

Instituto Nacional de Pesquisas da Amazônia
Programa de Pós Graduação em Ecologia

Introdução ao uso do programa R

Victor Lemes Landeiro
Instituto Nacional de Pesquisas da Amazônia
Coordenação de Pesquisas em Ecologia
vllandeiro@gmail.com

The R Project for Statistical Computing



- O R é um software livre para computação estatística e construção de gráficos.
- Plataformas UNIX, Windows e MacOS.
- Para baixar o R vá ao Web site do R www.r-project.org
- Clique em CRAN
- Escolha o espelho de sua preferência (CRAN mirrors), existem 5 no Brasil
- Clique em *Windows 95 or later*
- Clique em *base* e salve o arquivo do R para Windows. Depois é só executar o arquivo.

24 de fevereiro de 2010



Conteúdo

Nota sobre o uso desta apostila:	3
A cara do R:	3
Noções gerais sobre o R.....	4
O <i>workspace</i> do R (área de trabalho).	4
Pacotes do R.....	5
Como usar um pacote do R.....	5
Como citar o R, ou um pacote do R em publicações	5
Usando o R.....	6
Demonstrações	6
O R como calculadora	6
Funções do R.....	6
Acessar o menu de ajuda do R (help)	7
Objetos do R (O que são?):	8
Como criar objetos.....	8
Objetos vetores com valores numéricos	8
Objetos vetores com caracteres (letras, variáveis categóricas).	8
Operações com vetores	9
Acessar valores dentro de um objeto [colchetes]	9
Transformar dados.....	9
Listar e remover objetos salvos	10
Como gerar valores.....	10
Gerar seqüências (usando : ou usando seq)	10
: (dois pontos)	10
seq.....	10
Gerar repetições (rep).....	10
rep	10
Gerar dados aleatórios.....	11
runif (Gerar dados com distribuição uniforme)	11
Tirar amostras aleatórias	11
sample.....	11
Ordenar e atribuir postos (<i>rank</i> s) aos dados	12
funções: sort, order e rank.....	12
sort	12
order.....	12
rank	12
Exercícios com operações básicas	13
Script do R.....	14
Usar o script do R para digitar os comandos	14
Exercícios com o script do R.....	15
Gráficos do R.....	15
PLOTS: gráficos para exploração de dados	15
Gráficos de barras	15
Gráficos de pizza	15
Gráfico de pontos (gráficos de dispersão)	15
Gráficos com variáveis numéricas.....	15
Alterando a aparência do gráfico.....	16
Adicionando linhas a um gráfico de pontos.....	16
Adicionar mais pontos ao gráfico.....	17
Gráficos com variáveis explanatórias que são categóricas.....	17
Inserir texto em gráficos	18



Dividir a janela dos gráficos	19
Salvar os gráficos.....	20
Resumo sobre gráficos.....	20
Exercícios com gráficos	20
Importar conjunto de dados para o R.....	22
Procurar os dados dentro do computador	22
Transformar vetores em matrizes e data frames	23
Acessar partes da tabela de dados (matrizes ou dataframes).....	23
Operações usando dataframes	24
Ordenar a tabela	24
Calcular a média de uma linha ou de uma coluna	24
Somar linhas e somar colunas	25
Medias das linhas e colunas.....	25
Exemplo com dados reais	25
As funções aggregate e by	28
Transpor uma tabela de dados	28
Exercícios com dataframes:	28
Comandos de lógica	28
Opções para manipular conjunto de dados.	28
which.....	29
ifelse	29
Exercícios com comando de lógica	Erro! Indicador não definido.
Criar Funções (programação).....	31
Sintaxe para escrever funções	31
Criando uma função (function)	31
Comando function	31
O comando for.....	32
Exercícios de criar funções:.....	37
Diferença entre criar uma função e escrever um código	37
Uso da função generico	39



Nota sobre o uso desta apostila:

O objetivo desta apostila é fazer uma breve introdução ao uso do programa R (R Development Core Team 2008). Seu intuito não é ensinar estatística.

O ideal para aprender a usar o R é "usá-lo!". Então, a melhor forma de se familiarizar com os comandos do R é ler um texto introdutório (como esta apostila) e ao mesmo tempo ir digitando os comandos no R e observando os resultados, gráficos, etc. Apenas ler esta apostila talvez não o ajude a fazer progressos no seu aprendizado do R, **acompanhe-a fazendo os cálculos no R**. Ler os manuais disponíveis na página do R como o "*An introduction to R*" que vem com o R e o "*Simple R*" de John Verzani [<http://www.math.csi.cuny.edu/Statistics/R/simpleR/printable/simpleR.pdf>], pode ser de grande ajuda no começo. Outro manual bem curto (49 páginas) e fácil de entender é o *The R Guide* de W. J. Owen disponível em <http://www.mathcs.richmond.edu/~wowen/TheRGuide.pdf>. Na página do R também existem 4 manuais em português, caso não goste de ler em inglês. Aprender a usar o R pode ser difícil e trabalhoso, mas lembre-se, o investimento será para você!

John Chambers (2008, pp. v) escreveu no prefácio de seu livro: "*Será que é proveitoso gastar tempo para desenvolver e estender habilidades em programação? Sim, porque o investimento pode "retribuir" em habilidade para fazer questões e na confiança que você terá nas respostas*". Veja no site do ecólogo Nicholas J. Gotelli alguns conselhos para ser um bom biólogo-ecólogo e para querer aprender a usar o R: <http://www.uvm.edu/~ngotelli/GradAdvice.html>.

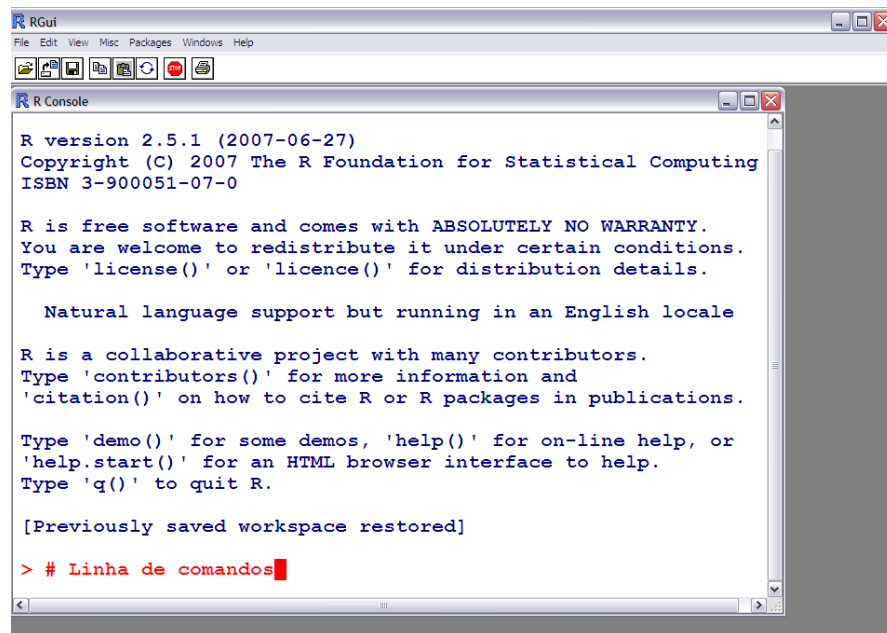
Nesta apostila as notas e explicações estão em letra arial, os comandos do R estão em letra Courier New. Os resultados dos comandos não aparecem na apostila, vocês devem conferir o resultado no R. Portanto, use os comandos em Courier New para ir acompanhando a apostila no R.

No R o sinal # (quadrado, jogo-da-velha) é usado para inserir comentários, é o mesmo que dizer: "a partir do # existem apenas comentários". O R não lê o que vem após o #. No decorrer desta apostila existem vários comentários após um sinal #, explicando o que foi feito.

A cara do R:

O R possui uma janela com algumas poucas opções para você se divertir clicando (figura abaixo). As análises feitas no R são digitadas diretamente na linha de comandos (i.e. você tem controle total sobre o que será feito). Na "linha de comandos" você irá digitar os comandos e funções que deseja usar. O sinal > (sinal de maior) indica o *prompt* e quer dizer que o R está pronto para receber comandos. Em alguns casos um sinal de + aparecerá no lugar do prompt, isso indica que ainda estão faltando comandos para terminar (ou que você errou alguma coisa na linha anterior). Se tiver errado pressione Esc para retornar ao prompt normal >. Note que na apostila, o começo de cada linha com os comandos do R possui um sinal do prompt, >, e em alguns casos um sinal de +, não digite estes sinais.

Os comandos que você digita aparecem em vermelho e o output do R aparece em azul. Após digitar os comandos tecla Enter para que eles sejam executados!



```
RGui
File Edit View Misc Packages Windows Help

R Console

R version 2.5.1 (2007-06-27)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> # Linha de comandos
```

Noções gerais sobre o R

Para usar o R é necessário conhecer e digitar comandos. Alguns usuários acostumados com outros programas notarão de início a falta de "menus" (opções para clicar). Na medida em que utilizam o programa, os usuários (ou boa parte deles) tendem a preferir o mecanismo de comandos, pois é mais flexível e com mais recursos. Algumas pessoas desenvolveram módulos de clique-clique para o R, porém eu não sei o nome nem como funcionam estes módulos. Eu acredito que ao usar um módulo de clique-clique perdemos uma das maiores potencialidades do R, que é a programação. Clicando você não aprende a linguagem R! Nós iremos começar a apostila apenas digitando comandos (i.e. "escrevendo códigos"), conforme aumente a familiaridade com a linguagem veremos, na parte final da apostila, como criar e escrever nossas próprias funções.

O R é *case-sensitive* (i.e. diferencia letras maiúsculas de minúsculas), portanto `A` é diferente de `a`. Use, com raríssimas exceções, sempre letras minúsculas. O separador de casas decimais é ponto `.`. A vírgula é usada para separar argumentos (informações). Não é recomendado o uso de acentos em palavras (qualquer nome que for salvar em um computador, não só no R, evite usar acentos. Acentos são comandos usados em programação e podem causar erros, por exemplo, em documentos do word e excel).

O *workspace* do R (área de trabalho).

A cada vez que você abre o R ele inicia uma "área de trabalho" (*workspace*). Neste *workspace* você fará suas análises, gráficos, etc. Ao final, tudo que foi feito durante uma sessão de uso do R pode ser salvo, salvando o *workspace* (área de trabalho). Sempre que for usar o R em um trabalho, antes de tudo, salve um *workspace* do R na pasta do trabalho em questão. Isso irá facilitar sua vida. No caso deste curso salve um *workspace* na pasta do curso (a pasta do curso será o seu diretório de trabalho). Ao salvar o *workspace*, na pasta irá aparecer um ícone do R, a partir do qual você irá abrir o R.

Para salvar o *workspace* abra o R vá em *"File"* e clique em *"Save workspace"* e salve-o na pasta desejada (diretório), não nomeie o arquivo. Feche o R e abra-o novamente clicando no ícone que apareceu na pasta escolhida, faça os exercícios e no final apenas feche o R, irá aparecer uma caixa perguntando se deseja salvar o *workspace*, diga que sim. **Não salve mais de um *workspace* na mesma pasta.**

Abra o R e salve um *workspace* na pasta que você usará para fazer este curso.

A partir daqui, sempre abra o R a partir do *workspace* salvo na pasta do seu trabalho. Para conferir o diretório de trabalho use:



```
> getwd()  
> [1] "C:/Documents and Settings/Victor Landeiro/My Documents"
```

Pacotes do R

O R é um programa leve (ocupa pouco espaço e memória) e geralmente roda rápido, até em computadores não muito bons. Isso porque ao instalarmos o R apenas as configurações mínimas para seu funcionamento básico são instaladas (o pacote *base*). Para realizar tarefas mais complicadas pode ser necessário instalar pacotes adicionais (*packages*). Um pacote bastante utilizado em ecologia é o *vegan*, vegetation analysis, criado e mantido por Jari Oksanen (Oksanen *et al.*, 2007). Para instalar um pacote vá ao site do R (www.r-project.org), clique em **CRAN**, escolha o espelho e clique em **packages**, uma lista com uma grande quantidade de pacotes está disponível. Clique no pacote que deseja e depois clique em **windows binary** e salve o arquivo. Abra o R e clique em *Packages*, depois em *Install Package(s) from local zip files* e selecione o arquivo do pacote que você baixou. Baixe e instale os pacotes **vegan** e o pacote **lattice**.

