Tutorial R/RStudio

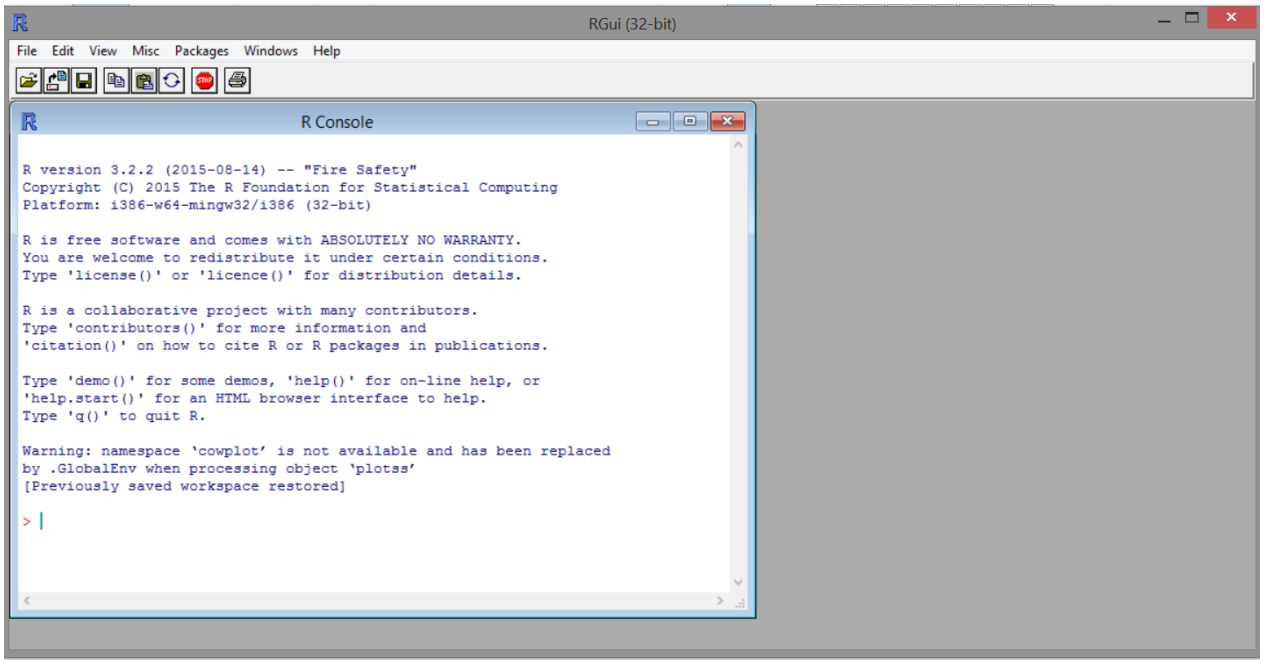
Verônica Santana
FEA-USP

2017

1 Instalação e Interface

1.1 R

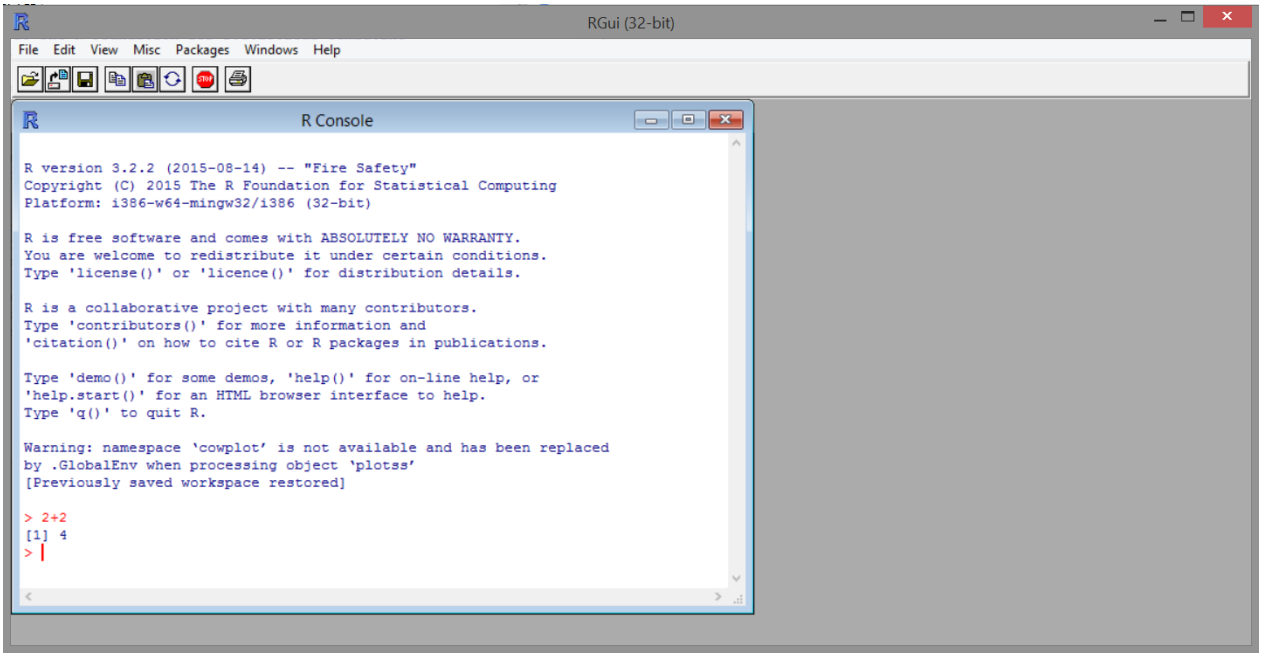
Vá em `https://www.r-project.org/`, e clique em CRAN (Comprehensive R Archive Network) à esquerda, e escolha um *mirror*, `https://vps.fmvz.usp.br/CRAN/`, por exemplo, para baixar o arquivo de instalação. Após instalado, o R tem uma interface assim, com apenas o console para digitar comandos.



