Roteiro 1

Melina Leite

Departamento de Ecologia IB-USP

Contents

O R e sua filosofia de trabalho

"Uma das coisas mais importantes que você pode fazer é dedicar um tempo para aprender uma linguagem de programação de verdade. Aprender a programar é como aprender outro idioma: exige tempo e treinamento, e não há resultados práticos imediatos. Mas se você supera essa primeira subida íngreme da curva de aprendizado, os ganhos como cientista são enormes. Programar não vai apenas livrar você da camisa de

força dos pacotes estatísticos, mas também irá aguçar suas habilidades analíticas e ampliar os horizontes de modelagem ecológica e estatística."
Gotelli & Ellison, 2004. A Primer of Ecological Statistics. Sunderland, Sinauer.

O R não é um software do tipo *aplicativo*, é um **ambiente de programação** com um conjunto integrado de ferramentas de software para manipulação de dados, cálculo e apresentação gráfica. A **linguagem R** foi elaborada inicialmente com o objetivo específico de análises de dados, na qual muitas técnicas estatísticas, clássicas e modernas, podem ser implementadas, entre outras coisas. Algumas dessas técnicas estão implementadas no ambiente báscio do R (**R base**), mas muitas estão implementadas em pacotes adicionas (**packages**).

O ambiente R foi desenvolvido baseado na linguagem S, no final da década de 90 início dos anos 2000. A sua estrutura de código aberto (que vem da linguagem S) e de software público e gratuito atraiu um grande número de desenvolvedores, sendo que há hoje inúmeros pacotes para o R.

Instalando o R

Vá para a página de download do R para o CRAN mais próximo (BRAZIL SP - FMVZ-USP) e baixe o pacote de instalação para o seu sistema operacional. O R já vem com uma interface gráfica para utilização. O tipo de interface depende do sistema operacional, se for o Windows será o RGUI. Esta interface distribuída ainda carece de muitos recursos que estão disponíveis em interfaces desenvolvidas para facilitar o trabalho e ser mais amigável com aqueles que não entendem muito de linha de comando e programação. Para isso usaremo o Rstudio, veja como instalar abaixo.

Instalando o Rstudio

O Rstudio é um IDE (Integrated Development Environment) de distribuição livre para o R. Ou seja, é um interface gráfica que permite que utilizemos o R de maneira mais amigável, simples e eficiente. É muito útil para aqueles que não estão familiarizados com ambientes de programação através de linha de comando.

Instalando pacotes no R

Pacotes nada mais são do que conjuntos de funcionalidades (funções e dados) distribuídos em conjunto para realizar tarefas específicas. Como já foi dito, o R vem com os pacotes base (R base) que possui grande parte das funções mais básicas em leitura, manipulação de dados e estatítica. Porém, uma imensa quantidade de funções muito úteis estão em pacotes adicionais (add-on packages), que podem ser baixados e instalados no R rapidamente através dos comandos dentro do console do R.

Como exemplo, o pacotes vegan carrega na sua área de trabalho (deixa disponível para uso) um conjunto de ferramentas para análises de dados de ecologia de comunidade. Para usar os pacotes disponíveis é necessário entender as diferenças entre baixar (download) o pacote do repositório e carregar em sua área de trabalho. Para baixar algum pacote disponível no repositório CRAN do R é necessário utilizar o comando `install.packages()` com o nome do pacote entre "" dentro do parenteses. Se não houver nenhuma mensagem de erro, significa que o download do pacote foi realizado com sucesso.

Se você quiser usar uma função que está implementada dentro do pacote instalado, você precisa carregá-lo para fazer parte do seu workspace. Para cada projeto, precisamos carregar aqueles pacotes que vamos necessitar (normalmente colocamos a função `library` nas primeiras linhas de comando do nosso código):

```
install.packages("vegan") #instalando o pacote vegan
library(vegan) # chamando o pacote para que você utilize suas funções
```

OBS: No Rstudio existe uma aba no painel direito inferior chamada `Packages` onde você pode ver os pacotes que já vem com o R base e os que você isntalou. Clique no botão de marcação do pacote se você quer ativá-lo

para usar suas funções (é o mesmo que a função `library()`). Também, para instalar um pacote você pode clicar no botão `Install` que será o mesmo que dar o comando `install.packages`.

O código é tudo!