O R também pode conectar-se diretamente à internet. Desta forma é possível, instalar e atualizar pacotes sem que seja necessário acessar a página do R. Dependendo da sua conexão com a internet é necessário especificar as configurações de proxy da sua rede. No INPA é necessário digitar a seguinte linha de comandos para poder acessar a internet com o R.

```
> Sys.setenv("HTTP_PROXY"="http://proxy.inpa.gov.br:3128")
```

Para instalar um pacote direto do R digite, por exemplo:

```
> install.packages("vegan") ## É necessário estar conectado!
```

Durante o curso, se o seu computador não estiver ligado à internet solicite a um dos monitores que o auxilie a instalar o R e os pacotes necessários.

Como usar um pacote do R

Não basta apenas instalar um pacote. Para usar o pacote instalado é necessário "rodar" o pacote sempre que você abrir o R e quiser usar o pacote. Use a função `library` para rodar um pacote. Por exemplo: Digite `library(vegan)` na linha de comandos do R.

```
> library(vegan) # Após isso as funcionalidades do vegan estarão prontas para serem usadas. Sempre que abrir o R será necessário rodar o pacote novamente.
```

Como citar o R, ou um pacote do R em publicações

No R existe um comando que mostra como citar o R ou um de seus pacotes. Veja como fazer

```
> citation() # Mostra como citar o R
```

R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.

Veja que na parte com o nome dos autores aparece "R development core team", isso está correto, cite o R desta forma. Algumas pessoas não sabem disso e citam o R com autor Anônimo, isto tira o crédito do time.

Para citar um pacote, por exemplo o *vegan*, basta colocar o nome do pacote entre aspas.

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



Jari Oksanen, Roeland Kindt, Pierre Legendre, Bob O'Hara, Gavin L. Simpson, M. Henry H. Stevens and Helene Wagner (2008). *vegan: Community Ecology Package*. R package version 1.13-1. <http://vegan.r-forge.r-project.org/>

Usando o R

Demonstrações

Algumas funções do R possuem demonstrações de uso. Estas demonstrações podem ser vistas usando a função `demo()`. Vamos ver algumas demonstrações de gráficos que podem ser feitos no R. Digite o seguinte na linha de comandos:

```
> demo(graphics) # Vai aparecer uma mensagem pedindo que tecle Enter para
prosseguir, depois vá clicando na janela dos gráficos para ver as opções.
> demo(persp)
> demo(image)
```

O R como calculadora

O uso mais básico do R é usá-lo como calculadora. Os operadores são: `+` para soma, `-` subtração, `*` multiplicação, `/` divisão, `^` exponenciação. Tente o seguinte na linha de comandos do R:

```
> 2+2
> 2*2
> 2/2
> 2-2
> 2^2
```

Use parênteses para separar partes dos cálculos, por exemplo, para fazer a conta $4+16$ dividido por 4 elevado ao quadrado:

```
> ((4+16)/4)^2
```

Funções do R

O R tem diversas funções que podemos usar para fazer os cálculos desejados. O uso básico de uma função é `função(argumentos)`. `função` especifica a função que iremos usar e `argumentos` especifica os argumentos que serão avaliados.

Por exemplo: `sqrt` é a função para calcular raiz quadrada

```
> sqrt(9)           # Tira a raiz quadrada dos argumentos entre parênteses, no caso 9
> sqrt(3*3^2)       # raiz quadrada de 27
> sqrt((3*3)^2)     # raiz quadrada de 81
```

```
> prod é a função para multiplicação
> prod(2,2)         # O mesmo que 2x2
> prod(2,2,3,4)     # 2x2x3x4
```

`log` é a função para calcular o logaritmo

```
> log(3)            # log natural de 3
> log(3,10)         # log de 3 na base 10
```



```
> log10(3) # o mesmo que acima! log 3 na base 10
```

abs é a função para pegar os valores em módulo

```
> abs(3-9) # abs = modulo, |3-9|
```

Para fazer o fatorial de algum número use factorial()

```
> factorial(4) #4 fatorial (4!)
```

Acessar o menu de ajuda do R (help)

Para ver os arquivos de ajuda do R use `help(função)` ou `?função`. A função para fazer anova é `aov`. Então vamos ver o help:

```
> help(aov) # abre o help sobre ANOVA
```

ou simplesmente

```
> ?aov
```

O help do R para cada função geralmente possui 9 tópicos:

- 1 - **Description** - faz um resumo geral da função
- 2 - **Usage** - mostra como a função deve ser utilizada e quais argumentos podem ser especificados
- 3 - **Arguments** - explica o que é cada um dos argumentos
- 4 - **Details** - explica alguns detalhes que é preciso estar atento ao usar a função
- 5 - **Value** - mostra o que sai no output após usar a função (os resultados)
- 6- **Note** - notas sobre a função
- 7 - **Authors** - lista os autores da função (quem escreveu os códigos em R)
- 8 - **References** - referências para os métodos usados
- 9 - **See also** - mostra outras funções relacionadas que podem ser consultadas
- 10 - **Examples** - exemplos do uso da função. Copie e cole os exemplos no R para ver como funciona

Quando for usar uma função que nunca utilizou é no help que você aprenderá a usá-la. Os tópicos **Usage** e **Arguments** são os mais importantes, pois mostram como os argumentos devem ser inseridos na função (*Usage*) e caso não saiba o que é algum desses argumentos existe uma explicação para cada um deles (*Arguments*).

Muitas pessoas têm dificuldade em entender o help do R e dizem que o help é ruim. Pelo contrário, ele possui tudo que você precisa saber. Nada a mais nem nada a menos do que o necessário. É **fundamental** aprender a usar o help para um bom uso do R.

Em diversos casos nós não sabemos o nome da função que faz a análise que desejamos. Nestes casos é possível pesquisar usando palavras chave com a função `help.search()`.

```
> help.search("anova") # procura possíveis funções para fazer a anova
```

Nas versões mais recentes do R, R 2.8.0 em diante, a função `help.search` pode ser substituída por apenas `??`

```
> ??anova
```

Também é possível buscar ajuda na internet, no site do R, com a função `RSiteSearch()`

```
> RSiteSearch("analysis of variance") # abre uma página na internet, mas só funcionará se seu computador estiver conectado à internet.
```




Objetos do R (O que são?):

O que são os Objetos do R? Existem muitos tipos de objetos no R que só passamos a conhecê-los bem com o tempo. Por enquanto vamos aprender os tipos básicos de objetos. Para uma leitura mais aprofundada sobre os tipos de objetos e definições da linguagem R leia o manual "R language definition" disponível no menu "HELP", "Manuais em PDF", "R language definition".

a) **vetores**: uma sequência de valores numéricos ou de caracteres (letras, palavras).

b) **matrizes**: coleção de vetores em linhas e colunas, todos os vetores devem ser do mesmo tipo (numérico ou de caracteres).

c) **dataframe**: O mesmo que uma matriz, mas aceita vetores de tipos diferentes. Em um experimento com desenho para ANOVA em geral usamos dataframes, na qual a primeira coluna tem os valores da variável resposta (numérica) e uma segunda coluna possui os níveis do fator (um vetor de caracteres).

d) **listas**: conjunto de vetores. Não precisam ter o mesmo comprimento, é a forma que a maioria das funções retorna os resultados.

e) **funções**: as funções criadas para fazer diversos cálculos também são objetos do R.

No decorrer do curso veremos exemplos de cada um destes objetos. Para maiores detalhes veja o manual "An introduction to R" páginas 7 a 12. Este manual, e outros, já vem com o R, clique em *help* no menu do R e em *Manuais* (em PDF) para abrir o arquivo.

Como criar objetos

Objetos vetores com valores numéricos

Vamos criar um conjunto de dados que contém o número de espécies de aves (riqueza) coletadas em 10 locais. As riquezas são 22, 28, 37, 34, 13, 24, 39, 5, 33, 32.

```
> aves<-c(22, 28, 37, 34, 13, 24, 39, 5, 33, 32)
```

O comando `<-` (sinal de menor e sinal de menos) significa assinalar (*assign*). Indica que tudo que vem após este comando será salvo com o nome que vem antes. É o mesmo que dizer "salve os dados a seguir com o nome de **aves**".

A letra `c` significa concatenar (colocar junto). Entenda como "agrupe os dados entre parênteses dentro do objeto que será criado" no caso `aves`.

Para ver os valores (o conteúdo de um objeto), basta digitar o nome do objeto na linha de comandos.

```
> aves
```

A função `length` fornece o número de observações (n) dentro do objeto.

```
> length(aves)
```

Objetos vetores com caracteres (letras, variáveis categóricas).

Também podemos criar objetos que contêm letras ou palavras ao invés de números. Porém, as letras ou palavras devem vir entre aspas " ".

```
> letras<-c("a", "b", "c", "da", "edw")
```

```
> letras
```

```
> palavras<-c("Manaus", "Boa Vista", "Belém", "Brasília")
```

```
> palavras
```

Crie um objeto "misto", com letras e com números. Funciona? Esses números realmente são números? Note a presença de aspas, isso indica que os números foram **convertidos em caracteres**. Evite criar vetores "mistos", a menos que tenha certeza do que está fazendo.



Operações com vetores

Podemos fazer operações aritméticas usando o objeto `aves`, criado acima.

```
> max(aves)      #valor máximo contido no objeto aves
> min(aves)      #valor mínimo
> sum(aves)       #Soma dos valores de aves
> aves^2         #...
> aves/10
```

Agora vamos usar o que já sabemos para calcular a média dos dados das aves.

```
> n.aves<-length(aves)    # número de observações (n)
> media.aves<-sum(aves)/n.aves    #média
```

Para ver os resultados basta digitar o nome dos objetos que você salvou

```
> n.aves    # para ver o número de observações (n)
> media.aves    # para ver a média
```

Você não ficará surpreso em saber que o R já tem uma função pronta para calcular a média.

```
> mean(aves)
```

Acessar valores dentro de um objeto [colchetes]

Caso queira acessar apenas um valor do conjunto de dados use colchetes `[]`. Isto é possível porque o R salva os objetos como vetores, ou seja, a sequência na qual você incluiu os dados é preservada. Por exemplo, vamos acessar o quinto valor do objeto `aves`.

```
> aves[5]    # Qual o quinto valor de aves?
> palavras[3] # Qual a terceira palavra?
```

Para acessar mais de um valor use `c` para concatenar dentro dos colchetes `[c(1,3,...)]`:

```
> aves[c(5,8,10)]    # acessa o quinto, oitavo e décimo valores
```

Para excluir um valor, ex: o primeiro, use:

```
> aves[-1]          # note que o valor 22, o primeiro do objeto aves, foi excluído
```

Caso tenha digitado um valor errado e queira corrigir o valor, especifique a posição do valor e o novo valor. Por exemplo, o primeiro valor de `aves` é 22, caso estivesse errado, ex: deveria ser 100, basta alterarmos o valor da seguinte maneira.

```
> aves[1]<-100    # O primeiro valor de aves deve ser 100
> aves
> aves[1]<-22    # Vamos voltar ao valor antigo
```

Transformar dados

Em alguns casos é necessário, ou recomendado, que você transforme seus dados antes de fazer suas análises. Transformações comumente utilizadas são log e raiz quadrada.

```
> sqrt(aves)      #Raiz quadrada dos valores de aves
> log10(aves)     #log(aves) na base 10, apenas
```



```
> log(aves)      # logaritmo natural de aves
```

Para salvar os dados transformados dê um nome ao resultado. Por exemplo:

```
> aves.log<-log10(aves) # salva um objeto com os valores de aves em log
```

Listar e remover objetos salvos

Para listar os objetos que já foram salvos use `ls()` que significa listar.

```
> ls()
```

Para remover objetos use `rm()` para remover o que está entre parênteses.

```
> rm(aves.log)
```

```
> aves.log # você verá a mensagem:
```

```
Error: object "aves.log" not found
```

Como gerar valores

Gerar seqüências (usando : ou usando seq)

: (dois pontos)

Dois pontos ":" é usado para gerar seqüências de um em um, por exemplo a seqüência de 1 a 10:

```
> 1:10      # O comando : é usado para especificar seqüências.
```

```
> 5:16      # Aqui a seqüência vai de 5 a 16
```

seq

A função `seq` é usada para gerar seqüências especificando os intervalos.

