Homework 3: Pandas, Regular Expressions, Visualizations

Due Date: Fri 4/22, 11:59 pm KST

Collaboration Policy: You may talk with others about the homework, but we ask that you **write your solutions individually**. If you do discuss the assignments with others, please **include their names** in the following line.

Collaborators: *list collaborators here (if applicable)*

Score Breakdown

Question	Points
Question 1a	2
Question 1b	1
Question 1c	2
Question 2	2
Question 3	1
Question 4	2
Question 5a	1
Question 5b	2
Question 5c	2
Question 6a	1
Question 6b	1
Question 6c	1
Question 6d	2
Question 6e	2
Total	22

Initialize your environment

This cell should run without error.

```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import json
import zipfile
from pprint import pprint # to get a more easily-readable view.

# 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
```

Some common utilities.

```
def utils_head(filename, lines=5):
    """
    Returns the first few lines of a file.

    filename: the name of the file to open
    lines: the number of lines to include

    return: A list of the first few lines from the file.
    """
    from itertools import islice
    with open(filename, "r") as f:
        return list(islice(f, lines))
```

Part 1: Bike Sharing

The data we are exploring is collected from a bike sharing system in Washington D.C.

The variables in this data frame are defined as:

Variable	Description
instant	record index
dteday	date
season	1. spring 2. summer 3. fall 4. winter
yr	year (0: 2011, 1:2012)
mnth	month (1 to 12)
hr	hour (0 to 23)
holiday	whether day is holiday or not
weekday	day of the week
workingday	if day is neither weekend nor holiday
weathersit	1. clear or partly cloudy 2. mist and clouds 3. light snow or rain 4. heavy rain or snow
temp	normalized temperature in Celsius (divided by 41)
atemp	normalized "feels-like" temperature in Celsius (divided by 50)
hum	normalized percent humidity (divided by 100)
windspeed	normalized wind speed (divided by 67)

Variable	Description
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes including casual and registered

Mount your Google Drive

When you run a code cell, Colab executes it on a temporary cloud instance. Every time you open the notebook, you will be assigned a different machine. All compute state and files saved on the previous machine will be lost. Therefore, you may need to re-download datasets or rerun code after a reset. Here, you can mount your Google drive to the temporary cloud instance's local filesystem using the following code snippet and save files under the specified directory (note that you will have to provide permission every time you run this).

```
# mount Google drive
from google.colab import drive
drive.mount('/content/drive')

# now you can see files
!echo -e "₩nNumber of Google drive files in /content/drive/My Drive/:"
!ls -l "/content/drive/My Drive/" | wc -l
# by the way, you can run any linux command by putting a ! at the start of the line

# by default everything gets executed and saved in /content/
!echo -e "₩nCurrent directory:"
!pwd
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/c

Number of Google drive files in /content/drive/My Drive/:
165

Current directory:
/content
```

```
workspace_path = '/content/drive/MyDrive/hw3/'  # Change this path!
for line in utils_head(workspace_path+'bikeshare.txt'):
    print(line, end="")
```

```
instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathersit,temp,atemp,hum,windspe
1,2011-01-01,1,0,1,0,0,6,0,1,0.24,0.2879,0.81,0,3,13,16
2,2011-01-01,1,0,1,1,0,6,0,1,0.22,0.2727,0.8,0,8,32,40
3,2011-01-01,1,0,1,2,0,6,0,1,0.22,0.2727,0.8,0,5,27,32
4,2011-01-01,1,0,1,3,0,6,0,1,0.24,0.2879,0.75,0,3,10,13
```

Loading the data

The following code loads the data into a Pandas DataFrame.

```
bike = pd.read_csv(workspace_path+'bikeshare.txt')
bike.head()
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit
0	1	2011-01-01	1	0	1	0	0	6	0	1
1	2	2011-01-01	1	0	1	1	0	6	0	1
2	3	2011-01-01	1	0	1	2	0	6	0	1

Below, we show the shape of the file. You should see that the size of the DataFrame matches the number of lines in the file, minus the header row.

```
bike.shape
```

```
(17379, 17)
```

Question 1: Data Preparation

A few of the variables that are numeric/integer actually encode categorical data. These include `holiday`, `weekday`, `workingday`, and `weathersit`. In the following problem, we will convert these four variables to strings specifying the categories. In particular, use 3-letter labels (`Sun`, `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, and `Sat`) for `weekday`. You may simply use `yes`/`no` for `holiday` and `workingday`.

In this exercise we will *mutate* the data frame, **overwriting the corresponding variables in the data frame.** However, our notebook will effectively document this in-place data transformation for future readers. Make sure to leave the underlying datafile `bikeshare.txt` unmodified.

Question 1a

Decode the `holiday`, `weekday`, `workingday`, and `weathersit` fields:

1. holiday: Convert to `yes` and `no`. **Hint**: There are fewer holidays...
2. weekday: It turns out that Monday is the day with the most holidays. Mutate the `'weekday'` column to use the 3-letter label (`'Sun'`, `'Mon'`, `'Tue'`, `'Wed'`, `'Thu'`, `'Fri'`, and `'Sat'`) instead of its current numerical values. Note `0` corresponds to `Sun`, `1` to `Mon` and so on.
3. workingday: Convert to `yes` and `no`.
4. weathersit: You should replace each value with one of `Clear`, `Mist`, `Light`, or `Heavy`.

Note: If you want to revert changes, run the cell that reloads the csv.

Hint: One simple approach is to use the replace method of the pandas DataFrame class. We haven't discussed how to do this so you'll need to look at the documentation. The most concise way is with the approach described in the documentation as `nested-dictonaries`, though there are many possible solutions. E.g. for a DataFrame nested dictionaries, e.g., `{'a': {'b': np.nan}}`, are read as follows: look in column `a` for the value `b` and replace it with `NaN`.

