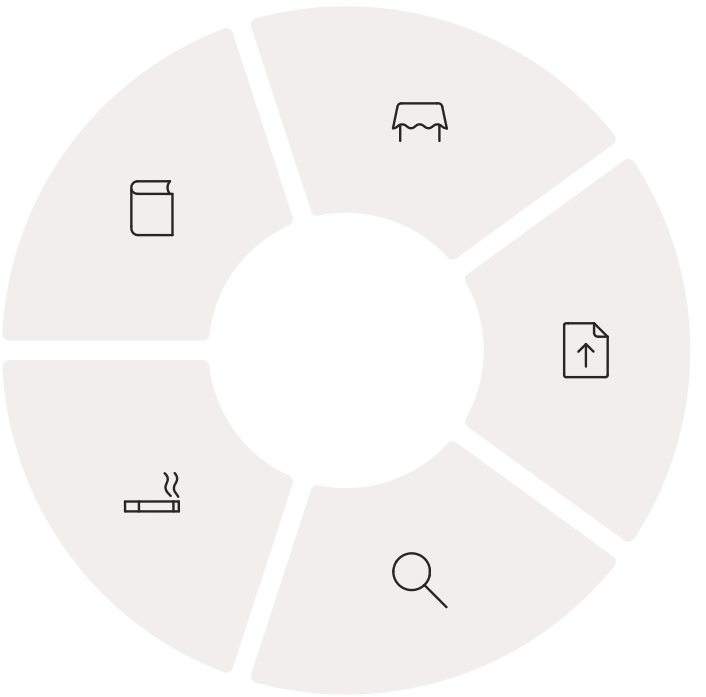
A beleza dessa abordagem é que você pode combinar múltiplas condições usando operadores lógicos (& para "e", | para "ou", ~ para "não"), criando filtros complexos e precisos. Dominar a filtragem é um passo fundamental para transformar grandes volumes de dados em subconjuntos gerenciáveis e significativos, permitindo que você responda a perguntas específicas com facilidade e rapidez.

Consolidação: Dominando as Estruturas Fundamentais do Pandas

Chegamos ao fim de nossa exploração das estruturas de dados fundamentais do Pandas. Nesta aula, desvendamos a importância do Pandas como a principal ferramenta para manipulação de dados em Python, essencial para qualquer aspirante a cientista de dados ou analista. Começamos com a Series, o array unidimensional com rótulos, que serve como o bloco de construção para a estrutura mais complexa. Em seguida, mergulhamos no DataFrame, a tabela bidimensional que se assemelha a uma planilha ou banco de dados, e que é o coração da análise de dados com Pandas.

Series

Array unidimensional com índices personalizados



DataFrame

Tabela bidimensional com colunas e linhas

Importação

Leitura de CSV, Excel e outros formatos

Seleção

Colunas e filtragem de linhas

Inspeção

.head(), .tail(), .info(), .describe()

Aprendemos a criar DataFrames a partir de dicionários, listas e, crucialmente, a partir de arquivos externos, que é a realidade da maioria dos projetos. Vimos como inspecionar rapidamente nossos dados usando .head(), .tail(), .info() e .describe(), ferramentas indispensáveis para entender a estrutura e a qualidade dos dados. Finalmente, exploramos as operações básicas de seleção de colunas e filtragem de linhas, que nos permitem focar nas informações mais relevantes.

Em Prática

Com o conhecimento adquirido, você agora pode:

- Carregar um conjunto de dados de diferentes fontes
- Dar uma olhada rápida em sua estrutura
- Verificar tipos de dados e estatísticas básicas
- Extrair subconjuntos de dados que atendam a critérios específicos

Essas habilidades são a base para qualquer análise de dados mais avançada e o ponto de partida para transformar dados brutos em insights valiosos.

Autoavaliação

1

Qual das seguintes afirmações melhor descreve uma Series no Pandas?

1. Uma tabela bidimensional com colunas e linhas.
2. Um array unidimensional com rótulos (índice).
3. Uma coleção de DataFrames.
4. Um tipo de dado para armazenar apenas números inteiros.

2

Para que serve o método .info() em um DataFrame?

1. Para exibir as primeiras 5 linhas do DataFrame.
2. Para calcular estatísticas descritivas das colunas numéricas.
3. Para obter um resumo conciso do DataFrame, incluindo tipos de dados e valores não nulos.
4. Para exportar o DataFrame para um arquivo CSV.

3

Você precisa selecionar as colunas 'Nome' e 'Idade' de um DataFrame chamado df_alunos. Qual é a sintaxe correta?

1. df_alunos['Nome', 'Idade']
2. df_alunos.select('Nome', 'Idade')
3. df_alunos[['Nome', 'Idade']]
4. df_alunos.columns('Nome', 'Idade')

4

Qual método você usaria para filtrar um DataFrame df_produtos para mostrar apenas os produtos onde o 'Estoque' é menor que 10?

1. df_produtos.filter_by_stock(10)
2. df_produtos[df_produtos['Estoque'] < 10]
3. df_produtos.query('Estoque < 10')
4. df_produtos.get_rows(stock_less_than=10)

Gabarito

1. b) | **2.** c) | **3.** c) | **4.** b)

Questão Discursiva

> Explique a diferença fundamental entre uma Series e um DataFrame no Pandas, e em que situações cada estrutura seria mais apropriada para representar um conjunto de dados.

Continue sua jornada

Próxima Aula

Na **Aula 6 – Importação e Exportação de Dados**, aprofundaremos nas técnicas de leitura e gravação de dados em diferentes formatos, como CSV, Excel e JSON, e exploraremos as melhores práticas para lidar com dados em ambientes de produção.

Recursos Adicionais

- **Documentação Oficial do Pandas**: Para detalhes técnicos e exemplos avançados.
- **Livro "Python for Data Analysis" (Wes McKinney)**: O criador do Pandas, com insights aprofundados.
- **Cursos Online (Coursera, Udemy, DataCamp)**: Para prática interativa e projetos guiados.

NOTA IMPORTANTE: As informações técnicas desta aula estão atualizadas até 2025. Consulte sempre a documentação oficial do Pandas para verificar as últimas atualizações e melhores práticas.