Instructions for Creating Your Own R Package*

In Song Kim† Phil Martin‡ Nina McMurry§ Andy Halterman¶

March 18, 2018

1 Introduction

The following is a step-by-step guide to creating your own R package. Even beyond this course, you may find this useful for storing functions you create for your own research or for editing existing R packages to suit your needs.

This guide contains three different sets of instructions. If you use RStudio, you can follow the "Basic Instructions" in Section 2 which involve using RStudio's interface. If you do not use RStudio or you do use RStudio but want a little bit more of control, follow the instructions in Section 3. Section 4 illustrates how to create a R package with functions written in C++ via Rcpp helper functions.

NOTE: Write all of your functions first (in R or RStudio) and make sure they work properly before you start compiling your package. You may also want to try compiling with a very simple function first (e.g. `myfun <- function(x){x + 7}`).

2 Basic Instructions (for RStudio Users Only)

All of the following should be done in RStudio, unless otherwise noted. Even if you build your package in RStudio using the "Basic Instructions," we strongly recommend that you carefully review the "Advanced Instructions" as well. RStudio has built-in tools that will do many of these steps for you, but knowing how to do them manually will make it easier for you to build and distribute your own packages in the future and/or adapt existing packages.

1. Start by opening a new `.R` file. Make sure your default directory is clear by typing `rm(list = ls())`. Check to see that it is empty using `ls()` (you should see `character(0)`).

2. Write the code for your functions in this `.R` file. You can create one file with all of your functions or create separate files for each function. Save these files somewhere where you can easily find them.

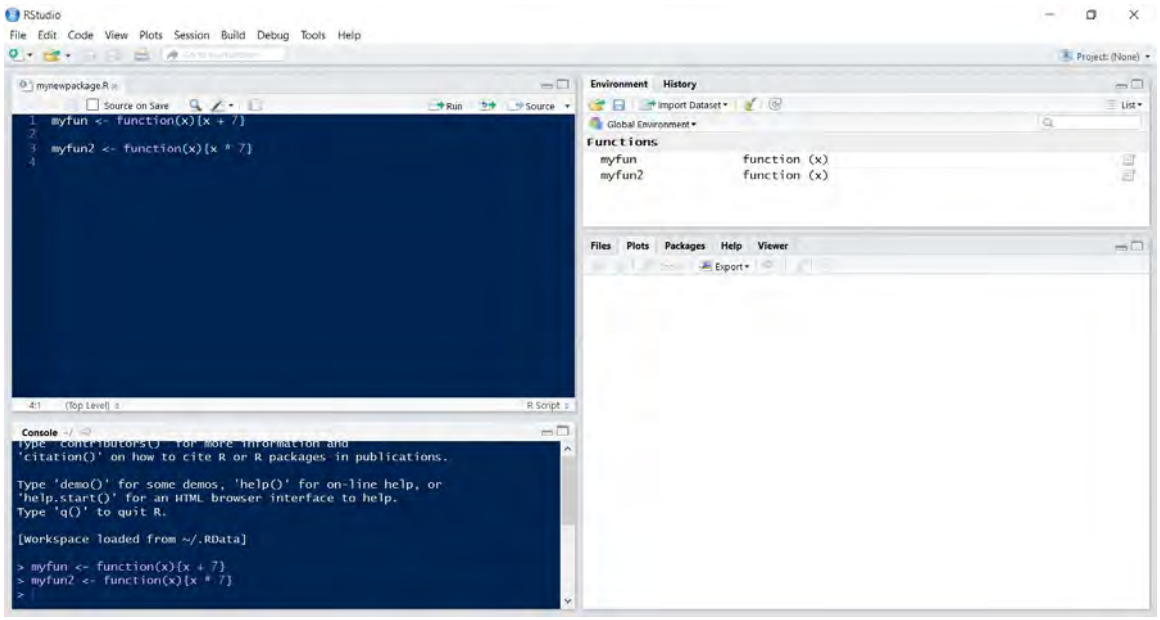
*We thank Min Hee Seo for her contribution.

†Assistant Professor, Department of Political Science, Massachusetts Institute of Technology, Cambridge, MA, 02139. Email: insong@mit.EDU, URL: http://web.mit.edu/insong/www/

‡Ph.D. student, Department of Political Science, Massachusetts Institute of Technology

§Ph.D. student, Department of Political Science, Massachusetts Institute of Technology

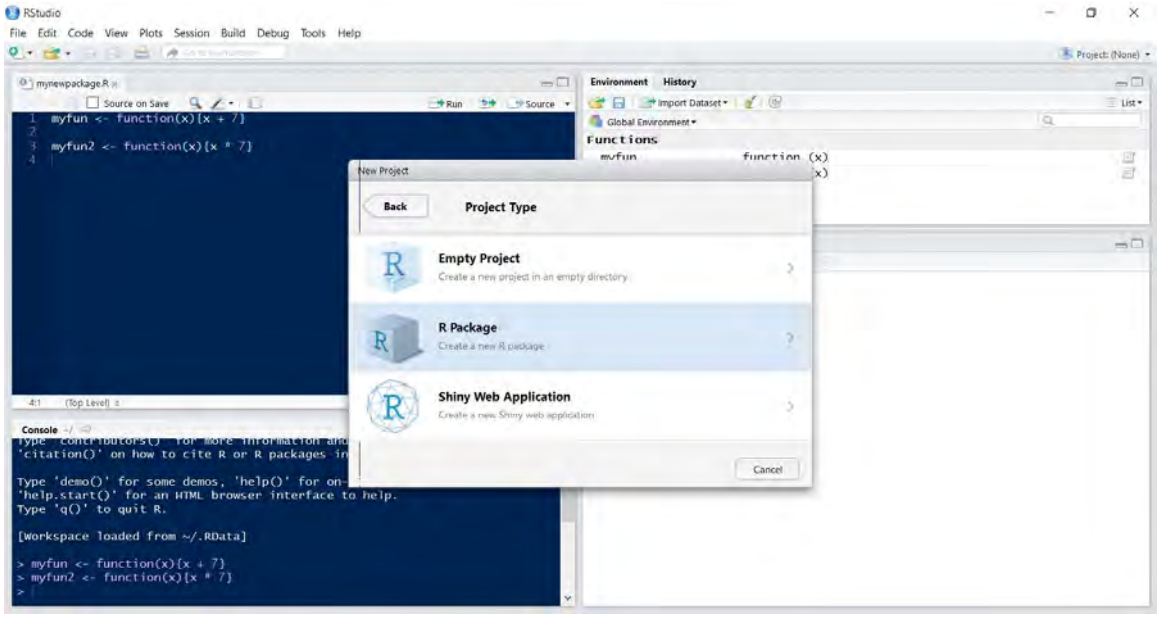
¶Ph.D. student, Department of Political Science, Massachusetts Institute of Technology



