Car Price Predictions

Hai Vu

February 14, 2023

TABLE OF CONTENTS

I. INTRODUCTION

This project aims to utilize the multivariate linear regression and random forest algorithms to predict the price of cars, given their characteristics. In this exercise, I will examine the significance of different predictors, their influence on the pricing of cars, and the accuracy of my predictive models. The data set used in this project is from: https://archive.ics.uci.edu/ml/datasets/Automobile.

II. ANALYSIS

1. Import data and perform data cleaning

First, we import the given car data set.

```
# Import data set
df_car <- read.csv("CarPrice_Data.csv")

# Overview of data set
str(df_car, width = 70, strict.width = "cut")
```

```
## 'data.frame':    205 obs. of  26 variables:
##  $ car_ID          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ symboling       : int  3 3 1 2 2 2 1 1 1 0 ...
##  $ CarName         : chr  "alfa-romero giulia" "alfa-romero stelvio"..
##  $ fueltype        : chr  "gas" "gas" "gas" "gas" ...
##  $ aspiration      : chr  "std" "std" "std" "std" ...
##  $ doornumber      : chr  "two" "two" "two" "four" ...
##  $ carbody         : chr  "convertible" "convertible" "hatchback" ""..
##  $ drivewheel      : chr  "rwd" "rwd" "rwd" "fwd" ...
##  $ enginelocation  : chr  "front" "front" "front" "front" ...
##  $ wheelbase       : num  88.6 88.6 94.5 99.8 99.4 ...
##  $ carlength       : num  169 169 171 177 177 ...
##  $ carwidth        : num  64.1 64.1 65.5 66.2 66.4 66.3 71.4 71.4 71..
##  $ carheight       : num  48.8 48.8 52.4 54.3 54.3 53.1 55.7 55.7 55..
##  $ curbweight      : int  2548 2548 2823 2337 2824 2507 2844 2954 30..
##  $ enginetype      : chr  "dohc" "dohc" "ohcv" "ohc" ...
##  $ cylindernumber  : chr  "four" "four" "six" "four" ...
##  $ enginesize      : int  130 130 152 109 136 136 136 136 131 131 ...
##  $ fuelsystem      : chr  "mpfi" "mpfi" "mpfi" "mpfi" ...
##  $ boreratio       : num  3.47 3.47 2.68 3.19 3.19 3.19 3.19 3.19 3...
##  $ stroke          : num  2.68 2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 3.4..
##  $ compressionratio: num  9 9 9 10 8 8.5 8.5 8.5 8.3 7 ...
##  $ horsepower      : int  111 111 154 102 115 110 110 110 140 160 ...
##  $ peakrpm         : int  5000 5000 5000 5500 5500 5500 5500 5500 55..
```

```
##  $ citympg          : int  21 21 19 24 18 19 19 19 17 16 ...
##  $ highwaympg       : int  27 27 26 30 22 25 25 25 20 22 ...
##  $ price            : num  13495 16500 16500 13950 17450 ...
```

The data set contains 205 records and 26 fields of data. Let's check for invalid values within the rows:

```
# Check for NAs
col_names <- colnames(df_car)
sapply(df_car, FUN = function(col_names) {
  table(is.na(df_car))
})
```

```
##           car_ID.FALSE        symboling.FALSE          CarName.FALSE
##                   5330                   5330                   5330
##         fueltype.FALSE       aspiration.FALSE       doornumber.FALSE
##                   5330                   5330                   5330
##          carbody.FALSE       drivewheel.FALSE   enginelocation.FALSE
##                   5330                   5330                   5330
##        wheelbase.FALSE        carlength.FALSE         carwidth.FALSE
##                   5330                   5330                   5330
##        carheight.FALSE       curbweight.FALSE       enginetype.FALSE
##                   5330                   5330                   5330
##   cylindernumber.FALSE       enginesize.FALSE       fuelsystem.FALSE
##                   5330                   5330                   5330
##        boreratio.FALSE           stroke.FALSE compressionratio.FALSE
##                   5330                   5330                   5330
##       horsepower.FALSE          peakrpm.FALSE          citympg.FALSE
##                   5330                   5330                   5330
##       highwaympg.FALSE            price.FALSE
##                   5330                   5330
```

```
# Check for blanks
sapply(df_car, FUN=function(col_names){
  is.element("",col_names)
})
```

```
##           car_ID        symboling          CarName         fueltype
##            FALSE            FALSE            FALSE            FALSE
##       aspiration       doornumber          carbody       drivewheel
##            FALSE            FALSE            FALSE            FALSE
##   enginelocation        wheelbase        carlength         carwidth
##            FALSE            FALSE            FALSE            FALSE
##        carheight       curbweight       enginetype   cylindernumber
```

```
##            FALSE            FALSE            FALSE            FALSE
##       enginesize       fuelsystem       boreratio           stroke
##            FALSE            FALSE            FALSE            FALSE
## compressionratio       horsepower          peakrpm          citympg
##            FALSE            FALSE            FALSE            FALSE
##       highwaympg            price
##            FALSE            FALSE
```

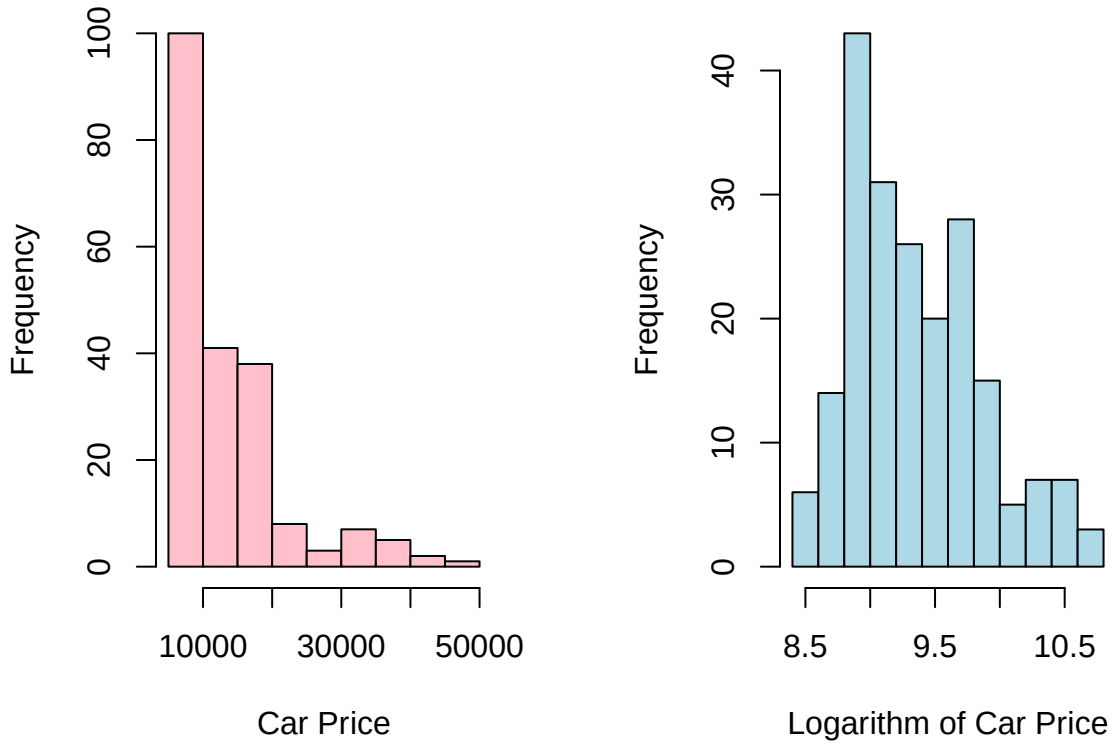
```
# Check for repeated values
paste("There are",nrow(df_car) - nrow(distinct(df_car)),
      "duplicated record(s) in the data set")
```

```
## [1] "There are 0 duplicated record(s) in the data set"
```

