Problem Set 6

Natalia Sarabia Vasquez

November 08, 2021

1 Problem 1

1.1 First part of the problem

I will connect to the database:

```
drv <- dbDriver("SQLite")
dir <- 'data'

# Path of the database
dbFilename <- 'stackoverflow-2016.db'

# Connect to the database
db <- dbConnect(drv, dbname = file.path(dir, dbFilename))

# Which tables are included in our database set up
dbListTables(db)
```

```
## [1] "answers"        "oneTag"         "questions"      "questions_tags"
## [5] "users"
```

```
# What fields does each table include?
dbListFields(db,'questions')
```

```
## [1] "questionid"   "creationdate" "score"        "viewcount"    "title"
## [6] "ownerid"
```

```
dbListFields(db,'users')
```

```
##  [1] "userid"         "creationdate"   "lastaccessdate" "location"
##  [5] "reputation"     "displayname"    "upvotes"        "downvotes"
##  [9] "age"            "accountid"
```

I will describe how I arrived to my final query:

A question can have a tag for R, or for Python or for both (and of course, of other languages). I am asked to find all the users that have asked at least one question of R and at least question of Python.

The way I thought about this problem is the following. First, I have to find the labels associated to each question and filter just the questions that at least have one of the labels I am interested on, that is 'R' or 'Python'. Then, I will summarize the information grouping by ownerid and tag (tag = 'Python' or 'R') and count this observations. For me, the result of the count is not important (for instance, I could have calculated the mean, etc.), this step is just an auxiliary step to know if a user have asked questions of Python, or R or both:

```
# How I constructed my query step-by-step:
# Here, I did what I just described. I performed a join between the tables
```

```
# questions_tags and questions to associate each question with its tags. Then, I
# filtered those I am interested on: Python and R related questions. Finally, I
# grouped by ownerid (the person who asked the question) and the tag:

subquery <- dbGetQuery(db,     "SELECT ownerid, tag, count(*) as count
                               FROM questions
                               LEFT JOIN questions_tags
                               ON questions.questionid = questions_tags.questionid
                               WHERE (tag == 'python') OR (tag == 'r')
                               GROUP BY ownerid, tag")
# Here is how it looks like:
head(subquery, n = 16)
```

```
##    ownerid    tag count
## 1       NA python   949
## 2       NA      r   178
## 3       56 python     1
## 4      116 python     6
## 5      150 python     1
## 6      194 python     2
## 7      258 python     1
## 8      260 python     1
## 9      357      r     1
## 10     459 python     2
## 11     476 python     3
## 12     688 python     1
## 13     688      r     1
## 14     740      r     1
## 15     826 python     1
## 16     956 python     1
```

For instance, after seeing the table, I know I will be interested in user 688 (because she has a row for Python and a row for R). She will be the only user I want to keep from the 16 I rows I showed, supposing that was the entire table.

I will generalize this idea and apply it to my big table. I observed that the data table contains missing values in the column 'ownerid' so I will remove them. The next step is to group the table by ownerid and count how many rows I have per 'ownerid'. This time the counting is important. I would like to keep those users which count is exactly equal to two (they have asked questions at least one question of R and at least one question of Python, one of their questions could have both labels, but my solution also incorporates this case):

```
# The final query uses the subquery I performed in the last chunk. Now, I will count
# the users that appear two times in my subquery. That will mean that they have
# asked questions with both tags or at least one including each tag:

query <- dbGetQuery(db, "SELECT ownerid, count(*) as count2
                         FROM    (SELECT ownerid, tag, count(*) as count
                                 FROM questions
                                 LEFT JOIN questions_tags
                                 ON questions.questionid = questions_tags.questionid
                                 WHERE (tag == 'python') OR (tag == 'r')
                                 GROUP BY ownerid, tag)
                         WHERE ownerid IS NOT NULL
                         GROUP BY ownerid
                         HAVING count2 == 2")
```

```
head(query, n = 10)
```

```
##    ownerid count2
## 1      688      2
## 2     2118      2
## 3     7648      2
## 4    15485      2
## 5    19410      2
## 6    34935      2
## 7    40106      2
## 8    41977      2
## 9    46503      2
## 10   51167      2
```

1.1.1 Putting all together:

Finally, I just have to count the rows:

```
# I included an extra count just to use pure SQL syntax. Note that it was no
# necessary. I could have complete my query and then use R syntax to count it.
# An nrow() would be enough. Therefore, the outer select just counts and
# computes the answer I was asked:

python_r <- dbGetQuery(db, " SELECT count(*)
                           FROM    (SELECT ownerid, count(*) as count2
                                    FROM    (SELECT ownerid, tag, count(*) as count
                                             FROM questions
                                             LEFT JOIN questions_tags
                                  ON questions.questionid = questions_tags.questionid
                                             WHERE (tag == 'python') OR (tag == 'r')
                                             GROUP BY ownerid, tag)
                                    WHERE ownerid IS NOT NULL
                                    GROUP BY ownerid
                                    HAVING count2 == 2)")

python_r
```

```
##   count(*)
## 1     2567
```

Finally, I have 2,567 users who have asked at least one question of R and at least one question of Python, and also the users who asked a question with both tags.

1.2 Second part of the problem

For the second part of the problem, I am interested in removing the users who asked questions that have both tags. It is clear that the query I used to solve the first problem will be exactly the same. What I need to modify is one of the sets I am performing the first join. In other words, I have to create a filtered version of questions_tags table.

