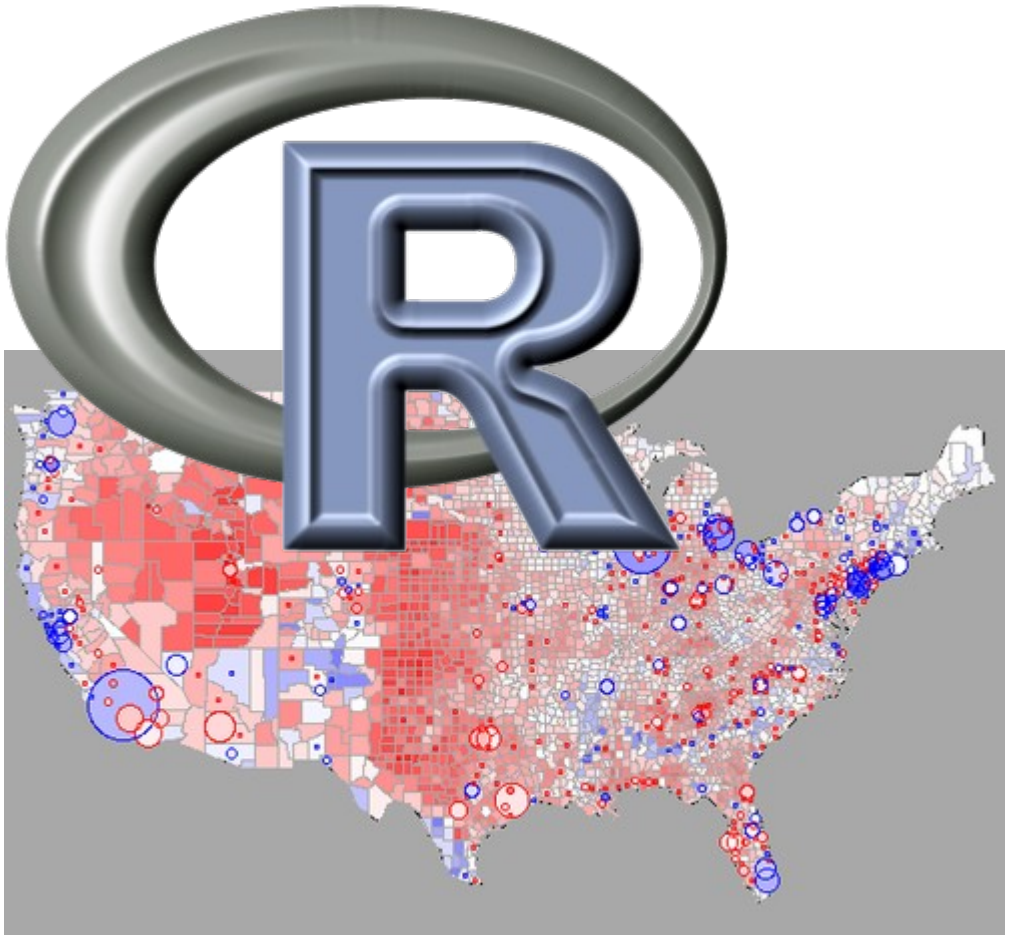
FIAP

PÓS-GRADUAÇÃO



MBA em Big Data

Introdução à Linguagem R

Encontro 3

Prof. Antonio Henrique Pinto Selvatici

antoniohps@gmail.com

Versão 1 – 10/2014

Professor

- **Antonio Henrique Pinto Selvatici**

- É engenheiro eletrônico formado pelo Instituto Tecnológico de Aeronáutica (ITA), com mestrado e doutorado pela Escola Politécnica (USP), e passagem pela Georgia Institute of Technology em Atlanta (EUA). Desde 2002, atua na indústria em projetos nas áreas de robótica, visão computacional e internet das coisas, aliando teoria e prática no desenvolvimento de soluções baseadas em Machine Learning, processamento paralelo e modelos probabilísticos, tendo desenvolvidos projetos para Avibrás, Rede Globo, IPT e Systax. Foi professor do curso de Ciência da Computação da Uninove de 2009 a 2013. Em 2012, tendo ajudado a fundar a Selsantech, participou do desenvolvimento do CatSearch, uma solução para Data Mining preparada para o paradigma MapReduce. É professor do MBA do curso de Big Data da FIAP e trabalha na reformulação do sistema de pagamento on-line eWally.

