- Tutorial
- Exercícios
- Apostila

3. Tutoriais de Leitura e Manipulação de dados no R

Os 10 mandamentos do R Agora que já tem alguma experiência com a linguagem, veja os 10 mandamentos do R! Ao final do curso revisite esses mandamentos e avalie quantos pecados deve confessar.

- **Os 10 Mandamentos do R**

Objetos de Dados

Vetores como estrutura básica de dados

Video

Nos tutoriais anteriores, nós usamos uma classe de objetos que é a estrutura básica de armazenamento de dados no R, a classe de objetos vetores. Vamos revisitar alguns conceitos neste tópico, criando um vetor de dados numéricos, chamado `num`, que seja uma sequência de valores de 0 a 15 divididos em 7 valores equidistantes.

| Criando Vetor

```
num <- seq(0, 15, len=7)
```

Agora vamos olhar os valores do objeto criado, em seguida elevar cada valor ao quadrado, atribuindo

o resultado ao objeto “num2”. Em seguida tire a raiz-quadrada desse vetor.

Operando vetor

```
num
length(num)
class(num)
num2 <- num^2
num2
sqrt(num2)
num
```

Agora vamos extrair apenas o quinto elemento e em seguida substituir esse elemento pela palavra “quinto_elemento”. Em seguida vamos repetir as operações anteriores:

Quinto elemento

```
num[5]
num[5] <- "quinto_elemento"
num
num[-5]

length(num)
class(num)
num^2
sqrt(num^2)
```

O que aconteceu aqui? Por que não conseguimos mais operar o vetor `num`? Vamos agora voltar o vetor para sua forma original e operar novamente:

Retorno do elemento

```
num[5] <- 10
num
num^2
sqrt(num^2)
```

Por que ainda não conseguimos? Vamos investigar o vetor e ajustá-lo.

| Classe do vetor

```
length(num)
class(num)
num <- as.numeric(num)
sqrt(num^2)
```

Como já havíamos visto, uma das característica do vetor é que **só armazena um tipo de natureza de dados** e o R faz a coerção da classe do objeto dependendo da característica desses dados. Incluir algo que não é dígito ou símbolo de decimal em um objeto da classe numérica faz com que a classe seja convertida automaticamente em `character`.

A raiz de um valor elevado ao quadrado é o próprio valor? O que aconteceria se o vetor fosse de números negativos? Veja a discussão sobre função quadrática e raiz quadrada como funções inversas nesse link

Leitura de Dados

A principal função para a leitura de dados no R é `read.table`. Ela é bem flexível e se aplica para a leitura de dados tabulares como uma planilha eletrônica usual, tendo colunas como variáveis e as linhas como observações. Esta estrutura é análoga a um conjunto de vetores lado a lado, de mesmo comprimento, como veremos a seguir. Antes da leitura de dados é importante garantir que temos eles bem organizados em uma planilha. O artigo Data Organization in Spreedsheets (Broman & Woo, 2018) faz uma ótima síntese de boas práticas para estruturar dados brutos em uma planilha, sua leitura é rápida e irá poupar muito tempo futuro e evitar muitos erros comuns que usuários de planilhas cometem. Os exemplos de erros comuns em planilhas de ecologia do datacarpentry são também muito bons, uma forma interessante de aprender é ser exposto ao que não devemos fazer.

Tendo a planilha eletrônica com os dados brutos bem estruturados, precisamos exportá-la como arquivo texto puro para fazer a leitura no R. Os arquivos de texto são uma forma eficiente de armazenar dados que tem uma estrutura simples de linhas e colunas. Além de poderem ser abertos em qualquer programa simples de texto e sistema operacional, são reconhecidos nas planilhas eletrônicas como estrutura de dados. Normalmente, utilizamos as extensões `.txt` ou `.csv` para designar arquivos texto com campos de dados separados por tabulação e vírgula, respectivamente[1].

Ao exportar os dados deve ficar atento para algumas opções de exportação da planilha, as principais são os caracteres para designar a separação de campo e o símbolo de decimal. Evite, sempre que possível, caracteres especiais como acentos e aspas (`'` , `` ` ``, “), se houver a opção de escolher a codificação de caracteres (“enconding”) opte pelo UTF-8.

Sabendo o formato que os dados foram salvos no arquivo texto, na maioria dos casos, precisamos apenas do seguintes argumentos para fazer a leitura dos dados no R [2]:

Principais argumentos do `read.table`

Argumento	Descrição	Padrão	Alternativas
file	nome do arquivo [3]	`"nome_arquivo.txt"`	`"/caminho_dir/nome_arquivo.txt"`
header	nome das variáveis [4]	FALSE	TRUE

Argumento	Descrição	Padrão	Alternativas
sep	separador [5]	" "	"," ";" "\t"
dec	símbolo de decimal	"."	","
as.is	mantenha caracteres [6]	TRUE	FALSE

Verificando a leitura

Logo após a leitura dos dados é recomendável fazer a verificação do objeto lido para ter certificar que os dados foram lidos corretamente. É possível retornar na tela todo o objeto criado, mas para conjunto de dados grandes, fica difícil a visualização. Para isso, usamos algumas funções acessórias que extraem atributos ou parte dos dados. As principais são:

- `str`: faz uma síntese da estrutura do objeto;
- `dim`: mostra as dimensões do objeto, primeiro o número de linhas (observações) e em seguida colunas (variáveis);
- `head`: mostra as primeiras linhas, por padrão as primeiras 6 observações;
- `names`: mostra o nome das variáveis (colunas).

