CEFET/RJ

Modificação de Valores em Python

Aprenda técnicas essenciais para carregar, salvar e modificar estruturas de dados em Python. O foco é dominar operações fundamentais com tabelas e vetores, usando arquivos locais, URLs e operações de modificação em colunas. Essas técnicas são a base da limpeza e transformação de dados.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Eduardo Ogasawara
eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

CONCEITOS FUNDAMENTAIS

Atribuição e Estado do Programa

Atribuição associa um nome a um valor na memória. Quando um valor muda, o estado do programa muda. Valores antigos não são lembrados automaticamente - o programa evolui conforme cada linha é executada.

Conceito-chave: Cada atribuição substitui o valor anterior, a menos que você o salve em outra variável.

```
x = 10
x = x + 5  # o valor anterior (10) não é preservado
print(x)   # 15
```

Saída real:
15

CONCEITOS FUNDAMENTAIS

Execução Sequencial

Ordem Importa

As instruções são executadas de cima para baixo, linha por linha.

Estado Atual

Cada linha depende do estado atual das variáveis no momento da execução.

Resultado Final

Trocar a ordem das linhas muda completamente o resultado.

```
a = 5
b = a * 2
a = 10
print(b)   # 10, não 20
```

Saída real:
10

O valor de b é calculado quando a = 5, então permanece 10 mesmo após a mudança em a.

CARREGAMENTO DE DADOS

Carregando Dados de uma URL

Dados podem ser carregados diretamente de uma URL sem precisar baixar o arquivo manualmente. O arquivo remoto é lido e convertido em uma tabela pandas, ficando disponível imediatamente para análise.

Vantagem: Acesso direto a dados atualizados sem gerenciar arquivos locais.

```
url = "https://raw.githubusercontent.com/eogasawara/tutorial-python/main/examples/cartas.csv"
baralho = pd.read_csv(url)
baralho.head()
```

Saída real:

```
face naipe
0 ás ouros
1 dois ouros
2 três ouros
3 quatro ouros
4 cinco ouros
```

PERSISTÊNCIA

Salvando Dados em Arquivo Local

Tabela na Memória

Dados carregados e processados

Salvar em Disco

Persiste o estado atual

Liberar Memória

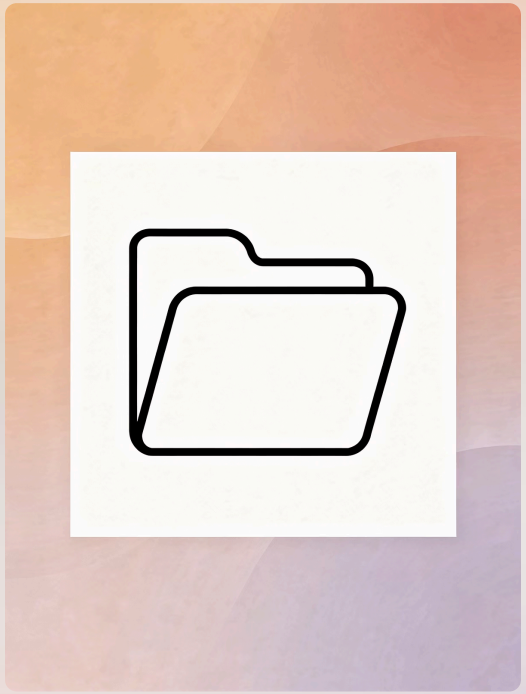
Remove da memória RAM

Arquivo Disponível

Permanece no diretório

```
baralho.to_csv("baralho.csv", index=False)
del baralho
```

O parâmetro `index=False` evita salvar a coluna de índice. O comando `del` remove o objeto da memória, mas o arquivo permanece no disco.



PERSISTÊNCIA

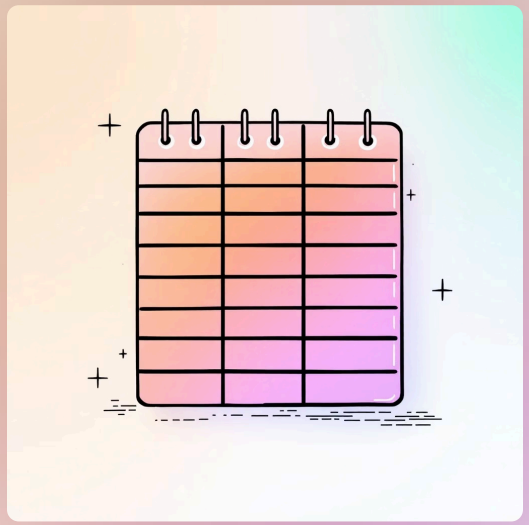
Recarregando Dados do Arquivo

Após salvar, os dados podem ser recarregados do disco a qualquer momento. O conteúdo volta exatamente como estava no momento do salvamento, permitindo interromper e retomar análises. O arquivo local funciona como armazenamento persistente.

```
import pandas as pd
baralho = pd.read_csv("baralho.csv")
baralho.head()
```

Saída real:

face naipe
0 ás ouros
1 dois ouros
2 três ouros
3 quatro ouros
4 cinco ouros



MODIFICAÇÃO DE TABELAS

Adicionando Colunas à Tabela

Uma nova coluna pode ser criada atribuindo valores diretamente à tabela. O tamanho do vetor deve corresponder ao número de linhas. A nova coluna aparece imediatamente e pode ser usada em operações subsequentes.

