TABLE OF CONTENTS

1 ANALYSIS OF TWITTER LANGUAGE ABOUT RYAN REYNOLDS 6

1.1 Searching For Tweets 6

1.1.1 Code Employed 6

1.2 Generating A Document Term Matrix 6

1.2.1 Code Employed 6

1.3 Generating A Word Cloud 7

1.3.1 Code Employed 7

1.3.2 Output Recieved 7

1.4 Findings About Language Used In Tweets Associated With Ryan Reynolds 7

2 CLUSTERING THE TWEETS 9

2.1 Finding The Number Of Clusters- Elbow Method 9

2.1.1 Code Employed 9

2.1.2 Output Recieved 9

2.2 Cluster Tweets Using K-Means Clustering 9

2.2.1 Code Employed 9

2.2.2 Findings 9

2.3 Comparing Sizes Of Clusters 10

2.3.1 Code Employed 10

2.3.2 Output Received 10

2.3.3 Findings 10

2.4 Visualisation- 2d Vector Space 10

2.4.1 Code Employed 10

2.4.2 Output Received 10

2.4 Dendrograms- Top 2 Clusters 10

2.4.1 Code Employed- Cluster 1 10

2.4.2 Output Recieved 11

2.4.3 Code Employed- Cluster 2 11

2.4.4 Output Received 11
2.4.5 Findings 11
2.5 Findings 11
3 WHO TO FOLLOW 12
3.1 Top 100 Tweets Retweeted The Most. 12
3.1.1 Code Employed 12
3.2 Users Of Top 100 Tweets Retweeted 13
3.2.1 Code Employed 13
3.3 Followers Count And Statuses Count 13
3.3.1 Code Employed 13
3.4 Relationship Between Follower And User 14
3.4.1 Code Employed 14
3.4.3 Output Received 19
3.4.4 Code Employed Continued Part 1 19
3.4.5 Output Recieved 20
3.4.6 Code Employed Continued Part 2 20
3.4.7 Output Recieved 20
3.5 Findings 20
4 BUILDING NETWORKS 21
4.1 Ten Most Popular Friends 21
4.1.1 Code Employed 21
4.1.2 Output Received 22
4.2 1.5-Degree Egocentic Graph 22
4.1.1 Code Employed 22
4.1.2 Output Received 23
4.3 Popularity Of Pages- Top Three 23
4.1.1 Code Employed 23
4.1.2 Output Received 24

4.1.3 Findings 24
4.2 Findings 25
REFERENCES 26
APPENDIX 27
Appendix 1 27
Appendix 2 28
Appendix 3 29
Appendix 4 30
Appendix 5 31
Appendix 6 32
Appendix 7 33

1 ANALYSIS OF TWITTER LANGUAGE ABOUT RYAN REYNOLDS

1.1 SEARCHING FOR TWEETS

To obtain 2000 tweets Ryan Reynolds, the search_tweets function of the rtweet library is used. First, twitter_token is created, containing the needed keys and tokens for Twitter's application programming interface (API). twitter_token is then used with the search_tweets function to search for 2000 recent tweets (excluding retweets), containing both the words "Ryan" and "Reynolds". These 2000 tweets are saved as tweets.

1.1.1 CODE EMPLOYED

```
library("rtweet")

app = "put app string here"
key = "put key string here"
secret = "put secret string here"
access_token = "put access token string here"
access_secret = "put access token secret string here"

twitter_token = create_token(app, key, secret, access_token, access_secret,
set_renv=FALSE)

tweets = search_tweets('Ryan Reynolds', n = 2000, type = "recent", token = twitter_token,
include_rts= FALSE)
```

1.2 GENERATING A DOCUMENT TERM MATRIX

To create the document-term matrix, a corpus is first created by extracting the text from the bodies of the previously generated tweets. This corpus is then used to create the document-term matrix, named dtm, with punctuation, numbers, and common English stop words removed, and all words converted to lowercase. The created document-term matrix is then weighted using the weightTfIdf function, with the results being saved as dtmw.

1.2.1 CODE EMPLOYED

```
library("tm")

# Create a corpus.
corpus = Corpus(VectorSource(tweets$text))

# Create document term matrix.
dtm = DocumentTermMatrix(corpus, control = list(removePunctuation = TRUE, stopwords
= TRUE, removeNumbers = TRUE, tolower = TRUE))
```

```
# Compute the weighted document term matrix.
dtmw = weightTfIdf(dtm)
tweets.matrix = as.matrix(dtmw)
```

1.3 GENERATING A WORD CLOUD

1.3.1 CODE EMPLOYED

```
library("wordcloud")
# Sum the weights.
tweets.matrix = as.matrix(dtmw)
freqsw = colSums(tweets.matrix)
freqsw = freqs[!is.na(freqsw)]
wordcloud(names(freqsw), freqsw, random.order = FALSE, min.freq = 12)
```

The authors of this report chose to have a minimum frequency of 12 rather than a smaller value, as it was found that smaller values diminished the readability of the word cloud generated from the document term matrix.

1.3.2 OUTPUT RECIEVED

See **Appendix 1** for output obtained after the code in *section 1.3.1* was entered.

1.4 FINDINGS ABOUT LANGUAGE USED IN TWEETS ASSOCIATED WITH RYAN REYNOLDS

The largest words present in the word cloud and therefore the ones with the most weight are “movie”, “free”, “guy”, “like”, “trailer”, “deadpool”, “lantern”, “trailer”, “green”, “loveque”, and “vancityreynolds”. Given Ryans recent body of work, it is understandable as to why many of these words would be present among recent discussions involving Ryan Reynolds. For example, on October 6th 2020 Ryan Reynolds sent out a tweet premiering the most recent trailer for his new upcoming movie “Free Guy” (Reynolds, 2020) thus it is understandable that “movie”, “trailer”, “free”, and “guy” would be dominating words in discussions regarding Reynolds.

Additionally, since the Disney studio acquisition of Fox studios, there has been speculation as to how the Character of Deadpool would be added into the Marvel Cinematic universe. Furthermore, it was reported on October 6th 2020 that Ryan Reynolds was signing a deal with Marvel Studios, which is reportedly one of the biggest deals in the history of the Marvel Cinematic Universe (Tomar, 2020). While it is unclear as to what the deal entails, it is understandable that Marvel character Deadpool which Ryan portrays, would once again be a hot topic of conversation while the public speculates not only the particulars of this deal and how it will impact future Marvel Cinematic Universe movies. It is likely that other words such as “highest-paid” and “marvel” are also present in the word cloud for this same reason, although they are not of as high a weight indicating that they are not used as frequently as the others during these discussions. However, given the lack of context the word Deadpool could therefore be discussing the movies, the character, the comics, the Disney merger, the Marvel contract, or it could be in relation to the countless number of advertising campaigns that they have both used Deadpool in or has been for Deadpool. With so many possibilities as to why the word Deadpool would be part of discussions involving Ryan Reynolds it is no wonder it is among the top weighted words in the word cloud.

It was surprising to note that there was no mention of the words “Aviation” and/or “Gin”. Aviation American Gin is a gin company that is partly owned by Ryan Reynolds and uses Reynolds heavily in the advertisement of their product. Reynolds discusses the product frequently and it was also used as part of the long running ‘feud’ he has with fellow actor Hugh Jackman, whose name is another noticeable absence in the word cloud. Since their collaboration in the movie *X-Men Origins: Wolverine*, both Ryan Reynolds and Hugh Jackman have had a playful, funny, and very public ‘feud’. While it is obvious that the two are very much friends, they often play pranks and make jokes at the other's expense (Estera, 2020). Given the frequency of these occurrences and how much attention they draw from the public, it is very surprising that “Hugh” or “Jackman” is not present at all in the word cloud. However, given that the last advertisement to feature Reynolds on Twitter for Aviation Gin was listed on September 25th (Reynolds, Twitter Post, 2020), it could have fallen out of the public eye and explain why it is not a current topic of conversation involving Ryan Reynolds.