There are no blank values as well as repeated values in the data set. Next, we check the normality assumption for the dependent variable "price":

```
# Check for dependent variables normality
par(mfcol=c(1,2),mai=c(1,1,0.5,0.5))
hist(df_car$price,col="pink",main="",xlab="Car Price")
hist(log(df_car$price),col="lightblue",main="",xlab="Logarithm of Car
 ↪  Price")
mtext(expression(bold("Figure 1. Histograms of car price")),
      side = 3, line = -1.5, outer = TRUE, cex=1.1)
```

Figure 1. Histograms of car price



From figure 1, we can see that the distribution of the car price is heavily right-skewed. Therefore, we use log transform method to make the car price distribution more normal. Our dependent variable will now be the logarithm of car price. Next, we look at the descriptive statistics for different variables in the data set. To do this, I first remove "car_ID" since it is only a primary key in the data set, and then separate the numerical and categorical features:

```
# Separate numerical and categorical values:
df_car2 <- df_car[ , !(names(df_car) %in% "car_ID")] # remove car_ID
df_num <- select_if(df_car2, is.numeric)
df_char <- select_if(df_car2, is.character)
```

The table 1 below captures the descriptive statistics for the distribution of numerical features in the data set, including their count, mean, median, standard deviation, minimum and maximum values:

```
# Descriptive stats of the data set
num_stats <- describe(df_num)[,c('n','mean','median','sd','min','max')]
num_stats <- as.data.frame(num_stats)

num_stats <-
```

```
  num_stats %>%
  mutate_at(vars(mean,sd), funs(round(.,1)))

knitr::kable(num_stats, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 1. Descriptive statistics of the numerical
              ↪  variables") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

Table 1. Descriptive statistics of the numerical variables

	n	mean	median	sd	min	max
symboling	205	0.8	1.00	1.2	-2.00	3.00
wheelbase	205	98.8	97.00	6.0	86.60	120.90
carlength	205	174.0	173.20	12.3	141.10	208.10
carwidth	205	65.9	65.50	2.1	60.30	72.30
carheight	205	53.7	54.10	2.4	47.80	59.80
curbweight	205	2555.6	2414.00	520.7	1488.00	4066.00
enginesize	205	126.9	120.00	41.6	61.00	326.00
boreratio	205	3.3	3.31	0.3	2.54	3.94
stroke	205	3.3	3.29	0.3	2.07	4.17
compressionratio	205	10.1	9.00	4.0	7.00	23.00
horsepower	205	104.1	95.00	39.5	48.00	288.00
peakrpm	205	5125.1	5200.00	477.0	4150.00	6600.00
citympg	205	25.2	24.00	6.5	13.00	49.00
highwaympg	205	30.8	30.00	6.9	16.00	54.00
price	205	13276.7	10295.00	7988.9	5118.00	45400.00

The table 2 below captures the descriptive statistics for the categorical features, including their count and number of unique values:

```
# Show descriptive statistics of the categorical values
char_uniq <- as.matrix(lengths(lapply(df_char,unique)))
char_n <- describe(df_char)[,c('n')]
char_stats <- cbind(char_uniq,char_n)
colnames(char_stats) <- c("Unique","Count")
char_stats <- as.data.frame(char_stats)
char_stats <- char_stats[order(-char_stats$Unique),]

knitr::kable(char_stats, "latex",
             booktabs = T,longtable=T,linesep = "",
```

```
            caption="Table 2. Descriptive statistics of the categorical
             ↪  variables") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

Table 2. Descriptive statistics of the categorical variables

	Unique	Count
CarName	147	205
fuelsystem	8	205
enginetype	7	205
cylindernumber	7	205
carbody	5	205
drivewheel	3	205
fueltype	2	205
aspiration	2	205
doornumber	2	205
enginelocation	2	205

From table 2 above, we can see that for `205` observations, `147` categories of "CarName" are perhaps too many for meaningful analysis. Thus, I'm looking to modify this column so that it contains less number of unique values. The original "CarName" include the brand name, followed by some model names or series (e.g.: `toyota corona, toyota corolla liftback, bmw x3`). Thus, I will transform this column to include only the brand names of the cars and try to reduce the number of unique categories:

- Transform "CarName":

```
# Remove serial numbers and lower the case:
df_car2$CarName <- trimws(tolower(gsub( " .*$", "", df_car$CarName)))

# Group similar brands
df_car2$CarName <- if_else(df_car2$CarName ==
 ↪  "toyouta","toyota",df_car2$CarName)
df_car2$CarName <- if_else(df_car2$CarName ==
 ↪  "maxda","mazda",df_car2$CarName)
df_car2$CarName <- if_else(df_car2$CarName == "vw" | df_car2$CarName ==
 ↪  "vokswagen",
                           "volkswagen",df_car2$CarName)
df_car2$CarName <- if_else(df_car2$CarName ==
 ↪  "porcshce","porsche",df_car2$CarName)
sort(table(df_car2$CarName))
```

```
##
##     mercury     renault alfa-romero   chevrolet      jaguar       isuzu
##           1           2           3           3           3           4
##     porsche        saab        audi    plymouth         bmw       buick
##           5           6           7           7           8           8
##       dodge     peugeot       volvo      subaru  volkswagen       honda
##           9          11          11          12          12          13
##  mitsubishi       mazda      nissan      toyota
##          13          17          18          32
```

```
# Further reduce number of CarName categories by grouping values with low
 ↪  count
`%notin%` <- Negate(`%in%`)
df_car2$CarName <- if_else(df_car2$CarName %notin%
 ↪  names(tail(sort(table(df_car2$CarName)),5)),"others",df_car2$CarName)

# Show current brand names value distribution
sort(table(df_car2$CarName))
```

```
##
##      honda mitsubishi      mazda     nissan     toyota     others
##         13         13         17         18         32        112
```

From 147 different car names, there are now only 6 unique car brands, which makes it much easier to conduct our analysis in the next steps. Next, I want to examine other categorical variables with high number of unique values as well, including "fuelsystem", "enginetype", and "cyclindernumber". My intention is to group categories with low counts into bigger categories, thus, reducing the total number of unique values overall.

- Transform "fuelsystem":

```
# Examine fuel system value distribution
sort(table(df_car2$fuelsystem))
```

```
##
##  mfi spfi 4bbl spdi 1bbl  idi 2bbl mpfi
##    1    1    3    9   11   20   66   94
```

```
# Regroup fuel system
df_car2$fuelsystem <- if_else(df_car$fuelsystem %in%
 ↪  names(head(sort(table(df_car2$fuelsystem)),6)),"others",df_car2$fuelsystem)

# Value distribution after regrouped
sort(table(df_car2$fuelsystem))
```

```
##
## others   2bbl   mpfi
##     45     66     94
```

By grouping “mfi”, “spfi”, “4bbl”, “spdi”, “1bbl”, and “idi”, I have managed to decrease the number of unique groups from 8 to 3.

- Transform “enginetype”:

```
# Examine engine type value distribution
sort(table(df_car2$enginetype))
```

```
##
## dohcv rotor  dohc     l  ohcv  ohcf   ohc
##     1     4    12    12    13    15   148
```

```
# Regroup engine type
df_car2$enginetype <- if_else(df_car$enginetype %notin%
 ↪  c("ohc"),"not_ohc",df_car2$enginetype)

# Value distribution after regrouped
sort(table(df_car2$enginetype))
```

```
##
## not_ohc     ohc
##      57     148
```

By grouping engine types that are different from “ohc”, which has the highest count, I have managed to decrease the number of unique groups from 7 to 2.

