

Introduction to R

Marcus W. Beck¹ Todd D. O'Brien²

¹ORISE, USEPA NHEERL Gulf Ecology Division

Email: beck.marcus@epa.gov

²NOAA/NMFS Copepod Project

Email: todd.obrien@noaa.gov

What you'll learn in this intro

- Course expectations and pre-workshop materials
- What is R?
- What's possible with R?
- R basics
 - ▶ Installation
 - ▶ Command-line interface
 - ▶ Coding basics
 - ▶ Functions and objects
 - ▶ Data import and manipulation
- Help!

Course expectations

The November 17th workshop will provide you with a set of tools for evaluating time series data from SWMP

Preparation for the workshop:

- Review pre-workshop toolkit materials
- Bring a computer with R version 3.0.0 or later
- Optionally, install RStudio in addition to R
- Have a basic proficiency using R, what this means:
 - ▶ You do not have to be an expert!
 - ▶ Understand the basics of a command-line interface
 - ▶ Know how to open R, load a script, execute functions
 - ▶ Know how to save your work
- Install the 'SWMPpr' package after installing R

Course expectations

This is not an R training workshop...

...but we will be using R exclusively to handle SWMP data

You will receive an overview of the theory behind time series analysis

We will use an R package developed to work with SWMP data

A package is a collection of functions written by others that can be installed within R

This package can automatically handle common problems working with SWMP data and time series, it is designed to make your life easier!

Course expectations

The pre-workshop toolkit includes:

- R installation instructions - *R_install_guide.pdf*
- Intro to R (current document) - *intro_to_R.pdf*
- Basics of data analysis with R - *r_for_data_analysis.pdf*
- Installing and working with the SWMP_r package - *intro_to_swmp_r.pdf*

The final pdf is optional as we will cover the content in the workshop, **but you should have the SWMP_r package installed prior to the 17th**

Please note that webinar attendees will have limited interaction with the instructors, although we will have moderators handling questions.

Course expectations

Copies of all instructional materials will be made available on the course website: copepod.org/nerrs-swmp-workshop/

Physical attendees will also receive a flashdrive with the course materials

The instructors can also be contacted with questions prior to the workshop:

- For R questions including toolkit and installation - Marcus Beck, beck.marcus@epa.gov
- For time series analysis questions - Todd O'Brien, todd.obrien@noaa.gov

Course expectations

Finally, the presentation materials combine content with R code and R output

R code that can be executed will look like this:

```
# here's some R code  
rnorm(10)
```

The output will look similar to this in R (without '##'):

```
## [1] -0.35  0.73 -0.87 -2.01 -0.30  0.06 -1.98  0.29  0.07  0.04
```

When applicable, R scripts are provided that contain only the executable code within each training module. You can use these in R as you read each presentation.

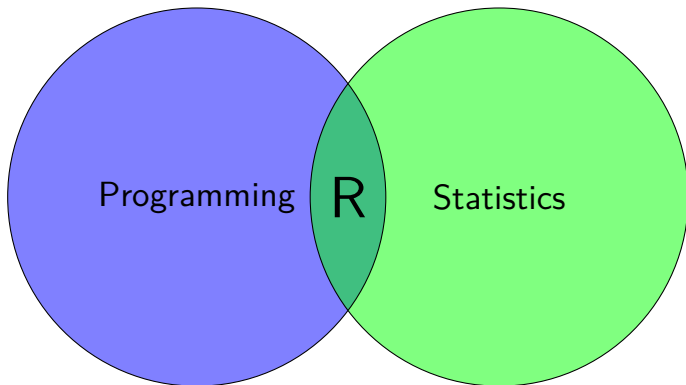
What is ?

R is a computer language that allows the user to program algorithms and use tools that have been programmed by others

Different from other statistics software because it provides both tools for analysis and it is also a programming language...

You do not have to be a programmer to use R!

What is ?

