Solutions to Assignment 1

Valentin Zulj & Vilgot Österlund

September 30, 2018

1 Linear Regression

In this first task we were given a funtction, `linear_regression`, computing OLS estimates of linear regression parameters, and asked to write a function which gives the confidence interval for any given parameter.

We begin by simulating a random set of data to run through the functions:

```
A <- data.frame(Y = rexp(100, rate = 2),          # Depentent variable
               X1 = rnorm(100),                   # Independent variable
               X2 = rnorm(100, mean = 4, sd = 2)) # Independent variable
```

And then proceed to estimate regression parameters, assinging the results to a new object:

```
lin_mod <- linear_regression(data = A, dep = 1, # y is the first column of 'data'
                             indep = c(2,3))     # Regressors are in columns 2 and 3
```

Now, the `linear_regression` function returns estimates of regression parameters and their standard errors, which are extracted from the object `lin_mod` as follows:

```
lin_mod$beta

                   X1         X2
0.35696017 0.03170915 0.02302410

lin_mod$se

                   X1         X2
0.11694248 0.04654479 0.02526974
```

Knowing that, we begin construction of our own function set to return a confidence interval.

1.1 Function

Our interval function takes three arguments, namely `lin_mod`, `pos`, and `alfa`. `lin_mod` consits of the regression output given by `linear_regression`, and `alfa` is simply the singificance level. The `pos` argument denotes the position of the parameter for which we want to estimate the interval, meaning that if the linear regression model contains an intercept `pos = 1`, will yield an interval for the intercept. This is perhaps best illustrated by the code shown at the very end of the last section. In code, the interval looks like:

```
ci <- function(lin_mod, pos, alfa){
  i <- pos                           # Determines which parameter to estimate
  beta <- lin_mod$beta[i]            # Extracting coefficients
  se <- lin_mod$se[i]                # Extracting standard errors
  lower <- beta - qnorm(1-(alfa/2))*se  # qnorm() gives standard normal quantiles
  upper <- beta + qnorm(1-(alfa/2))*se
  out <- list(lower = lower,
              upper = upper,
              c_level = 100*(1-alfa),
              var = pos)             # Objects to use in output
  class(out) <- "linear_regression_ci"  # Creating a new class for output
  return(out)}
```

As is stated, the intervals are calculated using the quantiles of the standard normal distribution, this, of course, being a result of the central limit theorem.

In our function, we create a new class and assign it to the output variable. We do this in order to edit the message printed when `print()` is called on our function. The editing of the function output will be described in Section 1.2.

1.2 Printing Method

As mentioned above, we will now edit the printing method connected to our `ci` function, seeing as we want it to print a message stating what parameter has been used, as well as the confidence level and, of course, the limits of the interval itself. To do this we make use of the fact that `print()` is a generic function, and that it can easily be edited in order to modify the output given when calling on it. In short, we edit the `print()` call of the new class given to the output object of our function, all int accordance with the following code:

```
print.linear_regression_ci <- function(obj){
    print(paste0("A ", obj$c_level,               # The confidence level of the interval
                 "% confidence interval for beta_",
                 obj$var,                          # Shows which parameter we use
                 " is given by: (",
                 round(obj$lower, digits = 3),     # Rounds lower limit to 3 decimals
                 ",",
                 round(obj$upper, digits = 3),     # Rounds upper limit to 3 decimals
                 ")."))}
```

Now, running the `print()` function on an object created using our `ci()` function will result in the following output:

```
int <- ci(lin_mod, pos = 1, alfa = 0.05)
int

[1] "A 95% confidence interval for beta_1 is given by: (0.128,0.586)."
```

Note that in this case, $\hat{\beta}_1$ should be interpreted as the the intercept of the regression model.

2 A Two-sample t-test for Stratified Data

In this task, we were asked to write a function that computes t-tests, and can do so for data that is either stratified or not. We begin by demonstrating the way in which we wrote the code used for stratified data, then move on the ordinary t-test, and finally we combine them into a `t_test()` function.

