2 Refactored race and outcome

Amber Lee

11/21/2020

https://stackoverflow.com/questions/1249548/side-by-side-plots-with-ggplot2

Set up

```
library(RMySQL)

library(tidyverse)
library(tidyr)
# library(broom)
library(lubridate)
library(stringr)
# library(tidycensus)
library(kableExtra)
# library(geofacet)
# library(naniar) # replace "NA" with NA, except too slow, use baseR solution
# library(revgeo)
```

```
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))]
```

```
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)
```

```
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 <- 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)

  }

}
```

```
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)
  }
```

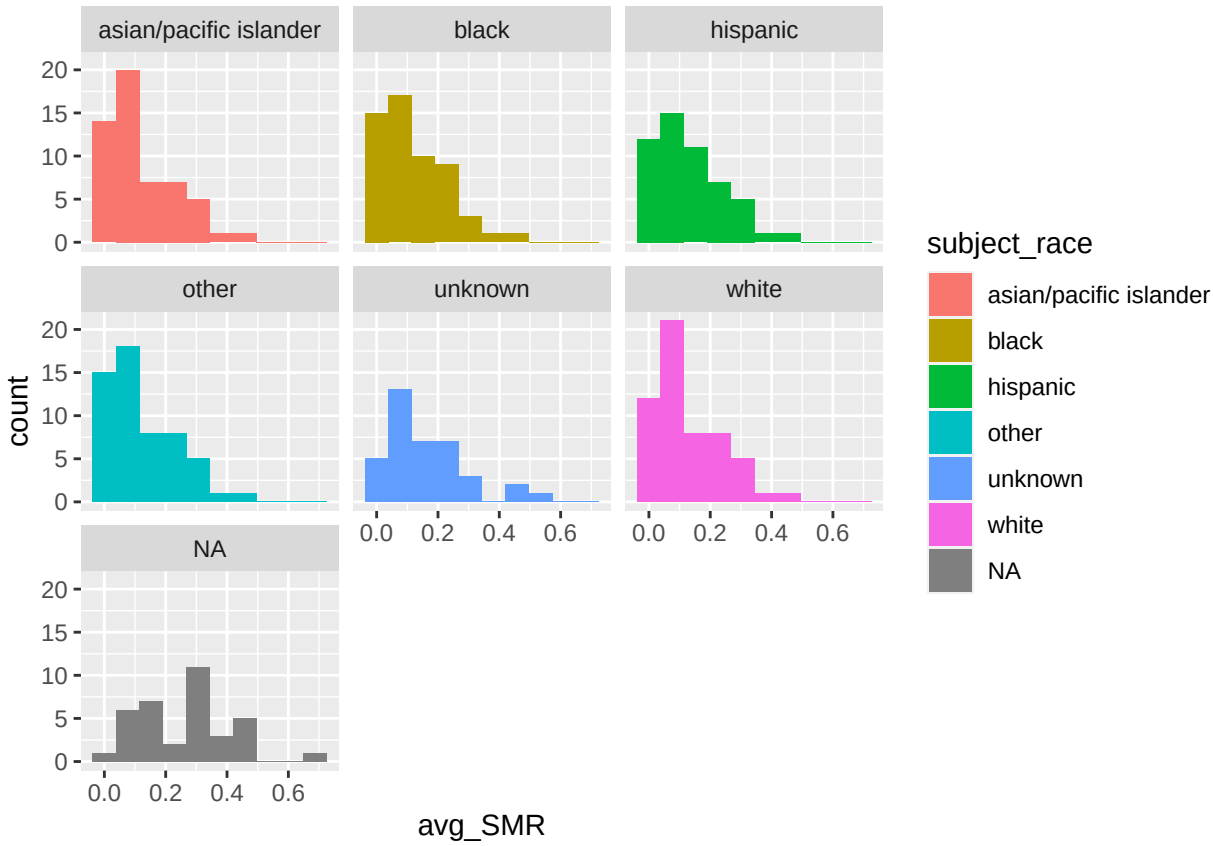
```
}

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]])

}
```

```
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)
```

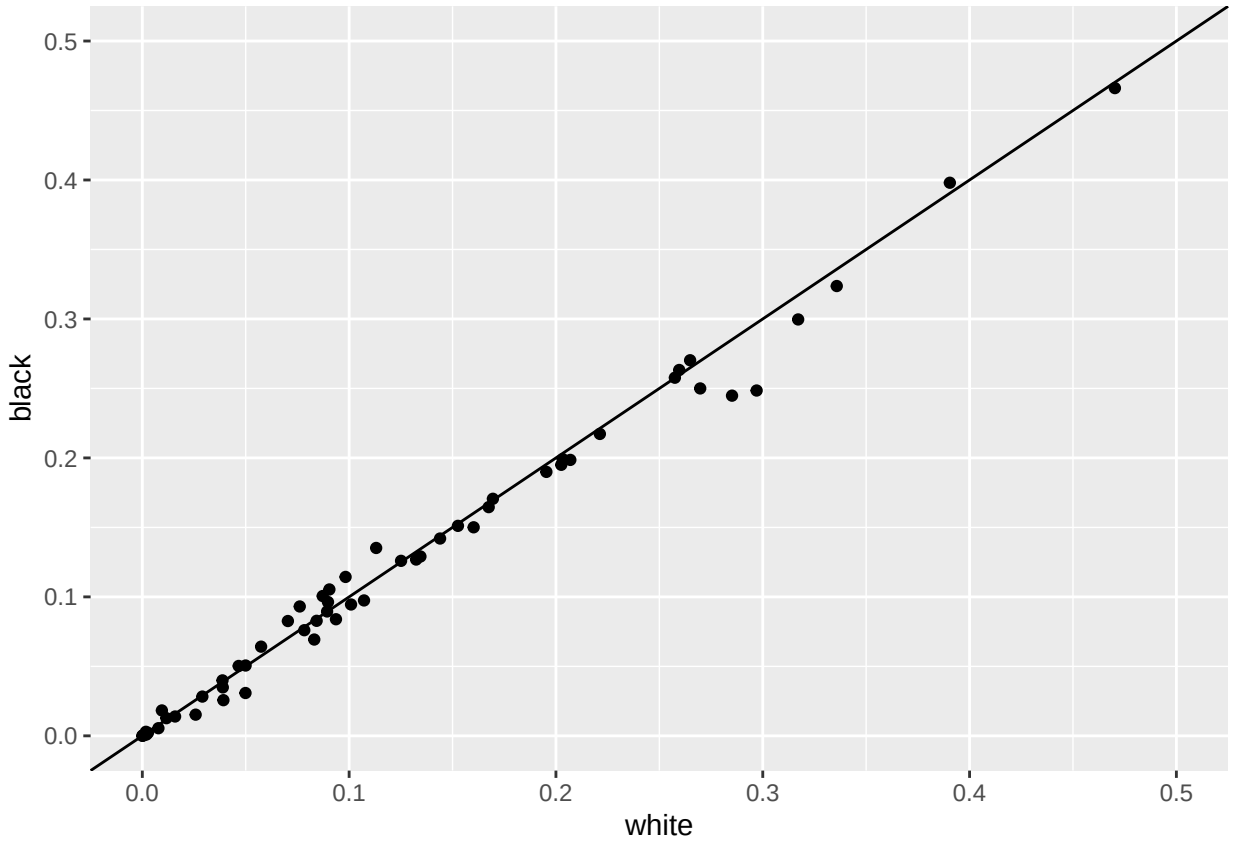
SMR by race

```
makeDF.SMRByRace <- function(dataset_lst){
  # input: list of datasets

  #helper function
  summarizeSMRByRace <- function(dataset){
    # input: <dataset> with missingness counted

    dataset <- dataset %>%
      ungroup() %>%
      group_by(subject_race, dataset_name) %>%
      summarize(avg_SMR = mean(stop_missing_rate), .groups = "drop")

    return(dataset)

  }

  race_lst <- lapply(dataset_lst, dataset_containing, "subject_race")
  race_lst <- race_lst[sapply(race_lst, function(x) isTRUE(nrow(x) > 0))]

  race_lst <- lapply(race_lst, countMissing, 1, TRUE, "subject_race")

  # filter out and count erorrs
  race_df <- bind_rows(lapply(race_lst, summarizeSMRByRace))
  # %>%
  #   spread(key = isRecorded, value = avg_SMR)

  return(race_df)

}

smrRace <- makeDF.SMRByRace(dataset_lst)
```

```
ggplot(data = smrRace, aes(x = avg_SMR, fill = subject_race)) +
  geom_histogram(bins = 10) +
  facet_wrap(~ subject_race)
```




```
smrRace %>%
  spread(key = subject_race, value = avg_SMR) %>%
  ggplot(aes(x = white, y = black)) +
  geom_point() +
  geom_abline() +
  scale_x_continuous(limits = c(0, 0.5)) +
  scale_y_continuous(limits = c(0, 0.5))
```




```
# todo make this into a square
```

Race and outcome analysis: outcome rates by race groups

Race and outcome analysis does not use SMR. Instead, it looks at outcome rates (search rates, arrest rates, etc.) for all race groups include NA race. The motivation for this section is: are NA-race drivers treated more like minority race or white drivers?

Clean outcomes variable

One hot encoding for the levels of `outcome` ("warning_issued", "citation_issued", "arrest_made", "search_conducted", "summon_issued") exist in the datasets recording such variables. They are recorded as doubles or characters (not booleans). We turn them into doubles for consistency

```
how_are_outcomes_recorded <- function(outcome_str, dataset_lst, want_type){

  outcome_sym <- sym(outcome_str)

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

  if (want_type) {

    outcomes <- lapply(outcome_lst, function(x) typeof(x[1,]))
```

```
    return(outcomes)

  }

  outcomes_lst <- lapply(outcome_lst,
                         function(x) x %>% distinct(!!outcome_sym) %>% pull(!!outcome_sym))

  return(outcomes_lst)

}

outcomes_type <- lapply(c("warning_issued", "citation_issued", "arrest_made", "search_conducted", "summ

outcomes_recorded <- lapply(list("warning_issued", "citation_issued", "arrest_made", "search_conducted"

data.frame(type = unlist(outcomes_type)) %>% distinct()
```

```
##        type
## 1    double
## 2 character
```

```
data.frame(recorded = unlist(outcomes_recorded)) %>% distinct()
```

```
##   recorded
## 1        0
## 2        1
## 3     TRUE
## 4    FALSE
## 5     <NA>
```

```
outcome_clean <- function(dataset, str_outcome){

    if(typeof(dataset[[str_outcome]]) == "character"){
      sym_outcome = sym(str_outcome)

      dataset <- dataset %>%
        mutate(!!sym_outcome := case_when(!!sym_outcome == "FALSE" ~ 0,
                                          !!sym_outcome == "TRUE" ~ 1,
                                          !!sym_outcome == "0" ~ 0,
                                          !!sym_outcome == "1" ~ 1))
    }

  return(dataset)

}
```

Outcomes by Race

For each type of outcome, we create a dataframe with the following variables/columns. For the sake of explaining, I take our outcome level of interest as the `search_conducted` variable.

