day_2

Cleo Falvey & Filip Stefanovic

7/31/2020

Welcome back!

It's been a week since we last met. And I am so thrilled to see y'all again because today we are diving into the Tidyverse!

Remember, before we do anything, we need to call our libraries!

We shouldn't have to install anything new! If R asks you to install the package again, something has gone wrong!

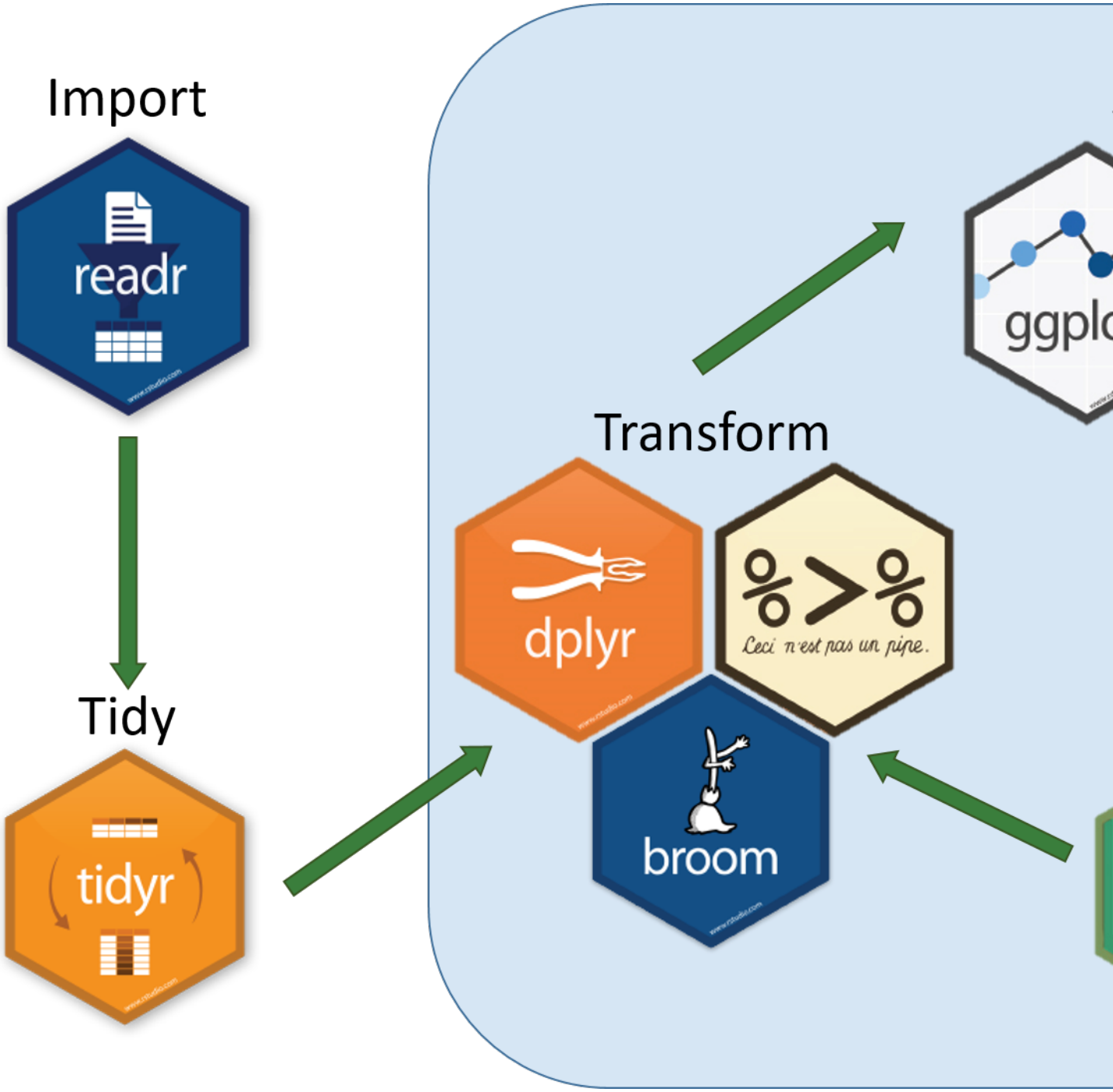
```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tidyverse

What's the Tidyverse? It's a really easy and clean workflow in R. We did a little bit of it yesterday when we read the .csv files in, but today, we are going to do a little bit more!

