Practicum 2

Jiaming Xu

Problem 1:

1 & 2:

```
library(tidyverse)
library(klaR)
library(gmodels)
library(broom)
library(psych)
```

Loading data from the local file since url needs update every time for accessing:

```
df <- read.csv("german.csv")
glimpse(df)
```

```
## Rows: 1,000
## Columns: 21
## $ checking_balance     <chr> "< 0 DM", "1 - 200 DM", "unknown", "< 0 DM", "< 0~
## $ months_loan_duration <int> 6, 48, 12, 42, 24, 36, 24, 36, 12, 30, 12, 48, 12~
## $ credit_history       <chr> "critical", "repaid", "critical", "repaid", "dela~
## $ purpose              <chr> "radio/tv", "radio/tv", "education", "furniture",~
## $ amount               <int> 1169, 5951, 2096, 7882, 4870, 9055, 2835, 6948, 3~
## $ savings_balance      <chr> "unknown", "< 100 DM", "< 100 DM", "< 100 DM", "<~
## $ employment_length    <chr> "> 7 yrs", "1 - 4 yrs", "4 - 7 yrs", "4 - 7 yrs",~
## $ installment_rate     <int> 4, 2, 2, 2, 3, 2, 3, 2, 2, 4, 3, 3, 1, 4, 2, 4, 4~
## $ personal_status      <chr> "single male", "female", "single male", "single m~
## $ other_debtors        <chr> "none", "none", "none", "guarantor", "none", "non~
## $ residence_history    <int> 4, 2, 3, 4, 4, 4, 4, 2, 4, 2, 1, 4, 1, 4, 4, 2, 4~
## $ property             <chr> "real estate", "real estate", "real estate", "bui~
## $ age                  <int> 67, 22, 49, 45, 53, 35, 53, 35, 61, 28, 25, 24, 2~
## $ installment_plan     <chr> "none", "none", "none", "none", "none", "none", "~
## $ housing              <chr> "own", "own", "own", "for free", "for free", "for~
## $ existing_credits     <int> 2, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2~
## $ default              <int> 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 2, 1, 2, 1~
## $ dependents           <int> 1, 1, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ telephone            <chr> "yes", "none", "none", "none", "none", "yes", "no~
## $ foreign_worker       <chr> "yes", "yes", "yes", "yes", "yes", "yes", "yes", ~
## $ job                  <chr> "skilled employee", "skilled employee", "unskille~
```

```
summary(df)
```

```
##   checking_balance   months_loan_duration credit_history       purpose
##   Length:1000        Min.   : 4.0         Length:1000        Length:1000
##   Class :character   1st Qu.:12.0         Class :character   Class :character
##   Mode  :character   Median :18.0         Mode  :character   Mode  :character
```

```
##                       Mean   :20.9
##                       3rd Qu.:24.0
##                       Max.   :72.0
##      amount       savings_balance    employment_length  installment_rate
##  Min.   :  250   Length:1000        Length:1000        Min.   :1.000
##  1st Qu.: 1366   Class :character   Class :character   1st Qu.:2.000
##  Median : 2320   Mode  :character   Mode  :character   Median :3.000
##  Mean   : 3271                                         Mean   :2.973
##  3rd Qu.: 3972                                         3rd Qu.:4.000
##  Max.   :18424                                         Max.   :4.000
##  personal_status    other_debtors      residence_history   property
##  Length:1000        Length:1000        Min.   :1.000     Length:1000
##  Class :character   Class :character   1st Qu.:2.000     Class :character
##  Mode  :character   Mode  :character   Median :3.000     Mode  :character
##                                        Mean   :2.845
##                                        3rd Qu.:4.000
##                                        Max.   :4.000
##       age        installment_plan     housing           existing_credits
##  Min.   :19.00   Length:1000        Length:1000        Min.   :1.000
##  1st Qu.:27.00   Class :character   Class :character   1st Qu.:1.000
##  Median :33.00   Mode  :character   Mode  :character   Median :1.000
##  Mean   :35.55                                         Mean   :1.407
##  3rd Qu.:42.00                                         3rd Qu.:2.000
##  Max.   :75.00                                         Max.   :4.000
##     default      dependents      telephone         foreign_worker
##  Min.   :1.0   Min.   :1.000   Length:1000        Length:1000
##  1st Qu.:1.0   1st Qu.:1.000   Class :character   Class :character
##  Median :1.0   Median :1.000   Mode  :character   Mode  :character
##  Mean   :1.3   Mean   :1.155
##  3rd Qu.:2.0   3rd Qu.:1.000
##  Max.   :2.0   Max.   :2.000
##      job
##  Length:1000
##  Class :character
##  Mode  :character
##
##
##
```

3:

Since default variable only has 1 and two, I can use ifelse to replace 1 with 'good', 2 with 'bad'.

```
unique(df$default)
```

```
## [1] 1 2
```

```
df1 <- mutate(df, default = ifelse(default == 1, "Good", "Bad"))
```

4:

Fix seed as 7, make index for train. df1_train and df1_test contain same ratio of good/bad.

```
set.seed(7)
train <- sample(nrow(df1), floor(0.7*nrow(df1)))
df1_train <- df1[train, ]
```

```
df1_test <-  df1[-train, ]
table(df1_train$default)
```

```
##
##  Bad Good
##  205  495
```

```
table(df1_test$default)
```

```
##
##  Bad Good
##   95  205
```

5:

1. Select Status of existing checking account (1), Credit history (3), Purpose (4), Credit amount (5), Installment rate in percentage of disposable income (8), Personal Status (9), Property (12), Age (13), Number of existing credits at this bank (16) and Job (21), save new data frame as df2.
2. Change age and amount variable to categorical using cut function. 6 bins for age and 5 bins for amount variable. Change installment_rate and existing_credits as factor.
3. Divide df2_lgr into train and test subsets, use NaiveBayes function to predict, save to default.nb.
4. Get prediction results as vector using predict function.

```
df2 <- df1[, c(1,3,4,5,8,9,12,13,16,17,21)]

df2_lgr <- df2 %>%
  mutate( age = cut(age, breaks = 6, labels = c(1,2,3,4,5,6))) %>%
  mutate( amount = cut(amount, breaks = 5, labels = c(1,2,3,4,5))) %>%
  mutate( installment_rate = factor(installment_rate)) %>%
  mutate( existing_credits = factor(existing_credits))

df2_train <- df2_lgr[train, ]
df2_test <- df2_lgr[-train, ]

default.nb <- NaiveBayes(factor(default) ~ ., data = df2_train )

predict.nb <- predict(default.nb, df2_test[, -10])
```

6:

Using CrossTable to build up a confusion matrix. 55 FP and 20 FN, 225 correct prediction over 300 testing set, overall 75% accuracy. 55 FP is relatively high to put bank at risk if lending money to risky users.

```
CrossTable(predict.nb$class, df2_test$default, dnn = c('predicted','actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
```

```
##
##
## Total Observations in Table:  300
##
##
##              | actual
##    predicted |       Bad |      Good | Row Total |
## -------------|-----------|-----------|-----------|
##          Bad |        40 |        20 |        60 |
##              |    23.211 |    10.756 |           |
##              |     0.667 |     0.333 |     0.200 |
##              |     0.421 |     0.098 |           |
##              |     0.133 |     0.067 |           |
## -------------|-----------|-----------|-----------|
##         Good |        55 |       185 |       240 |
##              |     5.803 |     2.689 |           |
##              |     0.229 |     0.771 |     0.800 |
##              |     0.579 |     0.902 |           |
##              |     0.183 |     0.617 |           |
## -------------|-----------|-----------|-----------|
## Column Total |        95 |       205 |       300 |
##              |     0.317 |     0.683 |           |
## -------------|-----------|-----------|-----------|
##
##
```

