R Basics

In this first lesson we are going to go over the very basics of R, cover some basic terminology, talk a little about syntax and point you to resources for getting help.

Lesson Outline:

- Working with R and RStudio
- Functions and basic syntax
- Packages
- Operators and objects
- Getting help

Lesson Exercises:

- Exercise 1.1
- Exercise 1.2
- Exercise 1.3
- Exercise 1.4

Code

Before we get started, the code for all three lessons has been consolidated into a single R script. If you'd like, download `r7_intro_code.R` and follow along during the live coding.

Working with R and RStudio

Over the last several years, RStudio has become a very popular IDE (integrated development environment) for R. In addition to interacting with the R Console, RStudio has many extras built in including version control integration, package building, reproducible research, de-bugging, and built-in text editor with smart highlighting and code completion. This is the environment we will be using for the workshop and should set you up for continued learning with R.

Before we get to the first exercise, let's spend a bit of time working with RStudio. Just watch as I demo this. The first exercise will have you doing these steps.

Fire up R and RStudio

Find the RStudio shortcut or menu (OS specific of course) and fire it up. Once done, it should look something like:

Let's take some time to look around. I'll show each of the different sections, or "panes" as they are known.

Projects

Projects are a way to organize your work in RStudio. Essentially they are folders, but with a few added files so that you can manage some options on a per project basis. To create a new project use File:New Project, or use the drop-down on the top right of the RStudio window. It will look like this after you select "New Project..."
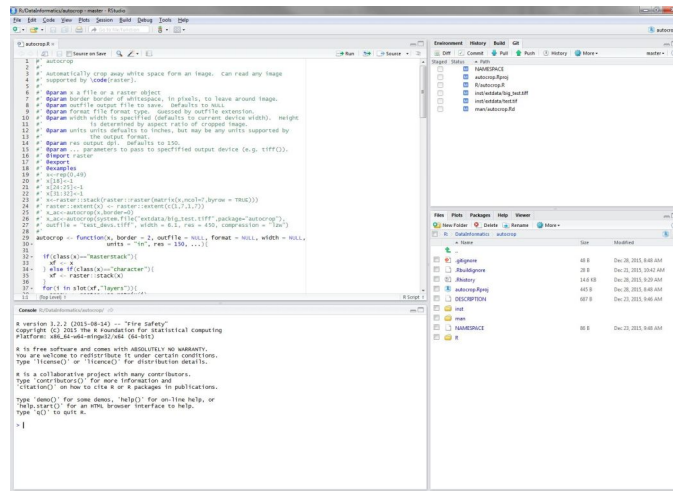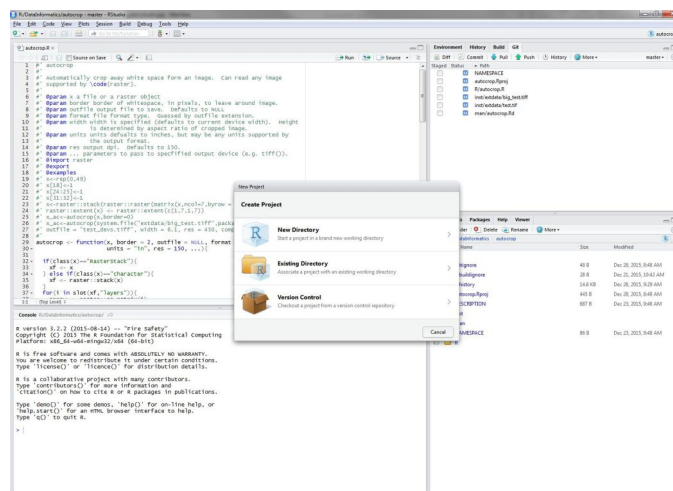
Figure 1: rstudio

