Zhen Zhang

STA141 Assignment4 Code

```r
load(url("http://eeyore.ucdavis.edu/stat141/Data/vehicles.rda"))

## 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
```

```r
  result <- sum(!is.na(body))
  cat('\n')
  cat(sprintf("The number of matching is %g", result))
  cat('\n')
  invisible(result)
}

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)

# 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))
```

```r
})

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

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

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

### (b)

# 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]{13}[0-9]{4}')

# 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)

# add to vposts
vposts$VIN <- VIN_from_body_single

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

### (c)

# 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
```

```r
  # 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')

# 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)

# add to vposts
vposts$phone <- phone_from_body_single

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

### (d)

# 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)')
```

```r
email_from_body <- extract_from_data(Reduce(paste_re, 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)

# add to vposts
vposts$email <- email_from_body_single

# show some results I get
email_from_body_single[which(!is.na(email_from_body_single))]

### (e)

# 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)

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

# show some results I get
```

```r
head(year_from_description_single, n = 10)

### (f)

# 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.

# extract the maker name
vposts_maker_names <- unique(vposts$maker)
vposts_maker_names <- vposts_maker_names[!is.na(vposts_maker_names)]

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
```

```r
  # 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 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)

# *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
```

```r
# 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. 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) {
```

```r
  # 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) {

    # 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
```

```r
  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)
}

# 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.
unlist(model_modify_rule['toyota'])

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)

## 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])
})

# 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
  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
}

# 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')

# 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)

# 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")

# 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)
```

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

# 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
plot(civic_fit_knn, main = 'plot for knn under model of civic')

# 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
plot(civic_fit_tree, main = 'plot for regression tree under model of civic')

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

# use cross validation to confirm the best tree depth
civic_fit_tree_cv <- cv.tree(civic_fit_tree)
civic_fit_tree_cv
# 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')
# It seems size 5 is the best option
civic_fit_tree_prune <- prune.tree(civic_fit_tree, best = 5)
```

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

# Now let's do a prediction, the data I use is the first observation:
civic_data[1,]
# let's use tree model to predict
civic_prediction_one <- predict(civic_fit_tree_prune, civic_data[1,])
civic_prediction_one
# It is very close!

# Step III, Analyze the result

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

# 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.

# 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)

# 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")

# 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
```

```r
# 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

# 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
plot(camry_fit_knn, main = 'plot for knn under model of camry')

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

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

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

# Now let's do a prediction, the data I use is the first observation:
camry_data[1,]
# let's use tree model to predict
camry_prediction_one <- predict(camry_fit_tree_prune, camry_data[1,])
camry_prediction_one
# It is very close, again!

# Step III, Analyze the result

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

# 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.
```