Basic Course on R: Introduction and History

Karl Brand* and Elizabeth Ribble†

14-18 May 2018

Contents

*brandk@gmauil.com
†emcclel3@msudenver.edu

1 Course Overview

In the first part of the course we are introduced to the `R` software package, or *environment*, and learn about `R`, how to interact with it and it's basic programming elements. In the second part of the course we'll learn how to use `R` for it's intended function: doing statistics quickly and effectively.

We will cover essential topics including how to use `R` and why we use it; objects, classes and functions; and creating, importing, saving, manipulating, combining, sub-setting and plotting data.

By the end of the first couple of introductory days I expect that you can write your own code and have a reasonable understanding of what's going on when you cut and paste other people's code, which is typical of how people get started with `R` (or any other language for that matter).

Day 1

- Introduction, `R` history, getting familiar with `R` console, RStudio
- Objects, data structures, classes, using functions, arguments
- Containers, names, selection rules, accessing data.frame elements, lists

Day 2

- Entering, importing & manipulating data with `c(), cbind(), rbind(), dim()`; importing from a file; working directory
- Basic plotting of boxplots, bargraphs, scatterplots & line graphs
- Hypothesis testing & confidence intervals part 1: summary statistics, t-test, Mann-Whitney U test

Day 3

- Hypothesis testing & confidence intervals part 2: correlations, ANOVA, chi-squared test
- Distribution-free ANOVA
- Plotting with ggplot2

Day 4

- Basic linear regression, diagnostics and plotting
- Basics of logistic regression
- Programming structures 1: creating your own functions, if/else functions, loops

Day 5

- Programming structures 2: scope, recursion, replacement, search path
- Object-oriented programming and performance enhancement: generic functions and methods, memory optimization

2 Introduction

2.1 History

Before R there was the statistical analysis program, S, developed at Bell Laboratories in 1976 by John Chambers. S was later commercialised as S-PLUS. John Chambers is also a current member of the R core group responsible for developing R.

Ross Ihaka and Robert Gentleman of Dept of Statistics, University of Auckland, New Zealand begin coding (1991) the S clone, R, as an open source alternative for academic use. Together with the 'R core group' they released version 1 under the GNU Public License (GPL) v2 and v3 in 2000. The name, R, probably derives from the first letter of the creator's names. http://cran.r-project.org/doc/FAQ/R-FAQ.html#Why-is-R-named-R_003f

2.2 Strengths and weaknesses

I attribute much of R's success to its GPL v2 license: created by users for users to run well on Linux, Apple and Windows. These licenses give everyone the freedom to use, the freedom to distribute and the freedom to contribute their own intellectual input to the project. The result is that performance and fitness of purpose is and always will be the primary focus. I attribute R's limited general use to its steep learning curve for non-programmers. Base R is also yet to take advantage of the multi-threaded architecture of current computing technology.

2.3 Get it

2.3.1 Install it

To get R on your PC **first** you need to install the core R application. Of course you'll need to download the version for your operating system and architecture first.

For Linux: https://cran.r-project.org/bin/linux/

For Mac: https://cran.r-project.org/bin/macosx/

For Windows: https://cran.r-project.org/bin/windows/

Although R for Windows (and Mac?) ships with a basic text editor, and it's possible to execute code from the terminal under linux, a fully featured editor increases convenience and productivity. Grab an awesome text editor:

RStudio: https://www.rstudio.com/products/rstudio/

Emacs/ESS: http://ess.r-project.org/index.php?Section=download

Here's a full list available editors: https://en.wikipedia.org/wiki/R_(programming_language)#Interfaces

2.4 Not convinced?

r4stats.com has a nice evaluation of R's value 'in the field'. http://r4stats.com/articles/popularity/

