POLS201 Spring 2019 Lab 3

Ian McDonald

April 12, 2019

Instructions for today's lab

Open the R_Lab_3.html file in your browser and open R_Lab_3.R script. Paste the content in the gray boxes in the R_Lab_3.html into the appropriate section of the R_Lab_3 script. Execute the code, review the output, and save the script.

A Note on Survey Weights

In this exercise we will be analyzing survey data. Generally when analyzing survey data you need to employ "weights" to correct for the fact that some groups are purposefully over-sampled and other groups may be unintentionally under sampled.

One of the DataCamp videos covers this topic.

We will not be doing that correction today. If you are doing survey data, make sure you have watched the video on sampling weights.

If you are using survey data in your paper, I'm not expecting you to apply weights for this class. But the veracity of your findings will ultimately depend on that. The Survey package in R simplifies that task, and if you are planning to adapt your project to a bigger scope (such as a capstone paper, expect to use weights).

Conceptually, weights are very simple: you use them to adjust individual observations, based on their attributes like race and gender, to their relative frequency in the population.

A Note on Summary Statistics

Recall that you have several R functions at your disposal to produce summary statistics. An excellent one is describe() from the Hmisc library. Others include stat_desc from the pastecs library. You can use these for entire tables, Another handy function is table(), which works for a single variable. You can express a single variable in this format: table_name$variable_name, separating the names with a "$" character.

1. Load the libraries

```
suppressMessages(library(Hmisc))
suppressMessages(library(tidyverse))
suppressMessages(library(haven))
suppressMessages(library(pastecs))
options(digits=4)
```

Why have I shown syntax like "Hmisc::describe" instead of just "describe"?

Remember we said that R in an "open source" language, and sometimes a publisher will create a library that reuses a function name that already exists elsewhere.

Some libraries like to use *describe()* and there is a way to manage the precedence of these different functions. If you know you want to use a function from a particular library and you want to make sure you use that function, you can explicitly state the library name in the function call.

describe(), unfortunately, is a popular function name. "*describe(table)*" will work perfectly if you haven't loaded a library after Hmisc with a describe function. So just in case, you can say *Hmisc::describe()*

Be sure to narrow down the number of variables before you run it. NES2012 starts with 1572 variables!

2. Load the data and make a smaller NES2012

```
NES2012 <- read_dta("ANES2012ftf.dta")
variables_to_select <- c("preknow_medicare", "ftgr_poor", "dem_raceeth", "ident_religid", "relig_import

NES2012_small <- NES2012 %>%
  select(variables_to_select)
```

There is no inherent reason why you have to create a variable like "variables_to_select"" before you run the *select()* function. In this case, it enhances readability, but readability lives in the eye of the beholder.

3. Run some summary stats on the thermometer ratings for "feelings about the poor"

```
Hmisc::describe(NES2012_small$ftgr_poor)
```

Looking at the output you will see that there are negative values. What is the lowest single value?

This isn't right. A feeling thermometer should only take the range of 0-100. In this dataset, people who refused to answer, didn't know, or weren't asked were coded as negative numbers. Let's change those to "missing" so they don't interfere with our analysis.

We can do this by creating a new variable and displaying the summary statistics for it.

```
NES2012_small <- NES2012_small %>%
  mutate(
    ftgr_poor_NM = ifelse(ftgr_poor < 0, NA, ftgr_poor)
    )

Hmisc::describe(NES2012_small$ftgr_poor_NM)
```

Note that you don't have to write this code on multiple lines. I choose to spread things out because I think that makes code more readable. Companies that use R extensively, like Google, publish style guides for those kinds of choices, but until you land that job at Google, you only need to please yourself.

