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 [8]:
```
NAME = "Matthew Brennan"
COLLABORATORS = "Connor McCormick"
```

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.

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 [9]:
```
# 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
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
```

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 [10]:
```
## 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 (http://tweepy.readthedocs.io/en/v3.5.0/getting_started.html), but we will give you example code.

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

Question 1

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

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

WARNING (Please Read) !!!!

Protect your Twitter Keys

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

Avoid making too many API calls.

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

Be careful about which functions you call!

This API can retweet tweets, follow and unfollow people, and modify your twitter settings. Be careful which functions you invoke! One of the sp18 instructors accidentally re-tweeted some tweets because that instructor typed `retweet` instead of `retweet_count`.

In [11]:
```
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 [12]:
```
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)
```

```
WARNING:root:There was a Tweepy error. Double check your API keys and try again.
WARNING:root:[{'code': 89, 'message': 'Invalid or expired token.'}]
```

Question 2

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

In [13]:
```
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)
```

```
---------------------------------------------------------------------------
TweepError                                Traceback (most recent call last)
<ipython-input-13-9fe391f374d4> in <module>()
     13     # will use the basic Python dictionaries stored in the _json field.
     14     example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id="BerkeleyData",
---> 15                                                      tweet_mode='extended').items()]
     16
     17     # Saving the tweets to a json file on disk for future analysis

<ipython-input-13-9fe391f374d4> in <listcomp>(.0)
     12     # The tweepy Cursor API actually returns "sophisticated" Status objects but we
     13     # will use the basic Python dictionaries stored in the _json field.
---> 14     example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id="BerkeleyData",
     15                                                      tweet_mode='extended').items()]
     16

/srv/conda/envs/data100/lib/python3.6/site-packages/tweepy/cursor.py in __next__(self)
     47
     48     def __next__(self):
```

```
---> 49         return self.next()
     50
     51     def next(self):

/srv/conda/envs/data100/lib/python3.6/site-packages/tweepy/cursor.py in next(self)
    195         if self.current_page is None or self.page_index == len(self.current_page) - 1:
    196             # Reached end of current page, get the next page...
--> 197             self.current_page = self.page_iterator.next()
    198             self.page_index = -1
    199         self.page_index += 1

/srv/conda/envs/data100/lib/python3.6/site-packages/tweepy/cursor.py in next(self)
    106
    107         if self.index >= len(self.results) - 1:
--> 108             data = self.method(max_id=self.max_id, parser=RawParser(), *self.args, **self.karg
s)
    109
    110             if hasattr(self.method, '__self__'):

/srv/conda/envs/data100/lib/python3.6/site-packages/tweepy/binder.py in _call(*args, **kwargs)
    243             return method
    244         else:
--> 245             return method.execute()
    246
    247     # Set pagination mode

/srv/conda/envs/data100/lib/python3.6/site-packages/tweepy/binder.py in execute(self)
    227                     raise RateLimitError(error_msg, resp)
    228                 else:
--> 229                     raise TweepError(error_msg, resp, api_code=api_error_code)
    230
    231             # Parse the response payload

TweepError: Twitter error response: status code = 401
```

Assuming everything ran correctly you should be able to look at the first tweet by running the cell below.
Warning Do not attempt to view all the tweets in a notebook. It will likely freeze your browser. The following would be a **bad idea**:
```python pprint(example_tweets) ```
```

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

Question 2a

What you need to do.

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

In [ ]:
```
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
    raise NotImplementedError()
```

In [ ]:
```
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
    raise NotImplementedError()
```

In [ ]:
```
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
    raise NotImplementedError()
```

In [ ]:
```
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
    raise NotImplementedError()
```

In [ ]:
```
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.
    """

    # YOUR CODE HERE
    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)

In [ ]:
```
# 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))
```

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.

In [14]:
```
# 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: Wed Oct 24 03:23:14 2018
MD5 hash of file: 216176fb098cd5d6b40b373b98bd3e6d
Located at data/realdonaldtrump_recent_tweets.json
```

In [15]:
```
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 [16]:
```
trump_tweets = load_tweets(dest_path)
```

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

In [17]:
```
assert 2000 <= len(trump_tweets) <= 4000
```

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

Question 2b

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

In [18]:
```
# Enter the number of the month of the oldest tweet (e.g. 1 for January)
#determined through investigation of the data
oldest_month = 10

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

In [ ]:
```

```

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 [19]:
```
# 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: Wed Oct 24 03:23:14 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 [20]:
```
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 [ ]:
```
pprint(old_trump_tweets[0])
```

As a dictionary we can also list the keys:

In [21]:
```
old_trump_tweets[0].keys()
```

Out[21]:
```
dict_keys(['created_at', 'id', 'id_str', 'text', 'truncated', 'entities', 'extended_entities', 'sourc
e', 'in_reply_to_status_id', 'in_reply_to_status_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_
str', 'in_reply_to_screen_name', 'user', 'geo', 'coordinates', 'place', 'contributors', 'is_quote_stat
us', 'retweet_count', 'favorite_count', 'favorited', 'retweeted', 'possibly_sensitive', 'lang'])
```

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.

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 [22]:
```
#needed a means to isolate the individual ids
trump_tweet_ids = []
for tweet in trump_tweets:
    trump_tweet_ids.append(tweet['id'])
```

In [23]:
```
all_tweets = []
for tweet in old_trump_tweets:
    if tweet['id'] in trump_tweet_ids:
        continue
    else:
        all_tweets.append(tweet)

#now must add the values from trump_tweets
for tweet in trump_tweets:
    all_tweets.append(tweet)

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

In [24]:
```
len(all_tweets)
```

Out[24]:
```
9086
```

In [25]:
```
assert len(all_tweets) > len(trump_tweets)
assert len(all_tweets) > len(old_trump_tweets)
```

Question 3b

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

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

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

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

In [26]:
```
all_tweets[0]['created_at']
```

Out[26]:
```
'Wed Oct 12 14:00:48 +0000 2016'
```

In [27]:
```
time = []
source = []
text = []
retweet_count = []
id_num = []
for tweet in all_tweets:
    time.append(pd.to_datetime(tweet['created_at']))
    source.append(tweet['source'])
    retweet_count.append(tweet['retweet_count'])
    id_num.append(tweet['id'])
    if 'text' in tweet.keys():
        text.append(tweet['text'])
    else:
        text.append(tweet['full_text'])
```