Resumo

- Introdução à programação em R
  - Executando um script
  - Estruturas de controle de fluxo
- Mais estruturas de dados do R
  - Matriz
  - Listas
- Lista de exercícios

4. Introdução à programação em R

1 – Scripts do R

- Scripts são arquivos de texto contendo uma sequência de expressões
- Cada expressão é executada geralmente em uma linha, mas caso o interpretador R perceba uma expressão incompleta, tentará preencher com o código da linha seguinte, até encontrar um erro ou finalizar a expressão
- Não há um finalizador da expressão (como `;` no C)
- O caractere # indica um comentário de linha
- O scripts R em geral levam a extensão .R
- Executando um script:
  - Da linha de comando do shell:
    - `R CMD BATCH [options] infile [outfile]`
    - `infile`: script a ser excutado
    - `outfile`: arquivo que guarda a saída do R
  - De dentro do console
    - `source(<nome-do-arquivo>)`
    - `source(file.choose()) # escolhe o arquivo a executar`

2 – Hello world!

- Para mostrar mensagens de saída, usamos a função `cat()`:
  - `cat("Hello world! ")`
  - `i <- 3`
  - `cat("i =", i)`
- Para pedir dados de entrada do teclado, usamos `scan()` ou `readline()`
- `scan()` lê um vetor de entrada, parando quando a leitura é vazia
  - `var <- scan(what=<type>)`
    - `x <- scan(what=double()) # padrão`
    - `texto <- scan(what="")`
- `readline()` lê um texto de entrada, sem fazer conversão, permitindo um texto de prompt
  - `x <- readline("Olá, qual é seu nome? ")`

3 – Estruturas de controle

- `if()...else:` Determina as expressões a serem executadas dependendo da condição ser TRUE ou FALSE
- `ifelse()`: Vetorização do `if()…else`
- `switch()`: Executa uma expressão diferente dependendo de um valor fornecido
- `for()`: Repete um conjunto de expressões n vezes
- `while()`: Repete um conjunto de expressões até que a condição seja FALSE
- `repeat()`: Repete um conjunto de expressões até que seja interrompido
- `break`: Sai do laço de repetição
- `next`: Para o processamento do laço e continua a próxima iteração
- `?Control`: mostra a documentação do R sobre as estruturas de controle

if()...else

- Uso (condição deve resolver para um valor lógico):
  - `if(condição) expressão #expressão deve ser de apenas uma linha`
  - ```
    if(condição) {
      expressões
    } #expressões são executadas se a condição for TRUE
    ```
  - ```
    if(condição) {
      expressões 1 #são executadas se a condição for TRUE
    } else { # else deve estar na mesma linha de }
      expressões 2 # são executadas se condição for FALSE
    }
    ```
- Exemplo: dizer se um número inteiro é par ou ímpar
  - `cat('Digite um número inteiro:')`
  - `x <- scan(what=integer())`
  - `if(x%%2 == 0) {`
  - `  cat(x,"é par")`
  - `} else {`
  - `  cat(x, "é ímpar")`
  - `}`

if()...else >> expressões lógicas

Expressão	Significado
!x	NOT x
x & y	x AND y (compara elemento a elemento)
x && y	x AND y (retorna um único valor)
x | y	x OR y (compara elemento a elemento)
x || y	x OR y (retorna um único valor)
x %in% y	x ESTÁ CONTIDO EM y
x > y, x < y	x É MAIOR QUE/MENOR QUE y
x <= y, x >= y	x É MENOR OU IGUAL/MAIOR OU IGUAL y
x == y, x ! = y	x É IGUAL A/DIFERENTE DE y
isTRUE(x)	testa se x é TRUE
all(...)	testa se todos os argumentos são TRUE
any(...)	testa se algum argumento é TRUE
identical(x,y)	Retorna apenas TRUE ou FALSE, sem o risco do NA

ifelse()

- Retorna um valor com a mesma estrutura que o elemento testado, preenchida com os argumentos SIM ou NÃO dependendo do teste de cada valor
- Corresponde à vetorização do famoso operador ternário do C e Java
- Uso: `ifelse(teste, SIM, NÃO)`
  -
```
x <- 1:10
ifelse(x > 5, "Maior", "Menor")
[1] "Menor" "Menor" "Menor" "Menor" "Menor" "Maior"
 "Maior" "Maior" "Maior" "Maior"
```
  -
```
x <- data.frame(valor=seq(10,100,by=10))
y <- ifelse(x > 65, "Passou", "Reprovou")
```

switch()