Vamos criar uma seqüência de 1 a 10 pegando valores de 2 em 2.

```
> seq(1,10,2) #seq é a função para gerar seqüências, o default é em intervalos de 1.
```

A função `seq` funciona assim:

seq(from = 1, to = 10, by = 2), seqüência(de um, a dez, em intervalos de 2)

```
> seq(1,100,5) #seqüência de 1 a 100 em intervalos de 5
```

```
> seq(0.01,1,0.02)
```

Gerar repetições (rep)

rep

Vamos usar a função `rep` para repetir algo n vezes.

```
> rep(5,10)      # repete 5 dez vezes
```



A função `rep` funciona assim :

`rep(x, times=y)` # rep(repita x, y vezes) # onde x é o valor ou conjunto de valores que deve ser repetido, e times é o número de vezes)

```
> rep(2, 5)
> rep("a", 5)          # repete a letra "a" 5 vezes
> rep(1:4, 2)           # repete a seqüência de 1 a 4 duas vezes
> rep(1:4, each=2)      # note a diferença ao usar o comando each=2
> rep(c("A", "B"), 5)  # repete A e B cinco vezes.
> rep(c("A", "B"), each=5) # repete A e B cinco vezes.
> rep(c("Três", "Dois", "Sete", "Quatro"), c(3, 2, 7, 4)) # Veja que
```

neste caso a primeira parte do comando indica as palavras que devem ser repetidas e a segunda parte indica quantas vezes cada palavra deve ser repetida.

Gerar dados aleatórios

runif (Gerar dados aleatórios com distribuição uniforme)

```
> runif(n, min=0, max=1) # gera uma distribuição uniforme com n valores, começando em min e terminando em max.
```

```
> runif(200, 80, 100)    # 200 valores com mínimo de 80 e máximo 100
> temp<-runif(200, 80, 100)
> hist(temp)             # Faz um histograma de freqüências dos valores
```

rnorm (Gerar dados aleatórios com distribuição normal)

```
> rnorm(n, mean=0, sd=1) # gera n valores com distribuição uniforme, com média 0 e desvio padrão 1.
```

```
> rnorm(200, 0, 1)       # 200 valores com média 0 e desvio padrão 1
> temp2<-rnorm(200, 8, 10) # 200 valores com média 8 e desvio padrão 10
> hist(temp2)            # Faz um histograma de freqüências dos valores
```

Tirar amostras aleatórias

sample

O comando `sample` é utilizado para realizar amostras aleatórias e funciona assim:

`sample(x, size=1, replace = FALSE)` # onde x é conjunto de dados do qual as amostras serão retiradas, size é o número de amostras e replace é onde você indica se a amostra deve ser feita com reposição (TRUE) ou sem reposição (FALSE).

```
> sample(1:10, 5) # tira 5 amostras com valores entre 1 e 10
> sample(1:10, 15) # erro, amostra maior que o conjunto de valores, temos 10 valores (1 a 10) portanto não é possível retirar 15 valores sem repetir nenhum!
> sample(1:10, 15, replace=TRUE) # agora sim!
```

Vamos criar uma moeda e "jogá-la" para ver quantas caras e quantas coroas saem em 10 jogadas.



```
> moeda<-c("CARA", "COROA")
> sample(moeda, 10)
#ops! Esqueci de colocar replace=TRUE
> sample(moeda, 10, replace=TRUE) # agora sim
```

Ordenar e atribuir postos (*ranks*) aos dados

funções: sort, order e rank

Vamos criar um vetor desordenado para servir de exemplo:

```
> exemplo<-sample(1:100, 10) # amostra de 10 valores entre 1 e 100
> exemplo # veja que os valores não estão em ordem. Talvez com muita sorte os seus
valores estejam.
[1] 94 27 89 82 24 51 2 54 37 38 # seus valores serão diferentes
```

sort

A função `sort` coloca os valores de um objeto em ordem crescente ou em ordem decrescente.

```
> sort(exemplo) # para colocar em ordem crescente
[1] 2 24 27 37 38 51 54 82 89 94
> sort(exemplo, decreasing=TRUE) # para colocar em ordem decrescente
[1] 94 89 82 54 51 38 37 27 24 2
```

order

A função `order` retorna a posição original de cada valor do objeto "exemplo" caso "exemplo" seja colocado em ordem.

```
> order(exemplo) #
[1] 7 5 2 9 10 6 8 4 3 1
```

Note que o primeiro valor acima é 7, isso indica que se quisermos colocar o objeto "exemplo" em ordem crescente o primeiro valor deve ser o sétimo valor do "exemplo", que é o valor 2. Na sequência devemos colocar o quinto valor do objeto "exemplo", que é 24, depois o segundo, depois o nono... até que objeto "exemplo" fique em ordem crescente.

```
> order(exemplo, decreasing=TRUE)
[1] 1 3 4 8 6 10 9 2 5 7
```

É importante entender o comando `order`, ele é um comando importante para colocar uma planilha de dados seguindo a ordem de uma de suas variáveis. Veremos como fazer isso adiante.

rank

A função `rank` atribui postos aos valores de um objeto.

```
> exemplo ## apenas para lembrar os valores do exemplo
[1] 94 27 89 82 24 51 2 54 37 38
```

Agora vamos ver o rank destes valores

```
> rank(exemplo) # Para atribuir postos (ranks) aos valores do exemplo
```



[1] 10 3 9 8 2 6 1 7 4 5

Veja que 94 é o maior valor do exemplo, portanto recebe o maior rank, no caso 10.

Exercícios com operações básicas

1- Suponha que você marcou o tempo que leva para chegar a cada uma de suas parcelas no campo. Os tempos em minutos foram: 18, 14, 14, 15, 14, 34, 16, 17, 21, 26. Passe estes valores para o R, chame o objeto de *tempo*. Usando funções do R ache o tempo máximo, mínimo e o tempo médio que você levou para chegar em suas parcelas.

1.1- Ops, o valor 34 foi um erro, ele na verdade é 15. Sem digitar tudo novamente, e usando colchetes [], mude o valor e calcule novamente o tempo médio.

2- Você consegue prever o resultado dos comandos abaixo? Caso não consiga, execute os comandos e veja o resultado:

```
x<-c(1,3,5,7,9)
```

```
y<-c(2,3,5,7,11,13)
```

a) `x+1`

b) `y*2`

c) `length(x)` e `length(y)`

d) `x + y`

e) `y[3]`

f) `y[-3]`

3. Use as funções `union`, `intersect` e `setdiff` para encontrar a união, o interseção e as diferenças entre os conjuntos A e B. Aprenda no help como utilizar estas funções.

A 1 2 3 4 5

B 4 5 6 7

4. Calcule a velocidade média de um objeto que percorreu 150 km em 2.5 horas. Formula:

$vm = d/t$

5. Calcule $|2^3 - 3^2|$. Módulo de $2^3 - 3^2$

6. Suponha que você coletou 10 amostras em duas reservas, as 5 primeiras amostras foram na reserva A e as 5 últimas na reserva B. Crie um objeto chamado "locais" que especifica as reservas onde as amostras foram coletadas.

7. Você deseja jogar na mega-sena, mas não sabe que números jogar, use a função `sample` do R para escolher os seis números para você jogar. A mega-sena tem valores de 1 a 60.

8. Crie uma sequência de dados de 1 a 30 apenas com números ímpares. Use a função `seq()`.

9. Einstein disse que Deus não joga dados, mas o R joga!

a) Simule o resultado de 25 jogadas de um dado. Você precisará criar o objeto dado, e usar a função `sample()`.

10. Crie um objeto com estes dados: 9 0 10 13 15 17 18 17 22 11 15 e chame-o de **temp**. Calcule a raiz quadrada de **temp**, o log natural de **temp**, $\log(x+1)$ de **temp**, e eleve os valores de **temp** ao quadrado.



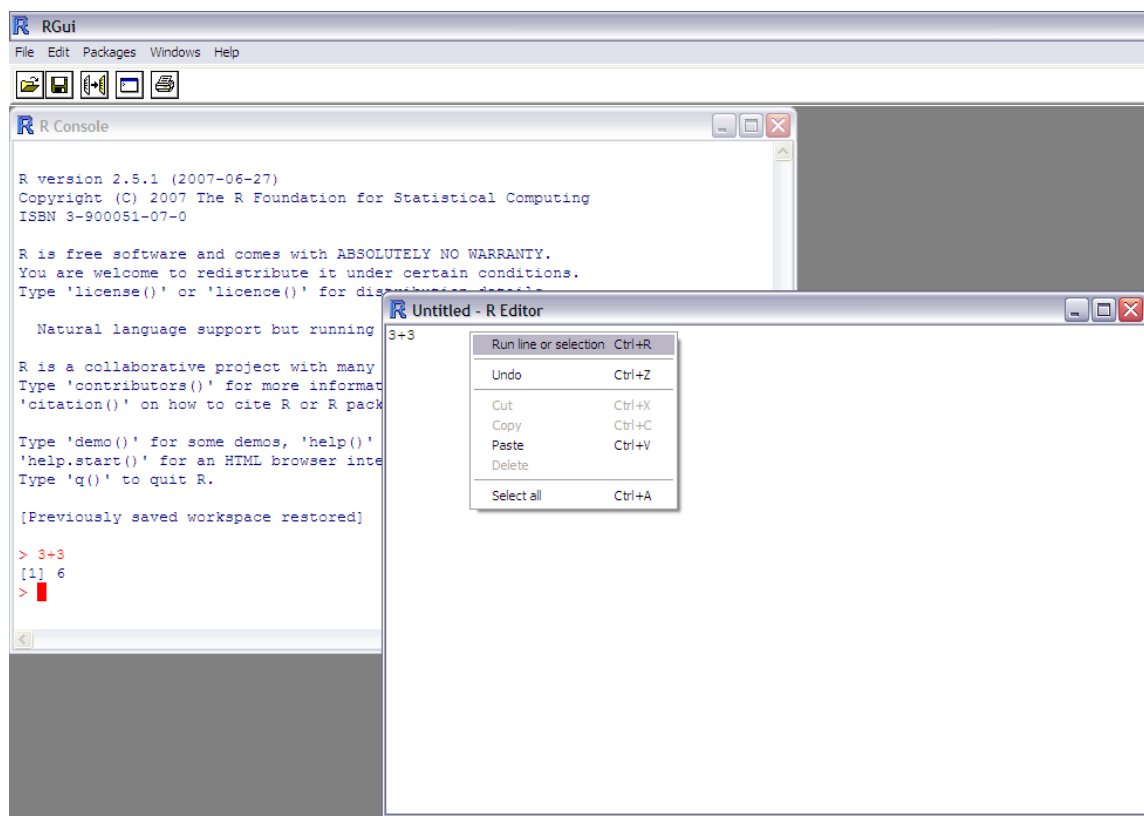
11. Crie um objeto chamado `info` que contem seu nome, idade, altura, peso, email e telefone.
12. Feche o R: Ao fechar aparecerá uma pergunta sobre salvar ou não o Workspace, diga que sim. Abra o R novamente e confira se seus objetos ainda estão salvos usando o comando `ls()`.

Script do R

Usar o script do R para digitar os comandos

Uma maneira que otimiza o uso do R e que poupa muito tempo é usar um script para digitar seus comandos. Neste caso, os comandos não são digitados na linha de comandos e sim em um editor de texto (R editor). Um script é um arquivo txt, onde você digita todas as análises e comandos. Com o script você facilmente faz alterações e correções, além de salvar o script e poder refazer rapidamente suas análises após algum tempo. Daqui em diante use o script para fazer os cálculos e exercícios da apostila. No curso de estatística (próxima semana) nós digitaremos os comandos no script e não diretamente na linha de comandos do R.

Para criar um script clique em *File* no menu do R e clique em *New script*. Uma janela será aberta (veja figura abaixo). O script aparece com o nome de R Editor (editor do R) e a extensão do arquivo é **.R**.