First, I will keep only the questions that have the tag 'R' or 'Python', but not both. That is, from the table questions_tags, I will filter the questions with tags 'R', 'Python' or both and then group by questionid and count how many tags a question has. I am interested in the questions with ONE tag (of the possible ones 'R' or 'Python', meaning I don't want questions associated to these two tags):

```
# From questions_tags I filter by the tag 'R' or 'Python'. Then, I group by questionid
# and will keep just the questions associated to one label (thus, eliminating the
# ones associated to both labels):

query <- dbGetQuery(db, "SELECT questionid, count(*) as count
                                FROM questions_tags
                                WHERE (tag == 'python') OR (tag == 'r')
                                GROUP BY questionid
                                HAVING count == 1")

head(query, n = 10)
```

```
##    questionid count
## 1    34552552     1
## 2    34552584     1
## 3    34552653     1
## 4    34552670     1
## 5    34552671     1
## 6    34552672     1
## 7    34552706     1
## 8    34552770     1
## 9    34552809     1
## 10   34552846     1
```

Once filtered, my idea is to 'generate' a new version of the table 'questions_tags' containing only the questionid and the tags of the questions that do not include both 'Python' and 'R' tags. This will be my new version of the 'questions_tags' table and the one I will use in my original query to obtain the desired output:

```
# Here I created my filtered version of the table questions_tags. I will use the
# subquery described in the last chunk. Then, I will perform a left join between the
# two tables to add the column tag. Again, I just kept relevant tags:
query <- dbGetQuery(db, "SELECT aux.questionid, tag
                                FROM (SELECT questionid, count(*) as count
                                     FROM questions_tags
                                     WHERE (tag == 'python') OR (tag == 'r')
                                     GROUP BY questionid
                                     HAVING count == 1) as aux
                                JOIN questions_tags
                                ON aux.questionid = questions_tags.questionid
                                WHERE (tag == 'python') OR (tag == 'r')")

head(query, n = 10)
```

```
##    questionid    tag
## 1    34553225      r
## 2    34553559 python
## 3    34556493 python
## 4    34557898 python
## 5    34560088 python
## 6    34560213 python
## 7    34560740 python
## 8    34560760 python
## 9    34560905 python
## 10   34561311 python
```

1.2.1 Putting all together:

Next step, I will create the view of the query I described in the last chunk. And finally, execute my query for the first part of the problem in this new version of the table questions_tags (VIEW oneTag):

```
# To remove the view from my prevois executions. This is not necessary if you
# do not have a VIEW called oneTag in memory:
dbExecute(db,"DROP VIEW oneTag")
```

```
## [1] 0
```

```
# I create my new version of the table questions_tags using a view:
dbExecute(db, "CREATE VIEW oneTag as
                               SELECT aux.questionid, tag
                               FROM (SELECT questionid, count(*) as count
                                    FROM questions_tags
                                    WHERE (tag == 'python') OR (tag == 'r')
                                    GROUP BY questionid
                                    HAVING count == 1) as aux
                               JOIN questions_tags
                               ON aux.questionid = questions_tags.questionid
                               WHERE (tag == 'python') OR (tag == 'r')")
```

```
## [1] 0
```

```
# And then execute my old query just changing the questions_tags table by its new
# version (oneTag):
query <- dbGetQuery(db, "SELECT count(*)
                         FROM    (SELECT ownerid, count(*) as count2
                                  FROM    (SELECT ownerid, tag, count(*) as count
                                           FROM questions
                                           LEFT JOIN oneTag
                                           ON questions.questionid = oneTag.questionid
                                           WHERE (tag == 'python') OR (tag == 'r')
                                           GROUP BY ownerid, tag)
                                  WHERE ownerid IS NOT NULL
                                  GROUP BY ownerid
                                  HAVING count2 == 2)")

query
```

```
##   count(*)
## 1     2221
```

Finally, I have 2,221 users who have asked at least one question of R and at least one question of Python, excluding the users who asked a question with both tags.

2 Problem 2

I first worked on my Python script in my local computer using Spyder:

```
# Importing libraries
import dask.multiprocessing
import dask.bag as db
import re
import time
```

```
# I have to wrap my code in the following instruction or it won't run:
if __name__ == '__main__':
    ## Define the set up for the multiprocessing
    ## I assigned 16 workers
   dask.config.set(scheduler='processes', num_workers = 16)

   ## Address of the full data
   path = '/var/local/s243/wikistats/dated_2017/'

   ## Read all the files starting with 'part-00' and ending with 'gz'
   wiki = db.read_text(path + 'part-00*gz')

   ## Define a function that will filter my data with the help of one regular
   ## expression. I will search for the string Barack_Obama in the column
   ## of the webpage and filter the observations by the ones in language == EN
   ## after
   def find(line, regex = 'Barack_Obama'):
    vals = line.split(' ')
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[3])
    if tmp is None:
        return(False)
    else:
        return(True)

   ## Filter my data
   df = wiki.filter(find)

   ## Convert the bags to a pandas data frame. First:
   def make_tuple(line):
       return(tuple(line.split(' ')))

   ## Initialize the types of each column:
   dtypes = {'date': 'object', 'time': 'object', 'language': 'object',
   'webpage': 'object', 'hits': 'float64', 'size': 'float64'}

   ## Create a Dask dataframe.
   obama_en = df.map(make_tuple).to_dataframe(dtypes)

   ## Create the Pandas df
   t0 = time.time()
   result = obama_en.compute()
   t1 = time.time()
   time = t1 - t0

   print(time)

   type(result)

   result = result[result['language'].str.contains("en")]

   ## Check how the result looks like:
```

```
   result.head()

   ## Group by day-hour
   result_ = result.groupby(["date", "time"])["hits"].sum()
   ## Export to a csv. I will then plot my results using Python on my local
   ## machine:
   result_.to_csv('obama_python.csv')
```

