#### **CHAPTER 1 :**

#### **Basics of Analyzing Twitter Data**

**Why Analyze Twitter Data?**

There are many reasons you may want to analyze Twitter data. Which of these is NOT an area of data science you could use analyzing Twitter data for?

**Answer the question**

**50 XP**

**Possible Answers**

* 

Analyzing the mentions of each political party in an election.

press1

* 

Detecting the reactions to the introduction of a new product.

press2

* 

Understanding the geographical scope of discussion of a news story.

press3

* 

Uncovering the motives of Twitter users following a hashtag. **(A)**

press4

Correct! You can't identify users unless they tweet.

**Uses of Twitter analysis**

You've been asked to identify the success (or failure) of a particular product. What Twitter analysis strategy could you use to best execute this?

**Answer the question**

**50 XP**

**Possible Answers**

* 

Collect mentions of the product and identify if people are talking about it positively.

press1

* 

Examine the size of the retweet network mentioning the product.

press2

* 

Analyzing the geographical penetration of users mentioning the product.

press3

* 

All of the above. **(A)**

press4

Correct! All of these are good ways of analyzing how a product may be received.

**Twitter APIs**

**True or False** : I could collect data from last year based on keyword searches with the Streaming API.

**Answer the question**

**50 XP**

**Possible Answers**

* 

True: The Streaming API allows historical data collection on keywords, user IDs, and locations.

press1

* 

False: The Streaming API only allows real-time data collection on ads.

press2

* 

False: The Streaming API only allows real-time data collection on keywords, user IDs, and locations. **(A)**

press3

* 

False: The Streaming API only allows access from the past week.

press4

Correct! The Streaming API only allows real-time data collection.

**Setting up tweepy authentication**

In the video, we saw how tweepy can be used to collect Twitter data with the Streaming API. tweepy requires a Twitter API key to authenticate with Twitter.

In this exercise, you will load several objects from tweepy and set up the authentication for the package.

The API keys access\_token, access\_token\_secret, consumer\_key, and consumer\_secret have already been defined for you.

**Instructions**

**100 XP**

* Import OAuthHandler and API from the tweepy module.
* Pass consumer\_key and consumer\_secret to OAuthHandler.
* Set the access tokens with access\_token and access\_token\_secret.
* Pass the auth object to the API.

from tweepy import OAuthHandler

from tweepy import API

# Consumer key authentication

auth = OAuthHandler(consumer\_key , consumer\_secret)

# Access key authentication

auth.set\_access\_token(access\_token , access\_token\_secret)

# Set up the API with the authentication handler

api = API(auth)

Great! You are now authenticated.

**Collecting data on keywords**

Now that we've set up the authentication, we can begin to collect Twitter data. Recall that with the Streaming API, we will be collecting real-time Twitter data based on either a sample or filtered by a keyword.

In our example, we will collect data on any tweet mentioning #rstats or #python in the tweet text, username, or user description with the filter endpoint.

The SListener module has already been defined and imported for you. You can find the full code for this module [**here**](https://github.com/SocialDataAnalytics-Winter2018/lab04/blob/master/slistener.py).

**Instructions**

**100 XP**

* Import Stream from tweepy.
* Set keywords\_to\_track to a list containing #rstats and #python.
* Pass the auth and listen objects to Stream.
* Set the keyword argument track equals to keywords\_to\_track.

from tweepy import Stream

# Set up words to track

keywords\_to\_track = ['#rstats' , '#python']

# Instantiate the SListener object

listen = SListener(api)

# Instantiate the Stream object

stream = Stream(auth , listen)

# Begin collecting data

stream.filter(track = keywords\_to\_track)

Good job! You are now collecting tweets.

**Loading and accessing tweets**

In the video, we loaded a tweet we collected using tweepy into Python. Tweets arrive from the Streaming API in JSON format and need to be converted into a Python data structure.

In this exercise, we'll load a single tweet into Python and print out some fields.

The tweet JSON has been loaded for you and is stored in tweet\_json.

**Instructions**

**100 XP**

* Import the json module.
* Convert the tweet JSON stored in tweet\_json from JSON to Python object using json's .loads() method.
* Print the tweet text and id using the appropriate keys.

# Load JSON

import json

# Convert from JSON to Python object

tweet = json.loads(tweet\_json)

# Print tweet text

print(tweet['text'])

# Print tweet id

print(tweet['id'])

<script.py> output:

Writing out the script of my @DataCamp class and I can't help but mentally read it back to myself in @hugobowne's voice.

986973961295720449

**Accessing user data**

Much of the data which we want to know about the Twitter data is stored in child JSON objects. We will access several parts of the user's information with the user child JSON object.

The tweet from the previous exercise has been loaded for you.

**Instructions**

**100 XP**

* Print the user handle with key screen\_name.
* Print the user follower count with key followers\_count.
* Print the user self-defined location with key location.
* Print the user self-defined description with key description.

# Print user handle

print(tweet['user']['screen\_name'])

# Print user follower count

print(tweet['user']['followers\_count'])

# Print user location

print(tweet['user']['location'])

# Print user description

print(tweet['user']['description'])

<script.py> output:

alexhanna

4267

Toronto, ON

Assistant professor @UofT. Protest, media, computation. Trans. Roller derby athlete @TOROLLERDERBY (Kate Silver #538). She/her

**Accessing retweet data**

Now we're going to work with a tweet JSON that contains a retweet. A retweet has the same structure as a regular tweet, except that it has another tweet stored in retweeted\_status.

The new tweet has been loaded as rt.

**Instructions**

**100 XP**

* Print the text of the tweet.
* Print the text of the tweet which has been retweeted, which is contained in retweeted\_status.
* Print the user handle of the tweet.
* Print the user handle of the tweet which has been retweeted, which is contained in retweeted\_status.

# Print the text of the tweet

print(rt['text'])

# Print the text of tweet which has been retweeted

print(rt['retweeted\_status']['text'])

# Print the user handle of the tweet

print(rt['user']['screen\_name'])

# Print the user handle of the tweet which has been retweeted

print(rt['retweeted\_status']['user']['screen\_name'])

<script.py> output:

RT @hannawallach: ICYMI: NIPS/ICML/ICLR are looking for a full-time programmer to run the conferences' submission/review processes. More in…

ICYMI: NIPS/ICML/ICLR are looking for a full-time programmer to run the conferences' submission/review processes. M… https://t.co/aB9Y5tTyHT

alexhanna

hannawallach

**CHAPTER 2 :**

#### **Processing Twitter text**

# Tweet Items and Tweet Flattening

There are multiple fields in the Twitter JSON which contains textual data. In a typical tweet, there's the tweet text, the user description, and the user location. In a tweet longer than 140 characters, there's the extended tweet child JSON. And in a quoted tweet, there's the original tweet text and the commentary with the quoted tweet.

For this exercise, you'll extract textual elements from a single quoted tweet in which the original tweet has more than 140 characters. Then, to analyze tweets at scale, we will want to **flatten** the tweet JSON into a single level. This will allow us to store the tweets in a DataFrame format.

quoted\_tweet has been loaded for you.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Print the tweet text.
* Print the quoted tweet text, that is, the text in quoted\_status.
* Print the quoted tweet's extended (140+ character) full\_text in extended\_tweet.
* Print the quoted tweet user's location.

**In [9]: quoted\_tweet['quoted\_status']['extended\_tweet']**

**Out[9]:**

**{'display\_text\_range': [0, 191],**

**'entities': {'hashtags': [], 'symbols': [], 'urls': [], 'user\_mentions': []},**

**'full\_text': 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou wilt not, be but sworn my love,\nAnd I’ll no longer be a 140 character tweet.'}**

# Print the tweet text

print(quoted\_tweet['text'])

# Print the quoted tweet text

print(quoted\_tweet['quoted\_status']['text'])

# Print the quoted tweet's extended (140+) text

print(quoted\_tweet['quoted\_status']['extended\_tweet']['full\_text'])

# Print the quoted user location

print(quoted\_tweet['quoted\_status']['user']['location'])

<script.py> output:

maybe if I quote tweet this lil guy https://t.co/BzbLDz9j6g

O 280 characters, 280 characters! Wherefore art thou 280 characters?

Deny thy JSON and refuse thy key.

Or, if thou… https://t.co/MlFg4qFnEC

O 280 characters, 280 characters! Wherefore art thou 280 characters?

Deny thy JSON and refuse thy key.

Or, if thou wilt not, be but sworn my love,

And I’ll no longer be a 140 character tweet.

Toronto, ON

##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))
* Store the user screen\_name in user-screen\_name.
* Store the quoted tweet's text in quoted\_status-text.
* Store the quoted tweet's extended text in quoted\_status-extended\_tweet-full\_text.

# Store the user screen\_name in 'user-screen\_name'

quoted\_tweet['user-screen\_name'] = quoted\_tweet['user']['screen\_name']

# Store the quoted\_status text in 'quoted\_status-text'

quoted\_tweet['quoted\_status-text'] = quoted\_tweet['quoted\_status']['text']

# Store the quoted tweet's extended (140+) text in

# 'quoted\_status-extended\_tweet-full\_text'

quoted\_tweet['quoted\_status-extended\_tweet-full\_text'] = quoted\_tweet['quoted\_status']['extended\_tweet']['full\_text']

<script.py> output:

maybe if I quote tweet this lil guy https://t.co/BzbLDz9j6g

O 280 characters, 280 characters! Wherefore art thou 280 characters?

Deny thy JSON and refuse thy key.

Or, if thou… https://t.co/MlFg4qFnEC

O 280 characters, 280 characters! Wherefore art thou 280 characters?

Deny thy JSON and refuse thy key.

Or, if thou wilt not, be but sworn my love,

And I’ll no longer be a 140 character tweet.

Toronto, ON

<script.py> output:

maybe if I quote tweet this lil guy https://t.co/BzbLDz9j6g

O 280 characters, 280 characters! Wherefore art thou 280 characters?

Deny thy JSON and refuse thy key.

Or, if thou… https://t.co/MlFg4qFnEC

O 280 characters, 280 characters! Wherefore art thou 280 characters?

Deny thy JSON and refuse thy key.

Or, if thou wilt not, be but sworn my love,

And I’ll no longer be a 140 character tweet.

Toronto, ON

In [1]: quoted\_tweet['user']['screen\_name']

Out[1]: 'alexhanna'

In [2]:

In [2]:

In [2]: quoted\_tweet['quoted\_status']

Out[2]:

{'contributors': None,

'coordinates': None,

'created\_at': 'Wed Apr 25 17:17:23 +0000 2018',

'entities': {'hashtags': [],

'symbols': [],

'urls': [{'display\_url': 'twitter.com/i/web/status/9…',

'expanded\_url': 'https://twitter.com/i/web/status/989191655759663105',

'indices': [116, 139],

'url': 'https://t.co/MlFg4qFnEC'}],

'user\_mentions': []},

'extended\_tweet': {'display\_text\_range': [0, 191],

'entities': {'hashtags': [], 'symbols': [], 'urls': [], 'user\_mentions': []},

'full\_text': 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou wilt not, be but sworn my love,\nAnd I’ll no longer be a 140 character tweet.'},

'favorite\_count': 1,

'favorited': False,

'filter\_level': 'low',

'geo': None,

'id': 989191655759663105,

'id\_str': '989191655759663105',

'in\_reply\_to\_screen\_name': None,

'in\_reply\_to\_status\_id': None,

'in\_reply\_to\_status\_id\_str': None,

'in\_reply\_to\_user\_id': None,

'in\_reply\_to\_user\_id\_str': None,

'is\_quote\_status': False,

'lang': 'en',

'place': None,

'quote\_count': 0,

'reply\_count': 1,

'retweet\_count': 0,

'retweeted': False,

'source': '<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>',

'text': 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou… https://t.co/MlFg4qFnEC',

'truncated': True,

'user': {'contributors\_enabled': False,

'created\_at': 'Thu Jan 18 20:37:52 +0000 2007',

'default\_profile': False,

'default\_profile\_image': False,

'description': 'Assistant professor @UofT. Protest, media, computation. Trans. Roller derby athlete @TOROLLERDERBY (Kate Silver #538). She/her.',

'favourites\_count': 23526,

'follow\_request\_sent': None,

'followers\_count': 4275,

'following': None,

'friends\_count': 2806,

'geo\_enabled': True,

'id': 661613,

'id\_str': '661613',

'is\_translator': False,

'lang': 'en',

'listed\_count': 246,

'location': 'Toronto, ON',

'name': 'Alex Hanna, Data Witch',

'notifications': None,

'profile\_background\_color': '000000',

'profile\_background\_image\_url': 'http://abs.twimg.com/images/themes/theme16/bg.gif',

'profile\_background\_image\_url\_https': 'https://abs.twimg.com/images/themes/theme16/bg.gif',

'profile\_background\_tile': False,

'profile\_banner\_url': 'https://pbs.twimg.com/profile\_banners/661613/1524231456',

'profile\_image\_url': 'http://pbs.twimg.com/profile\_images/980799823900180483/J9CDOX\_X\_normal.jpg',

'profile\_image\_url\_https': 'https://pbs.twimg.com/profile\_images/980799823900180483/J9CDOX\_X\_normal.jpg',

'profile\_link\_color': '0671B8',

'profile\_sidebar\_border\_color': '666666',

'profile\_sidebar\_fill\_color': 'CCCCCC',

'profile\_text\_color': '333333',

'profile\_use\_background\_image': False,

'protected': False,

'screen\_name': 'alexhanna',

'statuses\_count': 71925,

'time\_zone': 'Eastern Time (US & Canada)',

'translator\_type': 'regular',

'url': 'http://alex-hanna.com',

'utc\_offset': -14400,

'verified': False}}

In [3]:

In [3]:

In [3]: quoted\_tweet['quoted\_status']['text']

Out[3]: 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou… https://t.co/MlFg4qFnEC'

In [4]: quoted\_tweet['extended\_text']

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

quoted\_tweet['extended\_text']

KeyError: 'extended\_text'

In [5]:

In [5]:

In [5]: quoted\_tweet['quoted\_status']

Out[5]:

{'contributors': None,

'coordinates': None,

'created\_at': 'Wed Apr 25 17:17:23 +0000 2018',

'entities': {'hashtags': [],

'symbols': [],

'urls': [{'display\_url': 'twitter.com/i/web/status/9…',

'expanded\_url': 'https://twitter.com/i/web/status/989191655759663105',

'indices': [116, 139],

'url': 'https://t.co/MlFg4qFnEC'}],

'user\_mentions': []},

'extended\_tweet': {'display\_text\_range': [0, 191],

'entities': {'hashtags': [], 'symbols': [], 'urls': [], 'user\_mentions': []},

'full\_text': 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou wilt not, be but sworn my love,\nAnd I’ll no longer be a 140 character tweet.'},

'favorite\_count': 1,

'favorited': False,

'filter\_level': 'low',

'geo': None,

'id': 989191655759663105,

'id\_str': '989191655759663105',

'in\_reply\_to\_screen\_name': None,

'in\_reply\_to\_status\_id': None,

'in\_reply\_to\_status\_id\_str': None,

'in\_reply\_to\_user\_id': None,

'in\_reply\_to\_user\_id\_str': None,

'is\_quote\_status': False,

'lang': 'en',

'place': None,

'quote\_count': 0,

'reply\_count': 1,

'retweet\_count': 0,

'retweeted': False,

'source': '<a href="https://about.twitter.com/products/tweetdeck" rel="nofollow">TweetDeck</a>',