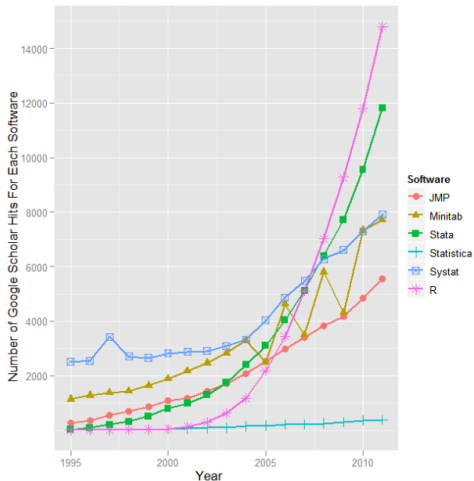


R is both... this creates a steep learning curve.

What is R?

R is becoming the statistical software of choice

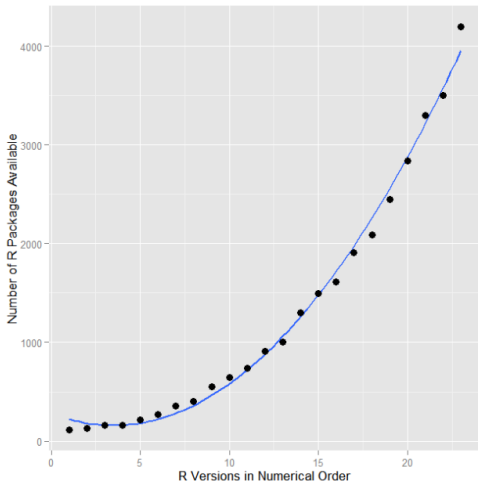
Plot of Google scholar hits over time for different software packages
[r4stats.com]



What is R?

R is becoming the statistical software of choice

Exponential growth in number of contributed packages [r4stats.com]



What's possible with ?

R is incredibly flexible, if you want something done, someone else has written the code...

R is open-source software, which mean it's free and is supported by a large network of contributors - the Comprehensive R Network [[CRAN](#)]

CRAN is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries [[R FAQ](#)]

Basically a repository of R utilities that others have written as well as the source files for R - the [CRAN task views](#) contain descriptions of contributed packages by category

What's possible with ?

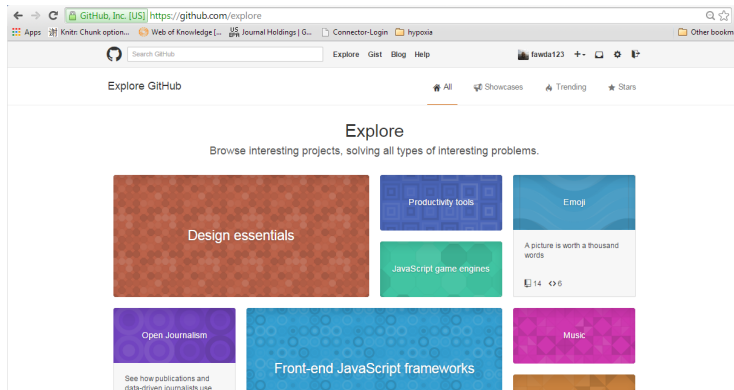
CRAN task views

CRAN Task Views

| | |
|--|---|
| Bayesian | Bayesian Inference |
| ChemPhys | Chemometrics and Computational Physics |
| ClinicalTrials | Clinical Trial Design, Monitoring, and Analysis |
| Cluster | Cluster Analysis & Finite Mixture Models |
| DifferentialEquations | Differential Equations |
| Distributions | Probability Distributions |
| Econometrics | Computational Econometrics |
| Environmetrics | Analysis of Ecological and Environmental Data |
| ExperimentalDesign | Design of Experiments (DoE) & Analysis of Experimental Data |
| Finance | Empirical Finance |
| Genetics | Statistical Genetics |
| Graphics | Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization |
| HighPerformanceComputing | High-Performance and Parallel Computing with R |
| MachineLearning | Machine Learning & Statistical Learning |
| MedicalImaging | Medical Image Analysis |
| MetaAnalysis | Meta-Analysis |
| Multivariate | Multivariate Statistics |

What's possible with ?

GitHub has also been increasingly used to store packages online - like an informal CRAN:



What's possible with ?

R comes with a base package that is included in installation, others are downloaded as needed, e.g.,

```
# download from CRAN  
install.packages('ggplot2')
```

The base package will be sufficient for most of your needs - includes arithmetic, input/output, basic programming support, graphics, etc.

Once a package is installed, it must be loaded to use its functions:

```
# load ggplot2  
library(ggplot2)
```

What's possible with ?