```
# BEGIN YOUR CODE
# -----------------------
factor_dict = {
bike['holiday'].replace({0: 'no', 1: 'yes'}, inplace = True),
bike['weekday'].replace({0: 'Sun', 1: 'Mon', 2:'Tue', 3: 'Wed', 4: 'Thu', 5:'Fri', 6:'Sat'}, inp
bike['workingday'].replace({0: 'no', 1: 'yes'}, inplace = True),
bike['weathersit'].replace({1: 'Clear', 2: 'Mist',3:'Light',4: 'Heavy'}, inplace = True)
}
# -----------------------
# END YOUR CODE
bike.replace(factor_dict, inplace=True)
bike.head()
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear

Explanation: Using the *dataframe.replace* method I placed *bike['name of the column']* and replaced the variables with the required words. Before replacement, the values were 0, 1, or 2, so had to replace them in order. After that I had to insert *inplace = True*, because in default is *inplace = False* and since I wanted the modified final output I had to choose the *inplace = True*.

```
assert isinstance(bike, pd.DataFrame) == True
assert bike['holiday'].dtype == np.dtype('O')
assert list(bike['holiday'].iloc[370:375]) == ['no', 'no', 'yes', 'yes', 'yes']
assert bike['weekday'].dtype == np.dtype('O')
assert bike['workingday'].dtype == np.dtype('O')
assert bike['weathersit'].dtype == np.dtype('O')
assert bike.shape == (17379, 17) or bike.shape == (17379, 18)
assert list(bike['weekday'].iloc[::2000]) == ['Sat', 'Tue', 'Mon', 'Mon', 'Mon', 'Sun', 'Sun', '

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

Question 1b

How many entries in the data correspond to holidays? Set the variable `num_holidays` to this value.

Hint: `value_counts`

```
num_holidays = bike['holiday'].value_counts(ascending= True)['yes']
num_holidays
```

```
500
```

Explanation: Since 'yes' in the holiday column corresponds to a holiday, I had to find the total number of 'yes' in the whole holiday column. To do that, I used value_counts, which returns the count of the unique value (in this example 'yes')

```
assert num_holidays == 500
assert 1 <= num_holidays <= 10000

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

Question 1c (Computing Daily Total Counts)

The granularity of this data is at the hourly level. However, for some of the analysis we will also want to compute daily statistics. In particular, in the next few questions we will be analyzing the daily number of registered and unregistered users.

Construct a data frame named `daily_counts` indexed by `dteday` with the following columns:

- `casual`: total number of casual riders for each day
- `registered`: total number of registered riders for each day
- `workingday`: whether that day is a working day or not (`yes` or `no`)

Hint: `groupby` and `agg`. For the `agg` method, please check the documentation for examples on applying different aggregations per column. If you use the capability to do different aggregations by column, you can do this task with a single call to `groupby` and `agg`. For the `workingday` column we can take any of the values since we are grouping by the day, thus the value will be the same within each group. Take a look at the `'first'` or `'last'` aggregation functions.

```
# BEGIN YOUR CODE
# -----------------------
daily_counts = bike.groupby("dteday").agg({'casual': sum, 'registered' : sum, 'workingday': 'las
# -----------------------
# END YOUR CODE
daily_counts.head()
```

	casual	registered	workingday
dteday			
2011-01-01	331	654	no
2011-01-02	131	670	no
2011-01-03	120	1229	yes
2011-01-04	108	1454	yes
2011-01-05	82	1518	yes

Explanation: Using the information learned in class and the instructions from the question, I knew I had to first *group it by "dteday"* to make sub-dataframes in terms of the index. Next, I used agg to apply the necessary function to the sub-data frames, which will then give the wanted output. The functions were given in the instruction which for 'casual' a total number was needed which is sum, the same goes for 'registered'. Finally, for 'workingday' it was said to use 'first' or 'last' aggregate functions, but 'first' shows the first non-NA item while 'last' shows the last non-NA item. Since 'workingday' has "yes" or "no" values I realized it didn't matter which one I chose.

```
assert np.round(daily_counts['casual'].mean()) == 848.0
assert np.round(daily_counts['casual'].var()) == 471450.0
assert np.round(daily_counts['registered'].mean()) == 3656.0
assert np.round(daily_counts['registered'].var()) == 2434400.0
assert sorted(list(daily_counts['workingday'].value_counts())) == [231, 500]

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

Part 2: Trump and Tweets

In this part, we will work with Twitter data in order to analyze Donald Trump's tweets.

Let's load data into our notebook. Run the cell below to read tweets from the json file into a list named `all_tweets`.