Um dos primeiros hábitos que você deve adquirir para trabalhar com o R é não digitar os comandos diretamente na linha de comando (Console), e sim em um arquivo texto, que chamamos de *script* ou código. Este novo *script*, com extensão .R ou .r, conterá todos os comandos que você vai usar para analisar seus dados e pode ser lido facilmente por qualquer editor de texto (bloco de notas do windows por exemplo). Os *scripts* são sequências de comandos que podem ser armazenados e usados de novo quantas vezes quisermos. No cotidiano do uso do R, é bem comum que precisemos fazer o mesmo procedimento diversas vezes com diferentes conjuntos de dados. Isto pode se tornar bem trabalhoso, o que torna muito prático armazenar linhas prontas para que possam ser usadas novamente.

Uma coisa muito importante é a possibilidade de inserir comentários nos scripts para você não se perder em suas análises. Para inserir cabeçalho, dicas, comentários ou até mesmo "desligar" um comando você colocará o símbolo de # antes de qualquer linha. É recomendável que os comentários sejam usados durante o processo de aprendizado, na forma de lembretes sobre o uso das funções. Eles também são bastante úteis durante a interpretação de resultados de análises, pois permitem que o usuário anote as suas primeiras conclusões e interpretações diretamente com os resultados. Nos exemplos deste tutorial existem vários comentários como exemplo.

Na interface do Rstudio você pode criar um novo editor de texto indo em FILE > NEW FILE > R SCRIPT, ou clicando no íncone (+) no canto esquerdo superior. Assim, você terá seu novo script aberto logo em cima do *console* do R. Para enviar comandos do script para o console, aperte CTRL+R ou CTRL+Enter (para usuários do Mac use Comand+Enter) em cima da linha do comando.

Primeiros passos

Abra o Rstudio [1] em seu computador. Após algumas informações sobre a versão, a licença de uso, os contribuidores e os mecanismos de ajuda você verá um prompt de comando em forma de sinal de maior (>) no *R console*. Digite no prompt cada linha do código abaixo e tecle enter para executar. Cuidado com erros de digitação, que possivelmente gerarão erros de sintaxe, o que resultará em uma mensagem de erro.

Execute os comandos no R:

```
contributors()
citation()
demo(colors) #bom para escolher cores de elementos gráficos
```

Preso no R: Um erro comum é teclar enter antes de finalizar corretamente o comando (falta de ',' , ')'...). Nesse casos o prompt ficará esperando a finalização do comando (com um sinal de '+') e não aparecerá mais o sinal de '>' na linha. Nesses casos use a tecla 'Esc' para abortar o comando e voltar ao >.

O comando mais usado no R: help!

Explore o resultado dos comandos abaixo. Veja o formato geral da página de ajuda. Um dos pedaços mais importantes das páginas de ajuda das funções fica no seu final, os exemplos de uso.

[1] e consequentemente o R irá abrir também!

```
help(help)
help( "*" )         # auxílio sobre o símbolo "*"
help( sin )         # auxílio sobre a função "sin" = seno
?sin                # variante da função "help"
```

Se você não sabe o nome da função que você quer usar, tente usar *??função*. O R abrirá uma página com funções que fazem referência à palavra buscada. Também vale usar a função _help.search()___ para buscar com mais de uma palavra. Experimente:

```
??variance
help.search("linear regression")
```

Se mesmo lendo o help da função você não entendeu bem como ela funciona, execute um exemplo da função. Os exemplos podem ser vistos no final da página do help, daí você copia e cola o exemplo no console, ou então você executa a função de exemplo:

```
example(mean)
```

OBS: no Rstudio, existe uma aba no layout do canto direito inferior para você fazer as buscas no Help diretamente, sem precisar dar o comando no Console.

Funções no R

As 'palavras' que você digitou assima são **funções**. Todas as funções no R seguem a mesma lógica de uso: basta digitar o nome da função seguido dos argumentos necessários, que devem estar entre parênteses. Ou seja, você dá os argumentos que a função pede, e ela te retornará o resultado pretendio. Não sabe quais são os argumentos de uma função? Vá para o **Help** da função ou digite 'args(funcao)!

A sintaxe básica é: **função**(argumento1=*valor1*,argumento2=*valor2*,...)

Abaixo, algumas funções matemáticas:

```
sqrt(4)
log(x=100,base=10)
log(100,10)
sin(pi) # pi é a constante 3,141593; um objeto que já "vem no R",
cos(pi)
```

OBS: Você pode omitir o nome do argumento na função apenas se os valores estiverem na mesma ordem/posição dos argumentos da função. No exemplo acima fizemos `log(100,10)` omitindo o nome dos argumentos `x=100` e `base=10`, a função funcionou apenas porque os valores 100 e 10 estão na mesma posição dos argumentos. Para evitar se confundir no início, digite sempre o nome dos argumentos!