Or you can download packages from GitHub, which requires using the devtools package:

```
# download devtools from CRAN  
install.package('devtools')  
  
# load devtools  
library(devtools)  
  
# install SWMPPr from GitHub, load  
install_github('fawda123/SWMPPr')  
library(SWMPPr)
```


What's possible with ?

Each package may also come with a demonstration

This provides a neat way to see what an R package has to offer

To see a list of packages with demonstrations, run this code:

```
# view packages with demos  
demo(package = .packages(all.available = TRUE))
```

To view a demonstration of basic graphic capabilities in R:

```
# view a demo for the graphics package  
demo(graphics)
```

What's possible with ?

Each package comes with extensive help documentation

Help files for a package:

```
# view help files for a package  
help(package = 'ggplot2')
```

Or for an individual function in a package:

```
# get help file  
help(mean, package = 'base')  
  
# or do this  
?mean
```

Following the instructions in our installation guide for step-by-step directions

Or visit r-project.org and follow directions

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

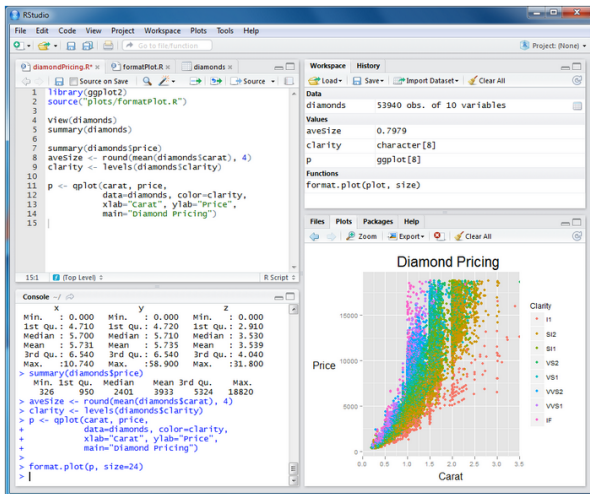
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2013-04-03, Masked Marvel): [R-3.0.0.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.

RStudio is also recommended but not required



How is R different from Excel? R is a command-line interface

```
R version 2.15.2 (2012-10-26) -- "Trick or Treat"  
Copyright (C) 2012 The R Foundation for Statistical Computing  
ISBN 3-900051-07-0  
Platform: x86_64-w64-mingw32/x64 (64-bit)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.
```

```
Natural language support but running in an English locale
```

```
R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R or R packages in publications.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for an HTML browser interface to help.  
Type 'q()' to quit R.
```

```
> |
```

What next??

Lines of code are executed by R at the prompt ($>$)

Enter the code and press enter, the output is returned

```
print('hello world!')
```