7:

1. Change Good to 1, Bad to 0 for applying logistic regression
2. Define train and test subsets.
3. Use glm function to apply Logistic Regression, using binomial regression.
4. Take a look of prediction results.

```r
df3 <- df2 %>%
  mutate( default = ifelse( default == 'Bad', 0, 1))

df3_train <- df3[ train,]
df3_test <- df3[ -train,]

default.lr <- glm(default ~ ., data = df3_train, family = 'binomial')
table(augment(default.lr, newdata = df3_test, type.predict = 'response')$.fitted > 0.5)
```

```
##
## FALSE  TRUE
##    53   247
```

```r
table(df3_test$default)
```

```
##
##   0   1
##  95 205
```

8:

1. Use augment with type.predict = 'response' to get response variable prediction, then use ifelse to make results as binomial variables.
2. Use CrossTable to generate the confusion table.

The results show that the overall accuracy is 224/300 ~ 75% which is identical to NaiveBayes prediction. Also similar as NaiveBayes, FP = 59 among testing dataset which is also worth to consider.

```
default.lr.prediction <- ifelse(augment(default.lr, newdata = df3_test, type.predict = 'response')$.fit
CrossTable(default.lr.prediction, df3_test$default, dnn = c('predicted','actual'))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  300
##
##
##              | actual
##    predicted |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |        36 |        17 |        53 |
##              |    22.003 |    10.196 |           |
##              |     0.679 |     0.321 |     0.177 |
##              |     0.379 |     0.083 |           |
##              |     0.120 |     0.057 |           |
## -------------|-----------|-----------|-----------|
##            1 |        59 |       188 |       247 |
##              |     4.721 |     2.188 |           |
##              |     0.239 |     0.761 |     0.823 |
##              |     0.621 |     0.917 |           |
##              |     0.197 |     0.627 |           |
## -------------|-----------|-----------|-----------|
## Column Total |        95 |       205 |       300 |
##              |     0.317 |     0.683 |           |
## -------------|-----------|-----------|-----------|
##
##
```

9, 10:

Decision Tree can only applied to categorical variables, while regression tree and model trees can be applied to continuous variables. However, regression tree do not use linear regression methods they make predictions based on the average value of examples that reach a leaf. Therefore, continuous variables are used for regression trees, and categorified data used for decision trees.

In my opinion, these two trees don't have too many differences therefore provide similar accuracy. The advantage of regression tree is categorification of continuous data is not necessary, therefore I prefer regression tree for less data preparation process.

```
library(rpart)
library(C50)

default.rp <- rpart(default ~ ., data = df2[train, ])
default.c50 <- C5.0(factor(default) ~. , data = df2_lgr[train, ])

predict.rp <- predict(default.rp ,df2[-train, ])
predict.c50 <- predict(default.c50, df2_lgr[-train, ])
```

11:

CrossTable is used here. As we can see, two trees don't give different predictions, regression trees provide silightly better results.

```
CrossTable(ifelse(predict.rp[,1] > .5, 'Bod', "Good"), df2_lgr[-train, ]$default)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  300
##
##
##                                             | df2_lgr[-train, ]$default
## ifelse(predict.rp[, 1] > 0.5, "Bod", "Good") |       Bad |      Good | Row Total |
## ---------------------------------------------|-----------|-----------|-----------|
##                                          Bod |        29 |        18 |        47 |
##                                              |    13.389 |     6.205 |           |
##                                              |     0.617 |     0.383 |     0.157 |
##                                              |     0.305 |     0.088 |           |
##                                              |     0.097 |     0.060 |           |
## ---------------------------------------------|-----------|-----------|-----------|
##                                         Good |        66 |       187 |       253 |
##                                              |     2.487 |     1.153 |           |
##                                              |     0.261 |     0.739 |     0.843 |
##                                              |     0.695 |     0.912 |           |
##                                              |     0.220 |     0.623 |           |
## ---------------------------------------------|-----------|-----------|-----------|
##                                 Column Total |        95 |       205 |       300 |
##                                              |     0.317 |     0.683 |           |
## ---------------------------------------------|-----------|-----------|-----------|
##
##
```

```r
CrossTable(predict.c50, df2_lgr[-train, ]$default)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  300
##
##
##              | df2_lgr[-train, ]$default
##  predict.c50 |       Bad |      Good | Row Total |
## -------------|-----------|-----------|-----------|
##          Bad |        21 |        16 |        37 |
##              |     7.355 |     3.409 |           |
##              |     0.568 |     0.432 |     0.123 |
##              |     0.221 |     0.078 |           |
##              |     0.070 |     0.053 |           |
## -------------|-----------|-----------|-----------|
##         Good |        74 |       189 |       263 |
##              |     1.035 |     0.480 |           |
##              |     0.281 |     0.719 |     0.877 |
##              |     0.779 |     0.922 |           |
##              |     0.247 |     0.630 |           |
## -------------|-----------|-----------|-----------|
## Column Total |        95 |       205 |       300 |
##              |     0.317 |     0.683 |           |
## -------------|-----------|-----------|-----------|
##
##
```

12:

1. Build getmode function and predictCreditRisk function. predictCreditRisk can take dataset as input.
2. Decide the final prediction by finding the mode of three predictions.

```r
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}

predictCreditRisk <- function(x){
  x.lgr <- x

  # turn age and amount to categorical variables by using bins predefined. For
  # categorical functions NaiveBayes and rpart.
  age.bins <- seq(18.9, 75.1, length.out=7)
```

```
  amount.bins <- seq(min(df2$amount), max(df2$amount), length.out=6)

  x.lgr$age <- factor(sapply(x$age, function(x){sum(x > age.bins)}))
  x.lgr$amount <- factor(sapply(x$amount, function(x){sum(x > amount.bins)}))

  x.lgr <- x.lgr %>%
    mutate(installment_rate = factor(installment_rate)) %>%
    mutate(existing_credits = factor(existing_credits))

  set.seed(7)
  predict.nb <- predict(default.nb, x.lgr)$class %>% as.character()
  predict.rp <- ifelse(predict(default.rp, x)[,1 ] > .5,
                       'Bad', 'Good') %>% as.character()
  predict.c50 <- predict(default.c50, x.lgr) %>% as.character()
  predict.all <- cbind(predict.nb,predict.rp,predict.c50)
  apply(predict.all,1, getmode)
}

predictCreditRisk(df2[1:10,])
```

```
##  [1] "Good" "Good" "Good" "Good" "Bad"  "Good" "Good" "Good" "Good" "Good"
```

13:

Assuming amount = 5000, installment_rate = 1, age = 47, other missing info replaced by the mode of that variable. Prediction shows this user is good for a loan.

```
new_obj <- data.frame( 'age' = 47, 'personal_status' = 'single male',
                       'credit_history' = 'critical', 'property' = "real estate",
                       'purpose' = 'car (new)', 'checking_balance' = 'unknown',
                       'amount' = 5000, 'installment_rate' = 1, 'existing_credits' = getmode(df2$existi

predictCreditRisk(new_obj)
```

```
## [1] "Good"
```

Problem 2:

1:

1. Loading data, select all continuous variables and two categorical variables, num_of_doors and num_of_cylinders.
2. Replace missing values by NA. Change the name of variables. Change written number to numeric of number_of_doors and num_of_cylinders. Change all columns to numeric type.

```
df <- read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data", col_na
```

```
## Rows: 205 Columns: 26
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (16): X2, X3, X4, X5, X6, X7, X8, X9, X15, X16, X18, X19, X20, X22, X23,...
## dbl (10): X1, X10, X11, X12, X13, X14, X17, X21, X24, X25
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
cars.df <- df[, c(2,6,10:14,16,17,19:26)]
colnames(cars.df) <- c("normalized-losses", "num_of_doors", "wheel-base",
                       "length","width","height","curb-weight","num_of_cylinders",
                       "engine-size","bore","stroke","compression-ratio","horsepower",
                       "peak-rpm","city-mpg","highway-mpg","price")
cars.df[cars.df == '?'] <- NA
cars.df <- cars.df %>%
  mutate( num_of_cylinders = str_replace_all(num_of_cylinders,
                                          c("two"='2',"three"='3',"four"='4',
                                            "five"='5', "six" = '6',
                                            "eight"='8', "twelve"='12'))) %>%
  mutate( num_of_doors = str_replace_all(num_of_doors, c("two"='2',"four"='4')))

cars.df <- sapply(cars.df, as.numeric) %>% as_tibble()
glimpse(cars.df)
```

```
## Rows: 205
## Columns: 17
## $ `normalized-losses` <dbl> NA, NA, NA, 164, 164, NA, 158, NA, 158, NA, 192, 1~
## $ num_of_doors        <dbl> 2, 2, 2, 4, 4, 2, 4, 4, 4, 2, 2, 4, 2, 4, 4, 4, 2,~
## $ `wheel-base`        <dbl> 88.6, 88.6, 94.5, 99.8, 99.4, 99.8, 105.8, 105.8, ~
## $ length              <dbl> 168.8, 168.8, 171.2, 176.6, 176.6, 177.3, 192.7, 1~
## $ width               <dbl> 64.1, 64.1, 65.5, 66.2, 66.4, 66.3, 71.4, 71.4, 71~
## $ height              <dbl> 48.8, 48.8, 52.4, 54.3, 54.3, 53.1, 55.7, 55.7, 55~
## $ `curb-weight`       <dbl> 2548, 2548, 2823, 2337, 2824, 2507, 2844, 2954, 30~
## $ num_of_cylinders    <dbl> 4, 4, 6, 4, 5, 5, 5, 5, 5, 5, 4, 4, 6, 6, 6, 6, 6,~
## $ `engine-size`       <dbl> 130, 130, 152, 109, 136, 136, 136, 136, 131, 131, ~
## $ bore                <dbl> 3.47, 3.47, 2.68, 3.19, 3.19, 3.19, 3.19, 3.19, 3.~
## $ stroke              <dbl> 2.68, 2.68, 3.47, 3.40, 3.40, 3.40, 3.40, 3.40, 3.~
## $ `compression-ratio` <dbl> 9.00, 9.00, 9.00, 10.00, 8.00, 8.50, 8.50, 8.50, 8~
## $ horsepower          <dbl> 111, 111, 154, 102, 115, 110, 110, 110, 140, 160, ~
## $ `peak-rpm`          <dbl> 5000, 5000, 5000, 5500, 5500, 5500, 5500, 5500, 55~
## $ `city-mpg`          <dbl> 21, 21, 19, 24, 18, 19, 19, 19, 17, 16, 23, 23, 21~
## $ `highway-mpg`       <dbl> 27, 27, 26, 30, 22, 25, 25, 25, 20, 22, 29, 29, 28~
## $ price               <dbl> 13495, 16500, 16500, 13950, 17450, 15250, 17710, 1~
```

2:

Outliers are detected by 3 times larger than z value for price and length variables.

Variables of outliers were determined by three highest correlation yield from AIC backward elimination, they are curb-weight, width, hoursepower, and num_of_cylinders.

```r
step(lm(price ~ . , data = drop_na(cars.df)), direction = 'backward',   trace = 0) %>% summary()
```

```
##
## Call:
## lm(formula = price ~ `normalized-losses` + `wheel-base` + length +
##     width + `curb-weight` + num_of_cylinders + horsepower, data = drop_na(cars.df))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5852.4 -1289.5  -160.1   979.4  7158.1
##
## Coefficients:
```

```
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -67134.496  11288.022  -5.947 1.82e-08 ***
## `normalized-losses`       9.414      5.739   1.640 0.103026
## `wheel-base`            189.567     86.682   2.187 0.030287 *
## length                  -88.677     42.598  -2.082 0.039054 *
## width                   788.133    217.336   3.626 0.000393 ***
## `curb-weight`             6.156      1.185   5.194 6.56e-07 ***
## num_of_cylinders       1231.638    394.460   3.122 0.002151 **
## horsepower               21.090     11.368   1.855 0.065517 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2353 on 151 degrees of freedom
## Multiple R-squared:  0.8469, Adjusted R-squared:  0.8398
## F-statistic: 119.3 on 7 and 151 DF,  p-value: < 2.2e-16
```

