Analysing Twitter for Ubisoft

Ryan Greenup

May 5, 2020

Contents

8.1 Analysing the Relationship Between Friends and Followers for Twitter Users

8.1.1 Retrieve the posts from Twitter

relevant posts can be retrieved from twitter by utilising the `rtweet` package, packages can be loaded for use in **R** thusly:

The `rtweet` API will search for tweets that contain all the words of a query regardless of uppercase or lowercase usage [5].

In order to leverage the *Twitter* API it is necessary to use tokens provided through a *Twitter* developer account:

and hence all tweets containing a mention of *Ubisoft* can be returned and saved to disk as shown in listing 3:

8.2.2 Count of Followers and Friends

In order to identify the number of users that are contained in the *tweets* the `unique()` function can be used to return a vector of names which can then be passed as an index to the vector of counts as shown in listing 4, this provides that 81.7% of the tweets are by unique users.

```r
# Load Packages
 ↪  ------------------------------------------------------------
setwd("~/Dropbox/Notes/DataSci/Social_Web_Analytics/SWA-Project/scripts
 ↪  /")

if (require("pacman")) {
  library(pacman)
} else{
  install.packages("pacman")
  library(pacman)
}

pacman::p_load(xts, sp, gstat, ggplot2, rmarkdown, reshape2,
               ggmap, parallel, dplyr, plotly, tidyverse,
               reticulate, UsingR, Rmpfr, swirl, corrplot,
               gridExtra, mise, latex2exp, tree, rpart,
               lattice, coin, primes, epitools, maps, clipr,
               ggmap, twitteR, ROAuth, tm, rtweet, base64enc,
               httpuv, SnowballC, RColorBrewer, wordcloud,
               ggwordcloud, tidyverse, boot)
```

Listing 1: Load the Packages for ***R***

8.1.3 Summary Statistics

The average number of friends and followers from users who posted tweets mentioning *Ubisoft* can be returned using the `mean()` as shown in listing 5 this provides that on average each user has 586 friends and 63,620 followers.

8.1.4 Above Average Followers

Each user can be compared to the average number of followers, by using a logical operator on the vector (e.g. `y > ybar`), this will return an output of logical values. ***R*** will coerce logicals into 1/0 values meaning that the mean value will return the proportion of `TRUE` responses as shown in listing 6. This provides that:

- 2.4% of the have identified have an above average **number of followers**.
- 20.6% of the users identified have an above average **number of friends**.

```r
# Set up Tokens ==============================================================

options(RCurlOptions = list(
  verbose = FALSE,
  capath = system.file("CurlSSL", "cacert.pem", package = "RCurl"),
  ssl.verifypeer = FALSE
))

setup_twitter_oauth(
  consumer_key = "************************",
  consumer_secret = "**************************************************",
  access_token = "**************************************************",
  access_secret = "******************************************"
)

# rtweet ======================================================================
tk <-    rtweet::create_token(
  app = "SWA",
  consumer_key    = "************************",
  consumer_secret = "**************************************************",
  access_token    = "**************************************************",
  access_secret   = "******************************************",
  set_renv        = FALSE
```

Listing 2: Import the twitter tokens (redacted)

```r
 n <- 1000
 tweets.company <- search_tweets(q = 'ubisoft', n = n, token = tk,
                                 include_rts = FALSE)
 save(tweets.company[,], file = "resources/Download_1.Rdata")
```

Listing 3: Save the Tweets to the HDD as an `rdata` file

```
(users <- unique(tweets.company$name)) %>% length()
x <- tweets.company$followers_count[duplicated(tweets.company$name)]
y <- tweets.company$friends_count[duplicated(tweets.company$name)]

## > [1] 817
```

Listing 4: Return follower count of twitter posts

```
x<- rnorm(090)
y<- rnorm(090)
(xbar <- mean(x))
(ybar <- mean(y))

## > [1] 4295.195
## > [1] 435.9449
```

Listing 5: Determine the average number of friends and followers

```
(px_hat <- mean(x>xbar))
(py_hat <- mean(y>ybar))

## > [1] 0.0244798
## > [1] 0.2729498
```

Listing 6: Calculate the proportion of users with above average follower counts

8.1.5 Bootstrap confidence intervals

a/b.) Generate a bootsrap distribution

A bootstrap assumes that the population is an infinitely large repetition of the sample and may be produces with respect to follower counts by resampling with replacement/repetition and plotted using the `ggplot2` library as deomonstrated in listings 7 and .1 and shown in figure 1.

This shows that the population follower counts is a non-normal skew-right distribution, which is expected because the number of friends is an integer value bound by zero [7].

```
## Resample the Data
(bt_pop <- sample(x, size = 10^6, replace = TRUE)) %>% head()

## > [1]   7 515 262 309 186 166
```

Listing 7: Bootstrapping a population from the sample.

```
## Make the Population
bt_pop_data <- tibble("Followers" = bt_pop)
ggplot(data = bt_pop_data, aes(x = Followers)) +
  geom_histogram(aes(y = ..density..), fill = "lightblue", bins = 35,
  ↪  col = "pink") +
  geom_density(col = "violetred2") +
  scale_x_continuous(limits = c(1, 800)) +
  theme_bw() +
  labs(x = "Number of Followers", y = "Density",
       title = "Bootstrapped population of Follower Numbers")
```

c.) Estimate a Confidence Interval for the population mean Follower Counts

In order to perform a bootrap for the population mean value of follower counts it is necessary to:

1. Resample the data with replacement
   - i.e. randomly select values from the sample allowing for repetition
2. Measure the statistic of concern
3. Replicate this a sufficient number of times
   - i.e. Greater than or equal to 1000 times [2, Ch. 5]

This is equivalent to drawing a sample from a population that is infinitely large and constructed of repetitions of the sample. This can be performed in ***R*** as shown in listing 8.

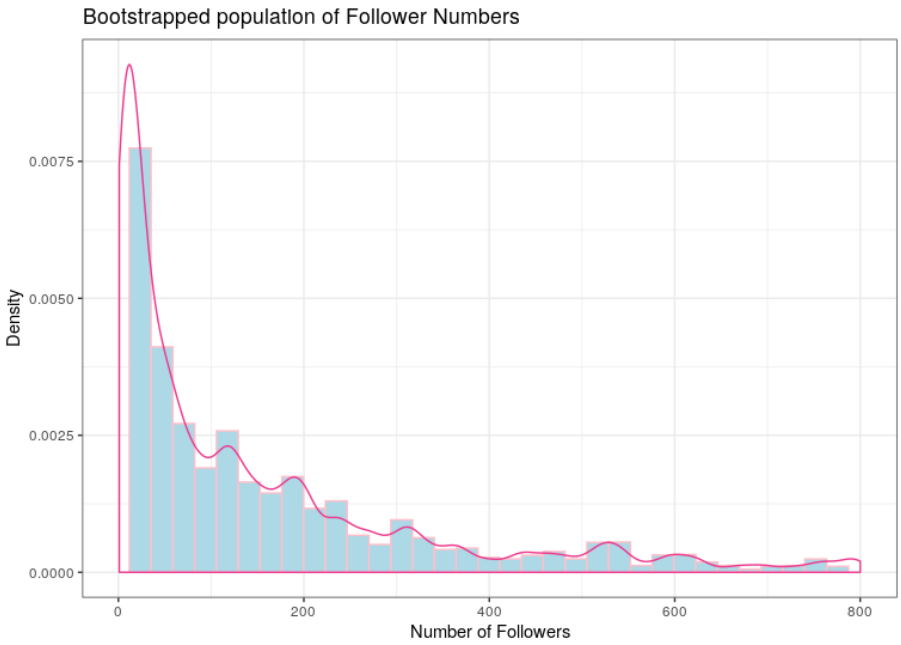

Figure 1: Histogram of the bootrapped population of follower counts

```
xbar_boot_loop <- replicate(10^3, {
  s <- sample(x, replace = TRUE)
  mean(s)
  })
quantile(xbar_boot_loop, c((1-0.97)/2, (1+0.97)/2))

##       1.5%      98.5%
##   588.4189 10228.7352
```

Listing 8: Confidence Interval of Mean Follower Count in Population

A 97% probability interval is such that a sample drawn from a population will contain the population mean in that interval 97% of the time, this means that it may be concluded with a high degree of certainty that the true population mean lies between 588 and 10228.

