Homework #4–5

Fatality Analysis Reporting System (FARS) in action

With R, so far you know how to read in, clean, explore, and visualize data. You're at the point where can use these skills to start to answer research questions at a larger level. Brady and Li (2013) wrote "Trends in Alcohol and Other Drugs Detected in Fatally Injured Drivers in the United States, 1999-2010" (http://aje.oxfordjournals.org/content/early/2014/01/27/aje.kwt327.full.pdf+html) using the same publicly available FARS data (from the National Highway and Traffic Safety Administration) that we've been working with in class. For this homework, you will be replicating some of the results in that article.

Questions about the article (due for Homework #4)

First, read through the paper to get a feel for Brady and Li's motivation for conducting this study, research question, and their overall results. Then, think about how they used the FARS dataset to answer their research question, and answer the following questions about the article. For many of these questions, you will need to consult the FARS documentation file (https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812447).

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)?
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.
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?

Set up an R Project (due for Homework #4— just the set-up of the project, not completion of all files within the project)

Now that we are doing larger-scale projects in R, you should set up your work as an R Project. Take the following steps to create a project for this assignment:

1. In RStudio, "File" -> "New Project" -> "New Directory" -> "Empty Project". Navigate to the directory on your computer where you'd like to save your project and save the project.
2. This will save a new directory on your computer. It will only have one file in it, a ".Rproj" file.
3. Add the following sub-directories:
    - `data-raw`: This is where you'll put the raw FARS data you download from the Department of Transportation's website. You will store all the yearly files in a sub-directory of `data-raw` called `yearly_person_data`. In the `data-raw` directory, you will also store an R script called "clean-data.R" that cleans the data into the final dataset you'll use for analysis. This script will include code defining a function called `clean_yearly_person_file` that will fully clean one year's

data, as well as code to iterate this function over every study year to create one large cleaned data frame called `clean_fars` that you will save in the `data` sub-directory. More details are given later in this assignment on the code that should be included in "clean-data.R".

- `data`: This is where you will store the cleaned dataset you'll use for analysis. You will store this cleaned data as "fars_data.Rdata". This file will be the output from the "clean-data.R" script in the `data-raw` sub-directory.
- `R`: This is where you will save an R script called "fars_functions.R". This script will include the code where you define three functions (`perc_cis`, `test_trend_ca`, and `test_trend_log_reg`) to analyze the cleaned FARS data. More details are given about writing these functions later in the assignment.
- `writing`: This is where you'll save a file called "cleaning_description.Rmd" with the cleaning script provided below, as well as a few sentences per commented line explaining each step. More details about this part of the assignment are given below. You'll also save a file called "fars_analysis.Rmd" that replicates some of the analysis in the Brady and Li paper. This document should render a file that looks like the example "fars_analysis.pdf" document. If you have TeX on your computer, render this as a pdf; otherwise render it as a Word document.

To turn in this final homework (Homework #5), you will push your entire project directory to GitHub and send Rachel and me the link. (We will work on setting this up in an in-course exercise.)

Hint: Throughout this homework, remember that if you're working at the console, your working directory by default is your project directory. If you're working on an R Markdown file, the default working directory is the directory where that .Rmd file is saved.

Pull the raw data and save it locally (due for Homework #4)

FARS raw data is available by year from here: ftp://ftp.nhtsa.dot.gov/fars/. Download the zipped FARS "dbf" data files for all years and save the "person" file for each year locally.

You may choose one of the following ways to do this step:

- If you are feeling less ambitious, you can download the files by hand. Extract the "person" file for each year and save all these in `data-raw/yearly_person_data`. Since the "person.dbf" files do not include the year in their file names, make sure to save them with the year. For example, you would save the person file from the 2005 zip file as "PERSON_2005.dbf".

- If you want an extra challenge, try to save download and save this data directly using R. If you do this, save your code in an R script called "download-data.R" in the `data-raw` sub-directory of the project. If you do this, you may find the `download.file` function very useful, as it can download a file with ftp. You'll need to check the naming conventions for the FARS ftp files. These conventions are different for 1999 and 2000 compared to other years, so you'll need an if / else statement within the function you write. For example, for 1999, you need to download the data from ftp://ftp.nhtsa.dot.gov/fars/1999/DBF/FARSDBF99.zip while for 2001 you need to download the data from ftp://ftp.nhtsa.dot.gov/fars/2001/DBF/FARS2001.zip. Save these zipped files in a separate sub-directory of `data-raw` (e.g., `yearly_fars_data`). You can then write more code to unzip each file, extract the "person" data (the `read.dbf` function from the `foreign` package will be useful for this), convert it to a `.csv` file using `write_csv`, and write those to the `yearly_person_data` sub-directory of `data-raw`. I will include an example of this code in the final homework solution.

