STAT243 - Problem Set 6

Cristobal Pais - cpaismz@berkeley.edu

November 1st, 2017

Problem 1: Simulation article

a) What are the goals of their simulation study and what are the metrics that they consider in assessing their method?

1. The main and specific goals of the simulation study are

   i) Develop and demonstrate the usefulness and accuracy of a new (adjusted) statistical test for determining if a random sample is drawn from a n-component normal mixture distribution or, as the alternative hypothesis if the sample is drawn from a l-component normal mixture.

   This test is derived from a theorem proposed by Vuong that it is extended in the current article and uses the likelihood ratio statistic based on the known Kullback-Leibler information criterion. One of the most relevant characteristics of this implementation is that "the results do not require that either of the competing models be correctly specified".

   ii) Demonstrate that the asymptotic distribution of the used LR statistic follows a distribution composed by a weighted sum of chi-squared random variables (independent) with one-degree of freedom when some assumptions (regularity conditions) are satisfied.

2. The metrics that the authors consider in assessing their method are:

   i) The authors are comparing the P-values obtained by both the unadjusted and adjusted models for different significance levels (95% and 99%) for each configuration implemented.

   ii) A second metric used consists of the value of "Simulated powers, in percentages" in order to compare different configurations and significance levels tested, as seen in Table 2.

   iii) "Simulated significance levels, in percentages" as seen in Table 3 when comparing the results of the unadjusted and adjusted models for a mixture of two normals versus a mixture of three normals using 1000 replications for estimating the parameters.

b) What choices did the authors have to make in designing their simulation study? What are the key aspects of the data generating mechanism that likely affect the statistical power of the test? Are there data-generating scenarios that the authors did not consider that would be useful to consider?

1. **Design**

   The choices that the authors made in order to design their simulation study were:

   i) The algorithm used to estimate the maximum likelihood value associated with the parameters/configuration under study. They implemented the EM algorithm.

   ii) The number of sets used for random starting values parameters. The selected size was 100 sets of random starting values.

   iii) The truncation error threshold. Defined as less than $10e-7$.

   iv) μ and σ values

   v) Sample sizes: $n = 50$, 75, 100, 150, 200, 300, 400, 500 and 1000 were considered.

vi) The number of replications per sample. 1000 replications were performed per sample size.

vii) Alpha values (significance level) tested. In this study, the authors selected the classic 0.05 and 0.01 values.

viii) Values of D, the variable that measures the distance between the two components of the normal mixture distribution.

ix) Mixing proportion π parameter.

2. **Key aspects**

As we will discuss in the last question of the current section, the key aspects of the data generating mechanism that likely affect the statistical power of the test are:

i) **Sample size**: As the authors indicate in the paper, the power of the test is worse (even poor) when the sample size n is less than 200 elements. In particular, this situation arises when the two components of the distribution are not well separated (when $D = 1$ or 2). In addition, a sample of at least 100 is required in order to reach a reasonable power when the components are pretty separated ($D = 3$).

Therefore, we can clearly see the key role of the sampling size, associated with the value of D. Different configurations are tested.

ii) **The number of replications:** Not explicitly mentioned in the article but the number of replications performed is clearly one of the most important elements when conducting a simulation study. The authors do not discuss how they decide the number of replications (1000), however, they are clearly taking into account possible bias and/or errors produced when the size of the sampling is not enough for obtaining a small confidence interval for the estimated parameters such that the "real" value that theywant to estimate is associated with a large error margin. More discussion about this phenomenon is found in the last question of this section.

iii) D **parameter value**: Depending on the distance between the components, the authors find that the statistical power of the test is significantly affected. Based on the results in Table 2 and 3, the authors indicate that the "approximation of the nominal sizes to the actual sizes of the unadjusted test is inadequate for $D = 2$ or 3".

Following the same logic, the article indicates that the rate of convergence obtained for the adjusted test toward the asymptotic distribution is affected by the amount of space between the components of the distribution and also, by the mixing proportion. However, besides the mixing proportion selected, we have that the distribution of the adjusted test experiences a very good convergence when $D = 1$ or 2. On the other hand, when $D = 3$ the authors found that the approximation is not very accurate but "acceptable".

As a conclusion, the authors indicate that "The results indicate that the factors with most influence on power are the spacings $D1$ and $D2$".

3. **Extra scenarios**

An interesting scenario for testing purposes would be to model distributions with different standard deviations/ σ (heteroscedasticity) because the paper covers only homoscedastic cases. Different convergence and limiting properties should be studied and contrasted in order to perform and implement the adjusted version of the test.

As the authors mention in the future work section, we think that an interesting setting would be to extend the current adjusted test to other distributions such as exponential, Poisson, gamma, etc.

c) Do their tables do a good job of presenting the simulation results and do you have any alternative suggestions for how to do this?

i) Table 1 is good for visualizing the results but it can be improved. Table 1 could have been ordered by significance and tests (unadjusted or not) such that the comparison between both tests can be easily seen by the reader, without needing to look at a second table below and then look again at the first sub-table on the top (and correct row) for being able to compare the values obtained.

Example:

		Sample size								
Nominal level	**Model**	**50**	**75**	**100**	**150**	**200**	**300**	**400**	**500**	**1000**
0.01	A	0.009	0.014	0.013	0.010	0.012	0.012	0.015	0.015	0.019
	B	0.005	0.009	0.006	0.008	0.008	0.009	0.008	0.010	0.016
0.05	A	0.078	0.076	0.072	0.075	0.061	0.070	0.067	0.069	0.061
	B	0.057	0.054	0.049	0.047	0.045	0.050	0.052	0.051	0.053

Table 1: New Table

ii) In this case, the comparison between the non-adjusted and adjusted tests is very easy to perform since the results are located side-by-side (as we suggested in the previous table). For visualization purposes (just a minor change), the mixing proportion value could be placed in the middle of the "Sample sizes" values, in the same row as 100 in this case. It would be more explicit to indicate that each batch of 3 sample sizes corresponds to a particular mixing proportion.

Important is to note that a reader who did not read all the previous content of the paper will not be able to understand the meaning of $D = I$, $I \in \{1, 2, 3\}$ since there is no explanation/caption associated with the table that adds information about these elements.

iii) The third Table is also good for presenting the results but its first column can be easily eliminated since there is only one value inside of it (Mixing proportion equal to 0.7) and thus, that information can be taken out from the Table and add it inside the caption or table title, in order to have a better looking (and clean) Table with the relevant results, allowing the authors to add more decimals if they want or simply have a better looking Table.

On the other hand, we can suppose that some readers are not reading 100% of the contents but they are interested in the results. In that case, a better caption/explanation should be useful to be with the Table in order to be able to gather the maximum information from them. In this particular case, we notice that a reader may not easily understand that $2LR^*$ corresponds to the adjusted test (there is no legend/caption indicating that so we have to guess based on the previous sections) and the meaning of D_i is not clear just looking at the Table. Therefore, it is not self-content.

As with Table 1 and 2, the numeric format (decimals separator) is not the best one in our opinion because the reader is used to dots/commas as separators instead of a middle position dot. However, it can be a matter of journal format and in that case, authors needed to follow that special format.

iv) Table 4 can be misleading due to the fact that two mixing proportions are being used but there is no column indicating the proportion values as before. In addition, too many results are included in one table, it may be good (depending on the journal and the expected audience of the article) to split the results into two separated tables with a small discussion in between them.

d) Interpret their tables on power (Tables 2 and 4) - do the results make sense in terms of how the power varies as a function of the data generating mechanism?

Remembering that the power of a binary hypothesis test consists of the probability that the test is rejecting the null hypothesis when the alternative one is true, we are interested in how the power of the article's test is affected by the data generating mechanism and configurations tested. In addition, a simple power analysis can be performed in order to determine the minimum sample size that is needed in order to be able to reasonably detect a certain effect of a given size.

Classic/General factors influencing the power of a test are:

a) The significance level (alpha) where we can usually increase the power level of the test by selecting a larger significance level (e.g. 0.05 instead of 0.01). This will increase the chances of rejecting the null hypothesis when it is false (reduces Type II error) but at the same time, it increases the risk of obtaining a "significant result" when the null hypothesis is not false (Type I error).

b) Sample size: effects are difficult to measure in smaller samples. Usually, larger sample sizes will increase the power of the test.

Other specific factors will influence the power of the statistical test depending on its particular characteristics.

Thus, based on the results from Tables 2 and 4, and all the previous development of the article about the characteristics of the test including its mathematical expression (adjusted and not adjusted), we have that:

1. **Table 2: single normal versus a two-component normal mixture**

   In this case, we can see that the power of the test is not the best (very low) when the size of the sample is small (n less than 200) and the distance between the distributions is not very significant in order to be able to clearly identify the underlying distribution. These results are consistent with the authors (and ours) expectations since based on the mathematical expression of the test we can clearly see that it will lose power depending on the data generating mechanisms. Here, having sample sizes less than 200 lead to a poor approximation of the test to the limiting distribution while values of $D = 1$ or 2 do not help the test to distinguish between distributions: therefore, these two elements drive the test to poor results.

   On the other hand, we can see that increasing the value of D (up to 3) will increase the power of the test as expected since it is easier to distinguish the components of one distribution from another. Hence, we can see from the paper that when $D = 3$ the minimum sample size needed for obtaining reasonable results decreases up to $n = 100$. Thus, there exists a clear and expected relationship between the possible configurations of the test and its outcome.

   One interesting result is obtained regarding the effect of the mixing proportion where the authors indicate that "there is no strong evidence that the power depends on the mixing proportion", a result that does not follow one of the studies indicated during the introduction of the paper where we have that "Goffinet et al. (1992) considered the case where the mixing proportions of a mixture distribution are known a priori. Their results suggest that the asymptotic distribution of the likelihood ratio test statistic depends on the value of the mixing proportion, the dimensionality of the problem and whether or not the within component variance is known".

   Finally, the conclusions derived from the unadjusted test where the authors indicate that its power is "inflated" because the approximation of both nominal levels (0.01 and 0.05) is not accurate to the actual levels of the test were expected based on the mathematical expression of the test.

   Hence, there exists a clear relationship between the data generating mechanism (and configurations) and the power of the test where, as expected, a larger size of the sample will increase the power of the test as well as the distance between the distributions (easier to identify them).

2. **Table 4: two-component normal mixture versus a three-component normal mixture**

   As with the previous results, in this case, the authors obtain similar results in terms of the effects that both the sample size and distance between the distribution components have in the power of the test. Like before, we have that the most important parameter in terms of the impact on the power of the test is the value of D, the spacing between the parameters of the distributions.

   Again, small values of $D1$ and $D2$ require larger sample sizes (n at least 200) in order to reach a good power level (and convergence of the test) while values of $D1$ and $D2$ greater or equal to 3 will need a smaller sample size (at least 100) in order to perform well in terms of p-values.

   In addition, similar results are obtained – quantitatively similar results – for the different weight sets tested, and thus, results are consistent with the ones obtained in the second table, following the same pattern where mixing proportions were not clearly significant in terms of their impact on the power of the test.

   Therefore, we can conclude that the results presented in the paper are consistent with our (and the authors) expectations and are coherent to the statistical theory regarding the power of a test, based on the data generating mechanism and configuration employed.

e) How do you think the authors decided to use 1000 simulations? Would 10 simulations be enough? How might we decide if 1000 simulations is enough?

When we perform simulations, it is very important to be aware of the fact that we are developing a sampling from a certain population/distribution and that we are not going to obtain the "real value" of a certain parameter. Thus, instead of obtaining a unique value, we usually obtain a confidence interval for the variable/parameter we are simulating with its corresponding significance level (usually 95% or 99%). Computing the confidence interval is associated with some assumptions regarding the underlying distribution of the samples where the most commons are the normal (sigma known) and t-student (unknown standard deviation) distributions. Depending on the sampling size, the level of uncertainty (width of the confidence interval) associated with the estimated value will be different and hence, the posterior analysis using the estimated values can include a larger error margin.

Therefore, we need to decide the number of replications taking into account that the total sample size will directly affect the confidence interval value and thus, our certainty that the values obtained are a "real" (at least as good as possible) representation of the original population/distribution.

In the particular setting of the paper, the authors could have decided the number of simulations (1000) following a similar logic to the one exposed above: since they are investigating the finite sample properties of the proposed test, they should be certain that the values obtained are good representations of the underlying distributions and thus, a large number of replications such as 1000 is selected in order to take into account that the random sampling (maybe using different random seeds) is associated with a small confidence interval for each sample size ($n = 50, 75, \ldots, 1000$).

Clearly, assuming that we do not know anything from the distribution/population, 10 replications seems a very small amount of replications in order to fulfill the described requirements. The sample can be completely biased due to the lack of more replications, inducing to completely different (and wrong) results and conclusions to the whole study. There are high chances that with this small amount of replications, the estimated values are not good representatives of the underlying distribution leading to wide confidence intervals for the relevant parameters/variables estimated, with the corresponding error factor that the study will "carry on" for each step/action performed after this step propagating the error to the rest of the paper results. Therefore, we might decide the number of replications (1000 or other value) taking into account the previous analysis regarding the potential error associated with the estimation. A simple way consists of looking at the confidence intervals obtained for each estimated parameter/variable and try to reduce them as much as we want within a certain threshold in order to balance the trade-off between accuracy and computational time (or money) needed to perform all the replications.

Problem 2: SQL and R

In this problem, we want to determine the number of users (distinct users) that have asked R-related questions but no Python-related questions. The approach is very simple: we will load the dataset using the *RSQLite* package in R in order to be able to perform classic SQL queries from within R. Since we have worked a lot in the past with SQL, it is clear that we can obtain the desired information by just one query, however, we will use a step-by-step approach for simplicity (easy to follow the logic) and visualization purposes.