Figure 2: rstudio proj

Scripts

Scripts are text files that hold the code you write. We will work both with scripts and the console during this workshop. To create a new script you use "File:New File:R Script".
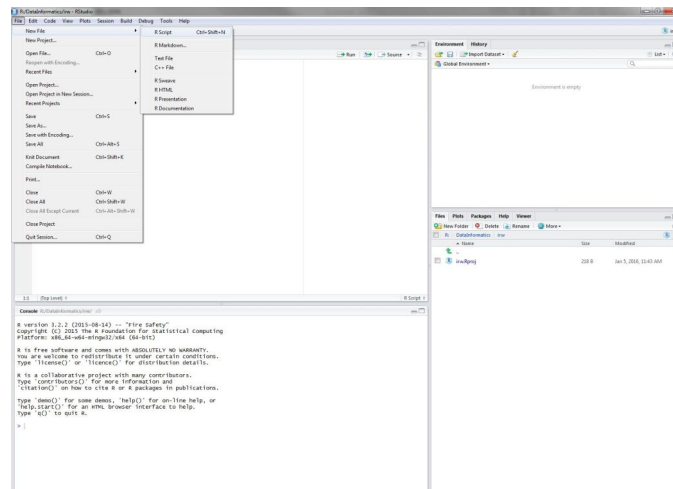

Figure 3: rstudio script

Interacting with R inside of RStudio

Once you have functions in your script, they still need to be sent to the R console. There are several ways to do this. There is the old stand-by of copying and pasting, but this is a bit cumbersome. Instead you can use the `Run` button in the upper right corner of the source pane, or even better (I think so, anyway) you can use `ctrl-enter`. Both the `Run` buttons and `ctrl-enter` will send the line that your cursor is on and move to the next line or it will send just selected text.

Exercise 1.1

This exercise will make sure R and RStudio are working and that you can get around the basics in RStudio. Use the green stickies when you have completed, and red stickies if you are running into problems.

1. Start RStudio: To start both R and RStudio requires only firing up RStudio. RStudio should be available from All Programs at the Start Menu. Fire up RStudio.

2. Take a few minutes to look around RStudio. Find the Console Pane. Find Global and Project Options (hint: look in Tools). Look at the Environment, History Pane. Look at the Files, Plots, Packages, etc. pane.

3. Create a new project. Name it "intro_r_workshop". We will use this for the rest of the workshop.

4. Create a new "R Script" in the Source Pane, save that file into your newly created project and name it "r1_introR.R". It'll just be a blank text file at this point.

5. Add in a comment line to separate this section. It should look something like: `# Exercise 1.1: Just Getting used to RStudio and Scripts`.

6. Add a single line to this script with the following text: `ls()`. This is an R function that lists objects in your current environment. Use the various ways we showed to send this command to the R Console. Also, try typing this directly into the R Console and hit `Enter` to run it.
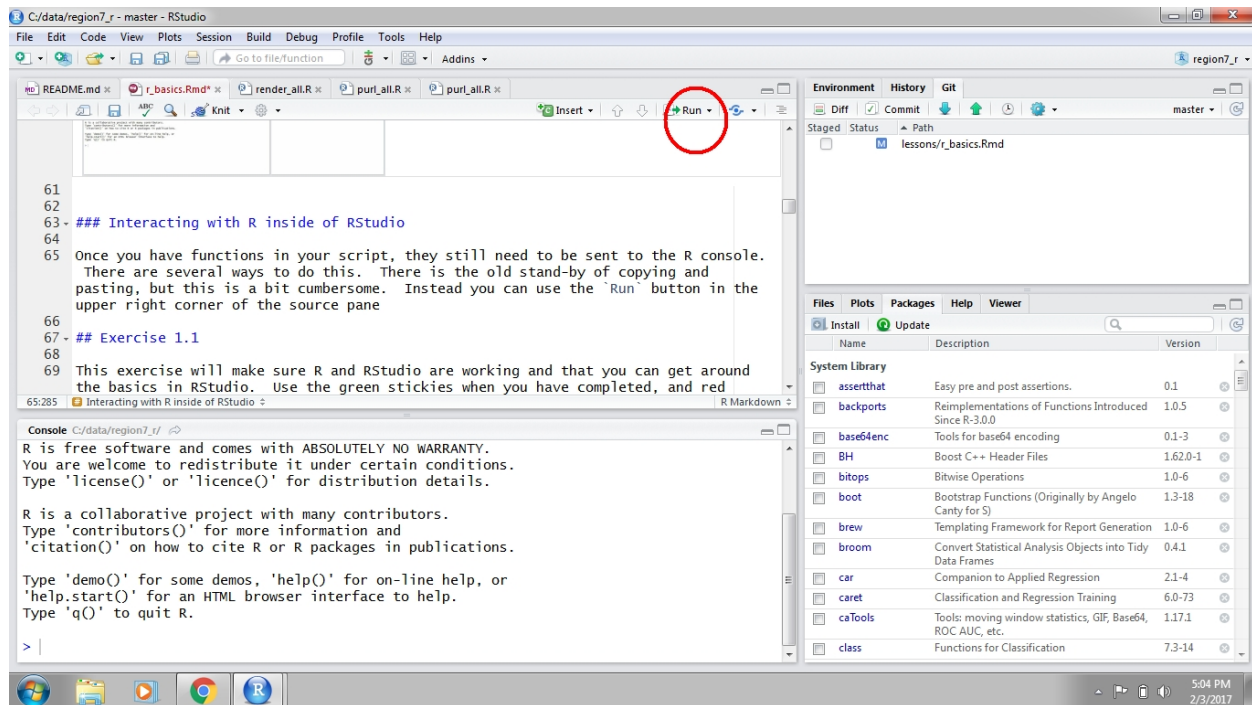
Figure 4: rstudio-script

