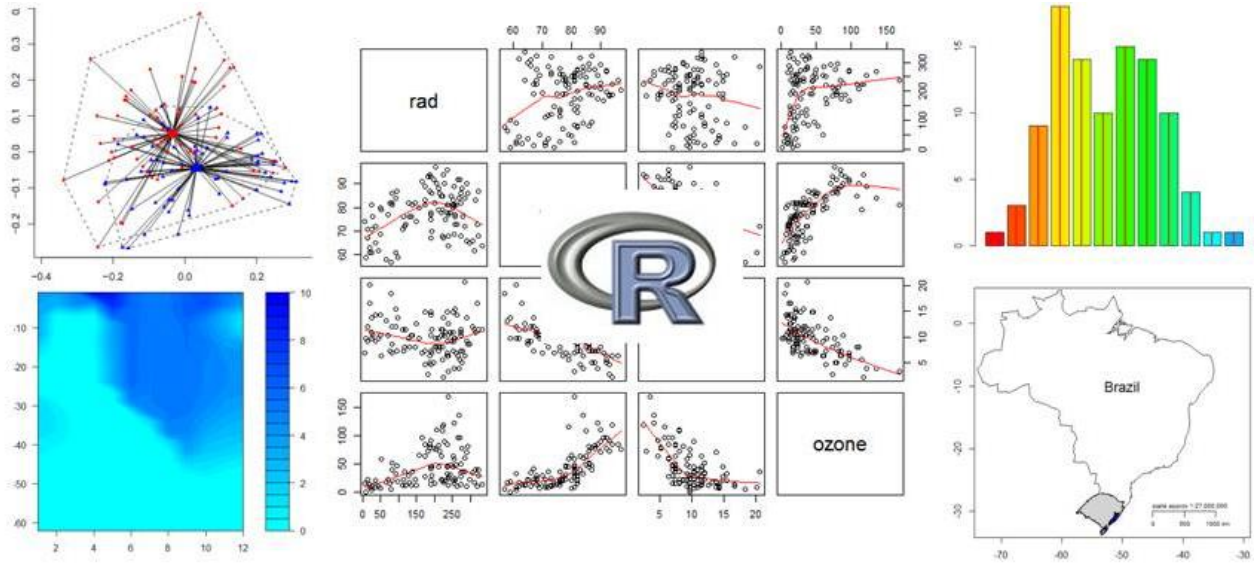
Introdução à Linguagem R e Aplicações em Ecologia



Apostila - versão 5

Elaborada por Fabiana Schneck

Rio Grande, agosto de 2016

Sumário

O QUE ESTE CURSO ABORDA

O foco deste curso é primeiramente fazer uma introdução à linguagem R. Após essa introdução, pretendo apresentar o R como um programa para a construção de gráficos e análise de dados. Neste ponto, não trataremos de detalhes matemáticos, explicações dos algoritmos ou delineamento experimental, focando somente nas aplicações de alguns dos principais testes estatísticos utilizados em Ecologia. Para quem tiver interesse em se aprofundar nos testes que serão apresentados aqui, sugiro o livro *Numerical Ecology* de Pierre e Louis Legendre, além de alguns outros (ver referências bibliográficas no final da apostila).

Para entender a lógica do R não basta somente você ler esta apostila (ou qualquer outro livro sobre R). **É preciso praticar!** Se você de fato tem interesse em utilizar o R, sugiro fortemente que você acompanhe os exercícios desta apostila e aprofunde seu estudo com os exercícios do livro de Michael Crawley (*The R Book* de 2007) e, principalmente, que você "brinque" no R (tente descobrir como fazer determinada análise, leia os arquivos de ajuda e faça os exemplos apresentados no final dos arquivos de ajuda). O R exige uma dedicação inicial, cujo investimento será recompensado no futuro. No início você poderá se perguntar para que de fato serve "gastar tempo" vendo algumas coisas que parecem um tanto banais, como utilizar o R como calculadora (eu mesma me perguntei isso no meu primeiro contato com o R). Porém, mais para frente você irá perceber que independente da complexidade das análises ou programações que você estiver fazendo, tudo no R segue a mesma lógica que você viu nos primeiros exercícios. Você irá perceber que as possibilidades são infinitas no R, pois aquilo que ele não tem, você cria!

Esta apostila foi elaborada com base no livro *The R Book* de Michael Crawley (2007) e no manual *Introdução ao uso do programa R* de Victor Lemes Landeiro (disponível na página do R www.r-project.org).

1.1 SOBRE O R

O R é tanto uma linguagem de programação como um software. Uma das suas principais vantagens é ser de uso livre, sendo muito usado em computação estatística, manipulação e armazenamento de dados e construção de gráficos. O R foi criado em 1996 por Ross Ihaka e Robert Gentleman da Universidade de Auckland, Nova Zelândia e desde então vem sendo desenvolvido com a colaboração de pessoas de todo o mundo.

Para baixar o R entre na página www.r-project.org, clique em CRAN (Comprehensive R Archive Network), escolha um espelho (CRAN Mirror). No nosso caso podemos usar, por exemplo, o espelho da Universidade Federal do Paraná. Em seguida clique em "Download R for..." (dependerá da plataforma que você utiliza: Linux, Windows ou Mac). Para Windows clique em *base* e salve o arquivo.

Neste curso vamos usar o R junto com um editor, o **RStudio**. Para baixá-lo entre em http://www.rstudio.com/ide/. O RStudio é mais amigável, pois permite a utilização de diferentes janelas (área de trabalho, script, gráficos) ao mesmo tempo, destaca as sintaxes de programação, ajudando a evitar erros, além de outras funcionalidades que veremos. Para começar a usar o R basta abrir o RStudio.

1.2 ÁREA DE TRABALHO

Ao abrir o R (ou o RStudio) ele inicia uma área de trabalho onde você fará suas análises e gráficos. Uma das principais diferenças do R em relação a outros pacotes estatísticos e gráficos é o uso de **linhas de comando** em que você irá digitar o que quer fazer. Serão poucas as vezes em que usaremos 'cliques'. Veja que existem alguns poucos botões em que é possível clicar, como para salvar área de trabalho ou instalar e carregar pacotes. Todo o resto é feito digitando os comandos. A principal vantagem de digitarmos linhas de comando é que temos total controle sobre as ações que serão executadas pelo programa. E exatamente por isso o R é uma ótima forma para aprender e estudar estatística.

Alguns símbolos que você verá na área de trabalho:

- O sinal de maior > indica o *prompt* e o R pronto para receber os comandos.
- O sinal + aparece no lugar do sinal > se um comando digitado não está completo, indicando que o comando não foi finalizado. Isso pode ocorrer devido a um erro de digitação (por

exemplo, é comum esquecer de fechar os parênteses das funções; neste caso pressione a tecla Esc para voltar ao *prompt*) ou pode indicar que o comando continua na próxima linha.

- Nesta apostila, o sinal > indica que uma nova linha de comando está sendo iniciada e o sinal + indica que o comando continua na próxima linha. **Atenção**: não digite > ou +.

Dica importante: sempre, antes de iniciar um novo trabalho, salve a área de trabalho do R na pasta do trabalho em questão (por exemplo, na pasta de um manuscrito ou de uma consultoria). Passe a abrir o R a partir da área de trabalho salva na sua pasta. Isso facilita muito quando você irá retomar o trabalho, pois todas as funções já utilizadas são recarregadas. Crie uma pasta deste curso. No RStudio você tem a opção de criar um **projeto** no qual serão armazenadas todas as informações que foram utilizadas anteriormente. Abra o RStudio e clique em "Project", "New project", "Existing directory" e selecione a pasta do curso. Feche o RStudio e abra-o a partir do ícone salvo na sua pasta. Quando você terminar suas atividades simplesmente feche o RStudio. Neste momento aparecerá uma caixa perguntando se você quer salvar seu trabalho. Clique em "Sim".

Quer conferir o diretório onde você salvou sua área de trabalho? Digite:

```
>getwd()
```

1.3 ALGUNS DETALHES IMPORTANTES

- O R diferencia letras maiúsculas de minúsculas (A ≠ a). Minha dica é que você utilize sempre somente letras minúsculas.
- Não use acentos e espaços nos nomes dos objetos e nos cabeçalhos de tabelas.
- Para separar casas decimais use ponto "."
- Vírgulas separam argumentos de uma função (veremos em seguida).
- Sempre que abrir um parêntesis, um colchete, aspas, não se esqueça de fechá-los (a grande maioria dos erros ocorre devido a isso).

1.4 PACOTES

O R possui atualmente mais de 9000 pacotes para os mais diversos objetivos. Estes pacotes também são gratuitos e podem ser baixados de duas formas:

1) a partir do site do R: após selecionar o espelho CRAN clique em "Packages", selecione o pacote e baixe o arquivo zip. No R vá até "Pacotes", "Instalar pacotes através de arquivos zip locais". **No RStudio clique em "Tools", "Install packages".**

2) diretamente no R, digite:

```
> install.packages ("nome do pacote")
```

Exemplo: instale o pacote *vegan:*

```
> install.packages ("vegan")
```

Depois de instalar o pacote você deve carregá-lo sempre que iniciar o R. Para isso, digite:

```
> library(nome do pacote)
```

Alguns pacotes úteis:

- *vegan*, criado por Jari Oksanen. É um dos pacotes mais usados em análises de dados ecológicos (análises de ordenação, Mantel, índices de dissimilaridade e diversidade, rarefação, etc.).
- *Lattice* e *ggplot2*: gráficos
- *ggmap* e *maps*: mapas

Uma boa forma de explorar os pacotes existentes e descobrir em qual pacote se encontra a análise que você procura é visitar o site http://cran.r-project.org/web/views/. Neste site, os pacotes estão separados em grupos de assuntos. Por exemplo, existe o assunto "*Environmetrics*" com um sub-assunto "*Ordination*" onde há um resumo das análises disponíveis e dos pacotes nos quais as análises são encontradas. Além disso, cada pacote possui um manual.

1.5 MANUAIS DISPONÍVEIS NO SITE DO R

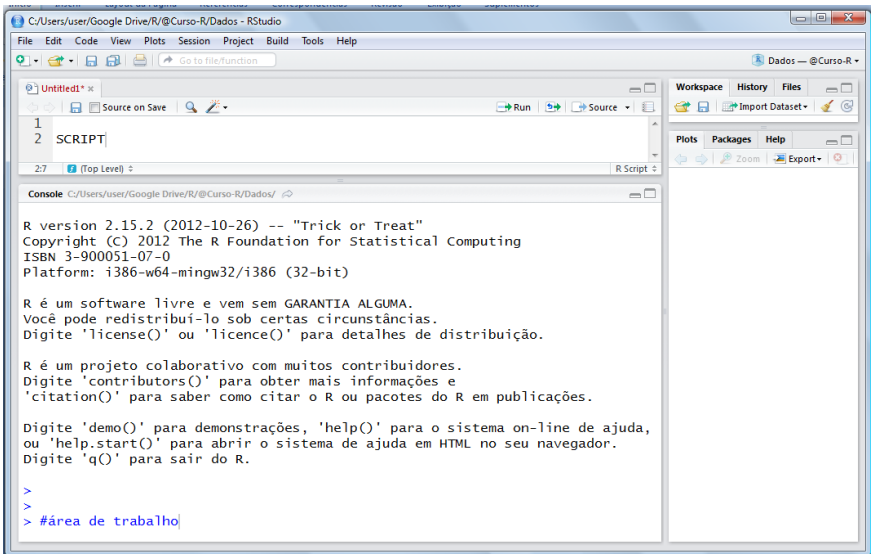
Há uma grande variedade de documentações e manuais disponíveis em várias línguas, incluindo português. Na página do R clique em "Manuals". Um bom manual para iniciantes é ***An Introduction to R*** de Bill Venables e David M. Smith. Além disso, clicando em "Contributed documentation" você terá disponível manuais mais específicos, como para análises estatísticas e construção de gráficos. Atualmente estão disponíveis sete manuais em português, dentre os quais

sugiro ***Introdução ao uso do programa R*** de Victor Lemes Landeiro e ***Estatística aplicada à ecologia usando o R*** de Diogo Borges Provete.

