STAT243 Problem Set2

Name: Chih Hui Wang SID: 26955255

September 5, 2015

1. (a) I will use R and bash to deal with the problem. Use bash to make the size of file small and R to do random sampling. Firt, I use **curl** to download the bz2 file. and rename it as **PS2data.bz2**. At this point, we are not allowd to directly unzip the data, so I use the modifier **-c** to output the content to observe data. It will not directly unzip the file.

```
#Download
curl -s "www.stat.berkeley.edu/share/paciorek/ss13hus.csv.bz2" -o PS2data.bz2
```

```
#Show data
bunzip2 -c PS2data.bz2 | head -n2

serialno,insp,RT,DIVISION,PUMA00,PUMA10,REGION,ST,ADJHSG,ADJINC,WGTP,NP,TYPE,ACR,AGS,BATH,BDSP,BLD,BU
2009000000015,,H,6,02100,-0009,3,01,1086032,1085467,00025,00,1,3,,1,03,01,2,0000,,,,,,,,,,1,06,,,1,9
bunzip2.exe: Invalid argument

bunzip2.exe: I/O or other error, bailing out.  Possible reason follows.
Input file = PS2data.bz2, output file = (stdout)
```

Now we turn to R to do some processing. Change my R working directory to **Work**, and use **list.files()** to check whether I am in the right directory.

```
setwd("C:/Users/jason/Desktop/Work")
#To check we are in the correct working directory
list.files()

[1] "#ps2.lyx#"   "ps2.lyx"     "ps2.lyx~"    "PS2data.bz2"
```

Because we only need certain column, I use **read.csv** to get the first row and later use R to process it. To avoid the string setting as factor, I use **stringsAsFactors=FALSE**.

```
allcol <- read.csv("PS2data.bz2", nrow=1, header=FALSE, stringsAsFactors=FALSE)
```

To see where those columns locate, I use **%in%**, which give us a row of TRUE/FALSE,and **which** , which give me the location,to get which columns are those we want. After I use paste with argument collapse to make the vector into a string, output the result to number.txt and I can use cut in bash to retrieve those row to decrease the file size.

```
#Column we want
col_want <- c("ST", "NP", "BDSP", "BLD", "RMSP", "TEN", "FINCP",
                          "FPARC", "HHL", "NOC", "MV", "VEH", "YBL")

#Find out which column we want to keep
```

```
coln_want <- which(allcol %in% col_want)
#Output the result for bash
write.table(paste(coln_want, collapse=","), "number.txt", row.names=FALSE, col.names=FALSE)
```

I store the column number in variable **number**. I use sed to remove " by nothing because when using the modifier **-f** for **cut**, we use **,** as seperator for column. For example, to pull out one and second columns → cut -d',' -f1,2 filename.

After we got those numbers for columns we want, we can use cut to take out those column and pipe it into command **gzip** to zip the result, named PS2data.gz.

```
#Get the colname we want
col_n=$(cat number.txt)
#Get all the column number
number=$(echo $col_n | sed 's/"//g')
echo $number
#Eliminate other columns and zip the result
bunzip2 -c PS2data.bz2 | cut -d',' -f$number | gzip -c > PS2data.gz

8,12,17,18,32,41,45,47,49,50,53,63,64
```

Before I start to write my function, I need to know how many rows in the data which then I can sample the row Index from it. I use **readLines** and file connection here. It will read 100000 row for each time until there are no data anymore by **while** loop. I initialize l by 1 to start the while loop, so in the end I have to substract 2 from **total_row**, 1 for the l and 1 for the header.

```
total_row <- 0
datacon <- file("PS2data.gz", open="r")
#Initial
l <- 1
while(l > 0){
        total_row <- total_row + l
        l <- length(readLines(datacon, n=100000))
}
close(datacon); rm(datacon)
total_row <- total_row - 2
#Total number of row in the data
total_row

[1] 7219000
```

