# Ladyball - the Story of a Hashtag

Anthony Munnelly

@anspailpin (https://twitter.com/anspailpin)

https://ie.linkedin.com/in/anthonymunnelly (https://ie.linkedin.com/in/anthonymunnelly)

# Ladyball - Where It Begins

On January 13th of this year of grace, 2016, at about lunchtime, the surfing classes of Erin got a shock. This shock:

Nobody knew what to make of it, except that nobody seemed much to like it. Nobody knew what this Ladyball was for. People were almost certain that it was a joke, but for a joke it didn't seem all that funny.

About a week later it transpired that Ladyball was a guerilla marketing campaign to announce the Lidl supermarket chain's sponsorship of Ladies' Gaelic Football. But the questions I want to figure out tonight are:

1. Was the campaign a success?
2. Can a search of the `#ladyball` hashtag on Twitter answer this question?

## Tweet Metadata

The Twitter api (https://dev.twitter.com/rest/reference/get/statuses/user_timeline) shows a sample tweet, broken down as a json object. Tweets are famous for being short and (sometimes) sweet. But the amount of information that's packed into one tweet is staggering, and a little bit frightening.

You can see a sample if you click the link above to the Twitter Developers but it's a long list of javascript objects, many of which contain javascript objects themselves, just like those Russian dolls.

## Tweepy

*Tweepy* is now the standard module for interacting with Twitter in Python. Each Tweepy search returns an iterable Tweepy result set object. Each iteration is a Tweepy object, that compares to the `.json` we saw just now like so:

| **.json** | **Tweepy** |
|---|---|
| `result[0]['created_at']` | result[0].created_at |
| `result[0]['user']['screen_name']` | result[0].user.screen_name |

## Collecting the Tweets

In the current case, tweets were collected by running a program that had two functions. The first function created the api that allowed us to interact with Twitter in the first place, and the second then collected and pickled the tweets.

## Creating the api

You have to register an app with Twitter in order to use the api. Once created, your app supplies you with four passwords - consumer_key, consumer_secret, access_token, and access_token_secret. It is, of course, extremely bad practice to write these into any program, so I prefer to store them as .json and load them at the start. This is the straight-forward code to do so. In previous iterations of tweepy, it was necessary to set a rate-limit-checker yourself - this is now part of the package. Progress is a wonderful thing.

```
def create_the_api(my_login):

    # Log in to Twitter
    with open(my_login, 'r') as f:
        keys = json.load(f)

    auth = tweepy.OAuthHandler(keys['consumer_key'],
                               keys['consumer_secret'])
    auth.set_access_token(keys['access_token'],
                          keys['access_token_secret'])

    return tweepy.API(auth, wait_on_rate_limit=True,
                      wait_on_rate_limit_notify=True)

api = create_the_api(MY_URL)
```

## Twitter Rate Limits

Twitter limits api access (https://dev.twitter.com/rest/public/rate-limiting). Depending on what you're looking for, you can get fifteen or 180 requests every fifteen minutes. No more. So once we hit that limit, we have no option but to wait out the fifteen minutes until we're ready to go again, and we have to allow for that in the code.

## Creating the tweet-finder function

Having created the api function, this is the function that finds the tweets, with a helper function that I'm about to explain.

```
def max_id_finder(temp):
    # A helper function to stop us collecting
    # the same data over and over again
    #(tweepy.Result) -> (int)

    ids = [int(tweet.id) for tweet in temp]
    ids.sort()

    return ids[0]
```

```
def hashtag_searcher(hashtag, count=180, api=api):
    if hashtag[0] == '#': # Allow for whether or not the
                          # user has included # as part of
                          # her hashtag
        hashtag = '%23' + hashtag[1:]
    else:
        hashtag = '%23' + hashtag

    tweets = []
    temp = []
    for i in range(count):
        print i
        if temp != []: # Checks to see if loop has run already,
                       # and thus needs the max_id parameter
            temp = api.search(hashtag,
                              max_id = max_id_finder(temp))
        else:
            temp = api.search(hashtag)

        [tweets.append(t) for t in temp]

    timestamp = datetime.datetime.today()
     # Get rid of the %23 to create a valid filename
    filename = hashtag[3:] + timestamp.strftime('%Y%m%d')

    with open(filename, 'w') as f:
        pickle.dump(tweets, f)

    return tweets
```

So this `hashtag_searcher()` function creates two lists, tweets and temp. When the api.search method is called, the results are returned to temp up to the api_rate_limit. Each of the results are then appended to the main, tweets, list, and the once the 15 minute interval has clicked away, the search renews. However, instead of starting over again, it starts at the max_id of the last returned value - it goes back to where it left off, in other words.

