Project 1: Trump, Twitter, and Text ¶

Welcome to the first project of Data 100! In this project, we will work with the Twitter API in order to analyze Donald Trump's tweets.

The project is due 11:59pm Tuesday, Feb 27, California Time.

Fair warning: This project involves significantly more challenging pandas operations than the previous homeworks. We strongly suggest you start early.

Fun:

We intended this project to be fun! You will analyze actual data from the Twitter API. You will also draw conclusions about the current (and often controversial) US President's tweet behavior. If you find yourself getting frustrated or stuck on one problem for too long, we suggest coming into office hours and working with friends in the class.

If you find yourself getting frustrated with the data we suggest you vote and/or encourage others to vote.

With that in mind, let's get started!

In [2]:
```
# Run this cell to set up your notebook
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import zipfile

# Ensure that Pandas shows at least 280 characters in columns, so we can se
pd.set_option('max_colwidth', 280)

%matplotlib inline
plt.style.use('fivethirtyeight')
import seaborn as sns
sns.set()
sns.set_context("talk")
import re
```

Downloading Recent Tweets

Since we'll be looking at Twitter data, we need to download the data from Twitter!

Twitter provides an API for downloading tweet data in large batches. The `tweepy` package makes it fairly easy to use.

In [3]:
```
## Make sure you are in your data100 conda environment if you are working l
# The following should run:
import tweepy
```

There are instructions on using `tweepy` here (http://tweepy.readthedocs.io/en/v3.5.0/getting_started.html), but we will give you example code.

Twitter requires you to have authentication keys to access their API. To get your keys, you'll have to sign up as a Twitter developer. The next question will walk you through this process.

Question 1

Follow the instructions below to get your Twitter API keys. **Read the instructions completely before starting.**

1. Create a Twitter account (https://twitter.com). You can use an existing account if you have one; if you prefer to not do this assignment under your regular account, feel free to create a throw-away account.
2. Under account settings, add your phone number to the account.
3. Create a Twitter developer account (https://dev.twitter.com/resources/signup). Attach it to your Twitter account.
4. Once you're logged into your developer account, create an application for this assignment (https://apps.twitter.com/app/new). You can call it whatever you want, and you can write any URL when it asks for a web site. You don't need to provide a callback URL.
5. On the page for that application, find your Consumer Key and Consumer Secret.
6. On the same page, create an Access Token. Record the resulting Access Token and Access Token Secret.
7. Edit the file keys.json (keys.json) and replace the placeholders with your keys.

WARNING (Please Read) !!!!

Protect your Twitter Keys

If someone has your authentication keys, they can access your Twitter account and post as you! So don't give them to anyone, and **don't write them down in this notebook**. The usual way to store sensitive information like this is to put it in a separate file and read it programmatically. That way, you can share the rest of your code without sharing your keys. That's why we're asking you to put your keys in `keys.json` for this assignment.

Avoid making too many API calls.

Twitter limits developers to a certain rate of requests for data. If you make too many requests in a short period of time, you'll have to wait awhile (around 15 minutes) before you can make more. So carefully follow the code examples you see and don't rerun cells without thinking. Instead, always save the data you've collected to a file. We've provided templates to help you do that.

Be careful about which functions you call!

This API can retweet tweets, follow and unfollow people, and modify your twitter settings. Be careful which functions you invoke! One of your instructors accidentally re-tweeted some tweets because that instructor typed `retweet` instead of `retweet_count`.

In [4]:
```
import json
key_file = 'keys.json'
# Loading your keys from keys.json (which you should have filled
# in in question 1):
with open(key_file) as f:
    keys = json.load(f)
# if you print or view the contents of keys be sure to delete the cell!
```

This cell tests the Twitter authentication. It should run without errors or warnings and display your Twitter username.

In [5]:
```
import tweepy
from tweepy import TweepError
import logging

try:
    auth = tweepy.OAuthHandler(keys["consumer_key"], keys["consumer_secret"
    auth.set_access_token(keys["access_token"], keys["access_token_secret"]
    api = tweepy.API(auth)
    print("Your username is:", api.auth.get_username())
except TweepError as e:
    logging.warning("There was a Tweepy error. Double check your API keys a
    logging.warning(e)
```

```
Your username is: fasky_based
```

Question 2

In the example below, we have loaded some tweets by @BerkeleyData. Run it and read the code.

In [6]:
```
from pathlib import Path
import json

ds_tweets_save_path = "BerkeleyData_recent_tweets.json"
# Guarding against attempts to download the data multiple
# times:
if not Path(ds_tweets_save_path).is_file():
    # Getting as many recent tweets by @BerkeleyData as Twitter will let us
    # We use tweet_mode='extended' so that Twitter gives us full 280 charac
    # This was a change introduced in September 2017.

    # The tweepy Cursor API actually returns "sophisticated" Status objects
    # will use the basic Python dictionaries stored in the _json field.
    example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id=
                                                     tweet_mode='extended').items()

    # Saving the tweets to a json file on disk for future analysis
    with open(ds_tweets_save_path, "w") as f:
        json.dump(example_tweets, f)

# Re-loading the json file:
with open(ds_tweets_save_path, "r") as f:
    example_tweets = json.load(f)
```

Assuming everything ran correctly you should be able to look at the first tweet by running the cell below.

Warning Do not attempt to view all the tweets in a notebook. It will likely freeze your browser. The following would be a **bad idea**: ```python pprint(example_tweets) ```

In [7]:
```
# Looking at one tweet object, which has type Status:
from pprint import pprint # ...to get a more easily-readable view.
pprint(example_tweets[0])
```

```
{'contributors': None,
 'coordinates': None,
 'created_at': 'Fri Feb 23 21:04:44 +0000 2018',
 'display_text_range': [0, 139],
 'entities': {'hashtags': [],
              'symbols': [],
              'urls': [],
              'user_mentions': [{'id': 921139702996127744,
                                 'id_str': '921139702996127744',
                                 'indices': [3, 17],
                                 'name': 'cybersecurity@berkeley',
                                 'screen_name': 'BerkeleyCyber'},
                                {'id': 176932593,
                                 'id_str': '176932593',
                                 'indices': [79, 90],
                                 'name': 'UC Berkeley',
                                 'screen_name': 'UCBerkeley'}]},
 'favorite_count': 0,
 'favorited': False,
```

Question 2a

What you need to do.

Re-factor the above code fragment into reusable snippets below. You should not need to make major modifications; this is mostly an exercise in understanding the above code block.

In [8]:
```
def load_keys(path):
    """Loads your Twitter authentication keys from a file on disk.

    Args:
        path (str): The path to your key file.  The file should
          be in JSON format and look like this (but filled in):
            {
                "consumer_key": "<your Consumer Key here>",
                "consumer_secret":  "<your Consumer Secret here>",
                "access_token": "<your Access Token here>",
                "access_token_secret": "<your Access Token Secret here>"
            }

    Returns:
        dict: A dictionary mapping key names (like "consumer_key") to
          key values."""
    key_file = path
    with open(key_file) as a:
        keys = json.load(a)
    return keys
```

In [9]:
```
def download_recent_tweets_by_user(user_account_name, keys):
    """Downloads tweets by one Twitter user.

    Args:
        user_account_name (str): The name of the Twitter account
          whose tweets will be downloaded.
        keys (dict): A Python dictionary with Twitter authentication
          keys (strings), like this (but filled in):
            {
                "consumer_key": "<your Consumer Key here>",
                "consumer_secret":  "<your Consumer Secret here>",
                "access_token": "<your Access Token here>",
                "access_token_secret": "<your Access Token Secret here>"
            }

    Returns:
        list: A list of Dictonary objects, each representing one tweet."""
    import tweepy
    tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id=user_acc
                                             tweet_mode='extended').items()
    return tweets
```

In [10]:
```
def save_tweets(tweets, path):
    """Saves a list of tweets to a file in the local filesystem.

    This function makes no guarantee about the format of the saved
    tweets, **except** that calling load_tweets(path) after
    save_tweets(tweets, path) will produce the same list of tweets
    and that only the file at the given path is used to store the
    tweets.  (That means you can implement this function however
    you want, as long as saving and loading works!)

    Args:
        tweets (list): A list of tweet objects (of type Dictionary) to
          be saved.
        path (str): The place where the tweets will be saved.

    Returns:
        None"""

    if not Path(path).is_file():
        with open(path, "w") as a:
            json.dump(tweets, a)
```