- Transform “cylindernumber”:

```
# Examine cylinder number value distribution
sort(table(df_car2$cylindernumber))
```

```
##
##  three twelve    two  eight   five    six   four
##      1      1      4      5     11     24    159
```

```
# Convert cylinder number into numeric data type
df_car2$cylindernumber <- as.numeric(sapply(df_car2$cylindernumber,
 ↪  words_to_numbers))

# Value distribution after regrouped
sort(table(df_car2$cylindernumber))
```

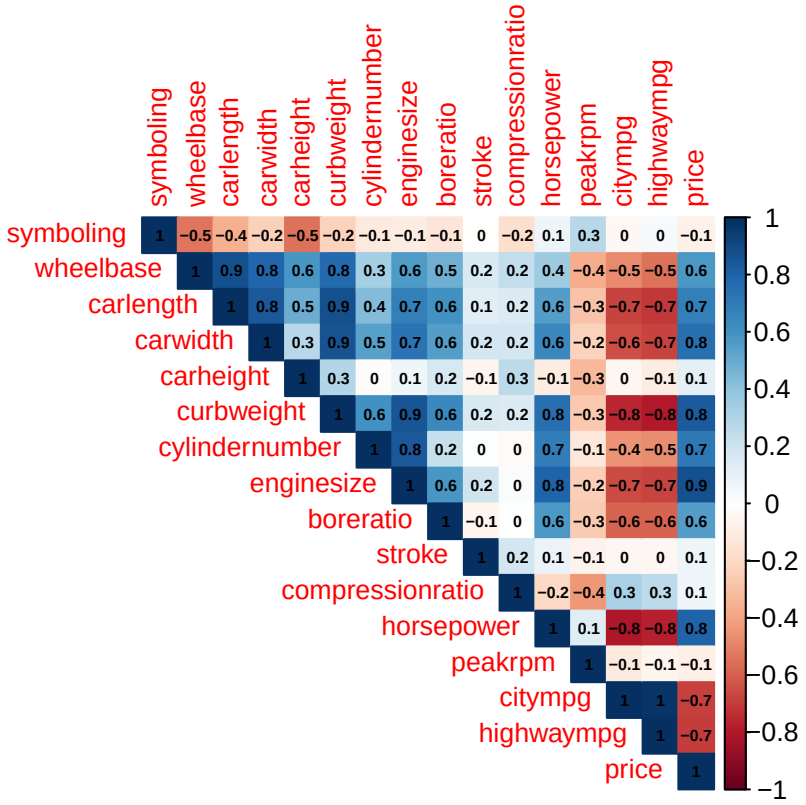
```
##
##    3  12   2   8   5   6   4
##    1   1   4   5  11  24 159
```

For the "cylindernumber" feature, I have noticed that their values are just numbers spelled out in words. Thus, using the function words_to_numbers() found on https://github.com/fsingletonthorn/words_to_numbers, I am able to convert these characters into numerical values.

Next, let's examine the correlation matrix between numerical variables in the data set:

```
# Correlation plot:
cors <- cor(select_if(df_car2, is.numeric), use="pairwise")
p_val <- cor.mtest(select_if(df_car2, is.numeric), conf.level=0.95)$p
corrplot(cors,type="upper",mar=c(0,0,1.5,0),method="color",
         number.cex=0.5,tl.cex=0.8,
         #col=brewer.pal(8,"RdYlBu"),
         addCoef.col = 1,number.digits = 1,
         title="Figure 2. Correlation matrix of numerical values")
```

Figure 2. Correlation matrix of numerical values



As seen from figure 2, there seems to be a number of highly correlated pairs of variables in the data set, such as wheel base & car length, or horse power & city mpg. When conducting

linear regression analysis, we should pay attention to these pairs of variables so as to avoid multi-collinearity events. We can try to remove a number of variables from our list of predictors. After examining figure 2, I decided to remove 7 variables, which are "curbweight", "highwaympg", "citympg", "carlength", "enginesize", "cylindernumber", and "carwidth":

```
# Remove correlated features
df_car3 <- df_car2[,!(names(df_car2) %in% c("curbweight", "highwaympg",
 ↪  "citympg", "carlength", "enginesize", "carwidth", "cylindernumber"))]
```

2. Create training & testing sets and construct linear regression models

First, I'd like to scale the numerical variables and convert categorical features into the factor data type:

```
# Convert categorical to factor
for (i in colnames(select_if(df_car3,is.character))){
  df_car3[,i] <- as.factor(df_car3[,i])
}

# Re-level factors of "CarName" & "fuelsystem"
df_car3$CarName <- factor(df_car3$CarName, levels =
 ↪  c("others","honda","mitsubishi","mazda","nissan","toyota"))
df_car3$fuelsystem <- factor(df_car3$fuelsystem, levels =
 ↪  c("others","2bbl","mpfi"))

# Define min max scaling function
minmax_scale <- function(x, na.rm = TRUE) {
    return((x- min(x)) /(max(x)-min(x)))
}

# Normalize numerical values in each set (except for the dependent
 ↪  variable)
for (i in colnames(select_if(df_car3[,!(names(df_car3) %in%
 ↪  "price")],is.numeric))){
  df_car3[,i] <- minmax_scale(df_car3[,i])
}

# Show current data set
str(df_car3, width = 70, strict.width = "cut")
```

```
## 'data.frame':    205 obs. of  18 variables:
##  $ symboling       : num  1 1 0.6 0.8 0.8 0.8 0.6 0.6 0.6 0.4 ...
##  $ CarName         : Factor w/ 6 levels "others","honda",..: 1 1 1 1..
```

```
##  $ fueltype        : Factor w/ 2 levels "diesel","gas": 2 2 2 2 2 2 ..
##  $ aspiration      : Factor w/ 2 levels "std","turbo": 1 1 1 1 1 1 1..
##  $ doornumber      : Factor w/ 2 levels "four","two": 2 2 2 1 1 2 1 ..
##  $ carbody         : Factor w/ 5 levels "convertible",..: 1 1 3 4 4 ..
##  $ drivewheel      : Factor w/ 3 levels "4wd","fwd","rwd": 3 3 3 2 1..
##  $ enginelocation  : Factor w/ 2 levels "front","rear": 1 1 1 1 1 1 ..
##  $ wheelbase       : num  0.0583 0.0583 0.2303 0.3848 0.3732 ...
##  $ carheight       : num  0.0833 0.0833 0.3833 0.5417 0.5417 ...
##  $ enginetype      : Factor w/ 2 levels "not_ohc","ohc": 1 1 1 2 2 2..
##  $ fuelsystem      : Factor w/ 3 levels "others","2bbl",..: 3 3 3 3 ..
##  $ boreratio       : num  0.664 0.664 0.1 0.464 0.464 ...
##  $ stroke          : num  0.29 0.29 0.667 0.633 0.633 ...
##  $ compressionratio: num  0.125 0.125 0.125 0.1875 0.0625 ...
##  $ horsepower      : num  0.263 0.263 0.442 0.225 0.279 ...
##  $ peakrpm         : num  0.347 0.347 0.347 0.551 0.551 ...
##  $ price           : num  13495 16500 16500 13950 17450 ...
```

After cleaning the data set and scaling the numerical values, we can go head and split the data set into training and testing sets. For this assignment, I will use the 80/20 split ratio:

```
# Split 80/20
set.seed(123)
trainRatio <- createDataPartition(df_car3$price,p=0.8,list=F,times=1)
df_train <- df_car3[trainRatio,]
df_test <- df_car3[-trainRatio,]

# Dimension of each set
dim_traintest <- data.frame(dim(df_train),dim(df_test))
row.names(dim_traintest) <- c("No. of Rows","No. of Columns")
colnames(dim_traintest) <- c("Training Set","Testing Set")
knitr::kable(dim_traintest, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 3. Dimension of training set and testing set")
              ↪ %>%
  kable_styling(position = "center",latex_options = c("repeat_header"))
```

