



Universidade Federal de Alagoas  
Instituto de Ciências Biológicas e da Saúde  
Laboratório de Ecologia Quantitativa



1

# Introdução ao uso do *software* R para as Ciências Biológicas

**Prof. Marcos Vinícius Carneiro Vital**  
[marcosvital.wordpress.com](http://marcosvital.wordpress.com)  
[cantinhodor.wordpress.com](http://cantinhodor.wordpress.com)

Março de 2015

## Introdução ao uso do *software* R para as Ciências Biológicas

*Versão (possivelmente final) de março de 2015*

Este documento nasceu como um guia para auxiliar os estudantes da disciplina de Bioestatística do ICBS/UFAL no uso do *software* R. Aos poucos, porém, comecei a achar que ele poderia ser útil para outras pessoas interessadas em aprender a usar o programa, e comecei a editá-lo para que possa ser usado por qualquer um. Caso você deseje usar e/ou distribuir este material, peço apenas que mantenha a sua total integridade e as informações de autoria. À parte disso, ele pode ser usado e distribuído gratuitamente. Ah, e um aviso importante: tenha em mente de que este é um guia **não** tem como objetivo ensinar estatística – ele visa apenas a sua aplicação prática no R.

Como estou começando a produzir novos materiais para R, esta é possivelmente a versão final. Mas ainda posso fazer revisões, então sugestões de modificações e indicações de erros são muito bem vindos. E se por acaso você não tenha obtido este material diretamente no meu blog, sugiro acessá-lo para conferir se a sua versão é de fato a mais nova: <http://marcosvital.wordpress.com/> ou <http://cantinodor.wordpress.com/>

Esta apostila foi concebida para ser usada de maneira prática, usando o R ao mesmo tempo que se lê o texto. Como sugestão de uso, então, o ideal é manter o R aberto durante a leitura, e ir repetindo os procedimentos que aparecem aqui. Todos os procedimentos descritos no texto aparecem em imagens do R com sua execução, para que você possa comparar facilmente com o que está fazendo. Caso você não tenha em mãos os arquivos com os dados utilizados ao longo dos exemplos, confira o Anexo 1 láááá no final para obtê-los. No final de cada seção eu incluí um resumo das funções utilizadas naquela parte, para facilitar a revisão e servir para referências rápidas durante o seu aprendizado. E para facilitar ainda mais as revisões, no final de tudo há um segundo anexo, no qual dou instruções rápidas e resumidas dos principais procedimentos vistos na apostila (para os alunos da minha disciplina, este pedaço é especialmente interessante).

Espero que todos aproveitem bem este manual e tenham uma ótima experiência utilizando o R!

Atenciosamente

Marcos Vinícius Carneiro Vital

## SUMÁRIO

<b>1. Introdução</b> .....	<b>5</b>
1.1. Obtendo e instalando o software no Windows. ....	5
1.1.1 Usando o R com um IDE.....	5
1.2. Primeiro contato. ....	6
1.3. O R como uma calculadora ....	7
1.4. Usando funções no R.....	8
1.5. Usando os mecanismos de ajuda.....	9
1.6. Criando objetos.....	10
1.7. Importando uma planilha de dados para o R ....	12
1.8. Instalando e ativando pacotes no R ....	15
1.8.1. Pacotes com interface gráfica.....	16
1.9. Usando um script, inserindo comentários e salvando tudo.....	17
1.10. Resolvendo problemas comuns no R.....	20
1.10.1. Objeto não encontrado.....	20
1.10.2. Não foi possível encontrar a função “qualquer”.....	21
1.10.3. Símbolo inesperado.....	21
1.10.4. Tio, um danado de um “maiszinho” fica aparecendo no lugar do ‘>!.....	22
1.10.5. Orientações gerais.....	23
1.11. Funções utilizadas – capítulo 1.....	24
<b>2. Dados e gráficos</b> .....	<b>25</b>
2.1. Reconhecendo variáveis de uma planilha.....	25
2.2. Medidas de tendência central e de dispersão.....	26
2.3. Lidando com subconjunto de dados.....	27
2.3.1. A função tapply (e por que nós a amamos!).....	28
2.4. Estudando a distribuição de frequências de uma variável: o histograma.....	29
2.5. Gráficos, gráficos, gráficos!.....	29
2.5.1. Gráficos de dispersão.....	31
2.5.2. Gráficos de barras.....	32
2.5.3. Boxplots e gráficos de média com barras de erro.....	32
2.6. Indo além dos gráficos básicos.....	34
2.7. Funções utilizadas – capítulo 2.....	35
<b>3. Alguns testes estatísticos</b> .....	<b>36</b>
3.1. A escolha de um teste estatístico.....	36

3.2. O teste de qui-quadrado.....	36
3.3. O teste T de Student.....	38
3.3.1. Caudalidade do teste T .....	39
3.4. A análise de variância (ANOVA) .....	40
3.5. A regressão linear simples .....	41
3.6. Pressupostos dos testes estatísticos .....	43
3.6.1. Pressupostos do qui-quadrado .....	43
3.6.2. Pressupostos do teste t.....	43
3.6.3. Pressupostos da ANOVA e da regressão linear simples.....	44
3.6. Funções utilizadas – capítulo 3 .....	46
<b>4. E agora, o que vamos fazer? .....</b>	<b>47</b>
4.1. Onde buscar ajuda?.....	47
4.1.1. Documentação básica e sistema de ajuda .....	47
4.1.2. Bibliografia .....	49
4.1.3. Google.....	49
4.1.4. Listas de email .....	49
<b>ANEXO 1: dados utilizados .....</b>	<b>51</b>
<b>ANEXO 2: guia rápido!.....</b>	<b>54</b>
A2.1. Usando a ajuda. ....	54
A2.2. Lendo os dados. ....	54
A2.3. Gráficos básicos. ....	54
A2.4. Testes estatísticos. ....	55
A2.5. Testes de pressupostos. ....	55

## 1. INTRODUÇÃO

Nesta introdução nós vamos: instalar e ter um primeiro contato com o programa; ter nosso primeiro contato com uma função; aprender a usar os mecanismos de ajuda; criar objetos; importar dados; instalar pacotes e, finalmente, criar e utilizar scripts! Depois de ler e executar as instruções apresentadas aqui, você estará preparado para começar a usar o R pra valer.

### 1.1. Obtendo e instalando o software no Windows.

O R é um *software* livre, o que significa que não tem custos e que seu código fonte está acessível para qualquer usuário. Ele pode ser obtido em seu site oficial:

<http://www.r-project.org/>

Basta acessar o link ‘CRAN’ no lado esquerdo da tela, selecionar um dos endereços para fazer download, e selecionar o sistema operacional. Supondo que você está usando um computador com *Windows*, clique em ‘*Download R for Windows*’, em seguida em ‘*base*’ e, por fim, no link para *download* que aparecerá no alto da tela. O R recebe atualizações constantes, e se você desejar instalar uma versão mais recente, saiba que ele não substituirá a versão anterior, que deve ser desinstalada manualmente pelo usuário (ou mantida, o que pode ser útil para utilizar pacotes antigos que não existam nas versões mais novas). O processo de instalação em si é simples, então não vou detalhá-lo aqui.

Nunca se esqueça de que o R possui uma quantidade imensa de material sobre seu uso nas mais diversas áreas da ciência, e buscar material é bastante importante como parte do processo de aprendizado. Na página inicial do programa há alguns manuais disponíveis (no link ‘*Manuals*’, no lado esquerdo), que podem ser um bom ponto de partida, e com qualquer busca na internet é bem fácil encontrar material gratuito disponível, com os mais variados graus de detalhe e abrangência. Na medida em que desejar avançar no uso do programa, faça buscas e aproveite a existência deste material, que é uma das várias vantagens de se usar o R. Quando desejar buscar na internet por algum material específico (como uma análise ou conjunto de análises de uma determinada área), um bom ponto de partida é buscar por: CRAN “análise desejada”.

#### 1.1.1 Usando o R com um IDE

Como você mesmo verá (ou já viu), o R não é um programa muito “vistoso”. Mais do que isso, com algum tempo você poderá pensar que algumas funcionalidades

poderiam estar mais facilmente acessíveis, como a visualização dos objetos utilizados durante uma sessão, o acesso às pastas de trabalho e outras coisas do tipo. Pensando nestes e em outros aspectos, algumas pessoas desenvolveram os IDEs, que é uma sigla para *Integrated Development Environment* (algo como ambiente de desenvolvimento integrado). Os IDEs são programas que adicionam ao R uma interface de usuário mais amigável e com mais funcionalidades. Para os usuários de Windows, existem pelo menos dois IDEs bem conhecidos para o R:

RStudio: <http://www.rstudio.com/>

Tinn-R: <http://nbcgib.uesc.br/lec/software/desenvolvidos/editores/tinn-r/en>

Usá-los é simples: basta ter o R instalado no computador e depois instalar um dos programas. O RStudio é bem conhecido e muito usado por aí; o Tinn-R não tem tanta visibilidade, mas é bem conhecido aqui no Brasil, pois é desenvolvido na Universidade Estadual de Santa Cruz (UESC), em Ilhéus, Bahia. Com certeza devem existir outras opções por aí, e pode ser interessante pesquisar um pouco antes de escolher um IDE para se usar. Se quiser saber mais, um bom ponto de partida é este texto do criador do Tinn-R: <http://nbcgib.uesc.br/lec/llec/avale-es/editor-gui-ide> Agora, é claro que não há nada de errado em usar o R “puro” (eu mesmo trabalho sempre com ele assim, sem nenhum acessório), mas dar uma olhada nos IDEs é uma boa, pois ele pode facilitar bastante a vida do usuário, especialmente para quem ainda está se habituando com o programa.

## 1.2. Primeiro contato.

Ao abrir o programa pela primeira vez, não vemos um visual típico da maioria dos programas de computador, com muitos menus e ícones: o que chama a atenção é uma janela central, o ‘*R console*’. Praticamente todas as tarefas que executaremos serão feitas nesta janela, via comandos que devem ser digitados manualmente. Após algumas informações sobre a versão, a licença de uso, os contribuidores e os mecanismos de ajuda, o console termina com um símbolo ‘>’ em vermelho, onde os comandos podem ser digitados. E é isso mesmo: o cursor do R vai ficar ali te olhando, piscando e esperando você digitar alguma coisa...

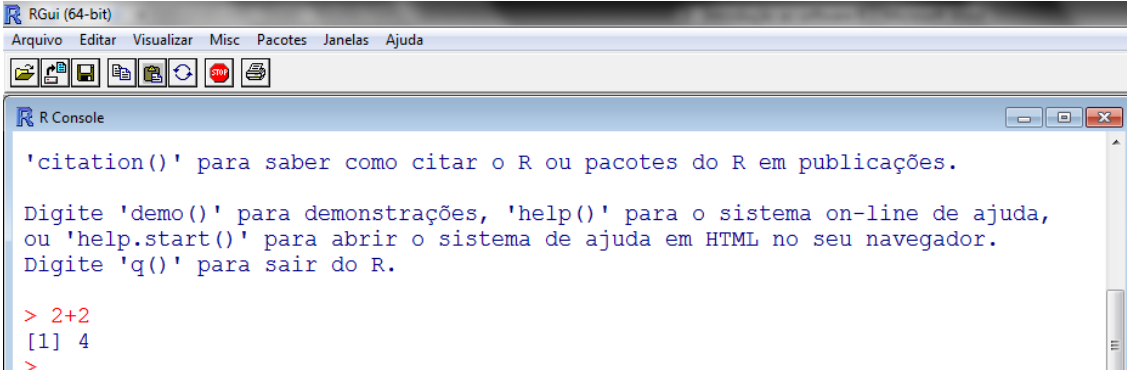
Se você nasceu no início da década de 80 (ou antes disso), provavelmente se lembrará do bom e velho MS-DOS e não achará tão estranho assim. Caso contrário, deve estar pensando: e agora?! Bom, o funcionamento básico na verdade é bem simples: basta digitar o comando desejado e teclar enter, e o programa responde executando a

tarefa desejada. Agora, é claro que se você escrever qualquer coisa, o R possivelmente retornará uma mensagem de erro, pois ele não saberá interpretar o que você “disse”. Mas não se preocupe; depois de aprender alguns comandos básicos, as coisas vão começar a fluir com mais tranquilidade e menos estranheza.

O fundamental, neste primeiro momento, é não ficar intimidado com este modo de uso em linhas de comando. Ele com certeza pode parecer assustador para quem está começando a usar o R e nunca usou programas que funcionam assim antes. Mas com algum tempo de uso esta reação inicial irá passando, e você irá ficar cada vez mais à vontade. Em resumo: não tenha medo, ok?

### 1.3. O R como uma calculadora

Nosso contato inicial com o programa R será feito usando-o para resolver operações matemáticas. Neste sentido, podemos pensar no R como uma grande e completa calculadora científica. Comece digitando uma operação bem simples, como uma soma, por exemplo, e depois tecele enter:



```

RGui (64-bit)
Arquivo  Editar  Visualizar  Misc  Pacotes  Janelas  Ajuda

R Console
'citation()' para saber como citar o R ou pacotes do R em publicações.
Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

> 2+2
[1] 4
>

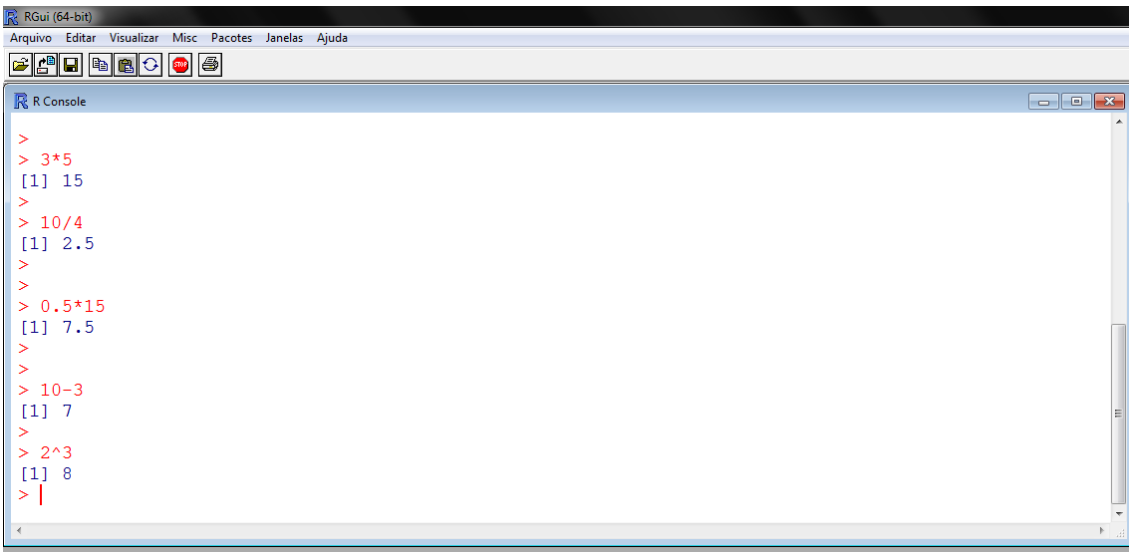
```

O programa, então, executa a operação e apresenta o resultado. Temos que chamar a atenção para duas coisas:

- o que foi executado vai ficando “para trás” no console, e não pode ser alterado ou usado novamente sem repetir a operação. Mais adiante iremos aprender a salvar os resultados de nosso interesse para posterior uso.

- o resultado aparece após o número 1 entre colchetes. Isto não é muito relevante quando temos apenas um valor como resultado, mas faz diferença quando o nosso resultado é um vetor com vários valores. Neste caso, o R sempre irá indicar, no início de cada linha do console, qual é o primeiro valor da linha (se uma linha começar com [1], por exemplo, isto quer dizer que o primeiro valor daquela linha é o 11º valor do vetor). Não entendeu? Não se preocupe, cedo ou tarde você verá isso acontecer na prática, e verá que é bem simples.

Agora siga adiante e faça algumas operações, para “sentir” um pouco o uso do R. Para multiplicação use ‘\*’, para divisão use ‘/’ e para calcular potências use o símbolo ‘^’ entre a base e o expoente (por exemplo, ‘2^3’ para dois elevado ao cubo). Lembre-se que para o R executar qualquer tarefa você deve pressionar a tecla enter quando terminar de escrever o comando desejado. Perceba, também, que o R entenderá o ponto como separador decimal, e não a vírgula (este comportamento está associado apenas aos dados inseridos diretamente no R; no caso de dados importados, o R normalmente segue a configuração sobre separador decimal do computador, convertendo as vírgulas em pontos sem problemas).



```
RGui (64-bit)
Arquivo  Editar  Visualizar  Misc  Pacotes  Janelas  Ajuda

R Console
>
> 3*5
[1] 15
>
> 10/4
[1] 2.5
>
>
> 0.5*15
[1] 7.5
>
>
> 10-3
[1] 7
>
> 2^3
[1] 8
> |
```

Bacana, não? Agora deixo uma dica bem geral sobre o uso do R. Como ele é um programa que a maioria das pessoas considera um pouco difícil de se dominar, é fundamental usá-lo com frequência, mesmo para as tarefas mais banais. Então tente se lembrar dele quando você estiver no computador e precisar da calculadora. No lugar de abrir a calculadora do Windows, abra o seu novo amigo R, e use-o para fazer as contas necessárias. Parece besteira, mas o hábito de usá-lo com certeza fará diferença no seu processo de aprendizado. E se precisar que o R resolva contas um pouco mais complicadas, veja a próxima seção, onde vamos conhecer as primeiras funções.