- `race.sc` is the search conducted rate for drivers identified of that race. This includes when drivers are identified as NA (`missing race.sc`).

- `race.scNA` is the rate of the search conducted variable *being missing/unrecorded* for drivers identified of that race. Drivers whose race information and search conducted are both missing are recorded under `missing race.scNA`.

The denominator for `race.sc` and `race.scNA` is the total number of drivers in that race group.

The same naming convention will be used for wi (warning issued), ci (citation issued), and am (arrest made) as well.

The data frame will also contain:

- `total_stops` is the total number of stops
- `total_raceNA` is the total number of stops with missing race
- `total_scNA` the total number of stops with the `search_conducted` variable unrecorded (same naming convention for wi, ci, am).

```
outcome_abbreviation <- function(str_outcome, for_NA){
  # returns an abbreviation for str_outcome
  # for example, arrest_made becomes .am

  first_word <- str_split(str_outcome, "_")[[1]][1]
  second_word <- str_split(str_outcome, "_")[[1]][2]

  abb <- paste(".", substr(first_word, 1, 1), substr(second_word, 1, 1), sep = "")

  if(!for_NA){
    return(abb)
  }

  # if for_NA, then tack on an NA
  abbNA <- paste(abb, "NA", sep = "")
  return(abbNA)

}

# outcome_abbreviation("warning_issued", TRUE)
```

```
makeDF.RaceOutcome <- function(str_outcome, dataset_lst){

  sym_outcome <- sym(str_outcome)

  # 1. filter for 3 variables
  outcome_lst <- lapply(dataset_lst, myfilter_for,
                        c("dataset_name", "subject_race", str_outcome), TRUE)
  outcome_lst <- outcome_lst[sapply(outcome_lst, function(x) isTRUE(nrow(x) > 0 ))]

  # 2. clean
  outcome_lst <- lapply(outcome_lst, outcome_clean, str_outcome)

  # 3. make df
  outcome_df <- data.frame(bind_rows(outcome_lst))

  # 4. numerator by counting total stops per outcome level and race group
  outcome_counts <- outcome_df %>%
    group_by(dataset_name, subject_race, !!sym_outcome) %>%
    summarize(count = n(), .groups = "drop") %>%
```

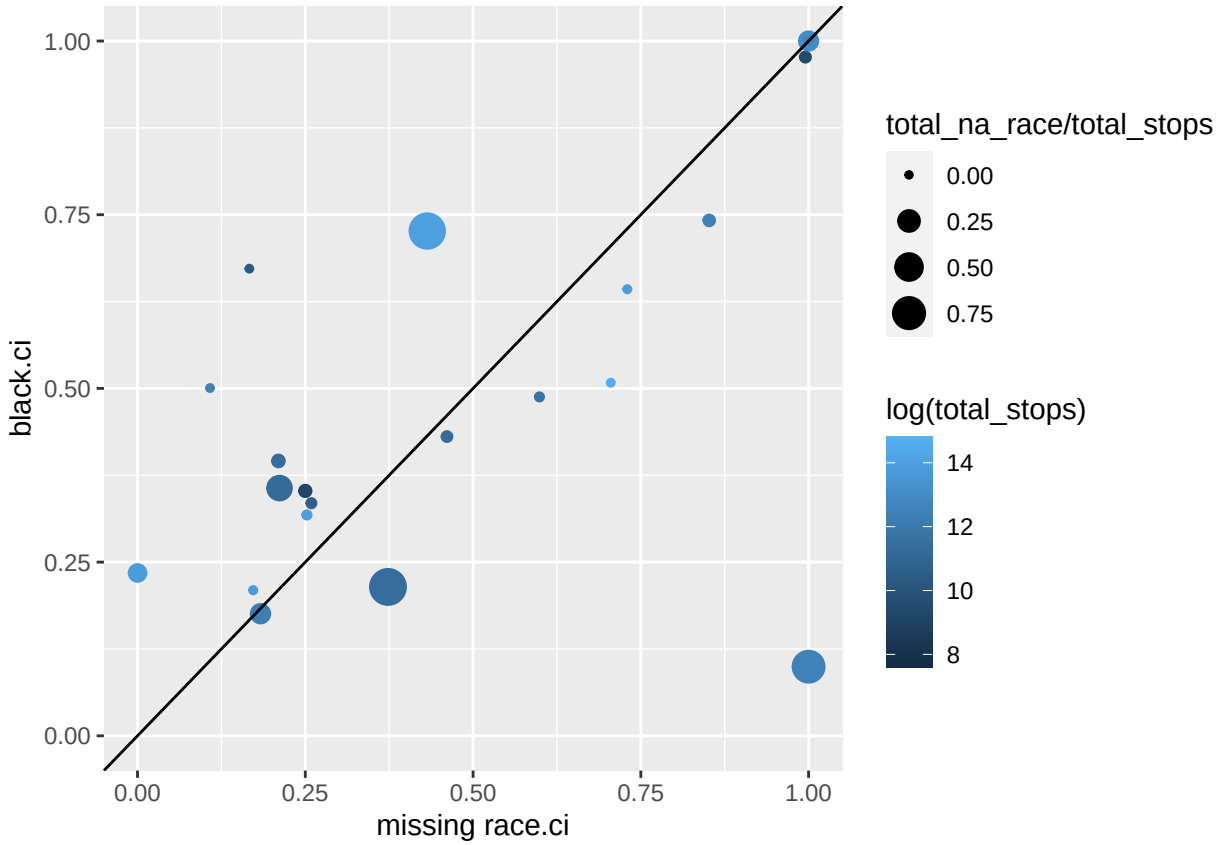
```r
  spread(key = !!sym_outcome, value = count)

# 5. denominators and other relevant statistics

# 5a. total stops per data set
total_stops <- outcome_df %>%
  group_by(dataset_name) %>%
  summarize(total_stops = n(), .groups = "drop")

# 5b. total stops per racial group
total_stops_race <- outcome_df %>%
  group_by(dataset_name, subject_race) %>%
  summarize(total_stops_race = n(), .groups = "drop")

# 5c. total stops NA (race and outcome)
total_na_race <- total_stops_race %>%
  filter(is.na(subject_race)) %>%
  rename(total_na_race = total_stops_race) %>%
  select(-subject_race)

total_na_outcome <- outcome_counts %>%
  group_by(dataset_name) %>%
  summarize(total_na_outcome = sum(`<NA>`, na.rm = TRUE), .groups = "drop")

# 6. calculate rates
all_rates <- outcome_counts %>%
  left_join(total_stops_race, by = c("dataset_name", "subject_race")) %>%
  # outcome == 1 denotes that the outcome happened
  # ex: search_conducted == 1 means that a search was conducted
  mutate(outcome_rate = `1` / total_stops_race,
         NA_outcome_rate = `<NA>` / total_stops_race) %>%
  select(dataset_name, subject_race, outcome_rate, NA_outcome_rate)

sc_rate <- all_rates %>%
  select(-NA_outcome_rate) %>%
  spread(key = subject_race, value = outcome_rate) %>%
  rename(`missing race` = `<NA>`)

scNA_rate <- all_rates %>%
  select(-outcome_rate) %>%
  spread(key = subject_race, value = NA_outcome_rate) %>%
  rename(`missing race` = `<NA>`)

# 7a. make abbreviation
abb <- outcome_abbreviation(str_outcome, FALSE)
abbNA <- outcome_abbreviation(str_outcome, TRUE)

# 7b. join
outcome_race_df <- sc_rate %>%
  left_join(scNA_rate, by = "dataset_name", suffix = c(abb, abbNA)) %>%
  left_join(total_stops, by = "dataset_name") %>%
  left_join(total_na_race, by = "dataset_name") %>%
  mutate(total_na_race = ifelse(is.na(total_na_race), 0, total_na_race)) %>%
```

```
    left_join(total_na_outcome, by = "dataset_name") %>%
    mutate(outcome = paste(unlist(str_split(str_outcome, "_")), collapse = " "))

  return(outcome_race_df)

}

raceoutcomes_lst <- lapply(c("warning_issued", "citation_issued",
                             "search_conducted", "arrest_made"),
                           makeDF.RaceOutcome, dataset_lst)
```

