STAT 209
Making Repetitive Tasks Modular

March 12, 2018

Colin Reimer Dawson

This Week

Today:

- Intro to Functions and Loops
- R syntax for said

Wednesday:

- Lab on the above

Friday:

- I think I will push web scraping to after break, and devote Friday to revisiting things that could use review

Project 2

- New group Slack channels
  - Figure out ASAP if there are major schedule incompatibilities, and let me know
- Project description here
- Key dates (might move back depending on how things go)
  - Workshop on Wednesday after break (meet with your group before this!)
  - Presentations the following Friday (should be a complete draft!)
  - Meet again with your group after this to coordinate final revisions
  - Final writeup due the Friday after that (4/6)

Outline

Repeated Similar Code

Ever find yourself doing this sort of thing?

```
my.data %>%
    ggplot(aes(x = variable1, y = variable2)) +
    geom_point() + geom_smooth() +
    xlab("Label for Variable 1") + ylab("Label for Variable 2")
my.data %>%
    ggplot(aes(x = variable3, y = variable4)) +
    geom_point() + geom_smooth() +
    xlab("Label for Variable 3") + ylab("Label for Variable 4")
my.other.data %>%
    ggplot(aes(x = variable1, y = variable2)) +
    geom_point() + geom_smooth() +
    xlab("Label for Variable 1") + ylab("Label for Variable 2")
```

There are only a few things changing here: the dataset, the variable names, and the axis labels.

A Plot "Template"

Inputs:

1. Dataset name (`dataset`)
2. "x" variable name (`xvar`)
3. "y" variable name (`yvar`)
4. "x" variable axis label (`xlabel`)
5. "y" variable axis label (`ylabel`)

Procedure:

1. Make a `ggplot` with `dataset`
2. Define an aesthetic: `aes(x = xvar, y = yvar)`
3. Add `geom_point()` and `geom_smooth()`
4. Add labels via `xlab(xlabel)` and `ylab(ylabel)`

R Syntax for "Template" (Function) Definition

```
## As always in R, placement of line breaks and indentation is a matter of
## style and preference, not structural

my_function_name <- function( # the word function is literal
    input1,                   # the argument names are up to you
    input2,
    input3 = "some default value",
    input4 = "another default value")
{
    ...                       # the dots just represent "doing stuff"
    some-R-code-here
    ...
    return(some_result)       # return() needs parens
}
```

- In this function, `input1` and `input2` are required, `input3` and `input4` are optional (will use default values if omitted)
- The return value of `some_result` is typically a variable name that is defined within the function

Using the Function

```
## Here's that definition again, for reference
my_function_name <- function(
  input1, input2, input3 = "default3", input4 = "default4") {...}

#### All of the following are equivalent

## (a) Calling with default arguments
my_function_name(x, y)
## (b) Or override them (in this case with the same thing)
my_function_name(x, y, "default3", "default4")
## (c) Can give arguments in any order if we give the names
## (input4 gets its default value)
my_function_name(input3 = "default3", input1 = x, input2 = y)
## (d) Or a combination of names and positions
## (input3 gets the default value)
my_function_name(x, input2 = y, input4 = "default4")
## (e) We can even use it with a pipe
## (the piped output goes in the first spot, the first
## provided argument in the second, etc.)
x %>% my_function_name(y)
```

Concrete Example

See R script

Some "Gotchas"

We unfortunately can't (directly) pass variable names to `ggplot2` or `dplyr` via arguments to custom functions, since variable names get interpreted in a special way by those packages

```
## This will _not_ work
myfunction <- function(dataset, xvar, yvar)
{
    dataset %>% ggplot(aes(x = xvar, y = yvar)) +
        geom_point() + geom_smooth()
}
## This will _not_ work
AllCountries %>% myfunction(xvar = BirthRate, yvar = LifeExpectancy)
```

Global Variables

```
## This will work as long as AllCountries is loaded when calling it
n_richest_countries <- function(n = 6)
{
    AllCountries %>%
        select(Country, Population, GDP) %>%
        arrange(desc(GDP)) %>%
        head(n = n)
} # the definition runs fine though we haven't loaded the data
```

```
n_richest_countries(n = 3) # Not going to run


    Error in eval(expr, envir, enclos):  object 'AllCountries' not found
```