In [28]:
```
trump = pd.DataFrame({'time': time,
                      'source': source,
                      'text': text,
                      'retweet_count': retweet_count,
                      'id': id_num
                     }).set_index('id').sort_index(ascending = True)

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

In [29]:
```
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')
```

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 [30]:
```
trump['source'].unique()
```

Out[30]:
```
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)
```

Question 4a

Remove the HTML tags from the source field.

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

In [31]:
```
## Uncomment and complete
trump['source'] = trump['source'].str.replace(r'<[^>]+>', '')

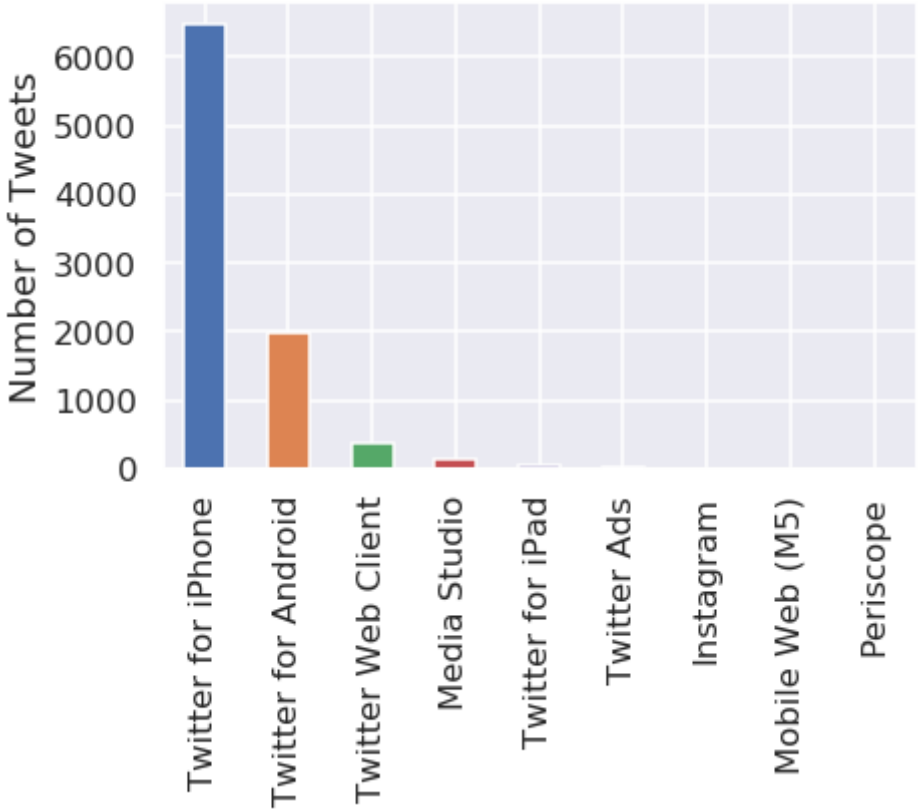

# YOUR CODE HERE
#raise NotImplementedError()
```

In [32]:
```
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()) == se
 '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 [33]:
```
trump['source'].value_counts().plot(kind="bar")
plt.ylabel("Number of Tweets")
```

Out[33]:
```
Text(0,0.5,'Number of Tweets')
```



Question 4b

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

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

In [34]:
```
for t in trump_tweets[0:3]:
    print(t['created_at'])
```

```
Tue Oct 16 16:22:11 +0000 2018
Tue Oct 16 16:18:08 +0000 2018
Tue Oct 16 15:26:33 +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 [35]:
```
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[35]:

id	retweet_count	source	text	time	est_time
690171032150237184	1059	Twitter for Android	"@bigop1: @realDonaldTrump @SarahPalinUSA https://t.co/3kYQGqeVyD"	2016-01-21 13:56:11	2016-01-21 08:56:11-05:00
690171403388104704	1339	Twitter for Android	"@AmericanAsPie: @glennbeck @SarahPalinUSA Remember when Glenn gave out gifts to ILLEGAL ALIENS at crossing the border? Me too!"	2016-01-21 13:57:39	2016-01-21 08:57:39-05:00
690173226341691392	2006	Twitter for Android	So sad that @CNN and many others refused to show the massive crowd at the arena yesterday in Oklahoma. Dishonest reporting!	2016-01-21 14:04:54	2016-01-21 09:04:54-05:00
690176882055114758	2266	Twitter for Android	Sad sack @JebBush has just done another ad on me, with special interest money, saying I won't beat Hillary - I WILL. But he can't beat me.	2016-01-21 14:19:26	2016-01-21 09:19:26-05:00
690180284189310976	2886	Twitter for Android	Low energy candidate @JebBush has wasted $80 million on his failed presidential campaign. Millions spent on me. He should go home and relax!	2016-01-21 14:32:57	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 [36]:
```
def time_math(timestamp):
    numeric_time = timestamp.hour + (timestamp.minute/60) + (timestamp.second/60**2)
    return numeric_time

trump['hour'] = trump['est_time'].apply(time_math)

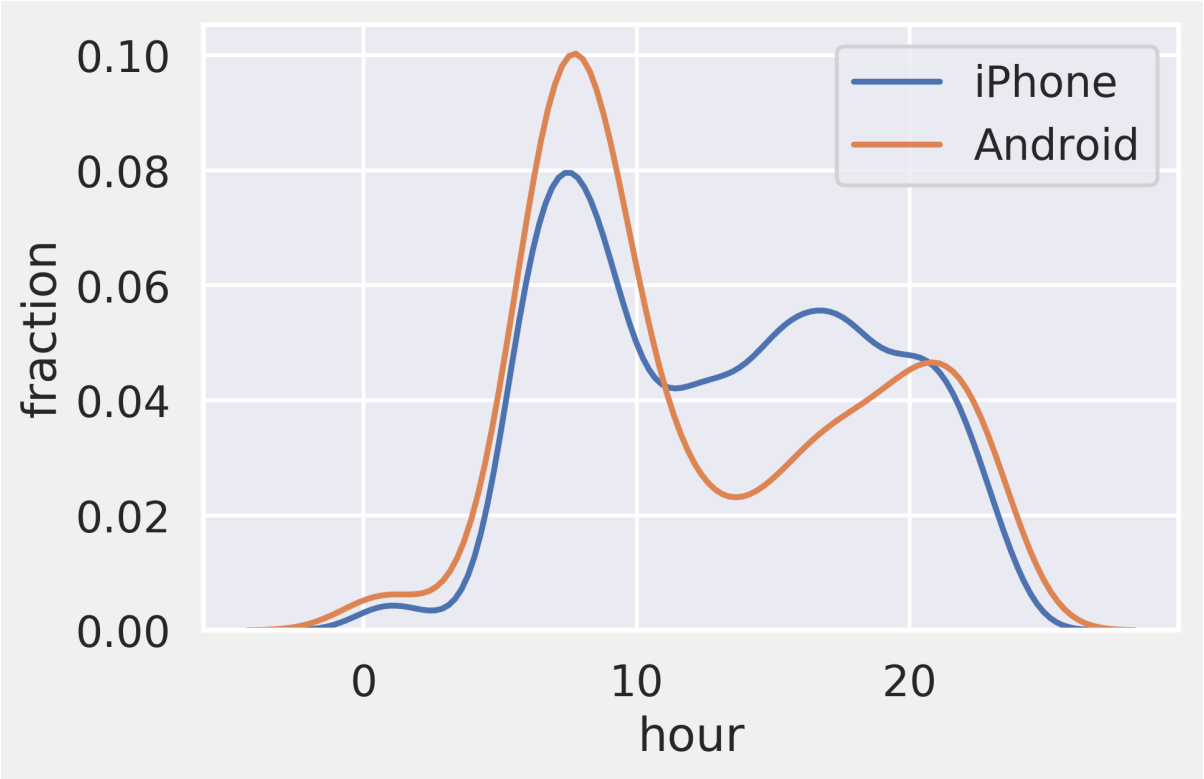
# YOUR CODE HERE
#raise NotImplementedError()
```