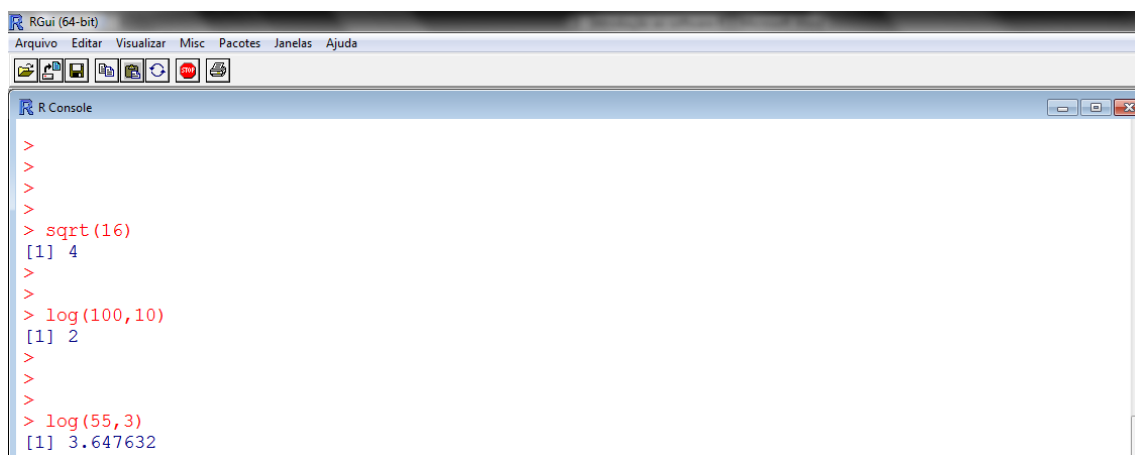
#### 1.4. Usando funções no R

As operações matemáticas básicas são bem fáceis de executar, e envolvem apenas a digitação dos operadores matemáticos em si. Mas é claro que o R permite ir além disso, e realizar outras operações a partir do uso de suas funções. Todas as funções



no R seguem a mesma lógica de uso: basta digitar o nome da função seguido dos argumentos necessários, que devem estar entre parênteses. Ao longo desta apostila nós vamos explorar muitas funções, e seu uso ficará cada vez mais familiar. Sempre que você vir, ao longo do texto, uma palavra em *itálico* seguida de “()”, estará diante de uma função.

Neste primeiro momento, vamos experimentar algumas funções matemáticas simples. Experimente, por exemplo, calcular a raiz quadrada de um número usando a função `sqrt()`. Se você digitar `sqrt(16)`, por exemplo, estará calculando a raiz quadrada de 16. Agora vamos calcular um logaritmo com a função `log()`, digitando `log(100,10)`, que é o comando para se calcular o logaritmo de 100 na base 10. Note que como a função precisou de dois argumentos, nós os separamos por uma vírgula, e este é um outro aspecto geral das funções no R: os argumentos são sempre separados por vírgulas (você também pode incluir espaços entre os argumentos, o que não afeta o seu funcionamento).



```
RGui (64-bit)
Arquivo Editar Visualizar Misc Pacotes Janelas Ajuda

R Console
>
>
>
> sqrt(16)
[1] 4
>
>
> log(100,10)
[1] 2
>
>
> log(55,3)
[1] 3.647632
```

### 1.5. Usando os mecanismos de ajuda

No tópico anterior, usamos a função `log()`, que utilizou dois argumentos: o número e a base. Um questionamento que pode surgir é: como saber quais são os argumentos de uma função? E a resposta é simples e muito útil: basta utilizar o mecanismo de ajuda do R, que pode ser chamado digitando um ponto de interrogação seguido pelo nome da função desejada. Experimente, então, usar o comando `?log`. O resultado é uma nova janela, aberta no navegador de internet, com informações da função `log()`. Tome algum tempo para explorar o formato geral da página de ajuda, pois ela segue um padrão em todas as funções. Um dos pedaços mais importantes das

páginas de ajuda das funções fica ao seu final: quase sempre há um ou mais exemplos de uso, o que é uma ótima ferramenta para aprendizado.

A próxima pergunta provavelmente é: mas e se eu nem mesmo sei o nome da função que desejo?! Aqui entra o mecanismo de busca do programa, que em sua versão mais simples pode ser chamado com uma dupla interrogação seguida do termo a ser buscado. Experimente, por exemplo, usar o comando `??variance` (para buscar funções que façam referência à palavra variância), e explore um pouco o resultado, composto por diversas funções que se relacionam à palavra utilizada. Uma versão um pouco mais completa desta mesma função é o `help.search()`, na qual termos com mais de uma palavra poderão ser buscados, bastando estarem entre aspas simples. Experimente o comando `help.search("linear regression")` e veja os resultados. Como toda documentação do R está em inglês, lembre-se sempre em buscar utilizando as palavras neste idioma, ok?

Uma característica bacana do sistema de help do R é o seu funcionamento no formato *html*, aberto pelo navegador de internet padrão do computador. Este sistema permite que você possa abrir novas janelas ou abas usando os links (que aparecem como palavras azuis grifadas) que aparecem em diversas palavras-chave que possuem páginas de ajuda próprias. Desta forma, a partir de uma função você pode explorar outras relacionadas, navegando de função em função. E não se engane: apesar de utilizar o navegador, as páginas de ajuda estão todas em seu computador, então não há necessidade de conexão com a internet.

```
>
> ?log
starting httpd help server ... done
>
>
> ??variance
>
>
>
> help.search('linear regression')
```

## 1.6. Criando objetos

A linguagem R faz parte de uma categoria de linguagens de programação chamada de “programação orientada a objetos”. A consequência prática para nós, meros mortais usuários do programa, é que quase tudo o que desejarmos fazer nele se relacionará com algo que ele identifica como objeto. Um objeto será reconhecido como um nome qualquer que nós mesmos atribuímos, e guardará “dentro” dele um conjunto de informações (valores, nomes, etc.). A criação de um objeto no R é simples e

intuitiva: basta criar um nome e “apontar uma seta” para ele. A tal seta é um comando chamado de *assign*, e é formada por um traço (o sinal de menos do teclado) e o símbolo ‘>’ ou ‘<’ (a escolha do símbolo depende da direção). Exemplificando, digite o comando: `objeto1<-15`. Se o R não der nenhuma resposta, ele criou corretamente o objeto denominado “objeto1”, que é composto pelo número 15. O fato do R ficar “silencioso” após o comando é normal: nós mandamos o programa criar o objeto, e só; ele só iria exibir algum tipo de resultado se fosse instruído para tanto. Se quiser, você pode dar o mesmo comando apontando a seta para o outro lado, assim: `15->objeto1`. O resultado é o mesmo.

Agora que criamos o objeto, podemos usá-lo à vontade. Por exemplo, podemos “chamar” o objeto no R, simplesmente digitando seu nome e teclando enter. Experimente fazer isso, e veja o resultado. Também podemos realizar operações matemáticas. Experimente, por exemplo, digitar a operação `objeto1*2`. O R exibirá o resultado, mas perceba que o objeto continua inalterado (basta chama-lo de novo e conferir)! Isto acontece porque nós apenas perguntamos ao R qual o resultado de se multiplicar o objeto por 2. Se quisermos que ele armazene o resultado no lugar do número original, então o comando seria: `objeto1<-objeto1*2`.

```
>
> objeto1<-15
>
> objeto1
[1] 15
>
> 15->objeto1
>
> objeto1
[1] 15
>
> objeto1*2
[1] 30
>
> objeto1
[1] 15
>
> objeto1<-objeto1*2
>
> objeto1
[1] 30
>
> .
```

The screenshot shows an R console session with several commands and their outputs. Blue callout boxes point to specific lines of code to highlight key concepts:

- A callout points to `objeto1<-15` and `objeto1` with the text: "Note que ele apenas calculou o resultado..."
- A callout points to `objeto1*2` with the text: "... sem salvá-lo no objeto!"
- A callout points to `objeto1<-objeto1*2` with the text: "Aqui sim o resultado foi salvo no objeto."
- A callout points to the final `objeto1` command with the text: "Viu só?"

Esta característica do R que você acabou de ver após a multiplicação do nosso objeto por 2 é bem típica das linguagens de programação, e precisamos nos acostumar com ela. Apesar os computadores serem capazes de processamentos incrivelmente rápidos e complicados, eles são, em essência, meio burrinhos... O que um computador sabe fazer muito bem é seguir instruções, mas a questão é que eles sempre as seguem ao pé da letra! Então, quando você achar que o R não está fazendo exatamente o que você

precisa, pense duas vezes sobre o comando que você utilizou, pois pode ser apenas uma questão de comunicação. Antes de seguirmos, o contexto me obriga a contar uma velha piada sobre programação! O programador chega em casa e encontra um bilhete da esposa: “vá à padaria e compre duas caixas de leite. Se tiver pão, compre dez.” Chegando na padaria, ele pergunta: tem pão? Como a resposta foi sim, ele comprou dez caixas de leite. Tun-tun-tss! (se ouvir a piada durante a aula, tenha um mínimo de educação, e ria um pouco para agradar o coitado do professor)

Mas voltemos ao que interessa. Um comando simples, mas bastante importante, permite criar objetos com sequências de valores (em um formato de vetor), e pode ser chamado no R simplesmente com a letra *c*. Se precisarmos de um objeto com os números inteiros de 1 a 10, por exemplo, poderíamos criá-lo assim: `sequência<-c(1,2,3,4,5,6,7,8,9,10)`.

E nunca se esqueça: ao nomear um objeto no R, você sempre deverá chamá-lo pelo nome exatamente como foi criado. Se mudar uma letra maiúscula para minúscula, um acento ou qualquer outro pequeno detalhe, o programa não irá entender. Por fim, o comando `ls()` sempre pode ser chamado quando se quiser saber quais objetos estão na memória.

```
>
> sequência<-c(1,2,3,4,5,6,7,8,9,10)
>
> sequência
[1] 1 2 3 4 5 6 7 8 9 10
>
> Sequência
Erro: objeto 'Sequência' não encontrado
>
> sequencia
Erro: objeto 'sequencia' não encontrado
>
> ls()
[1] "objeto1" "sequência"
>
> |
```

Uma letra maiúscula faz toda a diferença.

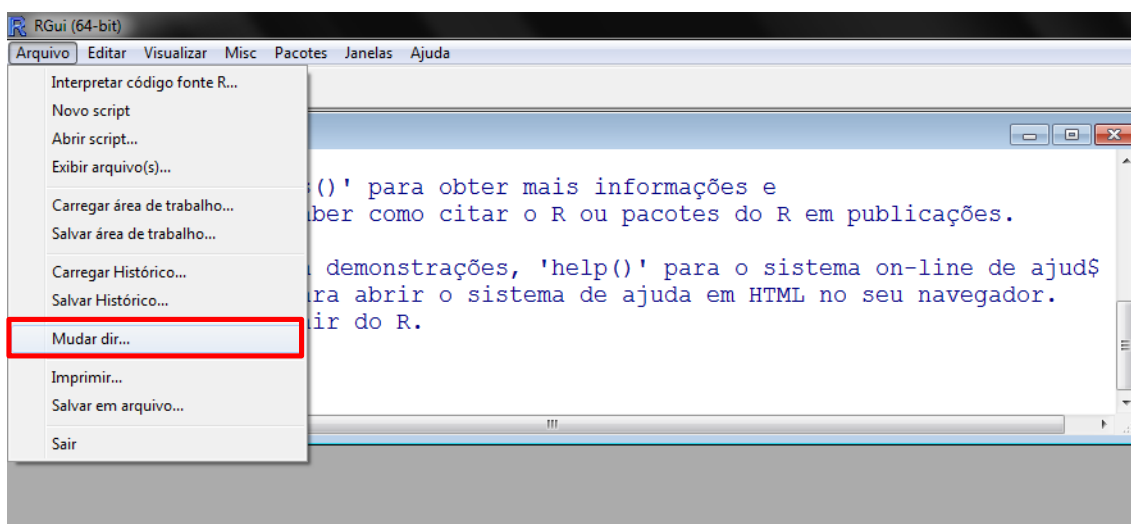
E um simples acento também.

## 1.7. Importando uma planilha de dados para o R

Agora que já temos uma idéia de como criar e utilizar objetos, vamos dar um passo além, e aprender a importar uma planilha de dados “para dentro” de um objeto do R. Esta é uma tarefa importantíssima, pois na esmagadora maioria das vezes nós criamos uma planilha de dados em um gerenciador de planilhas (como o Excel ou o Calc), que oferece um ambiente voltado para organização e armazenamento de

informações. O R oferece diversos comandos que permitem a importação de dados, envolvendo diversos formatos de arquivo diferentes. Como no dia à dia vocês podem se deparar com arquivos de diferentes extensões, uma maneira de simplificar tudo é transformar os dados em um formato chamado de “texto separado por tabulações” (com a extensão ‘.txt’), e importa-los a partir daí. O processo de conversão de um formato de planilha para o txt com tabulações é simples: podemos copiar e colar o conteúdo no bloco de notas e salvar normalmente por lá; ou usar a função de ‘salvar como’ do gerenciador e selecionar o formato apropriado.

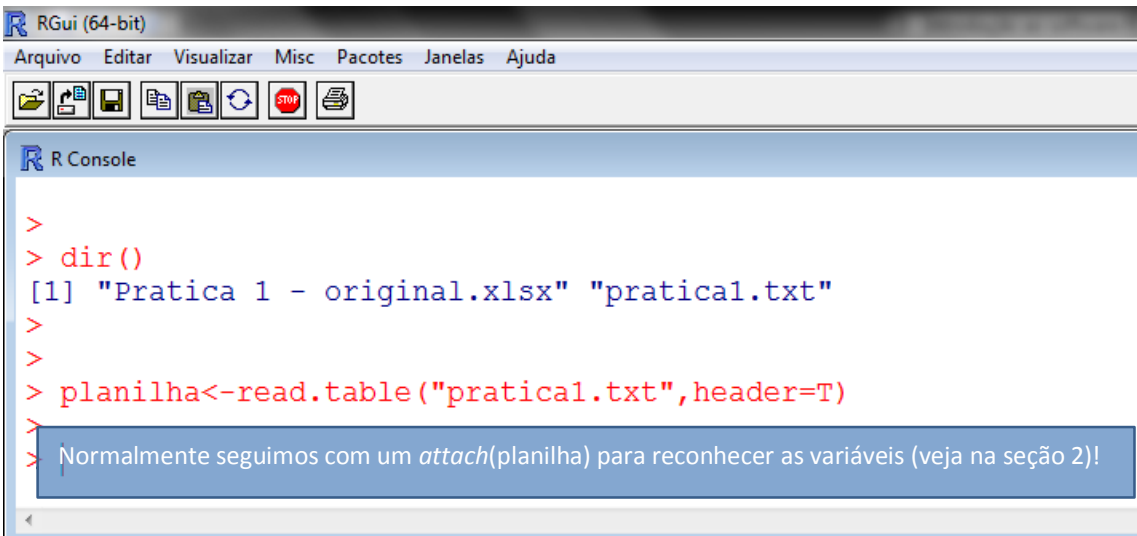
Uma vez criado o arquivo, devemos primeiro indicar para o R a sua localização, antes de importá-lo de fato. Este é um processo simples, feito pelo menu do R: basta acessar o menu ‘Arquivo’ e selecionar a opção ‘Mudar dir...’, e selecionar o diretório onde o arquivo se encontra (lembre-se de que o R não “verá” o arquivo, pois estamos apenas selecionando o diretório). Este procedimento estabelece o que chamamos de “diretório de trabalho”, e é um aspecto bem importante para termos um bom uso do programa. Uma vez que o diretório de trabalho está estabelecido, tudo o que tentarmos importar e tudo aquilo que exportarmos (ou seja, salvarmos “para fora” do R) será automaticamente feito ali. Uma boa recomendação é criar uma pasta de fácil acesso, e dentro dela criar subpastas para cada análise, projeto ou outra coisa relevante. E caso você deseje definir o diretório de trabalho “manualmente”, veja as funções `setwd()` para determinar o diretório e `getwd()` para saber qual o diretório atual.



