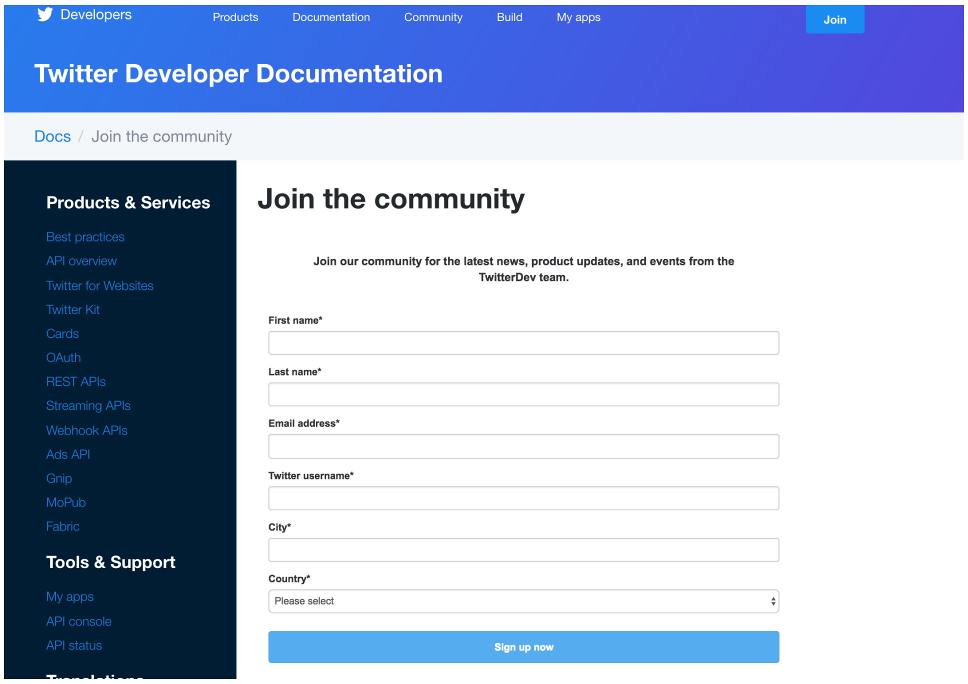
R-Ladies Melbourne - Twitter Workshop

Anna Quaglieri & Saskia Freytag

Contents

1 Set up a twitter API to get started with twitter

Before you can download data from Twitter, you need to setup your API by becoming a Twitter developer. First, to become a Twitter developer you need to have a Twitter account. Then, you can sign up here (Figure 1). Once you have signed up click on the *My Apps* buttom on the left panel as in Figure 2 and then click on *Create a new app*. You should provide a brief description about your App and a few other details. For today's workshop we set up the the *RLadies* App shown in Figure 3. By clicking onto your App's name you can now manage and see your settings as well as the codes that you need to get started with R. You can find the keys that you need in the *Keys and Access Tokens* button. Here you can find your four codes (highlighted in Figure 4) to use into R.



Twitter Developer Documentation

Join the community

Join our community for the latest news, product updates, and events from the TwitterDev team.

First name*

Last name*

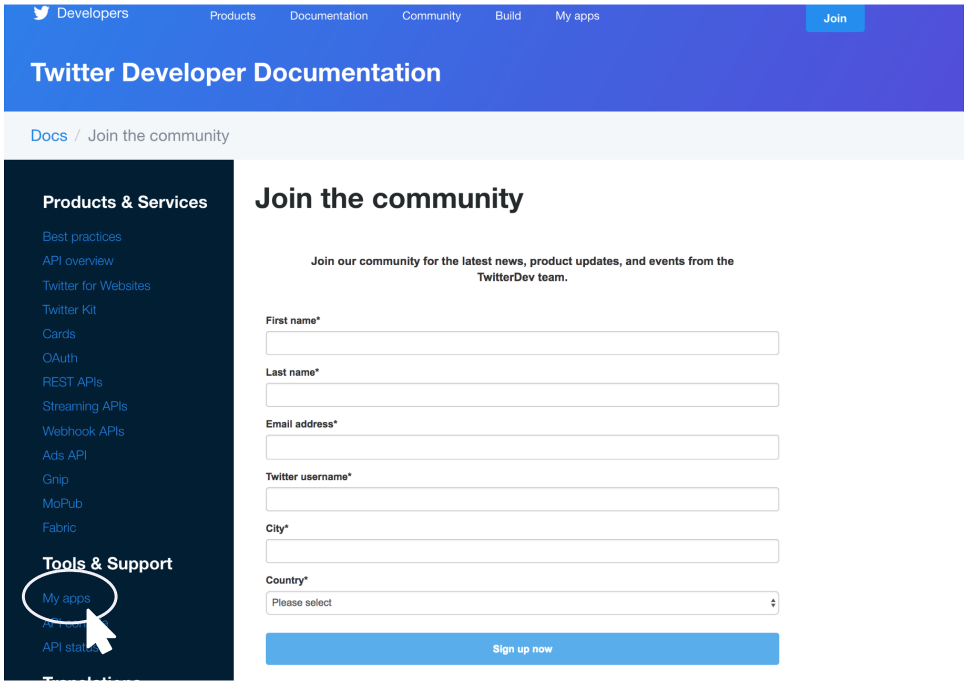
Email address*

Twitter username*

City*

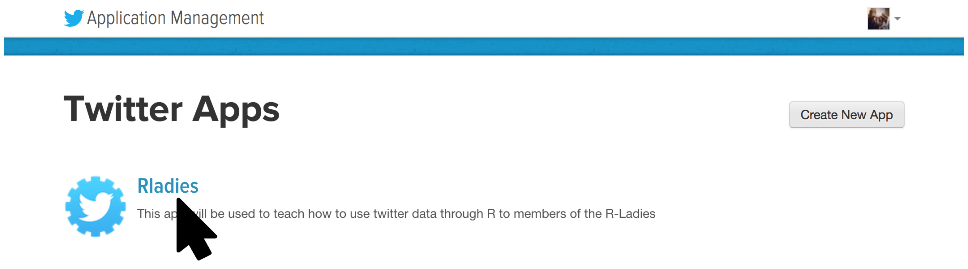
Country*

Figure 1: Signup at https://dev.twitter.com/resources/signup.



