Assignment 3

Mounica Subramani

January 25, 2018

```
# Import required library files

library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------
---------------------------- tidyverse 1.2.1 --
```

```
## v tibble  1.4.1     v purrr   0.2.4
## v tidyr   0.7.2     v dplyr   0.7.4
## v readr   1.1.1     v stringr 1.2.0
## v tibble  1.4.1     v forcats 0.2.0
```

```
## -- Conflicts ------------------------------------------------------------------------
---------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidytext)
```

```
## Warning: package 'tidytext' was built under R version 3.4.4
```

```
library(dplyr)
library(rmarkdown)
library(nycflights13)
library(maps)
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:purrr':
##
##     map
```

```
library(measurements)
library(modelr)
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 3.4.4
```

```
library(purrr)
library(tokenizers)
```

```
## Warning: package 'tokenizers' was built under R version 3.4.4
```

```
library(pryr)
```

```
## Warning: package 'pryr' was built under R version 3.4.4
```

```
##
## Attaching package: 'pryr'
```

```
## The following objects are masked from 'package:purrr':
##
##     compose, partial
```

```
library(tinytex)
```

```
## Warning: package 'tinytex' was built under R version 3.4.4
```

```
library(devtools)
library(roxygen2)
```

```
##
## Attaching package: 'roxygen2'
```

```
## The following objects are masked from 'package:pryr':
##
##     is_s3_generic, is_s3_method
```

```
library(testthat)
```

```
##
## Attaching package: 'testthat'
```

```
## The following object is masked from 'package:devtools':
##
##     setup
```

```
## The following object is masked from 'package:dplyr':
##
##     matches
```

```
## The following object is masked from 'package:purrr':
##
##     is_null
```

```
options(width = 90)
```

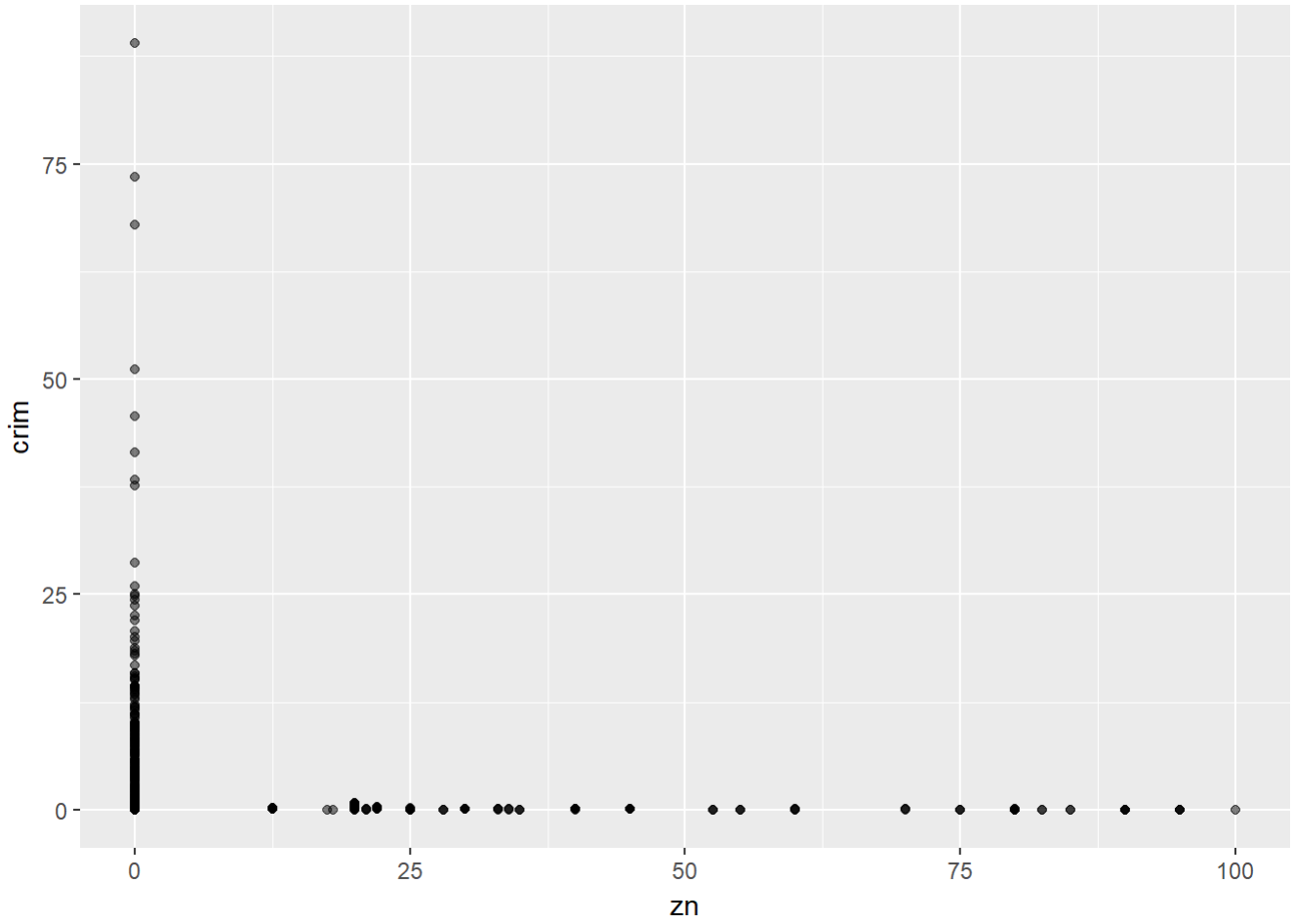
Part A

Problem 1

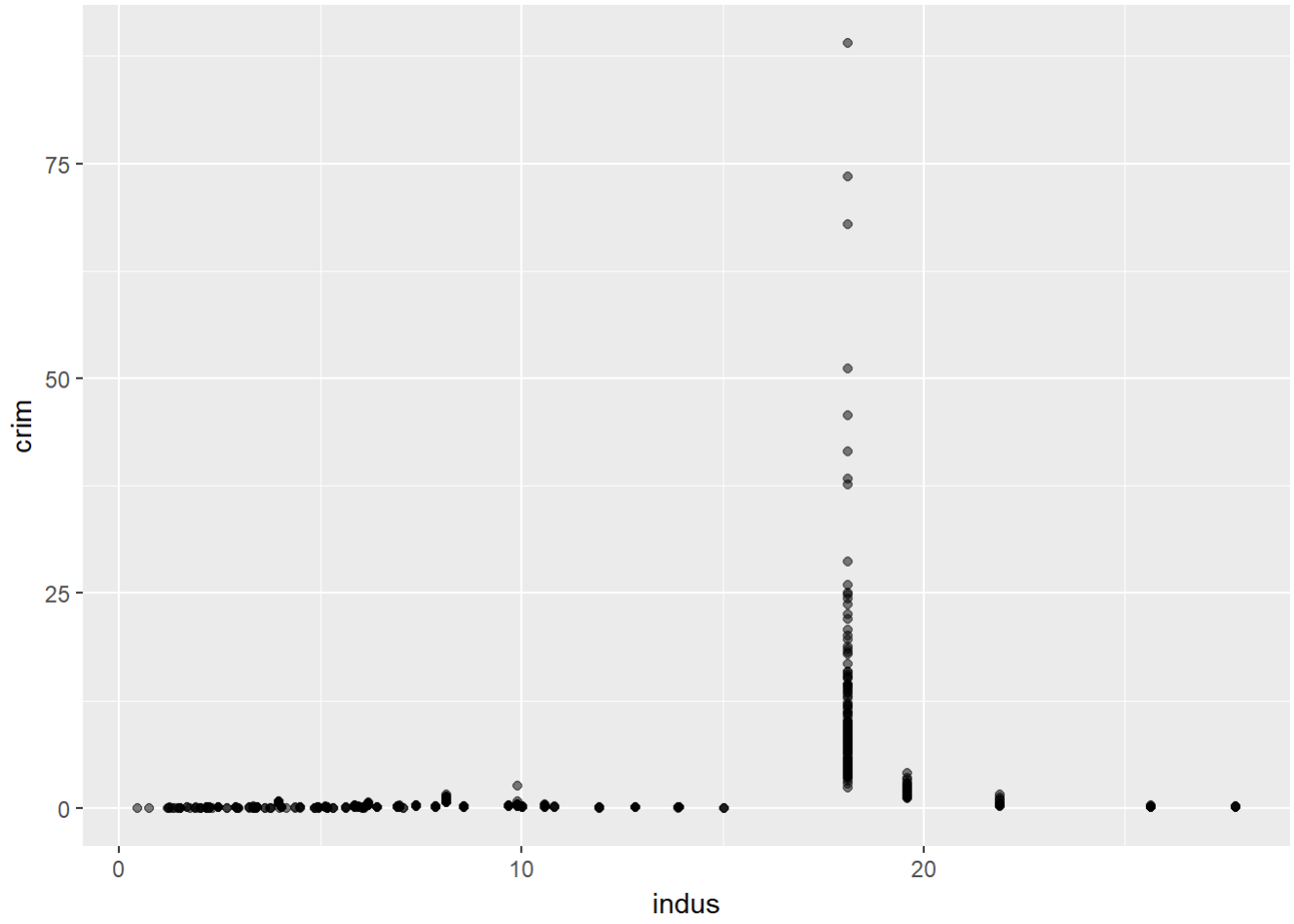
Fit a model that predicts per capita crime rate by town (crim) using only one predictor variable. Use plots to justify your choice of predictor variable and the appropriateness of any transformations you use. Print the values of the fitted model parameters.

```
data(BostonHousing)

set.seed(1)

# split_data <- resample_partition(BostonHousing,c(train = 0.9,
#                                  test = 0.1))
# split_data$train <- as_tibble(split_data$train)
```

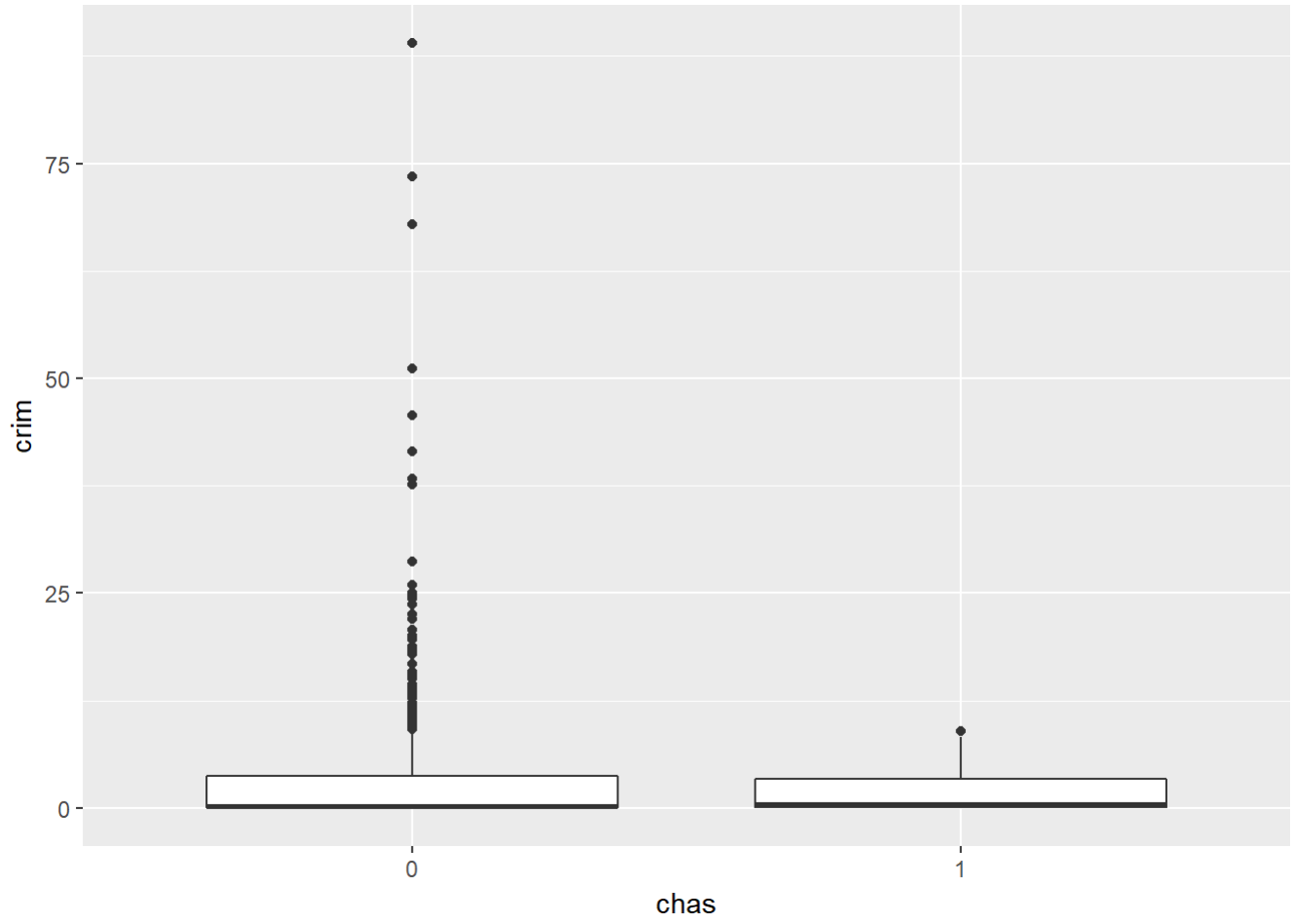
```
# per capita crime rate by town vs proportion of residential land zoned for lots over 25,000 sq.
ft
ggplot(BostonHousing, aes(zn, crim)) + geom_point(alpha = 0.5)
```




```
# per capita crime rate by town vs proportion of non-retail business acres per town
ggplot(BostonHousing, aes(indus, crim)) + geom_point(alpha = 0.5)
```

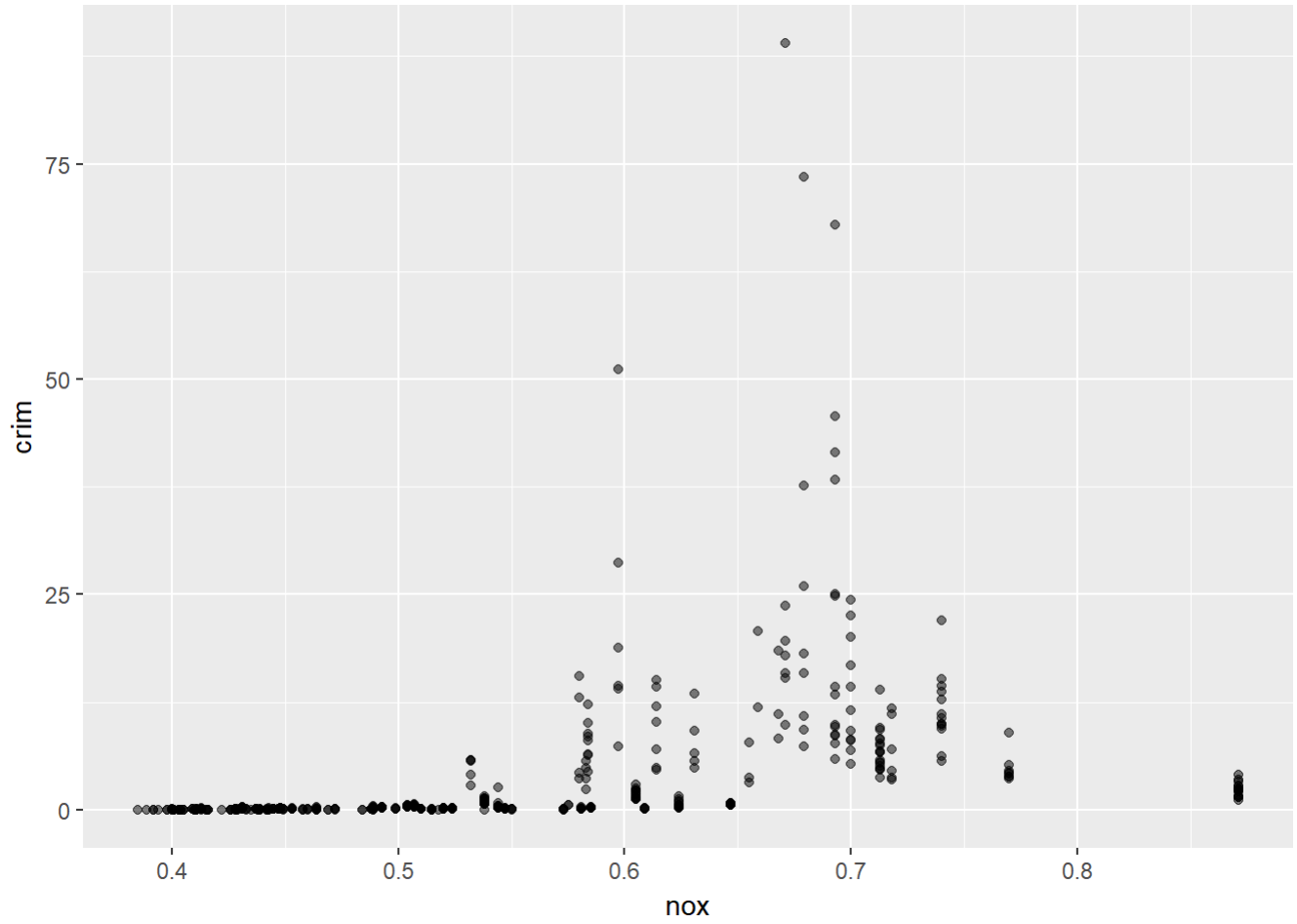



```
# per capita crime rate by town vs Charles River dummy variable (= 1 if tract bounds river; 0 ot
herwise)
ggplot(BostonHousing, aes(chas, crim)) + geom_boxplot()
```

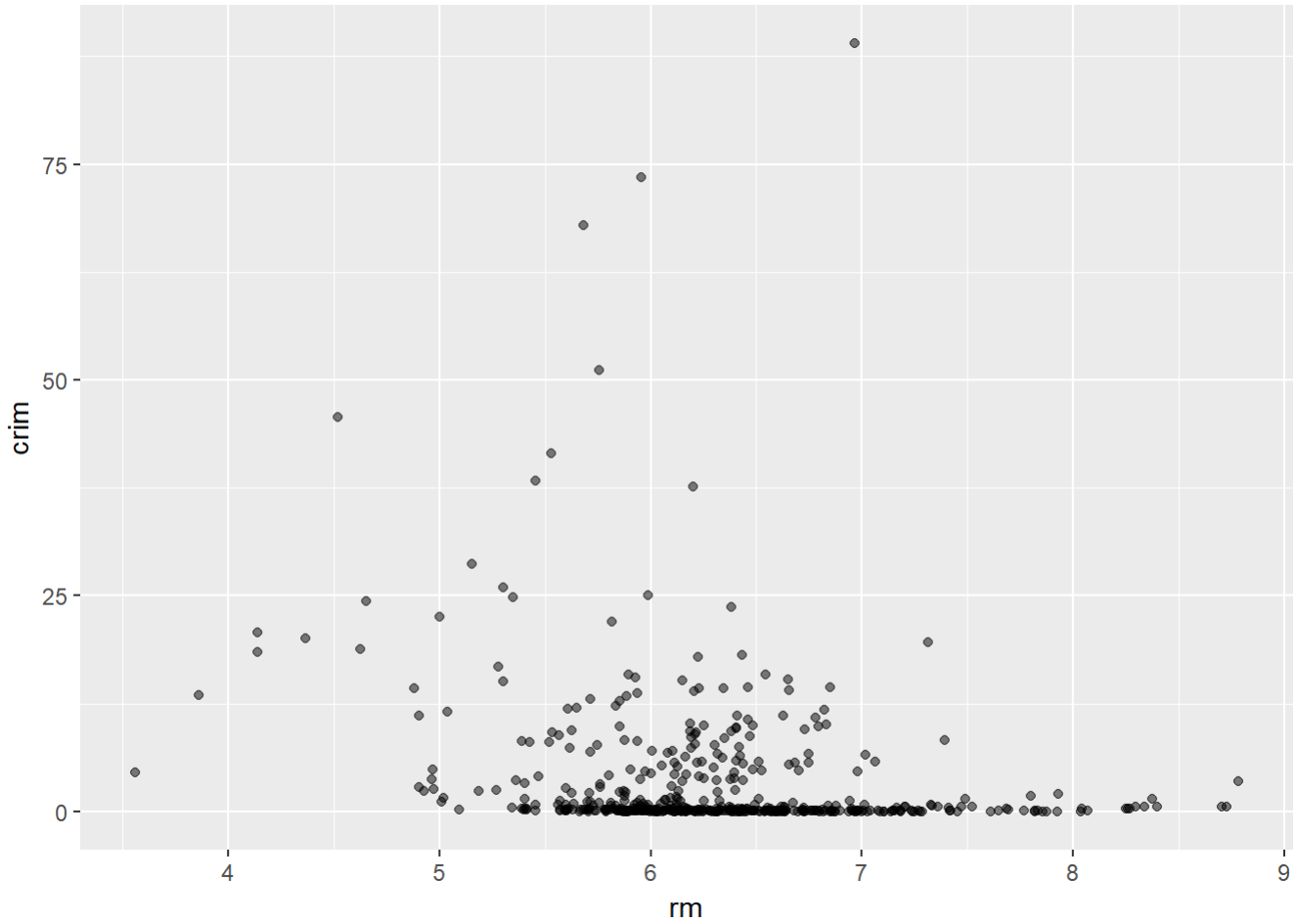



```
#chas seems to be a categorical variable and hence boxplot is used.
```

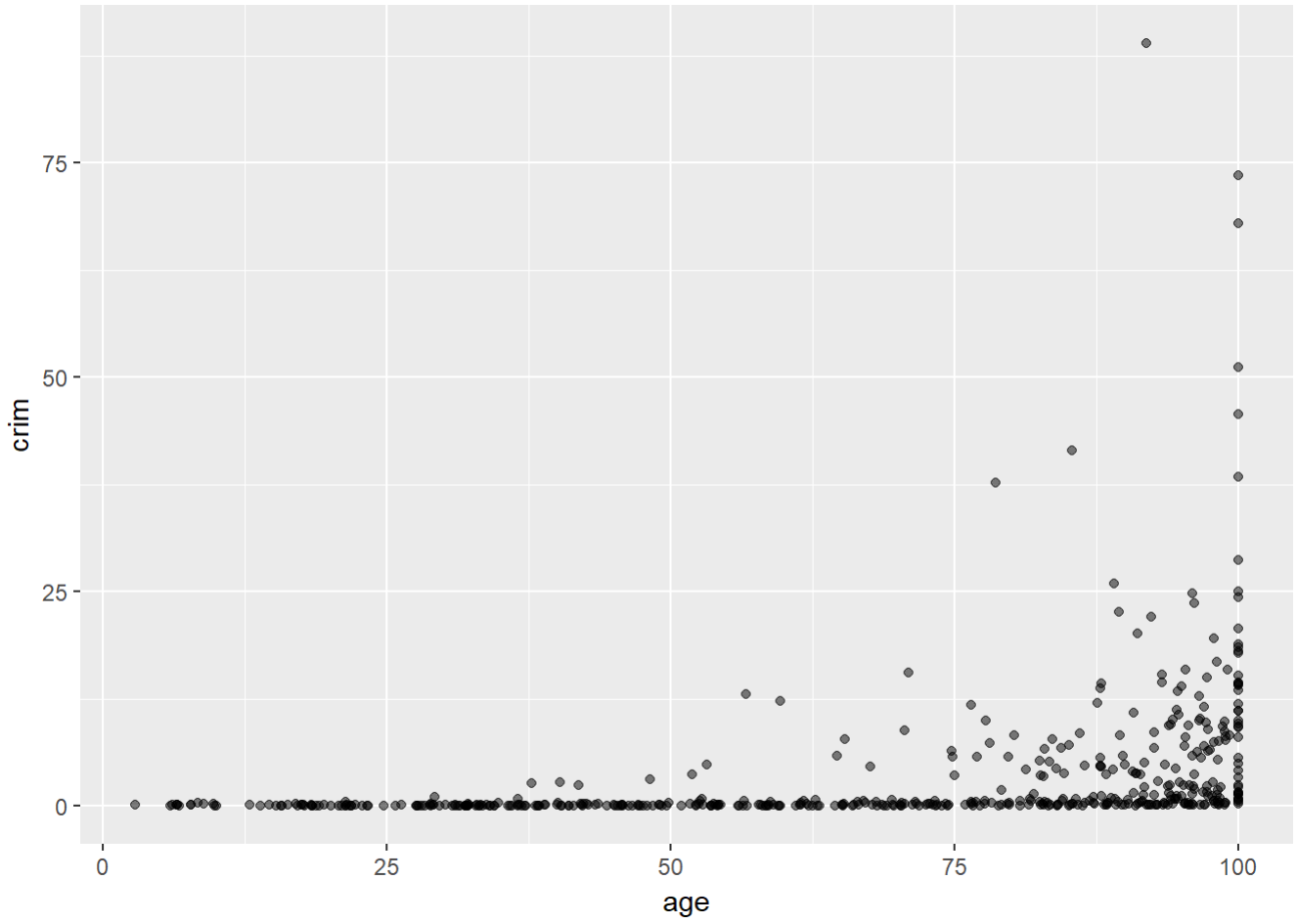
```
# per capita crime rate by town vs nitric oxides concentration (parts per 10 million)
ggplot(BostonHousing, aes(nox, crim)) + geom_point(alpha = 0.5)
```



