Enap

Aulas 5, 6, e 7 (manipulação de dados)

Marcelo Prudente e Rafael Giacomin

20 de março de 2018

1. Manipulação: básico
2. filter()
3. select()
4. arrange()
5. mutate()
6. summarise()
7. JOIN - esquecendo o PROCV()
8. Exercícios

Enap

Manipulação: básico

Manipulação dos dados

Enap

- A maior parte do tempo do analista de dados é gasta com manipulação de dados. Isso envolve:
  - Conhecer os dados
  - Promover as mudanças necessárias nos dados
- A manipulação permite melhorar a precisão dos dados analisados. Com o **R** é possível fazer isso de forma rápida e transparente.

Manipulação de dados com o R

Enap

- Entre os mais de 12.000 pacotes do **R**, alguns foram especificamente desenhados para manipular dados.
- Para as aulas, utilizaremos os pacotes do tidyverse:
  - dplyr
  - lubridate
  - reshape2
  - tidyr
  - ggplot2
- Menção honrosa: data.table()

Manipulação de dados com dplyr

Enap

- O pacote *dplyr* é considerado o melhor para a manipulação de dados
  - sintaxe amigável
  - diversos tutoriais disponíveis
- Inicialmente, utilizaremos o banco "dados_sociais.csv".

Começando pelo pipe

Enap

- Introduzido pelo pacote *magrittr*, O *pipe* **%>%** é uma ferramenta para expressar uma sequência de múltiplas operações com clareza.
- Como o **R** é uma linguagem funcional, o uso dos *pipes* ajuda a reduzir o número de parênteses nas funções
- Isso também auxilia a ler os códigos da direita para a esquerda.
- Além disso, para a manipulação de dados a ser feitac com o *dplyr*, o uso do *pipe* permite um acesso mais fácil às variáveis

Começando pelo pipe

Enap

- Basicamente, o pipe transforma **f(x)** em **x %>% y**.

```
x = c(1.555, 2.555, 3.555, 4.555)
# tirar o log de x
log(x)

# mesma coisa com o pipe
x %>% log()

# vc pode ir além!
x %>% log() %>% round(2) # UAU!
```

Comandos básicos do dplyr

Enap

- Os cinco mais importantes comandos (ou verbos) de manipulação de dados do *dplyr* são:
  1. filter()
  2. select()
  3. arrange()
  4. mutate()
  5. summarise() + group_by()
- Lembre desses comandos! Eles serão seus grandes amigos!!

filter()

filter (): filtrando as linhas

Enap

- Na aula anterior, tiramos subconjuntos dos dados. Com o *dplyr*, isso fica muito mais intuitivo.

```
# Dados apenas do ano de 2010
dados_sociais %>% filter( ano == 2000)
```

- **Exercício**: tente filtrar as linhas dos dados de tal forma que:
  - o ano seja 2010
  - taxa de analfabetismo seja maior que a média
  - o Estado seja do Nordeste

filter (): filtrando as linhas com o pipe

Enap

- Já perceberam como o pipe ajuda a identificar facilmente as variáveis?

```
# Jeito tradicional
filter(dados_sociais, ano == 2000)


# Pipe: as variáveis ficam identificadas!
dados_sociais %>% filter(ano == 2000)
```

- Tentem filtrar pela variável taxa de analfabetismo. Vejam o que acontece.

select()

select(): selecionando as colunas (1)

Enap

- No comando **subset()**, havia a opção *select*. Mais uma vez, o verbo *select* está dedicado a selecionar colunas.

```
dados_sociais %>% select(ano, uf, tx_analf_15m )
```

- **Exercício**: selecione cinco variáveis do banco *dados_sociais*.

select(): selecionando as colunas (2)

Enap

- **Atenção**: é possível selecionar um intervalo.

```
# Selecionar apenas as primeiras variáveis
dados_sociais %>% select(ano:municipio)
```

- **Atenção**: também é possível excluir variáveis

```
# Todas variáveis, exceto município
dados_sociais %>% select(-municipio)
```

select(): selecionando as colunas (3)

Enap

- Como em qualquer operação do **R**, é possível criar um vetor com as variáveis que se quer selecionar.

```
# Selecionar as variáveis uf e tx_analf_15m
minha_selecao <- c("uf", "tx_analf_15m")
ms <- dados_sociais %>% select(one_of(minha_selecao))
```

select(): selecionando as colunas (4)