Table 3. Dimension of training set and testing set

	Training Set	Testing Set
No. of Rows	165	40
No. of Columns	18	18

Next, using the training set, we construct the linear regression models:

```r
# Construct linear model:
lm_model1 <- lm(data=df_train,log(price)~.)
summary(lm_model1)
```

```
##
## Call:
## lm(formula = log(price) ~ ., data = df_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44653 -0.10762 -0.00021  0.10762  0.40206
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        10.19882    0.49554  20.581  < 2e-16 ***
## symboling           0.19451    0.08728   2.229 0.027457 *
## CarNamehonda        0.08005    0.09021   0.887 0.376410
## CarNamemitsubishi  -0.14003    0.06993  -2.002 0.047193 *
## CarNamemazda        0.05608    0.05554   1.010 0.314350
## CarNamenissan       0.01645    0.05215   0.315 0.752961
## CarNametoyota      -0.13411    0.04323  -3.102 0.002331 **
## fueltypegas        -0.88671    0.43180  -2.054 0.041910 *
## aspirationturbo    -0.07780    0.05392  -1.443 0.151353
## doornumbertwo      -0.08662    0.04559  -1.900 0.059488 .
## carbodyhardtop     -0.34481    0.10080  -3.421 0.000822 ***
## carbodyhatchback   -0.42132    0.08302  -5.075 1.23e-06 ***
## carbodysedan       -0.36015    0.08788  -4.098 7.07e-05 ***
## carbodywagon       -0.38393    0.10132  -3.789 0.000225 ***
## drivewheelfwd      -0.18858    0.07603  -2.480 0.014325 *
## drivewheelrwd      -0.02715    0.08325  -0.326 0.744806
## enginelocationrear  0.45764    0.14828   3.086 0.002450 **
## wheelbase           1.23404    0.18006   6.854 2.20e-10 ***
## carheight          -0.26732    0.12521  -2.135 0.034526 *
## enginetypeohc       0.19691    0.04375   4.501 1.42e-05 ***
## fuelsystem2bbl     -0.04118    0.07574  -0.544 0.587527
## fuelsystemmpfi      0.05929    0.07284   0.814 0.417053
## boreratio          -0.22698    0.10108  -2.246 0.026325 *
## stroke             -0.24695    0.12540  -1.969 0.050924 .
## compressionratio   -0.80624    0.46546  -1.732 0.085485 .
## horsepower          1.90214    0.15250  12.473  < 2e-16 ***
## peakrpm            -0.20222    0.10111  -2.000 0.047452 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.1616 on 138 degrees of freedom
## Multiple R-squared:  0.9136, Adjusted R-squared:  0.8973
## F-statistic: 56.11 on 26 and 138 DF,  p-value: < 2.2e-16
```

My first linear regression model uses every variables in the clean data set. To check for multi-collinearity issues with my model, I calculate the Variance Inflation Factor for each feature in my regression model. The VIF shows what percentage the variance is inflated by for each coefficient if there was no correlation with other predictors (Glen, 2015). The VIF values can be interpreted as follow:

- VIF close to 1 means no correlation
- VIF between 1 and 5 is considered moderately correlated, or an acceptable level
- VIF more than 5 is considered highly-correlated. In this case, we should try to remove one or more of those correlated variables to avoid multi-collinearity

To remove multi-collinearity cases, my goal is to keep the VIFs of all features below 5 in this case. The following table shows the VIF values for each independent variables in the first linear regression model:

```
options(scipen=999)

# Check for multi-collinearity using VIF
df_vif <- as.data.frame(car::vif(lm_model1))
colnames(df_vif) <- "VIF"
df_vif <- cbind(Variables = rownames(df_vif), df_vif)[,c(1,2)]
df_vif <- df_vif[order(-df_vif$VIF),]
rownames(df_vif) <- NULL
knitr::kable(df_vif, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 4. Variance Inflation Factor of variables in
              ↪  first model") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

Table 4. Variance Inflation Factor of variables in first model

Variables	VIF
fueltype	114.435550
compressionratio	92.334153
fuelsystem	14.297043
CarName	12.962957
carbody	6.802721
wheelbase	6.398170
carheight	4.079814

Table 4. Variance Inflation Factor of variables in first model *(continued)*

Variables	VIF
horsepower	4.051552
drivewheel	3.546272
doornumber	3.216936
symboling	2.948137
aspiration	2.731498
boreratio	2.434894
peakrpm	2.431303
enginetype	2.328775
stroke	2.223702
enginelocation	1.662590

From table 4, we can see that "fueltype", "compressionratio", "CarName", and "carbody" are the features with the highest VIFs. Thus, I decided to create a second linear regression model with less independent features:

```
# Create new linear regression model with less features:
lm_model2 <- lm(data=df_train,log(price)~. -fueltype -compressionratio
 ↪  -CarName -carbody)
summary(lm_model2)
```

```
##
## Call:
## lm(formula = log(price) ~ . - fueltype - compressionratio - CarName -
##     carbody, data = df_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.59043 -0.10247 -0.00324  0.10201  0.62297
##
## Coefficients:
##                   Estimate Std. Error t value             Pr(>|t|)
## (Intercept)        8.81996    0.15889  55.509 < 0.0000000000000002 ***
## symboling          0.28179    0.09107   3.094             0.002357 **
## aspirationturbo   -0.02956    0.05036  -0.587             0.558119
## doornumbertwo     -0.11030    0.04446  -2.481             0.014212 *
## drivewheelfwd     -0.16056    0.08429  -1.905             0.058721 .
## drivewheelrwd      0.05162    0.09154   0.564             0.573641
## enginelocationrear 0.62709    0.15711   3.991             0.000103 ***
## wheelbase          1.15461    0.17770   6.497        0.00000000115 ***
## carheight         -0.10698    0.11514  -0.929             0.354348
## enginetypeohc      0.13984    0.04634   3.018             0.002995 **
```

```
## fuelsystem2bbl     -0.14671    0.05131  -2.860             0.004851 **
## fuelsystemmpfi      0.00110    0.05279   0.021             0.983403
## boreratio          -0.19126    0.11124  -1.719             0.087609 .
## stroke             -0.18731    0.12853  -1.457             0.147136
## horsepower          1.72353    0.16721  10.308 < 0.0000000000000002 ***
## peakrpm            -0.25793    0.09311  -2.770             0.006315 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1888 on 149 degrees of freedom
## Multiple R-squared:  0.8728, Adjusted R-squared:  0.8599
## F-statistic: 68.13 on 15 and 149 DF,  p-value: < 0.00000000000000022
```

The VIFs of the new list of features are calculated in table 5 below:

```
options(scipen=999)

# Check for multi-collinearity using VIF
df_vif2 <- as.data.frame(car::vif(lm_model2))
colnames(df_vif2) <- "VIF"
df_vif2 <- cbind(Variables = rownames(df_vif2), df_vif2)[,c(1,2)]
df_vif2 <- df_vif2[order(-df_vif2$VIF),]
rownames(df_vif2) <- NULL
knitr::kable(df_vif2, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 5. Variance Inflation Factor of variables in
              ↪  second model") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

Table 5. Variance Inflation Factor of variables in second model

Variables	VIF
wheelbase	4.570008
fuelsystem	3.588618
horsepower	3.571896
drivewheel	2.744296
carheight	2.530199
symboling	2.353812
doornumber	2.243824
boreratio	2.162429
enginetype	1.916201
aspiration	1.746940