1. Alternative Approaches If this data was normally distributed it may have been appropriate to consider bootstrapping the standard error and using a t distribution, however it is more appropriate to use a percentile interval for skewed data such as this, in saying that however this method is not considered to be very accurate in the literature and is often too narrow. [3, Section 4.1]

    - It's worth noting that the normal t value bootstrap offers no advantage over using a t distribution (other than being illustrative of bootstrapping generally) [3, Section 4.1]

    The `boot` package is a bootstrapping library common among authors in the data science sphere [4, p. 295] [10, p. 237] that implements confidence intervals consistent with work by Davison and Hinkley [8] in there texbook *Bootstrap Methods and their Application*. In this work it is provided that the BC_a method of constructing confidence intervals is superior to mere percentile methods in terms of accuracy [2, Ch. 5], a sentiment echoed in the literature. [1, 2, Ch. 5]

    Such methods can be implemented in ***R*** by passing a function to the the `boot` call as shown in listing 9. This provides a broader interval, providing that the true confidence interval could lie between 1079 and 16227 followers.

```r
xbar_boot <- boot(data = x, statistic = mean_val, R = 10^3)
boot.ci(xbar_boot, conf = 0.97, type = "bca", index = 1)

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = xbar_boot, conf = 0.97, type = "bca", index = 1)
##
## Intervals :
## Level       BCa
## 97%   ( 1079, 16227 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
## Warning message:
## In norm.inter(t, adj.alpha) : extreme order statistics used as
↪  endpoints
```

Listing 9: Bootstrap of population mean follower count implementing the BC_a method

d.) Estimate a Confidence Interval for the population mean Friend Counts

A Confidence interval for the population mean friend counts may be constructed in a like wise fashion as shown in listings 10. This provides that the 97% confidence interval for the population mean friend

count is between 384 and 502 (or 387 and 496 if the BC_a method used, they're quite close and so the more conservative percentile method will be accepted).

FIXME 8.1.6 Estimate a 97% Confidence Interval for the High Friend Count Proportion

In order to bootstrap a confidence interval for the proportion of users with above average follower counts, repeteadly draw random samples from an infinitely large population composed entirely of the sample, and record the sampled proportion. this can be acheived by resampling the observations of above and below as shown in listing 11.

This provides that:

- The 97% confidence interval for the population proportion of users that have an above average number of friends is between 0.24 and 0.31.
  - i.e. The probability of any given sample containing the population mean within this interval would be 97%, although that doesn't however mean that there is a 97% probability that this interval contains the value, merely that we may be 97% *confident*

8.1.7 Is the Number of Friends Independent to the Number of Followers

One method to determine whether or not the number of followers is independent of the number of friends is to bin the counts and determine whether or not the distribution of users across those counts is consistent with the hypothesis of independence.

Bin the Follower and Friend Categories

The counts may be binned by performing a logical interval test as shown in listing 12.

Find the Group frequency

These values may be tabluated in order to count the occurence of users among these categories as shown in listing 13 and table 1.

Table 1: Table of Binned Friend and Follower counts, transposed relative to code.

	Followers	***Friends***
Tens	421	262
Hundreds	317	476
1 - Thousands	39	47
2 - Thousands	11	15
3 - Thousands	9	6
4 - Thousands	2	9
5 Thousand or More	18	2

```r
# d.) Estimate a Confidence Interval for the populattion mean Friend Count ===
# Using a Percentile Method ##################################################
ybar_boot_loop <- replicate(10^3, {
  s <- sample(y, replace = TRUE)
  mean(s)
  })
quantile(ybar_boot_loop, c(0.015, 0.985)

# Using BCA Method ###########################################################
mean_val <- function(data, index) {
  X = data[index]
  return(mean(X))
}

xbar_boot <- boot(data = y, statistic = mean_val, R = 10^3)
boot.ci(xbar_boot, conf = 0.97, type = "bca", index = 1)


##     1.5%    98.5%
## 383.7619 501.5903
##
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = xbar_boot, conf = 0.97, type = "bca", index = 1)
##
## Intervals :
## Level       BCa
## 97%   (386.8, 496.7 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Listing 10: Bootstrap of population mean follower count

```r
# 8.1.6 High Friend Count Proportion ----------------------------------------
prop <- factor(c("Below", "Above"))
## 1 is above average, 2 is below
py_hat_bt <- replicate(10^3, {
  rs      <- sample(c("Below", "Above"),
                    size = length(y),
                    prob = c(py_hat, 1-py_hat),
                    replace = TRUE)
isabove <- rs == "Above"
mean(isabove)
})
quantile(py_hat_bt, c(0.015, 0.985))


##      1.5%     98.5%
## 0.2399021 0.3072215
## > > > . + > > >
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = py_hat_boot, conf = 0.97, type = "bca")
##
## Intervals :
## Level       BCa
## 97%   ( 0.2399,  0.3072 )
## Calculations and Intervals on Original Scale
```

Listing 11: Bootstrap of Proportion of Friends above average

```r
## Assign Categories
x_df <- data.frame(x)
x_df$cat[0       <= x_df$x & x_df$x < 100] <- "Tens"
x_df$cat[100     <= x_df$x & x_df$x < 1000] <- "Hundreds"
x_df$cat[1000    <= x_df$x & x_df$x < 2000] <- "1Thousands"
x_df$cat[2000    <= x_df$x & x_df$x < 3000] <- "2Thousands"
x_df$cat[3000    <= x_df$x & x_df$x < 4000] <- "3Thousands"
x_df$cat[4000    <= x_df$x & x_df$x < 5000] <- "4Thousands"
x_df$cat[5000    <= x_df$x & x_df$x < Inf] <- "5ThousandOrMore"

### Make a factor
x_df$cat <- factor(x_df$cat, levels = var_levels, ordered = TRUE)

### Determine Frequencies
(x_freq <- table(x_df$cat) %>% as.matrix())

## ** b) Find the Friend Count Frequency ==========================================
## Assign Categories
y_df <- data.frame(y)
y_df$cat[0       <= y_df$y & y_df$y < 100] <- "Tens"
y_df$cat[100     <= y_df$y & y_df$y < 1000] <- "Hundreds"
y_df$cat[1000    <= y_df$y & y_df$y < 2000] <- "1Thousands"
y_df$cat[2000    <= y_df$y & y_df$y < 3000] <- "2Thousands"
y_df$cat[3000    <= y_df$y & y_df$y < 4000] <- "3Thousands"
y_df$cat[4000    <= y_df$y & y_df$y < 5000] <- "4Thousands"
y_df$cat[5000    <= y_df$y & y_df$y < Inf]  <- "5ThousandOrMore"

### Make a factor
y_df$cat <- factor(y_df$cat, levels = var_levels, ordered = TRUE)

### Determine Frequencies
(y_freq <- table(y_df$cat) %>% as.matrix())
```

Listing 12: Use Logical Test to Assign observations into bins

```r
vals <- t(cbind(x_freq, y_freq))
rownames(vals) <- c("Followers.x", "followers.y")
vals

