Review

Daniel Anderson

Week 9, Class 2

Agenda

- Review functions
- A shiny challenge
    - But first, a brief discussion on publishing shiny apps

Functions review

Remember

- Everything is a function

The following are equivalent

```
3 + 5
```

```
## [1] 8
```

```
`+`(3, 5)
```

```
## [1] 8
```

Using functions

- Most functions are bound to a name, e.g., `mean()`
- Anonymous functions are also common
    - Apply the function in a loop, and it only ever exists in the loop
- You can also store functions in lists
    - Helpful if you want to apply lots of operations to a single vector

Binding to a name

- Let's create a function that takes two arguments: (a) a data frame, and (b) the name of a discrete/categorical variable/column in the data frame.
- The function should return the count of each "level" in the categorical variable.
- For a small added challenge, have it optionally add the proportion

Example output with `palmerpenguins::penguins`.

```
##     species count
## 1    Adelie   152
## 2 Chinstrap    68
## 3    Gentoo   124
```

```
##     species count proportion
## 1    Adelie   152  0.4418605
## 2 Chinstrap    68  0.1976744
## 3    Gentoo   124  0.3604651
```

You try first

Test it out with the **palmerpenguins** dataset. Do you get the same results I did?

Note – the example I used included only base R functions. You can feel free to use **dplyr** or whatevs, just be careful with NSE.

You also don't have to return a data frame output – return it however you want

07:00

Where to start?

~~write a function~~

- Solve the problem for one example, generalize it to a function.

Use the **palmerpenguins** dataset *for* your example!

```
library(palmerpenguins)
penguins
```

```
## # A tibble: 344 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body
##    <fct>   <fct>              <dbl>         <dbl>             <int>
##  1 Adelie  Torgersen           39.1          18.7               181
##  2 Adelie  Torgersen           39.5          17.4               186
##  3 Adelie  Torgersen           40.3          18                 195
##  4 Adelie  Torgersen           NA            NA                  NA
##  5 Adelie  Torgersen           36.7          19.3               193
##  6 Adelie  Torgersen           39.3          20.6               190
##  7 Adelie  Torgersen           38.9          17.8               181
##  8 Adelie  Torgersen           39.2          19.6               195
```

How do you want to solve it?

Lots of ways, here's a base method

- First, split by species

```
splt <- split(penguins, penguins$species)
```

- Next, count how many rows (observations) in each split

```
sapply(splt, nrow)
```

```
##    Adelie Chinstrap    Gentoo
##       152        68       124
```

Could go on, but this is basically the output.

Wrap in a function

- What will the arguments be?

The data frame and the column

```
get_counts <- function(df, column) {

}
```

What will the body be?

Same as before

Just swap out the code for the arguments. Notice I'm indexing the columns differently. Why?

I'm also swapping out `sapply()` for `vapply()` to be a little more safe.

```
get_counts <- function(df, column) {
  splt <- split(df, df[[column]])
  vapply(splt, nrow, FUN.VALUE = integer(1))
}
```

Test it

```
get_counts(penguins, "species")
```

```
##    Adelie Chinstrap    Gentoo
##       152        68       124
```

```
get_counts(penguins, "island")
```

```
##    Biscoe     Dream Torgersen
##       168       124        52
```

Extensions

Let's say we want a data frame as the output.

Can you modify what we have now to make that so?

03:00

Data frame

```
get_counts <- function(df, column) {
  splt <- split(df, df[[column]])
  counts <- vapply(splt, nrow, FUN.VALUE = integer(1))

  tibble::tibble(
    var_levels = names(counts), # could use names(splt)
    count = counts
  )
}
```

Test it

```
get_counts(penguins, "species")
```

```
## # A tibble: 3 x 2
##   var_levels count
##   <chr>      <int>
## 1 Adelie       152
## 2 Chinstrap     68
## 3 Gentoo       124
```

```
get_counts(penguins, "island")
```

```
## # A tibble: 3 x 2
##   var_levels count
##   <chr>      <int>
## 1 Biscoe       168
## 2 Dream        124
## 3 Torgersen     52
```

