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 = "Junsheng Pei"
          COLLABORATORS = ""
```

Project 1: Trump, Twitter, and Text

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

The project is due 11:59pm Tuesday, October 23, 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

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

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

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

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

```
Your username is: PeterPei14
```

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])
```

```
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],
                                 '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/1052351185112506368/photo/1',
                                             '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/1052351185112506368/photo/1',
                                                      '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'}]},
                      '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</a>',
                                    'truncated': False,
                                    'user': {'contributors_enabled': False,
                                             'created_at': 'Wed Mar 09 06:13:42 +0000 2011',
                                             'default_profile': False,
                                             'default_profile_image': False,
                                             'description': 'The UC Berkeley School of '
                                                            'Information is a '
                                                            'multi-disciplinary program '
                                                            'devoted to enhancing the '
                                                            'accessibility, usability, '
                                                            'credibility & security of '
                                                            'information.',
                                             'entities': {'description': {'urls': []},
                                                          'url': {'urls': [{'display_url': 'ischool.berkeley.edu',
                                                                            'expanded_url': 'http://ischool.berkeley.edu',
                                                                            'indices': [0,
                                                                                        23],
                                                                            'url': 'https://t.co/5eXJ0wNlJd'}]}},
                                             'favourites_count': 3224,
                                             'follow_request_sent': False,
                                             'followers_count': 4982,
                                             'following': False,
                                             'friends_count': 650,
                                             '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/bg.png',
                                             '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/eAmDOL2o_normal.jpg',
                                             'profile_image_url_https': 'https://pbs.twimg.com/profile_images/971975267764789254/eAmDOL2o_normal.jpg',
                                             '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': 3950,
                                             'time_zone': None,
                                             'translator_type': 'none',
                                             'url': 'https://t.co/5eXJ0wNlJd',
                                             'utc_offset': None,
                                             'verified': False}},
                   'source': '<a href="http://twitter.com" rel="nofollow">Twitter Web Client</a>',
                   'truncated': False,
                   'user': {'contributors_enabled': False,
                            'created_at': 'Thu Feb 28 14:37:26 +0000 2013',
                            'default_profile': False,
                            'default_profile_image': False,
                            'description': 'An online Master of Information and Data Science '
                                           '(MIDS) degree from the UC Berkeley School of '
                                           'Information. Learn more at: http://t.co/zf6gfBWovQ',
                            'entities': {'description': {'urls': [{'display_url': 'bit.ly/tBerkeleyData',
                                                                   'expanded_url': 'http://bit.ly/tBerkeleyData',
                                                                   'indices': [122, 144],
                                                                   'url': 'http://t.co/zf6gfBWovQ'}]},
                                         'url': {'urls': [{'display_url': 'datascience.berkeley.edu',
                                                           'expanded_url': 'http://datascience.berkeley.edu',
                                                           'indices': [0, 22],
                                                           'url': 'http://t.co/S79Ul3oCaa'}]}},
                            'favourites_count': 172,
                            'follow_request_sent': False,
                            'followers_count': 11761,
                            '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_normal.jpg',
                            '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}}
```

```
In [8]:  len(example_tweets)
```

```
Out[8]:  2406
```

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.

```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."""
            with open(path) as f:
                keys = json.load(f)
            return keys

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

```python
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
             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)
             except TweepError as e:
                 logging.warning("There was a Tweepy error. Double check your API keys and try again.")
                 logging.warning(e)
             example_tweets = [t._json for t in tweepy.Cursor(api.user_timeline, id=user_account_name,
                                                             tweet_mode='extended').items()]

             return example_tweets


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

```python
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"""
             if not Path(path).is_file():
                 # Saving the tweets to a json file on disk for future analysis
                 with open(path, "w") as f:
                     json.dump(tweets, f)

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

```python
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
             # Re-loading the json file:
             with open(path, "r") as f:
                 example_tweets = json.load(f)
             return example_tweets
             #raise NotImplementedError()
```

```
In [13]: 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.
             """
             keys = load_keys(keys_path)
             ds_tweets_save_path = "TrumpData_recent_tweets.json"
             if not Path(ds_tweets_save_path).is_file():
                 example_tweets = download_recent_tweets_by_user(user_account_name, keys)
                 # 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:
             else :
                 with open(ds_tweets_save_path, "r") as f:
                     example_tweets = json.load(f)
             return example_tweets

             # 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 [14]: # 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: 3229
```

```
In [15]: assert 2000 <= len(trump_tweets) <= 4000
```

Question 2b

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

```
In [16]: # Enter the number of the month of the oldest tweet (e.g. 1 for January)
         oldest_month = 10

