Analyzing Social Media Data

Francesca Giannetti

2/17/2020

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.

```
6
```

Variables

A variable is a holder for a value.

```
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?

```
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 these lab 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
```

Collecting Data Redux

At last week's workshop on collecting Twitter data, we left off with the rtweet package. rtweet allows us to query Twitter's REST API using a variety of methods, including searching by strings (including account names and hashtags), user timelines, and latitude and longitude coordinates.

For help with the full rtweet package, type `help(package="rtweet")` at the console or visit the intro vignette at https://rtweet.info/articles/intro.html.

Note: the first time you run one of these fuctions, an authentication process will launch in a broswer window asking for you to connect your Twitter account to Kearney's Twitter Developer keys.

```
## a combined hashtag and phrase search
## don't run this because it'll take too long
bhm_tweets <- search_tweets(
  "#blackhistorymonth OR #bhm OR \"black history month\"", n = 18000, include_rts = TRUE, retryonrateli
)

## search by lat-lon pair
## you can look this up at https://www.latlong.net/
alexander_tweets <- search_tweets(
  geocode = "40.5045521,-74.4527554,5mi", include_rts = FALSE
)

## search the most recent 1,000 direct replies to @CuteEmergency
cute_tweets <- search_tweets(
  "to:CuteEmergency",
  n = 1000
)
```

Load (more) data

To give us a few more analysis options, let's also read in these prepared datasets as data frames. 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).

```
## load twitter datasets
bhm <- read_twitter_csv("data/2020-02-17_black-history-month.csv")
bls <- read.delim("data/bls.tsv", sep = "\t", encoding = "UTF-8", quote = "", header = TRUE, stringsAsFa
pennsound <- read.delim("data/2020-02-17_pennsound.tsv", sep = "\t", encoding = "UTF-8", quote = "", hea
ubu <- read.delim("data/ubuweb.tsv", sep = "\t", encoding = "UTF-8", quote = "", header = TRUE, strings

## remove duplicate tweets from the TAGS datasets
bls_dedup <- subset(bls, !duplicated(bls$id_str))
pennsound_dedup <- subset(pennsound, !duplicated(pennsound$id_str))
ubu_dedup <- subset(ubu, !duplicated(ubu$id_str))
```

Review of Tweet object

Let's spend some time looking at the JSON representation of a tweet, which you can access here. For reference, consult the Tweet Data Dictionary for definitions of the "root-level" field names. Having a familiarity with these fields will help you to develop your research questions.

Data summaries

We can get an overview of our datasets with `str()`, which displays the structure of an object. Another helpful function to retrieve just the headers of your dataset: `names()`.

```
str(pennsound_dedup)

names(pennsound_dedup)
```

R doesn't really intuit data types, as you may have noticed when you ran the `str()` function. You have to explicitly tell it the data type of your variable. In this case, we're going to use the values of the `time` column to create a brand new column called `timestamp` with a properly formated date and time that R will recognize. Note that we need to specify the time zone as well.

Notice how we specify a single variable from a data frame by using the dollar sign ($).

```
bhm$created_at <- as.POSIXct(bhm$created_at, format= "%Y-%m-%d %H:%M:%S", tz="GMT")
bls_dedup$timestamp <- as.POSIXct(bls_dedup$time, format= "%d/%m/%Y %H:%M:%S", tz="GMT")
pennsound_dedup$timestamp <- as.POSIXct(pennsound_dedup$time, format= "%d/%m/%Y %H:%M:%S", tz="GMT")
ubu_dedup$timestamp <- as.POSIXct(ubu_dedup$time, format= "%d/%m/%Y %H:%M:%S", tz="GMT")
```

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 (:).

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

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

Select

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!

```
bhm %>%
  select(screen_name, created_at) %>% View
```

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 `screen_name` values?