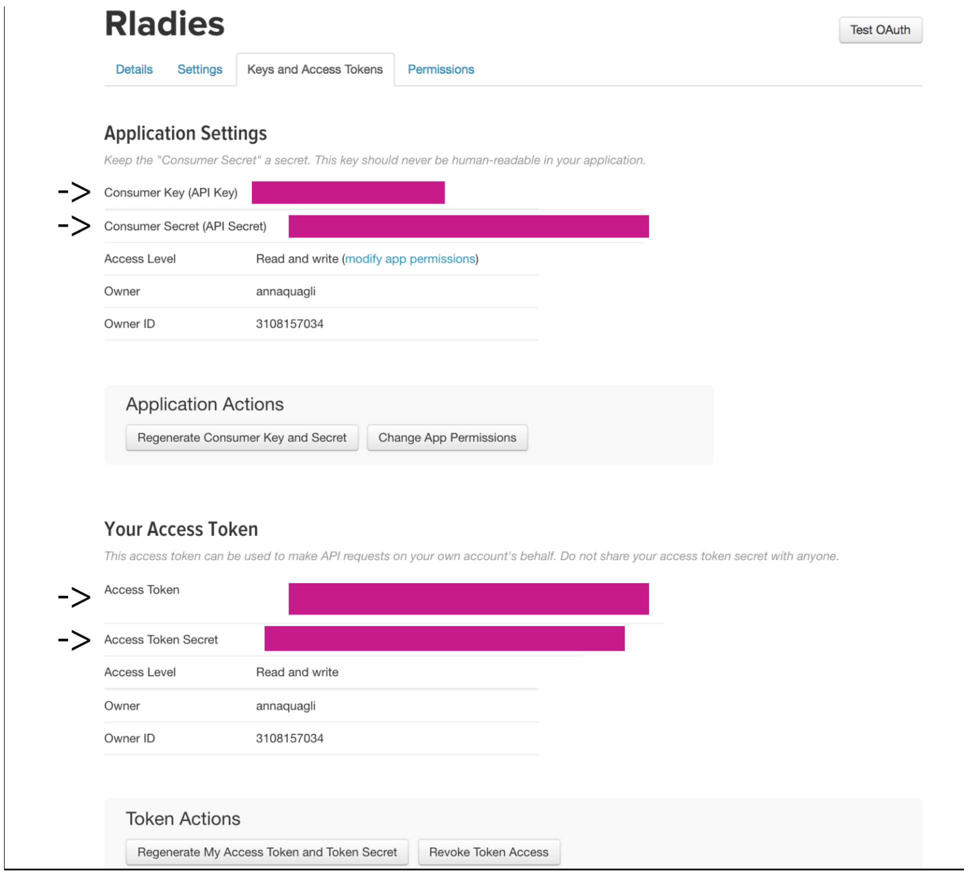
Join our community for the latest news, product updates, and events from the TwitterDev team.

Figure 2: After signing up click on ‘My Apps’ and then ’Create a new App’s.



This ap[illegible]ill be used to teach how to use twitter data through R to members of the R-Ladies

Figure 3: You can now manage your App’s settings.



Rladies

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

->	Consumer Key (API Key)	[redacted]
->	Consumer Secret (API Secret)	[redacted]
	Access Level	Read and write (modify app permissions)
	Owner	annaquagli
	Owner ID	3108157034

Application Actions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

->	Access Token	[redacted]
->	Access Token Secret	[redacted]
	Access Level	Read and write
	Owner	annaquagli
	Owner ID	3108157034

Token Actions

Figure 4: This is where you can find the access keys that you need to get started with R.

2 Let's get started with RuPaul's drag race's tweets!

RuPaul's drag race is an American reality competitions showing RuPaul's search for "America's next drag superstar". Just to tease your curiosity, Figure 5 shows you all the Queens which are competing in Season 9!

Figure 5: Drag queens competing in RuPaul's drag race Season 9.

2.1 Research your hashtags

The first thing to do when you want to analyse a topic using Twitter's data is to know what is or are the most useful *hastag/s* that can describe your questions. This means those hastags that are most commonly used when what you are interested in is tweeted. In this example we are interested into the trends of RuPaul's drag race Season 9. A little bit of research through Twitter or Goggle is needed to define a set of good hastags. For example, we found a web page listing the top hashtags for RuPaul's drag race https://top-hashtags.com/hashtag/rupaul/. For simplicity we will limit our search to two hastags and we decided to use one specific for season 9, *#rpdr9*, and a more general one, *#rupaulsdragrace*.