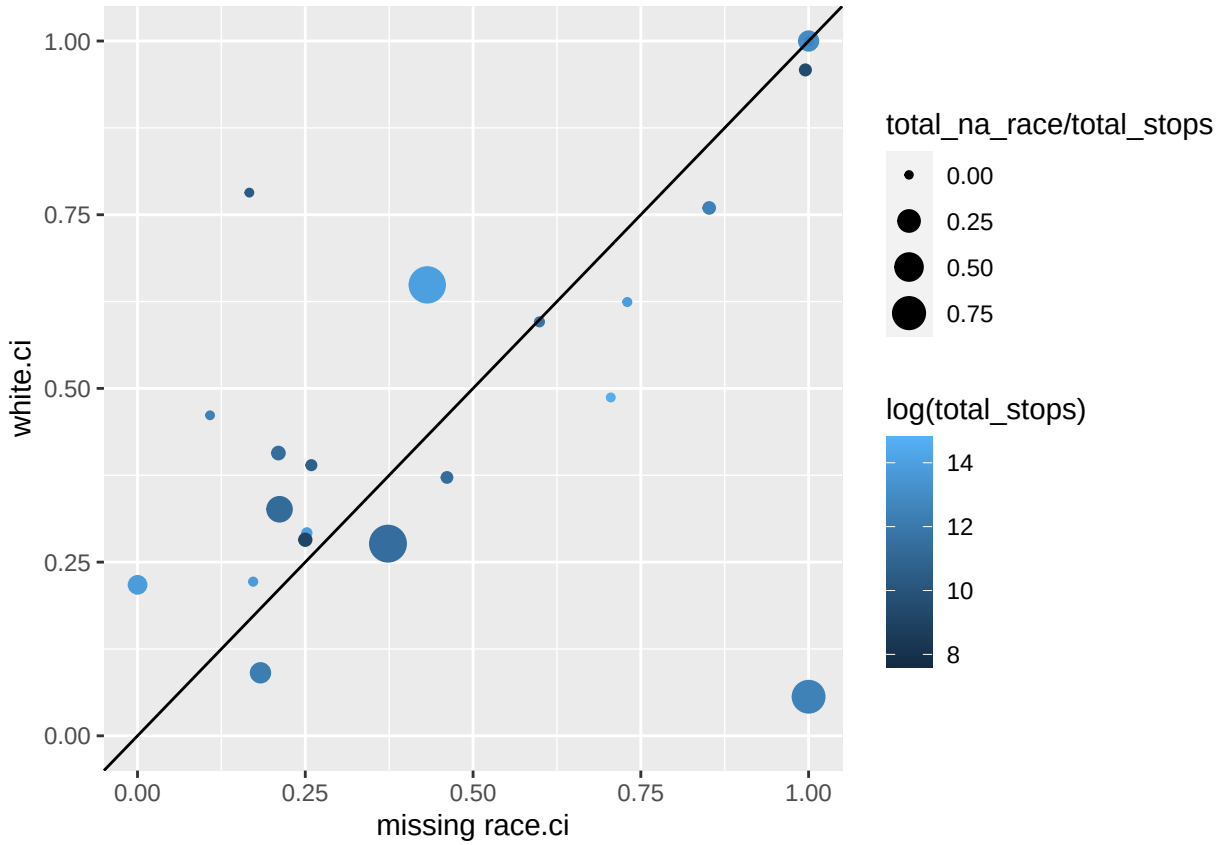
Plot

one dataset at a time

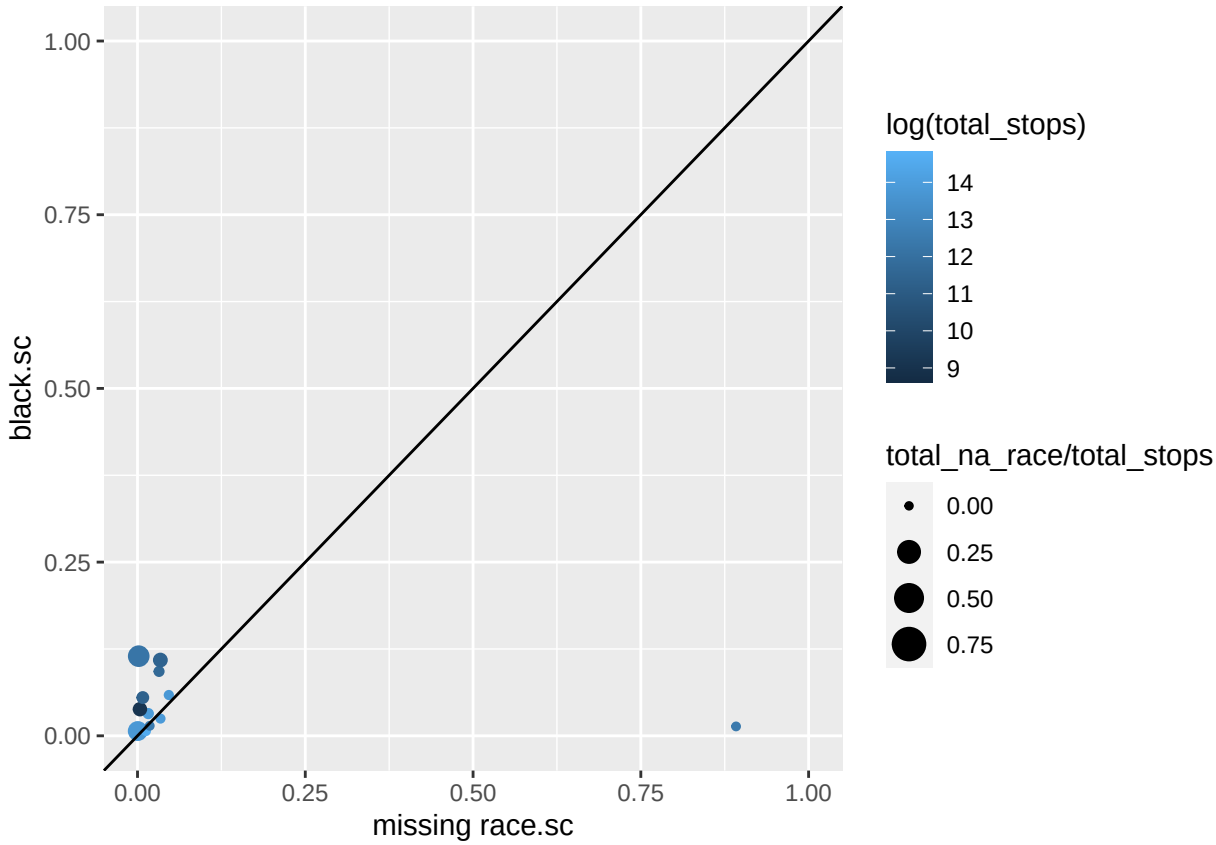
```
raceci_df <- raceoutcomes_lst[[2]]
racesc_df <- raceoutcomes_lst[[3]]
raceam_df <- raceoutcomes_lst[[4]]

default.point <- list(geom_point(aes(size = total_na_race/total_stops,
                                     color = log(total_stops))),
  geom_abline(),
  scale_x_continuous(limits = c(0, 1)),
  scale_y_continuous(limits = c(0, 1)))

ggplot(data = raceci_df, aes(x = `missing race.ci`, y = black.ci)) +
  default.point
```



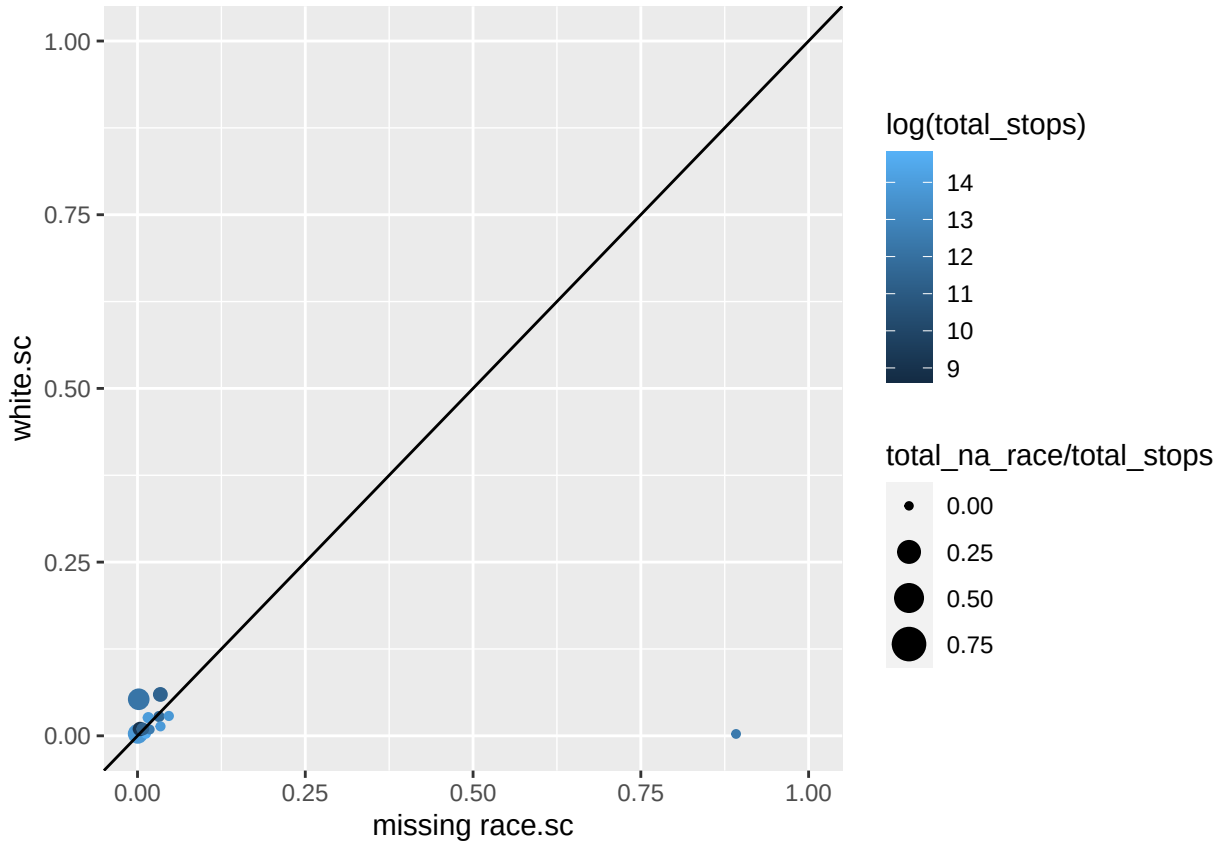

```
ggplot(data = raceci_df, aes(x = `missing race.ci`, y = white.ci)) +
  default.point
```



