Package data

Exported data

Many packages contain sample data (e.g. `nycflights13`, `babynames`, etc.)

Generally these files are made available by saving a single data object as an `.Rdata` file (using `save()`) into the `data/` directory of your package.

- Use `usethis::use_data(obj)` to create the necessary files
- Data is usually compressed, for large data sets it may be worth trying different options (there is a 5 Mb package size limit on CRAN)
- Exported data must be documented (possible via Roxygen)

Lazy data

By default when attaching a package all of that packages data is loaded - however if `LazyData: true` is set in the packages' `DESCRIPTION` then data is only loaded when used.

```
pryr::mem_used()
```

```
## 38.5 MB
```

```
library(nycflights13)
pryr::mem_used()
```

```
## 46.3 MB
```

```
invisible(flights)
pryr::mem_used()
```

```
## 87 MB
```

If you use `usethis::use_data()` this option will be set in `DESCRIPTION` automatically.

Raw data

The package should only contain the final data set, but it is important that the process to generate the data is documented as well as any necessary preliminary data.

- These can live any where but the general suggestion is to create a `data-raw/` directory which is included in `.Rbuildignore`
- Should contain scripts, data files, and anything else needed to generate the final object
- See examples babynames or nycflights
- Use `usethis::use_data_raw()` to create and ignore the `data-raw/` directory.

Internal data

If you have data that you want to have access to from within the package but not exported then it needs to live in a special Rdata object located at `R/sysdata.rda`.

- Can be created using `usethis::use_data(obj1, obj2, internal = TRUE)`
- Each call to the above will overwrite, so needs to include all objects
- No necessary for small data frames and similar objects - just create in a script. Use when you want the object to be compressed.
- Example nflplotR which contains team logos and colors for NFL teams.

Raw data files

If you want to include raw data files (e.g `.csv`, shapefiles, etc.) there are generally placed in folders within `inst/` so that they are installed with the package.

- Accessed using `system.file("dir", package = "package")` after install
- Use folders to keep things organized, Hadley recommends and uses `inst/extdata/`
- Example sf

Package checking

Package vigenette

Vignette

Long form documentation for your package that live in `vignette/`, use `browseVignette(pkg)` to see a package's vignettes.

- Not required, but adds a lot of value to a package
- Generally these are literate documents (`.Rmd`, `.Rnw`) that are compiled to `.html` or `.pdf` when the package is built.
- Built package retained the rendered document, the source document, and all source code
  - `vignette("colwise", package = "dplyr")` opens rendered version
  - `edit(vignette("colwise", package = "dplyr"))` opens code chunks
- Use `usethis::use_vignette()` to create a RMarkdown vignette template

Package testing

Basic test structure

Package tests live in `tests/`,

- Any R scripts found in the folder will be run when Checking the package (not Building)
- Generally tests fail on errors, but warnings are also tracked
- Testing is possible via base R, including comparison of output vs. a file but it is not recommended (See Writing R Extensions)


test
that
www.rstudio.com

testthat basics

Not the only option but probably the most widely used and with the best integration into RStudio.

Can be initialized in your project via `usethis::use_testthat()` which creates `tests/testthat/` and some basic scaffolding.

- `test/testthat.R` is what is run by the Check and runs your other tests - handles some basic config like loading package(s)
- Test scripts go in `tests/testthat/` and should start with `test_`, suffix is usually the file in `R/` that is being tested.

`usethis::use_testthat()` has an edition argument, this is a way of maintaining backwards compatibility, generally always use the latest if starting a new project

testthat script structure

From the bottom up,

- a single test is written as an expectation (e.g. `expect_equal()`, `expect_error()`, etc.)
- multiple related expectations are combined into a test group (`test_that()`), which provides
  - a human readable name and
  - local scope to contain the expectations and any temporary objects
- multiple test groups are combined into a file

Package publishing


THIS SIDE UP
pkgdown
www.rstudio.com

CRAN

More details than we can get into today, but see the the Release a package chapter of R Pkgs for more details.

- `devtools::release()` will take you through a number of basic checks and if everything is good upload your package to the submission queue of CRAN.