Enap

- Por fim, o select pode ter apenas a função de reordenar as colunas.

```
# Você pode reordenar grupos
dados_sociais %>% select(cod_ibge:rdpc, ano:uf)

# Você pode reordenar apenas 1 variável.
dados_sociais %>% select(cod_ibge, everything())
```

+ O **everything()** retorna todas outras variáveis.

rename()

Enap

- O *dplyr* apresenta uma forma bastente simplificada para renomear variáveis do banco.

```
# Renomear ano e uf
dados_sociais %>%
  rename(ANO = ano,
         UF = uf)
```

arrange()

arrange(): classificando os dados

Enap

- No *Excel* é comum ordenar os dados. O comando **arrange()** permite fazer isso com muita facilidade.

```
# Exibir os dados de acordo com a menor população
dados_sociais %>% arrange(pop)
```

- **Exercício**: de acordo com a base, quais os três municípios com a menor população no ano de 2010?

mutate()

mutate(): criar novas variáveis

Enap

- Na aula anterior, vimos o uso do **$** ou dos **[]** para criar novas variáveis.
- Com o *dplyr*, essa tarefa fica mais intuitiva. Veja:

```
# encontrar a renda total e logaritmo da populacao
dados_sociais <- dados_sociais %>% mutate(renda_total = rdpc *
                                  log_pop = log(pop))
```

summarise()

summarise() e group_by(): agregar os dados

Enap

- Na aula anterior também solicitamos um exercício para extrair as médias da esperança de vida para cada ano.
- Esse tipo de operação, em que agrupamos uma estatística por grupo, é muito comum e de fácil solução aqui.

```
dados_sociais %>%
  group_by(ano) %>% # agrupa por ano
  summarise(media = mean(esp_vida))
```

- Note que executamos duas operações com o *pipe*. Primeiro, agrupamos pela variável de interesse. Depois, pedimos a média para cada um desses anos.

summarise() e group_by(): agregar os dados

Enap

- **Exercício**: Tente extrair a média da esperança de vida por Estado a cada ano.
  - Armazene os dados em um objeto chamado: "med_esp_ano_uf"

JOIN - esquecendo o PROCV()

JOIN (merge): melhor que o PROCV

Enap

- Em geral, as bases de dados disponíveis não estão completas: precisam ser cruzadas para obter maiores informações.
- No caso da base **dados_sociais**, não foi especificada a região. Como proceder?
- Examinando os dados, nota-se que as regiões representam o primeiro número da variável *uf*.
- Portanto, essa é uma chave para mesclar as tabelas.

