Creating R Packages: A Tutorial

Sam Gabor

3/8/2021

1.0 Overview

Creating custom R packages allows users to exchange code logic and data effortlessly. In addition, even if R packages are not shared with others, they still provide utility for the R user because they group related functionality into clean, easily maintained and tested units.

This tutorial will illustrate the creation of a non-trivial R package covering all the steps that constitute a good work flow. It culminates in publishing the created package on GitHub where it can be shared with other users.

2.0 Motivation

The first step in creating an R package is deciding what functionality and/or data will be packaged as a unit. For this tutorial, I chose to package access to the CDC's "Provisional COVID-19 Death Counts by Sex, Age, and State" data set (see: https://dev.socrata.com/foundry/data.cdc.gov/9bhg-hcku (https://dev.socrata.com/foundry/data.cdc.gov/9bhg-hcku)). The CDC provides this periodically updated data set through download and a REST API. The site does not require an API key so it's easy to work with for illustrative purposes.

Of course, for a bulk, one-time analysis of the data, the file download may be adequate, but for multiple analysis over time, it's better to have an easier way to fetch cleaned data that can also be filtered by content (instead of downloading the whole data set and then filtering and cleaning).

Thus, the R package in this tutorial, dubbed PCD, will do the following:

1. Fetch the latest dataset, optionally filtered by State
2. Allow the user to specify the maximum number of records returned
3. Clean the fetched data (by converting strings to proper values, such as dates, integers, etc.)

The end result is that the package user will be able to start working with the latest data immediately for charts and analysis without having to worry about downloading or preparing the data.

3.0 Prerequisites

To create R packages, you should install the latest version of R and RStudio then the following packages in your environment:

```
install.packages(c("devtools", "roxygen2", "testthat", "knitr"))
```

In addition, you should download and install the R build toolchain from: https://cran.r-project.org/bin/windows/Rtools/ (https://cran.r-project.org/bin/windows/Rtools/). This may not be needed until you build packages in C/C++, but if it's not installed you will always see a warning during package builds (as you'll see in this tutorial).

4.0 Creating Initial Package

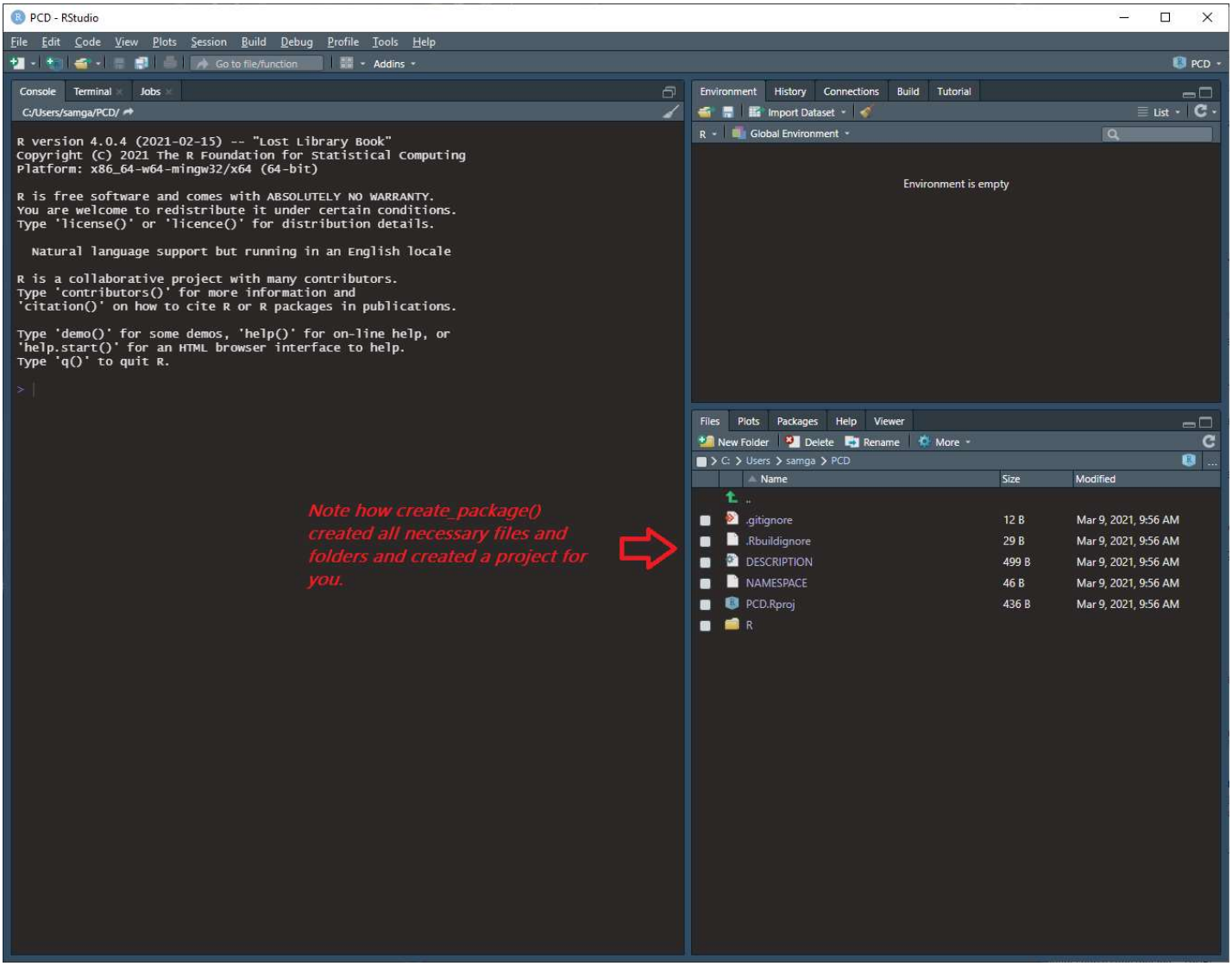
Once all the pre-requisites are installed, launch RStudio and run the following command:

```
library(devtools)
create_package("~/PCD")
```

