Lab 9 - Data Transformation

George Rhodes

October 29, 2017

```
#library packages!!!
library("dplyr")
```

```
## Warning: package 'dplyr' was built under R version 3.4.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("tidyr")
```

```
## Warning: package 'tidyr' was built under R version 3.4.2
```

```
# Read in your data with the appropriate function
load("/Users/george/Documents/School/UW/SOC321/honors_thesis/honors_thesis/NSDUH-2015-survey-data.rda")
  names(PUF2015_102016) <- tolower(names(PUF2015_102016))

#subset data to relevant variables
subset_nsduh2015 <- PUF2015_102016 %>%
  select(txevrrcvd, alclottm, sexage, newrace2, sexrace, eduhighcat, ireduhighst2, al30est, alcus30d, a
  # replace with variable's you wish to add

#recovered_respondants set to those that have recieved treatment AND not drank in last 12 months.
recovered_respondants <- subset_nsduh2015 %>%
  filter(txevrrcvd == 1 & alclottm == 93)
  # 925 observations.
```

1. In addition to simply naming variable names in select you can also use `:` to select a range of variables and `-` to exclude some variables, similar to indexing a `data.frame` with square brackets. You can use both variable's names as well as integer indexes.

a. Use `select()` to print out a tbl that contains only the first 3 columns of your dataset, called by name.
b. Print out a `tbl` with the last 3 columns of your dataset, called by name.
c. Find the most concise way to select the first 3 columns and the last 3 columns by name.

```
#1.a
first_three_columns <- recovered_respondants %>%
  select(txevrrcvd, alclottm, sexage)
first_three_columns[1:10,]
```

```
##    txevrrcvd alclottm sexage
## 1          1       93      5
## 2          1       93      2
## 3          1       93      5
## 4          1       93      5
## 5          1       93      3
## 6          1       93      5
## 7          1       93      5
## 8          1       93      5
## 9          1       93      4
## 10         1       93      5
```

```
#1.b
last_three_columns <- recovered_respondants %>%
  select(coutyp2, alcwd2sx, alcemopb)
last_three_columns[1:10,]
```

```
##    coutyp2 alcwd2sx alcemopb
## 1        2       93       93
## 2        1       93       93
## 3        3       93       93
## 4        3       93       93
## 5        3       93       93
## 6        2       93       93
## 7        1       93       93
## 8        1       93       93
## 9        1       93       93
## 10       2       93       93
```

```
#1.c
three_and_three <- recovered_respondants %>%
  select(1:3,14:16)
three_and_three[1:10,]
```

```
##    txevrrcvd alclottm sexage coutyp2 alcwd2sx alcemopb
## 1          1       93      5       2       93       93
## 2          1       93      2       1       93       93
## 3          1       93      5       3       93       93
## 4          1       93      5       3       93       93
## 5          1       93      3       3       93       93
## 6          1       93      5       2       93       93
## 7          1       93      5       1       93       93
## 8          1       93      5       1       93       93
## 9          1       93      4       1       93       93
## 10         1       93      5       2       93       93
```

2. `dplyr` comes with a set of helper functions that can help you select groups of variables inside a `select()` call:

- `starts_with("X")`: every name that starts with "X",
- `ends_with("X")`: every name that ends with "X",
- `contains("X")`: every name that contains "X",
- `matches("X")`: every name that matches "X", where "X" can be a regular expression,
- `num_range("x", 1:5)`: the variables named x01, x02, x03, x04 and x05,
- `one_of(x)`: every name that appears in x, which should be a character vector.