Digite `3+3` no script e aperte `Ctrl+R`. O `3*3` será enviado para a linha de comandos do R e o resultado aparecerá na tela. Para fazer outro comando aperte `Enter` e escreva outro comando na outra linha do script (cada comando deve ser digitado em uma linha diferente). Para enviar os comandos para o R também existe a opção de apontar com o mouse (ou selecionar) o que deseja enviar, clicar o botão direito do mouse e clicar em *Run line or selection*.

Daqui em diante sempre use o script do R. Com ele é fácil refazer análises ou alterar comandos! No script você também pode inserir observações sobre o que foi feito, usando `#` para indicar a presença de um comentário. Por exemplo:



```
> x<-sample(1:10,20,replace=TRUE) # Comentários: Aqui eu criei um objeto  
chamado x que é composto de 20 valores que foram amostrados ao acaso entre 1 e 10. O  
replace=TRUE indica que a amostragem foi feita com reposição.  
  
> mean(x) # Aqui eu uso a função mean para calcular a média de x.
```

Exercícios com o script do R

1- Faça alguns comandos que você já aprendeu usando um script. Por exemplo: crie objetos, seqüências, operações e insira comentários para lembrar o que foi feito. Ao terminar salve o script. Para salvar clique no "disquete" que aparece nos menus do R (veja figura acima), salve o script na pasta do curso, dê o nome que quiser usando a extensão **.R** para salvar (ex: meuScript.R) Feche o script, mas não feche o R. Para abrir o script novamente clique em *File* e em *Open script* selecione o script salvo para abri-lo.

Gráficos do R

PLOTS: gráficos para exploração de dados

Produzir gráficos de qualidade é uma ótima forma de apresentar e explorar dados. Os gráfico mais comuns são os gráficos de barras, pizza e de pontos (gráfico de dispersão).

Gráficos de barras

Para fazer gráficos de barras no R a função é `barplot`.

```
> barplot(sample(10:100,10))
```

Veja os exemplos de gráficos de barras:

```
> example(barplot)## clique na janela do gráfico para ir passando os exemplos.
```

Gráficos de pizza

Para fazer gráficos de pizza a função `pie`.

```
> pie(c(1,5,7,10))
```

Veja os exemplos de gráficos de pizza

```
> example(pie)
```

Gráfico de pontos (gráficos de dispersão)

Gráficos com variáveis numéricas

Primeiro vamos inserir os dados de duas variáveis numéricas. Lembre que a forma mais simples de inserir dados no R é usando a função de concatenar dados `"c"`.

```
> y<-c(110,120,90,70,50,80,40,40,50,30)
```

```
> x<-1:10
```

y geralmente é a letra usada em livros texto para indicar a variável resposta, a que aparece no eixo Y. Apesar de não ser uma norma, colocar a variável resposta no eixo y (vertical) dos gráficos é um consenso entre a maioria dos estatísticos, daí a letra y para dados resposta. x é chamada de variável independente e aparece no eixo x (horizontal).



É extremamente simples fazer um gráfico de pontos de y contra x no R. A função utilizada é `plot()` e precisa de apenas dois argumentos: o primeiro é o nome da variável do eixo X, o segundo é o da variável do eixo Y.

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

Alterando a aparência do gráfico

Para a proposta de apenas explorar os dados, o gráfico acima geralmente é o que você precisa. Mas em publicações é necessário saber melhorar a aparência do gráfico. Sempre é bom ter nomes informativos nos eixos (no R a opção default de nome das legendas é o próprio nome das variáveis). Suponha então que queremos mudar o nome da variável do eixo x para "Variável explicatória". Para isso, o argumento `xlab` ("*x label*") é utilizado. Use as setas do teclado para voltar ao comando anterior e coloque uma "vírgula" após o y e depois da vírgula coloque o comando da legenda do eixo x (`xlab="Variável explicatória"`)

```
> plot(x,y, xlab="Var explicatória")
```

Você pode alterar a legenda do eixo y da mesma forma, porém usando `ylab`. Use as setas e coloque o argumento da legenda do eixo y.

```
> plot(x,y,xlab="Var explicatória",ylab="Var resposta")
```

Também é fácil mudar os símbolos do gráfico, neste momento você está usando a opção default, que é a "bolinha vazia" (`pch=1`). Se você deseja que o símbolo seja um "x" use `pch=4`. Use a seta para cima para inserir o argumento `pch`.

```
> plot(x,y,xlab="Var explicatória",ylab="Var resposta" ,pch=4)
```

Use outros valores (de 1 a 25) para ver outros símbolos.

Para colocar título no gráfico uso o argumento `main`:

```
> plot(x,y,xlab="Var explicatória",ylab="Var resposta" ,pch=4,
main="Título do gráfico")
```

Adicionando linhas a um gráfico de pontos

A função utilizada para inserir linhas é `abline()`.

Vamos usar a função `abline` para inserir uma linha que mostra a média dos dados do eixo Y:

```
> abline(h=mean(y)) ## o h é de horizontal. Fará uma linha na horizontal
que passa pela média de y.
```

Para passar uma linha que passa pela média de x

```
> abline(v=mean(x)) ## o v é de vertical
```

Vamos passar uma linha que passa pelo sétimo valor do eixo x e mudar a cor da linha

```
> abline(v=7, col="red")# pode escrever o nome da cor ou números
(abaixo)
```

Também é possível inserir as duas linhas ao mesmo tempo:

```
> plot(x,y)
> abline(h=mean(y), v=mean(x), col=4)
```



Com cores diferentes

```
> abline(h=mean(y), v=mean(x), col=c(2, 4))
```

Adicionar mais pontos ao gráfico

Em alguns casos podemos querer inserir pontos de outro local no mesmo gráfico, usando símbolos diferentes para o novo local. Suponha que temos novos valores da variável resposta e da variável explanatória, coletados em outro local, e queremos adicionar estes valores no gráfico. Os valores são

```
v<- c(3, 4, 6, 8, 9) ## novos valores da variável explanatória
```

```
w<-c(80, 50, 60, 60, 70) ## novos valores da variável resposta
```

Para adicionar estes pontos ao gráfico, basta usar a função `points()`, e usar uma cor diferente para diferenciar.

Primeiro vamos refazer o gráfico com `x` e `y` e depois adicionar os novos pontos.

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

```
> points(v, w, col="blue")
```

Gráficos com variáveis explanatórias que são categóricas.

Variáveis categóricas são fatores com dois ou mais níveis (você verá isso no curso de estatística). Por exemplo, sexo é um fator com dois níveis (macho e fêmea). Podemos criar uma variável que indica o sexo como isto:

```
> sex<-c("macho", "fêmea")
```

A variável categórica é o fator sexo e os dois níveis são "macho" e "fêmea". Em princípio, os níveis do fator podem ser nomes ou números (1 para macho e 2 para fêmea). Use sempre nomes para facilitar.

Vamos supor que os 5 primeiros valores da nossa variável `y` eram machos e os 5 últimos eram fêmeas e criar a variável que informa isso.

```
> sexo<-c("Ma", "Ma", "Ma", "Ma", "Ma", "Fe", "Fe", "Fe", "Fe",  
+ "Fe") # Não digite o sinal de +
```

Para parecer um exemplo mais real vamos supor que `y` são valores de peso, para isso vamos apenas salvar um objeto chamado `peso` que é igual a `y`.

```
> peso<-y # peso é igual a y
```

```
> peso
```

Agora vamos fazer o gráfico:

```
> plot(sexo, peso)
```

Observe que o comando não funcionou, deu erro! Isso ocorreu porque não informamos que sexo é um fator. Vamos verificar o que o R acha que é a variável `sexo`.

```
> is(sexo)
```



Veja que o R trata a variável sexo como sendo um "vetor de caracteres". Mas nós sabemos que sexo é o nosso fator, então precisamos dar esta informação ao R. A função `factor`, transforma o vetor de caracteres em fator.

```
> factor(sexo)
```

Veja que o R mostra os "valores" e depois mostra os níveis do fator.

Agora podemos fazer o nosso gráfico adequadamente:

```
> plot(factor(sexo), peso)
```

Você também pode salvar a variável sexo já como um fator.

```
> sexo.f<-factor(sexo)
```

Gráficos do tipo boxplot são bons quando o número de observações (de dados) é muito grande. Neste caso, um gráfico com pontos seria melhor, pois podemos ver quantas observações foram utilizadas para produzir o gráfico.

Para fazer um gráfico de pontos quando uma variável é categórica precisamos usar a função `stripchart`.

```
> stripchart(peso~sexo) # faz o gráfico, mas na horizontal
```

```
> stripchart(peso~sexo, vertical=TRUE) # agora o gráfico está na vertical, porém os pontos aparecem nas extremidades. TRUE pode ser abreviado para apenas T.
```

```
> stripchart(peso~sexo, vertical=T, at=c(1.3, 1.7)) # agora os pontos estão centralizados, pois com o argumento at, nós especificamos a localização dos pontos no eixo X.
```

Note que agora só há um problema. Eram cinco fêmeas e no gráfico aparecem apenas 4. Isso ocorreu porque duas fêmeas tinham o mesmo peso. Para melhorar o gráfico é necessário usar o argumento `method="stack"`, para que os pontos não fiquem sobrepostos.

```
> stripchart(peso~sexo, vertical=T, at=c(1.5, 1.7),  
method="stack") # os pontos não estão mais totalmente sobrepostos, mas um símbolo ainda está sobre o outro. Usando o argumento offset conseguimos separá-los.
```

```
> stripchart(peso~sexo, vertical=T, at=c(1.3, 1.7), method=  
+ "stack", offset=1) ## Não digite o sinal de +
```

Inserir texto em gráficos

Para inserir textos em gráficos a função é `text`

Ela funciona assim:

```
text(cooX, cooY, "texto")
```

Neste caso `cooX` e `cooY` indicam as coordenadas do eixo X e do eixo Y onde o texto deverá ser inserido. Vamos criar um gráfico simples:

```
> plot(1:10, 1:10)
```

Agora vamos adicionar "seu nome" no gráfico nas coordenadas 6 e 7. Isso indica que seu nome ficará na posição 6 do eixo X e na posição 7 do eixo Y.



```
> text(6, 7, "Seu.Nome")
```

Para inserir duas palavras:

```
> text(c(2, 3), c(8, 6), c("nome", "sobrenome"))
```

Ou seja, a primeira palavra ficará na posição 2-8 e a segunda palavra ficará na posição 3-6.

Também é possível adicionar texto aos gráficos usando a função `locator(n)`. O `n` indica o número de pontos que você deseja indicar no gráfico. A função `locator` permite que você clique no gráfico com o *mouse* e adicione o texto na posição desejada.

```
> plot(1:10, 1:10)
```

```
> text(locator(1), "Texto") # clique com o mouse na posição desejada no gráfico
```

Para mais de um texto:

```
> plot(1:10, 1:10)
```

```
> text(locator(3), c("texto 1", "texto 2", "texto 3"))
```

Dividir a janela dos gráficos

Em alguns casos em que se deseja comparar dois ou mais gráficos é possível dividir a janela de gráficos do R. A função `par` controla diversas características dos gráficos. Sempre que quiser melhorar algum aspecto de seu gráfico consulte o help da função `par` (`?par`) e descubra o argumento necessário.

Para dividir a janela de gráficos o comando é:

`par(mfrow=c(nl, nc))` # `nl` indica o número de linhas e `nc` o número de colunas que a janela deverá ter. Primeiro vamos dividir a janela em duas colunas.

```
> par(mfrow=c(1, 2)) # uma linha e duas colunas
```

```
> plot(1)
```

```
> plot(2)
```

Agora vamos dividir a janela em duas linhas e uma coluna:

```
> par(mfrow=c(2, 1))
```

```
> plot(1)
```

```
> plot(2)
```

Agora vamos dividir a janela em duas linhas e duas colunas (para plotar 4 gráficos)

```
> par(mfrow=c(2, 2))
```

```
> plot(1)
```

```
> plot(2)
```

```
> plot(3)
```

```
> plot(4)
```



Salvar os gráficos

Existem diversas opções para salvar os gráficos do R. A mais simples é clicar com o botão direito do *mouse* sobre o gráfico e depois em "*save as metafile*". Para que usa o editor de texto LATEX e sabe o que é um arquivo postscript também existe esta opção com o "*save as postScript*".

Também é possível salvar os gráficos em PDF, JPEG, BMP, PNG ou outros formatos.