Functions and basic syntax

R is a functional programming language and as such, most everything you do uses a function.

The basic syntax of function follows the form: `function_name(arg1, arg2, ...)`. With the base install, you will gain access to many (2922 functions, to be exact). Some examples:

```r
#Print
print("hello world!")
```

```
## [1] "hello world!"
```

```r
#A sequence
seq(1,10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
#Random normal numbers
rnorm(100,mean=10,sd=2)
```

```
##   [1] 11.577396 10.834899 11.584148 12.722458 11.006765  9.696114  9.095874
##   [8] 10.879927  9.738208 10.220417  7.533173  8.808861  8.174826 12.499426
##  [15]  4.584983  9.583348  5.109972  6.868895  8.290827 15.123485 11.463143
##  [22]  9.168282 10.964457 10.403424  7.361352  7.238142 11.213432  9.271446
##  [29]  8.381155  8.264622  7.004233 10.668940 16.123963  7.911612 10.998632
##  [36] 13.215045 10.073366  6.280461 11.891838  7.396315 11.648739  9.795505
##  [43]  9.884113  9.253469 10.855037 10.227422  7.665298  7.821154  9.852340
##  [50] 11.277289  9.796722  8.227478 12.632325  7.504299 11.549919  8.095226
##  [57]  8.552409  8.334361  9.159477  6.528743  9.026390  9.308937 10.932928
##  [64] 10.909878 10.429064 11.115569  8.430110  7.127629 12.360166  7.734242
##  [71]  9.552490  8.693034 12.501501  8.082048  8.033423 11.374870 10.236369
```

```
## [78]  9.944902  9.896334  7.796932 11.003418 12.454189 14.262383  7.602814
## [85]  5.487293  9.894172 10.712634  9.958196 12.038335 10.037957  8.091014
## [92] 13.952111 12.999035 12.912654  9.876017 10.201739  4.020547 11.800015
## [99] 11.882173  9.266895
```

```
#Mean
mean(rnorm(100))
```

```
## [1] 0.05839271
```

```
#Sum
sum(rnorm(100))
```

```
## [1] -7.845657
```

A few side notes

There are several other characters that commonly show up in R code. These are:

```
# NOTES:
#Comments
#()
#[]
#{}
```

The `#` indicates a comment. You can put whatever else you'd like after this, but on the same line as the `#`. R will not evaluate it. Multiple `#####`, are still just seen as a comment. When commenting your code, err on the side of too much! Also, you will see (), [], and {} used in R code. The () indicates a function (almost always), the [] indicates indexing (grabbing values by the location), and the {} groups code that is meant to be run together and is usually used when programming functions in R.

Exercise 1.2

For this second exercise we are going to get used to using some basic functions. Use the green stickies when you have completed, and red stickies if you are running into problems.

1. Save your work in a script, make sure "r1_introR.R" file you created in Exercise 1 is open. Enter your commands into this script.

2. Add in a comment line to separate this section. It should look something like: `# Exercise 1.2: Getting used to functions and basic syntax`.
3. Use the `print` function. Remember, use the "run" button or `ctrl-enter` in RStudio to have R interpret it and print something to the screen.
4. Combine `mean` and `rnorm` to return the mean value of a set of random numbers.

Packages

The base install of R is quite powerful, but you will soon have a need or desire to go beyond this. Packages provide this ability. They are a standardized method for extending R with new methods, techniques, and programming functionality. There is a lot to say about packages regarding finding them, using them, etc., but for now let's focus just on the basics.