Dados de Caixeta

Entre na página levantamento de espécies em caixetais, leia a descrição dos dados e das variáveis e salve o arquivo de dados no diretório de trabalho.

Obs.: Se o arquivo abrir em uma aba do navegador, clique com o botão direito do mouse no link e selecione “salvar link”.

Vamos fazer a leitura do arquivo com o padrão de leitura da função `read.table` e verificar em seguida a classe e a formatação do arquivo:

caixeta

```
caixeta <- read.table(file = "caixeta.csv")
```

Esse código retornar o erro:

Lendo caixeta?

```
Error in scan(file = file, what = what, sep = sep, quote = quote, dec = 
dec,  : 
  line 5 did not have 3 elements
```

Indicando que tentou ler o arquivo, mas algumas linhas tinham dimensões diferentes, ou seja, o formato não era tabular. Isso ocorre, normalmente, devido ao tipo de separador de campo utilizado

no arquivo que não é espaço, o padrão da função. No caso, o separador é `,` como relatado na documentação dos dados. Vamos usar o argumento `sep` para indicar o separador. Em seguida vamos verificar a classe do objeto.

Relendo caixeta

```
caixeta <- read.table(file = "caixeta.csv", sep = ",")
class(caixeta)
```

Leitura de dados Online

A função `read.table` também funciona com endereço web como caminho para leitura de arquivos. Só precisamos fornecer a url do arquivo. No nosso caso, como os dados estão no nosso wiki, podemos usar:

Caixeta url

```
caixeta <- read.table(file =
"http://ecor.ib.usp.br/lib/exe/fetch.php?media=dados:caixeta.csv", sep = ",")
```

Data Frames

A classe do objeto é um `data.frame`. Esse é o segundo tipo de objeto para armazenar dados no R que apresentamos: o `vetor` tem apenas uma dimensão de dados, o `data.frame` tem duas. Os datas frames no R são conjuntos de vetores de mesmo tamanho, similares a uma planilha de dados. Todas as características que vimos do objeto `vetor`, valem para as colunas do `data.frame`. Uma outra forma de pensar o data frame é que são variáveis (colunas) de um conjunto de observações (linhas). Assim como deve ser uma planilha de dados bem estruturada.

Vamos insistir em uma sugestão importante, que evita muitos transtornos para quem está iniciando no R é: depois de efetuar a leitura **sempre verificar a estrutura dos dados antes de iniciar as análises**, usando as funções indicadas acima.

Verifica caixesta

```
dim(caixeta)
names(caixeta)
str(caixeta)
head(caixeta)
```

Apesar do objeto ter sido lido, ele não parece o que deveria. A dimensão do objeto mostra que tem uma observação a mais, os nomes das variáveis não corresponde ao que está definido no metadado e as variáveis foram todas lidas com `characters` ou `factor` dependendo da versão do R. O head mostra exatamente o que aconteceu: a linha com os nomes das variáveis foi lida como sendo uma observação. Com isso, todas as variáveis foram classificadas como caracteres. Vamos agora usar o argumento `header` e fazer novamente a verificação:

Caixeta header

```
caixeta <- read.table(file = "caixeta.csv", sep = ",", header = TRUE)
dim(caixeta)
names(caixeta)
str(caixeta)
head(caixeta)
```

No código acima temos resultados que podem variar dependendo da versão do R. Isso não é muito comum, pois a equipe de desenvolvedores busca manter a compatibilidade dos scripts entre versões.

stringsAsFactors

Por padrão, até a versão anterior a 4.0.0 de abril de 2020, o padrão das funções `read.table` e `data.frame` era classificar as variáveis com caracteres como sendo um fator. Isso era definido com os padrões dos argumentos `stringsAsFactors = TRUE` ou `as.is = FALSE`. Desde da versão 4.0.0 o padrão é classificar as variáveis que contém caracteres como sendo `character`. Essa conversão automática para fator é um legado da linguagem S[7].

Incluir Variáveis

Vamos então transformar as variáveis `local` e `especie` para fator, assim independente da versão do R instalada, estaremos todos com a mesma estrutura de objeto.

Fatores caixeta

```
caixeta$local <- factor(caixeta$local)
caixeta$ssp <- factor(caixeta$especie)
str(caixeta)
```

Percebam que na primeira linha de comando estamos sobrescrevendo a variável `local` como fator para a mesma coluna. No caso da conversão de `especie` em fator, criamos uma nova variável no nosso `data.frame` chamada `spp`, um fator, e preservamos a variável `especie` que continua como caracteres.