Para salvar em pdf ou outros formatos clique sobre o gráfico e depois clique em *Arquivo* e em *salvar como*. Escolha a opção que deseja salvar.

Resumo sobre gráficos

Plot: `plot(xaxis, yaxis)` faz um gráfico de pontos se *x* é uma variável contínua e um boxplot se *x* é uma variável categórica (se é um fator).

Lines: `lines(x, y)` traça uma linha reta de acordo com as coordenadas fornecidas. É possível mudar o tipo e a cor das linhas usando os argumentos `lty` para o tipo e `col` para a cor.

Points: `points(x, y)` adiciona mais pontos em um gráfico. É possível colocar os novos pontos com símbolo diferente, usando o argumento `pch=2` ou `pch="H"`.

stripchart: é uma função que faz gráficos de pontos, com a qual é possível separar pontos coincidentes.

Cores: Gráficos em preto e branco são bons na maioria dos casos, mas cores podem ser mudadas usando `col="red"` (escrevendo o nome da cor) ou `col=2` (usando números).

O comando abaixo mostra os números que especificam algumas cores

```
> pie(rep(1, 30), col=rainbow(30))
```

par: a função `par` é utilizada para alterar diversos aspectos dos gráficos, dentre eles, dividir a janela para que caiba mais de um gráfico.

Exercícios com gráficos

1- Um biólogo foi ao campo e contou o número de sapos em 20 locais. Ele também anotou a umidade e a temperatura em cada local. Faça dois gráficos de pontos para mostrar a relação do número de sapos com as variáveis temperatura e umidade. Use a função `par()` para dividir a janela em duas.

Os dados são:

sapos	6-5-10-11-26-16-17-37-18-21-22-15-24-25-29-31-32-13-39-40
umid	62-24-21-30-34-36-41-48-56-74-57-46-58-61-68-76-79-33-85-86
temp	31-23-28-30-15-16-24-27-18-10-17-13-25-22-34-12-29-35-26-19

2- Um biólogo interessado em saber se o número de aves está relacionado ao número de uma determinada espécie de árvore realizou amostras em 10 locais. Os valores obtidos foram:



```
aves<-c(22,28,37,34,13,24,39,5,33,32)
arvores<-c(25,26,40,30,10,20,35,8,35,28)
```

Faça um gráfico que mostra a relação entre o número de aves e o número de árvores.

Um colega coletou mais dados sobre aves e árvores, em outra área, que podemos aproveitar. Os dados são: `arvores2` (6,17,18,11,6,15,20,16,12,15),.

```
arvores2<-c(6,17,18,11,6,15,20,16,12,15)
aves2<-c(7,15,12,14,4,14,16,60,13,16)
```

Inclua estes novos pontos no gráfico com um símbolo diferente e cor azul.

Junte o seu arquivo de aves com o arquivo de aves do seu amigo, para que fique em um arquivo completo: `aves.c<-c(aves,aves2)`. Faça o mesmo para árvores.

3- Os dados do exercício anterior foram coletados em regiões diferentes (você coletou no local A e seu amigo no local B). Inclua esta informação no R, use a função `rep` se quiser facilitar.

Faça um gráfico para ver qual região tem mais aves e outro para ver qual tem mais árvores. Lembre que local deve ser um fator para fazer o gráfico. Use função `stripchart`.

```
stripchart(aves.c~locais, vertical=TRUE, pch=c(16,17))
```

Existem locais com o mesmo número de aves, e no gráfico estes pontos apareceram sobrepostos. Faça o gráfico sem pontos sobrepostos:

4 - Existem milhares de outros comandos para alterar a aparência de gráficos, veja esta página do help (`?par`) para ver alguns comandos sobre como alterar a aparência de gráficos. Não se preocupe se você ficar confuso sobre as opções do help(par), apenas praticando você irá começar a dominar estes comandos.



Importar conjunto de dados para o R

Passe a tabela abaixo (locais, abundância de macacos e de árvores frutificando) para o Excel e salve-a (na pasta do curso) em formato de "texto separado por tabulações" (No Excel clique em "salvar como" e em texto (separado por tabulações)). Salve com o nome de `amostras.txt`. **Deixe a célula onde está escrito *Amostra* em branco.**

Amostra	reserva	macacos	frutas
A1	A	22	25
A2	A	28	26
A3	A	37	40
A4	A	34	30
A5	A	13	10
A6	A	24	20
A7	A	39	35
A8	A	5	8
A9	A	33	35
A10	A	32	28
B1	B	7	6
B2	B	15	17
B3	B	12	18
B4	B	14	11
B5	B	4	6
B6	B	14	15
B7	B	16	20
B8	B	60	16
B9	B	13	12
B10	B	16	15

Note que esta tabela contém variáveis numéricas e categóricas, portanto este é um exemplo de objeto do tipo *dataframe*. Para importar a tabelas para o R a função é `read.table`.

```
> read.table("amostras.txt", header=TRUE)
```

O argumento `header=TRUE` informa que os dados possuem cabeçalho, ou seja, a primeira linha contém os nomes dos atributos (variáveis) e a primeira coluna possui o nome das amostras.

Não se esqueça de salvar os dados, não basta importar, é preciso salvar também.

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

Para ver os dados digite o nome do arquivo.

```
> macac
```

O objeto `macac` é um objeto do tipo *dataframe*. Isso quer dizer que `macac` é um objeto que possui linhas e colunas (observações nas linhas e variáveis (atributos) nas colunas).

Procurar os dados dentro do computador

Caso você não lembre o nome do arquivo de dados que deseja importar existe a opção de procurar os dados no computador com a função `file.choose()`:



```
> macac<-read.table(file.choose(),header=T) # encontre o arquivo  
macac.txt
```

Você também pode conferir se um determinado arquivo de dados existe no seu diretório:

```
> file.exists("macac.txt")
```

Transformar vetores em matrizes e data frames

Além de importar tabelas, existe opções juntar vetores em um arquivo dataframe ou matriz. Para criar uma matriz use `cbind` (column bind) ou `rbind` (row bind). Vamos ver como funciona. Vamos criar três vetores e depois juntá-los em uma matriz.

```
> aa<-c(1,3,5,7,9)  
> bb<-c(5,6,3,8,9)  
> cc<-c("a","a","b","a","b")  
> cbind(aa,bb) # junta os vetores em colunas  
> rbind(aa,bb) # junta os vetores em linhas
```

Lembre que matrizes podem conter apenas valores numéricos ou de caracteres. Por isso, se juntarmos o vetor `cc`, nossa matriz será transformada em valores de caracteres.

```
> cbind(aa,bb,cc) # junta os vetores em colunas, mas transforma números em  
caracteres.
```

Para criar uma dataframe com valores numéricos e de caracteres use a função `data.frame`:

```
> data.frame(aa,bb,cc)  
> data.frame(aa,bb,cc) # agora temos números e letras.
```

Acessar partes da tabela de dados (matrizes ou dataframes)

Agora vamos aprender a selecionar (extrair) apenas partes do nosso conjunto de dados `macac` usando `[]` colchetes. O uso de colchetes funciona assim: `[linhas, colunas]`, onde está escrito `linhas` você especifica as linhas desejadas, na maioria dos casos cada linha indica uma unidade amostral. Onde está escrito `colunas`, você pode especificar as colunas (atributos) que deseja selecionar. Veja abaixo:

```
> macac [,1] # extrai a primeira coluna e todas as linhas  
> macac [,2] # extrai a segunda coluna e todas as linhas  
> macac [1,] # extrai a primeira linha e todas as colunas  
> macac [3,3] # extrai a terceira linha e a terceira coluna, 1 valor  
> macac [1,3] # extrai o valor da primeira linha e terceira coluna  
> macac [c(1:5),c(2,3)] # extrai somente as linhas 1 a 5 e as colunas 2 e 3
```

Existem outras duas maneiras de extrair dados de uma dataframe. Uma é usando a função `attach()`, que torna as variáveis acessíveis apenas digitando o nome delas na linha de comandos



Digite macacos na linha de comandos e veja o que acontece!

```
> macacos  
Error: object "macacos" not found
```

Agora use a função `attach`, digite macacos novamente e veja o que acontece.

```
> attach(macac)  
> macacos # agora os dados macacos está disponível  
> frutas # para ver o número de arvores frutificando  
> reserva # para ver as reservas  
> plot(macacos, frutas)
```

IMPORTANTE: A função `attach` apesar de parecer "uma mão na roda" pode ser problemática. Se "atachamos" duas dataframes, e elas tiverem variáveis com o mesmo nome, é "perigoso" usarmos por engano a variável errada. Se você usar a função `attach` é seguro "desatachar" o objeto imediatamente após o seu uso. Para isso use `detach()`.

```
> detach(macac)  
macacos # macacos não está mais disponível
```

Uma melhor forma de acessar colunas pelo nome é usar o comando cifrão `$`, e não usar a função `attach()`. O uso é basicamente o seguinte `dados$variável` (dados especifica o conjunto de dados e `variável` a variável que deseja extrair). Por exemplo, para extrair os dados de macacos de use:

```
> macac$macacos
```

Ou então usar colchetes e o nome da variável:

```
> macac[, "macacos"]
```

Vamos fazer o gráfico de macacos X frutas usando `$`.

```
> plot(macac$frutas, macac$macacos)
```

Faça o gráfico de frutas X macacos usando colchetes ao invés de usar `$`.

Operações usando dataframes

Ordenar a tabela

Os dados de macac estão dispostos na ordem em que foram coletados. Em alguns casos podemos querer colocá-los em outra ordem. Por exemplo, na ordem crescente de quantidade de árvores frutificando. Para isso use a função `order`.

```
> macac[order(macac$frutas), ] # ordena os dados macac pela ordem de frutas  
# para colocar em ordem decrescente use o argumento decreasing=TRUE
```

Calcular a média de uma linha ou de uma coluna

Podemos calcular a média de cada coluna da dataframe usando.

```
> mean(macac[, "macacos"]) # média de macacos ou use  
> mean(macac$macacos)  
> mean(macac) # média de cada coluna
```

Repare que resultado para reservas foi NA e em seguida apareceu uma mensagem de aviso. NA indica *Non Available*, pois não é possível calcular a média de variáveis categóricas.



Acima nós calculamos a média de macacos, mas sem considerar a reserva onde as amostras foram coletadas. O que fazer para calcular a média de macacos em cada reserva? Basta selecionar as linhas correspondentes a cada reserva.

```
> mean(macac[1:10,2]) # média de macacos na reserva A
> mean(macac[11:20,2]) # média de macacos na reserva B
```

Para facilitar, principalmente quando tivermos muitas categorias, o melhor é usar a função `tapply()`. Ela funciona assim: `tapply(dados, grupos, função)`. Será calculada uma "função" nos "dados" em relação aos "grupos". Então, vamos calcular a média (função) dos macacos (dados) em cada reserva (grupos).

```
> tapply(macac$macacos, macac$reserva, mean)
```

Veja que agora calculamos a média de macacos na reserva A e a média na reserva B.

Somar linhas e somar colunas

Agora, vamos usar dados sobre a abundância de moluscos que foram coletados em dez parcelas. A tabela também contém informações sobre quantidade de chuva em cada parcela e em qual de duas reservas (A ou B) a parcela estava localizada. O arquivo com os dados é `moluscos.txt` e está na pasta da disciplina, importe-o para o R. Chame o arquivo de `mol` para facilitar.

Somar os valores de colunas ou linhas usando as funções `colSums` para somar colunas e `rowSums` para somar linhas.

```
> colSums(mol[,1:6])          #Note que estamos somando apenas as informações
sobre as espécies (colunas 1 a 6)
> rowSums(mol[,1:6])
```

Qual informação obteve ao usar os dois comandos acima?

Medias das linhas e colunas

Calcular a média das colunas e linhas. Abundância média por parcela e abundância média por espécie.

```
> colMeans(mol[,1:6])
> rowMeans(mol[,1:6])
```

E se quisermos calcular apenas a abundância média de moluscos na reserva A (linhas de 1 a 5)? Você consegue fazer este cálculo? Veja as quatro opções abaixo e diga qual delas fará o cálculo correto.