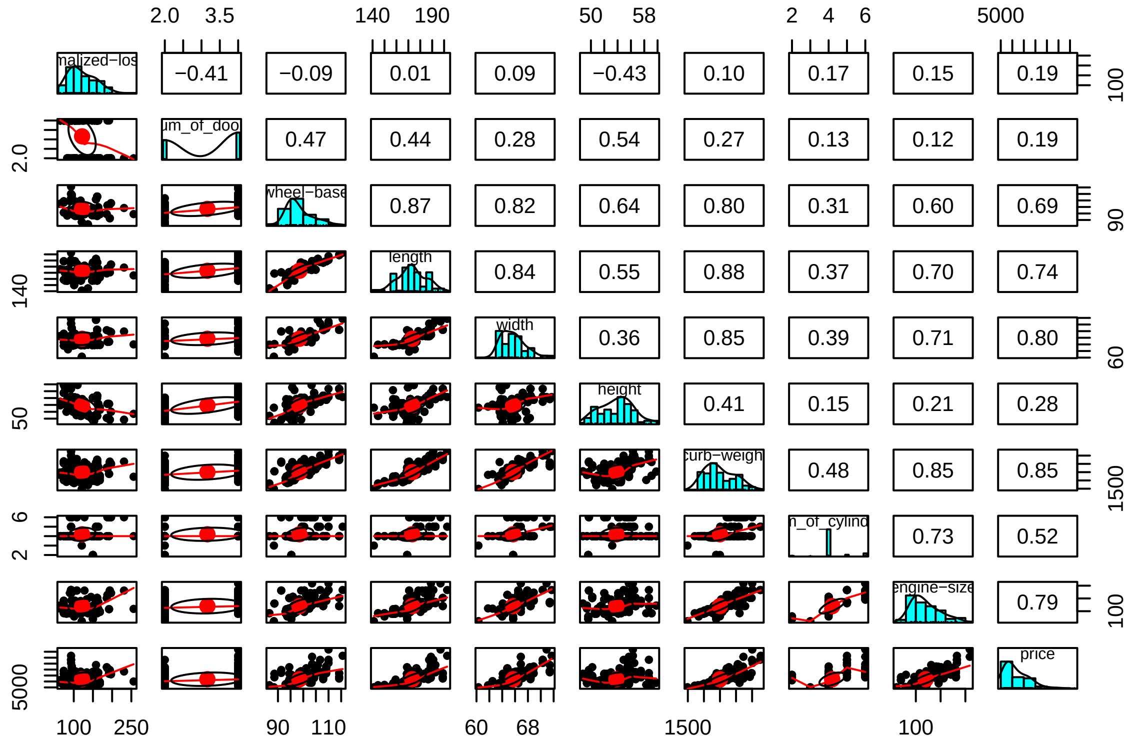
```
cars.no.df <- cars.df %>%
  mutate( z = (price - mean(price, na.rm=T))/sd(price, na.rm = T) ) %>%
  filter( abs(z) <= 3) %>%
  mutate( z = (width - mean(width, na.rm=T))/sd(width, na.rm = T) ) %>%
  filter( abs(z) <= 3) %>%
  mutate( z = (`curb-weight` - mean(`curb-weight`, na.rm=T))/sd(`curb-weight`, na.rm = T) ) %>%
  filter( abs(z) <= 3) %>%
  mutate( z = (num_of_cylinders - mean(num_of_cylinders, na.rm=T))/sd(num_of_cylinders, na.rm = T) ) %>%
  filter( abs(z) <= 3) %>%
  mutate( z = (horsepower - mean(horsepower, na.rm=T))/sd(horsepower, na.rm = T) ) %>%
  filter( abs(z) <= 3) %>%
  dplyr::select( -z)
```

3:

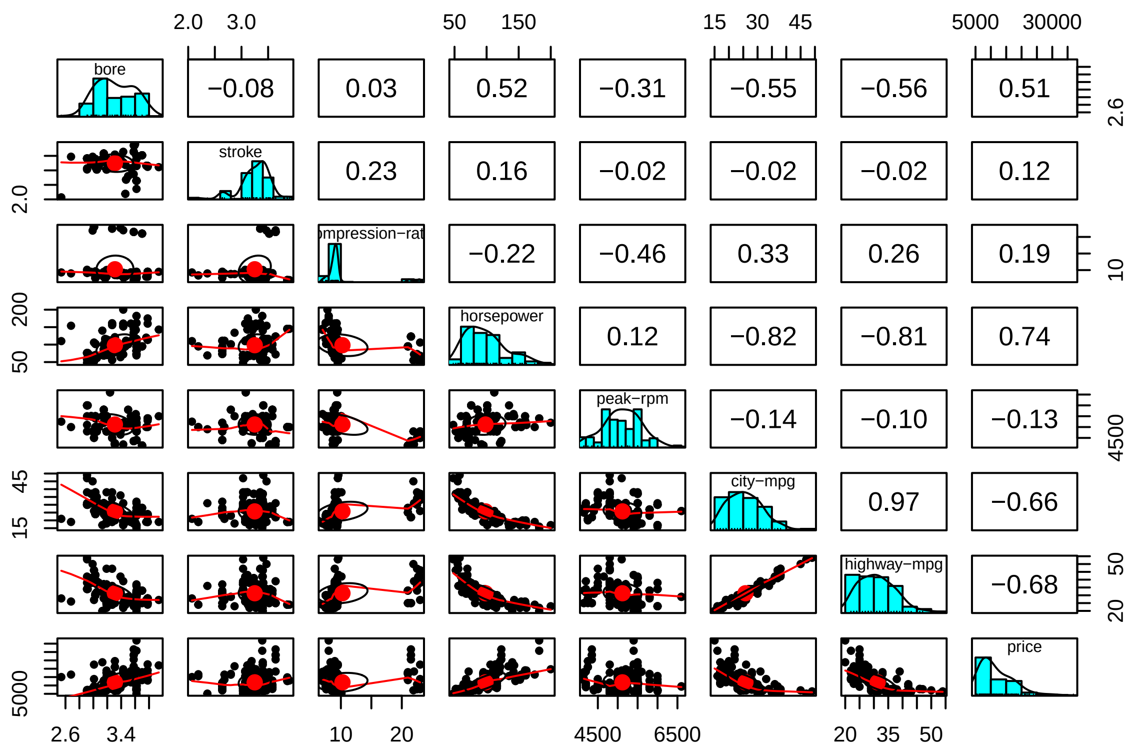
The dataset is too large to make output of pairs.panels visible, so I divided into to figures.

Prices have a strong correlation with wheel-base, length, width,curb-weight,num_of_cylinder, engine_size, highway-mpg, city-mpg,hoursepower, and bore. Some of them are not independent, such as highway-mpg, city-mpg, hoursepower, num_of_cylinder, etc. Also, bore and stroke also related with each other.

```
pairs.panels(cars.no.df[, c(1:9,17)])
```



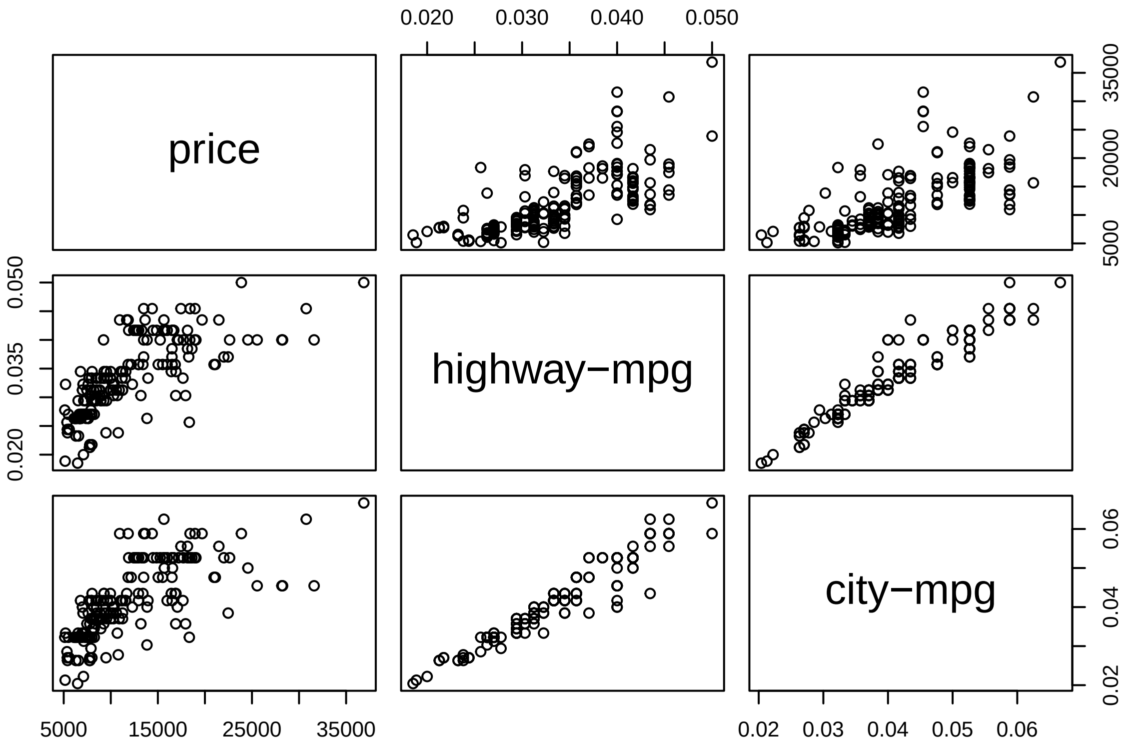

```
pairs.panels(cars.no.df[, c(10:17)])
```



