Aula 6 – Importação e Exportação de Dados

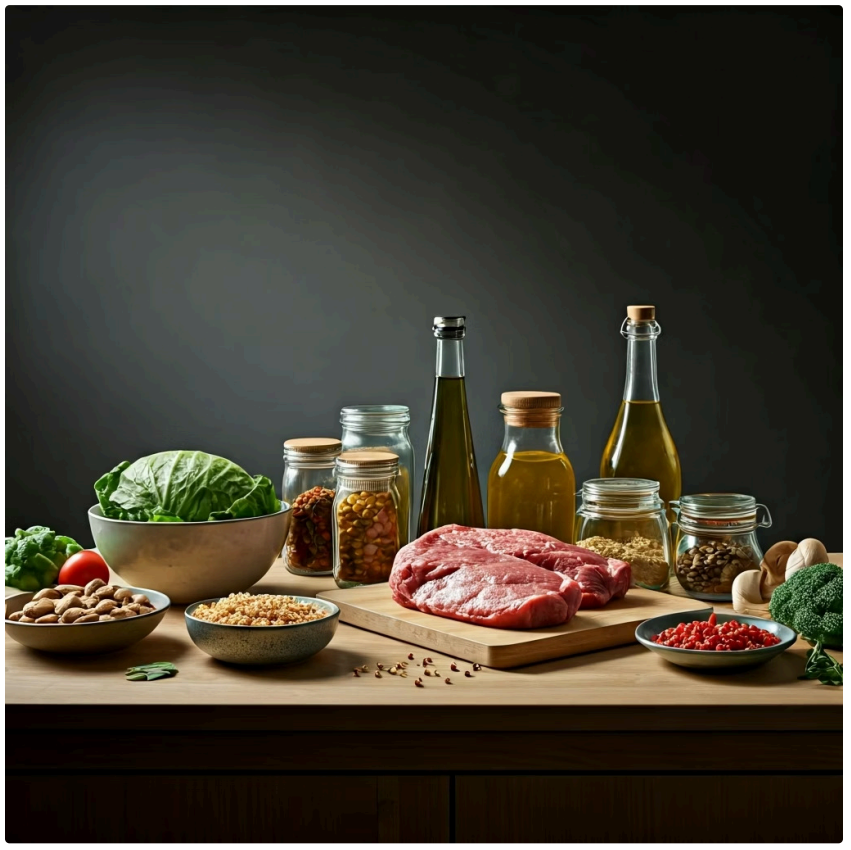
No universo da análise de dados, a primeira e talvez mais fundamental etapa é conseguir acessar as informações que você precisa e, depois de processá-las, compartilhá-las de forma compreensível. Imagine que você é um explorador em busca de tesouros: antes de mapear o terreno ou decifrar enigmas, você precisa primeiro encontrar o mapa e, ao final da jornada, registrar suas descobertas para outros. No mundo digital, esses "mapas" e "registros" são os arquivos de dados, e a habilidade de importá-los e exportá-los é a sua bússola e caneta.

Dominar a importação e exportação de dados com Python não é apenas uma técnica; é a porta de entrada para qualquer projeto de ciência de dados. Sem essa capacidade, seus modelos e análises mais sofisticados seriam como um carro sem combustível, incapazes de sair do lugar. É aqui que a biblioteca Pandas brilha, oferecendo ferramentas robustas e intuitivas para lidar com uma vasta gama de formatos, transformando dados brutos em informações estruturadas prontas para serem exploradas.

Nesta aula, vamos desvendar os segredos por trás da leitura e gravação de dados nos formatos mais comuns do mercado: CSV, Excel e JSON. Você aprenderá a lidar com os parâmetros cruciais que garantem que seus dados sejam interpretados corretamente e, mais importante, como superar os desafios de codificação que frequentemente surgem. Ao final, você estará apto a mover dados entre diferentes sistemas com confiança, pavimentando o caminho para análises mais profundas e eficazes.

O Ponto de Partida: Por Que Precisamos Importar Dados?

Antes de mergulharmos nos comandos e códigos, é essencial entender o "porquê" por trás da importação de dados. Pense na sua rotina diária: você interage com informações de diversas fontes – e-mails, planilhas, notícias online. Cada uma dessas fontes armazena dados de uma maneira específica. Para um analista de dados, essa realidade é amplificada, pois as informações podem vir de bancos de dados, APIs de serviços web, arquivos gerados por softwares específicos ou até mesmo de outros colegas.



O desafio reside em transformar essa miríade de formatos e estruturas em algo coeso e manipulável dentro do seu ambiente de análise. É como ter ingredientes de diferentes prateleiras do supermercado (frutas, legumes, enlatados) e precisar organizá-los na sua cozinha para preparar uma refeição. A importação de dados é exatamente esse processo de "organização inicial", onde trazemos os ingredientes para a nossa bancada de trabalho, o DataFrame do Pandas.

