# HarvardX DataScience CapStone-Project: Car Price

Rainer Baumgartner

2022-09-25

## Contents


**1 Introduction** **3**

**2 Loading R packages** **3**

**3 Introduction to the Automobile Data Set** **4**

**4 Data import** **5**

**5 A first view on the data** **6**

**6 Data pre-processing #1** **8**

**7 Data exploration and visualization** **13**
7.1 Price . . . . . 13
7.2 Attributes . . . . . 14

**8 Data pre-processing #2** **18**
8.1 About Reproducibility . . . . . 18
8.2 Impute data . . . . . 18
8.3 Create dummy variables . . . . . 20
8.4 Split data in to test and training . . . . . 23
8.5 Define RMSE . . . . . 23

**9 Methods and techniques** **23**
9.1 Pre-processing . . . . . 23
9.2 Parameter optimization . . . . . 24
9.3 Modelling steps . . . . . 24

**10 Linear Regression** **25**
10.1 Linear Regression simple (lm.1) . . . . . 25
10.2 Additional pre-processing . . . . . 29
10.3 Linear Regression with pre-processing #1 (lm.2) . . . . . 31
10.4 Linear Regression with pre-processing #2 (lm.3) . . . . . 33

**11 knn intro** **36**
11.1 knn simple (knn.1) . . . . . 36
11.2 knn with tuneGrid & trainControl (knn.2) . . . . . 39
11.3 knn with knn.2 + preProcess (knn.3) . . . . . 43

**12 Random Forest** **47**
12.1 rf.1 simple random forest . . . . . 47
12.2 rf.2 with tuneGrid . . . . . 50

12.3 rf.3 with tuneGrid & preProcess . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55

**13 Results** **59**

**14 Summary** **60**

# 1 Introduction

*'PH125.9x:Data Science: Capstone'* is the final course in the *'HarvardX Data Science Professional Certificate'* program.

One of the graded components of this course is *'Choose Your Own!'* , where the student will choose a project on its own from public available datasets and solve a problem of own choice.

The HarvardX course teaches R, thus the tool for the project is of course *R*.

The data '**Automobile Data Set'** for this project is provided by *UCI Machine Learning Repository*.

The self defined target is to predict a car price based on the characteristics of the car. Those are represented by the attributes in the dataset.

As requested for the project, machine learning techniques beyond standard linear regression will be applied. Linear regression will take the role as reference technique.

This machine learning techniques applies will be:

- k-nearest neighbors
- Random Forest

Apart from the techniques, the project will focus also on the *caret-package*, providing tools for data splitting, pre-processing, model tuning and other features.

# 2 Loading R packages

R has many built-in base functions. R packages provide additional functions for certain purposes, like improved graphics or algorithms.

As a first step we load (and install if needed) the R packages used in this project:

---

```
#define required packages
  packs <- c("tidyverse","caret","data.table", "lubridate" , "ggthemes", "knitr",
             "gridExtra","scales", "pryr","DataExplorer","skimr" ,
             "RANN", "ranger")

#install packages (if needed)  and load them
 for (package in packs) {
 if (!require(package, character.only=T, quietly=T)) {
  install.packages(package, repos = "http://cran.us.r-project.org")
  library(package, character.only=T)
  }
 }
```

---

For some part of the data exploration skimr package and DataExplorer are used. Skimr has nice histogram within text summaries, but this can create ugly results on windows machines in combination with locale settings. As a simple and robust approach, the nice histograms will not be shown. The following code maskes the orignial skim function.

---

```
# mask skim function
  skim <- function (x) {skim_without_charts(x)}  # display tables without histograms
```

---

There would be other remedy, but with possible side effects on locale settings: *fix_windows_histograms*

# 3 Introduction to the Automobile Data Set

The following information is cited from the *UCI data source*.

This data set consists of three types of entities:

(a) the specification of an auto in terms of various characteristics,

(b) its assigned insurance risk rating,

(c) its normalized losses in use as compared to other cars.

The second rating corresponds to the degree to which the auto is more risky than its price indicates.

Cars are initially assigned a risk factor symbol associated with its price. Then, if it is more risky (or less), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process "symboling". A value of +3 indicates that the auto is risky, -3 that it is probably pretty safe.

The third factor is the relative average loss payment per insured vehicle year. This value is normalized for all autos within a particular size classification (two-door small, station wagons, sports/speciality, etc…), and represents the average loss per car per year.

Attribute Information: (Attribute: Attribute Range)

1. symboling: -3, -2, -1, 0, 1, 2, 3.
2. normalized-losses: continuous from 65 to 256.
3. make:

alfa-romero, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz, mercury,

mitsubishi, nissan, peugot, plymouth, porsche, renault, saab, subaru, toyota, volkswagen, volvo

4. fuel-type: diesel, gas.
5. aspiration: std, turbo.
6. num-of-doors: four, two.
7. body-style: hardtop, wagon, sedan, hatchback, convertible.
8. drive-wheels: 4wd, fwd, rwd.
9. engine-location: front, rear.
10. wheel-base: continuous from 86.6 120.9.
11. length: continuous from 141.1 to 208.1.
12. width: continuous from 60.3 to 72.3.
13. height: continuous from 47.8 to 59.8.
14. curb-weight: continuous from 1488 to 4066.
15. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
16. num-of-cylinders: eight, five, four, six, three, twelve, two.
17. engine-size: continuous from 61 to 326.
18. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
19. bore: continuous from 2.54 to 3.94.

20. stroke: continuous from 2.07 to 4.17.
21. compression-ratio: continuous from 7 to 23.
22. horsepower: continuous from 48 to 288.
23. peak-rpm: continuous from 4150 to 6600.
24. city-mpg: continuous from 13 to 49.
25. highway-mpg: continuous from 16 to 54.
26. price: continuous from 5118 to 45400.

# 4 Data import

After the download, we change column names to be at bit more consistent and shorter.

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Import data ------------------------------------------------------------
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

automobile.dataset <-
  read.csv(
    'https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data',
    header = TRUE)

#  change column names
column_names <- c("symboling","losses","make","fuel","aspiration","doors","body_style",
                  "wheel_drive","engine_loc","wheel_base","length","width","height",
                  "curb_weight","engine_type","cylinders","engine_size","fuel_sys",
                  "bore","stroke","compr_ratio","hp","peak_rpm",
                  "city_mpg","hwy_mpg","price")

colnames(automobile.dataset) <-  column_names
```

# 5 A first view on the data

A glimpse of the data:

```r
# glimpse of the data:
head(automobile.dataset, 6)
```

```
##   symboling losses        make fuel aspiration doors  body_style wheel_drive
## 1         3      ? alfa-romero  gas        std   two convertible         rwd
## 2         1      ? alfa-romero  gas        std   two   hatchback         rwd
## 3         2    164        audi  gas        std  four       sedan         fwd
## 4         2    164        audi  gas        std  four       sedan         4wd
## 5         2      ?        audi  gas        std   two       sedan         fwd
## 6         1    158        audi  gas        std  four       sedan         fwd
##   engine_loc wheel_base length width height curb_weight engine_type cylinders
## 1      front       88.6  168.8  64.1   48.8        2548        dohc      four
## 2      front       94.5  171.2  65.5   52.4        2823        ohcv       six
## 3      front       99.8  176.6  66.2   54.3        2337         ohc      four
## 4      front       99.4  176.6  66.4   54.3        2824         ohc      five
## 5      front       99.8  177.3  66.3   53.1        2507         ohc      five
## 6      front      105.8  192.7  71.4   55.7        2844         ohc      five
##   engine_size fuel_sys bore stroke compr_ratio  hp peak_rpm city_mpg hwy_mpg
## 1         130     mpfi 3.47   2.68         9.0 111     5000       21      27
## 2         152     mpfi 2.68   3.47         9.0 154     5000       19      26
## 3         109     mpfi 3.19   3.40        10.0 102     5500       24      30
## 4         136     mpfi 3.19   3.40         8.0 115     5500       18      22
## 5         136     mpfi 3.19   3.40         8.5 110     5500       19      25
## 6         136     mpfi 3.19   3.40         8.5 110     5500       19      25
##   price
## 1 16500
## 2 16500
## 3 13950
## 4 17450
## 5 15250
## 6 17710
```

?'s in the data represent missing data. They are placeholder for NA, where the data-field is really empty. Are there real empty data-fields (NA)?

```r
# assessing NAs
sum(is.na(automobile.dataset))
```

```
## [1] 0
```

The result 0 indicated, no real NA's. Thus we need to manage the '?'-placeholders in our data pre-processing.

skimr provides a comprehensive summary of our data:

```
# summary of our data
skim(automobile.dataset)
```

Table 1: Data summary

| | |
|---|---|
| Name | x |
| Number of rows | 204 |
| Number of columns | 26 |
| _______________________ | |
| Column type frequency: | |
| character | 16 |
| numeric | 10 |
| ________________________ | |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| losses | 0 | 1 | 1 | 3 | 0 | 52 | 0 |
| make | 0 | 1 | 3 | 13 | 0 | 22 | 0 |
| fuel | 0 | 1 | 3 | 6 | 0 | 2 | 0 |
| aspiration | 0 | 1 | 3 | 5 | 0 | 2 | 0 |
| doors | 0 | 1 | 1 | 4 | 0 | 3 | 0 |
| body_style | 0 | 1 | 5 | 11 | 0 | 5 | 0 |
| wheel_drive | 0 | 1 | 3 | 3 | 0 | 3 | 0 |
| engine_loc | 0 | 1 | 4 | 5 | 0 | 2 | 0 |
| engine_type | 0 | 1 | 1 | 5 | 0 | 7 | 0 |
| cylinders | 0 | 1 | 3 | 6 | 0 | 7 | 0 |
| fuel_sys | 0 | 1 | 3 | 4 | 0 | 8 | 0 |
| bore | 0 | 1 | 1 | 4 | 0 | 39 | 0 |
| stroke | 0 | 1 | 1 | 4 | 0 | 37 | 0 |
| hp | 0 | 1 | 1 | 3 | 0 | 60 | 0 |
| peak_rpm | 0 | 1 | 1 | 4 | 0 | 24 | 0 |
| price | 0 | 1 | 1 | 5 | 0 | 186 | 0 |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| symboling | 0 | 1 | 0.82 | 1.24 | -2.0 | 0.00 | 1.0 | 2.00 | 3.0 |
| wheel_base | 0 | 1 | 98.81 | 5.99 | 86.6 | 94.50 | 97.0 | 102.40 | 120.9 |
| length | 0 | 1 | 174.07 | 12.36 | 141.1 | 166.30 | 173.2 | 183.20 | 208.1 |
| width | 0 | 1 | 65.92 | 2.15 | 60.3 | 64.07 | 65.5 | 66.90 | 72.3 |
| height | 0 | 1 | 53.75 | 2.42 | 47.8 | 52.00 | 54.1 | 55.50 | 59.8 |
| curb_weight | 0 | 1 | 2555.60 | 521.96 | 1488.0 | 2145.00 | 2414.0 | 2939.25 | 4066.0 |
| engine_size | 0 | 1 | 126.89 | 41.74 | 61.0 | 97.00 | 119.5 | 142.00 | 326.0 |
| compr_ratio | 0 | 1 | 10.15 | 3.98 | 7.0 | 8.57 | 9.0 | 9.40 | 23.0 |
| city_mpg | 0 | 1 | 25.24 | 6.55 | 13.0 | 19.00 | 24.0 | 30.00 | 49.0 |
| hwy_mpg | 0 | 1 | 30.77 | 6.90 | 16.0 | 25.00 | 30.0 | 34.50 | 54.0 |

# 6 Data pre-processing #1

Some data modification is needed. To preserve the data-input, we 'copy' the automobile.dataset into a new dataframe for our modifications.