```
bhm %>%
  select(screen_name, followers_count) %>%
  arrange(desc(followers_count)) %>% View
```

Question #1: How would you select the Twitter screen names 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.

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.

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

screen_name	n
Cyber_FM	91
Lidia07461636	85
TomthunkitsMind	69
FebBlackHistory	49
Ummismaelsf	44

screen_name	n
burrow	25
shamaralstonn	24
HotSausage00	22
IakoubiY	22
Spot225Bot	22

One might reasonably assume that Black History Month is primarily an American celebration. Can we get a sense of how many of the users in this dataset are English speakers? How important are other user languages?

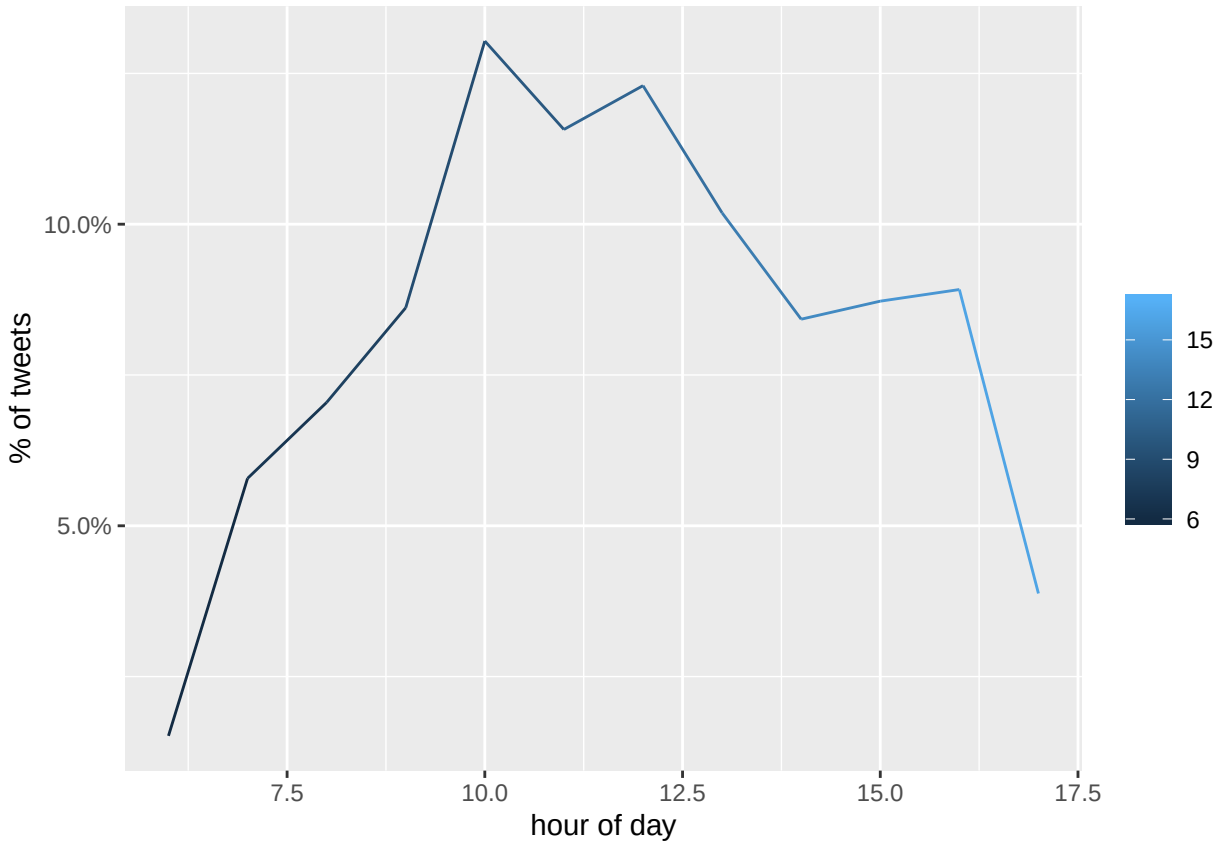
Question #2: Can we use `select()`, `group_by()`, `summarize()`, and `arrange()` to figure out how many tweets there are by user language?

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. This level of granularity might make the resulting visualization a bit messy. To make our plot more readable, we will count the total tweets posted in each hour and convert those hours into our time zone. 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.

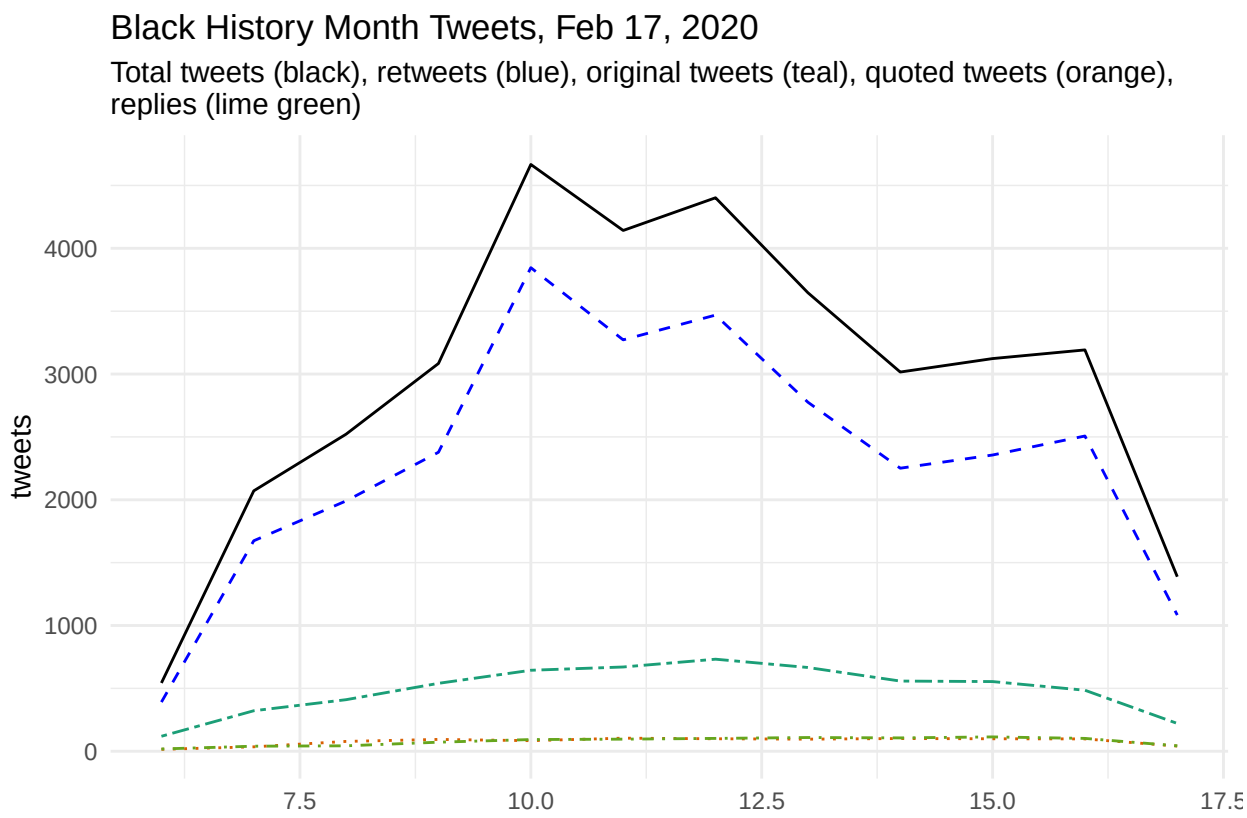
So what does that tweet volume over time look like?

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



But how many of those are retweets? Quoted tweets? Original tweets? Let's figure it out.

