Project II

Pablo Freyria

3/15/2021

1. First we want to use read.csv() to read the data. This data set is to large so a divide and conquer strategy was approached. First, I need to know how large is the file to know how to divide it. I see that the file is 7,219,001 lines, including header, this means the number of data rows (excluding header) is divisible by 8 and each chunk would be small enough for being opened in MS Excel. The following code chunk does this and takes about 30 minutes to run: 7' in knowing how many lines and 22' in loading the data in 8 dataframes.

```r
# R chunk to read the data into separate chunks, takes around 30 mins to run
library(R.utils)

# read how many lines it has
file <- bzfile("ss13hus.csv.bz2",open = "rb")
time_nLines <- system.time(
                            nrow_file <- countLines(file)
                          )
# file has 7,219,001 lines, about 7 mins (460s) to count all lines
close(file)

## read 8 chunks to have less than 1M lines per chunk (excel supports 1M lines)
# also 7,219 K (without the header) is divisible by 8
# calculate size for each chunks
chunk_size      <- floor(nrow_file/8)

file2 <- bzfile("ss13hus.csv.bz2",open = "rt")
# read data and time it
time_readAll <- system.time(
  {DT1 <- read.csv(file2,header = T,nrows = chunk_size);
   DT2 <- read.csv(file2,header = F,nrows = chunk_size);
   DT3 <- read.csv(file2,header = F,nrows = chunk_size);
   DT4 <- read.csv(file2,header = F,nrows = chunk_size);
   DT5 <- read.csv(file2,header = F,nrows = chunk_size);
   DT6 <- read.csv(file2,header = F,nrows = chunk_size);
   DT7 <- read.csv(file2,header = F,nrows = chunk_size);
   DT8 <- read.csv(file2,header = F,nrows = chunk_size);
  }
)
# around 22 mins (1304s) required to read all data
close(file)

# name columns in all the chunks
colnames(DT2) <- colnames(DT3) <- colnames(DT4) <- colnames(DT5) <-
   colnames(DT6) <- colnames(DT7) <- colnames(DT8) <- colnames(DT1)
```

2. Then I can write a function to sample a “sample_size” from the whole dataset, this can be done by knowing that the data is split evenly in 8 databases of an equal size (“chunk_size”). In this way, I can draw a random id sample, identify which bin it belongs to and use a modulo logic to extract the relative ID in the proper database.

```
#load.data("DataRead_inChunks.RData")

# Randomly sample 3M survey records and extract predetermined data fields
fields <- c("REGION", "ST", "ADJHSG","ADJINC", "NP", "ACR", "BDSP", "ELEP",
            "GASP", "RMSP", "VEH", "WATP", "FINCP", "HINCP")

#check that all data fields are matched
matches <- 0
for(i in fields){matches <- matches + sum(colnames(DT1)==i)}
matches==length(fields)
```

```
## [1] TRUE
```

```
set.seed(1000)
# create random sample
draw_sample <- function(sample_size,cols){
   ids <- sample(c(1:(nrow_file-1)),size = sample_size, replace = F)

   #bin ids to their DT chunk
   ids_bins <- .bincode(ids,c(0:8)*chunk_size,right = T)

   data_sample <- DT1[ids[ids_bins==1],cols]

   data_sample <- rbind(data_sample,DT2[ids[ids_bins==2]-1*chunk_size,cols])
   data_sample <- rbind(data_sample,DT3[ids[ids_bins==3]-2*chunk_size,cols])
   data_sample <- rbind(data_sample,DT4[ids[ids_bins==4]-3*chunk_size,cols])
   data_sample <- rbind(data_sample,DT5[ids[ids_bins==5]-4*chunk_size,cols])
   data_sample <- rbind(data_sample,DT6[ids[ids_bins==6]-5*chunk_size,cols])
   data_sample <- rbind(data_sample,DT7[ids[ids_bins==7]-6*chunk_size,cols])
   data_sample <- rbind(data_sample,DT8[ids[ids_bins==8]-7*chunk_size,cols])

   return(data_sample)
}

data_sample <- draw_sample(3e6,fields)
# write the csv file and save R data
write.csv(data_sample,file = "RandomSample3M.csv")
save.image("DataRead_inChunks.RData")
```

3. For the third task, I can straight forward try run the three proposed ways of reading the 3M sample generated on previous chunk.

```
# Compare three different data reading methods using the 3M rows file generated
# in previous R chunk
library(ff,quietly = T)
library(data.table,quietly = T)
```

```r
rm(list=ls()) #erase everything

time_readcsv <- system.time(
   readcsv <- read.csv("RandomSample3M.csv",header = T)
)

time_fread <- system.time(
   freaddata <- fread(file = "RandomSample3M.csv")
)

time_ffdf <- system.time(
   readcsvffdf <- read.csv.ffdf(x = NULL ,file = "RandomSample3M.csv",colClasses = NA)
)

print("Time elapsed with csv")
```

```
## [1] "Time elapsed with csv"
```

```r
time_readcsv
```

```
##    user  system elapsed
##   11.34    0.16   11.55
```

```r
print("Time elapsed with fread")
```

```
## [1] "Time elapsed with fread"
```

```r
time_fread
```

```
##    user  system elapsed
##    1.03    0.03    0.30
```

```r
print("Time elapsed with csv ffdf")
```

```
## [1] "Time elapsed with csv ffdf"
```

```r
time_ffdf
```

```
##    user  system elapsed
##    9.74    0.38   10.16
```

```r
# keep only 1 data set
data_sample <- freaddata
rm(readcsv)
rm(freaddata)
rm(readcsvffdf)
```

