proj1

October 26, 2018

Before you turn this problem 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).

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 = "William Sheu"
COLLABORATORS = ""

Update: For those of you who have trouble getting your twitter developer account working: 1. Run the cell to set up your notebook below (the one that starts with import csv). 2. Skip Question 1 and Question 2a. 3. Start with Question 2a-X instead. 4. After completing 2a-X, continue on to question 2b and proceed as normal.

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 Thursday, October 25, California Time.

You do not have to work on this project before the midterm, but you might find it helpful, since it goes over a lot of pandas materials that we haven't used in a while.

Fun:

We intend 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.

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
import json

```python
from pprint import pprint

# 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 by clicking the 'Apply' button on the top right of the page. Attach it to your Twitter account. You'll have to fill out a form describing what you want to do with the developer account. Explain that you are doing this for a class at UC Berkeley and that you don't know exactly what you're building yet and just need the account to get started. These applications are approved by some sort of AI system, so it doesn't matter exactly what you write. Just don't enter a bunch of alweiofalwiuhflawiuehflawuih-flaiwhfe type stuff or you might get rejected.
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 the sp18 instructors accidentally re-tweeted some tweets because that instructor typed retweet instead of retweet_count.

```python
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.

```python
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: ItsDenero

3

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 Oct 17 14:12:03 +0000 2018',
 'display_text_range': [0, 140],
 'entities': {'hashtags': [],
              'symbols': [],
              'urls': [],
              'user_mentions': [{'id': 263020833,
                                 'id_str': '263020833',
                                 'indices': [3, 19],
```

4

                                'name': 'Berkeley School of Information',
                                'screen_name': 'BerkeleyISchool'},
                           {'id': 24414370,
                            'id_str': '24414370',
                            'indices': [26, 34],
                            'name': 'Anno Saxenian',
                            'screen_name': 'annosax'}]},
 'favorite_count': 0,
 'favorited': False,
 'full_text': 'RT @BerkeleyISchool: Dean @annosax is one 3 women who have '
            'authored key works to understand the current Entrepreneurial '
            'Age. https://t.co/k',
 'geo': None,
 'id': 1052562888731320322,
 'id_str': '1052562888731320322',
 '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': 4,
 'retweeted': False,
 'retweeted_status': {'contributors': None,
                      'coordinates': None,
                      'created_at': 'Wed Oct 17 00:10:48 +0000 2018',
                      'display_text_range': [0, 198],
                      'entities': {'hashtags': [{'indices': [154, 168],
                                                 'text': 'SiliconValley'},
                                                {'indices': [169, 186],
                                                 'text': 'Entrepreneurship'}],
                                   'media': [{'display_url': 'pic.twitter.com/XDv65ShEbG',
                                              'expanded_url': 'https://twitter.com/BerkeleyISchool/status/10523511851125(
                                              'id': 1052351182730129408,
                                              'id_str': '1052351182730129408',
                                              'indices': [199, 222],
                                              'media_url': 'http://pbs.twimg.com/media/Dpqz0zmXcAAL2gF.png',
                                              'media_url_https': 'https://pbs.twimg.com/media/Dpqz0zmXcAAL2gF.png',
                                              'sizes': {'large': {'h': 456,
                                                                  'resize': 'fit',
                                                                  'w': 960},
                                                        'medium': {'h': 456,
                                                                   'resize': 'fit',
                                                                   'w': 960},
                                                        'small': {'h': 323,
                                                                  'resize': 'fit',

'w': 680},
                                  'thumb': {'h': 150,
                                            'resize': 'crop',
                                            'w': 150}},
                       'type': 'photo',
                       'url': 'https://t.co/XDv65ShEbG'}],
            'symbols': [],
            'urls': [{'display_url': 'buff.ly/2pYmNje',
                      'expanded_url': 'https://buff.ly/2pYmNje',
                      'indices': [104, 127],
                      'url': 'https://t.co/ke8memYKaQ'}],
            'user_mentions': [{'id': 24414370,
                               'id_str': '24414370',
                               'indices': [5, 13],
                               'name': 'Anno Saxenian',
                               'screen_name': 'annosax'},
                              {'id': 54711914,
                               'id_str': '54711914',
                               'indices': [131, 145],
                               'name': 'Nicolas Colin '
                                      '',
                               'screen_name': 'Nicolas_Colin'},
                              {'id': 91478624,
                               'id_str': '91478624',
                               'indices': [146, 153],
                               'name': 'Forbes',
                               'screen_name': 'Forbes'},
                              {'id': 176932593,
                               'id_str': '176932593',
                               'indices': [187, 198],
                               'name': 'UC Berkeley',
                               'screen_name': 'UCBerkeley'}]},
'extended_entities': {'media': [{'display_url': 'pic.twitter.com/XDv65ShEbG',
                                 'expanded_url': 'https://twitter.com/BerkeleyISchool/status/105235118
                                 'id': 1052351182730129408,
                                 'id_str': '1052351182730129408',
                                 'indices': [199, 222],
                                 'media_url': 'http://pbs.twimg.com/media/Dpqz0zmXcAAL2gF.png',
                                 'media_url_https': 'https://pbs.twimg.com/media/Dpqz0zmXcAAL2gF
                                 'sizes': {'large': {'h': 456,
                                                     'resize': 'fit',
                                                     'w': 960},
                                           'medium': {'h': 456,
                                                      'resize': 'fit',
                                                      'w': 960},
                                           'small': {'h': 323,
                                                     'resize': 'fit',
                                                     'w': 680},

                                                'thumb': {'h': 150,
                                                          'resize': 'crop',
                                                          'w': 150}},
                                'type': 'photo',
                                'url': 'https://t.co/XDv65ShEbG'}]},
'favorite_count': 9,
'favorited': False,
'full_text': 'Dean @annosax is one 3 women who have '
             'authored key works to understand the '
             'current Entrepreneurial Age. '
             'https://t.co/ke8memYKaQ By @Nicolas_Colin '
             '@Forbes #SiliconValley #Entrepreneurship '
             '@UCBerkeley https://t.co/XDv65ShEbG',
'geo': None,
'id': 1052351185112506368,
'id_str': '1052351185112506368',
'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': 4,
'retweeted': False,
'source': '<a href="https://buffer.com" '
          'rel="nofollow">Buffer',
'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': 3228,