```
knitr::include_graphics("tidyverse-package-workflow.png")
```



readr

Reading Data

We read data in with *read_csv()*. There's a version with a period (*read.csv*) but it is less good, and sometimes does finicky things when reading in data. That's why we always use the Tidy option!

Today's dataset is about Spotify songs from 2010-2019. It lives on Kaggle here. It's also in your working

directory as a file called spotify.csv.

Don't forget! R should have imported the data frame as a Tibble (a very cute word) that just refers to a tidy data frame.

```
spotify <- read_csv("spotify.csv")
```

```
## Parsed with column specification:
## cols(
##   title = col_character(),
##   artist = col_character(),
##   genre = col_character(),
##   year = col_double(),
##   bpm = col_double(),
##   energy = col_double(),
##   dance = col_double(),
##   dB = col_double(),
##   live = col_double(),
##   valence = col_double(),
##   duration = col_double(),
##   acoustic = col_double(),
##   speaking = col_double(),
##   popularity = col_double()
## )
```

Don't forget, we always want to know what our data is like.

```
head(spotify)
```

```
## # A tibble: 6 x 14
##   title artist genre  year   bpm energy dance    dB  live valence duration
##   <chr> <chr>  <chr> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
## 1 Hey,~ Train  neo ~  2010    97     89    67    -4     8      80      217
## 2 Love~ Eminem detr~  2010    87     93    75    -5    52      64      263
## 3 TiK ~ Kesha  danc~  2010   120     84    76    -3    29      71      200
## 4 Bad ~ Lady ~ danc~  2010   119     92    70    -4     8      71      295
## 5 Just~ Bruno~ pop    2010   109     84    64    -5     9      43      221
## 6 Baby  Justi~ cana~  2010    65     86    73    -5    11      54      214
## # ... with 3 more variables: acoustic <dbl>, speaking <dbl>, popularity <dbl>
```

```
class(spotify) # this command tells us what type of data frame this object is. If all's well, it should
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

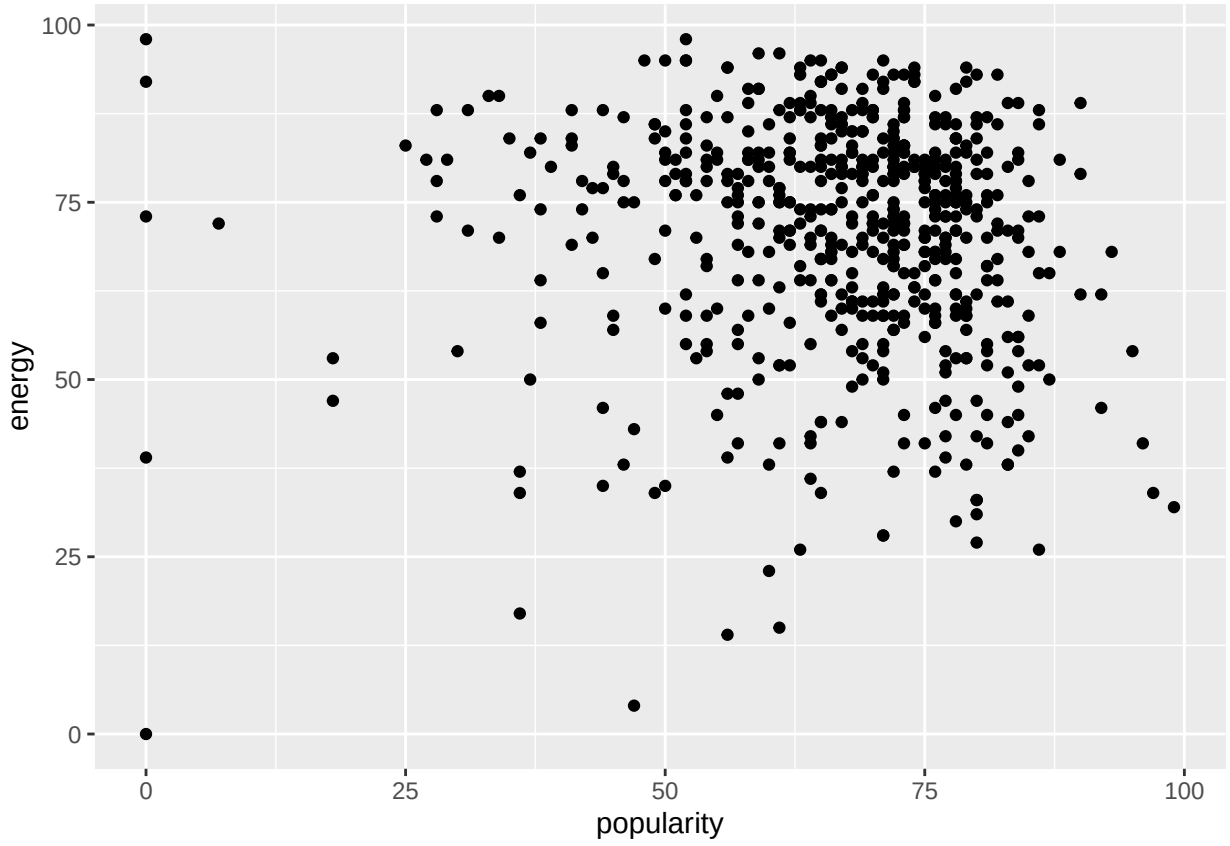
A lot of the column titles are very self-explanatory. However, some of the metrics may only make sense to the most well-attuned music lover. Which is not me, so I read the documentation attached to the dataset.