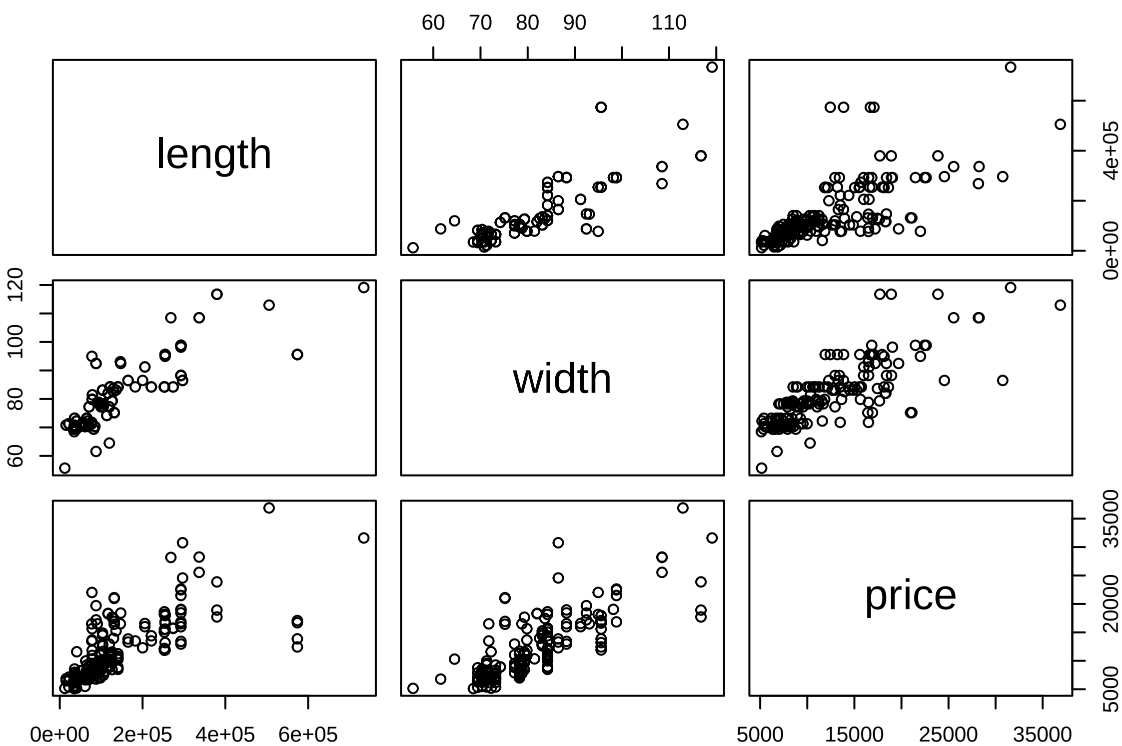
highway-mpg and city-mpg looks like inverse relationship with price. And in the figure below we can see price is linear relationship with reciprocal of highway-mpg and city-mpg.

width and length looks like have an logarithm relationship with price, so make exponential of normalized width and length. results show a better linear relationship.

```
pairs(cbind(cars.no.df[,'price'], 1/cars.no.df[,'highway-mpg'], 1/cars.no.df[,'city-mpg']))
```



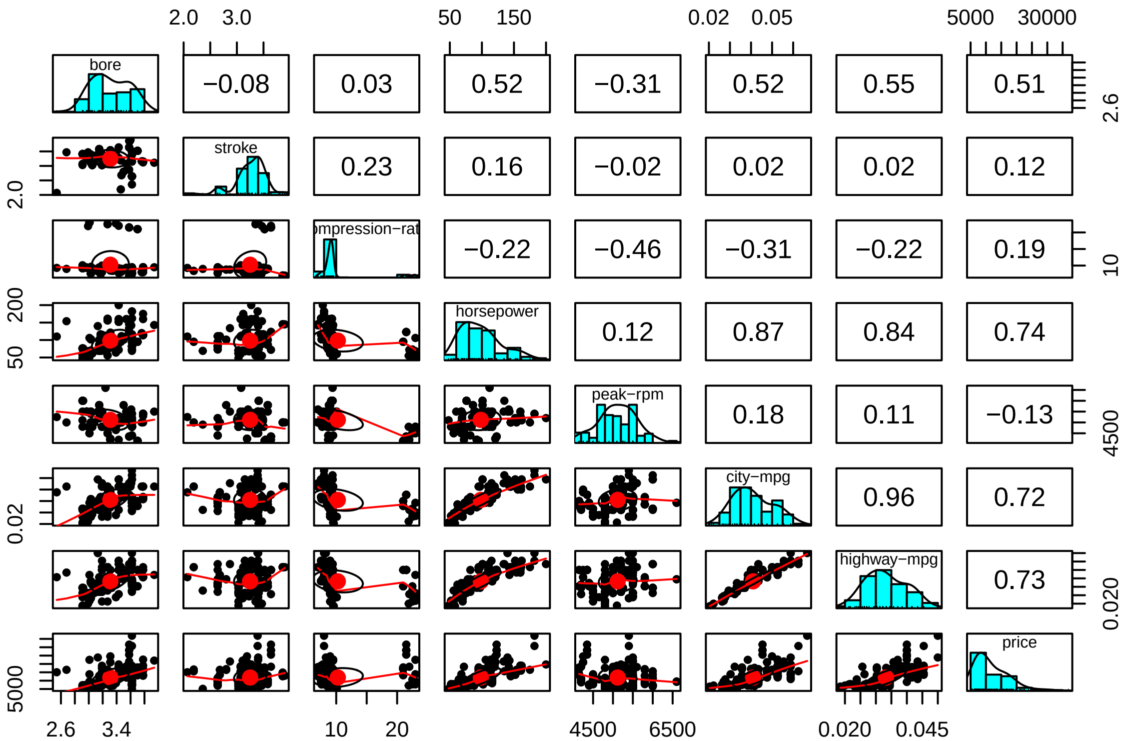

```
pairs(cbind(exp(cars.no.df[,'length']/15), exp(cars.no.df[,'width']/15), (cars.no.df[, 'price'])))
```



