Big Data Analytics

Análises de dados simples utilizando R

Agenda

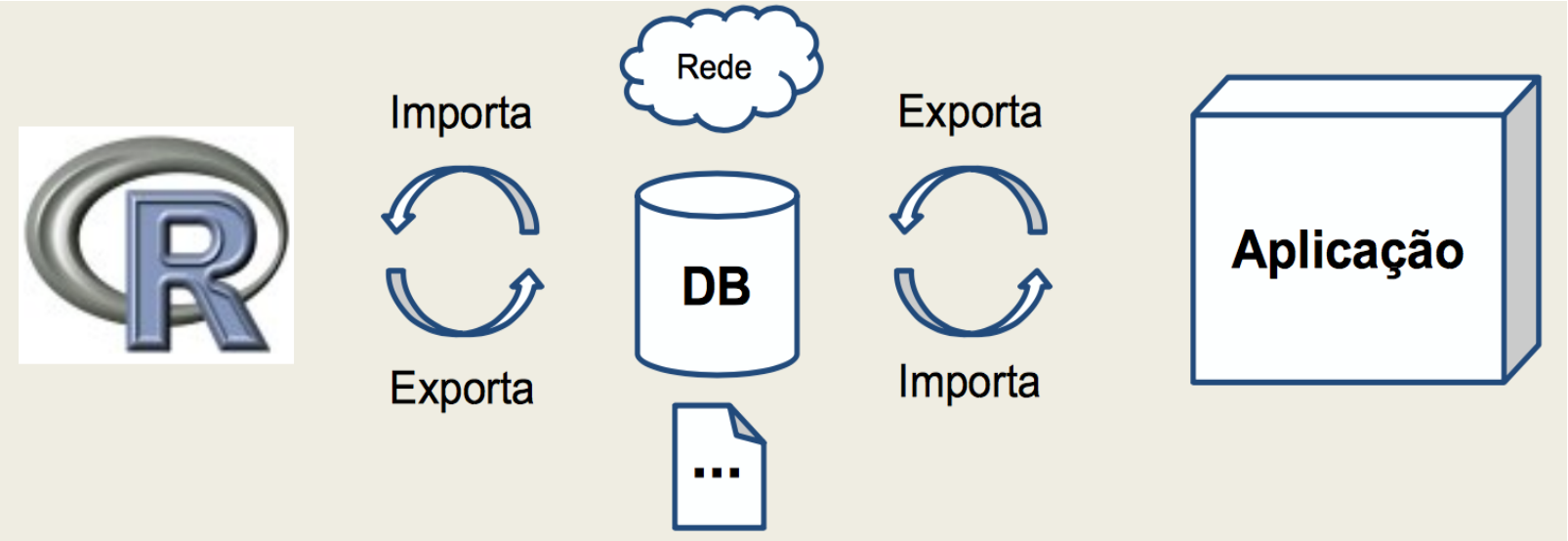
- Entrada e saída de dados simples utilizando R;
- Compreender as diferentes conexões a dados utilizando R.

Entrada e saída de dados simples utilizando R

Entrada de dados

- Temos diferentes maneiras para realizar uma entrada de dados no R.
- O formato mais adequado vai depender do tamanho do conjunto de dados, se os dados já existem em outro formato para serem importados ou se serão digitados diretamente no R.
- Vamos abordar algumas formas de entrada de dados, com exemplos de como utilizá-las.

Entrada de dados



Manipulando arquivos e pastas

- O R trabalha sempre em um diretório conhecido com *working directory* (pasta de trabalho) do sistema operacional.
- A linguagem oferece instruções para manipulação de arquivos e diretórios.