1. After downloading the StackOverflow dataset, we load the *RSQLite* library and we initialize the connection with it:

```
# Loading libraries
library(RSQLite)

# Load the db driver
drv <- dbDriver("SQLite")

# DB location directory
dir <- 'C:/Users/chile/Desktop/Stats243/HW/HW6/P2'

# DB filename
dbFilename <- 'stackoverflow-2016.db'

# Establish the connection
db <- dbConnect(drv, dbname = file.path(dir, dbFilename))
```

2. In order to be able to perform the queries, we take a look at the dataset tables and the relevant fields from the tables that we are going to use in our queries.

```
# Check the existing tables
dbListTables(db)
```

```
## [1] "answers"          "maxRepByQuestion" "questions"
## [4] "questionsAugment" "questions_tags"   "users"
```

```
# Testing query: everything from questions (5 rows)
dbGetQuery(db, "select * from questions limit 5")
```

```
##   questionid        creationdate score viewcount
## 1   34552550 2016-01-01 00:00:03     0       108
## 2   34552551 2016-01-01 00:00:07     1       151
## 3   34552552 2016-01-01 00:00:39     2      1942
## 4   34552554 2016-01-01 00:00:50     0       153
## 5   34552555 2016-01-01 00:00:51    -1        54
##                                                                                title
## 1                                                                  Scope between methods
## 2      Rails - Unknown Attribute - Unable to add a new field to a form on create/update
## 3 Selenium Firefox webdriver won't load a blank page after changing Firefox preferences
## 4                                                        Android Studio styles.xml Error
## 5                          Java: reference to non-finial local variables inside a thread
##   ownerid
```

```
## 1 5684416
## 2 2457617
## 3 5732525
## 4 5735112
## 5 4646288
```

```
# Check the fields of questions, users, and questions_tags
dbListFields(db, "questions")
```

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

```
dbListFields(db, "users")
```

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

```
dbListFields(db, "questions_tags")
```

```
## [1] "questionid" "tag"
```

3. For checking purposes, we perform two general queries in order to identify the users with R-related questions and the users with Python-related questions. Thanks to these queries, we will be able to know if the sintax is correct in order to later use them as steps for obtaining the final table we want. In this case, we generate two tables: one with the distinct users with R-related questions (and potentially Python-related questions) and another one with users asking Python-related questions (and potentially R-related questions). Clearly, we want to obtain the "difference" between the first table and the second one, eliminating those users who are also associated with Python-related questions.

```
# Samples: Checking queries
# Users with R questions
dbGetQuery(db, "select distinct users.userid from users, questions,
           questions_tags where users.userid = questions.ownerid
           and questions.questionid = questions_tags.questionid
           and questions_tags.tag = 'r' limit 5")
```

```
##    userid
## 1  575952
## 2 5492392
## 3 5738949
## 4 4802680
## 5 3507767
```

```
# Users with python questions
dbGetQuery(db, "select distinct users.userid from users, questions,
           questions_tags where users.userid = questions.ownerid
           and questions.questionid = questions_tags.questionid
           and questions_tags.tag = 'python' limit 5")
```

```
##    userid
## 1  845642
## 2 4458602
## 3 2927983
## 4 5736692
## 5 5636400
```

Based on the outputs, we can see that the queries are working and thus, we are ready to perform the main operation.

4. We create two temporal views: $UsersR$ and $UsersPython$ containing the previous - and tested - tables.

```
# Create temporal tables
# Users asking R
dbGetQuery(db, "create view UsersR as select distinct users.userid from
           users, questions, questions_tags where users.userid = questions.ownerid
           and questions.questionid = questions_tags.questionid and
           questions_tags.tag = 'r'")
```

```
## data frame with 0 columns and 0 rows
```

```
# Users asking Python
dbGetQuery(db, "create view UsersPython as select distinct users.userid from
           users, questions, questions_tags where users.userid = questions.ownerid
           and questions.questionid = questions_tags.questionid
           and questions_tags.tag = 'python'")
```

```
## data frame with 0 columns and 0 rows
```

5. Using these two views, we obtain the final table by just computing the table that contains the users asking R-related questions ($UsersR$ view) that do not have a match on the second table ($UsersPython$). This means that the final table will only contain users asking R-related questions who are not asking Python-related questions.

```
# Final query
RnoPythonFinal = dbGetQuery(db, "select distinct UsersR.* from UsersR
                            left join UsersPython on (UsersR.userid = UsersPython.userid)
                            where UsersPython.userid IS NULL")
length(RnoPythonFinal$userid)
```

```
## [1] 18611
```

Hence, we obtain the desired information. Notice that we can count the number of entries without using the length function, by just executing the following query, obtaining the same result as before.:

```
# Final query count
RnoPythonFinal2 = dbGetQuery(db, "select count(distinct UsersR.userid) from UsersR
                            left join UsersPython on (UsersR.userid = UsersPython.userid)
                            where UsersPython.userid IS NULL")
RnoPythonFinal2
```

```
##   count(distinct UsersR.userid)
## 1                         18611
```

For completeness, we clean/drop the created views:

```
dbGetQuery(db, "DROP VIEW UsersR")
```

```
## data frame with 0 columns and 0 rows
```

```
dbGetQuery(db, "DROP VIEW UsersPython")
```

```
## data frame with 0 columns and 0 rows
```

Problem 3: Wikipedia and time series analysis

The general strategy for this problem is as follows:

a) Based on the example developed in class about Spark and the code provided in Unit 8, we will use the same code structure.

b) Different expressions/names/concepts will be used in order to answer several questions from the Wikipedia data.

c) Once the results are obtained in Spark (one file containing the hits per date and language), the content will be downloaded and locally processed using a script developed in R in order to plot and make comparisons between different languages (zones) and the number of hits.

Thus, we will perform a series of simple analysis from the Wikipedia data and we will try to briefly explain/find the source of some interesting patterns that we will find when performing the filtering for the selected concepts/names we want to study. Notice that since we will perform several "queries" to the Wikipedia dataset, a series of Python files (.py files) will be uploaded to the SAVIO cluster in order to simply execute them inside the interactive mode when the *srun* command is provided.

i) Spark: Python code

The Python scripts are completely based on the code provided in Unit 8 (and during class) for the "Barack_Obama" example. Hence, the code starts with the reference to the dataset directory and performs some initial checks.

```
# Raw data directory
dir = '/global/scratch/paciorek/wikistats_full'

# Read data and do some checks
lines = sc.textFile(dir + '/' + 'dated')
lines.getNumPartitions()

# Check the number of lines
lines.count()

# Testing lines
testLines = lines.take(10)
testLines[0]
testLines[9]
```

A simple modification of the code provided in Unit 8 is used as the main processing script. Keeping the same structure and functions, we simply modify the *regexp* provided to the find function (in this example, we are interested in the number of hits related to *Chile*).

```
# Filtering and processing functions
# Importations
import re
from operator import add

# Find the relevant expression inside the partitions
def find(line, regex = "Chile", language = None):
    vals = line.split(' ')
```

```
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[3])
    if tmp is None or (language != None and vals[2] != language):
        return(False)
    else:
        return(True)

# Filter test and repartitioning
lines.filter(find).take(100)
chile = lines.filter(find).repartition(480) # 18 minutes for full

# Count the number of hits
chile.count()
```

Following the original code, we perform a stratification of the data containing the relevant expression by performing a reduction-by-key operation:

```
# Map-reduce step: Sums hits across date-time-language triplets
def stratify(line):
    # create key-value pairs where:
    # key = date-time-language
    # value = number of website hits
    vals = line.split(' ')
    return(vals[0] + '-' + vals[1] + '-' + vals[2], int(vals[4]))

# Sum the number of hits for each date-time-language value
counts = chile.map(stratify).reduceByKey(add)
```

Then, a transformation function is defined in order to split the main key into separate fields for the final file:

```
# Map step to prepare output
def transform(vals):
  # Split the key info back into separate fields
    key = vals[0].split('-')
    return(",".join((key[0], key[1], key[2], str(vals[1]))))
```

Finally, we simply redirect the output of the "query" to our local directory inside the cluster (associated with the relevant expression under study) in order to be able to get access and download it for further processing:

```
# Output results to a single file inside our home directory
outputDir = '/global/home/users/cpaismz/ChileFull'
counts.map(transform).repartition(1).saveAsTextFile(outputDir)
```

Hence, the Python code is identical to the one provided inside the Unit 8 with the slight difference of using different expressions for the filtering process. As we will see in the next sections, we will perform a series of queries for different expressions in order to analyze interesting situations and the reaction to them based on the zone (language) of the world.

ii) Spark: upload files and connection

Once the Python files (one per expression to be studied) are ready, we upload them from our local working directory to our user directory inside the SAVIO cluster. In order to perform this operation, we simply use

the UNIX module installed in our Windows machine and invoke the *scp* command that allows us to upload any file to a certain folder inside our SAVIO folders. Clearly, the user will need to input his/her password if needed by the cluster.

```
# Working directory
cd ~/ps6/Python/

# Copy Python files to ~/Python/ directory inside Savio (general expression)
scp FILE_TO_UPLOAD cpaismz@dtn.brc.berkeley.edu:~/Python/

# Specific example
scp ChileFull.py cpaismz@dtn.brc.berkeley.edu:~/Python/
```

After uploading the files, we connect to the cluster via *ssh* protocol and we initialize Spark by following the instructions provided in Unit 8.

```
# Connect via ssh (Unix in windows)
ssh cpaismz@hpc.brc.berkeley.edu

# Initialize Spark  (4 nodes)
srun -A ic_stat243 -p savio2 --nodes=4 -t 1:00:00 --pty bash
module load java spark
source /global/home/groups/allhands/bin/spark_helper.sh
spark-start

# Check the environmental variables
env | grep SPARK

# PySpark using Python 2.6.6 (default Python on Savio)
module unload python
pyspark --master $SPARK_URL --executor-memory 60G
```

iii) Spark: Run the Python code

When Spark is loaded (PySpark), we can simply call our Python scripts containing all the relevant functions and commands in order to perform the relevant queries to the Wikipedia dataset. In this case, we are performing six different queries in order to answer six simple questions:

i) **Obama extension**: Just for testing and visualization purposes, a simple extension where not only the hits performed by users using English as their main language but also in Spanish and French is performed. The idea is to see if the election of the former president Obama was important not only to English speaker users but relevant in other places around the world such as Latin America, France, and potentially Africa.

ii) **Chile**: In order to check the "popularity" of our country during the period covered by the dataset, we perform a query using "Chile" as the regular expression to match. Chile has been increasing its touristic attractive during the last years (it is currently the top one place to visit recommendation in Lonely Planet) and thus, we are interested in checking if people from different places around the world (English, Spanish, and French languages) were looking for Chile as a holiday destination during the period under study. As we will see, an interesting pattern arises from the French zone analysis.

iii) **O.J. Simpson**: 2008 was not the best year of O.J. Simpson life because *he was convicted and sentenced to 33 years imprisonment with a minimum of nine years without parole* in October. Therefore, we are

interested in the interest gathered by this new in people from the already mentioned three languages zones. The idea is to understand that in some places like in South America (mainly Spanish zone) we are not as interested as in U.S.A. or France in sports like American Football (or similar ones like Rugby) and thus, the impact of this new would be completely different depending on the world zone.

iv) **Christmas**: Following a similar logic to the previous question, we want to check the "popularity" of Christmas (as a concept) across different zones. As we know, Christmas is always depicted as a family celebration with all people gathered together inside a house with a chimney while snow is falling outside: a close moment with your relatives/friends. However, in South America Christmas is experienced in a different way: it is summer and people are usually taking holidays, traveling, or the activities are completely different in comparison to the northern hemisphere. Therefore, we want to compare the number of hits looking for "Christmas" (as a very simple analysis) for different world zones.

v) The period between 2007-2008 was highlighted by the well known global financial crisis. In particular, the 2008 crisis is considered (as mentioned in Wikipedia) *by many economists to have been the worst financial crisis since the Great Depression of the 1930s*. Remembering that period, the crisis started in 2007 with the sub-prime mortgage market in the US and it started to be an international banking crisis when the investment bank Lehman Brothers collapsed on September 15, 2008. Therefore, we want to check the number of hits looking for "recession" from different parts of the world.

vi) Finally, a simple question regarding the number of hits performed looking for "Nintendo" is performed. The idea is to visualize potential patterns regarding the popularity of Nintendo towards the Christmas period when the consoles and entertainment system usually are very demanded as gifts.

```
# Execute the python scripts
execfile( "/global/home/users/cpaismz/Python/ObamaFull.py")
execfile( "/global/home/users/cpaismz/Python/ChileFull.py")
execfile( "/global/home/users/cpaismz/Python/OjSimpsonFull.py")
execfile( "/global/home/users/cpaismz/Python/Christmas.py")
execfile( "/global/home/users/cpaismz/Python/RecessionFull.py")
execfile( "/global/home/users/cpaismz/Python/Nintendo.py")
```

iv) Spark: download results

Once the results are recorded in our local directory inside the cluster, we will download them to our local machine in order to perform the visual analysis. Since we are using R in Windows, we perform a copy of the resulting file from our UNIX directory to our Windows working directory for simplicity. Clearly, we perform this operation for all the files (Chile analysis in the example).

```
# Copy results to local folder
scp cpaismz@dtn.brc.berkeley.edu:~/ChileFull/* ~/ps6/downloads

# Rename the resulting file and create a copy in the Windows working directory
cd ~/ps6/downloads/
mv part-00000 chilefull
cp chilefull /mnt/c/Users/chile/Desktop/Stats243/HW/HW6/
```

iv) R logic and visual processing

In order to analyze the resulting files, we implement three different functions that will allow us to plot the time series data obtained from the SAVIO cluster.

1. The first function *ToPlotF*() takes the name of the file as the main input and the Title that we want to display in the plot as a secondary argument. In addition, the user can define the color of the graph (the lines and/or points color) by including an extra third optional argument (red by default). The function processes the filtered data from Wikipedia performing a series of transformations (as the ones included in the example of Barack Obama in class) of each field inside the generated DataFrame such defining the date and time formats plus the creation of an extra *chron* field mixing dates and times.

```
# Loading libraries
library(dplyr)
library(ggplot2)
library(chron)
library(scales)

# First function: English, Spanish, French, and Chinese DataFrames plus EN plot
ToPlotF <- function(filename, Title, color = "#ff3333"){
  # Read the resulting file (processed in Spark)
  dat <- read.csv(filename)

  # Change column names
  names(dat) <- c('date','time','lang','hits')

  # Date and time as characters
  dat$date <- as.character(dat$date)
  dat$time <- as.character(dat$time)

  # Time format
  dat$time[dat$time %in%  c("0", "1")] <- "000000"
  wh <- which(nchar(dat$time) == 5)
  dat$time[wh] <- paste0("0", dat$time[wh])

  # Chron object: Date and time format
  dat$chron <- chron(dat$date, dat$time,
                     format = c(dates = 'ymd', times = "hms"))
```

2. In our particular analysis, we will split the data based on the language. In this case, we separate the number of hits by English, Spanish, French, and Chinese (optional, not covered in our analysis). In addition, we perform a transformation of the *chron* field format by using the special function *as.POSIXlt*() that allows us to easily transform the date to a legible format and the time to the *EST* format without needing any other mathematical operation.

