Cleaning Description for Homework 4

Ethan Walker

October 25, 2017

# Question 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)?

## Answer 1

This study looked at drug and alcohol use in deaths that occurred within one hour of a motor vehicle crash, so the unit of observation will be an individual person and their corresponding information. All of this information is available in the person dataset from the FARS data.

# Question 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.

## Answer 2

* Years: database files for 1999 through 2010 were downloaded
* States:
  + Variable name = state
  + Values to keep = 6 (California), 15 (Hawaii), 17 (Illinois), 33 (New Hampshire), 44 (Rhode Island), 54 (West Virginia)
* Person Type:
  + Variable name = per\_type
  + Values to keep = 1 (driver)
* Injury Type:
  + Variable name = inj\_sev
  + Values to keep = 4 (fatal)
* Time from accident to death:
  + Variable names = lag\_hrs, lag\_mins
  + Values to keep: lag\_hrs = 1 and lag\_mins = 0

# Question 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?

## Answer 3

* Age category: This variable is coded as “age” in the FARS dataset, with discrete numeric values from 0-96 prior to 2009 and 0-120 after that. Prior to 2009, 97 is used for ages 97 and older, and 99 is used for unknown ages. For 2009 and later, 998 is used for unreported ages, and 999 is used for unknown ages.
* Year was taken from the name of the dataset, with each dataset containing information for a specific year. To code for a year, we will have to pull that number from the title of the dataset when combining datasets. We can then stratify by individual year, or combine years into year categories and stratify by those.
* Alcohol level is taken from the “alc\_res” variable. This is coded as a value from 0-93, and has some slight variations for years 2010 and prior to 2010. Values of 94 = 0.94 or greater blood alcohol content. Prior to 2010, 95 = test refused, while in 2010 it means not reported. In addition, 96 = not given, 97 = results unknown, 98 = positive reading with no actual value, and 99 = unknown.
* Sex is coded as 1 for male and 2 for female in the sex variable. 9 represents “unknown”, and in 2010, 8 represents “not reported.”
* Non-alcohol drugs were pulled from variables called drugres1, drugres2, or drugres3, which gives results from up to three drug tests that may have been performed. They are all coded similarly, with a range of numbers corresponding to a specific drug type. In 2010, 95 represents “not reported”, whereas prior to 2010 this is grouped into the 999 value which represents “unknown/no reported.”

# Code Description

# 1

A .csv file with raw FARS data for each year is saved in the "data-raw/yearly\_person\_data" directory as "person\_[year].csv". Since the working directory is the project directory, the person\_file object gives the path to the raw data file for whatever year is entered in the year argument of the function. The raw data for a year is read into R with the read\_csv function, which reads in a comma separated file as a tibble in R. The function takes the file path (person\_file) as its first argument, and the raw data frame is saved as an object called df.

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

This step is converting column names in each dataset from upper case to lower case. We are doing this because R is case sensitive, and keeping column names consistently lower case makes them easier to work with.

# 2. Convert all column names to lowercase.  
colnames(df) <- tolower(colnames(df))  
df <- df %>%

# 3

This is using the select function to select specific variables from the new df dataframe. We are pulling out only the variables we will need in order to perform the analysis in subsequent steps, which makes a smaller data frame that is easier to work with.

# 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

This is using the filter function to filter out all observations that are not per\_type = 1 (driver), and inj\_sev = 4 (fatal). This reduces the number of observations in our dataframes and limits them to only observations we are interested in. We then use the select function to exclude the previous two variables that we filtered with, since we will no longer use them in the analysis.

# 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

This is creating a new variable called unique id by combining the variables of st\_case, veh\_no, and per\_no. This first part uses the unite function to combine the variables into a new one. After this, the mutate function is used to create a new unique\_id variable that has year pasted onto the end. This ensures that each observation is truly unique and that there are no repeated id numbers.

# 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

This step uses the filter function to filter out all states except the 6 we are interested in using for the analysis (6, 15, 17, 33, 44, 54). The select function is then used to select all variables currently in the dataframe, except for state. state is selected out because we will no longer need it for analysis, and it is a part of the unique id already.

# 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

This step uses the mutate function to code a value of 9 in the sex variable to NA. We then take the updated sex variable and convert it to a factor with levels of Male and Female corresponding to 1 and 2. This will help us label the variable and its results later on in the analysis.

# 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

This is mutating the alc\_res variable so that results over 94 = NA, and other results are divided by 10. Then a new variable called Alcohol is created that is pulled from alc\_res when the value is greater or equal to 0.01. Lastly, the select function is used to select all variables in the dataframe except alc\_res. This was all done to give us an Alcohol variable only with values greater than 0.

# 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

This step is taking the lag\_mins variable and mutating it through an ifelse statement. If lag\_mins = 99, then these values are coded as NA. All other values are kept the same. This is done so that NAs aren't coded as a number, which will make the variable easier to work with during analysis.