Expect to see output similar to the below

```
√ Creating 'C:/Users/samga/PCD/'
√ Setting active project to 'C:/Users/samga/PCD'
√ Creating 'R/'
√ Writing 'DESCRIPTION'
Package: PCD
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R (parsed):
    * First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to
    pick a license
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
√ Writing 'NAMESPACE'
√ Writing 'PCD.Rproj'
√ Adding '^PCD\\.Rproj$' to '.Rbuildignore'
√ Adding '.Rproj.user' to '.gitignore'
√ Adding '^\\.Rproj\\.user$' to '.Rbuildignore'
√ Opening 'C:/Users/samga/PCD/' in new RStudio session
√ Setting active project to '<no active project>'
```

This will initialize a new package folder called PCD in your home directory and launch a new instance of RStudio. Your display should be similar to the below.



RStudio session (after create_package() call)

5.0 Git Integration

If you plan to share your package with others or track its development evolution over time, you should integrate git in the process early on. To do this, **in the new RStudio instance for your project**, enter the following:

```
use_git()
```

This will prompt you to commit the new files to the repo and also ask you if you want to restart the RStudio session to activate the Git tab. Your display should be similar to the below.

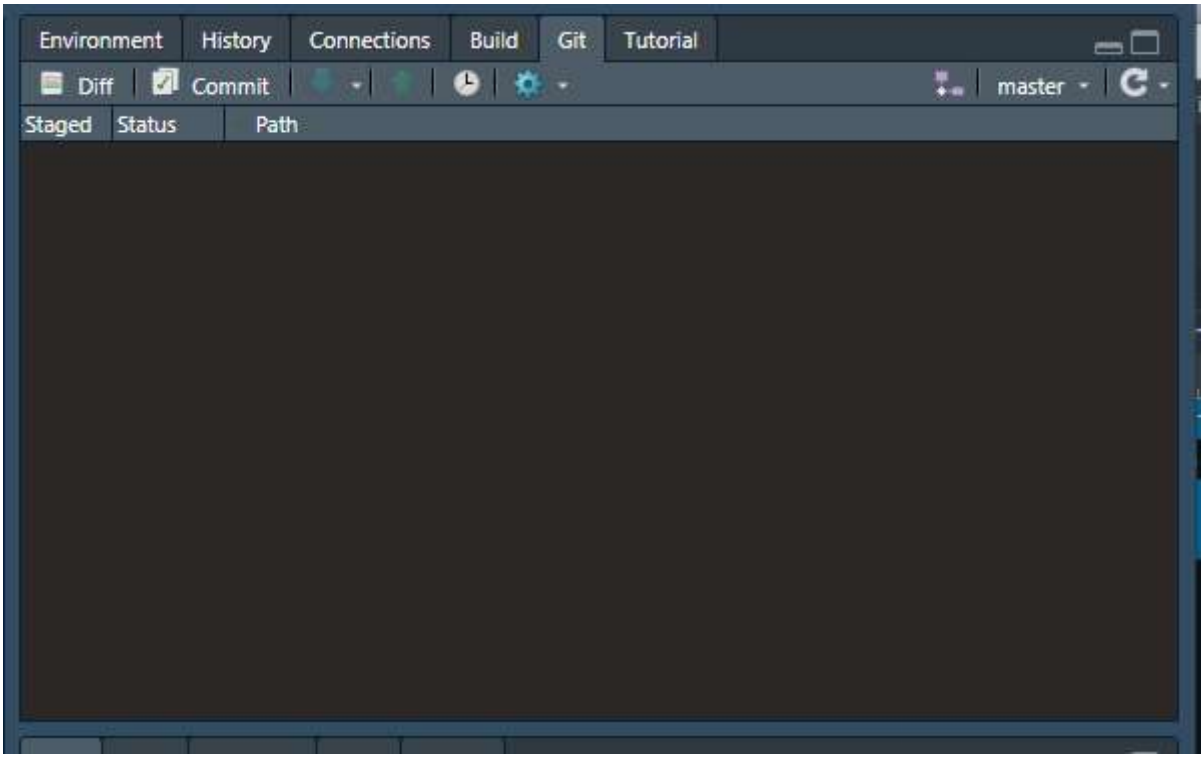
```
√ Setting active project to 'C:/Users/samga/PCD'
√ Initialising Git repo
√ Adding '.Rhistory', '.Rdata', '.httr-oauth', '.DS_Store' to '.gitignore'
There are 5 uncommitted files:
* '.gitignore'
* '.Rbuildignore'
* 'DESCRIPTION'
* 'NAMESPACE'
* 'PCD.Rproj'
Is it ok to commit them?

1: Negative
2: Absolutely not
3: I agree

Selection: 3
√ Adding files
√ Making a commit with message 'Initial commit'
* A restart of RStudio is required to activate the Git pane
Restart now?

1: Yeah
2: Absolutely not
3: No
Selection: 1
```

When RStudio is launched, you should see a Git tab in the Environment/History/Build pane as below:



Git Integration

6.0 Dependent Packages Inclusion

Your R code may depend on other R packages and those have to be included as well. In the case of the PCD package, it depends on the `jsonlite` package

```
use_package("jsonlite")
```

Note that `use_packages()` automatically updates the package's Imports information as it reports below:

```
✓ Setting active project to 'C:/Users/samga/PCD'
✓ Adding 'jsonlite' to Imports field in DESCRIPTION
* Refer to functions with `jsonlite::fun()`
```

