

A quick guide to build your own R Package!

Daniel Hua

11/01/2017

Introduction

Ever have the feeling that you just can't find the right functions to use? Have you ever created some handy functions that you wish you could hold on to? Well, now you can! This post aims to introduce the basic techniques needed to build your very own R packages with customized functions tailored to your needs. No more writing the same functions over and over again for each project. Instead, simply take a few minutes to create your own packages that you could use whenever and wherever you want.

Step 0: Setting up the environment

There are two critical packages required for building the packages, described as follow:

- **devtools**, a package with useful package development functions to make our lives easier
- **roxygen2**, a package for documenting your functions

Install them with the following command:

```
install.packages("devtools")
install.packages("roxygen2")
```

After installing the packages, load them into the environment:

```
library("devtools")
```

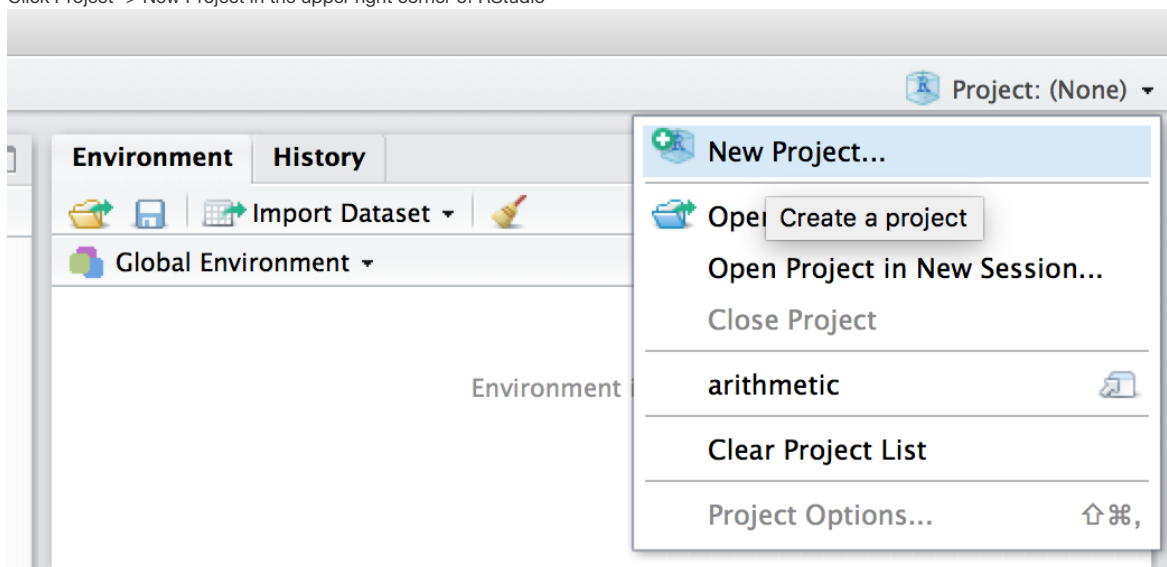
```
## Warning: package 'devtools' was built under R version 3.4.2
```

```
library("roxygen2")
```

