Introdução ao Software R

Primeiros Passos

www.de.ufpb.br

www.youtube.com/channel/UC8QTeEyzHqYRjojKneTgLbA

UFPB Departamento de ESTATÍSTICA

Comandos de lógica

- Primeiro vamos ver o significado dos comandos abaixo.

>	Maior que	>=	maior que ou igual a
<	Menor que	<=	menor que ou igual a
==	igualdade	!=	diferença

```
x <- c(1,2,9,4,5)
y <- c(1,2,6,7,8)
x > y # Retorna TRUE para os maiores e FALSE para os menores
[1] FALSE FALSE  TRUE FALSE FALSE

x >= y
[1]  TRUE  TRUE  TRUE FALSE FALSE

x < y
[1] FALSE FALSE FALSE  TRUE  TRUE

x == y # Retorna TRUE para os x que são iguais a y
[1]  TRUE  TRUE FALSE FALSE FALSE

x != y # Retorna TRUE para os x que são diferentes de y
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

Comandos de lógica

- `which`
  - A função `which` funciona como se fosse a pergunta: Quais?

```
a<-c(2,4,6,8,10,12,14,16,18,20)
# Retorna um vetor contendo TRUE se for maior
# e FALSE se for menor
a > 10
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE

# Equivale a pergunta: "Qual a posição dos valores
# de a que são maiores que 10?".
which(a > 10)
[1]  6  7  8  9 10

# Para saber os valores de a maiores que 10:
a[which(a > 10)]
[1] 12 14 16 18 20
```

- Note que a resposta é a posição dos valores (o sexto, o sétimo . . .) e não os valores que são maiores que 10.

Comandos de lógica

- `ifelse`
  - A função `ifelse` significa: se for isso, então seja aquilo, se não, seja aquilo outro.
- O comando funciona da seguinte maneira: ifelse(aplicamos um teste, especificamos o valor caso o teste for verdade, e o valor caso falso). Por exemplo:

```
salarios<-c(1000, 400, 1200, 3500, 380, 3000,
            855, 700, 1500, 500)
```

- As pessoas que ganham menos de 1000 ganham pouco, concorda? Então aplicamos o teste e pedimos para retornar *pouco* (`pc`) para quem ganha menos de 1000 e *muito* (`mt`) para quem ganha de 1000 ou mais.

```
ifelse(salarios<1000,"pc","mt")
 [1] "mt" "pc" "mt" "mt" "pc" "mt" "pc" "pc" "mt" "pc"
```

Comandos de lógica

- `for`
  - O comando `for` é usado para fazer loopings, e funciona da seguinte maneira:

`for(i in 1:n){comandos}`

- Isso quer dizer que: para cada valor *i* o `R` vai calcular os comandos que estão entre as chaves {comandos}. O `i in 1:n` indica que os valores de *i* serão $i = 1$ até $1 = n$. Ou seja, na primeira rodada do for o $i = 1$, na segunda $i = 2$, e assim por diante até $i = n$. Para salvar os resultados que serão calculados no for, precisamos criar um objeto vazio que receberá os valores calculados.

```
resu <- numeric(0) # cria vetor de zeros
for(i in 1:5){
 resu[i] <- i^2
 } # Fim do for
resu ## Para ver os resultados
```

Criando funções

- A sintaxe básica é:

```
function(lista de argumentos){corpo da função}
```

- Vamos ver como criar funções começando por uma função simples, que apenas simula a jogada de moedas (cara ou coroa). Neste caso a função terá dois argumentos (*x* e *n*). *x* será a *moeda* (cara e coroa) e *n* será o número de vezes que deseja jogar a moeda.
- Vamos dar o nome a esta função de `jogar.moeda`.

```
jogar.moeda<-function(x,n){
 sample(x,n, replace=T)
 } # Fim da função
moeda<-c("Cara","Coroa")
jogar.moeda(moeda,2)
jogar.moeda(moeda,10)
jogar.moeda(moeda,1000)
```

Função `apply`

- Esta função é útil para aplicar outra função nas linhas ou colunas de um matriz ou vetor.

```
apply(x, margin, função, ...)
```

- Por exemplo, vamos aplicar a soma nas linhas de uma matriz X.

```
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)
soma.col <- apply(A, 2, sum) # soma nas colunas
soma.lin <- apply(A, 1, sum) # soma nas linhas
```

Função `apply`

- Observe que para o caso da soma de linhas ou colunas de uma matriz, o resultado é o mesmo das funções `colSums` e `rowSums` vistas anteriormente:

```
# soma nas colunas
soma.col
[1] 7 8 3 8

colSums(A)
[1] 7 8 3 8

# soma nas linhas
soma.lin
[1] 2 9 8 7

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

Função `na.omit`

- Muitas vezes, em alguns bancos de dados, se tem informações faltando. Quando isso ocorre, o `R` lê essas informações faltantes como NA. E na hora de analisar os dados a presença desses NAs pode causar erros.

```
na.omit(object, ...)
```

- Esta função retira os NAs para que a análise seja feita.

```
DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
DF
  x  y
1 1  0
2 2 10
3 3 NA
```

Função `na.omit`

```
na.omit(DF$y)
[1]  0 10
attr(,"na.action")
[1] 3
attr(,"class")
[1] "omit"

na.omit(DF)
  x  y
1 1  0
2 2 10
```