'text': 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou… https://t.co/MlFg4qFnEC',

'truncated': True,

'user': {'contributors\_enabled': False,

'created\_at': 'Thu Jan 18 20:37:52 +0000 2007',

'default\_profile': False,

'default\_profile\_image': False,

'description': 'Assistant professor @UofT. Protest, media, computation. Trans. Roller derby athlete @TOROLLERDERBY (Kate Silver #538). She/her.',

'favourites\_count': 23526,

'follow\_request\_sent': None,

'followers\_count': 4275,

'following': None,

'friends\_count': 2806,

'geo\_enabled': True,

'id': 661613,

'id\_str': '661613',

'is\_translator': False,

'lang': 'en',

'listed\_count': 246,

'location': 'Toronto, ON',

'name': 'Alex Hanna, Data Witch',

'notifications': None,

'profile\_background\_color': '000000',

'profile\_background\_image\_url': 'http://abs.twimg.com/images/themes/theme16/bg.gif',

'profile\_background\_image\_url\_https': 'https://abs.twimg.com/images/themes/theme16/bg.gif',

'profile\_background\_tile': False,

'profile\_banner\_url': 'https://pbs.twimg.com/profile\_banners/661613/1524231456',

'profile\_image\_url': 'http://pbs.twimg.com/profile\_images/980799823900180483/J9CDOX\_X\_normal.jpg',

'profile\_image\_url\_https': 'https://pbs.twimg.com/profile\_images/980799823900180483/J9CDOX\_X\_normal.jpg',

'profile\_link\_color': '0671B8',

'profile\_sidebar\_border\_color': '666666',

'profile\_sidebar\_fill\_color': 'CCCCCC',

'profile\_text\_color': '333333',

'profile\_use\_background\_image': False,

'protected': False,

'screen\_name': 'alexhanna',

'statuses\_count': 71925,

'time\_zone': 'Eastern Time (US & Canada)',

'translator\_type': 'regular',

'url': 'http://alex-hanna.com',

'utc\_offset': -14400,

'verified': False}}

In [6]: quoted\_tweet['quoted\_status']['text']

Out[6]: 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou… https://t.co/MlFg4qFnEC'

In [7]: quoted\_tweet['quoted\_status']['full\_text']

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

quoted\_tweet['quoted\_status']['full\_text']

KeyError: 'full\_text'

In [8]: quoted\_tweet['quoted\_status']['text']

Out[8]: 'O 280 characters, 280 characters! Wherefore art thou 280 characters?\nDeny thy JSON and refuse thy key.\nOr, if thou… https://t.co/MlFg4qFnEC'

Traceback (most recent call last):

File "script.py", line 9, in <module>

quoted\_tweet['quoted\_status-extended\_tweet-full\_text'] = quoted\_tweet['quoted\_status']['extended\_tweet']['text']

KeyError: 'text'

**A tweet flattening function**

We are typically interested in hundreds or thousands of tweets. For this purpose, it makes sense to define a function to flatten JSON file full of tweets. Let's call this function flatten\_tweets(). We will use this function multiple times in this course and change it slightly as we deal with different types of data.

json has been loaded for you.

**Instructions**

**100 XP**

* Store the user screen name in user-screen\_name.
* Store the extended tweet text in extended\_tweet-full\_text.
* Store the retweet user screen name in retweeted\_status-user-screen\_name.
* Store the retweet text in retweeted\_status-text.

def flatten\_tweets(tweets\_json):

""" Flattens out tweet dictionaries so relevant JSON

is in a top-level dictionary."""

tweets\_list = []

# Iterate through each tweet

for tweet in tweets\_json:

tweet\_obj = json.loads(tweet)

# Store the user screen name in 'user-screen\_name'

tweet\_obj['user-screen\_name'] = tweet\_obj['user']['screen\_name']

# Check if this is a 140+ character tweet

if 'extended\_tweet' in tweet\_obj:

# Store the extended tweet text in 'extended\_tweet-full\_text'

tweet\_obj['extended\_tweet-full\_text'] = tweet\_obj['extended\_tweet']['full\_text']

if 'retweeted\_status' in tweet\_obj:

# Store the retweet user screen name in 'retweeted\_status-user-screen\_name'

tweet\_obj['retweeted\_status-user-screen\_name'] = tweet\_obj['retweeted\_status']['user']['screen\_name']

# Store the retweet text in 'retweeted\_status-text'

tweet\_obj['retweeted\_status-text'] = tweet\_obj['retweeted\_status']['text']

tweets\_list.append(tweet\_obj)

return tweets\_list

Great! Now we can flatten the JSON.

# Loading tweets into a DataFrame

Now it's time to import data into a pandas DataFrame so we can analyze tweets at scale.

We will work with a dataset of tweets which contain the hashtag '#rstats' or '#python'. This dataset is stored as a list of tweet JSON objects in data\_science\_json.

This course touches on a lot of concepts you may have forgotten, so if you ever need a quick refresher, download the [**pandas basics Cheat Sheet**](https://datacamp-community-prod.s3.amazonaws.com/fbc502d0-46b2-4e1b-b6b0-5402ff273251) and keep it handy!

Be aware that this is real data from Twitter and as such there is always a risk for the presence of profanity or other offensive content (in this exercise, and any following exercises that also use real Twitter data).

##### Instructions

**100 XP**

* Import pandas (remember, by convention we'll alias it as pd).
* Flatten the data\_science\_json tweets with flatten\_tweets() and store them in tweets.
* Create a DataFrame from tweets using pd.DataFrame().
* Print out the text from the first 5 tweets.

In [1]: data\_science\_json

Out[1]:

['{"created\_at":"Fri Mar 30 13:04:22 +0000 2018","id":979705897457942528,"id\_str":"979705897457942528","text":"RT @Dennboss: Hahahah Efteling maakt Maxi-Cosi\'s voor in de Python, duidelijk een perfect uitgewerkte 1 april grap en toch zijn er van die\\u2026","source":"\\u003ca href=\\"http:\\/\\/twitter.com\\/download\\/android\\" re

……… etc

# Import pandas

import pandas as pd

# Flatten the tweets and store in `tweets`

tweets = flatten\_tweets(data\_science\_json)

# Create a DataFrame from `tweets`

ds\_tweets = pd.DataFrame(tweets)

# Print out the first 5 tweets from this dataset

print(ds\_tweets['text'].values[0:5])

<script.py> output:

["RT @Dennboss: Hahahah Efteling maakt Maxi-Cosi's voor in de Python, duidelijk een perfect uitgewerkte 1 april grap en toch zijn er van die…"

'RT @PythonWeekly: Python Weekly - Issue 338 https://t.co/7gJSoLJj3V #python #django #flask #slack #blockchain #bitcoin #twilio #opencv #ma…'

'RT @dataandme: ICYMI, still 💜ing this: "Where do things live in R? R for Excel Users" by @StephdeSilva https://t.co/WiIIxFZzRX #rstats http…'

'RT @dataandme: 🕴@jaredlander knows how to put on a show…\n"New York R Conference" is gonna be 🔥\nhttps://t.co/BpFTYm1Peh via @rstatsnyc (duh)…'

'RT @llanga: I heard it\'s Py Day today so I made a thing! Meet "Black", the uncompromising #python code formatter!\n\nhttps://t.co/COA0YMfUNr…']

In [3]: ds\_tweets.head()

Out[3]:

contributors coordinates created\_at display\_text\_range \

0 None None Fri Mar 30 13:04:22 +0000 2018 NaN

1 None None Fri Mar 16 11:59:09 +0000 2018 NaN

2 None None Tue Mar 27 08:34:33 +0000 2018 NaN

3 None None Fri Mar 16 21:26:58 +0000 2018 NaN

4 None None Thu Mar 15 23:35:05 +0000 2018 NaN

entities extended\_entities \

0 {'user\_mentions': [{'screen\_name': 'Dennboss',... NaN

1 {'user\_mentions': [{'screen\_name': 'PythonWeek... NaN

2 {'user\_mentions': [{'screen\_name': 'dataandme'... NaN

3 {'user\_mentions': [{'screen\_name': 'dataandme'... NaN

4 {'user\_mentions': [{'screen\_name': 'llanga', '... NaN

extended\_tweet extended\_tweet-full\_text favorite\_count favorited \

0 NaN NaN 0 False

1 NaN NaN 0 False

2 NaN NaN 0 False

3 NaN NaN 0 False

4 NaN NaN 0 False

... retweeted\_status \

0 ... {'text': 'Hahahah Efteling maakt Maxi-Cosi's v...

1 ... {'text': 'Python Weekly - Issue 338 https://t....

2 ... {'text': 'ICYMI, still 💜ing this: "Where do th...

3 ... {'text': '🕴@jaredlander knows how to put on a ...

4 ... {'text': 'I heard it's Py Day today so I made ...

retweeted\_status-extended\_tweet-full\_text \

0 Hahahah Efteling maakt Maxi-Cosi's voor in de ...

1 Python Weekly - Issue 338 https://t.co/7gJSoLJ...

2 ICYMI, still 💜ing this: "Where do things live ...

3 🕴@jaredlander knows how to put on a show…\n"Ne...

4 I heard it's Py Day today so I made a thing! M...

retweeted\_status-text \

0 Hahahah Efteling maakt Maxi-Cosi's voor in de ...

1 Python Weekly - Issue 338 https://t.co/7gJSoLJ...

2 ICYMI, still 💜ing this: "Where do things live ...

3 🕴@jaredlander knows how to put on a show…\n"Ne...

4 I heard it's Py Day today so I made a thing! M...

retweeted\_status-user-screen\_name \

0 Dennboss

1 PythonWeekly

2 dataandme

3 dataandme

4 llanga

source \

0 <a href="http://twitter.com/download/android" ...

1 <a href="http://twitter.com/download/iphone" r...

2 <a href="http://twitter.com/download/iphone" r...

3 <a href="http://twitter.com/download/android" ...

4 <a href="http://twitter.com/download/iphone" r...

text timestamp\_ms \

0 RT @Dennboss: Hahahah Efteling maakt Maxi-Cosi... 1522415062666

1 RT @PythonWeekly: Python Weekly - Issue 338 ht... 1521201549661

2 RT @dataandme: ICYMI, still 💜ing this: "Where ... 1522139673666

3 RT @dataandme: 🕴@jaredlander knows how to put ... 1521235618658

4 RT @llanga: I heard it's Py Day today so I mad... 1521156905660

truncated user \

0 False {'profile\_sidebar\_border\_color': '000000', 'de...

1 False {'profile\_sidebar\_border\_color': '000000', 'de...

2 False {'profile\_sidebar\_border\_color': '000000', 'de...

3 False {'profile\_sidebar\_border\_color': '000000', 'de...

4 False {'profile\_sidebar\_border\_color': '181A1E', 'de...

user-screen\_name

0 mlvttweet

1 testdrivenio

2 drchriscole

3 sellorm

4 SimplicityGuy

[5 rows x 40 columns]

**Finding keywords**

Counting known keywords is one of the first ways you can analyze text data in a Twitter dataset. In this dataset, you're going to count the number of times specific hashtags occur in a collection of tweets about data science. To this end, you're going to use the string methods in the pandas Series object to do this.

pandas and numpy have been imported as pd and np, respectively. A more fully-featured flatten\_tweets and data\_science\_json have also been loaded for you.

**Instructions**

**100 XP**

* Flatten the tweets with flatten\_tweets() and store them in flat\_tweets.
* Convert tweets to DataFrame using the pandas DataFrame constructor.
* Find mentions of #python in 'text', ignoring case.
* Print proportion of tweets mentioning #python by summing python with np.sum() and dividing it by the total number of tweets.

# Flatten the tweets and store them

flat\_tweets = flatten\_tweets(data\_science\_json)

# Convert to DataFrame

ds\_tweets = pd.DataFrame(flat\_tweets)

# Find mentions of #python in 'text'

python = ds\_tweets['text'].str.contains('#python', case = False)

# Print proportion of tweets mentioning #python

print("Proportion of #python tweets:", np.sum(python) / len(ds\_tweets))

<script.py> output:

Proportion of #python tweets: 0.44533333333333336

**Looking for text in all the wrong places**

Recall that relevant text may not only be in the main text field of the tweet. It may also be in the extended\_tweet, the retweeted\_status, or the quoted\_status. We need to check all of these fields to make sure we've accounted for all the of the relevant text. We'll do this often so we're going to create a function which does this.

The first two lines check if the main text field or the extended\_tweet contain the text. You will need to check the rest.

**Instructions**

**100 XP**

Finish the check\_word\_in\_tweet function by doing the following:

* Check if the field quoted\_status-text contains the word.
* Check if the field quoted\_status-extended\_tweet-full\_text contains the word.
* Check if the field retweeted\_status-text contains the word.
* Check if the field retweeted\_status-extended\_tweet-full\_text contains the word.

def check\_word\_in\_tweet(word, data):

"""Checks if a word is in a Twitter dataset's text.

Checks text and extended tweet (140+ character tweets) for tweets,

retweets and quoted tweets.

Returns a logical pandas Series.

"""

contains\_column = data['text'].str.contains(word, case = False)

contains\_column |= data['extended\_tweet-full\_text'].str.contains(word, case = False)

contains\_column |= data['quoted\_status-text'].str.contains(word, case = False)

contains\_column |= data['quoted\_status-extended\_tweet-full\_text'].str.contains(word , case = False)

contains\_column |= data['retweeted\_status-text'].str.contains(word , case = False)

contains\_column |= data['retweeted\_status-extended\_tweet-full\_text'].str.contains(word , case = False)

return contains\_column

Great! Now you have a function that searces all text fields.

**Comparing #python to #rstats**

Now that we have a function to check whether or not the word is in the tweet in multiple places, we can deploy this across multiple words and compare them. Let's return to our example with the data science hashtag dataset. We want to see how many times that #rstats occurs compared to #python.

The data science hashtag dataset ds\_tweets has been loaded for you.

**Instructions**

**100 XP**

* Use the function check\_word\_in\_tweet() to find all instances of #python in the text fields of ds\_tweets.
* Do the same with #rstats.
* Print proportion of tweets mentioning #python by summing python with np.sum() and dividing it by ds\_tweets.shape[0].
* Do the same for rstats.

# Find mentions of #python in all text fields

python = check\_word\_in\_tweet('#python', ds\_tweets)

# Find mentions of #rstats in all text fields

rstats = check\_word\_in\_tweet('#rstats', ds\_tweets)

# Print proportion of tweets mentioning #python

print("Proportion of #python tweets:", np.sum(python) / len(ds\_tweets))

# Print proportion of tweets mentioning #rstats

print("Proportion of #rstats tweets:", np.sum(rstats) / len(ds\_tweets))

<script.py> output:

Proportion of #python tweets: 0.5733333333333334

Proportion of #rstats tweets: 0.4693333333333333

**Creating time series data frame**

Time series data is used when we want to analyze or explore variation over time. This is useful when exploring Twitter text data if we want to track the prevalence of a word or set of words.

