Packages

Naomi Tague

January, 2023

PACKAGES

- **Package**: An extension of the R base system with code, data and documentation in standardized format.
- **Library**: A directory containing installed packages.
- **Repository**: A website providing packages for installation.
- **Source**: The original version of a package with human-readable text and code.
- **Binary**: A compiled version of a package with computer-readable text and code, may work only on a specic platform. compile takes your source code and makes it machine readable
- **Base packages**: Part of the R source tree, maintained by R Core.
- **Recommended packages**: Part of every R installation, but not necessarily maintained by R Core.
- **Contributed packages**: All the rest. This does not mean that these packages are necessarily of lesser quality than the above, e.g., many contributed packages on CRAN are written and maintained by R Core members. They simply try to keep the base distribution as lean as possible.
- **User packages**: Packages that you write, share with a smaller community - not downloadable from CRAN, but can be downloaded (or sent as a zip file) and loaded into R

Why make packages

- to easily share functions and data with others with R-specific documentation
- have a set of tools (functions) that you often use, that can be easily loaded
- be part of the R community

Installing User Packages

- looks similar to CRAN packages
- uncompiled (they can be compiled but not always) so you can **see** the code
- you can transfer packages as a *tar* and *zipped* files, but most packages are now on git, so we can use devtools::install_github

You can load my package *ESM262Examples* by

This won't include Rmarkdowns, but will load all the functions into your workspace

- *devtools::install_github("naomitague/ESM_262_Examples")*
- *library(ESM262Examples)*
- to make sure it works,try the following
  - *help(compute_NPV)*
  - *spring_summary*
  - *note that this shows you the code used to implement spring_summary*

Making your own packages

I'm giving you are brief into - more can be found at

More about R-packages

First lets review the format for writing functions

- code your function in named **something.R**
- documentation at the top and inline (we will say more on this later)
- dont' have ANYTHING else in your **something.R** file
- use one of the functions that you've created in the course`

Steps for starting a package

- Think of a name for your package, it should be descriptive of the many related functions that you will store there

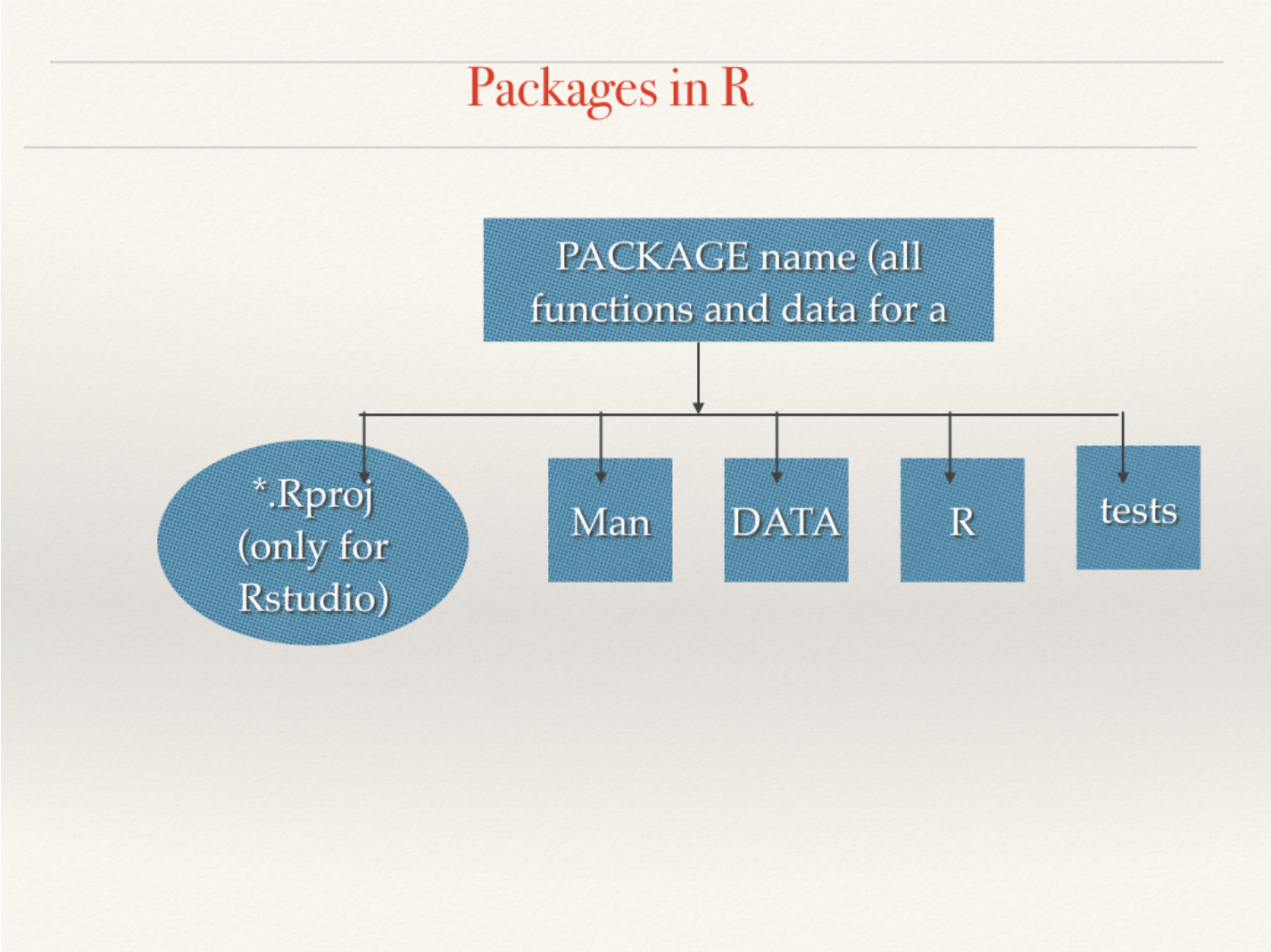
```
* mine will be **ESM262Examples**
```

