Automating Building RcppArmadillo dependent R Packages Using R and RStudio

James Yu

November 9, 2019

Abstract

This is a set of instructions on how to build an R package that requires RcppArmadillo. We explore two methods to cater for users of R and RStudio. Note that we do not cover documentation of packages in this set of notes. I also show how to make an R package that uses only R, and another that uses Rcpp.

Thanks to Weichang Yu for helping me figure this out.

1 Automating Building R Packages Using RStudio

The RStudio suite includes a click based application to help create R packages. Although it works well for R and Rcpp based functions, it requires some changes when used with RcppArmadillo. R functions that require functions from other packages also require some finessing. Additionally, the RStudio app seems to allow only one function to be imported at a time. With this in mind, we also explore a slightly more intensive method that is more suitable for larger packages.

1.1 For functions using R language

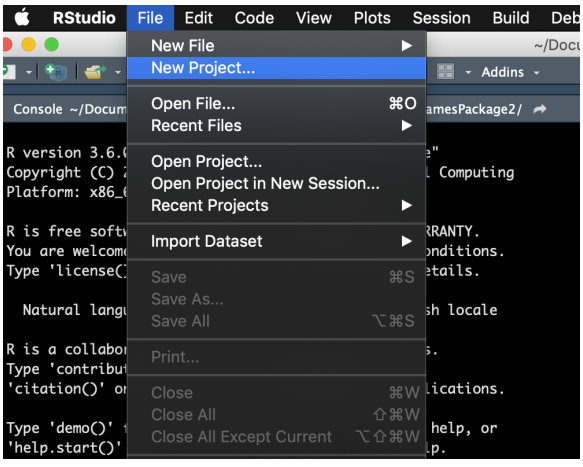
First consider an R package with the functions:

- vec.r
- matrixbuilder.R
- dummycol.R

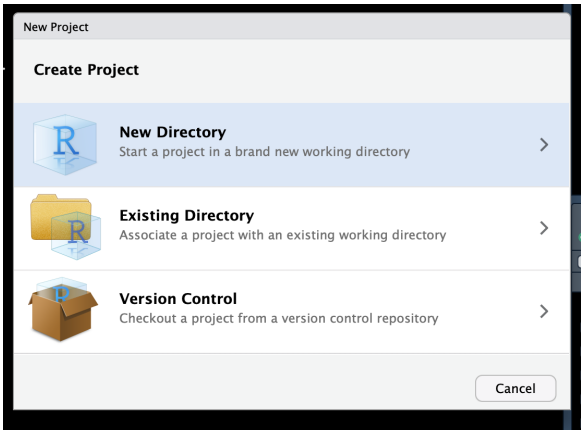
The vec() function in the "vec.r" script depends on only base R functions. The matrixbuilder() function in the "matrixbuilder.R" script depends on dummycol() function in the "dummycol.R" script and the R package "stringr". Although it is not difficult to link dependent functions, we must take care to update the namespace for the required packages other than base R.

1. We will first generate the required file structure and ".Rproj" file. To do this:

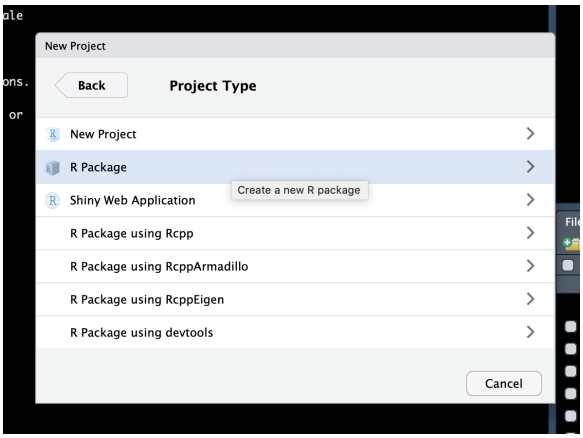
   (a) Click: File > New Project...