7.0 Writing the Package R Code

Now we add the main substance in the package. We have to add the R code that implements the logic we're packaging. To do this you can automatically create the associated R file by invoking the helper function `use_r()`. This will create and/or open the file name you provide.

```
use_r("getpcd")
```

You will see the messages below as the command creates and opens the new R file in RStudio.

```
use_r("getpcd")
```

Now, we add the code for the function `getpcd()` as below:

```
#' Get Provisional COVID-19 Death Counts by Sex, Age, and State
#'
#' @param state (default: blank, i.e. all states are returned. Note that this is thge full name
 of the state, not an abbreviation. For example: New Jersey.)
#' @param maxrecs (default: 500, i.e. only the first 500 records are returned. Note: Max=50000)
#'
#' @return data.frame
#' @export
#'
#' @examples
#' getpcd(maxrecs=100)
#' getpcd(state="New%20Jersey")
#'
getpcd <- function(state = "", maxrecs = 200) {
  base_url = "https://data.cdc.gov/resource/9bhg-hcku.json?$offset=0&$order=state"
  full_url = base_url
  if (state != "" && !is.na(state) && !is.null(state)) {
    full_url = paste(base_url,"&state=",state,"",sep="")
  }
  if (maxrecs != "" && !is.na(maxrecs) && !is.null(maxrecs)) {
    limit = as.integer(maxrecs)
    if (limit > 200 && limit <= 50000) {
      full_url = paste(full_url,"&$limit=",limit,sep="")
    }
    else {
      full_url = paste(full_url,"&$limit=",200,sep="")
    }
  }
  else {
    full_url = paste(full_url,"&$limit=",200,sep="")
  }
  print(paste("URL:",full_url))
  x <- jsonlite::fromJSON(full_url)
  data_as_of <-  as.Date(x$data_as_of[1])
  print(paste("Data as of: ",data_as_of))
  x <- subset (x, select = -data_as_of)
  x$start_date <- as.Date(x$start_date)
  x$end_date <- as.Date(x$end_date)
  x$year <- as.factor(x$year)
  x$month <- as.factor(x$month)
  x$group <- as.factor(x$group)
  x$state <- as.factor(x$state)
  x$sex <- as.factor(x$sex)
  x$age_group <- as.factor(x$age_group)
  x$covid_19_deaths <- as.integer(x$covid_19_deaths)
  x$total_deaths <- as.integer(x$total_deaths)
  x$pneumonia_deaths <- as.integer(x$pneumonia_deaths)
  x$pneumonia_and_covid_19_deaths <- as.integer(x$pneumonia_and_covid_19_deaths)
  x$pneumonia_influenza_or_covid <- as.integer(x$pneumonia_influenza_or_covid)
  x$influenza_deaths <- as.integer(x$influenza_deaths)
  (x)
}
```

The function takes 2 parameters and allows the user to specify the state for which the data is desired and how many rows to return. Note that the CDC site has a maximum of 50,000 records to return per request so paging is necessary if the data exceeds this number. As of March 9, 2021 the total number of records for all states was 46,818. A future enhancement to this package would allow the user to specify the offset, for example, so that data can be returned in chunks. In summary, the `getpcd()` function provides the following useful services:

1. Constructs an appropriate REST URL to retrieve the desired data
2. Outputs the constructed REST URL
3. Cleans the CDC data (makes date strings dates, etc.)
4. Removes the redundant `data_as_of` column and outputs that for user review.

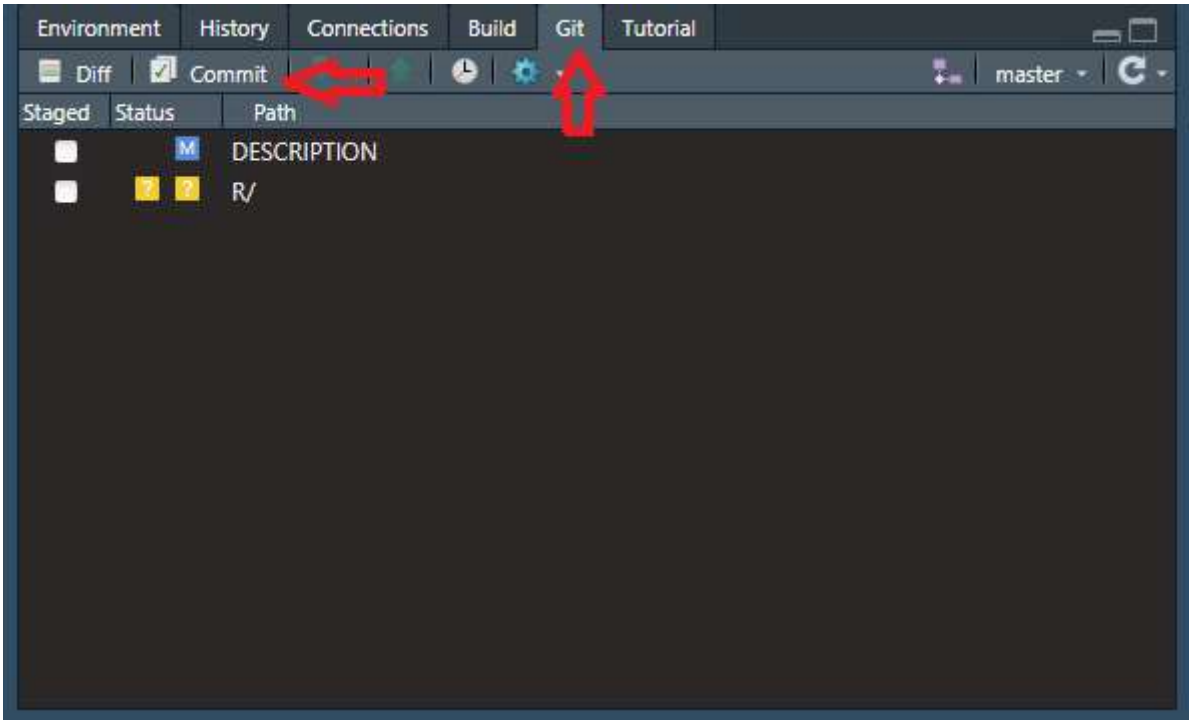
Now, to test this function, it's important that `load_all()` is used to make the function available for experimentation. The call to `load_all()` simulates the process of building, installing and attaching the package and gives an accurate sense of how the package is developing. It also does not pollute the global name spaces (as does the `source()` command, for example).

```
> load_all()
Loading PCD
```