Table 5. Variance Inflation Factor of variables in second model *(continued)*

Variables	VIF
stroke	1.713298
peakrpm	1.512062
enginelocation	1.368848

As shown in table 5, the VIFs of all features in the second model are below 5 now. Thus, we can safely assume that this model is free of multi-collinearity issues. Next, I want to compare the first model with the second model to see how the removal of independent features affect the models' training performance. The 2 models will be evaluated using the number of features used, r-squared, adjusted r-squared, Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE):

```
# Calculate r-squared, adjusted r-squared, rmse, mape
lm1_train_score <- rbind(format(nrow(df_vif),nsmall=0),
                         round(summary(lm_model1)$r.squared,4),
                         round(summary(lm_model1)$adj.r.squared,4),
                         round(rmse(df_train$price,
                          ↪  exp(lm_model1$fitted.values)),4),
                         round(mape(df_train$price,
                          ↪  exp(lm_model1$fitted.values)),4))
lm2_train_score <- rbind(format(nrow(df_vif2),nsmall=0),
                         round(summary(lm_model2)$r.squared,4),
                         round(summary(lm_model2)$adj.r.squared,4),
                         round(rmse(df_train$price,
                          ↪  exp(lm_model2$fitted.values)),4),
                         round(mape(df_train$price,
                          ↪  exp(lm_model2$fitted.values)),4))

# Create table that compare models
compare_lm <- as.data.frame(cbind(lm1_train_score,lm2_train_score))
colnames(compare_lm) <- c("First LR Model","Second LR Model")
rownames(compare_lm) <- c("# Features Used","R-Squared","Adjusted
 ↪  R-Squared","RMSE","MAPE")
knitr::kable(compare_lm, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 6. Linear Regression Models Training
              ↪  Performance") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

Table 6. Linear Regression Models Training Performance

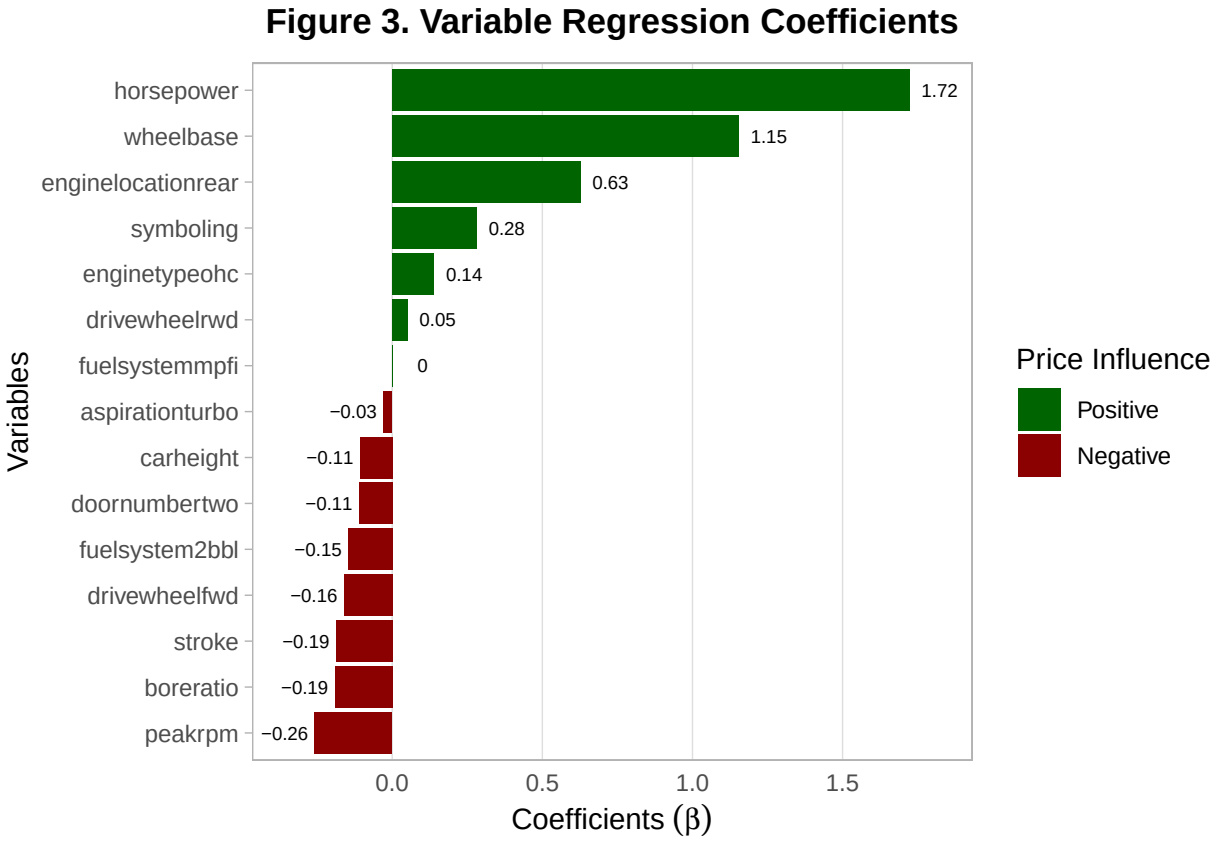
	First LR Model	Second LR Model
# Features Used	17	13
R-Squared	0.9136	0.8728
Adjusted R-Squared	0.8973	0.8599
RMSE	2675.7357	3289.9615
MAPE	0.1219	0.1386

From table 6, both models seem to have relatively high accuracy as represented by their r-squared and adjusted r-squared statistics. The model 1 r-squared and adjusted r-squared are `0.9136` and `0.8973`, respectively. These metrics are slightly higher compared to model 2, which have its r-squared and adjusted r-squared values at `0.8728` and `0.8599`, respectively. This means that model 1 is able to explain `91.36%` of the variances in the car prices, compared to only `85.99%` in model 2. In addition, we can see that with fewer independent features to work with, the second linear model generated a slightly lower r-squared, adjusted r-squared and higher RMSE compared to the first linear model. However, it's important to remember that the first model can be biased since the multi-collinearity issues may cause the models to over-prioritize similar predictors. Moreover, models that require less number of predictors can be considered more robust when it comes to predictions because they need less training data input. I will continue my analysis using the second linear regression model.

To observe the influence each variable has on price, we examine the regression coefficients of each predictor:

```
# Create data frame containing coefficients and p-values:
step_signif <-
  cbind(as.data.frame(summary(lm_model2)$coefficients[,1]), # Coefficients
        exp(as.data.frame(summary(lm_model2)$coefficients[,1])), #
         ↪ Exponential of coefficients
        as.data.frame(summary(lm_model2)$coefficients[,4])) # P-values
colnames(step_signif) <- c("Coefficients","Exp_Coefficients","P_values")
step_signif <- step_signif[!(row.names(step_signif) %in% "(Intercept)"),] #
 ↪  remove intercept
step_signif$Influence <- if_else(step_signif$Coefficients <
 ↪  0,"Negative","Positive") # group by price influence
step_signif <- step_signif[order(-step_signif$Exp_Coefficients),,drop=F] #
 ↪  order by exp coef
step_signif <- cbind(Variables = rownames(step_signif), step_signif) #
 ↪  create column from index
rownames(step_signif) <- NULL # remove index

# Show regression coefficients of features
ggplot(data=step_signif, aes(x=Coefficients,
                             y=reorder(Variables,Coefficients),
```