In [ ]:
```
```

In [37]:
```
assert np.isclose(trump.loc[690171032150237184]['hour'], 8.93639)
```

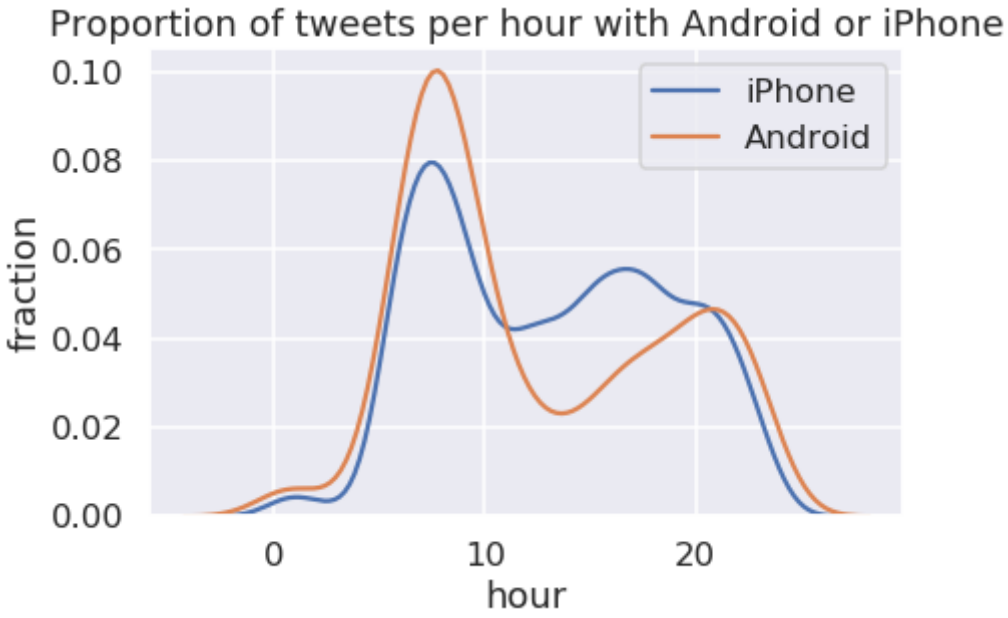
Question 4c

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



In [38]:
```
android_table = trump[trump["source"] == "Twitter for Android"][["source", 'hour']]
iphone_table = trump[trump["source"] == "Twitter for iPhone"][["source", 'hour']]
```

In [39]:
```
### make your plot here
sns.distplot(iphone_table['hour'], hist = False, label = 'iPhone')
sns.distplot(android_table["hour"], hist = False, label = 'Android')
plt.xlabel('hour')
plt.ylabel('fraction')
plt.title("Proportion of tweets per hour with Android or iPhone");
#raise NotImplementedError()
```



Question 4d

According to [this Verge article (https://www.theverge.com/2017/3/29/15103504/donald-trump-iphone-using-switched-android)](https://www.theverge.com/2017/3/29/15103504/donald-trump-iphone-using-switched-android), 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 [40]:
```
#we needed to move the function here so as to keep the information in order
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

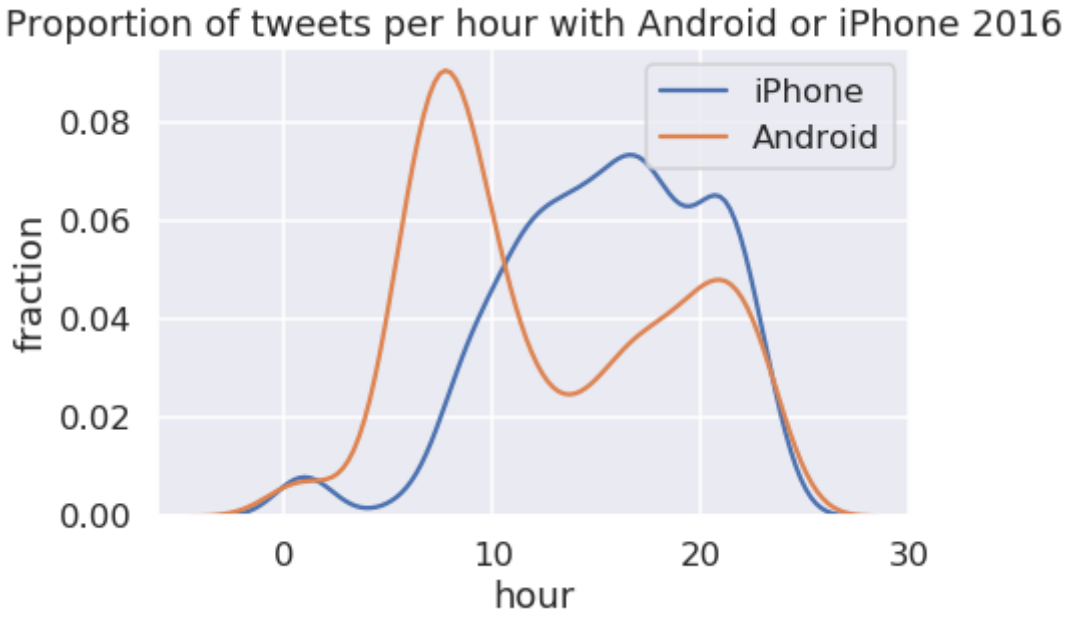

trump['year'] = trump['time'].apply(year_fraction)//1
trump
```