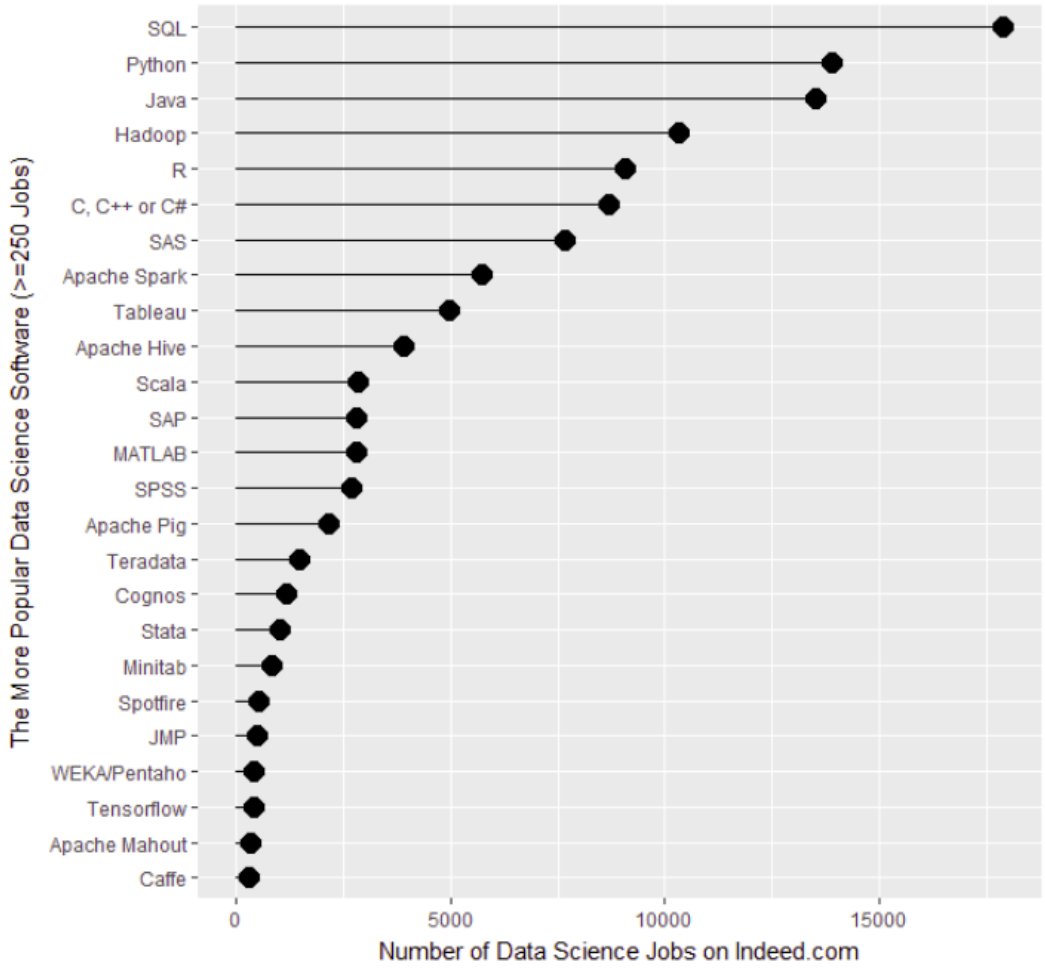
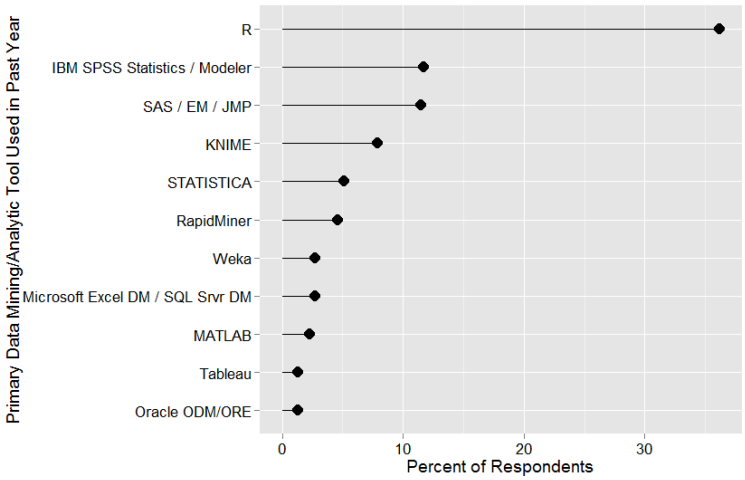
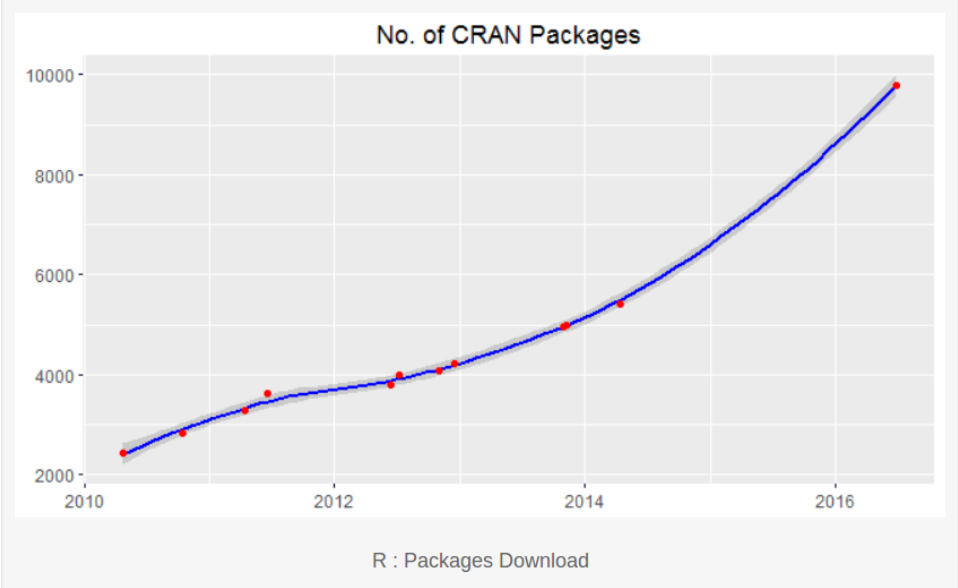