Função numpy.arange: Cria uma sequência numérica de 1 até o tamanho da tabela.

```
baralho["idx"] = np.arange(1, len(baralho) + 1)
baralho.head()
```

Saída real:

face naipe idx
0 ás ouros 1
1 dois ouros 2
2 três ouros 3
3 quatro ouros 4
4 cinco ouros 5

MODIFICAÇÃO DE TABELAS

Alterando Valores por Índice

01

Selecionar Posições

Use listas de índices para escolher quais elementos modificar

02

Atribuir Novos Valores

A modificação é feita diretamente na coluna especificada

03

Verificar Resultado

As mudanças aparecem imediatamente na tabela

```
baralho.loc[[0, 2, 4], "idx"] = 1
baralho.head()
```

Saída real:

face naipe idx
0 ás ouros 1
1 dois ouros 2
2 três ouros 1
3 quatro ouros 4
4 cinco ouros 1

Esse método é usado para correções pontuais em valores específicos da tabela.

MODIFICAÇÃO DE TABELAS

Alterando Valores por Intervalo

Faixas de posições podem ser modificadas em bloco usando intervalos. Operações matemáticas são aplicadas aos valores existentes, facilitando ajustes sistemáticos. O restante da coluna permanece inalterado.

```
baralho.loc[3:5, "idx"] = baralho.loc[3:5, "idx"] + 1
baralho.loc[0:6, ["idx"]]
```

Saída real:

```
idx
0 1
1 2
2 1
3 5
4 2
5 7
6 7
```

Nota importante: Em pandas, intervalos com `loc` incluem o último elemento (diferente do Python padrão).

FILTRAGEM DE DADOS

Criando Vetores Lógicos

Comparações geram vetores booleanos (True/False) que indicam quais linhas satisfazem uma condição. Operações matemáticas ajudam a criar critérios complexos. Esses vetores são a base da filtragem de dados.

Operação Matemática

Módulo % verifica divisibilidade

Comparação

== 1 identifica valores ímpares

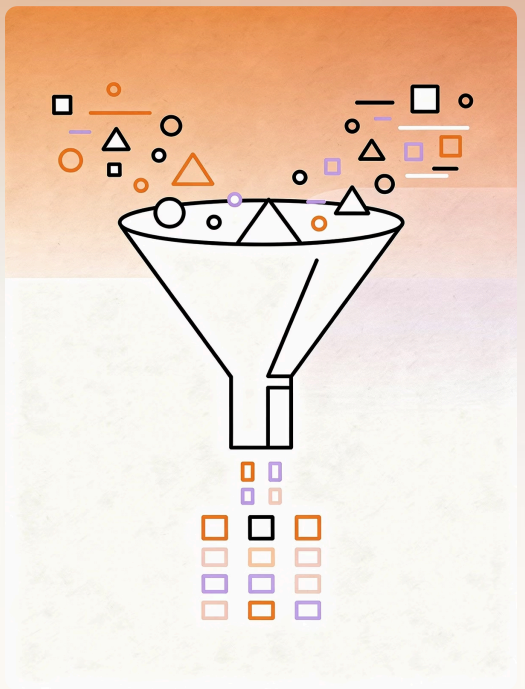
Resultado Booleano

Vetor de True/False para cada linha

```
vec = (baralho["idx"] % 2 == 1)
idx = baralho.loc[vec, "idx"]
idx.head()
```

Saída real:

0 1
2 1
3 5
5 7
6 7
Name: idx, dtype: int64



FILTRAGEM DE DADOS

Aplicando Filtros

Vetores booleanos podem selecionar linhas da tabela de forma eficiente. Somente as linhas onde o vetor é True são mantidas no resultado. Isso produz subconjuntos dos dados baseados em critérios específicos.

```
cartas = baralho[vec]
cartas.head()
```

Saída real:

face naipe idx
0 ás ouros 1
2 três ouros 1
3 quatro ouros 5
5 seis ouros 7
6 sete ouros 7

O resultado é uma nova tabela filtrada contendo apenas as linhas que atendem à condição original.

