A Quick Introduction to R and RStudio

Technical Report · November 2017

DOI: 10.13140/RG.2.2.10401.92009

Vlad Krotov
Murray State University
45 PUBLICATIONS **620** CITATIONS

11/17/2017

A Quick Introduction to R and RStudio®

Tutorial

© Vlad Krotov
vladkrotov@gmail.com

ABSTACT

This tutorial provides a brief introduction to R language and RStudio environment. This resource is intended for those who have a basic understanding of programming, but no previous exposure to R. First, the tutorial explains the main features of R language that make it suitable for data analysis. Second, this tutorial explains how to use RStudio®, an Integrated Development Environment (IDE) for R language, for entering and running R code. Third, the tutorial goes over the main elements of R syntax. The tutorial also demonstrates how to install and load and R packages. The tutorial ends with a list of recommended readings for those wishing to learn more about R.

LEARNING OUTCOMES

After completing this tutorial, you should be able to:

- Explain how R language is used for analyzing data
- Use RStudio for entering and running R-code
- Add comments to R code
- Use some utility functions in R, such as setwd() and getwd()
- Retrieve help information for R functions
- Perform simple arithmetic calculations in R
- Declare variables of different data types in R and determine data type of a variable
- Create and access data frames
- Write and analyze conditional statements and loops in R
- Install and load R packages

R LANGUAGE OVERVIEW

R is a programming language used for data manipulations, statistical analysis, and data visualization (CRAN 2017). Just like any programming language, R includes conditional statements, recursive functions, and input/output commands. But unlike a typical programming language, R includes the following features that make the language especially suited for data science (CRAN 2017):

- Numerous functions for data retrieval, manipulation and storage
- Built-in operators for calculations using arrays and matrices
- Built-in collection of tools for basic and intermediate statistical analysis (e.g. descriptive statistics, regression, PLS, etc.)
- Graphics packages for data analysis and visualization for producing both on-screen and hard-copy visualizations
- The ability to use R to run code written in C, C++, and Fortran (this is done for improving processing time)
- The ability to interface with code written in Python - another popular language for data science

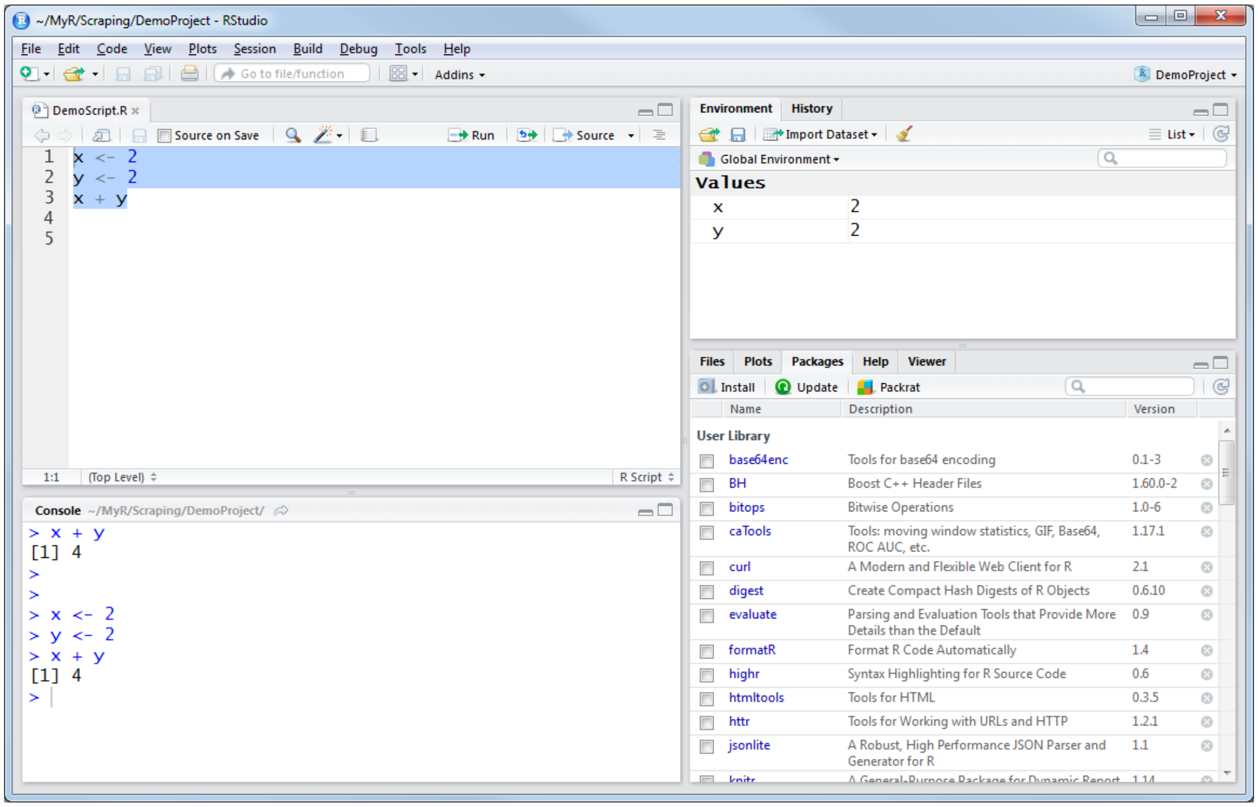
Since its original release in 1992 by Ross Ishaka and Rob Gentleman, two statisticians from the University of Auckland in New Zealand, the R language has grown tremendously in terms of its functionality and use. This can be seen by going through the resources provided online by the Comprehensive R Archive Network (CRAN) (https://cran.r-project.org/). The website hosts R project documentation, precompiled binary distributions of the R language for various operating systems, source code, R manuals, and even a scholarly journal called *The R Journal*.

