HW4

Weitong(Jessie) Lin

November 11, 2015

read data

```
options(width = 100)
setwd('~/Desktop/UC Davis/141/STA141 Assignment IV')
print(load("vehicles.rda"))
```

```
## [1] "vposts"
```

1. get price from 'body':

The purpose for this step is to exact "price" information from the "body" part. Although we already have a variable "price", however, we still try to see whether there are typo or missing values in 'price' can be fixed from the 'body' information.

- First, let's use the most common way to express the price: \$xx,xxx or \$xxxxx if there are 5 digits. For this type of price, what exists is a '\$' sign, following by some numbers. There may be also some ',' which is used to split the number every 3 digits.

```
#########
# Strategy 1: $xx,xxx or $xxxxx #
#########

# The meaning of rx_price1: find the things which has '$' sign at the begining,
# then following by one or more elements like '0-9' and ','
rx_price1 = "\\$[0-9,]+"
# to see which observation matches this first strategy
price_count1 = regexpr(rx_price1, vposts$body)
# extract the match part from each observation
price1 = regmatches(vposts$body, gregexpr(rx_price1, vposts$body))
# To see how many observations match this first strategy
table(price_count1 != -1)
```

```
##
## FALSE  TRUE
## 20265 14412
```

From the result we can see that there are 14412 obervations has the prices with the pattern of strategy one. We first convert them into integer type with no '\$' sign and ',' and also make those no-match value into "NA".

```
# clean matched price
price1 = sapply(price1, function(post){

    # if:
    #   - no value: insert "NA"
    #   - has value: remove "$", ','

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # remove "$" ","
    post = gsub("[$,]*", "", unlist(post))
    # then change into 'integer' type
    post = as.integer(post)
  }
 })
```

Now let's see whether there are some observations with multi-matches.

```
# get the number of matches for each observation
num_match = sapply(price1, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match)
```

```
## num_match
##     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16
## 30849  2521   498   367   149    65    58    61    13    34     7     4    30     2     2     2
##    17    18    19    20    23    24    26    29    31    35    38    40
##     1     2     3     1     1     1     1     1     1     1     1     1
```

From the result, we can see that there are way too much posts has multi-prices in body. After looking several posts(I won't show here because there are too much results), I find that a lot of posts write the price for different parts of car (such as Back up Camera and radio). So I decide to choose the largest number as the price of that car for those post which have multi-matches in body.

```
#assign the max value for each post
price1 = sapply(price1, max)
```

- Second, let's use the an other common way to express the price: $xxxk. For this type of price, what exists is a '$' sign, following by some numbers and then a 'k' which represents 'thousands'.

```
#########
# Strategy 2: $xxk #
#########

# The meaning of rx_price2: find the things which has '$' sign at the begining,
# then following by 1-3 elements like '0-9' and then following by a 'k'
rx_price2 = "\\$[0-9]{1,3}k"
# to see which observation matches this second strategy, ignore the case
price_count2 = regexpr(rx_price2, vposts$body, ignore.case = TRUE)
# extract the match part from each observation
price2 = regmatches(vposts$body, gregexpr(rx_price2, vposts$body, ignore.case = TRUE))
# To see how many observations match this first strategy
table(price_count2 != -1)
```

```
##
## FALSE  TRUE
## 34531   146
```

From the result, we can see that we do have some matches. Let's see how they looks like to make sure we indeed get the price, not other things.

```
# show all price except there are no price there
unlist(price2[which(price_count2 != -1)])
```

```
##   [1] "$35k"  "$10k"  "$10K"  "$10K"  "$55K"  "$4k"   "$4K"   "$3k"   "$60k"  "$7k"   "$3k"
##  [12] "$25k"  "$130k" "$9K"   "$11K"  "$20k"  "$9k"   "$16K"  "$45K"  "$40k"  "$16k"  "$4k"
##  [23] "$20K"  "$25k"  "$15K"  "$10k"  "$5k"   "$50k"  "$15K"  "$127K" "$70K"  "$52k"  "$43k"
##  [34] "$40k"  "$16k"  "$25k"  "$130k" "$17k"  "$2k"   "$1k"   "$100k" "$10K"  "$14k"  "$40k"
##  [45] "$67K"  "$10k"  "$100K" "$10k"  "$12k"  "$2K"   "$20K"  "$6k"   "$10k"  "$1k"   "$6k"
##  [56] "$6k"   "$2k"   "$5k"   "$12k"  "$6k"   "$20k"  "$3k"   "$30k"  "$55k"  "$40K"  "$2k"
##  [67] "$16K"  "$20K"  "$10k"  "$26k"  "$80k"  "$10k"  "$55K"  "$65K"  "$2K"   "$24k"  "$23k"
##  [78] "$27k"  "$10k"  "$5K"   "$73K"  "$30K"  "$120K" "$107k" "$2k"   "$3k"   "$20k"  "$80k"
##  [89] "$27k"  "$15k"  "$100K" "$126K" "$100k" "$13k"  "$100k" "$29k"  "$25k"  "$12k"  "$17k"
## [100] "$7k"   "$2k"   "$100k" "$15k"  "$6K"   "$30k"  "$45K"  "$100k" "$25K"  "$35K"  "$7k"
## [111] "$8k"   "$5k"   "$1k"   "$15k"  "$2K"   "$35k"  "$45k"  "$93K"  "$70K"  "$98K"  "$145K"
## [122] "$153K" "$116K" "$48k"  "$70k"  "$19k"  "$2k"   "$10k"  "$42k"  "$50K"  "$14K"  "$10K"
## [133] "$17k"  "$4K"   "$7k"   "$50k"  "$3k"   "$15k"  "$70K"  "$41K"  "$65K"  "$20k"  "$1K"
## [144] "$60K"  "$2k"   "$15k"  "$60K"  "$8k"   "$27K"  "$8k"   "$6K"   "$8K"   "$4k"   "$10K"
## [155] "$11K"  "$10K"  "$11K"  "$7K"   "$83K"  "$83k"  "$8k"   "$35K"  "$7k"   "$12K"  "$12K"
## [166] "$70K"  "$55K"  "$5k"   "$34K"  "$2k"   "$2k"   "$5k"
```

From the above results, I think what I exact are all prices. So now let's start deal with all prices under the second strategy.