Once the full count has run, the tweets list is both is pickled and returned - a belt and braces undertaking, in keeping with the long time these searches can take.

```
if __name__ == '__main__':
    api = create_the_api("my_twitter_login.json")
    tweets = hashtag_searcher('ladyball', 1000, api)
```

In [2]:
```
import tweepy
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import glob
```

I've pickled these `#ladyball` tweets since January 14th. So, what I'm going to do next is

1. Load up the pickles from where they're stored into a single list, `tweets_prime`.
2. Prune that list to remove the duplicates. Every tweet has its own unique id, identified as `id` or `id_str`. We'll use that to catch the spares.

In [3]:
```
ladyballs = glob.glob("/Users/anthonymunnelly/Documents/TECH/Python/Tutorials/ladyball/
ladyball---the-story-of-a-hashtag/pickled_ladyballs/*")
```

In [4]:
```
tweets_prime = []
for ball in ladyballs:
    with open(ball) as f:
        temp = pickle.load(f)
        [tweets_prime.append(t) for t in temp]
```

In [5]:
```
print type(tweets_prime[0])
len(tweets_prime)
```

```
<class 'tweepy.models.Status'>
```

Out[5]:
```
9686
```

In [6]:
```
id_watcher = []
ladyballs = []
for t in tweets_prime:
    if t.id_str not in id_watcher:
        ladyballs.append(t)
        id_watcher.append(t.id_str)

print len(ladyballs)
```

```
3774
```

## Looking at the Tweet Dates

The easiest way to look at the tweet dates is to create a data frame and use that to pull from the `ladyballs` list such data as a data frame is unsuited to storing, such as text. My method of choice for creating dataframes is to create a dictionary and then turn that dictionary into a dataframe. And for creating dictionary, there's nothing as handy as the defaultdict in the collections module. It's so neat and tidy.

In [7]:
```
from collections import defaultdict
my_initial_dict = defaultdict(list)

my_initial_dict['created_at'] = [l.created_at for l in ladyballs]
my_initial_dict['screen_name'] = [l.user.screen_name for l in ladyballs]
my_initial_dict['id_str'] = [l.id_str for l in ladyballs]
my_initial_dict['followers_count'] = [l.user.followers_count for l in ladyballs]
my_initial_dict['text'] = [l.text for l in ladyballs]
my_initial_dict['description'] = [l.user.description for l in ladyballs]
my_initial_dict['retweet_count'] = [l.retweet_count for l in ladyballs]

df_ladyballs = pd.DataFrame(my_initial_dict, columns = ['created_at',
                                                        'screen_name',
                                                        'id_str',
                                                        'followers_count',
                                                        'text'
                                                       'description',
                                                       'retweet_count'])

# Life is much handier if we sort the list by date.
df_ladyballs.sort_values('created_at', inplace=True, ascending=True)
```

## Looking at the Life Cycle

You can look at the first twenty or a hundred tweets on `Github` but take it from me, they were fairly negative. So let's look at the life cycle of the `ladyball` hashtag to see when people were talking about it, and we can do this by grouping by date - kind of. To group by date, we have to

1. Create a new column on the dataframe, with a string representation of the date
2. Group by this column. This is how that works out in this case:

In [8]:
```
day_month = [tweet.strftime('%d %b') for tweet in df_ladyballs['created_at']]
df_ladyballs['day'] = day_month
ladyballs_by_day = df_ladyballs.groupby('day')
for a, b in ladyballs_by_day:
    print a, b.created_at.count()
```