my.read.csv contain 3 arguments, file's name **(filename)**, number of sample **(n)**, number of total row **(total_row)**, how many rows we want to read each time **(maxrow)**. I first creat a variable, **breaks**, containing numbers from 1 to 7300001 by every 100000 for later usage. The reason to use 7300001 is that there are data from row 7200001 to 7219000. Then the function will first create a all-0 data frame by n and the length of colname. To avoid the mismatch name, it does not use the colname users provide; instead, it reads the first line, gets its name and puts it to our all-0 data frame. After sampling **(row_want)**, it uses file to open a reading connection, which can help us keep the pointer and escape the slow speed by skipping large number. Then it runs a for-loop which containing the process of reading 100000 each time (**read.csv** with argument **nrow=maxrow**) and also finds the corresponding row we want in our sample **(row_want)** as well as updates the row index **j**. Finally it closes the connection and renders the sample data for user.

There are several points that I should clarify in my function. First, the reason why I do not use the **col_want**(previous variable) as the name of the data is that when we compare the name by **%in%** and **which**, the output will not give location exactly same as the order of our col_want. Hence, it

may be safer to directly use data name. Second, when I wirte the for loop, I use **length(breaks) - 1**. Because the last number of breaks is 7300001 which is bigger than the length of our data, it should not go through the operation in the for loop. Thrid, I use **file** function to create a reading connection here. By doing so, I can avoid use the **skip** function in **read.csv** which makes the speed of reading the data in the last very slow; in other words, I have to skip a large number of row.

```
#function
my.read.csv <- function(filename, n, total_row,maxrow=100000){
                upper <- (total_row %/% maxrow) + 2
                breaks <- seq(1, maxrow*upper + 1, by=maxrow)
                datacon <- file(filename, open='r')
                col_name <- read.csv(datacon, nrow=1, header=FALSE, stringsAsFactors=FALSE)
                data <- matrix(0, nrow=n, ncol=length(col_name))
                data <- as.data.frame(data)
                #Avoid miss match
                names(data) <- col_name
                #Sample
                n.row <- sample(1:total_row, n)
                #Index
                j <- 1
                for(i in 1:(length(breaks) - 1)){
                        row_want <- which(n.row >= breaks[i] & n.row < breaks[i + 1])
                        if(length(row_want) == 0){
                                next
                        }else{
                        data[j:(j + length(row_want) - 1), ] <- read.csv(datacon, nrow=maxrow, header
                        j <- j + length(row_want)
                        }
                }
                close(datacon)
                data
}
set.seed(1)
newdata <- my.read.csv("PS2data.gz", n=10000, total_row, maxrow=100000)
write.csv(newdata, "NewPS2data.csv", row.names=FALSE)
head(newdata)

  ST NP BDSP BLD RMSP TEN VEH YBL FINCP FPARC HHL MV NOC
1  1  4    3   2    7   1   2   7 68800     2   1  5   2
2  1  6    2   2    6   1   4   7 41000     2   1  3   2
3  1  2    2   2    5   2   0   3 58000     4   1  1   0
4  1  2    3   2    5   1   3   6 91000     4   1  6   0
5  1  2    3   2    6   2   2   7  9900     4   1  5   0
6  1  2    2   2    6   4   2   5 28800     4   1  3   0
```

To check that I have output the file correctly.

```
ls -lh NewPS2data.csv
cat NewPS2data.csv | head -n5

-rw-r--r-- 1 jason jason 316K Sep 17 19:17 NewPS2data.csv
"ST","NP","BDSP","BLD","RMSP","TEN","VEH","YBL","FINCP","FPARC","HHL","MV","NOC"
1,4,3,2,7,1,2,7,68800,2,1,5,2
1,6,2,2,6,1,4,7,41000,2,1,3,2
1,2,2,2,5,2,0,3,58000,4,1,1,0
1,2,3,2,5,1,3,6,91000,4,1,6,0
```

(b) In this problem, I will compare three way to read the data and sample it, which is **read.csv**, **read_csv**, **readLines**. My function's strutcure will basically be the same as the one I write about (**my.read.csv**). I finished the one for read.csv.

