STA141 Assignment 4

Zhen Zhang

November 15, 2015

Problem 1

First I will introduce some functions I use to help me extract the pattern by gregexpr, combine the regular expressions in multiple patterns, and give me the summary statistics of the matched values.

```
extract_from_data <- function(pattern, original_string = vposts$body) {
    # This function operates by first using gregexpr to extract
    # the pattern I desire, and then calling
    # extract_from_gregexpr to process the gregexpr result.

    grepexpr_index <- gregexpr(pattern, original_string, perl = T)
    # make it invisible to the console
    invisible(extract_from_gregexpr(grepexpr_index, original_string))
}

extract_from_gregexpr <- function(grepexpr_index, original_string = vposts$body) {
    # This function deals with the gregexpr result and extract
    # the string implied by the starting index and the
    # match.length attribute. I use the Map function to iterate
    # the indices and the original string simultaneously, and at
    # each row, extract the relevant information.

    Map(function(x, y) {
        # how many matches in total
        N <- length(x)
        result <- sapply(1:N, function(i) {
            # extract the string of each match
            result_each <- substr(y, x[i], x[i] - 1 + attr(x,
                "match.length")[i])
            # if none is matched, set '' to NA
            ifelse(result_each == "", NA, result_each)
        })
    }, grepexpr_index, original_string)
}

stat_for_total_num <- function(body) {
    # This function calculates the statistics to remind me how
    # many is matched.

    # extract the Non-NA number and print it out
    result <- sum(!is.na(body))
    cat("\n")
    cat(sprintf("The number of matching is %g", result))
    cat("\n")
    invisible(result)
}
```

```r
paste_re <- function(x, y) {
    # This function combines different patterns, which is the
    # main function renders to Reduce to perform concatenation.

    paste(x, y, sep = "|")
}
```

Before the formal regular expression matching, I would like to talk about how my matching process is organized. The first step is to combine different matching patterns by calling `paste_re` and `Reduce`. Then, I will use the `extract_from_data` function to extract pattern from string, usually `vposts$body` by default, but there are other cases. Since there may be multiple matching for each row, I will then select a single one by some algorithm. At last, I will show how many is matched using this pattern by `stat_for_total_num`

(a)

```r
# Here I observe two patterns, one is for example, $6000 or
# 6000$, another is $6,000 or 6,000$. The first pattern is
# very easy, and to capture the second pattern, I use
# (?:pattern) to group digits and the corresponding ','.
price_pattern <- c("\\$[^0-9]{0,4}(?:[0-9]+[,])*[0-9]*", "(?:[0-9]+[,])*[0-9]*\\$[^0-9]{0,4}")

# extract relevant pattern strings
price_from_body <- extract_from_data(Reduce(paste_re, price_pattern))

# The first step is to remove $ symbol, and the second step
# is to remove ',' symbol if possible, and the last step is
# for multiple prices, return the largest one.
price_from_body_single <- sapply(price_from_body, function(x) {
    N <- length(x)
    price_total <- sapply(1:N, function(i) {
        # remove $
        price <- gsub("[^0-9]*([0-9,]+).*", "\\1", x[i])
        # remove ,
        price <- gsub("[,]", "", price)
        # sometimes the price and the year is concatenated together,
        # so I remove them
        price <- ifelse(nchar(price) > 8, substr(price, 1, nchar(price) -
            4), price)
    })
    # return the max price
    max(as.numeric(price_total))
})

# show how many is matched successfully.
stat_for_total_num(price_from_body_single)
```

```
##
## The number of matching is 14931
```

```r
# show how many is consistent with the price column
sum(price_from_body_single == vposts$price, na.rm = T)
```

```
## [1] 10412
```

```r
# show some results I get
head(price_from_body_single, n = 10)
```

```
##  [1] 29896 18797 15792 18288 26389 28996    NA 24995 15995
## [10]    NA
```

(b)

```r
# According to the VIN standard, VIN has 17 characters, where
# there is no latter I, O, Q.  In addition, the last six
# letters must be digits, so my regular expression is written
# as follows:
VIN_pattern <- c("(?![0-9]{17})[0-9A-HJ-NPR-Z]{11}[0-9]{6}")

# There is only one pattern, so I will not use Reduce
# function.
VIN_from_body <- extract_from_data(VIN_pattern)

# Here I will select the VIN if there are multiple results,
# since the VIN usually comes at first position of the body
# text.
VIN_from_body_single <- sapply(VIN_from_body, `[`, 1)

# show how many is matched successfully.
stat_for_total_num(VIN_from_body_single)
```

```
##
## The number of matching is 7581
```

```r
# add to vposts
vposts$VIN <- VIN_from_body_single

# show some results I get
head(VIN_from_body_single, n = 10)
```

```
##  [1] "2G1FT1EW1C9106920" "2GNFLNEK7D6324351"
##  [3] "1N4AL3AP5DN569028" "JNKCY01F29M851648"
##  [5] "JN1CV6AR2DM763007" "2HNYD2H20CH537485"
##  [7] "JTDBT4K31A1368794" "5J8TB1H28CA000511"
##  [9] "1G1PC5SB1E7464908" "JTHCK262695031859"
```

(c)

```r
# At first it may seem very easy to extract the pattern of
# price. But in the last line, there is '(916) 715-31 SEVEN
# ZERO'. It will not the extracted by the normal expression
# only focusing on the digits. So at first I write a funciton
# to transform natural language to digits.

# This is the vector that combines the natural language
# digits
nums <- c("zero", "one", "two", "three", "four", "five", "six",
    "seven", "eight", "nine")
num_to_digit <- function(text) {
    # This is the function that transform natural language to
    # digits. Note here my pattern is paste0('[ -]?', (nums[i]),
    # '[ -]?'), since when there is natural languages, people
    # tend to separate them by space or hyphens. For example,
    # (916) 715-31 SEVEN ZERO. This pattern will remove the
    # unnecessary spaces or hyphens after transforming to digits.

    for (i in seq_along(nums)) {
        text <- gsub(paste0("[ -]?", (nums[i]), "[ -]?"), i -
            1, text, ignore.case = T)
    }

    # return value
    text
}

# call the function above, to return the new texts for body
# particular for phones.
body_modified_for_phone <- sapply(vposts$body, num_to_digit)

# Now the string '(916) 715-31 SEVEN ZERO' can be transformed
# to:
num_to_digit("(916) 715-31 SEVEN ZERO")
```

```
## [1] "(916) 715-3170"
```

```r
# Ok, it is fine now.

# The pattern for phone number there are three parts for a
# phone number, each part inside cannot be separated, and
# between each part, there can be at most one space [ ] or
# hyphen [-]. Also, the first part can be surrounded by
# parenthesis ().
phone_pattern <- c("\\(?[0-9]{3}\\)?[. -]*[0-9]{3}[. -]*[0-9]{4}")

# extrac the phone numbers from body_modified_for_phone
phone_from_body <- extract_from_data(phone_pattern, body_modified_for_phone)
# get the first phone number if there is multiple
phone_from_body_single <- sapply(phone_from_body, `[`, 1)

# show how many is matched successfully
stat_for_total_num(phone_from_body_single)
```

```
##
## The number of matching is 17445
```

```r
# add to vposts
vposts$phone <- phone_from_body_single

# show some results I get
head(phone_from_body_single, n = 10)
```

```
##  [1] "(508) 205-1046" "(508) 205-1046" "(508) 205-1046"
##  [4] "(508) 205-1046" "(508) 205-1046" "(508) 205-1046"
##  [7] "(508) 213-4680" "(508) 205-1046" "(508) 205-1046"
## [10] "(508) 213-4680"
```

(d)

```r
# The email address always have @ symbol and
# xxx.xxx.com/net/org pattern. The thing should be kept in
# mind is the subdomain, so I use (?:) to group characters
# and dot, and let them emerge together. Also, keep
# com|org|net from emerging in the subdomain, since they
# should only come into being in top domain name.
email_pattern <- c("[[:alnum:]]+@(?:(?!com|org|net)[[:alnum:]]+[.])+(?:com|org|net)")

email_from_body <- extract_from_data(email_pattern)

# Here I will select the email if there are multiple results,
# since most of them are identical.
email_from_body_single <- sapply(email_from_body, `[`, 1)

# show how many is matched successfully
stat_for_total_num(email_from_body_single)
```

```
##
## The number of matching is 105
```

```r
# add to vposts
vposts$email <- email_from_body_single

# show some results I get Here I will display a subdomain
# email address, 'Leads@Chicagomotorcars.motosnap.com'
email_from_body_single[which(!is.na(email_from_body_single))[36:45]]
```

```
##  [1] "valueautomartinc@prodigy.net"
##  [2] "valueautomartinc@prodigy.net"
##  [3] "valueautomartinc@prodigy.net"
##  [4] "sales@chicagoautoplace.com"
##  [5] "gmmantrb@yahoo.com"
##  [6] "Leads@Chicagomotorcars.motosnap.com"
##  [7] "Leads@Chicagomotorcars.motosnap.com"
##  [8] "Leads@Chicagomotorcars.motosnap.com"
##  [9] "Leads@Chicagomotorcars.motosnap.com"
## [10] "BettyDalke263@gmail.com"
```

