Quick Tour of R and R Studio

Robert Olendorf

February 12, 2106

Figure 1: Screen shot from the typical default layout.

R Studio

R Studio is an Integrated Development Environment (IDE) for the R Statistical Package. You can run R fine from the command line and using text editors such as Notepad++ or Textmate, but it provides a nice comprehensive view of your current project. It's also probably easier for beginners to learn on.

When you open R Studio you should get something like the screenshot shown in figure 1. **Section A** is where your scripts and other documents are displayed. **Section B** is the ***Console***, where you do your interactive programming. **Section C** contains several tabs. It will usually contain at least your ***Environment***, a listing of all your objects (ie. variables, data) and a ***History*** tab for your ***Console*** input. **Section D** contains a number of tabs. Minimally it will contain a ***Files*** tab, a ***Plots*** tab, a ***Packages*** tab, and a ***Help*** tab. The ***Files*** tab allows you to easily navigate the directory structure of your project. The ***Plots*** tab displays previews of plots you generate. The ***Packages*** tab lets you easily manage your R packages. The ***Help*** tab displays the help.

Setting Up

Our first task is to define a default working direcotry. This is where R will start when you are not working on a project. Opening and saving files will be relative to this directory. Create a directory in your *Documents* called *rprojects*. You can of course choose your own location. Choose [**Preferences**] from the [**RStudio**] menu. Our first step will be to define a default working directory. This is where R will start, and save files if you are not working on a project. Choose [**Preferences**] the [**RStudio**] menu and choose the [**General**] tab. The top options edit or browse to set the default working directory to *rprojects*.

Next choose the [**Code**] tab. Check **Insert spaces for tab** and set it to 2. I also like to have **Insert matching parens/quotes** turned on.

Got the [**Appearance**] tab and choose an *Editor Theme*. I like *Solarized Dark* or *Monokai* but just make sure its one that works well for you.

Unless you have a reason to change it, leave the **Pane Layout** at the defaults. It will work better with this and other tutorials.

Leave the **Packages** as the default values.

In the **Sweave** tab set *Weave Rnw files usign:* to ***Knitr***. *Typeset LaTeX into PDF using:* ***pdfLaTeX***. Make sure *Clean auxilliary output after compile* and *Insert numbered sections and subsections* are checked. ... using ***RStudio Viewer***.

Leave the **Spelling** tab at the defaults.

I usually do my Git commands on a separate terminal window. However, sometimes you need to use the RStudio Git integration. To set this up in the **Git/SVN** make sure the Git executable is set up. This may vary with operating system. Also create an SSH key if you haven't already, or add the path to it if you have one.

Make sure *Enable publishing apps and documents* is checked in the **Publishing** tab.

R Basics

Command Line

R is a command line tool. This gives many people stress, but don't let it stress you. First learn the syntax. Then learn the commands you use most often. Also, don’t hesitate to Google something. I often do, especially when its something I don’t do a lot. Don’t waste time fiddling. First off, lets do something easy. Lets do some math.

```
1 + 1  # R will just print the answer

## [1] 2

# However, we rarely want to do this. Lets set it to a variable.
a <- 1 + 1  # a is the variable, we use the less than and dash
            # rather than equals
a # Notice the value isn't displayed until we enter the variable

## [1] 2

  # again
b = 1 + 1 # Equals still works, but its not preferred.
```

Variables can and function names may consist of letters, numbers, dots and underlines. They must start with a letter or a dot not followed by a number. Starting with a dot is valid, but not good style.

```
a <- 1  # valid
b.1 <- 2 # valid
b1 <- 3 # valid
b_1 <- 4 # valid
.b <- # valid
1b <- 5 # invalid
.1 <- # invalid
```

Rists tend to use the dot as a namespacing convention. However, there is no true significance to the dot in the name. R treats it as just another character.

Now look in your **Environment** tab. You will see the ***a*** and ***b*** variables in there. Choose the **History** tab. You will see both commands recorded. Its good to get in the habit of building a script of your work. So lets do that. Do [**File/New File/R Script**]. Now select those two lines(Shift Select) and click **To Source**. It should be in your script now. This is a good way to build up scripts. Play around in the console to get it working, then copy the command(s) to a script. You may have to tweak the script still, but its an efficient way to go.