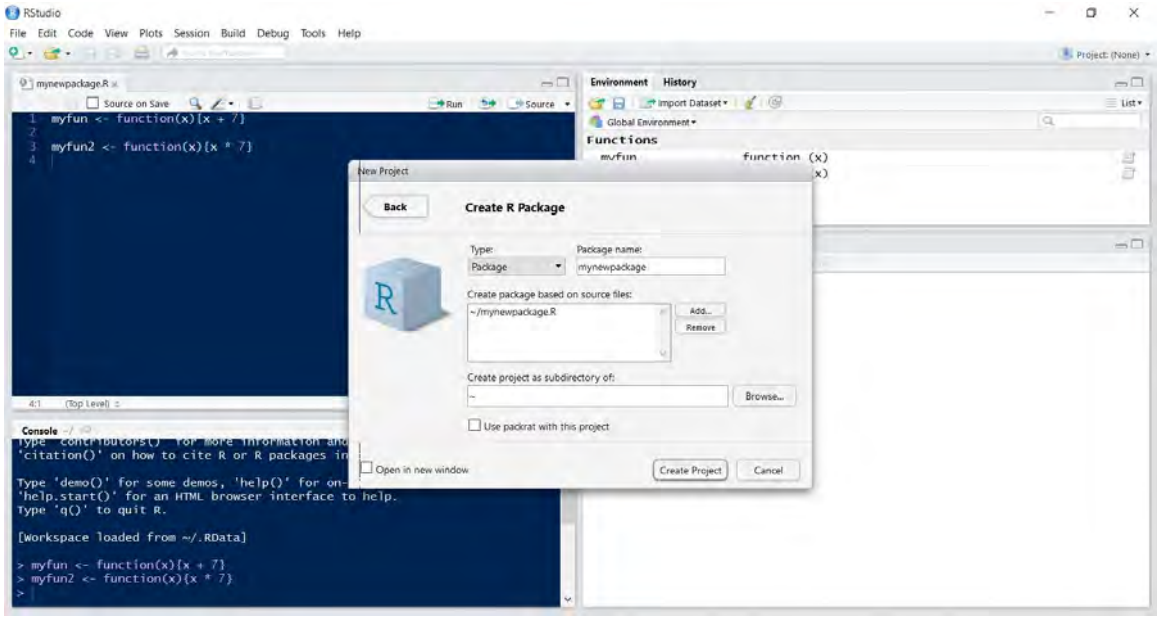
3. Install the 'devtools' package (`install.packages('devtools')`).

   Note. If you have a trouble installing `devtools`, 1) check your R version, 2) check whether `Rtools` is installed, 3) check your path, and 4) re-start your computer or R. If it still doesn't work, try `devtools::build_github_devtools()` instead.

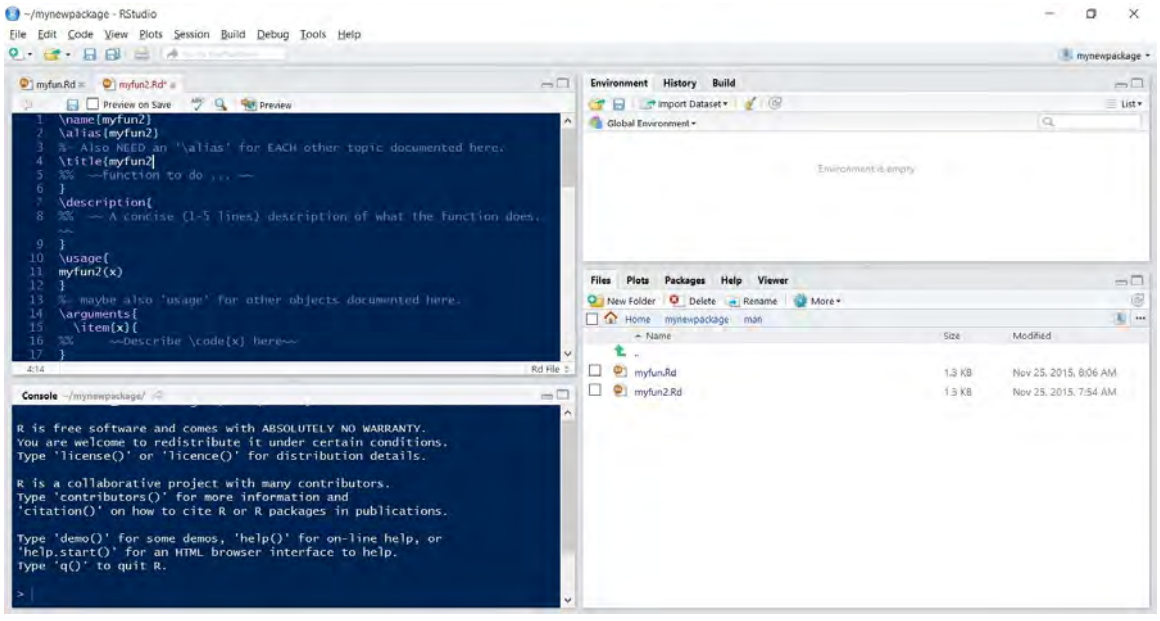
4. Open a new project in RStudio. Go to the 'File' menu and click on 'New Project.' Then select 'New Directory,' and 'R Package' to create a new R package.



5. Type the name of your package, then upload the `.R` file you created in step 1 under 'Create package based on source files'. Click 'Create project.'

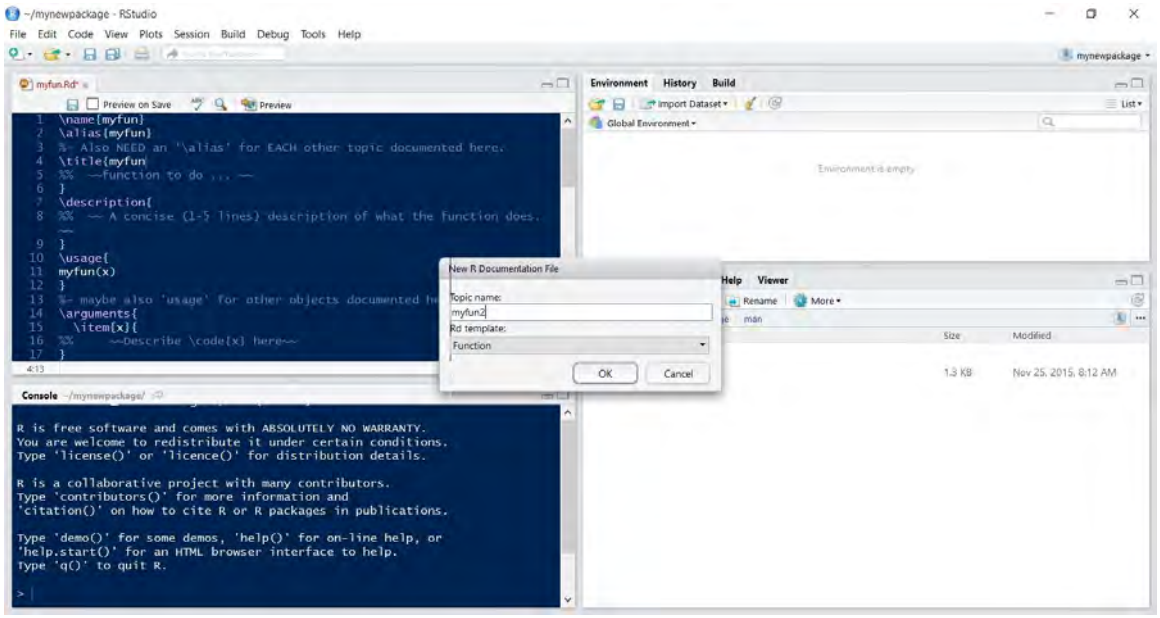


6. On the lower right hand side of your screen, you should see a file directory. The 'R' folder contains the code for your functions. The 'man' folder will contain the help files for each function in your package. Depending on your version of RStudio, the help files may have been generated automatically as `.Rd` or "R documentation" files when you created your package. If the 'man' folder already contains `.Rd` files, open each file, add a title under the 'title' heading, and save (if not, see step 7). You can go back and edit the content later, but you will need to add a title to each `.Rd` file in order to compile your package. Alternatively, you might find `Roxygen2` package useful for automating the process fo generating `.Rd` (and even NAMESPACE) files.

