# Mini curso Introdução prática ao R

## Parte 1: Introdução ao R e Rstudio

Vinícius M. de Sousa

Universidade do Estado de Santa Catarina

O que são?

- R é uma linguagem de programação estatística;
- Rstudio é um ambiente que torna mais amigavél/produtiva a utilização do R.

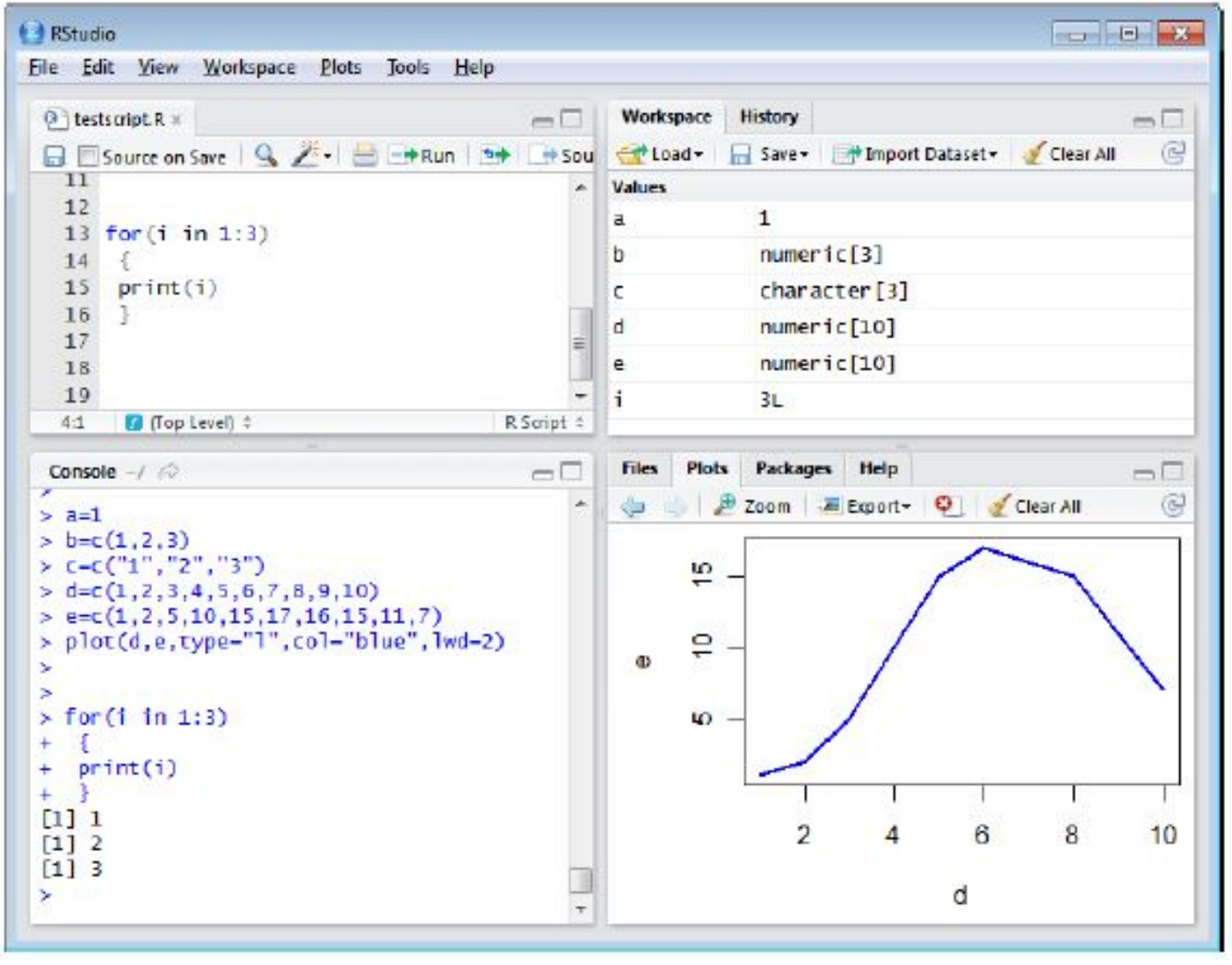
O que são?

