Analyzing Social Media Data

Francesca Giannetti

10/13/2020

R and RStudio

R is the programming language. RStudio is an integrated development environment (IDE) that makes scripting in R much easier. Both are free and open source software. If you'd like to continue experimenting with R and RStudio, but you'd rather not install them on your personal machine, you can instead create an RStudio Cloud account at https://rstudio.cloud.

When you launch RStudio, you'll notice the default panes:

- Source, or where your R scripts are (upper left)
- Console (lower left, with the `>` prompt)
- Environment/history (tabbed, in upper right)
- Files/plots/packages/help (tabbed, in lower right).

Values

The simplest kind of expression is a value. A value is usually a number or a character string.

Click the green arrow to run the following code chunk; or, place your cursor next to the value and click Ctrl + enter (PC) or Cmd + return (Mac) to get R to evaluate the line of code. As you will see, R simply parrots the value back to you.

```r
6
```

Variables

A variable is a holder for a value.

```r
msg <- "Howdie!"
print(msg)
print("msg") # what do the quotes change?
```

Assignment

In R, `<-` stores a value under a name which you can refer to later. In the example above, we assigned the value "Howdie!" to the variable `msg`. What do you think will happen in this example?

```r
a <- 2
b <- 3
a <- b
print(a)
```

Functions

R has tons of built in functions that map inputs to outputs. Here's a simple example using some fake data about the price of LPs and the `sum()` and `c()` functions. To get help on these functions, type `?sum()` or `?c()` at the console. If we wanted to find the average price of the LPs, we could try the `mean()` or `median()` functions.

```
lp_prices <- c(8.00, 4.00, 12.00, 24.99, 22.50, 19.95)

sum(lp_prices)
```

Memory management

R reads data into your computer's memory in order to manipulate it. Given that the RAM on computers is a finite thing, it makes sense to clean out your environment from time to time.

```
rm(b) # remove the object `b`

rm(list = ls()) # remove everything in your environment
```

Loading Data

In part 1 of this workshop on collecting Twitter data, we learned about TAGS and rtweet. TAGS supports queries by search strings (words, account names, and hashtags), and rtweet adds a few additional methods, including user timelines and latitude and longitude coordinates.

In today's workshop (part 2), we'll load two datasets gathered using each of these tools. The queries used the generate them are copied below for reference. If you want to allow your TAGS archive to continue updating over time so that you can periodically rerun your analyses in R, I show a method to do that below.[1] Conversely, we will load the rtweet archive as a CSV file.

The resulting `covid_tweets` and `quarantine_tweets` objects in your environment are what are called data frames in R. A data frame is a compound data type composed of variables (column headers) and observations (rows). Data frames can hold variables of different kinds, such as character data (the text of tweets), quantitative data (retweet counts), and categorical information (en, es, fr). Clicking on those data frames will open a table view in RStudio allowing you to scan and filter columns and rows.

Review of Tweet object

For reference, I include links to Twitter Developer's "Introduction to Tweet JSON" and the "Tweet Data Dictionary" for the types and definitions of field names. Having a familiarity with these fields will help you to develop and refine your research questions. For the purposes of this workshop, I will draw your attention to specific fields.

Data summaries

One of the first things you may notice is that the archive collected with rtweet has many more columns than the TAGS archive. This is a product of the fact that rtweet simply collects more data from the Twitter REST API, but also because rtweet's `write_as_csv()` function, which I used to write my Twitter archives to file, flattens some of the JSON structure into additional columns.

[1] I'm grateful to Kris Shaffer for showing me how to do this in https://pushpullfork.com/mining-twitter-data-tidy-text-tags/.

R has a handful of built-in functions that can provide us with additional information about our datasets. With `str()`, R displays the structure of an object, including data types (int, chr, date). Another helpful function to retrieve just the headers of your dataset: `names()`. Since we have so much character data, the `summary()` function is a little less helpful, but it does provide summary statistics for the numeric data.

```
str(covid_tweets) # structure, including data types

names(quarantine_tweets) # just the header names

summary(covid_tweets) # mean, median, 25th and 75th quartiles, min, max
```

Subsetting

An R data frame is a special case of a list, which is used to hold just about anything. How does one access parts of that list? Choose an element or elements from a list with `[]`. Sequences are denoted with a colon (:).

```
covid_tweets$text[1] # the first tweet

covid_tweets$text[2:11] # tweets 2 through 11
```

Text Analysis Examples

We'll do a bit of show-and-tell to see the sorts of text analysis that are possible with Twitter data before returning to exercises with dplyr and ggplot2, two very useful R libraries for manipulating and visualizing data. The following examples are inspired by code written by Julia Silge and David Robinson in *Text Mining with R: A Tidy Approach* and by Kris Shaffer in "Mining Twitter data with R, TidyText, and TAGS".