Read.csv takes 11.55 seconds to run, then read.csv.ffdf takes 10.16 seconds, and the fastest is fread with 0.3. It is of notice that fread returns a different object than the other two: Read csv and read csv ffdf read the first column as an integer, while freaddata read it as a list.

4. In the fourth task, I do scatter plot the family income vs the number of bedrooms in a house. Familiy income ('FINCP') was adjusted for inflation (ADJINC) to 2013 dollars, the variable was encoded without the decimal point and with 6 implied decimal figures. A gam (generalized additive model) smoothing filter with the span default of 75%. This parameter controls for how many closest points are used in smoothing (averaging) for each data point.

I first compiled the pdf with all points in plots which resulted in a 100MB file, so a sample of 10K points was taken for the plots and resulted in a 3MB file which was kept as the final.

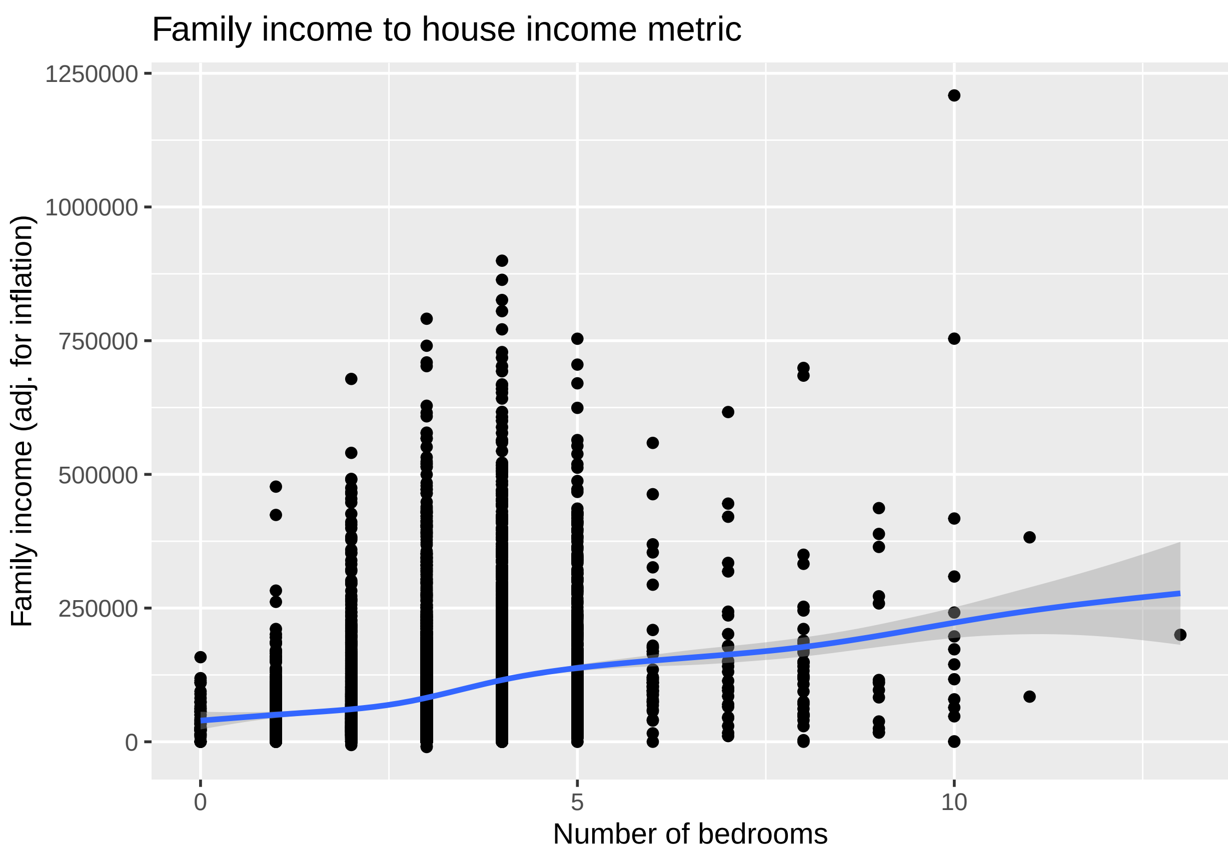
```r
library(ggplot2)
# Draw scatter plot BDSP on x-axis and FINCP y-axis
# using ADJINC to adjust FINCP, with gam smoother


plot_size <- 1e4

# retrieve the significant figures
adjinc <- data_sample$ADJINC/1e6

# get variables for the scaterplot

plot_data <- cbind.data.frame(BDSP = data_sample[,"BDSP"],FINCP = data_sample[,"FINCP"])
rm(data_sample)

plot_data$FINCP <- plot_data$FINCP*adjinc

summary(plot_data)
```

```
##       BDSP            FINCP
##  Min.   : 0.00    Min.   : -30827
##  1st Qu.: 2.00    1st Qu.:  36580
##  Median : 3.00    Median :  66105
##  Mean   : 2.79    Mean   :  87488
##  3rd Qu.: 3.00    3rd Qu.: 108836
##  Max.   :19.00    Max.   :2075551
##  NA's   :260397   NA's   :1322225
```

```r
# remove the NAs from the scatter plot
plot_data <- plot_data[!is.na(plot_data$FINCP),]
plot_data <- plot_data[!is.na(plot_data$BDSP),]

summary(plot_data)
```

```
##       BDSP            FINCP
##  Min.   : 0.000   Min.   : -30827
##  1st Qu.: 3.000   1st Qu.:  36580
##  Median : 3.000   Median :  66105
##  Mean   : 3.095   Mean   :  87488
##  3rd Qu.: 4.000   3rd Qu.: 108836
##  Max.   :19.000   Max.   :2075551
```

```r
plot_data <- plot_data[sample(c(1:nrow(plot_data)),plot_size, replace = FALSE),]

pl <- ggplot(data = plot_data, aes(x=BDSP,y=FINCP)) + geom_point() +
            stat_smooth(method = "gam")
pl <- pl + ggtitle("Family income to house income metric") +
      xlab("Number of bedrooms") + ylab("Family income (adj. for inflation)")

print(pl)
```



