Used Cars Pricing Modeling

Reese Karo

2024-04-02

Contents

- To refer to a Python EDA over a similar (but not the same) data set, https://medium.com/mlearning-ai/detailed-exploratory-data-analysis-eda-on-used-cars-data-1bacac746ff4

Introduction

In this projct, we dive into the car markets and attempt to use machine learning to predict a vehicle's value in terms of selling price. Here, we are leveraging a rich dataset from Kaggle's car auctions, and embark on a mission using advanced machine learning algorithms. Our goal? To pinpoint the key features that determine a car's value and employ state-of-the-art models to predict prices accurately. This isn't just about numbers;

it's about unlocking the secrets of car valuation through data-driven insights. Join us on this compact yet potent journey to transform how we understand car prices.

Motivation

As a child growing up, I had been obsessed with sports cars. Old vintage racing cars, newer race cars, F1 cars, you give me a sports car from any generation and I could probably give you some details on it. I also enjoyed working on them with my dad. I enjoyed them enough that my family pitched the idea of rebuilding an old 1971 Porsche 914 that had not been running for over 10 years.. Now I'm sure anyone reading this has no idea what the last 3 numbers mean, but most people know what the Porsche symbol/name stands for. The project rebuild took me on a journey that I am very fortunate for, and has taught me many valuable life lessons that can't be taught in other scenarios. After falling in love with the idea of buying, selling, and trading cars, I learned my way around auction sites and thinking to myself what are the most important aspects of a car that make them valuable.

Throughout college, being an applied math major, I had always heard about the hardships and wonders of machine learning. I didn't think there was a spot for me in the subject, until I had joined the data science department by tacking on another major. When I ended up taking PSTAT 131 and learned there was a final project where the student chose the topic, I immediately thought about predicting car values.

Data Description

The data set can be found here: Car Auction Prices | Kaggle. Where we have the following variables:

`Year`, `Make`, `Model`, `Trim`, `Body`, `Transmission`, `Vin`, `State`, `Condition`, `Odometer`, `Color`, `Interior`, `Seller`, `MMR`, `Sellingprice`, and lastly the `Saledate`. The `Year` tells us what year the vehicle was made, the `Make`, `Model` and `Trim` define and tell us which company made the vehicle and with what model and trim variant, which can depict whether or not the price of the car will be higher or lower. The `Body` is what type of vehicle (i.e. SUV, Sedan, coupe, etc.). `Transmission` is how the car changes gears and is a categorical variable with two levels (automatic, or manual). The `VIN` gives us the vehicle identification number, but is unique to every single vehicle on the road today, so this will not be in our interest. `State` tells us where the vehicle was sold, and `Condition` tells if the car is in good health and shape from 0 being the worst to 5 being the best and can be a floating digit number. Odometer is the mileage on a car over the entire lifetime of the vehicle. `Color` and `Interior` is a categorical value describing the exterior color and interior color of the car, `Seller` gives us the details on the name of the dealer/dealership that sold the vehicle. `MMR` is the average wholesale price, odometer and condition grade of recent transactions for a given year, make, model and style, but will not be used in our model due to the correlation to the price, as well as other variables already being used in the `MMR` calculation, it would be wise to remove this so we may have a more general model. However, if we were to include this, we would have to take a different approach to our model due to the high correlation. `Sellingprice` tells us the value that the vehicle sold for at auction, and lastly `Saledate`, tells us when the vehicle was sold, but since time series data is out of the scope of the class, we will avoid using this variable.

Project Outline

Given the data set outline, our plans for this machine learning project will involve using these 4 models to help us predict the final selling prices. Here are the steps we must take to ensure good prediction results. We must first clean the data set for any missing data, and values that may be in the wrong columns. After we clean, we will want to explore the distributions of our highest selling cars, the distribution of our variables such as transmission to gauge how many of each type are being sold. And we also want to explore the correlation between our numeric predictors. After we wrap up the EDA process, we will want to split the data into our training and testing sets which will then be transformed into a 10-fold cross validation set, and then start exploring our models and workflows.

Due to the size of the data set, we will only be running four models, though in the future more may be run and may pose better results. The four in mind are - Elastic Net to capture the Lasso/Ridge Mixture and penalty results through a tuning grid through our folds, K-Nearest-Neighbors using a tuning grid to test multiple neighbors, Gradient Boosted Tree and lastly a random forest. After our models have run on the folded sets, we will then collect our best model and fit the data to our whole training set to collect metrics and see how it performed. Then we will choose our best model and fit to the testing set to truly see how it predicts pricing vehicles based on the features of a car given.

Installing Dependent Packages

```
# install.packages("xgboost")
# install.packages("ranger")
library(tidyverse)
library(tidymodels)
library(readr)
library(ggplot2)
library(GGally) # easy plotting device
library(corrplot)
library(ggthemes)
library(dplyr)
library(kableExtra)
library(naniar) # visualize missing data
library(forcats) # data manipulation
library(stringr)
library(glmnet) # elastic net for linear regression
library(xgboost) # gradient boost trees
library(forcats)
library(janitor)
library(vip)

# For the neural Network model
#install.packages("torch")
#install.packages("AppliedPredictiveModeling")
#install.packages("brulee")
library(AppliedPredictiveModeling)
library(torch)
library(brulee) # engine for neural network
```

Now that we have all of the packages installed, we may now begin our Exploratory Data Analysis (EDA)

EDA

We first need to load our data using the read csv file and by setting our seed to a random number so that we can replicate the process every time with the splitting that will happen in future code.

```
tidymodels_prefer()
set.seed(1234)
car_data <- read.csv('/Users/reese/Documents/School/UC Santa Barbra /Junior/Winter/131/Project/car_pric

# running clean names through janitor package
car_data <- car_data %>%
  clean_names()

head(car_data)
```

```
##   year   make               model       trim  body transmission
## 1 2015    Kia             Sorento         LX   SUV    automatic
## 2 2015    Kia             Sorento         LX   SUV    automatic
## 3 2014    BMW            3 Series 328i SULEV Sedan    automatic
## 4 2015  Volvo                 S60         T5 Sedan    automatic
## 5 2014    BMW 6 Series Gran Coupe       650i Sedan    automatic
## 6 2015 Nissan              Altima      2.5 S Sedan    automatic
##                  vin state condition odometer color interior
## 1 5xyktca69fg566472    ca         5    16639 white    black
## 2 5xyktca69fg561319    ca         5     9393 white    beige
## 3 wba3c1c51ek116351    ca       4.5     1331  gray    black
## 4 yv1612tb4f1310987    ca       4.1    14282 white    black
## 5 wba6b2c57ed129731    ca       4.3     2641  gray    black
## 6 1n4al3ap1fn326013    ca         1     5554  gray    black
##                                                 seller   mmr sellingprice
## 1                           kia motors america, inc 20500        21500
## 2                           kia motors america, inc 20800        21500
## 3             financial services remarketing (lease) 31900        30000
## 4                           volvo na rep/world omni 27500        27750
## 5             financial services remarketing (lease) 66000        67000
## 6 enterprise vehicle exchange / tra / rental / tulsa 15350        10900
##                                saledate
## 1 Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
## 2 Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
## 3 Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
## 4 Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
## 5 Thu Dec 18 2014 12:30:00 GMT-0800 (PST)
## 6 Tue Dec 30 2014 12:00:00 GMT-0800 (PST)
```

By using the clean names feature, we are able to easily change the names, making them all lower case or upper case depending on the algorithm used in the `janitor` library. By using `head(car_data)`, we can view the first 6 observations and the entire column set as well. We also can use the `summary` feature to gauge what kind of data we are dealing with .

```
summary(car_data)
```

```
##      year              make              model              trim
##  Length:558863      Length:558863      Length:558863      Length:558863
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##      body           transmission           vin               state
##  Length:558863      Length:558863      Length:558863      Length:558863
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
##   condition            odometer         color             interior
##  Length:558863      Min.   :     1   Length:558863      Length:558863
##  Class :character   1st Qu.: 28371   Class :character   Class :character
```

```
##  Mode  :character   Median : 52254   Mode  :character   Mode  :character
##                     Mean   : 68320
##                     3rd Qu.: 99109
##                     Max.   :999999
##                     NA's   :120
##     seller             mmr            sellingprice      saledate
##  Length:558863      Length:558863      Min.   :     1   Length:558863
##  Class :character   Class :character   1st Qu.:  6900   Class :character
##  Mode  :character   Mode  :character   Median : 12100   Mode  :character
##                                        Mean   : 13611
##                                        3rd Qu.: 18200
##                                        Max.   :230000
##                                        NA's   :26
```

We immediately see that we have 558,863 observations, which for most modern computational hardare, is a lot. Though we may sacrifice the results, we are going to subset our data in order to save computational time and speed due to the limitations of hardware. We also see that the `year` and `condition` is of type character, so we must keep in mind we need to change this to type dbl and integer.

Random Sampling

Due to the size of the data set being roughly 558,000 observations, we need to take a random sample of the data due to the limitations of computation on generic computer specs. Thus we will take subset of 100,000 observations to maintain similar distribution. Achieving our goal of randomly subsetting is easy with `sample_n`, and setting the number to 1000.

```
# Subset with a random sample of 100,000 observations
car_data <- car_data %>%
  sample_n(100000)
```

Now that we have a solid group of 100,000 observations, we need to check how clean the data is.

Data Mutation

As a quick reminder there should only be two options for the choice of the transmission. We need to dive into the transmission column to see what type of classifications there are since we should only have either automatc, or manual.

```
table(car_data$transmission)
```

```
##
##           automatic    manual     sedan     Sedan
##     11831     84995      3169         3         2
```

We can see that there are some with no transmission character value rather than NA, and some as sedan and Sedan which need to be changed. We can do this below with a few assumptions. Allow me to explain. In this next block of code, the goal here is to make sure our data set shows whether the observations that are missing are going to be NA. This way, in a few blocks below, we will be able to truly see a representation for our missing data. From their we can either impute the missing data, or drop the NA terms. We also should explore the other categories by tabling the columns.

```
table(car_data$color, car_data$interior)
```

```
##
##                      - beige black  blue brown burgundy  gold  gray green
##               127     0     0     0     0     0        0     0     0     0
##   -             0     0   542  2142     3    93        1     4  1198     3
##   16633         0     0     0     0     0     0        0     0     0     0
```

```
##   18561         0     0     0     1     0     0        0     0     0     0
##   6864          0     0     0     0     0     0        0     0     1     0
##   9410          0     0     0     0     0     0        0     0     0     0
##   9562          0     0     0     0     0     0        0     0     0     0
##   beige         0    54   444   339     0    59        0     2   268     1
##   black         0   557  1720 10970    16   388       12    10  4697     7
##   blue          0   336   974  3276    66   118        3     3  3587     2
##   brown         0    43   292   339     1   117        2     1   169     1
##   burgundy      0    52   325   529     1    26        2     1   413     2
##   charcoal      0     2     3    46     0     3        1     0    27     0
##   gold          0    79   686   318     0    56        2    16   262     0
##   gray          0   418   839  7345    21   133        4     2  5342     5
##   green         0   105   420   427     2    52        0     1   682    11
##   lime          0     0     0     0     0     0        0     0     1     0
##   off-white     0     6    72    81     0     4        1     1    47     0
##   orange        0    18    23   208     0    12        0     0    82     0
##   pink          0     3     0     7     0     0        0     0     1     0
##   purple        0    11    38   136     1     4        0     0    68     0
##   red           0   242  1070  3330     6    70        2     8  2317     1
##   silver        0   480   525  7314    30    73        3     3  6265     8
##   turquoise     0     2     9     7     1     2        0     0    18     0
##   white         0   642  2759  6609    28   400        8     9  6360     8
##   yellow        0     7     2   163     0     0        0     0    53     0
##
##              off-white orange purple   red silver   tan white yellow
##                      0      0      0     0      0     0     0      0
##   -                 18      1      3    14      6   439     1      0
##   16633              0      0      0     0      1     0     0      0
##   18561              0      0      0     0      0     0     0      0
##   6864               0      0      0     0      0     0     0      0
##   9410               0      0      0     0      0     0     1      0
##   9562               0      0      0     0      1     0     0      0
##   beige              0      0      0     1      0   503     2      0
##   black             25      3     14    72     16  1294     5      0
##   blue               3      1      6     2     18   715     4      0
##   brown              0      1      0     0      0   236     0      0
##   burgundy           0      0      3     1      3   293     3      0
##   charcoal           0      0      0     0      0     8     0      0
##   gold               0      0      0     1      3   546     0      0
##   gray               6      3      4    20     38   480     5      0
##   green              0      0      0     0      3   345     2      0
##   lime               0      0      0     0      0     0     0      0
##   off-white          0      0      0     0      0    52     0      0
##   orange             0      4      0     0      0    15     0      0
##   pink               0      0      0     0      0     0     0      0
##   purple             0      0      1     2      0    23     0      1
##   red                3      0      5    39     17   755     3      0
##   silver             0      1     13    24     55   308     1      0
##   turquoise          0      0      0     0      0     4     0      0
##   white             24      3      9    57     46  2024    22      0
##   yellow             1      0      0     0      0     3     1      2
```