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

Question 3

IMPORTANT! PLEASE READ

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

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

```
In [17]: # 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 02:30:28 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 [18]: 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 [19]: 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/XqMoixLm83FzkAbn.jpg',
                        'media_url_https': 'https://pbs.twimg.com/ext_tw_video_thumb/786204885318561792/pu/img/XqMoixLm83FzkAbn.jpg',
                        'sizes': {'large': {'h': 576,
                                            'resize': 'fit',
                                            'w': 1024},
                                  'medium': {'h': 338,
                                             'resize': 'fit',
                                             'w': 600},
                                  'small': {'h': 191,
                                            'resize': 'fit',
                                            'w': 340},
                                  'thumb': {'h': 150,
                                            'resize': 'crop',
                                            'w': 150}},
                        'type': 'photo',
                        'url': 'https://t.co/wjsl8ITVvk'}],
             'symbols': [],
             'urls': [],
             'user_mentions': []},
 'extended_entities': {'media': [{'additional_media_info': {'monetizable': False},
                                  'display_url': 'pic.twitter.com/wjsl8ITVvk',
                                  'expanded_url': 'https://twitter.com/realDonaldTrump/status/786204978629185536/video/1',
                                  'id': 786204885318561792,
                                  'id_str': '786204885318561792',
                                  'indices': [39, 62],
                                  'media_url': 'http://pbs.twimg.com/ext_tw_video_thumb/786204885318561792/pu/img/XqMoixLm83FzkAbn.jpg',
                                  'media_url_https': 'https://pbs.twimg.com/ext_tw_video_thumb/786204885318561792/pu/img/XqMoixLm83FzkAbn.jpg',
                                  'sizes': {'large': {'h': 576,
                                                      'resize': 'fit',
                                                      'w': 1024},
                                            'medium': {'h': 338,
                                                       '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/786204885318561792/pu/vid/640x360/6vt24D3ZQSvYuDqe.mp4'},
                                                              {'bitrate': 2176000,
                                                               'content_type': 'video/mp4',
                                                               'url': 'https://video.twimg.com/ext_tw_video/786204885318561792/pu/vid/1280x720/rSbgQdvR9TPIlRWr.mp4'},
                                                              {'bitrate': 320000,
                                                               'content_type': 'video/mp4',
                                                               'url': 'https://video.twimg.com/ext_tw_video/786204885318561792/pu/vid/320x180/JuNJDqr1KHqoP83N.mp4'},
                                                              {'content_type': 'application/x-mpegURL',
                                                               'url': 'https://video.twimg.com/ext_tw_video/786204885318561792/pu/pl/IugUNii3a7lmjApS.m3u8'}]}}]},
 'favorite_count': 42242,
 'favorited': False,
 'geo': None,
 'id': 786204978629185536,
 'id_str': '786204978629185536',
 'in_reply_to_screen_name': None,
 'in_reply_to_status_id': None,
 'in_reply_to_status_id_str': None,
 'in_reply_to_user_id': None,
 'in_reply_to_user_id_str': None,
 'is_quote_status': False,
 'lang': 'en',
 'place': {'attributes': {},
           'bounding_box': {'coordinates': [[[-87.634643, 24.396308],
                                             [-79.974307, 24.396308],
                                             [-79.974307, 31.001056],
                                             [-87.634643, 31.001056]]],
                            'type': 'Polygon'},
           'contained_within': [],
           'country': 'United States',
           'country_code': 'US',
           'full_name': 'Florida, USA',
           'id': '4ec01c9dbc693497',
           'name': 'Florida',
           'place_type': 'admin',
           'url': 'https://api.twitter.com/1.1/geo/id/4ec01c9dbc693497.json'},
 'possibly_sensitive': False,
 'retweet_count': 24915,
 'retweeted': False,
 'source': '<a href="http://twitter.com/download/iphone" '
           'rel="nofollow">Twitter for iPhone</a>',
 'text': 'PAY TO PLAY POLITICS. \n#CrookedHillary https://t.co/wjsl8ITVvk',
 'truncated': False,
 'user': {'contributors_enabled': False,
          'created_at': 'Wed Mar 18 13:46:38 +0000 2009',
          'default_profile': False,
          'default_profile_image': False,
          'description': '45th President of the United States of America🇺🇸',
          'entities': {'description': {'urls': []}},
          'favourites_count': 12,
          'follow_request_sent': False,
          'followers_count': 35307313,
          'following': False,
          'friends_count': 45,
          'geo_enabled': True,
          'has_extended_profile': False,
          'id': 25073877,
          'id_str': '25073877',
          'is_translation_enabled': True,
          'is_translator': False,
          'lang': 'en',
          'listed_count': 74225,
          'location': 'Washington, DC',
          'name': 'Donald J. Trump',
          'notifications': False,
          'profile_background_color': '6D5C18',
          'profile_background_image_url': 'http://pbs.twimg.com/profile_background_images/530021613/trump_scotland__43_of_70_cc.jpg',
          'profile_background_image_url_https': 'https://pbs.twimg.com/profile_background_images/530021613/trump_scotland__43_of_70_cc.jpg',
          '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_normal.jpg',
                           '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 [20]: old_trump_tweets[0].keys()
```

```
Out[20]: dict_keys(['created_at', 'id', 'id_str', 'text', 'truncated', 'entities', 'extended_entities', 'source', 'in_reply_to_status_id', 'in_reply_to_status_id_str', 'in_reply_to_user_id', 'in_reply_to_user_id_str', 'in_reply_to_screen_name', 'user', 'geo', 'coordinates', 'place', 'contributors', 'is_quote_status', 'retweet_count', 'favorite_count', 'favorited', 'retweeted', 'possibly_sensitive', 'lang'])
```

```
In [21]: old_trump_tweets[3]['text']
```

```
Out[21]: 'Thank you Florida- a MOVEMENT that has never been seen before and will never be seen again. Lets get out &amp;… https://t.co/t9XM9wFDZI'
```

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!

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]: id_trump = []
         for x in trump_tweets:
             id_trump.append(x['id'])
         all_tweets = trump_tweets + [x for x in old_trump_tweets if x['id'] not in id_trump]

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

```
In [23]: 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 [24]: trump = pd.DataFrame(all_tweets,columns=['id','created_at','source','text','retweet_count','full_text'])
         for i in range(trump.shape[0]):
             if(pd.isnull(trump.loc[i,'text'])) :
                 trump.loc[i,'text'] = trump.loc[i,'full_text']
         trump = trump.drop(columns='full_text')
         trump = trump.set_index('id')
         trump.columns = ['time','source','text','retweet_count']
         trump['time'] = pd.to_datetime(trump['time'])
         trump = trump.sort_index()
         # YOUR CODE HERE
         #raise NotImplementedError()
         #trump
```

```
In [25]: 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 [26]: trump['source'].unique()
```

```
Out[26]: 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)
```
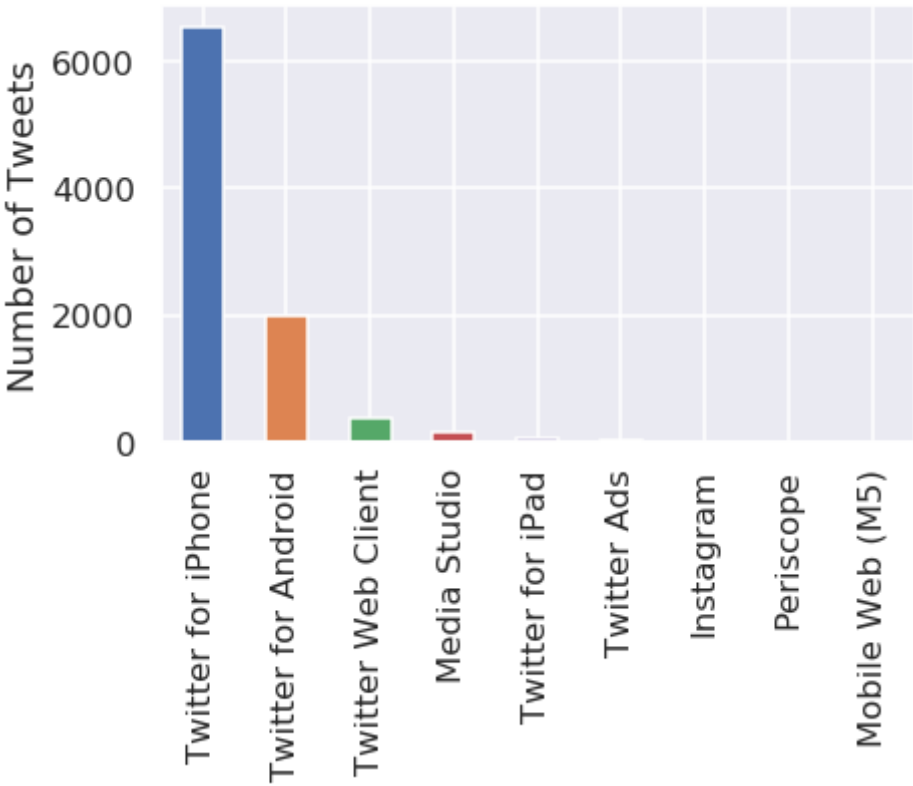
Question 4a

Remove the HTML tags from the source field.

