cleaning\_description

Rebecca Foos

November 1, 2017

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

fars\_dbf\_to\_csv <- function(year) {  
# Save the directory where .dbf files are saved.  
dir <- "data-raw/yearly\_person\_data"  
# Read the .dbf file for a year into R.  
person\_data <- foreign::read.dbf(paste0(dir,"/PERSON\_", year, ".DBF"))  
# Save each file as a csv to the "data-raw/yearly\_person\_data" directory.  
person\_file <- paste0("data-raw/yearly\_person\_data/person\_", year, ".csv")  
readr::write\_csv(person\_data,  
path = person\_file)  
# Return NULL so that the function doesn't print out anything.  
return(NULL)  
}  
# Iterate the fars\_dbf\_to\_csv across all files.  
purrr::map(1999:2010, fars\_dbf\_to\_csv)

This code chunk reads all of the .dbf files in as unique .csv by year. The foreign package is integral for this conversion.

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)  
}  
# 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")

1: This intro to the function allows for a relative reference to all of the annual files which will be pulled for the final clean dataframe. The function name has just been set to clean\_yearly\_person\_file with the only argument being year. The source files come from the "yearly\_person\_data" folder yearly .csv. Combining the constant "person\_" and variable year reads the file while transforming it into a dataframe object.

2: Here the column names are transformed from uppercase to lowercase. This is used to standardize all of the column names between years, as the FARS system is not always consistant, and lowercase letters are easier to work with in R (a factor of standardization, which is always good form). As will be seen within the full code chunk, the last symbol %>% initiates a pipe, which will build the code with greater organization and less redundancy. Piping also improves processing time, as dataframes are not saved between piped commands as in code sans piping.

3: The select function limits the dataframe to only the columns specified through the dplyr package. Of the selected variables, st\_case, veh\_no, and per\_no identifiers and the others coded values. The contains(drugres) selection catches all results for drugs regardless of the drug identifier. In other words, it will catch all columns containing the characters "drugres" All of these variables will be further cleaned in the next steps.

4: Filtering by per\_typ and inj\_sev reduces the dataframes to only the records for both person type 1 (coded for driver) and injury severity 4 (coded for fatality). The double equal sign indicates both parameters must be met exactly for the record to remain in the dataframe. Following this filter, the two columns have served their purpose and are removed from the dataframe using a negative select (select(-inj\_sev)).

5: The unite function from the tidyr package creates a new unique\_id column containing st\_case, veh\_no, and per\_no. Mutate from dplyr is then used to first create a year column and then tag the year onto the end of the new unique\_id with a separating underscore (sep = "\_"). This allowed for unique identifiers between annual dataframes as well.

6: The dplyr filter function was used to filter the states column to ids 6, 15, 17, 33, 44, and 54 (representing California, Hawaii, Illinois, New Hampshire, Rhode Island, and West Virginia). All other records with state codes not in this subset are filtered out of the dataframe. The select function is then used in the negative to unselect the state column, removing it from the dataframe as its purpose has been served.

7: Mutate is used to change the coded values for sex back into factor levels of "Male" and "Female" instead of the coded values "1" and "2". The code for missing values in sex, "9", is replaced with NA to indicate within statistical analysis that the value is missing and not of value 9! Factoring these levels is important, as a numerical class is not applicable within statistical interpretation.

8: Alcohol blood levels were interpreted by the mutate function to replace levels above 94 (unknown test results) with NA and values less than 94 with the value over 10. The threshold of .01 was then used as a logical test for the mutated alc\_res value to populate the new column "Alcohol" with YES/NO responses for the presence of alcohol. The alc\_res column was then removed using the negative select function.

9: Dplyr was used to mutate the lag\_mins variable, replacing the code "99" with its meaning, NA.

10: Similar to comment 9, this mutate of the lag\_hrs column replaces the codes "999" and "99" with the meaning NA for any year before 2009 or the code "999" with NA for years after 2008. This is accomplished by use of a conditional if/else statement, which is necessary due to inconsistant coding practices for NA values prior to 2009.

11: Filter is used for the lag\_hrs and lag\_mins variables to remove any fatalities which occurred either less than 1 hour after the accident (lag\_hrs < 1) or exactly 1 hour and 0 minutes after the accident (lag\_hrs == 1 & lag\_mins ==0). The negative select function is then used to remove both the lag columns from the dataframe.

12: Due again to inconsistant coding practices accross all years, the if/else statement is used to mutate the age column to replace code "99" with NA before 2009 (if) and codes "998" and "999" with NA for years after 2008 (else).

13: The age column is used to create a new column, agecat for age category. Breaks were specified within the age variable using the dplyr function cut. The age categories defined by the breaks were then labelled with the corresponding ranges. The include.lowest arguement is set to TRUE, indicating that the break value should be included in the next dataset as the lowest value, i.e. 25 should be included in the "25--44 years" category. The age column was then removed using a negative select, as the age category would now supplant the precise age.

14: The tidyr gather function is used to combine the multiple columns containing the character string "drugres" into one column called "drug\_type" into a new gathered dataframe. The "drug\_number" column is created first, combining all of the codes from "drug\_type\_raw" which is derived from "drugres" columns. The mutate function is used to create the "drug\_type" column, populating it series based the appropriate codes. First, "drug\_type\_raw" is queried for values between 100 and 295, which would correspond to the Narcotic drug codes. If the value is in the range, "drug\_type" will be populated with "Narcotic". If it isn't in the 100 to 295 range, the mutate function moves to the else - the next level test for "Depressant" in the 300:395 range. Multiple ifelse statements finally populate the drug\_type column with the appropriate label classification. The final line in the mutate converts the column drug\_type to class factor. Lastly, dplyr is used to unselect the drug\_type\_raw and drug\_number columns, which are no longer needed.

15: Filter is used with a BANG (!) in the negative to filter out records where both Alcohol and drug\_type are NAs.

16: A new dataframe object non\_missing\_drugs is created from gathered\_df. The second line filters for drug\_types which are not NA, removing all records with NA for drug\_type. The group\_by function groups the records by drug\_type and unique\_id in preparation for the next function, summarize. Summarize is used to specify that all grouped records are related to drugs (TRUE). Ungroup is used to to deconstruct the grouping organization. Mutate is used to create a new column, numbering all rows in the dataframe. Spread determines if "drug\_type" by the parameter "has\_drug" is true, filling any absent values with FALSE. Now that has\_drug is fully populated, row\_num can be unselected from the dataframe in the last line, removing the column.

17: The df dataframe is being overwritten by the first line with the fully piped "new" df. The second line unselects the "drugres" column, removing the column from the dataframe. Full\_join is used to join the non\_missing\_drugs dataframe and df (as piped in), retaining everything from both dataframes in the merge as dictated by "full". The join variable for this function is "unique\_id." The next line unselects "None" to remove the column. The tidyr gather function pulls drug type labels into the drug\_type column. It also creates the column "positive\_for\_drug" populated with TRUE or FALSE for the drug\_type. The mutate function reclassifies drug\_type as a factor. Unique limits the dataframe to unique records, eliminating any redundancy. Lastly, return(df) tells the function to print the clean dataframe at last!

18: The clean\_yearly\_person\_file function is called within the map\_df function on each year 1999 to 2010. The results of these yearly function calls are mapped to the clean\_fars dataframe, and then saved as clean\_fars to the file path specified.