Hacky Hour 2023_11_07: For Loops and Map()

Augustus Pendleton

Introduction

Hello! The purpose of today's lesson is to learn how to apply a function across a list of values in R. On the face of it, this may feel very "codey", but this is actually a very common problem for biologists who are working with multiple experiments or samples. In today's example, we have data saved in multiple spreadsheets which we eventually want to combine and analyze together. We'll learn two methods to help automate reading in these files, without having to manually read each one into R. We'll compare our strategies, and just for fun, make a fun plot to celebrate our success in using this data.

Our Data:

As a rigorous scientist **and** dedicated Swiftie, you've been doing your research.[1] In fact, you've listened to every album Taylor Swift has released, and collected the following data:

1. Track Name
2. Release Dat
3. Danceability
4. Energy
5. Loudness
6. Tempo

[1]Actually, you haven't - this is a subset of data from the taylor package

7. Duration (in milliseconds)

You've listened to each album and meticulously recorded your observations in a comma-separated file (.csv). You can find these data sheets in the "data" folder in this lesson's directory. Let's use the R function `list.files()` to view these.

```
list.files(path = "data") # The path argument specifies which directory to look in
```

```
 [1] "evermore.csv"
 [2] "fearless_taylors_version.csv"
 [3] "fearless.csv"
 [4] "folklore.csv"
 [5] "fonts"
 [6] "lover.csv"
 [7] "midnights.csv"
 [8] "red_taylors_version.csv"
 [9] "red.csv"
[10] "reputation.csv"
[11] "speak_now.csv"
[12] "taylor_swift_covers"
[13] "taylor_swift.csv"
[14] "the_taylor_swift_holiday_collection.csv"
[15] "x1989.csv"
```

As you can see, you've recorded your observations for each album in its own separate file - and there's 13 of them! Let's figure out how to read them into R so we can analyze them.

Reading in a CSV

Our goal is to read in each csv file as a data frame in R. We'll use the `read_csv` function in the `readr` package to do this. `readr` is part of a larger group of packages called the `tidyverse` which work well together; for convenience I usually load them all together.

```r
library(tidyverse) # Load packages in the tidyverse

fearless_df <- read_csv(file = "data/fearless.csv") # Read in a csv file as a data frame

glimpse(fearless_df) # Look at a summary of the rows and columns
```

```
Rows: 19
Columns: 7
$ album_release <date> 2008-11-11, 2008-11-11, 2008-11-11, 2008-11-11, 2008-11~
$ track_name    <chr> "Fearless", "Fifteen", "Love Story", "Hey Stephen", "Whi~
$ danceability  <dbl> 0.598, 0.559, 0.618, 0.843, 0.585, 0.687, 0.498, 0.601, ~
$ energy        <dbl> 0.714, 0.636, 0.736, 0.553, 0.346, 0.771, 0.486, 0.852, ~
$ loudness      <dbl> -4.397, -4.400, -3.937, -7.348, -8.014, -4.424, -7.384, ~
$ tempo         <dbl> 99.979, 95.485, 118.982, 115.997, 92.564, 129.964, 147.9~
$ duration_ms   <dbl> 242000, 294347, 235280, 254320, 234440, 231147, 263987, ~
```

This approach to reading in a csv file isn't too bad, as long as we only have a few files. However, we would need to type the same command 13 times to read in all of our data, which becomes a pain. Plus, if we moved these files to a different location, we would need to edit the path to all 13 files separately. Finally, in bioinformatics you'll often have hundreds if not thousands of files you need to analyze - at this point it would be impossible to type each file name individually. As such, we need a way to automatically read in each file in this directory.

Reading in multiple files using a for loop.

One method is to use a programming tool called a "for loop" to read in each file sequentially. Let's look at the basic syntax of a for loop:

```r
for(x in c(1,2,3)){
  print(x)
```

```
}
```

```
[1] 1
[1] 2
[1] 3
```

There are four main components here:

1. The for() command, which initiates the for loop
2. The name we chose for each element in the loop - here we picked x, but it could be anything
3. The "iterable object", which is the collection of things we want to loop across
4. The functions/code we want to execute on each element, which is enclosed by curly brackets. Here, we print each element to the console

So if I read this code in plain english, it would say something like:

"For each object 'x' in the vector c(1,2,3), print that object"

We can do multiple lines of code within each step of the for loop:

```
for(our_number in c(1,2,3)){
  new_number <- our_number^2
  print(new_number)
}
```

```
[1] 1
[1] 4
[1] 9
```

You can see here that I've used a different variable name, "our_number" to identify our object within the for loop. I can also make new variables ("new_number") within the for loop, and use them in the next line of code.