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

```
In [27]:  ## Uncomment and complete
          trump['source'] = trump['source'].str.replace(r'<.*?>','')

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

```
In [28]:  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()) == set(['Twitter Ads',
           'Twitter Web Client',
           'Twitter for Android',
           'Twitter for iPhone'])
```

```
In [29]:  trump['source'].unique()
```

```
Out[29]:  array(['Twitter for Android', 'Twitter for iPhone', 'Twitter Web Client',
                 'Mobile Web (M5)', 'Instagram', 'Twitter for iPad', 'Media Studio',
                 'Periscope', 'Twitter Ads'], dtype=object)
```

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

```
In [30]:  trump['source'].value_counts().plot(kind="bar")
          plt.ylabel("Number of Tweets")
```

```
Out[30]:  Text(0,0.5,'Number of Tweets')
```


Question 4b

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

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

```
In [31]:  trump.head(10)
```

Out[31]:

id	time	source	text	retweet_count
690171032150237184	2016-01-21 13:56:11	Twitter for Android	"@bigop1: @realDonaldTrump @SarahPalinUSA https://t.co/3kYQGqeVyD"	1059
690171403388104704	2016-01-21 13:57:39	Twitter for Android	"@AmericanAsPie: @glennbeck @SarahPalinUSA Remember when Glenn gave out gifts to ILLEGAL ALIENS at crossing the border? Me too!"	1339
690173226341691392	2016-01-21 14:04:54	Twitter for Android	So sad that @CNN and many others refused to show the massive crowd at the arena yesterday in Oklahoma. Dishonest reporting!	2006
690176882055114758	2016-01-21 14:19:26	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.	2266
690180284189310976	2016-01-21 14:32:57	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!	2886
690271688127213568	2016-01-21 20:36:09	Twitter for iPhone	New Day on CNN treats me very badly. @AlisynCamerota is a disaster. Not going to watch anymore.	1429
690272687168458754	2016-01-21 20:40:07	Twitter for Android	Happy birthday to my friend, the great @jacknicklaus - a totally special guy!	1053
690313350278819840	2016-01-21 23:21:42	Twitter for iPhone	Thank you, Iowa! #Trump2016 https://t.co/ryhEheTLqN	2329
690315202261155840	2016-01-21 23:29:04	Twitter for iPhone	Thank you! #Trump2016 https://t.co/pcdmylO1Zt	1463
690315366564626433	2016-01-21 23:29:43	Twitter for iPhone	Thank you, New Hampshire!\n#Trump2016 https://t.co/TG9oZKly4l	1761

```
In [32]:  for t in trump_tweets[0:3]:
              print(t['created_at'])

          Mon Oct 22 01:49:29 +0000 2018
          Sun Oct 21 22:48:55 +0000 2018
          Sun Oct 21 19:26:12 +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 [33]: 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[33]:

id	time	source	text	retweet_count	est_time
690171032150237184	2016-01-21 13:56:11	Twitter for Android	"@bigop1: @realDonaldTrump @SarahPalinUSA https://t.co/3kYQGqeVyD"	1059	2016-01-21 08:56:11-05:00
690171403388104704	2016-01-21 13:57:39	Twitter for Android	"@AmericanAsPie: @glennbeck @SarahPalinUSA Remember when Glenn gave out gifts to ILLEGAL ALIENS at crossing the border? Me too!"	1339	2016-01-21 08:57:39-05:00
690173226341691392	2016-01-21 14:04:54	Twitter for Android	So sad that @CNN and many others refused to show the massive crowd at the arena yesterday in Oklahoma. Dishonest reporting!	2006	2016-01-21 09:04:54-05:00
690176882055114758	2016-01-21 14:19:26	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.	2266	2016-01-21 09:19:26-05:00
690180284189310976	2016-01-21 14:32:57	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!	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:

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

```
In [34]: hour = [x.hour for x in trump['est_time']]
         minute = [x.minute for x in trump['est_time']]
         second = [x.second for x in trump['est_time']]
         trump['hour'] = np.array(hour) + np.array(minute)/60.0 + np.array(second)/60.0/60.0

         # make a bar plot here
         # YOUR CODE HERE
         #raise NotImplementedError()
         #trump['hour']
```

```
In [35]: 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.
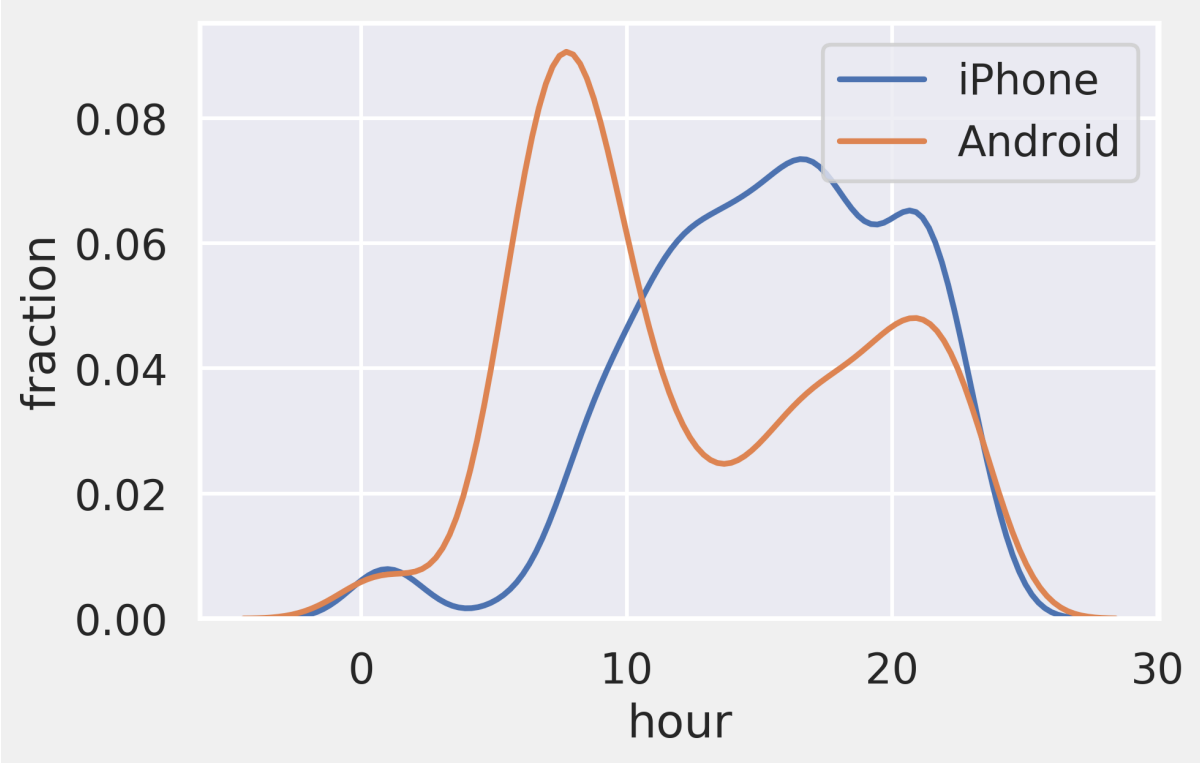