- 1 - `mean(mol[,1:6])`
- 2 - `rowMeans(mol[1:5,1:6])`
- 3 - `colMeans(mol[1:5,1:6])`
- 4 - `mean(rowSums(mol[1:5,1:6]))`

Exemplo com dados reais

Na pasta do curso existe um arquivo chamado `simu.txt`. Este arquivo contém amostras de Simuliidae (borrachudos - piuns) coletadas em 50 riachos da Chapada Diamantina - Bahia. Importe este arquivo e vamos ver alguns exemplos de opções que podemos fazer com este conjunto de dados.

```
> simu<-read.table("simu.txt",header=T)
```



Use `names()` para ver o nome das variáveis que estão no arquivo.

```
> names(simu)
```

Note que as 6 primeiras variáveis são ambientais e depois os dados de 20 espécies. Estes dados foram coletados em três municípios dentro da chapada diamantina (Lençóis, Mucugê e Rio de Contas).

Primeiro vamos separar os dados das espécies dos dados ambientais:

```
> ambi<-simu[,1:6]
```

```
> spp<-simu[,7:26]
```

Vamos ver os gráficos que mostram as relações entre as variáveis ambientais.

```
> plot(ambi[, "temperatura"], ambi[, "altitude"])
```

```
> plot(ambi[, "altitude"], ambi[, "pH"])
```

```
> plot(ambi[, "condutividade"], ambi[, "pH"])
```

No R a função `pairs()` faz um gráfico com todas essas comparações entre as variáveis ambientais.

```
> pairs(ambi)
```

Vamos calcular a abundância total de borrachudos em cada riacho.

```
> rowSums(spp) # soma das linhas (riachos)
```

Para salvar a abundância:

```
> abund<-rowSums(spp)
```

E para calcularmos a riqueza de espécies em cada riacho? Primeiro precisamos transformar os dados em presença e ausência (0 para ausência e 1 para presença). Primeiro vamos criar uma cópia do arquivo original.

```
> copia<-spp # cópia é igual a spp
```

Agora vamos criar o arquivo de presença e ausência:

```
> copia[copia>0]<-1 ## quando copia for maior que 0 o valor será substituído por
```

1

```
> pres.aus<-copia ## apenas para mudar o nome do arquivo
```

```
> pres.aus # veja que agora os dados estão apenas como 0 e 1
```

Para encontrar a riqueza de espécies basta somar as linhas do arquivo de presença e ausência.

```
> riqueza<-rowSums(pres.aus) # número de espécies por riacho (riqueza)
```



Para calcular a riqueza média:

```
> riq.media<-mean(rowSums(pres.aus))
```

Vamos calcular a riqueza média por município onde foram realizadas coletas.

```
> riq.muni<-tapply(riqueza, ambi[, "município"], mean)
```

Agora use a função `colSums()` para ver em quantos riachos cada espécie ocorreu e qual a abundância total de cada espécie. Qual a espécie que ocorreu em mais riachos? Qual era a mais abundante?

Para transformar os dados de abundância de espécies em log:

```
> simu.log<-log(spp)
```

Veja o resultado em log:

```
> simu.log      # Note a presença do valor -inf
```

O valor -inf ocorre porque não é possível calcular o log de 0. Veja:

```
> log(0)
```

Por isso é comum você ver em trabalhos os dados transformados em $\log(x+1)$:

```
> spp.log<-log(spp+1)
```

```
> spp.log
```

Agora vamos retornar ao nosso arquivo completo, `simu`, e ordenar a tabela de acordo com a altitude, de forma que o primeiro riacho seja o que está localizado em menor altitude.

```
> simu[order(simu[, "altitude"]), ] # tabela ordenada pela altitude
```

Vamos entender esse comando passo a passo:

Primeiro descobrimos a ordem que os dados devem estar para que a tabela fique em ordem:

```
> order(simu[, "altitude"]) # veja que o primeiro valor é o 18, que é o riacho que ocorre em menor altitude.
```

Então, para colocar a tabela inteira em ordem de altitude, nós usamos o resultado de `order(simu[, "altitude"])`.

```
> simu[order(simu[, "altitude"]), ] # assim os dados são ordenados pela altitude.
```

Agora vamos fazer 4 gráficos da riqueza de espécies em relação a altitude, pH, temperatura e condutividade. Primeiro vamos dividir a janela de gráficos em 4 partes.

```
> par(mfrow=c(2, 2))
```



Riqueza X altitude

```
> plot(simu[, "altitude"], rowSums(pres.aus))
```

Faça os outros três gráficos.

Agora vamos fazer um gráfico para ver a riqueza de espécies nas três áreas (Lençóis, Mucugê e Rio de Contas).

```
> stripchart(riqueza~factor(simu[, "município"]))
```

Coloque o gráfico na vertical e separe os pontos coincidentes.

As funções `aggregate` e `by`

As funções `by` e `aggregate` podem ser utilizadas para aplicar uma função em todas as colunas de uma tabela, enquanto a função `tapply` faz isso apenas uma coluna por vez. As duas funções retornam os mesmos resultados, mas de formas diferentes. Para calcular a média de cada espécie de borrachudo em cada um dos 3 locais use:

```
> aggregate(spp, list(simu[, 1]), mean)
```

A função `aggregate` retorna uma `dataframe` com os valores para cada local.

```
> by(spp, simu[, 1], mean)
```

A função `by` retorna uma lista com os valores para cada local. Para acessar os valores de uma lista é preciso usar dois colchetes:

```
> med.local<-by(spp, simu[, 1], mean)
```

```
> med.local
```

```
> med.local[["Lençóis"]]
```

```
> med.local[["Mucugê"]]
```

```
> med.local[["R.Contas"]]
```

Transpor uma tabela de dados

Em alguns casos é necessário transpor uma tabela de dados, ou seja, colocar as informações das linhas nas colunas e as colunas nas linhas. A função é `t()` e indica *transpose*.

```
> t(spp)
```

Comandos de lógica

Opções para manipular conjunto de dados.

Primeiro vamos ver o significado dos comandos abaixo.

> Maior que >= maior que ou igual a



```
<  Menor que          <=  menor que ou igual a
==  igualdade
!=  diferença

> x<-c(1,2,9,4,5)
> y<-c(1,2,6,7,8)
> x>y    # Retorna TRUE para os maiores e FALSE para os menores
> x>=y
> x<y
> x==y   # Retorna TRUE para os x que são iguais a y
> x!=y   # Retorna TRUE para os x que são diferentes de y
```

Agora vamos selecionar partes dos dados que obedecem a algum critério de seleção.

which

A função `which` funciona como se fosse a pergunta: Quais?

```
> a<-c(2,4,6,8,10,12,14,16,18,20)
> a>10    # Retorna um vetor contendo TRUE se for maior e FALSE se for menor
> which(a>10) # Equivale a pergunta: "Quais valores de a são maiores que 10?". Note
que a resposta é a posição dos valores (o sexto, o sétimo...) e não os valores que são maiores que 10.
```

```
> a[6]                # selecionamos o sexto valor de a

> a[c(6:10)]          # selecionamos do sexto ao décimo valor
```

Tente prever o que ocorrerá usando o comando abaixo.

```
> a[which(a>=14)]
```

Acertou? Selecionamos os valores de `a` que são maiores ou igual a que 14!

Também podemos usar a função `which` para selecionar partes de uma tabela de dados. Por exemplo, vamos selecionar apenas as parcelas dos dados de moluscos onde a chuva é maior que 1650 mm.

Lembre-se que para selecionarmos partes de uma tabela podemos usar colchetes `[linhas,colunas]` e especificar as linhas e colunas que desejamos usar. Então vamos usar o comando `which` para escolher apenas as linhas (parcelas) onde a chuva é maior que 1650mm.

```
> mol[which(mol$chuva>1650),]
```

Podemos escolher também apenas a parte da tabela que corresponde às amostras da reserva A.

```
> mol[which(mol$reserva=="A"),]
```

Também podemos incluir um segundo critério de escolha usando `&` (que significa "e"). Vamos escolher apenas as parcelas da reserva B e que tenham o valor de chuva maior que 1650mm.

```
> mol[which(mol$reserva=="B" & mol$chuva>1650),]
```

ifelse

Agora vamos aprender a usar o comando `ifelse` que significa: se for isso, então seja aquilo, se não, seja aquilo outro. O comando funciona da seguinte maneira: `ifelse(aplicamos um teste, especificamos o valor caso o teste for verdade, e o valor caso falso)`. Complicado? Vamos ver alguns exemplos para facilitar as coisas.



Primeiro crie no R um objeto com o valor do salário de dez pessoas. Os valores são:

```
> salarios<-c(1000, 400, 1200, 3500, 380, 3000, 855, 700,  
+ 1500, 500) ## Não digite o sinal de +
```

As pessoas que ganham menos de 1000 ganham pouco, concorda? Então aplicamos o teste e pedimos para retornar "pouco" para quem ganha menos de 1000 e "muito" para quem ganha mais de 1000.

```
> ifelse(salarios<1000, "pouco", "muito") # Se o salário é menor que 1000,  
seja pouco, se for maior seja muito.
```

Também podemos usar o comando `ifelse` para transformar os dados em presença e ausência. Vamos usar os dados das espécies de borrachudos (spp) da Chapada Diamantina.

```
> ifelse(spp>=1, 1, 0) # se o valor de spp for maior ou igual a 1 seja 1, se não, seja 0
```

Exercícios com dataframes e comandos de lógica:

- 1-Calcule a média de macacos e frutas dentro de cada reserva do conjunto de dados macac.
2. Quais informações podem ser obtidas da tabela de moluscos quando usamos os quatro comandos abaixo.

```
1 - mean(mol[,1:6])  
2 - rowMeans(mol[1:5,1:6])  
3 - colMeans(mol[1:5,1:6])  
4 - mean(rowSums(mol[1:5,1:6]))
```
3. Use a função `t` para transpor os dados de moluscos (apenas as espécies).
4. Multiplique o valor da abundância de cada espécie de molusco pelo valor de chuva da parcela correspondente. Por exemplo: Na parcela 1 a chuva foi de 1800 mm e nesta parcela ocorreram 10 indivíduos da espécie 1, portanto o novo valor para a sp1 na parcela 1 será de 1800 x 10 = 18000.
5. Faça um gráfico de pontos para comparar o número de macacos na reserva A com o número de macacos na reserva B. Use o conjunto de dados `macac`. Use a função `stripchart`.
6. Importe para o R o arquivo `minhocas.txt` que está na pasta do curso. Este arquivo possui dados sobre a densidade de minhocas em 20 fazendas. As variáveis medidas são: área, inclinação do terreno, tipo de vegetação, pH do solo, se o terreno é alagado ou não e a densidade de minhocas.
 - 6.1. Veja a tabela na forma original e depois a ordene de acordo com o tamanho da área.
 - 6.2. Faça um gráfico para ver a relação entre minhocas e área do terreno e outro gráfico para ver a relação entre minhocas e tipo de vegetação.
 - 6.3. Selecione a parte da tabela que contém apenas dados de locais alagados.
 - 6.4. Calcule a densidade média de minhocas, e a densidade média em locais alagados e em locais não alagados.
 - 6.5. Qual a área média das fazendas?



7. Podemos usar a função `ifelse` para transformar dados de abundância em dados de presença e ausência. Transforme os dados de abundância de moluscos (`mol`) em dados de presença e ausência.

8. Com os dados transformados em presença e ausência, use a função `rowSums` para ver quantas espécies de moluscos ocorrem em cada parcela. Use `colSums` para ver em quantas parcelas cada espécie estava presente.

Criar Funções (programação)

Enquanto o R é muito útil como uma ferramenta para análise de dados, muitos usuários às vezes precisam criar suas próprias funções. Esta é uma das maiores vantagens do R. Além de ser um programa para análises estatísticas, o R é acima de tudo uma linguagem de programação, com a qual podemos programar nossas próprias funções.

Sintaxe para escrever funções

A sintaxe básica é:

```
function(lista de argumentos){corpo da função}
```

A primeira parte da sintaxe (`function`) é a hora em que você diz para o R, "estou criando uma função".

A lista de argumentos é uma lista, separada por vírgulas, que apresenta os argumentos que serão avaliados pela sua função.

O corpo da função é a parte em que você escreve o "algoritmo" que será utilizado para fazer os cálculos que deseja. Esta parte vem entre chaves.