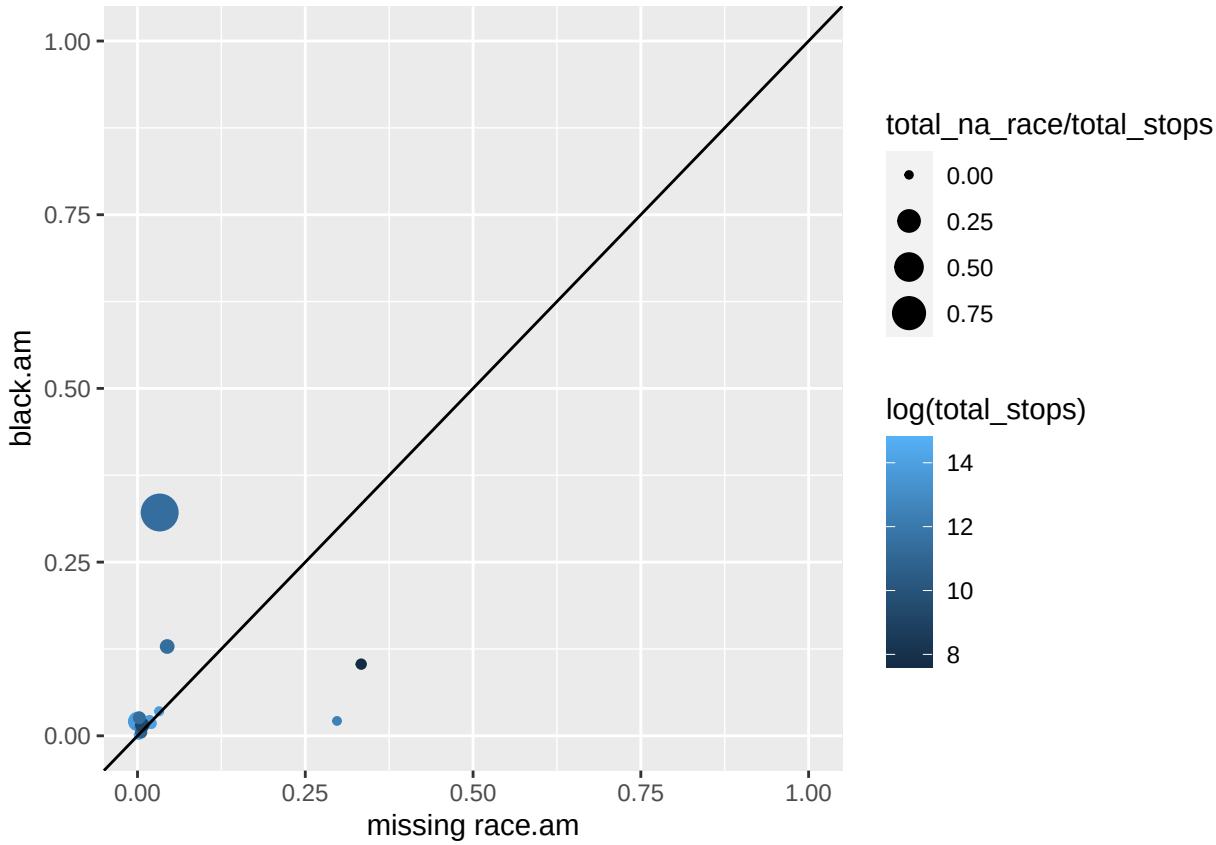

```
ggplot(data = racesc_df, aes(x = `missing race.sc`, y = black.sc)) +
  default.point
```



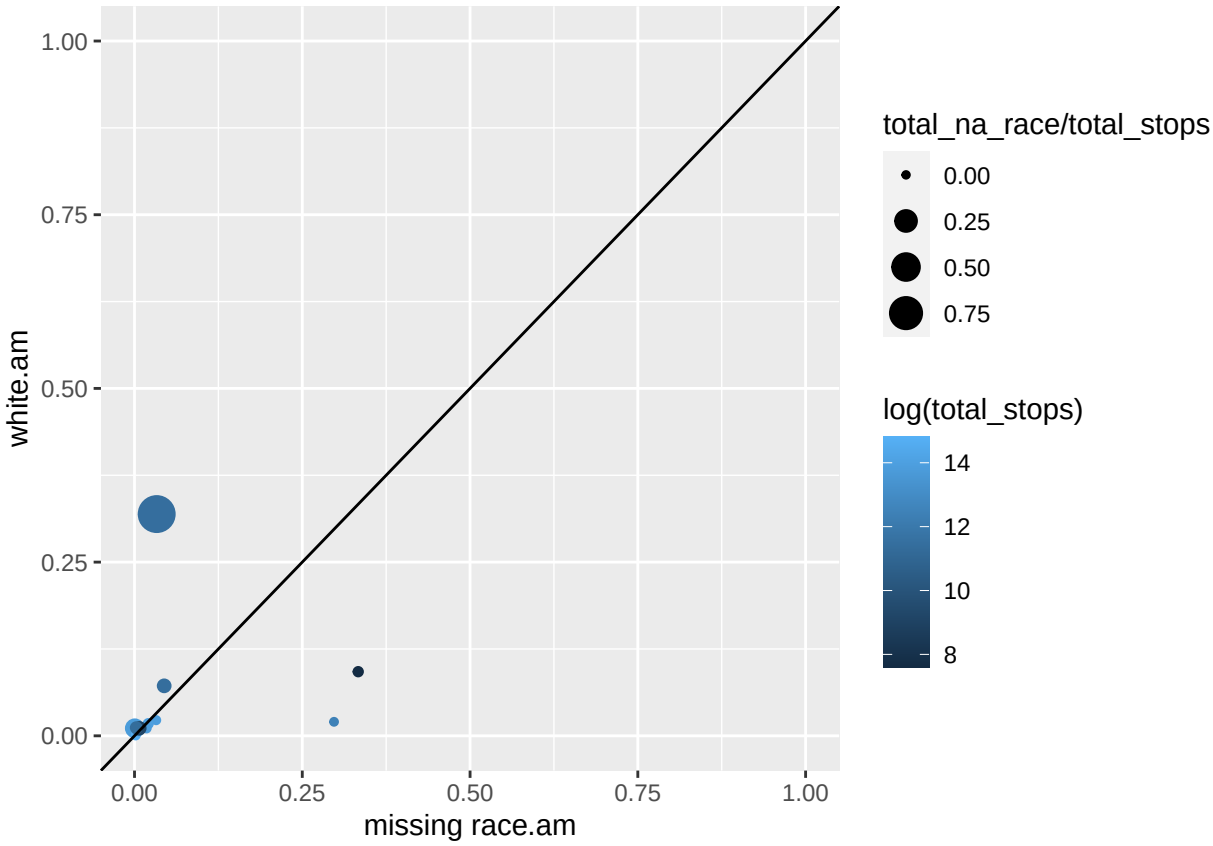

```
ggplot(data = racesc_df, aes(x = `missing race.sc`, y = white.sc)) +
  default.point
```




```
ggplot(data = raceam_df, aes(x = `missing race.am`, y = black.am)) +
  default.point
```




```
ggplot(data = raceam_df, aes(x = `missing race.am`, y = white.am)) +
  default.point
```



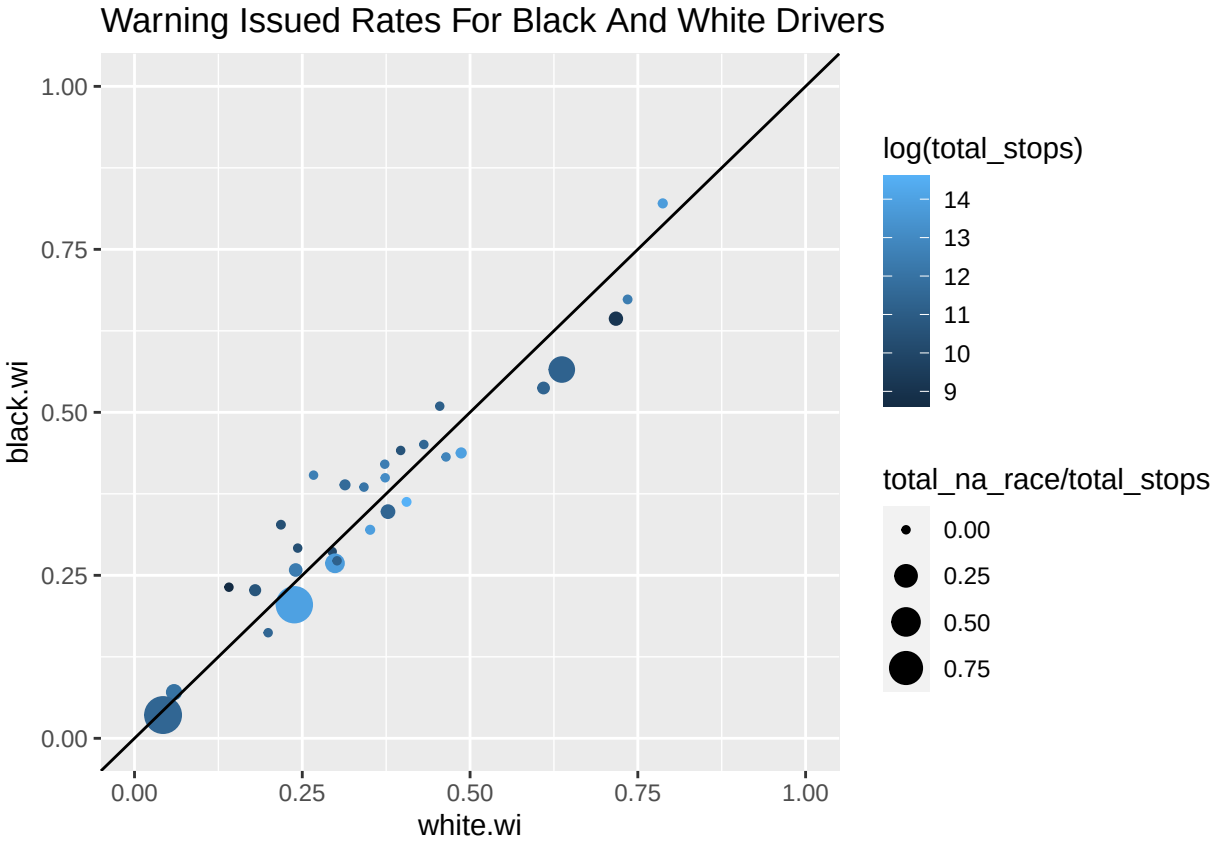
ggplot_raceoutcome plots the outcome rates for a fixed x and y axis race group

This function needs to editted so that race (as opposed to the outcome level) is iterated through. It is easier to comapre the treatment of Black and white drivers when the *same outcome level* is used to plot Black vs NA driver and white vs NA driver outcome rates.

