Computational Skills for Biostatistics I: Lecture 6

Amy Willis, Biostatistics, UW

08 May, 2019

Misc

- Please make sure you are effectively using `git` via the command line to submit your homework
  - Many companies, especially tech companies, manage their code bases using `git`, so make sure you actually learn how to use it!
  - Don't think that quantitative analysts are exempt!

Feedback

Thank you everyone for their helpful feedback

- We will do more exercises in pairs
- Beginner vs advanced track: wonderful idea! We're going to do this today

Today's agenda

1. One of the best R package developers I know is going to teach you about writing R packages (and making them available via git!)
2. "Advanced track"
   - Search path
   - Environments
   - Namespaces
   - Effective package development workflow

R packages

Why write an R package?

- The best way to contribute to science
- Allow others to build on your advances
- Promote yourself and your work
- Pass this class

$$\text{package} = \text{code} + \text{other stuff}$$

Bryan is going to teach you about the other stuff... since you already know how to code!

R Packages

Guest lecture by Bryan Martin (PhD Candidate, UW Statistics)

- `corncob`
- `DivNet`

R Packages by Bryan Martin

Storing objects in R

We know broadly how to create, store and overwrite objects in R

```
x <- 5
x <- 7
x
```

```
## [1] 7
```

What happens with clashes?

What does this do?

```
ggplot  <- function(...) "Ha! Overwritten!"
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     ggplot
```

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Width))
```

```
## [1] "Ha! Overwritten!"
```

Why?

Environments

R has an *ordered* search path

```
search()
```

```
##  [1] ".GlobalEnv"         "package:ggplot2"   "package:stats"
##  [4] "package:graphics"   "package:grDevices" "package:utils"
##  [7] "package:datasets"   "package:methods"   "Autoloads"
## [10] "package:base"
```

Where does R search? In what order?

```
search()
```

```
##  [1] ".GlobalEnv"        "package:ggplot2"   "package:stats"
##  [4] "package:graphics"  "package:grDevices" "package:utils"
##  [7] "package:datasets"  "package:methods"   "Autoloads"
## [10] "package:base"
```

- When looking for an object, R looks in the first position in `search()`, if it doesn't find it, it moves on to the second, etc.

Search path

- Global Environment: default storage
  - First place R looks for objects
- Then moves through packages. . . in the order specified!

Most recently we had loaded ggplot2, so it was first.

Packages are loaded into the second position of the search path

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Now we know how to interpret some warnings, too!

Packages are loaded into the second position of the search path

```
search()
```

```
##  [1] ".GlobalEnv"        "package:dplyr"     "package:ggplot2"
##  [4] "package:stats"     "package:graphics"  "package:grDevices"
##  [7] "package:utils"     "package:datasets"  "package:methods"
## [10] "Autoloads"         "package:base"
```

Activity: 1 minute

Explain to the person next to you why we got the result we did here:

```
ggplot  <- function(...) "Ha! Overwritten!"
library(ggplot2)
ggplot(iris, aes(x = Sepal.Length, y = Petal.Width))
```

```
## [1] "Ha! Overwritten!"
```

Namespaces

All packages have a `NAMESPACE` file: a collection of objects to be exported and imported

- To avoid overwriting users' variables
- To avoid ambiguity in function calls
- To ensure the package has everything it needs to run
- To encourage modular code

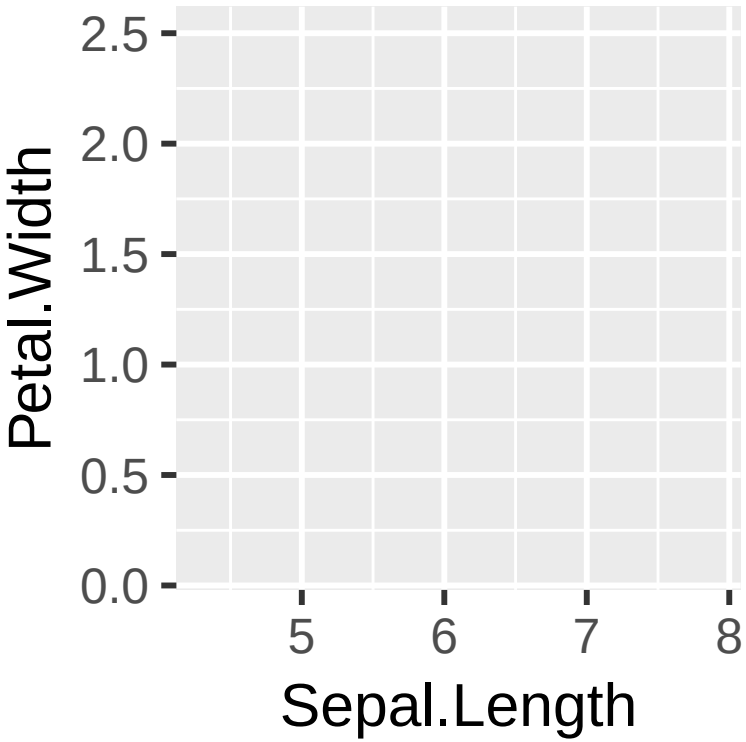
What exactly is loaded?

- `library(pkg)` adds the functions in the NAMESPACE of pkg to the search path
- Your `NAMESPACE` is auto generated using roxygen *tags*
  - never directly modify your `NAMESPACE` file
  - We will come back to roxygen comments/tags later in the lecture

The :: operation

The operator `x::y` means "the function `y` in the namespace of package `x`"

