clean_description

Katie Key

10/24/2017

Questions about the Article

1. In the in-course exercises, we have been analyzing data with accident as the observation unit. This study uses a different observation unit. What is the unit of observation in the Brady and Li study? When you download the FARS data for a year, you get a zipped folder with several different datasets. Which of the FARS datasets provides information at this observation level (and so will be the one you want to use for this analysis)?

The unit of observation in the Brady and Li study is presence of alcohol and/or non-alcohol drugs in drivers killed in a motor vehicle crash. To get the appropriate information at this observation level, we will want to use the PERSON data file from the FARS data for 1999 to 2010. From the PERSON data files, we will pull information on year, sex, age category, and drug type where the drug test was a positive result.

2. This study only analyzes a subset of the available FARS data. Enumerate all of the constraints that are used by the study to create the subset of data they use in their study (e.g., years, states, person type, injury type). Go through the FARS documentation and provide the variable names for the variables you will use to create filter statements in R to limit the data to this subset. Provide the values that you will want to keep from each variable.

The study looked at specific aspects of the FARS PERSON data files. The constraints that are used by the study to create the subset of data include sex, age, year, consecutive number, vehicle number, person number, state number, person type, lag hours and minutes, injury severity, alcohol test result, and drug test results. The variable names for the variables used to create filter statements are sex, age, st_case, veh_no, per_no, state, per_typ, lag_hrs, lag_mins, inj_sev, age, alc_res, drugres1, drugres2, and drugres3. Sex is expressed as male, female, unknown or not reported. Age is expressed in years. Year is based on the year of the data (1990-2010 for the study). For person type (per_typ), we want to keep only the people with 1, which is driver. For injury severity (in_sev), we want to only keep those with 4, which is fatal injury. For state, we want to select states 6, 15, 17, 33, 44, and 54, which are California, Hawaii, Illinois, New Hampshire, Rhode Island, and West Virginia, respectively. For alcohol test result (alc_res), we only want to keep results that are greater than or equal to a BAC of 0.01. It is important to note that for alc_res values, the decimal is implied. For the years we are interested in looking at, a BAC of 0.10 is coded as 10. For all the drug result variables (contains "drugres"), we will keep those classified as narcotic (100-295), depressant (300-395), stimulant (400-495), cannabinoid (600-695), and other (500-595,700-996).

3. The study gives results stratified by age category, year, year category (e.g., 1999–2002), alcohol level, non-alcohol drug category, and sex. For each of these stratifications, give the variable name for the FARS variable you could use to make the split (i.e., the column name you would use in a group by statement to create summaries within each category or the column name you would mutate to generate a categorical variable). Describe how each of these variables are coded in the data. Are there any values for missing data that you'll need to mutate to NA values in R? Are there any cases where coding has changed over the study period?

For 'age category', we would want to use the 'age' variable in the FARS data set, which is coded by age in years. The age categories we are going to make are 'less than 25', 'between 25 and 44', 'between 45 and 64', and '65 years or older'. For 'year', we would use the 'year' of the dataset since we would be pulling in each year (1999 through 2010) separately. For 'year category', we would group years into three-year groups. Our 'year category' groups would, therefore, be '1999-2002','2003-2006', and '2007-2010'. For 'alcohol level', we would be using the 'alc_res' variable from the FARS data set. This is examing the result of an alcohol test. For alcohol level, we would only want to include a BAC test result of 0.01 or greater. In the FARS data set,