In [11]:
```
def load_tweets(path):
    """Loads tweets that have previously been saved.

    Calling load_tweets(path) after save_tweets(tweets, path)
    will produce the same list of tweets.

    Args:
        path (str): The place where the tweets were be saved.

    Returns:
        list: A list of Dictionary objects, each representing one tweet."""
    with open(path, "r") as a:
        tweets = json.load(a)
    return tweets
```

In [12]:
```
def get_tweets_with_cache(user_account_name, keys_path):
    """Get recent tweets from one user, loading from a disk cache if availa

    The first time you call this function, it will download tweets by
    a user.  Subsequent calls will not re-download the tweets; instead
    they'll load the tweets from a save file in your local filesystem.
    All this is done using the functions you defined in the previous cell.
    This has benefits and drawbacks that often appear when you cache data:

    +: Using this function will prevent extraneous usage of the Twitter API
    +: You will get your data much faster after the first time it's called.
    -: If you really want to re-download the tweets (say, to get newer ones
       or because you screwed up something in the previous cell and your
       tweets aren't what you wanted), you'll have to find the save file
       (which will look like <something>_recent_tweets.pkl) and delete it.

    Args:
        user_account_name (str): The Twitter handle of a user, without the
        keys_path (str): The path to a JSON keys file in your filesystem.
    """
    keys = load_keys(keys_path)
    recent_tweets = download_recent_tweets_by_user(user_account_name, keys)
    save_tweets(recent_tweets, "new_user.json")
    load_tweets(keys_path)
    return recent_tweets
```

If everything was implemented correctly you should be able to obtain roughly the last 3000 tweets by the `realdonaldtrump` . (This may take a few minutes)

In [13]:
```
# When you are done, run this cell to load @realdonaldtrump's tweets.
# Note the function get_tweets_with_cache.  You may find it useful
# later.
trump_tweets = get_tweets_with_cache("realdonaldtrump", key_file)
print("Number of tweets downloaded:", len(trump_tweets))
```

```
Number of tweets downloaded: 3227
```

In [14]:
```
assert 2000 <= len(trump_tweets) <= 4000
```

Question 2b

We are limited to how many tweets we can download. In what month is the oldest tweet from Trump?

In [15]:
```
# Enter the number of the month of the oldest tweet (e.g. 1 for January)
import time

oldest = trump_tweets[0].get("created_at")
for tweet in np.arange(len(trump_tweets)):
    date = trump_tweets[tweet].get("created_at")
    time.strptime(date, "%a %b %d %X %z %Y")
    oldest = max(oldest, date)
oldest_month = time.strptime(oldest, "%a %b %d %X %z %Y").tm_mon
oldest_month
```

Out[15]: 9

In [ ]:
```

```

Question 3

IMPORTANT! PLEASE READ

Unfortunately, Twitter prevent us from going further back in time using the public APIs. Fortunately, we have a snapshot of earlier tweets that we can combine with our new data.

We will again use the `fetch_and_cache` utility to download the dataset.

In [16]:
```
# Download the dataset
from utils import fetch_and_cache
data_url = 'http://www.ds100.org/sp18/assets/datasets/old_trump_tweets.json
file_name = 'old_trump_tweets.json.zip'

dest_path = fetch_and_cache(data_url=data_url, file=file_name)
print(f'Located at {dest_path}')
```

```
Using version already downloaded: Mon Feb 26 22:45:47 2018
MD5 hash of file: d9419cad17e76c87fe646b587f6e8ca5
Located at data/old_trump_tweets.json.zip
```

Finally, we we will load the tweets directly from the compressed file without decompressing it first.

In [17]:
```
my_zip = zipfile.ZipFile(dest_path, 'r')
with my_zip.open("old_trump_tweets.json", "r") as f:
    old_trump_tweets = json.load(f)
```

This data is formatted identically to the recent tweets we just downloaded:

In [18]:
```
pprint(old_trump_tweets[0])
```

```
{'contributors': None,
 'coordinates': None,
 'created_at': 'Wed Oct 12 14:00:48 +0000 2016',
 'entities': {'hashtags': [{'indices': [23, 38], 'text': 'CrookedHillar
y'}],
              'media': [{'display_url': 'pic.twitter.com/wjsl8ITVvk',
                         'expanded_url': 'https://twitter.com/realDonaldT
rump/status/786204978629185536/video/1',
                         'id': 786204885318561792,
                         'id_str': '786204885318561792',
                         'indices': [39, 62],
                         'media_url': 'http://pbs.twimg.com/ext_tw_video_
thumb/786204885318561792/pu/img/XqMoixLm83FzkAbn.jpg',
                         'media_url_https': 'https://pbs.twimg.com/ext_tw
_video_thumb/786204885318561792/pu/img/XqMoixLm83FzkAbn.jpg',
                         'sizes': {'large': {'h': 576,
                                             'resize': 'fit',
                                             'w': 1024},
                                   'medium': {'h': 338,
```

As a dictionary we can also list the keys:

In [19]:
```
old_trump_tweets[0].keys()
```

Out[19]:
```
dict_keys(['created_at', 'id', 'id_str', 'text', 'truncated', 'entities',
'extended_entities', 'source', 'in_reply_to_status_id', 'in_reply_to_stat
us_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_str', 'in_reply_t
o_screen_name', 'user', 'geo', 'coordinates', 'place', 'contributors', 'i
s_quote_status', 'retweet_count', 'favorite_count', 'favorited', 'retweet
ed', 'possibly_sensitive', 'lang'])
```

In [20]:
```
trump_tweets[0].keys()
```

Out[20]:
```
dict_keys(['created_at', 'id', 'id_str', 'full_text', 'truncated', 'displ
ay_text_range', 'entities', 'source', 'in_reply_to_status_id', 'in_reply_
to_status_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_str', 'in_
reply_to_screen_name', 'user', 'geo', 'coordinates', 'place', 'contributo
rs', 'is_quote_status', 'retweet_count', 'favorite_count', 'favorited',
'retweeted', 'lang'])
```

Question 3a

Merge the `old_trump_tweets` and the `trump_tweets` we downloaded from twitter into one giant list of tweets.

Important: There may be some overlap so be sure to eliminate duplicate tweets.
Hint: the `id` of a tweet is always unique.

In [21]:
```
tweeter_dict = {}
for tweet in trump_tweets:
    if tweet["id"] not in tweeter_dict.keys():
        tweeter_dict[tweet["id"]] = tweet
for tweet in old_trump_tweets:
    if tweet["id"] not in tweeter_dict.keys():
        tweeter_dict[tweet["id"]] = tweet

all_tweets = [tweeter_dict[tweet_key] for tweet_key in tweeter_dict.keys()]
len(all_tweets)
```

Out[21]:
```
6802
```

In [22]:
```
assert len(all_tweets) > len(trump_tweets)
assert len(all_tweets) > len(old_trump_tweets)
```

Question 3b

Construct a DataFrame called `trump` containing all the tweets stored in `all_tweets`. The index of the dataframe should be the ID of each tweet (looks something like `907698529606541312`). It should have these columns:

- `time`: The time the tweet was created encoded as a datetime object. (Use `pd.to_datetime` to encode the timestamp.)
- `source`: The source device of the tweet.
- `text`: The text of the tweet.
- `retweet_count`: The retweet count of the tweet.

Finally, **the resulting dataframe should be sorted by the index.**

Warning: *Some tweets will store the text in the `text` field and other will use the `full_text` field.*

In [23]:
```
 #creating the intial data for the dataframe without text
datum = [[tweet_dict["created_at"], tweet_dict["source"],
          tweet_dict["retweet_count"]] for tweet_dict in all_tweets]

#creating the list of indexes
dex = [tweet_dict["id"] for tweet_dict in all_tweets]

#putting together a dataframe
trump = pd.DataFrame(data=datum, index=dex, columns=["time", "source", "ret

#changing the time to be correctly formatted
trump["time"] = [pd.to_datetime(i) for i in trump["time"]]

#creating a comprehensive list of full_text and text
all_text = [tweet_dict["full_text"] if "full_text" in tweet_dict else tweet

trump["text"] = all_text

trump
```

Out[23]:

	time	source	retweet_count	
968621347294318592	2018-02-27 22:58:19	Twitter for iPhone	6298	.@SenatorWicke supporter massive Tax Cut help on cut
968620661349470209	2018-02-27 22:55:36	Twitter for iPhone	8206	Texas LC Ge wasn't the pc back him now. A me from the Texas." Also s and Railroad
968550893015707649	2018-02-27 18:18:22	Twitter for iPhone	9458	"American c they've been sin boosting confide
				I want to encour to vote in the pr

In [24]:
```
assert isinstance(trump, pd.DataFrame)
assert trump.shape[0] < 8000
assert trump.shape[1] >= 4
assert 831846101179314177 in trump.index
assert 753063644578144260 in trump.index
assert all(col in trump.columns for col in ['time', 'source', 'text', 'retw
# If you fail these tests, you probably tried to use __dict__ or _json to r
assert np.sometrue([('Twitter for iPhone' in s) for s in trump['source'].un
assert trump['time'].dtype == np.dtype('<M8[ns]')
assert trump['text'].dtype == np.dtype('O')
assert trump['retweet_count'].dtype == np.dtype('int64')
```

Question 4: Tweet Source Analysis

In the following questions, we are going to find out the charateristics of Trump tweets and the devices used for the tweets.

