PS2 - STAT 243

Alexander Fred Ojala
Student ID: 26958060
afo@berkeley.edu

September 18th 2015

1 Problem 1

1.1 1. a)

First the analysis was setup. The data file was downloaded into the working directory with *download.file*. Then the header was extracted in order to define what column strings to work with. After that an implemented function, *findCols.R* was run, that returns the indices of the columns that we want to work with and a vector with the same length as the total number of columns in the data set that defines all column classes at the correct position (the class 'character' was used throughout, and not 'numeric, in order to capture all the zeros in front of a number). All other entries in the colclass vector are set to 'NULL'. Last the column indices were written to a file, that later will be used when pre-processing the data in bash (see Problem 1 c)):

```
## Initial setup ----
set.seed(0)
setwd("~/src/stat243/ps2/")
download.file('http://www.stat.berkeley.edu/share/paciorek/ss13hus.csv.bz2','dat.csv.bz2')
#Download file, name it dat.csv.bz2

## Find column indicies and set column classes ----

head<-readLines(bzfile("dat.csv.bz2"),1) # Extract data header
cols=c("ST", "NP", "BDSP", "BLD", "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH", "YBL")
#specify the columns to work with

source("findCols.R") #function to find column indices and column classes

index<-findCols(head,cols)[[1]]
colclass<-findCols(head,cols)[[2]]
cols<-findCols(head,cols)[[3]] #the correct placement of the columns in the vector

cols

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

cat(index,sep=",",file="index.txt") #input for bash pre-processing
```

After that the total number of rows in the data is counted below (this only needs to be done once, and is easier to do in bash while preprocessing the data - see the section 1 c)). When that was done the parameters for the analysis were defined, namely sample size, block size (chunks) and the total number of lines to be read (n).

The function sample is used to define the random sample (where the last row deliberately is excluded, because of a problem with readLines and the header in 1b))

```
## Count number of lines in data file ----
con <- file("dat.csv.bz2",open="r")
chunks <- 20000
nLines <- 0
( while((linesread <- length(readLines(con,chunks))) > 0 )
  nLines <- nLines+linesread )

## NULL

close(con)
nLines

## [1] 7219001

#Takes some time, only has to be done once
#Can be done using bash pre-processing, see 1 c)

## Setup data chunks, sample size and total number of lines ----
sampleSize=10000
blockSize=100000 #read in 100.000 rows at a time
n=nLines # Total lines to read in

use1 = sort(sample(n-1,sampleSize))
# Random sample from the whole data set
```

After that a random sample of the data is obtained by the implemented *readcsv.R* function and the output is the subset as well as the running time. The result was written to a .CSV file with the function write.table (where the function parameters were set to no quotations, no row.names, and the same seperator "," as the original data).

```
source("readcsv.R")
result<-readCSV(data = "dat.csv.bz2",blockSize = blockSize, sampleSize = sampleSize,
                n = n, use = use1)
subsetRCSV<-result[[1]]
RCSVtime<-result[[2]]
rm(result)
subsetRCSV[1:5,]

##   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

## Print out random sample to csv file
write.table(subsetRCSV,sep=",",quote=FALSE,row.names=FALSE,file="dat.csv")
#prints output to file dat.csv
```

The content in *readcsv.R* can be seen below. Where the function takes user defined input in order to extract a correct sample set (it has been verifiied to extract the exact correct random rows as compared to when just a large subset of the full data was read in and then the sam indicies were extracted). The method used to read in the file sequentially was to open a bzfile connection with the option 'r' and then extract the correct rows from every block / chunk that was read in. Also the option with specified *colClasses* was used to only read in the data that was needed and to speed up the process.

```
## read.csv method for extracting data

readCSV = function(data,blockSize,sampleSize,n,use)
{
subsetRCSV<-data.frame(matrix("NA",sampleSize,length(cols)),
                       stringsAsFactors=FALSE)
#create full data frame in advance
names(subsetRCSV)<-cols #correct header
it=1 #iteration
con <- bzfile(description=data, open="r")
RCSVtime<-print(system.time(for(i in 1:ceiling(n/blockSize)) {
  if(i==1) {
    tmp <- read.csv(con, nrow=blockSize, sep = ',', stringsAsFactors=FALSE,
                    header=TRUE, colClasses = colclass)
#don't extract header
  }
  else {
    tmp <- read.csv(con, nrow=blockSize, sep = ',', stringsAsFactors=FALSE,
                    header=FALSE, colClasses = colclass)
  }
  activeIndex<-which(use<=blockSize & use>0)
# see if we have reached index of random sample
  if(length(activeIndex)>0) {
    subsetRCSV[it:(it+length(activeIndex)-1),]<-tmp[use[activeIndex],]
    it=it+length(activeIndex)
#extract random sample(s)
  }
  use=use-blockSize
}
))
close(con)
return(list(subsetRCSV,RCSVtime))
}
```

