STA141_Assignment4

Chad Pickering

Monday, November 16, 2015

Corresponded with: Janice Luong, Sierra Tevlin, Rico Lin, Hannah Kosinovsky, Ricky Safran, Fanny Chow, Ricky Tran

Resources: Handling and Processing Strings in R (G. Sanchez), Regular Expressions in R (Video Lecture - R. Peng), ggplot2 Guide (zevross.com - Beautiful Plotting in R), Stack Overflow, predict() function specifications, Regression diagnostics: testing the assumptions of linear regression (http://people.duke.edu/~rnau/testing.htm)

```
load("C:/Users/cpickering/Syncplicity Folders/ChadSync/STATISTICS/STA141/vehicles.rda")
```

Adjustments made in Assignment 1: The following are adjustments I made back in Assignment 1 Part 2 in order to correct outliers, combine conditions, etc. I start Assignment 4 with this adjusted dataset.

Year Adjustments: Two adjustments: A mistyped 2022 was changed to 2002; a 4 was identified as likely 2004.

```
# adjusting the year 2022; keystrokes similar to 2002
vposts[which(vposts$year == 2022), "year"] <- 2002

# adjusting to the year written in the description
vposts[which(vposts$year == 4), "year"] <- 2004
```

Age Adjustments: The age variable is created here. Adjustments were made after viewing the minimum and maximum ages. For correcting six of seven entries with the same value, "magic numbers" is the easiest way to identify single row indexes and change what is needed. As shown throughout, I understand how to avoid using "magic numbers", but this instance is convenient and is explained to alleviate confusion.

```
vposts$age <- 2016 - vposts$year

# -6 before, set to likely keystroke 2022 -> 2002
vposts[which(vposts$age == -6), "age"] <- 14
# 2012 before, because year was "4", set to year 2004, took difference
vposts[which(vposts$age == 2012), "age"] <- 12

# Used which(vposts$age == max(vposts$age, na.rm=TRUE)) here.
# 6 of 7 116 year old entries set to NAs; 27557 is the first post of seven
vposts[27901, "age"] <- NA
vposts[27902, "age"] <- NA
vposts[28058, "age"] <- NA
vposts[28059, "age"] <- NA
vposts[28100, "age"] <- NA
vposts[28373, "age"] <- NA
```

Price Adjustments: After viewing maximum price values, I decided to edit some entries due to a specific reason (commented).

```
# Used which(vposts$price == max(vposts$price, na.rm=TRUE)) here.
# set first of two identical outliers to higher of entry attempts
vposts[4741, "price"] <- 30000
# set second of two identical outliers to lower of entry attempts
vposts[4880, "price"] <- 6000
# set third large outlier to average of entry attempts
vposts[8140, "price"] <- 2750
# set fourth large outlier to NA; no evidence to an alternative price
vposts[13937, "price"] <- NA
# Row 7101 has the new reasonable maximum of 569500.
```

Odometer Adjustments: Here, I identified some anomalous odometer readings and set them all to NA; there was no evidence for alternative readings in the description or body.

```
# Ridiculously high odometer readings set to NA
vposts[which(vposts$odometer == 1234567890), "odometer"] <- NA
vposts[which(vposts$odometer == 99999999), "odometer"] <- NA
vposts[which(vposts$odometer == 16000000), "odometer"] <- NA
vposts[which(vposts$odometer == 16000000), "odometer"] <- NA
vposts[which(vposts$odometer == 9500000), "odometer"] <- NA
```

Condition Adjustments: To re-organize conditions, I examined each level with only one or few frequencies and reassigned them to more appropriate condition definitions that used broader terms. My determinations stemmed from analyzing odometer, description, and whatever factors held clear evidence.

```
comb_condition <- vposts$condition
excellent <- c("excellent", "superb original", "very good")
used <- c("used", "preowned", "carfax guarantee!!", "pre-owned", "0used",
          "complete parts car, blown engine", "front side damage", "hit and run :( gently",
          "honnda", "mint", "pre owned", "preownes", "rough but runs", "certified",
          "muscle car restore", "nice rolling restoration", "restoration", "restore", "restored")
needs_restore <- c("needs bodywork", "project", "needs restoration!", "needs total restore",
                   "needs work", "not running", "salvage", "rebuildable project",
                   "restoration project", "needs work/for parts", "needs restored",
                   "needs restore", "project car", "parts")
good <- c("good", "207,400", "ac/heater", "nice", "nice teuck")
comb_condition <- as.character(vposts$condition)
upd_excellent <- comb_condition %in% excellent
upd_used <- comb_condition %in% used
upd_needsrestore <- comb_condition %in% needs_restore
upd_good <- comb_condition %in% good
comb_condition[upd_excellent] <- "excellent"
comb_condition[upd_used] <- "used"
comb_condition[upd_needsrestore] <- "needs work"
comb_condition[upd_good] <- "good"
vposts$updcondition <- factor(comb_condition)
```

Regular expressions:

R Markdown cuts off some regular expressions below, so please find all regular expressions on a separate page attached to this report.

Preface. When determining optimal regular expressions to use, I followed a few self-generated guidelines so as to maximize efficiency and minimize unwanted data extraction. I determined that there are 5 possible outcomes to using a regular expression. Here I list them in order from most to least optimal, in my opinion. *I will refer to data/strings we want to extract as "good data", and by default, all other strings as "bad data".*

1. Most optimal: Extract essentially all "good data" possible; extract no "bad data".
2. Extract a majority, but not all, "good data"; extract no "bad data".
3. Extract essentially all "good data" but extract some "bad data" along with it.
4. Extract both "good data" and "bad data" approximately evenly.
5. Least optimal: Extract essentially no "good data" and mostly all "bad data".

I strongly feel that it is **only** better to extract more "good data" when it does not come at the expense of extracting more "bad data". The central concept in my above five categories is that scenario 2 is more optimal than scenario 3. If we are trying to find some relationships between variables or collect data for outside purposes, the extractions had better contain mostly, or most optimally all, correct information. I tried to follow this principle where I could throughout all of these exercises.