From the table we can see that there are 146 obervations has the prices with the pattern of strategy two. We first convert them into integer type with no '$' sign and write them in numeric full type, not 'K' type, and also make those no-match value into "NA".

```
# clean matched price
price2 = sapply(price2, function(post){

    # if:
    #   - no value: insert "NA"
    #   - has value: remove "$", then write into full numeric type

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # remove "$" "k", ignore the case
    post = gsub("[$k]*", "", unlist(post),ignore.case = TRUE)
    # then change into 'integer' type then times 1000
    post = as.integer(post) * 1000
  }
 })
```

Now let's see whether there are some observations with multi-matches.

```
# get the number of matches for each observation
num_match2 = sapply(price2, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match2)
```

```
## num_match2
##     1     2     3     4     5
## 34659    13     3     1     1
```

From the result, we can see that there are some posts have multi-prices in body. The same as the strategy one, I decide to choose the largest number as the price of that car for those post which have multi-matches in body.

```
#assign the max value for each post
price2 = sapply(price2, max)
```

For these two strategies, it's hard to judge which one is right for the overlapping part. I've looked at some posts and decide to choose the larger one as my extracted price.

```
# define a new column which will be filled with the new price later on
price_final = rep(0, length(price1))
# get these three price together
price_extr=cbind(price_final, price1, price2)
# the function used to select the price from two strategies
new_price = function(row) {
    # INPUT:
    #   - row: a row of data
    #
    # OUTPUT: the updated value
    #
  if (table(is.na(row))[1] != 1){
    # if there are 0 or 1 NA exist, the price will the larger one
    row[1] = max(row[!is.na(row)])
  } else if (table(is.na(row))[1] == 1){
    # if both of them are all NA, then the price is NA
    row[1] = NA
  }
  # return back the price
  return (row[1])
}

# go through each row , get the new price
new_price = apply(price_extr, 1, new_price)
# see how many new price we extract
table(!is.na(new_price))
```

```
##
## FALSE  TRUE
## 20279 14398
```

Here we can see that we have 14398 posts that have price information from body.

Now let's see the extracted price and the real price column.

```
table(vposts$price == new_price)
```

```
##
## FALSE  TRUE
##  3476  9813
```

Here we can find that 9813 posts' prices are matched, ignoring "NA" posts. Recall from the Assignment I, there are a lot of unusual prices happened in 'price' column. Extracting price information from other column will improve the accuracy of 'price'. So I will take the larger one between new_price and original price as the final price.

```
# define a new column which will be filled with the new price later on
price_final = rep(0, length(new_price))
# get these three price together
price_extr=cbind(price_final, new_price, vposts$price)
# the function used to select the price from two strategies
final_price = function(row) {
    # INPUT:
    #   - row: a row of data
    #
    # OUTPUT: the updated value
    #
  if (table(is.na(row))[1] != 1){
    # if there are 0 or 1 NA exist, the price will the larger one
    row[1] = max(row[!is.na(row)])
  } else if (table(is.na(row))[1] == 1){
    # if both of them are all NA, then the price is NA
    row[1] = NA
  }
  # return back the price
  return (row[1])
}

# go through each row , get the new price
```

```
final_price = apply(price_extr, 1, final_price)

# add the adjusted price to the vposts
vposts$final_price = final_price
```

2. VIN

The purpose for this is to exact the VIN which is unique for each car. It can show year it was built, type and model of the car, safety features, body style, engine type, etc. It's a very important imformation for a second car. Now I will try to exact VIN code from body

- When I look at the website 'http://www.autocheck.com/vehiclehistory/autocheck/en/vinbasics', I learn that basiclly a standard VIN number has 17 digits. The first 11 digits can be numbers or letters. The last six digits must be 6 numbers. So I will go through this to do my strategy. Also, some posts may have "VIN:" some may not. Then I will make it optional.

```
#########
# Strategy: '[VIN:] optional, 11 [0-9] or [alpha] then 6 [0-9]' #
#########

# The meaning of rx_year: things that begin with 4 numbers
rx_vin = '[VIN: ]?[0-9a-z]{11}[0-9]{6}'
# to see which observation matches this strategy
vin_count = regexpr(rx_vin, vposts$body, ignore.case = TRUE)
# extract the match part from description of each observation
VIN = regmatches(vposts$body, gregexpr(rx_vin, vposts$body, ignore.case = TRUE))
# To see how many observationsmatch this strategy
table(vin_count != -1)
```

```
##
## FALSE  TRUE
## 26943  7734
```

There are 7734 posts have this pattern.

Now let's see whether there are some observations with multi-matches.

```
# get the number of matches for each observation
num_match7 = sapply(VIN, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match7)
```

```
## num_match7
##     0     1     2     3     6     7     8    19
## 26943  6862   321   536     1    12     1     1
```

From the result, we can see that there are way too many posts have multi-matches. So what I need to do is to assign the first element as the vin, and also change all 'character(0)' into 'NA'

```
# clean matched VIN
VIN = sapply(VIN, function(post){

    # if:
    #   - no value: insert "NA"

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # assign the first element
    post = post[[1]][1]
  }
 }
 )
```

Finally, we have 7734 posts containing Year information. We add this new column to vposts:

```
vposts$VIN = VIN
```

3. Phone number :

The purpose for this step is to exact "phone number" information from the "body" part. Since we don't have a column for phone number, this extraction will be very useful as a contact informationto call the people who selling this car.

- There are 10 digits for phone number. However, the form may vary a lot. Some people may put area code in'()'and some don't. Some people may put'-'or'.' between the numbers like (xxx) xxx-xxxx. Thus, I will consider all possible inserts and write the pattern.

```
#########
# Strategy: 10 digits, with all possible insert '-' and '()'  #
#########

# The meaning of rx_phonenum: first, begin with 3 numbers, '()' is optional, then,
# zero or more '-' and '.' are optional, then followed by 3 numbers, zero or more '-' and '.'
# are optional, then followed by 4 numbers.
rx_phonenum = '\\(?[0-9]{3}\\)?[.-]? *[0-9]{3}[.-]? *[.-]?[0-9]{4}'
# to see which observation matches this strategy
phonenum_count = regexpr(rx_phonenum,vposts$body)
# extract the match part from each observation
phone_number = regmatches(vposts$body,gregexpr(rx_phonenum,vposts$body))
# To see how many observationsmatch this strategy
table(phonenum_count != -1)
```

```
##
## FALSE  TRUE
## 18158 16519
```

There are 16519 posts have this pattern.