2.1 Stratified Data

We begin by sampling a set of stratified data to be used in our calculations:

```
set.seed(2018)
strat <- tibble(x = c(rnorm(200, 25), rnorm(200, 45), rnorm(200, 75)),
                treatment = rep(1:2, 300),
                strata = c(rep(1, 200), rep(2, 200), rep(3, 200)))
```

Having done that, we notice there is a need to subset the data, so that we can calculate one set of the needed variables for each treatment group. First, we group our data by treatment and then by stratum, and calculate the values of $\bar{x}$, n, and s^2 for every treatment group in every stratum:

```
d <- strat                      %>%
group_by(treatment, strata)   %>%
summarize(n = length(strata),     # Computing n
          s2 = var (x),           # Computing sample variances
          m = mean(x))            # Computing sample means
d

# A tibble: 6 x 5
# Groups:   treatment [?]
  treatment strata     n    s2     m
      <int>  <dbl> <int> <dbl> <dbl>
1         1      1   100 1.23   25.0
2         1      2   100 0.908  45.1
3         1      3   100 1.14   74.9
4         2      1   100 0.837  25.1
5         2      2   100 1.16   44.9
6         2      3   100 0.806  74.8
```

This allows us to group the remaining data set by stratum and calculate the sums and products given in the formulae:

```
d <- d                          %>%
group_by(strata)                %>%
mutate(sprod = s2*(n-1))        %>%
summarize(nsum = sum(n),           # Summing number of obs
          rnsum = sum(n) - 2,      # Subtracting 2
          ssum = sum(sprod),       # Summing the n-variance products
          nprod = prod(n),         # Multiplying the number of obs
          mdiff = m[1]-m[2])       # Computing the difference in means
```

The output of the above code is a tibble containing sums and products for every stratum, which is all we need to start assembling the test itself:

```
# A tibble: 3 x 6
  strata  nsum rnsum  ssum nprod   mdiff
   <dbl> <int> <dbl> <dbl> <dbl>   <dbl>
1      1   200   198  205. 10000 -0.0556
2      2   200   198  204. 10000  0.251
3      3   200   198  193. 10000  0.0585
```

Now, what we need to do is calculate the weights and the estimate the variances for each stratum, we do this using the formulae given in the assignment, and the following code:

```
d <- d                                         %>%
mutate(weights = (nprod/nsum) / sum(nprod/nsum),      # Computing weights
       sigma2 = (nsum/nprod) * (ssum/rnsum))   %>%   # Computing sigma2
select(mdiff, weights, sigma2)
```

Which yields the following tibble:

```
# A tibble: 3 x 3
    mdiff weights sigma2
    <dbl>   <dbl>  <dbl>
1 -0.0556   0.333 0.0207
2  0.251    0.333 0.0206
3  0.0585   0.333 0.0195
```

As can be seen from the tibble printed above, we have managed to calculate the weights, estimators of the variances, and differences beteen means, which are the last components needed in order to calculate the t-statistic. We calculate the statistic like this:

```
d <- d                                              %>%
summarize(numerator = sum(weights * mdiff),
          denominator = sqrt(sum((weights^2) * sigma2)),
          t_stat = numerator/denominator,                      # Test statistic
          stratified = TRUE)                        %>%        # Logical statement
select(t_stat, stratified)
d

# A tibble: 1 x 2
  t_stat stratified
   <dbl> <lgl>
1   1.03 TRUE
```

And find the result given in a tibble showing the test statistic itself, and a logical statement indicating whether the statistic has been computed using the method for stratified data sets.

In the following section, we will present a method of calculating an ordinary two-sample t-test, which is a simpler form of what we have shown above.

2.2 Classical t-test

Again, we begin by simulating a data set we can use for our calculations:

```
set.seed(2018)
test <- tibble(x = c(rnorm(200, 25), rnorm(200, 45), rnorm(200, 75)),
               treatment = rep(1:2, 300))
```

Then we group our data by treatment only – since there are no strata – and carry on as we did in Section 2.1. Hence, our code will not be broken up and commented as thoroughly as above.

```
t <- test                             %>%
group_by(treatment)                   %>%
summarize(n = length(treatment),
          s2 = var(x),
          m = mean(x))                %>%
mutate(sprod = s2*(n-1))              %>%
summarize(nsum = sum(n),              # Summing number of obs
          rnsum = sum(n) - 2,         # Subtracting 2
          ssum = sum(sprod),          # Summing the n-variance products
          nprod = prod(n),            # Multiplying the number of obs
          mdiff = m[1]-m[2])          # Difference in means
```