```
In [36]: ### make your plot here
         iphone = trump[trump['source'] == 'Twitter for iPhone']['hour']
         android = trump[trump['source'] == 'Twitter for Android']['hour']
         plt.figure(figsize = (8,6))
         sns.distplot(iphone, hist = False, kde = True,label='iPhone')
         sns.distplot(android, hist = False, kde = True,label='Android')
         plt.xlabel('hour')
         plt.ylabel('fraction')
         plt.title("the distribution over hours that Trump tweets on iPhone and Anroid")
         plt.legend()
         # YOUR CODE HERE
         #raise NotImplementedError()
```

Out[36]: <matplotlib.legend.Legend at 0x7f6dedfc5c88>
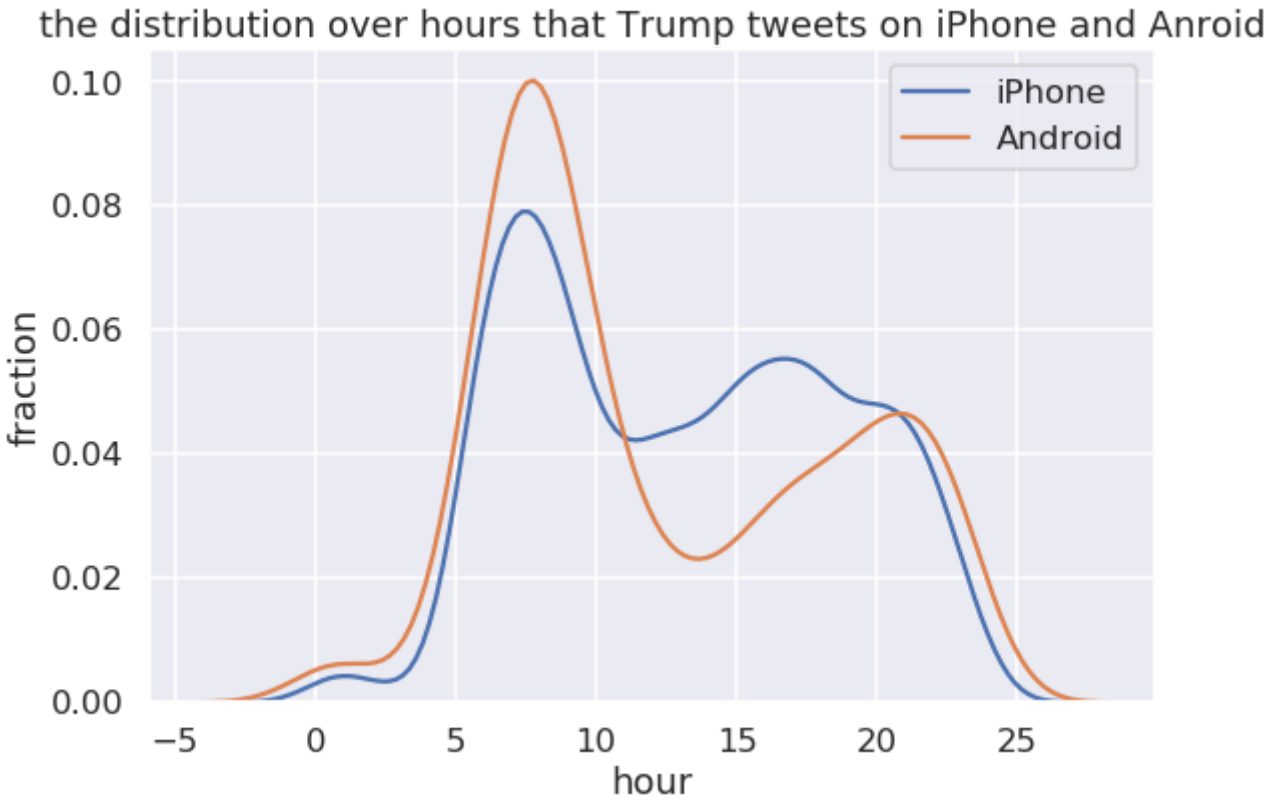
Question 4d

According to this Verge article (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.

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 [37]: ### make your plot here
         # YOUR CODE HERE
         iphone_2016 = trump[(trump['source'] == 'Twitter for iPhone') & (trump['est_time'].dt.year == 2016)]['hour']
         android_2016 = trump[(trump['source'] == 'Twitter for Android') & (trump['est_time'].dt.year == 2016)]['hour']
         plt.figure(figsize = (8,6))
         sns.distplot(iphone_2016, hist = False, kde = True,label='iPhone')
         sns.distplot(android_2016, hist = False, kde = True,label='Android')
         plt.xlabel('hour')
         plt.ylabel('fraction')
         plt.title("The distribution over hours that Trump tweets on iPhone and Anroid in 2016")
         plt.legend()
         #raise NotImplementedError()
```