Além de manuais, existem muitas listas de discussão sobre o R, como as listas R-help, R-SIG-ecology e R-SIG-Geo (ver em http://www.r-project.org/mail.html). Também há muitos blogs que podem ser bastante úteis, como por exemplo, o R-Bloggers (http://www.r-bloggers.com) e o Quick-R (http://www.statmethods.net) que trazem tutoriais e novidades sobre funcionalidades do R.

1.6 SCRIPT

Ao invés de digitar seus comandos diretamente na área de trabalho do R, é bastante prático usar um *script* que nada mais é que um arquivo de textos do próprio R. Você digitará todos os comandos neste arquivo. Assim, se você precisar refazer a análise ou o gráfico basta retornar a este arquivo e em um clique rodar novamente os comandos. Para criar um script no RStudio clique em "File", "New R script" e para enviar o seu comando do script para a área de trabalho, tecle Ctrl+Enter. Salve o script na mesma pasta do projeto.



1.7 CALCULADORA

Vamos iniciar digitando alguns comandos simples:

```
>1+2+3+4+5
>5-3
>4*2
>6/3
```

```
>2^3
>((10-2)*3)^2
```

2 FUNÇÕES

No R você vai trabalhar basicamente com funções já existentes. As funções são usadas da seguinte forma:

```
função (argumento1, argumento2,...)
```

Veja abaixo alguns exemplos de funções básicas:

```
> sum(2,2)
> sum(2,2,2)
> sqrt(4)
> sqrt(4*4)
> prod(2,2,4)
> min(2,3,10)
> max(1,10)
> abs(3-9)
> log(10)
```

Vamos usar uma função simples que indica como citar o R:

```
>citation()
To cite R in publications use:
R Core Team (2016). R: A language and environment for statistical
computing. R Foundation for Statistical Computing, Vienna, Austria.
URL https://www.R-project.org/.
```

Para citar pacotes:

```
>citation("vegan")
```

Mas e se você não souber qual função deve usar para determinado propósito? A melhor maneira de descobrir qual função usar e como usá-la é usando a ajuda do próprio R.

3 O ARQUIVO DE AJUDA

Utilizar a ajuda do R é uma excelente maneira de aprender a utilizar as funções. Existem três formas de obter ajuda no R.

Digamos que você **sabe o nome da função** que você irá usar, mas não sabe como usá-la. Vamos ver como usar a função `log` que calcula logaritmos. Basta digitar:

```
> help(log)
```

Ou então:

```
> ?log
```

Irá abrir um arquivo de ajuda desta função. Este arquivo contém alguns tópicos:

Description - apresenta um breve resumo sobre o uso da função

Usage - como utilizar a função e quais os seus argumentos

Arguments - explica cada argumento

Details - detalhes sobre o uso e aplicação da função

Value - mostra a saída (*output*) da função, ou seja, os resultados

Notes - notas gerais sobre a função

Authors - lista os autores da função no R

References – cita as referências dos métodos usados

See also - mostra funções relacionadas

Examples - exemplos de uso da função. São muito úteis para ver como funciona a função e como utilizá-la. Para isto, copie e cole os exemplos no R

Com a ajuda do R descobrimos que a função `log` tem dois argumentos: o primeiro especifica o valor para o qual vamos calcular o logaritmo e o segundo especifica a base do logaritmo. O default é logaritmo natural. Então:

```
> log(10)                    #logaritmo natural de 10;
```

```
> log(10,base = 10)    #logaritmo de 10 na base 10
```

Porém, muitas vezes **não sabemos o nome da função**. Neste caso, podemos pesquisar o assunto sobre o qual queremos ajuda. Por exemplo, se você quiser procurar uma função para calcular logaritmo:

```
> help.search("logarithm")
```

Ao invés de digitar `help.search` você pode simplesmente digitar:

```
>??logarithm
>??"analysis of variance"
```

Se você estiver conectado à internet é possível fazer uma procura nos arquivos de ajuda do site do R:

```
>RSiteSearch("logarithm")
```

Documentos dos pacotes

No RStudio existe a aba *Packages* onde estão listados todos os pacotes que você baixou. Se você clicar em um destes pacotes irá abrir a documentação referente ao pacote. Esta documentação traz todos os arquivos de ajuda das funções disponíveis no pacote. Na aba *Packages* clique em *vegan*, procure pela função `cca` e veja a documentação de ajuda desta função.

4 TIPOS DE OBJETOS

- **Vetores**: sequência de valores numéricos, de caracteres ou lógicos.
- **Matrizes**: conjunto bi-dimensional de vetores (em linhas e colunas); todos os vetores devem ser do **mesmo tipo**.
- **Dataframes**: o mesmo que uma matriz, porém aceita vetores de **diferentes tipos**.
- **Listas**: conjunto de objetos (vetores, matrizes, dataframes); muitas funções criam listas para exibir os resultados.
- **Funções**

5 CRIAÇÃO E MANIPULAÇÃO DE DADOS

5.1 VETORES

TIPOS DE VETORES

Vetor numérico

```
> a<-1
> peixes<-c(20,15,10,11,16,5,3,12)
```

O comando **<-** significa assinalar, ou seja, indica que tudo que está posicionado após **<-** será salvo como um objeto com o nome que vem antes de **<-**. Estes valores digitados acima foram salvos com o nome "peixes". A letra '**c**' é uma função muito importante e que você usará seguidamente. Ela significa combinar e agrupa no objeto "peixes" todos os valores que estão dentro dos parênteses.

```
> peixes
> dados.campo<-c(1:10)      # : gera sequência
```

Vetor caracter

Um vetor de caracter possui letras ou palavras ao invés de números. Veja que as palavras devem estar entre aspas, pois são caracteres.

```
>nomes<-c("Maria", "Joao", "Rafael")
>siglas<-c ("x", "y","z")
```

Fator

Em muitos estudos trabalhamos com variáveis explanatórias categóricas. Estas variáveis são fatores com dois ou mais níveis. Por exemplo, imagine um experimento em que foi avaliado se a adição de nitrogênio aumenta a produção fitoplanctônica em comparação ao tratamento sem adição de nitrogênio. A variável nitrogênio é um fator com dois níveis (com adição e sem adição). A função `factor` transforma um vetor em um fator.

Compare os dois objetos criados abaixo:

```
>nitro.caracter <- c("com","com","sem","sem")
```

```
>nitro.caracter

>nitro.fator <- factor(c("com","com","sem","sem"))
>nitro.fator
```

Fatores também podem ser criados utilizando números ao invés de caracteres:

```
>tratamento<-factor(c(1,1,2,2,3,3,4,4))
>tratamento
```

Vetor lógico

```
>as.logical(c(T,T,T,F,F,F))
```

OPERAÇÕES COM VETORES

```
> a<-sum(2,2)
> a
> sqrt(a)
```

Veja que o resultado da soma 2+2 foi salvo no objeto "a" utilizando a 'flecha'.

Alguns exemplos utilizando o objeto 'peixes' que criamos anteriormente:

```
>peixes
>max(peixes)
>min(peixes)
>peixes/2
>log(peixes)
>peixes.log<-log(peixes)  #salva os valores de peixes logaritmizados

>soma<-sum(peixes)
>comp<-length(peixes)
>media<-soma/comp
>media
```

Resumo de algumas funções básicas para operações com vetores:

Função	**Descrição**
`max(x)`	Valor máximo em x
`min(x)`	Valor mínimo em x
`sum(x)`	Soma de todos os valores em x
`prod(x)`	Produto
`mean(x)`	Média aritmética
`median(x)`	Mediana
`range(x)`	Amplitude de valores
`sd(x)`	Desvio padrão
`var(x)`	Variância
`sqrt(x)`	Raiz quadrada
`log(x)`	Logaritmo natural
`cor(x,y)`	Correlação entre os vetores x e y
`sort(x)`	Valores arranjados em ordem crescente
`rank(x)`	Vetor com os postos dos valores
`cumsum(x)`	Soma de todos os valores até aquele ponto
`sin(x)`	Seno de x
`cos(x)`	Cosseno de x
`abs(x)`	Valor absoluto de x

MANIPULAÇÃO DE VETORES. USO DOS COLCHETES:

```
>peixes[3]          #o terceiro valor do objeto 'peixes'
>peixes[c(2,4,8)]   #2°,4° e 8° valores
```

Veja que para acessar mais de um valor foi utilizada a função **`c`** (combinar).

É possível excluir valores:

```
> peixes[-1]        # o primeiro valor (20) foi excluído
```

Lembre-se que usando somente o comando acima você não irá salvar a operação. Se você digitar `peixes` no R, todos os valores estarão presentes. Você pode **salvar um novo objeto** em que o primeiro valor foi excluído:

```
>novo.peixes<-peixes[-1]
>novo.peixes
>length(novo.peixes)          #comprimento do vetor
>length(peixes)        #o objeto antigo tem um valor a mais que o objeto novo.peixes

>siglas       #caracteres: x,y,z
>siglas[-c(1,3)]    #exclui caracteres x e z
```

É possível substituir valores:

```
>peixes[1]<- 8      #o primeiro valor (20) do objeto peixes é substituído pelo valor 8
>peixes[2]<-10      #o segundo valor (15) do objeto peixes é substituído pelo valor 10
```

OPERAÇÕES LÓGICAS

Operadores relacionais e operadores lógicos

Símbolo	**Descrição**
<	Menor
<=	Menor ou igual
>	Maior
>=	Maior ou igual
==	Igual
!=	Diferente
&	E
|	Ou
!	Não
TRUE ou 1	Verdadeiro
FALSE ou 0	Falso

Crie o objeto `a`:

```
>a<-1
>a
```

```
>a<1        #‘a’ é menor que 1?
>a==1       #‘a’ é igual a 1?
>a>=1       #‘a’ é maior ou igual a 1?
>a!=2       #‘a’ é diferente de 2?
```

```
>peixes<6
>all(peixes >4)
>any(peixes==0)
>peixes==5
>peixes!=5
>sum(peixes!=5)  #o resultado é a soma do número de observações (valores) cujo valor é
                  diferente de 5.
```

Veja que em todos os casos o comando retorna um vetor com TRUE ou FALSE para cada valor. Se quisermos que sejam retornados os valores correspondentes ao teste lógico aplicado, devemos fazer uso dos colchetes:

```
>peixes[peixes<6]
>peixes[peixes!=5]
```

Para extrair do conjunto de dados os valores correspondentes ao teste lógico, basta criar um novo objeto:

```
>peixes.6<-peixes[peixes<6]
>peixes.6
```

Este novo objeto pode ser usado normalmente para fazer operações:

```
>log(peixes.6)
>sum(peixes.6)
```

Indo um pouco mais adiante...

Agora vamos criar um vetor chamado 'invertebrados' para executarmos alguns testes lógicos mais complexos.

```
>invertebrados<-c(70,30,10,20,15,1,21,14)
```

```
> if(any(invertebrados == 0) | any(peixes > 1)) sum(peixes)
```

O comando acima está dizendo: "se algum valor do vetor invertebrados for igual a zero OU se algum valor do vetor peixes for maior que 1, faça a soma do vetor peixes".

```
>if(any(invertebrados!=1) & any(peixes <= 3)) "conferir valores"
```

O comando acima está dizendo: "se algum valor do vetor invertebrados for diferente de 1 E se algum valor do vetor peixes for menor ou igual a 3, dê um aviso para conferir valores".

O comando **|** significa **OU** e o comando **&** significa **E**.

Uma função muito útil é `ifelse` que funciona da seguinte forma:

`ifelse(teste,valor se o teste for verdadeiro, valor se o teste for falso)`

Imagine que você tem um conjunto de dados de abundância de uma espécie de roedor em seis áreas. Porém, você precisa somente de dados de presença ou ausência da espécie e não de abundância. Usando `ifelse` é possível transformar os dados de abundância em dados de presença e ausência:

```
>roedor<-c(10,0,4,2,0,5)
>ifelse(roedor>0,1,0)
```

A função cima diz: se o valor de roedor for maior que zero, coloque 1, caso contrário coloque zero.