Figure 1a. The number of data science jobs for the more popular software (those with 250 jobs or more, 2/2017).

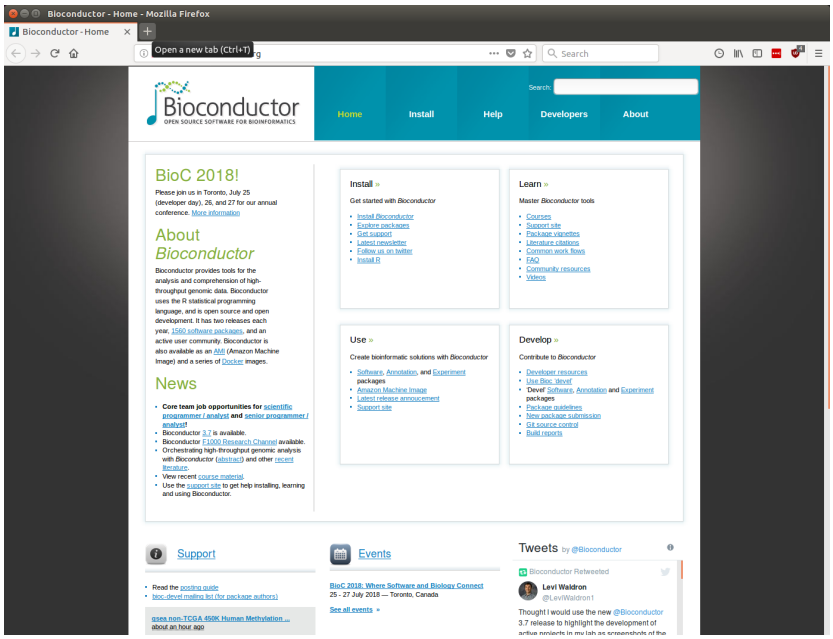
https://i0.wp.com/r4stats.com/wp-content/uploads/2017/02/Fig-1a-IndeedJobs-2017.png



