DATA SOCIETY™

"If you can't explain it simply, you don't understand it well enough."

- Albert Einstein

Which problems will you solve?

Goals for this course:

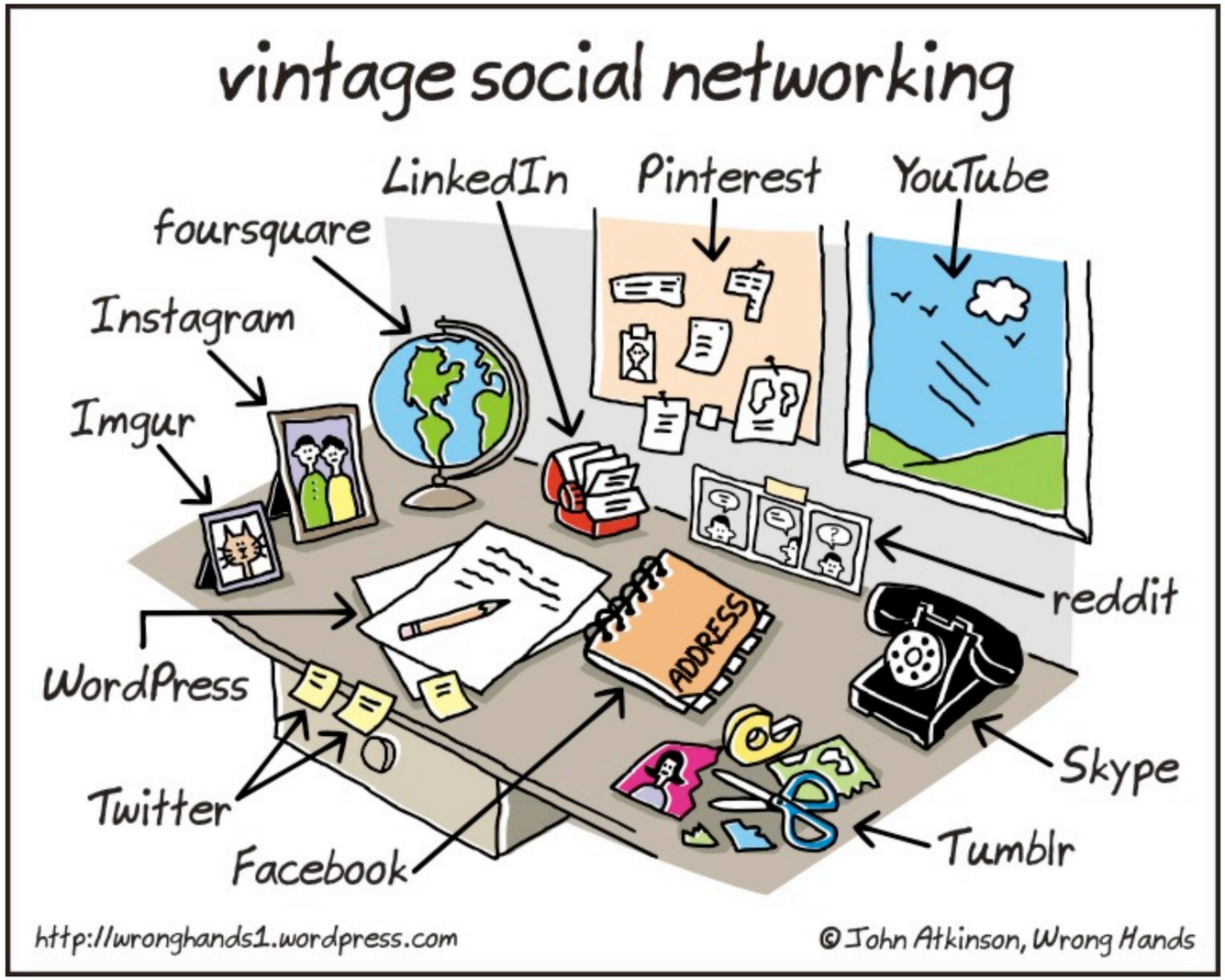
1. Identify key influencers and discover which connectors are most important
2. Create a strategy to spread your message most effectively and test your model in a simulation!
3. Visualize networks with several interactive visualizations that will allow you to drill deeper into network data

Setting expectations

Data science takes dedication! You will need to:

1. Take this course ☺
2. Practice coding
3. Review class material on your own
4. Practice coding
5. Complete concept reviews and exercises outside of class
6. Practice coding
7. Share and read latest news

Social media: then and now

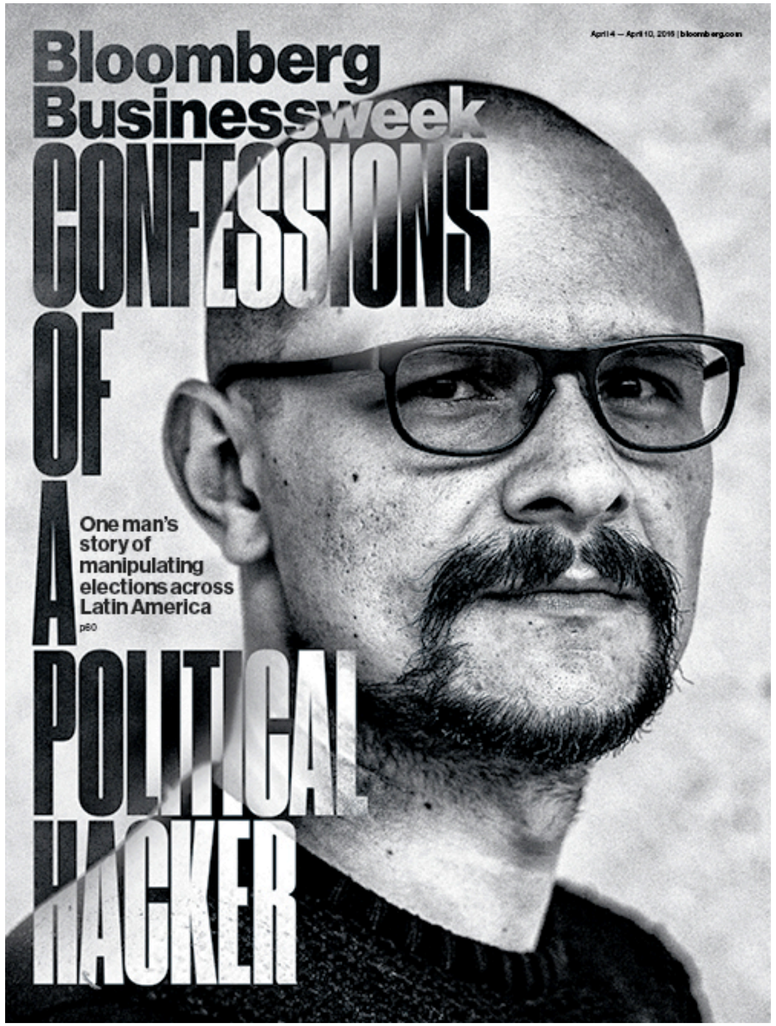


Social media networks

- Relationships between people are continuously recorded on the internet
- Social media is used by criminals and law enforcement – the best users understand what drives interactions and how people can be affected most effectively
- Chain reactions can trigger cascades of human activity from voting to the spread of disease to energy conservation and other conscientious behavior

Social media manipulation

- Andrés Sepúlveda manipulated public perceptions with social media to influence elections
- He managed thousands of fake Twitter profiles and disrupted opponents' campaigns by hacking their sites and emails
- Since being imprisoned, the U.S. government has used a modified version of his software to counter terrorism



Bloomberg Businessweek, April 4, 2016

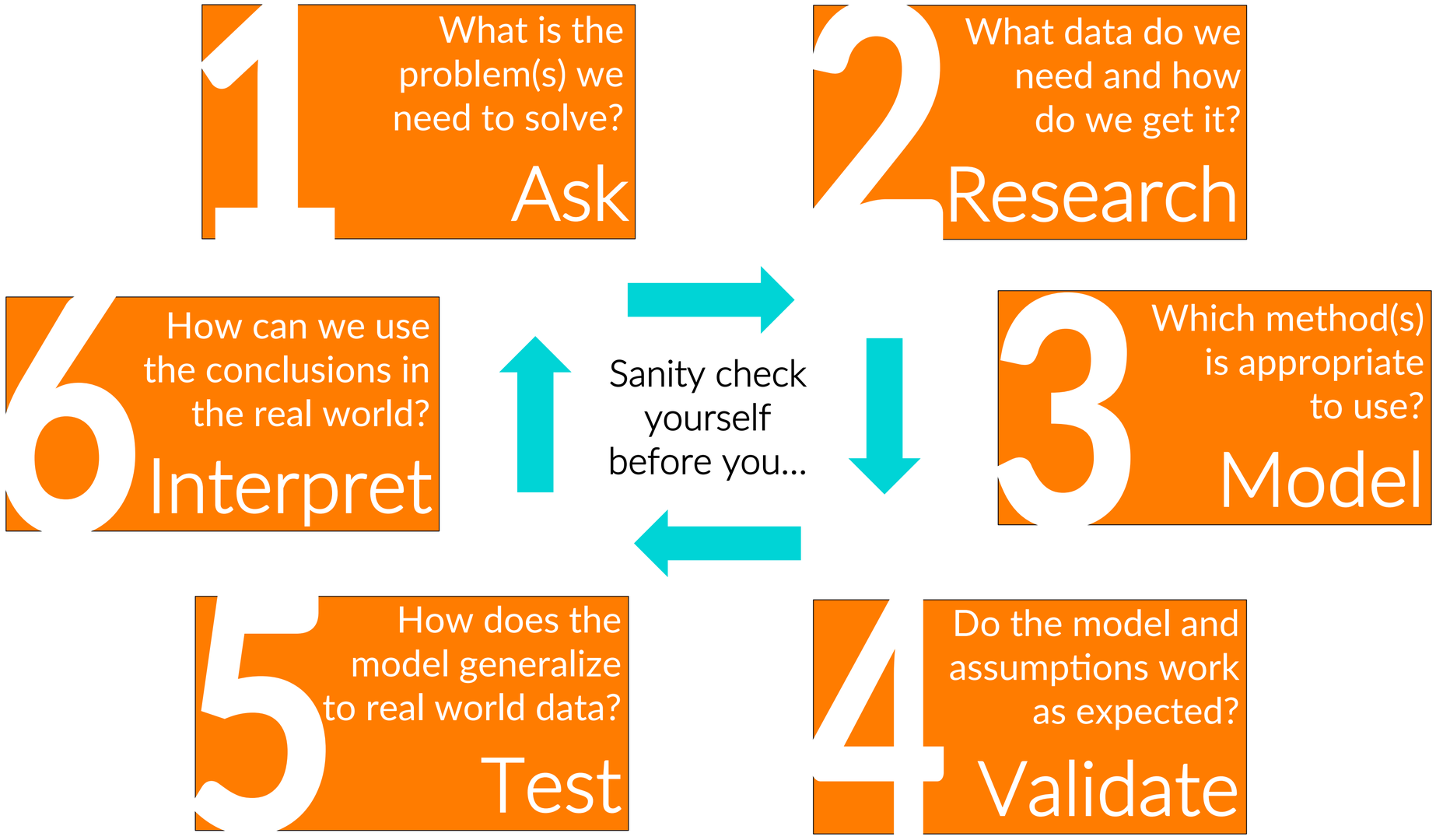
What is data mining?

- Extracting information from large quantities of data to find insights, patterns and latent connections

(Courtesy of Flickr user Jeffrey Beall)

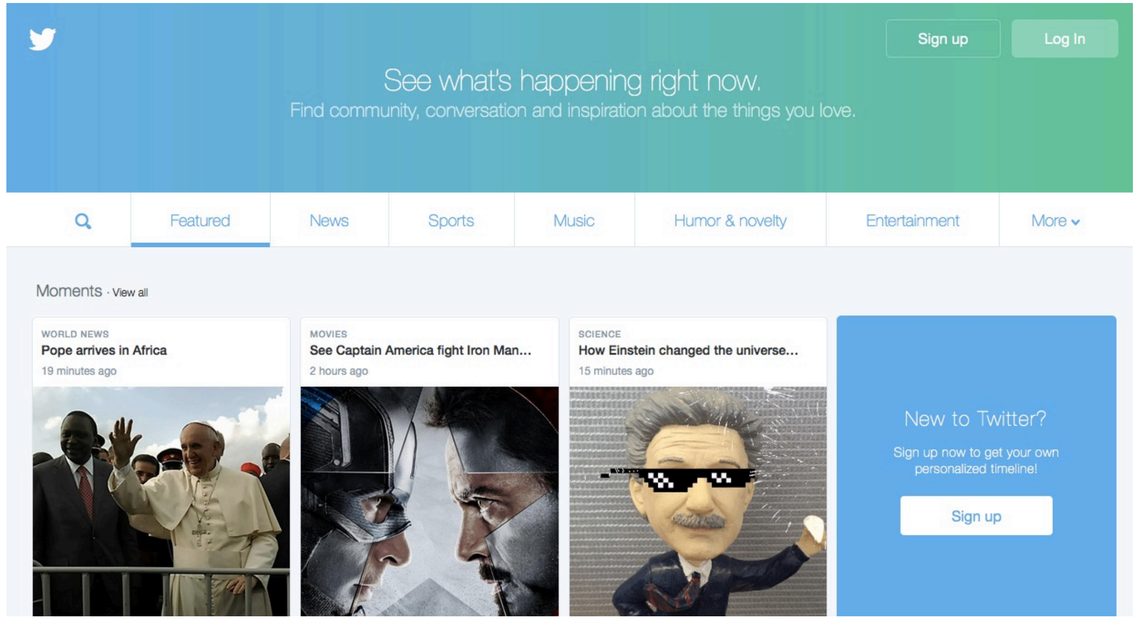
Photo from HGTV

Data science control cycle



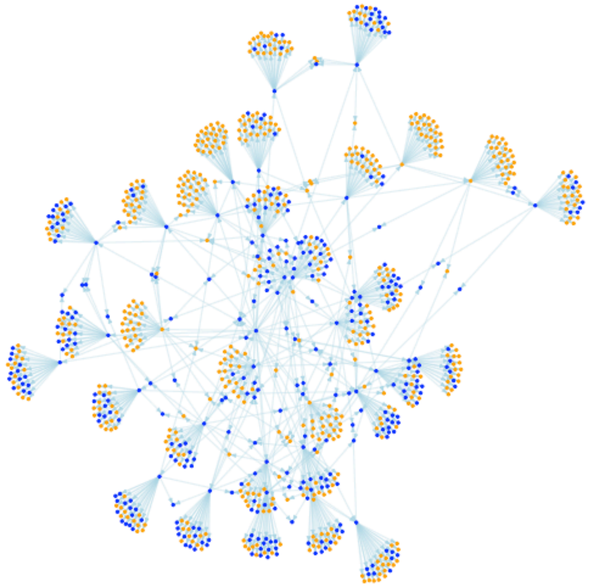
Twitter

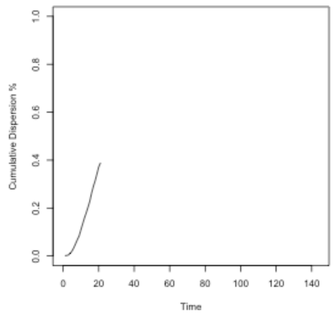
- Twitter is a micro blogging site with over 300 million users where people can post messages of up to 140 characters
- People can follow others in order to see what they are talking about
- 2 people don't have to follow each other, which makes Twitter a directed network
- Twitter is used by some as a news aggregator

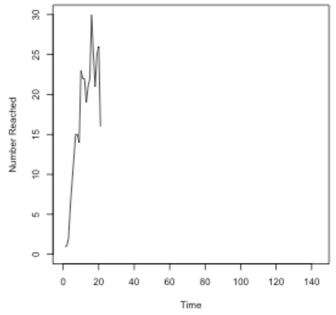


Twitter

- Users can re-broadcast other users' messages, we can see how news propagates through the network
- Twitter's following model creates more of an interest network, as opposed to a social network
- Knowing what people are interested in allows us to make recommendations and identify trends



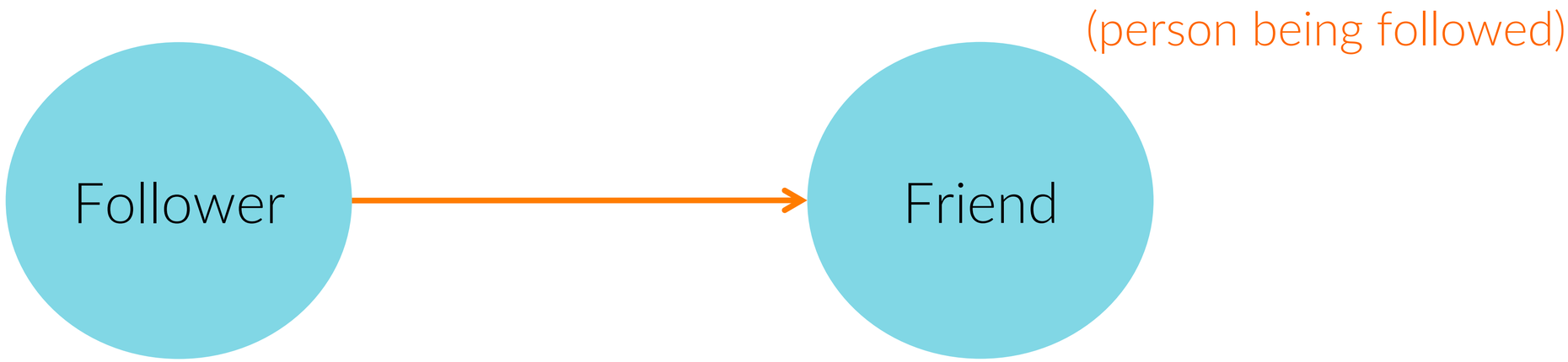




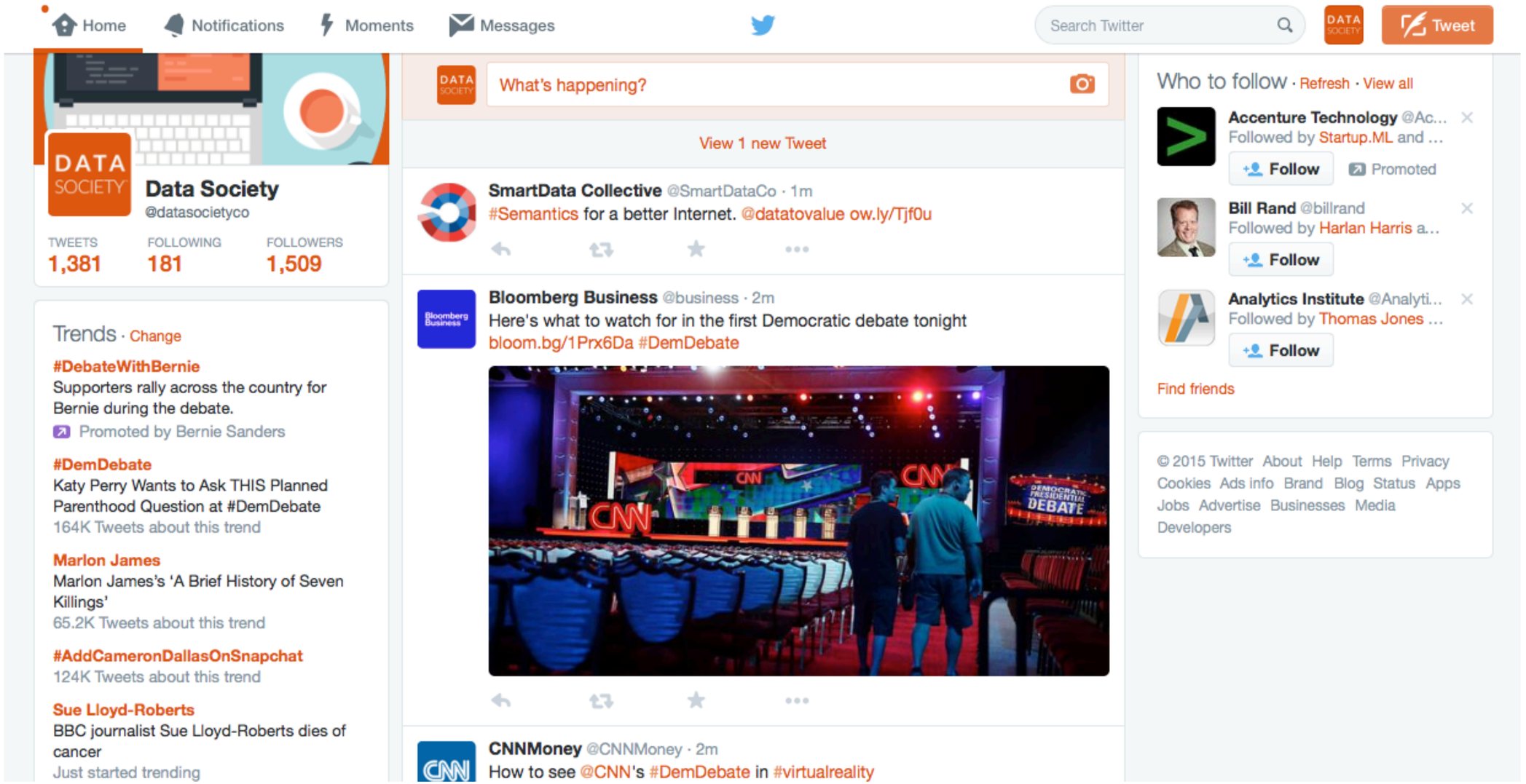
How fast does a message spread?
You'll know by the end of this course!

Some terminology

- # (Hashtag): relates to topics mentioned by other users
- Twitter handle: the user name of a person on Twitter
- @ (at): signals the start of a user's Twitter handle



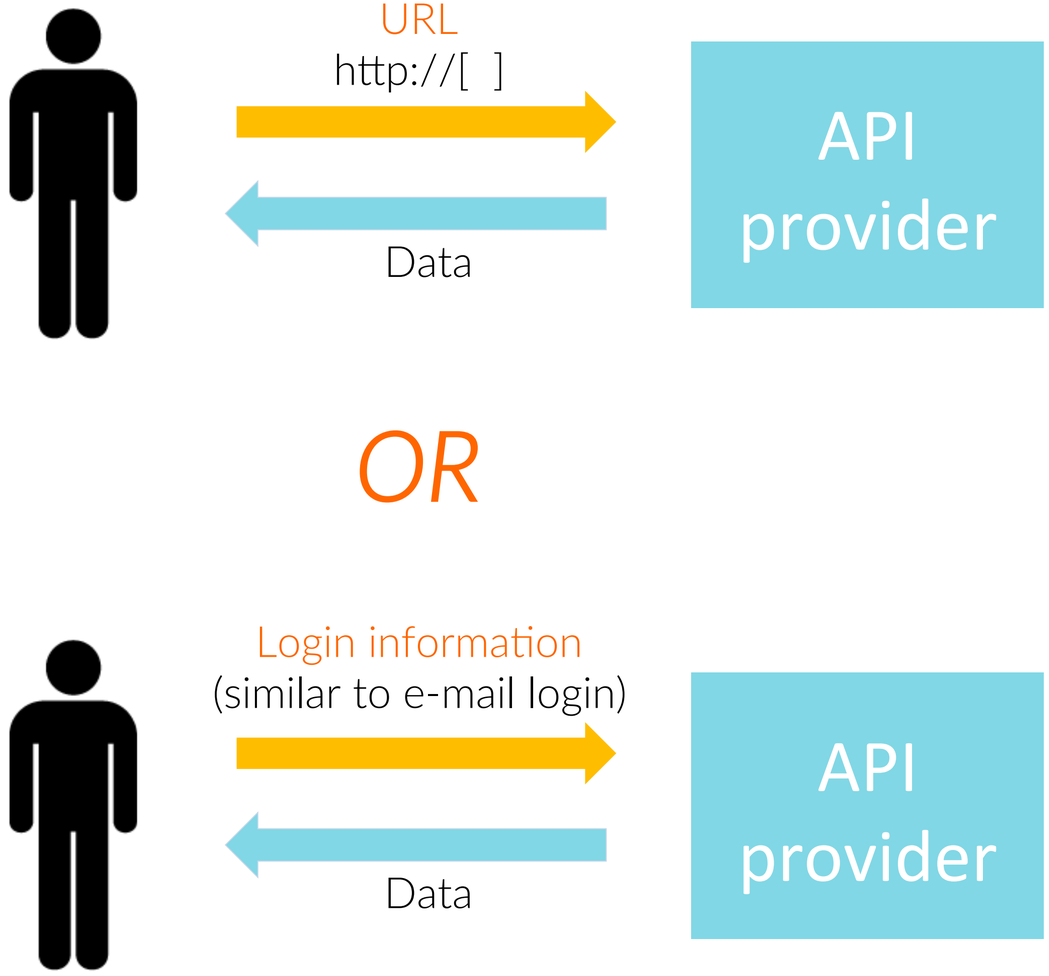
Twitter


Home
Notifications
Moments
Messages
Search Twitter
Tweet
DATA SOCIETY
Data Society
@datasocietyco
TWEETS 1,381
FOLLOWING 181
FOLLOWERS 1,509
Trends · Change
#DebateWithBernie
Supporters rally across the country for Bernie during the debate.
Promoted by Bernie Sanders
#DemDebate
Katy Perry Wants to Ask THIS Planned Parenthood Question at #DemDebate
164K Tweets about this trend
Marlon James
Marlon James's 'A Brief History of Seven Killings'
65.2K Tweets about this trend
#AddCameronDallasOnSnapchat
124K Tweets about this trend
Sue Lloyd-Roberts
BBC journalist Sue Lloyd-Roberts dies of cancer
Just started trending
What's happening?
View 1 new Tweet
SmartData Collective @SmartDataCo · 1m
#Semantics for a better Internet. @datatovalue ow.ly/Tjf0u
Bloomberg Business @business · 2m
Here's what to watch for in the first Democratic debate tonight
bloom.bg/1Prx6Da #DemDebate
CNN
DEMOCRATIC PRESIDENTIAL DEBATE
CNNMoney @CNNMoney · 2m
How to see @CNN's #DemDebate in #virtualreality
Who to follow · Refresh · View all
Accenture Technology @Ac...
Followed by Startup.ML and ...
Follow
Promoted
Bill Rand @billrand
Followed by Harlan Harris a...
Follow
Analytics Institute @Analyti...
Followed by Thomas Jones ...
Follow
Find friends
© 2015 Twitter About Help Terms Privacy Cookies Ads info Brand Blog Status Apps Jobs Advertise Businesses Media Developers

API = a fire hose of data

What is an API?

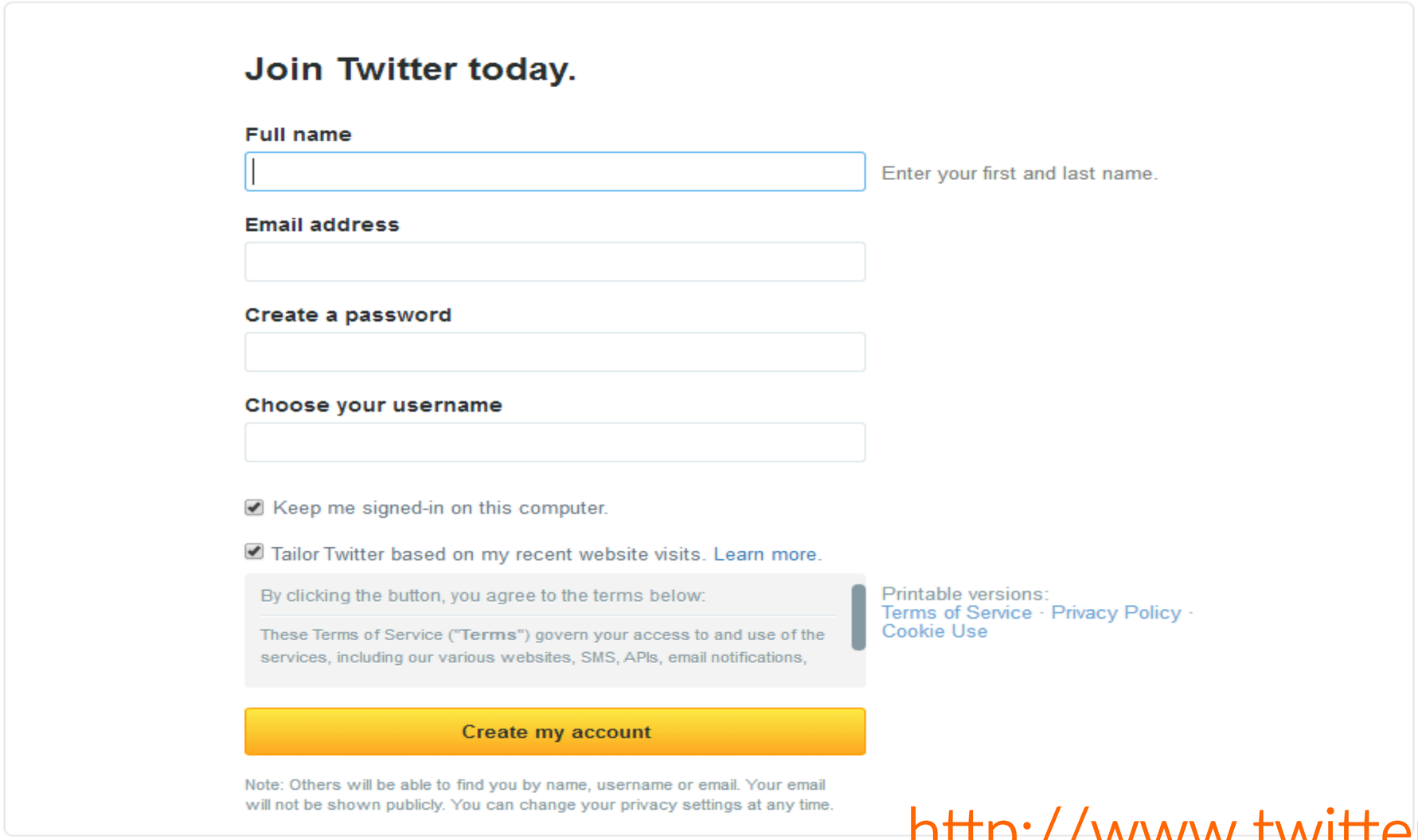
- API stands for *application programming interface*
- APIs allow you to use the data of sites like Google Maps, Pinterest, Twitter, WalMart and Best Buy
- Allows you to download large amounts of data directly from the provider's repository
- APIs allow you to custom select the subset of data that you want to use



What data can we get from Twitter?

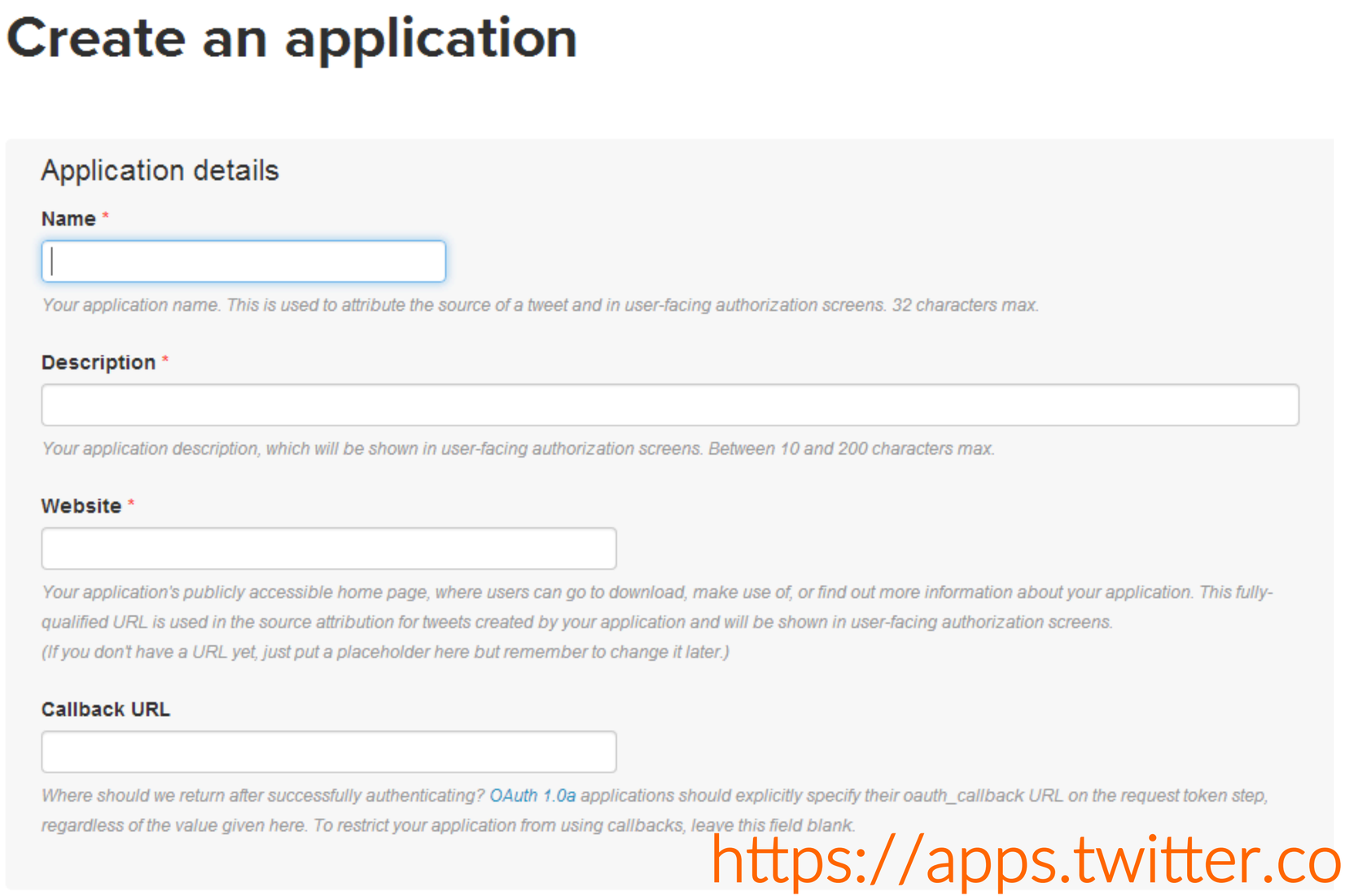
1. Text
2. Links
3. Author
4. Blurb about author
5. Location of author at time of tweet
6. Author's followers
7. Author's friends (people followed)
8. Retweets from other users
9. Time the tweet was made
10. Trending topics

Create: a Twitter account



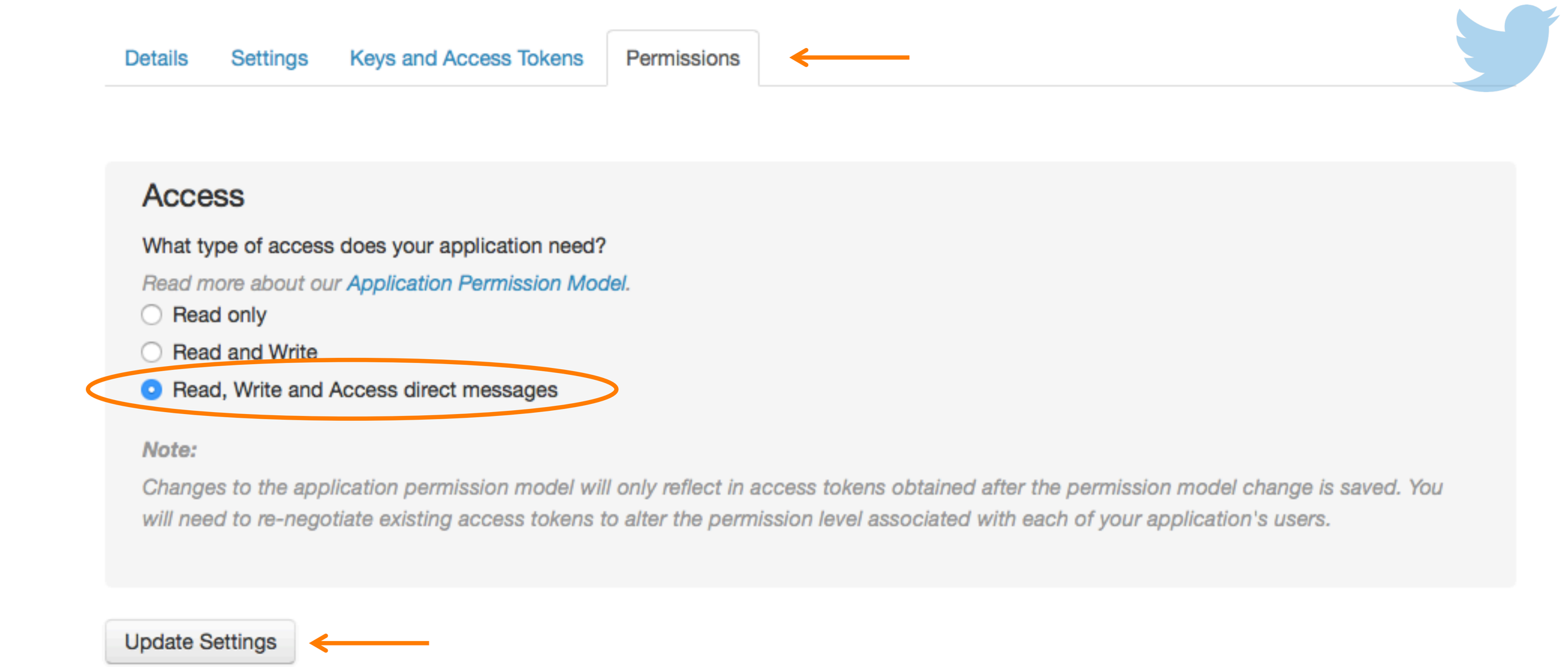
http://www.twitter.com/signup

Create: new application

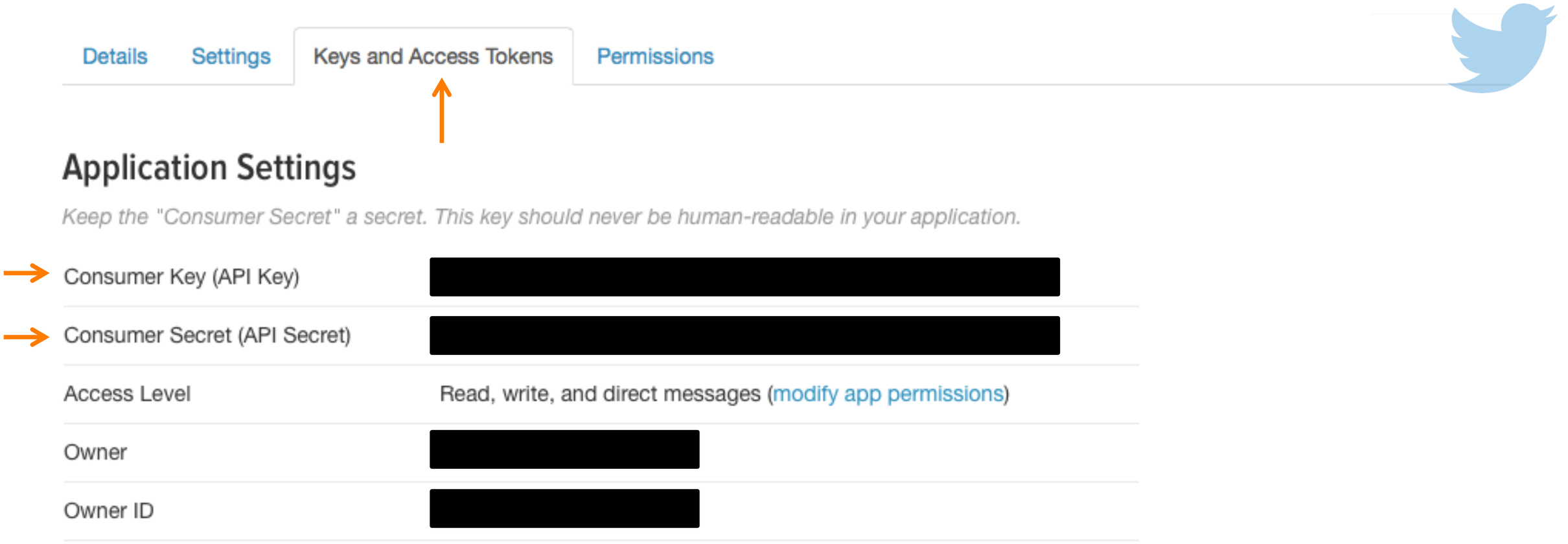


https://apps.twitter.com/app/new

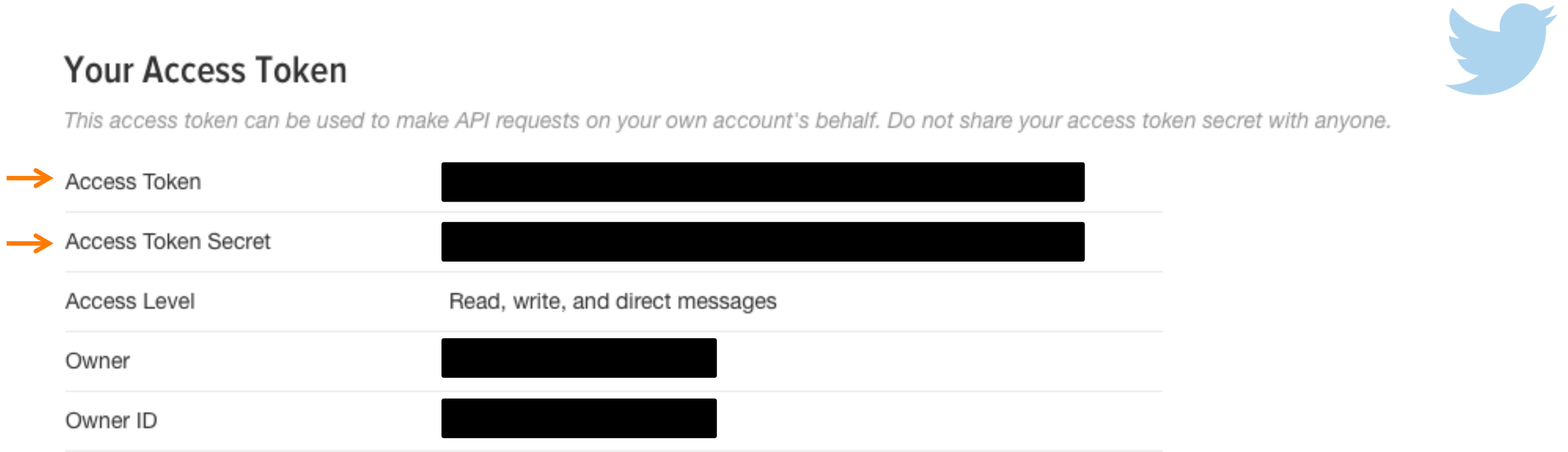
Grant: read, write, access



Store: API key, secret



Store: access token/secret



Twitter API limits

- The free version of the Twitter API allows you to search tweets from only the last 7 days, but paid version lets you access tweets going further back

Install `twitteR` package

Script

```
# Install the twitteR package.
install.packages("twitteR")
library(twitteR)
library(help = twitteR)

# Load authentication credentials.
api_key = "XxxxXXXXxxxXXX#####"
api_secret = "XXXXXxxXXXXXXXxXXXXXxxxxxxxXXXXXX#########"
access_token = "##########XXXXXxxXXXXXXXxXXXXXxxxxxxxXXXXXX#########"
access_secret = "######XXXXXxxXXXXXXXxxxxxxxXXXXXX#########"

# Ask Twitter for authentication.
setup_twitter_oauth(api_key, api_secret, access_token, access_secret)

# Once prompted between 2 choices, select 1 and press "Enter".
# If you get an error here, try re-installing or updating the twitteR package and
# check the credentials you're entering.

# Set working directory.
setwd("~/Desktop/Network Analysis/Twitter")
```

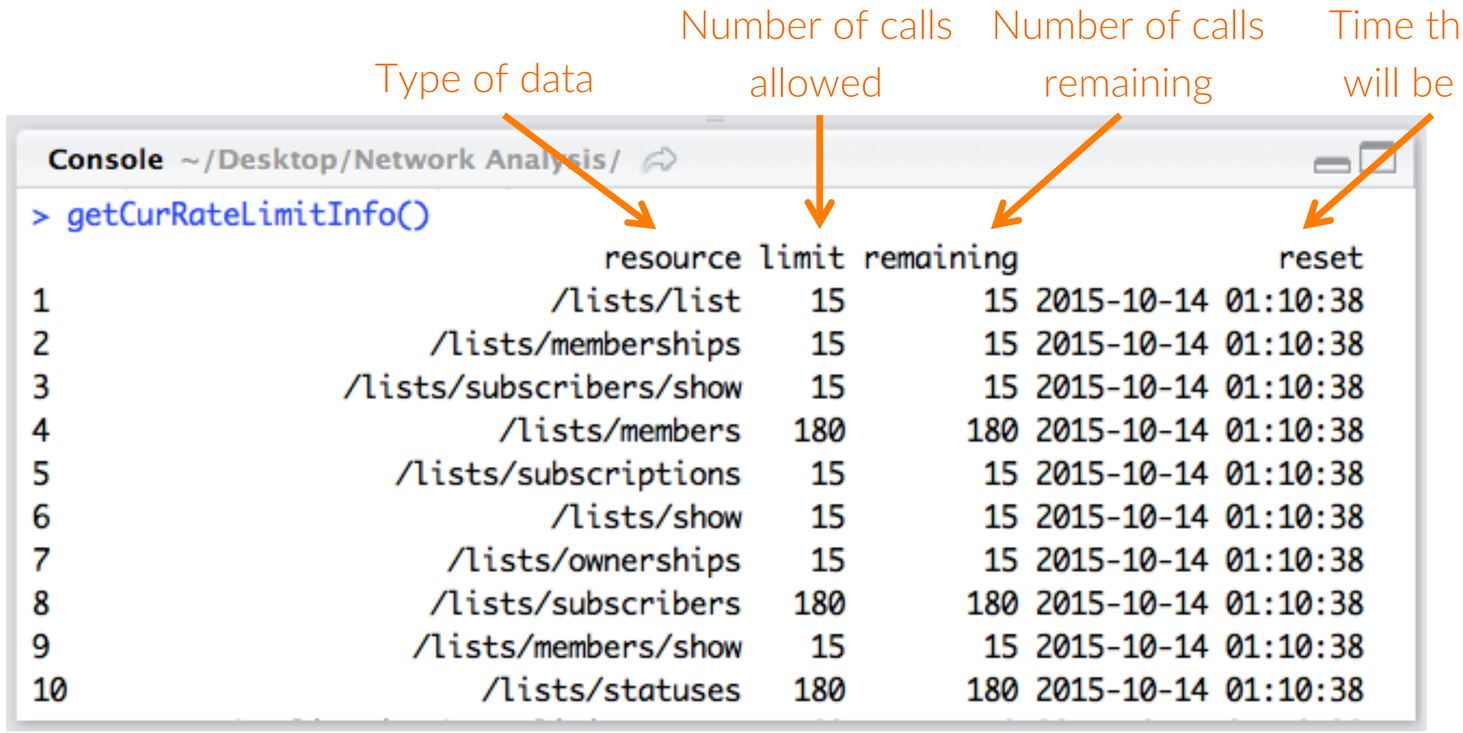
Authenticate access to Twitter


```
Console ~/Desktop/Network Analysis/
> setup_twitter_oauth(api_key, api_secret, access_token, access_secret)
[1] "Using direct authentication"
Use a local file to cache OAuth access credentials between R sessions?
1: Yes
2: No

Selection: 1
```

Basic Twitter functions

Script

```
# Get rate limit information.
getCurRateLimitInfo()

# Search Twitter for people, accounts and Tweets.
?searchTwitter
```



`searchTwitter()` functions

Argument	Function
`searchString`	Search query to issue to twitter. Use "+" to separate query terms
`n`	Maximum number of tweets to return
`lang`	Languages to use
`since`	Earliest date to pull tweets, formatted as YYYY-MM-DD
`until`	Latest date to pull tweets, formatted as YYYY-MM-DD
`geocode`	Tweets by users located within a given radius of the given latitude/longitude
`sinceID`	Tweets with IDs greater (i.e. newer) than the specified ID
`maxID`	Tweets with IDs smaller (i.e. older) than the specified ID
`resultType`	`mixed` = popular + real time results `recent` = most recent results `popular` = only the most popular results

Basic Twitter functions

Script

```
# Get information about the user. You can use our Twitter handle, @datasocietyco,
# or any other username for this example.
DSco = getUser("@datasocietyco")
DSco = t(as.data.frame(DSco))
View(DSco)

# Check how many followers the user has.
DSco = getUser("@datasocietyco")
DSco_followers = DSco$getFollowersCount()
DSco_followers

# Get a list of followers and their attributes.
followers = DSco$getFollowers()
followers
```

Produces a list of followers

`do.call()` passes a list of arguments to the `rbind()` function

```
# Convert into a data frame.
followers = do.call(rbind, lapply(followers, as.data.frame))
View(followers)
```

Combine by row

Take the followers data

Convert it to a list of data frames

```
# Get the list of column names.
colnames(followers)
```

Basic Twitter functions

Network Analysis.R | DSco

Filter

	V1
description	Data Society is an online data science education comp...
statusesCount	1381
followersCount	1509
favoritesCount	53
friendsCount	182
url	http://t.co/PvmdZct6VR
name	Data Society
created	2014-11-12 18:34:18
protected	FALSE
verified	FALSE
screenName	datasocietyco
location	Washington, DC
lang	en
id	2874107368
listedCount	186
followRequestSe...	FALSE
profileImageUrl	http://pbs.twimg.com/profile_images/545699322805...