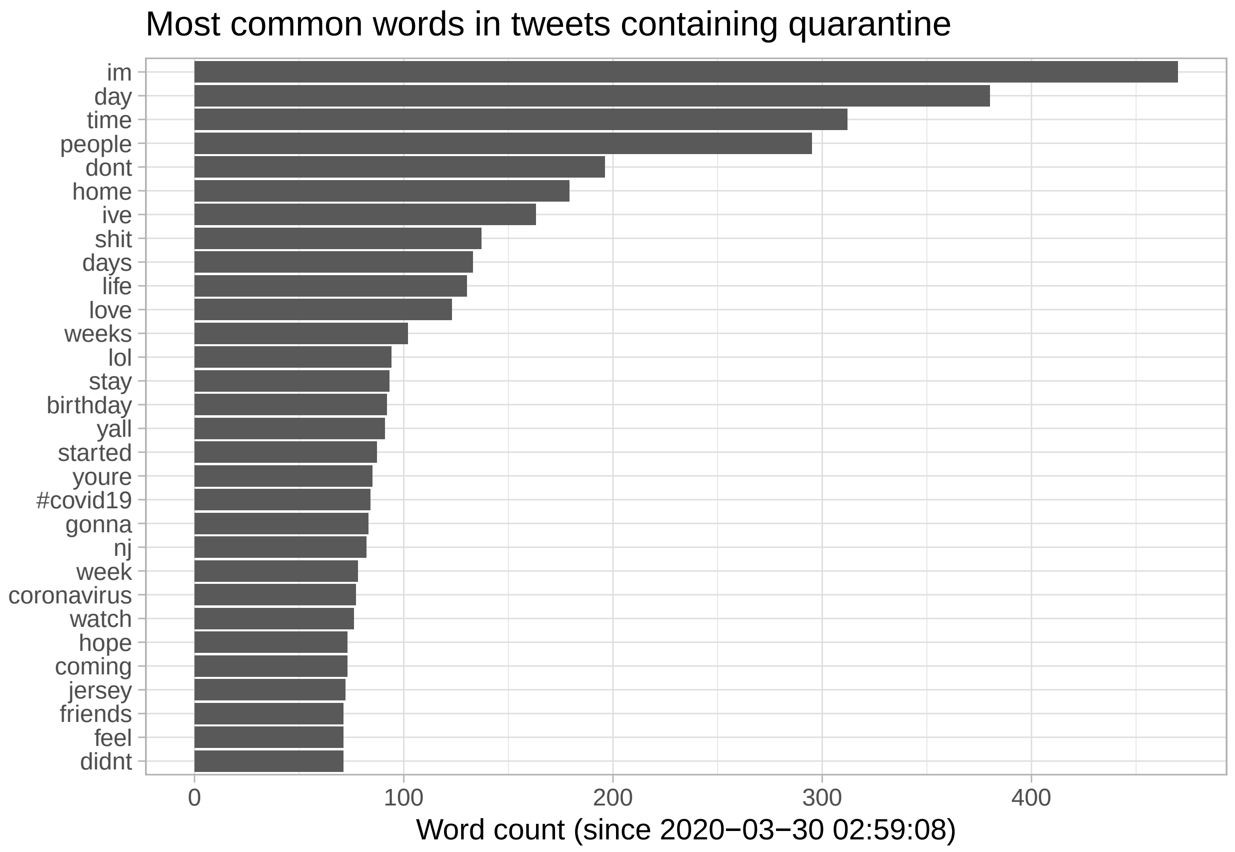
First, we create a regular expression (`remove_reg`) to remove some special characters like ampersands. Then we filter out retweets to analyze only the original tweets in this dataset. Next, we tokenize or split the tweets into words, and create a data frame that places each word in its own row together with the accompanying tweet metadata. Lastly, we'll run another filter to remove some common stop words like a, an, the, of, and numbers.

```
remove_reg <- "&amp;|&lt;|&gt;"
tidy_quarantine_tweets <- quarantine_tweets %>%
  filter(!str_detect(is_retweet, "^TRUE")) %>%
  mutate(text = str_remove_all(text, remove_reg)) %>%
  unnest_tokens(word, text, token = "tweets") %>%
  filter(!word %in% stop_words$word,
         str_detect(word, "[a-z]"))
```

```
## Using `to_lower = TRUE` with `token = 'tweets'` may not preserve URLs.
```

Next, we find the most common words in this Twitter archive. We'll pass another filter to remove the word "quarantine" since that was how this dataset was constructed to begin with and it won't be meaningful here. Next, we reorder the words by frequency and use ggplot2 to visualize them as a bar plot.

```
tidy_quarantine_tweets %>%
  count(word, sort=TRUE) %>%
  filter(n > 70, word != 'quarantine', word !='#quarantine') %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x=word, y=n)) +
  geom_bar(stat = 'identity') +
  labs(x = NULL, y = paste('Word count (since ', min(tidy_quarantine_tweets$created_at), ')', sep = "")
  theme_light() +
  coord_flip()
```



Now let's take a look to see if there are any changes in word use over time. Since the `quarantine_tweets` dataset spans several months, we are going to create a new time variable that aggregates tweets by the week in which they were posted. We use `floor_date()` from lubridate to do this.

After deciding to bin tweets by week, we count how many times a word gets used in each bin. We then use `filter()` to keep words that have been used a minimum number of times (50, in this case).

```
words_by_time <- tidy_quarantine_tweets %>%
  filter(!str_detect(word, "quarantine")) %>%
  mutate(time_floor = floor_date(created_at, unit = "week")) %>%
  count(time_floor, word) %>%
  group_by(time_floor) %>%
  mutate(time_total = sum(n)) %>%
  group_by(word) %>%
  mutate(word_total = sum(n)) %>%
  ungroup() %>%
  rename(count = n) %>%
  filter(word_total > 50)
```

Here, we're using `nest()` from tidyr to make a data frame with a list column that contains little miniature data frames for each word.

In the following line of code, we use `map()` from purrr to apply a modeling procedure to each of those little data frames inside our big data frame. Since this is count data, we will use `glm()` (generalized linear model) with `family="binomial"` for modeling.

The next step is to use `map()` and `tidy()` from the broom package to pull out the slopes for each of these models and find the important ones. We are comparing many slopes here and some of them are not statistically significant, so let's apply an adjustment to the p-values for multiple comparisons.

Finally, we find the top slopes by filtering for the words that have changed in frequency at a moderately significant level. We visualize our results using a smoothed line plot (`geom_smooth()`) showing frequency over time.