The first step in doing this is converting the DataFrame into a format which can be handled using pandas time series methods. That can be done by converting the index to a datetime type.

**Instructions**

**100 XP**

* Print the first five rows of created\_at in ds\_tweets with the .head() method.
* Convert that column to a datetime type with the Pandas' .to\_datetime() method.
* Print the first five rows once again.
* Set index to created\_at with .set\_index().

# Print created\_at to see the original format of datetime in Twitter data

print(ds\_tweets['created\_at'].head())

# Convert the created\_at column to np.datetime object

ds\_tweets['created\_at'] = pd.to\_datetime(ds\_tweets['created\_at'])

# Print created\_at to see new format

print(ds\_tweets['created\_at'].head())

# Set the index of ds\_tweets to created\_at

ds\_tweets = ds\_tweets.set\_index('created\_at')

<script.py> output:

0 Fri Mar 30 13:04:22 +0000 2018

1 Fri Mar 16 11:59:09 +0000 2018

2 Tue Mar 27 08:34:33 +0000 2018

3 Fri Mar 16 21:26:58 +0000 2018

4 Thu Mar 15 23:35:05 +0000 2018

Name: created\_at, dtype: object

0 2018-03-30 13:04:22

1 2018-03-16 11:59:09

2 2018-03-27 08:34:33

3 2018-03-16 21:26:58

4 2018-03-15 23:35:05

Name: created\_at, dtype: datetime64[ns]

**Generating mean frequency**

We need to produce a metric which can be graphed over time. Our function check\_word\_in\_tweet() returns a boolean Series. Remember that the boolean value True == 1, so we can produce a column for each keyword we're interested in and use it to understand its over time prevalence.

**Instructions**

**100 XP**

* Create a column called python and store the results of check\_word\_in\_tweet() for the string '#python' in it.
* Do the same, but with a column called rstats and the string '#rstats'.

# Create a python column

ds\_tweets['python'] = check\_word\_in\_tweet('#python', ds\_tweets)

# Create an rstats column

ds\_tweets['rstats'] = check\_word\_in\_tweet('#rstats', ds\_tweets)

**Plotting mean frequency**

Lastly, we'll create a per-day average of the mentions of both hashtags and plot them across time. We'll first create proportions from the two boolean Series by the day, then we'll plot them.

matplotlib.pyplot has been imported as plt and ds\_tweets has been loaded for you.

**Instructions**

**100 XP**

* Generate the mean number of tweets with the python column with .resample() and .mean() methods. .resample() takes one argument, '1 d', to produce daily averages.
* Do the same with the rstats column.
* Plot a line for #python usage with mean\_python. Use mean\_python.index.day as the x-axis.
* Do the same with mean\_rstats.

# Average of python column by day

mean\_python = ds\_tweets['python'].resample('1 d').mean()

# Average of rstats column by day

mean\_rstats = ds\_tweets['rstats'].resample('1 d').mean()

# Plot mean python by day(green)/mean rstats by day(blue)

plt.plot(mean\_python.index.day , mean\_python, color = 'green')

plt.plot(mean\_rstats.index.day , mean\_rstats, color = 'blue')

# Add labels and show

plt.xlabel('Day'); plt.ylabel('Frequency')

plt.title('Language mentions over time')

plt.legend(('#python', '#rstats'))

plt.show()

**Loading VADER**

Sentiment analysis provides us a small glimpse of the meaning of texts with a rather directly interpretable method. While it has its limitations, it's a good place to begin working with textual data. There's a number of out-of-the-box tools in Python we can use for sentiment analysis.

ds\_tweets with the datetime index has been loaded for you.

**Instructions**

**100 XP**

* Load SentimentIntensityAnalyzer from nltk.sentiment.vader.
* Instantiate a new SentimentIntensityAnalyzer object.
* Generate sentiment scores with the .apply() method and the analyzer's polarity\_scores() function.

In [1]: ds\_tweets['text']

Out[1]:

created\_at

2018-03-30 13:04:22 RT @Dennboss: Hahahah Efteling maakt Maxi-Cosi...

2018-03-16 11:59:09 RT @PythonWeekly: Python Weekly - Issue 338 ht...

2018-03-27 08:34:33 RT @dataandme: ICYMI, still 💜ing this: "Where ...

2018-03-16 21:26:58 RT @dataandme: 🕴@jaredlander knows how to put ...

2018-03-15 23:35:05 RT @llanga: I heard it's Py Day today so I mad...

2018-03-25 01:00:25 #Spectrum My #InternetSpeed :\nPing: 34.319 ms...

2018-03-09 15:59:42 RT @blarson424: R: linear regression https://t...

2018-03-29 08:29:00 RT @CMastication: using #Python and #rstats in...

2018-03-24 10:00:53 RT @jrlarsen: Go Teams! 40% off at @ManningBoo...

2018-03-26 17:04:23 RT @dataandme: 📉🎞 head-to-head, w/ code:\n"gga...

2018-03-13 22:21:43 RT @dataandme: ICYMI, 👍 series on useful 📦s!\n...

2018-03-29 16:23:18 RT @PythonWeekly: Python Weekly - Issue 340 ht...

2018-03-03 16:00:41 #Spectrum My #InternetSpeed :\nPing: 34.611 ms...

2018-03-26 15:33:52 RT @rstudio: reticulate: R interface to Python...

2018-03-12 12:22:36 Read here Top 10 #technology #javascript #pyth...

2018-03-05 03:18:58 An empty bag is an island. #proverb #python

2018-03-14 18:22:49 A #sloth #elephants #giraffes #rhinos #monkeys...

2018-03-06 00:46:06 RT @dataandme: 'nother ⭐️tidy eval post by @ro...

2018-03-29 12:47:34 RT @PaulMinda1: Very cool tutorial on box plot...

2018-03-20 12:59:06 RT @rar\_21: Day 2 - Starting with #APICem #Dev...

2018-03-20 05:52:40 RT @blarson424: Intro to R Shiny: build a web ...

2018-03-04 08:15:14 RT @alistaire: A little tutorial for how to re...

2018-03-20 23:42:59 RT @sgrifter: #RLadies NYC Meetup tonight host...

2018-03-19 16:30:14 bar plot number of 0s in a dataframe. Fill bas...

2018-03-13 22:30:44 Pubs these days. And no Defenders on 5\* #pytho...

2018-03-20 17:45:09 This Hours Photo: #weather #minnesota #photo #...

2018-03-21 15:32:36 Связной список / бинарное дерево https://t.co/...

2018-03-20 12:59:59 RT @coolbutuseless: Adding checks to an existi...

2018-03-02 07:31:54 RT @ReactDOM: All the Best Rated Courses on sa...

2018-03-09 16:02:59 RT @Cruz\_Julian\_: Rbloggers Some Tools for Wri...

...

2018-03-09 12:52:58 RT @egreen460: https://t.co/zlhXALd8iD is For ...

2018-03-03 23:27:26 RT @elfosardo: day 47, added option to use tex...

2018-03-08 14:42:10 My new #rstats project:\n\nMake it so that whe...

2018-03-25 08:17:09 RT @ahmedjr\_16: Review of #MachineLearning A-Z...

2018-03-08 10:46:29 RT @reinforcelabtwt: Review of #MachineLearnin...

2018-03-12 17:49:10 RT @Tchai1812: I love @TwitchCreates so I did ...

2018-03-06 18:51:27 RT @rob\_arista: 🖍🖍🖍 HSL Color Diagram 🖍🖍🖍\nmad...

2018-03-10 14:54:02 Obtain both headers and content from single GE...

2018-03-06 22:04:43 RT @mvachhar: Improve your code, save time, do...

2018-03-09 02:19:46 After a week learning python from scratch, ver...

2018-03-12 21:07:27 This #rstats course needs some marketing. Awes...

2018-03-06 09:58:00 Tableau in Practice - A Step by Step Complete ...

2018-03-08 22:35:57 It would be great if #postgres, #R, and #pytho...

2018-03-30 20:39:02 Extract first column of a 2D list along with i...

2018-03-18 20:05:32 Utiliza #python con tu #arduino: Tutorial de A...

2018-03-03 22:40:39 RT @kpavlovsky\_pro: Right before I went to sle...

2018-03-12 11:39:58 RT @hadleywickham: I've written a brand new ch...

2018-03-22 19:46:17 RT @MattDowle: And big-time thanks to @krlmlr ...

2018-03-21 23:51:03 RT @OCCRP: Got skills with #linux, #python, ma...

2018-03-04 18:00:39 #Spectrum My #InternetSpeed :\nPing: 31.845 ms...

2018-03-20 14:53:18 One more event conducted successfully. We orga...

2018-03-22 11:14:50 RT @python\_devv: Python Tutorial: Creating Web...

2018-03-09 10:23:42 RT @schochastics: Recreation of Minard's March...

2018-03-07 15:15:38 TL;DR people got tired of using the word "Tidy...

2018-03-29 16:05:54 RT @RLangTip: Extracting data from PDF graphic...

2018-03-13 08:23:25 RT @tkb: Want to tell stories about global dev...

2018-03-03 17:42:59 Самый дешёвый CDN из всех что я знаю https://t...

2018-03-16 07:46:03 RT @json\_stack: python json.loads Unterminated...

2018-03-25 23:36:39 RT @KKulma: My last blog post on hints how to ...

2018-03-13 20:04:32 After seeing "📦s" I'm tempted to write a filte...

Name: text, dtype: object

# Load SentimentIntensityAnalyzer

from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Instantiate new SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

# Generate sentiment scores

sentiment\_scores = ds\_tweets['text'].apply(sid.polarity\_scores)

In [5]: sentiment\_scores

Out[5]:

created\_at

2018-03-30 13:04:22 {'neg': 0.0, 'pos': 0.15, 'compound': 0.5719, ...

2018-03-16 11:59:09 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-27 08:34:33 {'neg': 0.0, 'pos': 0.136, 'compound': 0.4588,...

2018-03-16 21:26:58 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-15 23:35:05 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-25 01:00:25 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-09 15:59:42 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-29 08:29:00 {'neg': 0.0, 'pos': 0.278, 'compound': 0.8074,...

2018-03-24 10:00:53 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-26 17:04:23 {'neg': 0.0, 'pos': 0.146, 'compound': 0.4404,...

2018-03-13 22:21:43 {'neg': 0.0, 'pos': 0.151, 'compound': 0.4926,...

2018-03-29 16:23:18 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-03 16:00:41 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-26 15:33:52 {'neg': 0.0, 'pos': 0.125, 'compound': 0.25, '...

2018-03-12 12:22:36 {'neg': 0.0, 'pos': 0.294, 'compound': 0.6114,...

2018-03-05 03:18:58 {'neg': 0.205, 'pos': 0.0, 'compound': -0.2023...

2018-03-14 18:22:49 {'neg': 0.0, 'pos': 0.188, 'compound': 0.5719,...

2018-03-06 00:46:06 {'neg': 0.157, 'pos': 0.0, 'compound': -0.4199...

2018-03-29 12:47:34 {'neg': 0.0, 'pos': 0.26, 'compound': 0.7713, ...

2018-03-20 12:59:06 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-20 05:52:40 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-04 08:15:14 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-20 23:42:59 {'neg': 0.0, 'pos': 0.186, 'compound': 0.6249,...

2018-03-19 16:30:14 {'neg': 0.0, 'pos': 0.091, 'compound': 0.0772,...

2018-03-13 22:30:44 {'neg': 0.163, 'pos': 0.096, 'compound': -0.22...

2018-03-20 17:45:09 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-21 15:32:36 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-20 12:59:59 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-02 07:31:54 {'neg': 0.0, 'pos': 0.201, 'compound': 0.6988,...

2018-03-09 16:02:59 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

...

2018-03-09 12:52:58 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-03 23:27:26 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-08 14:42:10 {'neg': 0.114, 'pos': 0.0, 'compound': -0.4019...

2018-03-25 08:17:09 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-08 10:46:29 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-12 17:49:10 {'neg': 0.0, 'pos': 0.265, 'compound': 0.7579,...

2018-03-06 18:51:27 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-10 14:54:02 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-06 22:04:43 {'neg': 0.0, 'pos': 0.234, 'compound': 0.7269,...

2018-03-09 02:19:46 {'neg': 0.0, 'pos': 0.17, 'compound': 0.5697, ...

2018-03-12 21:07:27 {'neg': 0.0, 'pos': 0.485, 'compound': 0.8588,...

2018-03-06 09:58:00 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-08 22:35:57 {'neg': 0.0, 'pos': 0.248, 'compound': 0.765, ...

2018-03-30 20:39:02 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-18 20:05:32 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-03 22:40:39 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-12 11:39:58 {'neg': 0.0, 'pos': 0.207, 'compound': 0.5719,...

2018-03-22 19:46:17 {'neg': 0.0, 'pos': 0.146, 'compound': 0.4404,...

2018-03-21 23:51:03 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-04 18:00:39 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-20 14:53:18 {'neg': 0.0, 'pos': 0.148, 'compound': 0.5367,...

2018-03-22 11:14:50 {'neg': 0.0, 'pos': 0.136, 'compound': 0.296, ...

2018-03-09 10:23:42 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-07 15:15:38 {'neg': 0.222, 'pos': 0.089, 'compound': -0.50...

2018-03-29 16:05:54 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-13 08:23:25 {'neg': 0.0, 'pos': 0.143, 'compound': 0.3612,...

2018-03-03 17:42:59 {'neg': 0.0, 'pos': 0.0, 'compound': 0.0, 'neu...

2018-03-16 07:46:03 {'neg': 0.184, 'pos': 0.0, 'compound': -0.4019...

2018-03-25 23:36:39 {'neg': 0.0, 'pos': 0.104, 'compound': 0.4199,...

2018-03-13 20:04:32 {'neg': 0.12, 'pos': 0.0, 'compound': -0.3241,...

Name: text, dtype: object

**Calculating sentiment scores**

A rough measure of sentiment towards a particular hashtag is to measure average sentiment for tweets mentioning a particular hashtag. It's also possible that other things are happening in that tweet, so it's important to inspect both text as well as metrics generated by automated text methods.

**Instructions**

**100 XP**

* Print out the first example of a positive tweet. Select the DataFrame with sentiment > 0.6 and use .values to get the full text of the tweet.
* Do the same for negative tweets with the index sentiment < -0.6.
* Generate average sentiment score per day for #python with the index check\_word\_in\_tweet. Use .resample() with argument '1 d'. Then take the mean.
* Do the same with #rstats.

# Print out the text of a positive tweet

print(ds\_tweets[sentiment > 0.6]['text'].values[0])

# Print out the text of a negative tweet

print(ds\_tweets[sentiment < -0.6]['text'].values[0])

# Generate average sentiment scores for #python

sentiment\_py = sentiment[check\_word\_in\_tweet('#python', ds\_tweets)].resample('1 d').mean()

# Generate average sentiment scores for #rstats

sentiment\_r = sentiment[check\_word\_in\_tweet('#rstats' , ds\_tweets)].resample('1 d').mean()

<script.py> output:

RT @CMastication: using #Python and #rstats in the same RMarkdown document is pretty awesome with the Reticulate Package. Objects from Pyth…

RT @DiffusePrioR: Here's the evolution of Irish Population density 1841-&gt;2002 on a DED level. You can see the devastating impact of the Fam…

**Plotting sentiment scores**

Lastly, let's plot the sentiment of each hashtag over time. This is largely similar to plotting the prevalence of tweets.

