proj1

October 9, 2018

Before you turn this assignment in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All). Lastly, hit **Validate**.

If you worked locally, and then uploaded your work to the hub, make sure to follow these steps: - open your uploaded notebook **on the hub** - hit the validate button right above this cell, from inside the notebook

These steps should solve any issue related to submitting the notebook on the hub.

Make sure you fill in any place that says YOUR CODE HERE or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "Gilbert Antonius"
COLLABORATORS = ""
```

---

1 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 see full tweets
pd.set_option('max_colwidth', 280)

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

2 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 locally.
# The following should run:
import tweepy
```

There are instructions on using tweepy here, 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.

2.1 Question 1

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

1. Create a Twitter account. 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. Attach it to your Twitter account.
4. Once you're logged into your developer account, create an application for this assignment. 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 and replace the placeholders with your keys.

2.2 WARNING (Please Read) !!!!

2.2.1 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.

2.2.2 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.

2.2.3 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 and try again.")
            logging.warning(e)
```

```
Your username is: gilbertichwan
```

2.3 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 have.
            # We use tweet_mode='extended' so that Twitter gives us full 280 character tweets.
            # This was a change introduced in September 2017.

            # The tweepy Cursor API actually returns "sophisticated" Status objects but we
            # will use the basic Python dictionaries stored in the _json field.
            example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id="BerkeleyData",
                                                             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**:

```
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': 'Wed Feb 14 18:47:23 +0000 2018',
 'display_text_range': [0, 144],
 'entities': {'hashtags': [],
              'symbols': [],
              'urls': [],
              'user_mentions': [{'id': 263020833,
                                 'id_str': '263020833',
                                 'indices': [3, 19],
                                 'name': 'Berkeley School of Information',
                                 'screen_name': 'BerkeleyISchool'}]},
 'favorite_count': 0,
 'favorited': False,
```

```
'full_text': 'RT @BerkeleyISchool: We LOVE the I School community!  Our '
             'students, alumni, faculty &amp;amp; staff make the I School a '
             'incredible place to lear',
'geo': None,
'id': 963847153784369152,
'id_str': '963847153784369152',
'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,
'retweet_count': 3,
'retweeted': False,
'retweeted_status': {'contributors': None,
                     'coordinates': None,
                     'created_at': 'Wed Feb 14 18:47:00 +0000 2018',
                     'display_text_range': [0, 155],
                     'entities': {'hashtags': [{'indices': [140, 154],
                                                'text': 'ValentinesDay'}],
                                  'media': [{'display_url': 'pic.twitter.com/OpEShsJSUj',
                                             'expanded_url': 'https://twitter.com/BerkeleyISchool/status/963847057210421
                                             'id': 963843188598304768,
                                             'id_str': '963843188598304768',
                                             'indices': [156, 179],
                                             'media_url': 'http://pbs.twimg.com/media/DWBCRfuU8AArFBK.png',
                                             'media_url_https': 'https://pbs.twimg.com/media/DWBCRfuU8AArFBK.png
                                             'sizes': {'large': {'h': 1115,
                                                                 'resize': 'fit',
                                                                 'w': 1487},
                                                       'medium': {'h': 900,
                                                                  'resize': 'fit',
                                                                  'w': 1200},
                                                       'small': {'h': 510,
                                                                 'resize': 'fit',
                                                                 'w': 680},
                                                       'thumb': {'h': 150,
                                                                 'resize': 'crop',
                                                                 'w': 150}},
                                             'type': 'photo',
                                             'url': 'https://t.co/OpEShsJSUj'}],
                                  'symbols': [],
                                  'urls': [],
                                  'user_mentions': []},
                     'extended_entities': {'media': [{'display_url': 'pic.twitter.com/OpEShsJSUj',
                                                      'expanded_url': 'https://twitter.com/BerkeleyISchool/status/963847057
```

```
                                'id': 963843188598304768,
                                'id_str': '963843188598304768',
                                'indices': [156, 179],
                                'media_url': 'http://pbs.twimg.com/media/DWBCRfuU8AArFBK.png'
                                'media_url_https': 'https://pbs.twimg.com/media/DWBCRfuU8AArF
                                'sizes': {'large': {'h': 1115,
                                                    'resize': 'fit',
                                                    'w': 1487},
                                          'medium': {'h': 900,
                                                     'resize': 'fit',
                                                     'w': 1200},
                                          'small': {'h': 510,
                                                    'resize': 'fit',
                                                    'w': 680},
                                          'thumb': {'h': 150,
                                                    'resize': 'crop',
                                                    'w': 150}},
                                'type': 'photo',
                                'url': 'https://t.co/OpEShsJSUj'}]},
 'favorite_count': 10,
 'favorited': False,
 'full_text': 'We LOVE the I School community!  Our '
              'students, alumni, faculty &amp;amp; staff make '
              'the I School a incredible place to learn '
              'and grow. Happy #ValentinesDay! '
              'https://t.co/OpEShsJSUj',
 'geo': None,
 'id': 963847057210421248,
 'id_str': '963847057210421248',
 '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,
 'possibly_sensitive': False,
 'retweet_count': 3,
 'retweeted': False,
 'source': '<a '
           'href="https://about.twitter.com/products/tweetdeck" '
           'rel="nofollow">TweetDeck</a>',
 'truncated': False,
 'user': {'contributors_enabled': False,
          'created_at': 'Wed Mar 09 06:13:42 +0000 2011',
          'default_profile': False,
          'default_profile_image': False,
```

```
'description': 'The UC Berkeley School of '
               'Information is a '
               'multi-disciplinary program '
               'devoted to enhancing the '
               'accessibility, usability, '
               'credibility & security of '
               'information.',
'entities': {'description': {'urls': []},
             'url': {'urls': [{'display_url': 'ischool.berkeley.edu',
                               'expanded_url': 'http://ischool.berkeley.edu',
                               'indices': [0,
                                           23],
                               'url': 'https://t.co/5eXJ0wN1Jd'}]}},
'favourites_count': 2017,
'follow_request_sent': False,
'followers_count': 4510,
'following': False,
'friends_count': 561,
'geo_enabled': True,
'has_extended_profile': False,
'id': 263020833,
'id_str': '263020833',
'is_translation_enabled': False,
'is_translator': False,
'lang': 'en',
'listed_count': 205,
'location': 'Berkeley, California, USA',
'name': 'Berkeley School of Information',
'notifications': False,
'profile_background_color': '38628F',
'profile_background_image_url': 'http://pbs.twimg.com/profile_background_images/226
'profile_background_image_url_https': 'https://pbs.twimg.com/profile_background_ima
'profile_background_tile': False,
'profile_banner_url': 'https://pbs.twimg.com/profile_banners/263020833/1509549485',
'profile_image_url': 'http://pbs.twimg.com/profile_images/875733929902329856/K5Y9Y
'profile_image_url_https': 'https://pbs.twimg.com/profile_images/875733929902329856/
'profile_link_color': '3B7EA1',
'profile_sidebar_border_color': 'C0DEED',
'profile_sidebar_fill_color': 'DDEEF6',
'profile_text_color': '333333',
'profile_use_background_image': True,
'protected': False,
'screen_name': 'BerkeleyISchool',
'statuses_count': 3239,
'time_zone': 'Pacific Time (US & Canada)',
'translator_type': 'none',
'url': 'https://t.co/5eXJ0wN1Jd',
'utc_offset': -28800,
```