Of special importance are 11, 850 (as of November 2017) user-contributed packages (https://cran.r-project.org/web/packages/). These packages are free add-ons developed by the R community members that implement various data analysis and visualization techniques. With these packages, one can perform myriad forms of data analysis, ranging from simple data retrieval (e.g. *dicecrawler* package) and manipulations (e.g. *plyr* package) to advanced data science methods, such as text mining (e.g. *tm* package) and numerous machine learning algorithm implementations (e.g. *FCNN4R* package). Of special importance are graphical packages (e.g. *ggplot2* package) that contain extensive collections of graphics elements that allow users to automatically generate professional looking data visualizations with complete control over all the graphical elements and their appearance.

With thousands of R packages are available, it is not surprising that R is increasingly becoming a de facto language of data science. Unlike many of the commercial statistical packages that are geared towards a particular audience (e.g. SPSS is geared towards statistical analysis of quantitative data in business and social sciences), R is used across many industries and scientific disciplines (Lander 2014).

RSTUDIO ENVIRONMENT

Once R language is installed using an appropriate distributive from CRAN, the R language can be run using a console. Yet most people prefer to install an Integrated Development Environment (IDE) on the top of the R base installation. While there are many R IDEs, RStudio seems to be the most popular (https://www.rstudio.com/). RStudio includes a free version as well as several enterprise products (e.g. RStudio Commercial Desktop and RStudio Server Pro) that require an annual license fee. The free version of RStudio makes the R language much easier to use and facilitates development of advanced R scripts. RStudio includes a code editor together with advanced debugging and visualization tools. RStudio has many useful and free add-ons. For example, *Shiny* server provides a framework for building web applications using R without the knowledge of HTML, CSS, or JavaScript. RStudio's Graphical User Interface (GUI) can also be integrated with the *knitr* package (available from CRAN). The package *knitr* can be used in conjunction with the so-called *R Markdown* language (see http://rmarkdown.rstudio.com/) to dynamically generate reports in HTML, PDF, Word, and even PowerPoint formats. These reports mesh together R code, user text as well as output from R code execution (both numerical output and data visualization). A typical layout of RStudio IDE is shown in Figure 1 below:



```
x <- 2
y <- 2
x + y
```

```
> x + y
[1] 4
>
>
> x <- 2
> y <- 2
> x + y
[1] 4
>
```

