cleaning\_description

Marialexia Alfano

October 23, 2017

knitr::opts\_chunk$set(eval = FALSE, message = FALSE, warning = FALSE)

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)

Comment #1: This step introduces the framework of the function we are going to create which will be used to clean all the data in the end. We set the function name "clean\_yearly\_person\_file" and the argument "year". Then we are reading in the data from the folder "yearly\_person\_data" where all the individual yearly datasets are saved. Combining the terms "person\_" and "year" allows the files to be read in as they are saved (i.e. person\_1999.csv), so readr can be used to load in "person\_file" and create the "df" dataframe. This has all of the yearly\_person data from the .csv files, combined.

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

Comment #2: this step here changes the column names from uppercase to lowercase. The variables in the FARS dataset are labelled in uppercase and by changing these to lowercase, it makes it alot easier to work go ahead and select and clean the data without having to worry about switching to uppercase to refer to the variables that we need moving forward. The last line is setting up to edit the df object using piping.

# 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) %>%

Comment #3: This step is selecting and limiting the dataframe to include these certain variables, using the select function in dplyr. st\_case, veh\_no, and per\_no are all unique IDs where as the other variables contain codified values, which will be called upon in further cleaning steps. These variables represent the data that we need to use in further cleaning steps to properly limit the dataframe in order to replicate the Brady and Li 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) %>%

Comment #4: The piping of more dplyr functions continues in this step. More specifically the filter function is used to employ some of the constraints mentioned in the study such as, we want only drivers. Using the logic "equals" or ==, we can filter to keep just the observations for which per\_type is 1. Added to this constraint is also the fact that we want just the observations that are not only "drivers" but are also fatal crashes (i.e. inj\_sev == 4). With the & between them, we are requiring both these to be true in order for an observation to remain in the dataframe. Once we've used these variables to constrain the dataframe to the observations we want, we no longer need the values within them. We can get rid of these columns by selecting them "out" by using the "-" before the column name.

# 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 = "\_")) %>%

Comment #5: This step is using the variables mentioned earlier that are unique identifiers within the dataset to create a "unified" unique ID column. First the unite function within tidyr is used to combine the st\_case, veh\_no, and per\_no variables into the space of one column. Then the mutate funciton from dplyr is first used to create a column for the year each observation (driver fatality) occurred, then it is also used to paste the newly combined unique\_ids together with the year within the new column "unique\_id". The sep = "*" indicates an "*" be used as a separation between the unique IDs, and the year which are now combined into one result within a single column.

# 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) %>%

Comment #6: This next step is a continuation of employing the constraints on the dataset that will be necessary to reproduce the Brady and Li analyses. It was noted in the article that only certain states have consistent drug testing data available over the study period and so the dataframe must be confined to these states. The filter function is used to filter the observations to ones that occurred within (aka %in%) the states above (which are codified). The numbers 6, 15, 17, 33, 44, and 54 represent California, Hawaii, Illinois, New Hampshire, Rhode Island, and West Virginia, respectively. Once we use the state variable to further narrow down the observations we want to include in the analysis, we can get rid of the variable by selecting it out again "-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"))) %>%

Comment #7: The majority of the values within the variables in this dataset have been codified. So categorical values such as sex (i.e. male/female) have been assigned numbers instead. The goal in this step is to convert the codified version back to the typical male/female level. The mutate function is used to make these changes to the "sex" variable column. The ifelse statement determines that where sex == 9, it will be considered an NA, since 9 is the codified value for sex = "unknown". The factor statement following that changes the levels in the order 1, 2 to be factors labeled "Male" and "Female", in that same order (i.e. 1 = Male, 2 = Female). This will create a column for sex that shows for each observation -- "male, female, male, male"" etc. instead of "1, 2, 1, 1".

# 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) %>%

Comment #8: This step uses the data from the variable alc\_res, which are alcohol test results, to create a logical output for whether or not the driver had alcohol in their system. The mutate function is used to make the changes to the alc\_res variable/column. The ifelse statement here is saying that if the value of alc\_res is greater than 94, it is considered NA because any of the numbers higher than "94", are coded to represent situations such as "unknown test result". The next statement says that if alc\_res is any number greater than or equal to 0.01, then the output will be "Alcohol" in the column, not the numerical value. Once these changes are made using this variable, it can be removed using the select out (-alc\_res) because it is no longer needed as an column on it's own within the dataframe for the analyses.

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

Comment #9: This step uses mutate to change the lag\_mins variable. More specifically, since the values within this variable were codified, the ifelse statement is saying that if the lag\_mins are equal to 99, the number "99" should be replaced with an NA, which is what the 99 represents in this case.

# 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))  
 }

Comment #10: This step results in a similar outcome to the previous step, but this time it's a little different because the system for coding the information changed over the study period for lag\_hrs. Up until 2008, the lag\_hrs values that were "unknown" were recorded as 99 and then after 2008, the "unknown" values were recorded as 999. Therefore, when mutating this variable, it must be specified to mutate the 99 values to NA values if the year is <= 2008, but otherwise, as the second statement indicates, if lag\_hrs is equal to 999, that value should be mutated to an NA value.

