This gave me all the column names from the first PLUS call I set up to get phone numbers

- I kept this here so that I could reference this later and tweak my query below to only return the columns I needed - this will make sense WAY down the line

In [11]:
```
print list(phone.columns.values)
```

```
[u'address1', u'address2', u'city', u'company', u'contact', u'employee_count', u'pho
ne', u'phone_type', u'state', u'url', u'zip']
```

In [2]:
```
%reload_ext ishbook
import pandas as pd
import numpy as np
import datetime as dt
import time
import iql
import plus
import re
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important;
 }</style>"))
```

Pull in all advertisers who clicked on "needHelp" from June 2016 to September 2016

- References the *dradisturnstileclick* index
- Grab the *advertiser* info, what *page* they were visiting at the time, and the *unixtime* to capture the time they requested help

In [3]:
```
%%ish

needhelp_advids = from dradisturnstileclick 2016-06-01 20
16-09-01 WHERE section = crawl element = needHelpButton a
dvid > 0 GROUP BY advid, element, page, unixtime
elaborate(needhelp_advids.advid)
output = None
```

Just checking to see how many rows are in the ouput and what the df looks like

- Because I called **elaborate** on the advids in my initial pull, I can see that some of them are labeled as *blank*
- What the blanks tell me are the advertisers that started setting up a page and never actually completed their set-up: they don't have any data in their adCard

Filter out those advids that are labeled *blank* by making a copy of the original df

- Basically creating a condition saying anywhere where **_advid** is not blank from the *needhelp* df, place that in the *no_blanks* df as a copy of the original df

In [4]:
```
no_blanks = needhelp_advids[needhelp_advids["_advid"] != "BLANK"].copy()
```

Checking the lengths of each df to make sure no data was lost

In [5]:
```
print len(needhelp_advids)
print len(no_blanks)
print len(needhelp_advids) - len(no_blanks)
```

```
203808
183029
20779
```

Create Date and Time columns based on the unixtime stamp of when the advertiser clicked on needHelp

- This is going to allow it to be a little more readable further along in the notebook
- I'm also going to use this in a few of my groupBys, to be able to tie when an advertiser was stuck to when they contacted us

```
Syntax is:
    df[column_name] = [dt.datetime.fromtimestamp(x).strft
ime('%Y-%-m-%d') for x in df[unxtime]
```

Essentially, what this is saying is 'Create two columns, Date and Time, on the existing dataframe'

For every row in no_blanks[unixtime], create a string representation of a datetime object and populate it in the appropriate column

```
    - '%Y-%m-%d' formats the date as *YEAR*-*MONTH*-*DAY*
 (2016-09-10)
    - '%H:%M:%S' formats the time as *HOURS(24H)*:*MINUTE
S*:*SECONDS* (20:15:66)
```

In [6]:
```
no_blanks["Need Help Date"] =
[dt.datetime.fromtimestamp(x).strftime('%Y-%m-%d') for x
in no_blanks.unixtime]
no_blanks["Need Help Time"] =
[dt.datetime.fromtimestamp(x).strftime('%H:%M:%S') for x
in no_blanks.unixtime]
```

I also had to get the *billing_country* and *locale* for each advertiser, as I needed to split up the populations by USA only and Rest of the World

- Due to the limitation of feeding more than ~100,000 advids into a PLUS call, I had to create an ARRAY which I labeled *advids* - again, my variable names aren't that great, *try to be more specific!!!*

- Then I created two dfs, feeding the list of advids by splitting them in half using this syntax: series[0:number]
  - What this does is tells Python give me a SLICE of that series, from 0 to 60000 (essentially give me the first 60000 elements in my ARRAY)
  - For the remaining advids, feed it into the second plus call

- The other thing you might notice is I'm only returning the two columns that I need from that PLUS call... Unfortunately PLUS will still pull ALL of the columns for each of those advertisers but the final df will just be those two columns

In [9]:
```
advids = no_blanks.advid.unique()
```

In [11]:
```
no_blanks_list_1 = plus.get_advertiser_info(advids[0:6000
0])[['billing_country', 'locale']]
no_blanks_list_2 =
plus.get_advertiser_info(advids[60000:])[['billing_countr
y', 'locale']]
```

```
LDAP name:
dmoncada
Password:
........
```

Combine the billing_country information with my original population to be able to split them further down in the notebook