2 CLUSTERING THE TWEETS

2.1 FINDING THE NUMBER OF CLUSTERS- ELBOW METHOD

2.1.1 CODE EMPLOYED

```
norm.tweets.matrix = diag(1/sqrt(rowSums(tweets.matrix^2))) %*% tweets.matrix
D = dist(norm.tweets.matrix, method = "euclidean")^2/2
mds.tweets.matrix <- cmdscale(D, k=100)

n = 15
SSW = rep(0, n)
for (a in 1:n) {
  set.seed(40) #Ensures same data when executed
  K = kmeans(mds.tweets.matrix, a, nstart = 20)
  SSW[a] = K$tot.withinss
}
plot(1:15, SSW, type = "b")
```

2.1.2 OUTPUT RECIEVED

See **Appendix 2** for output obtained.

2.2 CLUSTER TWEETS USING K-MEANS CLUSTERING

2.2.1 CODE EMPLOYED

```
#cluster the tweets using K-means clustering
K = kmeans(mds.tweets.matrix, 4, nstart = 20)
```

2.2.2 FINDINGS

Based on the decision to use four clusters in the previous section, these four clusters are then computed using the kmeans function, which performs k-means clustering, and the results stored in K.

2.3 COMPARING SIZES OF CLUSTERS

2.3.1 CODE EMPLOYED

```
# Viewing 4 clusters:
table(K$cluster)
```

2.3.2 OUTPUT RECEIVED

```
  1    2    3    4
  20   32  190 1758
```

2.3.3 FINDINGS

Using the table function, the number of tweets of each of the previously computed four clusters is listed. From this output it can be seen that four is the largest cluster.

2.4 VISUALISATION- 2D VECTOR SPACE

2.4.1 CODE EMPLOYED

```
# Visualise the clusters.
mds2.tweets.matrix <- cmdscale(D, k = 2)
plot(mds2.tweets.matrix, col = K$cluster)
```

2.4.2 OUTPUT RECEIVED

See **Appendix 3** for output received

2.4 DENDROGRAMS- TOP 2 CLUSTERS

2.4.1 CODE EMPLOYED- CLUSTER 1

```
# Find position of tweets in cluster 3.
cluster3TweetsId = which(K$cluster == 3)
# Extract tweets vectors for cluster 3.
cluster3Tweets = tweets.matrix[cluster3TweetsId,]
# Find only the terms that appear in at least 5 tweets.
cluster3.frequent.words = which(colSums(cluster3Tweets > 0) > 5)
cluster3.term.matrix = cluster3Tweets[,cluster3.frequent.words]
# Make sure that all term vectors (columns of matrix) have a norm of 1.
cluster3.norm.term.matrix = cluster3.term.matrix %*%
diag(1/sqrt(colSums(cluster3.term.matrix^2)))
# Preserve column names.
colnames(cluster3.norm.term.matrix) = colnames(cluster3.term.matrix)
# Compute the Euclidean distance for cluster3.norm.term.matrix.
cluster3.D = dist(t(cluster3.norm.term.matrix), method = "euclidean")^2/2
```

```
# Hierarchical clustering.
cluster3.h = hclust(cluster3.D, method="complete")
plot(cluster3.h, main = "Cluster 3 Dendrogram")
```

2.4.2 OUTPUT RECIEVED

See **Appendix 4** for output generated from the code above

2.4.3 CODE EMPLOYED- CLUSTER 2

```
# Find position of tweets in cluster 4.
cluster4TweetsId = which(K$cluster == 4)
# Extract tweets vectors for cluster 4.
cluster4Tweets = tweets.matrix[cluster4TweetsId,]
# Find only the terms that appear in at least 30 tweets.
cluster4.frequent.words = which(colSums(cluster4Tweets > 0) > 30)
cluster4.term.matrix = cluster4Tweets[,cluster4.frequent.words]
# Make sure that all term vectors (columns of matrix) have a norm of 1.
cluster4.norm.term.matrix = cluster4.term.matrix %*%
diag(1/sqrt(colSums(cluster4.term.matrix^2)))
# Preserve column names.
colnames(cluster4.norm.term.matrix) = colnames(cluster4.term.matrix)
# Compute the Euclidean distance for cluster4.norm.term.matrix.
cluster4.D = dist(t(cluster4.norm.term.matrix), method = "euclidean")^2/2
# Hierarchical clustering.
cluster4.h = hclust(cluster4.D, method="complete")
plot(cluster4.h, main = "Cluster 4 Dendrogram")
```

2.4.4 OUTPUT RECEIVED

See **Appendix 5** for output generated from the code above

2.4.5 FINDINGS

Based on the output of *section 2.3* which found clusters three and four to be the largest. Both of these clusters have been analysed individually to create dendrograms of their words.

2.5 FINDINGS

Based on the output of *section 2.1*, four clusters has been determined to be the most suitable number to use for subsequent sections. When examining the results of the SSW output (in *section 2.1*), there is an elbow at four clusters, and then a further drop afterwards, suggesting a possible cluster of clusters. While there is also an elbow at eight clusters, the elbow at four clusters seems more prominent, thus four was found to be the most appropriate number to use.

Within these four cluster outputs, we can see a low start from cluster one to a noticeable jump in size between two and three, and then a major leap in the last cluster which is also the largest of the four.

Based on the results of *section 2.3*, clusters three and four are the largest. Both clusters have been analysed individually to create dendrograms of their words. While cluster three has a limit of words with at least five tweets, cluster four has a limit of 30 tweets since it is a significantly larger cluster.

The words appearing in the dendrograms are very similar to those in the word cloud. Based on the dendrogram *(see* **Appendix 4***),* the noticeable difference pertains to the inclusion of foreign words, with a few being in Spanish and a sizable cluster in French. Cluster three's dendrogram showcases new topics of conversation, while cluster four's dendrogram (*see* **Appendix 5***)* borrows the same words from the word cloud.

The group of French words "nouvelles", "video", and "jeu", in cluster 3's dendrogram, translate to the English words "news", "video", and "game", respectively. It is a logical connection for the French language to be linked to Ryan Reynolds because of his Canadian background, and French being one of the two official languages of Canada. However, the inclusion of these French words could also demonstrate Reynolds' presence within French media culture, possibly suggesting that he transcends media formats and is recognised between news, film and game enthusiasts alike. The word "jeu" or "game", is highly likely to be associated with Reynolds' new upcoming film *Free Guy* where the protagonist realises he's been living in a video game world. This has been highly marketed and is even mentioned in the clusters.

3 WHO TO FOLLOW

3.1 TOP 100 TWEETS RETWEETED THE MOST.

3.1.1 CODE EMPLOYED

```
# Sort tweets by retweet_count.
top100.retweets = tweets[order(-tweets$retweet_count),]

# Isolate the top 100.
```

```
top100.retweets = head(top100.retweets, n = 100)
```

3.2 USERS OF TOP 100 TWEETS RETWEETED

3.2.1 CODE EMPLOYED

```
library("rtweet")

# Get info on the retweeters.
top100.retweeters.info = users_data(top100.retweets)

# Remove duplicates.
top100.retweeters.info = unique(top100.retweeters.info)
```

3.3 FOLLOWERS COUNT AND STATUSES COUNT

3.3.1 CODE EMPLOYED

```
# Extract status and follower count.
top100.retweeters.counts = data.frame(top100.retweeters.info$user_id,
top100.retweeters.info$statuses_count, top100.retweeters.info$followers_count)

# Fix column names.
names(top100.retweeters.counts) <- c("user_id","statuses_count","followers_count")
```