1.2 1. b)

Approximately the same solution was implemented for the *readLines* solution as for the *read.csv*.

The skip option was also tested (to skip to the nth row in the data and extract that exact row sample) for both of the commands readLines and read.csv, but that slowed down the process immensely as R had to read through the whole data file in order to get to every n:th row.

The result obtained was that the readLines method only took about seven minutes, while the read.csv method took about 25 minutes to read (the first run, when not compiling the pdf it took about 20 mins).

The result can be seen in the first code section below.
This was kind of surprising (my notion was that *read.csv()* was gonna perform better, since it had done so for all sample sizes and total number of lines (n) that was significantly smaller). The data in the readLines method is extracted in an inefficient way as I could not find a solution on how to extract the correct rows and columns without using *strsplit* on tmp and then implement the for loop inside the last if statement extracting one row at a time as data frame from tmp. This clearly makes the code inefficient and there should be a more elegant solution, but evidently it was more time efficient for the whole data set as can be seen in the output.

```
## Problem b) Compare with readLine
source("readL.R")
result<-readL(data = "dat.csv.bz2",blockSize = blockSize,
              sampleSize = sampleSize, n = n, use = use1)
subsetRL<-result[[1]]
RLtime<-result[[2]]
rm(result)

subsetRL[1:5,] #to check against subsetRCSV

##   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

RCSVtime

##     user   system  elapsed
## 1410.116    3.046 1422.721

RLtime

##    user  system elapsed
## 369.016   1.715 372.994
```

```
## readLines method to read in sample

readL = function(data,blockSize,sampleSize,n,use)
{
  subsetRL<-data.frame(matrix("NA",sampleSize,length(cols)),
                       stringsAsFactors=FALSE)
  names(subsetRL)<-cols
  use=use1+1
  it=1 #iteration

  con <- bzfile(description="dat.csv.bz2", open="r")
  RLtime1<-print(system.time(for(i in 1:10) {
    tmp <- readLines(con,n=blockSize)
    tmp<-strsplit(tmp,",")
    activeIndex<-which(use<=blockSize & use>0)
    if(length(activeIndex)>0) {
      for(j in 1:length(activeIndex)) {
        subsetRL[it,]<-tmp[[use[activeIndex[j]]]][index]
        it=it+1
      }
    }
    use=use-blockSize
  }

  ))
  close(con)
  return(list(subsetRCSV,RLtime1))
}
```

1.3 1. c)

The data can be pre-processed in bash so that we obtain a data file with only the columns of interest and the row index can be added to the beginning of the data at every line - so that the data contains a unique sample number. This also directly gives us the number of rows in the data.

The bash code to do this is specified below. Where `bunzip2 -c` reads the data sequentially, `cut` extracts the columns of interest (using index.txt as created with R in problem 1 a)), `nl -s, -w1 -v0` prepends the line number in the correct format and `bzip2` stores the data in a zip file again. Below only showed for the first five rows

```
## Problem c) Pre-processing in bash ----
bunzip2 -c dat.csv.bz2 | head -10 | cut -d, -f$(cat index.txt) | nl -s, -w1 -v0 | bzip2 > dat2.csv.bz2

bunzip2 -c dat2.csv.bz2 | head -5

## 0,ST,NP,BDSP,BLD,RMSP,TEN,VEH,YBL,FINCP,FPARC,HHL,MV,NOC
## 1,01,00,03,01,06,,,07,,,,,
## 2,01,01,,,,,,,,,,,
## 3,01,02,02,02,06,2,2,06,000091400,4,1,6,00
## 4,01,02,03,02,06,1,2,02,000020000,4,1,5,00
```