First let's examine the source field:

In [25]:
```
trump['source'].unique()
```

Out[25]:
```
array([ '<a href="http://twitter.com/download/iphone" rel="nofollow">Twit
ter for iPhone</a>',
       '<a href="https://studio.twitter.com" rel="nofollow">Media Studio
</a>',
       '<a href="http://twitter.com/#!/download/ipad" rel="nofollow">Twit
ter for iPad</a>',
       '<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a
>',
       '<a href="https://ads.twitter.com" rel="nofollow">Twitter Ads</a
>',
       '<a href="http://twitter.com/download/android" rel="nofollow">Twit
ter for Android</a>',
       '<a href="https://periscope.tv" rel="nofollow">Periscope</a>',
       '<a href="http://instagram.com" rel="nofollow">Instagram</a>',
       '<a href="https://mobile.twitter.com" rel="nofollow">Mobile Web (M
5)</a>'], dtype=object)
```

Question 4a

Remove the HTML tags from the source field.

Hint: Use `trump['source'].str.replace` and your favorite regular expression.

In [26]:
```
## Uncomment and complete
trump['source'] = trump['source'].str.replace(r'<[^>]+>', '')
trump['source']
```

Out[26]:
```
968621347294318592     Twitter for iPhone
968620661349470209     Twitter for iPhone
968550893015707649     Twitter for iPhone
968549117625602054     Twitter for iPhone
968468176639004672     Twitter for iPhone
968467243192537088     Twitter for iPhone
968462966864609280     Twitter for iPhone
968455547094753281     Twitter for iPhone
967847560248426498     Twitter for iPhone
967609114238050304     Twitter for iPhone
967597887545896961     Twitter for iPhone
967597639293382657     Twitter for iPhone
967564998238142471     Twitter for iPhone
967563946063523840     Twitter for iPhone
967545724362739712     Twitter for iPhone
967539664692350977     Twitter for iPhone
967538684789739520     Twitter for iPhone
967526110249537542     Twitter for iPhone
967508938425069568     Twitter for iPhone
967506350283599874     Twitter for iPhone
967493467046899712     Twitter for iPhone
967472757025001472     Twitter for iPhone
967389553362423808     Twitter for iPhone
967388904952344577     Twitter for iPhone
967168890232082433     Twitter for iPhone
967146956094083073     Twitter for iPhone
967084806726127616     Twitter for iPhone
967083810981597185     Twitter for iPhone
967083281974951939     Twitter for iPhone
967023714268319744     Twitter for iPhone
                              ...
787425145489072128    Twitter for Android
787359730465329152    Twitter for Android
787355131062943744     Twitter for iPhone
787320961934688257     Twitter for iPhone
787320326573228032     Twitter for iPhone
787319977711923200     Twitter for iPhone
787267564405653505    Twitter for Android
787266044213723138     Twitter for iPhone
787258211283918848    Twitter for Android
787244543003467776    Twitter for Android
787127225581707265     Twitter for iPhone
787099202291634177     Twitter for iPhone
787025537483046913     Twitter Web Client
787012170630455297     Twitter for iPhone
786950598826532864     Twitter for iPhone
786737820669009921     Twitter for iPhone
786716631644897280     Twitter for iPhone
786710113163812864     Twitter for iPhone
786709861245526017     Twitter for iPhone
786700864752914433     Twitter for iPhone
786691718062235649     Twitter for iPhone
786658827429154816     Twitter for iPhone
```

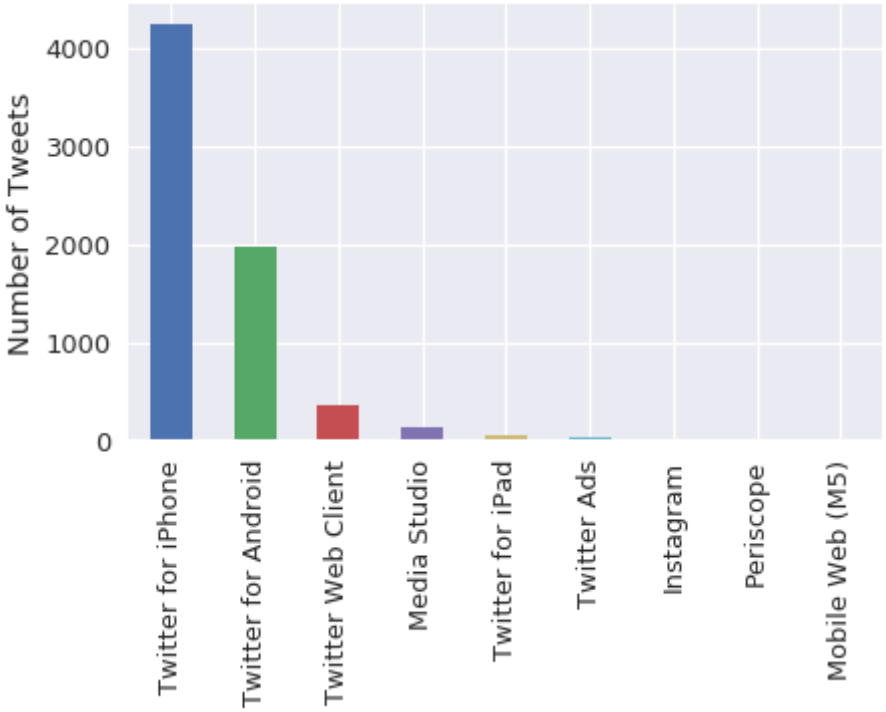
```
786589454991499264     Twitter for iPhone
786589172911964161     Twitter for iPhone
786565208663785476     Twitter for iPhone
786560925113266176     Twitter Web Client
786554517680693248    Twitter for Android
786340623804751872     Twitter for iPhone
786310855843512320       Twitter for iPad
786285509668696065     Twitter for iPhone
Name: source, Length: 6802, dtype: object
```

In [27]:
```
from datetime import datetime
ELEC_DATE = datetime(2016, 11, 8)
INAUG_DATE = datetime(2017, 1, 20)
assert set(trump[(trump['time'] > ELEC_DATE) & (trump['time'] < INAUG_DATE)
 'Twitter Web Client',
 'Twitter for Android',
 'Twitter for iPhone'])
```

We can see in the following plot that there are two device types that are more commonly used

In [28]:
```
trump['source'].value_counts().plot(kind="bar")
plt.ylabel("Number of Tweets")
```

Out[28]:
```
Text(0,0.5,'Number of Tweets')
```



Question 4b

Is there a difference between his Tweet behavior across these devices? We will attempt to answer this question in our subsequent analysis.

First, we'll take a look at whether Trump's tweets from an Android come at different times than his tweets from an iPhone. Note that Twitter gives us his tweets in the UTC timezone (https://www.wikiwand.com/en/List_of_UTC_time_offsets) (notice the `+0000` in the first few tweets)

In [29]:
```
for t in trump_tweets[0:3]:
    print(t['created_at'])
```

```
Tue Feb 27 22:58:19 +0000 2018
Tue Feb 27 22:55:36 +0000 2018
Tue Feb 27 18:18:22 +0000 2018
```

We'll convert the tweet times to US Eastern Time, the timezone of New York and Washington D.C., since those are the places we would expect the most tweet activity from Trump.