We can see the table of those with the following color exterior interior combinations. If we look at the first 2 rows and the first two columns we see a dash and colors with a value of " ". This can be visualized easier

below.

```
head(car_data %>%
       filter(color==""&interior=="") %>%
       select(year, make, model, color, interior)
)
```

```
##   year          make    model color interior
## 1 2012 Mercedes-Benz  E-Class
## 2 2008          Ford    F-150
## 3 2013 Mercedes-Benz  E-Class
## 4 2014      Infiniti      Q50
## 5 1993        Saturn S-Series
## 6 2012 Mercedes-Benz  E-Class
```

Converting to NA

We must now save our cleaned car data set as a new variable and continue to update the new variable. We can change all observations with transmission being equal to sedan, and Sedan to automatic due to the majority of cars being automatic, with the mutate feature. By changing the variables with values that are " " to NA, we will be able to visualize and drop the NA values much easier.

```
car_data_cleaned <- car_data %>%
  mutate(
    transmission = ifelse(tolower(transmission) == "sedan", "automatic", transmission), # this reads as
    transmission = ifelse(transmission == "", NA, transmission), # if no value for the transmission slo
    make = ifelse(make == "", NA, make),
    model = ifelse(model == "", NA, model),
    body = ifelse(tolower(body) == "sedan", "sedan", body),
    body = ifelse(tolower(body) == "", NA, body),
    body = tolower(body), # requiring all lower case for body
    color = ifelse(tolower(color) == "-"| tolower(color) == "", NA, color), # this reads: if color has
    interior = ifelse(interior =="-" | interior == "", NA, interior))
```

Converting Body

In this block of code we can shorten down the body styles to just a few categories very easily with the `mutate` function and `case_when` function. This way we are only left with convertible, coupe, hatchback, pickup, sedan, suv/wagon, and van - The majority of cars on the road. Lastly this will help us deal with data in the wrong columns. Lastly, lumping the types of bodies to a general name will save a lot of computation time since we will not have to dummy code all of these later on. We will use this technique moving forward for other variables.

```
car_data_cleaned <- car_data_cleaned %>%
  mutate(body = case_when(
    body %in% c("access cab", "cab plus", "cab plus 4", "club cab", "crew cab", "crewmax cab", "double
    body %in% c("beetle convertible", "convertible", "cts-v coupe", "cts coupe", "e-series van", "elanti
    body %in% c("coupe", "cts-v coupe", "cts coupe", "g coupe", "g37 coupe", "genesis coupe", "q60 coupe
    body %in% c("hatchback") ~ "hatchback",
    body %in% c("minivan", "promaster cargo van", "transit van", "ram van") ~ "van",
    body %in% c("sedan", "g sedan") ~ "sedan",
    body %in% c("suv", "wagon", "cts wagon", "cts-v wagon", "tsx sport wagon", "navigation", "navitgati
    TRUE ~ body # Keeps the original value if none of the above conditions are met
  ))
table(car_data_cleaned$body)
```

```
##
```

```
## convertible       coupe   hatchback      pickup       sedan   suv/wagon
##        2383        3562        4626        8441       44499       28685
##         van
##        5418
```

Now there are 0 sedan transmissions, and more NA data, and less values for which the `body` variable could fall under. Also by viewing the data set, we see that Transmission data is also missing some observations. This means that some observations do not have values that fall in "automatic" or "manual". We can fix this by either imputing values when creating the recipe by KNN or another method, or by removing some observations.

```
# Note: It's crucial to maintain the modifications (like lowercase transformation) through all subseque
car_data_cleaned <- car_data_cleaned %>%
  mutate(
    year = as.numeric(year),
    make = tolower(make) %>% as.factor(),
    model = tolower(model) %>% as.factor(),
    trim = tolower(trim) %>% as.factor(),
    body = tolower(body) %>% as.factor(),
    color = tolower(color) %>% as.factor(),
    interior = tolower(interior) %>% as.factor(),
    transmission = tolower(transmission) %>% as.factor()
  )
```

Correcting state and vin

If we now examine the state columns and vin table, we would visualize too many differnet values since a VIN is unique to every car on the road. However we can see the table for states.

```
table(car_data_cleaned$state)
```

```
##
##                   3vwd17aj2fm258506 3vwd17aj5fm206111 3vwd17aj5fm219943
##                 5                 1                 1                 1
## 3vwd17aj5fm225953 3vwd17aj8fm239622                ab                al
##                 1                 1               154                 4
##                az                ca                co                fl
##              1602             13068              1461             14784
##                ga                hi                il                in
##              6358               218              4157               719
##                la                ma                md                mi
##               401              1182              1995              2670
##                mn                mo                ms                nc
##              1641              2936               334              3893
##                ne                nj                nm                ns
##               766              4865                40                11
##                nv                ny                oh                ok
##              2279              1024              3877                 7
##                on                or                pa                pr
##               644               206              9696               471
##                qc                sc                tn                tx
##               248               738              3796              8138
##                ut                va                wa                wi
##               318              2163              1396              1730
```

We see we have some values that should be in the vin column rather than the state column. We need to clean

up the data so State and VIN and condition don't have the wrong data or data in the wrong columns. This is so that we can transform the data and find a correlation of condition and price and odometer. Thus, we may create a variable storing the values of the vins in the state list. From there we can filter out observations that have those values, and or values that are " ".

```
### Correcting misclassified VINs in the state column and removing empty states
state_vin_list <- c("3vwd17aj2fm258506", "3vwd17aj5fm206111", "3vwd17aj5fm219943", "3vwd17aj5fm225953",
car_data_cleaned <- car_data_cleaned %>%
  filter(!(state %in% state_vin_list)) %>%
  filter(state != "")
```

Converting character columns to numeric columns

We still need to change our numeric data from characters to integers or to double(decimals in R language). Thus using as.integer and as.double we can comvert very easily.

```
### Conversion of character columns to numeric where applicable
car_data_cleaned <- car_data_cleaned %>%
  mutate(
    condition = as.double(condition),
    mmr = as.integer(mmr),
    odometer = as.integer(odometer)
  )
```

We make note that we must set our variables that are characters to factors so that the models later on can pick up on when a level is repeated in the large data set of observations. And of course ensuring that they are lowercase.

Dropping NA make observations

The reasoning behind this choice is that there are roughly observations with missing make, model, and odometer. Due to the size of the data set, removing these observations is an easy choice. We can do this

```
### Removing observations with NA or empty strings in 'make', 'model', and 'odometer'
car_data_cleaned <- car_data_cleaned %>%
  filter(make != "", !is.na(make), model != "", !is.na(model), !is.na(odometer))
table(car_data_cleaned$make)
```

```
##
##         acura     airstream  aston martin          audi       bentley
##          1045             1             5          1046            17
##           bmw         buick      cadillac     chevrolet      chrysler
##          3646           969          1355         10786          3132
##        daewoo         dodge      dodge tk           dot       ferrari
##             1          5492             1             1             2
##          fiat          ford           geo           gmc     gmc truck
##           136         16809             3          1907             3
##         honda        hummer       hyundai      infiniti         isuzu
##          4912           146          3978          2678            33
##        jaguar          jeep           kia   lamborghini    land rover
##           238          2776          3255             1           347
##     landrover         lexus       lincoln         lotus      maserati
##             4          2121          1071             1            18
##         mazda      mercedes    mercedes-b mercedes-benz       mercury
##          1550            16             1          2983           358
##          mini    mitsubishi        nissan    oldsmobile      plymouth
##           567           748          9671            67             4
```

```
##       pontiac       porsche           ram   rolls-royce          saab
##           850           240           817             3           102
##        saturn         scion         smart        subaru        suzuki
##           500           320            68           901           163
##         tesla        toyota    volkswagen         volvo            vw
##             5          7292          2277           665             7
```

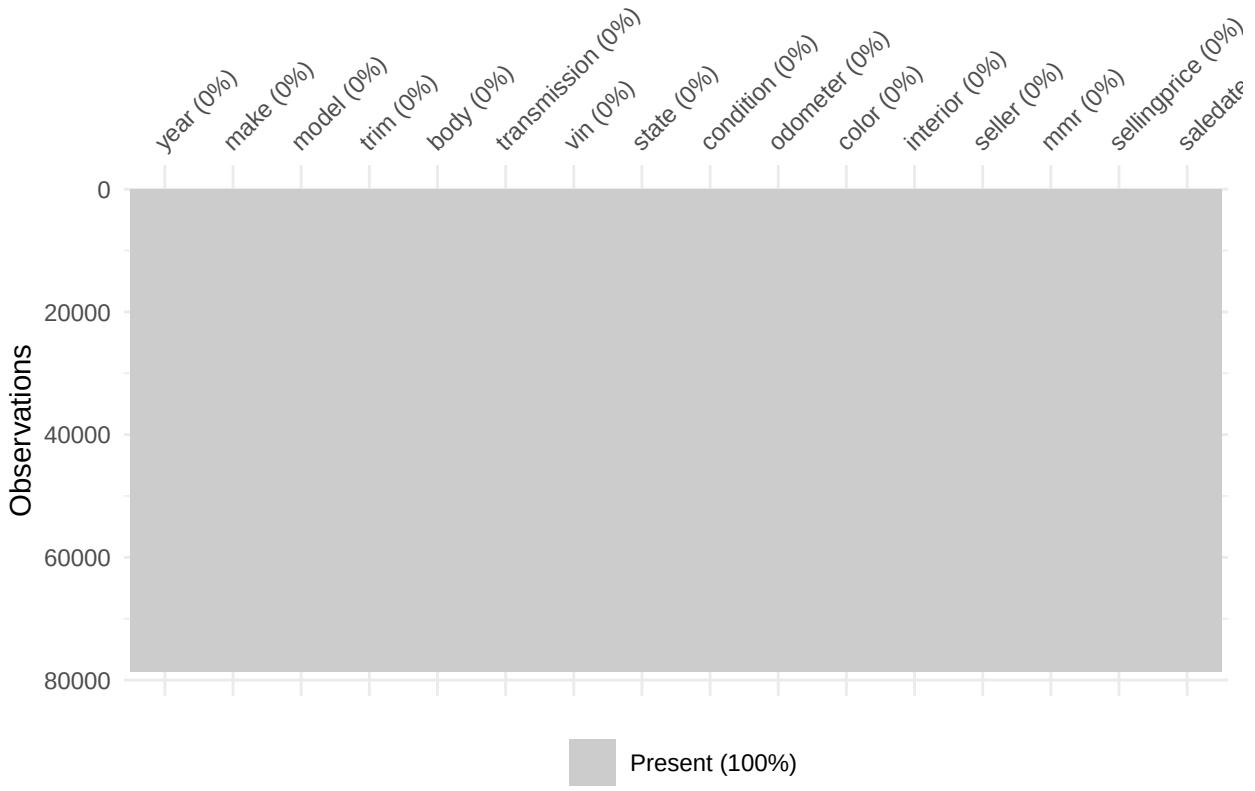
Last thing we need to address is the issue of there being levels with names "truck" and "tk" after the company. The reason we need address this is because for companies that sell trucks (i.e. Chevy, Dodge/Ram, Toyota, etc.) the label in this data set for those companies with trucks are stand alone makes that go by "chevy truck", "dodge tk", "mazda tk". Therefore we must change our data so that these fall under the same make and model so that there is no mismatch in makes, so that our recipe later on can run properly. One last note is that the factors for make has levels land rover, and landrover, mercedes, mercedes-b, and mercedes-benz, which we can just leave as one for each factor. We will add this in the code below.

```
### Combining Makes for trucks and aligning subsidiary brands with parent companies
car_data_cleaned <- car_data_cleaned %>%
  mutate(
    make = case_when(
      str_detect(make, "truck$") ~ str_replace(make, " truck", ""), # Removes " truck" from the end
      str_detect(make, "tk$") ~ str_replace(make, " tk", ""), # Removes " tk" from the end
      str_detect(make, "landrover$") ~ str_replace(make, "landrover", "land rover"),
      str_detect(make, "mercedes$") ~ str_replace(make, "mercedes", "mercedes-benz"),
      str_detect(make, "mercedes-b$") ~ str_replace(make, "mercedes-b", "mercedes-benz"),
      TRUE ~ make
    ) %>%
    str_trim() %>%
    as.factor() # Ensuring the 'make' column remains as a factor after manipulation
  )
```

```
# lump together make, model, and trim to ensure enough in each for our splitting.
car_data_cleaned <- car_data_cleaned %>%
  mutate(
    make = fct_lump_n(make, n = 35),
    model = fct_lump_n(model, n = 400),
    trim = fct_lump_n(model, n = 600)
  )
car_data_cleaned <- car_data_cleaned %>% drop_na() # finally use drop_na to remove any others we didnt
```