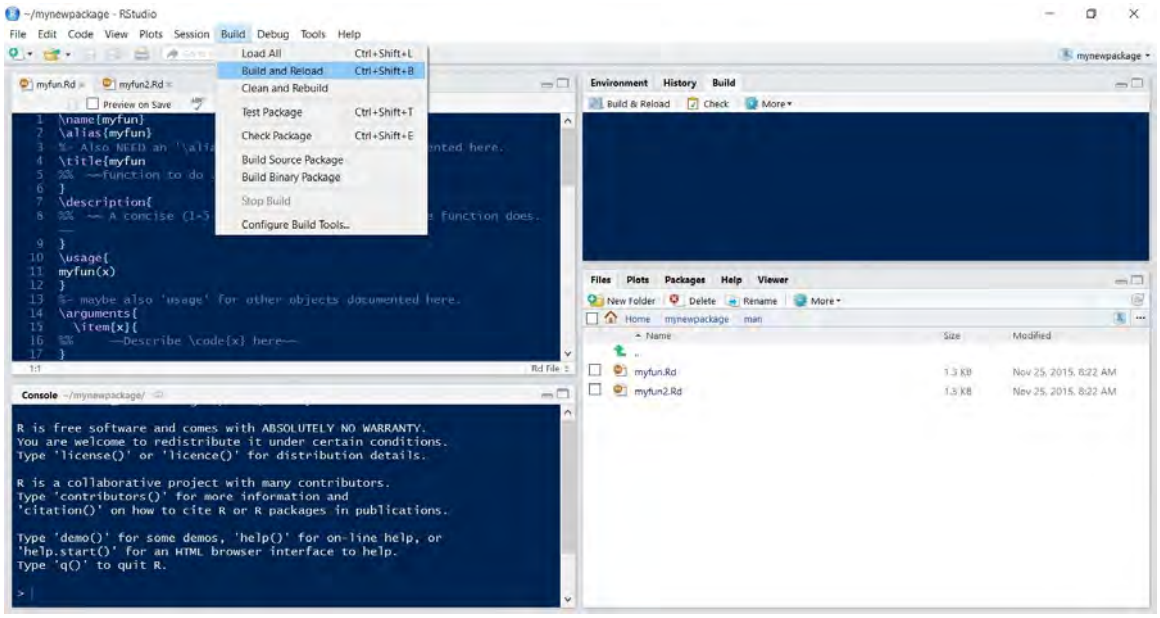


7. If your 'man' file is empty, you will have to manually create a `.Rd` file for each function. To do this, go to File > New File > R Documentation, enter the title of the function and select 'Function' under the 'Rd template' menu. Edit your new file to include something in the 'title' field (again, you may make other edits now or go back and make edits later, but your package will not compile if the 'title' field is empty). Save each `.Rd` file in the 'man' folder. *NOTE: You will need to complete this step if you add more functions to your package at a*

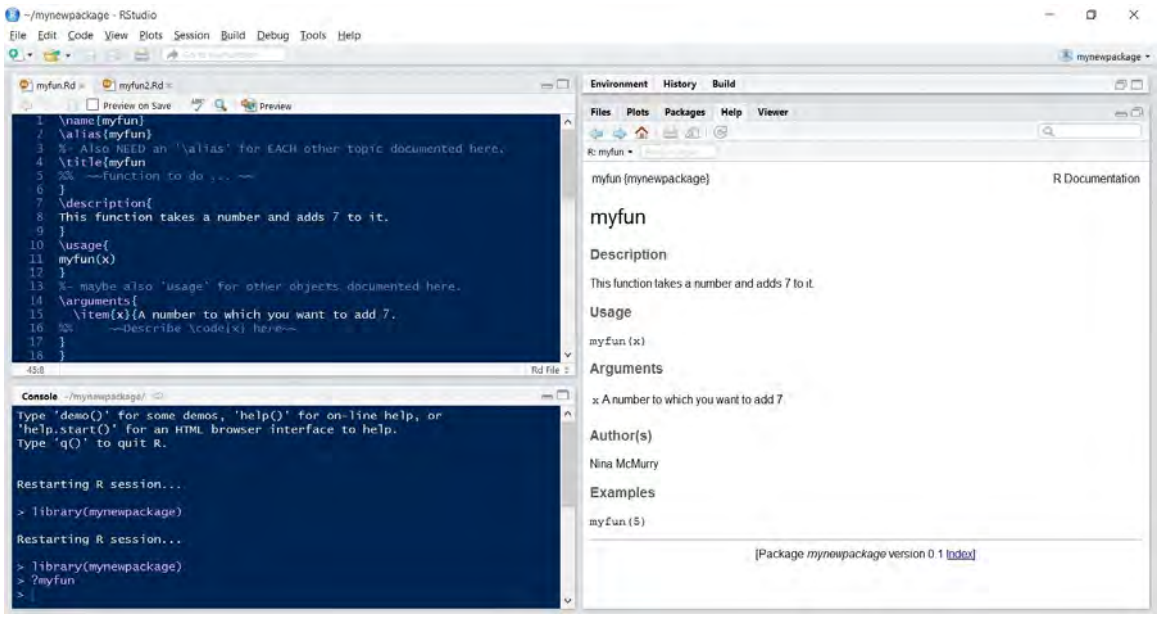
later point, even if RStudio automatically generated R documentation files when you initially created the package.



8. Now you are ready to compile your package. Go to 'Build' on the top toolbar and select 'Build and Reload' (note you can also use the keyboard shortcut Ctrl+Shift+B). If this works, your package will automatically load and you will see `library(mynewpackage)` at the bottom of your console. Test your functions to make sure they work.



9. Go back and edit the documentation (the help file) for each function. Open each `.Rd` file, add a brief description of the package, define its arguments and, if applicable, values, and include at least one example. Then, re-compile your package and test out your documentation in the R console (`?myfun`). *NOTE: You will need to re-compile (repeating step 8) each time you make changes to your functions or documentation.*



10. Once you have finished creating your functions and documentation, compiled your package, and double checked that the functions and help files work, copy the entire folder containing your package to the Dropbox folder with your name on it.