```
bhm_summary <- bhm %>%
  select(created_at, is_retweet, is_quote, reply_to_screen_name) %>%
  mutate(hour = hour(with_tz(created_at, "America/New_York"))) %>%
  group_by(hour) %>%
  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(bhm_summary, aes(x=hour)) +
  geom_line(aes(y=retweets), linetype="dashed", color="blue") +
  geom_line(aes(y=original), linetype="twodash", color="#1b9e77") +
  geom_line(aes(y=quoted), linetype="dotted", color="#d95f02") +
  geom_line(aes(y=reply), linetype="dotdash", color="#66a61e") +
  geom_line(aes(y=total), linetype="solid", color="black") +
  theme_minimal() +
  labs(x="", y="tweets", title="Black History Month Tweets, Feb 17, 2020", subtitle = "Total tweets (bla
```




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

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 `reply_to_user_id` 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)?

```
bhm %>%
  filter(!is.na(reply_to_user_id)) %>%
  select(reply_to_screen_name, created_at) %>%
  group_by(reply_to_screen_name) %>%
  summarize(replies = n()) %>%
  arrange(desc(replies)) %>% View
```

Question #3: 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?

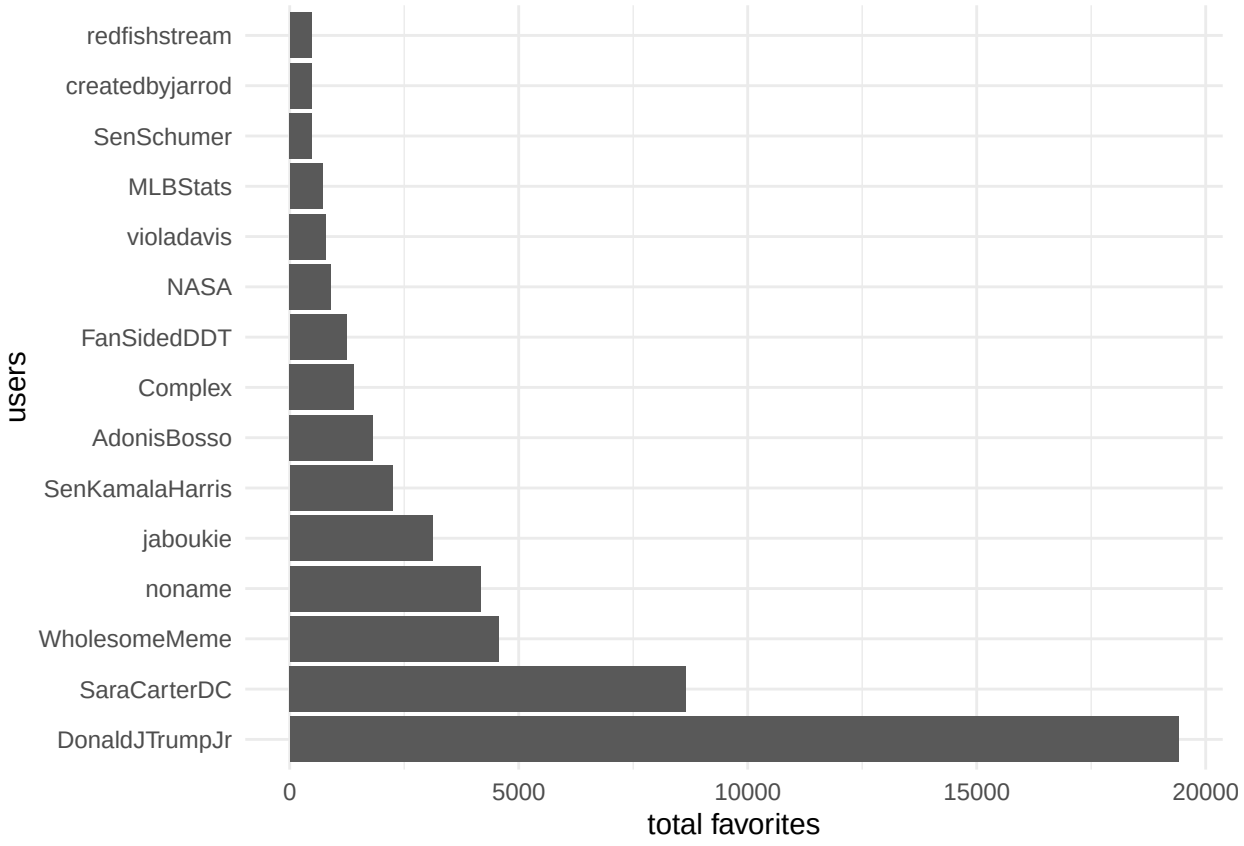
Question #4: How could we use the dplyr verbs we just learned to figure out which users have more followers

than people they follow (a rough indicator of influence)? You might want to make use of some statistical summaries, like max().

Plots

Let's get some practice with using `ggplot2`. Which users in the Black History Month dataset had the most favorited tweets?

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

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

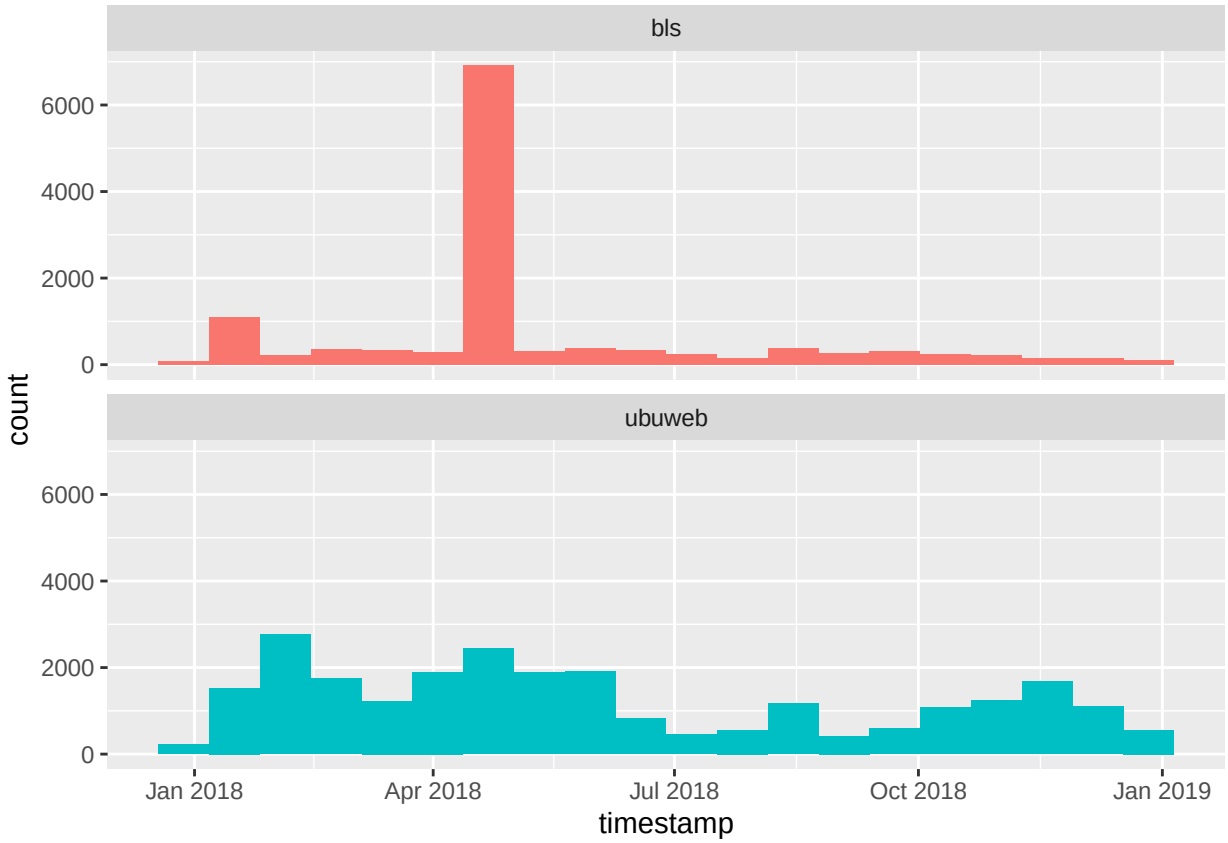


Let's make a plot of the most retweeted users of the Black History Month dataset.

Let's turn now to the sound archive tweets. What does the British Library Sounds tweet volume look like compared to Ubuweb?