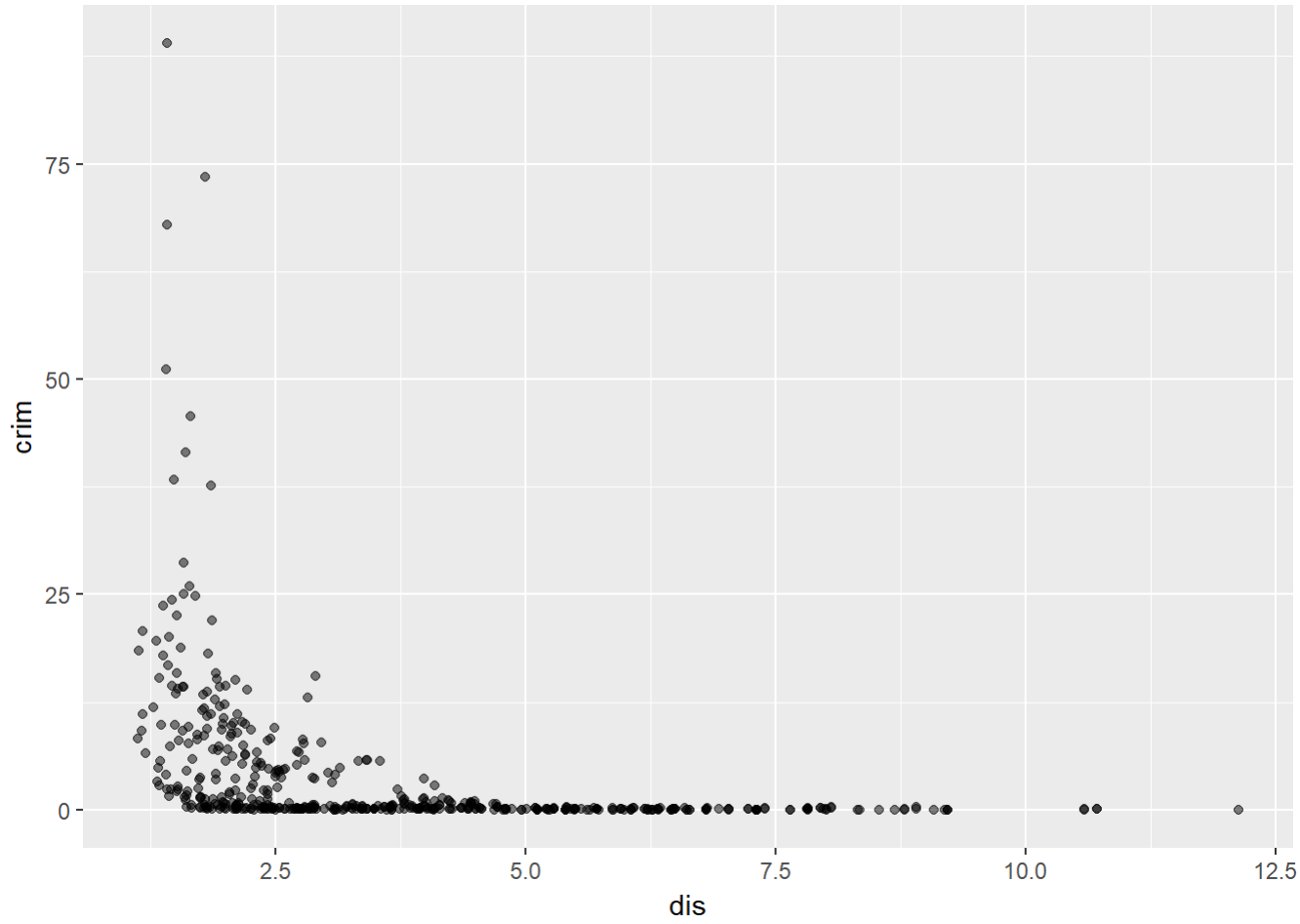

```
# per capita crime rate by town vs average number of rooms per dwelling
ggplot(BostonHousing, aes(rm, crim)) + geom_point(alpha = 0.5)
```



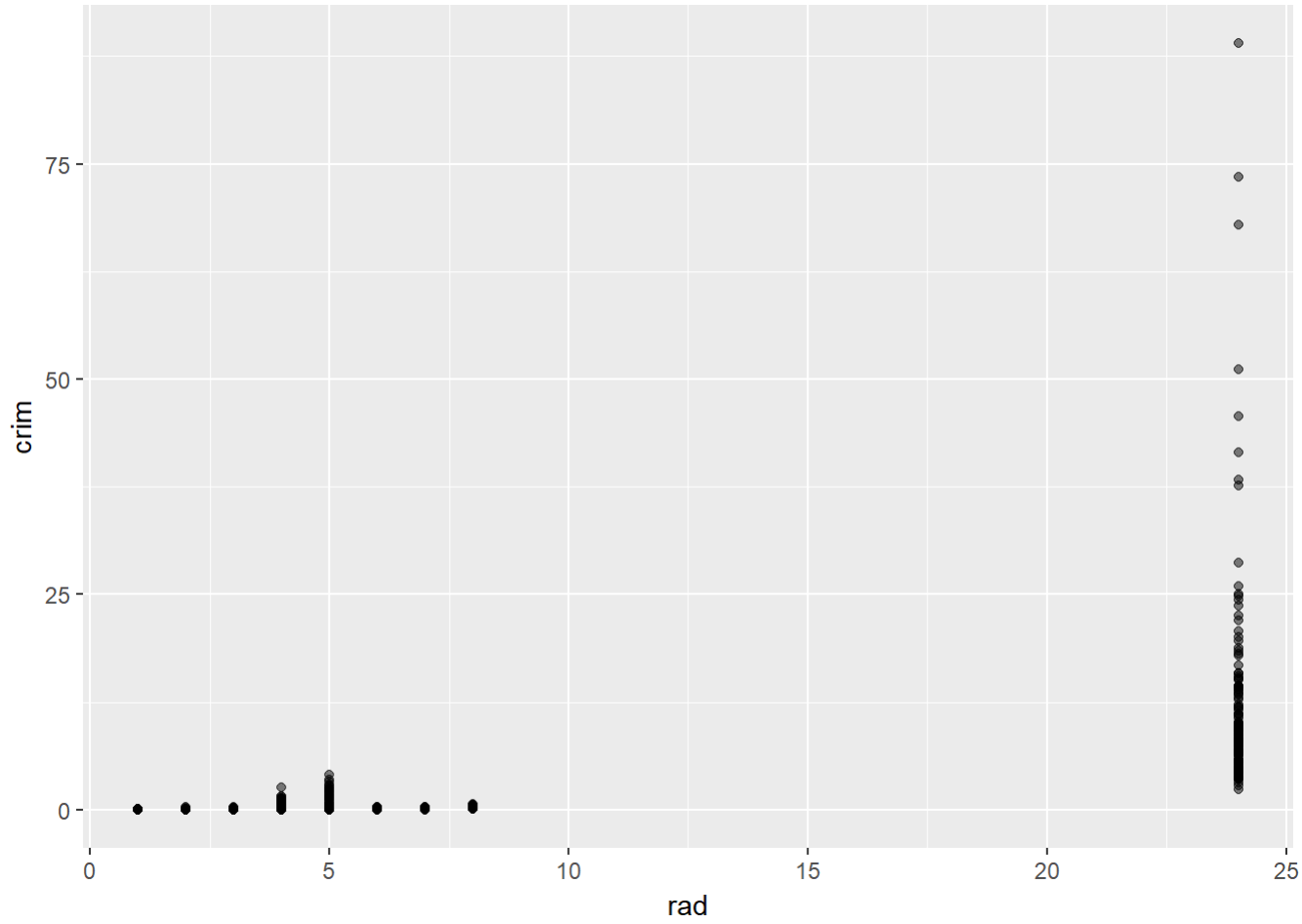

```
# per capita crime rate by town vs proportion of owner-occupied units built prior to 1940
ggplot(BostonHousing, aes(age, crim)) + geom_point(alpha = 0.5)
```



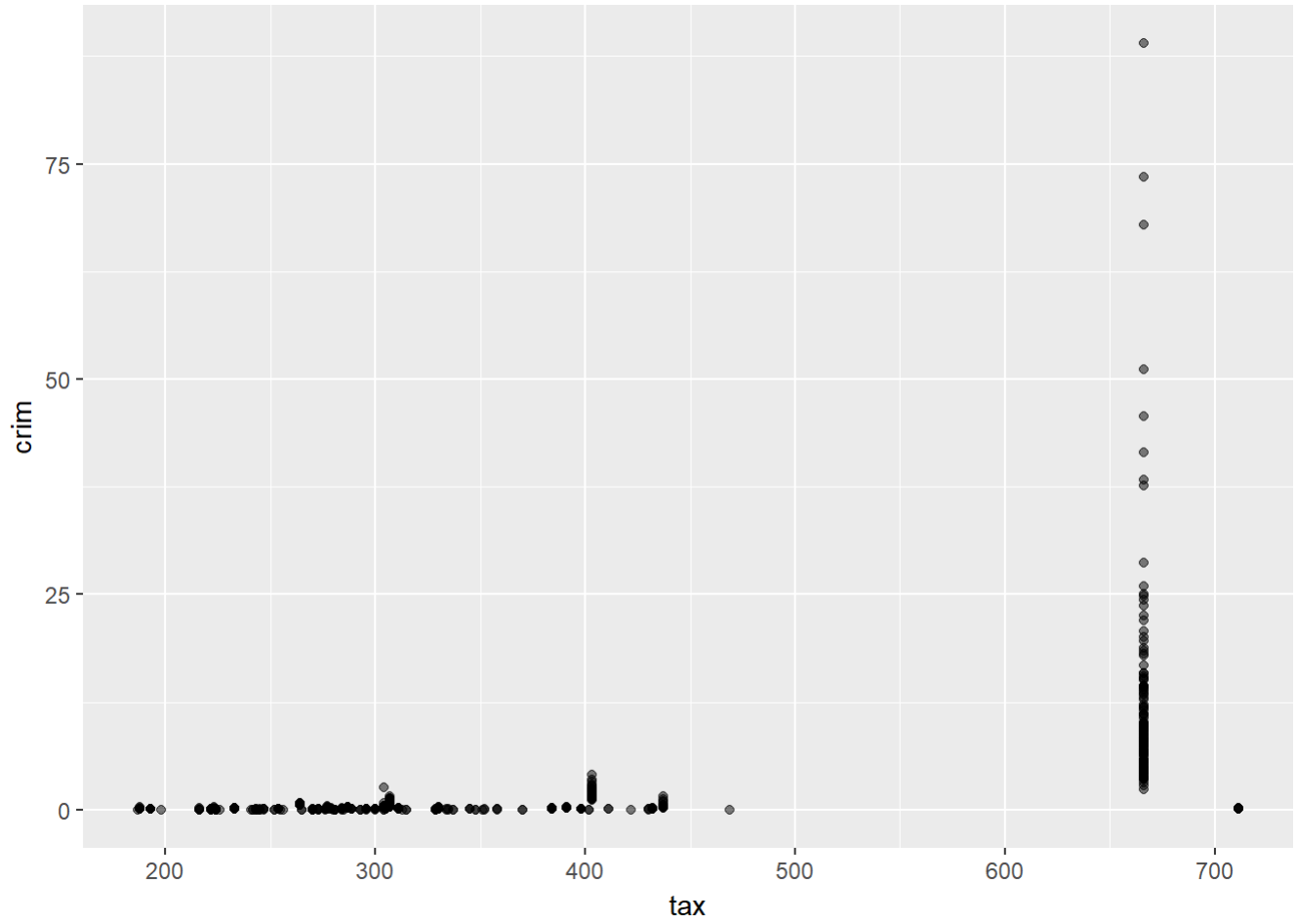

```
# per capita crime rate by town vs weighted distances to five Boston employment centres
ggplot(BostonHousing, aes(dis, crim)) + geom_point(alpha = 0.5)
```



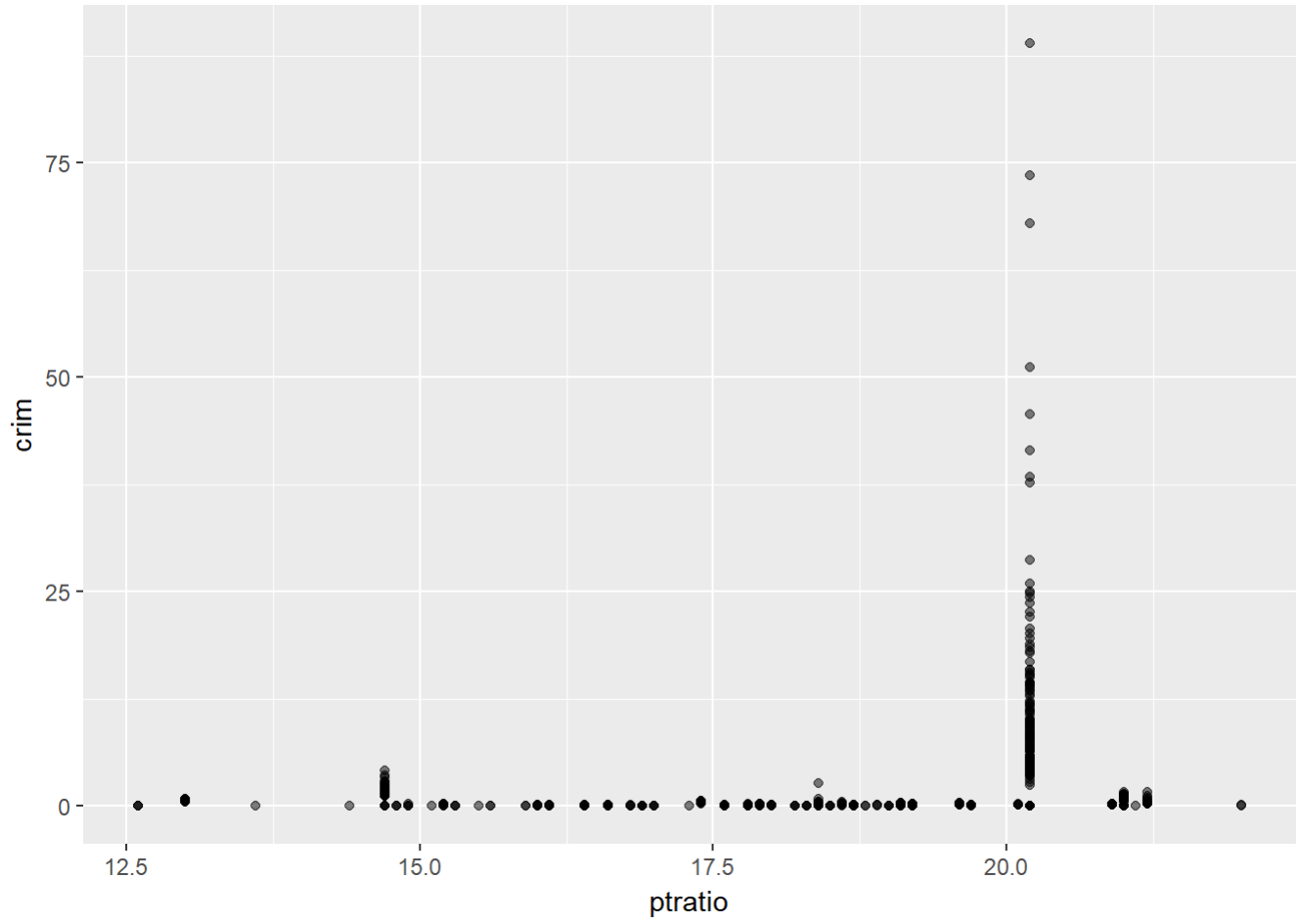

```
# per capita crime rate by town vs index of accessibility to radial highways
ggplot(BostonHousing, aes(rad, crim)) + geom_point(alpha = 0.5)
```



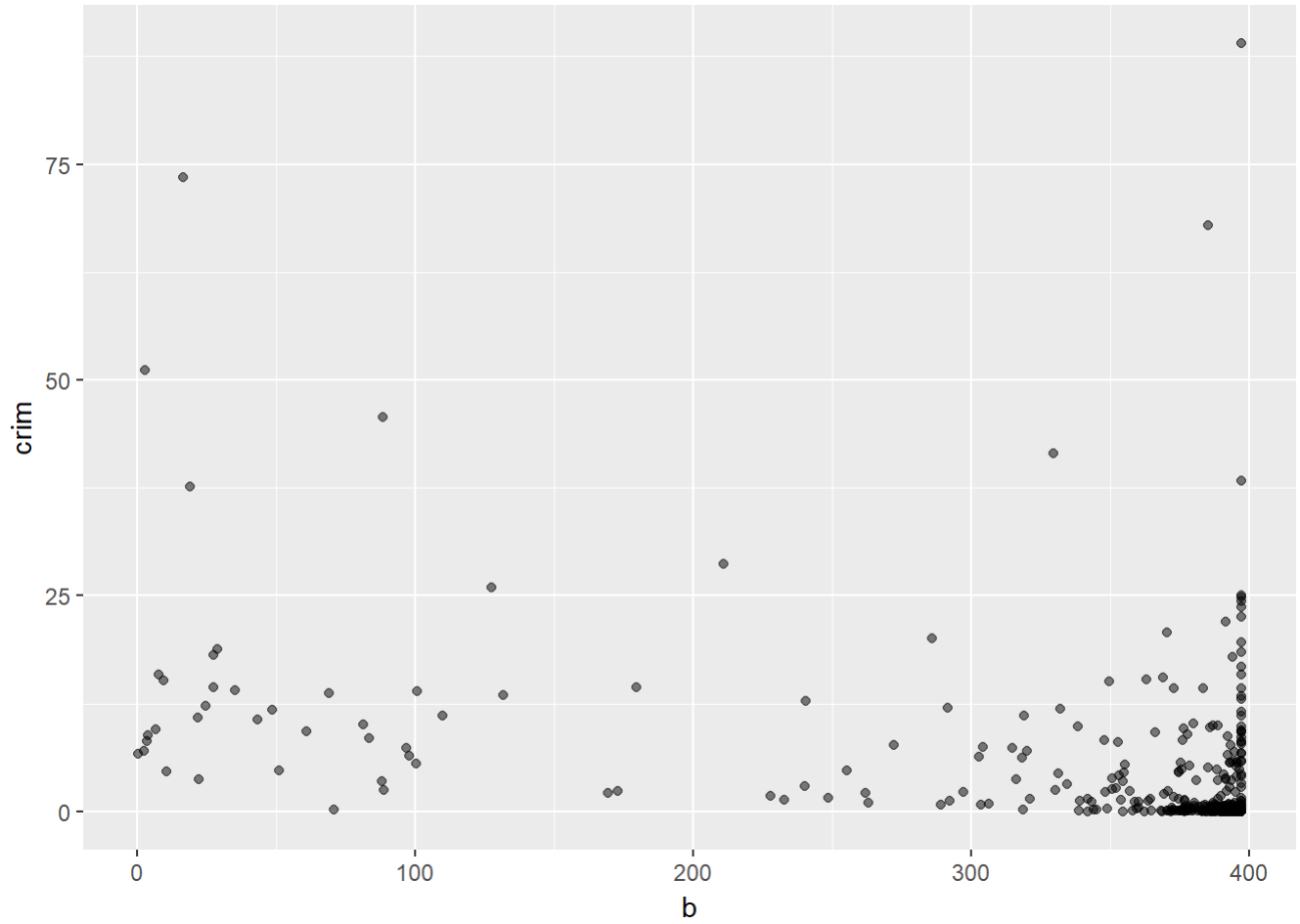

```
# per capita crime rate by town vs full-value property-tax rate per USD 10,000
ggplot(BostonHousing, aes(tax, crim)) + geom_point(alpha = 0.5)
```



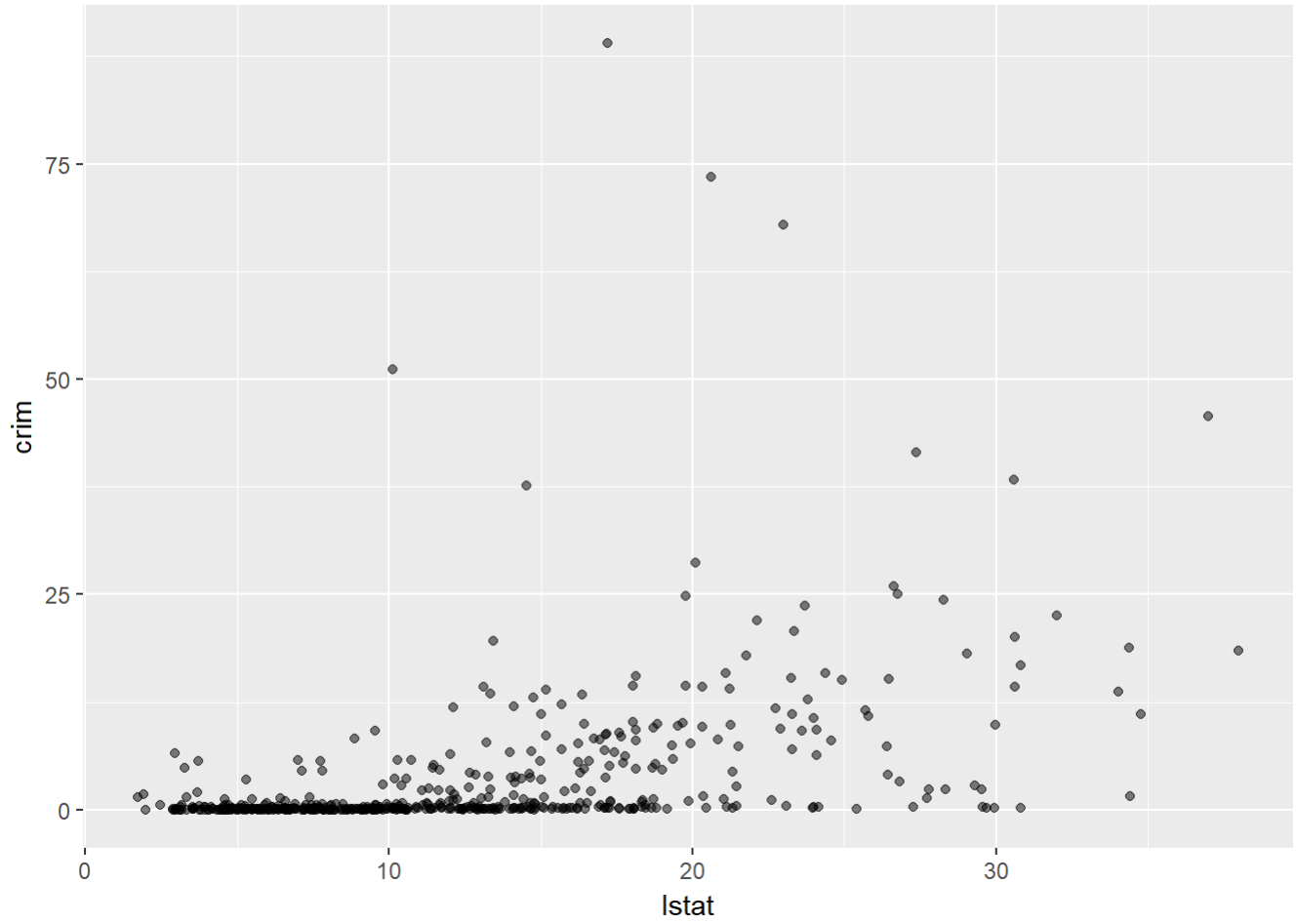

```
# per capita crime rate by town vs pupil-teacher ratio by town
ggplot(BostonHousing, aes(ptratio, crim)) + geom_point(alpha = 0.5)
```



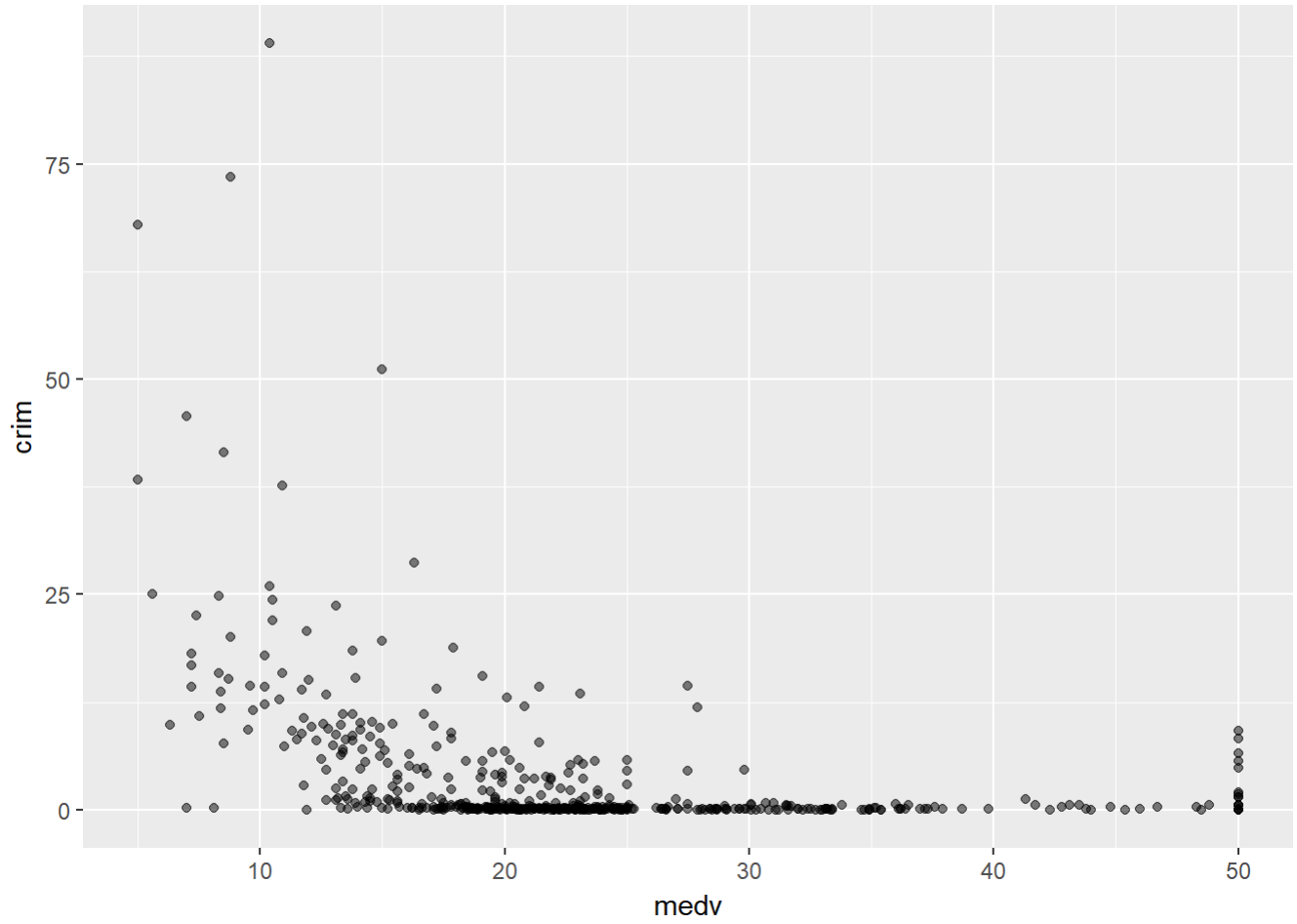

```
# per capita crime rate by town vs 1000(B - 0.63)^2 where B is the proportion of blacks by town
ggplot(BostonHousing, aes(b, crim)) + geom_point(alpha = 0.5)
```




```
# per capita crime rate by town vs percentage of lower status of the population
ggplot(BostonHousing, aes(lstat, crim)) + geom_point(alpha = 0.5)
```