Salve em um novo objeto o vetor de presença e ausência de roedores:

```
>roedor.pa<-ifelse(roedor>0,1,0)
```

GERAR SEQUÊNCIAS E REPETIÇÕES

Para gerar sequências é possível utilizar : (dois pontos):

```
>1:12
>-10:0
```

Para gerar sequências com intervalos específicos use a função `seq`:

`seq(from=0,to=10, by=2)`

```
>seq(0,10,2)
>seq(0.3,9,0.5)
```

Para gerar repetições use a função `rep`:

`rep(x,times=y)`, ou seja, rep(repita x, y vezes)

```
>rep(2,10)
>rep(1:5,2)          #repete a sequência 1 a 5 duas vezes
>rep("Maria",6)      #repete "Maria" seis vezes

>rep(c(1,4),times=3)
>rep(c(1,4),each=3)
>rep(c(1,4),each=2, times=3)
>rep(c(1,2,3,4),times=c(1,2,3,4))
```

GERAR AMOSTRAS ALEATÓRIAS

Esta função é bastante útil para aleatorizar dados. Veja abaixo como usá-la:

`sample(dados,tamanho,reposição)`

'dados' é o conjunto de dados do qual as amostras serão retiradas, 'tamanho' é o tamanho da amostra e 'reposição' indica se a amostragem deve ser feita com (TRUE) ou sem reposição (FALSE).

```
>sample(1:10,size=20,replace=T)
>sample(1:10,size=20,replace=F)
```

```
>sample(peixes,size=5)
>sample(peixes,size=length(peixes))
```

EXERCÍCIOS:

1. O pH de amostras de água coletadas em 10 pontos de uma lagoa foi analisado e os valores encontrados foram: 6,6 – 7 – 6,5 – 7,2 – 8 – 5,9 – 7,6 – 6,6 – 8,1 – 7,5.

a) Crie um objeto que contenha estes valores e chame-o de pH. Utilizando as funções que vimos anteriormente, calcule os valores mínimo, máximo e a média do pH da lagoa.

b) O valor de pH 5,9 está errado devido a um erro do aparelho e você decide excluir esta medição. Exclua o valor 5,9 do seu conjunto de dados e recalcule o pH médio.

c) Ordene os valores de pH do menor ao maior.

2. Duas áreas foram selecionadas para um levantamento de vegetação e você denominou estas áreas de: “FLONA” e “Pró-Mata”. Em cada área foram coletadas 3 amostras.

a) Usando a função `rep` crie um objeto chamado `areas` e que contenha o nome “FLONA” três vezes e em seguida o nome “Pró-Mata” também três vezes.
b) Ao invés de um vetor de caracteres, crie um fator com os níveis “FLONA” e “Pró-Mata” de forma que cada nível seja repetido três vezes. Além da função `rep` você deve utilizar a função `factor`.

3. Calcule:
a) o logaritmo natural de 4;
b) o logaritmo de 10 na base 10;
c) raiz quadrada de 36;
d) Crie um objeto chamado ‘x’ e que contenha os valores: 3, 4, 2, 7, 5, 3, 9,1 e em seguida eleve os valores ao quadrado.

4. Vamos pegar uma moeda e ver quantas vezes cai cara e quantas vezes cai coroa em 100 tentativas. Como fazer isso no R? (exercício retirado do manual de Introdução ao R de Victor Landeiro).
a. Criar uma moeda (dica: basta criar um objeto que contenha um vetor de caracteres "cara" e "coroa")
b. Criar um procedimento de amostragem (dica: usar replace = TRUE; não esqueça de salvar um objeto que contenha a amostragem)
c. Quantas caras e quantas coroas apareceram? (dica: basta somar o número de observações de cada tipo (cara ou coroa) no objeto que contém os resultados da amostragem; utilize a função `sum` e o símbolo lógico ==).
d. Use a função `table` para acessar o número de caras e coroas que foram sorteadas:

```
table (nome do objeto)
```

5. Gere um conjunto de amostras aleatórias de tamanho= 10, cujos valores foram obtidos a partir de uma sequência de números entre 1 e 200 com valores de 10 em 10. Use as funções `sample` e `seq`.

6. Crie os objetos:

```
a<-c(2,3,4,5,1)
```

```
b<-c(0,1,5,3,6)
```

a) Teste se **algum** valor de ‘a’ é menor ou igual a 2.
b) Teste se **todos** os valores de ‘a’ são maiores que 3.
c) Teste se os valores de ‘b’ são diferentes de 5.
d) Agora teste se pelo menos algum destes objetos possui valor menor que 1 e crie um “aviso” em caso positivo.
e) Usando a função `ifelse` crie um novo objeto em que todos os valores de ‘b’ maiores que zero sejam substituídos pelo valor 1.

7. Use as funções `union, intersect` e `setdiff` para encontrar a união, o intersecto e as diferenças entre os dois conjuntos de dados abaixo. Aprenda no help como utilizar estas funções (exercício retirado do manual de Introdução ao R de Victor Landeiro).

```
x<-c(1,2,3,4,5)
y<-c(4,5,6,7)
```

5.2 MATRIZES E DATAFRAMES

Relembrando: matriz é um grupo de vetores em linhas e colunas, sendo que todos os vetores devem ser do mesmo tipo (numéricos ou de caracteres). Já um dataframe aceita vetores de diferentes tipos.

TRANSFORMAÇÃO DE VETORES EM MATRIZES E DATAFRAMES

Criando uma matriz:

```
>area.km<-c(50,165,110,400,76,11)    #vetor numérico
>distancia<-c(20,14,2,65,32,15)      #vetor numérico

>cbind(area.km,distancia)        #agrupa os vetores em colunas
>rbind(area.km,distancia)        #agrupa os vetores em linhas
```

```
>matriz<-cbind(area.km,distancia)
>matriz
>is.matrix(matriz)
>is.matrix(area.km)
```

Vamos criar um vetor de caracteres e agrupar este vetor com os vetores numéricos ‘area.km’ e ‘distancia’:

```
>localiza<-c("A","B","C","D","E","F")      #vetor caracteres
>matriz.comp<-cbind(area.km,distancia,localiza)
>matriz.comp
```

O que tem de errado com essa matriz? Veja que os valores numéricos dos objetos ‘area.km’ e ‘distancia’ foram transformados em caracteres. Isso aconteceu porque matrizes somente aceitam vetores do mesmo tipo.

Vamos criar um dataframe com os três vetores. Para isso, ao invés de usar a função `cbind` usamos a função `data.frame`:

```
>dados<-data.frame(area.km,distancia,localiza)
>dados
>is.data.frame(dados)
```

Também é possível transformar o objeto ‘matriz.comp’ diretamente em um dataframe:

```
>dados.1<-as.data.frame(matriz.comp)
>dados.1
```

IMPORTAÇÃO (E EXPORTAÇÃO) DE CONJUNTOS DE DADOS

Anteriormente você criou vetores e os agrupou para criar matrizes e dataframes. Porém, podemos importar os dados ao invés de criar no R.

Exemplo de uma planilha de dados no formato de dataframe. Nas colunas há variáveis numéricas (área, declividade, pH do solo e densidade de minhocas) e categóricas (local e tipo de vegetação). Leia as páginas 107-109 do livro ***The R Book*** de Michael Crawley para mais detalhes sobre a formatação de planilhas do tipo dataframe. **Veja que todos os valores de uma mesma variável devem estar na mesma coluna, sempre!** Desta forma, as observações (amostras) estão nas linhas e as variáveis nas colunas. Isto não é válido somente para o R, sendo a formatação padrão de tabelas de dados.

Local	Área	Declividade	pH solo	Vegetação	Densidade de minhocas
A	3,6	11	4,1	Campo	4
B	5,1	2	5,2	Mata	7
C	2,8	3	4,3	Pinus	2
D	2,4	5	4,9	Mata	5
E	3,8	0	4,2	Mata	6
F	3,1	2	3,9	Campo	2
G	3,5	3	4,2	Pinus	3
H	2,1	0	4,8	Pinus	4
I	1,9	0	5,7	Mata	7
J	1,5	4	5,0	Campo	9
K	2,9	10	5,2	Pinus	5
L	3,3	1	4,1	Campo	6

Existem várias maneiras de utilizar a função `read.table` para importar uma planilha de dados para o R. Esta função lê um arquivo no formato de tabela e cria um dataframe. O mais comum é criar a tabela em planilhas Excel e salvá-las como texto separado por tabulações. No Excel selecione a planilha de interesse, clique em ‘salvar como’ e ‘texto (separado por tabulações)’ e salve o arquivo.txt (dentro do diretório de trabalho que está sendo usado, ou seja, junto ao arquivo do **projeto** que foi criado anteriormente). Em seguida importe o arquivo para o R. Para **importar e salvar** seu arquivo no R, digite:

```
>minhocas<-read.table("minhocas.txt", header=TRUE)
```

O argumento `header=TRUE` deve ser usado quando os dados têm cabeçalho, ou seja, quando a primeira linha contém o nome das variáveis. Caso os dados não tenham cabeçalho deve-se usar `header=FALSE.`

Você também pode procurar os dados no computador:

```
>minhocas<-read.table(file.choose(), header=TRUE)
```

Procure o arquivo minhocas.txt e importe-o e depois visualize o conjunto de dados importado:

```
>minhocas
```

Importante: repare que as colunas com vetores de caracteres (local, vegetação e encharcado) foram transformadas automaticamente em fatores. Confira usando os comandos:

```
> minhocas$encharcado
> minhocas$vegetacao
```

O mais comum em análises de dados é os caracteres representarem diferentes níveis de uma variável categórica (fator), como é o caso das variáveis 'local', 'vegetacao' e 'encharcado'. Caso você não queira que a variável de caracter seja transformada em fator basta acrescentar o argumento `as.is(nome da coluna)` à função:

```
>minhocas<-read.table("minhocas.txt", header=TRUE,
+as.is="encharcado")
> minhocas$encharcado     #viu a diferença?
```

Uma terceira opção é copiar os dados de uma planilha Excel e 'colar' no R. Com esta função os objetos que estiverem na área de transferência serão importados. Porém, para que este comando funcione seu Excel deve estar com o separador decimal configurado como ponto e não vírgula. Selecione uma planilha de dados no Excel usando Ctrl+C. No R digite o comando abaixo:

```
>minhocas<-read.table("clipboard", header=TRUE)
```

Veja que seu conjunto de dados foi copiado para o R:

```
>minhocas
```

```
>names(minhocas)     #nomes das variáveis
```

Também é possível exportar dados do R para outros programas. Caso você queira transferir o dataframe 'minhocas' para o Excel, digite o comando abaixo:

```
>write.table(minhocas,"minhocas.xls",sep="\t",col.names=TRUE,
+row.names=FALSE)
```

Ou ainda, digite este comando e depois no Excel digite Ctrl+V:

```
>write.table(minhocas,"clipboard",sep="\t",col.names=TRUE,
+row.names=FALSE)
```

Para exportar o dataframe para um arquivo .txt:

```
>write.table(minhocas,"minhocas.txt",sep="\t",col.names=TRUE,
+row.names=FALSE, quote=FALSE)
```

Observação: Para mais opções de importação e exportação de dados veja o capítulo 3 do livro *The R Book* de Michael Crawley.

