Hannah Kosinovsky
Section: 8 am Olson
Worked with Richard Safran, Chad Pickering, Janice Luong,
Sierra Tevlin, Rico Lin, Ricky Tran, Fanny Chow
9/11/15

Assignment 4

Part 1: Extracting data with regular expressions:

1. To extract the price I used the regular expression:

'[$][0-9]{0,3}[:space:]?\\,?[0-9]{3}'

I looked at several body listings and saw that the price started with a literal dollar sign. Then I wanted to extract zero to three characters that could be digits 0-9. Then there was sometimes a space and sometimes not so I put a question mark after it to signify the options. Then I signified that I wanted to find a literal comma, but it might not be there which is why I put a question mark at the end. Then, this would be followed by three characters zero to nine.

Using this regular expression I got 14125 matches in the body, which is a satisfactory amount since I cannot account for every variation.

Then in order to compare the prices extracted from the body to those in the price column, I need to have only number prices without the dollar sign and so on. I accomplished this using the gsub function and took out everything that wasn't a number and replaced it with empty space. Then I split these by the empty space and made them into a numeric class.

Now I was able to compare the prices in the price column to the prices in the body. Then the success rate would simply be all the matches divided by the length of vposts.

The success rate I got is roughly 77.78%

2. To extract the vin number I used the regular expression:

"[Vv][Ii][Nn]\\:[[:space:]]*([A-HJ-NPR-Za-hj-npr-z0-9]{11}[0-9]{6})"

In the body I noticed that most VIN numbers began with vin, a colon, and a space, followed by the actual number, so I included this in my regular expression but only put parenthesis around what would be the actual vin number that I want to extract. I accounted for capitalized and lower case letters when looking for "vin". Then, for the vin number, I knew from looking at vin number information online (autocheck.com), that the vin number is 17 characters long, and there are no "I" and "O" letters in vin numbers, so I did not include those as options. Otherwise I made

the regular expression look for digits and letters since vin numbers are a mixture of both.

One problem I had was that on autocheck.com, it said that the last six digits of the vin number should be numbers, and not letters, but when I looked in the body I noticed many vin numbers had letters included in the last six characters of many vins in the body. I looked at other sources like on http://www.edmunds.com/driving-tips/making-sense-of-your-vin.html, and http://www.cdxetextbook.com/safetyInfo/vehInfo/information/decodevin.html and these sources confirmed that the last six characters should be digits, so I made the assumption that the vin numbers with letters in the last 6 spaces of the vin number were typos and could be disregarded, and kept the constraints of digits in the last six places in my regular expression.

This regular expression was able to extract 5125 observations.

So my success rate of finding vin numbers in the body was 14.78%

3. I used the regular expression:

"\\(?([[:space:][:punct:]]?([0-9]|zero|two|three|four|five|six|seven|eight|nine)){3}\\)?([[:space:][:punct:]]?([0-9]|zero|two|three|four|five|six|seven|eight|nine)){7}"

To extract phone numbers. I used both digit numbers and written out numbers because I noticed in the body some people wrote out the numbers. This was able to capture more phone numbers total than just looking for digit characters. I knew that usually it was an area code followed by a space or punctuation and then the remaining seven digits so I accounted for this format in the expression. Some people put parenthesis around the area code, but others didn't, so I accounted for this by using a question mark after the parenthesis. I also put a question mark after the space and puntuation because some people used only one or neither after the area code.

I was able to get 17419 phone numbers in this way. This translates to a 51.04% success rate at matching phone numbers.

4. To get emails I used the regular expression:

"[[a-z][a-z0-9\\._\\-]*\\@[A-Za-z0-9_\\-]{1,}\\.[a-z0-9]{2,}"

but it only matched 103 emails. When I reviewed several postings I noticed they don't always include an email, probably because people don't want spam. Then I decided to include websites instead because there were many website listings in the body. A challenge I encountered was that many of the websites were identical, but at a closer glance I realized that many of the postings were reposts, so it makes sense

that those websites were identical. To extract the websites I used the regular expression

"((https?:\\/\\/)?([0-9a-z\\-\\/]{1,}\\.){1,}(com|org|net)([[:punct:]a-z0-9]{1,}[^\\.[:space:]])?)"

