Week 12: Looping

11/14/2019

Jake Campbell

Applying Functions to Multiple Columns

- Most functions are difficult to apply to multiple columns at once
  - For example, we can't calculate `mean()` on multiple columns at once
- We need to find a way to loop over our dataset, applying our function to each column on each loop
  - Not limited to looping over columns; we can loop over anything
- Several ways to do this R
  - Some better than others

The `for` Loop

- The `for` loop is written similar in style to how we made our own functions
- First specify your `for()` argument
    - What you are looping over
- `i` is a placeholder for the multiple values
- Then within `{}`, specify the function you're applying to each element, `i`
- If we want to save the output somewhere, we should allocate space in an empty object (a list or vector for example)

```
for(i in 1:5){
  print(mean(mtcars[, i], na.rm=T))
}
```

```
## [1] 20.09062
## [1] 6.1875
## [1] 230.7219
## [1] 146.6875
## [1] 3.596563
```

The Apply Family of Functions

- This looping process encapsulates the call into a single function
- Could be slightly faster than traditional looping due to vectorization
    - Speed upgrade is usually overstated

apply()

- `apply()` takes three arguments
  - The object you are looping over
  - The dimension you are looping over (`1` for rows, `2` for columns)
  - The function you are applying
- If you're function has multiple arguments, we can specify `function(x)` to create a placeholder, similar to `i`

```
apply(mtcars[, c(1:5)], 2, function(x) mean(x, na.rm = T))
```

```
##        mpg        cyl       disp         hp       drat
##  20.090625   6.187500 230.721875 146.687500   3.596563
```

`lapply()` and `sapply()`

- These functions can be applied to list objects
- `lapply()` returns a list object, while `sapply()` returns a simpified object
    - The simplified object could be a vector, matrix, or dataframe
- Don't need to specify a dimension we are looping over

`dplyr` and Looping

- We can use several `dplyr` functions in conjunction with `_all` or `_at` to loop over different columns
  - `_all` would apply the function to every column
  - `_at` would apply the function to specific columns
- We can specify the function we want to perform with the `.funs` argument
  - We set this argument to be a list of functions, which we specify with `~`

```
mtcars %>%
  mutate_at(.vars = vars(mpg:drat), .funs = list(~ as.character(.))) %>%
  str()
```

purrr

- `purrr` is a tidyverse package focused on iteration
- Very similar to the apply family of functions
- Centered around the `map()` function
- As a part of the tidyverse, they work well with the `%>%`

purrr

- Different forms of `map()` return different output
  - `map()` returns a list
  - `map_dbl()` and `map_chr()` return numeric and character vectors
  - `map_dfr()` returns a tibble
- If we need to add additional arguments to the function we are iterating, we can use `~` and use `.` as a placeholder for what we are looping over

```
mtcars %>%
  map_dfr(~ mean(., na.rm = T))
```

```
## # A tibble: 1 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  20.1  6.19  231.  147.  3.60  3.22  17.8 0.438 0.406  3.69  2.81
```