

R: Primeiros passos (inclusive em Linux!)

Esse post é pra você que nunca usou o R, ou pra você que usa o R mas nunca adquiriu uma base, ou pra você que manja de R, vai ensinar R pra alguém e não sabe por onde começar. Vou falar de: como instalar o R; criar um script; fazer algumas operações matemáticas; criar alguns objetos; abrir uma planilha de dados; ~~e ser feliz~~. Como qualquer começo, este começo pode ser chato, mas recomendo ler até o final e praticar em R enquanto lê ou depois :-)

Existem inúmeros tutoriais, em texto e vídeo, ensinando a usar o R. Acho que tem até um site com tudo isso organizado! Aqui neste blog tem o script-base do minicurso de R que ministro <<https://anothercoblog.files.wordpress.com/2015/09/introducaoar1.pdf>>. Mas como acho mais divertido escrever coisas do que organizar coisas já escritas, segue aqui a minha contribuição a esta literatura. :-)

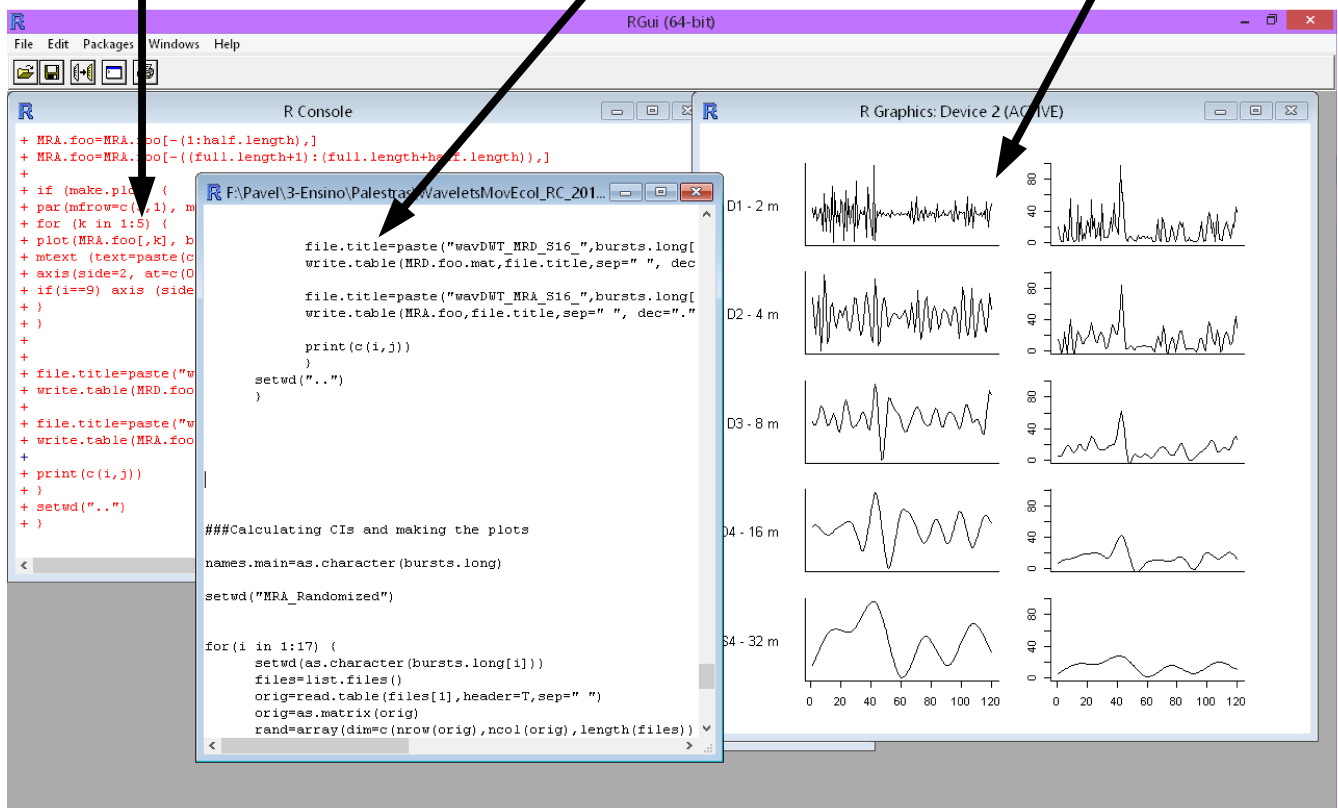
Bom, a primeira coisa a fazer é ir no site do R e instalar o R! <<https://cran.r-project.org/>> Em Windows basta baixar o arquivo; em Linux o site fornece as instruções. Eu acho que simplesmente rodei o comando `sudo apt-get install r-base`. Tendo instalado o R, podemos abrir ele e começar a brincar!