Now, let us first deal with the function for **read_csv** in the package ***readr***. For **read_csv**, the connection will be unable to use. When it read by the connection, it will automatically close the connection; therefore we lose the pointer which make me have to use **skip** argument in it. Although using **skip** for **read_csv** will be faster than **read.csv**, its speed is still very low.

```
library(readr)
```

```
Warning:  package 'readr' was built under R version 3.2.2
```

```
my.read_csv <- function(filename, n, total_row, maxrow=100000){
                upper <- (total_row %/% maxrow) + 1
                breaks <- seq(1, maxrow*upper + 1, by=maxrow)
                col_name <- names(read_csv(filename, n_max=1))
                data <- matrix(0, nrow=n, ncol=length(col_name))
                data <- as.data.frame(data)
                #Avoid miss match
                names(data) <- col_name
                #Sample
                n.row <- sample(1:total_row, n)
                #Index
                j <- 1
                for(i in 1:(length(breaks) - 1)){
                        row_want <- which(n.row >= breaks[i] & n.row < breaks[i + 1])
                        if(length(row_want) == 0){
                                next
                        }else{
                                data[j:(j + length(row_want) - 1), ] <- read_csv(filename, n_max=maxr
                                j <- j + length(row_want)
                        }
                }
                data
}

set.seed(1)
data <- my.read_csv("PS2data.gz", n=10000, total_row, maxrow=100000)
```

```
|=                                                                      |   1%     3 MB
|=                                                                      |   1%     3 MB
|=                                                                      |   1%     3 MB
|=                                                                      |   1%     3 MB
|=                                                                      |   1%     3 MB
|=                                                                      |   1%     3 MB
|=                                                                      |   1%     3 MB
|=                                                                      |   2%     3 MB
```

```
|=                                                                      |   2%     3 MB
|=                                                                      |   2%     3 MB
|=                                                                      |   2%     3 MB
|==                                                                     |   2%     3 MB
|==                                                                     |   3%     3 MB
|==                                                                     |   3%     3 MB
|===                                                                    |   4%     3 MB
|====                                                                   |   5%     3 MB
|====                                                                   |   6%     3 MB
```

```
head(data)
```

```
  ST NP BDSP BLD RMSP TEN VEH YBL FINCP FPARC HHL MV NOC
1  1  4    3   2    7   1   2   7 68800     2   1  5   2
2  1  6    2   2    6   1   4   7 41000     2   1  3   2
3  1  2    2   2    5   2   0   3 58000     4   1  1   0
4  1  2    3   2    5   1   3   6 91000     4   1  6   0
5  1  2    3   2    6   2   2   7  9900     4   1  5   0
6  1  2    2   2    6   4   2   5 28800     4   1  3   0
```

The next one is **readLines**. It read in each line as a character. Hence, without any processing, I cannot just put the sample into the all-0 data frame. Since the file is seperate by ,. I can use the function **strsplit** to split the character into different columns by ,. However, when I do this operation, some rows' length is not 13 which is our total column number. The probelm present as below. If there is a space next to the last comma, the **strsplit** function will not give us that space. To addfress this problem, I use **gsub** which replace all the , by , (with a space in the end). Then using the **strsplit**, I can get the last space, but another problem occurs. After spliting, the class of the output is list. My strategy is to make the data into a numeric matrix by the function, unlist and putting it into another matrix. By doing so, I can input the value into our data frame.

```
eg <- "I,am,"
strsplit(eg, split=",")
```

```
[[1]]
[1] "I"  "am"
```

```
my.readLines <- function(filename, n, total_row, maxrow=100000){
                upper <- (total_row %/% maxrow) + 1
                breaks <- seq(1, maxrow*upper + 1, by=maxrow)
                datacon <- file(filename, open="r")
                col_name <- strsplit(readLines(datacon, n=1), split=",")[[1]]
                data <- matrix(0, nrow=n, ncol=length(col_name))
                data <- as.data.frame(data)
                names(data) <- col_name
                #Sample
```

```
                n.row <- sample(1:total_row, n)
                j <- 1
                for(i in 1:(length(breaks) - 1)){
                        row_want <- which(n.row >= breaks[i] & n.row < breaks[i + 1])
                        if(length(row_want) == 0){
                                next
                        }else{
                                data_row <- readLines(datacon, n=maxrow)[n.row[row_want] - breaks[i]
                                process <- as.numeric(unlist(strsplit(gsub(",", " ", data_row), spli
                                process_matrix <- matrix(process, nrow=length(row_want), ncol=length(
                                data[j:(j + length(row_want) - 1), ] <- process_matrix
                                j <- j + length(row_want)
                        }
                }
                close(datacon)
                data
}

set.seed(1)
data <- my.readLines("PS2data.gz", n=10000, total_row, maxrow=100000)
head(data)

  ST NP BDSP BLD RMSP TEN VEH YBL FINCP FPARC HHL MV NOC
1  1  4    3   2    7   1   2   7 68800     2   1  5   2
2  1  6    2   2    6   1   4   7 41000     2   1  3   2
3  1  2    2   2    5   2   0   3 58000     4   1  1   0
4  1  2    3   2    5   1   3   6 91000     4   1  6   0
5  1  2    3   2    6   2   2   7  9900     4   1  5   0
6  1  2    2   2    6   4   2   5 28800     4   1  3   0
```

