STAT 243 Problem Set 2

Tianyi Zhang

September 17, 2015

1a

In Problem 1a, I first downloaded the csv.bz file to be ss13hus.csv.bz2 using the wget command in bash, then I need to use the command read.csv to read the file chunk by chunk, with the chunk size set to be 10000. But first we need to specify the Colclasses to make the class of columns we don' t want to be NULL so that it will skip these columns. I wrote a function called selectcolumn so that I can call Detailed explanations are along side with the code.

```
selectcolumn<-function(filename){
        selectnames<-c("ST", "NP", "BDSP", "BLD", "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "
        confirst=file(filename, "r")
##This command will read the column names to be a vector called Allnames
        Allnames<-read.csv(confirst,header=FALSE,nrow=1)
        close(confirst)
##Set the columnclass to be an empty vector, and this for loop will make all positions
## of selected columns to be NA and others to be Null
        columnclass=c()
        for (i in Allnames){
                if(is.element(i,selectnames))
                columnclass=c(columnclass,NA)
                else
                columnclass=c(columnclass,"NULL")
        }
        return(columnclass)
}


selectnames<-c("ST", "NP", "BDSP", "BLD", "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH", "Y

##make the first file connection to that zipped file.
confirst=bzfile("ss13hus.csv.bz2", "r")
##This command will read the column names to be a vector called Allnames
Allnames<-read.csv(confirst,header=FALSE,nrow=1)
close(confirst)

## We created a vector called nameposition which records what are the corresponding
## positions in the Allnames, so it's easy for us to add colnames back to the sample.
Allnamesvector<-as.vector(as.matrix(Allnames))
nameposition<-match(selectnames,Allnamesvector)
## The right order vector is the colnames we add back in each step.
rightorder<-Allnamesvector[sort(nameposition)]
print(rightorder)
```

```
##  [1] "ST"    "NP"    "BDSP"  "BLD"   "RMSP"  "TEN"   "VEH"   "YBL"
##  [9] "FINCP" "FPARC" "HHL"   "MV"    "NOC"
```

First we need to see how many rows are there in this file so that we can determine the length of the random vector. We use the bash command wc -l to look at it.

Then I made a vector of length 10000 using the sample function from 7219001, which is the number of rows of the data file. I started from 2 because I put Header=False in the future steps and I don't want to take the column names row to become one of my sample. Then I created a True False vector, and put the selected positions of this vector to be TRUE based on the previous random sample.

```
bzcat ss13hus.csv.bz2 | wc -l

## 7219001
```

```
set.seed(1)
randomsample<-sort(sample(2:7219001,10000))
randomvector<-rep(FALSE,7300000)
randomvector[randomsample]<-TRUE
```

I wrote a function called csvread, which took filename, blocksize, and number of total columns wanted to read as input. In this specific assignment, I would read 10000 chunks each time. To aviod using the command skip in the read.csv, I will set up a file connection called con, which is dynamic. I will also created an empty dataframe with size 10000*13 to place sample extracted from each chunk. By creating a for loop, I am also able to remove each data chunk at the end of the loop to save my memory.

Detailed explanation is alongside with my code.

It took me around 15 mins to finish the sampling of 10000 rows from this sample.

```
csvread<-function(filename,blocksize,numcolumns){
  con<-file(filename,open="r")
##Creating the empty sample dataframe.
  sampledata<-data.frame(matrix(numeric(0),ncol=13, nrow = 10000),stringsAsFactors=FALSE)
##Initialize the position called position_record so that I know
## where to insert in that sample dataframe.
  position_record=0
  columnclass<-selectcolumn(filename)
  for (i in 1:ceiling(numcolumns/blocksize)){
    chunck<-read.csv(con,nrows=blocksize,header=FALSE,colClasses=columnclass,stringsAsFactors=FALSE)
##We extract the sample from this specific chunk, and the upper bound is
## i*blocksize, and the good thing is we can directly use the logical
## vector to take the subset.
    samplefromchunck<-chunck[randomvector[((i-1)*blocksize+1):(i*blocksize)],]
## We subsistute the part of the pre-created data frame to be the one just extracted.
    sampledata[(position_record+1):(position_record+nrow(samplefromchunck)),]<-samplefromchunck
## Update the Position.
    position_record=position_record+nrow(samplefromchunck)
## Remove used data to save memory
    rm(chunck)
    rm(samplefromchunck)
  }
  colnames(sampledata)<-rightorder
  return(sampledata)
  close(con)
}
```

```
system.time(test<-csvread("ss13hus.csv.bz2",100000,7219001))

##    user  system elapsed
## 469.448   0.116 469.066

head(test)

##   ST NP BDSP BLD RMSP TEN VEH YBL     FINCP FPARC HHL MV NOC
## 1 01 01   02  01   06   1   1  07                 1  5  00
## 2 01 05   03  02   06   1   1  06 000047500     1   1  6  00
## 3 01 03   02  06   04   3   1  06 000018000     2   1  1  02
## 4 01 04   03  02   05   1   3  07 000102900     2   1  3  01
## 5 01 03   03  04   05   3   2  06 000000670     1   1  3  01
## 6 01 01   03  01   06   1   1  07                 1  5  00
```

1b

Here we compare readLines and read.csv commands, and their system time. It was found that readLines would be a little faster than read.csv. readLines also returns a vector, so we need to extract that vector to a dataframe each time. We use the package stringr to split",". Then we turn it into another vector, then we flip this vector up to be a matrix with nrow=# of elements in that sample, and now it becomes a dataframe after we applying the vector "rightorder" to take the right columns of that . Other techniques are essentially the same as it was in 1a.