This chunk of code will do the exact same thing as it did above, calculating $\bar{x}$, n, s^2 as well as the different sums and products needed to compute the test statistic. What is worth noting, however, is that in the following code the weights will be given as one, seeing as we have no strata to use:

```
t <- t %>%
mutate(weights = (nprod/nsum)/sum(nprod/nsum),  # Computing weights
       sigma2 = (nsum/nprod)*(ssum/rnsum))      # Computing sigma2
t

# A tibble: 1 x 7
   nsum rnsum    ssum nprod  mdiff weights sigma2
  <int> <dbl>   <dbl> <dbl>  <dbl>   <dbl>  <dbl>
1   600   598 251538. 90000 0.0847       1   2.80
```

And finally, we compute the t-statistic:

```
t <- t                              %>%
select(mdiff, weights, sigma2)      %>%
summarize(numerator = sum(weights * mdiff),
          denominator = sqrt(sum((weights^2) * sigma2)),
          t_stat = numerator / denominator,       # Test statistic
          stratified = FALSE)       %>%           # Logical statement
select(t_stat, stratified)
```

Once again producing a tibble containing the test statistic and a logical statement. Notice, however, the logical statement reading FALSE, meaning that the stratified t-test has not been used.

```
# A tibble: 1 x 2
  t_stat stratified
   <dbl> <lgl>
1 0.0506 FALSE
```

Now, the only thing left to do is put everything together in a function. This will be done in Section 2.3.

2.3 Test Function

Finally, we turn to putting a t_test function together. In order to save space and paper, we will denote the methods used to calculate the tests as d and t, just as we have done in previous sections. Instead we will focus on the composition of the function itself.

We begin by specifying an if statement, that makes sure the function will only accept input data in the form of a tibble or a data frame. If the function is called on any other type of data, it will return a warning message:

```
t_test <- function(data){
  if(is.tibble(data) & is.data.frame(data)){ # If statement limiting data
  } else {
    print("Data input needs to be a tibble or a data frame")
  } # Closes first if statement
}   # Closes function
```

Moreover, we want the function to look for a column called 'strata', and if there is no such column in the data, want it to perform a classical t-test rather than a stratified one:

```
t_test <- function(data){
  if(is.tibble(data) & is.data.frame(data)){   # If statement limiting data
    if(any(colnames(data) == "strata")){       # Stratified test
      dstr <- d
      return(print.data.frame(dstr))
    } else {        # Classical test
      tcls <- t
      return(print.data.frame(tcls))
    }               # Closes test selection
  } else {
     print("Data input needs to be a tibble or a data frame")
  }                 # Closes first if statement
}                   # Closes function
```

Finally, we create a vector of data just to try whether the if statements work. We also test the function using the data set we have previously simulated, namely **`strat`** and **`test`**:

```
vilgot <- 1:500 # Sample vector
t_test(strat)   # Stratified test

    t_stat stratified
1 1.030414       TRUE

t_test(test)    # Classical test

      t_stat stratified
1 0.05057784      FALSE

t_test(vilgot)  # Test using vector input

[1] "Data input needs to be a tibble or a data frame"
```

So we see that the test function returns the values – or messages – we want it to. That concludes the writing of our function computing t-test, and thereby also this section of the paper.

3 Presidential Election

3.1 Data Preparation

In this task we were given three data files. We were asked to join them, clean them and then move on to some analysis of the data. We begin by reading the data files:

```
crime_data <- read_tsv("crime_data.tsv")
dictionary_county_facts <- read_csv("county_facts_dictionary.csv")
county_facts <- read_csv("county_facts.csv")
results <- read_csv("general_result.csv")
```

After loading the data to `R` we want to merge the data into one data set. Each row is supposed to contain observations of a specific county, coded by the so called fips-code. First of all, we have to remove some rows in the county facts file since they are summaries for each state and the whole US. After that, we start off by joining the result file and the county facts file.

```
results <- results %>%
arrange(combined_fips)                                              # Arranging the data by fips
colnames(results)[2] <- "fips"                                      # Changing column name
county_facts <- na.omit(county_facts, cols = "state_abbreviation") # Removing state summary rows
colnames(county_facts)[1] <- "fips"                                 # Assigning matching column names
county_facts$fips <- as.character(county_facts$fips)               # Need for same class to join
results$fips <- as.character(results$fips)
results_county <- full_join(county_facts, results)                 # Joining data
```

In the crime file the fips-codes are separated into two columns, one with the state-code and one with the county-code. This causes some problems. To get the correct fips-code, we add one or two zeros between the state- and county codes. For a county with county-code 1 and state-code 1, we want the fips-code to be 1001. We solve this with a for loop based on logical arguments. After uniting the two columns into one fips-code column, we can join the crime data file with the rest of the data. We decide not to include the code in our report, but it can be found in the attached .R file.