Column name

Can we make the output from the data frame have the same column that we fed it?

```
get_counts <- function(df, column) {
  splt <- split(df, df[[column]])
  counts <- vapply(splt, nrow, FUN.VALUE = integer(1))

  d <- tibble::tibble(
    var_levels = names(counts), # could use names(splt)
    count = counts
  )
  names(d)[1] <- column
  d
}
```

02:00

Test it

```
get_counts(penguins, "species")
```

```
## # A tibble: 3 x 2
##   species   count
##   <chr>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

```
get_counts(penguins, "island")
```

```
## # A tibble: 3 x 2
##   island    count
##   <chr>     <int>
## 1 Biscoe      168
## 2 Dream       124
## 3 Torgersen    52
```

{dplyr} version

Can we replicate this function using dplyr?

We'll have to use non–standard evaluation

First, solve it on a use case

```
penguins %>%
  count(species)
```

```
## # A tibble: 3 x 2
##   species       n
##   <fct>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

Function

Will this work?

```
get_counts <- function(df, column) {
  df %>%
    count(column)
}
```

```
get_counts(penguins, species)
```

```
## Error: Must group by variables found in `.data`.
## * Column `column` is not found.
```

Use NSE

```
get_counts <- function(df, column) {
  df %>%
    count({{column}})
}
```

```
get_counts(penguins, species)
```

```
## # A tibble: 3 x 2
##   species       n
##   <fct>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

```
get_counts(penguins, island)
```

```
## # A tibble: 3 x 2
##   island        n
##   <fct>     <int>
## 1 Biscoe      168
## 2 Dream       124
## 3 Torgersen    52
```

Pass the dots

Alternatively, you could just pass the dots

Bonus, this will now give you the counts for multiple columns

```
get_counts <- function(df, ...) {
  df %>%
    count(...)
}
```

Test it

```
get_counts(penguins, species)
```

```
## # A tibble: 3 x 2
##   species       n
##   <fct>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

```
get_counts(penguins, species, island)
```

```
## # A tibble: 5 x 3
##   species   island        n
##   <fct>     <fct>     <int>
## 1 Adelie    Biscoe       44
## 2 Adelie    Dream        56
## 3 Adelie    Torgersen    52
## 4 Chinstrap Dream        68
## 5 Gentoo    Biscoe      124
```

Conditions

- Let's add a condition that optionally reports the proportions in addition to the counts.
- What will be the first step?
- Add a new argument (and consider setting defaults for that argument)

```
get_counts <- function(df, column, return_proportions = FALSE) {
  df %>%
    count({{column}})
}
```

Set conditional block

Create a block for operations to conduct **when the condition is TRUE**

```
get_counts <- function(df, column, return_proportions = FALSE) {
  counts <- df %>%
    count({{column}})

  if (isTRUE(return_proportions)) {

  }
  counts
}
```

Write condition

In the block, include the code that is only evaluated when the condition is TRUE.

```
get_counts <- function(df, column, return_proportions = FALSE) {
  counts <- df %>%
    count({{column}})

  if (isTRUE(return_proportions)) {
    counts <- counts %>%
      mutate(proportion = n / sum(n))
  }
  counts
}
```

Test it

```
get_counts(penguins, species)
```

```
## # A tibble: 3 x 2
##   species       n
##   <fct>     <int>
## 1 Adelie      152
## 2 Chinstrap    68
## 3 Gentoo      124
```

```
get_counts(penguins, species, return_proportions = TRUE)
```

```
## # A tibble: 3 x 3
##   species       n proportion
##   <fct>     <int>      <dbl>
## 1 Adelie      152  0.4418605
## 2 Chinstrap    68  0.1976744
## 3 Gentoo      124  0.3604651
```

Challenge

Now that we have a basic function, can you write a **new** function that *calls this function* to add the proportions and/or counts to a data frame?