Ao criar uma função você vai querer salva-la, para isso basta dar um nome

```
minha.função<-function(lista de argumentos){corpo da função}
```

Criando uma função (function)

Comando function

Vamos ver como criar funções começando por uma função simples, que apenas simula a jogada de moedas (cara ou coroa). Neste caso a função terá dois argumentos (`x` e `n`). `x` será a "moeda" `c("cara", "coroa")` e `n` será o número de vezes que deseja jogar a moeda.

Vamos dar o nome a esta função de `jogar.moeda`

```
> jogar.moeda<-function(x,n) { ## Não digite tudo na mesma linha  
+ sample(x,n, replace=T)  
+ } # Fim da função
```

A primeira chave "{" indica o início do algoritmo da função e a outra indica o fim "}".

O comando `sample(x,n)` indica que desejamos amostrar `x`, `n` vezes (jogar a moeda `n` vezes).

Agora vamos usar nossa função, mas primeiro vamos criar a "moeda".



```
> moeda<-c("Cara", "Coroa")
> jogar.moeda(moeda, 2)
> jogar.moeda(moeda, 10)
> jogar.moeda(moeda, 1000)
```

Veja que jogando 1000 moedas ficou difícil saber quantas caras e quantas coroas saíram. Com a função `table()`, podemos descobrir isso facilmente.

```
> table(jogar.moeda(moeda, 1000))
```

Veja que também podemos usar a função `jogar.moedas` para jogar dados:

```
> dado<-1:6
> jogar.moeda(dado, 2)
> table(jogar.moeda(dado, 200))
```

Esta função que criamos para jogar moedas é muito simples e nem é necessária, pois podemos jogar moedas sem ela.

```
> sample(c("cara", "coroa"), 10, replace=T)
```

Agora suponha que você não sabe qual função do R calcula a média, mas você sabe a fórmula que calcula a média.

$$m\acute{e}dia = \frac{\sum Y_i}{n}$$

Conhecendo a fórmula você pode criar sua própria função que calcula a média. Note que você pode inserir comentários dentro da função para depois lembrar o que fez:

```
> media<-function(dados) {                                #função chamada media
+   n<-length(dados)                                       ## n é o número de observações
+   med<-sum(dados)/n                                     ## calcula a média
+   return(med)                                           # return que retorna os resultados calculados
+ } # Fim da função
```

Vamos usar a função criada (`media`) para calcular a média dos valores abaixo:

```
> valores<-c(21, 23, 25, 19, 10, 1, 30, 20, 14, 13)
> média(valores)
```

Use a função do R `mean()` para verificar se o cálculo foi feito corretamente.

O comando `for`

O comando `for` é usado para fazer *loopings*, e funciona da seguinte maneira:

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

Isso quer dizer que: para cada valor i o R vai calcular os comandos que estão entre as chaves `{comandos}`. O `"i in 1:n"` indica que os valores de i serão $i = 1$ até $i = n$. Ou seja, na primeira rodada do `for` o i será igual a 1, na segunda $i = 2$, e assim por diante até $i = n$. Para salvar os resultados que serão calculados no `for` precisamos criar um objeto vazio que receberá os valores calculados. Criamos este objeto vazio assim:

```
> resu<-numeric(0)
```



Agora vamos usar o `for` para elevar `i` valores ao quadrado:

```
> for(i in 1:5) {                                     ## o i será i = 1; i = 2; i = 3; i = 4; i = 5
+   resu[i] <- i^2                                     ## Na primeira rodada o i = 1, então será 1^2
+ } # Fim do for (i)
> resu                                                ## Para ver os resultados
```

Antes de continuar, vamos fazer uma espécie de "filminho" mostrando o que o `for` faz:

Primeiro vamos fazer um gráfico com limites `xlim=0:10` e `ylim=0:10`.

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

Agora vamos usar o `for` para inserir textos no gráfico a cada passo do `for`:

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

O R fez tudo muito rápido, de forma que não conseguimos ver os passos que ele deu. Vamos fazer novamente, mas agora inserindo uma função que retarde o R em 1 segundo. Ou seja, cada passo irá demorar 1 segundo.

```
> plot(0:10, 0:10, type="n")
> for(i in 1:9) {
+   text(i, i, paste("Passo", i))
+   Sys.sleep(1)                                     ## retarda os passos em 1 segundo
+ }
```

Entendeu o que está sendo feito? No primeiro passo do `for`, o `i` é igual a 1, portanto apareceu o texto "passo 1" na coordenada `x=1, y=1` do gráfico. No segundo passo o `i` é igual a 2 e aparece o texto "passo 2" na coordenada `x=2, y=2`, e assim por diante.

O `for` é um comando bastante utilizado em diversas funções e na simulação de dados. Fique atento ao uso do "`for`" nas funções seguintes e pergunte caso não tenha entendido.

A sequência de Fibonacci é uma sequência famosa na matemática (Braun & Murdoch 2007). Os dois primeiros números da sequência são [1, 1]. Os números subsequentes são compostos pela soma dos dois números anteriores. Assim, o terceiro número da sequência de Fibonacci é $1+1=2$, o quarto é $1+2=3$, e assim por diante. Vamos usar a função `for` para descobrir os 12 primeiros números da sequência de Fibonacci (Este exemplo foi retirado e adaptado de: Braun & Murdoch 2007 pp, 48).

```
> Fibonacci <- numeric(0)
> Fibonacci[c(1, 2)] <- 1 # o 1º e 2º valores da sequência devem ser = 1
> for (i in 3:12) { # 3 a 12 porque já temos os dois primeiros números [1,1]
+   Fibonacci[i] <- Fibonacci[i-2] + Fibonacci[i-1]
+ }
```



```
> Fibonacci
```

Exercícios: Modifique o código para gerar a sequência de Fibonacci de forma que:

- a) mude os dois primeiros elementos da sequência para 2 e 2
- b) mude os dois primeiros elementos da sequência para 3 e 2
- c) modifique o código para que os valores sejam compostos pela diferença entre os dois valores anteriores.
- d) modifique o código para que os valores sejam compostos pela diferença entre os dois valores imediatamente anteriores **somada** ao terceiro valor imediatamente anterior. Faça inicialmente com que a sequência Fibonacci comece com 3 valores [1,1,1].

Agora vamos usar o `for` para fazer uma função mais complexa, que calcula o índice de diversidade de Shannon. A fórmula do índice de Shannon é:

$$\text{Shannon } H = - \sum P_i \ln P_i$$

onde P_i é o valor i em proporção e \ln é o logaritmo natural de i .

```
> shannon<-function(dados) {      #Criamos a função shannon
+ prop<-dados/sum(dados)          ## calcula as proporções
+ resu<-numeric()                # objeto numérico "vazio" que receberá os valores do for
+ n<-length(prop)
+ for(i in 1:n){
+   resu[i]<-if(prop[i]>0){prop[i]*log(prop[i])}
+   else{0} # veja abaixo sobre o uso do if e do else
+ } # Fim do for (i)
+ H<- -sum(resu)
+ return(H)
+ } # Fim da função
```

Nota: Durante o cálculo do índice de Shannon precisamos pular do calculo quando o valor de proporção é zero, porque $\log(0)$ não existe ($-\infty$). Por isso usamos o `if` e o `else`, ou seja, se (`if`) o valor for maior que zero faça o calculo, se não (`else`) retorne 0.

```
> log(0)      # veja o resultado do log de zero.
```

Agora vamos usar a função `shannon` para calcular o índice de Shannon dos dados apresentados no livro da Anne Magurran página 45, tabela 2.1 (Diversidad Ecológica y su Medición, 1988). Os dados são:

```
> magur<-c(235,218,192,0,20,11,11,8,7,4,3,2,2,1,1)# O valor 0
foi alterado propositalmente, no livro o valor é 87.
```

```
> shannon(magur)
```

Esta função que criamos para calcular o índice de diversidade de Shannon calcula o índice de apenas um local por vez. Agora vamos criar uma função que calcula o índice de



Shannon de vários locais ao mesmo tempo. Neste caso teremos que usar o comando `for` duas vezes.

```
> shan<-function(dados) { # nome de shan para diferenciar da função acima
+   nl<-nrow(dados)      # número de locais (linhas)
+   nc<-ncol(dados)
+   H<-numeric()         # variável vazia que receberá os valores H de cada local
+   for(i in 1:nl){
+     prop<-dados[i,]/sum(dados[i,]) ## dados do local i em proporção
+     resu<-numeric()
+     for(k in 1:nc){
+       resu[k]<-if(prop[1,k]>0){prop[1,k]*log(prop[1,k])}
+       else{0}
+     } # Fim do for (k)
+     H[i]<--sum(resu)
+   } # Fim do for (i)
+   return(H)
+ } # Fim da função
```

Vamos calcular o índice de Shannon dos dados de borrachudos (spp).

```
> shan(spp)
```

Na ecologia um dos índices de similaridade (resemblance measures) utilizado é o índice de Jaccard:

$$S(X1,X2) = \frac{a}{a + b + c} \quad \text{equação 2.10 (Legendre e Legendre 1998, pp 256)}$$

Onde, no caso de uma matriz de espécies: a = número de espécies compartilhadas pelos locais 1 e 2; b = número de espécies presentes apenas no local 1; c = número de espécies presentes apenas no local 2. Vamos criar uma função para calcular o índice de similaridade de Jaccard:

```
> jaccard<-function(dados.spp) {
+   # Vamos inserir uma mensagem de erro, caso dados não sejam de presença-ausência
+   if(any(dados.spp>1))
+     stop("Erro: é preciso fornecer uma tabela com dados de
+     presença e ausência")
+   n<-nrow(dados.spp) # número de locais
+   jac<-matrix(NA,n,n) ## Matriz que receberá os valores de similaridade
+   colnames(jac)<-rownames(dados.spp) ## Dar os nomes dos locais
```



```
+ rownames(jac)<-rownames(dados.spp) ## Dar os nomes dos locais
+   for(i in 1:n){
+     for(j in 1:n){
+       # número de espécies presentes em ambos locais, i e j ( a )
+       a<-sum(dados.spp[i,]==1 & dados.spp[j,]==1)
+       # número de espécies presentes apenas no local i ( b )
+       b<-sum(dados.spp[i,]==1 & dados.spp[j,]==0)
+       # número de espécies presentes apenas no local j ( c )
+       c<-sum(dados.spp[j,]==1 & dados.spp[i,]==0)
+       jac[i,j]<- a / (a+b+c)    ## Fórmula de Jaccard
+     }
+   }
+ return(as.dist(jac))
+ }
```



```
> jac<-jaccard(pres.aus)    ## os cálculos demoram um pouco
> jac
> jac.vegan<-1-vegdist(pres.aus,"jaccard") ## Cálculo do Jaccard
no vegan. OBS: O R não calcula similaridades e sim distâncias (dissimilaridades), por isso usamos o
1-
> jac.vegan
> cbind(jac,jac.vegan) # coloque os valores lado a lado e compare. Está
correto?
```

Outro índice de similaridade muito utilizado em ecologia, mas para dados de abundância é o índice de Bray-Curtis:

$$D(X1,X2) = \frac{\sum |y_{1j} - y_{2j}|}{\sum (y_{1j} + y_{2j})} \text{ Equação 7.57 (Legendre \& Legendre 1998, pp 287)}$$

Onde: y_{1j} é a abundância da espécie j no local 1 e y_{2j} é a abundância da espécie j no local 2. Esta fórmula calcula a dissimilaridade e não similaridade (diferente do exemplo anterior do Jaccard). Então vamos criar uma função para calcular o índice de similaridade de Bray-Curtis.

```
> bray<-function(dados.spp){
+   n<-nrow(dados.spp)
+   BrayCurtis<-matrix(NA,n,n)
+   for(i in 1:n){
+     for(j in 1:n){
+       numerador<-sum(abs(dados.spp[i,]-dados.spp[j,]))
+       denominador<-sum(dados.spp[i,]+dados.spp[j,])
+       BrayCurtis[i,j]<- numerador/denominador
+     }
+   }
+ }
```



```

+     }
+     }
+ return(as.dist(BrayCurtis))
+ }

```

Vamos usar nossa função `bray` usando os dados de borrachudos e comparar com o resultado obtido usando a função `vegdist` do pacote `vegan`.

```

> bra<-bray(spp)
> bra.vegan<-vegdist(spp,"bray")
> cbind(bra,bra.vegan)
> bra==bra.vegan

```

Exercícios de criar funções:

1 - Crie uma função para calcular o índice de diversidade de Simpson dos dados de borrachudos da Chapada Diamantina (`spp`):

$$D = 1 - \sum p_i^2 \quad \text{onde } p_i \text{ é o valor } i \text{ em proporção}$$

Confira o resultado usando a função `diversity` do pacote `vegan`:

```

> library(vegan)
> diversity(spp,"simpson",MARGIN=1) # O argumento MARGIN indica
se você quer fazer os cálculos por linhas ou por colunas. Use 1 para linhas e 2 para colunas.

