Project 3: Computing

Rachael Phillips

April 9, 2018

The file `ss13hus.csv.bz2` under `bCourses/Files/Data/` contains household-specific data from the 2009-2013 US Census American Community Survey. This survey obtains a wealth of information on people and households every year, with about 1% of the total population surveyed in each year. The dictionary describing all the data fields is available as `PUMS-Data-Dictionary-2009-2013.pdf` under the same directory. The zipped file is about 600MB, and be careful about unzipping it. You are *required* to use `R` for this computing project, and need to include your computer code and output in the report. You are *required* to use `Rmd` to write the report, which can easily include the `R` code. There is *no page limit* on this report.

Set up workspace

```
library(dplyr)
library(ff)
library(data.table)
library(ggplot2)
library(biglm)
```

1. Try 3 different commands of reading the zipped data `ss13hus.csv.bz2` into R: `read.csv()`, `scan()`, and `readLines()`. Use system.time() to record and report the time each function requires to read in the data.

```
time_scan <- system.time(dat_scan <- scan("~/Desktop/ss13hus.csv.bz2", what = "character",
                                          quiet = TRUE))
```

```
time_scan
```

```
##    user  system elapsed
## 830.788   4.628 835.352
```

```
time_readLines <- system.time(dat_readLines <- readLines("~/Desktop/ss13hus.csv.bz2"))
```

```
time_readLines
```

```
##    user  system elapsed
## 682.771   3.196 686.997
```

```
time_read.csv <- system.time(dat_csv <- read.csv("~/Desktop/ss13hus.csv.bz2"))
```

```
time_read.csv
```

```
##     user   system  elapsed
## 1261.316   91.665 1358.210
```

The `elapsed` time is the wall clock time taken to execute the function, plus any benchmarking code wrapping it. The `user` time gives the CPU time spent by the current R session and `system` time gives the CPU time spent by the operating system on behalf of the current process.

We see that `readLines` read the file fastest followed by `scan` and `read.csv` was the slowest to read the file.

2. Create a subset of data by randomly sampling 1,000,000 survey records from `ss13hus.csv.bz2`. Extract the following data fields: `REGION`, `ST`, `ADJHSG`, `ADJINC`, `NP`, `ACR`, `BDSP`, `ELEP`, `GASP`, `RMSP`, `VEH`, `WATP`, `FINCP`, `HINCP`. Save the file as a `csv` for subsequent analysis, with rows representing survey records and columns different data fields. Hint: This is *not* a trivial task, considering the data size, and it involves some amount of programming. You may use a "divide-and-conquer" strategy. In addition, for reproducibility, please use set.seed(1000) to set the random seed.

```
set.seed(1000)
dat_subset <- sample_n(dat, 1000000)
dat_subset <- dat_subset %>%
  select(c(REGION, ST, ADJHSG, ADJINC, NP, ACR, BDSP,
  ELEP, GASP, RMSP, VEH, WATP, FINCP, HINCP))
write.csv(dat_subset, "~/Desktop/ss13hus_subset.csv")
```

3. Try 3 different commands of reading the data you create in Step 2 into `R`: `read.csv()`, `data.table()`, and `ff()`. Use `system.time()` to record and report the time each function requires to read in the data.

```
time_subset_csv <- system.time(dat_subset_csv <-read.csv("~/Desktop/ss13hus_subset.csv"))
```

```
time_subset_csv
```

```
##    user  system elapsed
##   7.764   0.111   7.880
```

```
time_subset_dt <-system.time(dat_subset_dt <- data.table::fread("~/Desktop/ss13hus_subset.csv"))
```

```
time_subset_dt
```

```
##    user  system elapsed
##   0.003   0.000   0.027
```

```
time_subset_ff <-system.time(dat_subset_ff <- read.csv.ffdf(file="~/Desktop/ss13hus_subset.csv",
                                                            header = TRUE, colClasses=NA))
```

```
time_subset_ff
```

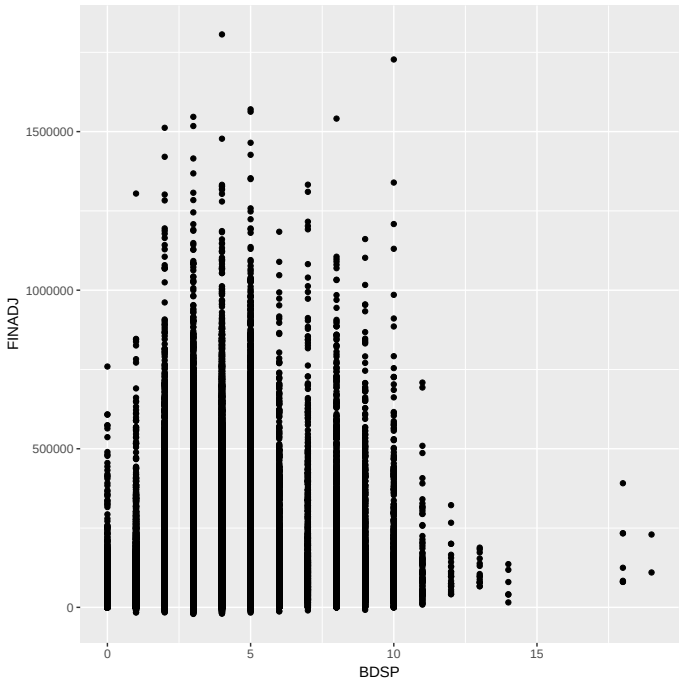
```
##    user  system elapsed
##   8.239   0.267   8.543
```

We see that `data.table::fread` read the subsetted data into R the fastest followed by `read.csv` and `ff::read.csv.ffdf` was just a little slower than `read.csv`. The `data.table::fread` reading time was noticably faster (taking less than 1 second!) than the other two reading times.

