R Packages

A Simple Guide

Part I - Getting Started

Follow the instructions as best you can.

Work with the people around you and don't be afraid to ask them or me any questions.

Don't worry if you don't finish a section before we move on, it's not a race and you can pick it back up another time.

Create Package

1. File -> New Project -> New Package -> "R Package using devtools".
2. Ctrl-Shift-N (new file) -> Ctrl-S (save to ./R/add.r).
3. Add the following to the file: `add <- function(x, y) { x + y }`
4. Ctrl-Shift-L (creates and reloads package).
5. Input `add(1, 2)` to the console.
6. Create an `equals(x, y)` function similar to the above that returns a boolean (`TRUE` or `FALSE`)
   a. Don't forget to reload!
   b. Can you think of any edge-cases where it might not work correctly? Test them.
   c. How might you improve `equals(x, y)`?
   d. Does R have a better way of doing this?

Documentation

Add roxygen2 comments to document the *add* function using the below as a template:

```
#' Return a "hello world" string
#'
#' @return "Hello, world!"
#' @examples
#' hello()
#' print(hello())
#' @export
hello <- function() {
 print("Hello, world!")
}
```

Tips

The function name and the file name should generally be the same.

The output of the last expression in a function is returned.

For further reading about packages and their metadata: http://r-pkgs.had.co.nz/description.html

`@return` *and* `@examples` *are keywords, which you use to tell roxygen2 how to build the documentation.*

@export keyword allows anyone who installs your package (e.g. from Github or CRAN) to use the function. Otherwise it is kept internal to the

package and is not "exported". See http://r-pkgs.had.co.nz/namespace.html for more details.

1. Write the documentation.
2. Build the documentation with either Ctrl-Shift-D or input `devtools::document()` to the console.
3. Input `help(add)` to the console to view the documentation. Compare the roxygen comments to the documentation. Pretty neat huh?
4. You will need to use @param keyword to describe the two input parameters, x and y. See http://r-pkgs.had.co.nz/man.html for more details.
5. Build the documentation again.

Part II - Testing With *testthat*

Hello World

Input the following into the console: `devtools::use_testthat()`
This will create the correct files and folders in your package to allow the use of *testthat* for testing.

Create the following two files then hit Ctrl-Shift-T to run your test and see if it passes. The test output will be displayed in the "Build" pane in RStudio.

```
./R/hello.R
hello <- function() {
 print("Hello, world!")
}
```

```
./tests/testthat/test_hello.R
context("Hello")
test_that("hello prints", {
  expect_that(expected_results <- hello(),
prints_text("Hello, world!"))
  expect_is(expected_results, "numeric")
})
```

This folder is where tests using testthat should go.

1. There is an error in one of the tests, can you fix it?
2. Do you understand what's going on in each line of code above?
3. Why did I evaluate hello() inside the expect_that() function call? (see hint)

Try evaluating hello() in each test case instead. What happened?

Generate Test Data

Create a new file and function as below:

```
generate_test_data <- function() {
  test_data <- data.frame("person"=c("Bob", "Alice",
                                      "Stephen", "Mary"),
                          "score"=seq(0, 9, 2.5),
                          "age"=round(rnorm(4, 15, 1)),
                          stringsAsFactors = FALSE)
  test_data
}
```

Check the test data frame for the following properties using *expect_is* and *expect_equal* functions from *testthat:*

1. It should be a data frame.
2. *score* should have a mean of 3.75.
3. It should have no missing values (see hint ->).

What does na.rm do to a dataframe that does not have any missing values?

Slope

Create a new test called *test_slope* for a new function called *slope* (that you haven't written yet!). The test should expect the correct value for a slope of the line, given coordinates (x1, y1) and (x2,y2). I.e., slope(x1, y1, x2, y2)

Formula -> slope = y2 - y1 / x2 - x1

Will the test fail?

Write the function *slope()* that passes this test.

1. Can you think of any *edge cases* that you could write tests for? Create these tests!
2. Does your function pass these new tests?

Edge cases are often used in unit testing. For example, a function that divides two numbers might be tested using very large, very small or negative numbers. The assumption being that the function should behave well for values in between these edge cases.

Testing Models

This next section is a bit more advanced so if you get a bit lost, don't worry just try to follow the steps as closely as you can.

We can even use tests to make sure any simulated or test data is being modelled correctly. This isn't really a unit test but you can imagine how this might be useful in validating model performance in a build pipeline, e.g. as part of a suite of integration tests.

1. Create a new test called test_model.R and give it a new context.
2. Use the following model for testing:
   ```
   lm(score ~ age, generate_test_data())
   ```
3. Check that the coefficients have the following properties:
   a. Length 2.
   b. No NA values.
4. Check that all the residuals are less than 5.
5. The mean of the residuals is less than 2.

Store the model results in a new variable within the test.
You access objects within the model object like you do with data frames, e.g. model$call*.*