Manipulando arquivos e pastas

```
# Qual é a pasta de trabalho atual?
getwd()

# Podemos alterar a pasta de trabalho
# a qualquer momento.
setwd("D:/Dados")
getwd()
```

Manipulando arquivos e pastas

- Perceba que o separador é diferente do habitual do Windows. Isso deve-se ao fato da barra (“\”) ser um comando especial da linguagem.
- Caso queira manter o padrão do Windows, basta usar duas barras:

```
setwd("D:\\Dados")
```

Manipulando arquivos e pastas

- A função **`file.path()`** cria o caminho no formato apropriado e pode ser usada para criação de caminhos de forma dinâmica durante a execução de um script.

```
caminho = file.path("C:","pasta_1",
"pasta_2","meu_arquivo.txt")
```

Manipulando arquivos e pastas

```
# listar os arquivos e pastas.
list.files()

# listar pastas e subpastas.
list.dirs()

# verificando a existência de um arquivo.
file.exists("teste.txt")
file.exists("teste1.txt")
```

Manipulando arquivos e pastas

```
# remover arquivo
file.remove("teste.txt")
file.exists("teste.txt")

# criar e remover pastas
dir.create("Nova Pasta")
list.dirs()
file.remove("Nova Pasta")
list.dirs()
```

Manipulando arquivos e pastas

```
# Em última instância podemos usar
# comandos do SO diretamente.
shell("md teste")
```

Formatos próprios do R

- A linguagem R possui dois formatos próprios de arquivos:
  - **RData** ou **rda**: salva um ou vários objetos da área de trabalho. Espera pelo mesmo nome ao ser carregado.
  - **rds**: salva apenas um objeto. Pode ser carregado com nome diferente.

Manipulando RData e rda

```
mtcars <- mtcars
# Salva como RData
save(mtcars, file="mtcars.RData")
rm(mtcars)
ls()

# carrega o arquivo
load(file = "mtcars.RData")
ls()
```

Manipulando rds

```
# Salvando como rds
saveRDS(mtcars, file="mtcars.rds")
rm(mtcars)

# carregando o objeto com nome
# diferente
dados <- readRDS("mtcars.rds")
ls()
```

Dados e o R

- Para analisarmos dados precisamos carregar os mesmos para processá-los na linguagem R.
- Os dados podem estar em diversos formatos de texto, como:
  - CSV (Comma Separated Values);
  - JSON (JavaScript Object Notation);
  - XML (Extensible Markup Language).

Dados e R

- Desta forma, vamos ver como importar dados nestes formatos para o trabalho posterior no R.

CSV

- O formato CSV é um dos melhores para representar conjuntos de registros onde cada um destes registros possui uma lista idêntica de campos.
- Este cenário corresponde a uma relação simples em um banco de dados relacional ou dados (não calculados) em uma planilha.

Lendo um CSV

- Vamos usar a base de dados disponível no repositório UCI, chamada Auto MPG.
- A função **`read.csv()`** cria um objeto Data Frame a partir dos dados em um arquivo CSV.
- O parâmetro **`header=TRUE`** lê a primeira linha do CSV como o cabeçalho do conjunto de dados.

Lendo um CSV

- Já o parâmetros **`sep=","`** indica o separador de campos a ser usado.
- Estes dois parâmetros são opções padrão da instrução, logo podem ser omitidos.

Lendo um CSV

- Lendo o CSV:

```
auto <- read.csv("auto-mpg.csv",
header=TRUE, sep=",")
```

- Verificando o resultado:

```
names(auto)
```

Delimitadores

- Como visto anteriormente, podemos definir o delimitador decimal e o separador de campos para uma correta importação.
- Nas regiões onde o separador decimal é a vírgula (como a nossa), usa-se o `';'` como separador de campos.

Delimitadores

- Existe uma função de leitura de CSVs que já incorpora os padrões que utilizamos normalmente, chamada **`read.csv2()`**.
- Para arquivos onde os campos são separados por TABs, pode-se usar o parâmetro **`sep="\t"`**.

Arquivos sem cabeçalho

- Caso o arquivo de dados não possua cabeçalho, devemos usar o parâmetro **`header=FALSE`**.
- Vamos tentar importar o arquivo “auto-mpg-noheader.csv”. Veremos que o R definirá um cabeçalho após a importação.

```
auto <- read.csv(“auto-mpg-noheader.csv", header=FALSE)
head(auto,2)
```

`names()` e `head()`

- Nos últimos exemplos utilizamos duas instruções do R muito úteis na investigação inicial dos dados: **`names()`** e **`head()`**.
- A primeira apresenta o cabeçalho dos dados.
- A segunda apresenta uma quantidade inicial de linhas do data frame. Podemos especificar a quantidade a ser apresentada por meio de um parâmetro.