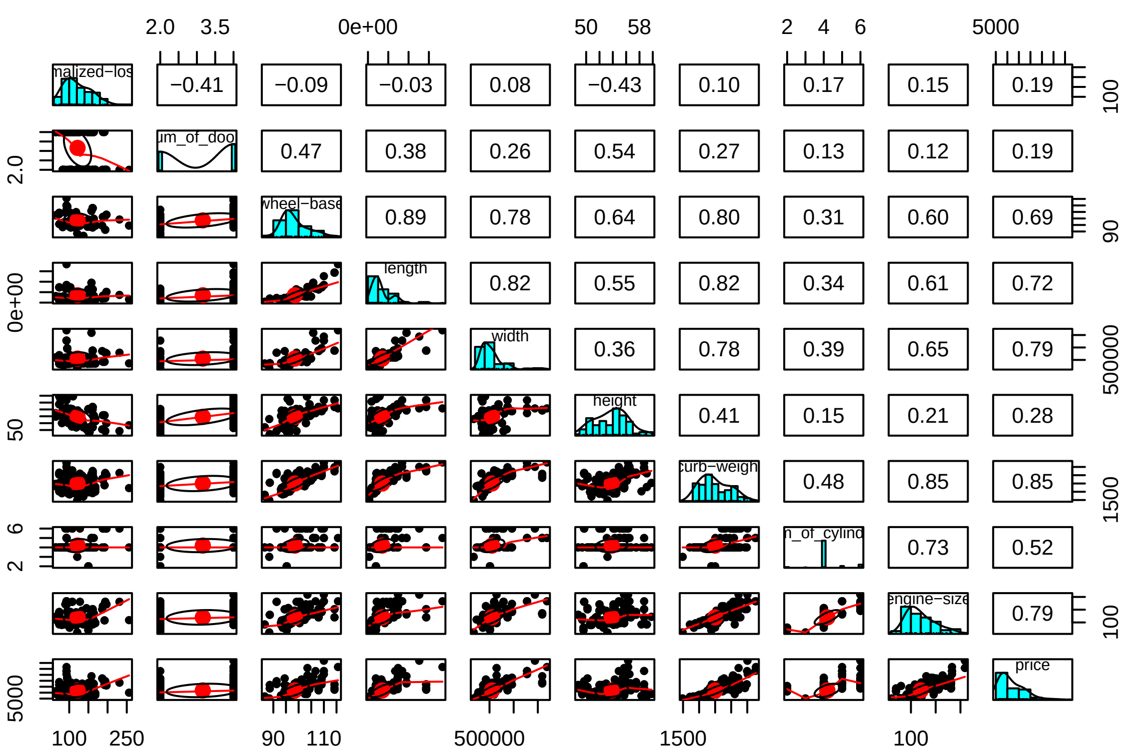

```
cars.tx <- cars.no.df %>%
  mutate( `highway-mpg` = 1/`highway-mpg`) %>%
  mutate( `city-mpg` = 1/ `city-mpg`) %>%
```

```
  mutate( width = exp(width/5)) %>%
  mutate( length = exp(length/15))

pairs.panels(cars.tx[, c(10:17)])
```




```
pairs.panels(cars.tx[, c(1:9,17)])
```



Now the panels are linearized.

4:

There are many collinearities in this dataset. For example, length~curb-weight = 0.87129108, length~width=0.83833846, engine-size~curb-weight=0.88862611, highway-mpg~hoursepower = 0.88862611.

```
cor(x = cars.no.df, use='complete.obs')
```

```
##                   normalized-losses num_of_doors wheel-base      length
## normalized-losses        1.00000000  -0.39618384 -0.0732285  0.02336782
## num_of_doors            -0.39618384   1.00000000  0.4248848  0.41661780
## wheel-base              -0.07322850   0.42488483  1.0000000  0.86960451
## length                   0.02336782   0.41661780  0.8696045  1.00000000
## width                    0.09499348   0.26655780  0.8325194  0.84092197
## height                  -0.41029353   0.46701781  0.5811568  0.52635990
## curb-weight              0.10858655   0.27954134  0.8228514  0.87666119
## num_of_cylinders         0.27223910   0.01871990  0.3263755  0.38157818
## engine-size              0.19818181   0.10355970  0.6691519  0.74163260
## bore                    -0.03943223   0.21553584  0.5755800  0.64136225
## stroke                   0.05338898  -0.05087042  0.1158964  0.07960074
## compression-ratio       -0.12370564   0.11543792  0.3088120  0.19912006
## horsepower               0.28246709   0.03265747  0.5034824  0.65919925
## peak-rpm                 0.24556274  -0.17272948 -0.2852275 -0.22405800
## city-mpg                -0.22630296  -0.17422594 -0.5769268 -0.71729638
## highway-mpg             -0.17723299  -0.17334555 -0.6126441 -0.71868203
## price                    0.19141328   0.19482196  0.7708349  0.77862010
##                         width       height curb-weight num_of_cylinders
## normalized-losses  0.09499348 -0.410293527   0.1085865      0.272239101
## num_of_doors       0.26655780  0.467017811   0.2795413      0.018719900
## wheel-base         0.83251942  0.581156811   0.8228514      0.326375511
## length             0.84092197  0.526359900   0.8766612      0.381578176
## width              1.00000000  0.334695620   0.8638213      0.456113315
## height             0.33469562  1.000000000   0.4270857     -0.001620483
## curb-weight        0.86382133  0.427085712   1.0000000      0.547958302
## num_of_cylinders   0.45611331 -0.001620483   0.5479583      1.000000000
## engine-size        0.77431711  0.180869230   0.8796917      0.729950370
## bore               0.56953839  0.267332524   0.6511216      0.107093090
## stroke             0.17391752 -0.088921062   0.1250422      0.113289268
## compression-ratio  0.28358275  0.229389257   0.2592218      0.101261653
## horsepower         0.66111504  0.061834394   0.7727500      0.602390374
## peak-rpm          -0.21765869 -0.257872799  -0.2457257     -0.088898890
## city-mpg          -0.64964057 -0.227038225  -0.7523050     -0.459739342
## highway-mpg       -0.67343595 -0.260724199  -0.7766266     -0.482367973
## price              0.83918215  0.324254621   0.8842688      0.557589684
##                   engine-size        bore       stroke compression-ratio
## normalized-losses   0.1981818 -0.03943223  0.053388977       -0.12370564
## num_of_doors        0.1035597  0.21553584 -0.050870421        0.11543792
## wheel-base          0.6691519  0.57558000  0.115896449        0.30881205
## length              0.7416326  0.64136225  0.079600738        0.19912006
## width               0.7743171  0.56953839  0.173917520        0.28358275
## height              0.1808692  0.26733252 -0.088921062        0.22938926
## curb-weight         0.8796917  0.65112161  0.125042193        0.25922183
## num_of_cylinders    0.7299504  0.10709309  0.113289268        0.10126165
## engine-size         1.0000000  0.62276221  0.256542772        0.19116954
```

```
## bore                0.6227622  1.00000000 -0.130851123        0.02130648
## stroke              0.2565428 -0.13085112  1.000000000        0.26200022
## compression-ratio   0.1911695  0.02130648  0.262000223        1.00000000
## horsepower          0.8103673  0.55448046  0.108865110       -0.15302464
## peak-rpm           -0.2769479 -0.30613641  0.002311259       -0.42417112
## city-mpg           -0.6985298 -0.58474242  0.013901796        0.27284489
## highway-mpg        -0.7039959 -0.58680658  0.023138015        0.21443082
## price               0.8055341  0.54617803  0.115237815        0.25909305
##                    horsepower     peak-rpm    city-mpg highway-mpg      price
## normalized-losses  0.28246709  0.245562741 -0.22630296 -0.17723299  0.1914133
## num_of_doors       0.03265747 -0.172729479 -0.17422594 -0.17334555  0.1948220
## wheel-base         0.50348239 -0.285227465 -0.57692676 -0.61264407  0.7708349
## length             0.65919925 -0.224057999 -0.71729638 -0.71868203  0.7786201
## width              0.66111504 -0.217658686 -0.64964057 -0.67343595  0.8391821
## height             0.06183439 -0.257872799 -0.22703823 -0.26072420  0.3242546
## curb-weight        0.77275000 -0.245725665 -0.75230504 -0.77662658  0.8842688
## num_of_cylinders   0.60239037 -0.088898890 -0.45973934 -0.48236797  0.5575897
## engine-size        0.81036734 -0.276947914 -0.69852978 -0.70399588  0.8055341
## bore               0.55448046 -0.306136410 -0.58474242 -0.58680658  0.5461780
## stroke             0.10886511  0.002311259  0.01390180  0.02313802  0.1152378
## compression-ratio -0.15302464 -0.424171117  0.27284489  0.21443082  0.2590931
## horsepower         1.00000000  0.100597597 -0.82975793 -0.81793768  0.7462063
## peak-rpm           0.10059760  1.000000000 -0.07285725 -0.05564826 -0.1489875
## city-mpg          -0.82975793 -0.072857250  1.00000000  0.97132988 -0.6841016
## highway-mpg       -0.81793768 -0.055648262  0.97132988  1.00000000 -0.7043106
## price              0.74620631 -0.148987546 -0.68410164 -0.70431063  1.0000000
```