Another useful format function is stat.desc from the pastecs package

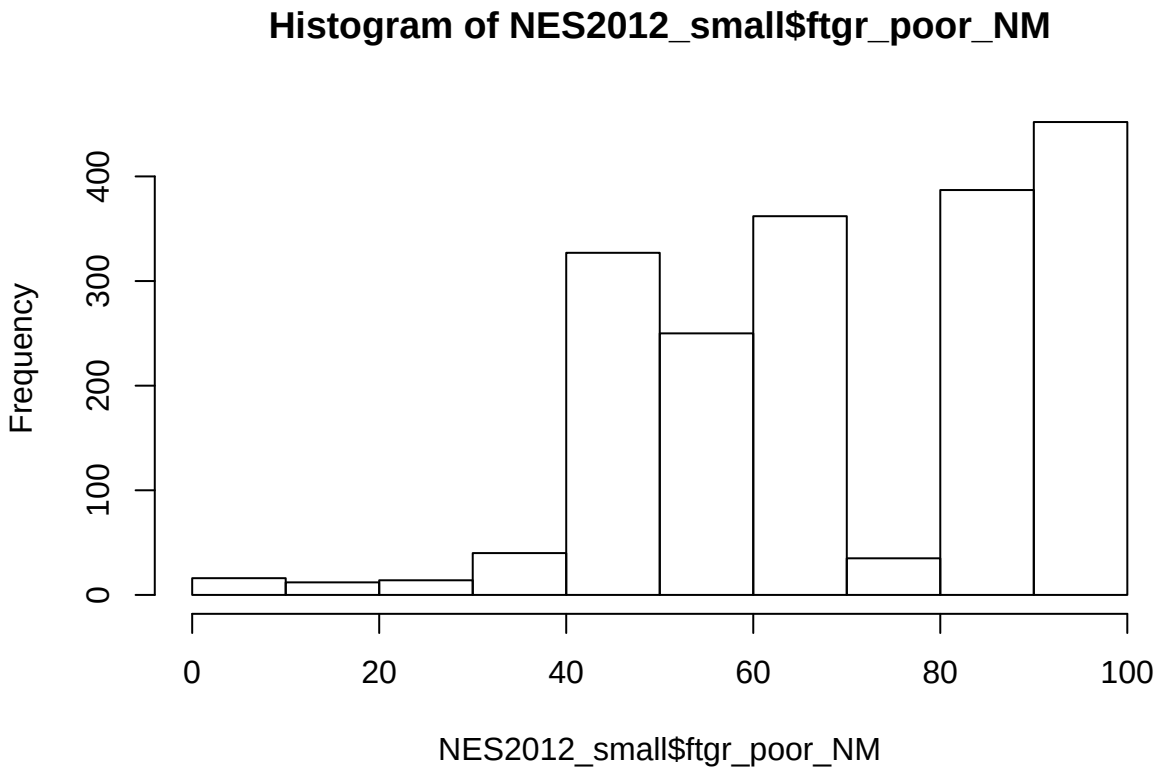
```
format(stat.desc(NES2012_small$ftgr_poor_NM), basic=TRUE, scientific=FALSE)
```

4. Make some histograms

We can visualize the frequency distribution of a single variable with a histogram. Here's a simple one and a slightly fancier one.

Here's a simple version:

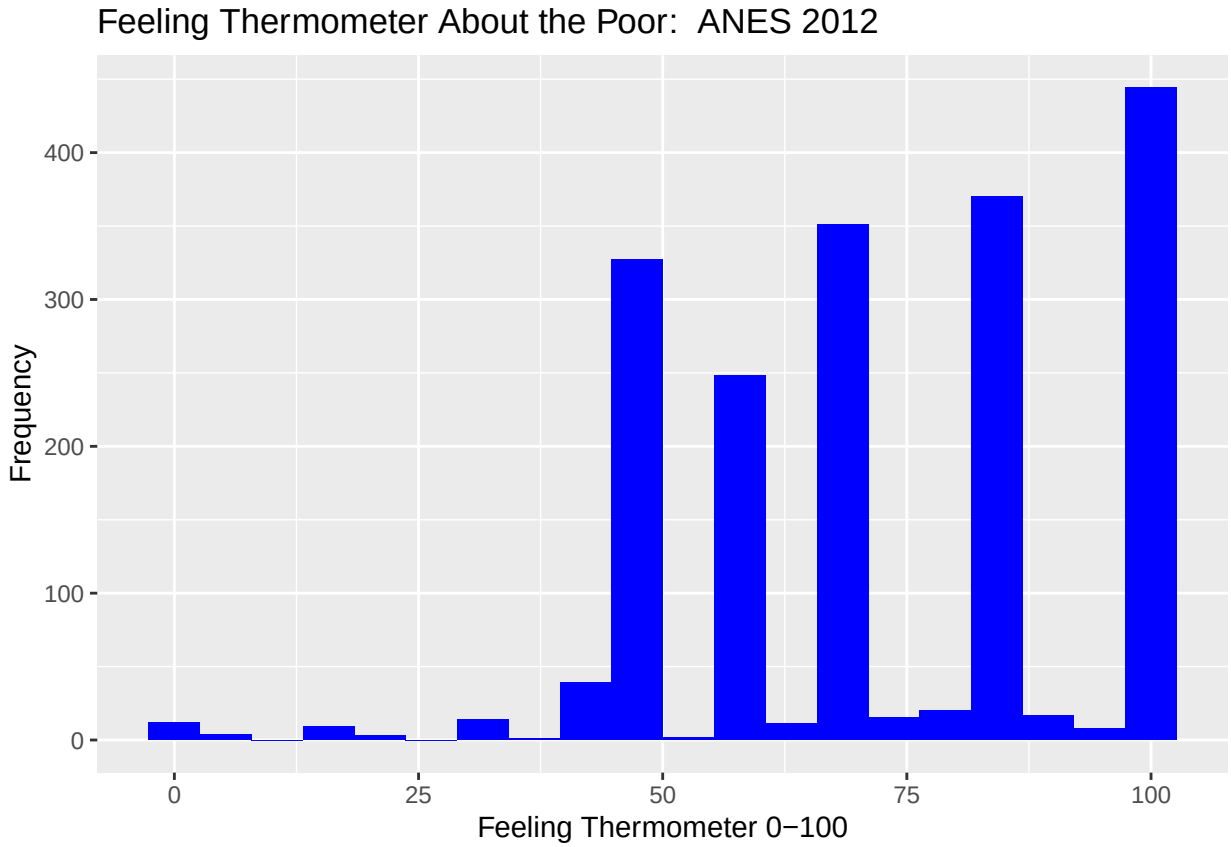
```
hist(NES2012_small$ftgr_poor_NM)
```