```
with open(workspace_path+"hw3-realdonaldtrump_tweets.json", "r") as f:
    all_tweets = json.load(f)
```

Here is what a typical tweet from `all_tweets` looks like:

```
pprint(all_tweets[-1])
```

```
 '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': 13493,
 'retweeted': False,
 'source': '<a href="http://twitter.com/download/iphone" '
           'rel="nofollow">Twitter for iPhone</a>',
 '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 Americaus',
          'entities': {'description': {'urls': []},
                       'url': {'urls': [{'display_url': 'Instagram.com/realDonaldTrump',
                                         'expanded_url': 'http://www.Instagram.com/realDona
                                         'indices': [0, 23],
                                         'url': 'https://t.co/OMxB0x7xC5'}]}},
          'favourites_count': 7,
          'follow_request_sent': False,
          'followers_count': 58311576,
          'following': True,
          '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': 100264,
          'location': 'Washington, DC',
          'name': 'Donald J. Trump',
          'notifications': False,
          'profile_background_color': '6D5C18',
          'profile_background_image_url': 'http://abs.twimg.com/images/themes/theme1/bg.png
          'profile_background_image_url_https': 'https://abs.twimg.com/images/themes/theme
          'profile_background_tile': True,
          'profile_banner_url': 'https://pbs.twimg.com/profile_banners/25073877/1550087458
          'profile_image_url': 'http://pbs.twimg.com/profile_images/874276197357596672/kUuh
          'profile_image_url_https': 'https://pbs.twimg.com/profile_images/87427619735759 6
          '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': 40563,
          'time_zone': None,
          'translator_type': 'regular',
          'url': 'https://t.co/OMxB0x7xC5',
          'utc_offset': None,
```

Question 2

Construct a DataFrame called `trump` containing data from 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.*

```
# BEGIN YOUR CODE
# -----------------------
column_settings = {
    'id' : [tweet['id'] for tweet in all_tweets],
    'time' : pd.to_datetime([tweet['created_at'] for tweet in all_tweets]),
    'source' : [tweet['source'] for tweet in all_tweets],
    'text' : [tweet['text'] if "text" in tweet else tweet['full_text'] for tweet in all_tweets],
    'retweet_count' : [tweet['retweet_count'] for tweet in all_tweets]
}

trump = pd.DataFrame( column_settings,
    columns = ['id','time', 'source', 'text', 'retweet_count']).set_index('id').sort_index()
# -----------------------
# END YOUR CODE
trump.head()
```

	time	source	
id			
690171032150237184	2016-01-21 13:56:11+00:00	<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>	@rea @ https://t.co/
690171403388104704	2016-01-21 13:57:39+00:00	<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>	"@A @ Remembe gave out gi ALIENS a bor
690173226341691392	2016-01-21	<a href="http://twitter.com/download/android"	So sad th many oth show the r

Explanation: To construct a dataframe where the data is stored in all_tweets, I immediately realized I had to create new series and include that series in the main dataframe as a column.

First, I had to gather the data to make the columns, which I did at column_settings. For example, to get the id data at dataframe: tweet, I had to use the *for in statement* to receive all the data. Now, the two important parts here are that I had to use pd.to_datetime for the time column to properly encode the timestamps and had to place an *if-else statement* in the text column because sometimes when the text field is empty the full_text field is used instead. Hence, to not leave any empty fields I had to consider those situations.

Next, I constructed the dataframe named trump, which still needed some additional settings. I placed my column_settings to receive the data, named the columns, set 'id' as the index, and sorted the dataframe by index with sort_index() because it was a requirement.

```
assert isinstance(trump, pd.DataFrame)
assert 10000 < trump.shape[0] < 11000
assert trump.shape[1] >= 4
assert 831846101179314177 in trump.index
assert all(col in trump.columns for col in ['time', 'source', 'text', 'retweet_count'])
assert np.sometrue([('Twitter for iPhone' in s) for s in trump['source'].unique()])
assert trump['text'].dtype == np.dtype('O')
assert trump['retweet_count'].dtype == np.dtype('int64')
assert 753063644578144260 in trump.index

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

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:

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

```
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>',
       '<a href="https://studio.twitter.com" rel="nofollow">Twitter Media Studio</a>'],
      dtype=object)
```

Question 3

Notice how sources like "Twitter for Android" or "Instagram" are surrounded by HTML tags. In the cell below, clean up the `source` field by removing the HTML tags from each `source` entry.

Hints:

- Use `trump['source'].str.replace` along with a regular expression.
- You may find it helpful to experiment with regular expressions at regex101.com

```
# BEGIN YOUR CODE
# -----------------------
trump['source'] = trump['source'].str.replace('<[^>]+>','')
# -----------------------
# END YOUR CODE
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: FutureWarning: The default va
  This is separate from the ipykernel package so we can avoid doing imports until
```

Explanation: Because the instructions were to clean up the source field and remove the HTML tags, all I could think of was to use *str.replace['regex pattern' ,'')]*.