Since we are interested in the results of the Republican party, we remove the rows which have missing values regarding the Republican results. In doing so, we end up with 3112 observations.

```
sammanslaget <- na.omit(sammanslaget, cols = "per_gop_2016")
```

We also create a new variable, `gop_win`, which takes on the value 1 if the Republicans got more votes than the Democrats

```
sammanslaget <- sammanslaget                        %>%
mutate(gop_win = per_gop_2016 - per_dem_2016)    %>%  # Creating gop_win
select(gop_win, everything())

for (i in 1:nrow(sammanslaget)) { # gop_win is in first row
  if(sammanslaget[i, 1] > 0){      # Takes value 1 if GOP > Dems
    sammanslaget[i, 1] <- 1
  } else {
    sammanslaget[i, 1] <- 0        # Takes value 0 if Dems > GOP
  }
}
sammanslaget$gop_win <- as.factor(sammanslaget$gop_win) # gop_win as factor variable
```

3.2 Data analysis

In this section, we focus on exploratory data analysis. This consists mostly of data visualisation and providing summaries of our data sets.

Figure 1 shows the final result for the republican party explained by the median income. The best fit line shows that there is a negative relation between household income and support for the republican party. It seems republicans tend to have greater support in poor areas. Assuming wages tend to be higher in urban areas – or along the coasts of the mainland – this could suggest that a fair share of Trump's supporters are part of the rural population, seeing as the Republicans tend to do better in inland states.

Figure 2 shows a comparison of Republican election results over the past two elections. As can be seen, the GOP managed to attain a qualified majority in more counties in 2016 than in 2012, indicating that the result was indeed pleasing from Trump's point of view. However, at first glance it is hard to tell whether the lower frequencies in the interval stretching from from about 40% to around 60% of the vote are compensated by the greater frequencies at higher percentages.

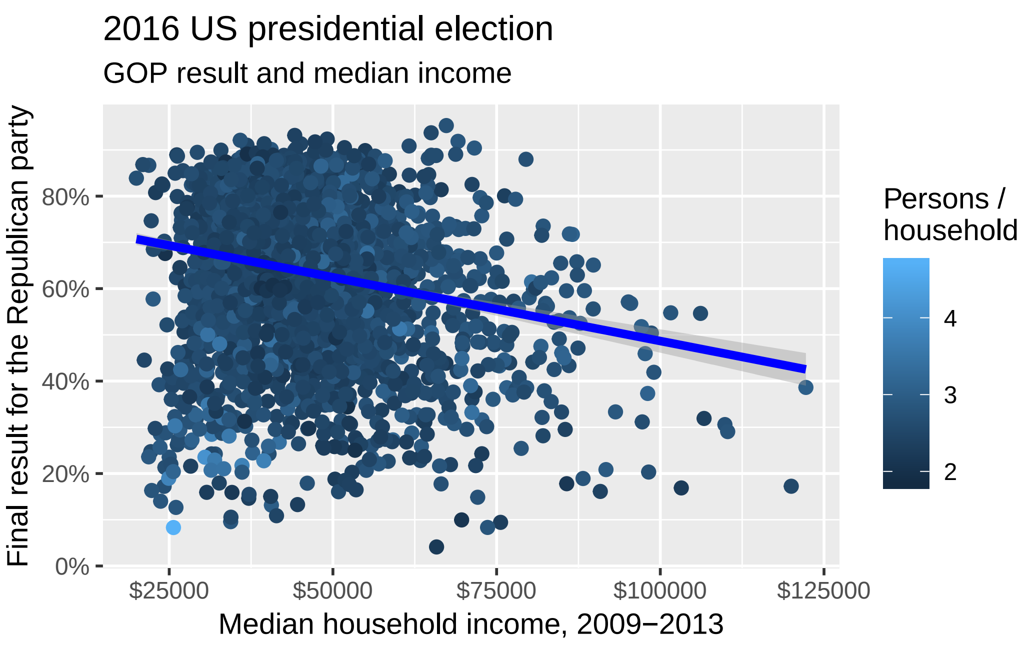

Figure 1: Scatterplot of republican election result and median income in that county