alc_rest is expressed as an integer because the decimal is implied. For example, a blood alcohol concentration (BAC), which is expressed in grams per deciliter (g/dL), of 0.10 is coded as 10. Therefore, our 'alcohol level' variable would be pulling data for 0-94 from the 'alc_res' column of the FARS data. The 'non-drug category' is created by combining the 'drugres1', 'drugres2', and 'drugres3' variables from the FARS data. For these variables, drug results are coded using numbers to represent different drug types. Using the FARS data numbering system, 'narcotic' is 100-295; 'depressant' is 300-395; 'stimulant' is 400-495; 'cannabinoid' is 600-695; and 'other' is 500-595 as well as 700-996. The 'sex' variable is from the 'sex' column of the FARS data. The 'sex' column is also coded using numbers. It translates as 1 equal to males, 2 equal to females, 8 not reported, and 9 as unknown. For 'sex' values that are not 1 (males) or 2 (females), the value will be mutated to 'NA' in R. The same goes for 'alc_res' values that are not greater than or equal to a BAC of 0.01. For some of the variables, the coding changed over the study period. In 2010, 8 was added to 'sex' and 998 was added to 'age' to be equal to 'not reported'. Between 2008 and 2009, 'vehicle number', 'person number', and 'lag hours' went from being a two-digit code to being a three-digit code. For 'person type', 2010 had 88 equals 'not reported' but did not have 99 equals 'unknown', which is the opposite for data from 1999 to 2009. 'Alc_res' and the 'drug_res' columns also had some different coding for 2010 compared to years 1999 through 2009.

Cleaning Data Code

```
clean_yearly_person_file <- function(year) {
  # 1. Read data in.
  person_file <- paste0("data-raw/yearly_person_data/person_", year, ".csv")
  df <- readr::read_csv(person_file)
  # 2. Convert all column names to lowercase.
  colnames(df) <- tolower(colnames(df))
  df <- df %>%
    # 3. Limit variables.
    dplyr::select(st_case, veh_no, per_no, state, per_typ, lag_hrs, lag_mins,
                  inj_sev, age, alc_res, contains("drugres"), sex) %>%
    # 4. Limit to relevant `per_typ` and `inj_sev` values, then remove those variables.
    dplyr::filter(per_typ == 1 & inj_sev == 4) %>%
    dplyr::select(-per_typ, -inj_sev) %>%
    # 5. Create a `unique_id`. Note: to be unique, `year` needs to be pasted on.
    tidyr::unite(unique_id, st_case, veh_no, per_no) %>%
    dplyr::mutate(year = year,
                  unique_id = paste(unique_id, year, sep = "_")) %>%
    # 6. Limit to study states and then remove the `state` variable.
    dplyr::filter(state %in% c(6,
                               15,
                               17,
                               33,
                               44,
                               54)) %>%
    dplyr::select(-state) %>%
    # 7. Convert `sex` to a factor with levels "Male" and "Female".
    dplyr::mutate(sex = ifelse(sex == 9, NA, sex),
                  sex = factor(sex, levels = c(1, 2),
                               labels = c("Male", "Female"))) %>%
    # 8. Use measured alcohol blood level to create `Alcohol` (logical for whether
    # alcohol was present). Then remove the `alc_res` variable.
    dplyr::mutate(alc_res = ifelse(alc_res > 94, NA, alc_res / 10),
                  Alcohol = alc_res >= 0.01) %>%
    dplyr::select(-alc_res) %>%
```

```
    # 9. Specify missing values for the lag minutes.
    dplyr::mutate(lag_mins = ifelse(lag_mins == 99, NA, lag_mins))
  # 10. Save lag hours coded as missing as `NA`.
if(year <= 2008){
    df <- df %>%
      dplyr::mutate(lag_hrs = ifelse(lag_hrs %in% c(99, 999), NA, lag_hrs))
  } else {
    df <- df %>%
      dplyr::mutate(lag_hrs = ifelse(lag_hrs == 999, NA, lag_hrs))
  }

  # 11. Limit to deaths within an hour of the accident then remove those variables.
  df <- df %>%
    dplyr::filter((lag_hrs < 1) | (lag_hrs == 1 & lag_mins == 0)) %>%
    dplyr::select(-lag_hrs, -lag_mins)
  # 12. Save age values coded as missing as `NA`.
if(year <= 2008){
    df <- df %>%
      dplyr::mutate(age = ifelse(age == 99, NA, age))
  } else {
    df <- df %>%
      dplyr::mutate(age = ifelse(age %in% c(998, 999), NA, age))
  }
  # 13. Use age to create age categories and then remove `age` variable.
  df <- df %>%
    dplyr::mutate(agecat = cut(age, breaks = c(0, 25, 45, 65, 1000),
                               labels = c("< 25 years",
                                          "25--44 years",
                                          "45--64 years",
                                          "65 years +"),
                               include.lowest = TRUE, right = FALSE)) %>%
    dplyr::select(-age)
  # 14. Gather all the columns with different drug listings (i.e., `drugres1`,
  # `drugres2`, `drugres3`). Convert from the numeric code listings to
  # drug categories.
  gathered_df <- df %>%
    tidyr::gather(drug_number, drug_type_raw, contains("drugres")) %>%
    dplyr::mutate(drug_type = ifelse(drug_type_raw %in% 100:295,
                                     "Narcotic", NA),
                  drug_type = ifelse(drug_type_raw %in% 300:395,
                                     "Depressant", drug_type),
                  drug_type = ifelse(drug_type_raw %in% 400:495,
                                     "Stimulant", drug_type),
                  drug_type = ifelse(drug_type_raw %in% 600:695,
                                     "Cannabinoid", drug_type),
                  drug_type = ifelse(drug_type_raw %in% c(500:595, 700:996),
                                     "Other", drug_type),
                  drug_type = ifelse(drug_type_raw == 1,
                                     "None", drug_type),
                  drug_type = factor(drug_type)) %>%
    dplyr::select(-drug_type_raw, -drug_number) %>%
    # 15. Filter out any observations where both alcohol and drug data is missing.
    dplyr::filter(!(is.na(Alcohol) & is.na(drug_type)))
```

```
  # 16. Create a subset with only individuals with at least one non-missing
  # listing for drugs. (Write a sentence or two for each step in this pipe chain.)
  non_missing_drugs <- gathered_df %>%
    filter(!is.na(drug_type)) %>%
    group_by(unique_id, drug_type) %>%
    summarize(has_drug = TRUE) %>%
  ungroup() %>%
    mutate(row_num = 1:n()) %>%
    spread(drug_type, has_drug, fill = FALSE) %>%
    select(-row_num)
  # 17. Join this back into the full dataset. (Write a sentence or two for each
  # step in this pipe chain.)
  df <- df %>%
    dplyr::select(-contains("drugres")) %>%
    dplyr::full_join(non_missing_drugs, by = "unique_id") %>%
    dplyr::select(-None) %>%
    tidyr::gather(drug_type, positive_for_drug, Alcohol, Cannabinoid,
                  Depressant, Narcotic, Other, Stimulant) %>%
    dplyr::mutate(drug_type = factor(drug_type)) %>%
    unique()
  return(df)
}
```

