Problem Set 2

Thibault Doutre, ID : 26980469

STAT 243 : Introduction to Statistical Computing

I worked on my own.

1 Data processing using bash

Download file.

```
curl -o data.csv.bz2 "www.stat.berkeley.edu/share/paciorek
/ss13hus.csv.bz2"
```

Create script with vim editor.

```
vim script.sh
```

The script contains a function with 4 parameters :

- arg1 : compressed data (.csv.bz2 format)
- arg2 : number of random lines to get
- arg3 : regular expression in order to match the columns we want
- arg4 : set seed, for reproducibility purposes

Algorithm :

- Set seed.
- Set the counter variable "var" to zero. It will indicate the position of every field we want to extract.
- Set the "index" string variable to NULL. This variable will used for the "cut" command. It corresponds to the positions of the fields in the data set.
- Set IFS to "," for convenience in the "for" loop. The machine will interpret the first line of the file as a list of strings.
- For every field, do :
  - Increment "var".
  - Make a regular expression in order to state if the selected field is among the ones we want. If it is not the case nothing happens for this step. Otherwise :
    - Add the position of the field to the "index" variable.
    - Add coma. I differentiate by case when the variable "index" is empty or not in order to avoid beginning with a coma.

  At the end of the loop, "index" looks like "7,12,..." depending on the positions of the fields. We must notice that the list is ordered by index, but it does not matter since variables will be labeled by name.

– Reset IFS to a space (default value)
– Put the first line of the file (i.e. the header) to a new file
– Add 10,000 random lines by doing these steps :
  - Open the csv file
  - Select lines corresponding to the selected fields by using the index variable into the cut command
  - Remove the first line I have already added to the file
  - Shuffle the rows in order to have a random selection
  - Take the first 10,000 rows and add them into the file.

Here is the bash script.

```
#!/bin/bash

subset_data()
{
# arg1 = compressed data, .csv.bz2 format
# arg2 = number of random lines to get
# arg3 = regular expression
# arg4 = seed

RANDOM=$4
var=0
index=""
IFS=","
for i in $( bzcat $1 | head -1 )
do
        var=$((var+1))
        if [[ "$i" =~ $3 ]]
        then
                if [ -n "$index" ]
                then
                        index=$index$","$var
                else
                        index=$var
                fi
        fi
done
IFS=" "
bzcat $1 | head -n 1 | cut -d',' -f $index > data.subset.csv
bzcat $1 | cut -d',' -f $index | sed '1d' | gshuf | head -$2 >>
data.subset.csv
}
```

Execute the script with the bash command. This takes approximately 2 minutes and 30 seconds to run on my laptop.

```
source script.sh ; subset_data data.csv.bz2 10000 "^("ST"|
"NP"|"BDSP"|"BLD"|"RMSP"|"TEN"|"FINCP"|"FPARC"|"HHL"|"NOC"
|"MV"|"VEH"|"YBL")$" 1
```

The CSV file of a random sample of 10,000 households containing the following fields for those households: "ST", "NP", "BDSP", "BLD", "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH" and "YBL" is created.

2 R code

I import the dataset from the CSV file. It has only 10,000 rows so it is much easier for R to deal with it now.

```
setwd('~/Documents/stat243/ps2')
data=as.data.frame(read.csv('./data.subset.csv'))
attach(data)
head(data)

##   ST NP BDSP BLD RMSP TEN VEH YBL FINCP FPARC HHL MV NOC
## 1 17  1    3   2    6   2   1   5    NA    NA   1  7   0
## 2 32  4    4   2    7   3   1   7 28000     3   1  6   0
## 3 19  1   NA  NA   NA  NA  NA  NA    NA    NA  NA NA  NA
## 4 48  2    3   2    6   2   2   3 66000     4   1  5   0
## 5 48  1    3   2    6   2   1   3    NA    NA   2  7   0
## 6 48  0    3   2    8  NA  NA   5    NA    NA  NA NA  NA

nrow(data)

## [1] 10000
```

By calculating the correlation, we can see that NP and NOC have the strongest correlation coefficient among every pairs of variables. It is confirmed when doing a cross-tabulation or plotting the data.