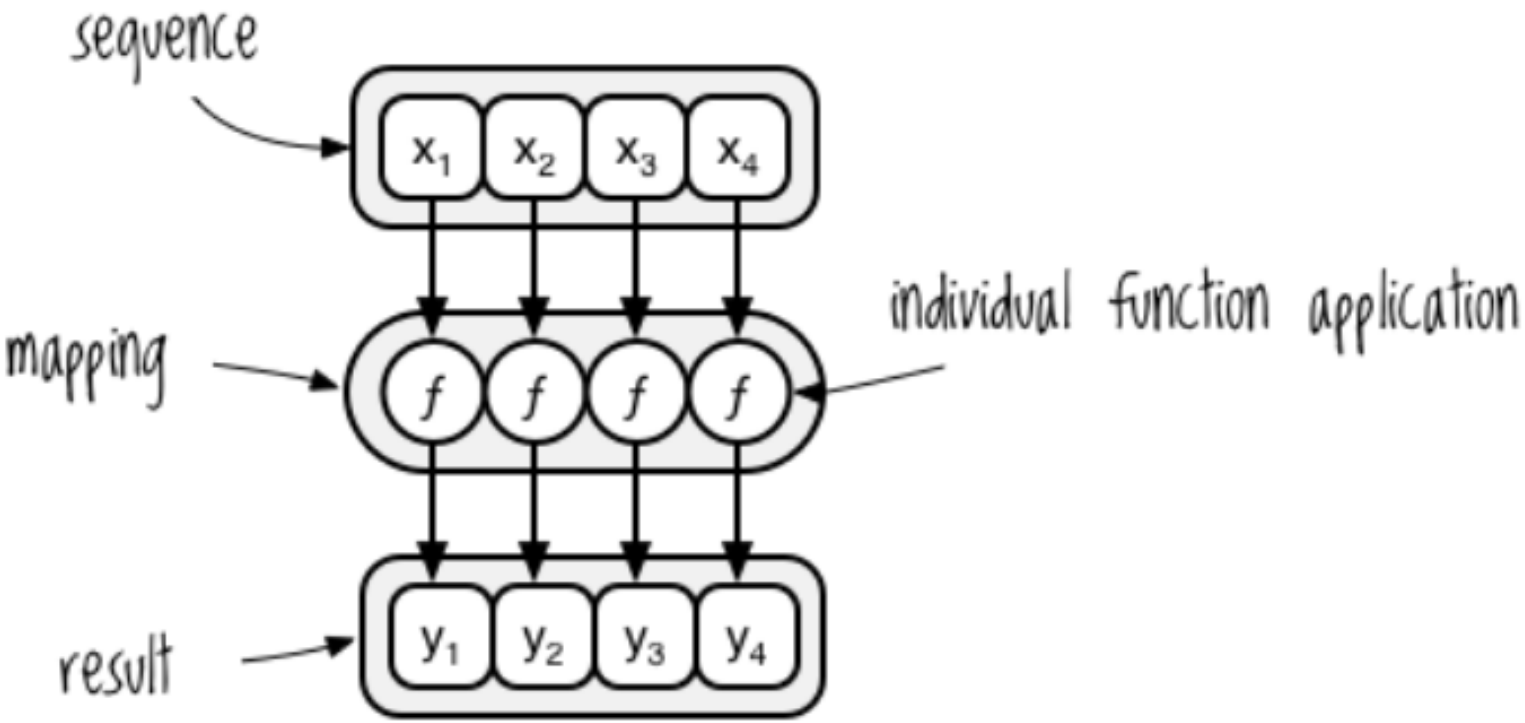
```
data(AllCountries, package = "Lock5Data")
n_richest_countries(n = 3) # Now it's ok


          Country Population       GDP
     1  Luxembourg      0.489 105437.67
     2      Norway      4.768  84538.24
     3 Switzerland      7.648  67463.71
```