NOTE: For step 18, the code does not work in its stated format for the R Markdown document because the R Markdown document exists in a different directory than the one used for the coding in the cleaning-data.R file. The clean_fars dataframe was saved in the 'data' folder of the fars analysis directory while this R Markdown is saved in the 'writing' folder. However, you can see the code worked in the next code chunk, which produces a table representing the cleaned data.

```
# 18. Iterate the clean_yearly_person_file function across study years to
# create and save a single dataset.
# Note: map_df() is similar to map(), but it binds elements of the
# list resulting from map() together. To understand this step, try
# running this code with map instead of map_df, check out documentation
# for map and map_df, and look at the map_df() function by typing
# `map_df` in your console.

# CODE:
# clean_fars <- map_df(1999:2010, clean_yearly_person_file)
# save(clean_fars, file = "data/clean_fars.RData")
```

```
load("../data/clean_fars.RData")

clean_fars %>%
  mutate(year_cat = cut(year, breaks = c(1999, 2002, 2006, 2010),
                        labels = c("1999-2002", "2003-2006",
                                   "2007-2010"),
                        include.lowest = TRUE, right = TRUE)) %>%
  filter(!is.na(sex)) %>%
  group_by(drug_type, sex, year_cat) %>%
  summarize(n_non_missing = sum(!is.na(positive_for_drug)),
            positive_test = sum(positive_for_drug, na.rm = TRUE),
            perc_positive = round(100 * positive_test / n_non_missing, 1)) %>%
  select(drug_type, sex, year_cat, perc_positive) %>%
  unite(sex_year_cat, sex, year_cat) %>%
```

```
spread(sex_year_cat, perc_positive) %>%
knitr::kable(col.names = c("Drug type", "F 1999-2002",
                           "F 2003-2006", "F 2007-2010",
                           "M 1999-2002", "M 2003-2006",
                           "M 2007-2010"))
```