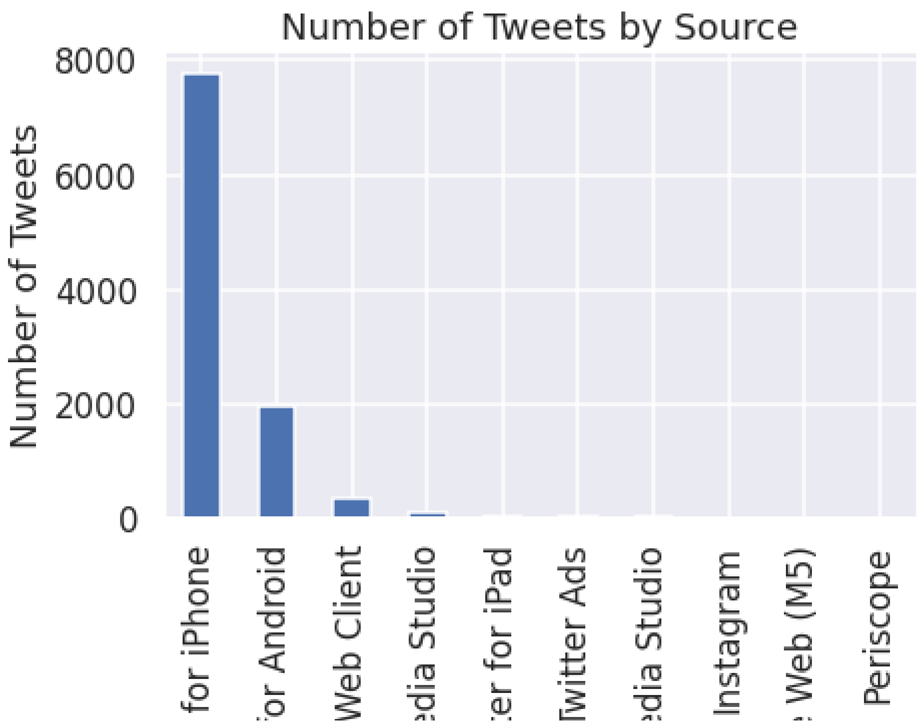
Now, the regular expression: <[^>]+> can be divided into three parts: < , [^>]+ , and >. The first < is to match a "<" character, then [^>]+ means to match any character that is not > and the match must be at least one, finally the > is to match a ">" character. So this means to match anything that starts with < and ends with >, which properly points to the HTML tags, and then by replacing it with '', it removes the tag from each source entry.

```
assert set(trump['source'].unique()) == set(['Twitter for Android', 'Twitter for iPhone', 'Twitt
       'Mobile Web (M5)', 'Instagram', 'Twitter for iPad', 'Media Studio',
       'Periscope', 'Twitter Ads', 'Twitter Media Studio'])

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

In the following plot, we see that there are two device types that are more commonly used than others.

```
plt.figure(figsize=(6, 4))
trump['source'].value_counts().plot(kind="bar")
plt.ylabel("Number of Tweets")
plt.title("Number of Tweets by Source");
```



Question 4

Now that we have cleaned up the `source` field, let's now look at which device Trump 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)

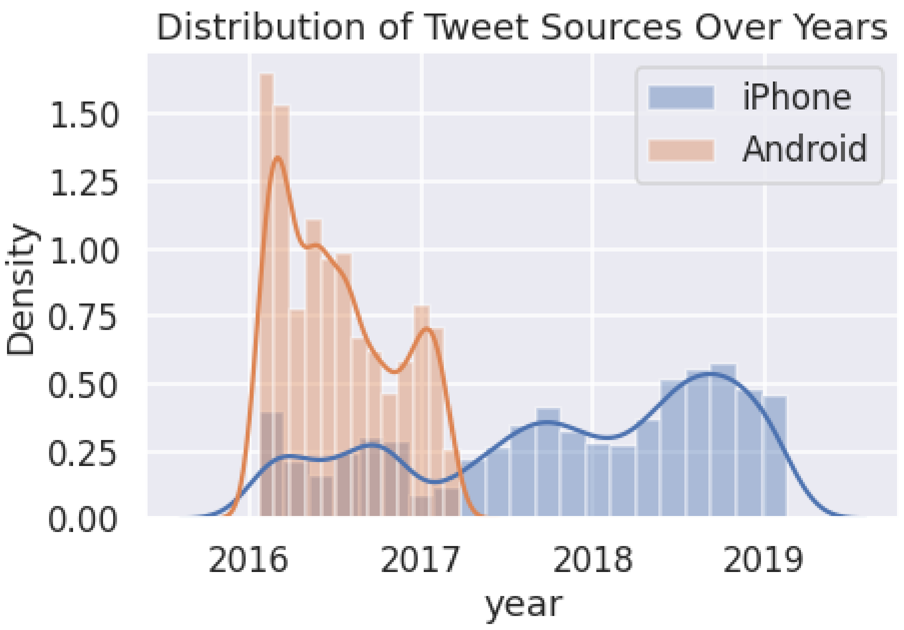
```
import datetime
def year_fraction(date):
    start = datetime.date(date.year, 1, 1).toordinal()
    year_length = datetime.date(date.year+1, 1, 1).toordinal() - start
    return date.year + float(date.toordinal() - start) / year_length

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

Now, use `sns.distplot` to overlay the distributions of Trump's 2 most frequently used web technologies over the years.

```
# BEGIN YOUR CODE
# -----------------------
top_devices = trump['source'].value_counts().head(2).index
sns.distplot(trump[trump['source'] == 'Twitter for iPhone']['year'], label = 'iPhone')
sns.distplot(trump[trump['source'] == 'Twitter for Android']['year'], label = 'Android')
plt.title("Distribution of Tweet Sources Over Years")
plt.legend();
```

```
# -----------------------
# END YOUR CODE
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplo
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplo
  warnings.warn(msg, FutureWarning)
```



Explanation: The instructions were to display the two most frequently used web technologies, so I had to use the 'source' field from the dataframe and had to retrieve the appropriate data from the previous plots, hence I used "Twitter for iPhone" and "Twitter for Android".