- R é uma linguagem de programação estatística;
- Rstudio é um ambiente que torna mais amigavél/produtiva a utilização do R.

Onde "consegui-los"?

- R: disponível no site do R http://www.r-project.org
- Rstudio disponível no site do Rstudio http://www.rstudio.org

# Layout do RStudio



# Scripts

- R é um interpretador *command line based*;
- Digita-se os comandos ao invés de usar o mouse e menus;
- Os comandos digitados são salvos em arquivos chamados de *scripts*;
- *ctrl+shift+n* abre um novo script.

# Working Direcory

- Local no computador onde você está "trabalhando";
- Quando você pede para um arquivo ser carregado é no working directory que o arquivo é buscado;
- Quando você salva algum objeto é no working directory que ele fica;
- Comandos importantes:

```
setwd("F:/Users/Vinicius/Documents/Economia/Curso - Econometria no R/wd")

getwd()

## [1] "F:/Users/Vinicius/Documents/Economia/Curso - Econometria no R/wd"
```

# Pacotes

- As análises são feitas por meio de funções que são organizadas em *Pacotes*;
- Instalando e carregando pacotes

```
install.packages("nome_do_pacote")

library(nome_do_pacote)
```

# Calculadora

Operadores básicos do R

| Operação | Descrição |
|---|---|
| x + y | Adição |
| x - y | Subtração |
| x * y | Multiplicação |
| x / y | Divisão |
| x $\wedge$ y | Exponenciação |
| x %% y | Aritmético Modular |
| x %/% y | Inteiro da Divisão |
| x == y | Teste de Igualdade |
| x <= y | Teste de menor ou Igual |
| x >= y | Teste de maior ou Igual |
| x & y | Operador Boleano "E" |
| x \| y | Operador Boleano "OU" |
| !x | Operador Boleano de negação |

# Calculadora

Exemplos:

```
# Soma
2+2
```

```
## [1] 4
```

```
# Inteiro da divisão
14 %/% 5
```

```
## [1] 2
```

```
# Testes e Boleanos retornam vetores Lógicos
2 <= 3
```

```
## [1] TRUE
```

# Calculadora

## Atividade

Calcule o percentual da sua vida que você está nesta universidade (contando apenas os anos).

# Calculadora

```
# meu caso

((2017-2014)/(2017-1996))*100

## [1] 14.28571
```

# Workspace

Você pode salvar em variáveis seus cálculos e utilizar para computações futuras:

```
a <- 12

a/3

## [1] 4
```

Perceba que a variável *a* está no workspace.

# Workspace

Você pode salvar em variáveis seus cálculos e utilizar para computações futuras:

```
a <- 12

a/3

## [1] 4
```

Perceba que a variável *a* está no workspace.

## Atividade

Salvar com o nome de *a* o percentual calculade anteriormente.

# Funções

Meio de se automatizar um "processo" onde o usuário fornece os argumentos (inputs) e a função retorna um resultado (outputs). Exemplo:

```
rnorm(n = 4,mean = 2,sd = 1.2)

## [1] 1.9951730 2.7582913 0.8652806 1.0796933
```

Mostrar como ver os argumentos da função

# Funções

Podemos calcular a soma $3 + 4 + 5$ através da calculadora

```
3+4+5
```

```
## [1] 12
```

# Funções

Podemos calcular a soma $3 + 4 + 5$ através da calculadora

```
3+4+5
```

```
## [1] 12
```

Que claramente não é eficiente.

# Funções

Utilizando uma função...

```
a <- c(3,4,5)
a

## [1] 3 4 5

sum(x = a)

## [1] 12
```

# Funções

Utilizando uma função...

```
a <- c(3,4,5)
a

## [1] 3 4 5

sum(x = a)

## [1] 12
```

## Atividade

Gere uma sequência de 100 números aleatórios, com média 1 e desvio padrão 1, e salve-a como "x". Depois verifique com a função *mean*() que a média é próxima à que você colocou no argumento.