Now, let's write code to run a for loop which will read in all of our Taylor Swift data sheets

```
our_files <- list.files(path = "data", pattern = "*.csv", full.names = TRUE)


for(file in our_files){
  read_csv(file) # Use the read_csv command for each file
}
```

You'll notice I included some additional arguments in list.files - pattern makes sure we only list csv files, in case there were many different files in the directory, and full.names means I get the full path (from our working directory) for each file, which means we can feed the results to read_csv directly

Okay, we can see from the console that we were reading each file…but there's no dataframes in our environment? That must be because we didn't assign them to an object!

```
taylor_dfs <- for(file in our_files){
  read_csv(file) # Use the read_csv command for each file
}
```

Hmm, it didn't like that either. It looks like R doesn't like assign an entire for loop, which is an active process, into a file.

Maybe we need to do the assignment within the loop itself?

```
for(file in our_files){
  taylor_dfs <- read_csv(file) # Use the read_csv command for each file
}
```

Woohoo! Now we have a dataframe in our environment. However, if we view it, we see that there's only results from one album…1989. Each step of the for loop re-wrote our taylor_dfs object with that step's file. How can we get all of the data in one place?? Here's one method, where we'll take advantage of **indexing** to slot each file's results into a matching location in a list:

```
taylor_dfs_list <- list() # Initialize an empty list, so we have somewhere to put our data

for(index in 1:length(our_files)){ # Instead of looping on each filename in our_files, 'in
  taylor_dfs_list[[index]] <- read_csv(our_files[index]) # For each slot in the list, add
}
```

If you explore our taylor_dfs_list object in the Environment pane, you can see we now have a list of 13 dataframes, all of which have matching columns. There are some downsides to this approach, however. For one, we've don't know which dataframe is from which csv file, or which album. We would need to manually add that data in. Plus. we want to have all of these data in one big dataframe.

Both of these problems are fixable using base R, but there are easier approaches which address them directly using the purrr package. Purrr is designed to make iterating across objects cleaner, clearer, and safer (less chances of invisible errors), and it all revolves around the "map" function. Let's do the same thing we did above, but using map from purrr.

```
library(purrr)

taylor_dfs_map <- map(our_files, read_csv) # Our first argument is our iterable object, ou
```

Wow, with only one line of code, we can see that we produced exactly the same result. We didn't need to initialize an empty list, because map returns a list by default. We still don't know which dataframe is from which csv, however. The nice thing about map is that if our iterable object (our_files) has 'names', those names will automatically be used for the resulting list. Let's add names to our_files, and run the map command again.

```
names(our_files) <- our_files # Assign our file names as the names for the our_files vecto

names(our_files)
```

```
 [1] "data/evermore.csv"
 [2] "data/fearless_taylors_version.csv"
 [3] "data/fearless.csv"
 [4] "data/folklore.csv"
 [5] "data/lover.csv"
 [6] "data/midnights.csv"
 [7] "data/red_taylors_version.csv"
 [8] "data/red.csv"
 [9] "data/reputation.csv"
[10] "data/speak_now.csv"
[11] "data/taylor_swift.csv"
[12] "data/the_taylor_swift_holiday_collection.csv"
[13] "data/x1989.csv"
```

```r
taylor_dfs_map <- map(our_files, read_csv) # read in data again

taylor_dfs_map[1] # Look at the list - notice each element now has a name
```

```
$`data/evermore.csv`
# A tibble: 17 x 7
   album_release track_name       danceability energy loudness tempo duration_ms
   <date>        <chr>                   <dbl>  <dbl>    <dbl> <dbl>       <dbl>
 1 2020-12-11    willow                  0.392  0.574    -9.20  81.1      214707
 2 2020-12-11    champagne probl~        0.462  0.24    -12.1  171.       244000
 3 2020-12-11    gold rush               0.512  0.462   -10.5  112.       185320
 4 2020-12-11    'tis the damn s~        0.575  0.434    -8.19 146.       229840
 5 2020-12-11    tolerate it             0.316  0.361   -10.4   75.0      245440
 6 2020-12-11    no body, no cri~        0.546  0.613    -7.59  79.0      215627
 7 2020-12-11    happiness               0.559  0.334   -10.7  122.       315147
```

```
 8 2020-12-11    dorothea                0.605  0.488    -8.32 120.       225880
 9 2020-12-11    coney island            0.537  0.537   -11.3  108.       275320
10 2020-12-11    ivy                     0.515  0.545    -9.28  88.9      260440
11 2020-12-11    cowboy like me          0.604  0.517    -9.01 128.       275040
12 2020-12-11    long story short        0.546  0.73     -7.70 158.       215920
13 2020-12-11    marjorie                0.535  0.561   -11.6   96.1      257773
14 2020-12-11    closure                 0.689  0.704   -10.8  152.       180653
15 2020-12-11    evermore                0.39   0.27    -10.7  125.       304107
16 2020-12-11    right where you~        0.581  0.619    -6.52 138.       245027
17 2020-12-11    it's time to go         0.592  0.41    -12.4  152.       254640
```

