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",
                header = T,
                sep = ",")
```

Importando/Exportando csv

Visualizando as primeiras linhas

```
head(df1)
```

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

Importando/Exportando csv

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

```
# Exportando
write.csv(df1,"dados/petalas.csv")
```

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',sheetIndex = 1)

# Exportando dados
write.xlsx(df1,"dados/arquivo.xlsx")
```

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)
```

Selecionando colunas necessárias

```
head(df2)
```

```
## # A tibble: 6 x 3
##   QUANTITYORDERED PRICEEACH COUNTRY
##             <dbl>     <dbl> <chr>
## 1              30      95.7 USA
## 2              34      81.4 France
## 3              41      94.7 France
## 4              45      83.3 USA
## 5              49     100   USA
## 6              36      96.7 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")
```

Filtrando o País

```
head(df2)
```

```
## # A tibble: 6 x 3
##   QUANTITYORDERED PRICEEACH COUNTRY
##             <dbl>     <dbl> <chr>
## 1              34      81.4 France
## 2              41      94.7 France
## 3              29      86.1 France
## 4              48     100   Norway
## 5              41     100   France
## 6              37     100   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)
```

Calculando receita total por produto

```
head(df2)
```

```
## # A tibble: 6 x 4
##   QUANTITYORDERED PRICEEACH COUNTRY   Revenue
##             <dbl>     <dbl> <chr>       <dbl>
## 1              34      81.4 France      2766.
## 2              41      94.7 France      3884.
## 3              29      86.1 France      2498.
## 4              48     100   Norway      4800
## 5              41     100   France      4100
## 6              37     100   Australia   3700
```

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) %>%
  summarise(Total_Revenue = sum(Revenue))
```

Como agrupar e sumarizar os dados

```
head(df2)
```

```
## # A tibble: 6 x 2
##   COUNTRY   Total_Revenue
##   <chr>             <dbl>
## 1 Australia       449646.
## 2 Austria         147470.
## 3 Belgium          76835 
## 4 Canada          129257.
## 5 Denmark         158226.
## 6 Finland         213164.
```

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))
```

Como ordenar os dados?

```
head(df2)
```

```
## # A tibble: 6 x 2
##   COUNTRY   Total_Revenue
##   <chr>             <dbl>
## 1 Spain           801604.
## 2 France          718411.
## 3 Australia       449646.
## 4 UK              328618.
## 5 Italy           235042.
## 6 Finland         213164.
```

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()
```

```
## # A tibble: 6 x 1
##   CUSTOMERNAME
##   <chr>
## 1 Land of Toys Inc.
## 2 Reims Collectables
## 3 Lyon Souveniers
## 4 Toys4GrownUps.com
## 5 Corporate Gift Ideas Co.
## 6 Technics Stores Inc.
```

Função count()

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

```
## # A tibble: 6 x 2
##   CUSTOMERNAME                      n
##   <chr>                         <int>
## 1 Euro Shopping Channel           205
## 2 Mini Gifts Distributors Ltd.    156
## 3 Muscle Machine Inc               46
## 4 Australian Collectors, Co.       44
## 5 AV Stores, Co.                   44
## 6 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
```

```
## # A tibble: 8 x 2
##   CUSTOMERNAME                      n
##   <chr>                         <int>
## 1 Euro Shopping Channel           205
## 2 Mini Gifts Distributors Ltd.    156
## 3 Muscle Machine Inc               46
## 4 Australian Collectors, Co.       44
## 5 AV Stores, Co.                   44
## 6 Anna's Decorations, Ltd          43
## 7 Land of Toys Inc.                40
## 8 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_long
```

```
##    Ano  Receita  Despesa
## 1 2008 1805.647 1457.500
## 2 2009 2931.691 1068.010
## 3 2010 1926.208 1913.592
## 4 2011 1107.315 2271.948
## 5 2012 1926.233 1522.654
```