```
# create subset for same time window in both datasets
bls_sample <- bls_dedup %>%
  filter(timestamp > as.POSIXlt("2018-01-01 00:00:00", tz="GMT"),
```

```
        timestamp < as.POSIXlt("2018-12-31 00:00:00", tz="GMT")) %>%
  select(text,timestamp,from_user)

ubu_sample <- ubu_dedup %>%
  filter(timestamp > as.POSIXlt("2018-01-01 00:00:00", tz="GMT"),
         timestamp < as.POSIXlt("2018-12-31 00:00:00", tz="GMT")) %>%
  select(text,timestamp,from_user)

# bind them together and plot histogram
sound_tweets <- bind_rows(bls_sample %>%
                      mutate(dataset = "bls"),
                    ubu_sample %>%
                      mutate(dataset = "ubuweb"))

ggplot(sound_tweets, aes(x = timestamp, fill = dataset)) +
  geom_histogram(position = "identity", bins = 20, show.legend = FALSE) +
  facet_wrap(~dataset, ncol = 1)
```

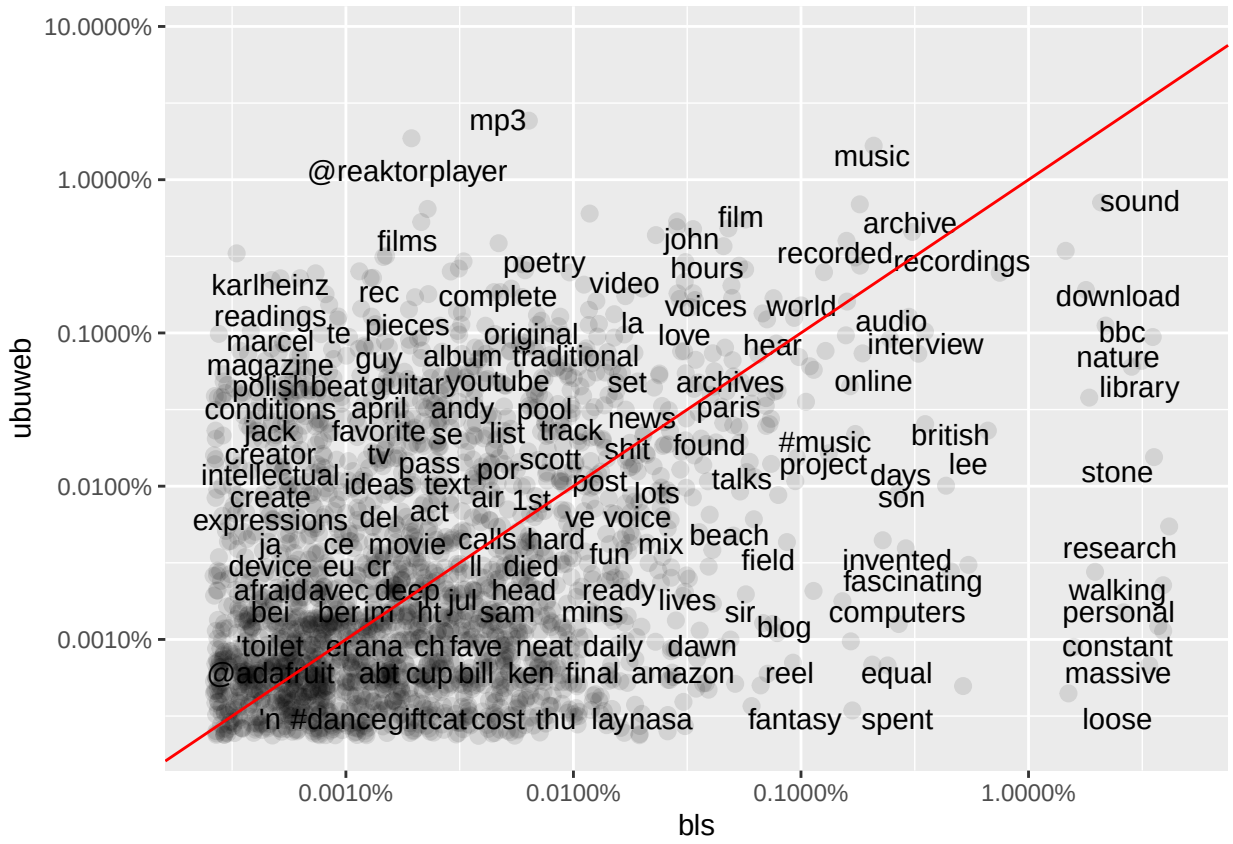


Word frequencies

What was everybody talking about? Here, we'll use the package tidytext to process the text of the tweets.

In the plot below, the terms that hover close to the reference line are used with about equal frequency by both the British Library Sounds and the Ubuweb tweets. The further away the terms are from the reference line, the more characteristic they are of the individual sound archive.

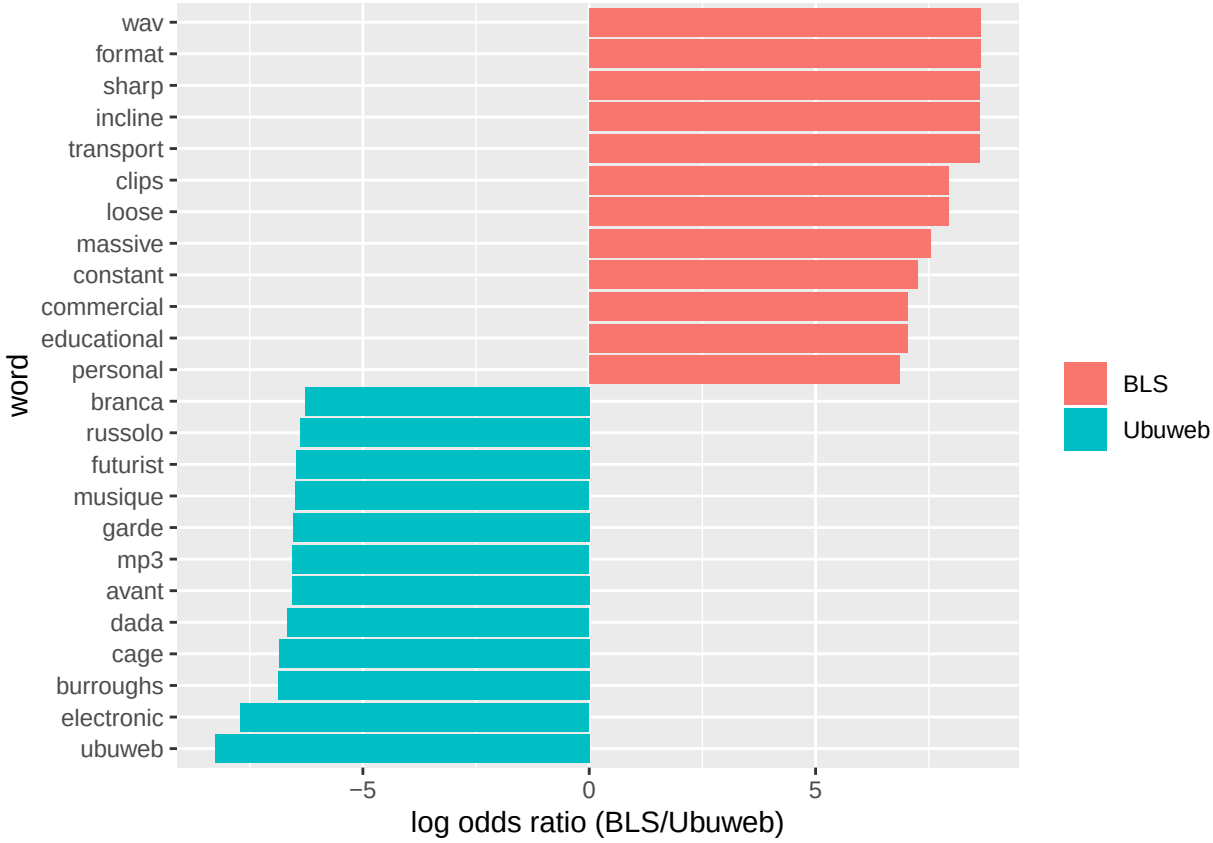
```r
# we use a regular expression to discard some common Twitter words from the analysis
# we split words using non all non-word characters, e.g. spaces and punctuation
replace_reg <- "https://t.co/[A-Za-z\\d]+|http://[A-Za-z\\d]+|&amp;|&lt;|&gt;|RT|https"
unnest_reg <- "([^A-Za-z_\\d#@']|'(?![A-Za-z_\\d#@]))"