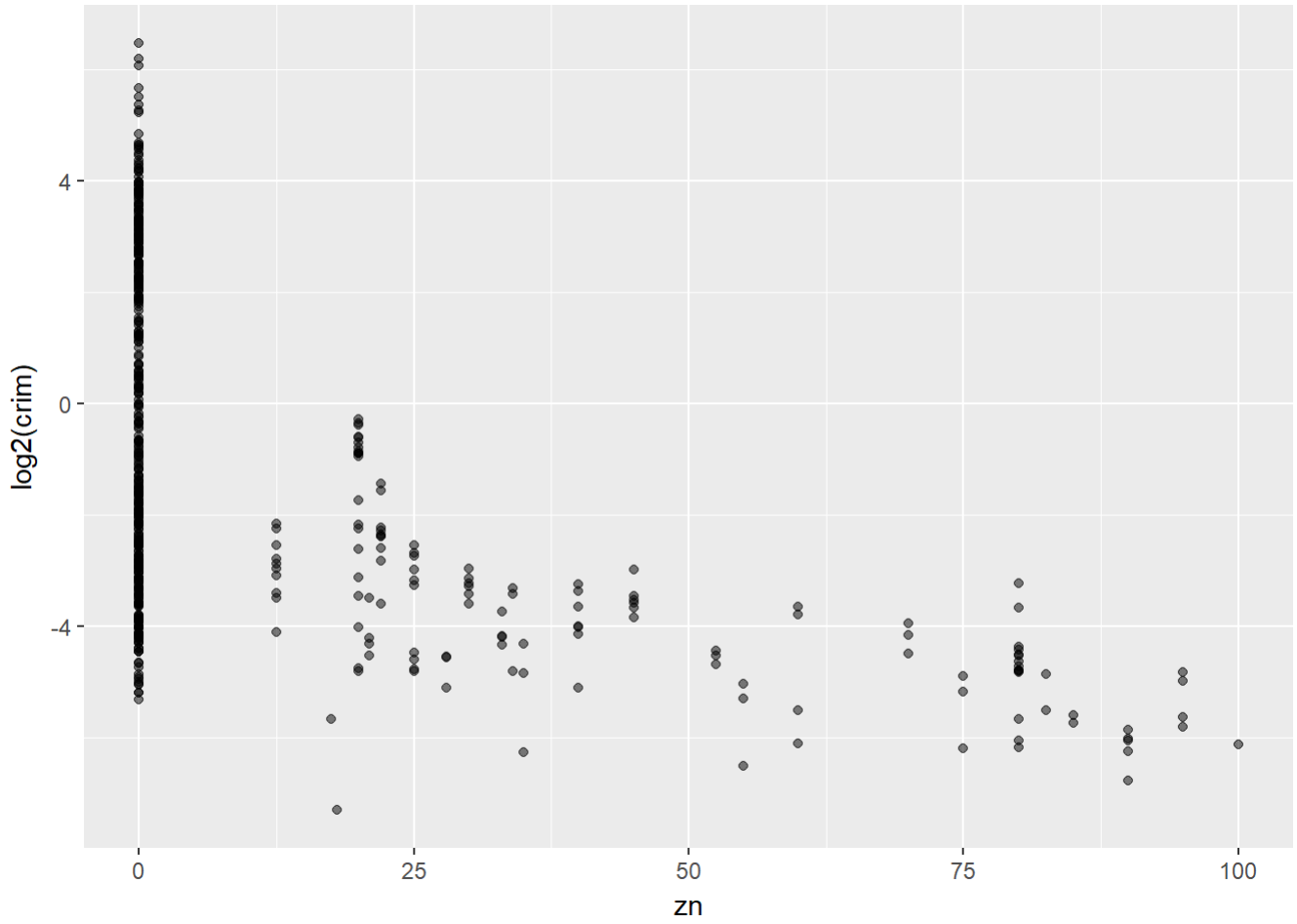



```
# per capita crime rate by town vs median value of owner-occupied homes in USD 1000's
ggplot(BostonHousing, aes(medv, crim)) + geom_point(alpha = 0.5)
```

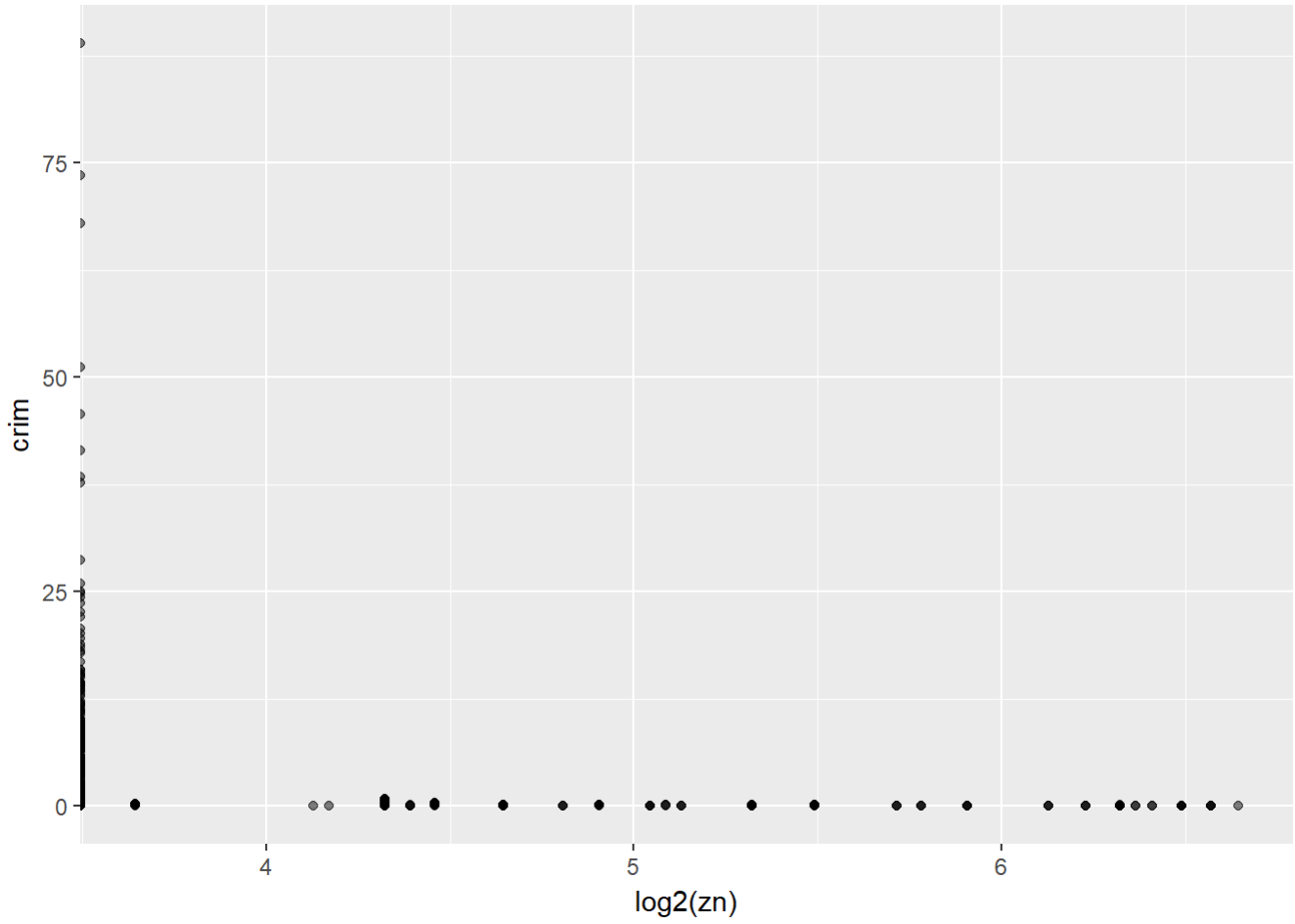


We see no obvious pattern and also the points are closely knitted which is making prediction a complex one. So we are trying to scatter the points in the plot.

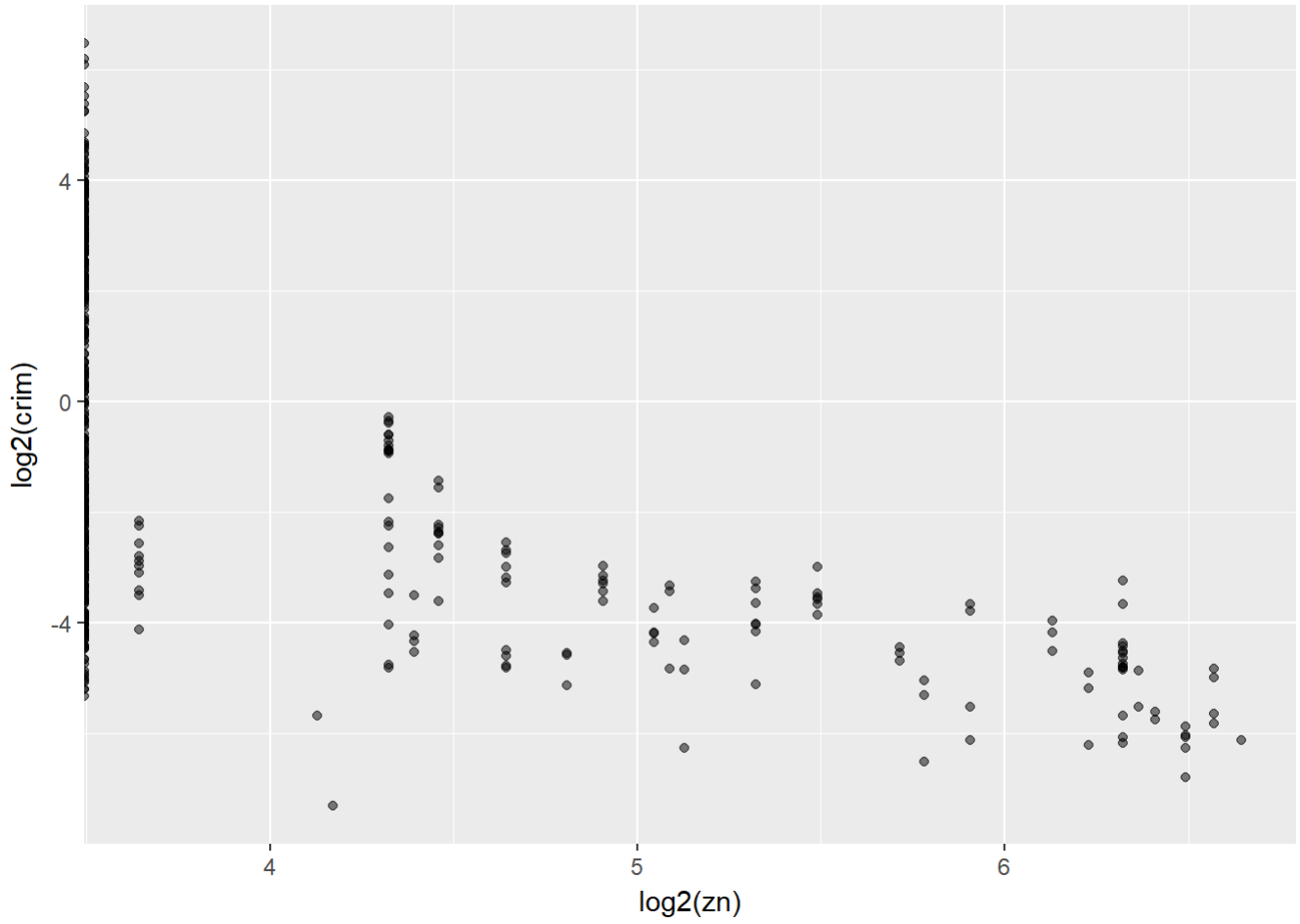
```
# trying to scatter the points on the plot using log2/log for better visibility.
# per capita crime rate by town vs proportion of residential land zoned for lots over 25,000 sq. ft
ggplot(BostonHousing, aes(zn, log2(crim))) + geom_point(alpha = 0.5)
```




```
#trying log2/log with predictor variable to check if any significant differences or patterns can
 be observed.
# per capita crime rate by town vs proportion of residential land zoned for lots over 25,000 sq.
ft
ggplot(BostonHousing, aes(log2(zn), crim)) + geom_point(alpha = 0.5)
```

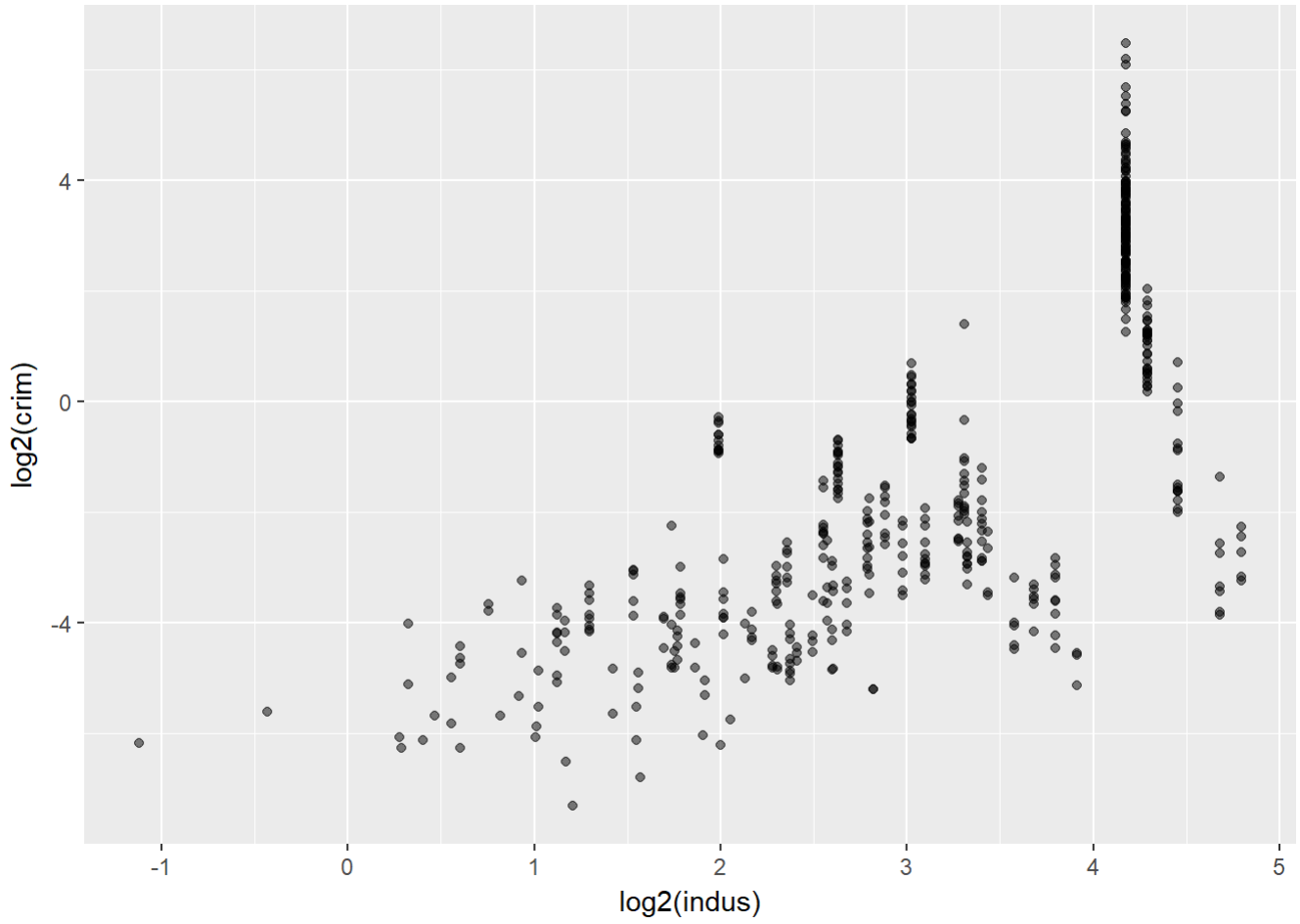



```
# using log2/log for both the variables to check if any correlation can be observed.
# per capita crime rate by town vs proportion of residential land zoned for lots over 25,000 sq. ft
ggplot(BostonHousing, aes(log2(zn), log2(crim))) + geom_point(alpha = 0.5)
```



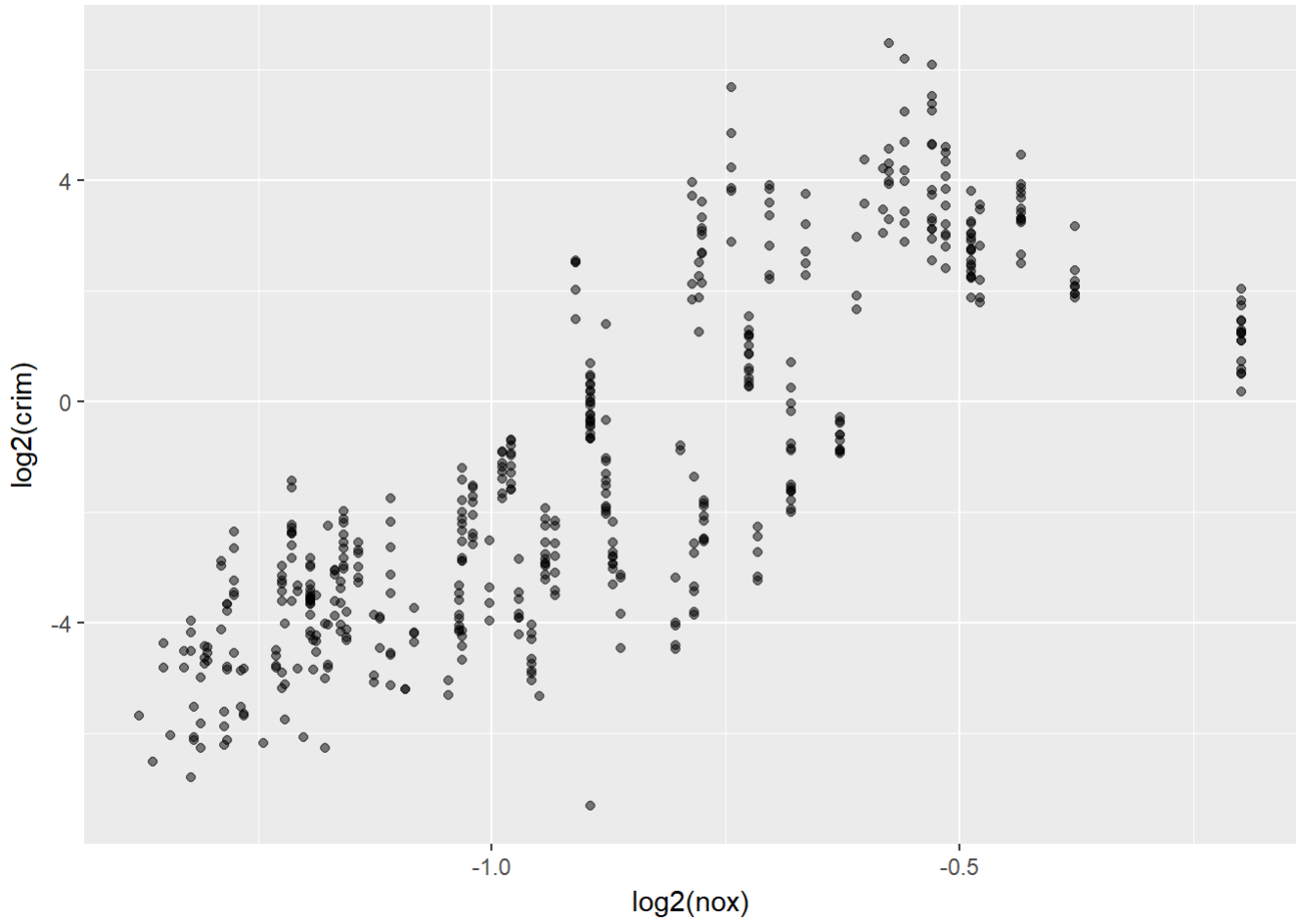

```
#We see no obvious pattern, so we won't use `zn` as a predictor variable.
```

```
# per capita crime rate by town vs proportion of non-retail business acres per town
ggplot(BostonHousing, aes(log2(indus), log2(crim))) + geom_point(alpha = 0.5)
```



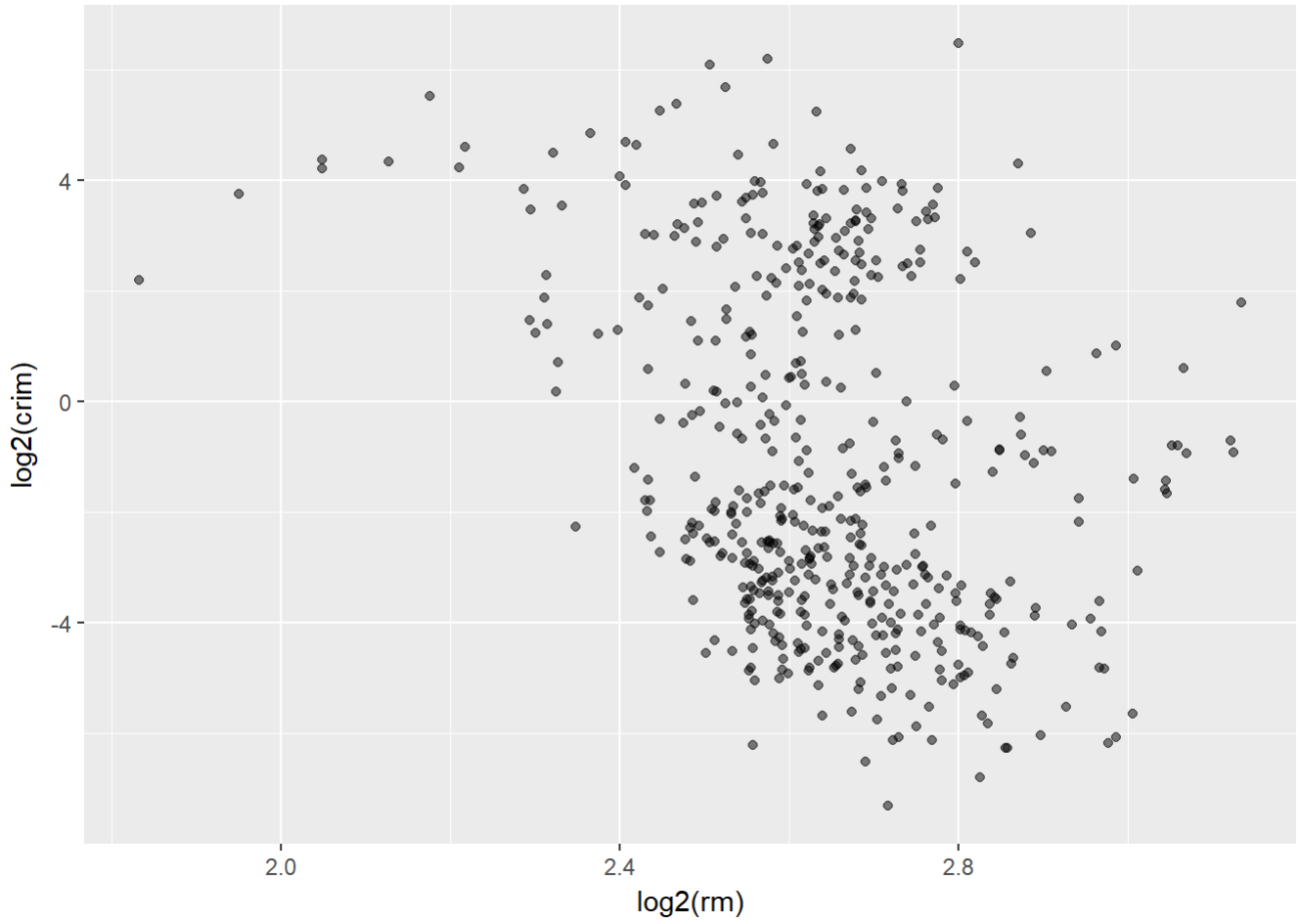

```
#We see no obvious pattern, so we won't use `indus` as a predictor variable.
```

```
# per capita crime rate by town vs nitric oxides concentration (parts per 10 million)
ggplot(BostonHousing, aes(log2(nox), log2(crim))) + geom_point(alpha = 0.5)
```



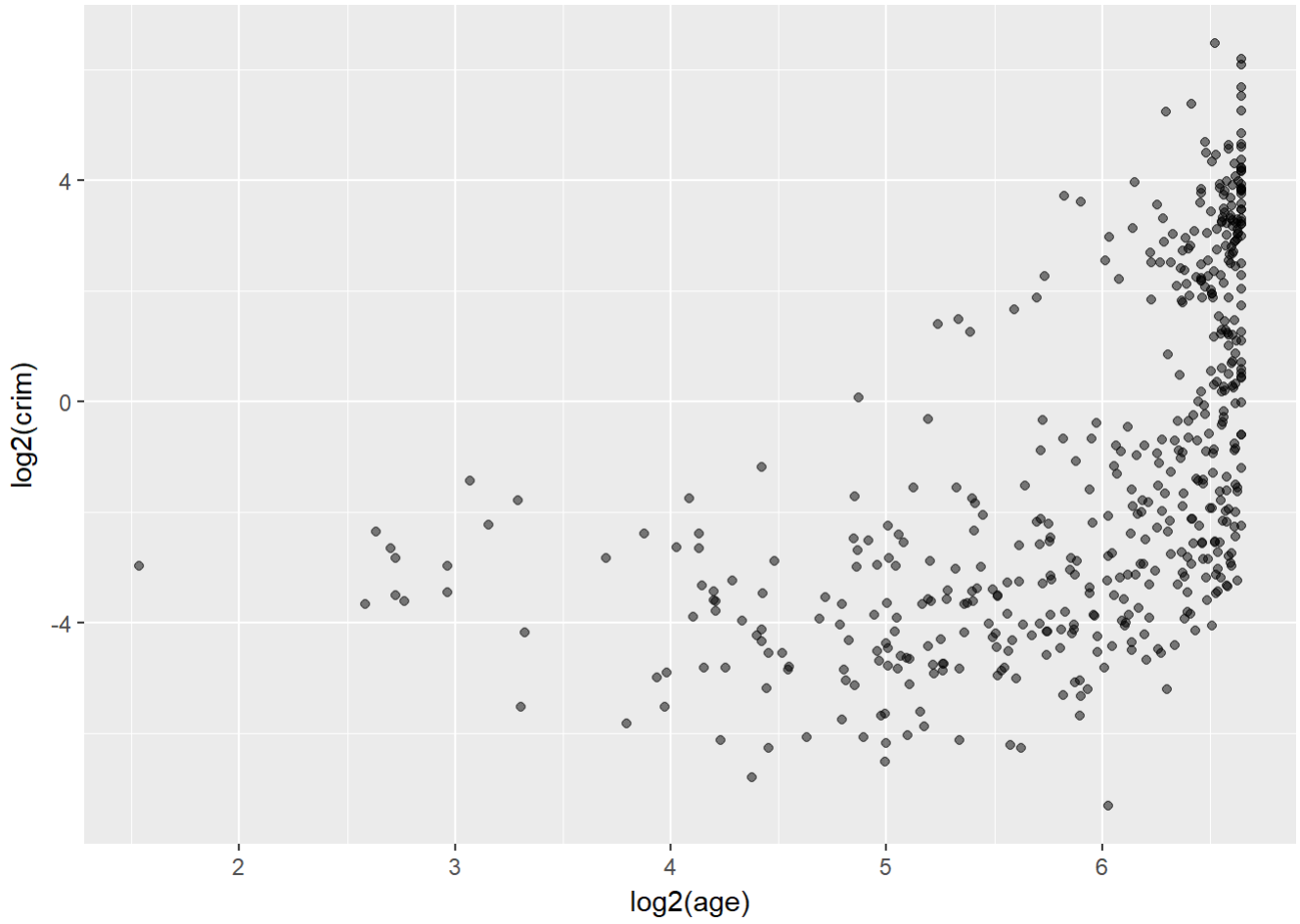

```
#We see a slight positive linear relation, but lets check with other variables too.
```

```
# per capita crime rate by town vs average number of rooms per dwelling
ggplot(BostonHousing, aes(log2(rm), log2(crim))) + geom_point(alpha = 0.5)
```