Objetos no R

No R tudo é considerad um objeto, até mesmo as funções. O R faz parte de uma categoria de linguagem de programação chamada de **"programação orientada a objetos"**. Um objeto será reconhecido como um nome a ele atribuído[2], e guarda "dentro" dele um conjunto de informações (valores, nomes, funções,etc.). Para ver um objeto no R, basta digitar o seu NOME. Para funções, o nome do objeto é a função sem os parênteses. Experimente os objetos abaixo:

[2]por nós mesmos ou por quem criou a função, por exemplo.

```
help
a <- c(1,2,3) #um objeto vetorial, veja o help da função c(), ela é MUITO útil!
a #depois de criado o objeto você o "chama", digitando o nome dele
```

Alguns exemplos de objetos e operações com os objetos:

```
x = 2
x
y = 7
y
z = x * y
z
w = x - y
w

k = c(1.3, -5, 6.7, 4.8)
k
x * k #observe que você não criou nenhum objeto novo aqui
m = c(0, 1, 1, 0, 1, 1)
m
m * k #qua foi a mensagem do R quando você executou? porque será?
```

Algumas regras para nomes de objetos no R:
- há diferenças entre palavras maiúsculas e minúscuas: y não é igual à Y
- nomes de objetos **não** podem começar com números (ex. 1x) ou símbolos (ex. %x)
- nomes de objetos **não** podem ter espaços em branco (use `meu_objeto` ao invés de `meu objeto`)

A área de trabalho

Quando você cria um objeto, este é salvo em um ÁREA DE TRABALHO (**workspace**). Para saber quais objetos você tem na área de trabalho use a função:

```
ls()
```

Para apagar objetos indesejados, utilize a função `rm()`, fornecendo os objetos que você deseja apagar.

```
rm(x,y,z)
ls()
```

OBS: No Rstudio, você pode ver quais são seus objetos na aba superior direita em 'Environement'. Lá, estarão listados todos os objetos criados, assim com suas classes e outras informações. Se você quiser apagar TODOS os objetos da sua área de trabalho, basta clicar em 'clear'.

Onde está sua área de trabalho no seu computador?

Afinal, em qual diretório (pasta) você está trabalhando e salvando seus dados (arquivo oculto .Rdata)?. Descubra com a função:

```
getwd()
```

Para mudar de diretório:

```
setwd("caminho do diretório") #ex: ("/users/meunoem/analises/R")
```

Você deve salvar nesse diretório o *script* que você criou e está trabalhando. Também é recomendável você salvar os arquivos de dados (veja abaixo) no mesmo diretório que estiver seu *script* e o workspace (com o arquivo oculto dos objetos salvos .Rdata).

Lendo dados para dentro do R: formato csv

Para colocar os dados de uma planilha eletrônica (ex: excel), você tem que primeiro salvar os dados em formato CSV ou TXT, aqui vamos falar apenas de CSV, mas para TXT a lógica é muito parecida.
O formato CSV (comma-separated values) consiste em uma única tabela (spreadsheet) da sual planilha eletrônica, onde os valores são gravados linha-a-linha, sendo que numa mesma linha os valores são separados por vírgula.

O formato UNIVERSAL de tabela de dados para análise estatística é o seguinte:

- cada LINHA é uma observação
- cada COLUNA é uma variável ou atributo que foi tomado em cada observação

No formato CSV, cada linha será uma observação e as colunas serão separadas por vírgulas[3]

DICA: Na sua planilha de dados, vai ser muito mais fácil trabalhar com nomes de colunas que não possuem espaço (ex. `abund_1` ou `abund.1` ao invés de `abund 1`). O R ao ler espaços vai coloar um ponto (.) no lugar do espaço. E evite, ou melhor **NUNCA coloque acentos e caracteres especiais** na sua planilha, tanto nos nomes de colunas quanto em fatores e caracteres de dados. Ao converter os dados pro R, sempre sarião caracteres muito esquisitos no lugar do acento e isso vai dificultar muito a sua vida na hora de manipular seus dados.

Lendo um Arquivo CSV

Como exemplo, usaremos a tabela de dados `pratica1.csv`, baixados aqui[4]. Salve o arquivo no diretório que estiver trabalhando com o R: `pratica1.csv`. Esse arquivo pode ser visualizado em qualquer editor de textos, pois o formato CSV é um formato texto.

