In [1]:

```
# ! pip install -U okpy  # Uncomment if you have an error
from client.api.notebook import Notebook
ok = Notebook('hw2.ok')
```

```
=====================================================================
Assignment: hw2
OK, version v1.18.1
=====================================================================
```

Homework 2: Exploratory Data Analysis (EDA)

Due Date: Fri 4/9, 11:59 pm PST

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.

In [3]:

```python
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.
import ds100_utils

# Ensure that Pandas shows at least 280 characters in columns, so we can see full tweet
pd.set_option('max_colwidth', 280)

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

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)

Variable	Description
hum	normalized percent humidity (divided by 100)
windspeed	normalized wind speed (divided by 67)
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes including casual and registered

In [4]:

```
for line in ds100_utils.head('data/bikeshare.txt'):
    print(line, end="")
```

```
instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathersit,temp,atemp,hum,wi
ndspeed,casual,registered,cnt
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.

In [5]:

```
bike = pd.read_csv('data/bikeshare.txt')
bike.head()
```

Out[5]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	h
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	(
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	(
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	(
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	(
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	(

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.

In [6]:

```
bike.shape
```

Out[6]:

```
(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` .

In [7]:

```
# BEGIN YOUR CODE
# -----------------------

bike = bike.replace({'holiday': {0: 'no', 1: 'yes'}})
bike = bike.replace({'weekday': {0: 'Sun', 1: 'Mon',2:'Tue',3:'Wed',4:'Thu',5:'Fri',6:'
bike = bike.replace({'workingday': {0: 'no', 1: 'yes'}})
bike = bike.replace({'weathersit': {1: 'Clear', 2: 'Mist',3:'Light',4:'Heavy'}})








# -----------------------
# END YOUR CODE
bike.head()
```

Out[7]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	h
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24	0.2879	(
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22	0.2727	(
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22	0.2727	(
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24	0.2879	(
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24	0.2879	(

In [8]:

```
ok.grade("q1a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 10
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 1b

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

Hint: `value_counts`

In [9]:

```
num_holidays = bike.holiday.value_counts().yes
print(num_holidays)
```

```
500
```

In [10]:

```
ok.grade("q1b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

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.

In [11]:

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

Out[11]:

	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

In [12]:

```
ok.grade("q1c");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 5
    Failed: 0
[ooooooooook] 100.0% passed
```

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` .

In [13]:
```
with open("data/hw2-realdonaldtrump_tweets.json", "r") as f:
    all_tweets = json.load(f)
```

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

In [14]:
```
pprint(all_tweets[-1])
```

```
{'contributors': None,
 'coordinates': None,
 'created_at': 'Tue Oct 16 18:40:18 +0000 2018',
 'display_text_range': [0, 174],
 'entities': {'hashtags': [], 'symbols': [], 'urls': [], 'user_mentions': []},
 'favorite_count': 52115,
 'favorited': False,
 'full_text': 'Just spoke with the Crown Prince of Saudi Arabia who totally '
              'denied any knowledge of what took place in their Turkish '
              'Consulate. He was with Secretary of State Mike Pompeo...',
 'geo': None,
 'id': 1052268011900555265,
 'id_str': '1052268011900555265',
 '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/realD
onaldTrump',
                                         '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/the
me1/bg.png',
          'profile_background_tile': True,
          'profile_banner_url': 'https://pbs.twimg.com/profile_banners/25073877/15500874
58',
          'profile_image_url': 'http://pbs.twimg.com/profile_images/874276197357596672/k
Uuht00m_normal.jpg',
          'profile_image_url_https': 'https://pbs.twimg.com/profile_images/8742761973575
96672/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': 40563,
          'time_zone': None,
          'translator_type': 'regular',
          'url': 'https://t.co/OMxB0x7xC5',
          'utc_offset': None,
          'verified': True}}
```

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 as below.**

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

In [15]:

```
# BEGIN YOUR CODE
# -----------------------
```

```
trump = pd.DataFrame.from_dict(all_tweets[0:])
trump = trump.set_index('id')
trump = trump.filter(items=['id','created_at','source','text','retweet_count'])
trump.rename(columns = {'created_at': 'time'},inplace = True)
trump['time']=pd.to_datetime(trump['time']).dt.tz_localize(None)
# ------------------------
# END YOUR CODE
trump.head()
```

Out[15]:

	time	source	
id			
786204978629185536	2016-10-12 14:00:48	<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>	PAY TO PLAY POLITICS. \n#Croo https://t.co/\
786201435486781440	2016-10-12 13:46:43	<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>	Very little pick-up by the dishonest incredible information pr WikiLeaks. So dishonest! Rigge
786189446274248704	2016-10-12 12:59:05	<a href="http://twitter.com/download/android" rel="nofollow">Twitter for Android</a>	Crooked Hillary Clinton likes to t the things she will do but she has b for 30 years - why didn't she
786054986534969344	2016-10-12 04:04:47	<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>	Thank you Florida- a MOVEMEN never been seen before and wil seen again. Lets get ou https://t.co/t9X
786007502639038464	2016-10-12 00:56:06	<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</a>	Join me Thursday in Flori Ohio!\nWest Palm Be noon:\nhttps://t.co/jwbZnQhxg9\n(OH this 7:30pm:\nhttps://t.co/5\

In [16]:

```
ok.grade("q2");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 11
    Failed: 0
[ooooooooook] 100.0% passed
```

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 [17]:

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

Out[17]:

```
array(['<a href="http://twitter.com/download/iphone" rel="nofollow">Twitter for iPhone</
a>',
```

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

In [18]:

```
# BEGIN YOUR CODE
# -----------------------
trump['source'] = trump['source'].str.replace(r"\<.*?.>",'')
trump['source'].head()
# -----------------------
# END YOUR CODE
```

Out[18]:

```
id
786204978629185536      Twitter for iPhone
786201435486781440      Twitter for iPhone
786189446274248704    Twitter for Android
786054986534969344      Twitter for iPhone
786007502639038464      Twitter for iPhone
Name: source, dtype: object
```

In [19]:

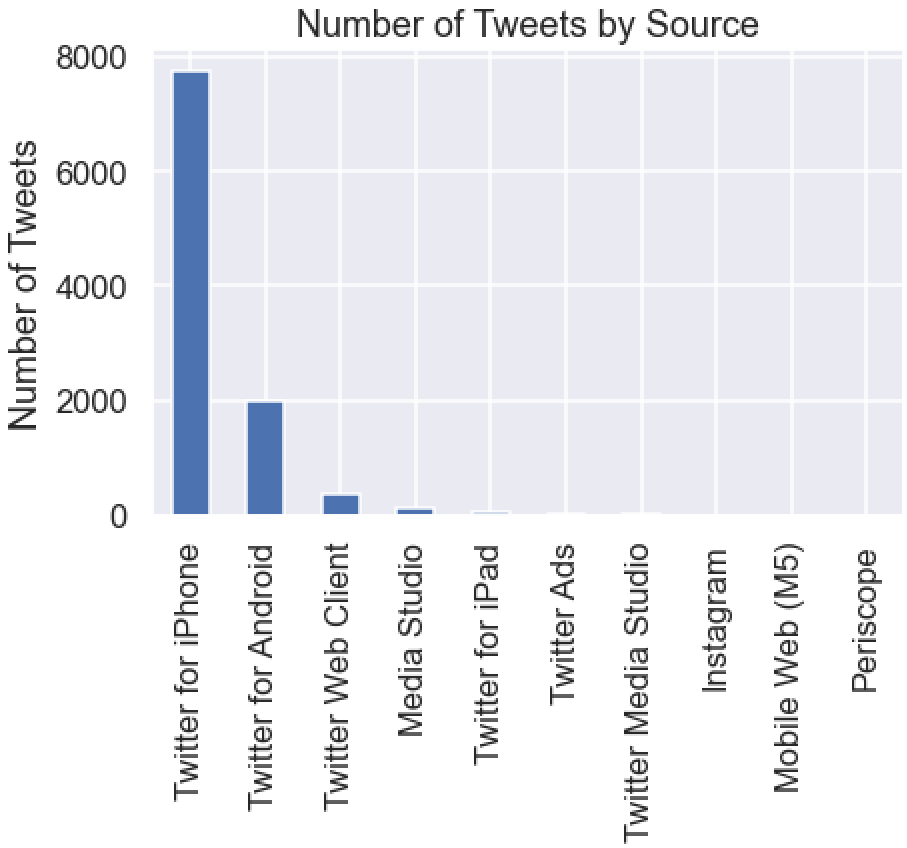
```
ok.grade("q3");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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