Após a escolha do diretório desejado, caso queira você poderá conferir quais arquivos estão salvos dentro dele com o comando `dir()`. Após conferir se o arquivo desejado está ali, basta importá-lo para um objeto usando o comando `read.table()`. O

comando pede apenas um argumento obrigatório: o nome do arquivo (que deve estar entre aspas e com a extensão). Mas usaremos outro argumento, este opcional: informaremos ao R que a nossa planilha tem um cabeçalho, que é uma linha que identifica o nome das variáveis. O comando completo, então, será: `read.table("nomedoarquivo.txt",header=T)`. O argumento `header=T` é a indicação de que existe um cabeçalho (o T é a abreviação de TRUE).

Neste momento, siga os exemplos das imagens ilustrativas, e importe o arquivo "pratica1.txt" para o R, pois vamos continuar a usá-lo nas próximas seções. Caso não tenha o arquivo, pule a parte da importação e veja como obter os dados no Anexo 1 desta apostila.



```

RGui (64-bit)
Arquivo Editar Visualizar Misc Pacotes Janelas Ajuda

R Console
>
> dir()
[1] "Pratica 1 - original.xlsx" "pratica1.txt"
>
>
> planilha<-read.table("pratica1.txt",header=T)
>
> Normalmente seguimos com um attach(planilha) para reconhecer as variáveis (veja na seção 2)!

```

Se por acaso você precisar abrir um arquivo que, por algum motivo, está fora do seu diretório de trabalho, experimente usar a função `file.choose()` como argumento no lugar do nome do arquivo, e veja o que acontece! Legal, não é?

Agora que conseguimos importar um conjunto de dados para o R, podemos conferir o que foi importado. A maneira mais básica de se fazer isso é simples: basta "chamar" o objeto que recebeu os dados. Este método, porém, não é lá muito prático, pois todos os dados irão aparecer no console, o que não é útil para planilhas com muitas linhas e/ou colunas. Vamos preferir, então, usar funções que permitam uma visão mais compacta dos nossos dados. A função `head()` é uma delas: ela fará o R mostrar no console apenas as seis primeiras linhas da tabela importada, permitindo, por exemplo, conferir os nomes das variáveis. Outra função bastante útil é `summary()`, que nos dá algumas estatísticas descritivas dos dados (o resultado exato depende da natureza das variáveis).

O comando `summary()` nos dá: para variáveis qualitativas, o número de vezes que uma classe apareceu; e para variáveis quantitativas, o valor mínimo, o primeiro quartil, a mediana, a média, o terceiro quartil e o valor máximo (nesta ordem). Além desta utilidade básica, `summary()` é uma função bastante útil para se inspecionar diversos objetos do R, como objetos que guardam os resultados de um teste estatístico, por exemplo. Na medida em que aprender a usar o R, use esta função de diversos tipos de objetos para que você possa se explorá-la ao máximo. Neste contexto de exploração de objetos também há outra função bem importante: a `str()`. O seu resultado não é lá muito intuitivo para um usuário iniciante, mas na medida em que compreendemos melhor o funcionamento do R esta se torna uma função muito útil para se inspecionar todo tipo de objeto e até mesmo funções.

```
>
> head(planilha)
  UA Ambiente Área Riqueza Abund_sp1
1  1 primário 101     35      3
2  2 primário 115     31      3
3  3 primário 143     39      1
4  4 primário  92     25      6
5  5 primário  51     22      1
6  6 primário  89     35      0
>
>
> summary(planilha)
  UA                Ambiente      Área      Riqueza      Abund_sp1
Min. : 1.00   primário :22   Min. : 40.0   Min. :14.00   Min. :0.00
1st Qu.:13.25 secundário:28 1st Qu.: 79.5 1st Qu.:23.00 1st Qu.:1.00
Median :25.50                Median :102.5 Median :29.50 Median :2.00
Mean   :25.50                Mean   :102.5 Mean   :29.22 Mean   :2.16
3rd Qu.:37.75                3rd Qu.:127.8 3rd Qu.:35.00 3rd Qu.:3.00
Max.   :50.00                Max.   :167.0 Max.   :48.00 Max.   :8.00
>
```

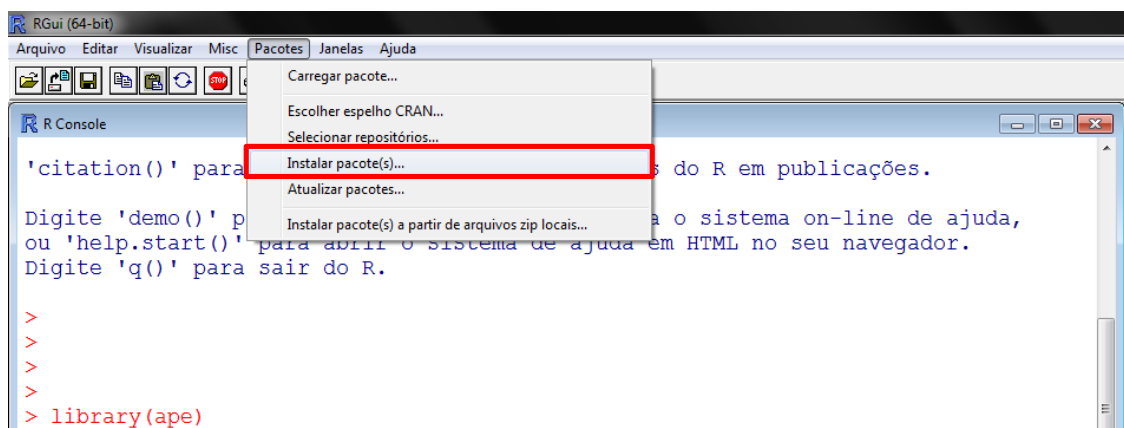
**Para os mais desesperados para mexer nos dados:** se você for muito ansioso e quiser continuar a explorar os dados, pule as próximas páginas e rume direto para o capítulo 2, onde continuaremos a mexer neles. Mas depois volte aqui e veja o que perdeu, pois há informações bem importantes, ok?

## 1.8. Instalando e ativando pacotes no R

Agora que os aspectos mais gerais do R já foram cobertos, você já deve ser capaz de usar o programa, tanto seguindo roteiros e sugestões de funções quanto descobrindo por conta própria funções que possam fazer o que você precisa. Cedo ou tarde, entretanto, você irá descobrir que o R é instalado apenas com um conjunto relativamente básico de funções, o que pode tornar necessário o uso de um pacote. Um

pacote do R é um conjunto de funções compiladas juntas com um objetivo e/ou tema em comum. Há pacotes voltados para análise de diversidade biológica (como o *vegan* e o *biodiversityR*, por exemplo), pacotes para o uso de métodos filogenéticos comparativos (como o pacote *ape*), pacotes com interface gráfica de usuário (como o *Rcmdr*), e muito mais.

Carregar um pacote que já esteja no seu computador é bem simples: basta usar o comando `library()` com o nome do pacote. Caso você ainda precise instalar o pacote, uma das maneiras mais simples é fazer isso pelo próprio R pelo menu ‘Pacotes’, na opção ‘Instalar Pacotes’. Esta função necessita que o computador tenha uma conexão com a internet, e irá perguntar qual servidor usar para o pacote ser baixado. Uma vez baixado e instalado, o pacote já estará na pasta adequada, e poderá ser carregado com a função `library()`. Ah, e se você usar o comando `library()` sem nenhum argumento, só com os parênteses vazios mesmo, o R mostrará para você a lista de pacotes que já está no seu computador.



Por fim, para saber qual o objetivo geral dos pacotes, você pode acessar a *homepage* do R e, seguindo o mesmo caminho para baixar o programa, acessar o link *packages* e acessar a opção “*Table of available packages, sorted by name*”. Você verá, em seguida, a lista de todos os pacotes oficiais do programa, com uma breve descrição do seu propósito.

### 1.8.1. Pacotes com interface gráfica

O R possui alguns pacotes que, além de adicionar novas funções à nossa disposição, adicionam novas maneiras de interagirmos com o programa ao criarem uma nova janela de interface com o usuário. Dois pacotes são particularmente interessantes: o *Rcmdr* e o *GrapheR*. O primeiro é focado na realização dos principais testes estatísticos, e o segundo é voltado para a produção de gráficos. Se, por um lado, é



fantástico termos uma interface de usuário mais amigável, por outro devemos ter muita cautela de não dependermos disso, pois pacotes com estes são sempre, por definição, limitados a algumas funções.

Uma boa maneira de usá-los é como uma ferramenta acessória no aprendizado: ambos permitem que você veja os comandos das funções executadas, o que pode ser muito útil para se aprender a usá-las. Em suma: vale à pena conhecê-los, mas eles devem sempre ser usados com muita cautela, para que o usuário não acabe dependendo completamente deles e acabe não dominando o uso do R. Se você não ficar atento a isso, pode acabar se dando muito mal no futuro, quando precisar executar uma função que não existe em um pacote destes (o que, na prática, é a realidade da maioria das funções)! Então resista à tentação de usá-los o tempo todo (especialmente o *Rcmdr*), e não se deixe seduzir pelo lado negro da força, ok?

### **1.9. Usando um script, inserindo comentários e salvando tudo**

Comparado com a maioria dos outros *softwares* de estatística, o R pode ser considerado um tanto exigente com o usuário, uma vez que praticamente todas as suas funções devem ser executadas por linhas de comando. Para lidar com isso sem grandes dificuldades, podemos nos aproveitar de algumas funcionalidades do programa: a inserção de comentários, o uso de scripts e o armazenamento das informações do console.

Inserir comentários no R é uma tarefa bem simples: basta começar a linha com o símbolo #, e tudo o que for escrito em seguida não será executado. É recomendável que os comentários sejam usados durante o processo de aprendizado, na forma de lembretes sobre o uso das funções. Eles também são bastante úteis durante a interpretação de resultados de análises, pois permitem que o usuário anote as suas primeiras conclusões e interpretações diretamente com os resultados.

```

RGui (64-bit)
Arquivo  Editar  Visualizar  Misc  Pacotes  Janelas  Ajuda

R Console

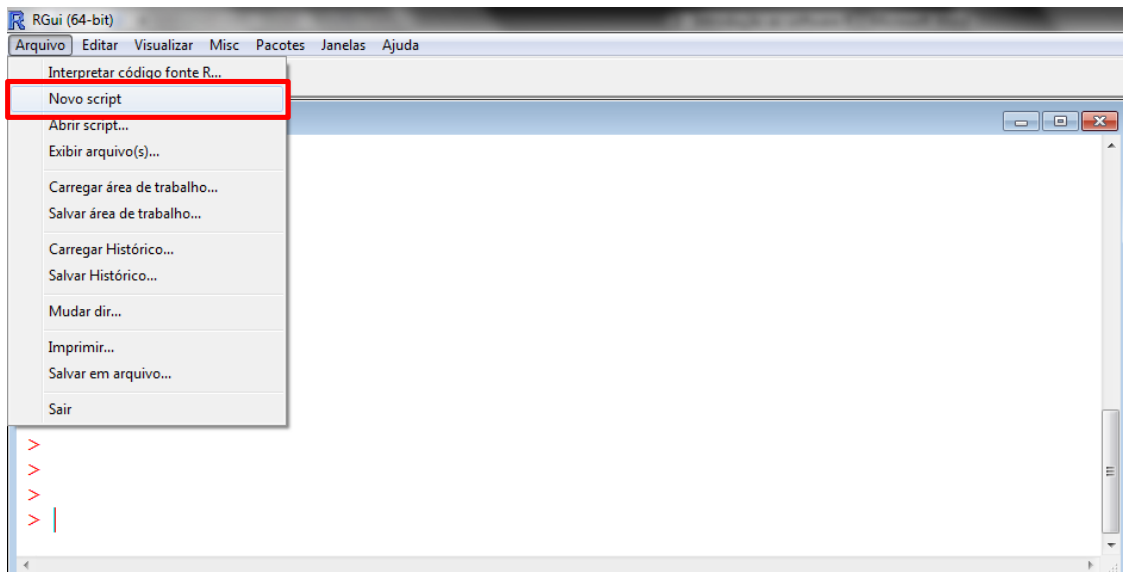
Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

>
>
>
>
>
> #Comentários podem ser acrescentados com o símbolo '#'
> #E o R irá ignorar a execução das linhas que começam com este símbolo.
> #Você pode até mesmo escrever uma função, que ela não será executada:
> #dir()
> |

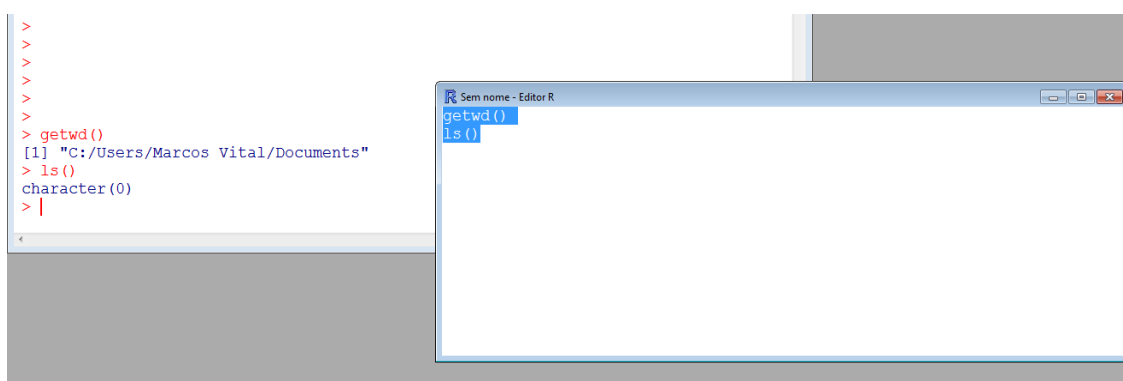
```

Mas você pode se organizar ainda mais do que isso! Aliás, não só pode como deve. O console do R tende a ficar um tanto bagunçado na medida que vamos usando o programa, especialmente estamos aprendendo a usar. Ele vai juntando mensagens de erro, resultados repetidos e coisas do tipo, e no final fica bem feioso de se usar para alguma coisa. É aí que entram os scripts. Eles são seqüências de comandos que podem ser armazenados e usados de novo quantas vezes quisermos. Podem ter quantos comentários quanto acharmos necessário, e não acumulam informações de resultados ou mensagens de erros. E tem mais: no cotidiano do uso do R, é bem comum que precisemos fazer o mesmo procedimento diversas vezes com diferentes conjuntos de dados. Isto pode se tornar bem trabalhoso (para não dizer um pé no saco!) quando temos que repetir linhas de comando muito longas, o que torna muito prático armazenar linhas prontas para que possam ser usadas novamente. E aí entram novamente os scripts, que podem conter modelos de análises, para você usar e modificar várias vezes.

Criar um script no R é bem fácil: vá no menu ‘Arquivo’ e selecione a opção ‘Novo script’. O R então irá abrir uma janela em branco, onde podemos digitar o nosso script.



Um script do R é uma simples sequência de comandos (que podem incluir comentários), que podem ser enviados para serem executados no console quando o usuário assim o desejar. Basta selecionar a(s) linha(s) desejada(s) (ou selecionar todas usando o atalho `ctrl+a`) e usar o atalho `ctrl+r`, e o console irá executar tudo. Uma vez que a janela do script estiver selecionada, ele poderá ser salvo pelo menu ‘Arquivo’ na opção ‘Salvar’. Os arquivos de script do R são salvos como a extensão `.R`, mas podem ser abertos pelo bloco de notas se você quiser.



Podemos criar scripts para diversas situações: para importação de dados, carregamento de pacotes, execução de análises estatísticas, etc. É claro que o usuário deve ficar atento para os nomes de arquivos, objetos e variáveis, pois quaisquer mudanças podem impedir a execução correta do script. Uma dica é tentar, quando possível, usar nomes “genéricos” de arquivos e objetos, evitando que o script tenha que ser muito modificado a cada nova execução. Para você que está aprendendo, abuse dos

comentários nos seus scripts! Assim você alia a utilidade prática deles para realizar análises com o processo de aprendizado. Recomendo fortemente usá-los no dia à dia do R, pois com o tempo eles serão grandes aliados no seu uso e aprendizado.

Por fim, após uma sessão de uso do R, você pode achar interessante salvar tudo o que se passou no console, para se ter registrados os procedimentos, seus resultados e os eventuais comentários adicionados. Este é um procedimento bem simples: basta ir novamente ao menu ‘Arquivo’ e selecionar a opção ‘salvar em arquivo’. O R então irá salvar um arquivo de extensão .txt, que poderá ser aberto em qualquer editor de texto. Caso você tenha problemas em abrir o arquivo depois de salvá-lo, verifique se o R inclui a extensão; caso ele não a tenha incluído, basta renomear o arquivo incluindo a extensão .txt no final do nome.

## 1.10. Resolvendo problemas comuns no R

AIMEUDEUSDOCÉUORNÃOFAZOQUEEUQUERO!!!!!!1!!!

Calma, muita calma. Quando o R não faz o que queremos, ele normalmente nos responde com uma mensagem de erro, que pode ser bastante informativa sobre o que está acontecendo. Lembre-se sempre de ler com calma as mensagens de erro que surgirem, e tentar interpretá-las, pois na imensa maioria das vezes a solução é bem simples. Na medida em que nos habituamos com o programa, começamos a perceber quais são os erros mais comuns, e as coisas vão ficando cada vez mais fáceis. Vamos ver, a seguir, alguns dos erros mais comuns que costumamos ter com o R e como podemos solucioná-los.