```
                          'verified': False}},
 'source': '<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>',
 'truncated': False,
 'user': {'contributors_enabled': False,
          'created_at': 'Thu Feb 28 14:37:26 +0000 2013',
          'default_profile': False,
          'default_profile_image': False,
          'description': 'An online Master of Information and Data Science '
                         '(MIDS) degree from the UC Berkeley School of '
                         'Information. Learn more at: http://t.co/zf6gfBWovQ',
          'entities': {'description': {'urls': [{'display_url': 'bit.ly/tBerkeleyData',
                                                 'expanded_url': 'http://bit.ly/tBerkeleyData',
                                                 'indices': [122, 144],
                                                 'url': 'http://t.co/zf6gfBWovQ'}]},
                       'url': {'urls': [{'display_url': 'datascience.berkeley.edu',
                                         'expanded_url': 'http://datascience.berkeley.edu',
                                         'indices': [0, 22],
                                         'url': 'http://t.co/S79Ul3oCaa'}]}},
          'favourites_count': 88,
          'follow_request_sent': False,
          'followers_count': 10949,
          'following': False,
          'friends_count': 409,
          'geo_enabled': False,
          'has_extended_profile': False,
          'id': 1227698863,
          'id_str': '1227698863',
          'is_translation_enabled': False,
          'is_translator': False,
          'lang': 'en',
          'listed_count': 474,
          'location': 'Berkeley, CA',
          'name': 'datascience@berkeley',
          'notifications': False,
          'profile_background_color': 'CCCCCC',
          'profile_background_image_url': 'http://pbs.twimg.com/profile_background_images/3788000009657121
          'profile_background_image_url_https': 'https://pbs.twimg.com/profile_background_images/378800000
          'profile_background_tile': False,
          'profile_banner_url': 'https://pbs.twimg.com/profile_banners/1227698863/1502212054',
          'profile_image_url': 'http://pbs.twimg.com/profile_images/894968224973897728/lI8iiF3J_normal.jpg',
          'profile_image_url_https': 'https://pbs.twimg.com/profile_images/894968224973897728/lI8iiF3J_norma
          'profile_link_color': '5173B6',
          'profile_sidebar_border_color': 'FFFFFF',
          'profile_sidebar_fill_color': 'DDEEF6',
          'profile_text_color': '333333',
          'profile_use_background_image': True,
          'protected': False,
          'screen_name': 'BerkeleyData',
```

```
'statuses_count': 2227,
'time_zone': 'Eastern Time (US & Canada)',
'translator_type': 'none',
'url': 'http://t.co/S79Ul3oCaa',
'utc_offset': -18000,
'verified': False}}
```

2.4 Question 2a

2.4.1 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."""
            import json
            with open(path) as f:
                auth_keys = json.load(f)
            return auth_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
    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:", user_account_name)
    except TweepError as e:
        logging.warning("There was a Tweepy error. Double check your API keys and try again.")
        logging.warning(e)

    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 have.
        # We use tweet_mode='extended' so that Twitter gives us full 280 character tweets.
        # This was a change introduced in September 2017.

        # The tweepy Cursor API actually returns "sophisticated" Status objects but we
        # will use the basic Python dictionaries stored in the _json field.
    example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id=user_account_name,
                                                     tweet_mode='extended').items()]
    return example_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"""
    # Saving the tweets to a json file on disk for future analysis
    with open(path, "w") as f:
        json.dump(tweets, f)
```

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."""

    # Re-loading the json file:
    with open(path, "r") as f:
        example_tweets = json.load(f)
    return example_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 available.

    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.
    """

    tweets_save_path = user_account_name + "_recent_tweets.json"
    if not Path(tweets_save_path).is_file():
```

```
        save_tweets(download_recent_tweets_by_user(user_account_name, load_keys(keys_path)), twee
    return load_tweets(tweets_save_path)
```

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: 3217
```

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

2.4.2 Question 2b

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

In [15]:
```
import re
import datetime

# Enter the number of the month of the oldest tweet (e.g. 1 for January)
temp = trump_tweets
oldest_month = temp[-1]['created_at']
oldest_month = pd.to_datetime(oldest_month)
oldest_month = oldest_month.month
oldest_month
```

Out[15]: 10

2.5 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.zip'
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: Tue Feb 20 04:12:59 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': 'CrookedHillary'}],
              'media': [{'display_url': 'pic.twitter.com/wjsl8ITVvk',
                         'expanded_url': 'https://twitter.com/realDonaldTrump/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/XqN
                         'media_url_https': 'https://pbs.twimg.com/ext_tw_video_thumb/786204885318561792/pu/i
                         'sizes': {'large': {'h': 576,
                                             'resize': 'fit',
                                             'w': 1024},
                                   'medium': {'h': 338,
                                              'resize': 'fit',
                                              'w': 600},
                                   'small': {'h': 191,
                                             'resize': 'fit',
                                             'w': 340},
                                   'thumb': {'h': 150,
                                             'resize': 'crop',
                                             'w': 150}},
                         'type': 'photo',
                         'url': 'https://t.co/wjsl8ITVvk'}],
              'symbols': [],
              'urls': [],
              'user_mentions': []},
 'extended_entities': {'media': [{'additional_media_info': {'monetizable': False},
                                  'display_url': 'pic.twitter.com/wjsl8ITVvk',
                                  'expanded_url': 'https://twitter.com/realDonaldTrump/status/786204978629185536/vi
                                  'id': 786204885318561792,
                                  'id_str': '786204885318561792',
```

```
                    'indices': [39, 62],
                    'media_url': 'http://pbs.twimg.com/ext_tw_video_thumb/786204885318561792/pu/in
                    'media_url_https': 'https://pbs.twimg.com/ext_tw_video_thumb/78620488531856179
                    'sizes': {'large': {'h': 576,
                                        'resize': 'fit',
                                        'w': 1024},
                              'medium': {'h': 338,
                                         'resize': 'fit',
                                         'w': 600},
                              'small': {'h': 191,
                                        'resize': 'fit',
                                        'w': 340},
                              'thumb': {'h': 150,
                                        'resize': 'crop',
                                        'w': 150}},
                    'type': 'video',
                    'url': 'https://t.co/wjsl8ITVvk',
                    'video_info': {'aspect_ratio': [16, 9],
                                   'duration_millis': 30106,
                                   'variants': [{'bitrate': 832000,
                                                 'content_type': 'video/mp4',
                                                 'url': 'https://video.twimg.com/ext_tw_video/7862048853185617
                                                {'bitrate': 2176000,
                                                 'content_type': 'video/mp4',
                                                 'url': 'https://video.twimg.com/ext_tw_video/7862048853185617
                                                {'bitrate': 320000,
                                                 'content_type': 'video/mp4',
                                                 'url': 'https://video.twimg.com/ext_tw_video/7862048853185617
                                                {'content_type': 'application/x-mpegURL',
                                                 'url': 'https://video.twimg.com/ext_tw_video/7862048853185617
'favorite_count': 42242,
'favorited': False,
'geo': None,
'id': 786204978629185536,
'id_str': '786204978629185536',
'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': {'attributes': {},
          'bounding_box': {'coordinates': [[[-87.634643, 24.396308],
                                            [-79.974307, 24.396308],
                                            [-79.974307, 31.001056],
                                            [-87.634643, 31.001056]]],
                           'type': 'Polygon'},
```

```
        'contained_within': [],
        'country': 'United States',
        'country_code': 'US',
        'full_name': 'Florida, USA',
        'id': '4ec01c9dbc693497',
        'name': 'Florida',
        'place_type': 'admin',
        'url': 'https://api.twitter.com/1.1/geo/id/4ec01c9dbc693497.json'},
'possibly_sensitive': False,
'retweet_count': 24915,
'retweeted': False,
'source': '<a href="http://twitter.com/download/iphone" '
        'rel="nofollow">Twitter for iPhone</a>',
'text': 'PAY TO PLAY POLITICS. \n#CrookedHillary https://t.co/wjsl8ITVvk',
'truncated': False,
'user': {'contributors_enabled': False,
        'created_at': 'Wed Mar 18 13:46:38 +0000 2009',
        'default_profile': False,
        'default_profile_image': False,
        'description': '45th President of the United States of America',
        'entities': {'description': {'urls': []}},
        'favourites_count': 12,
        'follow_request_sent': False,
        'followers_count': 35307313,
        'following': False,
        'friends_count': 45,
        'geo_enabled': True,
        'has_extended_profile': False,
        'id': 25073877,
        'id_str': '25073877',
        'is_translation_enabled': True,
        'is_translator': False,
        'lang': 'en',
        'listed_count': 74225,
        'location': 'Washington, DC',
        'name': 'Donald J. Trump',
        'notifications': False,
        'profile_background_color': '6D5C18',
        'profile_background_image_url': 'http://pbs.twimg.com/profile_background_images/530021613/trump_
        'profile_background_image_url_https': 'https://pbs.twimg.com/profile_background_images/530021613/
        'profile_background_tile': True,
        'profile_banner_url': 'https://pbs.twimg.com/profile_banners/25073877/1501916634',
        'profile_image_url': 'http://pbs.twimg.com/profile_images/874276197357596672/kUuht00m_normal.jpg'
        'profile_image_url_https': 'https://pbs.twimg.com/profile_images/874276197357596672/kUuht00m_nor
        'profile_link_color': '1B95E0',
        'profile_sidebar_border_color': 'BDDCAD',
        'profile_sidebar_fill_color': 'C5CEC0',
        'profile_text_color': '333333',
```