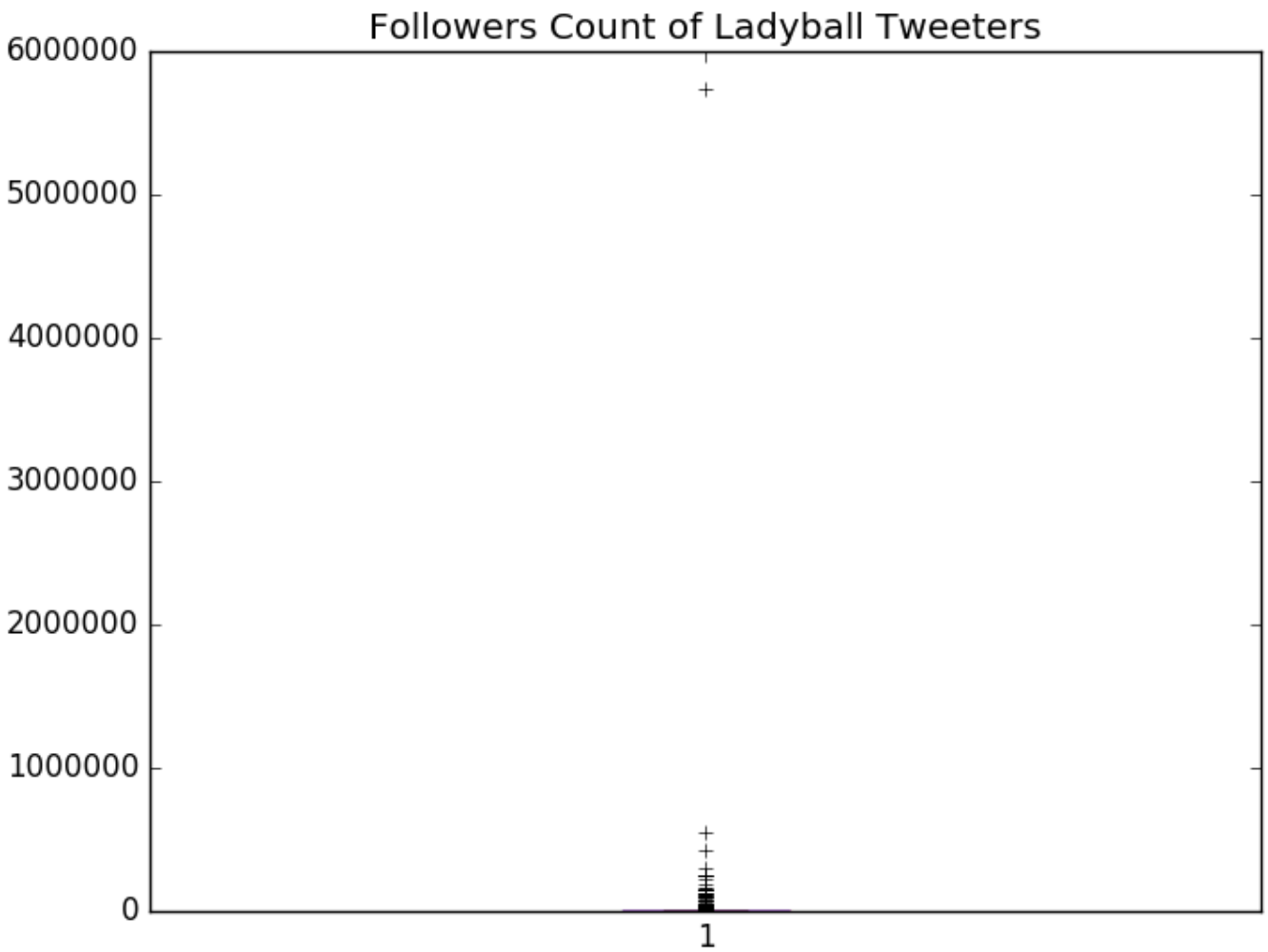
```
09 Jan 1
11 Jan 1
13 Jan 901
14 Jan 966
15 Jan 1208
16 Jan 381
17 Jan 75
18 Jan 68
19 Jan 90
20 Jan 26
21 Jan 36
22 Jan 18
23 Jan 3
```

## Looking at the Exposure

It's not true to say that a tweet is only seen by someone's followers. A tweet can be quoted or retweeted, it can found in a search, it can be found when a hashtag is trending. All these things. But as there's no point in throwing up our hands and saying ah, who knows?, the convention has become to look at a tweet's reach, or exposure, as being dependent on that particular tweeter's followers. So, what is the follower distribution like among the `#ladyball` tweeters?