Out[37]: <matplotlib.legend.Legend at 0x7f6dedf305c0>
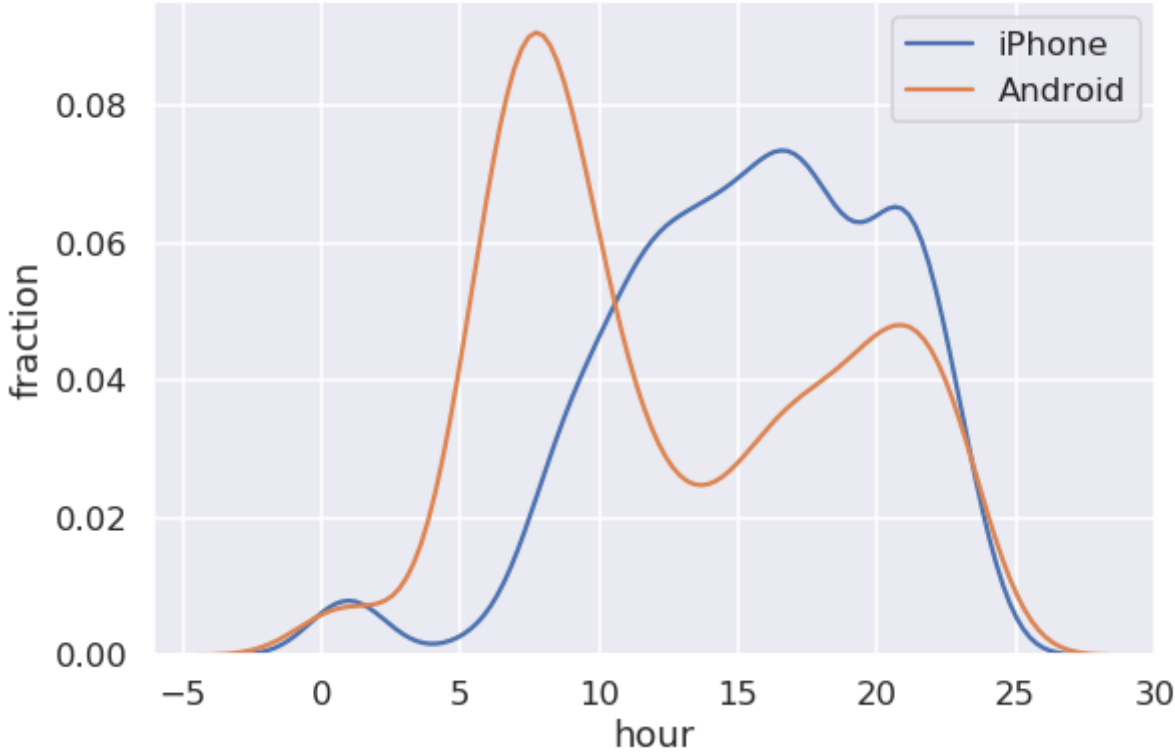
Yes, the figure gave support to the theory because most tweets from Andriod are sent in the early moring, however, most tweets from iPhone are sent in the afternoon. For Trump, he was much busier in the afternoon than in the early morning. And Trump's staff didn't work in the early morning.

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


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

Question 5a

Use the `sns.distplot` to overlay the distributions of the 2 most frequently used web technologies over the years. Your final plot should look like:
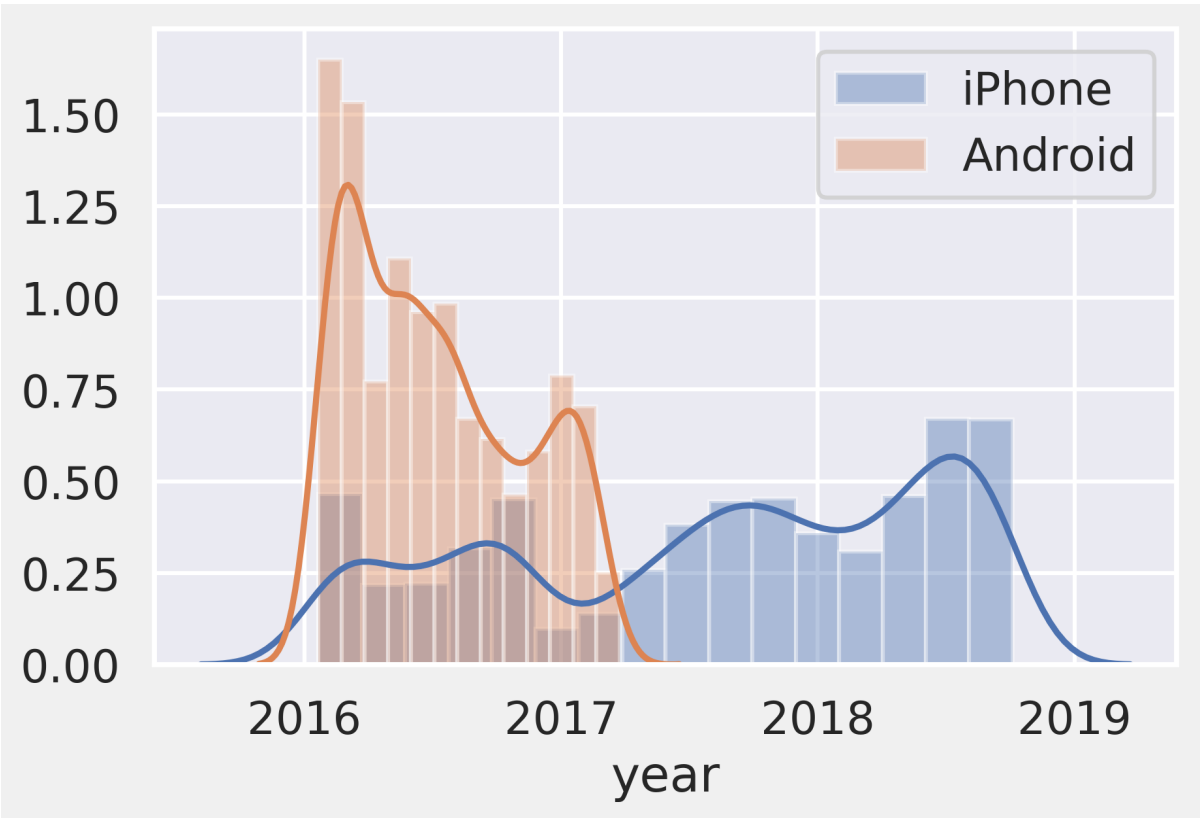
```
In [39]:  # YOUR CODE HERE
          iPhone_year = android = trump[trump['source'] == 'Twitter for iPhone']['year']
          Android_year = android = trump[trump['source'] == 'Twitter for Android']['year']
          plt.figure(figsize = (8,6))
          sns.distplot(iPhone_year, hist = True, kde = True,label='iPhone')
          sns.distplot(Android_year, hist = True, kde = True,label='Android')
          plt.xticks(np.arange(4)+2016, np.arange(4)+2016)
          plt.yticks(np.arange(7)/4, np.arange(7)/4)
          plt.xlabel('year')
          plt.title("The distributions of the 2 most frequently used web technologies over the years")
          plt.legend()
          #raise NotImplementedError()
```

```
Out[39]: <matplotlib.legend.Legend at 0x7f6dede670f0>
```


