GG606

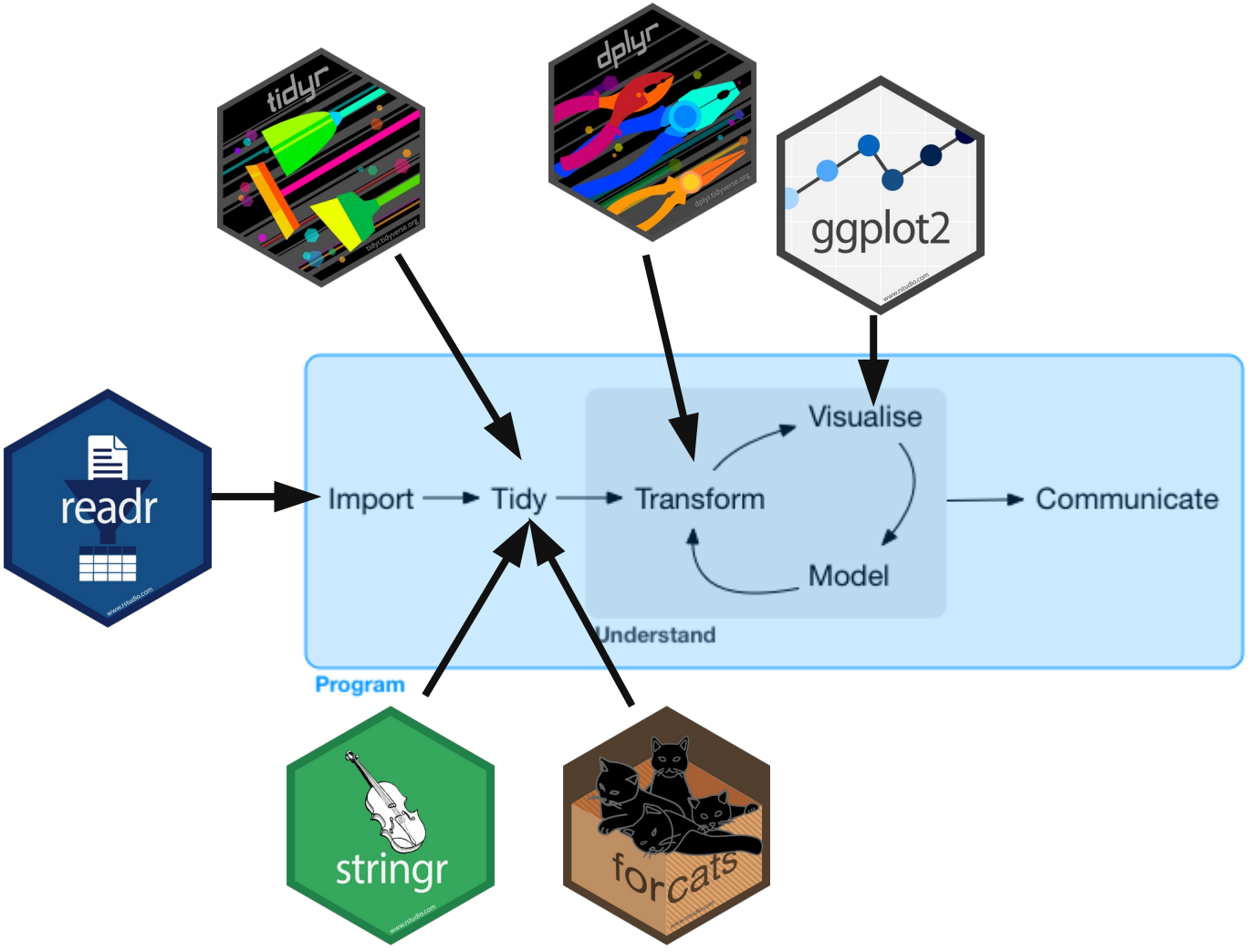
Functions & Packages

Homework/Review

- Functions for variance and skewness
- Function to load the penguins.csv
  do a calculation, and make a figure
- Call the function like this:
  `my_fancy_function(filename)`

$$\mathrm{Var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$\mathrm{Skew}(x) = \frac{\frac{1}{n-2}\left(\sum_{i=1}^{n}(x_i - \bar{x})^3\right)}{\mathrm{Var}(x)^{3/2}}$$


tidyr
dplyr
ggplot2
readr
Import
Tidy
Transform
Visualise
Model
Communicate
Understand
Program
stringr
forcats