2.2 Initialize functions needed for the analyses

First, start by installing and loading all the necessary packages.

```
> # Packages that you need to setup your
> # API in R and download twitter data
> install.packages("twitteR")
> install.packages("ROAuth")
> install.packages("RCurl")
> install.packages("RJSONIO")
>
> # Packages for sentiment analyses and
> # wordclouds
> install.packages("RSentiment")
> install.packages("tm")
> install.packages("wordcloud")
>
> # Genral R packages for plotting and
> # manipulating data
> install.packages("tidyr")
> install.packages("dplyr")
> install.packages("ggplot2")
> install.packages("plotly")
```

```
> # Require all packages
> require("tm")
> require("wordcloud")
> require("twitteR")
> require("RSentiment")
> require("ROAuth")
> require("RCurl")
> require("RJSONIO")
> require("tidyr")
> require("dplyr")
> require("ggplot2")
> require("plotly")
```

Then setup your *access keys* as below. If you don't remember where to find these codes see Figure 1).

```
> api_key <- "yourAPIKey"
> api_secret <- "yourAPISecret"
> token <- "yourToken"
> token_secret <- "yourTokenSecret"
>
```

```
> twitteR::setup_twitter_oauth(api_key, api_secret,
+     token, token_secret)
```

2.3 Start the search

Twitter only allows you to retrieve tweets up to about two weeks before. Therefore, if you are planning to analyse some particular tweets across few weeks or months you will have to plan the downloading about every two weeks. For example, for this workshop we downloaded data on the 5th of May and on the 14th of May using the code below:

2.3.1 Download the tweets

```
> # Download only English tweets
> twittes_rpdr9 <- twitteR::searchTwitter("#rpdr9", n = 3000, lang= "en", since= "2017-03-01")
> # Convert the list to a data.frame
> toDF_rupol <- twitteR::twListToDF(twitter_rpdr9)
>
> # Save data frame as an R object
> saveRDS(twitter_rpdr9, file.path(dir,"Data","twitter_rpdr9.rds")
```

Now, read into R the two sets of tweets that we previously downloaded.

```
> twitter_rpdr9_1 <- readRDS(file.path(dir,
+     "Data", "twitter_rpdr9.rds"))
> twitter_rpdr9_2 <- readRDS(file.path(dir,
+     "Data", "twitter_rpdr9_2.rds"))
> twitter_rpdr <- readRDS(file.path(dir, "Data",
+     "twitter_rupoldragrace.rds"))
> class(twitter_rpdr9_1)
```

```
## [1] "list"
```

```
> # Convert the lists to data.frames
> toDF_rupol_1 <- twitteR::twListToDF(twitter_rpdr9_1)
> toDF_rupol_2 <- twitteR::twListToDF(twitter_rpdr9_2)
> toDF_rupol_3 <- twitteR::twListToDF(twitter_rpdr)
```

Now we would lile to combine the two sets of tweets and remove overlapping tweets since there is overlap in the days.

```
> # Time interval of the tweets dates Set 1
> min(toDF_rupol_1$created)
```

```
## [1] "2017-04-26 13:22:29 UTC"
```

```
> max(toDF_rupol_1$created)
```

```
## [1] "2017-05-06 00:56:42 UTC"
```

```
> # Set 2
> min(toDF_rupol_2$created)
```

```
## [1] "2017-05-04 21:01:33 UTC"
```

```
> max(toDF_rupol_2$created)
```

```
## [1] "2017-05-14 01:56:04 UTC"
```

```
> # Set 3
> min(toDF_rupol_3$created)
```

```
## [1] "2017-05-04 21:48:35 UTC"
```

```
> max(toDF_rupol_3$created)
```

```
## [1] "2017-05-14 03:29:49 UTC"
```

```
> # There is a two days overlap
>
> # Combine the data.frame and remove
> # duplicated tweets
> combineTweetsDupl <- rbind(toDF_rupol_1,
+     toDF_rupol_2, toDF_rupol_3)
> duplicatedRows <- duplicated(combineTweetsDupl)
> sum(duplicatedRows)
```

```
## [1] 162
```

```
> combineTweets <- combineTweetsDupl[!duplicatedRows,
+     ]
> sum(duplicated(combineTweets))
```

```
## [1] 0
```

```
> colnames(combineTweets)
```

```
##  [1] "text"          "favorited"     "favoriteCount" "replyToSN"
##  [5] "created"       "truncated"     "replyToSID"    "id"
##  [9] "replyToUID"    "statusSource"  "screenName"    "retweetCount"
## [13] "isRetweet"     "retweeted"     "longitude"     "latitude"
```

```
> # save to file
> write.csv(combineTweets, file.path(dir, "Data",
+     "tweets_hastags_combined.csv"), row.names = FALSE)
```

With the help of tidyr we can reorganise and manipulate the dataset. For example, we can plot the number of tweets per day.

```
> toDF_rupol_daily <- combineTweets %>% tidyr::separate(created,
+     into = c("Day", "Time"), sep = " ") %>%
+     dplyr::group_by(Day) %>% dplyr::summarise(tweetsPerDay = length(text)) %>%
+     ggplot(aes(x = Day, y = tweetsPerDay)) +
+     geom_bar(stat = "identity", fill = "#88398A") +
+     theme_bw() + theme(axis.text.x = element_text(angle = 45,
+     hjust = 1)) + coord_flip()
> toDF_rupol_daily
```

April 29th is the first day of our sets of tweets with a higher frequency of tweets and indeed it corresponds to RuPaul's Drag Race Episode 6 which in America was the night of April 28th (for the list of all the episodes see RuPaul's Wikia). The same thing apply for May 6th. You could also stratify every day by time and check at which time the highest number tweets was written. This is not going to be addressed here but for examples on how to deal with dates and times in R you can look at this tutorial from Berkeley https://www.stat.berkeley.edu/~s133/dates.html.