Em Windows, se a instalação foi feita de modo padrão, o R abre mostrando o *console*, onde aparecem os comandos, os resultados deles e toda a outra informação útil. Você pode escrever os comandos diretamente no console, mas não recomendo fazer isso. É melhor ir em *arquivo (file)* -> *novo script (new script)* (ou algo assim, não lembro mais). Isso abre um script para você escrever os comandos. Para rodar um comando no script, você pode selecionar a parte que quer rodar e apertar CTRL+R (em Windows) ou CMD+ENTER (em Mac). Se não selecionar uma parte do script, ele vai rodar a linha onde o cursor está inteira. Este script pode ser salvo (arquivo, salvar script; isso só funciona se a janela do script estiver ativa. Recomendo fazê-lo ao criar um script novo, e periodicamente ao trabalhar nele. Nunca se sabe quando ~~alguém pode reiniciar seu computador~~ seu computador pode travar sem nenhum motivo aparente). Ao fazermos um gráfico, ele aparece em uma outra janela.

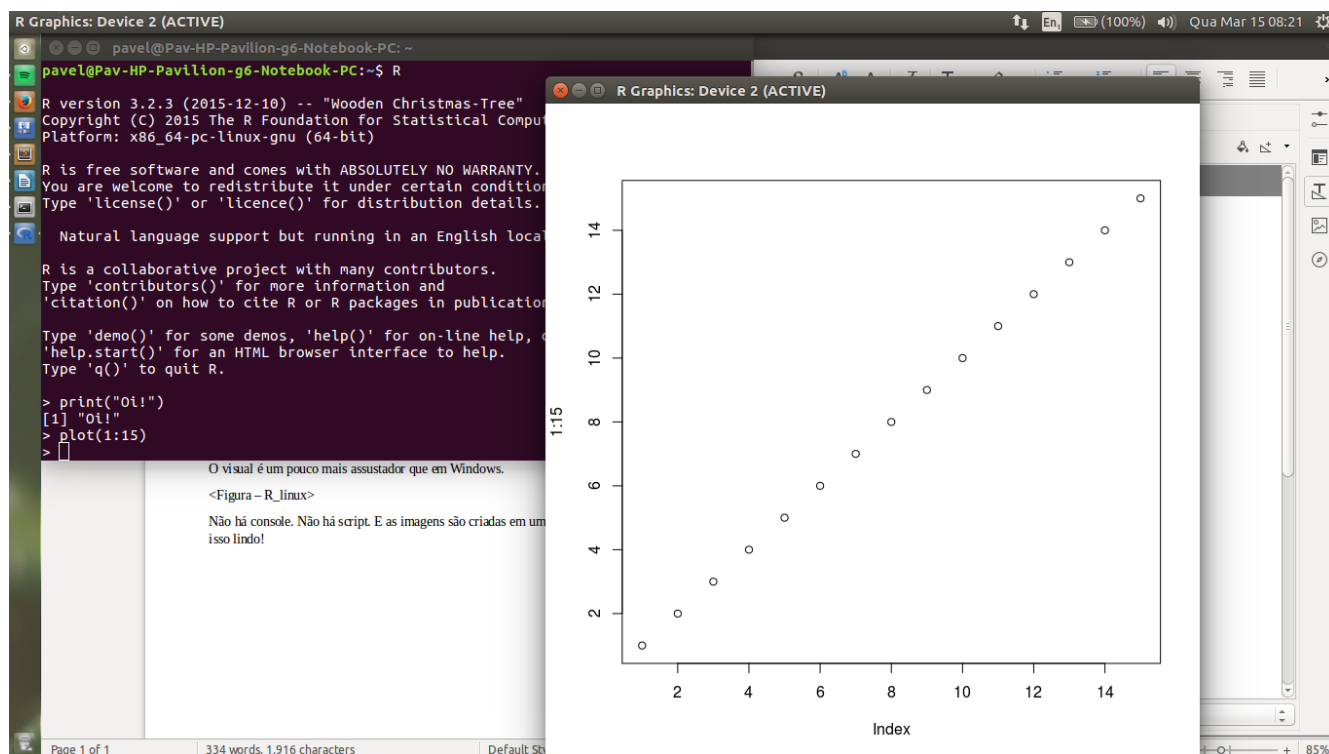
Console

Script

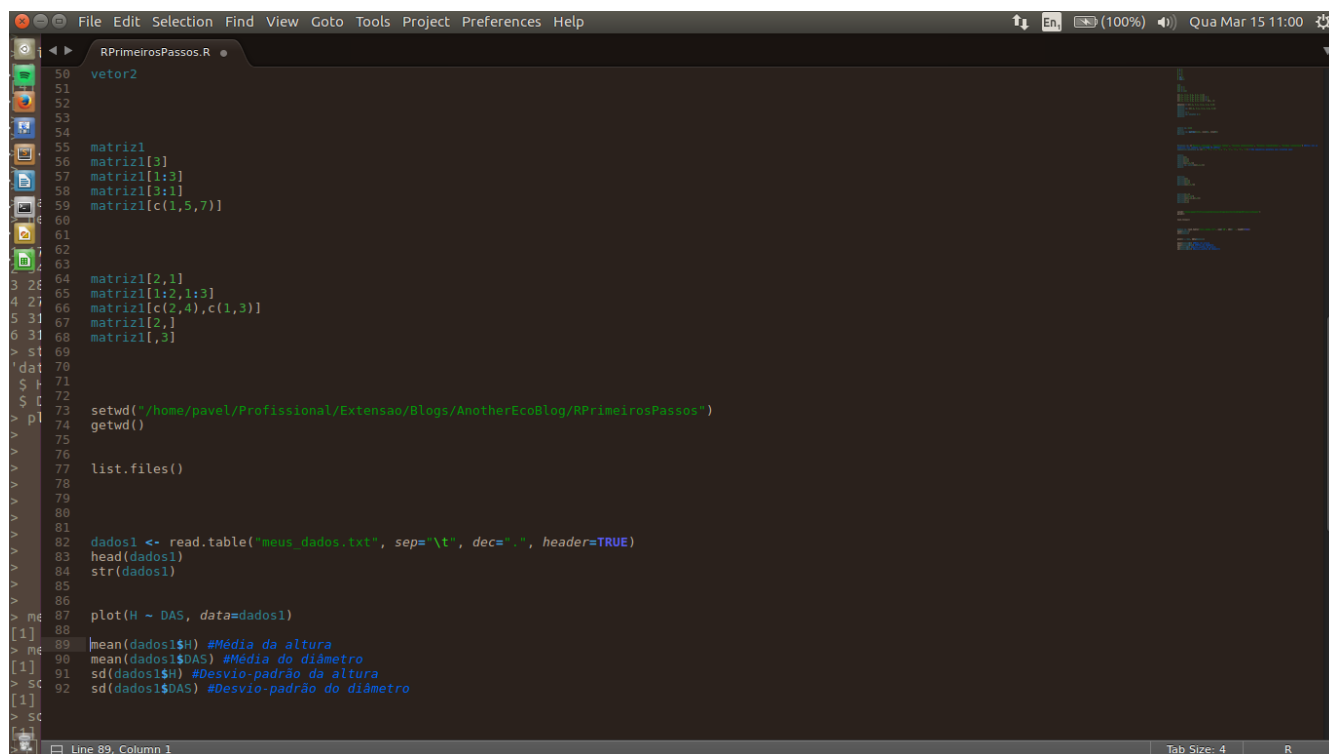
Gráficos



Em Linux, podemos abrir o R a partir do terminal – basta ligar o terminal, escrever R e apertar ENTER. O visual é um pouco mais assustador que em Windows.