```
'profile_use_background_image': True,
'protected': False,
'screen_name': 'realDonaldTrump',
'statuses_count': 35480,
'time_zone': 'Eastern Time (US & Canada)',
'translator_type': 'regular',
'url': None,
'utc_offset': -14400,
'verified': True}}
```

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_repl
```

2.5.1 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 [20]: all_tweets = old_trump_tweets[:]

         i = 0
         while (i < len(trump_tweets)):
             count = 0
             j = 0
             while (j < len(old_trump_tweets)):
                 if (trump_tweets[i]['id'] != old_trump_tweets[j]['id']):
                     count += 1
                 if (count == len(old_trump_tweets)):
                     all_tweets.append(trump_tweets[i])
                 j += 1
             i += 1
```

```
In [21]: assert len(all_tweets) > len(trump_tweets)
         assert len(all_tweets) > len(old_trump_tweets)
```

2.5.2 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 [22]: trump = pd.DataFrame(all_tweets)[['created_at', 'source', 'text', 'full_text', 'retweet_count', 'id']]
         trump['created_at'] = pd.to_datetime(trump['created_at'])
         trump = trump.sort_values("id", ascending=True)
         trump = trump.set_index('id')
         trump.rename(columns={'created_at': 'time'}, inplace=True)
         #trump['text'].fillna(trump['full_text'])
         trump['text'].fillna(trump['full_text'], inplace=True)
         trump = trump.drop('full_text', 1)
```

```
In [23]: 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', 'retweet_count'])
         # If you fail these tests, you probably tried to use __dict__ or _json to read in the tweets
         assert np.sometrue([('Twitter for iPhone' in s) for s in trump['source'].unique()])
         assert trump['time'].dtype == np.dtype('<M8[ns]')
         assert trump['text'].dtype == np.dtype('O')
         assert trump['retweet_count'].dtype == np.dtype('int64')
```

2.6 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 [24]: trump['source'].unique()
```

```
Out[24]: array([ '<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>',
                 '<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>',
                 '<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>',
                 '<a href="https://mobile.twitter.com" rel="nofollow">Mobile Web (M5)</a>',
                 '<a href="http://instagram.com" rel="nofollow">Instagram</a>',
                 '<a href="http://twitter.com/#!/download/ipad" rel="nofollow">Twitter for iPad</a>',
                 '<a href="https://studio.twitter.com" rel="nofollow">Media Studio</a>',
                 '<a href="https://periscope.tv" rel="nofollow">Periscope</a>',
                 '<a href="https://ads.twitter.com" rel="nofollow">Twitter Ads</a>'], dtype=object)
```

2.7 Question 4a

Remove the HTML tags from the source field.

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

In [25]:
```
trump['source'] = trump['source'].str.replace("<[^>]*>", "")
trump['source']
```

Out[25]:

id	
690171032150237184	Twitter for Android
690171403388104704	Twitter for Android
690173226341691392	Twitter for Android
690176882055114758	Twitter for Android
690180284189310976	Twitter for Android
690271688127213568	Twitter for iPhone
690272687168458754	Twitter for Android
690313350278819840	Twitter for iPhone
690315202261155840	Twitter for iPhone
690315366564626433	Twitter for iPhone
690315667636023296	Twitter for iPhone
690336644281581568	Twitter for iPhone
690337376061788161	Twitter for iPhone
690382564494839809	Twitter for iPhone
690382619213742082	Twitter for iPhone
690382722162913280	Twitter for iPhone
690404308010057728	Twitter for iPhone
690528062190944256	Twitter for Android
690528407117889538	Twitter for Android
690528526181601281	Twitter for Android
690529122326413314	Twitter for Android
690529690205818880	Twitter for Android
690530164711624705	Twitter for Android
690532959363866625	Twitter for Android
690534215478173697	Twitter for Android
690534576066719744	Twitter for Android
690537121916923904	Twitter for Android
690540484154896384	Twitter for Android
690560125916975104	Twitter for Android
690560942430523392	Twitter for Android
...	...
964190995687591936	Media Studio
964218319783055360	Twitter for iPad
964219102683377665	Twitter for iPad
964219299211735040	Twitter for iPad
964232645495459840	Twitter for iPhone
964509154357411840	Twitter for iPhone
964512164865363968	Twitter for iPhone
964594780088033282	Twitter for iPhone
964724390637244417	Twitter for iPhone
964938678362628096	Twitter for iPhone
964944088696049666	Twitter for iPhone
964946611502747649	Twitter for iPhone
964949269374529538	Twitter for iPhone

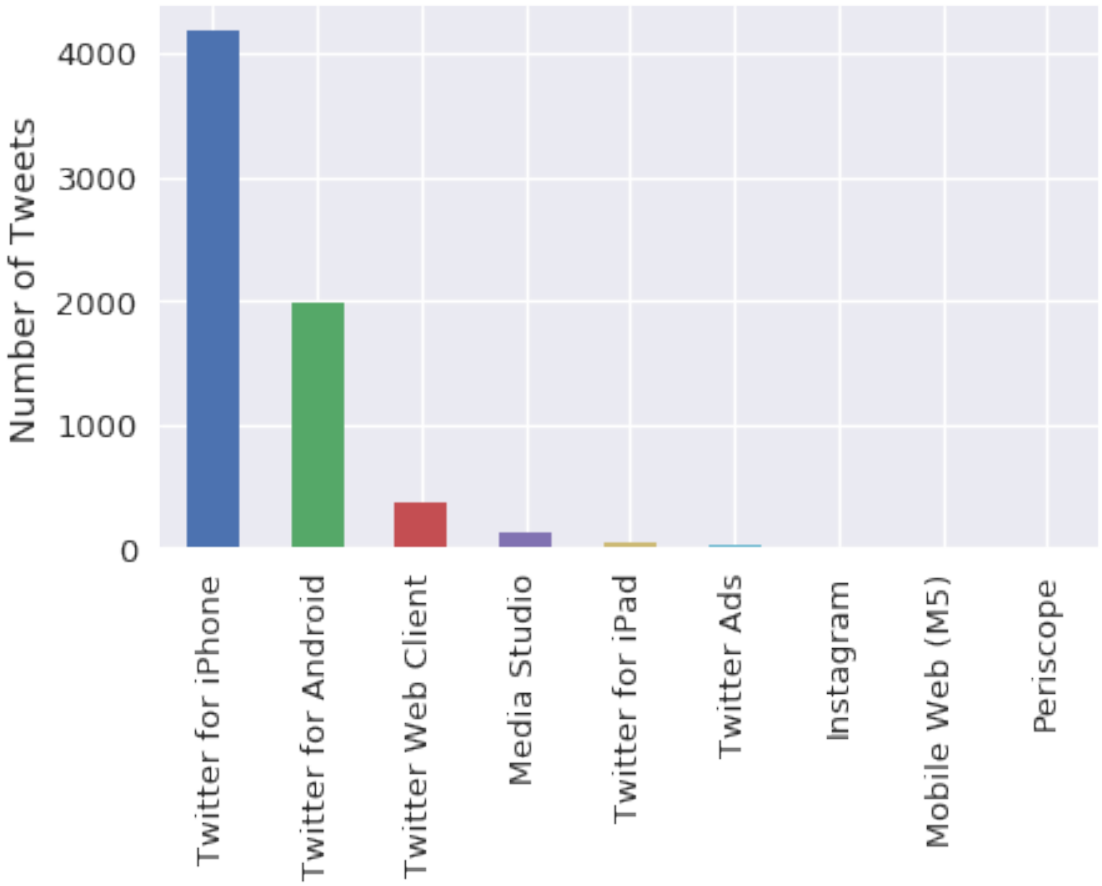
```
964955496137535488    Twitter for iPhone
964956781670694912    Twitter for iPhone
965009332042596352    Twitter for iPhone
965075589274177536    Twitter for iPhone
965079126829871104    Twitter for iPhone
965194903142719489    Twitter for iPhone
965199840471810049    Twitter for iPhone
965202556204003328    Twitter for iPhone
965205208191168512    Twitter for iPhone
965207569852780544    Twitter for iPhone
965212168449941505    Twitter for iPhone
965221024496279552    Twitter for iPhone
965223354633457665    Twitter for iPhone
965272331978407937    Twitter for iPhone
965303158229622785    Twitter for iPhone
965442990134251520    Twitter for iPhone
965582280772276224    Twitter for iPhone
Name: source, Length: 6737, dtype: object
```

In [26]:
```
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) ]['source'].unique())
 '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 [27]:
```
trump['source'].value_counts().plot(kind="bar")
plt.ylabel("Number of Tweets")
```