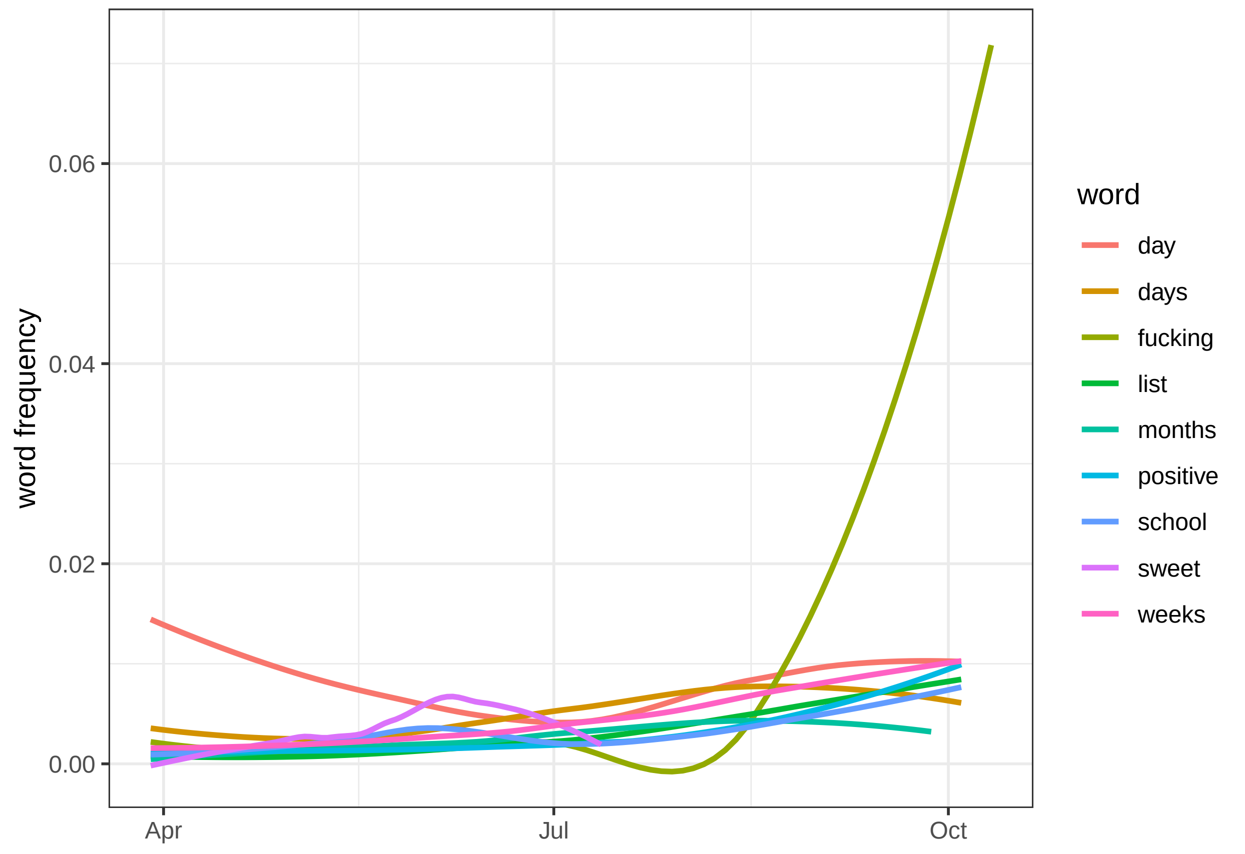
```
nested_data <- words_by_time %>% nest(data = c(time_floor, count, time_total, word_total))

nested_models <- nested_data %>%
  mutate(models = map(data, ~ glm(cbind(count, time_total) ~ time_floor, ., family = "binomial")))

slopes <- nested_models %>%
  mutate(models = map(models, tidy)) %>%
  unnest(cols = c(models)) %>%
  filter(term == "time_floor") %>%
  mutate(adjusted.p.value = p.adjust(p.value))

top_slopes <- slopes %>%
  filter(adjusted.p.value < 0.05)

words_by_time %>%
  inner_join(top_slopes, by = "word") %>%
  filter(!str_detect(word, "nj|covid")) %>% # removing some less interesting slopes
  ggplot(aes(x=time_floor, y= count/time_total, color = word)) +
  geom_smooth(se = FALSE) +
  labs(x = NULL, y = "word frequency") +
  theme_bw()
```