##             Tens Hundreds 1Thousands 2Thousands 3Thousands 4Thousands
## Followers.x  421      317         39         11          9          2
## followers.y  262      476         47         15          6          9
##             5ThousandOrMore
## Followers.x              18
## followers.y               2
```

Listing 13: Tabulate the binned counts for the distribution of users among among amount and status.

Find the Expected Counts under each group and test for independence

The expected count of each cell, under the assumption that the two metrics are independent, will be the proportion users per bracket multiplied by the number of users in that status group. This implies that any cell will be:

- the product of the row sum, multiplied by the column sum divided by the number of counts.

This can be equivalently expressed as an outer product as shown in equation (1), in ***R*** this operation is denoted by the %o% operator, which is shorthand for the `outer()` function, this and other summary statistics may be evaluated as shown in listing 14.

The outer product is such that:

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathsf{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}.$$

This means the matrix of expected frequencies can be expressed as an outer product thusly:

$$\tilde{\mathbf{e}} = \frac{1}{n} \times \begin{bmatrix} \sum_{j=1}^{n} [o_{1j}] \\ \sum_{j=1}^{n} [o_{2j}] \\ \sum_{j=1}^{n} [o_{3j}] \\ \sum_{j=1}^{n} [o_{4j}] \\ \vdots \\ \sum_{j=1}^{n} [o_{nj}] \end{bmatrix} \begin{bmatrix} \sum_{j=1}^{n} [o_{i1}] \\ \sum_{j=1}^{n} [o_{i2}] \\ \sum_{j=1}^{n} [o_{i3}] \\ \cdots \\ \sum_{j=1}^{n} [o_{in}] \end{bmatrix}^{\mathrm{T}} \tag{1}$$

1. Testing Independence In order to test whether or not the distribution of users among brackets is independent of being a follower or friend a χ^2 test may be used, this can be evaluated from a model or simulated, in ***R***, the simulated test is shown in listing 15, this provides a p -value $<$ 0.0005, which means that the hypothesis of independence may be rejected with a high degree of certainty.

```r
## ***** Calculate Summary Stats
n <- sum(vals)
bracket_prop <- colSums(vals) / n
metric_prop  <- rowSums(vals) / n
o <- vals
e <- rowSums(vals) %o% colSums(vals) / n
chi_obs <- sum((e-o)^2/e)
```

Listing 14: Calculate Expected frequency of values under the assumption of independence.

```r
chisq.test(vals, simulate.p.value = TRUE)


## ^^IPearson's Chi-squared test with simulated p-value (based on 2000
## ^^Ireplicates)
##
## data:  vals
## X-squared = 88.109, df = NA, p-value = 0.0004998
```

Listing 15: Chi-Square testing for independence between friend and follower bin categories.

(a) From First Principles The χ^2 statistic may be performed from first principles by randomly sampling the values at the rate at which they occured, tabulating those counts, measuring the χ^2 -value and then repeating this many times.

Because the samples are random they must be independent and average number of positives is hence an estimate for the *FPR*, which is in turn an estimate for the p -value. This technique is demonstrated in listing 16, the p-value being returned as 0.0004, this value is consistent with the value produced by ***R***'s built in `chisq.test` function and so is accepted.

FIXME Conclusion

The p -value measures the probability of rejecting the null hypothesis when it is true, i.e. the probability of a detecting a *false positive*, a very small p -value is hence good evidence that the null hypothesis should be rejected (because doing so would unlikely to be a mistake).

In saying that however the p -value is distinct from the *power* statistic, which is a measure of */the probability of accepting the alternative hypothesis* when it is true, a low p -value is not a measurement of the probability of being correct.

Hence me way conclude, with a high degree of certainty, that the follower and friend counts are not independent of one another.

8.2 Finding Themes in tweets

```r
## ***** Create Vectors of factor levels
brackets <- unique(x_df$cat)
metrics <- c("follower", "friend")

## ***** Simulate the data Assuming H_0
## I.e. assuming that the null hypothesis is true in that
## the brackets assigned to followers are independent of the friends
## (this is a symmetric relation)

s <- replicate(10^4,{
  ## Sample the set of Metrics
  m <- sample(metrics, size = n, replace = TRUE, prob = metric_prop)

  ## Sample the set of Brackets (i.e. which performance bracket the
  ↪  user falls in)
  b <- sample(brackets, size = n, replace = TRUE, prob = bracket_prop)

  ## Make a table of results
  o <- table(m, b)
  o

  ## Find What the expected value would be
  e_sim <- t(colSums(e) %o% rowSums(e) / n)

  ## Calculate the Chi Stat
  chi_sim <- sum((e_sim-o)^2/e_sim)
  chi_sim

  ## Is this more extreme, i.e. would we reject null hypothesis?
  chi_sim > chi_obs

})

mean(s)

## [1] 4e-04
```

Listing 16: Performing a χ^2 statistic from first principles

8.2.8 Find Users with Above Average Friend Counts

Users with Above average Friend Counts can be identified by filtering the tweets data frame for two conditions:

1. non-duplicated `user-id`
2. `friend_count` greater than average

This can be acheived easily using the `dplyr` package as shown in 17, the top 20 of these users are shown in table 4 of the appendix

```
select <- dplyr::select
filter <- dplyr::filter
interested_vars <- c("user_id", "friends_count")
(friend_counts <- tweets.company %>%
  select(interested_vars) %>%
  filter(!duplicated(user_id)))

(high_friends <- friend_counts %>%
  filter(friends_count > mean(friends_count, na.rm = TRUE)))

## Export Friends List
write.csv(high_friends[order(
  high_friends$friends_count,
  decreasing = TRUE),], file = "/tmp/highfriend.csv")
```

Listing 17: Use `dplyr` to Filter for Users with a high Friend Count

8.2.9 Find Users with Below Average Friend Counts

Users with high friends may be determined by a similar method (or by taking the complement of the high friends) as shown in listing 18, the lowest 20 of these users are shown in table 5 of the appendix.

8.2.10 Find the *Tweets* corresponding to users with high or low friend counts

The tweets corresponding to users with high and low friend counts can be identified by filtering the dataframe based on the friend count and using that to the index the tweets from the data frame [1],

[1] This works because the `tm` package preserves the order of the data, this can be confirmed by using a dataframe source as opposed to a vector source (e.g. in listing 20) and comparing the ID's before/after transformation.

```
(low_friends <- friend_counts %>%
  filter(friends_count <= mean(friends_count, na.rm = TRUE)))

 low_friends <- low_friends[order(
   low_friends$friends_count,
   decreasing = TRUE),]

## Export Users
write.csv(low_friends[order(
  low_friends$friends_count,
  decreasing = FALSE),], file = "/tmp/lowfriend.csv")
```

Listing 18: Use `dplyr` to Filter for Users with a low Friend Count

alternatively it is possible to test whether or not the ID of a user appears in the high or low vector set using the `%in%` operator as shown in listing 19.

8.2.11 Clean the tweets

Create a Corpus Object

In order to clean the tweets it is necessary to create a corpus object as shown in listing 20, it is possible to pass a dataframe source in order to include the user ID, this isn't strictly necessary however because the `tm` package preserves order when performing transformations.

Next it is necessary to choose an enoding, a primary consideration of this is whether or not the use of *emoji* characters will influence the model performance. There is research to suggest that Emoji's can be used as predictive features [6] and that they can improve *sentiment analysis* models [9] that implement a *bag of words* approach. For these reasons *emoji* characters will be preserved and UTF-8 implemented.

In order to encode the data as *UTF-8*, the `iconv` function can be used as shown in listing 21.

Process the tweets

Before analysis the tweets should be modified to remove characters that may interfere with categorising words, this is referred to as cleaning, in particular the following should be implemented:

1. Remove URL's
2. Remove Usernames
3. remove numbers
4. remove punctuation
5. remove whitespace
6. case fold all characters to lower case