Values

x	2
y	2

Name	Description	Version
User Library		
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.60.0-2
bitops	Bitwise Operations	1.0-6
caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.	1.17.1
curl	A Modern and Flexible Web Client for R	2.1
digest	Create Compact Hash Digests of R Objects	0.6.10
evaluate	Parsing and Evaluation Tools that Provide More Details than the Default	0.9
formatR	Format R Code Automatically	1.4
highr	Syntax Highlighting for R Source Code	0.6
htmltools	Tools for HTML	0.3.5
httr	Tools for Working with URLs and HTTP	1.2.1
jsonlite	A Robust, High Performance JSON Parser and Generator for R	1.1
knitr	A General-Purpose Package for Dynamic Report	1.14

Figure 1. RStudio GUI

In the top left corner of the screen one can see a script editor window. Within this pane one can edit his or her R script. The script can be run by selecting the lines that one wishes to be executed with a mouse and then pressing the "Run" button at the top of the script editing pane.

The results of the script execution together with the script lines that generated these results will be displayed in the Console window located in the bottom left corner of the screen. The top right pane of the screen provides information about the variables and data structures used or generated by the script. This is the so-called “Environment” window. The window on the bottom right corner of the screen shows information about the files and packages used by the project and allows one to view plots (or visualizations) generated by R and also access help about various elements of R syntax.

R LANGUAGE BASICS

Although R syntax is similar to any procedural programming language, it also has many featured of the Object Oriented Programming (OOP) paradigm (Wickham 2011). Most data analysis scripts in R are written using the procedural programing style: each line of code represents a particular command and, overall, the progression of the code is quite linear in nature. There are some particularities of R that may not be that common among other procedural languages. Some of these particularities are related to the language syntax, while others are related to the fact that R was developed specifically for data analysis. Some of these particularities are discussed in this section using some examples from Lander (2014).

Most of the examples in this section have the following format:

```
> Some R code
Some system message
[1] Some system output related to the R code
```

Please note that the “greater” sign before the comment (or an angular bracket) denoted with “>” is not a part of the code. This is how RStudio console invited to enter a line of code. Each line of code entered in the console (either directly or by using the “Run” button to execute several lines at a time) is marked by “>” as well to separate R code from its output, which is marked with [1]. Also, a “+” character is used by the system to show that a particular line of code is a part of a bigger language construction (e.g. a nested if statement).

Comments in R code are marked with a hash tag (#):