> 💡 **Insight Importante:** A biblioteca Pandas, com sua estrutura de DataFrame, atua como essa bancada de trabalho universal. Ela permite que você leia dados de quase qualquer formato e os transforme em uma tabela padronizada, onde linhas representam observações e colunas representam variáveis. Essa padronização é o que nos permite aplicar as mesmas técnicas de limpeza, transformação e análise, independentemente da origem dos dados. É a base para construir qualquer projeto de análise de dados robusto e escalável.

CSV: O Formato Mais Popular e Seus Segredos

Simplicidade

Arquivo de texto puro com valores separados por delimitador

Portabilidade

Pode ser aberto por qualquer editor de texto

Armadilhas

Separadores variados e configurações regionais diferentes

O formato CSV (Comma Separated Values) é o "pão com manteiga" da análise de dados. Sua simplicidade é a chave de sua popularidade: é um arquivo de texto puro onde cada linha representa um registro e os valores dentro de cada linha são separados por um delimitador, geralmente uma vírgula. É como uma planilha eletrônica simplificada, que pode ser aberta por qualquer editor de texto, tornando-o extremamente portátil e fácil de compartilhar.

No entanto, essa simplicidade esconde algumas armadilhas. Nem todo CSV usa vírgulas como separador; alguns utilizam ponto e vírgula, tabulações ou outros caracteres, especialmente em diferentes configurações regionais. Além disso, a primeira linha pode ou não conter os nomes das colunas, e a forma como os dados são indexados pode variar. É como tentar ler um mapa onde a legenda pode mudar sem aviso prévio, exigindo que você preste atenção aos detalhes.

Para lidar com CSVs em Python, a função `pd.read_csv()` do Pandas é a sua melhor amiga. Ela é poderosa e flexível, permitindo que você especifique esses detalhes cruciais. Vamos ver um exemplo básico e, em seguida, explorar os parâmetros que nos dão controle total sobre o processo de importação.

```
import pandas as pd
# Exemplo básico de leitura de um arquivo CSV
# Suponha que temos um arquivo 'dados_vendas.csv'
# com colunas: Produto, Quantidade, Preco, Data
try:
    df_vendas = pd.read_csv('dados_vendas.csv')
    print("DataFrame de Vendas:")
    print(df_vendas.head())
except FileNotFoundError:
    print("Arquivo 'dados_vendas.csv' não encontrado. Criando um exemplo...")
    # Criando um DataFrame de exemplo para simular o arquivo
    data = {'Produto': ['Notebook', 'Mouse', 'Teclado', 'Monitor'],
            'Quantidade': [2, 5, 3, 1],
            'Preco': [3500.00, 120.00, 250.00, 1500.00],
            'Data': ['2023-01-15', '2023-01-16', '2023-01-15', '2023-01-17']}
    df_vendas = pd.DataFrame(data)
    df_vendas.to_csv('dados_vendas.csv', index=False)  # Salva para uso futuro
    print("Arquivo 'dados_vendas.csv' criado e lido com sucesso.")
    print(df_vendas.head())
```

Este exemplo demonstra a facilidade de carregar um CSV. No entanto, a vida real raramente é tão simples, e é aí que os parâmetros de importação se tornam indispensáveis.

Parâmetros Essenciais na Importação: sep, header e index_col

A flexibilidade do `pd.read_csv()` reside em seus parâmetros, que nos permitem ajustar a leitura do arquivo às suas particularidades. Dominar esses parâmetros é como ter um kit de ferramentas completo para abrir qualquer tipo de "caixa" de dados CSV, garantindo que o conteúdo seja extraído da forma correta.

sep (separador)

Define o caractere que separa os valores. Use `sep=';'` para ponto e vírgula ou `sep='\t'` para tabulação.

header (cabeçalho)

Indica qual linha contém os nomes das colunas. Use `header=None` se não houver cabeçalho ou `header=2` para a terceira linha.

index_col (coluna índice)