Next, the necessary parameter for sns.displot is the observed data, which could be series, one-dimensional arrays, or lists. I decided to use series *(trump[trump['source'] == 'Twitter for iPhone'])* and have a name attribute be called 'year'*(['year'])*, which is used to label the data axis. I also used the label parameter because I had to show the legend at the end.

A plot is incomplete if it doesn't have a title or a legend, which is why I inserted the title by plt.title and legend by plt.legend()

Question 5

Is there a difference between Trump's 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 device come at different times than his tweets from an iPhone. Note that Twitter gives us his tweets in the UTC timezone (notice the `+0000` in the first few tweets).

```
for tweet in all_tweets[:3]:
    print(tweet['created_at'])
```

```
Wed Oct 12 14:00:48 +0000 2016
```

```
    Wed Oct 12 13:46:43 +0000 2016
    Wed Oct 12 12:59:05 +0000 2016
```

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.

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

Question 5a

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

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

- **Hint:** See the cell above for an example of working with dt accessors.

```
# BEGIN YOUR CODE
# -----------------------
trump['hour'] = trump['est_time'].dt.hour + (trump['est_time'].dt.minute/60) + (trump['est_time'
# -----------------------
# END YOUR CODE
```

Explanation: This one was simple since all I had to do was follow the given formula. The *s.dt.hour* returns the hour of the time in the data of the Series object, *s.dt.minute* returns the minute, and *s.dt.second* returns the second. So by using *trump['est_time']* as the Series, I followed the formula by adding the hour, minute, and second while dividing the minute and second by 60 and 3600 respectively.

```
assert np.isclose(trump.loc[690171032150237184]['hour'], 8.93639) == True

print('Passed all unit tests!')
```

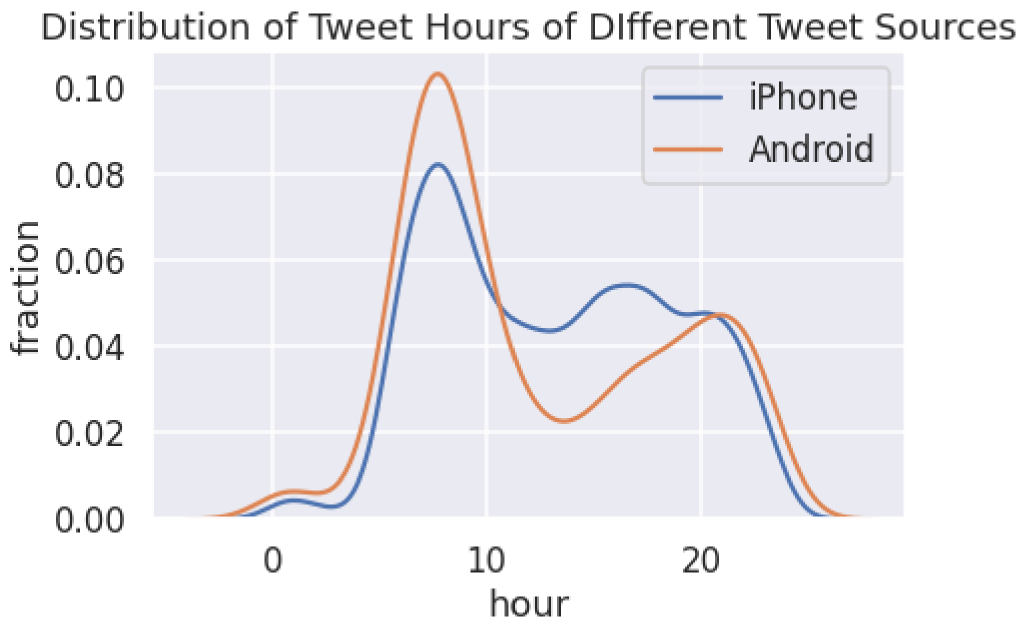
```
    Passed all unit tests!
```

Question 5b

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.

```
# BEGIN YOUR CODE
# -----------------------
top_devices = trump['source'].value_counts().head(2).index
sns.distplot(trump[trump['source'] == 'Twitter for iPhone']['hour'], label = 'iPhone', hist = Fa
sns.distplot(trump[trump['source'] == 'Twitter for Android']['hour'], label = 'Android', hist =
plt.ylabel('fraction')
plt.legend();
plt.title('Distribution of Tweet Hours of DIfferent Tweet Sources')
# -----------------------
# END YOUR CODE
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplc
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplc
  warnings.warn(msg, FutureWarning)
Text(0.5, 1.0, 'Distribution of Tweet Hours of DIfferent Tweet Sources')
```



Explanation: This one is very similar to Question 4, so I used the same format. However, there are some slight changes: the first being that instead of 'year' it's '*hour*' for the axis label since the requirements were to examine the distribution over the hours of the day of trump tweets. Second, the required plot only wanted the Kernel Density Curve, so I had to set the hist = False, which means to not plot a normed histogram (removing the boxes). Third, the y axis also had to be labeled so *plt.ylabel('fraction')* and finally cannot forget the title and legend.