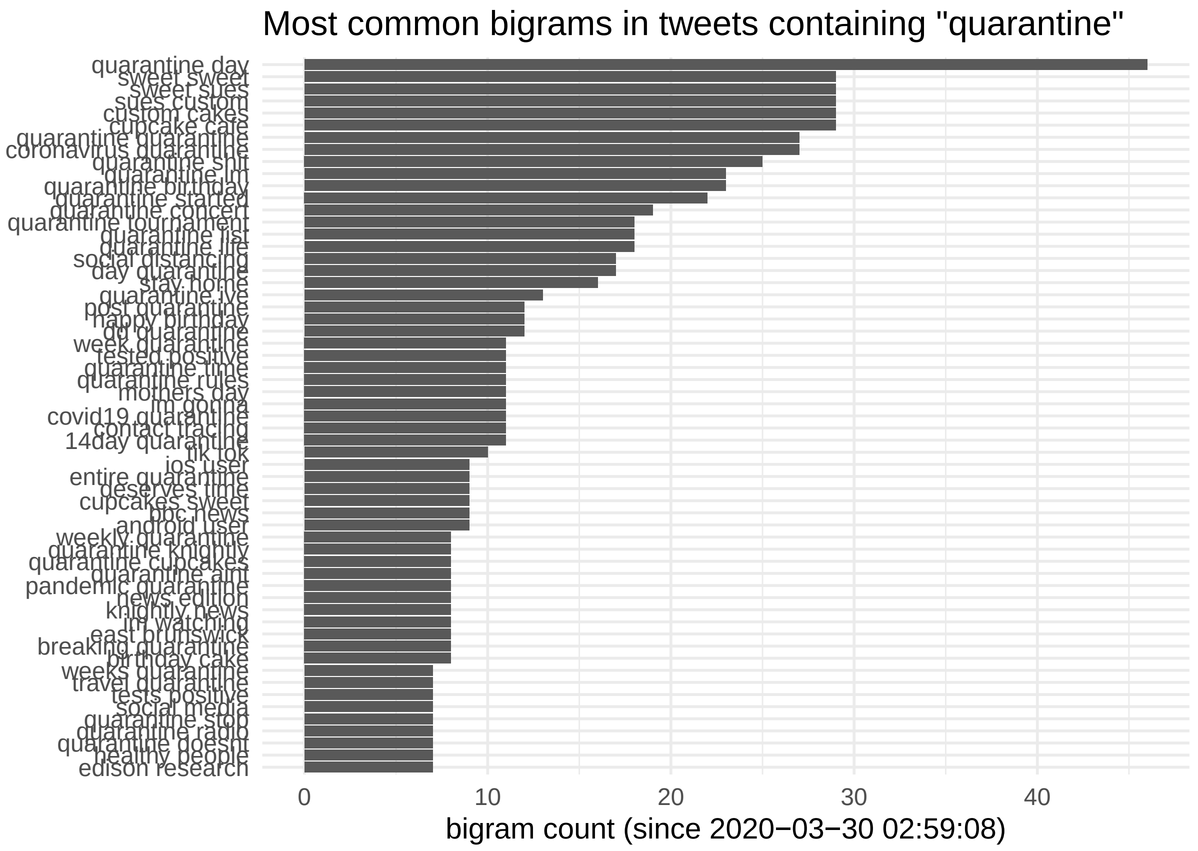


Instead of word frequencies (unigrams), let's take a look at bigrams or two-word phrases to see if they show us anything more instructive about our dataset. We'll tokenize our tweets as before, and then use dplyr's `lead()` fuction to append the following word to each word, and then `unite()` the two into a bigram (but only if they both belong to the same tweet).

```r
tidy_quarantine_bigrams <- quarantine_tweets %>%
  filter(!str_detect(is_retweet, "^TRUE")) %>%
  mutate(text = str_remove_all(text, remove_reg)) %>%
  unnest_tokens(word, text, token = "tweets") %>%
  mutate(next_word = lead(word)) %>%
  filter(!word %in% stop_words$word, # remove stop words
         !next_word %in% stop_words$word, # remove stop words
         substr(word, 1, 1) != '@', # remove user handles to protect privacy
         substr(next_word, 1, 1) != '@', # remove user handles to protect privacy
         substr(word, 1, 1) != '#', # remove hashtags
         substr(next_word, 1, 1) != '#',
         str_detect(word, "[a-z]"), # remove words containing ony numbers or symbols
         str_detect(next_word, "[a-z]")) %>% # remove words containing ony numbers or symbols
  filter(status_id == lead(status_id)) %>% # needed to ensure bigrams to cross from one tweet into the
  unite(bigram, word, next_word, sep = ' ') %>%
  select(bigram, screen_name, created_at, status_id, followers_count, friends_count, location)

p1 <- tidy_quarantine_bigrams %>%
  count(bigram, sort=TRUE) %>%
  filter(n >= 7) %>%
  mutate(bigram = reorder(bigram, n)) %>%
  ggplot(aes(x=bigram, y=n)) +
  geom_bar(stat = 'identity') +
  labs(x = '', y = paste('bigram count (since ', min(quarantine_tweets$created_at), ')', sep = ''), tit
  theme_minimal() +
  coord_flip()

p1
```

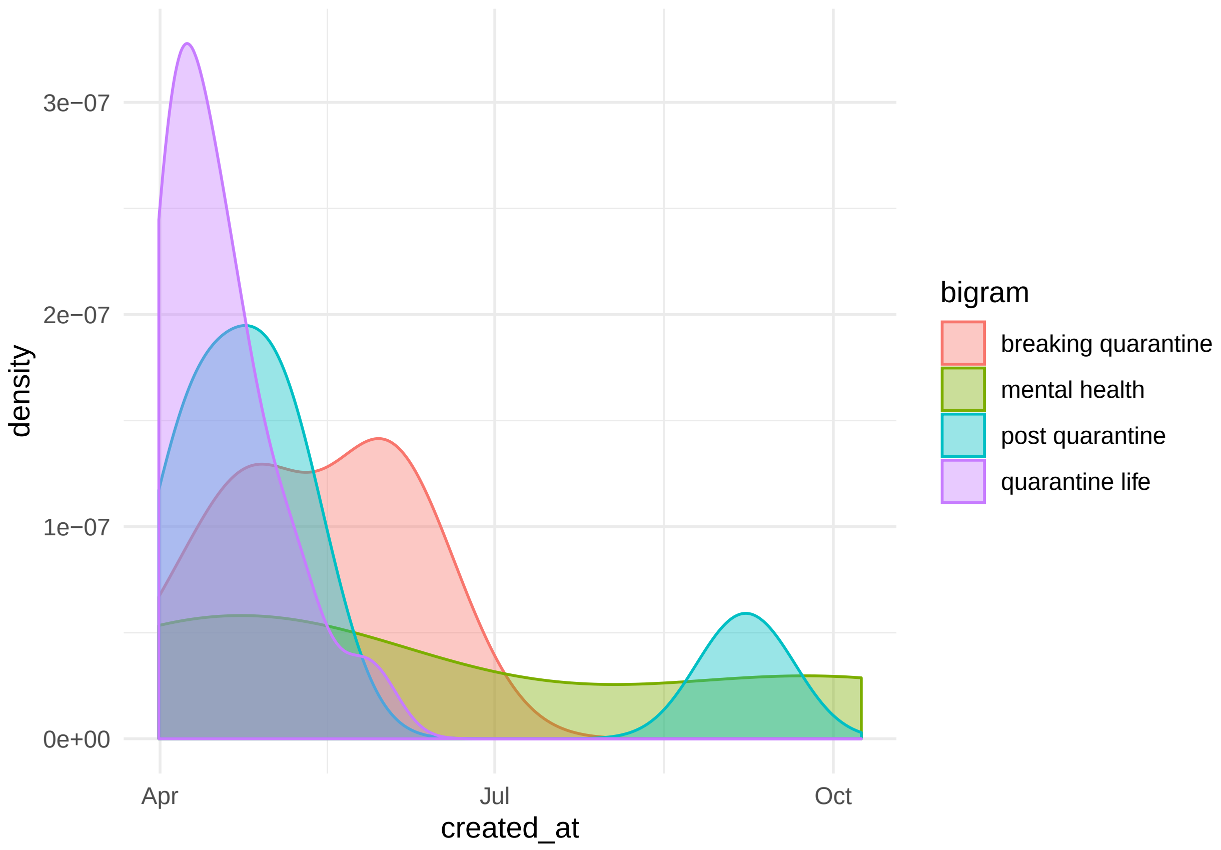


Since there's a lot going on with our bigrams as well, I'm going to show how one might subset for only those bigrams of interest to us. Cherry picking our data is perhaps not the best in terms of statistical significance, but permissable inasmuch as not all these bigrams will be equally interesting. First we will create a mini-object with a handful of the interesting bigrams. Then we will use the `subset()` fuction inside our ggplot2 call to plot only those bigrams that match our interesting bigrams mini-object. Here, we use a density plot (`geom_density()`) to estimate the underlying distribution.

```
bigrams <- c("post quarantine", "quarantine life", "mental health", "breaking quarantine")

p2 <- ggplot(data = subset(tidy_quarantine_bigrams, subset = bigram %in% bigrams),
            mapping = aes(x = created_at, fill = bigram, color = bigram))

p2 + geom_density(alpha=0.4) + theme_minimal()
```



Dplyr: Summary

The next several functions we'll examine are data manipulation verbs that belong to the `dplyr` package. How about some basic descriptive statistics? We can select which columns we want by putting the name of the column in the `select()` function. Here we pick two columns.