   Then, we order all the filtered DataFrames by the *chron* field (from October to December 2008 and by the hour), obtaining a series of sorted tables easy to analyze and visualize potential trends.

```
  # Filter dataset: English, Spanish, French, and Chinese
  ENdat <- dat %>% filter(dat$lang == 'en')
  SPdat <- dat %>% filter(dat$lang == 'es')
  FRdat <- dat %>% filter(dat$lang == 'fr')
  CHdat <- dat %>% filter(dat$lang == 'zh')

  # Date format
  ENdat$date <- as.Date(ENdat$date, "%Y%m%d")
  SPdat$date <- as.Date(SPdat$date, "%Y%m%d")
  FRdat$date <- as.Date(FRdat$date, "%Y%m%d")
  CHdat$date <- as.Date(CHdat$date, "%Y%m%d")
```

```r
# Time zone format for Date and Hour (GMT to EST)
ENdat$chron <- as.POSIXlt(ENdat$chron, tz = "EST")
SPdat$chron <- as.POSIXlt(SPdat$chron, tz = "EST")
FRdat$chron <- as.POSIXlt(FRdat$chron, tz = "EST")
CHdat$chron <- as.POSIXlt(CHdat$chron, tz = "EST")

# Order by date (for simplicity)
ENdat <- ENdat[order(ENdat$chron),]
SPdat <- SPdat[order(SPdat$chron),]
FRdat <- FRdat[order(FRdat$chron),]
CHdat <- CHdat[order(CHdat$chron),]
```

3. Once the DataFrames are ready, we create a pdf file (open a connection) in order to plot the English dataframe as a first analysis. We include a series of visual enhancements in order to produce a nice looking and easy to follow graph. In this case, we create two plots: a scatter plot where breakpoints consist of weeks and a line plot where breakpoints are the months of the year. The main difference between both plots consists of the fact that the scatter plot groups the data by day (multiple points per day since the hour is not taken into account) while the line plot takes into account the hour, plotting unique observations per x-axis point.

```r
# Create pdf file with plots
pdf(paste(filename,'-traffic.pdf', sep = ""), width = 9, height = 5)

# Scatter plot by weeks, separated by days (not hours)
plotfile <- ggplot(ENdat, aes(x=date,y=hits)) +

  # Visual themes
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  theme(axis.line = element_line(size = 1, colour = "black"),
        panel.grid.major = element_line(colour = "#d3d3d3"),
        panel.grid.minor = element_blank(),
        panel.border = element_blank(), panel.background = element_blank()) +
  theme(legend.position="none") +

  # Add points
  geom_point(colour = color, alpha = 0.5) +

  # Axes scale
  scale_x_date(breaks = date_breaks("weeks"), labels = date_format("%d-%b")) +
  scale_y_continuous(breaks = c(0, 1e5, 2e5, 3e5)) +

  # Labels
  labs(x = "Time [Weeks]", y = "Number of hits",
       title = paste(Title, " frequency vs Time [weeks]", sep = ''))

# Print the plot
print(plotfile)

# Close the connection with pdf file
dev.off()
```

4. As with the scatter plot, the line plot is generated in a second pdf following the same logic structure in terms of code.

```
  # Line plot: X-axis separated by hour
  pdf(paste(filename,'-traffic-line.pdf', sep = ""), width = 9, height = 5)

  # Create the line plot
  plotfile2 <- ggplot(ENdat, aes(x=chron,y=hits)) +

    # Visual themes
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(), panel.background = element_blank()) +
    theme(legend.position="none") +

    # Plot the line graph
    geom_line(colour = color, alpha = 0.8) +

    # Labels: separated by months
    labs(x = "Time [months]", y = "Number of hits",
         title = paste(Title, " frequency vs Time [months]", sep = ""))

  # Print the graph
  print(plotfile2)

  # Close the connection
  dev.off()
```

5. Finally, we return the ordered DataFrames as the main output from the function for further analysis or manipulation.

```
  # Return dataframes (EN, SP, FR, and CH)
  toReturn <- list(ENdat, SPdat, FRdat, CHdat)
  return (toReturn)
}
```

The second function *ToPlotInd*() allows us to plot a certain DataFrame following the format from the ones created by the previous function, without returning any object (void method).

1. In this case, we have exactly the same logic as before but no processing is performed to the given DataFrame.

```
# Inidividual plots function
ToPlotInd <- function(DF, Title, color = "#ff3333"){

  # Creat Line plot inside a pdf file
  pdf(paste(Title,'-traffic-line.pdf', sep = ""), width = 9, height = 5)
  plotfile2 <- ggplot(DF, aes(x=chron,y=hits)) +

    # Visual themes
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
```

```
          panel.grid.minor = element_blank(),
          panel.border = element_blank(), panel.background = element_blank()) +
    theme(legend.position="none") +

    # Create line plot
    geom_line(colour = color, alpha = 0.8) +

    # Add Labels
    labs(x = "Time [months]", y = "Number of hits",
         title = paste(Title, " frequency vs Time [months]", sep = ""))

  # Print the plot and close the file
  print(plotfile2)
  dev.off()
}
```

1. Finally, we implemented the $ToPlotMult()$ function that allows us to plot three different DataFrames in one pdf file. The idea is to plot the English, Spanish, and French results in only one file for visualization (and sharing) purposes since it will be easier to perform the comparison by just looking one file instead of three separated ones. In this report, we will include the syntax for generating those plots but we are going to include the individual plots (separated) for simplicity and space usage.

   The logic of the function is exactly the same as the previous ones but in this case only one connection is established with a pdf file instead of multiple ones.

```
# Multiple plots in one file function (EN, SP, and FR plots)
ToPlotMult <- function(DF1, DF2, DF3, Title){
  # Line plots: create one pdf file with three line plots
  pdf(paste(Title,'-traffic-lines.pdf', sep = ""), width = 6, height = 3)

  # Plot 1: EN
  plotfile1 <- ggplot(DF1, aes(x=chron,y=hits)) +

    # Visual themes
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(), panel.background = element_blank()) +
    theme(legend.position="none") +

    # Create plot, labels and print the graph
    geom_line(colour = "#ff3333", alpha = 0.8) +
    labs(x = "Time [months]", y = "Number of hits",
         title = paste(Title, " EN frequency vs Time [months]", sep = ""))
  print(plotfile1)

  # Plot 2: SP
  plotfile2 <- ggplot(DF2, aes(x=chron,y=hits,colour= "#0066ff" )) +

    # Visual themes
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
```

```
          panel.grid.minor = element_blank(),
          panel.border = element_blank(), panel.background = element_blank()) +
    theme(legend.position="none") +

    # Create plot, labels and print the graph
    geom_line(colour = "#0066ff", alpha = 0.8) +
    labs(x = "Time [months]", y = "Number of hits",
         title = paste(Title, " SP frequency vs Time [months]", sep = ""))
  print(plotfile2)

  # Plot 3: FR
  plotfile3 <- ggplot(DF3, aes(x=chron,y=hits)) +

    # Visual themes
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    theme(axis.line = element_line(size = 1, colour = "black"),
          panel.grid.major = element_line(colour = "#d3d3d3"),
          panel.grid.minor = element_blank(),
          panel.border = element_blank(), panel.background = element_blank()) +
    theme(legend.position="none") +

    # Create plot, labels and print the graph
    geom_line(colour = "#47d147", alpha = 0.8) +
    labs(x = "Time [months]", y = "Number of hits",
         title = paste(Title, " FR frequency vs Time [months]", sep = ""))
  print(plotfile3)

  # Close connection
  dev.off()
}
```

Now, we can use our previous functions in order to analyze the outputs obtained from the SAVIO cluster.

a) Obama Extension

As mentioned above, we will plot the Barack Obama number of hits per time-based on the filtered Wikipedia dataset. The idea is to observe if his election was a worldwide phenomenon (interest from other languages than English) or just a local one. Clearly, we expect the first option to be the right one since the president elections of the U.S. is usually followed around the world (for good and bad reasons).

Hence, we load the previous defined functions from the .R file containing them $ToPlotFull.R$. Then, plots are generated as pdf files and loaded back into the current report in order to be able to share those files independent of the content of this document. This step can be easily implemented with the basic latex syntax (includegraphics).

```
# Do not output warnings
options(warn = -1)

# Set working directory
setwd('C:/Users/chile/Desktop/Stats243/HW/HW6')

# Load libraries
suppressMessages(library(dplyr))
```

```
library(ggplot2)
library(chron)
library(scales)

# Load functions
source('C:/Users/chile/Desktop/Stats243/HW/HW6/RFunctions/ToPlotFull.R')

## Obama
Results <- ToPlotF('obamafull', "Obama mentions")
OENDat <- Results[[1]]
OSPDat <- Results[[2]]
OFRDat <- Results[[3]]

# Inividual plots
ToPlotInd(OSPDat, "Obama SP", "#0066ff")
```

```
## pdf
##   2
```

```
ToPlotInd(OENDat, "Obama EN")
```

```
## pdf
##   2
```

```
ToPlotInd(OFRDat, "Obama FR", "#47d147")
```

```
## pdf
##   2
```

```
# Multiple plots
ToPlotMult(OENDat, OSPDat, OFRDat, "Obama comparison")
```

```
## pdf
##   2
```

```
# Max value meaning investigation
index <- match(max(OENDat$hits), OENDat$hits)
OENDat$chron[index]
```

```
## [1] "2008-11-05 01:00:00 EST"
```

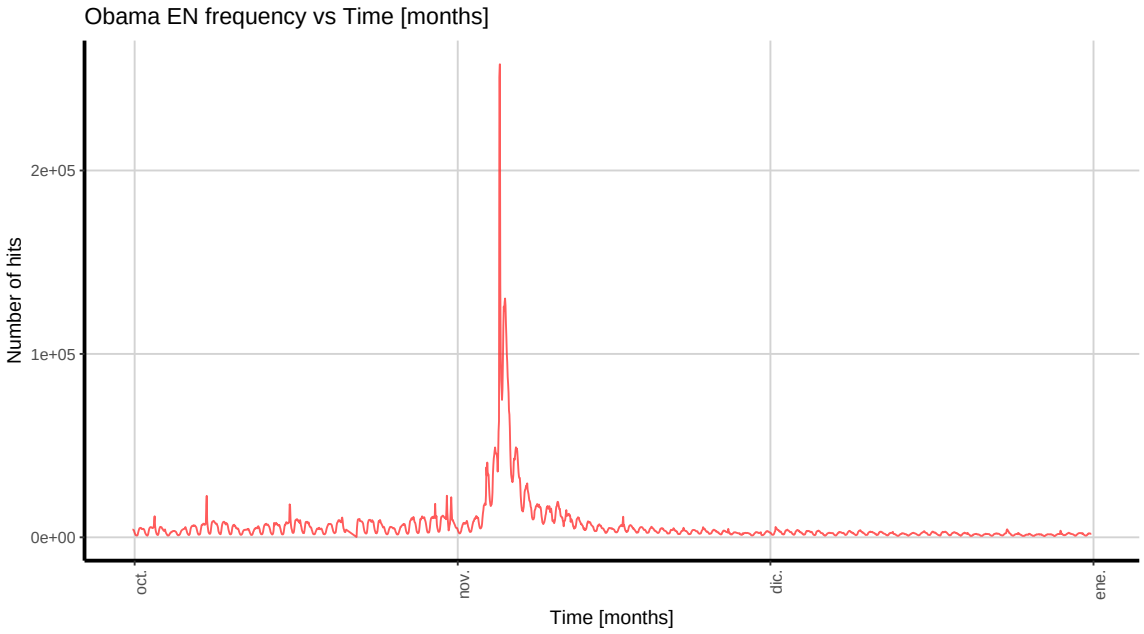

Figure 1: Barack Obama number of hits between 10/08-12/08 English

As we saw in the example in class, the number of hits looking for Barack Obama information from English speakers is very high, having its peak during the 5th of November of 2008: one day after Barack Obama won the elections. Therefore, people were trying to know more about the new president of the U.S. (a shame that it was after the elections...).

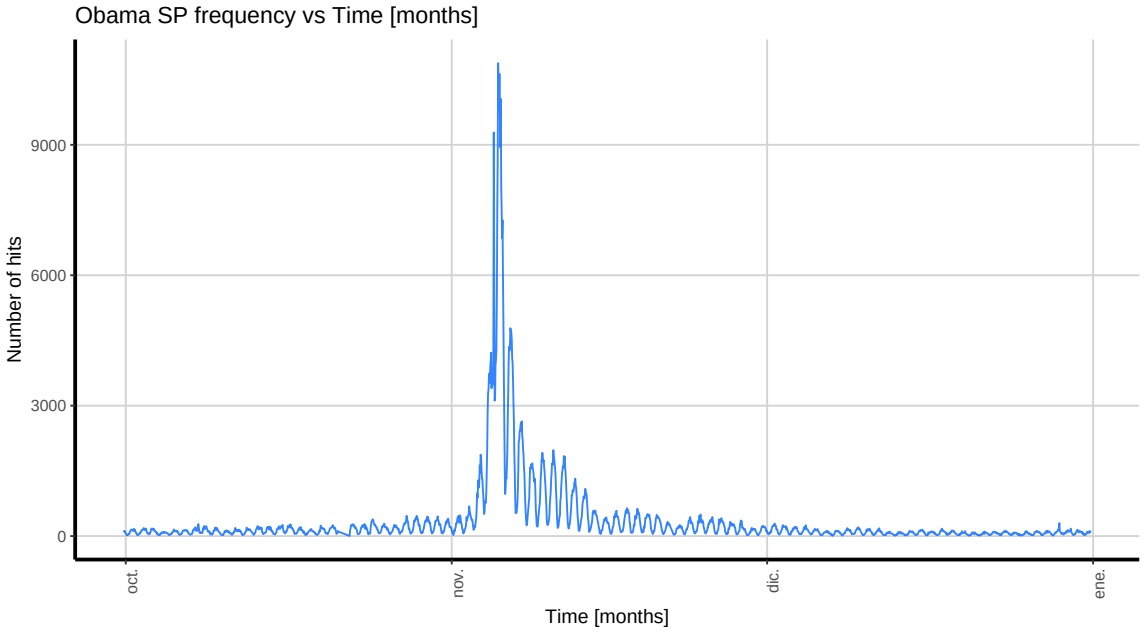

Figure 2: Barack Obama number of hits between 10/08-12/08 Spanish

As expected, the same pattern can be seen when we analyze the Spanish speakers' data. Of course, the magnitude (number of hits) is not as high as with the English speakers data, however, it follows the same pattern.

