Importação de Dados — Apresentação

Felipe Basilio, Luca Misi Felipe Popic

Introdução

- Este trabalho analisa o processo de importação e manipulação de dados em larga escala utilizando **R**, **Python** e **Julia**.
- O dataset utilizado é o **NYC Yellow Taxi Trip Data**, que possui arquivos mensais que podem chegar a **7 GB** descompactados.
- O objetivo é comparar o comportamento das três linguagens na leitura de arquivos grandes, avaliando:
  - tempo de execução;
  - uso de memória;
  - eficiência das estratégias de importação.
- O foco é explicar claramente:
  - como os dados foram carregados;
  - quais técnicas foram usadas para evitar estouros de memória;
  - como otimizar a importação em diferentes ecossistemas computacionais.

Objetivos da Apresentação

- Demonstrar como os dados foram carregados em **R**, **Python** e **Julia** com diferentes abordagens.
- Apresentar métodos eficientes para leitura de grandes arquivos, incluindo:
  - leitura por **chunks**;
  - leitura parcial do arquivo;
  - seleção de colunas específicas.
- Comparar o desempenho das linguagens em termos de:
  - tempo de execução;
  - consumo de memória;
  - capacidade de lidar com big data dentro das limitações do ambiente Quarto/Beamer.
- Detalhar as decisões metodológicas adotadas para evitar erros como:
  - `std::bad_alloc`;
  - `ArrayMemoryError`;
  - e outras limitações impostas pela RAM disponível durante a renderização.

Metodologia

- Utilização de três linguagens para comparar desempenho de leitura de dados: **R**, **Python** e **Julia**.
- Em R, foram testados métodos como:
  - `readr::read_csv()`
  - `vroom::vroom()`
  - leitura por chunks com `read_delim_chunked()`
  - importação otimizada com `data.table::fread()`
- Em Python:
  - uso de `pandas.read_csv(chunksize=...)` para evitar estouros de memória
  - testes com Polars para leitura mais rápida
- Em Julia:
  - `CSV.read()` com DataFrames sendo utilizado para leitura completa ou parcial (`limit`)
  - capacidade de processar grandes volumes com estabilidade
- Como o dataset tem ~7 GB descompactado, priorizou-se:
  - leitura parcial
  - chunking
  - seleção de colunas

Setup Julia

```
Sys.setenv(R_HOME = R.home())
Sys.setenv(JULIA_HOME = "C:/Users/Windows 10/AppData/Local/Pro

library(JuliaCall)
julia_setup(JULIA_HOME = "C:/Users/Windows 10/AppData/Local/Pr
```

Setup Python

```
library(reticulate)
```

Pacotes R

```
library(readr)
library(data.table)
library(dplyr)
```

Infraestrutura de Teste:

Hardware uniforme para todos os testes

Mesmas condições iniciais de memória e processamento

Ambiente controlado para garantir comparabilidade

Estratégias Implementadas:

Leitura completa: Carregamento integral do dataset na memória

Leitura parcial: Importação de subconjuntos representativos

Processamento por chunks: Divisão do arquivo em partes gerenciáveis

Seleção seletiva: Carregamento apenas de colunas relevantes

Carregamento CSV no R com tempo

```
# Abordagem básica com utils
system.time({
  df_base <- read.csv("yellow_tripdata_2015-01.csv", nrows = 5
})
```

```
   usuário    sistema decorrido
      0.35       0.01       0.36
```

```
# Abordagem otimizada com readr
system.time({
  df_readr <- read_csv("yellow_tripdata_2015-01.csv", n_max =
})
```

```
   usuário    sistema decorrido
      0.89       0.17       0.39
```

Vantagens readr:

Leitura mais rápida

Carregamento CSV em Julia

```
using CSV, DataFrames

@time df = CSV.read(
    "yellow_tripdata_2015-01.csv",
    DataFrame;
    limit = 50000
)
```

```
  5.329374 seconds (15.15 M allocations: 760.007 MiB, 7.40% g
```

```
50000×19 DataFrame
   Row   VendorID  tpep_pickup_datetime  tpep_dropoff_datetime
         Int64     String31              String31

     1          2  2015-01-15 19:05:39   2015-01-15 19:23:42
     2          1  2015-01-10 20:33:38   2015-01-10 20:53:28
     3          1  2015-01-10 20:33:38   2015-01-10 20:43:41
     4          1  2015-01-10 20:33:39   2015-01-10 20:35:31
```