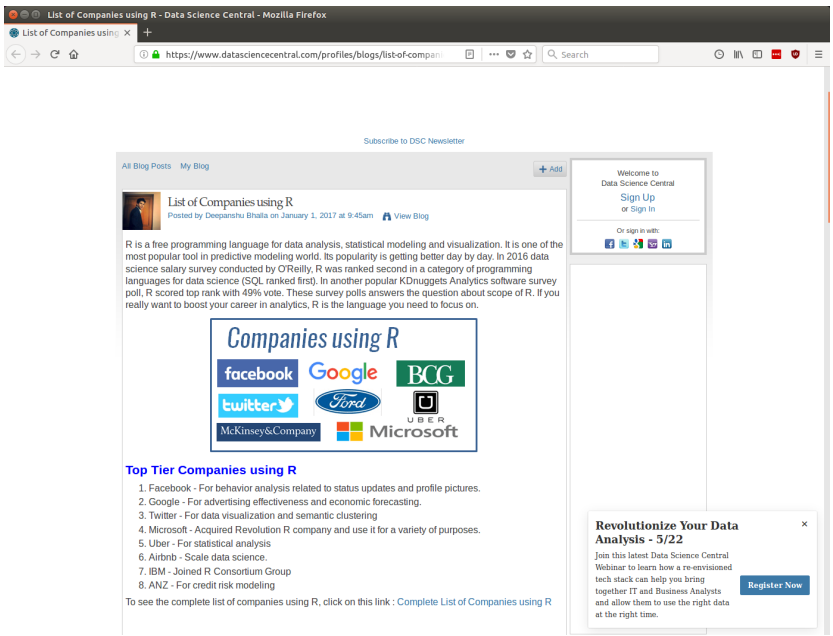
R is the most used software for getting data mining and big data "done". https://i1.wp.com/datasciencepopularity.com/wp-content/uploads/2015/10/fig_6a_rexersurveyprimarytc png



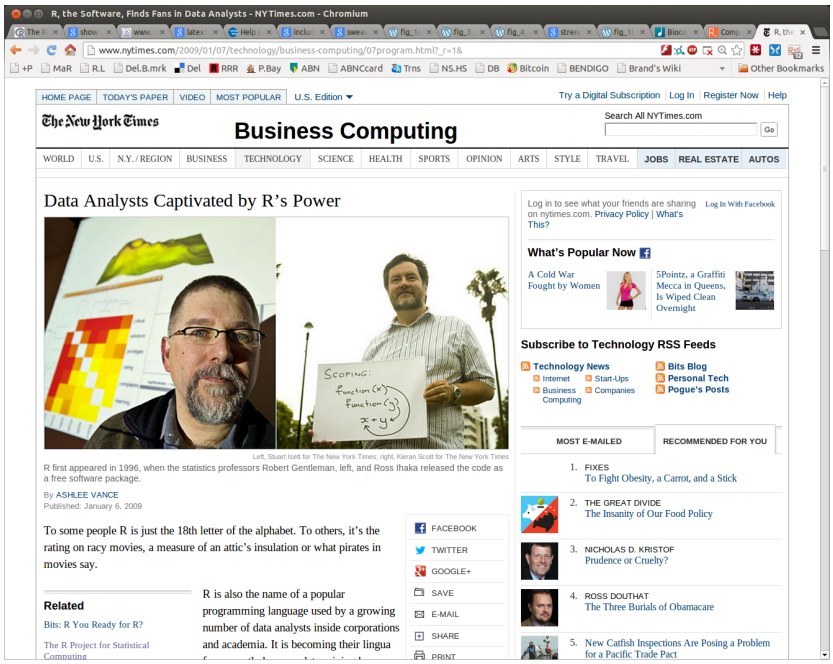
There's an enormous number of packages offering custom functionality. https://www.listendata.com/2016/12/companies-using-r.html