I knew that the websites usually started with https, and then a colon and two literal forward slashes, just because that is standard url formatting. Then it can be followed by any combination of one or more digits or letters or a dash or slash. Then this combination has to be followed by a literal period, and then either “com”, “org” or “net” depending on the type of website it is. Then we want to capture everything afterwards which is extra direction to the url location within the website. This is usually some time of punctuation like periods or slashes, and more characters and digits, so I simply grabbed for all letters a-z and all digits 0-9. However, we don’t want the regular expression to grab *everything* from the body after the website type which is why we use a “carrot” symbol and space or literal period to help signify the end of the website.

I was able to match 13951 which was a much higher number than the number of emails. I got a success rate of 40.12% for extracting the websites from the body.

5. To get the years from the body I used the regular expression:

'(19[0-9]{2}|20[0-1][0-9])[^0-9A-Za-z]'

because I knew the year had to start with either 19 or 20, since cars were not made before 1900. I assumed that there would be no postings made for years above 2015 but if there were typos, my regular expression could capture these. However, when looking at the matches in the body I did not see any such typos. Otherwise I simply used “19” followed by two digits between zero and nine or “20” followed by two digits zero to one and zero to nine.

I got a success rate of 73.06% for extracting the year from the body.

I then compared the years extracted from the body to the years in the vposts data column and got a success rate of 88.03%

6. I decided to extract the model types from the header column in vposts because it was easiest to see a pattern in the way the listings were presented.

In order to extract the model I:

1. Split each element in the header by a space.
2. Used agrep function and found the index of where maker is. I know that the model will most likely be the information after the maker in the header based on looking at

several header outputs, and now the element can be found that matches it because I have split along the spaces.
3. Then I implemented the knowledge from header output that the second index is generally model, by using " +1" to signify this pattern. I also said that if there was no match for maker(if I get 0) , then I want the output to be "na". I also used the function "tolower" to account for upper case and lower case discrepancies in the model title.
4. I then used "sort" and "table" to find the ten most common models and got the following output:

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

I do have many na values, and I also notice by looking at the whole output of the table that some of the model names are errors.

If I look at the total number of models this method extracted I get 1561, but since I know that some of the model names are errors and inaccurate so the actual number of models in the listings is lower than this number.

Challenges and Conclusions

For this section of the assignment, I did my best to create regular expressions to extract the information I wanted from a body of text that had more information than I needed. I did this by looking at the bulk of text I would like to sift through and finding patterns associated with the information I would like to retrieve. This could include ids before the information, or simply common starting characters that most of them share.

Although there are many tricks you can use to account for many possibilities in regular expressions, like "?" or "|" it is nearly impossible to account for every possible format, especially since there will be human error in postings made on craigslist. None of my matching success rates were every perfect.

Part 2: Modeling

I decided to look at the Honda Accord and the Jeep Grand Cherokee because they were two of the most common models in the data.

I used a linear model for this data, which is not exactly accurate, or contains fallacies because older cars sometimes have high prices etc. In order to use a linear model, you need to assume that the data is linear, there's little to no multicollinearity, and there is multivariate normality. Even though I chose to look at Cherokee and Accord; I looked at other models to see if we could assume that there were no sub correlations depending on the model. It turned out that the model type did effect