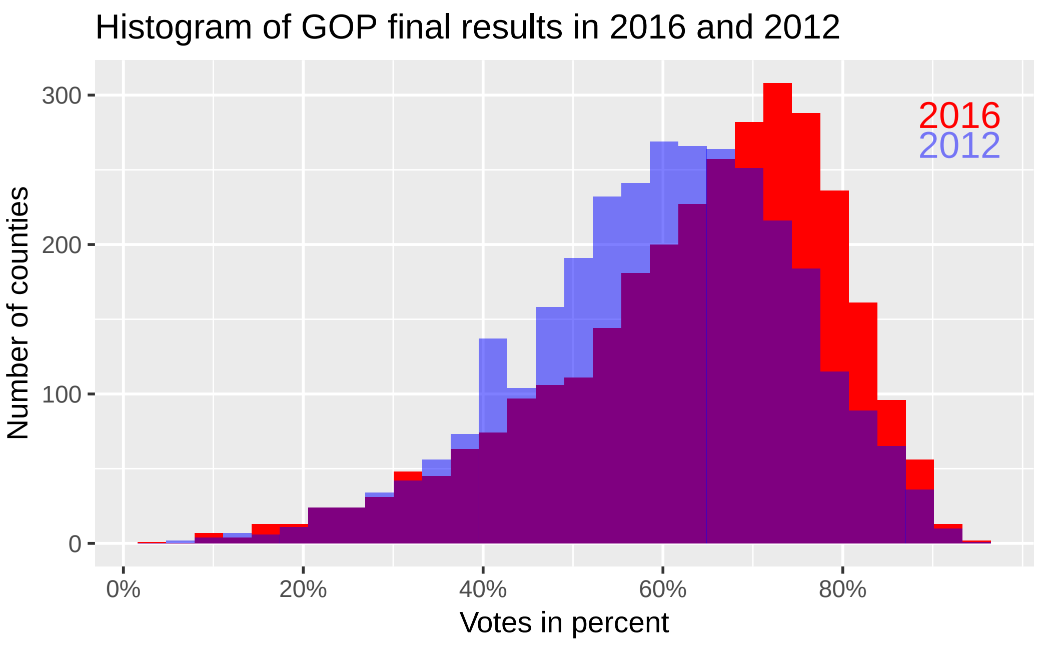

Figure 2: Histogram of republican elecetion result in 2016 and 2012

Figure 3 shows two boxplots, one for the level of education in counties won by the democrats and one for the level of education in counties won by the Republicans. We can see that the level of education tends to be higher in counties won by the democrats.