```
ggplot  <- function(...) "Ha! Overwritten!"
ggplot2::ggplot(iris, aes(x = Sepal.Length,
                          y = Petal.Width))
```



Internal and external functions

As a package developer, you need to decide what the user has access to and what is for internal use.

You can access non-exported functions using `:::`

```
dplyr::abort_case_when_formula      # throws error
dplyr:::abort_case_when_formula
```

To *not export* a function to the NAMESPACE of a package, best practice is to just *not add* an `@export` roxygen flag

Seeing what's available

- See where R looks for things using `search()`
- See what is available in the *n*th position of the search path using `ls(..., pos = n)`

You can attach data frames

You can also attach data frames to the search path...

```
attach(women)
search()
```

```
##  [1] ".GlobalEnv"        "women"             "package:dp
##  [4] "package:ggplot2"   "package:stats"     "package:gr
##  [7] "package:grDevices" "package:utils"     "package:da
## [10] "package:methods"   "Autoloads"         "package:ba
```

But don't!

```
height <- height*2.54
height[1:5]
```

```
## [1] 147.32 149.86 152.40 154.94 157.48
```

```
women$height[1:5]
```

```
## [1] 58 59 60 61 62
```

but don't do this!

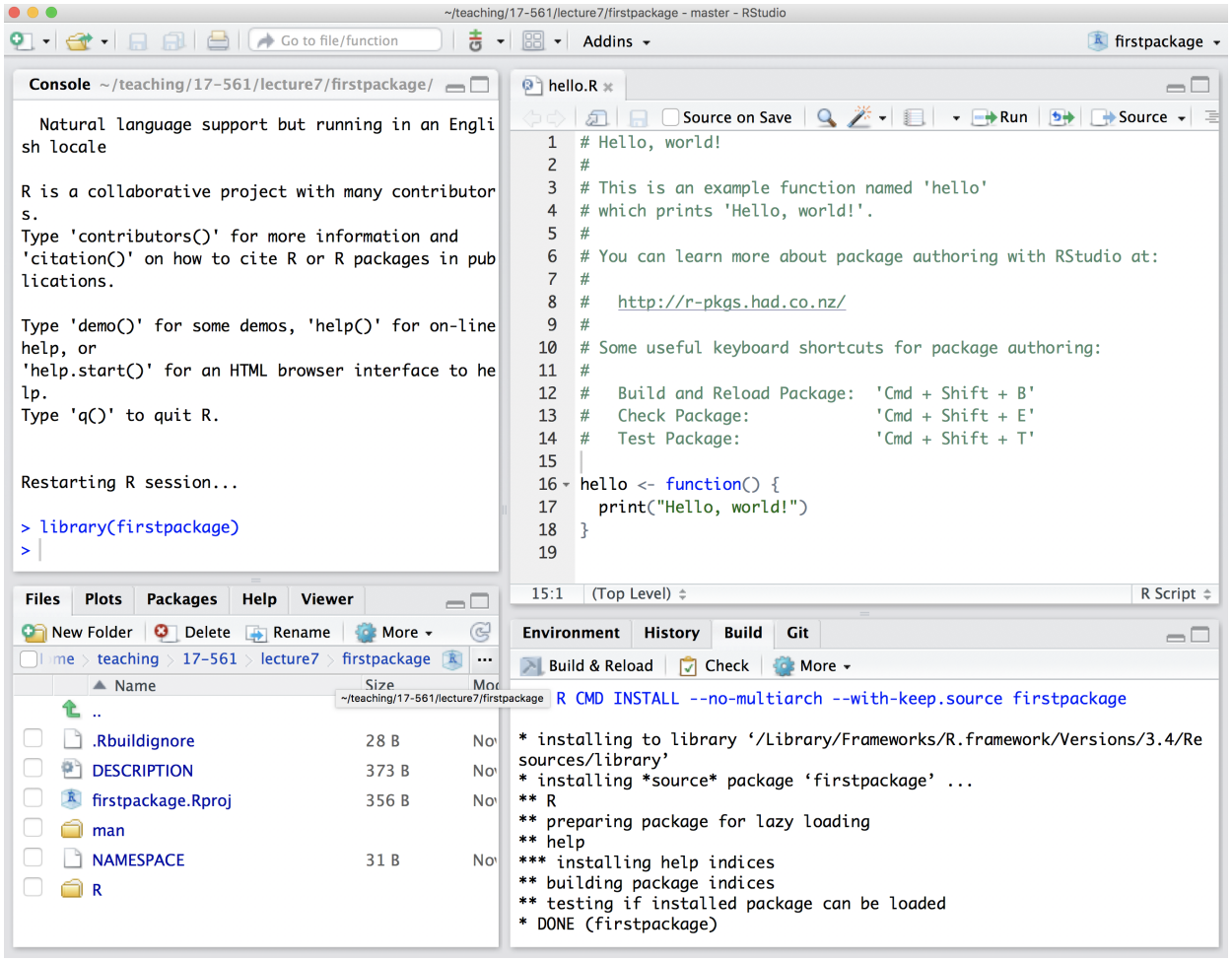
Attach