# 9. Specify missing values for the lag minutes.  
dplyr::mutate(lag\_mins = ifelse(lag\_mins == 99, NA, lag\_mins))

# 10

This step is similar to the previous one, except it is recoding NA values for the lag\_hrs variables. NAs were coded differently prior to 2009 (value = 99) than they are 2009 and later (value = 999). This code is dividing the dataframe into sections before and after 2009, searching for the specified values of 99 or 999, and recoding them as NA. This is done for the same reason as the section above - to make the variable easier to work with later on.

# 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

This step is using the filter fuction to select values where the time of death after the accident was 1 hour or less. This is done by filtering for when lag\_hrs is less than 1, or when lag\_hrs = 1 and lag\_mins = 0. After this, the select function is used to select out the lag\_hrs and lag\_mins variables. Since the dataframe has now been filtered by those variables, we don't need them and they can be removed to make things more tidy.

# 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

Similar to the lag\_hrs step above, age is coded differently prior to and after 2009. This step is looking through the dataframe for years prior to 2009, and coding ages of 99 as NA with the mutate function. Similarly, it then looks through the rest of the years in the dataframe and codes values of 998 and 999 as NA. This is done so that NA values are not a number, which could cause problems in analysis we do later on.

# 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

This step is using the mutate function to create an age category variable agecat from the continuous age variable. The new variable includes break points of 25, 45 and 65 to create categories of < 25, 25-44, 45-64 and 65+. Corresponding labels are then created for each category. right = FALSE is indicating that intervals should not be closed on the right. For example, this makes the first interval < 25 instead of < 26. include.lowest = TRUE is saying that any values equal to the highest break should be included. Lastly, the age variable is selected out, meaning that we will only use the agecat variable going forward.

# 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

This step is taking all columns that contain "drugres" and putting them into a tidy format where the old column name is found under a column called drug\_number and the old column content is found under a column called drug\_type\_raw. The mutate function is then used to create a new variable called drug\_type that labels each drug type as it is coded in the FARS manual. The new drug\_type variable is then classified as a factor, and the two variables that resulted from the gather step are then selected out. This is all done to give us a single variable for drug type that will be easier to work with in the subsequent analysis.

# 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

This step filters out any missing data in the Alcohol and drug\_type variables by using the filter function, and remembering to add the "!" to filter out the missing data instead of everything else. Since alcohol and drug use are our main dependent variable, there is no sense in looking at observations where each of them are missing.

# 15. Filter out any observations where both alcohol and drug data is missing.  
dplyr::filter(!(is.na(Alcohol) & is.na(drug\_type)))

# 16

This is creating a dataframe that includes only individuals who had at least one drug in their system after the accident. We are starting with the full dataframe of people who had data for alcohol or drugs, and piping this into the new dataframe. The next line is filtering out everyone who had "NA" for the variable drug\_type. Next, the group\_by function groups the data first by unique\_id, and then by drug\_type. The summarize function then takes the grouped data and adds a new logical variable called had\_drug that is filled with "TRUE", since all of these individuals did have drugs in their system. Next, the data is ungrouped, and then a new variables is created with each observation's row number. This tells us how many unique instances of drug use there were in the data dataframe. The spread function is then used to turn each drug type into a column header with the result of has\_drug as its fill, and the rest of the values are filled in with has\_drug = FALSE. Lastly, row\_num is selected out before we move on to the next step of combining datasets.

# 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

This step will join the two dataframes together to give us a single dataframe where we can easily tell if each unique id number had used drugs or alcohol, and what type. First, we select everything from the full dataframe except for the drugres variable, since we will be joining full drug results from the subsetted dataframe. Next, we join all of both datasets together by unique\_id. After the dataframes are joined, we select all variables except None (no drug use). Using the gather function, we then create a drug\_type column from each of the specific drug type columns, and create a new positive\_for\_drug variable that is filled with the result of each specific drug type. We then classify the drug\_type variable as a factor, and then use the unique function to return a dataframe with duplicate elements removed. The last step is to then close out this function by returning the new, complete and clean df.

# 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)  
}

# 18

This last step is using the new function we made to clean each of the yearly files and save a compiled dataframe called clean\_far. Since the only input of the function clean\_yearly\_person\_file is the year of the file, we can input the years we want (1999:2010) into the map\_df function, and tell that function to input each year we specified as a call into our function. The map\_df function then runs our function in a loop until a file for each specified year has been cleaned. The map piece of the function will process each file and compile the results for each year into a list, but then the df piece of the function binds each item in the list together to form a single, final dataframe called clean\_fars. This is then saved into the file specified in the last line of code. That's pretty neat.

# 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.  
clean\_fars <- purrr::map\_df(1999:2010, clean\_yearly\_person\_file)  
save(clean\_fars, file = "data/clean\_fars.RData")