JOIN (merge): formas de mesclar dados

Enap

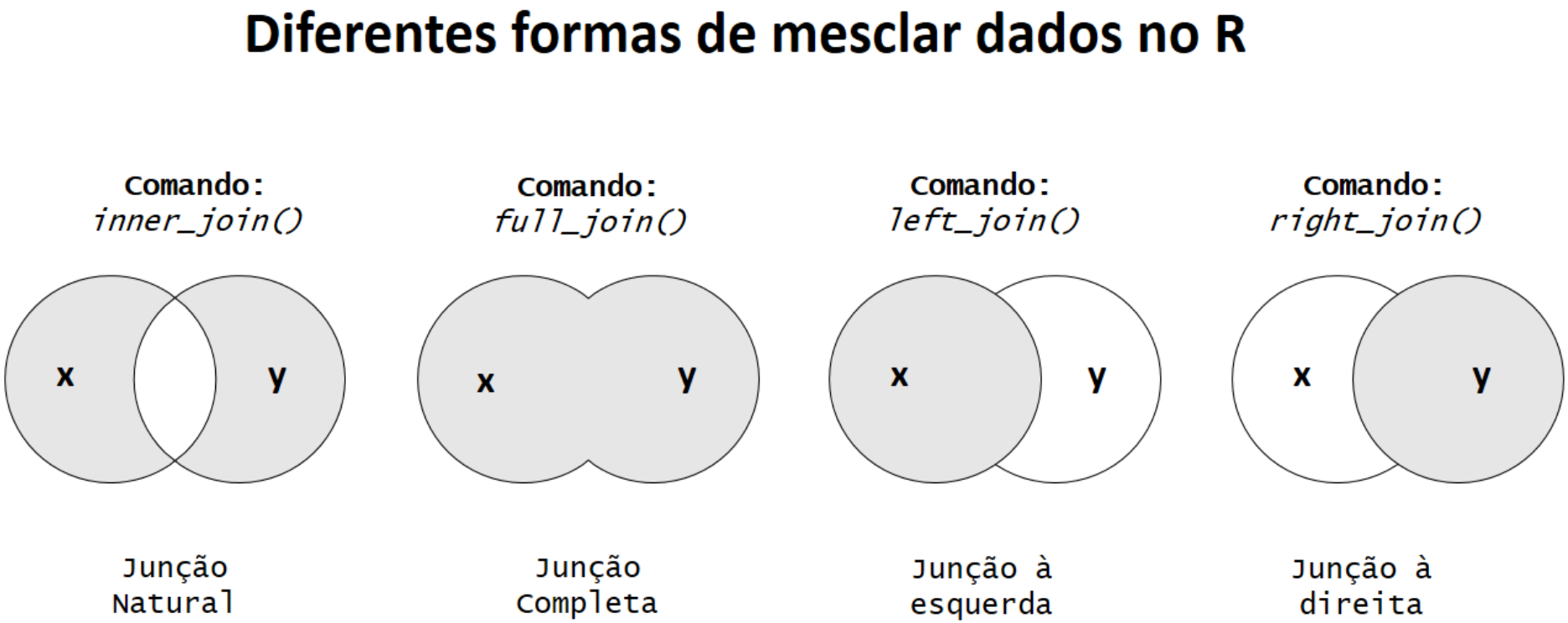

Figure 1: Ilustração das formas de merge no R

JOIN (merge): atenção para as chaves

Enap

Se os nomes das variáveis chaves são iguais, o *dplyr* as identifica

```
inner_join( x , y )     é igual a    inner_join( x , y , by = "chave" )
```

Se os nomes das variáveis chaves são distintos, devem ser identificados

```
inner_join( x , y , by = c( "chave_de_x" = "chave_de_y"  ) )


inner_join( x , y , by = c( "chave1_de_x" = "chave1_de_y" ,
                            "chave2_de_x" = "chave2_de_y" ) )
```

Figure 2: Ilustração das formas de merge no R

JOIN (merge): junção natural

Enap

- Para ilustrar as formas de mesclar dados no **R**, vamos criar dois pequenos dataframes.

```
df1 <- tibble(letras = letters[1:8], X = 1:8)
df2 <- tibble(letras = letters[5:12], Y = 1:8)
```

- Assim, o comando geral é:

```
# Apenas os dados em comum
inner_join(df1, df2)
# Idêntico, mas preferível!
inner_join(df1, df2, by = "letras")
```

JOIN (merge): outros casos

Enap

```
# Junção total
full_join(df1, df2, by = "letras")

# Junção à esquerda
left_join(df1, df2, by = "letras")

# Junção à direita
right_join(df1, df2, by = "letras")
```

- Ainda, pode-se mesclar apenas os dados não coincidentes.

```
anti_join(df1, df2, by ="letras")
anti_join(df2, df1, by ="letras")
```

Exercícios

Fixação do comandos básicos

Enap

- Aplicaremos os comandos aprendidos para efetura análises dos dados sobre o programa Financiamento Estudantil (FIES) e do banco *dados_sociais*.
- Essa base administrativa retrata a população dos alunos matriculados no programa FIES.
- Para esse execício, será utilizada uma amostra de 10% das observações.
- Acesse o arquivo: **exercicio_fixacao_dplyr.pdf**

Primeiro passo: quem são as variáveis?

Enap

- Vamos seguir os seguintes passos:

1. No diretório atual encontram-se os arquivos? Tente utilizar **list.files()**.
2. Baixe o arquivo **fies_sample.csv**.
3. Cheque a estrutura do banco. **str()**
4. Identifique o nome das variáveis **colnames()**. Se quiser, salve em um objeto.

select(): selecionando variáveis relevantes

Enap

- O banco tem 50 variáveis. Nem todas são relevantes.
- Selecione algumas variáveis relevantes e salve no objeto fies_sub:
  - UF, código do contrato, raça, sexo, valor da mensalidade, nome da mantenedora, a data de nascimento, quantidade de semestres financiados, descrição e código do curso e situação de ensino médio escola pública .
- **Pergunta:** é possível selecionar as variáveis apenas por alguns atributos dos nomes (ex: DS, CO, NO, ST ou QT)?