8.0 First Git Commit

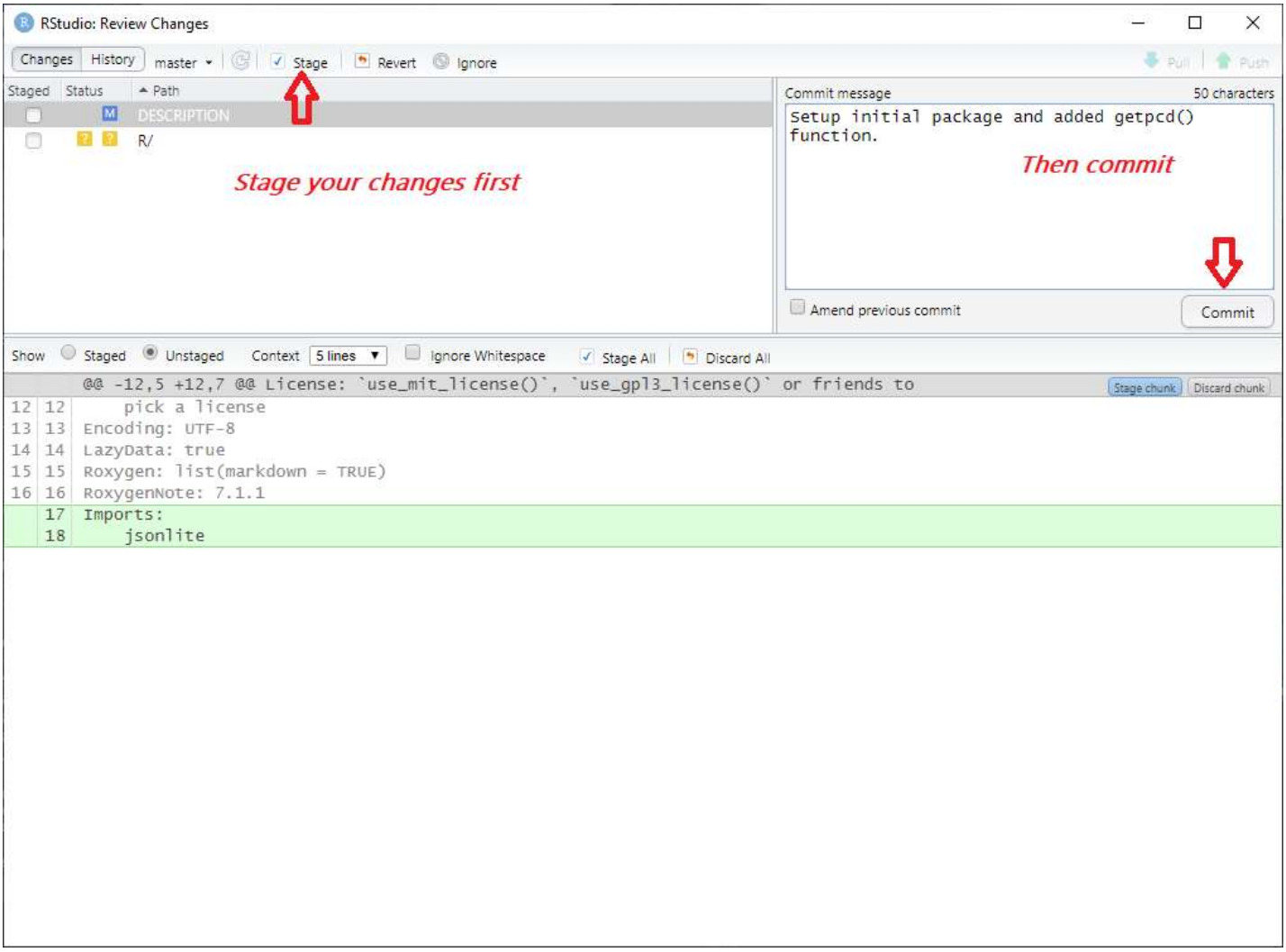
Now that we've created the R function, it's time to do our first commit so our work so far can be saved.