Dplyr imports the pipe operator `%>%` from the magrittr package. This operator allows you to pipe the output from one function to the input of another function. It is truly a girl's best friend.

`View` invokes a spreadsheet-style data viewer. Very handy for debugging your work!

```r
covid_tweets %>%
  select(from_user, created_at) %>% View
```

Dplyr: Arrange

We can sort them by number of followers with `arrange()`. We'll add the `desc()` function, since it's more likely we're interested in the users with the most followers.

What do you notice about the repeated `from_user` values?

```r
covid_tweets %>%
  select(from_user, user_followers_count) %>%
  arrange(desc(user_followers_count)) %>% View
```

Question #1: Using the `covid_tweets` dataset, how would you select the Twitter usernames and their number of friends, and arrange them in descending order of number of friends? What is the difference between a follower and a friend? Check back with the User Data Dictionary if necessary.

Dplyr: Group by and summarize

Notice that `arrange()` sorted our entire dataframe in ascending (or descending) order. What if we wanted to calculate the total number of tweets by user account name?

We can solve this kind of problem with what Hadley Wickham calls the "split-apply-combine" pattern of data analysis. Think of it this way. First we can *split* the big data frame into separate data frames, one for each screen name. Then we can *apply* our logic to get the results we want; in this case, that means counting the tweets per screen name. We might also want to get just the top ten rows with the highest number of tweets. Then we can *combine* those split apart data frames into a new data frame.

Observe how this works. If we want to get a list of the top 10 tweeters, we can use the following code.

```
covid_tweets %>%
  select(from_user) %>%
  group_by(from_user) %>%
  summarize(n=n()) %>%
  arrange(desc(n)) %>%
  top_n(10, n) %>%
  kable() # print a nice table of the results
```

from_user	n
MatthewBoedy	360
mass_union	57
goldingkentucky	44
bu_phd_students	42
BBNToday	38
devmoy1	33
DrCatfromMars	27
bubbyscoffee	23
Collegecovid19	22
JG_college	22

One might assume that most of the users in the `quarantine_tweets` dataset are English speakers, since these tweets were pulled from within a 10-mile radius of Old Queens. How important are other user languages?

Question #2: Using the `quarantine_tweets` dataset, can we use `select()`, `group_by()`, `summarize()`, and `arrange()` to figure out how many tweets there are by user language (HINT: you want the `user_lang` column)?

The rtweet package does us the favor of extracting all the hashtags from the `entities` JSON object into its own column. Look for them in the `hashtags` column.

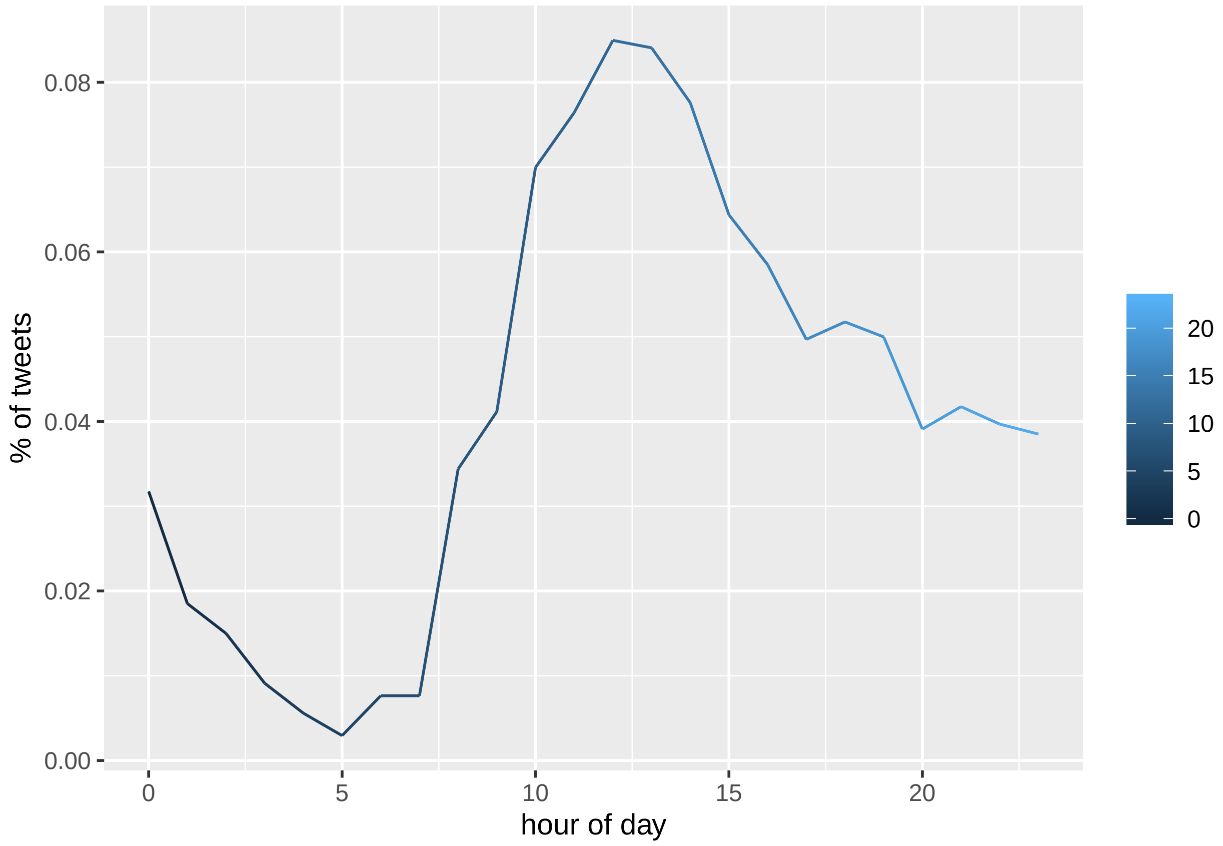
Question #3: Using the `quarantine_tweets` dataset, can we use `select()`, `group_by()`, `summarize()`, and `arrange()` to create a table of hashtags and their frequencies?