Pay attention here: When you refer to columns directly inside `select()`, you don't use quotes. If you use the helper functions, you do use quotes.

a. Use `select()` and a helper function to print out a `tbl` that selects only variables that contain a specific character string.

```r
edu_variables <- recovered_respondants %>%
  select(contains("edu"))
edu_variables[1:10,]
```

```
##    eduhighcat ireduhighst2
## 1           3            9
## 2           5            7
## 3           1            5
## 4           1            7
## 5           3            9
## 6           2            8
## 7           3            9
## 8           2            8
## 9           1            6
## 10          4           11
```

b. Use `select()` and a helper function to print out a `tbl` that selects only variables that start with a certain letter or string of letters.

```r
alc_variables <- recovered_respondants %>%
  select(starts_with("alc"))
alc_variables[1:10,]
```

```
##    alclottm alcus30d alcbng30d alcwd2sx alcemopb
## 1        93        1         0       93       93
## 2        93      993        93       93       93
## 3        93      993        93       93       93
## 4        93      993        93       93       93
## 5        93      993        93       93       93
## 6        93        6         2       93       93
## 7        93        3         0       93       93
## 8        93      993        93       93       93
## 9        93      993        93       93       93
## 10       93      993        93       93       93
```

```r
income_variables <- recovered_respondants %>%
  select(11:12)

poverty_variables <- recovered_respondants %>%
  select(13:14)

demog_variables <- recovered_respondants %>%
  select(3:5)
```

4. Are there any mutations you wish to carry out on your data (i.e. new variables you wish to create based upon the values of already existing variables)? If so, describe what they are and what you will name them.

I do not think so. Most of the work will be recoding the survey codes into relevant info.

5. You can use mutate() to add multiple variables at once. To create more than one variable, place a comma between each variable that you define inside `mutate()`.

a. Carry out any and all of the mutations you wish to perform on your dataset and print the results to the console.

I don't think this applies to my data set.

6. `R` comes with a set of logical operators that you can use inside `filter()`:

- `x < y`, `TRUE` if `x` is less than `y`
- `x <= y`, `TRUE` if `x` is less than or equal to `y`
- `x == y`, `TRUE` if `x` equals `y`
- `x != y`, `TRUE` if `x` does not equal `y`
- `x >= y`, `TRUE` if `x` is greater than or equal to `y`
- `x > y`, `TRUE` if `x` is greater than `y`
- `x %in% c(a, b, c)`, `TRUE` if `x` is in the vector `c(a, b, c)`

a. What are some potential subsets of your data that seem interesting and worth investigation to you?
b. Use at least two of the logical operators presented above to print these subsets of your data.

7. `R` also comes with a set of boolean operators that you can use to combine multiple logical tests into a single test. These include `&` (and), `|` (or), and `!` (not). Instead of using the `&` operator, you can also pass several logical tests to `filter()`, separated by commas. `is.na()` will also come in handy.

a. Use `R`'s logical and boolean operators to select just the rows in your data that meet a specific boolean condition.
b. Print out all of the observations in your data in which none of variables are `NA`.

I did this at the beginning of the lab.

recovered_respondants <- subset_nsduh2015 %>% filter(txevrrcvd == 1 & alclottm == 93) recovered_respondants

8. `arrange()` can be used to rearrange rows according to any type of data. If you pass `arrange()` a character variable, for example, `R` will rearrange the rows in alphabetical order according to values of the variable. If you pass a factor variable, `R` will rearrange the rows according to the order of the levels in your factor (running `levels()` on the variable reveals this order).

By default, `arrange()` arranges the rows from smallest to largest. Rows with the smallest value of the variable will appear at the top of the data set. You can reverse this behavior with the `desc()` function. `arrange()` will reorder the rows from largest to smallest values of a variable if you wrap the variable name in `desc()` before passing it to `arrange()`.

a. Which variable(s) in your dataset would be logical to arrange your data on? Explain your reasoning.

   Arranging data on level of education or income would to understand how those who recieved treatment and are no longer drinking (recovered_respondants) vary by SES.