Moral of the story: Don't attach data frames, but it's fine to attach packages via `library(...)`

Starting an R package

Ideally, create packages from scratch as soon as you begin on a project

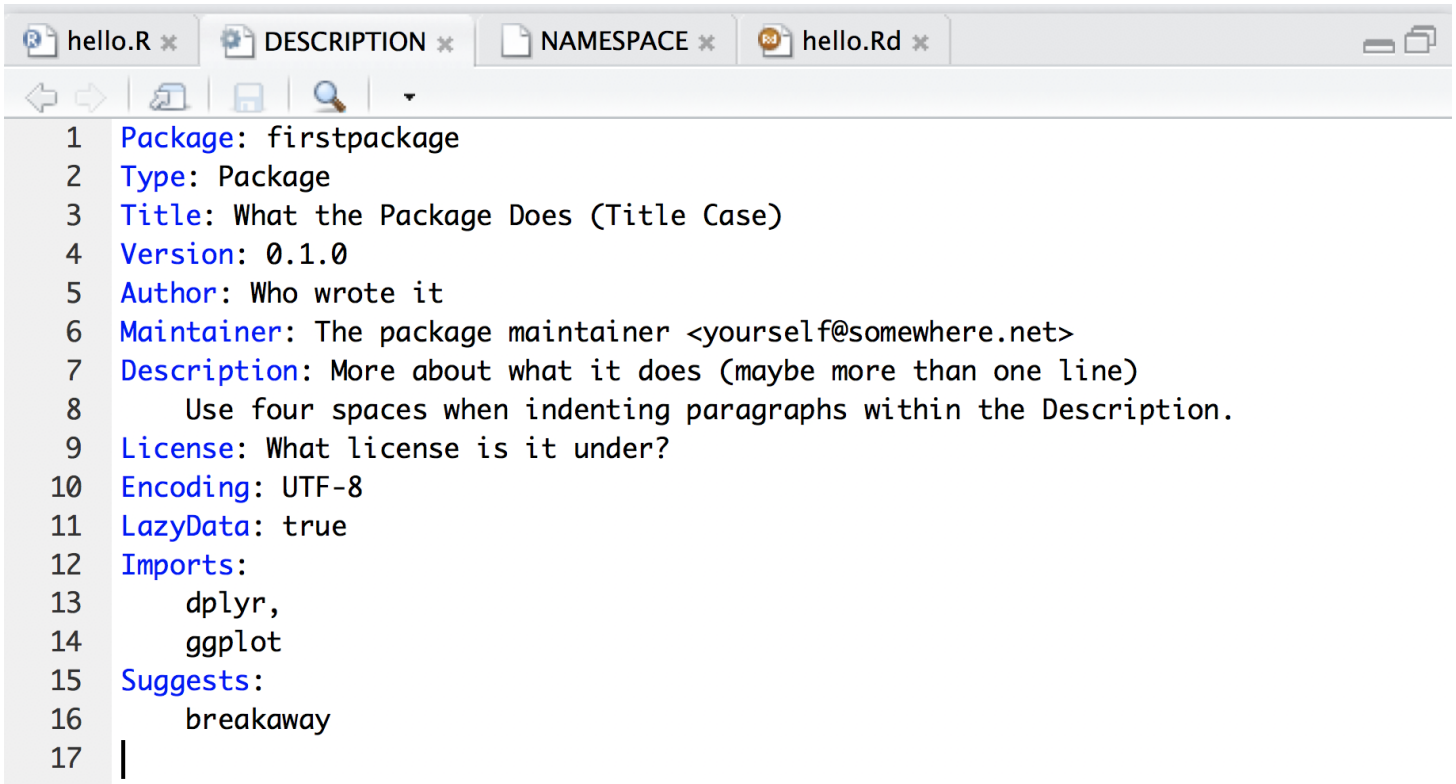
- RStudio->File->New project->New Directory->R Package



Starting an R package: DESCRIPTION

Short-term: Keeps track of imports (dependencies)

Long-term: Help others find your package



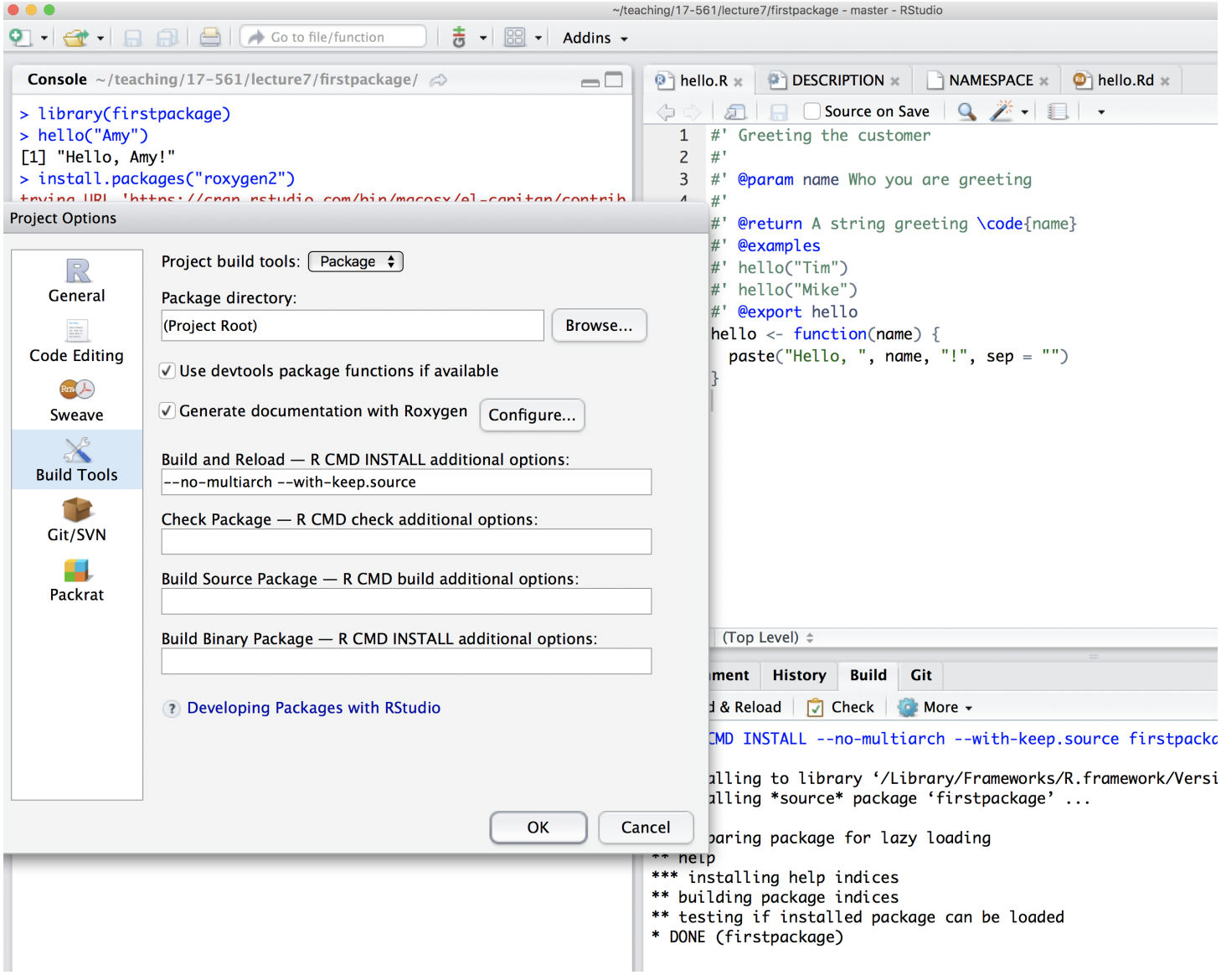
Writing R packages: documentation

`roxygen` documentation makes everything incredibly easy

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

Writing R packages: documentation