Experimente um comando: `2+2`, cujo output é `4`:

```
2+2
```

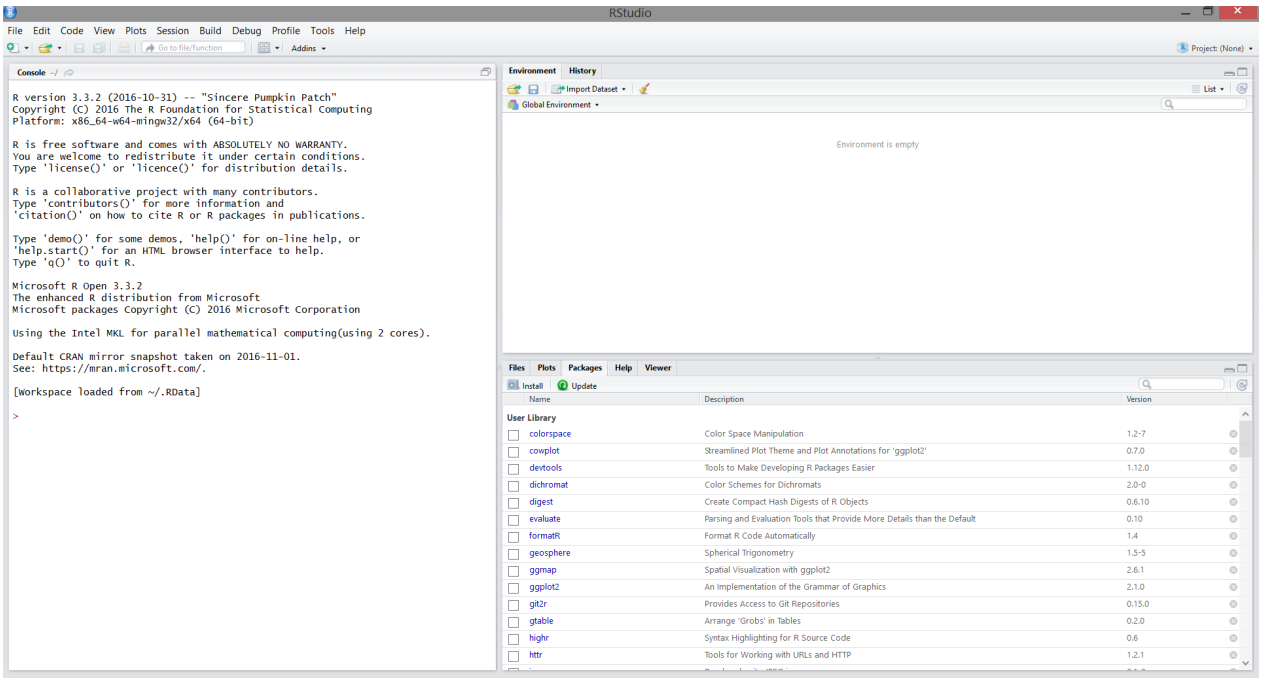
```
## [1] 4
```



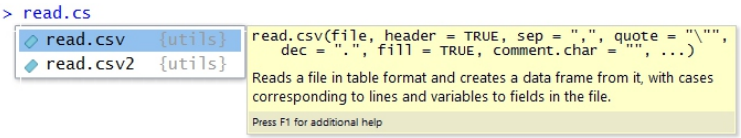
1.2 RStudio

O RStudio é uma interface funcional e amigável para o R. Para baixar e instalar o RStudio, vá em `https://www.rstudio.com/products/rstudio/download/` e escolha a opção *RStudio Desktop*.

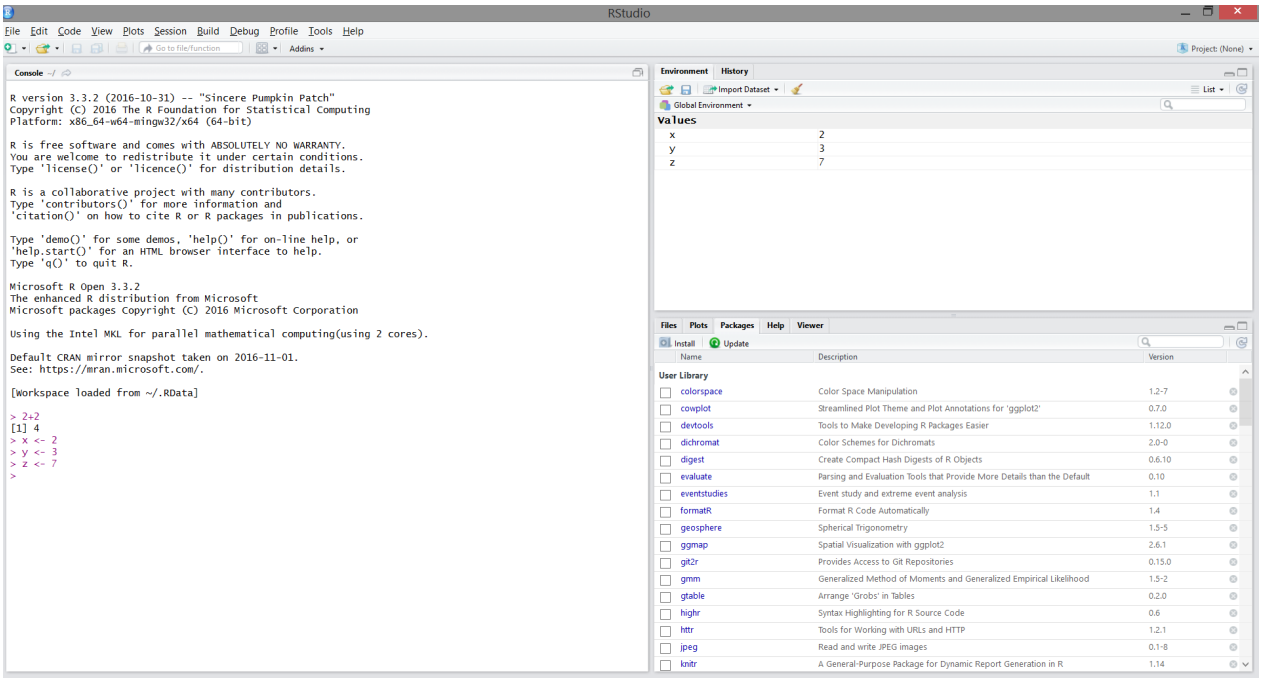
A interface do RStudio é dividida, inicialmente, em 3 partes:



Do lado esquerdo fica o console, onde os comandos podem ser digitados e onde ficam os *outputs*. O diferencial do RStudio em relação ao R no console é que os comandos são autocompletáveis. Experimente, por exemplo, começar a escrever o comando `read.csv()`, usado para importar dados no formato *comma separated values* do Excel:



No lado superior direito há duas abas: (i) *Environment*, que é onde ficam armazendos os objetos criados, bases de dados importadas, etc; e (ii) *History*, onde ficam o histórico dos comandos executados. Na aba *Environment* são mostrados os objetos que foram criados, como, no exemplo, os objetos `x`, `y` e `z`:



Bases de dados importadas também são mostradas neste espaço, assim como funções criadas, equações salvas, etc. A aba *History* mostra o histórico de comandos.

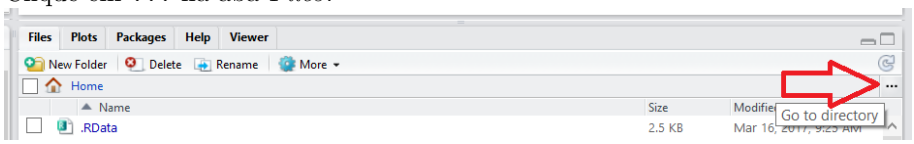
No lado inferior direito há 5 abas. Na aba *Files* aparecem os arquivos constantes do diretório de trabalho, ou *working directory*, cujos detalhes estão na seção 2. Na aba *Plots* ficam os gráficos gerados, na aba *Packages* estão listados os pacotes instalados, cujos detalhes também estão na seção 2, e a aba *View* é para visualização de conteúdo da web.

2 Funções básicas

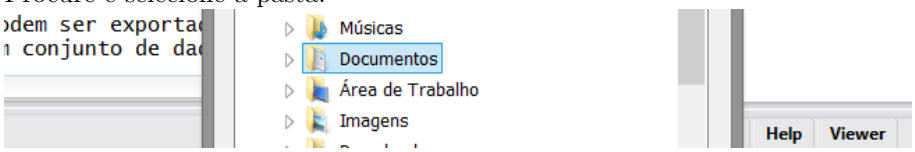
2.1 *Working Directory*

O *Working Directory* é uma pasta no seu computador que interage com o R/RStudio. Assim, antes de começar a trabalhar com o *software*, crie uma pasta em seu computador. Em seguida, ao abrir o RStudio, o primeiro passo é dizer ao software qual pasta será o *Working Directory*, pois é nesta pasta que ele procurará as bases de dados para importar e é para esta pasta que os arquivos gerados pelo R serão exportados, como, por exemplo, gráficos. Para tanto, siga os passos a seguir:

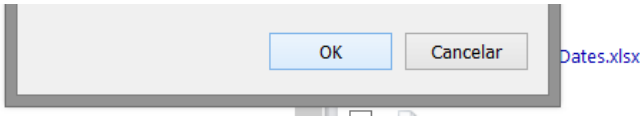
1. Clique em ... na aba *Files*:



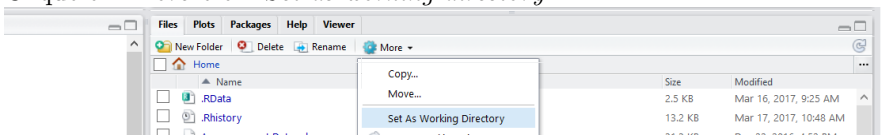
2. Procure e selecione a pasta:



3. Clique em OK:



4. Clique em *More* e em *Set as working directory*:

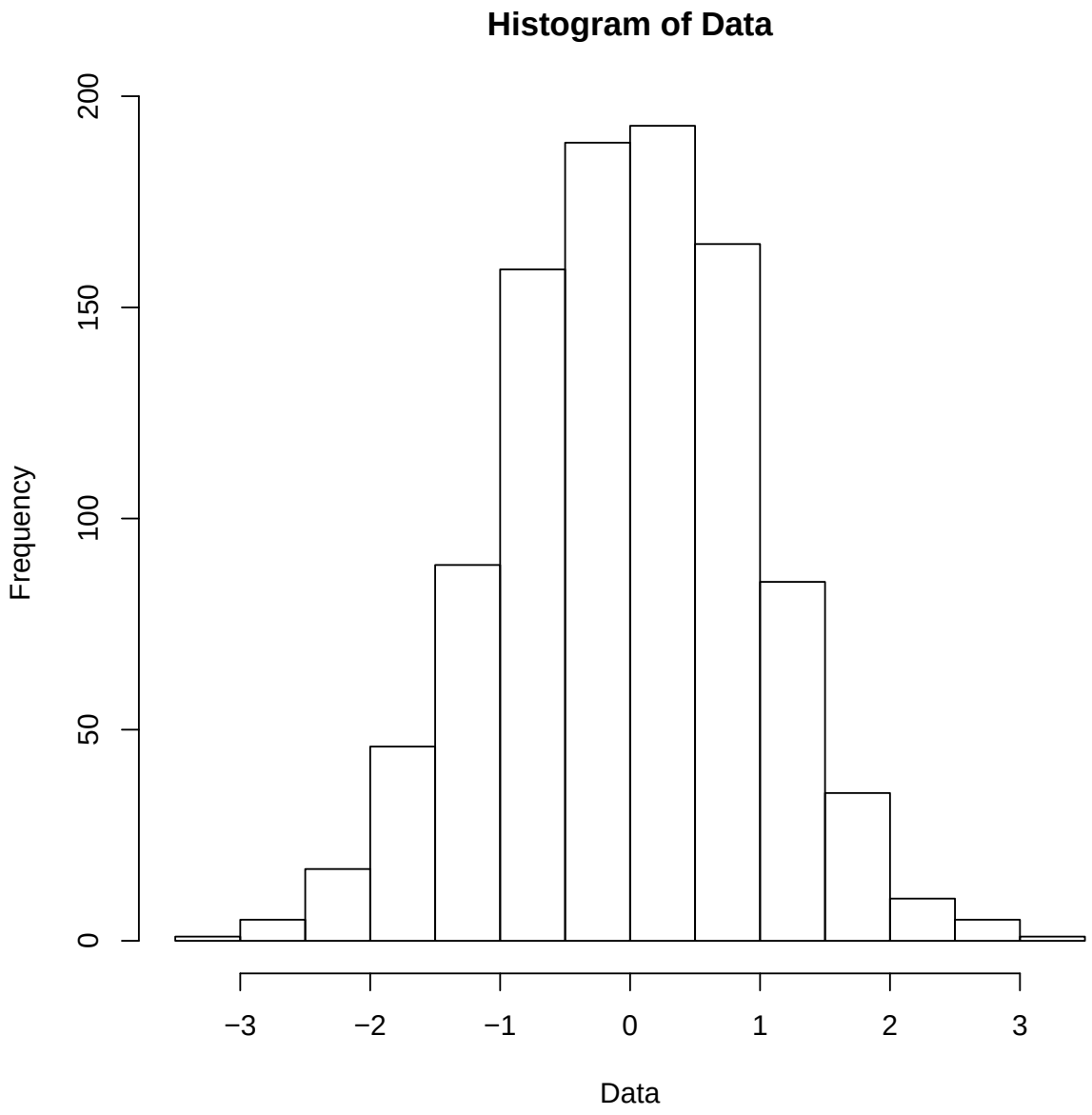


Observe que ao final deste procedimento, aparece um comando `setwd("C:/Users/etc...")` no Console. Assim, este procedimento é equivalente a simplesmente usar o comando `setwd()` (*set working directory*).

2.2 Gráficos

Os gráficos gerados no R aparecem na aba *Plots*, e podem ser exportados como imagem (JPEG, TIFF, EPS, etc) ou PDF. Como exemplo, veja o histograma de um conjunto de dados com distribuição normal:

```
Data <- rnorm(1000, mean = 0, sd = 1) # Gerando 100 obs se uma variável com distribuição
# normal padrão e chamando-a de "Data"
hist(Data) # Histograma
```



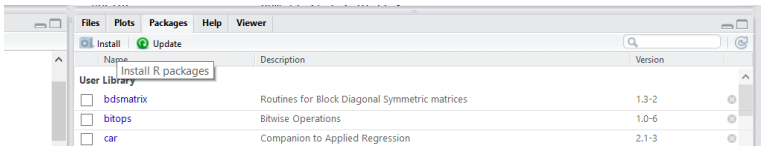
Para exportar, clique em *Export* e escolha o formato. O arquivo com o gráfico será salvo no *working directory*.

2.3 Pacotes

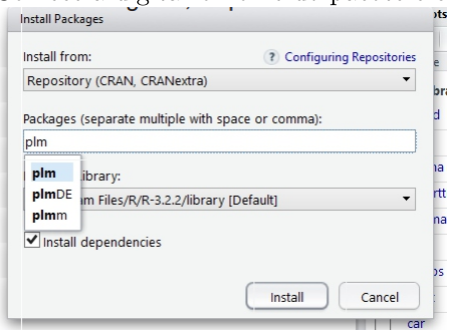
As funções básicas de estatística do R já vêm programadas. No entanto, funções mais avançadas requerem o uso de pacotes. Por exemplo, para estimar regressões via *Ordinary Least Squares* (OLS) usamos o comando `lm()` (*linear model*) que não necessita de um pacote específico. Porém, para estimações com dados em painel é necessário o pacote `plm`. Uma vez instalado o R, um pacote precisa ser *instalado* apenas uma vez. Porém, ele precisa ser *carregado* todas as vezes que o programa for aberto e o pacote precisar ser usado.

Para instalar um pacote:

1. Clique em "Install":



2. Comece a digitar o nome do pacote e clique no desejado. Em seguida, clique em "Install":



Observe que, ao final deste procedimento aparece no Console o comando `install.packages("plm")`. Portanto, este procedimeto pode ser feito digitando `install.packages("plm")` no console.