Dicas

Enap

- Em caso de dúvida, use os mecanismos de ajuda:
  - help(comando): help(mutate)
  - ?comando: ?mutate
  - também use os *cheetsheets* do *dplyr* **aqui** ou na pasta *cheet_sheets*
  - ou acesse a página oficial do *dplyr* **aqui**

Mais funções para manipulação de dados

funções auxiliares

Enap

- Na sumarização dos dados, algumas funções são muito úteis para

Função	Decrição
n()	Número de observações no grupo
n_distinct()	Valores únicos de um vetor
cumsum()	Soma cumulativa
rank()	Ranqueia Variáveis
any()	Alguns valores são verdadeiros?
all()	Todos valores são verdadeiros?
quantile()	Quantis

Acesse a pasta cheat_sheet para mais dicas

distinct(): extirpando linhas repetidas

Enap

- Um problema comum em dados administrativos é a repetição de registros.
- As estimativas que fizemos dos contratos do FIES não são tão precisas pois há grande números de contratos repetidos.

```
# há casos duplicados?
fies_sub %>% select(CO_CONTRATO_FIES) %>%
  duplicated() %>% sum()
# quantos casos únicos?
fies_sample %>%
  summarise(unicos = n_distinct(CO_CONTRATO_FIES))
```

distinct(): extirpando linhas repetidas

Enap

- Como a base administrativa do FIES é semestral, cada contrato tem diversas observações para cada mês. Então, é necessário encontrar as observações únicas:

```
fies_sub_dist <- fies_sub %>%
  distinct(CO_CONTRATO_FIES, .keep_all = TRUE)
```

- O argumento *.keep_all = TRUE* mantém todas as variáveis no banco.

Exemplo

Enap

- Veja um exemplo de como tratar os dados do fies

```
fies_sub %>%
  mutate(total = n())
  group_by(SG_UF) %>%
  select(SG_UF, VL_MENSALIDADE)%>%
  summarise(media = mea n(VL_MENSALIDADE, na.rm = TRUE))%>%
  mutate(if_else(media_mens >=1000, "Acima de R$1.000,00",
                 "Abaixo de R$1.000,00"))
```

- Veja como o pipe **%>%** permite encadear uma grande quantidade de comandos.

round() - arredondar

Enap

- No *Excel* arredondar os números exige pouco esforço. Assim também ocorre no **R**.

```
# Gerar uma distribuição normal aleatória
x <- rnorm(10, 5, 1)
# Arredondar
round(x)
# Arredondar com duas casas decimais
round(x, digits = 2)
# Ou ainda...
round(x, 2)
```

any() - algum valor é verdadeiro?

Enap

- Em um banco grande não é possível inspecionar visualmente elementos como os *NAs* ou outras informações.
- A função *any()* permite identificar facilmente se algum elemento possui determinada característica especificada.

```
# Algum elemento do banco "df_na" é NA?
any(is.na(df_na))
# A coluna letras do banco "df_na" é NA?
any(is.na(df_na$letras))
# A coluna letras do banco "df_na" contém a letra E?
any(df_na == "E")
any(df_na$idade >10)
```

cut(): transformar dados numéricos em categóricos

Enap

- Imagine que voc? tem um vetor com a idade de diversos indiv?duos.

```
d <- tibble( id = seq(1:100),
             idade = round(rnorm(100, mean = 35, 15)))
```

- Agora, imagine que voc? tem interesse em reclassificar essa vari?vel num?rica em diversas categorias. Por exemplo, a cada 5 anos.

```
# cortar as idades em intervalos de 5 anos
d$idade_cut <- cut(d$idade, seq(0,100, 5))
```

paste(): concatenar strings

Enap

- Muitas vezes faz-se necessário editar elementos de texto no **R**. Por exemplo, os nomes de um banco.
- A função paste permite fazer isso de forma direta.

```
# Irmãos Peixoto
irmaos <- c("Edgar", "Edclésia", "Edmar", "Edésia", "Edésio")

# Como colocar os sobrenomes?
paste(irmaos, "Peixoto")
```