1.4 1. d)

Lastly, two cross tabulations were executed. The first one checking the number of bedrooms in a unit against the total number of rooms. The second the number of persons in a unit against the number of children. In order for the analysis to be somewhat correct the pattern should be that the majority of values are concentrated to the diagonal of the tables.

```
## Problem d) Cross-tabulation
nbrBedrooms<-as.numeric(subsetRCSV$BDSP)
nbrRooms<-as.numeric(subsetRCSV$RMSP)
table(nbrBedrooms,nbrRooms)

##            nbrRooms
## nbrBedrooms    1    2    3    4    5    6    7    8    9   10   11   12
##           0  140   13    6    0    0    0    0    0    0    0    0    0
##           1    0  175  472  194   59   27    0    0    0    0    0    0
##          2    0    0  224 1006  642  297  114   59   17   11    1    0
##          3    0    0    0  199 1135 1166  716  328  130   66   23   10
##          4    0    0    0    0   54  236  333  365  228  136   59   31
##          5    0    0    0    0    0    8   36   62   64   54   37   24
##          6    0    0    0    0    0    3    2    3    3    6    1    3
##          7    0    0    0    0    0    0    1    0    1    2    1    4
##          8    0    0    0    0    0    1    0    0    3    5    2    6
##          9    0    0    0    0    0    0    0    1    2    2    2    4
##          10   0    0    0    0    0    1    1    1    3    1    1    1
##          11   0    0    0    0    0    0    0    0    0    0    0    1
##          12   0    0    0    0    0    0    0    0    0    0    0    1
##          18   0    0    0    0    0    0    0    0    0    1    0    0
##           nbrRooms
## nbrBedrooms   13   14   15   16   17   18   19   20   22   23   25
##          0    0    0    0    0    0    0    0    0    0    0    0
##          1    0    0    0    0    0    0    0    0    0    0    0
##          2    0    0    0    0    0    0    0    1    0    0    0
##          3    3    0    2    2    1    0    0    0    0    0    0
##          4    9    6    1    6    6    1    5    3    1    0    0
##          5    7    2    1    3    3    2    0    2    2    0    1
##          6    0    0    2    0    0    0    2    0    0    0    0
##          7    1    1    0    0    2    0    0    0    0    0    0
##          8    1    0    0    0    2    1    5    1    0    0    0
##          9    0    0    0    1    0    0    2    1    0    1    0
##          10   0    0    0    1    2    0    0    0    0    0    0
##          11   0    0    0    0    0    0    0    0    0    0    0
##          12   0    0    0    1    0    0    0    0    0    0    0
##          18   0    0    0    0    0    0    0    0    0    0    0

#number of rooms cross-tabbed with number of bedrooms

nbrPersons<-as.numeric(subsetRCSV$NP)
nbrChildren<-as.numeric(subsetRCSV$NOC)
table(nbrChildren,nbrPersons)

##            nbrPersons
## nbrChildren    0    1    2    3    4    5    6    7    8    9   10   11
##          0    0 2308 2795  592  222   80   39   19    3    6    1    0
##          1    0    0  182  551  169   60   23    3    1    1    1    1
##          2    0    0    0  113  561   66   21    7    1    0    1    1
##          3    0    0    0    0   48  225   37    8    0    1    2    1
##          4    0    0    0    0    0   13   63    8    7    1    1    1
##          5    0    0    0    0    0    0    5   15    4    0    0    0
##          6    0    0    0    0    0    0    0    1    2    0    0    1
##          7    0    0    0    0    0    0    0    0    1    2    2    0
##          8    0    0    0    0    0    0    0    0    0    0    2    0
##          10   0    0    0    0    0    0    0    0    0    0    0    0
##          11   0    0    0    0    0    0    0    0    0    0    0    0
##            nbrPersons
## nbrChildren   12   13   14
##          0    1    0    0
##          1    0    0    1
##          2    0    0    0
##          3    0    0    0
##           4     0    0    0
##           5     1    0    0
##           6     1    0    0
##           7     0    0    0
##           8     0    0    0
##           10    1    0    0
##           11    0    1    0

#number of persons in household cross-tabbed with number of children
```