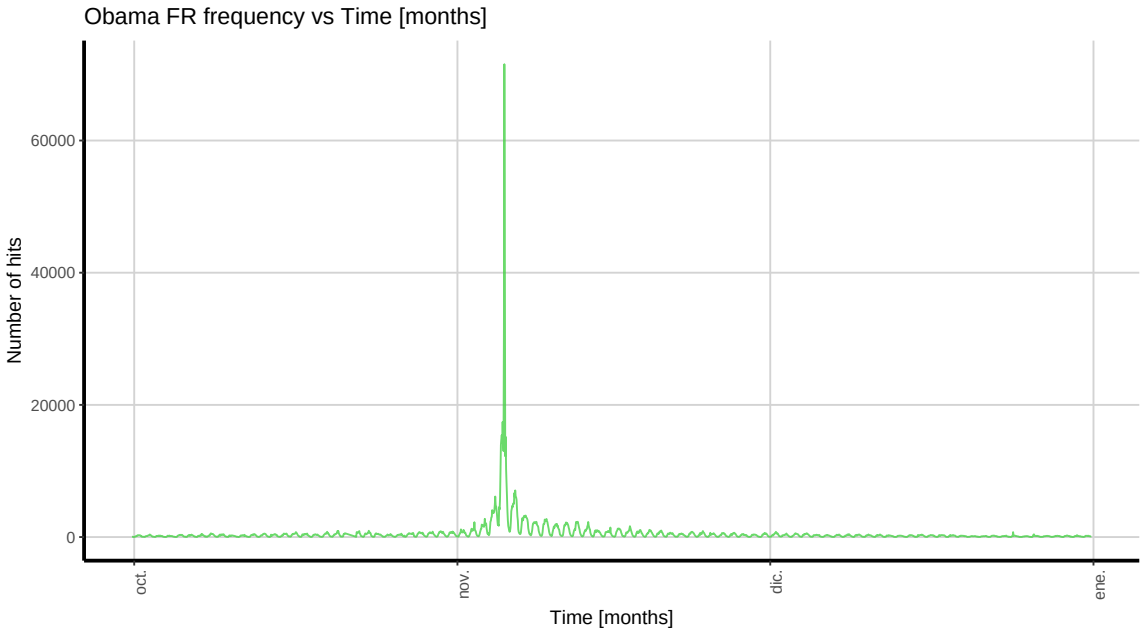

Figure 3: Barack Obama number of hits between 10/08-12/08 French

Again, we can easily check that the election of the U.S. president is a worldwide phenomenon.

b) Chile Mentions

Since the author of this report is Chilean, it is interesting to analyze the number of hits related to this country inside the Wikipedia dataset. As mentioned above, we mainly want to analyze the possible mentions related to the upcoming holidays period.

```
# Do not output warnings
options(warn = -1)

# Set working directory
setwd('C:/Users/chile/Desktop/Stats243/HW/HW6')

# Load libraries
suppressMessages(library(dplyr))
library(ggplot2)
library(chron)
library(scales)

# Load functions
source('C:/Users/chile/Desktop/Stats243/HW/HW6/RFunctions/ToPlotFull.R')

## Chile
Results <- ToPlotF('chilefull', "Chile mentions")
ChENDat <- Results[[1]]
ChSPDat <- Results[[2]]
ChFRDat <- Results[[3]]

# Inividual plots
ToPlotInd(ChSPDat, "Chile SP", "#0066ff")
```

```
## pdf
##   2
```

```
ToPlotInd(ChENDat, "Chile EN")
```

```
## pdf
##   2
```

```
ToPlotInd(ChFRDat, "Chile FR", "#47d147")
```

```
## pdf
##   2
```

```
# Multiple plots
ToPlotMult(ChENDat, ChSPDat, ChFRDat, "Chile comparison")
```

```
## pdf
##   2
```

```r
# Max value meaning investigation
index <- match(max(ChFRDat$hits), ChFRDat$hits)
ChFRDat$chron[index]
```

```
## [1] "2008-12-09 10:00:00 EST"
```

```r
#
# http://www.ohchr.org/EN/HRBodies/UPR/Pages/Highlights9December2008pm.aspx
```

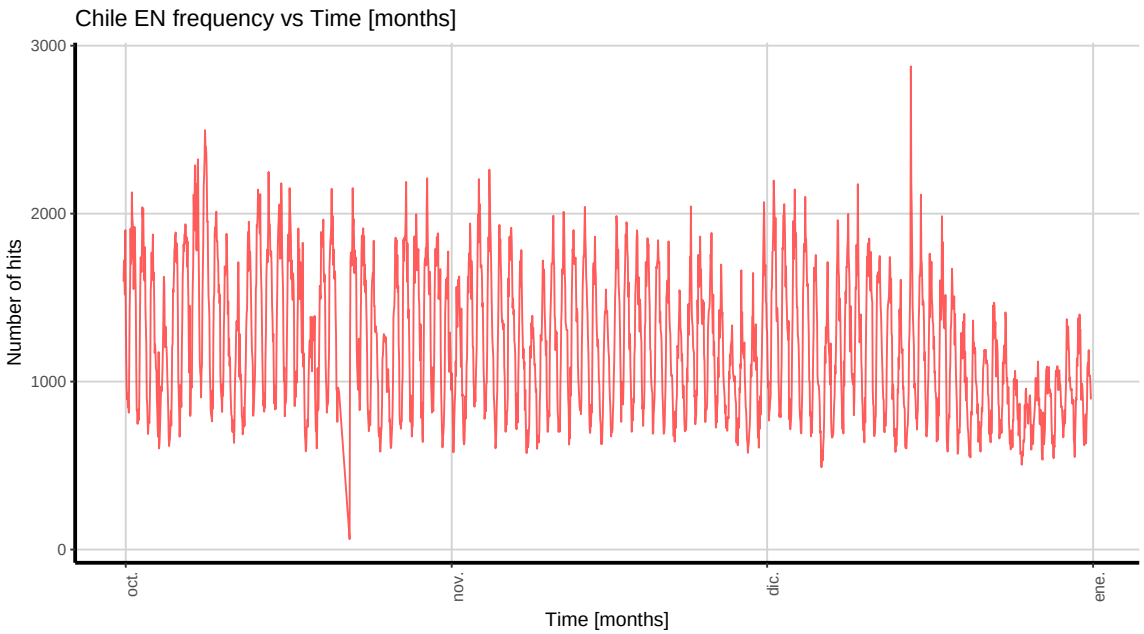

Figure 4: Chile number of hits between 10/08-12/08 English

Based on the results, we can see that the number of hits related to Chile in English does not show a particular pattern. However, there is one peak around the middle of December with no specific explanation.

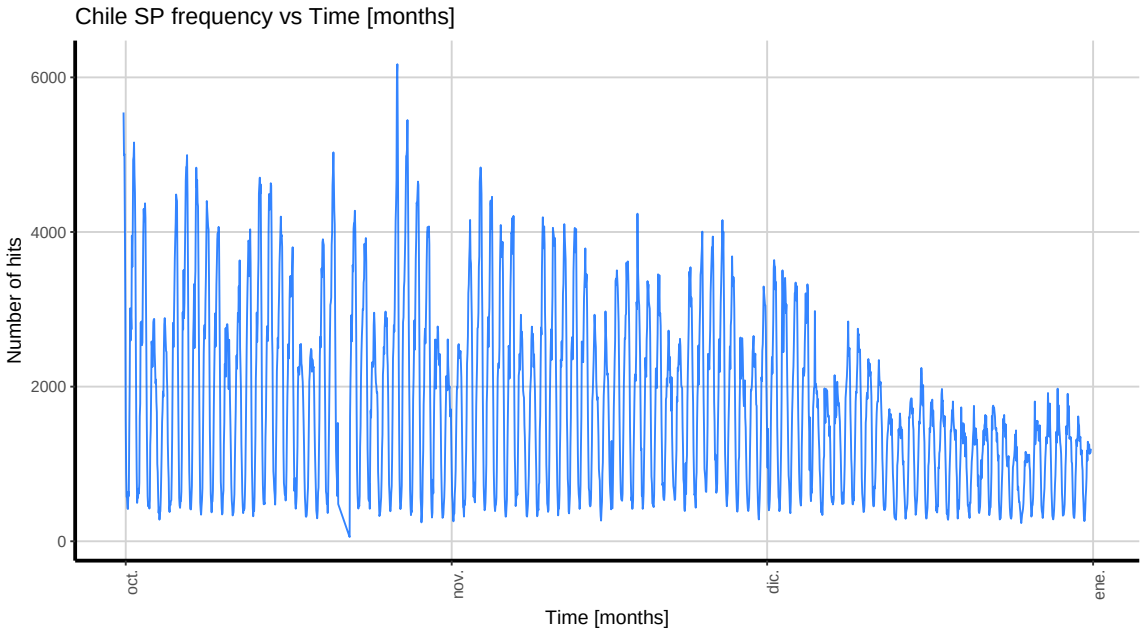

Figure 5: Chile number of hits between 10/08-12/08 Spanish

In this case, we can see that there exists some sort of a decreasing trend with the number of hits looking for Chile information between October to December. One possible explanation of this patter lies in the fact that people are organizing their holidays the previous months before their holidays (usually January and February in South America) and then, December would experience a decrease in the number of hits due to the fact that users have already decided where to go.

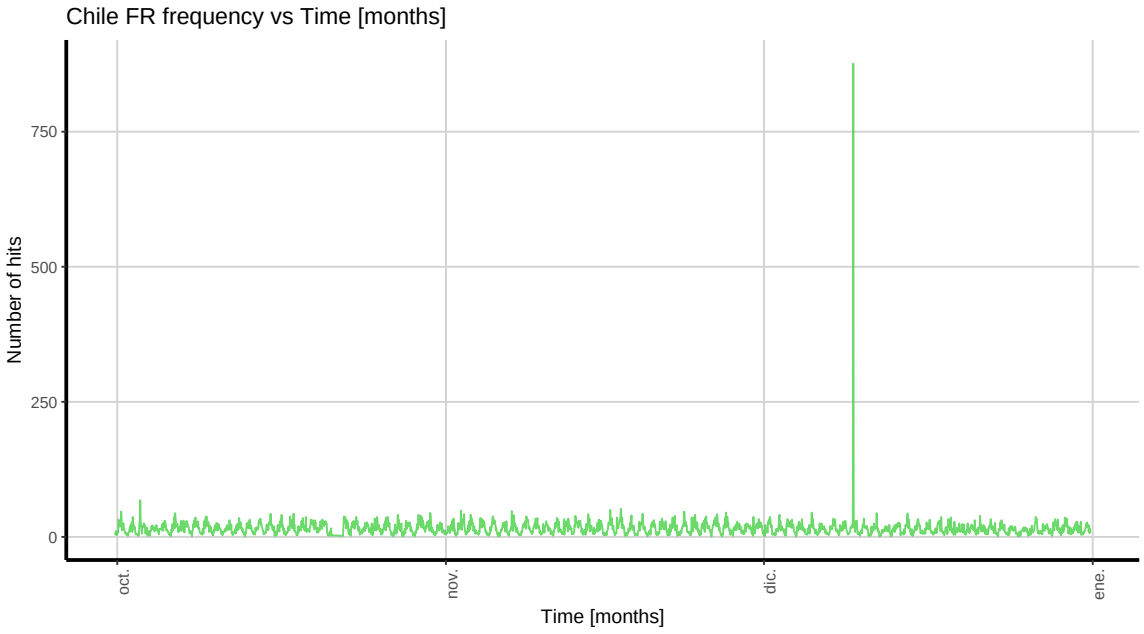

Figure 6: Chile number of hits between 10/08-12/08 French

This is the most interesting pattern found in our analysis. In this case, we can see that French speakers

are not interested in Chile at all during the majority of the time between October and December of 2008. However, there exists an impressive peak on December 9th of 2008. Trying to find the explanation behind this particular pattern (and since the wife of the author of this document is French), we found two main possible explanations:

i) During December 9th of 2008 there was the Universal Periodic Review of the Human Right council where Chile and France were participating [1]. However, there is no strong evidence that this particular event would have triggered the observed pattern.

ii) There was a very popular video in YouTube France [2] regarding some OVNI (UFO in English) observations in Chile. As can be seen from the comments on YouTube, a lot of French speakers were interested in the strange phenomena occurred in Chile and maybe they were curious about the country after watching it (this is the funny explanation, more details in [3]).

[1] http://www.ohchr.org/EN/HRBodies/UPR/Pages/Highlights9December2008pm.aspx
[2] https://www.youtube.com/watch?v=xI_IVvhFaXc Ovni
[3] http://odhtv-archives.kazeo.com/09-12-08-chili-santiago-flotille-d-ovni-a120950822

c) OJ Simpson Trial

Like we mentioned above, we want to study the scope of the O.J. Simpson trial based on the language of the Wikipedia users.

```
# Do not output warnings
options(warn = -1)

# Set working directory
setwd('C:/Users/chile/Desktop/Stats243/HW/HW6')

# Load libraries
suppressMessages(library(dplyr))
library(ggplot2)
library(chron)
library(scales)

# Load functions
source('C:/Users/chile/Desktop/Stats243/HW/HW6/RFunctions/ToPlotFull.R')

## OJSimpson
Results <- ToPlotF('ojsimpsonfull', "O.J. Simpson mentions")
OJENDat <- Results[[1]]
OJSPDat <- Results[[2]]
OJFRDat <- Results[[3]]
OJCHDat <- Results[[4]]

# Inividual plots
ToPlotInd(OJSPDat, "O.J. Simpson SP", "#0066ff")
```

```
## pdf
##   2
```

```
ToPlotInd(OJENDat, "O.J. Simpson EN")
```

```
## pdf
##   2
```

```
ToPlotInd(OJFRDat, "O.J. Simpson FR", "#47d147")
```

```
## pdf
##   2
```

```
ToPlotInd(OJCHDat, "O.J. Simpson ZH", "#8000ff")
```

```
## geom_path: Each group consists of only one observation. Do you need to
## adjust the group aesthetic?

## pdf
##   2
```

```
# Multiple plots
ToPlotMult(OJENDat, OJSPDat, OJFRDat, "O.J. Simpson comparison")
```

```
## pdf
##   2
```

```
# Max value meaning investigation
index <- match(max(OJENDat$hits), OJENDat$hits)
OJENDat$chron[index]
```

```
## [1] "2008-10-04 02:00:00 EST"
```

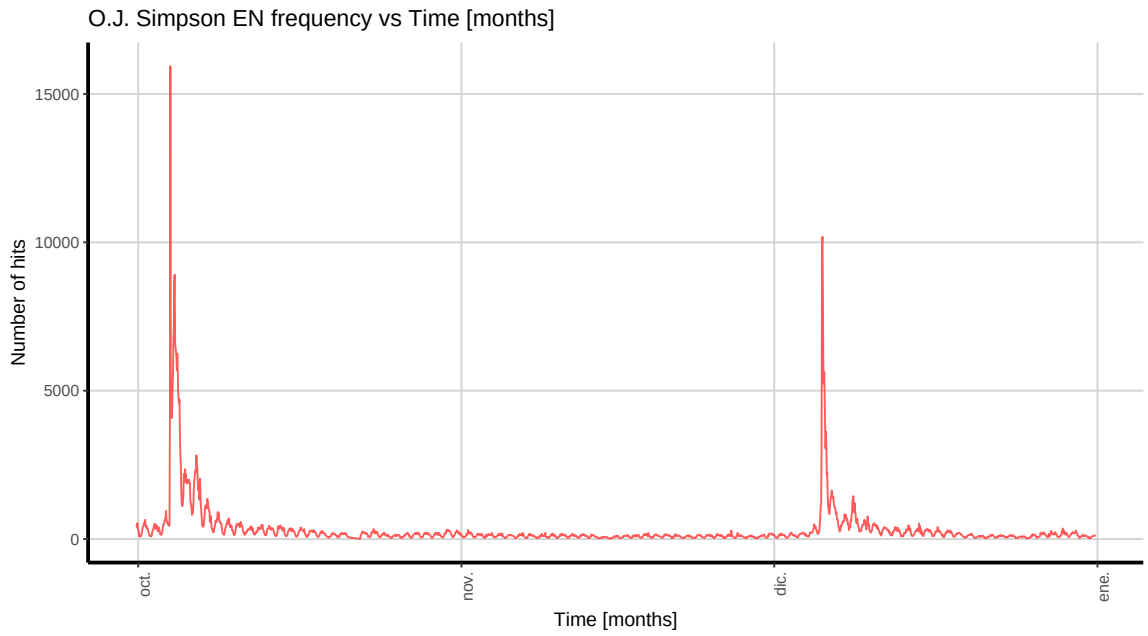

Figure 7: O.J. Simpson number of hits between 10/08-12/08 English

Based on the plot, we can clearly see that it was a very important and covered case for English speaker users. As we know from Wikipedia, *In 2007, Simpson was arrested in Las Vegas, Nevada, and charged with the felonies of armed robbery and kidnapping. In 2008, he was convicted and sentenced to 33 years imprisonment, with a minimum of nine years without parole.* That happened during the 3rd of October 2008 (he was found guilty) and then he was sentenced on December 5th of the same year, corresponding to the two peaks we can identify in the plot.

