[redacted]

File: [redacted]

Question 1 ¶

In [1]:

```
import json
from collections import Counter
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from copy import deepcopy
```

1. Tweets1.json corresponds to tweets received before a Presidential debate in 2016 and all other data files correspond to tweets received immediately after the same Presidential debate.

Please write codes to read the data files tweets1.json to tweets5.json and combine tweets2.json to tweets5.json to a single file, named tweets2.json. Determine the number of tweets in tweets1.json and tweets2.json

In [2]:

```
#read tweet.json files
import os
os.chdir("[redacted]")
with open('tweets1.json','r') as f1:
    tweets1_json = f1.readlines()
with open('tweets2.json','r') as f2:
    tweets2_json = f2.readlines()
def mergetotweetws2(file, tweets2_json):
    for lines in file:
        tweets2_json.append(lines)
with open('tweets3.json','r') as f3:
    mergetotweetws2(f3, tweets2_json)
with open('tweets4.json','r') as f4:
    mergetotweetws2(f4, tweets2_json)
with open('tweets5.json','r') as f5:
    mergetotweetws2(f5, tweets2_json)
```

Solution 1： Load the json file and output the length of it

In [3]:

```
def convertion(file):
    tweets = []
    for i in range(len(file)):
        try:
            tweet = json.loads(file[i])
        except:
            continue
        tweets.append(tweet)
    return tweets
tweets1_json = convertion(tweets1_json)
tweets2_json = convertion(tweets2_json)
print('The number of tweets in tweets1_json is:', len(tweets1_json))
print('The number of tweets in tweets2_json is:', len(tweets2_json))
```

```
The number of tweets in tweets1_json is: 82947
The number of tweets in tweets2_json is: 62334
```

Solution 2: If the format of {'limit': {'track': 14, 'timestamp_ms': '1475975530873'}} is not tweet, then it should be removed

In [4]:

```
def deletenulltweet(file):
    list = deepcopy(file)
    removelist = []
    for i in list:
        if 'user' not in i.keys():
            removelist.append(i)
    list = [x for x in list if x not in removelist]
    return list
newlist1 = deletenulltweet(tweets1_json)
newlist2 = deletenulltweet(tweets2_json)
print('The remaining number of tweets in tweets1_json is:', len(newlist1))
print('The remaining number of tweets in tweets2_json is:', len(newlist2))
```

```
The remaining number of tweets in tweets1_json is: 77333
The remaining number of tweets in tweets2_json is: 57554
```

2. In order to clean the tweets in each file with a focus on extracting hashtags, we observe that 'retweeted_status' is another tweet within a tweet. We select tweets using the following criteria:

- Non-empty number of hashtags either in 'entities' or in its 'retweeted_status'.
- There is a timestamp.
- There is a legitimate location.
- Extract hashtags that were written in English or convert hashtags that were written partially in English (Ignore non-english characters).

Write a function to return a dictionary of acceptable tweets, locations and hashtags for both tweets1.json and the tweets2.json respectively.

In [5]:

```
# Defien a new list adding all the qualified retweeted_status
def selectedlist(original):
    # Adding qualified retweeted_status to the list
    result = []
    count = 0
    for i in range(len(original)):
        if 'retweeted_status' in original[i].keys():
            if type(original[i]['retweeted_status']['user']['location']) == str
and original[i]['retweeted_status']['entities']['hashtags'] != []:
                result.append(original[i]['retweeted_status'])
                count += 1
        else:
            continue
    print('the number of qualified retweeted tweet is:', count)
    count = 0
    # for all the tweets, delete the one has empty hashtags and location
    for i in range(len(original)):
        if type(original[i]['user']['location']) == str and original[i]['entitie
s']['hashtags'] != []:
                result.append(original[i])
                count += 1
        else:
            continue
    print('The number of qualified tweet is:', count)

    # change all the create_date to datetime and all the hashtags in legitimate
 format
    for m in range(len(result)):
        f = "%a %b %d %H:%M:%S +0000 %Y"
        legitimate = set("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01
23456789")
        result[m]['created_at'] = datetime.strptime(result[m]['created_at'], f)
        for text in result[m]['entities']['hashtags']:
            text = ''.join(char for char in text['text'] if char in legitimate)

    # return the result dictionary
    resultlist = {'tweets': [], 'CreatedDate': [], 'location': [], 'hashtags':
[]}
    for item in result:
        resultlist['tweets'].append(item)
        resultlist['CreatedDate'].append(item['created_at'])
        resultlist['location'].append(item['user']['location'])
        resultlist['hashtags'].append(item['entities']['hashtags'])
    return resultlist
```

In [6]:

```
new_tweets1_json = selectedlist(newlist1)
new_tweets2_json = selectedlist(newlist2)
```

```
the number of qualified retweeted tweet is: 8551
The number of qualified tweet is: 9296
the number of qualified retweeted tweet is: 13454
The number of qualified tweet is: 10961
```

```
In [7]:
```

```
print('The final number of tweets in tweets1_json is:', len(new_tweets1_json['tw
eets']))
print('The final number of tweets in tweets2_json is:', len(new_tweets2_json['tw
eets']))
```

```
The final number of tweets in tweets1_json is: 17847
The final number of tweets in tweets2_json is: 24415
```

3. Write a function to extract the top n tweeted hashtags of a given hashtag list. Use the function to find the top n tweeted hashtags of the tweets1.json and the tweets2.json respectively.

```
In [8]:
```

```
def topnhashtags(file,n):
    hashtags = []
    for i in file['hashtags']:
        for item in i:
            hashtags.append(item['text'])
    countertdata = Counter(hashtags)
    return countertdata.most_common(n)
```

```
In [9]:
```

```
#print an example
print('The top 10 hashtags for tweets1.json are:')
topnhashtags(new_tweets1_json,10)
```

```
The top 10 hashtags for tweets1.json are:
```

```
Out[9]:
```

```
[('Trump', 3140),
 ('debate', 1697),
 ('TrumpTapes', 1478),
 ('MAGA', 1095),
 ('Hillary', 592),
 ('Debate', 587),
 ('TrumpTrain', 479),
 ('trump', 438),
 ('JusticeForJuanita', 425),
 ('NeverTrump', 363)]
```

In [11]:

```
#print an example
print('The top 10 hashtags for tweets2.json are:')
topnhashtags(new_tweets2_json,10)
```

```
The top 10 hashtags for tweets2.json are:
```

Out[11]:

```
[('debate', 13026),
 ('Debate', 3371),
 ('debates', 1440),
 ('Trump', 1306),
 ('Debates2016', 901),
 ('DEBATE', 675),
 ('Debates', 609),
 ('debatenight', 431),
 ('MAGA', 403),
 ('DebateNight', 358)]
```

4. Write a function to return a data frame which contains the top n tweeted hashtags of a given hashtag list. The columns in the returned data frame are 'hashtag' and 'freq'.

In [12]:

```
def dataframe(file,n):
    a = topnhashtags(file,n)
    result = []
    for i in a:
        result.append({'hashtag': i[0], 'freq': i[1]})
    result = pd.DataFrame(result)
    result = result[['hashtag','freq']]
    return result
```

In [13]:

```
#print an example
print('The top 5 hashtags for tweets1.json are:')
dataframe(new_tweets1_json,5)
```

```
The top 5 hashtags for tweets1.json are:
```

Out[13]:

	hashtag	freq
0	Trump	3140
1	debate	1697
2	TrumpTapes	1478
3	MAGA	1095
4	Hillary	592

```
In [14]:
```

```
#print an example
print('The top 5 hashtags for tweets2.json are:')
dataframe(new_tweets2_json,5)
```

```
The top 5 hashtags for tweets2.json are:
```

```
Out[14]:
```

	hashtag	freq
0	debate	13026
1	Debate	3371
2	debates	1440
3	Trump	1306
4	Debates2016	901

5. Use the function to produce a horizontal bar chart of the top n tweeted hashtags of the tweets1.json and tweets2.json respectively.