                        'follow_request_sent': False,
                        'followers_count': 4983,
                        'following': False,
                        'friends_count': 652,
                        'geo_enabled': True,
                        'has_extended_profile': False,
                        'id': 263020833,
                        'id_str': '263020833',
                        'is_translation_enabled': False,
                        'is_translator': False,
                        'lang': 'en',
                        'listed_count': 218,
                        'location': 'Berkeley, California, USA',
                        'name': 'Berkeley School of Information',
                        'notifications': False,
                        'profile_background_color': '38628F',
                        'profile_background_image_url': 'http://abs.twimg.com/images/themes/theme1/bg.png'
                        'profile_background_image_url_https': 'https://abs.twimg.com/images/themes/theme1/
                        'profile_background_tile': False,
                        'profile_banner_url': 'https://pbs.twimg.com/profile_banners/263020833/1520571883',
                        'profile_image_url': 'http://pbs.twimg.com/profile_images/971975267764789254/eAmDC
                        'profile_image_url_https': 'https://pbs.twimg.com/profile_images/971975267764789254/
                        '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': 3952,
                        'time_zone': None,
                        'translator_type': 'none',
                        'url': 'https://t.co/5eXJ0wN1Jd',
                        'utc_offset': None,
                        'verified': False}},
 'source': 'Twitter Web Client',
 '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': 172,
 'follow_request_sent': False,
 'followers_count': 11764,
 'following': False,
 'friends_count': 412,
 'geo_enabled': False,
 'has_extended_profile': False,
 'id': 1227698863,
 'id_str': '1227698863',
 'is_translation_enabled': False,
 'is_translator': False,
 'lang': 'en',
 'listed_count': 487,
 'location': 'Berkeley, CA',
 'name': 'datascience@berkeley',
 'notifications': False,
 'profile_background_color': 'CCCCCC',
 'profile_background_image_url': 'http://abs.twimg.com/images/themes/theme1/bg.png',
 'profile_background_image_url_https': 'https://abs.twimg.com/images/themes/theme1/bg.png',
 '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': 2407,
 'time_zone': None,
 'translator_type': 'none',
 'url': 'http://t.co/S79Ul3oCaa',
 'utc_offset': None,
 '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.

```python
In [9]: 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."""

    # YOUR CODE HERE
    with open(path) as f:
        keys = json.load(f)
    return keys
    #raise NotImplementedError()

In [10]: 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

    # YOUR CODE HERE
    auth = tweepy.OAuthHandler(keys["consumer_key"], keys["consumer_secret"])
    auth.set_access_token(keys["access_token"], keys["access_token_secret"])
    api = tweepy.API(auth)
    return [t._json for t in tweepy.Cursor(api.user_timeline, id=user_account_name, tweet_mode='exten
    #raise NotImplementedError()
```

In [11]: 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"""
    # YOUR CODE HERE
    with open(path, "w") as f:
        json.dump(tweets, f)
    return
    #raise NotImplementedError()

In [12]: 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."""

    # YOUR CODE HERE
    with open(path, "r") as f:
        return json.load(f)
    #raise NotImplementedError()

In [18]: 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.
"""

```python
# YOUR CODE HERE
import json
tweets_save_path = user_account_name + "_recent_tweets.json"
if not Path(tweets_save_path).is_file():
    keys = load_keys(keys_path)
    tweets = download_recent_tweets_by_user(user_account_name, keys)
    save_tweets(tweets, tweets_save_path)
return load_tweets(tweets_save_path)
#raise NotImplementedError()
```

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)

```python
In [26]: # 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: 3009

2.4.2 Question 2a-X

This alternate starting point is here for those of you who have trouble getting a Twitter developer account. You should only use this if Twitter has rejected your application or if they have gone silent on you for at least a day. You'll miss out on some of the learning experience... but it seems like this might be necessary for some. If your twitter account is working fine, skip this problem!

Start by running the following cells, which will download and then load Donald Trump's most recent tweets.

```python
In [20]: # Download the dataset
         from utils import fetch_and_cache
         data_url = 'http://www.ds100.org/fa18/assets/datasets/realdonaldtrump_recent_tweets.json'
         file_name = 'realdonaldtrump_recent_tweets.json'
```

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

Using version already downloaded: Mon Oct 22 23:46:38 2018
MD5 hash of file: 216176fb098cd5d6b40b373b98bd3e6d
Located at data/realdonaldtrump_recent_tweets.json

```
In [21]: 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, "rb") as f:
                 import json
                 return json.load(f)
```

```
In [22]: trump_tweets = load_tweets(dest_path)
```

If everything is working correctly correctly this should load roughly the last 3000 tweets by realdonaldtrump.

```
In [23]: assert 2000 <= len(trump_tweets) <= 4000
```

If the assert statement above works, then continue on to question 2b.

2.4.3   Question 2b

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

```
In [27]: # Enter the number of the month of the oldest tweet (e.g. 1 for January)
         oldest_month = 10

         # YOUR CODE HERE
         pprint(trump_tweets[len(trump_tweets)-1]['created_at'])
         #raise NotImplementedError()
```

'Wed Nov 22 10:25:22 +0000 2017'

2.5 Question 3

IMPORTANT! PLEASE READ

Unfortunately, even if you have a working Twitter developer account, you cannot download older tweets using the public APIs. Fortunately, we have a snapshot of earlier tweets that we can combine with the newer data that you downloaded

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

```
In [28]: # Download the dataset
         from utils import fetch_and_cache
         data_url = 'http://www.ds100.org/fa18/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: Mon Oct 22 23:46:38 2018
MD5 hash of file: b6e33874de91d1a40207cdf9f9b51a09
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 [29]: 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 [30]: 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},

14

                                               '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/im
                                   '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',
'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 [31]: old_trump_tweets[0].keys()

Out[31]: dict_keys(['created_at', 'id', 'id_str', 'text', 'truncated', 'entities', 'extended_entities', 'source', 'in_repl

Since we're giving you a zipfile of old tweets, you may wonder why we didn't just give you a zipfile of ALL tweets and save you the trouble of creating a Twitter developer account. The reason is that we wanted you to see what it's like to collect data from the real world on your own. It can be a pain!

And for those of you that never got your developer accounts, you can see it can be even more of a pain that we expected. Sorry to anybody that wasted a bunch of time trying to get things working.

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.