Importanto a tabela de dados para o R. Lembre-se que é necessário gravar a leitura do arquivo em um objeto:

```
dados <- read.table("pratica1.csv", header=TRUE, sep=',', dec='.')
dados <- read.csv("pratica1.csv", header=TRUE, sep=';', dec='.',row.names = 1)
```

Nestas funções: `header=TRUE` indica que a sua planilha tem cabeçalho (e quase todas tem!); `sep=","` diz que o separador das colunas é vírgula; e `dec="."` significa que o separador de decimal da tabela é ponto.

A função `read.table()` é mais genérica e pode importar arquivos .txt e .csv. Como o arquivo de dados é `csv` você podem também usar a função `read.csv()` para importar os dados, neste exemplo é a mesma coisa!

Outros argumentos importantes das funções `read.table` e `read.csv` são:

- Converte as colunas com caracteres convertidos em classe **character** (`as.is=T`). Por default, o R sempre converte as colunas que tenha caracteres em fatores. (veja mais adiante as classes de objetos no R)

[3] às vezes quando salvamos do excel a separação é ; (ponto-vírgula).
[4] clique com o botão direito no arquivo e mande `save link as`.

- Se sua planilha contém os nomes das linhas, você usa o argumento `row.names=` para especificar qual coluna você quer usar como nome das linhas. Se você não especifica o nome das linhas, o R automaticamente numera as linhas e atribui os números aos nomes.

CUIDADO! nos computadores em português, o separador de decimal geralmente é vírgula. Se for o seu caso, você tem algumas opções:

- Mudar o separador de decimal do seu computador, assim as planilhas de excel mudarão automaticamente.
- ao importar a tabela de dados, especificar o separado de decimal para vírgula. Neste caso, o seu separador de colunas será um ponto-vírgula, e precisará ser mudado também.
- usar a função `read.csv2()`, que já entende que seu separador de decimal é vírgula e que o separado de colunas é ponto-vírgula.

Agora você terá um objeto no R, chamado `dados`, que contém a tabela de dados de `pratica1.csv`.

Verificando a tabela importada

Logo após importar a tabela de dados, você precisa verifica se ela foi importada corretamente. Um erro comum é quando não especificamos o cabeçalho ou não colocamos o separador de colunas corretamente.

Algumas funções úteis:

```
head(dados) # retorna as primeiras linhas da tabela
tail(dados) #retorna as últimas linhas da tabela

dim(dados) #dimensões da tabela: número de linhas e número de colunas

colnames(dados) #nomes das colunas
rownames(dados) #nomes das linhas

str(dados) #estrutura dos dados

summary(dados) #sumário estatísico das variáveis (colunas)
```

OBS: no Rstudio você pode olhar a tabela de dados clicando no nome do objeto no paniel `Environment`(canto direito superior). Ele abrirá uma aba mostrando a tabela de dados.

Classes dos objetos

No R, cada objeto que você cria ou importa pertence à uma classe. A forma mais simples de saber a qual classe aquele objeto pertence é usando a função `class`. Abaixo estão alguns exemplos de **vetores**, ou seja, conjuntos de elementos do mesmo tipo. Há várias classes de vetores:

```
a <- c(1,2,3,4)
class(a)

b <- c("arroz","feijão","farofa")
class(b)
```

```
c <- gl(2,4,8, labels=c("femea","macho")) #generate levels - níveis de um fator
class(c)

d <- c(TRUE,FALSE)
class(d)
```

Matrizes

Matrizes são vetores cujos valores são referenciados por dois índices, o número da linha e o número da coluna. A função `matrix` cria uma matriz com os valores do argumento data. O números de linhas e colunas são definidos pelos argumentos nrow e ncol:

```
minha.matriz <- matrix(data=1:12,nrow=3,ncol=4)
minha.matriz
```

Como o default do argumento `data` é NA, se ele é omitido o resultado é uma matriz vazia:

```
matriz.vazia <- matrix(nrow=3,ncol=4)
matriz.vazia
```

Também por default, os valores são preenchidos por coluna. Para preencher por linha basta o alterar o argumento `byrow` para TRUE:

```
minha.matriz <- matrix(data=1:12,nrow=3,ncol=4,byrow=T)
minha.matriz
```