# 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)

Comment #11: Once we've specified what codified values should be changed to NA values, we can move forward with using the lag\_hrs and lag\_mins to further limit our observations to drivers that strictly died within an hour of the crash. This step filters the observations to include only lag\_hrs that are less than 1 or, if the lag\_hrs equal 1, it can't be more than 1 so therefore the lag\_mins would also have to equal 0. Once we filter the observations to fit these constraints, we can go ahead and remove the lag\_hrs and lag\_mins using the select out (-lag\_hrs, -lag\_mins) because used the variable to select what we need and we no longer need those values in our dataframe.

# 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))  
 }

Comment #12: What is happening here with the age variable is similar to what was happening before with the NA values in lag\_hrs versus lag\_mins as a result of the change of the coding system over the study period. Up until 2008, the age values that were "unknown" were recorded as 99 and then after 2008, the "unknown" values were recorded as 999. Therefore, when mutating this variable, it must be specified to mutate the 99 values to NA values if the year is <= 2008, but otherwise, as the second statement indicates, if age is equal to 999, that value should be mutated to an NA value.

# 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)

Comment #13: This step is using the age variable to create a new column that has the age data broken down into categories. The mutate function is used to create a new column "agecat" which uses the age data cut into 5 different breaks in the values. Breaking at 0, 25, 45, 65, and 1000 allows for 4 categories (i.e. 25 or younger, 25-44, 45-64, and 65 or more). Whichever category each observation falls within is the category that is presented in the agecat column. Once we've use the age variable to create this agecat column, we no longer need the values in the age column so we can get rid of the column by selecting out (-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) %>%

Comment #14: The dataframe we are working with at this point still has several different columns for drug\_res (i.e. drug\_res1, drug\_res2 etc.). This step is about gathering those columns together into one column, "drug\_type" and creating a df that indicates this gathering (i.e. gathered\_df). First a "drug\_number" column is created to combine all of the different codified outcomes "drug\_type\_raw", which contains the results from all the drug\_res columns, into one. But we really only want to consider certain drugs that were tested for (i.e. narcotics, depressants, stimulants, cannabinoids). The mutate function here is then used to create the final column, "drug\_type" and the ifelse statements bring in the specific data we are looking for in this analysis by saying any result within (%in%) drug\_type\_raw that is the codified value 600:695, should be written in as "Cannabinoid" in the drug\_type column. The drug\_type\_raw and drug\_number were mainly a scaffold to help create the final drug\_type variable and we no longer need all those values, so we can get rid of them using the select out (-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)))

Comment #15: This step is looking to apply yet another constraint that was detailed in the Brady and Li analysis. They selected for observations that had drug data available and filtered out any observations that didn't have this data available. This step is employing this constraint by filtering out missing or NA data related to alcohol and drug testing.

# 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)

Comment #16: *Pipe 1*: Defines object non\_missing\_drugs which references the gathered\_df defined object *Pipe 2*: Filters out any values within drug\_type variable in gathered\_df that are NA *Pipe 3*: The group\_by function groups the observations by drug\_type and unique\_id *Pipe 4*: Summarize function creates has\_drug variable that is TRUE for all observations. We filtered out NA values, and grouped by unique\_id AND drug\_type (which was originally filtered to include observations where alcohol/drug result data were available - step#15) *Pipe 5*: ungroup unique\_id and drug\_type so that the has\_drug variable can be used in conjunction with the other variables in the dataframe. *Pipe 6*: this mutate function provies row numbers for all the observations that are in the dataframe at this point *Pipe 7*: spread takes the category "drug\_type" and provides whether it's true or false that it has drugs (the true or false statements given by has\_drug). Fill = FALSE fills any observations that do not have has\_drug = TRUE, with "FALSE" *Pipe 8*: once the remainder of the observations are accounted for in regards to the has\_drug result (TRUE or FALSE), by reference of row number, the row\_num column can be removed using 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)

Comment #17: *Pipe 1*: This redefines the df dataframe *Pipe 2*: The df dataframe still has the "drug\_res" columns because the gathered\_df dataframe is the one that contains the gathered version of those variables. This step is selecting to remove those variables. *Pipe 3*: The full\_join function puts together all variables from each dataset that is being joined (df and non\_missing\_drugs, in this case are the two that are being joined). These dataframes are joined by a common variable, "unique\_id". *Pipe 4*: This selects out the variable called "None" from the dataframe *Pipe 5*: This step gathers the drug types(i.e.alcohol, cannabinoid, depressant, narcotics, other, stimulants) into the drug\_type column - key argument = drug\_type. It also presents the column, "positive\_for\_drug" which gives TRUE/FALSE for the drug\_type - value argument = positive\_for\_drug. *Pipe 6*: The mutate function changes the drug\_type variable into a factor/level like we did earlier with the sex (male/female).

unique() prints out the unique values that can be found within a variable. return(df) prints out the complete, cleaned dataframe after the function created is run.

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

Comment #18: The } marks the close of the function that was created for which the code above will run. By running map\_df(1999:2010, clean\_yearly\_person\_file), we are reiterating this function for every person\_year file that was uploaded from 1999-2010. This cleans each year's dataset the same way and joins them all together in the end. The clean\_fars object is defined, which is the output of this function iterated across all the .csv files uploaded for this analysis. The clean\_fars object that is defined is the compilation of all the datasets and in a format cleaned for further use. The save function here saves this clean\_fars file into the data folder (instead of the data\_raw folder within the project).