17

```
In [32]: current_ids = [tweet['id'] for tweet in trump_tweets]
         all_tweets = trump_tweets + [tweet for tweet in old_trump_tweets if tweet['id'] not in current_ids]
         # YOUR CODE HERE
         #raise NotImplementedError()

In [33]: 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 [34]: def find_text(tweet):
             if 'text' in tweet:
                 return tweet['text']
             return tweet['full_text']
         trumpm = [[tweet['id'], pd.to_datetime(tweet['created_at']), tweet['source'], find_text(tweet), tweet['retwe
         trump = pd.DataFrame(trumpm, columns = ['id', 'time', 'source', 'text', 'retweet_count']).sort_values('id
         trump.head(10)
         # YOUR CODE HERE
         #raise NotImplementedError()
```

```
Out[34]:                                    time   \
         id
         690171032150237184 2016-01-21 13:56:11
         690171403388104704 2016-01-21 13:57:39
         690173226341691392 2016-01-21 14:04:54
         690176882055114758 2016-01-21 14:19:26
         690180284189310976 2016-01-21 14:32:57
         690271688127213568 2016-01-21 20:36:09
         690272687168458754 2016-01-21 20:40:07
         690313350278819840 2016-01-21 23:21:42
         690315202261155840 2016-01-21 23:29:04
         690315366564626433 2016-01-21 23:29:43


                                                                           source   \
         id
         690171032150237184  <a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andro
```

18

	source
690171403388104704	\<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andro
690173226341691392	\<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andro
690176882055114758	\<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andro
690180284189310976	\<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andro
690271688127213568	\<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhon
690272687168458754	\<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Andro
690313350278819840	\<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhon
690315202261155840	\<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhon
690315366564626433	\<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhon

id	text
690171032150237184	"@bigop1: @realDonaldTrum
690171403388104704	"@AmericanAsPie: @glennbeck @SarahPalinUSA Remember when Glenn ga
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
690271688127213568	New Day on CNN treats me very badly. @AlisynC
690272687168458754	Happy birthday to my friend, the gre
690313350278819840	Thank you, Iowa
690315202261155840	Thank you!
690315366564626433	Thank you, New Hamps

id	retweet_count
690171032150237184	1059
690171403388104704	1339
690173226341691392	2006
690176882055114758	2266
690180284189310976	2886
690271688127213568	1429
690272687168458754	1053
690313350278819840	2329
690315202261155840	1463
690315366564626433	1761

```
In [35]: assert isinstance(trump, pd.DataFrame)
         assert trump.shape[0] < 11000
         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')
```

19

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 [36]: trump['source'].unique()

Out[36]: array(['Twitter for Android',
          'Twitter for iPhone',
          'Twitter Web Client',
          'Mobile Web (M5)',
          'Instagram',
          'Twitter for iPad',
          'Media Studio',
          'Periscope',
          'Twitter Ads'], 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 [37]: ## Uncomment and complete
         trump['source'] = trump['source'].str.replace('<[^>]*>', '')
         trump['source'].unique()

   # YOUR CODE HERE
         #raise NotImplementedError()

Out[37]: array(['Twitter for Android', 'Twitter for iPhone', 'Twitter Web Client',
          'Mobile Web (M5)', 'Instagram', 'Twitter for iPad', 'Media Studio',
          'Periscope', 'Twitter Ads'], dtype=object)

In [38]: 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 [39]: trump['source'].value_counts().plot(kind="bar")
         plt.ylabel("Number of Tweets")

Out[39]: Text(0,0.5,'Number of Tweets')
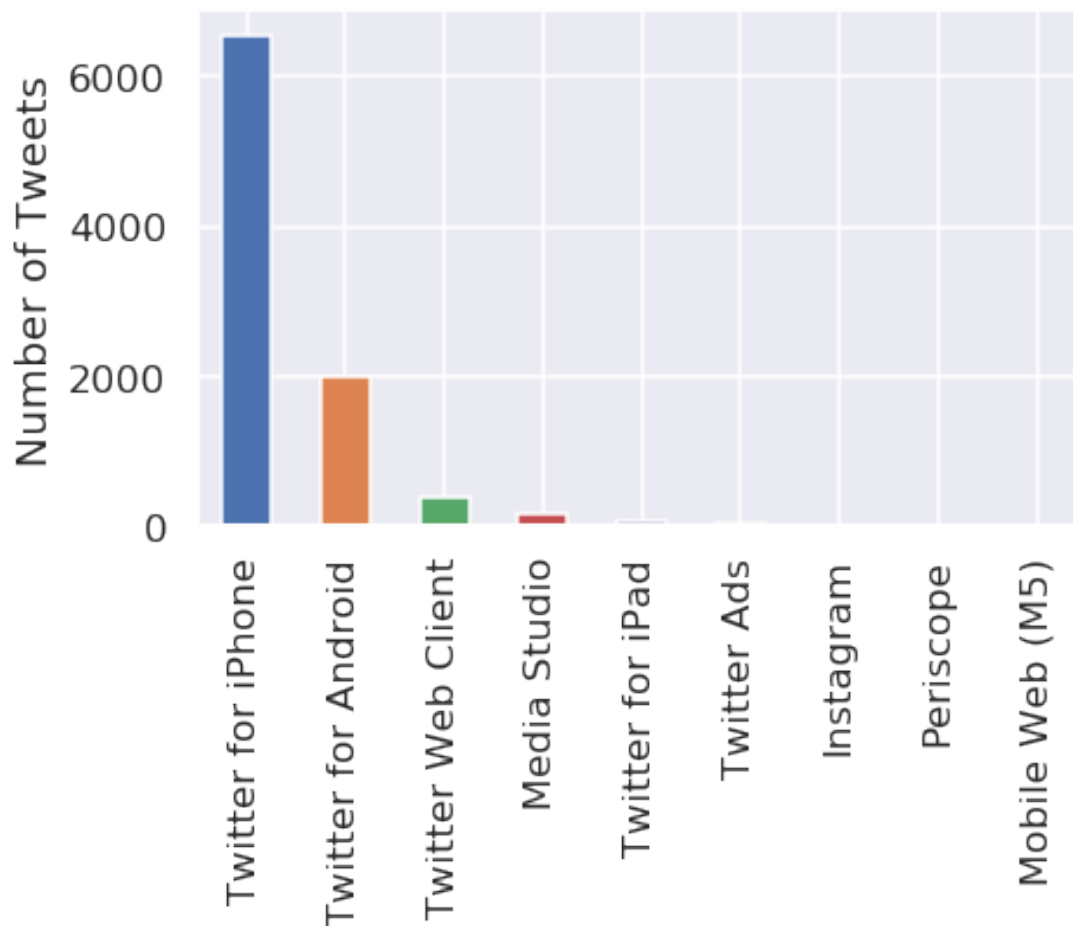
2.8 Question 4b

