Set up

November 8, 2020

About this notebook

These functions will be used for subsequent analysis on stop missingness rate (SMR). These functions query and clean the data; can be used for manipulating a list of dataframes; and count the instances of missingness.

Libraries

```
library(RMySQL)

library(tidyverse)
library(tidyr)
# library(broom) # make tidy the regression outputs
# library(lubridate) # dates
library(stringr) # string manipulation
# library(tidycensus)
# library(kableExtra) # make nice tables
library(ggrepel)
# library(geofacet)
# library(naniar) # replace "NA" with NA, except too slow, use baseR solution
# library(revgeo)
```

Querying Data

```
con <- dbConnect(
  MySQL(), host = "traffic.st47s.com", user = "student",
  password = "Sagehen47", dbname = "traffic")
```

```
dataset_names <- dbGetQuery(con, "SHOW TABLES")[[1]]

# remove datasets with "_" in the name
dataset_names <- dataset_names[str_detect(dataset_names, "_", negate = TRUE)]

query_sample <- function(dataset_str, percent){
  # input is dataset_str (str) with dataset name, and percent (dbl) for the random sample %
  # output is the dataframe with a column added for the name of dataset and character NA's
  # replaced with NA's
  # global variable con is the SQL connection

  command <- paste("SELECT * FROM", dataset_str, "WHERE rand() <=", percent,
                   # in SQL, filter for vehicular stops
```

```
                    " AND type = 'vehicular'",
                    sep = " ")

  df <- dbGetQuery(con, command) %>% mutate(dataset_name = dataset_str)

  # do not consider empty datasets
  if (dim(df)[1] == 0){
    return(NULL)
  }

  # replace character NA's with NA
  if (sum(is.na(df) == 0)){

    df[df == "NA"] = NA

  }

  return(df %>% dplyr::select(-type))

}

dataset_lst <- lapply(dataset_names, query_sample, 0.3)

# remove empty datasets through logical indexing
dataset_lst <- dataset_lst[sapply(dataset_lst, function(x) isTRUE(nrow(x) > 0))]
```

Cleaning data

Some columns are empty for all observations, so we want to remove them.

```
check_nonempty <- function(var, dataset, n_obsv){
  # helper function for removing empty columns

  # the function environment has the parameter dataset
  col_str <- paste("dataset$", var, sep = "")
  col <- eval(parse(text = col_str))
  isCollected <- sum(is.na(col)) < n_obsv

  return(isCollected)

}

remove_empty_col <- function(dataset){
  # a variable is 'collected' if there is a column for it in the dataset
  # but being collected doesn't imply nonempty

  collected_var <- names(dataset)
  n_obsv <- dim(dataset)[1]

  nonempty_bools <- unlist(lapply(collected_var, check_nonempty, dataset, n_obsv))

  # use logial indexing!
```

```
  nonempty_var <- collected_var[nonempty_bools]

  ## in case i need this information
  # empty_var <- collected_var[!nonempty_bools]

  return(dataset %>% dplyr::select({{ nonempty_var }}))

}

dataset_lst <- lapply(dataset_lst, remove_empty_col)
```

Functions for manipulating a list of dataframes

`myfilter_for` can be thought of as – I have a list of dataframes, and I am *filtering for* these particular variables in each dataframe. If I *need containment*, then I only look for the dataframes with ALL of the variables in `var_vect`.

```
myfilter_for <- function(dataset, var_vect, need_containment){
  # if need_containment is true, then function only returns
  # datasets containing ALL variables specified in var_vect

  # need_containment = TRUE results in more restrictive filtering

  dataset_var <- names(dataset)
  intersection <- var_vect[var_vect %in% dataset_var]

  if (need_containment){

    if (length(intersection) == length(var_vect)) {
      # embrace syntax from dplyr programming
      return(dataset %>% dplyr::select({{ var_vect }}))
    } else {
      return(NULL)
    }

  } else if (!need_containment){

    if (length(intersection > 0)) {
      # embrace syntax from dplyr programming
      return(dataset %>% dplyr::select({{ intersection }}))
    } else{
      return(NULL)
    }

  }

}
```