Out[27]:
```
Text(0,0.5,'Number of Tweets')
```



2.8 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 (notice the +0000 in the first few tweets)

```
In [28]: for t in trump_tweets[0:3]:
             print(t['created_at'])
```

```
Mon Feb 19 13:42:10 +0000 2018
Mon Feb 19 04:28:40 +0000 2018
Sun Feb 18 19:13:02 +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 [29]: 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[29]:

id	time	source	\
690171032150237184	2016-01-21 13:56:11	Twitter for Android	
690171403388104704	2016-01-21 13:57:39	Twitter for Android	
690173226341691392	2016-01-21 14:04:54	Twitter for Android	
690176882055114758	2016-01-21 14:19:26	Twitter for Android	
690180284189310976	2016-01-21 14:32:57	Twitter for Android	

id	
690171032150237184	"@bigop1: @realDonaldTrum
690171403388104704	"@AmericanAsPie: @glennbeck @SarahPalinUSA Remember when Glenn g
690173226341691392	So sad that @CNN and many others refused to show the massive crowd
690176882055114758	Sad sack @JebBush has just done another ad on me, with special interest money, s
690180284189310976	Low energy candidate @JebBush has wasted $80 million on his failed presidential ca

id	retweet_count	est_time
690171032150237184	1059	2016-01-21 08:56:11-05:00
690171403388104704	1339	2016-01-21 08:57:39-05:00
690173226341691392	2006	2016-01-21 09:04:54-05:00
690176882055114758	2266	2016-01-21 09:19:26-05:00
690180284189310976	2886	2016-01-21 09:32:57-05:00

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 [30]: trump['hour'] = trump['est_time'].apply(lambda x: (x.hour + (x.minute / 60) + (x.second / (60**2))))
```

```
In [31]: assert np.isclose(trump.loc[690171032150237184]['hour'], 8.93639)
```