Summarizng the missing data

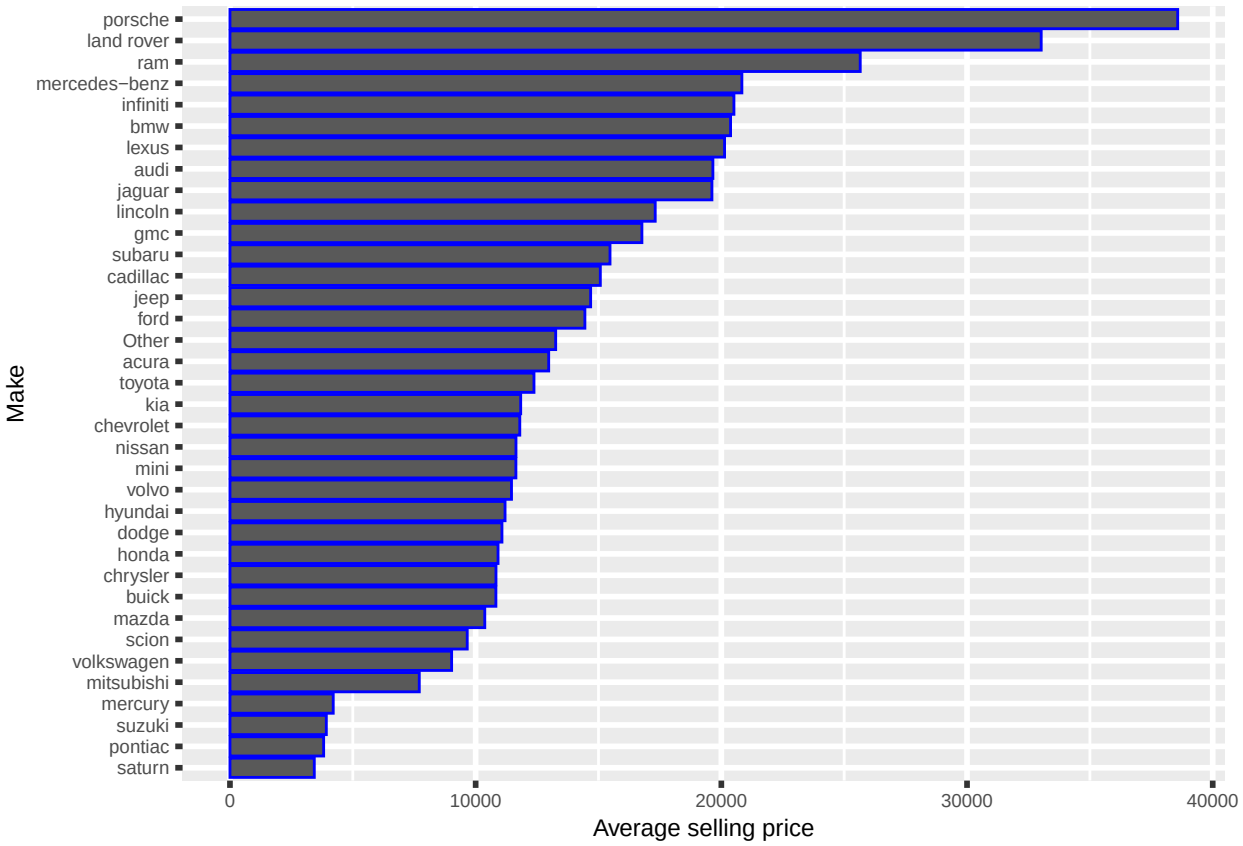
```
vis_miss(car_data_cleaned, warn_large_data = F)
```



From the `vis_miss` plot, which displays information about missing data for each variable, we have no missing values after filtering, and dropping NA values.

Avg pricing and counted observations for each make

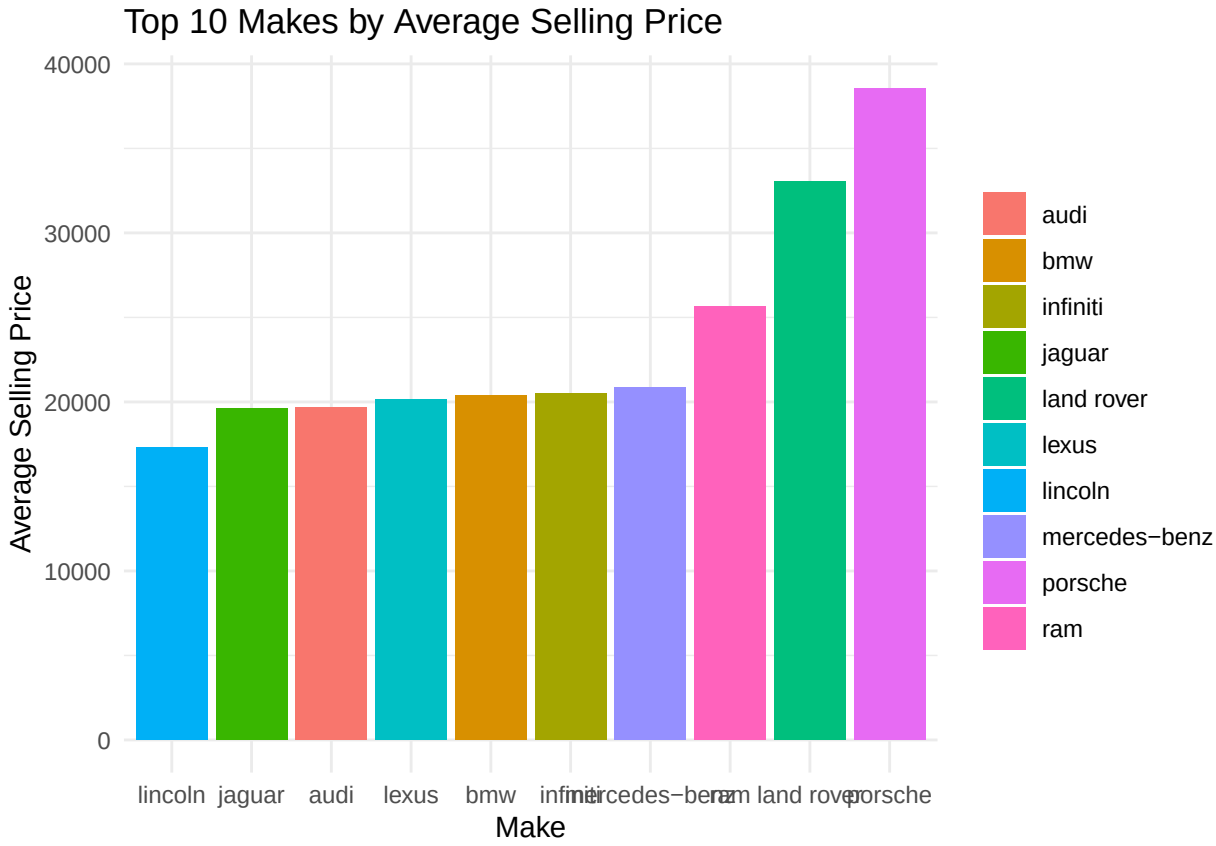
```
### summarizing the sell prices of each make
car_sorted_byprice <- car_data_cleaned %>%
  group_by(make) %>% # grouping by the make
  summarise(avg_sell_price = mean(sellingprice, na.rm = TRUE)) %>% # summarizing by the mean of the sel
  arrange(desc(avg_sell_price)) # descending avg selling price

### plotting using a bar chart
barplot_sorted <- ggplot(car_sorted_byprice, aes(x = reorder(make, avg_sell_price), y = avg_sell_price)
  geom_bar(stat = "identity", color = "blue") +
  coord_flip() +
  labs(x= "Make", y = "Average selling price") +
  theme(text = element_text(size = 9),element_line(size =1))

print(barplot_sorted)
```



In the plot above, we see that Porsche's have the highest average selling price after we have lumped together the top 35 makes. Then followed by Range Rover, Ram, Mercedes-Benz, Infiniti, BMW, Lexus, Audi, and so on. This is pretty standard in todays world as well, but what is interesting is the make "Ram". Ram does sell high valued pick-up trucks, but we can see that they tend to hold their value well compared to other truck companies such as Ford or Chevrolet.

Top 10 selling cars

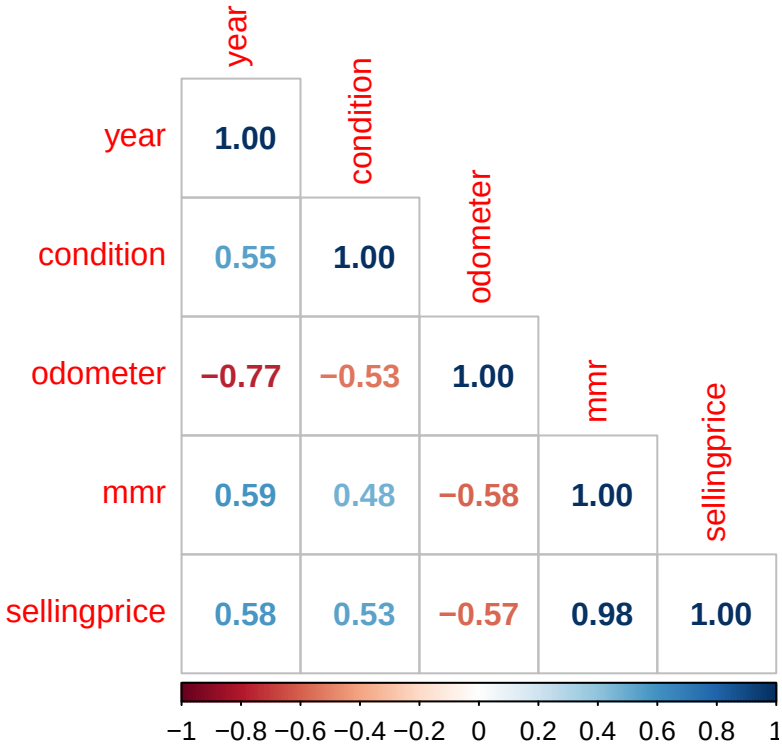
```
library(ggplot2)
library(dplyr)
### summarizing the sell prices of each make for the top 10 makes
car_sorted_byprice <- car_data_cleaned %>%
  group_by(make) %>%
  summarise(avg_sell_price = mean(sellingprice, na.rm = TRUE)) %>%
  arrange(desc(avg_sell_price)) %>%
  top_n(10, avg_sell_price)
# Selecting the colors for the graph

### plotting using a bar chart
barplot_sorted <- ggplot(car_sorted_byprice, aes(x = reorder(make, avg_sell_price), y = avg_sell_price,
  geom_bar(stat = "identity") +
  labs(title = "Top 10 Makes by Average Selling Price", x = "Make", y = "Average Selling Price") +
  theme_minimal() +
  theme(legend.title = element_blank())

barplot_sorted
```



Again, we can visualize (with better colors) the top 10 companies with the highest average selling price, from lowest, Lincoln, on the left, to highest, Porsche, on the right.