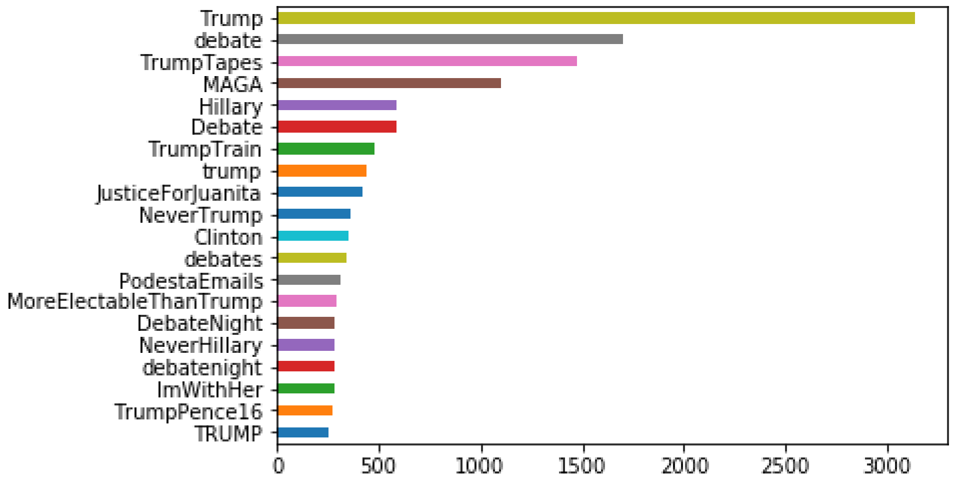
Method : Using pandas default plot function Result for tweets 1

```
In [15]:
```

```
example1 = dataframe(new_tweets1_json,20).set_index('hashtag').to_dict()['freq']
example1 = pd.Series(example1,index=example1.keys())
example1.iloc[:20].sort_values(ascending=True).plot(kind='barh')
```

```
Out[15]:
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c5348780>
```



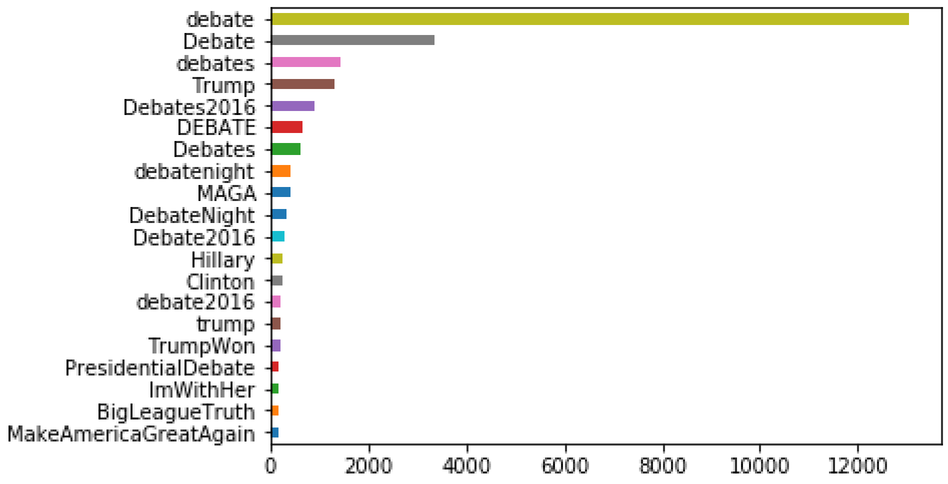
Result for tweet 2

In [16]:

```
example2 = dataframe(new_tweets2_json,20).set_index('hashtag').to_dict()['freq']
example2 = pd.Series(example2,index=example2.keys())
example2.iloc[:20].sort_values(ascending=True).plot(kind='barh')
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d1c682e8>
```



6. Find the max time and min time of the tweets1.json and the tweets2.json respectively.

In [17]:

```
#Define the timestamp list
def timelist(file):
    timelist = file['CreatedDate']
    return timelist
tweets1_time = timelist(new_tweets1_json)
tweets2_time = timelist(new_tweets2_json)
```

Tweet 1 The number of qualified tweet is: 9296 the number of qualified retweeted tweet is: 8551

In [18]:

```
print('For the total tweets (including retweet):')
print('the max time is:',max(tweets1_time),'\n','the min time is:',min(tweets1_t
ime),'\n')
print('For the original number of qualified tweets (not including retweet):')
slice1 = tweets1_time[9296:]
print('the max time is:',max(slice1),'\n','the min time is:',min(slice1),'\n')
```

```
For the total tweets (including retweet):
the max time is: 2016-10-10 00:52:27
 the min time is: 2014-02-02 00:19:56

For the original number of qualified tweets (not including retweet):
the max time is: 2016-10-10 00:52:27
 the min time is: 2016-10-09 01:14:31
```

Tweet 2 The number of qualified tweet is: 10961 the number of qualified retweeted tweet is: 13454

[redacted]

```
In [19]:
```

```
print('For the total tweets (including retweet):')
print('the max time is:',max(tweets2_time),'\n','the min time is:',min(tweets2_t
ime),'\n')
print('For the original number of qualified tweets (not including retweet):')
slice2 = tweets1_time[10961:]
print('the max time is:',max(slice2),'\n','the min time is:',min(slice2),'\n')
```

```
For the total tweets (including retweet):
the max time is: 2016-10-10 03:32:15
 the min time is: 2016-03-10 06:13:35

For the original number of qualified tweets (not including retweet):
the max time is: 2016-10-10 00:52:27
 the min time is: 2016-10-09 08:39:21
```

7. For each interval defined by (min time, max time), divide it into 10 equally spaced periods respectively

Using all the tweets (including retweeted ones)

Method: Use built-in function in pandas

```
In [20]:
```

```
spacedperiods1 = pd.date_range(start=min(tweets1_time), end=max(tweets1_time), p
eriods=11)
spacedperiods1
```

```
Out[20]:
```

```
DatetimeIndex([       '2014-02-02 00:19:56', '2014-05-11 02:47:11.10
0000',
               '2014-08-17 05:14:26.200000', '2014-11-23 07:41:41.30
0000',
               '2015-03-01 10:08:56.400000', '2015-06-07 12:36:11.50
0000',
               '2015-09-13 15:03:26.600000', '2015-12-20 17:30:41.70
0000',
               '2016-03-27 19:57:56.800000', '2016-07-03 22:25:11.90
0000',
                      '2016-10-10 00:52:27'],
              dtype='datetime64[ns]', freq=None)
```

In [21]:

```
spacedperiods2 = pd.date_range(start=min(tweets2_time), end=max(tweets2_time), p
eriods=11)
spacedperiods2
```

Out[21]:

```
DatetimeIndex(['2016-03-10 06:13:35', '2016-03-31 15:33:27',
               '2016-04-22 00:53:19', '2016-05-13 10:13:11',
               '2016-06-03 19:33:03', '2016-06-25 04:52:55',
               '2016-07-16 14:12:47', '2016-08-06 23:32:39',
               '2016-08-28 08:52:31', '2016-09-18 18:12:23',
               '2016-10-10 03:32:15'],
              dtype='datetime64[ns]', freq=None)
```

8. For a given collection of tweets, write a function to return a data frame with two columns, hashtags and their time of creation. Use the function to produce data frames for the tweets1.json and the tweets2.json. Use pandas.cut or else, create a third column 'level' in each data frame which cuts the time of creation by the corresponding interval obtained in part 7 respectively.