Dplyr: Mutate

The `mutate()` function lets us create new columns out of existing columns. The `created_at` column indicates the time of the tweet down to the second in UTC or Greenwich Mean Time. Let's try to get a sense of when during the day most users are tweeting. We will count the total tweets posted in each hour and convert those hours into our time zone (Eastern). Then we'll use `mutate()` to calculate the percentage of our total tweets that were posted in that hour. Finally, we'll use ggplot2 to plot the tweet percentages by the hour.

So what does that daily tweet pattern look like?

```
covid_tweets %>%
  count(hour = hour(with_tz(timestamp, "America/New_York"))) %>%
  mutate(percent = n / sum(n)) %>%
  ggplot(aes(x = hour, y = percent, color = hour)) +
  geom_line() +
  labs(x = "hour of day",
       y = "% of tweets",
       color = "")
```

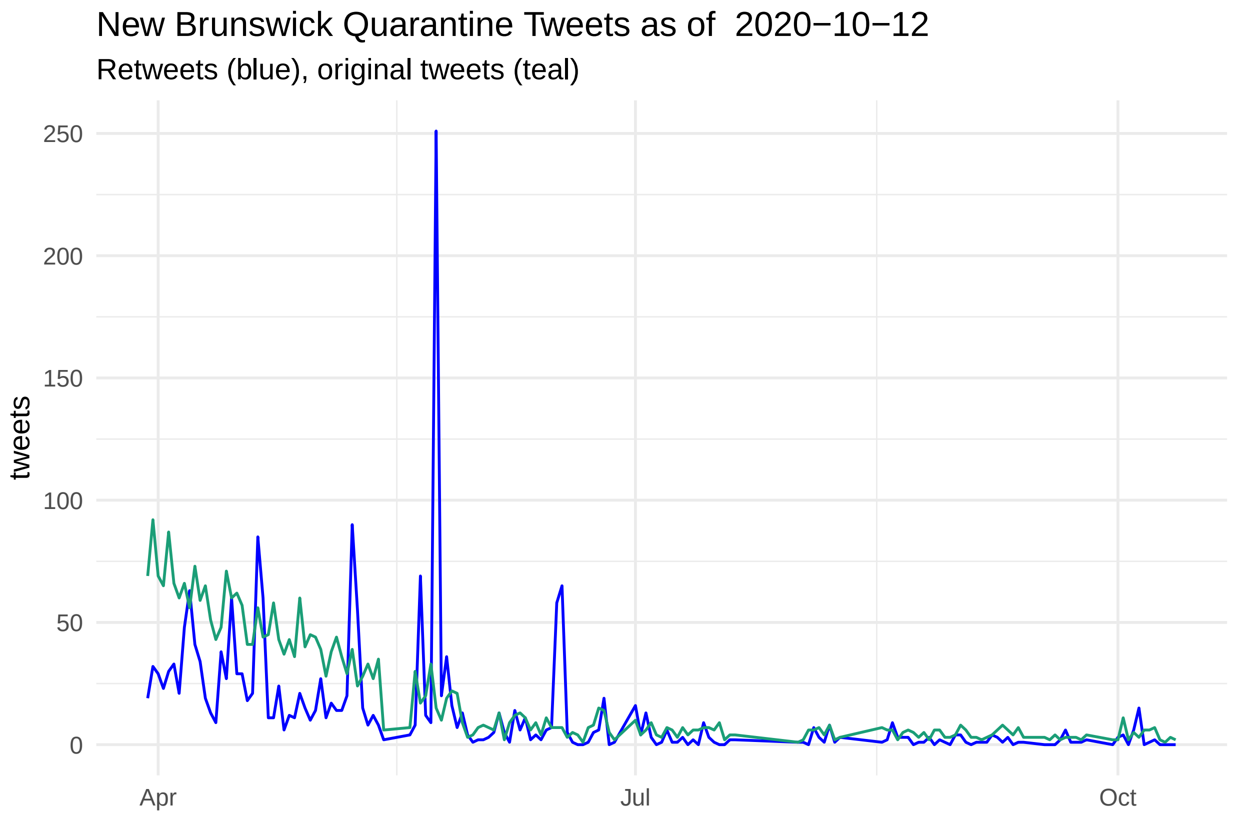


```
# save an image to file
# ggsave(paste0("covid_timeseries.png"), width = 7.5, height = 5)
```

Perhaps we could use mutate again to look at how many of those tweets are retweets vs. original tweets? Since `quarantine_tweets` provides us with a bit more information than `covid_tweets`, we will use that dataset.

```
quarantine_summary <- quarantine_tweets %>%
  select(created_at, is_retweet, is_quote, reply_to_screen_name) %>%
  mutate(day = date(created_at)) %>%
  group_by(day) %>%
  summarize(total = n(),
            retweets = sum(is_retweet == TRUE),
            quoted = sum(is_quote == TRUE),
            reply = sum(!is.na(reply_to_screen_name)),
         original = total - retweets - quoted - reply)

ggplot(quarantine_summary, aes(x=day)) +
  geom_line(aes(y=retweets), color="blue") +
  geom_line(aes(y=original), color="#1b9e77") +
  theme_minimal() +
  labs(x="", y="tweets", title=paste("New Brunswick Quarantine Tweets as of ", max(quarantine_summary$da
```



Question #4: Can we use `mutate()` to redo the plot above as percentages of retweets and original tweets (instead of counts) with respect to all tweets posted on a given day? Reuse the `quarantine_summary` object above to do so.

Dplyr: Filter

To keep certain rows, and drop others, we pass the `filter()` function, which creates a vector of `TRUE` and `FALSE` values, one for each row. The most common way to do that is to use a comparison operator on a column of the data. In this case, we'll use the `is.na()` function, which itself returns a response of `TRUE` for the rows where the `in_reply_to_user_id_str` column is empty (and `FALSE` for where it's filled). But we'll preceed it with the `!` character, which means "is not equal to." In human language, what we're asking R to do is filter, or return, only those rows where `in_reply_to_user_id_str` is filled with a value, and drop the ones where it is empty.