5:

The sample is applied to each dataset separately since they don't have the same number of rows, then dividing them into train and test subsets.

```
set.seed(7)
train <- sample(nrow(cars.df), 0.7*nrow(cars.df))
train.no <- sample(nrow(cars.no.df), 0.7*nrow(cars.no.df))
train.tx <- sample(nrow(cars.tx), 0.7*nrow(cars.tx))

cars.training <- cars.df[train,]
cars.testing <- cars.df[-train,]
cars.no.training <- cars.no.df[train.no,]
cars.no.testing <- cars.no.df[-train.no,]
cars.tx.training <- cars.tx[train.tx,]
cars.tx.testing <- cars.tx[-train.tx,]
```

6:

I made a backward p value elimination function for "lm" or "glm" functions called lm_backelimq(), which prune variables with p values greater than specific value (default is 0.1).

Also, I used step() function for AIC backward elimination. Rows containing NA are removed since step function requries the structure of data set at constant during backward prune process.

```
set.seed(7)

cars.training <- drop_na(cars.training)
cars.no.training <- drop_na(cars.no.training)
```

```r
cars.tx.training <- drop_na(cars.tx.training)

cars.testing <- drop_na(cars.testing)
cars.no.testing <- drop_na(cars.no.testing)
cars.tx.testing <- drop_na(cars.tx.testing)


lm_backelimq <- function(x_train, keep = FALSE, loop = 10, trace = 1, p = 0.1){
  test <- x_train
  x.lm <- lm(formula = price ~ . , data = test)
  for (q in 1:loop) {
    n = 1
    if (trace == 1) {
      print(q)
      print(summary(x.lm)$coefficients)
    }
    lm.summary <- summary(x.lm)$coefficients[, 4]
    p.to.drop <- names(sort(lm.summary, decreasing = TRUE))[n]
    ## Jump to second or third highest p value if highest is in keep input.
    while(any(p.to.drop == keep)){
      n = n + 1
      p.to.drop <- names(sort(lm.summary, decreasing = TRUE))[n]
    }

    if ( lm.summary[p.to.drop] > p ) {
      p.to.drop <- str_remove_all(p.to.drop, '`')
      test <- dplyr::select(test, -p.to.drop)
    }
    x.lm <- lm(formula = price ~ . , data = test)
  }
  x.lm
}

cars.lm.p <- lm_backelimq(cars.df, trace = 0, p = 0.05, loop = 20)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(p.to.drop)` instead of `p.to.drop` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```r
cars.no.lm.p <- lm_backelimq(cars.no.df, trace = 0, p = 0.05, loop = 20)
cars.tx.lm.p <- lm_backelimq(cars.tx, trace = 0 , p = 0.05, loop = 20)

## draft

cars.lm.aic <- step(lm(price ~ . , data = cars.training), direction = 'backward',   trace = 0)
cars.no.lm.aic <- step(lm(price ~ ., data = cars.no.training), direction = 'backward',   trace = 0)
cars.tx.lm.aic <- step(lm(price ~ ., data = cars.tx.training), direction = 'backward',   trace = 0)
```

7:

Build a Regression Tree model using rpart package for predicting price: one with cars.training, one with cars.no.training, and one with cars.tx.training.

```
set.seed(7)
cars.rp <- rpart(price ~ ., data = cars.df)
cars.no.rp <- rpart(price ~ ., data = cars.no.df)
cars.tx.rp <- rpart(price ~ ., data = cars.tx)
```

8:

Pridiction evaluation: Mean absolute scaled error (MASE) decreased when outliers removed, and further decreased after data linear normalization. Adjusted R squared only defined for linear regression models.

```
set.seed(7)
MASE <- function(x, y){
  return (sum(abs(x - y))/(length(y)-1))
}

RMSE <- function(x, y){
  return ( sqrt( mean( (x - y) ^2)))
}

summary_table <- tibble( 'ARS' = c(summary(cars.lm.p)$adj.r.squared, summary(cars.no.lm.p)$adj.r.square
                       summary(cars.tx.lm.p)$adj.r.squared, summary(cars.lm.aic)$adj.r.squared,
                       summary(cars.no.lm.aic)$adj.r.squared, summary(cars.tx.lm.aic)$adj.r.squared,
                       NA,NA,NA),
             'MASE' = c(MASE(predict(cars.lm.p, newdata = cars.testing), cars.testing$price),
                        MASE(predict(cars.no.lm.p, newdata = cars.no.testing), cars.no.testing$price),
                        MASE(predict(cars.tx.lm.p, newdata = cars.tx.testing), cars.tx.testing$price),
                        MASE(predict(cars.lm.aic, newdata = cars.testing), cars.testing$price),
                        MASE(predict(cars.no.lm.aic, newdata = cars.no.testing), cars.no.testing$price)
                        MASE(predict(cars.tx.lm.aic, newdata = cars.tx.testing), cars.tx.testing$price)
                        MASE(predict(cars.rp, newdata = cars.testing), cars.testing$price),
                        MASE(predict(cars.no.rp, newdata = cars.no.testing), cars.no.testing$price),
                        MASE(predict(cars.tx.rp, newdata = cars.tx.testing), cars.tx.testing$price)),
             'RMSE' = c(RMSE(predict(cars.lm.p, newdata = cars.testing), cars.testing$price),
                        RMSE(predict(cars.no.lm.p, newdata = cars.no.testing), cars.no.testing$price),
                        RMSE(predict(cars.tx.lm.p, newdata = cars.tx.testing), cars.tx.testing$price),
                        RMSE(predict(cars.lm.aic, newdata = cars.testing), cars.testing$price),
                        RMSE(predict(cars.no.lm.aic, newdata = cars.no.testing), cars.no.testing$price)
                        RMSE(predict(cars.tx.lm.aic, newdata = cars.tx.testing), cars.tx.testing$price)
                        RMSE(predict(cars.rp, newdata = cars.testing), cars.testing$price),
                        RMSE(predict(cars.no.rp, newdata = cars.no.testing), cars.no.testing$price),
                        RMSE(predict(cars.tx.rp, newdata = cars.tx.testing), cars.tx.testing$price)),

             ) %>% as.data.frame()
rownames(summary_table) <- c('p_elim_df', 'p_elim_no', 'p_elim_tx',
                 'aic_elim_df', 'aic_elim_no', 'aic_elim_tx',
                 'reg_tree_df', 'reg_tree_no', 'reg_tree_tx')
summary_table
```

```
##                    ARS     MASE     RMSE
## p_elim_df   0.7772423 2304.710 2815.262
## p_elim_no   0.8259724 1393.977 2038.659
```

```
## p_elim_tx   0.8022448 1578.988 2058.414
## aic_elim_df 0.8475268 2109.880 3060.339
## aic_elim_no 0.8527024 1861.995 2859.075
## aic_elim_tx 0.8407162 1589.525 2300.719
## reg_tree_df        NA 1921.473 2606.267
## reg_tree_no        NA 1571.636 2104.560
## reg_tree_tx        NA 1774.602 2376.369
```