The distributions of the 2 most frequently used web technologies over the years
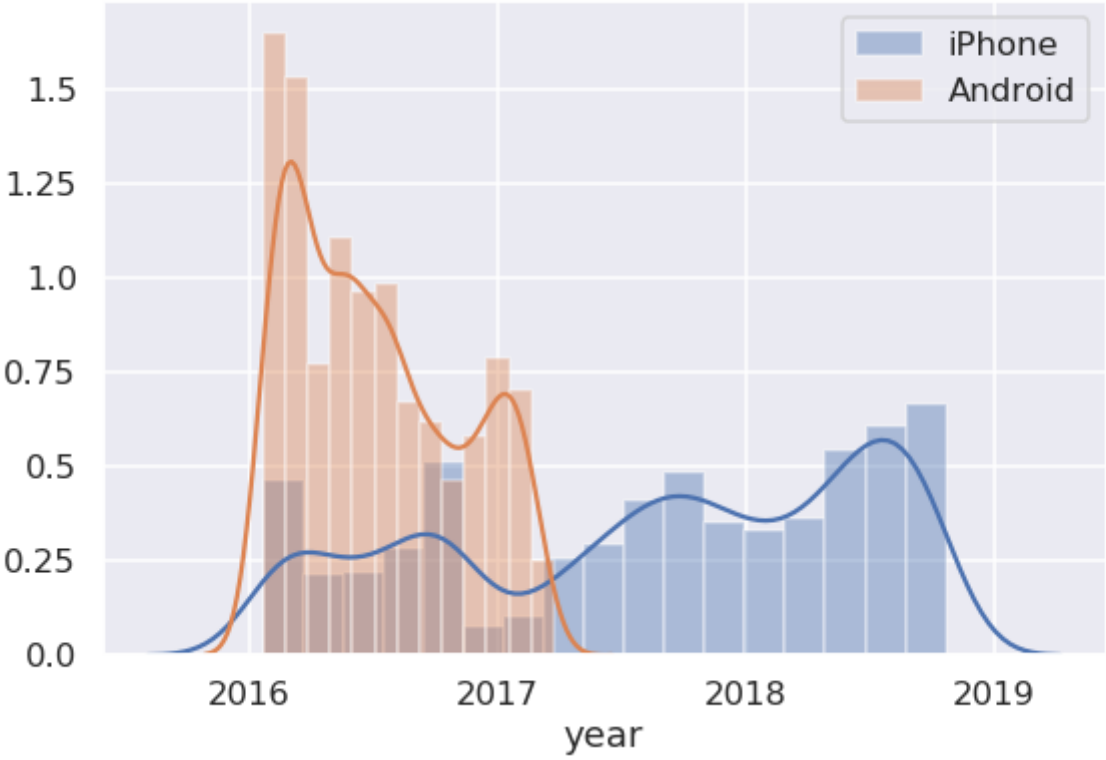
Question 6: Sentiment Analysis

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 [40]:  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 [41]:  file = open("vader_lexicon.txt")
          sent_dict = {
              'tokens': [],
              'polarity': []
          }
          line = file.readline()
          while line:
              token, polarity = line.split('\t')[0:2]
              sent_dict['tokens'].append(token)
              sent_dict['polarity'].append(polarity)
              line = file.readline()
          sent = pd.DataFrame(sent_dict, dtype = 'float')
          sent = sent.set_index('tokens')
          # YOUR CODE HERE
          #raise NotImplementedError()
```

```
In [42]:  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 [43]:  # YOUR CODE HERE
          trump['text'] = trump['text'].str.lower()
          #raise NotImplementedError()
```

```
In [44]:  assert trump['text'].loc[884740553040175104] == 'working hard to get the olympics for the united states (l.a.). stay tuned!'
```

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

Out[45]:

id	time	source	text	retweet_count	est_time	hour	year
690171032150237184	2016-01-21 13:56:11	Twitter for Android	"@bigop1: @realdonaldtrump @sarahpalinusa https://t.co/3kyqgqevyd"	1059	2016-01-21 08:56:11-05:00	8.936389	2016.054645
690171403388104704	2016-01-21 13:57:39	Twitter for Android	"@americanaspie: @glennbeck @sarahpalinusa remember when glenn gave out gifts to illegal aliens at crossing the border? me too!"	1339	2016-01-21 08:57:39-05:00	8.960833	2016.054645
690173226341691392	2016-01-21 14:04:54	Twitter for Android	so sad that @cnn and many others refused to show the massive crowd at the arena yesterday in oklahoma. dishonest reporting!	2006	2016-01-21 09:04:54-05:00	9.081667	2016.054645
690176882055114758	2016-01-21 14:19:26	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.	2266	2016-01-21 09:19:26-05:00	9.323889	2016.054645
690180284189310976	2016-01-21 14:32:57	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!	2886	2016-01-21 09:32:57-05:00	9.549167	2016.054645

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

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

```
In [47]: 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 night  it is a totally one sided  biased show   nothing funn
         y at all  equal time for us '
         assert trump['no_punc'].loc[894620077634592769] == 'on  purpleheartday i thank all the brave men and women who have sacrificed in battle for this great nation  usa
             https   t co qmfdlslp6p'
         # 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.a.). stay tuned!'
```

Question 6d:

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

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

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

	num	word
894661651760377856	0	i
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 [48]: tidy_format = trump['no_punc'].str.split(expand = True).stack().reset_index(level = 1)
         tidy_format.columns = ['num','word']
         # YOUR CODE HERE
         #raise NotImplementedError()
         #tidy_format
```

```
In [49]: assert tidy_format.loc[894661651760377856].shape == (27, 2)
         assert ' '.join(list(tidy_format.loc[894661651760377856]['word'])) == 'i think senator blumenthal should take a nice long vacation in vietnam where he lied about hi
         s service so he can at least say he was there'
```

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 [50]: merged_polarity= pd.merge(tidy_format, sent, how = 'left', left_on = 'word',right_index = True)
         trump['polarity'] = merged_polarity.groupby(merged_polarity.index)['polarity'].sum()
         # YOUR CODE HERE
         #raise NotImplementedError()
```

```
In [51]: 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 [52]: 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 & 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 [53]: 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 doral 5 years ago, people saw his great talent, and a bright future ahead. now he is the masters champion!