Não há console. Não há script. E as imagens são criadas em uma outra janela. Eu pessoalmente acho isso lindo! Gosto inclusive de apertar F11 para trabalhar com o R em tela cheia. Mas aí precisamos de um outro editor de texto para fazer o script. Qualquer editor de texto serve, mas alguns têm funcionalidades específicas para programação. Eu gosto muito do Sublime Text <<https://www.sublimetext.com/>>. Ele tem versões para Windows, OS X, e Ubuntu. Ele é pago (70 dólares – barato para um software!), mas pode ser baixado e usado por tempo indeterminado sem a licença, ao custo de receber periodicamente lembretes amigáveis do tipo “Oi, você ainda não pagou pelo seu Sublime Text! Quer fazer isso agora? :-)”.



```
50 vetor2
51
52
53
54
55
56 matriz1
57 matriz1[3]
58 matriz1[1:3]
59 matriz1[c(1,5,7)]
60
61
62
63
64 matriz1[2,1]
65 matriz1[1:2,1:3]
66 matriz1[c(2,4),c(1,3)]
67 matriz1[2,]
68 matriz1[,3]
69
70
71
72
73 setwd("/home/pavel/Profissional/Extensao/Blogs/AnotherEcoBlog/RPrimeirosPassos")
74 getwd()
75
76
77 list.files()
78
79
80
81
82 dados1 <- read.table("meus_dados.txt", sep="\t", dec=".", header=TRUE)
83 head(dados1)
84 str(dados1)
85
86
87 plot(H ~ DAS, data=dados1)
88
89 mean(dados1$H) #Média da altura
90 mean(dados1$DAS) #Média do diâmetro
91 sd(dados1$H) #Desvio-padrão da altura
92 sd(dados1$DAS) #Desvio-padrão do diâmetro
```

Eu escrevo o código no Sublime Text, seleciono ele, vou ao R usando ALT+TAB, e insiro o código usando SHIFT+INSERT. É rápido!

(Às vezes, em Linux, quando digitamos ou colamos uma linha de comando longo, ele sobreescreve o começo da linha ao invés de fazer uma quebra de linha. Ou seja, ao invés de aparecer isso:

"Oi! Eu sou uma linha muito longa digitada para mostrar o que pode acontecer quando você digita uma linha muito longa em R!"

Aparece isso!

acontecer quando você digita uma linha muito longa em R!" que pode

Sim, isso acontece. A solução que achei na internet para isso é

<<http://stackoverflow.com/questions/2024884/commandline-overwrites-itself-when-the-commands-get-to-long>> , antes de ligar o R, usar o comando `shopt -s checkwinsize`.)

Bom, agora que já ligamos o R e abrimos o nosso script (no R ou em um editor de texto), vamos brincar um pouquinho. :-)

Primeiro: Podemos usar o R como uma calculadora! Vamos inserir o seguinte código:

5 + 2

```
5 - 2
5 * 2
5 / 2
5 ^ 2
5 %% 2
5 %/% 2
```

O resultado que aparece no R é o seguinte:

```
> 5 + 2
[1] 7
> 5 - 2
[1] 3
> 5 * 2
[1] 10
> 5 / 2
[1] 2.5
> 5 ^ 2
[1] 25
> 5 %% 2
[1] 1
> 5 %/% 2
[1] 2
```

Nas linhas acima, as linhas que começam com um `>` são o código inserido, e as outras, que começam com um `[1]`, são o resultado fornecido pelo R. No restante do texto mostrarei simultaneamente o código e o *output*; ao inserir este código no R, não insira os `>`.

As operações matemáticas usadas são: soma (+), subtração (-), multiplicação (*), divisão (/), potenciação (^), resto de divisão (%%), e divisão inteira, sem resto (%/%). Resto de divisão e divisão inteira são bem úteis em alguns casos, por exemplo para testar se um número é par ou ímpar.

E não precisamos fazer isso com um único número por vez. Podemos juntar números e fazer as operações em todos os números simultaneamente:

```
> 1:5
[1] 1 2 3 4 5
> 1:5 + 2
[1] 3 4 5 6 7
```

```
> 1:5 / 2
[1] 0.5 1.0 1.5 2.0 2.5
> 1:5 + 2:6
[1] 3 5 7 9 11
```

O comando `1:5` faz uma sequência de 1 até 5. O segundo comando soma 2 a cada um destes números: 1+2, 2+2, ..., 5+2. O terceiro comando faz a mesma coisa mas dividindo. E o quarto comando soma a sequência de 2 a 6 à primeira sequência: 1+2, 2+3, ..., 5+6.