Detailed explanations are along side with the code.

Readline command is a little bit faster than read.csv compared with the result in (a), but not so significant.

```
print(nameposition)

##  [1]  8 12 17 18 32 41 49 50 53 64 63 45 47

##Here is the function that "breaks" every element in the row in to another vector.
library(stringr)
splitvector<-function(x){
        str_split(x, ",")
}
lineread<-function(filename,blocksize,numcolumns){
        con2<-bzfile(filename, "r")
        con2<-bzfile(filename, "r")
        sampledata<-data.frame(matrix(numeric(0),ncol=13, nrow = 10000),stringsAsFactors=FALSE)
        position_record=0
        for (i in 1:ceiling(numcolumns/blocksize)){
                chunck<-readLines(con2,blocksize)
                samplefromchunck<-chunck[randomvector[((i-1)*blocksize+1):(i*blocksize)]]
                samplelines<-lapply(samplefromchunck,splitvector)
## This command transforms a large list, with all elements listed vertically to a dataframe,
## We know the number of rows should be equal to the number of samples in this chunk, thus
## we just set nrow=length and we are done.
                sampledataframe<- data.frame(matrix(unlist(samplelines), nrow=length(samplefromchunck),
                sampledata[(position_record+1):(position_record+nrow(sampledataframe)),]<-sampledatafra
                position_record=position_record+nrow(sampledataframe)
                rm(chunck)
                rm(sampledataframe)
                rm(samplefromchunck)
```

```
                rm(samplelines)
}
        close(con2)
        colnames(sampledata)<-rightorder
        return(sampledata)
}
system.time(b<-lineread("ss13hus.csv.bz2",100000,7219001))

##    user  system elapsed
## 402.636   0.148 402.363

head(b)

##   ST NP BDSP BLD RMSP TEN VEH YBL     FINCP FPARC HHL MV NOC
## 1 01 01   02  01   06   1   1  07                 1  5  00
## 2 01 05   03  02   06   1   1  06 000047500     1   1  6  00
## 3 01 03   02  06   04   3   1  06 000018000     2   1  1  02
## 4 01 04   03  02   05   1   3  07 000102900     2   1  3  01
## 5 01 03   03  04   05   3   2  06 000000670     1   1  3  01
## 6 01 01   03  01   06   1   1  07                 1  5  00
```

1c

In 1c, we utilize bash to speed up our process. We can first cut out those selected columns, and then there are only 13 variables in the file so we apply the same technique as in (a) to the cut dataset. First we store the name position vector we created in part a, and make it to be a sequence of numbers variable in bash. Then we are able to use the bunzip2 command to cut the fields. Notice that bunzip2 -c would not unzip the whole file, instead it will read the zipped fly like a streaming, and we store the field we want to be newdata.csv, which has size 252MB. We obeserved that with preprocessing in bash, it only take less than a mintue to sample the data in R. The rest is the same with part a, we can just call the function csvread wrote in part a to read the file which has been preprocessed.

```
write(nameposition,file="nameposition.txt")
```

```
##Here I make all spaces and newlines to be commas, so that it
## can be read by the cut command in bash.
nameposition=$(sed 's/ /,/g' nameposition.txt | sed 'N;s/\n/,/' | s\
ed 'N;s/\n/,/')

## This function cut the fields needed, this time nameposition is a sequence of numbers.
bunzip2 -c ss13hus.csv.bz2 | cut -d',' -f$nameposition > newdata.csv
```