In [22]:

```
def cleanthehashtags(file):
# change the hashtag from list of list of dict to list of list
    for i in file['hashtags']:
        for m in range(len(i)):
            i[m] = i[m]['text']
cleanthehashtags(new_tweets1_json)
cleanthehashtags(new_tweets2_json)
```

In [23]:

```
def hashtag_time(file1,file2,file3):
    # create a dataframe with two columns, hashtags and their time of creation.
    result = pd.DataFrame()
    result['creationtime'] = pd.Series(i for i in file1)
    result['hashtag'] = pd.Series(i for i in file2['hashtags'])
    a = result.drop('hashtag', axis=1)
    dateinterval = [i for i in range(1,11)]
    result['level'] = pd.cut(a.creationtime.astype(np.int64), bins=file3.astype(
np.int64), labels= dateinterval)
    return result
time_tag_level_tweets1 = hashtag_time(tweets1_time,new_tweets1_json,spacedperiod
s1)
time_tag_level_tweets2 = hashtag_time(tweets2_time,new_tweets2_json,spacedperiod
s2)
time_tag_level_tweets1['level'][724] = 1
time_tag_level_tweets2['level'][10716] = 1
```

In [24]:

```
# print the result for tweets1.json
time_tag_level_tweets1[:5]
```

Out[24]:

	creationtime	hashtag	level
0	2016-10-08 04:28:50	[TrumpTapes]	10
1	2016-10-08 20:48:44	[TrumpTrain, NYC, MakeAmericaGreatAgain]	10
2	2016-10-06 20:44:45	[Trump, MAGA3X, MAGA]	10
3	2016-10-08 00:42:51	[PodestaEmails]	10
4	2016-10-08 13:06:02	[BillClintonIsARapist]	10

In [25]:

```
# print the result for tweets2.json
time_tag_level_tweets2[:5]
```

Out[25]:

	creationtime	hashtag	level
0	2016-10-10 02:40:29	[debate]	10
1	2016-10-10 01:31:30	[Debate]	10
2	2016-10-10 02:40:03	[debate]	10
3	2016-10-10 01:44:00	[sniffles, debate]	10
4	2016-10-10 02:40:41	[debate]	10

9. Use pandas.pivot or else, create a numpy array or a pandas data frame whose rows are time period defined in part 7 and whose columns are hashtags. The entry for the ith time period and jth hashtag is the number of occurrence of the jth hashtag in ith time period. Fill the entry without data by zero. Do this for tweets1.json and the tweets2.json respectively.

In [26]:

```
#get the distinct hashtags
def distincthashtags(file):
    hashtags = []
    for i in file['hashtags']:
        for item in i:
            hashtags.append(item)
    hashtags = list(set(hashtags))
    return hashtags
distinctlist1 = distincthashtags(new_tweets1_json)
distinctlist2 = distincthashtags(new_tweets2_json)
```

In [27]:

```
#create a dataframe whose columns are hashtags and rows are time period
def countframe(file1,file2):
    d = pd.DataFrame(0, index=np.arange(1,11), columns=file1)
    for i in range(len(file2)):
        for item in file2.iloc[i]['hashtag']:
            d[item][file2.iloc[i]['level']] = d[item][file2.iloc[i]['level']]+1
    return d
hashtagandperiod1 = countframe(distinctlist1,time_tag_level_tweets1)
hashtagandperiod2 = countframe(distinctlist2,time_tag_level_tweets2)
```

In [28]:

```
#print an example
hashtagandperiod1['Trump']
```

Out[28]:

```
1        0
2        0
3        0
4        0
5        0
6        0
7        2
8       11
9        3
10    3124
Name: Trump, dtype: int64
```

10. Following part 9, what is the number of occurrence of hashtag 'trump' in the sixth period in the tweets1.json? What is the number of occurrence of hashtag 'trump' in the eighth period in the tweets2.json?

In [29]:

```
hashtagandperiod1['trump'][6]
```

Out[29]:

```
0
```

In [30]:

```
hashtagandperiod2['trump'][8]
```

Out[30]:

```
0
```

11. Using the tables obtained in part 9, we can also find the total number of occurrences for each hashtag. Rank these hashtags in decreasing order and obtain a time plot for the top 20 hashtags in a single graph. Rescale the size of the graph so that it is not too small nor too large. Do this for both tweets1.json and the tweets2.json respectively.

In [31]:

```
# add the total number to each hashtags
hashtagandperiod1.loc['Total']= hashtagandperiod1.sum()
hashtagandperiod2.loc['Total']= hashtagandperiod2.sum()
```

In [32]:

```
# sort the columns by the total hashtag count and get the first 20 columns
first20_tweets1 = hashtagandperiod1.T.sort_values(by="Total" , ascending=False).
T
first20_tweets1 = first20_tweets1.iloc[:, : 20]
first20_tweets2 = hashtagandperiod2.T.sort_values(by="Total" , ascending=False).
T
first20_tweets2 = first20_tweets2.iloc[:, : 20]
```

In [32]:

```
# for convience for plotting line, add the x column in each dataframe
first20_tweets1['x'] = range(1,12)
first20_tweets2['x'] = range(1,12)
```

Figure for tweet1

In [34]:

```
# Initialize the figure
plt.style.use('seaborn-darkgrid')
# create a color palette
palette = plt.get_cmap('Set1')
# multiple line plot
num = 0
for column in first20_tweets1.drop('x', axis=1):
    num += 1
    # Find the right spot on the plot
    plt.subplot(5,4, num)
    # Plot the lineplot
    plt.plot(first20_tweets1['x'], first20_tweets1[column], marker='', color=pal
ette(num), linewidth=1.9, alpha=0.9, label=column)
    # Same limits for everybody!
    plt.xlim(0,10)
    plt.ylim(-50,500)
    # Not ticks everywhere
    if num in range(17) :
        plt.tick_params(labelbottom='off')
    if num not in [1,5,9,13,17] :
        plt.tick_params(labelleft='off')
    # Add title
    plt.title(column, loc='left', fontsize=12, fontweight=0, color=palette(num))

# general title
plt.suptitle("Hashtags occurrence over time(tweets1)", fontsize=13, fontweight=0
, color='black', style='italic')
```

[redacted]

Out[34]:

```
Text(0.5,0.98,'Hashtags occurrence over time(tweets1)')
```

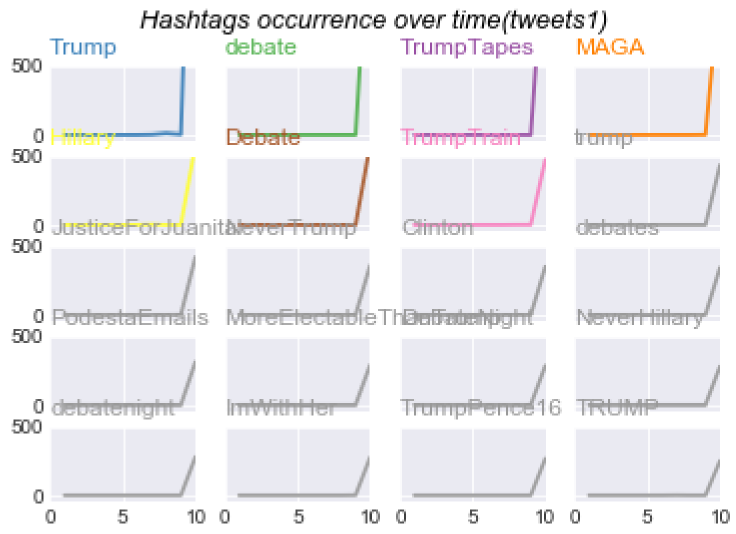


figure for tweet 2

In [35]:

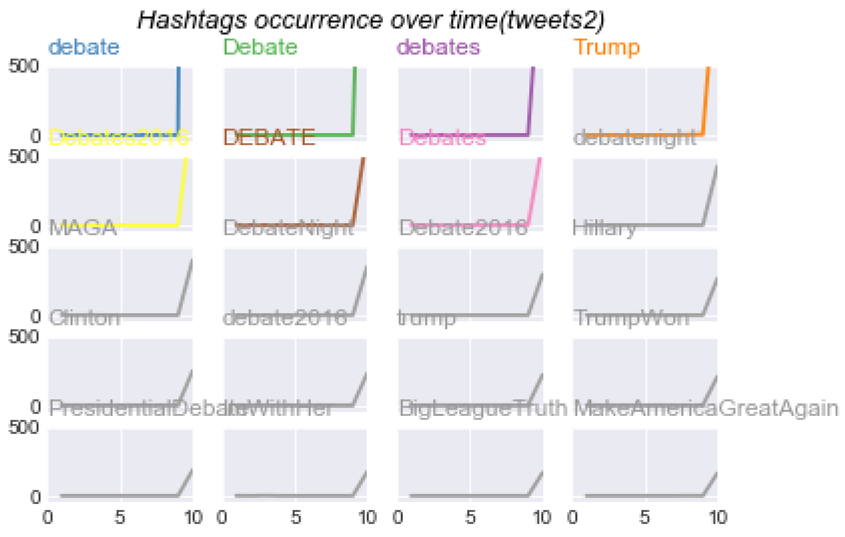
```
# Initialize the figure
plt.style.use('seaborn-darkgrid')
# create a color palette
palette = plt.get_cmap('Set1')
# multiple line plot
num = 0
for column in first20_tweets2.drop('x', axis=1):
    num += 1
    # Find the right spot on the plot
    plt.subplot(5,4, num)
    # Plot the lineplot
    plt.plot(first20_tweets2['x'], first20_tweets2[column], marker='', color=pal
ette(num), linewidth=1.9, alpha=0.9, label=column)
    # Same limits for everybody!
    plt.xlim(0,10)
    plt.ylim(-50,500)
    # Not ticks everywhere
    if num in range(17) :
        plt.tick_params(labelbottom='off')
    if num not in [1,5,9,13,17] :
        plt.tick_params(labelleft='off')
    # Add title
    plt.title(column, loc='left', fontsize=12, fontweight=0, color=palette(num))

# general title
plt.suptitle("Hashtags occurrence over time(tweets2)", fontsize=13, fontweight=0
, color='black', style='italic')
```