Se o argumento data tem menos elementos do que a matriz, eles são repetidos até preenchê-la:

```
elementos <- matrix(c("ar","água","terra","fogo","Leeloo"),ncol=4,nrow=4)
# alguma mensagem aqui?
elementos
```

Duas maneiras de modificar os nomes de linhas e colunas em uma matriz:

```
colmanes(matriz.vazia) <- c("c1","c2","c3","c4")
rownames(matriz.vazia) <- c("l1","l2","l3")

matriz.vazia <- matrix(nrow=3,ncol=4, dimnames = list(c("l1","l2","l3"),
                                                      c("c1","c2","c3","c4")))
#veja o argumento dimnames da função!

dimnames(matriz.vazia) #não confundir a função com o argumento de matrix()
```

Arrays

Os arrays são a generalização das matrizes para mais de duas dimensões. Um exemplo é o objeto Titanic presente no pacote base do R, com as seguintes dimensões:

```
Titanic
dim(Titanic)
dimnames(Titanic)
```

Todas as operações aplicáveis a matrizes também o são para arrays:

```
adultos.por.sexo <- apply(Titanic, c(2, 4), sum)
adultos.por.sexo
#Vendo a porcentagem de mortos/sobreviventes por sexo
adultos.por.sexo/apply(adultos.por.sexo,1,sum)
```

Data frame

Com a função `data.frame` reunimos vetores de mesmo comprimento em um só objeto:

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

# O mesmo, em um só comando:
trapalhoes <- data.frame(nomes=c("Didi","Dedé","Mussum","Zacarias"),
                         ano.nasc=c(1936,1936,1941,1934),
                         vive=c("V","V","F","F"))
```

Para mudar nomes de linhas e colunas em um data frame, usamos as mesmas funções `colnames` e `rownames` usadas anteriormente para matrizes.

Classes de colunas em data frame

Em objetos da classe data frame, cada coluna também tem sua classe. A diferença de um data frame para uma matriz é que a matriz aceita apenas uma classe para toda a matriz, ou seja as colunas pertencem apenas a uma classe de valores.

```
class(dados$Ambiente) # utilize $ para selecionar uma coluna do data frame
class(dados$Abund_sp1)

#matrix de números
mat <- matrix(1,ncol = 2,nrow = 2)
mat
class(mat)
mode(mat) # outra forma de ver o que são as "coisas" dentro da matriz
class(mat[,1]) # veja mais adiante como selecionar colunas na matriz [,]
class(mat[,2])

#matriz de caracteres
mat2 <- matrix("a",ncol=2,nrow=2)
mat2
class(mat2)
mode(mat2)
class(mat2[,1])
```

Convertendo a classe de um objeto: coerção

Algumas classes não podem ser convertidas em outras, por exemplo, caracteres em números. Mas você pode converter números em caracteres:

```
as.numeric(b)
as.numeric(c) #o que aconteceu com os fatores?

as.factor(a)
as.character(a)
```

Objetos da classe data:

```
hoje <- "26/09/2015"
depois <- "22/10/2015"
class(hoje)
depois - hoje # para saber a diferença das datas em dias. deu certo?

hoje <- as.Date(hoje, "%d/%m/%Y")
depois <- as.Date(depois, "%d/%m/%Y")
depois - hoje # E agora?
```

NOTA: o argumento format da função `as.Date()` informa o formato em que está o conjunto de caracteres que deve ser transformado em data, no caso dia/mês/ano (%d/%m/%y), todos com dois algarismos. Veja a ajuda da função para outros formatos.

As listas!

Listas são objetos que contém outros objetos de qualquer classe. É como um varal que você pode ir pregando objetos como vetores, data frames, matrizes, modelos...

```
a <- c(3,2,4,5,2,2)
a
area = c(100,235,449, 98, 147, 214, 346, 89)
area
riqueza <- c(56,62,70,33,49,67,71,45)
riqueza



modelo1 <- lm(area~riqueza) # uma regresão linear entre a área e a riqueza
class(modelo1)  #class: linear model
modelo1

minha.lista <- list(um.vetor=a,
                    um.data.frame=dados,
                    um.modelo=modelo1)

minha.lista
```

Para selecionar um elemento da lista: (nome da lista)$(nome do vetor):

```
minha.lista$um.vetor
minha.lista$um.modelo
```

OBS: parece que não, mas data frames são listas com vetores de mesmo tamanho! E é por isso que selecionamos listas e colunas em data frames com $.

Manipulando dados

Selecionando e criando colunas em data frame:

O símbolo $ é o responsável pela seleção das colunas de um data frame, antes do $ vem o nome do objeto, depois o nome da coluna.