In [30]:
```
trump['est_time'] = (
    trump['time'].dt.tz_localize("UTC") # Set initial timezone to UTC
                 .dt.tz_convert("EST") # Convert to Eastern Time
)
trump.head()
```

Out[30]:

	time	source	retweet_count	text	est_time
968621347294318592	2018-02-27 22:58:19	Twitter for iPhone	6298	.@SenatorWicker of Mississippi has been a great supporter and incredible help in getting our massive Tax Cut Bill done and approved. Also big help on cutting regs. I am with him in his re-election all the way!	2018-02-27 17:58:19-05:00
968620661349470209	2018-02-27 22:55:36	Twitter for iPhone	8206	Texas LC George P. Bush backed me when it wasn't the politically correct thing to do, and I back him now. Also, AC Sid Miller has been with me from the beginning, he is "Trump's Man in Texas." Also support Comptroller Glenn Hegar, and Railroad Commissioner Christi Craddick.	2018-02-27 17:55:36-05:00
968550893015707649	2018-02-27 18:18:22	Twitter for iPhone	9458	"American consumers are the most confident they've been since 2000....A strong job market is boosting confidence. The unemployment rate has stayed at a 17-year low." https://t.co/aL7aVoR7XC	2018-02-27 13:18:22-05:00
968549117625602054	2018-02-27 18:11:19	Twitter for iPhone	19528	I want to encourage all of my many Texas friends to vote in the primary for Governor Greg Abbott, Senator Ted Cruz, Lt. Gov. Dan Patrick, and Attorney General Ken Paxton. They are helping me to Make America Great Again! Vote early or on March 6th.	2018-02-27 13:11:19-05:00
968468176639004672	2018-02-27 12:49:41	Twitter for iPhone	20274	WITCH HUNT!	2018-02-27 07:49:41-05:00

What you need to do:

Add a column called `hour` to the `trump` table which contains the hour of the day as floating point number computed by:

$$\text{hour} + \frac{\text{minute}}{60} + \frac{\text{second}}{60^2}$$

In [31]:
```
time = trump['est_time']
trump['hour'] = time.dt.hour + (1/60) * time.dt.minute + (1/60**2) * time.d
trump['hour']
```

Out[31]:

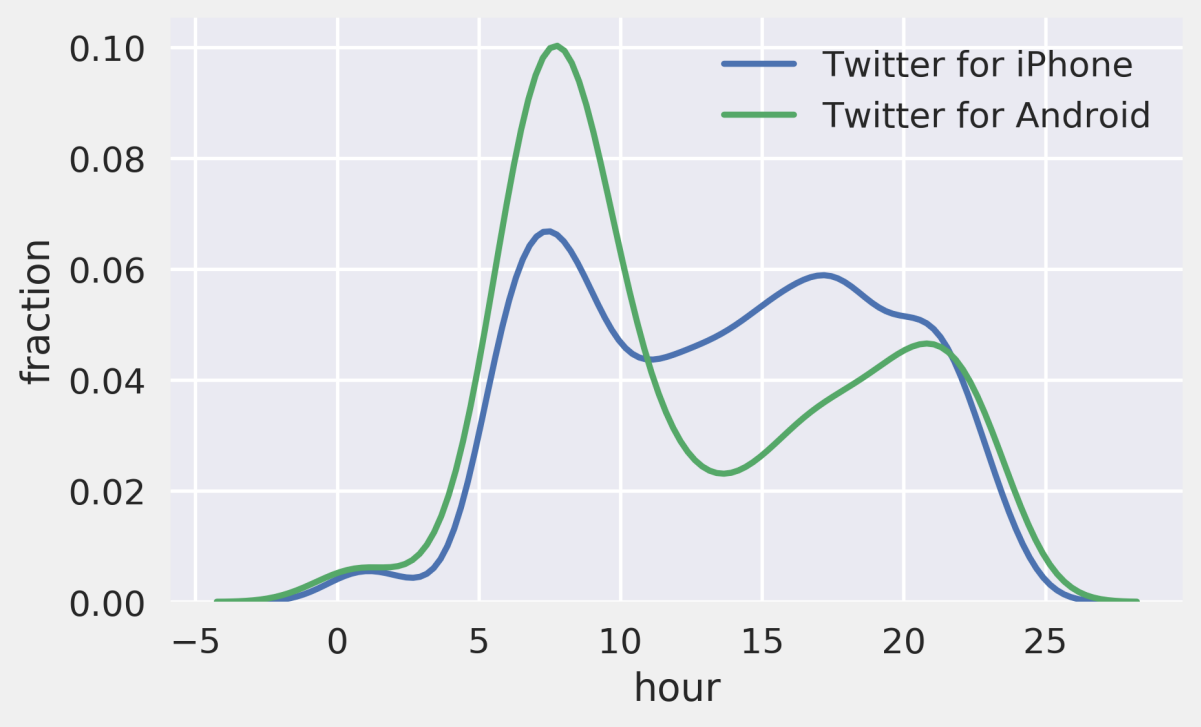
968621347294318592	17.971944
968620661349470209	17.926667
968550893015707649	13.306111
968549117625602054	13.188611
968468176639004672	7.828056
968467243192537088	7.766111
968462966864609280	7.483056
968455547094753281	6.991667
967847560248426498	14.726111
967609114238050304	22.934444
967597887545896961	22.191111
967597639293382657	22.174444
967564998238142471	20.012778
967563946063523840	19.943056
967545724362739712	18.736389
967539664692350977	18.335000
967538684789739520	18.270278
967526110249537542	17.437500
967508938425069568	16.300278
967506350283599874	16.128611
967493467046899712	15.275556
967472757025001472	13.903889
967389553362423808	8.393611
967388904952344577	8.350556
967168890232082433	17.779722
967146956094083073	16.326944
967084806726127616	12.211111
967083810981597185	12.145000
967083281974951939	12.110000
967023714268319744	8.165000
	...
787425145489072128	17.821667
787359730465329152	13.489444
787355131062943744	13.185000
787320961934688257	10.921944
787320326573228032	10.880000
787319977711923200	10.856944
787267564405653505	7.385556
787266044213723138	7.285000
787258211283918848	6.766111
787244543003467776	5.861111
787127225581707265	22.091389
787099202291634177	20.235556
787025537483046913	15.356944
787012170630455297	14.471667
786950598826532864	10.393889
786737820669009921	20.301944
786716631644897280	18.898889
786710113163812864	18.467222
786709861245526017	18.450278
786700864752914433	17.854722
786691718062235649	17.248889
786658827429154816	15.070556

```
786589454991499264    10.476111
786589172911964161    10.457500
786565208663785476     8.870556
786560925113266176     8.586667
786554517680693248     8.162500
786340623804751872    17.996667
786310855843512320    16.025278
786285509668696065    14.346667
Name: hour, Length: 6802, dtype: float64
```

In [32]:
```
assert np.isclose(trump.loc[690171032150237184]['hour'], 8.93639)
```

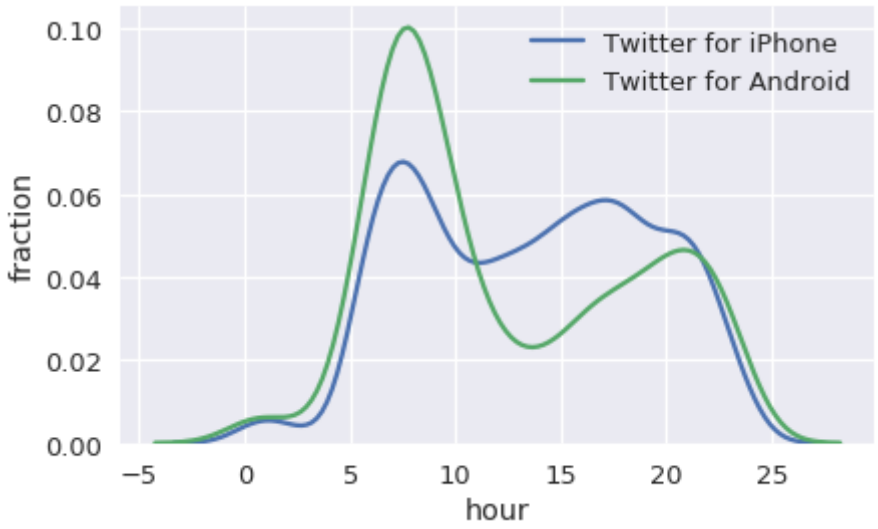
Question 4c

Use this data along with the seaborn `distplot` function to examine the distribution over hours of the day in `easter time` that trump tweets on each device for the 2 most commonly used devices. Your plot should look similar to the following.



In [33]:
```
iphone = trump[trump["source"] == "Twitter for iPhone"]["hour"]
android = trump[trump["source"] == "Twitter for Android"]["hour"]
sns.distplot(iphone, hist=False, label="Twitter for iPhone")
sns.distplot(android, hist=False, label="Twitter for Android")
plt.ylabel("fraction")
plt.legend()
```

Out[33]:
```
<matplotlib.legend.Legend at 0x7fefe9e1bf60>
```



Question 4d

Are there any striking differences between these curves. If someone told you that Trump tends to tweet early in the morning and then later in the evening, which device might you conclude is most likely his?