3.4 RELATIONSHIP BETWEEN FOLLOWER AND USER

3.4.1 CODE EMPLOYED

```
# Initialise an empty matrix to store the counts.
followers.statuses = matrix(0, nrow = 11, ncol = 6)
# Row names for follower counts.
rownames(followers.statuses) <- c("0-99999", "100000-199999", "200000-299999", "300000-399999", "400000-499999", "500000-599999", "600000-
699999", "700000-799999", "800000-899999", "900000-999999", "1000000+")
# Column names for status counts.
colnames(followers.statuses) <- c("0-9999", "10000-19999", "20000-29999", "30000-39999", "40000-49999","50000+")

# Check each row in top100.retweeters.counts and categorise it into followers.statuses.
for (row in 1:nrow(top100.retweeters.counts)) {
        # Check follower count, then status count, then increment the appropriate cell in followers.statuses.
        if (top100.retweeters.counts[row, "followers_count"] >= 0 & top100.retweeters.counts[row, "followers_count"] <= 99999) {
                if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
                        followers.statuses["0-99999", "0-9999"] = followers.statuses["0-99999", "0-9999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
                        followers.statuses["0-99999", "10000-19999"] = followers.statuses["0-99999", "10000-19999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
                        followers.statuses["0-99999", "20000-29999"] = followers.statuses["0-99999", "20000-29999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
                        followers.statuses["0-99999", "30000-39999"] = followers.statuses["0-99999", "30000-39999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
                        followers.statuses["0-99999", "40000-49999"] = followers.statuses["0-99999", "40000-49999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
                        followers.statuses["0-99999", "50000+"] = followers.statuses["0-99999", "50000+"] + 1
                }
        } else if (top100.retweeters.counts[row, "followers_count"] >= 100000 & top100.retweeters.counts[row, "followers_count"] <= 199999) {
                if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
                        followers.statuses["100000-199999", "0-9999"] = followers.statuses["100000-199999", "0-9999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
                        followers.statuses["100000-199999", "10000-19999"] = followers.statuses["100000-199999", "10000-19999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
                        followers.statuses["100000-199999", "20000-29999"] = followers.statuses["100000-199999", "20000-29999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
            followers.statuses["100000-199999", "30000-39999"] = followers.statuses["100000-199999", "30000-39999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
            followers.statuses["100000-199999", "40000-49999"] = followers.statuses["100000-199999", "40000-49999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
            followers.statuses["100000-199999", "50000+"] = followers.statuses["100000-199999", "50000+"] + 1
        }
    } else if (top100.retweeters.counts[row, "followers_count"] >= 200000 & top100.retweeters.counts[row, "followers_count"] <= 299999) {
        if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
            followers.statuses["200000-299999", "0-9999"] = followers.statuses["200000-299999", "0-9999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
            followers.statuses["200000-299999", "10000-19999"] = followers.statuses["200000-299999", "10000-19999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
            followers.statuses["200000-299999", "20000-29999"] = followers.statuses["200000-299999", "20000-29999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
            followers.statuses["200000-299999", "30000-39999"] = followers.statuses["200000-299999", "30000-39999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
            followers.statuses["200000-299999", "40000-49999"] = followers.statuses["200000-299999", "40000-49999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
            followers.statuses["200000-299999", "50000+"] = followers.statuses["200000-299999", "50000+"] + 1
        }
    } else if (top100.retweeters.counts[row, "followers_count"] >= 300000 & top100.retweeters.counts[row, "followers_count"] <= 399999) {
        if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
            followers.statuses["300000-399999", "0-9999"] = followers.statuses["300000-399999", "0-9999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
            followers.statuses["300000-399999", "10000-19999"] = followers.statuses["300000-399999", "10000-19999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
            followers.statuses["300000-399999", "20000-29999"] = followers.statuses["300000-399999", "20000-29999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
            followers.statuses["300000-399999", "30000-39999"] = followers.statuses["300000-399999", "30000-39999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
            followers.statuses["300000-399999", "40000-49999"] = followers.statuses["300000-399999", "40000-49999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
            followers.statuses["300000-399999", "50000+"] = followers.statuses["300000-399999", "50000+"] + 1
        }
    } else if (top100.retweeters.counts[row, "followers_count"] >= 400000 & top100.retweeters.counts[row, "followers_count"] <= 499999) {
        if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
            followers.statuses["400000-499999", "0-9999"] = followers.statuses["400000-499999", "0-9999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
            followers.statuses["400000-499999", "10000-19999"] = followers.statuses["400000-499999", "10000-19999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
            followers.statuses["400000-499999", "20000-29999"] = followers.statuses["400000-499999", "20000-29999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
            followers.statuses["400000-499999", "30000-39999"] = followers.statuses["400000-499999", "30000-39999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
            followers.statuses["400000-499999", "40000-49999"] = followers.statuses["400000-499999", "40000-49999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
            followers.statuses["400000-499999", "50000+"] = followers.statuses["400000-499999", "50000+"] + 1
        }
    } else if (top100.retweeters.counts[row, "followers_count"] >= 500000 & top100.retweeters.counts[row, "followers_count"] <= 599999) {
        if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
            followers.statuses["500000-599999", "0-9999"] = followers.statuses["500000-599999", "0-9999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
            followers.statuses["500000-599999", "10000-19999"] = followers.statuses["500000-599999", "10000-19999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
            followers.statuses["500000-599999", "20000-29999"] = followers.statuses["500000-599999", "20000-29999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
            followers.statuses["500000-599999", "30000-39999"] = followers.statuses["500000-599999", "30000-39999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
            followers.statuses["500000-599999", "40000-49999"] = followers.statuses["500000-599999", "40000-49999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
            followers.statuses["500000-599999", "50000+"] = followers.statuses["500000-599999", "50000+"] + 1
        }
    } else if (top100.retweeters.counts[row, "followers_count"] >= 600000 & top100.retweeters.counts[row, "followers_count"] <= 699999) {
        if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
            followers.statuses["600000-699999", "0-9999"] = followers.statuses["600000-699999", "0-9999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
            followers.statuses["600000-699999", "10000-19999"] = followers.statuses["600000-699999", "10000-19999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
            followers.statuses["600000-699999", "20000-29999"] = followers.statuses["600000-699999", "20000-29999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
            followers.statuses["600000-699999", "30000-39999"] = followers.statuses["600000-699999", "30000-39999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
            followers.statuses["600000-699999", "40000-49999"] = followers.statuses["600000-699999", "40000-49999"] + 1
        } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
            followers.statuses["600000-699999", "50000+"] = followers.statuses["600000-699999", "50000+"] + 1
```

```
		}
	} else if (top100.retweeters.counts[row, "followers_count"] >= 700000 & top100.retweeters.counts[row, "followers_count"] <= 799999) {
		if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
			followers.statuses["700000-799999", "0-9999"] = followers.statuses["700000-799999", "0-9999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
			followers.statuses["700000-799999", "10000-19999"] = followers.statuses["700000-799999", "10000-19999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
			followers.statuses["700000-799999", "20000-29999"] = followers.statuses["700000-799999", "20000-29999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
			followers.statuses["700000-799999", "30000-39999"] = followers.statuses["700000-799999", "30000-39999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
			followers.statuses["700000-799999", "40000-49999"] = followers.statuses["700000-799999", "40000-49999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
			followers.statuses["700000-799999", "50000+"] = followers.statuses["700000-799999", "50000+"] + 1
		}
	} else if (top100.retweeters.counts[row, "followers_count"] >= 800000 & top100.retweeters.counts[row, "followers_count"] <= 899999) {
		if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
			followers.statuses["800000-899999", "0-9999"] = followers.statuses["800000-899999", "0-9999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
			followers.statuses["800000-899999", "10000-19999"] = followers.statuses["800000-899999", "10000-19999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
			followers.statuses["800000-899999", "20000-29999"] = followers.statuses["800000-899999", "20000-29999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
			followers.statuses["800000-899999", "30000-39999"] = followers.statuses["800000-899999", "30000-39999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
			followers.statuses["800000-899999", "40000-49999"] = followers.statuses["800000-899999", "40000-49999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
			followers.statuses["800000-899999", "50000+"] = followers.statuses["800000-899999", "50000+"] + 1
		}
	} else if (top100.retweeters.counts[row, "followers_count"] >= 900000 & top100.retweeters.counts[row, "followers_count"] <= 999999) {
		if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
			followers.statuses["900000-999999", "0-9999"] = followers.statuses["900000-999999", "0-9999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
			followers.statuses["900000-999999", "10000-19999"] = followers.statuses["900000-999999", "10000-19999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
			followers.statuses["900000-999999", "20000-29999"] = followers.statuses["900000-999999", "20000-29999"] + 1
		} else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
			followers.statuses["900000-999999", "30000-39999"] = followers.statuses["900000-999999", "30000-39999"] + 1
```

```
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
                        followers.statuses["900000-999999", "40000-49999"] = followers.statuses["900000-999999", "40000-49999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
                        followers.statuses["900000-999999", "50000+"] = followers.statuses["900000-999999", "50000+"] + 1
                }
        } else if (top100.retweeters.counts[row, "followers_count"] >= 1000000) {
                if(top100.retweeters.counts[row, "statuses_count"] >= 0 & top100.retweeters.counts[row, "statuses_count"] <= 9999) {
                        followers.statuses["1000000+", "0-9999"] = followers.statuses["1000000+", "0-9999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 10000 & top100.retweeters.counts[row, "statuses_count"] <= 19999) {
                        followers.statuses["1000000+", "10000-19999"] = followers.statuses["1000000+", "10000-19999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 20000 & top100.retweeters.counts[row, "statuses_count"] <= 29999) {
                        followers.statuses["1000000+", "20000-29999"] = followers.statuses["1000000+", "20000-29999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 30000 & top100.retweeters.counts[row, "statuses_count"] <= 39999) {
                        followers.statuses["1000000+", "30000-39999"] = followers.statuses["1000000+", "30000-39999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 40000 & top100.retweeters.counts[row, "statuses_count"] <= 49999) {
                        followers.statuses["1000000+", "40000-49999"] = followers.statuses["1000000+", "40000-49999"] + 1
                } else if (top100.retweeters.counts[row, "statuses_count"] >= 50000) {
                        followers.statuses["1000000+", "50000+"] = followers.statuses["1000000+", "50000+"] + 1
                }
        }
}

# Create function to stretch the matrix into a data frame table.
stretchTable = function(tab, variableNames) {
        tabx = rep(rownames(tab), rowSums(tab))
        l = ncol(tab)
        m = nrow(tab)
        cn = colnames(tab)
        taby = c()
        for (a in 1:m) {
                for (b in 1:l) {
                        taby = c(taby, rep(cn[b], tab[a,b]))
                }
        }
        d = data.frame(x = tabx, y = taby)
        colnames(d) = variableNames
        return(d)
}
```

```
stretched.followers.statuses = stretchTable(followers.statuses, c("Followers","Statuses"))
# Display the table.
table(stretched.followers.statuses)
```