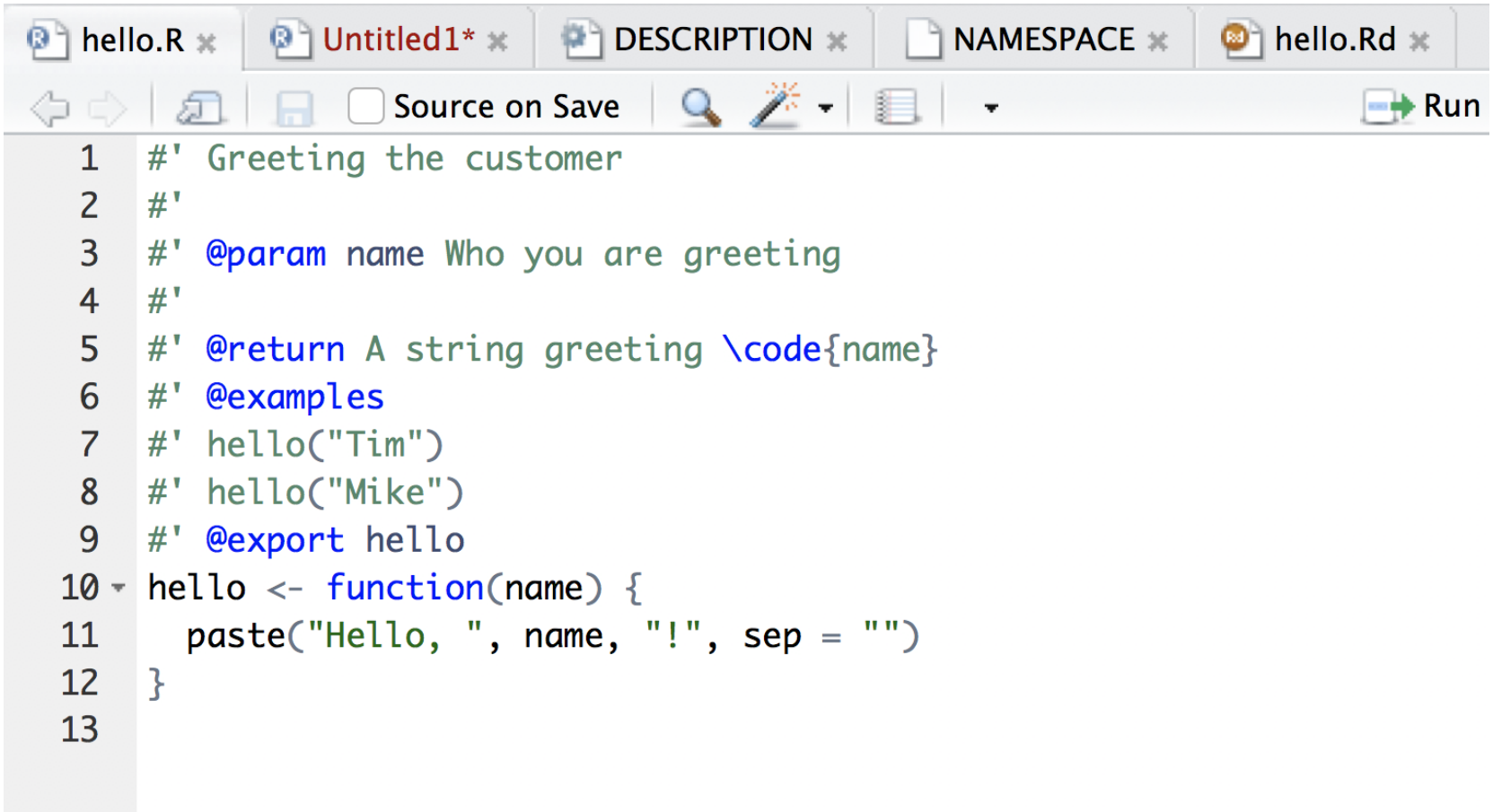
Under the `Build` tab, under Build tools, check `Generate documentation with Roxygen`



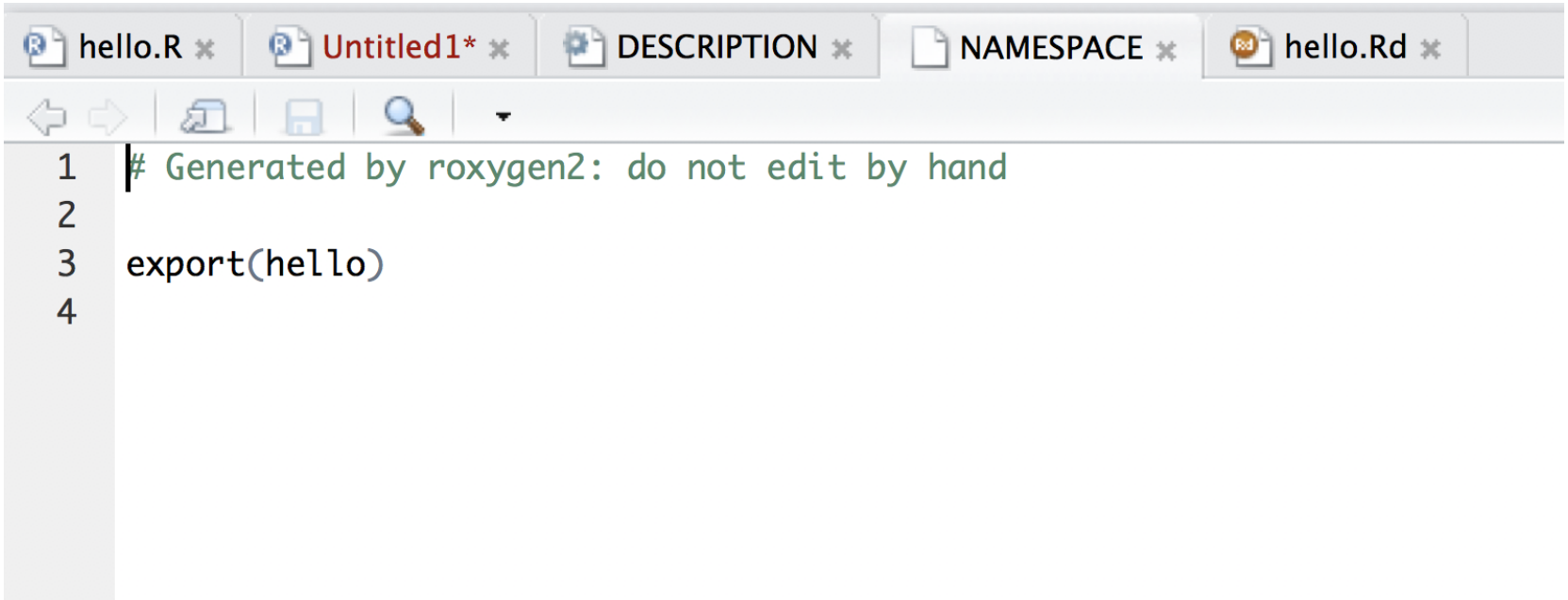
Writing R packages: documentation

1. Delete the default `NAMESPACE` file and the `man` folder
2. Write `roxygen` comments in your `.R` file
   - `roxygen` comments start with `#'`
3. `cmd/ctrl + shift + b` builds the package
4. `cmd/ctrl + shift + d` autogenerates documentation based on your roxygen comments

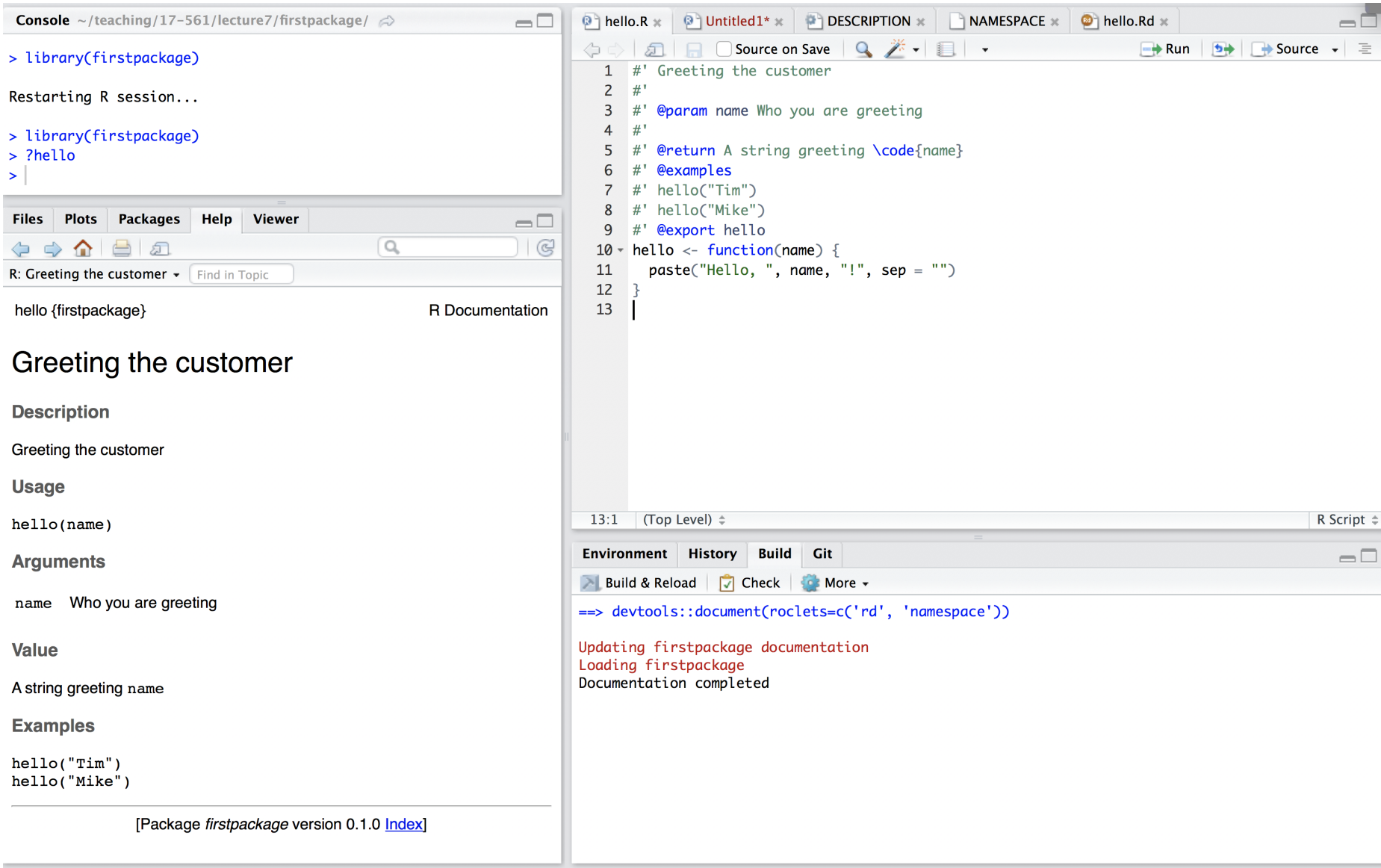
Writing R packages: documentation



Writing R packages: documentation



Writing R packages: documentation



Writing R packages: tags