Correlation plot of numerical data

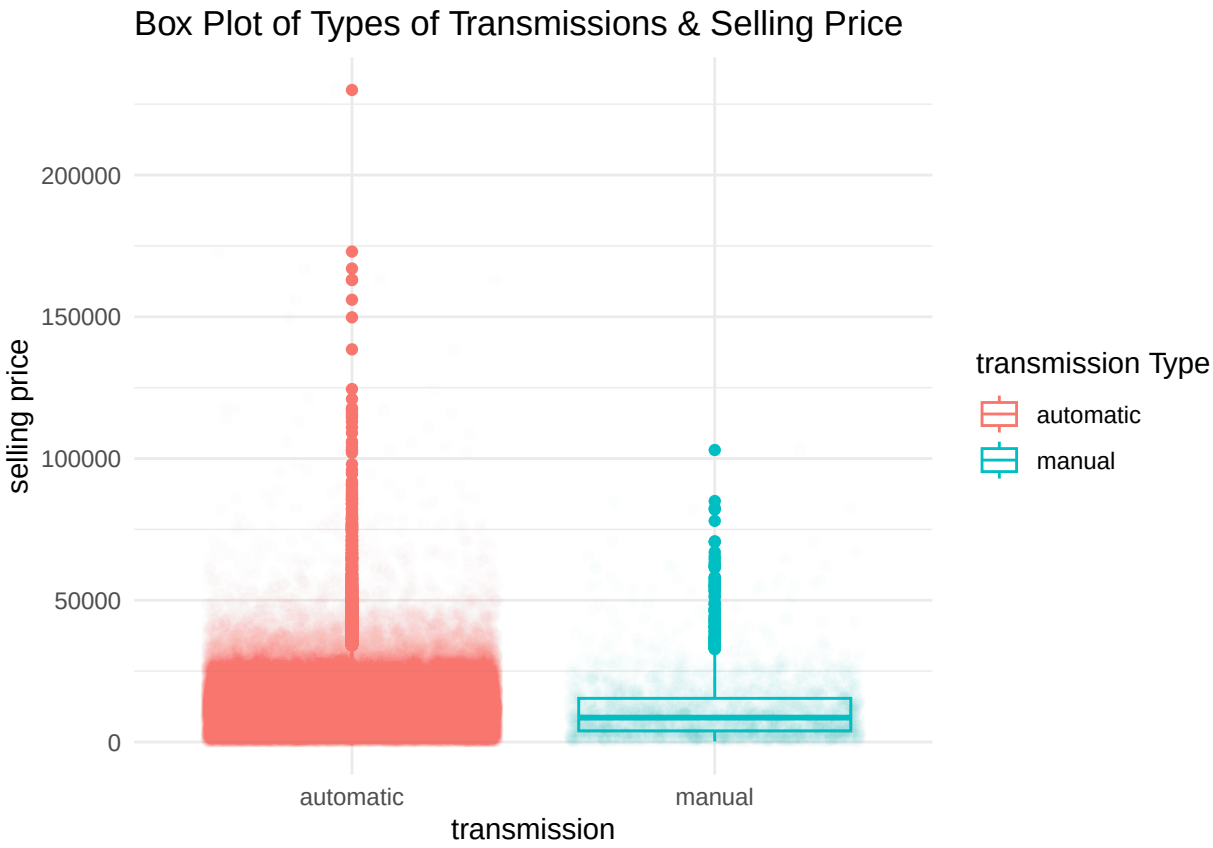
```
library(corrplot)
car_data_cleaned %>%
  select(is.numeric) %>%
  cor(use = "pairwise.complete.obs") %>%
  corrplot(method = "number", type = "lower", diag = T)
```



The correlation plot above can give us a lot of information about our numeric data with just a few lines of code. We can see there is positive correlation between the mmr and condition at 0.48, as well as the selling price and the condition, 0.53. This can be interpreted as when the car sells for more or the mmr is higher, the condition will be higher. Ford the condition and year variables, we see a positive correlation of 0.55, which, intuitively, makes sense. Given the year a car was made, the newer it is, typically the nicer the condition it is in, because it is newer, and typically less owners the car has gone through. For the odometer (mileage) on the car, we can view this as negative correlation, or interpreted as, lower mileage indicates a higher condition number (or a higher mileage and lower condition number). Similarly for odometer and year, as the year of the car increases higher, we see the odometer typically is lower, as shown in the correlation plot.

Transmission Types versus Selling Price

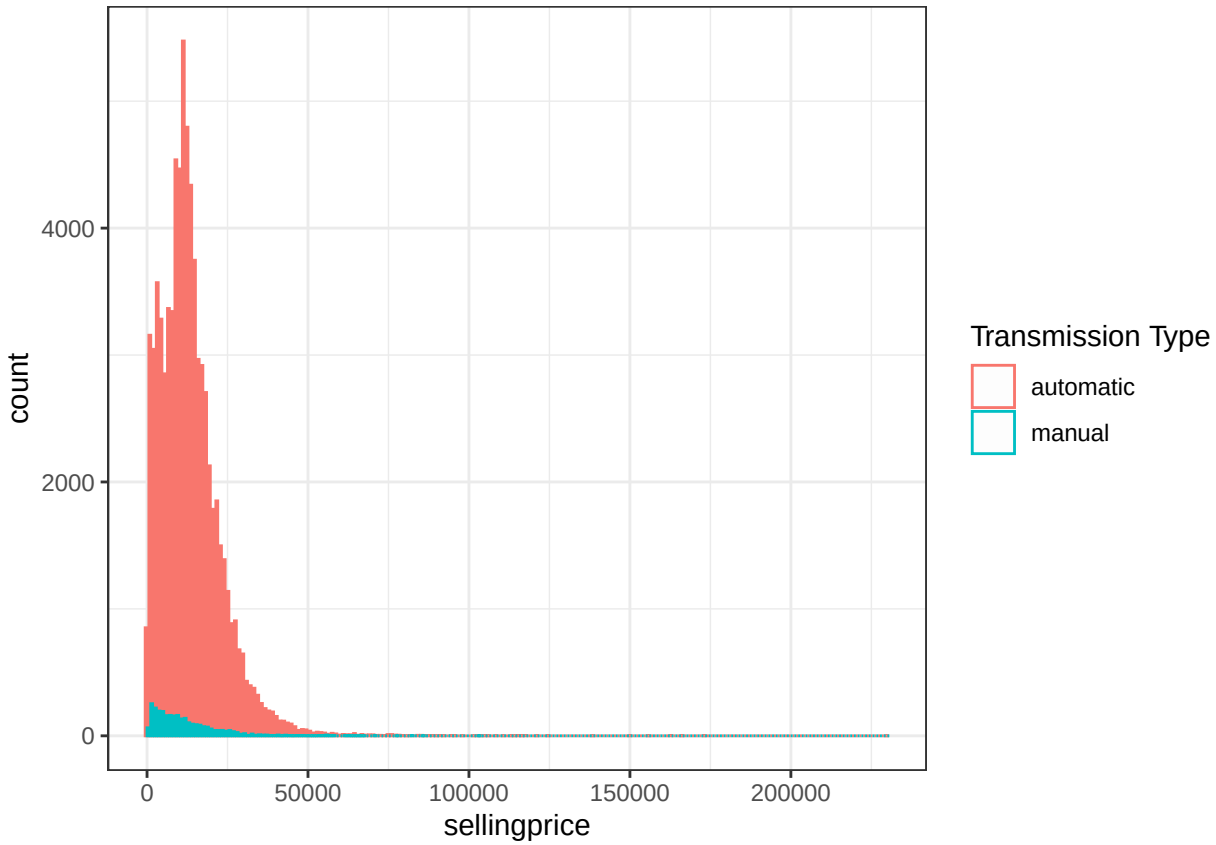
```
# creating a box plot for the transmissions
car_data_cleaned %>%
  ggplot(aes(x = transmission, y = sellingprice, color = transmission)) +
  geom_boxplot() +
  geom_jitter(alpha = 0.01) +
  labs(title = "Box Plot of Types of Transmissions & Selling Price", x = "transmission", y = "selling pr
  theme_minimal()
```



From this plot we can see that most of the cars selling for each transmission is below the $50,000 selling price mark. This graph also shows that there are fewer cars with manual transmissions that sell on the market.

Selling Price and Odometer distribution curves

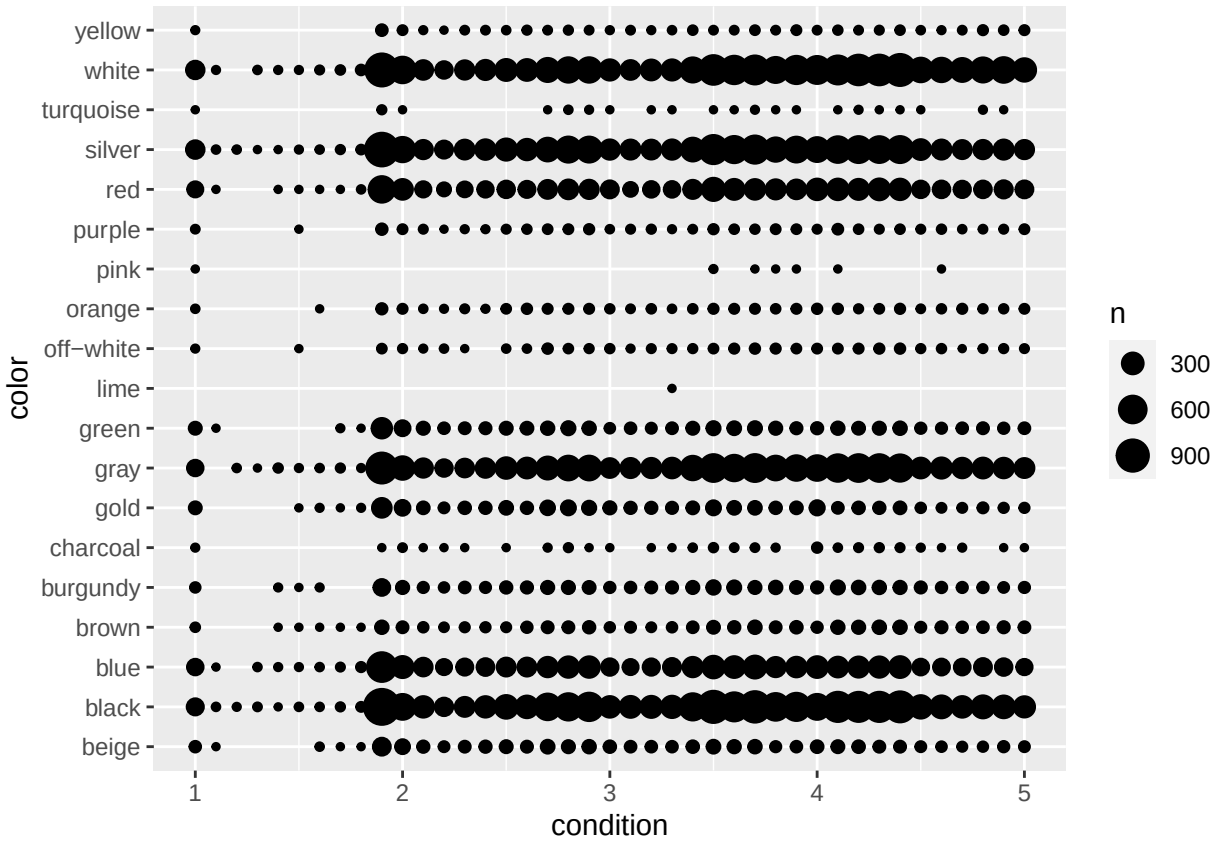
```
ggplot(aes(x = sellingprice, color = transmission), data = car_data_cleaned) +
  geom_histogram(bins = 200, alpha = 0.01, position = "dodge") +
  labs(color = "Transmission Type") +
  theme_bw()
```



This plot helps show the distribution of the outcome variable (selling price) with respect to a count on the y-axis. We can see it has a higher peak of cars selling at the 18,000-22,000 price range and is skewed to the right since it has a longer tail on the right side. This means, as it should, less cars are being sold at the 50,000 value and beyond.

Condition of Cars vs the Color

```
# represents and shows the relationship between condition and color of the cars
ggplot(car_data_cleaned, aes(x = condition, y = color)) +
  geom_count()
```



This count plot can help shows us that vehicles in white, silver, red, gray, blue and black are the most popular and we can see that their conditions are dense around 2 - 5 condition level. Thus we can easily select a few of the colors to do work with including the interior, instead of using all of the colors.

```
n_top_colors = 10

car_data_cleaned <- car_data_cleaned %>%
  mutate(color = fct_lump_n(color, n = n_top_colors)) %>%
  mutate(interior = fct_lump_n(interior, n = n_top_colors))
```