Função gather

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

Função gather

```
##      Ano Indicadores  Valores
## 1   2008     Receita 1805.647
## 2   2009     Receita 2931.691
## 3   2010     Receita 1926.208
## 4   2011     Receita 1107.315
## 5   2012     Receita 1926.233
## 6   2008     Despesa 1457.500
## 7   2009     Despesa 1068.010
## 8   2010     Despesa 1913.592
## 9   2011     Despesa 2271.948
## 10  2012     Despesa 1522.654
```

spread

spread

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

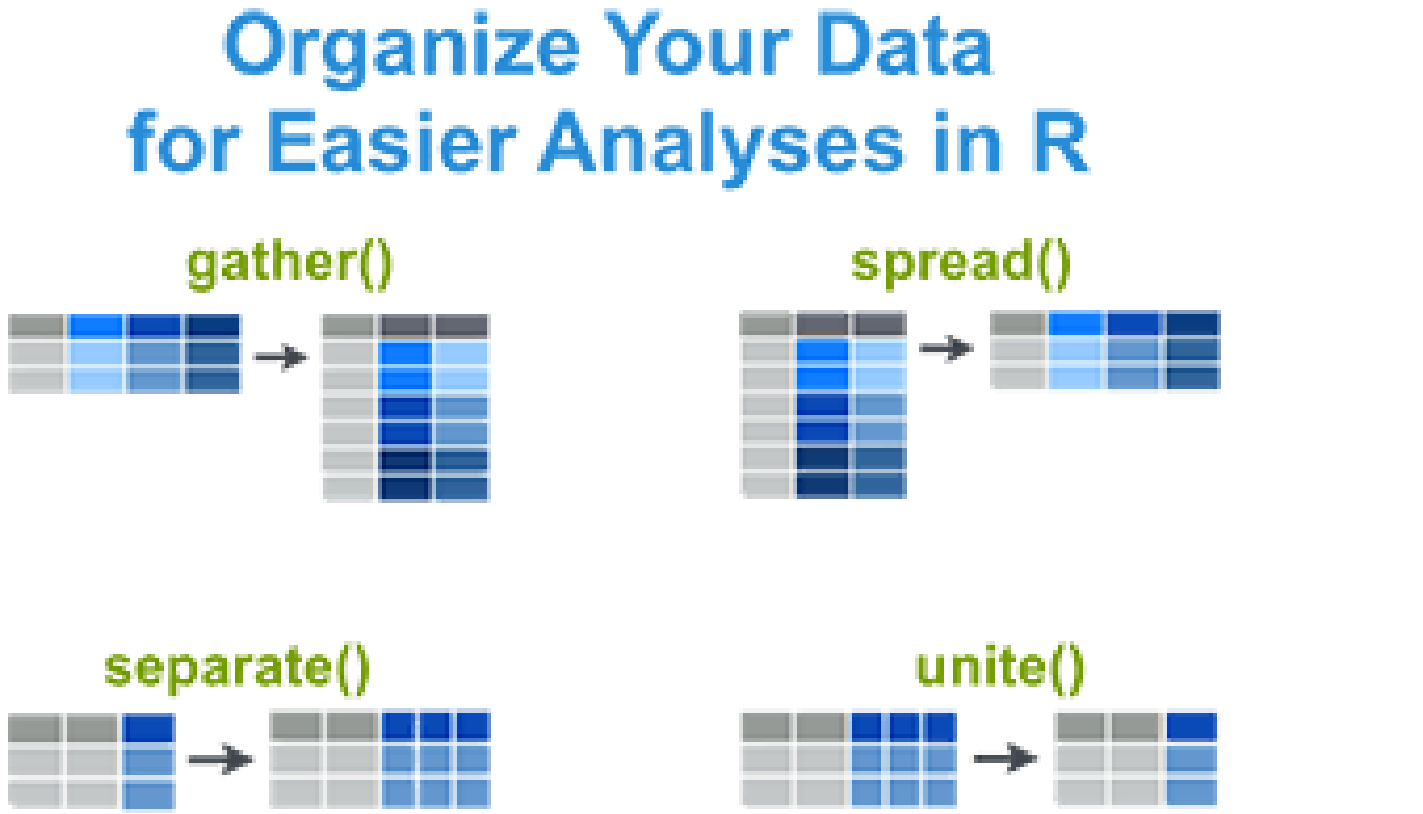
```
df_gather %>% spread(Indicadores,Valores)
```

spread

```
##    Ano  Despesa  Receita
## 1 2008 1457.500 1805.647
## 2 2009 1068.010 2931.691
## 3 2010 1913.592 1926.208
## 4 2011 2271.948 1107.315
## 5 2012 1522.654 1926.233
```

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 rotarmos, 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