The tweets coming from the iPhone are more dispersed throughout the day, whereas the Android tweets are much more concentrated in the morning and evening. One could conclude that Trump uses an Android.

Question 5

Let's now look at which device he has used over the entire time period of this dataset.

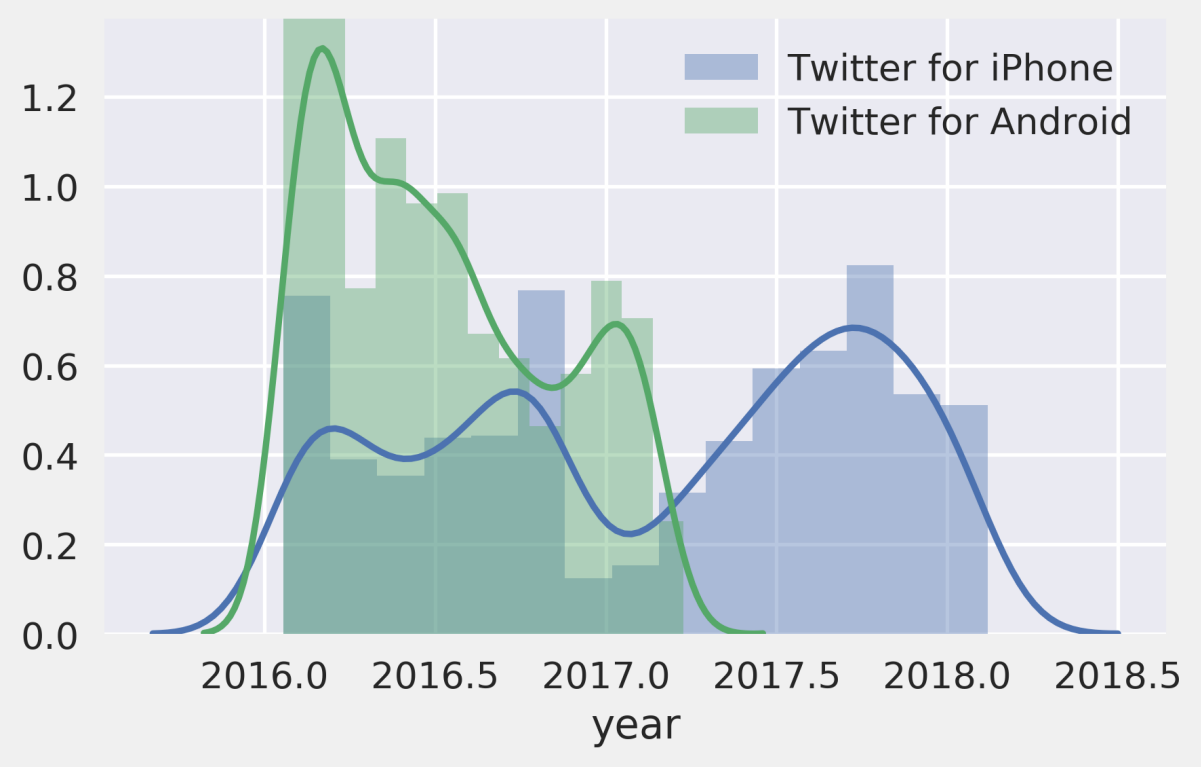
To examine the distribution of dates we will convert the date to a fractional year that can be plotted as a distribution.

(Code borrowed from https://stackoverflow.com/questions/6451655/python-how-to-convert-datetime-dates-to-decimal-years (https://stackoverflow.com/questions/6451655/python-how-to-convert-datetime-dates-to-decimal-years))

In [34]:
```
import datetime
def year_fraction(date):
    start = datetime.date(date.year, 1, 1).toordinal()
    year_length = datetime.date(date.year+1, 1, 1).toordinal() - start
    return date.year + float(date.toordinal() - start) / year_length


trump['year'] = trump['time'].apply(year_fraction)
```

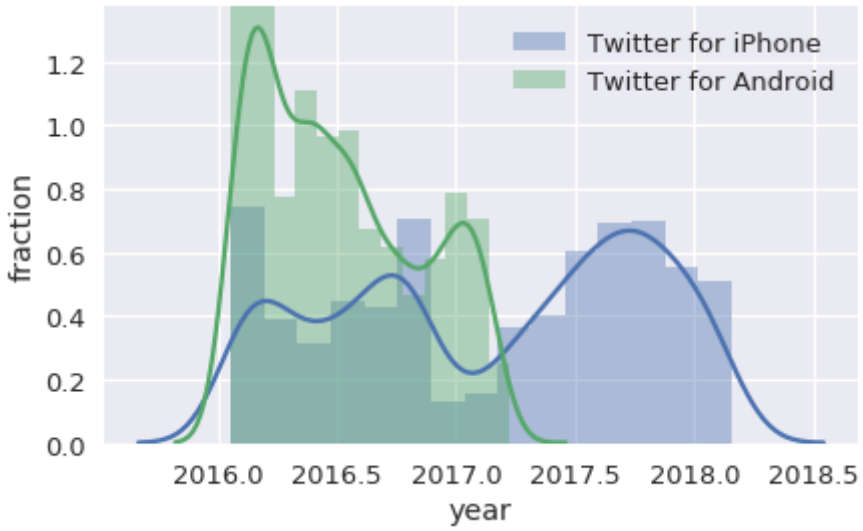
Question 5a

Use the `sns.distplot` to overlay the distributions of the 2 most frequently used web technologies over the years. Your final plot should look like:



In [35]:
```
iphone = trump[trump["source"] == "Twitter for iPhone"]["year"]
android = trump[trump["source"] == "Twitter for Android"]["year"]
sns.distplot(iphone, label="Twitter for iPhone")
sns.distplot(android, label="Twitter for Android")
plt.ylabel("fraction")
plt.legend()
```

Out[35]:
```
<matplotlib.legend.Legend at 0x7fefe9e04b38>
```



Question 5b

According to the plot, Trump's tweets come from many different sources. It turns out that many of his tweets were not from Trump himself but from his staff. [Take a look at this Verge article. (https://www.theverge.com/2017/3/29/15103504/donald-trump-iphone-using-switched-android)](https://www.theverge.com/2017/3/29/15103504/donald-trump-iphone-using-switched-android)

Does the data support the information in the article? What else do you find out about changes in Trump's tweets sources from the plot?

This data does support the findings made in the article. Trump did change away from Android to iPhone, and his tweets are coming from both him and his staffers now. Trump was initally very active on his twitter (via Android) before he was sworn in as president. After he took office, that number went down and staffers began tweeting for him much more.

Question 6: Sentiment Analysis

It turns out that we can use the words in Trump's tweets to calculate a measure of the sentiment of the tweet. For example, the sentence "I love America!" has positive sentiment, whereas the sentence "I hate taxes!" has a negative sentiment. In addition, some words have stronger positive / negative sentiment than others: "I love America." is more positive than "I like America."

We will use the VADER (Valence Aware Dictionary and sEntiment Reasoner) (https://github.com/cjhutto/vaderSentiment) lexicon to analyze the sentiment of Trump's tweets. VADER is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media which is great for our usage.

The VADER lexicon gives the sentiment of individual words. Run the following cell to show the first few rows of the lexicon:

In [36]:
```
print(''.join(open("vader_lexicon.txt").readlines()[:10]))
```

```
$:	-1.5	0.80623	[-1, -1, -1, -1, -3, -1, -3, -1, -2, -1]
%)	-0.4	1.0198	[-1, 0, -1, 0, 0, -2, -1, 2, -1, 0]
%-)	-1.5	1.43178	[-2, 0, -2, -2, -1, 2, -2, -3, -2, -3]
&-:	-0.4	1.42829	[-3, -1, 0, 0, -1, -1, -1, 2, -1, 2]
&:	-0.7	0.64031	[0, -1, -1, -1, 1, -1, -1, -1, -1, -1]
( '}{' )	1.6	0.66332	[1, 2, 2, 1, 1, 2, 2, 1, 3, 1]
(%	-0.9	0.9434	[0, 0, 1, -1, -1, -1, -2, -2, -1, -2]
('-:	2.2	1.16619	[4, 1, 4, 3, 1, 2, 3, 1, 2, 1]
(':	2.3	0.9	[1, 3, 3, 2, 2, 4, 2, 3, 1, 2]
((-:	2.1	0.53852	[2, 2, 2, 1, 2, 3, 2, 2, 3, 2]
```

Question 6a

As you can see, the lexicon contains emojis too! The first column of the lexicon is the *token*, or the word itself. The second column is the *polarity* of the word, or how positive / negative it is.