- Combined the two dfs from the above PLUS call together by using pd.concat and creating a new df labeled *all_countries*
- Merge the *all_countries* df with my original *no_blanks* df

```
how = 'left'
```

I want to merge all of the records **LEFT** ONTO the original df, since they should be a 1-to-1 match and I should have a row for each one

```
left_on = 'advid'
```

The original df (the LEFT one) has 'advid' as the unique identifier - that's how I will tie to the RIGHT df

```
right_index = True
```

The RIGHT df (*all_countries*), I'll be using the INDEX as my unique identifier, since what is returned from PLUS has the advertiser ID as the index

In [12]:
```
all_countries = pd.concat([no_blanks_list_1, no_blanks_li
st_2])
```

In [13]:
```
no_blanks_countries = no_blanks.merge(all_countries, how
= 'left', left_on = 'advid', right_index = True)
```

Create a string list of UNIQUE advids to feed into the IQL queries that I am using to obtain the follow-ups submitted by the advertisers that got stuck

- Going to be feeding these advids into two different IQL queries
- I create one df *contactwebform*, feeding the advids from the orignal no_blanks df, for the same date range, to get all CWFs submitted during that time by those same advertisers
  - Feeding the specific adveritsers helps me return a much more specific result set; otherwise I'd get ALL CWFs submitted for that time frame
- I then create another df *zen_desk*, using a similar process
  - I rename the columns that are returned to match those in the *contactwebform* df, so that when I concat the two dfs later it saves me the step of renaming them later

```
Syntax is:
    GROUP BY [column_in_iql]/*new_column_name*/
```

- I tend to put `Output = None` after my IQL queries just so that the results don't auto populate and I can check the output after the fact

In [16]:
```
no_blank_list = ','.join(str(x) for x in pd.unique(no_bla
nks['advid']))
```

I use the print statements to see the length of each df as well as what the columns that are being returned look like - to make sure I can concatenate them

```
Syntax is:
    print 'This string/sentence + {}'.format(the length o
f the dataframe)
```

- The *.format* function converts the length of the df (which is an int/number) into a string and places it where the squiggly brackets{} are in my statement
- I also print the first five lines of the dfs to make sure the column names match up

In [22]:
```
print 'The length of zen_desk is
{}'.format(len(zen_desk))
print
print zen_desk.head(5)
print
print 'The length of contactwebform is {}'.format(len(con
tactwebform))
print
print contactwebform.head()
```

```
The length of zen_desk is 35701

   advertiser_id  activity_type    unixtime
0        6908211  zendeskButton  1464776005  1
1        7373056  zendeskButton  1464776920  1
2        7068691  zendeskButton  1464777545  1
3        5158947  zendeskButton  1464778120  1
4        3105282  zendeskButton  1464778880  1

The length of contactwebform is 27767

   advertiser_id     activity_type    unixtime
0        6051871  CONTACT_WEB_FORM  1464763613  1
1        5569753  CONTACT_WEB_FORM  1464764103  1
2        6412012  CONTACT_WEB_FORM  1464767466  1
3        7242318  CONTACT_WEB_FORM  1464767662  1
4        7275440  CONTACT_WEB_FORM  1464767743  1
```

Combine the two dfs from the IQL queries together, create Date & Time stamps similar to what was done previously, and combine it with the no_blanks_countries df

In [23]:
```
all_contact_types = pd.concat([contactwebform, zen_desk])
```

In [24]:
```
all_contact_types["Contact Date"] = [dt.datetime.fromtime
stamp(x).strftime('%Y-%m-%d') for x in
all_contact_types.unixtime]
all_contact_types["Contact Time"] = [dt.datetime.fromtime
stamp(x).strftime('%H:%M:%S') for x in
all_contact_types.unixtime]
```

In [25]:
```
no_blanks_countries.rename(columns = {'advid' : 'advertis
er_id'}, inplace = True)
```

THE BIG KAHUNA

- I merge the original dataset (all of the advertisers who clicked on 'needHelp' that have their billing country associated with them) with those advertisers in that list that submitted a CWF or a ZenDesk escalation
- In order to do so, I used an `inner` join, as this is going to match up the two data sets on the CLOSEST time between *NEED HELP DATE* and *CONTACT DATE*
- Essentially the merge says "Find any instance where the same date an advertiser ID clicked on "NEED HELP" they ALSO had a CWF or zenDesk submitted
  - Then, following that, find the closest time between the two
  - This does return some instances where they submitted a CWF or zenDesk BEFORE they clicked "NEED HELP" - we'll be removing these