Incluir Variáveis no `data.frame`

Para incluir novas variáveis no `data.frame` só é preciso fornecer os valores e indicar a coluna onde esses valores devem ser atribuídos, tomando o cuidado de indicar um nome diferente das variável existentes caso não queira sobrescrever. O vetor da nova variável deve ter o mesmo tamanho do número de linhas do `data.frame`. A regra de ciclagem vale aqui também, veja o que acontece com o código:

Ciclando NA

```
caixeta$dap <- NA
str(caixeta)
```

No quadro acima criamos uma variável chamada `dap` que contém só NAs (not available). Vamos incluir agora, nessa coluna, os valores de diâmetro à altura do peito. Para isso, vamos usar o valor de `cap` (circunferência à altura do peito) e a relação:

$$ dap = \frac{cap}{pi} $$

Cap para dap

```
caixeta$dap <- caixeta$cap/pi
str(caixeta)
```

Aqui também temos a regra da ciclagem com o valor da constante `pi` sendo ciclado para operar todos os elementos da variável `cap`.

Indexação

Acima usamos a indexação da coluna do `data.frame` pelo nome da variável, usando o operador $. Veja o que temos ao pedir a estrutura de uma variável do `data.frame`

Estrutura da variável

```
str(caixeta$especie)
str(caixeta$cap)
```

O que temos é a estrutura básica de armazenamento de dados: vetores das classes básicas de dado. Portanto, para indexar as posições nesse vetores, podemos usar a indexação básica de vetores:

Indexando posição

```
caixeta$dap[1]
caixeta$dap[c(1, 3, 5, 7, 9)]
caixeta$dap[seq(1, 19, by= 2)]
```

Uma outra forma de indexar é pelas posições da linha e coluna. No caso do `data.frame` isso é feito pela indicação das duas posições separadas por vírgula entre colchetes `[ , ]`. Sempre o valor antes da vírgula se refere a primeira dimensão do `data.frame` que são as linhas (observações) e o valor depois refere-se a coluna (variável).

Indexando duas dimensões

```
caixeta[1, 9]
caixeta[c(1, 3, 5, 7, 9), 9]
caixeta[seq(1, 19, by= 2), 9]
```

Por fim, é possível também combinar nomes das variáveis com as posições das observações:

Indexando por nome

```
caixeta[1, "dap"]
caixeta[c(1, 3, 5, 7, 9), "dap"]
caixeta[seq(1, 19, by= 2), "dap"]
```

Inclusive, combinar nomes de variáveis:

Indexando duas variáveis

```
caixeta[1, c("dap", "cap")]
caixeta[c(1, 3, 5, 7, 9), c("dap", "cap")]
caixeta[seq(1, 19, by= 2), c("dap", "cap")]
```

Indexação com Lógica

Para apresentar essa última forma de indexação, vamos construir um `data.frame` sem fazer a leitura, usando a função `data.frame` que recebe os vetores de dados de mesmo tamanho.

Os trapalhões

```
trapalhoes <- data.frame(nomes = c("Didi","Dedé","Mussum","Zacarias"),
ano.nasc = c(1936, 1936, 1941, 1934), vive = c("V", "V", "F", "F"))
```

Vamos agora calcular a idade dos meninos:

Operando variáveis

```
ano.atual <- as.numeric(format(Sys.Date(), "%Y"))
ano.atual - trapalhoes$ano.nasc
trapalhoes$idade <- ano.atual - trapalhoes$ano.nasc
```

Tá bom! Vou explicar a primeira linha de comando, ao mesmo tempo, indicar o caminho para entender comandos mais complicados em geral. Primeiro temos a característica da sintaxe aninhada, padrão do R. Para entender o que está sendo feito, pegue a função mais interna e veja o que ela está retornando, em seguida veja o que função externa a ela está fazendo com esse resultado, e assim por diante!

Dilacerando o código

```
Sys.Date()
format(Sys.Date(), "%Y")
as.numeric(format(Sys.Date(), "%Y"))
```

Por que?

Pois bem, estamos primeiro pegando a data atual armazenada no computador, depois pegando apenas o ano e em seguida transformando o ano em número. Ué? Não seria mais fácil digitar o ano diretamente? Sim, mas depois teria que atualizar o código todos os anos antes do curso!

Disponível em: <educrealmirian.blogspot.com>. Acesso em: 27 nov. 2013

Para extrair informações específicas do `data.frame` usamos a indexação com um vetor lógico. Por exemplo, podemos nos perguntar: Quais os nomes dos trapalhões que nasceram antes de 1940?

Operação lógica

```
antes40 <- trapalhoes$ano.nasc < 1940
```

```
antes40
```

O resultado desse teste lógico é um vetor de TRUE e FALSE:

```
[1]  TRUE  TRUE FALSE  TRUE
```

Ele nos diz que a condição é verdadeira para todas as posições menos para a terceira. Como o vetor lógico tem o mesmo comprimento do número de linhas, podemos então indexar o nosso `data.frame` com esse vetor, da seguinte forma:

Indexação com lógica

```
trapalhoes[antes40, ]
```

ou diretamente, sem o objeto intermediário:

Indexação com operação lógica

```
trapalhoes[trapalhoes$ano.nasc < 1940, ]
```

Note que fizemos a indexação pelas linhas, deixando a indexação da coluna vazia, ou seja, solicitando retornar todas as colunas para as linhas em que o vetor `antes40` é TRUE. Como `data.frame` tem duas dimensões, a vírgula é mandatória para a indexação.