Functions for humans & computers

- verb(nouns)

```
# Too short
f()

# Not a verb, or descriptive
my_awesome_function()

# Long, but clear
impute_missing()
collapse_years()
```

Functions for humans & computers

- So much wrong!
- *snake_case*
- *camelCase*

```
# Never do this!
col_mins <- function(x, y) {}
rowMaxes <- function(y, x) {}
```

Functions for humans & computers

- Common prefix
- `stringr::str_`
- Avoid overwriting

```
# Good
input_select()
input_checkbox()
input_text()

# Not so good
select_input()
checkbox_input()
text_input()
```

Functions for humans & computers

- Hilarious joke to play on your friends

```
# Don't do this!
T <- FALSE
c <- 10
mean <- function(x) sum(x)
```

Puzzle

- ```
  f1 ← function(string, prefix) {
    substr(string, 1, nchar(prefix)) == prefix
  }
  ```
- ```
  f2 ← function(x) {
    if (length(x) ⩽ 1) return(NULL)
    x[-length(x)]
  }
  ```
- ```
  f3 ← function(x, y) {
    rep(y, length.out = length(x))
  }
  ```
  ```

Puzzle

- ```
  f1 ← function(string, prefix) {
    substr(string, 1, nchar(prefix)) == prefix
  }
  ```
  has_prefix()
- ```
  f2 ← function(x) {
    if (length(x) ⩽ 1) return(NULL)
    x[-length(x)]
  }
  ```
  drop_last()
- ```
  f3 ← function(x, y) {
    rep(y, length.out = length(x))
  }
  ```
  recycle()
  expand()
  ```

Conditions

- R: `if() elseif() else`
- Python: `if : else:`
- C++: `if() elseif() else`
- Matlab: `if elseif else end`
- Excel: `IF(a,b,c)`

How many have experience with if statements?

Simple Example

- What does it do
- How are `if` and `else` working?
- What will output(s) look like?

```r
has_name <- function(x) {
  nms <- names(x)
  if (is.null(nms)) {
    rep(FALSE, length(x))
  } else {
    !is.na(nms) & nms != ""
  }
}
```

Code Style

```
# Good
if (y < 0 && debug) {
  message("Y is negative")
}

if (y == 0) {
  log(x)
} else {
  y ^ x
}
```

```
# Bad
if (y < 0 && debug)
message("Y is negative")

if (y == 0) {
  log(x)
}
else {
  y ^ x
}
```

Function arguments

- Data first then details
- `log(x =, base = )`
- `mean(x = , trim = , na.rm = )`
- `t.test(x = )`

Type `log(` and hit TAB

White space is your friend

Argument names

- Longer & descriptive
- Some common short names
- Match existing arguments

- `x`, `y`, `z`: vectors.
- `w`: a vector of weights.
- `df`: a data frame.
- `i`, `j`: numeric indices (typically rows and columns).
- `n`: length, or number of rows.
- `p`: number of columns.

df is not a good idea

Check some|all conditions

```r
wt_mean <- function(x, w) {
  sum(x * w) / sum(w)
}
```

```r
wt_mean <- function(x, w) {
  if (length(x) != length(w)) {
    stop("`x` and `w` must be the same length", call. = FALSE)
  }
  sum(w * x) / sum(w)
}
```

`?stopifnot`

Ensure the Truth of R Expressions

Weighted mean

R's recycling rules

Dot-dot-dot (...)

- Arbitrary number of inputs
- ... captures them and makes them available to other functions

```
sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
#> [1] 55
stringr::str_c("a", "b", "c", "d", "e", "f")
#> [1] "abcdef"
```

Examples?