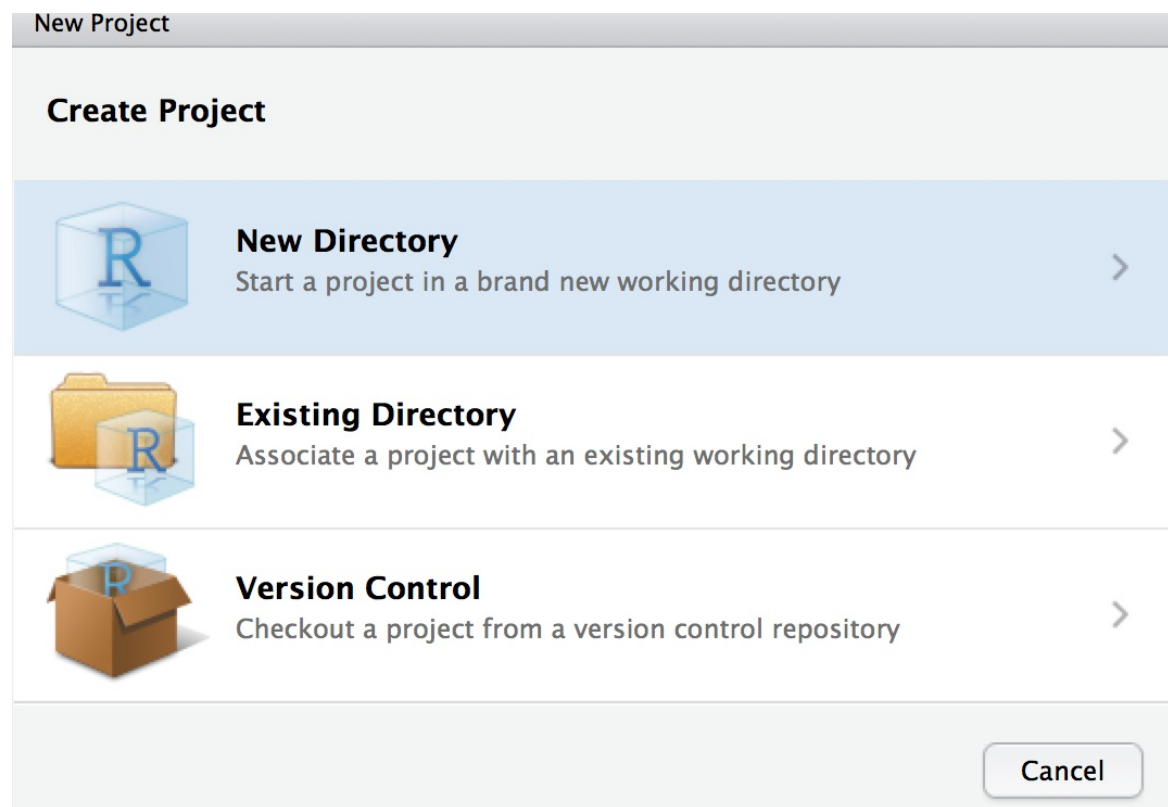
Step 1: Create the Framework for the package

In RStudio, simply follow the steps below to create the directory for your packages:

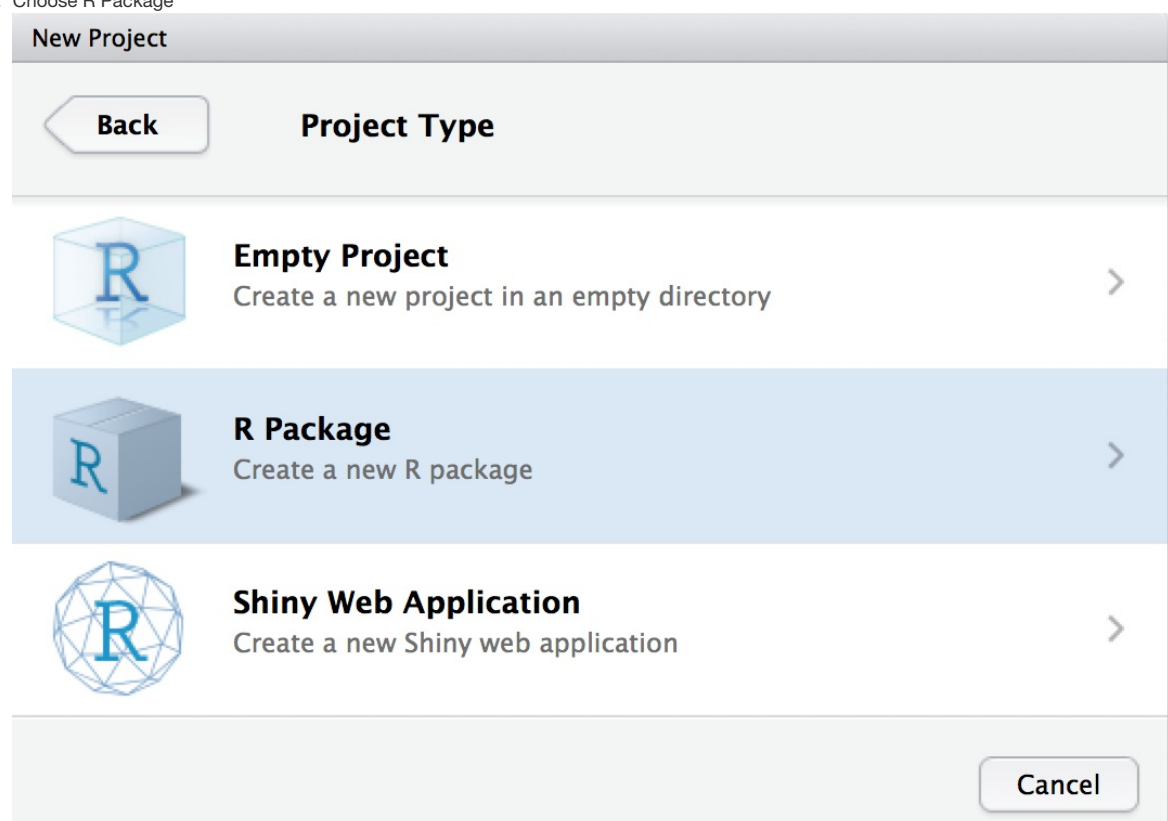
1. Click Project -> New Project in the upper right corner of RStudio