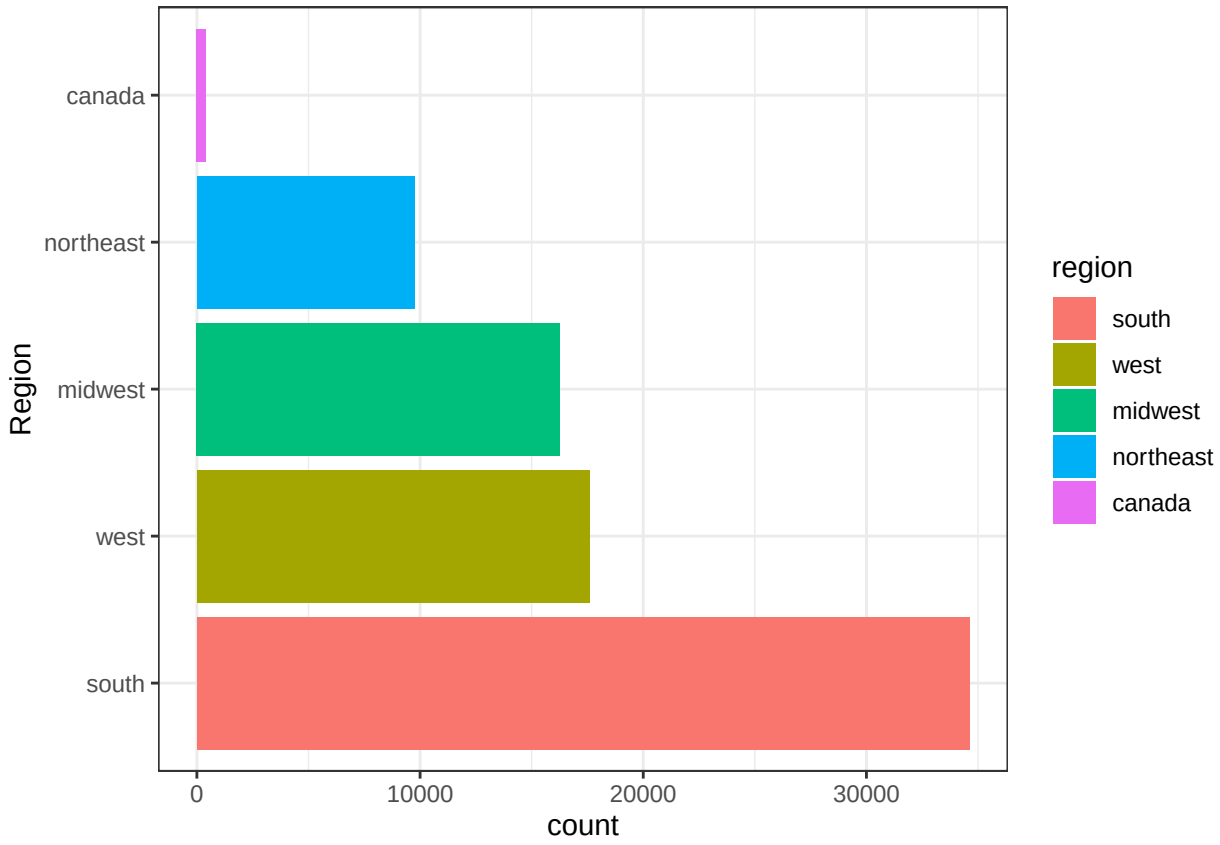
Lumping States

Lumping together states with the `forcats` library is an easy choice because there are so many states, where instead we can lump them into regions. This is done to minimize the dummy code variables from all states and Canada regions in the set, to only using 5 regions.

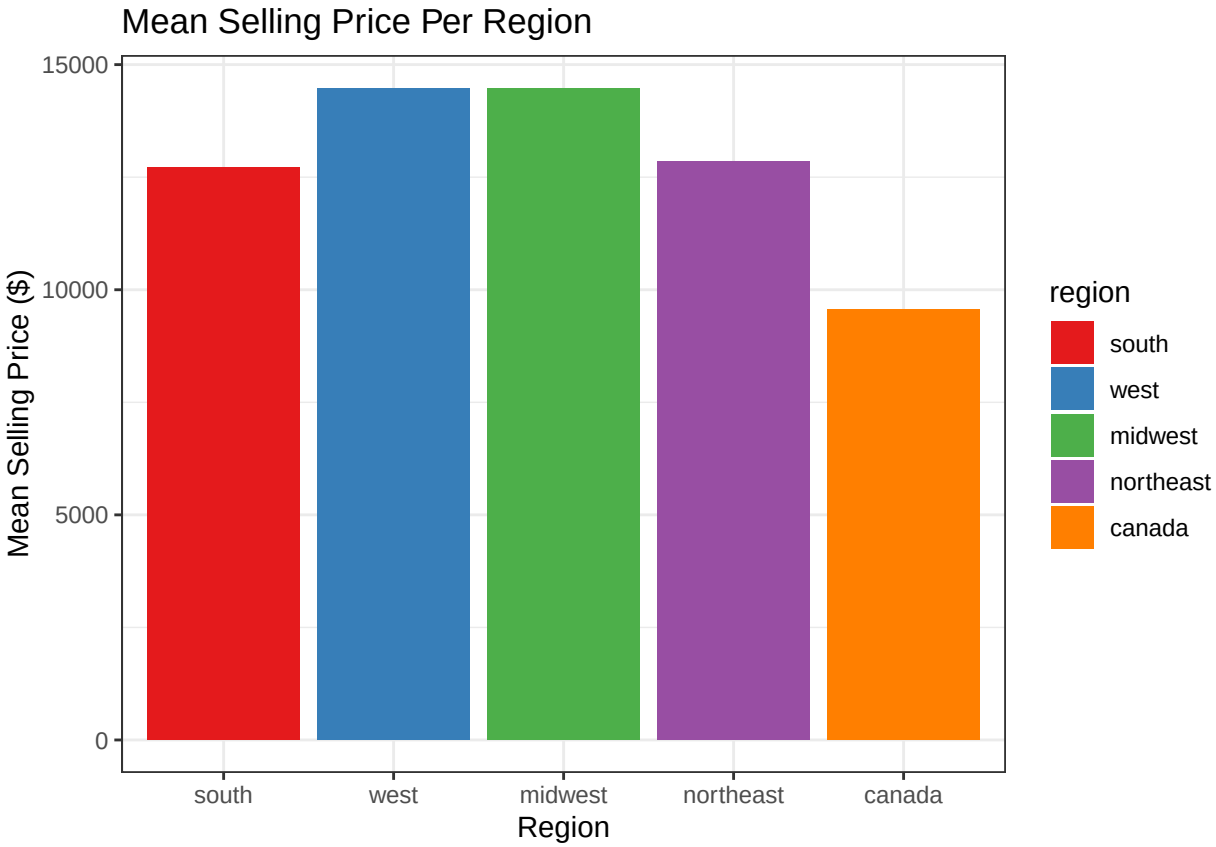
```
# Convert state abbreviations to lowercase if they are not already
car_data_cleaned$state <- tolower(car_data_cleaned$state)

# Mutate to add a region column based on state, then drop the state column
car_data_cleaned <- car_data_cleaned %>%
  mutate(region = forcats::fct_collapse(state,
                                        west = c("ca", "or", "wa", "id", "mt", "nv", "wy", "ut", "co",
                                        midwest = c("nd", "sd", "ne", "ks", "mn", "ia", "mo", "wi", "il
                                        northeast = c("me", "vt", "nh", "ma", "ct", "ri", "ny", "pa", "
                                        south = c("tx", "ok", "ar", "la", "ky", "tn", "ms", "al", "wv",
                                        canada = c("ns", "ab", "qc", "pr", "on"))) %>%
  select(-state)
```

Visualizations for Regional Cars

```
library(forcats)
library(ggplot2)

car_data_cleaned %>%
  ggplot(aes(y = forcats::fct_infreq(region), fill = region)) +
  geom_bar() +
  theme_bw() +
  ylab("Region")
```



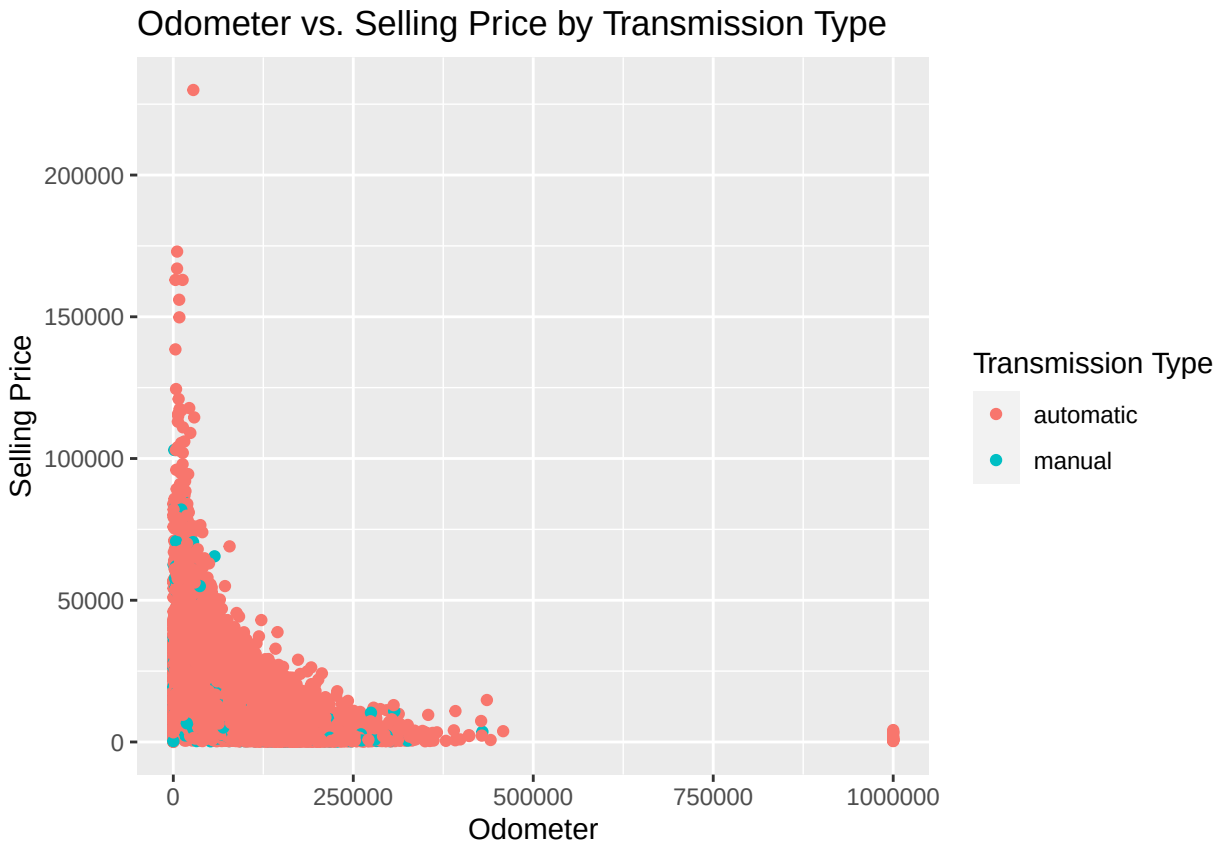

```
### Region versus mean price
car_data_cleaned %>%
  group_by(region) %>%
  summarise(avg_sell_price = mean(sellingprice, na.rm = TRUE)) %>% # Ensure your column name is correct
  ggplot(aes(x = region, y = avg_sell_price, fill = region)) + # Use 'fill' for color inside the bars
  geom_bar(stat = "identity") +
  labs(title = "Mean Selling Price Per Region", x = "Region", y = "Mean Selling Price ($)") +
  theme_bw() +
  scale_fill_brewer(palette = "Set1") # Optional: Use a color palette for visual appeal
```



In the first plot, we can get a quick visualization of how many cars are selling in each region. We see that the south is selling roughly the same amount of cars than the west and the midwest combined. In the second plot, the top 2 mean selling regions are the West and Midwest. We note that Canada does have ~2,000 cars selling and still has a mean selling price of roughly 10,000. Though, it is not clear without delving into the outliers in the Canada region to see whether this is USD or in CAD. Another note to add is that the south has substantially more cars to sell but their mean price is a bit lower, indicating that the quality of cars in the south have higher mileage or are in lesser condition.

Odometer versus Selling Price

```
car_data_cleaned %>%
  ggplot(aes(x = odometer, y = sellingprice, color = transmission)) +
  geom_point() +
  labs(title = "Odometer vs. Selling Price by Transmission Type",
       x = "Odometer",
       y = "Selling Price",
       color = "Transmission Type")
```



In this plot, the trend shows that as the odometer variable increase to the maximum of the set, or 1,000,000, the selling price decreases. The opposite can be said as well, which is conveyed in the correlation plot a few chunks above.