```
#' \item{seest}{ The standard error in the diversity estimate.  } \item{full}{
#' The chosen nonlinear model for frequency ratios.  } \item{ci}{ An asymmetric
#' 95\% confidence interval for diversity.  }
#' @note \samp{breakaway} presents an estimator of species richness that is
#' well-suited to the high-diversity/microbial setting. However, many microbial
#' datasets display more diversity than the Kemp-type models can permit. In
#' this case, the log-transformed WLRM diversity estimator of Rocchetti et. al.
#' (2011) is returned. The authors' experience suggests that some datasets that
#' require the log-transformed WLRM contain ``false'' diversity, that is,
#' diversity attributable to sequencing errors (via an inflated singleton
#' count). The authors encourage judicious use of diversity estimators when the
#' dataset may contain these errors, and recommend the use of
#' \code{\link{breakaway_nof1}} as an exploratory tool in this case.
#' @author Amy Willis
#' @seealso \code{\link{breakaway_nof1}}; \code{\link{apples}}
#' @references Willis, A. and Bunge, J. (2015). Estimating diversity via
#' frequency ratios. \emph{Biometrics}, \bold{71}(4), 1042--1049.
#'
#' Rocchetti, I., Bunge, J. and Bohning, D. (2011). Population size estimation
#' based upon ratios of recapture probabilities. \emph{Annals of Applied
#' Statistics}, \bold{5}.
#' @keywords diversity microbial models nonlinear
#' @examples
#'
#' breakaway(apples)
#' breakaway(apples, plot = FALSE, output = FALSE, answers = TRUE)
#'
#'
#' @export breakaway
breakaway <- function(my_data, output=TRUE, plot=FALSE, answers=FALSE, force=FALSE, useAll=