In [9]:
```
%matplotlib qt
plt.boxplot(df_ladyballs.followers_count)
plt.title('Followers Count of Ladyball Tweeters')
```

Out[9]:
```
<matplotlib.text.Text at 0x12b6792d0>
```



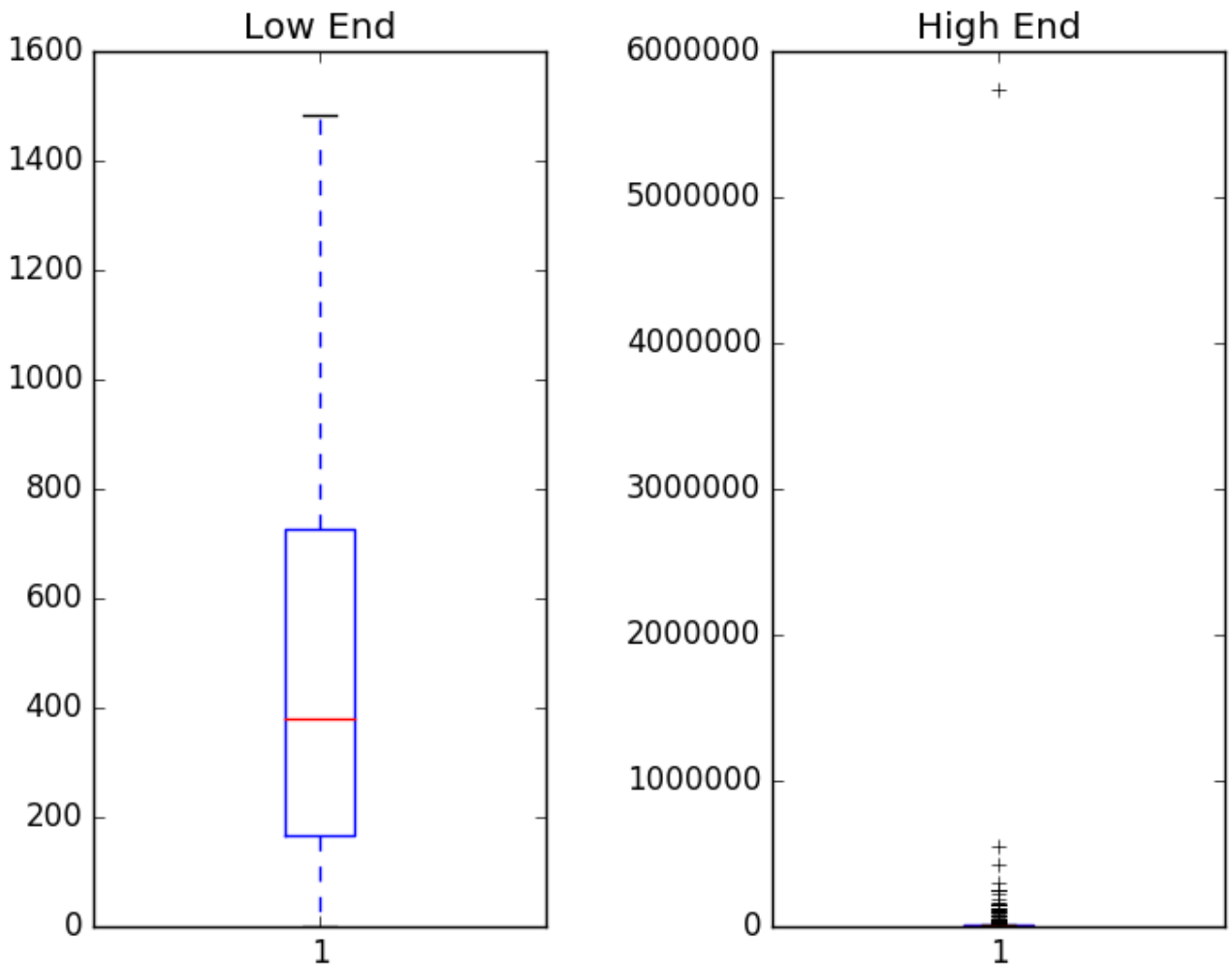

That's one crazy-looking boxplot. Let's see the actual figures.