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

```
Mon Oct 22 21:35:03 +0000 2018
Mon Oct 22 19:55:58 +0000 2018
Mon Oct 22 19:54:04 +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 [41]: 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[41]:                                time            source  \
         id
         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

                          retweet_count              est_time
         id
         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 [42]: trump['hour'] = [t.hour + t.minute/60 + t.second/3600 for t in trump['est_time']]
         trump.head()

         # make a bar plot here
         # YOUR CODE HERE
         #raise NotImplementedError()
```

```
Out[42]:                                time            source  \
         id
         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
```

22

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

                     retweet_count   est_time   hour
id
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

In [43]: 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 eastern time that trump tweets on each device for the 2 most commonly used devices.
Your plot should look similar to the following.

```
In [45]: ### make your plot here
         ax=sns.distplot(trump[trump['source'] == 'Twitter for iPhone']['hour'], hist=False, label="iPhone")
         ax=sns.distplot(trump[trump['source'] == 'Twitter for Android']['hour'], hist=False, label="Android")
         ax.set_ylabel('fraction')
         # YOUR CODE HERE
         #raise NotImplementedError()
```

Out[45]: Text(0,0.5,'fraction')
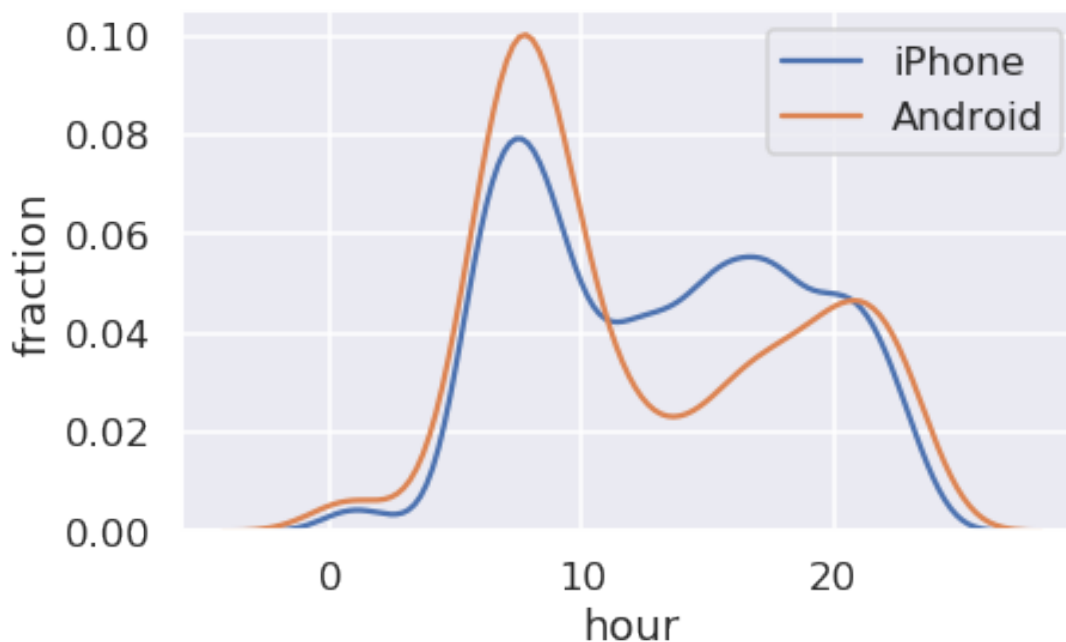
2.10 Question 4d

According to this Verge article, Donald Trump switched from an Android to an iPhone sometime in March 2017.

Create a figure identical to your figure from 4c, except that you should show the results only from 2016. If you get stuck consider looking at the `year_fraction` function from the next problem.

During the campaign, it was theorized that Donald Trump's tweets from Android were written by him personally, and the tweets from iPhone were from his staff. Does your figure give support to this theory?

```
In [46]: ### make your plot here
         # YOUR CODE HERE
         trump['year1'] = [t.year for t in trump['est_time']]
         ax=sns.distplot(trump[(trump['source'] == 'Twitter for iPhone') & (trump['year1'] == 2016)]['hour'], hist=
         ax=sns.distplot(trump[(trump['source'] == 'Twitter for Android') & (trump['year1'] == 2016)]['hour'], his
         trump = trump.drop(['year1'], axis = 1)
         ax.set_ylabel('fraction')
         #raise NotImplementedError()
```
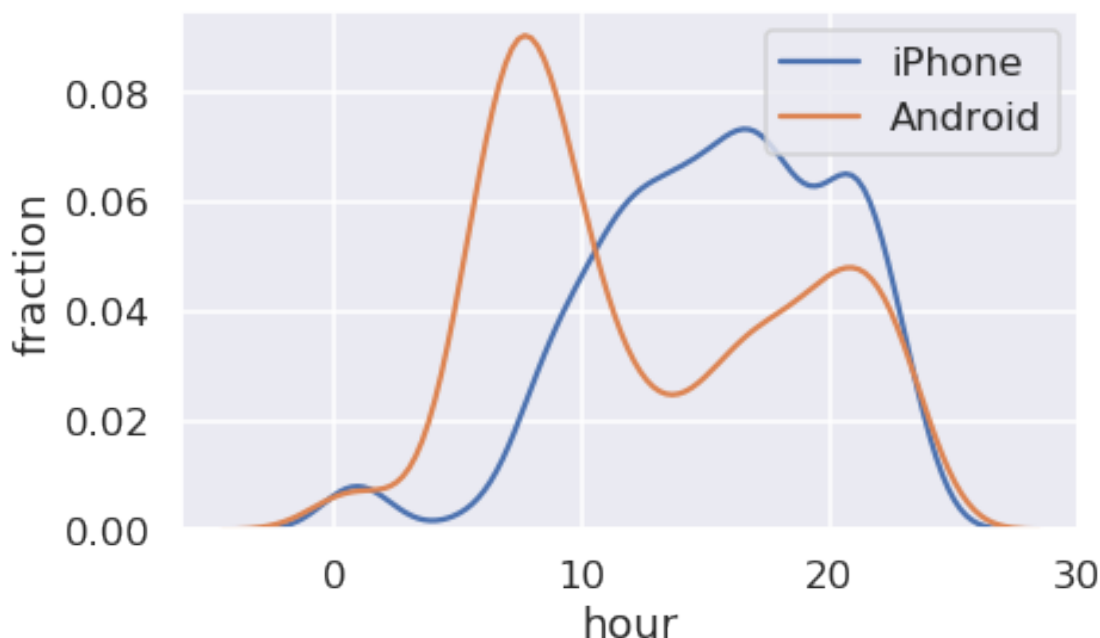
```
Out[46]: Text(0,0.5,'fraction')
```

Yes, my figure supports this theory, since the distribution of times for his iPhone's twitter is distinctly different to this Android's twitter. If he was posting these tweets himself, then it doesn't really make sense that there are two distinct time schedules.