  ## read in data
  if( !(is.matrix(my_data) || is.data.frame(my_data))) {
```

Writing R packages: documentation

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

export(betta)
export(betta_pic)
export(betta_points)
export(betta_random)
export(breakaway)
export(breakaway_clean)
export(breakaway_nof1)
export(build_frequency_count_tables)
export(chao1)
export(chao1_bc)
export(chao_bunge)
export(estimate_alpha_better)
export(frequency_count_or_proportion_or_column)
export(gini)
export(hill)
export(hill_better)
export(hill_pic)
export(inverse_simpson)
export(inverse_simpson_better)
export(make_frequency_count_table)
export(objective_bayes_geometric)
export(objective_bayes_mixedgeo)
export(objective_bayes_negbin)
export(objective_bayes_poisson)
export(plot_alpha)
export(proportions_instead)
export(resample_estimate)
export(rnbinomtable)
export(rztnbinomtable)
export(sample_size_estimate)
export(sample_size_figure)
export(shannon)
export(shannon_better)
export(simpson)
export(subsample_otu)
export(to_proportions)
export(wlrm_transformed)
export(wlrm_untransformed)
import(reshape2)
importFrom(MASS,mvrnorm)
importFrom(ggplot2,element_text)
importFrom(ggplot2,facet_wrap)
importFrom(ggplot2,geom_errorbar)
importFrom(plyr,is.discrete)
importFrom(stats,acf)
```