### 1.10.1. Objeto não encontrado

```
>
> dadoz
Erro: objeto 'dadoz' não encontrado
> |
```

Essa não, um erro! :p

O erro acima ocorre quando tentamos fazer referência a um objeto que não existe. Pode ocorrer quando nos esquecemos de importar os dados ou quando simplesmente escrevemos errado o nome do objeto. **Solução 1: confira quais objetos estão na memória do R e quais os seus nomes com um ls().**

```
>
> ls ()
[1] "dados"
```

Convenhamos, este era óbvio. Quem iria escrever dados com z?!

O mesmo erro costuma aparecer quando importamos os dados mas nos esquecemos de usar o comando `attach()` (veja-o na próxima seção) para que o R reconheça os nomes das variáveis. **Solução 2: use o comando `attach()` nos dados importados.**

```
> Riqueza
Erro: objeto 'Riqueza' não encontrado
>
> attach(dados)
>
> Riqueza
 [1] 35 31 39 25 22 35 43 48 35 38 32 33 40 16 31 23 32 35 22 41 24 18 39 32 19
[26] 20 31 26 20 33 36 32 28 23 28 16 44 27 37 24 19 28 27 28 27 20 34 20 31 14
```

### 1.10.2. Não foi possível encontrar a função “qualquer”

A mensagem é bem clara: você tentou chamar uma função que não existe. Provavelmente você não digitou corretamente o nome da função. **Solução 1: reveja o que escreveu em busca de erros (fique atento ao detalhes, como letras maiúsculas e símbolos).**

```
>
> dados<-read.tab("practical.txt",header=T)
Erro: não foi possível encontrar a função "read.tab"
>
> dados<-read.table("practical.txt",header=T)
```

Outra possibilidade é ter tentado chamar uma função de um pacote sem o ter carregado previamente. **Solução 2: use o comando `library()` para carregar o pacote necessário.**

```
> lineplot.CI(Ambiente,Riqueza)
Erro: não foi possível encontrar a função "lineplot.CI"
>
> library(sciplot)
> lineplot.CI(Ambiente,Riqueza)
```

### 1.10.3. Símbolo inesperado

Um erro comum, normalmente gerado quando erramos pontuação e/ou espaçamento. No caso de objetos, costuma ocorrer quando adicionamos um espaço ou símbolo que não existe no nome original (e o espaço é encarado como um símbolo pelo R). **Solução 1: novamente, confira o que você escreveu, e se necessário use o `ls()`.**

```

> aula pratica
Erro: símbolo inesperado in "aula pratica"
>
> aula,pratica
Erro: ',' inesperado in "aula,"
>
> ls()
[1] "aula.pratica"
>
> aula.pratica
[1] 1 4 1 0 1 1 2 0 1 0

```

Quando associado à uma função, o mais provável é ter esquecido uma vírgula entre os argumentos ou ter feito alguma confusão com aspas ou parênteses. **Solução 2: confira o comando, e lembre-se de que todos os argumentos de uma função devem ser separados por vírgula.**

```

> plot(Área Riqueza)
Erro: símbolo inesperado in "plot(Área Riqueza)"
>
> plot(Área,Riqueza)
> |

```

#### 1.10.4. Tio, um danado de um “maisinho” fica aparecendo no lugar do ‘>’!

Pois é, o famoso símbolo de “mais” atormenta muita gente, mas é algo muito simples e não é um erro em si: ele surge como parte de uma funcionalidade do próprio R, que é permitir que um comando seja escrito em várias linhas. Caso você esteja escrevendo uma função bem longa, com muitos argumentos, o R permite que você a escreva aos poucos, teclando *enter* entre um conjunto de argumentos caso ache necessário. Para isso ser possível, basta que você tenha começado a escrever os argumentos e tecla *enter* antes de fechar os parênteses daquela função ou aspas de uma parte da função; depois, basta terminar de escrever e teclar *enter* novamente para que tudo volte ao normal.

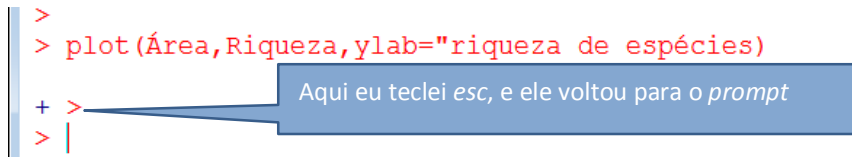
```

<
> plot(Área,Riqueza,
+ ylab="Riqueza de espécies")
> |

```

Eventualmente erramos ao escrever uma função, e o símbolo de mais surge porque esquecemos de fechar parênteses ou aspas. Se não for mais possível completar o comando, basta teclar *esc* para que o R cancele o comando e volte para o *prompt*.

```
>  
> plot(Área,Riqueza,ylab="riqueza de espécies")  
+ >  
> |
```

A screenshot of an R console window. The first line shows a prompt character followed by the command `plot(Área,Riqueza,ylab="riqueza de espécies")`. The second line shows a prompt character followed by a plus sign and another prompt character. The third line shows a prompt character followed by a vertical bar. A blue callout box with a pointer to the plus sign contains the text "Aqui eu teclei esc, e ele voltou para o prompt".

Aqui eu teclei *esc*, e ele voltou para o *prompt*

#### 1.10.5. Orientações gerais

A maioria dos erros com os quais nos deparamos usando o R são coisas pequenas: um errinho de digitação aqui, uma vírgula mal colocada ali, um parênteses não fechado acolá, e por aí vai. Quando topar com um erro, então, sempre comece revisando com calma todas as etapas que você seguiu, conferindo tudo o que você escreveu e comparando com o que for relevante (nomes de arquivo ou de objetos, etc). Quase sempre será bem fácil de resolver, e na medida em que descobrimos os nossos próprios erros aprendemos mais e diminuimos a chance de cometê-los de novo. Agora, quando o erro persiste e você não entende por quê, não há nada de errado em procurar ajuda! Fóruns na internet, listas de discussão, colegas e professores: mais cedo ou mais tarde você irá encontrar alguém que possa ajudar. Um dos aspectos mais interessantes do R é que existe uma grande comunidade de usuários por trás do programa, e devemos saber usar isso a nosso favor.

### 1.11. Funções utilizadas – capítulo 1

Estas foram as funções que vimos até aqui:

`+`, `-`, `*`, `/` → operadores matemáticos

`sqrt()` → raiz quadrada

`log()` → logarítimo

`?`, `??`, `help.search()` → mecanismos de ajuda

`<-`, `->` → associa elementos (valores, nomes, planilhas, etc) a objetos do R

`c` → combina elementos em um vetor ou uma lista

`ls()` → lista os objetos na memória do R

`dir()` → lista os arquivos no diretório de trabalho

`read.table()` → lê uma planilhas em arquivo externo

`head()` → lista as seis primeiras linhas de uma planilha

`summary()` → resume as informações de um objeto do R

`str()` → função para inspecionar um objeto ou função, mostrando detalhes de sua estrutura.

`library()` → carrega um pacote que já esteja no computador

`#` → permite a inserção de comentários



## 2. DADOS E GRÁFICOS

Neste capítulo nós vamos lidar com conjuntos de dados no R, dando os primeiros “passos estatísticos” com o programa. Especificamente, nós vamos aprender funções básicas para sumarizar e manipular dados, além de aprender a criar os nossos primeiros gráficos. Em outras palavras: vamos meter a mão na massa e fuçar os dados!

### 2.1. Reconhecendo variáveis de uma planilha

Na seção passada, aprendemos a usar a função `read.table()` para importar os dados de uma planilha para o R, na forma de um objeto. O programa pode lidar com informações em vários formatos, mas como o nosso foco é em estatística, vamos lidar com dados organizados em planilhas nas quais as unidades amostrais estejam nas linhas e as variáveis estejam nas colunas. Digitar os dados diretamente no R também seria possível, mas não é lá muito prático, e o ideal é usar um bom e velho gerenciador de planilhas para isso. Agora, se você estiver muito curioso, dê uma espiada nas funções `data.frame()`, `edit()` e `fix()` para saber mais sobre como inserir dados diretamente no R, sem importação.

Uma vez que um conjunto de dados esteja inserido no R na forma de um objeto, é conveniente que sejamos capazes de fazer o programa reconhecer as variáveis pelo seu nome. Existem duas maneiras de fazer isso: a mais direta, sem usar funções intermediárias, é usar o símbolo `$`. Basta escrever `objeto$variável` para que o R seja capaz de entender a referência. Este detalhe sobre a linguagem do R pode ser bem útil na manipulação de objetos que são gerados como resultado de análises estatísticas, então é muito importante se lembrar disso. Já a outra maneira, que é bastante prática, é usar o comando `attach()`, que faz com que todas as variáveis em uma planilha possam ser identificadas pelo nome, da mesma forma que um objeto qualquer (mesmo que elas não se transformem em objetos de fato). Há também, por fim, um comando que desfaz o `attach()`: é o comando `detach()`. Este é um comando importante quando vamos lidar com diversos conjuntos de dados diferentes em uma mesma seção do R.

E não se esqueça: o R é “sensível”, tadinho. Se ao escrever o nome da variável você esquecer de um acento, adicionar uma letra maiúscula ou coisa do tipo, ele vai responder bem magoado com uma mensagem de erro... Já falamos sobre isso antes, mas não custa lembrar mais uma vez, não é?

```

> summary(dados)
      UA      Ambiente      Área      Riqueza      Abund_spl
Min.   : 1.00   primário :22   Min.   : 40.0   Min.   :14.00   Min.   :0.00
1st Qu.:13.25   secundário:28   1st Qu.: 79.5   1st Qu.:23.00   1st Qu.:1.00
Median :25.50                    Median :102.5   Median :29.50   Median :2.00
Mean   :25.50                    Mean   :102.5   Mean   :29.22   Mean   :2.16
3rd Qu.:37.75                    3rd Qu.:127.8   3rd Qu.:35.00   3rd Qu.:3.00
Max.   :50.00                    Max.   :167.0   Max.   :48.00   Max.   :8.00
>
>
> dados$Riqueza
 [1] 35 31 39 25 22 35 43 48 35 38 32 33 40 16 31 23 32 35 22 41 24 18 39 32 19
[26] 20 31 26 20 33 36 32 28 23 28 16 44 27 37 24 19 28 27 28 27 20 34 20 31 14
>
> Riqueza
Erro: objeto 'Riqueza' não encontrado
>
> attach(dados)
>
> Riqueza
 [1] 35 31 39 25 22 35 43 48 35 38 32 33 40 16 31 23 32 35 22 41 24 18 39 32 19
[26] 20 31 26 20 33 36 32 28 23 28 16 44 27 37 24 19 28 27 28 27 20 34 20 31 14
>
> detach(dados)
>
> Riqueza
Erro: objeto 'Riqueza' não encontrado
>
> |

```

Sem o `attach()`, a variável não é reconhecida.

O `detach()` desfaz o que o `attach()` fez.

## 2.2. Medidas de tendência central e de dispersão

Na primeira parte deste guia, vimos que o comando `summary()` gera algumas medidas descritivas sobre os nossos dados: a média, a mediana, os quartis e os valores máximos e mínimos. Estas mesmas métricas podem ser calculadas individualmente, bem como outras que podem ser úteis na descrição de conjuntos de dados.

```

RGui (64-bit)
Arquivo Editar Visualizar Misc Pacotes Janelas Ajuda
[Ícones]

R Console
> mean(Riqueza)
 [1] 29.22
>
> sd(Riqueza)
 [1] 8.031291
>
> var(Riqueza)
 [1] 64.50163
>
> quantile(Riqueza)
 0% 25% 50% 75% 100%
14.0 23.0 29.5 35.0 48.0
>
> #Calculando apenas mediana:
> quantile(Riqueza,0.5)
 50%
29.5
>
> #Calculando percentis 10, 50 e 90:
> quantile(Riqueza,c(0.1,0.5,0.9))
 10% 50% 90%
19.0 29.5 39.1

```

Aqui estabelecemos os percentis que desejamos, usando decimais.

As funções *mean()*, *var()* e *sd()* calculam, respectivamente, a média, a variância e o desvio padrão. A função *quantile()* permite calcular a mediana, os quartis e quaisquer percentis desejados (veja o exemplo acima para entender o uso desta última função).

### 2.3. Lidando com subconjunto de dados

O conjunto de dados que apareceu nos nossos exemplos até agora possui a variável categórica “Ambiente”, que possui duas classes (que também podemos chamar de níveis). Dado o contexto deste conjunto de dados, uma pergunta óbvia que poderia ser feita é: há alguma diferença na riqueza média de espécies entre os dois tipos de ambiente? Nós trataremos desta pergunta diretamente quando realizarmos um teste T, mas neste momento seria interessante aprendermos como calcular as medidas acima (como média e desvio, por exemplo) de maneira restrita, de acordo com a nossa variável categórica. Para fazer isso, vamos usar colchetes para criar condições no R. A sintaxe básica disso seria algo como `variável[condição]`, onde sinais como ‘>’, ‘<’ ou ‘==’ podem ser usados para criar as condições desejadas. Lembre-se de que a igualdade deve ser sempre representada com dois sinais de igual, e que os níveis de uma variável categórica devem sempre ser representados entre aspas. Os exemplos a seguir devem ser o suficiente para se aprender como usar este mecanismo:

```

RGui (64-bit)
Arquivo  Editar  Visualizar  Misc  Pacotes  Janelas  Ajuda

R Console
> mean(Riqueza[Ambiente=="secundário"])
[1] 27.25
> mean(Riqueza[Ambiente=="primário"])
[1] 31.72727
> Riqueza[Riqueza>=30]
[1] 35 31 39 35 43 48 35 38 32 33 40 31 32 35 41 39 32 31 33 36 32 44 37 34 31
> mean(Riqueza[Área>100])
[1] 33.38462
> mean(Abund_sp1[Área<=70])
[1] 2.8
> |
  
```

Resultados usando apenas dados com Ambiente da classe "secundário".

Resultados usando apenas dados com Área maior do que 100.

Legal. Mas e se eu quiser usar mais de uma condição de uma vez? Ou ainda, e se eu quiser usar duas condições alternativas?! O R entende ‘&’ como ‘e’, e entende ‘|’

como ‘ou’. Confuso? Veja os exemplos a seguir, pois na verdade é bem simples e prático.

RGui (64-bit)

Arquivo Editar Visualizar Misc Pacotes Janelas Ajuda

R Console

```
>
>
>
> mean(Riqueza[Área>100 | Abund_sp1>=2])
[1] 30.04762
>
> mean(Riqueza[Área>100 & Ambiente=="secundário"])
[1] 30.14286
>
> |
```

Riqueza média das unidades com área maior do que 100 ou com abundância da espécie 1 maior ou igual a 2.

Riqueza média para ambientes da classe "secundário" e com área maior do que 100.

### 2.3.1. A função *tapply* (e por que nós a amamos!)

Legal, acabamos de ver como eu posso calcular, por exemplo, a média de riqueza por ambiente. No exemplo que estamos usando, temos apenas duas categorias de ambiente: primário e secundário. E se eu tivesse cinco categorias? Ou 12? Ou 37?! Bom, eu iria desconfiar seriamente da sua sanidade mental ao usar uma variável com 37 variáveis, mas vamos lá. A questão, neste caso, é que daria um trabalhão repetir o comando um monte de vezes, para cada classe da nossa variável categórica. É aí que entra a função *tapply*()).