(How did they decide the polarities of these words? What are the other two columns in the lexicon? See the link above.)

Read in the lexicon into a DataFrame called `sent`. The index of the DF should be the tokens in the lexicon. `sent` should have one column: `polarity` : The polarity of each token.

In [37]:
```
sent = pd.read_table("vader_lexicon.txt", header=None, usecols=[0,1])
sent = sent.rename(columns={sent.columns[0]: "token", sent.columns[1]: "pol
sent = sent.set_index("token")
sent
```

Out[37]:

token	polarity
$:	-1.5
%)	-0.4
%-)	-1.5
&-:	-0.4
&:	-0.7
('}{')	1.6
(%	-0.9
('-:	2.2
(':	2.3
((-:	2.1
(*	1.1
(-%	-0.7
(-*	1.3
(-:	1.6
(-:0	2.8
(-:<	-0.4
(-:o	1.5
(-:O	1.5
(-:{	-0.1
(-:|>*	1.9
(-;	1.3
(-;|	2.1
(8	2.6
(:	2.2
(:0	2.4
(:<	-0.2
(:o	2.5
(:O	2.5
(;	1.1
(;<	0.3

token	polarity
...	...
xd	2.8
xp	1.6
yay	2.4
yeah	1.2
yearning	0.5
yeees	1.7
yep	1.2
yes	1.7
youthful	1.3
yucky	-1.8
yummy	2.4
zealot	-1.9
zealots	-0.8
zealous	0.5
{:	1.8
|-0	-1.2
|-:	-0.8
|-:>	-1.6
|-o	-1.2
|:	-0.5
|;-)	2.2
|=	-0.4
|^:	-1.1
|o:	-0.9
||-:	-2.3
}:	-2.1
}:(	-2.0
}:)	0.4
}:-(	-2.1
}:-)	0.3

7517 rows × 1 columns

In [38]:
```
assert isinstance(sent, pd.DataFrame)
assert sent.shape == (7517, 1)
assert list(sent.index[5000:5005]) == ['paranoids', 'pardon', 'pardoned', '
assert np.allclose(sent['polarity'].head(), [-1.5, -0.4, -1.5, -0.4, -0.7])
```

Question 6b

Now, let's use this lexicon to calculate the overall sentiment for each of Trump's tweets. Here's the basic idea:

1. For each tweet, find the sentiment of each word.
2. Calculate the sentiment of each tweet by taking the sum of the sentiments of its words.

First, let's lowercase the text in the tweets since the lexicon is also lowercase. Set the `text` column of the `trump` DF to be the lowercased text of each tweet.

In [39]:
```
trump["text"] = [str.lower(i) for i in trump["text"]]
```

In [40]:
```
assert trump['text'].loc[884740553040175104] == 'working hard to get the ol
```

Question 6c

Now, let's get rid of punctuation since it'll cause us to fail to match words. Create a new column called `no_punc` in the `trump` DF to be the lowercased text of each tweet with all punctuation replaced by a single space. We consider punctuation characters to be any character that isn't a Unicode word character or a whitespace character. You may want to consult the Python documentation on regexes for this problem.

(Why don't we simply remove punctuation instead of replacing with a space? See if you can figure this out by looking at the tweet data.)

In [41]:
```
# Save your regex in punct_re
punct_re = r'[^\s\w]'
trump['no_punc'] = [re.sub(punct_re, " ", i, len(i)) for i in trump["text"]]
trump['no_punc']
```

Out[41]:
```
968621347294318592
senatorwicker of mississippi has been a great supporter and incredible he
lp in getting our massive tax cut bill done and approved  also big help o
n cutting regs  i am with him in his re election all the way
968620661349470209         texas lc george p  bush backed me when it wasn
t the politically correct thing to do  and i back him now  also  ac sid m
iller has been with me from the beginning  he is  trump s man in texas
also support comptroller glenn hegar  and railroad commissioner christi c
raddick
968550893015707649
american consumers are the most confident they ve been since 2000    a st
rong job market is boosting confidence  the unemployment rate has stayed
at a 17 year low   https   t co al7avor7xc
968549117625602054                                        i want to encourage
all of my many texas friends to vote in the primary for governor greg abb
ott  senator ted cruz  lt  gov  dan patrick  and attorney general ken pax
ton  they are helping me to make america great again  vote early or on ma
rch 6th
968468176639004672
```

In [42]:
```
assert isinstance(punct_re, str)
assert re.search(punct_re, 'this') is None
assert re.search(punct_re, 'this is ok') is None
assert re.search(punct_re, 'this is\nok') is None
assert re.search(punct_re, 'this is not ok.') is not None
assert re.search(punct_re, 'this#is#ok') is not None
assert re.search(punct_re, 'this^is ok') is not None
assert trump['no_punc'].loc[800329364986626048] == 'i watched parts of  nbc
assert trump['no_punc'].loc[894620077634592769] == 'on  purpleheartday i th
# If you fail these tests, you accidentally changed the text column
assert trump['text'].loc[884740553040175104] == 'working hard to get the ol
```

Question 6d:

Now, let's convert the tweets into what's called a *tidy format* (https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html) to make the sentiments easier to calculate. Use the `no_punc` column of `trump` to create a table called `tidy_format`. The index of the table should be the IDs of the tweets, repeated once for every word in the tweet. It has two columns:

1. `num`: The location of the word in the tweet. For example, if the tweet was "i love america", then the location of the word "i" is 0, "love" is 1, and "america" is 2.
2. `word`: The individual words of each tweet.

The first few rows of our `tidy_format` table look like:

	num	word
894661651760377856	0	i

	num	word
894661651760377856	1	think
894661651760377856	2	senator
894661651760377856	3	blumenthal
894661651760377856	4	should

Note that you'll get different results depending on when you pulled in the tweets. However, you can double check that your tweet with ID `894661651760377856` has the same rows as ours. Our tests don't check whether your table looks exactly like ours.

This will require some rather advanced Pandas hacking, but our solution uses a chain of 5 methods on the `trump` DF.

- **Hint 1:** Try looking at the `expand` argument to pandas' `str.split`.
- **Hint 2:** Try looking at the `stack()` method.

In [43]:
```
tidy_format = trump["no_punc"].str.split(expand=True)
tidy_format = pd.DataFrame(tidy_format.stack())
tidy_format = tidy_format.reset_index()
tidy_format = tidy_format.rename(columns={tidy_format.columns[1]: "num", ti
tidy_format = tidy_format.set_index(tidy_format.columns[0])
del tidy_format.index.name
tidy_format
```

Out[43]:

	num	word
968621347294318592	0	senatorwicker
968621347294318592	1	of
968621347294318592	2	mississippi
968621347294318592	3	has
968621347294318592	4	been
968621347294318592	5	a
968621347294318592	6	great
968621347294318592	7	supporter
968621347294318592	8	and
968621347294318592	9	incredible
968621347294318592	10	help
968621347294318592	11	in
968621347294318592	12	getting
968621347294318592	13	our
968621347294318592	14	massive
968621347294318592	15	tax
968621347294318592	16	cut
968621347294318592	17	bill
968621347294318592	18	done
968621347294318592	19	and
968621347294318592	20	approved
968621347294318592	21	also
968621347294318592	22	big
968621347294318592	23	help
968621347294318592	24	on
968621347294318592	25	cutting
968621347294318592	26	regs
968621347294318592	27	i
968621347294318592	28	am

	num	word
968621347294318592	29	with
...	...	...
786310855843512320	14	t
786310855843512320	15	co
786310855843512320	16	hfihperfgz
786310855843512320	17	amp
786310855843512320	18	https
786310855843512320	19	t
786310855843512320	20	co
786310855843512320	21	pygilshrgv
786285509668696065	0	the
786285509668696065	1	people
786285509668696065	2	of
786285509668696065	3	cuba
786285509668696065	4	have
786285509668696065	5	struggled
786285509668696065	6	too
786285509668696065	7	long
786285509668696065	8	will
786285509668696065	9	reverse
786285509668696065	10	obama
786285509668696065	11	s
786285509668696065	12	executive
786285509668696065	13	orders
786285509668696065	14	and
786285509668696065	15	concessions
786285509668696065	16	towards
786285509668696065	17	cuba
786285509668696065	18	until
786285509668696065	19	freedoms
786285509668696065	20	are
786285509668696065	21	restored

145344 rows × 2 columns

In [44]:
```
assert tidy_format.loc[894661651760377856].shape == (27, 2)
assert ' '.join(list(tidy_format.loc[894661651760377856]['word'])) == 'i th
```

Question 6e:

Now that we have this table in the tidy format, it becomes much easier to find the sentiment of each tweet: we can join the table with the lexicon table.