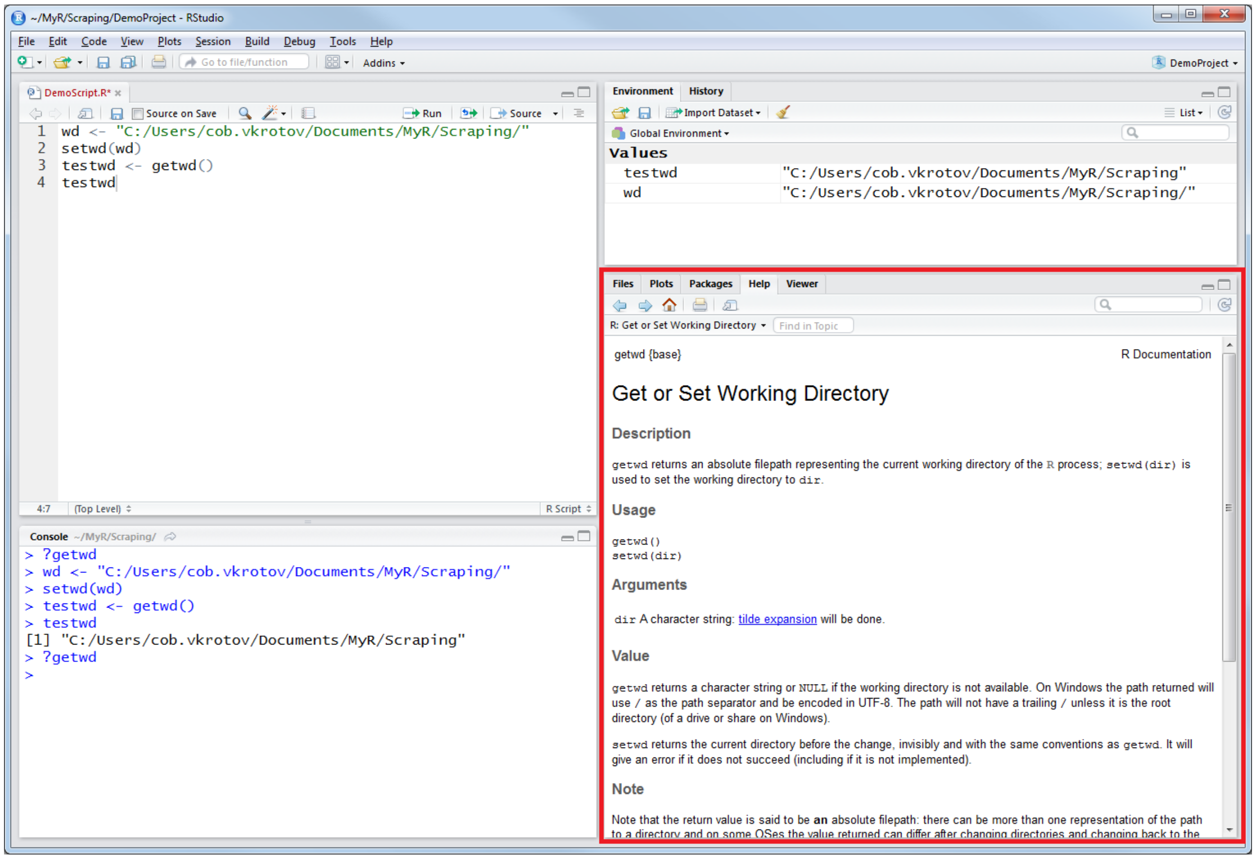
```
> #This is a comment
```

Another useful command that allows one to get started with his or her R project is for getting and setting a working directory where all the script and all the file outputs will be saved by default. The command below retrieved the current working director for R and saves it into a variable called `wd`:

```
> wd <-  getwd()
> wd
[1] "C:/Users/Documents/MyR/Scraping/DemoProject"
```

Please note that a forward slash “/” is used in the working directory address and not a backslash “\” which is used in the Windows operating system.

Help in relation to this or any other function can be accessed by typing “?” before a function (see the help window highlighted with red in the Figure 2 below):



```
wd <- "C:/Users/cob.vkrotov/Documents/MyR/Scraping/"
setwd(wd)
testwd <- getwd()
testwd
```

```
Values
  testwd        "C:/Users/cob.vkrotov/Documents/MyR/Scraping"
  wd            "C:/Users/cob.vkrotov/Documents/MyR/Scraping/"
```

```
> ?getwd
> wd <- "C:/Users/cob.vkrotov/Documents/MyR/Scraping/"
> setwd(wd)
> testwd <- getwd()
> testwd
[1] "C:/Users/cob.vkrotov/Documents/MyR/Scraping"
> ?getwd
>
```

getwd {base}   R Documentation

Get or Set Working Directory

Description

getwd returns an absolute filepath representing the current working directory of the R process; setwd(dir) is used to set the working directory to dir.

Usage

```
getwd()
setwd(dir)
```

Arguments

dir A character string: tilde expansion will be done.

Value

getwd returns a character string or NULL if the working directory is not available. On Windows the path returned will use / as the path separator and be encoded in UTF-8. The path will not have a trailing / unless it is the root directory (of a drive or share on Windows).

setwd returns the current directory before the change, invisibly and with the same conventions as getwd. It will give an error if it does not succeed (including if it is not implemented).

