Lab 9 - Data Transformation

Ignacio Cabezudo

October 29, 2017

```
library(dplyr)
library(readr)

###Vars
gtd_filter <- c("iyear", "imonth", "country_txt", "latitude", "longitude","nkill")

first3 <- c("iyear", "imonth", "country_txt") #make a list of the first 3 vars
last3 <- c("latitude", "longitude", "nkill") #last 3 vars
```

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.

```
#Load GTD Database - We'll prep the data to be cleaned here

  gtd_full <- read_csv("GTD FULL DB.csv") #load the database

#Grab all successful attacks that resulted in at least one death and apply our custom filter (gtd_filte

  boko <- gtd_full %>%
  filter(gname == "Boko Haram" & success == 1 & nkill > 0) %>%
  select(gtd_filter)

  select(boko, first3) #Using select, call the list of the of the first 3
```

```
## # A tibble: 1,517 x 3
##    iyear imonth country_txt
##    <int>  <int>       <chr>
##  1  2009      7     Nigeria
##  2  2009      7     Nigeria
##  3  2009      7     Nigeria
##  4  2009      7     Nigeria
##  5  2010      9     Nigeria
##  6  2010      9     Nigeria
##  7  2010     10     Nigeria
##  8  2010     10     Nigeria
##  9  2010     10     Nigeria
## 10  2010     10     Nigeria
## # ... with 1,507 more rows
```

```
  select(boko, last3)  #second 3
```

```
## # A tibble: 1,517 x 3
##    latitude longitude nkill
##       <dbl>     <dbl> <int>
##  1 11.83333 13.150000    77
##  2 11.83333 13.150000    76
##  3 11.83333 13.150000    76
##  4 11.83333 13.150000    75
##  5 11.51889 13.684167     1
##  6 11.83425 13.080144     2
##  7 11.83333 13.150000     1
##  8 11.83333 13.150000     2
##  9 11.83333 13.150000     1
## 10 11.62740  9.959413     1
## # ... with 1,507 more rows
```

```
concise <- c(first3, last3) #To make this easier, merge the list into 1
select(boko, concise) #call the merged list.
```

```
## # A tibble: 1,517 x 6
##    iyear imonth country_txt latitude longitude nkill
##    <int>  <int>       <chr>    <dbl>     <dbl> <int>
##  1  2009      7     Nigeria 11.83333 13.150000    77
##  2  2009      7     Nigeria 11.83333 13.150000    76
##  3  2009      7     Nigeria 11.83333 13.150000    76
##  4  2009      7     Nigeria 11.83333 13.150000    75
##  5  2010      9     Nigeria 11.51889 13.684167     1
##  6  2010      9     Nigeria 11.83425 13.080144     2
##  7  2010     10     Nigeria 11.83333 13.150000     1
##  8  2010     10     Nigeria 11.83333 13.150000     2
##  9  2010     10     Nigeria 11.83333 13.150000     1
## 10  2010     10     Nigeria 11.62740  9.959413     1
## # ... with 1,507 more rows
```

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.
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.

```
select(gtd_full, contains("weapdet"))
```

```
## # A tibble: 170,350 x 1
##                                               weapdetail
##                                                    <chr>
##  1                                                  <NA>
##  2                                                  <NA>
##  3                                                  <NA>
##  4                                             Explosive
## 5                                            Incendiary
## 6                              Several gunshots were fired.
## 7                                     Automatic firearm
## 8                                                  <NA>
## 9                           Firebomb consisting of gasoline
## 10 Poured gasoline on the floor and lit it with a match
## # ... with 170,340 more rows
```

```
select(gtd_full, starts_with("cou"))
```

```
## # A tibble: 170,350 x 2
##    country        country_txt
##      <int>              <chr>
##  1      58 Dominican Republic
##  2     130             Mexico
##  3     160        Philippines
##  4      78             Greece
##  5     101              Japan
##  6     217      United States
##  7     218            Uruguay
##  8     217      United States
##  9     217      United States
## 10     217      United States
## # ... with 170,340 more rows
```

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.

Not currently applicable

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()`.

I don't think I need to use this.

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

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?

