Packages

Wickham: http://r-pkgs.had.co.nz/
Developing Packages with RStudio: https://support.rstudio.com/hc/en-us/articles/200486488-Developing-Packages-with-RStu
Writing an R Package From Scratch: https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch

---

Before you begin, make sure the `devtools`,`roxygen2` packages are installed.

Directory structure

Create a new directory for your package (e.g., `myPackage`). In that directory, you will need to also create a directory for your R functions, called `myPackage/R`. In this directory, each file will contain the code defining a function along with its documentation.

In the same folder, create a text file for the package metadata, called `DESCRIPTION`, that has the following lines:

```
Package: myPackage
Title: An Example Package
Version: 0.1
Authors@R: person("Eric", "Archer", email = "eric.archer@noaa.gov", role = c("aut", "cre"))
Description: This is a test package to demonstrate package creation. This is the metadata DESCRIPTION f
Depends: R (>= 3.1.0)
License: GPL
LazyData: true
```

A shortcut for this process that also creates an associated R project and initializes a repository is `devtools::create(path)`. Alternatively, you can use `package.skeleton()`, which has functionality to create source files and basic documentation based on a list of objects.

Functions and documentation

In your `myPackage/R` folder, create a new script file that will contain a single function. It is often good to name the file the same as the function:

myPackage/R/smrzVector.R

```
smrzVector <- function(x) {
  num.na <- sum(is.na(x))
  mn <- mean(x, na.rm = TRUE)
  md <- median(x, na.rm = TRUE)
  vr <- var(x, na.rm = TRUE)
  c(NAs = num.na, mean = mn, median = md, variance = vr)
}
```

The next step is to document your function using `roxygen` tags:

```
#' @title Summarize A Vector
#' @description Produce standard summary measures for a numeric vector.
#'
#' @param x a vector of numbers
#'
#' @return a vector of summary values
```

```
#' @export

smrzVector <- function(x) {
  num.na <- sum(is.na(x))
  mn <- mean(x, na.rm = TRUE)
  md <- median(x, na.rm = TRUE)
  vr <- var(x, na.rm = TRUE)
  c(NAs = num.na, mean = mn, median = md, variance = vr)
}
```

To parse the tags and create the .Rd documentation file, use the `devtools` function `document()`. A list of .Rd tags can be found in https://cran.r-project.org/doc/manuals/R-exts.html#Writing-R-documentation-files. A list of `roxygen2` documentation tags can be found in `?rd_roclet`.

Add examples to your documentation as a way of providing some unit testing. The examples should be self-contained, not take too long to run, and should be somewhat self explanatory in what they do and the expected result is:

```
#' @title Summarize A Vector
#' @description Produce standard summary measures for a numeric vector.
#'
#' @param x a vector of numbers
#'
#' @return a vector of summary values
#'
#' @example
#' x <- runif(100)
#' smrzVector(x)
#'
#' @export

smrzVector <- function(x) {
  num.na <- sum(is.na(x))
  mn <- mean(x, na.rm = TRUE)
  md <- median(x, na.rm = TRUE)
  vr <- var(x, na.rm = TRUE)
  c(NAs = num.na, mean = mn, median = md, variance = vr)
}
```

Text in the documentation file can be formatted to provide italics, bold, show code, or provide links to other documentation. The latter is achieved with the `\link[pkg]{function}` command. This is often wrapped in `\code{\link[pkg]{function}}`.

Dependencies

Packages are rarely completely standalone and will often have dependencies to other packages. You should never use `library()` or `require()` to load a package in a function that is part of your package. These dependencies get listed in the `DESCRIPTION` file under `Imports`, `Suggests`, or sometimes `Depends`. For a good overview, read Wickham's descriptions of Dependencies and Namespaces. In brief, use `Imports` for packages that your code requires. Use `Suggests` for packages that small, rarely used pieces of your code use. Use `Depends` for packages that absolutely must be installed and loaded for your code to make sense - for example, if your package extends the utility of another package.

Building Packages

https://support.rstudio.com/hc/en-us/articles/200486518-Customizing-Package-Build-Options

`R CMD check`

Performs diagnostic check of package structure and files

`R CMD build`

Builds a package from sources

`R CMD INSTALL`

Installs a package to a library

Submitting a package to CRAN

1) Make sure it passes all checks.
2) Run it through http://win-builder.r-project.org/upload.aspx using R-devel.
3) Submit to https://cran.r-project.org/submit.html