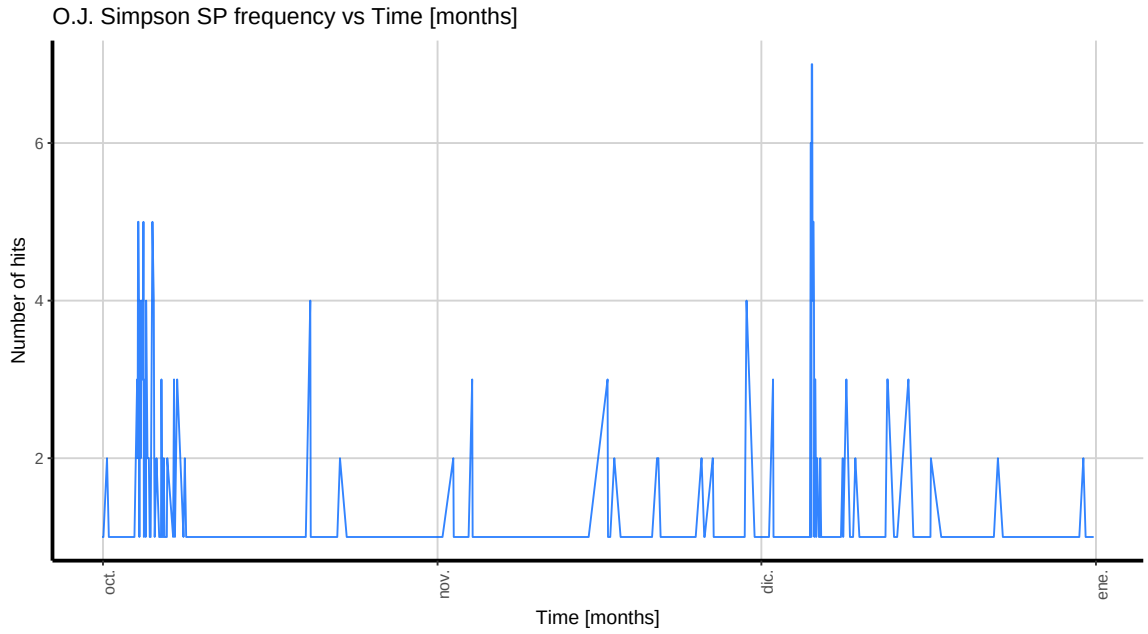

Figure 8: O.J. Simpson number of hits between 10/08-12/08 Spanish

As expected, the case was not followed by a lot of people in Spanish speaker countries due to the lack (possible explanation) of interest in sports such as American football in general (soccer is the most popular sport). Although the number of hits is very low, we can still see that the same pattern occurs regarding the peaks of the hits distribution.

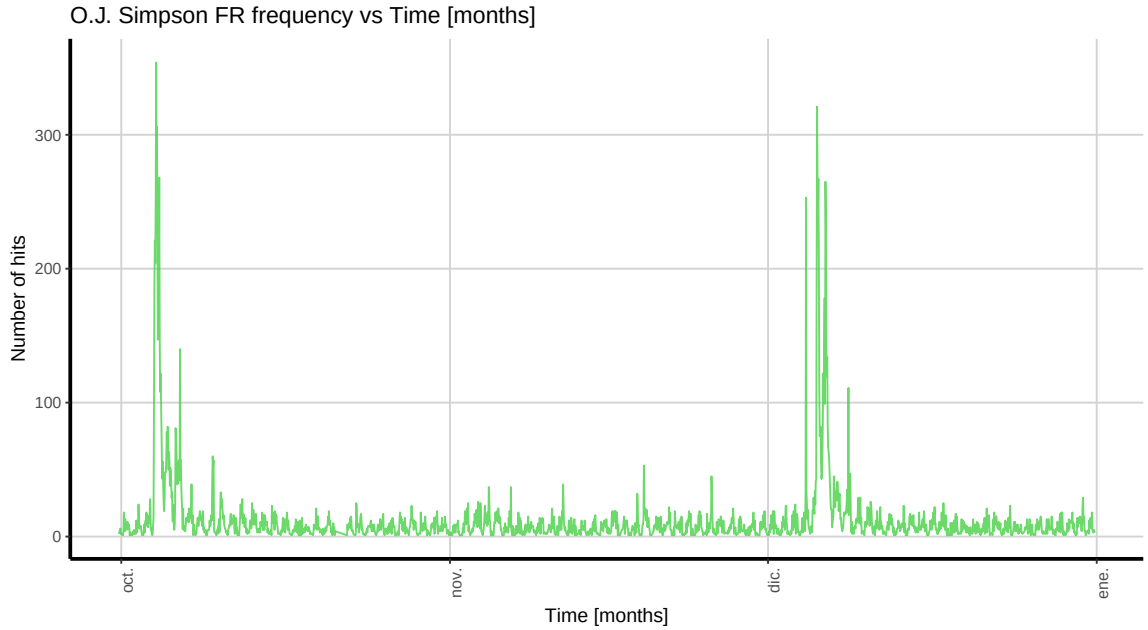

Figure 9: O.J. Simpson number of hits between 10/08-12/08 French

Since French speaker countries tend to follow sports that are closer to the American football (such as Rugby), the O.J. Simpson case was followed with more interest than in the Spanish nations.

d) Christmas

Based on the explanation mentioned at the beginning of this section, we want to study the "popularity" of the Christmas concept in Wikipedia for the period under study.

```
# Do not output warnings
options(warn = -1)

# Set working directory
setwd('C:/Users/chile/Desktop/Stats243/HW/HW6')

# Load libraries
suppressMessages(library(dplyr))
library(ggplot2)
library(chron)
library(scales)

# Load functions
source('C:/Users/chile/Desktop/Stats243/HW/HW6/RFunctions/ToPlotFull.R')

## Christmas
Results <- ToPlotF('christmasfull', "Christmas mentions")
XmasENDat <- Results[[1]]
XmasSPDat <- Results[[2]]
XmasFRDat <- Results[[3]]

# Inividual plots
ToPlotInd(XmasSPDat, "Xmas SP", "#0066ff")
```

```
## pdf
##   2
```

```
ToPlotInd(XmasENDat, "Xmas EN")
```

```
## pdf
##   2
```

```
ToPlotInd(XmasFRDat, "Xmas FR", "#47d147")
```

```
## pdf
##   2
```

```
# Multiple plots
ToPlotMult(XmasENDat, XmasSPDat, XmasFRDat, "Xmas comparison")
```

```
## pdf
##   2
```

```
# Max value meaning investigation
index <- match(max(XmasENDat$hits), XmasENDat$hits)
XmasENDat$chron[index]
```

```
## [1] "2008-12-24 12:00:00 EST"
```

```
index <- match(max(XmasSPDat$hits), XmasSPDat$hits)
XmasSPDat$chron[index]
```

```
## [1] "2008-12-30 20:00:00 EST"
```

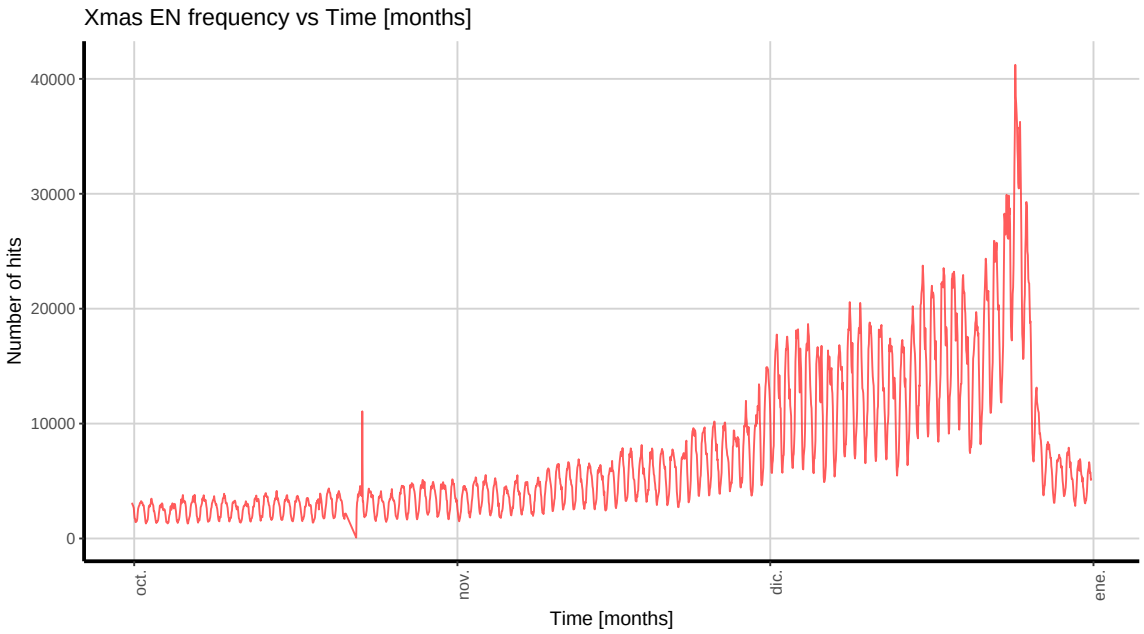

Figure 10: Xmas number of hits between 10/08-12/08 English

As expected, the number of hits looking for Christmas in Wikipedia tend to increase from October to December, reaching its peak on the 24th of December.

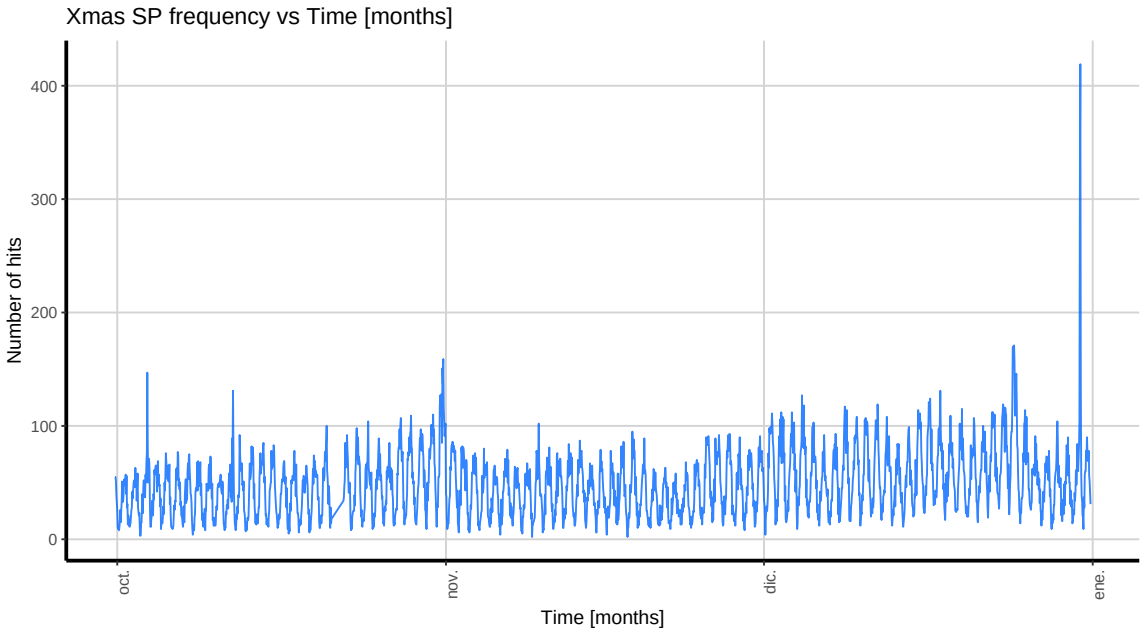

Figure 11: Xmas number of hits between 10/08-12/08 Spanish

Different is the pattern obtained from the Spanish speakers. In this case, there is no explicit trend like in the previous plot and the peak occurs after Christmas, on December 30th. One of the possible explanations behind this pattern can be the fact that in South America Christmas is not "experienced" in exactly the same way as in the northern hemisphere (different arguments were already discussed above).

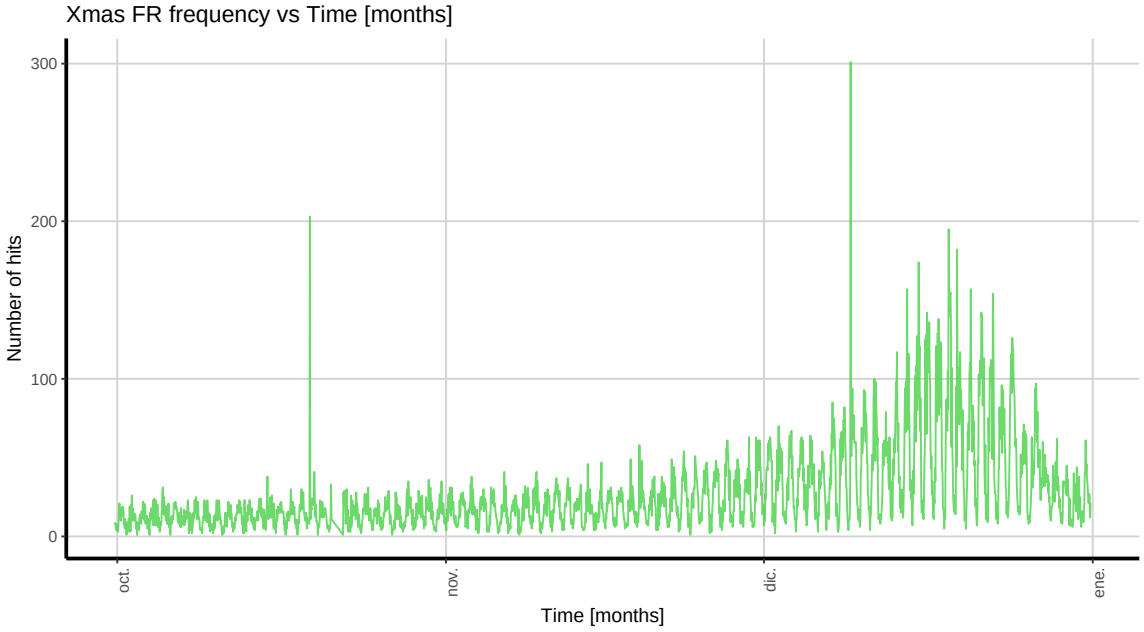

Figure 12: Xmas number of hits between 10/08-12/08 French

Like the first plot, we can easily find an increasing trend in the number of hits associated with the Christmas expression. Thus, a potential relationship with the searches for Christmas and the world hemisphere can exist and inferred.

e) Nintendo

Following the Christmas analysis, we would like to know if there exists an increasing trend (and potential positive correlation) between Nintendo - an entertainment company - and the Christmas season.

```
# Do not output warnings
options(warn = -1)

# Set working directory
setwd('C:/Users/chile/Desktop/Stats243/HW/HW6')

# Load libraries
suppressMessages(library(dplyr))
library(ggplot2)
library(chron)
library(scales)

# Load functions
source('C:/Users/chile/Desktop/Stats243/HW/HW6/RFunctions/ToPlotFull.R')

## Nintendo
Results <- ToPlotF('nintendofull', "Nintendo mentions")
NinENDat <- Results[[1]]
NinSPDat <- Results[[2]]
NinFRDat <- Results[[3]]
NinCHDat <- Results[[4]]

# Inividual plots
ToPlotInd(NinSPDat, "Nintendo SP", "#0066ff")
```

```
## pdf
##   2
```

```
ToPlotInd(NinENDat, "Nintendo EN")
```

```
## pdf
##   2
```

```
ToPlotInd(NinFRDat, "Nintendo FR", "#47d147")
```

```
## pdf
##   2
```

```
ToPlotInd(NinCHDat, "Nintendo ZH", "#8000ff")
```

```
## pdf
##   2
```

```
# Multiple plots (uncomment if wanted)
#ToPlotMult(NinENDat, NinSPDat, NinFRDat, "Nintendo comparison")
```

In contrast to our expectations, we cannot see an explicit trend in the plot for English speakers. However, we can find three main peaks that are associated with the Nintendo E3 presentation in San Francisco.