- A famosa estrutura de seleção, mas no formato de função.
- Uso: `switch(teste, val1=EXPR1, val2=EXPR2, … , DEFAULT)`
- Se `teste` for igual a `val1`, `val2`, etc., a expressão correspondente é retornada. Caso contrário, o retorno é `DEFAULT`.
- Exemplo:
  - `cat("Entre com dois valores a serem avaliados”)`
  - `x <- scan()`
  - `cat (“Entre com o símbolo da operação”)`
  - `op <- scan(what=“”)`
  - `res <- switch(op, “+”=x[1]+x[2], “-”=x[1]-x[2], “*”=x[1]*x[2], “/”=x[1]/x[2], NA)`
  - `cat(res)`

Laço for()

- Executa as expressões entre chaves para cada valor dentro de um vetor
-
```
for(var in valores) {
     expressions
}
```
- var é o nome da variável que recebe os valores do vetor valores (um a cada iteração do laço)
- Por padrão, o ambiente do R buferiza a saída criada dentro de um laço. Para mostrar a saída a cada iteração, chame flush.console() no fim do laço
- Exemplo: somar os números de 1 a 100

```
soma <- 0
for(i in 1:100) {
soma <- soma+i
}
cat("Resultado ->",soma,"  sum(1:100) ->", sum(1:100))
```

Laço `while()`

- Executa as expressões enclausuradas enquanto a `condição` for TRUE
-
```
while(condição) {
     expressions
}
```
- `condição` deve ser um valor lógico simples (não pode ser NA)
- Exemplo: somar os números de 1 a 100

```
soma <- 0
i <- 1
while(i <= 100) {
  soma <- soma+i
  i <- i + 1
}
cat("Resultado ->",soma,"  sum(1:100) ->", sum(1:100))
```

Laço repeat

- Só para a execução quando `break` é executado
- ```
  repeat {
       expressions
       if(cond) break
  }
  ```
- Exemplo: somar os números de 1 a 100

```
soma <- 0
i <- 1
repeat {
  soma <- soma+i
  i <- i + 1
  if(i > 100) break;
}
cat("Resultado ->",soma,"  sum(1:100) ->", sum(1:100))
```
```

4 – Funções do R

- Funcões do R são objetos executáveis que recebem argumentos e podem retornar outro objeto
- Declarar uma função é simples:
- ```
  <nome-da-função> <- function(<lista-de-argumentos>) {
      expressões
  }
  ```
- A lista de argumentos é uma lista de argumentos formais separadas por vírgula, podendo ser declarada como
  - `nome.do.argumento`
  - `nome.do.argumento=<valor-padrão>`
  - `...` # lista de argumentos que não coincidem com nenhum argumento formal
- Cuidado com nome da função, pois pode sobrescrever alguma função do R!
  ```

4 – Funções do R >> Exemplo

```
imprimeseq <- function(ini=0, fim, passo=1) {
  cat(seq(ini,fim,passo))
}
imprimeseq(1,20,5)
imprimeseq(fim=10)
```

- O retorno da função é o resultado da última expressão executada

```
soma5 <- function(num) {
  num + 5
}
soma5(3)
x <- soma5(10)
cat("x vale",x)
```

5 – Funções do R >> Retornando objetos

- Caso não queiramos esperar a execução da última linha, há duas funções para o retorno de objetos
  - return(): retorna seus argumentos, imprimindo-os na tela caso não sejam atribuídos a alguma variável
  - invisible(): semelhante a return(), não imprime o resultado caso eles não sejam atribuídos
- Exemplo:

```
ehpar <- function(n) {
  n <- as.integer(n)
  if(n%%2 == 0) {
    return(T);
  } else {
    return(F)
  }
}
```

5 – Funções do R >> Escopo de variáveis

- Quando variáveis são criadas, elas são alocadas dentro de um “ambiente”. É um conceito mais complexo do que o simples escopo, pois o ambiente guarda uma tabela com os nomes das variáveis e seus valores
- Variáveis atribuídas durante uma sessão do R são armazenadas no ambiente globa
- Variáveis atribuídas dentro do corpo de uma função, incluindo seus argumentos reais, são guardads dentro de um abiente específico daquela função, que pode também acessar o escopo global
- Se uma variável no ambiente local da função tiver o mesmo nome de outra no ambiente global, o R sempre dará prioridade à variàvel local.