Nossa pergunta foi mais específica, perguntamos o nome e não pedimos as outras informações:

Indexando ambas dimensões

```
trapalhoes[antes40, "nomes" ]
```

Nesse caso, a indexação da coluna pode ser feita pelo nome da variável ou pela posição e também pode ser combinada.

E se quisermos o inverso, o nome e a idade dos que nasceram depois de 1940? Precisamos só inverter o vetor de TRUE e FALSE usando o operador `!`:

Invertendo a lógica

```
trapalhoes[!antes40, c("nomes","idade") ]
```

Essas são as principais instrumentações de indexação no R. Aplicadas em um `data.frame` com quatro observações não parece muito vantajoso. Vamos nos perguntar algo então para os dados das `caixeta` que tem mais de mil observações:

Quais árvores, na amostra da Juréia, tem mais de 100 mm de dap?

Combinando lógica

```
jureiaVF <- caixeta$local == "jureia"
bigtree <- caixeta$dap > 100
caixeta[jureiaVF & bigtree, ]
```

Aqui utilizamos o operador lógico & para perguntar para o R quais posições de `jureiaVF` e `bigtree` são verdadeiras em ambos os vetores.

E para saber quantas árvores tem na amostra da Juréia, quantas dessas são grandes e em que proporção? Lembra que o TRUE é operado algebricamente como sendo o valor 1?

Opera lógica

```
sum(jureiaVF)
sum(bigtree & jureiaVF)
sum(bigtree & jureiaVF)/sum(jureiaVF)
```

Salvando Data Frame

Após manipular os dados no R podemos salvar uma nova versão em um arquivo texto. Para salvar a nossa nova versão dos dados de caixetais, que incorpora o dap, usamos a função `write.table`. Os

parâmetros são parecidos com a função de leitura, só que precisamos indicar qual o objeto a ser salvo e nome do arquivo no qual ele será gravado. Caso queira salvar em um local diferente do diretório de trabalho deve também fornecer o caminho das pastas do computador. Abaixo estamos salvando os dados no arquivo `caixeta.txt`, com os campos separados por tabulação (`\t`) e indicando que não queremos salvar nomes de linhas[8] (`row.names= FALSE`)

Salvando arquivos de dados

```
write.table(caixeta, file = "dados/caixeta.txt", sep = "\t", dec = ",",
row.names = FALSE)
```

Caso receba a mensagem de erro:

```
Error in file(file, ifelse(append, "a", "w")) :
  cannot open the connection
In addition: Warning message:
In file(file, ifelse(append, "a", "w")) :
  cannot open file 'dados/caixeta.txt': No such file or directory
```

Significa que o caminho para gravar o arquivo tem algum problema, no caso não há a pasta `dados` subordinada ao seu diretório de trabalho. Para criar uma pasta pelo R, use o comando abaixo:

Criando pastas

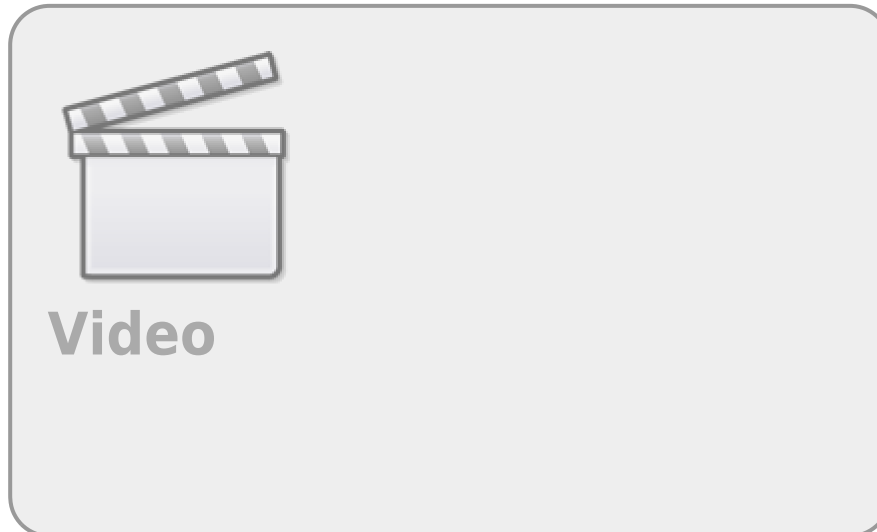
```
dir.create("dados")
dir.exists("dados")
dir()
```

Agora podemos rodar novamente a linha de comando do `write.table` e gravar o nosso arquivo `caixeta.txt` no diretório `dados`, subordinado ao diretório de trabalho que a sessão do R está associado.