- Create repository for this package on **github**, each package needs its own repository - where you store only material for the package
- Back in Rstudio, start a new project, using the same name for the project as your package name
  - *choose the option in package creation to **create a package***
  - *make sure you click the box that also creates a local git repository*
  - *If you already have an R function (you do!) you can add it here*
    - make sure your function is named *something.R* and follows the format for functions that we have been using
    - try adding one of the functions that you created
  - *make sure that the path to the directory you create is not in the course directory, or other directory that is already under version control (git)*
  - *first thing to do once you've created the package*
  - *before you commit - link to your **Github** repository*
    - make sure you are in the project directory
    - in your project, in a *terminal* window execute the following commands
      - *git remote add origin 'http link from git'*
      - *git pull origin main*
    - now in Rstudio, checkout a new branch called *main*, it will ask you if you want to checkout existing branch - say yes
      - commit the package files you have so far
      - push to **github** as usual
- Install libraries **devtools** and **roxygen2** from CRAN

Initial Package Structure

You will notice that this creates a directory structure that looks something like this

R Package Directory



You R function will be stored in the **R** subdirectory

IF **man** and **data**, **tests** subdirectories are not created - created them

This package (e.g *ESM262Examples*) directory structures store your code and documentation; that is easy for R to read. It include

- A file DESCRIPTION with descriptions of the package, author, and license conditions
- A man/ subdirectory of documentation files.
- An R/ subdirectory of R code.
- A data/ subdirectory of datasets.

How to work on your package

As you work on your package - if you make a change save it, then re-install using either:

- **Build** - **Build and Reload** from menu
- **load_all()** command from command line

This will Load your functions in your package that is under-development

You can add additional functions to the R subdirectory, when you do that you need to rebuild and reload (see above)

Note: You can add DESCRIPTION to add useful information particularly if you are going to share this package; it also includes dependencies - other packages that your package depends upon (e.g *tidyverse*) - we will come back to this

It is useful to keep an Rmarkdown file or R script that lets you run your function -and helps you in the development but this will not be part of your package!

Remember to commit your results as you go and push to **Github**

Descripition file Dependencies

Your package may use other packages - to make sure they are available in your package - you can add them to DESCRIPTION by using

usethis::use_package("name")

so for example

usethis::use_package("purrr")

now your functions can use *purrr* functions - its good idea in writing your own packages to explicitly reference the package that any function comes from So for example

purrr::map_df(fishes, fish_summary)

Tidyverse is a bit different as its not really a package - to include tidyverse in your package

use_tidy_dependencies()

You can and should edit the following in the **DESCRIPTION** file Authors, Description, Title