1. Extract price from "body" column. Extracting prices was a different kind of process than the following five exercises, in that there already exists a column that specifically shows the price reported and they can be compared to extracted prices. How prices are represented in the body column are extremely diverse because a person could report anything in free-form. Because not all strangely positioned prices could be anticipated, not all were extracted. If I tweaked the regular expression to extract extremely specific price formats, it would extract more "bad data" and would violate my preface concept.
It is regrettable that I had to make the dollar sign non-optional, because there are many cases where the seller did not use a dollar sign before a price. However, an optional dollar sign caught hundreds, if not thousands in some cases, of values that were not prices, and this violates my preface concept. I felt most confident when making a comfortable range of digits before and after an optional comma, and also considering the possibility of a price less than $1000. I also considered the case where the price is followed by the dollar sign, and I ensured that the dollar sign was not followed by other numbers. I confirmed through several iterations that all regular expressions used significantly minimize "bad data" extraction. I was able to extract **14573** prices. Because multiple prices were extracted from some rows, I determined that the largest mentioned is most likely the asking price. I then filtered out the maximum prices that ended up being larger than $570000 (the outlier threshold I defined in Assignment 1), and made them NA. I ended up matching about 75% of the extracted material with the corresponding value in the original price column. Find more explanations in the comments below for #1. Because #1 shares a majority of the same processes found in the following exercises, I demonstrated understanding through comments only once here in #1.

```
# Capture indexes in each body at which a match occurs (and corresponding lengths of matches)
prices_ind <- gregexpr("\\$[:space:]?[0-9]{0,3},?[0-9]{3}|\\$[:space:]?[0-9]{1,3}|[0-9]{0,3},?[0-9]{3}\

# Show actual extracted prices that correspond with indexes from prices_ind
prices_raw <- regmatches(vposts$body, prices_ind)

# Replace everything that isn't a number or space with nothing
prices_onlynum <- gsub("[^0-9 ]", "", prices_raw)

# Split string with possibly many elements e.g. "3500 4600 11800 8450"
# by " " so that each individual price found is separated
prices_split <- strsplit(prices_onlynum, " ")
```

```
# Convert class of all extracted elements to numeric
prices_final <- lapply(prices_split, as.numeric)

# Find the largest price for each row
prices_desired <- lapply(prices_final, max)

# Replace large values that are beyond Assignment 1 outlier threshold with NA
prices_high <- as.logical(prices_desired >= 570000)
prices_desired[prices_high] <- NA
prices_good <- as.numeric(prices_desired)

# If 0, set to NA - no such thing as price 0
prices_final <- as.numeric(gsub("^0$", "NA", prices_good))

# Total extracted:
sum(!is.na(prices_final))
```

```
## [1] 14573
```

```
prices_df <- as.data.frame(prices_final)

# True if extracted matches with existing column
price_check <- sapply(1:nrow(vposts), function(x) vposts$price[x] %in% prices_final[x])

# Success rate of extracted items to existing items - proportion matched
price_success <- table(vposts$price == prices_df)
price_success[[2]]/(price_success[[1]]+price_success[[2]])
```

```
## [1] 0.7453965
```

2. Extract VIN from "body" column. VINs, upon research, vary significantly, and specific parts of the string identify certain characteristics of a vehicle. Pre-1981, the VIN number was less than 17 digits long, so although they are always 17 digits long today, I had to consider all possible lengths when developing my regular expression. Both cases of the letters I, O, and Q are never used, and are excluded from the expression (although some people read a zero as an O and reported it that way, therefore excluding that VIN from among the extracted), and I considered the most popular options of what I call "identifiers" before the actual VIN number; the character string could be preceded by a space, a colon, another exotic use of punctuation, or none of the above.
The term "VIN" could also be represented in many ways, but I did not consider that the VIN number could be merely dropped into the ocean of verbage without context, so those have not been extracted. If I made the "identifiers" optional, then I would have violated my preface concept because I would have grabbed some URL suffixes and other reported codes that are irrelevant to this assignment. Ultimately, I found **6588** VINs. I included two examples of pre-1981 VINs to confirm their direct correlation with their build year.

```
# Reg. ex., extract all matches, pull out second element that contains wanted piece
vin_indices <- regexec("[Vv][Ii][Nn]\\:*[[:space:][:punct:]]*([A-HJ-NPR-Za-hj-npr-z0-9]{11,17})", vposts
vin_match <- regmatches(vposts$body, vin_indices)
vin_output <- sapply(vin_match, function(x) x[2])

# Create vposts$vin, add vins to corr. rows
vposts$vin <- vin_output
sum(!is.na(vin_output))
```

```
## [1] 6588
```

```
head(vin_output[which(!is.na(vin_output))], 30)
```

```
##  [1] "2G1FT1EW1C9106920" "2GNFLNEK7D6324351" "1N4AL3AP5DN569028"
##  [4] "JNKCY01F29M851648" "JN1CV6AR2DM763007" "2HNYD2H20CH537485"
##  [7] "JTDBT4K31A1368794" "5J8TB1H28CA000511" "1G1PC5SB1E7464908"
## [10] "JTHCK262695031859" "4T1BF1FK4DU269358" "JHMZF1D48ES002209"
## [13] "WBAVD53548A285108" "JN8AF5MV7DT211274" "1C4RDJAG8DC693420"
## [16] "4T1BF1FK2CU592080" "4T3ZA3BBXCU060002" "JN8AF5MV2ET364209"
## [19] "2T1BU4EE7CC880942" "5FNYF4H56CB028710" "3FA6P0HR4DR305750"
## [22] "2T1BU4EE7CC880942" "JN8AS5MV7CW366271" "4T1BE32K45U431320"
## [25] "1C3CCBBG9DN623897" "JM1BL1TG1D1814382" "1G1ZC5E03CF344114"
## [28] "1ZVBP8AM8E5313888" "JH4CU2F4XCC010060" "5YFBU4EE2DP121617"
```

```
# Example of <17 character VIN (pre-1981):
lessthan17_ex1 <- which(vin_output == '2U87W9L188720')
vposts$body[lessthan17_ex1]
```

```
## [1] "\n        1979 Pontiac Firebird Formula - $7000\nVIN: 2U87W9L188720\nEngine: 301 cu in (4.9 L) I
```

```
lessthan17_ex2 <- which(vin_output == '1D37U7B428570')[4]
vposts$body[lessthan17_ex2]
```

```
## [1] "\n        1977 Chevrolet Malibu Classic - $3500\nhttp://www.romanchariotcars.com/1977-Chevrolet-
```