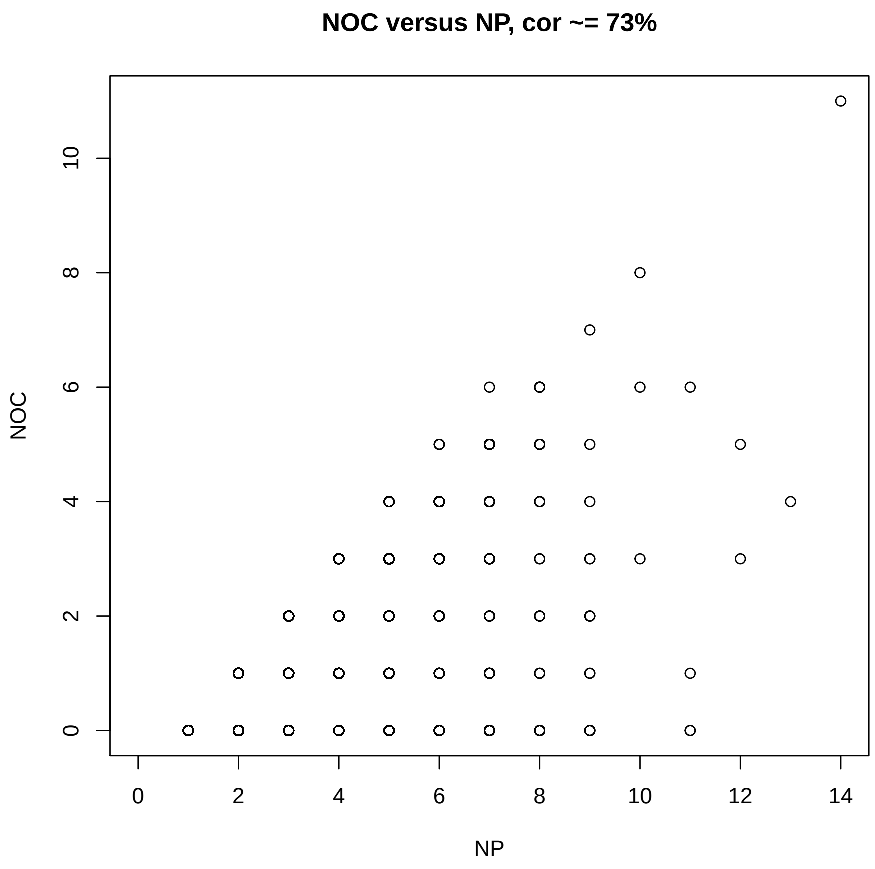
```
cor(na.omit(data)$NP,na.omit(data)$NOC,method="kendall")

## [1] 0.6501196

table(NP,NOC)

##     NOC
## NP      0    1    2    3    4    5    6    7    8   11
##   0     0    0    0    0    0    0    0    0    0    0
##   1  2278    0    0    0    0    0    0    0    0    0
##   2  2767  181    0    0    0    0    0    0    0    0
##   3   639  522  109    0    0    0    0    0    0    0
##    4   247  167  606   43    0    0    0    0    0    0
##    5    99   62   84  219   14    0    0    0    0    0
##    6    31   11   19   32   54    3    0    0    0    0
##    7    14   10    7   10    8   20    1    0    0    0
##    8     8    3    4    2    3    4    4    0    0    0
##    9     5    2    3    2    1    1    0    1    0    0
##    10    0    0    0    1    0    0    1    0    1    0
##    11    2    1    0    0    0    0    1    0    0    0
##    12    0    0    0    1    0    1    0    0    0    0
##    13    0    0    0    0    1    0    0    0    0    0
##    14    0    0    0    0    0    0    0    0    0    1

plot(NP,NOC,main="NOC versus NP, cor ~= 73%")
```



3 Time complexity

Once the bzfile downloaded, the bash script takes approximately 2 minutes and 30 seconds to run on my computer. I think it is more than fair. I could have been much faster by taking the first rows of the dataset with the ”head” command. But in this case, I would have not taken random lines on the whole dataset. As for the R commands, they run in less than a second because it only reads 10,000 rows and a few columns.

4 Comparison between scan() and read.csv()

We obtain better results with opening a file with scan than read.csv. To get better results, we can bzip the file, open a connection to the compressed file and then use scan. However, when trying to compute these two functions to the whole compressed data file, it takes a very long time compared to the required time for the bash script to run.

```r
setwd('~/Documents/stat243/ps2')
con=file('./data.subset.csv')
system.time(read.csv(con))

##    user  system elapsed
##   0.066   0.002   0.069

con=file('./data.subset.csv')
system.time(scan(con,what="integer",sep=","))

##    user  system elapsed
##   0.054   0.000   0.055

close(con)
```