

21 Recipes for Mining Twitter Data with rtweet

Bob Rudis

2018-01-03

Contents

Preface	7
About the Author	9
1 Using OAuth to Access Twitter APIs	11
1.1 Problem	11
1.2 Solution	11
1.3 Discussion	11
1.4 See Also	16
2 Looking Up the Trending Topics	17
2.1 Problem	17
2.2 Solution	17
2.3 Discussion	17
2.4 See Also	19
3 Extracting Tweet Entities	21
3.1 Problem	21
3.2 Solution	21
3.3 Discussion	21
3.4 See Also	23
4 Searching for Tweets	25
4.1 Problem	25
4.2 Solution	25
4.3 Discussion	25
4.4 See Also	26
5 Extracting a Retweet's Origins	27
5.1 Problem	27
5.2 Solution	27
5.3 Discussion	27
5.4 See Also	29
6 Creating a Graph of Retweet Relationships	31
6.1 Problem	31
6.2 Solution	31
6.3 Discussion	31
6.4 See Also	32
7 Visualizing a Graph of Retweet Relationships	33
7.1 Problem	33
7.2 Solution	33

7.3 See Also	35
8 Capturing Tweets in Real-time with the Streaming API	37
8.1 Problem	37
8.2 Solution	37
8.3 Discussion	37
8.4 See Also	39
9 Making Robust Twitter Requests	41
9.1 Problem	41
9.2 Solution	41
9.3 Discussion	41
9.4 See Also	41
10 Harvesting Tweets	43
10.1 Problem	43
10.2 Solution	43
10.3 Discussion	43
11 Creating a Tag Cloud from Tweet Entities	45
11.1 Problem	45
11.2 Solution	45
11.3 Discussion	45
12 Summarizing Link Targets	47
12.1 Problem	47
12.2 Solution	47
12.3 Discussion	47
12.4 See Also	48
13 Harvesting Friends and Followers	49
13.1 Problem	49
13.2 Solution	49
13.3 Discussion	49
13.4 See Also	50
14 Performing Setwise Operations on Friendship Data	51
14.1 Problem	51
14.2 Solution	51
14.3 Discussion	51
14.4 See Also	52
15 Resolving User Profile Information	53
15.1 Problem	53
15.2 Solution	53
15.3 Discussion	53
15.4 See Also	54
16 Analyzing the Authors of Tweets that Appear in Search Results	55
16.1 Problem	55
16.2 Solution	55
17 Visualizing Geodata with a Dorling Cartogram	57
17.1 Problem	57
17.2 Solution	57
17.3 Discussion	57

18 Geocoding Locations from Profiles (or Elsewhere)	61
18.1 Problem	61
18.2 Solution	61
18.3 Discussion	61
18.4 See Also	62

Preface

I'm using this as way to familiarize myself with bookdown so I don't make as many mistakes with my web scraping field guide book.

It's based on Matthew R. Russell's book. That book is out of distribution and much of the content is in Matthew's "Mining the Social Web" book. There will be *many* similarities between his "21 Recipes" book and this book *on purpose*. I am not claiming originality in this work, just making an R-centric version of the cookbook.

As he states in his tome, "this intentionally terse recipe collection provides you with 21 easily adaptable Twitter mining recipes".

The recipes contained in this book use the `rtweet` package by Michael W. Kearney. I'll be using the GitHub version of the package since it has some cutting-edge features and bug-fixes in it.

You can install the GitHub version of `[rtweet]` by first installing the `devtools` package via:

```
install.packages("devtools")
```

then installing the GitHub `rtweet` package via:

```
devtools::install_github("mkearney/rtweet")
```

NOTE: If you try to run examples in this book and receive an error about a package not being found or not available, you'll need to triage it by using one of the above methods. If any GitHub packages are used, each initial `library()` call will have a comment after it noting which `repository/package name` to use the `devtools` method with.

Matthew also states that "one other thing you should consider doing up front, if you haven't already, is quickly skimming through the official Twitter API documentation and related development documents linked on that page. Twitter has a very easy-to-use API with a lot of degrees of freedom". Michael has documented `rtweet` well, but reading the official documentation will really help.

This book also makes extensive use of the `tidyverse` meta-package. You will need to:

```
install.packages("tidyverse")
```

if you have not used packages from it before (it may take a few minutes, especially on Linux systems).

About the Author

Bob Rudis is a cybersecurity researcher and R aficionado presently thwarting cyber evildoers as [Master] Chief Data Scientist at Rapid7. He was formerly a Security Data Scientist & Managing Principal at Verizon, overseeing the team that produces the annual Data Breach Investigations Report. Bob is a serial tweeter (<https://twitter.com/hrbrmstr>), avid blogger (<https://rud.is/b>), author (Data-Driven Security — [<http://amzn.to/2CKvrqX>]), Stack Overflow contributor (<https://stackoverflow.com/users/1457051/hrbrmstr>), speaker, and regular contributor to the open source community (<https://github.com/hrbrmstr>). He is the author of several packages on CRAN including `**ggalt*`, **hrbrthemes**, **waffle**, **statebins**.

Chapter 1

Using OAuth to Access Twitter APIs

1.1 Problem

You want to access your own data or another user's data for analysis.

1.2 Solution

Take advantage of Twitter's OAuth implementation to gain full access to Twitter's entire API.

1.3 Discussion

Twitter uses OAuth Core 1.0 Revision A ("OAuth 1.0a" for short & to further reduce verbosity, "oauth" from now on). A few, key purposes of oauth in the context of Twitter are:

- to ensure end-users know an application is registered with Twitter, and
- know who the author(s) fo the application are;
- enable limiting what operations an application can perform with your Twitter account;
- obviate the need to share your actual Twitter username and password with a third party, which also
- enables revocation of application access to your Twitter account without resetting your password.

The `rtweet` package takes this one step further by having *you* create an "application", which is nothing more than you setting up some basic configuration information. To do so, you must visit apps.twitter.com and create a new application. You will need to provide values for the following fields:

- **Name** : something you'll remember
- **Description** : another place you can remind yourself what this is for
- **Website** : something that points to information you can use to associate this app when you've forgotten about it 5 years from now
- **Callback URL** : This **must** be `http://127.0.0.1:1410` (we'll see why in a moment)
- tick the agreement checkbox

Create an application

Application Details

Name *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? [OAuth 1.0a](#) applications should explicitly specify their `oauth_callback` URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Agreement

☒ Yes, I have read and agree to the [Twitter Developer Agreement](#).

Create your Twitter application

Once you submit that form, you'll see a new page:

Your application has been created. Please take a moment to review and adjust your application's settings.

my_rtweet_application


Test OAuth

Details

Settings

Keys and Access Tokens

Permissions



21 Recipes Example

<https://github.com/hrbrmstr/21-recipes>

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to [authenticate](#) requests to the Twitter Platform.

Access level	Read and write (modify app permissions)
Consumer Key (API Key)	akNTqsfSjJFQse1c55Vrm6BcZ (manage keys and access tokens)
Callback URL	http://127.0.0.1:1410
Callback URL Locked	No
Sign in with Twitter	Yes
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

Application Actions

Delete Application

Select the “Keys and Access Tokens” tab to see important information you’ll need:

my_rtweet_application

[Test OAuth](#)
[Details](#)
[Settings](#)
[Keys and Access Tokens](#)
[Permissions](#)

Application Settings

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

Consumer Key (API Key) akNTqsfSjJFQse1c55Vrm6BcZ

Consumer Secret (API Secret) HFF77rxG5HTx4Ui7RbxYVjoyUup5h0nc1s92Q88ddE0n4YFJZN

Access Level Read and write ([modify app permissions](#))

Owner hrbmstr

Owner ID 5685812

Application Actions

[Regenerate Consumer Key and Secret](#)
[Change App Permissions](#)

Your Access Token

You haven't authorized this application for your own account yet.

By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions

[Create my access token](#)

From the previous page and this page, you'll need the:

- Application **Name** (which is `my_rtweet_application` in this example but you need to use the one you supplied)
- **Consumer Key (API Key)** (which is `akNTqsfSjJFQse1c55Vrm6BcZ` in this example but you need to use your own)
- **Consumer Secret (API Secret)** (which is `HFF77rxG5HTx4Ui7RbxYVjoyUup5h0nc1s92Q88ddE0n4YFJZN` in this example, but — again — you need to use your own)

Store both of those in your `~/.Renvi` file. If you're unfamiliar with how to do that, see this handy section from "Efficient R Programming". I prefer storing these as such:

```
TWITTER_APP=my_rtweet_application
TWITTER_CONSUMER_KEY=akNTqsfSjJFQse1c55Vrm6BcZ
TWITTER_CONSUMER_SECRET=HFF77rxG5HTx4Ui7RbxYVjoyUup5h0nc1s92Q88ddE0n4YFJZN
```