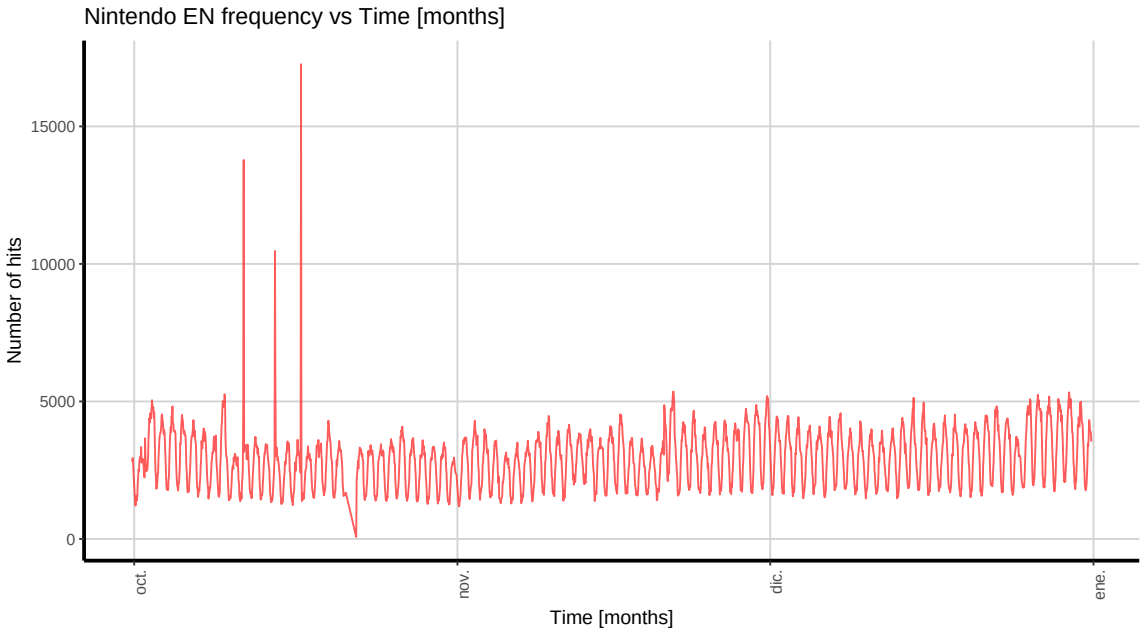

Figure 13: Nintendo number of hits between 10/08-12/08 English

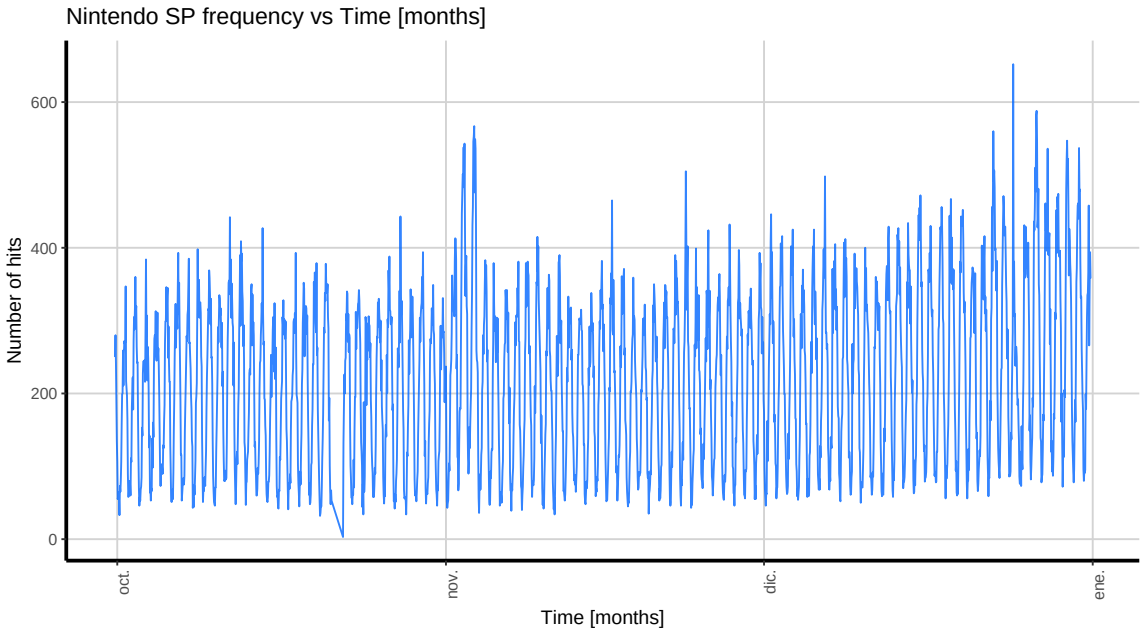

Figure 14: Nintendo number of hits between 10/08-12/08 Spanish

A slight increasing trend can be observed from the data toward the Christmas. However, it is not as strong as expected.

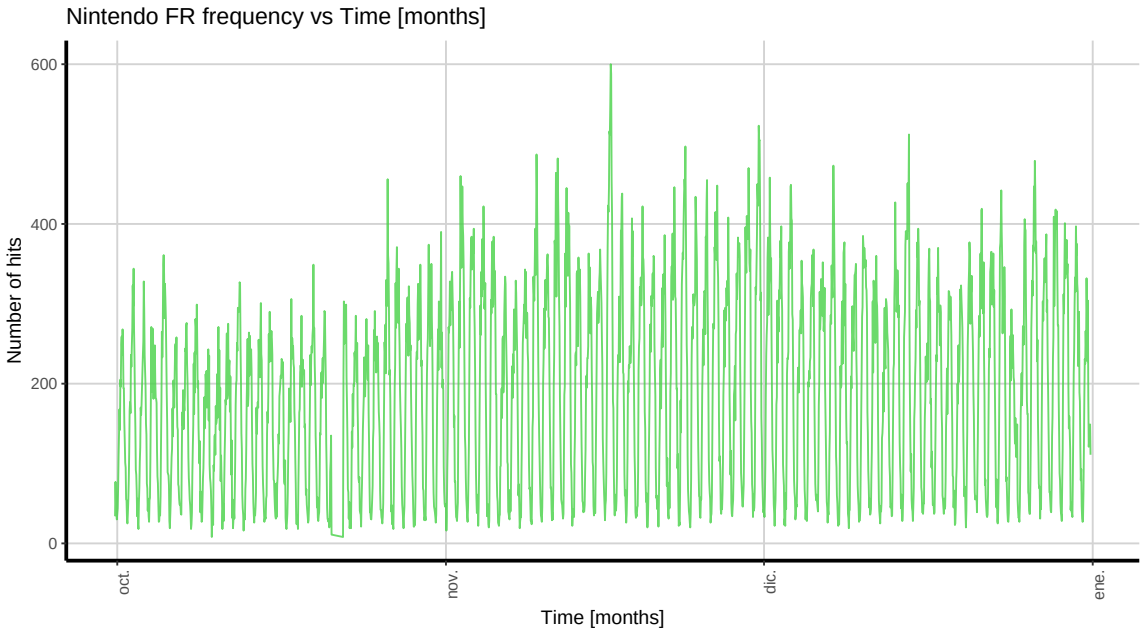

Figure 15: Nintendo number of hits between 10/08-12/08 French

Similar conclusions can be obtained from the French speakers' analysis.

f) Recession

Finally, we want to check the number of hits registered in Wikipedia associated with the financial crisis of 2008 and identify possible patterns depending on the users regions.

```
# Do not output warnings
options(warn = -1)

# Set working directory
setwd('C:/Users/chile/Desktop/Stats243/HW/HW6')

# Load libraries
suppressMessages(library(dplyr))
library(ggplot2)
library(chron)
library(scales)

# Load functions
source('C:/Users/chile/Desktop/Stats243/HW/HW6/RFunctions/ToPlotFull.R')

## Recession
Results <- ToPlotF('recessionfull', "Recession mentions")
RENDat <- Results[[1]]
RSPDat <- Results[[2]]
RFRDat <- Results[[3]]
RCHDat <- Results[[4]]

# Inividual plots
ToPlotInd(RSPDat, "Recession SP", "#0066ff")
```

```
## pdf
##   2
```

```
ToPlotInd(RENDat, "Recession EN")
```

```
## pdf
##   2
```

```
ToPlotInd(RFRDat, "Recession FR", "#47d147")
```

```
## pdf
##   2
```

```
ToPlotInd(RCHDat, "Recession ZH", "#8000ff")
```

```
## pdf
##   2
```

```
# Multiple plots (uncomment if wanted)
#ToPlotMult(RENDat, RSPDat, RFRDat, "Recession comparison")
```

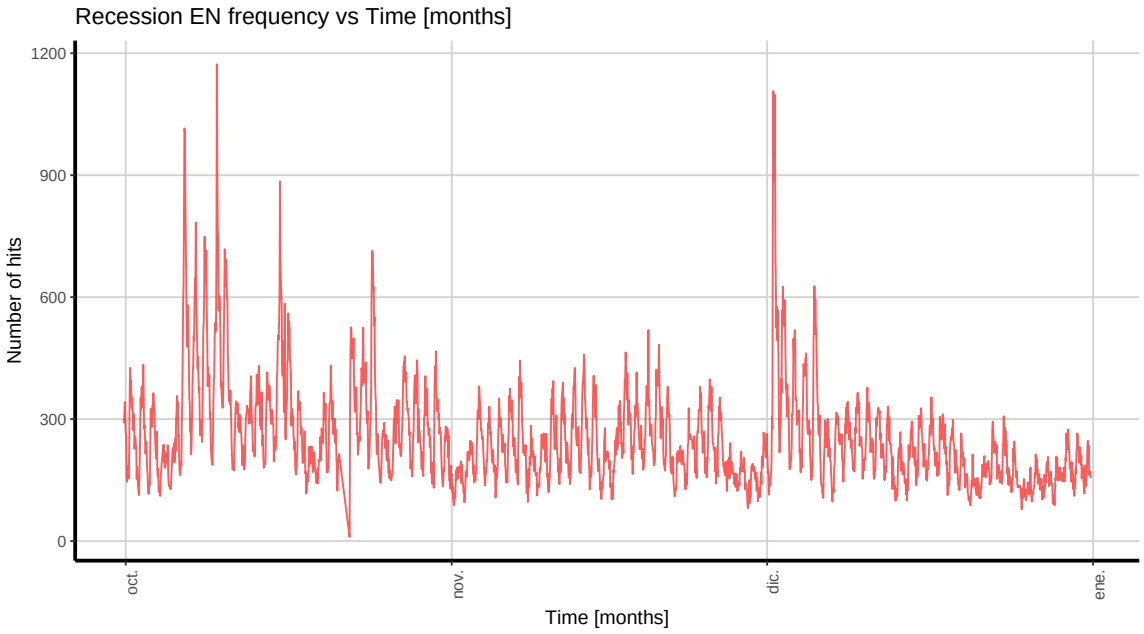

Figure 16: Recession number of hits between 10/08-12/08 English

As expected, the graph associated with English regions tends to present a series of peaks correlated with the evolution of the crisis. As mentioned above, October and December were critical months in the development of the crisis (and its internationalization), being coherent with the information displayed in the graph.

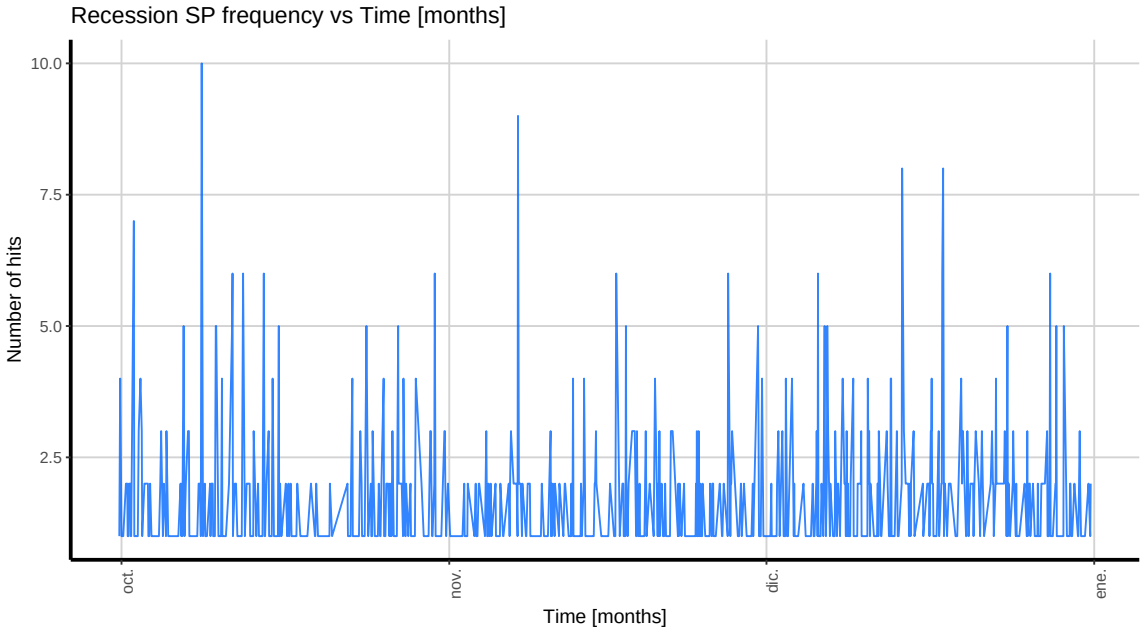

Figure 17: Recession number of hits between 10/08-12/08 Spanish

On the other hand, we can see how the recession was not a very important/relevant topic for Hispanic regions because it was just the "beginning of the end" at that point before the crisis hit all the rest of the world.