```
                          fill=reorder(Influence,-Coefficients),
                          label=round(Coefficients,2))) +
  geom_bar(stat="identity") +
  ggtitle(expression(bold("Figure 3. Variable Regression Coefficients"))) +
  ylab("Variables") + xlab(expression("Coefficients" ~ (beta))) +
  theme_light() +
  theme(
    legend.position="right",
    plot.title = element_text(hjust = 0.5),
    #panel.grid.major.x = element_blank(),
    panel.grid.minor.x = element_blank(),
    panel.grid.major.y = element_blank(),
    #panel.border = element_blank(),
    #axis.ticks.y = element_blank(),
    axis.ticks.x = element_blank()) +
  guides(fill=guide_legend(title="Price Influence")) +
  geom_text(aes(x = Coefficients + 0.1 * sign(Coefficients)), size = 2.5) +
  scale_fill_manual(values=c("darkgreen","darkred"))
```



From figure 3, it seems that the top 3 most influential variables on the car price are "horsepower", "wheelbase" and "enginelocation". Among these 3, "horsepower" has the

greatest positive influence of the car prices, with its coefficient value at `1.72`. It's important to keep in mind that the current dependent variable is the natural logarithm of the car prices and not the car prices themselves, thus, for better interpretability, we need to convert these regression coefficients (denoted β) into the percentage increase/decrease in price by using the formula: $\beta * 100\%$. This means that each unit increase in "horsepower" is associated with a $\beta_{horsepower} * 100\% =$ `172.35%` change in car prices, or a `72.35%` increase in car prices. On the contrary, each unit increase in "peakrpm" (the car's peak round per minute) is associated with a $\beta_{peakrpm} * 100\% =$ `-25.79%` change in car prices, or a `74.21%` decrease in car prices.

3. Create training & testing sets and construct random forest models

In this section, I will employ the random forest algorithm for regression to help make the car price predictions. Random Forest utilizes methods such as bootstrap sampling and feature sampling (or bagging), i.e. row sampling and column sampling. Therefore, Random Forest is generally not affected by multi-collinearity too much since it is picking different set of features for different models and of course every model sees a different set of data points. However, there is still chances of highly-correlated features getting picked up together, and when that happens we will see some trace of it in the feature importance. This is because when features have similar effects or there is a relation in between the features, it can be difficult to rank the relative importance of features (Raj, 2019). For this, I intend to create 2 different models that use two different training data sets, one before the correlated features are removed and one after:

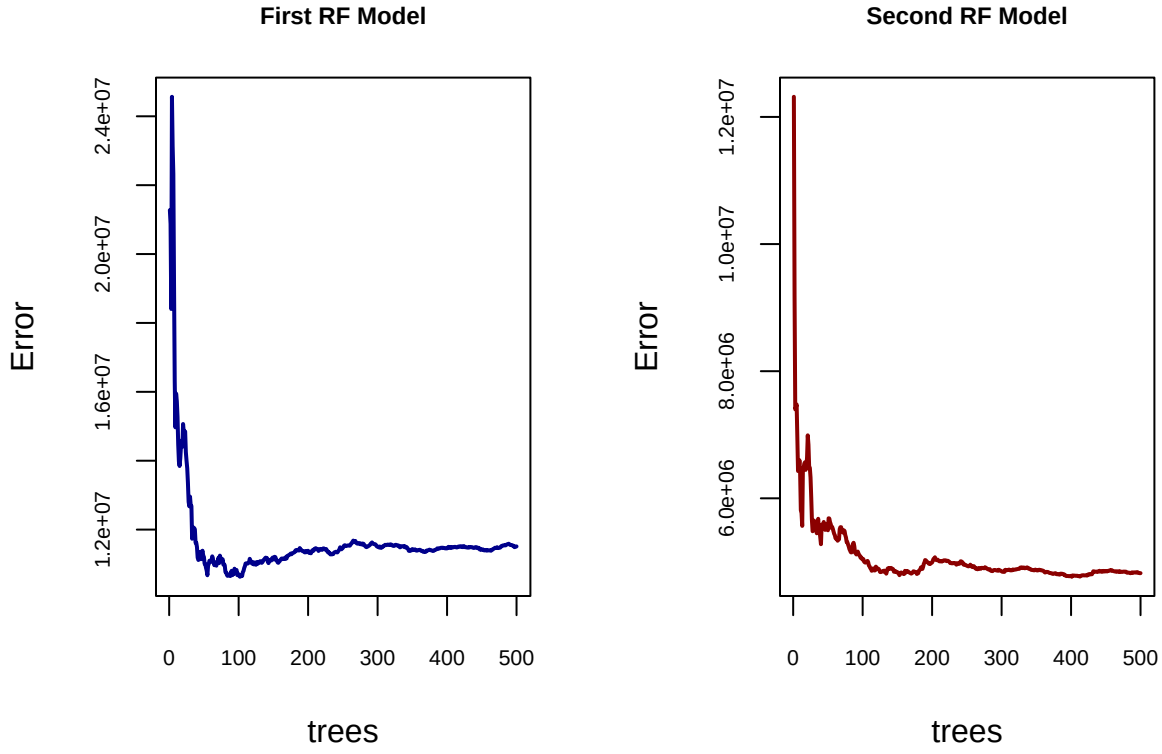
```
# Convert categorical to factor for data set before features removal
for (i in colnames(select_if(df_car2,is.character))){
  df_car2[,i] <- as.factor(df_car2[,i])
}

df_car2$CarName <- factor(df_car2$CarName, levels =
 ↪  c("others","honda","mitsubishi","mazda","nissan","toyota"))
df_car2$fuelsystem <- factor(df_car2$fuelsystem, levels =
 ↪  c("others","2bbl","mpfi"))

# Split 80/20 for data set before features removal
set.seed(123)
trainRatio2 <- createDataPartition(df_car2$price,p=0.8,list=F,times=1)
df_train2 <- df_car2[trainRatio2,]
df_test2 <- df_car2[-trainRatio2,]

# Fit 2 random forest models
rf_model1 = randomForest(price ~ . -fueltype -compressionratio -CarName
 ↪  -carbody -curbweight -highwaympg -citympg -carlength -enginesize
 ↪  -carwidth -cylindernumber, data=df_train2, ntree=500, proximity=T,
 ↪  importance=T, nPerm=2) # without correlated features
```

```
rf_model2 = randomForest(price ~ .,data=df_train2, ntree=500, proximity=T,
 ↪  importance=T, nPerm=2) # with correlated features
```

This random forest examines the performances of 500 trees to produce the best model. Typically, the bagging method will randomly select about one-third of all the available independent features to train a sample decision tree. The first model randomly chooses 4 out of 13 independent variables at each split when creating sample trees and roughly 83.56% of the variance is explained by the model. The second model randomly picks 8 out of 24 independent features at each split when creating sample trees and roughly 92.63% of the variance is explained by this model. The figure below helps visualize the models' improvements in terms of error between observations and predictions by the number of trees examined:

```
# Plot forest model error
options(scipen=0)
par(mai=c(1,1,0.8,0.3), mfcol=c(1,2))
plot(rf_model1,main="\nFirst RF Model",
     lwd=2,cex.axis=0.7,cex.main=0.75, col="darkblue")
plot(rf_model2,main="\nSecond RF Model",
     lwd=2,cex.axis=0.7,cex.main=0.75, col="darkred")
mtext(expression(bold("Figure 4. Error reduction vs the number of trees")),
      side = 3, line = -1.8, outer = TRUE, cex=1.1)
```

Figure 4. Error reduction vs the number of trees



From figure 4, we can observe that both models are able to achieve great reduction in errors as the number of trees trained increases. Still, the reduction in errors for the first random forest model is slightly less stable compared to the second one. Next, let's compare the two models' training performance using r-squared, adjusted r-squared, RMSE, and MAPE:

```
# Compare training performance

# R2 <- 1 - (sum((actual-predicted)^2)/sum((actual-mean(actual))^2))
rf_model1_r2 <- 1 - (sum((df_train$price-rf_model1$predicted)^2)/
                     sum((df_train$price-mean(rf_model1$predicted))^2))
rf_model2_r2 <- 1 - (sum((df_train2$price-rf_model2$predicted)^2)/
                     sum((df_train2$price-mean(rf_model2$predicted))^2))

# Adj_R2 <- 1-((1-r_squared)*(n-1)/(n-k-1))
n_train <- nrow(df_train) # number of observations (same for both models)
k_train <- nrow(df_vif2) # number of predictors
rf_model1_adj_r2 <- 1-((1-rf_model1_r2)*(n_train-1)/(n_train-k_train-1))

k_train2 <- ncol(df_train2)-1 # number of predictors
rf_model2_adj_r2 <- 1-((1-rf_model2_r2)*(n_train-1)/(n_train-k_train2-1))
```

```r
# Combine performance scores into a data frame
rf_model1_train <- rbind(k_train,
                         rf_model1_r2,
                         rf_model1_adj_r2,
                         rmse(df_train2$price, rf_model1$predicted),
                         mape(df_train2$price, rf_model1$predicted))

rf_model2_train <- rbind(k_train2,
                         rf_model2_r2,
                         rf_model2_adj_r2,
                         rmse(df_train2$price, rf_model2$predicted),
                         mape(df_train2$price, rf_model2$predicted))

compare_model <- as.data.frame(cbind(compare_lm$`Second LR Model`,
                                     round(rf_model1_train,4),
                                     round(rf_model2_train,4)))
colnames(compare_model) <- c("Linear Regression Model","First RF
 ↪  Model","Second RF Model")
rownames(compare_model) <- c("# Features Used","R-Squared","Adjusted
 ↪  R-Squared","RMSE","MAPE")
knitr::kable(compare_model, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 7. Linear Regression vs Random Forest Training
              ↪  Performance") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