O R, como linguagem de programação, permitiria que eu escrevesse uma função para automatizar este procedimento, usando um tal de *loop*. É uma coisa muito legal, mas neste momento inicial podemos usar uma versão “pré-programada”, que está na função *tapply*()). Esta funçãozinha mágica faz o seguinte: aplica uma função qualquer (como média, por exemplo) a uma variável quantitativa para cada classe de uma variável categórica. Em outras palavras: no lugar de calcular cada média de uma vez, eu faço tudo isso em um comando só. A sintaxe é fácil: *tapply*(var\_quantitativa, var\_categórica, função\_desejada). Veja o exemplo:

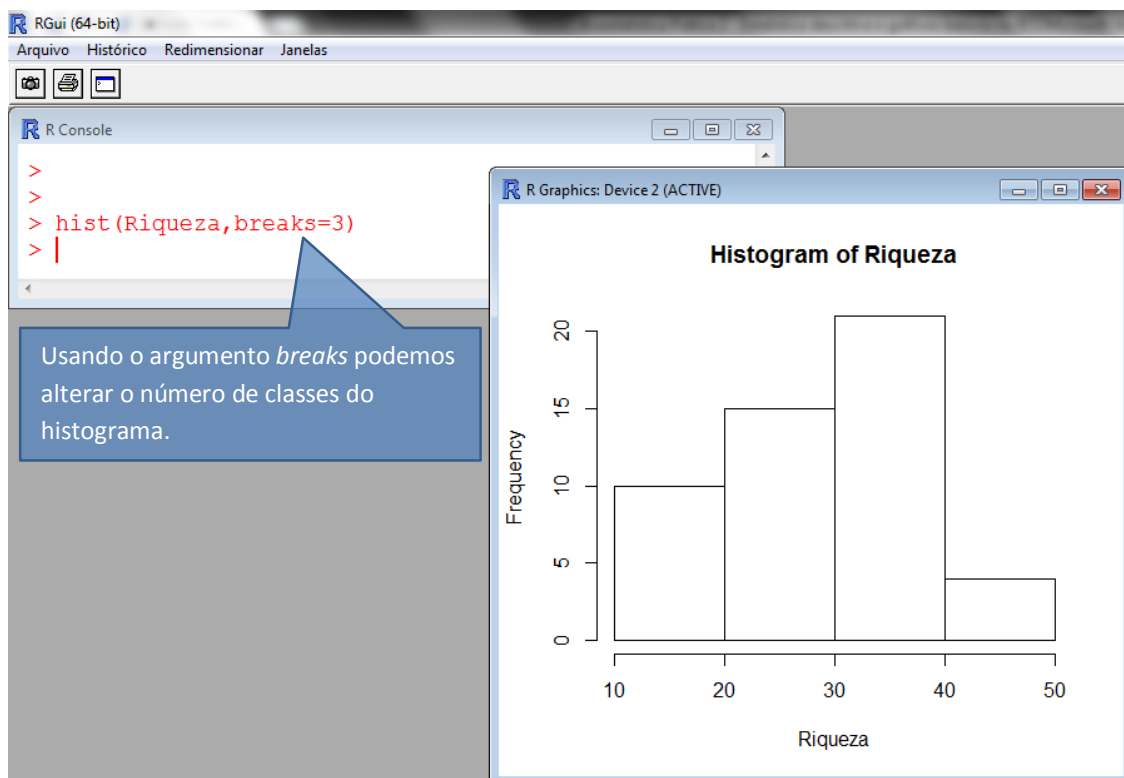
```
>
>
> tapply(Riqueza, Ambiente, mean)
primário secundário
31.72727 27.25000
>
> tapply(Riqueza, Ambiente, sd)
primário secundário
8.424351 7.260369
> |
```

Aqui pedi a média.

E nesta pedi o desvio padrão.

## 2.4. Estudando a distribuição de frequências de uma variável: o histograma

Os histogramas são gráficos que representam a distribuição de frequências dos valores de uma variável quantitativa. É uma excelente maneira de se começar a explorar um conjunto de dados, e criá-los no R é bastante simples: basta usar a função `hist()`. No uso básico da função o próprio R irá “decidir” em quantas classes dividir os intervalos da variável, o que pode ser alterado pelo argumento `breaks`. Tome algum tempo para explorar o help da função, dando especial atenção para os exemplos.



## 2.5. Gráficos, gráficos, gráficos!

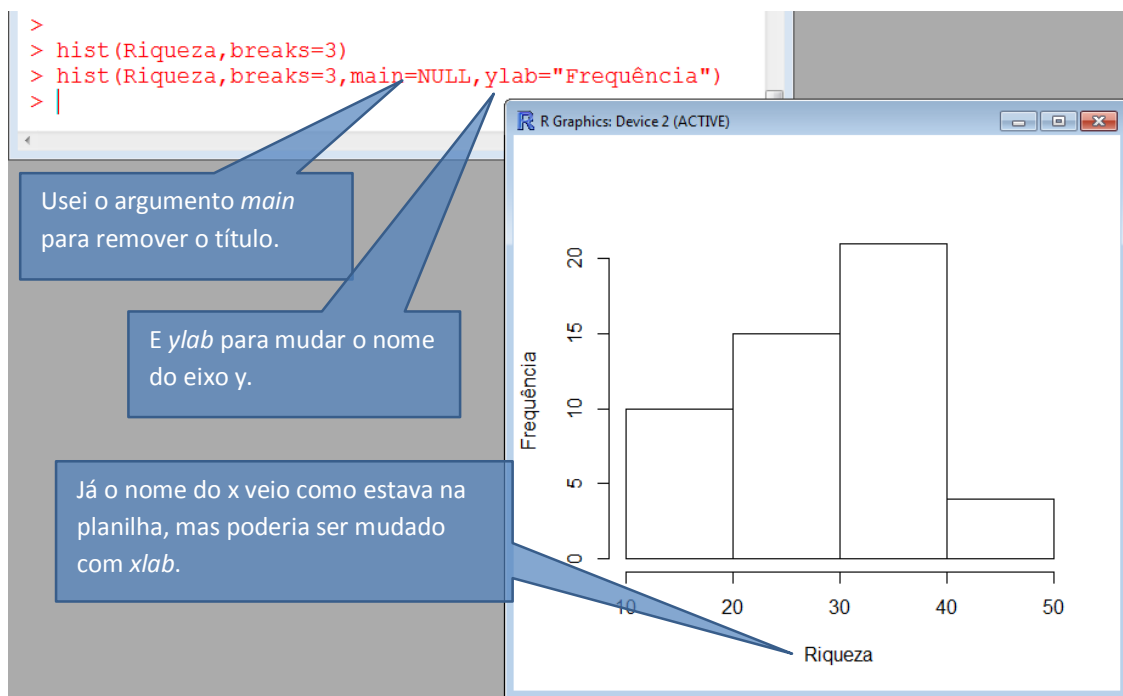
Se esta apostila representa o seu primeiro contato com o R, então parabéns: você acaba de criar o seu primeiro gráfico neste programa! Aêêê! Vamos fazer mais? Bom, este material **não** tem por objetivo explorar muito as funcionalidades gráficas do R (que são imensas, por sinal), mas é conveniente chamar a atenção para alguns detalhes.

Começando do começo: você deve ter notado que o R abriu uma nova janela quando você executou o comando `hist()` para fazer o gráfico. Se você fechar a janela e executar o comando novamente (ou executar um novo comando gráfico), uma nova janela se abrirá. Se, por outro lado, você executar um novo comando sem fechar a janela, o novo gráfico irá sobrescrever o anterior (e, neste caso, a janela não irá “pular”

para a frente, e continuará em segundo plano ou minimizada). Caso deseje salvar o gráfico, clique com o botão direito do mouse sobre a imagem ou, com a janela em primeiro plano, vá ao menu 'Arquivo' e explore as opções disponíveis.

Existem muitos argumentos comuns a todas as funções gráficas do R, como *xlab* e *ylab* (para dar nomes aos eixos), *main* (para atribuir títulos ao gráfico), *axes* (para se adicionar ou remover os eixos) e muitas outras. A documentação de ajuda da função *par()* é uma excelente fonte de informações sobre argumentos gerais que podem ser usados em diversos gráficos diferentes. Ao explorar as funções gráficas do R, use e abuse dos exemplos que aparecem nas páginas de ajuda, pois eles são uma das melhores maneiras de se aprender. Caso esteja com dificuldades, tente criar seu gráfico passo à passo: crie um gráfico básico e depois vá acrescentando, um a um, os argumentos desejados, até chegar ao ponto ideal.

A partir deste ponto, vou considerar que você, leitor, já é crescidinho, e vai ser capaz de fuçar um pouco as coisas sem a minha ajuda. Então vou adicionar, nos exemplos, alguns argumentos que você deverá explorar, mesmo quando eu não der muitas informações detalhadas sobre eles, ok? Coragem! Mexer em argumentos (e errar muitas vezes!) faz parte da experiência de aprendizado no R. Comece vendo o exemplo a seguir:

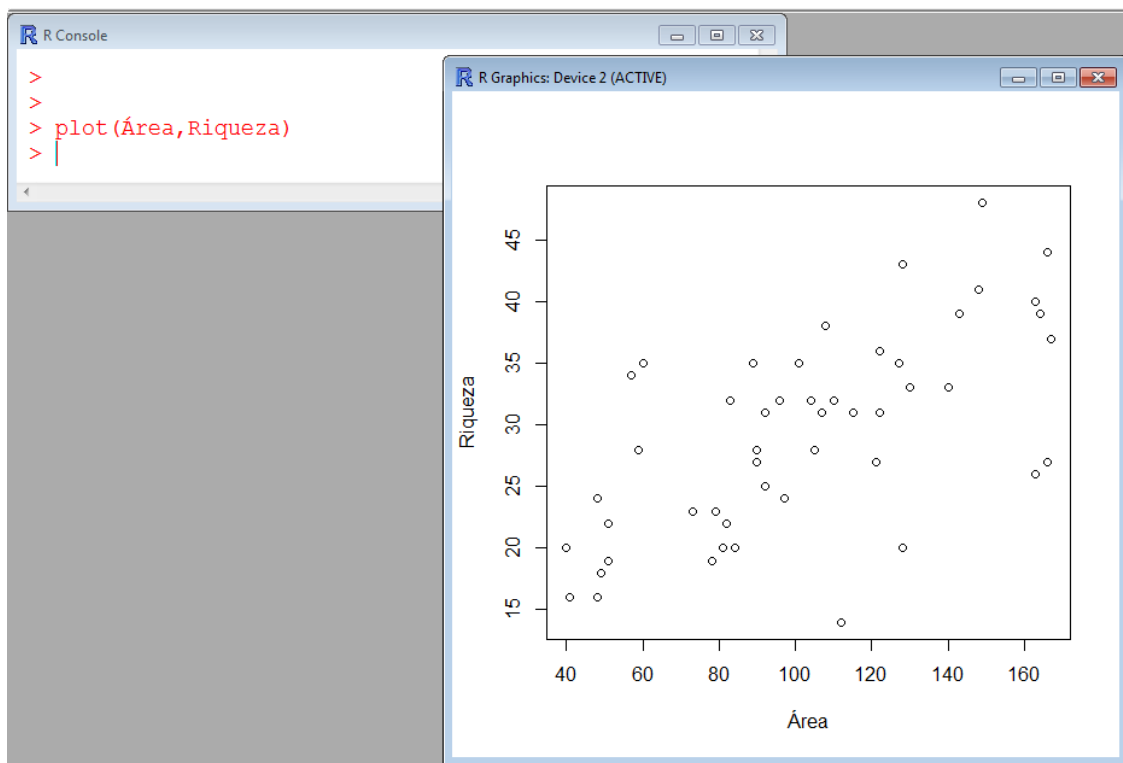


Vamos olhar agora, bem brevemente, para outros gráficos típicos da estatística. Estes são gráficos que mostram a relação entre duas variáveis, situação na qual existe

uma convenção geral: sempre que existir uma variável explicativa (ou independente), ela deve ser representada no eixo x, enquanto a variável resposta (ou dependente) deve ser representada no y. A exceção, claro, são os gráficos de barras, nos quais o eixo y representa apenas frequências, de maneira similar aos histogramas. Alguns destes gráficos deles serão vistos novamente mais adiante, quando lidarmos com os testes estatísticos, e na ocasião poderemos ver mais detalhes e argumentos conforme necessário.

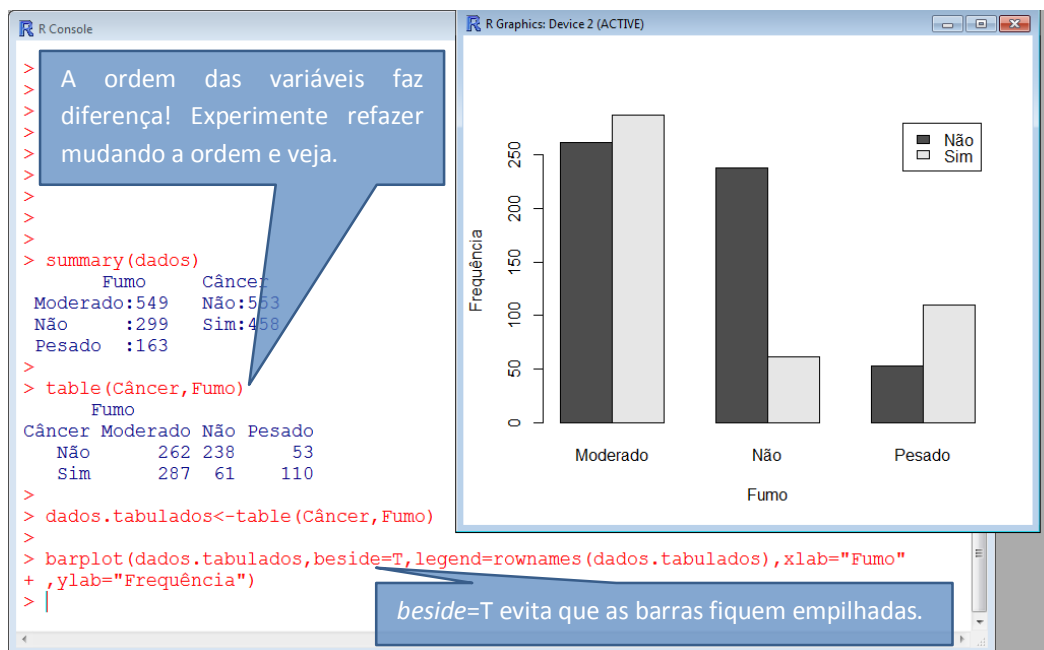
### 2.5.1. Gráficos de dispersão

Os gráficos de dispersão representam a relação entre duas variáveis quantitativas. Criá-los no R é extremamente simples: basta usar a função `plot()`. A sintaxe é simples: `plot(eixo_x, eixo_y)`. Veja o exemplo, e depois reveja este tipo de gráfico quando falarmos de regressão linear. Ah, se você não gostar muito das bolinhas vazias, experimente usar o argumento `lty` e veja o que pode ser feito. `lty=16` é uma boa pedida.



### 2.5.2. Gráficos de barras

Os gráficos de barras podem ser usados para representar uma ou duas variáveis categóricas, e são construídos no R pela função `barplot()`. Em planilhas de dados comuns, nas quais as linhas são as unidades amostrais e as colunas variáveis, a função `table()` é um intermediário importante para chegarmos ao gráfico desejado. Veja o exemplo a seguir para aprender o funcionamento básico da função (percebam que usei o argumento `legend` para identificar as cores das colunas).



### 2.5.3. Boxplots e gráficos de média com barras de erro

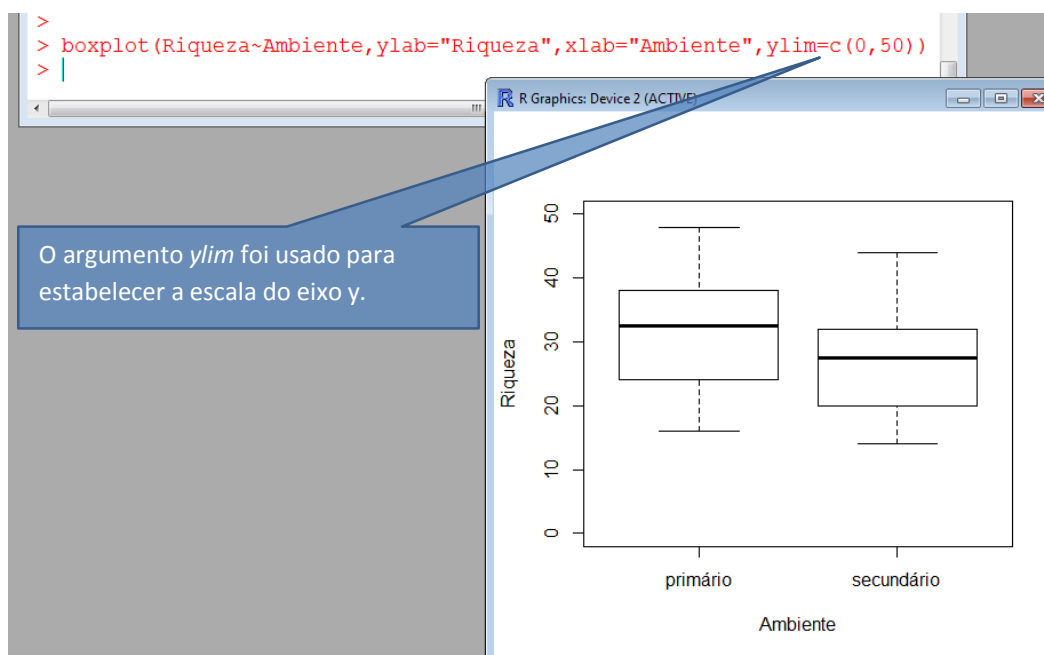
Quando desejamos comparar valores de uma variável quantitativa de acordo com as classes determinadas por uma variável categórica, podemos usar gráficos que representem alguma medida de tendência central acompanhada de uma ou mais medidas de variação e/ou erro. A escolha de uma medida ou outra depende da natureza dos dados: a média, por exemplo, é uma medida que não faz sentido em dados com uma distribuição de frequência muito assimétrica.