(e)

```r
# year pattern, including 19xx, 200x, 201x, 9x, 0x
year_pattern <- c("19[5-9][0-9]", "20[0-1][0-9]", "[09][0-9]")

# extract year by year_pattern
year_from_description <- extract_from_data(Reduce(paste_re, year_pattern),
    vposts$description)

# If there are multiple results, select the first one. Also,
# if there is 9x, 0x pattern, convert them to 199x, 200x.
year_from_description_single <- sapply(year_from_description,
    function(x) {
        result <- x[1]
        if (nchar(result) == 2 && !is.na(result)) {
            x_first_digit <- substr(result, 1, 1)
            if (x_first_digit == "0") {
                result <- paste0("20", result)
            } else {
                result <- paste0("19", result)
            }
        }
        result
    })

# show how many is matched successfully.
stat_for_total_num(year_from_description_single)
```

```
##
## The number of matching is 31576
```

```r
# how many is wrong
sum(vposts$year != year_from_description_single, na.rm = T)
```

```
## [1] 1342
```

```r
# show some results I get
head(year_from_description_single, n = 10)
```

```
##  [1] "2012" "2013" "2013" "2009" "2013" "2012" "2010" "2012"
##  [9] "2014" "2009"
```

(f)

```r
# Here I extract model name from two string texts, one is
# vposts$header, another is vposts$description. Most of the
# model names come at the second word after time. Sometimes
# there is no such string in vposts$header, so I will try to
# serach it in vposts$description.
```

```r
time_identifier_model <- function(x) {
    # This is the function that implements the first algorithm,
    # time identifier. Using the splitted result, it will first
    # match the time pattern, after that, get the model index,
    # then get the model

    # match the time pattern
    x_time_index <- which(sapply("[0-9]{4}", grepl, x))
    # get the model index
    x_model_index <- x_time_index + 2
    # get the model
    x_model <- x[x_model_index]

    # set unmatch to NA
    if (length(x_model) == 0) {
        x_model <- NA
    }

    # return the result
    x_model
}

vposts_split <- function(string) {
    # This function split the string by identifier not digits,
    # alphabets or hyphens. Most of the time, the split one will
    # be space, and there are some situations that , will be the
    # separate tag. Here I preserve -, since there are many
    # vehicles that has the model name of '3-Series'

    string_elements <- unlist(strsplit(string, "[^0-9a-zA-Z-]",
        perl = T))
    string_elements <- string_elements[string_elements != ""]
}

model_extract <- function(string) {
    # This function first does some preparation, namely the split
    # job, then render it to time identifier and maker
    # identifier. It connects different small jobs into a whole
    # process.

    unlist(lapply(string, function(x) {
        # split the string
        x_elements <- vposts_split(x)

        # the first algorithm, using the time identifier
        x_model <- time_identifier_model(x_elements)[1]
    }))
}

model_extract_total <- function() {
    # This function calculates the model extracted from header
    # and description, and then will use the result from header
    # by default, and only use the result from description if the
```

```r
    # result from header is NA.

    model_from_header <- model_extract(vposts$header)
    model_from_description <- model_extract(vposts$description)

    Map(function(x, y) {
        ifelse(!is.na(x), x, y)
    }, model_from_header, model_from_description)
}

model_from_header_description <- unlist(model_extract_total())

# convert to lower case, being case insensitive
model_from_header_description <- tolower(model_from_header_description)

# show some results I get
head(model_from_header_description, n = 10)
```

```
##    Camaro   Equinox    Altima      M35x      G37x       MDX
##  "camaro" "equinox"  "altima"    "m35x"    "g37x"     "mdx"
##     Yaris       RDX     Cruze        IS
##   "yaris"     "rdx"   "cruze"      "is"
```

```r
# *Correct the typos in model name*

# The algorithm is a little complicated, so I will explain
# the basic idea. First, I will create a dataframe of maker
# to model, and then classify the data based the maker. After
# that, for each maker, I will split the model by its
# frequency. If the frequency is higher than a threshold
# (which is a parameter, confidence_threshold), then it will
# be normal names, otherwise, I think it is mistyped. The
# mistyped names will then calculate the distance with the
# normal names, see if it is close enough to anyone of the
# normal names. Note here that the inclusion of partial = T
# when calculating adist will give us also the result of
# abbreviations. The distance threshold is 1, so if it is
# small than 1, the mistyped names will be substitute by the
# normal name, and if there is a tie, it will be substituted
# by the most frequent normal name. Also, if none of the
# distance is smaller than 1, then I also think it is not
# mistyped. At last, for each mistyped names, I will create a
# rule controlling how it should be modified, which is the
# result of this function.

x_lower_names_transform <- function(x_lower_names, x_higher_names,
    x_adist, x_table) {
    # This function change the mistyped names to normal names, by
    # the adist matrix.  The distance threshold is 1, so the
    # distance is small than 1, the mistyped names will be
    # substitute by the normal name, and if there is a tie, it
    # will be substituted by the most frequent normal name. Note
    # if distance is 0, if means partial matching is successful,
```

```r
    # which is for instance, 3 in 3-series. Also, if none of the
    # distance is smaller than 1, then I also think it is not
    # mistyped.

    sapply(1:length(x_lower_names), function(i) {
        # initialized the result
        result <- x_lower_names[i]
        # calculate the nearest distance
        name_adist <- x_adist[i, ]
        name_adist_min <- min(name_adist)
        # if the distance is one, set the name to it
        if (name_adist_min <= 1) {
            name_adist_min_index <- which(name_adist == name_adist_min)
            name_adist_min_index_names <- x_higher_names[name_adist_min_index]
            if (length(name_adist_min_index_names) == 1) {
                result <- name_adist_min_index_names
            } else {
                # in case of tie, select the one with the most frequency
                compare_table <- x_table[name_adist_min_index_names]
                result <- names(which.max(compare_table))
            }
        }
        # return value
        result
    })
}

x_change_chain_rule <- function(x_lower_names, x_lower_names_transformed) {
    # create the change rule, and only return the one that is
    # really changed under my algorithm, others will leave them
    # unchanged, meaning the rule is NULL.  Here I find the Map
    # function is most convenient.

    Map(function(x, y) {
        if (x != y) {
            y
        }
    }, x_lower_names, x_lower_names_transformed)
}

modify_typo_model_maker <- function(x, confidence_threshold) {
    # This function operates on the abstract level of each maker,
    # calculateing each level's lowest names, for each maker, and
    # also convert the corresponding level to the nearest model.
    # It first does some preparation, and then call
    # x_lower_names_transform function to return the modify model
    # values, and at last, call the x_change_chain_rule function
    # to create the change chain.

    # Some maker only has one model, so at that case there is
    # nothing to do
    if (length(x) > 1) {
```

```r
        # calculate lowest names, higher names, table for future
        # calculation
        x_table <- table(x)
        x_table_names <- names(x_table)
        x_lower_names <- x_table_names[x_table <= confidence_threshold]
        x_higher_names <- x_table_names[x_table > confidence_threshold]
        # calculate the adist matrix It should be noted here that,
        # using the option partial = T will also caputre abbreviation
        # cases.
        x_adist <- adist(x_lower_names, x_higher_names, partial = T)

        # I must ensure there is both the mistyped models and normal
        # models
        if (length(x_adist) != 0) {

            # modify the model name, if the adist value is 1. if tie,
            # choose the one with the most frequency. If no adist value
            # is 1 related to that name, keep it unchanged
            x_lower_names_transformed <- x_lower_names_transform(x_lower_names,
                x_higher_names, x_adist, x_table)

            # create the change rule
            x_change_chain <- x_change_chain_rule(x_lower_names,
                x_lower_names_transformed)

            # return the change rule
            x_change_chain
        }
    }
}

modify_typo_model <- function(model_from_header, confidence_threshold) {
    # This function is the main function to implement the correct
    # typos. It first calculates the dataframe and the splitted
    # dataframe by maker, then supplies these results to
    # modify_typo_model_maker function. It connects different
    # parts of the correcting job, and return the value I really
    # want.

    # preparation

    # combine the data into a daraframe
    maker_model_data <- data.frame(maker = vposts$maker, model = model_from_header,
        stringsAsFactors = F)
    # split by the maker
    maker_model_data_split <- split(maker_model_data$model, maker_model_data$maker)

    # Now comes to the main part. It operates on the abstract
    # level of each maker.  It calculate each level's lowest
    # names, for each maker, and also convert to the nearest
    # model
    lapply(maker_model_data_split, modify_typo_model_maker, confidence_threshold)
}
```

```r
# call the modify function to modify the model name
model_modify_rule <- modify_typo_model(model_from_header_description,
    3)

# show the rule, with maker toyota, here if the value under a
# model is null, there is no modification to it, if there is
# some value under a model, this is the one should be
# modified to. Here I will only give the models that need to
# be corrected under my algorithm for maker toyota.
unlist(model_modify_rule["toyota"])
```

```
##         toyota.-   toyota.4runeer         toyota.a
##          "rav-4"        "4runner"          "camry"
##       toyota.awd       toyota.cab    toyota.camary
##      "highlander"        "camry"          "camry"
##    toyota.camery    toyota.camryu       toyota.car
##          "camry"          "camry"          "camry"
##   toyota.carolla    toyota.corola    toyota.coroll
##        "corolla"        "corolla"        "corolla"
##  toyota.corrolla      toyota.fj60      toyota.ghia
##        "corolla"           "fj40"     "highlander"
## toyota.highlader         toyota.l    toyota.metrix
##     "highlander"        "corolla"         "matrix"
##        toyota.mr   toyota.pick-up        toyota.se
##          "camry"         "pickup"        "sequoia"
##     toyota.siena    toyota.solora        toyota.tc
##         "sienna"         "solara"          "camry"
##    toyota.tindra        toyota.up       toyota.van
##         "tundra"         "pickup"     "highlander"
##      toyota.yais
##          "yaris"
```

```r
model_final <- unlist(lapply(1:length(vposts$header), function(i) {
    # present the values of maker, model, and modified model
    maker <- vposts$maker[i]
    model <- model_from_header_description[i]
    modified_model <- model_modify_rule[[maker]][[model]]
    # if it is not null, then there should be a rule to change it
    if (!is.null(modified_model))
        model <- modified_model
    # return result
    model
}))

# add the model to the vposts dataframe
vposts$model <- model_final

# show some results
head(model_final, n = 10)
```

```
##    Camaro   Equinox    Altima      M35x      G37x       MDX
##  "camaro" "equinox"  "altima"    "m35x"    "g37x"     "mdx"
```

```
##     Yaris       RDX     Cruze        IS
##   "yaris"     "rdx"   "cruze"      "is"
```