b. Arrange your data by this/these variables and print the results.

```r
by_family_income <- arrange(recovered_respondants, irfamin3)
by_family_income[1:10,]
```

```
##    txevrrcvd alclottm sexage newrace2 sexrace eduhighcat ireduhighst2
## 1          1       93      5        1       1          2            8
## 2          1       93      5        3       7          1            6
## 3          1       93      5        1       2          1            5
## 4          1       93      5        1       2          2            8
## 5          1       93      5        2       3          1            6
## 6          1       93      3        1       1          2            8
## 7          1       93      5        1       1          3            9
## 8          1       93      5        1       2          2            8
## 9          1       93      5        6       7          2            8
## 10         1       93      3        7       5          1            7
##    al30est alcus30d alcbng30d irpinc3 irfamin3 poverty3 coutyp2 alcwd2sx
## 1        93      993         93       1        1        1       3        93
## 2        93      993         93       1        1        1       3        93
## 3        93      993         93       1        1        1       3        93
## 4        93      993         93       1        1        1       3        93
## 5        93      993         93       1        1        1       2        93
## 6        93      993         93       1        1        1       1        93
## 7        93      993         93       1        1        1       3        93
## 8        93      993         93       1        1        1       2        93
## 9        93      993         93       1        1        1       1        93
## 10       93      993         93       1        1        1       1        93
##    alcemopb
## 1        93
## 2        93
## 3        93
## 4        93
## 5        93
## 6        93
## 7        93
## 8        93
## 9        93
## 10       93
```

```
by_edu <- arrange(recovered_respondants, eduhighcat)
by_edu[1:10,]
```

```
##    txevrrcvd alclottm sexage newrace2 sexrace eduhighcat ireduhighst2
## 1          1       93      5        1       1          1            5
## 2          1       93      5        1       1          1            7
## 3          1       93      4        2       4          1            6
## 4          1       93      5        7       5          1            6
## 5          1       93      5        1       1          1            4
## 6          1       93      5        3       7          1            6
## 7          1       93      5        1       1          1            7
## 8          1       93      5        1       2          1            5
## 9          1       93      3        7       5          1            3
## 10         1       93      5        2       3          1            6
##    al30est alcus30d alcbng30d irpinc3 irfamin3 poverty3 coutyp2 alcwd2sx
## 1       93      993        93       2        3        2       3       93
## 2       93      993        93       5        7        3       3       93
## 3       93      993        93       1        2        1       1       93
## 4       93      993        93       3        5        3       1       93
## 5       93      993        93       4        7        3       3       93
## 6       93      993        93       1        1        1       3       93
## 7       93      993        93       2        2        1       3       93
## 8       93      993        93       1        1        1       3       93
## 9       93      993        93       2        2        1       1       93
## 10      93      993        93       1        1        1       2       93
##    alcemopb
## 1        93
## 2        93
## 3        93
## 4        93
## 5        93
## 6        93
## 7        93
## 8         93
## 9         93
## 10        93
```

9. You can use any function you like in `summarise()` so long as the function can take a vector of data and return a single number. `R` contains many aggregating functions, as `dplyr` calls them:

- `min(x)` - minimum value of vector `x`.
- `max(x)` - maximum value of vector `x`.
- `mean(x)` - mean value of vector `x`.
- `median(x)` - median value of vector `x`.
- `quantile(x, p)` - pth quantile of vector `x`.
- `sd(x)` - standard deviation of vector `x`.
- `var(x)` - variance of vector `x`.
- `IQR(x)` - Inter Quartile Range (IQR) of vector `x`.
- `diff(range(x))` - total range of vector `x`.

a. Pick at least one variable of interest to your project analysis.
b. Print out at least three summary statistics using `summarise()`.

```
 fincome_summary <- income_variables %>%
    summarize(min_fincome = min(irfamin3),
    max_fincome = max(irfamin3),
    mean_fincome = mean(irfamin3),
    median_fincome = median(irfamin3))

income_summary <- income_variables %>%
    summarize(min_income = min(irpinc3),
    max_income = max(irpinc3),
    mean_income = mean(irpinc3),
    median_income = median(irpinc3))

edu_recode_summary <- edu_variables %>%
    summarize(min_edu = min(eduhighcat),
            max_edu = max(eduhighcat),
            mean_edu = mean(eduhighcat),
            median_edu = median(eduhighcat))

edu_recode_summary
```

```
##   min_edu max_edu mean_edu median_edu
## 1       1       5 2.568649          2
```

10. `dplyr` provides several helpful aggregate functions of its own, in addition to the ones that are already defined in `R`. These include:

- `first(x)` - The first element of vector `x`.
- `last(x)` - The last element of vector `x`.
- `nth(x, n)` - The nth element of vector `x`.
- `n()` - The number of rows in the data.frame or group of observations that `summarise()` describes.
- `n_distinct(x)` - The number of unique values in vector `x`.