Your final output of this step should be a `yearly_person_data` sub-directory in the `data-raw` sub-directory with a separate "dbf" or "csv" file for each year that contains the person-level FARS data for that year.

Explain each step of cleaning (due for Homework #4)

If you downloaded the yearly person files by hand and you have "PERSON_year.dbf" files saved in your `data-raw/yearly_person_data` directory, you can use the following code to write out .dbf files as .csv files. Note: in order for this function to run, each file should be saved as "PERSON_year.dbf" ("PERSON_1999.dbf", for example). If each of your FARS files is already saved as a .csv, skip this step.

```
# 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 <- "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)
```

Next, use the following script to explain each step of the cleaning process, and to create a clean version of the data. The script uses a function called `clean_yearly_person_file` that takes `year` as its only argument.

The function is written to filter out any observations that should be excluded from analysis based on the Brady and Li paper (e.g., non-drivers, fatality not within one hour, etc.). The cleaned dataset has the following variables:

- `unique_id`: A unique identifier for each driver.
- `sex`: A factor with levels of "Male" and "Female".
- `year`: An integer with the 4-digit study year.
- `agecat`: A factor with age categories, as defined in Figures 1 and 3 of the Brady and Li paper.
- `drug_type`: A factor with the levels "Alcohol", "Cannabinoid", "Depressant", "Narcotic", "Stimulant", and "Other". These categorizations are based on categories defined in the FARS data documentation.
- `positive_for_drug`: Logical, whether that driver tested positive for that drug. For "Alcohol", this is based on a BAC $\geq$ 0.01 g/dL, as specified in the Brady and Li paper.

Note that this is not a tidy dataset— there are two levels of observation in this dataset (person and person-drug combination), which results in lots of repeated information (e.g., `age_cat` will be the same for a person across all their observations for different drugs). However, this data frame is in a format that will make it quick and convenient to write code to replicate results in the paper.

Create an R Markdown document in the `writing` sub-directory and call it "cleaning_description.Rmd". Include the script provided below in a code chunk. Commented lines (#1 though 18) briefly describe what is going on at each step. Below the code chunk in your "cleaning_description" R Markdown document, use the FARS documentation and the Brady and Li study to write a few sentences for each numbered comment that (A) expands on *what* is happening on each cleaning step and (B) explains *why* we are taking this step.

```r
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))
  }
```

```r
# 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")
```

For example:

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`.

Generate a clean dataset (due for Homework #4)

Next, save the script above to "clean-data.R" in the `data-raw` sub-directory. You should be able to use this script to iterate the `clean_yearly_person_file` function across the study years and save one large dataset called `clean_fars` to the `data` sub-directory. **Note:** Be sure that when you run this code, your working directory is the project directory.

After running the script above, you should have one large dataset called "clean_fars.RData" saved in the `data` sub-directory. Summary statistics for your cleaned data should look something like this (do not worry if they don't exactly match the Brady and Li study, but they should be in the general ballpark):

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

```
## [1] 156413      6
```

```
length(unique(clean_fars$unique_id))
```

```
## [1] 25593
```

```
summary(clean_fars)
```

```
##   unique_id             sex              year              agecat
##  Length:156413      Male  :121072   Min.   :1999   < 25 years  :39149
##  Class :character   Female: 35335   1st Qu.:2002   25--44 years:61235
##  Mode  :character   NA's  :     6   Median :2004   45--64 years:39870
##                                     Mean   :2004   65 years +  :16108
##                                     3rd Qu.:2007   NA's        :   51
##                                     Max.   :2010
##        drug_type     positive_for_drug
##  Alcohol    :25593   Mode :logical
##  Cannabinoid:26260   FALSE:127597
##  Depressant :25988   TRUE :16894
##  Narcotic   :26086   NA's :11922
##  Other      :26179
##  Stimulant  :26307
```