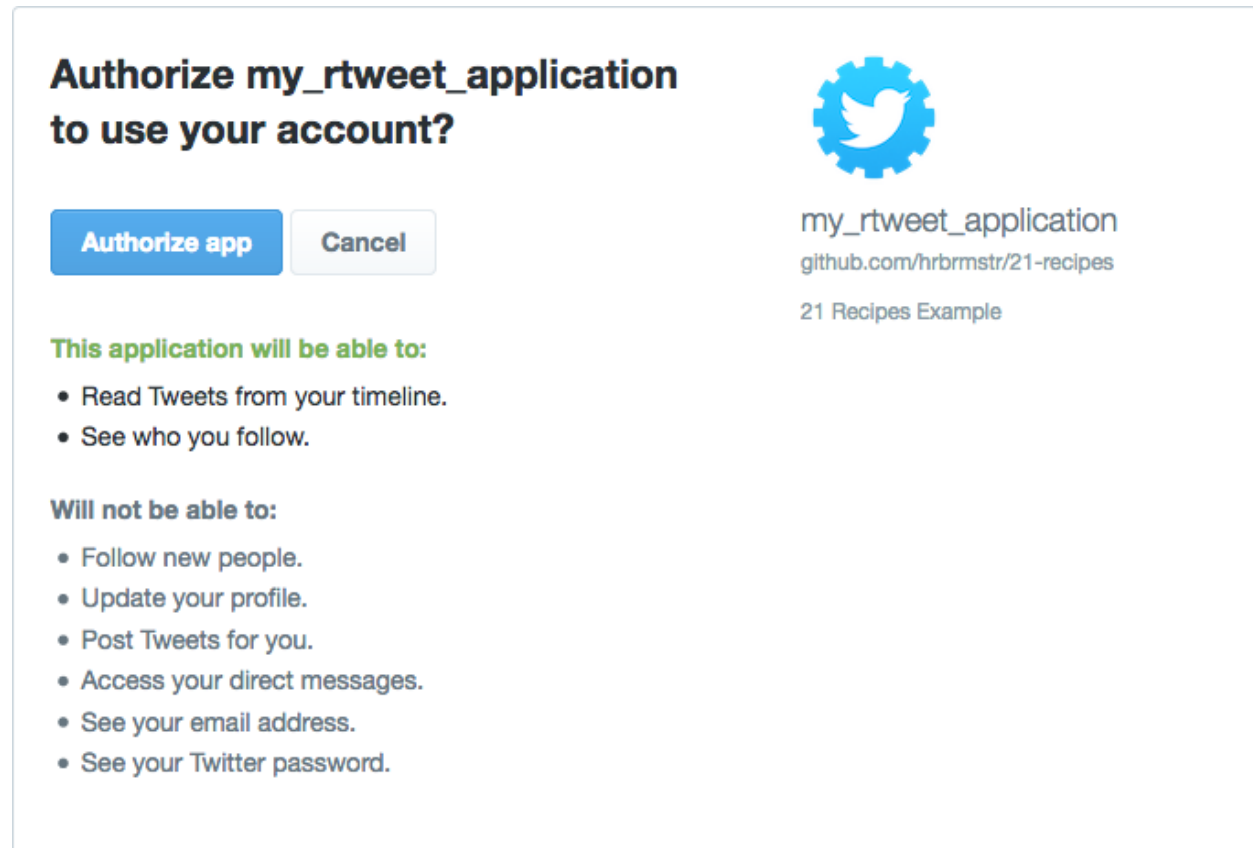
By storing these values in `~/.Renvi` you avoid exposing them in subdirectories or within scripts and will

always be able to reference them.

Now you can enable your Twitter account with this application and create a *token*:

```
create_token(
  app = Sys.getenv("TWITTER_APP"),
  consumer_key = Sys.getenv("TWITTER_CONSUMER_KEY"),
  consumer_secret = Sys.getenv("TWITTER_CONSUMER_SECRET")
) -> twitter_token
```

You should see a browser window appear that has an authorization form in it:



The image shows a Twitter authorization page. At the top, it says "Authorize my_rtweet_application to use your account?". Below this are two buttons: "Authorize app" (blue) and "Cancel" (grey). To the right of the buttons is a blue Twitter bird icon inside a gear, with the text "my_rtweet_application", "github.com/hrbrmstr/21-recipes", and "21 Recipes Example" below it. Under the "Authorize app" button, there is a section titled "This application will be able to:" followed by a bulleted list: "Read Tweets from your timeline." and "See who you follow." Below this is a section titled "Will not be able to:" followed by a bulleted list: "Follow new people.", "Update your profile.", "Post Tweets for you.", "Access your direct messages.", "See your email address.", and "See your Twitter password."

You'll also see:

```
Waiting for authentication in browser...
Press Esc/Ctrl + C to abort
```

in the R console.

The `rtweet` package used `httr` to send an oauth request to Twitter and then started up a local web server (this is why that weird `localhost` URL from before is necessary). When you authorize the application, the browser sends a response back to the web server `httr` spun up with some important, secret information that will make it possible for you to never have to do this oauth dance again.

If everything was successful, you'll see:

```
Authentication complete. Please close this page and return to R.
```

in the browser window, and:

```
Authentication complete.
```

in the R console.

The next step is *very important*.

Save the secret token you just received this way:

```
saveRDS(twitter_token, "~/rtweet.rds")
```

then create *one more* environment variable in `~/.Renviron`:

```
TWITTER_PAT=~/rtweet.rds
```

That last step will help ensure you never have to deal with oauth again (until you want to).

Keep this token file safe!! It enables anyone who has it to do virtually anything with your account. If you believe it has been exposed, go back to `apps.twitter.com` and delete the application (you can also choose to regenerate the **Consumer Key** and **Consumer Secret**, but it's often easier to just make a new application). You should also review your Twitter apps and ensure it's removed from there as well. Use the **Revoke access** button if it is:



1.4 See Also

- The official `rtweet` authentication vignette

Chapter 2

Looking Up the Trending Topics

2.1 Problem

You want to keep track of the trending topics on Twitter over a period of time.

2.2 Solution

Use `rtweet::trends_available()` to see trend areas and `rtweet::get_trends()` to pull trends, after which you can setup a task to retrieve and cache the trend data periodically.

2.3 Discussion

Twitter has extensive information on trending topics and their API enables you to see topics that are trending globally or regionally. Twitter uses Yahoo! Where on Earth identifiers (WOEIDs) for the regions which can be obtained from `rtweet::trends_available()`:

```
library(rtweet)
library(tidyverse)
```

```
(trends_avail <- trends_available())
```

```
## # A tibble: 467 x 8
##       name                                url parentid
## *   <chr>                                <chr>   <int>
## 1 Worldwide      http://where.yahooapis.com/v1/place/1         0
## 2 Winnipeg      http://where.yahooapis.com/v1/place/2972 23424775
## 3 Ottawa        http://where.yahooapis.com/v1/place/3369 23424775
## 4 Quebec        http://where.yahooapis.com/v1/place/3444 23424775
## 5 Montreal      http://where.yahooapis.com/v1/place/3534 23424775
## 6 Toronto       http://where.yahooapis.com/v1/place/4118 23424775
## 7 Edmonton      http://where.yahooapis.com/v1/place/8676 23424775
## 8 Calgary       http://where.yahooapis.com/v1/place/8775 23424775
## 9 Vancouver     http://where.yahooapis.com/v1/place/9807 23424775
## 10 Birmingham   http://where.yahooapis.com/v1/place/12723 23424975
## # ... with 457 more rows, and 5 more variables: country <chr>,
## #   woeid <int>, countryCode <chr>, code <int>, place_type <chr>
```

```
glimpse(trends_avail)
```

```
## Observations: 467
## Variables: 8
## $ name      <chr> "Worldwide", "Winnipeg", "Ottawa", "Quebec", "Mont...
## $ url       <chr> "http://where.yahooapis.com/v1/place/1", "http://w...
## $ parentid  <int> 0, 23424775, 23424775, 23424775, 2342477...
## $ country   <chr> "", "Canada", "Canada", "Canada", "Canada", "Canad...
## $ woeid     <int> 1, 2972, 3369, 3444, 3534, 4118, 8676, 8775, 9807,...
## $ countryCode <chr> NA, "CA", "CA", "CA", "CA", "CA", "CA", "CA", "CA"...
## $ code      <int> 19, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7...
## $ place_type <chr> "Supername", "Town", "Town", "Town", "Town", "Town..."
```

The Twitter API is somewhat unforgiving and unfriendly when you use it directly since it requires the use of a WOEID. Michael has made life much easier for us all by enabling the use of names or regular expressions when asking for trends from a particular place. That means we don't even need to care about capitalization:

```
(us <- get_trends("united states"))
```

```
## # A tibble: 50 x 9
##           trend
## *           <chr>
## 1 #WednesdayWisdom
## 2 #BiggestMisconceptionAboutMe
## 3 Tallahassee
## 4 Trump Tower
## 5 #MakeAMovieGross
## 6 #BombCyclone
## 7 Dominion Energy
## 8 Chip and Joanna Gaines
## 9 #11lined
## 10 Thomas S. Monson
## # ... with 40 more rows, and 8 more variables: url <chr>,
## #   promoted_content <lgl>, query <chr>, tweet_volume <int>, place <chr>,
## #   woeid <int>, as_of <dtm>, created_at <dtm>
```

```
glimpse(us)
```

```
## Observations: 50
## Variables: 9
## $ trend      <chr> "#WednesdayWisdom", "#BiggestMisconceptionAbo...
## $ url        <chr> "http://twitter.com/search?q=%23WednesdayWisd...
## $ promoted_content <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ query      <chr> "%23WednesdayWisdom", "%23BiggestMisconceptio...
## $ tweet_volume <int> 32943, NA, 10480, 13499, NA, NA, NA, NA, ...
## $ place      <chr> "United States", "United States", "United Sta...
## $ woeid      <int> 23424977, 23424977, 23424977, 23424977, 23424...
## $ as_of      <dtm> 2018-01-03 14:30:48, 2018-01-03 14:30:48, 20...
## $ created_at <dtm> 2018-01-03 14:27:51, 2018-01-03 14:27:51, 20..."
```

Twitter's documentation states that trends are updated every 5 minutes, which means you should not call the API more frequently than that and their current API rate-limit (Twitter puts some restrictions on how frequently you can call certain API targets) is 75 requests per 15-minute window.

The `rtweet::get_trends()` function returns a data frame. Our ultimate goal is to retrieve the trends data on a schedule and cache it. There are numerous — and usually complex — ways to schedule jobs. One cross-platform solution is to use R itself to run a task periodically. This means keeping an R console open

and running at all times, so is far from an optimal solution. See the `taskscheduleR` package for other ideas on how to setup more robust scheduled jobs.