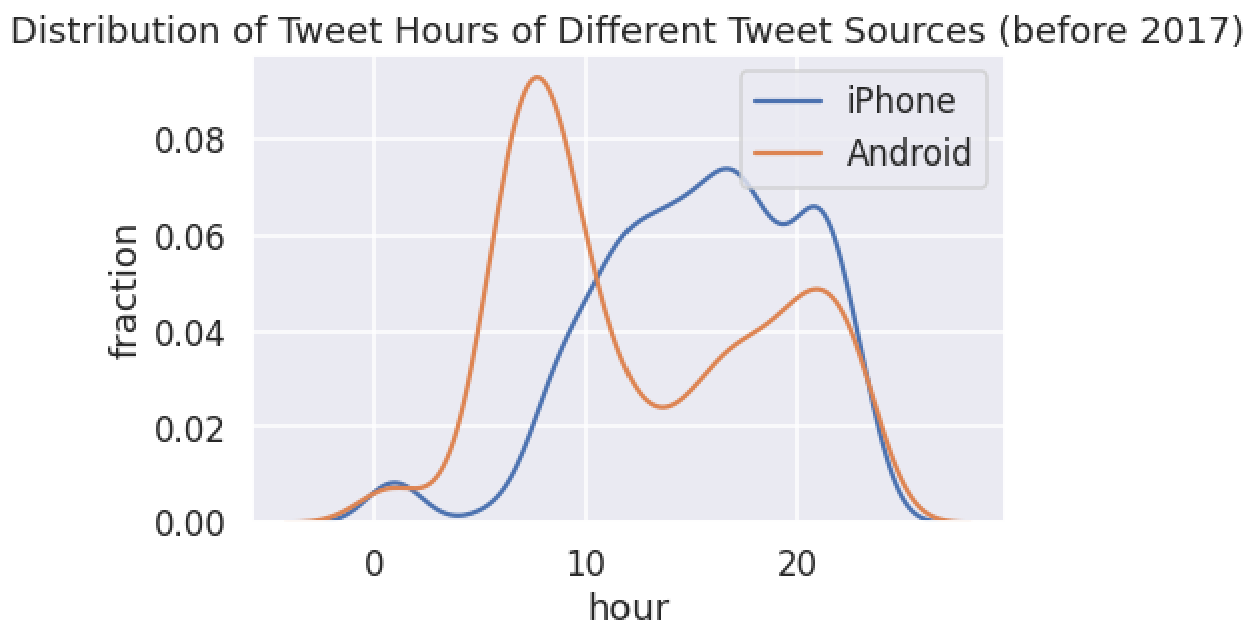
Question 5c

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

Let's see if this information significantly changes our plot. Create a figure similar to your figure from question 5b, but this time, only use tweets that were tweeted before 2017.

```
# BEGIN YOUR CODE
# -----------------------
before_2017= trump[trump['year'] < 2017]
top_devices = before_2017['source'].value_counts().head(2).index
sns.distplot(before_2017[before_2017['source'] == 'Twitter for iPhone']['hour'], label = 'iPhone
sns.distplot(before_2017[before_2017['source'] == 'Twitter for Android']['hour'], label = 'Andro
plt.ylabel('fraction')
plt.legend();
plt.title('Distribution of Tweet Hours of Different Tweet Sources (before 2017)')
# -----------------------
# END YOUR CODE
```

```
    /usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplc
      warnings.warn(msg, FutureWarning)
    /usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplc
      warnings.warn(msg, FutureWarning)
    Text(0.5, 1.0, 'Distribution of Tweet Hours of Different Tweet Sources (before 2017)')
```



Explanation: This one is similar to Question 5b but with the date difference. The question asked for a plot that Trump tweeted before 2017 so I made a dataframe named *before_2017* that only displays data where the year column has values smaller than 2017. Therefore, when the *sns.displot* gather data from *before_2017* and plot the distribution it will be the correct one.

Question 5d

During the campaign, it was theorized that Donald Trump's tweets from Android devices were written by him personally, and the tweets from iPhones were from his staff. Does your figure give support to this theory? What kinds of additional analysis could help support or reject this claim?

Answer: It can be supported because when Trump had an Android he tweeted alot around the hour of 10, and when he switched to Apple in 2017 the number of tweets were also posted close to the hour of 10.

Part 3: Sentiment Analysis

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

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

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

```
print(''.join(open(workspace_path+"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 6

As you can see, the lexicon contains emojis too! Each row contains a word and the *polarity* of that word, measuring how positive or negative the word is.

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

Question 6a

Read in the lexicon into a DataFrame called `sent`. The index of the DataFrame should be the words in the lexicon. `sent` should have one column named `polarity`, storing the polarity of each word.

- **Hint:** The `pd.read_csv` function may help here.