tidy_tweets <- sound_tweets %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "regex", pattern = unnest_reg) %>%
  filter(!word %in% stop_words$word,
         str_detect(word, "[a-z]"))

# calculate word frequencies for each dataset
frequency <- tidy_tweets %>%
  group_by(dataset) %>%
  count(word, sort = TRUE) %>%
  left_join(tidy_tweets %>%
              group_by(dataset) %>%
              summarise(total = n())) %>%
  mutate(freq = n/total)

frequency <- frequency %>%
  select(dataset, word, freq) %>%
  pivot_wider(names_from=dataset, values_from = freq) %>% #pivot_wider() from the tidyr package
  arrange(bls, ubuweb)

ggplot(frequency, aes(bls, ubuweb)) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  geom_abline(color = "red")
```



Distinctive word usage

Now let's find which words are more or less likely to come from each sound archive's account using the log odds ratio.

```
word_ratios <- tidy_tweets %>%
  filter(!str_detect(word, "^@")) %>% # remove screen names
  count(word, dataset) %>%
  filter(sum(n) >= 10) %>%
  ungroup() %>%
  pivot_wider(names_from=dataset, values_from=n, values_fill = list(n=0)) %>%
  mutate_if(is.numeric, list(~(. + 1) / (sum(.) + 1))) %>%
  mutate(logratio = log(bls / ubuweb)) %>%
  arrange(desc(logratio))

word_ratios %>%
  group_by(logratio < 0) %>%
  top_n(12, abs(logratio)) %>%
  ungroup() %>%
  mutate(word = reorder(word, logratio)) %>%
  ggplot(aes(word, logratio, fill = logratio < 0)) +
  geom_col() +
  coord_flip() +
  ylab("log odds ratio (BLS/Ubuweb)") +
  scale_fill_discrete(name = "", labels = c("BLS", "Ubuweb"))
```



What are some of the words that are equally as likely to come from both sound archives' tweets in 2018?