```
# new dataset
am <- automobile.dataset
```

Now we take care of the ?'s and replace them by nothing (i.e. NA), which can be handled with caret. …and check the data again with skimr.

```
# Replace ? with NA
am[am == '?'] <- NA

# view data again
skim(am)
```

Table 4: Data summary

| | |
|---|---|
| Name | x |
| Number of rows | 204 |
| Number of columns | 26 |
| Column type frequency: | |
| character | 16 |
| numeric | 10 |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| losses | 40 | 0.80 | 2 | 3 | 0 | 51 | 0 |
| make | 0 | 1.00 | 3 | 13 | 0 | 22 | 0 |
| fuel | 0 | 1.00 | 3 | 6 | 0 | 2 | 0 |
| aspiration | 0 | 1.00 | 3 | 5 | 0 | 2 | 0 |
| doors | 2 | 0.99 | 3 | 4 | 0 | 2 | 0 |
| body_style | 0 | 1.00 | 5 | 11 | 0 | 5 | 0 |
| wheel_drive | 0 | 1.00 | 3 | 3 | 0 | 3 | 0 |
| engine_loc | 0 | 1.00 | 4 | 5 | 0 | 2 | 0 |
| engine_type | 0 | 1.00 | 1 | 5 | 0 | 7 | 0 |
| cylinders | 0 | 1.00 | 3 | 6 | 0 | 7 | 0 |
| fuel_sys | 0 | 1.00 | 3 | 4 | 0 | 8 | 0 |
| bore | 4 | 0.98 | 4 | 4 | 0 | 38 | 0 |
| stroke | 4 | 0.98 | 4 | 4 | 0 | 36 | 0 |
| hp | 2 | 0.99 | 2 | 3 | 0 | 59 | 0 |
| peak_rpm | 2 | 0.99 | 4 | 4 | 0 | 23 | 0 |
| price | 4 | 0.98 | 4 | 5 | 0 | 185 | 0 |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| symboling | 0 | 1 | 0.82 | 1.24 | -2.0 | 0.00 | 1.0 | 2.00 | 3.0 |
| wheel_base | 0 | 1 | 98.81 | 5.99 | 86.6 | 94.50 | 97.0 | 102.40 | 120.9 |
| length | 0 | 1 | 174.07 | 12.36 | 141.1 | 166.30 | 173.2 | 183.20 | 208.1 |
| width | 0 | 1 | 65.92 | 2.15 | 60.3 | 64.07 | 65.5 | 66.90 | 72.3 |
| height | 0 | 1 | 53.75 | 2.42 | 47.8 | 52.00 | 54.1 | 55.50 | 59.8 |
| curb_weight | 0 | 1 | 2555.60 | 521.96 | 1488.0 | 2145.00 | 2414.0 | 2939.25 | 4066.0 |
| engine_size | 0 | 1 | 126.89 | 41.74 | 61.0 | 97.00 | 119.5 | 142.00 | 326.0 |
| compr_ratio | 0 | 1 | 10.15 | 3.98 | 7.0 | 8.57 | 9.0 | 9.40 | 23.0 |
| city_mpg | 0 | 1 | 25.24 | 6.55 | 13.0 | 19.00 | 24.0 | 30.00 | 49.0 |
| hwy_mpg | 0 | 1 | 30.77 | 6.90 | 16.0 | 25.00 | 30.0 | 34.50 | 54.0 |

Now we see missing values instead of ?'s in losses, doors, bore, stroke, hp, peak_rpm, price. Losses has even 18% of missing data!

How to handle the missing prices? The target is to predict the price, so the rows with no price info are not useful to train a model. We remove rows with missing price right now:

```
# remove rows with missing price
am <- am %>% filter (!is.na(price))
```

The other missing values will be handled later.

Back to the the other variables and their types:

Basically, numeric values are preferred as the prediction methods usually calculate. Factors at least 'mask' each value with a number. Some columns can be converted from character to numeric, the others will be converted to factors - at least for a while. Numeric columns will remain as they are.

Referring to the 'Attribute Information' (see chapter 'Introduction to the Automobile Data Set'), the columns to be factors are specified and converted:

```
# convert char to factor
to.factor <-
  c('make','fuel','aspiration','body_style','wheel_drive',
    'engine_loc', 'engine_type','fuel_sys')

am <- am %>% mutate(across(all_of(to.factor), as.factor) )
```

What about the remaining character attributes?

```
# review data again
skim(am) %>% filter(skim_type == 'character')
```

Table 7: Data summary

| | |
|---|---|
| Name | x |
| Number of rows | 200 |
| Number of columns | 26 |
| Column type frequency: | |
| character | 8 |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| losses | 36 | 0.82 | 2 | 3 | 0 | 51 | 0 |
| doors | 2 | 0.99 | 3 | 4 | 0 | 2 | 0 |
| cylinders | 0 | 1.00 | 3 | 6 | 0 | 7 | 0 |
| bore | 4 | 0.98 | 4 | 4 | 0 | 38 | 0 |
| stroke | 4 | 0.98 | 4 | 4 | 0 | 36 | 0 |
| hp | 2 | 0.99 | 2 | 3 | 0 | 58 | 0 |
| peak_rpm | 2 | 0.99 | 4 | 4 | 0 | 22 | 0 |
| price | 0 | 1.00 | 4 | 5 | 0 | 185 | 0 |

We perform data transformation from character to numeric. We need to check each variable:

```
# Check chr values for numerical transformation
am.char<- skim(am) %>% filter(skim_type == 'character') %>% select(skim_variable) %>% pull()

am %>% select(all_of(am.char)) %>% head(6)
##   losses doors cylinders bore stroke  hp peak_rpm price
## 1   <NA>   two      four 3.47   2.68 111     5000 16500
## 2   <NA>   two       six 2.68   3.47 154     5000 16500
## 3    164  four      four 3.19   3.40 102     5500 13950
## 4    164  four      five 3.19   3.40 115     5500 17450
## 5   <NA>   two      five 3.19   3.40 110     5500 15250
## 6    158  four      five 3.19   3.40 110     5500 17710
```

Doors and cylinders can be converted by mapping to numeric, the rest can be converted via 'as.numeric' function.

Doors:

```
# mapping doors to numeric
unique(am$doors)
```

```
## [1] "two"  "four" NA
```

Only 2 values need to be converted:

```
am$doors <- ifelse(am$doors =="two", 2, 4)

am$doors
##   [1]  2  2  4  4  2  4  4  4  2  4  2  4  4  4  2  4  2  2  4  2  2  2  4  4  4
##  [26] NA  4  2  2  2  2  2  2  4  4  2  2  4  4  4  2  4  2  4  4  2  2  2  2  4
##  [51]  4  2  2  2  2  2  4  2  4 NA  4  4  4  4  4  2  4  4  2  4  2  2  2  2  2
##  [76]  2  2  2  2  2  2  4  4  4  4  2  2  2  4  4  2  2  4  4  2  4  4  4  4  4
## [101]  2  2  2  4  4  4  4  4  4  4  4  4  4  4  2  2  4  4  4  4  2  2  2  2  2
## [126]  4  2  2  4  2  4  2  4  2  2  2  4  4  4  4  4  4  4  4  4  2  2  4  4  4
## [151]  4  4  4  4  4  4  4  4  2  2  2  2  2  2  2  2  2  2  4  4  4  4  4  2  2
## [176]  4  4  2  2  4  4  4  4  4  2  2  4  4  4  4  4  4  4  4  4  4  4  4  4  4
```

Cylinders:

```
# mapping of cylinders to numeric
unique(am$cylinders)
```

```
## [1] "four"   "six"    "five"   "three"  "twelve" "two"    "eight"
```

Mapping text to number:

```
am <- am %>% mutate (cylinders = case_when(
        cylinders=='four' ~ 4,
        cylinders=='six' ~ 6,
        cylinders=='five' ~ 5,
        cylinders=='three' ~ 3,
        cylinders=='twelve' ~ 12,
        cylinders=='two' ~ 2,
        cylinders=='eight' ~ 8,
        TRUE ~ as.numeric(cylinders)
      ))
```

```
## Warning in eval_tidy(pair$rhs, env = default_env): NAs durch Umwandlung erzeugt
```

Convert the remaining characters to numeric:

```
# Convert the remaining characters to numeric:
am.char <- skim(am) %>% filter(skim_type == 'character') %>%
  select(skim_variable) %>% pull()

for ( x in am.char) {
      am[,paste(x)] <- as.numeric(am[,paste(x)])
    }
```

'How are' our data now?