```
# BEGIN YOUR CODE
# -----------------------
sent = pd.read_csv('/content/drive/MyDrive/hw3/vader_lexicon.txt', sep = '₩t',header= None).rena
# -----------------------
# END YOUR CODE
sent.head()
```

	polarity
token	
$:	-1.5
%)	-0.4
%-)	-1.5
&-:	-0.4
&:	-0.7

Explanation: The first parameter of pandas.read_csv is the file path and the second one is the separator. The separator or sep identifies what separates each sentence by a regular expression. Looking at the vader_lexicon.txt, I noticed that a tab is what separates the words and the sentiments, hence I set the *sep as* '*\t*'. Since the vader_lexicon.txt file has no header it was important to set *header = none*.

For the *.rename()*, because the default column name of the dataframe is a number, I had to rename them as required: 0 to token and 1 to polarity.

Now, to make the words or tokens in the lexicon as index I used *set_index('token')* to do so, and because I had to show the polarity column I used *iloc[:, 0:1]*, which returns all rows and only the first column (since iloc is exclusive), which in this case is polarity.

```
assert np.allclose(sent['polarity'].head(), [-1.5, -0.4, -1.5, -0.4, -0.7]) == True
assert list(sent.index[5000:5005]) == ['paranoids', 'pardon', 'pardoned', 'pardoning', 'pardons'

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

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` DataFrame to be the lowercased text of each tweet.

```
# BEGIN YOUR CODE
# -----------------------
trump['text'] = trump['text'].str.lower()
# -----------------------
# END YOUR CODE
trump.head()
```

	time	source	text	retweet_count	
id					
690171032150237184	2016-01-21 13:56:11+00:00	Twitter for Android	"@bigop1: @realdonaldtrump @sarahpalinusa https://t.co/3kyqgqevyd"	1059	20
690171403388104704	2016-01-21 13:57:39+00:00	Twitter for Android	"@americanaspie: @glennbeck @sarahpalinusa remember when glenn gave out gifts to illegal aliens at crossing the border? me too!"	1339	20
690173226341691392	2016-01-21	Twitter for	so sad that @cnn and many others refused to show the massive crowd	2006	20

Explanation: To lowercase the text in the tweets all I had to use was *series.str.lower()*. The text column needed the transformation so *trump['text']* with *.str.lower()* did the job

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

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

Question 6c

Now, let's get rid of punctuation since it will cause us to fail to match words. Create a new column called `no_punc` in the `trump` DataFrame 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.)

```
# BEGIN YOUR CODE
# -----------------------
punct_re = r'[^₩w₩s]'  # Save your regex in punct_re
```

```
trump['no_punc'] = trump['text'].str.replace(punct_re, ' ')
# -----------------------
# END YOUR CODE
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: FutureWarning: The default va
  after removing the cwd from sys.path.
```

Explanation: Getting rid of punctuation characters is the same thing as removing characters that aren't a Unicode word character or a whitespace character. Luckily, there are regular expressions that match word characters and whitespace characters: * \w and \s* respectively. Now the important part is that punctuation character is NOT a word character or a whitespace character, so inserting the negation ^ is necessary.

[^\w\s] matches any character that is not a word character or a whitespace character.

Since I am specifying a pattern it is recommended to use raw strings: r

Now that the pattern *punct_re* has been made, I can use the *.str.replace* and replace the pattern with a space: " ", which will successfully remove the punctuation characters. Afterward, make that be the data of the newly created new column called no_punc in the trump dataframe *(trump['no_punc']).*

```
assert re.search(punct_re, 'this') == None
assert re.search(punct_re, 'this is not ok.') != None
assert re.search(punct_re, 'this#is#ok') != None
assert re.search(punct_re, 'this^is ok') != None

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

Question 6d

Now, let's convert the tweets into what's called a *tidy format* to make the sentiments easier to calculate. Use the `no_punc` column of `trump` to create a table called `tidy_format`. The index of the table should be the IDs of the tweets, repeated once for every word in the tweet. It has two columns:

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

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

	num	word
894661651760377856	0	i
894661651760377856	1	think

	num	word
894661651760377856	2	senator
894661651760377856	3	blumenthal
894661651760377856	4	should

Note that your DataFrame may look different from the one above. 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` DataFrame, 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.

```
# BEGIN YOUR CODE
# -----------------------
tidy_format = (trump['no_punc'].str.split(expand = True).stack().reset_index(level = 1).rename(c
# -----------------------
# END YOUR CODE
tidy_format.head()
```

	num	word
id		
690171032150237184	0	bigop1
690171032150237184	1	realdonaldtrump
690171032150237184	2	sarahpalinusa
690171032150237184	3	https
690171032150237184	4	t

Explanation: Use *str.split* to split the texts in the sentence of no_punc and have *expand = true* to split the texts into separate columns. The *stack()* will convert the column to a row and now that there are multiple indices in the Series I added a *reset_index(level = 1)*, which resets the index at level 1 (converts the index to columns) and not at level 0 because level 0 is the original index. Now that there are no more multi-level indices, I chose to rename the converted columns to the required names: level_1 to num and 0 to work.

```
assert tidy_format.loc[894661651760377856].shape == (27,2)
assert ' '.join(list(tidy_format.loc[894661651760377856]['word'])) == 'i think senator blumentha

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

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.