I picked my data because it was already a very narrowed down set. I used nkill and imonth to look into how many events occured in which at least 10 people were killed in the 1st quarter of the year.

b. Use at least two of the logical operators presented above to print these subsets of your data.

```
#Filter out only attacks in the 1st Quarter, which resulted in at least 10 deaths
  filter(boko, nkill >= 10 & imonth == 1 | imonth == 2 | imonth == 3)
```

```
## # A tibble: 296 x 6
##    iyear imonth country_txt latitude longitude nkill
##    <int>  <int>       <chr>    <dbl>     <dbl> <int>
##  1  2011      2     Nigeria 10.61110  12.19500     1
##  2  2011      2     Nigeria 11.83333  13.15000     2
##  3  2011      2     Nigeria 10.99642  10.40865     1
##  4  2011      2     Nigeria 10.99642  10.40865     1
##  5  2011      2     Nigeria 11.83333  13.15000     1
##  6  2011      3     Nigeria 11.83326  13.18665     1
##  7  2011      3     Nigeria 11.81793  13.06962     1
##  8  2011      3     Nigeria 11.83333  13.15000     1
##  9  2011      3     Nigeria 11.83333  13.15000     1
## 10  2011      3     Nigeria 11.83333  13.15000     1
## # ... with 286 more rows
```

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.

```
#Only attacks in Nigeria in 2009
  filter(boko, country_txt == "Nigeria" & iyear == "2009")
```

```
## # A tibble: 4 x 6
##   iyear imonth country_txt latitude longitude nkill
##   <int>  <int>       <chr>    <dbl>     <dbl> <int>
## 1  2009      7     Nigeria 11.83333     13.15    77
## 2  2009      7     Nigeria 11.83333     13.15    76
## 3  2009      7     Nigeria 11.83333     13.15    76
## 4  2009      7     Nigeria 11.83333     13.15    75
```

b. Print out all of the observations in your data in which none of variables are NA.

I am confused about how to apply this to my whole df. Need some explanation here. I've tried googling for answers, went through Stack, etc.

```
#print(!is.na(boko)
```

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.

Nkill, as that is my primary data point

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

```
  arrange(boko, imonth)
```

```
## # A tibble: 1,517 x 6
##    iyear imonth country_txt  latitude longitude nkill
##    <int>  <int>       <chr>     <dbl>     <dbl> <int>
## 1  2011      1     Nigeria 11.836123  13.17764     1
## 2  2011      1     Nigeria 10.283343  11.16663     8
## 3  2011      1     Nigeria 11.833333  13.15000     7
## 4  2012      1     Nigeria 12.786790  10.23771     1
## 5  2012      1     Nigeria 11.833333  13.15000     1
## 6  2012      1     Nigeria 11.748860  11.96603     1
## 7  2012      1     Nigeria 10.266667  13.26667     4
## 8  2012      1     Nigeria 10.282400  11.17479     6
## 9  2012      1     Nigeria 10.266667  13.26667    20
## 10  2012      1     Nigeria  9.203496  12.49539     8
## # ... with 1,507 more rows
```

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()`.

```
summarise(boko, mean(nkill))
```

```
## # A tibble: 1 x 1
##   `mean(nkill)`
##           <dbl>
## 1      11.92617
```

```
summarise(boko, median(nkill))
```

```
## # A tibble: 1 x 1
##   `median(nkill)`
##             <int>
## 1               5
```

```
summarise(boko, IQR(nkill))
```

```
## # A tibble: 1 x 1
##   `IQR(nkill)`
##          <dbl>
## 1           10
```

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.

```
#boko %>%
  #summarise(last(imonth)) %>%
  #summarise(first(imonth))

#Why doesn't the above work? Things work just fine when I don't pipe
#them, but as soon as I do just about everything I attempt to do fails.

  summarise(boko, last(imonth))
```

```
## # A tibble: 1 x 1
##   `last(imonth)`
##            <int>
## 1             12
```

```
  summarise(boko, first(imonth))
```

```
## # A tibble: 1 x 1
##   `first(imonth)`
##             <int>
## 1               7
```

b. Why did you choose the ones you did? What did you learn about your data from these summaries?

I don't think I have any data that is applicable to this particular question, so I just picked imonth.

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.