Here's a fancier one:

```
ggplot(NES2012_small, aes(x = ftgr_poor_NM)) + geom_histogram(fill = "blue", bins = 20) + ggtitle("Feeli
```

```
## Warning: Removed 159 rows containing non-finite values (stat_bin).
```



###5. Generate Frequency Distributions

A frequency distribution is a simple way to learn how often a variable takes on each of its possible values. This is best employed for discrete or categorical data. The tab command creates frequency distributions for single variables (one-way tables) or combinations of variables (two-way tables). It is recommended that frequencies be produced for all categorical variables for which you wish to recode, especially from a table like the one from the ANES2012ftf.dta file.

That will allow you to view how the data is coded and distributed. Once recoded, you should again use the *table()* function to make sure that the recode was done correctly.

Let's use this function to examine a variable that asks respondents a knowledge question about Medicare. The correct answer is 1 ("A program run by the U.S. federal government to pay for old people's health care")

```
table(NES2012_small$preknow_medicare)
```

The results show you exactly how this variable was coded and how many people fall into each category.

If we were to really analyze this data we would actually want to know who got this correct versus incorrect. Let's create a new variable that measures what we really care about.

```
NES2012_small <- NES2012_small %>%
  mutate(know_medicare_correct = case_when(
    preknow_medicare == -2 ~ NA_real_,
    preknow_medicare == 1 ~ 1,
    preknow_medicare >= 2 ~ 0
    ))
```

A step by step interpretation of this function would be:

1. Create a new version of NES2012 (<-) using the original NES2012.

2. In NES2012, create a new variable called *know_medicare_correct*.
3. Depending on the value of preknow_medicare, assign a value of 1, 0, or NA. We say NA_real_ to assign a numeric data type; otherwise R thinks we want the alphanumeric label "NA".
4. The statements in the _case_when*()_ function take this format:

   Condition ~ Value to Assign if Condition is met.

The operators used to determine a condition can take these forms (see your cheat sheet for a complete list):

- x == y means "x equals y", (notice the two equal signs, not just one)
- x != y means "x does not equal y",
- x > y "x is greater than y",
- x == y & x == z means "x equals y AND x equals z",
- x == y | x == z neabs "x equals y OR x equals z"

For the case_when function:

- TRUE ~ value assigns a value to observations that have not passed any of the criteria.
- The statements are evaluated in order, and assigns the value of the first match, so arrange them from the most specific to the most general.

Let's check to make sure this worked:

```
table(NES2012_small$know_medicare_correct)
```

6. Dummying Out Variables

Often, when using survey data, dummy variables are appropriate. An exception is a thermometer scale, but dummies are appropriate when you're using ordinal variables with no rational definition from one value to the next.