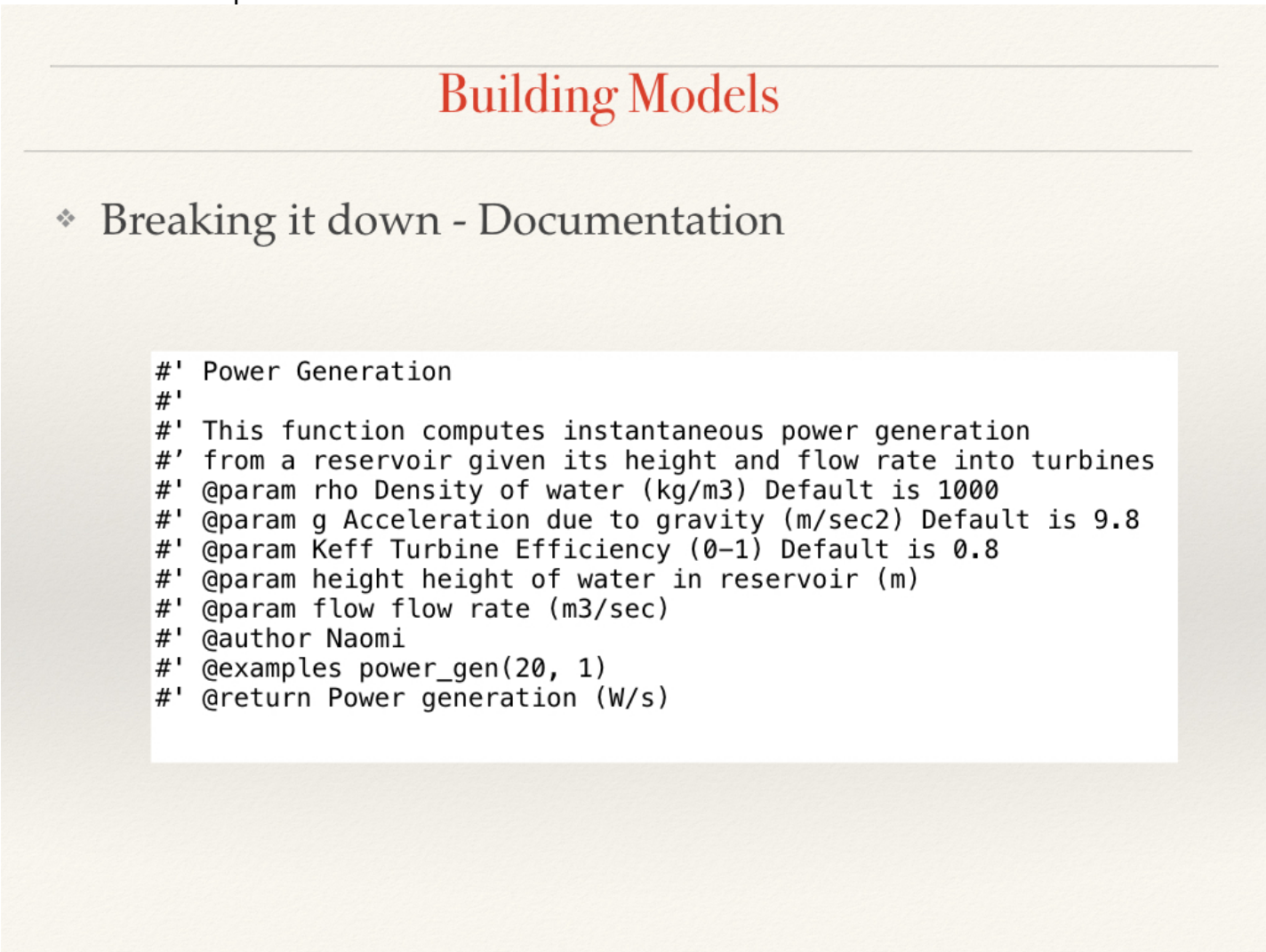
Documentation

- There is a standard format for documentation that can be read by automatic programs (roxygen2) - an R package that generate "standard" R documentation - manual or help pages
- These automated approaches for building documentation (like roxygen2) and meta data (descriptions of data sets) are increasingly common - so you should get into the practice of being structured in your approach to documentation
- We will use the conventions that work with roxygen2 - and then use this program to generate formal R documentation. Roxygen is similar to Doxygen which is used for C code…so its a widely used format

Documentation is placed at the top of the *something.R* file all lines start with #'

Here's an example

Documentation example



Three Parts to R documentation

- **Description** - summary of what your model/function does
- **Tagged (using special "key" words)**

Here are some examples there are many others

- @*param* inputs/parameter description
- @*return* what your function returns (outputs)
- @*example* how to use it
- @*references* citations or urls
- @*author* YOU

(you don't need all of these and there are more tags, but start with at least param and return, example is a good idea

- **Within function** Any use of # within the body of the function (this does not get used in the Help)

Creating Help pages

If you use this approach then *roxygen* can generate help pages - here are the steps

- Add the documentation information as described above to your function
- Save your function, make sure that it is in the *R/* subdirectory your project directory eg *esm237examples/R/compute_NPV.R*
- Set your working directory to your project directory
- Make sure you've run *library(roxygen2)*
- Run *document()* from Rstudio drop down menu or *document()* from command line - this will create documentation for ANY of the *.R files you have in the R directory (you can have many of them)
- try *help(test.R)* to see the results
- you don't have to re-run document the next time your run R, to load (used when you have project and are developing your new package), if it is another package this will be part of the install
- if you change one of your R functons or the documentation, you need to build and re-load
- try adding another small function - something simple with documentation