STAT243: PS 2

Xinyue Zhou

September 21, 2015

1.
a) Reading in using read.csv()
step0: Get a general idea of out data set

```
cd /Users/Xinyue_star/src/stat243/ps2
wget www.stat.berkeley.edu/share/paciorek/ss13hus.csv.bz2
bzcat ss13hus.csv.bz2.1 |wc -l
```

There are 7219001 obs in total. For convenience, so I decided to read 72190 obs in each chunk. And for the first obs, I just use it to select the column.
step1: Make the connection

```
con = file("/Users/Xinyue_star/src/stat243/ps2/ss13hus.csv.bz2.1", "r")
```

step2: Write some helper functions
Helper function 1: Select columns
Extract the first line for selecting. If any element in firstline matchs target, it returns a "TRUE" in corresponding position in vector judge. And finally this function returns a logical vector with 205 elements. 13 of them are TRUE.

```
selColTF = function(target,con){
  firstline = read.csv(con, nrows=1)
  judge = is.element(colnames(firstline), target)
  return(judge)
}
```

Helper function 2: select columns (this one is using in Part b)
A further step for select columns. It's a helper function for buildChunks. This function can returns to a "numeric" "NULL" vector, which can help buildChunks() to skip the columns we don't want.

```
selCol = function(target,con){
  judge = selColTF(target,con)
  filterit = character(length(judge))
  for (i in 1:length(judge))
  {
    if (judge[i]) filterit[i] = "numeric"
    else filterit[i] = "NULL"
  }
  return (filterit)
}
```

Helper function 3: Sample rows, get logical val of sampled index
Notice we set.seed() here in convenience for comparing difference method. In practice, this line should be delete to promise a randomized dataset.

```
sampleInd = function(pop, numSam){
  set.seed(0)
  mysample=sample(1:pop, numSam)
  sampleindex=is.element(c(1:pop), mysample)
  return (sampleindex)
}
```

Helper function 4: Main function of building a Chunks
The for loop is to read the data chunk by chunk to our empty dataset. colClass is the option to skip the columm which signified as "NULL".

```
buildChunks = function(blockSize, pop, numSam,target){
  sampleindex = sampleInd(pop, numSam)
  filterit = selCol(target, con)
  mydata = data.frame(matrix(NA, nrow = numSam, ncol=length(target)))
  lines=1
  sta = 1
  for(i in 1:ceiling(pop/blockSize)){
    end =sta + sum(sampleindex[lines:min(pop, (lines+blockSize-1))])-1
    if (end>sta){
      mydata[sta:end,] = read.csv(
        con, nrow = blockSize, colClasses = filterit)[
          sampleindex[lines:min(pop, (lines+blockSize-1))],]
    }
    sta = end+1
    lines = lines + blockSize
  }
  colnames(mydata) = target
  return (mydata)
}
```

step3: Make the connection
Run the main function to get our dataset. I use system.time() to find of the computing time of the function.

```
target = c("ST", "NP", "BDSP", "BLD",
           "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH", "YBL")
blockSize = 72190
nLines = 7219001
numSam = 10000
system.time(mydataCSV <- buildChunks(blockSize,nLines, numSam,target))

##     user   system  elapsed
## 1442.482    3.907 1448.906

close(con)
```

b) Reading in using read.Lines()
Following the same way in a), we can get judge vector to select column, and we have sampleindex to select rows randomly.

```
buildChunksRL= function(blockSize, pop, numSam,target){
  judge = selColTF(target,con)
  sampleindex = sampleInd(pop, numSam)
  mydataRL = data.frame(matrix(NA, nrow = numSam, ncol=length(target)))
  STA = 1
  lines = 1
```

```
  for(i in 1:ceiling(pop / blockSize)){
    END  = STA + sum(sampleindex[lines:min(pop, (lines+blockSize-1))])-1
    if (END>=STA){
      mydata2 = readLines(con, n = blockSize)[sampleindex[lines:min(pop,lines+blockSize-1)]]
      mydata2.1 = str_split(mydata2,",")
      mydata2.2 = matrix(unlist(mydata2.1), byrow = TRUE, ncol=length(judge))[,judge]
      mydataRL[STA:END,] = mydata2.2
    }
    STA = END+1
    lines = lines +blockSize
  }
  mydataRL = data.frame(data.matrix(mydataRL))
  return(mydataRL)
}
```