The question we'll explore here: how many of those tweets are direct replies to some other user's tweet (meaning the text of the tweet begins with the `@` character)?

```
covid_tweets %>%
  filter(!is.na(in_reply_to_user_id_str)) %>%
  select(in_reply_to_screen_name, timestamp) %>%
  group_by(in_reply_to_screen_name) %>%
  summarize(replies = n()) %>%
  arrange(desc(replies)) %>% View
```

Question #5: This time using the `quarantine_tweets` dataset, how can we modify the code above to return the text of the top 10 quoted tweets that have been the most favorited in the dataset. Hint: we'll need the `quoted_favorite_count` and `quoted_text` fields, and we'll use the `arrange()` and `top_n()` functions. What does this information tell us about our dataset?

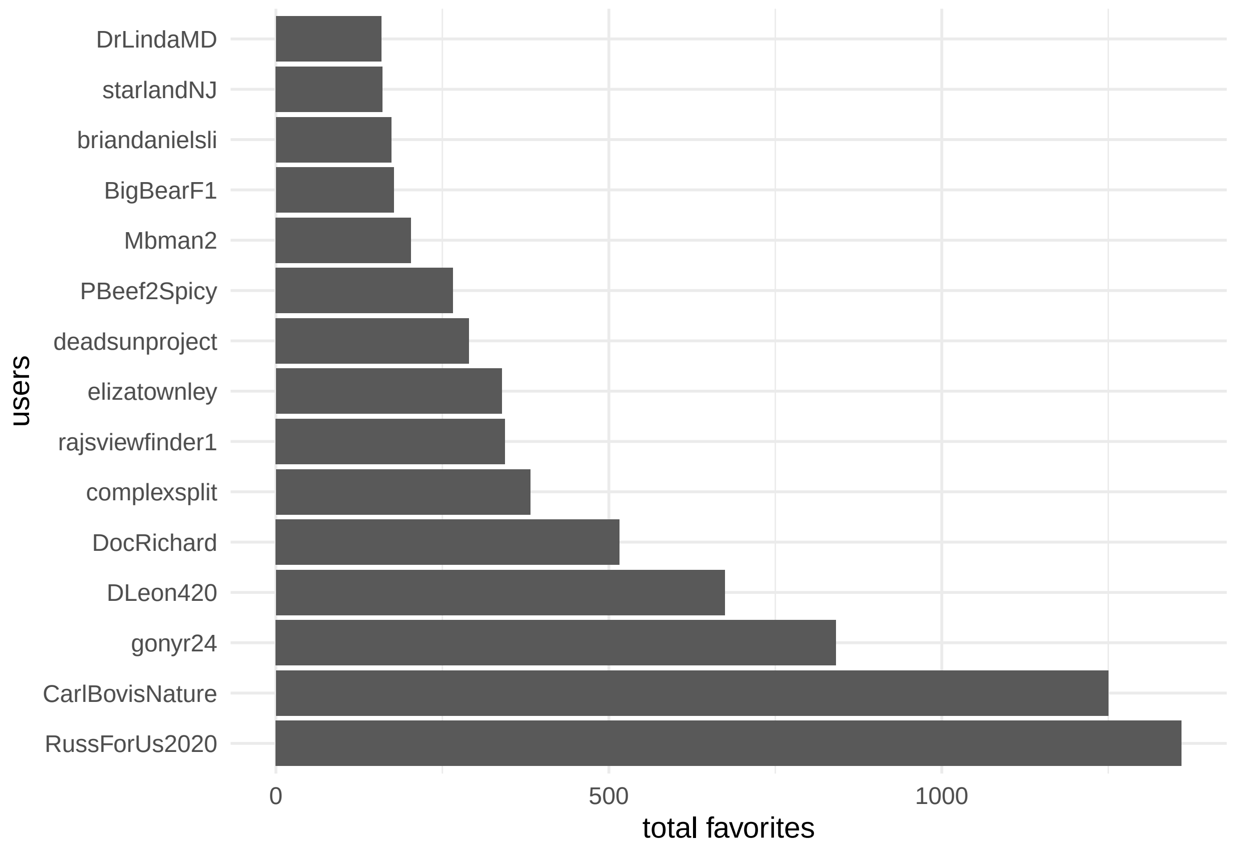
Question #6: How could we use the dplyr verbs we just learned to figure out which users have more followers than people they follow (in other words, the ratio, a rough indicator of influence)? You might want to make use of some statistical summaries, like `max()`.

Ggplot2

Let's get some practice with using ggplot2. Which users in the New Brunswick `quarantine_tweets` dataset had the most favorited tweets?

```
faves <- quarantine_tweets %>%
  group_by(screen_name) %>%
  summarize(total_faves = sum(favorite_count, na.rm = TRUE)) %>% #drop NA values
  arrange(desc(total_faves)) %>%
  top_n(15) # just the top 15

ggplot(faves, aes(x=reorder(screen_name, -total_faves), y=total_faves)) + # reorder bars by total_faves
  geom_bar(stat = "identity") +
  coord_flip() + # flip x and y axes
  theme_minimal() +
  labs(x="users", y="total favorites")
```