I also setup my Obama file that will help me to execute MyScript.py:

```
#!/bin/bash

#####################
# SBATCH OPTIONS
#####################
#SBATCH --job-name=obama_python          # job name fore queue, default may be u$
#SBATCH --partition=low                  # high/low/gpu, default if empty is low
#SBATCH --error=obama.err                # error file, default if empty is slurm$
#SBATCH --output=obama.out               # standard out file, no default
#SBATCH --time=05:00:00                  # optional, max runtime of job h:m:s
#SBATCH --nodes=1                        # only use 1 node, MPI option
#SBATCH --ntasks=1                       # how many tasks to start
#SBATCH --cpus-per-task=16               # number of cores to use, multi-core/mu$

#####################
# What to run
#####################

python MyScript.py > obama.pyout
```

After this, I copied two files from '/var/local/s243/wikistats/dated_2017/' to my arwen machine. I sent MyScript.py and Obama (batch) files to my arwen account and I executed my script (changing the path to my local computer) on these two files to test everything was working as expected. I adjusted MyScript.py (set the path of all the files) and then I ran my script in all the files:

```
# reticulate::py_install("pandas")

# While at my local machine, I sent MyScript.py and Obama  (batch) to my arwen account:
scp MyScript.py Obama natalia_sarabia10@arwen.berkeley.edu:~/

# Connect to my SCF account:
ssh natalia_sarabia10@arwen.berkeley.edu

# Execute my Script with:
sbatch Obama

# MyScript.py creates a obama_python.csv file, with the grouping by date and hour.
# I sent back the output to my local machine to make the plots:
scp natalia_sarabia10@arwen.berkeley.edu:~/obama_python.csv ~/Documents/STAT243/PS6/.
```

```
include_graphics('Obama.png')
```

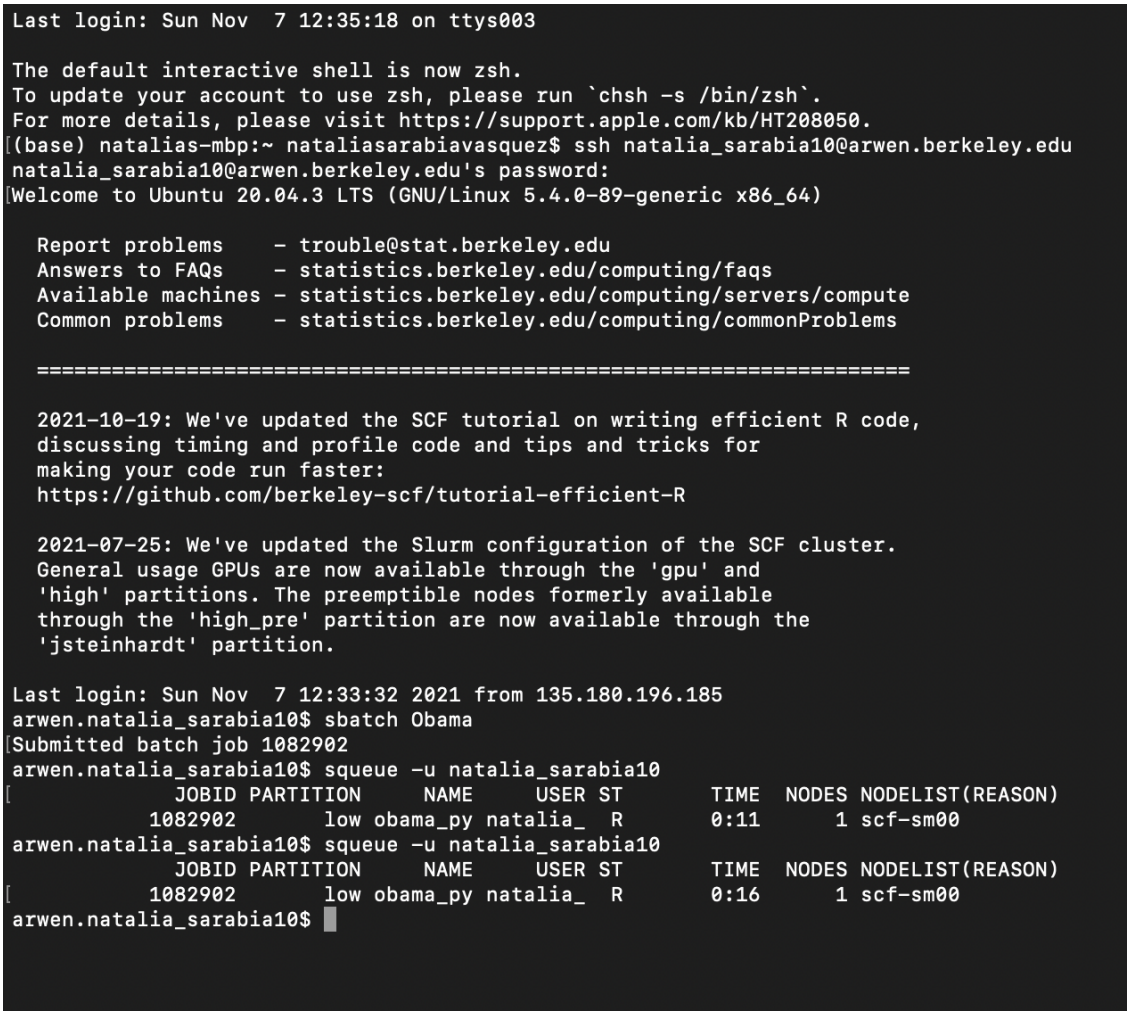


Figure 1: Obama

I also exported the file obama.pyout and this is the time it took to produce the final output (the .csv):

```
5108.481792211533
```

The units of this quantity are seconds. Translated into hours, this amount of data took 1.419023 hrs.