Showing 1 to 17 of 17 entries

Network Analysis.R | DSco | followers

Filter

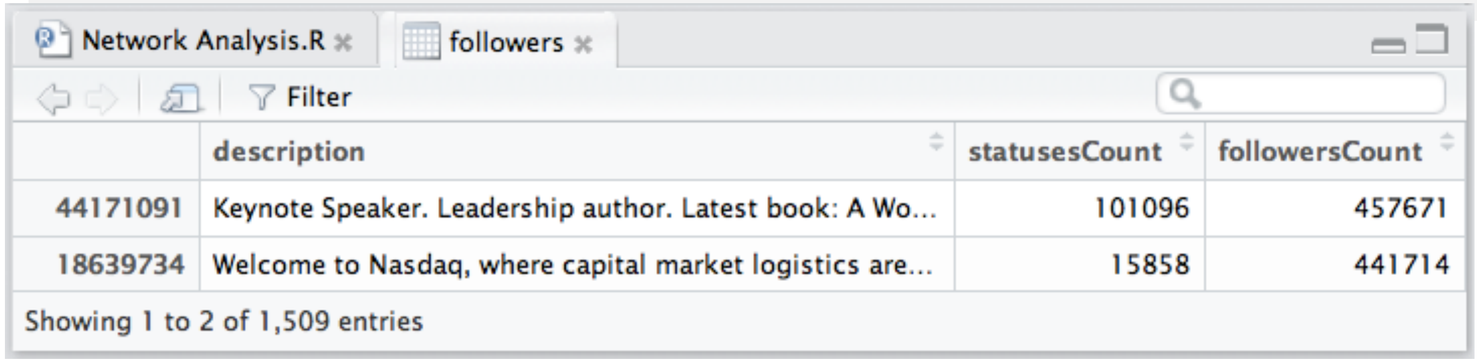
	description	statusesCount	followersCount
469446362	UXer. Consultant. Traveller. Not just another geek.	95	65
15306740...	Leading provider for #PredictiveApplications. Based on...	3899	2983
337213424	Seeking to be a data scientist, curious, maker, dreamer.	355	223
42832158	Data, DevOps, Agilité, Front-End, Back-End, Craftsman...	5494	3978
12222352...	Data Science & Front-End. Catholic. Gender Equality. Ci...	1318	135
27976026...	CTO @EXASOLAG - world's fastest database for BigData...	5322	17208
296483261	Founder and CEO Mobacar	1079	807
29788619	Saving the future, one reality at a time.	701	41
106744742	Data & Strategy Implementation,Business Intelligence I...	294	68
25590702...	Founder of @studentdatalabs. Work: Open Data Institu...	629	240
34357416...	Stock Investing Company Founded By Alexander Valent...	45	10808
200909393	OCP, CISSP, CCAH, CCDH. Interests -Databases, Hadoo...	573	253
18460009...	The world's fastest big data exploration platform, usin...	424	939
23793190...		62	154
800988624	.:: Barış Ergül ::. .:: ESOGU Dept. of Statistics ::. .:: Rese...	1612	424
32189282...	Interest: Health	4276	388
45989114	all things health & healthcare research. some yoga, hip...	2592	1153

Showing 1 to 18 of 1,509 entries

Get top followers

Script

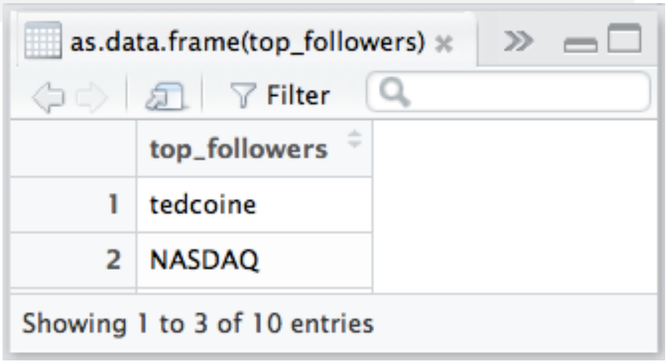
```
# Get the top 10 followers with the largest number of followers.

# Order followers by followersCount.
followers = followers[order(followers$followersCount,
                            decreasing = TRUE),]
View(followers)

# Save the file of Twitter followers.
write.csv(followers, "new Twitter followers 1.csv", row.names = FALSE)

# Select top 10 rows (top 10 followers by follower count).
top_followers = followers$screenName[1:10]
View(as.data.frame(top_followers))
```


Network Analysis.R | followers

Filter

	description	statusesCount	followersCount
44171091	Keynote Speaker. Leadership author. Latest book: A Wo...	101096	457671
18639734	Welcome to Nasdaq, where capital market logistics are...	15858	441714

Showing 1 to 2 of 1,509 entries


as.data.frame(top_followers)

Filter

	top_followers
1	tedcoine
2	NASDAQ

Showing 1 to 3 of 10 entries

Get top followers of the 10 top followers

1. Select the first Twitter user from the list you generated
2. Get 1,000 followers (starting from the most recent ones)
3. Convert to a data frame

Script

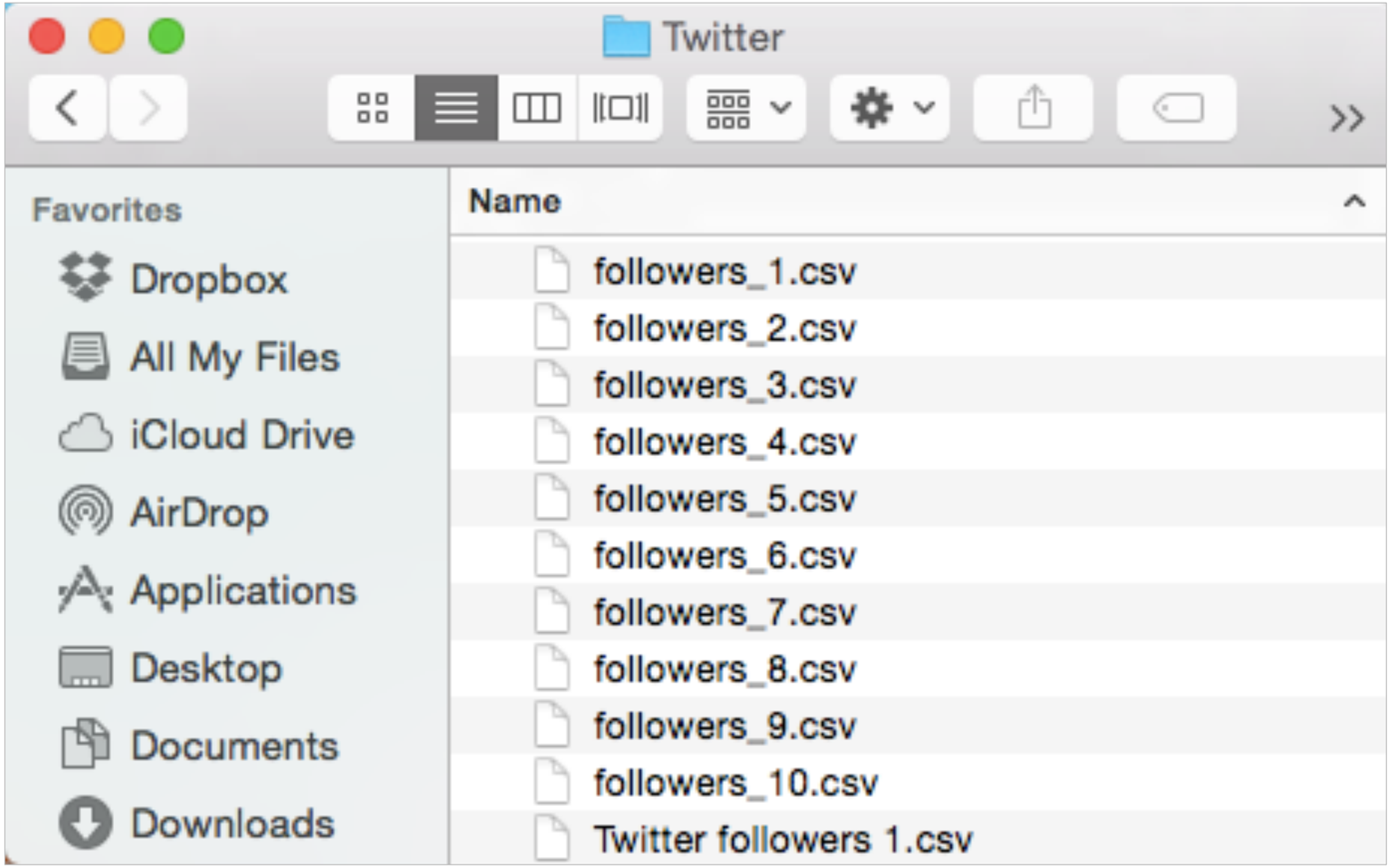
```
# Start with the 1st follower.
first = getUser(top_followers[1])
followers_2 = first$getFollowers(n = 1000)
followers_2 = do.call(rbind,
                      lapply(followers_2,
                             as.data.frame))
View(followers_2)
```

Script

```
# Create a loop to automatically save the data for the top 10 followers.
for(i in 1:length(top_followers)){
 write.csv(assign(paste0("followers_", i),
    do.call(rbind,lapply(getUser(top_followers[i])
                         $getFollowers(n = 1000),
                           as.data.frame))),
    paste0("followers_", i, ".csv"),
    row.names = FALSE)
}
```

1. Use the `write.csv()` function to save the files
2. The `assign()` function assigns data or an output to a named variable
3. `paste0()` creates a character string based on the components
4. `row.names()` prevents the column names from being in the first column of the saved file

Get top followers of the 10 top followers



Load data for top 10 followers

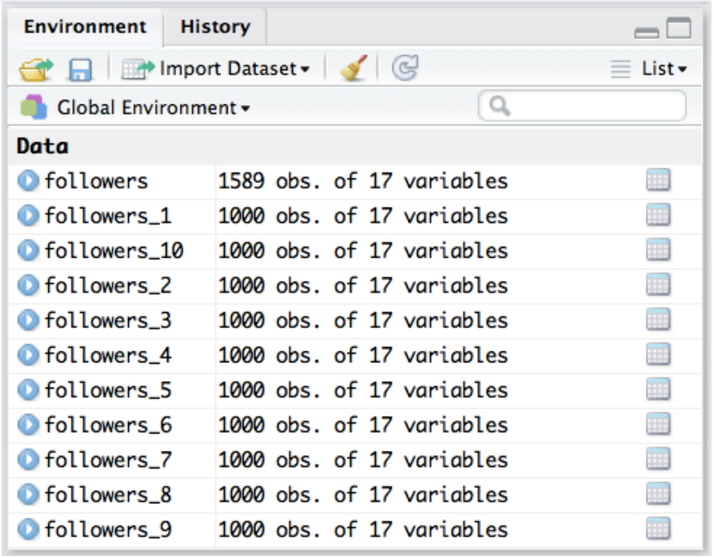
Script

```
# The next time you run this analysis, you may not want to pull new data
# all over again. To make your life easier, you can use the below loop
# to automatically load as many files into R as you want (if your computer
# can handle it) at once. This saves a lot of time!

for(i in 1:10){
  assign(paste0("followers_", i),
         read.csv(paste0("followers_", i, ".csv")))
}
```

Use the `read.csv()` function to read in the files

The `paste0()` function creates a character string based on the components

The `assign()` function assigns data or an output to a named variable



Sort data for top 10 followers

Script

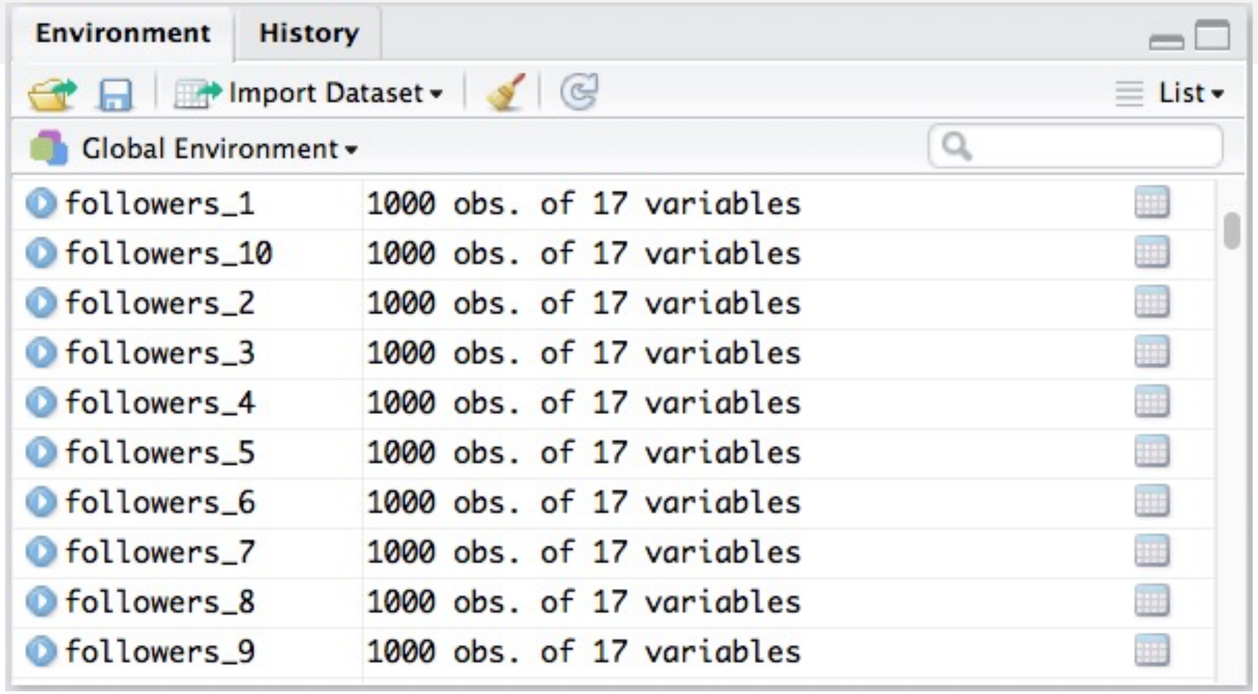
```
# Sort all the 10 data sets by number of followers.
followers = followers[order(followers$followersCount, decreasing = TRUE),]
View(followers)

# Create a loop to perform the above operation on all 10 files.
# First, have a look at a couple of functions that we will use.
paste0("followers_", 10)
parse(text = paste0("followers_", 10))
eval(parse(text = paste0("followers_", 10)))

# Compare to the output of the
# followers_10 term.
followers_10

# This formula will combine the line above with this expression:
# data[order(data$column, decreasing = TRUE),]
# data will be defined by eval(parse(text = paste0("followers_", 10)))
eval(parse(text = paste0("followers_", 10)))[order(eval(parse(text =
                                                  paste0("followers_", 10)))
                                                  $followersCount,
                                                decreasing = TRUE),]
```

1. `paste0()` creates a text string
2. `parse()` converts the text into an expression that can be manipulated, used in a calculation, etc.
3. `eval()` evaluates the expression, effectively running the variable followers_10; in this case, it displays it

Sort data for top 10 followers

Script

```
# Create a loop that will perform the operation from before.
for(i in 1:10){
  assign(paste0("followers_", i),
         eval(parse(text = paste0("followers_",
                                  i)))[order(eval(parse(text = paste0("followers_",
                                                                      i)))$followersCount,
                                             decreasing = TRUE),])
}
```



Combine 10 files into 1

Script

```
# Check the columns you're selecting every time you run this
# analysis as Twitter's response format may change.
colnames(followers)
```

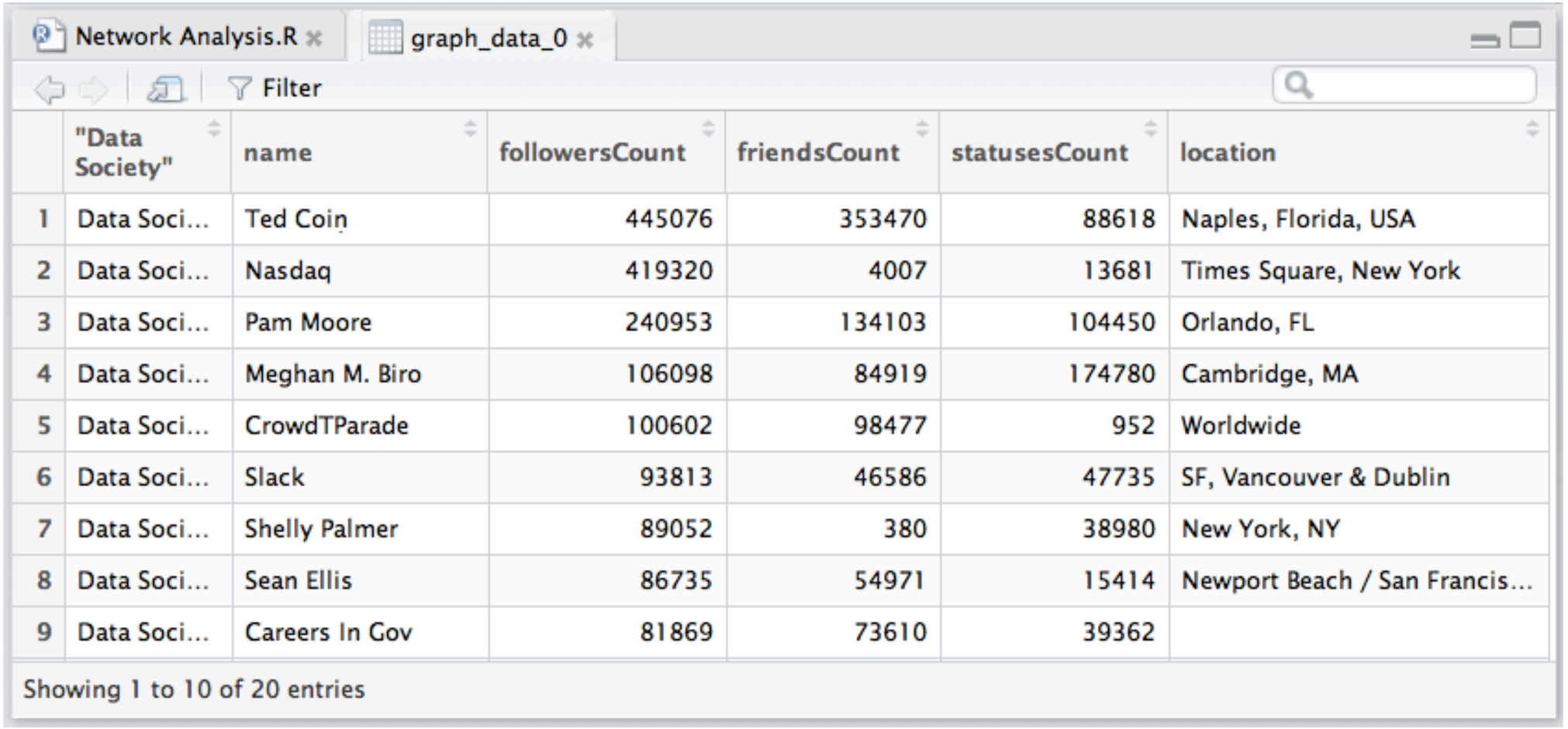
```
Console ~/Desktop/Network Analysis/Twitter/
> colnames(followers)
 [1] "description"      "statusesCount"    "followersCount"   "favoritesCount"
 [5] "friendsCount"     "url"              "name"             "created"
 [9] "protected"        "verified"         "screenName"       "location"
[13] "lang"             "id"               "listedCount"      "followRequestSent"
[17] "profileImageUrl"
```

Script

```
# We'll create a data set with the following columns:
# name, followersCount, friendsCount, statusesCount, and location
# Start with the first file.
graph_data_0 = followers[1:20, c(7, 3, 5, 2, 12)]
graph_data_0 = cbind("Data Society", graph_data_0)
View(graph_data_0)
```

1. Subset the first 20 rows and the columns you want
2. Combine the data you subsetted with a column that has the name "Data Society" so we can see the connection between the 'friend' and the 'follower'

Combine 10 files into 1



Network Analysis.R | graph_data_0

Filter

	"Data Society"	name	followersCount	friendsCount	statusesCount	location
1	Data Soci...	Ted Coin	445076	353470	88618	Naples, Florida, USA
2	Data Soci...	Nasdaq	419320	4007	13681	Times Square, New York
3	Data Soci...	Pam Moore	240953	134103	104450	Orlando, FL
4	Data Soci...	Meghan M. Biro	106098	84919	174780	Cambridge, MA
5	Data Soci...	CrowdTParade	100602	98477	952	Worldwide
6	Data Soci...	Slack	93813	46586	47735	SF, Vancouver & Dublin
7	Data Soci...	Shelly Palmer	89052	380	38980	New York, NY
8	Data Soci...	Sean Ellis	86735	54971	15414	Newport Beach / San Francis...
9	Data Soci...	Careers In Gov	81869	73610	39362	

Showing 1 to 10 of 20 entries

Combine 10 files into 1

Script

```
# Let's write a loop to automate this process.
# Let's first create a list of the names of the top 10 followers.
top_names = as.character(followers$name)[1:10]
str(top_names)
```

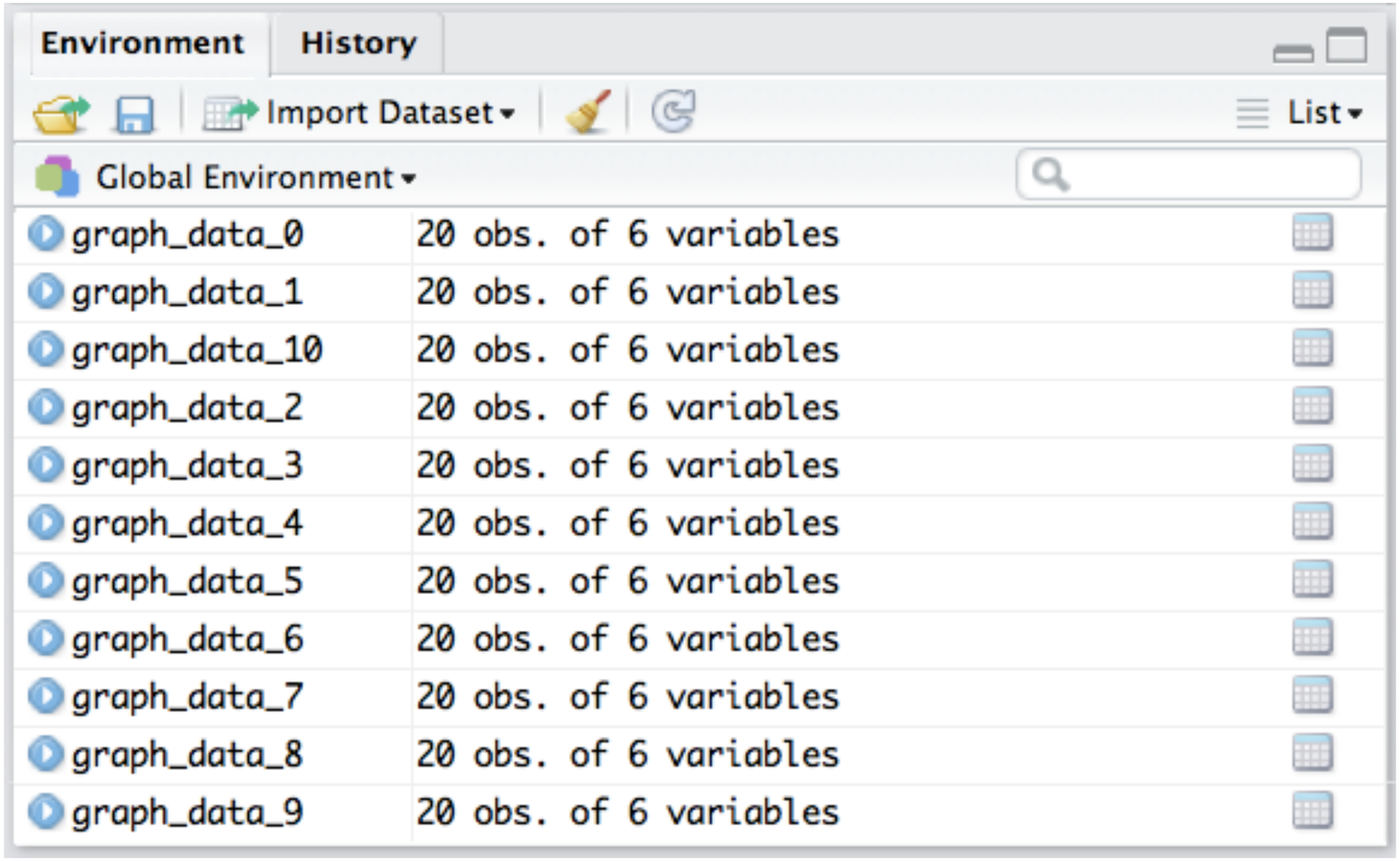
```
Console ~/Desktop/Network Analysis/Twitter/
> str(top_names)
 chr [1:10] "Ted Coin""| __truncated__ "Nasdaq" "Pam Moore" "Meghan M. Biro" "CrowdTParade" "Slack" ...
```

Script

```
for(i in 1:10){
  assign(paste0("graph_data_", i),
         cbind(top_names[i],
               eval(parse(text = paste0("followers_", i)))[1:20, c(7, 3, 5, 2, 12)]))
}
```

1. `paste0()` creates a text string
2. `parse()` converts the text into an expression that can be manipulated, used in a calculation, etc.
3. `eval()` evaluates the expression, effectively running the variable followers_10, in this case it displays it

Combine 10 files into 1

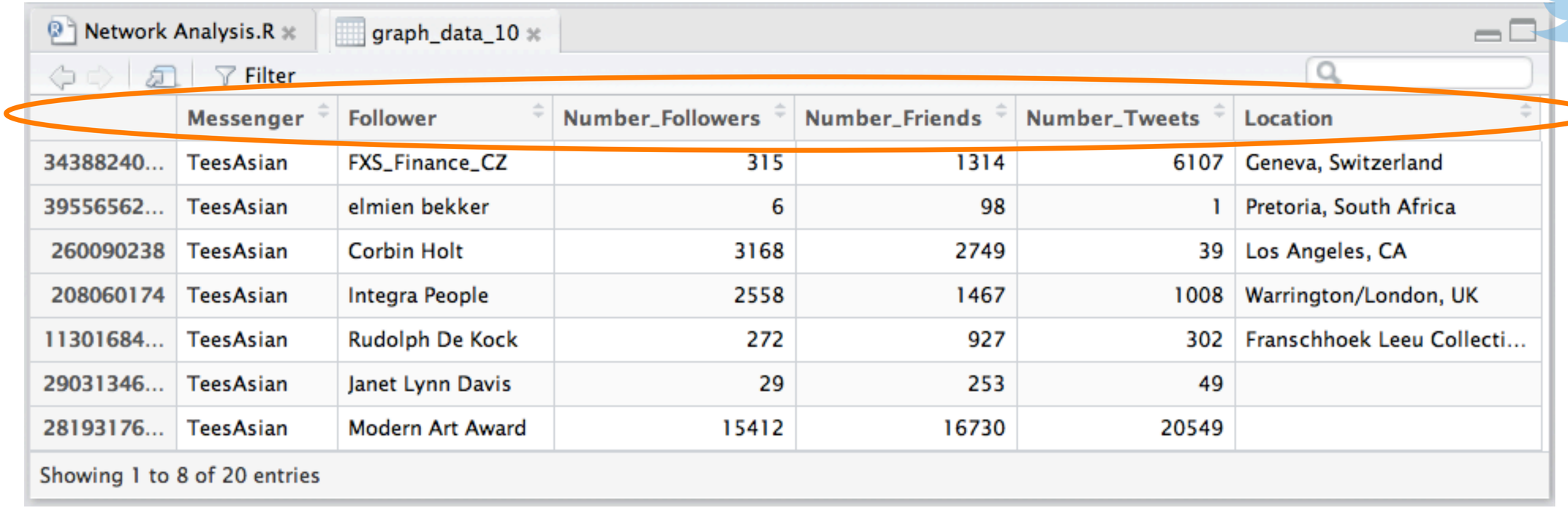


Combine 10 files into 1

Script

```
# Let's rename the columns of all the files we just created using the
# setnames() function, which will rename all the columns:
# Messenger, Follower, Number_Followers, Number_Friends, Number_Tweets and Location
# We'll use the data.table package for that.
install.packages("data.table")
library(data.table)
library(help = data.table)

setnames(graph_data_0,
         c("Messenger", "Follower", "Number_Followers",
           "Number_Friends", "Number_Tweets", "Location"))
View(graph_data_0)

# Let's automate this process with a loop.
for(i in 1:10){
  setnames(eval(parse(text = paste0("graph_data_", i))),
           c("Messenger", "Follower", "Number_Followers",
             "Number_Friends", "Number_Tweets", "Location"))
}
```

Combine 10 files into 1



Network Analysis.R | graph_data_10

Filter

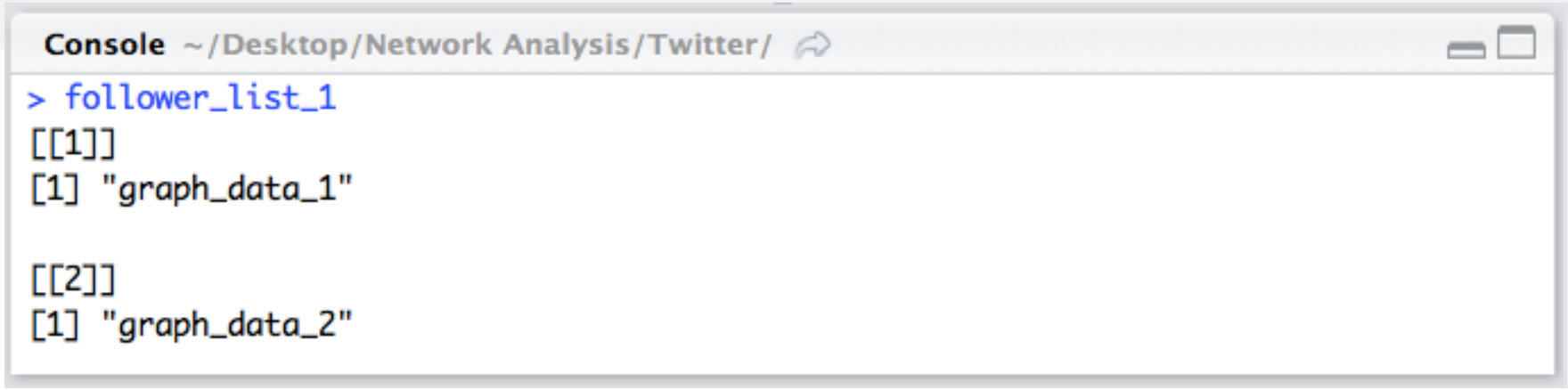
	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location
34388240...	TeesAsian	FXS_Finance_CZ	315	1314	6107	Geneva, Switzerland
39556562...	TeesAsian	elmien bekker	6	98	1	Pretoria, South Africa
260090238	TeesAsian	Corbin Holt	3168	2749	39	Los Angeles, CA
208060174	TeesAsian	Integra People	2558	1467	1008	Warrington/London, UK
11301684...	TeesAsian	Rudolph De Kock	272	927	302	Franschhoek Leeu Collecti...
29031346...	TeesAsian	Janet Lynn Davis	29	253	49	
28193176...	TeesAsian	Modern Art Award	15412	16730	20549	

Showing 1 to 8 of 20 entries

Combine 10 files into 1

Script

```
# Now combine all of these files into 1 data frame.
# First, create a blank list with file names.
follower_list_1 = list()

# Create a loop to automatically paste all the file names
# into the empty list you just created.
for(i in 1:10){
  follower_list_1[i] = c(paste0("graph_data_", i))
}

# Look at your results.
follower_list_1
```

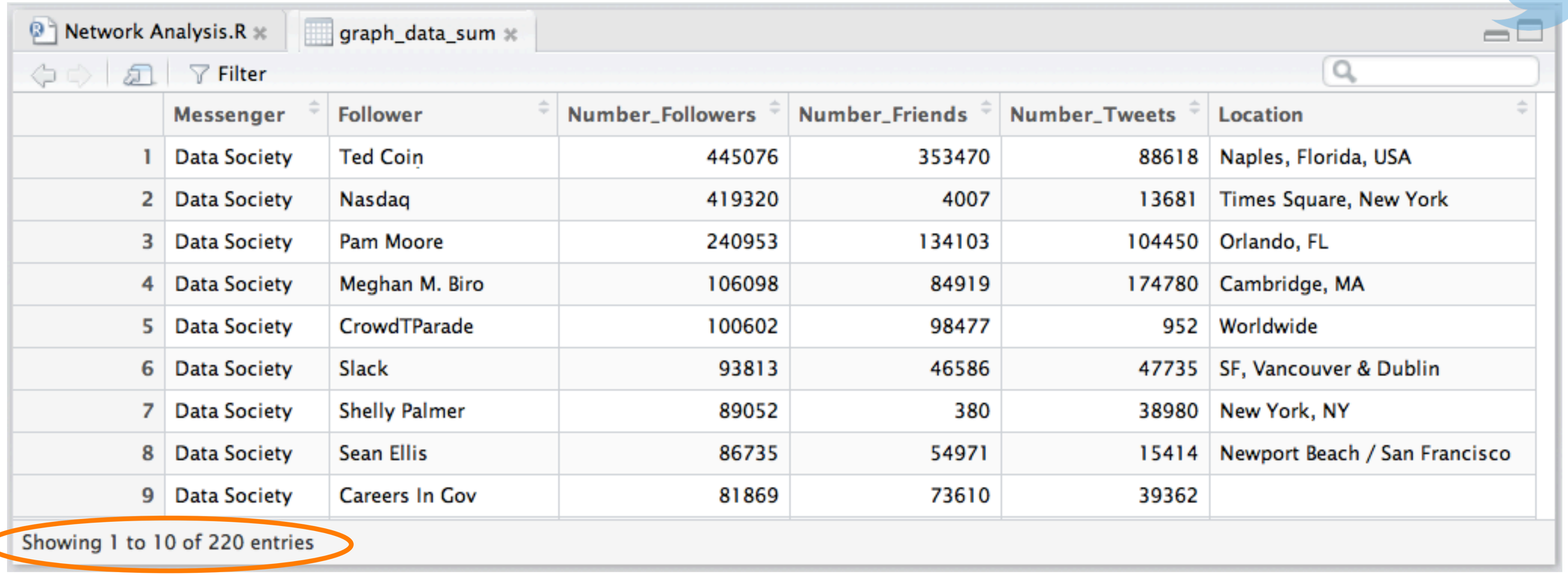


Combine 10 files into 1

Script

```
# Add all the data from each data frame that you'd like to combine as an
# element in the list you just created "follower_list_1". The lapply() command
# applies a function to a list in the format "lapply(list, function)".
# The get() command searches for an object and returns its contents.
get("graph_data_1")

follower_data_list_1 = lapply(follower_list_1, get)
follower_data_list_1

# Combine the data by combining the data to create a new data frame.
# The do.call() function executes a function onto a list of arguments.
graph_data_sum_0 = do.call(rbind, follower_data_list_1)
View(graph_data_sum_0)

# Add the original graph_data_0 file you created.
graph_data_sum = rbind(graph_data_0, graph_data_sum_0)
View(graph_data_sum)

# Save your work so you don't lose the data.
write.csv(graph_data_sum, "new_Twitter followers summary.csv", row.names = FALSE)
```

Combine 10 files into 1



Network Analysis.R | graph_data_sum

Filter

	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location
1	Data Society	Ted Coin	445076	353470	88618	Naples, Florida, USA
2	Data Society	Nasdaq	419320	4007	13681	Times Square, New York
3	Data Society	Pam Moore	240953	134103	104450	Orlando, FL
4	Data Society	Meghan M. Biro	106098	84919	174780	Cambridge, MA
5	Data Society	CrowdTParade	100602	98477	952	Worldwide
6	Data Society	Slack	93813	46586	47735	SF, Vancouver & Dublin
7	Data Society	Shelly Palmer	89052	380	38980	New York, NY
8	Data Society	Sean Ellis	86735	54971	15414	Newport Beach / San Francisco
9	Data Society	Careers In Gov	81869	73610	39362	

Showing 1 to 10 of 220 entries

Clean the data

Script

```
# Read in the clean data frame we provided for you.
graph_data_sum = read.csv("Twitter followers summary.csv",
                          check.names = FALSE)

# Remove punctuation with the gsub() function - the [[:punct:]] notation identifies
# punctuation and the "" argument tells R to replace it with nothing.
graph_data_sum$Messenger = gsub("[[:punct:]]", "", graph_data_sum$Messenger)
graph_data_sum$Follower = gsub("[[:punct:]]", "", graph_data_sum$Follower)

# View the data
View(graph_data_sum)
```

Set up network data

Script

```
# Make sure the igraph package is loaded. No need to do it again if you've,
# already loaded the library in your current R session.
install.packages("igraph")
library(igraph)
library(help = igraph)

# Set up the network data.
network_graph = graph.data.frame(graph_data_sum,
                                 directed = TRUE)

str(network_graph)
```

Since Twitter is a directed network, let's present it that way

```
Console ~/Desktop/Network Analysis/Twitter/
> str(network_graph)
IGRAPH DN-- 214 220 --
+ attr: name (v/c), Number_Followers (e/n), Number_Friends (e/n),
| Number_Tweets (e/n), Location (e/c)
+ edges (vertex names):
  [1] Data Society  ->Ted Coin
  [2] Data Society  ->Nasdaq
  [3] Data Society  ->Pam Moore
```

This is showing 214 nodes and 220 edges

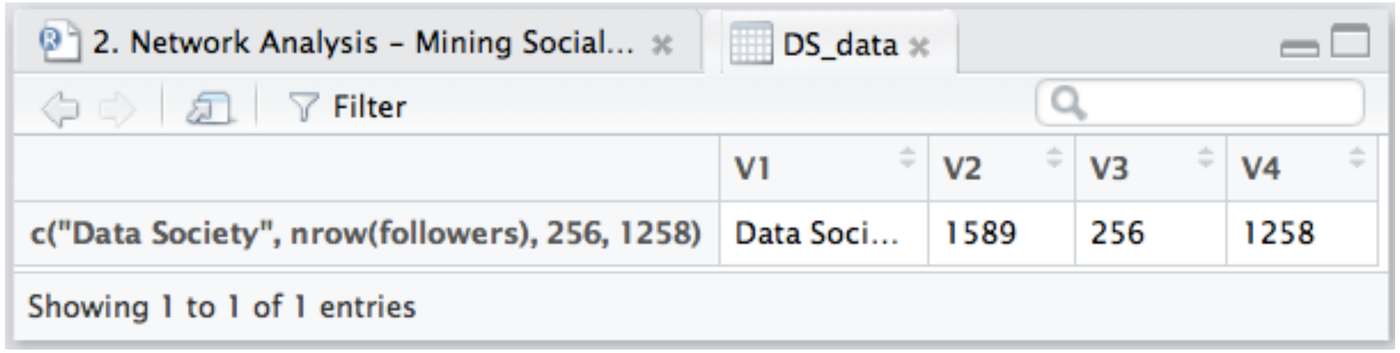
R thinks the attributes are related to the edges instead of vertices

Set up network data

Script

```
# Check the column names of the original data set.
colnames(graph_data_sum)

# Add another row to the data with attributes about the first node, Data Society.
# The transpose function "t()" will make the data set horizontal.
DS_data = t(as.data.frame(c("Data Society", nrow(followers), 182, 1258)))
View(DS_data)

# Rename the rows and columns to conform to the summary data set and
# cast it as a data frame.
colnames(DS_data) = c("Follower", "Number_Followers", "Number_Friends",
                      "Number_Tweets")
row.names(DS_data) = c("1")
DS_data = as.data.frame(DS_data)

# To combine the data frames the columns need to use the same structure.
str(graph_data_sum[, 2:5])
str(DS_data)
```

Set up network data



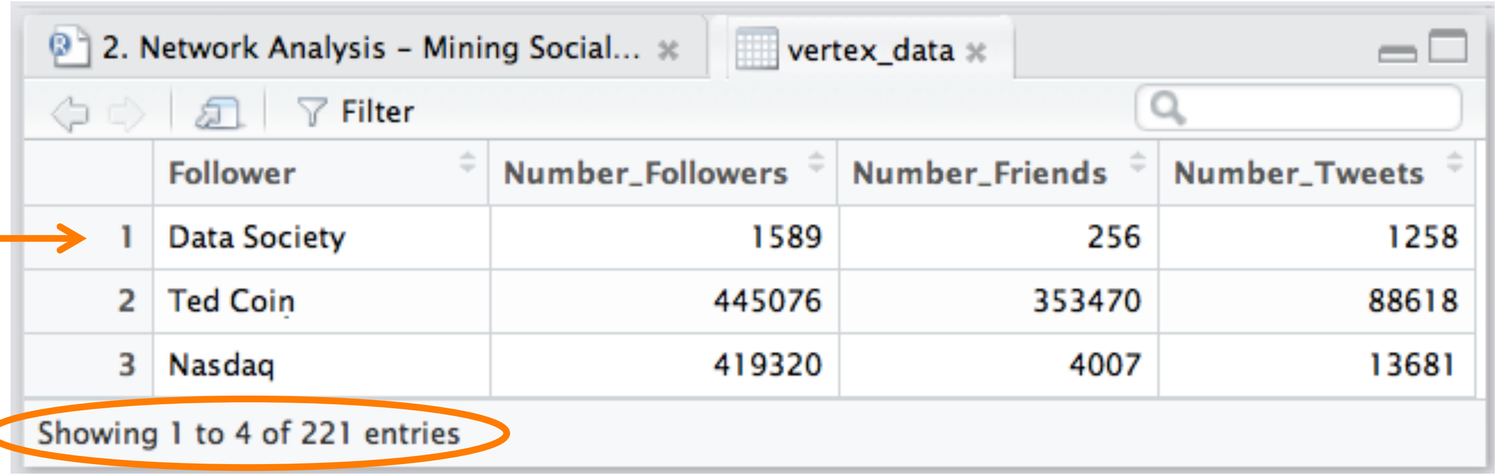

```
Console ~/Desktop/Network Analysis/Twitter/
> str(graph_data_sum[, 2:5])
'data.frame':   220 obs. of  4 variables:
 $ Follower        : chr  "Ted Coin" "Nasdaq" "Pam Moore" "Meghan M Biro" ...
 $ Number_Followers: int  445076 419320 240953 106098 100602 93813 89052 86735 81869 70876 ...
 $ Number_Friends  : int  353470 4007 134103 84919 98477 46586 380 54971 73610 67204 ...
 $ Number_Tweets   : num  88618 13681 104450 174780 952 ...
> str(DS_data)
'data.frame':   1 obs. of  4 variables:
 $ Follower        : Factor w/ 1 level "Data Society": 1
 $ Number_Followers: Factor w/ 1 level "1589": 1
 $ Number_Friends  : Factor w/ 1 level "256": 1
 $ Number_Tweets   : Factor w/ 1 level "1258": 1
```

To merge the data frames the columns need to have the same structure

Set up network data

Script

```
# Convert the structure of the DS_data columns to match that of graph_data_sum.
DS_data$Follower = as.character(DS_data$Follower)
DS_data$Number_Followers = as.integer(as.character(DS_data$Number_Followers))
DS_data$Number_Friends = as.integer(as.character(DS_data$Number_Friends))
DS_data$Number_Tweets = as.numeric(as.character(DS_data$Number_Tweets))

# Combine the data sets.
vertex_data = rbind(DS_data, graph_data_sum[, 2:5])
View(vertex_data)
```



2. Network Analysis - Mining Social... | vertex_data

Filter

	Follower	Number_Followers	Number_Friends	Number_Tweets
1	Data Society	1589	256	1258
2	Ted Coin	445076	353470	88618
3	Nasdaq	419320	4007	13681

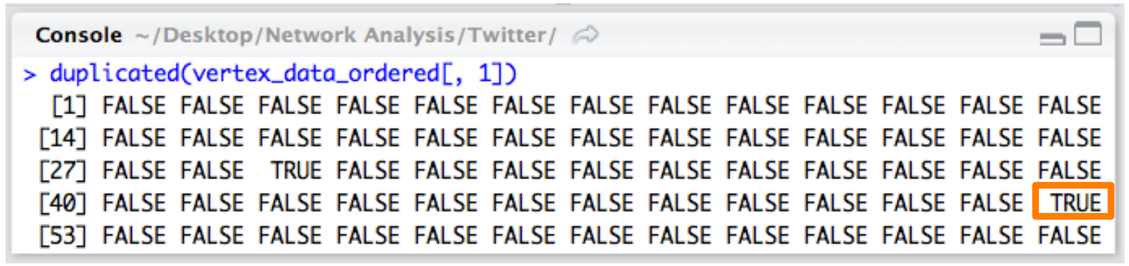
Showing 1 to 4 of 221 entries

Set up network data

Script

```
# Let's eliminate rows where the Twitter account is duplicated
# and only keep the one with the greatest number of followers. That is likely
# the most current record.

# First, sort the data set by the decreasing number of followers.
vertex_data_ordered = vertex_data[order(vertex_data$Number_Followers,
                                        decreasing = TRUE), ]

# The duplicated() function in the data.table package returns a vector
# indicating which rows are duplicates of prior rows. Take a look at what
# the function returns when looking at the first column of vertex_data.
library(data.table)
duplicated(vertex_data_ordered[, 1])
View(vertex_data_ordered)

# Let's remove the duplicate rows by using the subset function.
# Recall that "!" means "not" in R.
vertex_data_clean = subset(vertex_data_ordered,
                           !duplicated(vertex_data_ordered[, 1]))
View(vertex_data_clean)
```

Set up network data



The `duplicated()` function identifies the 2nd row that repeats a value in a data frame

2. Network Analysis – Mining Social... | vertex_data_ordered

Filter

	Follower	Number_Followers	Number_Friends	Number_Tweets
194	Ethan Silver	191926	36736	2815
85	Ethan Silver	191925	36736	2815
44	JUAN JOSÉ MOLINA B	187349	186216	20892
86	Prowebsite Solutions	184647	12398	10005
168	Joe Floccari TV News	172275	176017	946

Showing 51 to 56 of 221 entries

2. Network Analysis – Mining Social... | vertex_data_clean

Filter

	Follower	Number_Followers	Number_Friends	Number_Tweets
202	Alaattin Çağıl	2304423	329820	2032
203	Ali Spagnola	2189177	1420892	20426
204	Brian Hazard	1838439	1439238	30194
182	АРТЕМ КЛЮШИН □	1438429	505503	25533
162	Old Photos Bacon	1129098	298760	18690

Showing 1 to 6 of 214 entries

By removing duplicate account entries we cut the data set down by 7 records

Setup network data

Script

```
# Set up the network data with the graph.data.frame() function.
network_graph = graph.data.frame(graph_data_sum[, c(1:2, 5)],
                                 directed = TRUE,
                                 vertices = vertex_data_clean)

# Check the structure of the output.
str(network_graph)
```

1. Attributes that define edges
2. Since Twitter is a directed network, let's present it that way
3. This data set defines the attributes of the vertices

```
Console ~/Desktop/Network Analysis/Twitter/
> str(network_graph)
IGRAPH DN-- 214 220 --
+ attr: name (v/c), Number_Followers (v/n), Number_Friends (v/n), Number_Tweets (v/n), Number_Tweets (e/n)
+ edges (vertex names):
  [1] Data Society  ->Ted Coin                Data Society  ->Nasdaq
  [3] Data Society  ->Pam Moore               Data Society  ->Meghan M Biro
  [5] Data Society  ->CrowdTParade            Data Society  ->Slack
```

`(v/n)` denotes vertex attributes and `(e/n)` denotes edge attributes

Plot a directed graph

Script

```
# Check to make sure graph attributes look right before you use them in your
# visualization. The V() function pulls attributes of graph vertices.
# The E() function pulls attributes of graph edges.
paste0(V(network_graph)$name,
       " ",
       V(network_graph)$Number_Followers)
```

1. The name of the vertex
2. A blank space
3. The number of followers of a vertex

```
Console ~/Desktop/Network Analysis/Twitter/
> paste0(V(network_graph)$name,
+        " ",
+        V(network_graph)$Number_Followers)
  [1] "Alaattin Çağıl 2304423"              "Ali Spagnola 2189177"
  [3] "Brian Hazard 1838439"                "АРТЕМ КЛЮШИН □ 1438429"
  [5] "Old Photos  Bacon 1129098"           "Sam Pepper 1064218"
```

2. Network Analysis - Mining Social... | vertex_data_clean

Filter

	Follower	Number_Followers	Number_Friends	Number_Tweets
202	Alaattin Çağıl	2304423	329820	2032
203	Ali Spagnola	2189177	1420892	20426
204	Brian Hazard	1838439	1439238	30194
182	АРТЕМ КЛЮШИН □	1438429	505503	25533
162	Old Photos Bacon	1129098	298760	18690
42	Sam Pepper	1064218	7329	17002

Showing 1 to 7 of 214 entries

Plot a directed graph

Script

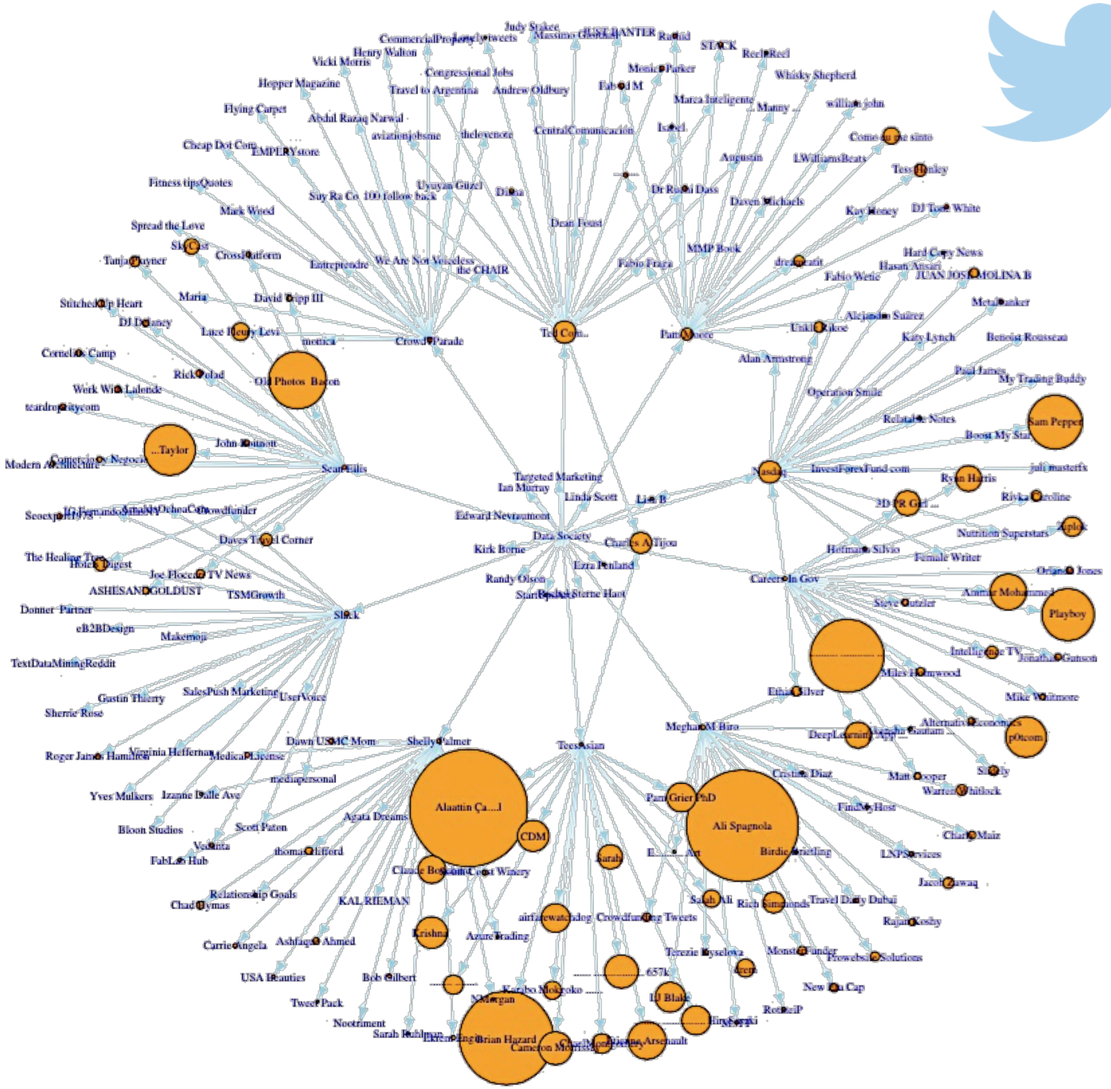
```
# If you're working with large amounts of data that would take R long to view,
# you can save the image as a pdf first, run the plotting command and then
# run dev.off() to complete the saving process. The image won't
# appear in R but you can open the pdf from your directory.
pdf("Twitter followers_directed_220.pdf",
    width = 20,
    height = 20)

# Plot the graph and set vertex (node) size based on number of followers.
set.seed(5)
plot(layout = layout.kamada.kawai,
     network_graph,
     # vertex.label = NA,
     vertex.label.cex = 1,
     vertex.size = V(network_graph)$Number_Followers/100000,
     edge.arrow.size = 1,
     vertex.color = "orange",
     edge.color = "light blue")

dev.off()
```

1. Method used to plot graph
2. Data set used
3. The graph will be labeled
4. Size of labels
5. Vertex size
6. Size of arrows

