# Desafio3

## Desafio 03

### Nome: Luis Felipe Godoy

### RA: 277197

Nesse desafio, irei explicar o que são os arquivos parquet e JSON; trazer um exemplo de cada um; e carregar cada um deles no R de forma que o arquivo carregado seja um dataframe.

### O que é um Arquivo Parquet?

O arquivo **parquet** é otimizado para processar grandes volumes de dados, e permite compactar cada coluna de forma diferente. Devido a sua estrutura colunar, ele fornece uma melhor compactação e desempenho ao se trabalhar com grandes quantidades de dados. Os metadados de um arquivo Parquet contém informações como versão, schema, tipo, entre outros pontos importantes e são gravados sempre após os dados. Sua extensão é **.parquet**

### Exemplo de Arquivo Parquet:

Como arquivo parquet, usarei um código adaptado do arquivo gerado pelo site https://www.amandanascimento.com/post/arquivo-no-formato-parquet

```
library(arrow)

# Dados fictícios
dados <- data.frame(
  nome = c(
    "amanda nascimento", "joão silva", "maria oliveira",
    "carla souza", "lucas pereira", "renata almeida"
  ),
```

```
  data_nasci = c(
    "31/12/1992", "12/03/1990", "07/08/1995",
    "23/01/1998", "10/11/1988", "27/06/1993"
  ),
  cpf = c(
    "006.444.556-01", "234.555.789-10", "111.222.333-44",
    "999.888.777-66", "555.666.777-88", "123.456.789-00"
  ),
  curso = c("ads", "ti", "enfermagem", "direito", "engenharia", "medicina"),
  idade = c(34, 33, 28, 26, 35, 31),
  stringsAsFactors = FALSE
)

# Salvando em formato Parquet
write_parquet(dados, "dados_alunos.parquet")
```

Para gerar este arquivo em parquet, foi necessário instalar o pacote "arrow". O arrow oferece funções muito rápidas para entrada e saída de dados, em especial parquet (read_parquet(), write_parquet()), visto que o R base não consegue ler ou gerar arquivos parquet.

### Carregando o Arquivo Parquet:

Agora, carregaremos o arquivo "userdata.parquet" no formato dataframe usando a função **read_parquet()**:

```
library(arrow)
library(dplyr)
library(tidyr)

# Ler o arquivo Parquet
df <- read_parquet("dados_alunos.parquet")

# Verificar a estrutura inicial
str(df)
```

```
tibble [6 x 5] (S3: tbl_df/tbl/data.frame)
 $ nome      : chr [1:6] "amanda nascimento" "joão silva" "maria oliveira" "carla souza" ...
 $ data_nasci: chr [1:6] "31/12/1992" "12/03/1990" "07/08/1995" "23/01/1998" ...
 $ cpf       : chr [1:6] "006.444.556-01" "234.555.789-10" "111.222.333-44" "999.888.777-66"
 $ curso     : chr [1:6] "ads" "ti" "enfermagem" "direito" ...
 $ idade     : num [1:6] 34 33 28 26 35 31
```

```r
# Caso alguma coluna seja lista, transformamos em vetor
# Exemplo genérico para todas as colunas
df <- df %>%
  mutate(across(where(is.list), ~ sapply(., `[`, 1)))

# Visualizar o data.frame
head(df)
```

```
# A tibble: 6 x 5
  nome              data_nasci cpf            curso      idade
  <chr>             <chr>      <chr>          <chr>      <dbl>
1 amanda nascimento 31/12/1992 006.444.556-01 ads           34
2 joão silva        12/03/1990 234.555.789-10 ti            33
3 maria oliveira    07/08/1995 111.222.333-44 enfermagem    28
4 carla souza       23/01/1998 999.888.777-66 direito       26
5 lucas pereira     10/11/1988 555.666.777-88 engenharia    35
6 renata almeida    27/06/1993 123.456.789-00 medicina      31
```

Vale notar que o arquivo em parquet ao ser carregado já está classificado como "data.frame", visto que foi utilizada a função **read_parquet** proveniente do pacote arrow. Agora, veremos como isso se aplica a arquivos **JSON**

### O que é o arquivo JSON?

Um **JSON** (JavaScript Object Notation) é um formato de armazenamento e troca de dados estruturados. Ele é simples, leve, baseado em texto e pode ser lido tanto por humanos quanto por máquinas.É muito usado em aplicações web, APIs, bancos de dados e até em configurações de programas. Sua extensão é **.json**