Out[40]:

id	retweet_count	source	text	time	est_time	hour	year
690171032150237184	1059	Twitter for Android	"@bigop1: @realDonaldTrump @SarahPalinUSA https://t.co/3kYQGqeVyD"	2016-01-21 13:56:11	2016-01-21 08:56:11-05:00	8.936389	2016.0
690171403388104704	1339	Twitter for Android	"@AmericanAsPie: @glennbeck @SarahPalinUSA Remember when Glenn gave out gifts to ILLEGAL ALIENS at crossing the border? Me too!"	2016-01-21 13:57:39	2016-01-21 08:57:39-05:00	8.960833	2016.0
690173226341691392	2006	Twitter for Android	So sad that @CNN and many others refused to show the massive crowd at the arena yesterday in Oklahoma. Dishonest reporting!	2016-01-21 14:04:54	2016-01-21 09:04:54-05:00	9.081667	2016.0
690176882055114758	2266	Twitter for Android	Sad sack @JebBush has just done another ad on me, with special interest money, saying I won't beat Hillary - I WILL. But he can't beat me.	2016-01-21 14:19:26	2016-01-21 09:19:26-05:00	9.323889	2016.0

In [41]:
```
android_table_year = trump[trump["source"] == "Twitter for Android"][["source", 'hour', 'year']]
android_table_2016 = android_table_year[android_table_year['year'] == 2016][["source", 'hour']]
iphone_table_year = trump[trump["source"] == "Twitter for iPhone"][["source", 'hour', 'year']]
iphone_table_2016 = iphone_table_year[iphone_table_year['year'] == 2016][["source", "hour"]]
```

In [42]:
```
### make your plot here
sns.distplot(iphone_table_2016['hour'], hist = False, label = 'iPhone')
sns.distplot(android_table_2016["hour"], hist = False, label = 'Android')
plt.xlabel('hour')
plt.ylabel('fraction')
plt.title("Proportion of tweets per hour with Android or iPhone 2016");
#raise NotImplementedError()
```



By comparing the two graphs: the one for android and iphone tweets for 2016 and the one for any date provided, we can assume that Trump made his own tweets on the Android. One such reason for this stems from the similarities we see in the iphone curves for both graphs, which appear to resemble each other as far as the pattern. This uncanny structure causes one to assume that the tweets for the iPhone were rather systematic, meaning that they were likely planned. For the Android tweers, we see that there was a spike in tweets in 2016 relative to the other graph and the graph does not follow that of the iPhone pattern, which suggests that it may correspond with the tweeting nature of a very volatile Donald Trump. Further, we see that in the graph corresponding to all of the data the plot for Android almost mirrors that of iPhone suggesting that apart from the 2016 election season the tweeting is more planned out.

Question 5

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

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

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

In [43]:
```
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

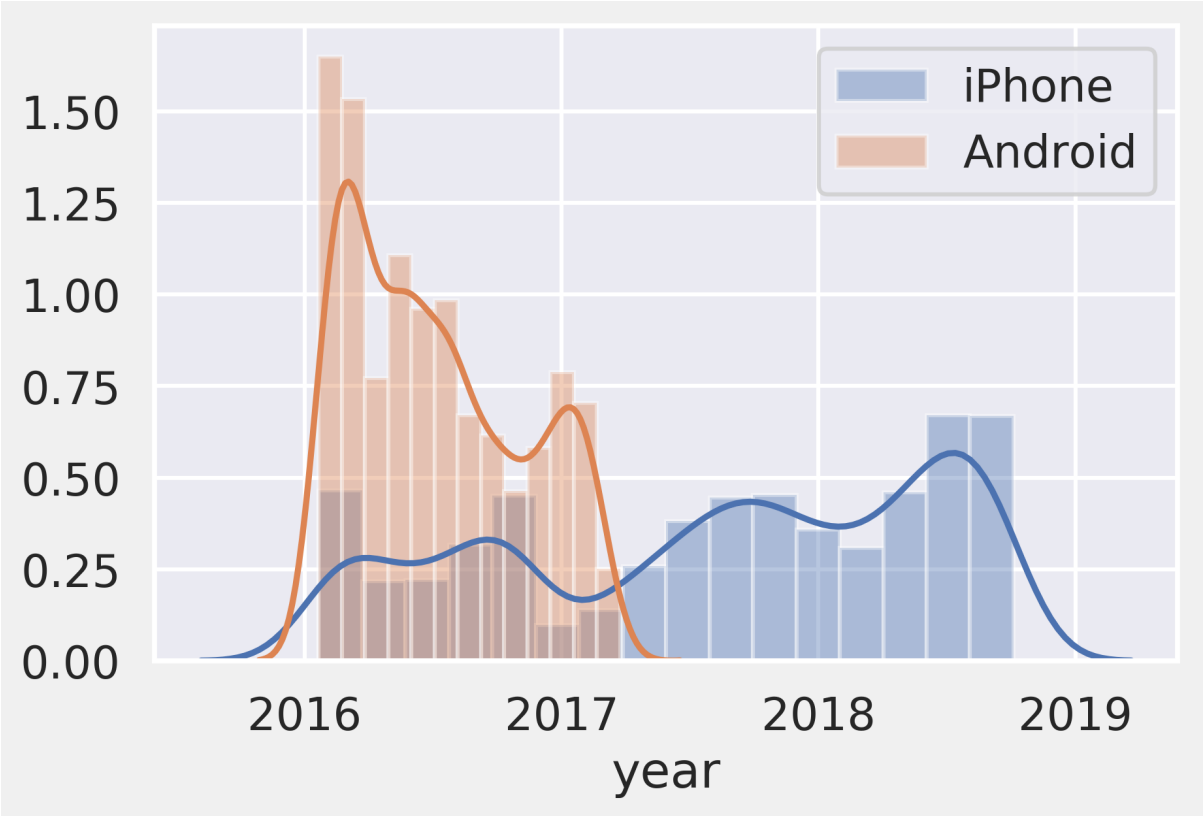

trump['year'] = trump['time'].apply(year_fraction)
```