Next to these `dplyr`-specific functions, you can also turn a logical test into an aggregating function with `sum()` or `mean()`. A logical test returns a vector of TRUE's and FALSE's. When you apply `sum()` or `mean()` to such a vector, `R` coerces each `TRUE` to a `1` and each `FALSE` to a `0`. `sum()` then represents the total number of observations that passed the test; `mean()` represents the proportion.

a. Print out a summary of your data using at least two of these `dplyr`-specific aggregate functions.

```r
edu_recode_summary <- edu_variables %>%
    summarize(min_edu = min(eduhighcat),
            max_edu = max(eduhighcat),
            mean_edu = mean(eduhighcat),
            median_edu = median(eduhighcat),
            last_edu = last(eduhighcat),
            number_observations = n(),
            nth_edu = nth(eduhighcat, 543),
            distinct_options_edu = n_distinct(eduhighcat))
```

b. Why did you choose the ones you did? What did you learn about your data from these summaries? Chose mostly by random for practice. last() will give the last observation's variable value. nth() will give the 543rd observation's variable value. n_distinct() will return how many options were anwswered in this survey for this variable.

11. You can also combine `group_by()` with `mutate()`. When you mutate grouped data, `mutate()` will calculate the new variables independently for each group. This is particularly useful when `mutate()` uses the `rank()` function, that calculates within-group rankings. `rank()` takes a group of values and calculates the rank of each value within the group, e.g.

```
rank(c(21, 22, 24, 23))
```

has the output

```
[1] 1 2 4 3
```

As with `arrange()`, `rank()` ranks values from the smallest to the largest.

a. Using the `%>%` operator, first group your dataset by a meaningful variable, then perform a mutation that you're interested in.

Still struggling with mutate.

arranged_mutated_meaningless <- recovered_respondants %>% rank(c(1:10)) %>% mutate(edu2 = eduhighcat * 2, fam2 = irfamin3 *2)

b. What do the results tell you about different groups in you data?

I was unable to run this code. Goal was to rank the first 10 reovered respondants, then mutate that ranked group to output 2 * the education level and 2* the family income as edu2 and fam2 respectively.

12. The exercises so far have tried to get you to think about how to apply the five verbs of `dplyr` to your data.

a. Are there any specific transformations you want to make to your data? What are they and what aspect of your research question will they help you to answer?

Specifically, I want to filter for respondants that have recieved treatment at some point and are currently sober. From there, I want to compare levels of education, income, and rural/urban. I will also want to compare and/or control for demographics. I don't think there are any transformations that I need to perform. Mostly I just need to work backwards through the codebook to make the data meaningful again.

b. In a code chunk below, carry out all the data transformations you wish to perform on your data. Utilize the `%>%` operator to tie multiple commands together and make your code more readable and efficient. Remember to comment your code so it is clear why you doing things a certain way.

recovered_respondants %>% select(contains("edu")) %>%

```r
# Read in your data with the appropriate function
load("/Users/george/Documents/School/UW/SOC321/honors_thesis/honors_thesis/NSDUH-2015-survey-data.rda")
  names(PUF2015_102016) <- tolower(names(PUF2015_102016))
```

```
#subset data to relevant variables
subset_nsduh2015 <- PUF2015_102016 %>%
  select(txevrrcvd, alclottm, sexage, newrace2, sexrace, eduhighcat, ireduhighst2, al30est, alcus30d, al
  # replace with variable's you wish to add

#recovered_respondants set to those that have recieved treatment AND not drank in last 12 months.
recovered_respondants <- subset_nsduh2015 %>%   #57146 observations of 16 varirables
 filter(txevrrcvd == 1 & alclottm == 93)  # 925 observations.
```

Not sure how to add more lines with pipe operator. income_variables <- recovered_respondants %>% select(11:12)

poverty_variables <- recovered_respondants %>% select(13:14)

demog_variables <- recovered_respondants %>% select(3:5)