The last one is kind of a mix for above funtion(**readLines** and **read.csv** or **read_csv**). I notice that the **readLines** with connection is really fast when reading data. I take advantage of this point; however I do not make any processing in my for-loop. Instead, I just retrieve those row I want by **readLines**. To solve the problem, I can use the **writeLines** function to write the output into csv file and use read_csv to read the csv file. Therefore, I can save some time for processing the output of **readLines**.

```
library(readr)
my.read.mix <- function(filename, n, total_row, maxrow=100000){
                upper <- (total_row %/% maxrow) + 1
                breaks <- seq(1, maxrow*upper + 1, by=maxrow)
                datacon <- file(filename, open="r")
                data <- matrix(0, nrow=n + 1)
                data[1] <- readLines(datacon, n=1)
                #Sample
                n.row <- sample(1:total_row, n)
                #Index
                j <- 2
                for(i in 1:(length(breaks) - 1)){
                        row_want <- which(n.row >= breaks[i] & n.row < breaks[i + 1])
                        if(length(row_want) == 0){
                                next
                        }else{
                                data[j:(j + length(row_want) - 1)] <- readLines(datacon, n=maxrow)[n.
```

```
                                j <- j + length(row_want)
                        }
                }
                close(datacon)
                writeLines(data, "data.csv")
                read_csv("data.csv")
}

set.seed(1)
data <- my.read.mix("PS2data.gz", n=10000, total_row, maxrow=100000)
head(data)

  ST NP BDSP BLD RMSP TEN VEH YBL FINCP FPARC HHL MV NOC
1  1  4    3   2    7   1   2   7 68800     2   1  5   2
2  1  6    2   2    6   1   4   7 41000     2   1  3   2
3  1  2    2   2    5   2   0   3 58000     4   1  1   0
4  1  2    3   2    5   1   3   6 91000     4   1  6   0
5  1  2    3   2    6   2   2   7  9900     4   1  5   0
6  1  2    2   2    6   4   2   5 28800     4   1  3   0
```