Ends the saving process

Plot a directed graph

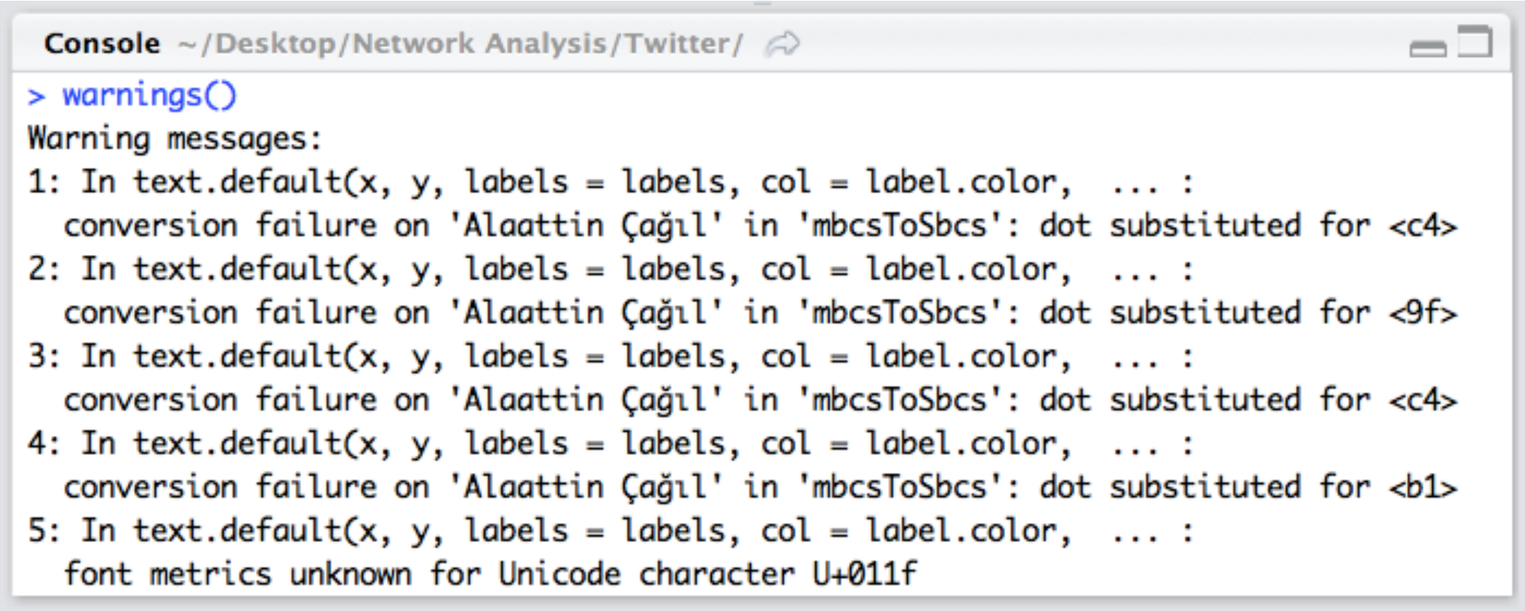
- What can you see (hint: view your pdf and zoom in to see better)?
- You see how the original user is connected to other users with many followers
- Arrows show the flow of information, they go towards the follower



Plot a directed graph

Script

```
# Learn about other layouts you can use for network graphs.
?layout.kamada.kawai

# Check the warnings to see if there is anything you should be concerned about.
warnings()
```



This error tells you that R is having trouble displaying the text correctly. You can adjust the encoding of the text with the `Encoding()` function but that's outside of the scope of this course

View connections by # of tweets

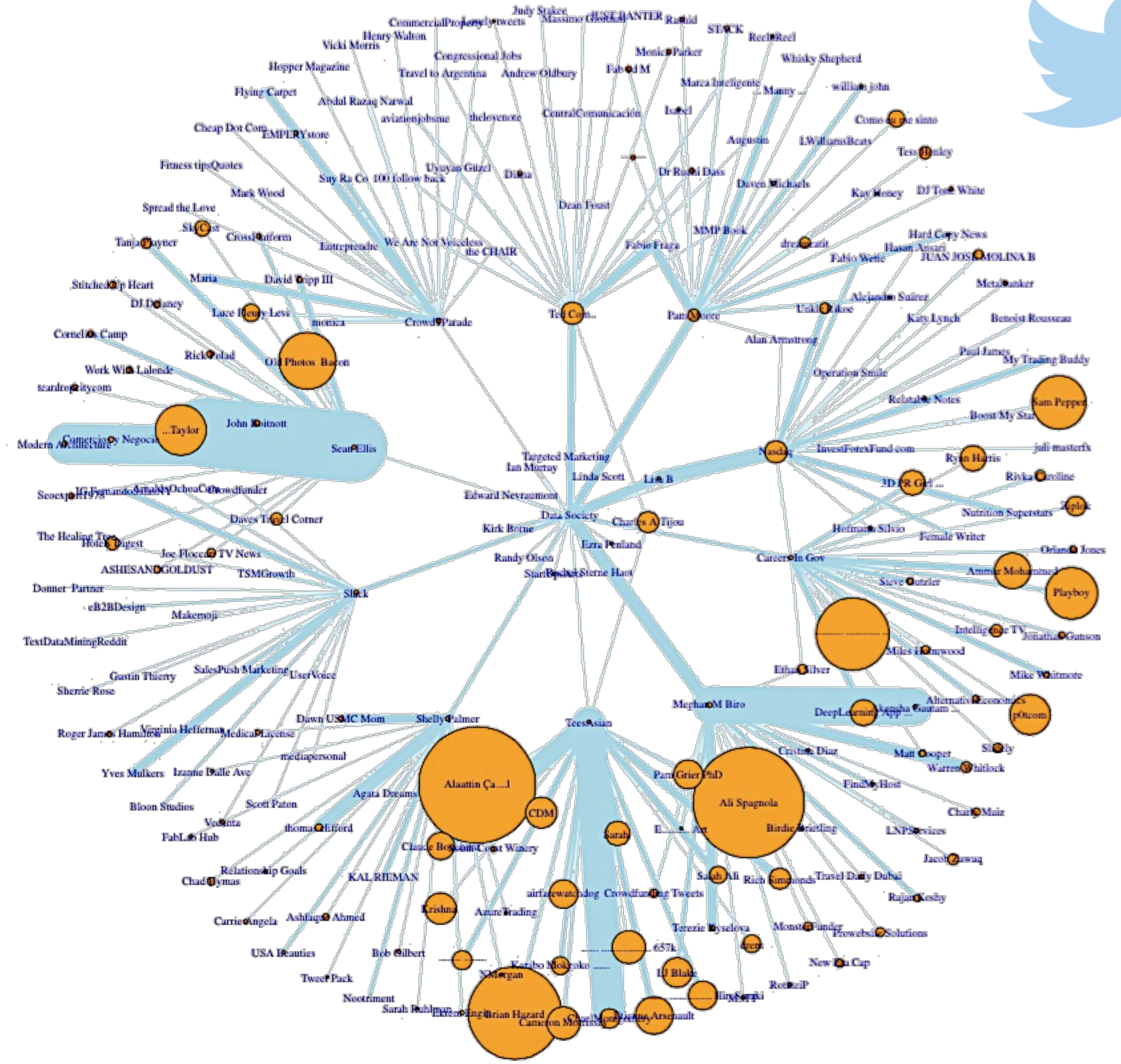
Script

```
# Create an undirected network graph.
network_graph_2 = graph.data.frame(graph_data_sum[, c(1:2, 5)],
                                   directed = FALSE,
                                   vertices = vertex_data_clean)

# Save as a PDF first so it's easier to see.
pdf("Twitter followers_tweets_220.pdf", width = 20, height = 20)

# Plot the graph and set vertex (node) size based on number of followers.
set.seed(5)
plot(layout = layout.kamada.kawai,
     network_graph,
     # vertex.label = NA,
     vertex.label.cex = 1,
     vertex.size = V(network_graph_2)$Number_Followers/100000,
     edge.width = E(network_graph_2)$Number_Tweets/10000,
     vertex.color = "orange",
     edge.color = "light blue",
     margin = 0)

dev.off()
```

1. Method used to plot graph
2. Data set used
3. The graph will be labeled
4. Size of labels
5. Vertex size
6. Size of edge width
7. Vertex color
8. Edge color
9. Margin around the graph

View connections by # of tweets

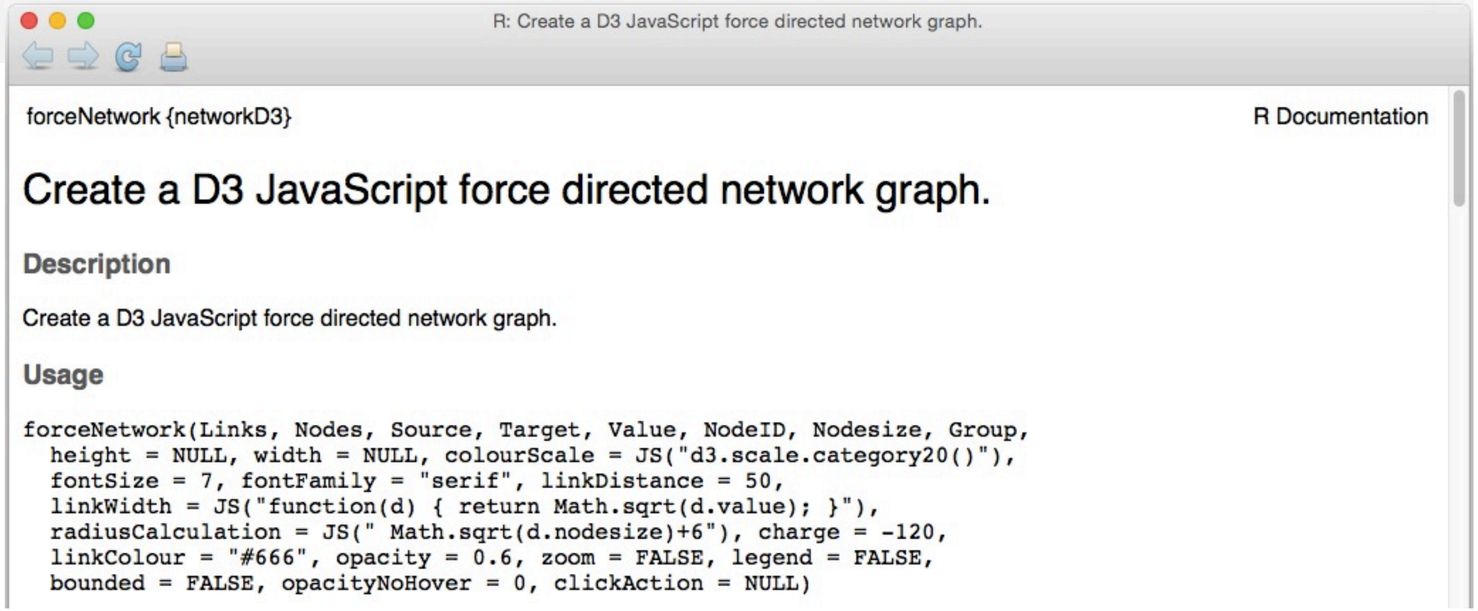
- What can you see (hint: view your pdf and zoom in to see better)?
- You see how the original user is connected to other users with many followers
- You can see how much people actually tweet – note quantity ≠ quality
  - Look at tweets over time



Interactive vis: setting up data

Script

```
# Let's plot a dynamic network graph where the name of each vertex
# comes up when you scroll over it.
install.packages("networkD3")
library(networkD3)
library(help = networkD3)

# We need to create 2 files for a forceNetwork graph to display correctly.
# Learn about the function first.
?forceNetwork
```



R: Create a D3 JavaScript force directed network graph.

forceNetwork {networkD3} R Documentation

Create a D3 JavaScript force directed network graph.

Description

Create a D3 JavaScript force directed network graph.

Usage

```
forceNetwork(Links, Nodes, Source, Target, Value, NodeID, Nodesize, Group,
  height = NULL, width = NULL, colourScale = JS("d3.scale.category20()"),
  fontSize = 7, fontFamily = "serif", linkDistance = 50,
  linkWidth = JS("function(d) { return Math.sqrt(d.value); }"),
  radiusCalculation = JS(" Math.sqrt(d.nodesize)+6"), charge = -120,
  linkColour = "#666", opacity = 0.6, zoom = FALSE, legend = FALSE,
  bounded = FALSE, opacityNoHover = 0, clickAction = NULL)
```

Interactive vis: setting up data

Script

```
# We will need to create 2 files.
# The first file will have 3 columns:
# beginning of an edge, end of an edge, and thickness of the connecting line.
# The second file will have 2 columns:
# names of the vertices and group number that each vertex belongs to.
View(graph_data_sum)
```

2. Network Analysis - Mining Social... | graph_data_sum

Filter

	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location
1	Data Society	Ted Coin	445076	353470	88618	Naples, Florida, USA
2	Data Society	Nasdaq	419320	4007	13681	Times Square, New York
3	Data Society	Pam Moore	240953	134103	104450	Orlando, FL

Showing 1 to 3 of 220 entries

Interactive vis: setting up data

Script

```
# Let's assign a number to each person in our network.
str(vertex_data)
vertex_names_unique = unique(vertex_data$Follower)

# Let's assign each name to a number by adding a column with 214 numbers.
# The forceNetwork function we will use requires that the first number be 0.
number = seq(length(vertex_names_unique))
network_names = cbind(vertex_names_unique, number - 1)
network_names = as.data.frame(network_names)
colnames(network_names) = c("Messenger", "Source")

# Make sure R is reading the data correctly.
str(network_names)
network_names$Messenger = as.character(network_names$Messenger)
network_names$Source = as.numeric(as.character(network_names$Source))
```

1. Limit the sequence to the number of nodes in the network

```
Console ~/Desktop/Network Analysis/Twitter/
> str(network_names)
'data.frame':   214 obs. of  2 variables:
 $ Messenger: Factor w/ 214 levels " Ethan Silver ",..: 50 183 133 143 122 49 170 167 165 30 ...
 $ Source   : Factor w/ 214 levels "0","1","10","100",..: 1 2 113 138 149 160 171 182 193 204 ...
```

Interactive vis: setting up data

Script

```
# Now let's create our first data set for the network graph with 3 columns:
# A. Beginning of an edge; B. End of an edge; C. Thickness of the connecting line
library(plyr)
Links_1 = join(as.data.frame(graph_data_sum),
               as.data.frame(network_names),
               by = "Messenger")
View(Links_1)

# Now let's repeat this process with the "Follower" column.
network_names_2 = network_names
colnames(network_names_2) = c("Follower", "Target")
Links_2 = join(as.data.frame(Links_1),
               as.data.frame(network_names_2),
               by = "Follower")
View(Links_2)
```

← Rename the columns so R can use `join()`

2. Network Analysis - Mining Social... | Links_2

Filter

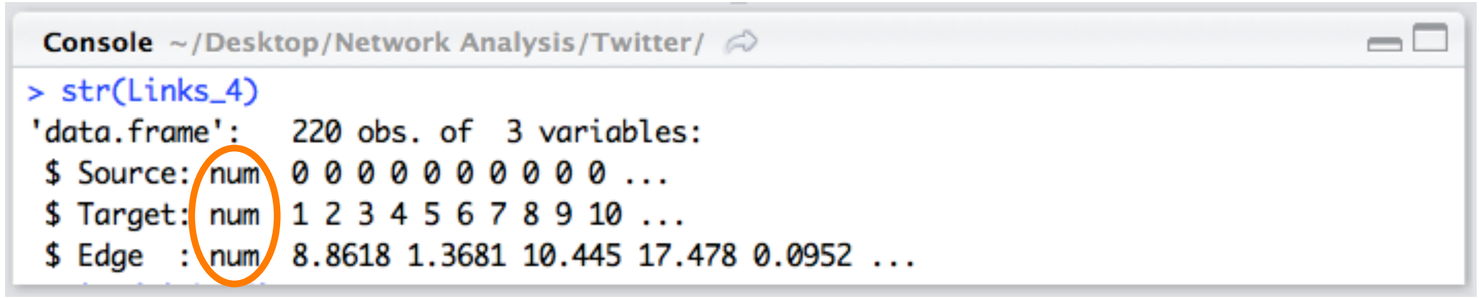
	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location	Source	Target
1	Data Society	Ted Coin	445076	353470	88618	Naples, Florida, USA	0	1

Showing 1 to 1 of 220 entries

Interactive vis: setting up data

Script

```
# Now let's subset the columns we want for the first data set.
Links_3 = Links_2[, 7:8]
str(Links_3)

# Make sure the ID columns are read as numbers. You'll need to use the
# as.numeric(as.character()) construct in order for the factors to convert correctly.
Links_3$Source = as.numeric(as.character(Links_3$Source))
Links_3$Target = as.numeric(as.character(Links_3$Target))
str(Links_3)

# Let's use the number of tweets to define the thickness of a connecting line.
Links_4 = cbind(Links_3, graph_data_sum$Number_Tweets/10000)
colnames(Links_4) = c("Source", "Target", "Edge")
str(Links_4)
```



Interactive vis: setting up data

Script

```
# Now let's create the second data set with 2 columns:
# A. Names of the vertices
# B. Group number that each vertex belongs to
# Each row represents the number of the vertex and contains the associated name.
# The "Group" column identifies the color of the point.
# Let's assign the color based on the primary person followed or
# 1 color for every 20 entries (11 colors in total for 220 entries).

# The RColorBrewer package contains nice pre-set color schemes.
install.packages(RColorBrewer)
library(RColorBrewer)
library(help = RColorBrewer)

# Display all palettes, zoom in to see the names better.
display.brewer.all()

# Spectral looks like a nice palette, let's use that one.
# brewer.pal() selects a number of colors from the palette we specify.
spectral_colors = brewer.pal(11, "Spectral")
View(as.data.frame(spectral_colors))
```

Interactive vis: setting up data

as.data.frame(spectral_colors)

Filter

	spectral_colors
1	#9E0142
2	#D53E4F
3	#F46D43
4	#FDAE61
5	#FEE08B
6	#FFFFBF
7	#E6F598
8	#ABDDA4
9	#66C2A5
10	#3288BD
11	#5E4FA2

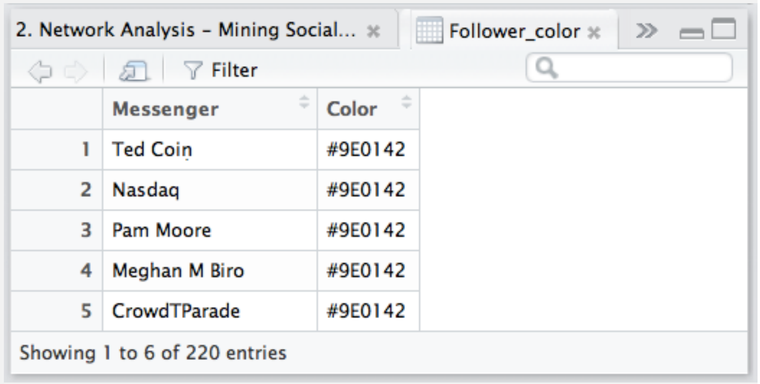
Showing 1 to 11 of 11 entries

Interactive vis: setting up data

Script

1. Select an element of spectral_colors
2. Repeat it 20 times

```
# Create a data frame of colors.
group_colors = c(rep(spectral_colors[1], 20),
                 rep(spectral_colors[2], 20),
                 rep(spectral_colors[3], 20),
                 ...continue spectral colors until 11...
                rep(spectral_colors[11], 20))
View(as.data.frame(group_colors))

# Combine with the "Follower" column of Links_2.
# Again make sure R is reading all data types correctly.
View(Links_2)
Follower_color = cbind(as.character(Links_2$Follower), group_colors)
colnames(Follower_color) = c("Messenger", "Color")
View(Follower_color)

# Recall that there are 214 unique nodes in the network.
# Remove any rows where the Messenger name is duplicated because they're
# following several people, and keep just the first assigned color.
Follower_color_clean = subset(Follower_color,
                              !duplicated(Follower_color[, 1]))
```



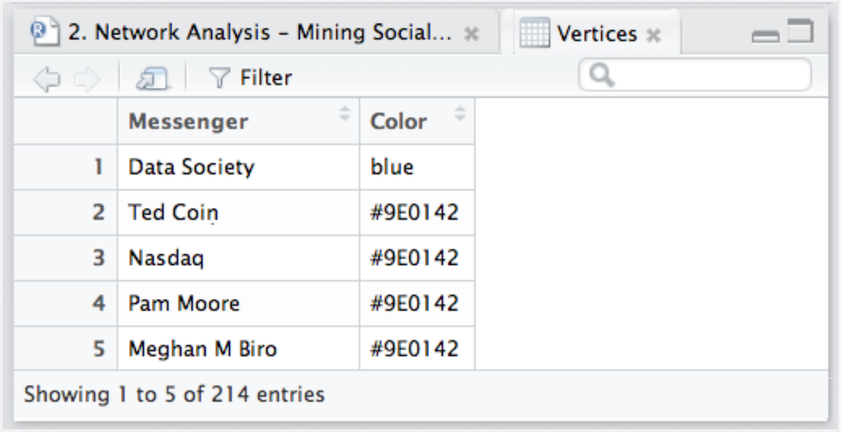
	Messenger	Color
1	Ted Coin	#9E0142
2	Nasdaq	#9E0142
3	Pam Moore	#9E0142
4	Meghan M Biro	#9E0142
5	CrowdTParade	#9E0142

Showing 1 to 6 of 220 entries

Interactive vis: setting up data

Script

```
# Join the Follower_color_clean variable with the
# network_names data set. Remember that inputs must be data frames.
str(network_names)
str(Follow_color_clean)
Vertices = join(as.data.frame(network_names),
                as.data.frame(Follower_color_clean)
View(Vertices)

# Subset the 1st and the 3rd columns.
Vertices = Vertices[, c(1, 3)]

str(Vertices)
Vertices$Messenger = as.character(Vertices$Messenger)
Vertices$Color = as.character(Vertices$Color)

# Insert a color into the first blank spot. Remember, in this network the first
# node is not following anyone.
Vertices$Color[1] = "blue"
View(Vertices)
```



	Messenger	Color
1	Data Society	blue
2	Ted Coin	#9E0142
3	Nasdaq	#9E0142
4	Pam Moore	#9E0142
5	Meghan M Biro	#9E0142

Showing 1 to 5 of 214 entries

Interactive vis: setting up data

Script

```
# forceNetwork can only graph data frames, make sure the data is read as such.
Links_4 = as.data.frame(Links_4)
Vertices = as.data.frame(Vertices)

# Save your data for re-use.
write.csv(Links_4,
          "new_forceNetwork links.csv",
          row.names = FALSE)

write.csv(Vertices,
          "new_forceNetwork vertices.csv",
          row.names = FALSE)
```

Interactive vis: view plot

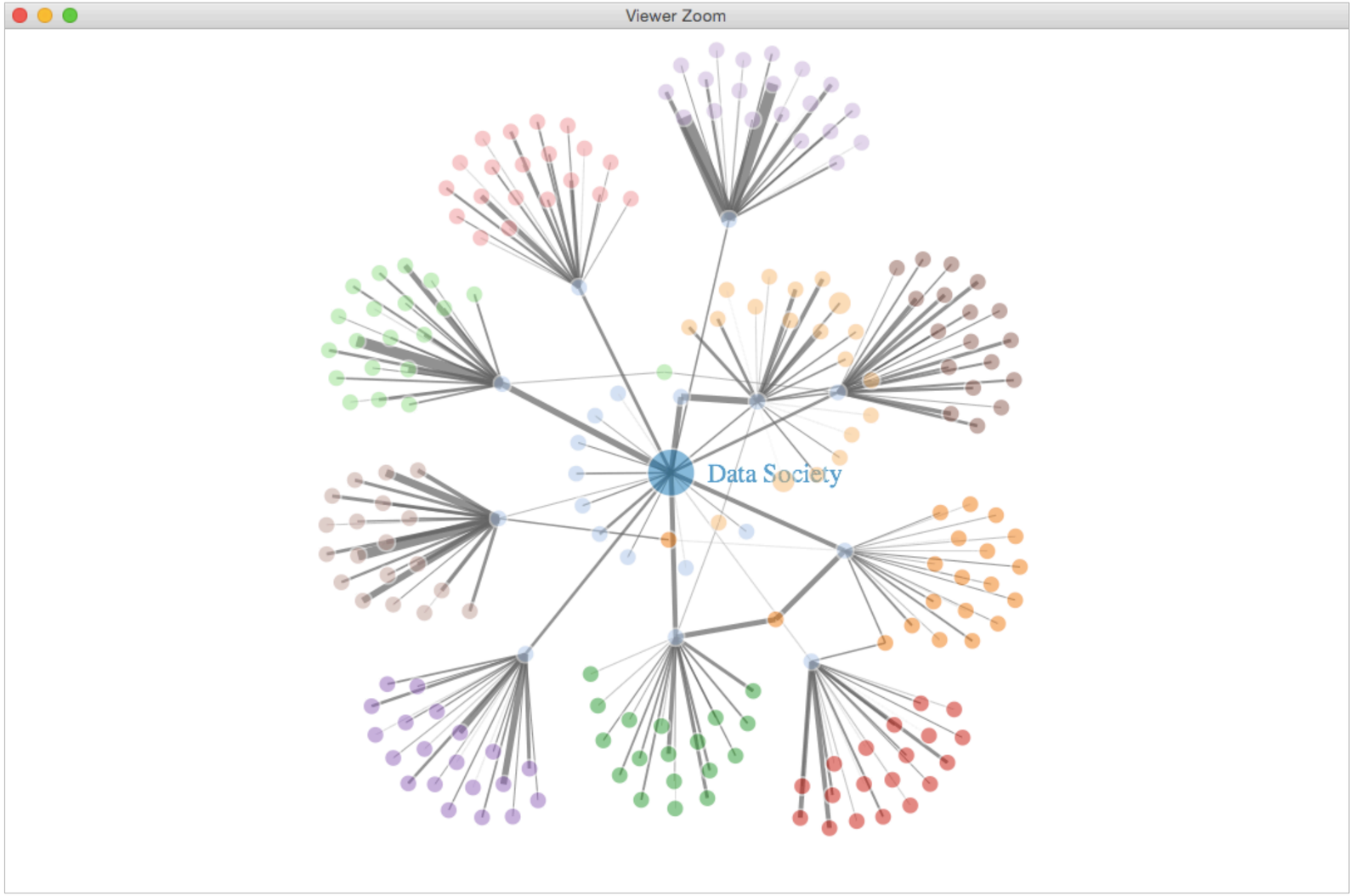
Script

```
# Plot the network graph, remember to use ?forceNetwork to learn about
# additional functionality.
forceNetwork(Links = Links_4,
             Nodes = Vertices,
             Source = "Source",
             Target = "Target",
             Value = "Edge",
             NodeID = "Messenger",
             Group = "Color",
             fontSize = 20,
             opacity = 0.7,
             charge = -120)
```

1. Where the edges start
2. Where the edges go to
3. How thick the edges are
4. Names of the points
5. Colors of the points
6. Font size of the node labels
7. Transparency of the elements in the graph
8. Defines how zoomed in or out the graph is, positive to zoom in, negative to zoom out

```
# The forceNetwork graph can slow down your computer so you can press
# "Clear All" in the viewer window when you're done working with the graph.
```

Interactive vis: view plot



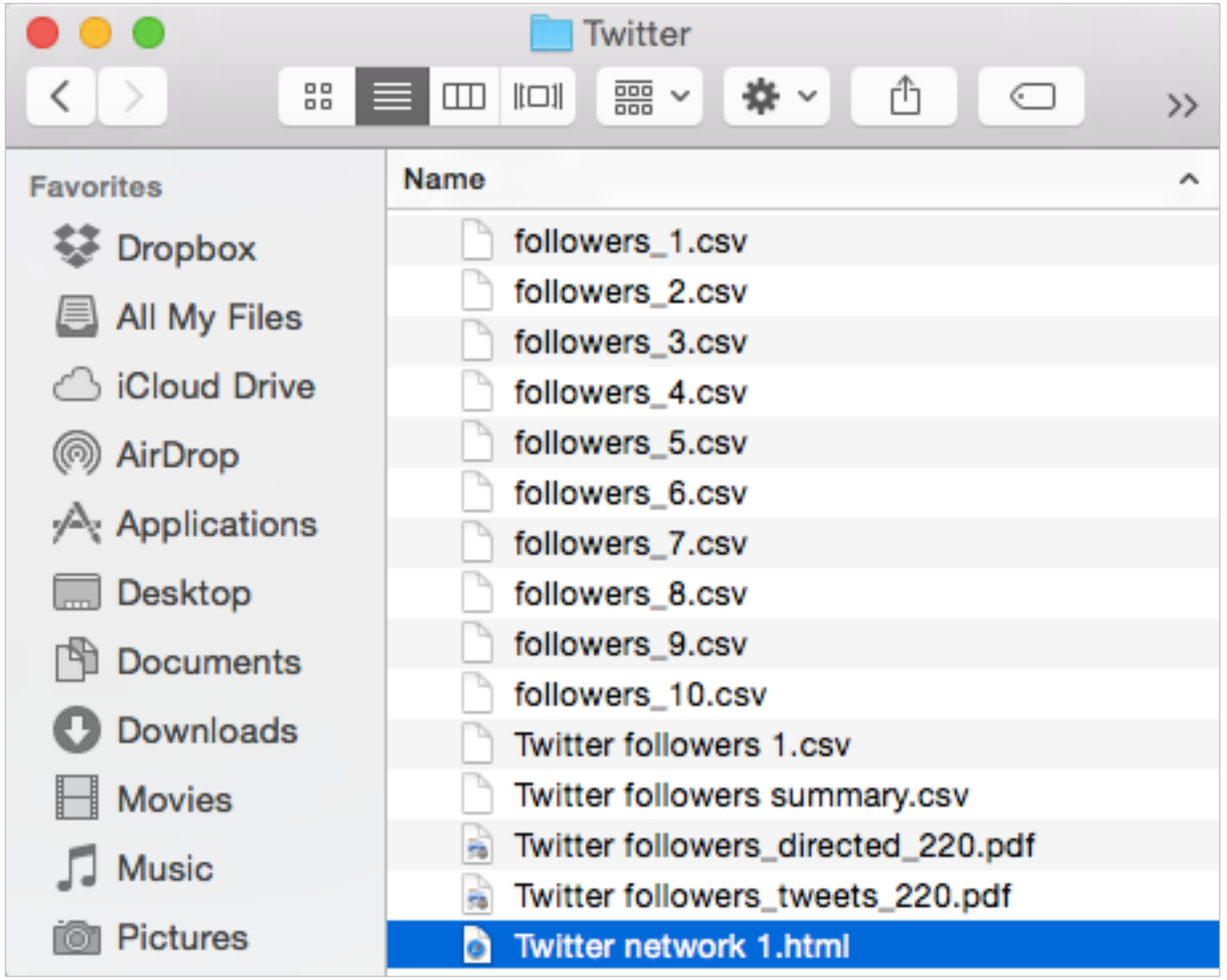
Interactive vis: save plot as html file

Script

```
# Dynamic plots display faster in a web browser window so let's save your
# output as an html file. This will also allow you to share it with others.
install.packages("magrittr")
library(magrittr)
library(help = magrittr)

# The "%>% saveNetwork()" argument will tell R to save the forceNetwork
# file as an html. Make sure magrittr is the last library loaded before you run this.
forceNetwork(Links = Links_4,
             Nodes = Vertices,
             Source = "Source",
             Target = "Target",
             Value = "Edge",
             NodeID = "Messenger",
             Group = "Color",
             fontSize = 20,
             opacity = 0.7,
             charge = -120) %>% saveNetwork(file = "Twitter network 1.html")
```

1. Pass the output into the next function
2. Assign a file name

Interactive vis: save plot as html file



Zoom-enabled plot: setup the data

Script

```
# A visNetwork can be more helpful by zooming into the network and
# getting additional information about the nodes and edges.
install.packages("visNetwork")
library(visNetwork)
library(help = visNetwork)

# First, set up the data frame that will define the node attributes.
# Here we can use the same setup as we did for the prior graph.
# Let's add the number of followers to the Vertices data set that we will use to
# define point size.
View(Vertices)
View(graph_data_sum)

# Rename columns of Vertices so you can combine the data sets
# with the join() function.
colnames(Vertices) = c("Follower", "Color")
Vertices_followers = join(Vertices,
                          graph_data_sum[, 2:3],
                          by = "Follower")
View(Vertices_followers)
```

You can join data sets based on a subset of the data

Zoom-enabled plot: setup the data

2. Network Analysis – Mining Social... | Vertices_followers

Filter

	Follower	Color	Number_Followers
1	Data Society	blue	*NA*
2	Ted Coin	#9E0142	445076
3	Nasdaq	#9E0142	419320
4	Nasdaq	#9E0142	441788
5	Pam Moore	#9E0142	240953
6	Meghan M Biro	#9E0142	106098
7	CrowdTParade	#9E0142	100602
8	Slack	#9E0142	93813
9	Shelly Palmer	#9E0142	89052
10	Sean Ellis	#9E0142	86735

Showing 1 to 11 of 221 entries

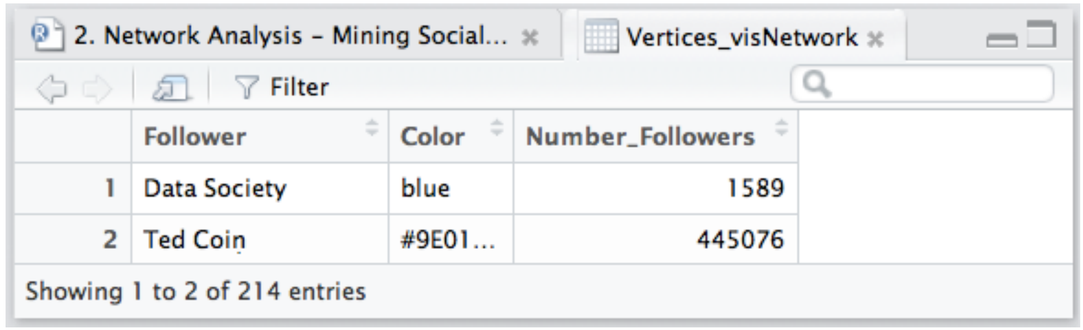
Zoom-enabled plot: setup the data

Script

```
# Add the number of followers for the 1st account.
# Read in the "followers" file if you restarted R since you loaded it.
# The number of rows, if you recall, is the number of followers.
followers = read.csv("Twitter followers 1.csv",
                     check.names = FALSE)
Vertices_followers[1, 3] = nrow(followers)
View(Vertices_followers)

# Remove any duplicated vertices that could have been created if you
# pulled the data at different points in time.
Vertices_visNetwork = subset(Vertices_followers,
                             !duplicated(Vertices_followers[, 1]))
View(Vertices_visNetwork)
```

1. Ensures the column labels load as you have them in the csv file

2. Which data set to subset
3. `duplicated()` determines which elements are duplicated so you're selecting ones that are not duplicated as the "!" means not



	Follower	Color	Number_Followers
1	Data Society	blue	1589
2	Ted Coin	#9E01...	445076

Showing 1 to 2 of 214 entries

Zoom-enabled plot: setup the data

Script

```
TwitterNodes = data.frame(id = Vertices_visNetwork$Follower,
                          label = Vertices_visNetwork$Follower,
                          value = Vertices_visNetwork$Number_Followers/10000,
                          shape = c("circle"),
                          title = paste0("<p><b>",
                                         Vertices_visNetwork$Follower,
                                         "</p></b>"),
                          color = Vertices_visNetwork$Color)

# Next set up the data frame that will define the edges.
# We already have this data.
View(graph_data_sum)

TwitterEdges = data.frame(from = graph_data_sum$Messenger,
                          to = graph_data_sum$Follower,
                          # label = graph_data_sum$Number_Tweets,
                          arrows = c("to"),
                          title = graph_data_sum$Number_Tweets,
                          width = graph_data_sum$Number_Tweets/30000)
```

1. Names of the nodes
2. Labels to appear on the graph
3. Size of the points
4. Different shapes can be assigned to points
5. Label on the pop-up
6. Color of the points

- Scale the size to make the visualization easy to read

- Comment out so it doesn't display for now

7. Labels to appear on the graph
8. Arrows showing direction of connection
9. Label on the pop-up
10. Width of the lines

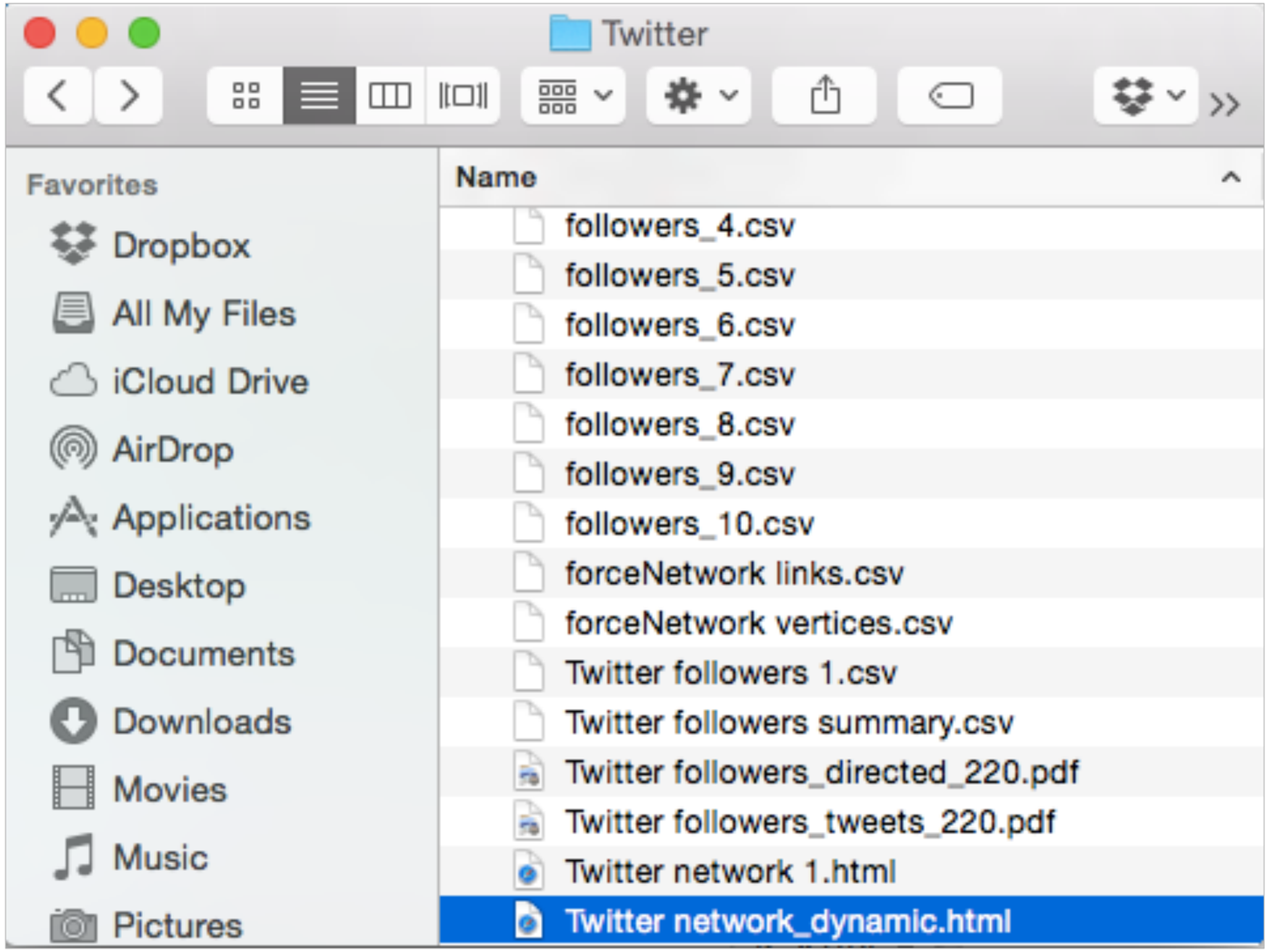
Zoom-enabled plot: create the graph

Script

```
# Assign the visualization to a variable and set display options.
Twitter_dynamic_network = visNetwork(TwitterNodes,
                                     TwitterEdges) %>%
  visOptions(highlightNearest = TRUE,
             nodesIdSelection = TRUE,
             width = 1200,
             height = 800) %>%
  visInteraction(navigationButtons = TRUE) %>%
  visPhysics(maxVelocity = 50)

# Save your visualization as an html file.
htmlwidgets::saveWidget(Twitter_dynamic_network, "Twitter network_dynamic.html",
                        selfcontained = TRUE, libdir = NULL,
                        background = "white")

# You can also look at the memory profile of R using the memory.profile()
# and you can adjust the settings using the options() function.
memory.profile()
options()

# Check the size of the variable you're working with you can use the object.size() function.
object.size(Twitter_dynamic_network)
```

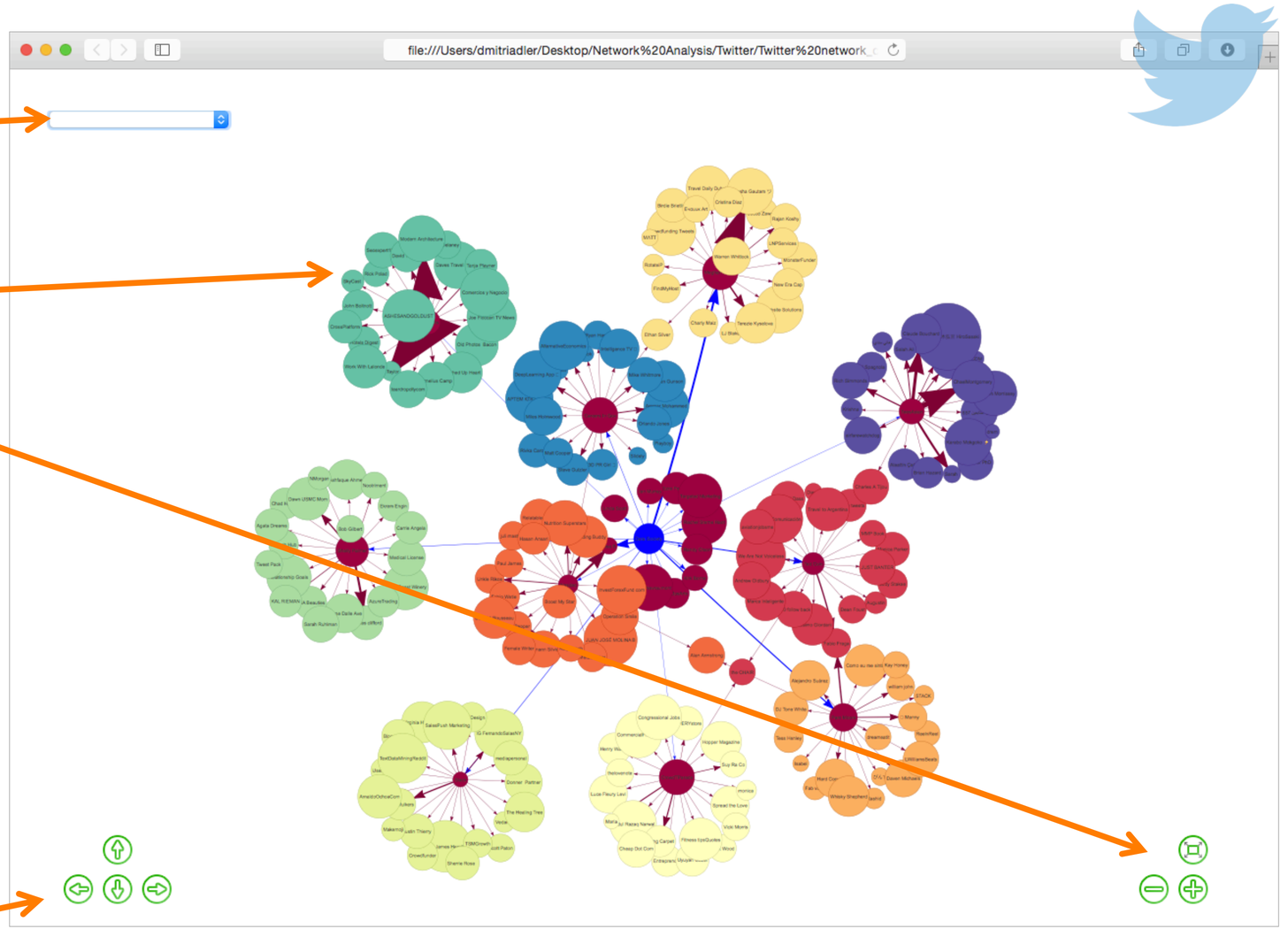
1. Highlight nodes connected to the one you clicked
2. Drop down menu to select particular nodes
3. Width of output
4. Height of output
5. Navigation buttons to help move around the network
6. The physics module limits the maximum velocity of the nodes to decrease the time to stabilization
7. `selfcontained` determines if the HTML will be a self-contained file or a file with external data placed in an adjacent folder
8. If `selfcontained = FALSE` then `libdir` needs to specify the directory where to save the underlying data

Zoom-enabled plot: create the graph



Zoom-enabled plot: create the graph

- Drop-down menu allows you to select nodes
- Point and click functionality
- Smooth zooming functionality
- Edges have pop-up labels
- Drag nodes around for clarity
- Click on the graph and move it around or use these arrows



Let's do some analysis

- Interactive graphs such as this one can tell us a lot just by looking at it

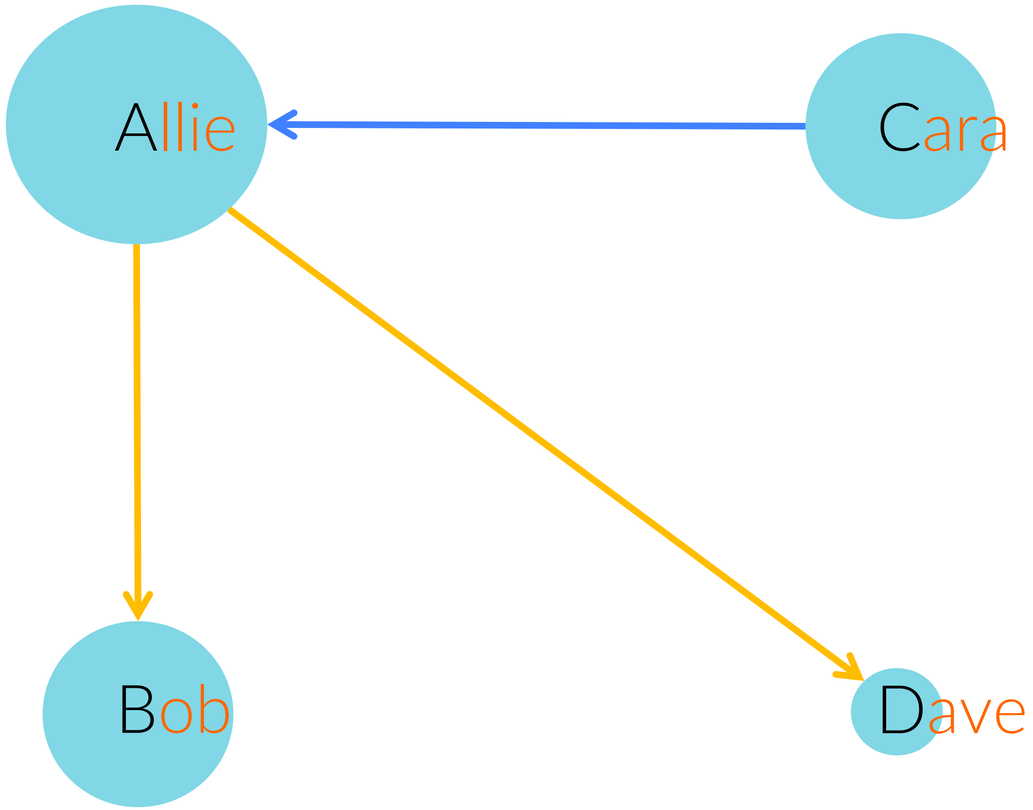
- Challenges arise when the data is large and the network is complex

- We can pull data on 900,000 Twitter users and use it to figure out:
  1. Who drives opinion
  2. Who are the key connectors in the network
  3. Who are drones or information collectors
  4. Who people trust
  5. How information spreads

Look at the add-on module to learn how to collect data and manipulate it into the proper format.

Measuring a directed network: counting

- The in-degree is the number of edges going to a node
- The out-degree is the number of edges going out of a node
- Allie has an in-degree of 1 and an out-degree of 2

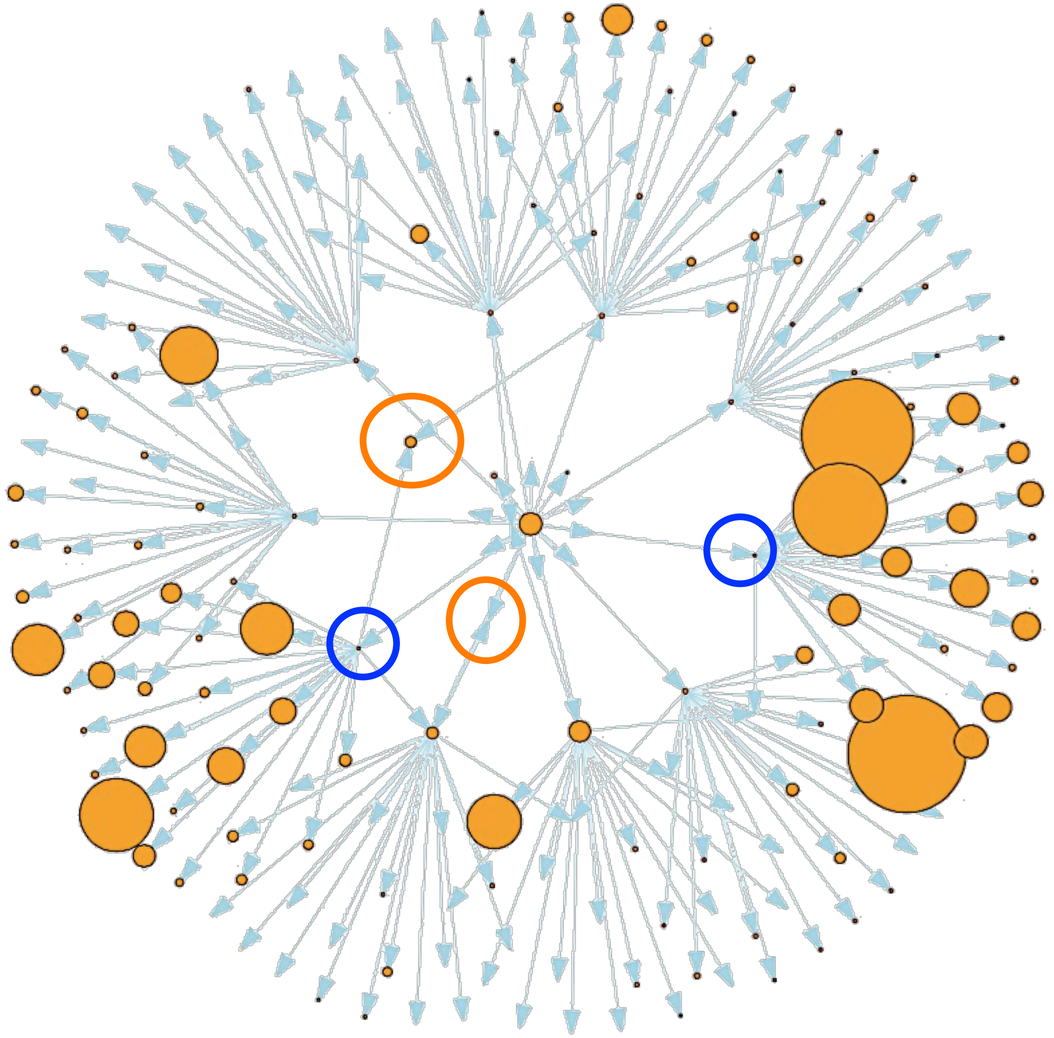


- *Question to answer:*
  - *Which way does communication or production flow?*

Twitter: directed degree centrality

- In a communication network, what does it mean when someone "listens" or "follows" others but has no followers or outgoing communication?
  - Could be a bot
  - Could be a spy
  - Could be a fraudulent account

- What does it mean when someone communicates a lot or has a lot of followers but has no incoming messages and follows few others?
  - Could be an opinion leader
  - A celebrity



Direction of degree: other applications

- How do the elements of a human cell interact, which ones are key?
- What does an economic system rely on?

