# ESM 244 Discussion Week 5: Making an R package

Note: this discussion document is based largely on a handout by Julien Brun (NCEAS) and Allison Horst for the UCSB Data Science Capstone course.

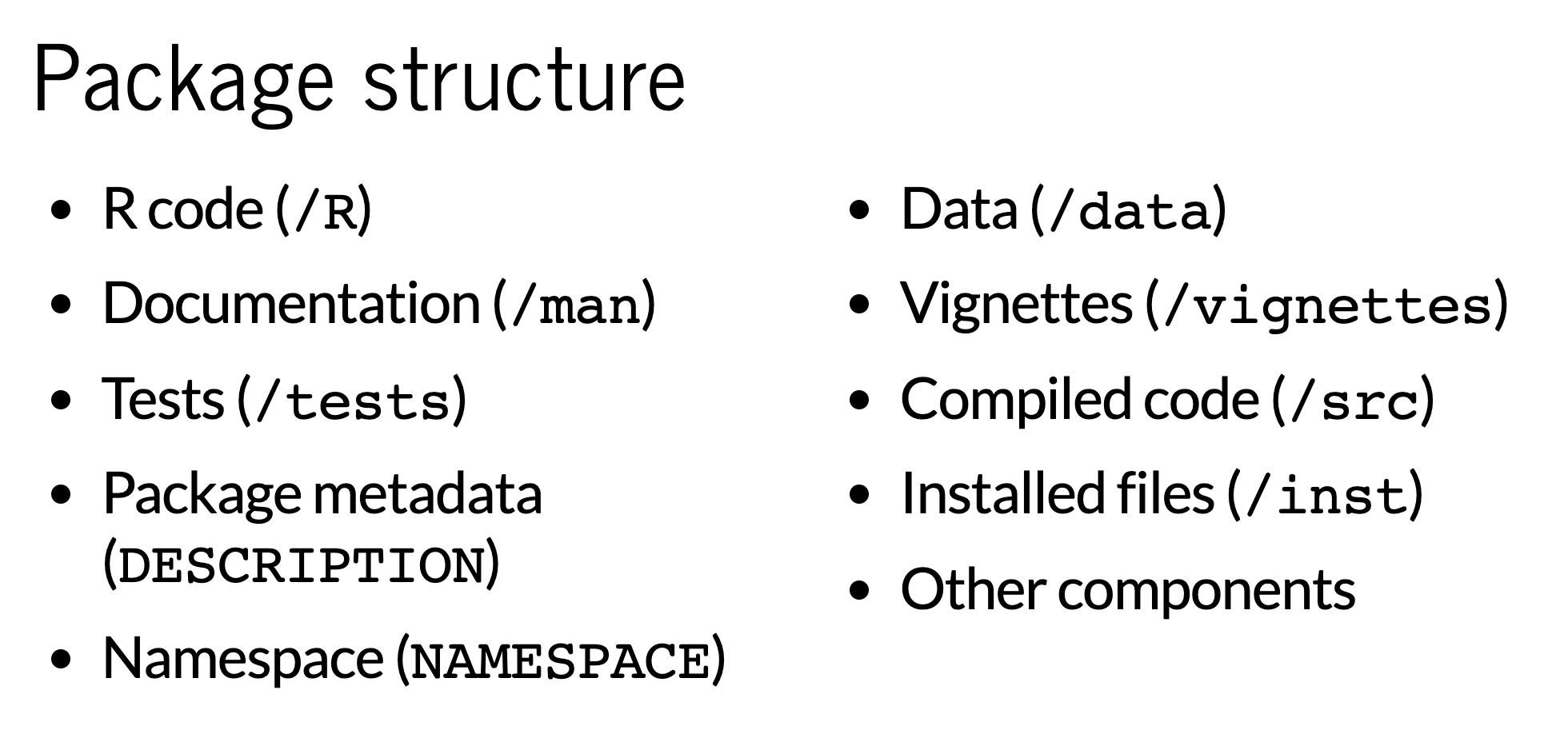
# **Overview: R packages, why & what?**

## Why?

* Package your work and share it!
  + Organize code **and** data
  + Reuse and share your work more easily
* Test code
* Share code with others

## What?

Mostly organizing your work in a file structure plus a few special files that help to bundle everything together.