Now let's see whether there are some observations with multi-matches.

```
# get the number of matches for each observation
num_match4 = sapply(phone_number, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match4)
```

```
## num_match4
##     0     1     2     3     4     5     6     7     8     9    10    11    12    14    16    18
## 18158  5425  4376  4340  1468   509   218    39    91    25     4     2     5     3     4     3
##    21    34
##     6     1
```

From the result, we can see that there are way too much posts have multi-matches in body. I assign the first phone number as their phone numbers and also change all 'character(0)' into 'NA'

```
# clean matched phone number
phone_number = sapply(phone_number, function(post){

    # if:
    #   - no value: insert "NA"
    #   - has value: assign the first element

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # assign the first element
    post = post[[1]][1]
  }
 }
 )
```

Finally, we have 16519 posts containing phone_number information. We add this new column to vposts:

```
vposts$phone_number = phone_number
```

4. Email:

The purpose for this step is to exact "email" information from the "body" part. Since we don't have a column for email, this extraction will be very useful as a contact information

- Let's use the most common way to express the email: 'xxxxxx@xxx.(com|org|edu|net|gov|int|mil)'. The reference of the list of Internet top-level domains is from the internet : 'https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains'

```
#########
# Strategy: 'xxxxxx@xxx.(com|org|edu|net|gov|int|mil)' #
#########

# The meaning of rx_email: find the things which have zero or more elements which can be
# alphabets, numbers or '-', then following by a '@', then again following by zero or
# more elements which can be alphabets or number then by '.' and Internet top-level domains.
rx_email = '([-a-z0-9]*\\@[a-z0-9]*[.](com|org|edu|net|gov|int|mil))'
# to see which observation matches this strategy, ignore the case
email_count = regexpr(rx_email, vposts$body, ignore.case = TRUE)
# extract the match part from each observation
email = regmatches(vposts$body, gregexpr(rx_email, vposts$body, ignore.case = TRUE))
# To see how many observationsmatch this strategy
table(email_count != -1)
```

```
##
## FALSE  TRUE
## 34573   104
```

There are 104 posts have this pattern. Since there only 104 observations, let's see whether they are what we want.

```
# show all emails except there are no email there.
# For this report I will only show the first 6 email
head(unlist(email[which(email_count != -1)]))
```

```
## [1] "imports4sale@yahoo.com" "sales@rt28motors.com"   "bylosauto@aol.com"
## [4] "bylosauto@aol.com"      "Amber44@charter.net"    "smdwoods@aol.com"
```

Here we can find that all emails look fine except one:

```
#find the wild one
email[which(email_count != -1)][51]
```

```
## [[1]]
## [1] "@mytruckcity.com"
```

It implied that something missing before '@', so let's see the original vposts$body of it

```
vposts[which((email_count != -1))[51],'body']
```

```
## [1] "\n        For Sale: 2010 Freightliner Cascadia Mid-roof XT (extra tall). This is a beautiful, w
```

Then we can find that there is a white space in front of '@' which made the first part of email missing match with our pattern. Thus we fix this email:

```
#change value
email[which(email_count != -1)][51] = 'Nathaniel@mytruckcity.com'
```

Now let's see whether there are some observations with multi-matches.

```
# get the number of matches for each observation
num_match3 = sapply(email, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match3)
```

```
## num_match3
##     0     1     2
## 34573   100     4
```

From the result, we can see that there are 4 posts have multi-email in body. I assign the first email as their emails and also change all 'character(0)' into 'NA'

```
# clean matched email
email = sapply(email, function(post){

    # if:
    #   - no value: insert "NA"
    #   - has value: assign the first element

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # assign the first element
    post = post[[1]][1]
  }
 }
 )
```

Finally, we have 104 posts containing Email information. We add this new column to vposts:

```
vposts$email = email
```

5. Year

The purpose for this is to exact the year when this car produced. This is also a very important information because usuall the older the car, the lower the price. This information can let the buyer filter the year with their expaected age of car.

- When I see the vposts$description, I find that most of them begin with the year with 4 digits

```
#########
# Strategy: 'xxxx' #
#########
```

```
# The meaning of rx_year: things that begin with 4 numbers
rx_year = '^[0-9]{4}'
# to see which observation matches this strategy
year_count = regexpr(rx_year, vposts$description)
# extract the match part from description of each observation
Year_descrip = regmatches(vposts$description, gregexpr(rx_year, vposts$description))
# To see how many observationsmatch this strategy
table(year_count != -1)
```

```
##
## FALSE  TRUE
##  8544 26124
```

There are 26124 posts have this pattern.

Now let's see whether there are some observations with multi-matches.

```
# get the number of matches for each observation
num_match5 = sapply(Year_descrip, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match5)
```

```
## num_match5
##     0     1
##  8553 26124
```

From the result, we can see that all them are the unique result . So what I need to do is to change all 'character(0)' into 'NA'

```
# clean matched year
Year_descrip = sapply(Year_descrip, function(post){

    # if:
    #   - no value: insert "NA"

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # change into integer type
    post = as.integer(post)
  }
 }
 )
```

Finally, we have 26124 posts containing Year information. We add this new column to vposts:

```
vposts$Year_descrip = Year_descrip
```

Now let's compare what the year column and the year we extract from 'description':

```
# to see how many of them are the same result
table(vposts$year == vposts$Year_descrip)
```

```
##
## FALSE  TRUE
##   345 25779
```

Here we can see that ignoring those posts which has "NA" in both two "year"s, there are 345 posts which are not the same for two 'year'. However, we can not conclude which pne is right because typos may happen in either 'year' or 'Year_descrip'. We can not decide which is the right directily. Thus, what I do is to extract the year infromation from 'vposts$header' to make sure that the accuracy of year.