```r
## Method 1
friend_test <- tweets.company$friends_count >
↪  mean(tweets.company$friends_count)
tweets_high <- tweets.company$text[friend_test]
tweets_low <- tweets.company$text[!friend_test]

## Method 2
↪      :15b5a74:
tweets_high <- tweets.company$text[tweets.company$user_id %in%
↪  high_friends$user_id]
tweets_low  <- tweets.company$text[tweets.company$user_id %in%
↪  low_friends$user_id]
tweets <- c(tweets_high, tweets_low)

## Mark as High or Low
tweets_low <- cbind(tweets_low, rep("Low_Friend", length(tweets_low)))
tweets_high <- cbind(tweets_high, rep("High_Friend",
↪  length(tweets_high)))
tweets <- as.data.frame(rbind(tweets_high, tweets_low))
tweets$Friend_Status  <- factor(tweets$Friend_Status)
```

Listing 19: Identify tweets corresponding to users with high and low friend counts

```r
tweet_source <- tm::VectorSource(tweets$text)
tweet_corpus <- tm::Corpus(x = tweet_source)
```

Listing 20: Create a Corpus from the tweets

```r
encode <- function(x) {
    iconv(x, to = "UTF-8")
#    iconv(x, to = "latin1")
  #  iconv(x, to = "ASCII")
}

tweet_corpus <- tm_map(x = tweet_corpus, FUN = encode)
tweet_corpus_raw <- tweet_corpus
```

Listing 21: Encode the Data as UTF-8

7. remove a set of stop words

8. reduce each word to its stem

In particular it is important to reduce words to lower case before removing stop words otherwise an unorthodox use of capitalisation may prevent the word from being removed throughout.

The stop word `ubisoft` will also be used, this was the query term so it's expected to turn up at a very high frequency, the words `can` and `'s` also occured quite frequently and so were removed.

The cleaning can be implemented by mapping functions over the corpus, which is fundamentally a list, this can be performed via the `tm_map` function as shown in listing 22.

```
1  mystop <- c(stopwords(), "s", "can", "ubisoft", "@ubisoft",
   ↪  "#ubisoft")# <<stphere>>
2
3  clean_corp <- function(corpus) {
4    ## Remove URL's
5    corpus <- tm_map(corpus,content_transformer(function(x)
     ↪  gsub("(f|ht)tp(s?)://\\S+","",x)))
6    ## Remove Usernames
7    corpus <- tm_map(corpus,content_transformer(function(x)
     ↪  gsub("@\\w+","",x)))
8    ## Misc
9    corpus <- tm_map(corpus, FUN = removeNumbers)
10   corpus <- tm_map(corpus, FUN = removePunctuation)
11   corpus <- tm_map(corpus, FUN = stripWhitespace)
12   corpus <- tm_map(corpus, FUN = tolower)
13   corpus <- tm_map(corpus, FUN = removeWords, mystop)
14   ## stopwords() returns characters and is fead as second argument
15   corpus <- tm_map(corpus, FUN = stemDocument)
16   return(corpus)
17 }
18
19 tweet_corpus_clean <- clean_corp(tweet_corpus)
```

Listing 22: Use the `tm_map` function to clean the tweets

8.2.12 Display the first two tweets before/after processing

The tweets can be viewed from inside the corpus by selecting with the `[` function [2] as demonstrated in listing 23, the first *tweet* was rendered empty by the processing and the following two tweets were:

[2]The `[` function is actually shorthand for `Extract()`, most things in ***R*** are functions, this is similar to *LISP* and has to do with the origins of the language, e.g. `sum(1:10) == (sum (1:10))`, also relevant see the relevant xkcd in figure 9.

- ***Pre-Processing***
  - "Today was the first time in over a month that I have gone 24 hours without checking the coronavirus death toll. Thanks `@Ubisoft`."
  - "`@btwimskrank` `@TheDivisionGame` `@UbiMassive` `@Ubisoft` `@jgerighty` `@hamishbode` `@Tideman92` `@janeyo_jane` `@slimjd` Very odd... I'll even post a video about it."
- ***Post-Processing***
  - "today first time month gone hour without check coronavirus death toll thank"
  - "odd ill even post video"

```
tweet_corpus_raw[[1]]$content
tweet_corpus_clean[[1]]$content
tweet_corpus_raw[[2]]$content
tweet_corpus_clean[[2]]$content
tweet_corpus_raw[[3]]$content
tweet_corpus_clean[[3]]$content
```

Listing 23: Load the Packages for ***R***

8.2.13 Create a Term Document Matrix

Apply Weighting Manually

1. Create Term Documen Matrix A term Document matrix (and it's transpose) can be constructed from a `corpus` using the `tm:TermDocumentMatrix` function as shown in listing 24.

```
tweet_matrix_tdm <- TermDocumentMatrix(tweet_corpus_clean)
tweet_matrix_dtm <- DocumentTermMatrix(tweet_corpus_clean)
```

Listing 24: Load the Packages for ***R***

2. Apply TF-IDF Weighting Weighted term frequency is defined as shown in equation (2), where:
   - $f_{d,t}$ is the frequency of a given term t in the a document d
   - $w_{d,t}$ is the weight of a given term t in the a document d
   - N is the number of documents
   - f_t is the number of documents containing t

$$
\begin{aligned}
w_{d,t} &= \mathrm{TF}_t \times \mathrm{IDF}_{d,t} \\
&= \log_e\left(f_{d,t}+1\right)\log_e\left(\frac{N}{f_t}\right)
\end{aligned}
\tag{2}
$$

This would requre multiplying each term of each row of the TF_t matrix by the corresponding vector element of $\mathrm{IDF}_{d,t}$, this can be implemented by taking the matrix product of a diagonalised matrix, this is shown in listing 25.

```
N <- nrow(as.matrix(tweet_matrix_dtm))   # Number of Documents
ft=colSums(as.matrix(tweet_matrix_dtm) > 0) #in how many documents term
 ↪  t appeared in,

TF <- log(as.matrix(tweet_matrix_dtm) + 1)  # built in uses log2()
IDF <- log(N/ft)

tweet_weighted           <- TF %*% diag(IDF)
colnames(tweet_weighted) <- colnames(tweet_matrix_dtm)
```

Listing 25: Apply TF-IDF Weigting

There is however a function built in to the `tm` package that will weight term document matrices and this will instead be implented to analyse the data because it will produce more maintainable code.

Apply Weighting with Built in Function

In order to create a term document matrix (and its transpose) with TF-IDF weighted values, the `weighting` argument may be specified as `weightTfIdf` by passing an appropriate list to the `control` argument of the `TermDocumentMatrix`, as shown in listing 26

```
tweet_weighted_tdm <- tm::TermDocumentMatrix(x = tweet_corpus_clean,
 ↪  control = list(weighting = weightTfIdf))
tweet_weighted_dtm <- as.DocumentTermMatrix()  %>%
                          as.matrix()
```

Listing 26: Create a Document Term Matrix by transforming a Term Document Matrix

Remove Empty Documents

Empty Documents may be removed from the matrix by a logical test as shown in listing 27 [3] this provides that 328 documents were empty following the processing. A summary of the first rows and columns of

[3] It is important not to filter based the logic of an empty vector, because otherwise an empty vector will returned, hence the if statement in listing 27.

this matrix, following the removal of empty documents, is provided in table 6 of the appendix.