Também podemos juntar, ou concatenar, um conjunto de números:

```
> c(2.2, 3.1, 6.4, 4.2, 9.8)
[1] 2.2 3.1 6.4 4.2 9.8
> c(2.2, 3.1, 6.4, 4.2, 9.8) + 2
[1] 4.2 5.1 8.4 6.2 11.8
> c(2.2, 3.1, 6.4, 4.2, 9.8) / 2
[1] 1.10 1.55 3.20 2.10 4.90
> c(2.2, 3.1, 6.4, 4.2, 9.8) * c(1, 2)
[1] 2.2 6.2 6.4 8.4 9.8
```

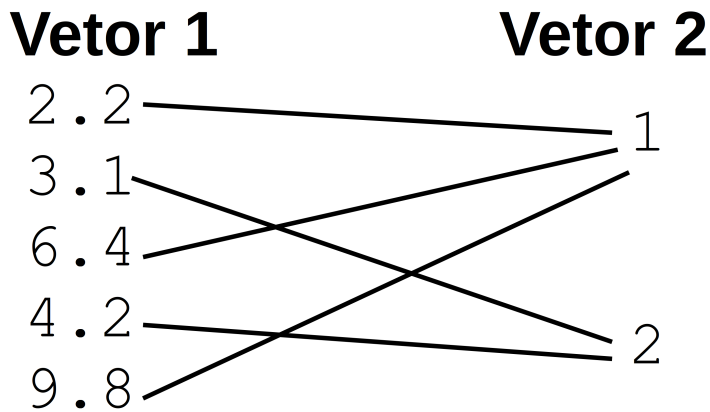
Warning message:

```
In c(2.2, 3.1, 6.4, 4.2, 9.8) * c(1, 2) :
```

```
longer object length is not a multiple of shorter object length
```

No primeiro comando, criamos um **vetor** de números: 2.2, 3.1, 6.4, 4.2, 9.8. Um **vetor** nada mais é que uma sequência de números (ou de texto...), e eu diria que é a unidade básica com que trabalhamos em R. Reparem que uso o ponto como separador decimal, não a vírgula! É assim que o R trabalha.

No segundo comando somamos 2 a cada um destes números, e no terceiro comando subtraímos 2. E no terceiro comando fazemos algo mais interessante: multiplicamos um vetor com cinco elementos por um vetor com 2 elementos que contém os números 1 e 2. Neste caso, o R multiplica o primeiro elementos de um vetor pelo primeiro elemento do outro vetor, o segundo elemento pelo segundo, o terceiro pelo primeiro, o quarto pelo segundo, e assim vai.



O aviso no final diz que o número de elementos no vetor mais longo, com cinco elementos, não é múltiplo do número de elementos no vetor mais curto, com dois elementos. Assim, o primeiro elemento do vetor mais curto (1) foi usado três vezes, enquanto o segundo (2) foi usado apenas duas.

Mas não faz sentido ficar digitando os números toda vez que formos fazer algo – especialmente quando trabalhamos com dados reais que frequentemente têm centenas ou milhares de valores! Para isso que servem, em R, os **objetos**. Um **objeto** pode ser entendido como bem... Bem, como uma coisa que contém números e/ou texto, com uma certa organização. Para transformarmos a sequência de números que usamos acima em um objeto para trabalharmos, precisamos dar um nome a ela:

```
> objeto1 = c(2.2, 3.1, 6.4, 4.2, 9.8)
> objeto1
[1] 2.2 3.1 6.4 4.2 9.8
> objeto1 <- c(2.2, 3.1, 6.4, 4.2, 9.8)
> objeto1
[1] 2.2 3.1 6.4 4.2 9.8
> objeto1 + 2
[1] 4.2 5.1 8.4 6.2 11.8
> objeto2 <- objeto1 + 2
> objeto2
[1] 4.2 5.1 8.4 6.2 11.8
```

Na primeira linha, criamos um objeto, chamado objeto1, que tem cinco números. Usamos o sinal de = para criar o objeto. Na segunda linha “chamamos” este objeto – como se falamos “Ei, objeto1! Mostre-se!”, ou “Prezado objeto1, por favor, mostre-se para mim”, ou “objeto1, eu te invoco, em nome de

Mumm-Ra, O de Vida Eterna!”. Depois fazemos a mesma coisa, mas usando o sinal `<-` ao invés de `=`. O resultado é o mesmo; eu tenho usado o `<-` porque o Sublime Text mostra o código de um jeito mais bonitinho quando faço assim. Mas alguns livros com foco em R usam o `=`. Façam como acharem melhor; embora acho que quem programa mais em R usa `<-`. A seguir realizamos as mesmas operações que antes: somamos 2 a cada elemento deste nosso objeto, agora invocando (a palavra invocar é tão legal, né? rs) o objeto já criado ao invés de digitar a sequência de números toda vez. Depois criamos um novo objeto, que guarda em si os resultados desta somatória, e o invocamos.