Formato de Arquivo de Dados

Arquivos de dados no formato de texto são uma forma segura de salvar conjunto de dados pequenos ou medianos (~100 mil registros). Os formatos mais usados são os com campo separados por espaço (padrão do `read.table`), separados por vírgula ou ponto e vírgula, normalmente com extensão `.csv`, ou tabulação, normalmente com extensão `.txt`. A extensão é apenas uma indicação de formato e como é possível salvar o arquivo com qualquer extensão, precisamos saber qual estrutura foi utilizada para salvar os dados. O excel em sistemas operacionais em português salva arquivos `csv` separados por `;`, com símbolo de decimal `,`, o que causa bastante confusão. Nossa sugestão é que configure seu computador para decimal com `.` e estabeleça o seu padrão de separação de campo, deixando indicado em um arquivo acessório de metadados. Antes de ler um arquivo de dados de texto que desconheça a formatação, abra em um arquivo de edição de texto simples, como o bloco de notas, para verificar os símbolos de separação de campo e decimal.

Matrix e Array

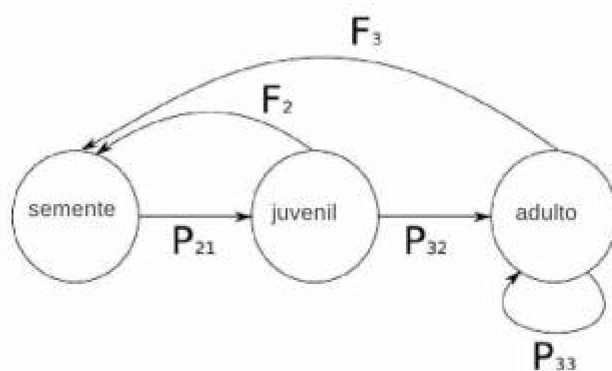


Matrizes

Outro tipo de objeto no R com estrutura tabular é a `matrix`, que está associado à álgebra linear. A principal diferença com `data.frame` é que `matrix` só aceita uma classe de dados, assim como `vector`. A `matrix` é usada no sentido de matrizes da álgebra linear e operações matriciais.

Vamos aprender o objeto `matrix` associada ao estudo de dinâmica populacional.

Operações Matriciais



As matrizes de transição são uma maneira conveniente de modelar o crescimento de uma população dividida em faixas etárias, ou estágios de desenvolvimento [9]. Uma forma de representar essas populações estruturadas[10] é com diagramas mostrando as transições e permanências entre estados.

Uma forma compacta de representar matematicamente essas transições e permanência é utilizando a forma de matriz, que nesse caso se chama ***matriz de transição***, como na figura abaixo:

$$\mathbf{A} = \begin{pmatrix} 0 & F_2 & F_3 \\ P_{21} & 0 & 0 \\ 0 & P_{32} & P_{33} \end{pmatrix}$$

Uma população de *Escobaria robbinsiorum* (Cactaceae) no deserto do Arizona, com três estágios de desenvolvimento, tinha a seguinte a matriz de transição:

Escobaria robbinsiorum

0,43	0	0,56
0,33	0,61	0
0	0,30	0,96

Os elementos da matriz, com exceção da fecundidade (F2 e F3 no diagrama), são as probabilidades de transição, num intervalo de tempo, do estágio correspondente ao número da coluna para o estágio correspondente ao número da linha. Por exemplo, a chance de um indivíduo passar do estágio 1 (plântula) para o 2 (jovem) é 0,33, e de permanecer em 1 é de 0,43. A fecundidade, representada nessa matriz como o elemento da primeira linha e terceira coluna (0,56) é o único que não é uma probabilidade e representa a contribuição, em número de jovens do estágio 1, que um adulto produz em média a cada ciclo.

Vamos criar um objeto da classe matriz com esses valores. Isso nos permitirá realizar operações matriciais para prever o tamanho da população.

Criando matriz

```
matCory <- matrix( c(0.43, 0.33, 0, 0, 0.61, 0.3, 0.52, 0, 0.96), nrow
= 3, ncol = 3)
matCory
```

A indexação da `matrix` é similar ao `data.frame` quando usamos o `[`. A indexação utilizando `$` não se aplica nesse caso. Por exemplo, a fecundidade foi digitada errada no código acima e precisamos ajustá-la:

Indexando matriz

```
matCory[1, 3]
```

```
matCory[1, 3] <- 0.56
matCory
```

Aqui estamos usando uma regra básica dos objetos de dados no R: se consegue extrair um elemento de um objeto é possível atribuir algo a essa posição.

Agora crie um vetor com as abundâncias iniciais de indivíduos em cada classe. Vamos começar com uma população de 50 plântulas , 25 jovens e 10 adultos.

Vetor de tamanho populacional

```
popN0 <- c(50, 25, 10)
popN0
```

Para calcular o número de indivíduos em cada estágio após um intervalo de tempo, basta multiplicar a matriz de transição pelas abundâncias dos indivíduos em cada estágio. Usamos o operador de multiplicação matricial `%*%` para isso. Qual será o número de plantas em cada estágio após três intervalos?