    my supporters are the smartest, strongest, most hard working and most loyal that we have seen in our 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 trump, 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 attend the "greenbrier classic — salute to service dinner" in west virginia! god bless our veterans. god bless america - and happy independence day to all! 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 [54]: # We only consider the words are exactly 'nyt' or 'fox', and we don't count words like anything that contasins 'nyt'
         id_nyt = []
         id_fox = []
         for tweet_id in trump.index:
             if(isinstance(tidy_format.loc[tweet_id,'word'], str)):
                 if tidy_format.loc[tweet_id,'word'] == 'nyt':
                     id_nyt.append(tweet_id)
                 if tidy_format.loc[tweet_id,'word'] == 'fox':
                     id_fox.append(tweet_id)
             else:
                 if(tidy_format.loc[tweet_id,'word'].str.contains(r'^nyt$').any()):
                     id_nyt.append(tweet_id)
                 if(tidy_format.loc[tweet_id,'word'].str.contains(r'^fox$').any()):
                     id_fox.append(tweet_id)
         plt.figure(figsize = (8,6))
         sns.distplot(trump.loc[id_nyt,'polarity'], hist = True, kde = True,label='nyt')
         sns.distplot(trump.loc[id_fox,'polarity'], hist = True, kde = True,label='fox')
         plt.xlabel('sentition_polarity')
         plt.title("The distribution of sentiments for tweets containing nyt and fox")
         plt.legend()
         # YOUR CODE HERE
         #raise NotImplementedError()
```

Out[54]: <matplotlib.legend.Legend at 0x7f6decb24860>
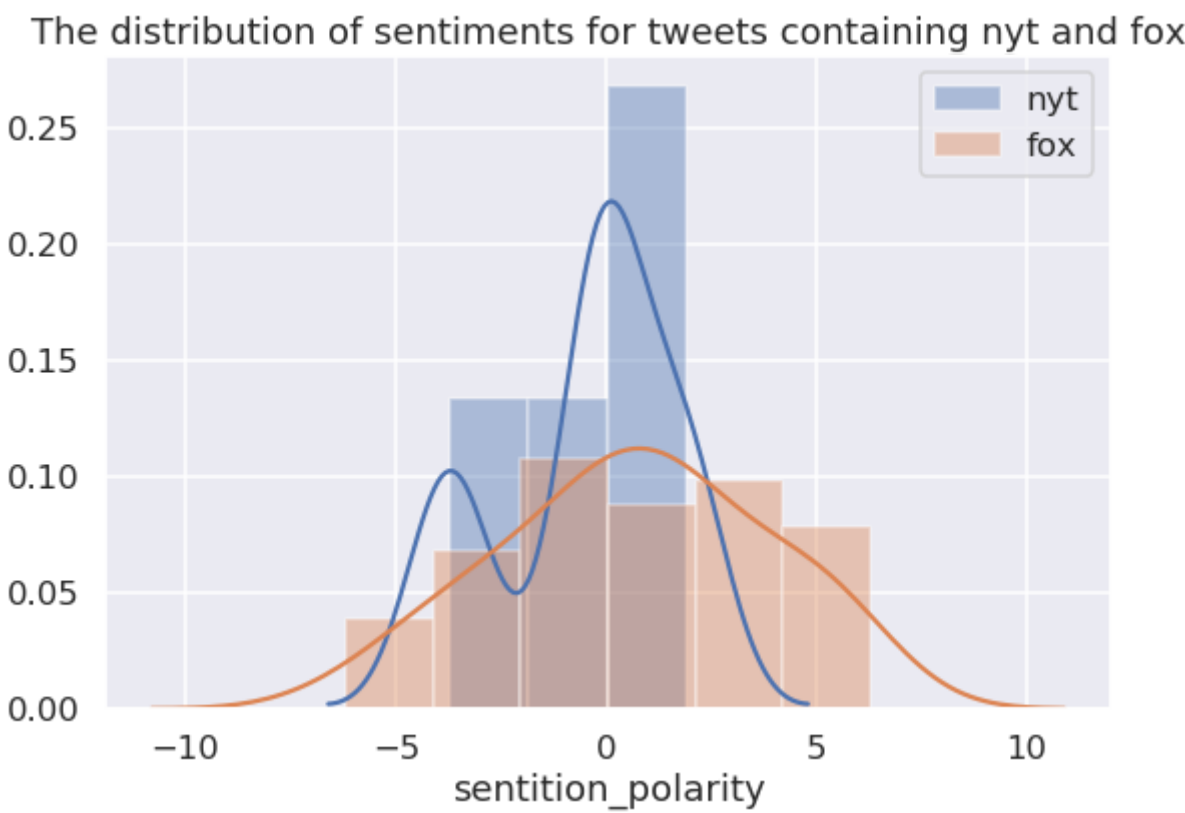
Comment on what you observe:

The majority sentiments for tweets containing nyt and fox were neutral. When Trump mentioned nyt, the tweet would be more likely to be negative. And when Trump mentioned fox the tweet would be more likely to be positive.

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 [55]: top_20 = pd.merge(tidy_format, trump, how = 'left', left_index = True,  right_index=True
                          )[['word','retweet_count']
                          ].groupby('word'
                                  ).filter(lambda x : x['word'].count()>=25
                                      ).groupby('word'
                                          ).mean().sort_values(by = 'retweet_count',ascending = False
                                              ).iloc[:20]

         # YOUR CODE HERE
         #raise NotImplementedError()
         top_20
```

Out[55]:

	retweet_count
word	
jong	39583.100000
try	32490.600000
un	31913.707317
kim	31607.836735
maybe	29803.038462
kavanaugh	28601.629630
nuclear	28556.240000
illegally	28370.303030
mccabe	27624.806452
old	27481.757576
players	26828.933333
lowest	26465.000000
flag	26325.366667
nfl	26172.805556
emails	25967.000000
judge	25125.738095
fbi	25001.265000
obstruction	24922.945946
anthem	24860.243243
bless	24715.031250

```
In [56]: # 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 [57]: top_20['retweet_count'].sort_values().plot.barh(figsize=(10, 8));
```
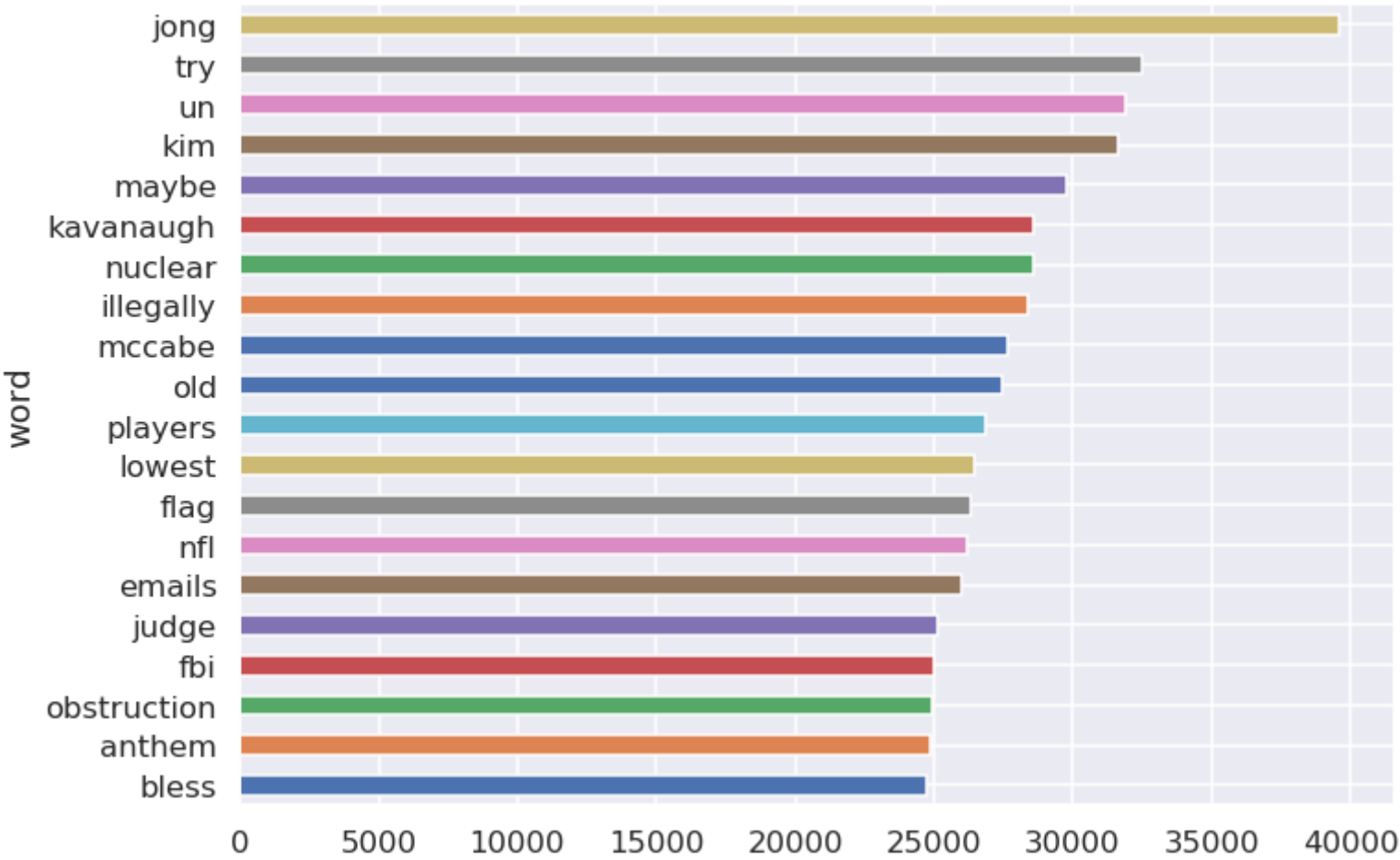
Question 7b

"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.

We can say that Trump's tweets involving jong un are really poplular, but it's hard to tell that they are the most popular.

1. The statement is true, but it couldn't make jong un related tweets higher because if we apply the retrictions, the retweet_counts of tweets related to jong un are still high
2. The statement is true, and it could have made kim jong un related tweets higher. Because if Trump had a few very popular tweets where he mentioned kim jong un many times and the total number of tweets containing kim jong un was not small, it would make the average retweet count very high.
3. The statement is true, and it could have made kim jong un related tweets higher. Because kim jong un could be a polular topic recently, and the number of follwers of Trump has increased, the recent tweet would have higher retweet count.

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 [58]:   # I would like to explore the first idea
           plt.figure(figsize = (8,6))
           plt.scatter(trump['text'].str.len(), trump['polarity'],s =5)
           plt.title('How has the sentiment changed with length of the tweets')
           plt.xlabel('length of tweets')
           plt.ylabel('sentiment')
           # YOUR CODE HERE
           #raise NotImplementedError()
```

```
Out[58]:   Text(0,0.5,'sentiment')
```
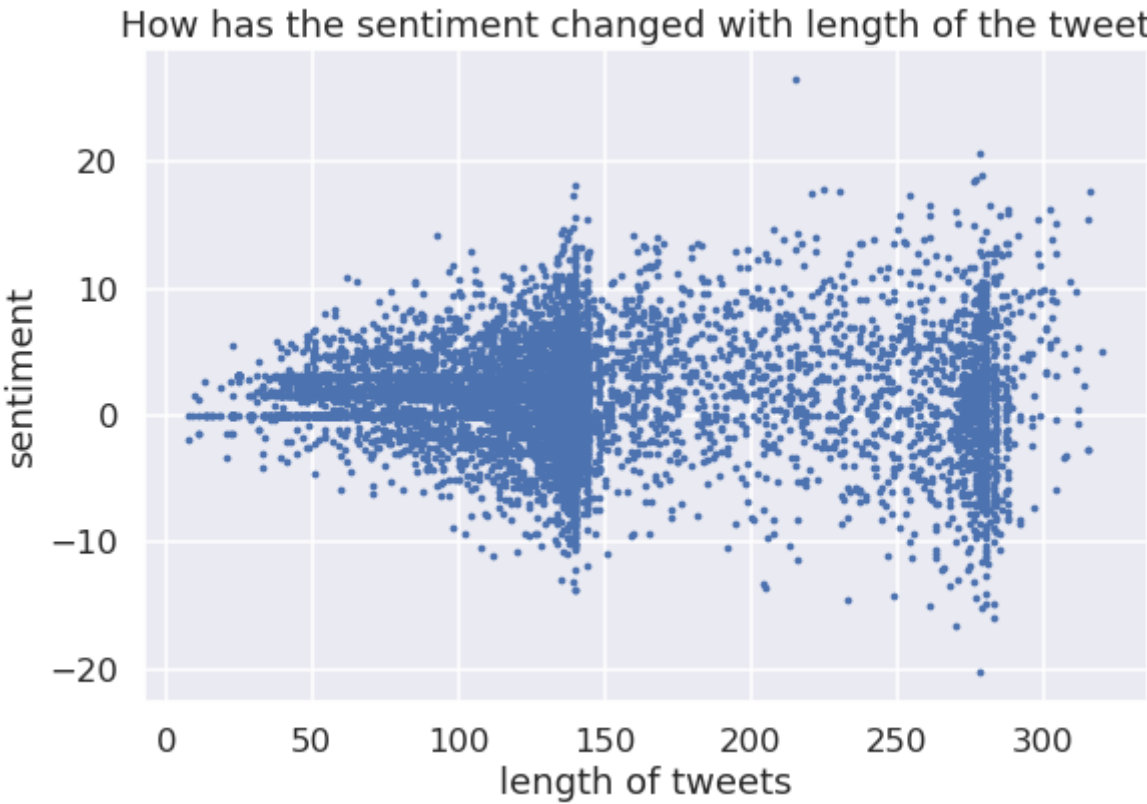

Discussion of Your Plot:

As we can see from the plot, as the length of tweets increases, the sentiment value will have a larger range. But the sentiment value doesn't have a straight relationship with the length of tweet because the plot is symmetric about the line y = 0.

Submission

Congrats, you just finished Project 1!

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