4. Draw a scatter plot of `BDSP` (the number of bedrooms; a measure of house size) on the x-axis, and `FINCP` (the family income; use `ADJINC` to adjust `FINCP` to constant dollars) on the y-axis. Add a loess smoother, with standard error shading, on the scatter plot using the R package `ggplot2`.

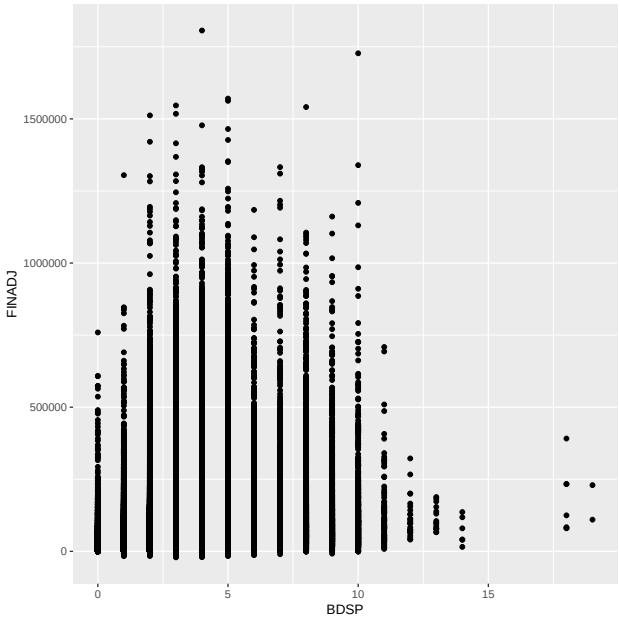
```
dat_subset$FINADJ <- dat_subset$FINCP*((1e-6)*dat_subset$ADJINC)


plot <- ggplot(dat_subset, aes(x = as.factor(BDSP), y = FINADJ)) +
  geom_point() + stat_smooth(method = "loess", se = TRUE)
# If we use all 1,000,000 rows of dat_subset we get the following warning:
# Computation failed in `stat_smooth()`:
# 'Calloc' could not allocate memory (18446744072603478016 of 4 bytes)


# Below, we attempt to reduce the memory allocation by
# removing the NA values in BDSP and FINADJ
dat_sub <- dat_subset %>%
  select(c(BDSP, FINADJ))
dat_sub <- na.omit(dat_sub)
dim(dat_sub)
# [1] 559896      2
plot_reduced <- ggplot(dat_sub, aes(x = as.factor(BDSP), y = FINADJ)) +
  geom_point() + stat_smooth(method = "loess", se = TRUE)
# Even if we use the reduced 559,896 rows of dat_sub we get the following warning:
# Computation failed in `stat_smooth()`:
# 'Calloc' could not allocate memory (18446744072603478016 of 4 bytes)


# Finally, we attempt to reduce the memory allocation by
# selecting random subset of 10,000 from dat_sub
set.seed(1000)
dat_sub_sub <- sample_n(dat_sub, 10000)
plot_reduced2 <- ggplot(dat_sub_sub, aes(x = as.factor(BDSP), y = FINADJ)) +
  geom_point() + stat_smooth(method = "loess", se = TRUE) +
  ggtitle("Adjusted Family Income vs. Number of Bedrooms with Loess Smoother") +
  labs(x = "Number of Bedrooms", y = "Adjusted Family Income")
# No memory warnings this time! stat_smooth() was successful.



# We proceed to examine the output of all 3 plots...
```

```
plot
```