Let's create a new version of the NES2012_small table.,

```
Hmisc::describe(NES2012_small$dem_raceeth)
```

From the codebook, we can tell that the values of 1-4 are the important ones, and if we looked at the codebook we'd see: 1=White (non-hispanic), 2 Black (non-hispanic), 3 Hispanic and 4 Other non-hispanic. We need to change the -9 and -8 values to 4.

```
NES2012_small <- NES2012_small %>%
  mutate(race = case_when(
    dem_raceeth == (-9) ~ 4,
    dem_raceeth == (-8) ~ 4,
    TRUE ~ as.numeric(dem_raceeth)
    )
  )
```

The SDA service from UC Berkeley can give us this list if we don't have the codebook handy. See: http://sda.berkeley.edu/archive.htm

```
table(NES2012_small$race)
```

When a categorical variable like this one is used in a regression, we are likely to want to use it as a dummy. There are a couple of strategies for converting them. The most efficient way to to convert the variable to a data type called a factor. When R uses a factor in a regression, it automatically treats it as a dummy variable. To illustrate:

```
NES2012_small <- NES2012_small %>%
  mutate(race.f = as.factor(race), fake_dv = rnorm(nrow(NES2012_small)))
```

```
  summary(lm(fake_dv ~ race.f, data=NES2012_small))
```

Another variation is to simply declare the variable you want to "dummy" as a factor in your regression statement.

```
my_model <- lm(fake_dv ~ as.factor(race), data=NES2012_small)

summary(my_model)
```

This version works too.

```
summary(lm(fake_dv ~ as.factor(race), data=NES2012_small))
```

You can do it the hard way if you want. In this case, we'll also rename the variable:

```
NES2012_small <- NES2012_small %>%
  mutate(race_label = case_when(
    race == 1 ~ "White non-his",
    race == 2 ~ "Black non-his",
    race == 3 ~ "Hispanic",
    race == 4 ~ "Other non-hispanic"
  ))

NES2012_small <- NES2012_small %>%
    mutate(black_indicator = ifelse(race == 2, 1, 0)) %>%
    mutate(hispanic_indicator = ifelse(race == 3, 1, 0)) %>%
    mutate(other_indicator = ifelse(race == 4, 1, 0))

summary(lm(fake_dv ~ black_indicator + hispanic_indicator + other_indicator, data = NES2012_small))
```

You can also insert *as.factor* within the lm function

```
summary(lm(fake_dv ~ as.factor(race_label), data = NES2012_small))
```

7. More on linear regressions using the lm() function

You have now seen the basic format of a linear regression model in R. The *lm()* function creates an object in R, and you can them take this object and perform various reporting functions on it. The most useful is *summary()*. The basic syntax is:

object <- lm(dv ~ iv1 + iv2 + iv3 [and so on..], data = datatable_name)

And as you have seen, if you convert, say, iv1 to "as.factor(iv1) in the formula", lm() will convert iv(1) to a dummy. It will guess which value you want to "hold out". If you want to force a particular value to hold out, use the relevel() function which you can look up.

Here's a simple example. Let's take the ident_religid variable and see what it contains, which we reported earlier:

```
Hmisc::describe(NES2012_small$ident_religid)
```

Let's convert the values that show as less than zero to NA using NA_real_.

```
NES2012_small <- NES2012_small %>%
      mutate(ident_religid_NM = ifelse(ident_religid >=0, ident_religid, NA_real_))

Hmisc::describe(NES2012_small$ident_religid_NM)
```

Now we'll create a regression model and assign the name my_model.

```
my_model <- lm(ftgr_poor_NM ~ ident_religid_NM, data = NES2012_small)
```

Notice that we haven't yet displayed anything; we have stored the results into the object named my_model. We can display the summary with:

```
summary(my_model)
```

You can tell from the R-squared: this is not a very strong relationship. But it does generate a prediction.

We can also extract the coefficients with:

```
coef(my_model)
```

and the predicted values with predict(). Here, we'll just report the predicted values of the first ten observations.

```
head(predict(my_model), 10)
```

Do these predictions match our observed values?

```
head(NES2012_small$ftgr_poor_NM, 10)
```

```
head(residuals(my_model), 10)
```

We should expect to see the actual value of the DV, minus the predicted value, to equal the residual. This doesn't appear for obsevations with NA's, however. Do you see that?

For observation 2, the numbers works out to 70 - 76.84218 = -6.84218 And the residual is... -6.84218 as shown.