Users are categorised based on their follower and statuses counts, with this being stored in the matrix followers.statuses. This was then stretched into a data frame table and stored in stretched.followers.statuses. Using the table function, stretched.followers.statuses produces the output in *section 3.4.3*

3.4.3 OUTPUT RECEIVED

```
                 Statuses
Followers          0-9999 10000-19999 20000-29999 30000-39999 40000-49999 50000+
  0-99999              31           9           5           7           1     18
  100000-199999         1           0           0           0           0      2
  1000000+              0           1           1           0           0      5
  200000-299999         0           0           1           0           0      3
  300000-399999         0           0           0           0           0      1
  400000-499999         0           0           0           0           0      1
  500000-599999         0           0           0           0           0      1
  900000-999999         0           0           0           0           0      1
```

3.4.4 CODE EMPLOYED CONTINUED PART 1

```
# Create the functions for calculating chi-squared statistic.
expectedIndependent = function(X) {
        n = sum(X)
        p = rowSums(X)/sum(X)
        q = colSums(X)/sum(X)
        return(p %o% q * n) # outer product creates table
}

chiSquaredStatistic = function(X, E) {
        return(sum((X - E)^2/E))
}

# Compute the expected table.
E = expectedIndependent(table(stretched.followers.statuses)) # compute expected counts
if independent

# Compute the randomisation distribution.
x2dist = replicate(1000, { # compute 1000 randomised chi-squared statistics
        followersShuffle = sample(stretched.followers.statuses$Followers)
        statusesShuffle = sample(stretched.followers.statuses$Statuses)
        Yindep = table(followersShuffle, statusesShuffle)
        chiSquaredStatistic(Yindep, E)
})
hist(x2dist)
```

A function is created to calculate a chi-squared statistic later. This function is named chiSquaredStatistic. Another function is created to compute the table of expected counts. The aforementioned table of expected counts is computed and stored as E. The chi-squared randomisation distribution is then generated and stored in x2dist. The hist function is used to create and display a histogram of x2dist, with the results found in *section 3.4.5*

3.4.5 OUTPUT RECIEVED

See **Appendix 6** for output generated from the code above

3.4.6 CODE EMPLOYED CONTINUED PART 2

```
x2 = chiSquaredStatistic(table(stretched.followers.statuses), E)

# pval is the proportion of x2dist that is greater than x2.
pval = mean(x2dist > x2)
print(pval)
```

3.4.7 OUTPUT RECIEVED

```
[1] 0.74
```

Using the previously created chiSquaredStatistic function, a chi-squared statistic is computed for followers.statuses and stored in x2. The p-value for this test is then computed and stored in pval.

3.5 FINDINGS

To test the relationship between follower and statuses count, a chi-squared (χ^2) test for independence was chosen. The chi-squared test for independence test is used to test if two variables, in this case the follower and statuses count, are independent of each other. While there are numerous other statistical tests to apply, the purpose of the chi-squared test for independence test aligns the most with the goal of this section, which is to find if follower and statuses count have a relationship, i.e., whether they are independent of each other or not.

After isolating the most retweeted 100 tweets from the previously found 2000 tweets, the users of these tweets as well as their follower and statuses count were obtained.

The chi-squared test of independence was used to test the relationship between the two categorical variables: follower count and statuses count. As the chi-squared test for independence test is used, this is essentially a hypothesis test. The two hypotheses are as follows:

- H_0: Follower and statuses count are independent.
- H_A: Follower and statuses count are not independent.

As the p-value (calculated in *section 3.4*) is large, and therefore indicates a significantly larger randomised distribution than chi-squared statistic, H_0 cannot be rejected. Consequently the belief that being active on Twitter will result in an increase in followers cannot be proven as there is not enough evidence to conclude that follower count and statuses count are dependent upon one another.

4 BUILDING NETWORKS

4.1 TEN MOST POPULAR FRIENDS

4.1.1 CODE EMPLOYED

```
library("twitteR")
setup_twitter_oauth(key, secret, access_token, access_secret)

user = getUser("VancityReynolds")
#identify how many friends he has
infor = user$toDataFrame()
#download his friends
friends = user$getFriends(infor$friendsCount)

#examine how many followers do these friends have
follower.count = c()
for (a in c(1:length(friends))){
  follower.count[a] = friends[[a]]$getFollowersCount()
}

#find the top 10 most popular friends - 10 friends that have the most followers
count.followers = function(friends) {
  follower.count = c()
  for (a in c(1:length(friends))) {
    follower.count[a] = friends[[a]]$getFollowersCount()
  }
  return(follower.count)
}
friendFollowCount = count.followers(friends)
friendPosition = order(friendFollowCount, decreasing = TRUE)[1:10]
topFriends = friends[friendPosition]

#get their name
```

```
friend.names = c()
n = length(topFriends)
for (a in 1:n) {
  friend.names[a] = topFriends[[a]]$getScreenName()
}

# List their names
friend.names
```

4.1.2 OUTPUT RECEIVED

```
[1] "BarackObama"
[2] "TheEllenShow"
[3] "jimmyfallon"
[4] "nytimes"
[5] "MileyCyrus"
[6] "NiallOfficial"
[7] "elonmusk"
[8] "KevinHart4real"
[9] "Pink"
[10] "aliciakeys"
```