Should return the original data frame, but with the counts/proportions added in as a new column.

04:00

One solution

```
add_counts <- function(data, column, add_proportions = FALSE) {
  counts <- get_counts(data, {{column}}, add_proportions)
  left_join(data, counts)
}
```

Test it out

I'm selecting variables after just so we can see the counts

```
add_counts(penguins, species) %>%
  select(species, island, n)
```

```
## # A tibble: 344 x 3
##    species island        n
##    <fct>   <fct>     <int>
##  1 Adelie  Torgersen   152
##  2 Adelie  Torgersen   152
##  3 Adelie  Torgersen   152
##  4 Adelie  Torgersen   152
##  5 Adelie  Torgersen   152
##  6 Adelie  Torgersen   152
##  7 Adelie  Torgersen   152
##  8 Adelie  Torgersen   152
##  9 Adelie  Torgersen   152
## 10 Adelie  Torgersen   152
## # … with 334 more rows
```

Test it again

This time let's add the proportions

```
add_counts(penguins, species, add_proportions = TRUE) %>%
  select(species, island, n, proportion)
```

```
## # A tibble: 344 x 4
##    species island        n proportion
##    <fct>   <fct>     <int>      <dbl>
##  1 Adelie  Torgersen   152  0.4418605
##  2 Adelie  Torgersen   152  0.4418605
##  3 Adelie  Torgersen   152  0.4418605
##  4 Adelie  Torgersen   152  0.4418605
##  5 Adelie  Torgersen   152  0.4418605
##  6 Adelie  Torgersen   152  0.4418605
##  7 Adelie  Torgersen   152  0.4418605
##  8 Adelie  Torgersen   152  0.4418605
##  9 Adelie  Torgersen   152  0.4418605
## 10 Adelie  Torgersen   152  0.4418605
## # … with 334 more rows
```

Embed checks

Can you embed a warning or error (your choice) if the column fed to the function is not discrete?

Note – this is more difficult with our **dplyr** version. Try using `dplyr::pull()`.

03:00

```r
get_counts <- function(df, column, return_proportions = FALSE) {

  column_vec <- dplyr::pull(df, {{column}})

  if(is.numeric(column_vec)) {
    stop("Numeric column passed to function. Counts must be compu
  }

  counts <- df %>%
    count({{column}})

  if (isTRUE(return_proportions)) {
    counts <- counts %>%
      mutate(proportion = n / sum(n))
  }
  counts
}
```

Test it out

Note we can test it with either the `get_counts()` or `add_counts()` functions

```
get_counts(penguins, bill_length_mm)
```

```
## Error in get_counts(penguins, bill_length_mm): Numeric column passed to
```

```
add_counts(penguins, bill_length_mm)
```

```
## Error in get_counts(data, {: Numeric column passed to function. Counts m
```

Shiny

Publishing

- We never talked about publishing shiny apps

See here for a nice step–by–step walkthrough for publishing with https://www.shinyapps.io/

- Register an account with https://www.shinyapps.io/
- Add a token to your account on shinyapps
- Back locally, set your account info with the token and secret via

```
rsconnect::setAccountInfo(
  name = "myaccount", # replace with your account name
  token = "mytokencopiedfromshinyappsio", # your token
  secret = "mysecretcopiedfromshinyappsio"
)
```