2.9 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 [32]: trump
```

Out[32]:

id	time	source
690171032150237184	2016-01-21 13:56:11	Twitter for Android
690171403388104704	2016-01-21 13:57:39	Twitter for Android
690173226341691392	2016-01-21 14:04:54	Twitter for Android
690176882055114758	2016-01-21 14:19:26	Twitter for Android
690180284189310976	2016-01-21 14:32:57	Twitter for Android
690271688127213568	2016-01-21 20:36:09	Twitter for iPhone
690272687168458754	2016-01-21 20:40:07	Twitter for Android
690313350278819840	2016-01-21 23:21:42	Twitter for iPhone
690315202261155840	2016-01-21 23:29:04	Twitter for iPhone
690315366564626433	2016-01-21 23:29:43	Twitter for iPhone
690315667636023296	2016-01-21 23:30:55	Twitter for iPhone
690336644281581568	2016-01-22 00:54:16	Twitter for iPhone
690337376061788161	2016-01-22 00:57:10	Twitter for iPhone
690382564494839809	2016-01-22 03:56:44	Twitter for iPhone
690382619213742082	2016-01-22 03:56:57	Twitter for iPhone
690382722162913280	2016-01-22 03:57:22	Twitter for iPhone
690404308010057728	2016-01-22 05:23:08	Twitter for iPhone
690528062190944256	2016-01-22 13:34:54	Twitter for Android
690528407117889538	2016-01-22 13:36:16	Twitter for Android
690528526181601281	2016-01-22 13:36:44	Twitter for Android
690529122326413314	2016-01-22 13:39:06	Twitter for Android
690529690205818880	2016-01-22 13:41:22	Twitter for Android
690530164711624705	2016-01-22 13:43:15	Twitter for Android
690532959363866625	2016-01-22 13:54:21	Twitter for Android
690534215478173697	2016-01-22 13:59:21	Twitter for Android
690534576066719744	2016-01-22 14:00:47	Twitter for Android
690537121916923904	2016-01-22 14:10:54	Twitter for Android
690540484154896384	2016-01-22 14:24:15	Twitter for Android
690560125916975104	2016-01-22 15:42:18	Twitter for Android
690560942430523392	2016-01-22 15:45:33	Twitter for Android
...	...	...
964190995687591936	2018-02-15 17:33:41	Media Studio
964218319783055360	2018-02-15 19:22:16	Twitter for iPad
964219102683377665	2018-02-15 19:25:23	Twitter for iPad
964219299211735040	2018-02-15 19:26:09	Twitter for iPad
964232645495459840	2018-02-15 20:19:11	Twitter for iPhone
964509154357411840	2018-02-16 14:37:56	Twitter for iPhone
964512164865363968	2018-02-16 14:49:54	Twitter for iPhone
964594780088033282	2018-02-16 20:18:11	Twitter for iPhone
964724390637244417	2018-02-17 04:53:13	Twitter for iPhone
964938678362628096	2018-02-17 19:04:43	Twitter for iPhone
964944088696049666	2018-02-17 19:26:13	Twitter for iPhone
964946611502747649	2018-02-17 19:36:14	Twitter for iPhone
964949269374529538	2018-02-17 19:46:48	Twitter for iPhone
964955496137535488	2018-02-17 20:11:32	Twitter for iPhone
964956781670694912	2018-02-17 20:16:39	Twitter for iPhone

965009332042596352	2018-02-17 23:45:28	Twitter for iPhone
965075589274177536	2018-02-18 04:08:45	Twitter for iPhone
965079126829871104	2018-02-18 04:22:48	Twitter for iPhone
965194903142719489	2018-02-18 12:02:52	Twitter for iPhone
965199840471810049	2018-02-18 12:22:29	Twitter for iPhone
965202556204003328	2018-02-18 12:33:16	Twitter for iPhone
965205208191168512	2018-02-18 12:43:48	Twitter for iPhone
965207569852780544	2018-02-18 12:53:11	Twitter for iPhone
965212168449941505	2018-02-18 13:11:28	Twitter for iPhone
965221024496279552	2018-02-18 13:46:39	Twitter for iPhone
965223354633457665	2018-02-18 13:55:55	Twitter for iPhone
965272331978407937	2018-02-18 17:10:32	Twitter for iPhone
965303158229622785	2018-02-18 19:13:02	Twitter for iPhone
965442990134251520	2018-02-19 04:28:40	Twitter for iPhone
965582280772276224	2018-02-19 13:42:10	Twitter for iPhone

id
690171032150237184
690171403388104704
690173226341691392
690176882055114758
690180284189310976
690271688127213568
690272687168458754
690313350278819840
690315202261155840
690315366564626433
690315667636023296
690336644281581568
690337376061788161
690382564494839809
690382619213742082
690382722162913280
690404308010057728
690528062190944256
690528407117889538
690528526181601281
690529122326413314
690529690205818880
690530164711624705
690532959363866625
690534215478173697
690534576066719744
690537121916923904
690540484154896384
690560125916975104
690560942430523392

...

964190995687591936	In times of tragedy, the bonds that sustain us are those of family, faith, community
964218319783055360	
964219102683377665	The Schumer-Rounds-Collins immigration bill would be a total catastrophe. @DHSg
964219299211735040	...lottery, continues deadly catch-and-release, and bars enforcement even for FUTUI
964232645495459840	
964509154357411840	I will be leav
964512164865363968	
964594780088033282	Russia started their anti-U
964724390637244417	Our entire Nation, w/one heavy heart, continues to pray for the victims & their
964938678362628096	
964944088696049666	Charges Deal Don A Big Win, written by Michael Goodwin of the @nypost, succi
964946611502747649	Deputy A.G. Rod Rosenstein stated at the News Conference: There is no allegation
964949269374529538	Funny how the
964955496137535488	
964956781670694912	I have seen all c
965009332042596352	Just like they dont want to solve the DACA proble
965075589274177536	Very sad that the FBI missed all of the many signals sent out by the Florida schoo
965079126829871104	General McMaster forgot to say that the results of the 2016 election were not impa
965194903142719489	
965199840471810049	Finally, Liddle Adam Schiff, the leakin monster of no control, is now
965202556204003328	I never said Russia did not meddle in the election, I said it may be Russia, or China
965205208191168512	Now that Adam Schiff is starting to blame President Obama for Russian meddli
965207569852780544	
965212168449941505	If it was the GOAL of Russia to create discord, disruption and chaos within the
965221024496279552	
965223354633457665	Great Pollster John McLaughlin now has the GOP up in the G
965272331978407937	Thank y
965303158229622785	
965442990134251520	Just watched a very insecure Oprah Winfrey, who at one point I knew
965582280772276224	

id	retweet_count	est_time	hour
690171032150237184	1059	2016-01-21 08:56:11-05:00	8.936389
690171403388104704	1339	2016-01-21 08:57:39-05:00	8.960833
690173226341691392	2006	2016-01-21 09:04:54-05:00	9.081667
690176882055114758	2266	2016-01-21 09:19:26-05:00	9.323889
690180284189310976	2886	2016-01-21 09:32:57-05:00	9.549167
690271688127213568	1429	2016-01-21 15:36:09-05:00	15.602500
690272687168458754	1053	2016-01-21 15:40:07-05:00	15.668611
690313350278819840	2329	2016-01-21 18:21:42-05:00	18.361667
690315202261155840	1463	2016-01-21 18:29:04-05:00	18.484444
690315366564626433	1761	2016-01-21 18:29:43-05:00	18.495278
690315667636023296	2217	2016-01-21 18:30:55-05:00	18.515278
690336644281581568	1576	2016-01-21 19:54:16-05:00	19.904444
690337376061788161	2422	2016-01-21 19:57:10-05:00	19.952778
690382564494839809	2187	2016-01-21 22:56:44-05:00	22.945556

690382619213742082	1817	2016-01-21 22:56:57-05:00	22.949167
690382722162913280	2236	2016-01-21 22:57:22-05:00	22.956111
690404308010057728	9144	2016-01-22 00:23:08-05:00	0.385556
690528062190944256	1595	2016-01-22 08:34:54-05:00	8.581667
690528407117889538	909	2016-01-22 08:36:16-05:00	8.604444
690528526181601281	676	2016-01-22 08:36:44-05:00	8.612222
690529122326413314	773	2016-01-22 08:39:06-05:00	8.651667
690529690205818880	753	2016-01-22 08:41:22-05:00	8.689444
690530164711624705	637	2016-01-22 08:43:15-05:00	8.720833
690532959363866625	1937	2016-01-22 08:54:21-05:00	8.905833
690534215478173697	875	2016-01-22 08:59:21-05:00	8.989167
690534576066719744	799	2016-01-22 09:00:47-05:00	9.013056
690537121916923904	1738	2016-01-22 09:10:54-05:00	9.181667
690540484154896384	2116	2016-01-22 09:24:15-05:00	9.404167
690560125916975104	1661	2016-01-22 10:42:18-05:00	10.705000
690560942430523392	1110	2016-01-22 10:45:33-05:00	10.759167
...	...	...	...
964190995687591936	19482	2018-02-15 12:33:41-05:00	12.561389
964218319783055360	7970	2018-02-15 14:22:16-05:00	14.371111
964219102683377665	20028	2018-02-15 14:25:23-05:00	14.423056
964219299211735040	15186	2018-02-15 14:26:09-05:00	14.435833
964232645495459840	13514	2018-02-15 15:19:11-05:00	15.319722
964509154357411840	19897	2018-02-16 09:37:56-05:00	9.632222
964512164865363968	21720	2018-02-16 09:49:54-05:00	9.831667
964594780088033282	37340	2018-02-16 15:18:11-05:00	15.303056
964724390637244417	24271	2018-02-16 23:53:13-05:00	23.886944
964938678362628096	14274	2018-02-17 14:04:43-05:00	14.078611
964944088696049666	18635	2018-02-17 14:26:13-05:00	14.436944
964946611502747649	21483	2018-02-17 14:36:14-05:00	14.603889
964949269374529538	25200	2018-02-17 14:46:48-05:00	14.780000
964955496137535488	18180	2018-02-17 15:11:32-05:00	15.192222
964956781670694912	16464	2018-02-17 15:16:39-05:00	15.277500
965009332042596352	31500	2018-02-17 18:45:28-05:00	18.757778
965075589274177536	35826	2018-02-17 23:08:45-05:00	23.145833
965079126829871104	26168	2018-02-17 23:22:48-05:00	23.380000
965194903142719489	35247	2018-02-18 07:02:52-05:00	7.047778
965199840471810049	26929	2018-02-18 07:22:29-05:00	7.374722
965202556204003328	29636	2018-02-18 07:33:16-05:00	7.554444
965205208191168512	18633	2018-02-18 07:43:48-05:00	7.730000
965207569852780544	11640	2018-02-18 07:53:11-05:00	7.886389
965212168449941505	31259	2018-02-18 08:11:28-05:00	8.191111
965221024496279552	19078	2018-02-18 08:46:39-05:00	8.777500
965223354633457665	20684	2018-02-18 08:55:55-05:00	8.931944
965272331978407937	16444	2018-02-18 12:10:32-05:00	12.175556
965303158229622785	14405	2018-02-18 14:13:02-05:00	14.217222
965442990134251520	28286	2018-02-18 23:28:40-05:00	23.477778
965582280772276224	16832	2018-02-19 08:42:10-05:00	8.702778

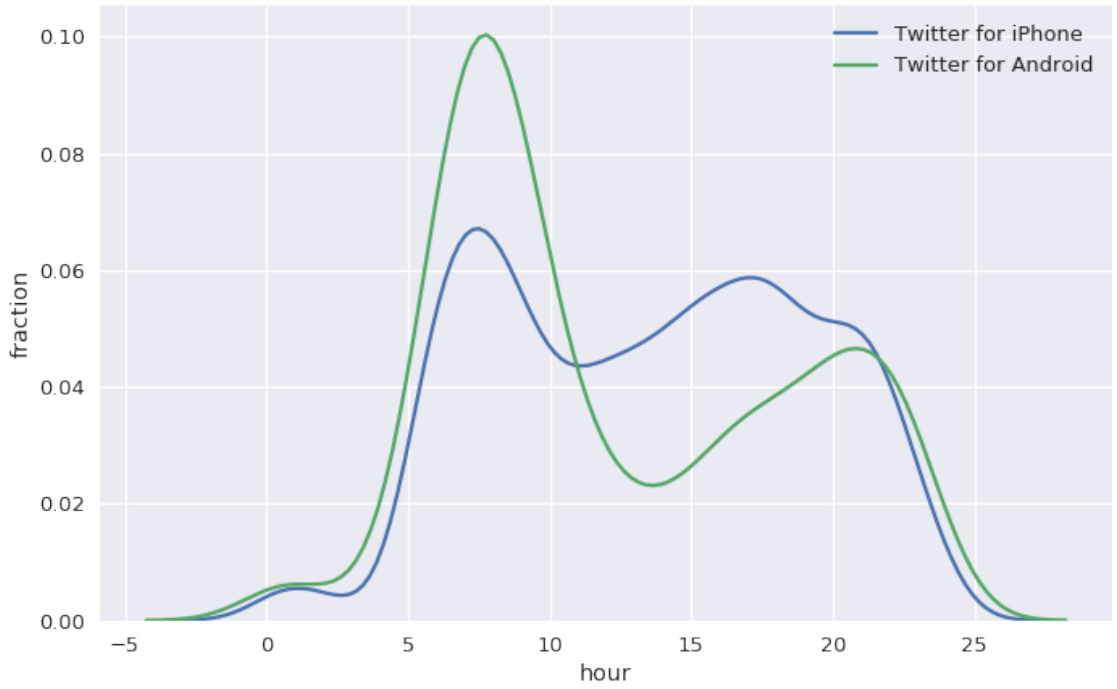
[6737 rows x 6 columns]

In [33]:
```
trump_android = trump[['source', 'hour']].loc[trump['source'] == 'Twitter for Android', :]
trump_android = trump_android['hour']
trump_iphone = trump[['source', 'hour']].loc[trump['source'] == 'Twitter for iPhone', :]
trump_iphone = trump_iphone['hour']