Multiplicação matricial

```
popN1 <- matCory %*% popN0
popN1
popN2 <- matCory %*% popN1
popN2
popN3 <- matCory %*% popN2
popN3
```

Vamos agora armazenar a trajetória do tamanho dos estágios na população em uma matrix:

Matriz de tamanho populacional

```
pop <- matrix(c(popN0, popN1, popN2, popN3), ncol = 3, byrow = TRUE)
colnames(pop) <- c("plântula", "jovem", "adulto")
pop
```

O que acontece aqui?

```
pop <- rbind(pop, t(matCory %*% pop[nrow(pop),]))
```

No código acima, fazemos a multiplicação matricial `%*%` de duas matrizes: `matCory` e a última linha do objeto `pop`. Em seguida transpomos[11] a matriz resultante com a função `t` para depois combinar o resultado transposto com o objeto `pop`. Para isso, usamos a função `rbind` que combina as linhas de dois objetos tabulares. Por último, sobrescrevemos o objeto `pop` com essa nova combinação.

Veja como funciona o código acima executando cada passo separadamente, do mais interno para o

mais externo:

Entendendo o código

```
nrow(pop)
pop[nrow(pop),]
matCory %*% pop[nrow(pop),]
t(matCory %*% pop[nrow(pop),])
rbind(pop, t(matCory %*% pop[nrow(pop),]))
```

Podemos fazer mais algumas multiplicações, guardando o resultado diretamente no objeto pop, usando somente a mesma linha de comando novamente.

Repetição

```
pop <- rbind(pop, t(matCory %*% pop[nrow(pop),]))
pop <- rbind(pop, t(matCory %*% pop[nrow(pop),]))
pop <- rbind(pop, t(matCory %*% pop[nrow(pop),]))
pop <- rbind(pop, t(matCory %*% pop[nrow(pop),]))
pop <- rbind(pop, t(matCory %*% pop[nrow(pop),]))
## ...
```

Vamos agora nomear essas linhas usando a função `paste` que combina caracteres.

Nomeando linhas

```
rownames(pop) <- paste("t", seq(from = 0, to = (nrow(pop) - 1)), sep =
"")
```

A função paste junta caracteres de dois vetores com os princípios da equivalência de posição e ciclagem, próprios das operações com vetores. Caso não tenha entendido o código acima, faça a separação de cada passo e veja a documentação das funções. Esse é um procedimento básico para entender códigos quando estamos iniciando no aprendizado de uma linguagem computacional.

Por fim, vamos olhar esses dados em um gráfico no objeto pop . O código abaixo vai ser abordado na aula de gráficos, por enquanto apenas copie e execute o código no R:

Grafico de matriz

```
matplot(x = 0:(nrow(pop)-1), pop, type = "l", xlab = "Tempo", ylab =
"Número de indivíduos", lty = 2, lwd = 2, col = c("red", "blue",
"green"), cex.lab = 1.2, cex.axis = 1.2, las = 1)
legend("topleft", legend = colnames(pop), bty = "n", lty = 1, col =
c("red", "blue", "green"))
```

Modelos Matriciais de Dinâmica Populacional

Com um pouco mais de álgebra linear você pode obter muito mais informações sobre características da população biológica, apenas a partir das informações intrínsecas da matriz de transição (autovalores e autovetores). Por exemplo, a taxa de crescimento (λ) da população é o primeiro autovalor da matriz de transição, enquanto que o valor reprodutivo e a distribuição da proporção dos estágios no equilíbrio estão relacionados aos autovetores. Esses valores podem ser obtidos com a função `eigen`[12]. Para saber como calcular esses valores veja o roteiro da disciplina de ecologia de populações na qual esse tópico foi baseado.

Autovalores da matriz

```
eigen(matCory)
```

Matrix de comunidade

Um formato de dados clássico em ecologia de comunidades é o de espécies por localidade, como a ocorrência ou a contagem de indivíduos. Já usamos a função `table` para contagem de uma variável tipo `fator`. Vamos construir essa nova estrutura de dados a partir do objeto `caixeta` e fazer a coerção para a classe matrix.

Matriz caixeta

```
str(caixeta)
caixTable <- table(caixeta$especie, caixeta$local)
str(caixTable)
class(caixTable)
caixTable[,"chauas"]
caixMatrix <- as.matrix(caixTable)
str(caixMatrix)
identical(caixMatrix, caixTable)
class(caixMatrix)
```

Apesar de parecer que a coerção do objeto `table` para `matrix` não tenha funcionado, a coerção funcionou como deveria. Os objetos da classe `table` são matrizes especiais de tabelas de contingência[13] usadas para os testes de dependência entre variáveis categóricas. Para o R é um objeto subordinado à classe `matrix`.

Vamos agora fazer a manipulação desses dados para saber quantos indivíduos e espécies temos na localidade `jureia`. Primeiro precisamos manipular o objeto `caixMatrix` para que ela tenha apenas a informação de ocorrência

Operando caixeta

```
caixMatrix[caixTable > 0 ] <- 1
head(caixTable)
head(caixMatrix)
```

Agora podemos contar a coluna `jureia` em cada uma dos objetos:

Operando jureia

```
sum(caixTable[, "jureia"])
sum(caixMatrix[, "jureia"])
```

Poderíamos usar a função `apply` para fazer a contagem de todas as colunas de uma vez. A função `apply` aplica uma função para alguma dimensão dos dados, no caso, coluna ou linhas, e cicla a função para todas as posições dessa dimensão.