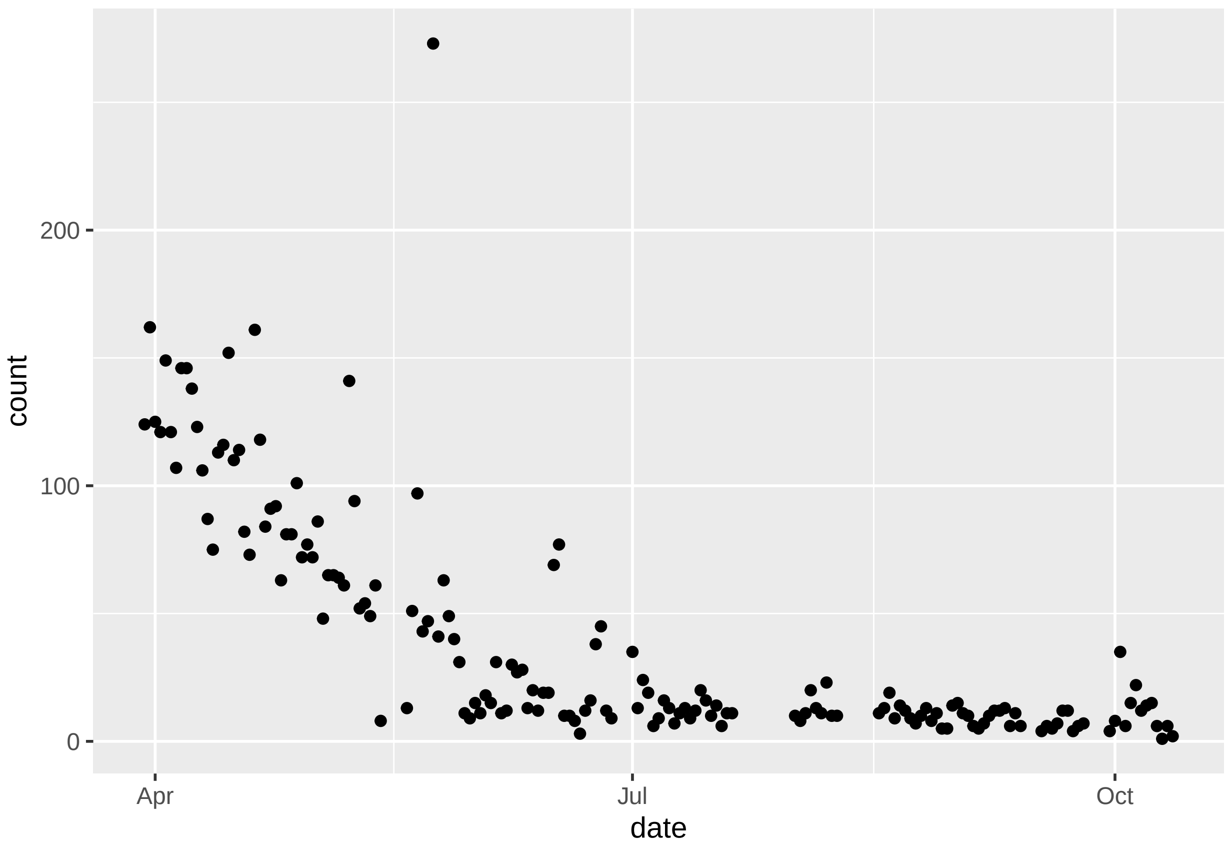


Question #7: Let's make a plot of the most retweeted users of the New Brunswick `quarantine_tweets` dataset. (HINT: you'll need the `retweet_screen_name`, the `retweet_count`, and possibly also the `retweet_text` fields)

Question #8: What devices are most users tweeting from? Let's use the `source` column to plot a bar graph. Since there are more devices than you might guess, it might make sense to take only the top 10 or so.

Lastly, let's try a different geom to look at tweet volume over time.

```r
quarantine_tweets  %>%
  select(created_at) %>%
  mutate(date = date(created_at)) %>% # bin tweet counts by date
  group_by(date) %>%
  summarize(count = n()) %>%
  ggplot(aes(x=date, y=count)) + geom_point()
```



Question #9: Can we create a comparable scatter plot for the `covid_tweets` dataset? Remeber to use the `timestamp` column in `covid_tweets`.

Answer key

These are potential solutions to the questions raised in this worksheet.

```
# Q1: How would you select the Twitter screen names and their number of friends, and arrange them in de

covid_tweets %>%
  select(from_user, user_friends_count) %>%
  arrange(desc(user_friends_count)) %>% View
```

```
# Q2: Can we use `select()`, `group_by()`, `summarize()`, and `arrange()` to figure out how many tweets

quarantine_tweets %>%
  select(lang) %>%
  group_by(tolower(lang)) %>% # `tolower()` converts to lowercase
  summarize(n = n()) %>%
  arrange(desc(n)) %>% View
```

```
# Q3: Using the `quarantine_tweets` dataset, an we use `select()`, `group_by()`, `summarize()`, and `ar

quarantine_tweets %>%
  select(hashtags) %>%
  filter(hashtags != 'NA') %>% # remove NA vals
  group_by(hashtags) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(15, count) %>%
  kable()
```

```
# Q4: Can we use `mutate()` to redo the plot above as percentages of retweets and original tweets (inst

quarantine_summary %>%
  mutate(percent_retweet = round(retweets / total * 100, digits = 2), # use `round()` to truncate the d
         percent_original = round(original / total * 100, digits = 2)) %>% View()
```

```
# Q5: How can we modify the code above to return the text of the top 10 most favorited quoted tweets in

quarantine_tweets %>%
  filter(is_quote == TRUE) %>%
  select(quoted_screen_name, quoted_text, quoted_favorite_count) %>%
  arrange(desc(quoted_favorite_count)) %>%
  top_n(10) %>% View
```

```
# Q6: How could we use the dplyr verbs we just learned to figure out which users have more followers th

covid_tweets %>%
  select(from_user, user_followers_count, user_friends_count) %>%
  filter(user_followers_count > user_friends_count) %>%
  mutate(surplus = user_followers_count - user_friends_count) %>%
  group_by(from_user) %>%
  summarize(max_followers = max(user_followers_count),
            max_friends = max(user_friends_count),
            max_surplus = max(surplus)) %>%
  arrange(desc(max_surplus)) %>%
  top_n(25) %>%  # just taking the top 25 screen names
  kable()
```

```
# Q7: Let's make a plot of the most retweeted users of the New Brunswick `quarantine_tweets` dataset. (

quarantine_tweets %>%
  select(retweet_count, retweet_text, retweet_screen_name) %>%
  group_by(retweet_screen_name) %>%
  filter(!is.na(retweet_text)) %>% # remove na vals
  summarize(total_retweets = sum(retweet_count)) %>%
  arrange(desc(total_retweets)) %>%
  top_n(10, total_retweets) %>% # lets just take the top 10
  ggplot(aes(x=retweet_screen_name, y=total_retweets)) + geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # rotate X axis labels a bit for readability
```

```
# Q8: What devices are most users tweeting from? Let's use the `source` column to plot a bar graph. Sin

quarantine_tweets %>%
  select(source) %>%
  group_by(source) %>%
  summarize(total_by_source = n()) %>%
  top_n(10, total_by_source) %>%
  ggplot(aes(x=source, y=total_by_source)) + geom_bar(stat = "identity") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) # rotating labels again
```

```r
# Q9: Can we create a comparable scatter plot for the `covid_tweets` dataset? Remeber to use the `times

covid_tweets %>%
  select(timestamp) %>%
  mutate(date = date(timestamp)) %>%
  group_by(date) %>%
  summarize(count = n()) %>%
  ggplot(aes(x=date, y=count)) + geom_point()
```