**Instructions**

**100 XP**

* Plot a line for #python usage with sentiment\_py. Use sentiment\_py.index.day as the x-axis.
* Do the same with sentiment\_r.

In [1]: sentiment\_py

Out[1]:

created\_at

2018-03-01 0.280120

2018-03-02 0.192238

2018-03-03 0.136630

2018-03-04 0.000000

2018-03-05 0.180233

2018-03-06 0.221950

2018-03-07 -0.021100

2018-03-08 0.269278

2018-03-09 0.163015

2018-03-10 0.092540

2018-03-11 0.221167

2018-03-12 0.175300

2018-03-13 -0.045260

2018-03-14 0.248231

2018-03-15 0.150400

2018-03-16 -0.080380

2018-03-17 0.052440

2018-03-18 0.000000

2018-03-19 0.095725

2018-03-20 0.182258

2018-03-21 0.215714

2018-03-22 0.201650

2018-03-23 0.205388

2018-03-24 0.175050

2018-03-25 -0.049033

2018-03-26 0.164200

2018-03-27 0.326650

2018-03-28 0.179750

2018-03-29 0.531350

2018-03-30 0.402675

2018-03-31 0.104575

Freq: D, Name: text, dtype: float64

# Import matplotlib

import matplotlib.pyplot as plt

# Plot average #python sentiment per day

plt.plot(sentiment\_py.index.day ,sentiment\_py , color = 'green')

# Plot average #rstats sentiment per day

plt.plot(sentiment\_r.index.day , sentiment\_r, color = 'blue')

plt.xlabel('Day')

plt.ylabel('Sentiment')

plt.title('Sentiment of data science languages')

plt.legend(('#python', '#rstats'))

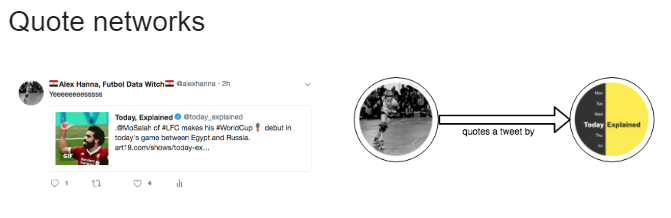
plt.show()

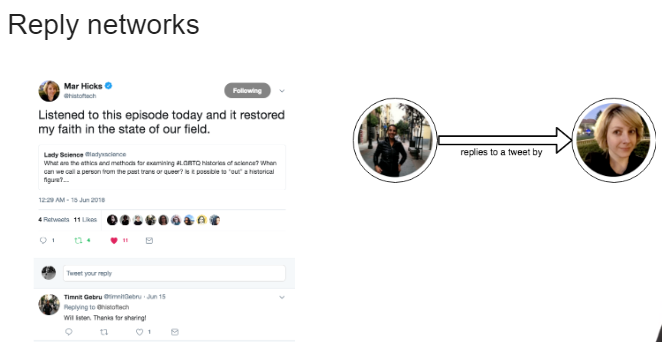
**CHAPTER 3**

#### **Twitter Networks**

Rwetweet networks:







**Types of Twitter networks**

Which one of these are **not** a type of Twitter network which can be constructed from the Twitter Streaming API?

**Answer the question**

**50 XP**

**Possible Answers**

* 

Reply network

press1

* 

Favorite network **(A)**

press2

* 

Retweet network

press3

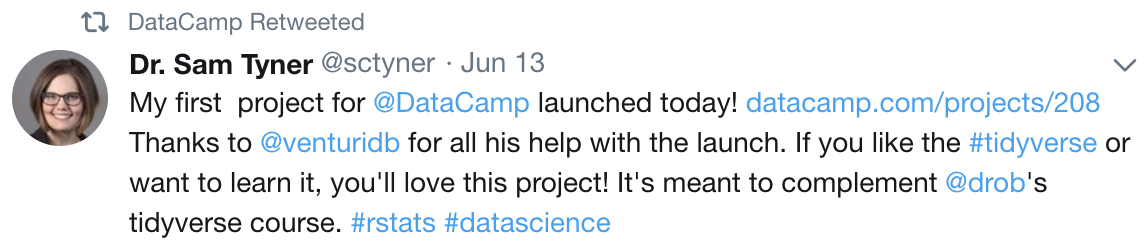
* 

Quoted tweet network

press4

Correct! With the Twitter Streaming API you can't obtain favorited data and therefore can't construct a network of favorites.

**Which direction is the arrow?**



Which direction does the edge go in this retweet? To be clear, @DataCamp retweets @sctyner.

**Answer the question**

**50 XP**

**Possible Answers**

* 

@DataCamp -> @sctyner **(A)**

press1

* 

@sctyner -> @DataCamp

press2

* 

This is not a directed network.

press3

Correct! The edge signifies that User1 retweets User2, so @DataCamp -> @sctyner.

**Creating retweet network**

Social media is, by nature, networked data. Twitter networks manifest in multiple ways. One of the most important types of networks that appear in Twitter are retweet networks. We can represent these as *directed* graphs, with the retweeting user as the source and the retweeted person as the target. With Twitter data in our flattened DataFrame, we can import these into networkx and create a retweet network.

For this exercise and the rest of this course we'll be using a dataset based on the **2018 State of the Union speech** given by Donald Trump. Those tweets have been loaded for you in sotu\_retweets.

**Instructions**

**100 XP**

* Import networkx as nx.
* Use the user's screen name as the source argument.
* Use the retweeted user's screen name as the target argument.
* Ensure that the network is a directed graph in the create\_using argument.

# Import networkx

import networkx as nx

# Create retweet network from edgelist

G\_rt = nx.from\_pandas\_edgelist(

sotu\_retweets,

source = 'user-screen\_name',

target = 'retweeted\_status-user-screen\_name',

create\_using = nx.DiGraph())

# Print the number of nodes

print('Nodes in RT network:', len(G\_rt.nodes()))

# Print the number of edges

print('Edges in RT network:', len(G\_rt.edges()))

<script.py> output:

Nodes in RT network: 2287

Edges in RT network: 2340

**Creating reply network**

Reply networks have a markedly different structure to retweet networks. While retweet networks often signal agreement, replies can signal discussion, deliberation, and disagreement. The network properties are the same, however: the network is directed, the source is the replier and the target is the user who is being replied to.

For this exercise we are going to create a reply network from a slightly different sample of State of the Union tweets. Those tweets have been loaded for you in sotu\_replies.

**Instructions**

**100 XP**

* Create the reply network from a pandas edge list.
* Use the user's screen name as the source argument.
* Use the screen name being replied to as the target argument.
* Ensure that the network is a directed graph in the create\_using argument.

# Import networkx

import networkx as nx

# Create reply network from edgelist

G\_reply = nx.from\_pandas\_edgelist(

sotu\_replies,

source = 'user-screen\_name',

target = 'in\_reply\_to\_screen\_name',

create\_using = nx.DiGraph())

# Print the number of nodes

print('Nodes in reply network:', len(G\_reply.nodes()))

# Print the number of edges

print('Edges in reply network:', len(G\_reply.edges()))

<script.py> output:

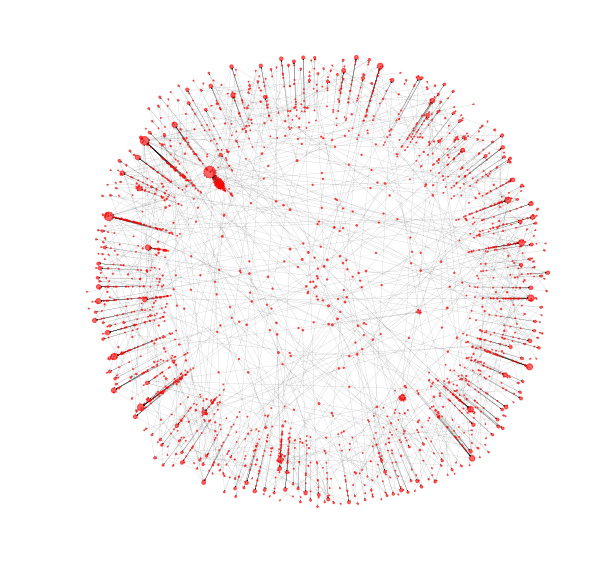
Nodes in reply network: 2622

Edges in reply network: 1904

**Visualizing retweet network**

Visualizing retweets networks is an important exploratory data analysis step because it allows us to visually inspect the structure of the network, understand if there is any user that has disproportionate influence, and if there are different spheres of conversation.

A retweet network visualized with a *force directed* algorithm may look something like this.



We are going to use a layout which runs quicker to see the plot, but the syntax is nearly the same.

networkx has been imported as nx, and the network has been loaded in G\_rt for you.

**Instructions**

**100 XP**

* Generate sizes with a list comprehension. Obtain the second item in x for all elements returned by the .degree() method.
* Pass the network name as the first argument to nx.draw\_networkx().
* Pass the layout positions as the second argument to draw\_networkx.
* Pass the sizes list to node\_size.

# Create random layout positions

pos = nx.random\_layout(G\_rt)

# Create size list

sizes = [x[1] for x in G\_rt.degree()]

# Draw the network

nx.draw\_networkx(G\_rt , pos ,

with\_labels = False,

node\_size = sizes,

width = 0.1, alpha = 0.7,

arrowsize = 2, linewidths = 0)

# Turn axis off and show

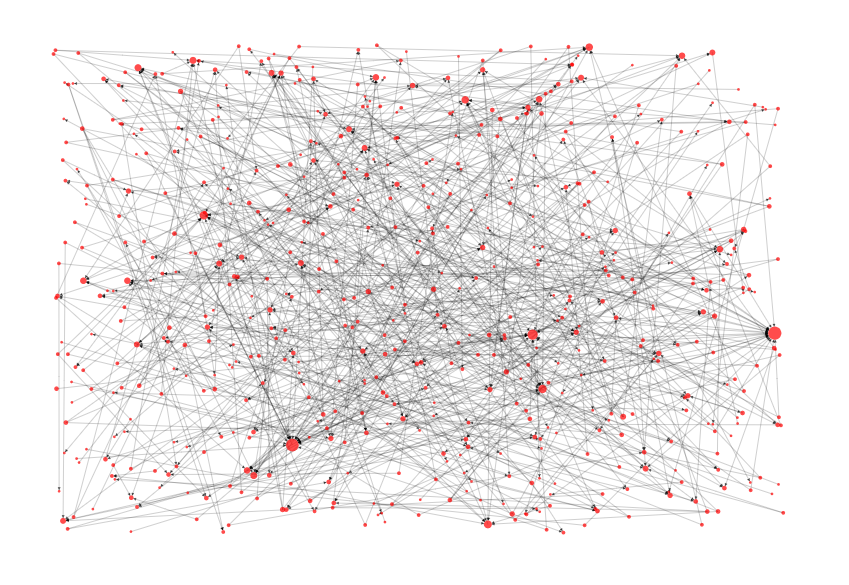
plt.axis('off'); plt.show()

   
The second item in x is x[1], and all degrees are returned by G\_rt.degree().

 The network is called G\_rt.

 The layout positions are stored in pos.

 The sizes are stored in sizes.



In [2]: pos

Out[2]:

{'1776tamara': array([0.48112833, 0.4668497 ], dtype=float32),

'2Run26': array([0.35506514, 0.95414394], dtype=float32),

'2old2kare': array([0.9930333, 0.2364624], dtype=float32),

'4AllSoulKind': array([0.65766746, 0.58490425], dtype=float32),

'4everNeverTrump': array([0.71934575, 0.3734127 ], dtype=float32),

'ALGOP': array([0.6120134 , 0.16206944], dtype=float32),

'APOYODECHILE': array([0.39756405, 0.47777805], dtype=float32),

'AStandUpGirl': array([0.7011222, 0.8217207], dtype=float32),

'AWhiteis': array([0.24244711, 0.67392796], dtype=float32),

'AdamBaldwin': array([0.37898585, 0.66838396], dtype=float32),

'Adamo55f': array([0.47068876, 0.8089639 ], dtype=float32),

'AddamLanza': array([0.65215385, 0.2329799 ], dtype=float32),

'AggieDave': array([0.52348757, 0.1090882 ], dtype=float32),

'AlanaKStewart': array([0.68018615, 0.07828025], dtype=float32),

'Alaskans4Trump': array([0.9538184 , 0.10287988], dtype=float32),

'AmandiOnAir': array([0.2324001 , 0.00136164], dtype=float32),

'AmerPatriot1': array([0.29371405, 0.63097614], dtype=float32),

'AmericanHotLips': array([0.6239529 , 0.11561839], dtype=float32),

'AmericanMex067': array([0.09086525, 0.8379785 ], dtype=float32),

'AmyMek': array([0.88955873, 0.4523405 ], dtype=float32),

'AngrierWHStaff': array([0.66116786, 0.04909713], dtype=float32),

'Arteesta1959': array([0.00353221, 0.48088905], dtype=float32),

'AveryChasten': array([0.45372763, 0.02164686], dtype=float32),

'B75434425': array([0.9183975 , 0.09129634], dtype=float32),

'BJCollins131': array([0.39863908, 0.18488604], dtype=float32),

'Bamadad1985': array([0.38319403, 0.6936554 ], dtype=float32),

'Baneff5': array([0.08120085, 0.8457982 ], dtype=float32),

'BasedMonitored': array([0.9219199, 0.5505297], dtype=float32),

'BestRDSP': array([0.46514797, 0.4349857 ], dtype=float32),

'BleedingRedDan': array([0.376677 , 0.10388424], dtype=float32),

'BloombergTV': array([0.50940174, 0.29690152], dtype=float32),

'Blueeyes2luv': array([0.2297636 , 0.19634645], dtype=float32),

'BobBoobka45': array([0.72704273, 0.01501615], dtype=float32),

'BorisEP': array([0.53865826, 0.14603573], dtype=float32),

'BraineaterBob': array([0.7956046 , 0.49197605], dtype=float32),

'BrandyJewellE': array([0.96750104, 0.11254009], dtype=float32),

'BrendanLonsway': array([0.8069687 , 0.39437005], dtype=float32),

'Butterfly5564': array([0.2924894, 0.5200102], dtype=float32),

'C9\_kas10': array([0.6470196, 0.2497462], dtype=float32),

'CBSNews': array([0.19100696, 0.9674943 ], dtype=float32),

'CBSThisMorning': array([0.66526145, 0.11139217], dtype=float32),

'CBreezypop': array([0.11739844, 0.30105335], dtype=float32),

'C\_Ethan\_Edsall': array([0.2082483, 0.4433677], dtype=float32),

'CaliConsrvative': array([0.8515481 , 0.12761222], dtype=float32),

'Call\_Me\_Dutch': array([0.42609566, 0.3056464 ], dtype=float32),

'CallanGray4': array([0.6533649, 0.9960863], dtype=float32),

'CaptnMikeD': array([0.02474115, 0.30875087], dtype=float32),

'CarolynSteger': array([0.8949783 , 0.04322289], dtype=float32),

'CaryFrazee': array([0.79229933, 0.5187166 ], dtype=float32),

'CassandraRules': array([0.5681032 , 0.22546336], dtype=float32),

'Cernovich': array([0.5399771, 0.9317029], dtype=float32),

'CesarCCardenas': array([0.6340848 , 0.26439747], dtype=float32),

'ChadHastyRadio': array([0.02661112, 0.9201492 ], dtype=float32),

'Chelseax05': array([0.7989227, 0.7988463], dtype=float32),

'ChrisVanHollen': array([0.90255654, 0.22215706], dtype=float32),

'ChristiChat': array([0.5946219 , 0.01694831], dtype=float32),

'CiAnDy3': array([0.31366894, 0.04733954], dtype=float32),

'ClaraHoton': array([0.4809319 , 0.39211753], dtype=float32),

'CollinRugg': array([0.1356733, 0.5636646], dtype=float32),

'ColumbiaBugle': array([0.86037344, 0.97702885], dtype=float32),

'CommUNITY\_USA': array([0.2207307 , 0.29397413], dtype=float32),

'CosmicRabbit42': array([0.10605526, 0.0020919 ], dtype=float32),

'Crimson3Pointer': array([0.44267255, 0.9984591 ], dtype=float32),

'CyndiRocks1': array([0.9432006, 0.2451306], dtype=float32),

'CynthiaMN5': array([0.32490966, 0.10021446], dtype=float32),

'DGComedy': array([0.01414381, 0.847647 ], dtype=float32),

'DLoesch': array([0.10118195, 0.516166 ], dtype=float32),

'DailyCaller': array([0.4760266, 0.8703705], dtype=float32),

'Darkmdd': array([0.30194685, 0.9805822 ], dtype=float32),

'DaveNYviii': array([0.5722038, 0.1048713], dtype=float32),

'DebMeinhardt': array([0.18292129, 0.8053267 ], dtype=float32),

'DebraBuenik': array([0.6397211 , 0.31516176], dtype=float32),

'Dee33305261': array([0.7792453, 0.522892 ], dtype=float32),

'Dee601': array([0.8209689, 0.4588211], dtype=float32),

'DeepSouthProud': array([0.8987127 , 0.62099135], dtype=float32),

'DemAwakening': array([0.66116875, 0.37836936], dtype=float32),

'Deniseflemings': array([0.01273756, 0.76326865], dtype=float32),

'Deplorable1st': array([0.13897383, 0.12648569], dtype=float32),

'DewsNewz': array([0.7701692, 0.440462 ], dtype=float32),

'DgThejudge54': array([0.4164366, 0.9181599], dtype=float32),

'DianaChic1': array([0.20613153, 0.16506644], dtype=float32),

'Discoveringme40': array([0.1330024, 0.5166163], dtype=float32),

'DoWhatsRight77': array([0.3821092, 0.0511263], dtype=float32),

'DomenicoNPR': array([0.08211067, 0.2867871 ], dtype=float32),

'DonaldJTrumpJr': array([0.53766346, 0.7168034 ], dtype=float32),

'DonaldUS2016': array([0.41550356, 0.74461544], dtype=float32),

'Dsandz15': array([0.33837712, 0.77413607], dtype=float32),

'DyingSpirit': array([0.565642 , 0.08490416], dtype=float32),

'EARTH3R': array([0.16486584, 0.3359916 ], dtype=float32),

'EF517\_V2': array([0.6578151, 0.7728778], dtype=float32),

'Earthjustice': array([0.7554744, 0.5448133], dtype=float32),

'EddieDonovan': array([0.15678327, 0.39126256], dtype=float32),

'EjHirschberger': array([0.6457984, 0.8589131], dtype=float32),

'EliseErosenb827': array([0.12392301, 0.5964869 ], dtype=float32),

'EllaBowenVirgo': array([0.9376684, 0.9080111], dtype=float32),

'Engineer\_David6': array([0.751644 , 0.07914896], dtype=float32),

'ErinBSullivan': array([0.3981379 , 0.72719103], dtype=float32),

'Eviesmamaw': array([0.42667234, 0.01574544], dtype=float32),

'FAIRImmigration': array([0.9807642, 0.6848297], dtype=float32),

'FMBRedemption': array([0.7086974 , 0.83924335], dtype=float32),

'FaithCompassion': array([0.07384551, 0.49559817], dtype=float32),

'FedupWithSwamp': array([0.36181727, 0.8633534 ], dtype=float32),

'ForAmerica': array([0.22549841, 0.8751245 ], dtype=float32),

'ForTrump': array([0.33258027, 0.95369714], dtype=float32),

'Forbespaula': array([0.36357632, 0.5399599 ], dtype=float32),

'FoxBusiness': array([0.20583779, 0.65004325], dtype=float32),

'FoxNews': array([0.3269682 , 0.17939018], dtype=float32),

'FrankLuntz': array([0.2212976 , 0.94843996], dtype=float32),

'FrankieRusso1': array([0.15307051, 0.6955295 ], dtype=float32),

'FreebyrdWil': array([0.21803544, 0.8457525 ], dtype=float32),

'FriedasMom7': array([0.7244553, 0.6110235], dtype=float32),

'Fuctupmind': array([0.33867085, 0.5523701 ], dtype=float32),

'FullFrontalSamB': array([0.20926443, 0.07080518], dtype=float32),

'G1eor2ges3': array([0.827655 , 0.35170335], dtype=float32),

'GOP': array([0.04090779, 0.51611084], dtype=float32),

'GaryLeeThrasher': array([0.99173194, 0.5604223 ], dtype=float32),

'GayRepublicSwag': array([0.71560127, 0.41051978], dtype=float32),

'Gays4ChickFilA': array([0.87545687, 0.51042235], dtype=float32),

'GordonCrash': array([0.44735938, 0.8732953 ], dtype=float32),

'GottaLaff': array([0.7224434 , 0.32295892], dtype=float32),

'GramaD386': array([0.5066322, 0.7105607], dtype=float32),

'GrantJKidney': array([0.93193215, 0.58217543], dtype=float32),

'GretchenRenae': array([0.4170222 , 0.68130076], dtype=float32),

'Guiller92839869': array([0.26812443, 0.19449784], dtype=float32),

'HighRawMama': array([0.5826711, 0.8148437], dtype=float32),

'Hoosiers1986': array([0.30061013, 0.63444227], dtype=float32),

'HouseAppropsGOP': array([0.08407621, 0.1092332 ], dtype=float32),

'HouseGOP': array([0.6694412 , 0.29429385], dtype=float32),

'HowardKurtz': array([0.16499801, 0.18532191], dtype=float32),

'HuffPost': array([0.94711953, 0.61766 ], dtype=float32),

'HuffPostPol': array([0.03219793, 0.74478066], dtype=float32),

'InciteTruth': array([0.49993595, 0.03604485], dtype=float32),

'IndivChi\_South': array([0.45334613, 0.40465134], dtype=float32),

'IndivisibleIL': array([0.7096232, 0.9593452], dtype=float32),

'InfobaeAmerica': array([0.9021715 , 0.88069475], dtype=float32),

'Innrdreams': array([0.02754293, 0.03191799], dtype=float32),

'IronStache': array([0.5931769 , 0.69170177], dtype=float32),

'Italians4Trump': array([0.33633953, 0.3270999 ], dtype=float32),

'Iza77789584': array([0.6249035 , 0.67468905], dtype=float32),

'JGonza28': array([0.43086275, 0.4936851 ], dtype=float32),

'Jali\_Cat': array([0.92238235, 0.30338073], dtype=float32),

'JaneOrLean': array([0.5020711 , 0.06744864], dtype=float32),

'Jans\_Arbor': array([0.3742922 , 0.21401191], dtype=float32),

'JasonMillerinDC': array([0.7545229 , 0.35166937], dtype=float32),

'JeffWedge': array([0.45063648, 0.5477636 ], dtype=float32),

'Jillibean557': array([0.66558844, 0.04281892], dtype=float32),

'Joeyswife': array([0.6279218 , 0.03833161], dtype=float32),

'JonnyBoGermany': array([0.18769394, 0.5733189 ], dtype=float32),

'JordanSekulow': array([0.18249173, 0.17545176], dtype=float32),

'JoyAnnReid': array([0.56362796, 0.88667816], dtype=float32),

'Jrzladyvjs': array([0.67964774, 0.05498161], dtype=float32),

'JudgeJeanine': array([0.23659982, 0.9168323 ], dtype=float32),

'JustinTHaskins': array([0.90987164, 0.1286312 ], dtype=float32),

'K1erry': array([0.19915816, 0.9665794 ], dtype=float32),

'KamVTV': array([0.44132414, 0.93284255], dtype=float32),

'KassyDillon': array([0.5874938, 0.9482524], dtype=float32),

'KatWahamaa': array([0.76857424, 0.85288227], dtype=float32),

'KatiePavlich': array([0.84407747, 0.07620406], dtype=float32),

'KazeSkyz': array([0.6160065 , 0.05893948], dtype=float32),

'Keque\_Mage': array([0.62724894, 0.7234163 ], dtype=float32),

'KerenizKeren': array([0.09832809, 0.5944671 ], dtype=float32),

'KimStrassel': array([0.08189444, 0.49503964], dtype=float32),

'Kita71': array([0.18162884, 0.3213189 ], dtype=float32),

'KostJason': array([0.27999994, 0.65766025], dtype=float32),

'KrissyMAGA3X': array([0.84234244, 0.08319499], dtype=float32),

'KyleKulinski': array([0.3186676 , 0.04717327], dtype=float32),

'LKMwgm': array([0.2536353 , 0.99656916], dtype=float32),

'LOV\_MotherEarth': array([0.2013995, 0.8116444], dtype=float32),

'Lainstep': array([0.41156924, 0.48102626], dtype=float32),

'LateNightSeth': array([0.24632101, 0.38527146], dtype=float32),

'Lauradeanneh': array([0.89642566, 0.22327048], dtype=float32),

'LeahR77': array([0.5785515, 0.5215331], dtype=float32),

'LessGovMoreFun': array([0.50118977, 0.20933399], dtype=float32),

'LindaGarriss': array([0.61985105, 0.32927912], dtype=float32),

'LindaSuhler': array([0.47024208, 0.7137683 ], dtype=float32),

'LisaMarieBoothe': array([0.6626364, 0.5578699], dtype=float32),

'LizCrokin': array([0.55095327, 0.81331366], dtype=float32),

'LoConservative': array([0.26968327, 0.44006166], dtype=float32),

'LobbyistandPR': array([0.01940709, 0.00772982], dtype=float32),

'LoneStar\_Dad': array([0.85244954, 0.956312 ], dtype=float32),

'LukeMAGA1': array([0.8948778 , 0.75320435], dtype=float32),

'Lynne\_Shine': array([0.10590848, 0.13089494], dtype=float32),

'MAAyala53': array([0.70495886, 0.99535847], dtype=float32),

'MABardino': array([0.60752904, 0.8119533 ], dtype=float32),

'MAGARoseTaylor': array([0.03009363, 0.33909923], dtype=float32),

'MAGATRUMPYEAR2': array([0.22780207, 0.58941543], dtype=float32),

'MIGOP': array([0.7689587, 0.6187618], dtype=float32),

'Maggieb1B': array([0.01116088, 0.3901253 ], dtype=float32),

'MaggioMichelle': array([0.585902 , 0.83538866], dtype=float32),

'MarilynHanbyDur': array([0.89775276, 0.9470675 ], dtype=float32),

'Marjori77680062': array([0.75606 , 0.5370798], dtype=float32),

'MarkDice': array([0.2985792, 0.941807 ], dtype=float32),

'MarkYoungTruth': array([0.5013243 , 0.59712535], dtype=float32),

'MarkeeceYBR': array([0.5182427, 0.6232691], dtype=float32),

'MattOswaltVA': array([0.78555673, 0.08909597], dtype=float32),

'MeetThePress': array([0.481115 , 0.44921246], dtype=float32),

'MichaelDelauzon': array([0.5192804, 0.6943149], dtype=float32),

'MichiganDems': array([0.17108183, 0.82911265], dtype=float32),

'MikeLevinCA': array([0.80402637, 0.85765177], dtype=float32),

'MinnPatriot': array([0.6587676 , 0.19600947], dtype=float32),

'Mo\_An2016': array([0.27868748, 0.06998219], dtype=float32),

'MonaSalama\_': array([0.06862339, 0.9297699 ], dtype=float32),

'MonicaCrowley': array([0.8822761 , 0.82230383], dtype=float32),

'MoveOn': array([0.64950097, 0.8713263 ], dtype=float32),

'MyPlace4U': array([0.777769 , 0.20040132], dtype=float32),

'MyraTye': array([0.24168564, 0.09552964], dtype=float32),

'NBCNews': array([0.31153616, 0.941467 ], dtype=float32),

'NBCPolitics': array([0.48303425, 0.98555976], dtype=float32),

'NIVIsa4031': array([0.10263526, 0.69983405], dtype=float32),

'NWOinPanicMode': array([0.750717 , 0.57406384], dtype=float32),