3. Extract phone numbers from "body" column. When extracting phone numbers, I considered numbers that had been formatted in the most popular ways, then integrated some less frequent choices. Fortunately, there is not much variation; most have parentheses around the area code or dashes/dots between groups. Eventually I included exotic punctuation choices, but I ended up not including entries that spelled out one or more digits that attempt to avoid parsing. Although this last category was skipped because there were so few of them and the surrounding data was so varied, **16232** phone numbers were found. I then used gsub() to delete out all punctuation choices and spaces to make all phone numbers merely a string of ten digits for the vposts$phone column.

```
phone_indices <- regexec("(\\(*[0-9]{3}\\)*[[:space:][:punct:]*][0-9]{3}[[:space:][:punct:]*][0-9]{4})"
phone_match <- regmatches(vposts$body, phone_indices)
phone_output <- sapply(phone_match, function(x) x[2])

# Format numbers into consistent 10 digit string
format_phone <- gsub("[^0-9]", "", phone_output)

vposts$phone <- format_phone
sum(!is.na(format_phone))
```

```
## [1] 16232
```

```
# Examples of extracted phone numbers
head(format_phone[which(!is.na(format_phone))], 30)
```

```
##  [1] "5082051046" "5082051046" "5082051046" "5082051046" "5082051046"
##  [6] "5082051046" "5082134680" "5082051046" "5082051046" "5082134680"
## [11] "5082051046" "5082051046" "5082134680" "5082051046" "5082051046"
## [16] "5082326214" "5082051046" "5082051046" "5082134680" "5082051046"
## [21] "6177998738" "5082326214" "6177998738" "9785347666" "5082051046"
## [26] "5082051046" "9782510888" "5082134680" "5082051046" "8773997023"
```

4. Extract email addresses from "body" column. There were not many email addresses found, most likely because sellers/dealers wish to avoid spam or a large influx of irrelevant messages. This being said, I also went ahead and found as many website URLs as possible.

Email:

I attempted to capture as many variations of punctuation use as possible in email usernames, which is particularly difficult because periods are commonly used, which in the regular expression can be confused as the "." required in all email addresses. Additionally, I did limit the number of possible characters after the required "." to 3. Ultimately, I extracted **94** email addresses. 100% of them were legitimate upon checking.

```
email_indices <- regexec("(([A-Za-z0-9_?-]+)@([0-9a-z\\.-]+)\\.([a-z\\.]{3}))", vposts$body)
email_match <- regmatches(vposts$body, email_indices)
email_output <- sapply(email_match, function(x) x[2])

vposts$email <- email_output
sum(!is.na(email_output))
```

```
## [1] 94
```

```
# Example of email output
head(email_output[which(!is.na(email_output))], 30)
```

```
##  [1] "imports4sale@yahoo.com"      "sales@rt28motors.com"
##  [3] "bylosauto@aol.com"           "bylosauto@aol.com"
##  [5] "Amber44@charter.net"         "smdwoods@aol.com"
##  [7] "imports4sale@yahoo.com"      "imports4sale@yahoo.com"
##  [9] "imports4sale@yahoo.com"      "imports4sale@yahoo.com"
## [11] "imports4sale@yahoo.com"      "imports4sale@yahoo.com"
## [13] "imports4sale@yahoo.com"      "imports4sale@yahoo.com"
## [15] "imports4sale@yahoo.com"      "imports4sale@yahoo.com"
## [17] "judi45p@gmail.com"           "clarson71@hotmail.com"
## [19] "catherineburgos1q@gmail.com" "sarabaron69@gmail.com"
## [21] "Patriciae285@gmail.com"      "jessica38reyes@gmail.com"
## [23] "lara33powell@gmail.com"      "lara33powell@gmail.com"
## [25] "sales@chicagoautoplace.com"  "sales@chicagoautoplace.com"
## [27] "gmmantrb@yahoo.com"          "sales@chicagoautoplace.com"
## [29] "sales@chicagoautoplace.com"  "sales@chicagoautoplace.com"
```

Websites:

Extracting websites was probably the most difficult process of the bunch because many websites have no space after its true endpoint - they are just dropped into the chaotic ocean of text. For a vast majority of dealers, the portion of the website after the .com bit is the unique identifier of the posting itself (so it is most critical to extract) - that section was the hardest to cleanly identify and extract using a regular expression. However, after iterating several times, I concluded that the most common item I wrongly extracted at the end of the string was "Address:", so I just used gsub() to remove them. I ended up getting **12495** website URLs in some form.

```
website_indices <- regexec("[^\\@a-z]((https?:\\/\\/)?([0-9a-z\\-\\/]{1,}\\.){1,}(com|org|net)[^A-Za-z0-
website_match <- regmatches(vposts$body, website_indices)
website_raw <- sapply(website_match, function(x) x[2])
website_output <- gsub("Address:", "", website_raw)

# Example of website output
head(website_output[which(!is.na(website_output))], 30)
```

```
##  [1] "automaxpreowned.com/"         "automaxpreowned.com/"
##  [3] "automaxpreowned.com/"         "automaxpreowned.com/"
##  [5] "automaxpreowned.com/"         "automaxpreowned.com/"
##  [7] "automaxpreowned.com/"         "automaxpreowned.com/"
##  [9] "automaxpreowned.com/"         "automaxpreowned.com/"
## [11] "automaxpreowned.com/"         "automaxpreowned.com/"
## [13] "www.fafama.com/"              "automaxpreowned.com/"
## [15] "automaxpreowned.com/"         "automaxpreowned.com/"
## [17] "www.fafama.com/"              "www.airportsales.com "
## [19] "automaxpreowned.com/"         "automaxpreowned.com/"
## [21] "automaxpreowned.com/"         "sale.com/79078594.html"
## [23] "sale.com/78738592.html"       "www.fafama.com/"
## [25] "automaxpreowned.com/"         "http://www.ifoundmycar.com/"
## [27] "http://www.ifoundmycar.com/" "http://www.ifoundmycar.com/"
## [29] "http://www.ifoundmycar.com/" "http://www.ifoundmycar.com/"
```

```
vposts$website <- website_output
sum(!is.na(website_output))
```

```
## [1] 12495
```