After loading your cleaned dataset as `clean_fars`, you should be able to run the following code to show measurements of the prevalence of positive drug tests over the full study period by drug type and get results that are fairly similar to those shown below (this table will be included in your "fars_analysis" document):

```
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

Figures (due for Homework #5)

Use the cleaned dataset to recreate the three figures from the Brady and Li paper in your "fars_analysis" document. The final figures should look similar to the ones shown in the example "fars_analysis.pdf" document.

Write functions for `fars_functions.R` (due for Homework #5)

You will write three functions to help analyze the cleaned FARS data. Details are given for each function below. You should save the code defining these functions in a file called "fars_functions.R" in the `R` sub-directory.

Confidence intervals for proportions (`perc_cis`)

The text of the results gives estimates of percentages of drivers fatally injured who tested positive for a given drug by year, along with 95% confidence intervals for these percentages.

Write a function called `perc_cis` that inputs `x` (number of drivers testing positive for a drug) and `n` (total number of non-missing observations) and outputs a character vector for the percentage of drivers testing positive and the associated 95% confidence interval.

Within the code of this function, you will first want to calculate the proportion of drivers that test positive:

$$\hat{p} = \frac{x}{n}$$

Then, you can calculate an estimate of the standard of this proportion:

$$se(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Then calculate the upper and lower 95% confidence intervals as:

$$\hat{p} \pm 1.96 * se(\hat{p})$$

Finally, convert the proportion and its 95% CI to percentages, round the values, and paste everything together to create a one-element character vector that you could include in the text of results or in a table. For example, you would change a proportion of 0.1011 with confidence intervals of 0.0951 and 0.1070 to "10.1% (9.5%, 10.7%)".

Here is an example of running the function when 9,000 drivers test positive for a drug out of 23,000 total observations:

```
perc_cis(x = 9000, n = 23000)
```

```
## [1] "39.1% (38.5%, 39.8%)"
```

Once you've written the `perc_cis` function, you will use it to generate a table with percentages and 95% CIs by drug type for the years 1999 and 2010 (see the example "fars_analysis.pdf" document).

Testing for trend using Cochran-Armitage trend test (`test_trend_ca`)

You can use the `prop.trend.test` function, which comes with base R in the `stats` package, to conduct a Cochran-Armitage test for trend in the proportion of drivers fatally injured who tested positive for a given drug. In the results from this function, the Z^2 statistic for the Cochran-Armitage test is given as the element `statistic` and its p-value is given by `p.value`. You can get the absolute value of the Z-statistic (closer to what is reported in the Brady and Li paper) by taking the square root of the Z^2 statistic. This function requires you to input a summary of the data: `x`, a vector with the number of drivers testing positive in each year, and `n`, a vector with the total number of non-missing observations in each year.

For example, to test for a trend for alcohol, you can run:

```
to_test <- clean_fars %>%
  filter(drug_type == "Alcohol") %>%
  group_by(year) %>%
  summarize(positive = sum(positive_for_drug, na.rm = TRUE),
            trials = sum(!is.na(positive_for_drug)))
ca_alcohol <- prop.trend.test(x = to_test$positive,
                              n = to_test$trials)
sqrt(ca_alcohol$statistic)
```

```
## X-squared
##  1.206804
```

```
ca_alcohol$p.value
```

```
## [1] 0.2275077
```

Write a function that will input the `clean_fars` dataset and `drug` (a character vector giving one of the drug types or "Nonalcohol" for all non-alcohol drugs). The function should output a one-row data frame with one column for the absolute value of the Z test statistic for the Cochran-Armitage trend test over the 12 study years for that drug and one column for the associated p-value. You will need to use an if / else statement within your code to run separate summarizing code for "Nonalcohol" versus other drug types. Set `data` to have a default value of `clean_fars`.

Here are some examples of calls and output from the function you will write:

```
test_trend_ca(drug = "Stimulant", data = clean_fars)
```

```
## # A tibble: 1 x 2
##       Z p.value
##   <dbl>   <dbl>
## 1   0.5   0.604
```

```
test_trend_ca(drug = "Alcohol")
```

```
## # A tibble: 1 x 2
##       Z p.value
##   <dbl>   <dbl>
## 1   1.2   0.228
```

```
test_trend_ca(drug = "Nonalcohol")
```

```
## # A tibble: 1 x 2
##       Z p.value
##   <dbl>   <dbl>
## 1   9.9       0
```

Once you've written the function, you will use `lapply` to apply it over all the drug categories (including all non-alcohol drugs), using the code below or similar code, to generate a table of results for tests of trends

over the study years. You will use this code to write one of the tables in your "fars_analysis" R Markdown document.