In [10]:
```
print df_ladyballs.followers_count.describe()
```

```
count       3774.000000
mean        5543.403286
std        95562.944061
min            0.000000
25%          236.000000
50%          578.500000
75%         1486.250000
max      5738471.000000
Name: followers_count, dtype: float64
```

Isn't that interesting? 75% of our tweets, 2,530 of them, have follower counts of 1,486 or lower. The remaining 994 have over five and a half-million between them. Let's see who they are, by slicing the data frame into a `high_end`, those accounts with more than 1,486 followers, and `low_end`, the remaining majority of accounts.

In [11]:
```
%matplotlib qt
high_end = df_ladyballs[df_ladyballs['followers_count'] > 1486]
low_end = df_ladyballs[df_ladyballs['followers_count'] < 1486]
fig, ax = plt.subplots(1,2)
ax[0].boxplot(low_end.followers_count.values)
ax[0].set_title('Low End')
ax[1].boxplot(high_end.followers_count.values)
ax[1].set_title('High End')
fig.subplots_adjust(wspace = 0.5)
```


Low End
1600
1400
1200
1000
800
600
400
200
0
1
High End
6000000
5000000
4000000
3000000
2000000
1000000
0
1

### Plotting Aside

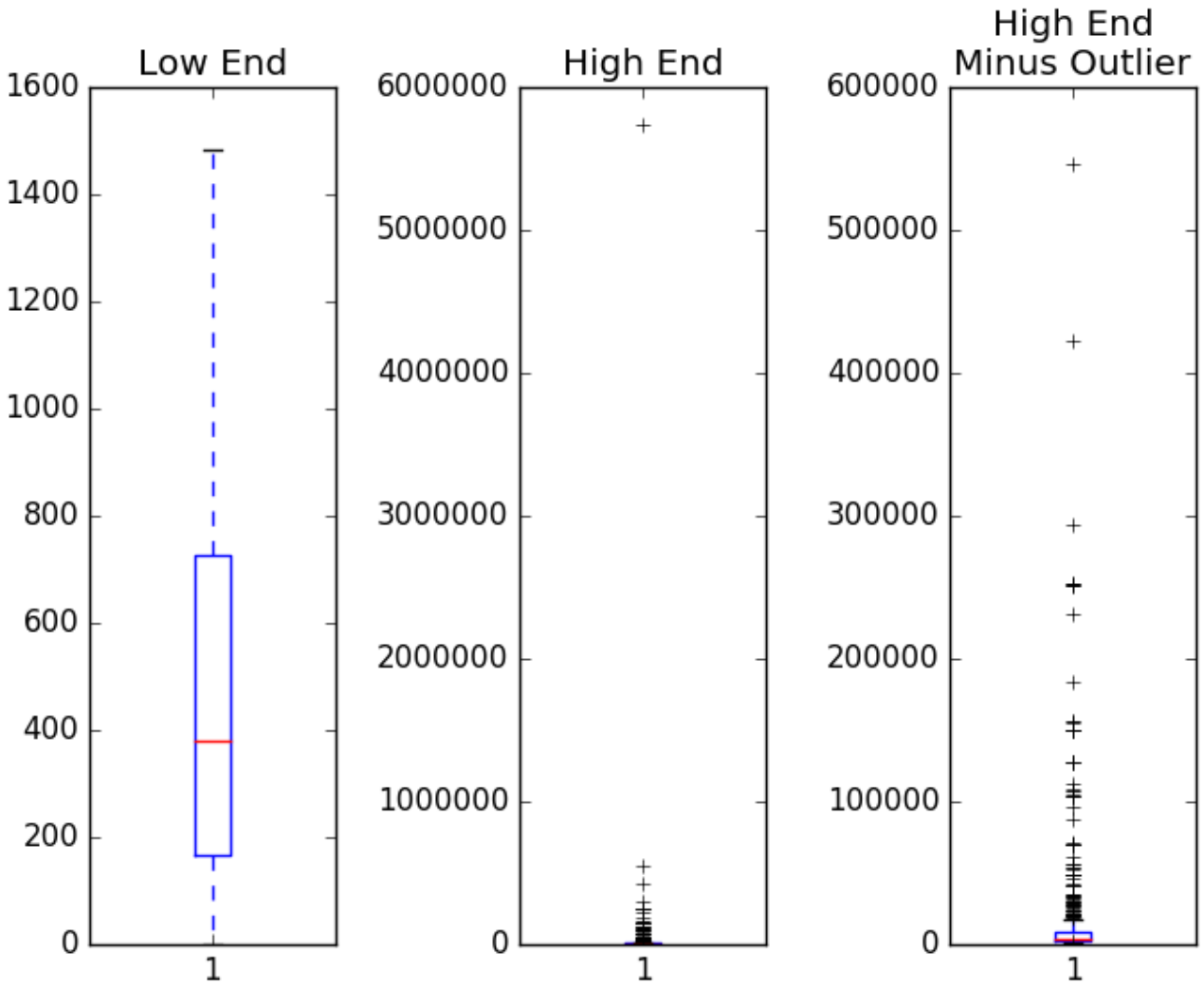
We shuld note that we're plotting `high_end.followers_count.values` rather than `high_end.followers_count`. This is because we've taken a slice of the original `df_ladyballs` data frame, this throws off the indexing, and the thrown indexing goes on to throw `matplotlib`. There may be a more elegant way around this problem than calling `values`, but calling `values` works and that's enough for me.