In this example, we will:

- use a SQLite database to store the trends
- use the DBI and RSQLite packages to work with this database
- setup a never-ending loop with `Sys.sleep()` providing a pause between requests

```
library(DBI)
library(RSQLite)
library(rtweet) # mkearney/rtweet

repeat {
  message("Retrieveing trends...") # optional
  us <- get_trends("united states")
  db_con <- dbConnect(RSQLite::SQLite(), "data/us-trends.db")
  dbWriteTable(db_con, "us_trends", us, append=TRUE) # append=TRUE will update the table vs overwrite a
  dbDisconnect(db_con)
  Sys.sleep(10 * 60) # sleep for 10 minutes
}
```

Later on, we can look at this data with `dplyr/dbplyr`:

```
library(dplyr)

trends_db <- src_sqlite("data/us-trends.db")
us <- tbl(trends_db, "us_trends")
select(us, trend)
```

```
## # Source:   lazy query [?? x 1]
## # Database: sqlite 3.19.3
## #       [/Users/hrbrmstr/Development/21-recipes/data/us-trends.db]
##           trend
##           <chr>
## 1      #TuesdayThoughts
## 2      #backtowork
## 3      #SavannahHodaTODAY
## 4      Justin Timberlake
## 5 #MyTVShowWasCanceledBecause
## 6      #AM2DM
## 7      The Trump Effect
## 8      Carrie Underwood
## 9      Sean Ryan
## 10     Larry Krasner
## # ... with more rows
```

2.4 See Also

- RSQLite quick reference
- Introduction to dbplyr : <http://dbplyr.tidyverse.org/articles/dbplyr.html>

Chapter 3

Extracting Tweet Entities

3.1 Problem

You want to extract tweet entities such as @mentions, #hashtags, and short URLs from Twitter search results or other batches of tweets.

3.2 Solution

Use `rtweet::search_tweets()` or any of the *timeline* functions in `rtweet`.

3.3 Discussion

Michael has provided a very powerful search interface for Twitter data mining. `rtweet::search_tweets()` retrieves, parses and extracts an asounding amount of data for you to then use. Let's search Twitter for the #rstats hashtag and see what is available:

```
library(rtweet)
library(tidyverse)
```

```
(rstats <- search_tweets("#rstats", n=300)) # pull 300 tweets that used the "#rstats" hashtag
```

```
## # A tibble: 300 x 42
##       status_id      created_at      user_id  screen_name
##       <chr>         <dtm>         <chr>      <chr>
##  1 948562167787409417 2018-01-03 14:30:18    20444825    strnr
##  2 948560715606052866 2018-01-03 14:24:32    354581685   JanMulkens
##  3 948560507753107457 2018-01-03 14:23:43    14773239   albertcardona
##  4 948560413733597184 2018-01-03 14:23:20    433881702    estres
##  5 948560017799593985 2018-01-03 14:21:46 731230965700251648 chumblebiome
##  6 948559686936223745 2018-01-03 14:20:27    2872304195   jsgdatsci
##  7 948558836222263297 2018-01-03 14:17:04    440138503    paavopdf
##  8 948558543678070784 2018-01-03 14:15:54 722588617231769601 RLadiesLondon
##  9 948558535583043586 2018-01-03 14:15:52    90790110     Mardyb_
## 10 948558133462478848 2018-01-03 14:14:17 770490229769789440 RLadiesGlobal
## # ... with 290 more rows, and 38 more variables: text <chr>, source <chr>,
## #   reply_to_status_id <chr>, reply_to_user_id <chr>,
```

```
## #   reply_to_screen_name <chr>, is_quote <lgl>, is_retweet <lgl>,
## #   favorite_count <int>, retweet_count <int>, hashtags <list>,
## #   symbols <list>, urls_url <list>, urls_t.co <list>,
## #   urls_expanded_url <list>, media_url <list>, media_t.co <list>,
## #   media_expanded_url <list>, media_type <list>, ext_media_url <list>,
## #   ext_media_t.co <list>, ext_media_expanded_url <list>,
## #   ext_media_type <lgl>, mentions_user_id <list>,
## #   mentions_screen_name <list>, lang <chr>, quoted_status_id <chr>,
## #   quoted_text <chr>, retweet_status_id <chr>, retweet_text <chr>,
## #   place_url <chr>, place_name <chr>, place_full_name <chr>,
## #   place_type <chr>, country <chr>, country_code <chr>,
## #   geo_coords <list>, coords_coords <list>, bbox_coords <list>
```

```
glimpse(rstats)
```

```
## Observations: 300
## Variables: 42
## $ status_id          <chr> "948562167787409417", "9485607156060528...
## $ created_at         <dtm> 2018-01-03 14:30:18, 2018-01-03 14:24:...
## $ user_id            <chr> "20444825", "354581685", "14773239", "4...
## $ screen_name        <chr> "strnr", "JanMulkens", "albertcardona",...
## $ text               <chr> "Ten quick tips for machine learning in...
## $ source             <chr> "Buffer", "Twitter for Android", "Twitt...
## $ reply_to_status_id <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ reply_to_user_id   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ reply_to_screen_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ is_quote           <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
## $ is_retweet         <lgl> FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FA...
## $ favorite_count     <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ retweet_count      <int> 0, 2, 3, 2, 129, 7, 0, 8, 8, 8, 3, 2, 1...
## $ hashtags           <list> ["Rstats", "<netlify", "rstats", "visN...
## $ symbols            <list> [NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ urls_url           <list> ["biodatamining.biomedcentral.com/arti...
## $ urls_t.co          <list> ["https://t.co/Ir4gwHhyFR", NA, "https...
## $ urls_expanded_url  <list> ["https://biodatamining.biomedcentral...
## $ media_url          <list> ["https://pbs.twimg.com/media/DSn4Q4FW4...
## $ media_t.co         <list> ["https://t.co/BdEGlKEkkN", NA, NA, NA...
## $ media_expanded_url <list> ["https://twitter.com/strnr/status/948...
## $ media_type         <list> ["photo", NA, NA, NA, NA, NA, NA, NA, ...
## $ ext_media_url      <list> [<"http://pbs.twimg.com/media/DSn4Q4FW...
## $ ext_media_t.co     <list> [<"https://t.co/BdEGlKEkkN", "https://...
## $ ext_media_expanded_url <list> [<"https://twitter.com/strnr/status/94...
## $ ext_media_type     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ mentions_user_id   <list> [NA, "554300827", "3880760903", "24402...
## $ mentions_screen_name <list> [NA, "Nujcharee", "StephenEglen", "kie...
## $ lang               <chr> "en", "en", "en", "en", "en", "en", "en...
## $ quoted_status_id   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ quoted_text        <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_status_id  <chr> NA, "948512238528290816", "948479381701...
## $ retweet_text       <chr> NA, "Live from New York it's Saturday! ...
## $ place_url          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ place_name         <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ place_full_name    <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ place_type         <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ country            <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
```

```
## $ country_code      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ geo_coords        <list> [<NA, NA>, <NA, NA>, <NA, NA>, <NA, NA...
## $ coords_coords     <list> [<NA, NA>, <NA, NA>, <NA, NA>, <NA, NA...
## $ bbox_coords       <list> [<NA, NA, NA, NA, NA, NA, NA, NA>, <NA...
```

From the output, you can see that all the URLs (short and expanded), status id's, user id's and other hashtags are all available and all in a tidy data frame.

What are the top 10 (with ties) other hashtags used in conjunction with `#rstats` (for this search group)?

```
select(rstats, hashtags) %>%
  unnest() %>%
  mutate(hashtags = tolower(hashtags)) %>%
  count(hashtags, sort=TRUE) %>%
  filter(hashtags != "rstats") %>%
  top_n(10)
```

```
## # A tibble: 10 x 2
##   hashtags      n
##   <chr> <int>
## 1 datascience  91
## 2 bigdata      45
## 3 abdsc        24
## 4 statistics   24
## 5 machinelearning 21
## 6 dbplyr       20
## 7 dplyr        20
## 8 ai           16
## 9 python       16
## 10 rlang        13
```

3.4 See Also

- Official Twitter search API documentation
- Twitter entites information
- The tidyverse introduction.

Chapter 4

Searching for Tweets

4.1 Problem

You want to collect a sample of tweets from the public timeline for a custom query.

4.2 Solution

Use `rtweet::search_tweets()` and custom search operators.

4.3 Discussion

The Twitter API has free and paid tiers. The free tier is what many of us use and there are a number of operators that can be added to a search query to refine the results. We saw one of those in Recipe 3 by using the `#rstats` hashtag in the search query. But there are far more options at our disposal.

We can see all the `#rstats` tweets that aren't retweets:

```
library(rtweet)
library(tidyverse)

search_tweets("#rstats -filter:retweets") %>%
  select(text)

## # A tibble: 94 x 1
##                                     text
##                                <chr>
## 1 Ten quick tips for machine learning in computational biology https://t.co/I
## 2 Here's an issue I encountered during graph analysis regarding #tidygraph by
## 3 CRAN updates: BIGL biogas csv dbplyr dlsem encode ExtremeBounds forecastHyb
## 4 New CRAN package attempt with initial version 0.1.0 https://t.co/0QdYpl5sce
## 5 "\U0001f389 updates to a great \U0001f4e6: \"huxtable 2.0.0\" for writing L
## 6 Get started with #PredictiveAnalytics using R => see Chap.12 of online F
## 7 "Variable Width Bar Charts: Bar Mekko \U0001f4e6 #rstats #datascience https
## 8 "Next @rusersoxford meeting announced: @mspitschan on \"Using colour in R:
## 9 "@SteffLocke My @Bioconductor #rstats package has been accepted \U0001f600
## 10 "#RStats \U0001f4e6 - {attempt} is now on CRAN \U0001f389\nhttps://t.co/3sf
## # ... with 84 more rows
```