`dataset_containing` can be thought as – I have a list of dataframes, and I want to find the datasets that contain all these variables in `var_vect`. I still want the whole dataframe, though.

```
dataset_containing <- function(dataset, var_vect){
  # var_vect is str with the variables we WANT
```

```
  # returns the whole dataset

  if(var_vect %in% names(dataset)){
    return(dataset)

  } else {
    return(NULL)

  }

}
```

`mysearch_dataset` takes the dataset name in str and returns a single dataset. If the dataset name isn't in the list of dataframes, then this function will return an error of index out of bounds. Can be fixed in the future!

```
find_dataset <- function(dataset, name_str){

  # TODO == 0 or <= 1 ???
  # there's an occaisional error w this function that can be fixed!
  if(dim(dataset)[1] <= 1){
    return(NULL)
  }

  if(dataset$dataset_name[1] == name_str){
    return(dataset)
  }

}

mysearch_dataset <- function(dataset_list, name_str){

  df <- lapply(dataset_list, find_dataset, name_str)
  df <- df[sapply(df, function(x) isTRUE(nrow(x) > 0))]

  return(df[[1]])

}
```

`countMissing` is used to tally the NA values. We can exclude this function from counting NA values (missingness) in specified columns with `exclude_bool` and `exclude_var`.

```
check_missing <- function(n_threshold, df){

  col <- df %>%
    mutate("isMissing_{{ n_threshold }}" := case_when(missing >= n_threshold ~ TRUE,
                                                      TRUE ~ FALSE)) %>%
    select(starts_with("isMissing"))

  return(col)

}

countMissing <- function(dataset, n_threshold, exclude_bool, exclude_var){
  # <n_threshold> is used to classify the observations with
```

```
  # at least n_threshold missing values as completely missing

  # <exclude_var> is str specifying which variables we don't count for NA's

  n_var <- dim(dataset)[2]

  if (exclude_bool){
    missing <- list(missing = rowSums(is.na(dataset %>% select(-all_of(exclude_var)))))

  } else {
    missing <- list(missing = rowSums(is.na(dataset)))

  }

  dataset <- dataset %>%
    bind_cols(list(missing), .id = NULL) %>%
    mutate(stop_missing_rate = missing/n_var)

  dataset <- dataset %>%
    # check_missing operates on dataset with missingness already counted
    bind_cols(lapply(1:n_threshold, check_missing, dataset))

  return(as.data.frame(dataset))

}

missing_lst <- lapply(dataset_lst, countMissing, 1, FALSE)
```

About `dataset_lst`

Which variables are most frequently recorded across the datasets?

```
# 15 most frequently recorded variables
freq_var <- data.frame("var" = unlist(lapply(dataset_lst, function(dataset) names(dataset)))) %>%
  group_by(var) %>%
  summarize(count = n(), .groups = "drop") %>%
  # drop type (either pedestrian or vehicular)
  filter(var != "type" & str_detect(var, "row", negate = TRUE)) %>%
  # n = 16 because dataset_name is a variable
  slice_max(count, n = 16) %>%
  pull(var)

freq_var
```

```
##  [1] "dataset_name"     "date"             "subject_race"     "location"
##  [5] "time"             "subject_sex"      "outcome"          "citation_issued"
##  [9] "lat"              "lng"              "subject_age"      "arrest_made"
## [13] "warning_issued"   "violation"        "search_conducted" "officer_id_hash"
```

How many datasets record a certain variable?

```
count_datasets <- function(dataset_lst, var_vect){

  dataset_lst <- lapply(dataset_lst, myfilter_for, var_vect, TRUE)
  dataset_lst <- dataset_lst[sapply(dataset_lst, function(x) isTRUE(nrow(x) > 0))]

  return(length(dataset_lst))

}

count_datasets(dataset_lst, "search_conducted")
```