3 Building R Package with Command Line Tools

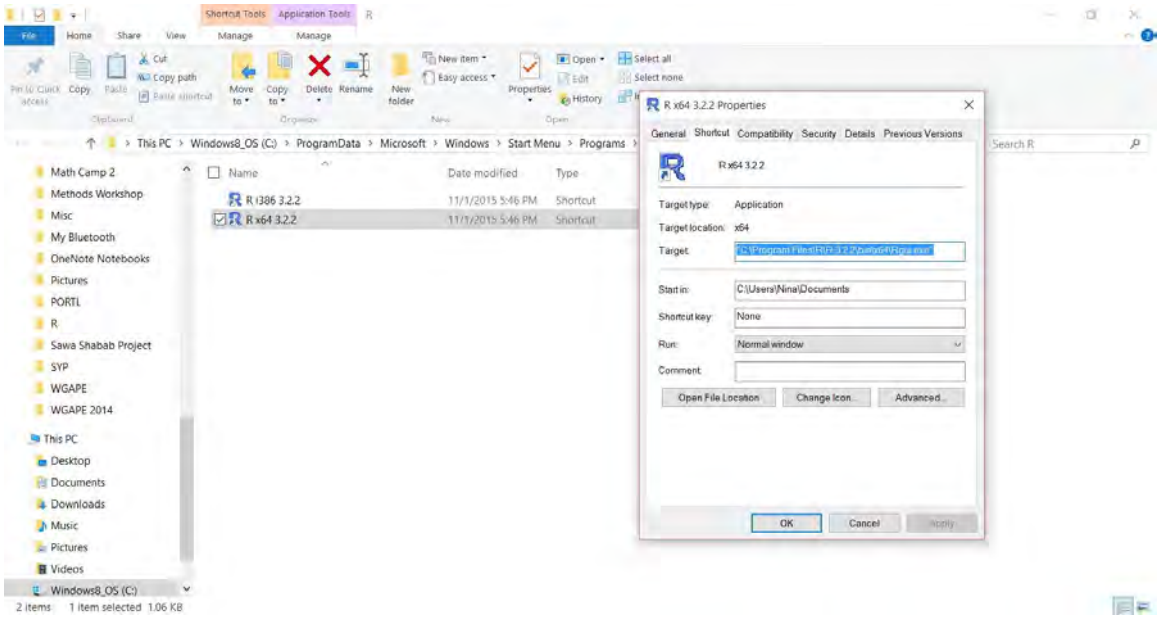
Note that there are some additional set-up requirements for Windows users only. Mac users may skip to step 6.

For Windows Users Only:

1. Install the latest version of R here: `https://cran.r-project.org/mirrors.html`. Be sure to uninstall previous versions of R (note that you will have to re-install all non-base packages).

2. Download and install Rtools here: `https://cran.r-project.org/bin/windows/Rtools/`. Make sure that the version of Rtools is compatible with your version of R.

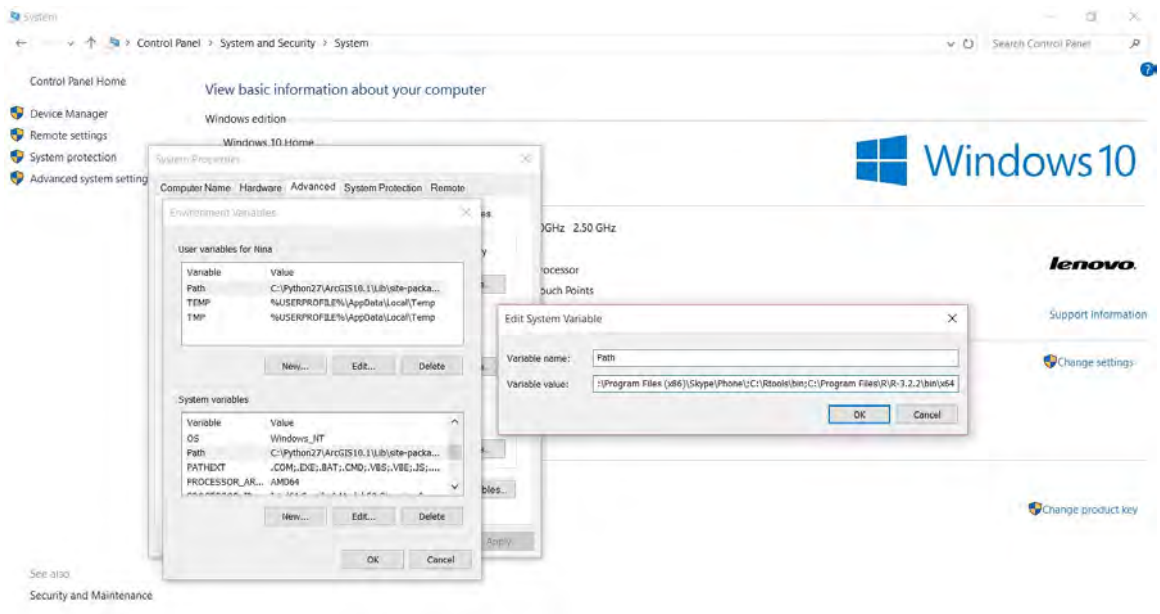
3. Now you will have to edit the environment variables in your system. Start by locating the R shortcut on your computer (*not* RStudio). Right click on the shortcut and select 'Properties.' Then copy the file path in the 'Target' field and paste it into Word or Notepad.



4. Open the Control Panel, then go to System and Security > System > Advanced System Settings > Environment Variables. Find the system variable "Path" and edit its variable value. Add the following to the variable value, separating each item from the others (and from the existing path) with semi-colons.

   – The file path for R that you copied down in step 3, but with the executable file at the end removed. For example, if the path is "C:\Program Files\R\R\-3.2.2\bin\x64\Rgui.exe" you should type "C:\Program Files\R\R\-3.2.2\bin\x64 "
   – The file path for Rtools: "C:\Rtools\bin" (make sure this is where Rtools is located on your computer)

   Your full addition to the existing path should look something like this: ";C:\Program Files\R\R\- 3.2.2\bin\x64;C:\Rtools\bin"



5. Open the terminal using the "Command Prompt" application. Type `path` and press return. The path should include the extensions you just added. If it does not, re-start your computer and try again.

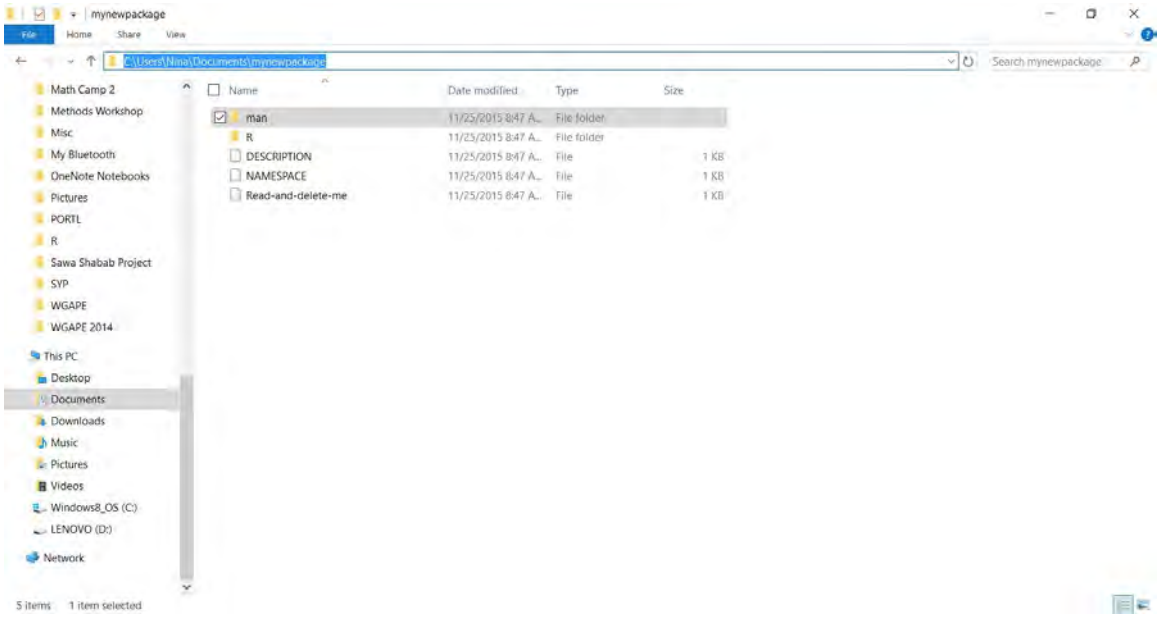
Mac Users Start Here:

6. Open R. You can use RStudio for this step if you wish. Start by checking your current directory using `getwd()` (e.g. "C:\Users\Nina\Documents").

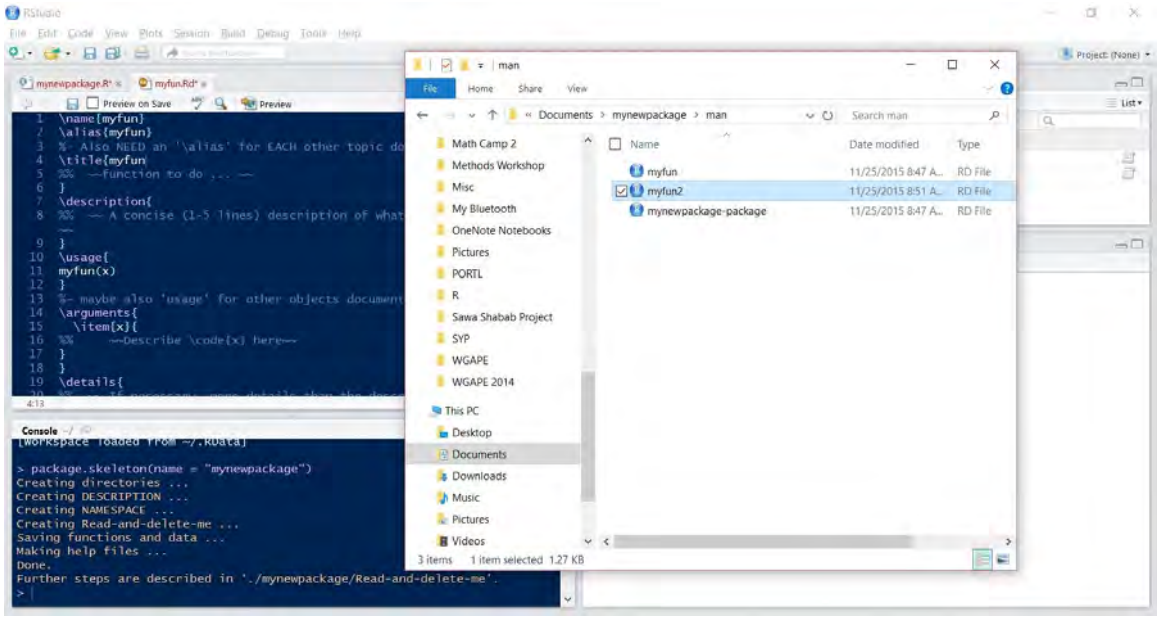
7. Remove everything from this directory using `rm(list = ls())`. Check to see that it is empty using `ls()` (you should see `character(0)`).

8. Open a new R script and write the code for your functions. In the same file, run `package.skeleton(name = "mynewpackage")`, inserting the name of your package in the `name` argument. This will create a new folder in the directory you found in step 6.

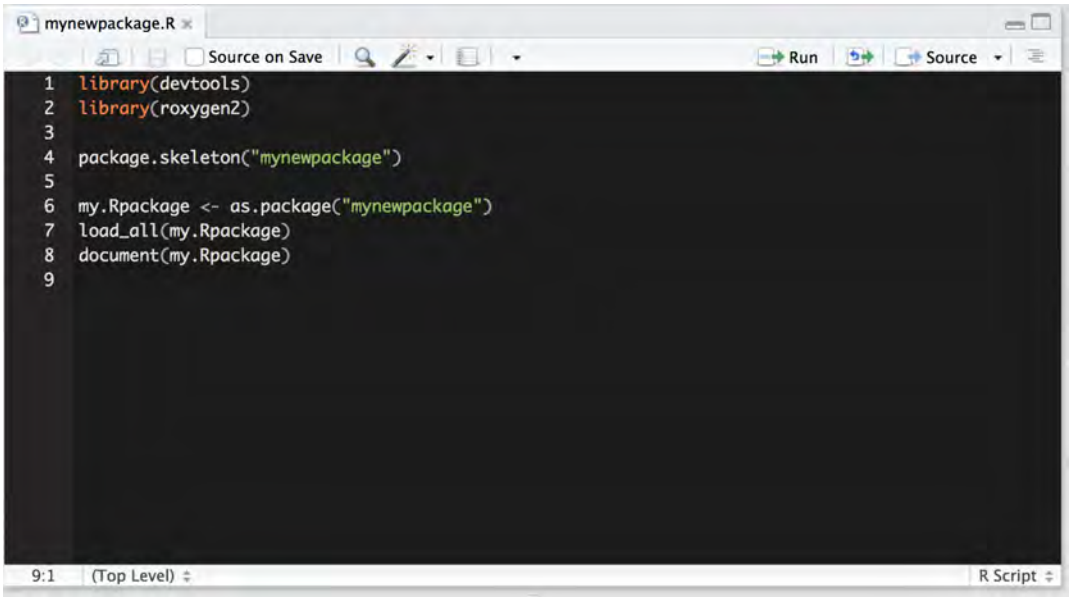
9. Navigate to this directory ("C:\Users\Nina\Documents"). You should see a folder with the name of your package. Navigate to the 'man' folder, which contains the help files for your functions in LaTeX code (e.g. "C:\Users\Nina\Documents\mynewpackage\man").