```
dados$Ambiente
dados$Area

vetor.area <- dados$Area
```

Para criar uma nova coluna no data frame usando $, antes do $ o nome do objeto e depois o nome da nova coluna. Daí você atribui (<-) valores a esta coluna:

```
dados$log.area <- log(dados$Area)

head(dados) # para verificar a mudança no data frame
dados$log.area
```

Operações em vetores

Nas operações vetoriais o comprimento dos vetores é muito importante, pois o R permite operações entre dois vetores de comprimento diferentes com a seguite regra:

Regra da ciclagem: Operações entre vetores de comprimentos diferentes são realizadas pareando-se seus elementos. Os elementos do vetor mais curto são repetidos sequencialmente até que a operação seja aplicada a todos os elementos do vetor mais longo.

Quando o comprimento do vetor maior não é múltiplo do comprimento do maior, o R retorna o resultado e um aviso:

```
b <- rep(0:1,4)
c <- 1:3

b*c #veja a mensagem de erro, é sempre importante ler estas mensagens

length(b) #para ver o comprimento do vetor
length(c)
```

Mas se o comprimento do vetor maior é um múltiplo do maior, o R retorna apenas o resultado, sem nenhum alerta:

```
a <-1:2
b

a*b # entendeu como o R fez a multiplicação?
```

Portanto muito cuidado com as operações entre vetores de diferentes comprimentos. A regra da ciclagem é um recurso poderoso da linguagem R, mas se você não tiver clareza do que deseja fazer, pode obter resultados indesejados.

Operações com vetores em matrizes e data frames

Operações que envolvem comparações entre um vetor e um outro valor:

```
a <- 1:8

a < 7 #quais valores de a são menores que 7?
a >=7 #quais valores de a são maiores ou iguais a 7?
```

Operadores lógicos no R

Operador	Descrição
"=="	igual
"!="	diferente
">"	maior
"<"	menor
">="	maior ou igual
"<="	menor ou igual
"&"	e (and)
"|"	ou (or)
"!"	não

Uma maneira simples de quantificar frequências

Para ter frequência de dados que satisfaçam uma certa condição basta somar o vetor lógico resultante:

```
notas.dos.alunos <- c(6.0,5.1,6.8,2.8,6.1,9.0,4.3,10.4,6.0,7.9,
                      8.9,6.8,9.8,4.6,11.3,8.0,6.7,4.5)

##Quantos valores iguais ou maiores que cinco?
sum(notas.dos.alunos>=5)

##Qual a proporção deste valores em relação ao total?
sum(notas.dos.alunos>=5)/length(notas.dos.alunos)
```

Mais exemplos de operações lógicas:

```
altura <- c(1.85,1.78,1.92,1.63,1.81,1.55)
sexo <- c(rep("M",3),rep("F",3))

altura
sexo

altura >1.80 #valoes de altura maiores que 1.80
sexo == "M"
machos.altos = altura>1.80 & sexo=="M" #o  que foi feito auqui?
machos.altos
```

Subconjuntos e indexação

Freqüentemente teremos que trabalhar não com um vetor inteiro, mas com um subconjunto dele. Para obter subconjuntos de um vetor temos que realizar operações de indexação, isto é, associar ao vetor um outro vetor de mesmo tamanho com os índices do elementos selecionados.

O operador de indexação é o colchetes '[]'. Veja o help dos colchetes `?"["` para entender melhor.

Para selecionar valores em vetores:

```
altura
altura[1] # valor da posição 1 do vetor altura
altura[5]
altura[2:4] #posições de 2 a 4 de altura

altura[-1] #todos os valores menos aquele da posição 1
```

Usando vetores lógicos, os elementos do vetor lógico correspondentes a 'TRUE' são selecionados, os elementos correspondentes a 'FALSE' são excluídos.