Modeling

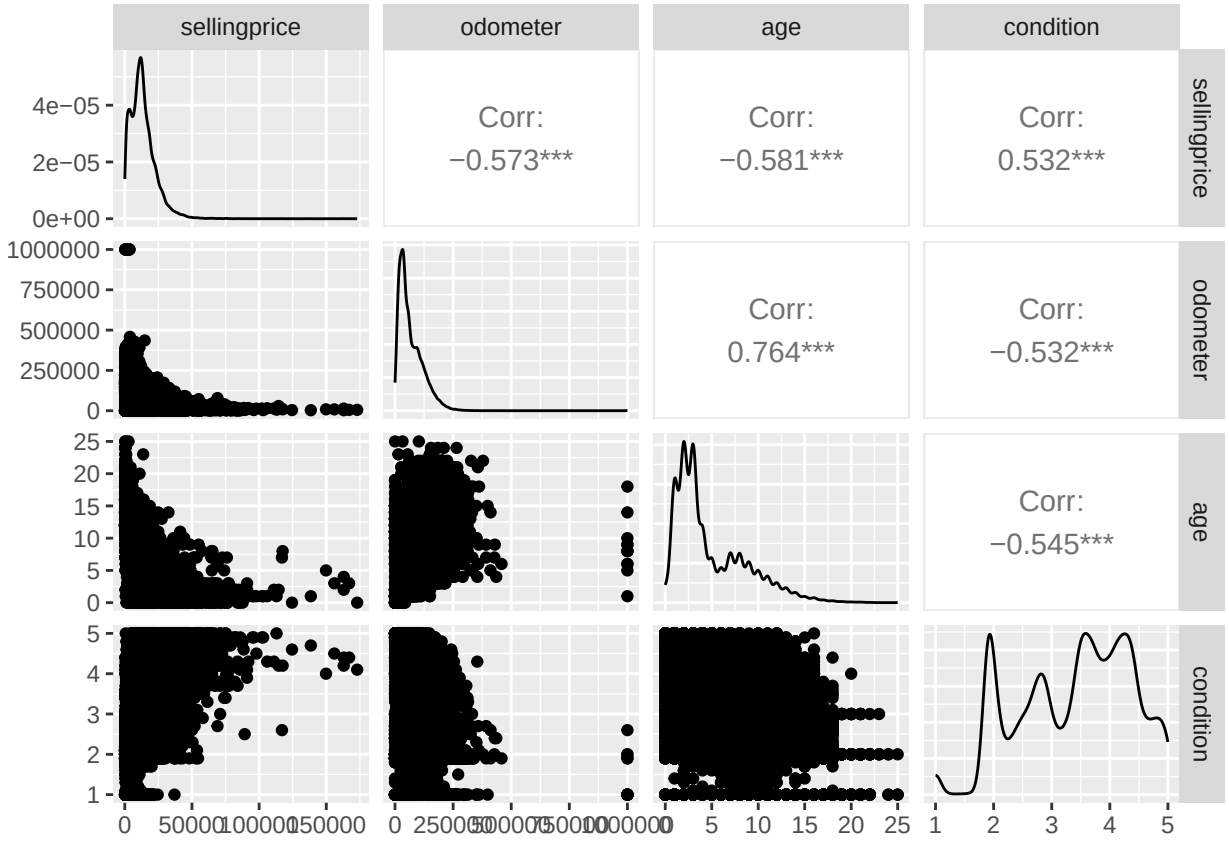
In this section of the project, we will be starting the process of modeling by: 1. first splitting our data into a training and testing set, then splitting our training data into folds. 2. Next is to create the recipe for our models. 3. Setup models and workflows. 4. Setup tuning grids for the hyperparameters. 5. Tune the models on the folded data. 6. Collect metrics on all models through autoplots and RMSE values. 7. Re-fitting models on the training set. 8. Finally finishing by fitting our best model to the testing set.

Lets begin!

Splitting Data

```
# Removing seller, mmr (similar to selling price), saledate, and year since age is based off of year
car_data_cleaned <- car_data_cleaned %>%
  mutate(age = max(year) - year)
car_data_cleaned <- select(car_data_cleaned, -c(seller, mmr, saledate, vin, year))

# Split into train and testing
car_data_split <- initial_split(car_data_cleaned, prop = 7/10, strata = sellingprice)
car_data_train <- training(car_data_split)
car_data_test <- testing(car_data_split)

# Put training into folds
```

```
car_data_fold <- vfold_cv(car_data_train, v = 10, strata = sellingprice) # for computational time

car_data_train %>% select(sellingprice, odometer, age, condition) %>% GGally::ggpairs()
```



The plot above gives us a quick representation using the `GGally` library, of a few of our predictors in the training set. We can visualize the distribution of a few predictors as a pdf with count as the y-axis, scatter plot, and see the correlation between the predictors. We can see that the condition has a pretty even spread across the domain of numbers from 1, to 5 which is a good when we are trying to predict car auction selling prices. Also added in the code, was the removal of the variables: `seller`, `mmr`, `saledate`, `vin`, and `year`. These were unessary since `mmr` has a high positive correlation with price, `age` variable comes from `year`, `saledate` is time series data which is out of the scope of the class, and `vin` is obvious.

Car Recipe

We now are at the stage to make our recipe. we will be using `step_novel` to handle any new categories that come up in the folds since each make, and model can differ. The step function will convert the value to a new level for each nominal predictor called new. `step_dummy` will handle converting nominal predictor into k-1 dummy variables for each variable. The `step_zv` function is a saftey function incase we have have predictors that have zero variance. This overall helps our model since we need to normalize the data later on.

```
# Car recipe
car_recipe <- recipe(sellingprice ~., data = car_data_train) %>%
  step_novel(all_nominal_predictors()) %>%  # Handle novel categories
  step_dummy(all_nominal_predictors()) %>%  # Convert specified vars to dummies
  step_zv(all_predictors()) %>% # to remove predictors with zero variance
  step_normalize(all_numeric_predictors(), -all_outcomes()) # Scale numeric data, excluding outcome var
#prep(car_recipe) %>% bake(car_data_train)
```

Models for the recipe

- We will be creating 4 different models for our regression problem, with both K = 10 fold cross validation
  - Regularized Regression with elastic Net
  - Gradient Boosted trees
  - Random Forest
  - Deep Neural Network (DNN)

```
# Create linear regression
lin_reg_mod <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# Create regularized model
regression_net_mod <- linear_reg(mixture = tune(), penalty = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("regression")

# Create a gradient boosted tree model
gbt_reg_mod <- boost_tree(mtry = tune(),
                          trees = tune(),
                          learn_rate = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

# Create a Random Forest Model
rand_forest_mod <- rand_forest(mtry = tune(),
                               trees = tune(),
                               min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")

# Creat a Neural Network Model
nnet_mod <- mlp(epochs = tune(), hidden_units = tune(), penalty = tune(), learn_rate = 0.1) %>%
  set_engine("brulee") %>%
  set_mode("regression")
```

Workflow Creation

```
# linear regression workflow
lin_reg_wf <- workflow() %>%
  add_recipe(car_recipe) %>%
  add_model(lin_reg_mod)

# Regularization workflow
reg_wf <- workflow() %>%
  add_recipe(car_recipe) %>%
  add_model(regression_net_mod)

# Gradient Boosted tree Workflow
gbt_reg_wf <- workflow() %>%
  add_recipe(car_recipe) %>%
```

```
  add_model(gbt_reg_mod)

# Random tree workflow
rf_wf <- workflow() %>%
  add_recipe(car_recipe) %>%
  add_model(rand_forest_mod)

# Neural Network workflow
nnet_wf <- workflow() %>%
  add_recipe(car_recipe) %>%
  add_model(nnet_mod)
```

Creating Tuning Grids

```
# Elastic net grid
reg_net_grid <- grid_regular(
  mixture(range =c(0, 1)),
  penalty(range = c(1, 10),
          trans = identity_trans()),
  levels = 10
)
# gradient boosted Trees
gbt_grid <- grid_regular(
  mtry(range = c(1, 6)),
  trees(range = c(200, 600)),
  learn_rate(range = c(-10, -1)),
  levels = 5
)
# random forest grid
rf_grid <- grid_regular(
  mtry(range = c(1, 6)),
  trees(range = c(200, 600)),
  min_n(range = c(10, 20)),
  levels = 5
)
# neural network grid
nnet_grid <- grid_regular(
  epochs(range = c(200, 600)),
  hidden_units(range = c(10, 50)),
  penalty(range = c(.01, .1)),
  levels = 5
)
```

Tuning the grids to our folds

```
# Tuning our models with our K = 10 CV folded training data

# Tune Linear Regression
lm_fit_cv <- lin_reg_wf %>%
  fit_resamples(resamples = car_data_fold)

# Tune Elastic Net
```

```
tune_regnet_grid <- tune_grid(
  object = reg_wf,
  resamples = car_data_fold,
  grid = reg_net_grid
)
# Tune Gradient Boosted Trees
tune_gbt_grid <- tune_grid(
  object = gbt_reg_wf,
  resamples = car_data_fold,
  grid = gbt_grid
)
## Tune Random Forest
tune_rf_grid <- tune_grid(
  object = rf_wf,
  resamples = car_data_fold,
  grid = rf_grid
)
# Tune the neural net to the folded data
tune_nnet_grid <- tune_grid(
  object = nnet_wf,
  resamples = car_data_fold,
  grid = nnet_grid
)
```

Saving the tunes

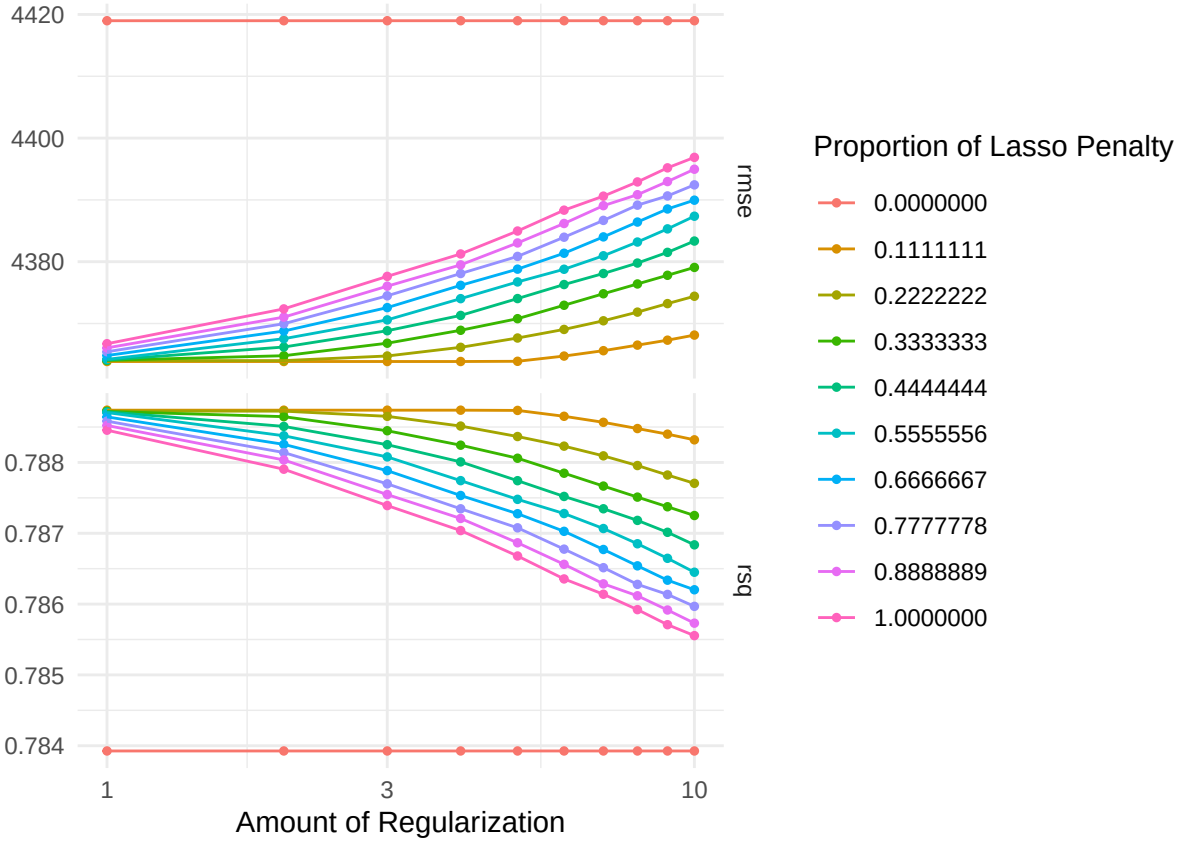
```
#### using write rds to save tunes to a file so we dont have to rerun the code
# linear regression
write_rds(lm_fit_cv, file = "Tuningdata/linreg2.rds")
# elastic net
write_rds(tune_regnet_grid, file = "Tuningdata/Elasticnet_tune.rds")
# gradient boosted trees
write_rds(tune_gbt_grid, file = "Tuningdata/Gradient_boost_tune.rds")
# random forest
write_rds(tune_rf_grid, file = "Tuningdata/rf_tune.rds")
# NNet tune
write_rds(tune_nnet_grid, file = "Tuningdata/NNET_tune.rds")
```

Loading the tunes

```
# using read_rds for efficiency of loading

#load linear regression file
lm_fit_cv <- read_rds("Tuningdata/linreg.rds")
# load elastic net tune
elastic_tune <- read_rds("Tuningdata/Elasticnet_tune.rds")
# load Gradient boosted trees tune
boosted_tune <- read_rds("Tuningdata/Gradient_boost_tune.rds")
# load Random Forest Tune
rf_tune <- read_rds("Tuningdata/rf_tune.rds")
# load NNET
nnet_tune <- read_rds("Tuningdata/NNET_tune.rds")
```