5. Extract year from "body" column, compare with vposts$year. I developed a rather straightforward regular expression that captures any year from 1900 to 2016. Unlike price, the body is more likely to contain only one mentioning of the year, so it is easier to match the true year of the vehicle rather than some other mention of year in some irrelevant description. I made sure that the year I extracted matched with the corresponding year; with the current regular expression and process, the success rate is about 98.5%. Overall, I extracted **29433** years.

```
year_indices <- regexec("(19[0-9]{2}|200[0-9]|201[0-6])", vposts$description)
year_match <- regmatches(vposts$description, year_indices)
year_output <- sapply(year_match, function(x) x[2])

vposts$ext_year <- as.numeric(year_output)
sum(!is.na(year_output))
```

```
## [1] 29433
```

```
# Example of year output
head(year_output[which(!is.na(year_output))], 50)
```

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

```
## [11] "2013" "2014" "2008" "2013" "2013" "2012" "2012" "2014" "2012" "2012"
## [21] "2008" "2013" "2012" "2008" "2012" "2012" "2005" "2013" "2013" "2012"
## [31] "2014" "2012" "2013" "2010" "2014" "2013" "2008" "2012" "2011" "2013"
## [41] "2012" "2012" "2012" "2012" "2014" "2012" "2014" "2013" "2012" "2012"
```

```
comp_year <- table(vposts$year == vposts$ext_year)

# Proportion of extracted years agreed with original years column
comp_year[[2]]/(comp_year[[1]]+comp_year[[2]])
```

```
## [1] 0.9854585
```

6. Determine the model of the car. Upon reviewing the different columns in vposts that include the car model, I found that the "header" column is the least crowded, so finding the model should be more straightforward. (Of course, if there was a column that was more crowded but had consistent surrounding identifiers to models that would ensure a higher success rate, I would surely use that.) I first split each word found in "header" by a space, then looked for the pattern of any maker in the "maker" column. I made sure to use agrep() here, and it is confirmed to find at least a few more members of each category at the end compared to if grep() was used - apparently some model names were misspelled. Then, using the corresponding index of where a maker was matched, the next function adds 1 to find the model; if a maker is not recognized in the full string of words from the particular "header" entry in vposts, then "NA" is returned, because I assume here that model follows immediately after maker name. This process also assumes that models are only one word long, which is not always true. Time permitting, this was the most efficient way to extract a vast majority of members of each model, especially the popular ones, two of which I use for my statistical models. Also, to avoid having the same models split up by choice of capitalization, I normalized all extractions with tolower(). I found **30516** models that were not missing or dubbed "NA".

```
# Split each word in header by a space
header_split <- sapply(vposts$header, function(x) strsplit(x," "))

# Look for pattern (maker), find header_split element that matches
match_header <- lapply(1:nrow(vposts),
                   function(x) agrep(vposts$maker[x], header_split[[x]], ignore.case=TRUE))

# Finds model by adding 1 to match_header index; if maker is not recognized, set to NA
find_model <- function(x)
{
  if (length(match_header[[x]])!=0)
  {
    header_split[[x]][min(match_header[[x]])+1]
  }
  else return(NA)
}

models_final <- tolower(lapply(1:nrow(vposts), find_model))
sort(table(unlist(models_final)), decreasing=TRUE)[1:10] # top 10 models
```

```
##
##      na   civic  accord   grand   camry  altima corolla mustang  maxima
##    2544     854     842     785     754     665     537     431     384
##     ram
##     383
```

```r
# Total models found
sum(!is.na(models_final)) - table(models_final=="na")[[2]]
```

```
## [1] 30516
```

```r
#Make the models a column in vposts
vposts$model <- models_final
```

Statistical Models: Here, I am interested in exploring the relationship between price, odometer, age, and condition for two separate models of vehicle, the Toyota Corolla and the Honda Civic, and to see if the city they were posted to makes any difference. Using the ggplot2 package, I was able to plot price on the y axis as the response variable and odometer on the x axis; age is represented by the color gradient, and condition is represented by the size of the dot. The linear model I have below works roughly in some conditions, but should be changed to an exponential decay model for more accurate results (if the method to do so were known - see unresolved Piazza post @1032). The plots were generated from the Corolla and Civic subsets, but if any of the three columns (condition, age, or odometer) had an NA value, then that particular observation is not plotted. I did consider substituting the mean or median of the existing observations of a particular variable for the missing data, but this would create a drastic bias towards the mean or median of all three numeric variables. Find a more thorough analysis in the modeling section below, including why I chose to disregard some observations where odometer and price fell below a certain value.

Let it be known that I have not taken STA108 yet, and I am analyzing the remaining part of this assignment with the knowledge I have from other classes.