Let’s check out this package: <https://github.com/isteves/r-pkg-intro>

## 

## 

## **Let’s make one of our own!**

2 helpful packages when developing R packages (we’ll skip a lot of usethis functionality today, but it’s awesome):

* devtools
* usethis

## Creating a package

1. In RStudio, create a new R Project that is an R Package (New Project > New directory > R Package)

The **only** thing you need to add here is the package name. You can name it whatever you want (I recommend a combination of your favorite food and an animal) as long as there are no spaces or special characters. Here are some ideas, but you are encouraged to be creative:

BurritoWolf

lasagnashark

oatmealhawk

1. Check out the existing infrastructure for your R package, which currently contains a single function hello(), which prints “Hello, world!” Check out the R/hello.R file to see where that function is created.   
     
   A. Now, in the **Build** tab, click **Install and Restart**. You should see that the package is automatically attached in the Console.   
     
   B. Run the hello() function in the Console, to see that it works.
2. Make your own function in the package!
3. Create a new **R script**
4. Within the script, **write a function** that is whatever you want (remember: write the right-hand-side of the equation, select it, Cmd + Alt + X to make it into a function!)
5. **Save the R script in to the R/ folder** (with the function name as the filename, for simplicity)
6. Put your cursor anywhere in the function code within your R script. In the top menu of RStudio, select **Code > Insert Roxygen skeleton**. The information added is important - it specifies the params (arguments of the function) and more that will appear in the documentation, which we’ll create next.   
     
   **Save** the .R file, which now contains your function and the Roxygen information.
7. Document the function by running **devtools::document()** in the Console. This will create a new .Rd file in the man/ folder, containing important documentation information about your function.
8. **Build > Install and Restart**
9. In the Console, run **?function\_name**, replacing function\_name with your new function instead. It will bring up the documentation, and let you know your function exists! Now go ahead and try to \*use\* your function by running it in the console (specifying the arguments and assigning values).
10. Make your own ggplot theme!
11. In your package project, add a new R script.
12. Copy and paste this code into your R script. Then **update the colors** (in quotations) to colors that R will recognize ([here’s a list](https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatsheet.pdf), see Pg 2) and **change the function name to YOURNAME\_theme** (e.g. allison\_theme).

my\_theme <- function() {

theme(

panel.background = element\_rect(fill = "yellow"),

panel.grid.major.x = element\_line(colour = "purple", linetype = 3, size = 0.5),

panel.grid.minor.x = element\_blank(),

panel.grid.major.y = element\_line(colour = "cyan4", linetype = 3, size = 0.5),

axis.text = element\_text(colour = "red"),

axis.title = element\_text(colour = "orange")

)

}

1. **Save the R script in to the R/ folder** (with the function name as the filename, for simplicity)
2. Put your cursor anywhere in the function code within your R script. In the top menu of RStudio, select **Code > Insert Roxygen skeleton**. The information added is important - it specifies the params (arguments of the function) and more that will appear in the documentation, which we’ll create next.   
     
   **Save** the .R file, which now contains your function and the Roxygen information.
3. Document the function by running **devtools::document()** in the Console. This will create a new .Rd file in the man/ folder, containing important documentation information about your function.
4. **Build > Install and Restart**
5. In the Console, **run ?function\_name**, replacing function\_name there instead. It will bring up the documentation, and let you know your function exists! Now go ahead and try to \*use\* your function by making a graph in the Console.   
     
   library(tidyverse)

ggplot(data = msleep, aes(x = sleep\_total, y = sleep\_rem)) + geom\_point() + **THEME\_NAME()**

1. Make it a repo that you can share!

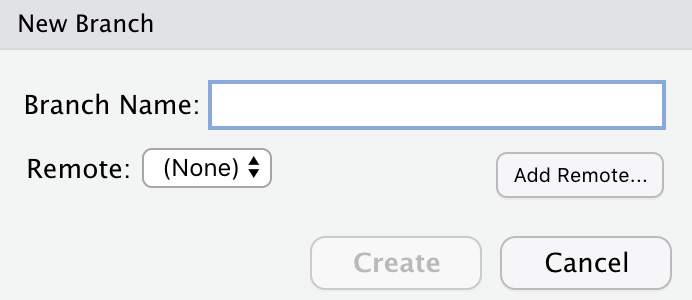
Cool, so now you’ve added your own functions to this package. How can we make it so other people can access it? Let’s add it to a GitHub repo (this also introduces how you can add version control to an *existing* R Project). There are a number of different ways to do this. We’ll use one, but see more options in Jenny Bryan’s amazing book [Happy git with R](https://happygitwithr.com/). This method is from Ch. 17.