plt.figure(figsize=(10, 7))
sns.distplot(trump_iphone, hist=False, kde_kws={"label": "Twitter for iPhone"})
sns.distplot(trump_android, hist=False, kde_kws={"label": "Twitter for Android"})

plt.ylabel('fraction')
```

Out[33]: Text(0,0.5,'fraction')



2.10 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 two curves are similar; they both have peaks at around the seventh and eight hours as well as another peak ar around 17th and 21st hours. If someone told me that Trump tends to tweet early in the morning and then later in the evening, I would tend to answer that the Android device might more strongly suppport this claim and it is Trump's device. But, the iPhone also have a similar pattern; thus, Trump might also use the iPhone.

2.11 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)

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)
```

2.11.1 Question 5a

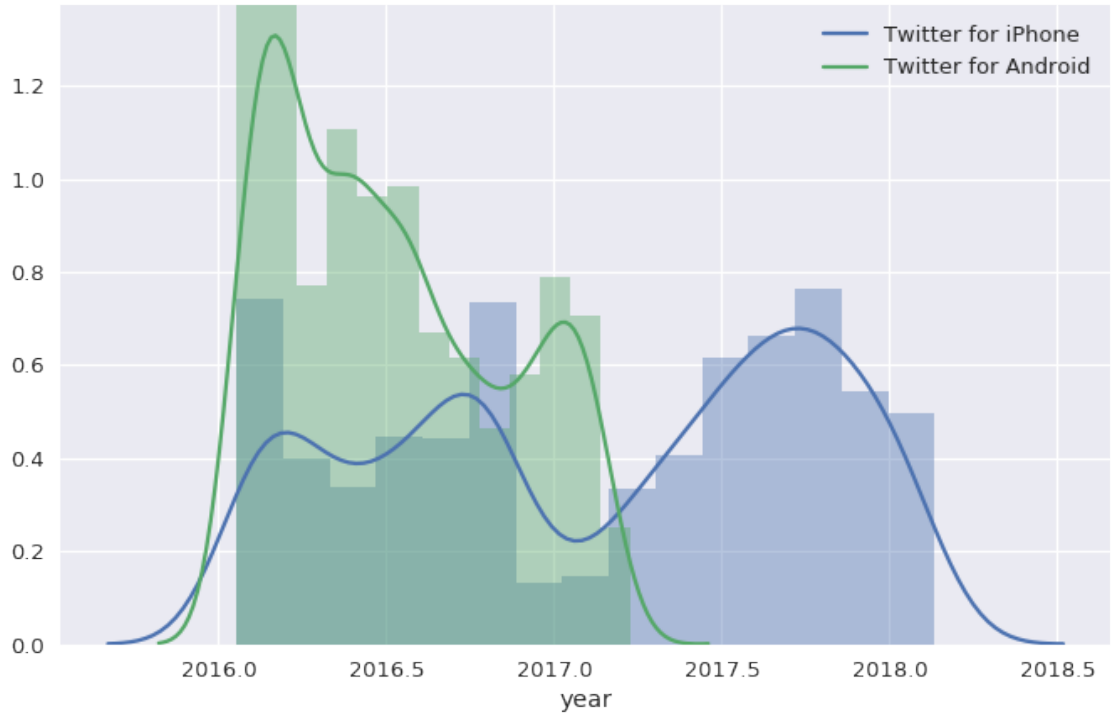
Use the sns.distplot to overlay the distributions of the 2 most frequently used web technologies over the years. Your final plot should look like:

In [35]:
```
trump_android_yr = trump.loc[trump['source'] == 'Twitter for Android', :]
trump_android_yr = trump_android_yr['year']

trump_iphone_yr = trump.loc[trump['source'] == 'Twitter for iPhone', :]
trump_iphone_yr = trump_iphone_yr['year']

plt.figure(figsize=(10, 7))
sns.distplot(trump_iphone_yr, kde_kws={"label": "Twitter for iPhone"})
sns.distplot(trump_android_yr, kde_kws={"label": "Twitter for Android"})
```

Out[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d98bbdb38>
```



2.11.2 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.

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?

The data supports the information in the article since the graph shows that the Tweets sent after his inauguration on January 2017 are sent from the iPhone instead of Android. Another thing that can be observed is that there are tweets sent from Trump's account that are sent from the iPhone before the inauguration. This means that Trump might use two devices (iPhone and Android) to tweet or the iPhone tweets are not made by Trump himself, but by someone else on his behalf.

2.12 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) 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]
```

2.13 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]: open("vader_lexicon.txt").readlines()[:10]
```

```
Out[37]: ['$:\t-1.5\t0.80623\t[-1, -1, -1, -1, -3, -1, -3, -1, -2, -1]\n',
          '%)\t-0.4\t1.0198\t[-1, 0, -1, 0, 0, -2, -1, 2, -1, 0]\n',
          '%-)\t-1.5\t1.43178\t[-2, 0, -2, -2, -1, 2, -2, -3, -2, -3]\n',
          '&-:\t-0.4\t1.42829\t[-3, -1, 0, 0, -1, -1, -1, 2, -1, 2]\n',
          '&:\t-0.7\t0.64031\t[0, -1, -1, -1, 1, -1, -1, -1, -1, -1]\n',
          "( '}{' )\t1.6\t0.66332\t[1, 2, 2, 1, 1, 2, 2, 1, 3, 1]\n",
          '(%\t-0.9\t0.9434\t[0, 0, 1, -1, -1, -1, -2, -2, -1, -2]\n',
          "('-:\t2.2\t1.16619\t[4, 1, 4, 3, 1, 2, 3, 1, 2, 1]\n",
          "(':\t2.3\t0.9\t[1, 3, 3, 2, 2, 4, 2, 3, 1, 2]\n",
          '((-:\t2.1\t0.53852\t[2, 2, 2, 1, 2, 3, 2, 2, 3, 2]\n']
```

```
In [38]: sent = pd.read_table("vader_lexicon.txt", header=None, usecols=[0,1])
         #sent.reindex(sent[0])

         sent.rename({0: 'index', 1: 'polarity'}, axis='columns', inplace=True)
         sent.set_index(sent['index'], inplace=True)
         sent.drop('index', axis=1, inplace=True)
```

```
In [39]: assert isinstance(sent, pd.DataFrame)
         assert sent.shape == (7517, 1)
         assert list(sent.index[5000:5005]) == ['paranoids', 'pardon', 'pardoned', 'pardoning', 'pardons']
         assert np.allclose(sent['polarity'].head(), [-1.5, -0.4, -1.5, -0.4, -0.7])
```

2.14 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 [40]: trump['text'] = trump['text'].apply(str.lower)
```

```
In [41]: assert trump['text'].loc[884740553040175104] == 'working hard to get the olympics for the united states (
```

2.15 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 [42]: # Save your regex in punct_re
         punct_re = r'[^\w\s]'
         trump['no_punc'] = trump['text'].str.replace(punct_re, ' ')
```