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

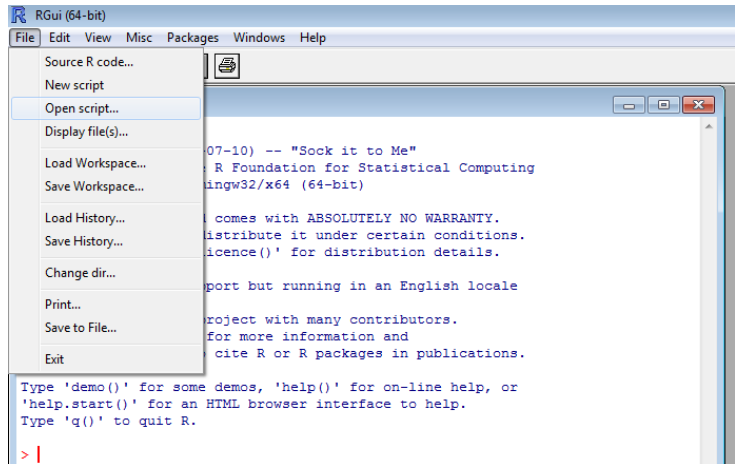
```
2 + 2
```

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

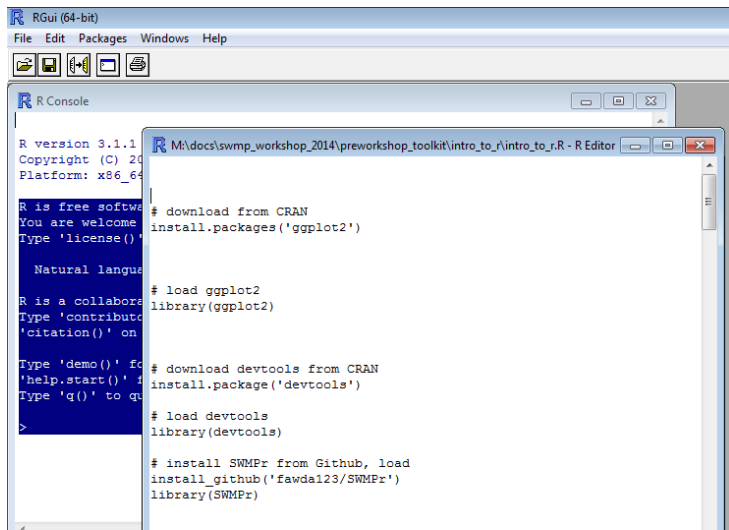
```
(2 + 2) / 4
```

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

The best way to run code on the command prompt is to have a script file that contains all your code – this can be saved and opened later



The best way to run code on the command prompt is to have a script file that contains all your code



The screenshot shows the R GUI (64-bit) with a menu bar (File, Edit, Packages, Windows, Help) and a toolbar. The R Console window displays the R version 3.1.1 welcome message. The R Editor window, titled 'R Editor', shows a script file with the following code:

```
# download from CRAN
install.packages('ggplot2')

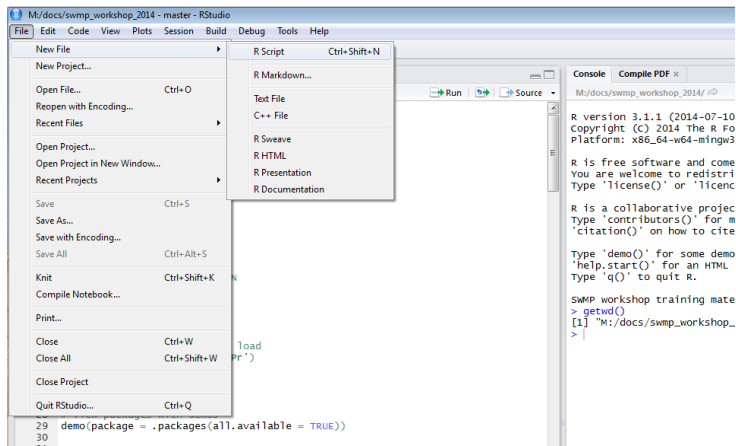
# load ggplot2
library(ggplot2)

# download devtools from CRAN
install.packages('devtools')

# load devtools
library(devtools)

# install SWMPPr from Github, load
install_github('fawda123/SWMPPr')
library(SWMPPr)
```


Opening a new script in RStudio is similar



The script is a text file with a `.r` extension

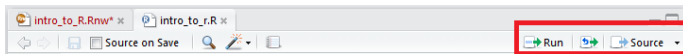
Lines that are preceded by `'#'` are comments that are ignored when executed in the console

You can copy and paste lines from the script direct to the console - but this is inefficient

An easier way is to run code line-by-line using `'Ctrl + r'` - this runs a line and advances the cursor

Or you can run the whole script - `'Ctrl + a'` followed by `'Ctrl + r'` - don't do this unless you know what the script is going to do.

RStudio has buttons on the top for sending code to the console, but it's easier using keyboard shortcuts (`'Ctrl + r'`)



The R console is ready to accept a command when you see this: >

If you see a plus sign when you are running code, the console is expecting more input before the command is executed – either your code is incorrect or you have not run the entire command

After a command is executed, output will be returned on the console

It is common to see bracketed numbers preceding returned values, for example:

```
seq(1, 30) # create a sequence of numbers from 1 to 30
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
## [26] 26 27 28 29 30
```

These numbers specify the index number of the value returned at the start of each new row

All code that is entered must be 100 % correct

```
# getting an error  
2 + a  
  
## Error: object 'a' not found
```

In this example, we get an error because there is no object named 'a'

Note the distinction here:

```
# note the difference  
2 + 'a'  
  
## Error: non-numeric argument to binary operator
```