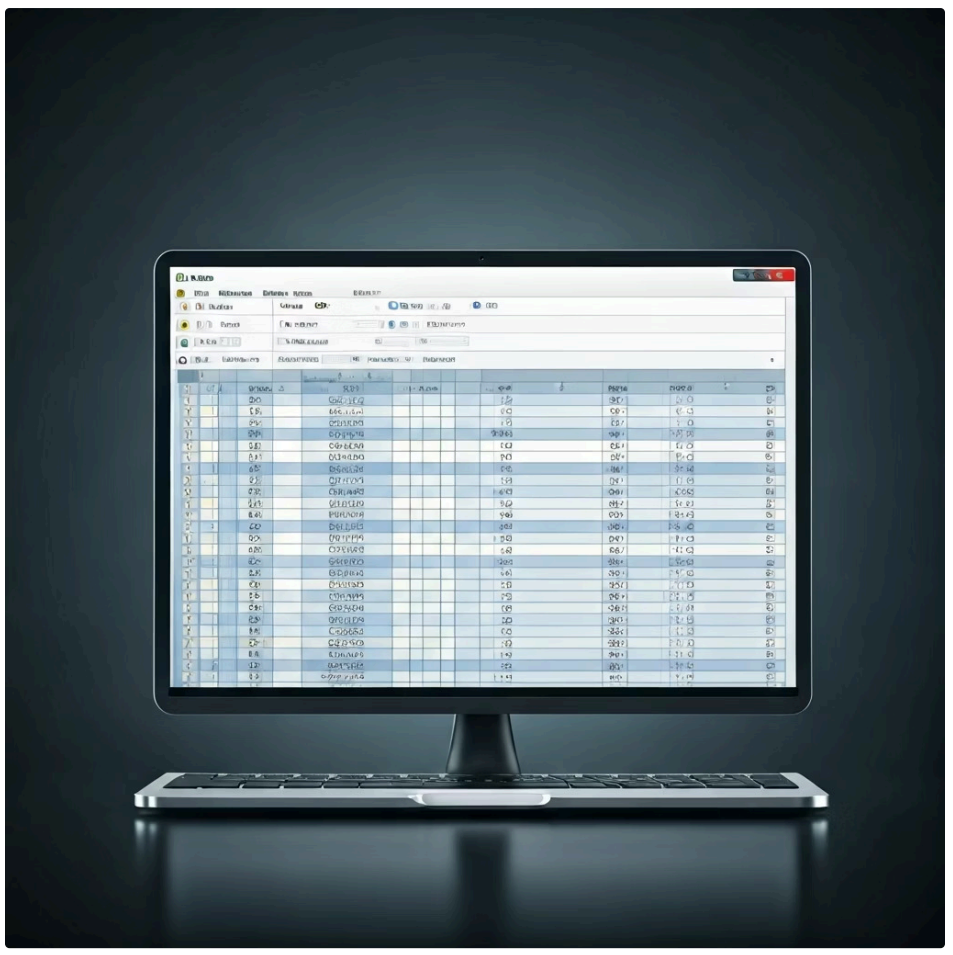
Define uma ou mais colunas como índice do DataFrame. Use `index_col='ID'` ou `index_col=0` para a primeira coluna.

Imagine que você está montando um quebra-cabeça. O `sep` é a forma como as peças são cortadas (vírgula, ponto e vírgula). O `header` é a imagem da caixa, que mostra como as peças devem se encaixar para formar o desenho completo. E o `index_col` é a decisão de usar uma parte específica do desenho como ponto de referência para começar a montagem. Sem essas instruções, o resultado pode ser uma bagunça ilegível.

```
import pandas as pd

# Criando um arquivo CSV de exemplo com separador diferente e sem cabeçalho
data_sem_header = """Notebook;2;3500.00;2023-01-15
Mouse;5;120.00;2023-01-16
Teclado;3;250.00;2023-01-15"""

with open('dados_sem_header.csv', 'w') as f:
    f.write(data_sem_header)

# Lendo o arquivo com separador diferente e sem cabeçalho
print("\nLendo 'dados_sem_header.csv' com sep=';' e header=None:")
df_sem_header = pd.read_csv('dados_sem_header.csv', sep=';', header=None)
print(df_sem_header.head())

# Adicionando nomes de colunas manualmente após a importação
df_sem_header.columns = ['Produto', 'Quantidade', 'Preco', 'Data']
print("\nDataFrame com colunas nomeadas:")
print(df_sem_header.head())

# Criando um arquivo CSV com uma coluna que queremos usar como índice
data_com_index = """ID,Produto,Quantidade,Preco,Data
1,Notebook,2,3500.00,2023-01-15
2,Mouse,5,120.00,2023-01-16
3,Teclado,3,250.00,2023-01-15"""

with open('dados_com_index.csv', 'w') as f:
    f.write(data_com_index)

# Lendo o arquivo usando a coluna 'ID' como índice
print("\nLendo 'dados_com_index.csv' com index_col='ID':")
df_com_index = pd.read_csv('dados_com_index.csv', index_col='ID')
print(df_com_index.head())
```

✅ **Dica Prática:** Esses parâmetros são cruciais para garantir que seus dados sejam carregados corretamente, evitando erros de interpretação que poderiam comprometer toda a sua análise. A prática constante com diferentes tipos de arquivos CSV irá solidificar seu entendimento.