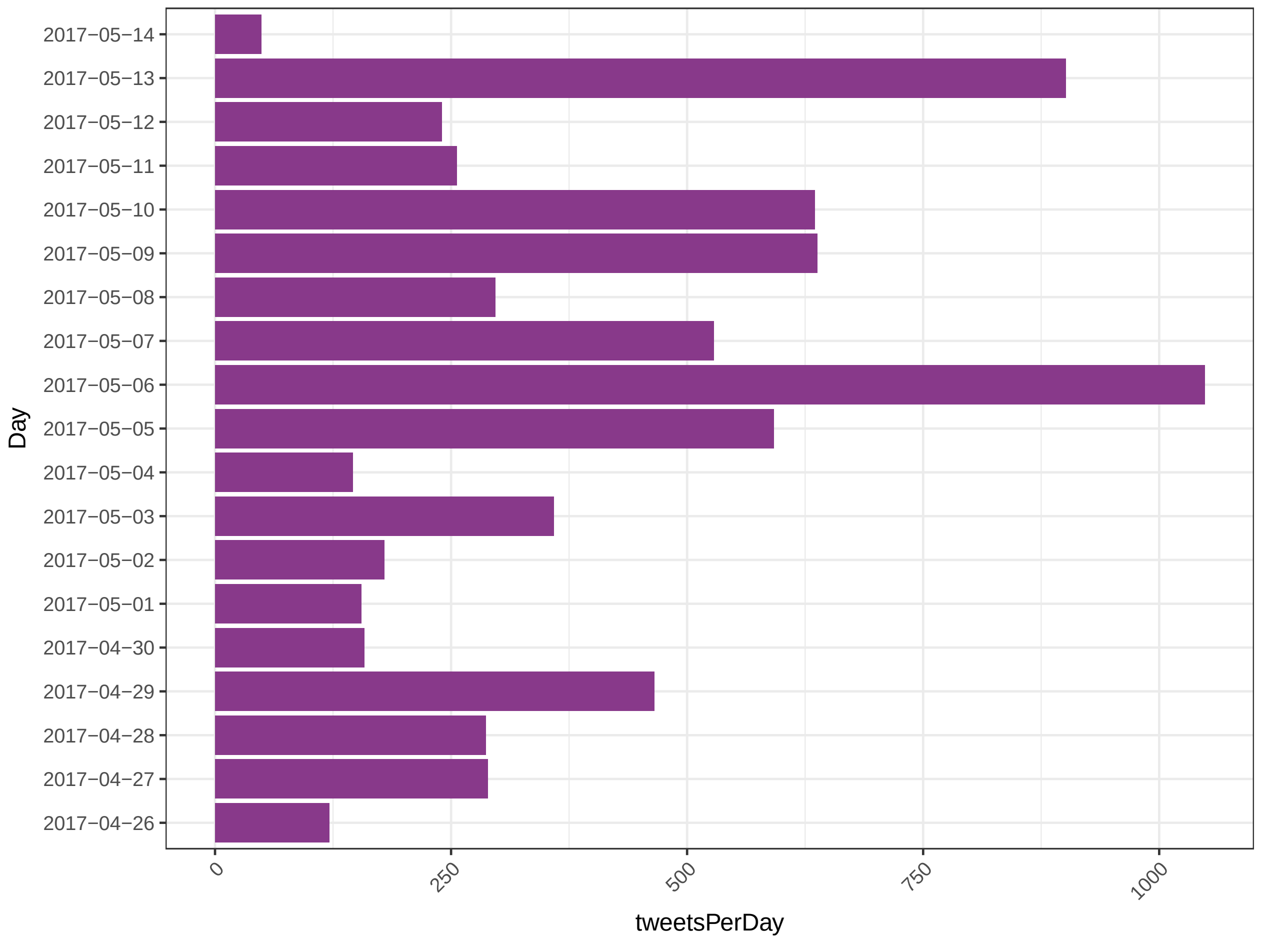


Figure 6: Total number of tweets per day

2.3.2 User's information

Another functionality of *twitteR* is to download the information about the users that wrote the the tweets. This can be done with the functions *getUser()* and for multiple lookups with *lookupUsers()* and . Of course, you are able to access user's information according to the user's permission. Therefore, you normally expect a lot of missing or fake data! The search can be done by providing on of *screenNames* or *userIDs*.

```
> head(combineTweets[, c("id", "screenName")])
```

id	screenName
860659555574984705	lukeistrouble
860659109741432832	ConnieBigBallz
860658889301348352	barely_sober_
860658879625089024	lukeistrouble
860658862780817408	junemberism
860658798150787072	DarrenTheDino

```
> # Create list of screenNames to look up
> # for
> screen_names <- unique(as.character(combineTweets$screenName))
```

There is a cap in the number of users that can be looked up in one go. The maximum is capped at 100 as stated on the Twitter Developer Documentation. Therefore, we need to look up *screen_names* in blocks and then combine the results.

```
> block_size <- round(seq(1, length(screen_names),
+     length.out = 50))
> # Size of the blocks
> diff(block_size)
>
> # Create list of screen names
> screen_names_blocks <- list()
> for (i in 1:(length(block_size) - 1)) {
+     start <- ifelse(block_size[i] == 1, 1,
+         block_size[i] + 1)
+     end <- block_size[i + 1]
+     screen_names_blocks[[i]] <- screen_names[start:end]
+ }
>
> # Wrapper for lookupUsers
> getUser_wrapper <- function(name_block) {
+     user_infos <- twitteR::lookupUsers(name_block,
+         includeNA = FALSE)
+     user_infosToDF <- twitteR::twListToDF(user_infos)
+     return(user_infosToDF)
+ }
>
> user_infos_list <- lapply(screen_names_blocks,
+     function(x) getUser_wrapper(x))
> user_infosDF <- do.call(rbind, user_infos_list)
>
> saveRDS(user_infosDF, file.path(dir, "Data",
+     "Users_infos.rds"))
```

