Dataframes

Fatec 2025

Métodos para melhorar o desempenho

- Escolha do algoritmo
  - Evite iterações desnecessárias
- Escolha da estrutura de dados
  - É mais rápido procurar um valor em um dicionário do que em uma lista
- Use funções built-in
  - Muitas são implementadas em C
- Compilando Python
  - Cython (superconjunto do Pyton), Numba (subconjunto do python) e PyPy (reimplementação do Python)
  - Código Assíncrono
  - Realiza uma tarefa enquanto aguarda outra
- Computação paralela e computação distribuída
  - MapReduce
    - Modelo de programação para BigData

Pandas

- Fundamental para manipular e analisar dados
- Reputação de ser lento e consumir muita memória
- Baseado na NumPy
- Duas estruturas principais:
  - Dataframes
  - Séries

Séries

- `Usa_data = pd.Series([13.33,14.02,14.25]), index=[“2000”,”2001.”2002”,”2003”])`
- Onde o ano é o índice
- Criada em um bloco contínuo de memória

Dataframe

- Arranjo bidimensional de estruturas Series do pandas, com um índice de coluna também

- `india_data = pd.Series)[9.02, 9.01, 8.84,8.84],index=[“2000”,”2001”,”2002,”2003”])`
- `df= pd.DataFrame({“USA”, usa_data,”India”, índia_data})`

- Ao contrário de um Array, cada coluna de um Dataframe pode ser de um tipo diferente

Carregando o dataset (conjunto de dados)

```
[21] import pandas as pd
```

```
[22] df= pd.read_csv("gapminder.tsv", sep="\t")
     print(df.head())
```

```
[23] print(type(df))
```

```
[24] print(df.shape)
```

```
[25] print(df.columns)
```

```
[26] #atributo dtype
     print(df.dtypes)
```

```
[27] #método info
     print(df.info())
```

Pandas x Tipos

Pandas	Tipos Python	Descrição
Object	String	Tipo de dados mais comum
Int64	Int	Inteiros
Float64	Float	Reais
Datetime64	Datetime	Encontra-se na biblioteca padrão, ou seja, deve ser importado

Observando linhas e colunas: colunas

```
[29] country_df= df["country"]
     print(country_df.tail())
```

```
[31] subset = df[["country", "continent", "year"]]
     print(subset.head())
```

Subconjunto de linhas

Método	Descrição
Loc	Baseado no nome da linha
Iloc	Baseado no índice

Subconjunto de linhas

```
[34] print(df.loc[0])
     print(df.loc[99])
     print(df.loc[-1])
```

Não existe a posição -1

```
[36] number_of_rows = df.shape[0]
     last_row_index = number_of_rows - 1
     print(df.loc[last_row_index])

     print(df.tail(n=1))
```

```
[37] print(df.loc[[0,99,999]])
```

```
[41] print(df.iloc[1])
     print(df.iloc[99])
     print(df.iloc[-1])
     print(df.iloc[[0,99,999]])
```

E agora?

Combinando

: = seleciona todas as linhas

Iloc -1 = última coluna

```
[52] subset = df.loc[:,["year", "pop"]]
     print(subset.head())
```

```
[53] subset = df.iloc[:,[2,4,-1]]
     print(subset.head())
```

```
subset = df.iloc[:,[2,4,-1]]
print(subset.head())
```

Subconjunto de várias linhas e colunas

```
[72] print(df.iloc[[0,99,999], [0,3,5]])
```

```
print(df.loc[[0,99,999],["country","lifeExp", "gdpPercap"]])
```

Obtendo colunas por intervalo

```
small_range=list(range(5))
print(small_range)
```

```
subset = df.iloc[:,small_range]
print(subset.head())
```

```
small_range = list(range(3,6))
print(small_range)
subset = df.iloc[:,small_range]
print(subset.head())
```

O que acontece se for estipulado um intervalo além do número de colunas existentes no dataframe?

Fatiando colunas

```
[62] small_range = list(range(3))
     subset = df.iloc[:,small_range]
     print(subset.head())
```

Enquanto Range() cria um gerador que será convertido em lista, o uso do dois-pontos só fará sentido para fatiar o obter conjunto de valores

```
[64] subset = df.iloc[:,:3]
     print(subset.head())
```

Fatiar as 3 primeiras colunas

```
[66] small_range = list(range(3,6))
     subset = df.iloc[:,small_range]
     print(subset.head())
```

```
[67] subset = df.iloc[:,3:6]
     print(subset.head())
```

Fatiar as colunas de 3 a 5 inclusive

```
[68] small_range = list(range(0,6,2))
     subset = df.iloc[:,small_range]
     print(subset.head())
```

Para fatiar as 5 primeiras colunas alternadamente troque para `[:, 6:2]`

O que acontecerá se usar o método de fatiamento com dois-pontos, mas deixar de especificar um valor? Por exemplo, qual será o resultado obtido nos casos abaixo?

```
A) df.iloc[:,0:6:]
b) df.iloc[:,0::2]
c) df.iloc[:,:6:2]
d) df.iloc[:,::2]
e) df.iloc[:,::]
```

Dataframes e memória

- Por padrão, são carregados em memória
- Dataframe maior que a memória = problema
- Solução:
  - Carregar apenas as colunas necessárias
    - Argumento usecols em read_csv
  - Argumento chunksize para criar um iterador que permite trabalhar comum subset
  - Biblioteca Dask
  - Biblioteca Polars

Dask -

- O Dask é mais útil quando o DataFrame é muito grande (milhões de linhas)
- npartitions pode ser ajustado para otimizar a performance (quanto mais dados, mais partições você pode querer).

```
import pandas as pd
import dask.dataframe as dd
dados = {
    'nome': ['Ana', 'Bruno', 'Carla', 'Daniel'],
    'idade': [25, 30, 22, 40],
    'salario': [3500, 4200, 3000, 5000]
}
df_pandas = pd.DataFrame(dados)

# Converter o DataFrame pandas para um DataFrame Dask
df_dask = dd.from_pandas(df_pandas, npartitions=2)  #Pode escolher o número de partições para trabalhar de forma paralela

print("Primeiras linhas no Dask:")
print(df_dask.head())
media_salario = df_dask['salario'].mean()
print("\nMédia dos salários:", media_salario.compute())
```

Polars

```
import polars as pl
# Criar um DataFrame diretamente com Polars
df_polars = pl.DataFrame({
    'nome': ['Ana', 'Bruno', 'Carla', 'Daniel'],
    'idade': [25, 30, 22, 40],
    'salario': [3500, 4200, 3000, 5000]
})
print("DataFrame em Polars:")
print(df_polars)

media_salario = df_polars.select(pl.col('salario').mean())

print("\nMédia dos salários:")
Print(media_salario)
```

Comparando

	Pandas	Dask	Polars
Execução	Imediata	Avaliação tardia (lazy evaluation)	Imediata
Uso da memória	Carrega tudo na memória	Processa em partições, utiliza disco se necessário	Otimizada: toda na Memória e rápida
Melhor para...	Lento e pesado para grandes volumes, mas ótimo para dados pequenos/médios e exploração rápida.	Quebra o problema em pedaços menores, paraleliza, aguenta bem volumes que extrapolam a RAM.	Extremamente rápido, usa paralelismo por padrão, consome menos memória mesmo em datasets grandes