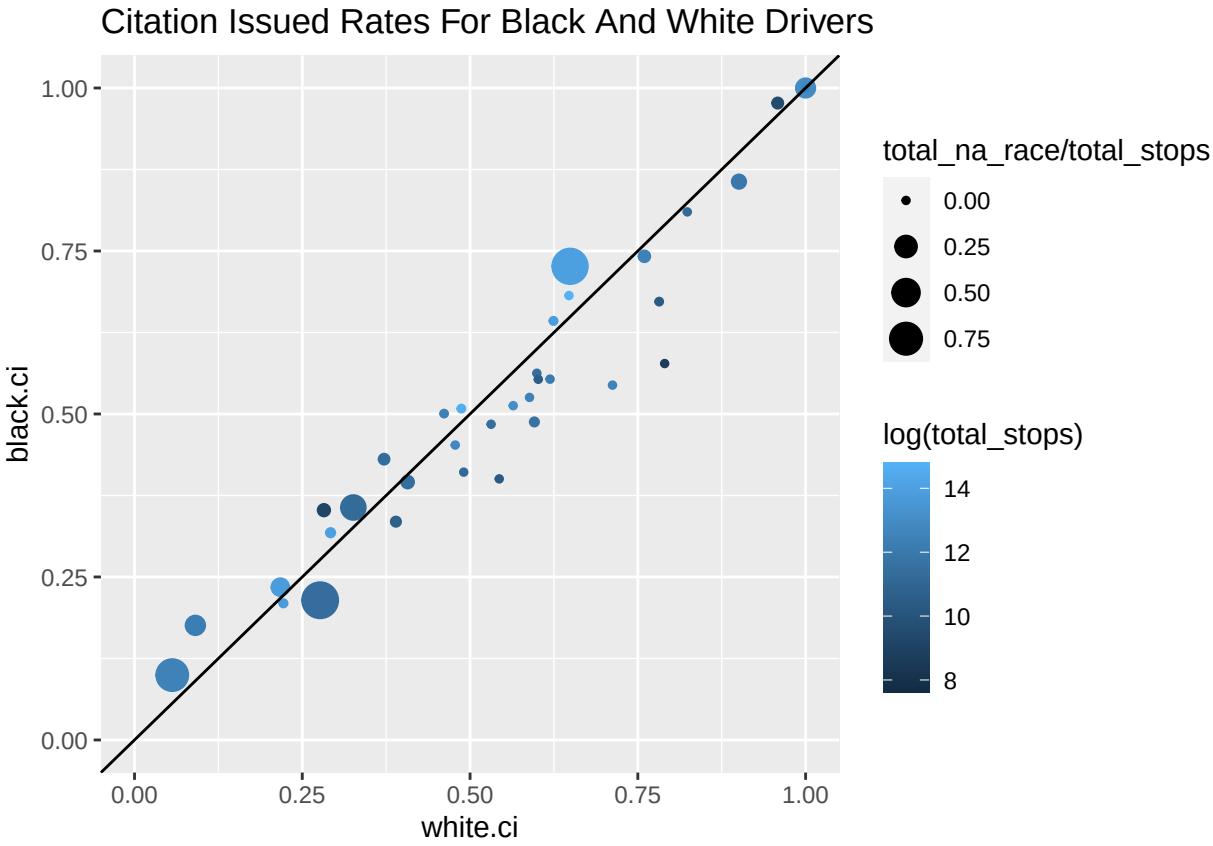
```
ggplot_raceoutcome <- function(dataset, str_x, str_y, plot_NA,
                               scale_x, scale_y, save_plot){

  var <- names(dataset)
  # keep NA's if plot_NA true (so negate would be false)
  var <- var[str_detect(var, "NA", negate = !plot_NA)]

  x_axis <- sym(var[str_detect(var, str_x)])
  y_axis <- sym(var[str_detect(var, str_y)])

  # string manipulation to make the title
  title <- paste(dataset$outcome[1], "rates for", str_y, "and", str_x, "drivers", sep = " ")
  title <- lapply(str_split(title, " "),
                  function(x) paste(toupper(substr(x, 1, 1)),
                                    substr(x, 2, length(x)), sep = ""))
  title <- paste(unlist(title), collapse = " ")

  p <- ggplot(dataset, aes(x = !!x_axis, y = !!y_axis)) +
    geom_point(aes(size = total_na_race/total_stops, color = log(total_stops))) +
    geom_abline() +
```

```
    scale_x_continuous(limits = scale_x) +
    scale_y_continuous(limits = scale_y) +
    ggtitle(title)

  if (save_plot) {ggsave(paste(title, ".png", sep = ""), plot = p)}

  return(p)

}

lapply(raceoutcomes_lst, ggplot_raceoutcome, "white", "black", FALSE, c(0, 1), c(0, 1), FALSE)
```

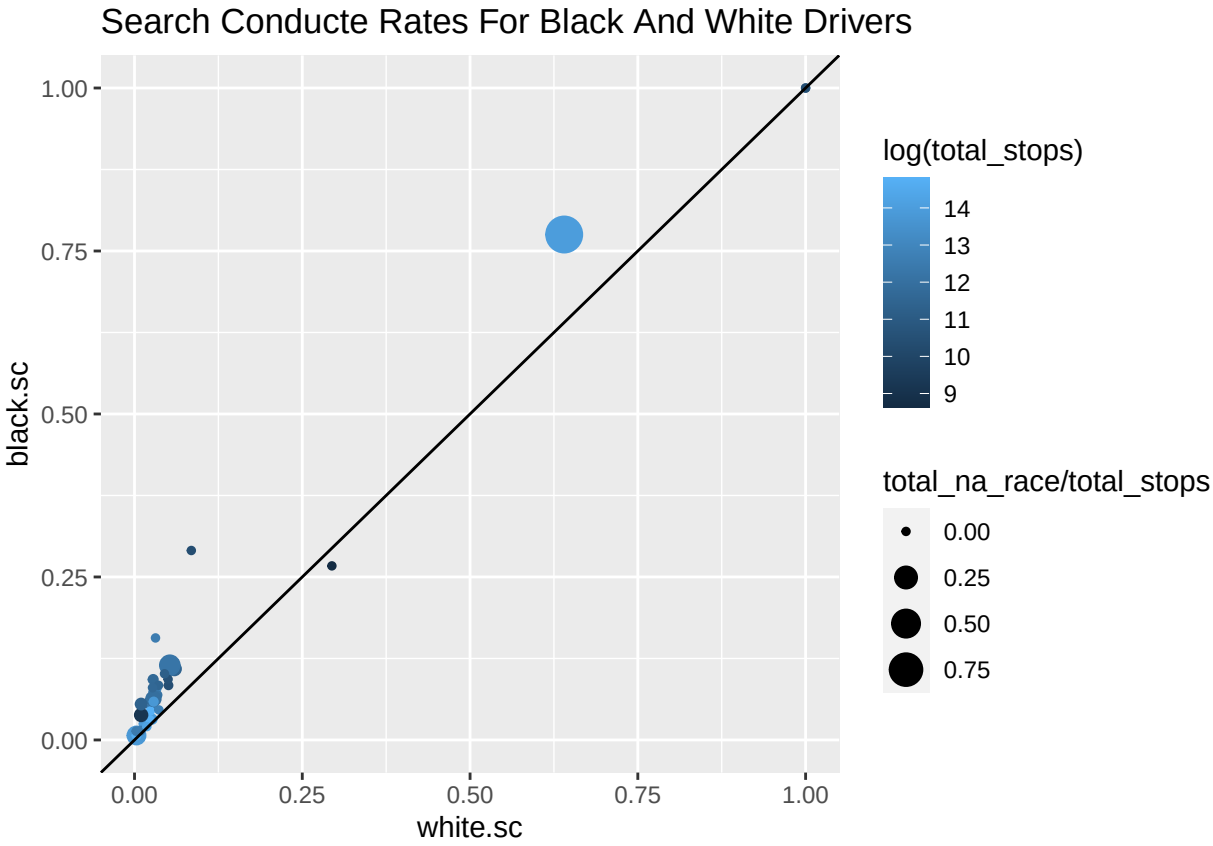
```
## [[1]]
```



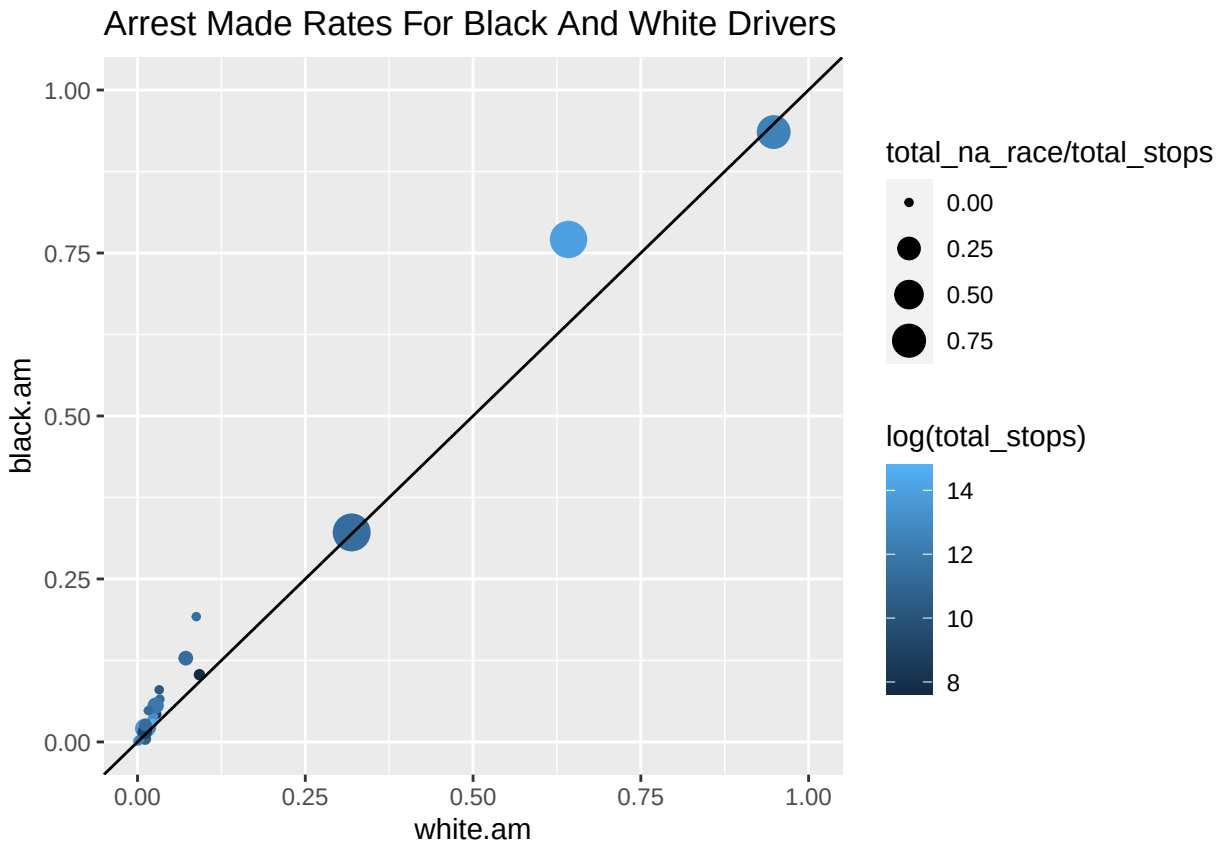

```
##
## [[2]]
```


Citation Issued Rates For Black And White Drivers
black.ci
white.ci
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
total_na_race/total_stops
0.00
0.25
0.50
0.75
log(total_stops)
14
12
10
8

```
##
## [[3]]
```