'NewsNumb': array([0.1633377, 0.9841283], dtype=float32),

'Nig0u': array([0.09210494, 0.4337012 ], dtype=float32),

'NorahODonnell': array([0.37239647, 0.85715306], dtype=float32),

'Nvehecnycrrcom1': array([0.39804426, 0.7379954 ], dtype=float32),

'ODL\_5sosjrt1d': array([0.93289167, 0.31435135], dtype=float32),

'OldSargeUSAF': array([0.23976947, 0.01519766], dtype=float32),

'PPact': array([0.16593789, 0.78099793], dtype=float32),

'PamelaB98884515': array([0.4539898 , 0.36552063], dtype=float32),

'Patrici81767862': array([0.96300447, 0.3418306 ], dtype=float32),

'PatriotBrigidFL': array([0.5721468, 0.6609518], dtype=float32),

'PattyPmb37': array([0.3219806 , 0.66156435], dtype=float32),

'PaulLee85': array([0.45308894, 0.93235064], dtype=float32),

'PennsylvaniaRed': array([0.08740775, 0.34679472], dtype=float32),

'PeteS77252077': array([0.5103764 , 0.75694585], dtype=float32),

'PinkBelgium': array([0.6006986 , 0.86586446], dtype=float32),

'Pink\_About\_it': array([0.7446199 , 0.04423721], dtype=float32),

'PolitiBunny': array([0.35227695, 0.8020757 ], dtype=float32),

'PoliticalShort': array([0.21065263, 0.42120007], dtype=float32),

'PoliticallyRYT': array([0.627174 , 0.7064798], dtype=float32),

'PolitixGal': array([0.42351806, 0.37199178], dtype=float32),

'Population\_Grow': array([0.00416849, 0.2948947 ], dtype=float32),

'Pray4Liberty\_12': array([0.03503317, 0.14360097], dtype=float32),

'PressSec': array([0.26361033, 0.27147985], dtype=float32),

'PrisonPlanet': array([0.72216326, 0.93208873], dtype=float32),

'ProChoiceKills': array([0.38574347, 0.24862377], dtype=float32),

'PublicHealth': array([0.45718172, 0.22294624], dtype=float32),

'Public\_Citizen': array([0.91213214, 0.5807132 ], dtype=float32),

'QAnonPatriot': array([0.14526373, 0.0921861 ], dtype=float32),

'REALtrumpbureau': array([0.4263513 , 0.89338917], dtype=float32),

'RI4A': array([0.20705006, 0.65555155], dtype=float32),

'RachelStoltz': array([0.23824991, 0.8077911 ], dtype=float32),

'RadioFreeUnite': array([0.2408559, 0.343456 ], dtype=float32),

'RandyWiksten': array([0.99180466, 0.86853385], dtype=float32),

'RealAlexJones': array([0.66487247, 0.8878568 ], dtype=float32),

'RealDrGina': array([0.33981085, 0.5950739 ], dtype=float32),

'RealJTP': array([0.9251325, 0.84167 ], dtype=float32),

'RealJack': array([0.81051344, 0.13111511], dtype=float32),

'RealKyleMorris': array([0.38367274, 0.06073962], dtype=float32),

'RealMattCouch': array([0.11574454, 0.9526027 ], dtype=float32),

'RealSaavedra': array([0.50324744, 0.34889293], dtype=float32),

'RealStone0016': array([0.9452366 , 0.29008642], dtype=float32),

'RedNationRising': array([0.01484058, 0.16322003], dtype=float32),

'RedWaveRising': array([0.35595766, 0.9297637 ], dtype=float32),

'ReneeCarrollAZ': array([0.5577987, 0.7448045], dtype=float32),

'RepBarbaraLee': array([0.41729107, 0.98903453], dtype=float32),

'RepDavidEPrice': array([0.66807276, 0.17261174], dtype=float32),

'RepStevenSmith': array([0.1575336, 0.1190697], dtype=float32),

'RepublicanChick': array([0.845533 , 0.18690374], dtype=float32),

'RevChuckCurrie': array([0.5544121, 0.9756713], dtype=float32),

'RicoColon1': array([0.42254335, 0.24503304], dtype=float32),

'RightWingAngel': array([0.37977287, 0.19305472], dtype=float32),

'Rmhenry1Henry': array([0.67393595, 0.97009856], dtype=float32),

'Roberts5Don': array([0.30725464, 0.7511212 ], dtype=float32),

'Rockprincess818': array([0.7418622 , 0.04857903], dtype=float32),

'RogerWicker': array([0.36178866, 0.22826323], dtype=float32),

'RoniSeale': array([0.9543783 , 0.08680182], dtype=float32),

'Rvogt1\_cba': array([0.88271296, 0.91947246], dtype=float32),

'RyanAFournier': array([0.32833537, 0.36805326], dtype=float32),

'SAMMIESMILES8': array([0.32297394, 0.97209823], dtype=float32),

'SCovitz': array([0.6680013, 0.8587266], dtype=float32),

'SOLDHTRENDY8': array([0.43857226, 0.0596779 ], dtype=float32),

'STEPHMHAMILL': array([0.31876642, 0.6919703 ], dtype=float32),

'STrimel': array([0.927455 , 0.19836569], dtype=float32),

'SaraCarterDC': array([0.09571252, 0.8853268 ], dtype=float32),

'ScottPresler': array([0.9873511 , 0.40866014], dtype=float32),

'Scottforpa': array([0.56457037, 0.19133572], dtype=float32),

'Seattle1227': array([0.55371374, 0.75600547], dtype=float32),

'SecretService': array([0.6458225 , 0.03869959], dtype=float32),

'SecretaryRoss': array([0.71946895, 0.42310646], dtype=float32),

'SecretarySonny': array([0.08940295, 0.9165543 ], dtype=float32),

'SecureAmerica': array([6.9805866e-01, 1.2010149e-04], dtype=float32),

'SenateGOP': array([0.2780236 , 0.74176043], dtype=float32),

'Sewapalooza': array([0.95989954, 0.49291372], dtype=float32),

'Shakestweetz': array([0.06369086, 0.48514938], dtype=float32),

'Shareblue': array([0.40453854, 0.82653016], dtype=float32),

'Shelley2021': array([0.2499543 , 0.35053933], dtype=float32),

'SiddonsDan': array([0.9019114, 0.9836309], dtype=float32),

'SierraClub': array([0.47841585, 0.23329356], dtype=float32),

'Slate': array([0.05209113, 0.4067789 ], dtype=float32),

'SoCal4Trump': array([0.6665271 , 0.19203015], dtype=float32),

'SonofLiberty357': array([0.42866346, 0.73276746], dtype=float32),

'SpeakerRyan': array([0.58754015, 0.584139 ], dtype=float32),

'StaceyJeffries9': array([0.28123477, 0.36227676], dtype=float32),

'SteveJa39296543': array([0.3513573, 0.0651576], dtype=float32),

'StockMonsterVIP': array([0.5974331 , 0.29315248], dtype=float32),

'Stump4TrumpPAC': array([0.52049375, 0.23929325], dtype=float32),

'SueSuebarker12': array([0.08983187, 0.6484497 ], dtype=float32),

'Surabees': array([0.09715961, 0.20678315], dtype=float32),

'Susan40370948': array([0.4562706 , 0.27980202], dtype=float32),

'Susan\_Citizen': array([0.3186339 , 0.48905918], dtype=float32),

'THETXEMBASSY': array([0.46798757, 0.8079382 ], dtype=float32),

'TangledTight': array([0.14772284, 0.28421924], dtype=float32),

'TeamCavuto': array([0.4754678, 0.9674366], dtype=float32),

'TeenVogue': array([0.56221837, 0.12224355], dtype=float32),

'TeresanMS': array([0.61718607, 0.4047395 ], dtype=float32),

'TheIvanVolter': array([0.3692912, 0.2422198], dtype=float32),

'TheMominatrixx': array([0.95025164, 0.81596607], dtype=float32),

'TheMorningSpew': array([0.6964692 , 0.28613934], dtype=float32),

'ThePureGal': array([0.38987422, 0.7547971 ], dtype=float32),

'TheRickyDavila': array([0.10652894, 0.07156011], dtype=float32),

'The\_Rumpus': array([0.36302817, 0.49382883], dtype=float32),

'The\_Trump\_Train': array([0.82852733, 0.36726555], dtype=float32),

'ThomasMHern': array([0.2982454 , 0.41862684], dtype=float32),

'TikiMaximus': array([0.5429888, 0.6939847], dtype=float32),

'TimRunsHisMouth': array([0.22060332, 0.32265514], dtype=float32),

'TinaDupuy': array([0.43087566, 0.6249645 ], dtype=float32),

'Tombx7M': array([0.24751316, 0.5248662 ], dtype=float32),

'TomiLahren': array([0.01315983, 0.02414841], dtype=float32),

'Trey\_VonDinkis': array([0.27384222, 0.20611513], dtype=float32),

'TrumpSuperPAC': array([0.617764 , 0.39770743], dtype=float32),

'TrumpTheHill': array([0.2845492, 0.9658863], dtype=float32),

'Trumperland': array([0.92758423, 0.56900376], dtype=float32),

'Trumpfan1995': array([0.7602103 , 0.23008996], dtype=float32),

'TrumpingNow': array([0.97104615, 0.87168294], dtype=float32),

'UNITEDWEDREAM': array([0.14906201, 0.5308294 ], dtype=float32),

'USATODAY': array([0.22685145, 0.5513148 ], dtype=float32),

'USATrump45': array([0.7181593, 0.9771652], dtype=float32),

'UTHornsRawk': array([0.27149183, 0.48421976], dtype=float32),

'Uncle\_Jimbo': array([0.2964633 , 0.45638236], dtype=float32),

'V\_robigg': array([0.16555998, 0.6126823 ], dtype=float32),

'Veteran4Trump': array([0.11438483, 0.05258034], dtype=float32),

'Veteran\_4\_Trump': array([0.18380737, 0.08158329], dtype=float32),

'Vielle\_Beignet': array([0.23268639, 0.7466976 ], dtype=float32),

'VoCommunism': array([0.7670693, 0.5581781], dtype=float32),

'VoteIndpndt': array([0.5876157 , 0.96736187], dtype=float32),

'WLTX': array([0.24465978, 0.3385822 ], dtype=float32),

'WashTimes': array([0.49155182, 0.9160914 ], dtype=float32),

'WendyVerran': array([0.8205742 , 0.46493486], dtype=float32),

'WhiteHatGuy': array([0.04503958, 0.6726985 ], dtype=float32),

'WhiteHouse': array([0.6559231, 0.4056532], dtype=float32),

'WhosGoneGalt': array([0.46805966, 0.6259065 ], dtype=float32),

'WidowFike': array([0.70087135, 0.4583325 ], dtype=float32),

'WilliamAmos': array([0.6736261, 0.8993135], dtype=float32),

'WomenforTrump': array([0.44935188, 0.03991305], dtype=float32),

'YoungDems4Trump': array([0.2387834 , 0.70477855], dtype=float32),

'ZN\_007': array([0.3305821, 0.9203304], dtype=float32),

'adams5861': array([0.79265136, 0.24296218], dtype=float32),

'adpackman': array([0.21868214, 0.45551965], dtype=float32),

'aedaw9i': array([0.90502846, 0.4435963 ], dtype=float32),

'alabamamangreg': array([0.05190095, 0.29430693], dtype=float32),

'ali': array([0.6769059 , 0.21550545], dtype=float32),

'alicenmary': array([0.7600458 , 0.52694064], dtype=float32),

'alivitali': array([0.52571154, 0.44624835], dtype=float32),

'allaboutme4545': array([0.49857163, 0.57371294], dtype=float32),

'amprog': array([0.5597379 , 0.33483642], dtype=float32),

'amras': array([0.85938907, 0.8215041 ], dtype=float32),

'annespo': array([0.90534157, 0.20763586], dtype=float32),

'anyalovesgod': array([0.9364719 , 0.22357963], dtype=float32),

'atlaswon': array([0.69794226, 0.8053969 ], dtype=float32),

'bacon\_texas': array([0.457412, 0.753526], dtype=float32),

'bannerite': array([0.07119437, 0.15040252], dtype=float32),

'basskicker11': array([0.01740016, 0.799489 ], dtype=float32),

'bbschrei': array([0.6963113 , 0.44032788], dtype=float32),

'bcuzimdamomma': array([0.2128315 , 0.39230406], dtype=float32),

'beinpulse': array([0.9989184 , 0.04061612], dtype=float32),

'bellabellariley': array([0.8757712 , 0.52071834], dtype=float32),

'benshapiro': array([0.7473259 , 0.20959195], dtype=float32),

'bibeydoll11': array([0.68837434, 0.20430411], dtype=float32),

'bighead\_buddy': array([0.7399227, 0.7382937], dtype=float32),

'bk2sq1': array([0.25177708, 0.52388066], dtype=float32),

'bkindivisible': array([0.6029322 , 0.36418745], dtype=float32),

'bleedingonpaper': array([0.00594284, 0.36571914], dtype=float32),

'bomiksu': array([0.58452487, 0.06660292], dtype=float32),

'brandongroeny': array([0.4974709 , 0.34590432], dtype=float32),

'bryanbehar': array([0.46680987, 0.26328105], dtype=float32),

'c5hardtop1999': array([0.90971464, 0.04341809], dtype=float32),

'caitie1877': array([0.46365273, 0.50221634], dtype=float32),

'caligurll17': array([0.12695803, 0.77716243], dtype=float32),

'camerabug1975': array([0.26798216, 0.67540896], dtype=float32),

'captnavarre': array([0.6474325 , 0.03739211], dtype=float32),

'carrieksada': array([0.08000652, 0.11612538], dtype=float32),

'cbarnella': array([0.3242216, 0.7543916], dtype=float32),

'ceceoo': array([0.8012768 , 0.87545985], dtype=float32),

'cgpb': array([0.03911935, 0.58388394], dtype=float32),

'charliekirk11': array([0.20609573, 0.7177576 ], dtype=float32),

'charlottesnead': array([0.9361801 , 0.23563835], dtype=float32),

'chicksonright': array([0.08334827, 0.196309 ], dtype=float32),

'chinovvv': array([0.05395579, 0.04824937], dtype=float32),

'ciarokee': array([0.8993916, 0.5136793], dtype=float32),

'cleanh2oaction': array([0.99123967, 0.22087726], dtype=float32),

'cliffbaker9': array([8.1887614e-05, 9.8059732e-01], dtype=float32),

'cmec3180': array([0.01975609, 0.36556727], dtype=float32),

'cnni': array([0.7101616 , 0.95850974], dtype=float32),

'colaresisle': array([0.5194851 , 0.61289454], dtype=float32),

'coliciamingo': array([0.04068319, 0.33275363], dtype=float32),

'consmover': array([0.01612921, 0.5944319 ], dtype=float32),

'corinnec': array([0.6909154 , 0.34714606], dtype=float32),

'couerfidele': array([0.4699548, 0.8610075], dtype=float32),

'craymer\_theresa': array([0.39342034, 0.81232625], dtype=float32),

'cristinalaila1': array([0.16748165, 0.5340765 ], dtype=float32),

'dailyhillster': array([0.90317774, 0.36903888], dtype=float32),

'dango1103': array([0.6864636 , 0.10043585], dtype=float32),

'datrumpnation1': array([0.2680211, 0.990239 ], dtype=float32),

'david1952\_david': array([0.581123 , 0.36434168], dtype=float32),

'davidpsdem': array([0.08178008, 0.13841558], dtype=float32),

'debhernandez': array([0.8599411 , 0.10352092], dtype=float32),

'debra\_bonanno': array([0.9153545 , 0.75451833], dtype=float32),

'dfalls1552': array([0.680903, 0.904226], dtype=float32),

'dhookstead': array([0.11415849, 0.38553154], dtype=float32),

'dixiefortrump': array([0.8705456, 0.6943861], dtype=float32),

'dknott409': array([0.6547527 , 0.10174605], dtype=float32),

'dlortner': array([0.03941017, 0.8278215 ], dtype=float32),

'ebonyangel529': array([0.39937872, 0.42430687], dtype=float32),

'ejporg': array([0.3240644 , 0.57427835], dtype=float32),

'emilykholcomb': array([0.6116714, 0.8475023], dtype=float32),

'emmaladyrose': array([0.6563432, 0.6473851], dtype=float32),

'ems56rn': array([0.12062866, 0.8263408 ], dtype=float32),

'esaagar': array([0.8544525, 0.3848378], dtype=float32),

'eyesonolive': array([0.9798863, 0.6684331], dtype=float32),

'fastnana69': array([0.25734812, 0.08265267], dtype=float32),