#### Boxplots:

Uma das maneiras mais comuns de se representar medianas, normalmente acompanhadas de quartis e/ou percentis, são os *boxplots*. Criá-los no R é feito com a função `boxplot()`, como no exemplo a seguir (no qual usei alguns argumentos a mais para modificar o gráfico básico). Para usar este comando, a relação entre as variáveis foi determinada usando o símbolo `'~'`. Esta representação é bem comum no R, e irá



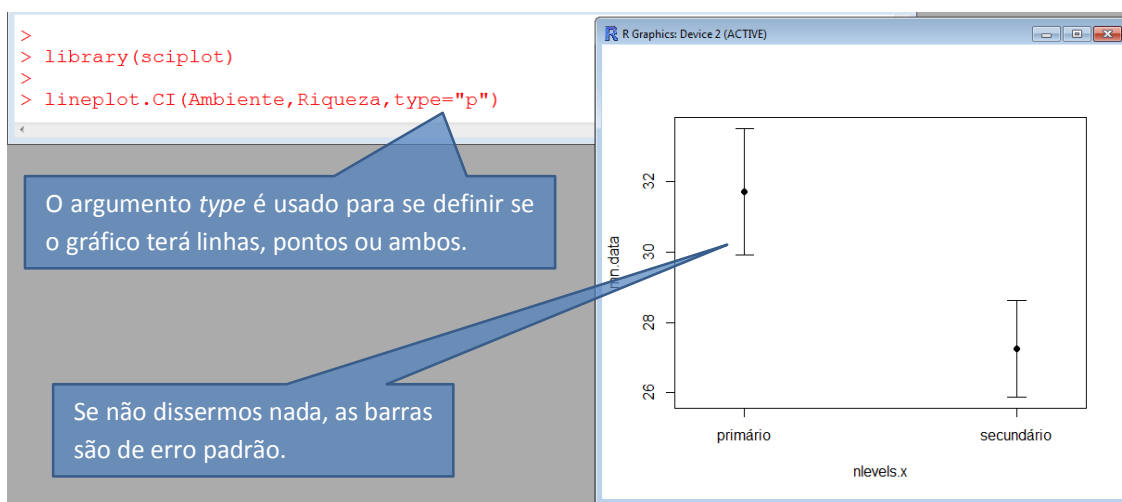
aparecer no uso de diversos modelos estatísticos. A lógica desta representação será sempre ‘variável\_resposta~variável\_explicativa’.



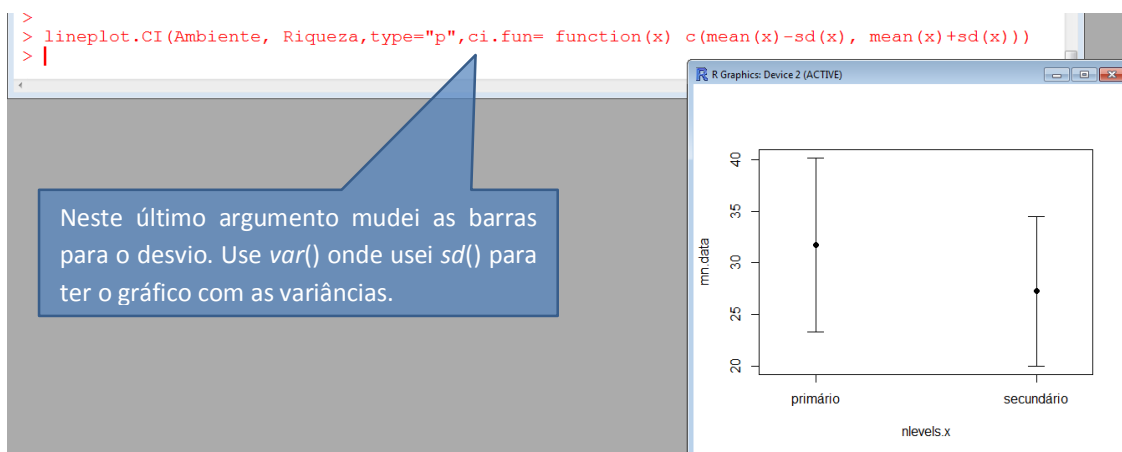
#### *Médias e barras de desvio ou erro:*

Para representarmos médias e alguma medida de desvio ou de erro, o mais comum é usarmos um gráfico no qual a média é representada por um ponto (ou barra) e o desvio ou erro por uma barra de variação em torno da média. Não há função básica no R para conseguir isto diretamente, então temos que apelar para algum pacote que facilite este processo.

A opção que é possivelmente a mais simples é utilizar a função *lineplot.CI()* do pacote *sciplot*. Esta função cria um gráfico visualmente muito bom (e o argumento *type* permite usar apenas os pontos, sem as linhas). O seu uso básico trabalha apenas com barras que representem o erro padrão, mas é possível alterá-la para usarmos outras medidas (veja um dos exemplos, onde usei desvio).



Agora refazendo, mas desta vez representando o desvio padrão nas barras. A função fica mais longa, mas não é nada absurdo. Veja o help da função se quiser entender melhor o que eu fiz aqui:



## 2.6. Indo além dos gráficos básicos

O que fizemos até agora foi usar os gráficos básicos do R, que são, digamos, os “gráficos de fábrica”. Existem outras opções bem interessantes, que são bem amplas e constituem quase que um mundo à parte, pois usam argumentos e comandos completamente diferentes do que vimos até aqui. Não vou explorá-los, mas recomendo que os usuários interessados deem uma olhada nos pacotes *lattice* e *ggplot2*. Ambos estão disponíveis no CRAN, e possuem um monte de material: livros, apostilas, blogs e coisas do tipo. Faça uma busca sobre eles no *google* e dê uma espiada para ver o tipo de coisa que eles podem fazer. Se ficar interessado, instale os pacotes e comece executando os exemplos, pois será uma boa maneira de entender como eles funcionam.

## 2.7. Funções utilizadas – capítulo 2

Vou resumir, a seguir, as funções que vimos nesta seção.

*data.frame()*, *edit()*, *fix()* → funções para se editar planilhas no R

*\$* → permite visualizar parte de um objeto (exemplo *planilha\$variável*)

*attach()* → faz o R reconhecer os nomes das variáveis de uma planilha

*detach()* → desfaz o *attach()*

*mean()* → calcula a média aritmética

*var()* → calcula a variância amostral

*sd()* → calcula o desvio padrão amostral

*quantile()* → calcula percentis

*[]* → permite criar condições/restrições ao se ver ou manipular um objeto ou variável

'&' e '|' → símbolos para “e” e “ou”, respectivamente

*hist()* → histogramas

*par()* → parâmetros gráficos gerais

*plot()* → gráficos de dispersão

*barplot()* → gráficos de barra

*table()* → tabelas de contingência

*boxplot()* → boxplots

*lineplot.CI()* → gráficos de média (pacote *sciplot*)

### 3. ALGUNS TESTES ESTATÍSTICOS

Beleza! Você agora é uma pessoa quase íntima do R (ui!) e daqui para frente só precisa de prática, prática e mais um pouco de prática. Quanto mais atividades aprender a realizar no R, mais sua linguagem ficará familiar, e mais fácil será usá-lo. Neste capítulo vamos aprender a realizar algumas análises estatísticas que são bastante úteis no dia a dia de um biólogo. Serão quatro testes: qui-quadrado, teste T, ANOVA e a regressão linear simples. Aprendê-los, além de útil do ponto de vista prático, servirá de base para que você possa dominar outros testes no programa.

Neste ponto é bom lembrar uma coisa: este não é um guia para se aprender estatística! O objetivo deste material é aprender a dominar o R, e somente isso. Caso você consiga fazer um teste mas não saiba bem o que está fazendo, pare tudo e vá estudar, ok? Executar um teste estatístico que não conhecemos pode ser bem fácil na prática, mas as chances de chegarmos a conclusões completamente equivocadas (leia-se: as chances de fazer cagada!) ficam muito grandes

#### 3.1. A escolha de um teste estatístico

A escolha de um teste é definida, em parte, pela natureza das variáveis estudadas e a sua (presumida) relação. O quadro a seguir é um resumo um pouco grosseiro de como é feita esta escolha:

		Var. explicativa	
		Categórica	Quantitativa
Var. resposta	Categórica	Qui-quadrado	Regressão logística
	Quantitativa	Teste T / ANOVA	Regressão linear

Com exceção do teste de qui-quadrado, os demais três testes que veremos são ditos paramétricos, e dependem de alguns pressupostos para que seus resultados sejam considerados confiáveis. Para simplificar um pouco as coisas, falarei sobre eles apenas ao final do capítulo, após termos visto todos os testes.

#### 3.2. O teste de qui-quadrado

Uma possível associação entre duas variáveis categóricas pode ser facilmente visualizada se organizarmos os dados em uma tabela de contingência, que mostra a contagem do número de vezes que cada par de categorias apareceram juntos. No R, podemos associar as duas variáveis categóricas com a função `table()` (que vimos na prática anterior ao construirmos um gráfico de barras) e, em seguida, testar a

significância desta associação pelo teste de Qui-quadrado com a função `chisq.test()`. A função mostra, como resultado, o valor de qui-quadrado, os graus de liberdade e o valor de  $p$ , nesta ordem.

```
> summary(dados)
      Fumo   Câncer
Moderado:549 Não:553
Não      :299  Sim:458
Pesado   :163

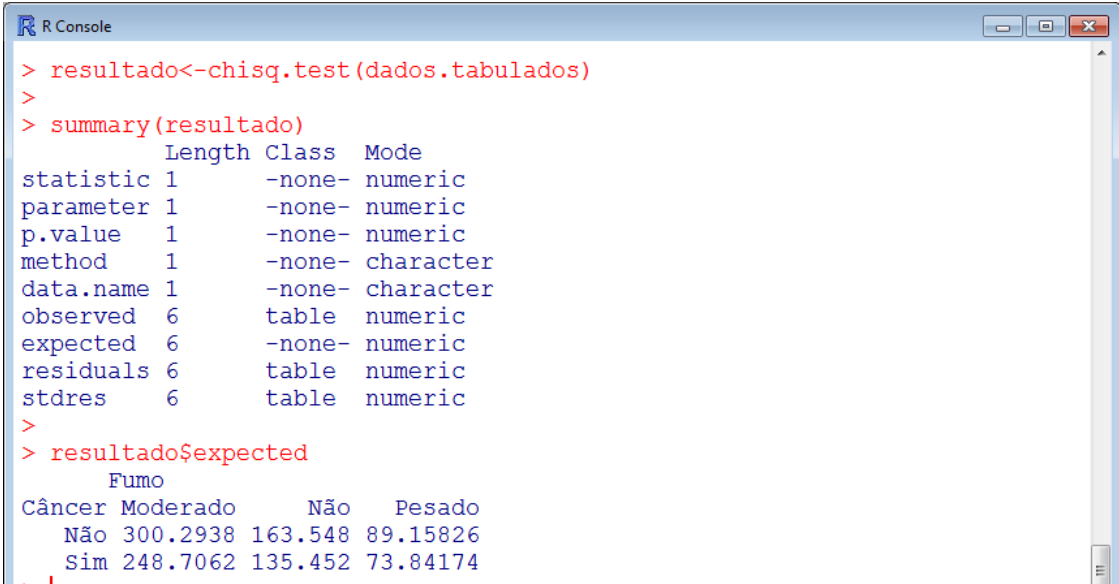
>
> dados.tabulados<-table(Câncer,Fumo)
>
> chisq.test(dados.tabulados)

      Pearson's Chi-squared test

data:  dados.tabulados
X-squared = 117.965, df = 2, p-value < 2.2e-16
```

Em muitos casos (e este não é uma exceção), é interessante associar o resultado de um teste a um novo objeto, pois isto permite explorar resultados que não seriam mostrados com a simples execução da função. No caso do teste de qui-quadrado, podemos desejar ver, por exemplo, os valores das frequências esperadas por acaso. Se associarmos o resultado do teste a um objeto e o inspecionarmos com a função `summary()`, notaremos que existem vários componentes, e que um deles contém estes valores: o componente *expected*. Observe, no exemplo a seguir, como podemos usar o símbolo '\$' para ter acesso a estes valores (eu disse que ele seria útil!).

Por fim, para compreendermos o resultado do teste, é essencial investigarmos os dados em si, o que normalmente é feito de forma gráfica. Veja o exemplo de gráfico de barras na apostila anterior.



```
R Console
> resultado<-chisq.test(dados.tabulados)
>
> summary(resultado)
      Length Class  Mode
statistic 1      -none- numeric
parameter 1      -none- numeric
p.value    1      -none- numeric
method     1      -none- character
data.name  1      -none- character
observed   6      table  numeric
expected   6      -none- numeric
residuals  6      table  numeric
stdres     6      table  numeric
>
> resultado$expected
      Fumo
Câncer Moderado   Não   Pesado
Não 300.2938 163.548 89.15826
Sim 248.7062 135.452 73.84174
~ |
```

### 3.3. O teste T de Student

O teste T possui três variantes básicas: a mais usada é normalmente chamada de teste t para amostras independentes, e permite a comparação de duas médias; o teste t para uma amostra compara uma média com um valor fixo; e o teste t pareado (ou teste t para amostras dependentes) compara médias de unidades amostrais dependentes aos pares. Todas estas variações do teste t podem ser executadas no R pela função `t.test()`, e o uso do teste t pareado e do teste t para uma amostra pode ser ativado com o uso dos argumentos `paired` e `mu`, respectivamente, como pode ser visto nos exemplos a seguir.

Usualmente, o resultado de um teste t vem acompanhado de uma representação gráfica. Veja o exemplo de gráfico de médias com barras de erro na apostila anterior.

Independente da variante de teste T realizado, a apresentação do seu resultado no R sempre começará com os valores de t, de graus de liberdade e de p, nesta ordem. Veja:

```
data: Riqueza by Ambiente
t = 2.0171, df = 48, p-value = 0.0493
```

O teste T “comum”, visto em qualquer livro básico de estatística, presume variâncias homogêneas. No R, isto deve ser informado com o argumento `var.equal=T`. Caso isto não seja feito, o teste realizado será para variâncias heterogêneas. Fique muito atento para isso, pois o teste para variâncias heterogêneas consome mais graus de liberdade, e só deveria ser usado quando for essencial.

```
> summary(dados)
  UA      Ambiente      Área      Riqueza      Abund_spl
Min. : 1.00  primário :22  Min. : 40.0  Min. :14.00  Min. :0.00
1st Qu.:13.25  secundário:28  1st Qu.: 79.5  1st Qu.:23.00  1st Qu.:1.00
Median :25.50                Median :102.5  Median :29.50  Median :2.00
Mean   :25.50                Mean   :102.5  Mean   :29.22  Mean   :2.16
3rd Qu.:37.75                3rd Qu.:127.8  3rd Qu.:35.00  3rd Qu.:3.00
Max.   :50.00                Max.   :167.0  Max.   :48.00  Max.   :8.00
>
> #Teste t para amostras duas independentes.
> t.test(Riqueza~Ambiente,var.equal=T)
Two Sample t-test

data: Riqueza by Ambiente
t = 2.0171, df = 48, p-value = 0.0493
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.01431068 8.94023478
sample estimates:
 mean in group primário mean in group secundário
          31.72727          27.25000
```

Com `var.equal=T`, informamos que presumimos variâncias homogêneas. Este é o teste T mais comum.

No teste T para uma amostra (bem pouco usado na biologia) comparamos uma média com um valor de referência qualquer:

```
> #Agora um teste t para uma amostra, comparando a riqueza média com o valor 30
>
> t.test(Riqueza,var.equal=T,mu=30)
```

One Sample t-test

data: Riqueza  
t = -0.6867, df = 49, p-value = 0.4955  
alternative hypothesis: true mean is not equal to 30  
95 percent confidence interval:  
26.93753 31.50247  
sample estimates:  
mean of x  
29.22

O argumento *mu* permite o teste t para uma amostra.

O teste T pareado é feito no R a partir de duas colunas, uma vez que duas medidas são feitas em uma mesma unidade amostral:

```
> summary(dados)
  medidal   medida2
Min.   :20.00  Min.   :21.0
1st Qu.:24.25  1st Qu.:27.5
Median :26.50  Median :29.5
Mean   :26.90  Mean   :29.8
3rd Qu.:30.00  3rd Qu.:31.5
Max.   :34.00  Max.   :41.0
>
> #Agora um teste t para amostras dependentes (ou pareadas)
> #Note que neste caso são duas colunas com valores, pois as medidas são
> #dependentes das unidades amostrais (que estão nas linhas).
>
> t.test(medidal,medida2,var.equal=T,paired=T)
```

Paired t-test

data: medidal and medida2  
t = -2.6364, df = 9, p-value = 0.02707  
alternative hypothesis: true difference in means is not equal to 0  
95 percent confidence interval:  
-5.3883729 -0.4116271  
sample estimates:  
mean of the differences  
-2.9

O argumento *paired* define o teste como pareado.