CRAN

One of the reasons for R's popularity is CRAN, The Comprehensive R Archive Network. This is where you download R and also where most will gain access to packages (there are other places, but that is for later). Not much else to say about this now other than to be aware of it. As of 2017-02-06, there are 10009 on CRAN!

Installing packages

When a package gets installed, that means the source is downloaded and put into your library. A default library location is set for you so no need to worry about that. In fact on Windows most of this is pretty automatic. Let's give it a shot.

```
#Installing Packages from CRAN
#Install dplyr and ggplot2
install.packages("ggplot2")
install.packages("dplyr")

#You can also put more than one in like
install.packages(c("quickmapr","formatR"))
```

Using packages

One source of confusion that many have is when they cannot access a package that they just installed. This is because getting to this point requires an extra step, loading (or attaching) the package.

```
#Loading packages into your library
#Add libraries to your R Session
library("ggplot2")
library("dplyr")

#You can also access functions without loading by using package::function
dplyr::mutate
```

```
## function (.data, ...)
## {
##     mutate_(.data, .dots = lazyeval::lazy_dots(...))
## }
## <environment: namespace:dplyr>
```

You will often see people use `require()` to load a package. It is better form to not do this. For a more detailed explanation of why `library()` and not `require()` see Yihui Xie's post on the subject

And now for a little pedantry. You will often hear people use the terms "library" and "package" interchangeably. This is not correct. A package is what is submitted to CRAN, it is what contains a group of functions that address a common problem, and it is what has allowed R to expand. A library is, more or less, where you packages are stored. You have a path to that library and this is where R puts new packages that you install (e.g. via `install.packages()`). These two terms are related, but most certainly different. Apologies up front if I slip and use one when I actually mean the other...

Exercise 1.3

For this third exercise we are going to get used to using some basic functions, working with scripts and not just the console, and look through some task views and get used to basic navigation around packages. Use

the green stickies when you have completed, and red stickies if you are running into problems.

1. Install the `dplyr` and `ggplot2` packages. Do this directly in the console.
2. Why might we not want to add `install.packages` into the script?
3. Add in a comment line to separate this section. It should look something like: `# Exercise 1.3: Adding packages`.
4. Load `ggplot` and `dplyr` into your library. Do this in your script. Make sure that it gets sent to the console.
5. Why might we want to add the `library()` calls in our script?

Operators and objects

As I mentioned above, the console and using R interactively is very powerful. We will do this quite a bit. Let's spend a little time playing around in the console and learn a few new functions.

R can be used as a calculator and a way to compare values. Some examples of the basic operators:

```r
#A really powerful calculator!
1+1 #Add
```

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

```r
10-4 #Subtract
```

```
## [1] 6
```

```r
3*2 #Multiply
```

```
## [1] 6
```

```r
3^3 #Exponents
```

```
## [1] 27
```

```r
100/10 #Divide
```

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

```r
5%%2 #Modulus
```

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

```r
5>2 #Greater than
```

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

```r
4<5 #Less than
```

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

```r
5<=5 #Less than or equal
```

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

```r
8>=2 #Greater than or equal
```

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

```r
2==2 #Equality: notice that it is TWO equal signs!
```

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

```r
5!=7 #Not Equals
```

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

That's neat, but so what...

Well, it could be interesting to do something with those values and save them for re-use. We can do that with objects (everything in R is an object) and use the assignment operator, `<-`. Know that object names cannot start with a number, contain spaces, or (most) special characters. Underscores and periods are allowed.

NOTE: If you have experience with other, object-oriented languages, then just be aware that R objects, at least the general use of the term, are different.

```r
#Numeric assignment
x<-5
x
```

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

```r
y<-x+1
y
```

```
## [1] 6
```

```r
z<-x+y
z
```

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

```r
#Character
a<-"Bob"
a
```

```
## [1] "Bob"
```

```r
b<-"Sue"
b
```

```
## [1] "Sue"
```

```r
a2<-"Larry"
a2
```

```
## [1] "Larry"
```

Now that we have a little experience working in the console and creating objects with `<-`, we might want to be able to do some additional things to navigate around, look at these objects etc. You can do a lot of this directly in RStudio in the Environment, History pane which is likely in the upper right corner of the window. Alternatively, you can explore your current environment via the console. Some functions that you might find useful for working with your R workspace:

```r
#List all objects in current workspace
ls()
ls(pattern="a")

#Remove an object
rm(x)

#Save your workspace
#Saves the whole thing to a file called lesson2.RData
save.image("lesson2.RData")
#Saves just the a and y objects to a file called lesson2_ay.RData
save(a,y,file="lesson2_ay.RData")
```