2. Choose New Directory



3. Choose R Package




4. give your package a name and choose its directory. Then click on Create Project

New Project

Back

Create R Package



Type:

Package

Package name:

EuclideanNorm

Create package based on source files:

Add...

Remove

Create project as subdirectory of:

~/Desktop/stat133/stat133-hws-fall17/post02

Browse...

☐ Open in new session

Create Project

Cancel

Alternatively, you could also type in the following command at the desired location if you are working on the console:

```
devtools::create("EuclideanNorm")
```

Structure of the package directory

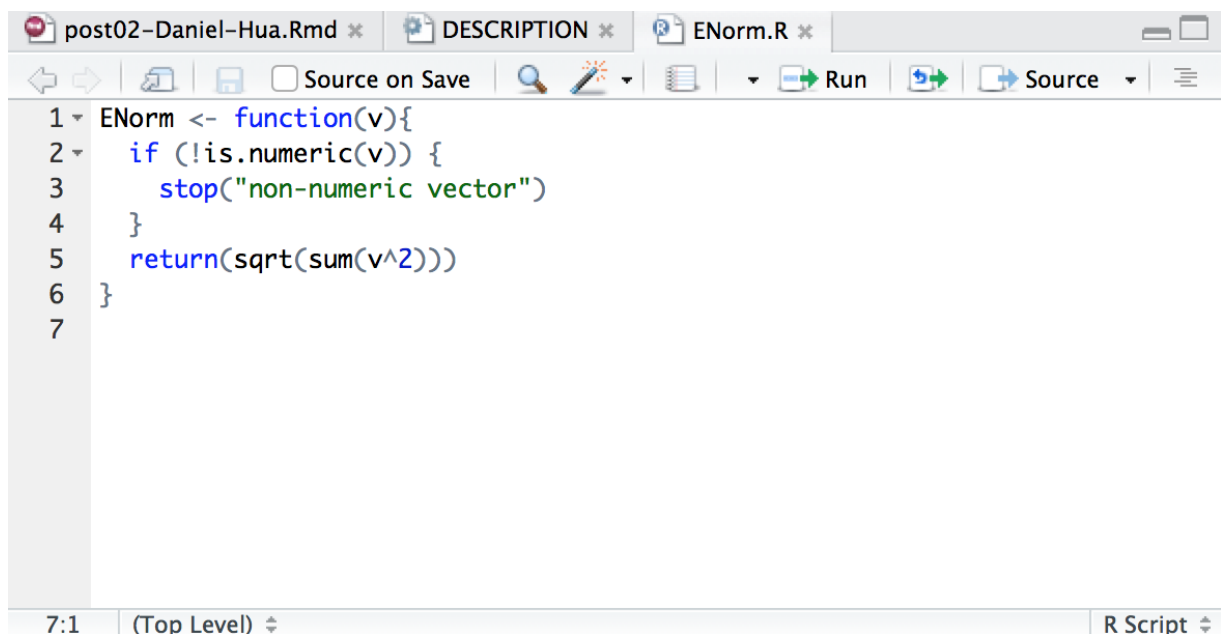
In the directory created from the steps above, it should contain the following files and folders:

- DESCRIPTION : A general description of the package, the content should be modified accordingly
 - Package: EuclideanNorm
 - Type: Package
 - Title: Taking the Euclidean norm of a vector
 - Version: 0.1.0
 - Author: Daniel Hua
 - Maintainer: The package maintainer <daniel.hua@berkeley.edu>
 - Description: Return the Euclidean norm(2-norm) of a vector
 - License: What license is it under?
 - Encoding: UTF-8
 - LazyData: true

- EuclideanNorm.Rproj: a RStudio file for the package
- NAMESPACE: A file indicates what needs to be exposed to users for your R package
- R: A folder containing all the R functions for the package
- man: A folder containing all the documentation for the R functions

Step 2: Adding your own functions to the package

As mentioned above, all the function files should be saved to the R folder. In this demo, I created a function called ENorm which returns the Euclidean norm of a vector and saved it in a R script file in the R folder



Depending on your preference, you could group several functions into a single R script file or you could also write a R script file for each function.

Step 3: Documentation for your functions

Documentation is the page you look at when you searched for the help for the function in R using the following command:

```
?ENorm
```

Writing documentations could be tedious in the base format since it requires knowledge of LaTeX, a typesetting system like html as we learned in class. However, thanks to the package **roxygen2**, writing documentations for our functions becomes way more easier since we only needs to add a few line of comments above the function and **roxygen2** would automatically generate the documentations in the man folder based on our comments.

The comments to document the ENorm function is shown as follow:

```

#' Euclidean Norm
#'
#' This function calculates the Euclidean Norm(2-norm) of a vector
#' and returns the result.
#' @param v a numeric vector for calculating the norm
#' @return a numeric value of the Euclidean norm of the input vector
#' @export
ENorm <- function(v){
  if (!is.numeric(v)) {
    stop("non-numeric vector")
  }
  return(sqrt(sum(v^2)))
}

```

Notice that each line start with #' and each @ specify a parameter for the function. the last line:

```
#' @export
```

Indicates that this function would be exposed to users.

Step 4: Using functions from other package

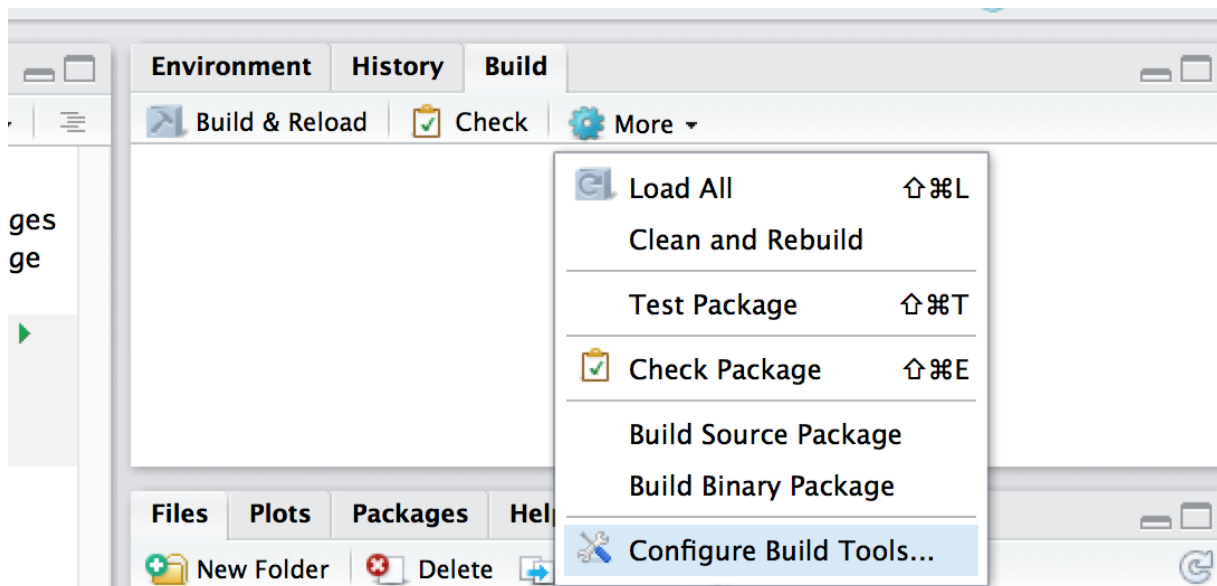
Sometimes, we might want to use functions from other external packages in our own functions. To do this, we first need to import the package needed by adding the following code in our DESCRIPTION file:

```
Imports:
  dplyr
```