### 3.3.1. Caudalidade do teste T

E um teste de hipótese no qual a sua predição tenha uma “direção” (ou seja, você tem uma expectativa prévia de qual média é maior), é importantíssimo definir a caudalidade do teste. Isto é feito no teste T do R com o argumento *alternative*:

```
>
> t.test(Riqueza~Ambiente, var.equal=T, alternative="greater")
```

Two Sample t-test

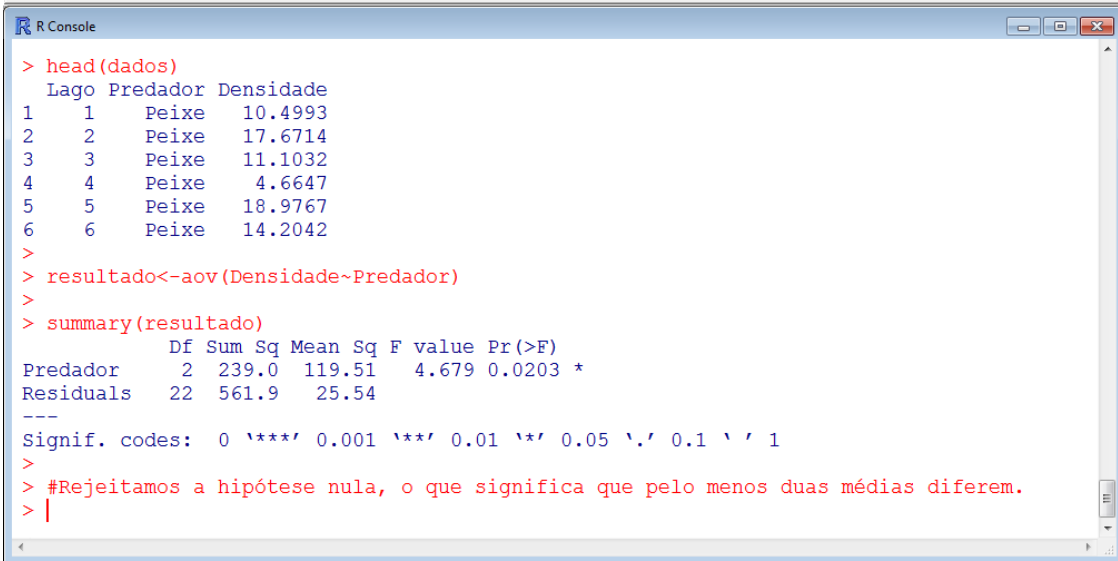
data: Riqueza by Ambiente  
t = 2.0171, df = 48, p-value = 0.02465  
alternative hypothesis: true difference in means is greater than 0  
95 percent confidence interval:  
0.7543748 Inf  
sample estimates:  
mean in group primário mean in group secundário  
31.72727 27.25000

O argumento *alternative* definiu o teste como uni-caudal.

Caso você esteja realizando um teste T sem definir a caudalidade a priori, pense duas vezes. Definir isto é interessante tanto de um ponto de vista científico (a hipótese normalmente será mais clara quando estabelecemos uma direção) quanto do ponto de vista estatístico (aumenta o poder do teste).

### 3.4. A análise de variância (ANOVA)

A ANOVA é uma grande família de testes, e neste guia veremos a sua versão mais simples, normalmente chamada de *one-way* ANOVA. Assim como o teste t, este é um teste para comparação de médias, com a diferença de que na ANOVA nós vamos comparar três ou mais classes da variável categórica (ou níveis do fator, na terminologia específica desta análise). A ANOVA faz parte de uma família ainda maior de análises, que chamamos de modelos lineares. O R lida com todos os modelos lineares de maneira similar, e um detalhe importante é que todas as análises desta família devem ser salvas em um objeto, para seu resultado ser apresentado após o comando `summary()`. No caso da ANOVA, a função para realizar a análise é `aov()`. Veja o exemplo a seguir para compreender seu uso:



```

> head(dados)
  Lago Predador Densidade
1    1    Peixe  10.4993
2    2    Peixe  17.6714
3    3    Peixe  11.1032
4    4    Peixe   4.6647
5    5    Peixe  18.9767
6    6    Peixe  14.2042
>
> resultado<-aov(Densidade~Predador)
>
> summary(resultado)
              Df Sum Sq Mean Sq F value Pr(>F)
Predador      2  239.0   119.51   4.679 0.0203 *
Residuals    22   561.9    25.54
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> #Rejeitamos a hipótese nula, o que significa que pelo menos duas médias diferem.
> |

```

Perceba que a hipótese nula da ANOVA é de que as médias não diferem. Ao rejeitá-la, então, nós apenas sabemos que pelo menos duas das médias são diferentes entre si, mas não sabemos quais são. Uma das formas mais comuns de saber isso (mas não a única) são os testes *a posteriori*, como o teste de Tukey, chamado pela função `TukeyHSD()`, que atua diretamente no objeto com o resultado da ANOVA:



```

> TukeyHSD(resultado)
Tukey multiple comparisons of means
 95% family-wise confidence level

Fit: aov(formula = Densidade ~ Predador)

$Predador
      diff      lwr      upr    p adj
Nenhum-Libélula -7.18221 -13.204276 -1.160144 0.0175643
Peixe-Libélula  -2.59820  -9.168813  3.972413 0.5886784
Peixe-Nenhum    4.58401  -1.672465 10.840485 0.1799912

>
> #Então a diferença está entre as classes "Nenhum" e "Libélula".

```

Por fim, a interpretação final do resultado de uma ANOVA costuma ser acompanhada por uma representação gráfica. Assim como no teste t, veja na apostila anterior o exemplo de gráfico de médias com barras de erro.

### 3.5. A regressão linear simples

A regressão linear simples testa a existência de uma relação linear de causa e efeito entre uma variável explicativa e outra resposta, ambas quantitativas. Também faz parte da família dos modelos lineares. No R, pode ser chamado pelo comando *lm()*. Veja o exemplo a seguir:

```

> resultado<-lm(Riqueza~Área)
>
> summary(resultado)

Call:
lm(formula = Riqueza ~ Área)

Residuals:
    Min       1Q   Median       3Q      Max
-16.5419  -3.6136  -0.0566   3.9721  12.3206

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 14.99044    2.63204   5.695 7.30e-07 ***
Área         0.13885    0.02419   5.739 6.25e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.249 on 48 degrees of freedom
Multiple R-squared:  0.407,    Adjusted R-squared:  0.3946
F-statistic: 32.94 on 1 and 48 DF,  p-value: 6.253e-07

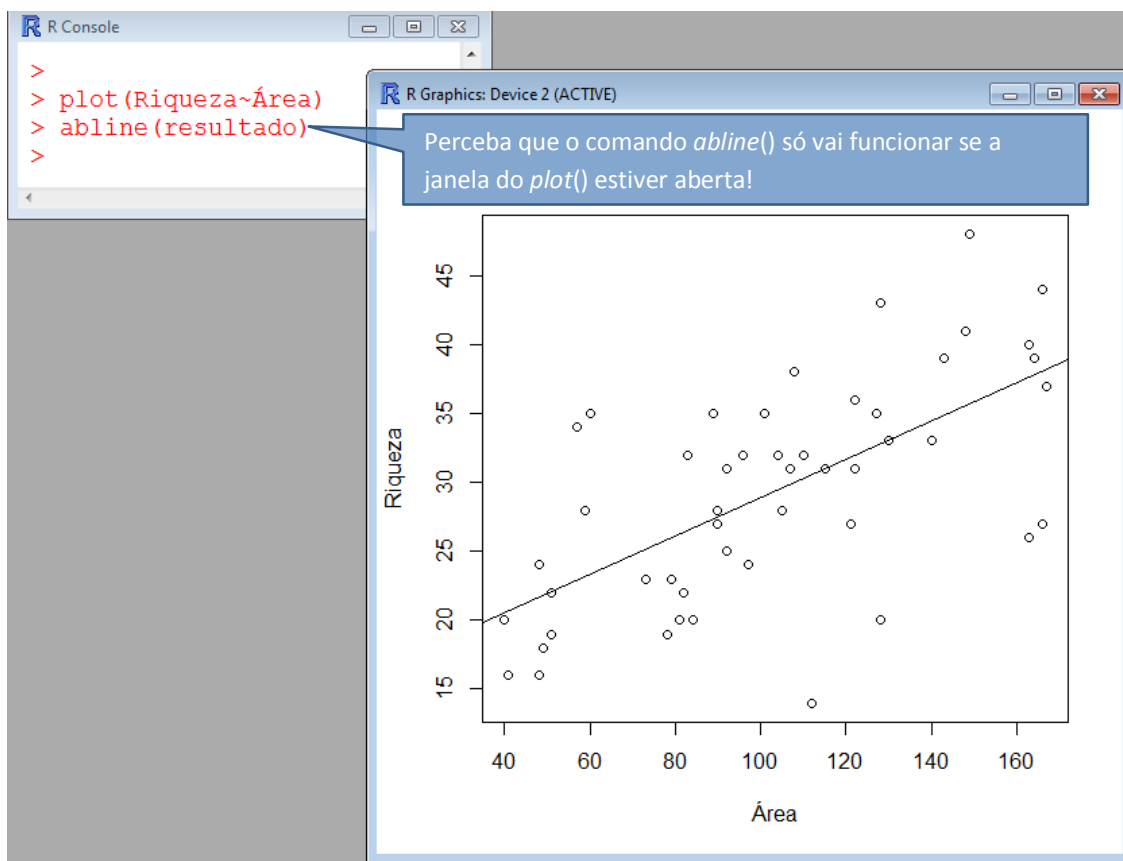
>
> |

```

Os resultados são parecidos com os de uma ANOVA, mas conta com alguns detalhes a mais. No exemplo acima, a linha (Intercept) refere-se ao intercepto da equação linear (o  $b$ , na equação  $y = ax + b$ ), e verifica a hipótese nula de que ele seria igual a zero; de uma forma geral, nós raramente nos preocupamos com a esta parte do resultado, pois o intercepto normalmente não faz parte da nossa pergunta. Já a linha

abaixo, com o nome da variável explicativa, é sempre do nosso interesse: ela verifica a hipótese nula de que o parâmetro  $a$  da equação seja igual a zero; ou seja, ao rejeitarmos a hipótese nula nós concluímos que de fato há uma relação estatisticamente significativa entre as duas variáveis, e que o efeito desta relação é medido por este parâmetro. Por fim, a interpretação da correlação, que está representada no parâmetro *Adjusted R-squared* também é importante, e mede o quanto da variação na variável resposta foi de fato explicada pela variável explicativa. Confuso? Então vejamos os resultados do exemplo acima: o teste rejeita a hipótese nula de que a riqueza de espécies não varia em função da área do fragmento de mata (o valor de  $p$  foi pequenininho, está apresentado em potência de dez:  $6,25 \cdot 10^{-6}$ ). O resultado da regressão permite concluirmos que, em média, cada incremento de uma unidade de área aumenta a riqueza em 0,13885 (este é o valor de  $a$  da equação). Por fim, o valor de  $R^2$  ajustado nos diz que 39,46% da riqueza de espécies é explicada pelo tamanho da área dos fragmentos.

Os resultados de uma regressão linear que rejeita a hipótese nula são comumente apresentados com uma representação gráfica que mostra os dados (em um gráfico de dispersão) e a equação (na forma de uma reta). Para adicionar a reta da equação ao gráfico criado pelo comando `plot()`, basta, com o gráfico aberto, executar o comando `abline()`, fazendo referência ao objeto que contém o resultado da regressão:



### 3.6. Pressupostos dos testes estatísticos

Todos os testes estatísticos possuem pressupostos, que são coisas que devem ser verdade para que o resultado do teste seja confiável. Alguns destes pressupostos estão ligados ao delineamento experimental: por exemplo, a maioria dos testes pressupõe que as suas unidades amostrais sejam independentes. Mas também existem pressupostos específicos, que podem ser verificados nos dados em si, e é sobre eles que falaremos brevemente a seguir. Meu objetivo aqui é tratar deles apenas rapidamente para fins práticos, mas lembre-se de que verificar os pressupostos rotineiramente é importantíssimo nos testes estatísticos, ok? Um bom livro de estatística sempre vai mencionar os pressupostos, então basta consultar um caso você fique em dúvida. E se o livro não falar nada sobre o assunto? Bom, provavelmente ele não é um bom livro de estatística...

#### 3.6.1. Pressupostos do qui-quadrado

O teste de qui-quadrado não possui pressupostos “testáveis” como os que veremos para os demais testes, então basta ficar atento para um pressuposto bem simples: devemos observar se a tabela de contingência possui no máximo 20% dos seus valores menores do que cinco. Se este pressuposto for violado, o resultado do teste não é confiável.

#### 3.6.2. Pressupostos do teste $t$

O teste  $t$  possui dois pressupostos básicos sobre a variável quantitativa: que as suas variâncias sejam homogêneas e que os dados não difiram significativamente da distribuição normal. Ambos podem ser testados rapidamente no R, pelos testes de Levene e de Shapiro-Wilk, respectivamente. O teste de Levene para homogeneidade de variância pode ser feito com o comando `leveneTest()`, do pacote `car`, enquanto o teste de Shapiro-Wilk para normalidade pode ser chamado pela função `shapiro.test()`. Note que para ambos os testes nós “queremos” a hipótese nula! O que acontece é que o teste de Levene tem como hipótese nula a homogeneidade das variâncias, e o teste de Shapiro-Wilk tem como hipótese nula a normalidade dos dados; ou seja, os pressupostos do teste  $t$  serão aceitos se os testes mencionados não rejeitarem a  $H_0$ . Pode parecer meio confuso no início, mas com o tempo a gente se acostuma. Como estes pressupostos são sobre os dados em si, devemos ter o costume de testá-los antes de iniciar o teste (o que é um pouco diferente nos modelos lineares, como veremos a seguir).

Como vimos acima, se o pressuposto da homogeneidade de variâncias for rejeitado, há uma saída prática: o teste t para variâncias heterogêneas. Na prática, isto está automaticamente feito no comando `t.test()`, a não ser que o usuário acrescente o argumento `var.equal=T` (como fizemos no exemplo acima). Ou seja, se nada for dito, o R já fará o teste t para variâncias heterogêneas.

Agora vejamos os exemplos. Primeiro o teste de Levene:

```
> leveneTest(Riqueza~Ambiente)
Levene's Test for Homogeneity of Variance (center = median)
  Df F value Pr(>F)
group 1 0.5134 0.4771
      48
>
> #Conclusão: as variâncias da variável Riqueza são homogêneas
> |
```

E agora o de Shapiro-Wilk. Veja com atenção o desdobramento do exemplo, no qual repeti o teste para cada classe da variável categórica. Isto é muito importante, ok?

```
> shapiro.test(Riqueza)

      Shapiro-Wilk normality test

data:  Riqueza
W = 0.9833, p-value = 0.6987

> #A conclusão é de que os dados seguem a distribuição normal.
>
> #Mas o correto para o teste t é fazer isso para cada um dos dois grupos, assim:
>
> shapiro.test(Riqueza[Ambiente=="primário"])

      Shapiro-Wilk normality test

data:  Riqueza[Ambiente == "primário"]
W = 0.9693, p-value = 0.6937

>
> shapiro.test(Riqueza[Ambiente=="secundário"])

      Shapiro-Wilk normality test

data:  Riqueza[Ambiente == "secundário"]
W = 0.9773, p-value = 0.7815
```

Vimos que se rejeitarmos a homogeneidade de variâncias temos um a solução prática “dentro” do próprio teste T. Já no caso da rejeição do pressuposto de normalidade (que deve sempre ser testado, aliás, para os dados agrupados nos dois grupos a serem comparados, como foi feito no exemplo acima), uma saída é recorrer a um teste não paramétrico. Não vamos explorar esta saída aqui, mas caso você deseje conhecê-la, veja a documentação do comando `wilcox.test()` para saber mais sobre o teste de Mann-Whitney.

### 3.6.3. Pressupostos da ANOVA e da regressão linear simples

Ué, por que juntar os dois?! Eu mencionei, talvez mais de uma vez (foi mesmo? não sei, mas não vou rever todo o texto só por isso...), que ANOVA e regressão são membros de uma “família” mais geral, os modelos lineares. Acontece que os modelos lineares possuem pressupostos em comum, e por isso podemos verifica-los usando as mesmas técnicas. Bacana, né?