The procedure is the same as the one in 'description'. The only defference is that we do it in 'header' this time

```
#########
# Strategy: 'xxxx' #
#########

# The meaning of rx_year: things that begin with 4 numbers
rx_year = '^[0-9]{4}'
# to see which observation matches this strategy
year_count1 = regexpr(rx_year, vposts$header)
# extract the match part from description of each observation
Year_header = regmatches(vposts$header, gregexpr(rx_year, vposts$header))
# To see how many observationsmatch this strategy
table(year_count1 != -1)
```

```
##
## FALSE  TRUE
##     1 34676
```

There are 34676 posts have this pattern. Only one don't match. Now let's see what happen for this post.

```
vposts[which(year_count1 == -1), 'header']
```

```
## [1] "04 vctvhmfdk"
```

Here we can see that there are only two digits exists, which is like the last two digits for yea. And I pretty sure it's 2004. So I fix it.

```
Year_header[which(year_count1 == -1)] = '2004'
```

Now let's see whether there are some observations with multi-matches.

```r
# get the number of matches for each observation
num_match6 = sapply(Year_header, function(data){

  #length for each list
  length(unlist(data))

  })

table(num_match6)
```

```
## num_match6
##     1
## 34677
```

From the result, we can see that all them are the unique result also no 'NA'. So I will change this list into a vector with integer type

```r
# clean matched year
Year_header = sapply(Year_header, as.integer)
```

Finally, we have 34677 posts containing Year information in header. We add this new column to vposts:

```r
vposts$Year_header = Year_header
```

Now let's compare what the year column and the year we extract from 'header':

```r
# to see how many of them are the same result
table(vposts$Year_header == vposts$year)
```

```
##
## FALSE  TRUE
##     1 34676
```

```r
vposts[vposts$Year_header != vposts$year, 'title']
```

```
## [1] "argolic eni-04 JEeP wraNgler Clean lEATHeR - $2532 (chicago)"
```

Here we can see that the only differnece is the post which has the last two digits '04' that I've just talked about. So Year_header is more accuracy than year.

So we use Year_header to compare with Year_descrip

```r
table(vposts$Year_header == vposts$Year_descrip)
```

```
##
## FALSE  TRUE
##   345 25779
```

We can see that 25779 are the same. I've looked at the first five posts and the typos of year happens in both side. It's hard to decide which is right. However, I more tend to choose to use "Year_header" to be the year when the car produced.

6. Model of car

The purpose is to extract car model from the title part. Different models of the same makers also have a huge price variation. Thus, 'model' information is very important for buyer so that then can filter the car according to price and model. - When I look at some posts' title, I find some pattern that almost all the title start with year then followed by maker then followed by model then something else.

```
#########
# Strategy: one #
#########

# The meaning of rx_modle1: start with 2 or 4 numbers, then an optional white space,
# then one or more letters/number or '-', then an optional white space,
# then one or more letters/number or '-', then something else
rx_model1 = '^[0-9]{2,4} ([-a-z0-9]+)[ ]?([-0-9a-z ]+).*'
# to see which observation matches this strategy
model_count1 = grepl(rx_model1, vposts$header, ignore.case = TRUE)
# extract the match part from description of each observation
maker1  = gsub(rx_model1, '\\1', vposts$header, ignore.case = TRUE)
model1  = gsub(rx_model1, '\\2', vposts$header, ignore.case = TRUE)
# To see how many observations match this strategy
table(model_count1)
```

```
## model_count1
## FALSE  TRUE
##    31 34646
```

We get 34646 matched posts. Since there are 31 which are not match, let's see why they don't match

```
vposts$header[which(model_count1 == FALSE)]
```

```
##  [1] "2012 $500 DOWN PROGRAMS!"              "2012 $500 DOWN PROGRAMS!"
##  [3] "2012 $500 DOWN PROGRAMS!"              "2012 $500 DOWN PROGRAMS!"
##  [5] "2012 $500 DOWN PROGRAMS!"              "2012 $500 DOWN PROGRAMS!"
##  [7] "2012 $500 DOWN PROGRAMS!"              "2012 $500 DOWN PROGRAMS!"
##  [9] "2012 $500 DOWN PROGRAMS!"              "2012 $500 DOWN PROGRAMS!"
## [11] "2006 **2006 Toyota Sienna AWD**"       "2012 * Honda Civic"
## [13] "2003 * Mitsubishi Galant"              "2004 4/cyl"
## [15] "2008 Ã¢Â\u0080Â¢Nissan Altima 2.5 SL" "2005 v"
## [17] "2015 H&H 8x10 trailer"                 "2015 H&H 8x12 trailer"
## [19] "2016 n/a"                              "2007 [lucern]"
## [21] "2016 0"                                "2003 f"
## [23] "2015 Y"                                "2009 ?"
## [25] "2009 ?"                                "2009 ?"
## [27] "2009 ?"                                "2016 C"
## [29] "1997 4.6 Ford f150 4x4"                "2015 N/A"
## [31] "1972 '72  Volkswagen  Camper Bus"
```

We find that some of them start with '*', so we renew our rx_model

```
#########
# Strategy: two #
#########

# The meaning of rx_modle1: start with 2 or 4 numbers, then an optional white space,
# then one or more letters/number or '-*', then an optional white space,
# then one or more letters/number or '-', then something else
rx_model2 = '^[0-9]{2,4} ([-a-z0-9*&]+)[ ]?([-0-9a-z]+).*'
# to see which observation matches this strategy
model_count2 = grepl(rx_model2, vposts$header, ignore.case = TRUE)
# extract the match part from description of each observation
maker2  = gsub(rx_model2, '\\1', vposts$header, ignore.case = TRUE)
model2  = gsub(rx_model2, '\\2', vposts$header, ignore.case = TRUE)
# To see how many observations match this strategy
table(model_count2)
```

```
## model_count2
## FALSE  TRUE
##    26 34651
```

Since there are 26 which are not match, let's see why they don't match

```
vposts$header[which(model_count1 == FALSE)]
```

```
##  [1] "2012 $500 DOWN PROGRAMS!"             "2012 $500 DOWN PROGRAMS!"
##  [3] "2012 $500 DOWN PROGRAMS!"             "2012 $500 DOWN PROGRAMS!"
##  [5] "2012 $500 DOWN PROGRAMS!"             "2012 $500 DOWN PROGRAMS!"
##  [7] "2012 $500 DOWN PROGRAMS!"             "2012 $500 DOWN PROGRAMS!"
##  [9] "2012 $500 DOWN PROGRAMS!"             "2012 $500 DOWN PROGRAMS!"
## [11] "2006 **2006 Toyota Sienna AWD**"      "2012 * Honda Civic"
## [13] "2003 * Mitsubishi Galant"             "2004 4/cyl"
## [15] "2008 Ã¢Â\u0080Â¢Nissan Altima 2.5 SL" "2005 v"
## [17] "2015 H&H 8x10 trailer"                "2015 H&H 8x12 trailer"
## [19] "2016 n/a"                             "2007 [lucern]"
## [21] "2016 0"                               "2003 f"
## [23] "2015 Y"                               "2009 ?"
## [25] "2009 ?"                               "2009 ?"
## [27] "2009 ?"                               "2016 C"
## [29] "1997 4.6 Ford f150 4x4"               "2015 N/A"
## [31] "1972 '72  Volkswagen  Camper Bus"
```