```
# review data
skim(am)
```

Table 9: Data summary

| | |
|---|---|
| Name | x |
| Number of rows | 200 |
| Number of columns | 26 |
| Column type frequency: | |
| factor | 8 |
| numeric | 18 |
| Group variables | None |

**Variable type: factor**

| skim_variable | n_missing | complete_rate | ordered | n_unique | top_counts |
|---|---|---|---|---|---|
| make | 0 | 1 | FALSE | 22 | toy: 32, nis: 18, maz: 17, hon: 13 |
| fuel | 0 | 1 | FALSE | 2 | gas: 180, die: 20 |
| aspiration | 0 | 1 | FALSE | 2 | std: 164, tur: 36 |
| body_style | 0 | 1 | FALSE | 5 | sed: 94, hat: 68, wag: 25, har: 8 |
| wheel_drive | 0 | 1 | FALSE | 3 | fwd: 118, rwd: 74, 4wd: 8 |
| engine_loc | 0 | 1 | FALSE | 2 | fro: 197, rea: 3 |
| engine_type | 0 | 1 | FALSE | 6 | ohc: 145, ohc: 15, ohc: 13, l: 12 |
| fuel_sys | 0 | 1 | FALSE | 8 | mpf: 91, 2bb: 64, idi: 20, 1bb: 11 |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| symboling | 0 | 1.00 | 0.83 | 1.25 | -2.00 | 0.00 | 1.00 | 2.00 | 3.00 |
| losses | 36 | 0.82 | 122.00 | 35.44 | 65.00 | 94.00 | 115.00 | 150.00 | 256.00 |
| doors | 2 | 0.99 | 3.14 | 0.99 | 2.00 | 2.00 | 4.00 | 4.00 | 4.00 |
| wheel_base | 0 | 1.00 | 98.85 | 6.04 | 86.60 | 94.50 | 97.00 | 102.40 | 120.90 |
| length | 0 | 1.00 | 174.23 | 12.35 | 141.10 | 166.68 | 173.20 | 183.50 | 208.10 |
| width | 0 | 1.00 | 65.90 | 2.10 | 60.30 | 64.18 | 65.50 | 66.67 | 72.00 |
| height | 0 | 1.00 | 53.79 | 2.43 | 47.80 | 52.00 | 54.10 | 55.52 | 59.80 |
| curb_weight | 0 | 1.00 | 2555.70 | 518.59 | 1488.00 | 2163.00 | 2414.00 | 2928.25 | 4066.00 |
| cylinders | 0 | 1.00 | 4.36 | 1.06 | 2.00 | 4.00 | 4.00 | 4.00 | 12.00 |
| engine_size | 0 | 1.00 | 126.86 | 41.65 | 61.00 | 97.75 | 119.50 | 142.00 | 326.00 |
| bore | 4 | 0.98 | 3.33 | 0.27 | 2.54 | 3.15 | 3.31 | 3.59 | 3.94 |
| stroke | 4 | 0.98 | 3.26 | 0.32 | 2.07 | 3.11 | 3.29 | 3.41 | 4.17 |
| compr_ratio | 0 | 1.00 | 10.17 | 4.01 | 7.00 | 8.57 | 9.00 | 9.40 | 23.00 |
| hp | 2 | 0.99 | 103.36 | 37.65 | 48.00 | 70.00 | 95.00 | 116.00 | 262.00 |
| peak_rpm | 2 | 0.99 | 5118.18 | 481.67 | 4150.00 | 4800.00 | 5200.00 | 5500.00 | 6600.00 |
| city_mpg | 0 | 1.00 | 25.20 | 6.43 | 13.00 | 19.00 | 24.00 | 30.00 | 49.00 |
| hwy_mpg | 0 | 1.00 | 30.70 | 6.83 | 16.00 | 25.00 | 30.00 | 34.00 | 54.00 |
| price | 0 | 1.00 | 13205.69 | 7966.98 | 5118.00 | 7775.00 | 10270.00 | 16500.75 | 45400.00 |

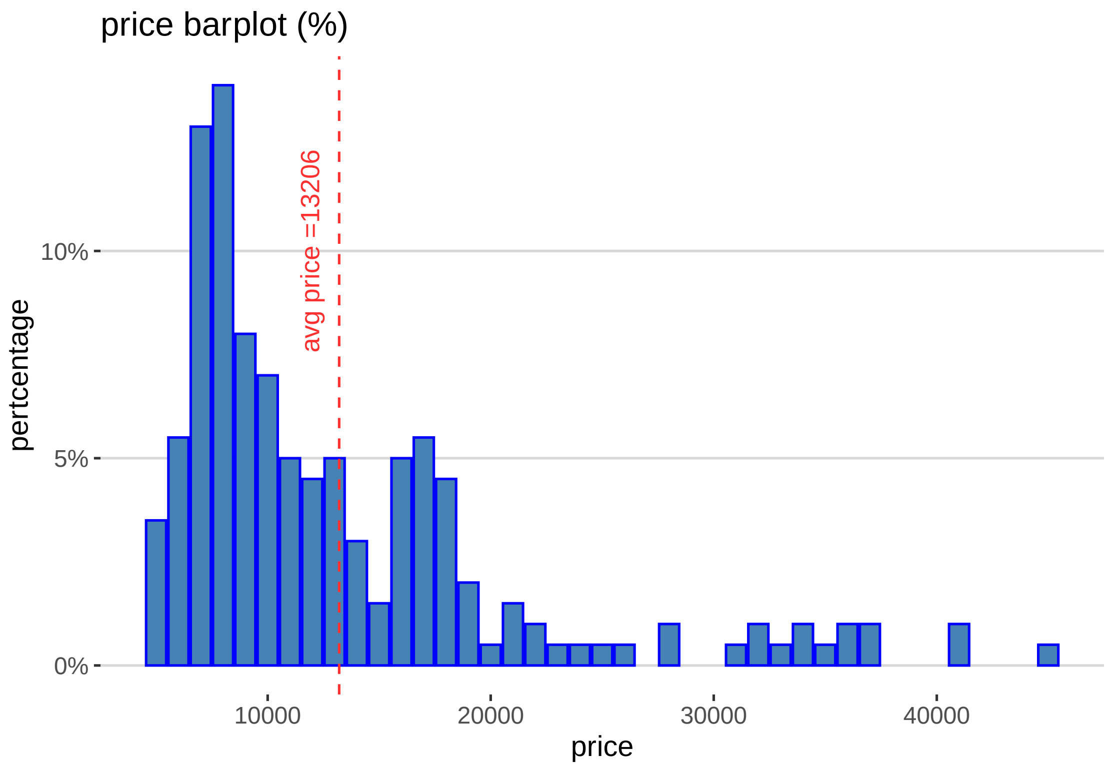
Now it is time to get more insight of our data.

# 7 Data exploration and visualization

## 7.1 Price

First we focus on 'price' - the variable we want to predict.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5118    7775   10270   13206   16501   45400
```




The data are right skewed, so most cars are low priced. The higher the prices, the more scattered the data.

## 7.2 Attributes

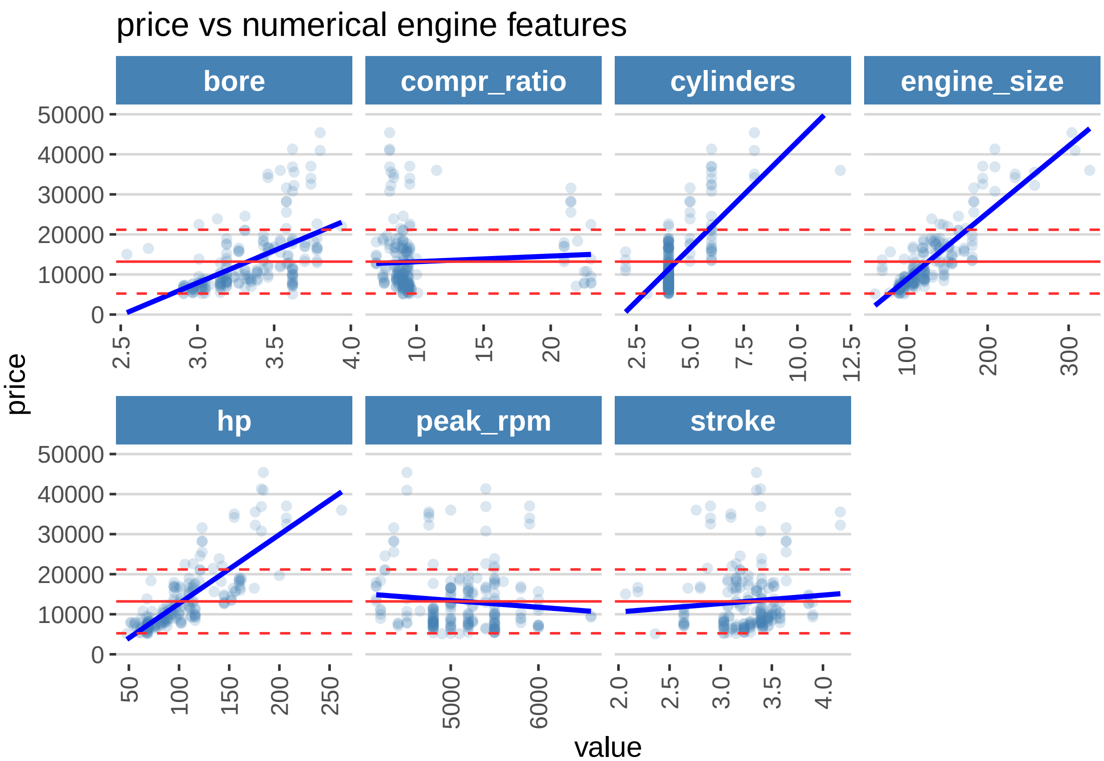
How do the attributes influencing the price? We plot each attribute against the price.

The plots are grouped by engine-, chassis- and economical attributes. Make gets an own plot.

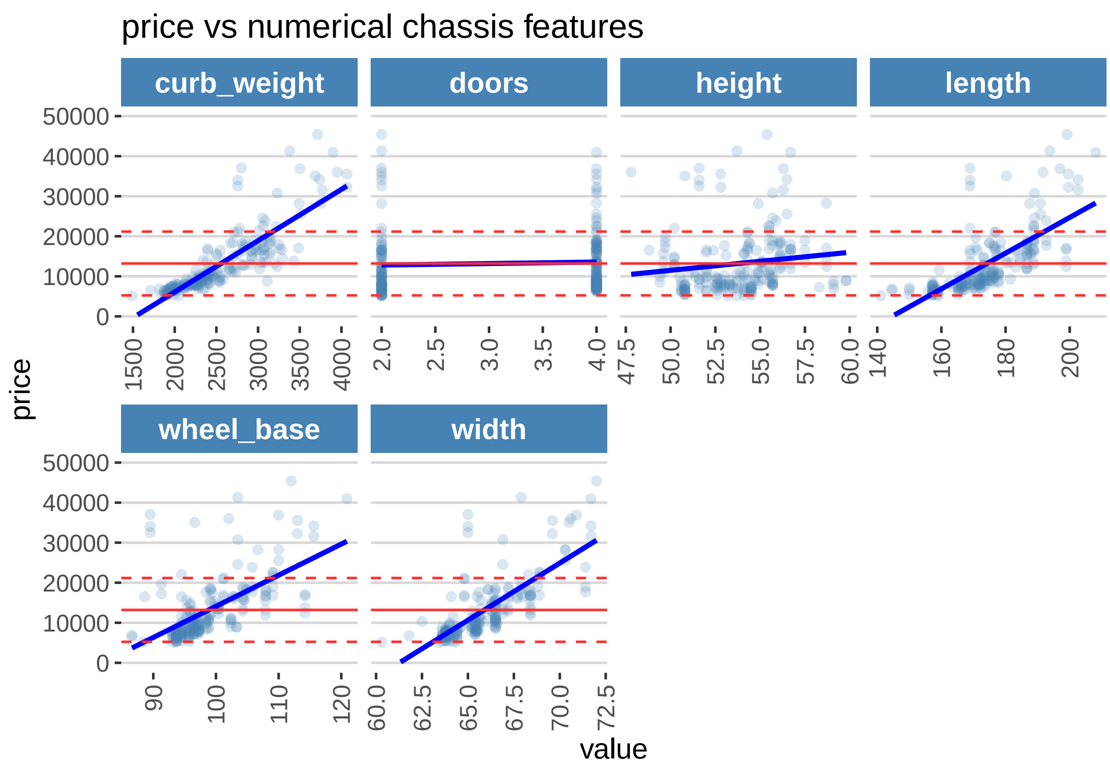
For better reference the mean and standard deviation are indicated with red lines.