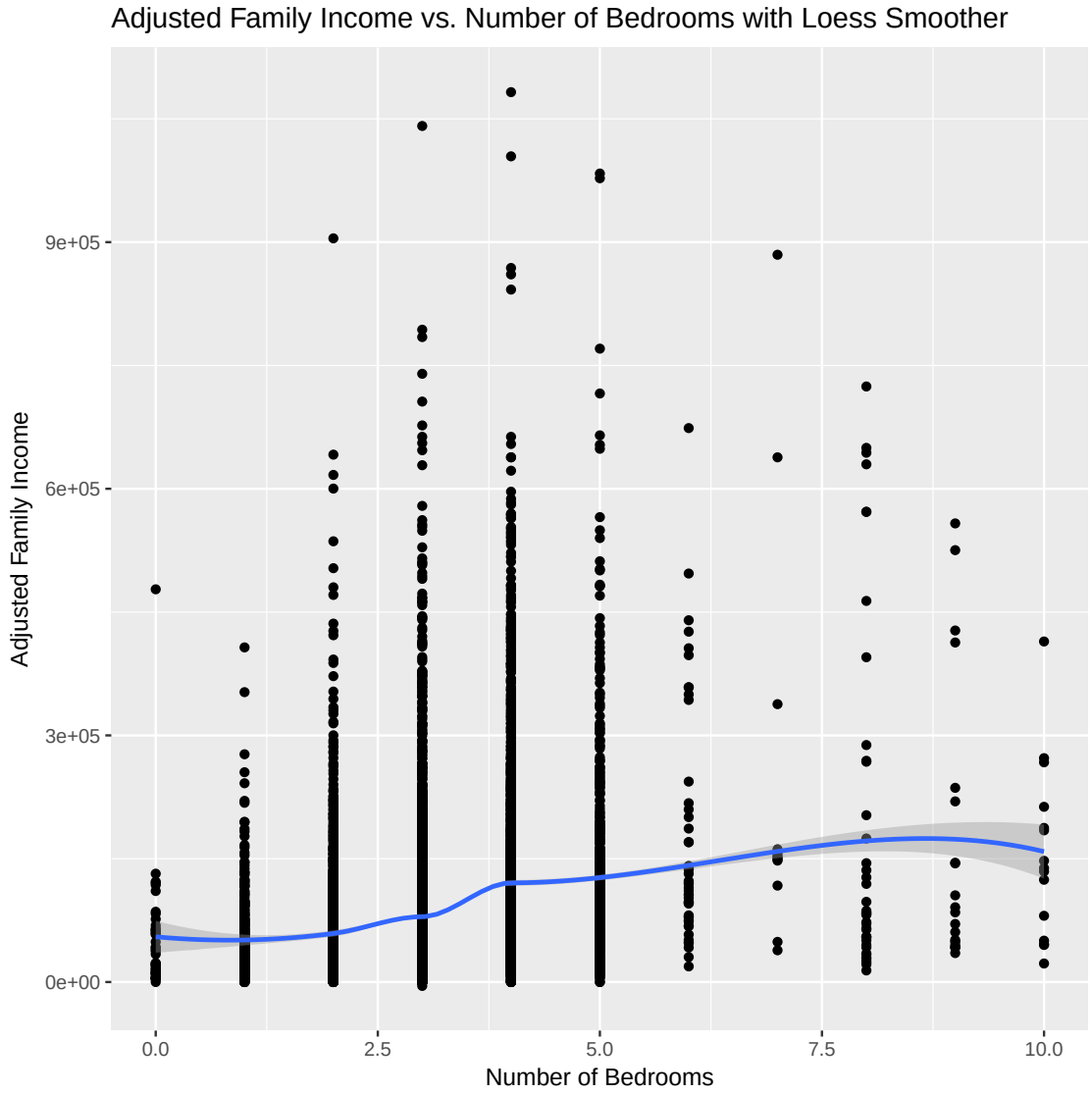
The scatterplot above, `plot`, considers all 1,000,000 values that were subsetted from the original dataset. Due to memory allocation issues, the loess smoother could not be plotted.

```
plot_reduced
```



The scatterplot above, `plot_reduced`, considers the 559,896 non-NA values from the subset of 1,000,000. Due to memory allocation issues, the loess smoother could not be plotted.

```
plot_reduced2
```



The scatterplot above, `plot_reduced2`, considers a random sample of 10,000 from the the 559,896 non-NA values considered in `plot_reduced`.

Even though only `plot_reduced2` fullfilled the requirement of loess smoothing with standard error shading, it is interesting to explore the differences between the three plots. `plot` and `plot_reduced` look the exact same (as expected) but the object size of `plot_reduced` is much smaller than the object size of `plot`. We do not believe it is appropriate to use a loess smoother with a categorical predictor and we still show the loess smoother in `plot_reduced2`, shown above. The standard error appears to be the largest at the tails of the loess smoother.

5. Fit a linear regression model with the adjusted family income as the response, and `BDSP` and `VEH` (the number of vehicles) as the predictors, using the `R` package `biglm`. Report the summary of the regression fitting.

```
linreg <- biglm(FINADJ ~ BDSP + VEH, data = dat_subset)
summ <- summary(linreg)
```

```
summ
```

```
## Large data regression model: biglm(FINADJ ~ BDSP + VEH, data = dat_sub)
## Sample size =  559896
##                   Coef      (95%      CI)       SE p
## (Intercept) -3208.444 -3931.249 -2485.64 361.4023 0
## BDSP        19199.932 18986.229 19413.63 106.8512 0
## VEH         14837.111 14617.969 15056.25 109.5712 0
```

We see that both the coefficient of `BDSP` (the number of bedrooms; a measure of house size) and the coefficient of `VEH` (the number of vehicles) are significant ($p < 0.05$) predictors of `FINADJ` (adjusted family income). Intuitively, these associations makes sense as well. Also, it should be noted that the sample size is not 1,000,000 because NA values are not utilized in the linear regression model. There were quite a few NA values in our random sample!