Now we can see that each element in our list is named with the csv file it came from. However, the names are a little ugly - they're all in the data folder, we have special characters like "/", and we know that they're all csv files. Below, I'm going to make cleaner names with two functions - basename() and str_remove(), so that our names in the end are nicer. str_remove() is from the stringr package, which is loaded with the tidyverse

```
base_names <- basename(our_files) # Get the base name (furthest to the right in the path)


base_names
```

```
[1] "evermore.csv"
[2] "fearless_taylors_version.csv"
[3] "fearless.csv"
[4] "folklore.csv"
[5] "lover.csv"
[6] "midnights.csv"
[7] "red_taylors_version.csv"
[8] "red.csv"
[9] "reputation.csv"
```

```
[10] "speak_now.csv"
[11] "taylor_swift.csv"
[12] "the_taylor_swift_holiday_collection.csv"
[13] "x1989.csv"
```

```
clean_names <- str_remove(base_names, pattern = ".csv") # Remove ".csv" from each name

clean_names
```

```
 [1] "evermore"                          "fearless_taylors_version"
 [3] "fearless"                          "folklore"
 [5] "lover"                             "midnights"
 [7] "red_taylors_version"               "red"
 [9] "reputation"                        "speak_now"
[11] "taylor_swift"                      "the_taylor_swift_holiday_collection"
[13] "x1989"
```

```
names(our_files) <- clean_names # Assign our clean names to the our_files vector

taylor_dfs_map <- map(our_files, read_csv) # Read in data again

taylor_dfs_map[1]
```

```
$evermore
# A tibble: 17 x 7
   album_release track_name       danceability energy loudness tempo duration_ms
   <date>        <chr>                   <dbl>  <dbl>    <dbl> <dbl>       <dbl>
 1 2020-12-11    willow                  0.392  0.574    -9.20  81.1      214707
 2 2020-12-11    champagne probl~        0.462  0.24    -12.1  171.       244000
 3 2020-12-11    gold rush               0.512  0.462   -10.5  112.       185320
```

```
 4 2020-12-11    'tis the damn s~        0.575  0.434    -8.19 146.       229840
 5 2020-12-11    tolerate it             0.316  0.361   -10.4   75.0      245440
 6 2020-12-11    no body, no cri~        0.546  0.613    -7.59  79.0      215627
 7 2020-12-11    happiness               0.559  0.334   -10.7  122.       315147
 8 2020-12-11    dorothea                0.605  0.488    -8.32 120.       225880
 9 2020-12-11    coney island            0.537  0.537   -11.3  108.       275320
10 2020-12-11    ivy                     0.515  0.545    -9.28  88.9      260440
11 2020-12-11    cowboy like me          0.604  0.517    -9.01 128.       275040
12 2020-12-11    long story short        0.546  0.73     -7.70 158.       215920
13 2020-12-11    marjorie                0.535  0.561   -11.6   96.1      257773
14 2020-12-11    closure                 0.689  0.704   -10.8  152.       180653
15 2020-12-11    evermore                0.39   0.27    -10.7  125.       304107
16 2020-12-11    right where you~        0.581  0.619    -6.52 138.       245027
17 2020-12-11    it's time to go         0.592  0.41    -12.4  152.       254640
```