- How do you optimize your supplier network?
- How do you protect your supply chain system from failure?

- Which members of an organization drive culture and opinion?
- Determine concentration of risk in the production process

- Optimize routing
- Optimize distribution of goods in your warehouses

- Distinguish between opinion leaders and followers
- Who has a reliable customer base? How can you extend it?

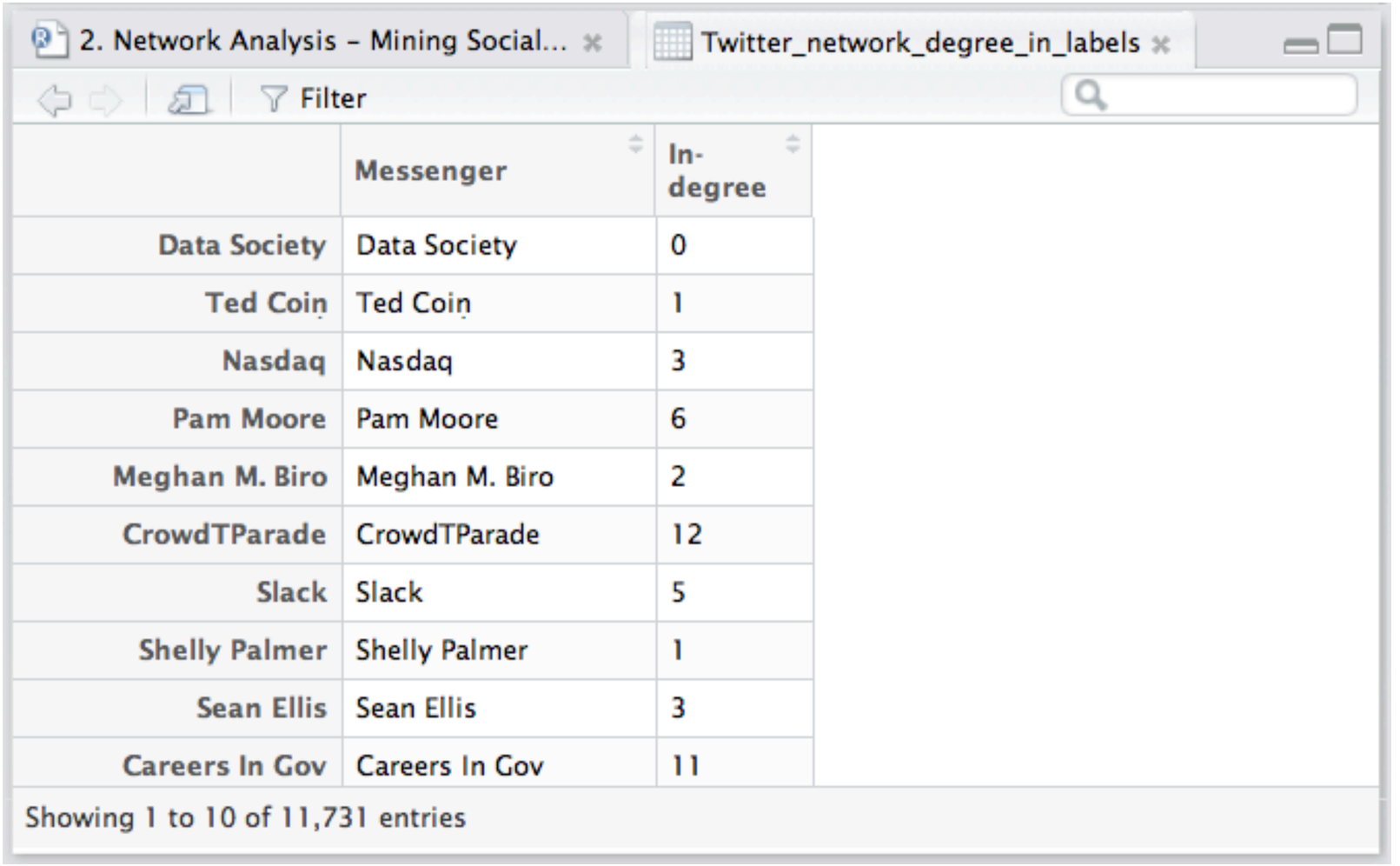
Twitter: directed degree centrality

Script

```
# Read in the clean data frame you saved or the one that we provided.
Twitter_network_clean = read.csv("Twitter network data.csv",
                                 check.names = FALSE)

# Let's start with in-degree centrality.
# First, convert the data into a graph object.
Twitter_network_clean_directed = graph.data.frame(Twitter_network_clean,
                                                  directed = TRUE)

Twitter_network_degree_in = degree(Twitter_network_clean_directed,
                                   mode = "in")

# Add the row names as a column to complete the data set.
Twitter_network_degree_in_names = rownames(as.data.frame(Twitter_network_degree_in))
Twitter_network_degree_in_labels = cbind(Twitter_network_degree_in_names,
                                         Twitter_network_degree_in)

# Rename columns.
colnames(Twitter_network_degree_in_labels) = c("Messenger", "In-degree")
View(Twitter_network_degree_in_labels)
```

Determines the type of degree centrality measured

Twitter: directed degree centrality



	Messenger	In-degree
Data Society	Data Society	0
Ted Coin	Ted Coin	1
Nasdaq	Nasdaq	3
Pam Moore	Pam Moore	6
Meghan M. Biro	Meghan M. Biro	2
CrowdTParade	CrowdTParade	12
Slack	Slack	5
Shelly Palmer	Shelly Palmer	1
Sean Ellis	Sean Ellis	3
Careers In Gov	Careers In Gov	11

Showing 1 to 10 of 11,731 entries

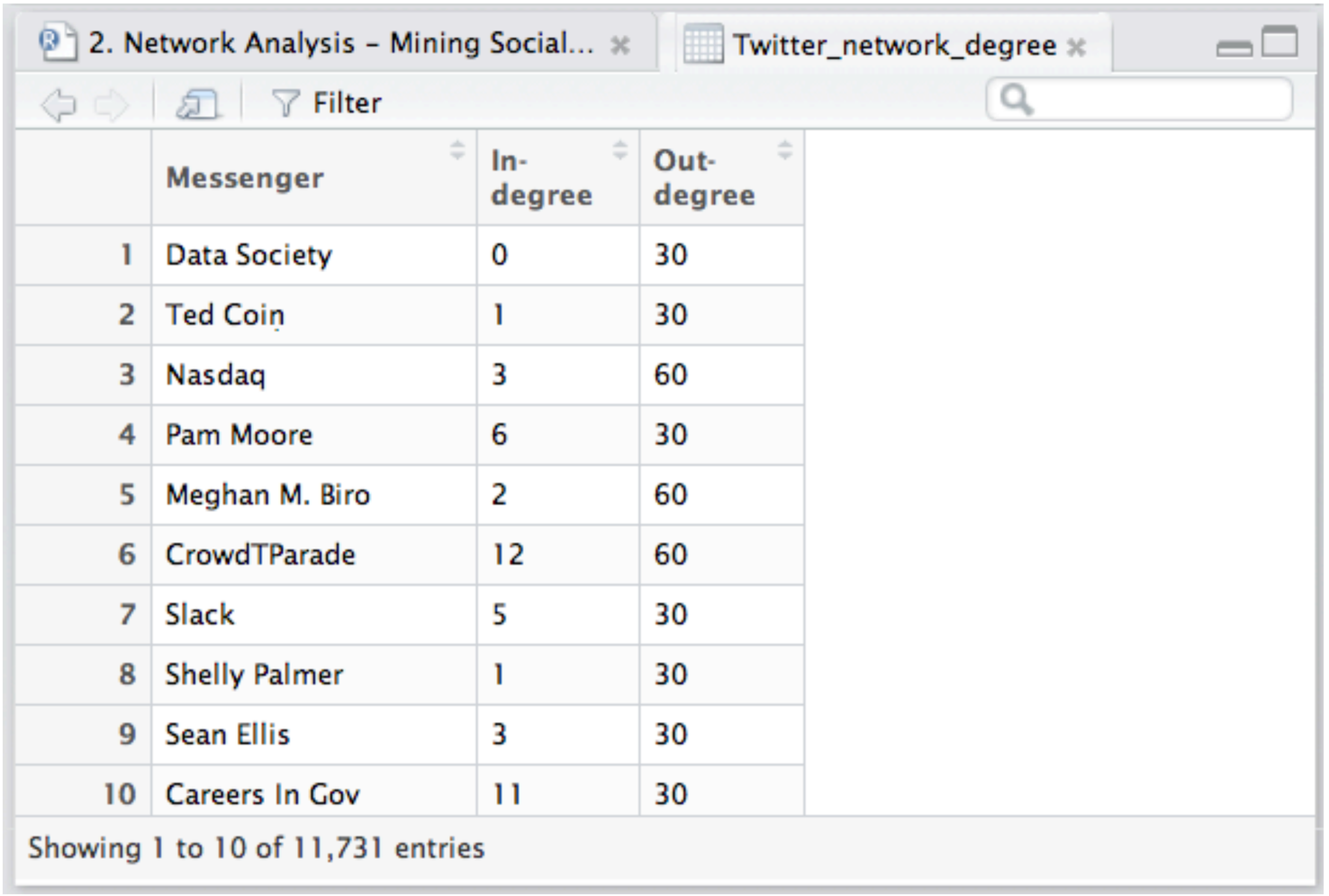
Twitter: directed degree centrality

Script

```
# Repeat these steps for the out-degree.
Twitter_network_degree_out = degree(Twitter_network_clean_directed,
                                    mode = "out")

# Add the row names as a column to complete the data set.
Twitter_network_degree_out_names = rownames(as.data.frame(Twitter_network_degree_out))
Twitter_network_degree_out_labels = cbind(Twitter_network_degree_out_names,
                                          Twitter_network_degree_out )

# Rename columns.
colnames(Twitter_network_degree_out_labels) = c("Messenger", "Out-degree")
View(Twitter_network_degree_out_labels)

# Join the 2 degree centrality data sets.
library(plyr)
Twitter_network_degree = join(as.data.frame(Twitter_network_degree_in_labels),
                              as.data.frame(Twitter_network_degree_out_labels))
View(Twitter_network_degree)
```

Determines the type of degree centrality measured

Twitter: directed degree centrality



	Messenger	In-degree	Out-degree
1	Data Society	0	30
2	Ted Coin	1	30
3	Nasdaq	3	60
4	Pam Moore	6	30
5	Meghan M. Biro	2	60
6	CrowdTParade	12	60
7	Slack	5	30
8	Shelly Palmer	1	30
9	Sean Ellis	3	30
10	Careers In Gov	11	30

Showing 1 to 10 of 11,731 entries

Add followers and friends data

- We just calculated the in- and out-degree of each Twitter account within the limits of the network that we generated

- The actual in- and out-degrees on Twitter are represented by total followers and total friends (the people that are followed)

- Let's add those 2 columns for completeness

Add followers and friends data

Script

```
# The summary file we created only has attributes related
# to the "Follower" column and not the "Messenger" column so you'll
# need to pull the data for the original or first account separately.
DSco = getUser("@datasocietyco")

DSco_followers = DSco$getFollowersCount()

DSco_friends = DSco$getFriendsCount()

DSco_data = cbind(as.character(Twitter_network_clean[1, 1]),
                  DSco_followers,
                  DSco_friends)

# In order to combine the data sets, make sure the column names and data types
# are the same.
Twitter_network_data = Twitter_network_clean[, 2:4]
View(Twitter_network_data)

colnames(Twitter_network_data)
colnames(DSco_data) = c("Follower", "Number_Followers", "Number_Friends")
```

Note the data you pull for this account will be different because Twitter information changes over time!

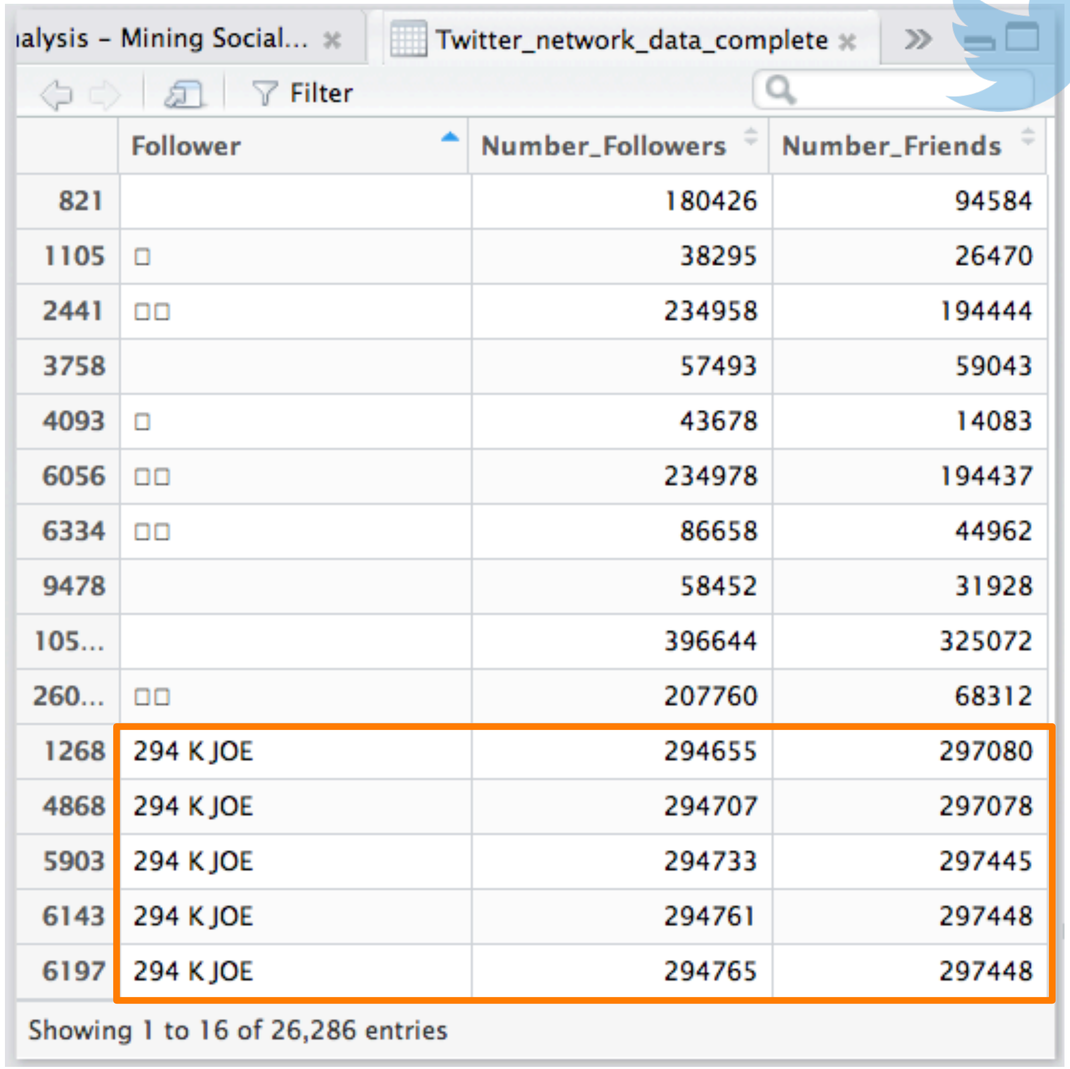
Add followers and friends data

Script

```
# To convert data types to the format you need make sure
# the data is read as a data frame.
DSco_data = as.data.frame(DSco_data)

# Check the structure of the 2 data sets you need to merge.
str(DSco_data)
str(Twitter_network_data)

# Make sure R reads numbers as numbers and not factors.
# To convert factors into numbers you first need to convert the numbers to characters.
DSco_data$Number_Followers = as.integer(as.character(DSco_data[1, 2]))
DSco_data$Number_Friends = as.integer(as.character(DSco_data[1, 3]))

DSco_data[, 1] = as.character(DSco_data[, 1])
Twitter_network_data$Follower = as.character(Twitter_network_data$Follower)

Twitter_network_data_complete = rbind(DSco_data, Twitter_network_data)
View(Twitter_network_data_complete)
```

Add followers and friends data

- You can probably see that some rows are repeated and some users have several entries with varying numbers of friends and followers (sort the data by the first column to see that)
- This is the challenge with real time data - if you downloaded all the data over time then the variance in users' accounts will show up
- To avoid this you can either create another loop to pull data for several thousand accounts, which would be time consuming or just take the last instance of each record we pulled



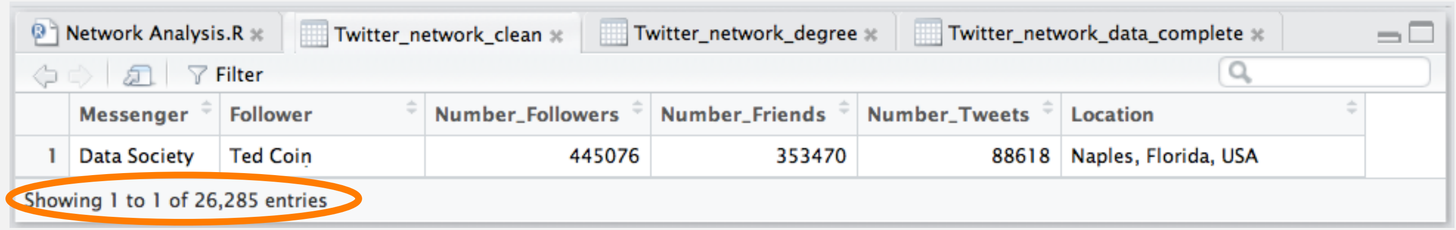
	Follower	Number_Followers	Number_Friends
821		180426	94584
1105	□	38295	26470
2441	□□	234958	194444
3758		57493	59043
4093	□	43678	14083
6056	□□	234978	194437
6334	□□	86658	44962
9478		58452	31928
105...		396644	325072
260...	□□	207760	68312
1268	294 K JOE	294655	297080
4868	294 K JOE	294707	297078
5903	294 K JOE	294733	297445
6143	294 K JOE	294761	297448
6197	294 K JOE	294765	297448

Showing 1 to 16 of 26,286 entries

Add followers and friends data

Script

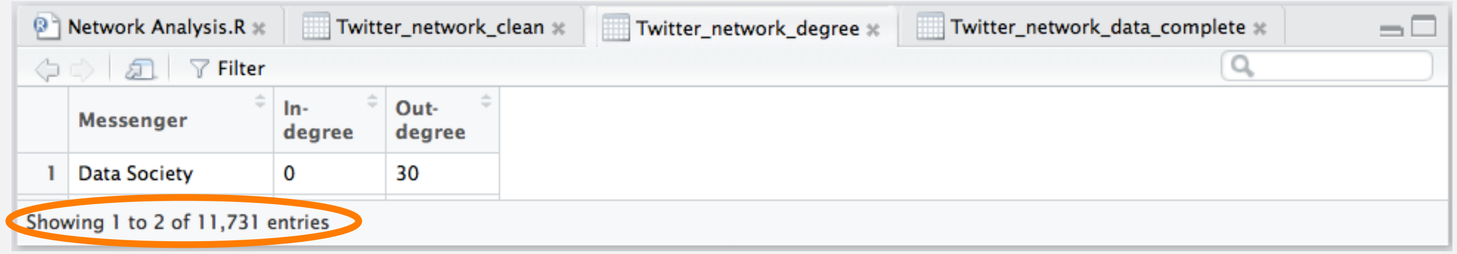
```
# You started with this data set.
View(Twitter_network_clean)
```



	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location
1	Data Society	Ted Coin	445076	353470	88618	Naples, Florida, USA

Showing 1 to 1 of 26,285 entries

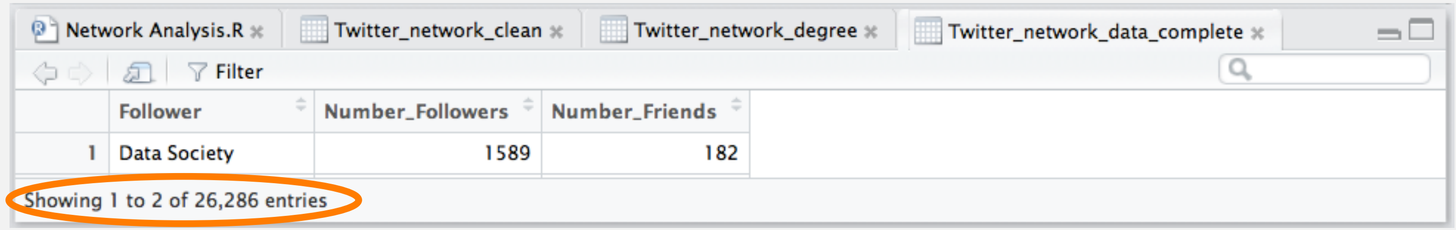
```
# Then created this data set.
View(Twitter_network_degree)
```



	Messenger	In-degree	Out-degree
1	Data Society	0	30

Showing 1 to 2 of 11,731 entries

```
# And added the original messenger to create this data set.
View(Twitter_network_data_complete)
```



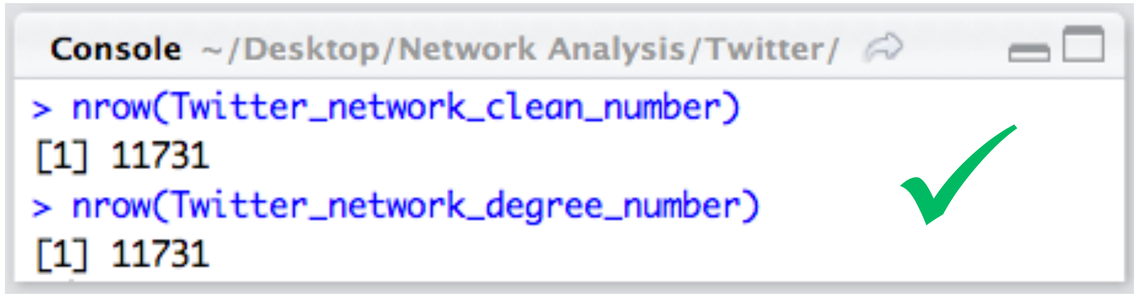
	Follower	Number_Followers	Number_Friends
1	Data Society	1589	182

Showing 1 to 2 of 26,286 entries

Add followers and friends data

Script

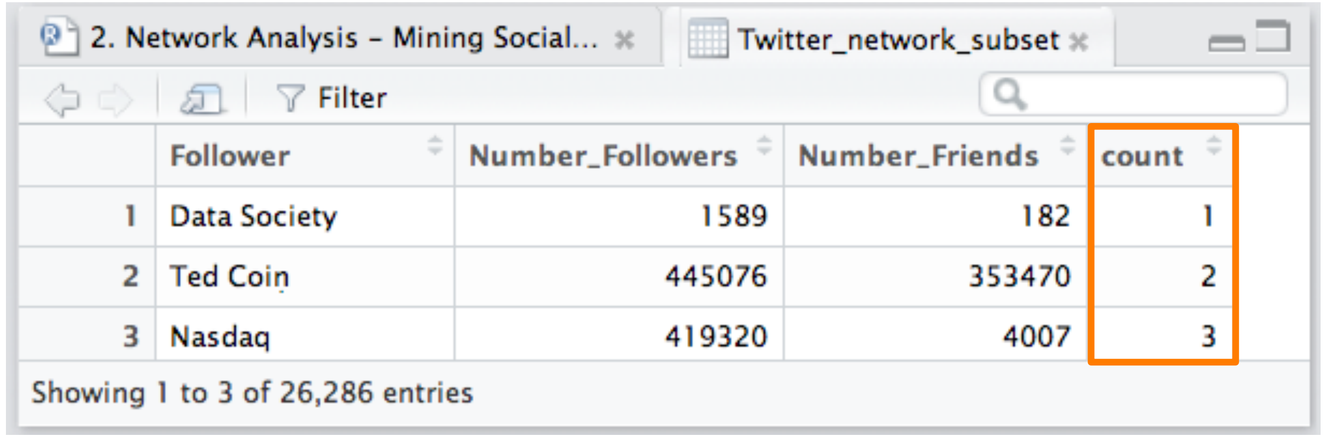
```
# Now confirm that you're working with the same number of users between
# all data sets. Being careful and confirming you are not making any mistakes
# along the way is very important when building analyses.
Twitter_network_clean_number = c(as.character(Twitter_network_clean$Messenger),
                                 as.character(Twitter_network_clean$Follower))
str(Twitter_network_clean_number)
Twitter_network_clean_number = unique(as.data.frame(Twitter_network_clean_number))

Twitter_network_degree_number = as.character(Twitter_network_degree$Messenger)
str(Twitter_network_degree_number)
Twitter_network_degree_number = unique(as.data.frame(Twitter_network_degree_number))

nrow(Twitter_network_clean_number)
nrow(Twitter_network_degree_number)
```



Add followers and friends data

Script

```
# For every user, we'll pick the latest values we pulled.
str(Twitter_network_data_complete)

# Add a column that will include the row number.
Twitter_network_subset = cbind(Twitter_network_data_complete,
                               seq(1:nrow(Twitter_network_data_complete)))

colnames(Twitter_network_subset) = c("Follower", "Number_Followers",
                                     "Number_Friends", "count")

View(Twitter_network_subset)
```



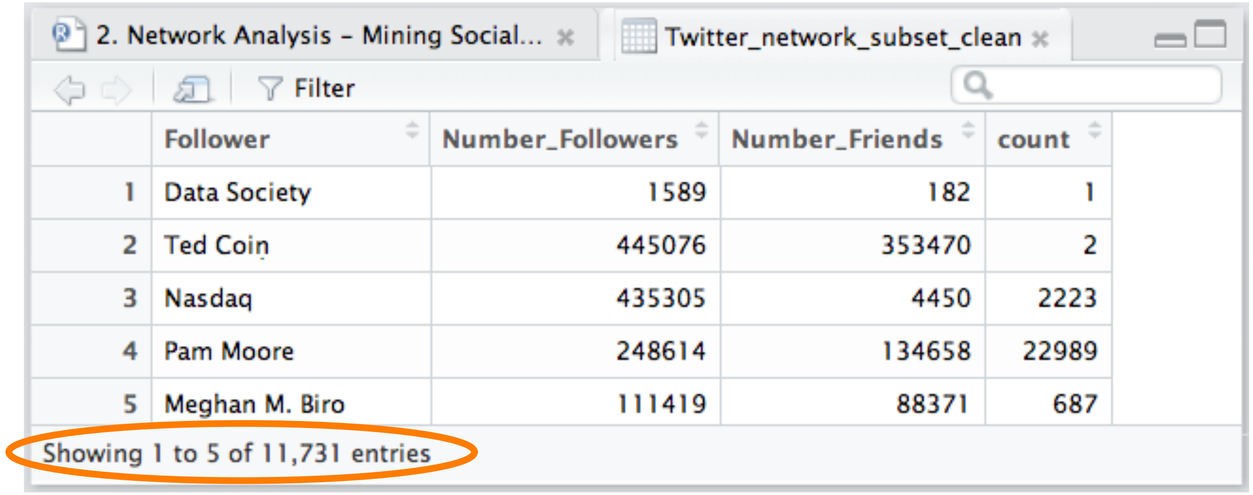
	Follower	Number_Followers	Number_Friends	count
1	Data Society	1589	182	1
2	Ted Coin	445076	353470	2
3	Nasdaq	419320	4007	3

Showing 1 to 3 of 26,286 entries

Add followers and friends data

Script

```
# We will use the ".SD" notation, which stands for "subset data table", to tell R
# to use the maximum of the "count" column to subset the data by the "Follower"
# column. The maximum "count" will reflect the latest record we pulled.
library(data.table)
Twitter_network_subset_clean = as.data.table(Twitter_network_subset)[,
                                                                     .SD[max(count) ==
                                                                           count],
                                                                     Follower]


nrow(Twitter_network_degree_number)
View(Twitter_network_subset_clean)
```



	Follower	Number_Followers	Number_Friends	count
1	Data Society	1589	182	1
2	Ted Coin	445076	353470	2
3	Nasdaq	435305	4450	2223
4	Pam Moore	248614	134658	22989
5	Meghan M. Biro	111419	88371	687

Showing 1 to 5 of 11,731 entries

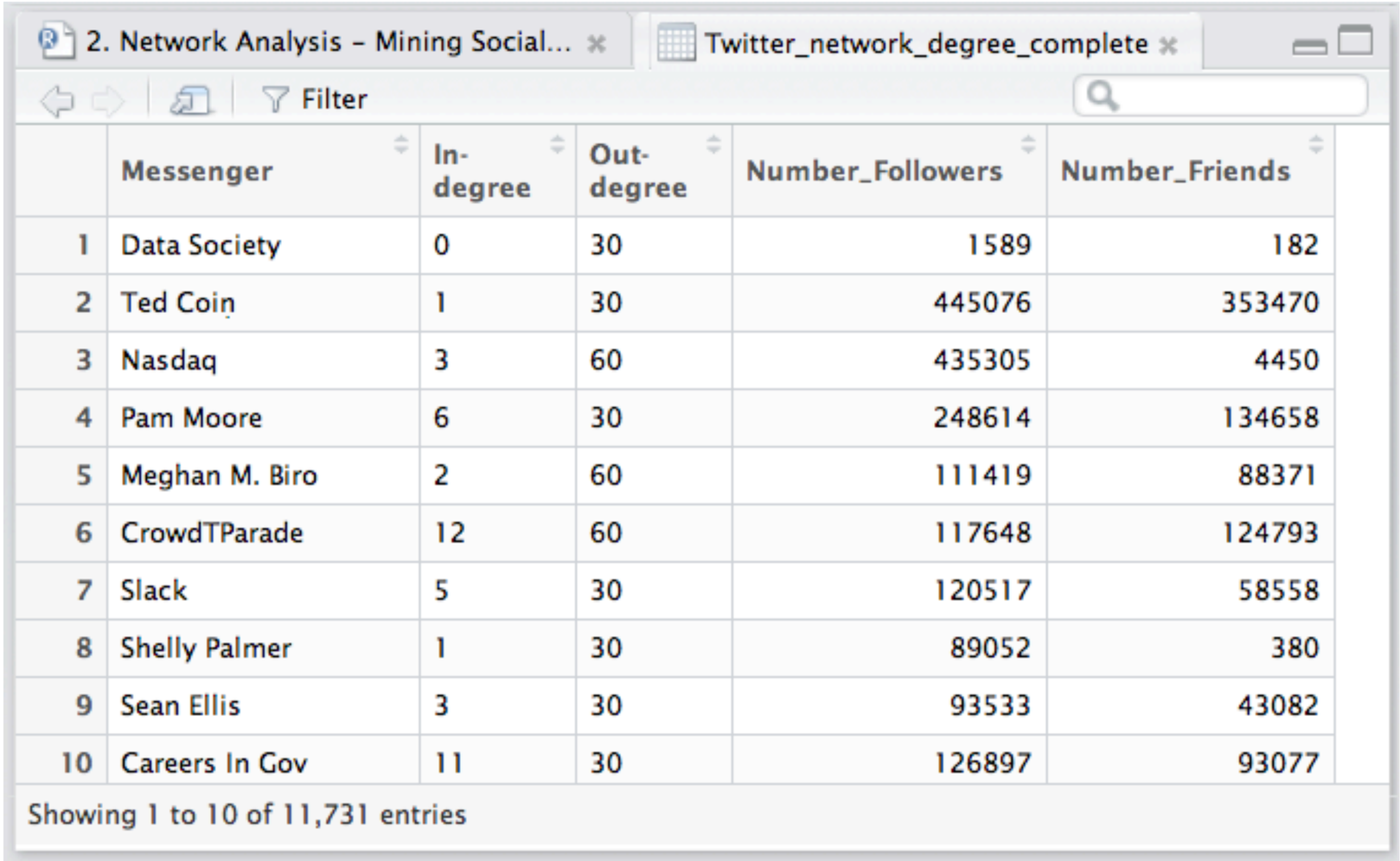
1. The comma means that the operation is performed on columns
2. "`.SD`" stands for "subset data table"
3. The subset condition is set inside the brackets, here it's the max of the column "count"
4. The last argument tells R which column to subset by

Add followers and friends data

Script

```
# Now you can join the clean data set you just created to the data
# set containing the degree counts for each Twitter account.

# Let's make sure the column we want to use to join the data
# sets has the same label, you can use the setnames() function to do that.
colnames(Twitter_network_degree)
colnames(Twitter_network_subset_clean)

library(data.table)
setnames(Twitter_network_subset_clean, old = c("Follower"), new = c("Messenger"))

# Make sure your data is read as a data frame before using join().
Twitter_network_degree = as.data.frame(Twitter_network_degree)
Twitter_network_subset_clean = as.data.frame(Twitter_network_subset_clean)
Twitter_network_degree_complete = join(Twitter_network_degree,
                                       Twitter_network_subset_clean[, 1:3])
View(Twitter_network_degree_complete)

# Save your work.
write.csv(Twitter_network_degree_complete,
          "new_Twitter degree data.csv", row.names = FALSE)
```

Add followers and friends data



2. Network Analysis – Mining Social... | Twitter_network_degree_complete

Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends
1	Data Society	0	30	1589	182
2	Ted Coin	1	30	445076	353470
3	Nasdaq	3	60	435305	4450
4	Pam Moore	6	30	248614	134658
5	Meghan M. Biro	2	60	111419	88371
6	CrowdTParade	12	60	117648	124793
7	Slack	5	30	120517	58558
8	Shelly Palmer	1	30	89052	380
9	Sean Ellis	3	30	93533	43082
10	Careers In Gov	11	30	126897	93077

Showing 1 to 10 of 11,731 entries

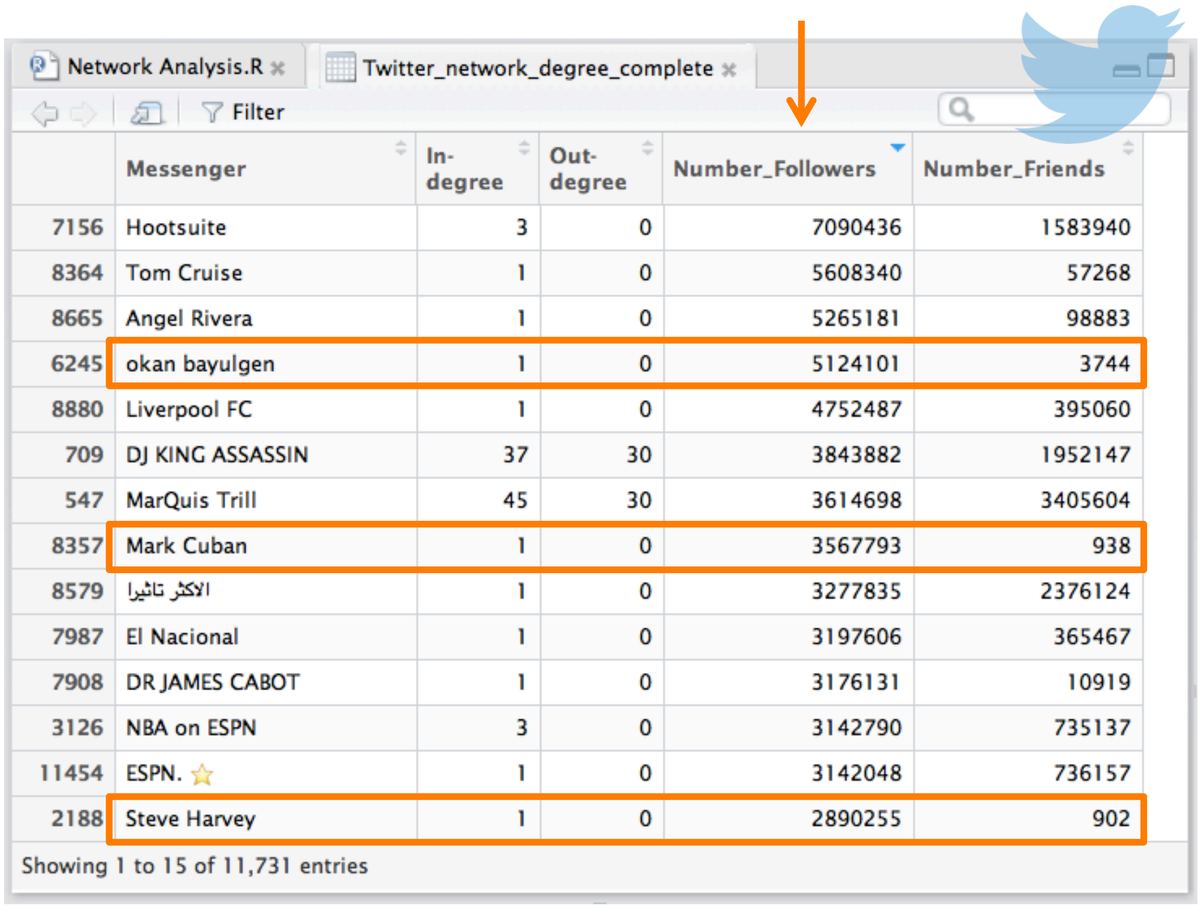
Analyze degree centrality

Script

```
# First, make sure that R is reading your data correctly.
str(Twitter_network_degree_complete)
Twitter_network_degree_complete$Messenger =
  as.character(Twitter_network_degree_complete$Messenger)

Twitter_network_degree_complete$`In-degree` =
  as.integer(as.character(Twitter_network_degree_complete$`In-degree`))

Twitter_network_degree_complete$`Out-degree` =
  as.integer(as.character(Twitter_network_degree_complete$`Out-degree`))


# Now you can sort the data to get insights about the structure
# of this network.

View(Twitter_network_degree_complete)
```

Analyze degree centrality: global

- Sort the data based on the number of followers
- Note that 3 accounts stand out:
  - Okan Bayülgen is a Turkish actor, variety and talk show host and comedian
  - Mark Cuban is an American entrepreneur and owner of the Dallas Mavericks basketball team
  - Steve Harvey is an American comedian, television host, radio personality, actor, and author



	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends
7156	Hootsuite	3	0	7090436	1583940
8364	Tom Cruise	1	0	5608340	57268
8665	Angel Rivera	1	0	5265181	98883
6245	okan bayulgen	1	0	5124101	3744
8880	Liverpool FC	1	0	4752487	395060
709	DJ KING ASSASSIN	37	30	3843882	1952147
547	MarQuis Trill	45	30	3614698	3405604
8357	Mark Cuban	1	0	3567793	938
8579	الاكثر تاثيرا	1	0	3277835	2376124
7987	El Nacional	1	0	3197606	365467
7908	DR JAMES CABOT	1	0	3176131	10919
3126	NBA on ESPN	3	0	3142790	735137
11454	ESPN. ☆	1	0	3142048	736157
2188	Steve Harvey	1	0	2890255	902

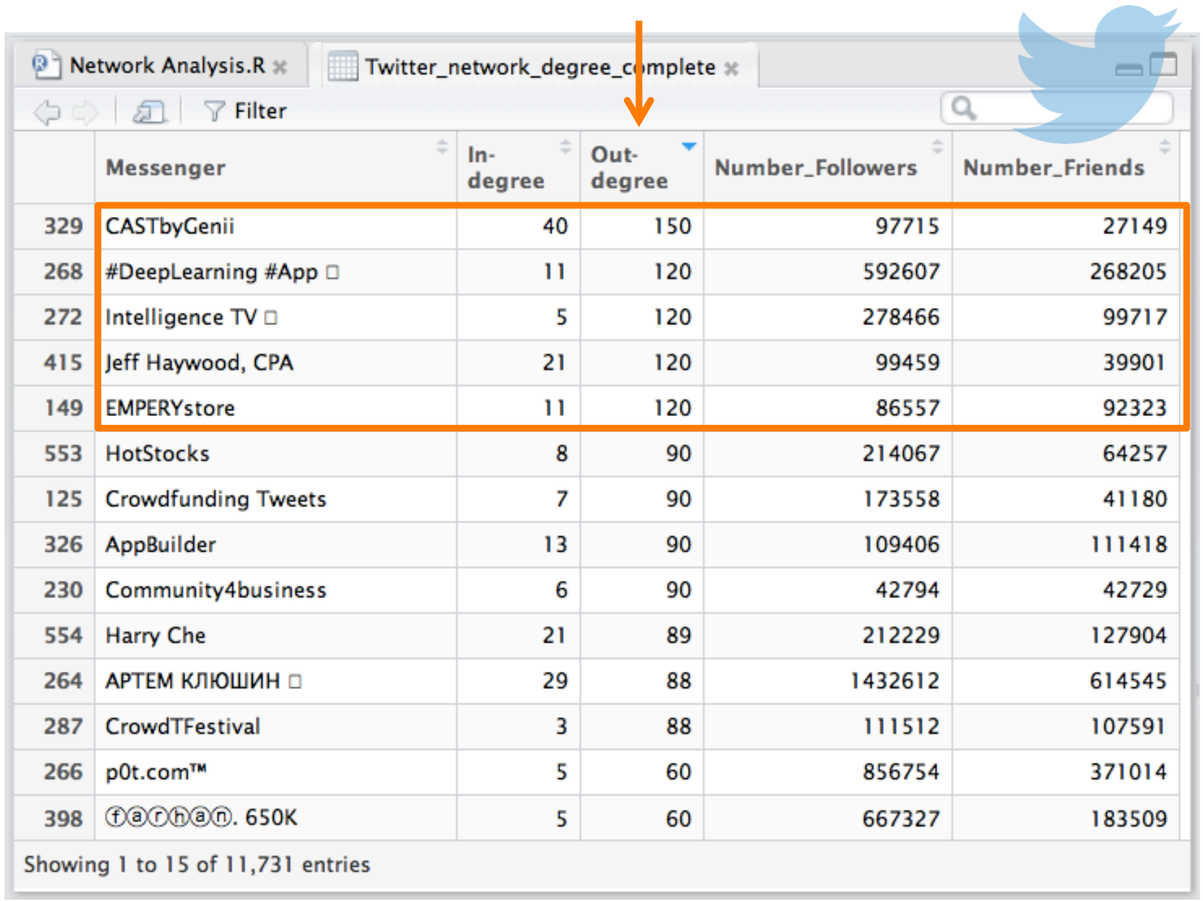
Showing 1 to 15 of 11,731 entries

These amazing people are no more than 3 Twitter degrees of separation from our original account!

How could this knowledge impact your Twitter strategy for spreading your message?

Analyze degree centrality: this network

- Sort the data based on out-degree
- This tells us which people are most followed within our 11.7k person network
- If you want to reach this group, these are the people that would be most important



	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends
329	CASTbyGenii	40	150	97715	27149
268	#DeepLearning #App □	11	120	592607	268205
272	Intelligence TV □	5	120	278466	99717
415	Jeff Haywood, CPA	21	120	99459	39901
149	EMPERYstore	11	120	86557	92323
553	HotStocks	8	90	214067	64257
125	Crowdfunding Tweets	7	90	173558	41180
326	AppBuilder	13	90	109406	111418
230	Community4business	6	90	42794	42729
554	Harry Che	21	89	212229	127904
264	АРТЕМ КЛЮШИН □	29	88	1432612	614545
287	CrowdTFestival	3	88	111512	107591
266	p0t.com™	5	60	856754	371014
398	ⓕⓐⓡⓗⓐⓝ. 650K	5	60	667327	183509

Showing 1 to 15 of 11,731 entries

Recall that we downloaded data on almost 900,000 accounts.
Try this analysis with all that data!

Analyze degree centrality: this network

- Sort the data based on in-degree
- This tells us which people follow the most people within our 11.7k person network
- These are individuals who may be most interested in this particular group
- We may want to learn more about these people to understand who our primary audience is!

Network Analysis.R | Twitter_network_degree_complete

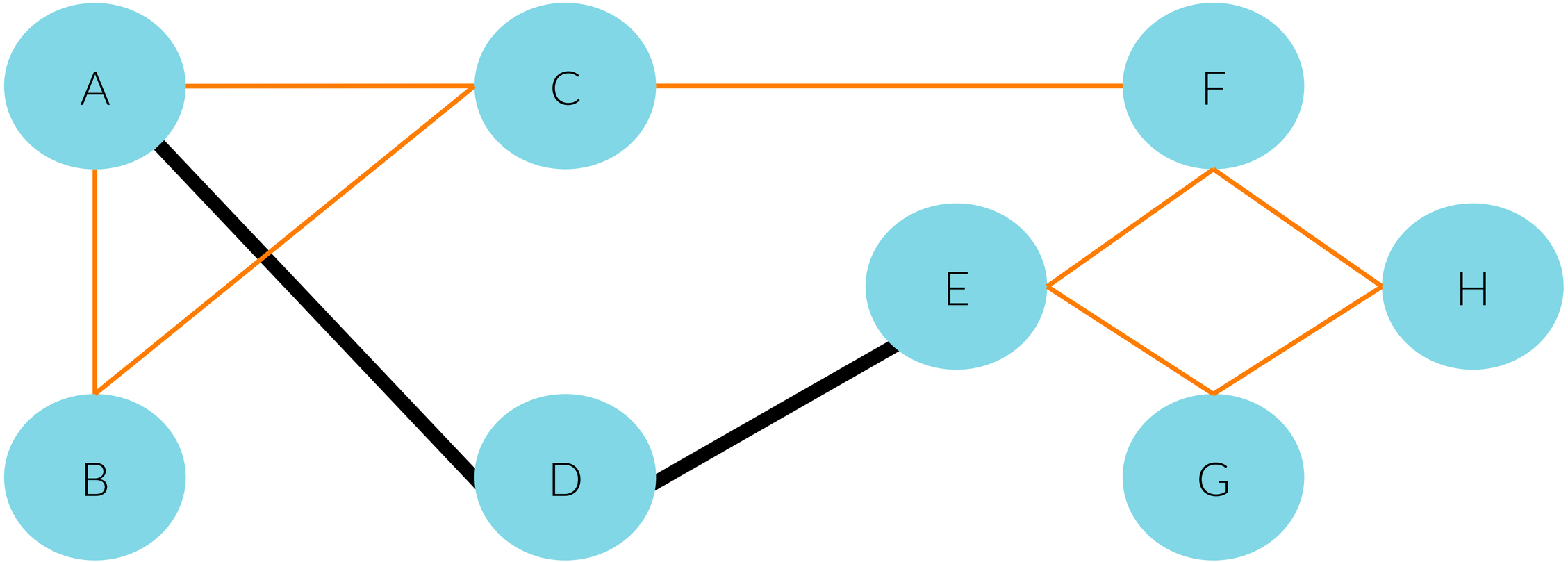
Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends
360	Start-Ups.Co	63	30	100498	104875
890	Wellbelove	49	0	356599	259655
1261	Mike Whitaker	48	0	425968	121023
374	Nicholas Hill	47	60	397985	94235
384	Kerry Butters	46	30	184156	178193
547	MarQuis Trill	45	30	3614698	3405604
323	Stewart Harding	45	30	681071	748980
812	JSWX	44	0	222042	242855
1315	Joshua Davidson	43	0	706818	240086
987	joshingstern	42	0	880551	681867
688	Jeff Bullas	42	30	358654	220066
1612	·(M G W V)·	40	0	106847	105046
329	CASTbyGenii	40	150	97715	27149
313	Shaun Frankson	38	30	296733	85664

Showing 1 to 15 of 11,731 entries

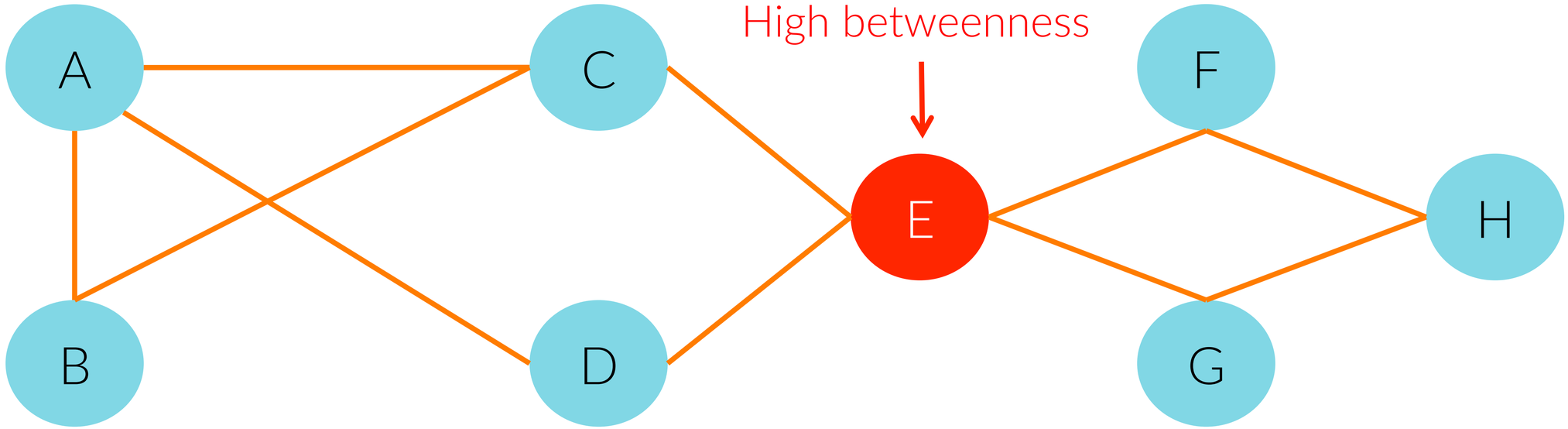
Measuring a network: shortest path

- Shortest path between A and E: 2



Betweenness centrality

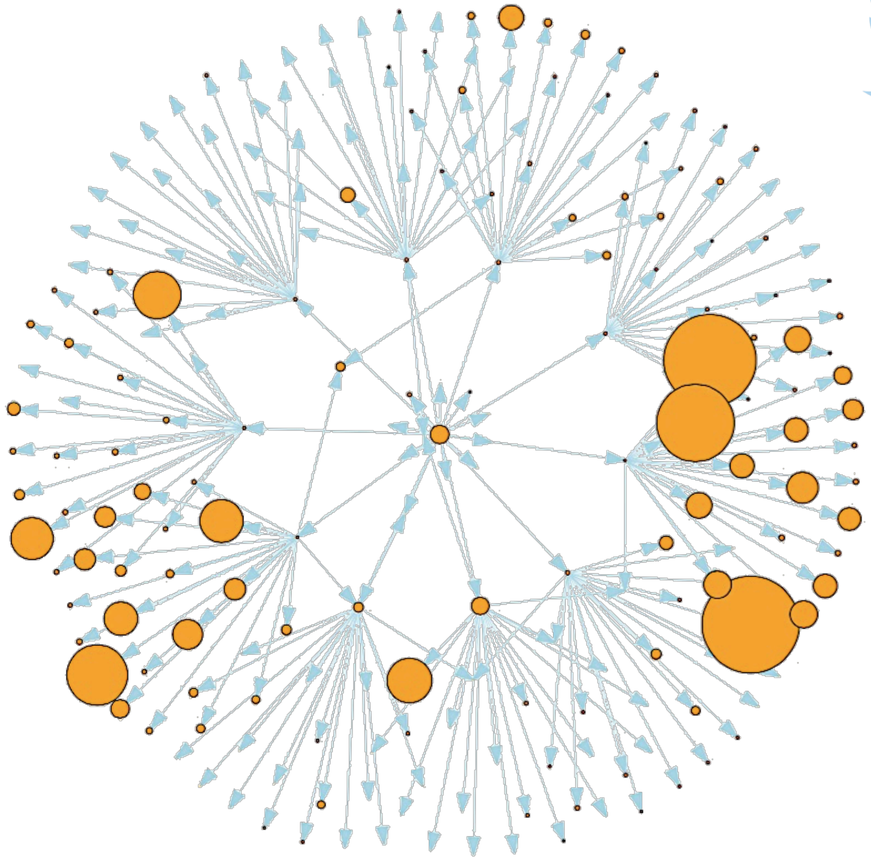
- Betweenness centrality – % of shortest paths in a network that include a given node
  - This metrics allows you to assess the extent to which someone is a prominent connector in a network



Betweenness: directed network

- Direction of of the relationship tells you the significance of betweenness:
  - Someone can be heavily followed by people and serve as an opinion leader
  - Someone can have few followers but follow many other people, this is a reader and potentially an automated bot

Spreading a message:
not all followers are equally valuable!