```
In [43]: 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  nbcsnl saturday night live last n
         assert trump['no_punc'].loc[894620077634592769] == 'on  purpleheartday i thank all the brave men and w
         # If you fail these tests, you accidentally changed the text column
         assert trump['text'].loc[884740553040175104] == 'working hard to get the olympics for the united states (l
```

2.16 Question 6d:

Now, let's convert the tweets into what's called a *tidy format* 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:

<tr style="text-align: right;">
  <th></th>
  <th>num</th>
  <th>word</th>
</tr>

<tr>
  <th>894661651760377856</th>
  <td>0</td>
  <td>i</td>
</tr>
<tr>
  <th>894661651760377856</th>
  <td>1</td>
  <td>think</td>
</tr>
<tr>
  <th>894661651760377856</th>
  <td>2</td>
  <td>senator</td>
</tr>
<tr>
  <th>894661651760377856</th>
  <td>3</td>
  <td>blumenthal</td>
</tr>
<tr>
  <th>894661651760377856</th>
  <td>4</td>
  <td>should</td>
</tr>

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 [44]: tidy_format = trump.loc[:, ['no_punc']]
         tidy_format = tidy_format['no_punc'].str.split(expand=True)
         tidy_format = tidy_format.stack().to_frame()
```

```
tidy_format = tidy_format.reset_index()
tidy_format = tidy_format.rename({'level_1': 'num', 0: 'word'}, axis='columns')
tidy_format.set_index(['id'], inplace=True)

tidy_format
```

Out[44]:

id	num	word
690171032150237184	0	bigop1
690171032150237184	1	realdonaldtrump
690171032150237184	2	sarahpalinusa
690171032150237184	3	https
690171032150237184	4	t
690171032150237184	5	co
690171032150237184	6	3kyqgqevyd
690171403388104704	0	americanaspie
690171403388104704	1	glennbeck
690171403388104704	2	sarahpalinusa
690171403388104704	3	remember
690171403388104704	4	when
690171403388104704	5	glenn
690171403388104704	6	gave
690171403388104704	7	out
690171403388104704	8	gifts
690171403388104704	9	to
690171403388104704	10	illegal
690171403388104704	11	aliens
690171403388104704	12	at
690171403388104704	13	crossing
690171403388104704	14	the
690171403388104704	15	border
690171403388104704	16	me
690171403388104704	17	too
690173226341691392	0	so
690173226341691392	1	sad
690173226341691392	2	that
690173226341691392	3	cnn
690173226341691392	4	and
...	...	...
965442990134251520	27	and
965442990134251520	28	slanted
965442990134251520	29	the
965442990134251520	30	facts
965442990134251520	31	incorrect
965442990134251520	32	hope
965442990134251520	33	oprah
965442990134251520	34	runs
965442990134251520	35	so

```
965442990134251520   36          she
965442990134251520   37          can
965442990134251520   38           be
965442990134251520   39      exposed
965442990134251520   40          and
965442990134251520   41     defeated
965442990134251520   42         just
965442990134251520   43         like
965442990134251520   44          all
965442990134251520   45           of
965442990134251520   46          the
965442990134251520   47       others
965582280772276224    0         have
965582280772276224    1            a
965582280772276224    2        great
965582280772276224    3          but
965582280772276224    4         very
965582280772276224    5   reflective
965582280772276224    6    president
965582280772276224    7            s
965582280772276224    8          day

[142477 rows x 2 columns]
```

```
In [45]: assert tidy_format.loc[894661651760377856].shape == (27, 2)
         assert ' '.join(list(tidy_format.loc[894661651760377856]['word'])) == 'i think senator blumenthal should ta
```

2.17 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 [46]: temp = tidy_format.join(sent, 'word')
         temp = temp.groupby(temp.index.values).agg(np.nansum)
         temp = temp.loc[:, ['polarity']]
         trump['polarity'] = temp
```

```
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 worshipers in egypt. the world cannot tolerat

   democrat jon ossoff would be a disaster in congress. very weak on crime and illegal immigration, bad for jobs a

   "@fiiibuster: @jeffzeleny pathetic - you have no sufficient evidence that donald trump did not suffer from voter

   nyc terrorist was happy as he asked to hang isis flag in his hospital room. he killed 8 people, badly injured 12. s

   yet another terrorist attack today in israel -- a father, shot at by a palestinian terrorist, was killed while:
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 courage. people will support you even more now.

   it was my great honor to celebrate the opening of two extraordinary museums-the mississippi state history mus

   rt @ivankatrump: 2016 has been one of the most eventful and exciting years of my life. i wish you peace, joy, lo

   today, it was my great honor to sign a new executive order to ensure veterans have the resources they need as t

   it was my great honor to welcome mayors from across america to the wh. my administration will always suppor
```

2.18 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?

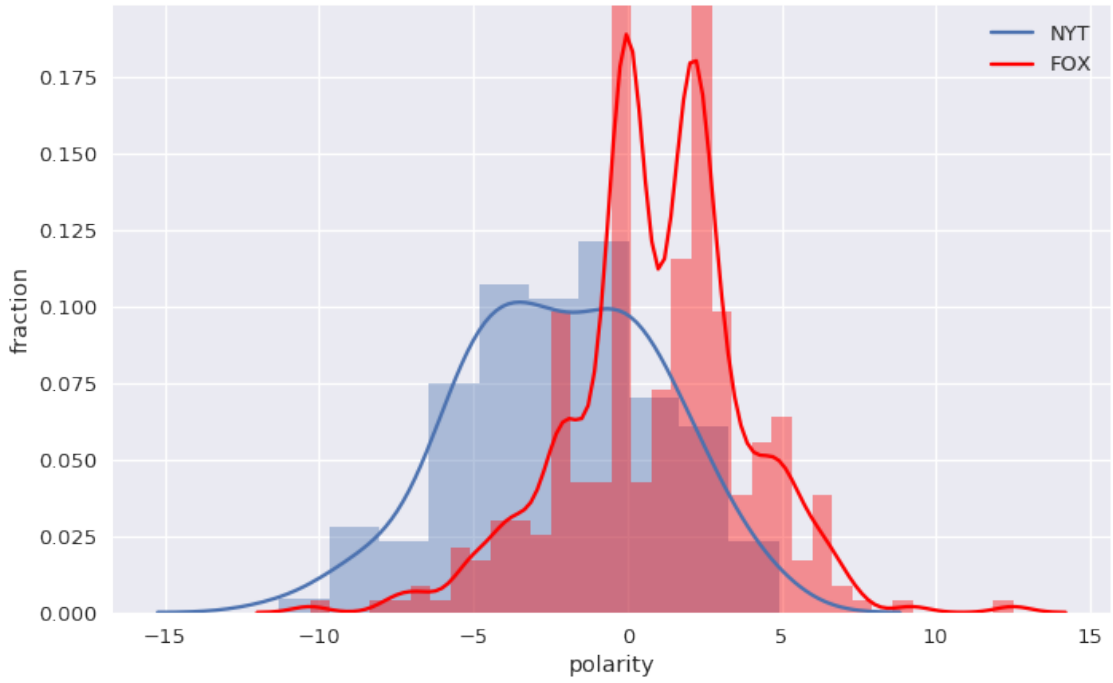
```
In [50]: lst_nyt = []
         lst_fox = []
```

```
for i in trump.index.values:
    if 'nyt' in trump['text'][i]:
        lst_nyt.append(trump['polarity'][i])
    if 'fox' in trump['text'][i]:
        lst_fox.append(trump['polarity'][i])

plt.figure(figsize=(10, 7))
sns.distplot(lst_nyt, kde_kws={"label": "NYT"})
sns.distplot(lst_fox, color='r', kde_kws={"label": "FOX"})

plt.ylabel('fraction')
plt.xlabel('polarity')
```

Out[50]: Text(0.5,0,'polarity')



Comment on what you observe: It can be seen that the distribution of tweets containing the word "nyt" centers on the negative value while the distribution of tweets containing the word "fox" centers at around 0 and 2.5 on the polarity scale. These two curves represent Trump's sentiment toward each of the two media companies; Trump favors Fox more than NYT, he even dislikes the NYT.

2.19 Question 7: Engagement

2.20 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:

<tr style="text-align: right;">
  <th></th>
  <th>retweet_count</th>
</tr>
<tr>
  <th>word</th>
  <th></th>
</tr>

<tr>
  <th>fake</th>
  <td>22963.0</td>
</tr>
<tr>
  <th>news</th>
  <td>20463.0</td>
</tr>
<tr>
  <th>ds100</th>
  <td>20432.0</td>
</tr>
<tr>
  <th>great</th>
  <td>20159.0</td>
</tr>
<tr>
  <th>class</th>
  <td>20121.0</td>
</tr>