Back in my local computer, I can plot the findings:

```
# Import required packages
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter
import re

# Read the data
df = pd.read_csv('~/Documents/STAT243/PS6/obama_python.csv', sep=',')
df.dtypes

# Change to string type the variable date. This will allow to subset it and
# extract day, year and month. After that, we convert back to number to be able
# to use the pd.to_datetime() function:
```

```
## date      int64
## time      int64
## hits    float64
## dtype: object
```

```
df['date'] = df['date'].astype(str)
df['year'] = pd.to_numeric(df['date'].str[0:4])
df['month'] = pd.to_numeric(df['date'].str[4:6])
df['day'] = pd.to_numeric(df['date'].str[6:8])

# Treat the hour variable. By default it is given as an integer, we divide by 10000
# and also add the field to the function pd.to_datetime():
df['hour'] = round(df['time'] / 10000)

# Create a variable with the date and time:
df['time_date'] = pd.to_datetime(df[['year', 'month', 'day', 'hour']],unit='ms').dt.tz_localize('UTC')

# Change the time zone to US/EST
df['time_date'] = df['time_date'].dt.tz_convert('US/Eastern')

# Just formatting: I will divide the number of hits by 1000 in order to make easier
# the reading of the 'y' axis:
df['hits'] = df['hits'] / 1000

# A quick look on how my data set looks like:
df.head()
```

```
##        date   time   hits  year  month  day  hour                 time_date
## 0  20081001      0  4.200  2008     10    1   0.0 2008-09-30 20:00:00-04:00
## 1  20081001  10000  4.270  2008     10    1   1.0 2008-09-30 21:00:00-04:00
## 2  20081001  20000  4.068  2008     10    1   2.0 2008-09-30 22:00:00-04:00
## 3  20081001  30000  3.973  2008     10    1   3.0 2008-09-30 23:00:00-04:00
## 4  20081001  40000  3.273  2008     10    1   4.0 2008-10-01 00:00:00-04:00
```

I plot my results using Python:

```
# Close all the open windows for a plot
plt.close('all')

# Create the time series plot using a dot as a marker:
fig, ax = plt.subplots()
ax.plot_date(df['time_date'],df['hits'], linestyle='-',color='blue',tz='US/Eastern',linewidth = 0.4, ms

# Giving format to the x axis:
```

```
## [<matplotlib.lines.Line2D object at 0x7f8692c21748>]
```

```
fig.autofmt_xdate()
ax.xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H:%M'))

# Giving format to the plot in general:
fig.suptitle('Wikipedia hits containing the string Barack_Obama', fontsize=12)
```

```
## Text(0.5, 0.98, 'Wikipedia hits containing the string Barack_Obama')
```

```
plt.xlabel('Time', fontsize=10)
```

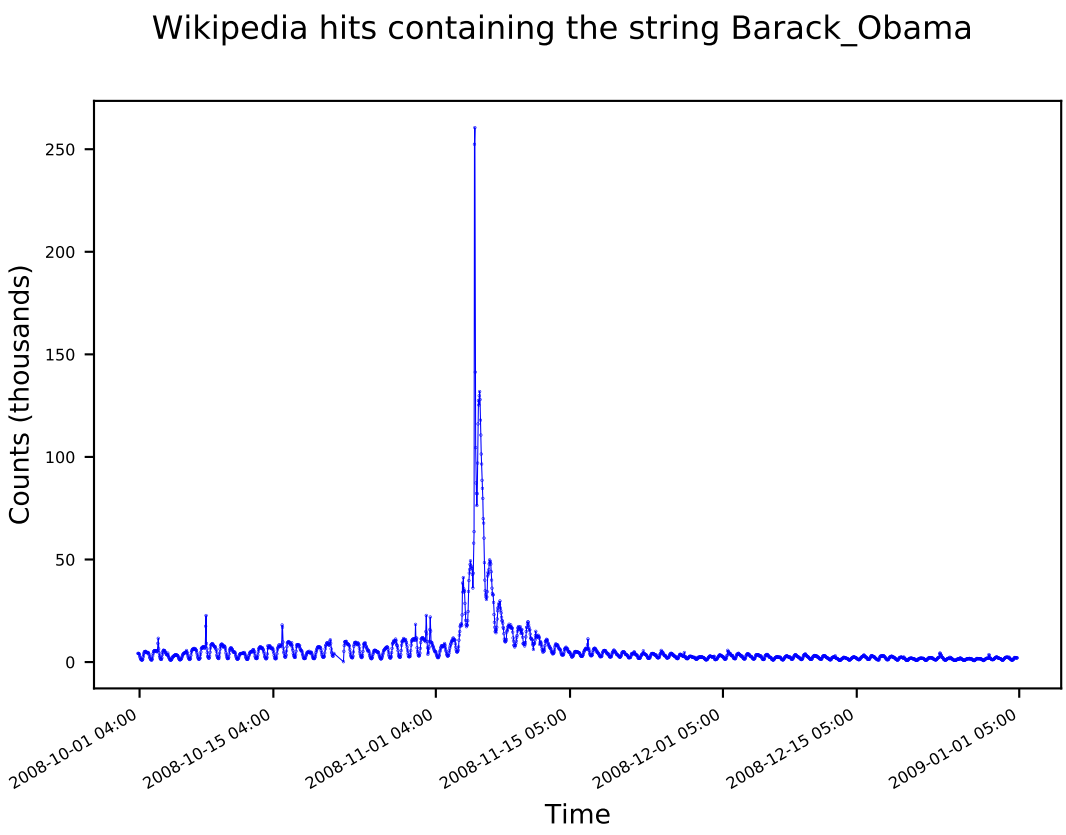
```
## Text(0.5, 0, 'Time')
```

```
plt.ylabel('Counts (thousands)', fontsize=10)
```

```
## Text(0, 0.5, 'Counts (thousands)')
```

```
plt.tick_params(labelsize=6)

# Actually show the plot:
plt.show()
```