## Dealing with the Outliers

There are still an enormous amount of outliers in the high end. Obviously, the 5.5 million is the biggest beast of all. Let's drop that and see what happens.

In [12]:
```
high_end_minus_outlier = high_end[high_end['followers_count'] < 5738471]

%matplotlib qt
fig, ax = plt.subplots(1,3)
ax[0].boxplot(low_end.followers_count.values)
ax[0].set_title('Low End')
ax[1].boxplot(high_end.followers_count.values)
ax[1].set_title('High End')
ax[2].boxplot(high_end_minus_outlier.followers_count.values)
ax[2].set_title('High End\nMinus Outlier')
fig.subplots_adjust(wspace = 0.75)
```




And again we're looking at distribution that skews very strongly to the right. We still can't even make out the box in the boxplot, such the outlier dominance in the dataset. Let's look at the top tweets by `follower_count`.

In [13]:
```
high_end_examined = high_end.sort_values('followers_count', ascending = False)
high_end_examined.head(25)
```

Out[13]:

| | created_at | screen_name | id_str | followers_count | textdescr |
|---|---|---|---|---|---|
| **3579** | 2016-01-19 21:39:25 | washingtonpost | 689562834330611712 | 5738471 | NaN |
| **2855** | 2016-01-15 18:00:29 | Vibra1049 | 688058186528522240 | 545959 | NaN |
| **1762** | 2016-01-16 22:29:05 | manuel_c | 688488168140713984 | 422373 | NaN |
| **1171** | 2016-01-13 21:48:22 | SimonHoneydew | 687390760174489600 | 293945 | NaN |
| **3003** | 2016-01-15 17:39:29 | spin1038 | 688052899390341120 | 252379 | NaN |
| **2847** | 2016-01-15 18:01:09 | spin1038 | 688058353411616771 | 252379 | NaN |
| **2388** | 2016-01-15 20:13:03 | spin1038 | 688091547414335490 | 252379 | NaN |
| **1471** | 2016-01-13 18:41:23 | Independent_ie | 687343701123792897 | 250647 | NaN |
| **542** | 2016-01-14 | | | | |

And there are some huge brands in there. Newtalk, the Indo, TheDrum, and biggest of all, The Washington Post, with five and a half-million followers. We need to go back to the original data to see the text descriptions of the accounts and the tweets in the top ten tweets here. We can use `id_str` to do that:

In [14]:
```
my_ids = list(high_end_examined.id_str)[:10]
description_catcher = [] #So we don't print the descriptions more than once.
for tweet in ladyballs:
    if tweet.id_str in my_ids:
        print tweet.user.screen_name
        print "Followers: {:,}".format(tweet.user.followers_count)
        if tweet.user.description not in description_catcher:
            print tweet.user.description
            description_catcher.append(tweet.user.description)
        print tweet.created_at
        print tweet.text
        print '\n'
```

## Chatter v Reach

Now we're coming to the heart of Twitter. The fundamental question you have to answer, as you're trying to get your message out on Twitter, is this: Does it matter who's talking about you? Are many tweets by tweeters with small amounts of followers as good as a single tweet by a user with a huge amount of followers?

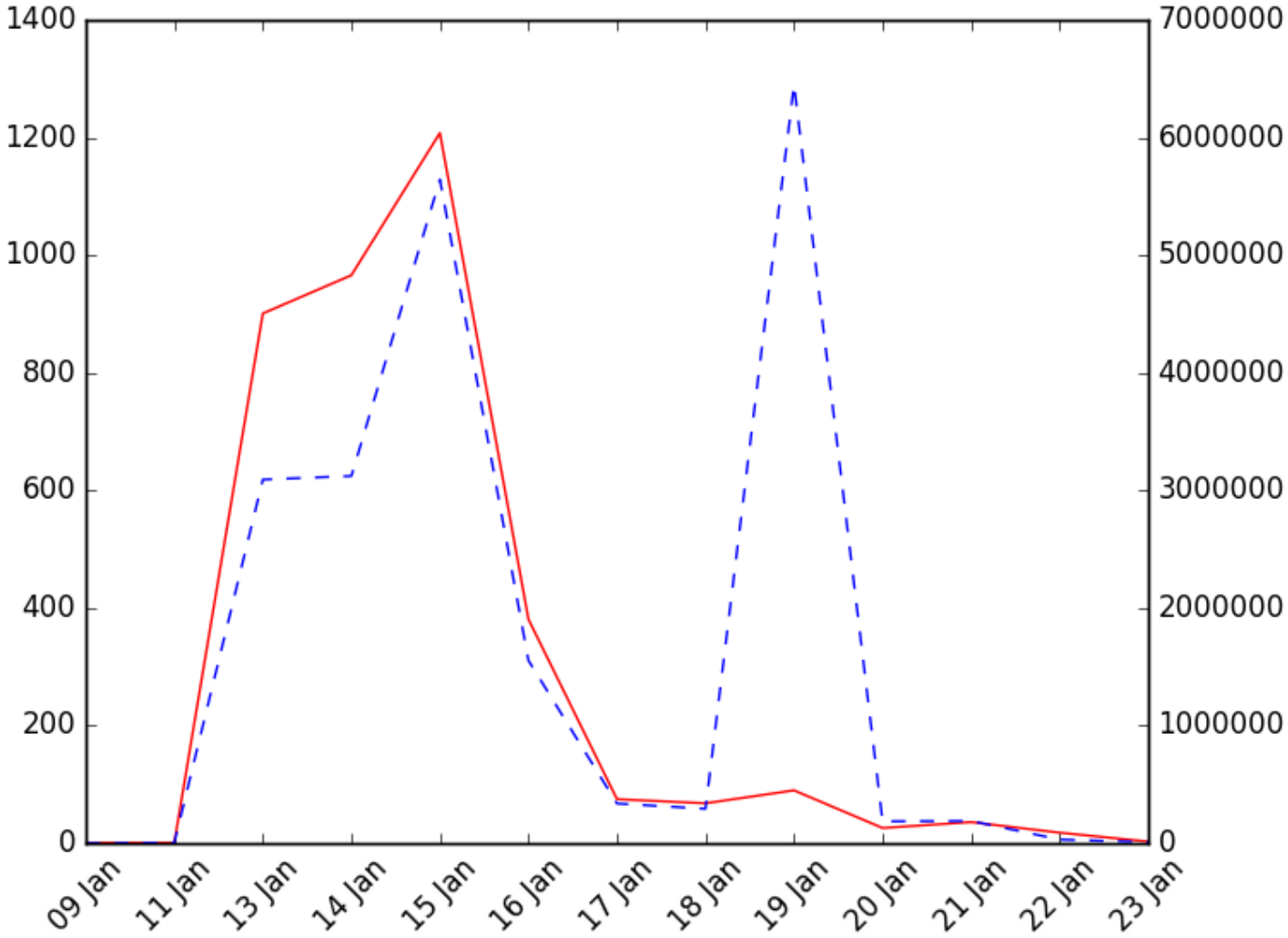
## Time v Reach

This is our final graph of the evening. We're plotting the reach of the `#ladyball` hashtag against the number of times it was mentioned. We'll plot on different axes as it makes for a better visual comparison.

In [15]:
```
tweet_count = []
tweet_reach = []
for a, b in ladyballs_by_day:
    tweet_count.append(b.created_at.count())
    tweet_reach.append(b.followers_count.sum())

days = []
[days.append(a) for a, b in ladyballs_by_day]
%matplotlib qt
fig, ax1 = plt.subplots()
ax1.plot(tweet_count, "r-", label = 'count')
ax2 = ax1.twinx()
ax2.plot(tweet_reach, 'b--', label = 'reach')
ax1.set_xticks(range(len(days)))
ax1.set_xticklabels(days, rotation = 45)
# http://matplotlib.org/examples/api/two_scales.html
```

Out[15]:
```
[<matplotlib.text.Text at 0x1270061d0>,
 <matplotlib.text.Text at 0x12903e810>,
 <matplotlib.text.Text at 0x135f0a6d0>,
 <matplotlib.text.Text at 0x135f0ae10>,
 <matplotlib.text.Text at 0x135f0e590>,
 <matplotlib.text.Text at 0x135f0ecd0>,
 <matplotlib.text.Text at 0x135f11450>,
 <matplotlib.text.Text at 0x135f11b90>,