Question two

```
# First I sort the table, and see which model is the most:
invisible(sort(table(model_final)))

# For the top five models, I want to see which maker they
# belong to:
top_five_models_names <- names(table(model_final))[order(table(model_final),
    decreasing = T)[1:5]]
# check if the model belong to the same maker
sapply(top_five_models_names, function(x) {
    x_index <- which(model_final == x)
    unique(vposts$maker[x_index])
})
```

```
## $civic
## [1] "honda"
##
## $accord
## [1] "honda"
##
## $grand
##  [1] "jeep"     "pontiac"  "dodge"    "mercury"  "buick"
##  [6] NA         "suzuki"   "plymouth" "chrysler" "nissan"
##
## $camry
## [1] "toyota"
##
## $altima
## [1] "nissan"
```

```
# I will pick the top two combination honda-civic,
# toyota-camry.

# select civic indices and camry indices:
select_indices <- function(model) {
    # this is the function that can return the indices related to
    # this model
    name_index <- which(model_final == model)
}

get_model_data <- function(model, intereste_cols_flag = TRUE) {
    # this funciton first calls select_indices, then use the
    # indices to select rows from vposts

    indices <- select_indices(model)
    # get the dataframe for price, age, odometer and condition
    model_data <- vposts[indices, ]
    model_data$age <- 2015 - model_data$year
```

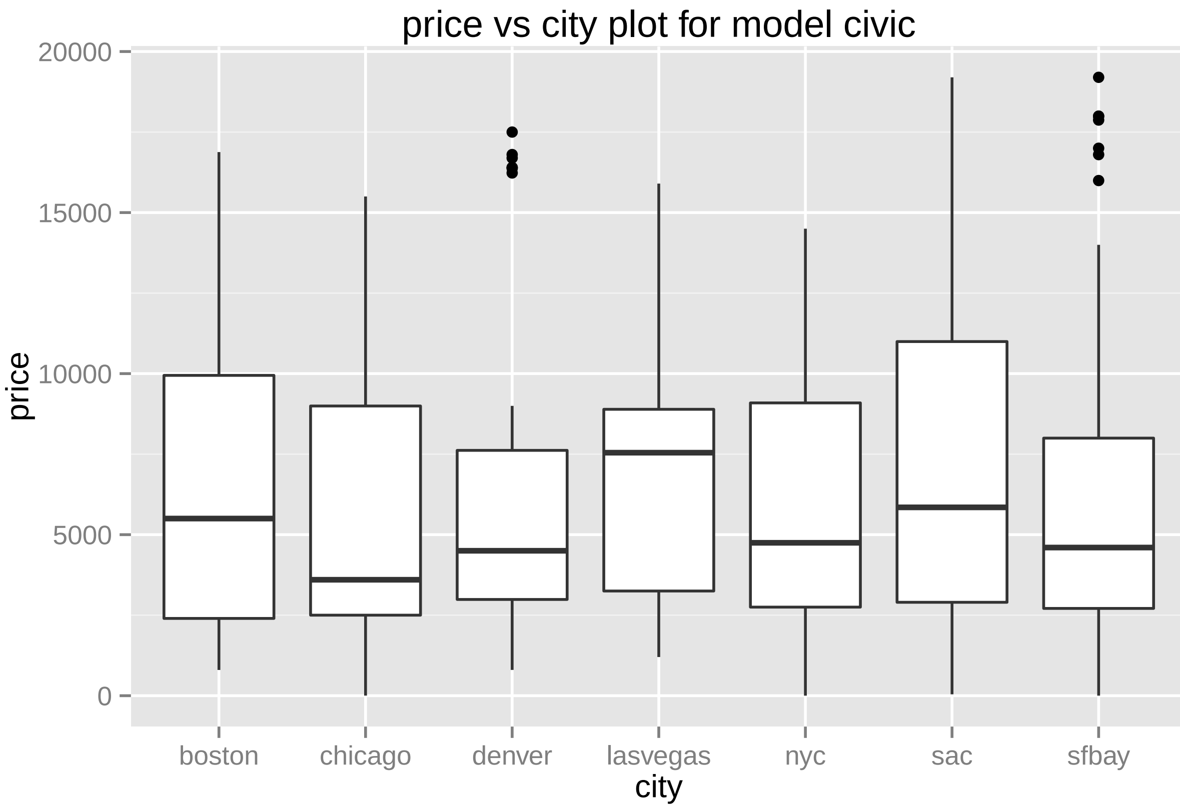
```r
    if (intereste_cols_flag) {
        model_data <- model_data[c("price", "odometer", "condition",
            "age")]
    } else {
        model_data <- model_data[c("price", "odometer", "condition",
            "age", "city")]
    }

    # get complete data without NA
    model_data <- model_data[complete.cases(model_data), ]

    # since there is a factor condition, if the frequency is so
    # low, it will not perform cross-validation. So I remove such
    # levels, with frequency less than 5.
    model_data$condition <- factor(model_data$condition)
    model_data_cond_table <- table(model_data$condition)
    model_data_cond_names_high <- names(model_data_cond_table)[model_data_cond_table >
        5]
    model_data <- subset(model_data, condition %in% model_data_cond_names_high)

    # return
    model_data
}
```

```r
# My Approach

# Before All, I will first draw a map that prints different
# values of price, odometer, age and condition.

# The basic idea of my implementation in this question is,
# first I will choose which model to select. Since there are
# multiple models, I will only try some of them, including
# linear regression (lm), knn, regression tree (tree). The
# selection method is to use the package of caret to do cross
# validation on each of them, and see which one has the
# lowest RMSE. After we have already got one model in hand,
# we still need to select the parameters. For example, knn
# has the k parameter indicating how many nearest points
# should be used to predict the point we are interested. So I
# will again use cross validation to select the parameter,
# and at last, get the final model.

# Note that the assumpmtions here:

# lm: The error terms are normally distributed around 0 with
# the same standard deviation, identically independent
# distributed (i.i.d.).

# knn and regression tree: They are non-parametric methods,
# so there is no assumption about them.

# load caret package
library("caret")
```

```r
# Part I, Model of civic

# get the data
civic_data_total <- get_model_data("civic", FALSE)
civic_data <- get_model_data("civic")

# show the data
head(civic_data, n = 10)
```

```
##            price odometer condition age
## posted138  3500    87100      good  13
## posted210 12888    31470  like new   3
## posted284 12995    92730 excellent   4
## posted339  2950   100000       new  11
## posted455 13400    38000  like new   3
## posted648 12995   580000  like new   3
## posted649  1900   155112      good  14
## posted685  7500   132639 excellent   6
## posted726 12000    34822  like new   3
## posted834 16879    33740 excellent   3
```

```r
# Step 0, draw the boxplot of city vs price.

library(ggplot2)
ggplot(civic_data_total, aes(city, price)) + geom_boxplot() +
    ggtitle("price vs city plot for model civic")
```




```r
# The result indicates that, city has an effect on the price
# of vehicles. While vehicles in lasvegas and sac have high
# price, the vehicles in boston, chicago, denver, nyc and
# sfbay have low price.

# Step I, find which model is the best

# Before I really do the regression, I will first set the
# cross validation parameter.  The parameters I choose is
# 5-fold cross validation, repeating 3 times.
set.seed(1234)
fitControl <- trainControl(method = "cv", number = 5, repeats = 3)

# The lm approach
civic_fit_lm <- train(price ~ ., data = civic_data, method = "lm",
    trControl = fitControl)
civic_fit_lm
```