I would like to make a zoom to this graph. Specifically, I would like to analize the behaviour of the hits in a window near the elections. I will filter the data to analyze 3-6 November, 2008 and plot the findings:

```
# Filter by day and month
election = df[((df['day']==3) | (df['day']==4) | (df['day']==5) |
(df['day']==6)) & (df['month']==11)]

# Close all the open windows for a plot
plt.close('all')

# Create the time series plot using a dot as a marker:
fig, ax = plt.subplots()
ax.plot_date(election['time_date'],election['hits'], marker='.',
linestyle='-',color='blue',tz='US/Eastern')

# Giving format to the x axis:
```

```
## [<matplotlib.lines.Line2D object at 0x7f86781b08d0>]
```

```
fig.autofmt_xdate()
ax.xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H:%M'))

# Giving format to the plot in general:
fig.suptitle(
  'Wikipedia hits containing the string Barack_Obama close the election'
  , fontsize=8)
```

```
## Text(0.5, 0.98, 'Wikipedia hits containing the string Barack_Obama close the election')
```

```
plt.xlabel('Time', fontsize=10)
```

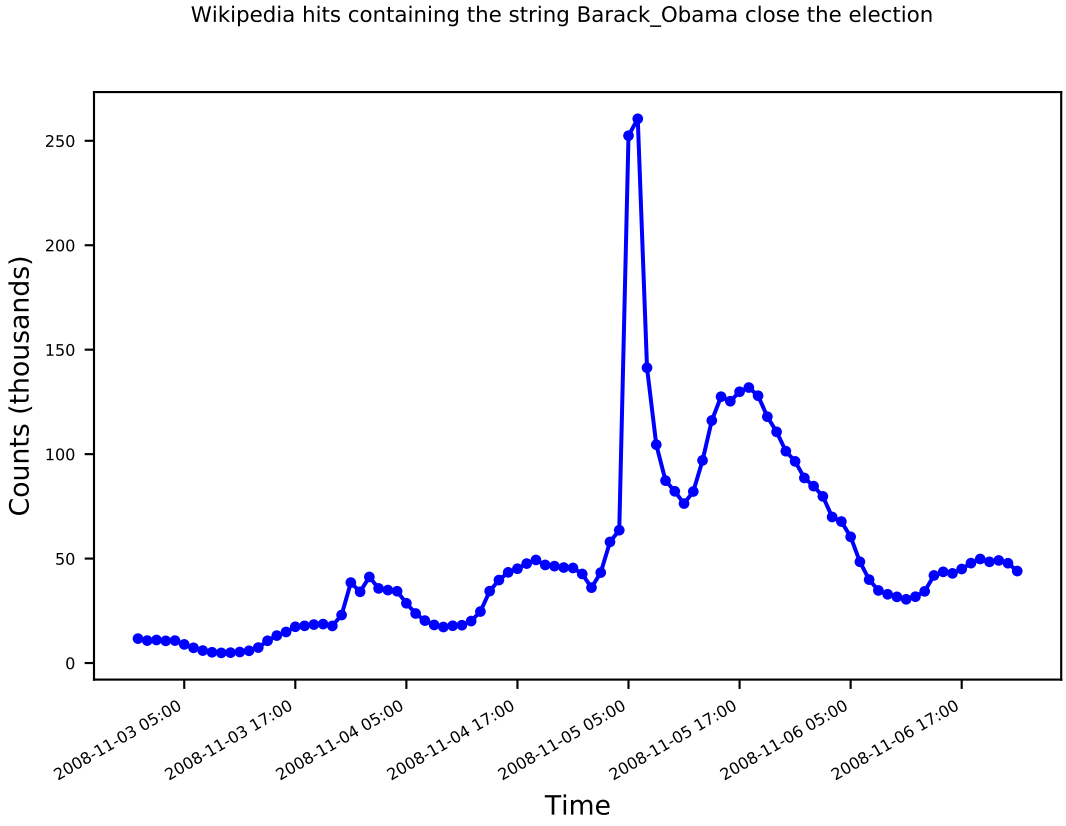
```
## Text(0.5, 0, 'Time')
```

```
plt.ylabel('Counts (thousands)', fontsize=10)
```

```
## Text(0, 0.5, 'Counts (thousands)')
```

```
plt.tick_params(labelsize=6)

# Actually show the plot:
plt.show()
```



I appears that the number of hits on Wikipedia websites containing the string 'Barack_Obama' increased substantially on the following early morning of the elections (For websites in English).