- Energy: How energetic a song is
- dB: How loud the song is
- Live: How likely that song was to be recorded live or not
- Valence: How positive a song is
- Duration: How long the song is
- Acoustic: How acoustic the song is
- Speaking: How many lyrics the song has
- Popularity: How popular the song is

ggplot2

Visualizing Data

```
ggplot(spotify, aes(popularity, energy)) +
  geom_point()
```



dplyr

dplyr and tidyr are amazing packages. It's a true fact. But the other true fact is that they can be kind of hard to get the hang of. While I was making this presentation, I StackOverflowed at least six things.

In order to use dplyr, we need to think about data frames as objects; they can be changed, mutated, filtered, and summarized.

Piping Data

%>%

The above symbol is what's known as a pipe in dplyr. The pipe signifies that you are taking the data frame (our example data frame is *spotify*) and putting the whole data frame through a series of functions to get a whole new data frame.

```
# makes new data frame of the top six rows of the spotify dataset.
short_spotify <- spotify %>% # take spotify data frame and pipe
  head() # pipe it through the head function to get only the top six rows.

short_spotify
```

```
## # A tibble: 6 x 14
##   title artist genre  year   bpm energy dance    dB  live valence duration
##   <chr> <chr>  <chr> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
## 1 Hey,~ Train  neo ~  2010    97     89    67    -4     8      80      217
## 2 Love~ Eminem detr~  2010    87     93    75    -5    52      64      263
## 3 TiK ~ Kesha  danc~  2010   120     84    76    -3    29      71      200
## 4 Bad ~ Lady ~ danc~  2010   119     92    70    -4     8      71      295
## 5 Just~ Bruno~ pop    2010   109     84    64    -5     9      43      221
## 6 Baby  Justi~ cana~  2010    65     86    73    -5    11      54      214
## # ... with 3 more variables: acoustic <dbl>, speaking <dbl>, popularity <dbl>
```

```
# Now, you try! Make a dataset called short_spotify2, which is the same thing as above except uses the
```

Filtering & Selecting Data

Let's take a look at our Spotify data. A lot of popular songs are by Bruno Mars. How many are there, and what are they? To figure this out, we're going to use *filter()*, which looks for matching rows in the dataset.

```
bruno_mars <- spotify %>%
  filter(artist=="Bruno Mars") #artist name must be equal to Bruno Mars. We use the double equals sign
head(bruno_mars)
```

```
## # A tibble: 6 x 14
##   title artist genre  year   bpm energy dance    dB  live valence duration
##   <chr> <chr>  <chr> <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
## 1 Just~ Bruno~ pop    2010   109     84    64    -5     9      43      221
## 2 Marr~ Bruno~ pop    2010   145     83    62    -5    10      48      230
## 3 Just~ Bruno~ pop    2011   109     84    64    -5     9      43      221
## 4 Gren~ Bruno~ pop    2011   110     56    71    -7    12      23      223
## 5 Marr~ Bruno~ pop    2011   145     83    62    -5    10      48      230
## 6 Lock~ Bruno~ pop    2012   144     70    73    -4    31      87      233
## # ... with 3 more variables: acoustic <dbl>, speaking <dbl>, popularity <dbl>
```