In [28]:
```
successful_contact = no_blanks_countries.merge(all_contac
t_types, how = 'inner', left_on = ['Need Help Date', 'adv
ertiser_id'], right_on = ['Contact Date',
'advertiser_id'])
```

I need to split the data set between US only and Rest of the World

- I create a `condition` - which is how you can filter results on a df

```
Syntax is:
    condition_name = df[column_you_filter_on] == 'the var
iable you are pulling out'
```

In [29]:
```
us_only = successful_contact['billing_country'] == 'US'
```

I then feed the condition back into the original dataframe - using that to create two separate dataframes

```
Syntax is:
    new_df_name = original_df[condition]
```

This says create a new df with ONLY results from my original df that meet this condition I made above - in the case below, ONLY advertisers that have a billing country that is 'US'

```
Syntax is:
    other_df_name = orginal_df[~condition]
```

The squiggly "~" (yep, that's the technical term) says create a separate df with any results from the original df that DON'T meet the condition - again, in the case below, only advertisers that have a billing country that IS NOT EQUAL to 'US'

In [30]:
```
successful_contact_us = successful_contact[us_only]
successful_contact_world = successful_contact[~us_only]
```

In [31]:
```
print len(successful_contact)
print len(successful_contact_us)
print len(successful_contact_world)
```

```
92295
52374
39921
```

I am creating more conditions based on the requirements of the data pull

- They want to see how many advertisers were able to successfully contact us during business hours and outside of business hours
- This was kind of a nightmare to figure out because when I first set up this notebook, I didn't take into account the 12 hour vs 24 hour clock, so creating a condition took me quite a few tries
- The two conditions are based on being between `8AM - 08:00:00` and `8PM - 20:00:00`
- Again, I created two different dfs based on these conditions - but THIS time I need to use slightly different syntax to get the result set I want

```
Syntax is:
    new_df = original_df[condition1 | condition2]
    new_df = orignal_df[~condition1 & ~condition2]
```

- For this example, what this says is create one df where it is EITHER before 8AM OR AFTER 8PM - this gets me all of the successful contacts outside of business hours
- Then create ANOTHER df where it is NOT after 8PM AND NOT before 8 AM - this gets me all of the successful contacts BETWEEN business hours

In [32]:
```
before_eight_am = successful_contact_us['Need Help Time']
 < '08:00:00'
after_eight_pm = successful_contact_us['Need Help Time']
> '20:00:00'
```

In [33]:
```
outside_business_hours_us = successful_contact_us[before_
eight_am | after_eight_pm]
inside_business_hours_us = successful_contact_us[~after_e
ight_pm & ~before_eight_am]
```

In [34]:
```
print len(successful_contact_us)
print len(outside_business_hours_us) + len(inside_busines
s_hours_us)
```

```
52374
52374
```

I have to do one more step here: I drop duplicates because we might have instances where an advertiser clicked on needHELP more than once

- This skews the data as the inner merge (done above) will match up EACH submitted CWF & zenDesk escalation with the when the advertiser clicked on "needHELP" - so I just get one of each instance by using drop_duplicates

```
Syntax is:
    new_df = original_df.drop_duplicates([columns that yo
u are de-dupping], keep = 'last')
```

- The critical piece is 'keep = last' - this tells Python to keep the last record after de-dupping

In [38]:
```
us_final_out =
outside_business_hours_us.drop_duplicates(['advertiser_i
d', 'Contact Date', 'Contact Time'], keep = 'last')
us_final_in =
inside_business_hours_us.drop_duplicates(['advertiser_id',
 'Contact Date', 'Contact Time'], keep = 'last')
```

Create two more conditions (man I really became obsessed with these)

- I need to only get those CWFs/zenDesks that were submitted within 1hr of clicking 'NeedHelp'
- I also need to grab ONLY those escalation that were submitted AFTER clicking 'needHelp'

```
Snytax is:
    condition = df[column1] - df[column2] < int
    condition2 = df[column1] - df[column2] > int
```

- In this example, I create the *within_1hr* condition by subtracting the unixtime of the submission from the unixtime of when they clicked on needHelp, which gives me a timedelta series in SECONDS, since that is how unixtime is calculated
  - I am grabbing anything LESS than 360 since 60secs (1min) * 60mins = 360
- I then create the *no_negative* condition by doing the same calculation and only grabbing anything GREATER than 0, since that is positive
  - A positive number means that they submitted the escalation AFTER the needHelp click