Para carregar o pacote, isto é, para fazer com que suas funções se tornem disponíveis para uso na sessão, use o comando `library(plm)`. Se o RStudio for fechado e reaberto, o comando `library()` precisa ser chamado novamente. Observe que o comando `install.packages()` requer que o nome do pacote seja digitado entre aspas e o comando `library()` não. Isto acontece porque antes de o pacote ser instalado o R não conhece o objeto `plm`, portanto, você precisa indicar o nome (caracteres), para que o R procure na internet, por exemplo, o que ele deve baixar. Já depois de instalado, `plm` é um objeto conhecido pelo R, logo as aspas não são mais necessárias.

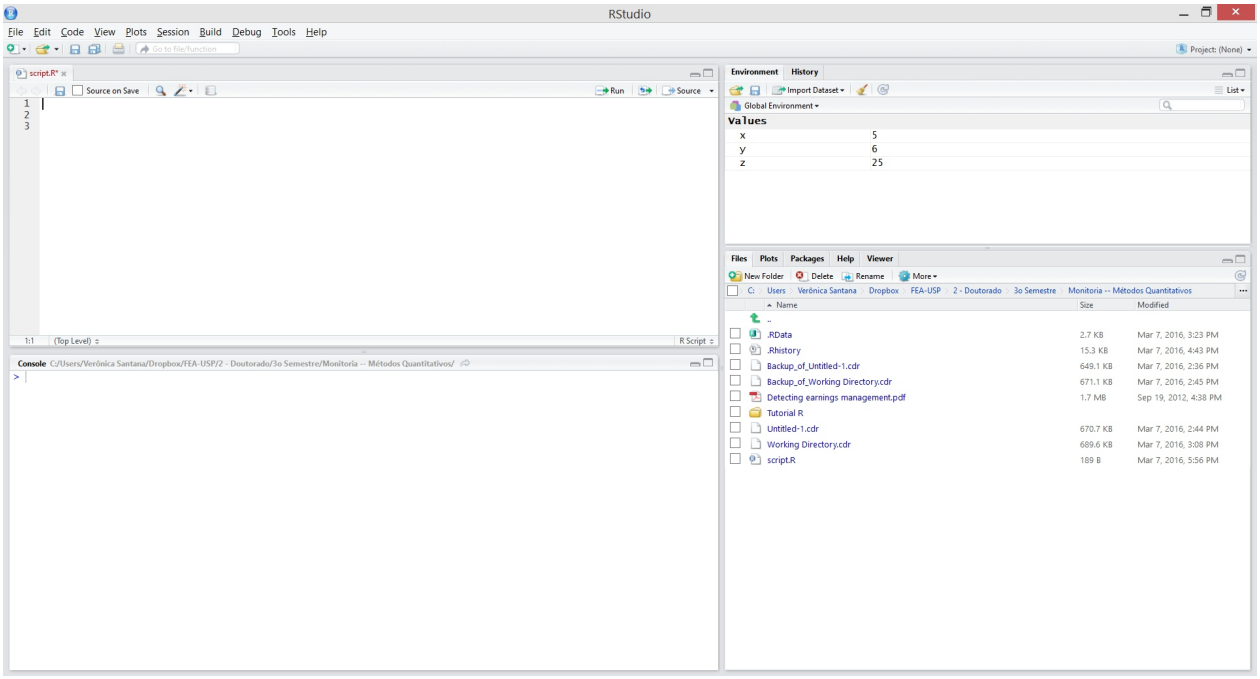
```
install.packages("plm")


## package 'plm' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Verônica Santana\AppData\Local\Temp\RtmpwLsbPU\downloaded_packages


library(plm)
```

2.4 *Scripts*

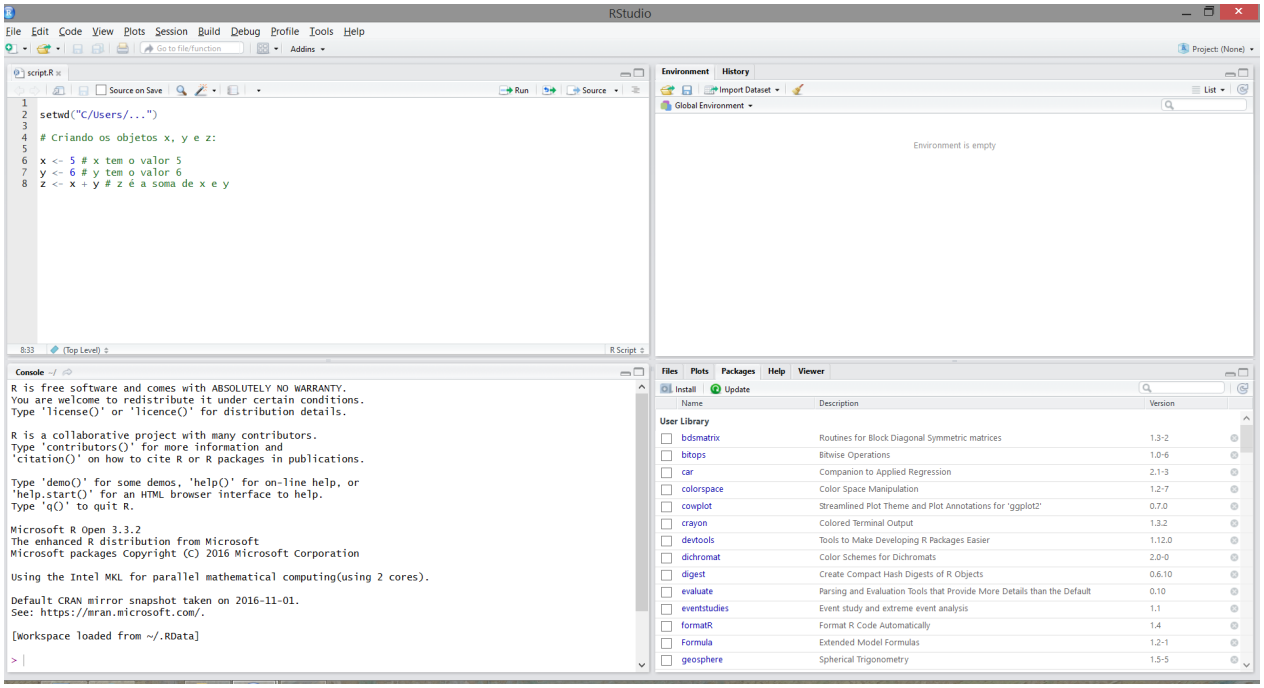
A forma mais eficiente e prática de usar o R ou o RStudio é através de um *script*. No RStudio, vá em *File* → *New File* → *R Script*. A interface agora fica dividida em 4 partes:



No *script* você pode digitar comandos a serem executados e também comentários. Os comandos são escritos como no console e tudo que é escrito após **#** são considerados apenas como comentários.

No *script*, diferentemente do Console, a tecla *Enter* apenas quebra uma linha. Em comandos muito longos, a quebra de linha é necessária para que o *script* fique visualmente compreensível. Para que o comando seja executado é necessário pressionar as teclas *Control+Enter*, ou *Control+R* no Windows. Textos podem ser copiados e colados no *script* e linhas em branco podem ser inseridas.

Veja um exemplo de *script*:

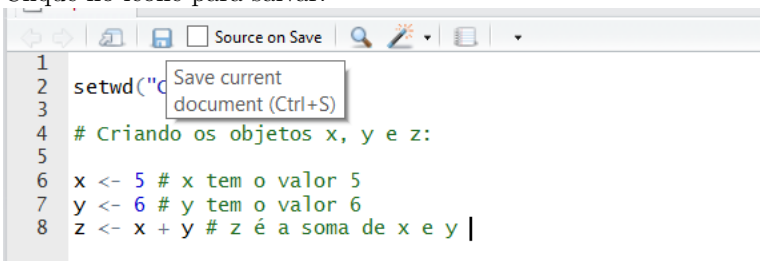


No *script*, os comandos são executados linha a linha. Assim, para criar o objeto **x** é necessário executar a linha 6 do *script*. Para isso, basta posicionar o cursor em qualquer local da linha 6 e pressionar *Control+Enter*. O

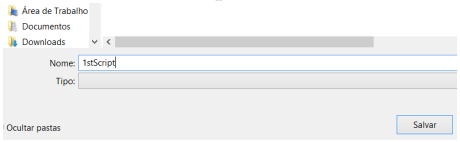
comando e o respectivo *output* aparecem no Console abaixo do *script*. Se você quiser executar as três linhas de comando de uma vez, basta selecioná-las e pressionar *Control+Enter*.

O *script* pode ser salvo da seguinte forma:

1. Clique no ícone para salvar:



2. Dê um nome e clique em Salvar:



Caso o RStudio pergunte pelo *encoding* do script, apenas clique em "OK". O arquivo será salvo no *working directory*. Após fechar o RStudio, para retomar o trabalho basta abrir o arquivo do *script* e rodar os comandos novamente. Assim, costuma ser mais prático salvar apenas o *script*, sendo, normalmente, desnecessário manter o *environment* salvo no computador. No entanto, quando o *environment* é composto por objetos criados por funções que tomam tempo, é mais viável manter o *environment* salvo.

3 Algumas Operações com Vetores e Matrizes

Usando o operador ::

```
1:5
```

```
## [1] 1 2 3 4 5
```

Criando uma sequência de 5 a 100 com incrementos de 5 unidades:

```
a <- seq(5, 100, 5) # Criando o objeto "a"
a # Vendo o objeto "a"
```

```
##  [1]   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85
## [18]  90  95 100
```

Criando repetições:

```
b <- rep(2, 10) # Número 2 repetido 10 vezes.
b
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2
```

```
c <- rep("hi!", 11) # Texto "hi!" repetido 11 vezes.
c
```

```
##  [1] "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!" "hi!"
```

Itens específicos, usando a função `c()` (concatenar).

```
d <- c(100, 34, 24, 56)
d
```

```
## [1] 100  34  24  56
```

```
e <- c(32, 9, 10, 888)
e
```

```
## [1]  32   9  10 888
```

```
f <- c("azul", "rosa", "preto", "amarelo")
f
```

```
## [1] "azul"    "rosa"    "preto"   "amarelo"
```

Combinando vetores usando as funções `cbind()` (combinando vetores por coluna) ou `rbind` (combinando vetores por linha):

```
g <- cbind(d, e)
g # Automaticamente, o nome das colunas é o nome dos vetores originais
```

```
##        d   e
## [1,] 100  32
## [2,]  34   9
## [3,]  24  10
## [4,]  56 888
```

```
h <- rbind(d, e)
h # Automaticamente, o nome das linhas é o nome dos vetores originais
```

```
##   [,1] [,2] [,3] [,4]
## d  100   34   24   56
## e   32    9   10  888
```

```
i <- c(1:3, 8)
j <- c(1:5, 2)
cbind(i,j)
```

```
## Warning in cbind(i, j):  number of rows of result is not a multiple of vector length (arg
1)
```

```
##      i j
## [1,] 1 1
## [2,] 2 2
## [3,] 3 3
## [4,] 8 4
## [5,] 1 5
## [6,] 2 2
```

```
# O R junta, porém reclama completando o que falta para formar a matriz com repetição.
# E, automaticamente, ele começa a repetir os itens do vetor de menor dimensão.
```

