Introdução ao Software R

Primeiros Passos

www.de.ufpb.br

www.youtube.com/channel/UC8QTeEyzHqYRjojKneTgLbA

UFPB Departamento de ESTATÍSTICA

Gerar sequências (usando : ou `seq`)

- : (dois pontos)
  - Dois pontos : é usado para gerar sequências de um em um, por exemplo a sequência de 1 a 10:

```
# o comando : é usado para especificar sequências
1:10
 [1]  1  2  3  4  5  6  7  8  9 10

5:16 # Aqui a sequência vai de 5 a 16
 [1]  5  6  7  8  9 10 11 12 13 14 15 16
```

- `seq`
  - A função `seq` é usada para gerar sequências especificando os intervalos. Vamos criar uma sequência de 1 a 10 pegando valores de 2 em 2.

```
# o default é em intervalos de 1.
seq(1,10,2)
[1] 1 3 5 7 9
```

Gerar sequências (usando : ou `seq`)

- A função `seq` funciona assim:

```
seq(from = 1, to = 10, by = 2 )
# sequência(de um, a dez, em intervalos de 2)
```

- `rep`
  - Vamos usar a função `rep` para repetir algo *n* vezes.

```
rep(5,10) # repete o valor 5 dez vezes
 [1] 5 5 5 5 5 5 5 5 5 5
```

- A função `rep` funciona assim:

```
rep(x, times=y) # rep(repita x, y vezes)
```

Ordenar e atribuir postos aos dados

- Funções: `sort`, `order` e `rank`
  - Primeiro vamos criar um vetor desordenado para servir de exemplo:

```
aves<-c(22,28,37,34,13,24,39,5,33,32)
```

- `sort`
  - A função `sort` coloca os valores de um objeto em ordem crescente ou em ordem decrescente.

```
sort(aves) # para colocar em ordem crescente
 [1]  5 13 22 24 28 32 33 34 37 39

# para colocar em ordem decrescente
sort(aves, decreasing=TRUE)
 [1] 39 37 34 33 32 28 24 22 13  5
```

Ordenar e atribuir postos aos dados

- `order`
  - A função `order` retorna a posição original de cada valor do objeto `aves` caso os valores do objeto `aves` sejam colocados em ordem crescente.

```
sort(aves)
 [1]  5 13 22 24 28 32 33 34 37 39

# retorna os valores da posição original
order(aves)
 [1]  8  5  1  6  2 10  9  4  3  7
```

- Note que o primeiro valor acima é 8, isso indica que se quisermos colocar o objeto `aves` em ordem crescente o primeiro valor deverá ser o oitavo valor de `aves`, que é o valor 5 (o menor deles).
- Na sequência devemos colocar o quinto valor do objeto `aves`, que é 13, depois o primeiro, depois o sexto . . . até que o objeto `aves` fique em ordem crescente.

Ordenar e atribuir postos aos dados

- `rank`
  - A função `rank` atribui postos aos valores de um objeto.

```
aves
 [1] 22 28 37 34 13 24 39  5 33 32
```

```
rank(aves)
 [1]  3  5  9  8  2  4 10  1  7  6
```

- Veja que 39 é o maior valor do exemplo, portanto recebe o maior rank, no caso 10.

Transformar vetores em matrizes

- Além de importar tabelas, existe opções juntar vetores em um arquivo dataframe ou matriz.
- Para criar uma matriz use `cbind` (colum bind) ou `rbind` (row bind).
- Vamos ver como funciona o `cbind`:

```
aa<-c(1,3,5,7,9)
bb<-c(5,6,3,8,9)
cc<-c("a","a","b","a","b")
cbind(aa,bb) # junta os vetores em colunas
     aa bb
[1,]  1  5
[2,]  3  6
[3,]  5  3
[4,]  7  8
[5,]  9  9
```

Transformar vetores em matrizes

- Vamos ver como funciona o `rbind`:

```
aa<-c(1,3,5,7,9)
bb<-c(5,6,3,8,9)
rbind(aa,bb) # junta os vetores em linhas
   [,1] [,2] [,3] [,4] [,5]
aa    1    3    5    7    9
bb    5    6    3    8    9
```