Drug type	F 1999-2002	F 2003-2006	F 2007-2010	M 1999-2002	M 2003-2006	M 2007-2010
Alcohol	26.4	24.3	27.1	43.2	42.9	43.3
Cannabinoid	2.8	5.7	7.3	5.8	10.3	11.8
Depressant	3.4	3.8	4.8	2.0	2.5	3.2
Narcotic	4.2	4.9	7.0	2.2	3.4	4.0
Other	5.6	6.6	7.2	4.3	4.5	4.2
Stimulant	7.2	9.1	8.7	10.5	11.9	9.2

Descriptions of Cleaning Code

Expanding on *what* is happening and *why*.

1. In this step, we are reading in the person_YEAR file that matches with the YEAR specified in the function. Since our working directory is fars_analysis, we use the relative pathname to get this file, which we name as "person_file". We do this to read in each person_YEAR for each of the years in the study (1999 - 2010) and name it as "person_file" to make it easier to read in the data file. We then read in each "person_file"", which is in a .csv format using the `readr` library and name it"df".

2. In the second step, we are changing all of the column names in "df" to lower case. This is done because it makes it easier to type out the column names when we begin cleaning the data in R.

3. The next step is to select all of the column names in the FARS data that will provide us with the information relevant to our study topic. The answer for the second question regarding the article mentioned above the cleaning code chunk explains the variables chosen from the FARS data in more detail.

4. In this step, we are filtering the person_type variable (per_typ) and injury severity variable (inj_sev) for driver and fatal injury, respectively. In the FARS data, per_typ equal to driver is coded with a 1 and inj_sev for fatal injury is coded with a 4. That is why we are filtering for 1 and 4 for those respective variables. The next part of this step, which uses the `select` function and minus sign, allows us to only select the filtered information from these two variables. We do this because only the information relevant to our study topic is needed.

5. In this step, we `unite` three different columns from the FARS data (st_case, veh_no, and per_no) and call it "unique_id". We then add a column called "year" that has the year of the study, which was specific in the function name. We also add a column called "unique_id" that adds the information from our newly united unique_id column and adds the year. The unique_id and year information is separated using an underscore in our new unique_id column. We do this to combine relevant information for each of the cases that meets inclusion criteria across all of the years of the study.

6. This step is similar to step 4 in that it is limiting the values in a column to be appropriate for the Brady and Li study. Here, we are filtering the state column to show states that were included in the Brady and Li study. These states are California, Hawaii, Illinois, New Hampshire, Rhode Island, and West Virginia. In the FARS data, the states are coded using numbers. California, Hawaii, Illinois, New Hampshire, Rhode Island, and West Virginia are coded as 6, 15, 17, 33, 44, and 54, respectively, which is why we are filtering the state column to these numbers. We then select just these filters states and unselect the rest by using the `select` command with the minus sign.

7. In the FARS data, the sex variable is also coded using numbers. This step first uses the `ifelse` function because there is some coding for the sex variable in the FARS data that does not equate to male or female. In the FARS data, 9 is coded as unknown. For the `ifelse` function, if sex is TRUE for 9 (unknown) then the function will put NA. If it is FALSE for the logical vector (sex == 9), the function is telling it to put the value for sex, which will be "Male" or "Female". The next line of code in this step is overriding the sex column with labels for the numerical levels. Since 1 is equal to male and 2 is equal to female in the FARS data, this command is telling the code to put "Male" in the place of 1 and "Female" in the place of 2. This is done to make the sex variable easily readable in the clean data set and subsequent graphs and tables.