Criando matrizes usando o comando `matrix()`.

```
c(1:3, 8) # Vetor linha 1x4

## [1] 1 2 3 8

matrix(c(1:3, 8), 2) # Matriz 2x2.

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    8

# Dado que há 4 elementos e escolhemos 2 linhas, R já sabe que a matriz tem de ser 2x2.
# Por default, a ordenação dos elementos é feita por linha.
# Se quisermos a ordenação por colunas:
matrix(c(1:3, 8), 2, byrow = F)

##      [,1] [,2]
## [1,]    1    3
## [2,]    2    8
```

Operações com matrizes/vetores:

```
Q <- matrix(c(0, 1, 2, 5), 2) # Matriz Q 1x3
Q

##      [,1] [,2]
## [1,]    0    2
## [2,]    1    5

E <- Q/Q # Dividindo a matriz Q por ela mesma elemento a elemento:
E

##      [,1] [,2]
## [1,]  NaN    1
## [2,]    1    1

# Como 0/0 é indeterminado, o R retorna "NaN", que indica "not a number".
```

Alguns operadores:

1. `<-` ou `=`: operadores "recebe". Assim, quando fazemos `x <- 2` ou `x = 2`, indicamos que x recebe o valor 2.
2. `+`, `-`, `*`, `/`, `^`, etc, são operadores "elemento a elemento":

```
2+2

## [1] 4

4-5
```

```
## [1] -1
```

```
2*5
```

```
## [1] 10
```

```
10/4
```

```
## [1] 2.5
```

```
A <- matrix(c(1,2,0,4),2)
B <- matrix(c(2,4,1,5),2)
A+B
```

```
##      [,1] [,2]
## [1,]    3    1
## [2,]    6    9
```

```
A-B
```

```
##      [,1] [,2]
## [1,]   -1   -1
## [2,]   -2   -1
```

```
B-5
```

```
##      [,1] [,2]
## [1,]   -3   -4
## [2,]   -1    0
```

```
A*B
```

```
##      [,1] [,2]
## [1,]    2    0
## [2,]    8   20
```

```
3*B
```

```
##      [,1] [,2]
## [1,]    6    3
## [2,]   12   15
```

```
B^2
```

```
##      [,1] [,2]
## [1,]    4    1
## [2,]   16   25
```

```
A/B
```

```
##      [,1] [,2]
## [1,]  0.5  0.0
## [2,]  0.5  0.8
```

```
B/A # O R retorna divisões por 0 como "Inf".
```

```
##      [,1] [,2]
## [1,]    2  Inf
## [2,]    2 1.25
```

3. `%*%`, `%x%`, `t()`, `solve()`, `det()` são operadores matriciais:

```
# Multiplicação de Matrizes:
A%*%B

##      [,1] [,2]
## [1,]    2    1
## [2,]   20   22

A%*%A

##      [,1] [,2]
## [1,]    1    0
## [2,]   10   16

# Produto de Kronecker:
A%x%B

##      [,1] [,2] [,3] [,4]
## [1,]    2    1    0    0
## [2,]    4    5    0    0
## [3,]    4    2    8    4
## [4,]    8   10   16   20

# Transposta:
t(A)

##      [,1] [,2]
## [1,]    1    2
## [2,]    0    4

# Inversa:
solve(B)

##            [,1]       [,2]
## [1,]  0.8333333 -0.1666667
## [2,] -0.6666667  0.3333333

# Determinante:
det(B)

## [1] 6
```

4. Funções:

```
# Logaritmo natural:
log(10)

## [1] 2.302585

# Logaritmo na base 10:
log10(10)
```

```
## [1] 1

# Logaritmo em qualquer base:
log(10, 2) # base 2

## [1] 3.321928

log(10, 4) # base 4

## [1] 1.660964

# Função exponencial:
exp(2)

## [1] 7.389056

exp(log(2))

## [1] 2

# Seno:
sin(10)

## [1] -0.5440211

# Cosseno:
cos(15)

## [1] -0.7596879

# Tangente:
tan(14)

## [1] 7.244607

# Raiz Quadrada:
sqrt(16)

## [1] 4

16^(1/2)

## [1] 4

# Raiz Cúbica:
64^(1/3)

## [1] 4
```

5. Funções Vetoriais:

```r
v <- c(2, 5, 10, 11, 3)
v
```

```
## [1]  2  5 10 11  3
```

```r
# Obtendo o valor máximo:
max(v)
```

```
## [1] 11
```

```r
# Valor mínimo:
min(v)
```

```
## [1] 2
```

```r
# Comprimento do vetor:
length(v)
```

```
## [1] 5
```

```r
# Dimensão de uma matrix:
dim(A)
```

```
## [1] 2 2
```

```r
# Soma dos elementos:
sum(v)
```

```
## [1] 31
```

```r
# Soma acumulada:
cumsum(v)
```

```
## [1]  2  7 17 28 31
```

```r
# Produto dos elementos:
prod(v)
```

```
## [1] 3300
```

```r
# Produto acumulado:
cumprod(v)
```

```
## [1]    2   10  100 1100 3300
```

```r
# Média:
mean(v)
```

```
## [1] 6.2
```

```r
# Variância:
var(v)
```

```
## [1] 16.7

# Desvio-padrão:
sd(v)

## [1] 4.086563

sqrt(var(v))

## [1] 4.086563

# Mediana:
median(v)

## [1] 5

# Estatísticas Descritivas:
summary(v)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0     3.0     5.0     6.2    10.0    11.0

# Ordenação do vetor
sort(v)

## [1]  2  3  5 10 11

sort(v, decreasing = T)

## [1] 11 10  5  3  2

-sort(-v)

## [1] 11 10  5  3  2

#
order(v)

## [1] 1 5 2 3 4
```