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 [47]: 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 [48]: # YOUR CODE HERE
```
ax=sns.distplot(trump[trump['source'] == 'Twitter for iPhone']['year'], label="iPhone")
ax=sns.distplot(trump[trump['source'] == 'Twitter for Android']['year'], label="Android")
ax.legend()
#raise NotImplementedError()
```

Out[48]: <matplotlib.legend.Legend at 0x7f10afd0eda0>
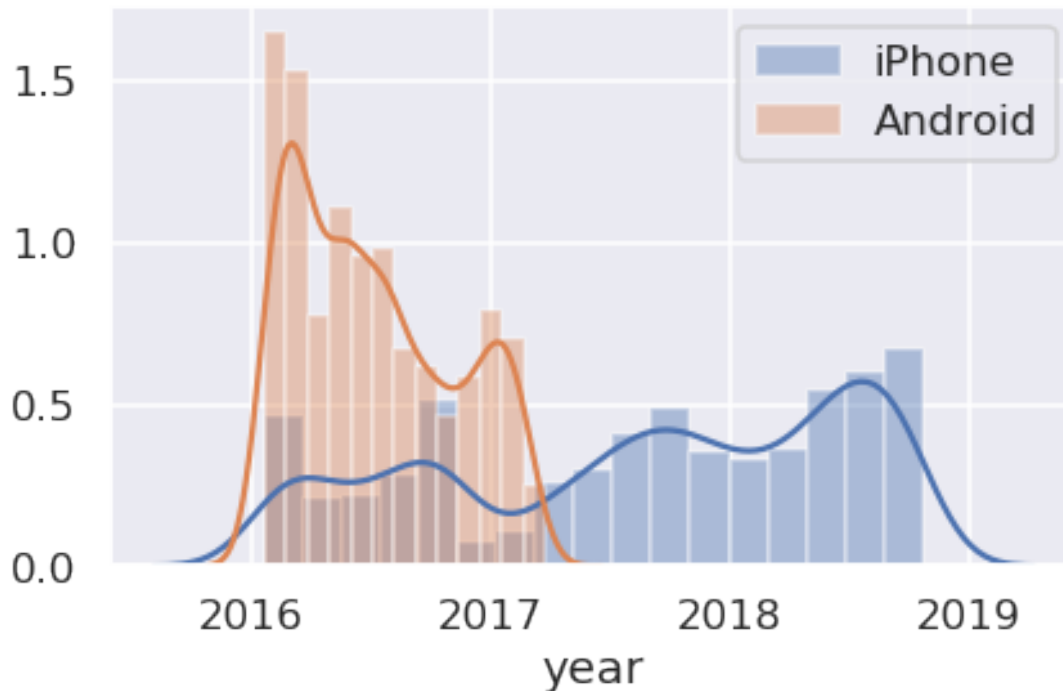
2.12   Question 6: Sentiment Analysis

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 [49]: 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]
```

26

```
%-)        -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 [51]: sent = pd.read_csv('vader_lexicon.txt', header=None, names=['polarity', '1', '2'], sep='\t').drop(column
         sent.head(10)

         # YOUR CODE HERE
         #raise NotImplementedError()

Out[51]:   polarity
         $:   -1.5
         %)   -0.4
         %-)   -1.5
         &-:   -0.4
         &:   -0.7
         ('}{')   1.6
         (%   -0.9
         ('-:   2.2
         (':   2.3
         ((-:   2.1

In [52]: 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 [54]: # YOUR CODE HERE
trump['text'] = [text.lower() for text in trump['text']]
trump.head(10)
#raise NotImplementedError()

Out[54]:   time   source   \
       id
       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

       id
       690171032150237184   "@bigop1: @realdonaldtrump
       690171403388104704   "@americanaspie: @glennbeck @sarahpalinusa remember when glenn gave o
       690173226341691392   so sad that @cnn and many others refused to show the massive crowd a
       690176882055114758   sad sack @jebbush has just done another ad on me, with special interest money, sa
       690180284189310976 low energy candidate @jebbush has wasted $80 million on his failed presidential cam
       690271688127213568   new day on cnn treats me very badly. @alisyncame
       690272687168458754   happy birthday to my friend, the gre
       690313350278819840   thank you, iowa!
       690315202261155840   thank you! #
       690315366564626433   thank you, new hampshi

                         retweet_count   est_time   hour \
       id
       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

28

```
                        year
id
690171032150237184   2016.054645
690171403388104704   2016.054645
690173226341691392   2016.054645
690176882055114758   2016.054645
690180284189310976   2016.054645
690271688127213568   2016.054645
690272687168458754   2016.054645
690313350278819840   2016.054645
690315202261155840   2016.054645
690315366564626433   2016.054645
```

In [55]: 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 [56]: # Save your regex in punct_re
         punct_re = r'[^(\w| | \\)\n]'
         trump['no_punc'] = trump['text'].str.replace(punct_re, ' ')
         trump.head(10)
         # YOUR CODE HERE
         #raise NotImplementedError()
```

```
Out[56]:                                 time            source  \
         id
         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


         id
         690171032150237184                                   "@bigop1: @realdonaldtrump
```

29

id		
690171403388104704	"@americanaspie: @glennbeck @sarahpalinusa remember when glenn gave o
690173226341691392	so sad that @cnn and many others refused to show the massive crowd a
690176882055114758	sad sack @jebbush has just done another ad on me, with special interest money, sa
690180284189310976	low energy candidate @jebbush has wasted $80 million on his failed presidential cam
690271688127213568	new day on cnn treats me very badly. @alisyncame
690272687168458754	happy birthday to my friend, the gre
690313350278819840	thank you, iowa!
690315202261155840	thank you! ≠
690315366564626433	thank you, new hampshi

	retweet_count	est_time	hour \
id			
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

	year \
id	
690171032150237184	2016.054645
690171403388104704	2016.054645
690173226341691392	2016.054645
690176882055114758	2016.054645
690180284189310976	2016.054645
690271688127213568	2016.054645
690272687168458754	2016.054645
690313350278819840	2016.054645
690315202261155840	2016.054645
690315366564626433	2016.054645

id		
690171032150237184	bigop1 realdonaldtrump
690171403388104704	americanaspie glennbeck sarahpalinusa remember when glenn gave out g
690173226341691392	so sad that cnn and many others refused to show the massive crowd at
690176882055114758	sad sack jebbush has just done another ad on me with special interest money say
690180284189310976	low energy candidate jebbush has wasted 80 million on his failed presidential campa
690271688127213568	new day on cnn treats me very badly alisyncame
690272687168458754	happy birthday to my friend the gre
690313350278819840	thank you iowa
690315202261155840	thank you
690315366564626433	thank you new hampshi

```
In [57]: 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.