8. Step 8 is similar to step 7 in that it uses the `ifelse` function to clean a variable that has more information than we want. For this step, we are looking at the alcohol result (alc_res). For the `ifelse` function, if the logical vector, which is alc_res greater than 94 is TRUE, then put NA. If it is FALSE, divide the alc_res by 10. We then add a new column called Alcohol, which reports the alc_res (when is less than or equal to 94 and is dividied by 10) if it is greater than or equal to 0.01. The second part of this step is to unselect all of the alc_res that do not fit the inclusion criteria. This is done to only included BAC values that are greater than or equal to 0.01 g/dL while also excluding other variables that are coded for in the FARS data for alc_res such as 96 being equal to "None Given".

9. In this step, we create a column called lag_mins that appropriately reports the data after the `ifelse` function. For this `ifelse` function, if lag_mins is coded as 99, which is unknown, the new lag_mins column will have NA as the value. If lag_mins is FALSE for lag_mins == 99, then the value will be the lag_mins value. We do this because 99 is coded as "unknown" in the FARS data and is, therefore, not an actual minute value to be reported and used in the analysis.

10. This step is similar to the step above, but it is a bit more complicated because the coding for lag_hrs changed between 2008 and 2009. The code is saying that if the year of the data is less than or equal to 2008, then lag hours that are coded for as 99 or 999 should have a value of NA since 99 is unknown for 1975 to 2008 in the FARS data. If lag_hrs is not equal to 99 or 999, then put the lag_hrs value. The next part of this code is that if the year is NOT less than or equal to 2008, run the subsequent `ifelse` function. That `ifelse` function says that if lag_hrs is exactly equal to 999 is TRUE, which is coded for as "unknown", then put NA. If it is FALSE, then put the lag_hrs value. Since 99 is an hours value for 2009 and 2010 data in our study, we want to make sure we are not coding it as "unknown", which it would be for 1999-2008 data.

11. This step is filtering the lag_hrs and lag_mins to match up with our study criteria, which limits the deaths to within an hour of the accident. We are, therefore, filtering the lag_hrs column to anything less than 1, which is less than 1 hour, *OR* to lag_hrs exactly equal to 1 and lag_mins exactly equal to 0. This allows the lag_hrs to be filtered to exactly 1 hour or less. We then deselect the lag_hrs and lag_mins that do not fit this inclusion criteria.

12. In this step, we do something very similar to what we do in step 10. Except for this step, we are working with the age variable instead of the lag_hrs. Similarly, the coding in the FARS data changes between 2008 and 2009 for the age variable. Therefore, we are saying that if the year is less than or equal to 2008, follow the subsequent code. The subsequent code uses the `ifelse` function and tells us if the age is exactly equal to 99, put NA, because 99 is "unknown". If age is not equal to 99, then put the value of age. The next part of the code says that if the year is NOT less than or equal to 2008, then follow that subsequent code. That code also uses the `ifelse` function, but it is slightly different since the coding changed between 2008 and 2009. In this `ifelse` code, if age is 998 or 999, then put NA, because 998 and 999 are coded as "not reported" and "unknown", respectively, in the FARS data. We do this to ensure that the change in coding across our study years does not affect the true values of age in our analysis.

13. In the Brady and Li paper, age is expressed using age categories. Step 13 puts age into these specificed categories. We add a new column called agecat and cut the age variable into different groups using the specified breaks. These breaks are 0, 25, 45, 65, and 1000, which correlate to ages. We use 1000 to ensure all ages above 65 are accounted for. We then label these breaks for our different age categories,

which are where the breaks lie. We specify that we want to include the lowest value for each group because our categories are "less than 25", "25 to 44,"45 to 64, and "65 or above" and our breaks are the the lowest values of these groups. After creating our agecat column, we deselect the age variable because we no longer needed it. We want to express age using age category (agecat).