FILTRAGEM DE DADOS

Operações de Comparação

Comparação Simples

Retorna um único booleano

Comparação Vetorial

Compara elemento por elemento

Verificação de Pertinência

Testa se elementos estão em conjunto

Operadores de comparação produzem valores booleanos. Quando aplicados a vetores, funcionam elemento a elemento, permitindo comparar estruturas inteiras de uma vez. O resultado é usado em operações de filtragem.

```
1 > 2
```

Saída: False

```
np.array([1, 2, 3]) == np.array([3, 2, 1])
```

Saída: array([False, True, False])

```
np.isin([1, 2, 3], [3, 4, 5])
```

Saída: array([False, False, True])

FILTRAGEM DE DADOS

Operadores Lógicos

Operador AND (&)

Exige que ambas as condições sejam verdadeiras simultaneamente. Útil para filtros restritivos.

Operador OR (|)

Aceita quando pelo menos uma condição é verdadeira. Útil para filtros mais abrangentes.

Exemplo com AND

```
x = (baralho["face"] == "dama") & (baralho["naipe"] == "espadas")
baralho[x].head()
```

Saída:

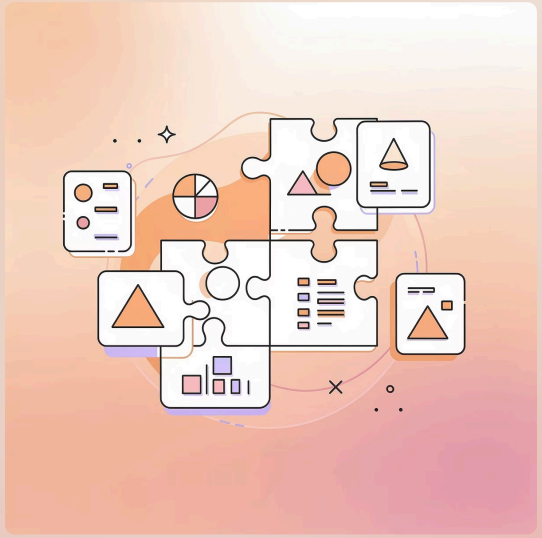
face naipe idx
50 dama espadas 51

Exemplo com OR

```
y = (baralho["face"] == "dama") | (baralho["naipe"] == "espadas")
baralho[y].head()
```

Saída:

face naipe idx
11 dama ouros 12
24 dama copas 25
37 dama paus 38
39 ás espadas 40
40 dois espadas 41



DADOS AUSENTES

Trabalhando com Valores Ausentes

Valores ausentes são representados por NaN (Not a Number). Operações com NaN normalmente produzem NaN, mas funções estatísticas específicas podem ignorar esses valores ausentes, evitando resultados inválidos.

Operação Padrão

`np.mean(v)` retorna NaN se houver ausentes

Operação Segura

`np.nanmean(v)` ignora NaN e calcula a média dos valores válidos

```
v = np.array([np.nan] + list(range(1, 51)))
np.mean(v)
np.nanmean(v)
```

Saídas reais:
nan
25.5

DADOS AUSENTES

Filtrando com Dados Ausentes

Comparações com valores ausentes retornam False ou NaN, podendo eliminar linhas inesperadamente. É fundamental testar e tratar ausentes antes de filtrar para evitar erros sutis na análise.

```
# Criando coluna "valor" a partir de "face" (ás=1 ... rei=13)
ordem = ["ás","dois","três","quatro","cinco","seis","sete","oito","nove","dez","valete","dama","rei"]
mapa = {face: i+1 for i, face in enumerate(ordem)}
baralho["valor"] = baralho["face"].map(mapa)

# Introduzindo alguns NaN (um por naipe)
baralho.loc[[0, 13, 26, 39], "valor"] = np.nan
```

```
baralho[baralho["valor"] < 3]
```

Saída (filtro valor < 3):

face naipe idx valor
1 dois ouros 2 2.0
14 dois copas 15 2.0
27 dois paus 28 2.0
40 dois espadas 41 2.0

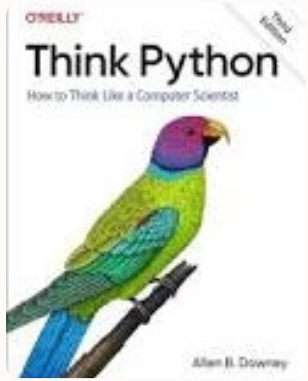
```
baralho["valor"].isna().sum()
```

Contagem de NaN:

4

Use `isna()` para identificar valores ausentes e `dropna()` ou `fillna()` para tratá-los.

Referências



Think Python

Downey, A. *Think Python: How to Think Like a Computer Scientist.* O'Reilly Media.

An essential introduction to programming fundamentals and computational thinking using Python.

Python Data Science Handbook

VanderPlas, J. *Python Data Science Handbook.* O'Reilly Media.

A comprehensive guide to essential tools for working with data in Python, including NumPy, Pandas, and visualization libraries.

Data Science from Scratch

Grus, J. *Data Science from Scratch.* O'Reilly Media.

Learn data science fundamentals by building algorithms and tools from the ground up using Python.