or, all the recent tweet-replies sent to @kearneywm:

```
search_tweets("to:kearneywm") %>%
  select(text)

## # A tibble: 100 x 1
##                                     text
##                                <chr>
## 1 @kearneywm @MattJStannard This is going to come in really handy for me in t
## 2                                     @kearneywm @slcathena And bam!
## 3 "@kearneywm @jhollist @dmi3k @ucfagls Could be worse: I brought an entire t
## 4                                     @kearneywm Well, the algorithm's namesake, in that case.
## 5 "@kearneywm His namesake died from an intestinal blockage. Where are the sh
## 6 @kearneywm @BreitbartNews First thing's first: Breitbart is NOT 'news'. Na
## 7 @kearneywm @Twitter interesting. This is what I see https://t.co/PVcuXp8u0r
## 8 "@kearneywm @Grantimus9 Dang, he was right again.\n\nDebate world is small!
## 9                                     @kearneywm Amazing. Indeed. @Grantimus9
## 10                                "@kearneywm Really good. Thanks. \n\n...Eating more fish"
## # ... with 90 more rows
```

and, even all the #rstats tweets that have GitHub links in them (but no #python hashtags):

```
search_tweets("#rstats url:github -#python") %>%
  select(text)

## # A tibble: 100 x 1
##                                     text
##                                <chr>
## 1 "RT @dataandme: \U0001f389 updates to a great \U0001f4e6: \"huxtable 2.0.0\"
## 2 "\U0001f389 updates to a great \U0001f4e6: \"huxtable 2.0.0\" for writing L
## 3 "@SteffLocke My @Bioconductor #rstats package has been accepted \U0001f600
## 4 RT @RR_Oxford: Is #reproducibility one of your 2018 resolutions? Register n
## 5 RT @RR_Oxford: Is #reproducibility one of your 2018 resolutions? Register n
## 6 RT @RR_Oxford: Is #reproducibility one of your 2018 resolutions? Register n
## 7 RT @noamross: Had a little New Year's brainstorm for an @rOpenSci #rstats p
## 8 RT @noamross: Had a little New Year's brainstorm for an @rOpenSci #rstats p
## 9 @dataandme #rstats huxtable 2.0.0 out: new quick_pdf, quick_html, quick_doc
## 10 RT @RR_Oxford: Registration is now open for our first workshop of 2018! @sw
## # ... with 90 more rows
```

4.4 See Also

- Twitter standard search operators

Chapter 5

Extracting a Retweet's Origins

5.1 Problem

You want to extract the originating source from a retweet.

5.2 Solution

If the tweet's `retweet_count` field is greater than 0, extract name out of the tweet's user field; also parse the text of the tweet with a regular expression.

5.3 Discussion

Twitter is *pretty darn good* about weaponizingutilizing the data on its platform. There aren't many cases nowadays when you need to parse RT or via in hand-crafted retweets, but it's good to have the tools in your arsenal when needed. We can pick out all the retweets from `#rstats` (warning: it's a retweet-heavy hashtag) and who they refer to using the `retweet_count` but also looking for a special regular expression (regex) and extracting data that way.

First, the modern, API-centric way:

```
library(rtweet)
library(tidyverse)

rstats <- search_tweets("#rstats", n=500)

glimpse(rstats)

## Observations: 500
## Variables: 42
## $ status_id      <chr> "948562167787409417", "9485607156060528...
## $ created_at     <dtm> 2018-01-03 14:30:18, 2018-01-03 14:24:...
## $ user_id        <chr> "20444825", "354581685", "14773239", "4...
## $ screen_name    <chr> "strnr", "JanMulkens", "albertcardona",...
## $ text           <chr> "Ten quick tips for machine learning in...
## $ source         <chr> "Buffer", "Twitter for Android", "Twitt...
## $ reply_to_status_id <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ reply_to_user_id <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
```

```
## $ reply_to_screen_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ is_quote <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
## $ is_retweet <lgl> FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, FA...
## $ favorite_count <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ retweet_count <int> 0, 2, 3, 2, 129, 7, 0, 8, 8, 8, 3, 2, 1...
## $ hashtags <list> ["Rstats", <"netlify", "rstats", "visN...
## $ symbols <list> [NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ urls_url <list> ["biodatamining.biomedcentral.com/arti...
## $ urls_t.co <list> ["https://t.co/Ir4gwHhyFR", NA, "https...
## $ urls_expanded_url <list> ["https://biodatamining.biomedcentral....
## $ media_url <list> ["http://pbs.twimg.com/media/DSn4Q4FW4...
## $ media_t.co <list> ["https://t.co/BdEGlKEkkN", NA, NA, NA...
## $ media_expanded_url <list> ["https://twitter.com/strnr/status/948...
## $ media_type <list> ["photo", NA, NA, NA, NA, NA, NA, NA, ...
## $ ext_media_url <list> [<"http://pbs.twimg.com/media/DSn4Q4FW...
## $ ext_media_t.co <list> [<"https://t.co/BdEGlKEkkN", "https://...
## $ ext_media_expanded_url <list> [<"https://twitter.com/strnr/status/94...
## $ ext_media_type <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ mentions_user_id <list> [NA, "554300827", "3880760903", "24402...
## $ mentions_screen_name <list> [NA, "Nujcharee", "StephenEglen", "kie...
## $ lang <chr> "en", "en", "en", "en", "en", "en", "en...
## $ quoted_status_id <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ quoted_text <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ retweet_status_id <chr> NA, "948512238528290816", "948479381701...
## $ retweet_text <chr> NA, "Live from New York it's Saturday! ...
## $ place_url <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ place_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ place_full_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ place_type <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ country <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ country_code <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ geo_coords <list> [<NA, NA>, <NA, NA>, <NA, NA>, <NA, NA...
## $ coords_coords <list> [<NA, NA>, <NA, NA>, <NA, NA>, <NA, NA...
## $ bbox_coords <list> [<NA, NA, NA, NA, NA, NA, NA, NA>, <NA...
```

```
filter(rstats, retweet_count > 0) %>%
  select(text, mentions_screen_name, retweet_count) %>%
  mutate(text = substr(text, 1, 30)) %>%
  unnest()
```

```
## # A tibble: 542 x 3
##           text retweet_count mentions_screen_name
##           <chr>          <int>          <chr>
## 1 RT @Nujcharee: Live from New York 2 Nujcharee
## 2 RT @StephenEglen: Here's our n 3 StephenEglen
## 3 RT @kierisi: literally one of 2 kierisi
## 4 RT @sellorm: Introducing the F 129 sellorm
## 5 RT @NicholasStrayer: I'm not s 7 NicholasStrayer
## 6 RT @clavitololo: Glad to announc 8 clavitololo
## 7 RT @clavitololo: Glad to announc 8 PLOSONE
## 8 RT @clavitololo: Glad to announc 8 clavitololo
## 9 RT @clavitololo: Glad to announc 8 PLOSONE
## 10 RT @clavitololo: Glad to announc 8 clavitololo
## # ... with 532 more rows
```

The `text` column was pared down for display brevity. If you run that code snippet you can examine it to see that it identifies the retweets and the first screen name is usually the main reference, but you get all of the screen names from the original tweet for free.

Here's the brute-force way. A regular expression is used that matches the vast majority of retweet formats. The pattern looks for them then extracts the first found screen name:

```
# regex mod from https://stackoverflow.com/questions/655903/python-regular-expression-for-retweets
filter(rstats, str_detect(text, "(RT|via)((?:[[:blank:]]|\\W*@\\w+)+)")) %>%
  select(text, mentions_screen_name, retweet_count) %>%
  mutate(extracted = str_match(text, "(RT|via)((?:[[:blank:]]|\\W*@\\w+)+)")[,3]) %>%
  mutate(text = substr(text, 1, 30)) %>%
  unnest()
```

```
## # A tibble: 476 x 4
##               text retweet_count extracted
##               <chr>         <int>      <chr>
## 1 RT @Nujcharee: Live from New Y           2 @Nujcharee
## 2 RT @StephenEglen: Here's our n           3 @StephenEglen
## 3 RT @kierisi: literally one of             2 @kierisi
## 4 RT @sellorm: Introducing the F          129 @sellorm
## 5 RT @NicholasStrayer: I'm not s           7 @NicholasStrayer
## 6 RT @clavitololo: Glad to announc          8 @clavitololo
## 7 RT @clavitololo: Glad to announc          8 @clavitololo
## 8 RT @clavitololo: Glad to announc          8 @clavitololo
## 9 RT @clavitololo: Glad to announc          8 @clavitololo
## 10 RT @clavitololo: Glad to announc         8 @clavitololo
## # ... with 466 more rows, and 1 more variables: mentions_screen_name <chr>
```

You should try the above snippets for other tags as there will be cases when the regex will pick up retweets Twitter has failed to capture.

5.4 See Also

- Twitter official documentation on what happens to retweets when origin tweets are deleted

Chapter 6

Creating a Graph of Retweet Relationships

6.1 Problem

You want to construct and analyze a graph data structure of retweet relationships for a set of query results.

6.2 Solution

Query for the topic, extract the retweet origins, and then use `igraph` to construct a graph to analyze.

6.3 Discussion

Recipes 4 and 5 introduced and expanded on searching Twitter plus looking for retweets. The `igraph` package can be used to capture and analyze details of relationships across retweets. We'll focus on just examining the Twitter user pair relationships.