Publish

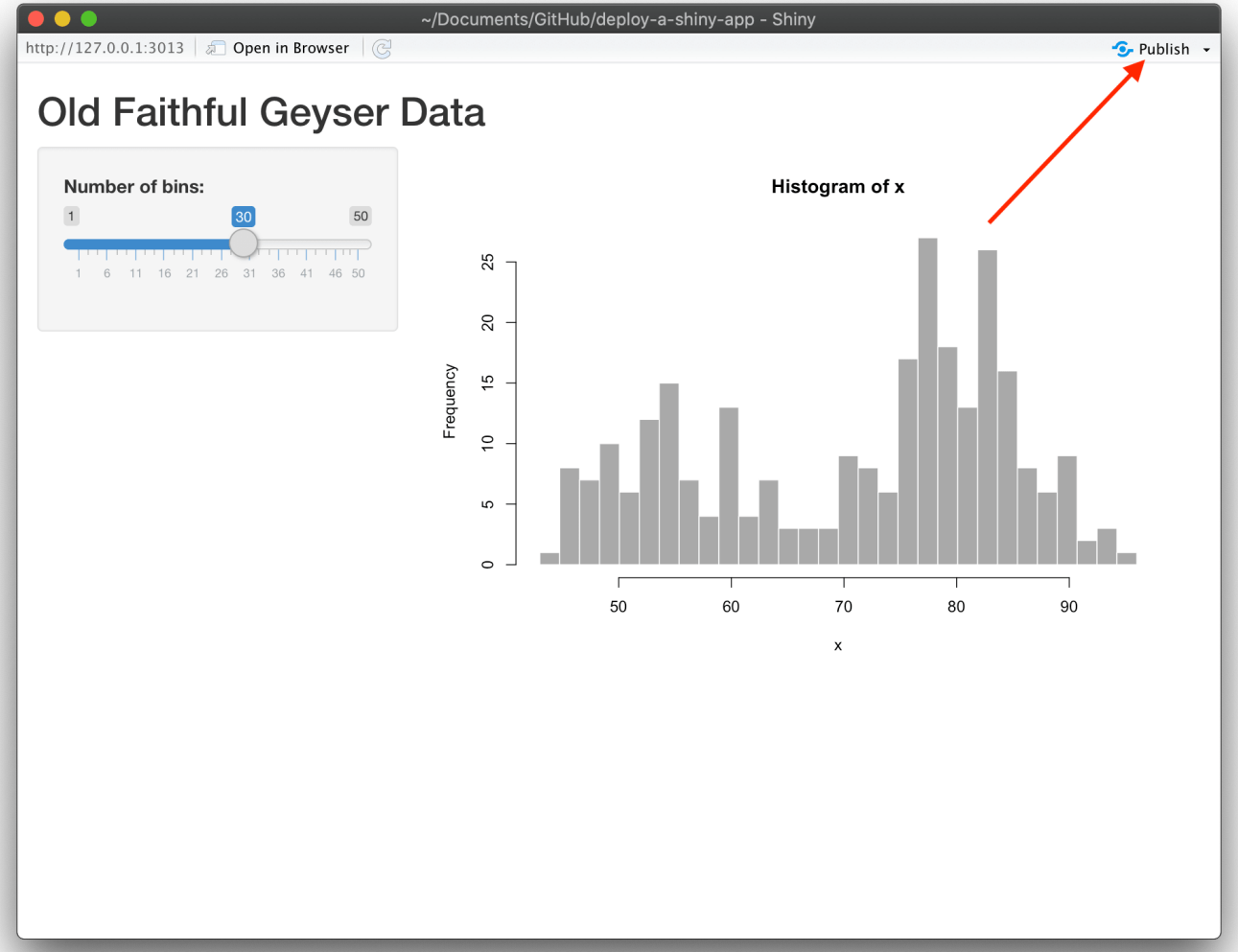

image from tutorial here

Shiny app

- Create a shiny app or shiny dashboard with the `palmerpenguins` dataset
- Allow the x and y axis to be selected by the user
    - Only numeric variables should be available to be selected
- Allow the points to be colored by any categorical variable
    - For an added challenge, try to add in a "no color" option, which should be the default

Once you've gone this far, try to publish your app. If you're successful, continue with challenge on next slide

Challenge continued

- Add a table to the app that reports descriptive data on the columns that are selected in the plot
    - e.g., `n()`, `mean()`, `sd()`
- Use tabs so the plot shows up in one tab, and the table shows up in a different tab

Now publish again to update it

Next time

No Class Monday

Package Development on Wednesday