'fidget5000': array([0.79236686, 0.9578448 ], dtype=float32),

'file1013': array([0.9865799 , 0.37812093], dtype=float32),

'flachas3': array([0.95248884, 0.49865767], dtype=float32),

'fouser\_renee': array([0.51312816, 0.66662455], dtype=float32),

'gaberobertson1': array([0.41641462, 0.4322392 ], dtype=float32),

'gailtalk': array([0.9599391, 0.8767047], dtype=float32),

'garza\_armandina': array([0.0435687, 0.6840411], dtype=float32),

'gkilti': array([0.6633928, 0.5494131], dtype=float32),

'glennbeck': array([0.6693138 , 0.58593655], dtype=float32),

'gotapointtomake': array([0.6320505, 0.0261966], dtype=float32),

'guypbenson': array([0.274225 , 0.11697052], dtype=float32),

'hamnitup': array([0.3047681 , 0.39818567], dtype=float32),

'hardball': array([0.3430861 , 0.63735676], dtype=float32),

'harden\_beverly': array([0.43296933, 0.88400304], dtype=float32),

'heresyoursighn': array([0.67758584, 0.3072528 ], dtype=float32),

'hogangidley45': array([0.25754207, 0.56435907], dtype=float32),

'hriley37': array([0.31728548, 0.4148262 ], dtype=float32),

'ilissamcclellan': array([0.7093857, 0.9245519], dtype=float32),

'iloveutothemoon': array([0.343178 , 0.7290497], dtype=float32),

'iowa\_trump': array([0.69105655, 0.91877925], dtype=float32),

'irritatedwoman': array([0.42981333, 0.8728789 ], dtype=float32),

'islandmafia138': array([0.52029043, 0.49868822], dtype=float32),

'itiolaakingbola': array([0.5187726 , 0.76465756], dtype=float32),

'jayoung1892': array([0.14877765, 0.940029 ], dtype=float32),

'jemmtweetmaga17': array([0.94477785, 0.62132835], dtype=float32),

'jennbaum1': array([0.33706638, 0.9275766 ], dtype=float32),

'jennybethm': array([0.89386535, 0.49650797], dtype=float32),

'jetrotter': array([0.5694477 , 0.40493056], dtype=float32),

'jhjulian1956': array([0.081199 , 0.7234656], dtype=float32),

'jjwills2': array([0.9835216 , 0.07936579], dtype=float32),

'joegooding': array([0.7331279, 0.6052268], dtype=float32),

'joelpollak': array([0.82181364, 0.35623983], dtype=float32),

'joey\_girardi': array([0.95201075, 0.5274567 ], dtype=float32),

'john\_abrahamson': array([0.6030601, 0.545068 ], dtype=float32),

'johncardillo': array([0.26509628, 0.12738848], dtype=float32),

'jordanbaglivi': array([0.76368284, 0.24366638], dtype=float32),

'jtblogs': array([0.92759734, 0.59808356], dtype=float32),

'jtoufas': array([0.00742638, 0.5515927 ], dtype=float32),

'judy\_piersol': array([0.7878174 , 0.03248247], dtype=float32),

'junktion3': array([0.9957818 , 0.21983595], dtype=float32),

'justicedems': array([0.76466423, 0.81031483], dtype=float32),

'jvera15': array([0.3167879 , 0.35426468], dtype=float32),

'jwbranda': array([0.62520856, 0.4416974 ], dtype=float32),

'kashanacauley': array([0.28889006, 0.6399919 ], dtype=float32),

'kaxelivar': array([0.00268806, 0.98834544], dtype=float32),

'kbari12': array([0.26432797, 0.9436147 ], dtype=float32),

'kelliwardaz': array([0.751615 , 0.47399187], dtype=float32),

'kima\_jones': array([0.16839236, 0.13561895], dtype=float32),

'kittylLaura': array([0.09703816, 0.46190876], dtype=float32),

'krassenstein': array([0.01639248, 0.7211844 ], dtype=float32),

'kristina\_wong': array([0.00554541, 0.48490945], dtype=float32),

'krystalwolfgrl': array([0.02057616, 0.918097 ], dtype=float32),

'kwilli1046': array([0.84650624, 0.55325735], dtype=float32),

'kxly4news': array([0.13949195, 0.9395414 ], dtype=float32),

'laamigable21': array([0.4102657, 0.6617165], dtype=float32),

'labuda\_robert': array([0.2865366 , 0.30646974], dtype=float32),

'larryelder': array([0.9747909, 0.8673763], dtype=float32),

'ldobbs1234': array([0.88759345, 0.01611863], dtype=float32),

'ldubindubc': array([0.4091723, 0.4204814], dtype=float32),

'les\_deplorable': array([0.72013456, 0.7147715 ], dtype=float32),

'lightning\_grl': array([0.03166893, 0.15172996], dtype=float32),

'lisarob61254794': array([0.8038433, 0.3823702], dtype=float32),

'liviapolise': array([0.10544587, 0.23247978], dtype=float32),

'ljblack6': array([0.3362299 , 0.99586105], dtype=float32),

'lovinurvibe': array([0.9088418 , 0.16200083], dtype=float32),

'mac9153': array([0.42353255, 0.8727353 ], dtype=float32),

'mariann\_benway': array([0.3871952 , 0.55585957], dtype=float32),

'mariestroughter': array([0.39109582, 0.31050873], dtype=float32),

'mat\_johnson': array([0.73812985, 0.2506108 ], dtype=float32),

'maziehirono': array([0.86448026, 0.27690178], dtype=float32),

'mbsally': array([0.02524236, 0.2669058 ], dtype=float32),

'mccroreychris': array([0.34951854, 0.27742395], dtype=float32),

'mcjagreb': array([0.8806071, 0.3913165], dtype=float32),

'mehdirhasan': array([0.68995255, 0.43430907], dtype=float32),

'melperezdad': array([0.20579751, 0.29065567], dtype=float32),

'mgrier1': array([0.2410406 , 0.50843716], dtype=float32),

'michael\_\_1776': array([0.8086261 , 0.16477937], dtype=float32),

'michaeldweiss': array([0.7693973, 0.5737741], dtype=float32),

'michaelfromolg': array([0.5543833, 0.3889506], dtype=float32),

'michaeljknowles': array([0.3262356, 0.4832169], dtype=float32),

'michelleking66': array([0.7745802, 0.1346135], dtype=float32),

'michellemalkin': array([0.87914246, 0.06393855], dtype=float32),

'misdmeanormom': array([0.55678517, 0.15895964], dtype=float32),

'mjgranger1': array([0.54647905, 0.8619121 ], dtype=float32),

'mms5048': array([0.5395048 , 0.62630934], dtype=float32),

'mng2mala': array([0.29451698, 0.971364 ], dtype=float32),

'mrctv': array([0.35739756, 0.04359146], dtype=float32),

'mskaahn': array([0.61217934, 0.98821497], dtype=float32),

'muellerew': array([0.44187927, 0.31843477], dtype=float32),

'nameredacted5': array([0.68482584, 0.48611167], dtype=float32),

'nancyleeholley': array([0.5426359 , 0.06677444], dtype=float32),

'nancypr19': array([0.70711505, 0.48388904], dtype=float32),

'nicole\_insley': array([0.4258303 , 0.31226122], dtype=float32),

'nljac000': array([0.7797667 , 0.23747823], dtype=float32),

'nonnieh65': array([0.7173541, 0.7157504], dtype=float32),

'noraconnor89': array([0.55603474, 0.5005614 ], dtype=float32),

'nprpolitics': array([0.86630917, 0.25045538], dtype=float32),

'oJibboB80': array([0.34879732, 0.6346381 ], dtype=float32),

'ohioMom10': array([0.33554387, 0.34956622], dtype=float32),

'oldgeekgal': array([0.65075034, 0.86545986], dtype=float32),

'organic2016': array([0.03395364, 0.98262256], dtype=float32),

'osescalante': array([0.6499713 , 0.34369767], dtype=float32),

'paphotog': array([0.24624293, 0.20534977], dtype=float32),

'pastorjohnb7': array([0.46113786, 0.68489146], dtype=float32),

'pearly2004': array([0.04513014, 0.17096888], dtype=float32),

'perfectsliders': array([0.53388596, 0.16201584], dtype=float32),

'picc\_susan': array([0.27017626, 0.3604237 ], dtype=float32),

'piercebboop': array([0.9811178, 0.7507475], dtype=float32),

'pinklily2003': array([0.99400455, 0.36389694], dtype=float32),

'pkelley196': array([0.472913 , 0.12175436], dtype=float32),

'pnehlen': array([0.56757414, 0.17582826], dtype=float32),

'princesssofpwr': array([0.17403209, 0.90411425], dtype=float32),

'profmichelson': array([0.9883285 , 0.37518552], dtype=float32),

'rabiasquared': array([0.11010519, 0.8170991 ], dtype=float32),

'razzleberry6769': array([0.45108834, 0.2871033 ], dtype=float32),

'realJoshuaHall': array([0.8683147, 0.280477 ], dtype=float32),

'realPolitiDiva': array([0.00773751, 0.08482227], dtype=float32),

'redostoneage': array([0.46733028, 0.37510914], dtype=float32),

'reg1776': array([0.4283473 , 0.20454286], dtype=float32),

'resterkat57': array([0.94416004, 0.50183666], dtype=float32),

'rfscottga': array([0.7586247 , 0.71876585], dtype=float32),

'richard73240238': array([0.11350908, 0.77536476], dtype=float32),

'rightlyaligned': array([0.36887485, 0.61197704], dtype=float32),

'rlturn1773': array([0.59640396, 0.40505677], dtype=float32),

'rob\_radams20': array([0.6750351 , 0.00602789], dtype=float32),

'robby\_snyder': array([0.34276384, 0.30412078], dtype=float32),

'robertthedeuce': array([0.19422296, 0.57245696], dtype=float32),

'roblee4': array([0.40278718, 0.12183953], dtype=float32),

'ronkerney': array([0.9475818, 0.8411639], dtype=float32),

'sakimasse': array([0.9168488, 0.5176235], dtype=float32),

'sallykohn': array([0.8327162 , 0.84605485], dtype=float32),

'samswey': array([0.70626616, 0.47559103], dtype=float32),

'sapmcdowell': array([0.098184 , 0.94318056], dtype=float32),

'sarahwoodwriter': array([0.5946436, 0.62415 ], dtype=float32),

'say3738': array([0.63440096, 0.8494318 ], dtype=float32),

'showlady102': array([0.02931972, 0.6359004 ], dtype=float32),

'sigstarget': array([0.54286045, 0.8589168 ], dtype=float32),

'sisp101': array([0.46335778, 0.0583869 ], dtype=float32),

'sisterwash65': array([0.7326012 , 0.67809534], dtype=float32),

'southernlady111': array([0.35013962, 0.19535235], dtype=float32),

'stanscott53': array([0.64816314, 0.85842764], dtype=float32),

'steph93065': array([0.4258677 , 0.78818715], dtype=float32),

'suzanb1776': array([0.52128386, 0.2660265 ], dtype=float32),

'sweetatertot2': array([0.4382144 , 0.76509607], dtype=float32),

'tamaraleighllc': array([0.94436556, 0.4911905 ], dtype=float32),

'texasGSDgirl': array([0.15867914, 0.41992316], dtype=float32),

'thatz2bad4u': array([0.04589523, 0.7109987 ], dtype=float32),

'thebradfordfile': array([0.7333954 , 0.99461037], dtype=float32),

'thecjpearson': array([0.7013598, 0.7075811], dtype=float32),

'thedreadedaw': array([0.1798238 , 0.38442123], dtype=float32),

'thequianafulton': array([0.09342707, 0.8374661 ], dtype=float32),

'therealcornett': array([0.53155136, 0.53182757], dtype=float32),

'thinkprogress': array([0.15112746, 0.39887628], dtype=float32),

'tkinder': array([0.35591486, 0.7625478 ], dtype=float32),

'toddstarnes': array([0.8423875, 0.6141724], dtype=float32),

'tponews': array([0.5351414 , 0.42297325], dtype=float32),

'tricia\_fierro': array([0.9879361, 0.8353431], dtype=float32),

'trumps\_feed': array([0.7270799, 0.6711266], dtype=float32),

'twiez07': array([0.0169915 , 0.22553489], dtype=float32),

'uafanct': array([0.09332671, 0.29686078], dtype=float32),

'ummmno21': array([0.01528584, 0.92197895], dtype=float32),

'up\_viking': array([0.99247843, 0.09885129], dtype=float32),

'usn112006': array([0.731073 , 0.16106902], dtype=float32),

'victorvr': array([0.5447634 , 0.34702516], dtype=float32),

'votevets': array([0.6630428, 0.7100752], dtype=float32),

'votolatino': array([0.2953973 , 0.11751797], dtype=float32),

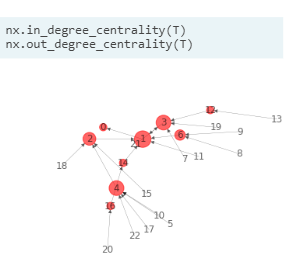
'wattsjim': array([0.44422105, 0.03632335], dtype=float32),

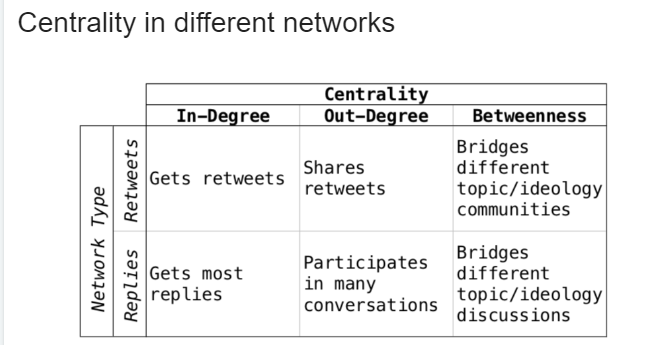
'whitneycovfefe': array([0.35986733, 0.7977326 ], dtype=float32),

'wormycat': array([0.19608405, 0.02734078], dtype=float32),

'yellowredsparks': array([0.22592354, 0.8518189 ], dtype=float32)}

**Individual-level network metrics**





**In-degree centrality**

Centrality is a measure of importance of a node to a network. There are many different types of centrality and each of them has slightly different meaning in Twitter networks. We are first focusing on degree centrality, since its calculation is straightforward and has an intuitive explanation.

For directed networks like Twitter, we need to be careful to distinguish between in-degree and out-degree centrality, especially in retweet networks. In-degree centrality for retweet networks signals users who are getting many retweets.

networkx has been imported as nx. Also, the networks G\_rt and G\_reply and column\_names = ['screen\_name', 'degree\_centrality'] have been loaded for you.

**Instructions**

**100 XP**

* Calculate in-degree centrality for the retweet network with nx.in\_degree\_centrality() and store it in rt\_centrality.
* Do the same for the reply network and store it in reply\_centrality.
* Pass the items (i.e. the key-value tuples) of the reply centralities to the DataFrame constructor.
* Do the same for the reply network.

# Generate in-degree centrality for retweets

rt\_centrality = nx.in\_degree\_centrality(G\_rt)

# Generate in-degree centrality for replies

reply\_centrality = nx.in\_degree\_centrality(G\_reply)

# Store centralities in DataFrame

rt = pd.DataFrame(list(rt\_centrality.items()), columns = column\_names)

reply = pd.DataFrame(list(reply\_centrality.items()), columns = column\_names)

# Print first five results in descending order of centrality

print(rt.sort\_values('degree\_centrality', ascending = False).head())

# Print first five results in descending order of centrality

print(reply.sort\_values('degree\_centrality', ascending = False).head())

<script.py> output:

screen\_name degree\_centrality

280 FoxNews 0.055993

1587 WhiteHouse 0.031059

282 ScottPresler 0.029746

1485 TomiLahren 0.019685

2219 KyleKulinski 0.017060

screen\_name degree\_centrality

2030 realDonaldTrump 0.057230

4 POTUS 0.019458

320 FoxNews 0.013735

1851 WhiteHouse 0.011064

508 FLOTUS 0.010301

**Betweenness Centrality**

Betweenness centrality for retweet and reply networks signals users who bridge between different Twitter communities. These communities may be tied together by topic or ideology.

networkx has been imported as nx. The networks G\_rt and G\_reply, and column\_names = ['screen\_name', 'betweenness\_centrality'] have been loaded for you.