Let's get a larger sample this time — 1,500 tweets in `#rstats`. We can use the technique from the previous recips and:

- find the retweets (using the API-provided data)
- expand out all the mentioned screen names
- create an `igraph` graph object
- look at some summary statistics for the graph

```
library(rtweet)
library(igraph)
library(hrbrthemes)
library(tidyverse)
```

```
rstats <- search_tweets("#rstats", n=1500)

filter(rstats, retweet_count > 0) %>%
  select(screen_name, mentions_screen_name) %>%
  unnest(mentions_screen_name) %>%
  filter(!is.na(mentions_screen_name)) %>%
  graph_from_data_frame() -> rt_g
```

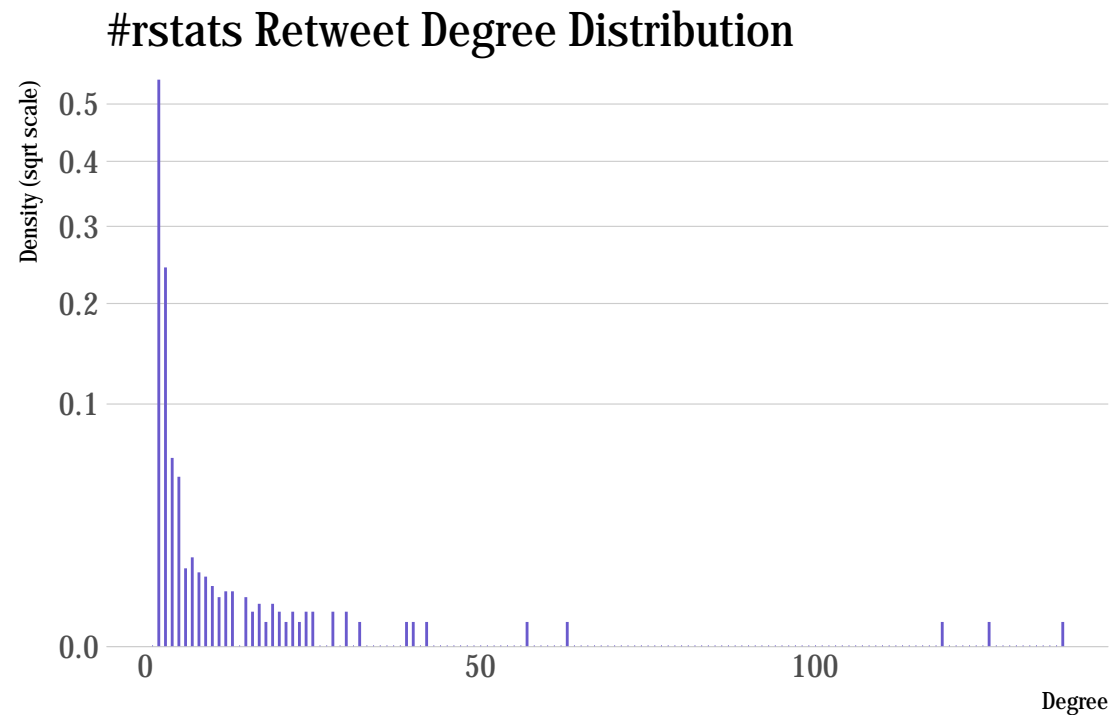
You can reference the `igraph print()` and `summary()` functions for more information on the output of `summary()` but output from the following line shows that the graph is Directed with Named vertices and it has 958 vertices and 1,498 edges.

```
summary(rt_g)
```

```
## IGRAPH 544f979 DN-- 958 1498 --
## + attr: name (v/c)
```

We'll produce more visualizations in the next recipe, but the *degree* of graph vertices is one of the most fundamental properties of a graph and it's much nicer to see the degree distribution than stare at a wall of numbers:

```
ggplot(data_frame(y=degree_distribution(rt_g), x=1:length(y))) +
  geom_segment(aes(x, y, xend=x, yend=0), color="slateblue") +
  scale_y_continuous(expand=c(0,0), trans="sqrt") +
  labs(x="Degree", y="Density (sqrt scale)", title="#rstats Retweet Degree Distribution") +
  theme_ipsum_rc(grid="Y", axis="x")
```



6.4 See Also

- `igraph`

Chapter 7

Visualizing a Graph of Retweet Relationships

7.1 Problem

You want to visualize a graph of retweets.

7.2 Solution

There are a plethora of ways to visualize graph structures in R. One recent and popular one is `ggraph`.

Given the cookbook-nature of this book, we'll cover one more visualization about retweet relationships. Let's explore the entire retweet network and label the screen names with the most retweets over a given search term (and use `#rstats` again, but gather more tweets this time to truly make a spaghetti chart):

```
library(rtweet)
library(igraph)
library(hrbrthemes)
library(ggraph)
library(tidyverse)

rstats <- search_tweets("#rstats", n=1500)

# same as previous recipe
filter(rstats, retweet_count > 0) %>%
  select(screen_name, mentions_screen_name) %>%
  unnest(mentions_screen_name) %>%
  filter(!is.na(mentions_screen_name)) %>%
  graph_from_data_frame() -> rt_g
```

To help de-clutter the vertex labels, we'll only add labels for nodes that have a degree of 20 or more (rough guess — you should look at the degree distribution for more formal work). We'll also include the degree for those nodes so we can size them properly:

```
V(rt_g)$node_label <- unname(ifelse(degree(rt_g)[V(rt_g)] > 20, names(V(rt_g)), ""))
V(rt_g)$node_size <- unname(ifelse(degree(rt_g)[V(rt_g)] > 20, degree(rt_g), 0))
```

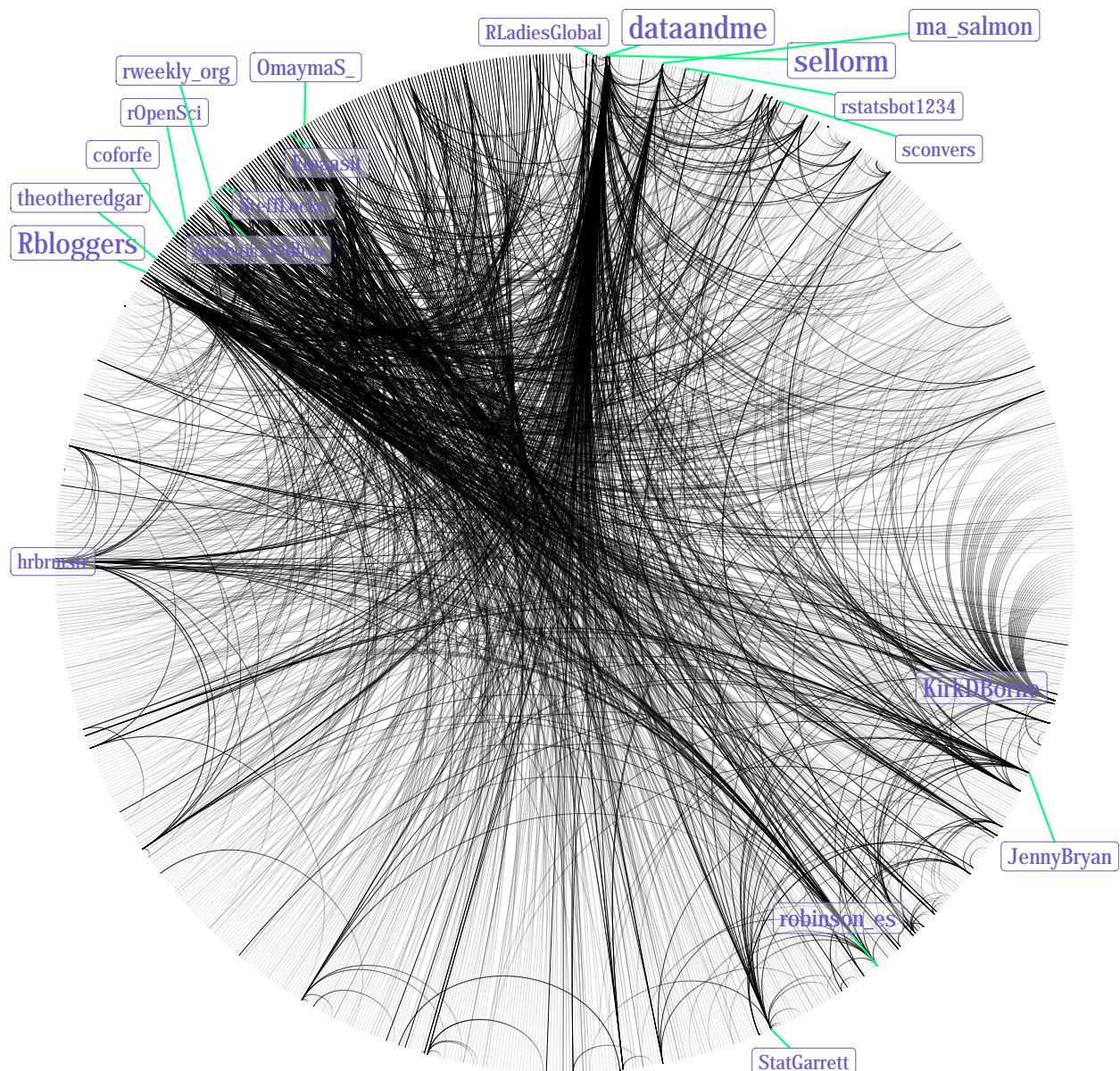
Now, we'll create the graph. Using `..index..` for the alpha channel will help show edge weight without too much extra effort. Note the *heavy* customization of `geom_node_label()`. Thomas made it way too easy

to make beautiful network graphs with `ggraph`:

```
ggraph(rt_g, layout = 'linear', circular = TRUE) +
  geom_edge_arc(edge_width=0.125, aes(alpha=..index..)) +
  geom_node_label(aes(label=node_label, size=node_size),
    label.size=0, fill="#ffffff66", segment.colour="springgreen",
    color="slateblue", repel=TRUE, family=font_rc, fontface="bold") +
  coord_fixed() +
  scale_size_area(trans="sqrt") +
  labs(title="Retweet Relationships", subtitle="Most retweeted screen names labeled. Darkers edges == m
  theme_graph(base_family=font_rc) +
  theme(legend.position="none")
```

Retweet Relationships

Most retweeted screen names labeled. Darkers edges == more retweets. Node size == larger degree



7.3 See Also

- Enter `twitter network analysis r` into Google (seriously!). Lots of folks have worked in this space and blogged or wrote about their efforts.

Chapter 8

Capturing Tweets in Real-time with the Streaming API

8.1 Problem

You want to capture a stream of public tweets in real-time, optionally filtering by select screen names or keywords in the text of the tweet.

8.2 Solution

Use `rtweet::stream_tweets()`.