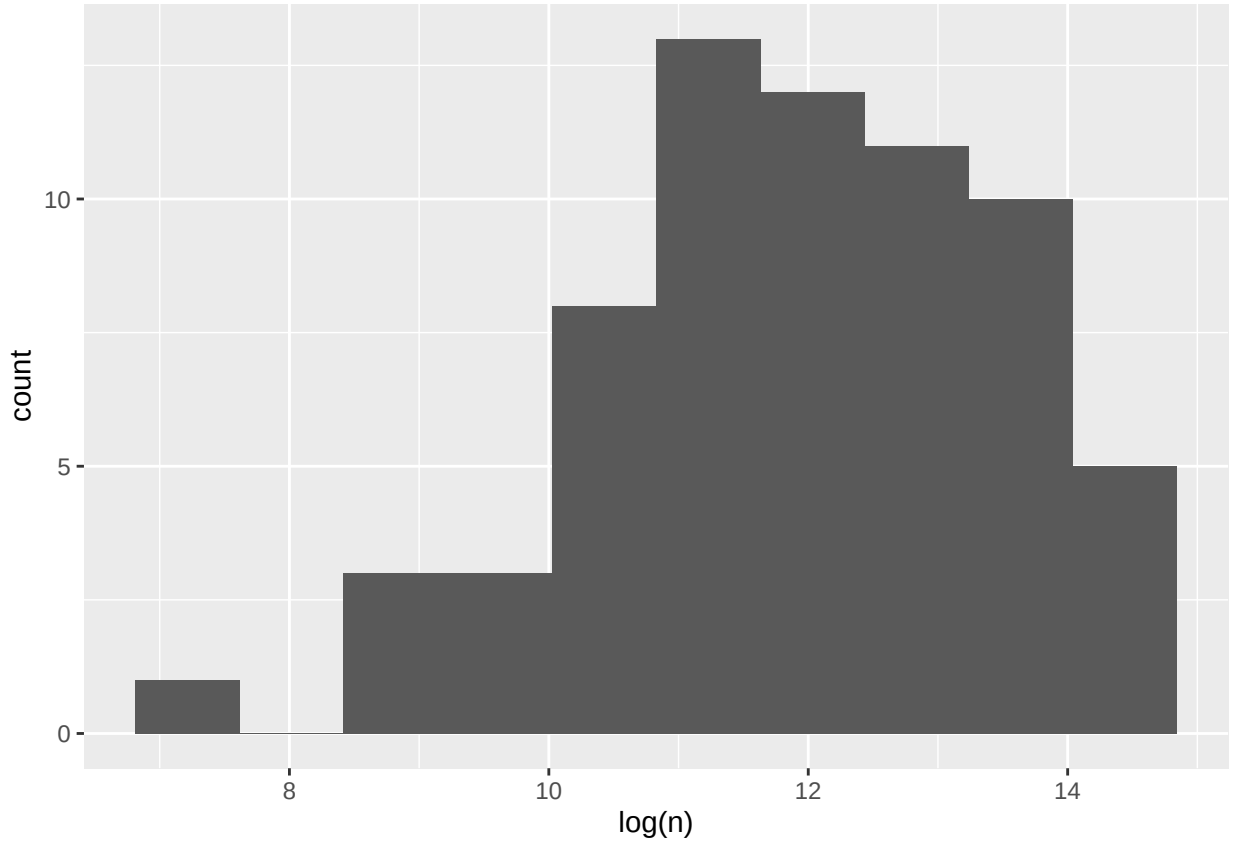
```
## [1] 32
```

```
count_datasets(dataset_lst, "arrest_made")
```

```
## [1] 36
```