Add a `polarity` column to the `trump` table. The `polarity` column should contain the sum of the sentiment polarity of each word in the text of the tweet.

Hint you will need to merge the `tidy_format` and `sent` tables and group the final answer.

In [45]:
```
trump.head(1)
```

Out[45]:

	time	source	retweet_count	text	est_time	hour	
968621347294318592	2018-02-27 22:58:19	Twitter for iPhone	6298	.@senatorwicker of mississippi has been a great supporter and incredible help in getting our massive tax cut bill done and approved. also big help on cutting regs. i am with him in his re-election all the way!	2018-02-27 17:58:19-05:00	17.971944	2018.15

In [46]:
```
merger = tidy_format.merge(sent, how="left", left_on="word", right_index=Tr
merger = merger.sort_index()
merger = merger.fillna(0)
merger = merger.groupby(merger.index)["polarity"].sum()
trump['polarity'] = merger
trump['polarity']
```

Out[46]:
```
968621347294318592     7.8
968620661349470209     1.8
968550893015707649     5.2
968549117625602054     9.0
968468176639004672    -1.5
968467243192537088    -0.2
968462966864609280    -1.3
968455547094753281     7.0
967847560248426498     4.8
967609114238050304     0.0
967597887545896961    -2.5
967597639293382657     0.0
967564998238142471     2.2
967563946063523840    -1.7
967545724362739712    -7.5
967539664692350977     2.8
967538684789739520    -6.8
967526110249537542     5.3
967508938425069568     8.9
967506350283599874    -7.4
967493467046899712     5.4
967472757025001472    11.1
967389553362423808     2.1
967388904952344577    -1.7
967168890232082433     7.5
967146956094083073     4.8
967084806726127616     7.0
967083810981597185     5.7
967083281974951939     4.9
967023714268319744     2.8
                       ...
787425145489072128    -2.0
787359730465329152    -1.8
787355131062943744     3.4
787320961934688257    -1.7
787320326573228032     3.0
787319977711923200     1.1
787267564405653505    -1.6
787266044213723138     1.2
787258211283918848    -4.4
787244543003467776    -2.5
787127225581707265     1.5
787099202291634177     4.3
787025537483046913    -1.0
787012170630455297     2.0
786950598826532864    -1.3
786737820669009921     1.1
786716631644897280     1.2
786710113163812864     1.2
786709861245526017     0.0
```

```
786700864752914433    -2.1
786691718062235649     7.7
786658827429154816     3.1
786589454991499264     0.0
786589172911964161     1.5
786565208663785476     1.2
786560925113266176    -4.2
786554517680693248     0.0
786340623804751872     1.2
786310855843512320     0.0
786285509668696065     1.2
Name: polarity, Length: 6802, dtype: float64
```

In [47]:
```
assert np.allclose(trump.loc[744701872456536064, 'polarity'], 8.4)
assert np.allclose(trump.loc[745304731346702336, 'polarity'], 2.5)
assert np.allclose(trump.loc[744519497764184064, 'polarity'], 1.7)
assert np.allclose(trump.loc[894661651760377856, 'polarity'], 0.2)
assert np.allclose(trump.loc[894620077634592769, 'polarity'], 5.4)
# If you fail this test, you dropped tweets with 0 polarity
assert np.allclose(trump.loc[744355251365511169, 'polarity'], 0.0)
```

Now we have a measure of the sentiment of each of his tweets! Note that this calculation is rather basic; you can read over the VADER readme to understand a more robust sentiment analysis.

Now, run the cells below to see the most positive and most negative tweets from Trump in your dataset:

In [48]:
```
print('Most negative tweets:')
for t in trump.sort_values('polarity').head()['text']:
    print('\n  ', t)
```

```
Most negative tweets:

   horrible and cowardly terrorist attack on innocent and defenseless wor
shipers in egypt. the world cannot tolerate terrorism, we must defeat the
m militarily and discredit the extremist ideology that forms the basis of
their existence!

   "@fiiibuster: @jeffzeleny pathetic - you have no sufficient evidence t
hat donald trump did not suffer from voter fraud, shame! bad reporter.

   democrat jon ossoff would be a disaster in congress. very weak on crim
e and illegal immigration, bad for jobs and wants higher taxes. say no

   nyc terrorist was happy as he asked to hang isis flag in his hospital
room. he killed 8 people, badly injured 12. should get death penalty!

   yet another terrorist attack today in israel -- a father, shot at by a
palestinian terrorist, was killed while:
https://t.co/cv1hzkvbit (https://t.co/cv1hzkvbit)
```

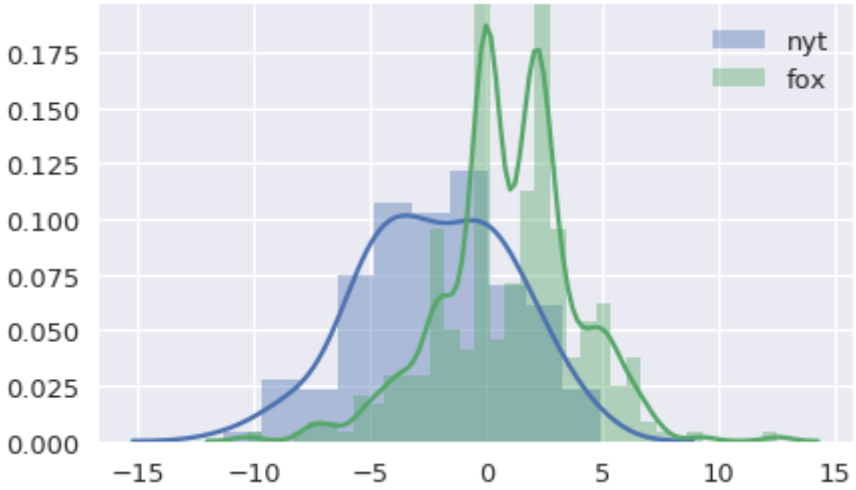
In [49]:
```
print('Most positive tweets:')
for t in trump.sort_values('polarity', ascending=False).head()['text']:
    print('\n  ', t)
```

```
Most positive tweets:

   thank you to linda bean of l.l.bean for your great support and courag
e. people will support you even more now. buy l.l.bean. @lbperfectmaine

   it was my great honor to celebrate the opening of two extraordinary mu
seums-the mississippi state history museum &amp;amp; the mississippi civil ri
ghts museum. we pay solemn tribute to our heroes of the past &amp;amp; dedica
te ourselves to building a future of freedom, equality, justice &amp;amp; pea
ce. https://t.co/5akgvpv8aa (https://t.co/5akgvpv8aa)

   rt @ivankatrump: 2016 has been one of the most eventful and exciting y
ears of my life. i wish you peace, joy, love and laughter. happy new…

   today, it was my great honor to sign a new executive order to ensure v
eterans have the resources they need as they transition back to civilian
 life. we must ensure that our heroes are given the care and support they
so richly deserve! https://t.co/0mdp9ddias (https://t.co/0mdp9ddias) http
s://t.co/lp2a8kcbap (https://t.co/lp2a8kcbap)

   it was my great honor to welcome mayor's from across america to the w
h. my administration will always support local government - and listen to
the leaders who know their communities best. together, we will usher in a
bold new era of peace and prosperity! https://t.co/dmyectnk0a (https://t.
co/dmyectnk0a) https://t.co/rsv7v7r0dt (https://t.co/rsv7v7r0dt)
```

Question 6g

Plot the distribution of tweet sentiments broken down by whether the text of the tweet contains `nyt` or `fox`. Then in the box below comment on what we observe?

In [50]:
```
nyt = [trump.loc[i, "polarity"] for i in trump.index if trump.loc[i, "text"]
fox = [trump.loc[i, "polarity"] for i in trump.index if trump.loc[i, "text"]
sns.distplot(nyt, label="nyt")
sns.distplot(fox, label="fox")
plt.legend();
```



Comment on what you observe:

Generally, Trump refers to "nyt" negatively, while he refers to "fox" positively. His tweets about fox seem to be a bit sporadic, as the values are more scattered throughout the plot. For "nyt", the distribution is more concentrated (on the negative side) and if there is positive sentiment, it is not too high.

Question 7: Engagement

Question 7a

Which of Trump's tweets had the most retweets? Were there certain words that often led to more retweets?