8.3 Discussion

Michael has — once again — made it way too easy to work with Twitter’s API. The `rtweet::stream_tweets()` function has tons of handy options to help capture tweets in real time. The primary `q` parameter is very versatile and has four possible capture modes:

- The default, `q = ""`, returns a small random sample of all publicly available Twitter statuses.
- To filter by keyword, provide a comma separated character string with the desired phrase(s) and keyword(s).
- Track users by providing a comma separated list of user IDs or screen names.
- Use four latitude/longitude bounding box points to stream by geo location. This must be provided via a vector of length 4, e.g., `c(-125, 26, -65, 49)`.

Let’s capture one minute of tweets in the good ol’ U S of A (this is one of Michael’s examples from the manual page for `rtweet::stream_tweets()`).

```
library(rtweet)
library(tidyverse)

stream_tweets(
  lookup_coords("usa"), # handy helper function in rtweet
  verbose = FALSE,
  timeout = (60 * 1),
) -> usa
```

```
##
Found 500 records...
Found 923 records...
Imported 923 records. Simplifying...
```

A 60 second stream resulted in well over 1,000 records.

Where are they tweeting from?

```
count(usa, place_full_name, sort=TRUE)
```

```
## # A tibble: 506 x 2
##   place_full_name      n
##   <chr> <int>
## 1 Manhattan, NY      27
## 2 Chicago, IL        24
## 3 Florida, USA       21
## 4 Georgia, USA       18
## 5 Toronto, Ontario   18
## 6 Los Angeles, CA    16
## 7 Washington, DC     15
## 8 Pennsylvania, USA  13
## 9 Columbus, OH       11
## 10 Del Aire, CA       11
## # ... with 496 more rows
```

What are they tweeting about?

```
unnest(usa, hashtags) %>%
  count(hashtags, sort=TRUE) %>%
  filter(!is.na(hashtags))
```

```
## # A tibble: 252 x 2
##   hashtags      n
##   <chr> <int>
## 1 job      47
## 2 WPMOYChallenge 47
## 3 CareerArc    31
## 4 Hiring      27
## 5 hiring      23
## 6 Job         7
## 7 Jobs         7
## 8 Retail       7
## 9 Veterans     7
## 10 Hospitality  6
## # ... with 242 more rows
```

What app are they using?

```
count(usa, source, sort=TRUE)
```

```
## # A tibble: 25 x 2
##   source      n
##   <chr> <int>
## 1 Twitter for iPhone 572
## 2 Twitter for Android 139
## 3 TweetMyJOBS       56
## 4 Instagram         53
## 5 Twitter Web Client 36
```

```
## 6    Twitter for iPad    11
## 7          Cities        9
## 8          ijg          8
## 9    Foursquare         7
## 10   Tweetbot for iOS    7
## # ... with 15 more rows
```

Michael covers the streaming topic in-depth in a vignette.

8.4 See Also

- Consuming streaming data

Chapter 9

Making Robust Twitter Requests

9.1 Problem

You want to write a long-running script that harvests large amounts of data, such as the friend and follower ids for a very popular Twitterer; however, the Twitter API is inherently unreliable and imposes rate limits that require you to always expect the unexpected.

9.2 Solution

Use `rtweet`.

9.3 Discussion

No code examples and not much expository in this chapter (unlike its Python counterpart). Michael has taken much of the pain away by having the package abstract the rate-limit issues and API wonkiness away from your code.

Having said that, you can work on making these Twitter scripts or other scripts more robust by wrapping potentially troublesome calls in `purrr::safely()` and testing for the `result` before continuing with data operations.

9.4 See Also

- `purrr::safely()`

Chapter 10

Harvesting Tweets

10.1 Problem

You want to harvest and store tweets from a collection of id values, or harvest entire timelines of tweets.

10.2 Solution

Use `rtweet`'s timeline and status functions.

10.3 Discussion

Recipe 2 showed how to do this with SQLite. Unlike other API's `rtweet` returns a tidy data frame which makes it easy to put data into such rectangular data stores.

Rather than repeat the example, let's take a quick look at all of the harvesting functions in `rtweet`:

- `get_collections`: Get collections by user or status id.
- `get_favorites`: Get tweets data for statuses favorited by one or more target users.
- `get_followers`: Get user IDs for accounts following target user.
- `get_friends`: Get user IDs of accounts followed by target user(s).
- `get_mentions`: Get mentions for the authenticating user.
- `get_retweeters`: Get user IDs of users who retweeted a given status.
- `get_retweets`: Get the most recent retweets of a specific Twitter status
- `get_timeline`: Get one or more user timelines (tweets posted by target user(s)).
- `get_timelines`: Get one or more user timelines (tweets posted by target user(s)).
- `lookup_collections`: Get collections by user or status id.
- `lookup_coords`: Get coordinates of specified location.
- `lookup_friendships`: Lookup friendship information between two specified users.
- `lookup_statuses`: Get tweets data for given statuses (status IDs).
- `lookup_tweets`: Get tweets data for given statuses (status IDs).
- `lookup_users`: Get Twitter users data for given users (user IDs or screen names).

- `search_tweets`: Get tweets data on statuses identified via search query.
- `search_tweets2`: Get tweets data on statuses identified via search query.
- `search_users`: Get users data on accounts identified via search query.
- `stream_tweets`: Collect a live stream of Twitter data.
- `stream_tweets2`: Collect a live stream of Twitter data.

One handy method for exporting this rectangular tweet data to a file format virtually any collaborator can use is `rtweet::write_as_csv()` which saves a flattened CSV (no nested column data).

Chapter 11

Creating a Tag Cloud from Tweet Entities

11.1 Problem

You want to make a meaningless word cloud.

11.2 Solution

Use harvesting techniques shown in previous recipes and pass the cloud-destined entities to an R wordcloud package.

11.3 Discussion

Word clouds are virtually devoid of meaning. Neiman Lab went to far as to call them harmful. But, this recipe is in the Python version of the book (figures, eh?) and this was designed to be a 1:1 mapping of said book, so let's proceed.

The following uses some handy text taming and word cloud packages to make a collage from #NationalScienceFictionDay tweets:

```
library(rtweet)
library(tidytext)
library(magick)
library(kumojars) # hrbrmstr/kumojars
library(kumo) # hrbrmstr/kumo
library(tidyverse)

scifi <- search_tweets("#NationalScienceFictionDay", n=1500)

data_frame(txt=str_replace_all(scifi$text, "#NationalScienceFictionDay", "")) %>%
  unnest_tokens(word, txt) %>%
  anti_join(stop_words, "word") %>%
  anti_join(rtweet::stopwordslangs, "word") %>%
  anti_join(data_frame(word=c("https", "t.co")), "word") %>% # need to make a more technical stopwords
  filter(nchar(word)>3) %>%
```


Chapter 12

Summarizing Link Targets

12.1 Problem

You want to summarize the text of a web page that's indicated by a short URL in a tweet.

12.2 Solution

Extract the text from the web page, and then use a natural language processing (NLP) toolkit to help you extract the most important sentences to create a machine-generated abstract.

12.3 Discussion

R has more than a few NLP tools to work with. We'll work with the `LSAfun` package for this exercise. As the acronym-laden package name implies, it uses Latent Semantic Analysis (LSA) to determine the most important bits in a set of text.

We'll use tweets by data journalist extraordinaire Matt Stiles. Matt works for the Los Angeles Times and I learn a *ton* from him on a daily basis. He's on top of *everything*. Let's summarise some news he shared recently from the New York Times, Reuters, Washington Post, Five Thirty-Eight and his employer.

We'll limit our exploration to the first three new links we find.

```
library(rtweet)
library(LSAfun)
library(jerichojars) # hrbrmstr/jerichojars
library(jericho) # hrbrmstr/jericho
library(tidyverse)

stiles <- get_timeline("stiles")

filter(stiles, str_detect(urls_expanded_url, "nyti|reut|wapo|lat\\.ms|53ei")) %>% # only get tweets wi
  pull(urls_expanded_url) %>% # extract the links
  flatten_chr() %>% # mush them into a nice character vector
  head(3) %>% # get the first 3
  map_chr(~{
    http::GET(.x) %>% # get the URL (I'm lazily calling "fair use" here vs check robots.txt since I'm s
    http::content(as="text") %>% # extract the HTML
```

```

    jericho::html_to_text() %>% # strip away extraneous HTML tags
    LSafun::genericSummary(k=3) %>% # summarise!
    paste0(collapse="\n\n") # easier to see
  }) %>%
  walk(cat)

```

```

## Continue reading the main story Advertisement Continue reading the main story The North Korea tweet near
##
## LEARN MORE » Sections Home Search Skip to content Skip to navigation View mobile version The New York Time
##
## He called for an aide to Hillary Clinton to be thrown in jail, threatened to cut off aid to Pakistan and th
and in the inevitable moments we fall short, we will continue to own up to our mistakes, and we'll strive to d
##
## We will continue to put the fairness and accuracy of everything we publish above all else -
and in the inevitable moments we fall short, we will continue to own up to our mistakes, and we'll strive to d
##
## Our report is stronger than ever, thanks to investments in new forms of journalism like interactive graph
##
## Trump essentially calls it fake, making no effort to pretend to be above it all, except to boast that he is
##
## "The hope would be that given the American people's reaction to the way he's handled the presidency, the p

```

12.4 See Also

As noted, there are other NLP packages. Check out the CRAN Task View on NLP for more resources.

Chapter 13

Harvesting Friends and Followers

13.1 Problem

You want to harvest all of the friends or followers for a particular user.

13.2 Solution

Use `rtweet::get_followers()` or `rtweet::get_friends()`.

13.3 Discussion

The aforementioned `rtweet` functions give us all the data we need and handle pagination and rate-limits.