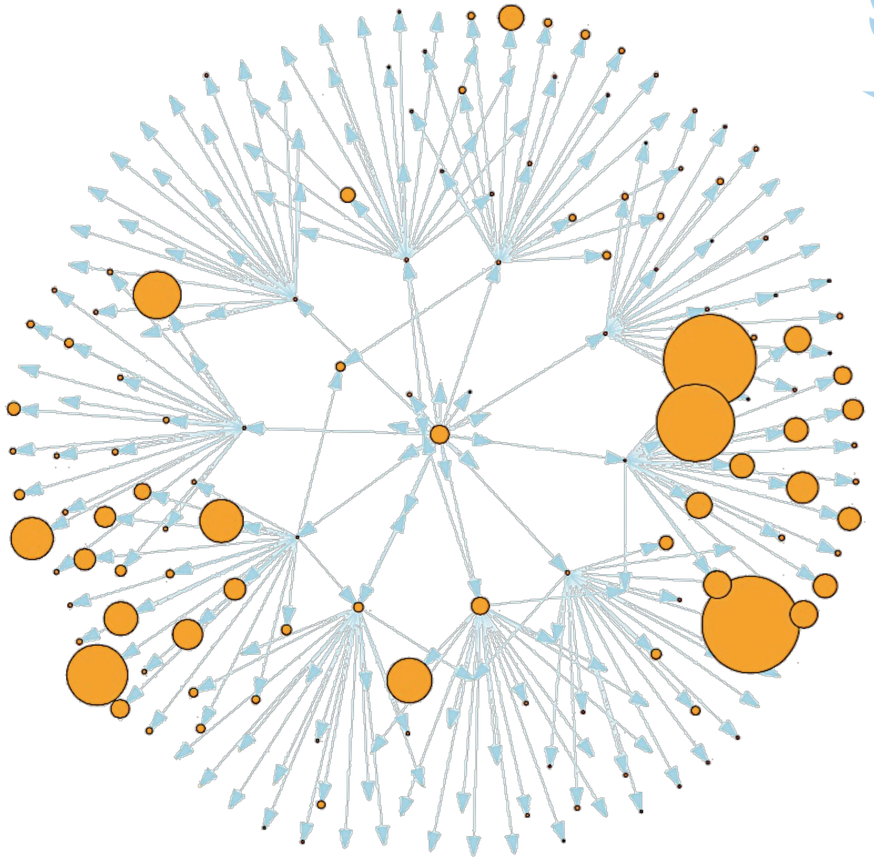


Arrows show direction of communication

Point size shows number of followers

Betweenness: directed network

- Direction of of the relationship tells you the significance of betweenness:
  - Betweenness in a directed network needs to compare in-degree and out-degree measurements in order to extract contextual meaning



Spreading a message:
not all followers are equally valuable!

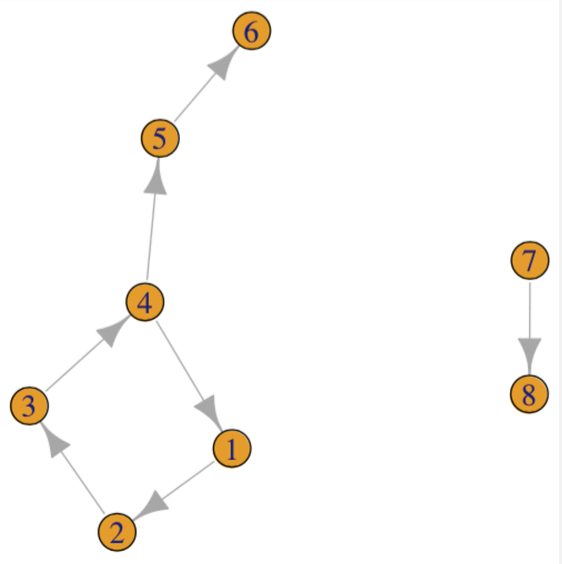
Arrows show direction of communication

Point size shows number of followers

Directed betweenness: worked example

Script

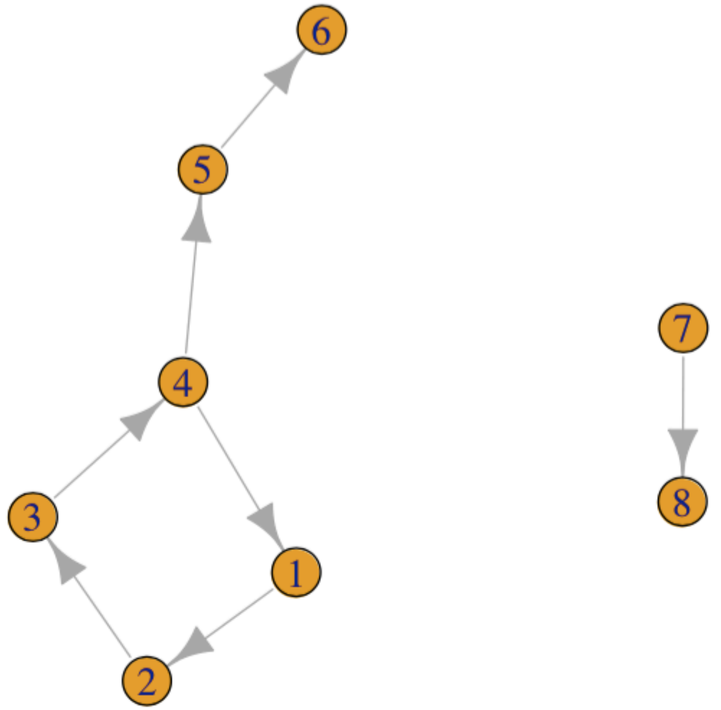
```
# First, create a graph. The make_graph() function in igraph
# creates a sample graph based on a vector of values that
# represent nodes.
k = make_graph(c(1, 2,
                 2, 3,
                 3, 4,
                 4, 1,
                 4, 5,
                 5, 6,
                 7, 8),
               directed = TRUE)

# Plot the network.
set.seed(1)
plot(k)

# Calculate directed betweenness.
bw_k = betweenness(k,
                   directed = TRUE)
bw_k
```



1. The graph object to use
2. Tell R that direction of connections matters

Directed betweenness: worked example

- 3 shortest paths pass through point 1:
  - 3 → 2
  - 4 → 2
  - 4 → 3
- 5 through point 2:
  - 4 → 3
  - 1 → 3
  - 1 → 4
  - 1 → 5
  - 1 → 6
- 7 through point 3:
  - 1 → 4
  - 1 → 5
  - 1 → 6
  - 2 → 4
  - 2 → 5
  - 2 → 6
  - 2 → 1




```
Console ~/Desktop/Network Analysis/Twitter/
> bw_k
[1] 3 5 7 9 4 0 0 0
```

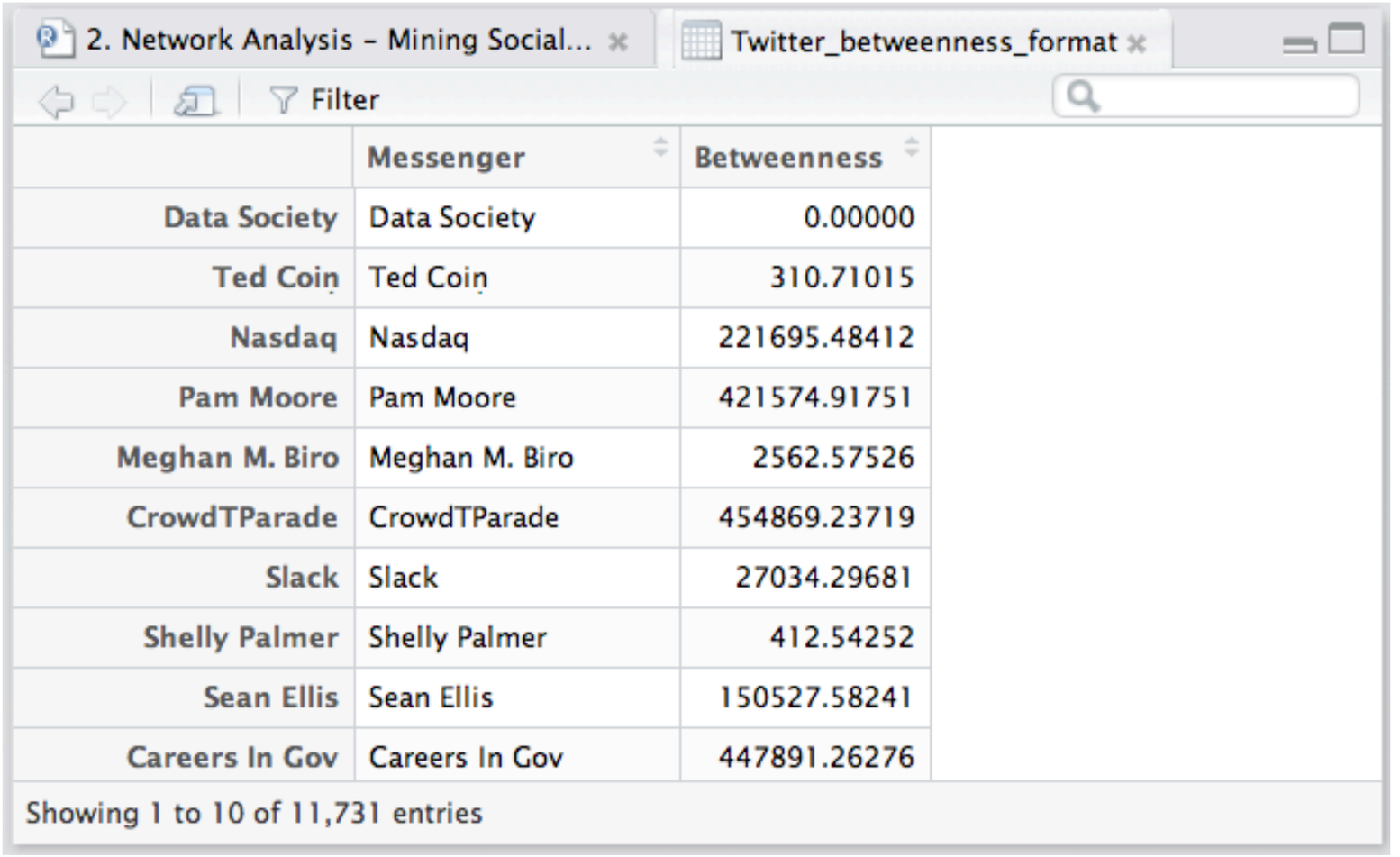
Twitter: directed betweenness

Script

```
# Create a directed graph object.
View(Twitter_network_clean)
Twitter_network_clean_directed = graph.data.frame(Twitter_network_clean,
                                                  directed = TRUE)

Twitter_betweenness = betweenness(Twitter_network_clean_directed,
                                  directed = TRUE)

# Format the output into a clean data frame.
Twitter_betweenness_data_frame = as.data.frame(Twitter_betweenness)
View(Twitter_betweenness_data_frame)

Twitter_betweenness_format = cbind(rownames(Twitter_betweenness_data_frame),
                                   Twitter_betweenness_data_frame)

colnames(Twitter_betweenness_format) = c("Messenger", "Betweenness")
View(Twitter_betweenness_format)
```

← Tell R that direction of connections matters

Twitter: directed betweenness



2. Network Analysis – Mining Social... | Twitter_betweenness_format

Filter

	Messenger	Betweenness
Data Society	Data Society	0.00000
Ted Coin	Ted Coin	310.71015
Nasdaq	Nasdaq	221695.48412
Pam Moore	Pam Moore	421574.91751
Meghan M. Biro	Meghan M. Biro	2562.57526
CrowdTParade	CrowdTParade	454869.23719
Slack	Slack	27034.29681
Shelly Palmer	Shelly Palmer	412.54252
Sean Ellis	Sean Ellis	150527.58241
Careers In Gov	Careers In Gov	447891.26276

Showing 1 to 10 of 11,731 entries

Combine degree and betweenness data

Script

```
# Check the structure of the data.
str(Twitter_betweenness_format)
Twitter_betweenness_format$Messenger = as.character(Twitter_betweenness_format
                                                    $Messenger)

# Combine the degree data set with the betweenness data set.
View(Twitter_network_degree_complete)
str(Twitter_network_degree_complete)

library(plyr)
Twitter_network_degree_betweenness = join(Twitter_network_degree_complete,
                                          Twitter_betweenness_format)

str(Twitter_network_degree_betweenness)
View(Twitter_network_degree_betweenness)

# Save your work.
write.csv(Twitter_network_degree_betweenness,
          "new Twitter_degree_betweenness.csv",
          row.names = FALSE)
```

Look for accounts with high betweenness and high out-degree centrality.

Those are the important connectors that spread information to the rest of the network!

Combine degree and betweenness data


```
Console ~/Desktop/Network Analysis/Twitter/
> str(Twitter_network_degree_betweenness)
'data.frame':   11731 obs. of  6 variables:
 $ Messenger       : chr  "Data Society" "Ted Coin" "Nasdaq" "Pam Moore" ...
 $ In-degree       : int  0 1 3 6 2 12 5 1 3 11 ...
 $ Out-degree      : int  30 30 60 30 60 60 30 30 30 30 ...
 $ Number_Followers: int  1589 445076 435305 248614 111419 117648 120517 89052 93533 126897 ...
 $ Number_Friends  : int  182 353470 4450 134658 88371 124793 58558 380 43082 93077 ...
 $ Betweenness     : num  0 311 221695 421575 2563 ...
```

2. Network Analysis - Mining Social... | Twitter_network_degree_betweenness

Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends	Betweenness
1	Data Society	0	30	1589	182	0.0000
2	Ted Coin	1	30	445076	353470	310.7101
3	Nasdaq	3	60	435305	4450	221695.4841
4	Pam Moore	6	30	248614	134658	421574.9175
5	Meghan M. Biro	2	60	111419	88371	2562.5753

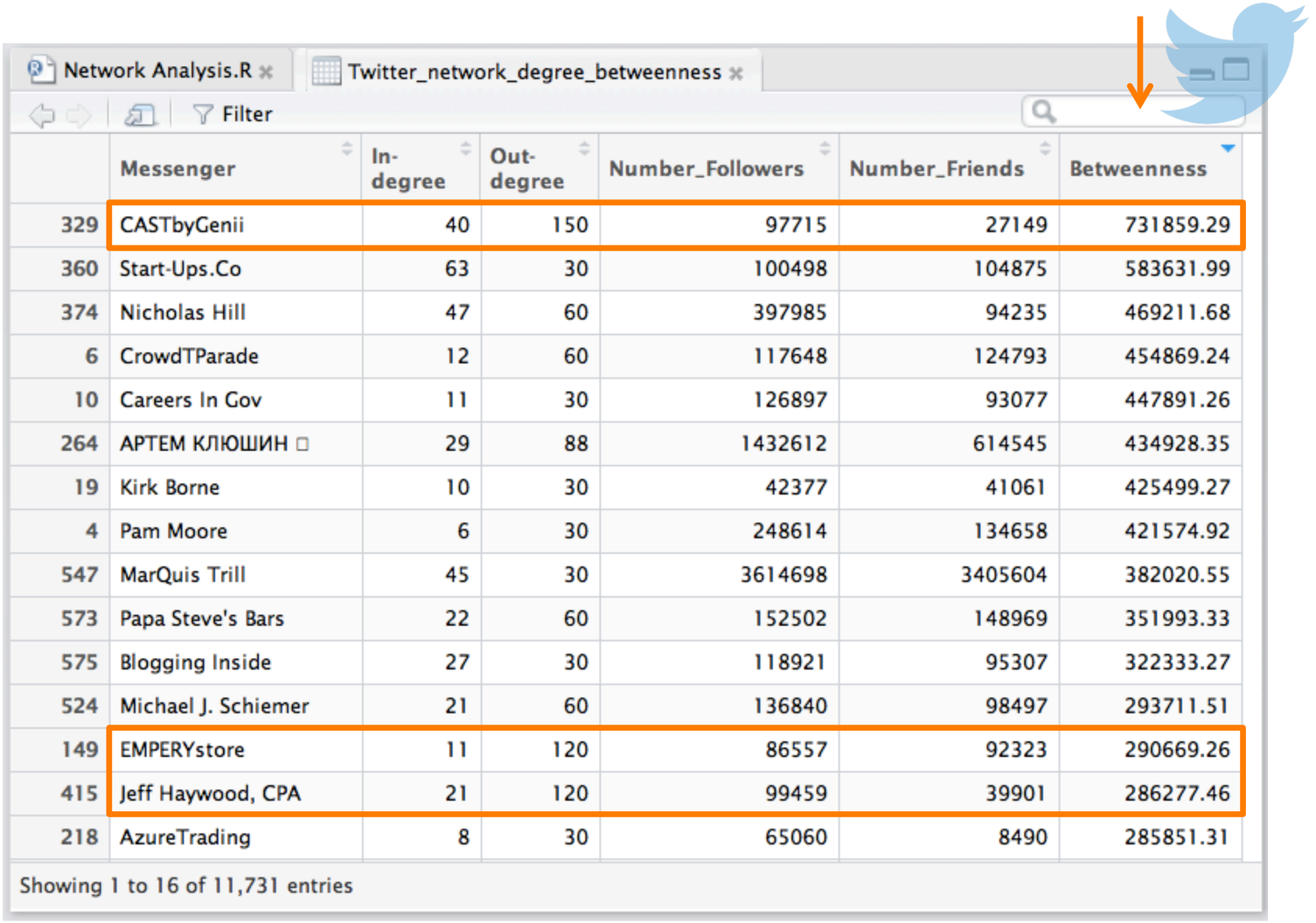
Showing 1 to 5 of 11,731 entries

Evaluate directed betweenness

- Sort the data based on betweenness

- You want to look for accounts with high betweenness and high out-degree centrality

Those are the important connectors that spread information to the rest of the network!



Network Analysis.R | Twitter_network_degree_betweenness

Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends	Betweenness
329	CASTbyGenii	40	150	97715	27149	731859.29
360	Start-Ups.Co	63	30	100498	104875	583631.99
374	Nicholas Hill	47	60	397985	94235	469211.68
6	CrowdTParade	12	60	117648	124793	454869.24
10	Careers In Gov	11	30	126897	93077	447891.26
264	АРТЕМ КЛЮШИН □	29	88	1432612	614545	434928.35
19	Kirk Borne	10	30	42377	41061	425499.27
4	Pam Moore	6	30	248614	134658	421574.92
547	MarQuis Trill	45	30	3614698	3405604	382020.55
573	Papa Steve's Bars	22	60	152502	148969	351993.33
575	Blogging Inside	27	30	118921	95307	322333.27
524	Michael J. Schiemer	21	60	136840	98497	293711.51
149	EMPERYstore	11	120	86557	92323	290669.26
415	Jeff Haywood, CPA	21	120	99459	39901	286277.46
218	AzureTrading	8	30	65060	8490	285851.31

Showing 1 to 16 of 11,731 entries

Evaluate directed betweenness

- If you sort the data:
  1. By decreasing in-degree
  2. By increasing betweenness

- You will see the top observers or collectors of information in this network

These accounts could be "spies" or fraudulent accounts in some situations

Network Analysis.R | Twitter_network_degree_betweenness

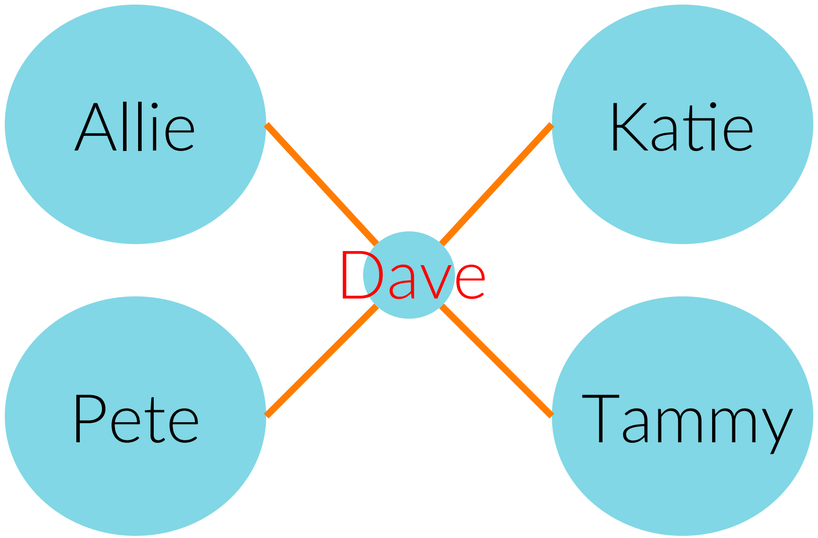
Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends	Betweenness
890	Wellbelove	49	0	356599	259655	0
1261	Mike Whitaker	48	0	425968	121023	0
812	JSWX	44	0	222042	242855	0
1315	Joshua Davidson	43	0	706818	240086	0
987	joshingstern	42	0	880551	681867	0
1612	⊰·(M G W V)·⊱	40	0	106847	105046	0
1178	Steve Keating CSE	36	0	742133	580806	0
1757	Martin Zwilling	35	0	795676	512289	0
1927	#AxelHappy	35	0	616804	228208	0
887	LOVE MUSIC TWEETS	32	0	575161	109378	0
889	uoʇuǝɹʇ uosɐɟ	29	0	412929	161428	0
1106	Emma Illies	29	0	468436	379259	0
1368	Terri Bauman	29	0	411169	411865	0
1433	Startup Companions	29	0	91441	5562	0
1942	Chop Dawg	28	0	505790	234041	0

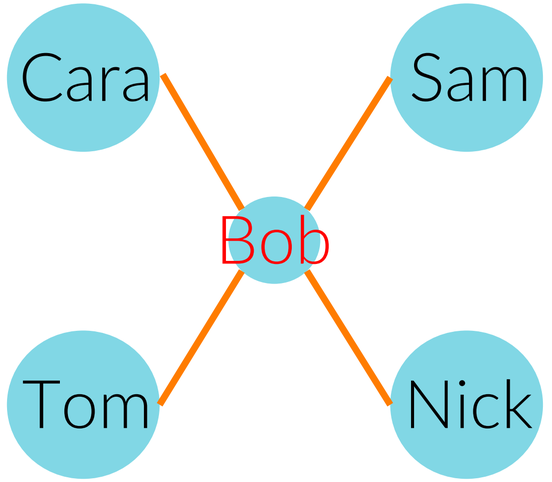
Showing 1 to 16 of 11,731 entries

Eigenvector (importance) centrality

- Eigenvector centrality – measures a node's (person's) importance by giving consideration to importance of the nodes (people) connected to it
  - Intuition: 300 very popular friends > 300 unpopular friends



Dave is more influential than Bob



Eigenvector centrality: Google

- Google's search algorithm is based on a method called PageRank

- PageRank quantifies the "importance" of a page based on how many "important" pages link to it and relate to it in other ways
  - Defining what's "important" is a large part of Google's search engine

- Use cases:
  1. Disease spread: who was patient #1?
     - Measure time of infection as "importance" for all patients to identify the culprit or "most important" one
  2. Who are someone's most important business and personal relationships?
     - Can be identified with social media data
     - Can be identified with e-mail communication data
  3. What are the key elements in a supply chain network?

Eigenvector centrality: measure trust

- We trust people we've known for a long time
  - Measure time
- We trust people who we consider "experts"
  - Measure number of publications or speaking events
- We trust people we interact with frequently
  - Measure frequency and duration of communications

#	Person 1	Person 2	Rank / seniority	Office location	Number of e-mails	Organizational relationship
1	Allie	Cara	CEO	New York	50	Sr.-Sub.
2	Bob	Allie	COO	Chicago	100	Sub-Sr.
3	Cara	Bob	CFO	Chicago	30	Team members
4	Dave	Allie	Assistant	New York	10	Sub.-Sr.

E-mail: who does Allie trust most?

Eigenvector centrality: measure trust

"A person trusts another if she is willing to take a risk based on her expectation that the trusted person's actions will lead to a positive outcome."

- Dr. Jennifer Golbeck, Professor of Network Analysis

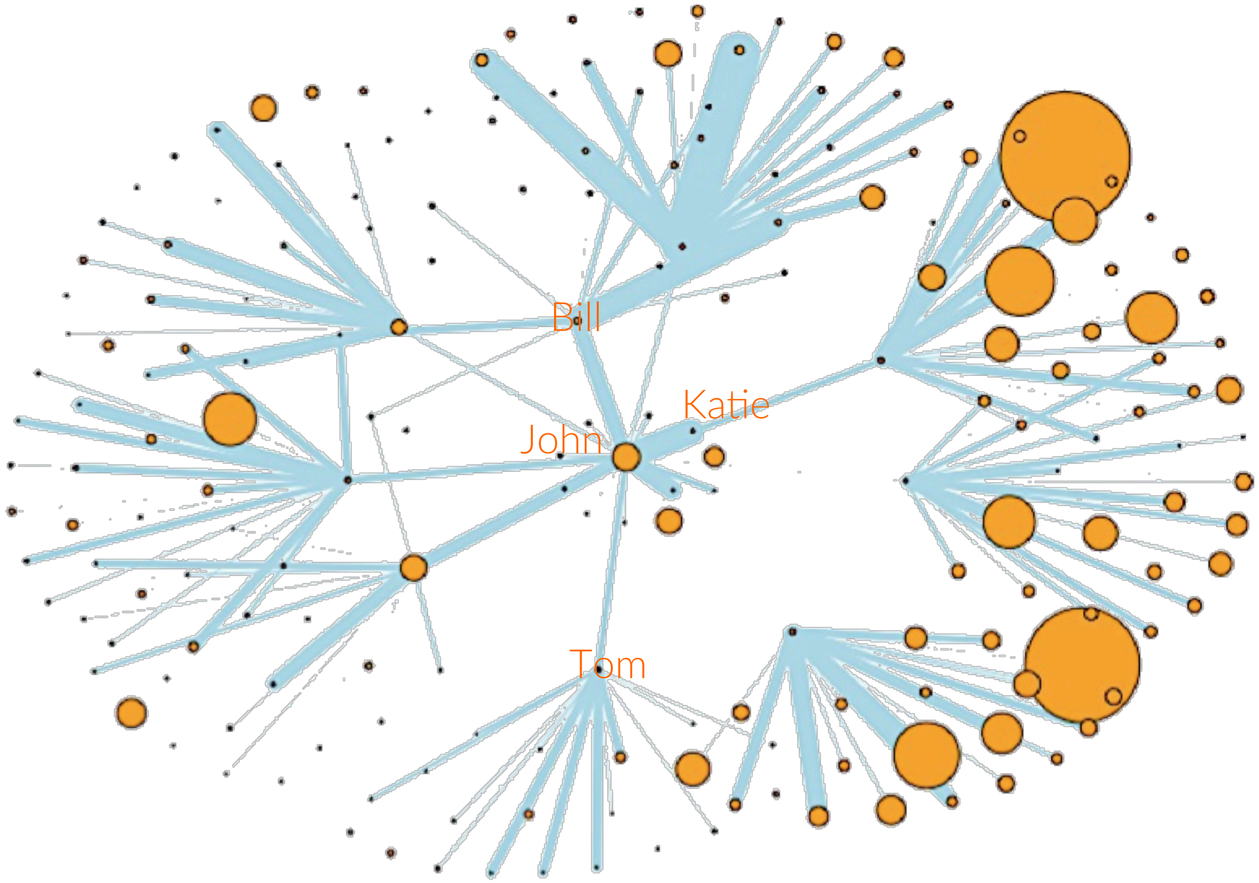
#	Person 1	Person 2	Rank / seniority	Office location	Number of e-mails	Organizational relationship
1	Allie	Cara	CEO	New York	50	Sr.-Sub.
2	Bob	Allie	COO	Chicago	100	Sub-Sr.
3	Cara	Bob	CFO	Chicago	30	Team members
4	Dave	Allie	Assistant	New York	10	Sub.-Sr.

E-mail: who does Allie trust most?

Eigenvector centrality: measure trust

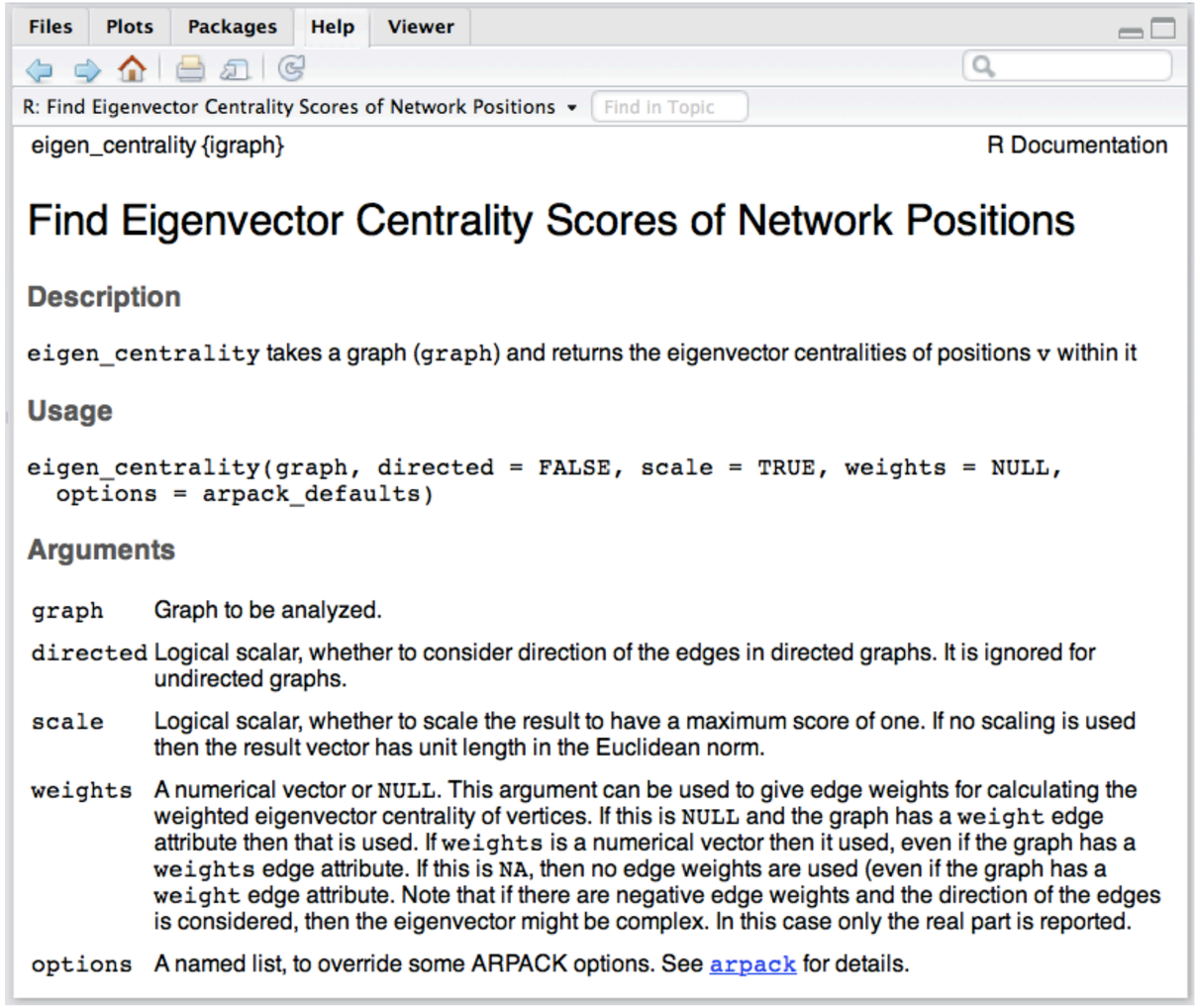
Twitter: whose opinion does John value?

Line thickness denotes frequency of re-tweets



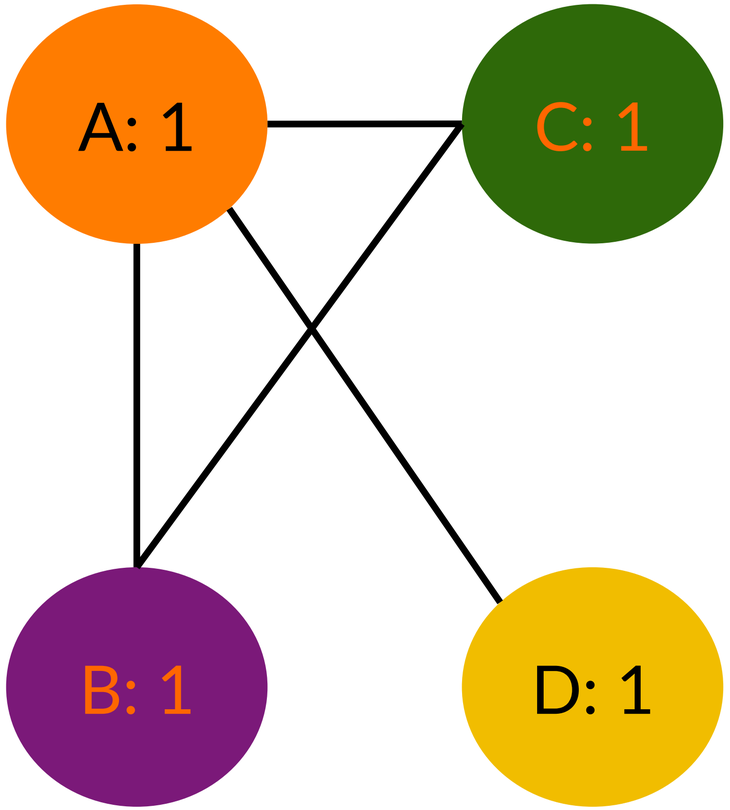
Eigenvector centrality in R

- Use `?eigen_centrality` to see the options you can use to adjust how eigenvector centrality is calculated
- The full math is explained in a supplement for those who are interested in learning more
- These materials explain the intuition behind the concept



Calculate undirected eigenvectors

1 = connection exists
0 = no connection

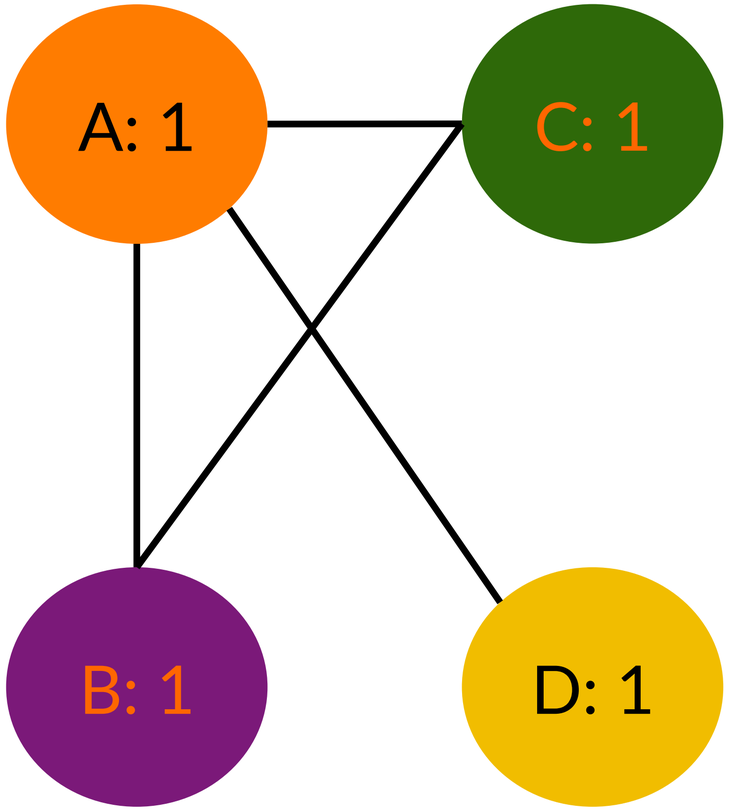


Adjacency Matrix

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

Calculate undirected eigenvectors

Step 1: calculate degree centrality by summing the rows and assigning those new values to the nodes



Adjacency Matrix

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

=

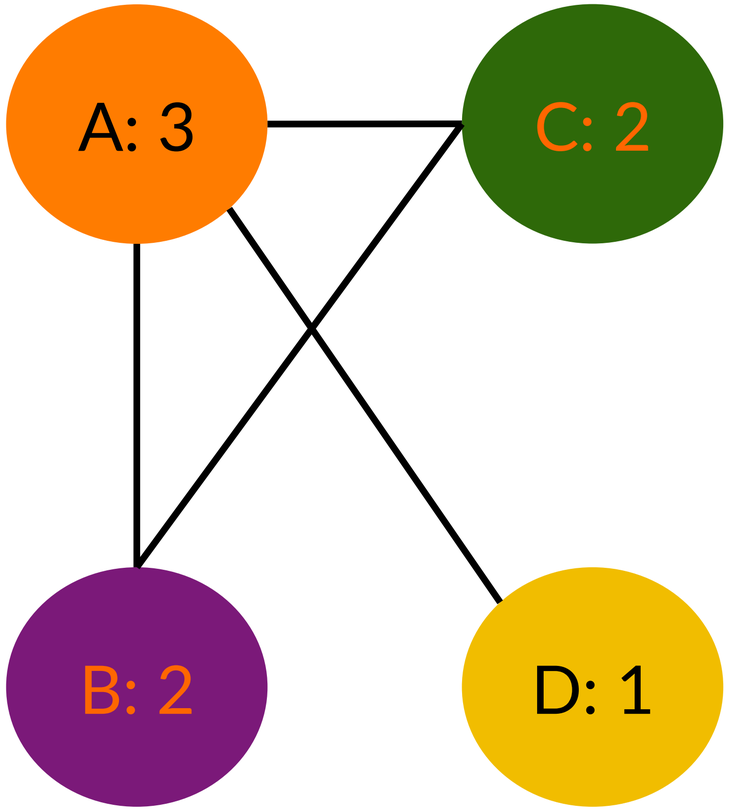
Expression
0+1+1+1
1+0+1+0
1+1+0+0
1+0+0+0

=

Sum
3
2
2
1

Calculate undirected eigenvectors

Step 2: sum the adjacent weights with the newly calculated values



Adjacency Matrix

	A	B	C	D
A	0	2	2	1
B	3	0	2	0
C	3	2	0	0
D	3	0	0	0

=

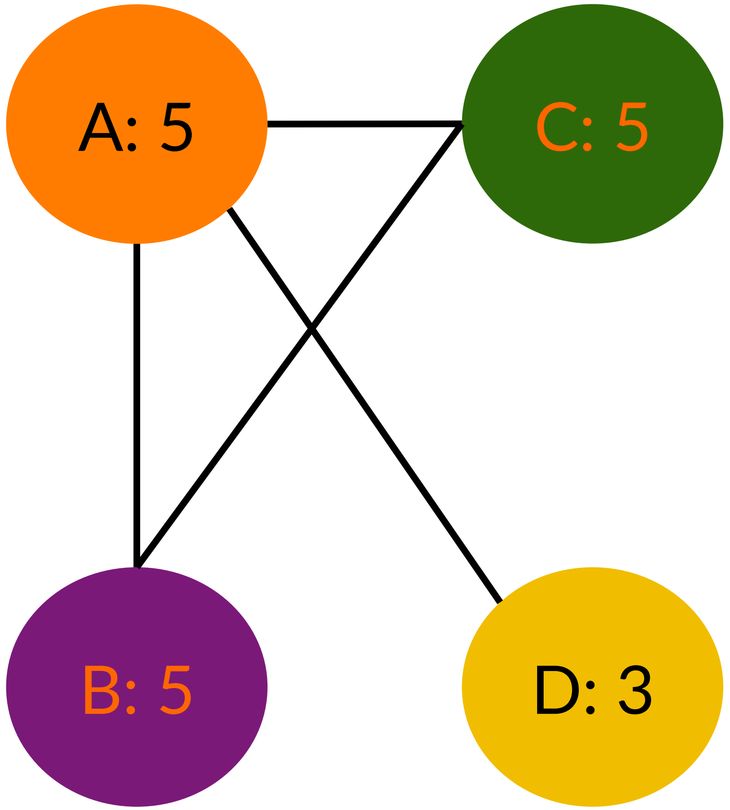
Expression
0+2+2+1
3+0+2+0
3+2+0+0
3+0+0+0

=

Sum
5
5
5
3

Calculate undirected eigenvectors

Step 3: repeat until the weights eventually converge
(see the supplement for how to do that with matrix math)



Adjacency Matrix

	A	B	C	D
A	0	5	5	3
B	5	0	5	0
C	5	5	0	0
D	5	0	0	0

=

Expression
0+5+5+3
5+0+5+0
5+5+0+0
5+0+0+0

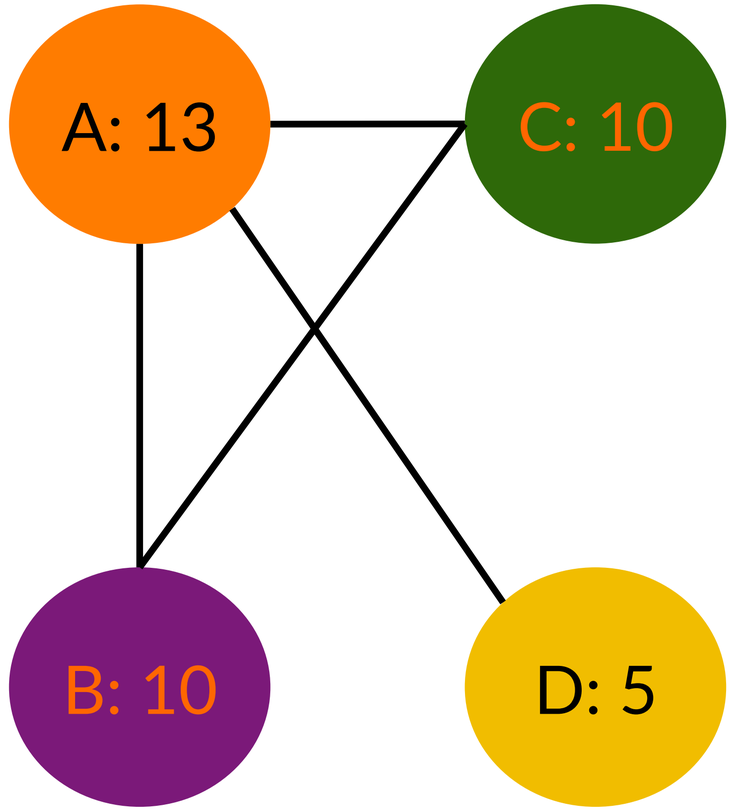
=

Sum
13
10
10
5

Calculate undirected eigenvectors

Step 4: when the values stabilize, they can be represented by a vector of values called an *eigenvector*

	A	B	C	D
Eigenvector	13	10	10	5



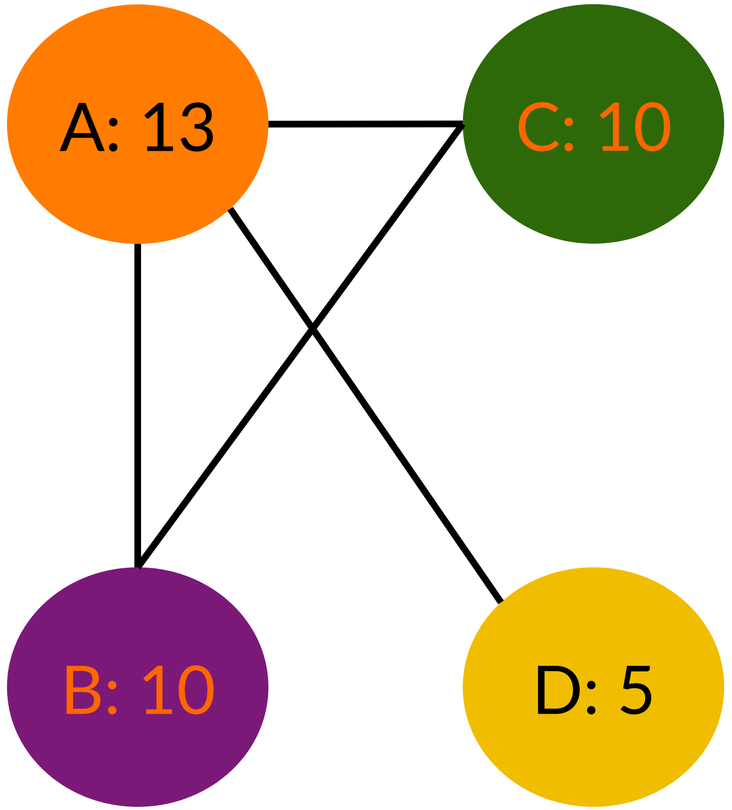
Calculate undirected eigenvectors

Step 5: each value in the eigenvector is divided by the top value to determine its relative eigenvector centrality

The relative values are the values that R outputs!

Note that they range from 0 to 1

	A	B	C	D
Eigenvector	13	10	10	5
Relative value	13/13	10/13	10/13	5/13



Calculating eigenvector centrality

value computed from the elements of the matrix

a matrix (i.e. adjacency matrix)

an eigenvalue of matrix A

identity matrix of A

$$\text{determinant}(A - \lambda * I) = 0$$

Let's use the following matrices:

Note that this is an adjacency matrix

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} \qquad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We want to find the value of λ

What's an identity matrix?

- An identity matrix = a diagonal with 1's and the rest of the numbers are 0's:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- When a matrix is multiplied by an identity matrix, it's left unchanged:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} a*1 + b*0 & a*0 + b*1 \\ c*1 + d*0 & c*0 + d*1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Matrix multiplication with scalars

- A scalar is just a regular number that is multiplied by a matrix:

$$2 \times \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 2a & 2b \\ 2c & 2d \end{bmatrix}$$

scalar

$$\lambda \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda \times 1 & \lambda \times 0 \\ \lambda \times 0 & \lambda \times 1 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

Identity matrix

Calculate eigenvalues

- A = a matrix, such as an adjacency matrix
- I = identity matrix of A
- λ = an eigenvalue of matrix A

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Find λ such that:
  determinant(A – λ * I) = 0

Effectively, an eigenvalue condenses a matrix to one number!

$$\begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} - \lambda \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} 2 - \lambda & 2 - 0 \\ 5 - 0 & -1 - \lambda \end{bmatrix}$$

Calculate eigenvalues

- A = a matrix, such as an adjacency matrix
- I = identity matrix of A
- λ = an eigenvalue of matrix A

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Find λ such that:

  $\text{determinant}(A - \lambda * I) = 0$

$$\text{determinant} \begin{bmatrix} 2 - \lambda & 2 - 0 \\ 5 - 0 & -1 - \lambda \end{bmatrix} = (2 - \lambda) * (-1 - \lambda) - (2 - 0) * (5 - 0)$$

Calculate eigenvalues

- A = a matrix, such as an adjacency matrix
- I = identity matrix of A
- λ = an eigenvalue of matrix A

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

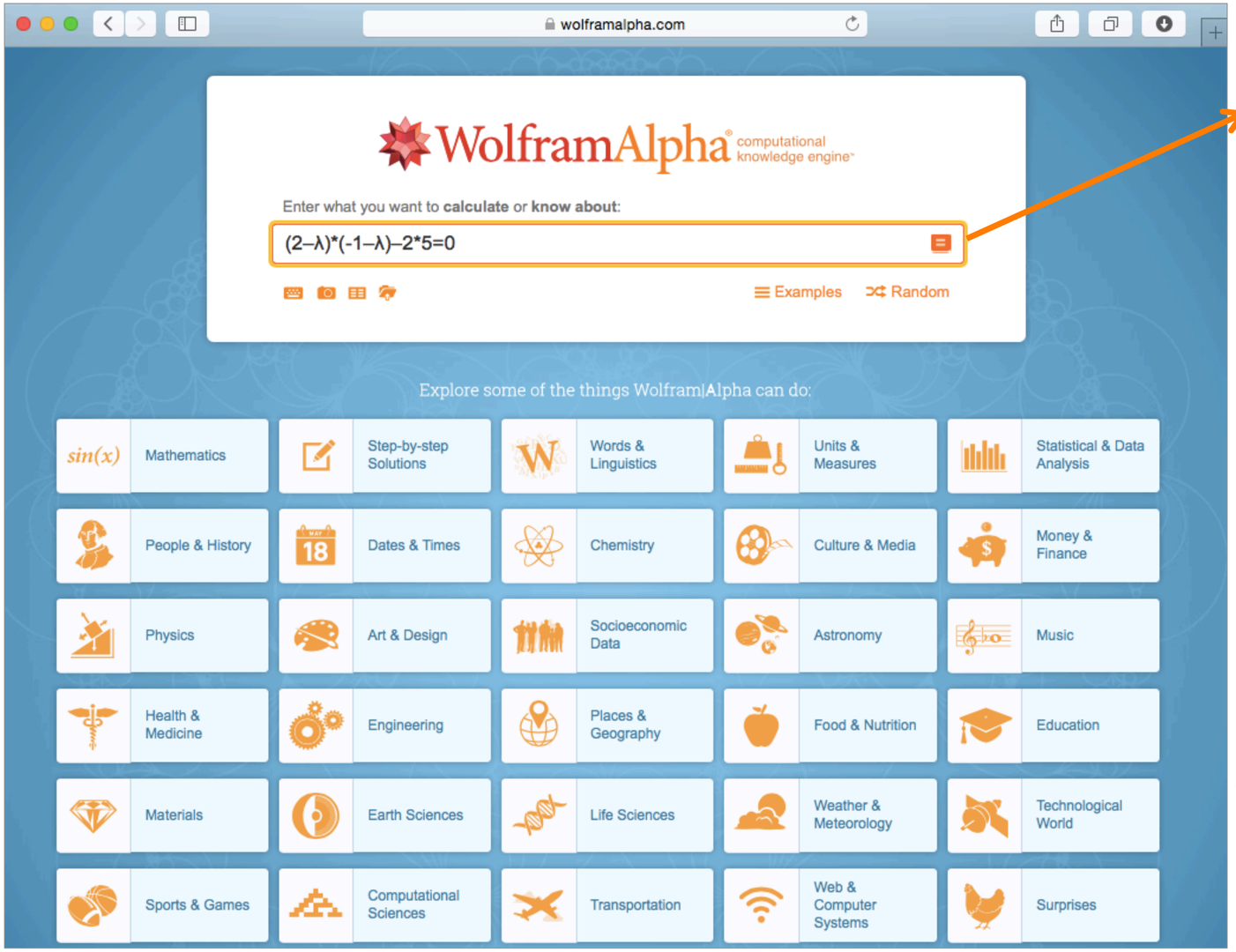
- Find λ such that:
  determinant(A – λ * I) = 0

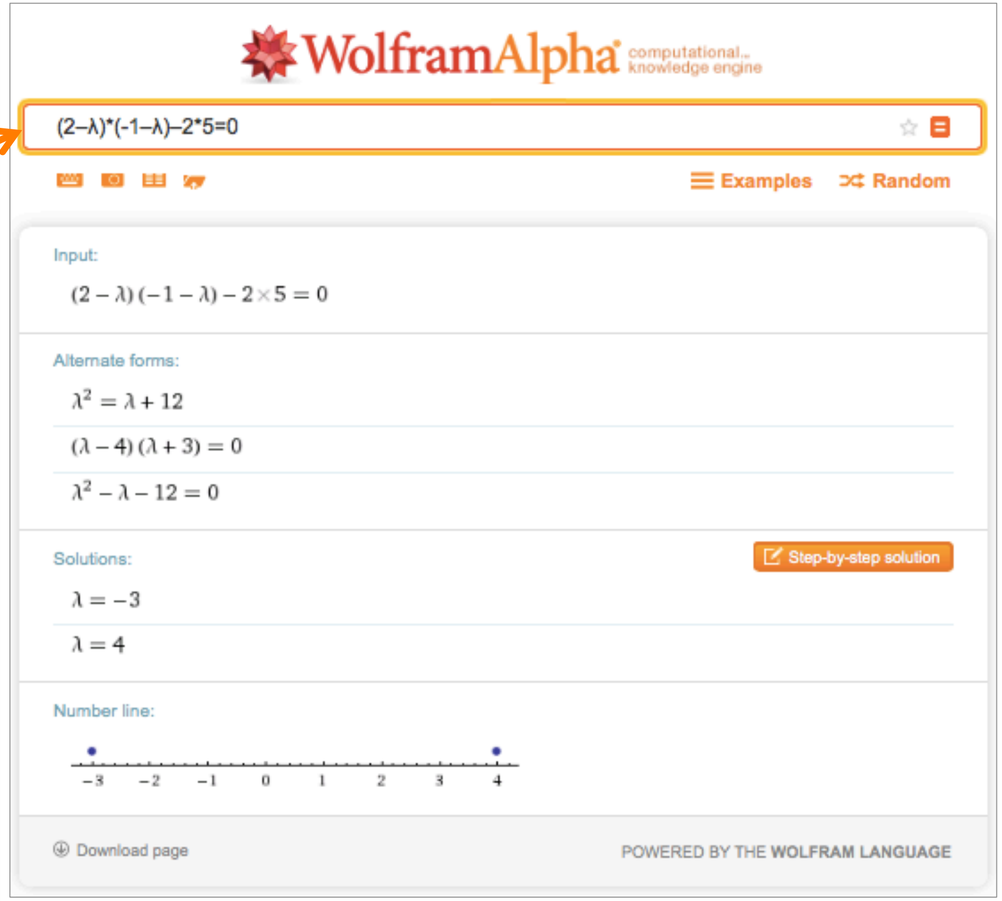
$$(2 - \lambda) * (-1 - \lambda) - (2 - 0) * (5 - 0) = 0$$

$$\lambda^2 - \lambda - 12 = 0$$

$$\lambda = -3, 4$$

Solving equations





Calculate eigenvector

a matrix (i.e. adjacency matrix)

eigenvector of matrix A

an eigenvalue of matrix A

eigenvector of matrix A

$$A * \vec{v} = \lambda * \vec{v}$$

$$A * \vec{v} - \lambda * \vec{v} = \vec{0}$$

identity matrix of A

$$A * \vec{v} - \lambda * \vec{v} * I = \vec{0}$$

We already proved that this…

$$(A - \lambda * I) * \vec{v} = \vec{0}$$

is equal to that

$$\begin{bmatrix} 2 - \lambda & 2 - 0 \\ 5 - 0 & -1 - \lambda \end{bmatrix} * \vec{v} = \vec{0}$$

Calculate eigenvector

- A = a matrix, such as an adjacency matrix
- I = identity matrix of A
- λ = an eigenvalue of matrix A
- $\vec{v}$ = eigenvector of matrix A

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 - \lambda & 2 - 0 \\ 5 - 0 & -1 - \lambda \end{bmatrix} * \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \vec{0}$$

Use the largest eigenvalue (you'll see why shortly)

$$\begin{bmatrix} 2 - 4 & 2 \\ 5 & -1 - 4 \end{bmatrix} * \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \vec{0} \Rightarrow \begin{matrix} -2 * \nu_1 + 2 * \nu_2 = 0 \\ 5 * \nu_1 - 5 * \nu_2 = 0 \end{matrix}$$

Calculate eigenvector

$$-2 * \nu_1 + 2 * \nu_2 = 0$$

$$5 * \nu_1 - 5 * \nu_2 = 0$$

(1) $2 * \nu_1 = 2 * \nu_2$

(2) $5 * \nu_1 = 5 * \nu_2$

Recall that an eigenvector compares the relative centrality of the nodes, so if we know that $v_1 = v_2$, then it doesn't matter which value we take. For simplicity, let's use 1.