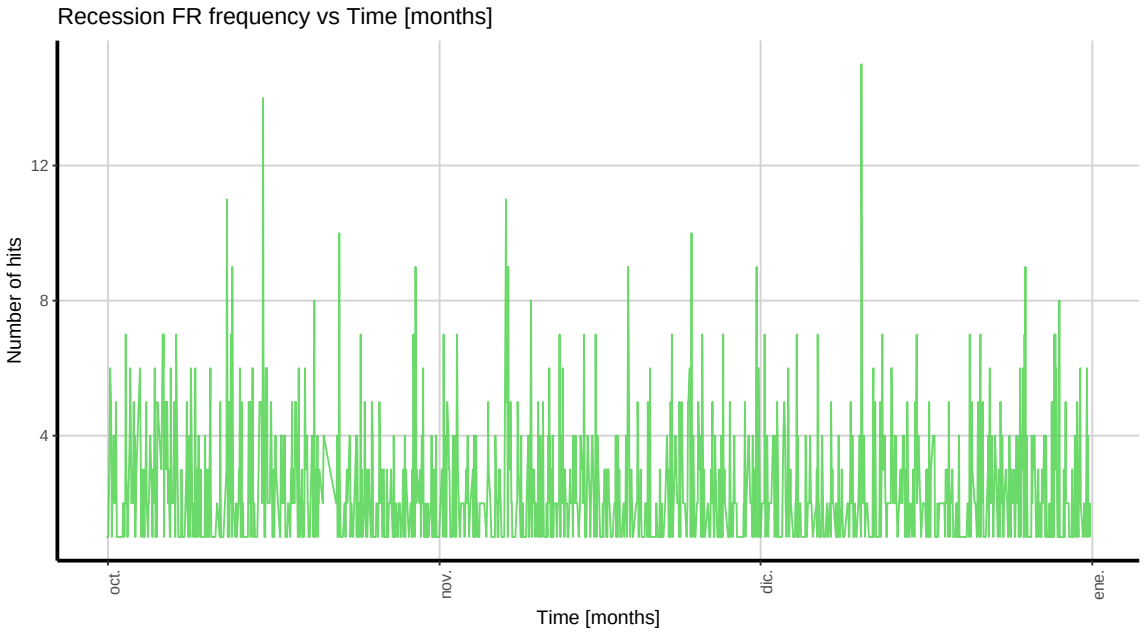

Figure 18: Recession number of hits between 10/08-12/08 French

Same pattern as with Hispanic regions can be identified in the French speakers' analysis: the recession concept was not popular at that point.

Therefore, we have made a series of analysis using the filtered Wikipedia dataset thanks to the processing power of SAVIO and Spark, allowing us to handle large (massive) amounts of data and perform operations that we would unable to implement on a normal laptop or computer.

Problem 4: Parallel processing R vs Spark

In this problem, we will use the SAVIO cluster with R in order to filter the original Wikipedia dataset to a unique DataFrame containing entries related to the "Barack_Obama" expression using a parallel processing approach. Then, in the second part of this section, we will compare our original parallel implementation using $foreach()$ or $parL/Sapply()$ on a single SAVIO node with the theoretical result using SAVIO's 96 cores and compare it with the straight SPARK implementation provided in Unit 8.

Finally, we compare the static and dynamic allocation of jobs (pre-scheduling) to the available cores when performing a parallel processing approach from within R. We will check if there exists a significant difference in terms of performance (total running time) between these two approaches in the third section of this problem.

a) R: foreach/parapply filtering

Our approach is taking advantage of the UNIX basic functions/commands like *grep*. As we already studied at the beginning of the semester, we know that these tools are very efficient when processing large data files such as the text files we want to filter and convert into a unique DataFrame containing the Barack Obama related rows. Therefore, our main strategy is as follows:

i) Based on a list containing all the relevant files (partitions), a loop is performed.

ii) Each file is filtered using the *grep* function, looking for the pattern "Barack_Obama", as asked in the statement of the problem. New filtered files are kept in our local directory inside the SAVIO cluster.

iii) Once the file is generated (in a particular iteration), we use the $read_delim()$ function from the *readr* package, generating a DataFrame containing the relevant information of the filtered file.

iv) Depending on the approach (serial or parallel), we have two different implementations: (1) The final DataFrame is obtained by binding all the individual DataFrames created from each iteration and (2) The final DataFrame is automatically generated by the $foreach()$ function when the *.combine* option is given as an argument (more details below).

For completeness, we include a serial implementation of our strategy in the first place, allowing us to perform a direct and formal comparison of the efficiency of our parallel implementation in relation to the serial one.

1. **Version 1 - Filtered files (serial)**: In this case, we use a traditional for loop implementation for filtering the files using the *grep* function from UNIX, invoking the shell command line from inside R thanks to the $system()$ function. In order to filter the files in an iterative approach, we modify the command passed as an argument to the function by changing the name of the file to be processed.

   After filtering the file, we simple read it using the $read_delim()$ function from the *readr* package.

```
## Version 1: Serial version using filtered files
SerialloopF <- function(files, ListDF, aux){

  # For i in files names
  for(i in files){
    # Command for filtering files by regexp using grep
    cmd <- paste("bash -c \"grep \"Barack_Obama\" ", i, " > ~/ps6/P4/" , i, "Fil",
                 "\"", sep = "")
    system(cmd, intern = TRUE)

    # Declare route to filtered file and read it
    filteredFile <-paste("~/ps6/P4/", i, "Fil", sep = "")
    DF <- readr::read_delim(file=filteredFile, delim = " ", quote = "\"")
```

2. On each iteration, the generated DataFrame is post-processed: the name of the columns is changed and the format of the date and time columns are modified. Finally, the individual DataFrame is stored inside a list that will contain all the 960 DataFrames generated from all the filtered files.

```
    # Postprocessing
    # Date and time as characters
    names(DF) <- c('date','time','lang','concept', 'hits', 'Y')
    DF$date <- as.character(DF$date)
    DF$time <- as.character(DF$time)

    # Time format
    DF$time[DF$time %in%  c("0", "1")] <- "000000"
    wh <- which(nchar(DF$time) == 5)
    DF$time[wh] <- paste0("0", DF$time[wh])

    # Date format
    DF$date <- as.Date(DF$date, "%Y%m%d")

    # List of DFs
    ListDF[[aux]] <- DF
    aux <- aux + 1
  }
```

3. After reading all the files, we remove the last DataFrame created in order to release its memory. Then, a binding operation is performed in order to combine all the individual DataFrames into a unique one containing all rows containing references to Barack Obama. As in the previous problem, we generate a new field containing the date-time information in the *EST* format and finally, the complete DataFrame is returned.

```
  # Remove DF
  remove(DF)

  # Total DataFrame
  TDF <- bind_rows(ListDF[1:aux - 1])
  TDF$date <- as.character(TDF$date)
  TDF$time <- as.character(TDF$time)

  # Chron object: Date and time format
  TDF$chron <- chron(TDF$date, TDF$time,
                     format = c(dates = 'y-m-d', times = "hms"))

  # Time zone format for Date and Hour
  TDF$chron <- as.POSIXlt(TDF$chron, tz = "EST")

  # Order by date (for simplicity)
  TDF <- TDF[order(TDF$chron),]

  # Return full DataFrame
  return(TDF)
}
```

4. Inside the same .R file, we include the main code to be executed in the cluster in order to test the Serial implementation function. We load the relevant libraries (previously installed in our SAVIO session), set

the working directory as the one containing the original non-filtered files (raw Wikipedia data), and a .text file containing all the names of the files is generated by simply use the *ls* command from the shell, thanks to the *system*() function in R.

The main list that will contain all the generated DataFrames is initialized as well as the auxiliary variable for keeping track of the iteration (*aux*). Finally, a micro benchmark call (with only one repetition) is implemented in order to obtain the total running time of the function.

```
## Main code after function (same .R file)
# Load libraries
library(readr)
library(chron)
library(dplyr)
library(microbenchmark)

# Set working directory
setwd("/global/scratch/paciorek/wikistats_full/dated_for_R/")
system('bash -c ls > ~/ps6/P4/filelist.txt')

# Read file names
files <- readLines('~/ps6/P4/filelist.txt')
nfiles <- length(files)

# Initial Data
ListDF <- list(seq(1, nfiles, 1))
aux <- 1

# Time benchmark
print(microbenchmark(TDF <- SerialloopF(files, ListDF, aux), times = 1))

# Check the DataFrame components
length(TDF$date)
TDF$date[1:5]
TDF$date[(length(TDF$date) - 5):length(TDF$date)]
TDF$chron[1:5]
TDF$chron[(length(TDF$chron) - 5):length(TDF$chron)]
```

5. After saving the file as a .R script, we simply upload the file to our SAVIO session. Then, we connect to the cluster, start an interactive session using the *srun* command asking for one node with 24 cores for our session, and finally, we load R in our current session.

```
# Upload R code
cd ~/ps6/RCode/
scp DFSerialSavio.R cpaismz@dtn.brc.berkeley.edu:~/ps6/P4/

# Connect to Savio
ssh cpaismz@hpc.brc.berkeley.edu

# Allocate resources (1 node)
srun -A ic_stat243 -p savio2 --nodes=1 -t 1:00:00 --pty bash

# Load and run R
module load r
R
```

6. Finally, we execute the .R script inside SAVIO:

```
# Run the code
source('~/ps6/P4/DFSerialSavio.R')
```

The outputs obtained from SAVIO are the following:

```
## Unit: seconds
## expr                                          min       lq     mean   median
## TDF <- SerialloopF(files, ListDF, aux) 3512.845 3512.845 3512.845 3512.845
##      uq       max neval
## 3512.845 3512.845     1

## length(TDF$date)
## [1] 432935

## TDF$date[1:5]
## [1] "2008-10-01" "2008-10-01" "2008-10-01" "2008-10-01" "2008-10-01"

## TDF$date[(length(TDF$date)-5):length(TDF$date)]
## [1] "2008-12-31" "2008-12-31" "2008-12-31" "2008-12-31" "2008-12-31"
## [6] "2008-12-31"

## TDF$chron[1:5]
## [1] "2008-09-30 19:00:00 EST" "2008-09-30 19:00:00 EST"
## [3] "2008-09-30 19:00:00 EST" "2008-09-30 19:00:00 EST"
## [5] "2008-09-30 19:00:00 EST"

## TDF$chron[(length(TDF$chron)-5):length(TDF$chron)]
## [1] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
## [3] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
## [5] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
```

Based on the results obtained, we can see that the total time needed for the Serial implementation approach is around 3500 seconds, in other words, the total running time of the filtering function is about one hour, besides the fact that we are implementing our code in a high-performance computer such as SAVIO. This result was expected since we are not taking advantage of the multiple cores inside SAVIO: we are just processing a series of huge data files with more cores than a normal computer without taking advantage of the inherent/natural parallel decomposition approach that can be implemented in a cluster like SAVIO.

1. **Version 2 - Filtered files (parallel)**: In this case, we replace the *for* loop by a *foreach* loop where we include the *.combine = rbind* option for generating the full DataFrame containing all the relevant entries associated with Barack Obama. We can see that since we are using the *foreach* function, the code is simpler since we do not need to create a list beforehand and populate it with the individual DataFrames: the parallel package is doing it for us.

   Therefore, we just need to pass the processed DataFrame to the environment in order to add it to the full DataFrame containing the information from the 960 partitions.

```
## Version 2: create filtered files parallel
ParallelloopF <- function(files){

  # Create a Full DF with foreach combining rows
  foreach(i = files,
          .combine = rbind) %dopar% {

    # Bash command for filtering files
    cmd <- paste("bash -c \"grep \"Barack_Obama\" ", i, " > ~/ps6/P4/" , i, "Fil",
                 "\"", sep = "")
    system(cmd, intern = TRUE)

    # Read filtered file
    filteredFile <-paste("~/ps6/P4/", i, "Fil", sep = "")
    DF <- readr::read_delim(file=filteredFile, delim = " ", quote = "\"")

    # Postprocessing
    # Date and time as characters
    names(DF) <- c('date','time','lang','concept', 'hits', 'Y')
    DF$date <- as.character(DF$date)
    DF$time <- as.character(DF$time)

    # Time format
    DF$time[DF$time %in%  c("0", "1")] <- "000000"
    wh <- which(nchar(DF$time) == 5)
    DF$time[wh] <- paste0("0", DF$time[wh])

    # Date format
    DF$date <- as.Date(DF$date, "%Y%m%d")

    # Return DF
    DF
  }
}
```

2. As before, we include the execution commands inside the same .R file alongside with the previous function. Again, we create a list with the name of all the files by using the shell *ls* command. However, in this case, we define a cluster object with 24 cores (1 node in SAVIO) and we register it in order to initialize the parallel implementation of the code with all the available cores. Then, we execute the microbenchmark as before and we finalize the script by stopping the cluster object, terminating the parallel R processes.

```
## Main code after the function (same .R file)
# Load libraries
require(parallel)
```

```
require(doParallel)
library(foreach)

library(readr)
library(chron)
library(dplyr)
library(microbenchmark)

# Set working directory and create filelist
setwd("/global/scratch/paciorek/wikistats_full/dated_for_R/")
system('bash -c ls > ~/ps6/P4/filelist.txt')

# Register and initialize the parallel environment (24 cores)
nCores <- makeCluster(24)
registerDoParallel(nCores)

# Read the filenames
files <- readLines('~/ps6/P4/filelist.txt')

# Time benchmark and release cores
print(microbenchmark(TDF <- ParallelloopF(files), times = 1))

# Check the DataFrame components
length(TDF$date)
TDF$date[1:5]
TDF$date[(length(TDF$date) - 5):length(TDF$date)]
TDF$chron[1:5]
TDF$chron[(length(TDF$chron) - 5):length(TDF$chron)]

# Realease resources (stop parallel environment)
stopCluster(nCores)
```

3. For completeness, a series of post-processing steps are performed to the full DataFrame: date and time formats plus the definition of the *chron* field. In addition, the DataFrame is sorted by date.

```
# Post-Processing
TDF$date <- as.character(TDF$date)
TDF$time <- as.character(TDF$time)

# Chron object: Date and time format
TDF$chron <- chron(TDF$date, TDF$time,
                   format = c(dates = 'y-m-d', times = "hms"))

# Time zone format for Date and Hour
TDF$chron <- as.POSIXlt(TDF$chron, tz = "EST")

# Order by date (for simplicity)
TDF <- TDF[order(TDF$chron),]
```

4. In order to execute our code, we upload it from the local working directory to our folder inside the cluster like before. After that, we establish a connection with SAVIO and we invoke the *srun* command for allocating 1 node (24 cores) for our job and we load and execute the R module.

```
# Upload R code
cd ~/ps6/RCode/
scp DFParallelSavio.R cpaismz@dtn.brc.berkeley.edu:~/ps6/P4/

# Connect to Savio
ssh cpaismz@hpc.brc.berkeley.edu

# Allocate resources (1 node)
srun -A ic_stat243 -p savio2 --nodes=1 -t 1:00:00 --pty bash

# Load and run R
module load r
R
```

