R para Economia

Lucas Mendes

24/03/2020

Pacotes

Pacotes

Muitos programadores criam funções com finalidades especificas, como para visualização de dados,modelagem, manipulação de dados e etc. Essas funções ficam armazenadas no que chamamos de pacote. A maioria dos pacotes fica armazenda no CRAN. Já outros ficam armazenadas no github pessoal do desenvolvedor do pacote

O R já vem instalado com alguns pacotes básicos, mas na maioria das vezes temos que baixa - los. Para isso uitlizamos a função:

- `install.packages("nome do pacote")`

Esse comando, baixa o pacote e instala - o na sua máquina. Porém para ativar as suas funcionalidades usamos o comando:

- `library(nome do pacote)`

Pacotes

Iremos agora instalar e carregar o pacote tidyverse

O pacote `tidyverse` fez uma verdadeira revolução na linguagem. Ele usa principios da filosofia tidy para organização dos dados. Ele mesmo não é um pacote em si, mas sim um agrupamento de diversos pacotes que utilizam a filosofia tidy como o dplyr, tidyr, ggplot2, purrr e outros que veremos mais a frente.

Caso você queira saber mais, é só clicar nesse link para o site do tidyverse

Tidyverse


dplyr
forcats
ggplot2
haven
lubridate
purrr
stringr
readxl
tibble
tidyr
tidyverse

Importação/Exportação de dados

Neste capítulo iremos estudar como importar e exportar arquivos com extensões:

- CSV
- XLSX (Excel)

Praticamente tudo é parecido, mudam - se pequenos detalhes.

Definindo diretório de trabalho

O diretório de trabalho é o *caminho* onde o R irá importar e exportar os arquivos.

Para defini - lo, podemos usar a função:

- `setwd("C:/cloud/project/cursoR)`.

Para ver em qual diretório de trabalho está localizado atualmente, podemos usar a função:

- `getwd()`

Importando/Exportando csv

CSV é uma extensão na qual as colunas são separadas por vírgulas (caso americano) e ponto e vírgula (caso brasileiro). Normalmente ele é aberto no excel e parece com uma planilha.

Para importar um csv usamos a função `read.csv()`

```
# Importando
df1 <- read.csv("dados/iris.csv", # Caminho
                header = T, # Há cabeçalho
                sep = ",") # separador de coluna
```

Importando/Exportando csv

Visualizando as primeiras linhas

```
head(df1)
```

X	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Importando/Exportando csv

Para exportar um data frame para um arquivo csv, utilizamos a função `write.csv()`

```
# Exportando
write.csv(df1, # Seu dataframe
          "dados/petalas.csv") # caminho e nome do arquivo
```

Excel

Não há função nativa para importar ou exportar arquivos em excel. Para isso há diversos pacotes disponiveis para tal. Eu irei demonstrar com o `xlsx`

```
# Carragando o pacote
library(xlsx)

# Importando dados
df1 <- read.xlsx('dados/arquivo1.xlsx', # caminho
                 sheetIndex = 1) # Número da Planilha

# Exportando dados
write.xlsx(df1,# data frame
           "dados/arquivo.xlsx") # caminho e nome
```

Funções rápidas

A maioria dos dados que importamos no R são csv. E existem duas funções de outros pacotes que lidam melhor com dados maiores ($>$ 10 MB). Em ordem de velocidade:

Do pacote `readr` a função `read_csv()`

Do pacote `data.table` a função `fread()`

Do pacote `vroom` a função `vroom()`

Para saber mais sobre qualquer outra função, experimente colocar o `?` na frente do nome da função e rode a linha. (Exercicio)

Exercicios

Introdução a manipulação de dados

Manipulação de dados

Para manipular dados, iremos utilizar extensivamente os pacotes `dplyr` e `tidyr`.

dplyr

O dplyr tem diversas funções para que possamos arranjar nossos dados da melhor forma, porém iremos destacar as seguintes:

- select
- filter
- group_by
- summarise
- arrange
- mutate

E a melhor parte é o uso do pipe %>%, que irá fazer nosso código virar uma receita de bolo.

Um pouco mais sobre o pipe %>%

O pipe é um operador que liga as funções, ou seja, você não precisa ficar mais escrevendo uma função dentro da outra.

Exemplo: Se quisessemos calcular a raiz quadrada do log de 4 elevado a 3?