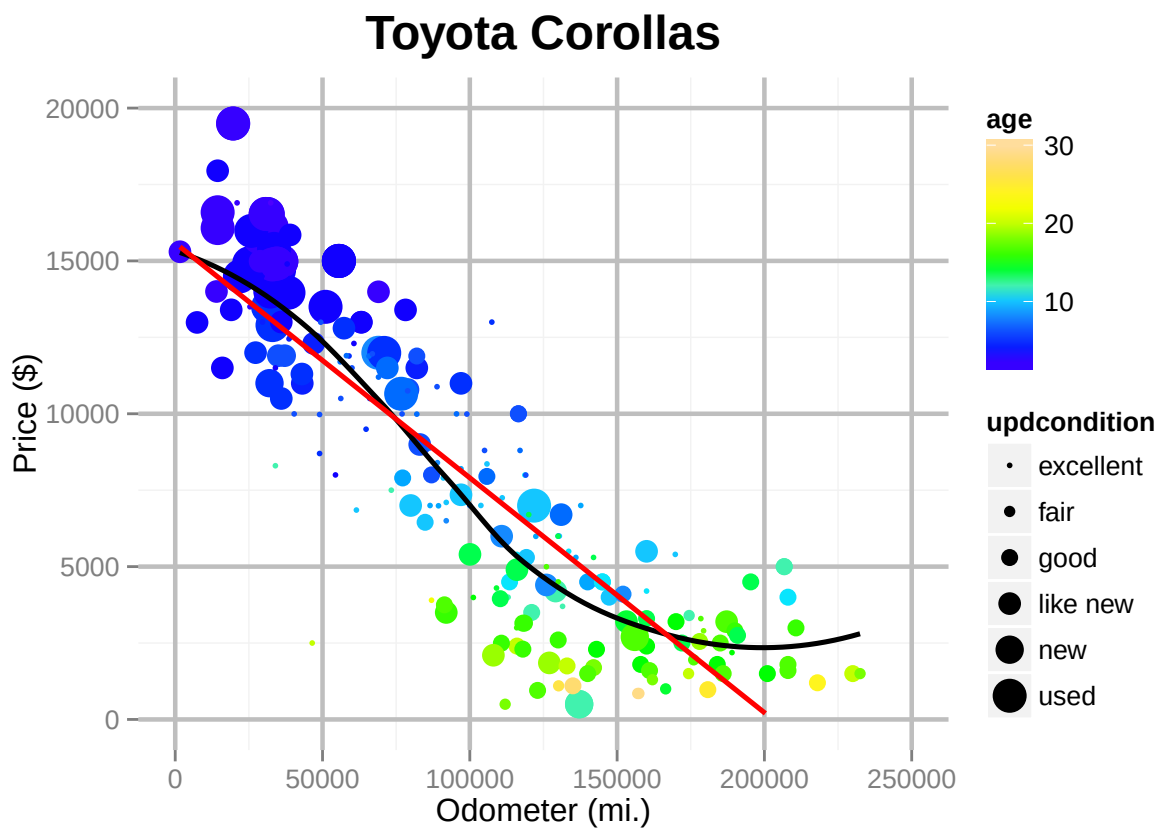
```r
library(ggplot2)

# Eliminate scientific notation
options(scipen = 999)

# Just Corollas
all_corolla <- vposts[which(vposts$model == "corolla"),]
all_corolla <- all_corolla[!all_corolla$odometer<=1000 &
                             !all_corolla$odometer>=1000000 & !all_corolla$price<=100 &
                             !all_corolla$price >= 200000 & !all_corolla$age >= 60,] #justification bel

# Just Civics
all_civic <- vposts[which(vposts$model == "civic"),]
all_civic <- all_civic[!all_civic$odometer<=1000 & !all_civic$price<=100,] #justification below

# Make conditions factors in this order...
vposts$updcondition <- factor(vposts$updcondition,
      levels = c("new", "like new", "excellent", "good", "fair", "used", "needs work"), ordered = TRUE)
```

Toyota Corollas: In both the linear and non-linear cases, the intercept β_0 occurs at just above \$15000; in a true exponential decay model, I would expect it to start at around \$20000 at age 0, odometer 0, and with condition "new", considering the price of an actual new Toyota Corolla in recent years is around that much. It is regrettable that this model does not consider the price of a legitimate new Toyota Corolla that could be purchased at a car dealership. An optimal model for the price of this car would consider the circumstances of the vehicle when it has age 0, odometer 0, and condition "new" - this event never occurs when the vehicle is NOT being sold at a dealership, at least in my experience. Overall, the observations show that as odometer increases, price decreases at an exponential rate (see model section below), and as odometer increases, age increases, as expected, and shown clearly by the color gradient created, with greens and yellows representing

older cars. As I stated in my analysis for Assignment 1, condition is subjective, unique to each poster's opinion of what constitutes a car in "good" condition versus "excellent" condition, and so on. As a direct result of this, I can discern only one pattern in the size of the dots - roughly, cars with a high price and low odometer tend to be cars with conditions like "excellent" and "like new", as expected. However, this trend is not immediately clear like age is (color is easier to visually understand than size). In fact, with Corollas, there is a strange cluster of cars that are labelled as "used" and "needs work" with relatively low odometers and with asking prices of near $15000. This anomaly will be discussed in the cities section below.

```
ggplot(all_corolla, aes(odometer, price, col = age))+
  geom_point(aes(size = updcondition))+
  labs(x = "Odometer (mi.)", y = "Price ($)", title = "Toyota Corollas")+
  xlim(c(0, 250000))+
  ylim(c(0, 20000))+
  theme(plot.title = element_text(size = 18, face = "bold", vjust = 1.5),
        panel.grid.major = element_line(colour = "grey", size = .8),
        panel.background = element_rect(fill = 'white'))+
  scale_color_gradientn(colours = topo.colors(16))+
  geom_smooth(method = "auto", se = FALSE, color = "black", size = 0.9)+
  geom_smooth(method = "lm", se = FALSE, color = "red", size = 0.9)
```

```
## geom_smooth: method="auto" and size of largest group is <1000, so using loess. Use 'method = x' to cl
```