Excel: Lidando com Planilhas Complexas



Arquivos Excel (.xlsx ou .xls) são onipresentes no ambiente corporativo. Eles são mais complexos que CSVs, pois podem conter múltiplas abas (sheets), formatação, fórmulas e até macros. Para o analista de dados, isso significa que, além de importar os dados, é preciso especificar qual aba de trabalho deve ser lida. É como ter um livro com vários capítulos e precisar indicar qual capítulo você quer ler.

A função `pd.read_excel()` do Pandas é a ferramenta para essa tarefa. Ela é robusta o suficiente para lidar com a complexidade dos arquivos Excel, permitindo que você selecione abas específicas pelo nome ou índice, e até mesmo leia apenas um intervalo de células. Isso é particularmente útil quando os dados relevantes estão aninhados em uma parte específica de uma planilha.

Pense em um arquivo Excel como um grande fichário. Cada aba é uma seção diferente, e dentro de cada seção, os dados podem estar organizados de várias maneiras. O `pd.read_excel()` é o seu assistente que sabe exatamente onde procurar a informação que você pediu, sem se perder entre as outras seções ou anotações irrelevantes.

```
import pandas as pd

# Criando um arquivo Excel de exemplo com múltiplas abas
with pd.ExcelWriter('relatorio_vendas.xlsx') as writer:
    df_vendas_jan = pd.DataFrame({
        'Produto': ['Notebook', 'Mouse'],
        'Quantidade': [10, 20],
        'Preco': [3000, 100]
    })
    df_vendas_fev = pd.DataFrame({
        'Produto': ['Teclado', 'Monitor'],
        'Quantidade': [15, 5],
        'Preco': [200, 1200]
    })
    df_vendas_jan.to_excel(writer, sheet_name='Janeiro', index=False)
    df_vendas_fev.to_excel(writer, sheet_name='Fevereiro', index=False)

# Lendo a aba 'Janeiro' do arquivo Excel
print("Lendo a aba 'Janeiro' do 'relatorio_vendas.xlsx':")
df_janeiro = pd.read_excel('relatorio_vendas.xlsx', sheet_name='Janeiro')
print(df_janeiro.head())

# Lendo a aba 'Fevereiro' pelo índice (0 para a primeira, 1 para a segunda)
print("\nLendo a aba 'Fevereiro' (índice 1) do 'relatorio_vendas.xlsx':")
df_fevereiro = pd.read_excel('relatorio_vendas.xlsx', sheet_name=1)
print(df_fevereiro.head())
```

A capacidade de especificar a aba (`sheet_name`) é o principal diferencial ao trabalhar com Excel, permitindo que você extraia exatamente o que precisa de arquivos que podem ser bastante densos e multifacetados.

JSON: Estruturas Aninhadas e Flexibilidade

Formato Moderno

JSON é o padrão para troca de dados em aplicações web, APIs e bancos NoSQL

Estrutura Aninhada

Permite representar dados hierárquicos e complexos de forma natural

Desafio Tabular

Requer normalização para transformar hierarquias em formato de tabela

JSON (JavaScript Object Notation) é um formato de dados leve e de fácil leitura para humanos, mas também facilmente interpretável por máquinas. Ele se tornou o padrão de fato para troca de dados em aplicações web, APIs e bancos de dados NoSQL. Ao contrário dos formatos tabulares como CSV e Excel, o JSON permite estruturas de dados aninhadas, o que significa que um valor pode ser outro objeto ou uma lista de objetos.

Essa flexibilidade é uma bênção e uma maldição. Por um lado, o JSON pode representar dados complexos de forma muito natural. Por outro, a sua estrutura aninhada pode ser um desafio para ser "achatada" em um DataFrame tabular. É como tentar organizar uma árvore genealógica complexa em uma única tabela: você precisa decidir como representar as relações hierárquicas em um formato linear.

A função `pd.read_json()` do Pandas é projetada para lidar com essa complexidade. Ela tenta inferir a estrutura dos dados e, muitas vezes, consegue transformá-los em um DataFrame de forma inteligente. No entanto, para JSONs muito aninhados, pode ser necessário um pré-processamento ou o uso de parâmetros adicionais para "normalizar" os dados.

```
import pandas as pd
import json

# Criando um arquivo JSON de exemplo com estrutura aninhada
json_data = """
[
  {
    "id": "prod001",
    "nome": "Notebook Gamer",
    "detalhes": {
      "marca": "XtremeTech",
      "processador": "Intel i7",
      "ram_gb": 16
    },
    "tags": ["eletronico", "gamer", "promocao"]
  },
  {
    "id": "prod002",
    "nome": "Mouse Sem Fio",
    "detalhes": {
      "marca": "ErgoClick",
      "dpi": 1600
    },
    "tags": ["acessorio", "escritorio"]
  }
]
"""

with open('produtos.json', 'w') as f:
    f.write(json_data)

# Lendo o arquivo JSON
print("Lendo 'produtos.json':")
df_produtos = pd.read_json('produtos.json')
print(df_produtos.head())

# Observe como a coluna 'detalhes' é um dicionário.
# Para acessar os campos aninhados, podemos usar pd.json_normalize() ou expandir manualmente.
print("\nColuna 'detalhes' após leitura:")
print(df_produtos['detalhes'].head())

# Exemplo de normalização para expandir a coluna 'detalhes'
print("\nDataFrame após normalizar a coluna 'detalhes':")
df_produtos_normalizado = pd.json_normalize(json.loads(json_data))
print(df_produtos_normalizado.head())
```