MANIPULAÇÃO DE MATRIZES E DATAFRAMES

Assim como para vetores, usa-se **[ ]** (colchetes) para acessar partes do conjunto de dados. Lembre que para acessar um valor dentro de um vetor usamos, por exemplo, o comando `peixes[3]` para acessar o terceiro valor. Em uma tabela de dados (matriz ou dataframe) devemos **informar as linhas e colunas** que queremos acessar, devendo o uso dos colchetes ser feito da seguinte maneira: **`[linhas,colunas].`**

Digite estes comandos no R e veja o resultado obtido:

```
>minhocas[2,]      #acessa a segunda linha e todas as colunas
>minhocas[4,]      #acessa a quarta linha e todas as colunas
>minhocas[,3]      #acessa os valores de todas as linhas na terceira coluna
>minhocas[2,3]     #acessa o valor da segunda linha e terceira coluna
>minhocas[4,2]     #acessa o valor da quarta linha e segunda coluna
```

```
>minhocas[c(2:4),c(3,5)]
```
#acessa as linhas 2 a 4 e as colunas 3 e 5

Repare que se você digitar os nomes das variáveis do objeto 'minhocas' elas não estarão acessíveis:

```
> densidade
Erro: objeto 'densidade' não encontrado
```

Existem diversas formas de acessar as variáveis em um objeto. Aqui usaremos o **$** (cifrão), pois seu uso é bastante simples: `dados$variável`

```
>minhocas$inclinacao
>minhocas$vegetacao
>minhocas$densidade
```

OPERAÇÕES COM DATAFRAMES

Agora vamos fazer algumas operações usando o objeto 'minhocas'. Primeiro vamos calcular a média da densidade de minhocas:

```
>mean(minhocas$densidade)
```

Também podemos calcular diretamente todas as **médias das variáveis (médias das colunas)** utilizando a função `colMeans`. Importante: a função não funciona se houverem variáveis categóricas.

```
>colMeans(minhocas)
```
#não funciona, pois há variáveis categóricas (local, vegetação e encharcado)

```
>colMeans(minhocas[,2:5])
```
#selecionamos somente as variáveis numéricas (2:5)

Porém, estamos interessados em saber a densidade média separadamente para cada tipo de vegetação. Para isso usamos a função **`tapply(dados, grupo, função)`**. No nosso caso, **dados** corresponde à **densidade**, **grupo** corresponde à **vegetação** e **função** corresponde à **média**.

```
>tapply(minhocas$densidade,minhocas$vegetacao,mean)
```

Agora calcule a inclinação média do terreno e o pH do solo para cada tipo de vegetação. Confira abaixo se você utilizou a função de forma correta:

```
>tapply(minhocas$inclinacao,minhocas$vegetacao,mean)
>tapply(minhocas$pH.solo, minhocas$vegetacao,mean)
```

Novamente estamos interessados nas médias de cada variável separadamente para cada tipo de vegetação. Aqui usamos a função `by` que é bastante semelhante à função `tapply`, porém enquanto `tapply` aplica a função desejada (no caso a média) em uma coluna de cada vez, `by` aplica a função em todas as colunas (variáveis).

```
>by(minhocas[,2:5],minhocas$vegetacao,colMeans)
>by(minhocas[,2:5],minhocas$encharcado,colMeans)
```

Para obtermos um sumário geral da tabela de dados:

```
>summary(minhocas)
```

Utilizando operações lógicas para selecionar dados em uma tabela:

Uma operação que comumente precisamos fazer é selecionar determinadas linhas do dataframe com base em valores de uma ou mais variáveis. Suponha que você queira selecionar somente os **dados dos ambientes encharcados**. Ou seja, devem ser selecionadas todas as colunas, mas somente algumas linhas específicas (aquelas que correspondem aos ambientes encharcados). Aqui, usamos novamente os colchetes utilizando a seguinte sintaxe: `[linhas,colunas]`. Devemos informar quais linhas queremos selecionar e que queremos todas as colunas: **`['quais linhas',]`.**

```
>minhocas[minhocas$encharcado=="sim",]  #selecionamos os dados dos ambientes encharcados
```

Podemos querer selecionar ambientes encharcados e que tenham uma densidade de minhocas maior que 5. Para isso, usamos o operador lógico **&**.

```
>minhocas[minhocas$encharcado=="sim" & minhocas$dens>5 ,]
```

Finalmente, vamos selecionar todos os dados que não são do local A (ou seja, **diferentes** de A). Você consegue imaginar como deve ser esse comando?

```
>minhocas[minhocas$local!="A",]
```

EXERCÍCIOS

Importe para o R o arquivo peixes.txt. Este conjunto de dados contém informações sobre a abundância da comunidade de peixes em 18 lagoas em que foram medidas as seguintes variáveis: área da lagoa, se a lagoa está localizada em uma área urbana ou não, oxigênio dissolvido, condutividade, fosfato, clorofila-a e composição da comunidade de peixes (abundância de cinco espécies).

a) Veja um sumário geral do conjunto de dados e obtenha os valores máximos de oxigênio dissolvido e clorofila-a.

b) Calcule a área média das lagoas.

c) Calcule as concentrações médias de oxigênio dissolvido em lagoas em zonas urbanas e zonas não urbanas usando a função `tapply`.

d) Selecione somente os dados coletados em zonas urbanas E em lagoas com área maior que 100 ha.

e) Calcule a abundância total de peixes **em cada lagoa**. Use a função `rowSums`.

f) Usando o objeto 'peixes' crie um novo objeto que contenha somente as cinco espécies de peixes. Chame este objeto de 'spp'. Crie outro objeto chamado 'amb' que contenha as variáveis ambientais (area,urbana, OD, condutividade, fosfato e clorofila).

g) Calcule a abundância total e a abundância média **de cada espécie de peixe** usando o objeto 'spp' que você criou. Use as funções `colSums` e `colMeans`.

h) Os dados de abundância das espécies de peixes podem ser transformados em dados de presença-ausência. Transforme os dados de abundância do objeto 'spp' em dados de presença e ausência, ou seja, 0 e 1. Salve com o nome 'spp.pa'.

i) Calcule a riqueza de espécies de peixes em cada lagoa usando o objeto 'spp.pa'. Use a função `rowSums`.

j) Imagine que você tenha dois conjuntos de dados que estão em dataframes separados (como é o caso dos conjuntos que criamos acima 'amb' e 'spp') e você quer juntá-los em um único dataframe. Vimos que para juntar dois vetores podemos usar a função `cbind`. Veja se esta mesma função pode ser usada para juntar dois dataframes.

6 CRIAÇÃO E MANIPULAÇÃO DE GRÁFICOS

O R produz muitos tipos de gráficos. Aqui veremos como criar alguns tipos mais comuns, como gráficos de dispersão, boxplots, de barras, pizza, etc., e como alterar a aparência dos gráficos. O capítulo 5 do livro The R Book (Crawley, 2007) traz uma grande variedade de gráficos disponíveis no R. Para uma demonstração das muitas possibilidades de gráficos, digite:

```
>example(plot)
>demo(graphics)
>demo(persp)
>demo(image)
```

Comumente queremos produzir gráficos que contenham uma variável resposta (no eixo y) e uma variável explanatória (no eixo x). O tipo de gráfico dependerá da natureza da variável explanatória: i) se ela for contínua (e.g., precipitação, área, tamanho), o gráfico adequado é um gráfico de dispersão; ii) se for categórica (e.g., tipo de vegetação, estação do ano, tratamento), os gráficos mais apropriados são *box-and-whisker plots* e gráficos de barras. As funções para fazer estes tipos de gráficos são bastante simples:

`plot(x,y)` #gráfico de dispersão de y em relação a x

`plot(fator,y)` #gráfico do tipo *box-and-whisker* de y nos níveis de x

`barplot(y)` #gráfico em barras dos valores de y

GRÁFICOS DE DISPERSÃO

Importe para o R o conjunto de dados 'dispersão1' e faça um gráfico usando a função `plot`:

```
>dados<-read.table("dispersao1.txt",h=T)
>names(dados)        #os nomes das variáveis são 'x' e 'y'
>plot(dados$x,dados$y)
```

Podemos alterar a aparência do gráfico. Primeiro vamos mudar a cor dos pontos usando o argumento `col`:

```
>plot(dados$x,dados$y,col="green")
```

Além do mudar a cor, podemos mudar o símbolo com o argumento `pch`. O símbolo default que usamos no gráfico acima é `pch=1` (círculo vazio). Vamos alterar o símbolo para um triângulo usando `pch=2`.

```
>plot(dados$x,dados$y,col="red",pch=2)
```

Digite os comandos abaixo para ver as cores e os principais símbolos disponíveis:

```
>plot(1:8,col=1:8,pch=15,cex=4)
>text(1:8,c("black","red","green","blue","cyan","magenta",
+"yellow","gray"))

>example(colors)
>pie(rep(1,times=25),col=rainbow(25))
```

Símbolos:

```
>plot(rep(1:5,times=5),rep(1:5,each=5),pch=1:25,cex=2,axes=F,
+ann=F,bg="red")
>text(rep(1:5,times=5),rep(1:5,each=5),labels=1:25,pos=1)
```

Vamos voltar ao gráfico com o objeto `dados` e adicionar um título com o argumento `main`:

```
>plot(dados$x,dados$y,col="red",pch=1,main="Gráfico de dispersão")
```

Também devemos adicionar títulos nos eixos com os argumentos `xlab` e `ylab` :

```
>plot(dados$x,dados$y,col="red",pch=1,main="Gráfico de dispersão",
+xlab="Variável explanatória", ylab="Variável resposta")
```

Podemos querer inserir um segundo conjunto de dados ao gráfico já existente. Imagine que os dados anteriores foram coletados em um determinado local e agora queremos adicionar ao gráfico dados coletados em outro local e diferenciar os pontos dos dois locais. Primeiro vamos importar o novo conjunto de dados dispersão2.txt:

```
>dados2<-read.table("dispersao2.txt",h=T)
```

Para inserir estes dados no gráfico usamos a função `points`. Vamos usar o símbolo de círculo cheio na cor azul e aumentar o tamanho do símbolo para diferenciar os novos pontos dos demais:

```
>points(dados2$x,dados2$y,pch=16,col="blue", cex=2)
```

O argumento `cex` altera o tamanho dos símbolos.

Se você reparou bem no gráfico deve ter percebido que alguns pontos estão fora dos limites do gráfico. Isso ocorreu porque o R criou os eixos do gráfico com base no primeiro conjunto de dados. Existem duas maneiras de resolver este problema.

A primeira forma é definir o limite dos eixos antes de fazer o primeiro gráfico. Para isso precisamos saber a amplitude das variáveis x e y considerando conjuntamente os dois conjuntos de dados:

```
>range(c(dados$x,dados2$x))  #amplitude da variável x nos dois conjuntos de dados
```

```
>range(c(dados$y,dados2$y))  #amplitude da variável y nos dois conjuntos de dados
```

Não precisamos alterar os limites do eixo x, mas devemos alterar do eixo y. Para plotar o primeiro gráfico já com os eixos escalonados de forma a conter todos os pontos vamos alterar o limite do eixo y de forma que ele comece em 10 e vá até 63. Depois basta adicionar o segundo conjunto de dados.

```
>plot(dados$x,dados$y,col="red",ylim=c(10,63))
>points(dados2$x,dados2$y,col="blue")
```

Outra possibilidade é plotarmos os eixos do gráfico diretamente com base nos dois conjuntos de dados. Assim, os eixos serão escalonados de forma a conter todos os pontos de todos os conjuntos de dados. O argumento `type="n"` significa que nada será plotado dentro do gráfico:

```
>plot(c(dados$x,dados2$x),c(dados$y,dados2$y),type="n")
```

Agora basta adicionar os pontos dos dois conjuntos de dados:

```
>points(dados$x,dados$y,col="red")
>points(dados2$x,dados2$y,col="blue")
```

Pronto, nosso gráfico está correto!

Também é possível inserir linhas com a função `abline`. Vamos inserir uma linha indicando o valor médio da variável y do primeiro conjunto de dados. O argumento `h` indica que a linha será plotada horizontalmente, indicando assim o ponto em que passa a média de y.

```
>abline(h=mean(dados$y),col="red")
```

E agora o valor médio da variável y do segundo conjunto de dados:

```
>abline(h=mean(dados2$y),col="blue")
```

Se utilizarmos `v` ao invés de `h`, a linha será plotada verticalmente. Vamos inserir a média de x do primeiro conjunto de dados:

```
>abline(v=mean(dados$x))
```

Muitas vezes queremos adicionar linhas de tendências (a partir de uma regressão linear) aos nossos dados. Como veremos mais adiante, a função `lm(y~x)` calcula uma regressão linear de y modelado por x. Neste momento vamos nos ater a utilizar a função `lm` para gerar uma linha de tendência dos dados.

