Ciência de Dados com R

Aula 3 | Manipulação de dados

Izabel Nolau
nolau@dme.ufrj.br

Importação e Exportação de dados

Importando Dados

- O R pode ler dados de arquivos importando-os para serem analisados
- Há duas maneiras de fazer isso:

  (i) especificando todo o caminho do arquivo

  (ii) especificando apenas o nome do arquivo se ele estiver no diretório de trabalho

Importando Dados (Alguns Comandos)

- `read.table( ··· )`: para ler arquivos *.txt*; se `header=TRUE`, considera a primeira linha como os nomes das colunas (o padrão é `FALSE`)
- `read.csv( ··· )`: para ler arquivos *.csv*; geralmente é preciso dizer como os dados são separados: `sep` (o padrão é espaço em branco); e de que forma são os números decimais: `dec` (o padrão é ponto)
- `scan( ··· )`: para ler arquivos em *.txt* ou *.csv* numéricos; empilha todos os dados em um vetor (empilhamento $\rightarrow$)

Exportando Dados

- Além de importar dados de arquivos, podemos também exportá-los
- Alguns comandos para exportação são:
  - `write.table(objeto,"nome.do.arquivo.txt")`: salva os dados em um arquivo *.txt*
  - `write.csv(objeto,"nome.do.arquivo.csv")`: salva os dados em um arquivo *.csv*
- Algumas opções úteis para os comando quando salvamos os dados podem ser `row.names=FALSE` e `col.names=FALSE`, o que faz com que somente os dados sejam gravados no arquivo

Manipulação de bases de dados

Trabalhando com bases de dados

Ao trabalhar com dados, você deve:

- Descobrir o que você deve/quer fazer
- Descrever essas tarefas na forma de um programa de computador
- Executar o programa

O pacote `dplyr` torna estas etapas rápidas e fáceis:

- Ao restringir suas opções, é possível organizar melhor seus desafios
- Fornece funções simples para executar tarefas comuns
- Ele usa *backends* eficientes, economizando tempo computacional

Base de Dados: `flights`

- Para explorar os verbos básicos de manipulação de dados do `dplyr`, usaremos o conjunto de dados `flights` do pacote `nycflights13`

- Este conjunto de dados contém todos os 336776 voos que partiram de Nova York em 2013

- Os dados são provenientes do Bureau de Estatísticas de Transporte dos EUA e estão documentados em `?flights`

Funções do `dplyr`

O pacote visa fornecer uma função para cada verbo básico de manipulação de dados:

- `filter()`: para selecionar dados com base em seus valores
- `arrange()`: para reordenar os dados
- `select()` e `rename()`: para selecionar variáveis baseado nos seus respectivos nomes
- `mutate()` e `transmute()`: para adicionar novas variáveis que são funções de variáveis existentes
- `summarise()`: para condensar vários valores para um único valor

Filtrar linhas com `filter()`

- É usada para selecionar um subconjunto de linhas em um conjunto de dados
- Como todas as outras funções desse pacote, o primeiro argumento são os dados
- O segundo argumento e os subsequentes referem-se a variáveis do DataFrame que queremos filtrar, selecionando linhas nas quais a expressão é TRUE
- Por exemplo, podemos selecionar todos os voos em 1º de janeiro com:

```
filter(flights, month == 1, day == 1)
```

que é equivalente ao seguinte código R:

```
flights[flights$month == 1 & flights$day == 1, ]
```

Organizar linhas com `arrange()`

- Essa função funciona de forma semelhante ao `filter()` mas ao invés de filtrar ou selecionar linhas, ela as reordena
- O DataFrame é reordenado a partir dos nomes das suas colunas
- Se mais de um nome for fornecido, cada coluna adicional será ordenada baseada na ordenação da coluna anterior
- Por exemplo, para ordenar por ano, mês e dia, fazemos:

```
arrange(flights, year, month, day)
```

- Use `desc()` para ordenar em a coluna em ordem decrescente, da seguinte forma:

```
arrange(flights, desc(arr_delay))
```

Selecionar colunas com `select()`

- Muitas vezes trabalhamos com grandes conjuntos de dados com muitas colunas, mas apenas algumas são realmente interessantes
- Essa função permite que você selecione rapidamente um subconjunto útil, basta identificar o DataFrame e as colunas que queremos selecionar

```
# Selecionando as colunas por nome
select(flights, year, month, day)
```

```
# Selecionando todas as colunas entre year e day
select(flights, year:day)
```

```
# Selecionando todas as colunas exceto year até day
select(flights, -(year:day))
```

Renomear colunas com `rename()`

- Podemos renomear variáveis utilizando a função `select()` usando argumentos nomeados:

```
select(flights, tail_num = tailnum)
```

- Mas como `select()` descarta todas as variáveis não mencionadas, para renomer uma variável é melhor utilizar a função `rename()`:

```
rename(flights, tail_num = tailnum)
```

Adicionar novas colunas com `mutate()` e `transmute()`

- Além de selecionar conjuntos de colunas existentes, pode ser útil adicionar novas colunas que sejam funções de colunas existentes

- Permite que você se refira às colunas que você acabou de criar

```
mutate(flights,
gain = arr_delay - dep_delay,
gain_per_hour = gain / (air_time / 60) )
```

- Se você quiser apenas manter as novas variáveis, use `transmute()`

```
transmute(flights,
gain = arr_delay - dep_delay,
gain_per_hour = gain / (air_time / 60) )
```

Resumir valores com `summarise()`

- Essa função reduz um DataFrame em uma única linha
- Funções úteis:
  - Centralidade: `mean()`, `median()`
  - Variabilidade: `sd()`, `IQR()`, `mad()`
  - Ordem: `min()`, `max()`, `quantile()`
  - Posição: `first()`, `last()`, `nth(,n)`
  - Contagem: `n()`, `n_distinct()`
  - Lógicos: `any()`, `all()`, `everything()`

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

Operações agrupadas

- Os verbos `dplyr` são úteis por si só, mas se tornam ainda mais poderosos quando você os aplica a grupos de observações dentro de um conjunto de dados
- No `dplyr`, você faz isso com a função `group_by()` que divide um conjunto de dados em grupos especificados de linhas
- Quando você aplicar os verbos acima no objeto resultante, eles serão aplicados automaticamente por grupo

Juntando com _join()

Atualmente, o `dplyr` suporta quatro tipos de junções:

- `inner_join()`: retorna todas as linhas de `x` onde há valores correspondentes em `y` e todas as colunas de `x` e `y`
- `left_join()`: retorna todas as linhas de `x` e todas as colunas de `x` e `y`. Linhas em `x` sem correspondência em `y` terão valores de `NA` nas novas colunas
- `right_join()`: retorna todas as linhas de `y` e todas as colunas de `x` e `y`. Linhas em y sem correspondência em `x` terão valores de `NA` nas novas colunas
- `full_join()`: retornar todas as linhas e colunas de `x` e `y`. Onde não há valores correspondentes, retorna `NA`

Bons estudos! ☺