Resumidamente: criamos um objeto; realizamos uma operação matemática nele; guardamos o resultado em um novo objeto; e invocamos este novo objeto para ver o resultado.

Em R, temos alguns tipos de objetos;

Vetor: Como dito acima, é uma sequência de números.

Matriz: Também uma sequência de números, mas distribuídos em linhas e colunas. Observem:

```
> vetor1 <- 1:12
> vetor1
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> matriz1 <- matrix(1:12, ncol=3, nrow=4)
> matriz1
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Os números são os mesmos, mas na matriz que criamos eles são distribuídos em quatro linhas e três colunas.

Array: Novamente uma sequência de números, mas com mais de duas dimensões. Ou seja, além de ter linhas e colunas, podemos ter uma terceira dimensão (imaginem um cubo preenchido de números), ou uma quarta dimensão (imaginem um cubo preenchido com números que vão mudando ao longo do tempo), ou uma quinta dimensão (imaginem que vocês conseguem imaginar um objeto de cinco dimensões), e assim vai.

E se você entende de R, já deve ter percebido que estou mentindo, mas só um pouquinho. :-) Na verdade, não precisam ser apenas números. Vetores, matrizes e arrays também podem conter texto – por exemplo, nomes de espécies ou sequência de DNA; e também podem ser do tipo verdadeiro-ou-falso, representados por `TRUE` e `FALSE` ou `T` e `F`.


```

> Miconias <- c("Miconia albicans", "Miconia fallax", "Miconia stenostachya",
"Miconia ligustroides", "Miconia rubiginosa") #Estas são as espécies que conheço no
cerrado da UFSCar
> sequencia.genetica <- c("A", "T", "C", "A", "C", "G", "A", "C", "G") # Uma
sequência genética que inventei aqui
> Miconias
[1] "Miconia albicans"      "Miconia fallax"      "Miconia stenostachya"
[4] "Miconia ligustroides" "Miconia rubiginosa"
> sequencia.genetica
[1] "A" "T" "C" "A" "C" "G" "A" "C" "G"

```

Na primeira linha, criei um vetor que contém nomes de espécies. Na segunda linha, fiz uma sequência genética. Repararam no #? Usamos # para colocar comentários no código. Comentários são partes do código que foram escritas para serem lidas por pessoas, não pelo computador. Ao rodar o código, o R ignora as partes precedidas por #.

Neste momento vocês talvez estejam curiosas/os sobre o [1] e [4] que aparece acima. Eles mostram o número do elemento na sequência. Então, “Miconia albicans” é o primeiro elemento e “Miconia ligustroides” é o quarto elemento. O R faz isso quando o *output* não cabe em uma única linha. A partir desta informação, podemos pegar um objeto de um elemento usando os colchetes:

```

> vetor1
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> vetor1[3]
[1] 3
> vetor1[1:3]
[1] 1 2 3
> vetor1[3:1]
[1] 3 2 1
> vetor1[c(1,5,7)]
[1] 1 5 7
> vetor2 <- vetor1[c(1,5,7)]
> vetor2
[1] 1 5 7

```

Temos aquele vetor que criamos antes; depois invocamos o terceiro elemento dele, depois os elementos 1 a 3, depois os elementos 3 a 1, e depois os elementos de números 1, 5 e 7. E finalmente criamos um novo vetor, *vetor2*, com estes elementos.

E com matrizes? Podemos fazer a mesma coisa:

```

> matriz1
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> matriz1[3]
[1] 3
> matriz1[1:3]
[1] 1 2 3
> matriz1[3:1]
[1] 3 2 1
> matriz1[c(1,5,7)]
[1] 1 5 7

```

O R, neste caso, trata a matriz como um vetor. Mas também podemos referenciar, ou **indexar**, usando os números das linhas e das colunas:

```

> matriz1[2,1]
[1] 2
> matriz1[1:2,1:3]
      [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
> matriz1[c(2,4),c(1,3)]
      [,1] [,2]
[1,]    2   10
[2,]    4   12
> matriz1[2,]
[1] 2 6 10
> matriz1[,3]
[1] 9 10 11 12

```

Primeiro pegamos o elemento da segunda linha e primeira coluna; depois as linhas 1 e 2 e colunas 1 a 3; depois as linhas 2 e 4 e as colunas 1 e 3; depois a segunda linha e todas as colunas; e depois a terceira coluna e todas as linhas. Reparem que, ao pegarmos uma única linha ou uma única coluna, o objeto resultante é um vetor, pois não temos mais múltiplas linhas e múltiplas colunas.

Tá, mas você não veio aqui pra ficar criando objetos aleatórios, né? (Se veio, não há problema algum! Sinta-se em casa!) Você deve querer saber como lidar com seus próprios dados. Existe mais de uma forma de inserir eles no R; vou mostrar uma delas aqui.

Faça o seguinte:

- Abra sua planilha de dados (em Excel ou Calc), ou crie uma planilha de dados. Para deixar a coisa mais simples, escolha uma planilha com duas variáveis quantitativas – diâmetro e altura de plantas, ou temperatura do ambiente e da preguiça, ou quantidade de ração consumida e de cercotrofia, ou o que mais você quiser.
- Selecione as duas colunas, com o cabeçalho (sem espaços nos nomes das linhas e colunas, pra facilitar), copie elas e cole num editor de texto apropriado (bloco de notas, Sublime Text... Word ou Wordpad ou Writer não servem.)
- Salve o arquivo, com uma extensão .txt. Podemos usar o nome meus_dados.txt.
- Descubra o caminho da pasta onde você salvou este arquivo
- Fale para o R onde está esta pasta, usando o seguinte comando:

```
setwd("/home/pavel/Profissional/Extensao/Blogs/AnotherEcoBlog/RPrimeirosPassos")
```

Este é o comando no meu computador; no seu o caminho da pasta será diferente. Duas coisas são importantes: deixar o caminho da pasta entre aspas; e usar a barra ascendente, /, e não a descendente, \, como separador de diretórios.

Verifique se o R entendeu usando o seguinte comando:

```
getwd()
```

Se tudo ocorreu certo, ele vai mostrar onde está o endereço da pasta.

```
> getwd()
```

```
[1]
```

```
"/home/pavel/Profissional/Extensao/Blogs/AnotherEcoBlog/RPrimeirosPassos"
```

Podemos usar o comando `list.files`, que vai mostrar os arquivos que temos na pasta. `list.files` é uma função, assim como `setwd`, `getwd`, e `matrix` que usamos antes. Funções são estruturas que “fazem alguma coisa”. `list.files` mostra que arquivos estão na pasta de trabalho, `setwd` define a pasta de trabalho, `getwd` mostra a pasta de trabalho, e `matrix` cria uma matriz. Funções precisam ser seguidas de parênteses: daí usamos `getwd()`, e não simplesmente `getwd`. Algumas funções têm

argumentos – no caso da `setwd`, o argumento é o caminho da pasta; no caso da função `matrix`, os argumentos são: que números farão parte da matriz, quantas linhas ela terá, e quantas colunas ela terá.

```
> list.files()