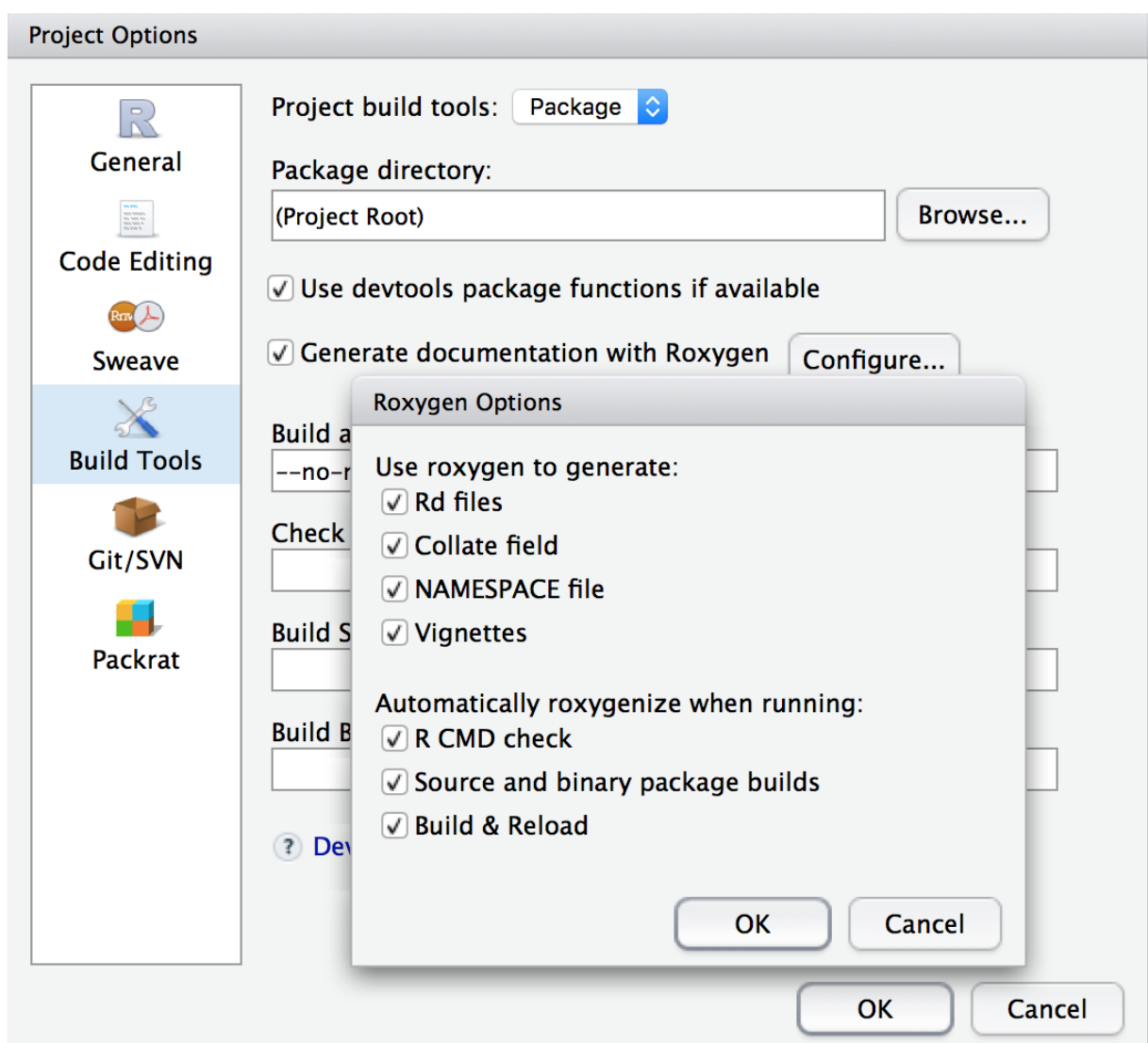
After modifying the DESCRIPTION file, functions from the indicated packages could be safely used in our own functions.