Elastic Net Results on CV

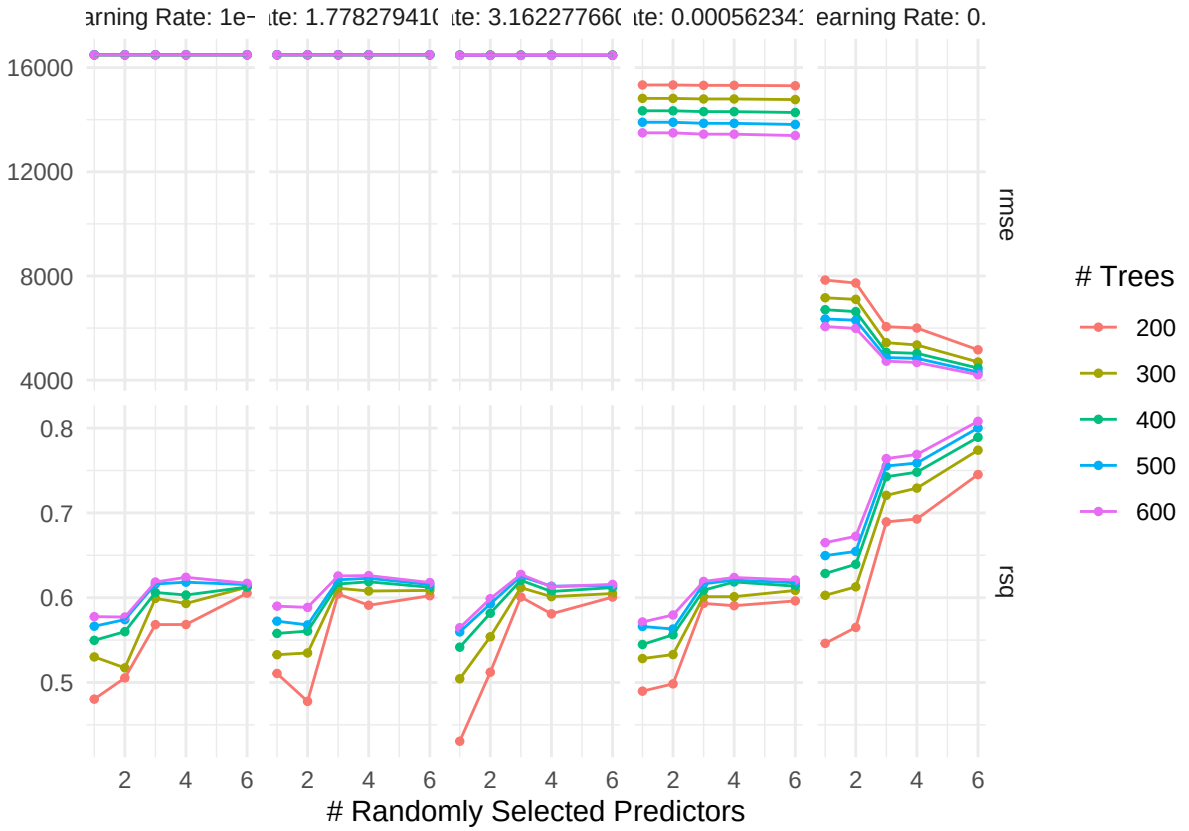
```
autoplot(elastic_tune) + theme_minimal()
```



The autplot above tells us about the resuts of our elastic net regression model. We conclude that increasing our regularization amount from 1 to 10 leads to a worse performing model for all proportions of lasso penalty. We also can interpret that the RMSE increases when we increase the Lasso penalty. Therefore our model doesnt perform as well when we zero out coefficients.

Gradient Boosted Tree Results on CV

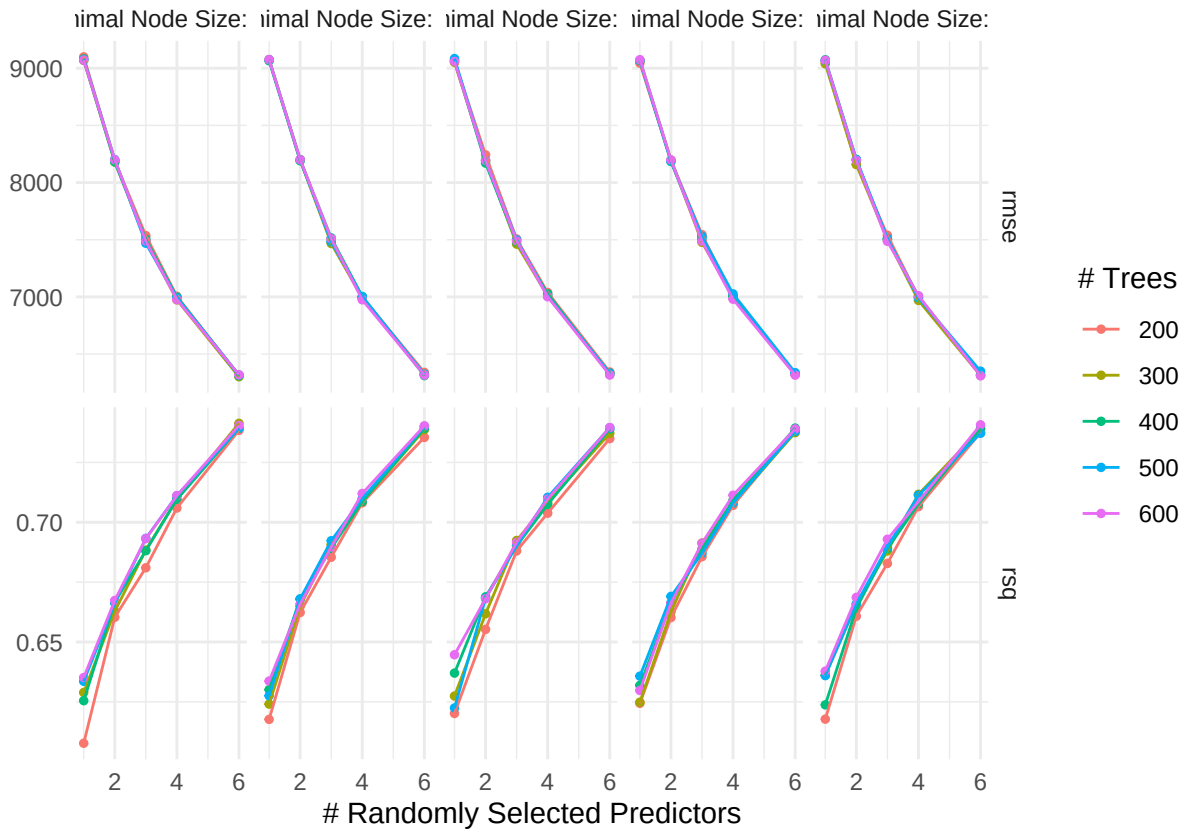
```
autoplot(boosted_tune) + theme_minimal()
```



From the Gradient Boosted Tree results on the 10-fold cross set, we see that increasing the learning rate, number of trees, and # of randomly predictors tends to give better results. However for learning rates below 0.1, all of the results hit their max around 16,500 RMSE. When the learning rate is at 0.0005623413, we see a dramatic decrease in RMSE, especially as we increase the number of trees. However, the number of randomly selected predictors doesn't change the RMSE.

Random Forest results

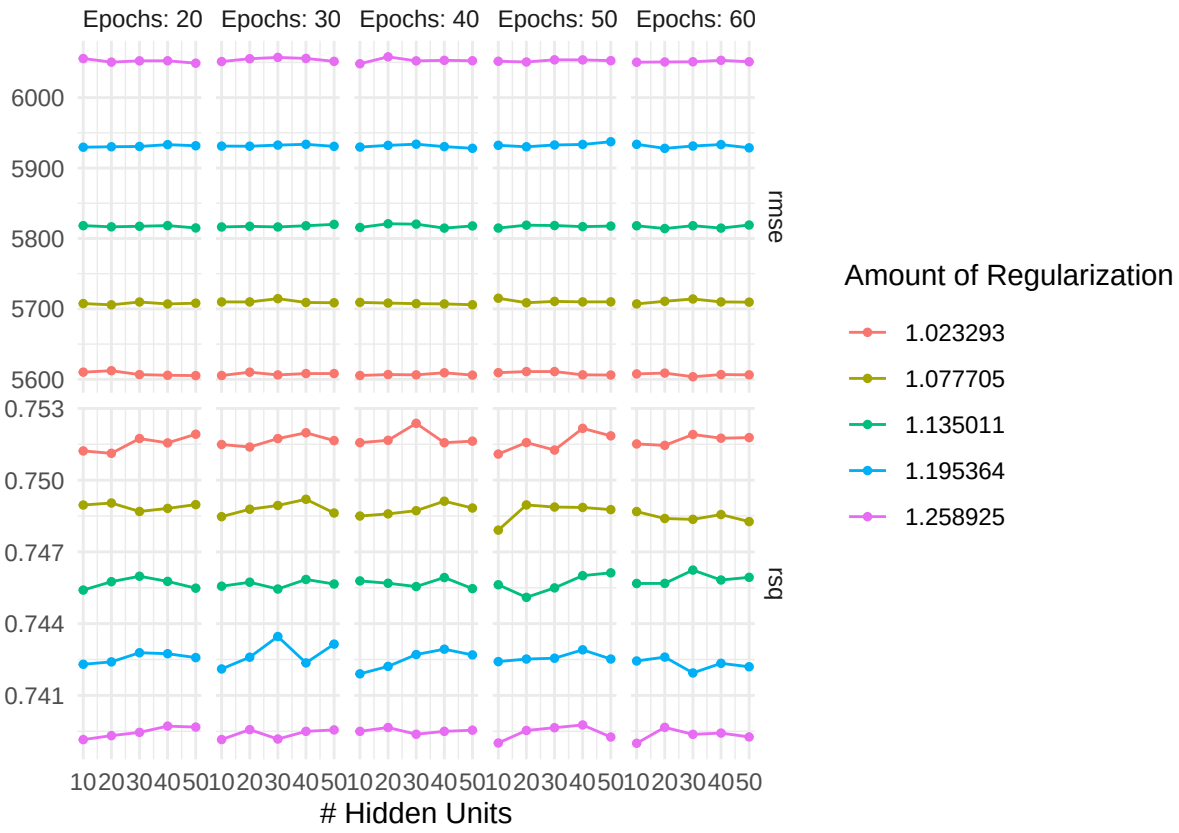
```
autoplot(rf_tune) + theme_minimal()
```



The random forest autoplot is very interesting to interpret. This because no matter which level of `mtry`, the results roughly follow the same pattern. Increasing the number of randomly selected predictors always tends to increase performance, no matter how many trees are used.

Neural Net results

```
autoplot(nnet_tune) + theme_minimal()
```




```
show_best(nnet_tune, metric='rmse') %>% slice(1,2,3)
```

```
## # A tibble: 3 x 9
##   hidden_units penalty epochs .metric .estimator  mean     n std_err .config
##          <int>   <dbl>  <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1           30    1.02    600 rmse    standard   5604.    10    85.6 Preprocess~
## 2           50    1.02    200 rmse    standard   5605.    10    86.0 Preprocess~
## 3           10    1.02    300 rmse    standard   5605.    10    85.8 Preprocess~
```

For the last plot displaying the neural network results. We see across the board for the number of `epochs`, or number of times the data has been seen, are relatively the same. Thus choosing a model with 200 `epochs` is a better choice than 600 due to potential over fitting, and efficiency purposes. Since all are resulting nearly the same, choosing the best will give us the model with 30 `hidden_units`, and 600 `epochs`.