To do this, switch to the Git tab and press the Commit button:



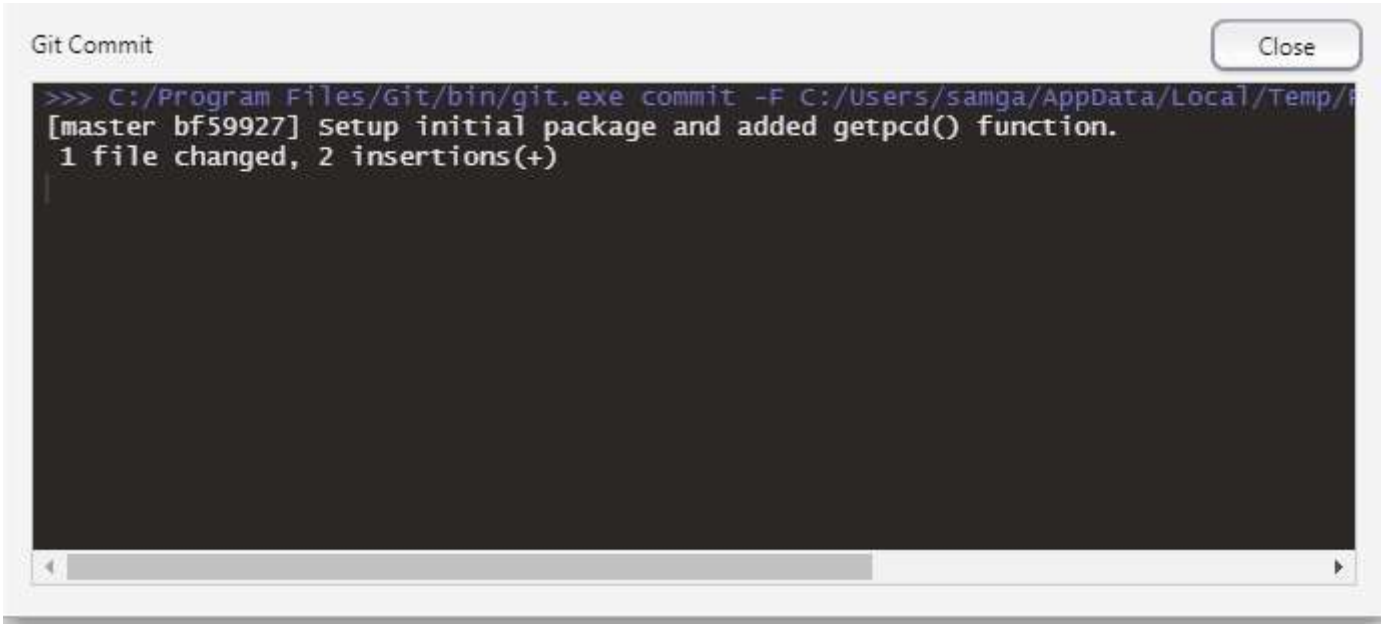
Git Commit

A new window will open showing you the changes you made from your last commit. Click the Stage button first to stage the changes, the the Commit button. You should also enter some helpful text to document the changes you are committing:



Git Commit

You will then see a confirmation window similar to the below:



Git Commit

9.0 Checking the Whole Package

Now that we successfully tested the `getpcd()` function and saved our changes, it's time to run some checks on the package as a whole (all the files, imports, documentation, etc.) to see if any issues exist. The command to do this is check():

```
check()
```

The command runs an extensive series of checks and outputs its findings:

```
Updating PCD documentation
Loading PCD
-- Building ------------------------------------------------------------------ PCD --
Setting env vars:
* CFLAGS    : -Wall -pedantic
* CXXFLAGS  : -Wall -pedantic
* CXX11FLAGS: -Wall -pedantic
-----------------------------------------------------------------------------------
WARNING: Rtools is required to build R packages, but is not currently installed.

Please download and install Rtools 4.0 from https://cran.r-project.org/bin/windows/Rtools/.
√  checking for file 'C:\Users\samga\PCD/DESCRIPTION' (481ms)
-  preparing 'PCD': (347ms)
√  checking DESCRIPTION meta-information ...
-  checking for LF line-endings in source and make files and shell scripts
-  checking for empty or unneeded directories
   Removed empty directory 'PCD/man'
-  building 'PCD_0.0.0.9000.tar.gz'

-- Checking ------------------------------------------------------------------ PCD --
Setting env vars:
* _R_CHECK_CRAN_INCOMING_REMOTE_: FALSE
* _R_CHECK_CRAN_INCOMING_       : FALSE
* _R_CHECK_FORCE_SUGGESTS_      : FALSE
* NOT_CRAN                      : true
-- R CMD check ---------------------------------------------------------------------
-  using log directory 'C:/Users/samga/AppData/Local/Temp/Rtmpmadly6/PCD.Rcheck' (514ms)
-  using R version 4.0.4 (2021-02-15)
-  using platform: x86_64-w64-mingw32 (64-bit)
-  using session charset: ISO8859-1
-  using options '--no-manual --as-cran'
√  checking for file 'PCD/DESCRIPTION' ...
-  this is package 'PCD' version '0.0.0.9000'
-  package encoding: UTF-8
√  checking package namespace information
√  checking package dependencies (1.2s)
√  checking if this is a source package ...
√  checking if there is a namespace
√  checking for .dll and .exe files
√  checking for hidden files and directories ...
√  checking for portable file names ...
√  checking serialization versions
√  checking whether package 'PCD' can be installed (4.8s)
√  checking package directory
√  checking for future file timestamps (3.4s)
W  checking DESCRIPTION meta-information (719ms)
   Non-standard license specification:
     `use_mit_license()`, `use_gpl3_license()` or friends to pick a
     license
   Standardizable: FALSE
√  checking top-level files ...
√  checking for left-over files ...
√  checking index information
```

```
√  checking package subdirectories ...
√  checking R files for non-ASCII characters ...
√  checking R files for syntax errors ...
√  checking whether the package can be loaded (342ms)
√  checking whether the package can be loaded with stated dependencies ...
√  checking whether the package can be unloaded cleanly ...
√  checking whether the namespace can be loaded with stated dependencies ...
√  checking whether the namespace can be unloaded cleanly (465ms)
√  checking loading without being on the library search path (434ms)
√  checking dependencies in R code (342ms)
√  checking S3 generic/method consistency (466ms)
√  checking replacement functions ...
√  checking foreign function calls (357ms)
√  checking R code for possible problems (3.5s)
√  checking for missing documentation entries ...
-  checking examples ... NONE (345ms)
√  checking for non-standard things in the check directory
√  checking for detritus in the temp directory ...

   See
     'C:/Users/samga/AppData/Local/Temp/Rtmpmadly6/PCD.Rcheck/00check.log'
   for details.


-- R CMD check results ---------------------------------------- PCD 0.0.0.9000 ----
Duration: 19.1s

> checking DESCRIPTION meta-information ... WARNING
  Non-standard license specification:
    `use_mit_license()`, `use_gpl3_license()` or friends to pick a
    license
  Standardizable: FALSE

0 errors √ | 1 warning x | 0 notes √
```

Notice that it generated 1 warning pertaining to the package license, which we'll address in the next section.