Note

Note that the return value is said to be **an** absolute filepath: there can be more than one representation of the path to a directory and on some OSes the value returned can differ after changing directories and changing back to the

Figure 2. Retrieving Help in R

Similar, the following code sets “Scraping”, a higher level directory, as a working directory and then tests the outcome of the operation with the getwd() function:

```
> wd <- "C:/Users/Documents/MyR/Scraping/"
> setwd(wd)
> testwd <- getwd()
> testwd
[1] "C:/Users/Documents/MyR/Scraping"
```

Basic arithmetic operations can be performed in R by entering numerical equations directly using the common arithmetic operating signs:

```
> 2 + 2
[1] 4
> 2 + 2 + 1
[1] 5
> 2*2
[1] 4
> 2*3*4
[1] 24
> 24/2
[1] 12
> 11/2
[1] 5.5
```

Please note that the mathematical operators are executed using the following order: Parenthesis, Exponents, Multiplication, Division, Addition and Subtraction (PEMDAS) (Lander 2014). This is why the two mathematical expressions below produce different results:

```
> 2*3+5
[1] 11
> 2*(3+5)
[1] 16
```

Unlike some highly structured programming languages as C++, R does not require to declare variable types before these variables are used:

```
> x <- 2
> y <- "Hello, World!"
> x
[1] 2
> y
[1] "Hello, World!"
```

Please note that the equal sign operator “=” can also be used to assign values to variables. Yet it is recommended to use the “<-“ or “arrow” operator to assign valued to variables. Unlike the “equal sign” The “arrow” operator is more indicative of the assignment operation and eliminated confusion when assignment operators are used together with comparison operator “==”:

```
> x <- 2
> y <- 2
> x == y
[1] TRUE
```

There are many data types that variables in R can store. The main types are numeric, character (same as string in other programming languages), date, and logical (TRUE/FALSE) (Lander 2014). In addition to that, variables in R can store R objects, such as a function or a graphical plot. One can check variable type using the class() function:

```
> x <- 2
> class(x)
[1] "numeric"
> s <- "Some String"
> class(s)
[1] "character"
> t <- TRUE
> f <- FALSE
> class(t)
[1] "logical"
> class(f)
[1] "logical"
> d <- as.Date("1900-01-01")
> class(d)
[1] "Date"
```

Vector is a data structure in R storing elements of the same data type. For example, the following vector (numvector) consists of six numbers being joined together into one single array:

```
> numvector <- c(2,3,4,3,2,19)
> numvector
[1]  2  3  4  3  2 19
```

And this (charector) consists of three words or strings being joined together into one single array:

```
> charvector <- c("Paris", "New York", "Tokyo")
> charvector
[1] "Paris"    "New York" "Tokyo"
```

Please note that R is a "vectorized language" (Lander 2014). This means that operators are automatically applied to all elements of the vector without the need to loop through all the elements:

```
> numvector <- c(2,3,4,3,2,19)
> numvector
[1]  2  3  4  3  2 19
> numvector - 2
[1]  0  1  2  1  0 17
> mean(numvector)
[1] 5.5
```

R language also contains a number of advanced data structures, such as matrices and data frames. Data frames are by far the most useful and most widely used advanced data structures in R. One can think of a data frame as a table or an Excel sheet containing data. This table can consists of many columns and row. Each column can be viewed as a vector of different data type. Each column in a data frame can also have its own name or label.

For example, the code below creates a 3x3 data frame consisting of the following columns of three columns (“City Name”, “Temperature in Celsius”, and “Sky Condition”):

```
> x <- c("Paris", "New York", "Tokyo")
> y <- c(10, -2, 8)
> z <- c("Sunny","Partially Cloudy", "Cloudy")
> weather <- data.frame("City" = x, "Temperature" = y, "Condition" = z)
> weather
      City Temperature        Condition
1    Paris          10            Sunny
2 New York          -2 Partially Cloudy
3    Tokyo           8           Cloudy
```

Data frames are complex objects with many attributes and operators that can be applied to them. For example, one can check number of rows and number of columns in a data frame and access individual columns or vectors within a data frame using the “$” notation:

```
> nrow(weather)
[1] 3
> ncol(weather)
[1] 3
> weather$Temperature
[1] 10 -2  8
> weather$Condition
[1] Sunny Partially Cloudy Cloudy
Levels: Cloudy Partially Cloudy Sunny
```

Individual values within the data frame can also be accessed using the [column, row] notation:

```
> weather[1,3]
[1] Sunny
Levels: Cloudy Partially Cloudy Sunny
> weather[3,1]
[1] Tokyo
Levels: New York Paris Tokyo
```

In addition to data frames, R supports other types of complex data structures, such as lists, matrices and arrays.