###8. Exercise with a political knowledge index

Let's tie this together by creating new variable that is similar to a variable Ben Gaskins created to measure "General Knowledge" in his paper "The Effects of Religious Commitment on Media Perception and the Acquisition of Political Information". In stata, use the search box in the "Variables" window to easily find each of the variables used to construct this index.

Consider these variables from NES2012:

preknow_prestimes: Do you happen to know how many times an individual can be elected President of the United States under current laws? [correct=2]

preknow_senterm: For how many years is a United States Senator elected that is, how many years are there in one full term of office for a U.S. Senator? [correct=6]

knowl_housemaj: Do you happen to know which party had the most members in the House of Representatives in Washington BEFORE the election [this/last] month? [correct=2]

knowl_senmaj: Do you happen to know which party had the most members in the U.S. Senate BEFORE the election [this/last] month? [correct=1]

candrel_dpc: Now we would like to ask you some questions about the religion of the presidential candidates. Would you say that Barack Obama is Protestant, Catholic, Jewish, Muslim, Mormon, some other religion, or is he not religious? [correct=1, though you should also allow for 10 ("Christian") which was volunteered by many respondents.

candrel_rpc: Would you say that Mitt Romney is Protestant, Catholic, Jewish, Muslim, Mormon, some other religion, or is he not religious? [correct=5]

Update this code to create a single variable that gives the sum of correct answers. You could create a set of variables with ones and zeroes, but if your code is organized, it's just as easy to do it with one statement.

```
Hmisc::describe(NES2012_small$relig_import)
```

Bother. More values that aren't 1 or 2 that need our attention.

```
NES2012_small <- NES2012_small %>%
  mutate(relig_import_NM = ifelse(relig_import > 0, relig_import, NA_real_)) %>%
  mutate(relig_import_NM = as.factor(relig_import_NM))

Hmisc::describe(NES2012_small$relig_import_NM)
```

Fixed. Now, we will add together respondents' successful answers to these questions

```
NES2012_small <- NES2012_small %>%

        mutate(answer_1 = ifelse(preknow_prestimes == 2, 1, 0)) %>%
        mutate(answer_2 = ifelse(preknow_senterm == 6, 1, 0)) %>%
        mutate(answer_3 = ifelse(knowl_housemaj == 2, 1, 0)) %>%
        mutate(answer_4 = ifelse(knowl_senmaj == 1, 1, 0)) %>%
        mutate(answer_5 = ifelse(candrel_dpc == 1 | candrel_dpc == 10, 1, 0)) %>%
        mutate(answer_6 = ifelse(candrel_rpc == 5 | candrel_rpc == 10, 1, 0)) %>%
        mutate(know_score = answer_1 + answer_2 + answer_3 + answer_4 + answer_5 + answer_6) %>%
        mutate(know_score_wt = know_score * weight_ftf)

Hmisc::describe(NES2012_small$know_score)
```

Does the t.test between *know_score* and *relig_import_NM* suggest that political knowledge changes with importance of religion among respondents?

We can use the t test in this way: if we compare the means of *relig_import_NM* with a value of 0 against those with a value of 1, we can test this null hypothesis: "the difference between the means of political knowledge between these two groups is zero."

So what do we see? Is zero within the 95 percent confidence interval?

```
t.test(know_score ~ relig_import_NM, data=NES2012_small)
```

9. Summarizing and Grouping

Often, we need to create subtotals of continuous variables (or means of others). Suppose, for example, we wish to compute sums and means of our know_score by state?

I can see that the state is stored inthe "sample_state" variable.

Here, I'm going to group the know_scores by state and compute a mean. We'll sort them and see who wins the title of "smartest state". Does the answer surprise you? Where is your state? And can you see why weighting this survey is important?

```
NES2012_small %>%
  group_by(sample_state) %>%
  summarise(state_smarts = mean(know_score), N = n()) %>%
  arrange(desc(state_smarts))
```

What if we reorder the states after adjusting for individual weights?

```
NES2012_small %>%
  group_by(sample_state) %>%
  summarise(state_smarts = mean(know_score),state_smarts_wt = mean(know_score_wt), N = n()) %>%
  arrange(desc(state_smarts_wt))
```

###10. For next Wednesday, we will use the UCLA Logit regression exercise found at: https://stats.idre.ucla.edu/r/dae/logit-regression/