Woohoo! Now this is looking much nicer. However, we still want to get all of our data into a dataframe! This is where one of my favorite functions comes in - map_dfr(), also from the purrr package. This will read in our data and automatically add each new dataframe onto the bottom of the last one - combining all of our separate dataframes into one!

```
taylor_full_df <- map_dfr(our_files, read_csv, .id = "Album") # .id argument is the column


glimpse(taylor_full_df)
```

```
Rows: 241
Columns: 8
$ Album         <chr> "evermore", "evermore", "evermore", "evermore", "evermor~
$ album_release <date> 2020-12-11, 2020-12-11, 2020-12-11, 2020-12-11, 2020-12~
$ track_name    <chr> "willow", "champagne problems", "gold rush", "'tis the d~
$ danceability  <dbl> 0.392, 0.462, 0.512, 0.575, 0.316, 0.546, 0.559, 0.605, ~
```

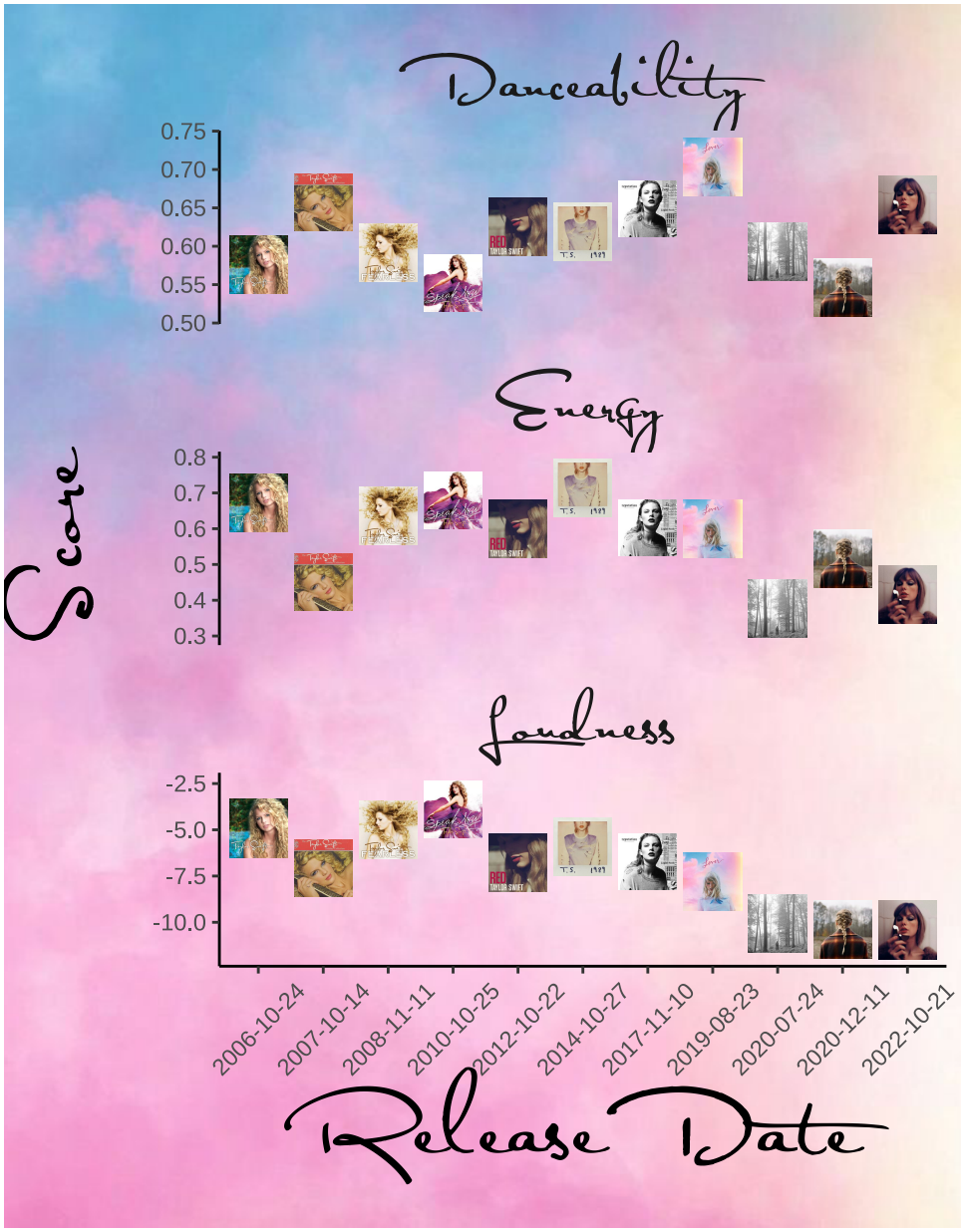
```
$ energy        <dbl> 0.574, 0.240, 0.462, 0.434, 0.361, 0.613, 0.334, 0.488, ~
$ loudness      <dbl> -9.195, -12.077, -10.491, -8.193, -10.381, -7.589, -10.7~
$ tempo         <dbl> 81.112, 171.319, 112.050, 145.916, 74.952, 79.015, 122.0~
$ duration_ms   <dbl> 214707, 244000, 185320, 229840, 245440, 215627, 315147, ~
```