Table 7. Linear Regression vs Random Forest Training Performance

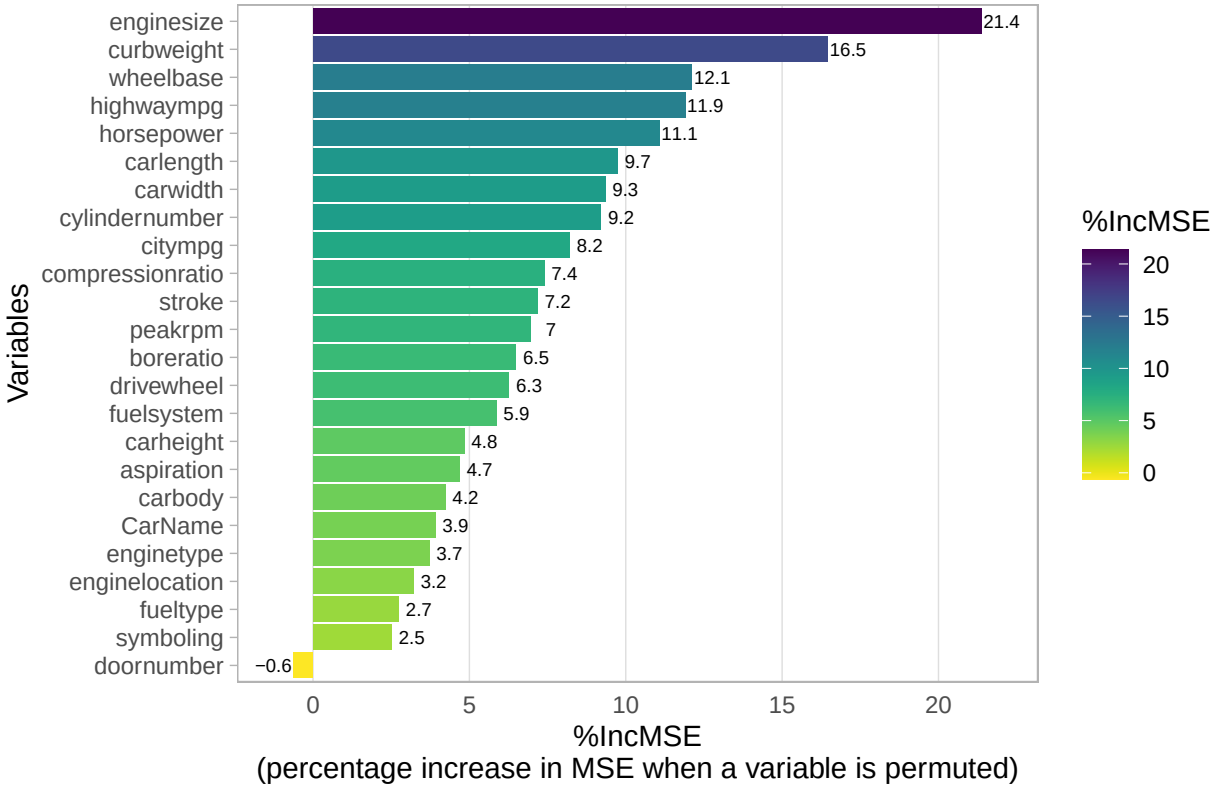
	Linear Regression Model	First RF Model	Second RF Model
# Features Used	13	13	24
R-Squared	0.8728	0.8221	0.9254
Adjusted R-Squared	0.8599	0.8068	0.9127
RMSE	3289.9615	3392.9834	2195.9931
MAPE	0.1386	0.1409	0.1128

From table 7, all 3 models seem to have relatively high accuracy as represented by their r-squared and adjusted r-squared statistics. The second random forest model performs the best out of the 3, which have its r-squared and adjusted r-squared values at `0.9254` and `0.9127`, respectively. This means that this model is able to explain `92.54%` of the variances in the car prices. Thus, I will proceed to use *the second random forest model* for future analysis. However, since the number of predictors used in the second random forest model is greater than the other 2 models, this model is less robust even though its prediction power remains the strongest.

Next, I'd like to examine to variable importance determined by this model. As mentioned earlier, in random forest, the training sets are sampled via a method called bagging, where not all but only part of the records are sampled, with replacement. After that, only a handful of features (in this case it is 8 features) out of all the available features are chosen to grow a sample decision tree. From that, the variable importance can be calculated as the percentage increase in mean squared error (MSE) of the predictions as a result of randomly removing each independent variable. The higher the increase, the more important the feature is in estimating the dependent variable.

```
# RF feature importance
var_imp <- as.data.frame(importance(rf_model2))
colnames(var_imp) <- c("PercIncMSE","IncNodePurity")
var_imp <- var_imp[order(var_imp$PercIncMSE,decreasing = T),]
var_imp <- rownames_to_column(var_imp,var="Variables")

# Create barplot to show feature importance
ggplot(var_imp, aes(x=Variables, y=PercIncMSE)) +
  geom_col(aes(x=reorder(Variables,PercIncMSE),
               y=PercIncMSE,fill=PercIncMSE),show.legend = T) +
  theme_light() +
  coord_flip() +
  ggtitle(expression(bold("Figure 5. Random Forest Variable Importance"))) +
  scale_fill_viridis(discrete=F, option="viridis", direction=-1) +
  labs(fill="%IncMSE") +
  theme(
    legend.position="right",
    plot.title = element_text(hjust = 0.5),
    #panel.grid.major.x = element_blank(),
    panel.grid.minor.x = element_blank(),
    panel.grid.major.y = element_blank(),
    #panel.border = element_blank(),
    #axis.ticks.y = element_blank(),
    axis.ticks.x = element_blank()) +
  geom_text(aes(x = Variables, y=PercIncMSE + 0.65*sign(PercIncMSE),
   ↪  label=round(PercIncMSE,1)), size = 2.5) +
  ylab(paste0("%IncMSE","\n","(percentage increase in MSE when a variable is
   ↪  permuted)"))
```

Figure 5. Random Forest Variable Importance



From figure 5, we can see that "enginesize", "curbweight", "wheelbase", "highwaympg", and "horsepower" are the top 5 most important predictors, in respective order, when it comes to forecasting car prices. This is rather similar to what was discovered in figure 3 for the linear regression model, where both "wheelbase" and "horsepower" seem to be the most influential factors affecting car prices. Interestingly, feature "doornumber" has negative importance score, which means that the removal of this feature actually improves the model accuracy.

4. Create predictions using the testing set and compare models' performances

In this part, we use the testing set to determine the accuracy of our best prediction models created so far. Similarly, these models will be evaluated using the number of features used, r-squared, adjusted r-squared, RMSE, and MAPE:

```
# Create predictions for both partitioned sets
lm_test_pred <- predict(lm_model2, df_test, interval="none")
rf_test_pred <- predict(rf_model2, df_test2, interval="none")

# Calculate R2
lm_test_r2 <- 1 - (sum((df_test$price-exp(lm_test_pred))^2)/
```

```
                    sum((df_test$price-mean(exp(lm_test_pred)))^2))
rf_test_r2 <- 1 - (sum((df_test2$price-rf_test_pred)^2)/
                    sum((df_test2$price-mean(rf_test_pred))^2))

# Calculate Adjusted_R2
n_test <- nrow(df_test2) # number of observations (same for both models)
k_test_lm <- nrow(df_vif2) # number of predictors for linear model
k_test_rf <- ncol(df_test2)-1 # number of predictors for rf model
lm_test_adj_r2 <- 1-((1-lm_test_r2)*(n_test-1)/(n_test-k_test_lm-1)) # adj
 ↪  r2 for linear model
rf_test_adj_r2 <- 1-((1-rf_test_r2)*(n_test-1)/(n_test-k_test_rf-1)) # adj
 ↪  r2 for rf model

# Calculate RMSE
lm_test_rmse <- rmse(df_test$price, exp(lm_test_pred))
rf_test_rmse <- rmse(df_test2$price, rf_test_pred)

# Calculate MAPE
lm_test_mape <- mape(df_test$price, exp(lm_test_pred))
rf_test_mape <- mape(df_test2$price, rf_test_pred)

# Combine values into a data frame
lm_model_test <- rbind(format(k_test_lm,nsmall=0),
                       round(lm_test_r2,4),
                       round(lm_test_adj_r2,4),
                       round(lm_test_rmse,4),
                       round(lm_test_mape,4))

rf_model_test <- rbind(format(k_test_rf,nsmall=0),
                       round(rf_test_r2,4),
                       round(rf_test_adj_r2,4),
                       round(rf_test_rmse,4),
                       round(rf_test_mape,4))

compare_summary <- as.data.frame(cbind(lm_model_test,rf_model_test))
colnames(compare_summary) <- c("Linear Regression","Random Forest")
rownames(compare_summary) <- c("# Features Used","R-Squared","Adjusted
 ↪  R-Squared","RMSE","MAPE")
knitr::kable(compare_summary, "latex",
             booktabs = T,longtable=T,linesep = "",
             caption="Table 8. Linear Regression vs Random Forest Testing
              ↪  Performance") %>%
  kable_styling(position = "center",latex_options = c("striped",
   ↪  "repeat_header"))
```