As usual, try to avoid using any for loops. Our solution uses a chain of 5 methods on the 'trump' DF, albeit using some rather advanced Pandas hacking.

- **Hint 1:** Try looking at the expand argument to pandas' str.split.

- **Hint 2:** Try looking at the stack() method.

- **Hint 3:** Try looking at the level parameter of the reset_index method.

In [58]: tidy_format = trump['no_punc'].str.split(expand=True).stack().to_frame().reset_index(level = 1)
         tidy_format.columns = ['num', 'word']
         tidy_format.head(10)
         # YOUR CODE HERE
         #raise NotImplementedError()

Out[58]:   num   word
         id
         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

In [59]: 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 [61]: mergedsent = sent.merge(tidy_format, how = "outer",left_index =True, right_on ='word').groupby(["id
         trump['polarity'] = mergedsent
         trump.head(10)

         # YOUR CODE HERE
         #raise NotImplementedError()
```

Out[61]:   time   source   \
         id
         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

         id
         690171032150237184   "@bigop1: @realdonaldtrump
         690171403388104704   "@americanaspie: @glennbeck @sarahpalinusa remember when glenn gave o
         690173226341691392   so sad that @cnn and many others refused to show the massive crowd a
         690176882055114758   sad sack @jebbush has just done another ad on me, with special interest money, sa
         690180284189310976 low energy candidate @jebbush has wasted $80 million on his failed presidential cam
         690271688127213568   new day on cnn treats me very badly. @alisyncame
         690272687168458754   happy birthday to my friend, the gre
         690313350278819840   thank you, iowa!
         690315202261155840   thank you! ≠
         690315366564626433   thank you, new hampshi

                        retweet_count   est_time   hour \
         id
         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

                        year \
         id

33
```

```
690171032150237184  2016.054645
690171403388104704  2016.054645
690173226341691392  2016.054645
690176882055114758  2016.054645
690180284189310976  2016.054645
690271688127213568  2016.054645
690272687168458754  2016.054645
690313350278819840  2016.054645
690315202261155840  2016.054645
690315366564626433  2016.054645


id
690171032150237184                                          bigop1   realdonaldtrump
690171403388104704           americanaspie    glennbeck  sarahpalinusa remember when glenn gave out
690173226341691392                 so sad that  cnn and many others refused to show the massive crowd at
690176882055114758     sad sack  jebbush has just done another ad on me  with special interest money  say
690180284189310976   low energy candidate  jebbush has wasted  80 million on his failed presidential camp
690271688127213568                            new day on cnn treats me very badly   alisyncame
690272687168458754                                  happy birthday to my friend  the gre
690313350278819840                                                thank you  iowa
690315202261155840                                                      thank you
690315366564626433                                             thank you  new hampshi


                polarity
id
690171032150237184       0.0
690171403388104704      -2.6
690173226341691392      -6.0
690176882055114758       4.3
690180284189310976      -2.6
690271688127213568      -5.2
690272687168458754       9.7
690313350278819840       1.5
690315202261155840       1.5
690315366564626433       1.5
```

In [62]: 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.

34

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

```
In [63]: print('Most negative tweets:')
         for t in trump.sort_values('polarity').head()['text']:
             print('\n ', t)
```

Most negative tweets:

   massive crowds inside and outside of the @toyotacenter in houston, texas. landing shortly - see everyone soon!

   it is outrageous that poisonous synthetic heroin fentanyl comes pouring into the u.s. postal system from china.

   the rigged russian witch hunt goes on and on as the originators and founders of this scam continue to be fired a

   james comey is a proven leaker & liar. virtually everyone in washington thought he should be fired for the

   there is no collusion! the robert mueller rigged witch hunt, headed now by 17 (increased from 13, including an o

```
In [64]: print('Most positive tweets:')
         for t in trump.sort_values('polarity', ascending=False).head()['text']:
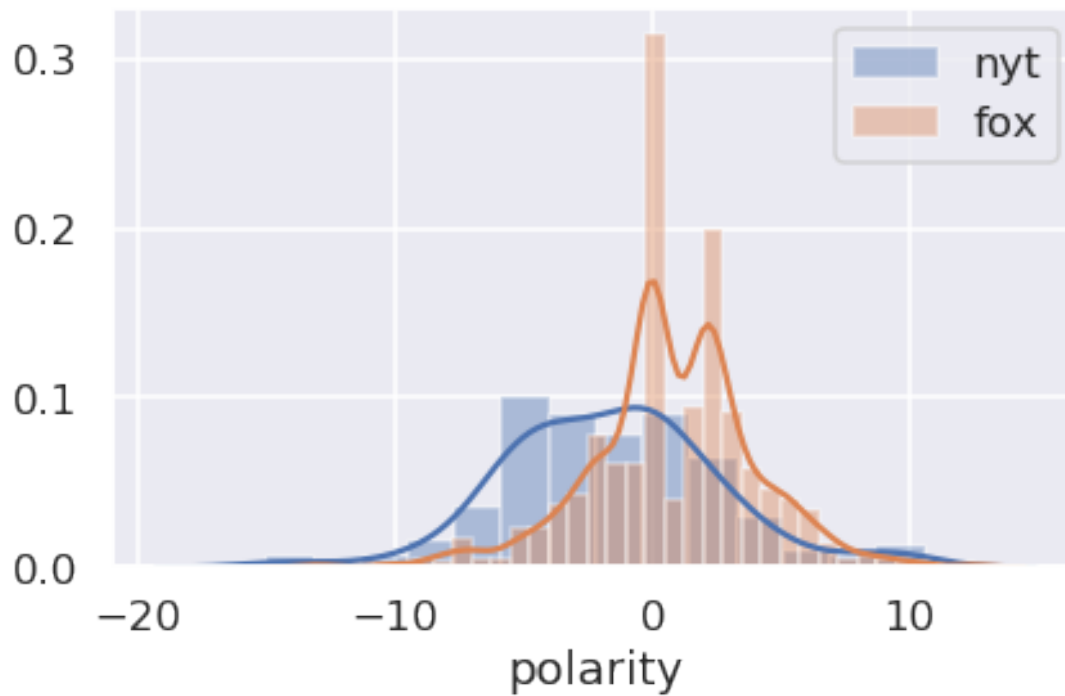             print('\n ', t)
```