For overall speed performance, **readLines** is faster than **read.csv** and **read.csv** is faster than **read_csv**. The reason why **readLines** is so fast is that it only read line by line data withou processing and putting them into data.frame while the **read.csv** does. I think why **read_csv** is slow in this situation is that when it read the compressed file, it will automatically uncompress it. Also, I do not find a method to use connection with **read_csv** which forces me to use the argument skip. Therefore, whenever it read the data in for-loop, it will uncompress the file. I believe that it is the main reason why **read_csv** cannot outperform over **read.csv** in this case. Actually, I tried to unzip the file and use the similar function(as **my.read_csv**) to read it. Its performance may not be slower than **my.read.csv**.

```
system.time(my.read.csv("PS2data.gz", n=10000, total_row, maxrow=100000))

   user  system elapsed
  32.09    0.02   32.11
```

```
system.time(my.read_csv("PS2data.gz", n=10000, total_row, maxrow=100000))


|=                                                                      |   1%     3 MB

|=                                                                      |   2%     3 MB

|==                                                                     |   2%     3 MB

|==                                                                     |   2%     3 MB

|===                                                                    |   4%     3 MB

|===                                                                    |   4%     3 MB

|=====                                                                  |   6%     3 MB

|======                                                                 |   7%     3 MB
```

```
|=======                                                               |   9%     3 MB
   user  system elapsed
 157.42    6.76  164.25
```

```
system.time(my.readLines("PS2data.gz", n=10000, total_row, maxrow=100000))

   user  system elapsed
  16.62    0.03   16.66
```

```
system.time(my.read.mix("PS2data.gz", n=10000, total_row, maxrow=100000))

   user  system elapsed
  16.56    0.03   16.60
```

(c) I have done it first when I read in the data, which is that I eliminated the column. The dim of processed data is 7219000x13. Compared to the original data, it reduced large size so I can read it faster.

(d) First of all, I make sure every column have the same value or range as the description. Indeed, the range is correct. I will select some variables to make some tables or graphs to detect wether there is any interesting relationship between variable.

```
#Check the field
summary(data)

       ST              NP              BDSP            BLD
 Min.   : 1.0   Min.   : 0.000   Min.   : 0.00   Min.   : 1.000
 1st Qu.:12.0   1st Qu.: 1.000   1st Qu.: 2.00   1st Qu.: 2.000
 Median :27.0   Median : 2.000   Median : 3.00   Median : 2.000
 Mean   :27.8   Mean   : 2.151   Mean   : 2.79   Mean   : 2.922
 3rd Qu.:42.0   3rd Qu.: 3.000   3rd Qu.: 3.00   3rd Qu.: 3.000
 Max.   :56.0   Max.   :14.000   Max.   :18.00   Max.   :10.000
                                 NA's   :909     NA's   :909
      RMSP             TEN             VEH             YBL
 Min.   : 1.000   Min.   :1.000   Min.   :0.000   Min.   : 1.000
 1st Qu.: 4.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.: 3.000
 Median : 6.000   Median :2.000   Median :2.000   Median : 5.000
 Mean   : 5.959   Mean   :1.862   Mean   :1.802   Mean   : 4.947
 3rd Qu.: 7.000   3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.: 7.000
 Max.   :23.000   Max.   :4.000   Max.   :6.000   Max.   :16.000
 NA's   :909      NA's   :1661    NA's   :1661    NA's   :909
     FINCP            FPARC           HHL             MV
 Min.   : -5900   Min.   :1.00   Min.   :1.000   Min.   :1.0
 1st Qu.: 34400   1st Qu.:2.00   1st Qu.:1.000   1st Qu.:3.0
 Median : 61325   Median :4.00   Median :1.000   Median :4.0
 Mean   : 81711   Mean   :3.12   Mean   :1.325   Mean   :4.2
 3rd Qu.:102575   3rd Qu.:4.00   3rd Qu.:1.000   3rd Qu.:5.0
 Max.   :861000   Max.   :4.00   Max.   :5.000   Max.   :7.0
 NA's   :4376     NA's   :4376   NA's   :1661    NA's   :1661
      NOC
 Min.   : 0.0000
 1st Qu.: 0.0000
 Median : 0.0000
 Mean   : 0.5099
```