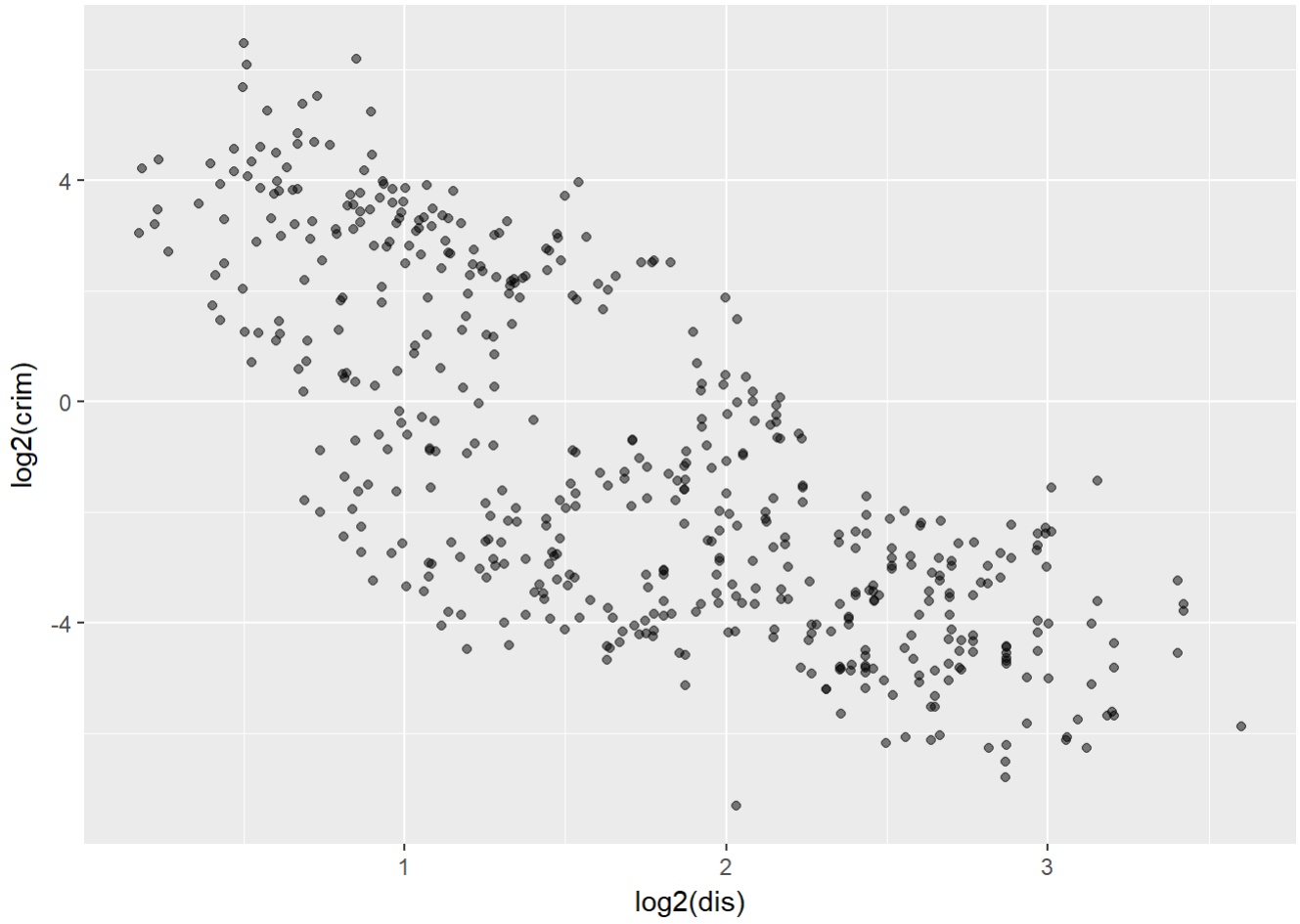

```
#We see no obvious pattern and its pretty random, so we won't use `rm` as a predictor variable.
```

```
# per capita crime rate by town vs proportion of owner-occupied units built prior to 1940
ggplot(BostonHousing, aes(log2(age), log2(crim))) + geom_point(alpha = 0.5)
```



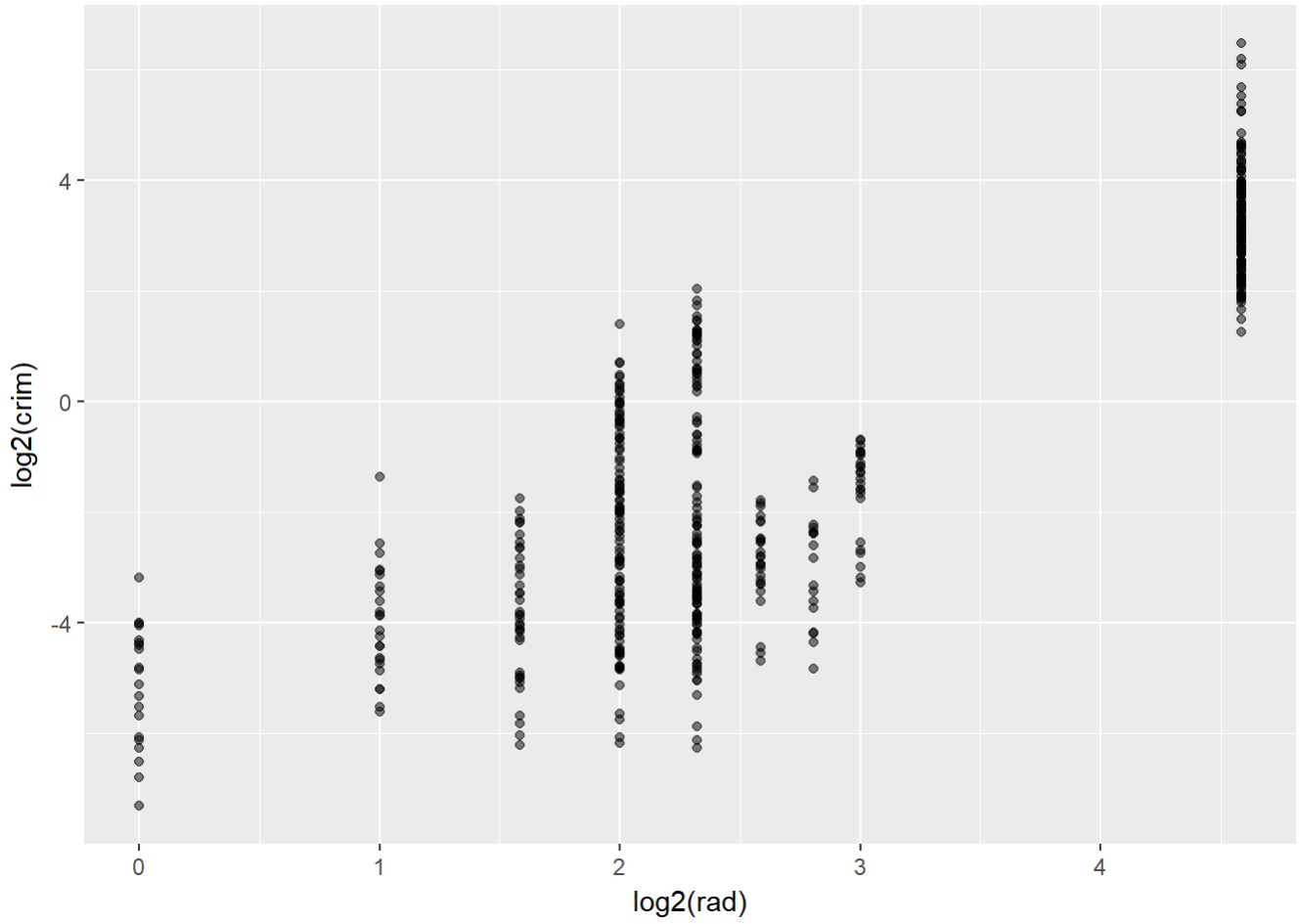

```
#We see no obvious pattern, so we won't use `age` as a predictor variable.
```

```
# per capita crime rate by town vs weighted distances to five Boston employment centres
ggplot(BostonHousing, aes(log2(dis), log2(crim))) + geom_point(alpha = 0.5)
```

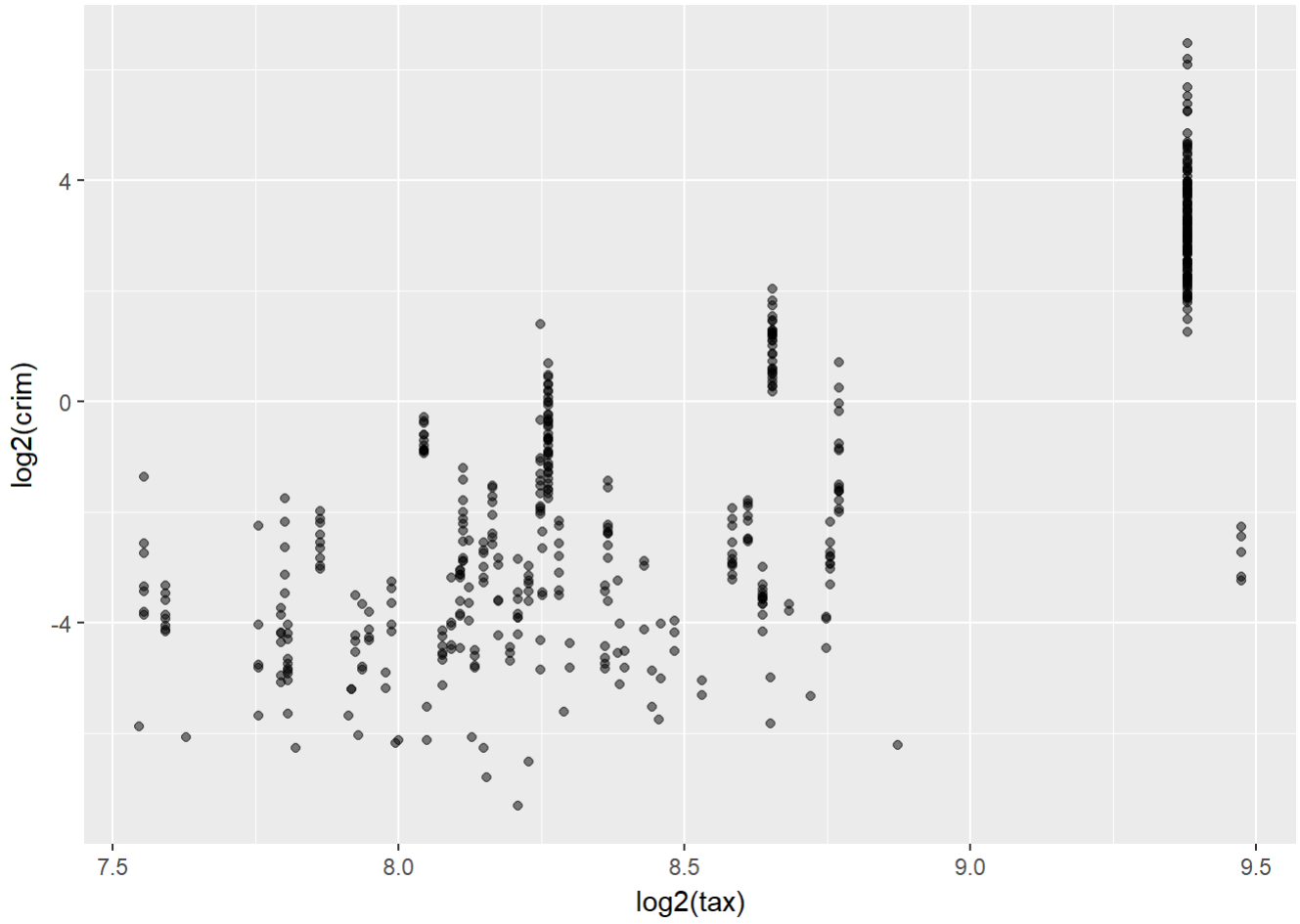


We see a strong negatively linear relationship between the variables, lets consider the variable.

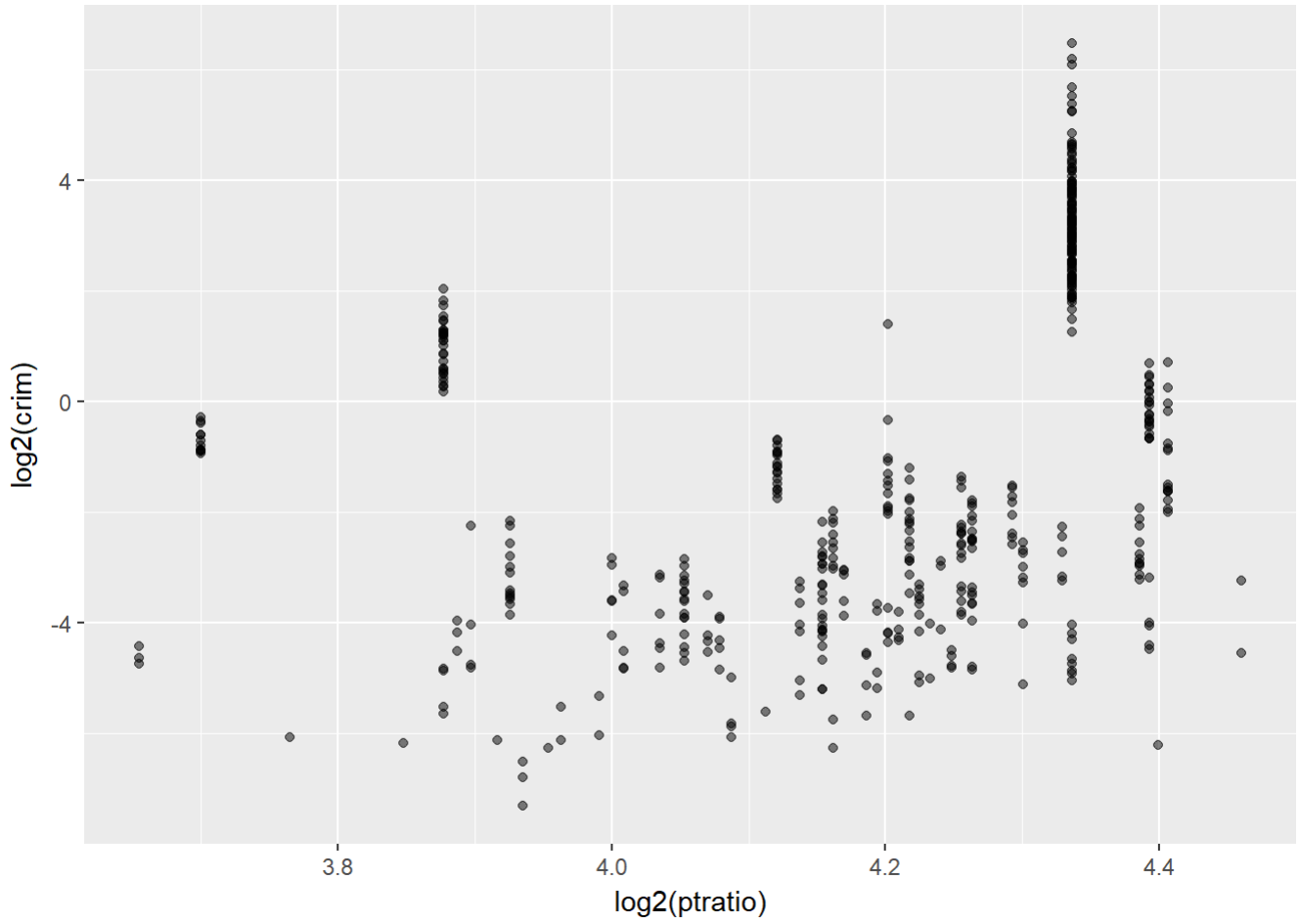
```
# per capita crime rate by town vs index of accessibility to radial highways
ggplot(BostonHousing, aes(log2(rad), log2(crim))) + geom_point(alpha = 0.5)
```



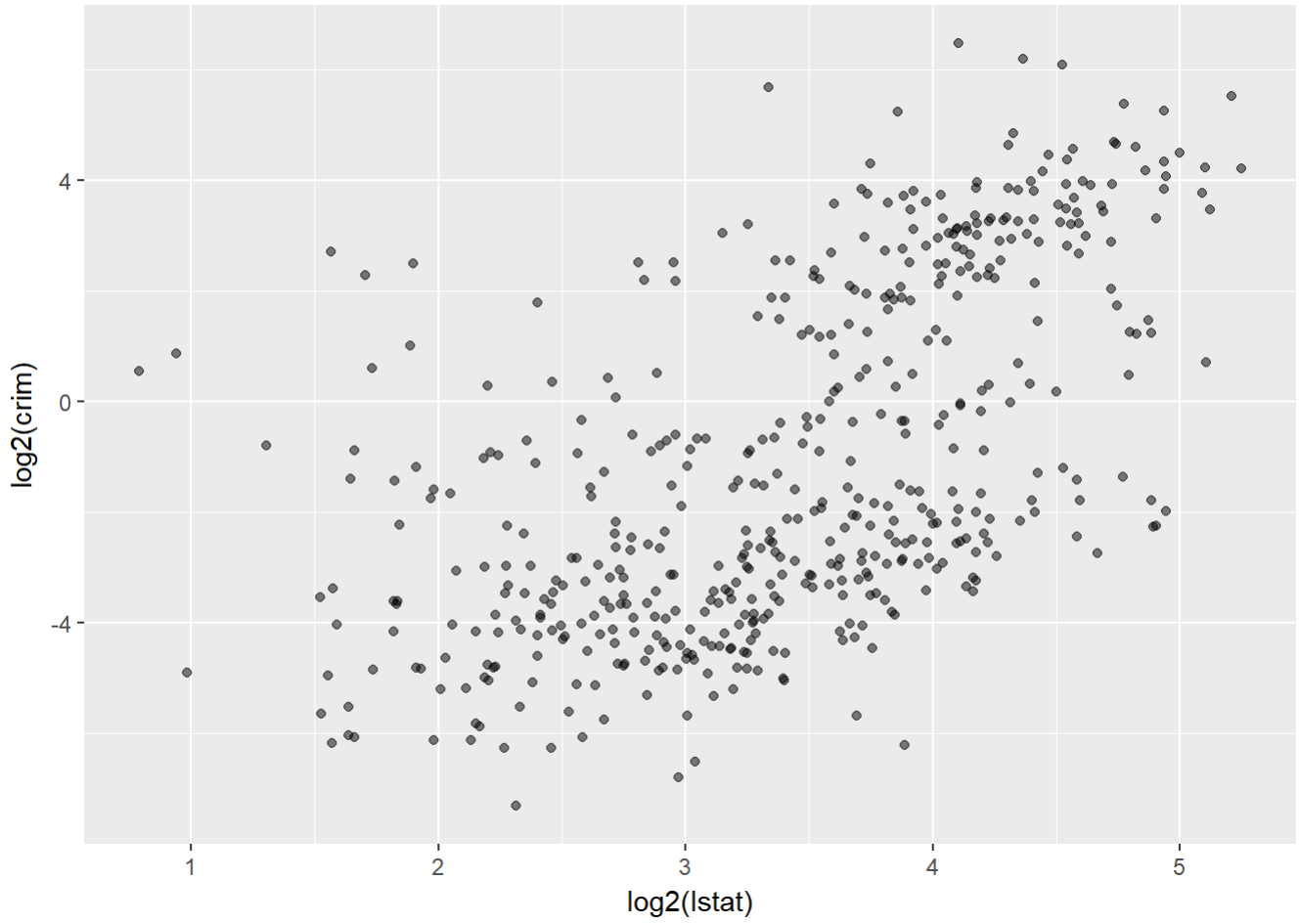

```
# per capita crime rate by town vs full-value property-tax rate per USD 10,000
ggplot(BostonHousing, aes(log2(tax), log2(crim))) + geom_point(alpha = 0.5)
```




```
# per capita crime rate by town vs pupil-teacher ratio by town
ggplot(BostonHousing, aes(log2(ptratio), log2(crim))) + geom_point(alpha = 0.5)
```

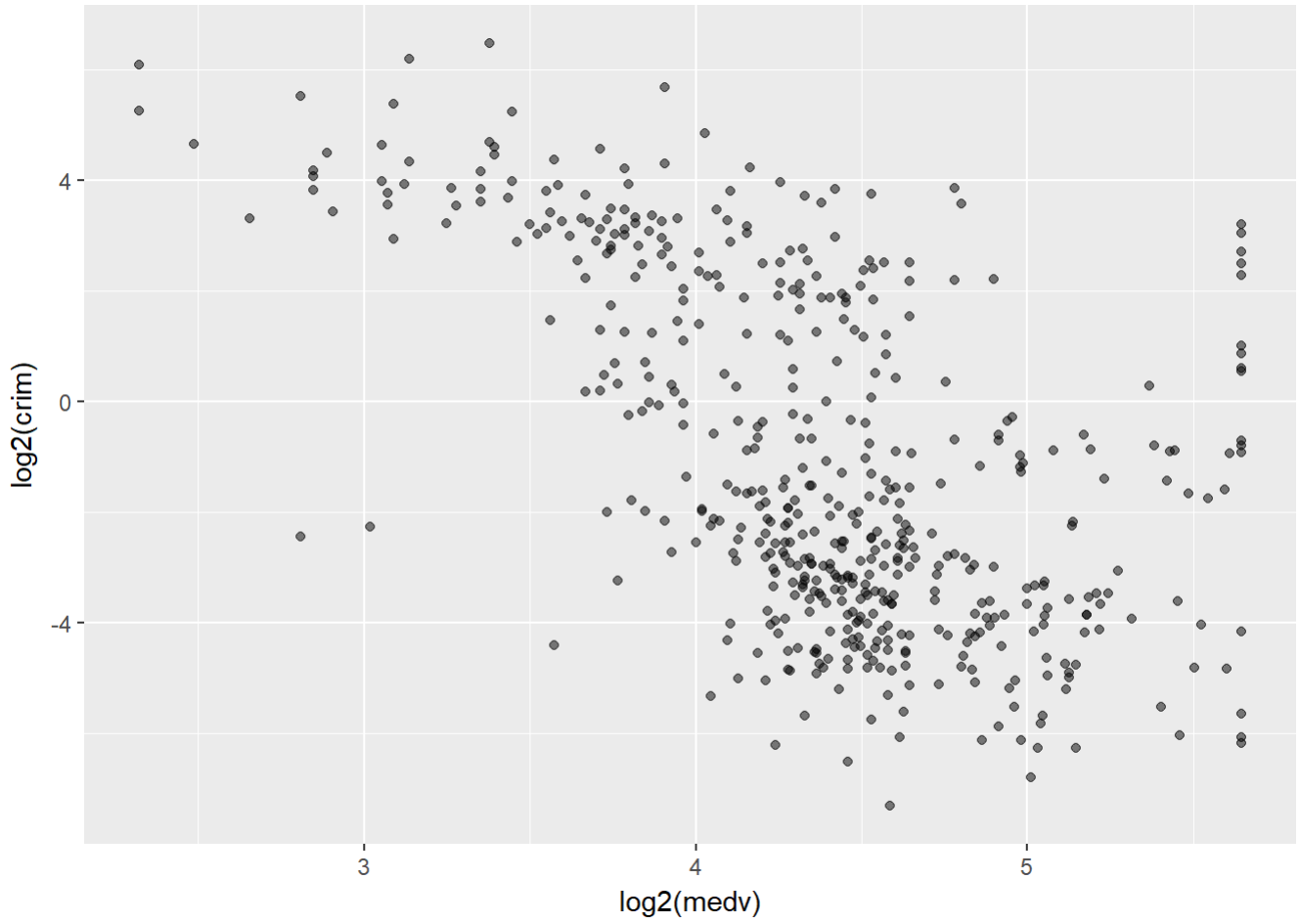



```
# per capita crime rate by town vs percentage of lower status of the population
ggplot(BostonHousing, aes(log2(lstat), log2(crim))) + geom_point(alpha = 0.5)
```



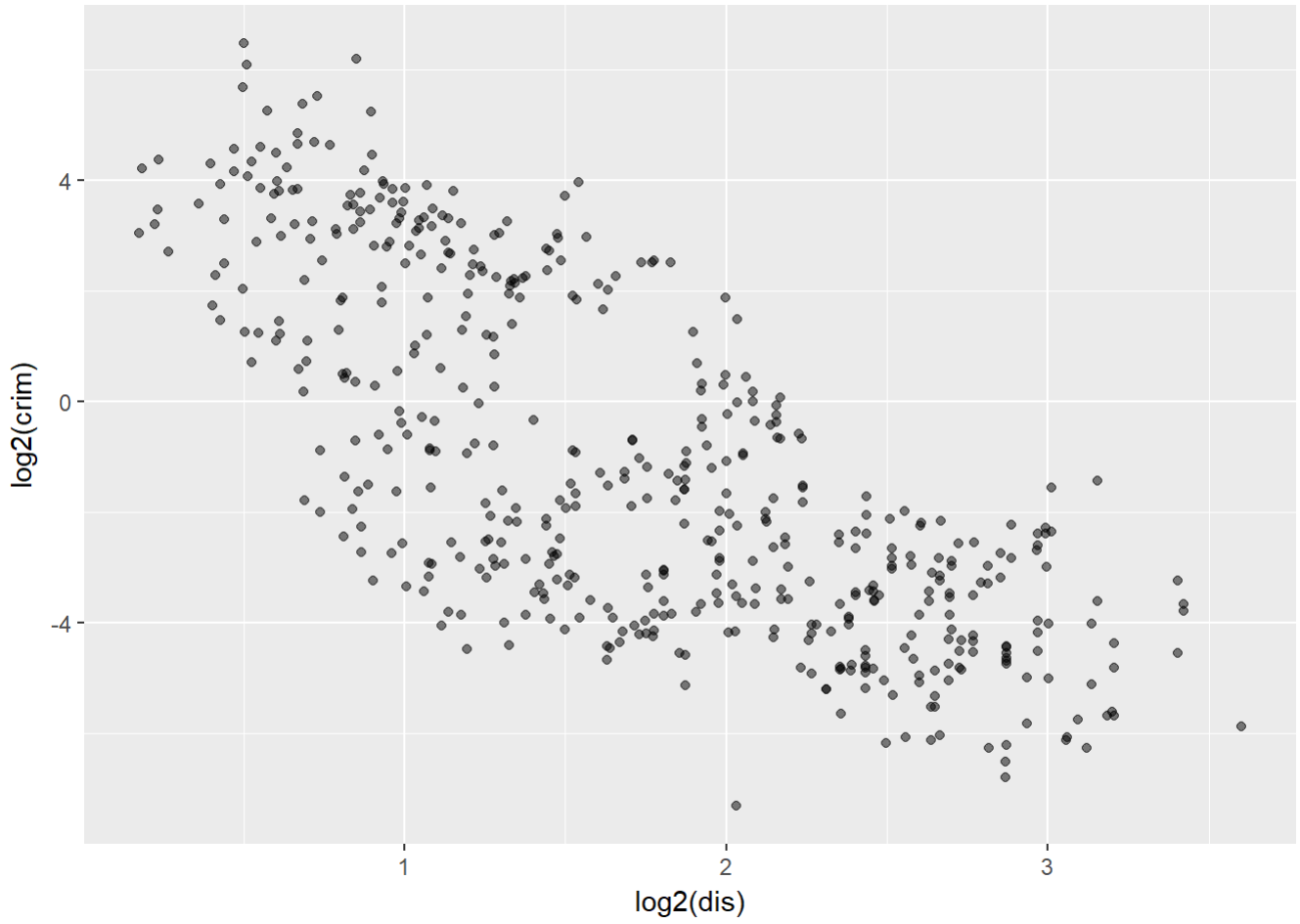

```
#We see a slight positively linear relationship.
```

```
# per capita crime rate by town vs median value of owner-occupied homes in USD 1000's
ggplot(BostonHousing, aes(log2(medv), log2(crim))) + geom_point(alpha = 0.5)
```