# Funções

```
x <- rnorm(n = 100,mean = 1,sd = 1)
mean(x)

## [1] 0.9336982
```

# Funções

Existem muitas funções prontas para serem usadas. Porém, **podemos escrever nossas próprias funções!**

# Ajuda e documentação

Digitando "?" seguido do nome da função ou pacote abre uma janela que descreve a função ou pacote;

## Atividade

Digite ?*mean*() no console.

# Ajuda e documentação

Grande comunidade da na internet. Por exemplo, digitando no google aparece sempre um link com a dúvida no *Stackoverflow*, Como criar uma matrix no R.

# Ajuda e documentação

Função *example*() é muito útil:

```
example(seq)

##
## seq> seq(0, 1, length.out = 11)
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
##
## seq> seq(stats::rnorm(20)) # effectively 'along'
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##
## seq> seq(1, 9, by = 2)     # matches 'end'
## [1] 1 3 5 7 9
##
## seq> seq(1, 9, by = pi)    # stays below 'end'
## [1] 1.000000 4.141593 7.283185
##
## seq> seq(1, 6, by = 3)
## [1] 1 4
##
```

# Tipos de dados

Existem várias maneiras de se armazenar informação, as principais delas são:

- Numéros - numeric;
- Caracteres textuais - strings;
- "Fatores" - factors (caracteres textuais com uma possilidade de finita, por exemplo tipo sanguíneo);
- Datas - dates.

# Tipos de dados

```
# função str() da um resumo do objeto
str(c(1,2,3))

##  num [1:3] 1 2 3

str(c('a','b','c','a'))

##  chr [1:4] "a" "b" "c" "a"

str(as.factor(c('a','b','c','a')))

##  Factor w/ 3 levels "a","b","c": 1 2 3 1

str(as.Date("2008-12-21"))

##  Date[1:1], format: "2008-12-21"
```

# Vetores

Estrutura básica de dados do R. Sequência com elementos que tem o **mesmo tipo**.

- Criando Vetores

```
a <- c(1,2,3,4,5) # com a função concatenar
a

## [1] 1 2 3 4 5

b <- seq(from = 1,to = 5,by = 1) # com a função seq
b

## [1] 1 2 3 4 5

c <- c('o','i','e')
c

## [1] "o" "i" "e"
```

# Vetores

## Atividade

Crie um vetor contendo caractere textual e número e depois verifique sua estrutura.

# Vetores

- Acessando elementos do vetor: utiliza-se "[i]" para acessar o i-ésimo elemento do vetor. Lembrando que a indexação começa em "1".

```
a <- seq(from = 1,length.out = 5,by = 1)

a[1:3] # pegando os 3 primeiros elementos

## [1] 1 2 3

a[length(a)] <- 2 # substituindo o ultimo elemento por 2
a

## [1] 1 2 3 4 2
```

# Vetores

- Operações: as funções do R são vetorizadas, isto quer dizer que ao aplicar uma função a um vetor a função será aplicada em cada elemento do vetor.

```
a <- c(1,4,9,16,25,36,49,64,81)
sqrt(a)

## [1] 1 2 3 4 5 6 7 8 9

a+1

## [1]  2  5 10 17 26 37 50 65 82
```

# Vetores

- Operações: as funções do R são vetorizadas, isto quer dizer que ao aplicar uma função a um vetor a função será aplicada em cada elemento do vetor.

```
a <- c(1,4,9,16,25,36,49,64,81)
sqrt(a)

## [1] 1 2 3 4 5 6 7 8 9

a+1

## [1]  2  5 10 17 26 37 50 65 82
```

## Atividade

O que acontece quando somamos vetores com comprimentos diferentes?
Some o vetor c(1,1,1,1) com o c(1,2)

# Matrizes

Nada além de vetores com 2 dimensões. Função *matrix*().

```
matriz_1 <- matrix(data = a,byrow = T,ncol = 3)
matriz_1

##      [,1] [,2] [,3]
## [1,]    1    4    9
## [2,]   16   25   36
## [3,]   49   64   81

x <- sqrt(a)
matriz_2 <- matrix(data = x,byrow = T,ncol = 3)
matriz_2

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

# Matrizes

Para operações matriciais coloca-se o operador entre "%". Fora isso as operações com matrizes são iguais às vetoriais.