In [39]:
```
within_1hr = us_final_out['unixtime_y'] - us_final_out['u
nixtime_x'] < 360
no_negative = us_final_out['unixtime_y'] -
us_final_out['unixtime_x'] > 0
within_1hr_in = us_final_in['unixtime_y'] -
us_final_in['unixtime_x'] < 360
no_negative_in = us_final_in['unixtime_y'] -
us_final_in['unixtime_x'] > 0
```

Filter the results and print out the data points I need

In [55]:
```
print 'This is how many successful escalations we receive
d OUTSIDE business hours in the US\n'
print us_final_in[within_1hr_in & no_negative_in]['activi
ty_type'].value_counts()
print
print 'This is how many successful escalations we receive
d INSIDE business hours in the US\n'
print us_final_out[within_1hr & no_negative]['activity_ty
pe'].value_counts()
```

```
This is how many successful escalations we received OUTSIDE business hours in the US

zendeskButton       13808
CONTACT_WEB_FORM     7905
Name: activity_type, dtype: int64

This is how many successful escalations we received INSIDE business hours in the US

zendeskButton       2764
CONTACT_WEB_FORM    1845
Name: activity_type, dtype: int64
```

Concat all the results together, then do a value_count on the 'page' to see what page the advertiser was visiting when they asked for assistance

In [46]:
```
us_final = pd.concat([us_final_out, us_final_in])
```

In [61]:
```
print 'This is the top 5 pages that the advertisers were
 visiting at the time they clicked needHelp in the US'

us_final['page'].value_counts().head()
```

```
This is the top 5 pages that the advertisers were visiting at the time they clicked
 needHelp in the US
```

Out[61]:
```
dashboard          8576
jobs               5489
candidates         2545
candidates/view    2244
edit-job           2108
Name: page, dtype: int64
```

In [130]:
```
print 'The top 10 countries we have results for:'
successful_contact['billing_country'].value_counts().hea
d(10)
```

```
The top 10 countries we have results for:
```

Out[130]:
```
US    52374
CA     7258
GB     6254
IN     4673
JP     3317
MX     2134
NL     1959
AU     1782
BR     1435
DE     1230
Name: billing_country, dtype: int64
```

Running through a similar set of steps for the rest of the world population, with one key exception

- I can't do any filtering on 'inside or outside business hours' because there is a ton of different timezones in the world and trying to factor them all in one place and then tie them back to EST would be way too resource intensive (the resource in this case is my brain and my precious, precious time)

In [52]:
```
successful_contact_world = successful_contact_world.drop_
duplicates(['advertiser_id', 'Contact Date', 'Contact Tim
e'], keep = 'last')
```

In [54]:
```
within_1hr_world = successful_contact_world['unixtime_y']
 - successful_contact_world['unixtime_x'] < 360
no_negative_world =
successful_contact_world['unixtime_y'] - successful_conta
ct_world['unixtime_x'] > 0
```

In [57]:
```
print 'This is how many successful escalations we receive
d in the Rest of the World OVERALL'

successful_contact_world[within_1hr_world & no_negative_w
orld]['activity_type'].value_counts()
```

```
This is how many successful escalations we received in the Rest of the World OVERALL
```

Out[57]:
```
zendeskButton       11691
CONTACT_WEB_FORM     7755
Name: activity_type, dtype: int64
```

In [62]:
```
print 'This is the top 5 pages that the advertisers were
 visiting at the time they clicked needHelp in the Rest o
f World'

successful_contact_world['page'].value_counts().head()
```

```
This is the top 5 pages that the advertisers were visiting at the time they clicked
 needHelp in the Rest of World
```

Out[62]:
```
jobs         6442
dashboard    4260
POST_JOB     2779
sponsor      2198
jobs/view    1786
Name: page, dtype: int64
```

NOW I am going to do a similar set of steps to tie phone records to advertisers - this is going to be quite the journey, I hope you are ready Freddy

- First step, I am running a PLUS call to get the company name and phone number for the advertisers that requested help
- Yet another cool function that I learned along the way

```
Syntax is:
    columns = [columns in the PLUS call I want to bring b
ack]
    new_df_name = plus.get_advertiser_cont(original_df[ad
vertiser id].unique())[columns]
```