Numerical data are plotted as scatterplot with a linear regression line.

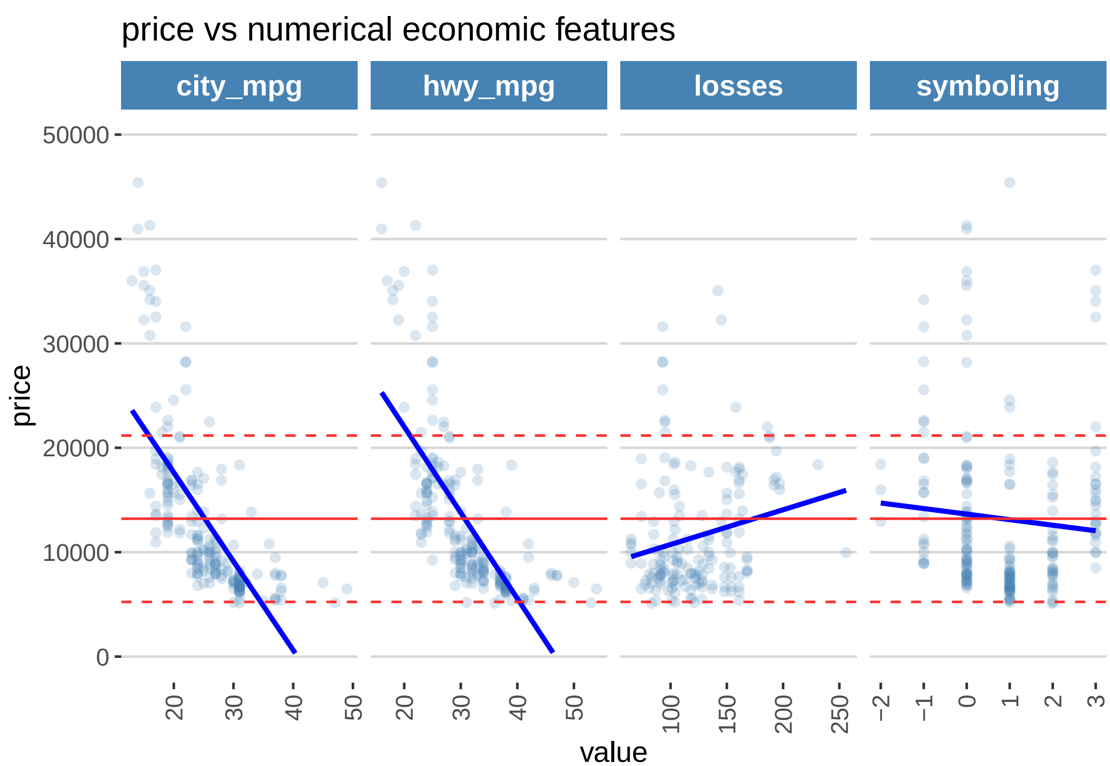
Categorical data are plotted as boxplot, attributes order descendant by median.



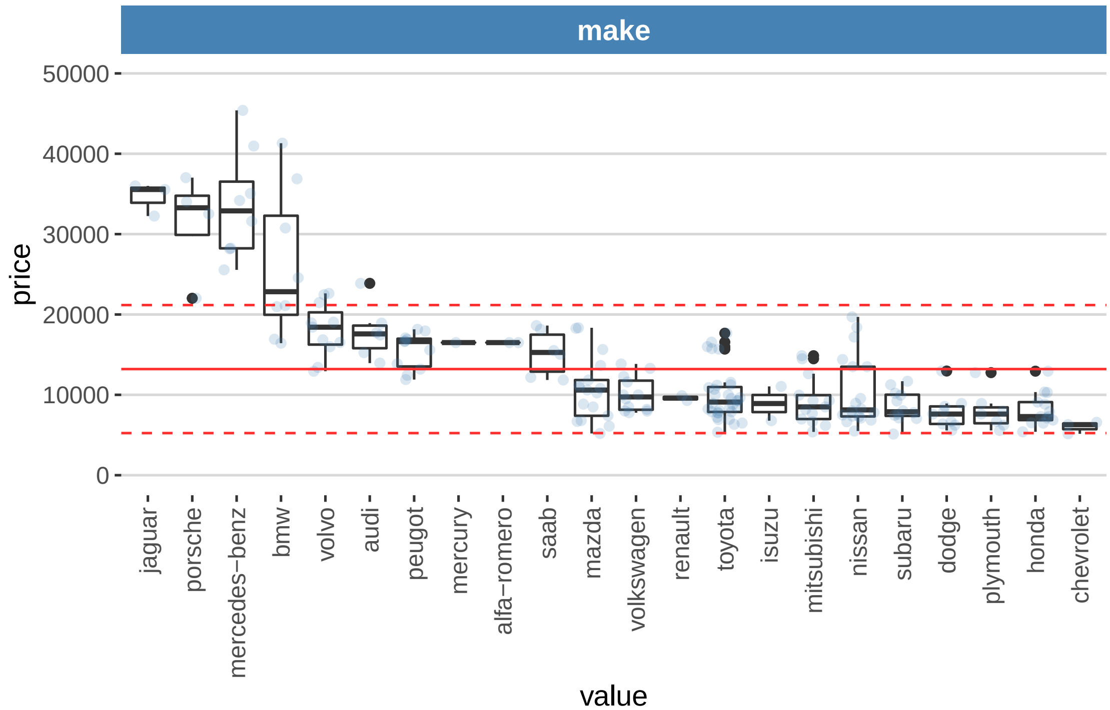

```
## [1] 6
```


price vs numerical chassis features
curb_weight
doors
height
length
wheel_base
width
price
value

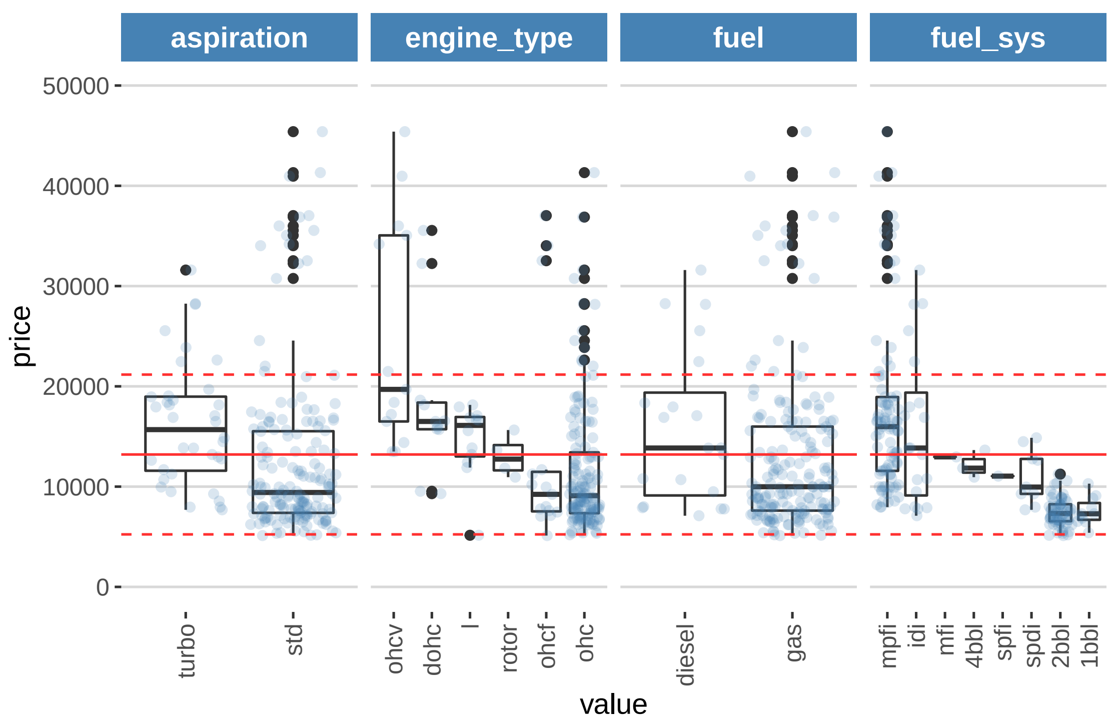
```
## [1] 4
```

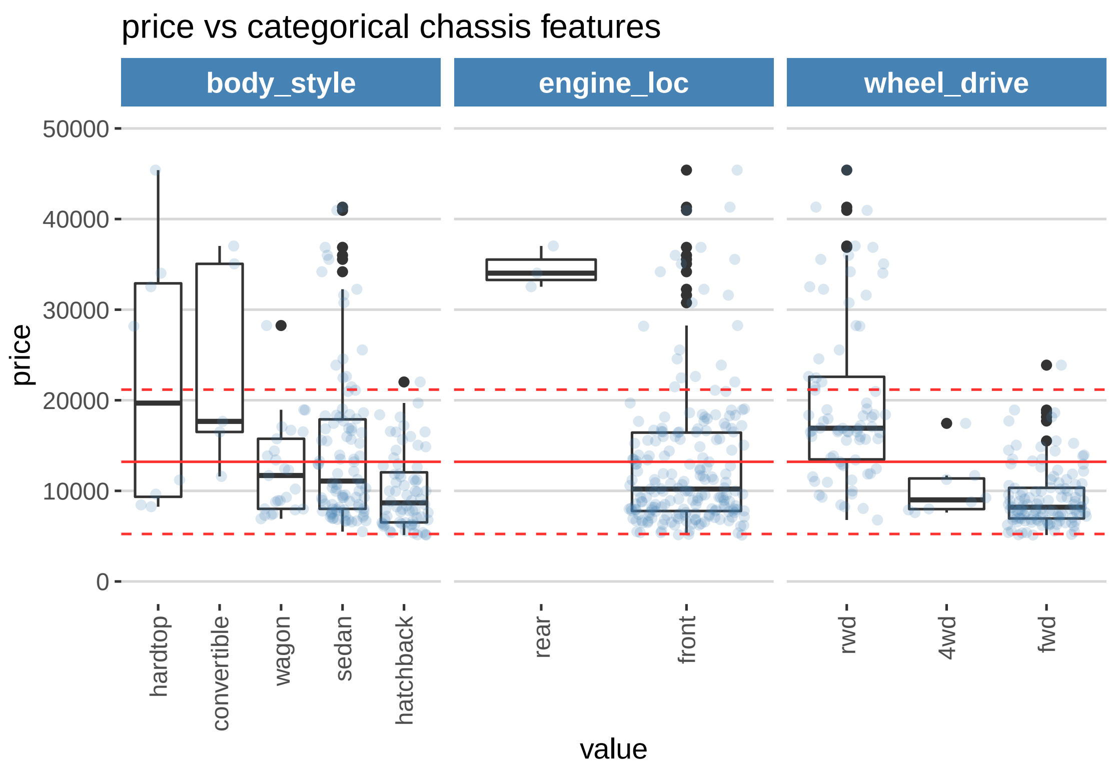

price vs numerical economic features
city_mpg
hwy_mpg
losses
symboling
price
value

price vs make




price vs categorical engine features






The attributes have quite different relations to the price. Some show a steep regression line, indicating an high influence on price, others are quite flat.