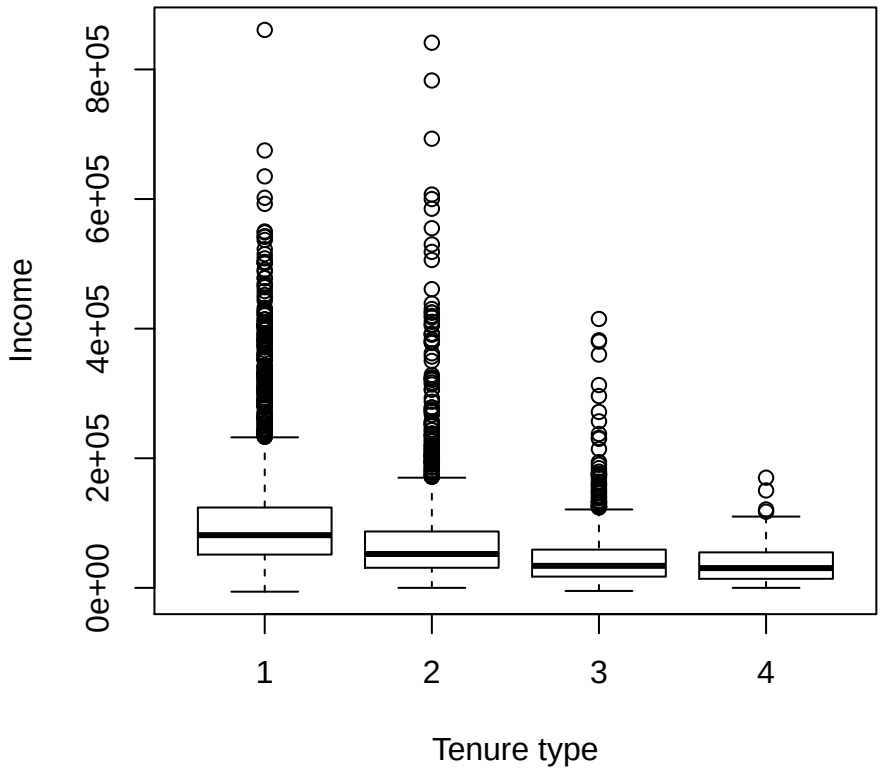
```
 3rd Qu.: 1.0000
 Max.   :11.0000
 NA's   :1661
```

```
round(cor(data, use="complete.obs"), 2)
```

```
         ST    NP  BDSP   BLD  RMSP   TEN   VEH   YBL FINCP FPARC   HHL
ST     1.00 -0.03  0.01 -0.06  0.07 -0.04  0.04 -0.02  0.01  0.01 -0.11
NP    -0.03  1.00  0.25 -0.03  0.11 -0.03  0.19  0.05  0.05 -0.52  0.14
BDSP   0.01  0.25  1.00 -0.35  0.65 -0.32  0.31  0.10  0.29 -0.05 -0.01
BLD   -0.06 -0.03 -0.35  1.00 -0.30  0.39 -0.27 -0.02 -0.11 -0.07  0.20
RMSP   0.07  0.11  0.65 -0.30  1.00 -0.32  0.29  0.06  0.36  0.01 -0.09
TEN   -0.04 -0.03 -0.32  0.39 -0.32  1.00 -0.32 -0.10 -0.29 -0.06  0.07
VEH    0.04  0.19  0.31 -0.27  0.29 -0.32  1.00  0.06  0.27  0.02 -0.05
YBL   -0.02  0.05  0.10 -0.02  0.06 -0.10  0.06  1.00  0.10 -0.08  0.03
FINCP  0.01  0.05  0.29 -0.11  0.36 -0.29  0.27  0.10  1.00  0.01  0.02
FPARC  0.01 -0.52 -0.05 -0.07  0.01 -0.06  0.02 -0.08  0.01  1.00 -0.06
HHL   -0.11  0.14 -0.01  0.20 -0.09  0.07 -0.05  0.03  0.02 -0.06  1.00
MV     0.03 -0.19  0.14 -0.26  0.16 -0.27  0.19 -0.27  0.06  0.35 -0.08
NOC   -0.01  0.74  0.13  0.02  0.05  0.04 -0.04  0.09  0.00 -0.59  0.07
         MV   NOC
ST     0.03 -0.01
NP    -0.19  0.74
BDSP   0.14  0.13
BLD   -0.26  0.02
RMSP   0.16  0.05
TEN   -0.27  0.04
VEH    0.19 -0.04
YBL   -0.27  0.09
FINCP  0.06  0.00
FPARC  0.35 -0.59
HHL   -0.08  0.07
MV     1.00 -0.31
NOC   -0.31  1.00
```