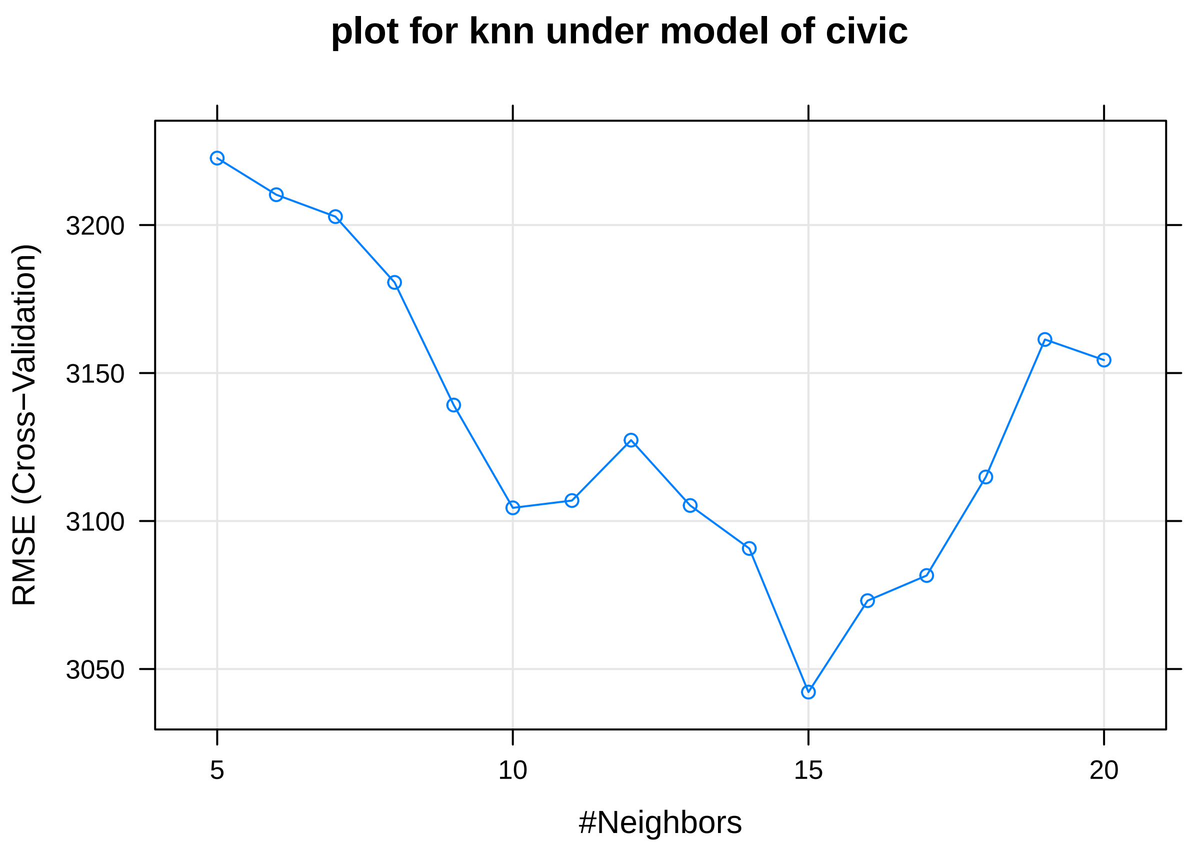
```
## Linear Regression
##
## 385 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 308, 309, 308, 307, 308
```

```
## Resampling results
##
##   RMSE      Rsquared   RMSE SD   Rsquared SD
##   2427.636  0.7465404  406.2788  0.06111017
##
##
```

```r
# It is easily seen that the lm fit result is RMSE: 2427.636

# The knn approach
civic_fit_knn <- train(price ~ ., data = civic_data, method = "knn",
    trControl = fitControl, tuneGrid = expand.grid(k = 5:20))
civic_fit_knn
```

```
## k-Nearest Neighbors
##
## 385 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 307, 310, 309, 307, 307
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   RMSE SD   Rsquared SD
##    5  3222.605  0.5547955  373.2698  0.1025050
##    6  3210.233  0.5527089  369.5447  0.1040402
##    7  3202.838  0.5527771  366.2770  0.1068240
##    8  3180.617  0.5563757  359.3339  0.1072440
##    9  3139.178  0.5678518  370.7077  0.1121173
##   10  3104.462  0.5755973  362.9433  0.1068860
##   11  3106.914  0.5731314  402.4827  0.1150875
##   12  3127.330  0.5673911  384.7854  0.1118827
##   13  3105.254  0.5728454  392.3084  0.1123711
##   14  3090.732  0.5760389  385.9250  0.1091862
##   15  3042.217  0.5893575  423.3836  0.1100051
##   16  3073.097  0.5810955  415.3030  0.1124095
##   17  3081.598  0.5792203  434.1722  0.1180324
##   18  3114.876  0.5701059  415.3934  0.1173847
##   19  3161.333  0.5573356  474.2057  0.1304835
##   20  3154.375  0.5596132  489.0885  0.1332095
##
## RMSE was used to select the optimal model using
##  the smallest value.
## The final value used for the model was k = 15.
```

```r
plot(civic_fit_knn, main = "plot for knn under model of civic")
```

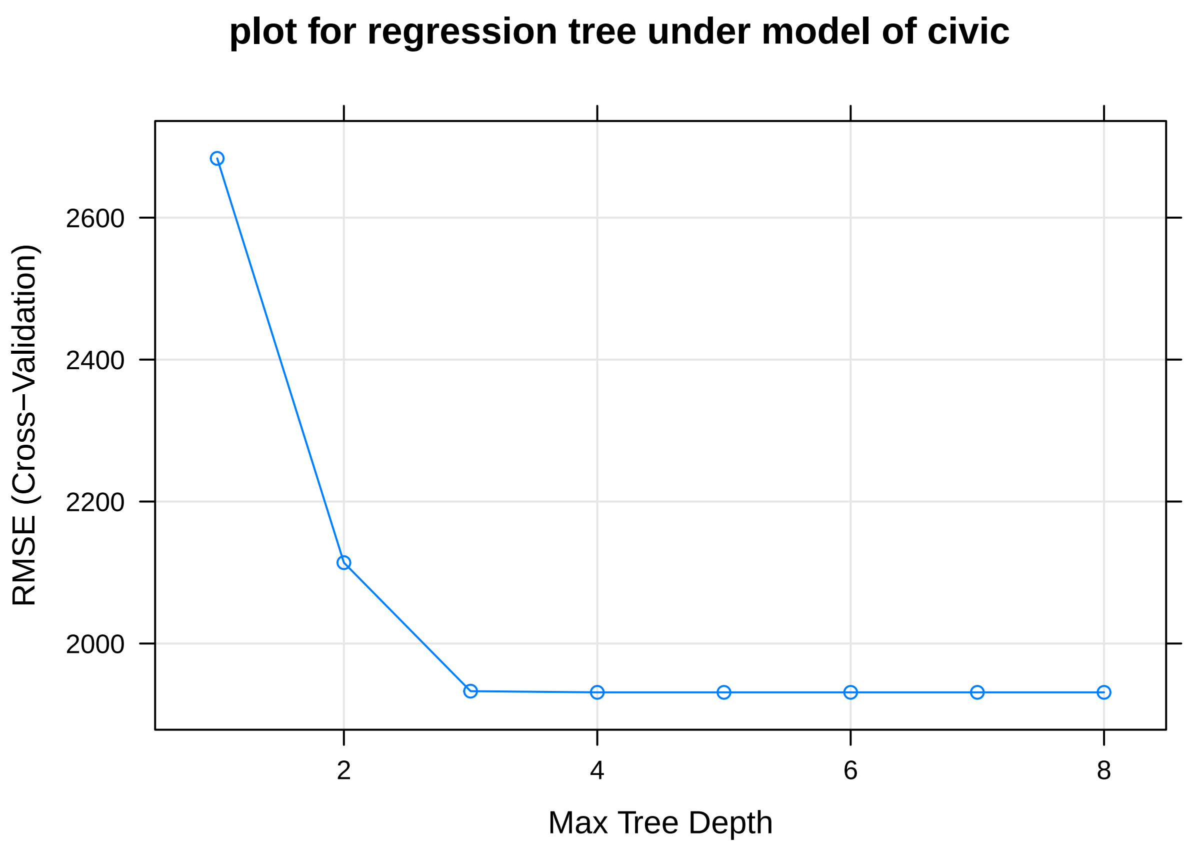



```r
# The best model for knn is k = 15, with RMSE: 3042.217

# The regression tree (tree) approach

civic_fit_tree <- train(price ~ ., data = civic_data, method = "rpart2",
    trControl = fitControl, tuneGrid = expand.grid(maxdepth = 1:8))
civic_fit_tree
```

```
## CART
##
## 385 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 307, 308, 308, 307, 310
## Resampling results across tuning parameters:
##
##   maxdepth  RMSE      Rsquared   RMSE SD   Rsquared SD
##   1         2683.479  0.6810051  267.3612  0.03991186
##   2         2113.911  0.8131563  177.6763  0.03523489
##   3         1932.837  0.8459422  179.5960  0.03710676
##   4         1931.201  0.8454661  156.3828  0.03576745
##   5         1931.201  0.8454661  156.3828  0.03576745
##   6         1931.201  0.8454661  156.3828  0.03576745
##   7         1931.201  0.8454661  156.3828  0.03576745
```

```
##   8         1931.201  0.8454661  156.3828  0.03576745
##
## RMSE was used to select the optimal model using
##  the smallest value.
## The final value used for the model was maxdepth = 4.
```

```r
plot(civic_fit_tree, main = "plot for regression tree under model of civic")
```