In [ ]:
```

```

Question 5a

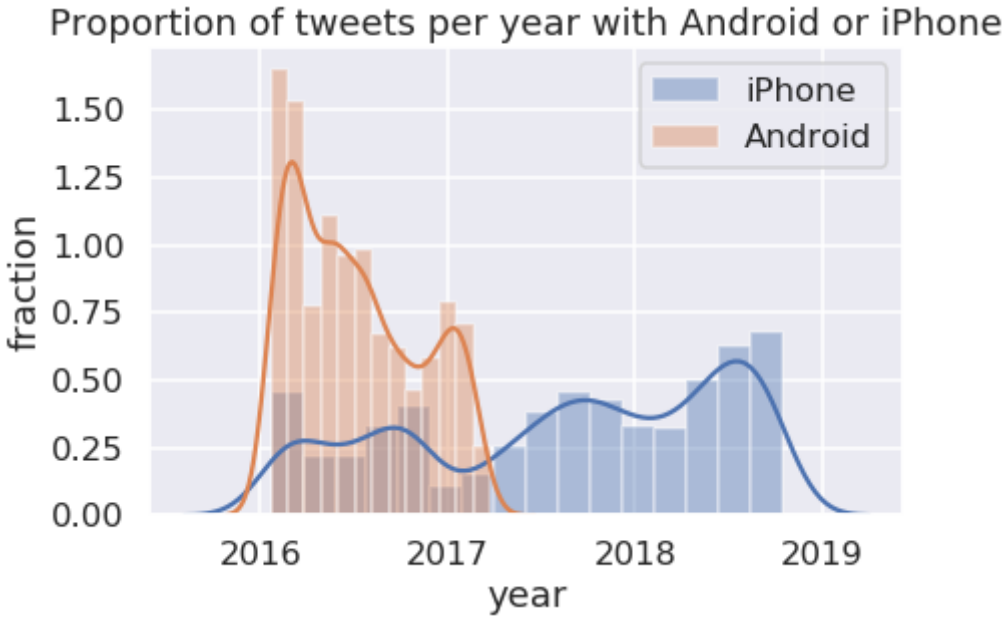
Use the `sns.distplot` to overlay the distributions of the 2 most frequently used web technologies over the years. Your final plot should look like:



In [44]:
```
android_table = trump[trump["source"] == "Twitter for Android"][["source", 'year']]
iphone_table = trump[trump["source"] == "Twitter for iPhone"][["source", 'year']]
iphone_table_merged = iphone_table.merge(android_table, how = 'outer')
```

In [45]:
```
# YOUR CODE HERE
sns.distplot(iphone_table['year'], label = "iPhone")
sns.distplot(android_table["year"], label = "Android")
plt.xlabel('year')
plt.ylabel('fraction')
plt.title("Proportion of tweets per year with Android or iPhone")
plt.legend();

#raise NotImplementedError()
```



Question 6: Sentiment Analysis

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

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

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

In [46]:
```
print(''.join(open("vader_lexicon.txt").readlines()[:10]))
```

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

Question 6a

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

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

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

In [47]:
```
sent = pd.read_csv("vader_lexicon.txt", sep ='\t', header = None)
sent = sent.set_index(0)
sent.rename(columns = {1: 'polarity'}, inplace = True)
sent = sent[["polarity"]]
sent
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[47]:

0	polarity
$:	-1.5
%)	-0.4
%-)	-1.5
&-:	-0.4
&:	-0.7
('}{')	1.6
(%	-0.9
('-:	2.2
(':	2.3
((-:	2.1
(*	1.1
(-%	-0.7
(-*	1.3
(-:	1.6
(-:0	2.8
(-:<	-0.4
(-:o	1.5
(-:O	1.5
(-:{	-0.1
(-:|>*	1.9

0	polarity
(-;	1.3
(-;|	2.1
(8	2.6
(:	2.2
(:0	2.4
(:<	-0.2
(:o	2.5
(:O	2.5
(;	1.1
(;<	0.3
...	...
xd	2.8
xp	1.6
yay	2.4
yeah	1.2
yearning	0.5
yeees	1.7
yep	1.2
yes	1.7
youthful	1.3
yucky	-1.8
yummy	2.4
zealot	-1.9
zealots	-0.8
zealous	0.5

	polarity
0	
{:	1.8
|-0	-1.2
|-:	-0.8
|-:>	-1.6
|-o	-1.2
|:	-0.5
|;-)	2.2
|=	-0.4
|^:	-1.1
|o:	-0.9
||-:	-2.3
}:	-2.1
}:(	-2.0
}:)	0.4
}:-(	-2.1
}:-)	0.3

7517 rows × 1 columns

In [48]:
```
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])
```

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 [49]:
```
# YOUR CODE HERE
def lower_new(text):
    return text.lower()
trump['text'] = trump['text'].apply(lower_new)
#raise NotImplementedError()
```

In [50]:
```
assert trump['text'].loc[884740553040175104] == 'working hard to get the olympics for the united states
```

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 [51]:
```
# Save your regex in punct_re
punct_re = r'[^\w\s]'
trump['no_punc'] = trump['text'].str.replace(punct_re, ' ')
trump
# YOUR CODE HERE
#raise NotImplementedError()
```

690382564494839809	2187	Twitter for iPhone	national review is a failing publication that has lost it's way. it's circulation is way down w its influence being at an all time low. sad!	2016-01-22 03:56:44	2016-01-21 22:56:44-05:00	22.945556	2016.057377	national rev failing publicat has lost it s circulation is wa w its influence an all time
690382619213742082	1817	Twitter for iPhone	very few people read the national review because it only knows how to criticize, but not how to lead.	2016-01-22 03:56:57	2016-01-21 22:56:57-05:00	22.949167	2016.057377	very few peo the nationa because it onl how to criticize how
690382722162913280	2236	Twitter for iPhone	the late, great, william f. buckley would be ashamed of what had happened to his prize, the dying national review!	2016-01-22 03:57:22	2016-01-21 22:57:22-05:00	22.956111	2016.057377	the late great buckley w ashamed of w happened to the dying
690404308010057728	9144	Twitter for iPhone	rt @williebosshog: make america great again! #trump2016 https://t.co/1h5j4dzdgy	2016-01-22 05:23:08	2016-01-22 00:23:08-05:00	0.385556	2016.057377	rt williebossho america gre trump2016 ht 1h5

In [52]:
```
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
assert trump['no_punc'].loc[894620077634592769] == 'on  purpleheartday i thank all the brave men and wo
# 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
```

Question 6d:

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

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

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

	num	word
894661651760377856	0	i
894661651760377856	1	think
894661651760377856	2	senator
894661651760377856	3	blumenthal
894661651760377856	4	should

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

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 [53]:
```
#now for the formatting of the data table
tidy_format = trump['no_punc'].str.split(expand = True).stack().reset_index(level
            = 1).rename(columns = {'level_1': 'num', 0: 'word'})
tidy_format
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[53]:

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

id	num	word
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
...	...	...
1052219253384994816	37	is
1052219253384994816	38	still
1052219253384994816	39	working
1052219253384994816	40	for
1052219253384994816	41	the
1052219253384994816	42	department
1052219253384994816	43	of
1052219253384994816	44	justice
1052219253384994816	45	can
1052219253384994816	46	this
1052219253384994816	47	really
1052219253384994816	48	be
1052219253384994816	49	so
1052232230972678145	0	rt

id	num	word
1052232230972678145	1	whitehouse
1052232230972678145	2	https
1052232230972678145	3	t
1052232230972678145	4	co
1052232230972678145	5	rnqlpots3o
1052233253040640001	0	register
1052233253040640001	1	to
1052233253040640001	2	https
1052233253040640001	3	t
1052233253040640001	4	co
1052233253040640001	5	0pwiwchgbh
1052233253040640001	6	maga
1052233253040640001	7	https
1052233253040640001	8	t
1052233253040640001	9	co
1052233253040640001	10	actme53tzu

217006 rows × 2 columns

In [54]:
```
assert tidy_format.loc[894661651760377856].shape == (27, 2)
assert ' '.join(list(tidy_format.loc[894661651760377856]['word'])) == 'i think senator blumenthal should
```

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 [55]:
```
trump['polarity'] = sent.merge(tidy_format, how = 'right', left_index = True,
                    right_on = "word").groupby('id').sum()['polarity']


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

In [56]:
```
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 [57]:
```
print('Most negative tweets:')
for t in trump.sort_values('polarity').head()['text']:
    print('\n  ', t)
```

```
Most negative tweets:

   it is outrageous that poisonous synthetic heroin fentanyl comes pouring into the u.s. postal system from china. we can, and must, end this now! the senate should pass the stop act — and firmly stop this poison from killing our children and destroying our country. no more delay!

   the rigged russian witch hunt goes on and on as the “originators and founders” of this scam continue to be fired and demoted for their corrupt and illegal activity. all credibility is gone from this terrible hoax, and much more will be lost as it proceeds. no collusion!

   james comey is a proven leaker &amp;amp; liar. virtually everyone in washington thought he should be fired for the terrible job he did-until he was, in fact, fired. he leaked classified information, for which he should be prosecuted. he lied to congress under oath. he is a weak and.....

   this is an illegally brought rigged witch hunt run by people who are totally corrupt and/or conflicted. it was started and paid for by crooked hillary and the democrats. phony dossier, fisa disgrace and so many lying and dishonest people already fired. 17 angry dems? stay tuned!

   where’s the collusion? they made up a phony crime called collusion, and when there was no collusion they say there was obstruction (of a phony crime that never existed). if you fight back or say anything bad about the rigged witch hunt, they scream obstruction!
```

In [58]:
```
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 amazi
ng win at doral 5 years ago, people saw his great talent, and a bright future ahead. now he is the mas
ters champion!

   my supporters are the smartest, strongest, most hard working and most loyal that we have seen in ou
r countries history. it is a beautiful thing to watch as we win elections and gather support from all
over the country. as we get stronger, so does our country. best numbers ever!

   thank you to all of my great supporters, really big progress being made. other countries wanting to
fix crazy trade deals. economy is roaring. supreme court pick getting great reviews. new poll says tru
mp, at over 90%, is the most popular republican in history of the party. wow!

   thank you, @wvgovernor jim justice, for that warm introduction. tonight, it was my great honor to a
ttend the “greenbrier classic – salute to service dinner” in west virginia! god bless our veterans. go
d bless america - and happy independence day to all! https://t.co/v35qvcn8m6 (https://t.co/v35qvcn8m6)

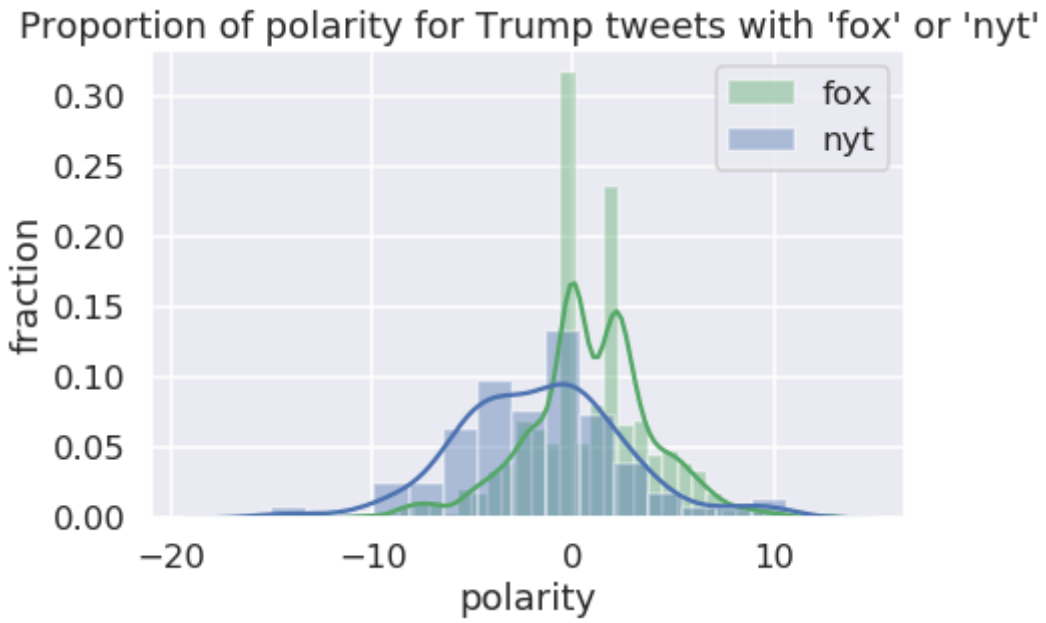
   the republican party had a great night. tremendous voter energy and excitement, and all candidates
 are those who have a great chance of winning in november. the economy is sooo strong, and with nancy
 pelosi wanting to end the big tax cuts and raise taxes, why wouldn’t we win?
```