```

2 - Crie uma função para calcular a distância (dissimilaridade) de Hellinger dos dados de abundância de borrachudos. A fórmula é:

$$D(X1,X2) = \sqrt{\sum \left[\sqrt{\frac{y_{1j}}{y_{1+}}} - \sqrt{\frac{y_{2j}}{y_{2+}}} \right]^2} \quad \text{Equação 7.55 (Legendre & Legendre 1998, pp 286)}$$

Onde: y_{1j} são os valores de abundância de cada espécie j no local 1; y_{2j} são os valores de abundância de cada espécie j no local 2; y_{1+} é a soma total das abundâncias do local 1; y_{2+} é a soma total das abundâncias do local 2.

Não se assuste com a fórmula, tente dividi-la em partes dentro da função.

Para calcular a distância de Hellinger no R e conferir com resultado da sua função use:

```

> hel.vegan<-vegdist(decostand(spp,"hellinger"), "euclid")

```

3 – Desafio: Calcule o índice de Shannon usando apenas 3 linhas de comando e o índice de Simpson usando apenas uma linha de comandos.

Diferença entre criar uma função e escrever um código

Existem milhares de formas de se trabalhar em R. Algumas pessoas preferem escrever pedaços de códigos e executá-lo quantas vezes seja necessário, enquanto outras preferem criar funções mais automatizadas, mas ambas produzindo o mesmo resultado. Em alguns casos vocês



irão se deparar com códigos e em outros casos com funções. É preciso saber diferenciar os dois tipos, pois seu uso é levemente diferente e pode gerar confusões.

Vamos gerar um código que escolhe números para a mega sena e depois vamos criar uma função que faz a mesma coisa.

```
> njogos<-20 # quantos jogos queremos produzir
> numeros<-matrix(NA, 6, njogos) # arquivo que irá receber números dos jogos
> for(i in 1:njogos){
+   numeros[,i]<-sample(1:60, 6)
+ }
> numeros
```

Agora, caso você queira mudar o número de jogos (njogos), é preciso mudar o valor e rodar o código todo novamente.

Crie agora 50 jogos.

Vamos transformar nosso código em uma função e ver o que muda, apesar dos comandos serem quase os mesmos.

```
> megasena<-function(njogos){ # cria a função com nome de megasena
>   numeros<-matrix(NA, 6, njogos) # cria o arquivo que recebe os jogos
>   for(i in 1:njogos){
>     numeros[,i]<-sample(1:60, 6)
>   }
>   return(numeros)
> }
> megasena(100)
```

O interessante em criar uma função, é que após ela estar funcionando você precisa mudar apenas um argumento, que nesse caso é o njogos.

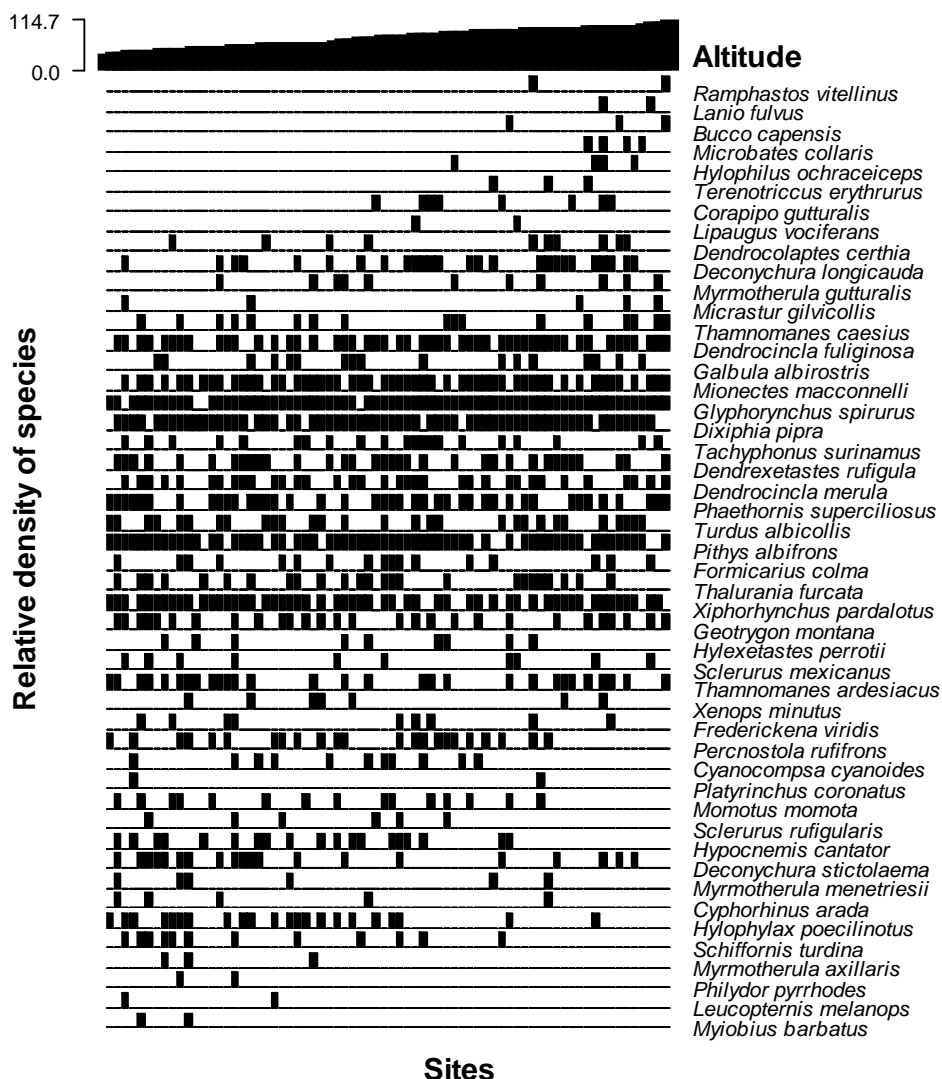
```
> megasena(10)
> megasena(5)
```

Em geral, muitas pessoas criam um código, conferem se ele está funcionando e depois transformam o código em uma função para facilitar o processo de repetição.



Criar uma função para fazer um gráfico composto (genérico)

A função `generico` foi criada para produzir um gráfico que ordena espécies em relação a um gradiente ambiental. Veja a figura abaixo onde uma comunidade de aves foi ordenada em relação ao gradiente de altitude. Esta função será bastante utilizada no curso de comunidades e no de estatística multivariada, portanto é importante que entenda como a função funciona. O código da função segue abaixo, bem como comentários que explicam o que fazer para usar corretamente a função.



Abaixo segue o código da função genérico, você irá copiar e colar o código no R da parte indicada abaixo até a parte final indicada no fim da função. Não modifique este código, uma vez que ele seja colado no R você não precisará copiar e colar novamente, desde que salve o seu workspace.

#####Copie e cole no R daqui! #####

Função para GRÁFICO COMPOSTO (Genérico)

Criada por Victor Lemes Landeiro

Última atualização em 18-07-2008

```
generico<-function(tabela,gradiente,at,grad,eixoY,eixoX){ ## Começo da
função.
```




#Veja cima que a função possui 6 argumentos, tabela, gradiente, at, grad, eixoY, eixoX. Abaixo seguem alguns comentários e exemplos sobre a função e seus argumentos.

Argumentos

```
# generico<-function(tabela, gradiente ,at,"grad","eixoY","eixoX")
```

tabela ### é a tabela de espécies, uma data frame ou matriz com os dados das espécies

gradiente ### arquivo com os valores do gradiente

at ### é usado para alterar a posição do nome das espécies no gráfico, comece com o valor 1 e vá aumentando até os nomes ficarem na posição desejada.

grad ##### é o texto para a legenda que deseja colocar no gráfico mostrando o nome do gradiente. Esta legenda não pode ser muito longa. Deve vir entre aspas.

eixoY e eixoX ##### São as legendas que deseja colocar nos eixos x e Y. Deve vir entre aspas.

##Para colocar o nome das espécies no local desejado é preciso mudar o valor do argumento at

Abaixo segue o código da função.

```
tabela<-as.matrix(tabela)
```

```
gradiente<-as.matrix(gradiente)
```

```
media.pond<-colSums(tabela*gradiente[,1])/colSums(tabela)
```

```
sub.orden<-tabela[order(gradiente[,1],decreasing=F),] # Ordenar parcelas de acordo com o gradiente
```

```
sub.orde<-sub.orden[,order(media.pond,decreasing=T)] # colocar espécies ordenadas pela média ponderada
```

```
dados.pa<-matrix(0,nrow(tabela),ncol(tabela))
```

```
dados.pa[tabela>0]<-1
```

```
ordenado<-sub.orde[,which(colSums(dados.pa)>0)] ## para deletar possíveis colunas vazias (espécie que não ocorreu)
```

```
par(mfrow=c(ncol(ordenado)+1,1),mar=c(0,4,0.2,10),oma=c(3,1,1,6))
```

```
layout(matrix(1:(ncol(ordenado)+1)),heights=c(3,rep(1,ncol(ordenado))))
```

```
plot(sort(gradiente[,1]),axes=F,ylab="",mfg=c(21,1),lwd=10,las=2,lend="butt",frame.plot=F,xaxt="n",type="h",col="black",ylim=c(min(gradiente),max(gradiente)))
```

```
axis(side=2,at=c(0,max(gradiente)),las=2)
```

```
mtext(grad,4,outer=T,font=2,line=-10,adj=-18.5,las=2)
```

```
for(i in 1:ncol(ordenado)){
```

```
barplot(ordenado[,i],bty="l",axisnames=F,axes=FALSE,col="black")
```

```
#axis(side=2,at=max(ordenado[,i]),las=2)
```

```
mtext(colnames(ordenado)[i],3,line=-1.0,adj=0,at=at,cex=.8,font=3)
```

```
}
```

```
mtext(eixoX,1,outer=T,font=2,line=1.2)
```

```
mtext(eixoY,2,font=2,outer=T,line=-2)
```

```
}
```

Este gráfico será construído através de uma matriz de dados de espécies (Presença e ausência ou abundância) que será ordenada de acordo com a média ponderada, calculada a partir dos dados de um gradiente ecológico.

Até Aqui!

Exemplo de como usar a função genérico##

Depois de copiar e colar a função você precisará digitar apenas a linha abaixo para produzir seu gráfico genérico. Vamos usar os dados dos borrachudos para fazer o gráfico

```
> ambi<-simu[,1:6] ## Variáveis ambientais
```



```
> spp<-simu[,7:26] ## Dados das espécies  
> altitude<-ambi[,6]
```

```
>generico(spp,altitude,1,"Altitude(m) ", "Densidade  
relativa de Simuliidae","Ordenado pela altitude")
```

Agora basta ir mudando o valor at para que o nome saia na posição desejada.

Em geral, esta função irá servir para propostas exploratórias, caso queria usar este gráfico em alguma publicação talvez seja necessário fazer algumas modificações no código. Uma das restrições é que o R permite que sua janela gráfica seja dividida em no máximo 50 linhas. Portanto, é **impossível** construir esse gráfico para ordenar **mais de 48 espécies**.



Referências

Braun, W.J. & Murdoch, D.J. (2007) *A first course in statistical programming with R*, -163, Cambridge University Press, Cambridge.

Chambers, J.M. (2008) *Software for data analysis: Programming with R*, -498.

Oksanen, J., Kindt, R., Legendre, P., O'Hara, R. B., and Stevens, M. H. H (2007) *vegan: Community Ecology Package*. R package version 1.8-8. <http://cran.r-project.org/>, <http://r-forge.r-project.org/projects/vegan/>

R Development Core Team (2008) *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, <http://www.R-project.org>.