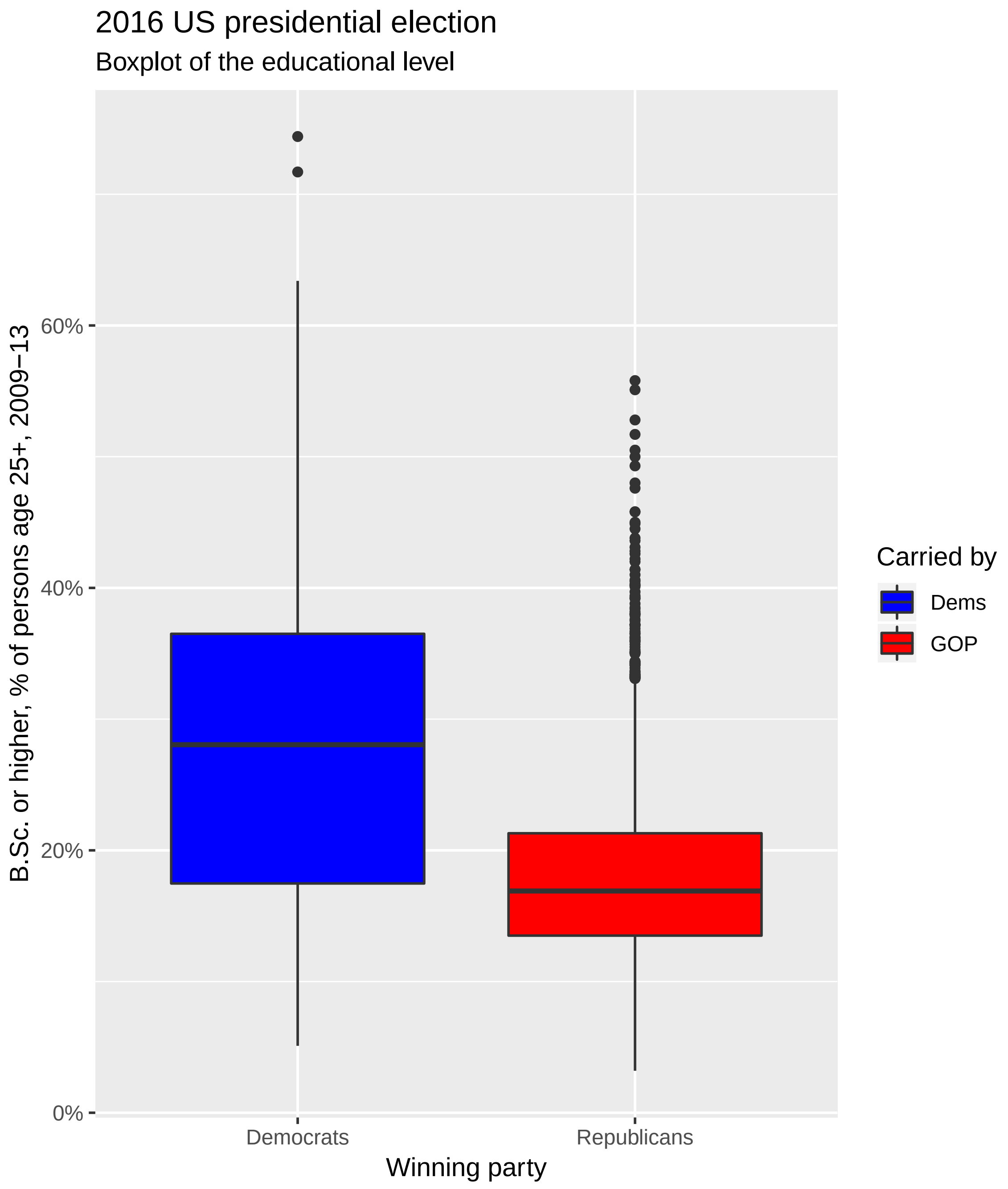

Figure 3: Boxplot of the level of education in counties won by each party.

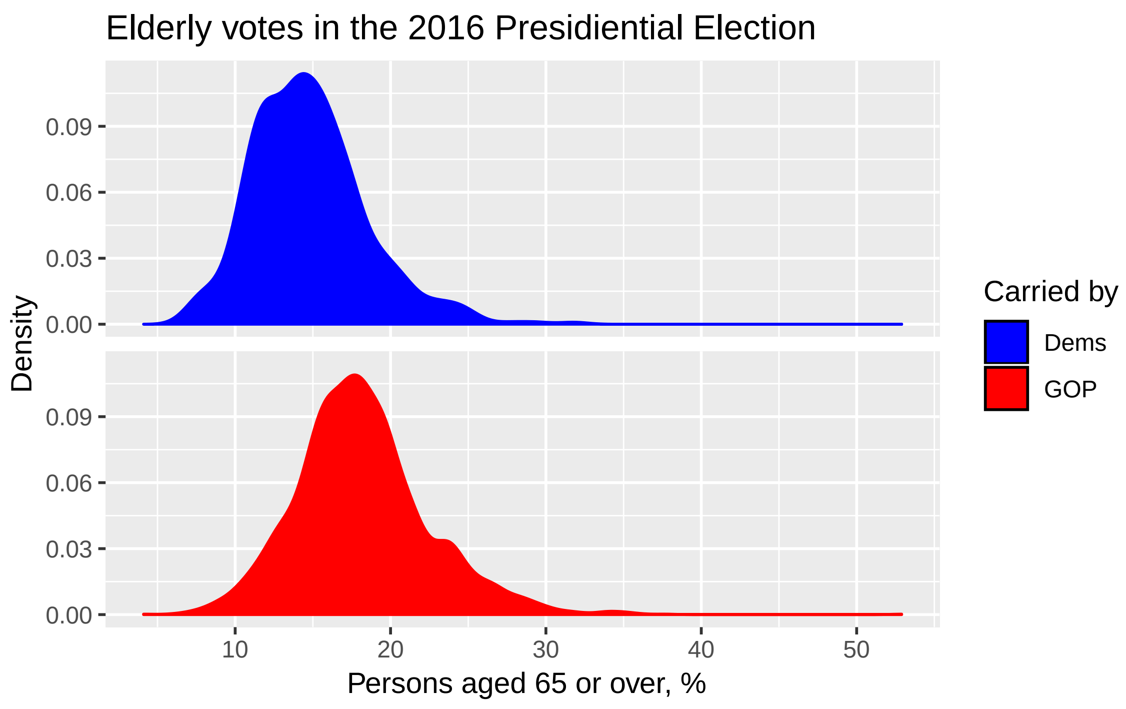

Figure 4: Distribution of the share of elderly, split by winning party

Figure 4 shows the distribution of the share of elderly people in counties won by each party. We can see that the GOP generally won in counties where elderly people had a bigger share of the total population, perhaps indicating to Trump that his make America great again slogan worked well among those who are old enough to remember the ”good old days”.