Here we can see that most of them are typos. However, 3 of them are really how to fit a general regualar expression. So I will fix them by hand.

```
maker2[which(model_count2 == FALSE)[11]] = 'Toyota'
model2[which(model_count2 == FALSE)[11]] = 'Sienna'
maker2[which(model_count2 == FALSE)[12]] = 'Nissan'
model2[which(model_count2 == FALSE)[12]] = 'Altima'
maker2[which(model_count2 == FALSE)[24]] = 'Ford'
model2[which(model_count2 == FALSE)[24]] = 'f150'
maker2[which(model_count2 == FALSE)[26]] = 'Volkswagen'
model2[which(model_count2 == FALSE)[26]] = 'Camper'
```

Now I will change the rest of them into NA

```
# clean matched model
model2 = sapply(model2, function(post){

    # if:
    #   - no value: insert "NA"

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # change into integer type
    post = post = as.character(post)
  }
 }
 )

# clean matched maker
maker2 = sapply(maker2, function(post){

    # if:
    #   - no value: insert "NA"

  if (length(post) == 0){
    # set as NA
    post = NA
  } else{
    # change into integer type
    post = post = as.character(post)
  }
 }
 )

# remove the colname
model2 = unname(model2)
```

I add this new column to vposts:

```
vposts$model_extr = tolower(model2)
```

Since we already know that we have 34646 posts have model, now let's see how many unique model we have:

```
length(unique(vposts$model_extr))
```

```
## [1] 1855
```

There are 1855 unique models. Now let see what are they and what're their frequency

```
# get the frequency for each model
Freq_model = as.data.frame(table(vposts$model_extr))
# order them in decreasing order
```

```r
Freq_model = Freq_model[order(Freq_model$Freq, decreasing = TRUE), ]
# show the first 10 results
colnames(Freq_model) = c('Model','Freq')
head(Freq_model,10)
```

```
##        Model Freq
## 504    civic  848
## 289   accord  840
## 913    grand  781
## 446    camry  757
## 307   altima  658
## 571  corolla  534
## 1206 mustang  436
## 1343     ram  386
## 1139  maxima  383
## 1008   jetta  346
```

Now I will set up a general model name list to correct other potencial misspelling words. My strategy is to get rid of all model whose frequency is smaller than 50, and also the charecter length is larger than three.

```r
# leave the model whose frequency is larger than or equal to 50
general_model = Freq_model[Freq_model$Freq >= 50, ]
# leave the model whose character is not one and two and three
general_model = general_model[nchar(as.character(general_model$Model)) > 3, ]
# to see how many models leave
dim(general_model)[1]
```

```
## [1] 112
```

We can see that we get 301 general model names list.

Now let write a function to correct all mispell model of the data.

```r
correct_model = function(general, data = vposts) {
  # INPUT:
  #      - general: each general model name
  #      - data: data set name
  # OUTPUT:
  #       set to the global environment and change vposts$model
  # find all match pattern of the general model name
  model_match = agrepl(pattern = general,
                       x = data[,'model_extr'],
                       max.distance = 1,
                       ignore.case = TRUE,
                       fixed = FALSE)
  # correct these matches
  data[model_match,'model_extr'] = general
  # set the change into a global environment, the code I find on the Internet
  assign('vposts', data, envir = .GlobalEnv)

}
```

Now let get the new correct_model

```r
# update the correction model
correction = sapply(as.character(general_model$Model), correct_model)
```

Now let's see the relevance between maker and model

```r
# get the data where maker and model are not NA
model_maker_data = subset(vposts, !is.na(vposts$maker) & !is.na(vposts$model_extr))
# extract maker and model
model_maker = data.frame(model_maker_data$maker, model_maker_data$model_extr)
# count the frequency
Freq_model_maker = data.frame(table(model_maker))
# order the frequency
Freq_model_maker = Freq_model_maker[order(Freq_model_maker$Freq, decreasing = TRUE), ]
# show the first 6 results
head(Freq_model_maker)
```

```
##       model_maker_data.maker model_maker_data.model_extr Freq
## 2037                    ford                        2500  906
## 24794                  honda                       civic  852
## 13778                  honda                      accord  842
## 22170                 toyota                       camry  761
## 14737                 nissan                      altima  658
## 28362                 toyota                     corolla  539
```

Here we can see that we correct a lot model name into a standard from

7. Modeling

First, I will pre-processing the data by using Duncan's code from the Assignment I partI - Condition

```r
# Print out conditions so we can cut and paste them into smaller
# categories. There's really no way out of this since a human has to decide
# what the new categories should be.
conditions = levels(vposts$condition)
conditions = sprintf('"%s",\n', conditions)
# Define new categories.
new_cats = list(
  excellent = c("excellent"),
  good = c("good", "very good"),
  "like new" = c("like new", "mint", "new", "pre owned", "pre-owned", "preowned", "preownes"),
  used = c("0used", "used"),
  fair = c("fair", "nice", "nice teuck"),
  salvage = c("complete parts car, blown engine", "front side damage", "hit and run :( gently",
              "muscle car restore", "needs bodywork", "needs restoration!", "needs restored",
              "needs total restore", "needs work", "needs work/for parts", "nice rolling restoration",
              "not running", "parts", "project", "project car", "rebuildable project", "restoration",
              "restoration project", "restore", "restored", "salvage", "rough but runs"),
  other = c("207,400", "ac/heater", "carfax guarantee!!", "certified", "honnda", "superb original" )
)

# Convert conditions to new categories.
```

```r
vposts$new_cond = vposts$condition
levels(vposts$new_cond) = c(levels(vposts$new_cond), "other")

for (i in seq_along(new_cats)) {
  new_cat = names(new_cats)[[i]]
  vposts$new_cond[vposts$new_cond %in% new_cats[[i]]] = new_cat
}

vposts$new_cond = factor(vposts$new_cond)
```

- age

```r
vposts$age = 2015 - vposts$Year_header
deal_age = subset(vposts, age < 80 & age > 0)
```

- odometer

```r
deal_odometer = subset(deal_age, odometer < 5e6)
```

- price

```r
deal_price = subset(deal_odometer, final_price < 2e5 & final_price > 1000)
```