Now we can merge tweets with user information.

```
> user_infosDF <- readRDS(file.path(dir, "Data",
+     "Users_infos.rds"))
>
> toDF_rupol_time <- combineTweets %>% tidyr::separate(created,
+     into = c("Day", "Time"), sep = " ", remove = FALSE)
> combine_data <- merge(toDF_rupol_time[, c("screenName",
+     "text", "Day", "Time", "retweetCount",
+     "isRetweet", "retweeted")], user_infosToDF[,
+     c("screenName", "description", "followersCount",
+         "friendsCount", "location")], all.x = TRUE,
+     all.y = TRUE)
> combine_data <- combine_data[!duplicated(combine_data),
+     ]
>
> write.csv(combine_data, file.path(dir, "Data",
+     "Users_infos_and_tweets.csv"), row.names = FALSE)
```

2.4 Some analysis

2.4.1 How many times is a drag queen mentioned daily?

Now that we have combined the tweets and the user's information we can start with some analyses. For example, how many times was each queen mentioned daily? Of course, each mention could have a positive or negative connotation but nonetheless this would tell us about the overall popularity of the queen.

First, let's load the data where we saved the queen names and their twitter screen name:

```
> queens <- read.csv(file.path(dir, "Data",
+     "queens_name.csv"))
> head(queens)
```

Queen.Name	Real.Name	Age	Hometown	Placement	Twitter.Name
Alexis Michelle	Alex Michaels	32	New York, New York		@AlexisLives
Farrah Moan	Cameron Clayton	22	Las Vegas, Nevada		@farrahrized
Nina Bo'nina Brown	Pierre Leverne Dease	34	Riverdale, Georgia		@atlsexyslim
Peppermint	Agnes Moore	37	New York, New York		@Peppermint247
Sasha Velour	Sasha Steinberg	29	Brooklyn, New York		@sasha_velour
Shea Coulez	Jaren Merrell	27	Chicago, Illinois		@SheaCoulee

We are going to do the search through the *grep* function and for every queen we need a unique vector of names. This is because a queen can be mentioned via her twitter name, her queen name or even her real name. For simplicity and to avoi overlap between queens we will perform the search using their drag queen names and their twitter names.

```
> combine_data <- read.csv(file.path(dir, "Data",
+     "Users_infos_and_tweets.csv"), stringsAsFactors = FALSE)
>
> # Separate their queen name into their
> # components
> queens <- queens %>% tidyr::separate(Queen.Name,
+     into = c("Queen1", "Queen2", "Queen3"),
+     sep = " ", remove = FALSE)
> head(queens[, c("Queen1", "Queen2", "Queen3",
+     "Twitter.Name")])
```

Queen1	Queen2	Queen3	Twitter.Name
Alexis	Michelle	NA	@AlexisLives
Farrah	Moan	NA	@farrahrized
Nina	Bo'nina	Brown	@atlsexyslim
Peppermint	NA	NA	@Peppermint247
Sasha	Velour	NA	@sasha_velour
Shea	Coulez	NA	@SheaCoulee

```
> # Wrapper function that creates a vector
> # of key names for every queen
> queen_vector <- function(x) {
+     vec <- c(x[c("Queen1", "Queen2", "Queen3",
+         "Twitter.Name")])
+     vec <- vec[!is.na(vec)]
+ }
>
> # List containing the vectors for every
> # queen
> queens_vecs <- apply(queens, 1, queen_vector)
> queens_grepKey_prepare <- lapply(queens_vecs,
+     function(x) paste0(x, collapse = "|"))
>
> # Set the encoding of the tweets as latin
> # to avoid issues with for example emoji
> Encoding(combine_data$text) <- "latin1"
> grep_queens <- lapply(queens_grepKey_prepare,
+     function(x) grep(x, combine_data$text))
> names(grep_queens) <- queens$Twitter.Name
> # Index referring to the raw in
> # combine_data where a queen was
> # mentioned
> head(grep_queens[[1]])
```

```
## [1]   2  21  62 172 179 202
```

```
> # Frequency of tweets per queen 1.
> # Exctract rows where a queen was
> # mentioned and extract only columns that
> # we need for this analysis
> freq_mention_Day <- lapply(grep_queens, function(x) {
+     mention_data <- combine_data[x, c("Day",
+         "Time", "text", "location", "followersCount",
+         "friendsCount", "retweetCount", "isRetweet",
+         "retweeted")]
+ })
> # 2. Combine mention for every queen into
> # a data.frame
> freq_mention_DayToDF <- do.call(rbind, freq_mention_Day)
> # 3. Creat a column $queen_name which
> # will tell us whose queen the tweet
> # belongs to
> number_mention <- sapply(freq_mention_Day,
+     function(x) nrow(x))
> freq_mention_DayToDF$queen_name <- rep(names(freq_mention_Day),
+     times = number_mention)
```

Now we can plot the number of time a queen was mentioned in a tweet every day.

```
> dailyMention <- freq_mention_DayToDF %>%
+     dplyr::group_by(Day, queen_name) %>%
+     dplyr::summarise(Nmention = length(text)) %>%
+     ggplot(aes(x = Day, y = Nmention, colour = queen_name,
+         group = queen_name)) + geom_line() +
+     theme_bw() + theme(axis.text.x = element_text(angle = 45,
+     hjust = 1)) + geom_vline(xintercept = c(4,
+     11, 18), linetype = "dotted")
> dailyMention
```