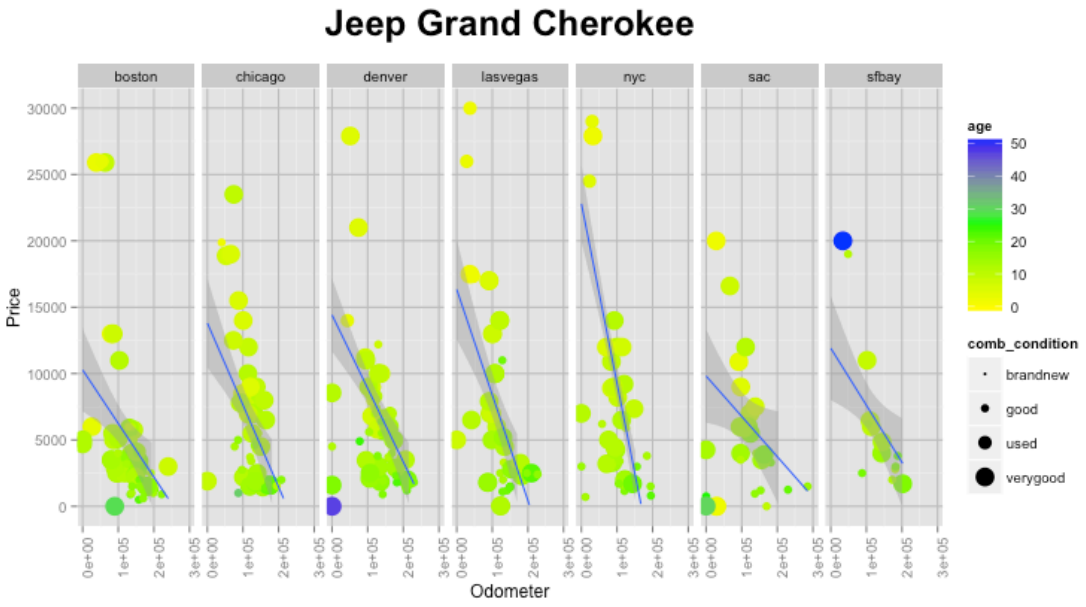
how well the linear model determined price, and therefore the linear model was not appropriate, even though from the plots of these two models and the correlation coefficients it looks like a possibility.

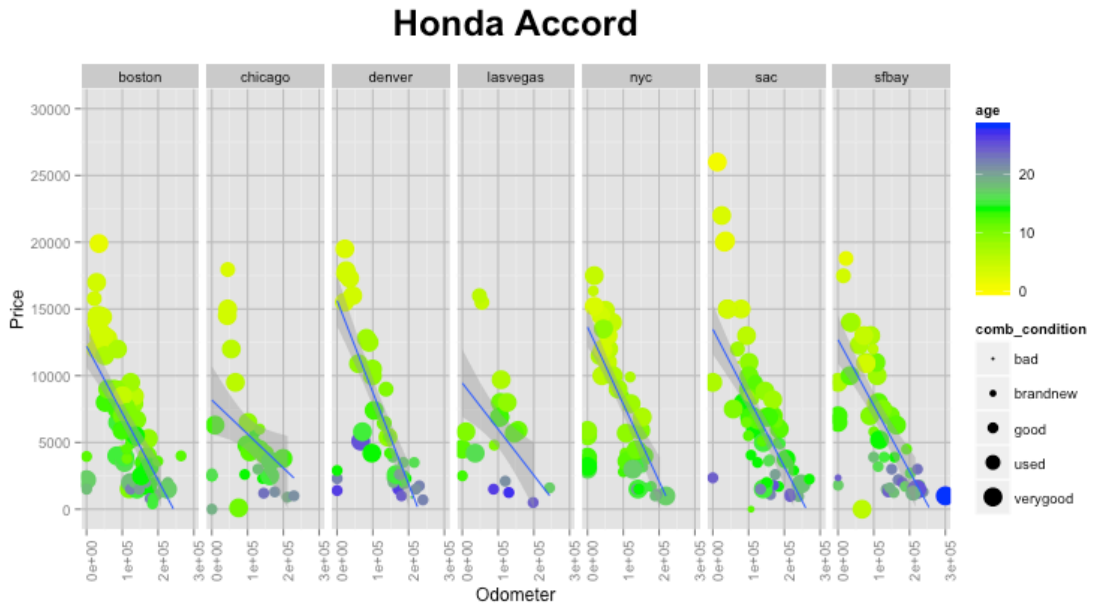
The linear model for Cherokee gave me an R^2 of 0.4471, and the linear model for accord gave me an R^2 of 0.4569.