Most positive tweets:

   congratulations to patrick reed on his great and courageous masters win! when patrick had his amazing win at

   my supporters are the smartest, strongest, most hard working and most loyal that we have seen in our countrie

   thank you to all of my great supporters, really big progress being made. other countries wanting to fix crazy tra

   thank you, @wvgovernor jim justice, for that warm introduction. tonight, it was my great honor to attend the g

   the republican party had a great night. tremendous voter energy and excitement, and all candidates are those v

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 [65]: # YOUR CODE HERE
         ax=sns.distplot(trump['polarity'][trump['no_punc'].str.contains("nyt")], label = "nyt")
         ax=sns.distplot(trump['polarity'][trump['no_punc'].str.contains("fox")], label = "fox")
         ax.legend()
         #raise NotImplementedError()
```

Out[65]: <matplotlib.legend.Legend at 0x7f10aeaef518>

Comment on what you observe: The tweets containing nyt are mostly negative, whereas the tweets containing foc are mostly positive.

In [66]: tidy_format

Out[66]:

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
...	...	...
1054461380797943809	7	0pwiwcq4mh
1054461380797943809	8	https
1054461380797943809	9	t
1054461380797943809	10	co
1054461380797943809	11	6kavm1taib
1054486312701517824	0	massive
1054486312701517824	1	crowds
1054486312701517824	2	inside
1054486312701517824	3	and
1054486312701517824	4	outside
1054486312701517824	5	of
1054486312701517824	6	the
1054486312701517824	7	toyotacenter
1054486312701517824	8	in
1054486312701517824	9	houston
1054486312701517824	10	texas
1054486312701517824	11	landing
1054486312701517824	12	shortly
1054486312701517824	13	see
1054486312701517824	14	everyone
1054486312701517824	15	soon
1054486312701517824	16	maga
1054486312701517824	17	https
1054486312701517824	18	t
1054486312701517824	19	co
1054486312701517824	20	0pwiwcq4mh
1054486312701517824	21	https
1054486312701517824	22	t
1054486312701517824	23	co
1054486312701517824	24	tj5s6z2gy7

[219481 rows x 2 columns]

2.19 Question 7: Engagement

2.20 Question 7a

In this problem, we'll explore which words led to a greater average number of retweets. For example, at the time of this writing, Donald Trump has two tweets that contain the word 'oakland' (tweets 932570628451954688 and 1016609920031117312) with 36757 and 10286 retweets respectively, for an average of 23,521.5.

Find the top 20 most retweeted words. Include only words that appear in at least 25 tweets. As usual, try to do this without any for loops. You can string together ~7 pandas commands and get everything done on one line.

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>jong</th>
  <td>40675.666667</td>
</tr>
<tr>
  <th>try</th>
  <td>33937.800000</td>
</tr>
<tr>
  <th>kim</th>
  <td>32849.595745</td>
</tr>
<tr>
  <th>un</th>
  <td>32741.731707</td>
</tr>
<tr>
  <th>maybe</th>
  <td>30473.192308</td>
</tr>
</tr>
```

```
In [68]: def meanif25(x):
             if len(x)>=24: #'try' only appears 24 times; it seems that within a month, 'try' has fallen below 25 twe
                 return np.mean(x)
             return 0
         top_20 = tidy_format.merge(trump, how='outer', left_index=True, right_index=True).loc[:,['word', 'retw
         top_20
```

```
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[68]:   retweet_count
         word
         jong   40041.466667
         try   33294.625000
         un   32271.804878
         kim   31886.122449
         maybe   30337.615385
         kavanaugh   28593.481481
         illegally   28569.250000
         nuclear   28242.163265
         old   27902.666667
         mccabe   27502.600000
         players   27116.900000
         lowest   26434.755556
         flag   26421.600000
         nfl   26312.500000
         emails   25965.294118
         ban   25622.041667
         judge   25120.666667
         fbi   25007.106061
         short   25006.000000
         anthem   24995.621622

In [69]: # Although it can't be guaranteed, it's very likely that the top 5 words will still be
         # in the top 20 words in the next month.
         assert 'jong'   in top_20.index
         assert 'try'   in top_20.index
         assert 'kim' in top_20.index
         assert 'un'   in top_20.index
         assert 'maybe'   in top_20.index

Here's a bar chart of your results:

In [70]: top_20['retweet_count'].sort_values().plot.barh(figsize=(10, 8));
```
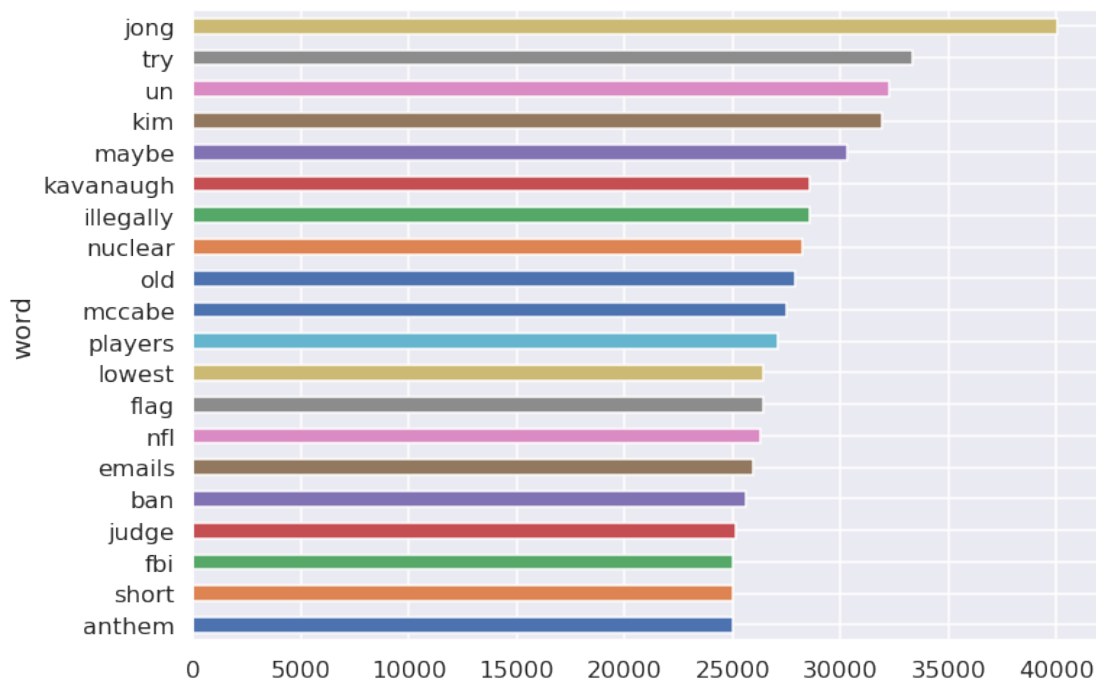
## 2.21   Question 7b