[1] "Figs.odg"          "meus_dados.txt"      "R_caption.png"
[4] "R_linux.png"       "RPrimeirosPassos.odt" "RPrimeirosPassos.R"
[7] "R_vetores.png"
```

Temos alguns arquivos nesta pasta (que é a pasta onde estou guardando os arquivos deste meu post no meu computador! Olhem só.), inclusive o `meus_dados.txt` que criamos. Vamos abrir ele e olhar como ele é:

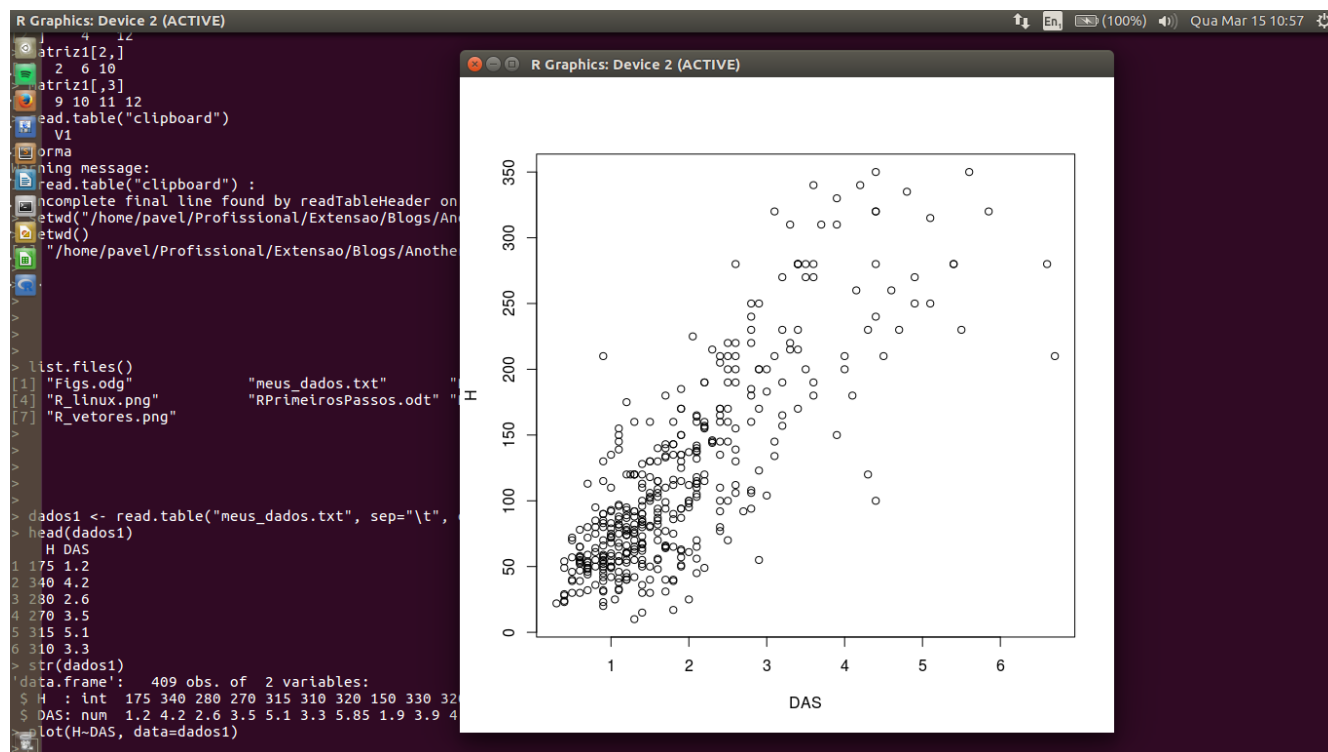
```
> dados1 <- read.table("meus_dados.txt", sep="\t", dec=".", header=TRUE)
> head(dados1)
      H DAS
1 175 1.2
2 340 4.2
3 280 2.6
4 270 3.5
5 315 5.1
6 310 3.3
> str(dados1)
'data.frame':   409 obs. of  2 variables:
 $ H  : int  175 340 280 270 315 310 320 150 330 320 ...
 $ DAS: num  1.2 4.2 2.6 3.5 5.1 3.3 5.85 1.9 3.9 4.4 ...
```

O comando `read.table` abre uma tabela de dados. Ele tem alguns argumentos: o nome do arquivo (`meus_dados.txt`, entre aspas); `sep="\t"` significa que as colunas são separadas por tabulações, que é o caso neste arquivo porque criamos ele copiando e colando a partir do Excel ou Calc; `dec="."` significa que o separador decimal é o ponto, e não a vírgula (se o seu Excel ou Calc está em português, talvez seja vírgula. Confira isso); `header=TRUE` significa que temos um cabeçalho, com os nomes das colunas.

É um objeto grande, então nos confundiríamos se olhássemos ele inteiro. Por isso usamos o comando `head(dados1)`, que mostra apenas as seis primeiras linhas. Olhando ele, sabemos que temos duas colunas, de nomes H (Height – altura) e DAS (Diâmetro à Altura do Peito). São dados de altura e diâmetro de plantinhas que medi muitos anos atrás <

E agora podemos fazer um gráfico disso aí! Usemos o seguinte comando (usando os nomes das colunas do objeto que você criou aí):

E o R mostrará lindamente a relação entre as suas variáveis :-)



```
> mean(dados1$H) #Média da altura
[1] 117.1418

> mean(dados1$DAS) #Média do diâmetro
[1] 1.893765

> sd(dados1$H) #Desvio-padrão da altura
[1] 74.64008

> sd(dados1$DAS) #Desvio-padrão do diâmetro
```

[1] 1.130262

E estes são os passos iniciais para trabalhar com R; explicados em pouco mais de 3000 palavras. O R tem uma curva de aprendizado bem íngreme no começo, mas vale a pena investir nesta base, pois depois ele pode facilitar muito a sua pesquisa e talvez até a sua vida (já fiz sorteio de amigo secreto em R, mais de uma vez. rs). Espero que tenham aproveitado! :-)