14. In the FARS data, there are three different columns that include results from drug tests. These are drugres1, drugres2, and drugres3. The first thing we do in this step is to gather all of the variables with "drugres" (since that is the commonality between the three variables) with the key = drug_number and value = drug_type_raw. Because the FARS data for the "drugres" variables is coded using numbers for different drugs, our drug_type_raw column will include those different numbers. Our next step in this part of the cleaning process is to convert those numeric codes into the drug categories. To do this, we add a new column called drug_type and use the `ifelse` function based on the coding in the FARS data. For the different drug types we want to select (Narcotic, Depressant, Stimulant, Cannabinoid, Other, and None), we say that if the drug_type_raw has the numbers that are coded in that FARS data for that drug type, then put the name of the drug type. If it does NOT, then put NA. This is done so that the drug_type column has all of the names of the different drug categories. We then unselect the drug_type_raw column and drug_number column since we want to express the drug category by its name and not by the number it is coded by in the FARS data. We name this whole step as a new object called "gathered_df".

15. In this step, we are filtering for values that are NOT labeled as NA in the Alcohol and drug_type columns we created in the cleaning process. We do this because we only want to include information where alcohol and drug_type were reported.

16. In this step, we are piping in to the object we created in step 14, which we called "gathered_df". We first filter the drug_type column to values that are NOT labeled as NA. We then group the data by unique_id as well as by drug_type. These groups are summarized in a name column called "has_drug" that all have TRUE as the value. The data is then ungrouped, and a new column called row_num is added that is equal to the row's number in the data frame. The `spread` command adds columns for each of the drug categories (Cannabinoid, Depressant, Narcotic, None, Other, and Stimulant) to each row. The TRUE that was added using the "has_drug" column is now under the column (drug category) that the values were grouped by previously in this step. The rest of the columns were filled with FALSE. Therefore, each row has a TRUE written in a column for the drug categories and the rest of the columns for drug categories say FALSE. The column we created with row numbers is then deselected leaving us with the unique_id and TRUE or FALSE for drug categories. This step, which pipes in to the "gather_df", is renamed as an object called "non_missing_drugs". This is done so we can see if what drug category the driver tested positive for in a fatal injury car accident.

17. In this step, we are back to cleaning up the "df" dataframe. We first select all of the columns except for the ones that contain "drugres". We do this because the object we created in steps 16 and 17, "non_missing_drugs", is the cleaned up version of the FAS columns that contained "drugres". We then join these two dataframes together by the unique_id column since that was used in both data frames. We now have the "non_missing_drugs" cleaned up dataframe added to our "df" dataframe. After we do this, we deselect the "None" column since we want to look at the presence of alcohol or other drugs. The "None" column does not give us any information we want to examine. We then gather all of our remaining drug category columns (Alcohol, Cannabinoid, Depressant, Narcotic, Other, and Stimulant) into a column that is labeled drug_type, which will list out one of the drug categories in the parentheses and another column labeled positive_for_drug. The positive_for_drug will give that TRUE or FALSE value we created in step 16. The next part of this step is to change the drug_type class into a factor class. Finally, we use the `unique` function that will remove any duplicate elements and/or rows from our cleaned up data. This is done to ensure we did not accidentally count a case twice, which could skew our analysis of the data. The very last thing mentioned in this step is returning the "df", which will show the dataframe we just cleaned up for the year we specified in our clean_yearly_person_file function.

18. In the final step of our data cleaning process, we run the function we created, "clean_yearly_person_file"

on all of the years of our study (1999 to 2010). To get the dataframes for each year into one dataframe, which we call `clean_fars`, we used the `map_df` function. This function is used because it applies the function to each element, in this case year, and returns a vector the same length as the input. It combines these individual dataframes into a single dataframe. Just using the `map` function would create a very large list and not a dataframe.