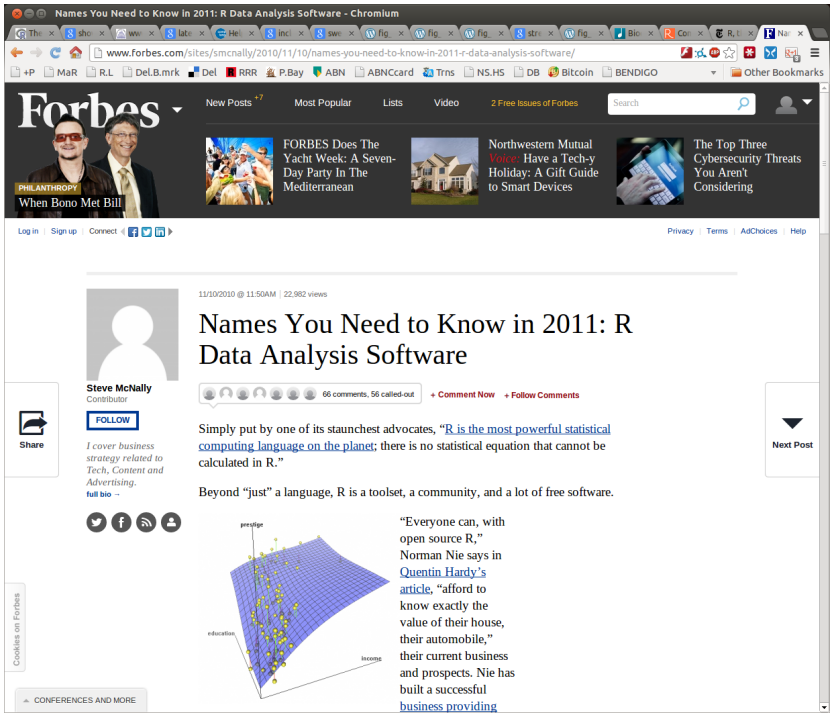
Bioconductor, initiated in 2001 for the analysis of genomics data, has grown from 15 to (currently) 1560 software packages for omics and life science analyses. `http://www.bioconductor.org/`



Used internally by lots of big, profitable companies. `https://www.datasciencecentral.com/profiles/blogs/list-of-companies-using-r`



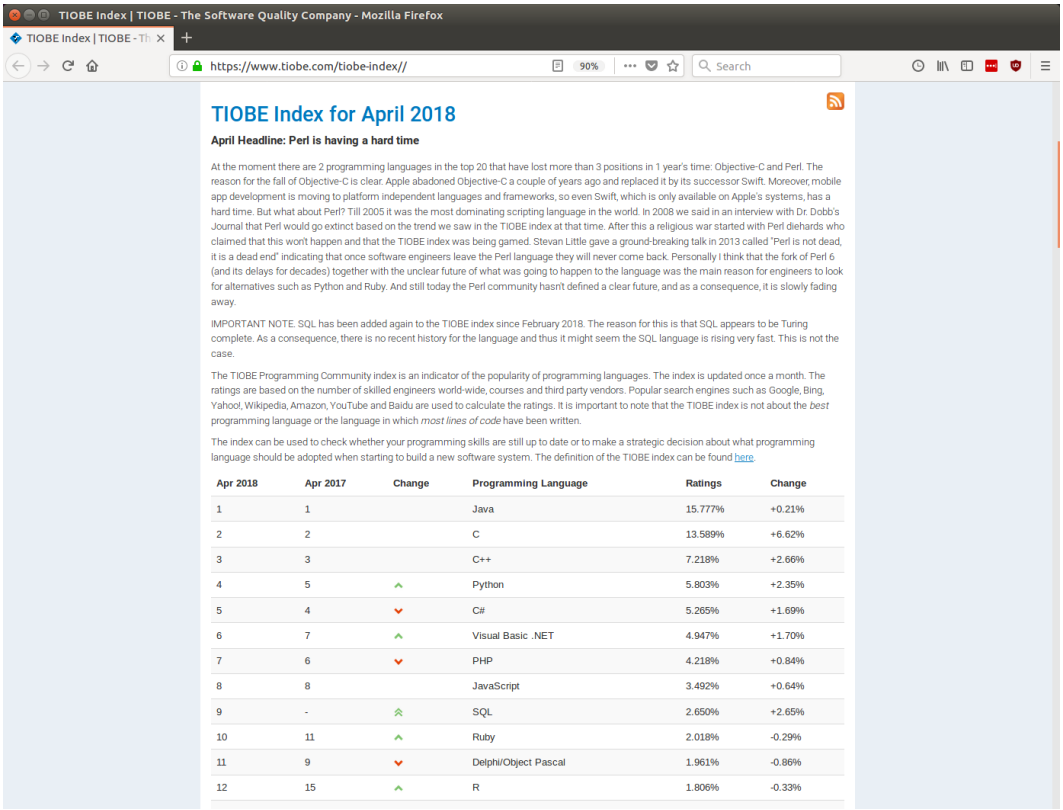
Has gained the popular media's attention. `http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=0`