Datatypes

Lets get some basics out of the way before we start building documents. We need to understand R Datatypes, Functions and learn some of the most basic functions first.

Scalars

R comes with the typical scalar datatypes shown below.

```
5 # an integer (numeric)
```

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

```
5.5 # float (numeric)
```

```
## [1] 5.5
```

```
"a" #character
```

```
## [1] "a"
```

```
"a string"  # a string (a vector of characters)
```

```
## [1] "a string"
```

```
FALSE # logical
```

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

Vectors

```
num.vec <- c(1, 2, 4, 5)  # creating a vector the hard way
num.vec
```

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

```r
  num.vec2 <- 1:5 # automating a vector creation
  num.vec2
```

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

```r
  num.vec3 <- seq(1, 5, 0.5)
  num.vec3
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```r
  alpha.vec <- c("a", "b", "c", "d")  # a vector of charaters
  alpha.vec
```

```
## [1] "a" "b" "c" "d"
```

```r
  # a vector of logicals
  logical.vec <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
  logical.vec
```

```
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

```r
  ##
  # A vector must be of the same type.
   # The integer gets converted to a character
  test.vec <- c("1", "2", 3)
  test.vec
```

```
## [1] "1" "2" "3"
```

You can access parts of vectors using the following methods.

```r
  v <- seq(1, 20, 2)
  v
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

```r
  # Get 1 item. Notice the index starts with 1 rather than
  # 0 as in other languages.
  v[1]


## [1] 1


  v[2:5]  # Get a slice


## [1] 3 5 7 9


  v[c(1, 5, 9)]  # Get specific indices


## [1]  1  9 17
```

Matrixes

Matrices are two dimensional vectors. Columns (but not row) in a matrix must have the same datatype (numeric, character, etc.).

```
  # Create a 5 x 4 matrix. R fills by column.
  matrix.1 <- matrix(1:20, nrow=5, ncol=4)
  matrix.1


##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20


  # You can fill by rows too

  matrix.row <- matrix(1:20, nrow = 5, ncol=4, byrow = TRUE)
  matrix.row


##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

```
  # If you don't fill the matrix R repeats
  # the sequence until its filled. It doesn't
  # have to be an even fill, but you will get a warning.
  matrix.2 <- matrix(1:7, nrow=5, ncol=4)
```

```
## Warning in matrix(1:7, nrow = 5, ncol = 4):  data length [7] is
not a sub-multiple or multiple of the number of rows [5]
```

```
  matrix.2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6    4    2
## [2,]    2    7    5    3
## [3,]    3    1    6    4
## [4,]    4    2    7    5
## [5,]    5    3    1    6
```

Accessing a matrix is as follows.

```
  matrix.1[,3]  # 4th column
```

```
## [1] 11 12 13 14 15
```

```
  matrix.1[2,]  # 2nd row
```

```
## [1]  2  7 12 17
```

```
  matrix.1[2:4, 1:3] # rows 2, 3 ,4 of colunmns 1, 2, 3
```

```
##      [,1] [,2] [,3]
## [1,]    2    7   12
## [2,]    3    8   13
## [3,]    4    9   14
```

Lists

Lists are an ordered collection of objects. They don't have to be the same type.

```r
# A list with a string, numeric, logical and a vector
# using a variable to insert the vector.
list.1 <- list("Fred", 24, FALSE, num.vec)
list.1
```

```
## [[1]]
## [1] "Fred"
##
## [[2]]
## [1] 24
##
## [[3]]
## [1] FALSE
##
## [[4]]
## [1] 1 2 4 5
```

```r
# Named list components
list.2 <- list(name="Sue", age=32, smoker=TRUE, data=num.vec2)
list.2
```

```
## $name
## [1] "Sue"
##
## $age
## [1] 32
##
## $smoker
## [1] TRUE
##
## $data
## [1] 1 2 3 4 5
```

```r
# We can make a vector of lists

list.vect <- c(list.1, list.2)
list.vect
```

```
## [[1]]
## [1] "Fred"
##
## [[2]]
## [1] 24
##
## [[3]]
## [1] FALSE
##
## [[4]]
## [1] 1 2 4 5
##
## $name
## [1] "Sue"
##
## $age
## [1] 32
##
## $smoker
## [1] TRUE
##
## $data
## [1] 1 2 3 4 5
```

We can access parts of a list like this.