Dot-dot-dot (...)

```
commas <- function(...) stringr::str_c(..., collapse = ", ")
commas(letters[1:10])
#> [1] "a, b, c, d, e, f, g, h, i, j"
```

Simplicity

Returns

- Functions returns something
- Return the last evaluation
- Can use `return( )`, but when?

```
complicated_function <- function(x, y, z) {
  if (length(x) == 0 || length(y) == 0) {
    return(0)
  }

  # Complicated code here
}
```

Returns

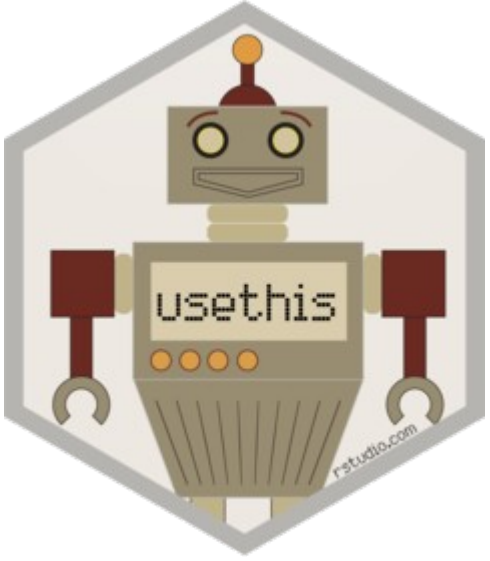
```
f <- function() {
  if (x) {
    # Do
    # something
    # that
    # takes
    # many
    # lines
    # to
    # express
  } else {
    # return something short
  }
}
```

```
f <- function() {
  if (!x) {
    return(something_short)
  }

  # Do
  # something
  # that
  # takes
  # many
  # lines
  # to
  # express
}
```

Packages

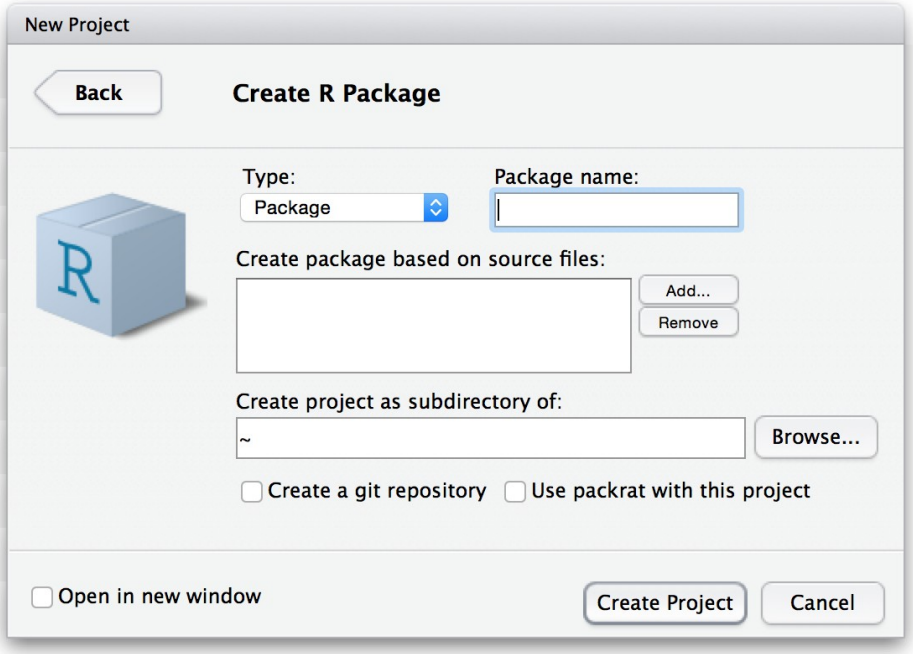
- Many ways to do this
  - Two simple ways: RStudio & `usethis`
- Walk through both
- Very few *required* parts to a package
- Storage/access



Why packages

- Keep all scripts together
- Create `ggplot2` themes
- Collection of small functions
- Curate data
- Templates
- Easy to add version control