(3) $\nu_1 = \nu_2 = 1$

(4) $\vec{\nu} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Eigenvectors: what does the math mean?

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} \longrightarrow$$

Adjacency Matrix

	A	B
A	2	2
B	5	-1

$=$

Expression
2 + 2
5 + -1

$=$

Sum
4
4

$\longrightarrow \vec{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$\text{determinant} \begin{bmatrix} 2 - \lambda & 2 - 0 \\ 5 - 0 & -1 - \lambda \end{bmatrix} = (2 - \lambda) * (-1 - \lambda) - (2 - 0) * (5 - 0)$$

Why the largest eigenvalue?

- A = a matrix, such as an adjacency matrix
- I = identity matrix of A
- λ = an eigenvalue of matrix A
- $\vec{v}$ = eigenvector of matrix A

$$A = \begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\vec{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\lambda = -3, 4$$

$$\begin{bmatrix} 2 + 3 & 2 \\ 5 & -1 + 3 \end{bmatrix} * \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \vec{0} \Rightarrow \begin{aligned} 5 * v_1 + 2 * v_2 &= 0 \\ 5 * v_1 + 2 * v_2 &= 0 \end{aligned}$$

The largest eigenvalue ensures that the eigenvector values are positive and not too large!

Determinant of a 3x3 matrix

$$\text{determinant} \begin{bmatrix} 1-\lambda & 2 & 2 \\ 2 & 1-\lambda & 2 \\ 2 & 2 & 1-\lambda \end{bmatrix} = 0$$

Adjacency Matrix

	A	B	C
A	1	2	2
B	2	1	2
C	2	2	1

Determinant of a 3x3 matrix

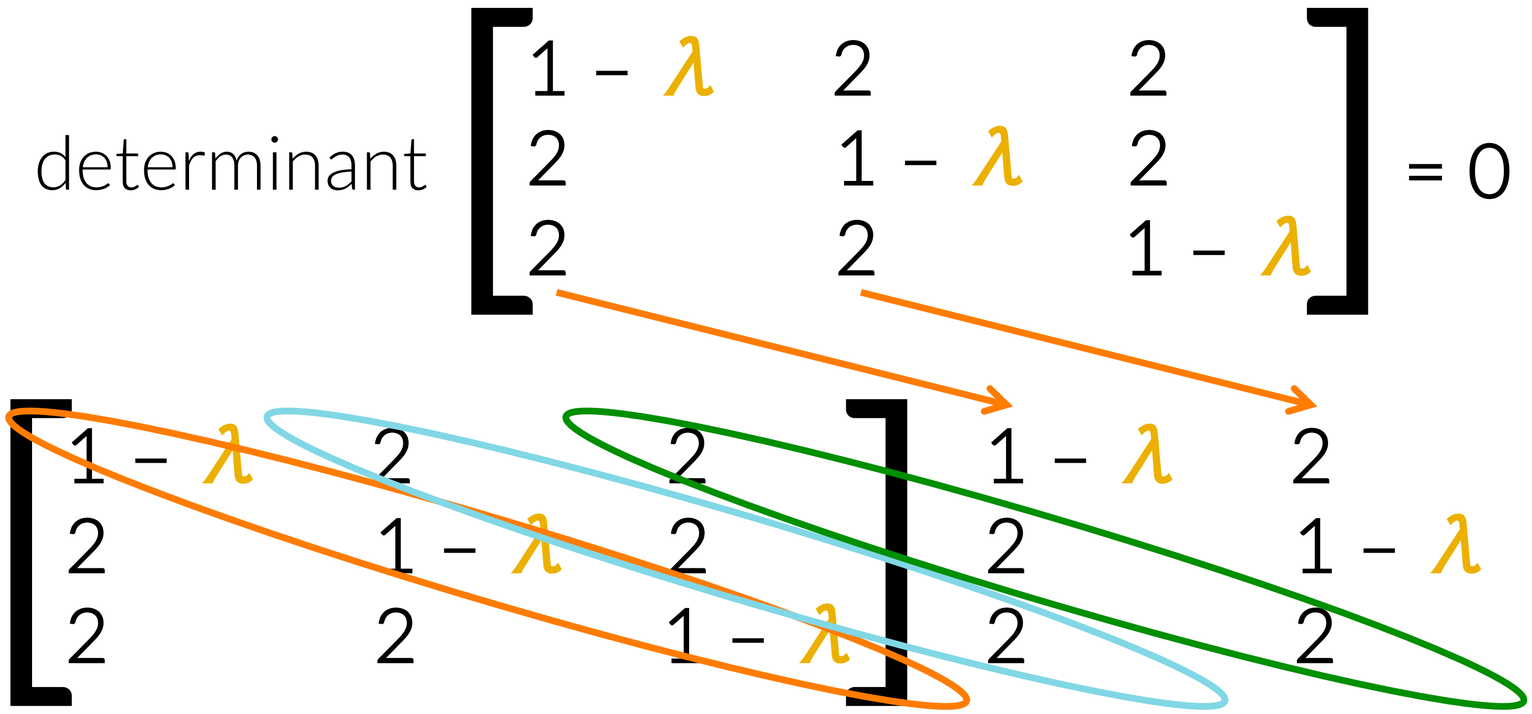
$$\text{determinant} \begin{bmatrix} 1-\lambda & 2 & 2 \\ 2 & 1-\lambda & 2 \\ 2 & 2 & 1-\lambda \end{bmatrix} = 0$$

$$\begin{bmatrix} 1-\lambda & 2 & 2 \\ 2 & 1-\lambda & 2 \\ 2 & 2 & 1-\lambda \end{bmatrix} \begin{matrix} 1-\lambda & 2 \\ 2 & 1-\lambda \\ 2 & 2 \end{matrix}$$

Adjacency Matrix

	A	B	C
A	1	2	2
B	2	1	2
C	2	2	1

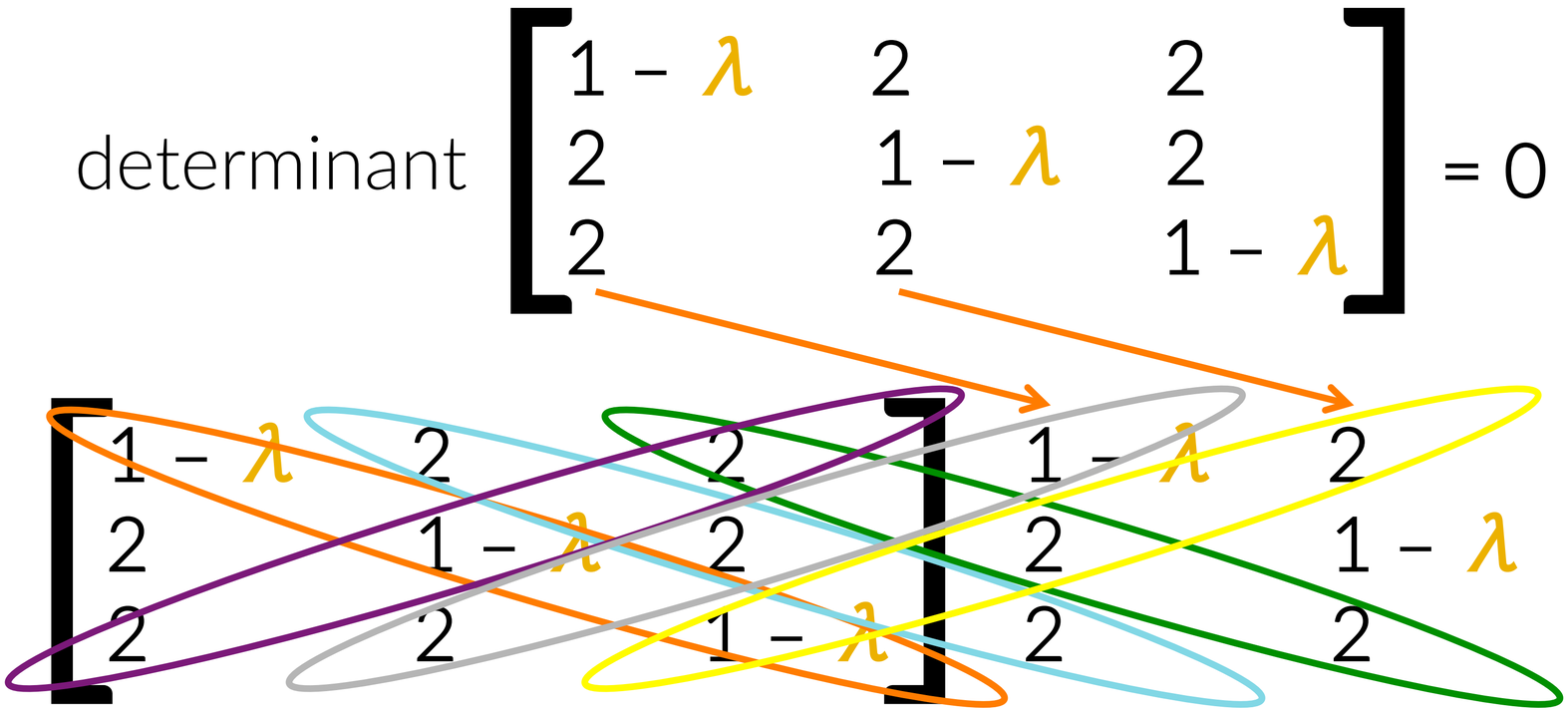
Determinant of a 3x3 matrix



Adjacency Matrix

	A	B	C
A	1	2	2
B	2	1	2
C	2	2	1

Determinant of a 3x3 matrix

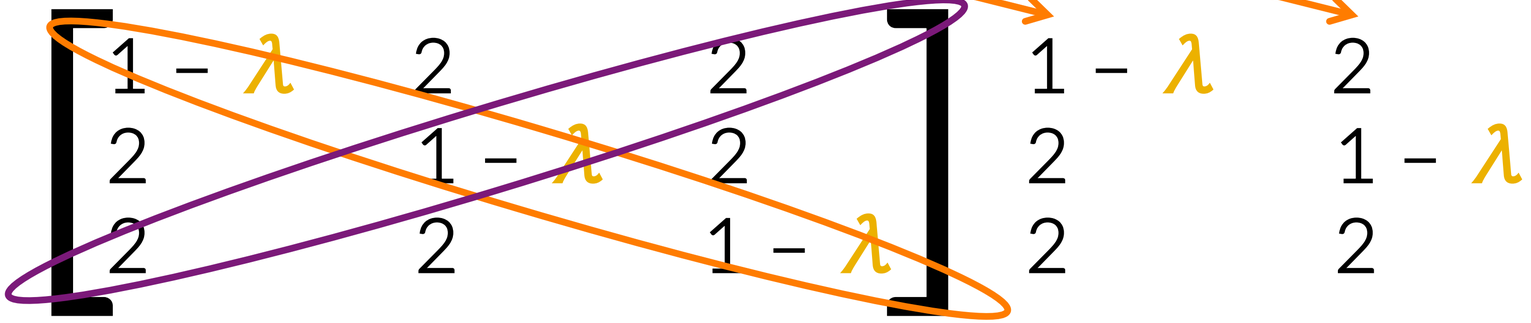


Adjacency Matrix

	A	B	C
A	1	2	2
B	2	1	2
C	2	2	1

Determinant of a 3x3 matrix

determinant $\begin{bmatrix} 1-\lambda & 2 & 2 \\ 2 & 1-\lambda & 2 \\ 2 & 2 & 1-\lambda \end{bmatrix} = 0$



Adjacency Matrix

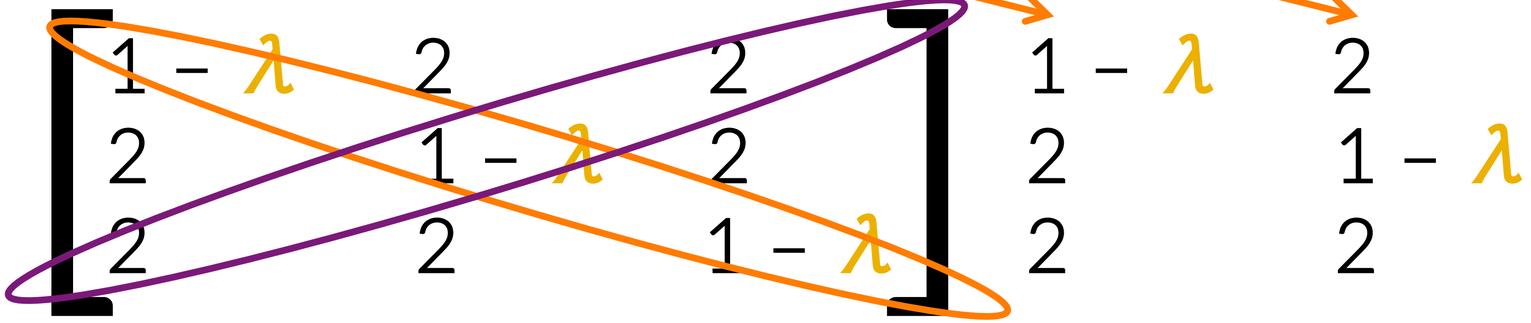
	A	B	C
A	1	2	2
B	2	1	2
C	2	2	1

$$((1-\lambda)*(1-\lambda)*(1-\lambda)) + (2*2*2) + (2*2*2)$$

$$-((2*(1-\lambda)*2) - (2*2*(1-\lambda)) - ((1-\lambda)*2*2) = 0$$

Determinant of a 3x3 matrix

determinant $\begin{bmatrix} 1-\lambda & 2 & 2 \\ 2 & 1-\lambda & 2 \\ 2 & 2 & 1-\lambda \end{bmatrix} = 0$



Adjacency Matrix

	A	B	C
A	1	2	2
B	2	1	2
C	2	2	1

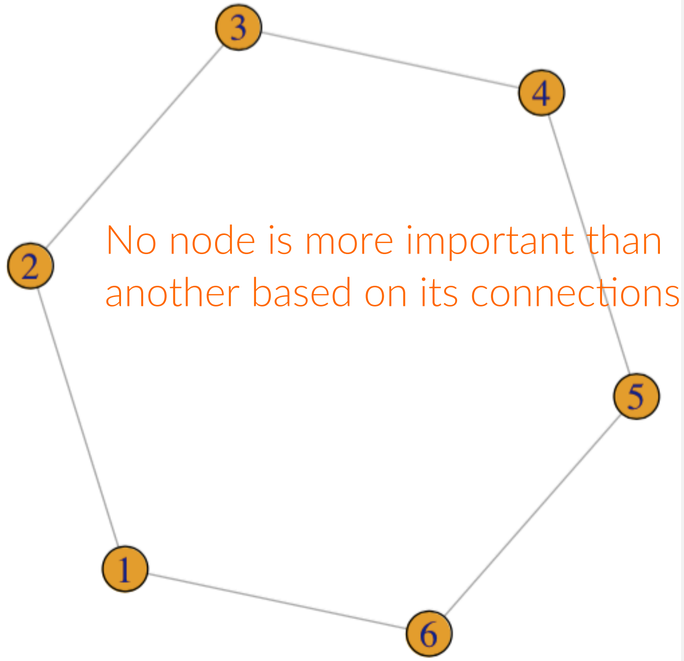
$$(1 - \lambda) - 16 * (1 - \lambda) + 16 = 0$$

$$\lambda = -2.4, -0.1, 5.4$$

Eigenvector centrality: example 1

Script

```
# Example 1: define node connections using make_graph().
k = make_graph(c(1, 2,
                 2, 3,
                 3, 4,
                 4, 5,
                 5, 6,
                 6, 1),
               directed = FALSE)
set.seed(2)
plot(k)

# Calculate eigenvector centrality.
eigen_k = eigen_centrality(k,
                           directed = FALSE)
eigen_k
```



No node is more important than another based on its connections

```
Console ~/Desktop/Network Analysis/Twitter/
> eigen_k
$vector
[1] 1 1 1 1 1 1
```

`eigen_centrality` has other outputs that we'll discuss in the supplement when we review the underlying math

Eigenvector centrality: example 2

Script

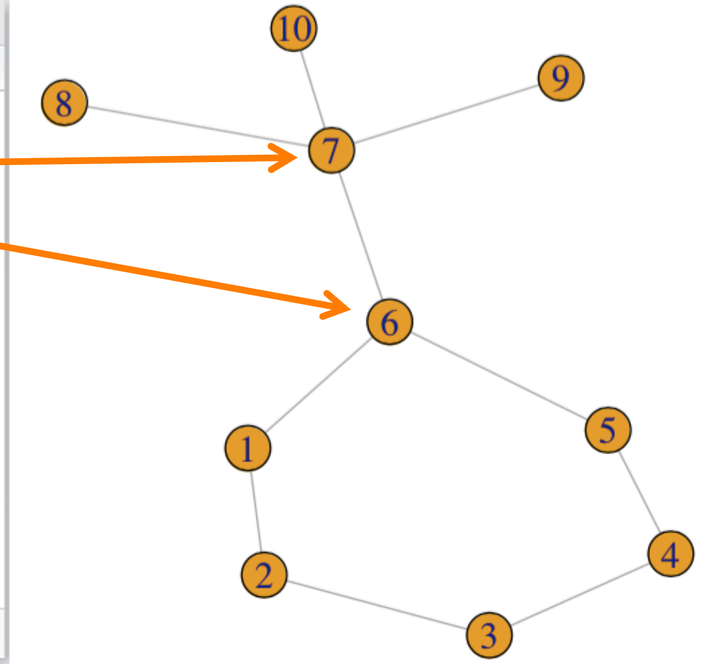
```
l = make_graph(c(1, 2,
                 2, 3,
                 3, 4,
                 4, 5,
                 5, 6,
                 6, 1,
                 6, 7,
                 7, 8,
                 7, 9,
                 7, 10),
               directed = FALSE)
set.seed(2)
plot(l)

# Calculate eigenvector centrality.
eigen_l = eigen_centrality(l,
                           directed = FALSE)
View(as.data.frame(eigen_l$vector))
```

as.data.frame(eigen_l$vector)

Filter

	eigen_l$vector
7	1.0000000
6	0.9771975
1	0.6180340
5	0.6180340
2	0.4370160
9	0.4370160
10	0.4370160
4	0.4370160
8	0.4370160
3	0.3819660

Showing 1 to 10 of 10 entries



Node 7 has the highest eigenvector centrality because it has a degree of 4 and it's connected to node 6, which is connected to many nodes that are in turn connected to other nodes

Eigenvector centrality: example 3

Script

```
m = make_graph(c(1, 2,
                 2, 3,
                 3, 4,
                 4, 5,
                 5, 6,
                 6, 1,
                 6, 7,
                 7, 8,
                 7, 9,
                 7, 10),
               directed = TRUE)
set.seed(2)
plot(m,
     edge.arrow.size = 1)

# Calculate eigenvector centrality.
eigen_m = eigen_centrality(m,
                           directed = TRUE)
View(as.data.frame(eigen_m$vector))
```

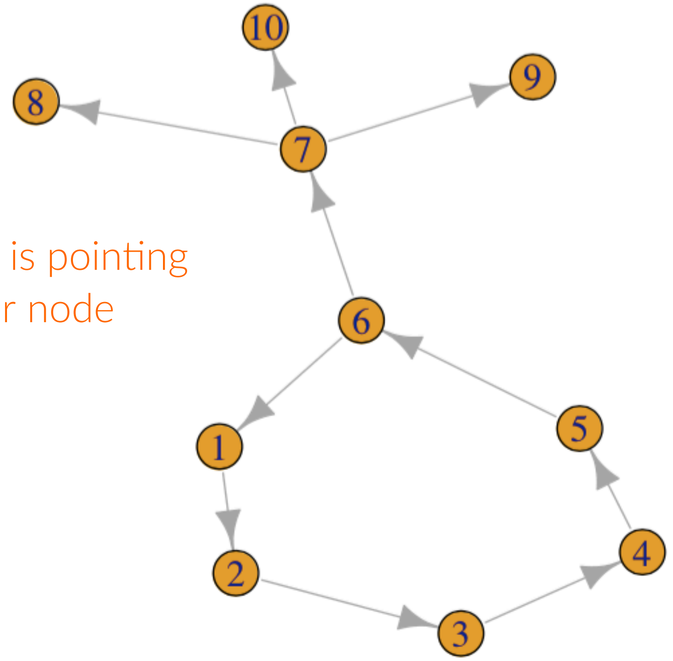
as.data.frame(eigen_m$vector)

Filter

	eigen_m$vector
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Showing 1 to 10 of 10 entries

Only 1 node is pointing to each other node



The nodes with the highest eigenvector centrality are the ones to which connections go from the greatest number of nodes with the highest eigenvector centrality

Eigenvector centrality: example 4

Script

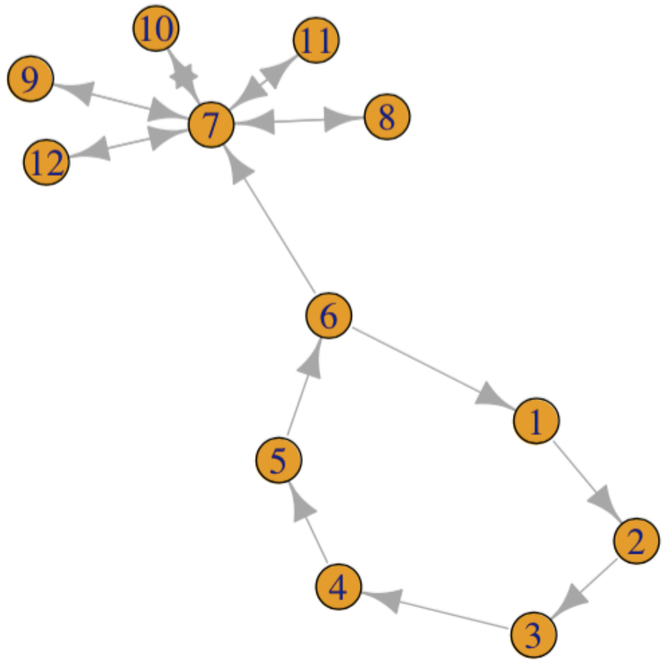
```
n = make_graph(c(1, 2, 2, 3,
                 3, 4, 4, 5,
                 5, 6, 6, 1,
                 6, 7, 7, 8,
                 7, 9, 7, 10,
                 7, 11,
                 7, 12,
                 12, 7,
                 11, 7,
                 10, 7,
                 9, 7,
                 8, 7),
               directed = TRUE)
set.seed(1)
plot(n, edge.arrow.size = 1)

# Calculate eigenvector centrality.
eigen_n = eigen_centrality(n,
                           directed = TRUE)
View(as.data.frame(eigen_n$vector))
```

as.data.frame(eigen_n$vector)

Filter

	eigen_n$vector
7	1.000000e+00
9	4.472136e-01
8	4.472136e-01
12	4.472136e-01
10	4.472136e-01
11	4.472136e-01
3	2.112615e-16
5	2.060866e-16
6	1.967335e-16
2	1.772966e-16
4	1.335570e-16
1	1.002606e-16

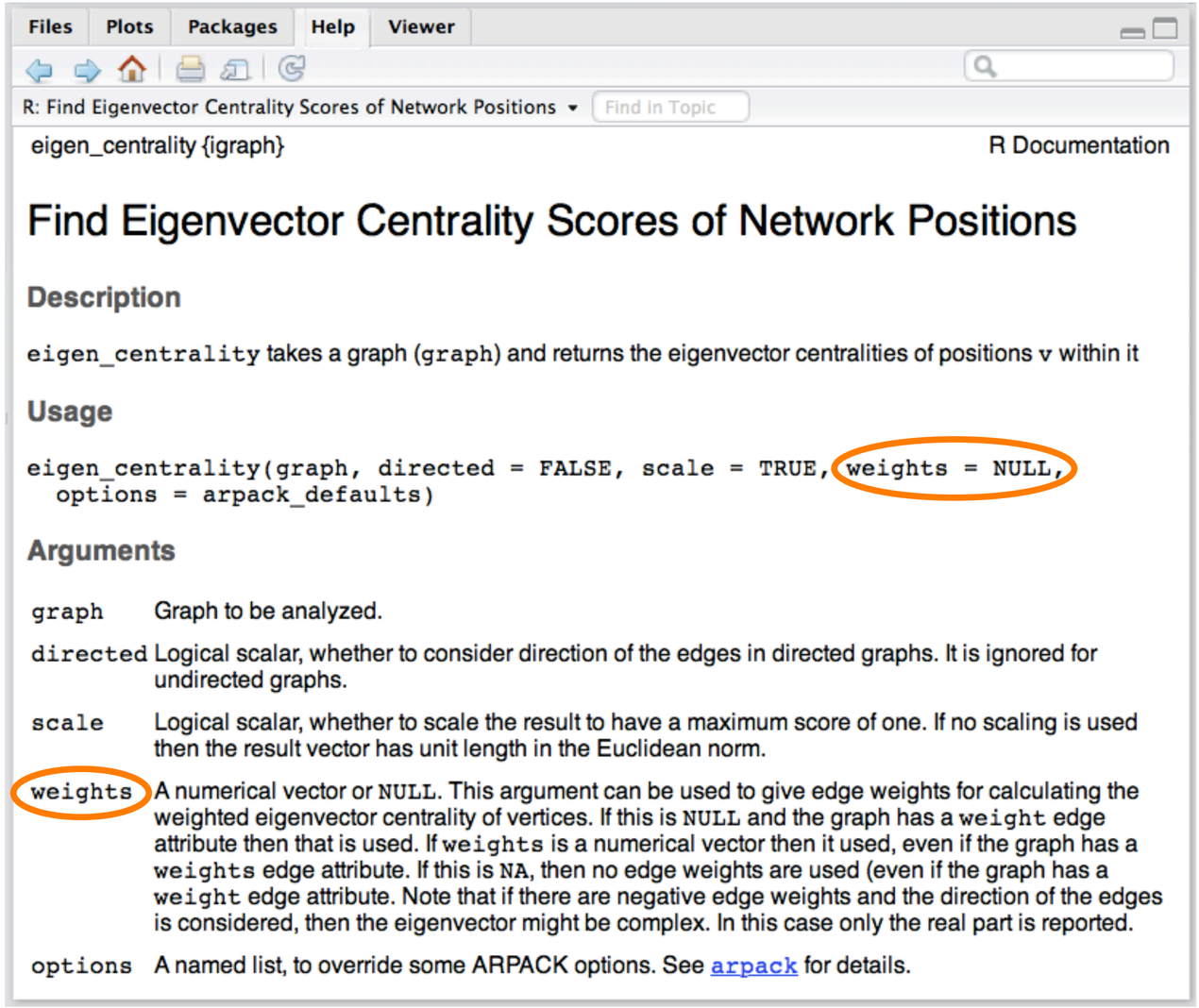
Showing 1 to 12 of 12 entries



The nodes with the highest eigenvector centrality are the ones to which connections go from the greatest number of nodes with the highest eigenvector centrality

Eigenvector centrality in R

- If you don't specify `weights`, then degree or edge centrality is used
- If edges are weighted, then the weights are used by default unless you set `weights = NULL`
- You can give R a vector that matches node names to use for calculating eigenvector centrality



Files | Plots | Packages | Help | Viewer

R: Find Eigenvector Centrality Scores of Network Positions — Find in Topic

eigen_centrality {igraph} — R Documentation

Find Eigenvector Centrality Scores of Network Positions

Description

`eigen_centrality` takes a graph (`graph`) and returns the eigenvector centralities of positions `v` within it

Usage

```
eigen_centrality(graph, directed = FALSE, scale = TRUE, weights = NULL,
  options = arpack_defaults)
```

Arguments

`graph`	Graph to be analyzed.
`directed`	Logical scalar, whether to consider direction of the edges in directed graphs. It is ignored for undirected graphs.
`scale`	Logical scalar, whether to scale the result to have a maximum score of one. If no scaling is used then the result vector has unit length in the Euclidean norm.
`weights`	A numerical vector or `NULL`. This argument can be used to give edge weights for calculating the weighted eigenvector centrality of vertices. If this is `NULL` and the graph has a `weight` edge attribute then that is used. If `weights` is a numerical vector then it used, even if the graph has a `weights` edge attribute. If this is `NA`, then no edge weights are used (even if the graph has a `weight` edge attribute. Note that if there are negative edge weights and the direction of the edges is considered, then the eigenvector might be complex. In this case only the real part is reported.
`options`	A named list, to override some ARPACK options. See `arpack` for details.

Twitter: eigenvector centrality

Script

```
# Since, in our network, the connection direction denotes the direction of
# information flow, we should reverse the order of the columns.
View(Twitter_network_clean)
Twitter_network_clean_reverse = Twitter_network_clean[, c(2, 1, 3:6)]

Twitter_network_clean_directed = graph.data.frame(Twitter_network_clean_reverse,
                                                  directed = TRUE)

# Calculate eigenvector centrality based on the properties of the given network.
Twitter_eigen = eigen_centrality(Twitter_network_clean_directed,
                                 directed = TRUE)

Twitter_eigen_data_frame = as.data.frame(Twitter_eigen)
View(Twitter_eigen_data_frame)

Twitter_eigen_format = cbind(rownames(Twitter_eigen_data_frame),
                             Twitter_eigen_data_frame$vector)

colnames(Twitter_eigen_format) = c("Messenger", "Eigenvector")
View(Twitter_eigen_format)
```

Tell R to take into account the direction of connections

Twitter: eigenvector centrality

2. Network Analysis – Mining Social... | Twitter_eigen_data_frame

Filter

	vector	value	options.bmat	options.n	options.which	options.nev	options.tol
Ted Coin	0.17735982...	5.1720...	I	11731	LR	1	0
Nasdaq	0.15840910...	5.1720...	I	11731	LR	1	0
Pam Moore	0.16230700...	5.1720...	I	11731	LR	1	0
Meghan M. Biro	0.16822100...	5.1720...	I	11731	LR	1	0
CrowdTParade	0.18735444...	5.1720...	I	11731	LR	1	0

Showing 1 to 5 of 11,731 entries

2. Network Analysis – Mining Social... | Twitter_eigen_format

Filter

	Messenger	Eigenvector
1	Ted Coin	0.1773598200
2	Nasdaq	0.1584091057
3	Pam Moore	0.1623070044
4	Meghan M. Biro	0.1682210090
5	CrowdTParade	0.1873544404

Showing 1 to 5 of 11,731 entries

Twitter: eigenvector centrality

Script

```
# Let's define eigenvector centrality based on the number of followers
# that an account has. The reasoning is that the more followers an
# account has the more it has the ability to influence public opinion.
Twitter_network_clean_reverse_followers =
                                  join(Twitter_network_clean_reverse[, c(1, 2, 4, 5, 6)],
                                       Twitter_network_degree_betweenness[, c(1, 4)],
                                       by = "Messenger")

Twitter_eigen_followers = eigen_centrality(Twitter_network_clean_directed,
                                           weights = Twitter_network_clean_reverse
                                                     $Number_Followers,
                                           directed = TRUE)

Twitter_eigen_followers_data_frame = as.data.frame(Twitter_eigen_followers)
View(Twitter_eigen_followers_data_frame)

Twitter_eigen_followers_format = cbind(rownames(Twitter_eigen_followers_data_frame),
                                       Twitter_eigen_followers_data_frame$vector)

colnames(Twitter_eigen_followers_format) = c("Messenger", "Eigenvector_Followers")
View(Twitter_eigen_followers_format)
```

1. Weight eigenvector centrality by number of followers
2. Tell R to take into account the direction of connections

Twitter: eigenvector centrality

2. Network Analysis – Mining Social... | Twitter_network_clean_reverse_foll...

Filter

	Follower	Messenger	Number_Friends	Number_Tweets	Location	Number_Followers
1	Ted Coin	Data Society	353470	88618	Naples, Florida, USA	1589
2	Nasdaq	Data Society	4007	13681	Times Square, New York	1589

Showing 1 to 2 of 26,285 entries

2. Network Analysis – Mining Social... | Twitter_eigen_followers_data_frame

Filter

	vector	value	options.bmat	options.n	options.which	options.nev	options.tol	options.ncv
Ted Coin	0.071741...	13864...	I	11731	LR	1	0	0
Nasdaq	0.080749...	13864...	I	11731	LR	1	0	0

Showing 1 to 2 of 11,731 entries

2. Network Analysis – Mining Social... | Twitter_eigen_followers_format

Filter

	Messenger	Eigenvector_Followers
1	Ted Coin	0.071741555715128
2	Nasdaq	0.0807495736165884

Showing 1 to 2 of 11,731 entries

Twitter: combine centrality metrics

Script

```
# join() can only work with data frames so make sure to cast your data that way.
Twitter_eigen_format = as.data.frame(Twitter_eigen_format)
Twitter_eigen_followers_format = as.data.frame(Twitter_eigen_followers_format)

# Check the structure of the data to understand how R is reading it.
str(Twitter_eigen_format)
Twitter_eigen_format$Messenger = as.character(Twitter_eigen_format$Messenger)
Twitter_eigen_format$Eigenvector = as.numeric(as.character(Twitter_eigen_format
                                                           $Eigenvector))

str(Twitter_eigen_followers_format)
Twitter_eigen_followers_format$Messenger = as.character(Twitter_eigen_followers_format
                                                        $Messenger)

Twitter_eigen_followers_format$Eigenvector_Followers =
  as.numeric(as.character(Twitter_eigen_followers_format$Eigenvector_Followers))
```

Twitter: combine centrality metrics

Script

```
# Combine the eigenvector centrality data with the betweenness data set.
View(Twitter_network_degree_betweenness)

library(plyr)
Twitter_network_degree_betweenness_eigen = join(Twitter_network_degree_betweenness,
                                                Twitter_eigen_format)

Twitter_network_degree_betweenness_eigen_followers =
  join(Twitter_network_degree_betweenness_eigen,
       Twitter_eigen_followers_format)

str(Twitter_network_degree_betweenness_eigen_followers)
View(Twitter_network_degree_betweenness_eigen_followers)

# Save your work.
write.csv(Twitter_network_degree_betweenness_eigen_followers,
          "new_Twitter network centrality summary.csv",
          row.names = FALSE)
```

Twitter: combine centrality metrics

2. Network Analysis – Mining Social... | Twitter_network_degree_betweenne...

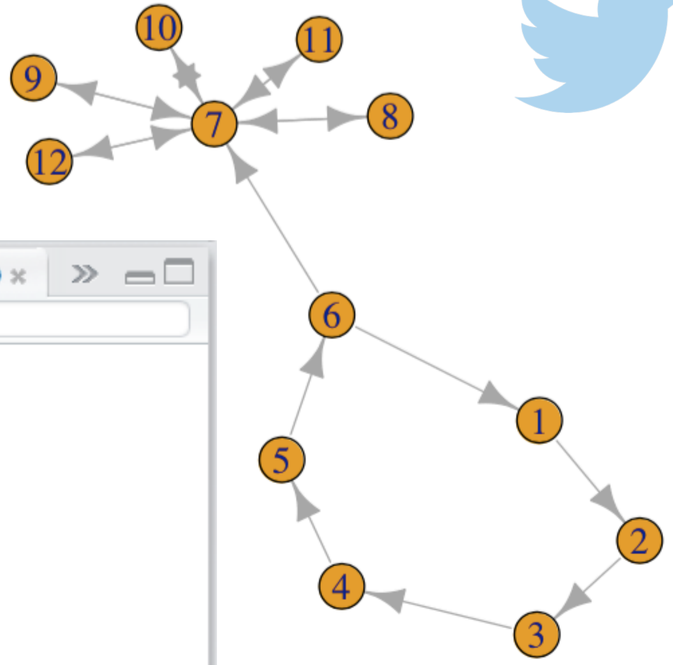
Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends	Betweenness	Eigenvector	Eigenvector_Followers
1	Data Society	0	30	1589	182	0.00000	1.000000e+00	1.368933e-03
2	Ted Coin	1	30	445076	353470	310.71015	1.773598e-01	7.174156e-02
3	Nasdaq	3	60	435305	4450	221695.48412	1.584091e-01	8.074957e-02
4	Pam Moore	6	30	248614	134658	421574.91751	1.623070e-01	5.961065e-02
5	Meghan M. Biro	2	60	111419	88371	2562.57526	1.682210e-01	3.515492e-02
6	CrowdTParade	12	60	117648	124793	454869.23719	1.873544e-01	4.431845e-02
7	Slack	5	30	120517	58558	27034.29681	2.840771e-01	1.128097e-02
8	Shelly Palmer	1	30	89052	380	412.54252	1.273188e-01	1.242976e-02
9	Sean Ellis	3	30	93533	43082	150527.58241	1.427920e-01	3.359538e-02
10	Careers In Gov	11	30	126897	93077	447891.26276	3.192034e-01	1.900727e-01

Showing 1 to 10 of 11,731 entries

Twitter: analyzing centrality

- Recall our prior example
- Node 7 not only had high eigenvector centrality, it also had high betweenness
- Nodes like #7 are the most important members of a network due to their ability to connect communities and disseminate a message
- Let's analyze our network through that lens



as.data.frame(eigen_n$vector)

Filter

	eigen_n$vector
7	1.000000e+00
9	4.472136e-01
8	4.472136e-01
12	4.472136e-01
10	4.472136e-01
11	4.472136e-01
3	2.112615e-16
5	2.060866e-16
6	1.967335e-16
2	1.772966e-16
4	1.335570e-16
1	1.002606e-16

Showing 1 to 12 of 12 entries

Twitter: analyzing centrality

- First, sort your data by betweenness, then by eigenvector centrality based on the number of followers

- These people in our Twitter network are the most important gatekeepers that connect us to most prominent accounts

2. Network Analysis – Mining Social... | Twitter_network_degree_betweenne...

Filter

	Messenger	In-degree	Out-degree	Number_Followers	Number_Friends	Betweenness	Eigenvector	Eigenvector_Followers
2...	Ali Spagnola	11	30	2033301	1042756	41346.34687	0.008402313	1.00000000
5...	MarQuis Trill	45	30	3614698	3405604	382020.54638	0.037700191	0.65968924
5...	فنان بمشاعري	3	30	397986	328841	36671.71623	0.036500882	0.47359798
7...	RainApp Social	1	30	495756	16	454.36226	0.032602407	0.38912342
2...	p0t.com™	5	60	856754	371014	98500.77418	0.132585976	0.35218394
3...	airfarewatchdog	4	30	561431	63455	17173.82503	0.016356553	0.30054668
2...	АРТЕМ КЛЮШИН □	29	88	1432612	614545	434928.34685	0.071231739	0.29693488
2...	#DeepLearning #App □	11	120	592607	268205	252956.63004	0.077405545	0.29567912
3...	陶芸家 佐々木弘吉 HiroSas...	8	30	565669	565243	48457.10583	0.007289239	0.26916130
5...	حجازي	4	30	394901	373888	111417.31330	0.064895285	0.25806970
5...	Ryan	5	30	530140	162347	20928.19663	0.007289239	0.25225559
2...	Intelligence TV □	5	120	278466	99717	133443.60678	0.178111935	0.25180526

Showing 1 to 13 of 11,731 entries

Visualize most important connections

Script

```
# Let's plot the network using the original node plus the top accounts measured
# first by follower eigenvector centrality, then by betweenness.
Twitter_network_top_sorted_1 =
  Twitter_network_degree_betweenness_eigen_followers[
    order(Twitter_network_degree_betweenness_eigen_followers$Betweenness,
          decreasing = TRUE),]

Twitter_network_top_sorted =
  Twitter_network_top_sorted_1[
    order(Twitter_network_top_sorted_1$Eigenvector_Followers,
                                  decreasing = TRUE),]
View(Twitter_network_top_sorted)

# Subset the top 20 names plus the original node.
View(Twitter_network_clean)
Twitter_top_names = c(as.character(Twitter_network_top_sorted$Messenger[1:17]),
                                     "Data Society")
Twitter_network_top = subset(Twitter_network_clean,
                             Twitter_network_clean$Messenger %in% c(Twitter_top_names) |
                               Twitter_network_clean$Follower %in% c(Twitter_top_names))
```

Since you're looking for top Twitter names in Messenger or Follower, use "%in%" to search the columns with "|" in between, which means "or"

Visualize most important connections

Script

```
View(Twitter_network_top)

# Save your data set.
write.csv(Twitter_network_top,
          "new_Twitter network_top_20_bwn_eigen.csv",
          row.names = FALSE)
```

2. Network Analysis - Mining Socia... | Twitter_network_top

Filter

	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location
1	Data Society	Ted Coin	445076	353470	88618	Naples, Florida, USA
2	Data Society	Nasdaq	419320	4007	13681	Times Square, New York
3	Data Society	Pam Moore	240953	134103	104450	Orlando, FL
4	Data Society	Meghan M Biro	106098	84919	174780	Cambridge, MA
5	Data Society	CrowdTParade	100602	98477	952	Worldwide
6	Data Society	Slack	93813	46586	47735	SF, Vancouver & Dublin
7	Data Society	Shelly Palmer	89052	380	38980	New York, NY
8	Data Society	Sean Ellis	86735	54971	15414	Newport Beach / San Francisco
9	Data Society	Careers In Gov	81869	73610	39362	
10	Data Society	TeesAsian	70876	67204	2822	Worldwide

Showing 1 to 10 of 1,176 entries

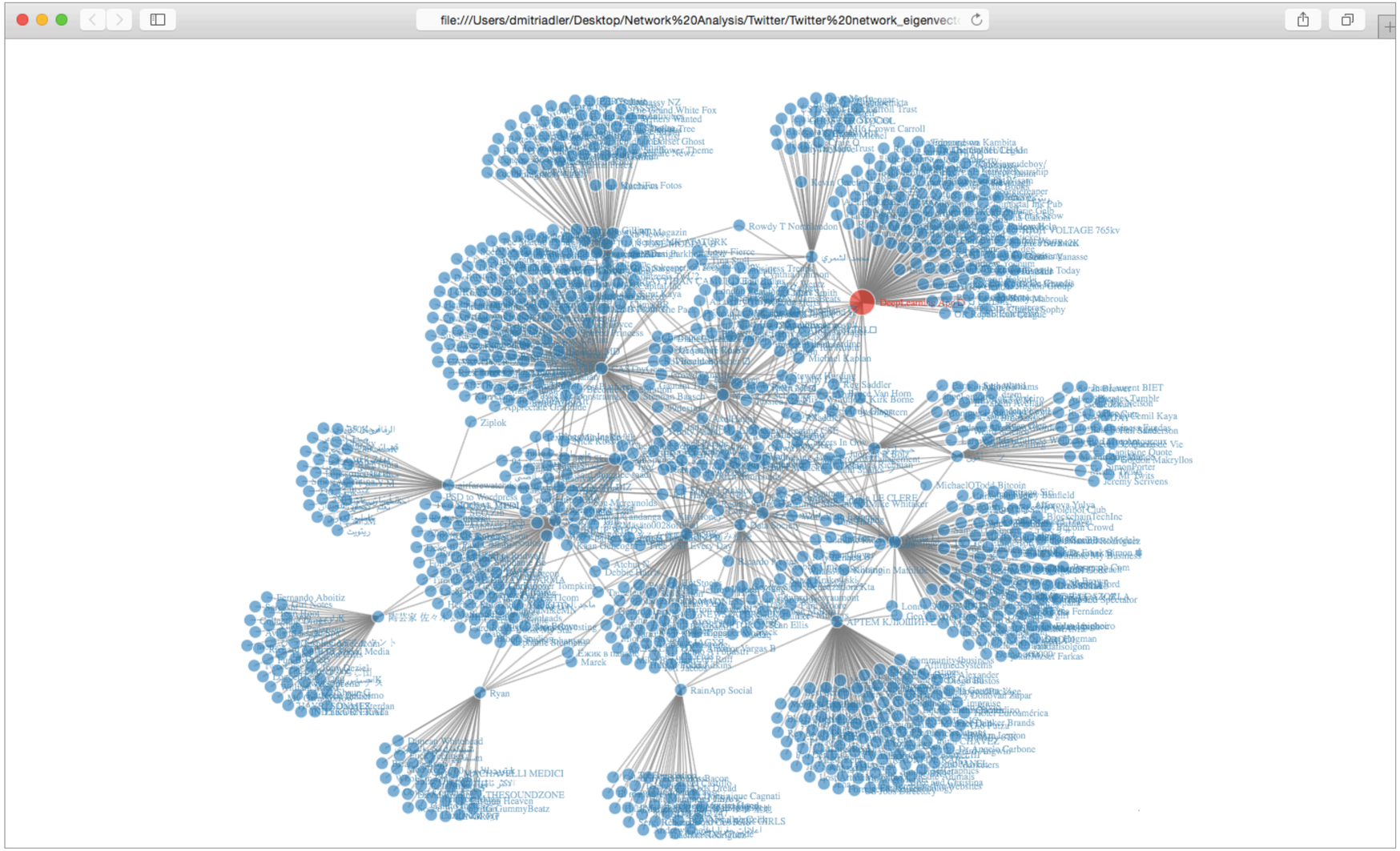
Visualize most important connections

Script

```
# Let's create another type of interactive plot using the networkD3
# package that we worked with earlier.
library(networkD3)

# Subset the 2 columns that represent the connections in the network.
Twitter_simpleNetwork = Twitter_network_top[, 1:2]

# Make sure that your data set is a data frame with data formatted as characters.
str(Twitter_simpleNetwork)

# Dynamic plots render faster in a web browser so let's save your
# output as an html file. This will also allow you to share it with others.
library(magrittr)
simpleNetwork(Twitter_simpleNetwork,
              charge = -100,
              fontSize = 12,
              textColour = "#3182bd",
              linkColour = "grey",
              nodeColour = "#3182bd",
              nodeClickColour = "red",
              opacity = 0.6) %>% saveNetwork(file = "Twitter network_eigenvector.html")
```

1. `charge` sets how far apart the nodes are
2. `fontSize` sets the font size
3. `nodeClickColour` determines what color the node turns once you click on it

Visualize most important connections



What do networks represent?