We took our spotify data and piped it to the *filter()* function in dplyr. We wanted only the songs that were by Bruno Mars. We need to use the double equals sign (==), because if you only use one equals sign (=), R throws an error. You can try it!

You can also use different computer logic symbols with *filter()*.

- == means EQUALS TO
- != means NOT EQUALS TO
- && means AND
- || means OR

```
# Now, you try! Pipe the Spotify dataset to a make a tibble of only songs by Katy Perry called katy_per

# Extra challenge. Pipe the Spotify dataset to a tibble by songs only by enrique iglesias OR lady gaga.
```

There are fifteen Spotify variables. What if we only wanted to see only the titles and the authors?

To answer that question, we'll use select, which selects different columns of your data. Note the use of the *c()* function to concatenate which functions.

```
title_artist <- spotify %>%
  select(c("title","artist"))
head(title_artist)
```

```
## # A tibble: 6 x 2
##   title                artist
##   <chr>                <chr>
## 1 Hey, Soul Sister     Train
## 2 Love The Way You Lie Eminem
## 3 TiK ToK              Kesha
## 4 Bad Romance          Lady Gaga
## 5 Just the Way You Are Bruno Mars
## 6 Baby                 Justin Bieber
```

Transforming Data

Sometimes, we need to transform data. For example, the Spotify data has a column called duration, but it's in seconds. What if we wanted to have a column for minutes and seconds (with seconds as a remainder?)

```
spotify_durations <- spotify %>%
  select(title, duration) %>%
  mutate(minutes = duration %/% 60) %>%
  mutate(seconds = duration %% 60)
head(spotify_durations)
```

```
## # A tibble: 6 x 4
##   title                duration minutes seconds
##   <chr>                   <dbl>   <dbl>   <dbl>
## 1 Hey, Soul Sister          217       3      37
## 2 Love The Way You Lie      263       4      23
## 3 TiK ToK                   200       3      20
## 4 Bad Romance               295       4      55
## 5 Just the Way You Are      221       3      41
## 6 Baby                      214       3      34
```

Chaining multiple commands together

Which artist has the most number of popular songs on Spotify? Could it be ... unfortunately... *Justin Bieber* ???

To answer this question, I took the Spotify data and sent it through a series of dplyr commands. I took the Spotify data and grouped it by the artist and counted the instances of each artist. Then, I arranged it in descending order.

```r
popular_artists <- spotify %>% # take spotify dataframe
  group_by(artist) %>% # group by artist
  count() %>% # count each instance of artist
  arrange(desc(n)) # arrange in descending order by n
head(popular_artists)
```

```
## # A tibble: 6 x 2
## # Groups:   artist [6]
##   artist            n
##   <chr>         <int>
## 1 Katy Perry       17
## 2 Justin Bieber    16
## 3 Maroon 5         15
## 4 Rihanna          15
## 5 Lady Gaga        14
## 6 Bruno Mars       13
```

What if my data is not in the right format?

That happens sometimes! For that, we use *pivot_longer()* and *pivot_wider()*. I cover how to use those functions in the bonus material!

Miscellaneous Takeaways: Day 2

- When I make an R script, I try to organize my chunks of code. The first chunk is always the libraries you need to use in the script. The second chunk is always reading in the data. The third (and sometimes fourth, fifth, sixth) is data manipulation. After that I put all the code you need to make the models and the graphs.
- *dplyr* can take a little getting used to. Again, I still find myself Googling lots of "simple" things. Reading the documentation can also help. There is a joke in the software development world regarding whether you should take 30 seconds to read the documentation or literally hours to debug. It's your choice (I don't always even choose correctly.)
-