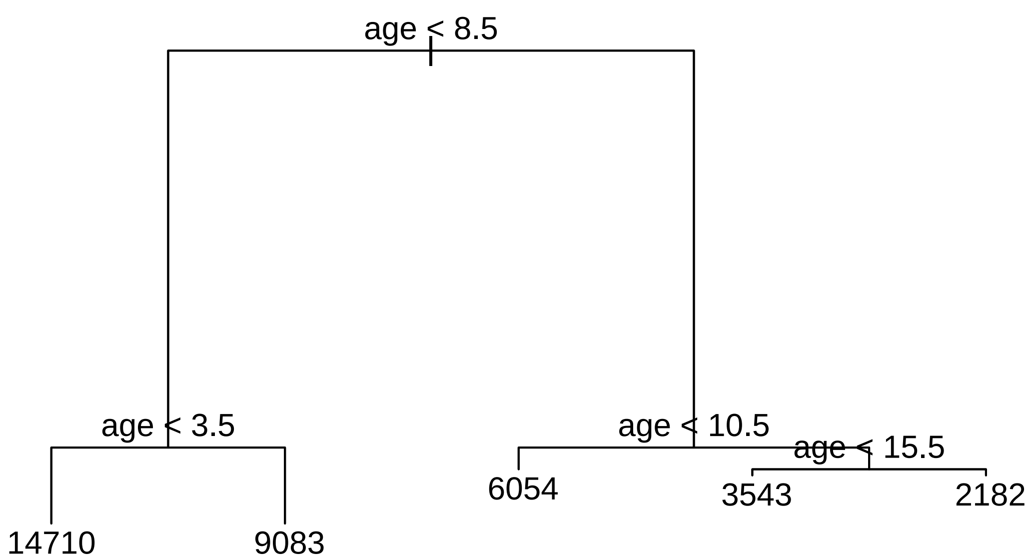



```r
# The best model for regression tree is k = 4, with RMSE:
# 1931.201

# Now we have the best model, which is regression tree, so I
# will dig deeply in regression tree

# Step II, find the best regression tree model

library(tree)

# regression tree details
civic_fit_tree <- tree(price ~ ., data = civic_data)
civic_fit_tree
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
```

```
##  1) root 385 8.602e+09  6502
##    2) age < 8.5 143 1.942e+09 11560
##      4) age < 3.5 63 3.191e+08 14710 *
##      5) age > 3.5 80 5.086e+08  9083 *
##    3) age > 8.5 242 8.386e+08  3513
##      6) age < 10.5 41 8.582e+07  6054 *
##      7) age > 10.5 201 4.342e+08  2995
##       14) age < 15.5 120 2.531e+08  3543 *
##       15) age > 15.5 81 9.158e+07  2182 *
```

```r
# plot the result
plot(civic_fit_tree)
text(civic_fit_tree, pretty = 0)
```

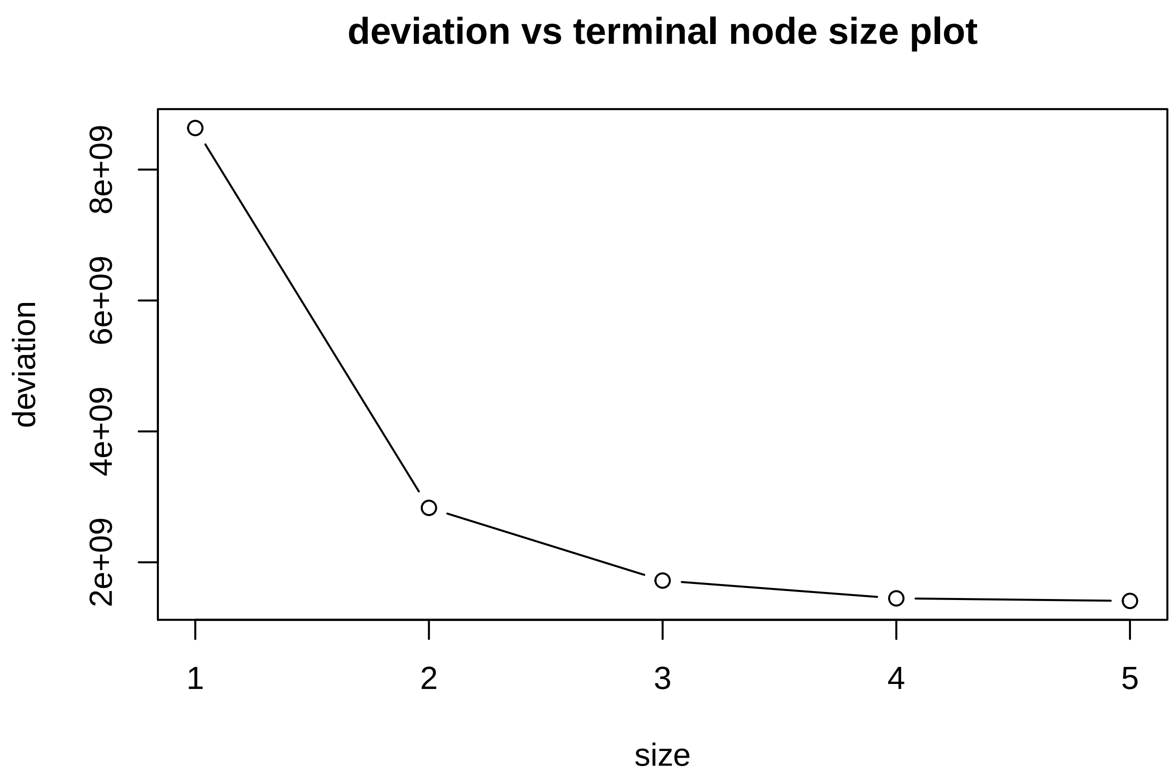



```r
# use cross validation to confirm the best tree depth
civic_fit_tree_cv <- cv.tree(civic_fit_tree)
civic_fit_tree_cv
```

```
## $size
## [1] 5 4 3 2 1
##
## $dev
## [1] 1409981525 1449142755 1720748313 2831907088 8634962622
##
## $k
```

```
## [1]       -Inf   89553964  318622673 1114362994 5821484477
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
# plot the result, the relationship between nodes and
# deviation
plot(civic_fit_tree_cv$size, civic_fit_tree_cv$dev, type = "b",
    xlab = "size", ylab = "deviation", main = "deviation vs terminal node size plot")
```

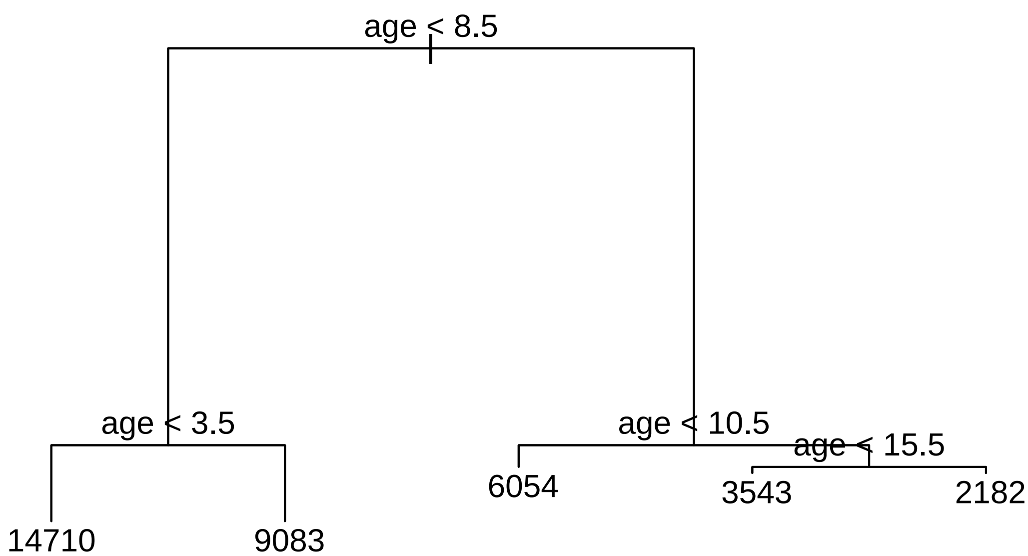



```r
# It seems size 5 is the best option
civic_fit_tree_prune <- prune.tree(civic_fit_tree, best = 5)
civic_fit_tree_prune
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 385 8.602e+09  6502
##    2) age < 8.5 143 1.942e+09 11560
##      4) age < 3.5 63 3.191e+08 14710 *
##      5) age > 3.5 80 5.086e+08  9083 *
##    3) age > 8.5 242 8.386e+08  3513
```

```
##      6) age < 10.5 41 8.582e+07  6054 *
##      7) age > 10.5 201 4.342e+08  2995
##       14) age < 15.5 120 2.531e+08  3543 *
##       15) age > 15.5 81 9.158e+07  2182 *
```

```r
# plot the best result
plot(civic_fit_tree_prune)
text(civic_fit_tree, pretty = 0)
```




```r
# Now let's do a prediction, the data I use is the first
# observation:
civic_data[1, ]
```

```
##           price odometer condition age
## posted138  3500    87100      good  13
```

```r
# let's use tree model to predict
civic_prediction_one <- predict(civic_fit_tree_prune, civic_data[1,
    ])
civic_prediction_one
```

```
## posted138
##    3543.1
```

```r
# It is very close!

# Step III, Analyze the result

# Let's look back the regression tree here:
civic_fit_tree_prune
```