An extreme example are doors, with nearly a flat line near to the mean.

city_mpg and hwy_mpg seem to be quite similar and might be correlated.

In this project we will do not manual variable reduction, we let caret do it if applicable. But the information in the graphs would help, if we wanted to do with our expertise.

# 8 Data pre-processing #2

## 8.1 About Reproducibility

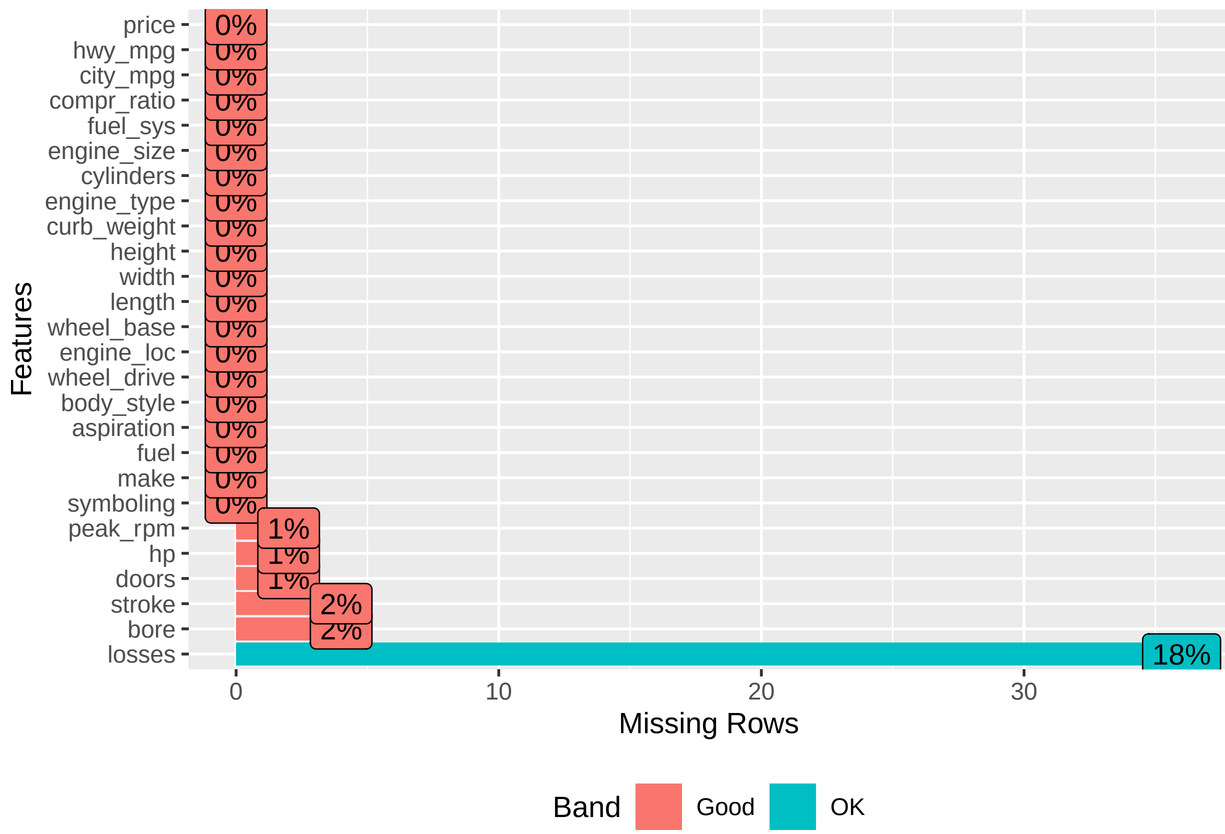
Some methods use 'random' procedures (e.g. bagging, folds split for cross validation, ...) influencing the results. In order to achive reproducable results, 'set.seed# is used before relevant functions (e.g. train-function, data splitting,..). We ignore the warning for non-uniform 'Rounding.

*https://topepo.github.io/caret/model-training-and-tuning.html#notes-on-reproducibility*

## 8.2 Impute data

As we have missing values, we can visualize them:

```
# Check for missing data
  plot_missing(am) #dataexplorer function
```




Losses has even 18% missing data. Three options:

1. Skip the losses attribute. Yet we do not know if we would loose valuable information.
2. Remove rows containing missing values. We would loose 18% of our (anyway small) dataset!
3. Replace values by an estimate.

Let's go for option 3 and apply this also to the other missing values. The caret preProcess function supports us:

```r
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Impute data ------------------------------------------------------------
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

set.seed(803, sample.kind = 'Rounding')  #
```

```
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

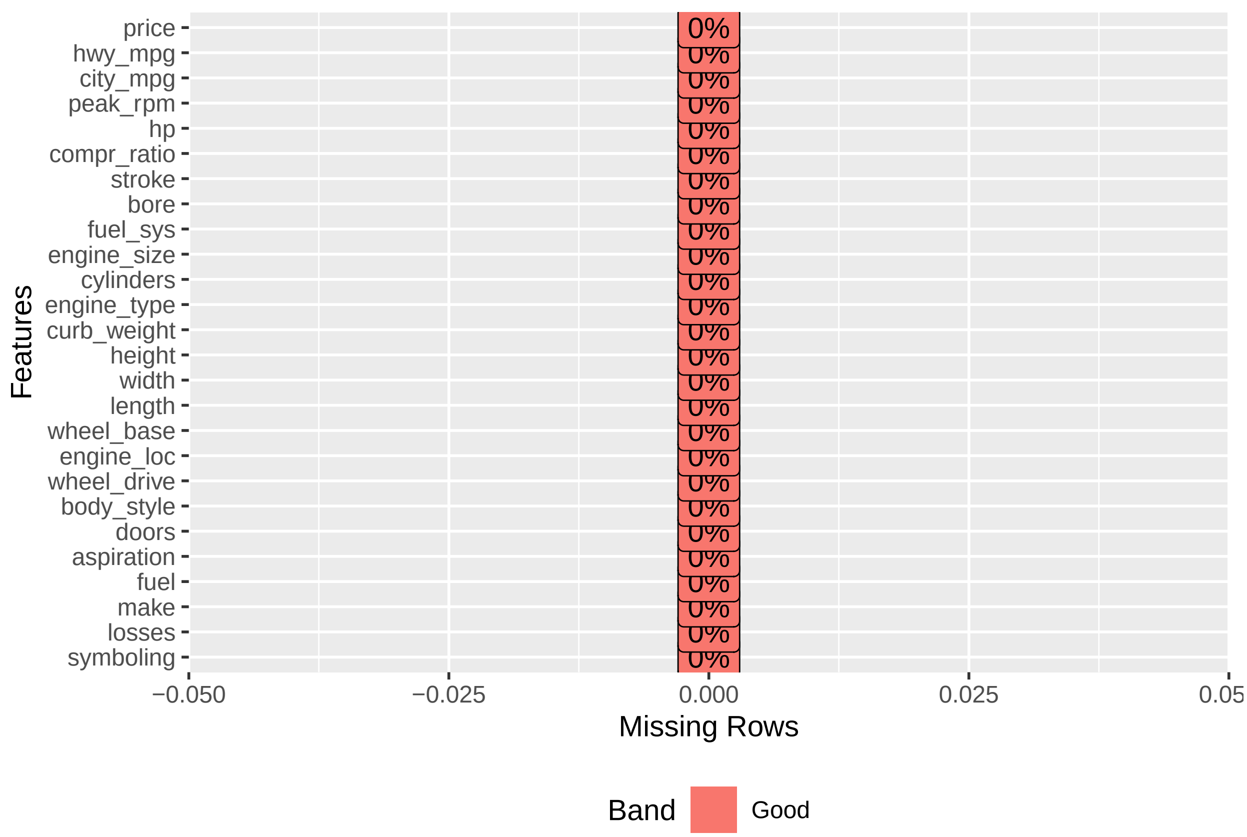
```r
# train a prediction model for missing values
am.pp.train <-preProcess(am, method = "bagImpute", k=5)

# predict the missing values based on the trained model
am.pp <- predict( am.pp.train, am)  #
sum(is.na(am.pp))
```

```
## [1] 0
```

---

The pre-processed am.pp data-set shows no missing values:

```r
# Check again for missing data
  plot_missing(am.pp)
```



## 8.3 Create dummy variables

(*Dummy variables*](https://dss.princeton.edu/online_help/analysis/dummy_variables.htm) represent categorical values as a number.

Again caret provides a function for this task.

Before we apply the function to all data, an example for engine_type:

---

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Dummy variables example -------------------------------------------------
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# create the dummy variables for engine_type
dummies <- dummyVars(price ~ engine_type, am.pp)
dummies
## Dummy Variable Object
##
## Formula: price ~ engine_type
## 2 variables, 1 factors
## Variables and levels will be separated by '.'
## A less than full rank encoding is used

# apply dummies to data
predict(dummies, am.pp)  %>% unique()  %>% kable()
```

| | engine_type.dohc | engine_type.l | engine_type.ohc | engine_type.ohcf | engine_type.ohcv | engine_type.rotor |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 |
| 17 | 0 | 1 | 0 | 0 | 0 | 0 |
| 52 | 0 | 0 | 0 | 0 | 0 | 1 |
| 123 | 0 | 0 | 0 | 1 | 0 | 0 |

```
predict(dummies, am.pp)  %>% colSums() %>% kable()
```

| | x |
|---|---|
| engine_type.dohc | 11 |
| engine_type.l | 12 |
| engine_type.ohc | 145 |
| engine_type.ohcf | 15 |
| engine_type.ohcv | 13 |
| engine_type.rotor | 4 |

---

Each engine_type is converted to an own attribute with value either 0 or 1.