Collecting Best Models

By using `select_best`, we can let tidyverse choose our best model for each model that has a hyperameter to tune on.

```
# Best hyperparemeter tunes
best_mix_pen <- select_best(elastic_tune, metric = 'rmse')
best_class <- select_best(boosted_tune, metric = 'rmse')
best_rf <- select_best(rf_tune, metric = 'rmse')
best_nnet <- select_best(nnet_tune, metric = 'rmse')
```

Finalizing Workflows

IN the next few chunks of r code, we will be finalizing the workflows with our best models, fitting the best models to the training set, and comparing the performance.

```
# finalizing workflows for all models
final_regnet <-finalize_workflow(reg_wf, best_mix_pen)
final_gbt <- finalize_workflow(gbt_reg_wf, best_class)
final_rf <- finalize_workflow(rf_wf, best_rf)
final_nnet <- finalize_workflow(nnet_wf, best_nnet)
# fitting to the training data
fit_lin_reg_wf <- fit(lin_reg_wf, car_data_train)
final_regnet <- fit(final_regnet, car_data_train)
final_gbt <- fit(final_gbt, car_data_train)
final_rf <- fit(final_rf, car_data_train)
final_nnet <- fit(final_nnet, car_data_train)
```

```
# saving files for the training data fitted model
write_rds(fit_lin_reg_wf, "fit_train_linreg.rds")
write_rds(final_regnet, "fit_train_regnet.rds")
write_rds(final_gbt, "fit_train_gbt.rds")
write_rds(final_rf, "fit_train_rf.rds")
write_rds(final_nnet, "fit_train_nnet.rds")
```

```
# loading and reading fit data to observe below
fit_train_linreg <- read_rds("Trainfitdata/fit_train_linreg.rds")
fit_train_regnet <- read_rds("Trainfitdata/fit_train_regnet.rds")
fit_train_gbt <- read_rds("Trainfitdata/fit_train_gbt.rds")
fit_train_rf <- read_rds("Trainfitdata/fit_train_rf.rds")
fit_train_nnet <- read_rds("Trainfitdata/fit_train_nnet.rds")
```

Training Model Accuracy

In this next block of code, we will be fitting our best models to the training data set. Now we can visualize and compare how well our models performed.

```
# Now using the augment feature we can collect rmse values for each model to select which is best.

# linear regression model
lm_rmse_train <- augment(fit_train_linreg, car_data_train) %>%
  rmse(sellingprice, estimate=.pred) %>%
  select(.estimate)
# elastic net model
reg_net_rmse_train <- augment(fit_train_regnet, car_data_train) %>%
  rmse(sellingprice, estimate=.pred) %>%
  select(.estimate)
# gradient boosted tree model
gbt_rmse_train <- augment(fit_train_gbt, car_data_train) %>%
  rmse(sellingprice, estimate=.pred) %>%
  select(.estimate)
# random forest model
rf_rmse_train <- augment(fit_train_rf, car_data_train) %>%
  rmse(sellingprice, estimate=.pred) %>%
  select(.estimate)
# neural network model
nnet_rmse_train <- augment(fit_train_nnet, car_data_train) %>%
  rmse(sellingprice, estimate=.pred) %>%
  select(.estimate)

sellingprices_rmse_train <- c(lm_rmse_train$.estimate,
```

```
                              reg_net_rmse_train$.estimate,
                              gbt_rmse_train$.estimate,
                              rf_rmse_train$.estimate,
                              nnet_rmse_train$.estimate)
car_mod_names <- c("Linear Regression", "Elastic Net", "Gradient Boosted Tree", "Random Forest", "Neural
```

```
sellingprice_results <- tibble(Model = car_mod_names,
                               RMSE = sellingprices_rmse_train) %>%
  arrange(RMSE)

sellingprice_results
```

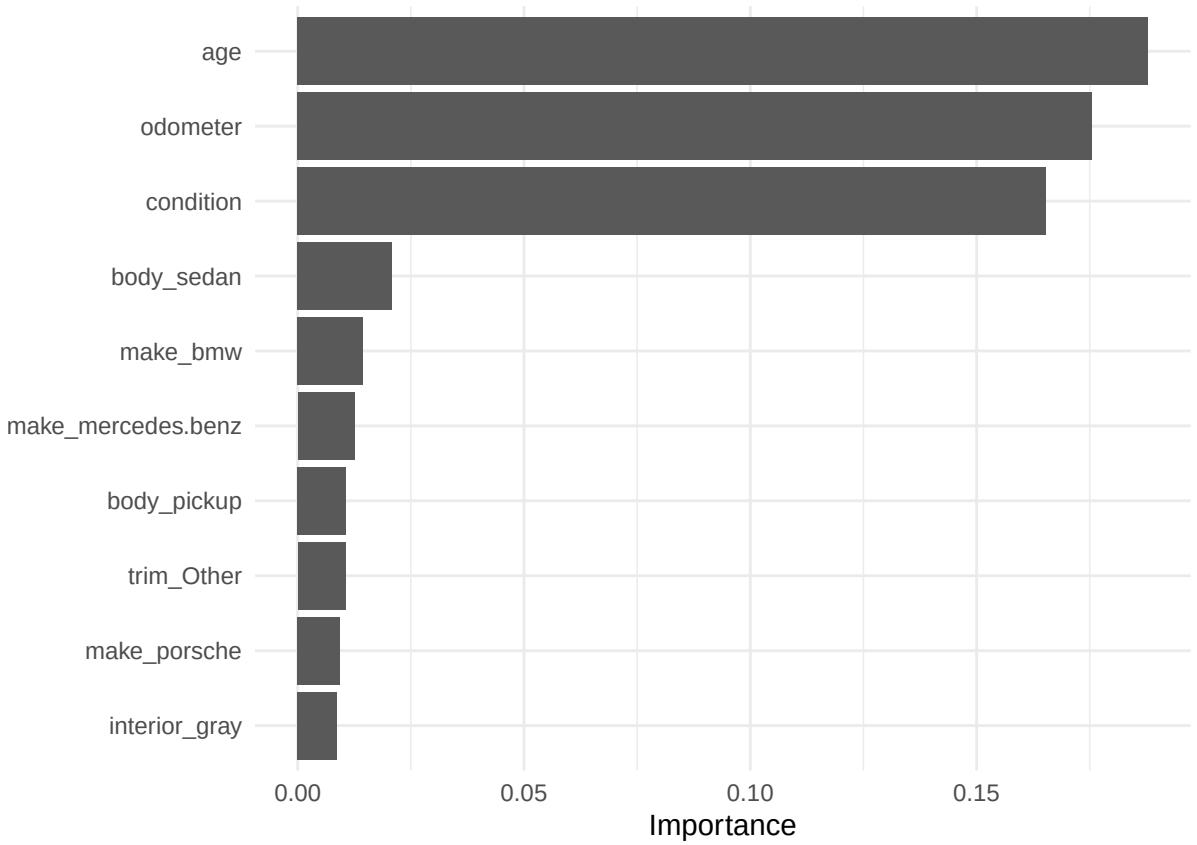
```
## # A tibble: 5 x 2
##   Model                  RMSE
##   <chr>                 <dbl>
## 1 Gradient Boosted Tree 4095.
## 2 Linear Regression     4284.
## 3 Elastic Net           4302.
## 4 Neural Network        5583.
## 5 Random Forest         6220.
```

From the table, gradient boosted tree performed the best out of the models with a RMSE value of 4094.508 After this was the linear regression model which is actually an interesting result since its the most flexible model with the least variance. The Linear regression model had a RMSE of 4283.508.

Useful Predictors

Now we can take a look at the gradient boosted tree's decisions in selecting which variables were most important by using `extract_fit_parsnip()` and the `vip()` plot.

```
# We can visualize the most important variables from the gradient boosted tree running its model using
fit_train_gbt %>% extract_fit_parsnip() %>%
  vip() +
  theme_minimal()
```



We notice that the most important predictors are `Age`, `Odometer`, and `Condition` for the gradient boosted tree. When comparing these results to our own intuition on what helps set the price of a vehicle, we should think about what year (or how old) the car is, how many miles has the car been driven, and lastly what kind of condition. If the car has been in a wreck, or the owner hasn't taken care of the vehicle, then the seller shouldn't expect to get as much value out of the vehicle if instead, it weren't wrecked or was perfectly maintained. If the car is newer, we'd expect the car to have a higher value because of depreciation effects. Using our intuition, we also expect a vehicle with higher mileage to come with more mechanical issues, and to be in a worse condition. Obviously as mentioned before, a vehicle that has a lower condition will have a lower price. Thus we can conclude the correlation between the three variables guides the model to predicting price.

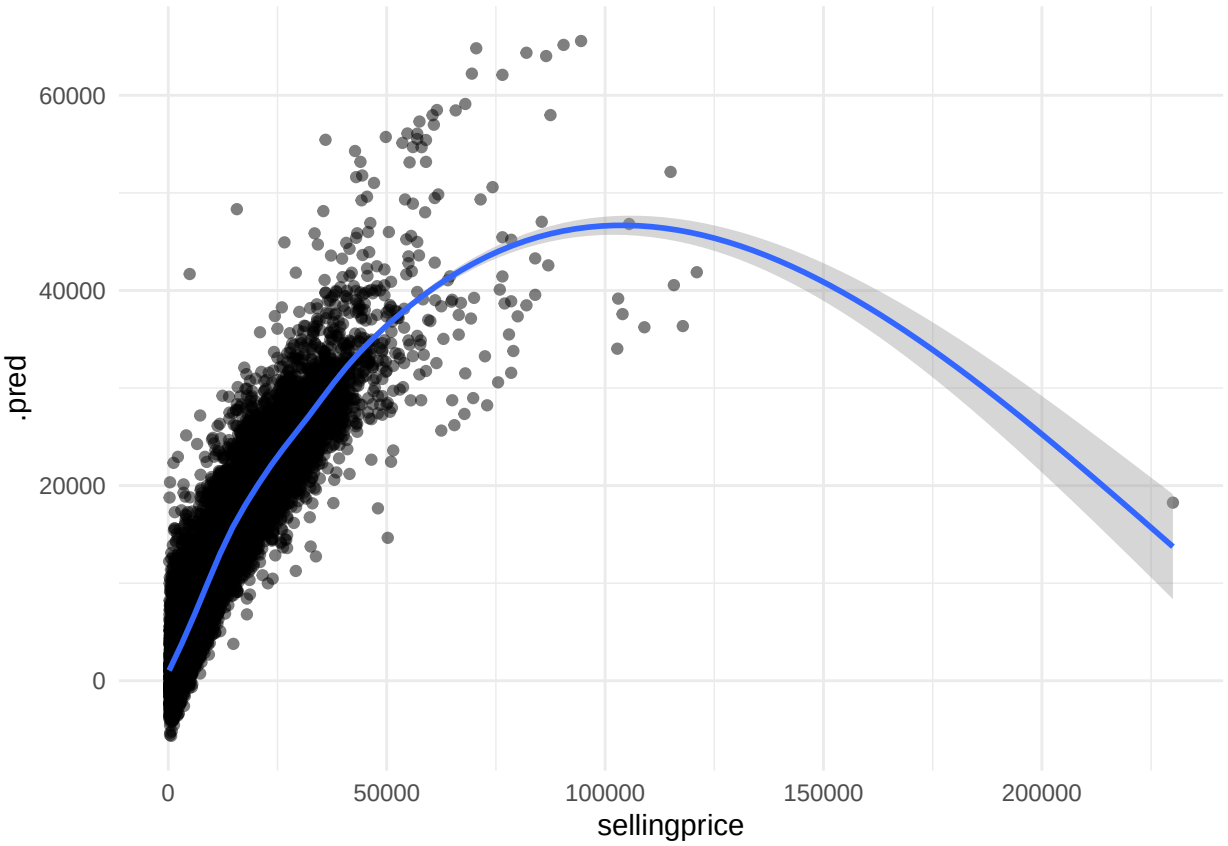
Testing the Best Model

Since we have the best model as the gradient boosted tree, we can now fit the gradient boosted tree to the car testing data set and get our final results.

```
final_gbt_test <- augment(fit_train_gbt, new_data = car_data_test)
rmse(final_gbt_test, truth = sellingprice, .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       4268.
```

```
final_gbt_test %>%
  ggplot(aes(x = sellingprice, y = .pred)) +
  geom_point(alpha = 0.5) +
  geom_smooth() +
  theme_minimal()
```



Conclusion

The final results of the gradient boosted tree were very interesting, but this is to be expected. From the results, we now know that predicting market prices is a very complex idea. From the scatter plot alone, we notice that most of the points follow a general regression line trend until we reach the cross section between the $\$50,000$ selling price mark on the x-axis and the $\$40,000$ prediction mark on the y-axis. We see the smoothed line changes and performs worse. This was an expected result because in the car world, there will always be unique cars that sell for more due to other factors. These include number of owners, modifications, and more. These unique cars are the outliers in the model that make it difficult to predict. As we can see on the right side of the plot, we have a car that sold for over $\$250,000$, but our model only predicted it to be less than $\$20,000$. But overall, the gradient boosted tree model performed well.

The model that performed the worse and that was the most interesting was the random forest. The data can be seen to be a more linear fit besides some outliers at higher values selling prices which is where the linear models seem to fall short on performance. Some thoughts on improvements to the performance of the model include having more information on the car, like owner history, and potentially having more information, giving a variable that has a binary response to if the vehicle has been in a wreck or not. I think in the future, adding a generalized additive model (GAM), could potentially do well. However because of the linearity, but with outliers, the gradient boosted tree performed as it should.

Moreover, in the future, I would like to explore more into more recent auction sites such as Bring-a-Trailer, or Cars&Bids, which are two of the biggest auctioning sites on the market for buying and selling cars. These sites have lots of information about the cars and also has a community of car enthusiast that can shift prices of the car due to the questions they ask the seller. They ask for information about the car to help buyers sway their decision to keep driving prices up.

Overall, I enjoyed my first machine learning project and will continue delving into new datasets, learning new

methods, and maybe even in a new programming languages. The exploration of neural networks, despite their complexity and my initial unfamiliarity, opens a new avenue for deep learning applications in market price prediction. My experience with this project has only deepened my enthusiasm for pursuing further studies and projects in the field.