```
In [51]: top_20 = trump.join(tidy_format).loc[:, ['word', 'retweet_count']]
         top_20 = top_20.groupby('word').filter(lambda x: len(x) >= 25)
         top_20 = top_20.groupby('word').agg(np.median).sort_values('retweet_count', ascending=False)
         top_20 = top_20.iloc[0:20]

         top_20

Out[51]:              retweet_count
         word
```

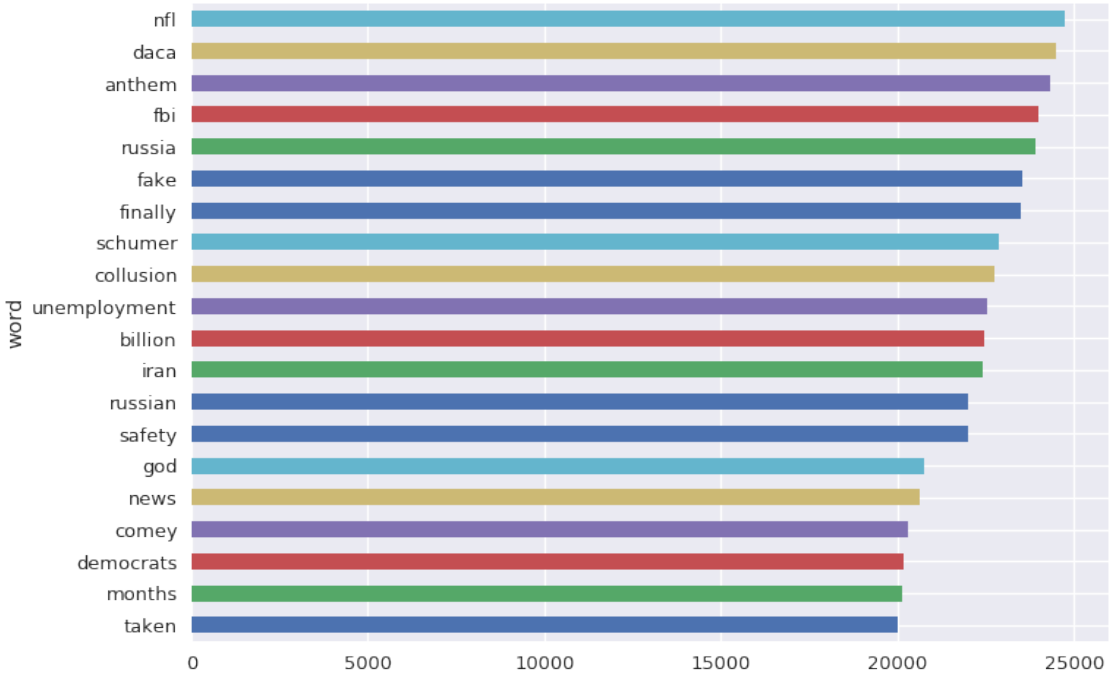
```
nfl              24748.0
daca             24495.0
anthem           24333.5
fbi              23975.0
russia           23922.0
fake             23530.0
finally          23468.0
schumer          22880.0
collusion        22748.0
unemployment     22533.0
billion          22437.0
iran             22412.0
russian          22004.0
safety           21975.0
god              20754.0
news             20617.0
comey            20290.5
democrats        20170.0
months           20108.0
taken            20000.0
```

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

# Although it can't be guaranteed, it's very likely that the top 7 words will still be
# 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));
```



2.21 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, unhelpful words can make into the top tweets. Based on the top_20 table, some of these unhelpful words include the word "finally" and "taken". The confounding factor would slightly change the position of "fake" and/or "news" on our list since there is only one unhelpful word that is located above the word "news"; by removing that unnecessary word ("finally"), we can increase the position of the word "news" to be one level above.
2. True. When we don't remove the hashtag, the counting of the same word differes depending if a hashtag is attached to its front or not; thus it would divide the total count of the same word into the count with the hashtag and without hashtag. In other words, this confounding factor might make "fake" and/or "news" lower in out ranking since it does not account to the words "fake" and/or "news" that are attached to a hashtag.
3. False, this factor does not really confound the list above since not only followers can retweet Trump's tweet. Anyone can retweet Trump's tweets.

2.22 Question 8

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

Ideas:

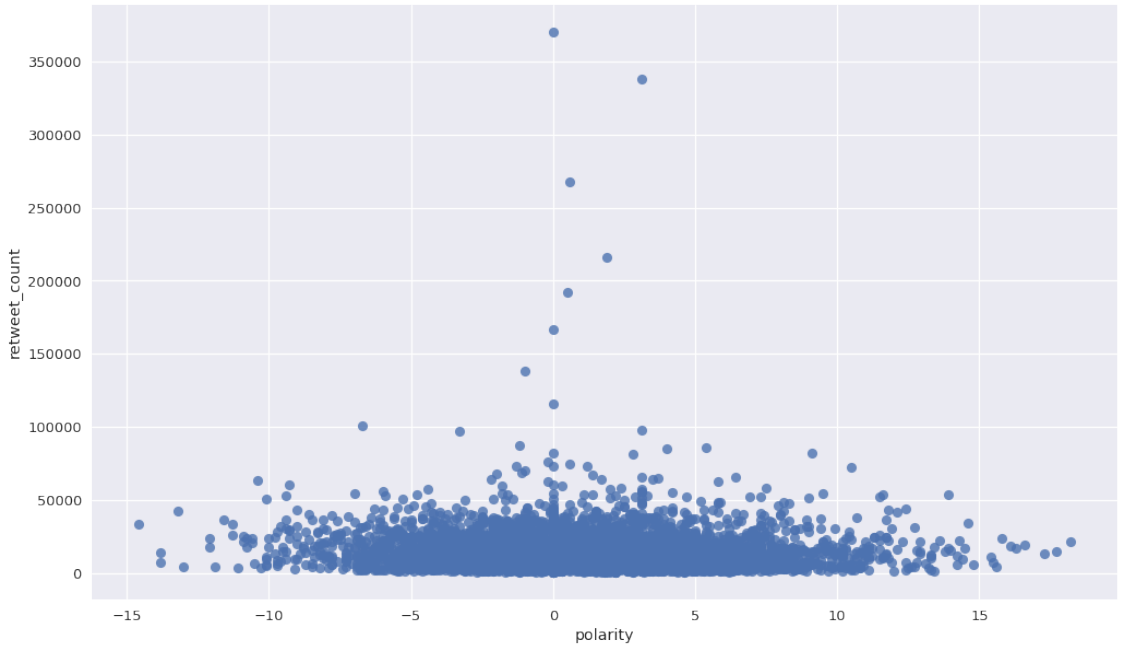
1. How has the sentiment changed with length of the tweets?
2. Does sentiment affect retweet count?
3. Are retweets more negative than regular tweets?
4. Are there any spikes in the number of retweets and do the correspond to world events?
5. *Bonus:* How many Russian twitter bots follow Trump?

You can look at other data sources and even tweets.

2.22.1 Plot:

In [54]:
```
sentiment_length = trump
sentiment_length = sentiment_length.loc[:, ['retweet_count', 'polarity']]
plt.figure(figsize=(15, 10))
sns.regplot('polarity', 'retweet_count', data=sentiment_length, fit_reg=False)
```

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d97ea7550>



2.22.2 Discussion:

It seems that the tweets with the most retweets are the tweets that have polarity values near the neutral range, specifically around between -1 and +4 polarity values. This might be due to the

different types of supporters; some are from the radical or extremist group while the others are from the typical conservative group. The extremists might tend to retweet more on the negative polarity tweets while the conservatives like tend to retweet more on the positive polarity tweets that include Trump's boasts on his accomplishments. Thus, the neutral-leaning polarity tweets have the most retweets since both spectrums of Trump's supporters as a whole tend to agree on the neutral-leaning polarity.

2.23 Submission

Congrats, you just finished Project 1!