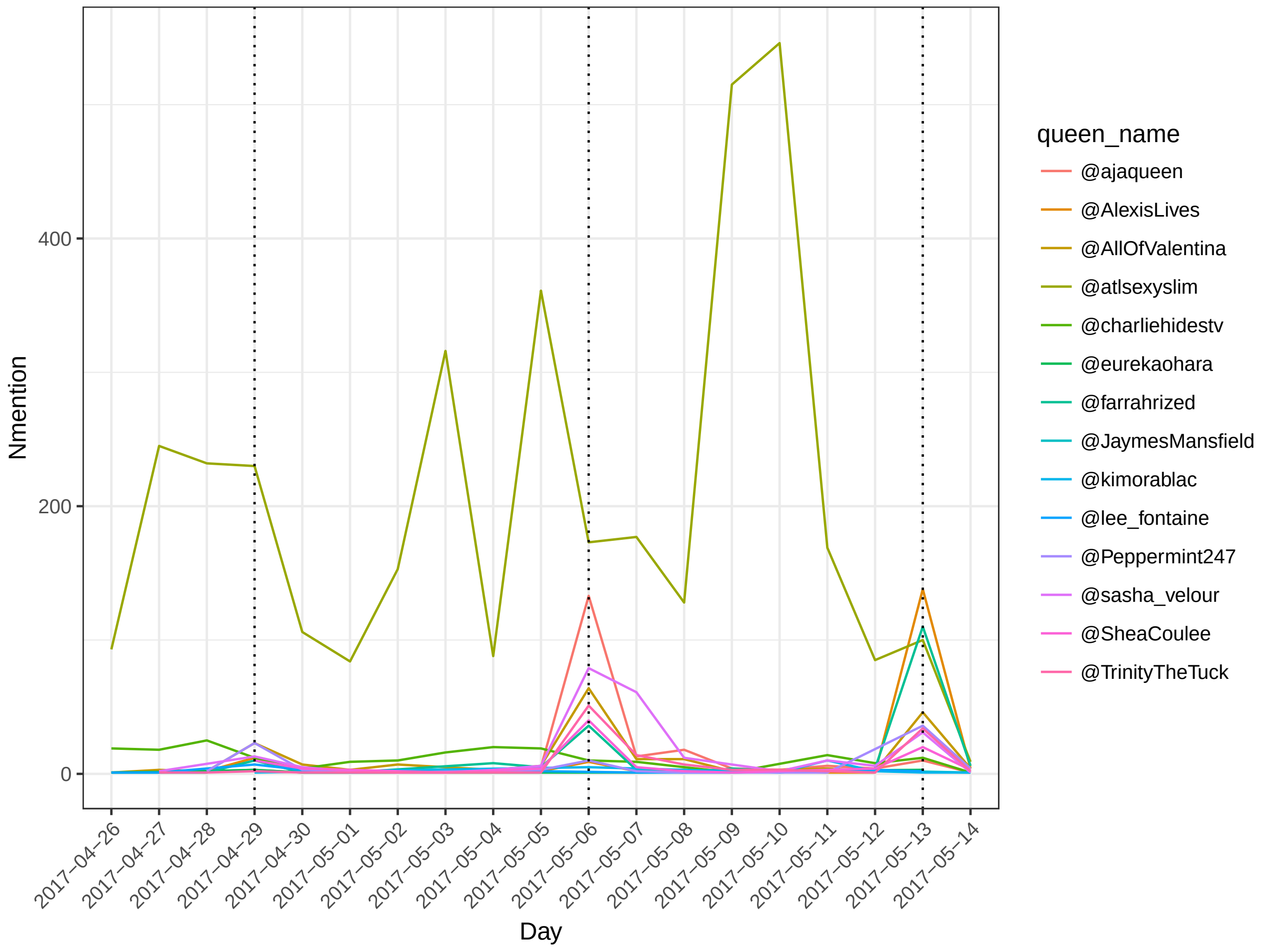


Figure 7: Total number of times that a queen was mentioned in a tweet.

```
> # ggplotly(dailyMention)
>
> # Episodes in America Airdate: April 28,
> # 2017 (29th in AU) Airdate: May 5th 2017
> # (6th in AU) Airdate: May 12th 2017
> # (13th in AU)
```

2.4.2 How come @atlsexyxlim has such huge number of tweets?

The answer is in the retweets!

```
> retweets <- freq_mention_DayToDF %>% dplyr::group_by(Day,
+     queen_name) %>% dplyr::summarise(NUniqueTweet = length(unique(text)),
+     Nretweet = sum(isRetweet)) %>% ggplot(aes(x = NUniqueTweet,
+     y = Nretweet, colour = queen_name)) +
+     geom_point() + theme_bw()
> retweets
```