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 [59]:
```
# YOUR CODE HERE
fox = trump.loc[trump['no_punc'].str.contains(r'fox') , :][['polarity']]
nyt = trump.loc[trump['no_punc'].str.contains(r'nyt') , :][['polarity']]
sns.distplot(fox['polarity'], color = 'g', label = 'fox')
sns.distplot(nyt['polarity'], label = 'nyt')
plt.xlabel('polarity')
plt.ylabel('fraction')
plt.title("Proportion of polarity for Trump tweets with 'fox' or 'nyt'")
plt.legend();

#sns.distplot()
#raise NotImplementedError()
```



Comment on what you observe:

We can see from the above plot that the distribution for polarity involving nyt is more shifted to the left than that of fox. This suggests that when Trump discusses fox he puts more positive words around the term 'fox' overall and when Trump discusses nyt he uses more negative words around 'nyt'. Additionally we see that the tweets with fox have less of a range of polarity, which implies that there are

limited situations where the sentiment toward fox is very negative or for that matter very postitive. However, with regards to the range of polarity for nyt, we can see that the polarity is less predictable as the sentiment could be neutral, very negative, or somewhat positive as it catches more of the extremes of the graph than tweets with 'fox'.

Question 7: Engagement

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:

	retweet_count
word	
jong	40675.666667
try	33937.800000
kim	32849.595745
un	32741.731707
maybe	30473.192308

In [60]:
```
top_20 = tidy_format.join(trump['retweet_count']).loc[:,
        ['word','retweet_count']].groupby('word').filter(lambda count: len(count) >=
        25).groupby('word').mean().sort_values('retweet_count', ascending = False).head(20)
top_20


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

Out[60]:

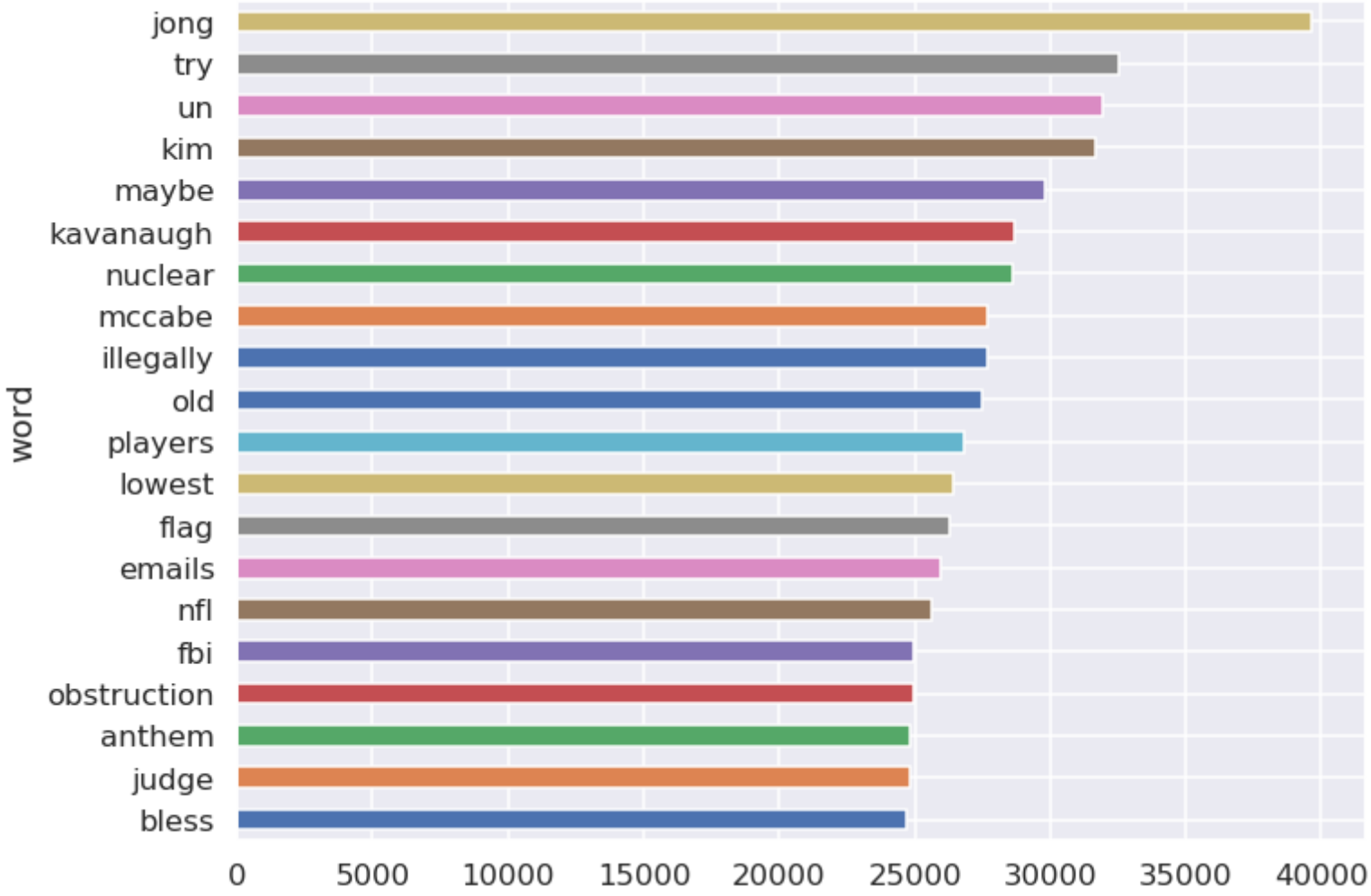
word	retweet_count
jong	39630.933333
try	32529.560000
un	31940.121951
kim	31648.387755
maybe	29837.076923
kavanaugh	28651.962963
nuclear	28592.080000
mccabe	27669.709677
illegally	27661.322581
old	27510.303030
players	26798.000000
lowest	26453.086957
flag	26286.433333
emails	25986.509434
nfl	25623.243243
fbi	24980.320000
obstruction	24951.324324
anthem	24834.189189
judge	24811.285714

word	retweet_count
bless	24723.125000

In [61]:
```
# 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 [62]:
```
top_20['retweet_count'].sort_values().plot.barh(figsize=(10, 8));
```