“dis” is considered as a suitable predictor with the observed negative linear pattern.

```
# per capita crime rate by town vs weighted distances to five Boston employment centres
ggplot(BostonHousing, aes(log2(dis), log2(crim))) + geom_point(alpha = 0.5)
```



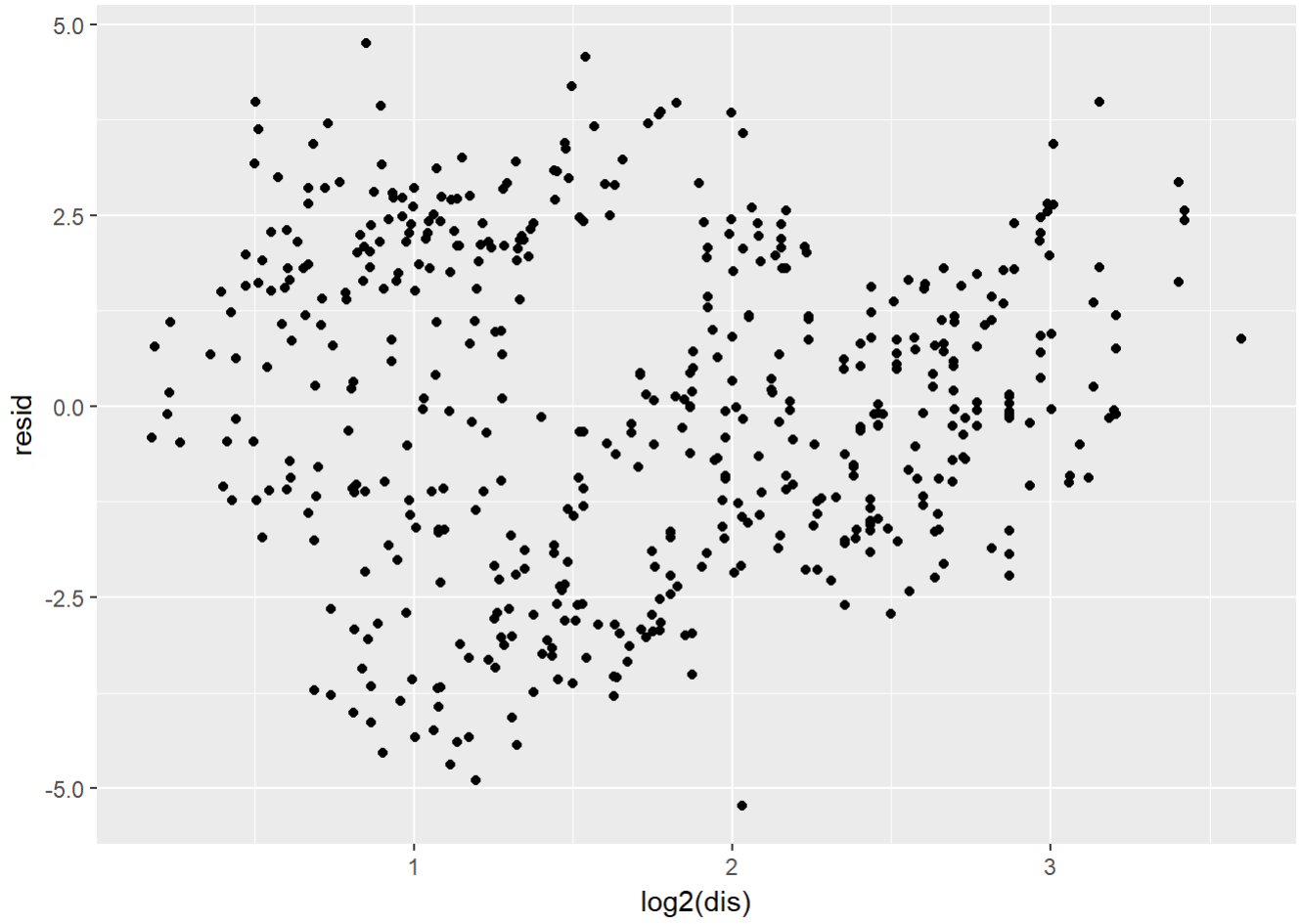

```
# fitting the model
fit <- lm(log2(crim) ~ log2(dis), data=BostonHousing)
fit
```

```
##
## Call:
## lm(formula = log2(crim) ~ log2(dis), data = BostonHousing)
##
## Coefficients:
## (Intercept)    log2(dis)
##       3.983       -2.981
```

Problem 2

Plot the residuals of the fitted model from Problem 1 against the predictor variable already in the model and against other potential predictor variables in the dataset. Comment on what you observe in each residual plot. (You do not need to include plots for predictor variables not in the model where you observe no systematic patterns in them.)

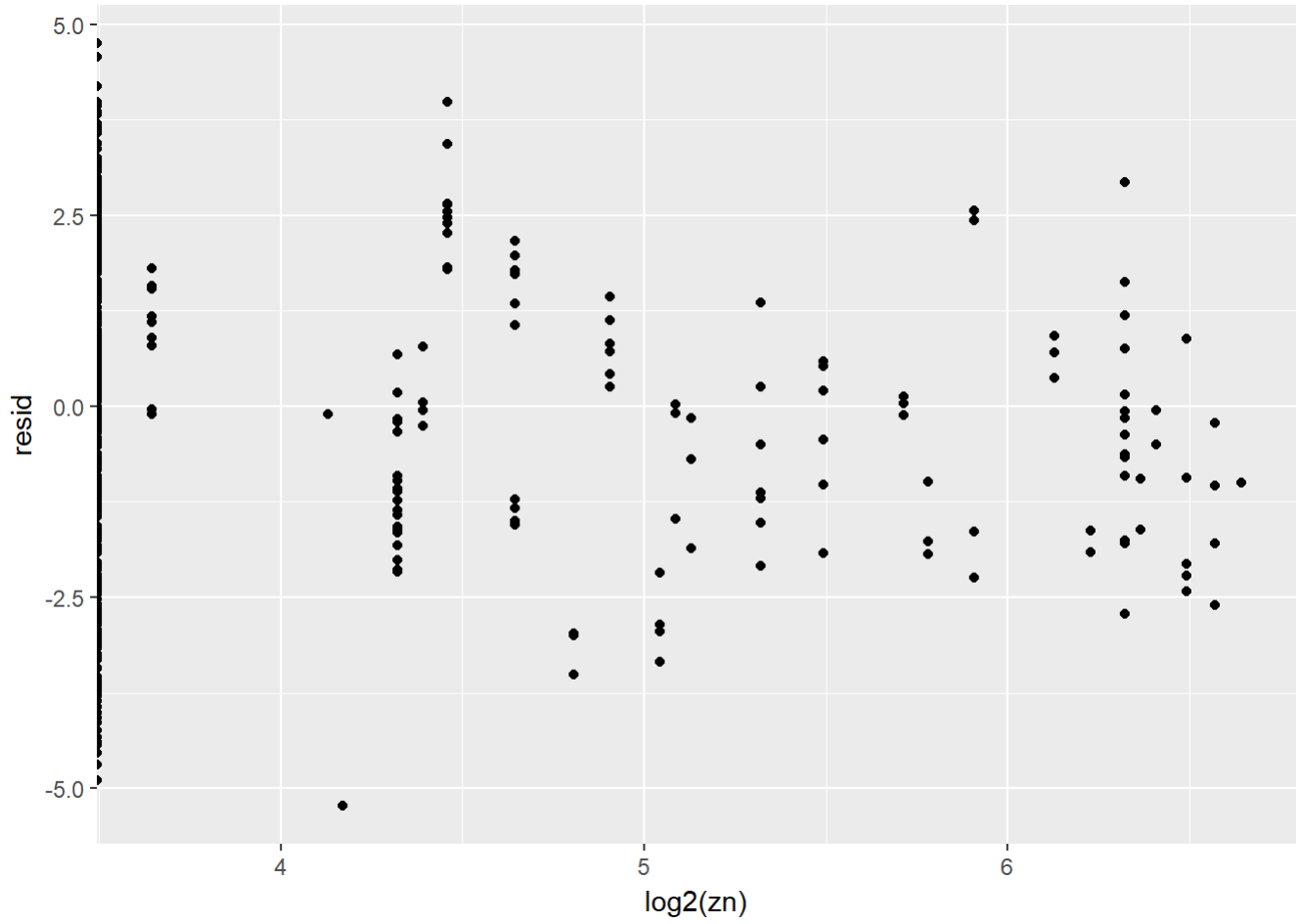
```
# Plotting the residuals of the fitted model from Problem 1 against the predictor variable alrea
dy in the model.

BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(dis), y=resid))
```



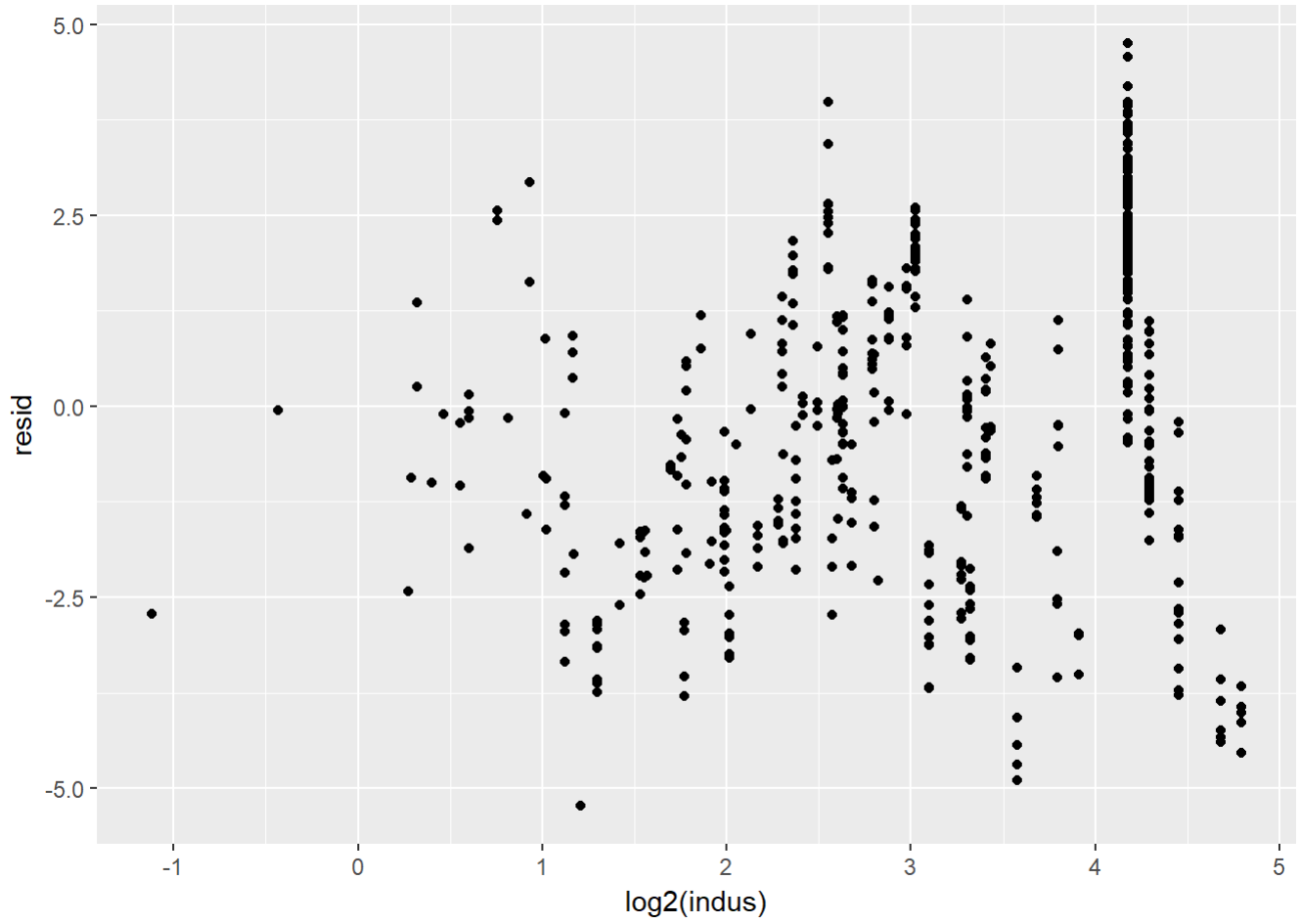

```
# the graph looks pretty random
```

Residual plot for variables not in model.

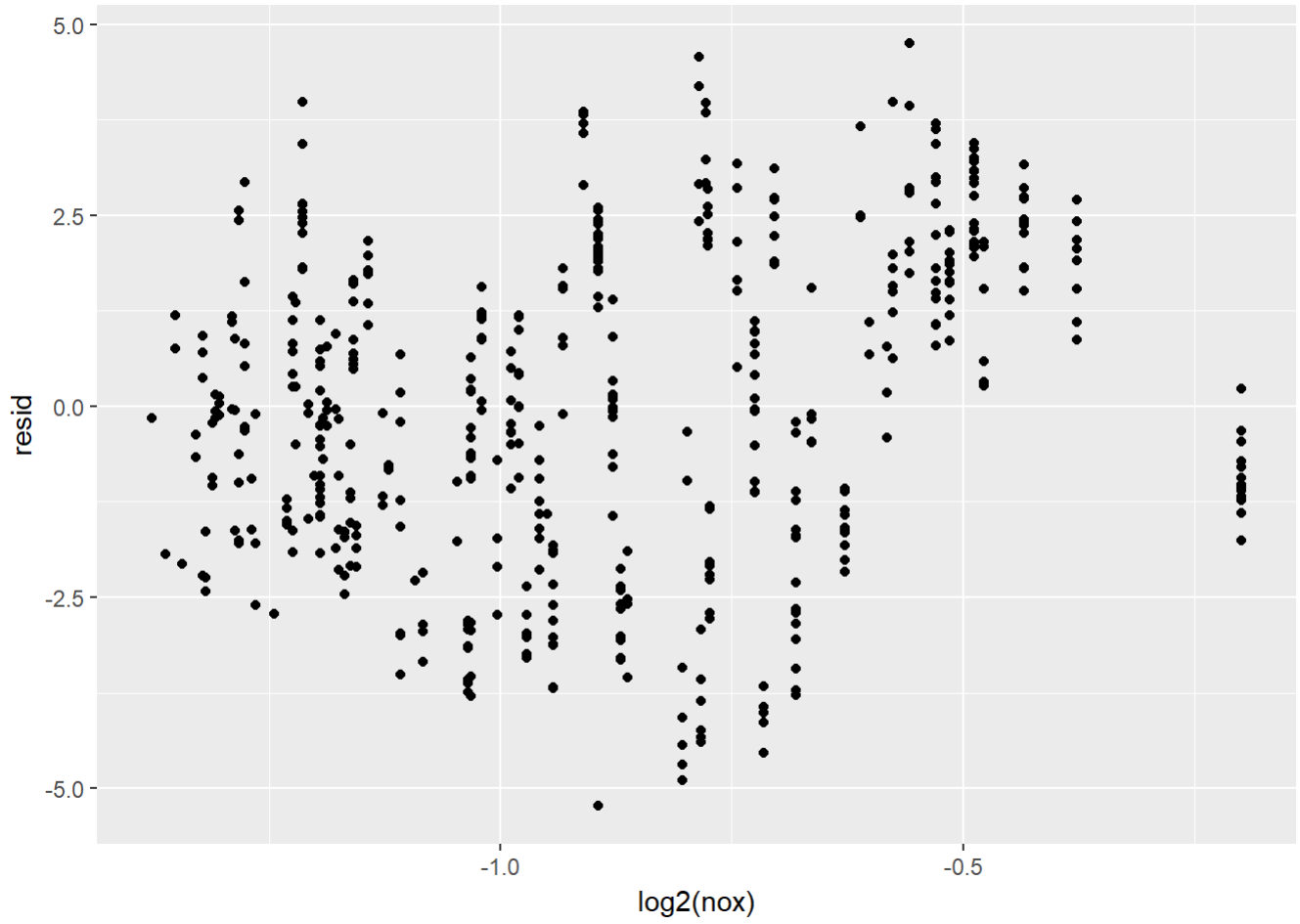
```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(zn), y=resid))
```



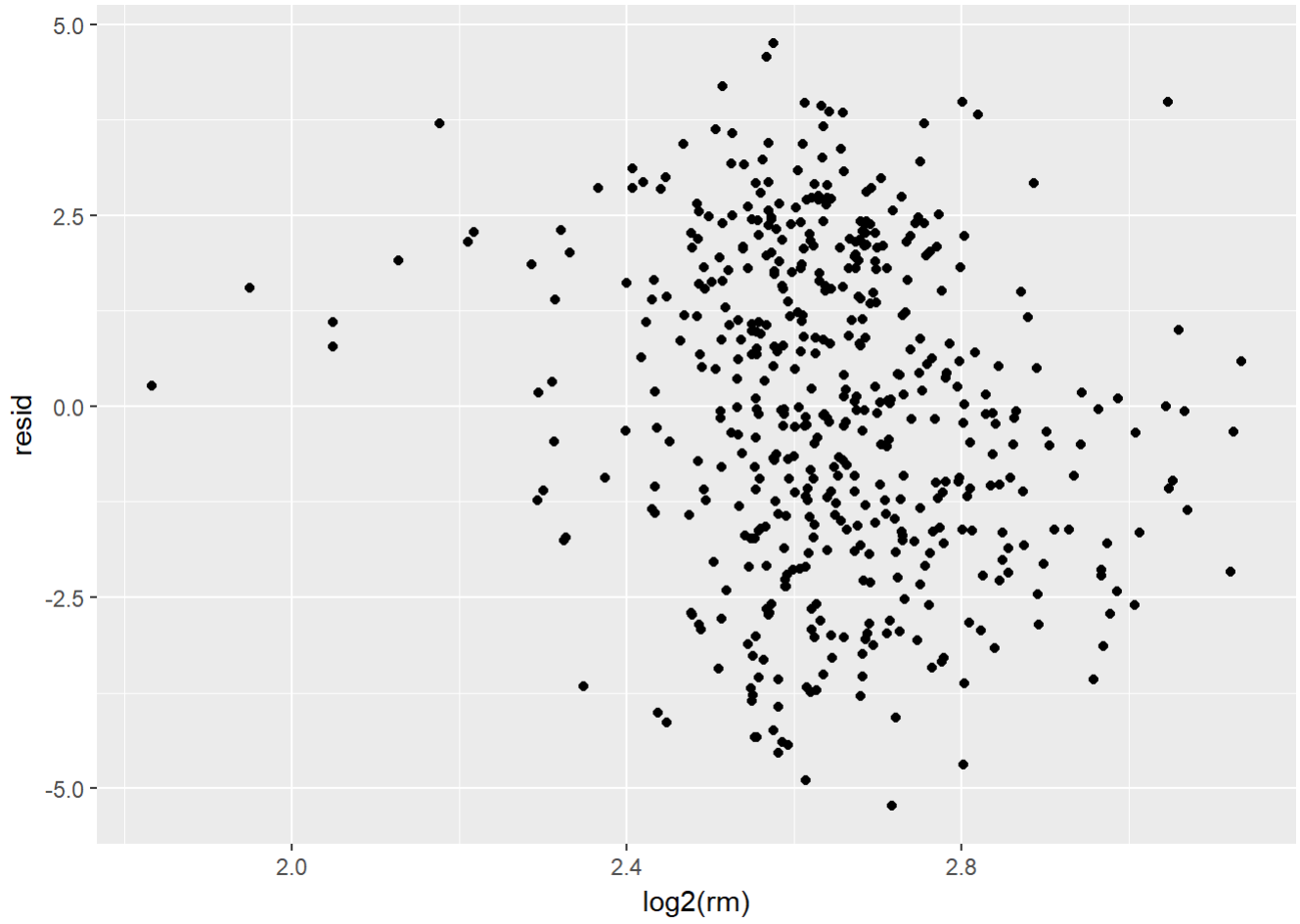

```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(indus), y=resid))
```



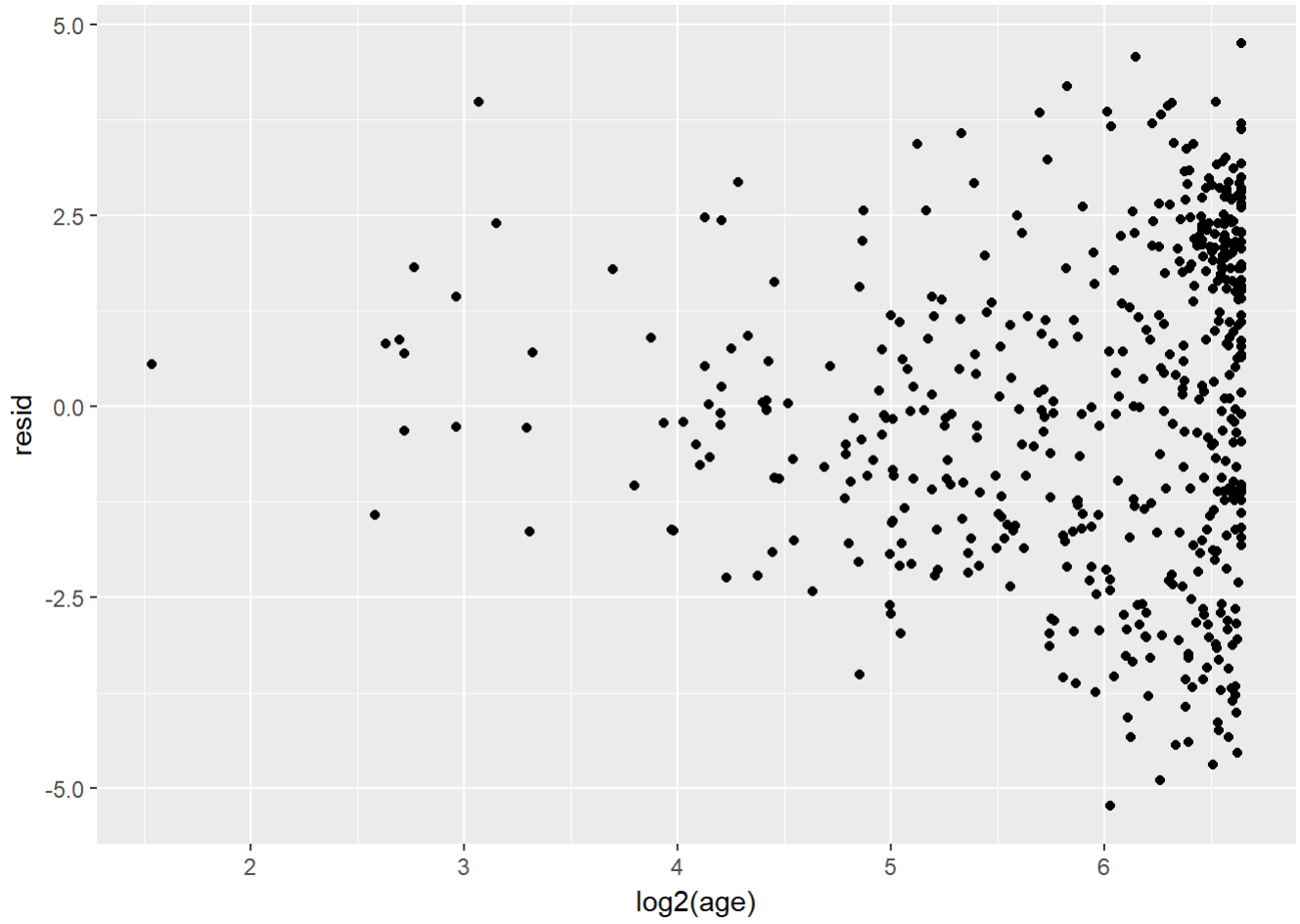

```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(nox), y=resid))
```