Um pressuposto em comum de todos os modelos lineares é a normalidade dos resíduos. Isso mesmo: dos resíduos, e não dos dados em si. É uma confusão bem comum testar a normalidade dos dados, mas isto é desnecessário e pouco parcimonioso. Bom, nós já vimos como testar normalidade: basta usar o nosso amigo Shapiro-Wilk, como vimos no teste T ali em cima. Mas como obter os resíduos? Se você seguiu a minha recomendação de salvar o resultado do teste em um objeto, então será bem fácil: `objeto$residuals` vai funcionar com qualquer modelo linear no R:

```
>
> shapiro.test(resultado$residuals)

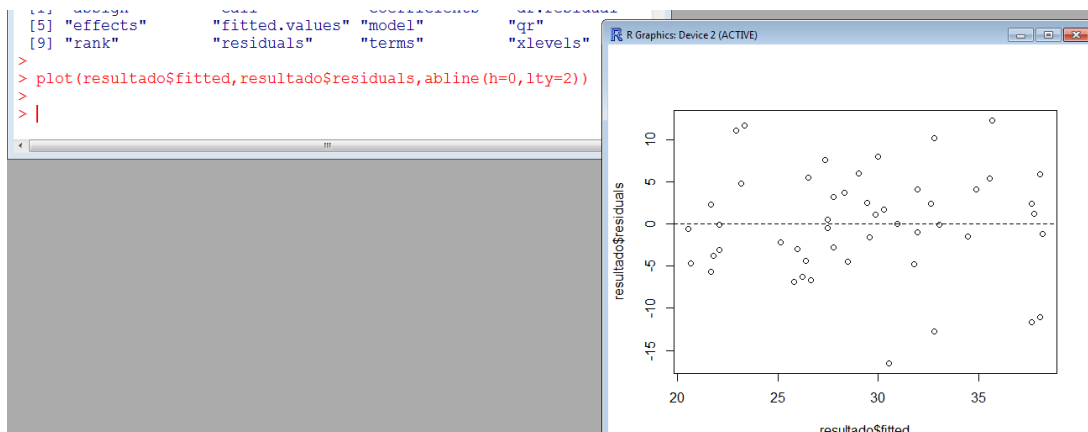
      Shapiro-Wilk normality test

data:  resultado$residuals
W = 0.9566, p-value = 0.3515
```

Neste exemplo, não rejeitamos a normalidade dos resíduos, então está tudo ok.

Na ANOVA, assim como no teste T, temos o pressuposto de homogeneidade de variâncias. Não tem complicação: se o pressuposto é o mesmo, o teste é o mesmo, então basta seguir com o teste de Levene como fizemos no teste T, ok?

Já na regressão, temos um pressuposto de que existe a chamada homocedasticidade dos dados (agora tente falar homocedasticidade bem rápido três vezes sem errar!). Conceitualmente, este é um pressuposto bem parecido com a homogeneidade de variâncias, mas aqui não temos classes. Podemos verifica-lo visualmente, comparando os resultados esperados pela regressão com os resíduos da análise. Não vamos entrar em detalhes sobre isto, então veja o exemplo da verificação visual da homocedasticidade a seguir e busque mais informações em um bom livro.



### 3.6. Funções utilizadas – capítulo 3

Vou resumir, a seguir, as funções que vimos nesta seção.

*chisq.test()* → teste de Qui-quadrado

*t.test()* → teste T (argumentos *paired* e *mu* podem ser usados para se obter o T pareado ou o T para uma amostra, respectivamente)

*aov()* → ANOVA (simples e outras variantes, só depende de como o modelo é construído)

*TukeyHSD()* → Teste *a posteriori* de Tukey para a ANOVA

*lm()* → regressão linear simples (ou outros modelos lineares, só depende de como o modelo é construído)

*shapiro.test()* → Teste de normalidade

*leveneTest()* → Teste de homogeneidade de variâncias

*abline()* → Adicionar linha (como a de um modelo) a um gráfico

## 4. E AGORA, O QUE VAMOS FAZER?

Parabéns! Se a minha apostila não foi uma total perda de tempo, você acaba de aprender o básico do nosso amigo R. Deste momento em diante, você começará a andar com as próprias pernas, então é uma boa hora para respirar fundo e tentar brincar um pouco com o R por conta própria, e tentar aprender algumas coisas “por aí”. E quando eu digo “por aí”, quero dizer na internet (e um pouco no próprio R também).

Neste capítulo quero deixar de lado os procedimentos, e falar um pouco sobre como podemos aprender a fazer uma análise no R mesmo que não tenhamos alguém nos ensinando. Fazer isso é importante por dois motivos. Primeiro, temos que lembrar que o R é, devido à sua natureza de software livre e de linguagem de programação, extremamente flexível (ou seja, ele faz coisa pra `c@#$$%&!.`). Então é obviamente possível que um usuário possa aprender tudo o que ele faz sem abrir mão de fazer outras coisas da sua vida, como comer, dormir, etc. Mas é aí que está a beleza da coisa: se você aprende a fazer um punhadinho de coisas no R (como aprendeu nos dois primeiros capítulos até aqui) e aprende a buscar informações por conta própria, está pronto para fazer qualquer tipo de análise nele. Segundo, o R tem um aspecto muito interessante, que é uma das suas maiores forças: uma grande comunidade de usuários e desenvolvedores. Isto significa que tem um monte de gente usando o R, um monte de gente com dúvidas e mais um outro monte tirando dúvidas. Se buscar informações sobre o R é fundamental, um lado bom é que há muita informação disponível!

O nosso objetivo neste capítulo final, então, é olhar rapidamente para algumas fontes importantes de informação, mecanismos de busca e comunidades de usuários.

### 4.1. Onde buscar ajuda?

#### 4.1.1. Documentação básica e sistema de ajuda

Ao ser instalado, o R já vem com alguns documentos básicos, que podem ser rapidamente acessados no menu ‘Ajuda’ do programa. Lá você poderá ver os documentos em PDF, acessar a central de ajuda em html no seu navegador e até mesmo chamar as funções de ajuda que vimos no primeiro capítulo. Além dos manuais que acompanham o programa, mais documentos podem ser encontrados no site oficial: na barra lateral à esquerda, clique em ‘Manuals’ para ter acesso tanto aos documentos principais quanto aos demais materiais (guias, tutoriais, etc.).

Caso você esteja usando muito as funções de um pacote, pode ser interessante visitar a página do pacote no CRAN, seguindo as instruções láááá da seção 1.8, e clicando no nome do pacote. Alguns pacotes possuem os *Vignettes*, que são manuais ou tutoriais sobre as capacidades do pacote. Veja, por exemplo, os documentos na página do pacote *vegan* na imagem a seguir:

**vegan: Community Ecology Package**  
 Ordination methods, diversity analysis and other functions for community and vegetation

Version: 2.0-10  
 Depends: [permute](#) (≥ 0.8-0), [lattice](#), R (≥ 2.15.0)  
 Suggests: [MASS](#), [mgcv](#), [cluster](#), [scatterplot3d](#), [rgl](#), [tcltk](#)  
 Published: 2013-12-12  
 Author: Jari Oksanen, F. Guillaume Blanchet, Roeland Kindt, Pierre Legendre, Stevens, Helene Wagner  
 Maintainer: Jari Oksanen <jari.oksanen@utu.fi>  
 License: [GPL-2](#)  
 URL: <http://cran.r-project.org>, <http://vegan.r-forge.r-project.org/>  
 NeedsCompilation: yes  
 Materials: [NEWS](#), [ChangeLog](#)  
 In views: [Environmetrics](#), [Multivariate](#), [Phylogenetics](#), [Psychometrics](#), [Spatial](#)  
 CRAN checks: [vegan results](#)

Downloads:

Reference manual: [vegan.pdf](#)  
 Vignettes: [Design decisions and implementation](#), [Diversity analysis in vegan](#), [Introduction to ordination in vegan](#)  
 Package source: [vegan\\_2.0-10.tar.gz](#)  
 MacOS X binary: [vegan\\_2.0-10.tgz](#)  
 Windows binary: [vegan\\_2.0-10.zip](#)

Por fim, já falamos um pouco sobre as funções de ajuda do R, e não há muito o que se acrescentar aqui. Todas as funções do R que estão em seus pacotes oficiais (ou seja, no CRAN) possuem uma estrutura básica de ajuda que é um ótimo ponto de partida para se aprender a usá-las. Os exemplos, como mencionado antes, são sempre uma boa coisa para se conferir. Se você quiser ter um acesso rápido aos exemplos, use a função *example()*, e o R executará tudo automaticamente para você.

```
R Console
> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
> |
```



#### 4.1.2. Bibliografia

Livros e apostilas sobre o R jamais vão faltar! Na verdade existe tanta coisa, mas tanta coisa, que pode ser difícil escolher o que usar... Mais importante do que deixar sugestões, então, é lembrar que a escolha da bibliografia acaba sendo bem pessoal neste caso. Então, caso você encontre um material de R e goste muito dele, siga adiante e use-o! Ainda assim, para aqueles mais perdidos, deixo aqui apenas duas sugestões de livros que considero bacanas para quem está aprendendo.

The R Book, Michael J. Crawley, 2012, Wiley:

Este é, como o nome sugere, **O** livro do R. É bem voltado para o uso de testes estatísticos, e um ótimo livro de referência para se aprender análises.

Numerical Ecology with R, D. Bocard, F. Gillet & P. Legendre, 2011, Springer:

Este é um excelente livro de base para se trabalhar com estatística multivariada no R, cobrindo boa parte dos métodos usados na ecologia.

#### 4.1.3. Google

Pois é: o bom e velho *google*, amigo de todas as horas de dúvida, é um ótimo companheiro para o usuário do R. Se você quer saber como se faz alguma análise, busque no *google* e veja o que aparece. Caso o que você busca seja algo mais ou menos comum, as chances de você encontrar um passo a passo bem mastigadinho de como fazer e interpretar a análise é altíssima! É claro que, como em qualquer pesquisa na internet, devemos ter um pouco de cautela com o que encontramos, já que existe muita coisa de má qualidade publicada sobre qualquer assunto. Mas basta ter um pouco de bom senso e olhar alguns sinais para fugir das coisas ruins online. Algumas sugestões:

- blogs e/ou sites muito movimentados e com muitos comentários costumam ser uma boa, pois os possíveis erros são facilmente detectados e corrigidos se muitas pessoas estão olhando.
- material escrito por autores que você já conhece ou que foram indicados também é uma boa pedida. Algumas vezes é possível encontrar sites mantidos por autores de pacotes oficiais do R, o que é um bom indicativo do material ser de qualidade.

#### 4.1.4. Listas de email

Existem muitas listas e grupos de email de usuários do R, normalmente organizados por áreas de interesse em comum. Caso você ache interessante participar,

comece dando uma olhada nas listas “oficiais”, que estão no CRAN. No site do R, vá na barra lateral e clique em “Mailing Lists”. A lista “R-sig”ecology” é bem interessante! Também há uma lista nacional bem ampla, a R-br: <http://r-br.2285057.n4.nabble.com/> Por fim, buscando na internet você certamente encontrará várias listas, grande e pequenas. Escolha as que achar mais adequadas e seja feliz! Mas lembre-se de algumas coisas importantes:

- antes de escrever pela primeira vez, procure saber se há um guia de postagens. Provavelmente ele existe, e deve conter as regras e sugestões sobre as mensagens da lista. Ignorar isso aumenta muito a chance de se levar um puxão de orelhas virtual...
- antes de pesquisar alguma coisa, busque no *google* e na lista, pois sempre existe a possibilidade da sua dúvida já ter sido respondida por alguém. Caso você ache que sua dúvida é muito básica, então isto provavelmente é verdade! Nestas listas é normalmente considerado uma falta de educação perguntar coisas cuja resposta pode ser encontrada com uma busca básica na internet.
- algumas listas são muito movimentadas, e participar delas significa receber um monte de mensagens todos os dias. Pondere isso antes de se cadastrar, e veja se a lista tem uma opção de receber o *digest*, que é um tipo de bloco de mensagens que você recebe de uma vez só no lugar das mensagens individuais.

## ANEXO 1: DADOS UTILIZADOS

Caso você não tenha os arquivos com os dados utilizados nos exemplos desta apostila, basta copiar e colar o texto da caixa correspondente no console do R e pronto!

*Dados do arquivo “pratica1.txt”:*

```
dados<-structure(list(UA = 1:50, Ambiente = structure(c(1L, 1L, 1L,
1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L,
1L, 1L, 1L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L,
2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L, 2L), .Label = c("primário",
"secundário"), class = "factor"), Área = c(101L, 115L, 143L,
92L, 51L, 89L, 128L, 149L, 127L, 108L, 83L, 140L, 163L, 41L,
107L, 79L, 104L, 60L, 82L, 148L, 97L, 49L, 164L, 110L, 51L, 40L,
92L, 163L, 128L, 130L, 122L, 96L, 59L, 73L, 90L, 48L, 166L, 121L,
167L, 48L, 78L, 105L, 90L, 105L, 166L, 84L, 57L, 81L, 122L, 112L
), Riqueza = c(35L, 31L, 39L, 25L, 22L, 35L, 43L, 48L, 35L, 38L,
32L, 33L, 40L, 16L, 31L, 23L, 32L, 35L, 22L, 41L, 24L, 18L, 39L,
32L, 19L, 20L, 31L, 26L, 20L, 33L, 36L, 32L, 28L, 23L, 28L, 16L,
44L, 27L, 37L, 24L, 19L, 28L, 27L, 28L, 27L, 20L, 34L, 20L, 31L,
14L), Abund_sp1 = c(3L, 3L, 1L, 6L, 1L, 0L, 3L, 5L, 0L, 2L, 0L,
2L, 2L, 3L, 5L, 3L, 1L, 4L, 3L, 0L, 2L, 3L, 1L, 3L, 3L, 1L, 3L,
1L, 2L, 2L, 2L, 2L, 8L, 5L, 1L, 0L, 3L, 0L, 3L, 2L, 0L, 2L, 0L,
0L, 1L, 3L, 3L, 4L, 1L, 0L)), .Names = c("UA", "Ambiente", "Área",
"Riqueza", "Abund_sp1"), class = "data.frame", row.names = c(NA,
-50L))

#Pronto, os dados estão salvos no objeto 'dados'; pressione enter siga adiante!
```



*Dados do arquivo “peixes.txt” (no exemplo de ANOVA):*

```
dados<-structure(list(Predador = structure(c(3L, 3L, 3L, 3L, 3L,
3L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 2L, 2L, 2L, 2L, 2L, 2L,
2L, 2L, 2L), .Label = c("Libélula", "Nenhum", "Peixe"), class = "factor"),
  Densidade = c(10.4993, 17.6714, 11.1032, 4.6647, 18.9767,
  14.2042, 14.8269, 11.8394, 20.9545, 7.3108, 20.2219, 21.4057,
  19.6589, 9.9067, 14.5693, 11.037, 11.1047, 4.9696, 7.7347,
  16.9973, 7.6902, 10.1636, 1.9043, 1.767, 12.1435)), .Names = c("Predador",
"Densidade"), class = "data.frame", row.names = c(NA, -25L))

#Pronto, os dados estão salvos no objeto 'dados'; pressione enter siga adiante!
```

## ANEXO 2: GUIA RÁPIDO!

Neste anexo, apresento um guia rápido de procedimentos no R. Pense nele como um resumo das principais funções vistas ao longo desta apostila. O objetivo é permitir que quem já leu a apostila possa rever rapidamente as funções durante o uso do programa.

### A2.1. Usando a ajuda.

- Abrir ajuda de uma função conhecida: ?(função)
- Procurar função buscando por uma palavra: ??palavra
- Procurar função buscando por uma expressão: help.search(“expressão desejada”)

### A2.2. Lendo os dados.

- Vá no menu ‘Arquivo’, selecione a opção ‘mudar dir’ e selecione a pasta onde está o arquivo que deseja ler.
- Se desejar, confira os nomes dos arquivos na pasta: dir()
- Use o comando: objeto<-read.table(“arquivo.txt”, header=T)
- Confira os dados: summary(objeto)
- Associe os nomes das variáveis: attach(objeto)
- Se necessário, carregue os pacotes que irá usar: library(pacote)

### A2.3. Gráficos básicos.

- Histograma: hist(variável)
- Dispersão: plot(variável\_x, variável\_y)
- Barras, primeiro use: tabela<-table(variável\_1, variável\_2);
  - depois use: barplot(tabela, beside=T, legend=rownames(tabela))
- Boxplots: boxplot(variável\_resposta~variável\_explicativa)
- Médias: lineplot.CI(variável\_explicativa, variável\_resposta, type=”p”)
  - depende de pacote, então antes de fazer, use: library(sciplot)

#### A2.4. Testes estatísticos.

- Qui-quadrado: `chisq.test(tabela)`
  - tabela deve ser gerada com: `tabela<-table(variável_1, variável_2)`
- Teste T para duas amostras: `t.test(variável_resposta~variável_explicativa, var.equal=T)`
  - Se as variâncias forem heterogêneas, omitir `var.equal=T`
- Teste T pareado: `t.test(medida1, medida2, paired=T)`
- Teste T para uma amostra: `t.test(variável, mu=valor_de_comparação)`
- ANOVA: `resultado<-aov(variável_resposta~variável_explicativa)`
  - e depois: `summary(resultado)`
  - se desejar a posteriori: `TukeyHSD(resultado)`
- Regressão linear: `resultado<-lm(variável_resposta~variável_explicativa)`
  - e depois: `summary(resultado)`
  - para o gráfico, primeiro: `plot(variável_explicativa, variável_resposta)`
  - depois: `abline(resultado)`

#### A2.5. Testes de pressupostos.

- Normalidade: `shapiro.test(variável)`
  - Nos modelo lineares: `shapiro.test(resultado$residuals)`
- Homogeneidade de variâncias: `leveneTest(variável_resposta~variável_explicativa)`
- Homocedasticidade, veja o gráfico: `plot(resultado$fitted, resultado$residuals)`