I checked how the linear model applied to:

All models: R^2 = 0.1384
Civic: R^2 = 0.7826

This showed that the linear model had varying effectiveness based on car model type and the assumptions to make a linear model do not hold despite the appearance of linearity in the graphs.





Overall even though a lot of the relationships in the graph appear to be linear at first, they show that an exponential decay model might have been a better approach, just by looking at the overall tendencies.

We do see some important relationships in these plots. There are more older Accords than Grand Cherokees. This makes sense because the Accord is an older model. We also see that as age increases (the dots get bluer), price drops. We also see that generally as the condition improves (the dots get bigger), the price increases, but there are more outliers with this parameter and this could be do to inaccurate grouping of conditions or inaccurate estimates of condition by the person selling the car.

I would not use this model to help me purchase a car.

Appendix:

```
load("~/Desktop/stat141/Data/vehicles.rda")
#run fixed vposts from assignment first so that condition column is fixed
#and outliers are gone

#part 1: price

extract_prices = regexec("[$][:space:]?[0-9]{0,3}[:space:]?\\,?[0-9]{3}",
vposts$body)

# Extract prices from body column that satisfy the regular expression.
prices_match = regmatches(vposts$body, extract_prices)
```

```
# remove non-digits and replace with empty space
remove = gsub("[^0-9 ]", "", prices_match)

# Split by the space so we have individual prices to use
individual_prices = strsplit(remove, " ")

# convert the prices to numeric class
num_prices = lapply(individual_prices, as.numeric)

#only want one(max) output per post
max_prices = lapply(num_prices, max)
upper_prices = as.logical(max_prices >= 600000)
max_prices[upper_prices] = NA
max_prices = as.numeric(max_prices)

final_prices = as.numeric(gsub("^0$", "NA", max_prices))

# Total extracted:
sum(!is.na(final_prices))
#14125

#convert to data frame class in order to compare matches
prices = as.data.frame(final_prices)

# Returns true if vpost$price is found in vpost$body for each posting
#%in% an element in
price_check = sapply(1:nrow(vposts), function(x) vposts$price[x] %in%
final_prices[x])

succ_rate_price = table(vposts$price == prices)
succ_rate_price[[2]]/(succ_rate_price[[1]]+succ_rate_price[[2]])
#0.7777863

#part 2: vin
#method from Roger Peng Channel on YouTube

getvins= regexec("[Vv][Ii][Nn]\\:[[:space:]]*([A-HJ-NPR-Za-hj-npr-z0-9]{11}[0-
9]{6})", vposts$body)
vin_matches = regmatches(vposts$body, getvins)
vin = sapply(vin_matches, function(x) x[2])
matched_vin =sum(!is.na(vin))
# 5125

#success rate
matched_vin/nrow(vposts)
```

```
#0.1477925

#make a column in vposts for vin numbers
vposts$vin = vin

#part 3: phone numbers

getphones= regexec("[(]?([[:space:][:punct:]]?([0-
9]|zero|two|three|four|five|six|seven|eight|nine)){3}[)]?([[:space:][:punct:]]?([0-
9]|zero|two|three|four|five|six|seven|eight|nine)){7}", vposts$body, ignore.case =
TRUE)
phone_matches =regmatches(vposts$body, getphones)
phones = sapply(phone_matches, function(x) x[1])
matched_phones =sum(!is.na(phones))
#17698

#success rate
matched_phones/nrow(vposts)
#0.5103671

vposts$phones = phones

##part 4: emails
emails= grep("(([a-zA-Z0-9]+)\\@([a-zA-Z0-9]+)\\.([a-zA-Z]{2,3}))", vposts$body)
length(emails)
#103 emails in body

#this is a really low number, look at websites instead
#websites
getwebs= regexec("((https?:\\/\\/)?([0-9a-z\\-
\\/]{1,}\\.){1,}(com|org|net)([[:punct:]a-z0-9]{1,}[^\\.[:space:]])?)", vposts$body,
ignore.case = TRUE)
web_matches =regmatches(vposts$body, getwebs)