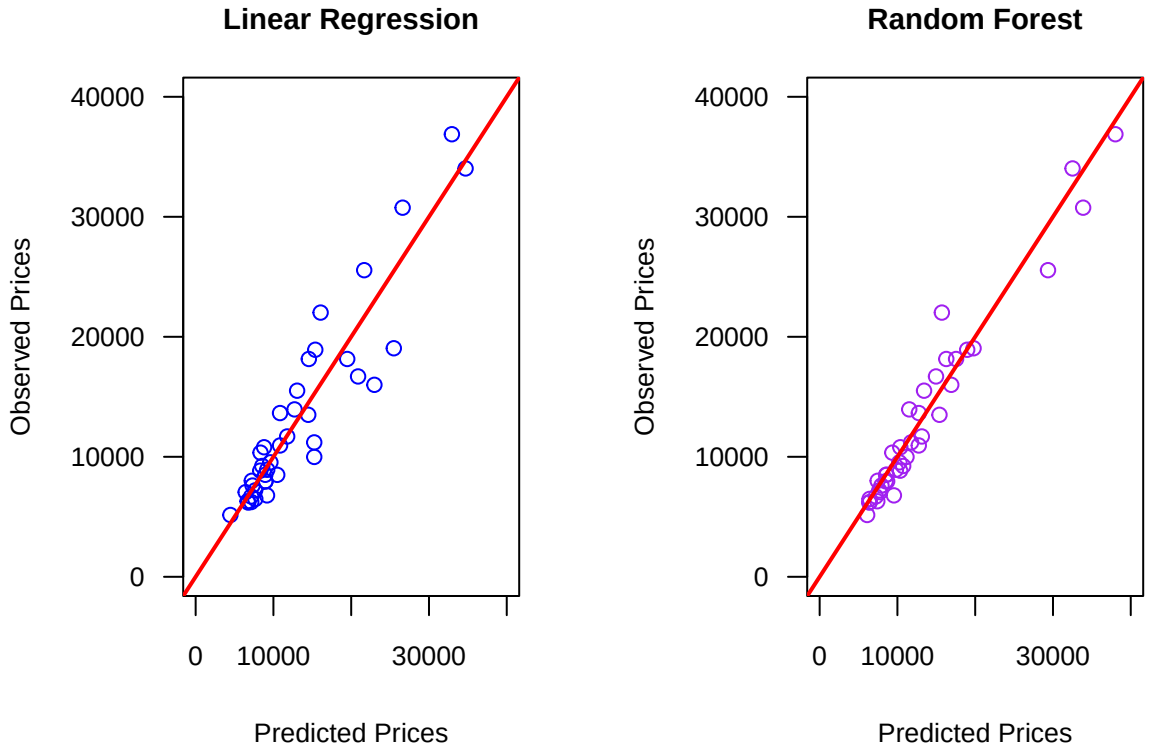
Table 8. Linear Regression vs Random Forest Testing Performance

	Linear Regression	Random Forest
# Features Used	13	24
R-Squared	0.8703	0.9504
Adjusted R-Squared	0.8054	0.8709
RMSE	2758.7155	1707.3803
MAPE	0.1502	0.0986

Table 8 summarizes the different metrics representing the predictive power of our linear regression and random forest models. Based on the testing performance, I'd say that both models produce pretty high accuracy, with r-squared being roughly between 0.87 to 0.95. Compared to the training performance shown in table 7, the random forest model's predictive power improves while the linear regression model's predictive power worsen. This shows that the random forest model does not over fit with the training data and has great capability at generating predictions using newly introduced data. Let's visualize the 2 models' fit using scatter plots:

```
# Plot predictions vs observations
par(mai=c(1,1.1,0.8,0.4),mfcol=c(1,2))

plot(x=exp(lm_test_pred),y=df_test$price,col="blue",
     ylab=paste0("Observed Prices","\n"),xlab="Predicted Prices",
     xlim=c(0,40000),ylim=c(0,40000),las=1,
     main=paste0("\n","Linear Regression"), cex.main=0.9, cex.axis=0.8,
      ↪  cex.lab=0.8)
abline(a=0,b=1,col="red",lwd=2)

plot(x=rf_test_pred,y=df_test2$price,col="purple",
     ylab=paste0("Observed Prices","\n"),xlab="Predicted Prices",
     xlim=c(0,40000),ylim=c(0,40000),las=1,
     main=paste0("\n","Random Forest"), cex.main=0.9, cex.axis=0.8,
      ↪  cex.lab=0.8)
abline(a=0,b=1,col="red",lwd=2)

mtext(expression(bold("Figure 6. Car price observations vs predictions
 ↪  across models")),
      side = 3, line = -1.8, outer = TRUE, cex=1.1)
```

Figure 6. Car price observations vs predictions across models



As seen from figure 6, the data points are densely scattered around the red diagonal line, indicating that the observed values are relatively close to the predicted values for both models. However, for the random forest model, the purple points are more aligned with the diagonal line, showing its higher competency in generating accurate predictions.

III. CONCLUSIONS

Both the linear regression and random forest are two popular techniques for regression predictions. In this project, they both perform relatively well in predicting prices of cars, but the random forest model still outperforms the linear regression one. Here are some pros and cons of each approach:

- Pros of Linear Regression (LR):
  - Interpretability: LR provides coefficients that can be interpreted to understand the relationship between the input variables and the output variable
  - Simplicity: LR is a simple and easy-to-understand algorithm that can be implemented quickly
  - Efficiency: LR can be faster than more complex models like random forests, especially for large datasets

  - Stability: LR tends to be more stable than random forests, as the results are less dependent on the specific sample used to build the model

- Cons of Linear Regression:

  - Linearity assumption: LR assumes a linear relationship between the input variables and the output variable, which may not be true in all cases
  - Limited flexibility: LR has limited flexibility to capture complex patterns in the data, especially when there are non-linear relationships between the variables
  - Sensitivity to outliers: LR can be sensitive to outliers, which can distort the relationship between the variables
  - Model assumptions: LR relies on certain assumptions about the data, such as normality and homoscedasticity, which may not hold in all cases

- Pros of Random Forest (RF):

  - Flexibility: RF is a highly flexible algorithm that can capture complex non-linear relationships between the input variables and the output variable
  - Robustness: RF is less sensitive to outliers and can handle missing data well, making it a more robust model
  - Accuracy: RF is often more accurate than linear regression, especially for complex datasets with many input variables
  - No assumptions: RF does not make any assumptions about the data, making it a more versatile algorithm

- Cons of Random Forest:

  - Complexity: RF can be a complex algorithm to understand and implement, especially for beginners
  - Overfitting: RF can be prone to overfitting, especially if the number of trees is high or the data is noisy
  - Interpretability: RF does not provide easily interpretable coefficients, making it difficult to understand the relationship between the input variables and the output variable
  - Training time: RF can be slower to train than linear regression, especially for large datasets

In summary, linear regression is a simple, efficient, and interpretable algorithm that can provide stable results, but it has limited flexibility and can be sensitive to outliers. Random forest, on the other hand, is a more complex and flexible algorithm that can provide more accurate results, but it can be prone to overfitting and is less interpretable. The choice between these two algorithms depends on the specific problem, the size and complexity of the dataset, and the priorities of the user (e.g., interpretability vs. accuracy).

IV. REFERENCES

Glen, S. (2015, September 21). *Variance inflation factor.* Statistics How To. Retrieved February 14, 2023, from https://www.statisticshowto.com/variance-inflationfactor/

Potters, C. (2021, April 30). *R-squared vs. adjusted R-squared: What's the difference?* Investopedia. Retrieved February 14, 2023, from https://www.investopedia.com/ask/answers/012615/whats-difference-betweenrsquared-and-adjusted-rsquared.asp

Raj, S. (2019, August 9). *Effects of multi-collinearity in logistic regression, SVM, RF.* Medium. Retrieved February 14, 2023, from https://medium.com/@raj5287/effects-of-multi-collinearity-in-logistic-regression-svm-rf-af6766d91f1b