We can find this out by using our `tidy_format` DataFrame. For each word in the `tidy_format` DF, find out the number of retweets that its tweet got. Filter out words that didn't appear in at least 25 tweets, find out the median number of retweets each word got, and save the top 20 most retweeted words into a DataFrame called `top_20`. Your `top_20` table should have this format:

word	retweet_count
fake	22963.0
news	20463.0
ds100	20432.0
great	20159.0

	retweet_count
word	
class	20121.0

In [51]:
```
retweet = [trump.loc[i, "retweet_count"] for i in tidy_format.index]
tidy_format["retweet_count"] = retweet
words = tidy_format.groupby("word").count()
more_than_25_retweets = words[words["num"] > 25]
filtered = tidy_format[tidy_format["word"].isin(more_than_25_retweets.index
filtered = filtered.groupby("word")[["retweet_count"]].median()
filtered = filtered.sort_values("retweet_count", ascending=False)
top_20 = filtered[:20]
top_20
```

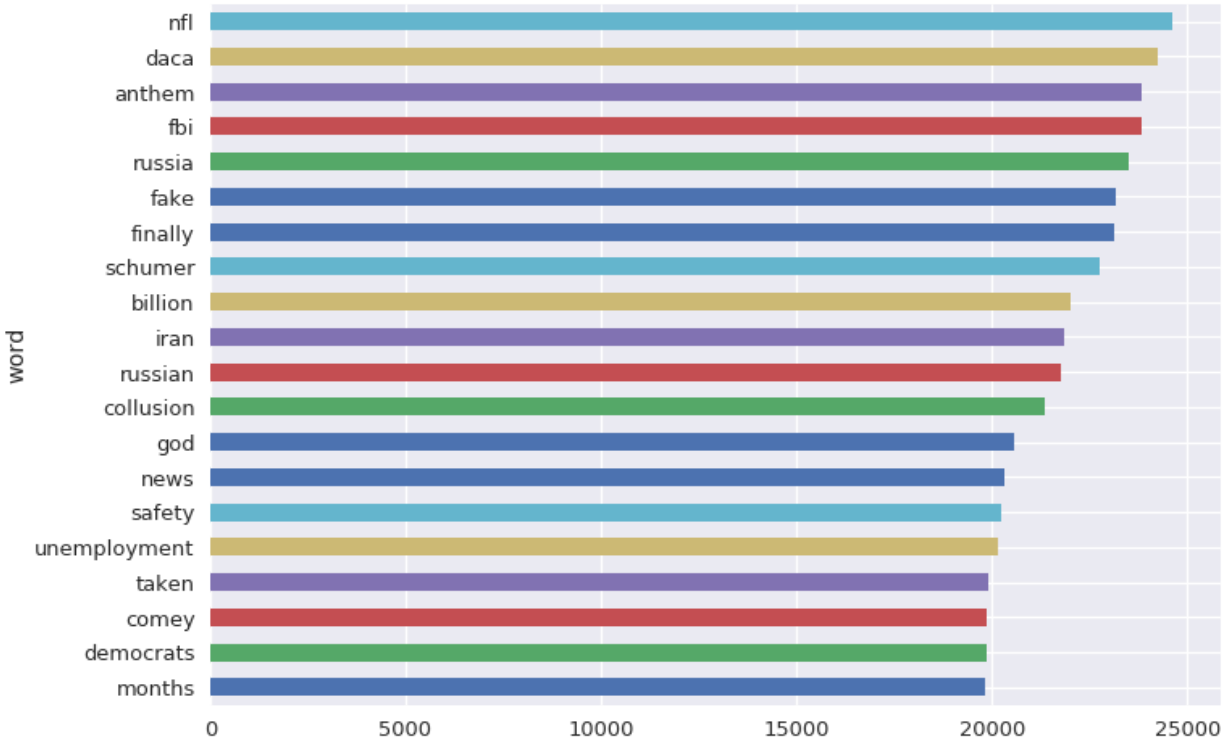
Out[51]:

word	retweet_count
nfl	24625.0
daca	24224.0
anthem	23820.0
fbi	23810.0
russia	23511.0
fake	23153.5
finally	23122.0
schumer	22746.0
billion	22007.0
iran	21855.0
russian	21744.0
collusion	21352.0
god	20581.0
news	20307.0
safety	20223.5
unemployment	20167.0
taken	19920.0
comey	19873.0
democrats	19850.5
months	19820.0

In [52]:
```
#### NOTE This Test is kind of iffy (very variable) - needs review before p

# Although it can't be guaranteed, it's very likely that the top 7 words wi
# in the top 20 words in the next month.
assert 'daca'     in top_20.index
assert 'nfl'      in top_20.index
assert 'anthem' in top_20.index
assert 'fbi'    in top_20.index
assert 'russia'    in top_20.index
```

Here's a bar chart of your results:

In [53]:
```
top_20['retweet_count'].sort_values().plot.barh(figsize=(10, 8));
```



Question 7b

The phrase "fake news" is apparently really popular! We can conclude that Trump's tweets containing "fake" and/or "news" result in the most retweets relative to words his other tweets. Or can we?

Consider each of the statements about possible confounding factors below. State whether each statement is true or false and explain. If the statement is true, state whether the confounding factor could have made "fake" and/or "news" higher on our list than they should be.

1. We didn't restrict our word list to nouns, so we have unhelpful words like "let" and "any" in our result.
2. We didn't remove hashtags in our text, so we have duplicate words (eg. #great and great).
3. We didn't account for the fact that Trump's follower count has increased over time.

1. True, but is not a confounding factor because these are neutral words and we are looking at their individual median, so their values won't have any affect on words like "fake" and their respective data.
2. False. There are not any duplicates.
3. True. If the retweets are increasing over time, there could be a disproportionate effect of on the words' median. The words used more recently would raise the median.

Question 8

Using the `trump` tweets construct an interesting plot describing a property of the data and discuss what you found below.

Ideas:

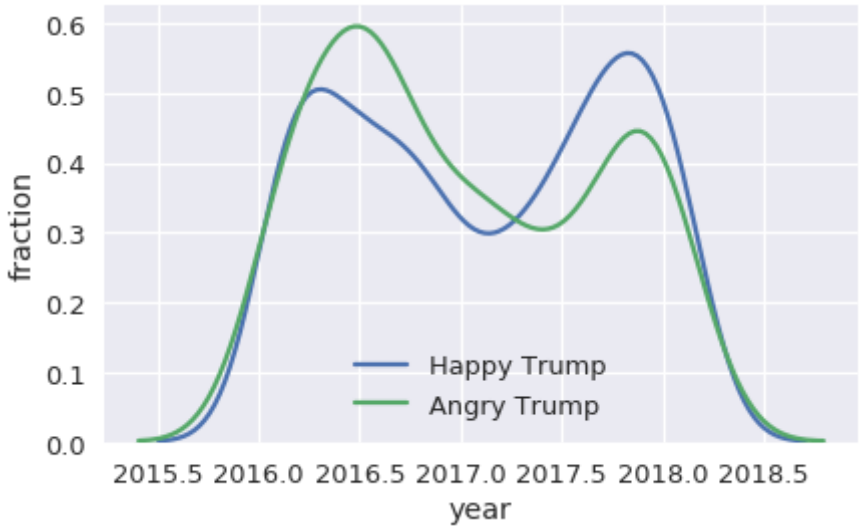
1. How has the sentiment changed with length of the tweets?
2. Does sentiment affect retweet count?
3. Are retweets more negative than regular tweets?
4. Are there any spikes in the number of retweets and do the correspond to world events?
5. *Bonus:* How many Russian twitter bots follow Trump?

You can look at other data sources and even tweets.

Plot:

In [54]:
```
happy = trump[trump["polarity"] >= 5]["year"]
angry = trump[trump["polarity"] <= -5]["year"]
sns.distplot(happy, label="Happy Trump", hist=False)
sns.distplot(angry, label="Angry Trump", hist=False)
plt.ylabel("fraction")
plt.legend()
```

Out[54]:
```
<matplotlib.legend.Legend at 0x7fefe6fa3630>
```



Discussion:

Trump seemed to be very angry during his presidential campaign, tweeting more tweets with a polarity below -5. Right after he took office, this trend shifts and Trump tweets more positively (tweets with a polarity greater than 5). We are analyzing his personal @realdonaldtrump account, but he also posts on the @potus account, which he got control of once he took office. It is possible that he posts more positive tweets on his personal account now, altough many other factors could have contributed to his rise in positivity.

Submission

Congrats, you just finished Project 1!