"kim", "jong" and "un" are apparently really popular in Trump's tweets! It seems like we can conclude that his tweets involving jong are more popular than 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 kim jong un related tweets higher in the 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.) This is true, and it certainly would have resulted in "kim" and "un" higher on the list, since "try" (which is not a useful word) is above "kim" and "un".

2.) This is false, since we did remove all punctuations, including hashtags.

3.) This is true; we did not account for the fact that Trump's follower count has increased over time. However, if we are trying to answer whether tweets involving "kim", "jong" and "un" are more popular than his others, then this factor is irrelevant. It may be the case that these tweets are more popular because his follower count has increased over time, but regardless we can still say that his tweets involving "jong" are more popular than his other tweets, if we define popularity as the number of people who have seen the tweet.

## 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?
6. What terms have an especially positive or negative sentiment?

You can look at other data sources and even tweets.

### 2.22.1 Plot:

```
In [80]: # YOUR CODE HERE
         plt.figure(figsize=(15, 9))
         plt.title("Boxplot comparison of retweet counts between different polarities")

         trumpall=[trump[(trump["polarity"]>=n-0.5)&(trump["polarity"]<n+0.5)]["retweet_count"] for n in np.a

         #trump0=trump[(trump["polarity"]>=-15)&(trump["polarity"]<=-10)]["retweet_count"]
         #trump1=trump[(trump["polarity"]>=-10)&(trump["polarity"]<=-5)]["retweet_count"]
         #trump2=trump[(trump["polarity"]>=-5)&(trump["polarity"]<=0)]["retweet_count"]
         #trump3=trump[(trump["polarity"]>=0)&(trump["polarity"]<=5)]["retweet_count"]
         #trump4=trump[(trump["polarity"]>=5)&(trump["polarity"]<=10)]["retweet_count"]
         #trump5=trump[(trump["polarity"]>=10)&(trump["polarity"]<=15)]["retweet_count"]

         p={str(x): trumpall[x+13] for x in np.arange(-13, 14)}
         #p={"-15":trumpall[0], "-10 to -5":trump1, "-5 to 0":trump2, "0 to 5":trump3, "5 to 10":trump4, "10 to 1
         trumpp=pd.DataFrame(data=p)
         trumpp=trumpp[[str(x) for x in np.arange(-13, 14)]]
         plt.ylabel('retweet count')
         plt.xlabel('polarity rounded to nearest integer')
         sns.boxplot(data=trumpp, showfliers=False)
         #raise NotImplementedError()

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x7f10add22208>
```
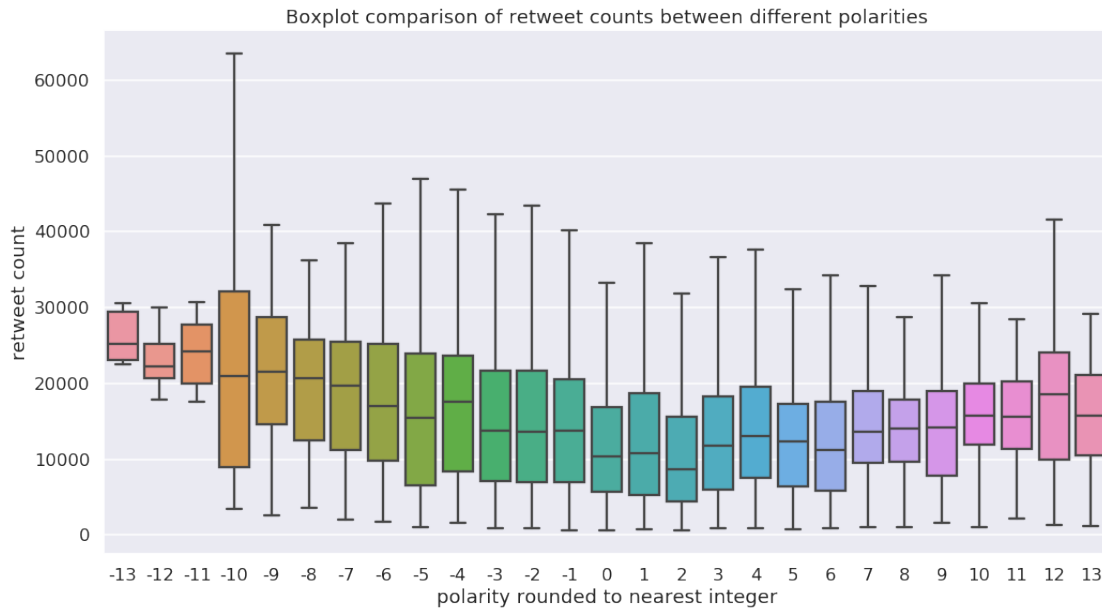
Boxplot comparison of retweet counts between different polarities

### 2.22.2 Discussion of Your Plot:

From this plot, we can tell that, while Trump's positive tweets get on average more retweets than his neutral tweets, Trump's negative tweets get much more retweets than both his positive and his neutral tweets. We could also try to draw a relation between polarity and retweet count; retweet counts seem to increase the further away a tweet gets from 0 polarity (neutral). Also it seems to increase at a faster rate if the polarity is negative and moving away from 0, than being positive and moving away from zero.

Another observation to make is that it seems as if the retweet count median seems to plateau after reaching a distance away from 0. It is especially true for the negative side, as the change of the median is much steeper from

## 2.23 Submission

Congrats, you just finished Project 1!

## 2.24 Submission

You're done!

Before submitting this assignment, ensure to:

1. Restart the Kernel (in the menubar, select Kernel->Restart & Run All)
2. Validate the notebook by clicking the "Validate" button

Finally, make sure to **submit** the assignment via the Assignments tab in Datahub