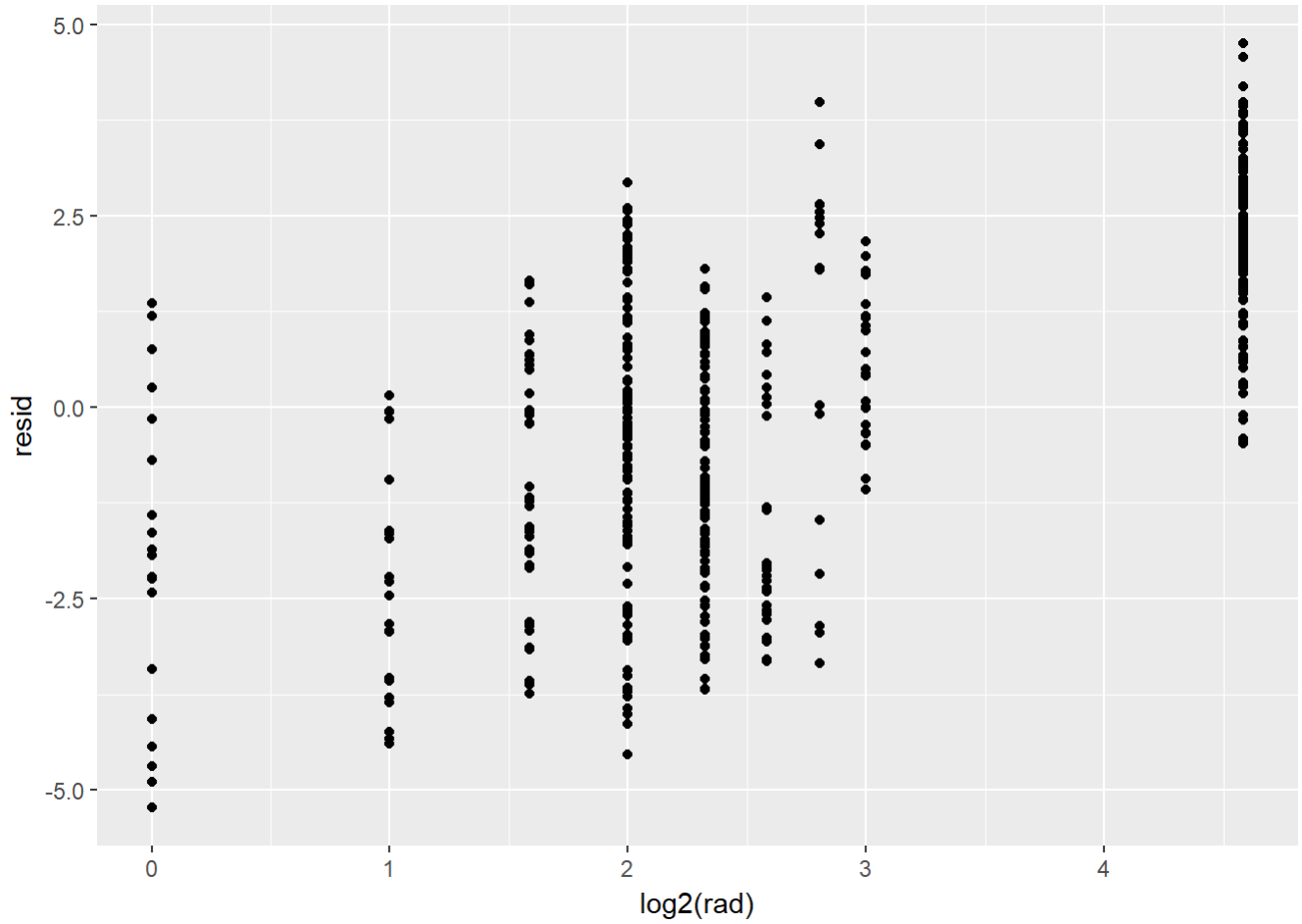

```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(rm), y=resid))
```




```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(age), y=resid))
```

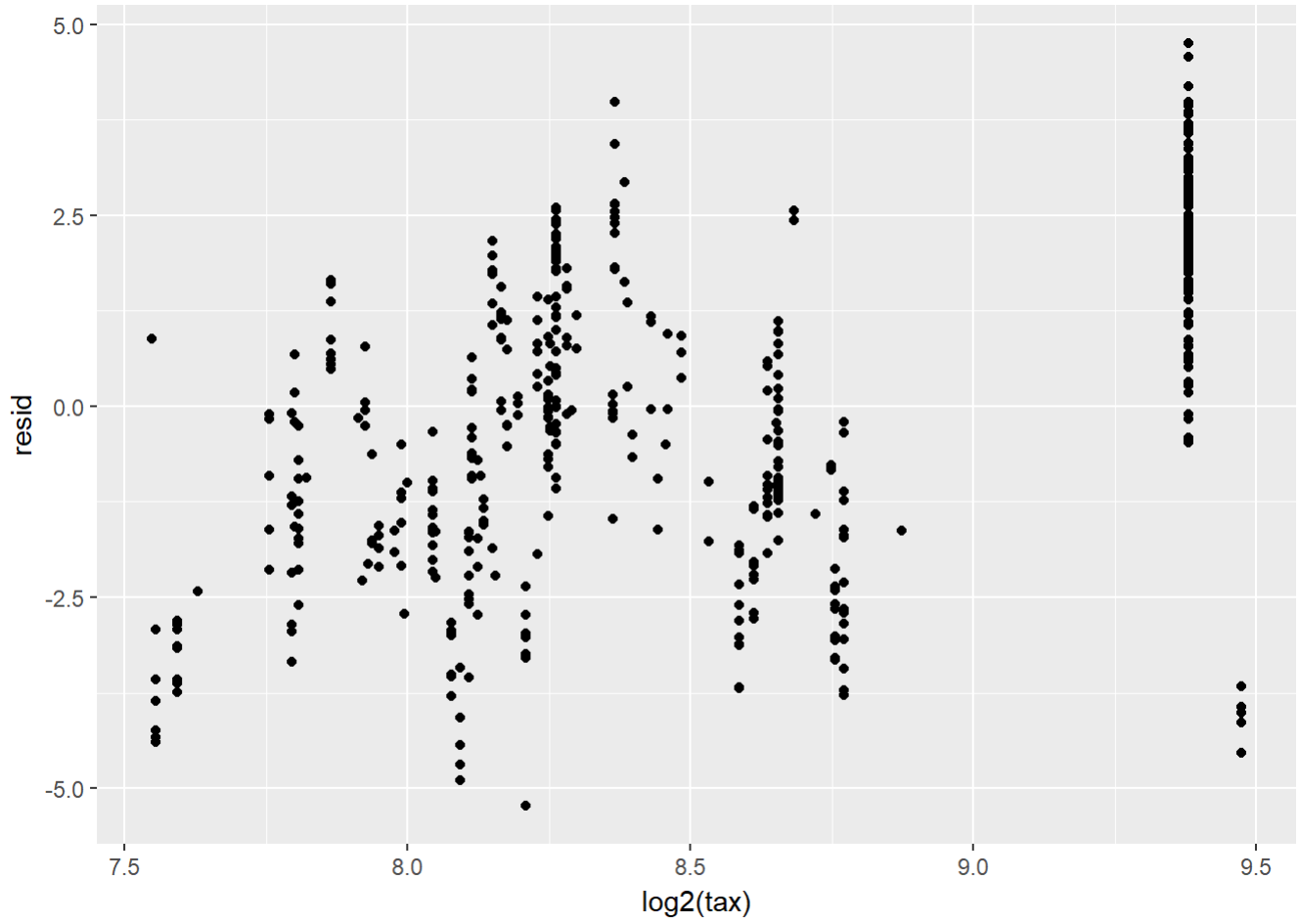



```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(rad), y=resid))
```

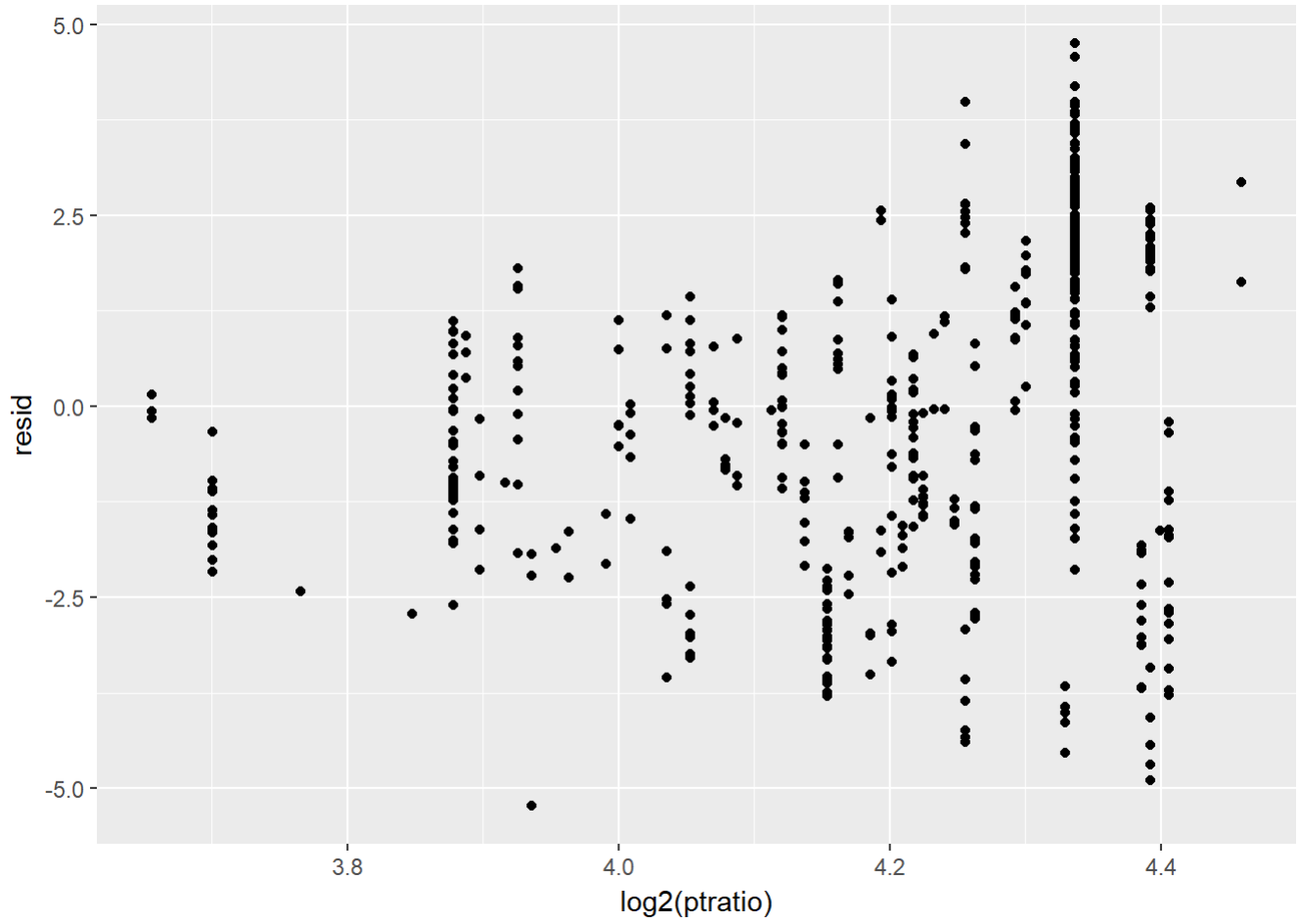


we see that this plot is not randomly scattered and seems to have a serious correlation so this variable can be potentially included in the model. Lets check other variables too.

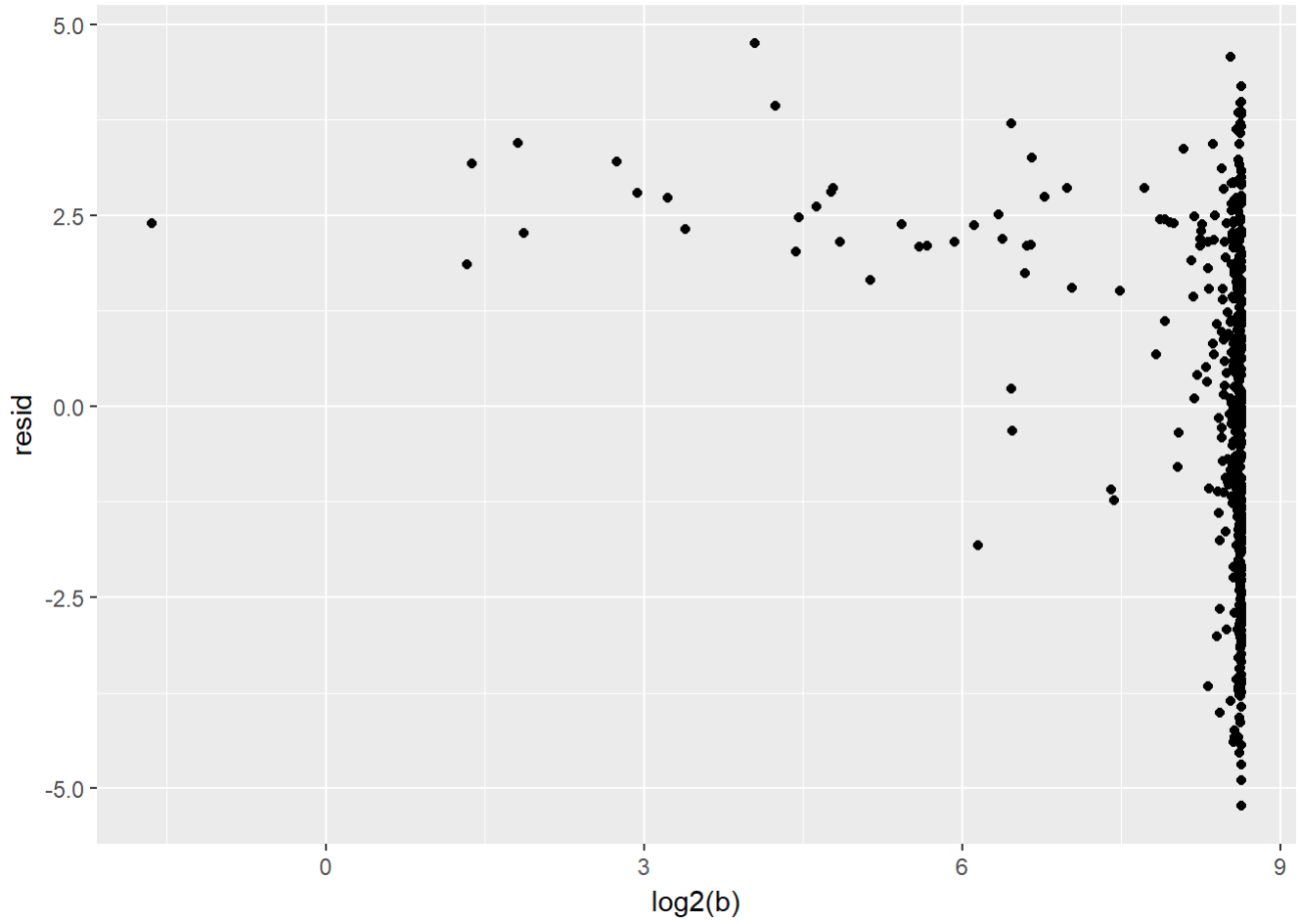
```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(tax), y=resid))
```



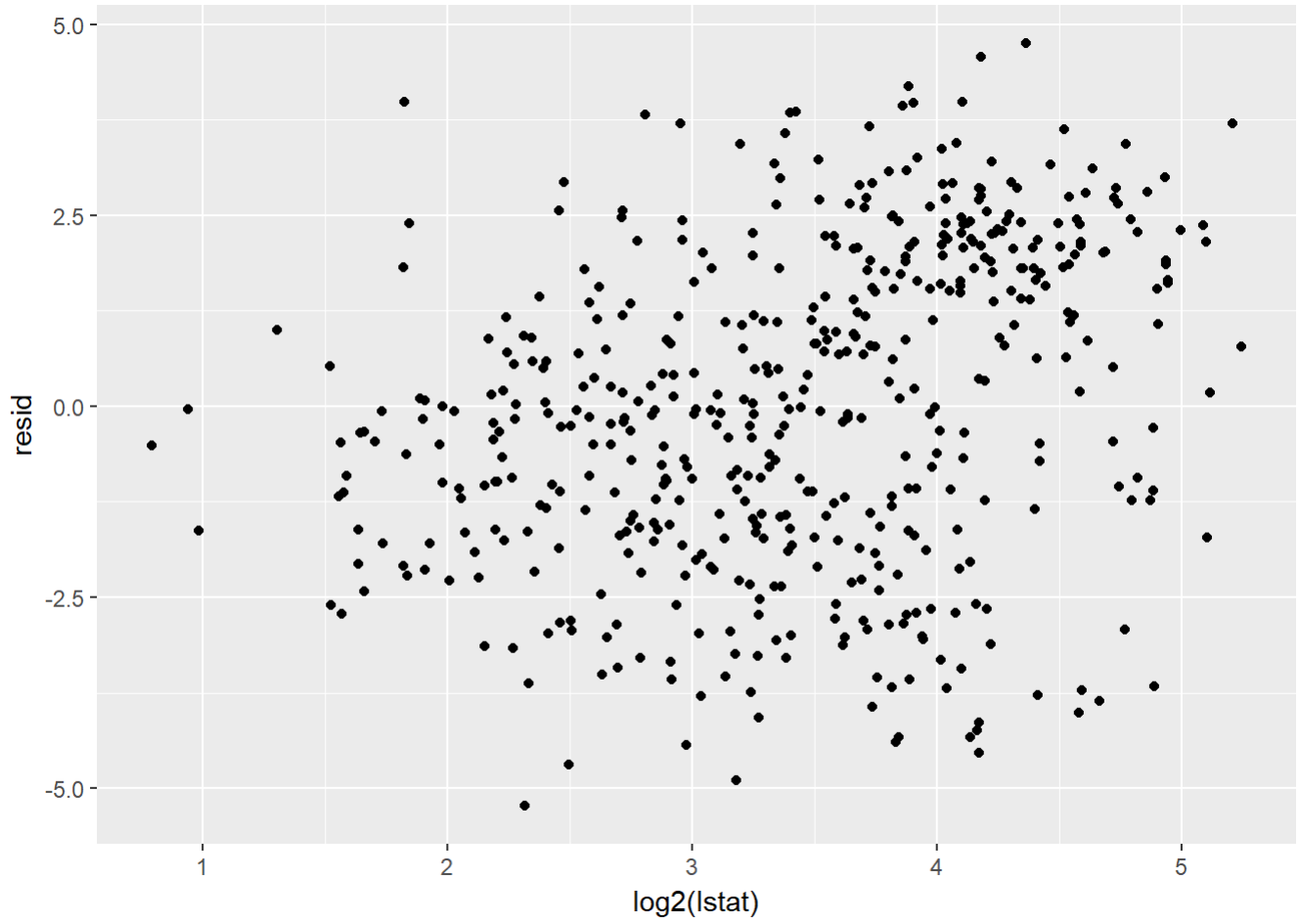

```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(ptratio), y=resid))
```



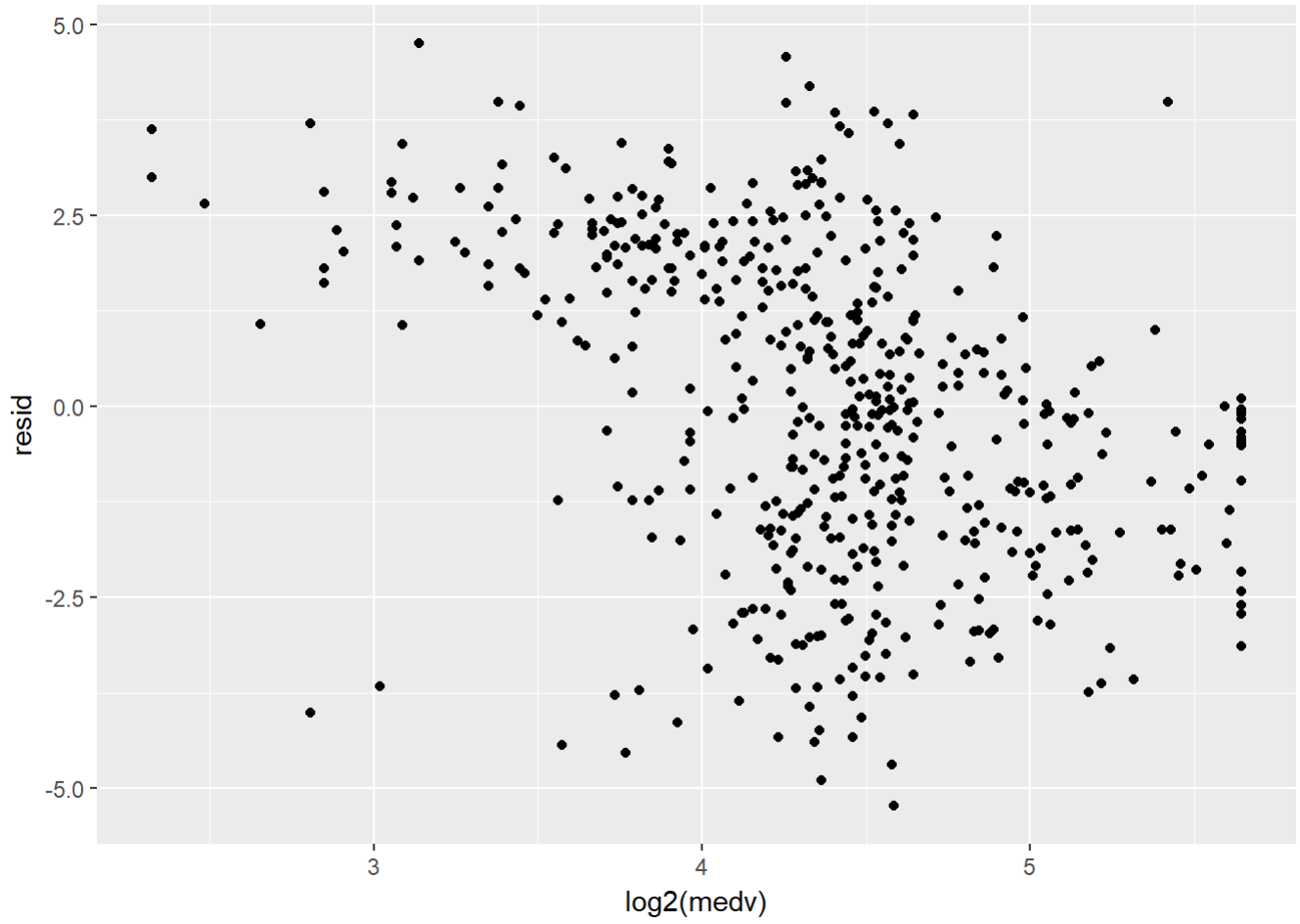

```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(b), y=resid))
```




```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(lstat), y=resid))
```




```
BostonHousing %>% add_residuals(fit) %>%
  ggplot() + geom_point(aes(x=log2(medv), y=resid))
```



we see that the plot with “rad” variable is not randomly scattered and seems to have a serious correlation, so this variable can be potentially included in the model.

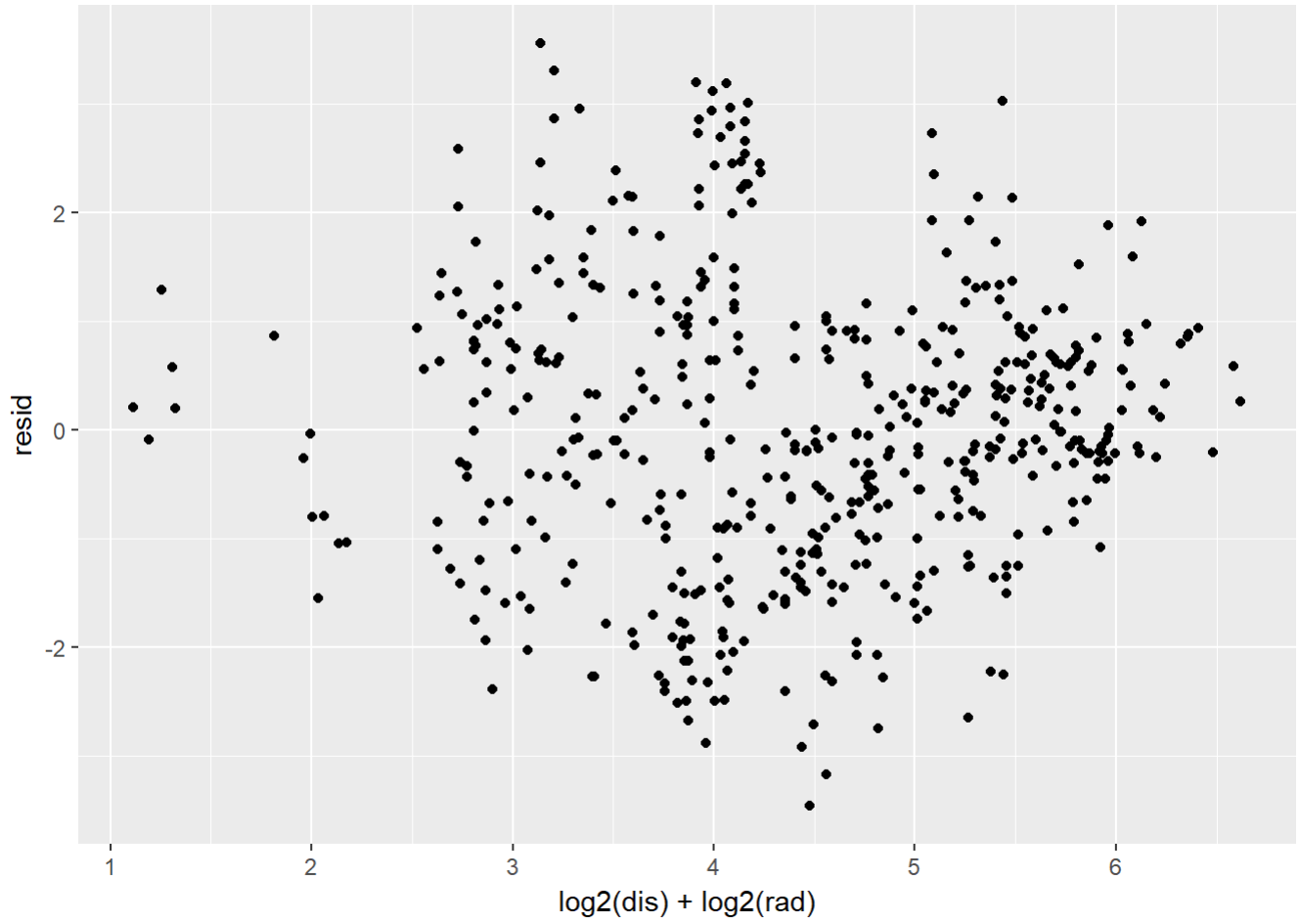
Problem 3

Fit a new model for predicting per capita crime rate by town, adding or removing variables based on the residual plots from Problem 2.

```
fit3 <- lm(log2(crim) ~ log2(dis)+log2(rad), data=BostonHousing)
fit3
```

```
##
## Call:
## lm(formula = log2(crim) ~ log2(dis) + log2(rad), data = BostonHousing)
##
## Coefficients:
## (Intercept)    log2(dis)    log2(rad)
##      -2.422       -1.641        1.525
```

```
BostonHousing %>% add_residuals(fit3) %>%
  ggplot() + geom_point(aes(x=log2(dis) + log2(rad) , y=resid))
```




```
# Based on the model predicted, the residual plot is made and it is evident that the points in t
he plot are randomly scattered.
```