4.2 1.5-DEGREE EGOCENTIC GRAPH

4.1.1 CODE EMPLOYED

```
library("igraph")

#download 100 friends from each of the 10 most popular friends
more.friends = list()
n = length(topFriends)
for (a in 1:n) {
  more.friends[[a]] = topFriends[[a]]$getFriends(100)
}

#store all 100 screen names in the variable friend.names.
friend.names = c()
n = length(friends)
for (a in 1:n) {
  friend.names[a] = friends[[a]]$getScreenName()
}

#write the function to build the edge list
user.to.edgelist <- function(user, friends) {
  # create the list of friend screen names
  friend.names = c()
  for (a in c(1:length(friends))) {
    friend.names[a] = friends[[a]]$getScreenName()
  }
  user.name = rep(user$getScreenName(), length(friends)) # repeat user's name
  el = cbind(user.name,friend.names) # bind the columns to create a matrix
  return(el)
}
el.ryan = user.to.edgelist(user, friends)
for (a in c(1:length(more.friends))) {
  el.friend = user.to.edgelist(topFriends[[a]], more.friends[[a]])
  el.ryan = rbind(el.ryan, el.friend)  # append the new edge list to the old one.
}

#create the graph
g = graph.edgelist(el.ryan)
plot(g, layout = layout.fruchterman.reingold, vertex.size = 5)
g2 = subgraph(g, which(degree(g, mode = "all") > 4))
plot(g2)
```

4.1.2 OUTPUT RECEIVED

See **Appendix 7** for output generated from the code above

4.3 POPULARITY OF PAGES- TOP THREE

4.1.1 CODE EMPLOYED

```
#obtain the probability transition matrix T
A = t(as.matrix(get.adjacency(g2)))
print(A)
normalise = function(x) {
  if(sum(x)!=0){
    return(x/sum(x))
  } else return(0)
}
adjacency.to.probability = function(A) {
  cols = ncol(A)
  for (a in 1:cols) {
    A[, a] = normalise(A[, a])
  }
  return(A)
}

T = adjacency.to.probability(A)

#create the random jump matrix J
J = matrix(rep(1/11, 11 * 11), 11, 11)

#Combine the T and J to obtain the matrix M
alpha = 0.8
M = alpha * T + (1 - alpha) * J
#Normalise the columns of M to obtain an ergodic transition probability matrix.
M = adjacency.to.probability(M)
difference = function(x,y) {
  return(sqrt(sum((x - y)^2)))
}
```

```
stationary.distribution = function(T) {
  # first create the initial state distribution
  n = ncol(T)
  p = rep(0, n)
  p[1] = 1

  # now take a random walk until the state distribution reaches the
  # stationary distribution.
  p.old = rep(0, n)
  while (difference(p, p.old) > 1e-06) {
    p.old = p
    p = T %*% p.old
  }
  return(p)
}
p = stationary.distribution(M)
print(p)
```

4.1.2 OUTPUT RECEIVED

```
                      [,1]
VancityReynolds 0.03438790
elonmusk        0.03713893
KevinHart4real  0.03713893
nytimes         0.03713893
NiallOfficial   0.03713893
MileyCyrus      0.22466001
aliciakeys      0.24657848
BarackObama     0.03713893
Pink            0.23440112
TheEllenShow    0.03713893
jimmyfallon     0.03713893
```

4.1.3 FINDINGS

According to the PageRank scores, the top three most popular people are Miley Cyrus (@MileyCyrus), Alicia Keys (@aliciakeys), and Pink (@Pink).

4.2 FINDINGS

As of 10th October 2020, Ryan Reynolds was following 993 Twitter accounts. Of these accounts, the ten most popular accounts were obtained (see *section 4.1*) to create an outlook of Ryan's Twitter network. For the purposes of this report, the term "friends" will refer to this condensed list of the top ten most popular Twitter accounts that are being followed by Reynolds.

In *section 4.2*, the data of the aforementioned friends list was used to construct a 1.5 egocentric graph (see **Appendix 7**) . The graph showcases not only the relationship between Reynolds and his friends, but it also indicates any interconnecting relationship(s) that may exist between all of the accounts that are listed. It can be observed that seven out of the ten accounts share a singular non-ergodic connection towards Ryan Reynolds (@VancityReynolds), as they do not connect to any other of the ten accounts.

However, the graph also depicts a visible close relationship between four of Reynolds' friends. According to the graph, the Twitter accounts of Miley Cyrus (@MileyCyrus), Pink (@Pink), Alicia Keys (@aliciakeys), and Kevin Hart (@KevinHart4real) share connections. While these four accounts are not all connected to each other, Alicia Keys, for example, is connected to Pink, however, Pink is not connected to Kevin Hart. This indicates that there is some form of interaction with each other's Twitter accounts. It cannot be determined using solely this graph as to the type or frequency of the interaction between the accounts, since they could simply be following each other, with no further interaction beyond that point. All that can be determined using the 1.5 egocentric graph is that a connection exists. While some of the connections in this section of the graph can be seen as ergodic, the complete graph is still considered non-ergodic.

Using the PageRank method, the probability and stationary distribution is used to construct scores between Reynolds' Twitter account and his friends'. Based on these scores, Ryan Reynolds' three most popular friends are Miley Cyrus (@MileyCyrus), Alicia Keys (@aliciakeys), and Pink (@Pink).

REFERENCES

Estera, C. (2020, October 13). *Ryan Reynolds expertly trolls Hugh Jackman on his birthday*. Retrieved from Honey- Celebrity: https://celebrity.nine.com.au/latest/ryan-reynolds-trolls-hugh-jackman-birthday-video-message-instagram/10ae2602-6f87-4fea-ac5d-5af351f8757e

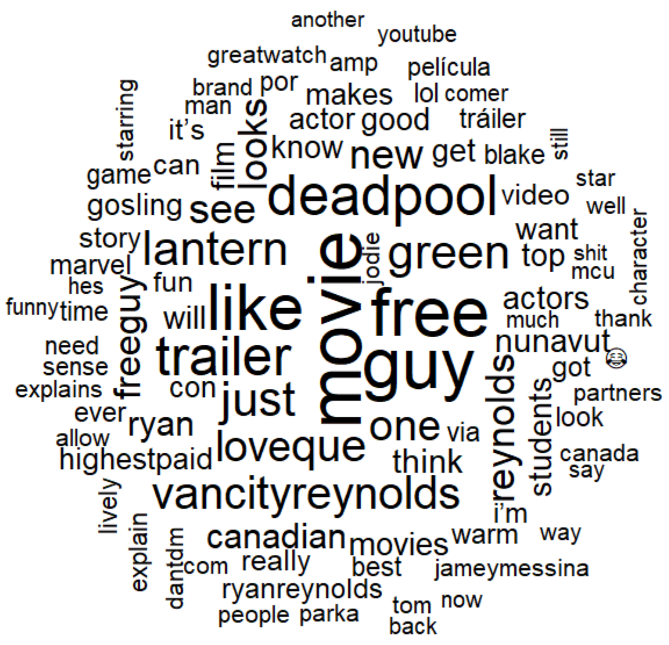
Reynolds, R. (2020, October 6). *Twitter Post*. Retrieved from Twitter: https://twitter.com/VancityReynolds/status/1313102072368291840?s=20

Reynolds, R. (2020, September 25). *Twitter Post*. Retrieved from Twitter: https://twitter.com/VancityReynolds/status/1309199586670579712?s=20