ifelse(): seleção condicional

Enap

- O *ifelse* é um função vetorizada. Ou seja, é possível passar um vetor em seus argumentos.

```
ifelse(teste_l?gico, valor_se_TRUE, valor_se_FALSE)
# Ent?o
tib <- tibble( x = seq(0L, 30L, 2L), y = LETTERS[1:16])
tib$x1 <- ifelse(tib$x >15, "Maior do que 10", "Menor do que
tib$x2 <- ifelse(tib$x >15, tib$x^2, tib$x)
```

bind_cols e bind_rows

Enap

- Os comandos **bind_** permitem ligar colunas e linhas de bancos com dimensões iguais.

```
# extrair linhas específicas
um  <-  dados_sociais[1:4, ]
dois <- dados_sociais[7011:7014, ]

# ligar em um novo objeto
meu_bind <- bind_rows(um, dois)
```

bind_cols e bind_rows

Enap

- Os comandos **bind_** permitem ligar colunas e linhas de bancos com dimensões iguais.

```
# extrair colunas
um   <- dados_sociais[ , 3 ]
dois <- dados_sociais[ , 8 ]

# ligar em um novo objeto
meu_bind2 <- bind_cols(um, dois)
```

NAs: valores não especificados ou perdidos

Enap

- Ao realizar o **full_join**, o resultado apresenta algumas observações como *NA* (ver exemplo acima).
- Os *NAs* podem representar tanto informações indeterminadas quanto valores propositadamente omitidos.
- De qualquer forma, lidar com os NAs é muito fácil:

```
# Data frame com NAs
df_na <- tibble( letras = LETTERS[1:10],
                 idade =  c( seq(25L, 40L, 5L), NA,
                             NA, 32L, rep(NA, 3)))

# é possível saber se um vetor é NA
is.na(df_na)
# Também é possível remover esses valores
na.omit(df_na)
```

replace_na() : substituir NAs

Enap

- Porém, não é recomendado retirar os NAs sem alguma reflexão. Afinal, eles podem dizer alguma coisa sobre os dados. Ou, ainda, serem apenas campos numúricos não preenchidos.
- Por isso, é melhor substituir os NAs.

```
# Substituindo NAs por números
replace_na(df_na, list(idade = 0))

# Substituindo NAs por textos
replace_na(df_na, list(idade = "Idade não informada"))
```

lubridate: tratar datas

Enap

- Lidar com datas pode ser útil quando lidamos com dados administrativos.
- Vamos utilizar o exemplo do seguro defeso.
- Primeiro, instale e ative o pacote *lubridate*
  - você lembra como instalar e ativar pacotes?
  - dica: *inst. . . .*

lubridate: transformando as datas em datas

Enap

- O lubridate opera datas levando em conta que:
  - $y =$ ano
  - $m =$ mês
  - $d =$ dia
- Assim, você pode transformar *characters* em datas assim:

```
# ano, mês e dia sem separador
ymd("20180131")
# mês, dia e ano com separador "-"
mdy("01-31-2018")
# dia, mês e ano com separador "/"
dmy("31/01/2018")
```

lubridate: acessando informações das datas

Enap

- Estruturar os dados para o formato de datas permite extrair informações básicas para realizar operações lógicas.

```
# criar um vetor de datas
datas<- ymd(c("20180131", "20170225", "20160512"))
# quais os anos
year(datas)
# quais os meses
months(datas); month(datas)
# dias
day(datas)
# dia da semana
wday(datas)
```

lubridate: lendo os dados do seguro defeso

Enap

- Leia os dados do seguro defeso na pasta dados. Lembre-se de efetuar uma inspeção visual antes.
- Peça o **head()** do banco. Quantas datas você identifica?
- Verifique se os vetores de data são datas. Utilize **is.Date()**

Exercício

Enap

- A partir do banco seguro defeso, verifique:
  - as datas presentes no banco tem a classe de data?
  - a data de início do defeso ocorre no período abrangido pelo banco de dados?
  - a data de início do defeso ocorre no período abrangido pelo banco de dados?
  - *melhor*: o defeso ocorre no período do banco de dados?
  - como retirar do banco as linhas em que o defeso não corresponda ao período abrangido pelo banco de dados?
  - em que dias os saques das parcelas ocorreram? Qual dia concentra mais saques?