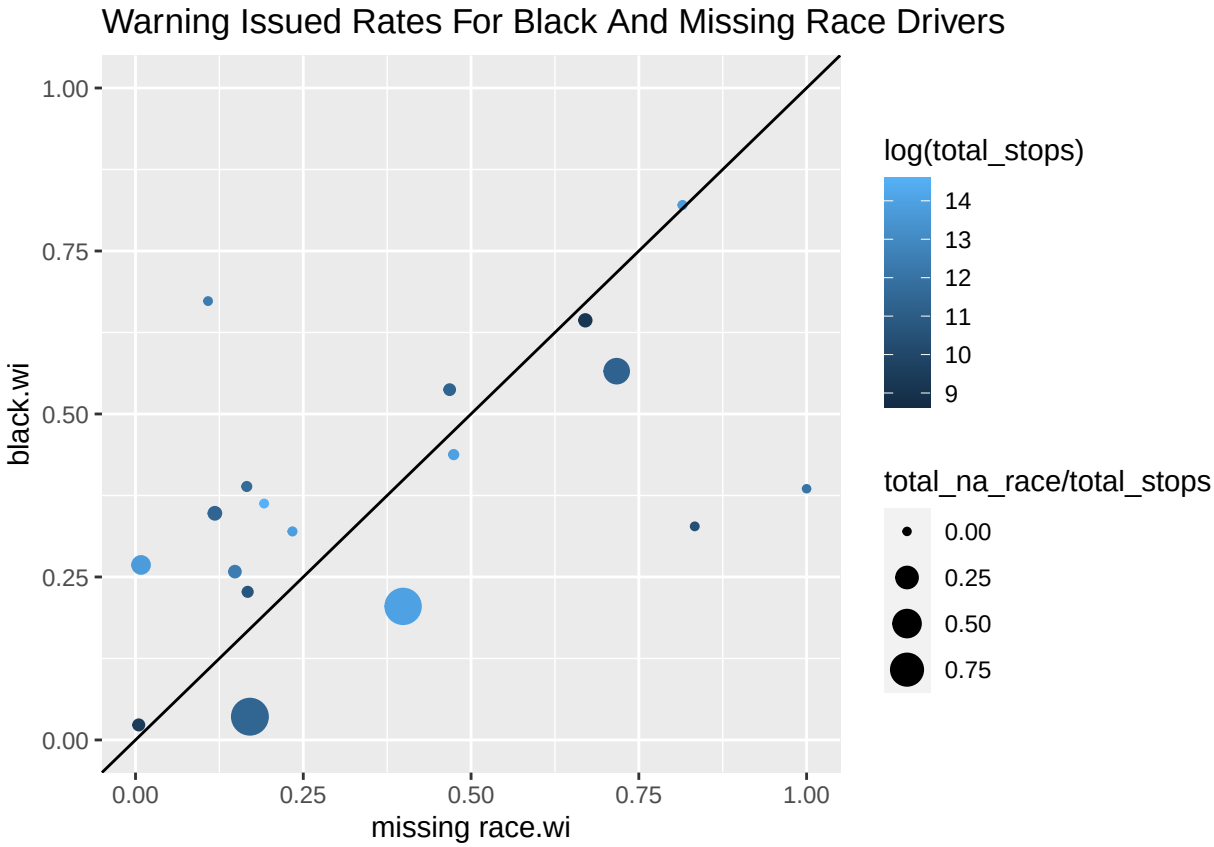

Search Conducte Rates For Black And White Drivers
1.00
0.75
0.50
0.25
0.00
black.sc
0.00
0.25
0.50
0.75
1.00
white.sc
log(total_stops)
14
13
12
11
10
9
total_na_race/total_stops
0.00
0.25
0.50
0.75

```
##
## [[4]]
```

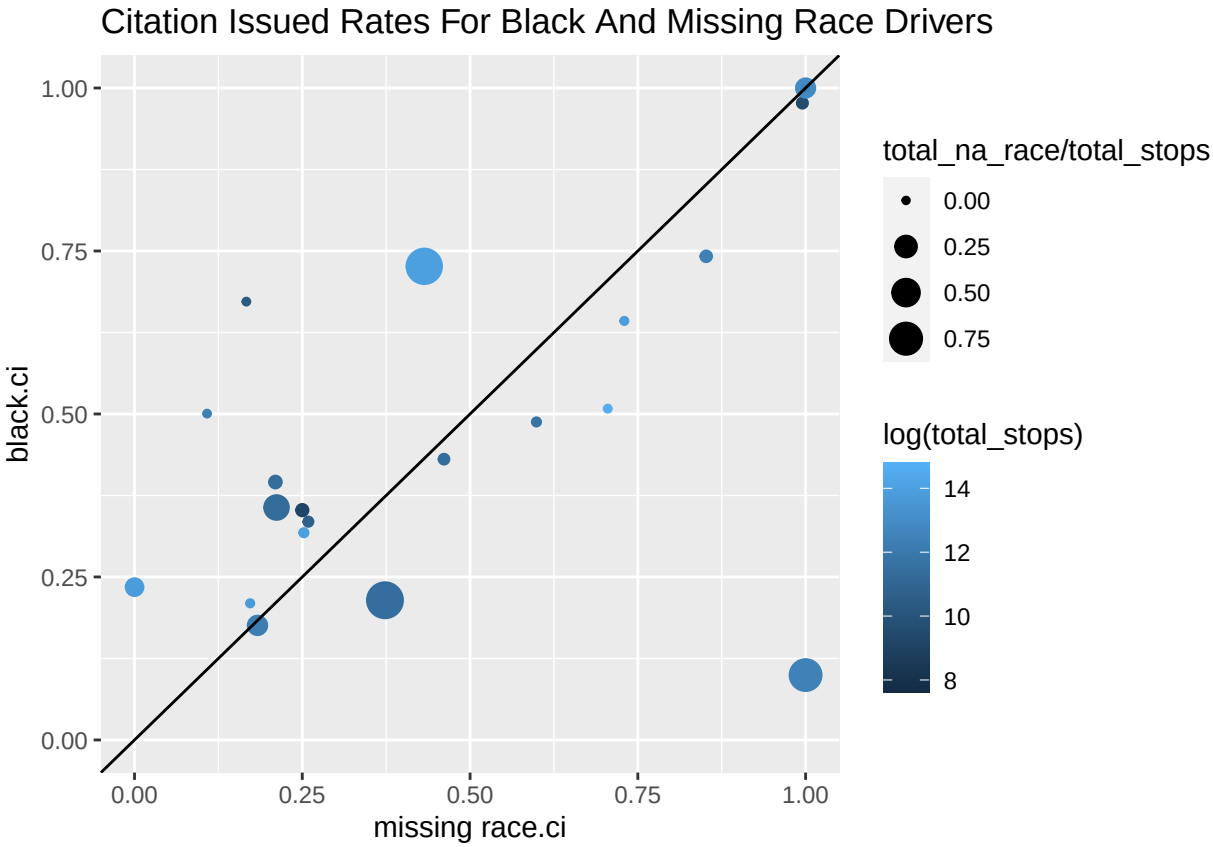



```
lapply(raceoutcomes_lst, ggplot_raceoutcome, "missing race", "black", FALSE, c(0, 1), c(0, 1), FALSE)
```