websites = sapply(web_matches, function(x) x[2])
#a lot of the listings are repsosts so it makes sense that many of the websites are the
same
matched_websites =sum(!is.na(websites))
#13951

#success rate
matched_websites/nrow(vposts)
#0.4012458

#add websites column
vposts$websites = websites
```

```
#part 5: years

getyears= regexec("(19[0-9]{2}|20[0-1][0-9])", vposts$body)
years_matches= regmatches(vposts$body, getyears)
years = sapply(years_matches, function(x) x[2])
matched_years =sum(!is.na(years))
#25336

#success rate
matched_years/nrow(vposts)
#0.7306284

#make a column in vposts for years
vposts$years = years

#compare years in body to year column in vposts
years_success = table(vposts$year == vposts$years)
#success of matching years in body to years in year column
years_success[[2]]/(years_success[[1]]+years_success[[2]])
#0.8803221

#part 6: model
#splitting each element by a space
s = sapply(vposts$header, function(x) strsplit(x," "))
#find the index of where maker is, difference between maker and header is the
model, finding the element in split that matches that
match.posi = lapply(1:nrow(vposts), function(x) agrep(vposts$maker[x], s[[x]],
ignore.case=TRUE))
# the second index is model, generally cars follow the pattern of maker, model so we
use +1
# if it's 0, make in na bc it didn't match a maker
model = tolower(lapply(1:nrow(vposts), function(x) if (length(match.posi[[x]])!=0)
s[[x]][min(match.posi[[x]])+1] else return(NA)))
sort(table(unlist(model)), decreasing=TRUE)[1:10] # top 10 models
#use nrow to find number of models extracted using this method
nrow(table(unlist(model)))
#1561

vposts$model=model

######### Part 2
vposts$age = 2016 - vposts$year

# chose accord and grand cherokee
# extract just data corresponing to accord and cherokee from model
accord = vposts[which(vposts$model == "accord"),]
```

```
grand_cherokee = vposts[which(vposts$model == "grand"),]

accord_test = lm(price ~ age + odometer + comb_condition + city, data = accord)

cherokee_test = lm(price ~ age + odometer + comb_condition + city, data =
grand_cherokee)

#check all models
all_test = lm(price ~ age + odometer + comb_condition + city, data = vposts)

#check other models
civic = vposts[which(vposts$model == "civic"),]
civic_test = lm(price ~ age + odometer + comb_condition + city, data = civic)

######### model graphing
#odometer readings lower for higher prices, age is lower for higher prices
#sometimes newer condition shows up on lower prices, this could be grouping
error
#there are more old Honda Accords

install.packages("ggplot2")
library(ggplot2)
##ggplot has many "magic" inputs,
#but basically using accord and cherokee data
#to fit linear models of price, odometer, condition, and city
ggplot(accord, aes(odometer, price, col = age))+geom_point(aes(size =
comb_condition))+
  labs(x = "Odometer", y = "Price",
       title = "Honda Accord")+
  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 = 90, hjust = 1))+
  scale_color_gradientn(colours = c("yellow", "green", "blue"))+
  scale_x_continuous(limits=c(0, 300000))+
  scale_y_continuous(limits=c(0, 30000), breaks=seq(0, 30000, 5000))+
  facet_wrap(~city, ncol = 7)+
  geom_smooth(method='lm')

ggplot(grand_cherokee, aes(odometer, price, col = age))+geom_point(aes(size =
comb_condition))+
  labs(x = "Odometer", y = "Price",
       title = "Jeep Grand Cherokee")+
  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 = 90, hjust = 1))+
  scale_color_gradientn(colours = c("yellow", "green", "blue"))+
```

```
scale_x_continuous(limits=c(0, 300000))+
scale_y_continuous(limits=c(0, 30000), breaks=seq(0, 30000, 5000))+
facet_wrap(~city, ncol = 7)+
geom_smooth(method='lm')
```