🔧 **Técnica Avançada:** A normalização de JSONs aninhados é uma técnica avançada que transforma estruturas hierárquicas em um formato tabular mais plano, facilitando a análise. É uma habilidade valiosa para quem trabalha com dados de APIs ou bancos de dados NoSQL.

Exportando DataFrames: Compartilhando Suas Descobertas

Depois de todo o trabalho de importação, limpeza e análise, é hora de compartilhar seus resultados. A exportação de DataFrames é tão crucial quanto a importação, pois permite que você salve seus dados processados em formatos que podem ser facilmente consumidos por outras ferramentas, sistemas ou colegas. É como, após cozinhar uma refeição deliciosa, você a serve em pratos adequados para que todos possam desfrutar.

to_csv()

Exporta para formato CSV universal

to_excel()

Cria planilhas Excel formatadas

to_json()

Gera arquivos JSON estruturados

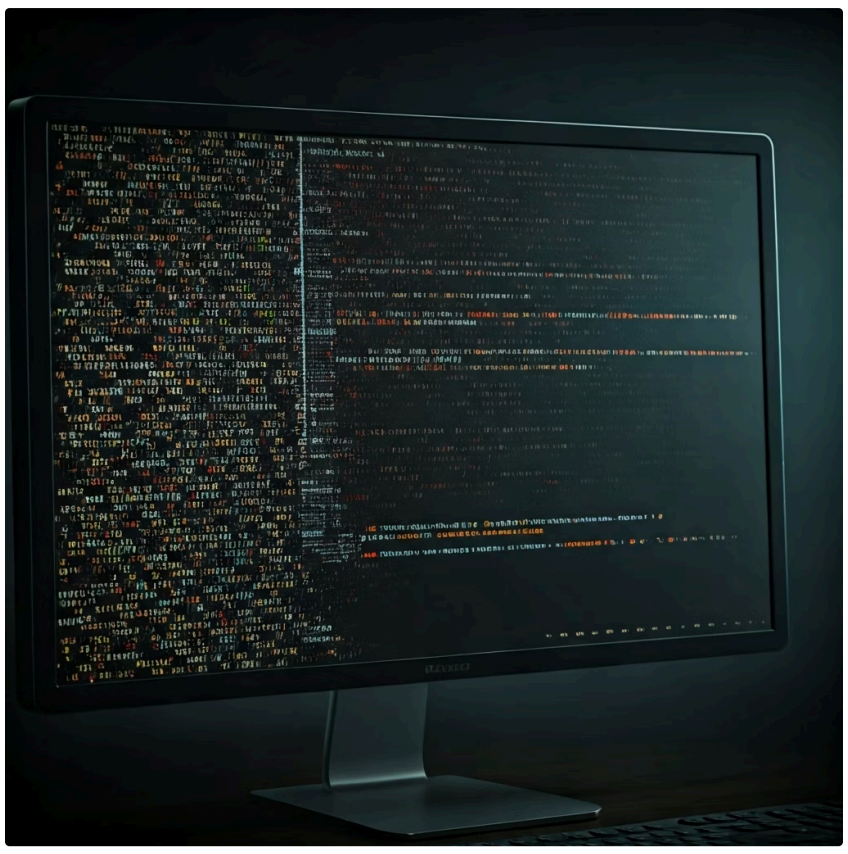
O Pandas oferece métodos intuitivos para exportar DataFrames para os mesmos formatos que você aprendeu a importar: CSV, Excel e JSON. As funções `df.to_csv()`, `df.to_excel()` e `df.to_json()` são os espelhos de suas contrapartes de leitura, permitindo que você grave os dados com controle sobre os detalhes do formato de saída.