According to table list above, I decide to choose 'toyota camry' and 'nissan altima' as my two models of car for the 'modeling' part.

```r
# get rid of all used variables with are NA
modeling_data = with(deal_price, subset(deal_price, !is.na(final_price)
                                        & !is.na(odometer)
                                        & !is.na(age)
                                        & !is.na(new_cond)
                                        & !is.na(city)))
# extract two dataset for two models
modeling_data_1 = modeling_data[modeling_data$maker=='toyota'
                                & modeling_data$model_extr=='camry',
                                c('final_price', 'odometer', 'age', 'new_cond', 'city')]
modeling_data_2 = modeling_data[modeling_data$maker=='nissan'
                                & modeling_data$model_extr=='altima',
                                c('final_price', 'odometer', 'age', 'new_cond', 'city')]
# to see how may data we have now
dim(modeling_data_1)[1]
```

```
## [1] 350
```

```r
dim(modeling_data_2)[1]
```

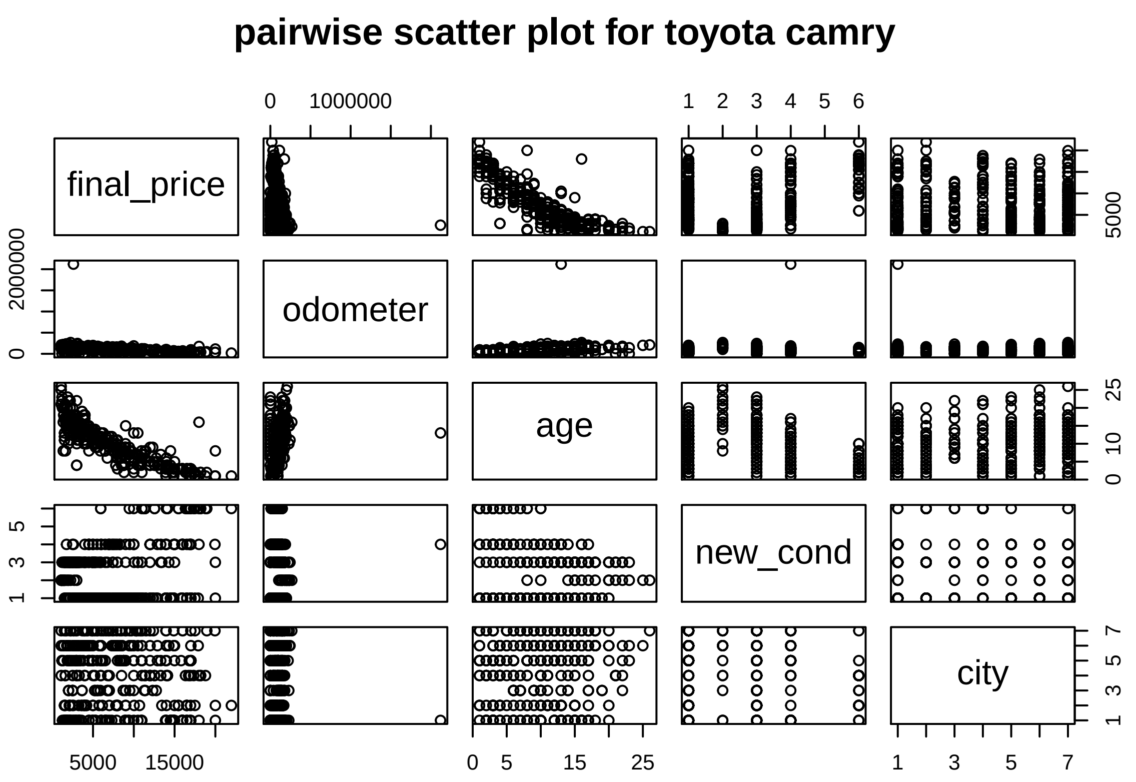
```
## [1] 240
```

Here we can see that we have 350 data for 'toyota camry' and 240 data for 'nissan altima'.

Now let do modeling part: ## general analysis First let's draw pairwise scatter plot for each model:

- ‘toyota camry’

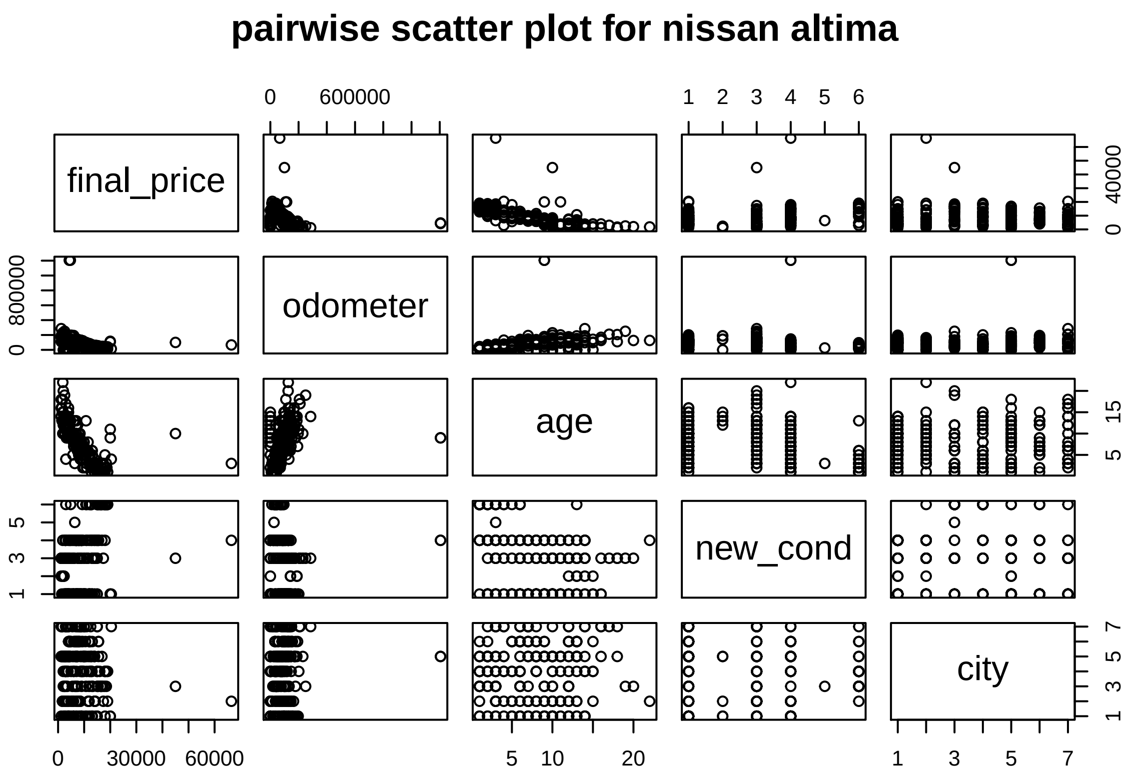
```r
pairs(modeling_data_1, main ='pairwise scatter plot for toyota camry')
```