First I compare two variable FINCP(family incom) and TEN(tenure). Originally, I think those family who have a higher income may not have house mortgage which should belong to second category in TEN(own free and clear). However, it turns out that overall family who owns mortgage may have a higher income, although the difference may not be statistically siginificant.

```
with(data, boxplot(FINCP ~ TEN, xlab="Tenure type", ylab="Income"))
```



Then I want to see the distribution of HHL(household language) among ST(state code). It is quite hard to detect any pattern by table. Therefore, I try to visualize them in a better way.

```
#Table
round(with(data, prop.table(table(HHL, ST), 2)), 2)

   ST
HHL    1    2    4    5    6    8    9   10   11   12   13   15   16   17
  1 0.96 0.84 0.73 0.96 0.58 0.86 0.79 0.95 0.90 0.74 0.85 0.64 0.93 0.79
  2 0.03 0.08 0.17 0.00 0.23 0.06 0.11 0.05 0.05 0.17 0.08 0.00 0.05 0.12
  3 0.01 0.08 0.05 0.01 0.06 0.04 0.09 0.00 0.05 0.06 0.03 0.03 0.00 0.07
  4 0.01 0.00 0.02 0.03 0.12 0.03 0.01 0.00 0.00 0.02 0.03 0.33 0.02 0.02
  5 0.00 0.00 0.03 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.01 0.00 0.00 0.00
   ST
HHL   18   19   20   21   22   23   24   25   26   27   28   29   30   31
  1 0.92 0.91 0.84 0.91 0.92 0.88 0.84 0.74 0.90 0.93 0.92 0.94 0.86 0.94
  2 0.04 0.01 0.12 0.03 0.04 0.00 0.08 0.06 0.04 0.03 0.05 0.03 0.03 0.04
  3 0.01 0.04 0.03 0.05 0.04 0.12 0.05 0.17 0.04 0.02 0.01 0.02 0.10 0.02
  4 0.03 0.02 0.01 0.01 0.01 0.00 0.03 0.03 0.01 0.03 0.01 0.01 0.00 0.00
  5 0.00 0.01 0.00 0.00 0.00 0.00 0.01 0.00 0.01 0.00 0.00 0.01 0.00 0.00
   ST
HHL   32   33   34   35   36   37   38   39   40   41   42   44   45   46
  1 0.66 0.97 0.79 0.62 0.73 0.93 0.91 0.94 0.93 0.90 0.88 0.91 0.92 0.95
  2 0.23 0.00 0.09 0.21 0.10 0.03 0.00 0.01 0.06 0.04 0.04 0.00 0.03 0.00
```