Now we see we've read in all the data, combined them into a large dataframe, we we have a new column called "Album" which tracks which album each song came from based on the names of the our_files vector.

That's all we'll cover concerning for loops and map today - hope you enjoyed yourself! Now go ahead and analyze this data however you want - I made a plot below for fun!

```
library(ggimage) # Will need for the images as points and background
library(showtext) # Will need for our custom fonts

font_add(family = "satisfaction", regular = "data/fonts/Satisfaction.ttf") # Add a new fon
showtext_auto() # Make sure ggplot knows to use this font!
showtext_opts(dpi = 300) # This stops the font sizes getting wonky when saving

plot <- taylor_full_df %>%
  filter(!str_detect(Album, "taylors_version")) %>% # I don't want repeat socres
  pivot_longer(!c(Album, album_release, track_name), names_to = "Metric", values_to = "Sco
  filter(Metric %in% c("danceability", "energy", "loudness")) %>% # Let's just plot 3 of t
  group_by(Album, album_release, Metric) %>% # Let's calculate median values for each metr
  summarize(across(where(is.numeric), median, na.rm = TRUE)) %>% # See last comment - find
  ungroup() %>%
  mutate(cover = paste0("data/taylor_swift_covers/", Album,".jpg"), # Make a new column wh
         album_release = factor(album_release), # I like them spaced out regularly instead
         Metric = factor(Metric, levels = c("danceability", "energy", "loudness"), labels
  ggplot(aes(x = album_release, y = Score)) + # Set our x and y axes
```

```
  facet_wrap(~Metric, ncol = 1, scales = "free_y") + # Facet by Metrix, stacked in one col
  scale_y_continuous(expand = expansion(mult = 0.3)) +  # Give some space on the y axis so
  geom_image(aes(image = cover), size = .3)+  # Add in images as our data points
  labs(x = "Release Date", y = "Score") +  # Relabel axes
  theme_classic() + # Use a cleaner theme
  theme(strip.text = element_text(size = 20, family = "satisfaction"), # Use our custom fo
        strip.background = element_rect(color = "transparent", fill = "transparent"), # Ma
        axis.title = element_text(size = 30, family = "satisfaction"), # Use our custom fo
        axis.text.x = element_text(angle = 45, vjust = .5)) # Rotate dates so they fit bet

final_plot <- ggbackground(plot, "data/taylor_swift_covers/lover_background.jpg") # Add th

ggsave(final_plot, filename = "outputs/Taylor_Plot.png", width = 5, height = 6.4, units =

final_plot
```


Danceability
0.75
0.70
0.65
0.60
0.55
0.50
Energy
Score
0.8
0.7
0.6
0.5
0.4
0.3
Loudness
-2.5
-5.0
-7.5
-10.0
2006-10-24
2007-10-14
2008-11-11
2010-10-25
2012-10-22
2014-10-27
2017-11-10
2019-08-23
2020-07-24
2020-12-11
2022-10-21
Release Date