From the plot, we can see that price may has a negative relationship with odometer and age , odometer and age have a slight positive relationship for toyota camry

- ‘nissan altima’

```r
pairs(modeling_data_2, main ='pairwise scatter plot for nissan altima')
```



From the plot, we can see that price may has a negative relationship with odometer and age, odometer and age have a slight positive relationship for nissan altima

a linear model

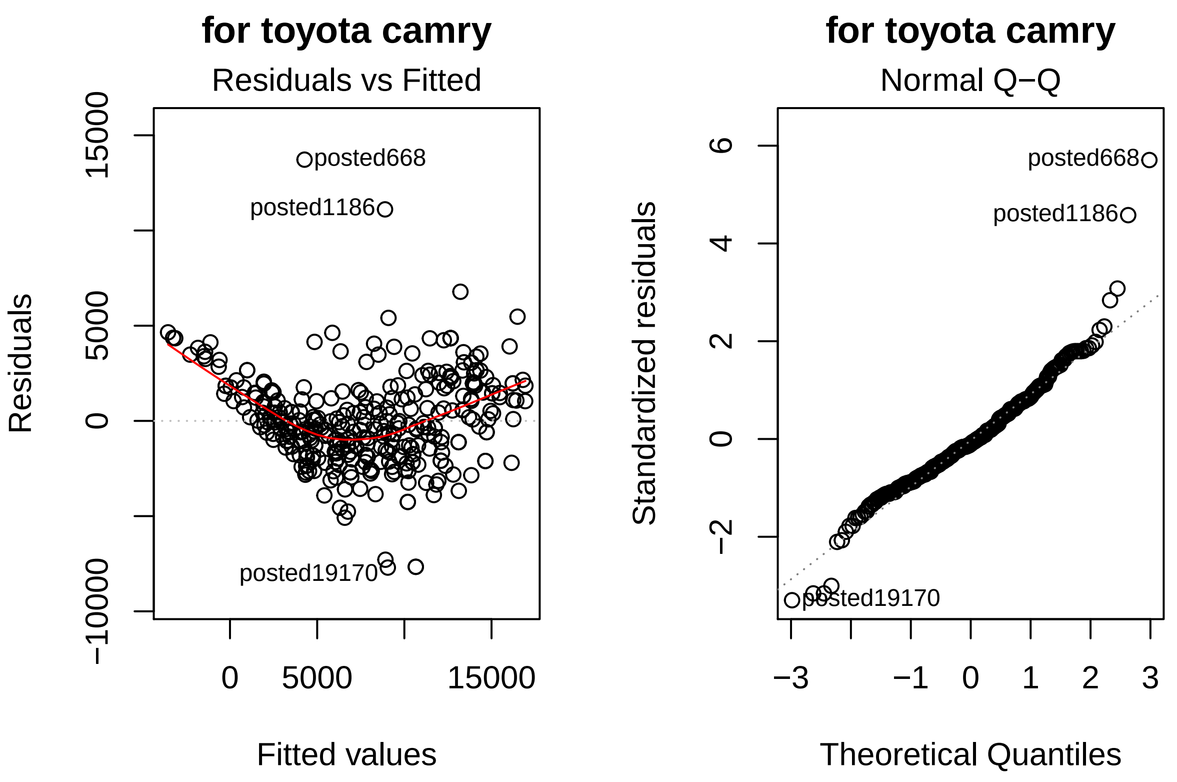
First, I will regress the price with odometer + age + new_cond + city. The assumption for a linear model is that the error is distributed as normal. Later on this assumption will be used when doing diagonitics.

- 'toyota camry'

```r
# fit a linear model
fit1 = lm(final_price ~ odometer + age + new_cond + city, data= modeling_data_1)
# show result
summary(fit1)
```

```
##
## Call:
## lm(formula = final_price ~ odometer + age + new_cond + city,
##     data = modeling_data_1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7702.3 -1616.5  -244.9  1448.8 13720.3
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.493e+04  4.422e+02  33.769  < 2e-16 ***
## odometer            -3.593e-03  1.111e-03  -3.234  0.00134 **
## age                -7.044e+02  2.959e+01 -23.807  < 2e-16 ***
## new_condfair       -1.102e+02  7.139e+02  -0.154  0.87741
## new_condgood       -4.681e+02  3.513e+02  -1.333  0.18352
## new_condlike new    1.239e+03  4.087e+02   3.032  0.00262 **
## new_condused        2.750e+03  5.821e+02   4.725 3.39e-06 ***
## citychicago        -4.253e+02  5.056e+02  -0.841  0.40091
## citydenver         -1.907e+02  6.072e+02  -0.314  0.75362
## citylasvegas        8.721e+01  5.578e+02   0.156  0.87586
## citynyc            -1.245e+03  4.472e+02  -2.785  0.00565 **
## citysac             2.409e+02  4.448e+02   0.542  0.58845
## citysfbay           6.972e+02  4.505e+02   1.548  0.12264
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2452 on 337 degrees of freedom
## Multiple R-squared:  0.7777, Adjusted R-squared:  0.7698
## F-statistic: 98.26 on 12 and 337 DF,  p-value: < 2.2e-16
```

```
# draw diagnostic plot
par(mfrow=c(1,2))
plot(fit1,1, main='for toyota camry')
plot(fit1,2, main='for toyota camry')
```



From the plot, we can conclude that the reisidual plot showes 'linear' and Q-Q plot shows that this model fit well. Also, since we have $R^2 = 0.7777$ and p-value is quite small, this model is appropriate.

```
# show coefficients
fit1$coefficients
```

```
##       (Intercept)          odometer                  age     new_condfair       new_condgood
##     1.493419e+04    -3.592947e-03    -7.044127e+02    -1.102142e+02    -4.681329e+02
## new_condlike new     new_condused      citychicago       citydenver      citylasvegas
##     1.239033e+03     2.750119e+03    -4.252524e+02    -1.907300e+02     8.720631e+01
##          citynyc          citysac       citysfbay
##    -1.245369e+03     2.409073e+02     6.972042e+02
```

