

desafio 10

Introdução ao Polars

```
from datetime import datetime

print("Executado em:", datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
```

Executado em: 2025-10-02 10:41:30

- Uma biblioteca Python de código aberto para manipulação e análise de dados.
- Conhecida por seu alto desempenho, especialmente em tarefas que envolvem grandes volumes de dados.
- Construída em Rust, o que contribui para sua velocidade e eficiência.
- Oferece uma API intuitiva e expressiva, similar à do Pandas, facilitando a adoção por usuários familiarizados com essa biblioteca.
- Velocidade: Polars é significativamente mais rápido que o Pandas em muitas operações, graças à sua arquitetura otimizada e ao uso de Rust.
- Eficiência de Memória: Utiliza a memória de forma mais eficiente, permitindo trabalhar com conjuntos de dados maiores sem sobrecarregar o sistema.
- Paralelismo: Aproveita o poder de processamento de múltiplos núcleos para executar tarefas em paralelo, acelerando ainda mais o processamento.
- Flexibilidade: Suporta uma ampla gama de operações de manipulação de dados, como filtragem, agregação, ordenação e junção.
- Integração: Funciona bem com outras bibliotecas populares do ecossistema Python, como NumPy e PyArrow.

- Cenários com Grandes Conjuntos de Dados: Se você trabalha com datasets que não cabem confortavelmente na memória ou que exigem processamento rápido, Polars é uma excelente opção.
- Tarefas de Análise Exploratória de Dados (EDA): A velocidade e eficiência do Polars permitem realizar análises exploratórias de forma mais ágil e interativa.
- Aplicações de Ciência de Dados: A capacidade de lidar com grandes volumes de dados e realizar operações complexas torna o Polars adequado para projetos de aprendizado de máquina e outras áreas da ciência de dados.

Instalação Polars

- O modo mais simples de instalar o Polars é utilizando o terminal do seu computador.
- A instalação pressupõe que o Python já esteja instalado.

```
reticulate::py_install("polars")
```

```
Using virtual environment "\\smb/ra277230/Documentos/.virtualenvs/r-reticulate" ...
```

```
+ "\\smb/ra277230/Documentos/.virtualenvs/r-reticulate/Scripts/python.exe" -m pip install --
```

```
reticulate::py_install("pyarrow")
```

```
Using virtual environment "\\smb/ra277230/Documentos/.virtualenvs/r-reticulate" ...
```

```
+ "\\smb/ra277230/Documentos/.virtualenvs/r-reticulate/Scripts/python.exe" -m pip install --
```

Carregando a biblioteca Polars

- Para utilizar as funções disponibilizadas pela biblioteca, você deve importá-la no início da sua sessão.
- No Python, com o intuito de proteger-se contra conflitos com os nomes de funções disponibilizadas por diferentes bibliotecas, utilizamos frequentemente o conceito de *namespace*.
- Assim, ao importarmos uma biblioteca, recomenda-se atribuir um nome curto (*alias*) à mesma.
- Sempre que formos empregar uma função daquela biblioteca, utilizamos a notação `alias.funcao()`.

```
import polars as pl
```

```
#!pip install polars  
#!pip install fastexcel
```

```
import polars as pl
```

Arquivos Tabulares

Leitura de Arquivos Tabulares

- Ao importar arquivos tabulares, o Polars os representa como um objeto de classe `DataFrame` no Python.
- Os tipos de arquivo abaixo podem ser importados da seguinte maneira:
 - Arquivos delimitados (como CSV e TSV): `pl.read_csv()`.
 - Arquivos Parquet: `pl.read_parquet()`.
 - Arquivos JSON: `pl.read_json()`.
 - Arquivos Excel: `pl.read_excel()`.
 - * Necessita da biblioteca `fastexcel`!

Argumentos Importantes

- `file`: Caminho para o arquivo ou objeto file-like (para URLs).
 - `pl.read_csv("dados.csv")`
- `columns`: Lista de colunas a serem lidas.
 - `pl.read_csv("dados.csv", columns=["coluna1", "coluna3"])`
- `dtypes`: Dicionário especificando os tipos de dados das colunas.
 - `pl.read_csv("dados.csv", dtypes={"idade": pl.Int32})`
- `infer_schema_length`: Número de linhas para inferir o esquema.
 - `pl.read_csv("dados.grande.csv", infer_schema_length=1000)`

- `has_header`: Indica se o arquivo possui cabeçalho.
 - `pl.read_csv("dados_sem_header.csv", has_header=False)`
- `delimiter`: Delimitador usado no arquivo (padrão é `,`).
 - `pl.read_csv("dados.tsv", delimiter="\t")`