5 – Funções do R >> Verificando argumentos

- Se tivermos valores inválidos de argumentos, provavelmente queremos parar a execução e dar uma mensagem de erro
- A função `nargs()` retorna o número de argumentos passados para a função atual
- A função `missing()` pode ser usada para testar se um valor foi especificado naquele argumento, retornando `TRUE` se tiver sido omitido ou `FALSE` se tiver sido especificado
- A `função stop( <mensagem> )` para a execução e emite uma mensagem de erro contendo `mensagem`
- A função `warning()` gera uma mensagem de aviso
- A função `message()` gera uma mensagem de diagnóstico, que são suprimidas quando a função é chamada dentro de `suppressMessages()`
- `stopifnot( <condições> )` funciona como um atalho para
  - `if(!all( <condições> )) stop()`

5 – Funções do R >> Verificando argumentos

```
ehpar <- function(n) {
  if(missing(n)) {
    stop("Falta especificar o parâmetro n")
  }
  if(!is.integer(n)) {
    warning("O parâmetro n será convertido para inteiro")
    n <- as.integer(n)
  }
  message(paste("O número de argumentos é",nargs()))
  if(n%%2 == 0) {
    return(T);
  } else {
    return(F)
  }
}
```

5. Outras estruturas de dados do R

1 – Matrizes

- Uma matriz é a generalização bidimensional do vetor
- Para criar uma matriz, usar `matrix()`:
  - `matrix(data=NA, nrow=1, ncol=1, byrow = FALSE, dimnames = NULL)`
    - » `data`: um vetor que fornece os valores da matriz
    - » `nrow`: número de linhas desejadas
    - » `ncol`: número de colunas desejadas
    - » `byrow`: se FALSE (default), a matriz é preenchida por colunas, se TRUE, por linhas
    - » `dimnames`: lista com dois vetores, contendo os nomes das linhas e colunas, respectivamente

1 – Matrizes >> Exemplo

```
x <- matrix(1:12, nrow=3, ncol=4, byrow=T,
dimnames=list(
          rows=c("l1", "l2", "l3"),
          cols=c("c1", "c2", "c3", "c4")))
x
rownames(x)
colnames(x)
rownames(x) <- c("Zezinho", "Huguinho", "Luisinho")
x
x <- matrix(1:12, nrow=3, ncol=4, byrow=F,
dimnames=list(rownames(x),colnames(x)))
x
```

2 – Acessando elements de uma matriz

- Podemos fazer exatamente como o acesso a elementos de data frames: através da notação `<matriz>[linhas,colunas]`, onde `linhas` e `colunas` podem ser vetores numéricos ou de texto
  - Atenção: a notação `<matriz>$<nome-da-coluna>` não serve para matrizes! Apenas para data frames e listas!
    - `x[2,3]`
    - `x["Huguinho",2:3]`
    - `x["Zezinho",c("c3","c2")]`

3 – Operações com matrizes

- Quando matrizes são usadas com operadores unários ou binários, essas operações são aplicadas elemento a elemento
- A álgebra linear define operações importantes entre vetores e matrizes, como multiplicação entre matrize, transposição, produto interno, produto vetorial, etc. R apresenta implementações para essas operações
- Multiplicação de matrizes ou produto interno de vetores: operador `%*%`
- Produto vetorial entre vetores: `%o%`
- Transposição de matrizes: `t(<matriz>)`
- Exemplo:
  - `M1 <- matrix(1:6, ncol=3)`
  - `M2 <- t(M1)`
  - `M1*M2 # matrizes não são recicladas como vetores`
  - `M1%*%M2`

3 – Operações com matrizes >> outras

FIAP

- Álgebra linear
  - `det(M)`: Determinante de M
  - `solve(A, b)`: Resolve a equação Ax=b, retornando x
  - `solve(M)`: Matriz inversa de M
  - `eigen(M)`: Autovalores e auto vetores de M
  - `diag(n)`: Cria uma matriz identidade n por n
  - `diag(M)`: Retorna os elementos da diagonal de M
  - `diag(x)`: Cria uma matriz diagonal a partir do vetor x
- Funções auxiliares
  - `apply()`: Aplica uma função às linhas ou colunas da matriz
  - `rbind()`: Combina os argumentos como linhas de uma matriz
  - `cbind()`: Combina os argumentos como colunas de uma matriz
  - `dim(M)`: Retorna ou ajusta as dimensões da matriz M
  - `nrow(M), ncol(M)`: Número de linhas/colunas da matriz M