In [20]:

```
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)

In [21]:

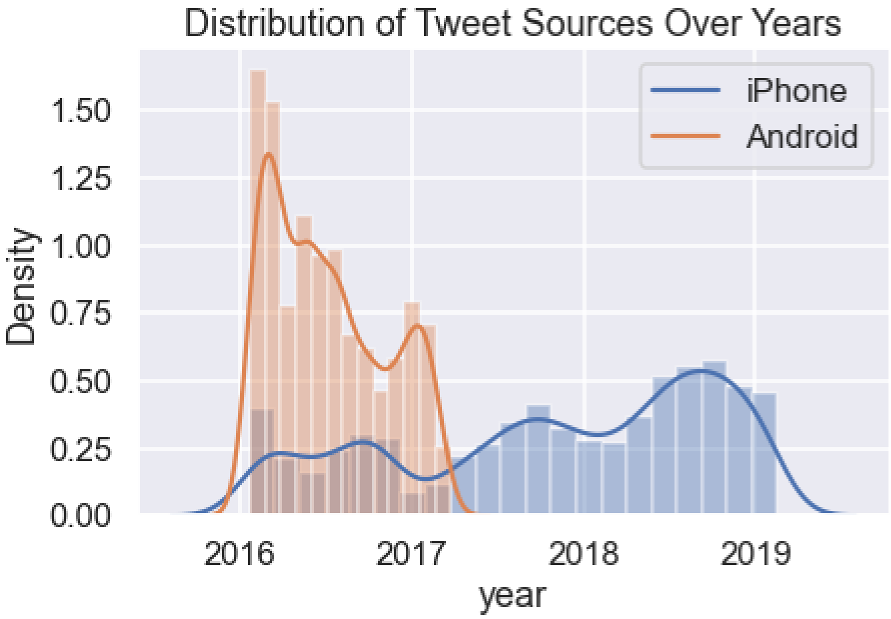
```
import datetime
def year_fraction(date):
    start = datetime.date(date.year, 1, 1).toordinal()
    year_length = datetime.date(date.year+1, 1, 1).toordinal() - start
    return date.year + float(date.toordinal() - start) / year_length

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. Your final plot should look like:

In [22]:
```
# BEGIN YOUR CODE
# -----------------------
iphone=trump[trump["source"]=="Twitter for iPhone"]
android=trump[trump["source"]=="Twitter for Android"]

sns.distplot(iphone["year"])
sns.distplot(android["year"])

plt.xlabel("year")
plt.title("Distribution of Tweet Sources Over Years")
plt.legend(['iPhone','Android'])
# -----------------------
# END YOUR CODE
```

```
c:\users\acer\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
c:\users\acer\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[22]:
```
<matplotlib.legend.Legend at 0x1d6cb2d93c8>
```



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).

In [23]:

```
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.

In [24]:

```
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[24]:

	time	source	text	retweet_count	
id					
786204978629185536	2016-10-12 14:00:48	Twitter for iPhone	PAY TO PLAY POLITICS. \n#CrookedHillary https://t.co/wjsl8ITVvk	24915	2016.7
786201435486781440	2016-10-12 13:46:43	Twitter for iPhone	Very little pick-up by the dishonest media of incredible information provided by WikiLeaks. So dishonest! Rigged system!	22609	2016.7
786189446274248704	2016-10-12 12:59:05	Twitter for Android	Crooked Hillary Clinton likes to talk about the things she will do but she has been there for 30 years - why didn't she do them?	18329	2016.7
786054986534969344	2016-10-12 04:04:47	Twitter for iPhone	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	18789	2016.7
786007502639038464	2016-10-12 00:56:06	Twitter for iPhone	Join me Thursday in Florida &amp; Ohio!\nWest Palm Beach, FL at noon:\nhttps://t.co/jwbZnQhxg9\nCincinnati, OH this 7:30pm:\nhttps://t.co/5w2UhalPIx	7761	2016.7

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.

In [25]:

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

In [26]:

```
ok.grade("q5a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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. Your plot should look similar to the following:

In [27]:

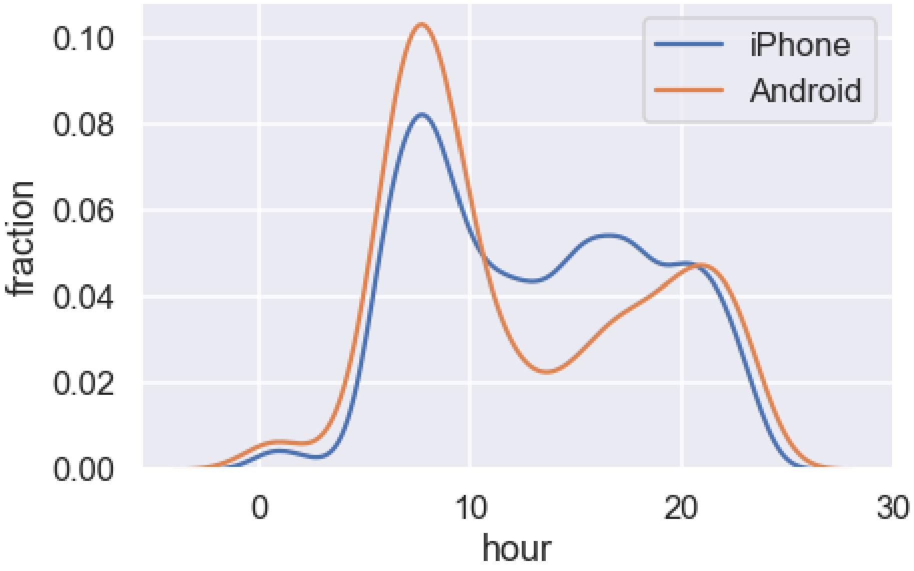
```
# BEGIN YOUR CODE
# -----------------------
iphone = trump[trump['source'] == 'Twitter for iPhone']
android = trump[trump['source'] == 'Twitter for Android']

sns.distplot(iphone['hour'], label = 'iPhone', hist=False)
sns.distplot(android['hour'], label = 'Android', hist=False)

plt.ylabel('fraction')
plt.xticks( [0, 10, 20, 30])
plt.legend()
# -----------------------
# END YOUR CODE
```

```
c:\users\acer\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
c:\users\acer\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
```

```
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```

Out[27]:

```
<matplotlib.legend.Legend at 0x1d6cf9ebf28>
```



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. Your plot should look similar to the following:

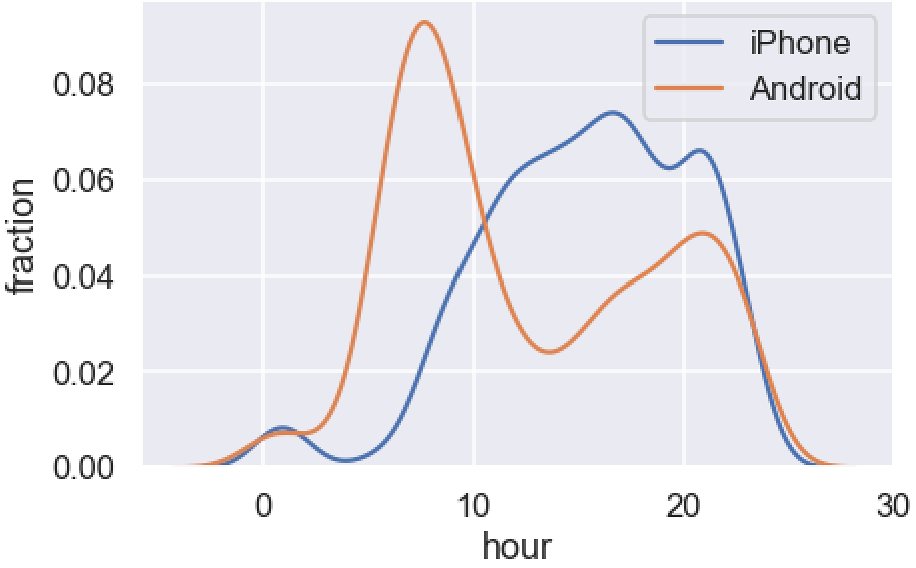
In [28]:

```
# BEGIN YOUR CODE
# -----------------------
trump['year'] = trump['est_time'].dt.year
iphone = trump[(trump['source'] == 'Twitter for iPhone')& (trump['year'] < 2017)]
android = trump[(trump['source'] == 'Twitter for Android')& (trump['year'] < 2017)]

sns.distplot(iphone['hour'], label = 'iPhone', hist=False)
sns.distplot(android['hour'], label = 'Android', hist=False)

plt.ylabel('fraction')
plt.xticks( [0, 10, 20, 30])
plt.legend()

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

```
c:\users\acer\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
c:\users\acer\miniconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning:
```

```
`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `kdeplot` (an axes-level function for kernel density plots).
  warnings.warn(msg, FutureWarning)
```

```
Out[28]: <matplotlib.legend.Legend at 0x1d6d1aca908>
```



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: `The result appeared from the graph above does not support the theory. If we look at the graph, we can see that the tweets posted from Android device are distributed with some sort of "pattern", the tweets are majorly done by the same time, while posts from Iphone do not show to fit any strict schedule. This makes me believe that the posts from Iphone,that do not necessarily depend on daytime, are done personally by Trump. On the other hand, tweets from Android seem to be posted by his staff.`

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:

In [29]:

```
print(''.join(open("data/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.

In [76]:

```
# BEGIN YOUR CODE
# -----------------------
x = pd.read_csv('data/vader_lexicon.txt', header = None,error_bad_lines=False).iloc[:,
polarity = {'polarity': pd.read_csv('data/vader_lexicon.txt', header = None,error_bad_l

sent = pd.DataFrame(data = polarity).set_index(x)
sent.index.name = None
# -----------------------
# END YOUR CODE
sent.head()
```

```
b'Skipping line 55: expected 10 fields, saw 11\nSkipping line 113: expected 10 fields, s
aw 11\n'
```

```
b'Skipping line 55: expected 10 fields, saw 11\nSkipping line 113: expected 10 fields, s
aw 11\n'
```

Out[76]:

	polarity
$:\t-1.5\t0.80623\t[-1	-1
%)\t-0.4\t1.0198\t[-1	0
%-)\t-1.5\t1.43178\t[-2	0
&-:\t-0.4\t1.42829\t[-3	-1
&:\t-0.7\t0.64031\t[0	-1

In [77]:

```
ok.grade("q6a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
q6a > Suite 1 > Case 2

>>> sent.shape
(7515, 1)

# Error: expected
#     (7517, 1)
# but got
#     (7515, 1)

Run only this test case with "python3 ok -q q6a --suite 1 --case 2"
---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 1
[ooooook.....] 50.0% passed
```

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.

In [78]:

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

Out[78]:

	time	source	text	retweet_count	year
id					

	time	source	text	retweet_count	year
id					
786204978629185536	2016-10-12 14:00:48	Twitter for iPhone	pay to play politics. \n#crookedhillary https://t.co/wjsl8itvvk	24915	2016
786201435486781440	2016-10-12 13:46:43	Twitter for iPhone	very little pick-up by the dishonest media of incredible information provided by wikileaks. so dishonest! rigged system!	22609	2016
786189446274248704	2016-10-12 12:59:05	Twitter for Android	crooked hillary clinton likes to talk about the things she will do but she has been there for 30 years - why didn't she do them?	18329	2016
786054986534969344	2016-10-12 04:04:47	Twitter for iPhone	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	18789	2016
786007502639038464	2016-10-12 00:56:06	Twitter for iPhone	join me thursday in florida &amp; ohio!\nwest palm beach, fl at noon:\nhttps://t.co/jwbznqhxg9\ncincinnati, oh this 7:30pm:\nhttps://t.co/5w2uhalpix	7761	2016

In [79]:

```
ok.grade("q6b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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.)