Let's see who Brooke Anderson follows and who follows her. She's an *incredibly talented* data scientist, weather expert and educator. We'll pull her followers and friends and work with her data a bit more in future recipes.

```
library(rtweet)
library(tidyverse)
```

```
(brooke_followers <- rtweet::get_followers("gbwanderson"))
```

```
## # A tibble: 256 x 1
##       user_id
##       <chr>
## 1 913819461727195138
## 2      25819761
## 3    2198622000
## 4    59655036
## 5 769616000593428480
## 6    2973406683
## 7    73603242
## 8    2790116012
## 9    392787202
## 10 920639877397364736
## # ... with 246 more rows
```

```
(brooke_friends <- rtweet::get_friends("gbwanderson"))
```

```
## # A tibble: 103 x 2
##       user          user_id
##   <chr>         <chr>
## 1 gbwanderson    3230388598
## 2 gbwanderson 776596392559177728
## 3 gbwanderson    1715370056
## 4 gbwanderson    131498466
## 5 gbwanderson    97464922
## 6 gbwanderson    363210621
## 7 gbwanderson    17203405
## 8 gbwanderson 910392773081104384
## 9 gbwanderson    91333167
## 10 gbwanderson   1568606814
## # ... with 93 more rows
```

13.4 See Also

- Official Twitter API documentation on friends and followers.

Chapter 14

Performing Setwise Operations on Friendship Data

14.1 Problem

You want to operate on collections of friends and followers to answer questions such as “*Who isn’t following me back?*”, “*Who are my mutual friends?*”, and “*What friends/followers do certain users have in common?*”.

14.2 Solution

Use R setwise operations and `rtweet::lookup_friendships()`.

14.3 Discussion

R has set operations and they’ll do *just fine* for helping us cook this recipe.

If you need a refresher on set operations, check out this introductory lesson from Khan Academy.

```
library(rtweet)
library(tidyverse)
```

```
brooke_followers <- rtweet::get_followers("gbwanderson")
brooke_friends <- rtweet::get_friends("gbwanderson")
```

Now we can see the count of mutual and disparate relationships:

```
# common
length(intersect(brooke_followers$user_id, brooke_friends$user_id))
```

```
## [1] 50
```

```
# diff
length(setdiff(brooke_followers$user_id, brooke_friends$user_id))
```

```
## [1] 206
```

The Python counterpart to this cookbook suggests Redis as a “big-ish” data solution for performing set operations at-scale. R has at least 3 packages that provide direct support for Redis, so if you need to

perform these operations at-scale, cache the info you retrieve from the Twitter API into Redis and then go crazy!

14.4 See Also

- Google (yes, seriously) `redis packages r` to see the impressive/diverse number of packages linking R to Redis
- Official Twitter API documentation on friends and followers.

Chapter 15

Resolving User Profile Information

15.1 Problem

You have a collection of ids and need to resolve basic profile information (such as screen names) for these users.

15.2 Solution

Use `rtweet::lookup_users()`.

15.3 Discussion

The `rtweet` interface to the Twitter API makes this task very straightforward.

```
library(rtweet)
library(tidyverse)
```

```
rstats <- rtweet::search_tweets("#rstats", n=30)

(recent_rtweeters <- lookup_users(unique(rstats$user_id)))
```

```
## # A tibble: 29 x 20
##       user_id          name  screen_name
##       <chr>          <chr>    <chr>
## 1    114258616 Daniela Vázquez d4tagirl
## 2    1221366938  Francesca    fdg10371
## 3    535209125   Giora Simchoni GioraSimchoni
## 4    850095740 Samantha Oliver limnoliver
## 5    20444825   Stephen Turner strnr
## 6    354581685   Jan Mulkens   JanMulkens
## 7    14773239   Albert Cardona albertcardona
## 8    433881702   Silas Tolliver esttres
## 9 731230965700251648 Mauna Dasari  chumblebiome
## 10 2872304195 John Stanton-Geddes jsgdatsci
## # ... with 19 more rows, and 17 more variables: location <chr>,
## #   description <chr>, url <chr>, protected <lgl>, followers_count <int>,
```

```
## # friends_count <int>, listed_count <int>, statuses_count <int>,
## # favourites_count <int>, account_created_at <dtm>, verified <lgl>,
## # profile_url <chr>, profile_expanded_url <chr>, account_lang <chr>,
## # profile_banner_url <chr>, profile_background_url <chr>,
## # profile_image_url <chr>
glimpse(recent_rtweeters)

## Observations: 29
## Variables: 20
## $ user_id          <chr> "114258616", "1221366938", "535209125",...
## $ name             <chr> "Daniela Vázquez", "Francesca", "Giora ...
## $ screen_name      <chr> "d4tagirl", "fdg10371", "GioraSimchoni"...
## $ location         <chr> "Montevideo, Uruguay", "Reading, UK", "...
## $ description      <chr> "Data Scientist at @IdathaUy, #NASADat...
## $ url              <chr> "https://t.co/zqjTY3aq9W", NA, "https:/...
## $ protected        <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
## $ followers_count  <int> 1086, 31, 174, 493, 15894, 665, 1192, 2...
## $ friends_count    <int> 276, 35, 77, 519, 373, 364, 142, 107, 9...
## $ listed_count     <int> 29, 0, 4, 9, 625, 302, 33, 0, 48, 7, 92...
## $ statuses_count   <int> 2057, 46, 119, 491, 14314, 4909, 2682, ...
## $ favourites_count <int> 6012, 108, 269, 843, 157, 1499, 2384, 1...
## $ account_created_at <dtm> 2010-02-14 18:57:42, 2013-02-26 11:49:...
## $ verified         <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALS...
## $ profile_url      <chr> "https://t.co/zqjTY3aq9W", NA, "https:/...
## $ profile_expanded_url <chr> "http://d4tagirl.com", NA, "http://gior...
## $ account_lang     <chr> "en", "en", "en", "en", "en", "en", "en...
## $ profile_banner_url <chr> "https://pbs.twimg.com/profile_banners/...
## $ profile_background_url <chr> "http://abs.twimg.com/images/themes/the...
## $ profile_image_url <chr> "http://pbs.twimg.com/profile_images/93..."
```

15.4 See Also

- Official Twitter API documentation on users.

Chapter 16

Analyzing the Authors of Tweets that Appear in Search Results

16.1 Problem

You want to analyze user profile information as it relates to the authors of tweets that appear in search results.

16.2 Solution

Covered in Recipe 15!

Chapter 17

Visualizing Geodata with a Dorling Cartogram

17.1 Problem

You want to visualize geolocation information (for example, the location field from user profile information, included in a batch of tweets such as a search query), in order to determine if there is a correlation between location and some other criterion.

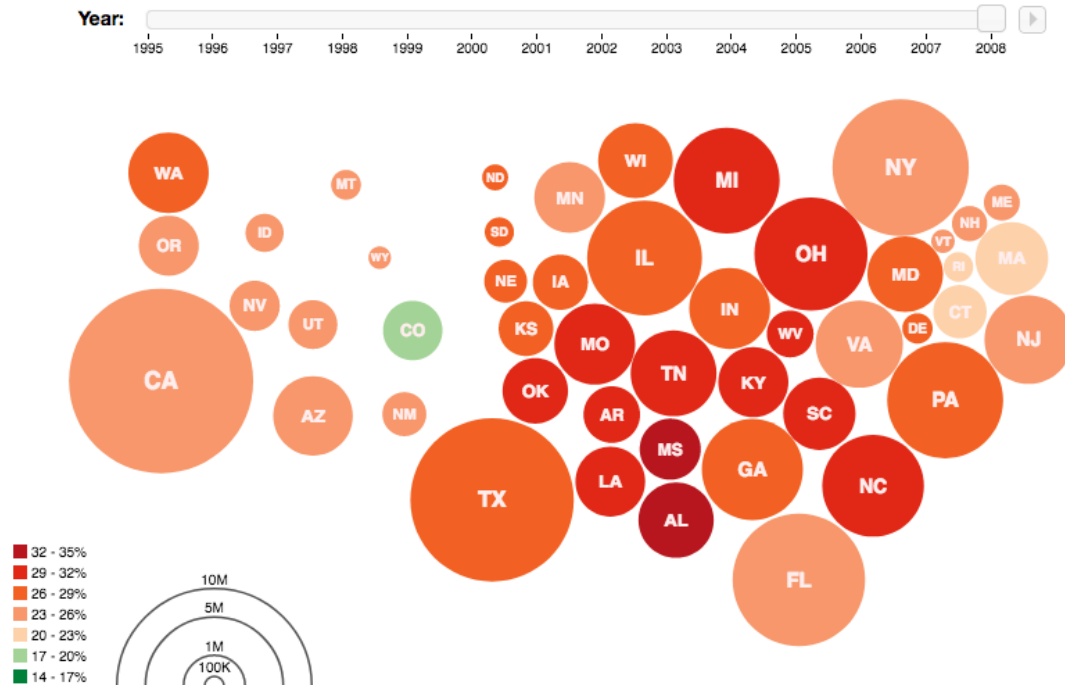
17.2 Solution

Devise a heuristic to extract the state from the location information in user profiles and visualize it with a Dorling Cartogram.

17.3 Discussion

(As this cookbook’s Python counterpart notes): *“A Dorling Cartogram is essentially a bubble chart where each bubble corresponds to a geographic area such as a state, and each bubble is situated as close as possible to its actual location on a map without overlapping with any other bubbles. Since the size and/or color of each bubble can be used to represent meaningful things, a Dorling Cartogram can give you a very intuitive view of data as it relates to geographic boundaries or regions of a larger land mass. The Protovis toolkit comes with some machinery for creating Dorling Cartograms for locations in the United States, and one interesting example of something that you could do builds upon Recipe 19, which demonstrated an approach that you could use to analyze the users who authored tweets from a targeted query.”*

One seminal, modern example for Dorling cartograms is Mike Bostock’s Protovis (a pre-cursor to D3) version



I have a love/hate relationship with cartograms. On the one hand, they capture attention by preying on the weakness most of us have to maps. On the other hand, most adults in my home country barely know where their state is on a map when they can see its shape, so most cartograms require extensive labeling and the shape distortions can initially disorient the viewer. Many cartograms also degrade the ability to readily discern the proportions or precision of the underlying data. But, we're working with Twitter data and the goal for this recipe is to pull a sample of tweets and see where the (geographic) action is at, so using this particular cartogram style for it is not a terrible choice.