5. Finally, I will compare the time to fit a linear regression model on a random sample of 1M rows and repeat the process for 1000 times. The model will include main terms, quadratic terms and the interaction

```r
library(biglm)

#load database and function to read data
load("DataRead_inChunks.RData")
repeat_runs <- 1e3
fields <- c("FINCP","BDSP", "VEH","ADJINC")

reg_formula <- as.formula(c("FINCP ~ BDSP + VEH + I(VEH*BDSP) +
                            I(BDSP^2) +I(VEH^2)"))

BDSP_coefs <- numeric(repeat_runs)
time_lm_total <-0
```

```r
time_biglm_total <- 0
for(i in c(1:repeat_runs)){
   #draw sample
   set.seed(round(i*100*runif(1,0,1)))
   lin_reg <- draw_sample(1e6,fields)
   lin_reg$FINCP <- lin_reg$FINCP*(lin_reg$ADJINC/1e6)

   time_lm <- system.time(
   lm1 <- lm(reg_formula,data = lin_reg)
)
   time_lm_total <- time_lm_total + time_lm[3]

time_biglm <- system.time(
   biglm1 <- biglm(reg_formula,data = lin_reg)
)
   time_biglm_total <- time_lm_total + time_biglm[3]
   BDSP_coefs[i] <- lm1$coefficients["BDSP"]
}

print(lm1)
```

```
##
## Call:
## lm(formula = reg_formula, data = lin_reg)
##
## Coefficients:
##   (Intercept)           BDSP            VEH  I(VEH * BDSP)      I(BDSP^2)
##       -5823.6        17313.5        20522.8         3998.8         -935.8
##      I(VEH^2)
##       -3794.1
```

```r
summary(biglm1)
```

```
## Large data regression model: biglm(reg_formula, data = lin_reg)
## Sample size =  559360
##                     Coef      (95%       CI)       SE p
## (Intercept)   -5823.5626 -7211.709 -4435.417 694.0730 0
## BDSP          17313.5252 16704.812 17922.238 304.3566 0
## VEH           20522.8399 19775.270 21270.410 373.7851 0
## I(VEH * BDSP)  3998.8091  3806.577  4191.042  96.1163 0
## I(BDSP^2)      -935.7546 -1009.720  -861.789  36.9828 0
## I(VEH^2)      -3794.1015 -3921.839 -3666.364  63.8688 0
```

Both fucntions produced the same model and the lm() function took 6393.91 seconds to run vs. 6394.13 required with the biglm function.

The coefficient for number of rooms had a mean value of 1.7542095×10^4 and a standard deviation of 401.6016531. We can also see the density plot for the coefficient estimates.

```r
library(ggplot2)
BDSP_coefs <- as.data.frame(BDSP_coefs)
colnames(BDSP_coefs) <- "Coeffs"
```

```
plot_dens <- ggplot(data = BDSP_coefs,aes(x = Coeffs)) + geom_density() +
  xlab("Density for BDSP coefficients")
print(plot_dens)
```