Essa capacidade de exportar é vital para a colaboração e para a integração de seus projetos de análise de dados em fluxos de trabalho maiores. Seja para gerar relatórios, alimentar dashboards ou preparar dados para um modelo de machine learning, a exportação é a ponte que conecta sua análise ao mundo exterior.

```
import pandas as pd

# Criando um DataFrame de exemplo para exportação
data = {'ID': [1, 2, 3],
        'Nome': ['Alice', 'Bob', 'Charlie'],
        'Idade': [25, 30, 35],
        'Cidade': ['São Paulo', 'Rio de Janeiro', 'Belo Horizonte']}
df_exemplo = pd.DataFrame(data)

# Exportando para CSV
# O parâmetro index=False evita que o índice do DataFrame seja gravado como uma coluna
df_exemplo.to_csv('dados_exportados.csv', index=False)
print("DataFrame exportado para 'dados_exportados.csv'")

# Exportando para Excel
# sheet_name define o nome da aba no arquivo Excel
df_exemplo.to_excel('dados_exportados.xlsx', sheet_name='Pessoas', index=False)
print("DataFrame exportado para 'dados_exportados.xlsx'")

# Exportando para JSON
# orient='records' é comum para JSON, onde cada linha é um objeto JSON
df_exemplo.to_json('dados_exportados.json', orient='records', indent=4)
print("DataFrame exportado para 'dados_exportados.json'")

# Verificando o conteúdo do CSV exportado
with open('dados_exportados.csv', 'r') as f:
    print("\nConteúdo de 'dados_exportados.csv':")
    print(f.read())
```

💼 **Dica Profissional:** Ao exportar, sempre considere o público-alvo ou o sistema que consumirá o arquivo. Por exemplo, `index=False` é frequentemente usado para CSV e Excel para evitar uma coluna de índice desnecessária, enquanto `orient='records'` é uma boa prática para JSONs que serão lidos por APIs.

Lidando com Problemas de Codificação de Caracteres (Encoding)



O Pesadelo dos Caracteres Estranhos

Um dos pesadelos mais comuns na importação de dados é o erro de codificação de caracteres. Você já abriu um arquivo e viu caracteres estranhos como Ã§ ou Ã£ em vez de ç ou ã? Isso acontece quando o software que lê o arquivo interpreta os bytes de forma diferente de como eles foram gravados. É como tentar ler um livro escrito em um alfabeto que você não conhece, mesmo que as palavras sejam do seu idioma.

UTF-8

O padrão moderno e mais abrangente, suporta todos os caracteres Unicode

latin-1 (ISO-8859-1)

Comum em sistemas legados, especialmente na Europa e América Latina

cp1252

Codificação Windows, variante do latin-1 com caracteres adicionais

A codificação de caracteres é o sistema que mapeia números (bytes) para caracteres visíveis. Os mais comuns são UTF-8 (o padrão moderno e mais abrangente) e latin-1 (também conhecido como ISO-8859-1, comum em sistemas legados, especialmente na Europa e América Latina). Quando há uma incompatibilidade entre a codificação do arquivo e a codificação que o Pandas tenta usar (que por padrão é UTF-8), ocorre o erro.

A solução é simples, mas requer que você saiba (ou adivinhe) a codificação correta do arquivo. O parâmetro `encoding` nas funções de leitura do Pandas permite que você especifique essa codificação. É uma tentativa e erro, mas com algumas codificações comuns, você geralmente acerta.

```
import pandas as pd

# Criando um arquivo CSV com codificação 'latin-1' para simular o problema
data_latin1 = "Nome;Cidade\nJoão;São Paulo\nMaria;Rio de Janeiro\nJosé;Curitiba"
with open('dados_latin1.csv', 'w', encoding='latin-1') as f:
    f.write(data_latin1)

# Tentando ler o arquivo sem especificar a codificação (provavelmente dará erro ou caracteres estranhos)
print("Tentando ler 'dados_latin1.csv' sem especificar encoding:")
try:
    df_erro_encoding = pd.read_csv('dados_latin1.csv', sep=';')
    print(df_erro_encoding)
except UnicodeDecodeError as e:
    print(f"Erro de codificação esperado: {e}")
    print("Caracteres podem aparecer corrompidos se a leitura for forçada.")

# Lendo o arquivo especificando a codificação correta
print("\nLendo 'dados_latin1.csv' com encoding='latin-1':")
df_correto = pd.read_csv('dados_latin1.csv', sep=';', encoding='latin-1')
print(df_correto.head())

# Exemplo de exportação com codificação específica
df_correto.to_csv('dados_utf8.csv', index=False, encoding='utf-8')
print("\nDataFrame exportado para 'dados_utf8.csv' com encoding='utf-8'")
```

🔍 **Estratégia de Resolução:** Sempre que encontrar caracteres ilegíveis, sua primeira ação deve ser tentar diferentes codificações, como `latin-1`, `ISO-8859-1`, `cp1252` ou até mesmo `utf-16`. A prática leva à perfeição em identificar a codificação correta.