Vamos primeiro refazer o gráfico:

```
>plot(c(dados$x,dados2$x),c(dados$y,dados2$y),type="n",
+xlab="Variável explanatória", ylab="Variável resposta")
>points(dados$x,dados$y,col="red")
>points(dados2$x,dados2$y,col="blue")
>abline(lm(dados$y~dados$x)) #linha de tendência do 1° conjunto
>abline(lm(dados2$y~dados2$x)) #linha de tendência do 2° conjunto
```

Agora vamos inserir um texto ao gráfico usando a função **`text(cooX,cooY,"texto");`** `cooX e cooY` indicam as coordenadas dos eixos X e Y onde o texto será inserido e `"texto"` indica o texto que será inserido.

```
>text(20,40,"ponto 1")
>text(60,30,"ponto 2")
```

A função `locator(n)` permite que você localize com cliques do mouse a posição no gráfico onde o texto será inserido. O `n` indica o número de pontos que você quer indicar no gráfico. Após inserir o comando abaixo no R clique com o *mouse* no local do gráfico onde você quer inserir o texto:

```
>text(locator(1),"ponto2") #clique em um ponto
>text(locator(3),c("ponto3","ponto4","ponto5")) #clique em três pontos
```

Para finalizar este gráfico vamos adicionar uma legenda:

```
>legend(locator(1),c("Vermelho","Azul"),pch=c(1,1),col=c("red",
+"blue"))
```

Adicionando polígonos a gráficos:

Vamos inserir dados de duas variáveis numéricas usando a função `rnorm` que gera dados aleatórios com distribuição normal:

```
>x<-rnorm(100)
>y<-rnorm(100)
```

Vamos fazer o gráfico:

```
>plot(x,y)
```

E agora adicionar um polígono ao redor dos dados usando as funções `polygon` e `chull`:

```
>polygon(x[chull(x,y)],y[chull(x,y)],dens=10, col="blue")
```

Em uma explicação bastante simplista, a função `chull` significa "convex hull" e delimita o contorno convexo dos pontos no espaço Euclidiano. A função `polygon` vai utilizar esses valores definidos do contorno para criar o polígono.

Agora vamos criar dois conjuntos de dados: o conjunto (x1,y1) e o conjunto (x2,y2):

```
>x1<-rnorm(10,-1)      #10 valores com média = -1
>y1<-rnorm(10,-5)      #10 valores com média = -5
>x2<-rnorm(10,10)
>y2<-rnorm(10,10)
```

Criar um gráfico "vazio" e escalonar os eixos para que o gráfico contenha todos os pontos:

```
>plot(c(x1,x2),c(y1,y2),type="n")
```

Inserir os pontos:

```
>points(x1,y1,col="red")
>points(x2,y2,col="blue")
```

Agora podemos inserir os polígonos que delimitam os dois conjuntos de dados:

```
>polygon(x1[chull(x1,y1)],y1[chull(x1,y1)],dens=15, col="red")
```

```
>polygon(x2[chull(x2,y2)],y2[chull(x2,y2)],dens=15, col="blue")
```

Altere o argumento `dens` para ver o que acontece.

GRÁFICOS COM VARIÁVEIS EXPLANATÓRIAS CATEGÓRICAS

Quando a variável explanatória (x) é categórica ao invés de contínua não é possível fazer gráficos de dispersão, mas sim gráficos do tipo *box-and-whiskers* (boxplot), *stripcharts* ou em barras (barplot). Lembre-se que variáveis categóricas são fatores com dois ou mais níveis. Por exemplo, vamos usar dados que representam a concentração de cianotoxinas em dois tratamentos: Com e sem a adição de nutrientes. A variável categórica é o fator 'nutrientes' e os dois níveis são 'com' e 'sem'.

BOXPLOTS

Importe o conjunto de dados ciano.txt:

```
ciano<-read.table("ciano.txt",h=T)
names(ciano)
```

Este conjunto de dados representa um experimento que avaliou, entre outros fatores, o efeito da adição de nutrientes sobre a produção de toxinas de cianobactérias. O fator nutrientes conta com dois níveis: controle e adição de P. Use a função `plot` para construir um gráfico da resposta das plantas aos tratamentos com nutrientes.

```
>plot(ciano$nutri,ciano$toxina,xlab="Tratamento",ylab="Toxina")
```

Vamos ver outro exemplo, o caso das macrófitas descrito anteriormente. Importe o conjunto de dados reservatório.txt:

```
>reserva<-read.table("reservatorio.txt",h=T)
>reserva

>reserva$periodo
```

Veja que o R considerou a variável período como uma variável numérica. Isso ocorreu porque colocamos na tabela os valores 1 e 2 representando os períodos antes e após, respectivamente. Até aqui sempre havíamos usado palavras ou letras para representar níveis de um fator e por isso o R reconhecia as variáveis como fatores com seus níveis. Porém, é muito comum usarmos números para designar níveis. Basta informarmos ao R que a variável período é um fator:

```
>periodo<-factor(reserva$periodo)
>periodo        #percebeu a diferença?
```

Para facilitar o uso das variáveis vamos salvar cada uma delas separadamente:

```
>enra<-reserva$enrai
>flut<-reserva$flutu
```

Agora, use a já conhecida função `plot` para construir um gráfico que represente a abundância de macrófitas enraizadas em cada um dos períodos:

```
>plot(periodo,enra)  #macrófitas enraizadas em cada período
```

Veja que como a variável x é um fator, a função `plot` faz automaticamente um boxplot ao invés de um gráfico de dispersão.
No eixo x do gráfico aparecem os números 1 e 2 indicando os períodos antes e após a formação do reservatório. Vamos nomear os níveis 1 e 2 e refazer o gráfico:

```
>levels(periodo)<-c("Antes","Após")
>plot(periodo,enra)
```

Adicione títulos nos eixos x e y.

```
>plot(periodo,enra,xlab="Período",ylab="Abundância de macrófitas")
```

Ao invés de fazermos gráficos separados para cada grupo de macrófitas, podemos inserir as informações em um único gráfico. Primeiro insira as macrófitas enraizadas:

```
> plot(periodo,enra, boxwex=0.3, at=1:2-0.2, col="orange",
+ylim=c(4,24), xlab="Período",ylab="Abundância",xaxt="n")
```

O argumento `boxwex` indica a largura da 'caixa' (box); o argumento `at` indica a posição em que as duas caixas serão plotadas no eixo x, neste caso 0,2 à esquerda das posições 1 e 2; o argumento `xaxt` indica que o texto do eixo x indicando os níveis do fator não será plotado.

Agora insira as macrófitas flutuantes. Repare no argumento `add=TRUE`. É este argumento que faz com que as informações do novo gráfico sejam adicionadas ao gráfico anterior.

```
>plot(periodo,flut, add=TRUE, boxwex=0.3, at=1:2+0.2,col="green",
+xaxt="n")
```

Vamos inserir o texto no eixo x indicando os níveis. Para isso usaremos a função `axis` que diz para plotar um eixo, neste caso o eixo x inferior (`side=1`).

```
>axis(side=1,labels=c("Antes","Após"),at=1:2)
```

Insira a legenda. Veja que aqui há um argumento que não havíamos visto quando adicionamos a legenda no gráfico de dispersão: `fill` indica as cores com as quais queremos preencher as caixas. Já o argumento `bty="n"` indica que a legenda não terá uma borda.

```
>legend(locator(1),c("Enraizadas", "Flutuantes"), fill=c("orange",
+"green"),bty="n")
```

STRIPCHARTS

Outra forma de representar estes dados é usar a função `stripchart`:

```
>stripchart(enra~periodo,vertical=T,at=c(1.3,1.7))
```

#vertical=T indica que o gráfico será na posição vertical

Repare que aparecem cinco pontos indicando a abundância de macrófitas em cada período, mas se olharmos na tabela veremos que cada período tem seis amostras (ou seja, deveriam ser seis pontos em cada). Isso aconteceu porque em cada um dos períodos temos duas amostras com os mesmos valores de abundância, de forma que estes pontos se sobrepuseram. Vamos usar o argumento `method="stack"` para separar os pontos sobrepostos.

```
>stripchart(enra~periodo,vertical=T,at=c(1.4,2), method="stack")
```

Os pontos ficaram visíveis, porém ainda estão sobrepostos. Usamos o argumento `offset` para separá-los:

```
>stripchart(enra~periodo,vertical=T,at=c(1.4,2),method="stack",
+offset=1)
```

GRÁFICOS DE BARRAS

Vamos criar três vetores, cada um representando uma espécie e contendo 4 valores com a abundância da espécie em 4 locais. E vamos agrupar estes vetores em uma matriz:

```
>sp1<-c(80,55,60,35)
>sp2<-c(70,45,50,30)
>sp3<-c(10,30,55,40)
>abund<-cbind(sp1,sp2,sp3)
```

Para fazer o gráfico use a função `barplot`:

```
>barplot(abund)
>barplot(abund,beside=T,legend=c("grad1","grad2","grad3","grad4"))
```

Ou ainda:

```
>barplot(t(abund))
```

A função `t` significa transpor.

Podemos transformar os dados para proporção para depois fazer um gráfico.

```
>abund.perc<-prop.table(abund,margin=2)
```

O argumento `margin` define se a proporção será calculada para linhas (`margin=1`) ou para colunas (`margin= 2`). Usando `margin= 2` calculamos a proporção de cada espécie por local.

Veja o conjunto de dados em proporção:

```
>abund.perc
```

Agora faça o barplot:

```
>barplot(abund.perc*100)
```

Veja alguns exemplos de gráficos de barras feitos no R:

```
>example(barplot)
```

OUTROS TIPOS DE GRÁFICOS

HISTOGRAMAS

Muitas vezes temos somente uma variável e queremos mostrar a distribuição de frequência desta variável a partir de um histograma. Vamos usar a função `hist` para fazer um histograma de 10000 valores aleatórios com distribuição normal:

```
>hist(rnorm(10000))
```

GRÁFICOS DE PIZZA

```
>pie(c(20,10,15,30,60))
>pie(c(20,10,15,30,60),labels=c("A","B","C","D","E"))
>pie(c(20,10,15,30,60),labels=c("A","B","C","D","E"),
+col=gray.colors(5))
>pie(c(20,10,15,30,60),col=c("maroon","aquamarine","lightyellow",
+"lavender","tomato"))
```

GRÁFICOS COM MÚLTIPLAS VARIÁVEIS

Veremos algumas funções que permitem a visualização de múltiplas variáveis em uma única figura. A função `pairs` compara todas as variáveis entre si. Veja como funciona:

```
>ozonio<-read.table("ozonio.txt",h=T)
>names(ozonio)
>pairs(ozonio)
```

Suponha que as 111 observações tenham sido coletadas em três diferentes locais (com 37 observações em cada local). Vamos colocar cores nos símbolos para diferenciar as três localidades:

```
>pairs(ozonio,col=rep(c(1,2,3),each=37))
```

Uma forma muito interessante de visualizar dados com mais de duas variáveis é usar gráficos de interação. O conjunto de dados ciano.txt (já importado com o nome `ciano`) ilustra um experimento com dois fatores: nutrientes (com dois níveis: controle e adição de P) e temperatura (controle e alta) em que o objetivo era avaliar se tais fatores apresentavam efeitos sobre a concentração de cianotoxinas. Vamos usar a função `interaction.plot` para visualizar estes dados. A ordem dos argumentos é um pouco diferente aqui:

`interaction.plot (fator no eixo x, fator representado por linhas, variável resposta)`

Vamos colocar no eixo x o fator nutrientes, em linhas o fator temperatura e como variável resposta a concentração de toxinas:

```
>interaction.plot (ciano$nutri,ciano$temp,ciano$tox)
```

```
>interaction.plot(ciano$nutri,ciano$temp,ciano$tox,xlab="Nutrient
es",ylab="Cianotoxinas",trace.label="Temperatura")
```

`#trace.label` adiciona título na legenda

O gráfico indica que a resposta das toxinas à adição de fósforo depende da temperatura: na temperatura controle a adição de P tem pouco efeito, mas em altas temperaturas o efeito de P é pronunciado. A interação é evidenciada pelas linhas não paralelas no gráfico.