`tail()`

- A função **`head()`** tem uma "função irmã" chamada **`tail()`**.
- Ela apresenta uma quantidade final de linhas do data frame.
- Mais uma vez, podemos especificar a quantidade a ser apresentada por meio de um parâmetro.

Ainda falando de cabeçalhos

- Caso o arquivo de dados possuir cabeçalho, e mesmo assim usarmos o parâmetro opcional **`header=FALSE`**, a função read.csv() irá criar um cabeçalho para os dados acrescentando um X aos dados da primeira linha do conjunto de dados.
- Isso normalmente gera bastante confusão. Portanto, cuidado!

Ainda falando de cabeçalhos

```
auto <- read.csv("auto-mpg-
noheader.csv")

head(auto)
```

Ainda falando de cabeçalhos

- Podemos usar o parâmetro **`col.names`** para especificar nomes para as colunas.
- No caso da opção explícita pelo parâmetro **`col.names`** os nomes da linha inicial serão ignorados, mesmo que o parâmetro **`header=TRUE`** seja especificado.

Ainda falando de cabeçalhos

```
auto <- read.csv("auto-mpg-
noheader.csv", header=FALSE,
col.names = c("No", "mpg", "cyl",
"dis","hp", "wt", "acc", "year",
"car_name"))

head(auto)
```

Valores ausentes

- Ao ler dados de arquivos do tipo texto, o R trata dados vazios (os chamados blanks) em variáveis numéricas como NA (*Not Available* – Não Disponível), significando ausência de dados.
- Como padrão, o R lê *blanks* em atributos categóricos como vazios e não como NAs.

Valores ausentes

- Para que o R considere *blanks* como NAs em atributos categóricos e variáveis do tipo texto, devemos utilizar o parâmetro **`na.strings=“”`**.

```
nome_dataframe <-
read.csv(“nome_csv.csv",
na.strings="")
```

Valores ausentes

- Caso o arquivo de dados utilize uma string específica como indicador de dados ausentes (Ex.: “N/A” ou “NA”), podemos usar o parâmetro **`na.strings`**.
- Por exemplo: **`na.strings= "N/A"`** ou **`na.strings = "NA"`**.

Coerção forçada

- Já vimos o conceito de coerção.
- Sabemos que é possível especificar a conversão se tipos durante a importação com a utilização do parâmetro `colClasses=Vetor_de_Classes`.

-
```
dados <-
read.csv("Dados/PETR4_VALE5.csv",
colClasses=c('myDate','numeric','n
umeric'), header=TRUE, sep=";",
dec=",")
```

Coerção forçada

- Já vimos o conceito de coerção.
- Sabemos que é possível especificar a conversão se tipos durante a importação com a utilização do parâmetro `colClasses=Vetor_de_Classes`.

-
```
dados <-
read.csv("Dados/PETR4_VALE5.csv",
colClasses=c('myDate','numeric','n
umeric'), header=TRUE, sep=";",
dec=",")
```

Mais um objeto: *factors*

- *Factors* são objetos criados com vetores. Armazena o vetor juntamente com valores distintos dos elementos do vetor como índice. Esta indexação é no formato texto, independentemente dos tipos dos dados armazenados.

Mais um objeto: *factors*

- Um importante uso de *factors* é na modelagem estatística pois os modelos tratam variáveis categóricas de forma diferente das variáveis contínuas. Armazenar dados categóricos como *factors* garante que estes dados sejam tratados de forma correta.

Mais um objeto: *factors*

```
# Cria um vetor.
apple_colors <-
c('green','green','yellow','red','red','r
ed','green')

# Cria um objeto do tipo factor.
factor_apple <- factor(apple_colors)

# Apresenta o objeto criado.
print(factor_apple)
print(nlevels(factor_apple))

#Contabiliza as ocorrências
table(factor_apple)
```

Importando como caracter, e não *factors*

- Como padrão, o R trata as *strings* como *factors*.
- Pode ser útil mantê-las como *strings* de caracteres. Para isso devemos usar o parâmetro **`stringAsFactors=FALSE`**.