This is probably a good spot to bring up quotes vs no quotes around arguments in a function. This is a very common stumbling block. The general rule is that no quotes are used only when referring to an object that

currently exists. Quotes are used in all other cases. For instance in `save(a,y,file="lesson2_ay.RData")` the objects `a` and `y` are not quoted because they are objects in the workspace. `file` is an argument of save and argument names are never quoted. We quote the name of the file "lesson2_ay.RData" because it is not an R object but the name of a file to be created. You will likely still have some issues with this. My recommendation is to think about if it is an object in your R workspace or not. If so, no quotes! This isn't foolproof, but works well most of the time.

Next thing you might want to do is navigate around your files and directories. While you can do this directly from the console, it is going to be better practice to mostly use RStudio projects to manage your folders, working directory etc. You can also navigate using the Files, etc. pane.

Exercise 1.4

1. Add in a comment line to separate this section. It should look something like: `# Exercise 1.4: Objects and Operators`.
2. Create two objects named `number1` and `number2` and give them the values of 25 and 4, respectively
3. Create two more objects named `string1` and `string2`, give them any character string that you would like.
4. Now using `number1,number2`, and the power of math create an object called `number3` that equals 100.
5. Create two more objects whose value is of your choosing
6. List the objects in your workspace
7. Remove `string2`
8. Try to add `string1` and `number1`. What happens?

Getting help

Being able to find help and interpret that help is probably one of the most important skills for learning a new language. R is no different. Help on functions and packages can be accessed directly from R, can be found on CRAN and other official R resources, searched on Google, found on StackOverflow, or from any number of fantastic online resources. I will cover a few of these here.

Help from the console

Getting help from the console is straightforward and can be done numerous ways.

```r
#Using the help command/shortcut
#When you know the name of a function
help("print") #Help on the print command
?print #Help on the print command using the `?` shortcut

#When you know the name of the package
help(package="dplyr") #Help on the package `dplyr`

#Don't know the exact name or just part of it
apropos("print") #Returns all available functions with "print" in the name
??print #Shortcut, but also searches demos and vignettes in a formatted page
```

Official R Resources

In addition to help from within R itself, CRAN and the R-Project have many resources available for support. Two of the most notable are the mailing lists and the task views.

- R Help Mailing List: The main mailing list for R help. Can be a bit daunting and some (although not most) senior folks can be, um, curmudgeonly...
- R-sig-ecology: A special interest group for use of R in ecology. Less daunting the the main help with participation from some big names in ecological modelling and statistics (e.g., Ben Bolker, Gavin Simpson, and Phil Dixon). One of the moderators is great, the other is a bit of a jerk (it's me).
- Environmetrics Task View: Task views are great in that they provide an annotated list of packages relevant to a particular field. This one is maintained by Gavin Simpson and has great info on packages relevant to much of the work at EPA.
- Spatial Analysis Task View: One I use a lot that lists all the relevant packages for spatial analysis, GIS, and Remote Sensing in R.

Google and StackOverflow

While the resources already mentioned are useful, often the quickest way is to just turn to Google. However, a search for "R" is a bit challenging. A few ways around this. Google works great if you search for a given package or function name. You can search for mailing lists directly (i.e. "R-sig-geo"). An R specific search tool, RSeek.org, has been created to facilitate this.

One specific resource that I use quite a bit is StackOverflow with the 'r' tag. StackOverflow is a discussion forum for all things related to programming. You can then use this tag and the search functions in StackOverflow and find answers to almost anything you can think of.

Other Resources

As I mention earlier, there are TOO many resources to mention and everyone has their favorites. Below are just a few that I like.

- R For Cats: Basic introduction site, meant to be a gentle and light-hearted introduction
- Advanced R: Web home of Hadley Wickham's new book. Gets into more advanced topics, but also covers the basics in a great way.
- Other Resources: A list I helped compile for a URI Class.
- CRAN Cheatsheets: A good cheat sheet from the official source
- RStudio Cheatsheets: Additional cheat sheets from RStudio. I am especially fond of the data wrangling one.