Pode-se também inserir uma variável no terceiro eixo de um gráfico com a função `plot`. Vamos fazer um gráfico de oxigênio dissolvido e temperatura ao longo de um perfil de profundidade:

```
>prof<-c(0,-5,-10,-15,-20,-25,-27)
>od<-c(8.97,6.67,5.41,6.41,5.31,3.25,3.15)
>temp<-c(26.4,22.1,21.8,24.7,21.5,21.0,21.3)
```

```
>plot(temp,prof, type="o",xlim=c(15,30),col=2,  xlab="Temperatura
+(°C)",ylab= "Profundidade (m)",las=1)
>par(new=T)
>plot(od,prof, type="o", xlim=c(0,10),col=4, axes=F, ann=F)
>axis(3)
```

```
>mtext(expression(OD~(mg~L^-1)),side=3, line=2)
```

Novos argumentos:

- `las` especifica a orientação do texto nos eixos (pode ser 1,2 ou 3)
- `axes` especifica se os eixos devem ser plotados (FALSE ou TRUE)
- `ann` especifica se devem ser plotados títulos automáticos nos eixos (FALSE ou TRUE)

Novas funções:

- `par` é uma função que controla várias características dos gráficos. Na ajuda desta função você encontrará muitos argumentos para alterar a aparência de gráficos. O argumento `new=T` especifica que o próximo comando não deve limpar a janela antes de plotar um **novo** gráfico, ou seja, o segundo gráfico será plotado "em cima" do primeiro.
- `axis` diz para plotar um eixo, neste caso o eixo 3 (do OD), `axis=1` plota o eixo x inferior e `axis=2` plota o eixo y.
- `mtext` insere textos nas margens dos gráficos; `side=3` especifica a posição (eixo x superior) e `line=2` especifica a a linha em que o texto aparecerá.

GRÁFICOS DE CONTORNO E 3D

Usaremos o conjunto de dados `volcano` (que representam as variações de nível de um terreno; disponível no R) e as funções **`filled.contour (x,y,z)`** e **`contour (x,y,z)`**, onde:

`x` deve ser uma sequência numérica com comprimento igual ao número de linhas de `z`,

`y` deve ser uma sequência numérica com comprimento igual ao número de colunas de `z`,

`z` é a matriz que contém os valores que serão plotados (`z` = matriz `volcano` com os valores de altitude do terreno).

```
>x<-1:nrow(volcano)
>y<-1:ncol(volcano)
>filled.contour(x,y,volcano)
```

Vamos alterar alguns detalhes de aparência: acrescentar legendas aos eixos (`xlab` e `ylab`), acrescentar um título à legenda de cores (`key.title`) e alterar as cores:

```
>filled.contour(x,y,volcano,xlab="Metros(x)",ylab="Metros(y)",
+key.title=title(main="Altura(m)",cex.main=0.9),color=
+terrain.colors)
```

Com a função **`contour (x,y,z)`** é possível criar curvas de nível:

```
>contour(x,y,volcano)
```

Instale e carregue o pacote `rgl` para uma demonstração de gráficos 3D:

```
>install.packages("rgl")
>library(rgl)
>demo(rgl)
```

Vamos continuar usando o conjunto de dados `volcano` como exemplo para a construção de gráficos 3D. Nestas funções também usamos a notação **`função (x,y,z)`** :

```
>plot3d(x,y,volcano,col=rainbow(1000))
>persp3d(x,y,volcano,col="lightblue")

>plot3d(x,y,volcano)
>surface3d(x,y,volcano,col="green")
```

Agora vamos fazer gráficos com dados que representam a riqueza de árvores (z) ao longo de um gradiente de produtividade (x) e de um gradiente de distúrbios (y). Importe o conjunto de dados `arvores` (com cabeçalho) e carregue o pacote `lattice`. As funções abaixo são escritas da seguinte forma: **`função(z~x*y)`**

```
>library(lattice)

>wireframe(riq~produti*disturb,data=arvores, drape=T,
+xlab="produtividade",ylab= "distúrbio")

>wireframe(volcano,shade=T)
```

Vejam outras opções com as funções `cloud e levelplot:`

```
>cloud(riq~produti*disturb,data=arvores,type="h",xlab="prod",
+ylab="disturb")   #altere type="h" para "p" ou "l"

>levelplot(riq~produti*disturb,data=arvores)
```

VÁRIOS GRÁFICOS EM UMA JANELA

Para dividir a janela dos gráficos usamos **`par(mfrow=c(n° linhas, n° colunas)`**, indicando o número de linhas e de colunas que a janela deverá ter. Vamos usar os dados de concentração de cianotoxinas (conjunto de dados ciano.txt).

```
>par(mfrow=c(1,2))              #uma linha e duas colunas
>plot(ciano$temp,ciano$tox)
>plot(ciano$nutri,ciano$tox)

>par(mfrow=c(2,2))              #duas linhas e duas colunas
>plot(ciano$temp,ciano$tox)
>plot(ciano$nutri,ciano$tox)
>plot(ciano$temp,ciano$tox)
>plot(ciano$nutri,ciano$tox)
```

7 CRIAÇÃO E MANIPULAÇÃO DE MAPAS

O R possui vários pacotes para criação e manipulação de mapas, desde pacotes que possuem bancos de dados com mapas de todo o mundo e pacotes que permitem a importação de imagens do Google Maps, até pacotes sofisticados de análises espaciais. Veja exemplos de mapas criados no R em: http://spatialanalysis.co.uk/

Vamos baixar e carregar o pacote *maps*:

```
>install.packages("maps")
>library(maps)
```

Neste pacote está disponível um banco de dados com os limites geográficos de todos os países. Para criar os mapas usamos a função `map` da seguinte maneira:

`map (banco de dados,região)`

```
>map("world")                #banco de dados “world”
>map("world", "Brazil")      #banco de dados “world”, região “Brazil”
```

Vamos fazer um mapa com todos os países da América do Sul:

```
>map("world",  c("Brazil",  "Argentina","Uruguay",  "Paraguay",
+"Chile","Venezuela","Peru","Ecuador","Colombia","Bolivia",
+"Suriname","Guyana","French Guiana"))
```

Podemos inserir pontos com a função `points` (a mesma que usamos para inserir pontos em gráficos). Use-a assim: `points (longitude, latitude)`

```
>points(-51.22, -30, pch=19, col=1)      #Porto Alegre
>points(-46.63, -23.53, pch=19, col=2)   #São Paulo
```

Também podemos inserir texto:

```
>text(-50,-10,"Brazil", cex=1.5)     #cex=tamanho da fonte
```

Veja o que fazer as funções `map.axes` e `map.scale`:

```
>map.axes()
>map.scale(-55,-40, cex=0.6)  #os valores -55 e -40 indicam a posição da escala
```

Outro banco de dados disponível é o `world.cities`. Vamos carregá-lo usando a função `data`:

```
>data(world.cities)
```

Veja quais as informações que ele traz: nome da cidade, país, população, latitude, longitude e se é ou não uma capital:

```
>names(world.cities)
```

Vamos inserir algumas cidades no nosso mapa do Brasil com a função `map.cities` e o banco de dados `world.cities`:

```
>map("world", "Brazil")
>map.cities(world.cities[(world.cities$name=="Porto Alegre"),],
+country="Brazil",cex=1, font=2)
```

Veja que usamos os colchetes `[linhas, colunas]` novamente para selecionar uma determinada informação. Vamos ver a informação contida nos colchetes:

```
world.cities [(world.cities$name=="Porto Alegre"),vazio]
```

Estamos dizendo que do conjunto de dados `world.cities` queremos selecionar [na coluna chamada name as linhas que sejam iguais a "Porto Alegre"**, e todas as colunas destas linhas**]. Para conferir, digite:

```
>world.cities[(world.cities$name=="Porto Alegre"),]
```

Insira outras cidades no mapa.

Usando comandos lógicos vamos selecionar as cidades brasileiras do conjunto de dados `world.cities`. Para isso precisamos informar que queremos selecionar na variável `country` (coluna 2) todas as linhas que tem a palavra "Brazil":

```
world.cities[linhas,colunas]

>world.cities[(world.cities$country=="Brazil"),]
```

Agora vamos selecionar todas as cidades chamadas Rio Grande, independente do país:

```
>world.cities[(world.cities$name=="Rio Grande"),]
```

Queremos somente cidades chamadas Rio Grande no Brasil. Veja que usamos o comando lógico **&** para selecionar todas as linhas que tem "Rio Grande" na coluna `name` E "Brazil" na coluna `country`:

```
>world.cities[(world.cities$name=="Rio Grande" &
+world.cities$country=="Brazil"),]
```

Mais um exercício: crie um mapa do Brasil e insira neste mapa todas as cidades do Brasil disponíveis no banco de dados `world.cities`. Crie outro mapa e insira neste mapa todas as cidades do Brasil com população maior que 1 milhão de pessoas. Conseguiu?

Agora vamos inserir ao mapa do Brasil pontos que representam os aeroportos do país. Vamos importar da internet um conjunto de dados que contém as coordenadas geográficas dos aeroportos. Veja no site http://openflights.org/data.html os dados que este conjunto possui.

```
>aero<-read.csv("https://raw.githubusercontent.com/jpatokal/
+openflights/ master/data/airports.dat", header = FALSE)
```

Vamos nomear as colunas com os nomes das variáveis:

```
>colnames(aero) <- c("ID", "nome", "cidade", "pais","IATA_FAA",
+"ICAO", "lat", "lon", "altitude", "timezone", "DST")
```

Vamos olhar as primeiras linhas do conjunto de dados:

```
>head(aero)
```

Esse conjunto de dados contém os aeroportos de todo o mundo. Vamos selecionar somente os aeroportos brasileiros:

```
>aero.br<-aero[aero$pais=="Brazil",]
```

Pronto. Crie o mapa e insira os pontos:

```
>map("world","Brazil")
>points(aero.br$lon,aero.br$lat)
>map.axes()
```

MANIPULAÇÃO DE ARQUIVOS *SHAPE*

O R possui muitos pacotes para leitura e manipulação de dados geográficos (como arquivos *shape*). Um deles é o *maptools*.

```
>install.packages("maptools")
>library(maptools)
```

Vamos usar alguns arquivos *shape* com informações do Rio Grande do Sul disponíveis na página da FEPAM (Fundação Estadual de Proteção Ambiental Henrique Luiz Roessler-RS; http://www.fepam.rs.gov.br/biblioteca/geo/bases_geo.asp). Para importar um arquivo *shape* para o R usamos a função `readShapeSpatial` e informamos o diretório dos arquivos. Use a função `getwd()` para ver o caminho completo:

```
>rs<-readShapeSpatial("C:/...../Curso-R/Dados/SHAPE/Limite_RS")
>plot(rs)

>grandes.lagos<-readShapeSpatial("C:/..../Curso-R/Dados/SHAPE/
+Grandes_Lagos")
>plot(grandes.lagos,add=T,col="blue")

>munic<-readShapeSpatial("C/..../Curso-R/Dados/SHAPE/
+municipios")
>plot(munic,add=T)
>plot(munic[munic$NOME=="PORTO ALEGRE",],add=T,col="red")
>plot(munic[munic$NOME=="RIO GRANDE",],add=T,col="green")
```

Vamos inserir o mapa do RS em um mapa do Brasil e alterar a aparência do mapa com funções já conhecidas:

```
>map("world","Brazil")
>plot(rs, add=T, col="lightgray")
>plot(grandes.lagos, add=T, col="blue")
>points(-51.22, -30, pch=20, col=2)
```

```
>text(-50,-10,"Brazil", cex=1.5)
>map.axes()
>map.scale(-45,-30, cex=0.6)
```

IMPORTANDO IMAGENS DO GOOGLE MAPS

```
>install.packages(c("ggmap","mapproj"))
>library(ggmap)
>library(mapproj)
```

Usamos a função `get_map` para importar imagens do Google Maps. Vamos ver o que é cada um dos argumentos da função:

- `location`: longitude e latitude central do mapa
- `maptype`: tipo de mapa: terreno (`"terrain"`), satélite (`"satellite"`), rodoviário (`"roadmap"`) ou híbrido (`"hybrid"`); algumas opções podem não estar disponíveis em todas as áreas
- `zoom`: zoom da imagem (números inteiros de 0 a 20); 0 = mapa mundi
- `color`: preto-e-branco `("bw")` ou colorido `("color")`
- `source`: a fonte da imagem

```
>mapa<-get_map(location="Brazil",maptype="terrain",zoom=4,
+color="color",source="google")
```

Para visualizar a imagem importada:

```
>ggmap(mapa)
```

Veja o mapa obtido usando o argumento `maptype=" satellite":`

```
>mapa1<-get_map(location="Brazil",maptype=" satellite",zoom=4,
+color="color",source="google")
>ggmap(mapa1)
```

Vamos inserir ao primeiro mapa o mesmo conjunto de dados já usado anteriormente representando os aeroportos do Brasil.

```
>ggmap(mapa) +
 geom_point(aes(x = lon, y = lat), data = aero.br,color="red")
```

O sinal **+** faz parte do comando.