2.1 Extra credit:

Remember that another important event occurred in 2008. The financial crisis hit during the 4th quarter of that year. As the time window coincides with the one I have, I will analize the number of hits on webpages containing the string 'Financial_crisis'. I will split the analysis on different time series grouped by language:

I first worked on my Python script in my local computer using Spyder:

```
# Importing libraries
import dask.multiprocessing
import dask.bag as db
import re
import time

# I have to wrap my code in the following instruction or it won't run:
if __name__ == '__main__':
    ## Define the set up for the multiprocessing
    ## I assigned 16 workers
   dask.config.set(scheduler='processes', num_workers = 16)

   ## Address full data
   path = '/var/local/s243/wikistats/dated_2017/'

   ## Read all the files starting with 'part-00' and ending with 'gz'
   wiki = db.read_text(path + 'part-00*gz')

   ## Define a function that will filter my data with the help of one regular
   ## expression on the string 'financial_crisis'. I will ignore the case of
   ## the words I match:
   def find(line, regex = 'Financial_crisis'):
       vals = line.split(' ')
       if len(vals) < 6:
          return(False)
       tmp = re.search(regex, vals[3], re.IGNORECASE)
       if tmp is None:
          return(False)
       else:
          return(True)

   ## Filter my data
   df = wiki.filter(find)

   ## Convert the bags to a pandas data frame. First:
   def make_tuple(line):
       return(tuple(line.split(' ')))

   ## Initialize the types of each column:
   dtypes = {'date': 'object', 'time': 'object', 'language': 'object',
   'webpage': 'object', 'hits': 'float64', 'size': 'float64'}

   ## Create a Dask dataframe.
   crisis = df.map(make_tuple).to_dataframe(dtypes)

   ## Create the Pandas df
   t0 = time.time()
```

```
result = crisis.compute()
t1 = time.time()
time = t1 - t0

print(time)

type(result)

## Check how the result looks like:
result.head()

## Group by day-hour
result_ = result.groupby(["date", "time","language"])["hits"].sum()

## Export to a csv. I will then plot my results using Python on my local
## machine:
result_.to_csv('crisis_python.csv')
```

I also setup my Extra file that will help me to execute MyScript_extra.py:

```
#!/bin/bash

######################
# SBATCH OPTIONS
######################
#SBATCH --job-name=extra_python          # job name fore queue, default may be u$
#SBATCH --partition=low                  # high/low/gpu, default if empty is low
#SBATCH --error=extra.err                # error file, default if empty is slurm$
#SBATCH --output=extra.out               # standard out file, no default
#SBATCH --time=05:00:00                  # optional, max runtime of job h:m:s
#SBATCH --nodes=1                        # only use 1 node, MPI option
#SBATCH --ntasks=1                       # how many tasks to start
#SBATCH --cpus-per-task=16               # number of cores to use, multi-core/mu$

######################
# What to run
######################

python3 MyScript_extra.py > extra.pyout
```

I repeated the same steps as in the previous exercise but using my new files:

```
# While at my local machine, I sent MyScript_extra.py and Extra (batch) to my arwen account:
scp MyScript_extra.py Extra natalia_sarabia10@arwen.berkeley.edu:~/

# Connect to my SCF account:
ssh natalia_sarabia10@arwen.berkeley.edu

# Execute my Script with:
sbatch Extra

# MyScript_extra.py creates a crisis_python.csv file, with the grouping by date,
# time and language. I sent back the output to my local machine to make the plots:
scp natalia_sarabia10@arwen.berkeley.edu:~/crisis_python.csv ~/Documents/STAT243/PS6/.
```

```
include_graphics('Extra.png')
```