Finally, loops are implemented in R as follows:

```
> for (index in 1:5)
+ {
+     print(index)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
>
> index <- 1
> while (index <= 5)
+ {
+     print(index)
+     index <- index + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

As can be seen from the output, both for and while loops perform the same function.

Please note that in R one can also loop through vectors:

```
> numvector <- c(1,2,3,4,5)
>
> for (index in numvector)
+   {
+       print(index)
+   }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
>
> charvector <- c("Julie", "Jane", "Janice", "Jordan", "Jamie")
>
> for (index in charvector)
+   {
+   print(index)
+   }
[1] "Julie"
[1] "Jane"
[1] "Janice"
[1] "Jordan"
```

```
[1] "Jamie"
```

A basic if statement has the following structure in R:

```
if (test_expression) {
  statement
}
```

For example, the following code will check whether x is a negative number:

```
> x <- 1
> if(x > 0){
+   print("x is positive")
+ }
[1] "x is positive"
```

An if…else statement has the following structure:

```
if (test_expression) {
   statement1
} else {
   statement2
}
```

For example, the following code will determine whether x is positive or negative:

```
> x <- -7
> if(x > 0){
+   print("x is positive")
+ } else {
+   print("x is negative")
+ }
[1] "x is negative"
```

A nested if…else statement has the following structure:

```
if (test_expression1) {
   statement1
} else if (test_expression2) {
   statement2
} else if (test_expression3) {
   statement3
} else
   statement4
```

For example, the following code will determine whether x is positive or negative:

```
> x <- 0
> if (x < 0) {
+   print("x is a negative number")
+ } else if (x > 0) {
+   print("x is a positive number")
+ } else
+   print("x is zero")
[1] "x is zero"
```

An R package can be installed using the following command:

```
> install.packages("package_name")
```

For example, to install a package called "dicrawler", the following command can be used:

```
> install.packages("dicecrawler")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/dicecrawler
_0.1.0.zip'
Content type 'application/zip' length 14526 bytes (14 KB)
downloaded 14 KB

package ‘dicecrawler’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
C:\Users\cob.user\AppData\Local\Temp\RtmpeIvpcM\downloaded_packages
```

An installed R package needs to be loaded to be used in an R script. Use the following command to load a previously installed package:

```
> require("dicecrawler")
Loading required package: dicecrawler
Loading required package: jsonlite
Loading required package: rvest
Loading required package: xml2
Loading required package: curl
```

Note that this command will also load all the packages that the loaded package is realying on (e.g. jsonlite, rvest, xml2, curl).

RECOMMENDED READINGS

Please refer to the books below if you would like to learn more about R and RStudio.

The following book provides a basic introduction to R and RStudio:

- Lander, J. P. (2014). R for Everyone: Advanced Analytics and Graphics. Boston, MA: Addison-Wesley.

This book by Hadeley Wickham, Chief Scientist at RStudio, provides good coverage of advanced topics in relation to R language:

- Wickham, H. (2014). Advanced R. Boca Raton, FL: CRC Press.

REFERENCES

Comprehensive R Archive Network (CRAN). 2016. Available at: https://cran.r-project.org/

Krotov, V. (2017). dicecrawler: downloads job descriptions from dice.com. R package version 0.1.0. Retrieved from https://cran.r-project.org/web/packages/dicecrawler/.

Lander, J. P. (2014). *R for Everyone: Advanced Analytics and Graphics*. Boston, MA: Addison-Wesley.

Wickham, H. (2014a). *Advanced R*. Boca Raton, FL: CRC Press.