Também é possível importar mapas a partir das coordenadas geográficas do local (coordenadas centrais):

```
>mang<-get_map(location=c(lon=-52.8,lat=-33.1),maptype="terrain",
+zoom=9,color="bw")
>ggmap(mang)
```

Vamos inserir os pontos de coleta. Importe o conjunto de dados 'pontos' que contém a latitude e longitude de 18 pontos na lagoa Mangueira:

```
>pontos<-read.table("pontos.txt",h=T)
```

```
>ggmap(mang) +
 geom_point(aes(x = long, y = lat), data = pontos,color="black")
```

8 PROGRAMAÇÃO

Além de ter muitas funções implementadas, uma das grandes utilidades to R é ser uma linguagem de programação, o que nos permite criar nossas próprias funções. Aprender a programar em R pode trazer muitas vantagens, pois, por exemplo, você não fica "preso" a análises existentes, podendo criar e adaptar análises específicas para os seus dados. Para criar uma função no R usamos:

```
nome.função<-function (argumento1,argumento2,...){corpo da função}
```

A parte antes de **<-** é o nome da função.

O primeiro comando, `function`, indica que estamos criando uma função. Em seguida, `(argumento1, argumento2,...)`, especifica os argumentos que serão avaliados na função. Entre chaves **{}** vem o corpo da função onde escrevemos a rotina que será usada para fazer os cálculos. A primeira chave indica o início da rotina e a última chave indica o fim.

Como um exemplo bem simples, vamos criar uma função para calcular a média de um conjunto de valores.

```
>minha.media<-function(dados){
+soma<-sum(dados)
+n<-length(dados)
+media<-soma/n
+return(media)
+}
```

Vamos testar nossa função e compará-la com a função `mean` implementada no R. Nossa função está correta?

```
>a<-c(1,2,3,4,5)
>minha.media(a)
>mean(a)
```

Em um exercício anterior criamos uma moeda e usando a função `sample` fizemos 100 amostragens para contar quantas vezes cai cara e quantas vezes cai coroa. Para relembrar vamos refazer os mesmos comandos:

```
>moeda<-c("cara","coroa")
>amostra<-sample(moeda,size=100,replace=T)
>table(amostra)
```

Vejam que definimos um valor fixo (100) para o tamanho da amostra. Se quisermos testar com 1000 valores, teremos que repetir os comandos e trocar 100 por 1000. Mas podemos também criar uma função para fazer isso. Na função abaixo temos dois argumentos:

- `objeto:` é o conjunto de dados, no caso, a moeda,
- `vezes:` vai especificar o número de amostras que queremos.

Na segunda linha da função, o comando `sample` será executado usando os dois argumentos que especificamos, ou seja, serão amostrados `vezes` valores do conjunto de dados `objeto`. Na terceira linha, os dados amostrados serão tabelados. Na quarta linha especificamos que queremos que a função retorne a tabela criada acima.

```
>jogo<-function(objeto,vezes){
+amostra<-sample(objeto,size=vezes,replace=T)
+tabela<-table(amostra)
+return(tabela)
+}
```

Agora vamos usar a função `jogo`:

```
>jogo(moeda,10)
>jogo(moeda,100)
>jogo(moeda,1000)
```

Podemos usar outro conjunto de dados. Suponha que ao invés de uma moeda você queira jogar dados:

```
>dado<-c(1,2,3,4,5,6)
>jogo(dado,4)
>jogo(dado,10000)
```

Um comando muito útil é o `for` que faz o R repetir uma tarefa tantas vezes quanto especificado. Ou seja, o comando `for` faz *loopings*, de forma que ao finalizar uma tarefa o R reiniciar essa mesma tarefa o número de vezes que foi especificado. Este comando é muito usado em diversas análises e para a simulação de dados. Para entendermos melhor vamos ver como esse comando funciona:

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

Para cada valor `i` serão executados os comandos que estão entre as chaves. Os valores de `i` são indicados por `i in 1:n` , indo de `i = 1` até `i = n`.

Na primeira rodada do `for`, o valor de `i` será 1, na segunda rodada `i` será igual a 2, na terceira rodada `i` será igual a 3, e assim por diante até que `i` seja igual a n.

```
>for(i in 1:5) {
+print(i)
```

```
+}
```

Muitas vezes pode ser necessário salvar os valores gerados pelo comando `for`. Para isso criamos um objeto vazio que receberá estes valores:

```
>tab.result<-numeric(0)
>tab.result
```

Agora vamos fazer um `for` semelhante ao anterior, mas salvando os valores no objeto `tab.result`:

```
>for(i in 1:5) {
+tab.result[i]<-i^2      #i elevado ao quadrado
+}
```

Para ver o resultado:

```
>tab.result
```

Vejamos outro exemplo do funcionamento do comando `for` (exemplo de Landeiro, 2012). Vamos fazer um gráfico "vazio" com limites xlim=0:10 e ylim=0:10:

```
>plot(0:10,0:10, type="n")
```

Usando o `for` vamos inserir um texto no gráfico a cada rodada do `for`. O `for` abaixo diz: quando `i=1` (ou seja, na primeira rodada), será inserido um texto dizendo 'Passo 1' nas coordenadas x=1 e y=1 do gráfico; quando `i=2`, será inserido um texto dizendo 'Passo 2' nas coordenadas x=2 e y=2, e assim por diante até `i=9`. Usamos a função Sys.sleep somente para conseguirmos visualizar os passos, pois esta função retarda os passos em 1 segundo.

```
>for(i in 1:9){
+text(i,i, paste("Passo", i))
```

```
+Sys.sleep(1)
+}
```

Insira `col=i` na segunda linha e veja o que acontece.
O valor de `i` não precisa começar em 1. Mude o `for` acima para um valor de `i` entre 2 e 6 e repita o comando.

Agora vamos criar uma função completa para calcular o índice de similaridade de Jaccard. Este índice é comumente usando em Ecologia para calcular a similaridade entre dois locais em relação, por exemplo, à composição de espécies. A equação deste índice é:
`a/a+b+c`,
onde a = número de espécies compartilhadas pelos locais 1 e 2, b = número de espécies que ocorrem somente no local 1, c = número de espécies que ocorrem somente no local 2.

```
>jaccard<-function(matriz){
+  if(any(matriz>1)) stop("Inserir matriz de presença-ausência")
+  n<-nrow(matriz)
+  result<-matrix(,n,n)
+  for(i in 1:n){
+    for(j in 1:n){
+      a<-sum(matriz[i,]==1 & matriz[j,]==1)
+      b<-sum(matriz[i,]==1 & matriz[j,]==0)
+      c<-sum(matriz[i,]==0 & matriz[j,]==1)
+      result[i,j]<-a/(a+b+c)
+    }
+  }
+  return(result)
+}
```

Vamos testar a função com o conjunto de dados ‘especies’:

```
>spp<-read.table("especies.txt")
```

```
>jaccard(spp)
```

Para conferir se nossa função está correta:

```
>library(vegan)
>1-vegdist(spp,"jaccard")
```

Agora crie uma função para calcular o índice de similaridade de Sorensen. Este índice é muito semelhante ao índice de Jaccard, porém ele dá uma importância maior às espécies compartilhadas (a) entre dois locais, como pode ser visto na equação:

`2a / 2a+b+c.`

```
>soren<-function(matriz){
+  if(any(matriz>1)) stop("Inserir matriz de presença-ausência")
+  n<-nrow(matriz)
+  result<-matrix(,n,n)
+  for(i in 1:n){
+    for(j in 1:n){
+      a<-sum(matriz[i,]==1 & matriz[j,]==1)
+      b<-sum(matriz[i,]==1 & matriz[j,]==0)
+      c<-sum(matriz[i,]==0 & matriz[j,]==1)
+      result[i,j]<-(2*a)/((2*a)+b+c)
+    }
+  }
+  return(result)
+}
```

Como você deve ter percebido, os comandos são todos idênticos aos comandos da função `jaccard`, com exceção do comando que calcula o índice: `result[i,j]<-(2*a)/((2*a)+b+c)`

Temos duas funções: `jaccard` e `soren`. Vamos criar uma função que nos permita escolher qual dos dois índices queremos calcular. Para isso usamos a função `switch`:

```
>indice<-function(matriz,metodo){
+   switch(metodo,
+          jac= indice<-jaccard(matriz),
+          sor= indice<-soren(matriz),
+          stop("Método não existente")
+          )
+return(indice)
+}
```

9 ANÁLISES ESTATÍSTICAS

Para quem tiver interesse em se aprofundar nos testes que serão apresentados aqui, sugiro os livros "Princípios de Estatística em Ecologia" de Nicholas Gotelli & Aaron Ellison (2011) e "Numerical Ecology" de Pierre Legendre & Louis Legendre (1998). Uma ótima referência sobre análises multivariadas no R é o livro "Numerical Ecology with R" de Daniel Borcard, François Gillet & Pierre Legendre (2011).

9.1 ANÁLISES UNIVARIADAS

Veremos alguns exemplos de como aplicar algumas das análises estatísticas mais comuns. A maioria das análises que veremos são especificadas usando a notação de fórmulas: **Y ~ X**, onde: **Y é a variável resposta**, **X é a variável explanatória** e ~ significa **"modelado por"**. A fórmula diz então que iremos analisar Y modelado por X ou Y em relação à X. Se tivermos mais de uma variável explanatória basta inserirmos ela no modelo: **Y ~ X1 + X2** e estaremos avaliando o efeito exclusivo de cada variável. Agora, se quisermos analisar também a interação entre as duas variáveis,

usamos o sinal de * ao invés do sinal de +: **Y ~ X1 * X2**, que diz: Y modelado por X1, X2 e pela interação X1X2.

Regressão linear simples

Vamos usar o conjunto de dados 'minhocas' e avaliar se a densidade de minhocas está relacionada ao pH do solo.

```
>dens<-minhocas[,"densidade"]
>pH<-minhocas[,"pH.solo"]
```

Acima criamos os objetos `dens` e `pH` que contém os dados de cada variável. Isto é uma opção ao uso de `minhocas$densidade` e `minhocas$pH` nas funções.