Apply em matriz

```
apply(caixTable, MARGIN = 2, FUN = sum)
apply(caixMatrix, 2, sum)
```

Combinando Dados Tabulares

É comum termos informações que estão em formato de dados tabulares e precisam ser agrupadas com outro conjunto de dados. Vamos olhar algumas das ferramentas para fazer isso.

Em construção

Combinando colunas ou linhas

As funções `rbind` e `cbind` são utilizadas para concatenar dados tabulares pelas linhas ou colunas. Entretanto, só funcionam se a estrutura é a mesma, ou seja as variáveis são as mesmas e na mesma posição para acrescentar novas observações, ou as observações estão nas mesmas linhas para acrescentar novas variáveis.

Combinando matrizes

```
trapa <-
read.table("http://ecor.ib.usp.br/lib/exe/fetch.php?media=dados:trapa.csv", header = TRUE, sep = ",")
str(trapa)
```

```
trapaNome <-
read.table("http://ecor.ib.usp.br/lib/exe/fetch.php?media=dados:trapa_nome.txt", header = TRUE, sep = ",")
str(trapaNome)

trapaNome
trapa

rbind(trapa, trapa)
trapabind <- cbind(trapa, trapaNome)
```

Outras formas de combinar

Quando colunas ou linhas não podem ser combinadas diretamente, porque as posições não são compatíveis, podemos usar as funções `merge` e `match`. O `merge` combina dois objetos por uma coluna que é a referência comum, combinando as variáveis de ambos `dataframe`. O `match` funciona também a partir de uma variável comum a ambos objetos e produz um índice de posição que ordena um dos objetos na mesma posição que o outro. O `match` é um pouco mais difícil de entender, mas é uma ferramenta poderosa para manipular dados tabulares. Abaixo apresentamos ambas funções manipulando dados muito simples para demonstrar como funcionam.

Merge

```
## merge

merge(trapa, trapaNome, by = "codinome")
trapa$codinome
trapaNome$codinome

merge(trapa, trapaNome, by = "codinome", all = TRUE)
trapa$codinome[2] <- trapaNome$codinome[3]
trapa$codinome
merge(trapa, trapaNome, by = "codinome")


## match

(matchtrap <- match(trapa$codinome, trapaNome$codinome))
trapaNome[matchtrap, ]
trapa$nome <- trapaNome[matchtrap, "nome"]
trapa
```

Array

O objeto `matrix` por sua vez é apenas um objeto da classe `array` com apenas duas dimensões,

assim como o `vector` é um array de uma dimensão. Veja a classe do objeto pop que criamos na sessão de multiplicação de matrizes:

Array

```
class(pop)
```

A classe é `matrix` mas a classe parental é `array`. Essa possibilidade de um `array` ter muitas dimensões permite efetuarmos operações e análises em múltiplas dimensões. As características e operações que fizemos em `matrix` se aplicam também para `array`.

Vamos avaliar um objeto dessa classe chamado `Titanic`. Primeiro vamos entender onde está esse objeto.

A função `search` mostra o caminho de busca nos compartimentos de memória da sessão do R. Nele há um pacote chamado `datasets` que é carregado por padrão ao abrirmos uma sessão do R.

Datasets

```
search()
ls("package:datasets")
ls("package:datasets", pattern = "Tit")
```

Isso significa que esse objeto está disponível para uso sem a necessidade de carregá-lo. Vamos investigar alguns atributos dele:

Titanic

```
is.array(Titanic)
dim(Titanic)
dimnames(Titanic)
```

Um tanto mais complicado de visualizar os dados do que uma planilha. Imagine os quatro níveis de `Class` como sendo as linhas de um planilha, `sex` como as colunas, age como sendo a repetição dessa estrutura para `Child` e `Adult`, por fim essa estrutura replicada para `Survived` igual a `No` e `Yes`. Como as quatro dimensões tem tamanhos pequenos é possível visualizar todos dados:

array

```
str(Titanic)
Titanic
```

Essa estrutura permite operar qualquer dimensão, ou mesmo algumas dimensões ao mesmo tempo. Vamos usar o `apply` para obter algumas informações interessantes sobre as vítimas do naufrágio.

Será que crianças e adultos tiveram a mesma proporção de vítimas?

Array apply

```
apply(Titanic, c("Age", "Survived"), sum)
```

Será que a proporção de vítimas entre homens e mulheres foi similar?

Array apply II

```
apply(Titanic, c("Sex", "Survived"), sum)
```

E entre os passageiros de diferentes classes?

Array apply III

```
apply(Titanic, c("Class", "Survived"), sum)
```

Mistério do Titanic: mulheres e crianças primeiro

Os grupos com maior sobrevivência no desastre do Titanic foram as mulheres e crianças da primeira classe. Analisando dados de múltiplos acidentes marinhos o artigo ''Gender, social norms, and survival in maritime disasters'', Elinder & Erixson (2012) verifica que a distribuição de sobrevivência do Titanic é uma exceção. As evidências nos dados desse artigo mostram que a maior sobrevivência está entre os homens da tripulação e a menor entre as crianças.