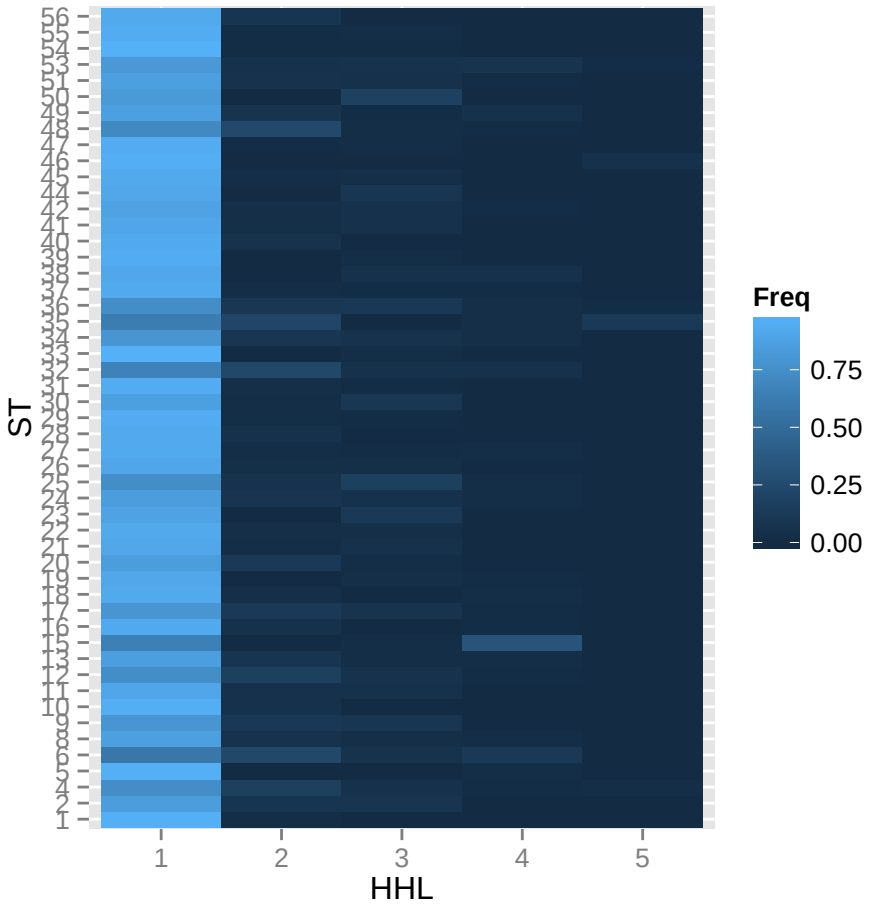
```
  3 0.05 0.03 0.06 0.00 0.11 0.02 0.05 0.03 0.00 0.05 0.05 0.09 0.04 0.00
  4 0.05 0.00 0.04 0.04 0.04 0.02 0.05 0.01 0.01 0.01 0.02 0.00 0.01 0.00
  5 0.00 0.00 0.01 0.12 0.02 0.00 0.00 0.01 0.00 0.00 0.01 0.00 0.00 0.05
   ST
HHL   47   48   49   50   51   53   54   55   56
  1 0.94 0.70 0.86 0.82 0.86 0.81 0.97 0.94 0.92
  2 0.02 0.24 0.07 0.00 0.06 0.05 0.02 0.02 0.08
  3 0.03 0.03 0.02 0.18 0.05 0.06 0.02 0.03 0.00
  4 0.00 0.02 0.05 0.00 0.02 0.07 0.00 0.01 0.00
  5 0.00 0.00 0.00 0.00 0.01 0.02 0.00 0.00 0.00
```

I used the ggplot to visualize the table by tile plot(geom_tile). We can find that there may be other second languages in household for those state coded as 48, 35, 32, 15, 6. It turns out that #48 is Texas, #35 is New Mexico, #32 is Nevado, #15 is Hawaii and #6 is California. Hawaii's Second language is other Indo-European languages while other states are Spanish.

```
t <- as.data.frame(round(with(data, prop.table(table(HHL, ST), 2)), 2))
library(ggplot2)

Loading required package:  methods

#Visualize
ggplot(t) + geom_tile(aes(x=HHL, y=ST, z=Freq, fill=Freq))
```



Lastly, I make a table for NP(number of person records following the house) and vehicle. I find that some family only have few person record like 2, 3 or even 1, but they have 4 or more vehicles, which is quite strange. Maybe they have different vehicle for diferent purpose.

```
with(data, table(NP, VEH))
```

```
    VEH
NP      0    1    2    3    4    5    6
  0     0    0    0    0    0    0    0
  1   422 1512  291   44    8    4    3
  2   142  691 1616  427   81   17   13
  3    74  247  500  349   84   15    9
  4    28  141  465  230  101   35    4
  5    22   61  184  119   45   15    1
  6     9   31   83   56   17   16    7
  7     4   10   23   14   10    6    2
  8     1    5    7    8    2    1    1
  9     0    3    2    4    0    1    1
  10    0    1    1    2    1    1    2
  11    0    1    2    0    1    0    0
  12    1    0    0    0    0    0    0
  13    0    0    0    1    0    0    0
  14    0    0    1    0    0    0    0
```