PART B

Problem 4

Write a function that performs cross-validation for a linear model (fit using lm) and returns the average root-mean-square-error across all folds. The function should take as arguments (1) a formula used to fit the model, (2) a dataset, and (3) the number of folds to use for cross-validation. The function should partition the dataset, fit a model on each training partition, make predictions on each test partition, and return the average root-mean-square-error.

```
cross_validation <- function(fitted, datasetq4, k){
  Housing_cv <- crossv_kfold(datasetq4, k)
  Housing_cv = Housing_cv %>% mutate(fit = purrr::map(train, ~lm(fitted, data = .))) %>%
    mutate(rmse_val = map2_dbl(fit, test, rmse))
    mean_rmse <- mean(Housing_cv$rmse_val)
    return(mean_rmse)
}

fitted <- log2(crim) ~ log2(dis)+log2(rad)

cross_validation(fitted, BostonHousing, 10)
```

```
## [1] 1.312546
```

Problem 5

Use your function from Problem 4 to compare the models you used from Part A with 5-fold cross-validation. Report the cross-validated root-mean-square-error for the models from Problems 1 and 3. Which model was more predictive?

```
#fit is the model being used in Problem 1.

#fit <- lm(log2(crim) ~ log2(dis), data=BostonHousing)
#fit

cross_validation(fit, BostonHousing, 5)
```

```
## [1] 2.085896
```

```
#fit3 is the model being used in Problem 3.

#fit3 <- lm(log2(crim) ~ log2(dis)+log2(rad), data=BostonHousing)
#fit3

cross_validation(fit3, BostonHousing, 5)
```

```
## [1] 1.326191
```

```
# The model derived in problem 3 is predictable and it's evident from the cross-validated root-m
ean-square-error value. The previous model's cv-rmse of model dervied from problem 1 is bit high
 comparitively.
```

Part C

Problem 6

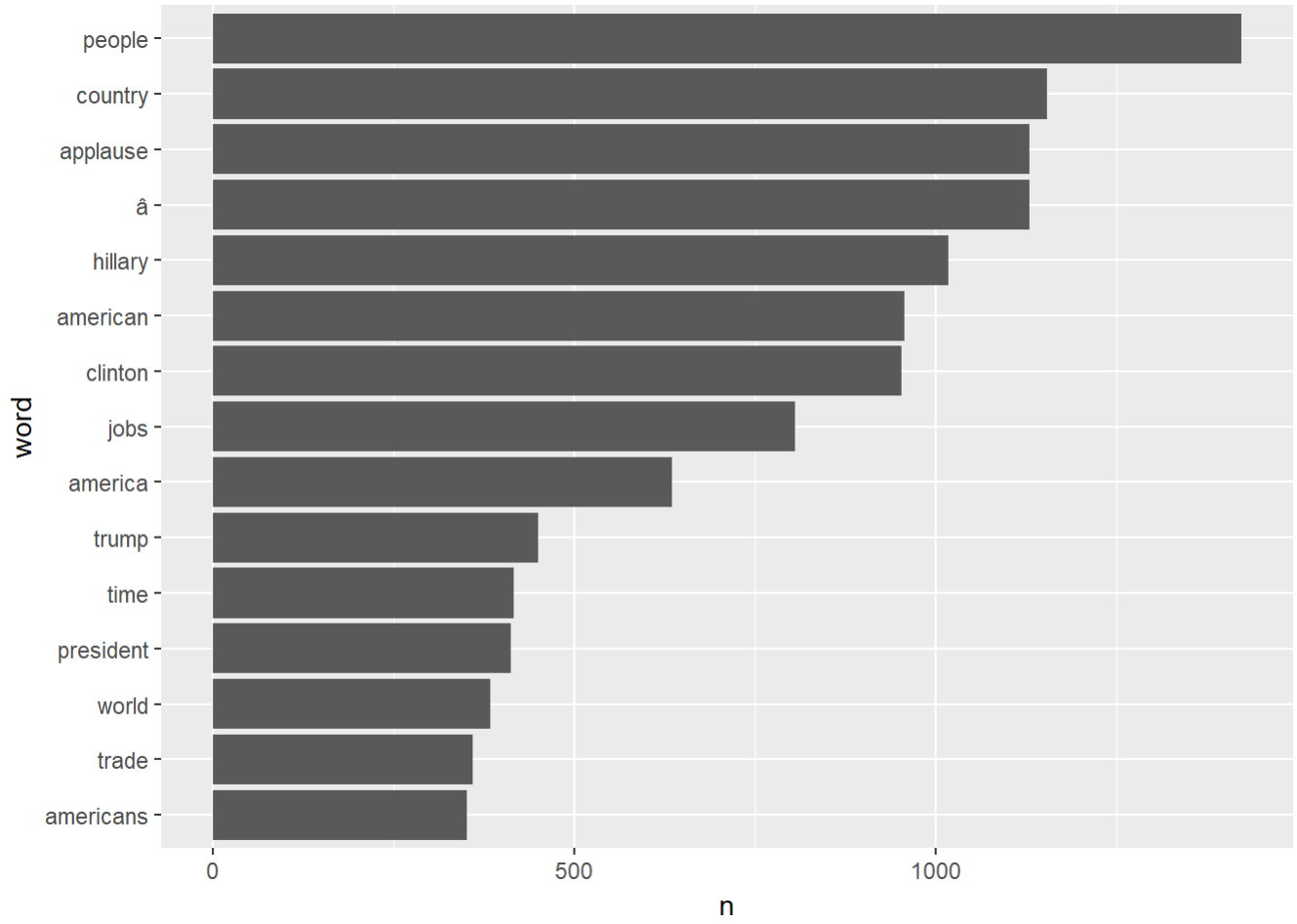
Import the text from all 56 Donald Trump speeches into R and tokenize the data into a tidy text data frame, using words as tokens. After removing stop words and the word “applause”, plot the top 15 most common words used in Trump’s speeches

```
# rm("fit1", "fit2", "fit3")

C_Data <- readLines("C:/Users/mouni/Downloads/Sem 1/R Lang/Datasets/full_speech.txt")

Trump <- tibble(line=1:length(C_Data), text=C_Data)

Trump_words <- unnest_tokens(Trump, word, text)
print(Trump_words, n=10)
```

```
## # A tibble: 236,368 x 2
##     line word
##    <int> <chr>
##  1     1 trump
##  2     1 wow
##  3     1 whoa
##  4     1 that
##  5     1 is
##  6     1 some
##  7     1 group
##  8     1 of
##  9     1 people
## 10     1 thousands
## # ... with 2.364e+05 more rows
```

```
Trump_words %>%
  anti_join(stop_words, by="word") %>%
  count(word, sort=TRUE) %>%
  top_n(15) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x=word, y=n)) +
  geom_col() +
  coord_flip()
```

```
## Selecting by n
```

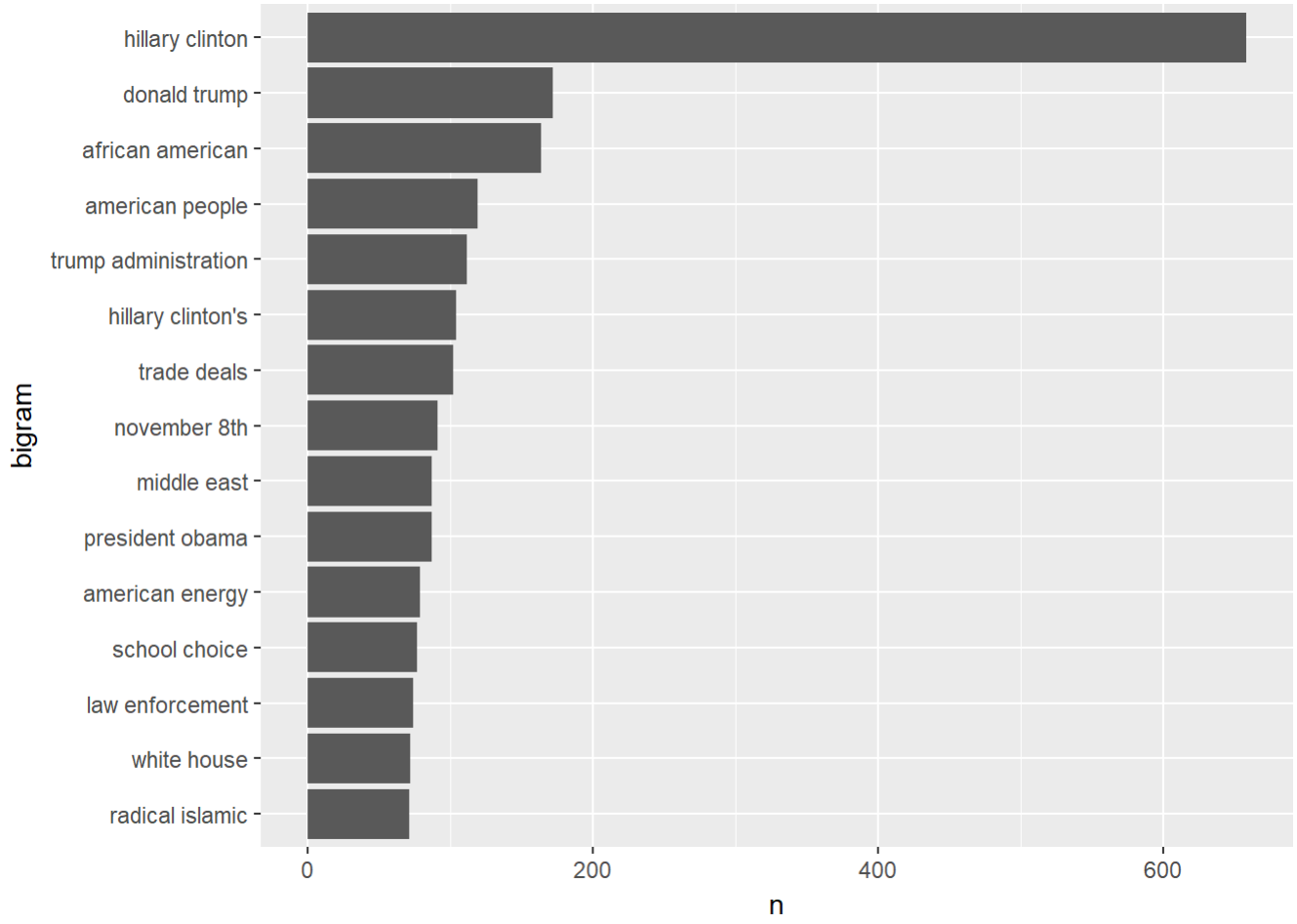


Problem 7

Re-tokenize the text of all 56 Donald Trump Speeches into a new tidy text data frame, using bigrams as tokens. Remove each bigram where either word is a stop word or the word “applause”. Then plot the top 15 most common bigrams in Trump’s speeches.

```
Trump_bigrams <- Trump %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

Trump_bigrams <- Trump_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word) %>%
  filter(!word1 == "applause") %>%
  filter(!word2 == "applause")

Trump_bigrams %>%
  unite(bigram, word1, word2, sep = " ") %>%
  count(bigram, sort=TRUE) %>%
  top_n(15) %>%
  mutate(bigram = factor(bigram, levels = rev(unique(bigram))))  %>%
  ggplot(aes(x=bigram, y=n)) +
  geom_col() +
  coord_flip()
```

```
## Selecting by n
```



Problem 8

We would like to do a sentiment analysis of Donald Trump's speeches. In order to make sure sentiments are assigned to appropriate contexts, first tokenize the speeches into bigrams, and then filter out all bigrams where the first word is any of "not", "no", or "never".

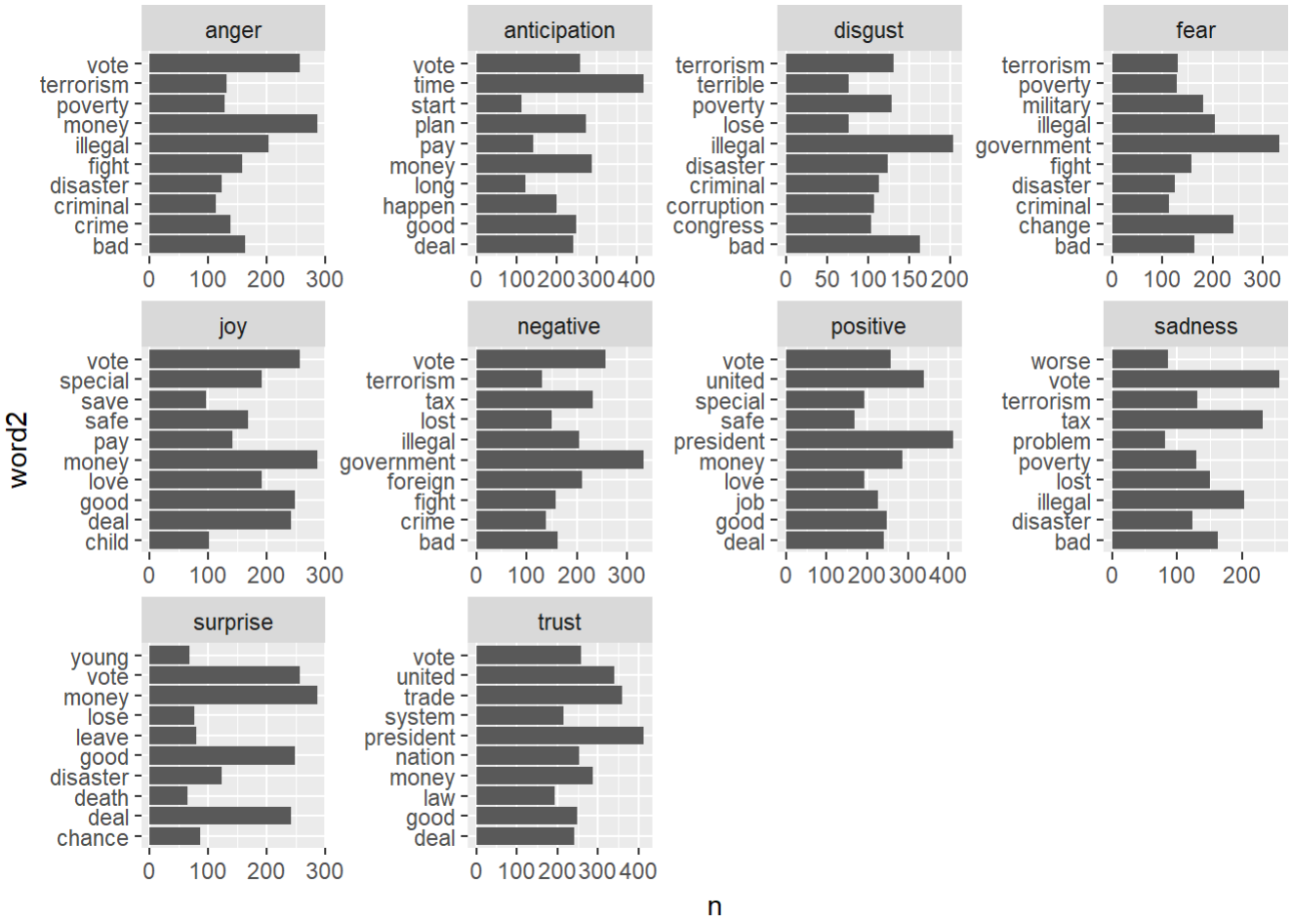
Now consider only the second word of each bigram. After filtering out the words "applause" and "trump", create a plot of the 10 most common words in Trump's speeches that are associated with each of the 10 sentiments in the "nrc" lexicon.

```
Trump_sentiment <- Trump %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

Trump_sentiment <- Trump_sentiment %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  filter(!word1 == "not" | !word1 == "no" | !word1 == "never") %>%
  filter(!word2 == "applause" & !word2 == "trump")

nrcjoy <- get_sentiments("nrc")

Trump_sentiment %>%
  inner_join(nrcjoy, by = c("word2" = "word")) %>%
  group_by(sentiment) %>%
  count(word2, sort = TRUE) %>%
  mutate(word2 = reorder(word2, n)) %>%
  top_n(10) %>%
  ggplot(aes(word2, n)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales="free") +
  coord_flip()
```

```
## Warning in mutate_impl(.data, dots): Unequal factor levels: coercing to character
```

```
## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector

## Warning in mutate_impl(.data, dots): binding character and factor vector, coercing into
## character vector
```

```
## Selecting by n
```



Problem 9

Create an S3 class called tidy_corpus which inherits from a tibble (tbl_df). A tidy_corpus is a tidy text data frame that always has at least one column called token, which gives the tokens, and an attribute called token_type which gives the type of token. A tidy_corpus object may also have an additional attribute called n when token_type is “ngrams”.

Create a constructor function that creates tidy_corpus objects called tidy_corpus(src, token, …) which takes three parameters: . src is either a character vector or a directory of text files . token gives the kind of tokenization to be performed . … are additional arguments passed to unnest_tokens()

Write a plot() method for the tidy_corpus class that plots the top n most common tokens in the corpus where n is a user parameter defaulting to 10. Include an optional parameter for removing stop words. The removal of stop words should support at least words and bigrams.

Test your class and methods on the data in Part C.

```
# Creating an S3 class

C_Text <- read_lines("C:/Users/mouni/Downloads/Sem 1/R Lang/Datasets/full_speech.txt")

tidy_corpus <- function(src, token, ...) {
if (any(grepl(".*.txt", src))) {
src <- read_lines(src)
}
corpus_tbl <- tibble(line=1:length(src), words = src)
corpus <- unnest_tokens(corpus_tbl, token = token, output = token, input = words, ...)
corpus <- structure(corpus, class = c("tbl_df", "tbl", "data.frame"))
element <- structure(list(corpus = corpus, token_type = token, ...), class = c("tidy_corpus"))
return(element)
}

# determine object type (whether "S3") with otype using speech file and words as arguments to ti
dy_corpus function created.

otype(tidy_corpus("C:/Users/mouni/Downloads/Sem 1/R Lang/Datasets/full_speech.txt", token = "wor
ds"))
```

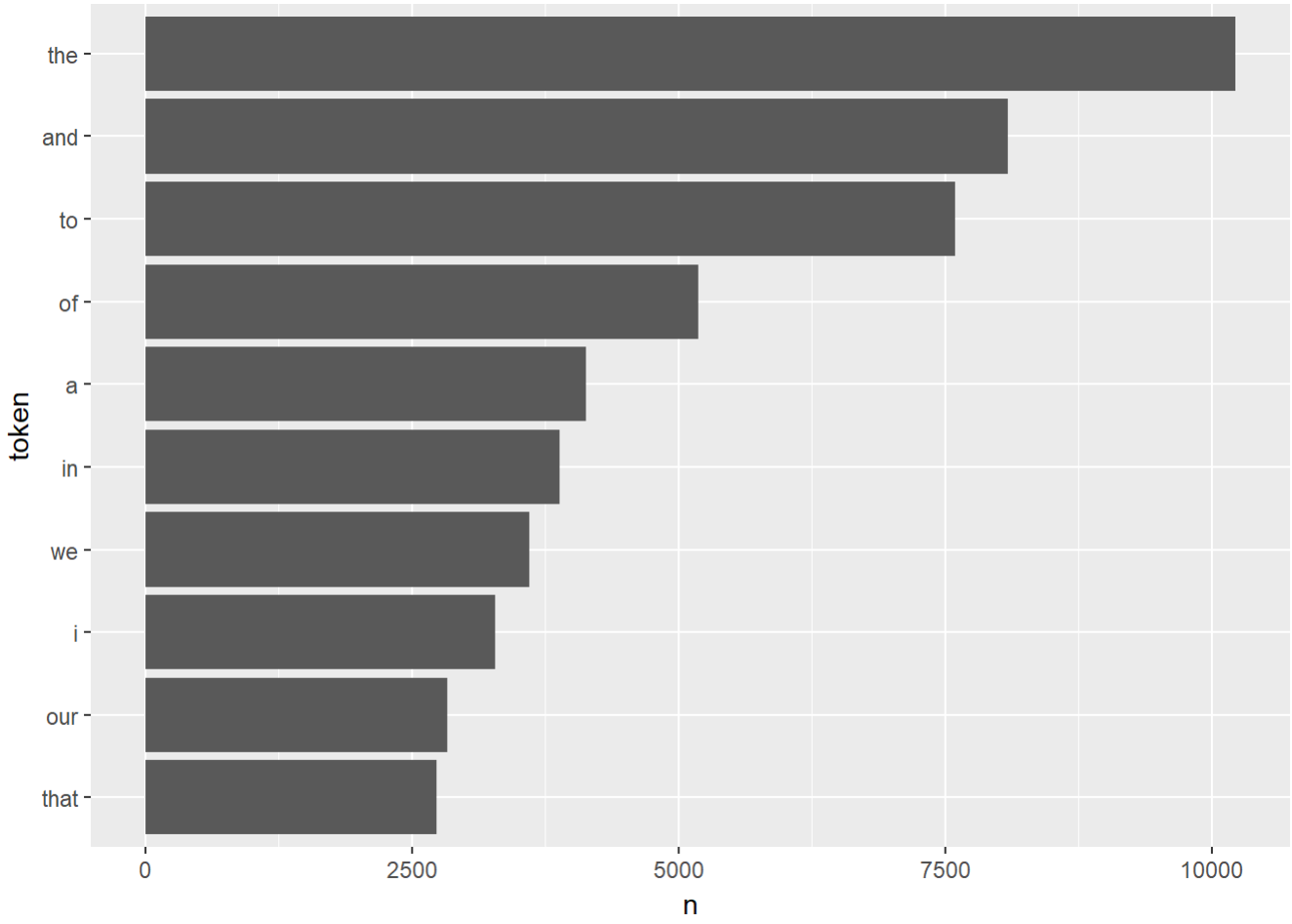
```
## [1] "S3"
```

```
# creating a function/method called plot

plot <- function(x) UseMethod("plot")

plot.tidy_corpus <- function(rand, n=10, rm_stop_words=FALSE) {
  if (length(rand)==2) {
    corpus <- rand$corpus
    token_type <- rand$token_type
    } else if (length(rand)==3) {
      corpus <- rand$corpus
      token_type <- rand$token_type
      n_grams <- rand$n
    }
  if (token_type=="words") {
    if (rm_stop_words==TRUE) {
      corpus %>%
        anti_join(stop_words, by=c("token"="word")) %>%
        count(token, sort=TRUE) %>%
        top_n(n) %>%
        mutate(token = reorder(token, n)) %>%
        ggplot(aes(x=token, y=n)) + geom_col() +
        coord_flip()
      } else {
        corpus %>%
          count(token, sort=TRUE) %>%
          top_n(n) %>%
          mutate(token = reorder(token, n)) %>%
          ggplot(aes(x=token, y=n)) + geom_col() +
          coord_flip()
      }
    } else if (token_type=="ngrams" && n_grams==2) {
      if (rm_stop_words==TRUE) {
        corpus <- corpus %>%
          separate(token, c("word1", "word2"), sep = " ")

        corpus <- corpus %>%
          filter((!word1 %in% stop_words$word)) %>%
          filter((!word2 %in% stop_words$word))

        corpus %>%
          unite(col = token, c("word1","word2"), sep=" ") %>%
          count(token, sort=TRUE) %>%
          top_n(n) %>%
          mutate(token = reorder(token, n)) %>%
          ggplot(aes(x=token, y=n)) + geom_col() +
          coord_flip()
        } else {
          corpus %>%
            count(token, sort=TRUE) %>%
            top_n(n) %>%
            mutate(bigrams = reorder(token, n)) %>%
            ggplot(aes(x=token, y=n)) + geom_col() +
            coord_flip()
```

```
    }
  } else {
    corpus %>%
      count(token, sort=TRUE) %>%
      top_n(n) %>%
      mutate(token = reorder(token, n)) %>%
      ggplot(aes(x=token, y=n)) + geom_col() +
      coord_flip()
  }
}
```

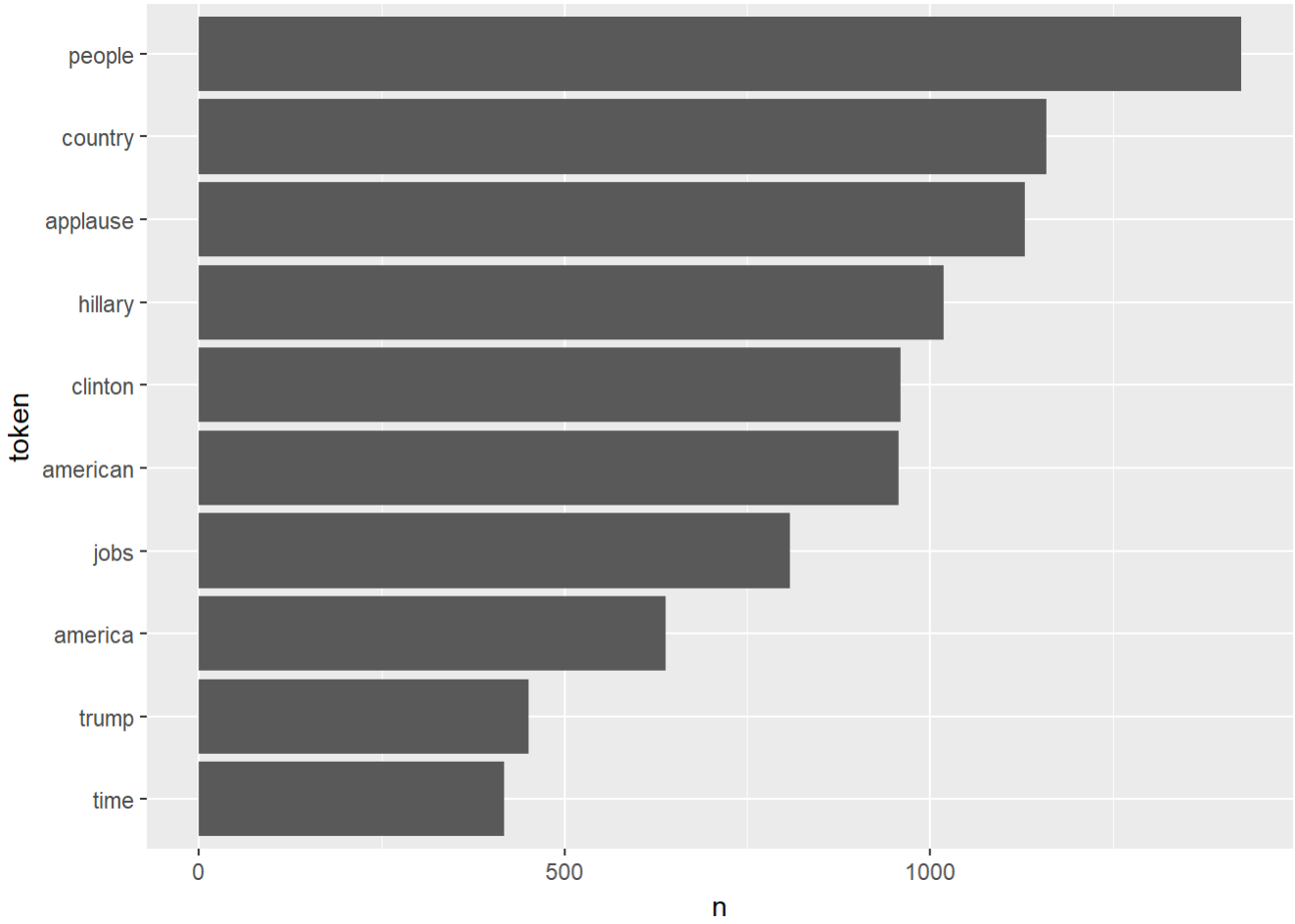
```
# check the tidy_corpus class with different arguments.

# (1) src as "c_Text" and tokens as "unigrams/words"
check_tidy_corpus <- tidy_corpus(src= C_Text, token= "words")
#check_tidy_corpus

# Non-removal of stop words
plot.tidy_corpus(check_tidy_corpus, n=10, rm_stop_words=FALSE)
```

```
## Selecting by n
```



