Session III Recap

N.Paterno

7/9/2021

This week we focused on chapters 4 and 5 of R for Data Science.

Chapter 4

To start, we had a brief conversation about naming conventions. For syntax, we want to choose either CamelCase or snake_case. I would recommend snake_case but there is nothing wrong with CamelCase. Some people use (unoffical name) dot.case. I recommend against this strongly as there are some functions written with that notation and we don't want to mix up variable names with function names.

When choosing a name for a data frame or variable, we want to be specific enough to know what the variable is without being too lengthy. For example if we had a dataset on Kobe Bryant's career stats, we wouldn't name the variable "kobe_bryant_points_per_game". Instead we would call the data frame "kobe_career_stats" and the variable "ppg" or "pts_game".

The next part of chapter 4 involves looking at functions. The syntax for a function is `function(input..)` where `input` could be a single value or multiple. For examples well use the `seq` function from base R.

```
seq(1,10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

As we can see, this function prints all numbers from the first input to the second. By default `to` and `from` are set to `1`. Then we can accomplish the same as above using:

```
seq(from = 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Another input or argument for this function is `by`. This will tell R how many numbers to count by. Default is to go by 1 (as above).

```
seq(1, 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```
seq(2, 10, by = 3)
```

```
## [1] 2 5 8
```

Alternatively, we can specify `length.out` which tells R the length of the output vector or how many numbers to print. Then `length.out = 15` tells R to print 15 even spaced numbers between and including `to` and `from`.

```
seq(1, 1000, length.out = 15)
```

```
##  [1]    1.00000   72.35714  143.71429  215.07143  286.42857  357.78571
##  [7]  429.14286  500.50000  571.85714  643.21429  714.57143  785.92857
## [13]  857.28571  928.64286 1000.00000
```

If we are saving something like this in a variable and want to check that it is evaluating correctly, we can wrap the definition inside parenthesis.

```
# Store the sequence in the variable a
a <- seq(1, 1000, length.out = 15)

# Store the sequence in the variable b AND print to console
(b <- seq(1, 1000, length.out = 15))
```

```
##  [1]    1.00000   72.35714  143.71429  215.07143  286.42857  357.78571
##  [7]  429.14286  500.50000  571.85714  643.21429  714.57143  785.92857
## [13]  857.28571  928.64286 1000.00000
```

Chapter 5

The focus of this chapter is data transformations or data wrangling; this is the process of manipulating our data into a certain format that makes summaries and/or plotting a specific characteristic easier. For these examples, we used the `scoobydoo` dataset from the most recent TidyTuesday.

Data wrangling is done largly with `{dplyr}`. The main functions are `filter`, `arrange`, `group_by`, `mutate`, `summarize` and `select`.

```
num_obs <- scoobydoo %>%
  group_by(format) %>%
  summarize(count = n()) %>%
  arrange(-count)

knitr::kable(num_obs)
```

format	count
TV Series	374
TV Series (segmented)	175
Movie	43
Crossover	8
Movie (Theatrical)	3

In the above:

- `group_by(format)` tells R that the manipulation in the following lines should be done for each category as a whole
- `summarize(count = n())` creates a new variable called `count` and condenses the dataframe so that it only has one observation for the `group_by` variable.
- `arrange(-count)` arranges the data by the `count` variable from greatest to least

The `mutate` function behaves very similar to `summarize`. The difference is that it will tack the new variable on to the end of the dataframe, instead of summarizing the data. Note: the example below code below is not displayed because the resulting data frame has 603 rows and 76 columns.

```
num_obs_2 <- scoobydoo %>%
  group_by(format) %>%
  mutate(count = n()) %>%
  arrange(-count)
```

Let's say we wanted to calculate the average imdb score for each season of each show.

```
avg_imdb_rating <- scoobydoo %>%
  filter(format == "TV Series") %>%
  select(c(series_name, season, imdb)) %>%
  group_by(series_name, season) %>%
  mutate(imdb = as.numeric(imdb)) %>%
  summarize(avg_imdb = mean(imdb))
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```
## `summarise()` has grouped output by 'series_name'. You can override using the `.groups` argument.
```

```
knitr::kable(avg_imdb_rating)
```

series_name	season	avg_imdb
A Pup Named Scooby-Doo	1	7.453846
A Pup Named Scooby-Doo	2	7.625000
A Pup Named Scooby-Doo	3	7.050000
A Pup Named Scooby-Doo	4	7.200000
Be Cool, Scooby-Doo!	1	7.538462
Be Cool, Scooby-Doo!	2	7.329630
Lego	Special	6.600000
Night of the Living Doo	Special	7.200000
Scooby-Doo and Guess Who?	1	7.865385
Scooby-Doo and Guess Who?	2	NA
Scooby-Doo and Scrappy-Doo (first series)	1	7.450000
Scooby-Doo Mystery Incorporated	1	8.180769
Scooby-Doo Mystery Incorporated	2	8.407692
Scooby Doo, Where Are You!	1	8.123529
Scooby Doo, Where Are You!	2	8.087500
Shaggy & Scooby-Doo Get a Clue!	1	5.800000
Shaggy & Scooby-Doo Get a Clue!	2	5.407692
The 13 Ghosts of Scooby-Doo	1	7.476923
The New Scooby-Doo Movies	1	7.743750
The New Scooby-Doo Movies	2	7.562500
The New Scooby-Doo Mysteries	1	7.516667
The New Scooby and Scrappy Doo Show	1	6.500000
The Scooby-Doo Project	Special	7.400000
The Scooby-Doo Show	1	7.481250
The Scooby-Doo Show	2	7.687500
The Scooby-Doo Show	3	7.450000
Warner Home Video	Special	6.900000
What's New Scooby-Doo?	1	7.450000
What's New Scooby-Doo?	2	7.407143
What's New Scooby-Doo?	3	7.428571

In the above code:

- `filter(format == "TV Series")` goes through the data and removes rows that have a format other than TV Series. In general, the `filter` function takes in a logical argument and then uses that to parse the dataset for rows where this is true.
- `select(c(series_name, season, imdb))` removes all variables except for those listed. Alternatively, a - in front of the `c(` would *remove* the variables listed and keep all others.
- `mutate(imdb = as.numeric(imdb))` This is the code that I couldn't see during our session but you

were able to figure out!

Next week, we'll focus on creating visualizations based on these manipulations (and others) as well as putting them into an RMarkdown document.