[redacted]

```
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

Out[35]:

```
Text(0.5,0.98,'Hashtags occurrence over time(tweets2)')
```



12. The zip_codes_states.csv contains city, state, county, latitude and longitude of US. Read the file.

In [36]:

```
os.chdir("[redacted]
[redacted])
zip_codes_states = pd.read_csv('zip_codes_states.csv')
#zip_codes_states = zip_codes_states.drop('zip_code',axis=1)
zip_codes_states[10:16]
```

Out[36]:

	zip_code	latitude	longitude	city	state	county
10	612	18.450674	-66.698262	Arecibo	PR	Arecibo
11	613	18.458093	-66.732732	Arecibo	PR	Arecibo
12	614	18.429675	-66.674506	Arecibo	PR	Arecibo
13	616	18.444792	-66.640678	Bajadero	PR	Arecibo
14	617	18.447092	-66.544255	Barceloneta	PR	Barceloneta
15	622	17.998531	-67.187318	Boqueron	PR	Cabo Rojo

13. Select tweets in tweets1.json and the tweets2.json with locations only in the zip_codes_states.csv. Remove also the location 'london'.

In [37]:

```
# remove the location information that after the comma. For example: remain only
 'Washington' in 'Washington, USA'
def cleanlocation(file):
    file['location'] = [i.split(',')[0] for i in file['location']]
cleanlocation(new_tweets1_json)
cleanlocation(new_tweets2_json)

# Get the unique list of cities
citylist = list(set(zip_codes_states['city']))
citylist.remove('London')
```

In [38]:

```
# Get the index for tweets with qualified location
def getqualifiedlocation(file):
    indexlist = []
    for i in range(len(file['location'])):
        if file['location'][i] in citylist:
            indexlist.append(i)
    return indexlist
indexlist1 = getqualifiedlocation(new_tweets1_json)
indexlist2 = getqualifiedlocation(new_tweets2_json)
```

In [39]:

```
# Get the result list of qualified locations
def gettweetwithlocation(file1,file2):
    # return the result dictionary
    resultlist = []
    for i in file1:
        resultlist.append(file2['location'][i])
    return resultlist
resultlocation_tweet1 = gettweetwithlocation(indexlist1, new_tweets1_json)
resultlocation_tweet2 = gettweetwithlocation(indexlist2, new_tweets2_json)
```

In [40]:

```
print('The number of qualified locations in tweet1 is:',len(resultlocation_tweet
1))
print('The number of qualified locations in tweet2 is:',len(resultlocation_tweet
2))
```

```
The number of qualified locations in tweet1 is: 6676
The number of qualified locations in tweet2 is: 11006
```

14. Find the top 20 tweeted locations in both tweets1.json and the tweets2.json respectively.

In [41]:

```
def topnlocations(file,n):
    countertdata = Counter(file)
    return countertdata.most_common(n)
```

In [42]:

```
#print the result
print('The top 20 locations for tweets1.json are:')
topnlocations(resultlocation_tweet1, 20)
```

```
The top 20 locations for tweets1.json are:
```

Out[42]:

```
[('New York', 641),
 ('Washington', 533),
 ('Los Angeles', 399),
 ('California', 374),
 ('Florida', 218),
 ('Brooklyn', 156),
 ('Chicago', 138),
 ('Austin', 117),
 ('San Francisco', 85),
 ('Atlanta', 76),
 ('Seattle', 69),
 ('Las Vegas', 67),
 ('Tennessee', 66),
 ('Cleveland', 63),
 ('Houston', 63),
 ('Virginia Beach', 62),
 ('Ohio', 60),
 ('Charlotte', 59),
 ('Virginia', 54),
 ('Nashville', 54)]
```

In [43]:

```
#print the result
print('The top 20 locations for tweets2.json are:')
topnlocations(resultlocation_tweet2, 20)
```

```
The top 20 locations for tweets2.json are:
```

Out[43]:

```
[('New York', 1716),
 ('Los Angeles', 1052),
 ('Washington', 992),
 ('Brooklyn', 561),
 ('California', 264),
 ('Chicago', 247),
 ('Las Vegas', 212),
 ('Tennessee', 179),
 ('Austin', 173),
 ('Florida', 172),
 ('Detroit', 137),
 ('Boston', 126),
 ('Virginia Beach', 114),
 ('Seattle', 113),
 ('San Francisco', 113),
 ('Houston', 96),
 ('Toronto', 82),
 ('Miami', 80),
 ('Atlanta', 80),
 ('Phoenix', 78)]
```

15. Since there are multiple (lon, lat) pairs for each location, write a function to return the average lon and the average lat of a given location. Use the function to generate the average lon and the average lat for every locations in tweets1.json and the tweets2.json.

In [44]:

```
# Define the dataframe including the average latitude and longitude information
 for all the locations
latitude_avg = zip_codes_states['latitude'].groupby(zip_codes_states['city']).me
an()
longtitude_avg = zip_codes_states['longitude'].groupby(zip_codes_states['city'])
.mean()
latitude_avg = latitude_avg.to_frame()
longtitude_avg = longtitude_avg.to_frame()
longtitude_avg['latitude'] = latitude_avg
```

16. Combine tweets1.json and tweets2.json. Then, create data frames which contain locations, counts, longitude and latitude in tweets1.json and the tweets2.json.

In [45]:

```
# Combine all the location together
combinelocation = resultlocation_tweet1 + resultlocation_tweet2
cityunique = list(set(combinelocation))

# append citylist to the dataframe in question 15
citylist.append('London')
citylist.sort()
longtitude_avg['city'] = citylist
# append zero list
#zeros = [0] * len(citylist)
#longtitude_avg['count'] = zeros

# Delete the locations not in tweets
finallocation = longtitude_avg[longtitude_avg['city'].isin(cityunique)]
```

In [46]:

```
countlocation = list(Counter(combinelocation).items())
countlocation.sort(key= lambda x:x[0])
countlist = [x[1] for x in countlocation]
finallocation['count'] = countlist
```

[redacted]

[redacted]

In [47]:

```
# Print an example
finallocation[:10]
```

Out[47]:

	longitude	latitude	city	count
city				
Aberdeen	-94.216481	40.621977	Aberdeen	2
Abilene	-99.574413	32.969564	Abilene	3
Ada	-89.473201	41.192957	Ada	2
Agana Heights	144.786297	13.444257	Agana Heights	1
Agua Dulce	-97.530388	27.836111	Agua Dulce	1
Aiken	-84.447299	33.661454	Aiken	2
Akron	-82.612012	40.977070	Akron	3
Alabama	-78.184813	42.998052	Alabama	57
Alachua	-82.484006	29.802745	Alachua	1
Alameda	-122.253923	37.752826	Alameda	1

17. Using the sharpfile of US states st99_d00 and the help of the website, produce the following graphs.

In [48]:

```
# list contaning Longitude, Latitude, and Count
a = finallocation.values.tolist()
for i in a:
    i.remove(i[2])
a[:10]
```

Out[48]:

```
[[-94.21648122222221, 40.62197722222222, 2],
 [-99.57441345454544, 32.969564, 3],
 [-89.473200625, 41.192957375, 2],
 [144.786297, 13.444257, 1],
 [-97.530388, 27.836111, 1],
 [-84.447299, 33.66145357142857, 2],
 [-82.61201150000002, 40.97706997368422, 3],
 [-78.184813, 42.998052, 57],
 [-82.48400649999999, 29.802745, 1],
 [-122.253923, 37.7528255, 1]]
```

In [49]:

```
import matplotlib.patches as mpatches
from mpl_toolkits.basemap import Basemap
```

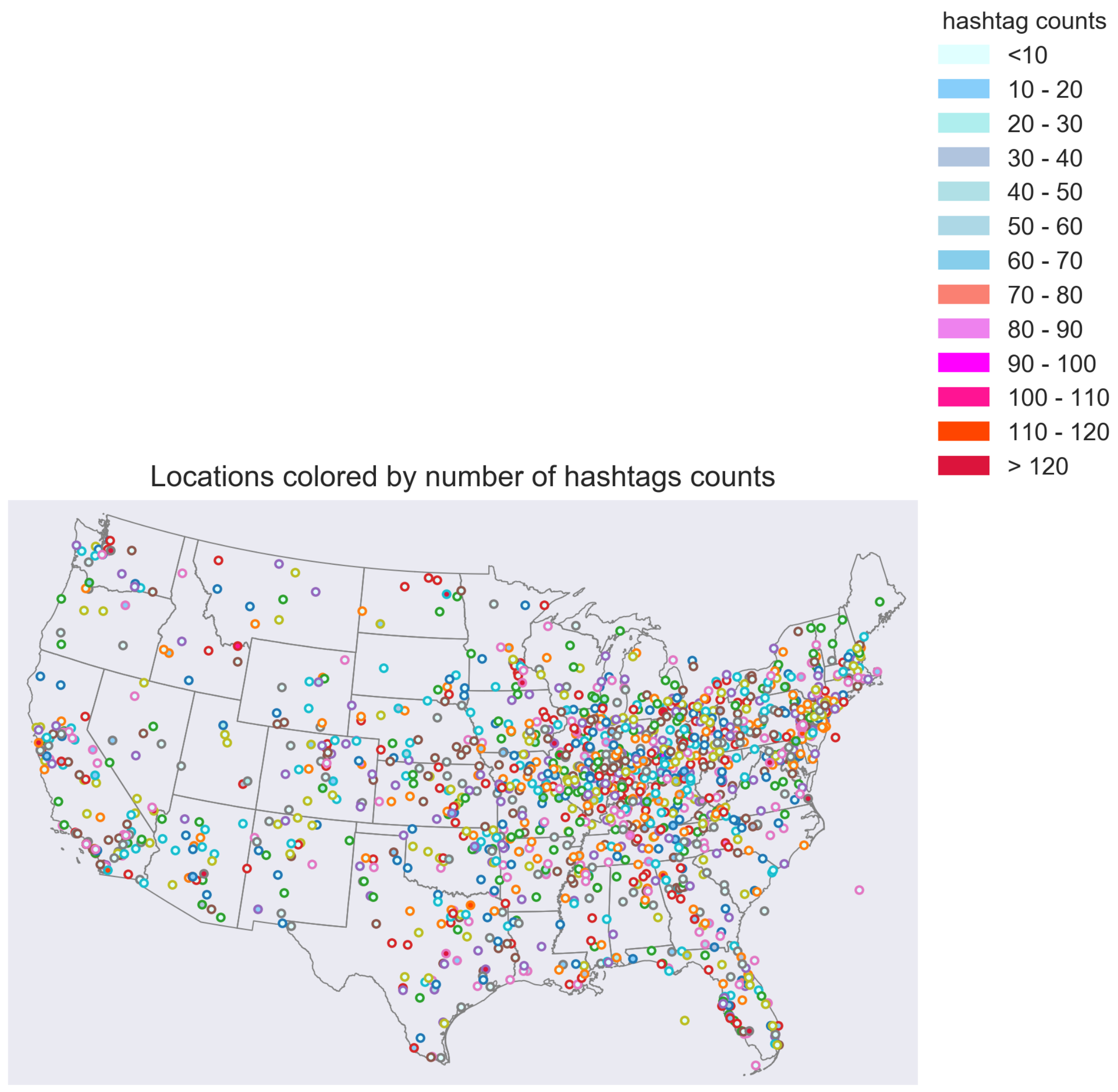
In [50]:

```
# Function to map the colors as a function of scaled populations
def pltcolor(count):
    if count < 10:
        col = 'lightcyan'
    elif count >= 10 and count < 20:
        col = 'lightskyblue'
    elif count >= 20 and count < 30:
        col = 'paleturquoise'
    elif count >= 30 and count < 40:
        col = 'lightsteelblue'
    elif count >= 40 and count < 50:
        col = 'powderblue'
    elif count >= 50 and count < 60:
        col = 'lightblue'
    elif count >= 60 and count < 70:
        col = 'skyblue'
    elif count >= 70 and count < 80:
        col = 'salmon'
    elif count >= 80 and count < 90:
        col = 'violet'
    elif count >= 90 and count < 100:
        col = 'magenta'
    elif count >= 100 and count < 110:
        col = 'deeppink'
    elif count >= 110 and count < 120:
        col = 'orangered'
    else:
        col = 'crimson'
    return col
```

In [51]:

```
# Set the dimension of the figure
my_dpi=300
plt.figure(figsize=(2000/my_dpi, 1800/my_dpi), dpi=my_dpi)

#codes to produce the map
m = Basemap(llcrnrlon=-119,llcrnrlat=22,urcrnrlon=-64,urcrnrlat=49,projection='l
cc',lat_1=33,lat_2=45,lon_0=-95)

#read the shapefile for us states
#os.chdir("[redacted]")
m.readshapefile('st99_d00', 'states', drawbounds=True, color='grey')

for i in a:
    x = i[0]
    y = i[1]
    count = i[2]
    x, y = m(x,y)
    m.plot(x, y, marker='o', markerfacecolor=pltcolor(count), markersize=3)

#create legend
levels = [mpatches.Patch(color='lightcyan', label='<10'),
          mpatches.Patch(color='lightskyblue', label='10 - 20'),
          mpatches.Patch(color='paleturquoise', label='20 - 30'),
          mpatches.Patch(color='lightsteelblue', label='30 - 40'),
          mpatches.Patch(color='powderblue', label='40 - 50'),
          mpatches.Patch(color='lightblue', label='50 - 60'),
          mpatches.Patch(color='skyblue', label='60 - 70'),
          mpatches.Patch(color='salmon', label='70 - 80'),
          mpatches.Patch(color='violet', label='80 - 90'),
          mpatches.Patch(color='magenta', label='90 - 100'),
          mpatches.Patch(color='deeppink', label='100 - 110'),
          mpatches.Patch(color='orangered', label='110 - 120'),
          mpatches.Patch(color='crimson', label='> 120')]
plt.legend(handles=levels, title='hashtag counts',bbox_to_anchor=(1.0, 1.0))

plt.title("Locations colored by number of hashtags counts")
plt.tight_layout()
plt.show()
#darker colors indicate higher hashtags count
```

[redacted]



18. (Optional) Using polygon patches and the help of the website, produce the following graph.

In [199]:

```
# get the distinct state list
statelist = list(zip_codes_states['state'].unique())

#create a dataframe whose columns are hashtags and rows are time period
stateandcount = pd.DataFrame(0, index=statelist, columns=range(1,2))

# add the state information to the dataframe in question 17
cityandstate = zip_codes_states[['city','state']]
cityandstate = cityandstate.loc[~cityandstate['city'].duplicated()]
cityandstate = cityandstate[cityandstate['city'].isin(list(finallocation['city'
]))]
cityandstate = cityandstate.sort_values(by='city')
finallocation['state'] = list(cityandstate['state'])

#create a dataframe whose columns are state and rows are hashtagcount
stateandcount = pd.DataFrame(0, index=statelist, columns=range(1,2))
for i in range(len(finallocation)):
    stateandcount.loc[finallocation.iloc[i]['state']][1] = stateandcount.loc[fin
allocation.iloc[i]['state']][1] + finallocation.iloc[i]['count']
```

[redacted]

In [243]:

```
stateabbre = {'AL':'Alabama', 'AK':'Alaska', 'AZ':'Arizona', 'AR':'Arkansas', 'C
A':'California',
              'CO':'Colorado', 'CT':'Connecticut', 'DE':'Delaware', 'FL':'Florid
a', 'GA':'Georgia',
              'HI':'Hawaii', 'ID':'Idaho', 'IL':'Illinois', 'IN':'Indiana', 'IA'
:'Iowa',
              'KS':'Kansas', 'KY':'Kentucky', 'LA':'Louisiana', 'ME':'Maine', 'M
D':'Maryland',
              'MA':'Massachusetts', 'MI':'Michigan', 'MN':'Minnesota', 'MS':'Mis
sissippi', 'MO':'Missouri',
              'MT':'Montana', 'NE':'Nebraska', 'NV':'Nevada', 'NH':'New Hampshir
e', 'NJ':'New Jersey',
              'NM':'New Mexico', 'NY':'New York', 'NC':'North Carolina', 'ND':'N
orth Dakota', 'OH':'Ohio',
              'OK':'Oklahoma', 'OR':'Oregon', 'PA':'Pennsylvania', 'RI':'Rhode I
sland', 'SC':'South Carolina',
              'SD':'South Dakota', 'TN':'Tennessee', 'TX':'Texas', 'UT':'Utah',
'VT':'Vermont',
              'VA':'Virginia', 'WA':'Washington', 'WV':'West Virginia', 'WI':'Wi
sconsin', 'WY':'Wyoming',
              'PR':'Puerto Rico', 'GU':'Guam'}
```

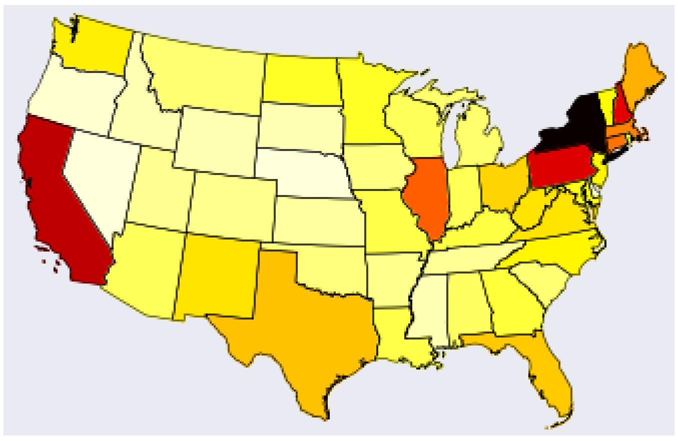
In [ ]:

```
# To make the data readable, convert the dataframe to dictionary
stateandcount['state'] = statelist
# delete the zero count state
stateandcount = stateandcount[(stateandcount != 0).all(1)]
state_count = {}
for i in range(len(stateandcount)):
    state_count[stateabbre[stateandcount.iloc[i]['state']]] = stateandcount.iloc
[i][1]
```

In [251]:

```
from matplotlib.colors import rgb2hex
from matplotlib.patches import Polygon
# Lambert Conformal map of lower 48 states.
m = Basemap(llcrnrlon=-119,llcrnrlat=22,urcrnrlon=-64,urcrnrlat=49,projection='l
cc',lat_1=33,lat_2=45,lon_0=-95)

#read the shapefile for us states
shp_info = m.readshapefile('st99_d00','states',drawbounds=True)

# choose a color for each state based on hashtags count.
colors={}
statenames=[]
cmap = plt.cm.hot # use 'hot' colormap
vmin = stateandcount[1].min(); vmax = stateandcount[1].max() # set range.
for shapedict in m.states_info:
    statename = shapedict['NAME']
    # skip DC and Puerto Rico.
    if statename not in ['District of Columbia']:
        pop = state_count[statename]
        # calling colormap with value between 0 and 1 returns
        # rgba value.  Invert color range (hot colors are high
        # take sqrt root to spread out colors more.
        colors[statename] = cmap(1.-np.sqrt((pop-vmin)/(vmax-vmin)))[:3]
    statenames.append(statename)
# cycle through state names, color each one.
ax = plt.gca() # get current axes instance
for nshape,seg in enumerate(m.states):
    # skip DC and Puerto Rico.
    if statenames[nshape] not in ['District of Columbia']:
        color = rgb2hex(colors[statenames[nshape]])
        poly = Polygon(seg,facecolor=color,edgecolor=color)
        ax.add_patch(poly)
plt.title('Filling State Polygons by Hashtag Counts')
plt.show()
```

Filling State Polygons by Hashtag Counts



Question 3 (extract hurricane paths)

The website provides hurricane paths data from 1850. We work to extract hurricane paths for a given year.http://weather.unisys.com (http://weather.unisys.com)

1. Since the link contains the hurricane information varies with years and the information is contained in multiple pages, we need to know the starting page and the total number of pages for a given year. What is the appropriate starting page for year = '2017'?

In [252]:

```
def StartPage(year):
    website = 'http://weather.unisys.com/hurricanes/search?field_ocean_target_id
=All&year[0]='\
              +str(year)+'-01-01&year[1]='+str(year)+'-12-31&year[2]=2017&catego
ry=All&type=All&items_per_page=12&page=0'
    print('The starting page should be:')
    return website
```

In [253]:

```
StartPage(2017)
```

```
The starting page should be:
```

Out[253]:

```
'http://weather.unisys.com/hurricanes/search?field_ocean_target_id=A
ll&year[0]=2017-01-01&year[1]=2017-12-31&year[2]=2017&category=All&t
ype=All&items_per_page=12&page=0'
```

2. In order to solve the second question, we try inputting a large number as the number of pages for a given year. Use an appropriate number, write a function to extract all links each of which holds information on the hurricanes in '2017'.

In [302]:

```
def getlinks(year, pageno):
    from bs4 import BeautifulSoup
    from urllib.request import urlopen
    sublinks = []
    for i in range(pageno):
        html = urlopen('http://weather.unisys.com/hurricanes/search?field_ocean_
target_id=All&year[0]='+str(year)+'-01-01&year[1]='+str(year)+'-12-31&year[2]='+
str(year)+'&category=All&type=All&sort_bef_combine=field_start_date_value%20DESC
&items_per_page=12&sort_by=field_start_date_value&sort_order=DESC&page='+str(i))
        soup = BeautifulSoup(html)
        links = []
        for link in soup.findAll('a', href=True):
            links.append(link)
        #Only contains the chunk from the result of findall where hold informati
on on the hurricane in '2017'
        links = links[-28:-15]
        sublinks.extend(links)
    finalresult = []
    for link in sublinks:
        if '/hurricanes/'+str(year)+'/' in link.get('href'):
            finalresult.append(link.get('href'))
    finalresult = [str(i) for i in finalresult]
    finalresult = ['http://weather.unisys.com'+i for i in finalresult]
    return finalresult

hurricanes_2017 = getlinks(2017, 6)
```

In [303]:

```
print('There are %a links in the result' %len(hurricanes_2017))
print('Print 10 links as example:')
hurricanes_2017[:10]
```

```
There are 64 links in the result
Print 10 links as example:
```

Out[303]:

```
['http://weather.unisys.com/hurricanes/2017/south-indian/hilda',
 'http://weather.unisys.com/hurricanes/2017/north-indian/four',
 'http://weather.unisys.com/hurricanes/2017/south-indian/dahlia',
 'http://weather.unisys.com/hurricanes/2017/north-indian/ockhi',
 'http://weather.unisys.com/hurricanes/2017/north-indian/twentynin
e',
 'http://weather.unisys.com/hurricanes/2017/atlantic/rina',
 'http://weather.unisys.com/hurricanes/2017/atlantic/philippe',
 'http://weather.unisys.com/hurricanes/2017/east-pacific/selma',
 'http://weather.unisys.com/hurricanes/2017/atlantic/ophelia',
 'http://weather.unisys.com/hurricanes/2017/atlantic/nate']
```

3. Some of the collected links provide summary of hurricanes which do not lead to correct tables. Remove those links.

Note: In the question 1, some of the unrelated links have already been moved. In question 3, I will use the result from question 2 to further extract the useful links.