**Instructions**

**100 XP**

* Calculate betweenness centrality for the retweet network using nx.betweenness\_centrality().
* Do the same for the reply network.
* Create a DataFrame out of retweet centralities.
* Do the same for the reply network.

# Generate betweenness centrality for retweets

rt\_centrality = nx.betweenness\_centrality(G\_rt)

# Generate betweenness centrality for replies

reply\_centrality = nx.betweenness\_centrality(G\_reply)

# Store centralities in data frames

rt = pd.DataFrame(list(rt\_centrality.items()), columns=column\_names)

reply = pd.DataFrame(list(reply\_centrality.items()), columns = column\_names)

# Print first five results in descending order of centrality

print(rt.sort\_values('betweenness\_centrality', ascending = False).head())

# Print first five results in descending order of centrality

print(reply.sort\_values('betweenness\_centrality', ascending = False).head())

<script.py> output:

screen\_name betweenness\_centrality

280 FoxNews 0.000026

157 Public\_Citizen 0.000009

1753 ChristiChat 0.000005

109 guypbenson 0.000005

52 johncardillo 0.000004

screen\_name betweenness\_centrality

1333 ScottPresler 1.019365e-06

1168 thebestcloser 2.912471e-07

2011 HRCNJVolunteers 2.912471e-07

721 RRN3 2.912471e-07

651 johncusack 1.456236e-07

**Ratios**

While not strictly a measure of importance to a network, the idea of being "ratio'd" is a network measure which is particular to Twitter and is typically used to judge the unpopularity of a tweet. "The Ratio," as it is called, is calculated by taking the number of replies and dividing it by the number of retweets. For our purposes, it makes conceptual sense to take only the in-degrees of both the retweet and reply networks.

The networks G\_rt and G\_reply, and column\_names = ['screen\_name', 'degree'] have been loaded for you.

**Instructions**

**100 XP**

* Calculate the in-degree for the retweet network with the graph method .in\_degree().
* Do the same for the reply network.
* Merge the two DataFrames together using .merge().
* Calculate the ratio. The column names are degree\_reply and degree\_rt.

# Calculate in-degrees and store in DataFrame

degree\_rt = pd.DataFrame(list(G\_rt.in\_degree()), columns = column\_names)

degree\_reply = pd.DataFrame(list(G\_reply.in\_degree()), columns = column\_names)

# Merge the two DataFrames on screen name

ratio = degree\_rt.merge(degree\_reply, on = 'screen\_name', suffixes = ('\_rt', '\_reply'))

# Calculate the ratio

ratio['ratio'] = ratio['degree\_reply'] / ratio['degree\_rt']

# Exclude any tweets with less than 5 retweets

ratio = ratio[ratio['degree\_rt'] >= 5]

# Print out first five with highest ratio

print(ratio.sort\_values('ratio', ascending = False).head())

<script.py> output:

screen\_name degree\_rt degree\_reply ratio

69 SpeakerRyan 8 15 1.875

66 NBCNews 20 18 0.900

74 benshapiro 5 4 0.800

12 SenateGOP 5 3 0.600

43 CBSThisMorning 6 3 0.500

#### **Chapter 4 :**

#### **Putting Twitter data on the map**

**Motivations**

You're interested in how your product changes in popularity across time in a single location. Location-based analysis is the most adequate for this analytics question.

**Answer the question**

**50 XP**

**Possible Answers**

* 

True

press1

* 

False **(A)**

press2

Correct! Location-based analysis would not be appropriate for this. We need some variation by geography.

**Comparisons**

You collect some Twitter data regarding your product and find that people are tweeting more about it in State A versus State B. What is the most plausible inference you could draw from these data?

**Answer the question**

**50 XP**

**Possible Answers**

* 

People in State A enjoy the product more than people in State B.

press1

* 

People with their locations enabled in State A are mentioning the product more than those in State B. **(A)**

press2

* 

People in State A are speaking about the product more than in State B.

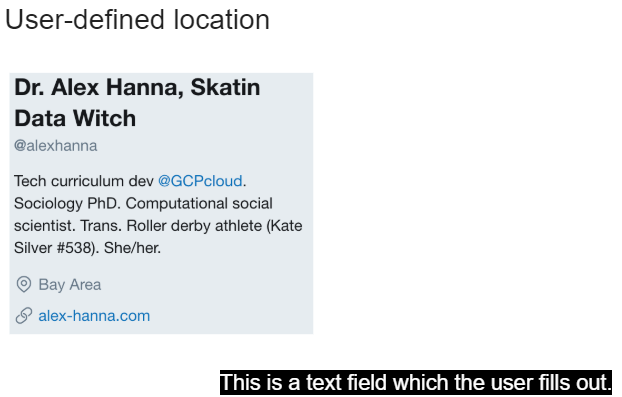
press3

* 

People with their locations enabled in State A enjoy the product more than those in State B.

press4

Correct! We are observing people with their locations enabled and they are talking about the product more in State A compared to State B.



**Coordinates and bounding boxes**

**True** or **False**: The bounding box is a single set of points which depict latitude and longitude.

**Answer the question**

**50 XP**

**Possible Answers**

* 

True

press1

* 

False **(A)**

press2

Correct! The bounding box is a set of four coordinates which outline a box-like geographical area.

**Accessing user-defined location**

In the slides, we saw that we could obtain user location via user-generated text, including the tweet itself and the location field in the user's description. These are the two most imprecise methods of obtaining user location, but also possibly more readily available.

In this exercise, you're going extract the user-defined location from a single example tweet as well as a large set of tweets. We've added another line to our flatten\_tweets() function which will allow you to access user-defined location within the data frame.

tweet\_obj['user-location'] = tweet\_obj['user']['location']

In addition, the single tweet in JSON format tweet\_json and the State of the Union tweets in JSON format tweets\_sotu\_json have been loaded for you.

**Instructions**

**100 XP**

**Instructions**

**100 XP**

* Print out the user-defined location of a single tweet in the Twitter JSON.
* Flatten the Twitter JSON of the State of the Union tweets.
* Print out the first 10 user-defined locations in tweets\_sotu with the value\_counts() and head() functions.

# Print out the location of a single tweet

print(tweet\_json['user']['location'])

# Flatten and load the SOTU tweets into a dataframe

tweets\_sotu = pd.DataFrame(flatten\_tweets(tweets\_sotu\_json))

# Print out top five user-defined locations

print(tweets\_sotu['user-location'].value\_counts().head())

<script.py> output:

Toronto, ON

Washington, DC 20

Los Angeles, CA 13

New York, NY 11

Brooklyn, NY 8

San Francisco, CA 7

Name: user-location, dtype: int64

**Accessing bounding box**

Most tweets which have coordinate-level geographical information attached to them typically come in the form of a bounding box. Bounding boxes are a set of four longitudinal/latitudinal coordinates which denote a particular area in which the user can be located. The bounding box is located in the place value of the Twitter JSON.

The dataset has been loaded for you as a DataFrame in tweets\_sotu.

**Instructions**

**100 XP**

* Complete the getBoundingBox() function by accessing the coordinates value within the bounding box dictionary.
* Apply the getBoundingBox() function to the appropriate column in the tweets\_sotu DataFrame.

def getBoundingBox(place):

""" Returns the bounding box coordinates."""

return place['bounding\_box']['coordinates']

# Apply the function which gets bounding box coordinates

bounding\_boxes = tweets\_sotu['place'].apply(getBoundingBox)

# Print out the first bounding box coordinates

print(bounding\_boxes.values[0])

<script.py> output:

[[[-94.043628, 28.855128], [-94.043628, 33.019544], [-88.758389, 33.019544], [-88.758389, 28.855128]]]

**Calculating the centroid**

The bounding box can range from a city block to a whole state or even country. For simplicity's sake, one way we can deal with handling these data is by translating the bounding box into what's called a *centroid*, or the center of the bounding box. The calculation of the centroid is straight forward -- we calculate the midpoints of the lines created by the latitude and longitudes.

numpy has been imported as np.

**Instructions**

**100 XP**

* Obtain the first set of coordinates from the place JSON.
* Calculate the central longitude by adding up the longitude list and dividing by two.
* Do the same for the latitudes.
* Apply the calculateCentroid() function to the place column.

def calculateCentroid(place):

""" Calculates the centroid from a bounding box."""

# Obtain the coordinates from the bounding box.

coordinates = place['bounding\_box']['coordinates'][0]

longs = np.unique( [x[0] for x in coordinates] )

lats = np.unique( [x[1] for x in coordinates] )

if len(longs) == 1 and len(lats) == 1:

# return a single coordinate

return (longs[0], lats[0])

elif len(longs) == 2 and len(lats) == 2:

# If we have two longs and lats, we have a box.

central\_long = np.sum(longs) / 2

central\_lat = np.sum(lats) / 2

else:

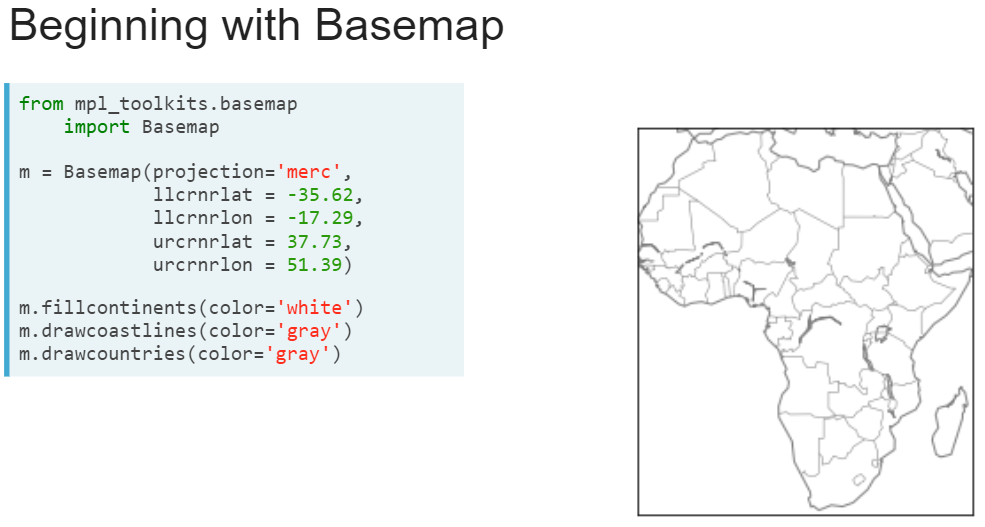
raise ValueError("Non-rectangular polygon not supported: %s" %

",".join(map(lambda x: str(x), coordinates)) )

return (central\_long, central\_lat)

# Calculate the centroids of place

centroids = tweets\_sotu['place'].apply(calculateCentroid)



# Creating Basemap map

Basemap allows you to create maps in Python. The library builds projections for latitude and longitude coordinates and then passes the plotting work on to matplotlib. This means you can build extra features based on the power of matplotlib.

In this exercise, we're going to set up a map of the continental United States on a Mercator projection. The corner coordinates of this map are provided for you below.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Import Basemap from mpl\_toolkits.basemap.
* Instantiate Basemap. Use each element of us\_boundingbox in order as arguments.

# Import Basemap

from mpl\_toolkits.basemap import Basemap

import matplotlib.pyplot as plt

# Set up the US bounding box

us\_boundingbox = [-125, 22, -64, 50]

# Set up the Basemap object

m = Basemap(llcrnrlon = us\_boundingbox[0],

llcrnrlat = us\_boundingbox[1],

urcrnrlon = us\_boundingbox[2],

urcrnrlat = us\_boundingbox[3],

projection='merc')

##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))
* Instantiate Basemap. Use each element of us\_boundingbox in order as arguments.
* Draw the continents , coastlines, and countries with Basemap methods .fillcontinents(), .drawcoastlines(), and .drawcountries(), respectively.

# Import Basemap

from mpl\_toolkits.basemap import Basemap

import matplotlib.pyplot as plt

# Set up the US bounding box

us\_boundingbox = [-125, 22, -64, 50]

# Set up the Basemap object

m = Basemap(llcrnrlon = us\_boundingbox[0],

llcrnrlat = us\_boundingbox[1],

urcrnrlon = us\_boundingbox[2],

urcrnrlat = us\_boundingbox[3],

projection='merc')

# Draw continents in white,

# coastlines and countries in gray

m.fillcontinents(color='white')

m.drawcoastlines(color='gray')

m.drawcountries(color='gray')

# Draw the states and show the plot

m.drawstates(color='gray')

plt.show()



**Plotting centroid coordinates**

Because we can't plot whole bounding boxes, we summarize the bounding box location into a single point called a centroid. Plotting these on a Basemap map is straightforward. Once we calculate the centroids, we separate the longitudes and latitudes, then pass to the .scatter() method.

The Basemap object m has been created for you. The dataset tweets\_sotu and function calculateCentroid() have also been loaded.

**Instructions**

**100 XP**

* Calculate the centroids and store in centroids.
* Set the argument zorder in fillcontinents such that the continents appear behind the points.
* Plot the points. Remember to set the latlon argument to the correct value.

# Calculate the centroids for the dataset

# and isolate longitudue and latitudes

centroids = tweets\_sotu['place'].apply(calculateCentroid)

lon = [x[0] for x in centroids]

lat = [x[1] for x in centroids]

# Draw continents, coastlines, countries, and states

m.fillcontinents(color='white', zorder = 0)

m.drawcoastlines(color='gray')

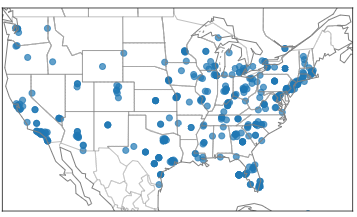
m.drawcountries(color='gray')

m.drawstates(color='gray')

# Draw the points and show the plot

m.scatter(lon , lat , latlon = True, alpha = 0.7)

plt.show()



**Coloring by sentiment**

We want to be able to differentiate by place with our Twitter analysis. One distinguishing factor between places is how the State of the Union speech was received. For this purpose, we'll use the sentiment analysis we covered in Chapter 2 to evaluate how the speech was received in different parts of the country.

The tweets\_sotu dataset has been loaded for you, as well as lon, lat, and the Basemap map m. SentimentIntensityAnalyzer is instantiated as sid in your workspace.

**Instructions**

**100 XP**

* Calculate the sentiment scores and store them.
* Draw the points, setting the color argument to sentiment\_score and the colormap to 'coolwarm'.

# Generate sentiment scores

sentiment\_scores = tweets\_sotu['text'].apply(sid.polarity\_scores)

# Isolate the compound element

sentiment\_scores = [x['compound'] for x in sentiment\_scores]

# Draw the points

m.scatter(lon , lat , latlon = True,

c = sentiment\_scores,

cmap = 'coolwarm', alpha = 0.7)

# Show the plot

plt.show()