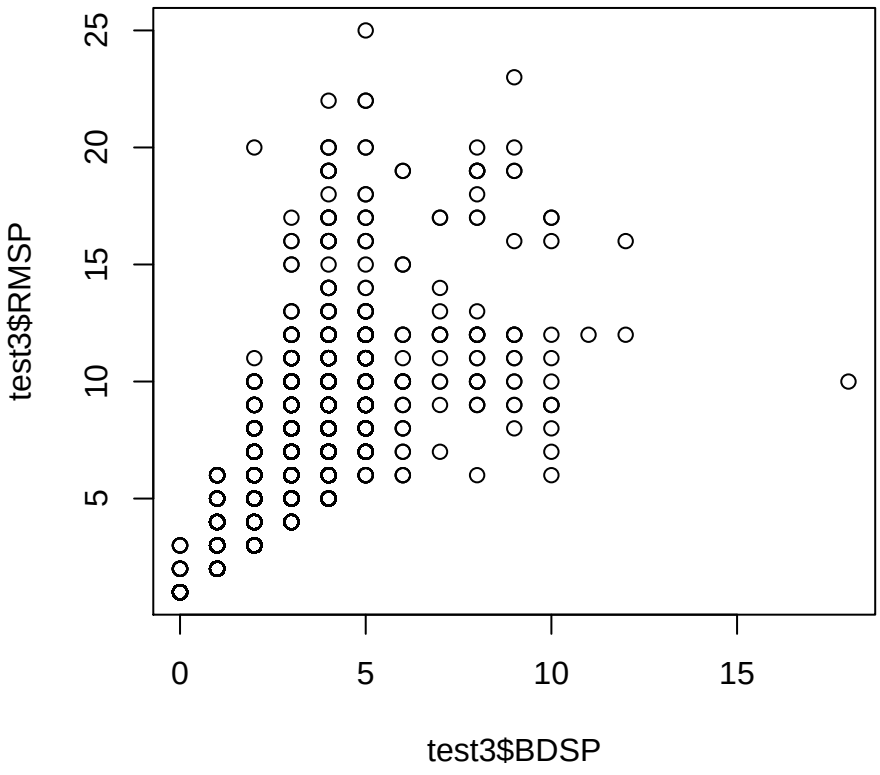
```
system.time(test3<-csvread("newdata.csv",100000,7219001))

##    user  system elapsed
##  18.768   0.060  19.415

head(test3)

##   ST NP BDSP BLD RMSP TEN VEH YBL     FINCP FPARC HHL MV NOC
## 1 01 01   02  01   06   1   1  07                 1  5  00
```

```
## 2 01 05   03  02   06   1   1  06 000047500     1   1  6  00
## 3 01 03   02  06   04   3   1  06 000018000     2   1  1  02
## 4 01 04   03  02   05   1   3  07 000102900     2   1  3  01
## 5 01 03   03  04   05   3   2  06 000000670     1   1  3  01
## 6 01 01   03  01   06   1   1  07                1   5  00
```

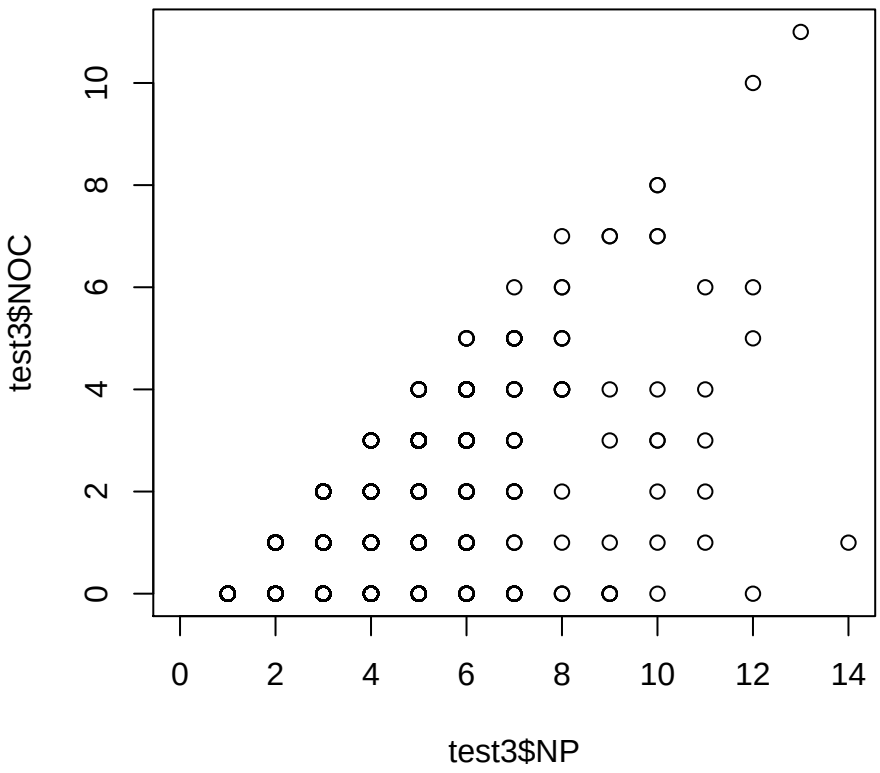
1d

I will do the cross-table for two pairs of varibales to see their correlations to see if our sample data make sense. The pairs I choose are (BDSP,RMSP)-the number of bedrooms and number of rooms and (NP,NOC)-# of people recorded and number of children in the family. Those two pairs should both have the positive correlations intutively, I will plot two scatter plots to illustrate this fact using the sample generated from the approach using bash-preprocessing.

```
plot(test3$BDSP,test3$RMSP)
```




```
plot(test3$NP,test3$NOC)
```



From the result, we observed that both of pairs reveal positive relations, which match the intuitions. Moreover, we can see that Number of children in a family will not exceed number of people in the family, which demonstrates that the sample is extracted correctly.

Notice:

I have consulted some technical points of this problem set with Yueqi Feng and Shamindra Shrotriya, including the mechanism of "file connection", because we originally used "skip" command, which was really slow.