Para fazermos uma regressão linear simples usamos a função `lm` que significa *linear models*. Veremos que essa função pode ser usada para fazer outras análises, como teste-t e ANOVA, pois todas elas são modelos lineares.

```
>result<-lm(dens~pH)
```

Usamos a função `summary` para ver os resultados:

```
>summary(result)
```

Vamos fazer um gráfico e inserir a linha de tendência da regressão:

```
>plot(pH,dens)
>abline(result)
```

Uma questão importante é avaliarmos homogeneidade de variância e normalidade dos dados. Uma forma de fazer isso é analisar graficamente os resíduos dos dados:

```
>plot(result)
```

Regressão múltipla

Ao invés de avaliarmos somente o efeito do pH do solo sobre a densidade de minhocas queremos avaliar também o efeito da inclinação. Basta inserir a segunda variável explanatória no modelo:

```
>inclina<-minhocas[,"inclinacao"]

>res.mult<-lm(dens~pH+inclina)      #densidade modelada por pH e inclinação
>summary(res.mult)
```

É possível indicar no modelo que queremos avaliar a interação entre as variáveis explanatórias:

```
>res.mult.2<-lm(dens~pH*inclina)
>summary(res.mult.2)
```

Teste-t

Testes-t podem ser realizados usando as funções `lm` ou `t.test`. Importe o conjunto de dados 'algas.txt' que contém dados de um estudo hipotético que avaliou a riqueza de algas em ambientes com e sem eutrofização artificial (com 10 réplicas em cada nível).

```
>algas<-read.table("algas.txt",h=T)
>algas
> plot(algas$eutrof,algas$riqueza)
```

Para fazer o teste-t:

```
> t.test(algas$riqueza~algas$eutrof)
```

Repita a análise usando `lm`:

```
>teste.t<-lm(algas$riqueza~algas$eutrof)
>summary.aov(teste.t)  # apresenta os resultados na forma de uma tabela de ANOVA
```

Os resultados são iguais! Lembre-se que $F = t^2$.

Teste-t pareado

Suponha que o estudo descrito acima foi realizado em 10 rios da seguinte maneira: em cada rio as coletas foram realizadas em dois pontos, um não eutrofizado à montante e outro eutrofizado à jusante. Ou seja, o estudo foi realizado de forma pareada (ou em blocos), sendo que cada rio

representa um bloco. Devemos então analisar os dados usando um teste-t pareado. Basta adicionar o argumento `paired=T` na função `t.test`:

```
>t.test(algas$riqueza~algas$eutrof,paired=T)
```

Novamente, podemos analisar os dados usando `lm`. Veja que o bloco entra no modelo como uma segunda variável explanatória:

```
>teste.t<-lm(algas$riqueza~algas$eutrof+algas$bloco)
>summary.aov(teste.t)
```

Como comentado na seção sobre gráficos, uma forma de visualizar dados com duas variáveis explanatórias categóricas é com o uso da função `interaction.plot`:

```
>interaction.plot(algas$eutrof,algas$bloco,algas$riqueza)
```

Podemos ver que a tendência foi a mesma em todos os blocos, com maiores valores de riqueza em ambientes não eutrofizados, mesmo havendo variações na riqueza de espécies entre os blocos (ou seja, entre os rios).

Análise de variância (ANOVA)

A análise de variância é muito semelhante ao teste-t, porém, enquanto no teste-t a variável categórica tem dois níveis, na ANOVA a variável categórica tem mais de dois níveis (e/ou há mais de uma variável categórica).

Suponha que um experimento avaliou a riqueza de espécies de algas em relação à adição de nutrientes (três níveis: controle; 0,1 mg L^{-1} de fósforo e 0,5 mg L^{-1} de fósforo). Use o conjunto de dados 'algas_exp.txt' para fazer um gráfico e depois analisar os dados usando a função `aov`:

```
>experi<-read.table("algas_exp.txt",h=T)
>plot(experi$nutriente,experi$riqueza)
>res.anova<-aov(experi$riqueza~experi$nutriente)
>summary(res.anova)
```

O fator nutrientes foi significativo, mas não sabemos quais tratamentos diferiram entre si. Podemos usar o teste de Tukey:

```
>TukeyHSD(res.anova)
```

9.2 ANÁLISES MULTIVARIADAS

Quando aplicamos análises multivariadas comumente precisamos padronizar os dados, utilizando, por exemplo, padronização pelo total, máximo, média, logaritmo, etc. A função `decostand` do pacote *vegan* traz todas essas transformações, além de outras. Veja o help da função.

```
>?decostand
```

Também é comum realizarmos análises que utilizam matrizes de dissimilaridade (ou de distância). Os índices de dissimilaridade mais usados em Ecologia são a distância Euclidiana, Bray-Curtis, Sørensen e Jaccard. Estes índices de dissimilaridade são o complemento da similaridade, ou seja, se quisermos calcular a similaridade entre dois conjuntos de dados usamos 1 - dissimilaridade. A função `vegdist` no pacote vegan calcula estes e outros índices de dissimilaridade.

```
>?vegdist
```

Análise de variância multivariada (MANOVA)

MANOVAs podem ser feitas com várias funções disponíveis em diferentes pacotes. Uma destas funções é `adonis` (pacote *vegan*) que faz análises de variância multivariadas baseadas em testes de permutação utilizando matrizes de distância. A análise feita pela função `adonis` é conhecida como db-MANOVA (MANOVA baseada em distância) ou P-MANOVA (permutacional) e não requer que os dados tenham uma distribuição normal. Além disso, as variáveis explanatórias podem ser contínuas e/ou categóricas. Para detalhes sobre esta análise ver Anderson (2001).

Vamos usar os dados `dune` e `dune.env` disponíveis no pacote *vegan* para exemplificar o uso desta análise. Vamos analisar se a estrutura da comunidade de plantas de dunas (30 espécies) está relacionada ao tipo de manejo do solo (fator com 4 níveis).

```
>library(vegan)
>data(dune)
>data(dune.env)
>names(dune)
>names(dune.env)
>adonis(dune ~ Management, data=dune.env, method="bray",
+permutations=999)        #variável explanatória categórica

>adonis(dune ~ Management*A1, data=dune.env, method="bray",
+permutations=999)        #variáveis explanatórias categórica e contínua (A1)
```

Análise de agrupamento

Para fazer análises de agrupamento podemos usar a função `hclust`. Antes devemos gerar uma matriz de distância dos dados:

```
>dune.dist<-vegdist(dune,method="euclidean")  #matriz distância Euclidiana
>agrup<-hclust(dune.dist,method="average")
>plot(agrup)
```

O argumento `method="average"` é o método de agrupamento UPGMA. Veja o help da função `hclust` para descobrir os outros métodos de agrupamento disponíveis.

Ordenações

Análise de Componentes Principais (PCA)

Usaremos a função `prcomp` (mas existem várias outras disponíveis).

```
>data(varechem)          #do pacote vegan, dados de variáveis químicas do solo
>res.pca<-prcomp(varechem,scale=T)
```

Usamos `scale=T` para padronizar as variáveis para terem média = 0 e desvio padrão = 1. Assim, todas variáveis estarão na mesma escala de variação (e terão o mesmo peso na análise). Para ver os resultados:

```
>res.pca
```

```
>summary(res.pca)
```

Gráfico:

```
>biplot(res.pca)
```

Análise de Coordenadas Principais (PCoA)

É muito usada em Ecologia, pois permite o uso de qualquer medida de distância. A PCA, ao contrário, é empregada somente em dados quantitativos e quando queremos preservar as distâncias Euclidianas entre as observações. Porém, o uso de distâncias Euclidianas pode não fazer sentido em muitos casos. Por exemplo, comumente temos uma matriz binária de presença e ausência ou uma matriz de composição de espécies com abundância. Nestes casos, a PCoA é melhor opção que a PCA, pois na PCoA podemos usar outros índices, como os índices de dissimilaridade de Sørensen e Bray-Curtis. Vamos usar a função `capscale`:

```
>data(varespec)    #do pacote vegan, dados de composição de espécies
>vare.dist<-vegdist(varespec,"bray")
>res.pcoa<-capscale(vare.dist~1)
>summary(res.pcoa)
>plot(res.pcoa)
```

Suponha que o conjunto de dados represente duas estações do ano e você queira diferenciar os pontos do gráfico de acordo com as estações:

```
>plot(res.pcoa,type="n")
>points(res.pcoa,pch=19,col=rep(c("red","green"),each=12))
```

Veja que para fazer a PCoA nós usamos `~1`, o que indica que estamos modelando a matriz resposta por uma constante. Isso é necessário porque a função `capscale` também é usada para fazer RDA baseada em distância (db-RDA): basta acrescentar as variáveis explanatórias na fórmula depois do ~ (til). Ou seja, se depois do ~ acrescentarmos variáveis explanatórias, faremos uma RDA baseada em distância; se acrescentarmos somente a constante 1, faremos uma PCoA. Para detalhes sobre db-RDA ver Legendre & Anderson (1999).

Análise de Correspondência Canônica (CCA)

A CCA é usada para analisar a composição de espécies (matriz biótica) em relação a um conjunto de variáveis ambientais (matriz ambiental). Ou seja, a composição de espécies representa a variável resposta multivariada e as variáveis ambientais representam a variável explanatória multivariada. Para exemplificar uma CCA vamos usar os dados `varespec` (44 espécies) e `varechem` (14 variáveis químicas do solo).

```
>vare.cca<-cca(varespec, varechem)
>summary(vare.cca)
>anova(vare.cca)      # para avaliar a significância do modelo da CCA
>plot(vare.cca)
```

Vamos alterar a aparência do gráfico:

```
>plot(vare.cca, type="n")
>text(vare.cca, display="bp")
>points(vare.cca, pch=19, col="red",  cex=1.2)
>text(vare.cca, "species", col="blue", cex=0.8)
```

Pode-se plotar somente parte da informação, por exemplo, só as espécies:

```
>plot(vare.cca, display="species")
>?plot.cca     #para mais opções
```

Análise de Redundância (RDA)

O uso mais comum da RDA é para examinar as relações entre composição de espécies e características ambientais, assim como a CCA (ver Legendre & Legendre, 1998 para detalhes sobre as diferenças entre as análises). **A RDA é a extensão multivariada da regressão múltipla.** Assim, a RDA pode ser usada para qualquer regressão múltipla que envolva múltiplas variáveis resposta e múltiplas variáveis explanatórias. Usa-se a função `rda`:

```
>vare.rda<-rda(varespec, varechem)
>summary(vare.rda)
```

```
>anova(vare.rda)
>plot(vare.rda)
```

OUTRAS ANÁLISES DE DADOS ECOLÓGICOS

Explore o pacote *vegan* (http://cran.r-project.org/web/packages/vegan/vegan.pdf) para encontrar possibilidades de análises úteis em Ecologia. Você encontrará funções para calcular índices de diversidade, curvas de acumulação de espécies, índices de diversidade funcional e filogenética, índices de diversidade beta, partição aditiva de diversidade, partição de variância, teste de Mantel, escalonamento multidimensional não-métrico (NMDS), modelos nulos para dados biológicos, entre muitas outras.

REFERÊNCIAS BIBLIOGRÁFICAS

Anderson, M.J. 2001. A new method for non-parametric multivariate analysis of variance. **Austral Ecology** 26: 32- 46.

Borcard, D.; Gillet, F. & Legendre, P. 2011. **Numerical Ecology with R**. Springer, New York. 319.

Crawley, M.J. 2007. **The R Book**. John Wiley & Sons, Chichester. 951 p.

Gotelli, N.J. & Ellison, A.M. 2011. **Princípios de estatística em Ecologia**. Artmed, Porto Alegre.

Landeiro, V.L. 2012. **Introdução ao uso do programa R**. Disponível em: www.r-project.org.

Legendre, P. & Legendre, L. 1998. **Numerical Ecology**. Elsevier, Amsterdam.

Legendre, P. & Anderson, M. J. 1999. Distance-based redundancy analysis: testing multispecies responses in multifactorial ecological experiments. **Ecological Monographs** 69: 1-24.

Provete, D.B. 2011. **Estatística aplicada à ecologia usando o R**. Disponível em: www.r-project.org.

Venables, W.N., Smith, D. M. & the R Development Core Team. **An Introduction to R**. 2012. Disponível em http://www.r-project.org.