RStudio



- Project
- New Project
- R Package
- Minimal contents appear

Files

- D: basic info
- man: manual/help
- N: functions
- R: code

First function

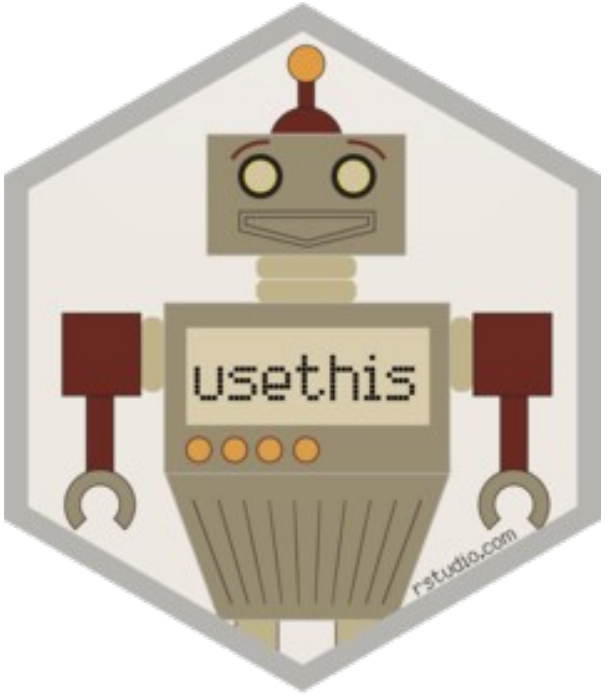
```r
# Hello, world!
#
# This is an example function named 'hello'
# which prints 'Hello, world!'.
#
# You can learn more about package authoring with RStudio at:
#
#   http://r-pkgs.had.co.nz/
#
# Some useful keyboard shortcuts for package authoring:
#
#   Install Package:           'Ctrl + Shift + B'
#   Check Package:             'Ctrl + Shift + E'
#   Test Package:              'Ctrl + Shift + T'

hello ← function() {
  print("Hello, world!")
}
```

- Comments
- *Help info*
- Function

https://r-pkgs.org/

usethis



- `create_package()`
- `use_*()`
- `load_all()`
- `check()`
- `document()`

https://usethis.r-lib.org/

usethis

```
> library(usethis)
> create_package("~/github/myTestPackage")
✔ Creating '/home/jason/github/myTestPackage/'
✔ Setting active project to '/home/jason/github/myTestPackage'
✔ Creating 'R/'
✔ Writing 'DESCRIPTION'
Package: myTestPackage
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R (parsed):
    * First Last <first.last@example.com> [aut, cre] (YOUR-ORCID-ID)
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends to
    pick a license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.2.3
✔ Writing 'NAMESPACE'
✔ Writing 'myTestPackage.Rproj'
✔ Adding '^myTestPackage\\.Rproj$' to '.Rbuildignore'
✔ Adding '.Rproj.user' to '.gitignore'
✔ Adding '^\\.Rproj\\.user$' to '.Rbuildignore'
✔ Opening '/home/jason/github/myTestPackage/' in new RStudio session
✔ Setting active project to '<no active project>'
```

- Creates the files
- Opens the project in a new window
- Continue…

https://usethis.r-lib.org/

Introduction to pkgdown

Source: `vignettes/pkgdown.Rmd`

The goal of pkgdown is to make it easy to make an elegant and useful package website with a minimum of work. You can get a basic website up and running in just a couple of minutes:

```
# Run once to configure package to use pkgdown
usethis::use_pkgdown()
# Run to build the website
pkgdown::build_site()
```

If you're using GitHub, we also recommend setting up GitHub actions to automatically build and publish your site:

```
usethis::use_pkgdown_github_pages()
```

While you'll get a decent website without any additional work, if you want a website that really pops, you'll need to read the rest of this vignette. It starts by showing you how to configure pkgdown with a `_pkgdown.yml`. You'll learn about the main components of the site (the home page, reference, articles, and news), and then how to publish and promote your site.

https://pkgdown.r-lib.org/

GitHub Pages

- Quarto is simple enough
- Jekyll, Hugo & Hexo are static site generators
- Plain markdown or code-markdown
- Blogdown

Quarto + GitHub Pages

- Simple instructions

1) Render to docs folder, check into GitHub;

2) Publish content rendered locally; *or*

3) GitHub Action to render files & publish

- Setup repo first: username.github.io/reponame

Quarto + GitHub Pages

- Setup repo first but it’s empty
- Clone with RStudio into a new project

1)Write by hand

2)Generate a template

- Some instructions assume you have content ready

Quarto + GitHub Pages

- Need `_quarto.yml`
- Need `index.qmd`
- Should have `.nojekyll`

Quarto + GitHub Pages

- Render (into docs/), commit, push (gui or cli)

```
quarto render
git add docs
git commit -m "Publish site to docs/"
git push
```

Quarto + GitHub Pages

- Configure GitHub repo to use docs/