Boas Práticas e Dicas para um Fluxo de Trabalho Eficiente

A importação e exportação de dados, embora pareçam tarefas básicas, são a fundação de um fluxo de trabalho de análise de dados robusto. Adotar boas práticas aqui pode economizar horas de depuração e garantir a integridade dos seus dados. Pense nisso como a construção de uma casa: a fundação precisa ser sólida para que toda a estrutura se mantenha de pé.

01

Inspeção Prévia

Sempre inspecione os primeiros registros do arquivo antes de importá-lo. Abra um CSV em um editor de texto ou um Excel no próprio programa para revelar o separador, a presença de cabeçalho e a codificação.

02

Especificação de Tipos

Use o parâmetro `dtype` em `pd.read_csv()` para especificar os tipos de dados das colunas, otimizando a memória e evitando erros de inferência de tipo.

03

Padronização UTF-8

Sempre que possível, padronize seus dados para UTF-8 ao exportar. Este é o padrão da indústria e garante a máxima compatibilidade entre diferentes sistemas e plataformas.

04

Documentação

Documente as decisões de importação/exportação em seu código, especialmente quando usar parâmetros não-padrão, para facilitar a manutenção futura.

Uma boa prática é sempre inspecionar os primeiros registros do arquivo antes de importá-lo, se possível. Abrir um CSV em um editor de texto ou um Excel no próprio programa pode revelar o separador, a presença de cabeçalho e a codificação. Outra dica é usar o parâmetro `dtype` em `pd.read_csv()` para especificar os tipos de dados das colunas, o que pode otimizar a memória e evitar erros de inferência de tipo.

Além disso, sempre que possível, padronize seus dados para UTF-8 ao exportar. Este é o padrão da indústria e garante a máxima compatibilidade entre diferentes sistemas e plataformas. A atenção aos detalhes nesta fase inicial do projeto pode prevenir uma série de problemas a jusante, tornando sua jornada de análise muito mais suave e produtiva.

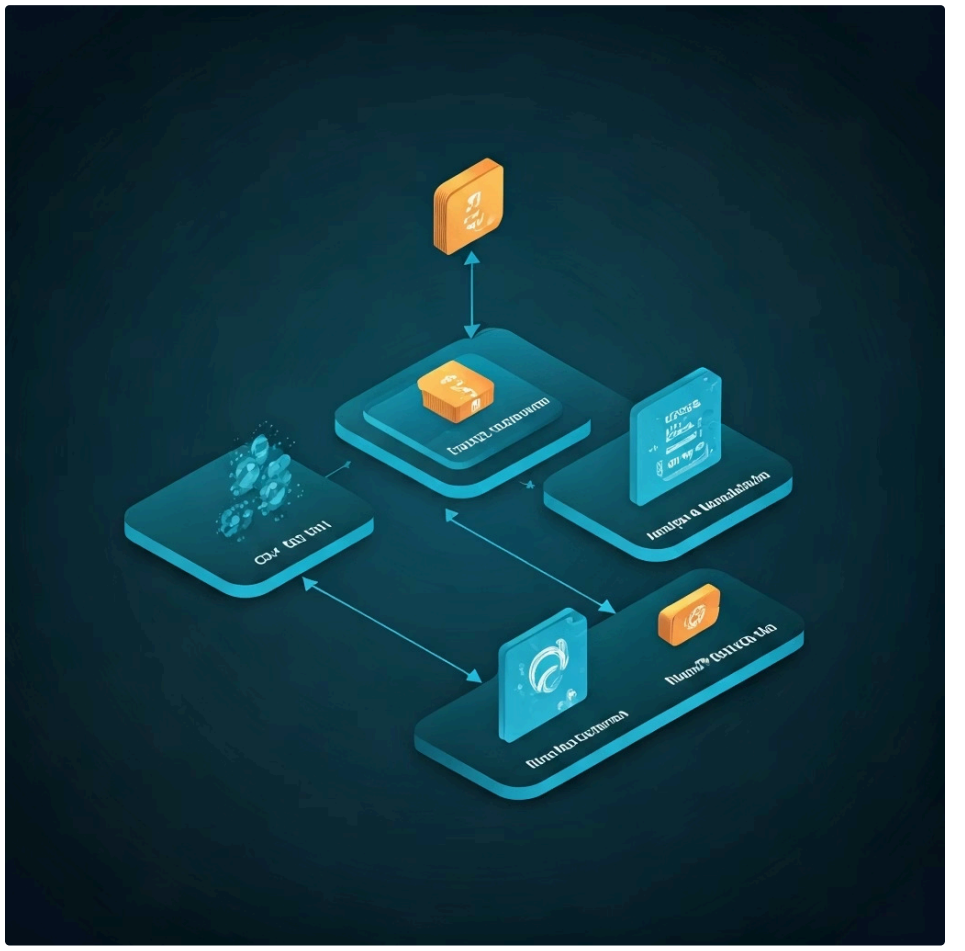
Quadro Comparativo: Formatos de Dados

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
CSV	Simples, universal, dados tabulares	Texto puro, delimitado	nome,idade,cidade
Excel	Planilhas complexas, múltiplas abas, formatação	Binário, proprietário (Microsoft)	relatorio.xlsx
JSON	Dados estruturados, aninhados, APIs web	Texto, chave-valor	{ "produto": "A", "preco": 10 }