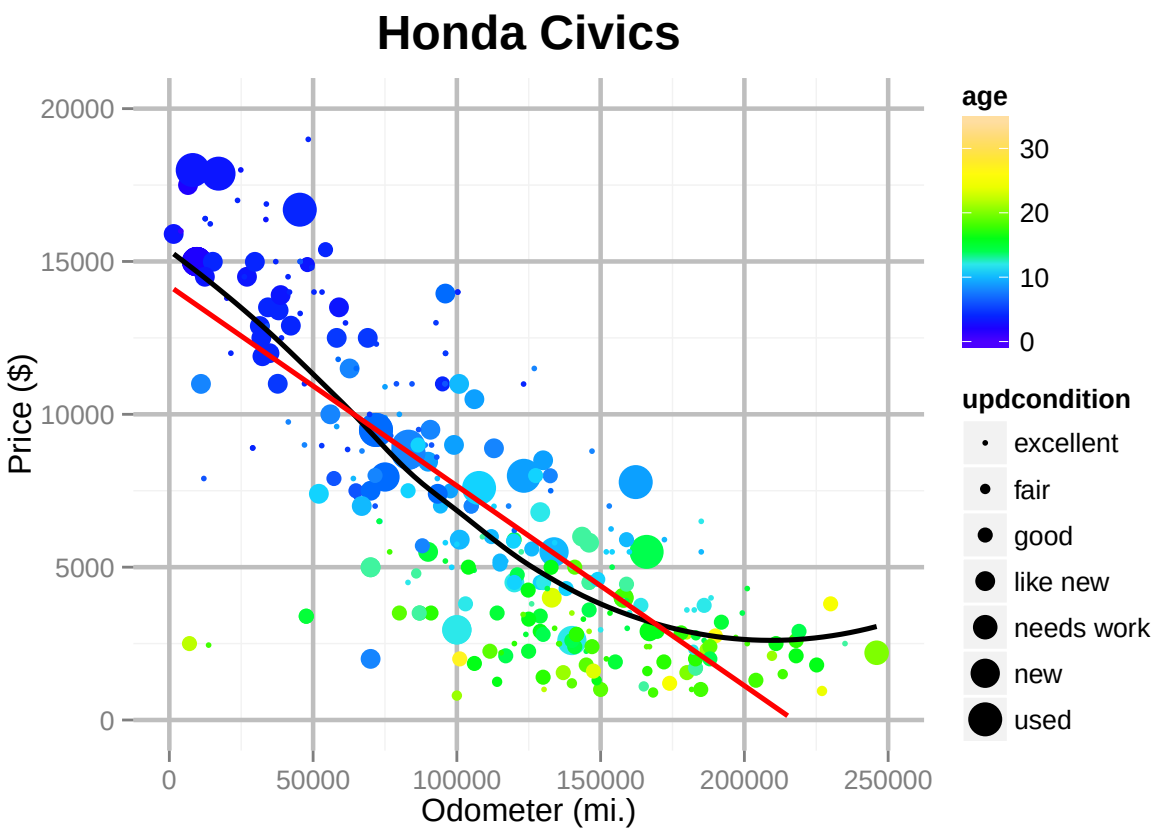


Honda Civics: Honda Civics have all of the basic relationships between variables that Toyota Corollas do, like between odometer and price (see above). I added the non-linear trendline here to demonstrate what I

would want in a model for this data. The β_0 element here is closer to describing the approximate price for a vehicle with odometer 0, per age and per condition (the categorical condition vertically shifts the model). The trendline descends and gradually levels out (in the true exponential decay model, the line would constantly descend, unlike the tail end of what is pictured), and the predicted value of the response variable (price) will never reach 0, consistent with monetary amounts. In the modeling section below, I explain why exactly this exponential decay model would fit this data so much better.
Among the few curiosities in the plot are the three young yet "used" cars that have less than 50000 miles on them and are going for top dollar - these are outliers in terms of what one would expect a Civic of that variable combination to be priced at. Looking at both Corollas and especially Civics, we can see that almost all vehicles priced at less than $5000 are approximately 10 or more years old. The condition anomalies seen in Corollas are not seen in Civics; it is easier to see the expected trend that the more miles are on a car and the older it is, the condition tends to worsen. Again, this is subjective; however, it is much more obvious to see on this plot.

```
ggplot(all_civic, aes(odometer, price, col = age))+
  geom_point(aes(size = updcondition))+
  labs(x = "Odometer (mi.)", y = "Price ($)", title = "Honda Civics")+
  xlim(c(0, 250000))+
  ylim(c(0, 20000))+
  theme(plot.title = element_text(size = 18, face = "bold", vjust = 1.5),
        panel.grid.major = element_line(colour = "grey", size = .8),
        panel.background = element_rect(fill = 'white'))+
  scale_color_gradientn(colours = topo.colors(16))+
  geom_smooth(method = "auto", se = FALSE, color = "black", size = 0.9)+
  geom_smooth(method = "lm", se = FALSE, color = "red", size = 0.9)
```

```
## geom_smooth: method="auto" and size of largest group is <1000, so using loess. Use 'method = x' to cl
```