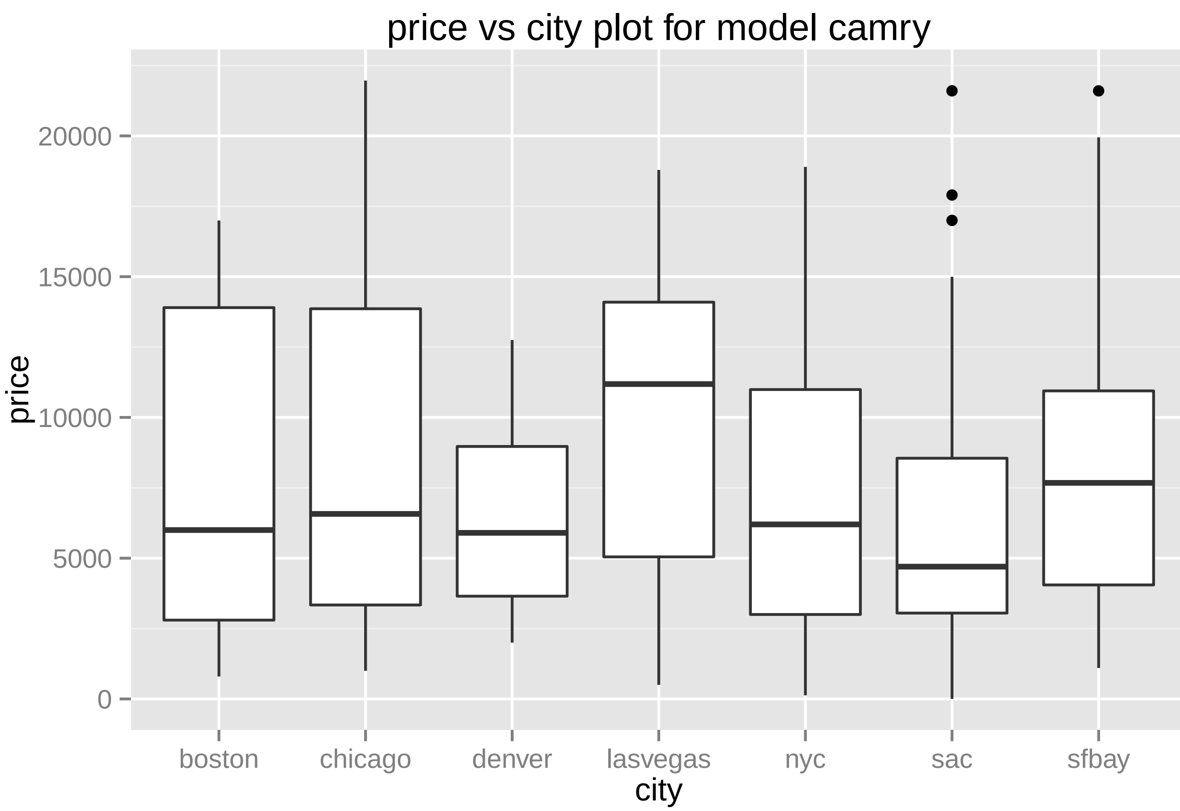
```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 385 8.602e+09  6502
##    2) age < 8.5 143 1.942e+09 11560
##      4) age < 3.5 63 3.191e+08 14710 *
##      5) age > 3.5 80 5.086e+08  9083 *
##    3) age > 8.5 242 8.386e+08  3513
##      6) age < 10.5 41 8.582e+07  6054 *
##      7) age > 10.5 201 4.342e+08  2995
##       14) age < 15.5 120 2.531e+08  3543 *
##       15) age > 15.5 81 9.158e+07  2182 *
```

```r
# It is seen that, only age is valid in determining price,
# which means price has little relationship with odometer and
# condition. It may seem unreasonable at first, but consider
# this: A vehicle with higher age usually comes with high
# odometer and poor condition.

# In other words, age is related with odometer and condition.
# So it is not wired that price only relates with age.
```

```r
# Part II, Model of camry

# get the data
camry_data_total <- get_model_data("camry", FALSE)
camry_data <- get_model_data("camry")

# show the data
head(camry_data, n = 10)
```

```
##           price odometer condition age
## posted483 11900    56000  like new   6
## posted561  3250   157000      good  12
## posted611 15995    25815  like new   3
## posted654  3895   187398  like new  13
## posted668  2995   174753  like new  16
## posted736 14900    49809 excellent   3
## posted761 14900    60214  like new   3
## posted895  8990   120000 excellent   6
## posted917 16900    60318  like new   3
## posted918 10900   109153 excellent   4
```

```r
# Step 0, draw the boxplot of city vs price.

library(ggplot2)
ggplot(camry_data_total, aes(city, price)) + geom_boxplot() +
    ggtitle("price vs city plot for model camry")
```