Step 5: Build and install your package

After finished with the documentation, we could actually get started with building the package! To do this, simply choose the Build tab on the upper left tabs of RStudio. Choose More -> Configure Build Tools as illustrated in the image:



Click on Generate documentation with Roxygen, in the Options popup, choose all the boxes. Then click OK.



After the configuration is done, we could finally build our package by click on the Build&Reload in the Build tab.

Congratulation, you just successfully installed your very own package in R!

Step 6: Checking the documentation

To check that we have correctly install the documentation for the package, first type in the name of our package: EuclideanNorm in the package tab as we would search for any other package. Click on the name of our package, we should see the documentation as follow:

Taking the Euclidean norm of a vector



Documentation for package 'EuclideanNorm' version 0.1.0

- [DESCRIPTION file](#).

Continue to click on the ENorm link in the helper page, we should be able to see the documentation for our function:

Euclidean Norm

Description

This function calculates the Euclidean Norm(2-norm) of a vector and returns the result.

Usage

`ENorm(v)`

Arguments

v a numeric vector for calculating the norm

Value

Step 7: Checking the functionality of the package

Confirmed that our documentation is indeed correct, we could check the actual functionality of our functions in the package. We could do this by doing something similar to the code chunk below:

```
library(EuclideanNorm)
ENorm(c(3,4))
```

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

Step 8: Testing

Testing is essential for any functions to have robust functionalities and become bug-free. To implement testing for your own package, simply do the above and write tests using **testthat** as we did for homework 4:

```
library(testthat)
context("ENorm")
test_that("Euclidean Norm is correct", {
  expect_equal(ENorm(c(3,4)), 5)
  expect_equal(ENorm(c(1,1,1,1)), 2)
  expect_equal(ENorm(c(10)), 10)
})
```

In your tests, try to include as many corner cases as possible to ensure your function's strength.

Step 9(optional): Distributing your package

If you would like to share your package for others to use, one simple way to do it would be to push your package repository to your github account. Afterwards, anyone who are interested could simply type in the following command to install your package:

```
devtools::install_github("yourusername/EuclideanNorm")
```

Conclusion and Take-home message

Creating your own package could seem intimidating at first. However, it is an extremely useful practice in data analysis since it is very often to perform the same set of algorithm over and over again for analyzing a large set of data. Taking a small amount of time to create a customized package could definitely be very useful and is only a very small investment in the long run.

For further information, please take a look at the reference list.

Reference list

- <http://tinyheero.github.io/jekyll/update/2015/07/26/making-your-first-R-package.html>
- <https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/>
- [Creating R Packages: A Tutorial](#)
- [Making an R Package \(R.M.Ripley\)](#)

Video tutorials I relied on:

- https://www.youtube.com/watch?v=WK3_JAPP7ZM
- <https://www.youtube.com/watch?v=9PyQlbAEujY&t=17s>
- <https://www.youtube.com/watch?v=rmiCnQEnB3g&t=6s>

Some advanced further readings:

- [Hadley Wickham's "R packages"](#)
- [More on testing](#)

Thanks for reading and have fun creating your own packages!