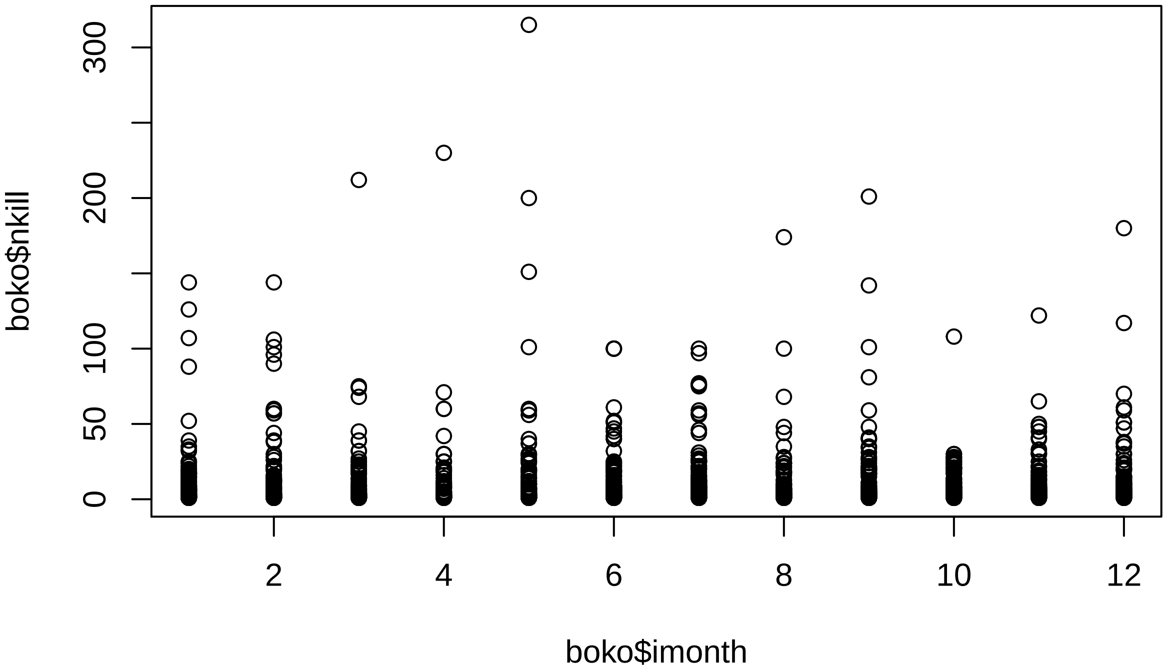
```
#Once again, I don't really know why I would use this for this data. Commented out as it returns 1500 r

#boko %>%
  #group_by(imonth) %>%
 # mutate(imonth = between(imonth, 1, 5))
```

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

Without a plot, not much, but if we were to plot the data, you can see that there is a trend of per-month deaths that stays relativly flat, with some outliers.

```
  plot(boko$imonth, boko$nkill)
```



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?

No. I'm honestly a bit overwhelmed with the amount of info in this lab, as most of it doesn't seem to apply what I'm doing. That makes it very difficult to learn how to use things as I'm so focused on getting the project done, that having to come up with uses of these just stresses me out. I'm worried that because of the complexity of the labs, with all the tools available, that my project is "too basic".

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.

I'm so confused by this. This is what I've been doing for the whole lab, is copying/pasting earlier stuff what I should do?

```
head(boko, 15)
```

```
## # A tibble: 15 x 6
##    iyear imonth country_txt  latitude longitude nkill
##    <int>  <int>       <chr>     <dbl>     <dbl> <int>
##  1  2009      7     Nigeria 11.833333 13.150000    77
##  2  2009      7     Nigeria 11.833333 13.150000    76
##  3  2009      7     Nigeria 11.833333 13.150000    76
##  4  2009      7     Nigeria 11.833333 13.150000    75
##  5  2010      9     Nigeria 11.518889 13.684167     1
##  6  2010      9     Nigeria 11.834254 13.080144     2
##  7  2010     10     Nigeria 11.833333 13.150000     1
##  8  2010     10     Nigeria 11.833333 13.150000     2
##  9  2010     10     Nigeria 11.833333 13.150000     1
## 10  2010     10     Nigeria 11.627401  9.959413     1
## 11  2010     11     Nigeria 11.833333 13.150000     3
## 12  2010     12     Nigeria 11.833333 13.150000     5
## 13  2010     12     Nigeria 11.833333 13.150000     1
## 14  2010     12     Nigeria  9.933333  8.883333    38
## 15  2010     12     Nigeria 11.869115 13.125143     3
```