Jeito normal

```
sqrt(log(4**3))
```

```
## [1] 2.039334
```

Com pipe

```
4**3 %>% log() %>% sqrt()
```

```
## [1] 2.039334
```

dplyr

Exemplo de caso

Imagine que você esteja trabalhando em uma grande varejista e o seu chefe te pede uma tabela de resumo de vendas por país excluindo os EUA por ordem descrescente. Dado que sua tabela tem essas colunas

```
##  [1] "QUANTITYORDERED" "PRICEEACH"       "ORDERDATE"
##  [5] "YEAR_ID"         "PRODUCTLINE"     "CUSTOMERNAME"
##  [9] "STATE"           "COUNTRY"
```

Função select

Selecionando colunas necessárias

Quais colunas seriam necessárias para fazer isso?

- Quantidade
- Preço
- País

Para selecionar essas colunas, usaremos a função `select()` do dplyr.

```
library(dplyr)

df2 <- df1 %>%
  select(QUANTITYORDERED,PRICEEACH,COUNTRY) # Colunas do df
```

Selecionando colunas necessárias

```
head(df2)
```

QUANTITYORDERED	PRICEEACH	COUNTRY
30	95.70	USA
34	81.35	France
41	94.74	France
45	83.26	USA
49	100.00	USA
36	96.66	USA

Função filter

Filtrando o País

Como nosso patrão não quer os EUA, iremos filtrá - lo, usando a função filter() do dplyr

```
df2 <- df2 %>%
  filter(COUNTRY != "USA") # Retira os USA da coluna Country
```

Filtrando o País

```
head(df2)
```

QUANTITYORDERED	PRICEEACH	COUNTRY
34	81.35	France
41	94.74	France
29	86.13	France
48	100.00	Norway
41	100.00	France
37	100.00	Australia

Função mutate

Calculando receita total por produto

Como se calcula receita? É só multiplicar a quantidade vendida pelo preço unitário, criando assim uma nova coluna chamada `Revenue`. Criaremos a coluna Revenue usando a função `mutate()` do dplyr

```
df2 <- df2 %>%
  mutate(Revenue = QUANTITYORDERED * PRICEEACH)
# Cria a coluna Revenue
```

Calculando receita total por produto

```
head(df2)
```

QUANTITYORDERED	PRICEEACH	COUNTRY	Revenue
34	81.35	France	2765.90
41	94.74	France	3884.34
29	86.13	France	2497.77
48	100.00	Norway	4800.00
41	100.00	France	4100.00
37	100.00	Australia	3700.00

Função group_by e summarize

Como agrupar e sumarizar os dados

Agora queremos um resumo da receita total por país. Para isso teremos de agrupar nossos dados por país e somar a receita deles. Para isso usaremos a função `group_by()` e `summarize()` do pacote dplyr.

```
df2 <- df2 %>%
  group_by(COUNTRY) %>% # Agrupa os países
  summarise(Total_Revenue = sum(Revenue)) # Sumariza a receit
```

Como agrupar e sumarizar os dados

```
head(df2)
```

COUNTRY	Total_Revenue
Australia	449646.2
Austria	147469.5
Belgium	76835.0
Canada	129257.3
Denmark	158226.1
Finland	213163.8

Função arrange

Como ordenar os dados?

Lembre que queremos as maiores receitas por país? Agora iremos ordenar a nossa tabela por ordem descrescente de receita, para isso usaremos a função arrange()

```
df2 <- df2 %>% arrange(desc(Total_Revenue)) # Ordena pela rec
```

Como ordenar os dados?

```
head(df2)
```

COUNTRY	Total_Revenue
Spain	801604.4
France	718411.4
Australia	449646.2
UK	328618.0
Italy	235042.2
Finland	213163.8

Função count

Função count()

Outra função muito importante é a função `count()`. Ela conta variáveis não númericas, normalmente caracteres ou factors. Imagine que queremos saber qual cliente mais aparece na nossa base de dados?

```
df1 %>% select(CUSTOMERNAME) %>% head()
```

CUSTOMERNAME
Land of Toys Inc.
Reims Collectables
Lyon Souveniers
Toys4GrownUps.com
Corporate Gift Ideas Co.
Technics Stores Inc.