We cannot add a numeric and character value

Assigning data to R objects is essential for analysis

Assignment is possible using `<-` or `=`

This saves data in your 'workspace' that you can call later

```
a <- 1
2 + a

## [1] 3

a = 1
2 + a

## [1] 3
```

More complex assignments are possible

```
a <- c(1, 2, 3, 4)
```

```
a
```

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

```
a <- seq(1, 4)
```

```
a
```

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

```
a <- c('a', 'b', 'c')
```

```
a
```

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

Anatomy of a function - functions perform tasks for you, much like in Excel

function(arguments)

```
c(1, 2) # concatenate function to combine value
```

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

```
mean(c(1, 2)) # mean function
```

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

```
seq(1, 4) # create a sequence of values
```

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

Understanding classes of R **objects** is necessary for analysis

An object is a variable of interest that will always have a class

Most common are 'numeric' or 'character' classes

```
class(1)

## [1] "numeric"

class('1')

## [1] "character"
```

'Factors' are also common, define categorical variables

Different types of methods are assigned to different class types

The class also imposes limits on how it interacts with other class

For example, we cannot add add two objects with different classes:

```
# this does not work  
'1' + 1  
  
## Error: non-numeric argument to binary operator
```

There is no 'addition' method for character objects...

Objects (and their classes) are stored in the computer's memory in different ways - aka the workspace for your R session

The most common structures are 'vectors' and 'data.frames'

Vectors are a collection of objects of the same class, whereas a data frame is analogous to a table with rows and columns (e.g., collection of vectors)

```
a <- c(1,2) # numeric vector
```

```
a
```

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

```
b <- c('a', 'b') # character
```

```
b
```

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

```
# data frame with both
```

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

```
c
```

```
##   a b
```

```
## 1 1 a
```

```
## 2 2 b
```

How are data imported into R?

R needs to know where the data are located on your computer:

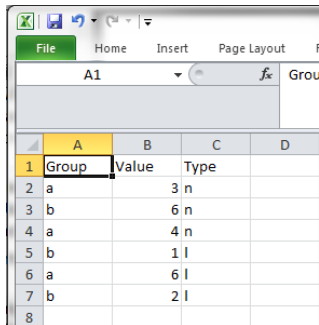
```
# the location where you want to work  
setwd('C:/my_directory')
```

This establishes a 'working directory' for data import/export

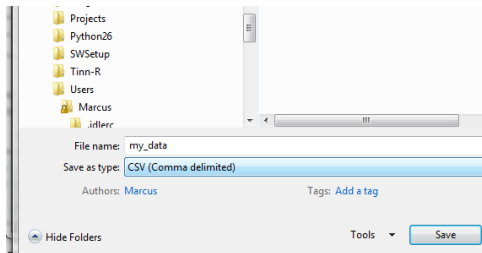
R can import almost any type of data but 'spreadsheet' or text-based files are most common

How are data imported into R?

The easiest approach is to save an Excel file as .csv or .txt file



| | A | B | C | D |
|---|-------|-------|------|---|
| 1 | Group | Value | Type | |
| 2 | a | 3 | n | |
| 3 | b | 6 | n | |
| 4 | a | 4 | n | |
| 5 | b | 1 | l | |
| 6 | a | 6 | l | |
| 7 | b | 2 | l | |
| 8 | | | | |



How are data imported into R?

Use the `read.table` or `read.csv` functions to import the data, must be in your working directory

```
# use read.csv if .csv  
dat <- read.csv('my_data.csv', header = T)  
  
# use read.table if .txt  
dat <- read.table('my_data.txt', sep = ',', header = T)
```

Imported data can be viewed several ways, view the whole object or parts

Rows or columns can be obtained by indexing with brackets separated by a comma: `data[row, column]`

```
dat
```

```
##      Group Value Type
## 1      a      3     n
## 2      b      6     n
## 3      a      4     n
## 4      b      1     l
## 5      a      6     l
## 6      b      2     l
```

```
dat[1, ] # row 1
```

```
##      Group Value Type
## 1      a      3     n
```

```
dat[, 2] # column 2
```

```
## [1] 3 6 4 1 6 2
```

```
dat[4, 1] # row 4, column 1
```

```
## [1] b
## Levels: a b
```