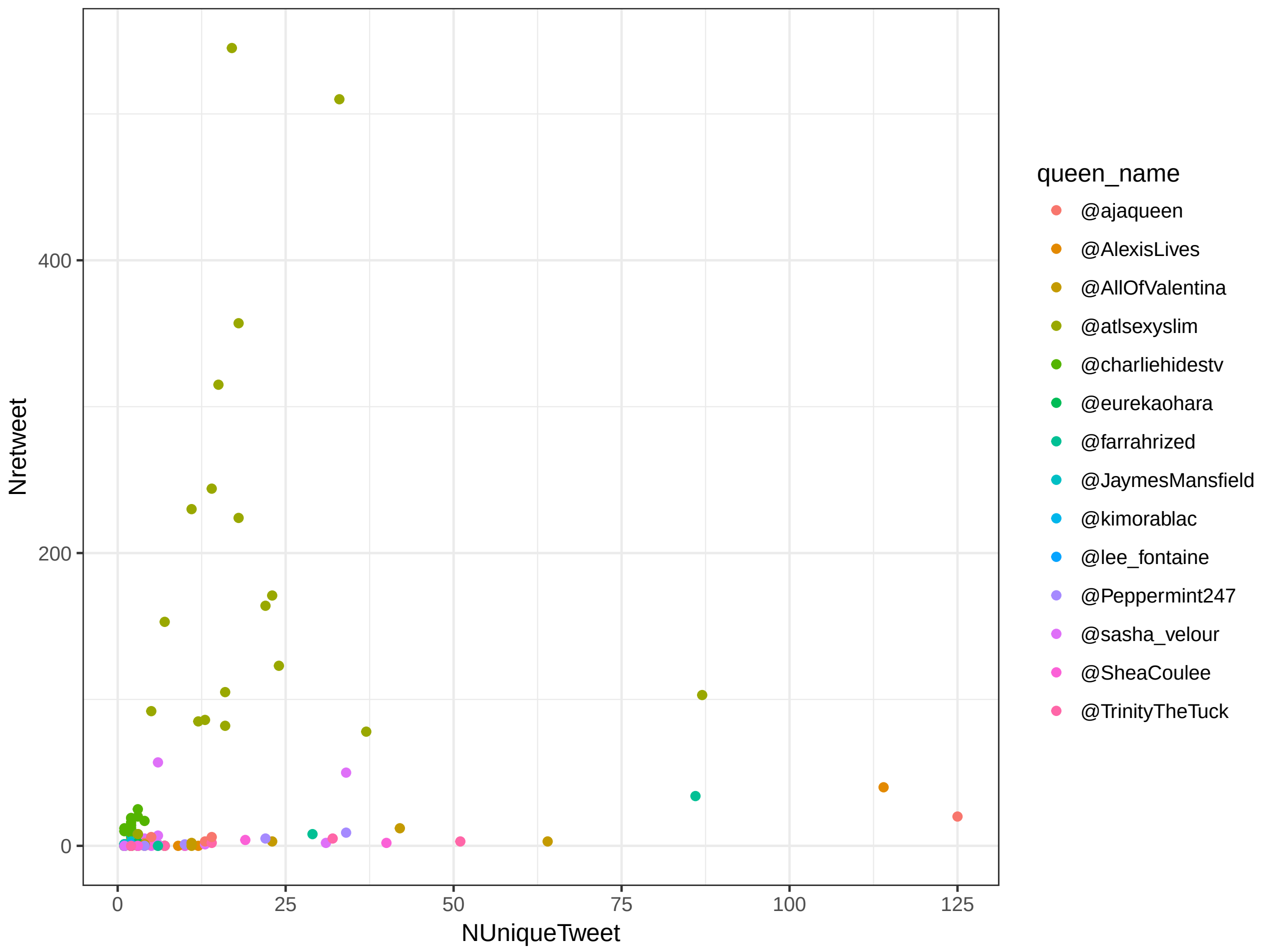


Figure 8: Number of retweets by number of unique tweets where a queen was mentioned per day.

```
> # ggplotly(retweets)
>
> atlslim <- subset(freq_mention_DayToDF, queen_name %in%
+     "@atlsexyslim")
```

2.4.3 What are the most common words used in @atlsexyslim tweets?

To answer this question we will generate a word cloud using @atlsexyslim tweets. At the beginning of this tutorial there is the list functions needed for this step.

```
> some_txt <- freq_mention_DayToDF$text[freq_mention_DayToDF$queen_name ==
+     "@atlsexyslim"]
> # Clean text remove punctuation
> some_txt = gsub("[[:punct:]]", "", some_txt)
> # remove numbers
> some_txt = gsub("[[:digit:]]", "", some_txt)
> # remove html links
> some_txt = gsub("http\\w+", "", some_txt)
> # remove unnecessary spaces
> some_txt = gsub("[ \t]{2,}", "", some_txt)
> some_txt = gsub("^\\s+|\\s+$", "", some_txt)
>
> mach_corpus = tm::Corpus(tm::VectorSource(some_txt))
>
> tdm = tm::TermDocumentMatrix(mach_corpus,
+     control = list(removePunctuation = TRUE,
+         stopwords = c("machine", "learning",
+             tm::stopwords("english")), removeNumbers = TRUE,
+         tolower = TRUE))
>
> # define tdm as matrix
> dm = as.matrix(tdm)
> # get word counts in decreasing order
> word_freqs = sort(rowSums(dm), decreasing = TRUE)
> # create a data frame with words and
> # their frequencies
> dm = data.frame(word = names(word_freqs),
+     freq = word_freqs)
```

Before plotting the word cloud it is always useful to check and maybe remove the first few words of the list which are probably going to be excpected words, like the hastags that we used for the search.

```
> head(dm)
```

	word	freq
rpdr	rpdr	7364
atlsexyslim	atlsexyslim	3711
dragqueen	dragqueen	3636
ninaboninabrown	ninaboninabrown	3440
music	music	854
time	time	791

```
> dm <- dm[-(1:5), ]
> # Plot the word cloud
> wordcloud::wordcloud(dm$word, dm$freq, random.order = FALSE,
+     max.words = 100, colors = brewer.pal(8,
+         "Dark2"))
```