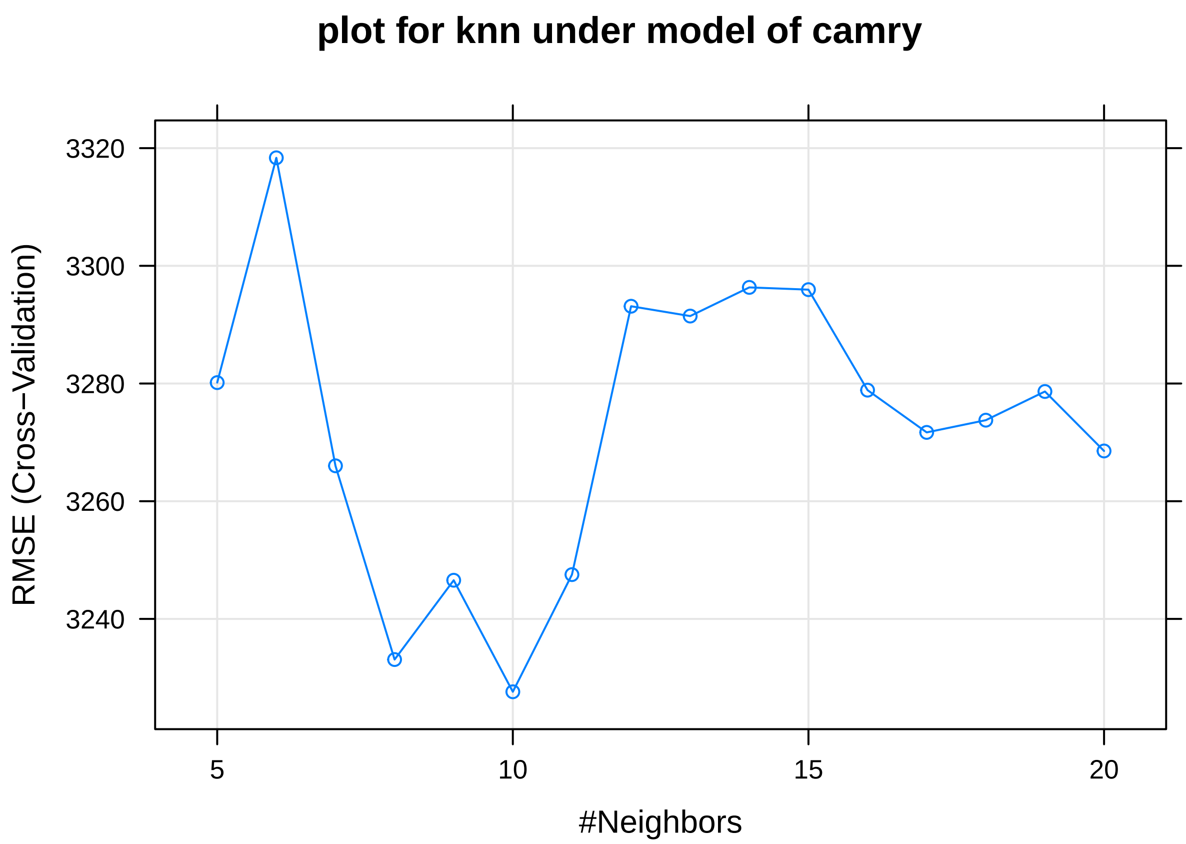

```r
# The result indicates that, city has almost no effect on
# price. The mean is almost identical, and the only
# difference relates to the standard deviation, namely the
# fluctuation of the price. The variance is high in city of
# boston, lasvegas and sac.

# Step I, find which model is the best

# As in part I, I will first set the cross validation
# parameter.  The parameters I choose is also 5-fold cross
# validation, repeating 3 times.
set.seed(1234)
fitControl <- trainControl(method = "cv", number = 5, repeats = 3)

# The lm approach
camry_fit_lm <- train(price ~ ., data = camry_data, method = "lm",
    trControl = fitControl)
camry_fit_lm
```

```
## Linear Regression
##
## 362 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 289, 289, 290, 290, 290
## Resampling results
##
##   RMSE      Rsquared   RMSE SD   Rsquared SD
##   2597.181  0.7597506  411.9615  0.08184612
##
##
```

```r
# It is easily seen that the lm fit result is RMSE: 2597.181

# The knn approach
camry_fit_knn <- train(price ~ ., data = camry_data, method = "knn",
    trControl = fitControl, tuneGrid = expand.grid(k = 5:20))
camry_fit_knn
```

```
## k-Nearest Neighbors
##
## 362 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 289, 290, 290, 289, 290
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   RMSE SD   Rsquared SD
##    5  3280.155  0.6313673  172.4323  0.02745973
##    6  3318.357  0.6209671  225.0221  0.03822056
##    7  3266.018  0.6304681  285.3505  0.04872618
##    8  3233.096  0.6376415  303.8355  0.05105653
##    9  3246.571  0.6340856  306.7217  0.05145727
##   10  3227.618  0.6376505  282.6197  0.04457102
##   11  3247.528  0.6322944  259.9219  0.04102766
##   12  3293.130  0.6215303  243.7536  0.03970984
##   13  3291.462  0.6218991  250.2624  0.04291379
##   14  3296.332  0.6203545  233.8302  0.04062346
##   15  3295.955  0.6210709  232.7443  0.03818591
##   16  3278.871  0.6250416  229.1795  0.03516115
##   17  3271.697  0.6260057  239.8512  0.03485988
##   18  3273.771  0.6257559  252.8094  0.03699070
##   19  3278.648  0.6248079  258.3618  0.03870118
##   20  3268.537  0.6274839  295.3273  0.04410654
##
## RMSE was used to select the optimal model using
##  the smallest value.
## The final value used for the model was k = 10.
```

```r
plot(camry_fit_knn, main = "plot for knn under model of camry")
```

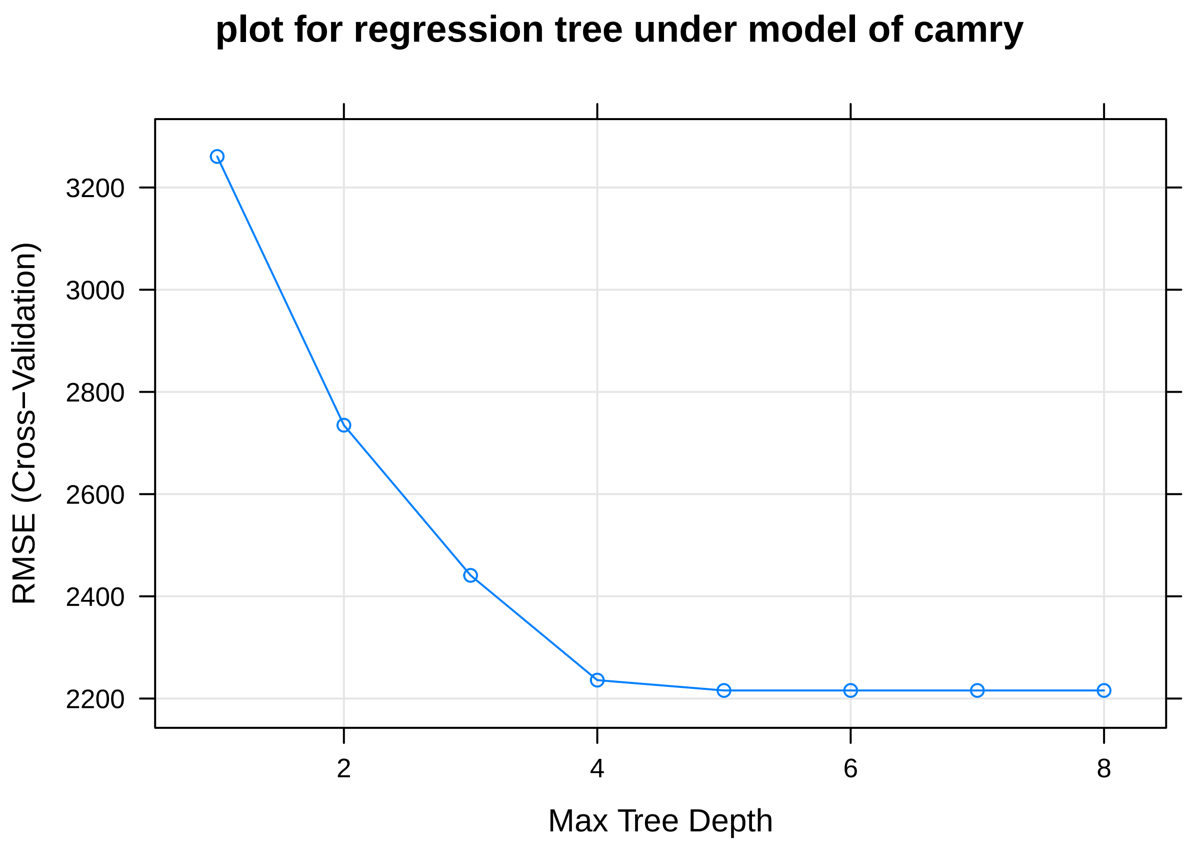



```r
# The best model for knn is k = 10, with RMSE: 3227.618

# The regression tree (tree) approach

camry_fit_tree <- train(price ~ ., data = camry_data, method = "rpart2",
    trControl = fitControl, tuneGrid = expand.grid(maxdepth = 1:8))
camry_fit_tree
```

```
## CART
##
## 362 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 290, 290, 289, 289, 290
## Resampling results across tuning parameters:
##
##   maxdepth  RMSE      Rsquared   RMSE SD   Rsquared SD
##   1         3260.676  0.6276138  335.8564  0.06324227
##   2         2734.880  0.7373570  313.3365  0.05283307
##   3         2441.003  0.7903440  271.6304  0.04049044
##   4         2236.051  0.8237073  230.3719  0.03220052
```

```
##   5         2215.835  0.8264884  262.6360  0.03693863
##   6         2215.835  0.8264884  262.6360  0.03693863
##   7         2215.835  0.8264884  262.6360  0.03693863
##   8         2215.835  0.8264884  262.6360  0.03693863
##
## RMSE was used to select the optimal model using
##  the smallest value.
## The final value used for the model was maxdepth = 5.
```

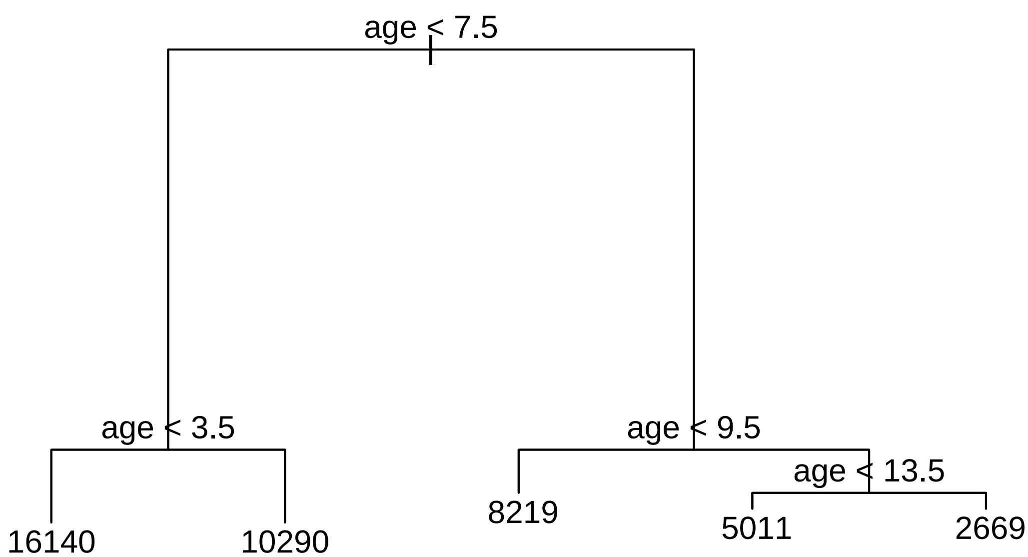
```
plot(camry_fit_tree, main = "plot for regression tree under model of camry")
```




```
# The best model for regression tree is k = 5, with RMSE:
# 2215.835

# Although knn is better than tree model, I will still use
# tree model, since it is more reasonable than a knn model
# with high k and easier to interpret.

# Step II, find the best regression tree model

library(tree)

# regression tree details
camry_fit_tree <- tree(price ~ ., data = camry_data)
camry_fit_tree
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 362 1.021e+10  7804
##    2) age < 7.5 135 2.223e+09 13240
##      4) age < 3.5 68 5.065e+08 16140 *
##      5) age > 3.5 67 5.620e+08 10290 *
##    3) age > 7.5 227 1.633e+09  4573
##      6) age < 9.5 42 2.546e+08  8219 *
##      7) age > 9.5 185 6.929e+08  3745
##       14) age < 13.5 85 2.820e+08  5011 *
##       15) age > 13.5 100 1.589e+08  2669 *
```

```r
# plot the result
plot(camry_fit_tree)
text(camry_fit_tree, pretty = 0)
```