10. Open each `.Rd` file using your text editor (e.g. RStudio or Notepad), add a title under the 'title' heading and save. You can go back and edit the content later, but you will need to add a title to each `.Rd` file in order to compile your package. *NOTE: If there are no* `.Rd` *files in the 'man' folder, see step 6 under "Basic Instructions" for directions on how to create the documentation files manually.*

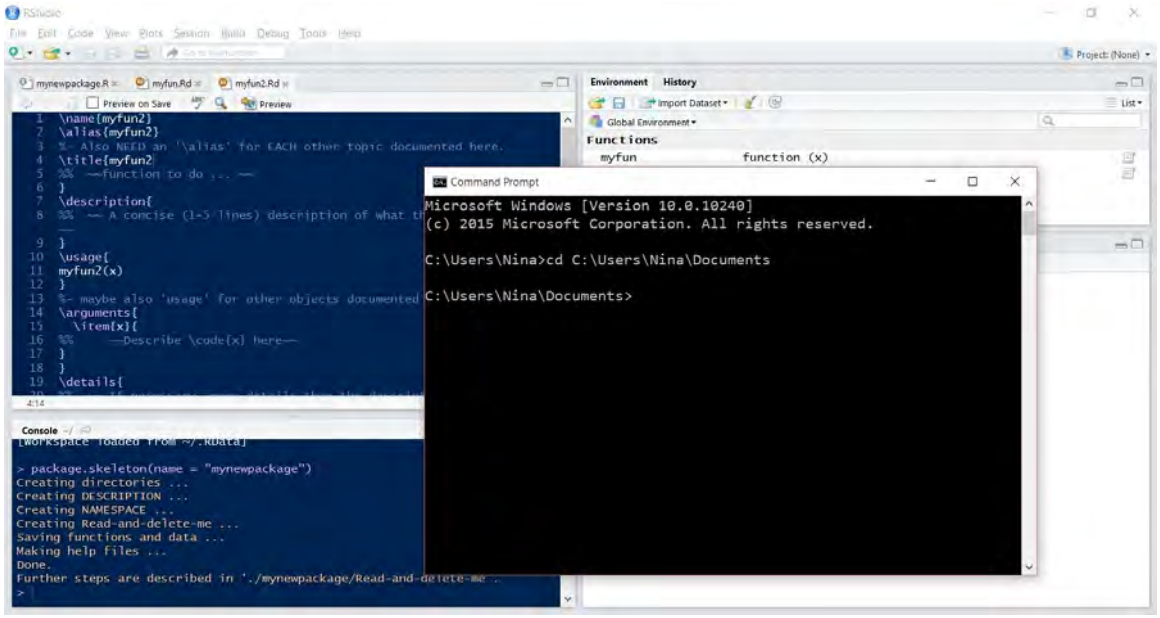


Before we build a package, we should check whether the functions are working and the package is loaded correctly. We can check this using `load_all` function. You can find out more information about `load_all` here.



On a side note, `roxygen2` would ease the documentation process. By running `document`, roxygen will automatically update the documentation. More information can be found in Hadley Wickham's writing. *NOTE: One should carefully examine the `Collate` field in `DESCRIPTION`. If a certain file should be loaded before another, the `Collate` order should follow it as well.*

11. Open the terminal. Windows users can open the application "Command Prompt." Mac users should open the "Terminal" application. Go to the directory where your package files are located by typing: `cd C:/Users/Nina/Documents/`



12. Now it is time to build your package.

    WINDOWS USERS: Type the following into the terminal (substituting your package name) and hit Return:

    ```
    Rcmd build --binary mynewpackage
    ```

    This will create `mynewpackage_1.0.tar.gz`, also known as a "tarball." Now install the package by typing the following into the terminal and hitting Return:

    ```
    Rcmd INSTALL mynewpackage_1.0.tar.gz
    ```

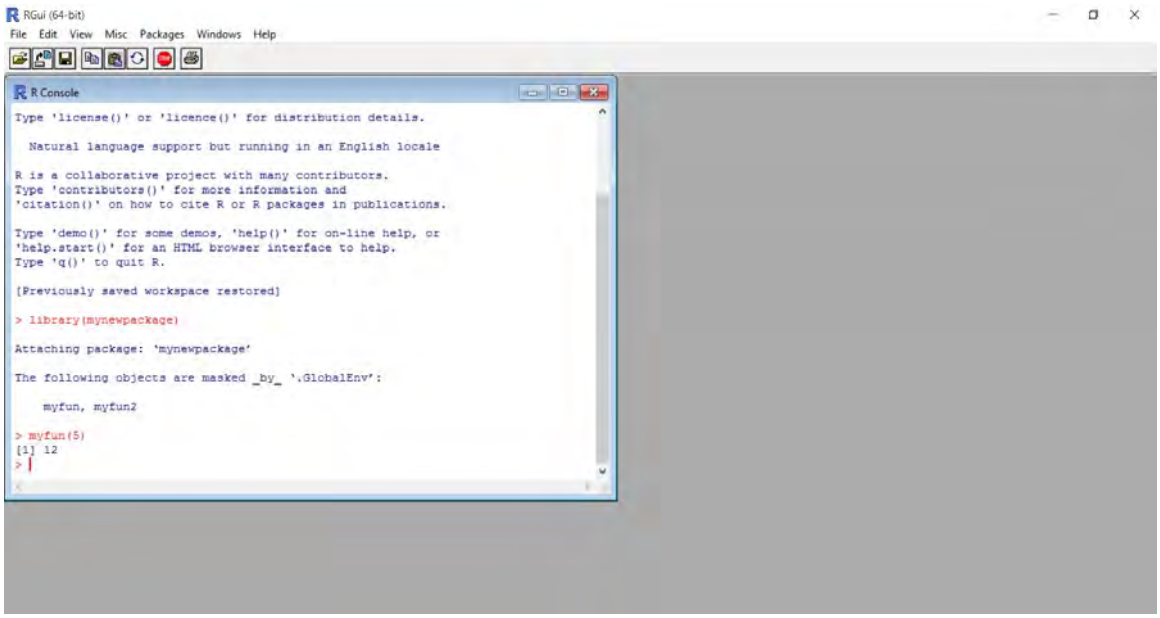
    You should see some outputs with `DONE(mynewpackage)` at the end.

    MAC USERS: Type the following into the terminal (substituting your package name) and hit Return after each line:

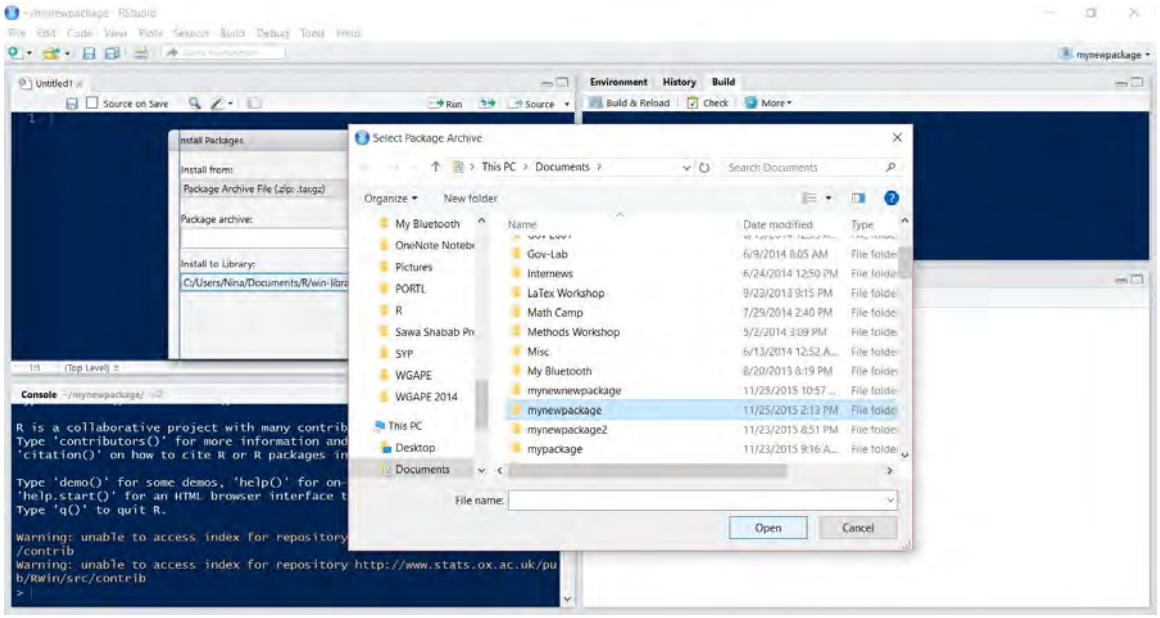
    ```
    R CMD build mynewpackage
    R CMD INSTALL mynewpackage_0.1.tar.gz
    ```

    Note. Additionally, before you build a package or submit it to CRAN, you would want to see if your package passes `R CMD check --as-cran mynewpackage_0.1.tar.gz`. Ideally, no errors or warnings should be found. You can easily execute this by running `check` function in `devtools`. More information can be found in here.