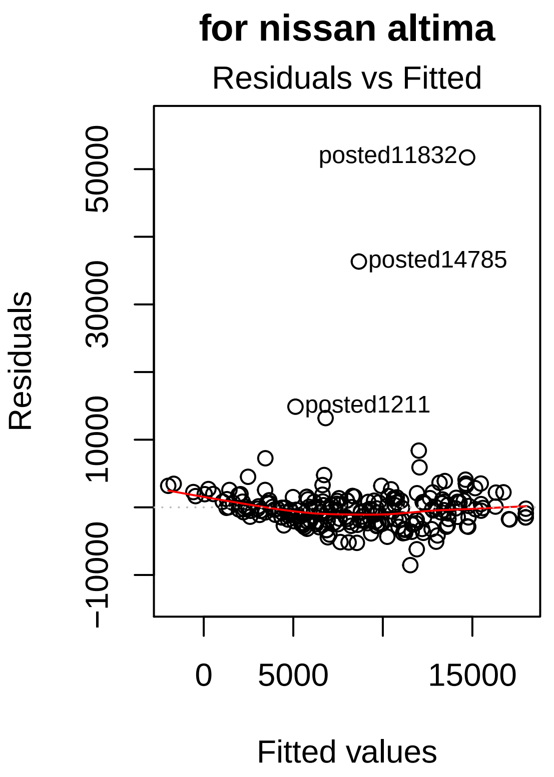
From the coefficient for each variables, odometter and age are negative, which inplied that the larger the odometer and age, the lower the price for 'toyota camry'. Also, when we see the condition, we find that the better the condition, the lower the price, which is not fit our common sence. May be this is the special aspect for toyota camry. Also, for the location, sfbay may increases the price the most, nyc may even lower the price.

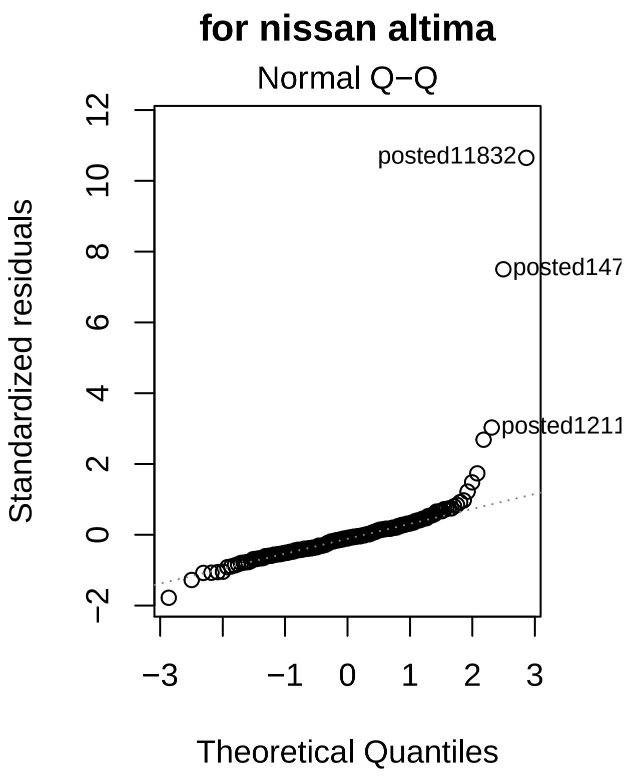
- 'nissan altima'

```r
# fit a linear model
fit2 = lm(final_price ~ odometer + age + new_cond + city, data= modeling_data_2)
# show result
summary(fit2)
```

```
##
## Call:
## lm(formula = final_price ~ odometer + age + new_cond + city,
##     data = modeling_data_2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -8545  -1922   -476    826  51725
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.483e+04  1.074e+03  13.809   <2e-16 ***
## odometer         -3.049e-03  2.974e-03  -1.025   0.3063
## age              -8.528e+02  8.924e+01  -9.555   <2e-16 ***
## new_condfair     -1.044e+03  2.349e+03  -0.445   0.6570
## new_condgood     -6.672e+01  8.942e+02  -0.075   0.9406
## new_condlike new  1.243e+03  8.806e+02   1.411   0.1595
## new_condsalvage  -8.524e+03  5.133e+03  -1.661   0.0982 .
## new_condused      1.397e+03  1.299e+03   1.075   0.2834
## citychicago       1.395e+03  1.141e+03   1.223   0.2225
## citydenver        2.730e+03  1.325e+03   2.060   0.0405 *
## citylasvegas      2.155e+02  1.250e+03   0.172   0.8633
## citynyc          -1.064e+03  9.929e+02  -1.071   0.2852
## citysac           3.095e+02  1.225e+03   0.253   0.8008
## citysfbay         6.313e+02  1.302e+03   0.485   0.6283
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4981 on 226 degrees of freedom
## Multiple R-squared:  0.4495, Adjusted R-squared:  0.4178
## F-statistic:  14.2 on 13 and 226 DF,  p-value: < 2.2e-16
```

```r
# draw diagnostic plot
par(mfrow=c(1,2))
plot(fit2,1, main='for nissan altima')
plot(fit2,2, main='for nissan altima')
```





From the plot, we can conclude that the reisidual plot showes 'linear' and Q-Q plot shows that this model fit well. However, there may exist some outliers which may have an impact on model fitting. Also, since we have $R^2 = 0.4495$ is relatively small. However, p-value is quite small, this may imply thatmodel is appropriate.

```
# show coefficients
fit2$coefficients
```

```
##      (Intercept)          odometer               age      new_condfair      new_condgood 
##     1.483150e+04     -3.049403e-03     -8.527610e+02     -1.044444e+03     -6.672116e+01 
## new_condlike new  new_condsalvage      new_condused       citychicago        citydenver 
##     1.242821e+03     -8.523962e+03      1.396594e+03      1.395260e+03      2.730034e+03 
##     citylasvegas           citynyc           citysac         citysfbay 
##     2.154804e+02     -1.063730e+03      3.094798e+02      6.312839e+02 
```

From the coefficient for each variables, odometter and age are negative, which inplied that the larger the odometer and age, the lower the price for 'nissan altima'. Also, when we see the condition, we find that 'used' condition may increase the price most, and 'salvage' condition may lower the price most, which is also not that fit our common sence. May be this is the special aspect for 'nissan altima'. Also, for the location, sfbay may increases the price the most, nyc may even lower the price.

These two models fit not bad. They can indeed reflect some useful infromation such as the lager the odometer and age, the lower the price. Also the location can also reveal where we can buy a cheaper car. If I'm a buyer or seller, I would like to use these two model to decide how to buy 'toyota camry' or 'nissan altima'.