Then run the buildChunksRL() using the exactly same parameters:blockSize, nLines, numSam,target. Evaluate the time as well.

```
require(stringr)

## Loading required package:  stringr

con = file("/Users/Xinyue_star/src/stat243/ps2/ss13hus.csv.bz2.1", "r")
target = c("ST", "NP", "BDSP", "BLD",
           "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH", "YBL")
blockSize = 72190
nLines = 7219001
numSam = 10000
system.time(mydataRL <- buildChunksRL(blockSize, nLines, numSam,target))

##     user   system  elapsed
## 1397.528    6.079 1418.689

close(con)
```

Notice that the time of using read.Lines() to read in is a little bit less than using read.csv.

c) Do some preprocessing using bash before read.csv()

First of all, I used R to select the index of column I want to select, and write this into a txt file.

```
con = file("/Users/Xinyue_star/src/stat243/ps2/ss13hus.csv.bz2.1", "r")
target = c("ST", "NP", "BDSP", "BLD",
           "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH", "YBL")
judge = selColTF(target,con)
indx = which(judge =="TRUE")
write(indx,file = "/Users/Xinyue_star/src/stat243/ps2/preprocessing.txt"
      ,ncolumns=length(target), sep=",")
close(con)
```

Then use the bash to filter the column required. Then we get a 257MB txt file, with 13 columns needed.

```
cd /Users/Xinyue_star/src/stat243/ps2/
index = $(cat preprocessing.txt )
bunzip2 -c ss13hus.csv.bz2.1 |cut -d"," -f$index >>/Users/Xinyue_star/Desktop/stat 243/ps2-R/Prob1/pre_dat
```

Then I wrote a function to build the dataset chunk by chunk. It is almost the same with buildChunks(), except that there's no need to use colClass= option any more, since we have very neat data with just 13 columns right now.

```
buildChunksPre = function(blockSize, pop, numSam,target){
  sampleindex = sampleInd(pop,numSam)
  mydataPre = data.frame(matrix(NA, nrow = numSam, ncol=length(target)))
  lines=1
  sta = 1
  for(i in 1:ceiling(pop/blockSize)){
    end =sta + sum(sampleindex[lines:min(pop, (lines+blockSize-1))])-1
    if (end>sta){
      mydataPre[sta:end,] =
        read.csv(con, nrow = blockSize,sep=",")[sampleindex[lines:min(pop, (lines+blockSize-1))],]
    }
    sta = end+1
    lines = lines + blockSize
  }
  colnames(mydataPre) = target
  return(mydataPre)
}
```

Then run the function above using the same parameters. To compare three methods, I deserted the first line.Of course, the runing time was also evaluated.

```
con = file("pre_data.txt", "r")
desert = read.csv(con, nrow = 1,sep=",")
target = c("ST", "NP", "BDSP", "BLD",
           "RMSP", "TEN", "FINCP","FPARC", "HHL", "NOC", "MV", "VEH", "YBL")
blockSize = 72190
nLines = 7219001
numSam = 10000
system.time(mydataPre <- buildChunksPre(blockSize, nLines, numSam,target))

##    user  system elapsed
##  44.238   1.401  46.251

close(con)
```

Notice that the running time of preprocessed data is much less than using read.csv() or read.Lines() directly in R. Therefore, we can conclude that bash is more suitable for preprocessing data.