Outline

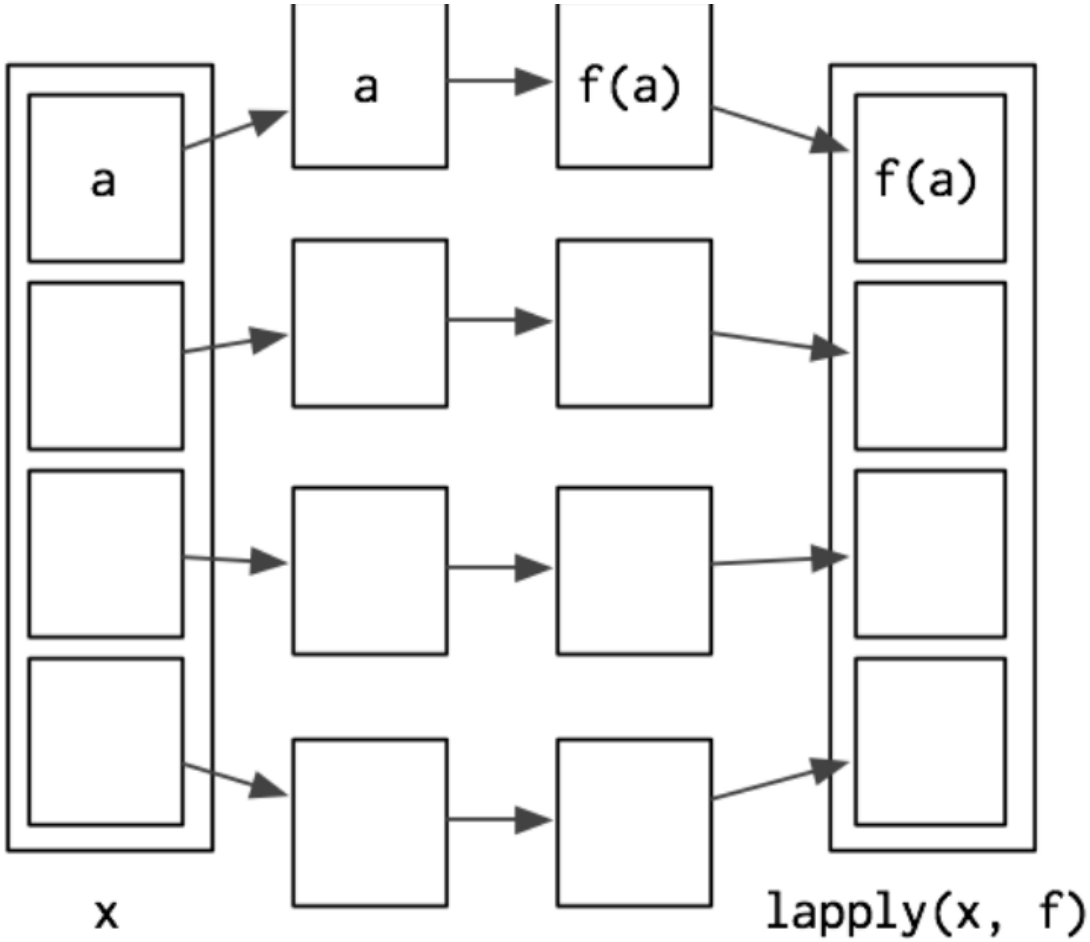
Iteration

- Sometimes you have
  - a function `f(val)` that returns something
  - and a list of `vals` you want to apply `f` to
- Could just copy and paste, but
  - the list might be long
  - this is a nightmare to make changes to
  - we want a nice way to "collect" the results

Mapping


sequence
x_1 x_2 x_3 x_4
mapping
f f f f
individual function application
result
y_1 y_2 y_3 y_4

The `lapply()` function



Working with lists

- `lapply()` always returns a `list`
- Items in a list can be of any type (even all different)
- We can use `unlist()` to convert a list into a vector, provided the elements are the same type
- The `bind_rows()` function flattens a list of data frames into one big data frame...
    - as long as they all have the same variables

Example

```
my_cars <- c("corolla", "civic")
get_avg_mpg <- function(mod) {
  mpg %>%
    filter(model == mod) %>%
    group_by(model) %>%
    summarize(mean_mpg = mean(hwy, na.rm = TRUE))
}
lapply(my_cars, get_avg_mpg) %>%
  bind_rows()
```

```
# A tibble: 2 x 2
    model mean_mpg
    <chr>    <dbl>
1 corolla 34.00000
2   civic 32.55556
```

The `do()` function

- Similar to `summarize()` in that it works over a collection of data frames
- Useful with `group_by()`

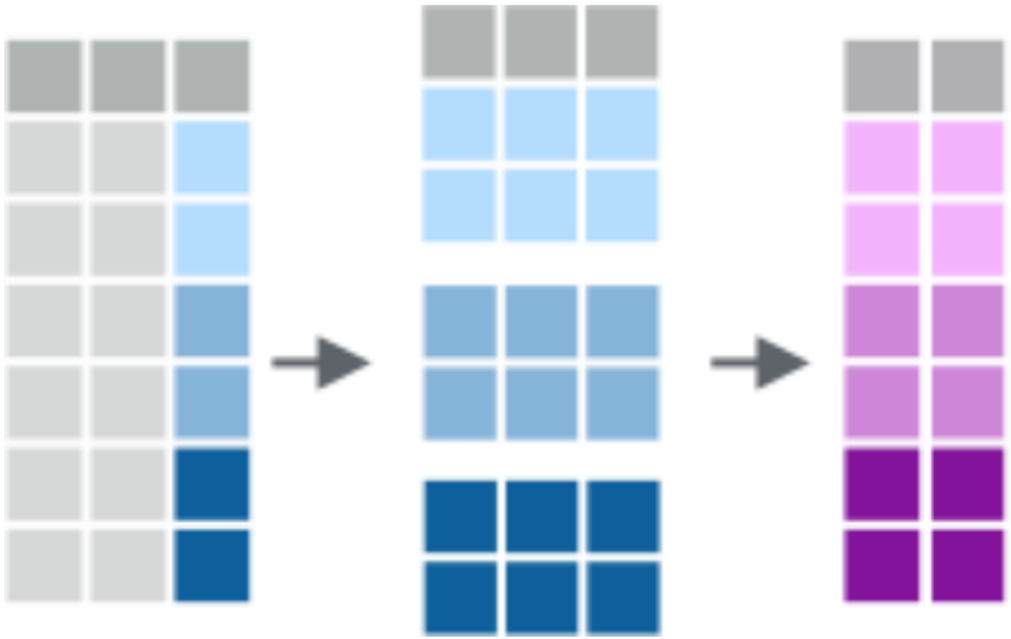
`group_by()` + `summarize()`

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

group_by() + do()



```
mtcars %>%
  group_by(cyl) %>%
  do(func(.))
```