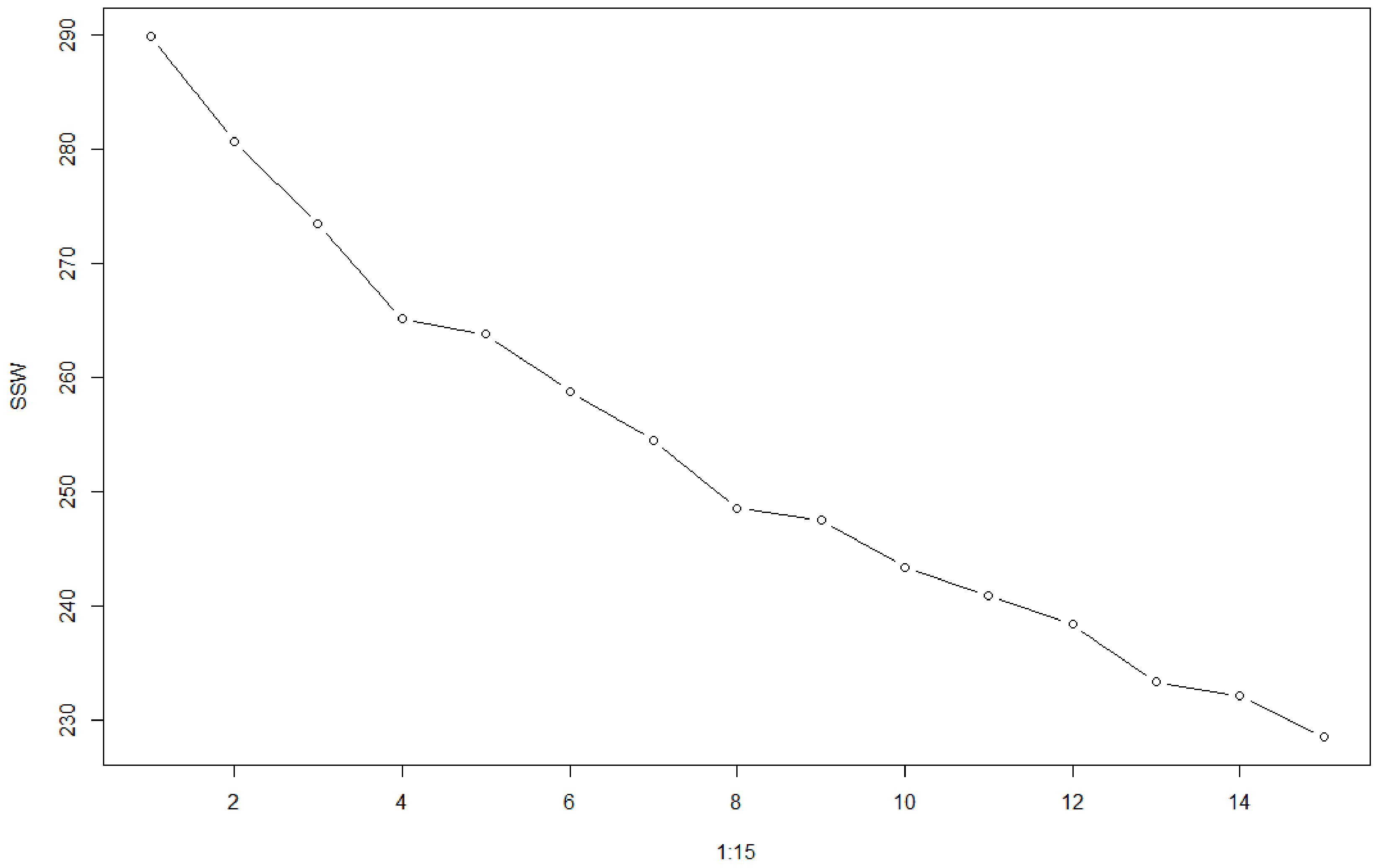
Tomar, M. (2020, October 6). *Deadpool Ryan Reynolds signing astronomical deal in MCU history.* Retrieved from Micky: https://micky.com.au/deadpool-ryan-reynolds-signing-astronomical-deal-in-mcu-history/