```
null = which(rowSums(as.matrix(tweet_weighted_dtm)) == 0)
rowSums(as.matrix(tweet_weighted_dtm)==0)

if(length(null)!=0){
  tweet_weighted_dtm = tweet_weighted_dtm[-null,]
}

length(null)

## [1] 328
```

Listing 27: Load the Packages for ***R***

8.2.14 How many Clusters are there

Use Cosine Distance

In order to consider clustering, it can be more effective to consider the distance between the weigted documents in terms of cosine distance, the cosine distance can be calculated from the euclidean distance using the identity shown in (11), and this can be performed in ***R*** by taking the matrix product of a diagonalised matrix as shown in listing 28.

$$\begin{aligned}
\text{dist}\left(\mathbf{X},\mathbf{Y}\right) &= ||\mathbf{X}-\mathbf{Y}|| && (3)\\
&= \sqrt{\sum_{i=1}^{n}\left[\left(x_i-y_i\right)^2\right]} && (4)\\
\text{dist}\left(\mathbf{X},\mathbf{Y}\right)^2 &= \sum_{i=1}^{n}\left[\left(x_i-y_i\right)^2\right] && (5)\\
&= \sum_{i=1}^{n}\left(x^2\right)+\sum_{i=1}^{n}\left(y_i^2\right)+2\sum_{i=1}^{n}\left(x_iy_i\right) && (6)\\
&= 1+1+2\times\frac{\sum_{i=1}^{n}\left(x_iy_i\right)}{(1)} && (7)\\
&= 2+2\times\frac{\sum_{i=1}^{n}\left(x_iy_i\right)}{||\mathbf{X}||\times||\mathbf{Y}||} && (8)\\
&= 2+2\cos\left(\mathbf{X},\mathbf{Y}\right) && (9)\\
& && (10)\\
\implies \left(1-\cos\left(\mathbf{X},\mathbf{Y}\right)\right) &= \frac{\text{dist}\left(\mathbf{X},\mathbf{Y}\right)}{2} && (11)
\end{aligned}$$

```r
norm.tweet_weighted_dtm = diag(
                      1/sqrt(rowSums(tweet_weighted_dtm^2))
                      ) %*% tweet_weighted_dtm
D =dist(norm.tweet_weighted_dtm, method = "euclidean")^2/2
```

Listing 28: Load the Packages for ***R***

Project into Euclidean Space

The cosine distance however is not appropriate to perform clustering on and so instead should be projected back into euclidean space, this can be acheived using *Multi-Dimensional Scaling* via the `cmdscale` functionas shown in listing 29. The distance is a measure of between document distance so the number of dimensions should correspond to the number of documents, however, if there are zero-value eigenvalues, these dimensions won't help explain the data in the projection, hence the number of eigenvalues has been used as the dimension of projection in this case.

```r
l <- min(nrow(tweet_weighted_dtm),
         ncol(tweet_weighted_dtm))
ev <- eigen(tweet_weighted_dtm[1:l, 1:l])
k <- (ev$values != 0) %>% sum()

mds.tweet_weighted_dtm <- cmdscale(D, k=k) #TODO What should K be? see
 ↪  issue #10
```

Listing 29: Load the Packages for ***R***

Measure Within Cluster Variance

In order to determine the appropriate number of clusters, the within cluster variance can be measurd, the number of clusters at which this value ceases to decrease is indicative of a potentially appropriate number of clusters. This is implemented in listing 30 and shown in figure 2.

Figure 2 Indicates a sudden stop of decrease in variance at 7 clusters and following that the within cluster variance begins to decrease at a slightly slower rate. For this reason 7 could be an appropriate candidate for the number of clusters, however the minimal amount of change in the within-cluster variance indicates that the data is most likely not clustered at all.

8.2.15 Find the Number of tweets in each cluster

Moving forward we'll use 3 clusters, 7 is too large and a smaller number will likely be more effective at categorising the data (particularly given that the stratification of the data appears to be quite limited from figure 2). The number of tweets in a cluster may be measured by using the `table` function as shown in listing 31 and table 2.

```r
set.seed(271)
n = 15 # Assume it bends at 7 clusters
SSW = rep(0, n)
for (a in 1:n) {
  K = kmeans(mds.tweet_weighted_dtm, a, nstart = 20)
  SSW[a] = K$tot.withinss
  paste(a*100/n, "%") %>% print()
}
SSW


SSW_tb <- tibble::enframe(SSW)
ggplot(SSW_tb, aes(x = name, y = value)) +
  geom_point(col = "#Cd5b45", size = 5) +
  geom_line(col = "#Da70d6") +
  geom_vline(xintercept = 7, lty  = 3, col = "blue") +
  theme_bw() +
  labs(x = "Number of Clusters",
       y = "Within Cluster Sum of Square Distance",
       title = "Within Cluster Variance across Clusters")
```

Listing 30: Use a loop to evaluate the performace of various cluster models, plot this with *ggplot2*

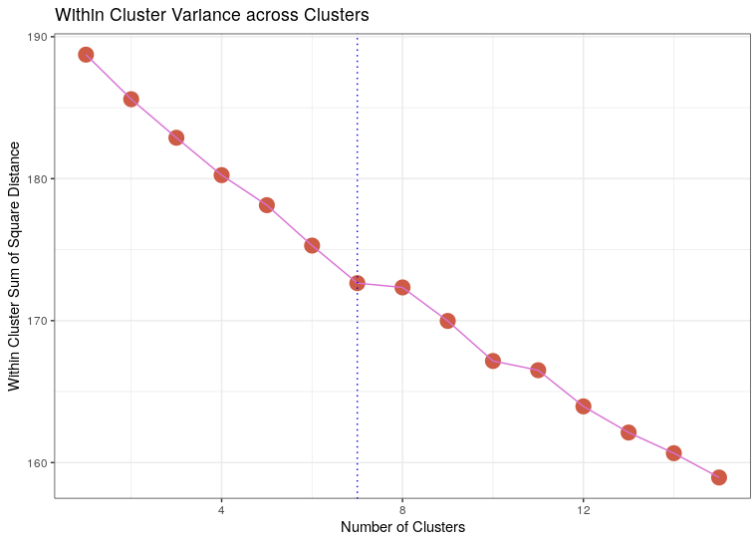

Figure 2: Plot of the Within Cluster Variance of the tweets using cosine distance (projected into Euclidean Space)

Table 2: Number of tweets in each cluster identified by k means clustering

Cluster	1	2	3
# of *tweets*	77	58	537

```r
K = kmeans(mds.tweet_weighted_dtm, 3, nstart = 20)
table(K$cluster)
```

Listing 31: The `table` function can count the number of tweets per cluster.

8.2.16 Visualise the Clusters

The clusters can be projected into 2D *Euclidean*-Space using *Multi-Dimensional Scaling*, these dimensions would represent the first two principle components of the cosine distance of the weigted tweets. This is demonstrated in listings 32. A plot of the friend countstaus mapped to shape and clusters mapped to colour is demonstrated in listing 34 shown in 4, In order to get a better understanding of the distribution of friend counts a plot of the friend count status mapped to colour is demonstrated in listing 33 and shown in figure 4.