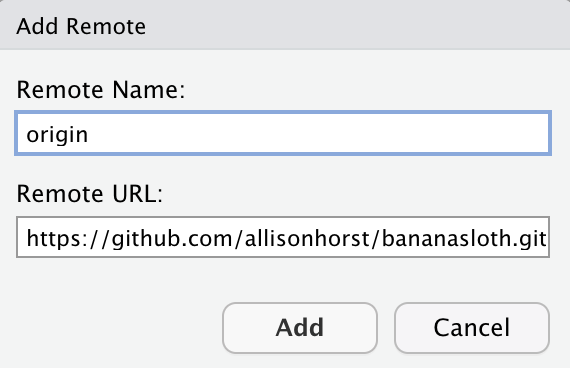
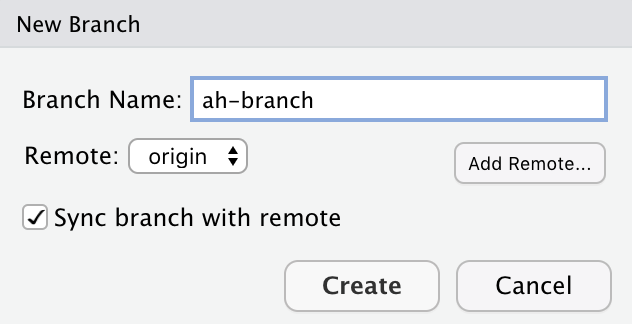
1. In RStudio, go to **Tools > Project Options > Git/SVN**. Update the Version Control System to ‘Git’ in the dropdown menu. Choose **YES** when it asks you if you want to initialize a git repository.

*Note: another great (my preferred…) way to do this is with the usethis package by running usethis::use\_git() in the Console.*

1. R should restart (yes), and when it does you should see the **git** tab appear in the IDE. **Stage and commit** all updated files (you can’t push yet because you haven’t created & connected it to a GitHub repo).
2. Go back to GitHub, make a new repo that **has the same name as your project, WITHOUT A README.** Copy the https URL for the repo.
3. Now connect your local repo to the remote repo on GitHub:

* Back in RStudio, **click on the ‘branches’ button in the git tab** (purple rectangles branching off a white diamond), which should bring up this window:



* **Click on ‘Add Remote’**, which should take you to a new window. From Happy git with R: “Paste the URL here and pick a remote name, almost certainly **origin**. Now “Add”.”  
    
  
* You should be back in the “New Branch” dialog. Enter a branch name that you want to work in & push this to, make sure “sync to remote” is selected.   
    
  

This will: create the branch locally & remotely **that is set as the default “main” branch if there is not an existing main - which there isn’t at this point**, switch you over to that branch locally, and push updates to the remote branch. *Note: you can avoid all this by running usethis::use\_github() instead, but to avoid issues in lab we’re doing it this way.*

1. Let your partner know how to install it from GitHub so they can test it!

* Now, share with your partner the code to install your package from GitHub, which will look like:

devtools::install\_github(“githubname/reponame”)

For example, if my package name (and repo name, since we’ll call them the same thing) is **tacobear**, then that code would look like:

devtools::install\_github(“allisonhorst/tacobear”)

* Ask your partner to test your functions (tell them the function names!)

1. Partners: install & use their package!

* Close the project you’re working in (your package project) (File > Close project). In RStudio, attach the devtools package in the Console.
* In the Console, install your partner’s R package using the install code they shared with you (the install\_github() code from above)
* In a new .Rmd or script, attach the tidyverse and your partner’s package, then try out their functions! Remember, you can use ?function\_name in the Console to get information about what the different function arguments are.

**Congratulations, you made an R package that anyone can use!**

**Note:** if you thought this process for adding a GitHub repo after the fact was a bit tedious, there’s an easier way (but you need to have the PAT set up):

usethis::use\_git()

usethis::use\_github()