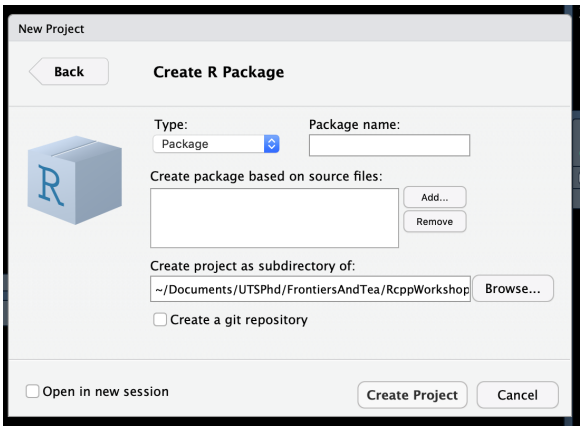
(b) Click: New Directory



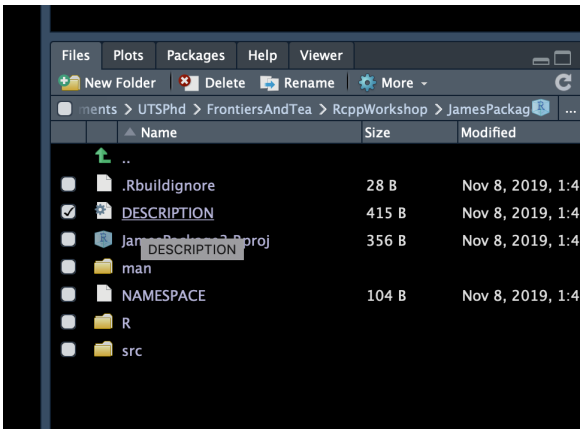
(c) Click: R Package



(d) Since we want to add multiple source files, we won't add them here. However, if you want to only add one file, you can do so by clickling the "Add" button under "Create package based on source file:". Under "Create project as subdirectory of:" check that the location is appropriate to make a new directory in. You can change the directory by clicking the "Browse..." button. Once happy, Click: Create Project.



2. There should now be a new directory structure that forms the R package in the location you specified in the previous step. You can view it from RStudio, via the "Files" tab on the bottom right pane.

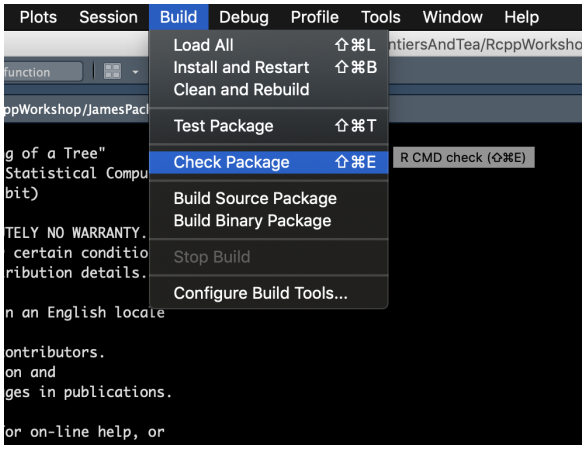


Since we are including a function that relies on another package, we must update the NAMESPACE and DESCRIPTION files to reflect this. The changes required for each file are as follows:

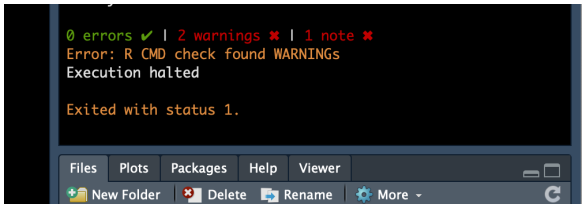
- NAMESPACE
  - Add a new line with the following: import (stringr)
- DESCRIPTION
  - Add a new line with the following: Imports: stringr

3. If you have not added in the source code, you can now do it. .r/.R scripts should be added to the "R" directory. You can do this however you feel comfortable.

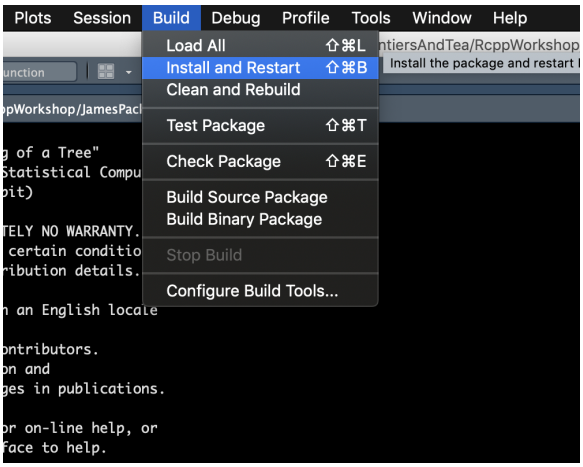
4. We now check whether the package is formatted correctly and suitable to build. To do this, Click: Build > Check Package



The status of the package check is given in the top right panel. Since we have not created any documentation for the package it is normal to get warning messages as shown below.