Link to reference

Many other statistical and analytical programs have implemented connectors to enable running `R` code within them, including 'competitors' like SPSS, MatLab and SAS. De-

spite this `R`'s popularity continues to grow. In contrast to the 'workhorse' languages like `C` and Python which are used to build many programs, `R` is first and foremost a statistics focussed language. Despite this, it's close to being in the top 10 of the most popular computing languages.



TIOBE Index | TIOBE - The Software Quality Company - Mozilla Firefox

https://www.tiobe.com/tiobe-index//

TIOBE Index for April 2018

April Headline: Perl is having a hard time

At the moment there are 2 programming languages in the top 20 that have lost more than 3 positions in 1 year's time: Objective-C and Perl. The reason for the fall of Objective-C is clear. Apple abadoned Objective-C a couple of years ago and replaced it by its successor Swift. Moreover, mobile app development is moving to platform independent languages and frameworks, so even Swift, which is only available on Apple's systems, has a hard time. But what about Perl? Till 2005 it was the most dominating scripting language in the world. In 2008 we said in an interview with Dr. Dobb's Journal that Perl would go extinct based on the trend we saw in the TIOBE index at that time. After this a religious war started with Perl diehards who claimed that this won't happen and that the TIOBE index was being gamed. Stevan Little gave a ground-breaking talk in 2013 called "Perl is not dead, it is a dead end" indicating that once software engineers leave the Perl language they will never come back. Personally I think that the fork of Perl 6 (and its delays for decades) together with the unclear future of what was going to happen to the language was the main reason for engineers to look for alternatives such as Python and Ruby. And still today the Perl community hasn't defined a clear future, and as a consequence, it is slowly fading away.

IMPORTANT NOTE. SQL has been added again to the TIOBE index since February 2018. The reason for this is that SQL appears to be Turing complete. As a consequence, there is no recent history for the language and thus it might seem the SQL language is rising very fast. This is not the case.

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Apr 2018	Apr 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.777%	+0.21%
2	2		C	13.589%	+6.62%
3	3		C++	7.218%	+2.66%
4	5	↑	Python	5.803%	+2.35%
5	4	↓	C#	5.265%	+1.69%
6	7	↑	Visual Basic .NET	4.947%	+1.70%
7	6	↓	PHP	4.218%	+0.84%
8	8		JavaScript	3.492%	+0.64%
9	-	⇈	SQL	2.650%	+2.65%
10	11	↑	Ruby	2.018%	-0.29%
11	9	↓	Delphi/Object Pascal	1.961%	-0.86%
12	15	↑	R	1.806%	-0.33%

Link to current Tiobe index

3 What *is* R and how do we use it

3.1 Definition(s)

- **Wikipedia**: 'R is a programming language and free software environment for statistical computing and graphics that is supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.' `http://en.wikipedia.org/wiki/R_(programming_language)`
- **Comprehensive R Archive Network (CRAN):** 'R is "GNU S", a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.' `https://cloud.r-project.org/`.
- **My definition:** A command based, object oriented and functional language; in contrast to excel which is a cell centric, spreadsheet managing graphical user interface (GUI).

```
newObject <- function(oldObject)
```

object(s):	`oldObject`, `newObject`, and `function()`
assignment:	`<-`
expression:	`function(oldObject)`

3.2 How does R work?

You type commands, also known as "expressions," into a text editor or terminal and send them to R for evaluation. R evaluates the command, prints the command (by default), then the result of the evaluated command, if any.