6. Operadores lógicos:

```
5 > 2

## [1] TRUE

2 > 5

## [1] FALSE

2 >= 2
```

```
## [1] TRUE

2 >= 1

## [1] TRUE

2 >= 5

## [1] FALSE

2 <= 5

## [1] TRUE

2 == 2

## [1] TRUE

2 == 3

## [1] FALSE

2 != 3

## [1] TRUE

2 != 2

## [1] FALSE

# Duas condições ao mesmo tempo usando o operador &, "e":
2 == 2 & 2 <= 3

## [1] TRUE

2 == 2 & 2 <= 1

## [1] FALSE

# Duas condições alternativas usando o operador |, "ou":
2 == 2 | 2 <= 1

## [1] TRUE

3 != 4 | 3 >= 4

## [1] TRUE

3 != 3 | 3 >= 4

## [1] FALSE
```

7. Função `if()...else()` e `ifelse()`:

```
if(2 > 3){"verdade!"} else{"mentira!"}
```

```
## [1] "mentira!"
```

```
if(2 < 3){"verdade!"} else{"mentira!"}
```

```
## [1] "verdade!"
```

```
ifelse(2 > 3, "verdade!", "mentira!")
```

```
## [1] "mentira!"
```

```
ifelse(2 < 3, "verdade!", "mentira!")
```

```
## [1] "verdade!"
```

8. Acessando elementos com `[]`:
Dada uma matriz $A_{3\times 2}$:

```
A <- matrix(c(1, 1, 1, 2, 4, 20),3)
A
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    4
## [3,]    1   20
```

```
# Para acessar o elemento (2,1):
A[2,1]
```

```
## [1] 1
```

```
# Para acessar o elemento (2,2):
A[2,2]
```

```
## [1] 4
```

```
# Para acessar toda a coluna 1:
A[,1]
```

```
## [1] 1 1 1
```

```
# Para acessar toda a linha 3:
A[3,]
```

```
## [1]  1 20
```

```
# Para acessar os elementos das linhas 2 a 3 e da coluna 2:
A[2:3, 2]
```

```
## [1]  4 20

# Para acessar todos os elementos da matriz A que são maiores que 3:
A[A > 3]

## [1]  4 20
```

Dado um vetor:

```
a <- c(177, 180, 165, 250)
a

## [1] 177 180 165 250

# Acessando o segundo elemento do vetor
a[2]

## [1] 180

# Acessando os elementos 2 a 4:
a[2:4]

## [1] 180 165 250

# Acessando os elementos 1 e 3:
a[c(1, 3)]

## [1] 177 165

# Acessando todos os elementos maiores que 175:
a[a > 175]

## [1] 177 180 250
```

4 *swirl*

`swirl` é um pacote do R construído para transformar o console em uma ferramenta interativa para aprender R. Para entender melhor do projeto, veja `http://swirlstats.com/`. Em `http://swirlstats.com/students.html` são dados os detalhes sobre como usar o swirl. Uma vez intalado e carregado o pacote, você é levado a efetuar tarefas:

```
> library(swirl)

| Hi! I see that you have some variables saved in your workspace. To keep things running smoothly, I
| recommend you clean up before starting swirl.

| Type ls() to see a list of the variables in your workspace. Then, type rm(list=ls()) to clear your
| workspace.

| Type swirl() when you are ready to begin.

> rm(list = ls())
> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use the same name as you did then. If
| you are new, call yourself something unique.

What shall I call you? Veronica

| Thanks, Veronica. Let's cover a couple of quick housekeeping items before we begin our first
| lesson. First of all, you should know that when you see '...', that means you should press Enter
| when you are done reading and ready to continue.

...  <-- That's your cue to press Enter to continue
```

O swirl dá acesso às tarefas de cursos de R que estão disponíveis também no Coursera, como o *R Programming: The basics of programming in R*, em `https://pt.coursera.org/learn/r-programming`. Além deste, estão disponíveis no *swirl*: *Regression Models: The basics of regression modeling in R*, *Statistical Inference: The basics of statistical inference in R*, e *Exploratory Data Analysis: The basics of exploring data in R*.

5 Buscando ajuda

Para obter ajuda sobre o funcionamento de comandos, a forma mais prática é usar a ajuda interna que acessa a documentação dos pacotes. Por exemplo:

```
?plot # ou
help(plot)
```

No entanto, para aqueles que ainda não estão acostumados com a linguagem, essa documentação pode parecer pouco amigável. A documentação dos pacotes é baixada na instalação, e é possível acessar o pdf diretamente. A página no CRAN do pacote `plm`, por exemplo, contém informações sobre a versão do pacote, os requisitos, os autores, etc: `https://cran.r-project.org/web/packages/plm/index.html`, enquanto na parte de downloads estão o *Reference manual*: `https://cran.r-project.org/web/packages/plm/plm.pdf` e as *Vignettes*: `https://cran.r-project.org/web/packages/plm/vignettes/plm.pdf`.

Além da documentação dos pacotes, existem alguns livros que podem ser muito úteis, como o *R Cookbook*.

No entanto, na maioria das vezes a forma mais prática de conseguir ajuda com uma dúvida específica é a busca em fóruns na internet, como o *Stack Overflow*:

Além disso, alguns blogs fornecem tutoriais, como o `https://www.r-bloggers.com/`. Um exemplo no R-bloggers: `https://www.r-bloggers.com/computing-and-visualizing-pca-in-r/`. O site *Quick-R* é particularmente bom para tratar de assuntos mais simples, como gráficos: `http://www.statmethods.net/graphs/line.html`.

Algumas páginas do Facebook também são interessantes para fornecer tutorias, dicas e curiosidades: `https://www.facebook.com/rtipaday/`, `https://www.facebook.com/rcomputing/`, `https://www.facebook.com/The-R-Project-for-Statistical-Computing-155948597756419/`.