```
In [304]:
```

```
usefullinks = deepcopy(hurricanes_2017)
usefullinks = usefullinks[:58]
print('There are %a useful links left in the result' %len(usefullinks))
```

```
There are 58 useful links left in the result
```

4. For each valid hurricane link, it contains four set of information:

- Date
- Hurricane classification
- Hurricane name
- A table of hurricane positions over dates

Since the entire information is contained in a text file provided in the corresponding webpage defined by the link, write a function to download the text file and read (without saving it to a local directory) the text file (at this moment, you don't need to convert the data to other format).

```
In [305]:
```

```
import requests
def get_text(url):
    page = requests.get(url)
    content = page.text
    return content
```

```
In [306]:
```

```
example = get_text(usefullinks[10])
example[:1000]
```

```
Out[306]:
```

```
'<!DOCTYPE html>\n<html  lang="en" dir="ltr" prefix="content: htt
p://purl.org/rss/1.0/modules/content/  dc: http://purl.org/dc/terms/
  foaf: http://xmlns.com/foaf/0.1/  og: http://ogp.me/ns#  rdfs: htt
p://www.w3.org/2000/01/rdf-schema#  schema: http://schema.org/  sio
c: http://rdfs.org/sioc/ns#  sioct: http://rdfs.org/sioc/types#  sko
s: http://www.w3.org/2004/02/skos/core#  xsd: http://www.w3.org/200
1/XMLSchema# ">\n  <head>\n    <meta charset="utf-8" />\n<script>(fu
nction(i,s,o,g,r,a,m){i["GoogleAnalyticsObject"]=r;i[r]=i[r]||functi
on(){(i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.cr
eateElement(o),m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.pa
rentNode.insertBefore(a,m)})(window,document,"script","https://www.g
oogle-analytics.com/analytics.js","ga");ga("create", "UA-58467683-
7", {"cookieDomain":"auto"});ga("set", "anonymizeIp", true);ga("sen
d", "pageview");</script>\n<meta name="title" content="RAMON | Unisy
s Weather" />\n<link rel="canonical" href="http://weather.unisy'
```

5. With the downloaded contents, write a function to convert the contents to a list of dictionaries. Each dictionary in the list contains the following keys: Date, Category of the hurricane, Name of the hurricane and a table of information for the hurricane path. Convert the Date in each dictionary to datetime objects. Since the recorded times for the hurricane paths used the Z-time, convert it to datetime object with the help of http://www.theweatherprediction.com/basic/ztime (http://www.theweatherprediction.com/basic/ztime)

In [316]:

```
#function to turn a page into a list of useful information
def get_dict(url,year):
    content = get_text(url)
    c = content.split('<td headers="view-field-adv-table-column" class="views-fi
eld views-field-field-forecast-position views-field-field-adv is-active">')
    b = [x.split('            </td>\n                  </tr>\n')[0] for x in c]
    c0 = [x.split('           </td>\n
                                                     ') for x in b]
    c0[0] = b[0].split('<span class="field field--name-title field--type-string
 field--label-hidden field--node--title field--node--title--storm">')[1]
    c0[0] = c0[0].split('<div class="field--item">\n      <time datetime=')
    c0[0][0] = c0[0][0].split('</div>\n           \n\n          ')[0]
    c0[0][0] = c0[0][0].split('</span></span>\n  </h1>\n\n              <section
 id="block-entityviewcontent" class="block--title-style-default block--theme-def
ault settings-tray-editable block--entity-viewnode--page-header block--entity-vi
ew block--entity-viewnode block--node-page-header block block-entity-viewnode bl
ock-region-page-header" data-drupal-settingstray="editable">\n      <div class
="block-wrapper block-wrapper--block-region-page-header">\n                 \n
                  \n          <div class="block-content">\n             \n
                <div class="field field--name-field-storm-category field--type-e
ntity-reference field--label-hidden field--node--field-storm-category field--nod
e--field-storm-category--storm field--item">')
    c0[0][1] = c0[0][1].split('</time>\n\n    </div>\n')[0]
    c0[0][0][0] = c0[0][0][0].replace(' <span class="storm-name">',' ')
    table = []
    for i in c0[1:]:
        for m in range(1, len(i)):
            i[m] = i[m].split('">')[1]
        i[1] = i[1].replace('Z','')
        i[1] = str(year)+'/'+i[1]
        i[1] = datetime.strptime(i[1],'%Y/%m/%d/%H')
        table.append(i)
    c0.insert(1,table)
    c0 = c0[:2]
    dict = {}
    dict['Name'] = c0[0][0][0]
    dict['Category'] = c0[0][0][1]
    dict['Date'] = [c0[1][0][1],c0[1][-1][1]]
    dict['TableOfPath'] = pd.DataFrame(c0[1])
    dict['TableOfPath'].rename(columns = {0:'ADV', 1:'Date', 2:'Latitude', 3:'Lo
ngtitude', 4:'Wind', 5:'Pressure', 6:'Status'}, inplace=True)
    return dict
```

In [320]:

```
# obtain all the result list
def getresult(usefullinks,year):
    resultlist = []
    for i in usefullinks:
        a = get_dict(i,year)
        resultlist.append(a)
    return resultlist
```

In [321]:

```
resultlist = getresult(usefullinks,2017)
print('There are %a dictionaries in the resultlist' %len(resultlist))

# print an example
resultlist[7]
```

```
There are 58 dictionaries in the resultlist
```

Out[321]:

```
{'Name': 'Tropical Storm Selma',
 'Category': 'Tropical Storm',
 'Date': [datetime.datetime(2017, 10, 27, 6, 0),
  datetime.datetime(2017, 10, 28, 12, 0)],
 'TableOfPath':   ADV                Date Latitude Longtitude Wind P
ressure  \
 0   1 2017-10-27 06:00:00    10.70     -89.50   35     1005
 1   2 2017-10-27 12:00:00    11.10     -89.50   35     1005
 2   3 2017-10-27 18:00:00    11.70     -89.40   35     1005
 3   4 2017-10-28 00:00:00    12.30     -89.00   35     1005
 4   5 2017-10-28 06:00:00    13.00     -88.80   35     1005
 5   6 2017-10-28 12:00:00    13.70     -88.50   30     1006

                 Status
 0       TROPICAL STORM
 1       TROPICAL STORM
 2       TROPICAL STORM
 3       TROPICAL STORM
 4       TROPICAL STORM
 5  TROPICAL DEPRESSION  }
```

6. We find some missing data in the Wind column of some tables. Since the classification of a hurricane at a given moment can be found in the Status column of the same table and the classification also relates to the wind speed at that moment, use the classification to impute the missing wind data. You may want to read the following website https://en.wikipedia.org/wiki/Tropical_cyclone_scales (https://en.wikipedia.org/wiki/Tropical_cyclone_scales)

In [294]:

```
def imputewind(category):
    # set the median number to impute
    if category == 'TROPICAL DEPRESSION':#winds(knot) belongs to [0, 33]
        return 16.5
    elif category == 'TROPICAL STORM':#winds(knot) belongs to [34, 63]
        return 48.5
    elif category == 'HURRICANE-1':#winds(knot) belongs to [64, 82]
        return 73
    elif category == 'HURRICANE-2':#winds(knot) belongs to [83, 95]
        return 88.5
    elif category == 'HURRICANE-3':#winds(knot) belongs to [96, 113]
        return 104
    elif category == 'HURRICANE-4':#winds(knot) belongs to [114, 135]
        return 124
    elif category == 'HURRICANE-5':#winds(knot) > 135, impute the data from medi
an of [136, 150]
        return 143
    elif category == 'CYCLONE-1':#winds(knot) belongs to [64, 84]
        return 74
    elif category == 'CYCLONE-2':#winds(knot) belongs to [85, 99]
        return 92
    elif category == 'CYCLONE-3':#winds(knot) belongs to [100, 114]
        return 107
    elif category == 'CYCLONE-4':#winds(knot) > 114, impute the data from median
 of [135, 130]
        return 122
```

In [295]:

```
#resultlist[10]['TableOfPath']
def fillmissingwind(resultlist):
    for i in resultlist:
        a = i['TableOfPath'][['Wind','Status']]
        for m in range(len(a['Wind'])):
            if a['Wind'][m] == 0 or a['Wind'][m] == '' or a['Wind'][m] == None:
                a['Wind'][m] = imputewind(a['Status'][m])
fillmissingwind(resultlist)
```

7. Plot the hurricane paths of year '2017' size by the wind speed and color by the classification status.

In [264]:

```
from mpl_toolkits.basemap import Basemap
```

```
In [325]:
```

```
# Merge all the hurricane paths information in 2017 together
def mergeallwithcolor(resultlist):
    totalpathinformtion = [resultlist[i]['TableOfPath'] for i in range(len(resul
tlist))]
    totalpathinformtion = pd.concat(totalpathinformtion)

    # Assign size and color to unique status
    totalpathinformtion['Size'] = totalpathinformtion.groupby(['Wind']).ngroup()
    totalpathinformtion['Color'] = totalpathinformtion.groupby(['Status']).ngrou
p()
    return totalpathinformtion

totalpathinformtion = mergeallwithcolor(resultlist)
totalpathinformtion[:5]
```

```
Out[325]:
```

	ADV	Date	Latitude	Longtitude	Wind	Pressure	Status	Size	Color
0	1	2017-12-27 18:00:00	18.80	121.60	40	0	TROPICAL STORM	18	11
1	2	2017-12-28 00:00:00	19.40	121.70	35	0	TROPICAL STORM	17	11
0	1	2017-12-08 18:00:00	18.30	86.80	40	0	TROPICAL STORM	18	11
1	2	2017-12-09 00:00:00	19.30	86.90	40	0	TROPICAL STORM	18	11
2	3	2017-12-09 06:00:00	18.80	86.80	35	0	TROPICAL STORM	17	11

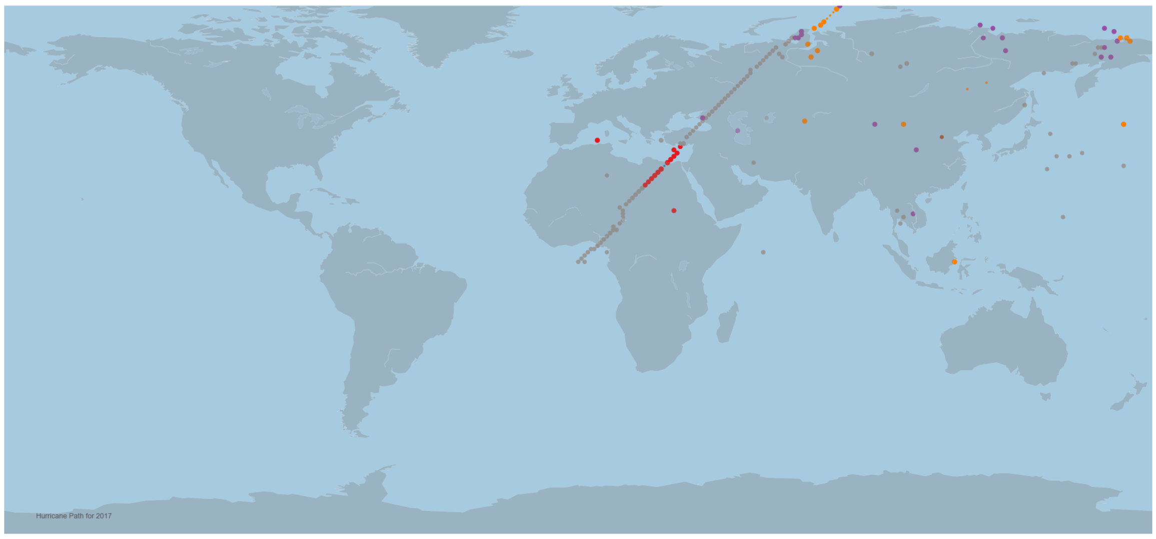
In [299]:

```
# Set the dimension of the figure
def plottheresult(totalpathinformtion):
    my_dpi=100
    plt.figure(figsize=(2600/my_dpi, 1600/my_dpi), dpi=my_dpi)

    # Make the background map
    m=Basemap(llcrnrlon=-180, llcrnrlat=-85,urcrnrlon=180,urcrnrlat=80)
    m.drawmapboundary(fill_color='#A6CAE0', linewidth=0)
    m.fillcontinents(color='grey', alpha=0.3)
    m.drawcoastlines(linewidth=0.1, color="white")

    # Determine size and color
    sizes = [z+3 for z in totalpathinformtion['Size']]
    cols = [w+2 for w in totalpathinformtion['Color']]

    # Add a point per position
    lons = list(totalpathinformtion['Longtitude'])
    lats = list(totalpathinformtion['Latitude'])
    x, y = m(lons, lats)

    m.scatter(x, y, s=sizes, c=cols, cmap="Set1")
    plt.text( -170, -80, 'Hurricane Path for 2017', size=9, color='#555555')
    m
    return m

plottheresult(totalpathinformtion)
```

[redacted]

Out[299]:

```
<mpl_toolkits.basemap.Basemap at 0x1c16e9940>
```



8. (Optional) Convert the above functions as function of year so that when we change year, you will be able to generate plot of the hurricane paths in that year easily.

In [288]:

```python
#  rewrite a function to remove summary links.
def getusefullinks(alllinks):
    usefullinks = []
    for i in range(len(alllinks)):
        if len(alllinks[i].split('/')) == 7:
            usefullinks.append(alllinks[i])
    return usefullinks
```

In [326]:

```
def getyearofpath(year,pageno):
    # 1. get all the links
    hurricane = getlinks(year,pageno)
    # 2. remove summary links
    usefullinks = getusefullinks(hurricane)
    # 3. make the links into a list of dictionaries containing informations(eg.
 path, hurriacane type, name,...)
    resultlist = getresult(usefullinks,year)
    # 4. if the wind value is missing, impute so
    fillmissingwind(resultlist)
    # 5. merge all the information into one dataframe, assign color for each sta
tus and size the point by wind
    totalpathinformtion = mergeallwithcolor(resultlist)
    # 6. plot the data
    return plottheresult(totalpathinformtion)
```

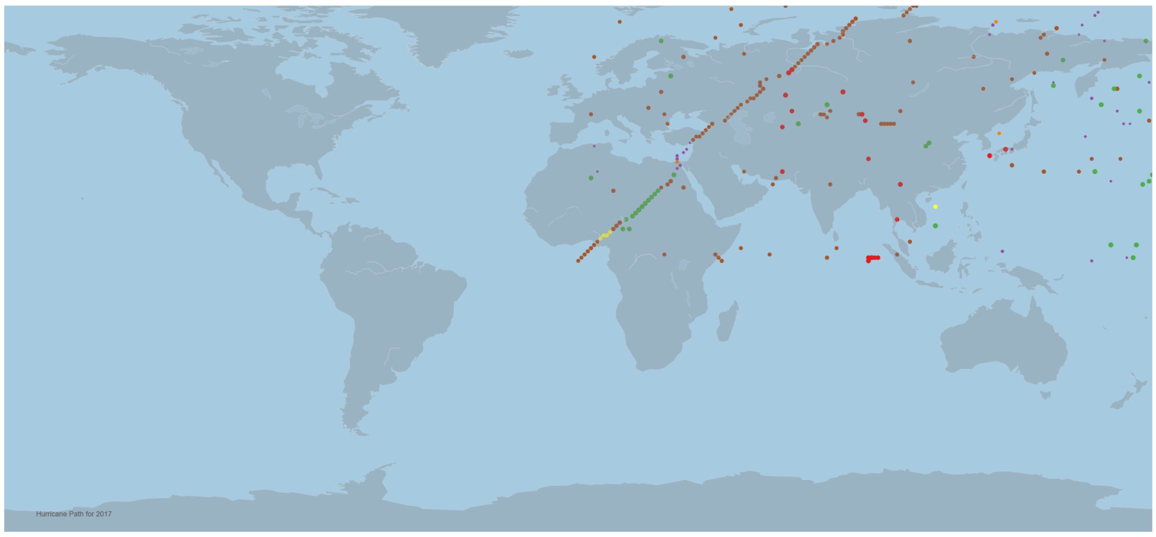
In [327]:

```
getyearofpath(2018, 7)
```

[redacted]

Out[327]:

```
<mpl_toolkits.basemap.Basemap at 0x1d2be82e8>
```


Hurricane Path for 2017