The regression line in Figure 5 shows a weak, negative relationship between the level of high school graduation and support for the Republican party. Moreover, it seems that counties that have a low rate of high school graduation also tend to have a larger number of people in each household.

This graph concludes the exploratory data analysis, and in the next couple of sections we will produce conditional and unconditional summaries of the data.

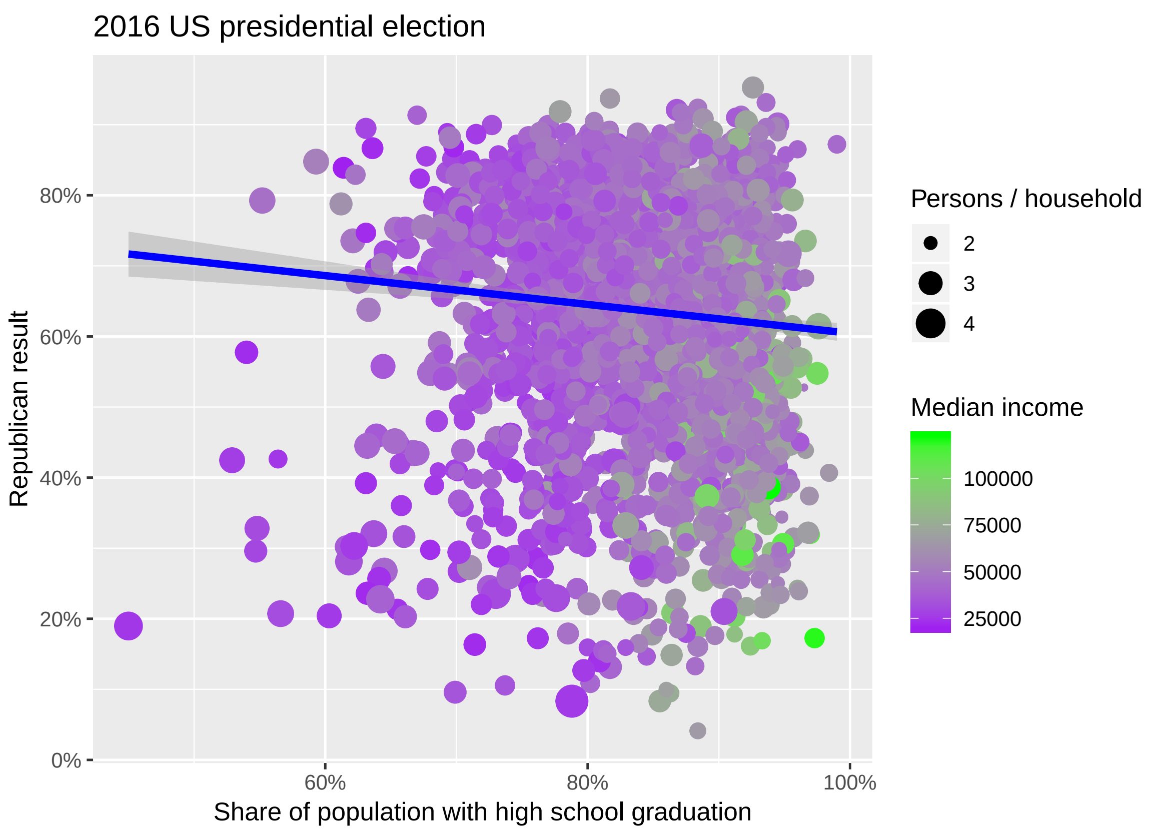

Figure 5: Scatterplot of republican result, educational level, income and persons per household

3.3 Unconditional Summaries

This section will provide three summaries in which the data has not been restricted by any conditions, meaning that there is no grouping or filtering whatsoever.

We begin with a table showing a summary of per capita money income over the past 12 months. All values are given in the 2013 value of the US Dollar, and the sample was taken during the period stretching from 2009 to 2013. We can see that the gap between the maximum and minimum incomes is strikingly big – almost $54,000 – which might indeed be the reason why Trump seemed to do well among poorer voters.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1	8768.00	19896.75	22889.00	23564.86	26187.25	62498.00

Table 1: Summary of the variable per capita income, dollars.

