cleaning\_description

Kate Huebner

10/20/2017

knitr::opts\_chunk$set(echo = TRUE, message = FALSE,  
 warning = FALSE, error = FALSE)

list.files("../data-raw/yearly\_person\_data/")

## [1] "person\_1999.csv" "person\_1999.dbf" "person\_2000.csv"  
## [4] "person\_2000.dbf" "person\_2001.csv" "person\_2001.dbf"  
## [7] "person\_2002.csv" "person\_2002.dbf" "person\_2003.csv"  
## [10] "person\_2003.dbf" "person\_2004.csv" "person\_2004.dbf"  
## [13] "person\_2005.csv" "person\_2005.dbf" "person\_2006.csv"  
## [16] "person\_2006.dbf" "person\_2007.csv" "person\_2007.dbf"  
## [19] "person\_2008.csv" "person\_2008.dbf" "person\_2009.csv"  
## [22] "person\_2009.dbf" "person\_2010.csv" "person\_2010.dbf"

raw\_data\_path <- "../data-raw/yearly\_person\_data"  
person\_99 <- foreign::read.dbf(paste(raw\_data\_path, "person\_1999.dbf", sep = "/"))  
person\_00 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2000.dbf", sep = "/"))  
person\_01 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2001.dbf", sep = "/"))  
person\_02 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2002.dbf", sep = "/"))  
person\_03 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2003.dbf", sep = "/"))  
person\_04 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2004.dbf", sep = "/"))  
person\_05 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2005.dbf", sep = "/"))  
person\_06 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2006.dbf", sep = "/"))  
person\_07 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2007.dbf", sep = "/"))  
person\_08 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2008.dbf", sep = "/"))  
person\_09 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2009.dbf", sep = "/"))  
person\_10 <- foreign::read.dbf(paste(raw\_data\_path, "person\_2010.dbf", sep = "/"))  
  
# Use the following code only if you downloaded .dbf files by hand, instead of  
# writing them out as .csv files using R. The fars\_dbf\_to\_csv function reads  
# in .dbf files as data frames, and then writes them as csv files to the  
# "data-raw/yearly\_person\_data" directory. The `map` step iterates the  
# function across all of the .dbf files saved in the  
# "data-raw/yearly\_person\_data" directory.  
fars\_dbf\_to\_csv <- function(year) {  
 # Save the directory where .dbf files are saved.  
 # dir <- "yearly\_person\_data"  
 # Read the .dbf file for a year into R.  
 person\_data <- foreign::read.dbf(paste0(raw\_data\_path, "/person\_", year, ".DBF"))  
 # Save each file as a csv to the "data-raw/yearly\_person\_data" directory.   
 person\_file <- paste0(raw\_data\_path, "/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)

## [[1]]  
## NULL  
##   
## [[2]]  
## NULL  
##   
## [[3]]  
## NULL  
##   
## [[4]]  
## NULL  
##   
## [[5]]  
## NULL  
##   
## [[6]]  
## NULL  
##   
## [[7]]  
## NULL  
##   
## [[8]]  
## NULL  
##   
## [[9]]  
## NULL  
##   
## [[10]]  
## NULL  
##   
## [[11]]  
## NULL  
##   
## [[12]]  
## NULL