Função count()

```
df3 <- df1 %>% count(CUSTOMERNAME) %>% arrange(desc(n))
head(df3)
```

CUSTOMERNAME	n
Euro Shopping Channel	205
Mini Gifts Distributors Ltd.	156
Muscle Machine Inc	46
Australian Collectors, Co.	44
AV Stores, Co.	44
Anna's Decorations, Ltd	43

Função slice

Fatiando o data frame

Imagine que queiramos saber apenas os 8 clientes que mais aparecem? para isso podemos usar a função `slice()`

```
df3 <- df3 %>% slice(1:8)
df3
```

CUSTOMERNAME	n
Euro Shopping Channel	205
Mini Gifts Distributors Ltd.	156
Muscle Machine Inc	46
Australian Collectors, Co.	44
AV Stores, Co.	44
Anna's Decorations, Ltd	43
Land of Toys Inc.	40
Corporate Gift Ideas Co.	38

Exercicios

tidyr

tidyr

O pacote tidyr é usado para fazer reshape dos dados. As principais funções utilizadas são:

- gather
- spread
- unite
- separate

gather

gather

A função gather transforma um data frame do formato wide em um data frame no formato long.

```
df_wide
```

Ano	Receita	Despesa
2008	1638.121	1401.636
2009	2705.365	1592.019
2010	1809.764	2174.995
2011	2304.728	2988.464
2012	2341.628	2997.793

Função gather

```
df_gather <- df %>% gather(Indicadores,Valores,-Ano)
```

Função gather

Ano	Indicadores	Valores
2008	Receita	1638.121
2009	Receita	2705.365
2010	Receita	1809.764
2011	Receita	2304.728
2012	Receita	2341.628
2008	Despesa	1401.636
2009	Despesa	1592.019
2010	Despesa	2174.995
2011	Despesa	2988.464
2012	Despesa	2997.793

spread

spread

A função `spread` faz exatamente o contrario da `gather`. Ela transforma um data frame do tipo long em um wide

```
df_gather %>% spread(Indicadores,Valores)
```

spread

Ano	Despesa	Receita
2008	1401.636	1638.121
2009	1592.019	2705.365
2010	2174.995	1809.764
2011	2988.464	2304.728
2012	2997.793	2341.628

unite

unite

Essa função tem como objetivo unir duas ou mais colunas em uma.

Nome	Sobrenome
Lucas	Mendes
Julia	Mesquita
Leandro	Lopes
Amanda	Cassar

unite

```
df %>%
  unite(Nome_Completo, # Nome da coluna de união
        c(Nome,Sobrenome)) # Colunas a serem unidas
```