- What the PLUS call is doing is returning ONLY the two columns I set in the *columns* variable (company, phone)
- For the unique list of advertisers from my original dataframe
- And populate that data into a new df call *phone_numbers*

In [63]:
```
columns = ['company', 'phone']
phone_numbers =
plus.get_advertiser_cont(needhelp_advids["advid"].unique()
[columns]
```

Read in phone records that I pulled from a CSV file that I used for people who called via Sandcrawler and grab only the columns that I need

```
Syntax is:
    df_name = pd.read_csv('filename.csv')[[columns that I
 want to use in the new df]]
```

Merge the advertiser phone data from the PLUS call with the original df and then filter out any blank phone numbers - since I won't be able to find those in the phone_record df

In [65]:
```
countries_phone =
no_blanks_countries.merge(phone_numbers, how = "left", le
ft_on = "advertiser_id", right_index = True)
countries_phone_two = countries_phone[countries_phone["ph
one"] != ""]
```

In [66]:
```
print len(countries_phone)
print
print len(countries_phone_two)
```

```
183029

178139
```

Create some new columns for the phone records df to tie it back to my original dataset

- Welp, unfortunately for me, the phone records df/csv doesn't have a unixtime stamp in it, so I had to create a function to convert a 'string datetime' object into a 'unixtime' object

```
Syntax is:
    function_name = lambda x: int(time.mktime(dt.datetim
e.strptime(x, '%Y-%m-%d %H:%M:%S').timetuple()))
```

- Essentially this function is saying for every row in the df, create a unixtime stamp FROM this string object
  - I have to give it the exact number of elements, as they appear in the phone records df, so the format is YEAR-MONTH-DAY 24HOUR-MINUTE-SECOND

- I then create a new column labled 'Call Log Unixtime' (because I'm so creative) and apply the new_date function to the "call_s" column in the phone records df - so I can use this to do my time delta later on

In [67]:
```
phone_record['activity_type'] = 'PHONE CALL'
new_date = lambda x:
int(time.mktime(dt.datetime.strptime(x, "%Y-%m-%d %H:%M:%
S").timetuple()))
phone_record["Call Log Unixtime"] =
phone_record["call_s"].apply(new_date)
```

I need to strip all of the special characters from the phone number information for BOTH dfs I will be merging on

- I just reassign the values to themselves since I do not have to create a separate column

```
Syntax is:
    df[column] = df[column].str.replace(r'[^0-9]+', '')
```

- What this is essentially saying is, for any NON-alphanumeric (letter or number) in the phone number fields for each of my dfs, strip those out and instead make them blank - this was a PAIN in the neck to figure out

In [68]:
```
phone_record["caller_number"] = phone_record["caller_numb
er"].str.replace(r'[^0-9]+', '')
countries_phone_two["phone"] = countries_phone_two["phon
e"].str.replace(r'[^0-9]+', '')
```

```
/Users/dmoncada/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:2: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/in
dexing.html#indexing-view-versus-copy
  from ipykernel import kernelapp as app
```

Very similar to the data munging we had to do above, I need to create Date and Time fields based on the unixtimes to be able to merge my phone records df to the advertisers who I was able to obtain a phone number via PLUS

In [77]:
```
phone_record["Phone Date"] =
[dt.datetime.fromtimestamp(x).strftime('%Y-%m-%d') for x
in phone_record['Call Log Unixtime']]
phone_record["Phone Time"] =
[dt.datetime.fromtimestamp(x).strftime('%H:%M:%S') for x
in phone_record['Call Log Unixtime']]
```

In [78]:
```
successful_phone =
countries_phone_two.merge(phone_record, how = 'inner', le
ft_on = ['phone', 'Need Help Date'],
                          right_on = ['caller_number', 'P
hone Date'])
```

Again, getting fancy with conditions and creating two different dfs based on US and non-US based advertisers

In [79]:
```
phone_usa = successful_phone['billing_country'] == 'US'
phone_world = successful_phone['billing_country'] != 'US'
```

In [80]:
```
phone_calls_usa = successful_phone[phone_usa]
phone_calls_world = successful_phone[phone_world]
```

In [81]:
```
print len(successful_phone)
print len(phone_calls_usa)
print len(phone_calls_world)
```

```
17507
15640
1867
```

Again, similar steps that I did above (and in hindsight, I could rewrite the notebook to do it all in one shot) but I needed to separate the phone data because the phone records were such a mess and trying to tie it to phone numbers via PLUS meant I didn't want to add complexity - even though this comment is probably more complex than anything I've done in the notebook