```
b <- 1:8
b
b>5

b[b>5] #selecionando os valores de b que sejam maiores que 5
```

Para indexa fatores não esqueca de colocá-los entre "":

```
x<-gl(2,3,12,labels=c("azul","vermelho"))
x
x[x=="vermelho"] # selecionando vermelhos, veja que o azula ainda é um nível
y <- c(10,11,9,20,19,20,13,8,10,18,23,20)

#selecionando os valores de y que correspondem em posição ao vermleho de x
y[x=="vermelho"]
```

Indexação de matrizes e data frames

O modo de indexação de matrizes é [linhas,colunas]:

```
minha.matriz <- matrix(data=1:12,nrow=3,ncol=4,byrow=T)
minha.matrix

minha.matriz[1,1]
minha.matriz[1:3,3]
```

A mesma notação é válida para data frames:

```
dados
dados[3,2]
dados[1:10,5:6]

dados[dados$local=="borda","log.area"]

dados[dados$local=="borda",5]
```

Para incluir todas as linhas ou colunas, omita o valor (mas mantenha a vírgula!):

```
minha.matriz[,1]
minha.matriz[1,]
dados[,2]
dados[1,]
```

Usando indexação para alterar valores

Combinando as operações de indexação e de atribuição é possível alterar os valores de qualquer parte de um objeto:

```
minha.matriz
minha.matriz[,2] <- 0 #substituiu os valores da segunda coluna por zero

minha.matriz[1,] <- NA
minha.matriz
```

Quando o objeto é um fator, as coisas são um pouquinho diferentes, isso porque a classe fator, não permite que você mude um valor que não esteja dentro dos níveis daquele fator. Por exemplo, a coluna `local` do objeto `dados` é um fator com dois níveis `borda` e `interior`. Se quisermos substituir o nível `borda` por algum outro nome, por exemplo `bordinha`, eu não vou conseguir e aparecerá uma mensagem de erro. Uma forma de fazer isso é primeiro transformar a coluna em classe `character`, aplicar a mudança pretendida, e depois transformar a coluna novamente para classe `factor`

```
dados$local
dados$local == "borda"
dados$local <- as.character(dados$local) #transformando o fator em caracter
#agora sim podemos mudar a palavra borda para qualquer outra coisa
dados[dados$local == "borda",5] <- "bordinha"
dados$local

#e voltamos com a classe fator para a coluna
dados$local <- as.factor(dados$local)
dados$local
```

Lidando com dados faltantes: NA

As vezes nossas planilhas de dados contém dados faltantes, ou seja, que não puderam ser coletados. Estes dados **não** podem ser convetidos em zero, pois isso vai influenciar nas análises estatísticas. Zero é dado! Para identificar os dados faltantes, o R utiliza o termo `NA`. Quando você importa uma planilha de dados para o R com células em branco, o R vai substituir esse 'branco' por NA.

Dados faltantes são geralmente fonte de irritação, pois eles afetam a forma com que diversas funções operam. Por exemplo a função para cálculo da média `mean()`, não funciona se houver `NA` no vetor.

Para saber se os dados contém `NA`, usamos a função:

```
vetor<- c(1,4,NA,5)
is.na(vetor)
```

Como vemos, ela retorna verdadeiro/falso para cada valor no vetor.

Essa função é muito útil quando estamos querendo excluir os dados faltantes para realizar alguma função. Podemos utilizá-la dentro da indexação:

```
vetor2<-vetor[!is.na(vetor)]
```

Excluindo os `NAs` do vetor conseguiremos calcular a média, por exemplo:

```
mean(vetor) # será que funciona?
mean(vetor2) # e agora?

mean(vetor, na.rm = T)
#tente entender o argumento na.rm=T, ele existe em algumas funções e é muito útil
```

Exportando tabela de dados

Depois de manipular seus dados e de fazer as análises pertinetes, você pode exportar facilmente as tabelas de resultados para um arquivo texto .csv ou .txt (mais usados) com a função `write.table`, que vai gravar o objeto de dados especificado na área de trabalho que estiver trabalhando

```
write.table(dados, "dados_out.csv")
```

Material de apoio

Abaixo listo as referências usadas para criação desse roteiro, assim como sites e livros interessantes para buscar mais informações:

- R reference card. Cartão com as funções mais básicas no R. Muito importante ter sempre contigo!
- Página oficial do R.
- Apostila online do Curso Relâmpago de R
- Apostila online Introdução ao R
- Material diponível da disciplina de Introdução ao R do Programa de Pós-Graduação em Ecologia da USP.
- Livro-texto de R, escrito por um ecólogo Crawley 2012 The R book.
- Página oficial do Rstudio

Exercícios!

1. Diretório de Trabalho

1. Crie um diretório para seus exercícios.
2. Chame o R, clicando no ícone da área de trabalho ou na barra de tarefas.
3. Verifique o seu diretório de trabalho.

4. Mude o diretório de trabalho para o diretório que você criou.

5. Verifique o conteúdo da área de trabalho.

6. Carregue o arquivo letras.RData (apagar extensão .pdf).

7. Verifique novamente sua área de trabalho.

8. Saia do R, tomando o cuidado de salvar sua área de trabalho.

9. Repita os passos 2 a 5.

Pergunta: Que problemas você percebeu? Há uma maneira de iniciar o R que evite esses problemas?

2. Use a ajuda para conhecer argumentos das funções

1. Execute o R, usando o diretório de trabalho criado no exercício anterior.

2. Use a função load para carregar o arquivo bichos.RData (apagar extensão .pdf) no workspace.

3. Consulte a ajuda da funções rm e ls para descobrir como apagar apenas os objetos cujos nomes começam com "temp".

3. Classes de Objetos

A distribuição básica do R vem com os objetos letters e LETTERS.

1. Descubra o que cada um contém.

2. Descubra a classe de cada um.

3. Como você os transformaria em objetos da classe fator?

4. Objetos de Data

A função `Sys.Date` retorna a data fornecida pela CPU do computador. Crie um objeto chamado hoje para guardar o resultado deste comando:

```
hoje <- Sys.Date()
```

1. Qual é a classe deste objeto?

2. Qual a diferença em dias entre esta data e o dia em que o Brasil foi tricampeão mundial[5]? Guarde esse valor em um objeto chamado dif.

3. Qual será a data de daqui a 43 dias? Guarde esse valor em um objeto chamado outrodia.

[5] 21/06/70

4. Sequências

Crie as seguintes sequências usando as funções rep e seq:

1. Objeto chamado “letra.a” com caracteres: a a a a a a
2. Objeto chamado “numeros” com valores numéricos inteiros: 1 1 1 2 2 2 3 3 3
3. Objeto chamado “decrescente” com valores numéricos inteiros: 1 1 1 2 2 3
4. Objeto chamado “sequencia” com valores numéricos inteiros: 1 2 3 4 5 4 3 2 1
5. Objeto chamado “impares”: sequência de números ímpares de 1 a 99

5. Área transversal de uma árvore

A área transversal de uma árvore é calculada assumindo que a secção transversal do tronco à altura do peito (1,3m) é perfeitamente circular.

1.Se o diâmetro à altura do peito (DAP) de uma árvore for 13.5cm, qual a área transversal?
2. Se uma árvore possui três fustes com DAPs de: 7cm, 9cm e 12cm, qual a sua área transversal?
3.Se uma árvore possui três fustes com DAPs de: 7cm, 9cm e 12cm, qual o diâmetro (único) que é equivalente à sua área transversal?

6. Construir uma matriz de distâncias

Abaixo estão listadas as distâncias por estradas entre quatro cidades da Europa, em quilômetros:
- Atenas a Madri: 3949
- Atenas a Paris: 3000
- Atenas a Estocolmo: 3927
- Madri a Paris: 1273
- Madri a Estocolomo: 3188
- Paris a Estocolmo: 1827

1. Crie um objeto da classe matrix denominado dist.cid com os valores acima.
2. Para facilitar o uso desse objeto, o nome das linhas e das colunas deve ser o nome das cidades.
3. Você consegue pensar em duas formas diferentes de criar a matriz com nomes nas linhas e colunas?

Para pensar: Compare sua matriz com o objeto `eurodist`, disponível no pacote `datasets`. Quais são as semelhanças e diferenças entre os dois objetos? **DICA:** as funções `lower.tri`,`upper.tri` e `diag` podem lhe ajudar.

7. Criação de um data frame

Imagine um experimento em que hamsters de dois fenótipos (claros e escuros) recebem três tipos diferentes de dieta, e no qual as diferenças dos pesos (g) entre o fim e o início do experimento sejam:

Cor	Dieta A	Dieta B	Dieta C
claro	0.1, 1.1, 3.7	5.7, -1.2, -1.5	3.0, -0.4, 0.6
escuro	1.5, -0.1, 2.0	0.6, -3.0, -0.3	-0.2, 0.3, 1.5

Crie um data frame com esses dados, na qual cada hamster seja uma linha, e as colunas sejam as variáveis cor, dieta e variação do peso.

Importante: o nome do objeto deve ser “hamsters”, e o nome das colunas deve ser “dieta”, “cor” e “pesos”, nessa ordem. Dieta deve ser um fator com os níveis “A”, “B” e “C”, em maiúsculas; “cor” deve ser um fator com níveis “claro” e “escuro” em minúsculas.

DICA: Use as funções de gerar repetições para criar os vetores dos tratamentos.