1. Organizational relationships
2. Communications patterns
3. Economic relationships
4. Environmental relationships
5. Connections based on interests, preferences, and similarities
6. Geographic relationships

Modeling network effects

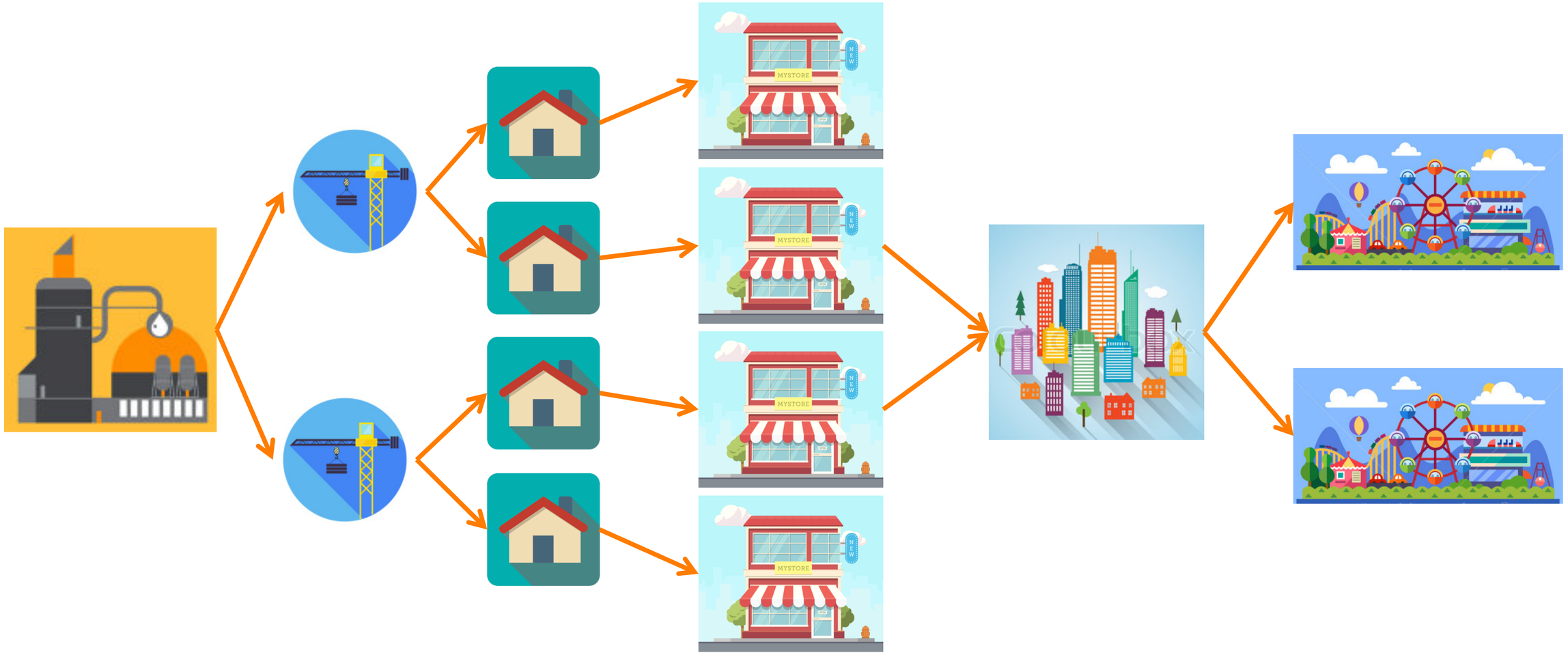
- Now that you're starting to think about how everything is connected, let's model how one event can impact the rest of the network

- What happens to one node can happen to other nodes in the network

- This is how:
  1. Diseases spread
  2. Messages spread
  3. Economic boom and bust cycles work
  4. Information technology and internet networks collapse
  5. Effects of cancer drugs on cell function are modeled
  6. Ecosystems grow or shrink

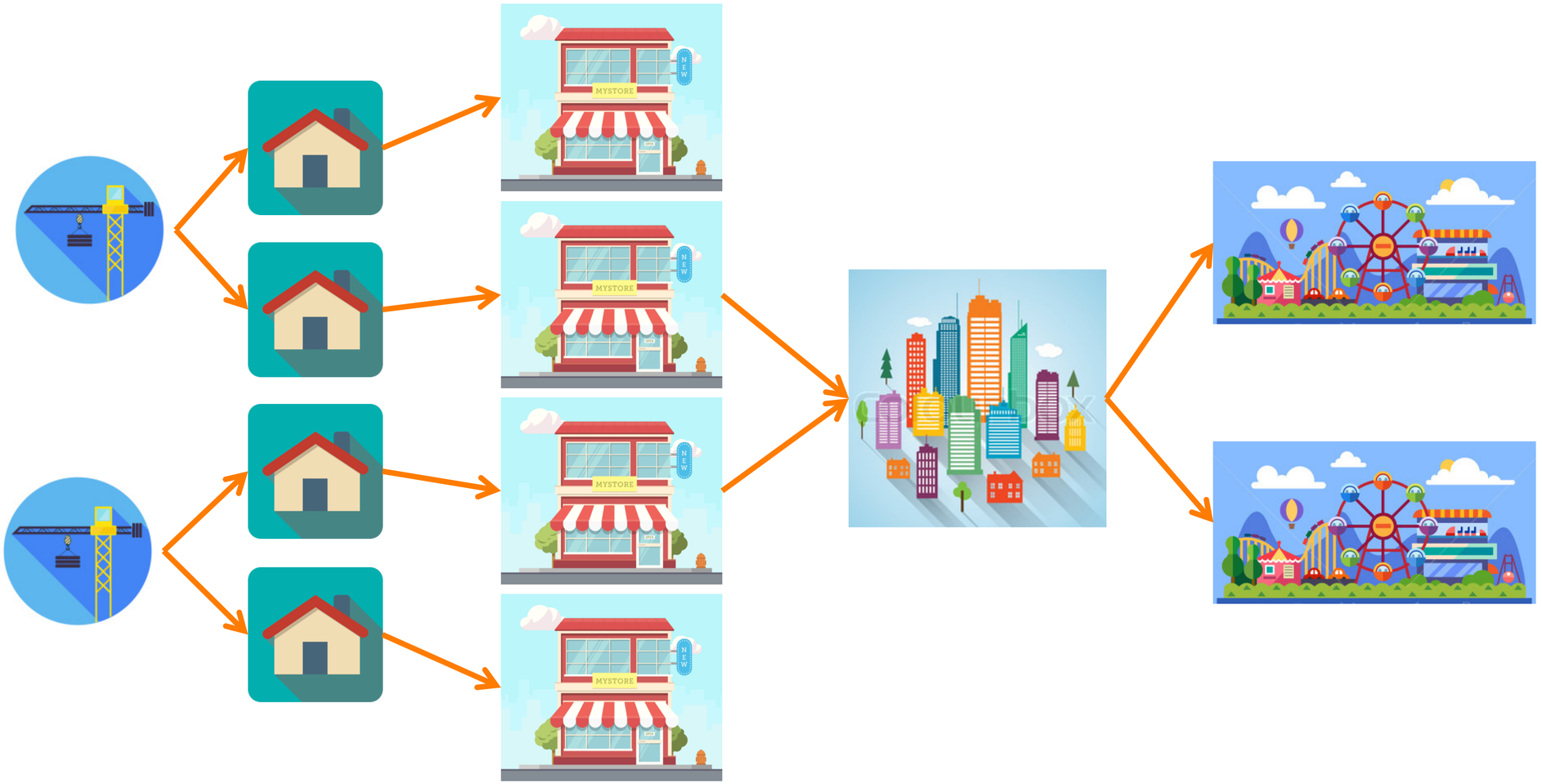
Application: cascading failures

- A factory that fuels growth in a city shuts down, what happens?



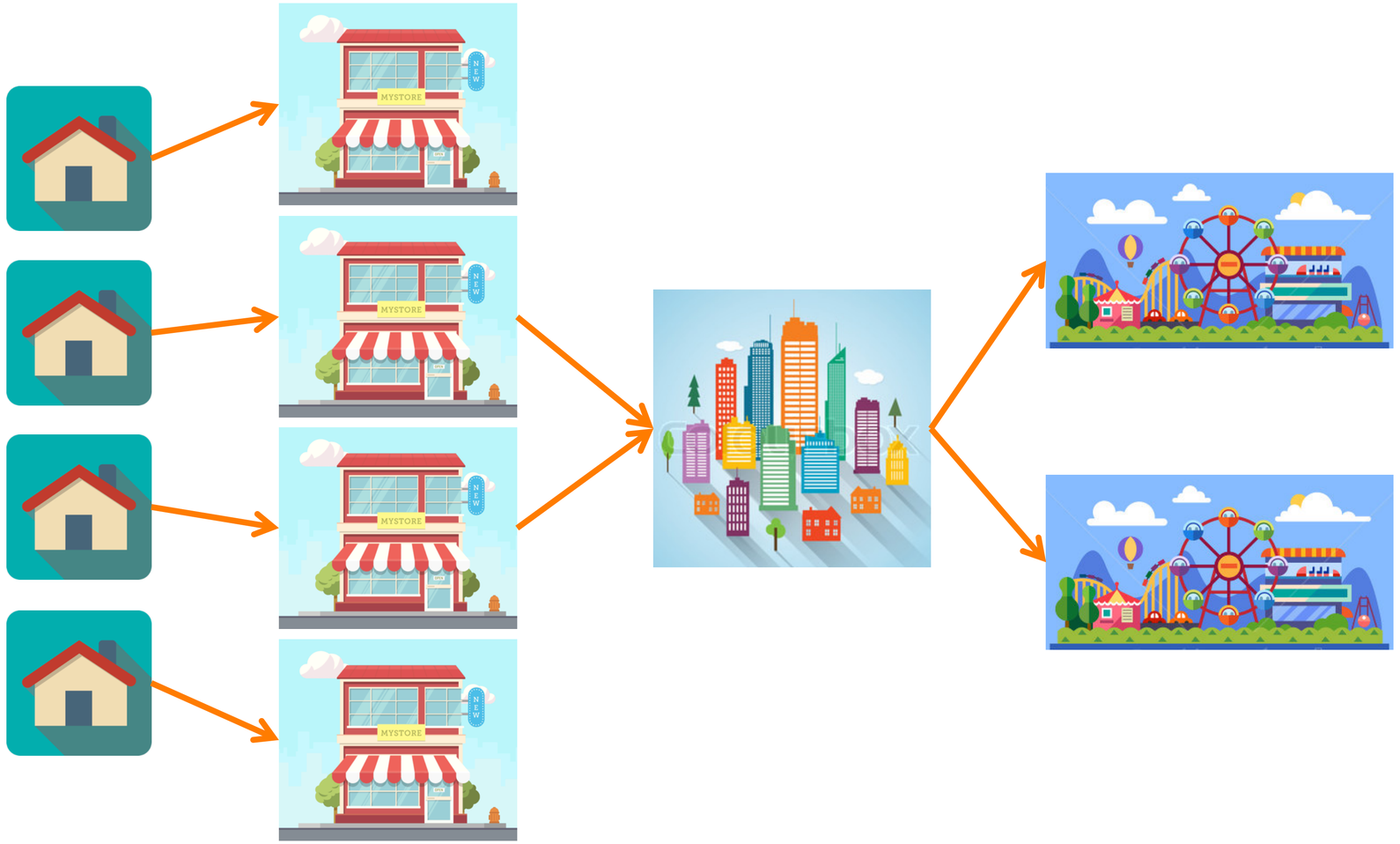
Application: cascading failures

- Workers and construction is affected



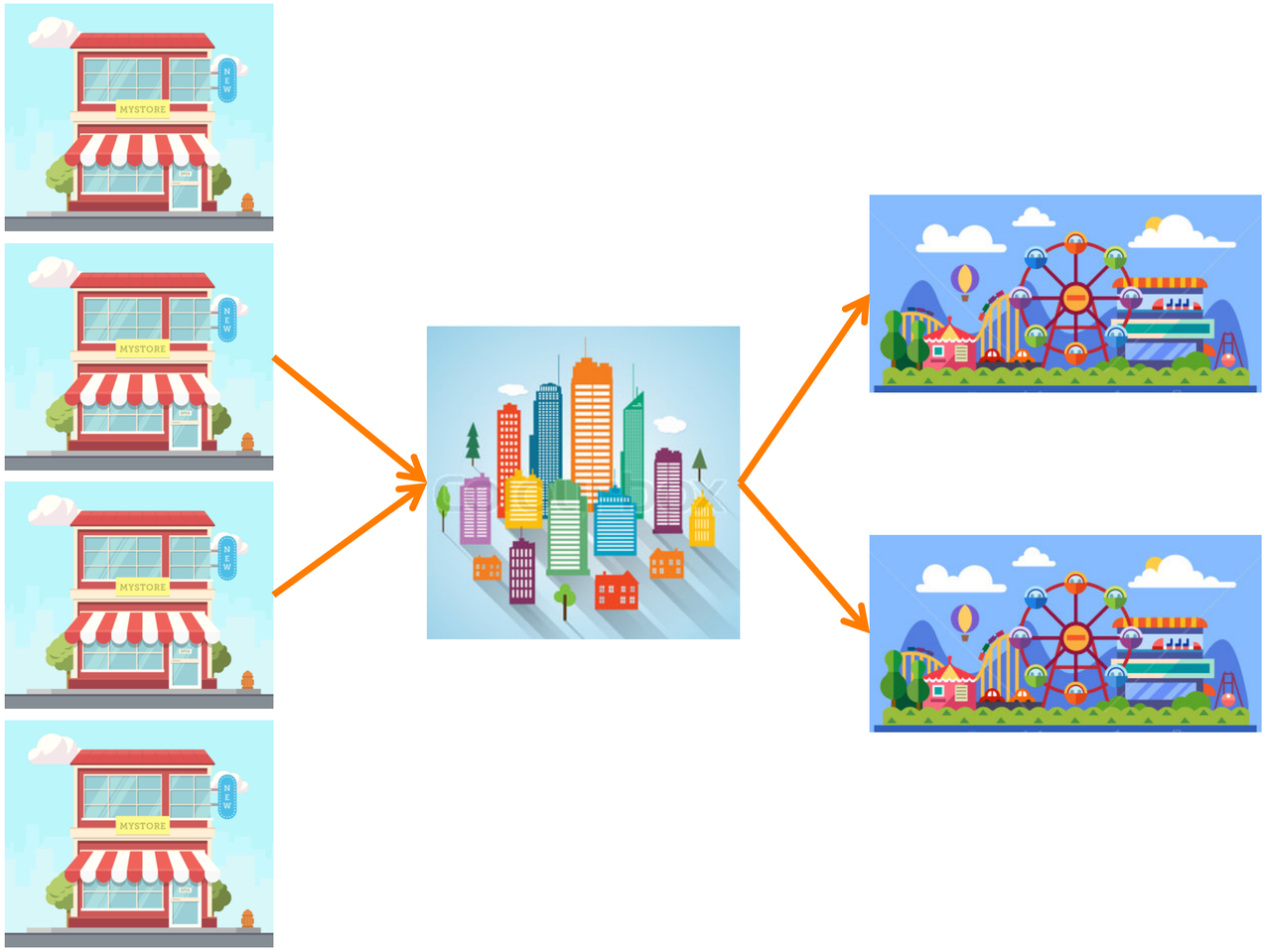
Application: cascading failures

- People leave and homes go empty, decreasing property values



Application: cascading failures

- With fewer people, stores shut down

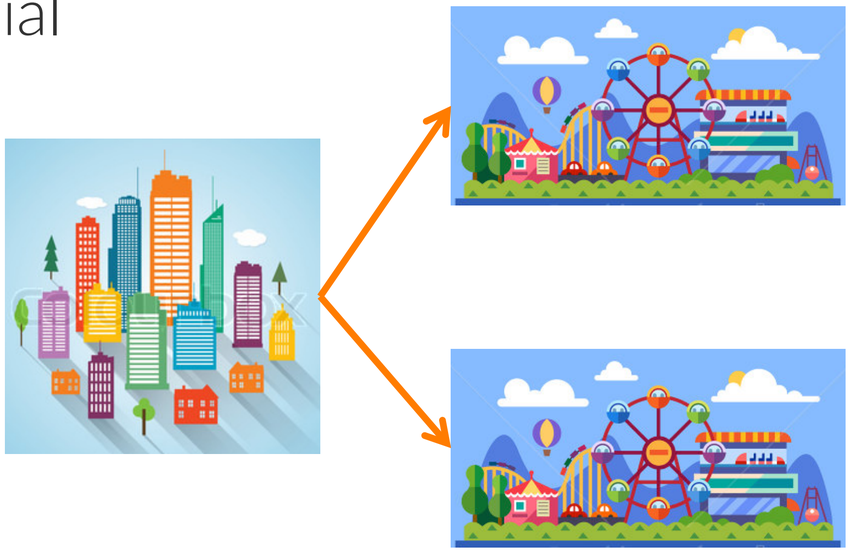


Application: cascading failures

- Entire cities decline: this is a process similar to what happened in Detroit, MI as a lot of car manufacturing jobs left or became automated

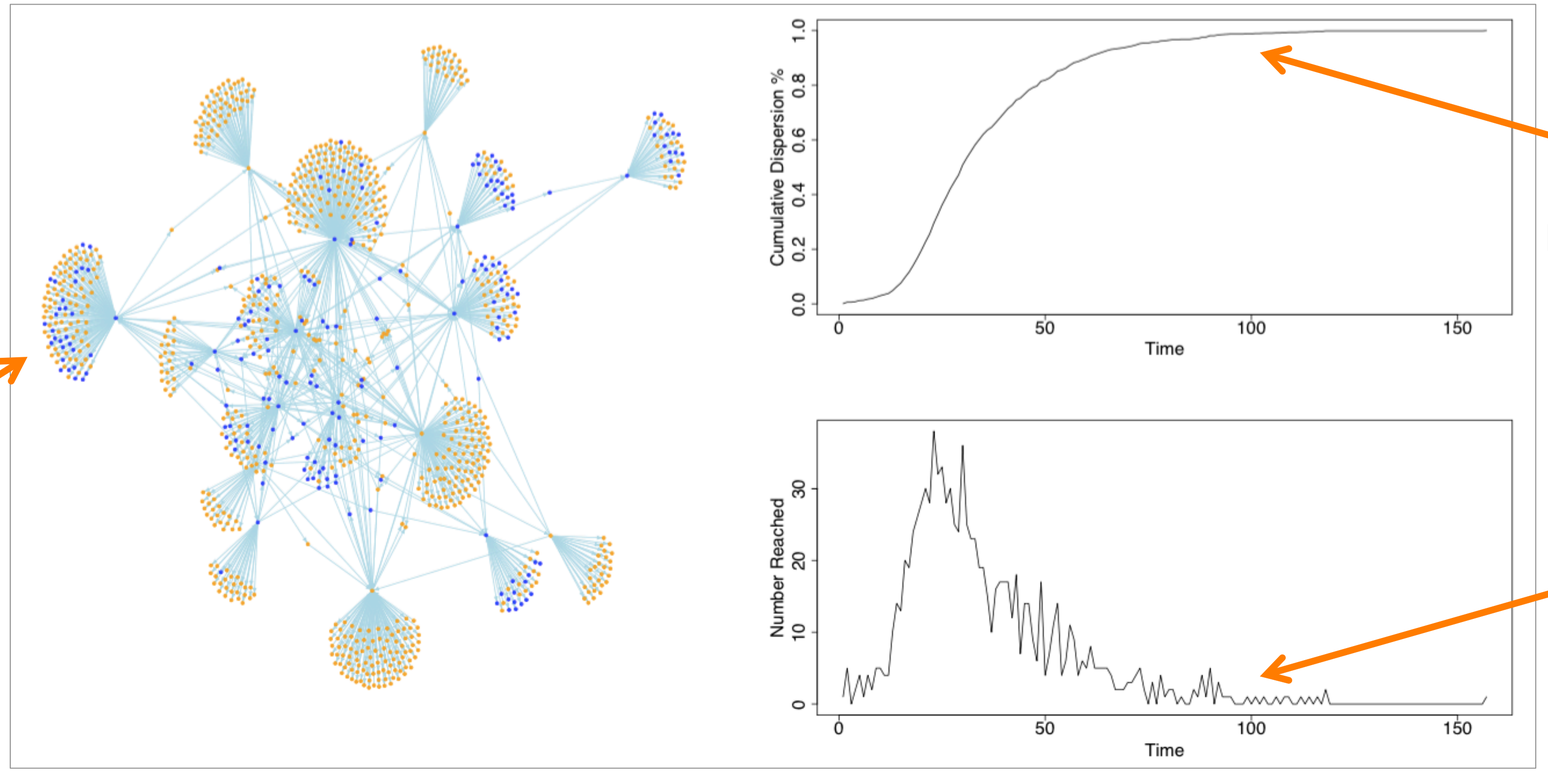
- Cascading economic failures is how a lot of financial crises spread such as the cascading failures of financial institutions in the U.S. in 2008 and 2009

- The Asian economic crisis of 1997 experienced similar events



Application: message propagation

- We will model how a message spreads through a community of people and create this video
- At each step, every affected node will broadcast a message



Affected nodes change color

Shows the percentage of the network that has received the message at each step

Shows the number of nodes affected at each step

Message propagation: framework

- Let's use the spread of disease as a conceptual framework

- There are 3 or 4 states that someone can have:
  1. Susceptible to infection
  2. Infected
  3. Recovered or removed – immune to future infection
  4. Susceptible to infection – again

- Defining these states is what allows you to create a dynamic propagation model

- You need to make assumptions about people in each state

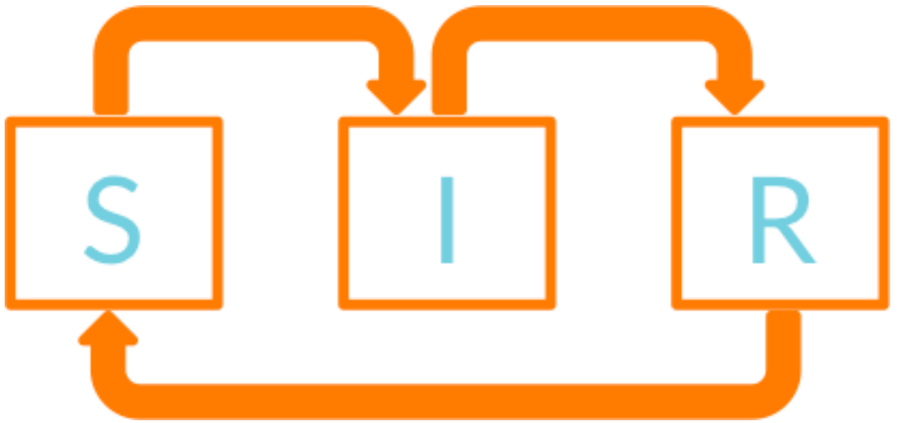
Message propagation: framework

- There are 3 or 4 states that someone can have:
  1. Susceptible to infection
  2. Infected
  3. Recovered
  4. Susceptible to infection – again

- Questions:
  1. What is the probability of infection for someone who is susceptible?
  2. What is the duration of infection?
  3. Why does someone recover?
  4. Why does someone become susceptible to disease again?

Message propagation: framework

- There are 3 or 4 states that someone can have:
  1. Susceptible to infection
  2. Infected
  3. Recovered
  4. Susceptible to infection – again



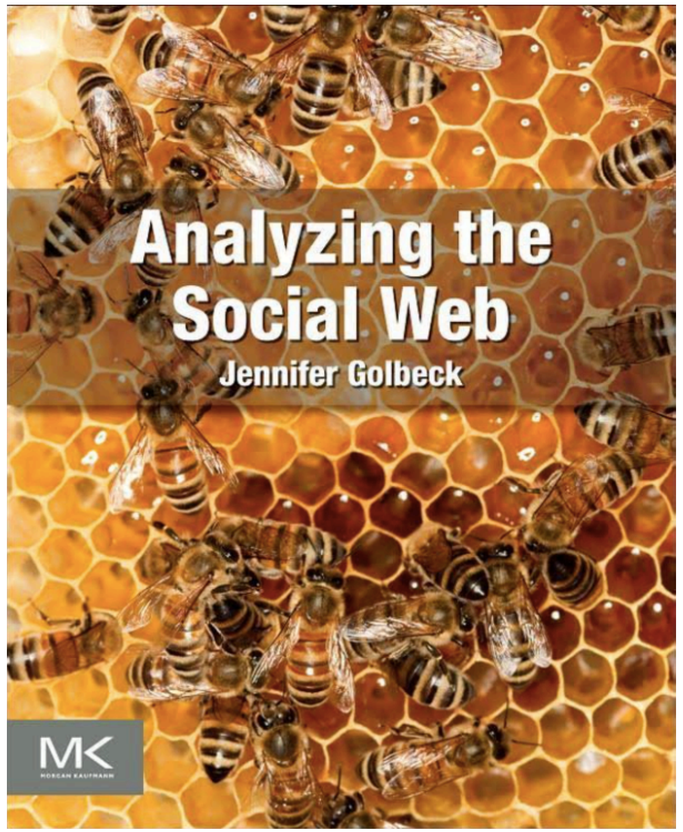
- Here are a few examples:
  1. HIV: a person is susceptible and is then forever infected. This is an SI model.
  2. Chickenpox: a person is susceptible, can recover and then is never at risk again. This is an SIR model.
  3. Common cold: someone can catch a cold, recover, enjoy a period of immunity and then be susceptible to another cold. This is an SIRS model

Message propagation: framework

- There are 3 or 4 states that someone can have:
  1. Susceptible to infection
  2. Infected
  3. Recovered
  4. Susceptible to infection – again

- Infection can depend on:
  1. The severity of the disease
  2. Duration of the disease
  3. The number of adjacent nodes infected
  4. Probability of infection given exposure
  5. Etc.

Recommended reading



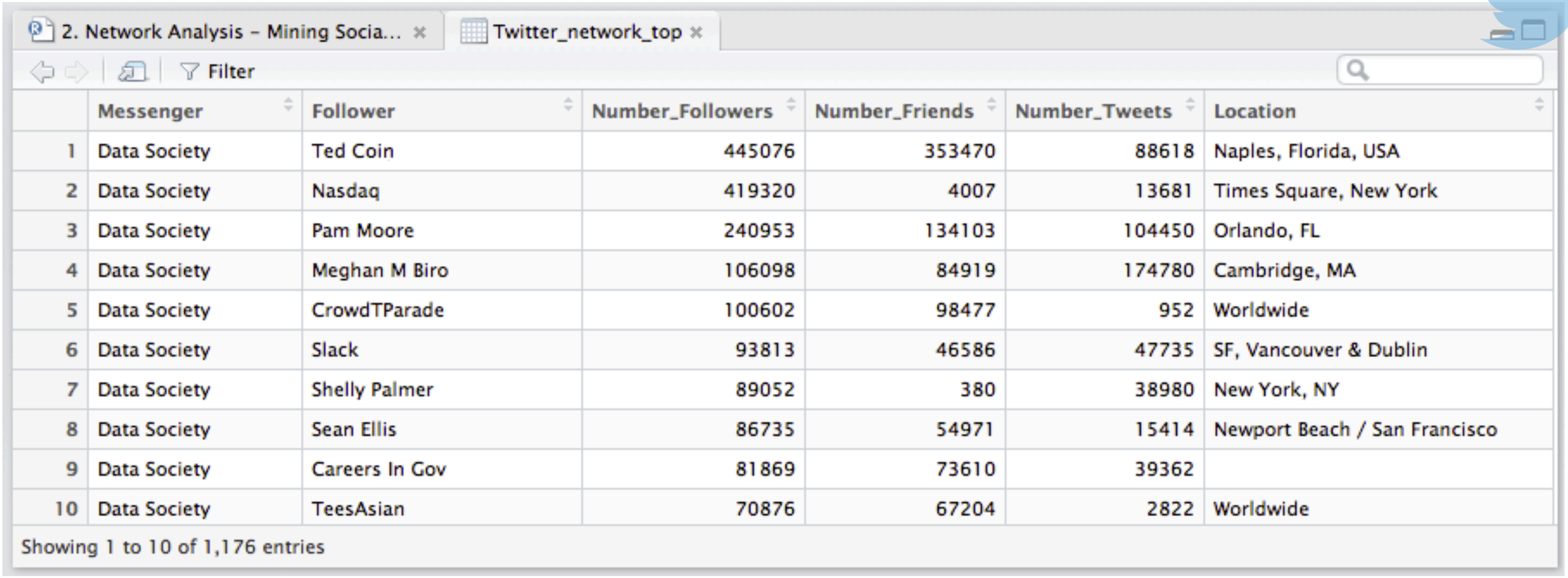
Twitter: network diffusion in R

- Now that we've measured our Twitter network and understand who the key actors are, let's see how information can spread if we target those people with good, thoughtful and helpful messages

Script

```
# Load the igraph library.
library(igraph)

# Load the Twitter network data you saved earlier.
Twitter_network_top = read.csv("Twitter network_top_20_bwn_eigen.csv",
                               check.names = FALSE)
View(Twitter_network_top)

str(Twitter_network_top)
Twitter_network_top$Messenger = as.character(Twitter_network_top$Messenger)
Twitter_network_top$Follower = as.character(Twitter_network_top$Follower)
Twitter_network_top$Location = as.character(Twitter_network_top$Location)
```

Load data with most important people



2. Network Analysis – Mining Socia... | Twitter_network_top

Filter

	Messenger	Follower	Number_Followers	Number_Friends	Number_Tweets	Location
1	Data Society	Ted Coin	445076	353470	88618	Naples, Florida, USA
2	Data Society	Nasdaq	419320	4007	13681	Times Square, New York
3	Data Society	Pam Moore	240953	134103	104450	Orlando, FL
4	Data Society	Meghan M Biro	106098	84919	174780	Cambridge, MA
5	Data Society	CrowdTParade	100602	98477	952	Worldwide
6	Data Society	Slack	93813	46586	47735	SF, Vancouver & Dublin
7	Data Society	Shelly Palmer	89052	380	38980	New York, NY
8	Data Society	Sean Ellis	86735	54971	15414	Newport Beach / San Francisco
9	Data Society	Careers In Gov	81869	73610	39362	
10	Data Society	TeesAsian	70876	67204	2822	Worldwide

Showing 1 to 10 of 1,176 entries

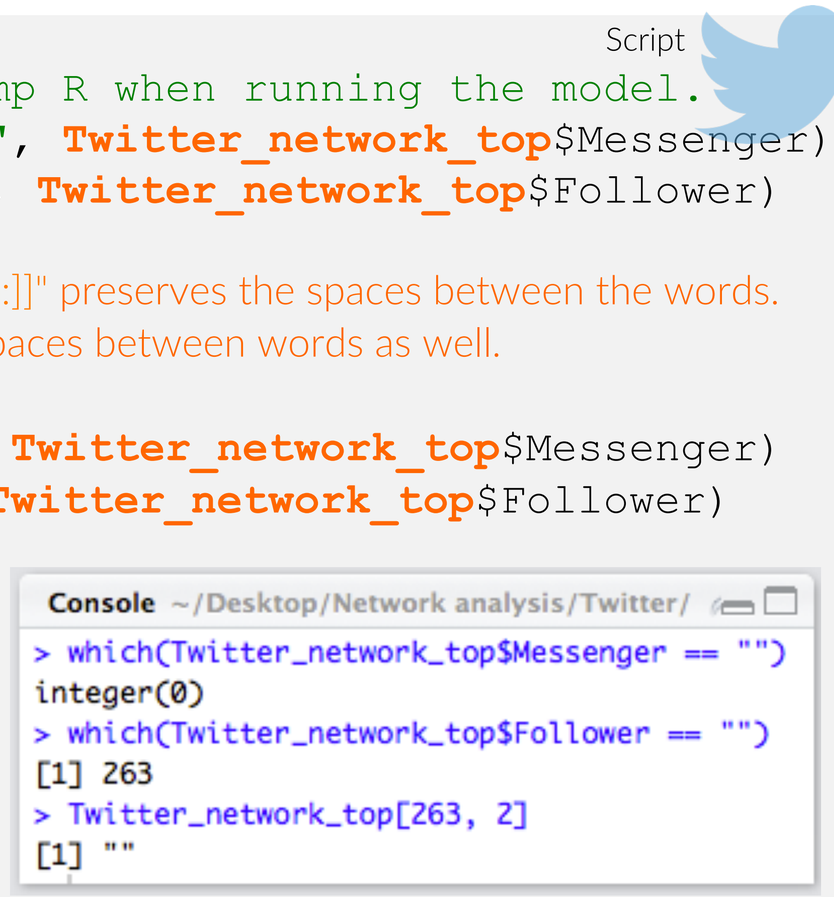
Clean node names

Script

```
# Clean the names so graphics and punctuation don't stump R when running the model.
Twitter_network_top$Messenger = gsub("[^ [:graph:]]", "", Twitter_network_top$Messenger)
Twitter_network_top$Follower = gsub("[^ [:graph:]]", "", Twitter_network_top$Follower)
```

The space after the carat in "[^ [:graph:]]" preserves the spaces between the words. Removing that space would remove spaces between words as well.

```
Twitter_network_top$Messenger = gsub("[[:punct:]]", "", Twitter_network_top$Messenger)
Twitter_network_top$Follower = gsub("[[:punct:]]", "", Twitter_network_top$Follower)

# Check if any names were rendered blank.
which(Twitter_network_top$Messenger == "")
which(Twitter_network_top$Follower == "")

# Name nodes with a blank name with terms that specify
# that you named them, such as [BLANK] or [BLANK_2].
Twitter_network_top[263, 2]
Twitter_network_top[263, 2] = "[BLANK]"

Twitter_network_simulation = Twitter_network_top[, 1:2]
str(Twitter_network_simulation)
```



Generate graph object

Script

```
# Generate a graph object.
Twitter_network_simulation_1_graph = graph.data.frame(Twitter_network_simulation,
                                                      directed = TRUE)

# Remove any duplicate connections.
Twitter_network_simulation_1_graph_simple = simplify(Twitter_network_simulation_1_graph,
                                                     edge.attr.comb = "ignore")
```

Ignore the edge weights when eliminating duplicate edges

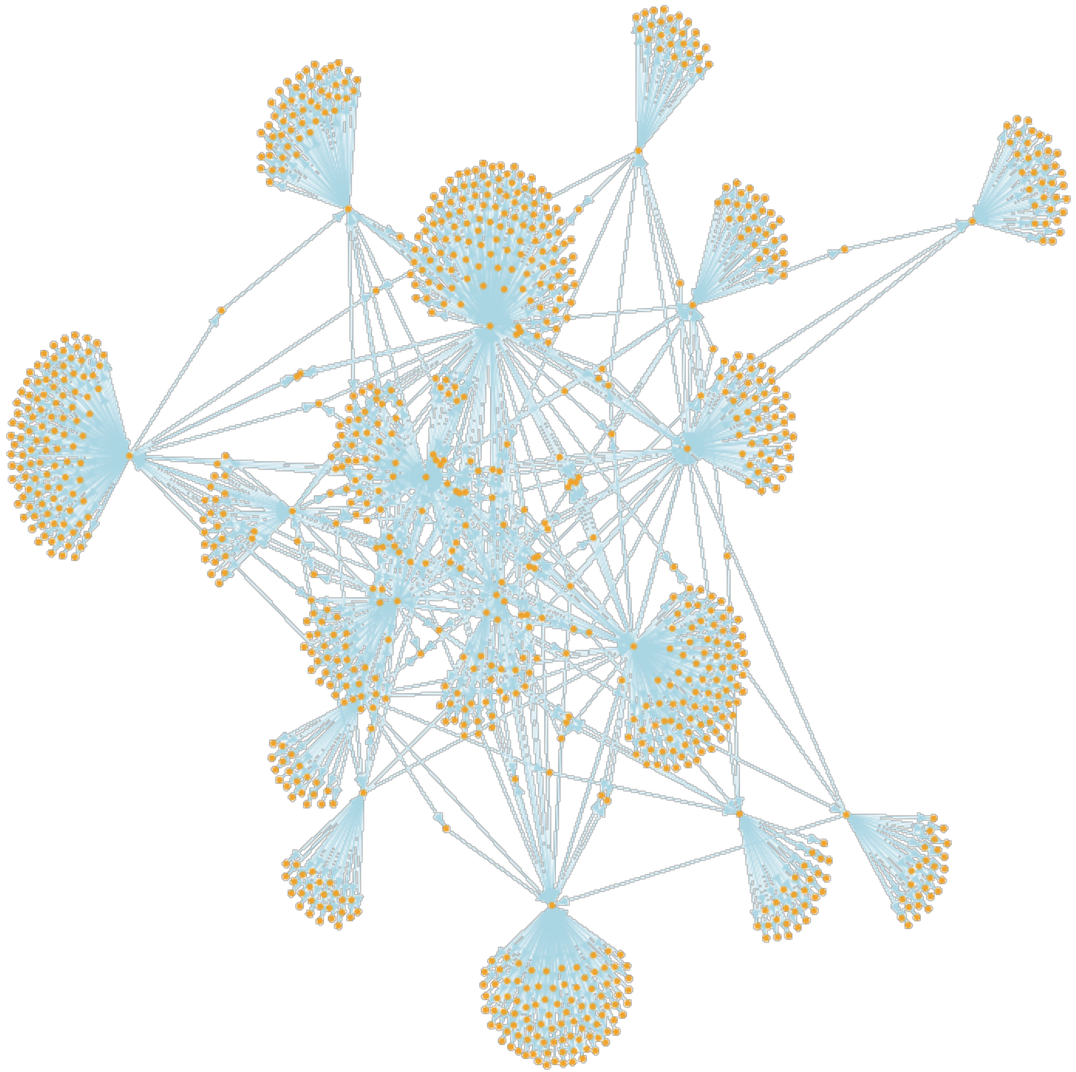
Plot the graph we'll use for the model

Script

```
# We will save directly to pdf without displaying the graph in R.
# To do that, run the pdf() command before running the plot() command.
pdf("Twitter network simulation 1.pdf",
    width = 20,
    height = 20)

# Set the seed to make the graph reproducible and assign the graph to a variable
set.seed(2)
Twitter_graph = plot(Twitter_network_simulation_1_graph_simple,
                     vertex.label = NA,
                     vertex.label.cex = 0.5,
                     vertex.color = "orange",
                     vertex.frame.color = "orange",
                     vertex.size = 1,
                     edge.color = "light blue",
                     edge.width = 0.5,
                     edge.arrow.size = 1)

# Tell R to complete the saving process.
dev.off()
```

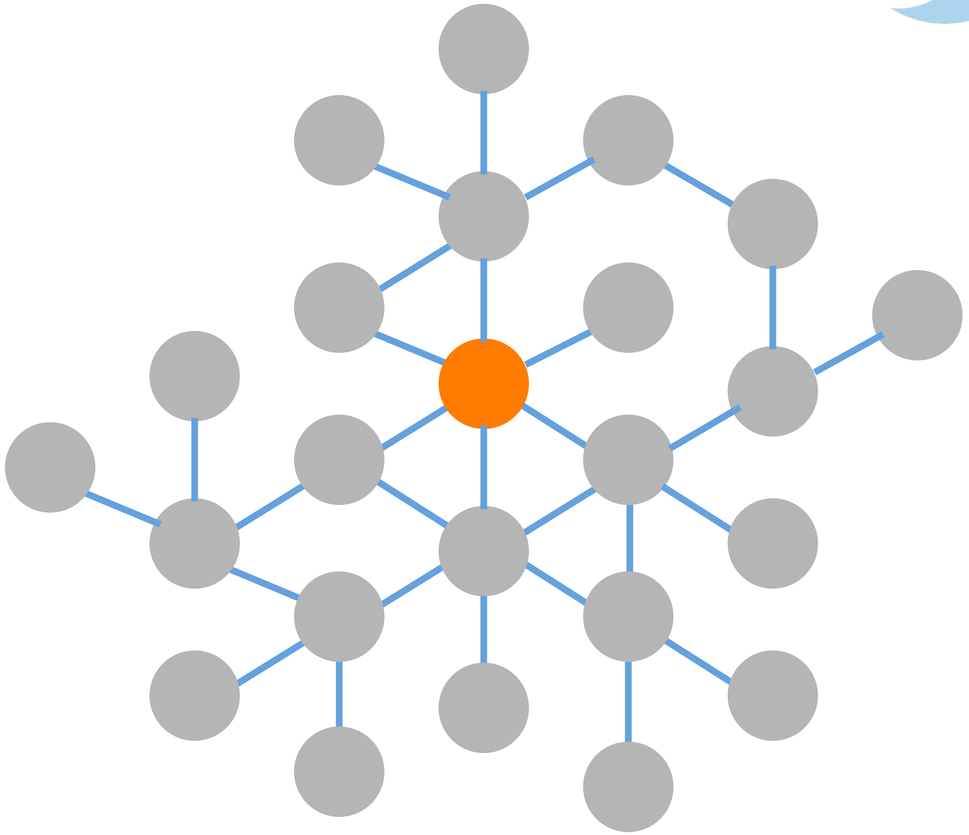
1. Exclude labels
2. Sets the size of the labels
3. Set the fill color of the points
4. Set the color of the outline of the points
5. Set the size of the points
6. Set the color of the connecting lines
7. Set the thickness of the connecting lines
8. Set the size of the arrows at the ends of the connecting lines

Plot the graph we'll use for the model



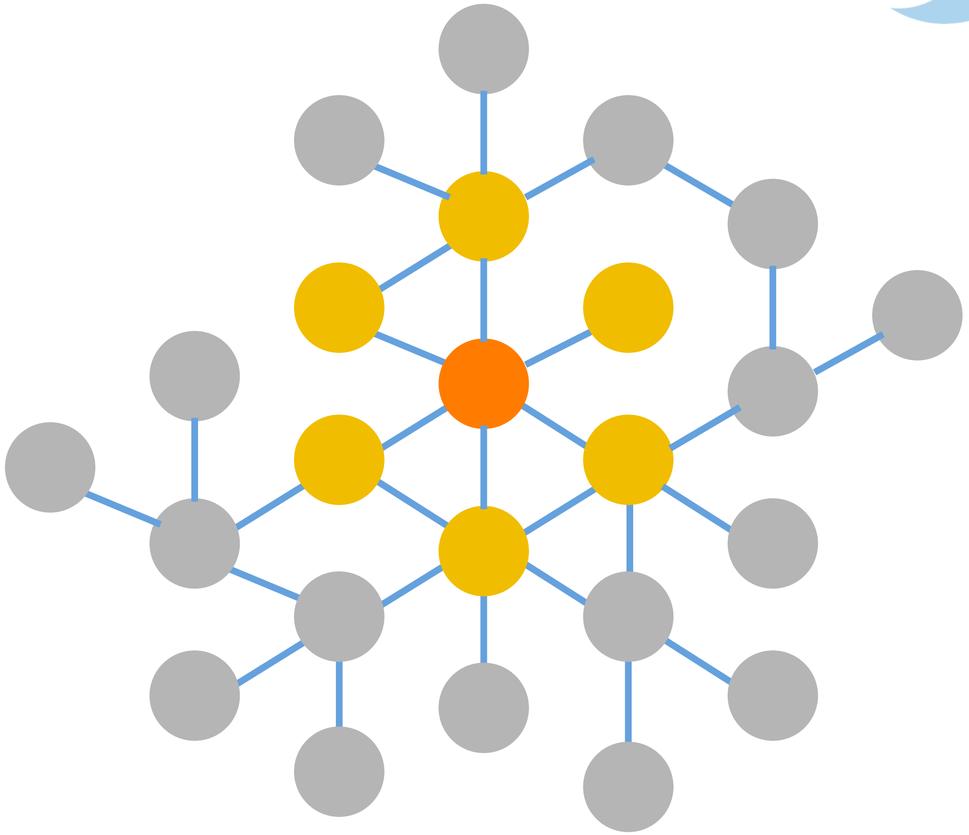
Intuition: message diffusion

1. Identify the point where the message originates from



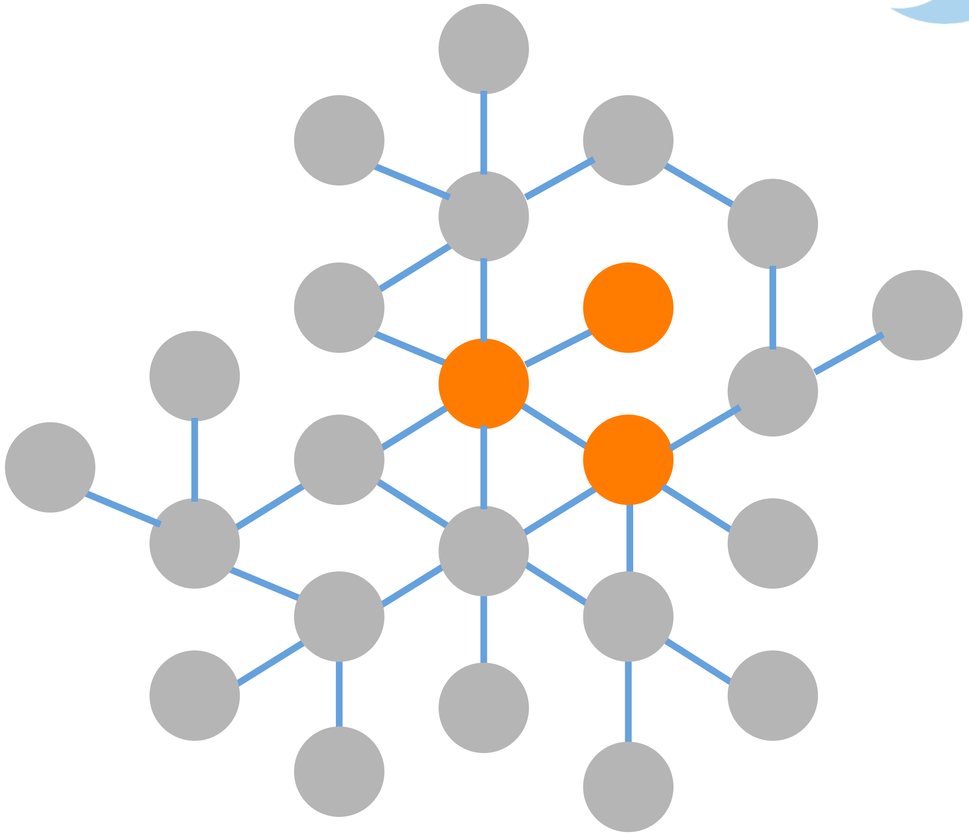
Intuition: message diffusion

1. Identify the point where the message originates from
2. Find the neighbors (direct connections) of original point



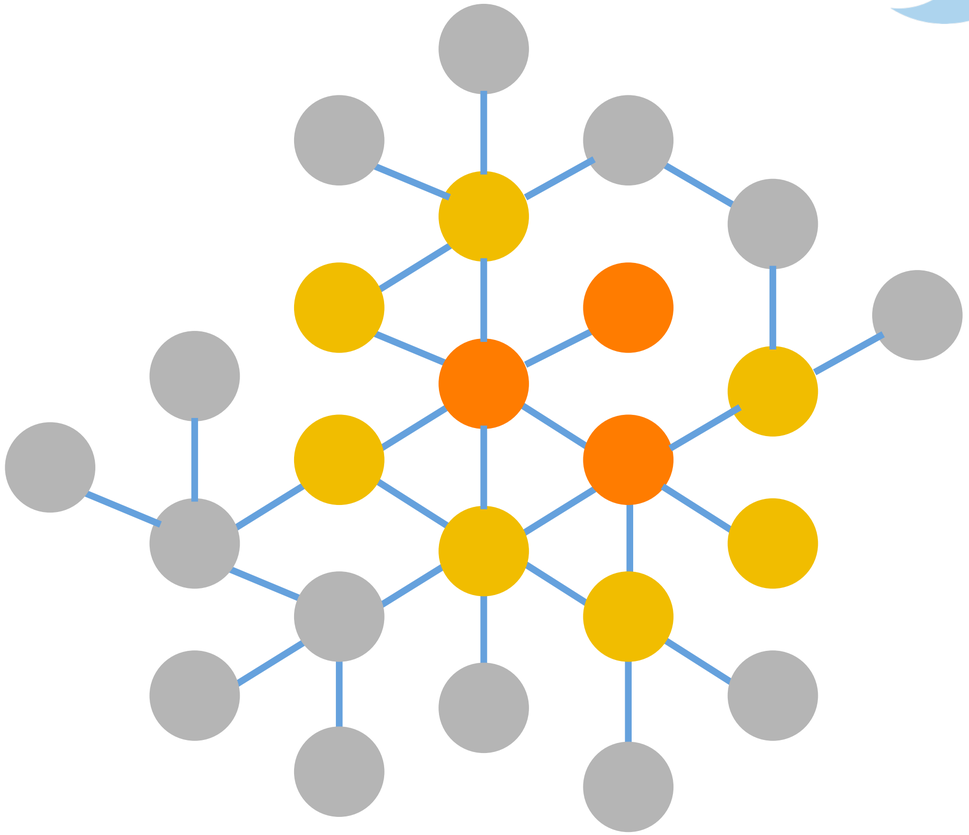
Intuition: message diffusion

1. Identify the point where the message originates from
2. Find the neighbors (direct connections) of original point
3. Run the algorithm that uses the probability/parameters to determine which (if any) neighbors receive the message



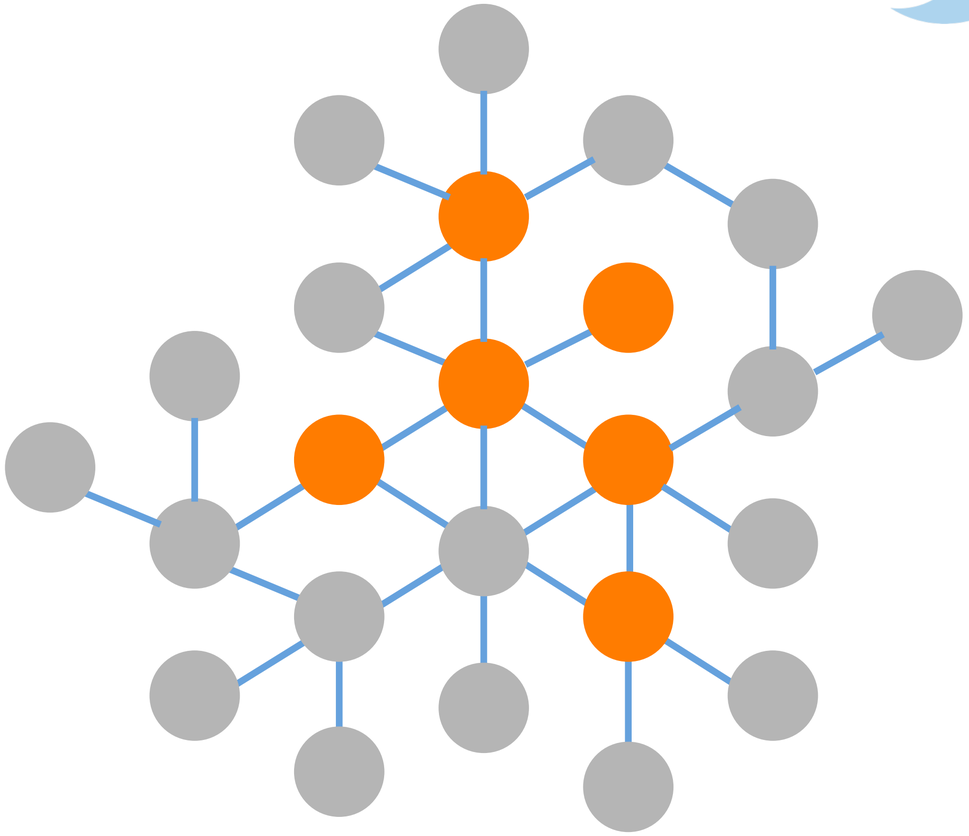
Intuition: message diffusion

1. Identify the point where the message originates from
2. Find the neighbors (direct connections) of original point
3. Run the algorithm that uses the probability/parameters to determine which (if any) neighbors receive the message
4. Recalculate the algorithm on the neighbors of the points who received the message



Intuition: message diffusion

1. Identify the point where the message originates from
2. Find the neighbors (direct connections) of original point
3. Run the algorithm that uses the probability/parameters to determine which (if any) neighbors receive the message
4. Recalculate the algorithm on the neighbors of the points who received the message
5. Continue until the parameters you set have been met



Create a function for message diffusion

Script

```
# First, create an empty list that we'll populate with lists of
# infected nodes at every stage. The final output will be a list of lists.
reached = list()

# Insert the node you'd like to start with as the first point in your simulation.
reached[[1]] = V(Twitter_network_simulation_1_graph_simple)[["Data Society"]]
reached

# Create a function that will accomplish the following 9 steps:

# 1. Select the immediate nodes connected to the starting node(s) note that the ego()
#     function cannot work with lists so you'll need to use the unlist() function.
nearest_neighbors_1 = ego(Twitter_network_simulation_1_graph_simple,
                          1,
                          reached[[1]])
nearest_neighbors_1
```