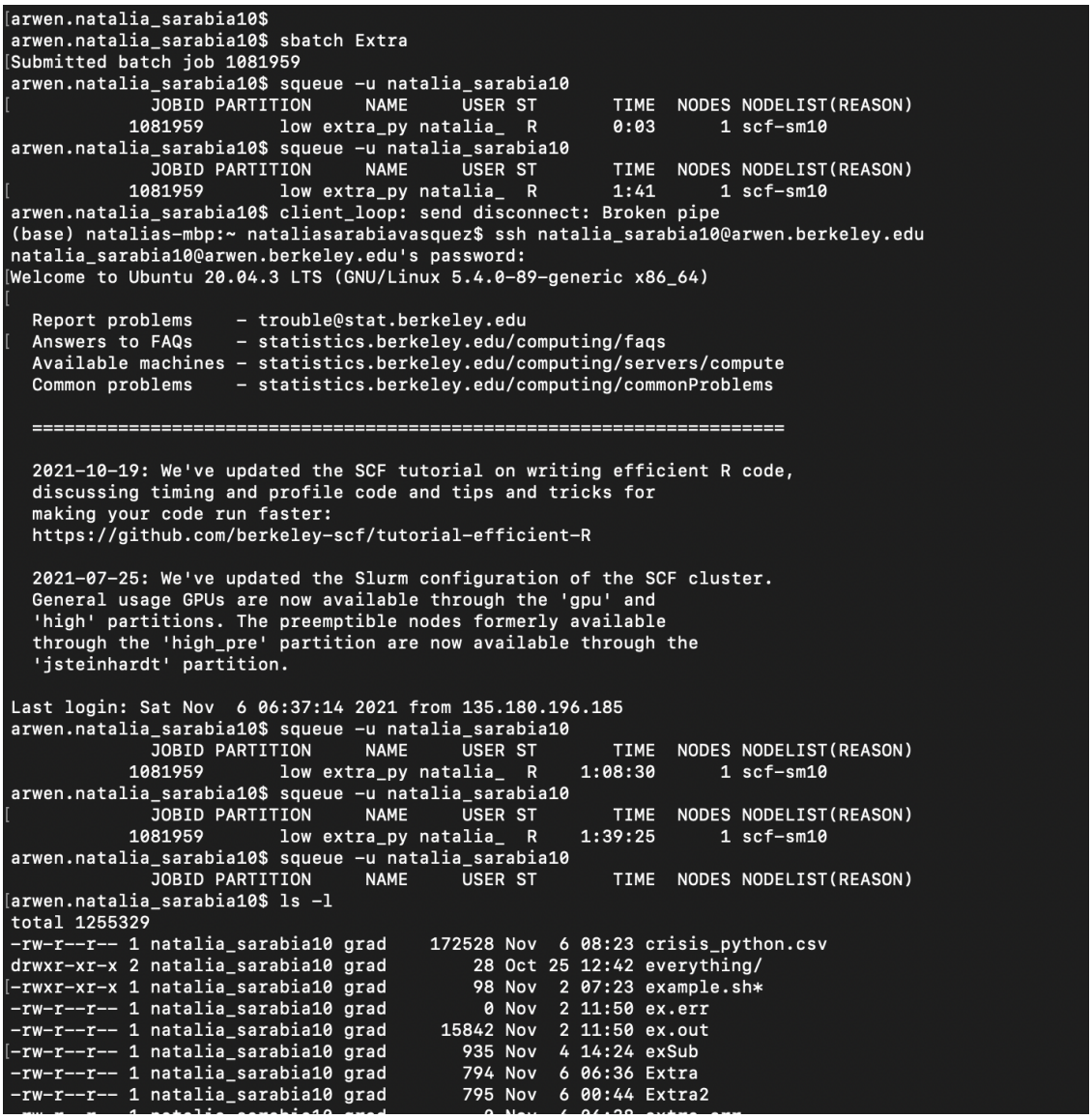


Figure 2: Extra

```
# Read the data
df = pd.read_csv('~/Documents/STAT243/PS6/crisis_python.csv', sep=',')
df.dtypes

# Change to string type the variable date. This will allow to subset it and
# extract day, year and month. After that, we convert back to number to be able
# to use the pd.to_datetime() function:
```

```
## date          int64
## time          int64
## language     object
## hits        float64
```

```
## dtype: object
```

```
df['date'] = df['date'].astype(str)
df['year'] = pd.to_numeric(df['date'].str[0:4])
df['month'] = pd.to_numeric(df['date'].str[4:6])
df['day'] = pd.to_numeric(df['date'].str[6:8])

# Treat the hour variable. By default it is given as an integer, we divide by 10000
# and also add the field to the function pd.to_datetime():
df['hour'] = round(df['time'] / 10000)

# Create a variable with the date and time:
df['time_date'] = pd.to_datetime(df[['year', 'month', 'day', 'hour']],unit='ms').dt.tz_localize('UTC')

# Change the time zone to US/EST
df['time_date'] = df['time_date'].dt.tz_convert('US/Eastern')

# Just formatting: I will divide the number of hits by 1000 in order to make easier
# the reading of the 'y' axis:
df['hits'] = df['hits'] / 1000

# A quick look on how my data set looks like:
df.head()
```

```
##        date   time language   hits  ...  month  day  hour                 time_date
## 0  20081001      0       en  0.528  ...     10    1   0.0 2008-09-30 20:00:00-04:00
## 1  20081001      0     en.n  0.001  ...     10    1   0.0 2008-09-30 20:00:00-04:00
## 2  20081001  10000       en  0.550  ...     10    1   1.0 2008-09-30 21:00:00-04:00
## 3  20081001  20000       en  0.491  ...     10    1   2.0 2008-09-30 22:00:00-04:00
## 4  20081001  30000       de  0.001  ...     10    1   3.0 2008-09-30 23:00:00-04:00
##
## [5 rows x 9 columns]
```

I have a table with different languages. I will perform some processing to the data before ploting my final results:

```
# I have many languages with variants. I will work on them. First, I will remove the
# variant of the language value and then group and count again by the 'new' language
# variable:
df['language2'] =  df['language'].apply(lambda x: re.sub(r'\..*','', str(x)))

# Give format to the data frame to make the plot process, easier. I converted it to
# what is often called a wider representation. I also filled the NaN with zeroes
# to have better graphs:
crisis = df.groupby(['time_date','language2'])['hits'].sum().reset_index()
crisis = crisis.pivot_table(index = 'time_date',
columns = 'language2', values = 'hits').reset_index().fillna(0)

# Close all the open windows for a plot
plt.close('all')

# Create the time series plot using different markers and colors:
fig1, axs = plt.subplots(2,2)
axs[0,0].plot_date(crisis['time_date'],crisis['en'], marker='.',
linestyle='-',color='blue',tz='US/Eastern',linewidth = 0.2, ms = 0.2)
```

```
## [<matplotlib.lines.Line2D object at 0x7f869301c518>]
```

```
axs[0,1].plot_date(crisis['time_date'],crisis['zh'], marker='^',
linestyle='-',color='green',tz='US/Eastern',linewidth = 0.2, ms = 2)
```

```
## [<matplotlib.lines.Line2D object at 0x7f869301cb00>]
```

```
axs[1,0].plot_date(crisis['time_date'],crisis['es'], marker='+',
linestyle='-',color='red',tz='US/Eastern',linewidth = 0.2, ms = 2)
```

```
## [<matplotlib.lines.Line2D object at 0x7f869301cfd0>]
```

```
axs[1,1].plot_date(crisis['time_date'],crisis['de'], marker='x',
linestyle='-',color='orange',tz='US/Eastern',linewidth = 0.2, ms = 2)

# Giving format to the x axis:
```

```
## [<matplotlib.lines.Line2D object at 0x7f86930b7710>]
```

```
fig1.autofmt_xdate()
axs[0,0].xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H:%M'))
axs[0,1].xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H:%M'))
axs[1,0].xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H:%M'))
axs[1,1].xaxis.set_major_formatter(DateFormatter('%Y-%m-%d %H:%M'))

# Set a title for each subgraph
axs[0,0].set_title(f'English',fontsize=9)
```

```
## Text(0.5, 1.0, 'English')
```

```
axs[0,1].set_title(f'Chinese',fontsize=9)
```

```
## Text(0.5, 1.0, 'Chinese')
```

```
axs[1,0].set_title(f'Spanish',fontsize=9)
```

```
## Text(0.5, 1.0, 'Spanish')
```

```
axs[1,1].set_title(f'German',fontsize=9)

# Giving format to the plot in general:
```

```
## Text(0.5, 1.0, 'German')
```

```
fig1.suptitle('Wikipedia hits containing the string "Financial_crisis"', fontsize=12)

# Adding title and x and y labels
```

```
## Text(0.5, 0.98, 'Wikipedia hits containing the string "Financial_crisis"')
```

```
fig1.text(0.5, 0.03, '$Time$', ha='center', va='center', fontsize=8)
```

```
## Text(0.5, 0.03, '$Time$')
```

```
fig1.text(0.04, 0.5, '$Counts \  (thousands)$', ha='center', va='center',
rotation='vertical', fontsize=8)

# Set the size of the labels of both axis for the 4 subplots:
```

```
## Text(0.04, 0.5, '$Counts \\  (thousands)$')
```

```
axs[0,0].tick_params(labelsize=6)
axs[0,1].tick_params(labelsize=6)
axs[1,0].tick_params(labelsize=6)
axs[1,1].tick_params(labelsize=6)

# Show the plot. For a strange reason, I was not able to show the plot. Every time I
# tried to show this specific plot, the other plots were acting very strange. For this
# reason, I decided to generate the plot, save it as an image and then call the image.
# This problem only arise with this plot:
plt.savefig('extra_plot.png')
```