```
matriz_1*matriz_2
```

```
##      [,1] [,2] [,3]
## [1,]    1    8   27
## [2,]   64  125  216
## [3,]  343  512  729
```

```
matriz_1%*%matriz_2
```

```
##      [,1] [,2] [,3]
## [1,]   80   94  108
## [2,]  368  445  522
## [3,]  872 1066 1260
```

# Matrizes

Acessa-se os elementos da matriz da mesma maneira.

```
matriz_1
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    9
## [2,]   16   25   36
## [3,]   49   64   81
```

```
matriz_1[c(1,2),2,drop = F]
```

```
##      [,1]
## [1,]    4
## [2,]   25
```

# Matrizes

## Atividade

Cria a matrix com o nome de "x"

$$\begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \end{pmatrix}$$

e depois salve em "y" somente os elementos da primeira linha.

# Matrizes

```
x <- matrix(data = c(11,12,13,14,21,22,23,24),
            nrow = 2,
            byrow = T)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   12   13   14
## [2,]   21   22   23   24
```

```
y <- x[1,,drop=F]
y
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   11   12   13   14
```

# Data frames

São no mesmo formato de matrizes, linhas e colunas de mesmo comprimento, porém apresenta duas vantagens principais:

- As colunas podem armazenar variveis com tipos de dados diferentes;
- Podemos acessar as variaveis pelos nomes das variaveis, sem saber exatamente qual a coluna ela se refere, através do "$".

```
df <- data.frame(nome=c('Ana','Beatriz','Carlos'),
                 idade=c(18,19,20),
                 sexo=c('F','F','M'))
df

##      nome idade sexo
## 1     Ana    18    F
## 2 Beatriz    19    F
## 3  Carlos    20    M
```

# Data frames

```
str(df)

## 'data.frame': 3 obs. of  3 variables:
##  $ nome : Factor w/ 3 levels "Ana","Beatriz",..: 1 2 3
##  $ idade: num  18 19 20
##  $ sexo : Factor w/ 2 levels "F","M": 1 1 2

df$idade

## [1] 18 19 20
```

# Data frames

Pode-se criar acrescentar colunas à um Data Frame já existente através do "$".

```
df$escolaridade <- c('Médio','Fundamental','Nenhuma')
df

##      nome idade sexo escolaridade
## 1     Ana    18    F        Médio
## 2 Beatriz    19    F  Fundamental
## 3  Carlos    20    M      Nenhuma
```

# Listas

"Coleção" de vetores, sendo sua principal vantagem:

- Os vetores não precisam ter o mesmo comprimento.

```
lista <- list(um=1,cinco=seq(from = 1,to = 5,by = 1),
              nomes=c('Ezequiel','Honolulu','Etc'))
str(lista)

## List of 3
##  $ um   : num 1
##  $ cinco: num [1:5] 1 2 3 4 5
##  $ nomes: chr [1:3] "Ezequiel" "Honolulu" "Etc"
```

# Listas

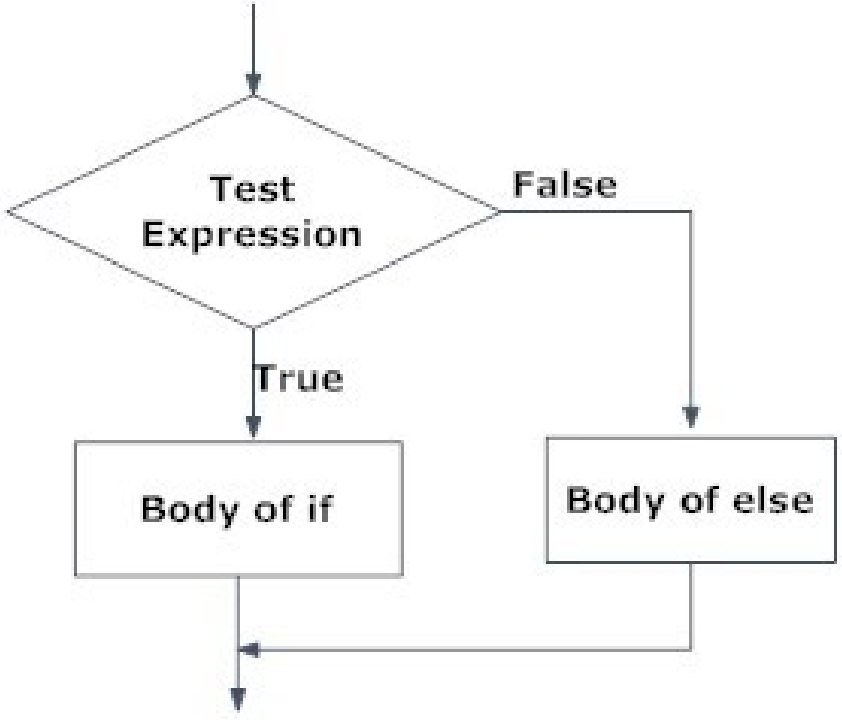
```
lista