13. You should now be able to load your package in R GUI or by opening R from the terminal (see the next step if you want to open your package in RStudio). Test your functions to make sure they work correctly and test your help files to make sure they come up (`?myfun`).



14. To open your package in RStudio, open RStudio and go to Tools > Install Packages. In the 'Install from' menu, select 'Package Archive File'. Navigate to your tarball and select it to install. Load the package in RStudio with `library(mynewpackage)`. Test your functions and your help files.



15. Every time you make changes to the R files (in the 'R' file in your package folder) or help files (in the 'man' file in your package folder), you will have to repeat step 12 to re-build and re-install the package. If you are using RStudio, you might also have to repeat step 14.

16. Once you have finished creating your functions and documentation, compiled your package, and double checked that the functions and help files work, copy your tarball to the Dropbox folder with your name on it.

4 Building a R package with your own Rcpp Functions

Building a R package is more flexible and powerful than what we can do with `evalCpp()`, `cppFunction()`, or `sourceCpp()`. The instructions in this section will work for users using Mac OS X or Linux.

Windows users might want to start with steps 1–5 in Section 3 before getting started.

1. Write your own C++ function (e.g., q4rcpp.cpp).

   (a) Make sure to put the following at the top of your cpp code.

   ```
   # include <RcppArmadillo.h>
   // [[Rcpp::depends(RcppArmadillo)]]

   using namespace Rcpp ;
   ```

   (b) Put `// [[Rcpp::export()]]` right above the function that you want to use in R. This is just a comment from the perspective of C++. However, the Rcpp helper functions look for this line to determine the files that they generate when we compile the package. See `RcppExports.R` file created after you complete the compilation.

```
1   # include <RcppArmadillo.h>
2   // [[Rcpp::depends(RcppArmadillo)]]
3
4   using namespace Rcpp ;
5
6   // [[Rcpp::export()]]
7   int sumCpp(Rcpp::IntegerVector x) {
8     int n = x.size();
9     int res = 0;
10    for (int i = 0; i < n; i++){
11      res += x[i];
12    }
13    return res;
14  }
```

2. Create skeleton files using `RcppArmadillo.package.skeleton()` function.

   - If you do not need Armadillo C++ code, you can use `Rcpp.package.skeleton()` function alternatively.
   - If your package does not have any C++ code to be compiled, you can use R's default `package.skeleton()` function alternatively.

```
> library(Rcpp)
> library(RcppArmadillo)
> RcppArmadillo.package.skeleton("q4rcpp")

Calling package.skeleton to create basic package.
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './q4rcpp/Read-and-delete-me'.

Adding RcppArmadillo settings
 >> added Imports: Rcpp
 >> added LinkingTo: Rcpp, RcppArmadillo
 >> added useDynLib and importFrom directives to NAMESPACE
 >> added Makevars file with Rcpp settings
 >> added Makevars.win file with RcppArmadillo settings
 >> added example src file using armadillo classes
 >> added example Rd file for using armadillo classes
 >> invoked Rcpp::compileAttributes to create wrappers
```

   This will create a folder for your package

```
insong@E53-405-1:~/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage$ ls
q4rcpp
```

The folder will have the following basic contents for your package

- src: (a directory for C++ code to be compiled)
- man: a directory for help files (the manual for users)
- R: a directory for R code
- DESCRIPTION: basic description of your package
- NAMESPACE
- Read-and-delete-me

```
insong@E53-405-1:~/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage/q4rcpp$ la
total 24
drwxr-xr-x@ 6 insong  staff  204 Feb 11 09:56 src
drwxr-xr-x@ 4 insong  staff  136 Feb 11 09:56 man
-rw-r--r--@ 1 insong  staff  420 Feb 11 09:56 Read-and-delete-me
drwxr-xr-x@ 3 insong  staff  102 Feb 11 09:56 R
-rw-r--r--@ 1 insong  staff   75 Feb 11 09:56 NAMESPACE
-rw-r--r--@ 1 insong  staff  336 Feb 11 09:56 DESCRIPTION
drwxr-xr-x@ 3 insong  staff  102 Feb 11 09:56 ..
drwxr-xr-x@ 8 insong  staff  272 Feb 11 09:56 .
```

3. Put your q4rcpp.cpp file into `src` directory. You may want to register your C++ functions using this tool

4. You need to expose your Rcpp functions so that you can use them in R. Go to your package directory and open R. Execute the following

```
> compileAttributes(verbose=TRUE)
```

```
> library(Rcpp)
> compileAttributes(verbose=TRUE)
Exports from /Users/insong/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage/q4rcpp/src/q4rcpp.cpp:
   int sumCpp(Rcpp::IntegerVector x)
   void myloop()
   int sum_cpp(int sumto)
   Rcpp::NumericVector expCpp(Rcpp::NumericVector x)
   Rcpp::List myMatrix(Rcpp::IntegerVector x)

Exports from /Users/insong/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage/q4rcpp/src/rcpparma_hello_world.cpp:
   arma::mat rcpparma_hello_world()
   arma::mat rcpparma_outerproduct(const arma::colvec& x)
   double rcpparma_innerproduct(const arma::colvec& x)
   Rcpp::List rcpparma_bothproducts(const arma::colvec& x)

/Users/insong/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage/q4rcpp/src/RcppExports.cpp updated.
/Users/insong/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage/q4rcpp/R/RcppExports.R updated.
> q()
```

5. Register your Rcpp (or c++ and Fortran) functions.

```
> library(tools)
> package_native_routine_registration_skeleton("package-root-directory")
```

Running the above command will return a text string. Copy and paste and save it as `/src/init.c`

6. Go to your package directory and build your package. This will create a .tar.gz file

```
insong@E53-405-1:~/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage$ R CMD build q4rcpp
* checking for file 'q4rcpp/DESCRIPTION' ... OK
* preparing 'q4rcpp':
* checking DESCRIPTION meta-information ... OK
* cleaning src
* installing the package to process help pages
* saving partial Rd database
* cleaning src
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building 'q4rcpp_1.0.tar.gz'

insong@E53-405-1:~/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage$ ls -lart
total 16
drwxr-xr-x@ 31 insong  staff  1054 Feb 11 10:00 ..
drwxr-xr-x@  8 insong  staff   272 Feb 11 10:01 q4rcpp
drwxr-xr-x@  4 insong  staff   136 Feb 11 10:03 .
-rw-r--r--@  1 insong  staff  7191 Feb 11 10:05 q4rcpp_1.0.tar.gz
```