clean\_yearly\_person\_file <- function(year) {  
 # 1. Read data in.  
 #A: Reading in the CSV file as a dataframe, renaming it to "df""   
 #B: To load the data in so that it is worked on in future steps  
person\_file <- paste0(raw\_data\_path, "/person\_", year, ".csv")   
df <- readr::read\_csv(person\_file)  
 # 2. Convert all column names to lowercase.  
 #A: Changing all column names to lowercase and piping this df into the following cleaning steps  
 #B: To make the names in a style that is more tidy and easy to manipulate.   
colnames(df) <- tolower(colnames(df))  
df <- df %>%  
 # 3. Limit variables.  
 #A: Selecting the 11 columns/variables of interest for this analysis  
 #B: This simplifies the dataframe. The study objective is looking at individual drivers who died of fatal injuries to see what drugs might influence fatal crashes.   
 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.  
 #A: Filtering is used to filter out rows where per\_typ = 1 (driver) and inj\_sev = 4 (fatal) and subtracting out everything else that was not 1 for per\_typ and not 4 for inj\_sev  
 #B: We are only interested in drivers that were in fatal crashes for this analysis  
 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.  
 #A: Unite is used to create a descriptive identification based on case number, vehicle number and person number into a new column called unique\_id. Mutate is used to create a new column called year which is pasted on to the unique id with an underscore  
 #B: This unique identification enables analysis using a single unit of observation, which is person id (tied to the other identifiers) allowing analysis such as counts, group by, etc.   
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 not present in that grouped in logical statement.  
 #A: Filtering to state numbers included in that vector, then removing the state column.   
 #B: Those states performed routine toxicological screening  
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".  
 #A: If sex was coded as 9 (which was unknown), the output is NA, otherwise it is coded as sex (male or female).Sex is then turned into a factor with levels 1 and 2, and labelled as male and female  
 #B: Easier to perform analysis to have meaningful variable names (male, female or NA) then code. This is to change this variable into a meaningful, integer value instead of code.   
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.   
 #A: Breaking down the alc\_res variable, where if alc\_res was >94 then NA is returned, if less than 94 its output is the absolute vale of alcohol content. The other values are then divided by 10. A new alcohol column is formed with logical class of TRUE or FALSE. Then the alc\_res column is then removed.  
 #B: Alc\_res > 94 is coded as NA because it represented unknown results, no test given, or a different test was performed, They divided by 10 because then it is signifying the blood alcohol content.This is to change this variable into a meaningful, integer value instead of code  
 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.  
 #A: if lag minutes were equal to 99, they were coded as NA, otherwise they return values 0-59  
 #B: This is to change this variable into a meaningful, integer value instead of code  
dplyr::mutate(lag\_mins = ifelse(lag\_mins == 99, NA, lag\_mins))  
 # 10. Save lag hours coded as missing as `NA`.  
 #A: If the year was < or = 2008 if lag hours were equal to 99 or 999, they were coded as NA, otherwise they return values. If the dataset was Not < or = 2008 (aka more recent), if the lag hours were 999 only they were returned as NA, otherwise they return the values.   
 #B: For dataset 1975 to 2008, hours were coded as 0 to 24, and 99 were NA. After 2009, it was coded as 0 to 719 hours, and 999 was unknown. This is to account for coding definition changes for all the data sets  
 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.  
 #A: Filter is used to retain individuals that died less than an hour or exactly equal to 1 hour and remaining values are removed.   
 #B: The analysis case criteria were fatal accidents within 1 hour, so the data need to be cleaned to include just that.   
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`.  
 #A: If the year was < or = 2008 if age was equal to 99, they were coded as NA, otherwise they return values for age. If the dataset was Not < or = 2008 (aka more recent), if the age was 998 and 999 only they were returned as NA, otherwise they return the values.   
 #B: For datasets 1975 to 2008, age were coded as 0, 1 to 96, 97 was greater than 96, and 99 and above were unknown. After 2009, it was coded as 0, 1 to 120, and 999 was unknown. In datasets 2010 and later, 998 was added to included not reported age. This step was included is to account for coding definition changes for all the data sets.   
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.  
 #A: we make a new column called agecat, then we cut age into 5 bins and labelling them according to their respective age distributions. The lower limit of each bin includes the specified low limit and the upper limit does not.   
 #B: Instead of specifying by a continuous discrete variable for age, we now have a categorical 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.  
 #A: Gather is used to create a new column called drug\_number (which will contain values 1,2 or 3) and the corresponding values (codes) will go into a column called drug\_type\_raw. Mutate is then used to create third column called drug)type where the character name for the corresponding code is entered into the column. Drug type is then turned into a factor. Select is used to eliminate drug type raw and drug number because they are no longer needed.   
 #B: Creating a tidy dataset (called gathered\_df) by putting the meaningful observations into columns instead of variable names, thereby switching from a wide to a long format. Drug type was renamed as a factor indicating a meaningful value instead of coded.   
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.  
 #A: Filter is used to remove missing values for Alcohol and drug\_type columns  
 #B: They are not relevant to the analysis because we are interested in those data.  
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 %>% #piping the gathered df into a new object called non\_missing\_drugs. We will manipulate the gathered df.  
filter(!is.na(drug\_type)) %>% # Filtering out missing values for drug type.  
 group\_by(unique\_id, drug\_type) %>% #Grouping by unique id (first), followed by drug type. This will ultimately change the order of drug type so that we can later summarize.  
 summarize(has\_drug = TRUE) %>% #generating a new column in the grouped data called has\_drug. Everything will be filled with to TRUE.   
 ungroup() %>% #ugrouping the drug type groups  
mutate(row\_num = 1:n()) %>% #making a new column called row\_num that is 1:the total count. This step is needed for spread.   
spread(drug\_type, has\_drug, fill = FALSE) %>% #Changing from long to wide format by drug type, so that each drug type is a new column (ie. narcotic). If there is nothing in that column it won't repeat or continue to fill if it isempty  
 select(-row\_num) #removing the row number column  
# 17. Join this back into the full dataset. (Write a sentence or two for each   
# step in this pipe chain.)  
df <- df %>% # piping the original dataframe into dataframe to edit. This pipe will be to join the new dataframe we have created to the original to add the non-drug user data by unique id.   
dplyr::select(-contains("drugres")) %>% #removing the category drug res because we have it already in our new dataframe  
 dplyr::full\_join(non\_missing\_drugs, by = "unique\_id") %>% #Full join by unique id requires both unique ids to be present in both dataframes in order to join, and all of the values from both  
 dplyr::select(-None) %>% #removing the column none  
tidyr::gather(drug\_type, positive\_for\_drug, Alcohol, Cannabinoid,  
Depressant, Narcotic, Other, Stimulant) %>% #gathering to change once again from wide to long format (from columns to rows)  
 dplyr::mutate(drug\_type = factor(drug\_type)) %>% #Turning drug type into a factor  
unique() #Extracting unique elements to remove duplicate ids   
return(df) #A return statement determines the value the function returns  
} #signifies end of the function  
  
# 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")