O Ecossistema Pandas e a Importância da Documentação

O Pandas é a espinha dorsal da manipulação de dados em Python, e suas funções de importação e exportação são apenas a ponta do iceberg. Ele se integra perfeitamente com outras bibliotecas do ecossistema Python, como NumPy para operações numéricas eficientes, e Matplotlib/Seaborn para visualização de dados. A capacidade de mover dados de e para DataFrames é o que permite que essas ferramentas trabalhem em conjunto de forma harmoniosa.

Para aprofundar seu conhecimento, a documentação oficial do Pandas é um recurso inestimável. Ela é detalhada, atualizada e oferece exemplos para todos os parâmetros e cenários possíveis. Além disso, plataformas como Jupyter Notebooks e Google Colab são ambientes ideais para praticar e experimentar com importação e exportação, permitindo que você execute código em blocos e visualize os resultados imediatamente.



Documentação Oficial

Recurso completo com exemplos detalhados de todos os parâmetros e funções do Pandas

Jupyter Notebooks

Ambiente interativo ideal para experimentar e visualizar resultados em tempo real

Google Colab

Plataforma gratuita na nuvem para praticar sem necessidade de instalação local

A importação e exportação de dados são habilidades que você usará em praticamente todos os projetos de análise de dados. Dominá-las significa ter controle sobre o fluxo de informações, garantindo que seus dados estejam sempre prontos para a próxima etapa da sua análise.

Em Prática e Autoavaliação

A importação e exportação de dados são as habilidades de "alfabetização" no mundo da análise de dados. Você aprendeu a ler e escrever dados nos formatos mais comuns, a ajustar parâmetros para lidar com as peculiaridades de cada arquivo e a resolver problemas de codificação. Lembre-se que a prática leva à perfeição: quanto mais você manipular diferentes tipos de arquivos, mais intuitivo se tornará o processo.

Em prática:

- Sempre inspecione o arquivo de origem antes de importar para identificar o separador, cabeçalho e codificação.
- Utilize os parâmetros sep, header, index_col e encoding para garantir a leitura correta.
- Ao exportar, use index=False para evitar colunas de índice desnecessárias e prefira encoding='utf-8' para máxima compatibilidade.
- Explore a documentação do Pandas para descobrir outros parâmetros úteis para cenários específicos.

Autoavaliação

1. Qual parâmetro da função pd.read_csv() é utilizado para especificar o caractere que separa os valores em um arquivo CSV? a) delimiter b) separator c) sep d) split_char
2. Ao importar um arquivo Excel com múltiplas abas, qual parâmetro de pd.read_excel() permite selecionar uma aba específica? a) sheet_index b) tab_name c) sheet_name d) worksheet
3. Você está lendo um arquivo CSV e percebe que os caracteres acentuados estão aparecendo de forma ilegível (ex: Ã§ em vez de ç). Qual é a causa mais provável e como você tentaria resolver? a) O arquivo está corrompido; tente baixá-lo novamente. b) Há um problema de codificação; tente usar o parâmetro encoding='latin-1'. c) O Pandas não suporta caracteres especiais; use uma biblioteca diferente. d) O separador está incorreto; ajuste o parâmetro sep.
4. Qual das seguintes afirmações sobre a exportação de DataFrames é a mais recomendada para garantir compatibilidade? a) Sempre exportar para JSON, pois é o formato mais moderno. b) Usar index=True para manter o índice do DataFrame como uma coluna. c) Preferir a codificação UTF-8 ao exportar para CSV ou JSON. d) Exportar apenas para Excel, pois é o formato mais utilizado em empresas.
5. Descreva a principal diferença estrutural entre um arquivo CSV e um arquivo JSON, e como essa diferença impacta a forma como você aborda a importação de dados para um DataFrame Pandas.

Gabarito:

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

Próxima Aula:

Aula 7 – Seleção e Filtragem de Dados (Indexação, loc e iloc)

Recursos Adicionais:

- **Documentação Oficial do Pandas:** Para detalhes aprofundados sobre todas as funções de I/O.
- **Artigos sobre Encoding:** Para entender melhor os diferentes tipos de codificação e como lidar com eles.
- **Tutoriais de Jupyter Notebook/Google Colab:** Para praticar a execução de código e visualização de resultados.

NOTA IMPORTANTE: As informações técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.