5. Finally, we execute the .R script in SAVIO:

```
# Run the code
source('~/ps6/P4/DFParallelSavio.R')
```

```
## Loading required package: parallel
## Loading required package: doParallel
## Loading required package: foreach
## foreach: simple, scalable parallel programming from Revolution Analytics
## Use Revolution R for scalability, fault tolerance and more.
## http://www.revolutionanalytics.com
## Loading required package: iterators

## Attaching package: 'chron'

## The following object is masked from 'package:foreach':

##    times


## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':

##    filter, lag

## The following objects are masked from 'package:base':

##    intersect, setdiff, setequal, union

## Unit: seconds
##                       expr      min       lq     mean   median       uq
## TDF <- ParallelloopF(files) 243.4427 243.4427 243.4427 243.4427 243.4427
##      max neval
## 243.4427     1

## length(TDF$date)
## [1] 432935
```

```
## TDF$date[1:5]
## [1] "2008-10-01" "2008-10-01" "2008-10-01" "2008-10-01" "2008-10-01"

## TDF$date[(length(TDF$date) - 5):length(TDF$date)]
## [1] "2008-12-31" "2008-12-31" "2008-12-31" "2008-12-31" "2008-12-31"
## [6] "2008-12-31"

## TDF$chron[1:5]
## [1] "2008-09-30 19:00:00 EST" "2008-09-30 19:00:00 EST"
## [3] "2008-09-30 19:00:00 EST" "2008-09-30 19:00:00 EST"
## [5] "2008-09-30 19:00:00 EST"

## TDF$chron[(length(TDF$chron) - 5):length(TDF$chron)]
## [1] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
## [3] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
## [5] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
```

Based on the previous outputs, we can clearly see how in this case the performance of the code is far better than before due to the fact that we are exploiting the natural parallel implementation of the algorithm (the *for* loop) inside the cluster. In this case, the total running time for obtaining exactly the same DataFrame is only about 240 seconds ($\approx$ 4 minutes), representing a 6.67% of the total running time needed for the serial implementation. Hence, the parallel approach is clearly the most suitable one for the problem under study.

b) R vs Spark theoretical comparison

Assuming that our code obtains a perfect scalability when using four times the number of cores (4 nodes instead of only one) as used in the previous section, we can easily calculate the expected running time of the algorithm:

1. Current running time: 243.44 [seconds] $\approx$ 4 [minutes]
2. Theoretical running time (4x): 243.44 / 4 = 60.85 [seconds] $\approx$ 1 [minute]

Comparing the total processing time of our implementation versus the original PySpark implementation with 96 cores, we can see that our approach obtains a 15-fold speed-up in comparison to the Python approach. This is mainly explained due to the fact that we are exploiting and using the very efficient shell commands such as *grep* for pre-processing the Wikipedia dataset partitions and just after filtering the relevant Barack Obama rows, we are using the extremely efficient R function $read_delim()$ for generating the unique Dataframe. Therefore, we never have the full files in the R memory when executing our code: Each filtered file is generated using shell commands from within R (generating smaller files) and then we load and read these filtered datasets (relevant information) into memory in R. Thus, we are just loading the relevant rows in R instead of the full dataset as with the PySpark approach.

c) Prescheduling: dynamic vs static allocation

Based on the documentation of the *doParallel* library [4], we can compare our previous implementation (with dynamic allocation by default) by a simple modification of our previous $ParallelloopF()$ function. In this case, we use the option $preschedule = FALSE$ as an input for the $foreach()$ function, indicating that in this case, the pre-schedule is disabled.

From the documentation we have the following information:

preschedule: *if set to TRUE then the computation is first divided to (at most) as many jobs are there are cores and then the jobs are started, each job possibly covering more than one value. If set to FALSE then one job is spawned for each value of X sequentially (if used with set.seed = FALSE then random number sequences will be identical for all values). The former is better for short computations or large number of values in X, the latter is better for jobs that have high variance of completion time and not too many values of X.*

1. Therefore, we simply add the following line for declaring the list of options we want to use inside the parallel loop:

```
# New option: Deactivate the pre-scheduling in foreach (inside the function
# declaration of ParallalelloopF)
mcoptions <- list(preschedule = FALSE)
```

2. Then, we include the options as an input for the $foreach()$ loop, indicating that we will perform the parallel operations without pre-scheduling them:

```
# Modification inside the foreach loop options: mcoptions is given as
# an input for the options
foreach(i = files,
          .options.multicore = mcoptions,
          .combine = rbind) %dopar% {
```

[4] https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

3. Finally, we upload the new function inside the script *DFParallelSavioNoPS.R*, identical to the previous *DFParallelSavio.R* file but including the new options for not performing the pre-scheduling allocation. The code for running it is similar to the one showed in the previous section:

```
# Upload R code
cd ~/ps6/RCode/
scp DFParallelSavioNoPS.R cpaismz@dtn.brc.berkeley.edu:~/ps6/P4/

# Connect to Savio
ssh cpaismz@hpc.brc.berkeley.edu

# Allocate resources (1 node)
srun -A ic_stat243 -p savio2 --nodes=1 -t 1:00:00 --pty bash

# Load and run R
module load r
R
```

4. As before, we simply run the R code by just calling the script from within the R console.

```
# Run the code
source('~/ps6/P4/DFParallelSavioNoPS.R')
```

```
## Loading required package: parallel
## Loading required package: doParallel
## Loading required package: foreach
## foreach: simple, scalable parallel programming from Revolution Analytics
## Use Revolution R for scalability, fault tolerance and more.
## http://www.revolutionanalytics.com
## Loading required package: iterators

## Attaching package: 'chron'

## The following object is masked from 'package:foreach':

##     times


## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':

##     filter, lag

## The following objects are masked from 'package:base':

##     intersect, setdiff, setequal, union

## Unit: seconds
##                         expr      min       lq     mean   median       uq
##  TDF <- ParallelloopF(files) 574.0687 574.0687 574.0687 574.0687 574.0687
##       max neval
##  574.0687     1
```

```
## length(TDF$date)
## [1] 432935

## TDF$date[1:5]
## [1] "2008-10-01" "2008-10-01" "2008-10-01" "2008-10-01" "2008-10-01"

## TDF$date[(length(TDF$date)-5):length(TDF$date)]
## [1] "2008-12-31" "2008-12-31" "2008-12-31" "2008-12-31" "2008-12-31"
## [6] "2008-12-31"

## TDF$chron[1:5]
## [1] "2008-09-30 19:00:00 EST" "2008-09-30 19:00:00 EST"
## [3] "2008-09-30 19:00:00 EST" "2008-09-30 19:00:00 EST"
## [5] "2008-09-30 19:00:00 EST"

## TDF$chron[(length(TDF$chron)-5):length(TDF$chron)]
## [1] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
## [3] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
## [5] "2008-12-31 18:59:59 EST" "2008-12-31 18:59:59 EST"
```

Based on the results, we can see that the running time of this new implementation is about twice as much as the running time of our original (static allocation; pre-scheduling = TRUE) parallel approach, needing $\sim$ 10 minutes to perform the same task. This was expected since the dynamic allocation incurs in a larger overhead due to the constant communication between the master process and the slaves (for allocating the jobs) and due to the fact that the jobs do not tend to have a high variance of completion since in average they have a similar amount of mentions to the relevant expression, and thus, it is not performing as good as the original static (pre-scheduled) approach.

Hence, we have compared both approaches: static and dynamic job allocation.

Problem 5: Cholesky decomposition analysis

In this section, we will develop a complexity analysis of the Cholesky decomposition algorithm as well as comment one possible storage technique for saving memory when performing the decomposition in a computer.

a) Number of operations

Based on the lecture notes, we have that the Cholesky decomposition algorithm is as follows:

Cholesky Algorithm: Lecture notes style

1) $U_{11} = \sqrt{A_{11}}$
2) **for** j from 2 to n:

$$U_{1j} = \frac{A_{1j}}{U_{11}}$$

3) **for** i from 2 to n:

$$U_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} U_{ki}^2}$$

3.2) **for** j from $i+1$ to n:

$$U_{ij} = \frac{A_{ij} - \sum_{k=1}^{i-1} U_{ki} Ukj}{U_{ii}}$$

Looking at the previous algorithm, we can count the total number of multiplications, divissions, additions/substractions and other operations in order to obtain the total number of calculations needed to perform the Cholesky descomposition.

1) **Square root:** Based on the algorithm, we can clearly see that there are a total of n square root operations.
   i) $U_{11} = \sqrt{A_{11}}$: $+1$
   ii) $U_{1j} = \frac{A_{1j}}{U_{11}}$: $+(n-2+1) = +(n-1)$
2) **Divissions:** In this case, the computation is more complex since we need to analyze the expression inside the inner for loop of the second part of the algorithm. Then, we can simply add these computations to the $n-1$ divisions performed in the first simple loop of the algorithm.
   i) $U_{1j} = \frac{A_{1j}}{U_{11}}$: $+(n-1)$
   ii) $U_{ij} = \frac{A_{ij} - \sum_{k=1}^{i-1} U_{ki} Ukj}{U_{ii}}$
      (ii.1) There are $(n-i)$ divisions inside the inner loop since j moves from $i+1$ to n.
      (ii.2) Then, we have that i moves from 2 to n:

$$\begin{aligned} \sum_{i=2}^{n}(n-i) &= n\sum_{i=2}^{n} 1 - \sum_{i=2}^{n} i && (1)\\ &= n(n-1) - \frac{n(n+1)}{2} + 1 && (2)\\ &= \frac{2n(n-1) - n(n+1) + 2}{2} && (3) \end{aligned}$$

iii) Now, summing the previous expressions we have:

$$\frac{2n(n-1)-n(n+1)+2}{2}+(n-1)=\frac{(n^2-n)}{2} \tag{4}$$

Therefore, we have that the total number of divissions is: $\frac{(n^2-n)}{2}$

3) **Multiplications:** Looking at the algorithm, we can see that the multiplcation operations are performed inside the second for loop, in both the outer and inner iterations. Hence, we first compute the number of multiplications inside the outer loop and then we will sum them with the number of calculations inside the inner one as follows:

i) $U_{ii}=\sqrt{A_{ii}-\sum_{k=1}^{i-1}U_{ki}^2}$: In this case, we have that there are $(i-1)$ multiplications inside each square-root operation. Thus, since i iterates from 2 to n, we have:

$$\sum_{i=2}^{n}(i-1)=\frac{n(n+1)}{2}-1-(n-1)=\frac{n(n+1)}{2}-n \tag{5}$$

ii) Inside the inner loop, we have that the number of multiplications (per expression) is equal to $(i-1)$ (total number of elements in the summation term). Then, since the j index is iterating from $i+1$ to n, we have that there are a total of $(n-i)$ similar operations, obtaining a total of $(n-i)(i-1)$ operations.

iii) Now, since i is iterating from 2 to n, we have that:

$$\begin{aligned}
\sum_{i=2}^{n}(n-i)(i-1) &= \sum_{i=2}^{n}ni-\sum_{i=2}^{n}n-\sum_{i=2}^{n}i^2+\sum_{i=2}^{n}i && (6)\\
&= n\left(\frac{n(n+1)}{2}-1\right)-n(n-1)-\frac{n(n+1)(2n+1)}{6}+1+\frac{n(n+1)}{2}-1 && (7)\\
&= (n+1)\left(\frac{n(n+1)}{2}-1\right)-n(n-1)-\frac{n(n+1)(2n+1)-6}{6} && (8)\\
&= \frac{3n(n^2+2n+1)-6(n+1)-6n(n-1)-n(2n^2+3n+1)+6}{6} && (9)\\
&= \frac{n^3-3n^2+2n}{6} && (10)
\end{aligned}$$

iv) Finally, we just need to sum all the computations:

$$\frac{n(n+1)}{2}-n+\frac{n^3-3n^2+2n}{6}=\frac{n^3-n}{6} \tag{11}$$

Hence, the total number of multiplications performed by this version of the Cholesky decomposition algorithm is equal to $\frac{n^3-n}{6}$

4) **Additions/Substractions:** By looking at the algorithm, we can clearly see that the number of additions/substractions is identical to the number of multiplications $\frac{n^3-n}{6}$

Based on the previous analysis, we can easily compute the total number of calculations (including and not including additions/substractions and square-roots):

1. **Multiplications/Divissions:** In this case, we have a total number of calculations equal to:

$$\frac{n^3-n}{6}+\frac{n(n-1)}{2} = \frac{n^3-n+3n^2-3n}{6} \quad (12)$$

$$= \frac{n^3}{6}+\frac{n^2}{2}-\frac{2n}{3} \quad (13)$$

2. **All calculations:** If we include the excluded operations, we have:

$$\frac{2(n^3-n)}{6}+\frac{n(n-1)}{2}+n = \frac{n^3-n}{3}+\frac{n^2-n}{2}+n \quad (14)$$

$$= \frac{2n^3-2n+3n^2-3n+6n}{6} \quad (15)$$

$$= \frac{n^3}{3}+\frac{n^2}{2}+\frac{n}{6} \quad (16)$$

When comparing our expressions with the ones presented in the lecture notes, we can see that the results are consistent in terms of the overall complexity of the algorithm, defined as $O(n^3)$, as we can clearly see in the previous expressions. In addition, we can see that the total number of computations include the $\frac{n^3}{6}$ term as in the lecture notes (including multiplications and divissions only) plus a second order term $O(n^2)=\frac{n^2}{2}$. Clearly, the linear term is ommited in the lecture notes. Therefore, we can clearly conclude that both calculations (from lecture notes and our own development) match in terms of the overall complexity of the algorithm and the number of calculations needed.

b) Memory storage analysis

Based on the previously defined algorithm, we can easily notice that we can store the information of the U matrix inside the original A matrix without needing to create a new object in memory. This is possible because we can see that we only need the upper diagonal elements of the original A matrix in order to calculate the components of U and we never use the lower diagonal elements. Therefore, we can easily store the current Cholesky decomposition upper triangular matrix in the lower triangular section of the current A matrix while we are calculating it without loosing any relevant information needed for performing the calculations of the decomposition algorithm.

This conclusion is coherent and consistent to what we saw during one of the classes where the Professor showed us some of the classic storage tricks used by computer scientist in order to save some memory and perform more efficient operations.

For completeness (and thanks to our background in CS), we include one implementation of the Cholesky algorithm that takes advantage of this approach [5]:

[5] See https://mediatum.ub.tum.de/doc/625604/625604 for details

Algorithm 2: Cholesky algorithm with modification in place

1) **for** $k = 1$ to n:

   1.2) **for** $j = k + 1$ to n:

$$A_{j,j:n} = A_{j,j:n} - \frac{A_{k,j:n}\bar{A}_{kj}}{A_{kk}}$$

$$A_{k,k:n} = \frac{A_{k,k:n}}{\sqrt{A_{kk}}}$$