```
drug_list <- c("Alcohol", "Nonalcohol", "Narcotic", "Depressant",
         "Stimulant", "Cannabinoid", "Other")
drug_trend_tests_ca <- lapply(drug_list, test_trend_ca)
drug_trend_tests_ca <- dplyr::bind_rows(drug_trend_tests_ca) %>%
  dplyr::mutate(drug = drug_list) %>%
  dplyr::select(drug, Z, p.value)
drug_trend_tests_ca %>% knitr::kable()
```

Testing for trend using logistic regression (`test_trend_log_reg`)

It turns out there are other ways to code for a trend test. Based on Agresti (*Categorical Data Analysis, Third Edition*, 2013):

> The Cochran-Armitage trend test seems unrelated to the linear logit model. However, this test statistic is equivalent to the score statistic for testing $H_0 : \beta = 0$ in that model. ... The Cochran-Armitage trend test (i.e., the score test) usually gives results similar to the Wald or likelihood-ratio test of $H_0 : \beta = 0$ in the linear logit model. The asymptotics work well even for quite small n when n_i are equal and x_i are equally spaced.

In our case, the approximation by the Wald statistic should work very well— we have a large n (total number of observations), the number of observations are fairly well distributed across the years (n_i denote the number of observations in each year, and we don't have lots in some years and few in others), and the years (x_i in this notation) are evenly spaced.

Here is a more practical comment on the choice of how to test for trend in proportions from an R help thread:

> It [i.e., the Cochran-Armitage test for trend] was taught us (epidemiologists) in the courses before we got our hands on logistic regression. ... I do not know of any advantages for the test over logistic regression or Poisson regression. You can take an ordered factor (or a non-ordered on if the levels are properly set up, coerce to numeric and do logistic regression with the numeric result and get pretty much the same result, and you would be doing so in the context of a much more flexible modeling environment. So I see it mainly as of historical interest, something to use when you only have a device that cannot run R.
>
> — David Winsemius

A linear logit (or logistic) model for the probability of a driver testing positive for a drug regressed on year is:

$$logit(\pi_i) = \alpha + \beta y_i$$

where y_i is the year of observation i and π_i is the probability a driver in that year tests positive for a given drug.

Remember that you can fit a logistic model like this in R using `glm` with `family = binomial(link = "logit")`. You can find the Wald statistic for testing $H_0 : \beta = 0$, as well as its p-value, in the `coefficients` element of the summary of the model object. For example, for alcohol, you could run:

```
to_test <- clean_fars %>%
      filter(drug_type == "Alcohol")
log_reg <- glm(positive_for_drug ~ year, data = to_test,
                 family = binomial(link = "logit"))
summary(log_reg)$coefficients
```

```
##                 Estimate  Std. Error   z value  Pr(>|z|)
## (Intercept) -9.986426093 7.912723795 -1.262072 0.2069229
```

```
## year         0.004764057 0.003947741  1.206780 0.2275167
```

The Wald statistic for a test of $H_0 : \beta = 0$ is 1.207, with a p-value of 0.228.

Write a function to fit the logistic model shown above to a specific drug category (including "Nonalcohol": all non-alcohol drugs). The function should take the same inputs as the function that you wrote for the Cochran-Armitage test (`drug` and `data`) and should create the same output (a one-row data frame with test statistic and associated p-value), but in this case, the function should output the Wald statistic testing if the coefficient for year in the logistic model is zero ($H_0 : \beta = 0$). Set `data` to have a default value of `clean_fars`.

Here are some examples of running the function you will write:

```
test_trend_log_reg(drug = "Stimulant", data = clean_fars)
```

```
## # A tibble: 1 x 2
##       Z p.value
##   <dbl>   <dbl>
## 1  -0.5   0.604
```

```
test_trend_log_reg(drug = "Alcohol")
```

```
## # A tibble: 1 x 2
##       Z p.value
##   <dbl>   <dbl>
## 1   1.2   0.228
```

```
test_trend_log_reg(drug = "Nonalcohol")
```

```
## # A tibble: 1 x 2
##       Z p.value
##   <dbl>   <dbl>
## 1   9.9       0
```

You will use `lapply` with this function (using the code below of something similar) to create a table of test statistics and associated p-values for trend in proportions of fatally injured drivers that tested positive for specific drug categories. Again, this table will go in your "fars_analysis" document.

```
drug_list <- c("Alcohol", "Nonalcohol", "Narcotic", "Depressant",
         "Stimulant", "Cannabinoid", "Other")
drug_trend_tests_log_reg <- lapply(drug_list, test_trend_log_reg)
drug_trend_tests_log_reg <- dplyr::bind_rows(drug_trend_tests_log_reg) %>%
  dplyr::mutate(drug = drug_list) %>%
  dplyr::select(drug, Z, p.value)
drug_trend_tests_log_reg %>% knitr::kable()
```

Write "fars_analysis" document (due for Homework #5)

Create an R Markdown document in the `writing` sub-directory and call it "fars_analysis.Rmd".

You should start the document by including a code chunk that loads all required libraries and also loads the cleaned dataset you created and all of the functions you wrote in "fars_functions.R". For example, this chunk might look like:

```
library(dplyr)
library(tidyr)
library(ggplot2)
```

```
load("../data/clean_fars.RData")
source("../R/fars_functions.R")
```

The final document should look like the example “fars_analysis.pdf” document. Render to pdf if you have TeX on your computer. Otherwise, render to a Word document.