Listas

Listas são os objetos mais versáteis para armazenar informação no R. Apesar de ter uma única

dimensão, suas posições comportam qualquer outra classe de objeto que vimos até então. Normalmente os objetos mais complexos no R, como por exemplo resultados de modelos estatísticos, são casos especiais de listas padronizadas. Vamos criar a nossa primeira lista com alguns objetos desse tutorial:

Criando listas

```
minhaLista <- list(vectorNum = num, dfTrapa = trapalhoes, matPop = pop,
arrayTit = Titanic)
str(minhaLista)
```

A indexação da lista é um misto das classes de objetos que vimos anteriormente. Na sua primeira dimensão aceita tanto o nome com $ como um `data.frame` quanto um novo indexador que é o `[[` colchete duplo.

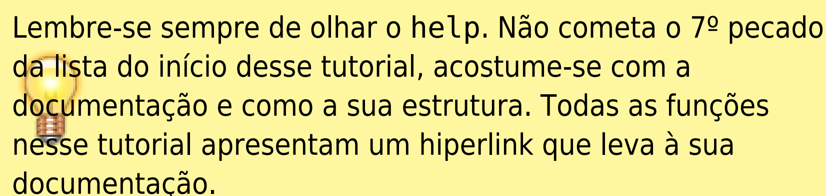
Indexando listas

```
minhaLista <- list(vectorNum = num, dfTrapa = trapalhoes, matPop = pop,
arrayTit = Titanic)
str(minhaLista)
minhaLista$dfTrapa
minhaLista[[3]]
minhaLista[["vectorNum"]]
```

Os elementos de cada posição podem ser acessados usando as indexações correspondentes a cada classe!

Niveis de indexação

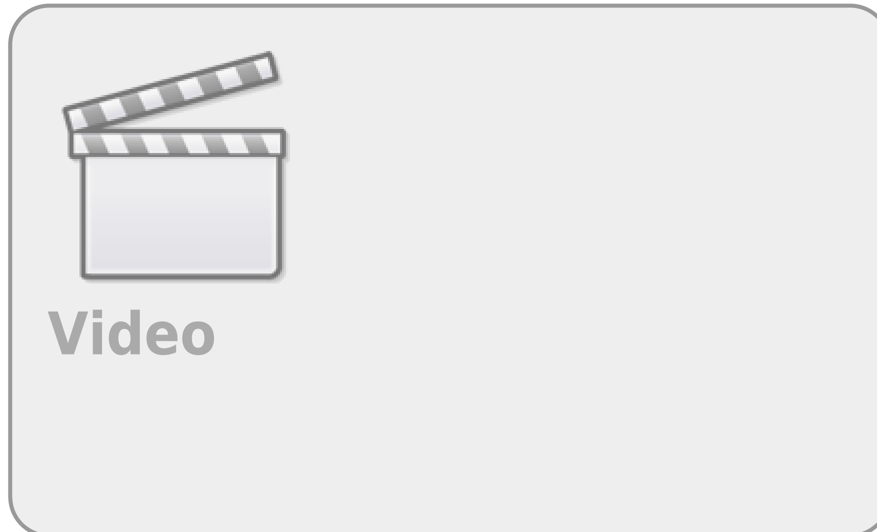
```
minhaLista$dfTrapa[, "nomes"]
minhaLista[[3]][1,]
minhaLista[["vectorNum"]] [7]
```



Lembre-se sempre de olhar o `help`. Não cometa o 7º pecado da lista do início desse tutorial, acostume-se com a documentação e como a sua estrutura. Todas as funções nesse tutorial apresentam um hiperlink que leva à sua documentação.

Um Exemplo

No vídeo abaixo apresento um exemplo de manipulação de dados de um estudo sobre dinâmica de população de uma espécie de árvore (*Guapira opposita*) da floresta atlântica, utilizando matrizes e arrays.



1)
existem muitos outros tipos de formatos de armazenamento de dados que incorporam dados mais complexos e georreferenciados

2)
`read.table` é muito flexível, veja a documentação!

3)
incluíndo a extensão e o caminho, caso não esteja no diretório de trabalho

4)
utiliza a primeira linha dos dados para o nome das colunas

5)
qual o símbolo separa os dados em uma linha. Ex: "\t" é tabulação

6)
O padrão até a versão 4.0 do R era transformar caracteres em fator na leitura. A partir dessa versão o padrão mudou e a versão mais recente da função não faz essa transformação automática.

7)
veja artigo sobre essa mudança em developer blog

8)
Por padrão a função `write.table` salvará o valor de indexação da linha como o nome da observação

9)
para saber mais veja o roteiro de população estruturada em nosso projeto **EcoVirtual**

10)
apresentam estágios diferentes

11)
ou seja, trocamos as linhas pelas colunas

12)
consulte a ajuda para interpretar o resultados dessa função

13)
tabelas de contagem

From:
http://ecor.ib.usp.br./ - **ecoR**

Permanent link:
http://ecor.ib.usp.br./doku.php?id=02_tutoriais:tutorial3:start&rev=1692045544

Last update: **2023/08/14 17:39**