Transformar vetores em matrizes

- Lembre que matrizes podem conter apenas valores numéricos ou de caracteres.
- Por isso, se juntarmos o vetor `cc`, nossa matriz será transformada em valores de caracteres.

```
# junta os vetores em colunas, mas
# transforma números em caracteres.
cbind(aa,bb,cc)
     aa  bb  cc
[1,] "1" "5" "a"
[2,] "3" "6" "a"
[3,] "5" "3" "b"
[4,] "7" "8" "a"
[5,] "9" "9" "b"
```

Transformar vetores em matrizes

- Para criar uma dataframe com valores numéricos e de caracteres use a função `data.frame`:

```
data.frame(aa,bb,cc)
  aa bb cc
1  1  5  a
2  3  6  a
3  5  3  b
4  7  8  a
5  9  9  b
```

Criar vetores e matrizes

- Os comandos para se criar vetores e matrizes são:

```
A <- matrix(c(3, -1, 2, -2, 3, 1, 1, 4, 1, 4, 0, 3,
        0, 4, 0, 3), nrow=4,ncol=4, byrow=TRUE)
```

- Também é possível fazer as operações com matrizes da seguinte forma:

```
B <- matrix(c(4, 0, 3, 0, 4, 1, 3, 1,2, 4, 0,
        3, 6, 4, 0, 3), nrow=4,ncol=4, byrow=TRUE)
t(B) # transposta de B
     [,1] [,2] [,3] [,4]
[1,]    4    4    2    6
[2,]    0    1    4    4
[3,]    3    3    0    0
[4,]    0    1    3    3
```

Criar vetores e matrizes

- Também é possível fazer as operações com matrizes da seguinte forma:

```
det(B) # determinante de B
[1] -12

solve(B) # inversa de B
              [,1]          [,2]       [,3]       [,4]
[1,]  2.960595e-16 -2.960595e-16 -0.2500000  0.2500000
[2,]  3.000000e+00 -3.000000e+00  1.5000000 -0.5000000
[3,]  3.333333e-01  5.921189e-16  0.3333333 -0.3333333
[4,] -4.000000e+00  4.000000e+00 -1.5000000  0.5000000
```

Criar vetores e matrizes

```
C <- A+B # soma de matrizes
C
     [,1] [,2] [,3] [,4]
[1,]    7   -1    5   -2
[2,]    7    2    4    5
[3,]    3    8    0    6
[4,]    6    8    0    6

D <- A%*%B # produto de matrizes
D
     [,1] [,2] [,3] [,4]
[1,]    0   -1    6   -1
[2,]   42   21   12   16
[3,]   38   16   15   13
[4,]   34   16   12   13
```

Acessar partes da matriz de dados

- Agora vamos aprender a selecionar (extrair) apenas partes do nosso conjunto de dados *A* usando [ ] colchetes.
- O uso de colchetes funciona assim: [linhas, colunas], onde está escrito linhas você especifica as linhas desejadas, na maioria dos casos cada linha indica uma unidade amostral. Veja abaixo:

```
A
     [,1] [,2] [,3] [,4]
[1,]    3   -1    2   -2
[2,]    3    1    1    4
[3,]    1    4    0    3
[4,]    0    4    0    3

A[,1] # extrai a 1 coluna de A
[1] 3 3 1 0

A[,2] # extrai a 2 coluna de A
[1] -1  1  4  4
```

Acessar partes da matriz de dados

```
A[1,] # extrai a 1 linha de todas as colunas
[1]  3 -1  2 -2

A[3,3] # extrai a 3 linha e a 3 coluna, 1 valor
[1] 0

A[1,3] # extrai o valor da linha 1 e coluna 3
[1] 2

# extrair somente as linhas 1 a 4
# e as colunas 2 e 3
A[c(1:4),c(2,3)]
     [,1] [,2]
[1,]   -1    2
[2,]    1    1
[3,]    4    0
[4,]    4    0
```

Somar linhas e somar colunas

- Somar os valores de colunas ou linhas usando as funções `colSums` para somar colunas e `rowSums` para somar linhas.

```
A
     [,1] [,2] [,3] [,4]
[1,]    3   -1    2   -2
[2,]    3    1    1    4
[3,]    1    4    0    3
[4,]    0    4    0    3

# Note que estamos somando apenas os
# elementos das colunas 2 a 3
colSums(A[,2:3])
[1] 8 3

rowSums(A[1:4,])
[1] 2 9 8 7
```