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.. True, we did not remove the unhelpful words from the list which is especially evident with 'try' as a word in the top 5. However, although there are some useless words that appear to be popular for Trump, this is not a confounding factor that could have made kim jong un tweets higher on the list. The reason for this is because had we removed these words from the list it would not alter the fact that kim jong un (which I am treating as nouns here as they are part of a name) would have been 3 of the words in the top 5 regardless.

2.. False, we removed Hashtags from our text as the creation of tidy_format makes use of the 'no_punc' column of the Trump dataframe. When the 'no_punc' column was created we were able to remove all hashtags as well as other punctuation marks. Therefore, this is not a confounding factor.

3.. True, we did not account for the fact that Trump's follower count has increased overtime, and likely very much so over the last three years, which include his election and presidency. This poses a potential issue as a confounding factor because tweets incolving kim jong un depict a foreign policy issue of importance that the president is discussing via twitter and therefore the American populus rationally

wants to follow. Had Trump been discussing Kim Jong Un 5 years ago, then his voice on the matter would carry significantly less attention and weight and these tweets back then would not likely be some of his more popular. i.e. who cares about a non-presidential candidate and non-president for that matter discussing kim jong un on twitter?

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.

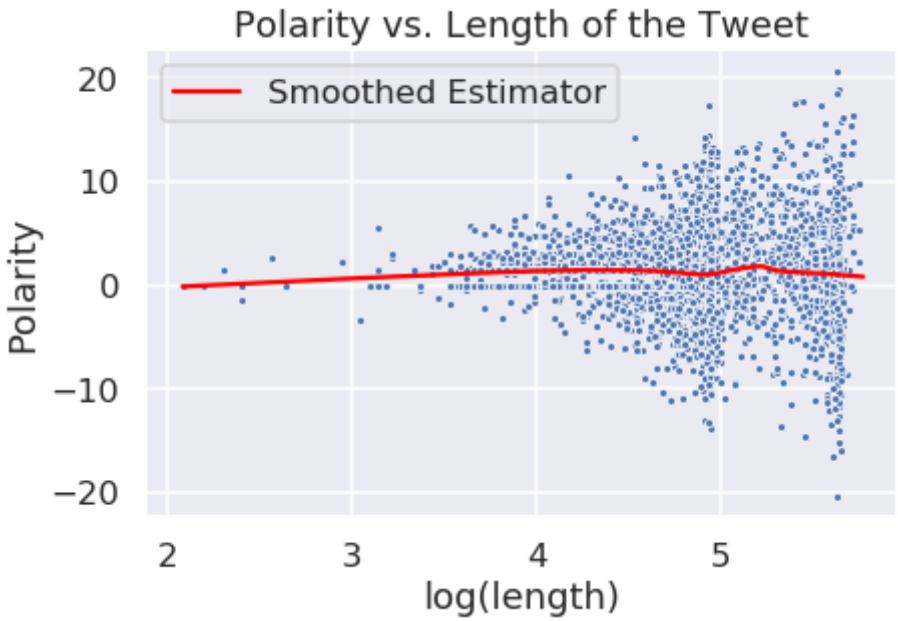
Plot:

In [ ]:
```
trump
```

In [63]:
```
# YOUR CODE HERE
#How has the sentiment changed with the length of the tweet?
from statsmodels.nonparametric.smoothers_lowess import lowess
print("avg polarity: " + str(trump['polarity'].mean()))
total_length = 0
len_texts = []
for i in trump['no_punc']:
    total_length += len(i)
    len_texts.append(len(i))
print("avg length (by characters): " + str(total_length/len(trump)))
#knowing these average values we can check to see if the average polarity is lengthier tweets
avg_polarity = trump['polarity'].mean()
trump['length'] = np.log(len_texts)
random_sample = trump.sample(5000)
ysmooth = lowess(trump['polarity'], trump['length'], return_sorted=False)
sns.lineplot(trump['length'], ysmooth, label="Smoothed Estimator", color='red')
sns.scatterplot('length', 'polarity', data = random_sample, s = 15)
plt.xlabel('log(length)')
plt.ylabel('Polarity')
plt.title('Polarity vs. Length of the Tweet');

#raise NotImplementedError()
```

```
avg polarity: 1.28366718028
avg length (by characters): 140.59267004182257
```



Discussion of Your Plot:

How has sentiment changed with the length of the tweet? From the above scatterplot, we are able to see that as the length of the tweet increases so does the variability in polarity for the tweet. This is especially evident by analyzing the greater range of polarity in dots as we move right along the x axis of the graph. Therefore, although there is roughly an even distribution of positive and negative tweets at larger lengths, we can see that sentiment is more likely to be in a larger range on the y-axis. i.e. at 4 on the x-axis we would expect values of (-5, 8) on the polarity versus at 5 on the x-axis we would expect values of (-13, 18) on the y-axis. When examining the data, I concluded that this visual trend in polarity would be easier to observe by taking the log(length) on the x-axis in order to lessen the x values. Additionally, in order to help alleviate some overplotting problems, I sought to lessen the size of each individual point and to take a random sample without replacement of 5000 points rather than the entire amount in order to make each point more distinct. We know that taking a random sample of a large enough quantity of points will allow for the same trend in the data to be observable. Lastly, as a means to further identify the trend of the data, I added a loess line to help see if there were substantially more positive points than negative points at larger lengths, and to thus prove whether overplotting is a disruptive factor in the graph. Due to the relative constant nature of the loess line around 0, we can assume that this lack of a trend proves that there is not that much more of a likelihood of a longer tweet being positive as opposed to negative.

Submission

Congrats, you just finished Project 1!

In [ ]:

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