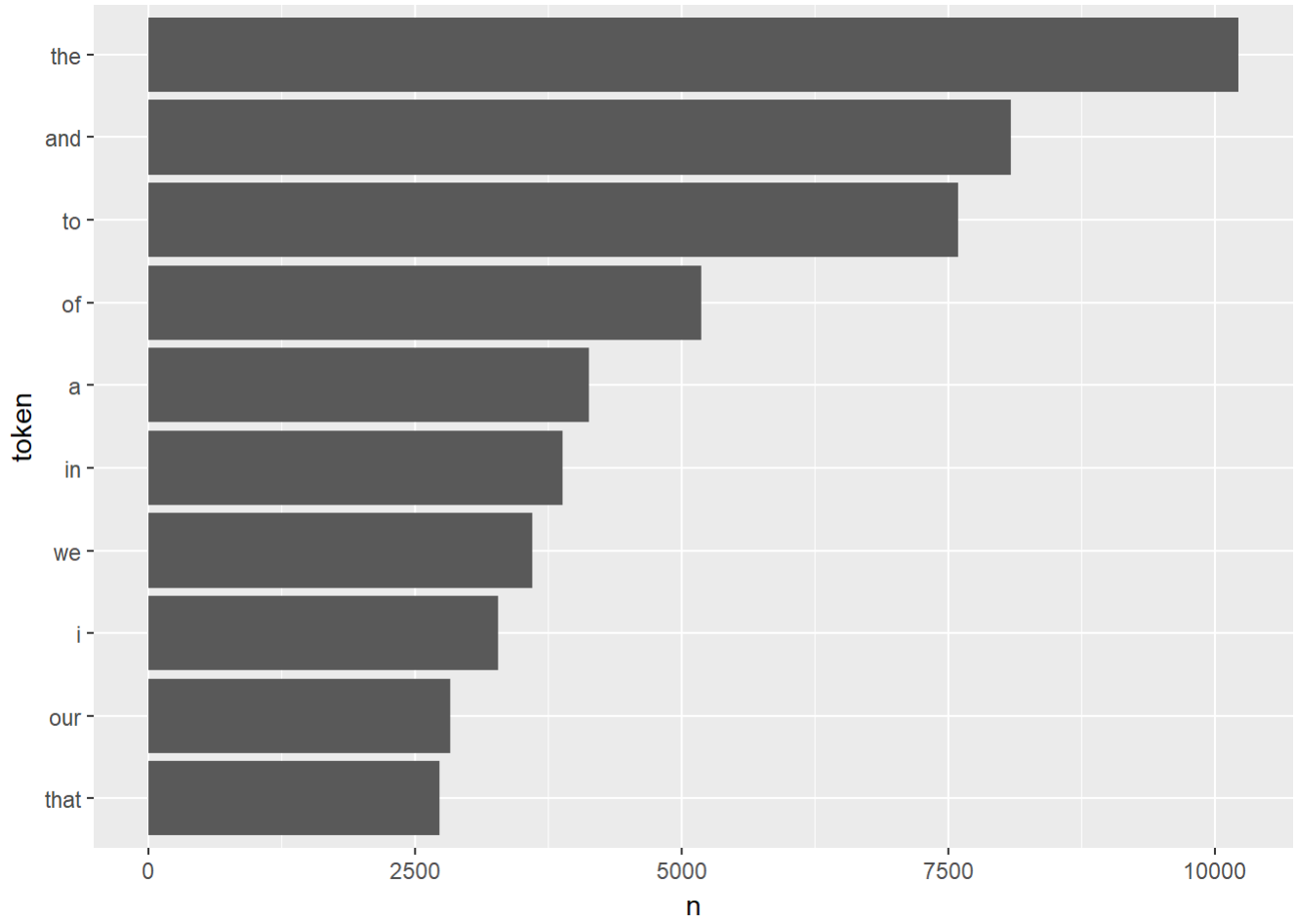

```
# Removal of stop words
plot.tidy_corpus(check_tidy_corpus, n=10, rm_stop_words=TRUE)
```

```
## Selecting by n
```



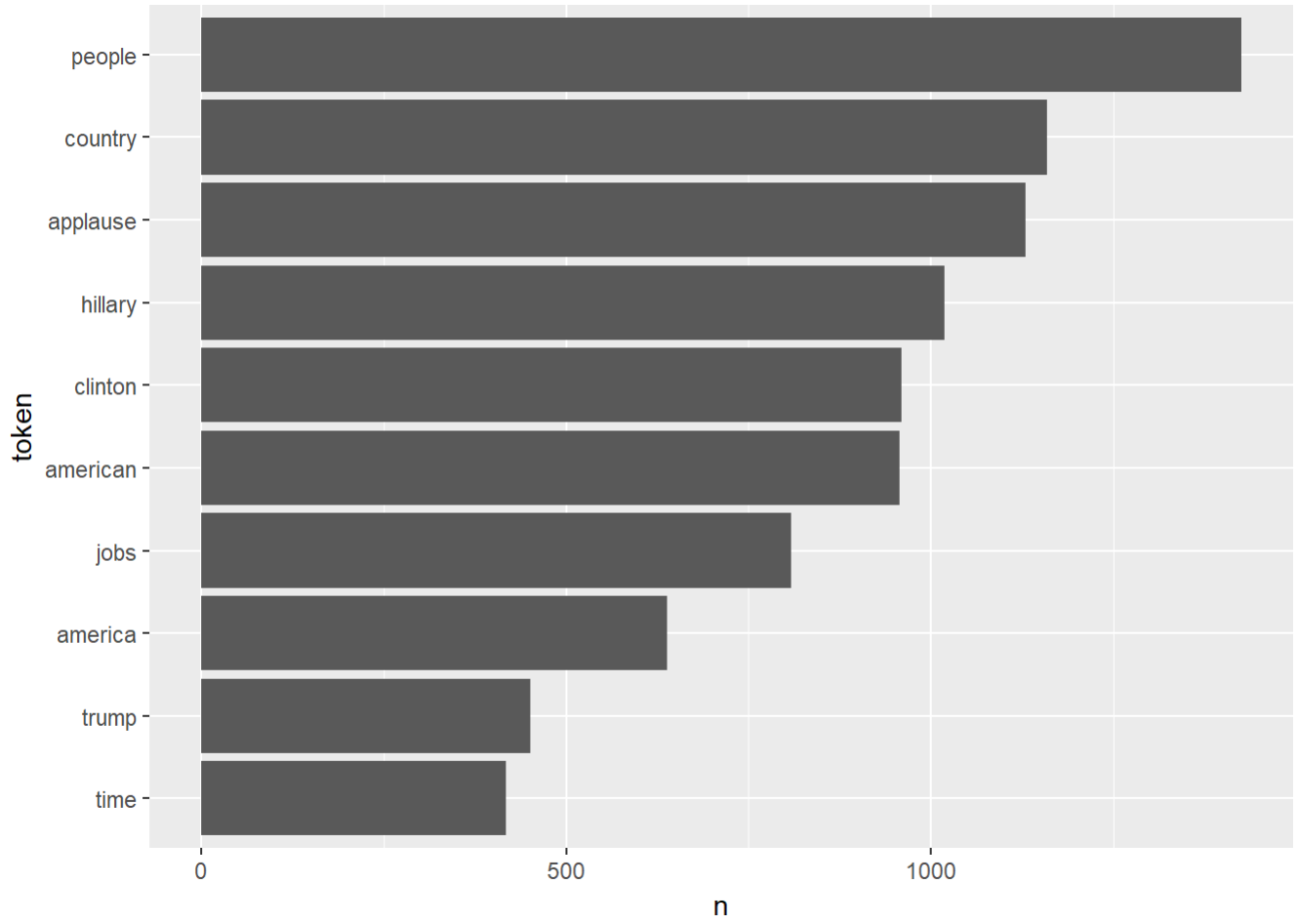

```
# (2) src as a "directory" and tokens as "unigrams/words"
check_tidy_corpus <- tidy_corpus("C:/Users/mouni/Downloads/Sem 1/R Lang/Datasets/full_speech.tx
t", token = "words")
#check_tidy_corpus

# Non-removal of stop words
plot.tidy_corpus(check_tidy_corpus, n=10, rm_stop_words=FALSE)
```

```
## Selecting by n
```



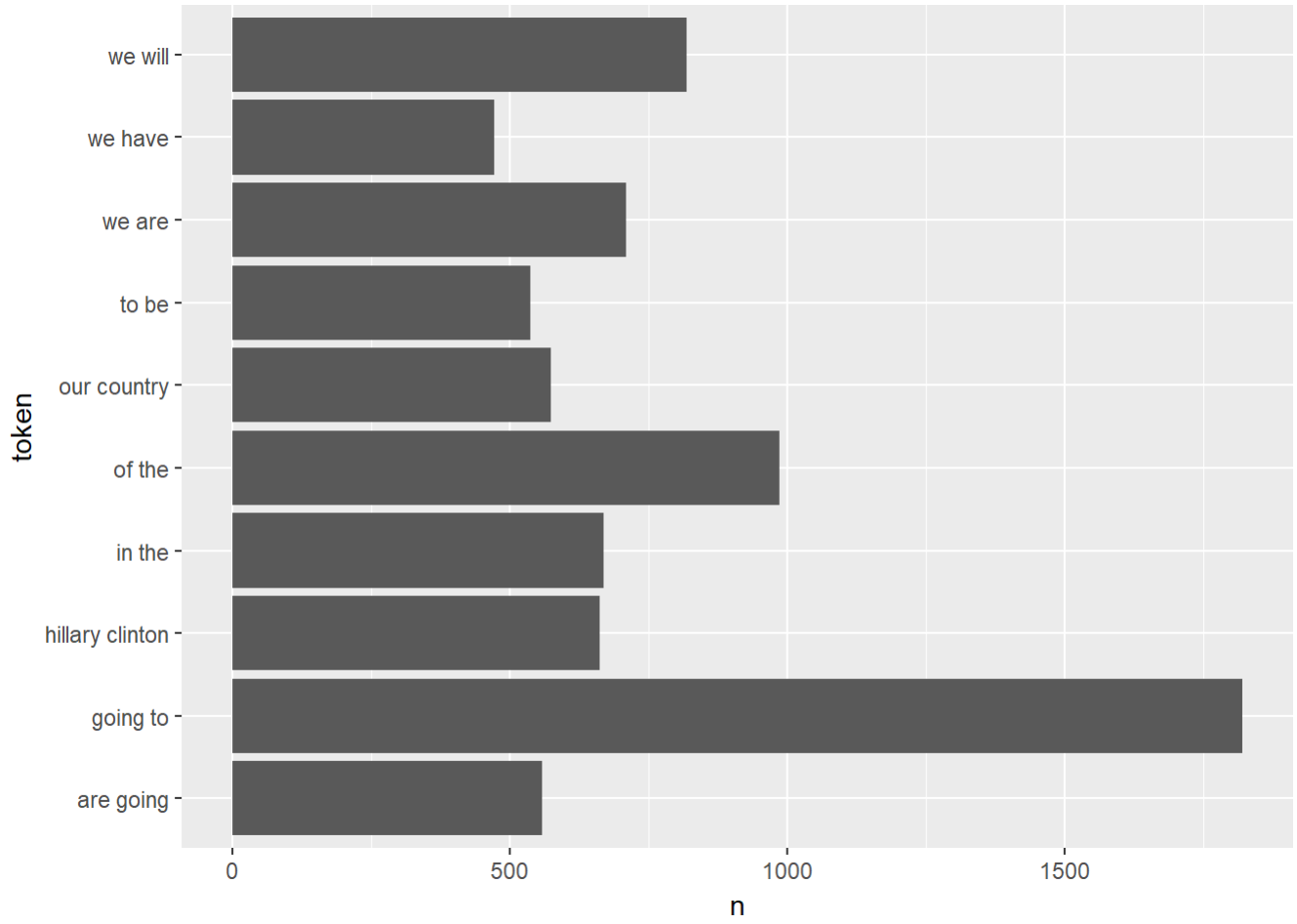

```
# Removal of stop words
plot.tidy_corpus(check_tidy_corpus, n=10, rm_stop_words=TRUE)
```

```
## Selecting by n
```



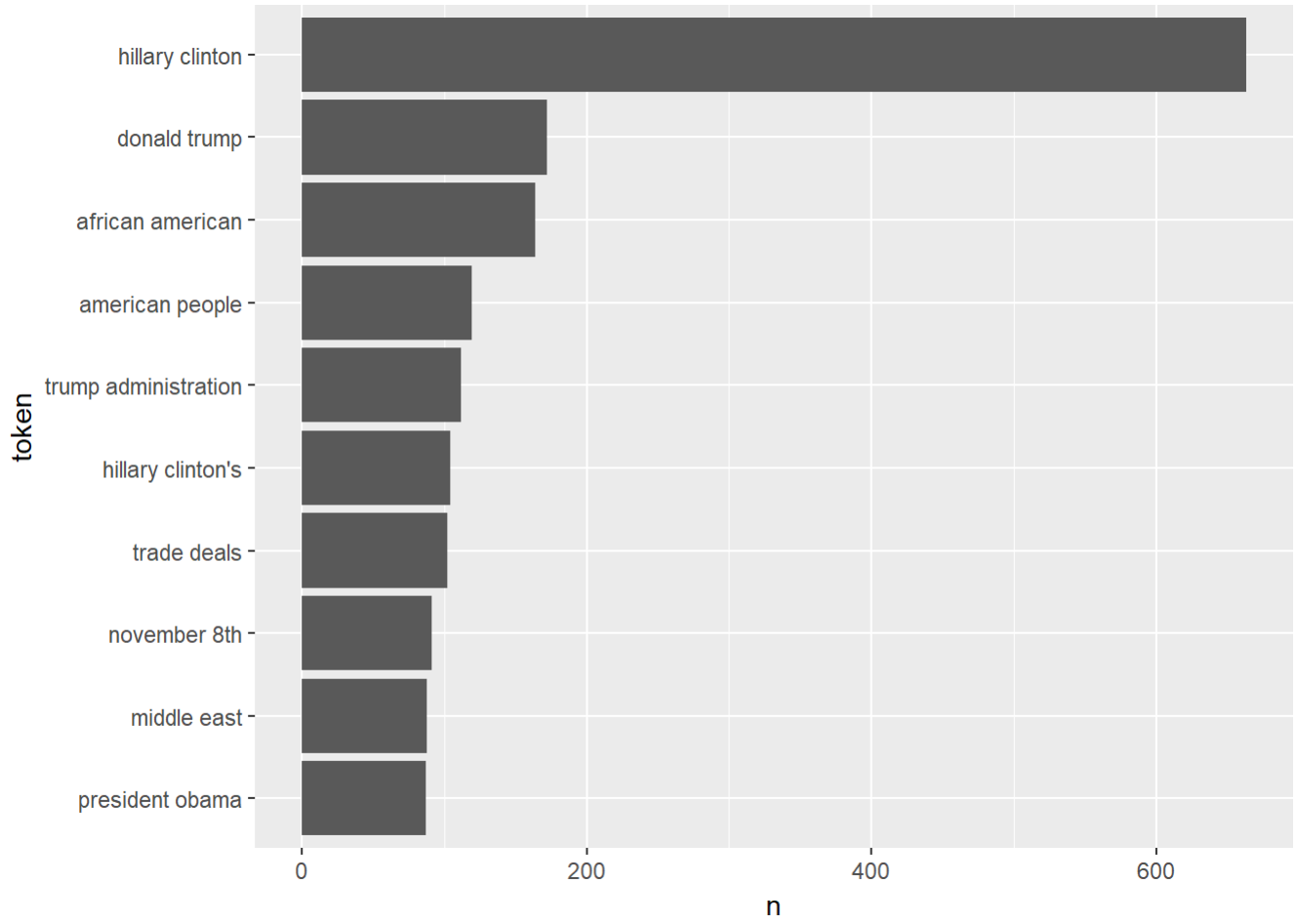

```
# (3) src as a "directory" and tokens as "bigrams"
check_tidy_corpus_bi <- tidy_corpus("C:/Users/mouni/Downloads/Sem 1/R Lang/Datasets/full_speech.
txt", token = "ngrams", n = 2)
#check_tidy_corpus_bi
# Non-removal of stop words
plot.tidy_corpus(check_tidy_corpus_bi, n=10, rm_stop_words=FALSE)
```

```
## Selecting by n
```




```
# Removal of stop words
plot.tidy_corpus(check_tidy_corpus_bi, n = 10, rm_stop_words = TRUE)
```

```
## Selecting by n
```

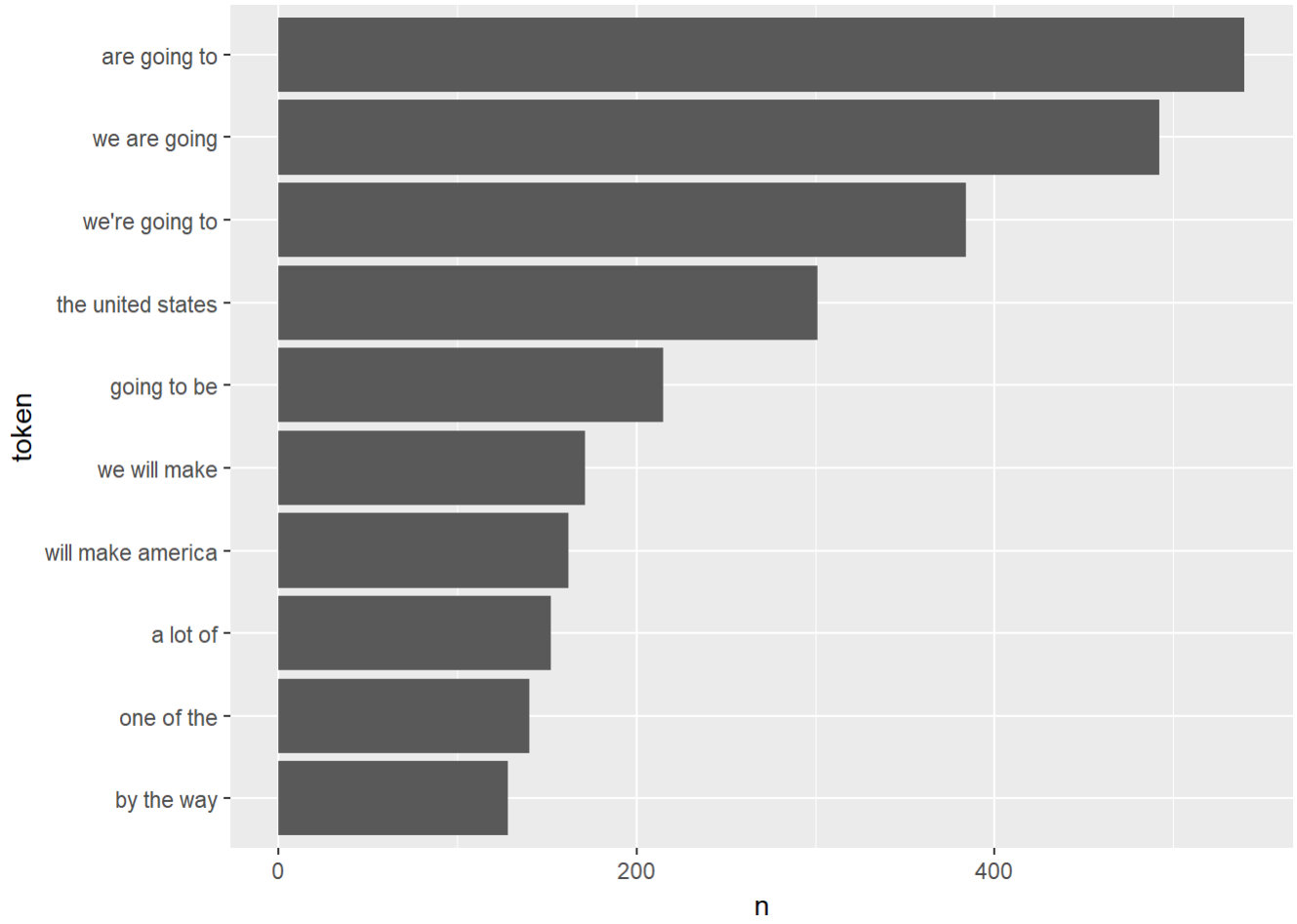



```
# (4) src as a "directory" and tokens as "trigrams"
check_tidy_corpus <- tidy_corpus("C:/Users/mouni/Downloads/Sem 1/R Lang/Datasets/full_speech.tx
t", token = "ngrams", n = 3)
check_tidy_corpus
```

```
## $corpus
## # A tibble: 235,089 x 2
##     line token
##  * <int> <chr>
##  1     1 trump wow whoa
##  2     1 wow whoa that
##  3     1 whoa that is
##  4     1 that is some
##  5     1 is some group
##  6     1 some group of
##  7     1 group of people
##  8     1 of people thousands
##  9     1 people thousands so
## 10     1 thousands so nice
## # ... with 235,079 more rows
##
## $token_type
## [1] "ngrams"
##
## $n
## [1] 3
##
## attr(,"class")
## [1] "tidy_corpus"
```

```
# Non-removal of stop words
plot.tidy_corpus(check_tidy_corpus, n=10, rm_stop_words=FALSE)
```

```
## Selecting by n
```



Problem 10

Create an R package for the class and methods you created in Problem 9. For full credit, it should pass R CMD check without errors. (Warnings are okay. You do not need to include documentation.) Build the package and include the .tar.gz compressed file in your homework directory.

```
# Creating an R package for the class and methods created in Problem 9

#package.skeleton(name = "tidycorpus", list=c("tidy_corpus", "plot", "plot.tidy_corpus"))

# commenting out the package creation code as the package has already been created and it might
 throw an error.

# install the created package
install("tidycorpus")
```

```
## Installing tidycorpus
```

```
## "C:/PROGRA~1/R/R-34~1.3/bin/x64/R" --no-site-file --no-environ --no-save --no-restore  \
##   --quiet CMD INSTALL "C:/Users/mouni/OneDrive/Documents/tidycorpus"  \
##   --library="C:/Users/mouni/OneDrive/Documents/R/win-library/3.4" --install-tests
```

```
##
```

```
# load the package
library("tidycorpus")
```

```
##
## Attaching package: 'tidycorpus'
```

```
## The following objects are masked _by_ '.GlobalEnv':
##
##     plot, plot.tidy_corpus, tidy_corpus
```

```
## The following object is masked from 'package:graphics':
##
##     plot
```

```
# R Check
check("tidycorpus")
```

```
## Updating tidycorpus documentation
```

```
## Loading tidycorpus
```

```
## First time using roxygen2. Upgrading automatically...
```

```
## Warning: The existing 'NAMESPACE' file was not generated by roxygen2, and will not be
## overwritten.
```

```
## Setting env vars ----------------------------------------------------------------
```

```
## CFLAGS  : -Wall -pedantic
## CXXFLAGS: -Wall -pedantic
```

```
## Building tidycorpus -------------------------------------------------------------
```

```
## "C:/PROGRA~1/R/R-34~1.3/bin/x64/R" --no-site-file --no-environ --no-save --no-restore  \
##   --quiet CMD build "C:\Users\mouni\OneDrive\Documents\tidycorpus" --no-resave-data  \
##   --no-manual
```

```
##
```

```
## Setting env vars ----------------------------------------------------------------
```

```
## _R_CHECK_CRAN_INCOMING_ : FALSE
## _R_CHECK_FORCE_SUGGESTS_: FALSE
```

```
## Checking tidycorpus ------------------------------------------------------------------
```

```
## "C:/PROGRA~1/R/R-34~1.3/bin/x64/R" --no-site-file --no-environ --no-save --no-restore  \
##   --quiet CMD check  \
##   "C:\Users\mouni\AppData\Local\Temp\Rtmpq69Yrm/tidycorpus_1.0.tar.gz" --as-cran  \
##   --timings --no-manual
```

```
##
```

```
## R CMD check results
```

```
## 0 errors | 2 warnings | 2 notes
```

```
## checking S3 generic/method consistency ... WARNING
## plot:
##   function(x)
## plot.tidy_corpus:
##   function(rand, n, rm_stop_words)
##
## See section 'Generic functions and methods' in the 'Writing R
## Extensions' manual.
##
## Found the following apparent S3 methods exported but not registered:
##   plot.tidy_corpus
## See section 'Registering S3 methods' in the 'Writing R Extensions'
## manual.
##
## checking for missing documentation entries ... WARNING
## Undocumented code objects:
##   'plot' 'plot.tidy_corpus' 'tidy_corpus'
## All user-level objects in a package should have documentation entries.
## See chapter 'Writing R documentation files' in the 'Writing R
## Extensions' manual.
##
## checking DESCRIPTION meta-information ... NOTE
## Malformed Description field: should contain one or more complete sentences.
## Non-standard license specification:
##   What license is it under?
## Standardizable: FALSE
##
## checking R code for possible problems ... NOTE
## plot.tidy_corpus: no visible global function definition for '%>%'
## plot.tidy_corpus: no visible global function definition for 'anti_join'
## plot.tidy_corpus: no visible binding for global variable 'stop_words'
## plot.tidy_corpus: no visible global function definition for 'count'
## plot.tidy_corpus: no visible binding for global variable 'token'
## plot.tidy_corpus: no visible global function definition for 'top_n'
## plot.tidy_corpus: no visible global function definition for 'mutate'
## plot.tidy_corpus: no visible global function definition for 'reorder'
## plot.tidy_corpus: no visible global function definition for 'ggplot'
## ... 10 lines ...
## tidy_corpus: no visible global function definition for 'read_lines'
## tidy_corpus: no visible global function definition for 'tibble'
## tidy_corpus: no visible global function definition for 'unnest_tokens'
## tidy_corpus: no visible binding for global variable 'words'
## Undefined global functions or variables:
##   %>% aes anti_join coord_flip count filter geom_col ggplot mutate
##   read_lines reorder separate stop_words tibble token top_n unite
##   unnest_tokens word1 word2 words
## Consider adding
##   importFrom("stats", "filter", "reorder")
## to your NAMESPACE file.
```

```
# compressing the file
library(utils)
tar("tidycorpus.tar.gz", "C:/Users/mouni/OneDrive/Documents/tidycorpus", compression = "gzip", t
ar = "tar")
```