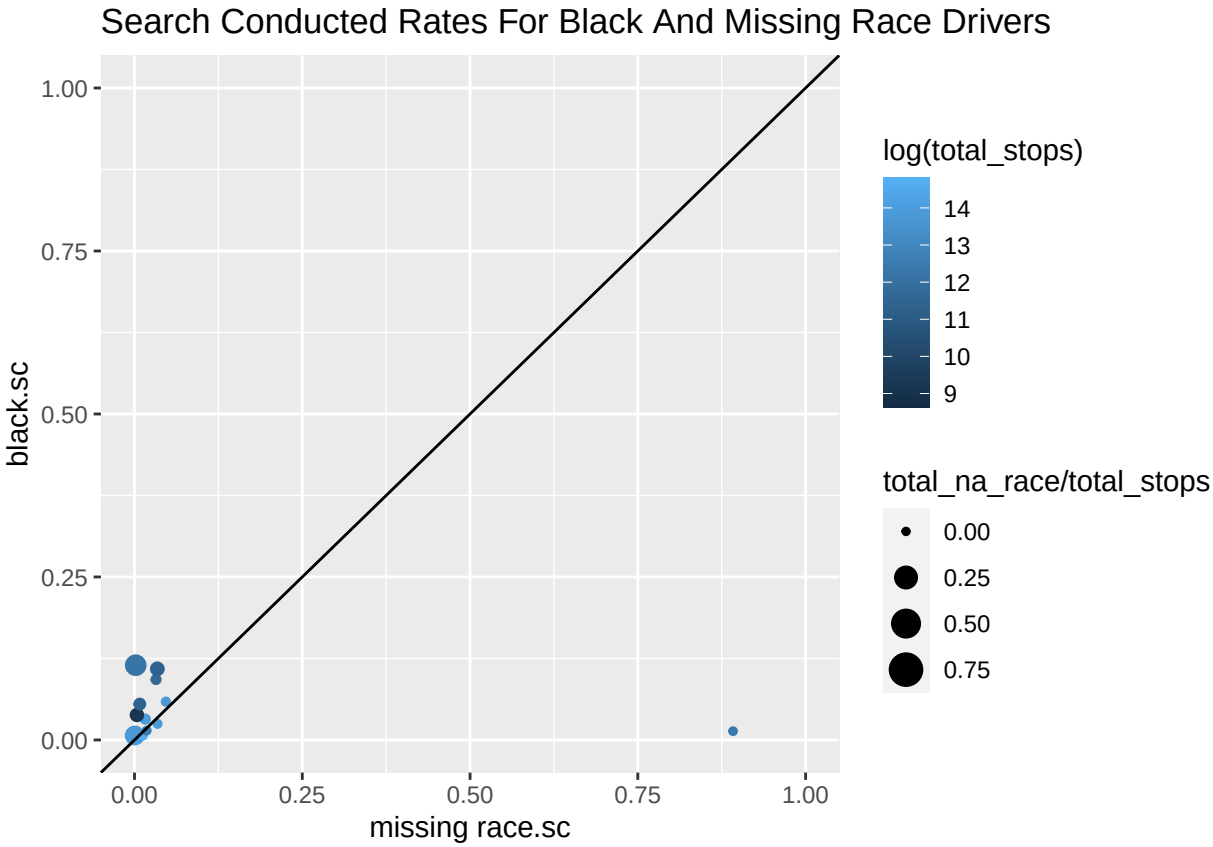
```
## [[1]]
```


Warning Issued Rates For Black And Missing Race Drivers
black.wi
missing race.wi
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
log(total_stops)
14
13
12
11
10
9
total_na_race/total_stops
0.00
0.25
0.50
0.75

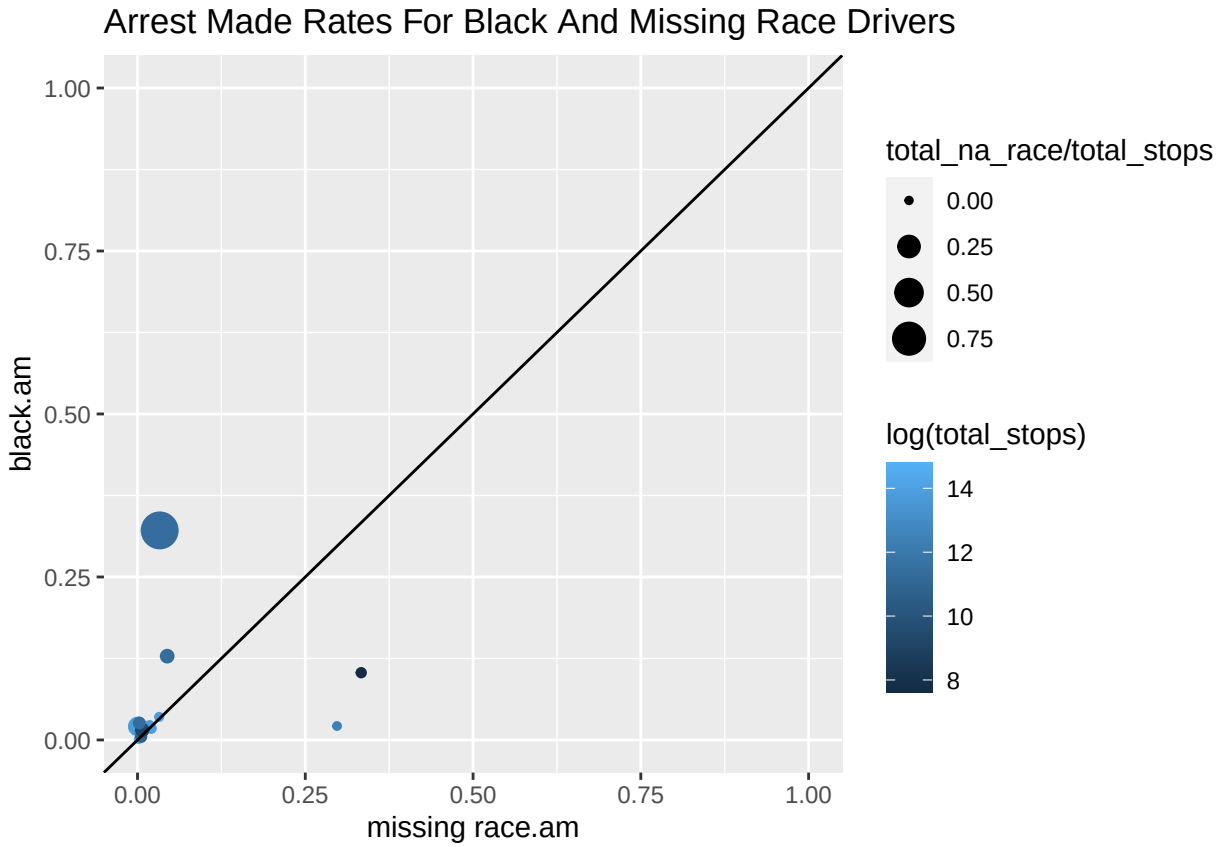
```
##
## [[2]]
```


Citation Issued Rates For Black And Missing Race Drivers
black.ci
missing race.ci
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
total_na_race/total_stops
0.00
0.25
0.50
0.75
log(total_stops)
14
12
10
8

```
##
## [[3]]
```

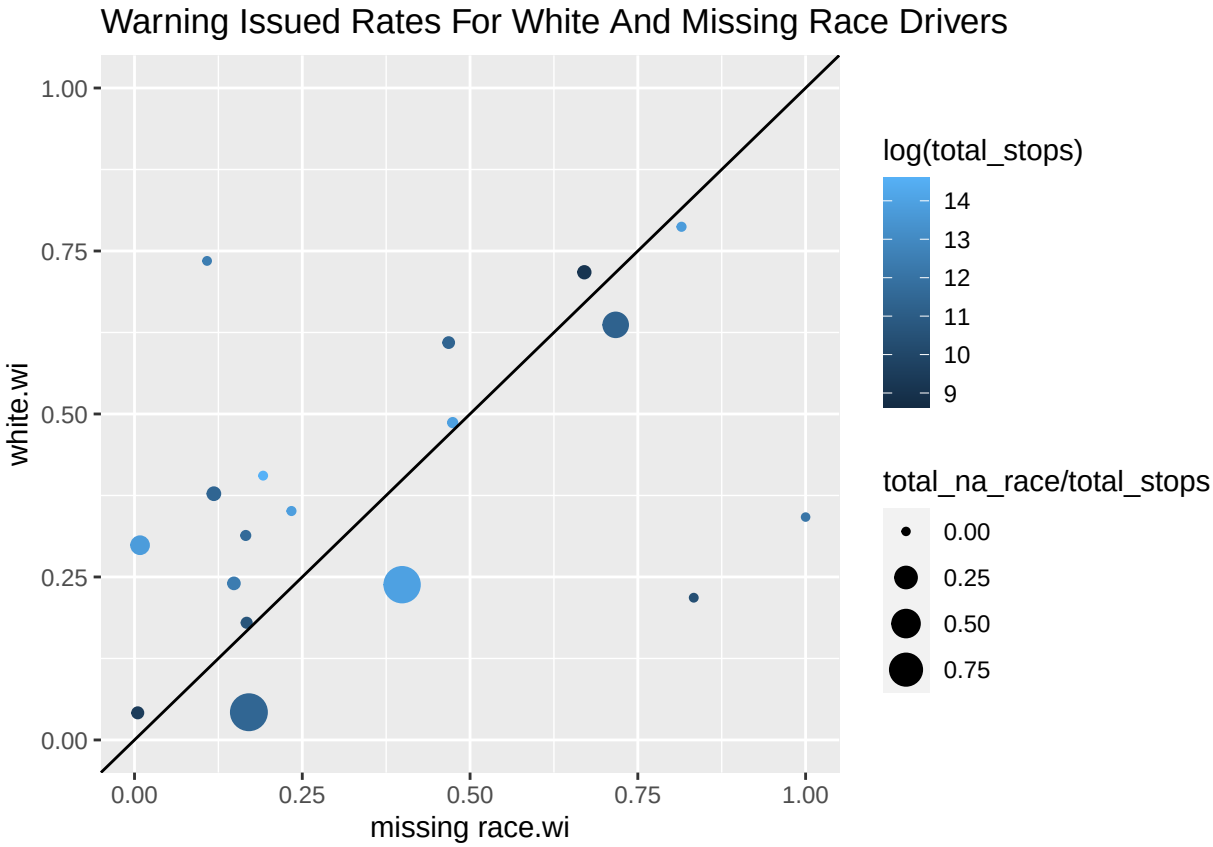

Search Conducted Rates For Black And Missing Race Drivers
black.sc
missing race.sc
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
log(total_stops)
14
13
12
11
10
9
total_na_race/total_stops
0.00
0.25
0.50
0.75

```
##
## [[4]]
```