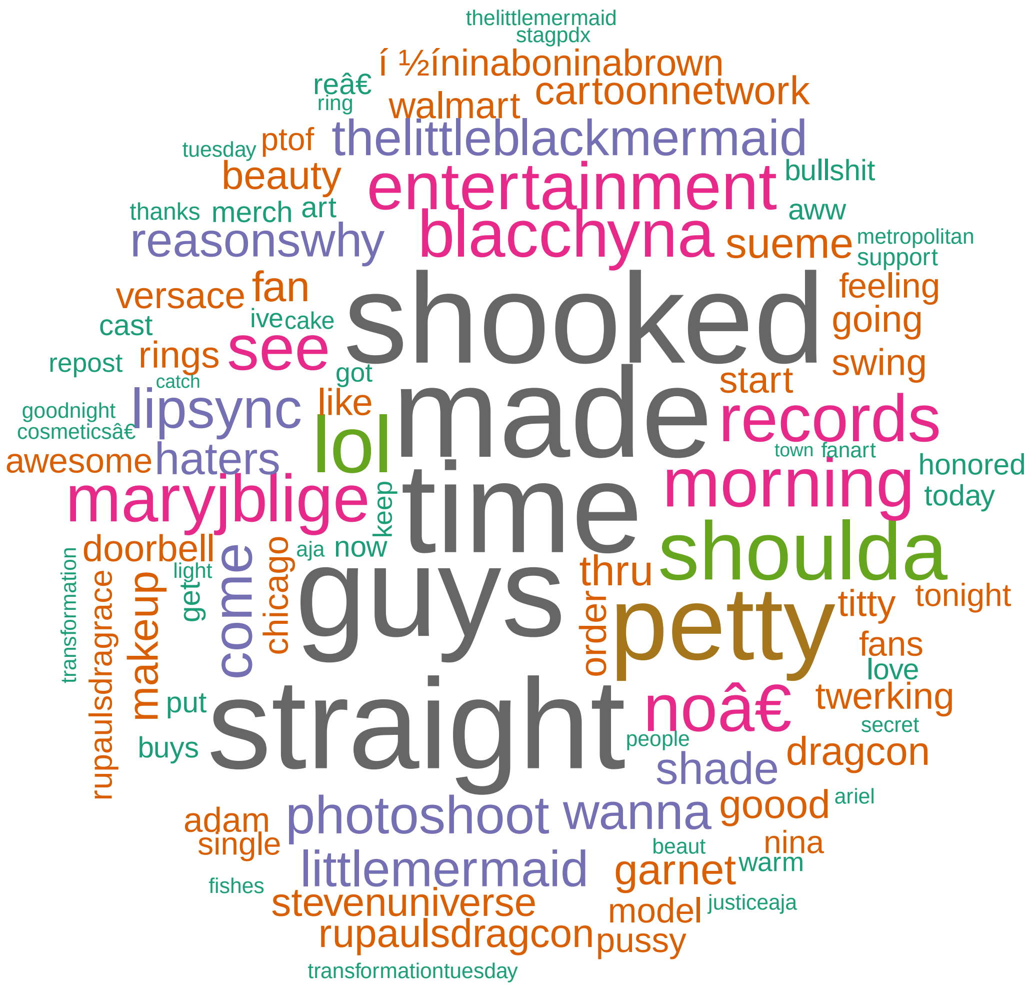


Figure 9: Wordcloud of the top 100 words for the tweets where @atlsexyxlim was mentioned.

2.4.4 Sentiment analysis

Sentiment analysis is defined as the process to computationally classify opinions from a text. Piece of text are sequences of qualitative information that can be seen as unstructured data. Sentiment analysis aims to classify the positivity or negativity of each sentence so that the results could be used quantitatively. To perform the following analysis we will use the package RSentiment.

```
> # Extract tweets and store them into a
> # character vector with a unique ID
> freq_mention_DayToDF$uniqueID <- rownames(freq_mention_DayToDF)
> some_txt <- freq_mention_DayToDF$text
> names(some_txt) <- freq_mention_DayToDF$uniqueID
```

Before we can process the data with *RSentiment* a set of data cleaning are necessary. Below is the list of the most important ones.

```
> # remove retweet entities
> some_txt = gsub("(RT|via)((?:\\b\\W*@\\w+)+)",
+     "", some_txt)
> # remove at people
> some_txt = gsub("@\\w+", "", some_txt)
> # remove punctuation
> some_txt = gsub("[[:punct:]]", "", some_txt)
> # remove numbers
> some_txt = gsub("[[:digit:]]", "", some_txt)
> # remove html links
> some_txt = gsub("http\\w+", "", some_txt)
> # remove unnecessary spaces
> some_txt = gsub("[ \t]{2,}", "", some_txt)
> some_txt = gsub("^\\s+|\\s+$", "", some_txt)
>
> # convert every word to lower case define
> # 'tolower error handling' function
> try.error = function(x) {
+     # create missing value
+     y = NA
+     # tryCatch error
+     try_error = tryCatch(tolower(x), error = function(e) e)
+     # if not an error
+     if (!inherits(try_error, "error"))
+         y = tolower(x)
+     # result
+     return(y)
+ }
> # lower case using try.error with sapply
> some_txt = sapply(some_txt, try.error)
>
> # remove NAs in some_txt
> some_txt = some_txt[!is.na(some_txt)]
> names(some_txt) = NULL
> some_txt <- gsub("\n", "", some_txt)
```

Now that the text is clean we can run the sentiment analysis with the *calculate_sentiment()* function!

```
> emotion <- RSentiment::calculate_sentiment(some_txt)
> emotion$uniqueID <- names(some_txt)
> combine_sentiment <- cbind(freq_mention_DayToDF,
+     emotion)
>
> ggplot(combine_sentiment, aes(x = queen_name,
+     fill = sentiment)) + geom_bar(stat = "count",
+     position = "fill") + coord_flip()
```