10.0 Specify Package License

For the `PCD` package, we'll use the MIT license as it's a very open license for open source developers and businesses. Another helper function exists for this purpose: `use_mit_license()` which we use below:

```
use_mit_license("Sam Gabor")
```

This will update the DESCRIPTION file and add a new LICENSE file.

```
√ Setting License field in DESCRIPTION to 'MIT + file LICENSE'
√ Writing 'LICENSE'
√ Writing 'LICENSE.md'
√ Adding '^LICENSE\\.md$' to '.Rbuildignore'
```

11.0 Add a Proper Description

For a package to be useful to others, it should have a good DESCRIPTION file so users can determine if it's helpful in their work. To that end, we edit the DESCRIPTION file for `PCD` with the below text:

```
Package: PCD
Title: Downloads Latest Provisional COVID-19 Death Counts from CDC Using REST API
Version: 0.0.0.9000
Authors@R:
    person(given = "Sam",
           family = "Gabor",
           role = c("aut", "cre"),
           email = "sg662@columbia.edu")
Description: PCD allows users to download the latest Provisional COVID-19 Death
  Counts data from the CDC in a clean dataframe. It also allows the user to only
  download a subset of data of interest, e.g. by reporting US state.
License: MIT + file LICENSE
Encoding: UTF-8
LazyData: true
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.1.1
Imports:
    jsonlite
```

12.0 Add Documentation

Of course, beyond the package description, we still need to document the `getpcd()` function. This requires that we add a special documentation file `man/getpcd.Rd` written in an R-specific markup language. Luckily, we can use the `roxygen2` package to help us generate this file without knowing the details of this markup language.

To get started with `roxygen2`, we open the `getpcd.R` file and position the cursor in the `getpcd() function, then we click *Code->Insert roxygen skeleton* in RStudio. Once that is done, we get a template to edit as in the below:

```
#' Get Provisional COVID-19 Death Counts by Sex, Age, and State
#'
#' @param state (default: blank, i.e. all states are returned. Note that this is thge full name
 of the state, not an abbreviation. For example: New Jersey.)
#' @param maxrecs (default: 500, i.e. only the first 500 records are returned. Note: Max=50000)
#'
#' @return data.frame
#' @export
#'
#' @examples
#' getpcd() - Returns the first 500 rows of the latest data
#' getpcd(maxrecs=50000) - Returns up to 50,000 records for all states
#' getpcd(state="New Jersey") - Returns up to 500 records for New Jersey only
```

To that end, a `document()` helper function exists that generates a template for us. When run, it will produce this output:

```
Updating PCD documentation
Loading PCD
Writing NAMESPACE
Writing getpcd.Rd
```

In addition to creating the `getpcd.Rd` file, `document()` also updated NAMESPACE file with the function that we're exporting:

```
# Generated by roxygen2: do not edit by hand

export(getpcd)
```

This export directive in the NAMESPACE file is what makes the package work for other users by disclosing the available functions.

13.0 Check Again

After making the above changes, it's now time to check our package again for any errors that may have crept in. Sure enough, running `check()` produced the below error:

```
  Running examples in 'PCD-Ex.R' failed
  The error most likely occurred in:

  > base::assign(".ptime", proc.time(), pos = "CheckExEnv")
  > ### Name: getpcd
  > ### Title: Get Provisional COVID-19 Death Counts by Sex, Age, and State
  > ### Aliases: getpcd
  >
  > ### ** Examples
  >
  > getpcd()
  [1] "URL: https://data.cdc.gov/resource/9bhg-hcku.json?$offset=0&$order=state&$limit=500"
  Error: Required package curl not found. Please run: install.packages('curl')
  Execution halted

1 error x | 0 warnings √ | 0 notes √
```

After further investigation (and a check of all files), it was determined that the dependency on the `curl` package must be from the already include `jsonlite` package. Thus, another `use_package("curl")` was required. Note that installing `curl` the package in the RStudio session does not resolve the error as this pertains to the package environment, not the interactive session.