### Exemplo de Arquivo JSON:

Como arquivo JSON, trouxe um código adaptado do site https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Core/Scripting/JSON

```r
library(jsonlite)

# Criando a lista que representa o JSON
squad <- list(
  squadName = "Super hero squad",
  homeTown = "Metro City",
```

```
  formed = 2016,
  secretBase = "Super tower",
  active = TRUE,
  members = list(
    list(
      name = "Molecule Man",
      age = 29,
      secretIdentity = "Dan Jukes",
      powers = c("Radiation resistance", "Turning tiny", "Radiation blast")
    ),
    list(
      name = "Madame Uppercut",
      age = 39,
      secretIdentity = "Jane Wilson",
      powers = c("Million tonne punch", "Damage resistance", "Superhuman reflexes")
    ),
    list(
      name = "Eternal Flame",
      age = 1000000,
      secretIdentity = "Unknown",
      powers = c("Immortality", "Heat Immunity", "Inferno", "Teleportation", "Interdimensiona
    )
  )
)

# Salvando a lista em arquivo JSON
write_json(squad, "super_hero_squad.json", pretty = TRUE)
```

Note que foi necessário instalar o pacote **jsonlite**, que é usado justamente para trabalhar com arquivos JSON, permitindo ler e escrever dados nesse formato de forma integrada ao R. No caso, usamos a função **write_json()** para tranformar essa lista em um arquivo JSON. Vale notar o argumento **pretty = TRUE** da função, que permite que o arquivo seja formatado com indentação e quebras de linha, em vez de ficar tudo em uma única linha.

### Carregando Arquivo JSON:

Agora, iremos carregar o arquivo "super_hero_squad.json" no formato dataframe. Para isso, usaremos a função proveniente do pacote jsonlite **fromJSON()**:

```
library(jsonlite)
library(dplyr)
```

```r
library(tidyr)

# Ler JSON como lista
squad <- fromJSON("super_hero_squad.json", simplifyDataFrame = FALSE)

# Extrair membros (lista de listas)
members <- squad$members

# Descobrir o maior número de poderes
max_powers <- max(sapply(members, function(x) length(x$powers)))

# Criar data.frame plano
members_df <- data.frame(
  name = sapply(members, function(x) x$name),
  age = sapply(members, function(x) x$age),
  secretIdentity = sapply(members, function(x) x$secretIdentity),
  stringsAsFactors = FALSE
)

# Adicionar cada poder como coluna separada
for(i in 1:max_powers){
  members_df[[paste0("power_", i)]] <- sapply(members, function(x) {
    if(length(x$powers) >= i) x$powers[i] else NA
  })
}

# Visualizar resultado
print(members_df)
```

```
             name     age secretIdentity              power_1           power_2
1    Molecule Man      29      Dan Jukes Radiation resistance      Turning tiny
2 Madame Uppercut      39    Jane Wilson  Million tonne punch Damage resistance
3   Eternal Flame 1000000        Unknown          Immortality     Heat Immunity
              power_3       power_4                 power_5
1     Radiation blast          <NA>                    <NA>
2 Superhuman reflexes          <NA>                    <NA>
3             Inferno Teleportation Interdimensional travel
```

Nesse caso, foi necessário primeiro tranformar o arquivo em uma lista (squad) usando a função **fromJSON()**, com o argumento **simplifyDataFrame = FALSE**, que impede a conversão do arquivo todo em um dataframe. Fazemos isso pois queremos que apenas a lista de listas

**members** vire um dataframe, porém nessa lista temos o vetor de tamanho variável **powers** que prejudicaria essa tranformação. Assim, deixamos tudo em linhas aninhadas para conseguirmos controlar manualmente como os dados serão organizados depois.

Em seguida isolamos a lista de listas members do arquivo para ser manipulada, e contamos qual é o maior número de poderes que um herói pode ter. Isso vai permitir que mais tarde possamos transformar o vetor **powers** em várias colunas de poderes. Depois pegamos os outros vetores como **name**, **age** e **secretIdentity** e os juntamos em um dataframe.

Por fim, adicionaremos as colunas de poderes ao **members_df**, nosso arquivo data frame resultante. Para isso, percorremos de 1 até max_powers, e cada iteração gera uma nova coluna chamada "**power_i**". Em seguida, para cada herói, checaremos a sua quantidade de poderes e então os colocaremos em suas respectivas colunas. Caso um herói não consiga preencher todas as colunas, essas serão preenchidas com NA