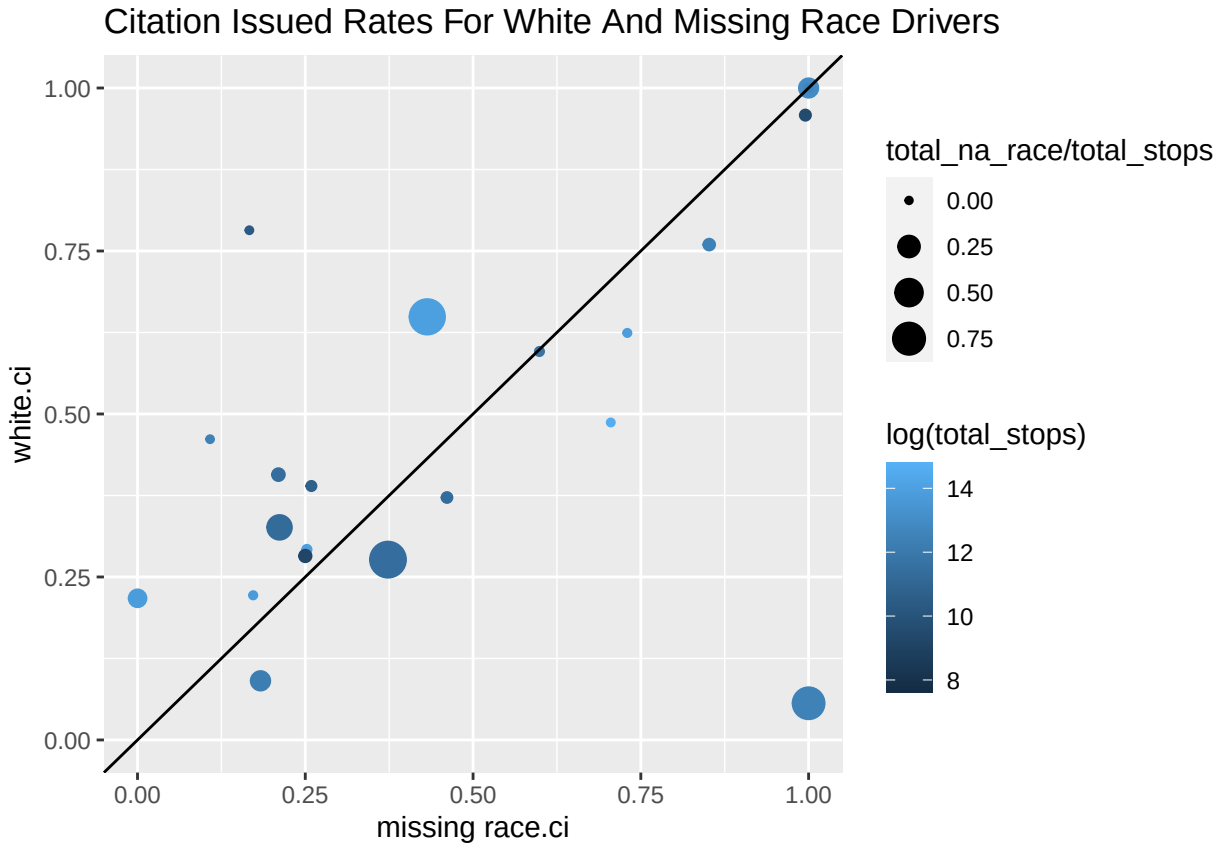



```
lapply(raceoutcomes_lst, ggplot_raceoutcome, "missing race", "white", FALSE, c(0, 1), c(0, 1), FALSE)
```

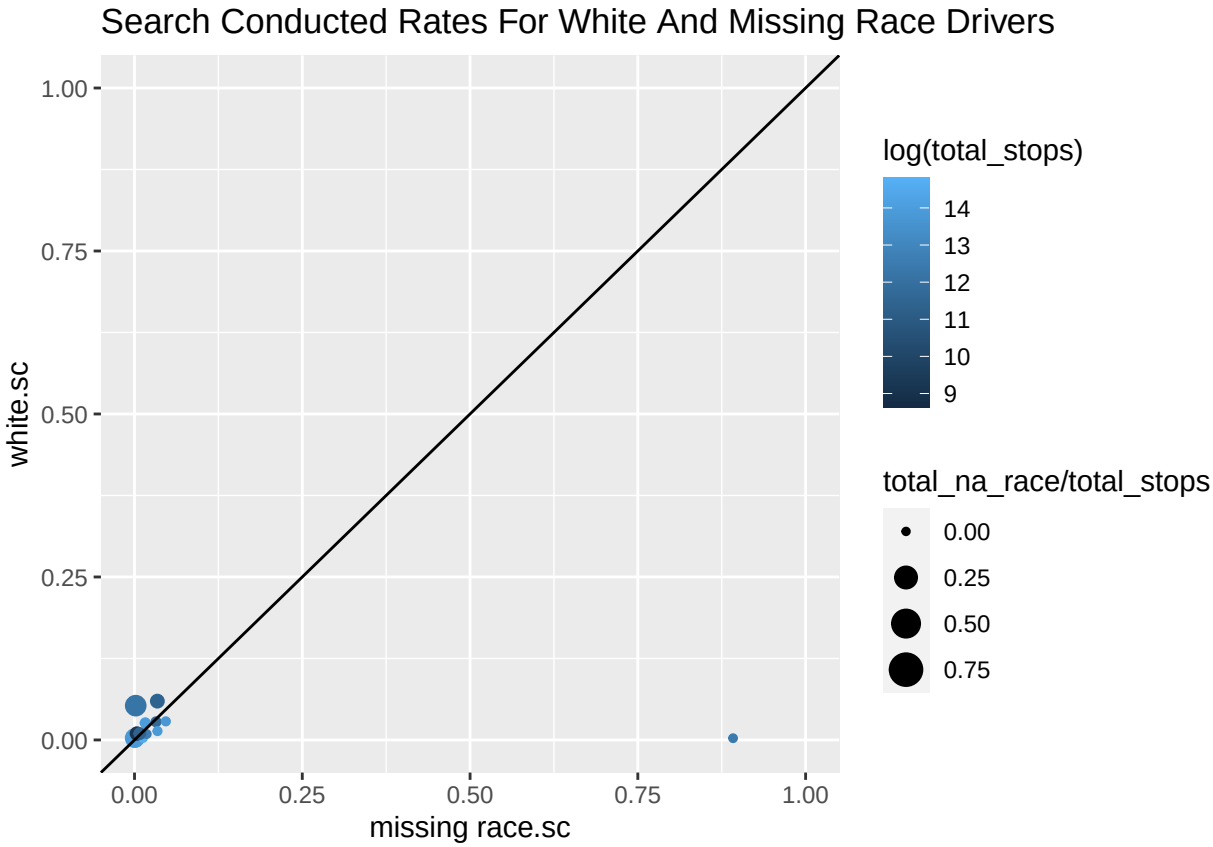
```
## [[1]]
```


Warning Issued Rates For White And Missing Race Drivers
white.wi
missing race.wi
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
log(total_stops)
14
13
12
11
10
9
total_na_race/total_stops
0.00
0.25
0.50
0.75

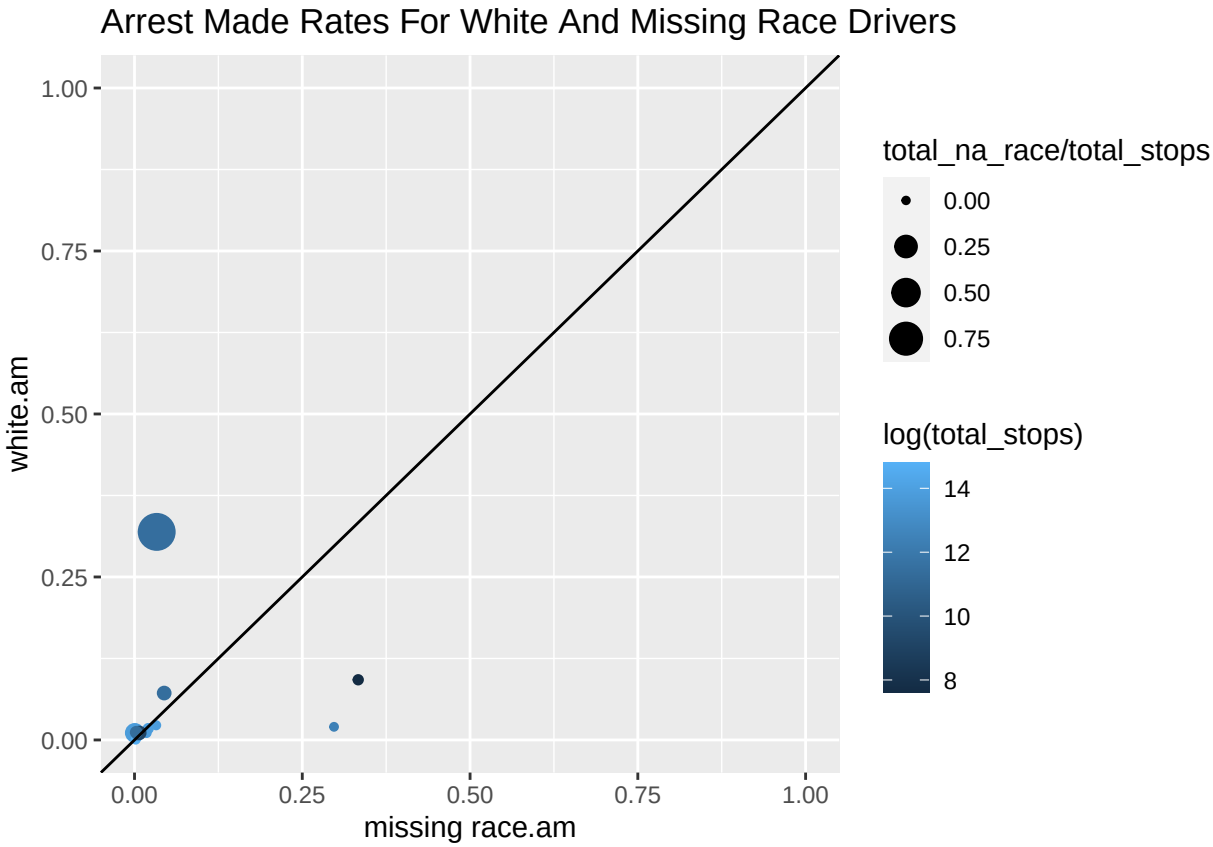
```
##
## [[2]]
```


Citation Issued Rates For White And Missing Race Drivers
white.ci
missing race.ci
0.00
0.25
0.50
0.75
1.00
total_na_race/total_stops
0.00
0.25
0.50
0.75
log(total_stops)
14
12
10
8

```
##
## [[3]]
```


Search Conducted Rates For White And Missing Race Drivers
white.sc
missing race.sc
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
log(total_stops)
14
13
12
11
10
9
total_na_race/total_stops
0.00
0.25
0.50
0.75

```
##
## [[4]]
```


Arrest Made Rates For White And Missing Race Drivers
white.am
missing race.am
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
total_na_race/total_stops
0.00
0.25
0.50
0.75
log(total_stops)
14
12
10
8