```r
MDS_Euclid_2D <- cmdscale(D, k=2) #TODO What should K be? see issue #10
mds_data$Cluster       <- factor(mds_data$Cluster)

if (nrow(MDS_Euclid_2D[,1:2]) == length(K$cluster)
   && length(K$cluster) == nrow(tweets[-null,])) {
mds_data <- cbind(MDS_Euclid_2D[,1:2], "Cluster" = K$cluster,
 ↪  tweets[-null,])
}

mds_data$Cluster       <- factor(mds_data$Cluster)
names(mds_data)[1:2] <- c("MDS1", "MDS2")
```

Listing 32: Use *Multi-Dimensional* scaling to project the data into 2 dimensions

```r
ggplot(pca_data, aes(x = MDS1, y = MDS2, col = Friend_Status)) +
  geom_point(aes(shape = Cluster), size = 2) +
  stat_ellipse(level = 0.95) +
  theme_classic() +
  labs(main = "Principal Components of Twitter Data",
       x = TeX("MDS_1"), y = TeX("MDS_2")) +
  scale_color_discrete(label = c("High Friends", "Low Friends")) +
  guides(col = guide_legend("Friend Count \n Status"))
```

Listing 33: Create a plot of the distribution of Friends among Clusters as shown in figure 3

```r
ggplot(pca_data, aes(x = MDS1, y = MDS2, col = Cluster)) +
  geom_point(aes(shape = Friend_Status), size = 2) +
  stat_ellipse(level = 0.9) +
  theme_classic() +
  labs(main = "Principal Components of Twitter Data",
       x = TeX("MDS_1"), y = TeX("MDS_2")) +
  scale_shape_discrete(label = c("High Friends", "Low Friends")) +
  guides(shape = guide_legend("Friend Count \n Status"))
```

Listing 34: Create a plot of the distribution of Friends Count Status as shown in figure 4

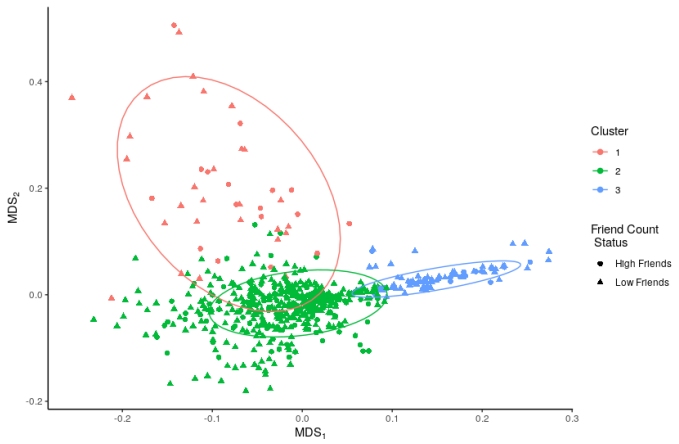

Figure 3: Distribution of Friend Count Status over Clusters

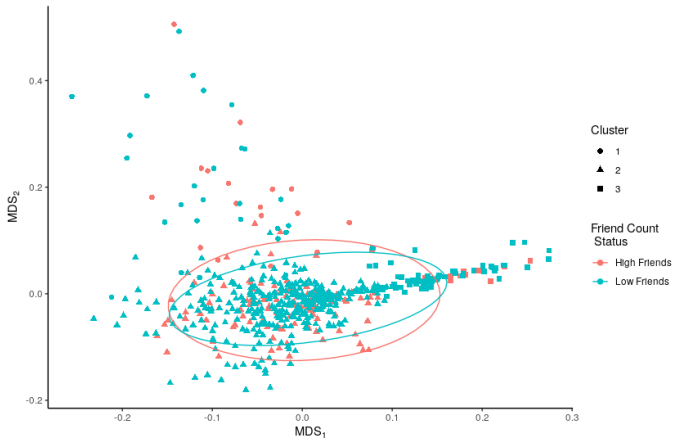

Figure 4: Distribution of Friend Count Status

8.2.17 Comment on the Visualisation

Figure 3 indicates that altough there isn't a clear distinction between clusters there is seperation of the tweets in such a way that does allow some degree of classification to occur. Figure 3 and 4 both indicate that having many or few friends is reasonably independent of the cluster that the tweet belongs to.

8.2.18 Cluster with Highest Number Friends

The Number of above or below friends corresponsing to a given cluster can be tabulated by using the `table` function as shown in listing 35 and table 3, this indicates that there is a difference in the proportion of users with above average friend counts between clusters with cluster #1 having the highest proportion of users with above average friend counts.

```
(clust_friend <- table(
  pca_data[,names(pca_data) %in% c("Cluster", "Friend_Status")])
)

cbind(clust_friend,
        "Proportion" = signif(
            (clust_friend[,1] / rowSums(clust_friend)),
            2)
      )
```

Listing 35: Tabulate the distribution of friends in

Table 3: Above Average Friend Counts Across Clusters

Cluster	*Above Average*	*Below Average*	*Proportion*
1	17	27	0.39
2	141	407	0.26
3	16	64	0.20

8.2.18 Sample Tweets from cluster

Tweets can be sampled from the clusters by using the `sample` function with a logical test, this can be combined with a `for` loop as shown in listing 36, this provides the following output.

- **Cluster 1**
  - @BelovedOfBayek @BayekOfSiwa @assassinscreed @Ubisoft @UbisoftMTL @Captured$_{\text{Collec}}$ @GamerGram$_{\text{GG}}$ @$_{\text{GameScreenshot}}$ And those smiles - and love - are contagious. Believe me, it makes me very happy! What you have is special! ~ Steffi

- @Ubisoft why is it wen I headshot sumone why dont it register a headshot is a 1 shot kill fix your game sir
- @BikiniBodhi I think we need another movement going. Obviously Ubisoft already has plans for the next few releases of ops but we really need an Op whose ability is to reinforce more walls than others. Especially another castle on the team but for walls. Like 4 walls instead of 2.
- The United Arab Emirates logged into my Ubisoft account. For why?
- @Atalagummy Está gratis este finde en lo de ubisoft creo

- **Cluster 2**
  - Here's everything you need to know about Ubisoft's Watch Dogs Legion in Hindi - Release date, Story - Everything we know about it till now. https://t.co/B4lMshJdqw via @YouTube
  - @Rainbow6Game @TheGodlyNoob I have never seen a game company ruin their reputation so fast and so careless as Ubisoft
  - tiltei com a ubisoft, dei block na minha conta sem querer, to mt puto, real
  - My first game that really hyped me was a game called rolling thunder back in the 80s https://t.co/mPWim2hwVY
  - @videogamemorals @PartisanClown Two more remakes ofăLunar: The Silver Star,ăLunar LegendăandăLunar: Silver Star Harmony, were released in 2002 byăMedia RingsăandăUbisoftăand in 2009 byăGungHo Online EntertainmentăandăXseed Games, respectively.\
- **Cluster 3**
  - @Ubisoft @UbisoftSupport crossplay between xbox and pc for Division 2 please?
  - @tornado$_{raphi}$ @Ubisoft @UbisoftDE Haha wollte auch einmal schlau sein :(
  - @Ubisoft I was in the middle in a game waiting for us to spawn in and it took forever and it somehow kicked me from the game for inactivity. Anyway to fix this https://t.co/Uz1yZ73R4M
  - @Operatedleech87 Yo lo veia al revez algo de Ubisoft en Girls Frontline, pero igual un juego en consola jalaria.
  - @VGPNetwork @GamerGram$_{GG}$ @Ubisoft Haha.. I love that film!

```
set.seed(8923)
  for(i in 1:3) {
    n <- sample(which(pca_data$Cluster == i), size = 5)
    print(tweets$text[n])
    print("============================")
    print("============================")
  }
```

Listing 36: Sample Tweets from the Individual Clusters

8.2.19 Identify themes in the Tweets

Highest Friends Count

The themes in the cluster with the highest friend count are:

- Discussion about upcoming titles
  - For example the *Sunborn-Ubisoft* collaboration
    - This is where the use of `collab` comes from in figure 5
  - The upcoming *Division 2* release
- Multiplayer games.

Lowest Friends Count

The themes in the cluster with the lowest friends count are:

- Discussion about older titles
  - e.g. `mincemeat` refers to *Farcry*, mozzi is a *Rainbow 6* character
  - the recurring use of `remember` and `earliest`
- Console games and crossplay between them
- Feature Requests

8.2.20 Create WordClouds

Creating a wordcloud from the stemmed words will make for a poor visualisation while creating a wordcloud based merely on the frequency of words will poorly identify words central to the theme of the cluster.

In light of this, the most appropriate method to generate wordclouds for the purpose visualisising the themes of the clusters is to create a document term matrix based on a corpus cleaned without word stemming as shown in listing 37. Following that wordclouds can then be generated by indexing for the cluster, this is demonstrated in listing 38 and shown in figures 5 and 6.

8.2.21 Use a Dendrogram to Display the Themes

A Dendrogram can be used to display the themes of the higest and lowest clusters. In order to do this the most frequent words of a cluster need to be filtered out in order to reduce the dimensions of the dendrogram, following that a dendrogram may be made to model the cosine distance between the terms. To acheive this in **R** the identity from equation (11) can be used and then a dendrogram modelled over the term document matrix.