Imported data can be viewed several ways, view the whole object or parts

Access using column names or the attach function

```
dat$Value  
  
## [1] 3 6 4 1 6 2  
  
dat[, 'Value']  
  
## [1] 3 6 4 1 6 2
```

```
attach(dat)  
Value  
  
## [1] 3 6 4 1 6 2
```

Imported data can be viewed several ways, view the whole object or parts

The View function can also be used

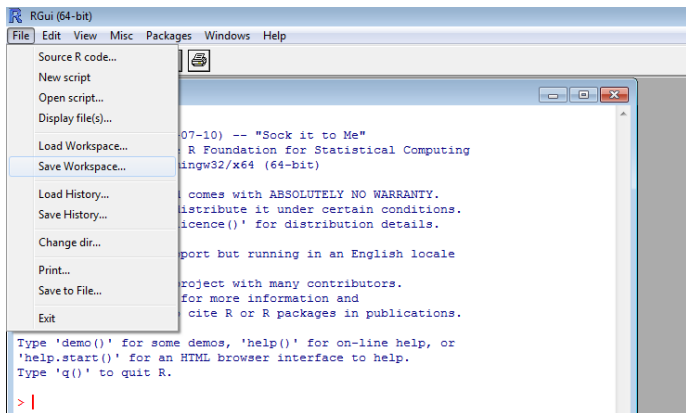
```
# use View to see the data  
View(dat)
```

This opens a separate spreadsheet-style window

It will be limited to 1000 rows for large datasets

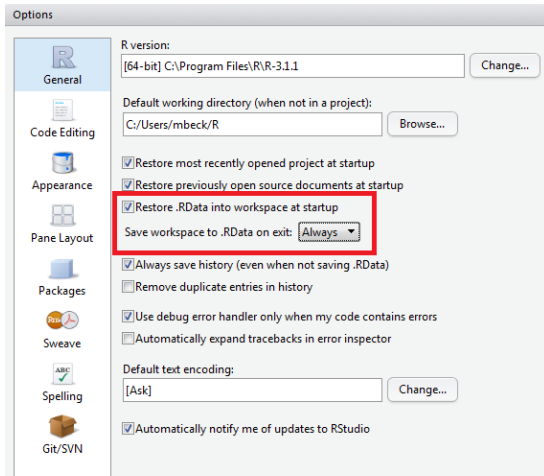
How can we save our work?

The easiest way is to save your workspace - i.e., the objects in your current R session



Saving your work in RStudio is easier - this is done automatically on close

Opening RStudio will automatically restore your previous workspace



Saving your workspace creates a '.RData' file of all the objects in your current working directory - to see the objects in your current workspace, use `ls()`

```
# view the objects in your workspace  
ls()
```

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

You can reload your workspace by double-clicking the .RData file or opening it from within R (File/Load Workspace)

You can also save individual objects as .RData files

```
# save only the dat object  
save(dat, file = 'my_data.RData')
```

```
# load .RData file of dat  
load(file = 'my_data.RData')
```

The .RData file is a binary format that can only be read by R

You can save your data in a more general format

```
# save as a csv file  
write.csv(dat, 'my_data.csv')  
  
# the defaults for write.csv can be changed  
write.csv(dat, 'my_data.csv', quote = F, row.names = F)  
  
# save as a txt file, comma separated  
write.table(dat, 'my_data.txt', sep = ',', quote = F,  
            row.names = F)
```

Once saved, the file can be imported as before (e.g., read.csv, read.table)

Finally, you'll want to save your R script (.r) file so you can use it next time

The R script is critical because it provides a working document of your code, whereas your workspace contains the data (in R format)

It's never a good idea to run commands using only the command prompt/console - write the code in a script first, then send it to the console when you're ready (ctrl + r)

Saving a script is simple - i.e., File, save

It's a good idea to keep your raw data, .RData workspace file, and scripts in the same directory - this will be the working directory you set earlier

Where to go for help?

- A user-friendly [intro to R](#)
- Several good introductory texts are available - Zuur et al. 2009. A Beginner's Guide to R. Springer.
- [R cheatsheet](#)
- Google is your friend
- Help files for each function using '?function' - may or may not be helpful
- An [intro to R](#) - very detailed
- Ask us!