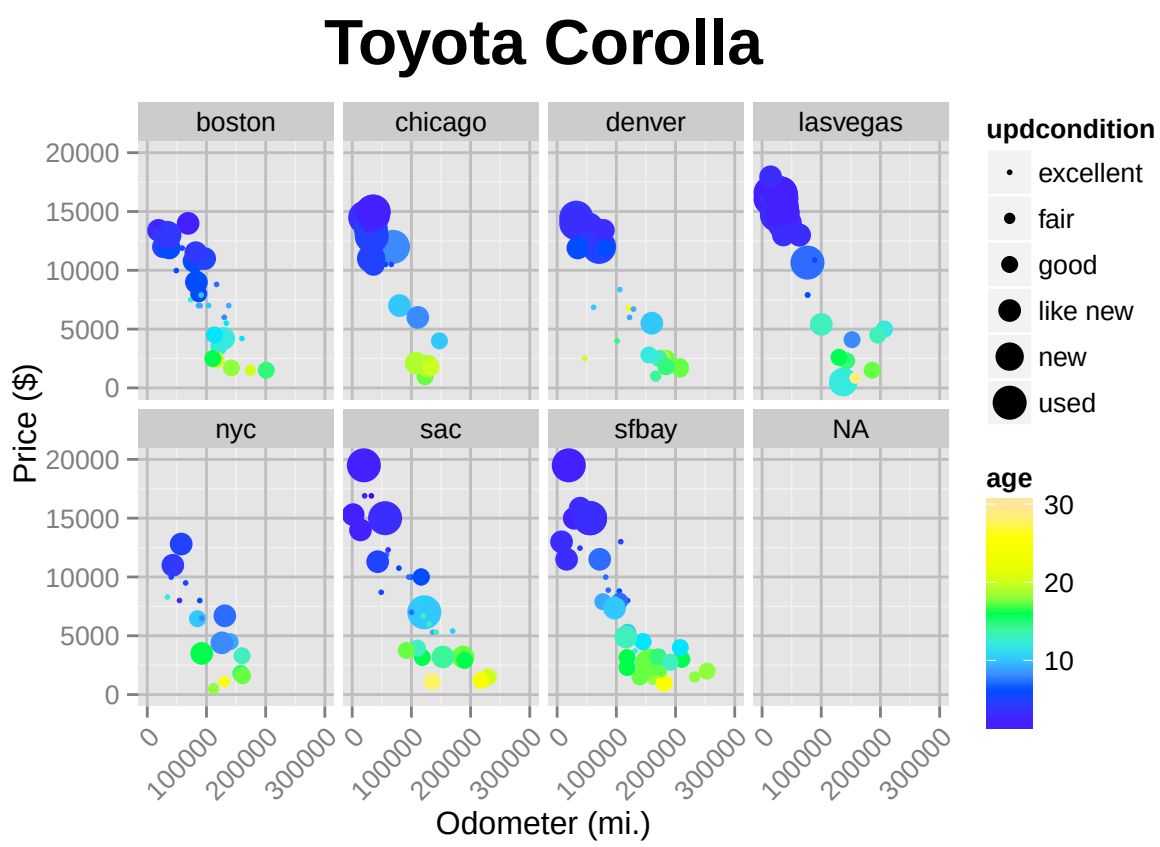
Does the city have an effect? When splitting the plots on location, anomalies in the data and differences in frequency and variance become apparent. However, it seems as if city has an insignificant role in the effect on price and what the relationship between the other four variables looks like.

In terms of Corollas, all cities have the same general pattern; there is an even amount of Corollas in each city so analyzing its effects on car price should not be biased in any way (e.g. if one city only had 5 observations whereas another had 150). It seems as though Corollas are pricier on the west coast and have a greater variance as opposed to the east coast cities. The particular cluster of anomalous "needs work" vehicles that were in the general Corolla model seem to mostly be all in Las Vegas. I would expect the reasoning to be because of the economic environment - there are many pawn shops and similar establishments there to promote quick sells of upgraded vehicles. These could also have been misclassified to begin with.

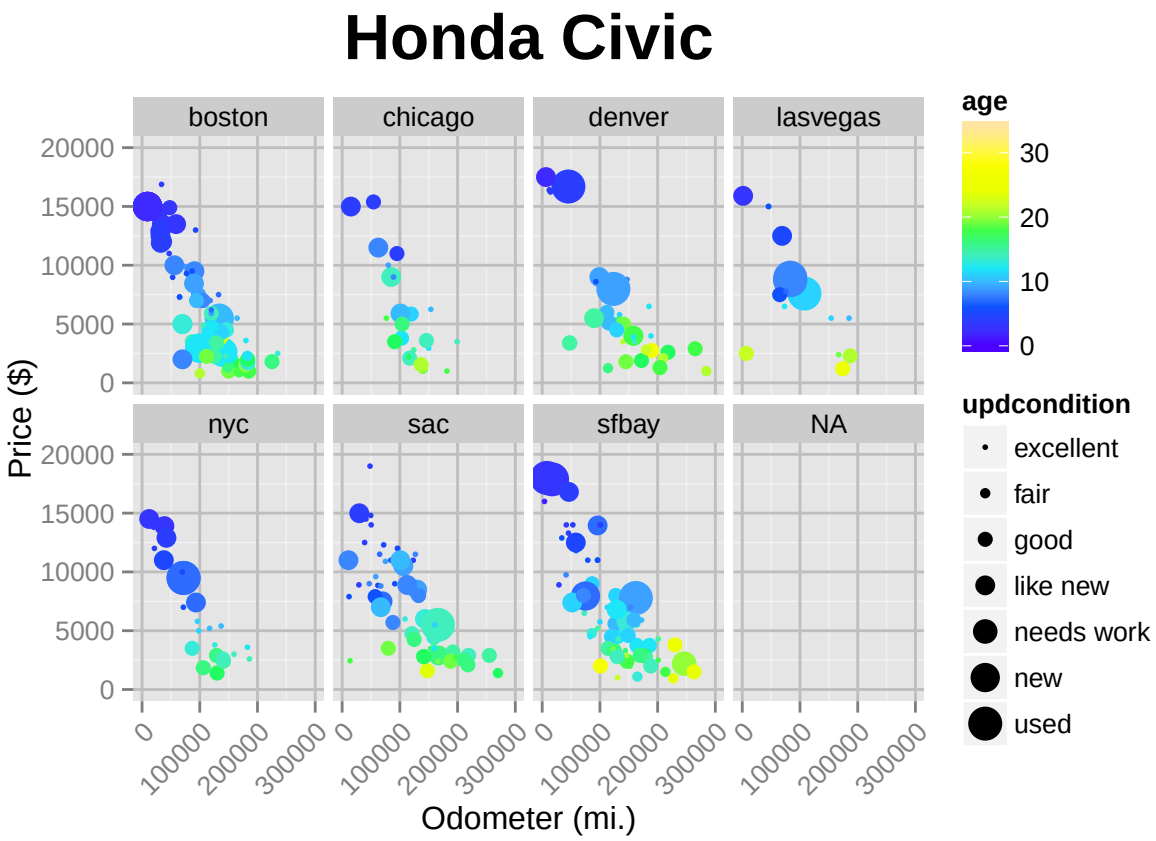
Civics have a greater difference in frequency, but the variance of price looks rather similar across all cities. Boston, Sacramento, and San Francisco all have healthy looking exponential decay distributions happening; this is what I would expect if more data were scraped from more databases/websites and the "sample size" for each city was larger. Chicago and Las Vegas look particularly weak in this respect. Lastly, Civics tend to have a wider range of odometer readings at all price values, whether they are low or high. Splitting the comprehensive model up by city make certain flaws of the source of all of the vposts data more obvious. For example, the two "used" Corollas that are going for $15000 and $20000 respectively in both Sacramento and San Francisco are the same two vehicles. The fact that the same vehicle post with the same information was included twice suggest that this same behavior continues throughout the remainder of the data. Duplicate posts should be filtered out to reduce bias.

```
ggplot(all_corolla, aes(odometer, price, col = age))+
  geom_point(aes(size = updcondition))+
  labs(x = "Odometer (mi.)", y = "Price ($)",
       title = "Toyota Corolla")+
```

```
  theme(plot.title = element_text(size = 24, face = "bold", vjust = 2),
        panel.grid.major = element_line(colour = "gray", size = .5),
        axis.text.x = element_text(angle = 45, hjust = 1))+
  scale_color_gradientn(colours = topo.colors(7))+
  scale_x_continuous(limits=c(0, 300000))+
  scale_y_continuous(limits=c(0, 20000), breaks=seq(0, 20000, 5000))+
  facet_wrap(~ city, nrow = 2, ncol = 4)
```




```
ggplot(all_civic, aes(odometer, price, col = age))+
  geom_point(aes(size = updcondition))+
  labs(x = "Odometer (mi.)", y = "Price ($)",
       title = "Honda Civic")+
  theme(plot.title = element_text(size = 24, face = "bold", vjust = 2),
        panel.grid.major = element_line(colour = "gray", size = .5),
        axis.text.x = element_text(angle = 45, hjust = 1))+
  scale_color_gradientn(colours = topo.colors(7))+
  scale_x_continuous(limits=c(0, 300000))+
  scale_y_continuous(limits=c(0, 20000), breaks=seq(0, 20000, 5000))+
  facet_wrap(~ city, nrow = 2, ncol = 4)
```