After adding `curl` the package checked out correctly as below:

```
Updating PCD documentation
Loading PCD
Writing NAMESPACE
Writing NAMESPACE
-- Building ------------------------------------------------------------------ PCD --
Setting env vars:
* CFLAGS    : -Wall -pedantic
* CXXFLAGS  : -Wall -pedantic
* CXX11FLAGS: -Wall -pedantic
-----------------------------------------------------------------------------------
WARNING: Rtools is required to build R packages, but is not currently installed.

Please download and install Rtools 4.0 from https://cran.r-project.org/bin/windows/Rtools/.
√  checking for file 'C:\Users\samga\PCD/DESCRIPTION' (477ms)
-  preparing 'PCD': (339ms)
√  checking DESCRIPTION meta-information
-  checking for LF line-endings in source and make files and shell scripts
-  checking for empty or unneeded directories
-  building 'PCD_0.0.0.9000.tar.gz'

-- Checking ------------------------------------------------------------------ PCD --
Setting env vars:
* _R_CHECK_CRAN_INCOMING_REMOTE_: FALSE
* _R_CHECK_CRAN_INCOMING_       : FALSE
* _R_CHECK_FORCE_SUGGESTS_      : FALSE
* NOT_CRAN                      : true
-- R CMD check ---------------------------------------------------------------------
-  using log directory 'C:/Users/samga/AppData/Local/Temp/RtmpQ1qWPa/PCD.Rcheck' (516ms)
-  using R version 4.0.4 (2021-02-15)
-  using platform: x86_64-w64-mingw32 (64-bit)
-  using session charset: ISO8859-1
-  using options '--no-manual --as-cran'
√  checking for file 'PCD/DESCRIPTION' ...
-  this is package 'PCD' version '0.0.0.9000'
-  package encoding: UTF-8
√  checking package namespace information ...
√  checking package dependencies (1.1s)
√  checking if this is a source package ...
√  checking if there is a namespace
√  checking for .dll and .exe files
√  checking for hidden files and directories ...
√  checking for portable file names ...
√  checking serialization versions
√  checking whether package 'PCD' can be installed (4.1s)
√  checking package directory
√  checking for future file timestamps (494ms)
√  checking DESCRIPTION meta-information (605ms)
√  checking top-level files ...
√  checking for left-over files ...
√  checking index information
√  checking package subdirectories ...
√  checking R files for non-ASCII characters ...
√  checking R files for syntax errors ...
```

```
√  checking whether the package can be loaded (341ms)
√  checking whether the package can be loaded with stated dependencies ...
√  checking whether the package can be unloaded cleanly ...
√  checking whether the namespace can be loaded with stated dependencies ...
√  checking whether the namespace can be unloaded cleanly (468ms)
√  checking loading without being on the library search path (415ms)
√  checking dependencies in R code (342ms)
√  checking S3 generic/method consistency (558ms)
√  checking replacement functions (342ms)
√  checking foreign function calls (341ms)
√  checking R code for possible problems (3.5s)
√  checking Rd files (343ms)
√  checking Rd metadata ...
√  checking Rd line widths ...
√  checking Rd cross-references (342ms)
√  checking for missing documentation entries (341ms)
√  checking for code/documentation mismatches (1s)
√  checking Rd \usage sections (806ms)
√  checking Rd contents ...
√  checking for unstated dependencies in examples ...
√  checking examples (1.9s)
√  checking for non-standard things in the check directory
√  checking for detritus in the temp directory


-- R CMD check results ---------------------------------------- PCD 0.0.0.9000 ----
Duration: 20.9s

0 errors √ | 0 warnings √ | 0 notes √
```

14.0 Installing and Using

To simulate end-user installation, we use the `install()` command, which installs `PCD` in our library and produces this output

```
WARNING: Rtools is required to build R packages, but is not currently installed.

Please download and install Rtools 4.0 from https://cran.r-project.org/bin/windows/Rtools/.
√  checking for file 'C:\Users\samga\PCD/DESCRIPTION' (929ms)
-  preparing 'PCD':
√  checking DESCRIPTION meta-information ...
-  checking for LF line-endings in source and make files and shell scripts
-  checking for empty or unneeded directories
-  building 'PCD_0.0.0.9000.tar.gz'

WARNING: Rtools is required to build R packages, but is not currently installed.

Please download and install Rtools 4.0 from https://cran.r-project.org/bin/windows/Rtools/.
Running "C:/PROGRA~1/R/R-4.0.4/bin/x64/Rcmd.exe" INSTALL \
  "C:\Users\samga\AppData\Local\Temp\RtmpQ1qWPa/PCD_0.0.0.9000.tar.gz" \
  --install-tests
* installing to library 'C:/Users/samga/Documents/R/win-library/4.0'
* installing *source* package 'PCD' ...
** using staged installation
** R
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
'PCD'verting help for package     finding HTML links ...
 done
/etpcd                                  html
** building package indices
** testing if installed package can be loaded from temporary location
*** arch - i386
*** arch - x64
** testing if installed package can be loaded from final location
*** arch - i386
*** arch - x64
** testing if installed package keeps a record of temporary installation path
* DONE (PCD)
```