Nome_Completo
Lucas_Mendes
Julia_Mesquita
Leandro_Lopes
Amanda_Cassar

Separate

Separate

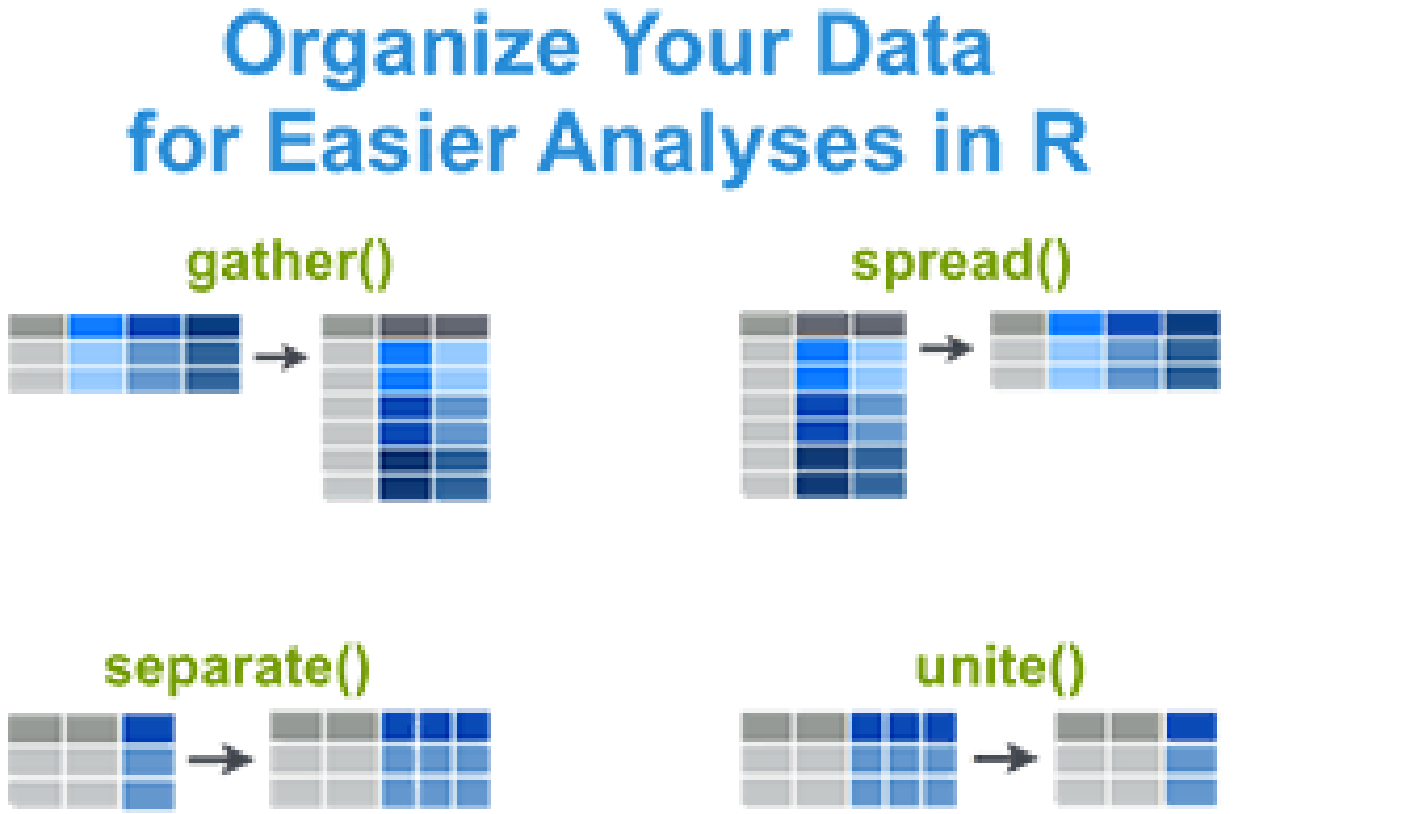
Essa função faz o contrário da `unite`, ou seja, separa uma coluna em várias

```
df %>%
  separate(Nome_Completo,
           c("Nome","Sobrenome"))
```

Nome	Sobrenome
Lucas	Mendes
Julia	Mesquita
Leandro	Lopes
Amanda	Cassar

Resumindo

Resumindo



- **gather()**: collapse multiple columns into key-pair values
- **spread()**: reverse of gather. Separate one column into multiple
- **separate()**: separate one column into multiple
- **unite()**: unite multiple columns into one

Exercicios

Manipulando datas

Manipulando datas

Datas eram muitas vezes uma dor de cabeça, ja que elas vem normalmente em diversos formatos e querem dizer a mesma coisa:

- 02/01/2009
- 2/1/2009
- 2009-2-1
- 1-fev-2009

Para isso, temos o pacote lubridate, que resolve esses problemas de formatação e outras coisas mais.

```
library(lubridate)
```

Lembrando que o R só aceita um formato de data, que é no estilo yyyy-mm-dd

Manipulando datas

```
data_1 <- "2009-02-02"
class(data_1)
```

```
## [1] "character"
```

Se notarmos, o código acima nos retorna um objeto do tipo `character` e nós queremos um objeto do tipo `date`. Logo usamos a função `as.Date`

```
data_1 %>% as.Date() %>% class()
```

```
## [1] "Date"
```

Manipulando datas

Se tentarmos fazer o procedimento com uma data não formatada, irá nos retormar uma data errada.

```
data_1 <- "02/02/2009"
data_1 %>% as.Date()
```

```
## [1] "2-02-20"
```

Para isso não acontecer, usaremos a função do lubridate de acordo com o formato da data

```
data_1 %>% dmy()
```

```
## [1] "2009-02-02"
```

Manipulando datas

Com outra formatação

```
data_2 <- "2009,2,2"
data_2 %>% ymd()
```

```
## [1] "2009-02-02"
```

Manipulando datas

Geralmente você irá usar uma dessas funções:

- ymd para datas no estilo yyyy-mm-dd
- dmy para datas no estilo dd-mm-yyyy
- mdy para datas no estilo mm-dd-yyyy

Para saber mais sobre manipualção de datas, recomendo a leitura do pacote **aqui**

Exercicios