Create dummyVars for the complete data:

---

```r
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Create dummy variables  -------------------------------------------------
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# create dummy variables for all attributes ( applied only to categorical data )
dummies <- dummyVars(price ~ ., am.pp)

# apply dummies to data
am.pp.dum.train <- predict(dummies, am.pp)
str(am.pp.dum.train)
```

```
##  num [1:200, 1:67] 3 1 2 2 2 1 1 1 2 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:200] "1" "2" "3" "4" ...
##   ..$ : chr [1:67] "symboling" "losses" "make.alfa-romero" "make.audi" ...
```

```r
class(am.pp.dum.train)
```

```
## [1] "matrix" "array"
```

```r
# am.pp.dum.train is a matrix without the price.
# generate the complete  data.frame including price:
am.pp.dum <- data.frame(price=am.pp$price, am.pp.dum.train)

skim(am.pp.dum)
```

Table 14: Data summary

| | |
|---|---|
| Name | x |
| Number of rows | 200 |
| Number of columns | 68 |
| | |
| Column type frequency: | |
| numeric | 68 |
| | |
| Group variables | None |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| price | 0 | 1 | 13205.69 | 7966.98 | 5118.00 | 7775.00 | 10270.00 | 16500.75 | 45400.00 |
| symboling | 0 | 1 | 0.83 | 1.25 | -2.00 | 0.00 | 1.00 | 2.00 | 3.00 |
| losses | 0 | 1 | 126.07 | 35.85 | 65.00 | 95.00 | 119.29 | 154.49 | 256.00 |
| make.alfa.romero | 0 | 1 | 0.01 | 0.10 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.audi | 0 | 1 | 0.03 | 0.17 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.bmw | 0 | 1 | 0.04 | 0.20 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.chevrolet | 0 | 1 | 0.01 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.dodge | 0 | 1 | 0.04 | 0.21 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.honda | 0 | 1 | 0.06 | 0.25 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.isuzu | 0 | 1 | 0.01 | 0.10 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.jaguar | 0 | 1 | 0.01 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.mazda | 0 | 1 | 0.09 | 0.28 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.mercedes.benz | 0 | 1 | 0.04 | 0.20 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| make.mercury | 0 | 1 | 0.00 | 0.07 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.mitsubishi | 0 | 1 | 0.06 | 0.25 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.nissan | 0 | 1 | 0.09 | 0.29 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.peugot | 0 | 1 | 0.06 | 0.23 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.plymouth | 0 | 1 | 0.04 | 0.18 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.porsche | 0 | 1 | 0.02 | 0.14 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.renault | 0 | 1 | 0.01 | 0.10 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.saab | 0 | 1 | 0.03 | 0.17 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.subaru | 0 | 1 | 0.06 | 0.24 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.toyota | 0 | 1 | 0.16 | 0.37 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.volkswagen | 0 | 1 | 0.06 | 0.24 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| make.volvo | 0 | 1 | 0.06 | 0.23 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel.diesel | 0 | 1 | 0.10 | 0.30 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel.gas | 0 | 1 | 0.90 | 0.30 | 0.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| aspiration.std | 0 | 1 | 0.82 | 0.39 | 0.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| aspiration.turbo | 0 | 1 | 0.18 | 0.39 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| doors | 0 | 1 | 3.15 | 0.99 | 2.00 | 2.00 | 4.00 | 4.00 | 4.00 |
| body_style.convertible | 0 | 1 | 0.03 | 0.16 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| body_style.hardtop | 0 | 1 | 0.04 | 0.20 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| body_style.hatchback | 0 | 1 | 0.34 | 0.47 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| body_style.sedan | 0 | 1 | 0.47 | 0.50 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| body_style.wagon | 0 | 1 | 0.12 | 0.33 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| wheel_drive.4wd | 0 | 1 | 0.04 | 0.20 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| wheel_drive.fwd | 0 | 1 | 0.59 | 0.49 | 0.00 | 0.00 | 1.00 | 1.00 | 1.00 |
| wheel_drive.rwd | 0 | 1 | 0.37 | 0.48 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| engine_loc.front | 0 | 1 | 0.98 | 0.12 | 0.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| engine_loc.rear | 0 | 1 | 0.01 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| wheel_base | 0 | 1 | 98.85 | 6.04 | 86.60 | 94.50 | 97.00 | 102.40 | 120.90 |
| length | 0 | 1 | 174.23 | 12.35 | 141.10 | 166.68 | 173.20 | 183.50 | 208.10 |
| width | 0 | 1 | 65.90 | 2.10 | 60.30 | 64.18 | 65.50 | 66.67 | 72.00 |
| height | 0 | 1 | 53.79 | 2.43 | 47.80 | 52.00 | 54.10 | 55.52 | 59.80 |
| curb_weight | 0 | 1 | 2555.70 | 518.59 | 1488.00 | 2163.00 | 2414.00 | 2928.25 | 4066.00 |
| engine_type.dohc | 0 | 1 | 0.06 | 0.23 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| engine_type.l | 0 | 1 | 0.06 | 0.24 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| engine_type.ohc | 0 | 1 | 0.72 | 0.45 | 0.00 | 0.00 | 1.00 | 1.00 | 1.00 |
| engine_type.ohcf | 0 | 1 | 0.07 | 0.26 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| engine_type.ohcv | 0 | 1 | 0.06 | 0.25 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| engine_type.rotor | 0 | 1 | 0.02 | 0.14 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| cylinders | 0 | 1 | 4.36 | 1.06 | 2.00 | 4.00 | 4.00 | 4.00 | 12.00 |
| engine_size | 0 | 1 | 126.86 | 41.65 | 61.00 | 97.75 | 119.50 | 142.00 | 326.00 |
| fuel_sys.1bbl | 0 | 1 | 0.06 | 0.23 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel_sys.2bbl | 0 | 1 | 0.32 | 0.47 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| fuel_sys.4bbl | 0 | 1 | 0.01 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel_sys.idi | 0 | 1 | 0.10 | 0.30 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel_sys.mfi | 0 | 1 | 0.00 | 0.07 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel_sys.mpfi | 0 | 1 | 0.46 | 0.50 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 |
| fuel_sys.spdi | 0 | 1 | 0.04 | 0.21 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| fuel_sys.spfi | 0 | 1 | 0.00 | 0.07 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |
| bore | 0 | 1 | 3.32 | 0.27 | 2.54 | 3.14 | 3.31 | 3.58 | 3.94 |
| stroke | 0 | 1 | 3.26 | 0.31 | 2.07 | 3.12 | 3.29 | 3.41 | 4.17 |
| compr_ratio | 0 | 1 | 10.17 | 4.01 | 7.00 | 8.57 | 9.00 | 9.40 | 23.00 |
| hp | 0 | 1 | 103.23 | 37.48 | 48.00 | 70.00 | 95.00 | 116.00 | 262.00 |

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| peak_rpm | 0 | 1 | 5117.53 | 479.41 | 4150.00 | 4800.00 | 5180.75 | 5500.00 | 6600.00 |
| city_mpg | 0 | 1 | 25.20 | 6.43 | 13.00 | 19.00 | 24.00 | 30.00 | 49.00 |
| hwy_mpg | 0 | 1 | 30.70 | 6.83 | 16.00 | 25.00 | 30.00 | 34.00 | 54.00 |

## 8.4 Split data in to test and training

We proceed with am.pp.dum and take 80% as training data and 20% as test data.

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Split data in to test and training ----------------------------------------
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
set.seed(803, sample.kind="Rounding")
```

```
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
train.id <-  createDataPartition(am.pp.dum$price, p = .8, list = FALSE, times = 1)

am.pp.dum.train <- am.pp.dum[train.id,]
am.pp.dum.test <- am.pp.dum[-train.id,]

#X and Y will serve for the validation of the test data.
X <- am.pp.dum.test[, !(names(am.pp.dum.test) %in% c('price'))]
Y <- am.pp.dum.test$price
```

For the modeling we use the data pre-processed till now. Please note, we only transformed the data. We converted the type of data, added missing values and created the dummy variables. The data is still complete, no information lost - rather enhanced via impute. No values where changed, just the appearance changed.

## 8.5 Define RMSE

For performance measurement of the prediction models we will use the Residual Mean Square Error (RMSE) and define a function for it:

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Define RMSE ---------------------------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

RMSE <- function(true_values, predicted_values){
  sqrt(mean((true_values - predicted_values)^2))}
```

# 9 Methods and techniques

As already mentioned in the intro, linear regression is the reference. The challengers are knn and random forest.

For every method we start with an out-of-the-box approach, by simply training a model with the default settings. We then try to improve by further pre-processing and parameter optimization.

## 9.1 Pre-processing

In contrast to the previous pre-processing, data will also be recalculated (values will change) and we also might skip variables. In previous pre-processing we used the caret functions to change the data as input for our models. In contrast, we will include further pre-processing into the caret train function - thus the data changes are done in the background.

## 9.2 Parameter optimization

Fore each method many parameters can be optimized. With caret we can handle some of them and will focus on those.

## 9.3 Modelling steps

For each method we follow the same process steps:

- Train
  - Train the model using the training-set with the methode.
  - If applicable, define the pre-processing functions and tuning parameters.
- Predict
  - Based on the trained model, predictions are calculated with the test-set.
- Measure
  - The RMSE will be determined by comparing true_values vs. predicted_values.
  - The duration of the model-training is stored as runtime in seconds.
- Store results
  - Detailed results for each datapoint (true_rating, predicted_rating) will be stored for each method and will be used for model comparisons.
  - Overall results (RMSE & computation time) plus extended info on the model will be stored for a result overview.
- Review results
  - The result overview helps us to compare the results of all the methods.
  - A graphical comparison of selected methods visualises the results by plotting the predicted_values vs. true_values. An 'ideal line' serves as reference. The selection consists of the latest prediction and the best-of-within-method predictions (e.g. only the best knn model is kept).
  - If applicable, the train-model is printed and plotted.
  - Based on the review, conclusions and/or decisions for the further procedure are taken.