5. Assuming there were no errors, we can build and install the package. We also restart R to ensure no objects are mis-specified. To do this, Click: Build > Install and Restart



6. You should now be able to access the functions specified in the packages.

1.2 For functions using both Rcpp and RcppArmadillo

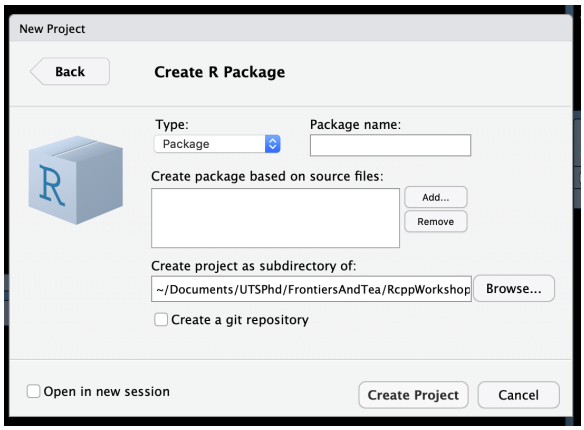
Leaving R fucntions aside for the moment, we will now show how to get an R package working with Rcpp and with RcppArmadillo. The functions we will use as are available for download on http://matt-wand.utsacademics.info/RCPPDG.html:

- cornflower.cpp
- courtJester.cpp
- yakMilk.cpp
- sumRecips.cpp
- myRecip.cpp
- prawnDumpling.cpp
- sumSqrts.cpp
- mySqrt.cpp
- cornflower.h
- myRecip.h
- mySqrt.h
- prawnDumpling.h

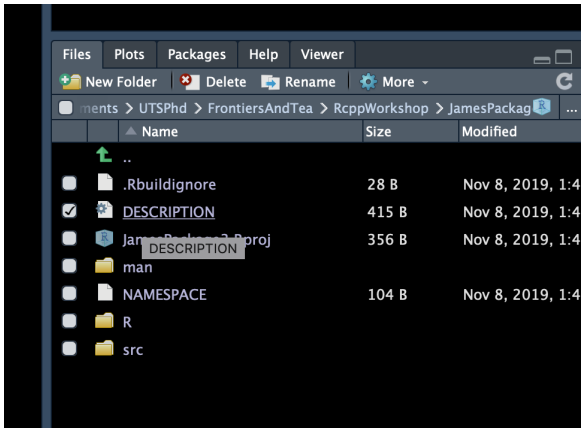
In RStudio, follow these steps (note steps 1.(a) to 1.(d) are the same as previously described):

1. We will first generate the required file structure and ".Rproj" file. To do this:

   (d) Since we want to add multiple source files, we won't add them here. However, if you want to only add one file, you can do so by clicking the "Add" button under "Create package based on source file:". Under "Create project as subdirectory of:" check that the location is appropriate to make a new directory in. You can change the directory by clicking the "Browse..." button. Once happy, Click: Create Project.



2. There should now be a new directory structure that forms the R package in the location you specified in the previous step. You can view it from RStudio, via the "Files" tab on the bottom right pane.



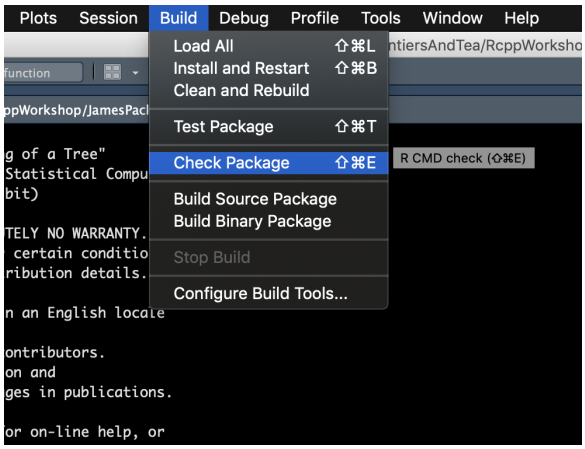
If your functions use RcppArmadillo code you must now make alternations to the NAMESPACE and DESCRIPTION files. The changes required for

each file are as follows:

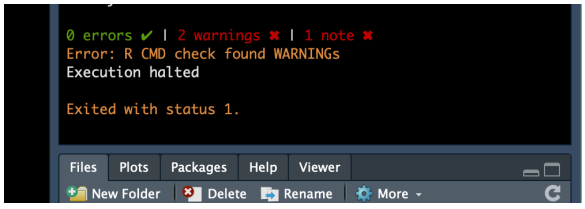
- NAMESPACE
  - Add a new line with the following: import (RcppArmadillo)
  - Delete the following: ",.registration = TRUE"
- DESCRIPTION
  - License: GPL-3
  - Imports: Rcpp,RcppArmadillo
  - LinkingTo: Rcpp,RcppArmadillo

3. Now we must add in the source code. Those containing .Cpp suffix should be addded to the "src" directory. If .r/.R scripts are included, they can be added to the "R" directory.

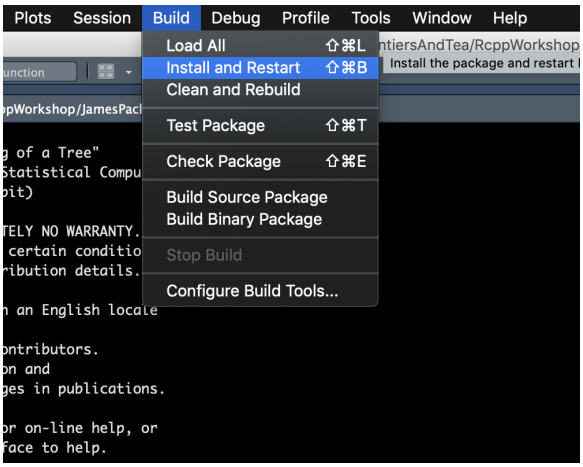
4. We now check whether the package is formatted correctly and suitable to build. To do this, Click: Build > Check Package



The status of the package check is given in the top right panel. Since we have not created any documentation for the package it is normal to get warning messages as shown below.



5. Assuming there were no errors, we can build and install the package. We also restart R to ensure no objects are mis-specified. To do this, Click: Build > Install and Restart



6. You should now be able to access the functions specified in the packages.

Note that if your package includes R scripts, you should add in the changes to the NAMESPACE and DESCRIPTION files as mentioned in the R language section.

2 Automating building R packages using R

In this section, I show how we can use R commands to achieve the same result as with the RStudio app. Again we do not consider documentation of the package, and also use the same functions as in the previous section.

2.1 For functions using Rcpp and RcppArmadillo

For functions that rely only on Rcpp, this method is brutally effective. For those that also require RcppArmadillo, the same ammendments are required. The following code creates the package skeleton:

```
Rcpp::Rcpp.package.skeleton(name = "PlayPen",
        example_code = FALSE,
        cpp_files = c("myRecip.cpp",
        "myRecip.h",
        "mySqrt.cpp",
        "mySqrt.h"),
        force = FALSE)
```

This code creates the package skeleton. Although we can specify all the .cpp files under the "cpp_files" agrument, since we have a large number we manually place the files into the src directory as we did in the previous section. We set the arguments "force" and example code to false. The "force" argument specifies whether or not to overwrite the a directory if one exists that has the same name. The "example_code" argument adds example .cpp code to the package, however since we have written our own scripts we do not worry about it.

If your functions use RcppArmadillo code you must now make alternations to the NAMESPACE and DESCRIPTION files. These are the same changes documented in the RStudio method (See step 2.). We then issue the command:

```
Rcpp::compileAttributes(pkgdir ="PlayPen",
        verbose = TRUE)
```

which compiles the cpp functions. Analogous to the previous section, we can then check whether package is working and subsequently install it by issuing the commands:

```
devtools::check(pkg = "PlayPen")
devtools::install(pkg = "PlayPen")
```

3 Automating building R packages using R

In this section, I show how we can use R commands to automate builidng R packages. To build file structure of the package, we issue the following command:

```
package.skeleton(name = "PlayPen",
code_files = c("dummycol.R",
        "matrixbuilder.R",
```

```
        "vec.r"),
        force=FALSE)
```

This skeleton includes a numer of documentation and help files that should be removed before attempting to check whether the package is working. Specifically, since for this tutorial we do not consider documentation, it is suggested that all files in the "man" directory are removed. Additionally, we must update the NAMESPACE and DESCRIPTION files as with RStudio. We can then issue the following commands to check and install the package.

```
devtools::check(pkg = "PlayPen")
devtools::install(pkg = "PlayPen")
```