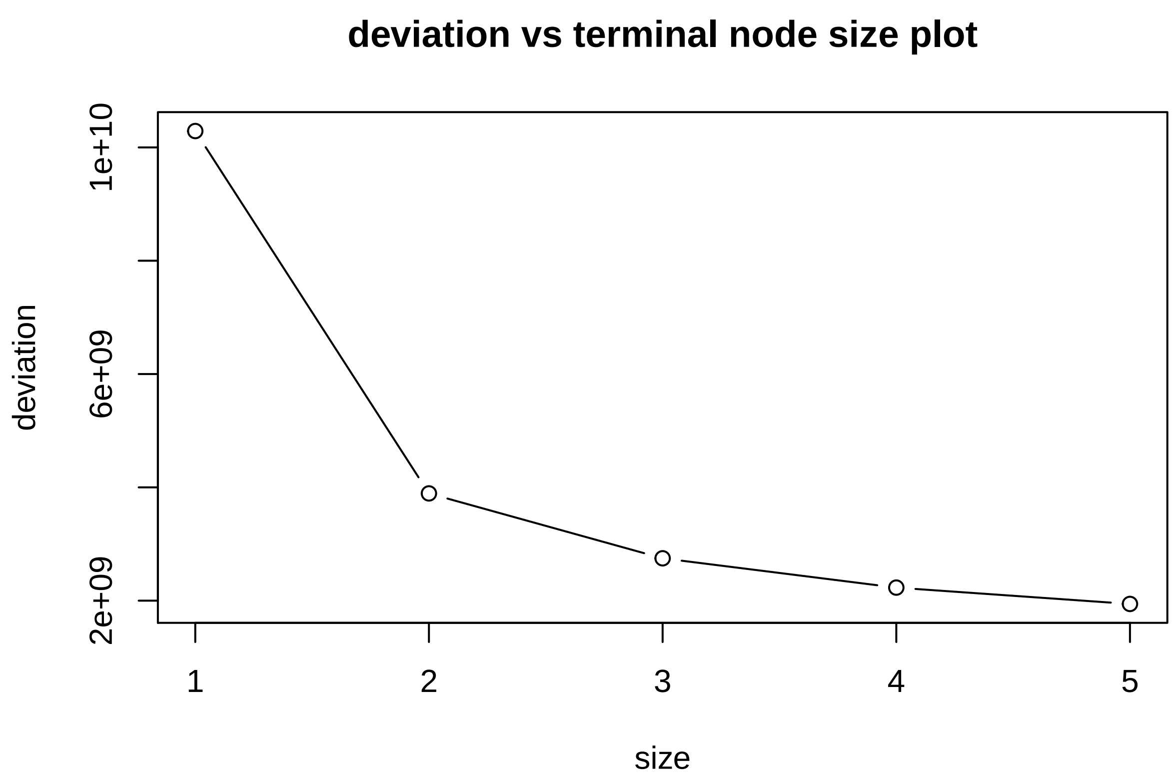



```r
# use cross validation to confirm the best tree depth
camry_fit_tree_cv <- cv.tree(camry_fit_tree)
camry_fit_tree_cv
```

```
## $size
## [1] 5 4 3 2 1
##
## $dev
```

```
## [1]  1942225203  2230394219  2747682473  3892874889
## [5] 10288916848
##
## $k
## [1]       -Inf  252069843  685282192 1154613657 6355735256
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
# plot the result, the relationship between nodes and
# deviation
plot(camry_fit_tree_cv$size, camry_fit_tree_cv$dev, type = "b",
    xlab = "size", ylab = "deviation", main = "deviation vs terminal node size plot")
```

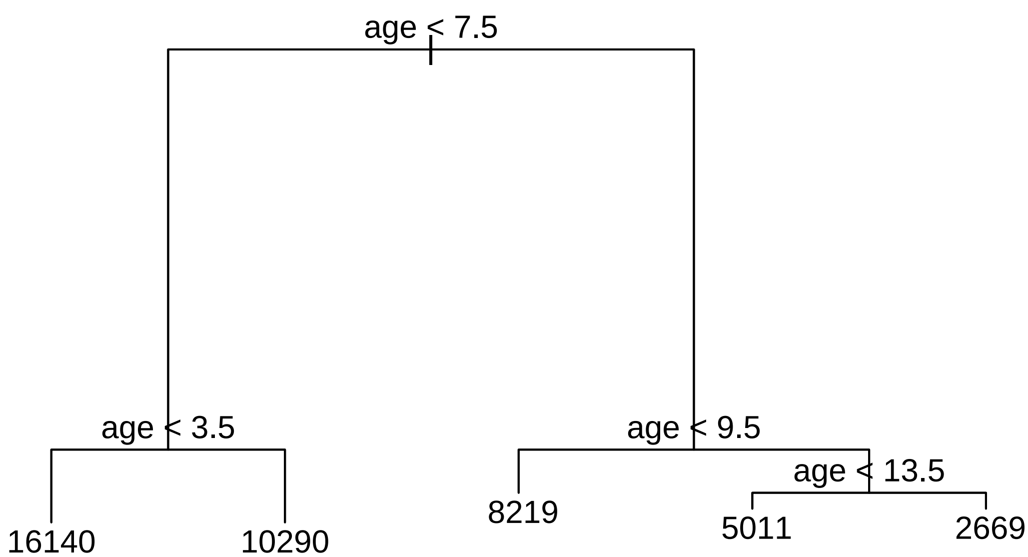



```
# It seems size 5 is the best option
camry_fit_tree_prune <- prune.tree(camry_fit_tree, best = 5)
camry_fit_tree_prune
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 362 1.021e+10  7804
```

```
##    2) age < 7.5 135 2.223e+09 13240
##      4) age < 3.5 68 5.065e+08 16140 *
##      5) age > 3.5 67 5.620e+08 10290 *
##    3) age > 7.5 227 1.633e+09  4573
##      6) age < 9.5 42 2.546e+08  8219 *
##      7) age > 9.5 185 6.929e+08  3745
##       14) age < 13.5 85 2.820e+08  5011 *
##       15) age > 13.5 100 1.589e+08  2669 *
```

```r
# plot the best result
plot(camry_fit_tree_prune)
text(camry_fit_tree, pretty = 0)
```




```r
# Now let's do a prediction, the data I use is the first
# observation:
camry_data[1, ]
```

```
##            price odometer condition age
## posted483 11900    56000  like new   6
```

```r
# let's use tree model to predict
camry_prediction_one <- predict(camry_fit_tree_prune, camry_data[1,
    ])
camry_prediction_one
```

```
## posted483
##  10291.24
```

```r
# It is very close, again!

# Step III, Analyze the result

# Let's look back the regression tree here:
camry_fit_tree_prune
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 362 1.021e+10  7804
##    2) age < 7.5 135 2.223e+09 13240
##      4) age < 3.5 68 5.065e+08 16140 *
##      5) age > 3.5 67 5.620e+08 10290 *
##    3) age > 7.5 227 1.633e+09  4573
##      6) age < 9.5 42 2.546e+08  8219 *
##      7) age > 9.5 185 6.929e+08  3745
##       14) age < 13.5 85 2.820e+08  5011 *
##       15) age > 13.5 100 1.589e+08  2669 *
```

```r
# It is quite like the situation of civic, so I will copy my
# previous analysis here:

# It is seen that, only age is valid in determining price,
# which means price has little relationship with odometer and
# condition. It may seem unreasonable at first, but consider
# this: A vehicle with higher age usually comes with high
# odometer and poor condition.

# In other words, age is related with odometer and condition.
# So it is not wired that price only relates with age.

# Conclusion

# To conclude, I will use my regression model, since I think
# the result is good, and it is reasonable, and can give me
# some guide.
```