## $um
## [1] 1
##
## $cinco
## [1] 1 2 3 4 5
##
## $nomes
## [1] "Ezequiel" "Honolulu" "Etc"

lista$nomes

## [1] "Ezequiel" "Honolulu" "Etc"
```

# IF - ELSE




Permite testarmos uma condição e dependendo do resultado tomar "caminhos diferentes".

# IF - ELSE

Implementando no R

```
if("teste"){
"ação caso o teste seja verdadeiro"
} else {
 "ação caso o teste seja falso"
}
```

# IF - ELSE

Exemplo 1: Verificar se o número é positivo ou negativo.

```
x <- -5

if(x>0){
  print("Positivo")
} else {
  print("Negativo")
}

## [1] "Negativo"
```

# IF - ELSE

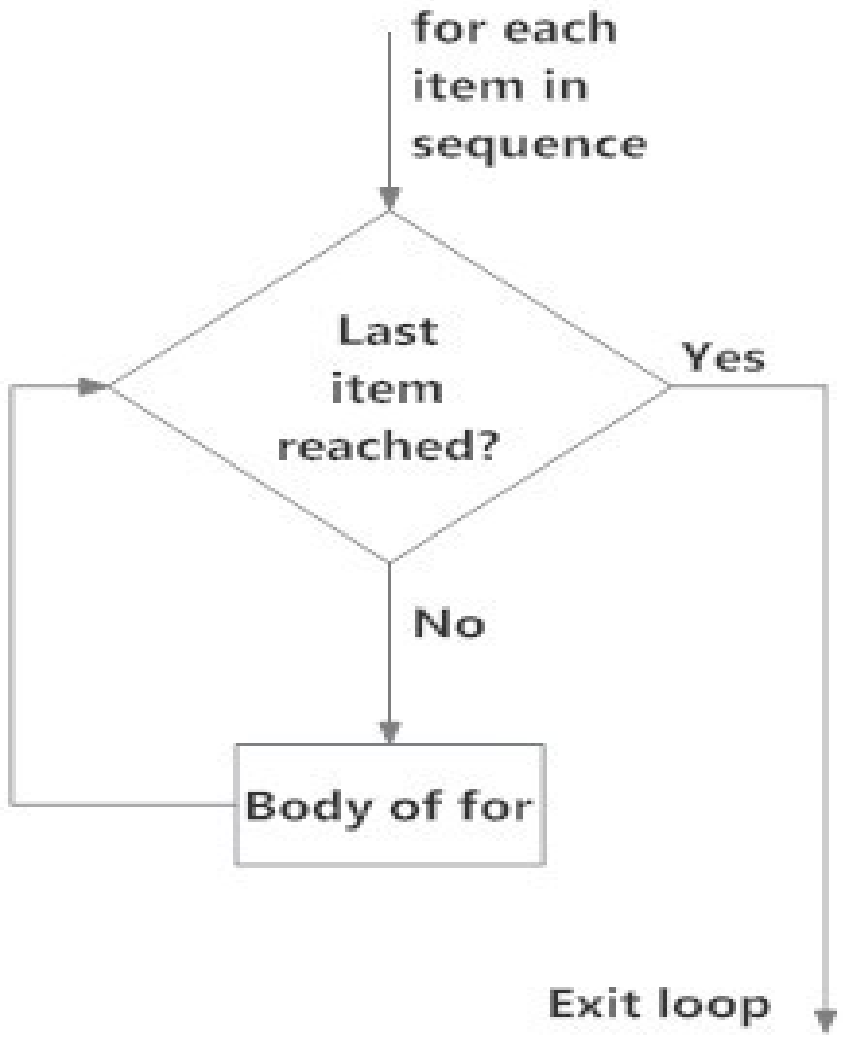
Exemplo 2: Verificar se o número é igual a 10 e atribuir o resultado a y.