d) Basic analysis

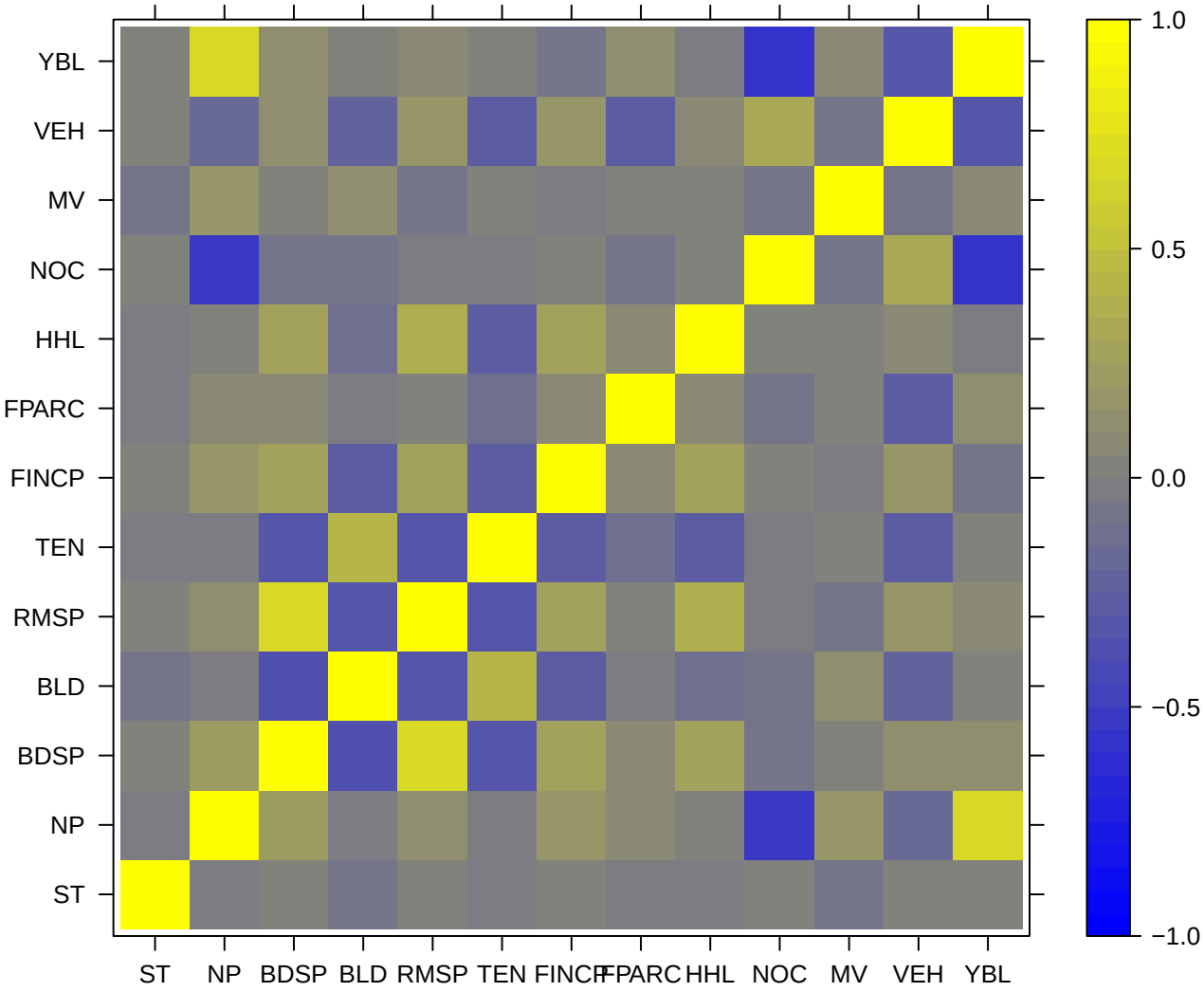
Since the three datasets I got right now are supposed to be the same. For convenience, I just us the third one to do the basic analysis. First compute a table showing correlation, then construct a plot to see the relationship between variables more intuitively.

```
my.cor = cor(mydataPre,use="complete.obs")
my.cor

##                ST          NP         BDSP         BLD         RMSP
## ST     1.000000000 -0.02201639  0.013800778 -0.05553728  0.049642415
## NP    -0.022016387  1.00000000  0.248541396 -0.02132191  0.121629092
## BDSP   0.013800778  0.24854140  1.000000000 -0.35267303  0.676137259
## BLD   -0.055537275 -0.02132191 -0.352673032  1.00000000 -0.321135112
## RMSP   0.049642415  0.12162909  0.676137259 -0.32113511  1.000000000
## TEN   -0.022648218 -0.01619083 -0.315334489  0.41466255 -0.327883713
## FINCP  0.009972303  0.18699882  0.296073997 -0.29810371  0.293805339
## FPARC -0.021765545  0.05249903  0.089886086 -0.01756666  0.037078658
## HHL   -0.015466913  0.04741415  0.288539682 -0.10961277  0.366831377
## NOC    0.020010967 -0.50985932 -0.068281741 -0.06275937 -0.006697494
```

```
## MV    -0.087147336  0.17914447  0.003187582  0.12319635 -0.073895504
## VEH    0.047498791 -0.18988254  0.121323126 -0.24744799  0.152325168
## YBL    0.016563546  0.68458594  0.145228973  0.02042899  0.073662841
##               TEN        FINCP       FPARC          HHL          NOC
## ST    -0.02264822  0.009972303 -0.02176554 -0.015466913  0.020010967
## NP    -0.01619083  0.186998815  0.05249903  0.047414154 -0.509859325
## BDSP  -0.31533449  0.296073997  0.08988609  0.288539682 -0.068281741
## BLD    0.41466255 -0.298103714 -0.01756666 -0.109612768 -0.062759365
## RMSP  -0.32788371  0.293805339  0.03707866  0.366831377 -0.006697494
## TEN    1.00000000 -0.290268525 -0.10794764 -0.273776692 -0.042850854
## FINCP -0.29026853  1.000000000  0.06316935  0.261297729  0.017950015
## FPARC -0.10794764  0.063169353  1.00000000  0.090674347 -0.085144672
## HHL   -0.27377669  0.261297729  0.09067435  1.000000000  0.013407659
## NOC   -0.04285085  0.017950015 -0.08514467  0.013407659  1.000000000
## MV     0.03377429 -0.039771759  0.04354168  0.015656592 -0.093425059
## VEH   -0.25714608  0.156191621 -0.29823782  0.060609743  0.330755438
## YBL    0.03387606 -0.053814942  0.11051934 -0.008500296 -0.590601819
##                MV         VEH          YBL
## ST    -0.087147336  0.04749879  0.016563546
## NP     0.179144472 -0.18988254  0.684585940
## BDSP   0.003187582  0.12132313  0.145228973
## BLD    0.123196350 -0.24744799  0.020428991
## RMSP  -0.073895504  0.15232517  0.073662841
## TEN    0.033774289 -0.25714608  0.033876057
## FINCP -0.039771759  0.15619162 -0.053814942
## FPARC  0.043541684 -0.29823782  0.110519341
## HHL    0.015656592  0.06060974 -0.008500296
## NOC   -0.093425059  0.33075544 -0.590601819
## MV     1.000000000 -0.09322961  0.091635691
## VEH   -0.093229613  1.00000000 -0.335797017
## YBL    0.091635691 -0.33579702  1.000000000
```

```
library(lattice)
#Build the horizontal and vertical axis information
hor <- target
ver <- target
#Build the fake correlation matrix
nrowcol <- length(ver)
#Build the plot
rgb.palette <- colorRampPalette(c("blue", "yellow"), space = "rgb")
levelplot(my.cor, main="stage 12-14 array correlation matrix",
          xlab="", ylab="", col.regions=rgb.palette(120), cuts=100, at=seq(-1,1,0.05))
```



The graph is straightforward. Yellow represent positive correlated, like the diagonal, which are all 1's. Blue shows the negative correlation, like NP and NOC.