In short you apply functions to objects, or "call" a function on an object, e.g.:

```
x <- c(1, 3, 5) #create an object
mean(x)         #calculate the arithmetic mean of the values in x
```

```
## [1] 3
```

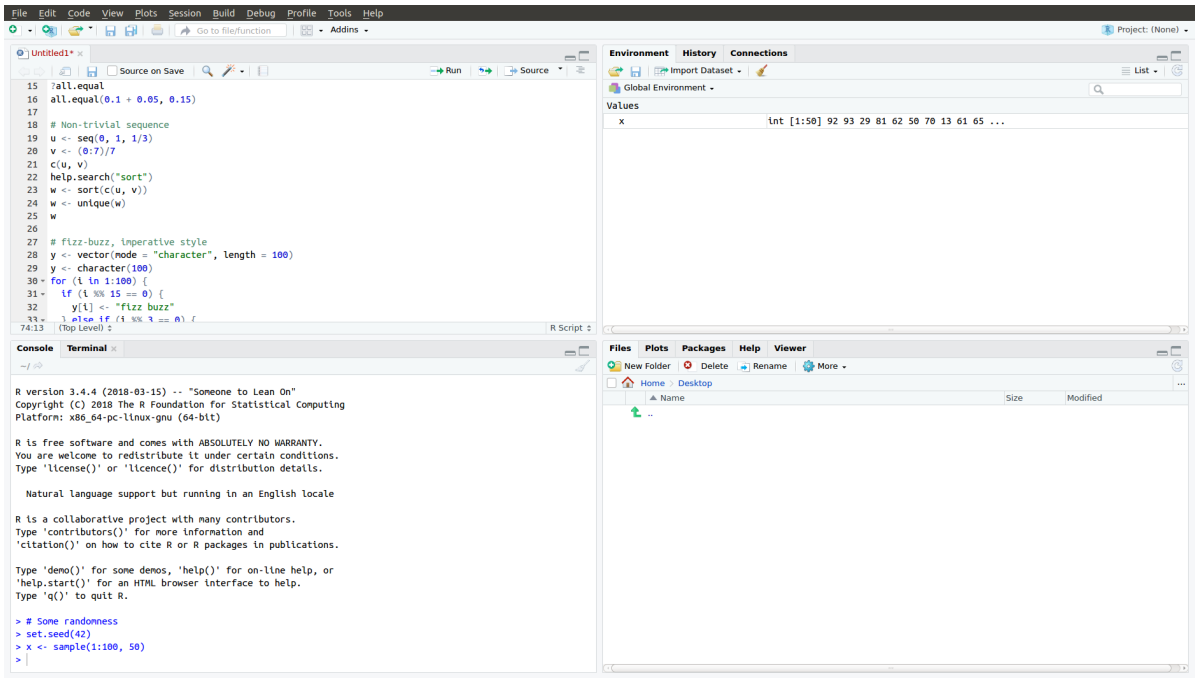
3.3 How we interact with R

Not surprisingly everything about R is built on the principle of KISS, i.e., keep it simple stupid.

- We get things done in R by typing commands and asking R to execute them with the `enter` key. Assuming you know the commands, it couldn't be any simpler.
- The "front end" of R, the part that receives your commands, is the *terminal*. You can type your commands directly into the terminal and hit `enter` to do your work. But if you need thousands of commands for a specific task, or want to share your method, this approach gets tiring fast.
- Since the commands are just text, it's easy to write a series of commands using any program built for working with text, i.e., text editors and word processors, to create *code* or a *script*, to get something done.
- The easiest, most efficient way to write R code is using a text editor or *integrated development environment*, like RStudio or Emacs/ESS as mentioned earlier.

3.3.1 RStudio: the easiest way to work with R

For most users, especially those new to R, RStudio is probably the best IDE to use. Like everything about R, you're free to choose whatever editor you like. For Windows platforms, you could just use the basic editor that's bundled with the downloadable installer. Let's have a look at an RStudio "session".



There are four panes.