Building packages

Core "other stuff" for a package

- Code: written by you, goes into `R` folder (no subdirectories allowed)
- `DESCRIPTION`: written by you
- Documentation: autogenerated by `roxygen` comments, goes into `man` folder
- `NAMESPACE`: autogenerated by `roxygen` comments

documentation: vignettes

hello.R | breakaway.Rmd | DESCRIPTION | NAMESPACE | hello.Rd

Knit | Insert | Run

```
---
title: "Getting started with breakaway"
author: "Amy Willis"
date: "`r Sys.Date()`"
output: rmarkdown::html_vignette
vignette: >
  %\VignetteIndexEntry{getting-started}
  %\VignetteEngine{knitr::rmarkdown}
  %\VignetteEncoding{UTF-8}
---

`breakaway` is a package for species richness estimation and modelling. As the pac
grown and users have requested more functionality, it contains some basic generali
the statistical philosophy that underpins `breakaway` to the general alpha diversi
Because of the flexibility of the modelling strategies, most users of breakaway ar
microbial ecologists with very large OTU tables, however, nothing should exclude a
macroecologist from using the same tools. If you have a macroecology dataset and w
this package, I would love to hear from you so please feel free to contact me (ema
Github's Issues/Projects infrastructure).

## Vignette Info

This vignette will lead you through

- loading in your data and covariate information
- creating frequency tables
- using `breakaway` to estimate species richness
- using the objective bayes methods to estimate species richness
- estimating various alpha diversity indices and providing a very basic resample-b
method for ascribing variability to them
- using `betta` to model changes in alpha diversity with covariates

If there is something that you would like explained please feel free to request it

## Loading the package and data

Download the latest version of the package from github.

```{r}
### Run the first two lines at home! ####
# install.packages("devtools")
# devtools::install_github("adw96/breakaway")
library(breakaway)
data(toy_otu_table)
data(toy_metadata)
```
```

documentation: vignettes

277 lines (199 sloc) | 13.3 KB

Raw | Blame | History

title	author	date	output	vignette
Getting started with breakaway	Amy Willis	`r Sys.Date()`	rmarkdown::html_vignette	%\VignetteIndexEntry{getting-started} %\VignetteEngine{knitr::rmarkdown} %\VignetteEncoding{UTF-8}

`breakaway` is a package for species richness estimation and modelling. As the package has grown and users have requested more functionality, it contains some basic generalisations of the statistical philosophy that underpins `breakaway` to the general alpha diversity case. Because of the flexibility of the modelling strategies, most users of breakaway are microbial ecologists with very large OTU tables, however, nothing should exclude a macroecologist from using the same tools. If you have a macroecology dataset and want to use this package, I would love to hear from you so please feel free to contact me (email or via Github's Issues/Projects infrastructure).

Vignette Info

This vignette will lead you through

- loading in your data and covariate information
- creating frequency tables
- using `breakaway` to estimate species richness
- using the objective bayes methods to estimate species richness
- estimating various alpha diversity indices and providing a very basic resample-based method for ascribing variability to them
- using `betta` to model changes in alpha diversity with covariates

If there is something that you would like explained please feel free to request it!

Loading the package and data

Download the latest version of the package from github.

```
### Run the first two lines at home! ####
# install.packages("devtools")
# devtools::install_github("adw96/breakaway")
library(breakaway)
data(toy_otu_table)
data(toy_metadata)

## For historical reasons we going to call them:
otu_data <- toy_otu_table
meta_data <- toy_metadata
```

Very important

The right way to debug/develop a function is to write a `testthat` test, and use `devtools::load_all()` + `testthat::test()`

- `devtools::load_all()` is a fast alternative to building and loading
  - Fast because it only loads what has changed

Very important

You should not have scripts to help you debug or develop a package. Everything should be a function (exported or not) in your `R/` folder, and you should load and test the package outlined above.

Fun fact: I did this incorrectly for years... and wasted so much time trying to fix "Error: object not found"

Very important

The wrong way to debug/develop a function is by loading it into the Global Environment and testing it there.

Don't think that `rm(list=ls(all=TRUE))` saves you

- It removes everything in the Global Environment... but not packages!
- Loading functions into the global environment is usually accompanied by loading variables into the global environment, and packages can't access objects that are not passed to it via arguments

Other things to keep in mind

1. `testthat` is a Hadley Wickham package to write unit tests
   - It's so easy to write tests!
   - Every error/bug should become a test
   - Manage your *TODOs* with tests
2. Datasets take a special format
   - Don't release data in the same way as functions
3. `pkgdown` easily turns your R package documentation into a website: `build_site()`

Coming up

- Homework 6: due next *Wednesday*
  - No homework 5
  - Write an R package to do something useful!