NOTE: This recipe and the next (and, final!) recipe both cover Twitter and geographic data. Any reasonable person should disable the association of location information to Tweets for a whole host of reasons which include security and safety. They should also make the location information in their profile human discernable but difficult for machines to process (better still — use fake location data in your profile). However, there are still a cadre of Twitter users who gleefully provide this information — or, disinformation — so we all can process it.

Rather than rely on the geolocation data of a tweet, let's pull down some more `#rstats` tweets and try to limit the locations of the tweets to just the U.S. (despite disabling association of location data, Twitter can still internally, generally figure out where you're tweeting from, especially if you're on a mobile device).

After that, we'll lookup the users of the tweets and extract the location information from their user profile.

Once we have that location information, we'll filter it a bit to discern which state it came from. We'll cover the visualization component after we deal with the data:

```
library(rtweet)
library(broom)
library(eechidna)
library(cartogram) # chxy/cartogram
library(hrbrthemes)
library(tidyverse)

# search twitter for tweets
rstats_us <- search_tweets("#rstats", 3000, geocode = "2.877742,-97.380979,3000mi") # geocode request i
```

```
# lookup each user (uniquely) so we can grab location information
user_info <- lookup_users(unique(rstats_us$user_id))

discard(user_info$location, `==`, "") %>% # ignore blank data
str_match(sprintf("(%s)", paste0(state.abb, collapse="|"))) %>% # try to match U.S. state abbreviations
[,2] %>% # the previous step creates a matrix with column 2 being the extracted information (if any)
discard(is.na) %>% # if no state match was found the value is NA so discard this one
table() %>% # some habits are hard to break
broom::tidy() %>% # but we can tidy them!
set_names(c("state", "n")) %>% # these are more representative names
tbl_df() %>% # not really necessary but I was printing this when testing
arrange(desc(n)) %>% # same as ^~
left_join(
  as_data_frame(maps::state.carto.center) %>% # join state cartographic center data
  mutate(state=state.abb)
) %>%
# the GitHub-only cartogram package has a data structure which holds state adjacency information
# by specifying that here, it will help make the force-directed cartogram circle positioning more precise
filter(state %in% names(cartogram::statenbrs)) -> for_dor

glimpse(for_dor)
```

```
## Observations: 37
## Variables: 4
## $ state <chr> "NY", "MA", "PA", "CA", "NC", "IL", "TX", "MD", "FL", "W...
## $ n      <int> 32, 28, 17, 13, 12, 11, 10, 9, 8, 8, 6, 6, 6, 5, 5, 5, 4...
## $ x      <dbl> 8.041600, 9.025060, 7.222050, 1.410693, 6.998536, 4.8230...
## $ y      <dbl> 8.746802, 8.684993, 8.029811, 7.337543, 6.571104, 7.7578...
```

The visualization component needs some explanation since it's a bit hack-ish. Xiaoyue Cheng has had an R `cartogram` package on GitHub for just over five years. It's not feature complete but has a (mostly) working Dorling cartogram generator. Carson Sievert incorporated and enhanced some of the `cartogram` functions for the rOpenSci `eechidna` package, but uses it internally. Finally, `cartogram` package has some data sets that make Dorling U.S. state cartograms a bit more visually appealing.

What that all means is we'll be calling an unexported function from `eechidna` and using some data from `cartogram`. This is far from an optimal situation, especially since it also means we won't be using `ggplot2` for the final visualization. But, it works!

The code block below starts by setting up some base R plotting parameters, one of which — `col="white"` — is going to cause you some grief if you don't remember to change it to `black` since it impacts the default color for base R plots. This is also an opportunity to set the font family for the text labels since there is no way to pass text label aesthetics in the `dorling()` function.

We pass in the:

- state label
- state center
- a scaled value for the tweet count
- state neighbor information

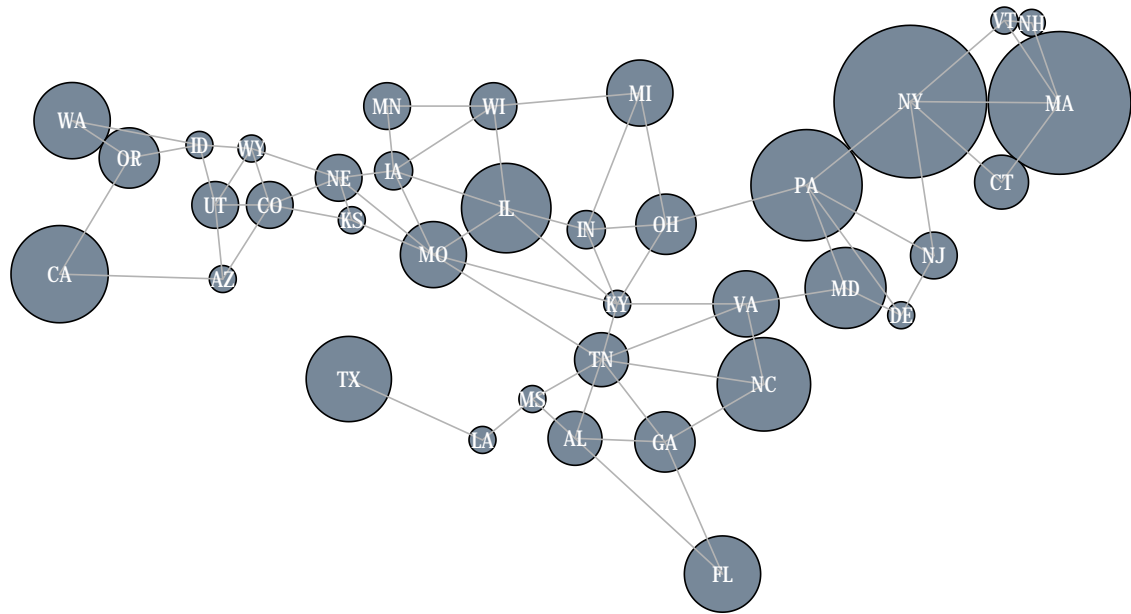
and tweak some aesthetics to produce the final plot. This example opted to “connect the dots” to more explicitly show neighbor information.

```
par(family=font_rc, col="white")

eechidna::dorling(
  for_dor$state, for_dor$x, for_dor$y, sqrt(for_dor$n), nbr=cartogram::statenbrs,
```

```
animation = FALSE, nbredge = TRUE, iteration=100, name.text=TRUE, dist.ratio=1.2,  
main="Dorling Cartogram of U.S. #rstats", xlab='', ylab='', col="lightslategray",  
frame=FALSE, asp=1, family=font_rc, cex.main=1.75, adj=0  
) -> dor
```

Dorling Cartogram of U.S. #rstats



Chapter 18

Geocoding Locations from Profiles (or Elsewhere)

18.1 Problem

You want to geocode information in tweets for situations beyond what the Twitter API provides and not just focus on U.S. states as Ecipe 20 did.

18.2 Solution

Use a geocoding service/package to translate location strings into more precise geographic information.

18.3 Discussion

Recipe 20 focused on extracting U.S. state information from user profiles. But, Twitter is a global service with millions of active users in many countries. Let's use the Google geocoding API function from the `ggmaps` package to try to translate user profile location strings into location data.

NOTE: Google's API has a limit of 2,500 calls per day for free, so you'll need to pay-up or work in daily batches if you have a large amount of Tweet location data to lookup.

```
library(rtweet)
library(ggmap)
library(tidyverse)

rstats_us <- search_tweets("#rstats", 3000)

user_info <- lookup_users(unique(rstats_us$user_id))

discard(user_info$location, `==`, "") %>%
  ggmap::geocode() -> coded

coded$location <- discard(user_info$location, `==`, "")

user_info <- left_join(user_info, coded, "location")
```

```
## # A tibble: 503 x 3
##           location      lat      lon
##           <chr>      <dbl>    <dbl>
## 1             Peru -9.189967 -75.015152
## 2 Richmond, B.C., Canada 49.166590 -123.133569
## 3      Massachusetts 42.407211 -71.382437
## 4      Frederick, MD 39.414269 -77.410541
## 5              Japan 36.204824 138.252924
## 6              FMU 34.190425 -79.651985
## 7      Chicago, IL 41.878114 -87.629798
## 8              36.204824 138.252924
## 9              43.072764 141.346112
## 10 Stuttgart, Germany 48.775846  9.182932
## 11      New York, NY 40.712775 -74.005973
## 12      Asbury Park, NJ 40.220391 -74.012082
## 13      Ann Arbor, MI 42.280826 -83.743038
## 14      Ithaca, NY 42.443961 -76.501881
## 15 ÛT: 36.1573208,-95.9526115 40.532392 -112.298984
## 16      Houston, TX 29.760427 -95.369803
## 17      Rome, NY 43.212847 -75.455730
## 18      Perth, Australia -31.950527 115.860457
## 19      Santiago, CL -33.448890 -70.669265
## 20      Johnston, IA 41.670983 -93.713049
## 21      Fort Collins, CO 40.585260 -105.084423
## 22      Hyderabad, India 17.385044  78.486671
## 23      Nashville, TN 36.162664 -86.781602
## 24      Canton, CHN 23.129110 113.264385
## 25      Bogotá  4.710989 -74.072092
## 26      3052, Australia -37.786236 144.947418
## 27      Charlottesville, VA 38.029306 -78.476678
## 28      Hobart, Tasmania -42.882138 147.327195
## 29              moon 40.516977 -80.221348
## 30      Toronto, Ontario 43.653226 -79.383184
## # ... with 473 more rows
```

18.4 See Also

Google's API is far from perfect, but they have also been collecting gnarly input data for map locations for over a decade, which makes them a good first-choice. You can find more R geocoding packages in the CRAN Web Technologies Task View.

Bibliography