- The top left is the source pane. **Do your typing in this pane**.
- The bottom left is your console. This is a window looking into what `R` is actually doing. **Even though it's fine to type your commands in this pane at the `R` '`>`' prompt, avoid this**: your commands are less conveniently stored and rerun than your source code in the editor pane.
- Top right: environment pane
- Bottom left: files pane

Type your commands in the source pane and send them to `R` *using the keyboard shortcut* `Ctrl + Enter`. This will send the line of code your cursor is on to `R` if no code is selected; or if there is code selected, then only the selected code is sent. Open RStudio on your PC and try it now.

A few other things about RStudio:

- move the focus between panes using the key board shortcuts `Ctrl + 1` and `Ctrl + 2` for the source and console panes respectively
- the little asterisk next your source code file name indicates unsaved code
- **save your source code!**

3.4 Backslashes

One more thing I want to mention is how `R` interprets backslashes and forwardslashes, i.e. `\` and `/` respectively, as relates to defining the locations of files on your PC. James McDonald does a better job of clarifying this than I otherwise would `http://cran.r-project.org/doc/contrib/Lemon-kickstart/kr_scrpt.html`:

"Let's pause to note a little historical bifurcation that has caused almost as much strife as the question of exactly what the `<Delete>` key should do. The convention that the slash (/) character is used as a separator in the notation for a filesystem path is not universal. PC-DOS, for example, used the backslash (\). Like Fords vs Chevrolets, French vs English, and whether you crack the egg at the big or small end, it doesn't really matter a rat's behind which one you use as long as it works. To make it work in R, however, where the *NIX convention of using the backslash as an escape character is respected, you will have to double backslashes in paths for them to be read properly. Thus if the filename above was:

`C:\JIM\PSYCH\ADOLDRUG\PARTYUSE1.R`

I would have to refer to it as:

`C:\\JIM\\PSYCH\\ADOLDRUG\\PARTYUSE1.R`

in an R script."

4 Getting Help

1. Online help. The first place to look. Also works when you're actually *offline*:

   ```
   ?help
   help(help)
   ```

   Broader search:

   ```
   ??help
   apropos("help")              # quotes are needed
   ```

   It's fun, try them out!

2. Google. And when you're actually *online*, often the *first* place and *last* place you'll go when you have no idea is Google.

3. Package Vignette's. Where the online help is devoid of the context often needed to grasp the full potential and logical use of functions, vignette's is the best

place to find such information, with extended examples, and often more. For example, see the limma `http://www.bioconductor.org/packages/release/bioc/vignettes/limma/inst/doc/usersguide.pdf`

4. Mailing Lists. R-help `http://www.r-project.org/mail.html`. Do check first using Google or the R-help searchable archives `http://tolstoy.newcastle.edu.au/~rking/R/` first if your question has already been asked and has the answer to solve your issue.
5. Other lists: `https://stat.ethz.ch/mailman/listinfo`
6. Stack Overflow `https://stackoverflow.com`. Probably more questions and answers posted here than on the mailing list.

4.1 Trouble shooting

- Clean your environment and start again to be sure your messy environment isn't the cause of your issue. In RStudio, click the broom in the Environment tab; in any R session you can also run:

```
# not run:
rm(list=ls())
ls()
```

- Start `R` without any customisations, i.e., omit loading the `.Rprofile` and `.Renviron` customisation files. From the command line:

  ```
  R --vanilla
  ```

- `trace.back()` function: very helpful pinpointing the source of an error, and thus its cause.

4.2 Further resources

- `R` focused search engine: `http://www.Rseek.org`
- Aggregator of all R blogs: `http://www.r-bloggers.com`
- Quick-R: `http://www.statmethods.net/`
- `R` on Youtube: `http://www.youtube.com/watch?v=mL27TAJGlWc`
- `R` in two minutes: `http://www.twotorials.com/`
- *The* list of reference texts: `http://www.r-project.org/doc/bib/R-publications.html`

- And more..
  `http://www.ats.ucla.edu/stat/r/`
  `http://zoonek2.free.fr/UNIX/48_R/02.html`

- *A First Course in Statistical Programming with R.* W. John Braun and Duncan J. Murdoch. Cambridge University Press, 2007
  *I love this book, beautifully explores the programming aspects of `R`.

5 Document License