Carregamento CSV em Python (pandas)

```
import pandas as pd
import time

start = time.time()
df = pd.read_csv("yellow_tripdata_2015-01.csv", nrows=50000)
end = time.time()

print(end - start)
```

```
0.2026219367980957
```

Características pandas:

Sintaxe intuitiva

Bom desempenho em arquivos médios

Flexibilidade na manipulação

Chunks em R

```
library(readr)
library(dplyr)

get_stats <- function(x, pos) {
  x %>%
    summarise(n = n())
}

in_chunks <- read_csv_chunked(
  "yellow_tripdata_2015-01.csv",
  callback = DataFrameCallback$new(get_stats),
  chunk_size = 50000
)

in_chunks
```

```
# A tibble: 255 x 1
```

Vantagens de Utilizar Chunks

evitar crashs

melhor gestao de memoria

Chunks em Python

```
import pandas as pd

chunks = pd.read_csv(
    "yellow_tripdata_2015-01.csv",
    chunksize=50_000
)

total_rows = 0
for c in chunks:
    total_rows += len(c)

total_rows
```

```
12748986
```

R — Leitura de colunas específicas

```r
readr::read_csv(
  "yellow_tripdata_2015-01.csv",
  col_types = cols_only(
    passenger_count = col_integer(),
    trip_distance = col_double()
  )
)
```

```
# A tibble: 12,748,986 x 2
   passenger_count trip_distance
             <int>         <dbl>
 1               1          1.59
 2               1          3.3
 3               1          1.8
 4               1          0.5
 5               1          3
 6               1          9
```

Python — Seleção de colunas

```
df = pd.read_csv(
    "yellow_tripdata_2015-01.csv",
    usecols=["passenger_count", "trip_distance"]
)
df
```

	passenger_count	trip_distance
0	1	1.59
1	1	3.30
2	1	1.80
3	1	0.50
4	1	3.00
...	...	...
12748981	2	1.00
12748982	2	0.80
12748983	1	3.40
12748984	1	1.30

Julia — Seleção de colunas

```
using CSV, DataFrames

df = CSV.read(
    "yellow_tripdata_2015-01.csv",
    DataFrame;
    select = ["passenger_count", "trip_distance"]
)
```

```
12748986×2 DataFrame
      Row   passenger_count   trip_distance
            Int64             Float64

        1                   1            1.59
        2                   1            3.3
        3                   1            1.8
        4                   1            0.5
        5                   1            3.0
        6                   1            2.0
```

Pontos Importantes do Código

- Configuração do ambiente permitindo a execução de R, Python e Julia no mesmo `.qmd`.
- Adaptação necessária devido ao tamanho do dataset:
  - R: evitar leitura completa com `read_csv()`
  - Python: substituir leitura integral por `chunksize`
  - Julia: leitura rápida, mas com sobrecarga inicial de compilação JIT
- Técnicas aplicadas para evitar estouros de memória:
  - chunking reduzido (50k–200k linhas)
  - callbacks no R
  - uso de `limit=` no CSV.jl
  - leitura de apenas algumas colunas
- Impressão de DataFrames para demonstração das diferenças entre tibbles, pandas DataFrames e DataFrames do Julia.

Resultados e Discussão

- Em **R**, funções como `read_csv()` e `vroom()` falharam devido a `std::bad_alloc`, mas métodos chunked funcionaram bem.
- Em **Julia**, a leitura foi extremamente rápida e estável após a compilação inicial.
- Em **Python**:
  - `pandas` consumiu muita RAM na leitura completa
  - com `chunksize`, tornou-se eficiente
  - Polars foi rápido, mas apresentou problemas no Quarto
- Técnicas baseadas em chunking foram essenciais para lidar com o dataset dentro das limitações de memória do Beamer.
- Conclusão parcial:
  - Julia foi a mais rápida
  - Python teve melhor flexibilidade
  - R funcionou com otimizações específicas

Conclusão

- Trabalhar com big data exige estratégias diferentes para cada linguagem.
- **Julia** apresentou o melhor desempenho bruto na leitura do arquivo.
- **Python**, usando leitura em chunks, se mostrou eficiente e balanceado no consumo de memória.
- **R** precisou de métodos chunked e seleção de colunas para evitar erros.
- A escolha do método depende de:
  - tamanho do arquivo
  - memória disponível
  - ambiente (no caso, o Quarto/Beamer, que limita RAM)
- Técnicas como chunking e leitura parcial são fundamentais em projetos com grandes volumes de dados.