 <matplotlib.text.Text at 0x135edf1d0>,
 <matplotlib.text.Text at 0x135eda310>,
 <matplotlib.text.Text at 0x12b600350>,
 <matplotlib.text.Text at 0x1295e6750>,
 <matplotlib.text.Text at 0x129d3ff90>]
```




So here we can see that reach is reasonably proportional to chatter / number of tweets during the peak of the `#ladyball` hashtag's life cycle, but the hashtag reaches its highest reach just as it's dying off and people aren't interested anymore. These are the numbers:

In [16]:
```
counter = len(tweet_count)
print "Date\tCount\tReach"
for c in range(counter):
    print "{}\t{:>5}\t{:>10,}".format(days[c], tweet_count[c], tweet_reach[c])
```

```
Date    Count    Reach
09 Jan      1           48
11 Jan      1        5,312
13 Jan    901    3,092,344
14 Jan    966    3,122,821
15 Jan   1208    5,644,684
16 Jan    381    1,554,422
17 Jan     75      338,193
18 Jan     68      293,447
19 Jan     90    6,451,310
20 Jan     26      186,680
21 Jan     36      189,709
22 Jan     18       31,835
23 Jan      3        9,999
```

## So. Was it Worth It?

Was the `#ladyball` tweet worth the controversy it caused?

### Yes

It got people talking, which is the point of any publicity and/or advertising campaign.

### Yes

The initial negative feedback was hugely overweighed by the positive mentions, especially the one in the Washington Post (http://www.washingtonpost).

### No

This is a little meta but: `#ladyball` was a success as a teaser to the campaign. But once the true point of `#ladyball` was revealed, what was left for people to be upset about, or happy about, or engaged with? There was no way to build from `#ladyball` on to the avowed purpose of it all, Lidl's sponsorship of ladies' football. The Washington Post piece was very worthy and is sure to be front-and-centre when the advertising agency put their bill together, but in what way did the `#ladyball`campaign get more people interested in ladies' Gaelic football or get more people shopping at Lidl? That case isn't proven.