```
x <- 10
y <- NULL
if(x/2 == 5){
  y <- "x é igual a 10"
} else {
  y <- "x é diferente de 10"
}
y

## [1] "x é igual a 10"
```

# For loops

Permite iterar sobre um objeto (vetor, coluna de matriz/data frame) e aplicar um mesmo procedimento para cada elemento



# For loops

Implementando no R

```
for(i in "sequencia"){
"ação para ser feita em cada elemento"
}
```

# For loops

Exemplo 1: Imprimir a sequência de 1 a 5.

```
for(i in 1:5){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

# For loops

Exemplo 2: dividir por 2 cada elemento do vetor c(1,2,3,4,5).

```
x <- seq(1,5,1)
for(i in 1:length(x)){
  x[i] <- x[i]/2
}
x

## [1] 0.5 1.0 1.5 2.0 2.5
```

# For loops

## Atividade

Crie um vetor c(5,4,3,2,1) e substitua cada elemento pela raiz quadrada do dobro do elemento.

# For loops

```
x <- 5:1

for(i in 1:length(x)){
  x[i] <- sqrt(x[i]*2)
}
x

## [1] 3.162278 2.828427 2.449490 2.000000 1.414214
```

# Testes dentro de loops

Podemos "enlaçar" os teste implementados a partir da estrutura IF - ELSE dentro dos loops. Por exemplo queremos saber se um número é positivo ou negativo

```
x <- data.frame(num = rnorm(5,0,1),
                sinal=rep(NA,5))


x

##          num sinal
## 1  0.2799573    NA
## 2 -0.2281846    NA
## 3  0.2256859    NA
## 4  0.5346357    NA
## 5  1.2107500    NA
```

# Testes dentro de loops

```
for(i in 1:nrow(x)){
  if(x[i,1] > 0){
    x[i,2] <- 'Positivo'
  } else {
    x[i,2] <- 'Negativo'
  }
}
x
```

```
##          num    sinal
## 1  0.2799573 Positivo
## 2 -0.2281846 Negativo
## 3  0.2256859 Positivo
## 4  0.5346357 Positivo
## 5  1.2107500 Positivo
```

# Testes dentro de loops

Função *ifelse*() é uma simplificação para tal "enlaçamento".

```
x <- rnorm(4,0,1)
x

## [1] -1.17138361  0.67077397  1.18150510  0.02112758

y <- ifelse(test = x > 0,yes = 'Positivo',no = 'Negativo')
y

## [1] "Negativo" "Positivo" "Positivo" "Positivo"
```

É útil para se criar novas variaveis.

# Testes dentro de loops

Função *ifelse*() é uma simplificação para tal "enlaçamento".

```
x <- rnorm(4,0,1)
x

## [1] -1.17138361  0.67077397  1.18150510  0.02112758

y <- ifelse(test = x > 0,yes = 'Positivo',no = 'Negativo')
y

## [1] "Negativo" "Positivo" "Positivo" "Positivo"
```

É útil para se criar novas variaveis.

## Atividade

Crie um data frame com uma coluna contendo 9 números aleatórios e média 0. Feito isso crie uma coluna onde cada linha da coluna indica se o número referente a linha é positivo ou negativo. (Dica lembre-se como podemos criar novas variaveis em data frames)