4 – Avaliando matrizes

- A função `apply()` é usada para aplicar funções que porcessam linhas, colunas ou todos os dados da matriz de cada vez
- Pode ser aplicada também a data frames
- Uso: apply(X, MARGIN, FUN, ...)
  - X: a matriz, array or dataframe a ser processada
  - MARGIN: O valor do índice (ou índices) indicando as dimensões serem processadas:
    - » 1 : por linhas
    - » 2 : por colunas
    - » c(1,2) : por linhas e colunas
  - FUN: função a ser applicada
  - ... : Argumentos opcionais para FUN
- Exemplo:
  - `x <- matrix(1:12, nrow=3))`
  - `apply(x, 1, sum)`
  - `apply(x, 2, sum)`

5 – Listas

- Uma lista é como um vetor, mas seus elementos podem ser de tipos variados.
- Os elementos de uma lista podem ser nomeados, assim como as colunas de um data frame
- Para criar uma lista usamos a função `list(...)`, que encadea os argumentos
  - Os argumento podem ser da forma `name=value` ou sem os nomes
  - `x <- list(seq=1:5, "Olá", identity=diag(5))`
- Para acessar elementos da lista podemos usar a notações `[ ]`, `[[ ]]` ou `$`.
  - `x[[2]]` # Segundo elemento da lista x
  - `x[["identity"]]` # Elemento de nome “identity”
  - `x$identity` # Elemento de nome “identity”
  - `x[3]` # Cria uma sublista com terceiro elemento
  - `x[1:2]` # Cria uma sublista com os dois primeiros elementos

6 – Funções úteis para listas

- `lapply()`: Retorna uma lista onde cada elemento é resultado da aplicação de uma função à cada elemento da lista de entrada
- `sapply()`: Retorna um vetor, matriz, ou lista (nesta ordem de preferência) onde cada elemento é resultado da aplicação de uma função à cada elemento da lista de entrada
- `vapply()`: Similiar a sapply(), mas pode nomear os resultados
- `replicate()`: Retorna um vetor, matriz, ou lista (nesta ordem de preferência) onde cada elemento é resultado da execução de uma expressão
- `unlist(x)`: Produz um vetor com todos os componentes de `x`
- `length(x)`: Retorna o número de objetos na lista `x`
- `names(x)`: Retorna ou atribui os nomes aos elementos da lista `x`

7 – Exemplo de uso de `lapply()` e similares

- Geração de uma lista x com 6 vetores
  - `x <- lapply(5:10, seq)` # o vetor 5:10 é transformado internamente em lista
- Calcula as média dos vetores na lista x
  - `lapply(x, mean)`
  - `sapply(x, mean)`
- Gera repetidos resultados da chamada a uma expressão
  - `replicate(5,runif(4))`

Lista de exercícios 3

- Faça um script do R que implemente a função fib(n), que retorna um vetor com a sequência de Fibonacci de 1 até n. A sequência de Fibonnaci é definida como f(1) = 1, f(2) = 1 e f(n) = f(n-1) + f(n-2) para n > 2
- Modele o seguinte probema na forma de um sistema de equações e resolva através da função solve do R. Escreva a solução na forma de um Markdown do R. **Problema:** Zezinho tem 5 pedras a mais do que Huguinho. Luisinho tinha 10 pedras antes de dar a Zezinho metade do que este tem hoje. Huguinho tem 3 pedras a menos do que Luisinho. Quantas pedras tem cada um?
- Faça um script do R analisador da bolsa de valores. Faça uma função que receba um data frame com as mesmas colunas daquele gerado por COTAHIST.A1997 e um vetor de códigos de ação. A função deve gerar a plotagem dos papeis ao longo dos dias, a plotagem das correlações dos dados históricos de preço de fechamento (PREULT) dos papéis, e o gráfico de pizza dos volumes negociados no ano daqueles papeis.

Referências

- Norman Matloff. *The Art Of R Programming*. No Starch Press, São Francisco, CA, 2011.
- Joseph Adler. *R in a Nutshell*. O'Reilly Media, Inc., Sebastopol, CA, 2012
- Mark Gardener. Beginning R: The Statistical Programming Language. John Wiley & Sons, Indiana, IN, 2012.

FIAP

A MELHOR FACULDADE DE TECNOLOGIA