# 10 Linear Regression

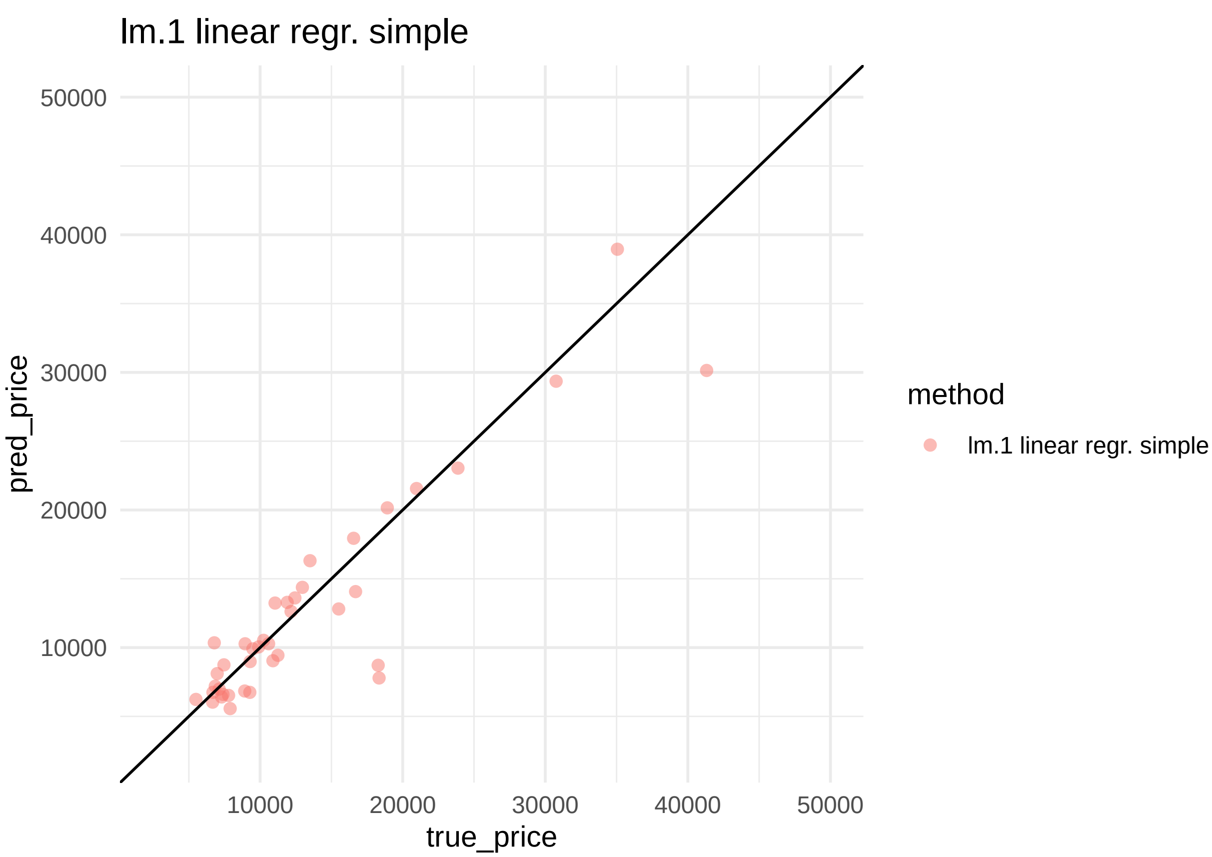
Linear Regression shall be the reference for comparing knn and random forest results.

## 10.1 Linear Regression simple (lm.1)

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# lm.1 LinReg simple ---------------------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  # Method
    method='lm.1 linear regr. simple'

  # Train
    start <- as.numeric(Sys.time())
      set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
      train_lm.1 <- train(price ~ .,method = "lm",
                          data = am.pp.dum.train)
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
    end <- as.numeric(Sys.time())

   # Predict
    pred.lm.1 <- predict(train_lm.1, newdata = X)
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading


  # Measure
    RMSE.lm.1 <- RMSE(Y, pred.lm.1)
    rmse <- RMSE.lm.1
    runtime = ceiling( end - start )
```

```r
# Store results

  comment = 'no train parameters ; warnings'

  results.details <- data.frame(method = method,
                                true_price = Y, pred_price = pred.lm.1)

  results <- data.frame(method = method, RMSE = rmse,
                        runtime = runtime, comment = comment)

  slctn.results <- c(method)

# Review results
  results.details %>% filter(method %in% all_of(slctn.results)) %>%
    ggplot(aes(true_price, pred_price, col=method, shape=method)) +
    geom_point(size=2, alpha=0.5) +
    geom_abline(slope=1, intercept = 0) +
    ylim(min.price*0.5,max.price*1.1) +
    xlim(min.price*0.5,max.price*1.1) +
    ggtitle(method) + theme_minimal()
```




```r
  train_lm.1
## Linear Regression
##
## 161 samples
```

```
##  67 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 161, 161, 161, 161, 161, 161, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   3088.753  0.8592917  2061.568
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
kable(results)
```

| method | RMSE | runtime | comment |
|---|---:|---:|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |

---

> In the graph we see the results mostly close to the 'ideal'-line with some outliers.
>
> The RMSE of 3307.2580721 will be the first reference for the 'challenge' against the upcoming models.

What about the warning messages we got?

We got a result anyway and it was a warning, not an error. We could just ignore it.

Here we get support: https://www.statology.org/prediction-from-rank-deficient-fit-may-be-misleading/

The website names two possible reasons:

- You have more model parameters than observations in the dataset.
  - Here are the dimensions of our training data: " 161, 68 " ; hence we can exclude this reason.
- Two predictor variables are perfectly correlated.
  - Can we get rid of the warning messages with pre-processing the data considering correlation?

    https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/preProcess

## 10.2 Additional pre-processing

Before we go for the next models with all the process steps, we just give a trial to training models with pre-processing and review if the warning messages show up again.

We include preProcess for

- zv = identify and remove 'zero variance'
- nzv = as above for 'near zero variance'
- corr = seeks to filter out highly correlated predictors
- center = subtracts the mean of the predictor's data
- scale = divides by the standard deviation

Of course this selection covers more than the identified issue of correlation. The generel intention is to improve also the RMSE.

---

```
# review for warnings with preProcess
train_lm.2 <- train(price ~ .,method = "lm",
                  preProcess = c('zv','nzv','corr','center', 'scale'),
                  data = am.pp.dum.train)
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
train_lm.2
## Linear Regression
##
## 161 samples
##  67 predictor
##
## Pre-processing: centered (37), scaled (37), remove (30)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 161, 161, 161, 161, 161, 161, ...
## Resampling results:
##
##   RMSE     Rsquared   MAE
##   3418.51  0.8195268  2411.542
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

---

We still get warnings.

*PCA* might remove the warnings, because the components are lineary uncorrelated. PCA will be added to preProcess:

---

```
# review for warnings with preProcess
train_lm.3 <- train(price ~ .,method = "lm",
                 preProcess  = c('zv','nzv','corr','center', 'scale','pca'),
                 data = am.pp.dum.train)

train_lm.3
## Linear Regression
##
## 161 samples
##  67 predictor
##
## Pre-processing: centered (37), scaled (37), principal component
##  signal extraction (37), remove (30)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 161, 161, 161, 161, 161, 161, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   3414.668  0.8190429  2524.39
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

---

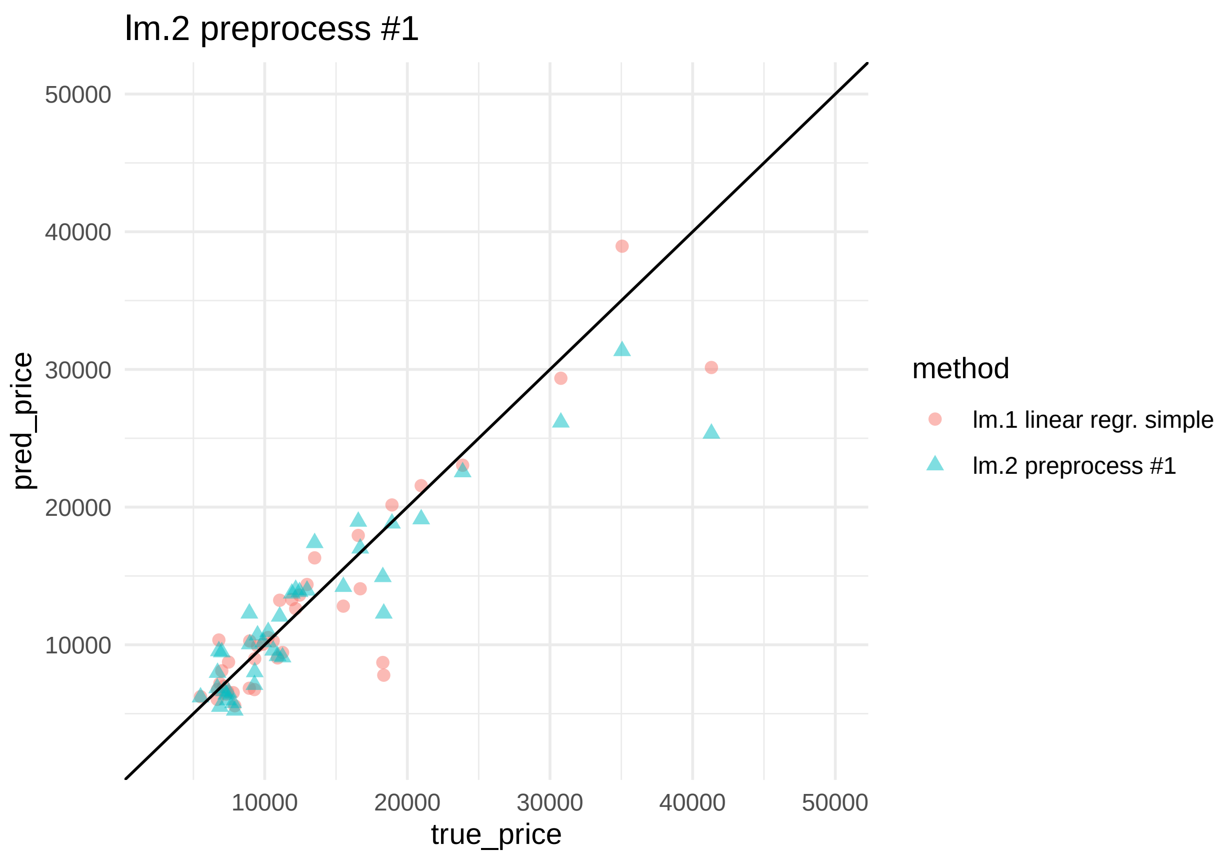
PCA removed variables, seems including the 'problem' variables. We will apply those two train variants to the test data.

### 10.3 Linear Regression with pre-processing #1 (lm.2)

We apply train_lm.2 as described before, but run the complete prodecure. The warnings will be not shown, we already know them.

---

```r
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# LinReg Preprocess V2 ----------------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  # Method
    method = 'lm.2 preprocess #1'

  # Train
    start <- as.numeric(Sys.time())
      set.seed(803, sample.kind = 'Rounding')
      train_lm.2 <-
        train(price ~ .,method = "lm",
              preProcess  = c('zv','nzv','corr','center', 'scale'),
              data = am.pp.dum.train)
    end <- as.numeric(Sys.time())

  # Predict
    pred.lm.2 <- predict(train_lm.2, newdata = X)

    # Measure
    RMSE.lm.2 <- RMSE(Y, pred.lm.2)
    rmse <- RMSE.lm.2
    runtime = ceiling( end - start )

  # Store results
    comment = 'preProcess zv/nzv/corr/center/scale; warnings'

    results.details.temp <- data.frame(method = method,
                                       true_price = Y, pred_price = pred.lm.2)
    results.details <- bind_rows(results.details, results.details.temp)

    results.temp <- data.frame(method = method,
                               RMSE = rmse, runtime = runtime, comment = comment)
    results <- bind_rows (results, results.temp)

    slctn.results <- c(slctn.results, method)

  # Review results
    results.details %>% filter(method %in% all_of(slctn.results)) %>%
      ggplot(aes(true_price, pred_price, col=method, shape=method)) +
      geom_point(size=2, alpha=0.5) +
      geom_abline(slope=1, intercept = 0) +
      ylim(min.price*0.5,max.price*1.1) +
      xlim(min.price*0.5,max.price*1.1) +
      ggtitle(method) + theme_minimal()
```




```
train_lm.2
## Linear Regression
##
## 161 samples
##  67 predictor
##
## Pre-processing: centered (37), scaled (37), remove (30)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 161, 161, 161, 161, 161, 161, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   3666.254  0.8035365  2476.482
##
## Tuning parameter 'intercept' was held constant at a value of TRUE

kable(results)
```

| method | RMSE | runtime | comment |
|---|---|---|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |

---

Despite preprocessing, the result is worse then 'lm.1' and still has warnings. For the graph-results we keep lm.1 a reference.