```
word_ratios %>%
  arrange(abs(logratio))
```

```
## # A tibble: 13,827 x 4
##    word       ubuweb       bls logratio
##    <chr>       <dbl>     <dbl>    <dbl>
##  1 paris   0.000525  0.000528   0.00542
##  2 hey     0.0000716 0.0000711 -0.00748
##  3 beauty  0.000191  0.000193   0.0102
##  4 michael 0.000191  0.000193   0.0102
##  5 helps   0.0000669 0.0000660 -0.0126
##  6 dec     0.000100  0.000102   0.0127
##  7 words   0.000191  0.000188  -0.0165
##  8 event   0.000334  0.000340   0.0177
##  9 1980s   0.000177  0.000173  -0.0230
## 10 include 0.0000573 0.0000559 -0.0255
## # ... with 13,817 more rows
```

Word tokens

Now we'll transform an existing table of tweets into a table with one row per word inside the tweet. We'll use the `unnest_tokens()` function from the tidytext package to do so.

```
tweet_words <- pennsound_dedup %>%
  select(id_str,
```

```
        from_user,
        text,
        timestamp) %>%
  unnest_tokens(word, text)

# let's look at the first few rows with the `head()` function
head(tweet_words)
```

```
##                  id_str       from_user           timestamp          word
## 1   1229176267691433984 ChrlesBernstein 2020-02-16 22:50:33      soundeye
## 1.1 1229176267691433984 ChrlesBernstein 2020-02-16 22:50:33 international
## 1.2 1229176267691433984 ChrlesBernstein 2020-02-16 22:50:33        poetry
## 1.3 1229176267691433984 ChrlesBernstein 2020-02-16 22:50:33      festival
## 1.4 1229176267691433984 ChrlesBernstein 2020-02-16 22:50:33             a
## 1.5 1229176267691433984 ChrlesBernstein 2020-02-16 22:50:33   documentary
```

Stop words

The `tidytext` package contains a `stop_words` table, which is a list of common words that are most commonly discarded before performing an analysis. We'll customize ours with a few words that are common to tweets.

```
# create a custom Twitter stopword list for the PennSound tweets
my_stop_words <- tibble(
  word = c(
    "https",
    "t.co",
    "rt",
    "amp",
    "rstats",
    "gt",
    "pennsound"
  ),
  lexicon = "twitter"
)

all_stop_words <- stop_words %>%
  bind_rows(my_stop_words)

# we'll pass an additional filter to remove words that are numbers
suppressWarnings({
  no_numbers <- tweet_words %>%
    filter(is.na(as.numeric(word)))
})

no_stop_words <- no_numbers %>%
  anti_join(all_stop_words, by = "word")

# over half of the words were stop words!
tibble(
  total_words = nrow(tweet_words),
  after_cleanup = nrow(no_stop_words)
)
```

```
## # A tibble: 1 x 2
```

```
##   total_words after_cleanup
##         <int>         <int>
## 1      101758         49835
```