In [80]:

```
# BEGIN YOUR CODE
# -----------------------
punct_re = r'[^a-zA-Z\d\s]'
trump['no_punc'] = trump['text'].str.replace(punct_re, ' ')
# -----------------------
# END YOUR CODE
```

In [81]:

```
ok.grade("q6c");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 10
    Failed: 0
[ooooooooook] 100.0% passed
```

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
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.

In [82]:

```
# BEGIN YOUR CODE
# -----------------------
tidy_format = trump['no_punc'].str.split(expand=True).stack().to_frame().reset_index(le
tidy_format.columns = ["num", "word"]
# -----------------------
# END YOUR CODE
tidy_format.head()
```

Out[82]:

	num	word
id		
786204978629185536	0	pay
786204978629185536	1	to
786204978629185536	2	play
786204978629185536	3	politics
786204978629185536	4	crookedhillary

In [83]:

```
ok.grade("q6d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

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.

In [90]:

```
# BEGIN YOUR CODE
# -----------------------
```

```
trump['polarity'] = (tidy_format.merge(sent, left_on = 'word',
                                       right_index = True)['polarity'].sum())
trump['polarity'].replace(np.float64('nan'), 0, inplace = True)
# -----------------------
# END YOUR CODE
trump[['text', 'polarity']].head()
```

Out[90]:

id	text	polarity
786204978629185536	pay to play politics. \n#crookedhillary https://t.co/wjsl8itvvk	0
786201435486781440	very little pick-up by the dishonest media of incredible information provided by wikileaks. so dishonest! rigged system!	0
786189446274248704	crooked hillary clinton likes to talk about the things she will do but she has been there for 30 years - why didn't she do them?	0
786054986534969344	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	0
786007502639038464	join me thursday in florida &amp; ohio!\nwest palm beach, fl at noon:\nhttps://t.co/jwbznqhxg9\ncincinnati, oh this 7:30pm:\nhttps://t.co/5w2uhalpix	0

In [91]:

```
ok.grade("q6e");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
q6e > Suite 1 > Case 1

>>> np.allclose(trump.loc[744701872456536064, 'polarity'], 8.4)
False

# Error: expected
#     True
# but got
#     False

Run only this test case with "python3 ok -q q6e --suite 1 --case 1"
---------------------------------------------------------------------
Test summary
    Passed: 0
    Failed: 1
[k..........] 0.0% passed
```

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 [92]:

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

```
Most negative tweets:

   pay to play politics.
#crookedhillary https://t.co/wjsl8itvvk

   nan

   nan

   nan

   nan
```

In [93]:

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

```
Most positive tweets:

   pay to play politics.
#crookedhillary https://t.co/wjsl8itvvk

   nan

   nan

   nan

   nan
```

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.

In [94]:

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-94-2c44e2f04d2d> in <module>
----> 1 sns.distplot(trump[trump['text'].str.lower().str.contains("nytimes")]['polarit
y'], label = 'nytimes')
      2 sns.distplot(trump[trump['text'].str.lower().str.contains("fox")]['polarity'],
 label = 'fox')
      3 plt.title('Distributions of Tweet Polarities (nytimes vs. fox)')
      4 plt.legend();

c:\users\acer\miniconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, ke
y)
   2894
   2895         # Do we have a (boolean) 1d indexer?
-> 2896         if com.is_bool_indexer(key):
   2897             return self._getitem_bool_array(key)
   2898
```

```
c:\users\acer\miniconda3\lib\site-packages\pandas\core\common.py in is_bool_indexer(key)
    132                 na_msg = "Cannot mask with non-boolean array containing NA / NaN
values"
    133                 if isna(key).any():
--> 134                     raise ValueError(na_msg)
    135                 return False
    136             return True

ValueError: Cannot mask with non-boolean array containing NA / NaN values
```

Congratulations! You have completed HW2.

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!

```
In [ ]:
```