I present my findings in the following plot:

```
include_graphics('extra_plot.png')
```

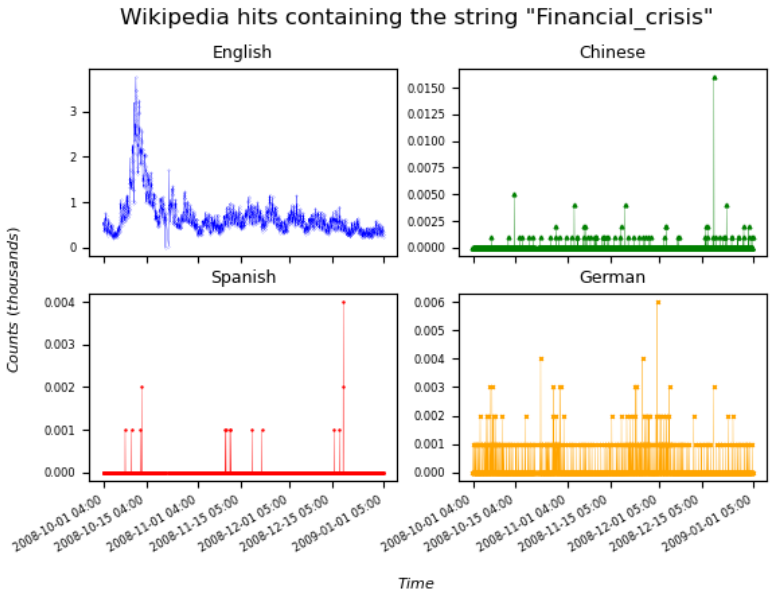


Figure 3: Financial crisis

Two things are interesting to me. First of all, the webpages on English were, by far the most popular. It would be interesting to know if it has something to do that the internet usage was more developed in countries where English is spoken, or if most pages were in English, or there might be another reason. The other interesting aspect is about the times in which we observed the outliers across the languages. I would have thought that they were not very far one from the other (in time), because although different countries have different time zones, a difference of a day would just shift a little bit the graphs.

3 Problem 3

Consider a simulation study that generates m = 100 simulated datasets. The parameter of interest is θ and in simulating the datasets, we have $\theta = 2$. The value of 2 is included in 85 of the 95% confidence intervals.

a. If you're interested in the coverage of the confidence interval procedure, what is $h(Y_i)$ in this setting? What is the expectation that is of interest here?

In this case, we are interested in the estimation of a probability, specifically (by the results seen in class):

$h(Y_i) = \mathbb{1}_{\theta \in CI(Y_i)}$ and the expectation of interest is: $\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{\theta \in CI(Y_i)}$

b. Based on the Monte Carlo uncertainty of the expectation of interest, do you think you have simulated enough datasets? Note that this is a somewhat subjective judgment.

My first thought was that I would expect to have θ been included in ≈ 95 of the intervals. In that case, I believe I don't have enough simulated datasets. So I could run more simulations. But then I talked to one of my classmates (Krissi Alari), and she explained me the following:

I am actually able to calculate the MC simulation error applying the formula we reviewed in class: $\hat{Var}(\hat{\theta}) = \frac{1}{m(m-1)} \sum_{i=1}^{m} (h(Y_i) - \hat{\theta})^2 = \frac{1}{100*99}(85 * (1 - 0.85)^2 + 15 * (0 - 0.85)^2) = 0.001287879$

Let's think about this result. This means that the estimated variance of our estimator is not really big, and therefore with this amount of simulated data, the variance is small and we are still far from the true parameter. This could potentially indicate that even if we increase the amount of simulations, we will not be able to have a 'better' approximation of $\hat{\theta}$. In any case, if there are no computational constrains, I will increase the number of simulations and see what happens.