The `V()` function tells R to subset certain vertices from a graph. The `[[ ]]` tells R which item to use. A node or vertex can either be specified by name or by its number

Create a function for message diffusion

Script

```
# 2. Convert the list of neighbors you just created into a table where one
#    column is the names of the nodes and the second column only contains 1s.
#    This column will be used to select which nodes receive the message.
nearest_neighbors_2 = data.frame(names(unlist(nearest_neighbors_1)))
nearest_neighbors_2

nearest_neighbors_3 = data.frame(cbind(nearest_neighbors_2, rep(1)))
nearest_neighbors_3

# 3. Remove the original node(s) from the data set you just created.
reached[[1]]
str(nearest_neighbors_3)

# Convert the 1st column into characters.
# Convert the 2nd column into numbers.
nearest_neighbors_3[, 1] = as.character(nearest_neighbors_3[, 1])
nearest_neighbors_3[, 2] = as.numeric(as.character(nearest_neighbors_3[, 2]))
```

1. Look at the list of the original nodes
2. Check the structure of the data set you created

Set assumptions for message diffusion

Script

```
# The "!" means not, the "%in%" means to use the set of items included in the term
# after the symbol.
nearest_neighbors_4 = subset(nearest_neighbors_3,
                             !(nearest_neighbors_3[, 1] %in% names(reached[[1]])))
nearest_neighbors_4

# Now you have a list of nodes except for the original node that broadcast the message.
# Let's say there is a 5% chance the person connected to a node receives a message.
p = 0.05

# Create a random sample list of 100 items with 5 "1"s that denote when the message
# transmission occurs and 95 "0"s when transmission does not occur (i.e. 5% chance).
random_set = c(rep(1, p * 100),
               rep(0, (1 - p) * 100))
View(as.data.frame(random_set))
sum(random_set)
```

1. `random_set` will be the master file that our simulation references when deciding if a node received the message or not
2. Check to make sure the calculation is correct. The total should be 5 or 5% of 100
3. Create a variable to denote the number of instances in `random_set`

Create a function for message diffusion

Script

```
# 4. Create a function to select the nodes that will receive the message.
#    Note that you can create different models that choose nodes based on their
#    attributes of importance, location, etc.
select_nodes = function(input_list){
  random_trial = NULL
  for(i in 1:input_list){
    random_trial[i] = sample(random_set,
                             1,
                             replace = TRUE)
  }

  return(random_trial)
}

# 5. Use the lapply() function to select the rows that you will keep.
#    Note that this will be a different, random output every time.
keep_nodes = unlist(lapply(nearest_neighbors_4[, 2],
                           select_nodes))
keep_nodes
```

1. Create a blank variable
2. Create a loop to run the random variable
3. Take a sample from the data set of random events we created earlier
4. Pick 1 value each time
5. Replace tells you to replace the value after it's been sampled so that it can be sampled again
6. Produce the output of the `random_trial`

7. Take the 2nd column where we included just 1s
8. Apply the node selection function to them to select the nodes that will be affected

Create a function for message diffusion

Script

```
# 6. Get the name(s) of the node(s) that you will keep.
#    The nodes for which our select_nodes function produced a 1.
new_messengers = as.character(nearest_neighbors_4[, 1][keep_nodes >= 1])
new_messengers

# 7. Combine nodes into the complete list of nodes that have received the
#    message up to this point in the simulation.
new_nodes = unique(c(as.character(names(reached[[1]])),
                     new_messengers))

# 8. Select all the affected nodes from the graph.
messengers = V(Twitter_network_simulation_1_graph_simple)[[new_nodes]]
messengers

# 9. Tell the function you're creating to output the items
#    in the variable "messengers".
return(messengers)
```

1. So far the reached list only has 1 element, so use that one
2. Add the people who just received the message `new_nodes`

Create a function for message diffusion

Script

```
# Remember to change the name of the function input so R does not get confused
# between the global variables that are stored in your environment and the "local"
# variables to be used in this specific function.
update_messengers = function(reached_input){
  nearest_neighbors_1 = ego(Twitter_network_simulation_1_graph_simple,
                            1,
                            reached_input)
  nearest_neighbors_2 = data.frame(names(unlist(nearest_neighbors_1)))
  nearest_neighbors_3 = data.frame(cbind(nearest_neighbors_2, rep(1)))
  nearest_neighbors_3[, 1] = as.character(nearest_neighbors_3[, 1])
  nearest_neighbors_3[, 2] = as.numeric(as.character(nearest_neighbors_3[, 2]))
  nearest_neighbors_4 = subset(nearest_neighbors_3,
                               !(nearest_neighbors_3[, 1] %in% names(reached_input)))
```

…continued on next page…

1. Find the adjacent nodes of the inputs of the function
2. Convert the names of those nodes into a data frame
3. Add a column of 1s to the data frame of node names
4. Make sure characters are read as characters and numbers as numbers
5. Exclude names in the original inputs of the function from the resulting data set
   This results in a list of only new nodes that could be affected

Create a function for message diffusion

Script

…continued from previous page…

```
  # Create a function to select which nodes will be affected or will receive
  # the message.
  select_nodes = function(input_list){
    random_trial = NULL
    for(i in 1:input_list){
      random_trial[i] = sample(random_set,
                               1,
                               replace = TRUE)
    }

    return(random_trial)
  }
```

1. Create a blank variable
2. Create a loop to run the random variable
3. Take a sample from the data set of random events we created earlier
4. Pick 1 value each time
5. Replace tells you to replace the value after it's been sampled so that it can be sampled again

6. Produce the output of the `random_trial`

…continued on next page…

Create a function for message diffusion

Script

```
…continued from previous page…

  # Apply the "select_nodes" function to the list of nodes that could be affected
  # and then subset all the affected nodes from the graph object.
  # The result is a list of nodes, in graph format that are added to the
  # original list that we created.
  keep_nodes = unlist(lapply(nearest_neighbors_4[, 2], select_nodes))
  new_messengers = as.character(nearest_neighbors_4[, 1][keep_nodes >= 1])
  new_nodes = unique(c(as.character(names(reached_input)),
                       new_messengers))
  messengers = V(Twitter_network_simulation_1_graph_simple)[[new_nodes]]
  return(messengers)
}
```

1. Apply the selection function to the column of 1s of the new nodes that could be affected
2. Keep only the affected nodes
3. Combine the previously affected nodes with the newly affected nodes
4. Subset all the affected nodes from the graph object
5. Output the affected nodes in graph format

Test your function

Script

```
# Set up your original inputs.
reached = list()
reached[[1]] = V(Twitter_network_simulation_1_graph_simple)[["Data Society"]]
reached

# Test the function "update_messengers".
reached[[2]] = update_messengers(reached[[1]])
reached

reached[[3]] = update_messengers(reached[[2]])
reached

reached[[4]] = update_messengers(reached[[3]])
reached

reached[[5]] = update_messengers(reached[[4]])
reached
```

- Note, the output will be different every time due to the randomized algorithm
- Recall that it's the `sample()` function that makes it random
- The way you know it's working is that each step retains all nodes from the prior step and potentially adds new ones

Run the network simulation

Script

```
# Run the message propagation simulation using the function you just created.
# Make sure you're starting with clean inputs.
reached = list()
reached[[1]] = V(Twitter_network_simulation_1_graph_simple)[["Data Society"]]

# Tell R when to stop running the simulation.
# The gorder() function counts the number of nodes (vertices) in a graph.
simulation_limit = gorder(Twitter_network_simulation_1_graph_simple)
simulation_limit

# Tell R to start the simulation from the first element in the "reached" list.
i = 1

# Run a while loop to execute the simulation.
while(length(reached[[i]]) < simulation_limit){
  reached[[i + 1]] = update_messengers(reached[[i]])

# Update i to go to the next value before continuing the loop.
  i = i + 1
}
```

The `while()` loop says that while the length of 'reached' is less than the simulation limit (the number of nodes), continue running the loop.

Your result will be a list of lists. Each list will contain all the nodes affected at each stage of the simulation.

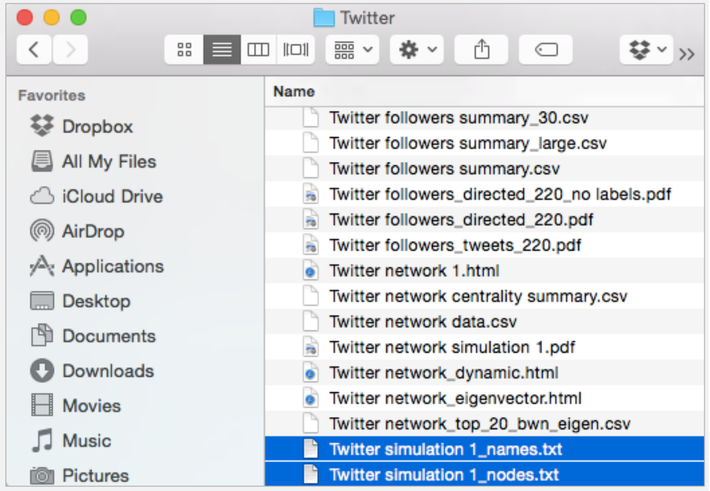
Check the results of your simulation

```
# Check the resulting output.
str(reached)

# Save your list as a text file.
# First save the numbers of the nodes in the graph.
lapply(reached, write,
       "new_Twitter simulation 1_nodes.txt",
       append = TRUE,
       ncolumns = length(reached[[length(reached)]]))

# Next save the names of the nodes.
reached_names = list()

for(i in 1:length(reached)){
  reached_names[[i]] = names(reached[[i]])
}
reached_names

lapply(reached_names, write,
       "new_Twitter simulation 1_names.txt",
       append = TRUE,
       ncolumns = length(reached[[length(reached)]]))
```

```
Console ~/Desktop/Network Analysis/Twitter/
> str(reached)
List of 157
 $ :Class 'igraph.vs'  atomic [1:1] 1
  .. ..- attr(*, "env")=<weakref>
  .. ..- attr(*, "graph")= chr "31fb1ca8-e7ec-42c2-a461-0b028785ac3e"
  .. ..- attr(*, "single")= logi TRUE
```

Script

1. Tells R to keep adding elements of the list
2. Tells R how many columns there are in the text file make sure that this number is at least as big as the number of items in the largest element of the list



Check the results of your simulation

Twitter simulation 1_nodes.txt

```
1
1 11 176 177 179 180
1 11 176 177 179 180
1 11 176 177 179 180 14 113
1 11 176 177 179 180 14 113 3 171 182 819
1 11 176 177 179 180 14 113 3 171 182 819 818
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774 174 760 778 689
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774 174 760 778 689 184 795 798 821 108 316 76 132 759 696
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774 174 760 778 689 184 795 798 821 108 316 76 132 759 696 13
87 96 809 26 307 18 763 768 773 389 405 698 851
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774 174 760 778 689 184 795 798 821 108 316 76 132 759 696 13
87 96 809 26 307 18 763 768 773 389 405 698 851 65 55 135 771 779 568 705 854 337 354 510
521 723
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774 174 760 778 689 184 795 798 821 108 316 76 132 759 696 13
87 96 809 26 307 18 763 768 773 389 405 698 851 65 55 135 771 779 568 705 854 337 354 510
521 723 7 808 813 310 323 24 99 120 682 44 122 138 579 342 351 239 792 417 442 606
1 11 176 177 179 180 14 113 3 171 182 819 818 185 46 800 804 8 75 178 817 820 324 328 10
173 112 312 313 115 106 304 774 174 760 778 689 184 795 798 821 108 316 76 132 759 696 13
```

Twitter simulation 1_names.txt

```
Data Society
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles ReinventMusic
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles ReinventMusic
jameskobielus Social In Manhattan 700 رتويت للأجمل  فكريه تسكعات k
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles ReinventMusic
jameskobielus Social In Manhattan 700 رتويت للأجمل  فكريه تسكعات k Randy Olson Ali Spagnola
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles ReinventMusic
jameskobielus Social In Manhattan 700 رتويت للأجمل  فكريه تسكعات k Randy Olson Ali Spagnola
StartUpsArts KINGRPG Luzdivina Funny Quotees うしくん
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles ReinventMusic
jameskobielus Social In Manhattan 700 رتويت للأجمل  فكريه تسكعات k Randy Olson Ali Spagnola
StartUpsArts KINGRPG Luzdivina Funny Quotees うしくん Ian Murray Pam Moore MarQuis Trill
SeyyahBirindar 孤独 帝越Mäsätö0028official
Data Society Targeted Marketing Shelly Palmer Sean Ellis Edward Nevraumont John Sonmez
Free VST Every Day Ryan TeesAsian Ted Coin George Sky Gallito Ingles ReinventMusic
jameskobielus Social In Manhattan 700 رتويت للأجمل  فكريه تسكعات k Randy Olson Ali Spagnola
StartUpsArts KINGRPG Luzdivina Funny Quotees うしくん Ian Murray Pam Moore MarQuis Trill
SeyyahBirindar 孤独 帝越Mäsätö0028official HotStocks فنان بمشاعري Party Vegas Style
tresensocial
```

Load simulation results into R

Script

```
# Load this file into R to make sure you know how to use it.
# readLines() will read each line as a separate character.
simulation_1_nodes = readLines("Twitter simulation 1_nodes.txt")
simulation_1_nodes

# strsplit() will split the characters based on the condition in the quotes.
# Here we'll just use a blank space.
simulation_1_nodes = strsplit(simulation_1_nodes, " ")
simulation_1_nodes

# Convert each element in the list to a number.
simulation_1_nodes = lapply(simulation_1_nodes, as.integer)
str(simulation_1_nodes)
```

Console ~/Desktop/Network Analysis/Twitter/

```
> str(simulation_1_nodes)
List of 157
 $ : int 1
 $ : int [1:6] 1 11 176 177 179 180
 $ : int [1:6] 1 11 176 177 179 180
```

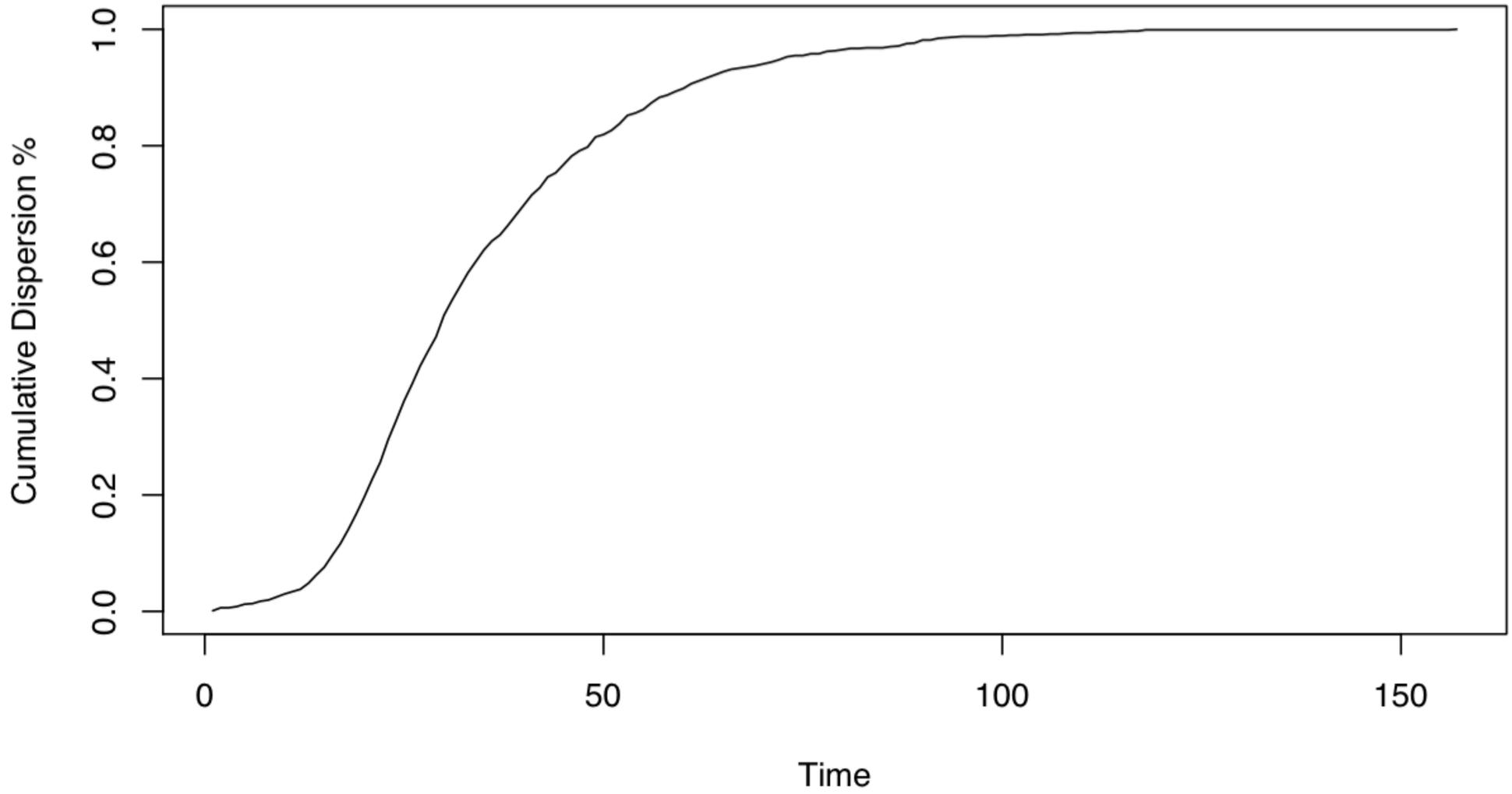
The result will be a list of lists where each one includes the numbers of the nodes affected at each step

Visualize simulation results: % over time

Script

```
# Let's create a data frame that shows the number of nodes that received
# the message at any given point in time.
# We'll use the sapply() function to do that. First create a function that counts the
# number of nodes in each element of our new list "reached".
count_function = function(x){
  length(reached[[x]])
}

dispersion = sapply(1:length(reached), count_function)
dispersion

# Create a data frame with a column that includes the percentage of the nodes in the
# network reached at every "time" or iteration.
dispersion_data = data.frame(1:length(reached),
                             dispersion / length(reached[[length(reached)]]))
View(dispersion_data)
colnames(dispersion_data) = c("Time", "Cumulative Dispersion %")

# Plot a graph of the percentage of network members reached over time.
plot(dispersion_data, type = "l")
```

Visualize simulation results: % over time

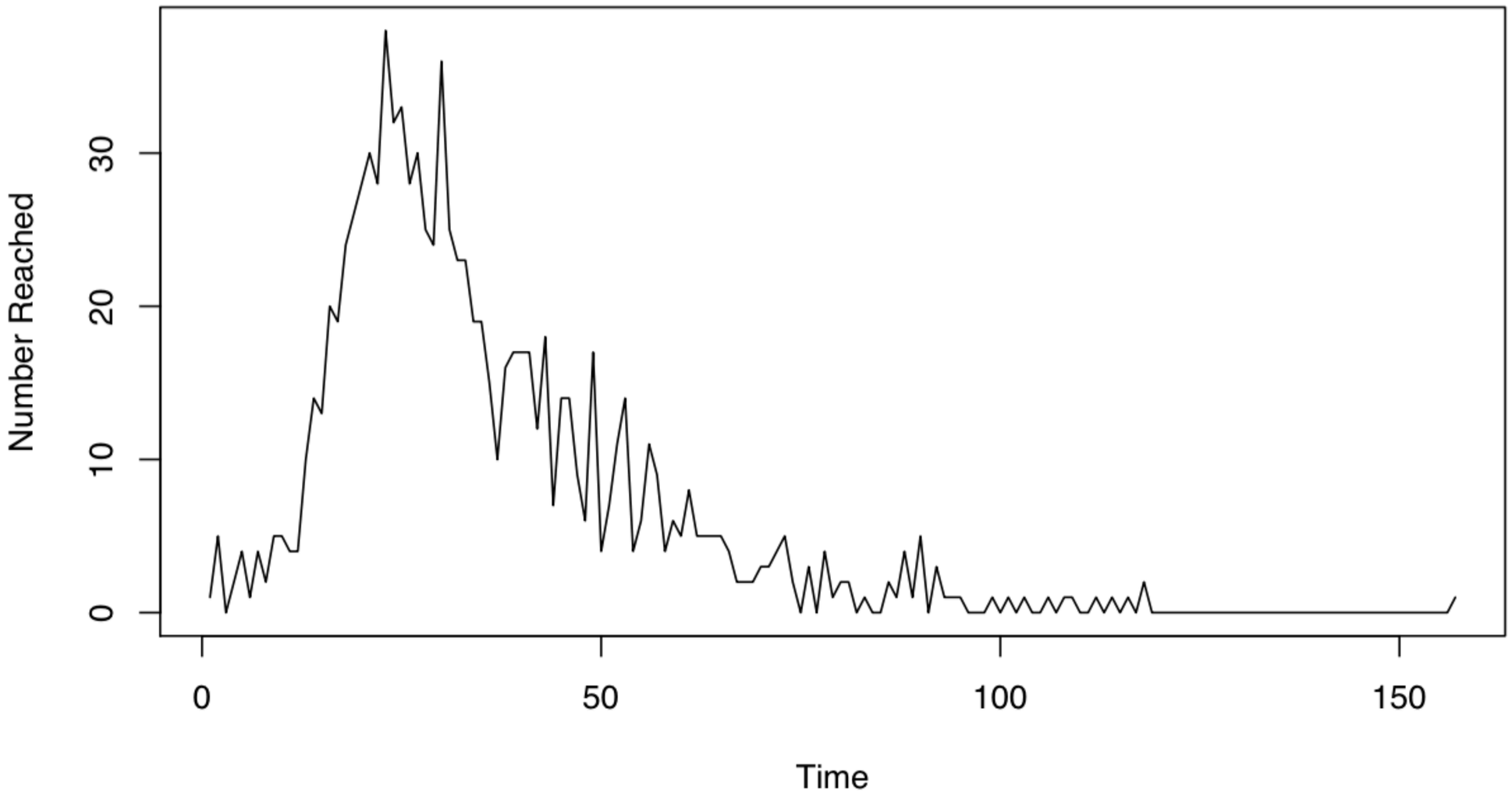


Visualize simulation results: # over time

Script

```
# Add a 0 to the beginning of the dispersion variable which includes the
# number of nodes reached at each iteration.
dispersion_1 = c(0, dispersion)

# Add the last number from dispersion to that data set so that this variable contains
# the same number of elements as the variable you just created.
dispersion_2 = c(dispersion, max(dispersion))

# Subtract the 2 data sets you just created to calculate the new number of nodes
# reached at each iteration.
dispersion_count = dispersion_2 - dispersion_1

# Remove the last instance because it now includes an extra 0.
dispersion_count = dispersion_count[1:length(dispersion_count) - 1]

# Convert your results to a clean data frame.
dispersion_number = data.frame(1:length(reached), dispersion_count)
colnames(dispersion_number) = c("Time", "Number Reached")

# Plot the results.
plot(dispersion_number, type = "l")
```

Visualize simulation results: # over time



Visualize simulation results: network

Script

```
# Let's color points on the graph as they receive the message. E() identifies
# the edges of the network graph.
E(Twitter_network_simulation_1_graph_simple)$color = "light blue"
V(Twitter_network_simulation_1_graph_simple)$color = "orange"
V(Twitter_network_simulation_1_graph_simple)$frame.color = "orange"
```

1. Set the color of the connecting lines
2. Set the fill color of the points
3. Set the color of the outline of the points

```
# Assign the points in the 20th element of the list "reached" to be a different color.
V(Twitter_network_simulation_1_graph_simple)
  $color[V(Twitter_network_simulation_1_graph_simple)
  %in% reached[[20]]] = "blue"

V(Twitter_network_simulation_1_graph_simple)
  $frame.color[V(Twitter_network_simulation_1_graph_simple)
  %in% reached[[20]]] = "blue"
```

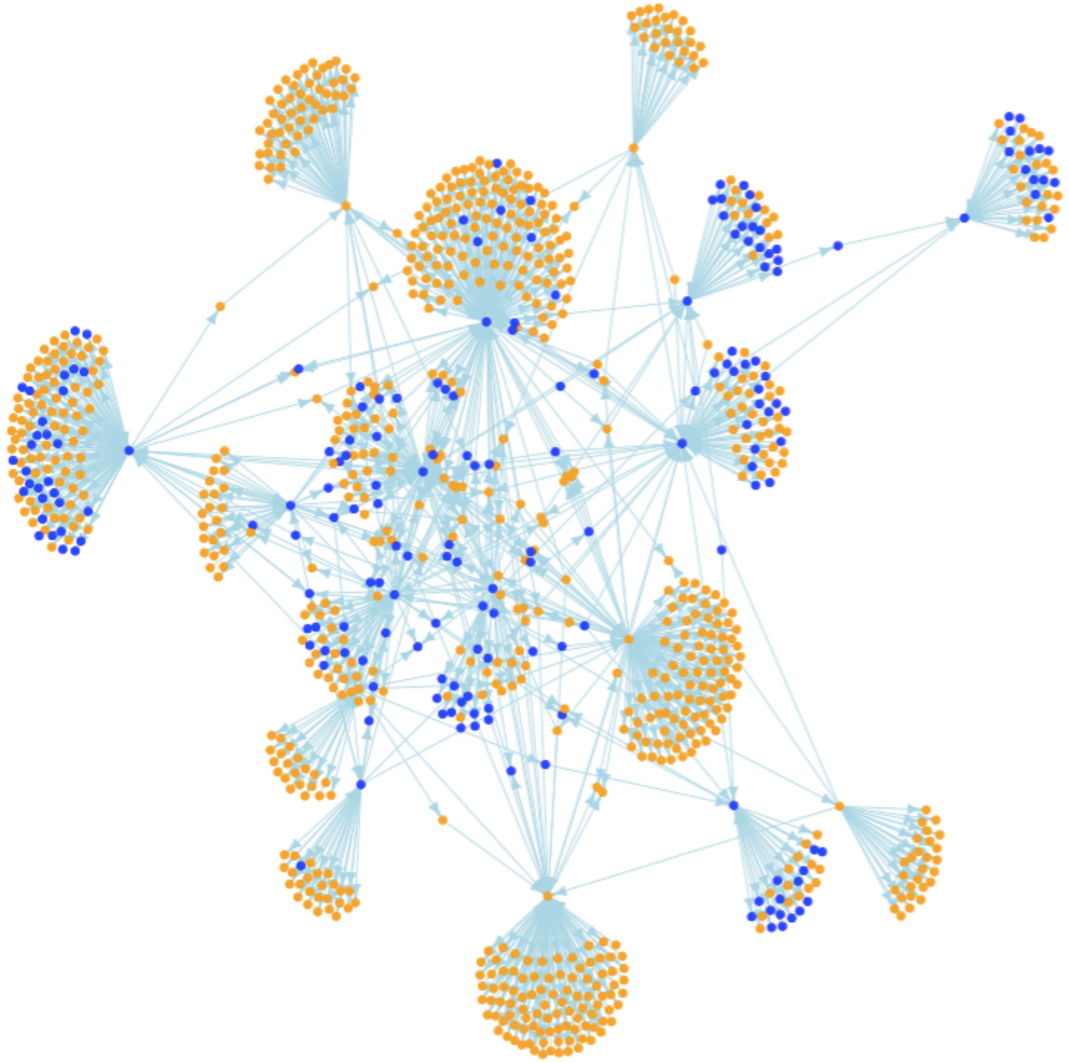
Visualize simulation results: network

Script

```
# Plot the updated graph, to speed this up, save it directly as a pdf.
pdf("Twitter network simulation_color.pdf",
    width = 10,
    height = 10)

# Set the seed to make the graph reproducible and assign the graph to a variable
set.seed(2)
Twitter_graph_color = plot(Twitter_network_simulation_1_graph_simple,
                           vertex.label = NA,
                           vertex.label.cex = 0.5,
                           vertex.size = 1.5,
                           edge.width = 0.5,
                           edge.arrow.size = 0.5)

# Tell R to complete the saving process.
dev.off()
```

1. Exclude labels
2. Sets the size of the labels
3. Set the size of the points
4. Set the thickness of the connecting lines
5. Set the size of the arrows at the ends of the connecting lines

Visualize simulation results: network



Visualize results: combine plots

Script

```
# First you'll need to set up the layout of the plots by telling R which
# portion of the plot should be occupied by each graph.
matrix(c(1, 1, 2, 2,
         1, 1, 3, 3),
       nrow = 2,
       ncol = 4,
       byrow = TRUE)
```

1. Tells R the order in which the plots should be displayed

2. `byrow = TRUE` tells R to fill the data by row, if this is not set, R will fill the data by column and your results won't look right

Visualize results: combine plots

Script

```
# Set up the pdf file.
pdf("Twitter network_3 charts.pdf",
    width = 20,
    height = 10)

# Set the margins around the graph to ensure that axis labels don't get cut off.
par(mar = c(5, 5, 5, 5))

# Set up layout for the plots.
layout(matrix(c(1, 1, 2, 2,
                1, 1, 3, 3),
              nrow = 2,
              ncol = 4,
              byrow = TRUE),
       widths = c(4, 4, 4, 4),
       heights = c(2, 2, 2, 2))
```

…continued on next page…

Note: `par()` stands for "parameters" and `mar()` stands for "margins"

1. Tells R the order in which the plots should be displayed

2. `byrow = TRUE` tells R to fill the data by row, if this is not set, R will fill the data by column and your results won't look right
3. `widths` determines how wide each column should be
4. `heights` determines how high each column should be

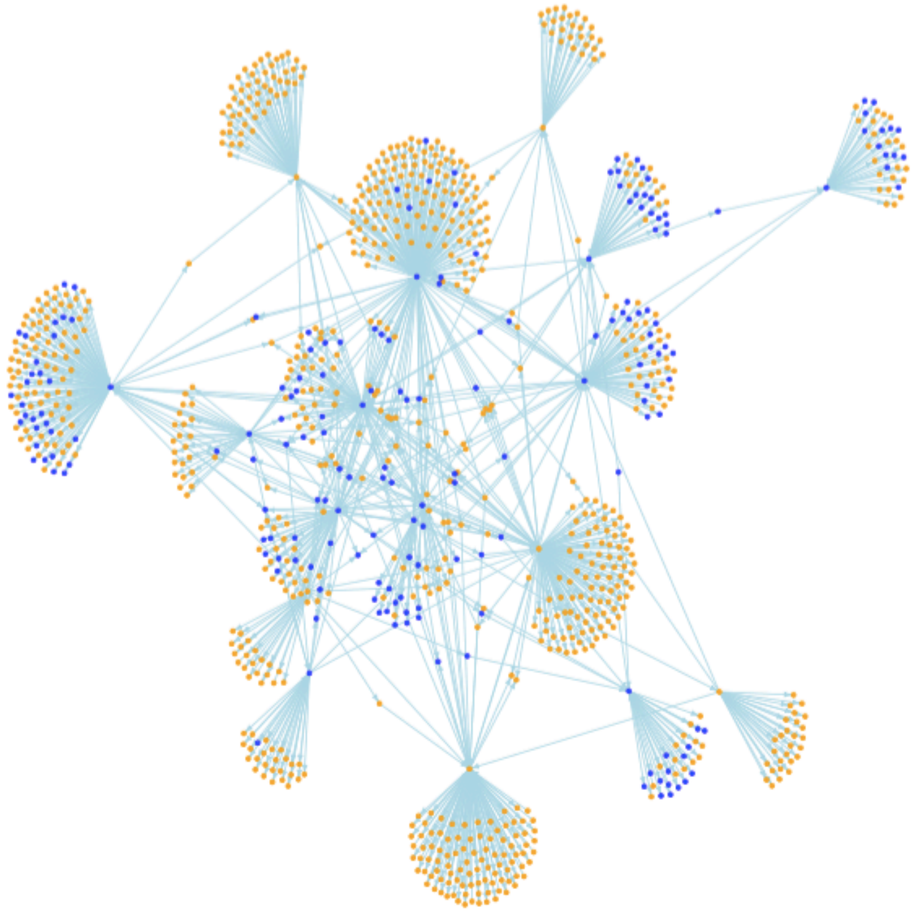
Visualize results: combine plots

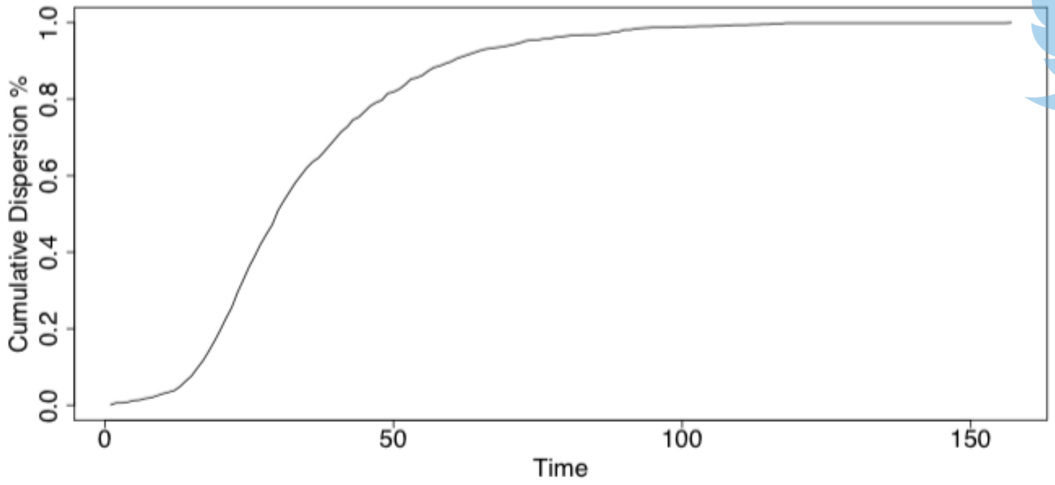
Script

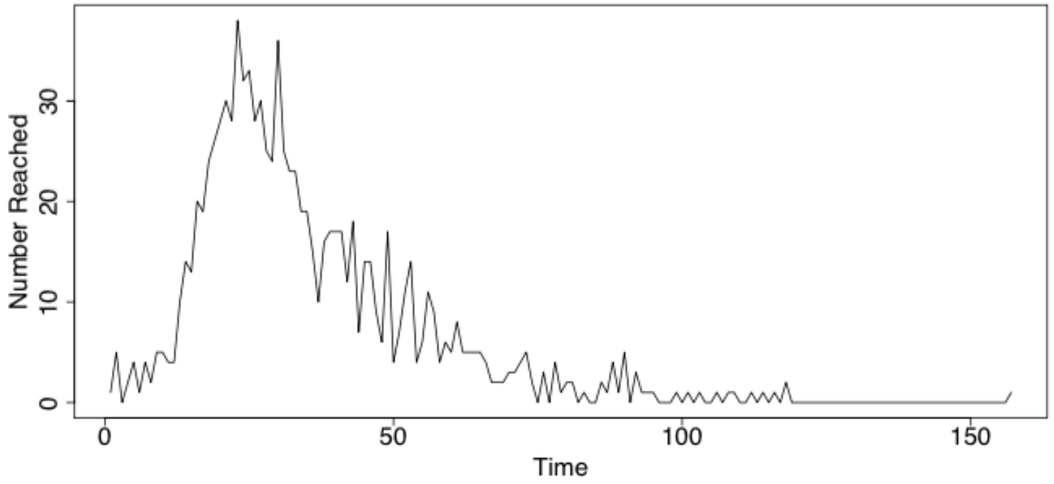
```
…continued from previous page…

# Display the plots in the order that you want them to map to your layout.
set.seed(2)
Twitter_graph_color = plot(Twitter_network_simulation_1_graph_simple,
                           vertex.label = NA,
                           vertex.label.cex = 0.5,
                           vertex.size = 1.5,
                           edge.width = 0.5,
                           edge.arrow.size = 0.3)

plot(dispersion_data, type = "l",
     cex.axis = 2,
     cex.lab = 2)

plot(dispersion_number, type = "l",
     cex.axis = 2, cex.lab = 2)

# Tell R to complete the saving process.
dev.off()
```

1. Percentage of base size of axis markers, 2 = 200%
2. Percentage of base size of axis labels, 2 = 200%

Visualize results: combine plots







Dispersion simulation: steps for video

1. Use the `saveHTML` function in the `animation` package
2. Set a new variable that denotes the step of the simulation equal to 1
3. Run a `while()` loop that runs starting with the number of the variable in step 2 and until every node in your graph is reached
4. For each step in the simulation update the color of the points
5. Set the layout of the image defining how the 3 or more graphs should be laid out
6. Plot the 1st graph
7. Plot the 2nd graph
8. Plot the 3rd graph
9. Update the variable from step 2 to tell R that we're on the next step of the simulation
10. Close the `while()` loop
11. Set the file name, image size and other output options

Dispersion simulation: create video

Script

```
# Save your network dispersion simulation as a video so you can share it with others.
install.packages("animation")
library(animation)
library(help = animation)

# To make the code easier to read, let's shorten the name of the graph.
Twitter_sim_graph = Twitter_network_simulation_1_graph_simple

# Save the video as an html file. Set it up such that each step updates each of the
# 3 graphs so that you can see the dispersion from several perspectives.
saveHTML({
  m = 1
  while(m <= length(reached)){

    # Update the node colors.
    V(Twitter_sim_graph)$color = "orange"
    V(Twitter_sim_graph)$frame.color = "orange"
    V(Twitter_sim_graph)$color[V(Twitter_sim_graph) %in% reached[[m]]] = "blue"
    V(Twitter_sim_graph)$frame.color[V(Twitter_sim_graph) %in% reached[[m]]] = "blue"
```

1. Set the color of the points
2. Set the color of point outlines
3. Set the color of the points that are effected at each step
4. Set the color of the outside of the points that are effected at each step

…continued on the next page…

Dispersion simulation: create video

Script

…continued from previous page…

```
    # Set the layout for the output of the plots.
    layout(matrix(c(1, 1, 2, 2,
                    1, 1, 3, 3),
                  nrow = 2,
                  ncol = 4,
                  byrow = TRUE),
           widths = c(4, 4, 4, 4),
           heights = c(2, 2, 2, 2))


    # Set the margins around the graph to ensure that axis labels don't get cut off.
    # Note: par() stands for "parameters" and mar() stands for "margins".
    par(mar = c(5, 5, 5, 5))
```

1. Tells R the order in which the plots should be displayed
2. `byrow = TRUE` tells R to fill the data by row, if this is not set, R will fill the data by column and your results won't look right
3. `widths` determines the number of graphs in the width
4. `heights` determines the number of graphs in the height

…continued on the next page…

Dispersion simulation: create video

Script

```
…continued from previous page…

    # Display the plots in the order that you want them to map to your layout.
    # 1st plot the network graph.
    set.seed(2)
    plot(Twitter_sim_graph,
         vertex.label = NA,
         vertex.label.cex = 0.5,
         vertex.size = 1,
         edge.width = 0.5,
         edge.arrow.size = 1)

…continued on the next page…
```

Dispersion simulation: create video

Script

…continued from previous page…

```
    # Plot the percentage of reach graph for the 2nd plot.
    count_function = function(m){
      length(reached[[m]])
    }
    dispersion = sapply(1:m, count_function)
    dispersion_data = data.frame(1:m,
                                 dispersion / length(reached[[length(reached)]]))
    colnames(dispersion_data) = c("Time", "Cumulative Dispersion %")
    plot(dispersion_data,
         type = "l",
         cex.axis = 2,
         cex.lab = 2,
         xlim = c(0, length(reached)),
         ylim = c(0, 1))
```

1. Percentage of base size of axis markers, 2 = 200%
2. Percentage of base size of axis labels, 2 = 200%
3. Set the x-axis limits so they don't change with every iteration
4. Set the y-axis limits so they don't change with every iteration

…continued on the next page…

Dispersion simulation: create video

Script

…continued from previous page…

```
    # Plot the number of nodes affected at each step on the 3rd plot.
    dispersion_1 = c(0, dispersion)
    dispersion_2 = c(dispersion, max(dispersion))
    dispersion_count = dispersion_2 - dispersion_1
    dispersion_count = dispersion_count[1:length(dispersion_count) - 1]
    dispersion_number = data.frame(1:m,
                                   dispersion_count)
    colnames(dispersion_number) = c("Time", "Number Reached")
    plot(dispersion_number,
         type = "l",
         cex.axis = 2,
         cex.lab = 2,
         xlim = c(0, length(reached)),
         ylim = c(0, 40))
```

1. Percentage of base size of axis markers, 2 = 200%
2. Percentage of base size of axis labels, 2 = 200%
3. Set the x-axis limits so they don't change with every iteration
4. Set the y-axis limits so they don't change with every iteration

…continued on the next page…

Dispersion simulation: create video

Script

…continued from previous page…

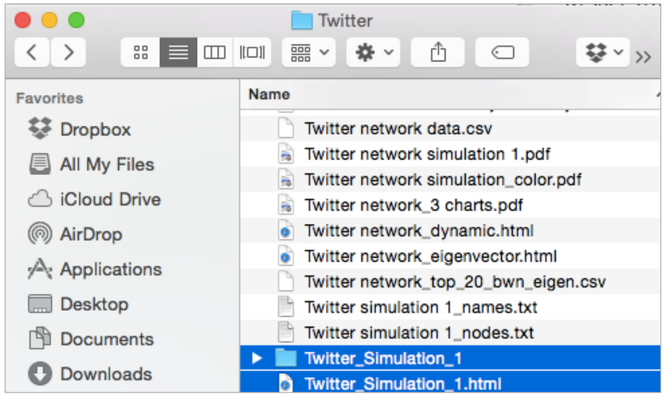
```
    m = m + 1}
  ani.options(interval = 0.2)
},
imgdir = "Twitter_Simulation_1",
img.name = "Twitter_Simulation_1",
htmlfile = "Twitter_Simulation_1.html",
navigator = TRUE,
ani.height = 600,
ani.width = 1000,
verbose = FALSE)
```

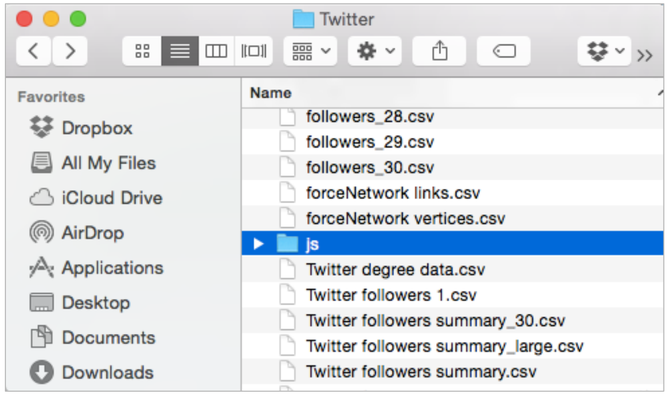
1. Update `m` to go to the next step in the `while()` loop
2. How long each step is shown in the video, in seconds

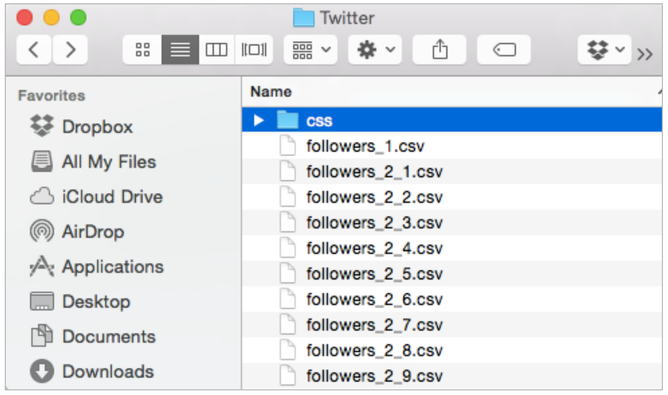
3. `imgdir` names the folder that will be created in your directory that will hold the individual images
4. `img.name` tells R how to name the saved images that comprise the file, *make sure the name matches the name of the html file to avoid glitches or mistakes*
5. `htmlfile` names the html file created, remember to *add the .html extension, otherwise this will not work*
6. `navigator` tells R whether to include a navigation bar
7. `ani.height` and `ani.width` set dimensions of the graph that are interpreted as pixels, this changes if you output pdf files instead of png files
8. `verbose` tells R whether to include the code at the bottom of the animation

Dispersion simulation: 4 outputs

1. html file that contains the simulation output
2. Folder that contains all the images for the animation
3. "js" folder that includes the execution files (you can open these in R)
4. "css" folder that defines the graphics of the output (you can open these in R)

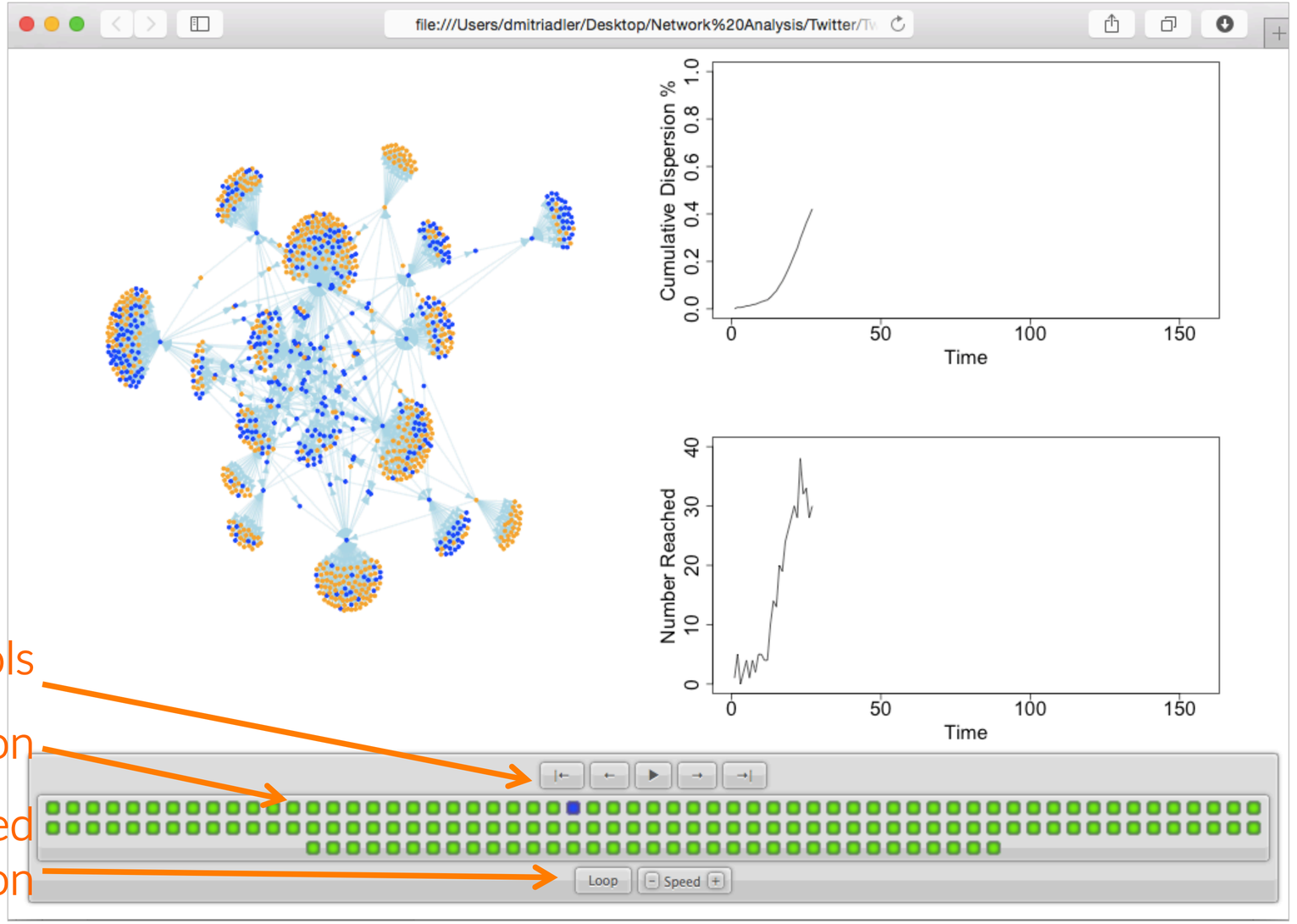






If you're going to share this simulation with anyone, you will need to send them all 4 files / folders!

Dispersion simulation: html file



Display controls

Step of the simulation

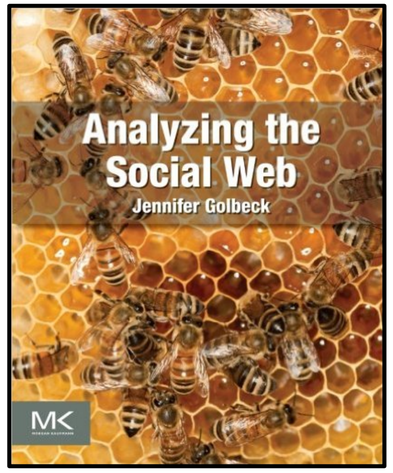
Change the speed of the animation

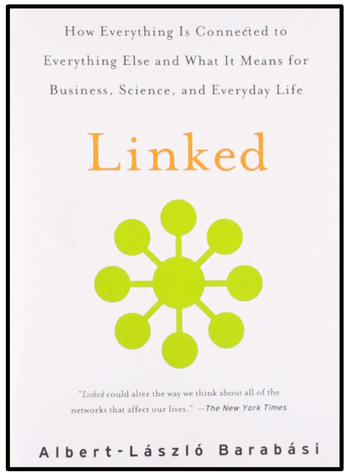
Ways to measure networks

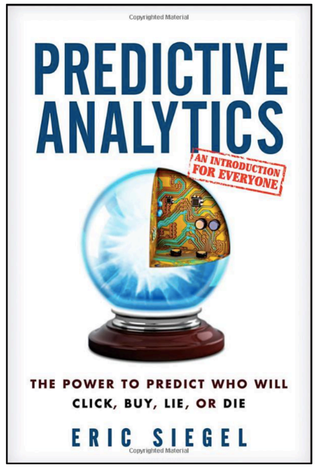
	Metric	Purpose
1	# of nodes	How many participants are included in the network?
2	# of edges	How many connections exist in a network?
3	Distance	How long does it take for information to travel through a network?
4	Degree (in-, out-)	Direction of connections, is someone a follower or an opinion leader?
5	Degree centrality	How many other people/objects can someone/something reach?
6	Closeness centrality	On average, how quickly can someone/something reach every other point in the network?
7	Betweenness centrality	How important is someone/something as a connector to the structure of the network?
8	Eigenvector centrality	How important is someone/something based on who/what else they are connected to?

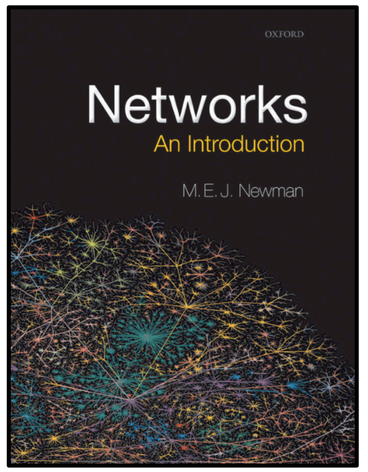
Additional resources

1. *Analyzing the Social Web* by Jennifer Golbeck
2. *Predictive Analytics* by Eric Siegel
3. *Linked* by Albert-László Barabási
4. *Networks* by Mark Newman









Congratulations!!

You have learned how to…

- ✓ visualize a large network
- ✓ identify key connectors and nodes
- ✓ create advanced interactive networks
- ✓ simulate the spread of information