The model itself: I first approached this part of the assignment from the naive point of view that the relationships were in the form of a strict linear model. This method, in my view, is completely inappropriate because upon looking at the behavior of the two plots, and an analogous plot for the entire vposts dataset, it is clear that an exponential decay model will fit the data much better, at least in terms of the relationship between odometer and price. Instead of a linear model like $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$, where β_0 is the intercept, the β_1 term is the age of the car, etc., $y = \beta_0 + \beta_1 e^{kx_1} + \beta_2 x_2 + \beta_3 x_3 ...$ would take into consideration the swooping downward trajectory (odometer vs. price), characteristic of exponential relationships, and the fact that price is strictly non-negative. The latter is an extremely critical point - the linear model ignores this fact, and if one extrapolates well past 200000 miles, a negative price would be predicted, which is impossible. In fact, at times, interpolating with certain age/condition combinations past 150000 miles yields a negative predicted price. Clearly this is not the correct model to be using.

To statistically test the assumptions for linearity, the following will need to be done: see if variables interact with each other/are correlated using cor(); test for normality of error terms using a normal quantile plot; test for linearity and homoscedasticity (constant variance of the error terms) using plot of residuals versus predicted values. This is beyond the scope of this assignment and beyond my skill level at the current time. I would not use this linear model in any circumstances - buying a car, selling a car, research, etc. I would look to develop an exponential decay model when I understand how to do so. Beyond that, to develop this alternative model, I would suggest collecting data from new car lots to establish a strong β_0 that represents what price a car with age 0, odometer 0, and condition "new" would go for, then increase sample size for all remaining used car database extractions.

Finally, I would like to discuss why I subsetted my Corolla and Civic data how I did above to generate both the plots and the models. These decisions were NOT made based on mere convenience. I was getting r^2 values around 0.02 for Corollas before I adjusted my subsetted data conditions. I subsetted out extreme outlier events that would never happen logically, and the data point(s) that were skewing my coefficient of determination were removed from the subset. Also, for both models of car, I chose to introduce two consistent

conditions to my subset to filter out the vehicles that had prices so low that they could be only parts or odometer readings so low AND the corresponding condition of the vehicle was not specified as "new" that the event could not logically happen.

Extra relevant information:

r^2 values generated: Corolla - 0.9086; Civic - 0.7794. A logarithmic transformation of one numeric predictor variable increases the r^2 value. Interpretation example: Approximately 90.9% of the variation in the response variable price can be explained by the variation in the predictor variables for Toyota Corollas. This would look like strong evidence of linearity if the other assumptions were not tested and the actual plot was not observed - the linear trendline suggests prices can be negative and the price of the car falls in a linear fashion throughout its lifetime - this is false.

```
# Subsetted data frames of just columns used in predictions
# corolla_predvars <- all_corolla[, c("age", "odometer", "condition")]
# civic_predvars <- all_civic[, c("age", "odometer", "condition")]

# Function to predict price from age, odometer, condition, and city
predict_model <- function(a, odom, cond, ci)
{
  results <- c()

  predictor_df <- data.frame(age = c(a),
                             odometer = c(odom),
                             updcondition = c(cond),
                             city = c(ci))

  lmprice_corolla <- lm(price ~ age + odometer + updcondition + city, data = all_corolla)
  predict_corolla <- predict(lmprice_corolla, predictor_df)
  results[1] <- predict_corolla

  lmprice_civic <- lm(price ~ age + odometer + updcondition + city, data = all_civic)
  predict_civic <- predict(lmprice_civic, predictor_df)
  results[2] <- predict_civic
  results

  # conditions: "needs work", "used", "fair", "good", "excellent", "like new", "new"
}

# Not reasonable
predict_model(0, 0, "new", "sac")
```

```
## [1] 13567.60 15427.54
```

```
# Makes no sense that used would be significantly higher than fair
predict_model(5, 80000, "fair", "sac")
```

```
## [1] 11061.55 10630.24
```

```
predict_model(5, 80000, "used", "sac")
```

```
## [1] 13760.51 12281.48
```