Table 2 shows a summary of the variable that counts the arrests and offenses related to murder. It shows that the median number of arrests or offences is 0, meaning that in at least half of the counties, no such event had occured during the period studied. Perhaps, this could serve as an indicator that Trump tended to overdo his rhetoric regarding the crime rates, and the safety of most Americans.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1	0.00	0.00	0.00	4.45	2.00	526.00

Table 2: Summary of the variable murder.

Table 3 provides a brief summary of the 2014 estimate of the population. We see that more than half of the counties have an estimated population of less than 26000, meaning that there is greater room for an election to be decided by a very narrow margin of votes in each county.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1	86.00	11157.75	25961.00	102223.73	68225.50	10116705.00

Table 3: Summary of the variable population.

3.4 Grouped Summaries

We beign the grouped summaries with a closer look at the differences between states carried by Republicans and states carried by Democrats. In Table 4, if GOP = 1, it means Trump carried a majority of the votes in more than 50% of the counties in a particular state.

	GOP	Robberies	Avg. robberies	Black Firms	Ppl per hshld	Share Veterans
1	0	251435	0.00079	8.672	2.614	0.06757
2	1	67756	0.00021	1.248	2.508	0.08373

Table 4: Variables grouped by 'carrying' party

The table shows that people living in states "carried" by Democrats run a greater risk of getting robbed, and that the states themselves have a higher rate of firms run by blacks, and a slightly higher number of people living in each household. However, states "carried" by the GOP have a greater share of veterans among their inhabitants.

In Figure 6 we have plotted the mean number of firms against the share of women in the population, grouped by state. This means that every point in the scatterplot represents an individual state, as is shown by the labels attached to each point. The graph suggests there is no relationship between the two variables.

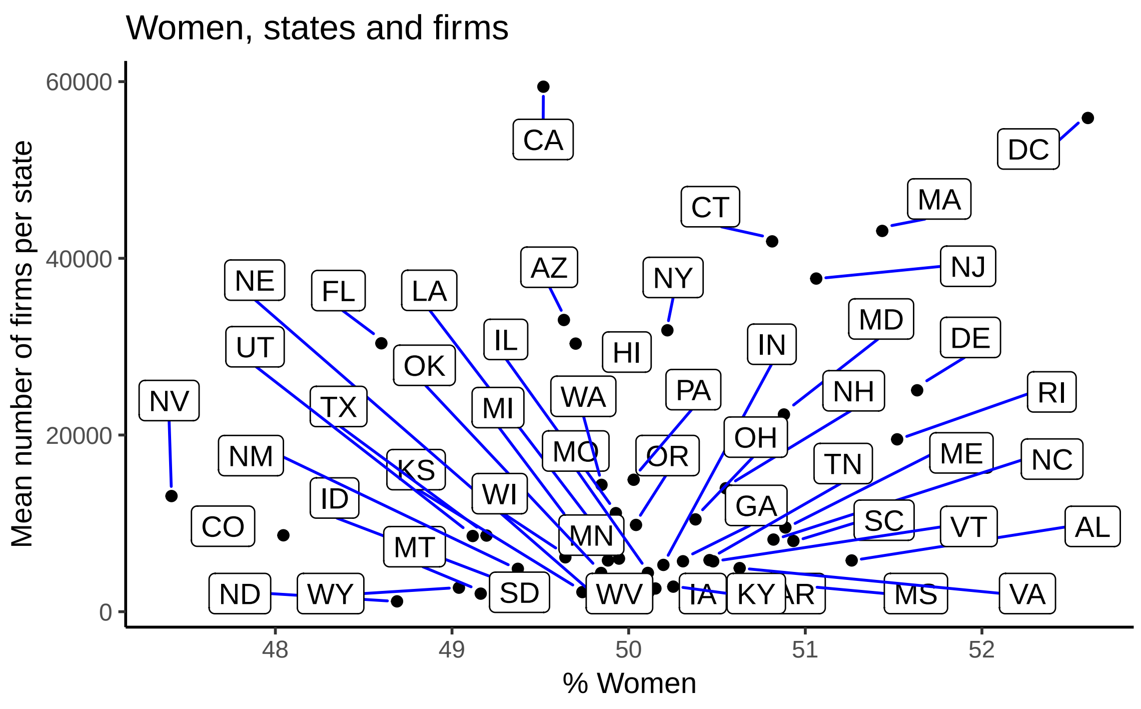

Figure 6: Pct women plotted against mean no. of firms, by state