APPENDIX

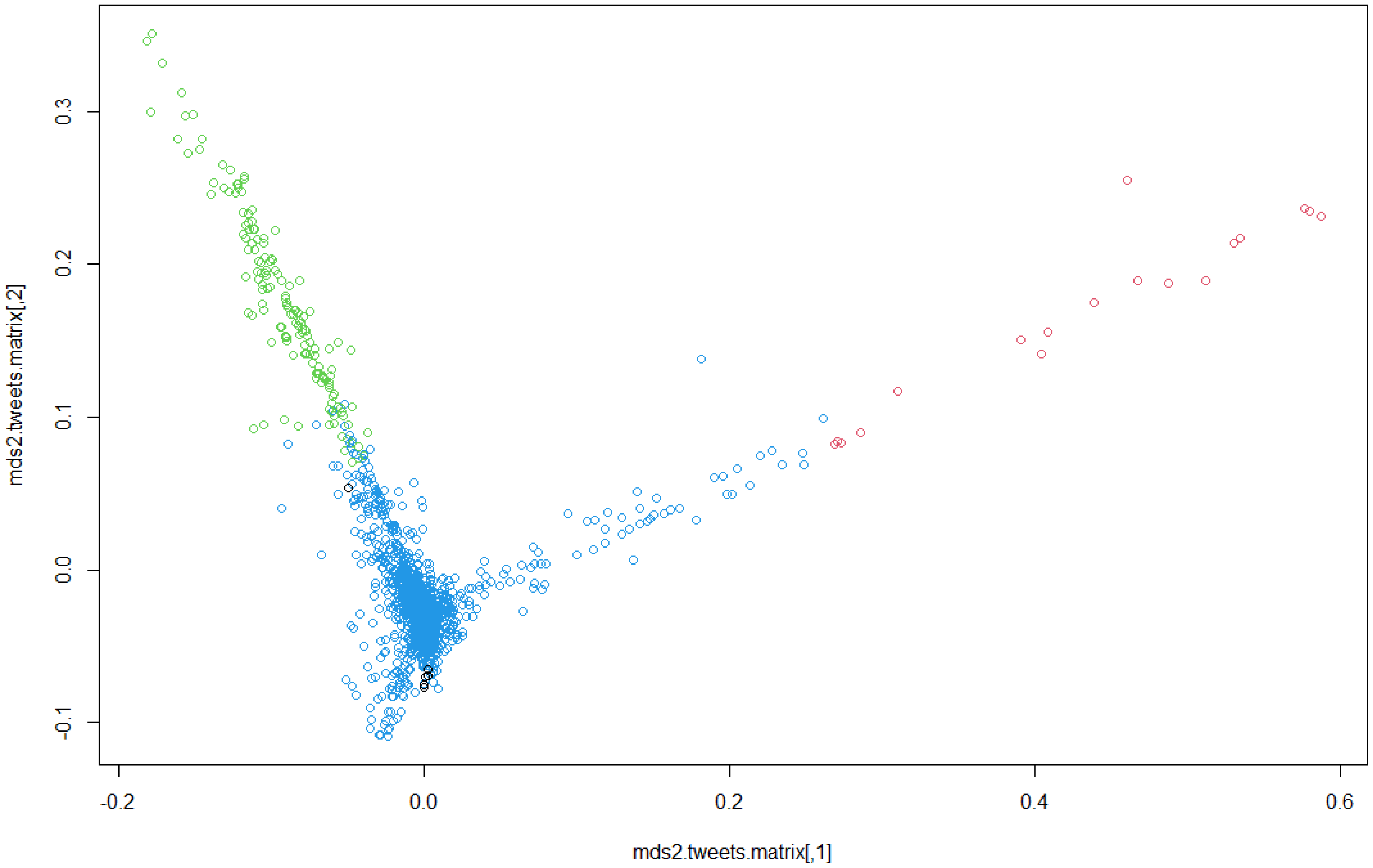
Appendix 1



Appendix 2

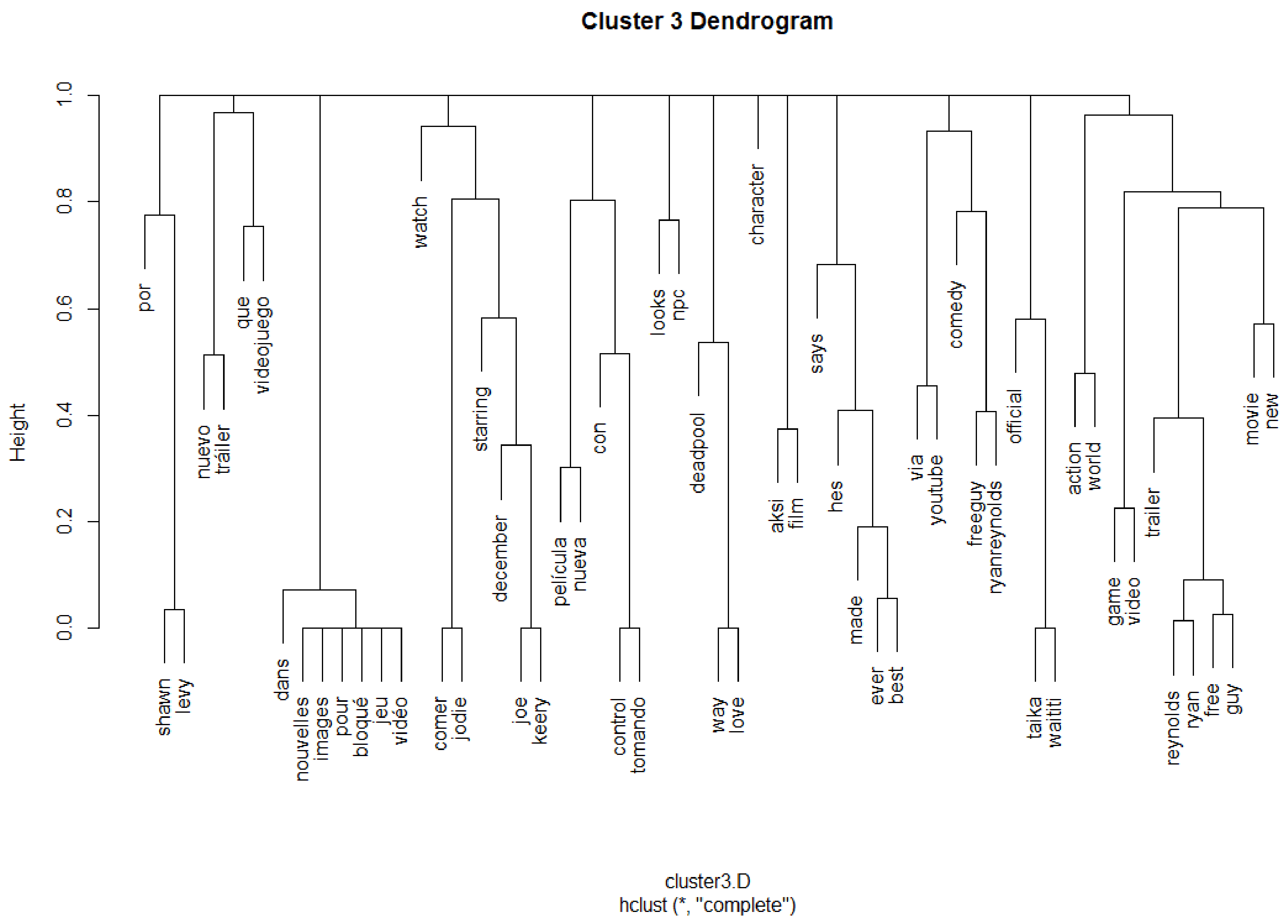


Appendix 3

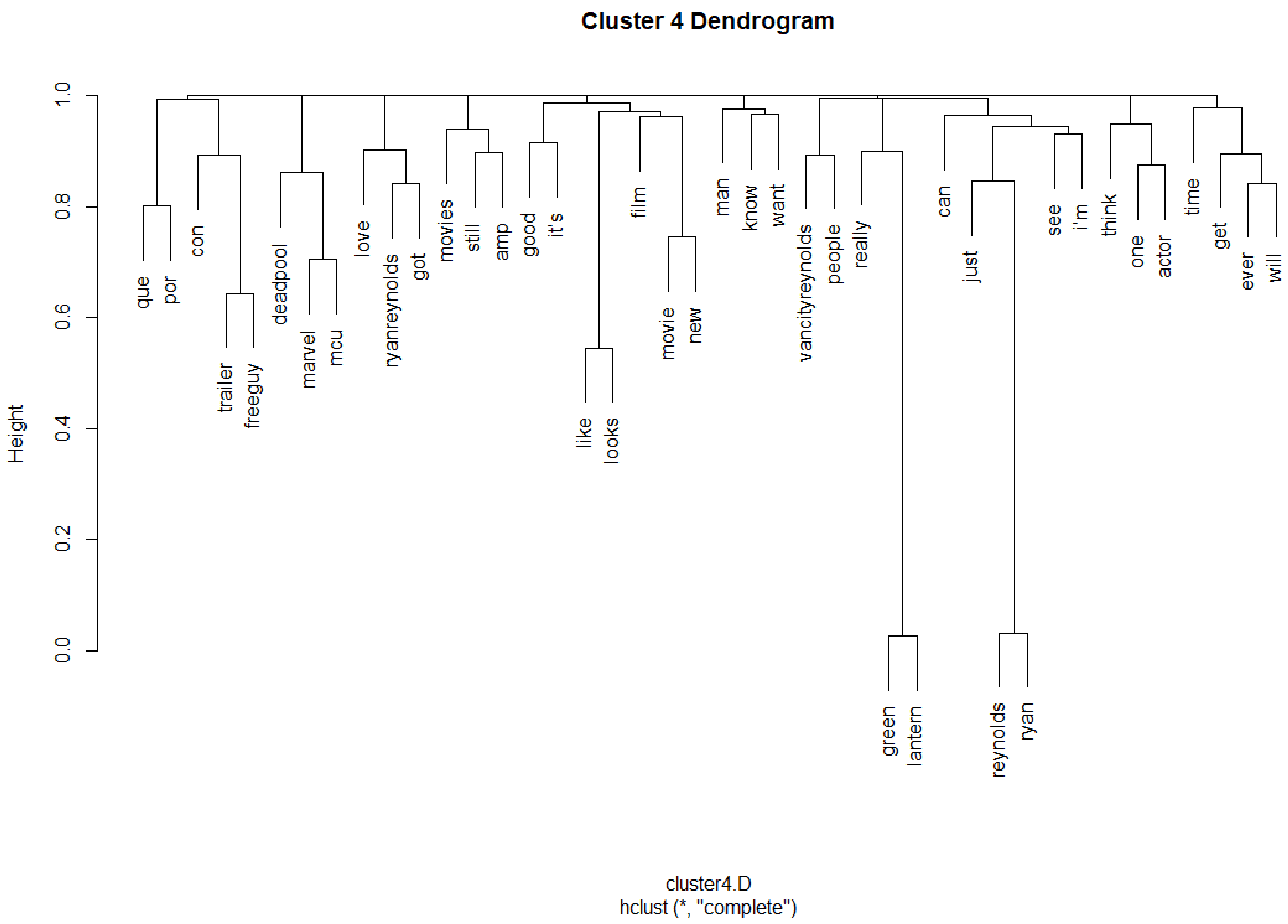


0.3
0.2
0.1
0.0
-0.1
mds2.tweets.matrix[,2]
-0.2
0.0
0.2
0.4
0.6
mds2.tweets.matrix[,1]