Exemplo `airports.csv`

```
getwd()
```

```
[1] "H:/Documentos/me315"
```

```
aeroportos = pl.read_csv("H:/Documentos/me315/airports.csv",
                          columns = ["IATA_CODE", "CITY", "STATE"])
aeroportos.head(2)
```

```
shape: (2, 3)
```

IATA_CODE	CITY	STATE
---	---	---
str	str	str
ABE	Allentown	PA
ABI	Abilene	TX

Exemplo `WDIEXCEL.xlsx`

```
wdi = pl.read_excel("H:/Documentos/me315/WDIEXCEL.xlsx",
                    sheet_name = "Country",
                    columns = ["Short Name", "Region"])
wdi.head(2)
```

shape: (2, 2)

Short Name	Region
---	---
str	str
Aruba	Latin America & Caribbean
Afghanistan	South Asia

Operações em DataFrames

Um Exemplo Simples

```
df = pl.DataFrame({
    "grupo": ["A", "A", "B", "B", "C"],
    "valor1": [10, 15, 10, None, 25],
    "valor2": [5, None, 20, 30, None]
})
df
```

shape: (5, 3)

grupo	valor1	valor2
---	---	---
str	i64	i64
A	10	5
A	15	null
B	10	20
B	null	30
C	25	null

Operando em valor1?

```
df["valor1"]
```

```
shape: (5,)
Series: 'valor1' [i64]
[
  10
  15
  10
  null
  25
]
```

```
df["valor1"].mean()
```

```
15.0
```

```
df["valor1"].drop_nulls()
```

```
shape: (4,)
Series: 'valor1' [i64]
[
  10
  15
  10
  25
]
```

```
df["valor1"].drop_nulls().mean()
```

```
15.0
```

Operando em Colunas

```
df.select([
  pl.col("valor1").mean().alias("media_v1"),
  pl.col("valor2").mean()
])
```

shape: (1, 2)

media_v1	valor2
---	---
f64	f64
15.0	18.333333

Exemplo

Quais são as médias da variável `valor1` e o valor mínimo da variável `valor2` para cada um dos grupos definidos por `grupo`?

```
df.groupby("grupo").agg([
    pl.col("valor1").mean().alias("media_valor1"),
    pl.col("valor2").min().alias("min_valor2")
]).sort("grupo")
```

shape: (3, 3)

grupo	media_valor1	min_valor2
---	---	---
str	f64	i64
A	12.5	5
B	10.0	20
C	25.0	null

De volta ao `flights.csv`

Calcule o percentual de vôos das cias. aéreas “AA” e “DL” que atrasaram pelo menos 30 minutos nas chegadas aos aeroportos “SEA”, “MIA” e “BWI”.

```
voos = pl.read_csv("H:/Documentos/me315/flights.csv",
    columns = ["AIRLINE", "ARRIVAL_DELAY", "DESTINATION_AIRPORT"],
    dtypes = {"AIRLINE": pl.Utf8,
              "ARRIVAL_DELAY": pl.Int32,
              "DESTINATION_AIRPORT": pl.Utf8})
```

<string>:1: DeprecationWarning: the argument `dtypes` for `read_csv` is deprecated. It was r

```
voos.shape
```

```
(5819079, 3)
```

```
voos.head(3)
```

```
shape: (3, 3)
```

AIRLINE	DESTINATION_AIRPORT	ARRIVAL_DELAY
---	---	---
str	str	i32
AS	SEA	-22
AA	PBI	-9
US	CLT	5

Calcule o percentual de vôos das cias. aéreas “AA” e “DL” que atrasaram pelo menos 30 minutos nas chegadas aos aeroportos “SEA”, “MIA” e “BWI”.

```
resultado = (  
    voos.drop_nulls(["AIRLINE", "DESTINATION_AIRPORT", "ARRIVAL_DELAY"])  
    .filter(  
        pl.col("AIRLINE").is_in(["AA", "DL"]) &  
        pl.col("DESTINATION_AIRPORT").is_in(["SEA", "MIA", "BWI"])  
    )  
    .group_by(["AIRLINE", "DESTINATION_AIRPORT"])  
    .agg([  
        (pl.col("ARRIVAL_DELAY") > 30).mean().alias("atraso_medio")  
    ])  
)
```

```
resultado.sort("atraso_medio")
```

```
shape: (6, 3)
```

AIRLINE	DESTINATION_AIRPORT	atraso_medio
---	---	---

str	str	f64
DL	BWI	0.069455
DL	SEA	0.072967
DL	MIA	0.090467
AA	MIA	0.117894
AA	SEA	0.124212
AA	BWI	0.127523