7. Compile your package

```
insong@E53-405-1:~/Dropbox (MIT)/quant4_2016/slides/rcpp/rpackage$ R CMD INSTALL q4rcpp_1.0.tar.gz
* installing to library '/Library/Frameworks/R.framework/Versions/3.2/Resources/library'
* installing *source* package 'q4rcpp' ...
** libs
clang++ -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG  -I/usr/local/include -I/usr/local/include/freetyp
e2 -I/opt/X11/include -I"/Library/Frameworks/R.framework/Versions/3.2/Resources/library/Rcpp/include" -I"/Library/Frame
works/R.framework/Versions/3.2/Resources/library/RcppArmadillo/include"   -fPIC  -Wall -mtune=core2 -g -O2  -c RcppExpo
rts.cpp -o RcppExports.o
clang++ -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG  -I/usr/local/include -I/usr/local/include/freetyp
e2 -I/opt/X11/include -I"/Library/Frameworks/R.framework/Versions/3.2/Resources/library/Rcpp/include" -I"/Library/Frame
works/R.framework/Versions/3.2/Resources/library/RcppArmadillo/include"   -fPIC  -Wall -mtune=core2 -g -O2  -c q4rcpp.c
pp -o q4rcpp.o
clang++ -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG  -I/usr/local/include -I/usr/local/include/freetyp
e2 -I/opt/X11/include -I"/Library/Frameworks/R.framework/Versions/3.2/Resources/library/Rcpp/include" -I"/Library/Frame
works/R.framework/Versions/3.2/Resources/library/RcppArmadillo/include"   -fPIC  -Wall -mtune=core2 -g -O2  -c rcpparma
_hello_world.cpp -o rcpparma_hello_world.o
clang++ -dynamiclib -Wl,-headerpad_max_install_names -undefined dynamic_lookup -single_module -multiply_defined suppres
s -L/Library/Frameworks/R.framework/Resources/lib -L/usr/local/lib -o q4rcpp.so RcppExports.o q4rcpp.o rcpparma_hello_w
orld.o -L/Library/Frameworks/R.framework/Resources/lib -lRlapack -L/Library/Frameworks/R.framework/Resources/lib -lRbla
s -L/usr/local/lib/gcc/x86_64-apple-darwin13.0.0/4.8.2 -lgfortran -lquadmath -lm -F/Library/Frameworks/R.framework/.. -
framework R -Wl,-framework -Wl,CoreFoundation
installing to /Library/Frameworks/R.framework/Versions/3.2/Resources/library/q4rcpp/libs
** R
** preparing package for lazy loading
** help
Warning: /private/var/folders/1f/pgl8r1_n26sfqyczxk8b83s00000gn/T/RtmpDXM2c9/Rbuild5c31dc80740/q4rcpp/man/q4rcpp-packag
e.Rd:27: All text must be in a section
Warning: /private/var/folders/1f/pgl8r1_n26sfqyczxk8b83s00000gn/T/RtmpDXM2c9/Rbuild5c31dc80740/q4rcpp/man/q4rcpp-packag
e.Rd:28: All text must be in a section
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (q4rcpp)
```

8. Now you can use your own Rcpp function!

```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (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.

> library(q4rcpp)
> a <- 1:10
> sumCpp(a)
[1] 55
```

Congratulations! You are now an R developer.

5 Useful Rcpp snippets

This section includes some useful Rcpp snippets that can be building blocks for more complex functions. All of these should be put in a `*.cpp` file and sourced through RStudio. R code can be included in a specially marked section and will run after the C++ code is compiled. All files should begin with

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;
```

This is a simple function to take two vectors from R, add them elementwise, and return a new vector. Note how Armadillo vectors are defined and note how it can be called from R.

```
1  // [[Rcpp::export]]
2  arma::vec add_vecs(arma::vec a,
3                     arma::vec b){
4    arma::vec c = a + b;
5    return(c);
6  }
7
8
9  /*** R
10 a <- c(1, 0, 0)
11 b <- c(1, 1, 0)
12 add_vecs(a, b)
13 */
```

Finding and subsetting elements of a vector by a condition is a common task in statistical programming. The `find` function will return the elements of a vector matching a condition. Note that vector positions for subsetting need to be stored in an "unsigned vector" (`uvec`) that only accepts positive integers. The `.elem()` method takes in a `uvec` and returns the elements of the vector corresponding to those positions. The equivalent for matrices is either `.rows()` or `.cols()`.

```
1  // [[Rcpp::export]]
2  arma::vec subset_vec(arma::vec a,
3                       arma::vec b){
4    arma::uvec subset = find(b > 1);
5    arma::vec c = a.elem(subset);
6    return(c);
7  }
8
9
10 /*** R
11 a <- c(1, 2, 3)
12 b <- c(1, 1, 5)
13 subset_vec(a, b)
14 */
```

A Appendix

A.1 Possible Errors with Solutions

1. Rcpp, RcppArmadillo Error on Mac OS "-lgfortran" and "-lquadmath"

```
curl -O http://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2
sudo tar fvxz gfortran-4.8.2-darwin13.tar.bz2 -C /
```

2. If you get an error similar to the following, install development tools available here

```
> Rcpp::sourceCpp("cmcmcprobit.cpp")
ld: warning: directory not found for option '-L/usr/local/lib/gcc/x86_64-apple-
    darwin13.0.0/4.8.2'
ld: library not found for -lgfortran
clang: error: linker command failed with exit code 1 (use -v to see invocation)
make: *** [sourceCpp_32.so] Error 1
clang++ -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG -I/usr/
    local/include -I/usr/local/include/freetype2 -I/opt/X11/include -I"/Library
    /Frameworks/R.framework/Versions/3.2/Resources/library/Rcpp/include" -I"/
    Library/Frameworks/R.framework/Versions/3.2/Resources/library/RcppArmadillo
    /include" -I"/Users/guillermotoral/Box Sync/Studies/PhD/Coursework/Methods/
    Quantitative Methods 4/Problem Sets/Problem set 2" -fPIC -Wall -mtune=core2
     -g -O2 -c cmcmcprobit.cpp -o cmcmcprobit.o
clang++ -dynamiclib -Wl,-headerpad_max_install_names -undefined dynamic_lookup
    -single_module -multiply_defined suppress -L/Library/Frameworks/R.framework
    /Resources/lib -L/usr/local/lib -o sourceCpp_32.so cmcmcprobit.o -L/Library
    /Frameworks/R.framework/Resources/lib -lRlapack -L/Library/Frameworks/
    R.framework/Resources/lib -lRblas -L/usr/local/lib/gcc/x86_64-apple-
    darwin13.0.0/4.8.2 -lgfortran -lquadmath -lm -F/Library/Frameworks/
    R.framework/.. -framework R -Wl,-framework -Wl,CoreFoundation
Error in Rcpp::sourceCpp("cmcmcprobit.cpp") :
Error 1 occurred building shared library.
```