Overall results are shown above. P value eliminations yield smaller Adj. R squared, means p elimination didn't do a better job than AIC elimination. Among three type of datasets (cars.df, cars.no.df, cars.tx), cars.tx shows smaller errors for linear regression predictions, but dataframe without outliers yield a better performance for Regression Tree.

Regression Trees did a better job than p_elimination and AIC elimination multivariable linear regressions on original and outliers removed datasets. AIC backward elimination yeilds a better results than Regression Tree on cars.tx dataset.

In general, Regression Tree yield a better prediction than multi-linear-regressions on less-processed datasets. AIC backward eliminated linear model did a better job than Regression Tree since four non-linear variables were manipulated to linear relationships with target - price. Also, there's no need to linearize dataset for Regression Tree.

9:

Use all features as inputs for multivariable linear regression.

```
set.seed(7)
cars.lm <- lm( price ~ ., data = cars.df)
cars.no.lm <- lm( price ~ ., data = cars.no.df)
cars.tx.lm <- lm( price ~ ., data = cars.tx)
```

```
summary(cars.lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = cars.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5755.2 -1185.3  -191.5   817.2  7329.7
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -6.625e+04  1.732e+04  -3.826 0.000195 ***
## `normalized-losses`   7.550e+00  7.141e+00   1.057 0.292150
## num_of_doors         -6.877e+01  2.626e+02  -0.262 0.793817
## `wheel-base`          1.951e+02  9.307e+01   2.097 0.037797 *
## length               -8.622e+01  4.899e+01  -1.760 0.080582 .
## width                 7.661e+02  2.349e+02   3.261 0.001389 **
## height                3.002e+01  1.381e+02   0.217 0.828223
## `curb-weight`         5.459e+00  1.672e+00   3.265 0.001372 **
## num_of_cylinders      7.976e+02  7.744e+02   1.030 0.304814
## `engine-size`         2.496e+01  3.096e+01   0.806 0.421604
## bore                 -7.092e+02  1.539e+03  -0.461 0.645519
## stroke               -1.314e+03  9.396e+02  -1.398 0.164297
## `compression-ratio`   9.685e+01  7.849e+01   1.234 0.219284
## horsepower            2.499e+01  1.681e+01   1.486 0.139391
```

```
## `peak-rpm`           7.245e-01  5.702e-01   1.270 0.205989
## `city-mpg`          -1.755e+01  1.598e+02  -0.110 0.912706
## `highway-mpg`        3.116e+01  1.464e+02   0.213 0.831752
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2387 on 142 degrees of freedom
##   (46 observations deleted due to missingness)
## Multiple R-squared:  0.8518, Adjusted R-squared:  0.8351
## F-statistic: 50.99 on 16 and 142 DF,  p-value: < 2.2e-16
```

```
summary(cars.tx.lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = cars.tx)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5117.0 -1079.0  -157.1   872.1  6742.0
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -2.279e+04  1.079e+04  -2.113 0.036404 *
## `normalized-losses`  9.333e+00  6.421e+00   1.453 0.148342
## num_of_doors        -1.184e+02  2.250e+02  -0.526 0.599430
## `wheel-base`         1.221e+02  8.459e+01   1.443 0.151261
## length               3.231e-03  4.818e-03   0.671 0.503562
## width                6.120e-03  1.614e-03   3.792 0.000222 ***
## height               4.747e+01  1.221e+02   0.389 0.698035
## `curb-weight`        2.533e+00  1.557e+00   1.627 0.106014
## num_of_cylinders     6.997e+02  7.369e+02   0.950 0.343957
## `engine-size`       -9.483e+00  3.001e+01  -0.316 0.752446
## bore                 2.988e+02  1.420e+03   0.210 0.833590
## stroke              -8.185e+02  8.502e+02  -0.963 0.337367
## `compression-ratio`  1.996e+02  7.223e+01   2.763 0.006493 **
## horsepower           4.408e+01  1.785e+01   2.469 0.014760 *
## `peak-rpm`           2.011e-01  5.135e-01   0.392 0.695872
## `city-mpg`           9.831e+04  9.250e+04   1.063 0.289732
## `highway-mpg`       -7.773e+04  1.162e+05  -0.669 0.504805
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2104 on 140 degrees of freedom
##   (31 observations deleted due to missingness)
## Multiple R-squared:  0.8609, Adjusted R-squared:  0.845
## F-statistic: 54.14 on 16 and 140 DF,  p-value: < 2.2e-16
```

For linear regression on cars.df, price will increase 3.116 or decrease 1.755 when 'highway-mpg' or 'city-mpg' increasing one unit with other variables keep fixed, respectively. No differences between cars.df and outliers removed. For cars.tx, price will increase 4.685e+04 or 5.582e+04 when 'highway-mpg' or 'city-mpg' increasing one unit with other variables keep fixed, respectively.

10:

```
set.seed(7)
cars.lm <- augment(cars.lm.aic, newdata = cars.testing)
cars.no.lm <- augment(cars.no.lm.aic, newdata = cars.no.testing)
cars.tx.lm <- augment(cars.tx.lm.aic, newdata = cars.tx.testing)
```

```
sprintf("AIC eliminated cars.df has 95%% Interval Prediction $%.2f +/- %.2f",mean(cars.lm$.fitted), 1.96
```

```
## [1] "AIC eliminated cars.df has 95% Interval Prediction $11900.79 +/- 5814.82"
```

```
sprintf("AIC eliminated cars.no.df has 95%% Interval Prediction $%.2f +/- %.2f",mean(cars.no.lm$.fitted)
```

```
## [1] "AIC eliminated cars.no.df has 95% Interval Prediction $11066.63 +/- 5658.61"
```

```
sprintf("AIC eliminated cars.tx has 95%% Interval Prediction $%.2f +/- %.2f",mean(cars.tx.lm$.fitted),
```

```
## [1] "AIC eliminated cars.tx has 95% Interval Prediction $11113.80 +/- 4482.92"
```