```r
clean_corp_ns <- function(corpus) {
  ## Remove URL's
  corpus <- tm_map(corpus,content_transformer(
    function(x) gsub("(f|ht)tp(s?)://\\S+","",x)))
  ## Remove Usernames
  corpus <- tm_map(corpus,content_transformer(function(x)
  ↪  gsub("@\\w+","",x)))
  ## Misc
  corpus <- tm_map(corpus, FUN = removeNumbers)
  corpus <- tm_map(corpus, FUN = removePunctuation)
  corpus <- tm_map(corpus, FUN = stripWhitespace)
  corpus <- tm_map(corpus, FUN = tolower)
  corpus <- tm_map(corpus, FUN = removeWords, mystop)
  ## stopwords() returns characters and is fead as second argument
  return(corpus)
}

tweet_corpus_clean_ns <- clean_corp(tweet_corpus)

tweet_raw_dtm <- tm::TermDocumentMatrix(x = tweet_corpus_ns,
   control = list(weighting = weightTfIdf)) %>%
  as.DocumentTermMatrix()  %>%
  as.matrix()

null = which(rowSums(as.matrix(tweet_matrix_dtm)) == 0)
length(null)

(rowSums(as.matrix(tweet_matrix_dtm))) %>% table()
if(length(null)!=0){
  tweet_matrix_dtm = tweet_matrix_dtm[-null,]
}
```

Listing 37: Apply *TF-IDF* weighting to an unstemmed corpus and then use a `for` loop to create wordclouds corresponding to each cluster.

```r
i <- 1

for (i in c(1,3)) {
  n <- which(pca_data$Cluster == i)

  (relevant <- sort(apply(tweet_raw_dtm[n,], 2, mean),
    decreasing = TRUE)[1:30]) %>% head()

  p <- brewer.pal(n = 5, name = "Set2")
    wordcloud(
    words = names(relevant),
    freq = relevant,
    colors = p,
    random.color = FALSE
  )

}
```

Listing 38: Apply *TF-IDF* weighting to an unstemmed corpus and then use a `for` loop to create wordclouds corresponding to each cluster.

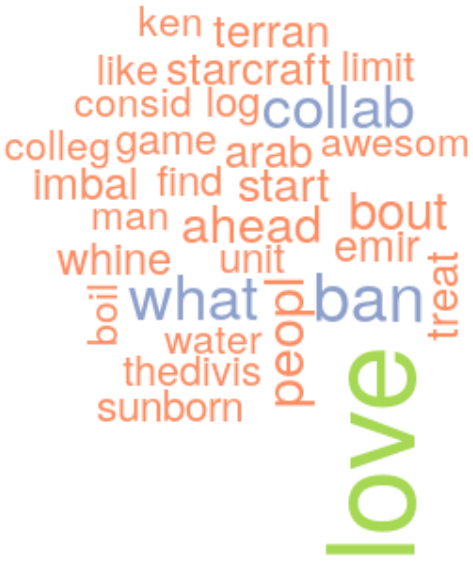

Figure 5: Wordcloud of Cluster # 1 using *TF-IDF* weighting

Figure 6: Wordcloud of Cluster # 3 using *TF-IDF* weighting

Highest Friend Count

Any word that occurs more than once was a candidate for the dendrogram visualising the themes of the cluster with the highest number of friends. This dendrogram is shown in figure 7 and the corresponding code to produce it is provided in listing 39.

Lowest Friend Count

Any word that occurs more than once was a candidate for the dendrogram visualising the themes of the cluster with the lowest number of friends. This dendrogram is shown in figure 8 and the corresponding code to produce it is provided in listing 40.

8.2.23 Conclusion

It appears that users that discuss newer titles and upcoming collaborations belong to a cluster with a higher proportion of users with an above average friend count, whereas users that discuss older titles (such as *Assasins Creed* as demonstrated in figure 8) tend to belong to a cluster with a lower proportion of users with high friend counts.

This could imply that users discussing new titles are more likely to have more friends or it could imply that twitter users with a high friend count are more likely to show interest in newer titles and material.

8.3 Building Networks

```r
## Filter the Data To match the Cluster
tweet_weighted_dtm_c1 <- tweet_weighted_dtm[pca_data$Cluster==1, ]

## Choose terms of a given frequency to reduce the dimensions
frequent.words = which(colSums(tweet_weighted_dtm_c1 > 0) > 1)
term.matrix = tweet_weighted_dtm_c1[,frequent.words]

## In order to use the Cosine Distance Make each vector have a
## magnitude of 1
unit_term.matrix = term.matrix %*% diag(1/sqrt(colSums(term.matrix^2)))

## Preserve the column Names
colnames(unit_term.matrix) = colnames(term.matrix)
colnames(unit_term.matrix)

## Find the Cosine Distance between the Terms
## (Distance between terms so transpose)
t(unit_term.matrix)
D = dist(t(unit_term.matrix), method = "euclidean")^2/2

## Perform Heirarchical Clustering
h = hclust(D, method="average")
plot(h, main = "Themes of Cluster with Highest Friend Count")
```

Listing 39: Create a dendrogram of the terms in the cluster with the higest friends count, average linkage was used.

Figure 7: Dendrogram of Terms in the cluster with the highest friend count using average Linkage.

```r
## Filter the Data To match the Cluster
tweet_weighted_dtm_c1 <- tweet_weighted_dtm[pca_data$Cluster==3, ]

## Choose terms of a given frequency to reduce the dimensions
frequent.words = which(colSums(tweet_weighted_dtm_c1 > 0) > 3)
term.matrix = tweet_weighted_dtm_c1[,frequent.words]

## In order to use the Cosine Distance Make each vector have a
## magnitude of 1
unit_term.matrix = term.matrix %*% diag(1/sqrt(colSums(term.matrix^2)))

## Preserve the column Names
colnames(unit_term.matrix) = colnames(term.matrix)
colnames(unit_term.matrix)

## Find the Cosine Distance between the Terms
## (Distance between terms so transpose)
t(unit_term.matrix)
D = dist(t(unit_term.matrix), method = "euclidean")^2/2