```
  # Notice the double bracket.
  list.2[[2]]  # the second entry

## [1] 32

  list.2[["name"]]

## [1] "Sue"
```

Data Frames

Data frames are something fairly unique to R. It is usually what you will use to contain tabuluar data. They are a bit like matrices, but each column can have a different datatype.

```
  a <- c(1:4)
  b <- c("George", "Tom", "Phineas", "Ferb")
  c <- c(TRUE, FALSE, FALSE, TRUE)

  df.1 <- data.frame(a, b, c)
  df.1
```

```
##   a       b     c
## 1 1  George  TRUE
## 2 2     Tom FALSE
## 3 3 Phineas FALSE
## 4 4    Ferb  TRUE
```

```
  # We can name the columsn as well.
  names(df.1) <- c("ID", "name", "smoker")
  df.1
```

```
##   ID    name smoker
## 1  1  George   TRUE
## 2  2     Tom  FALSE
## 3  3 Phineas  FALSE
## 4  4    Ferb   TRUE
```

We can access parts of a data frame

```
  df.1[2:4]  # columuns 2 - 4
```

```
## Error in `[.data.frame`(df.1, 2:4):  undefined columns selected
```

```
  df.1[c("ID", "name")]  # columns by name
```

```
##   ID    name
## 1  1  George
## 2  2     Tom
## 3  3 Phineas
## 4  4    Ferb
```

```
  df.1$ID  # variable ID (or column ID)
```

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

Factors

Sometimes our variable is categorical (nominal). WE can tell R this by creating a factor.

```
  color <- c(rep("red", 21), rep("green", 14), rep("blue", 18))
  head(color)
```

```
## [1] "red" "red" "red" "red" "red" "red"
```

```
  color.factor <- factor(color)
  head(color.factor)
```

```
## [1] red red red red red red
## Levels: blue green red
```

```
  # Summary will work on color.factor but not color
  summary(color)
```

```
##    Length     Class      Mode
##        53 character character
```

```
  summary(color.factor)
```

```
##  blue green   red
##    18    14    21
```

Functions

Functions in R look like ***mean()***. You've already seen some of them being used. We use them like this.

```
    mean(1:10) # Gets the mean of the vector of values
```

```
## [1] 5.5
```

```
    mean() # Will throw a "useful" error message
```

```
## Error in mean.default():  argument "x" is missing, with no default
```

```
mean # will show the source code or somethign similar
```

```
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x7faf60e01748>
## <environment: namespace:base>
```

Useful Functions

getwd()
Gets the current working directory.

setwd()
Sets teh current working directory.

help()
Gets help for something (eg help(”mean”))

data()
Lists currently installed demo data sets.

A Quick Analysis

Just as an example we can do a simple analysis. We are going to use the built in *cars* dataset. First click the ***broom*** icon in your ***Environment*** tab. For a real analysis we would also be saving useful code to an R Script or similar document, but we'll skip that for now.

First we'll load the data set and get to know the data a little.

```
data("cars")  # load the data

# "Peak at the data to see what it looks like
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
  help(cars)  # Look at the help.
```

We see that the *cars* data set is just two columun, speed and distance. First lets get look at the data itself before we analyze it.

```
    summary(cars) # Simple descriptive statistics
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

```
    # Boxplots are good to see the distribution
    # and detecting outliers. This looks good.
    boxplot(cars, notch=TRUE)
```

figure/cars-boxplot-1.pdf

```r
# We can also do histograms.
hist(cars$speed)
```

figure/cars-histogram1-1.pdf

```r
hist(cars$dist)
```

figure/cars-histogram2-1.pdf