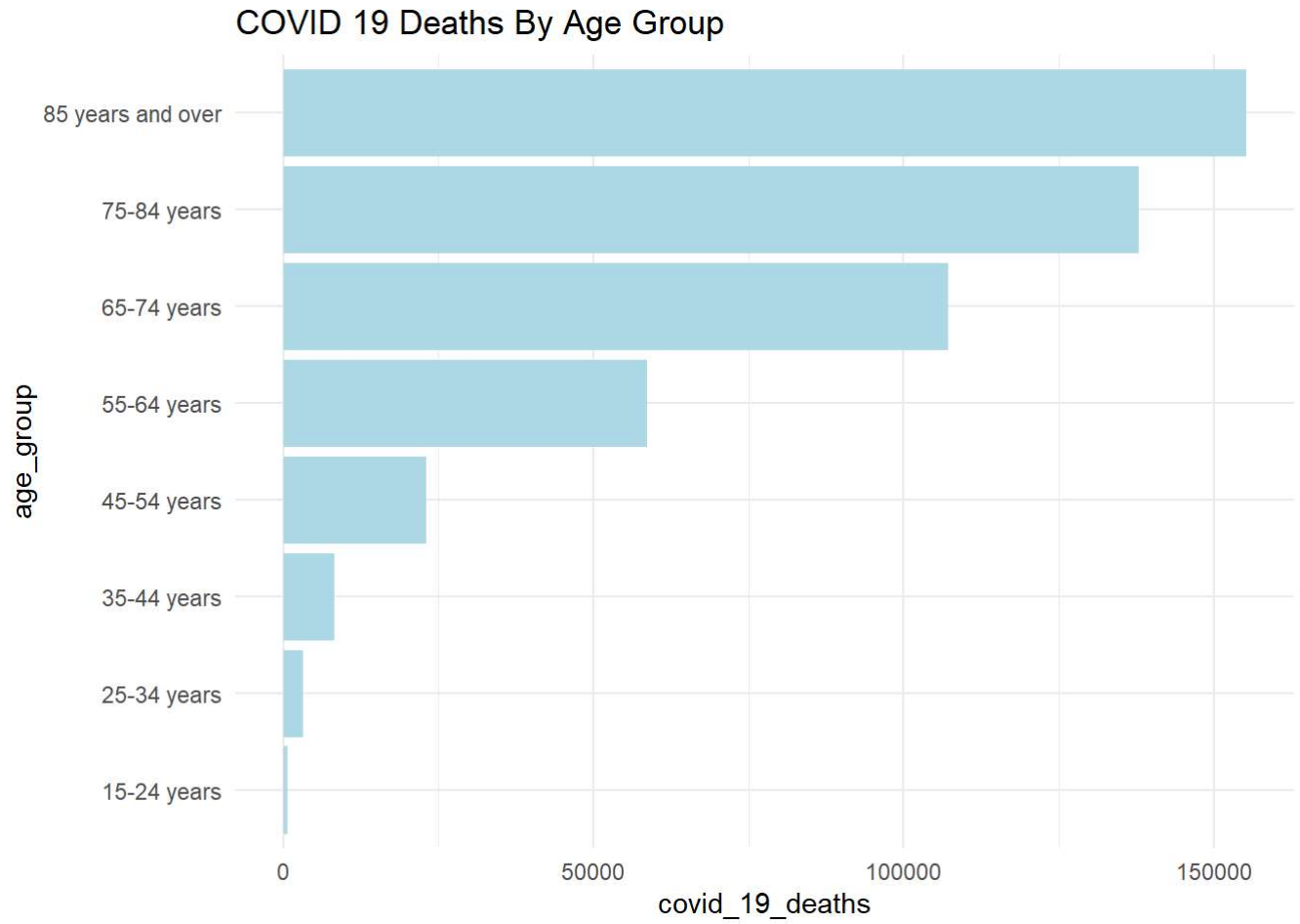
Finally, we can test our library by running a quick R script:

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.6     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## [1] "URL: https://data.cdc.gov/resource/9bhg-hcku.json?$offset=0&$order=state&state=United%20
States&$limit=1000"
## [1] "Data as of:  2021-03-03"
```



In using the `PCD` package, we're able to get the latest CDC data without worrying about low-level details such as downloading files and cleaning the data.

15.0 Unit Testing

We tested `PCD` informally in the previous section, but we can also include formal tests using the `use_test("getpcd")` command. This command creates a template in which we add formal tests of our function.

Here's an example test script that ensures that we return the number of records the user asked for:

```
test_that("getpcd() returns the requested number of records", {
  # If we ask for 300 records, we should get 300 back
  expect_equal(dim(getpcd(maxrecs=300))[1], 300)
})
```

After saving this script to the file `test-getpcd.R` file, we run `load_all()` followed by `test()` and we can review the succesful output as in the below:

```
== Results =========================================================================
Duration: 0.6 s

[ FAIL 1 | WARN 0 | SKIP 0 | PASS 0 ]
> load_all()
Loading PCD
> test()
Loading PCD
Testing PCD
√ |  OK F W S | Context
/ |   0       | getpcd                                                                  [1] "URL: h
ttps://data.cdc.gov/resource/9bhg-hcku.json?$offset=0&$order=state&$limit=300"
[1] "Data as of:  2021-03-03"
√ |   1       | getpcd [0.3 s]

== Results =========================================================================
Duration: 0.3 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ]
```

16.0 Create a GitHub README.md and Publish

Lastly, to ensure that potential users of this package can find and have enough information on the `PCD` package to see if its useful, it's best to publish this to GitHub create an appropriate README.md file.

Using the helper function `use_readme_rmd()` a template is quickly generated. After editing and saving this file, run `build_readme()` to create the required markdown for GitHub.

Finally, run the helper function `use_github()` to commit the changes you made through Git and publish your package to GitHub. The sequence is below:

```
> use_github()
√ Checking that current branch is default branch ('master')
√ Creating GitHub repository 'atsats/PCD'
√ Setting remote 'origin' to 'https://github.com/atsats/PCD.git'
√ Setting URL field in DESCRIPTION to 'https://github.com/atsats/PCD'
√ Setting BugReports field in DESCRIPTION to 'https://github.com/atsats/PCD/issues'
There is 1 uncommitted file:
* 'DESCRIPTION'
Is it ok to commit it?

1: Absolutely
2: Absolutely not
3: No

Selection: 1
√ Adding files
√ Making a commit with message 'Add GitHub links to DESCRIPTION'
√ Pushing 'master' branch to GitHub and setting 'origin/master' as upstream branch
√ Opening URL 'https://github.com/atsats/PCD'
```

Note that if you have any un-comitted changes, `use_github()` will warn you at this point.

As changes are made to the package, Git can push these changes to GitHub.

For the end result, you can visit the page here: https://github.com/atsats/PCD (https://github.com/atsats/PCD)

17.0 Future Directions

The `PCD` package can be enhanced in a number of ways. For example, instead of specify a space as `%20` in the `state` parameter, one can do the encoding directly in `getpcd()`. Another important enhancement would be paging of large data sets and possibly local caching to minimize the hits on the CDC site.

References

I'm indebted to Hadley Wickam's fantastic book *R Packages* (available online at: https://r-pkgs.org/index.html (https://r-pkgs.org/index.html)) for the information used to create this tutorial.