```
# let's look at a table of our top words
top_words <- no_stop_words %>%
  group_by(word) %>%
  tally %>%
  arrange(desc(n)) %>%
  head(10)
```

Sentiment matching

The `get_sentiments()` functions in `tidytext` match words against different lexicons. We'll choose the NRC lexicon for this example. The `get_sentiments()` function returns a data frame. Sentiment analysis is often a bit crude.

With the NRC lexicon, one word may have multiple sentiments. For example, the word "sonnet" has joy, positive, and sadness classifications. This means multiple matches for words that have more than one sentiment classification. You may or may not find these classifications reasonable. For this and other reasons, it is imperative to inspect the results of any computational analysis.

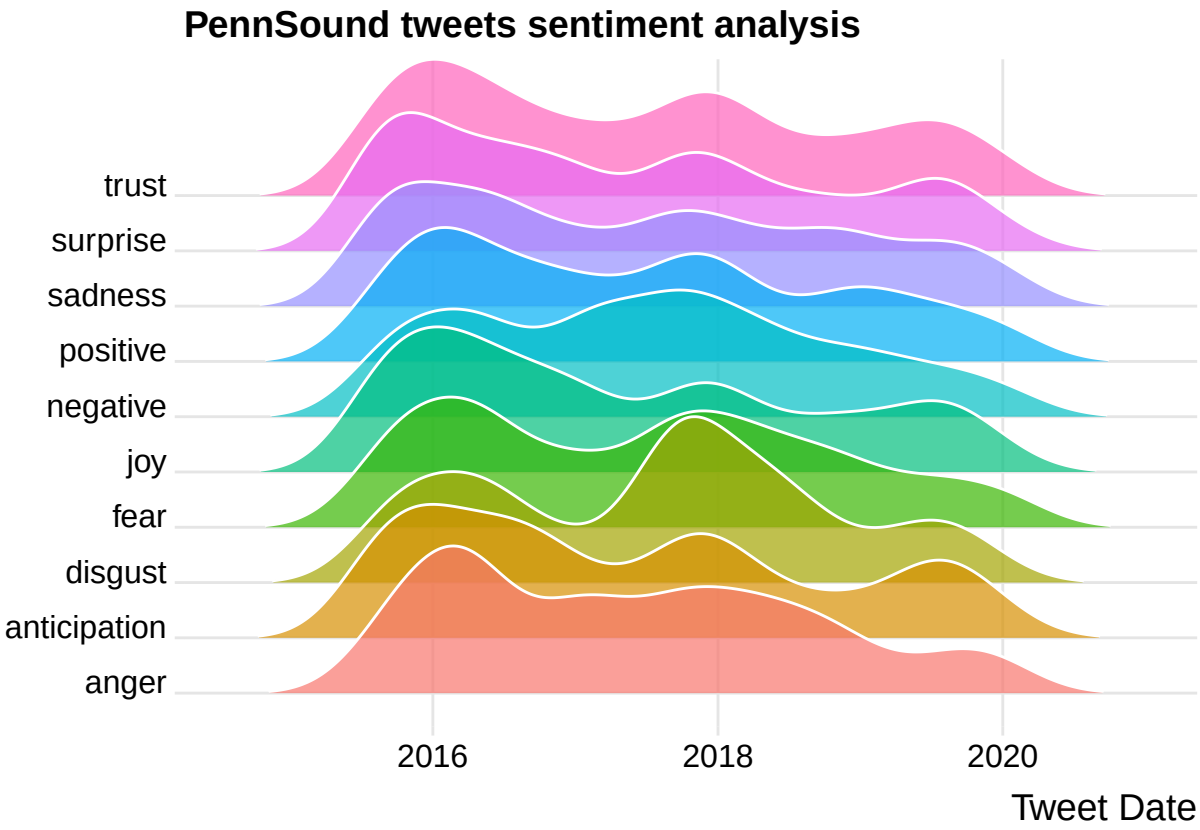
```
nrc_words <- no_stop_words %>%
  inner_join(get_sentiments("nrc"), by = "word")

# inspect results
nrc_words %>%
  group_by(sentiment) %>%
  tally %>%
  arrange(desc(n)) %>% View

# let's look at the words assigned to "disgust" what do we think?
nrc_words %>%
  filter(sentiment == "disgust") %>%
  group_by(word) %>%
  tally %>% View
```

Visualize results

We will use the `ggridges` package to visualize the results of our sentiment analysis as a ridgeline plot. Remember to type `help(package="ggridges")` at the console for the help pane.

```
ggplot(nrc_words) +
    geom_density_ridges(aes(
      x = timestamp,
      y = sentiment,
      fill = sentiment),
      rel_min_height = 0.01,
      alpha = 0.7,
      color = 'white',
      scale = 3) +
    theme_ridges() +
    labs(title = "PennSound tweets sentiment analysis",
         x = "Tweet Date",
```

```
        y = NULL) +
  scale_fill_discrete(guide=FALSE)
```



Save your data

You may (and should) write any new and derived data you want to keep to file. It's very easy to read it back into R to continue your work later.

```
#write to file
write_as_csv(bhm, "data/2020-02-17_black-history-month.csv", fileEncoding = "UTF-8")

# read back into R
bhm_take2 <- read_twitter_csv("data/2020-02-17_black-history-month.csv")
```

Feedback

Finally, I'd appreciate it if you could fill out our brief feedback survey: http://bit.ly/s20_dh_feedback. Thanks for coming!

Further Reading

Fitzgerald, J.D. "Sentiment analysis of (you guessed it!) Donald Trump's tweets." https://www.storybench.org/sentiment-analysis-of-you-guessed-it-donald-trumps-tweets/

Rudis, R. "21 Recipes for Mining Twitter with rtweet." https://rud.is/books/21-recipes/

Ruiz, E. “Sentiment analysis using tidytext.” https://www.edgarsdatalab.com/2017/09/04/sentiment-analysis-using-tidytext/

Silge, J. and Robinson, D. “Case study: comparing Twitter archives” *Text Mining with R.* (O’Reilly, 2017). http://tidytextmining.com/twitter.html.

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

bhm %>%
  select(screen_name, friends_count) %>%
  arrange(desc(friends_count)) %>% View
```

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

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

```
# Q3: How can we modify the code above to return the text of the top 10 most favorited quoted tweets in

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

```
# Q4: How could we use the dplyr verbs we just learned to figure out which users have more followers th

bhm %>%
  select(screen_name, followers_count, friends_count) %>%
  filter(followers_count > friends_count) %>%
  mutate(surplus = followers_count - friends_count) %>%
  group_by(screen_name) %>%
  summarize(max_followers = max(followers_count),
            max_friends = max(friends_count),
            max_surplus = max(surplus)) %>%
  arrange(desc(max_surplus)) %>%
  top_n(25) %>%  # just taking the top 25 screen names
  kable()
```