- Create a 'Time Difference' column based on when the call was logged vs when they actually clicked on 'needHelp'
- Create two conditions, one based on being within the hour, the other being the call being logged AFTER they clicked on 'needHelp'
- Reassign the df back to itself based on the two conditions
  - I don't recommend doing this the first time around when you are creating a dataset - it's really easy to screw up and have to re-run through a lot of steps from above to get back to square one... I like using .copy()

```
Syntax is:
    df_test = df.copy()
```

In [101]:
```
phone_calls_usa['Time Diff'] = phone_calls_usa['Call Log
 Unixtime'] - phone_calls_usa['unixtime']
```

```
/Users/dmoncada/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/in
dexing.html#indexing-view-versus-copy
  if __name__ == '__main__':
```

In [102]:
```
no_neg_us = phone_calls_usa['Time Diff'] > 0
less_hour_us = phone_calls_usa['Time Diff'] < 360
```

In [103]:
```
phone_calls_usa = phone_calls_usa[less_hour & no_neg]
```

Create another section of conditions for being between business hours - then split the df into two to get the values that I need

In [104]:
```
calls_eight_am = phone_calls_usa['Need Help Time'] < '0
8:00:00'
calls_eight_pm = phone_calls_usa['Need Help Time'] > '2
0:00:00'
calls_off_hours_usa = phone_calls_usa[calls_eight_am | c
alls_eight_pm]
calls_business_hours_usa = phone_calls_usa[~calls_eight_
pm & ~calls_eight_am]
```

In [114]:
```
print '''This is how many successful phone calls we rece
ived within 1HR of an advertiser clicking "needHelp" in
 the US during business hours: {}'''''.format(len(calls_
business_hours_usa))
print
print '''This is how many successful phone calls we rece
ived within 1HR of an advertiser clicking "needHelp" in
 the US OUTSIDE business hours: {}'''''.format(len(calls
_off_hours_usa))
```

```
This is how many successful phone calls we received within 1HR of an advertiser clic
king "needHelp" in the US during business hours: 10904

This is how many successful phone calls we received within 1HR of an advertiser clic
king "needHelp" in the US OUTSIDE business hours: 6
```

In [116]:
```
phone_calls_world['Time Diff'] = phone_calls_world['Call
 Log Unixtime'] - phone_calls_world['unixtime']
no_neg = phone_calls_world['Time Diff'] > 0
less_hour = phone_calls_world['Time Diff'] < 360
phone_calls_world = phone_calls_world[less_hour &
no_neg]
```

```
/Users/dmoncada/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/in
dexing.html#indexing-view-versus-copy
  if __name__ == '__main__':
```

In [120]:
```
print
print 'This is how many successful phone calls we receiv
ed within 1HR of an advertiser clicking "needHelp" in th
e Rest of the World: {}.'.format(len(phone_calls_world))
print
```

```
This is how many successful phone calls we received within 1HR of an advertiser clic
king "needHelp" in the Rest of the World: 1212.
```

In [129]:
```
print 'This is the top 5 pages that the advertisers were
 visiting when they called us in the US:'
print
print phone_calls_usa['page'].value_counts().head()
print
print 'This is the top 5 pages that the advertisers were
 visiting when they called us in the Rest of the World:'
print
print phone_calls_world['page'].value_counts().head()
```

```
This is the top 5 pages that the advertisers were visiting when they called us in th
e US:

dashboard     2520
jobs          1985
SPONSOR        923
candidates     729
sponsor        700
Name: page, dtype: int64

This is the top 5 pages that the advertisers were visiting when they called us in th
e Rest of the World:

jobs         289
dashboard    281
jobs/view    131
sponsor      110
edit-job      87
Name: page, dtype: int64
```

In [ ]:
```

```

In [ ]:
```
## THE VERY FIRST NOTES THAT I TOOK FOR THIS NOTEBOOK - T
HIS IS HOW I STARTED THE NOTEBOOK
## BLANK needs to be stripped out of the merged datafram
e.  These advertisers don't have phone numbers

## create a date field from the unixtime in the needhelp
 df
## pull in the CSV that contains the phone records
## create a unixttime from the date field in the phone re
cords
## strip the blank spaces, +, -, periods, ()
## create two datasets
#--- chat time > call time
#--- call time > chat time
# subtract unixtime from unixtime to get at this
```