Lendo dados diretamente de websites

- Se os dados estiverem disponíveis na web, podemos carrega-los diretamente para o R, sem precisar descarregá-lo e salvá-lo localmente.

```
dat <-
read.csv("http://www.exploredata.net/ftp/WHO.csv")
```

Lendo dados em XML

- Ao extrair dados de websites e/ou analisar dados fornecidos por terceiros, pode ser necessário importar dados nos formatos XML ou JSON.
- Vamos ver inicialmente a importação de um XML.

Lendo dados em XML

- Vamos iniciar pela instalação de um dos pacotes mais utilizados para processamento de XML em R.
- Depois faremos um passo a passo explicando cada uma das etapas. Esta "receita" pode ser reproduzida como base para importação de qualquer outro XML.

```
install.packages("XML")
```

Lendo dados em XML

```
# Carregando a biblioteca.
library(XML)

# Inicializando.
url <-
"http://www.w3schools.com/xml/cd_catal
og.xml"

# Fazendo o "parsing" do arquivo e
pegando o root node.
xmldoc <- xmlParse(url)
rootNode <- xmlRoot(xmldoc)
rootNode[1]
```

Lendo dados em XML

```
# Extraindo os dados do XML. Usamos
# uma função que varre iterativamente
# todos os filhos do root node e
# armazena em uma matriz.
data <- xmlSApply(rootNode, function(x)
xmlSApply(x, xmlValue))

# Conventendo em data frame. Para isso
# temos que fazer uma transposição da
# matriz.
catalogo.cd <-
data.frame(t(data),row.names=NULL)

# Verificando os resultados.
head(catalogo.cd)
```

Lendo dados em JSON

- Diversos serviços web conhecidos como RESTful entregam dados no formato JSON.
- Vamos instalar um dos pacotes que podem processar arquivos JSON em R.

```
install.packages("jsonlite")
```

Lendo dados em JSON

```
# Carregando a biblioteca.
library(jsonlite)

# Carregando dados dos arquivos JSON.
dat.1 <- fromJSON("students.json")
dat.2 <- fromJSON("student-
courses.json")

# Carregando um arquivo JSONda web.
url <-
"http://finance.yahoo.com/webservice/v
1/symbols/
allcurrencies/quote?format=json"
jsonDoc <- fromJSON(url)
```

Lendo dados em JSON

```
# Extraindo dados para os data
frames.
dat <-
jsonDoc$list$resources$resource$fie
lds

# Verificando os resultados.
head(dat)
head(dat.1)
head(dat.2)
```

Compreender as diferentes conexões a dados utilizando R

Conexões com dados

- Se os dados já estão disponíveis em formato eletrônico, isto é, já foram digitados em outro programa, você pode importar os dados para o R sem a necessidade de uma conversão intermediária para arquivo texto.

Conexões com dados

- Esta possibilidade é mais eficiente e requer menos memória do que converter para formato texto.
- Há funções para importar dados diretamente de aplicações como EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat e Octave.
- Muitas funções que permitem a importação de dados de outros programas são implementadas nos pacotes **`foreign`** e **`heaven`**.

Exemplos de funções do `foreign`

```
read.dbf() # Para arquivos DBASE

read.epiinfo() # Para arquivos .REC do
Epi-Info

read.mtp() # Para arquivos "Minitab
Portable Worksheet"

read.S() # Para arquivos do S-PLUS
restore.data() para "dumps"do S-PLUS

read.spss() # Para dados do SPSS

read.dta() # Para dados do STATA
```

Exemplos de funções do heaven

```
read_dta() # Para arquivos do STATA
read_stata() # Para arquivos do
STATA

read_por() # Para arquivos do SPSS
read_sav() # Para arquivos do SPSS
read_spss() # Para arquivos do SPSS

read_sas() # Para arquivos do SAS
```

Bancos de dados externos

- Finalmente, temos diversos pacotes disponíveis no R para fazer a conexão com SGBDs externos. Seguem alguns exemplos:

```
RMySQL # MySQL
RDOBC # Acessos via ODBC
RPostgres # Postgres
RSQLite # SQLite
rmongodb,mongolite # MongoDB
RCassandra #Cassandra
```