Hints:

- You will need to merge the `tidy_format` and `sent` tables and group the final answer.
- If certain words are not found in the `sent` table, set their polarities to 0.

```
# BEGIN YOUR CODE
# -----------------------
trump['polarity'] = tidy_format.merge(sent, how = 'left', left_on = 'word', right_index = True).
# -----------------------
# END YOUR CODE
trump[['text', 'polarity']].head()
```

id	text	polarity
690171032150237184	"@bigop1: @realdonaldtrump @sarahpalinusa https://t.co/3kyqgqevyd"	0.0
690171403388104704	"@americanaspie: @glennbeck @sarahpalinusa remember when glenn gave out gifts to illegal aliens at crossing the border? me too!"	-2.6
690173226341691392	so sad that @cnn and many others refused to show the massive crowd at the arena yesterday in oklahoma. dishonest reporting!	-6.0

Explanation: First, the requirement was to merge tidy_format(left) and sent(right) tables so *tidy_format.merge(sent)*. Next, it must be a left outer join, hence *how = 'left'*, and the joining key for the left table(tidy_format) should be the word column so *left_on = 'word'* and the joining key for the right table should be the index(token) so *right_index = True*.

Use *reset_index()* to convert the non-original index to column and set the id as the original index with *set_index('id')*. Afterward, inquire the dataframe displaying all rows, the only index, and one column: polarity with *loc[:,['polarity']]*.

Now, for the important part which is using groupby to create sub-dataframes according to 'id', then apply the sum() function(to the polarity column), which will appropriately level the overall length. Finally, I add a *fillna(0)* that fills NaN values with 0.

```
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)

print('Passed all unit tests!')
```

```
Passed all unit tests!
```

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:

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

```
Most negative tweets:

   the trump portrait of an unsustainable border crisis is dead on. “in the last two years,

   it is outrageous that poisonous synthetic heroin fentanyl comes pouring into the u.s. post

   the rigged russian witch hunt goes on and on as the “originators and founders” of this s

   ...this evil anti-semitic attack is an assault on humanity. it will take all of us working

   james comey is a proven leaker &amp; liar. virtually everyone in washington thought he sho
```

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

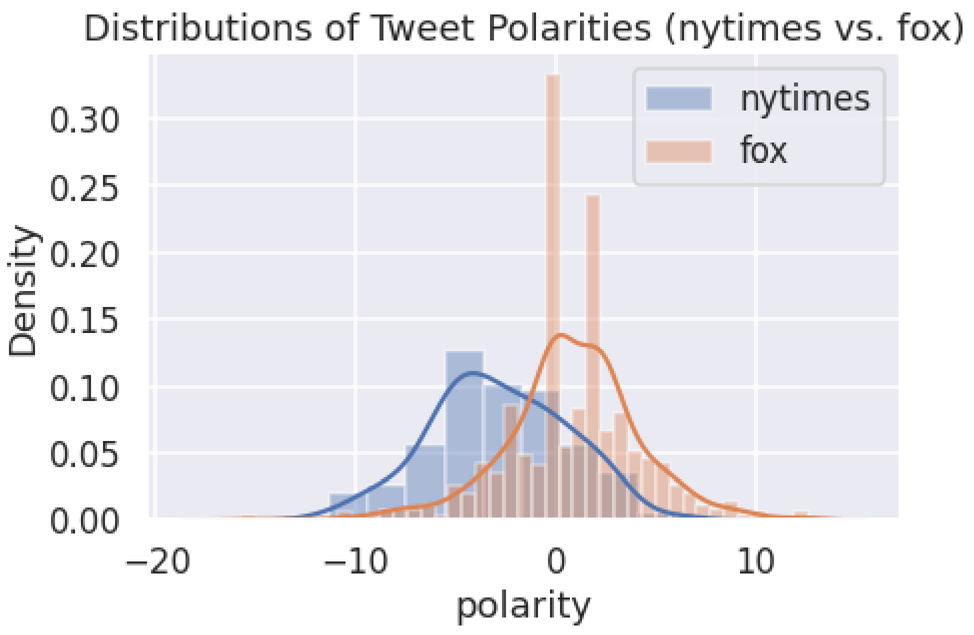
```
Most positive tweets:

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

   congratulations to a truly great football team, the clemson tigers, on an incredible win l

   my supporters are the smartest, strongest, most hard working and most loyal that we have s

   thank you to all of my great supporters, really big progress being made. other countries w

   thank you, @wvgovernor jim justice, for that warm introduction. tonight, it was my great h
```

Now, let's try looking at the distributions of sentiments for tweets containing certain keywords.

In the cell below, we create a single plot showing both the distribution of tweet sentiments for tweets containing `nytimes`, as well as the distribution of tweet sentiments for tweets containing `fox`. Here, we notice that the president appears to say more positive things about Fox than the New York Times.

```
sns.distplot(trump[trump['text'].str.lower().str.contains("nytimes")]['polarity'], label = 'nyti
sns.distplot(trump[trump['text'].str.lower().str.contains("fox")]['polarity'], label = 'fox')
plt.title('Distributions of Tweet Polarities (nytimes vs. fox)')
plt.legend();
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplc
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplc
  warnings.warn(msg, FutureWarning)
```



Congratulations! You have completed HW3.

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output.,

Please generate pdf as follows and submit it to Gradescope.

File > Print Preview > Print > Save as pdf

Please save before submitting!

✓ 0초 오후 2:13에 완료됨 ● ×