## Perform Heirarchical Clustering
h = hclust(D, method="complete")
plot(h, main = "Themes of Cluster with Lowest Friend Count")
```

Listing 40: Create a dendrogram of the terms in the cluster with the higest friends count, average linkage was used.

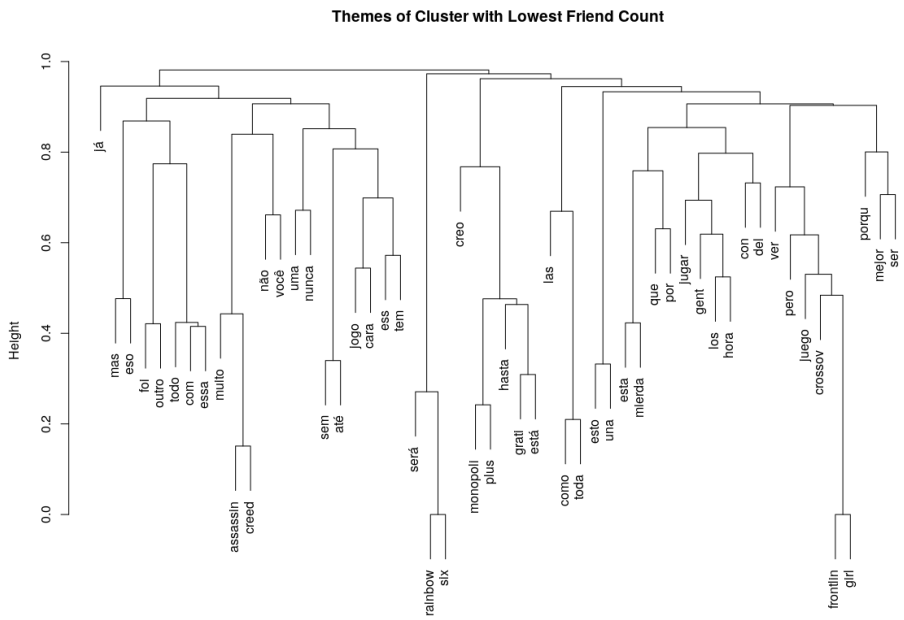

Figure 8: Dendrogram of Terms in the cluster with the highest friend count using average Linkage.

8.3.24 Find the 10 Most Popular Friends of the Twitter Handle

The `get_friends` function can be used to discover the friends of the @ubisoft account, these user *ID* values can then be passed as an argument to the `lookup_users` function, this will return the features of those given users and these results can be sorted by friend count in order to show the most popular friends, as shown in listing 41, this provides the following list of friends in order of the number of there popularity:

1. `BarackObama`	2. `XboxSupport`	3. `DEADLINE`	4. `gameinformer`	5. `Arbys`
6. `PeterHollens`	7. `HarleyPlays`	8. `G4C`	9. `Xbox`	10. `GailSimone`

8.3.25 Obtain a 2-degree egocentric graph centred at *Ubisoft*

8.3.26 Compute the closeness centrality score for each user

8.3.28

Appendix

```r
## * 8.2.24 Find 10 Most Popular Friends of the Twitter Handle
    ↪ ---------------------
## ## ** Get the User ID of Friends of Ubisoft
    ↪ ====================================
t <- get_friends("ubisoft", token = tk)
## *** Get More Information of Friends
    ↪ ##########################################
friends = lookup_users(t$user_id, token = tk)
## **** Inspect the friends
    ↪ ..................................................
dim(friends)
names(friends)

friends$screen_name[1] #name of friend at index 1
friends$followers_count[1] #examine the follower count of the first
    ↪ friend
friends$screen_name[2]

## ** Find the 10 Most Popular Friends
    ↪ ========================================
friendPosition = order(friends$friends_count, decreasing = TRUE)[1:10]
topFriends = friends[friendPosition,] #ids of top 10 friends
## *** Print the top 10 most popular friends
    ↪ ####################################
topFriends$screen_name

## [1] "BarackObama"  "XboxSupport"  "DEADLINE"     "gameinformer"
    ↪ "Arbys"
## [6] "PeterHollens" "HarleyPlays"  "G4C"          "Xbox"
    ↪ "GailSimone"
```

Listing 41: Use `rtweet` to obtain the friends of *Ubisoft* with the most friends

Users with High Friend Count

Table 4: User ID and Friend Count of users with above highest friend count in sample

User ID	*Friend Count*
274488119	8752
743771665	5002
1036014247	4999
2281452613	4992
1554453560	4958
981233818408570880	4944
931765564388921344	4836
807405140	4710
1112579152970842112	4514
2441577446	4322
552692862	4229
956297007127252992	3976
22493896	3675
255922782	3500
1067409881332936709	3312
27998570	3210
715118521555017728	3099
2356170174	2885
2372688230	2880
1868357425	2719

Users with Low Friend Count

TF-IDF Matrix

Relevant XKCD

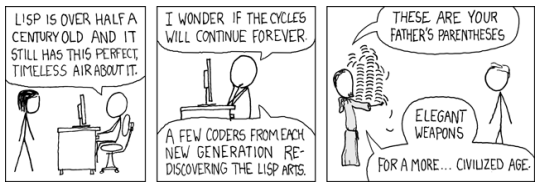

Figure 9: *xkcd* # 297

Table 5: User ID and Friend Count of users with above highest friend count in sample

User ID	*Friend Count*
1254280995592966145	0
875126772978913280	0
1254256124217319425	0
1250219450210480128	0
1214921087328411648	0
1254115699628421120	0
1217600080376520704	0
1253480062453600257	0
1254178435502571521	0
1251955545092718592	0
1106864828700712960	0
1160744587620524032	0
1254256536710504448	1
1129040408384868352	1
1254121201871589376	1
1248687797755658243	2
1210265263867932675	2
3380784928	3
1177274165239275520	3
54645521	3

Table 6: Document Term Matrix of first 6 documents and first 6 Words

check	**coronavirus**	**death**	**first**	**gone**	**hour**
0.615	0.747	0.830	0.596	0.747	0.698
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

References

[1] James Carpenter and John Bithell. "Bootstrap Confidence Intervals: When, Which, What? A Practical Guide for Medical Statisticians". en. In: *Statistics in Medicine* 19.9 (2000), pp. 1141–1164. ISSN: 1097-0258. DOI: 10.1002/(SICI)1097-0258(20000515)19:9<1141::AID-SIM479>3.0.CO;2-F. URL: https://doi-org.ezproxy.uws.edu.au/10.1002/(SICI)1097-0258(20000515)19:9%3C1141::AID-SIM479%3E3.0.CO;2-F (visited on 04/27/2020) (cit. on p. 6).

[2] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and Their Application*. Cambridge ; New York, NY, USA: Cambridge University Press, 1997. ISBN: 978-0-521-57391-7 978-0-521-57471-6 (cit. on p. 6).

[3] Tim C. Hesterberg. "What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum". In: *The American Statistician* 69.4 (Oct. 2015), pp. 371–386. ISSN: 0003-1305. DOI: 10.1080/00031305.2015.1089789. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4784504/ (visited on 04/26/2020) (cit. on p. 6).

[4] Gareth James et al., eds. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics 103. OCLC: ocn828488009. New York: Springer, 2013. ISBN: 978-1-4614-7137-0 (cit. on p. 6).

[5] Michael Kearney. *Get Tweets Data on Statuses Identified via Search Query. Search_tweets*. en. Manual. 2019. URL: https://rtweet.info/reference/search_tweets.html (visited on 04/26/2020) (cit. on p. 1).

[6] Travis LeCompte and Jianhua Chen. "Sentiment Analysis of Tweets Including Emoji Data". In: *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. Dec. 2017, pp. 793–798. DOI: 10.1109/CSCI.2017.137 (cit. on p. 15).

[7] NIST. *1.3.3.14.6. Histogram Interpretation: Skewed (Non-Normal) Right*. Oct. 2013. URL: https://www.itl.nist.gov/div898/handbook/eda/section3/histogr6.htm (visited on 04/26/2020) (cit. on p. 2).

[8] Brian Ripley. *Boot.Ci Function | R Documentation*. Apr. 2020. URL: https://www.rdocumentation.org/packages/boot/versions/1.3-25/topics/boot.ci (visited on 04/27/2020) (cit. on p. 6).

[9] Mohammed O. Shiha and Serkan Ayvaz. "The Effects of Emoji in Sentiment Analysis". In: *International Journal of Computer and Electrical Engineering* 9.1 (2017), pp. 360–369. ISSN: 17938163. DOI: 10.17706/IJCEE.2017.9.1.360-369. URL: http://www.ijcee.org/vol9/943-T048.pdf (visited on 04/29/2020) (cit. on p. 15).

[10] Matt Wiley and Joshua Wiley. *Advanced R Statistical Programming and Data Models*. New York, NY: Springer Berlin Heidelberg, 2019. ISBN: 978-1-4842-2871-5 (cit. on p. 6).