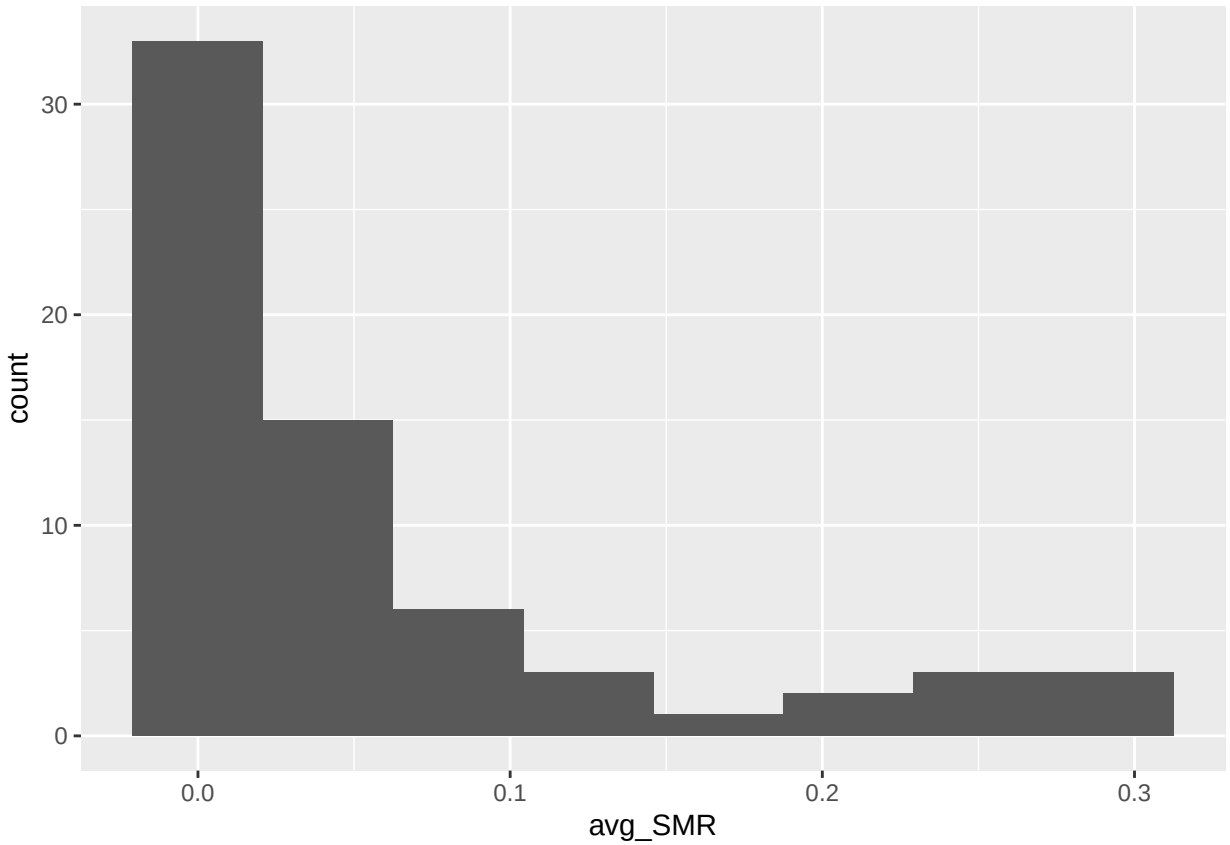
What is the distribution of the number of observations per dataset?

```
data.frame(n = sapply(dataset_lst, function(x) dim(x)[1])) %>%
  ggplot(aes(x = log(n))) +
  geom_histogram(bins = 10)
```



What is the distribution of the SMR per dataset?

```
ggplot_datasetSMR <- function(dataset_lst, var_vect){

  # only select the frequent variables to better compare!
  error_lst <- lapply(dataset_lst, myfilter_for, var_vect, FALSE)

  summarizeAvgSMR <- function(dataset){

    dataset <- dataset %>%
      group_by(dataset_name) %>%
      summarize(avg_SMR = mean(stop_missing_rate))

  }

  error_lst <- lapply(error_lst, countMissing, 1, FALSE)
  error_lst <- lapply(error_lst, summarizeAvgSMR)

  p <- bind_rows(error_lst) %>%
    ggplot(aes(x = avg_SMR)) +
    geom_histogram(bins = 8)

  return(p)

}

ggplot_datasetSMR(dataset_lst, freq_var)
```


30
20
10
0
count
0.0
0.1
0.2
0.3
avg_SMR