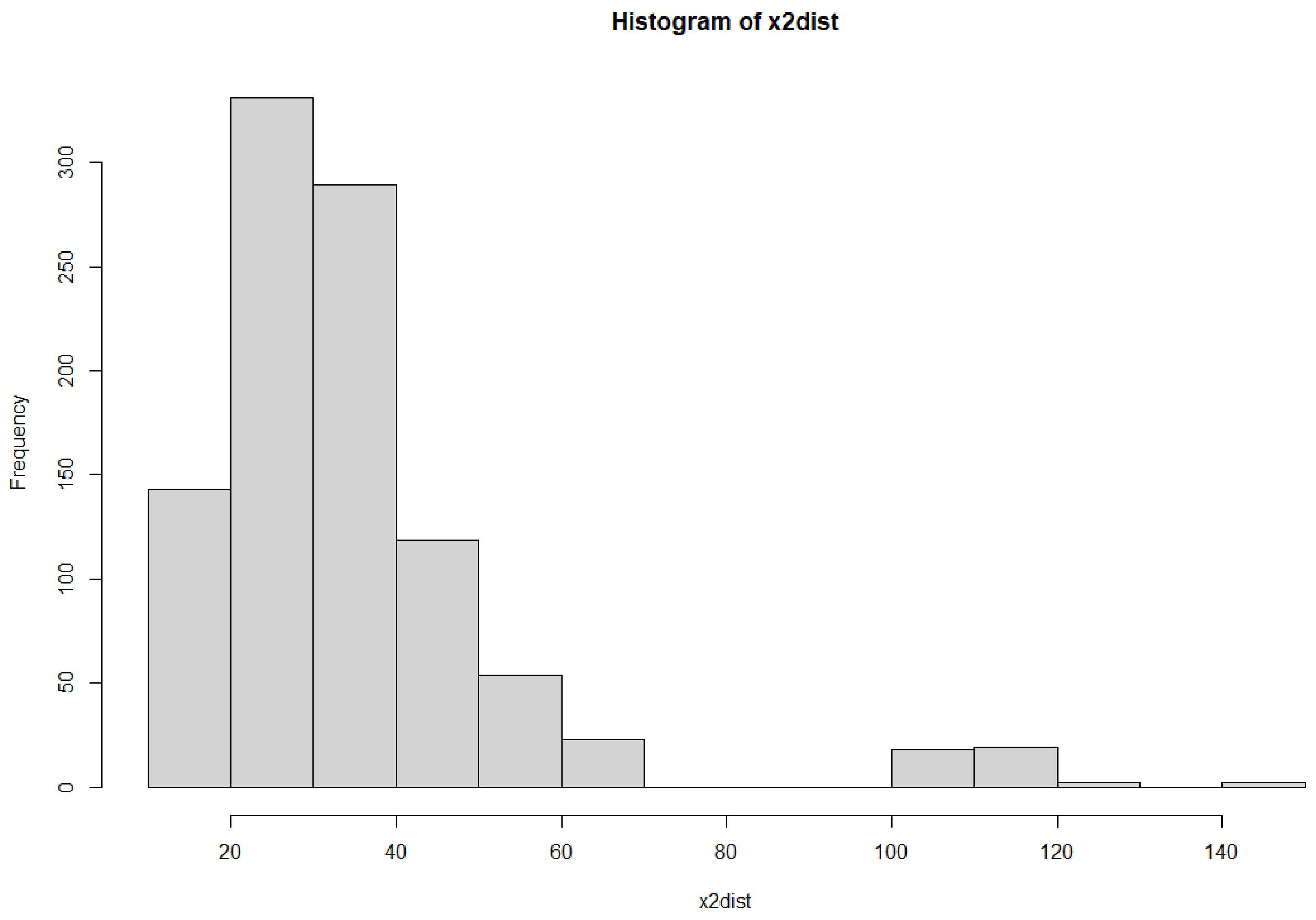
Appendix 4


Cluster 3 Dendrogram
Height
1.0
0.8
0.6
0.4
0.2
0.0
por
shawn
levy
nuevo
tráiler
que
videojuego
dans
nouvelles
images
pour
bloqué
jeu
vidéo
watch
comer
jodie
starring
december
joe
keery
película
nueva
con
control
tomando
looks
npc
deadpool
way
love
character
aksi
film
says
hes
made
ever
best
via
youtube
comedy
freeguy
ryanreynolds
official
taika
waititi
action
world
game
video
trailer
reynolds
ryan
free
guy
movie
new
cluster3.D
hclust (*, "complete")

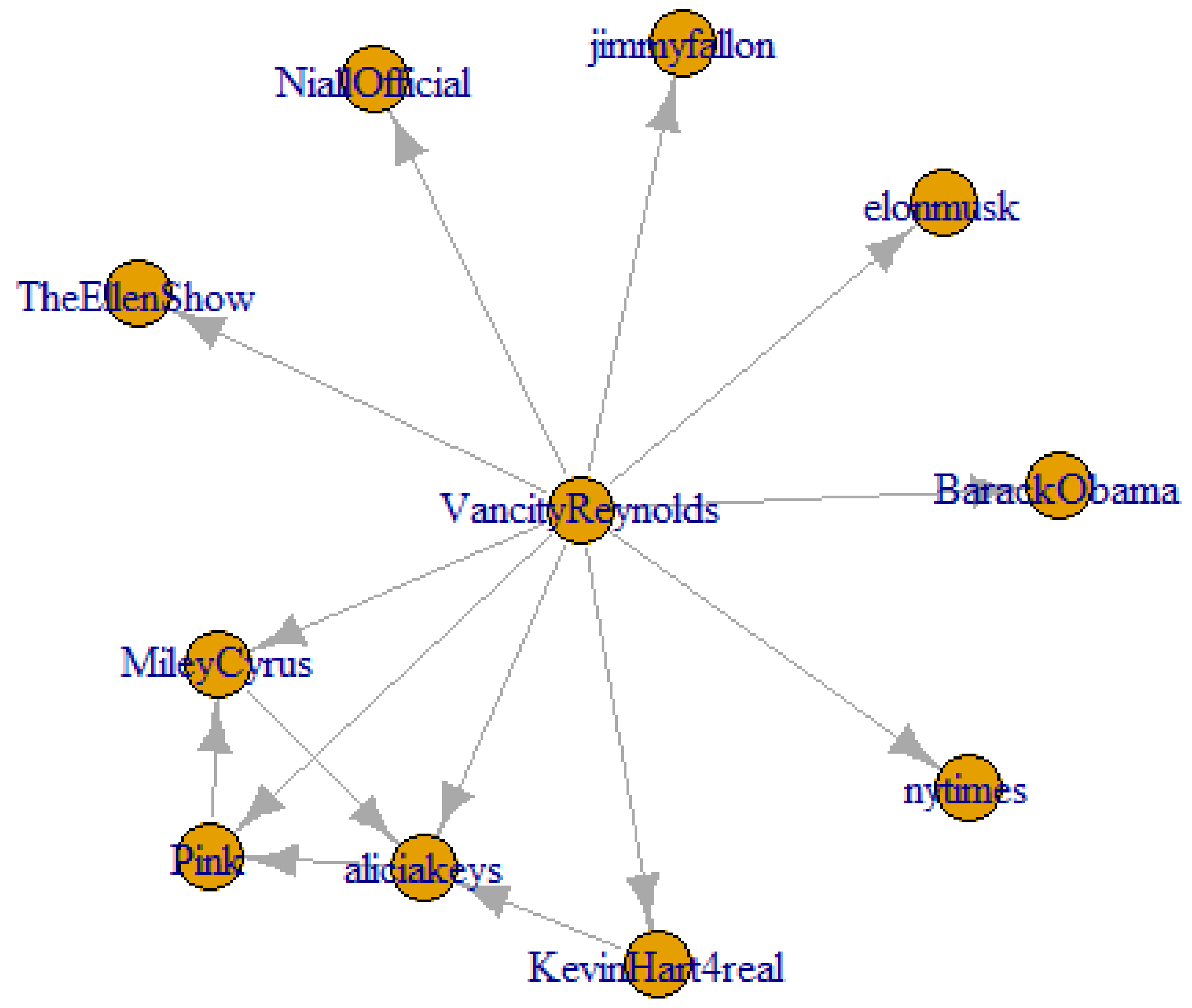
Appendix 5


Cluster 4 Dendrogram
Height
0.0
0.2
0.4
0.6
0.8
1.0
que
por
con
trailer
freeguy
deadpool
marvel
mcu
love
ryanreynolds
got
movies
still
amp
good
it's
like
looks
film
movie
new
man
know
want
vancityreynolds
people
really
green
lantern
can
just
reynolds
ryan
see
i'm
think
one
actor
time
get
ever
will
cluster4.D
hclust (*, "complete")

Appendix 6


Histogram of x2dist
Frequency
0
50
100
150
200
250
300
20
40
60
80
100
120
140
x2dist

Appendix 7


jimmyfallon
NiallOfficial
elonmusk
TheEllenShow
VancityReynolds
BarackObama
MileyCyrus
nytimes
Pink
aliciakeys
KevinHart4real