```
# Do not keep method in results graph
    slctn.results <- slctn.results[slctn.results  !='lm.2 preprocess #1']
```

## 10.4 Linear Regression with pre-processing #2 (lm.3)

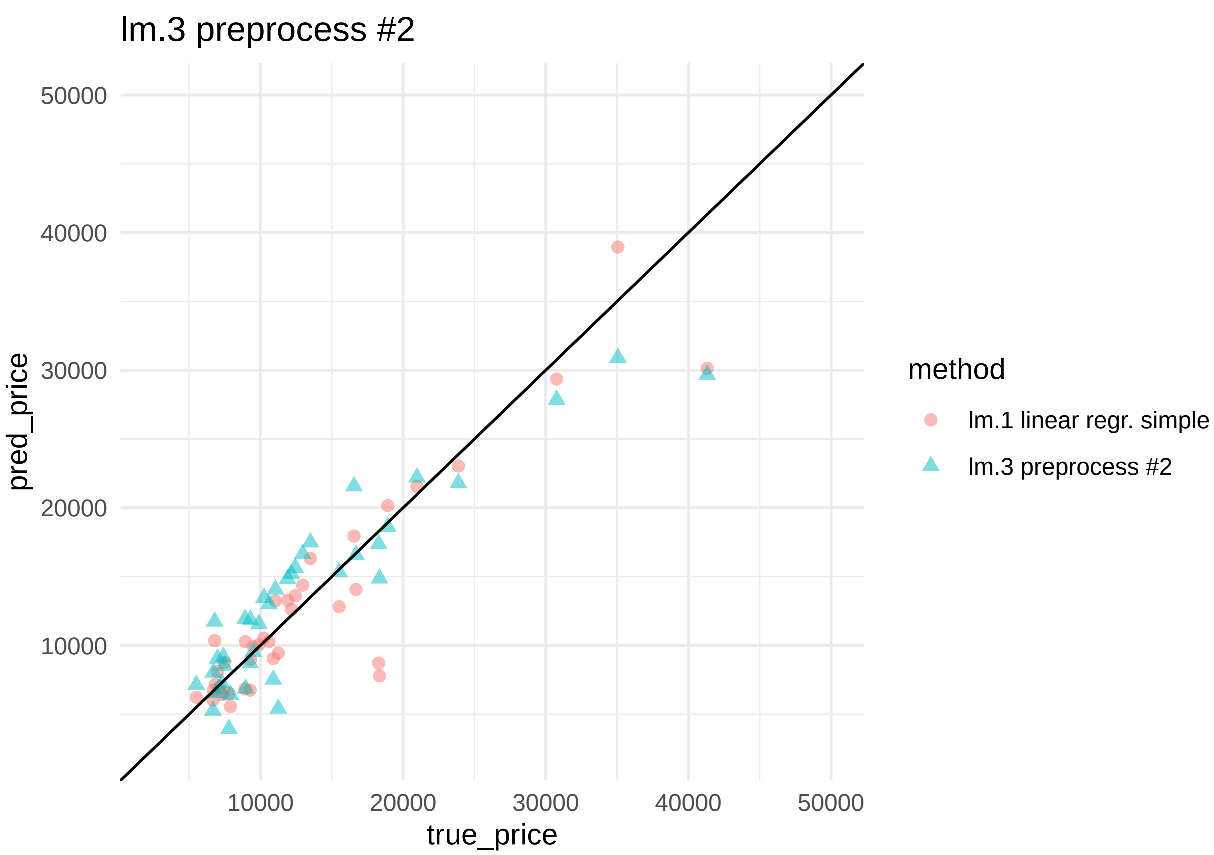
In lm.3 pca is added to preProcess. Warnings are not suppressed.

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# LinReg Preprocess V3 ----------------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  # Method
    method = 'lm.3 preprocess #2'

  # Train
    start <- as.numeric(Sys.time())
    set.seed(803, sample.kind = 'Rounding')
```

```
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
    train_lm.3 <-
      train(price ~ .,method = "lm",
                          preProcess  = c('zv','nzv','corr','center', 'scale','pca'),
                          data = am.pp.dum.train)
    end <- as.numeric(Sys.time())

  # No warnings!

  # Predict
    pred.lm.3 <- predict(train_lm.3, newdata = X)

  # Again no warnings!

  # Measure
    RMSE.lm.3 <- RMSE(Y, pred.lm.3)
    rmse <- RMSE.lm.3
    runtime = ceiling( end - start )

  # Store results
    comment = 'all of lm.2 + pca ; no warnings'

    results.details.temp <- data.frame(method = method, true_price = Y, pred_price = pred.lm.3)
    results.details <- bind_rows(results.details, results.details.temp)

    results.temp <- data.frame(method = method, RMSE = rmse, runtime = runtime, comment = comment)
    results <- bind_rows (results, results.temp)

    slctn.results <- c(slctn.results, method)

  # Review results
```

```r
results.details %>% filter(method %in% all_of(slctn.results)) %>%
  ggplot(aes(true_price, pred_price, col=method, shape=method)) +
  geom_point(size=2, alpha=0.5) +
  geom_abline(slope=1, intercept = 0) +
  ylim(min.price*0.5,max.price*1.1) +
  xlim(min.price*0.5,max.price*1.1) +
  ggtitle(method) + theme_minimal()
```




```r
train_lm.3
```

```
## Linear Regression
##
## 161 samples
##  67 predictor
##
## Pre-processing: centered (37), scaled (37), principal component
##  signal extraction (37), remove (30)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 161, 161, 161, 161, 161, 161, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   3687.114  0.7962284  2670.344
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
kable(results)
```

| method | RMSE | runtime | comment |
|---|---|---|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |

---

No warnings and slightly better then 'lm.1'. We keep only the best performing linear regression model as graphical reference.

```
# Do not keep method in results graph
    slctn.results <- slctn.results[slctn.results  !='lm.1 linear regr. simple']
```

# 11 knn intro

'k-nearest neighbors', short knn, is a popular machine learning algorithm. It is basically more a classification method, but can also be applied for regression.

We can optimize (tune) k (how many nearest neighbours to be considered) with the caret.

We use trainControl to apply cross-validation, to find the best k for our model.

*https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm*

*http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/142-knn-k-nearest-neighbors-essentials/*

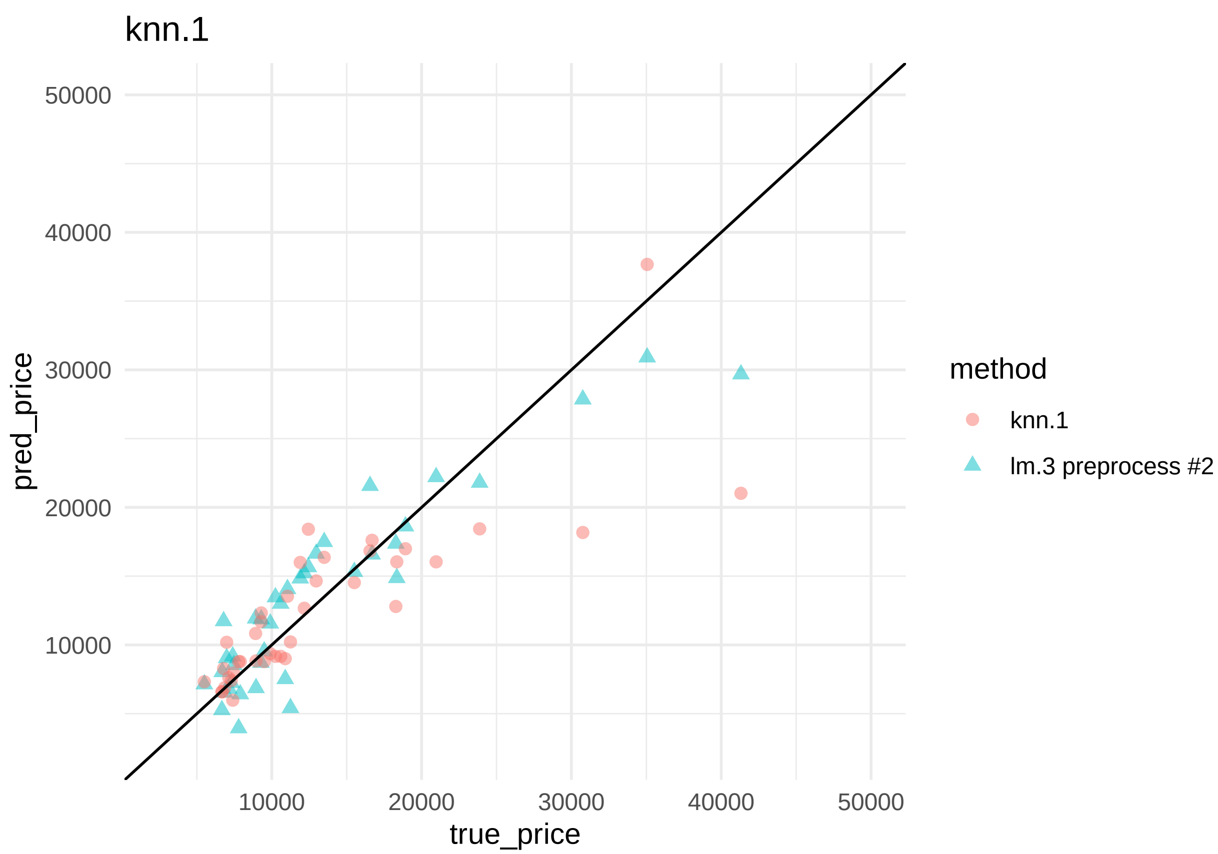
## 11.1 knn simple (knn.1)

Simple knn is just out of the box.

---

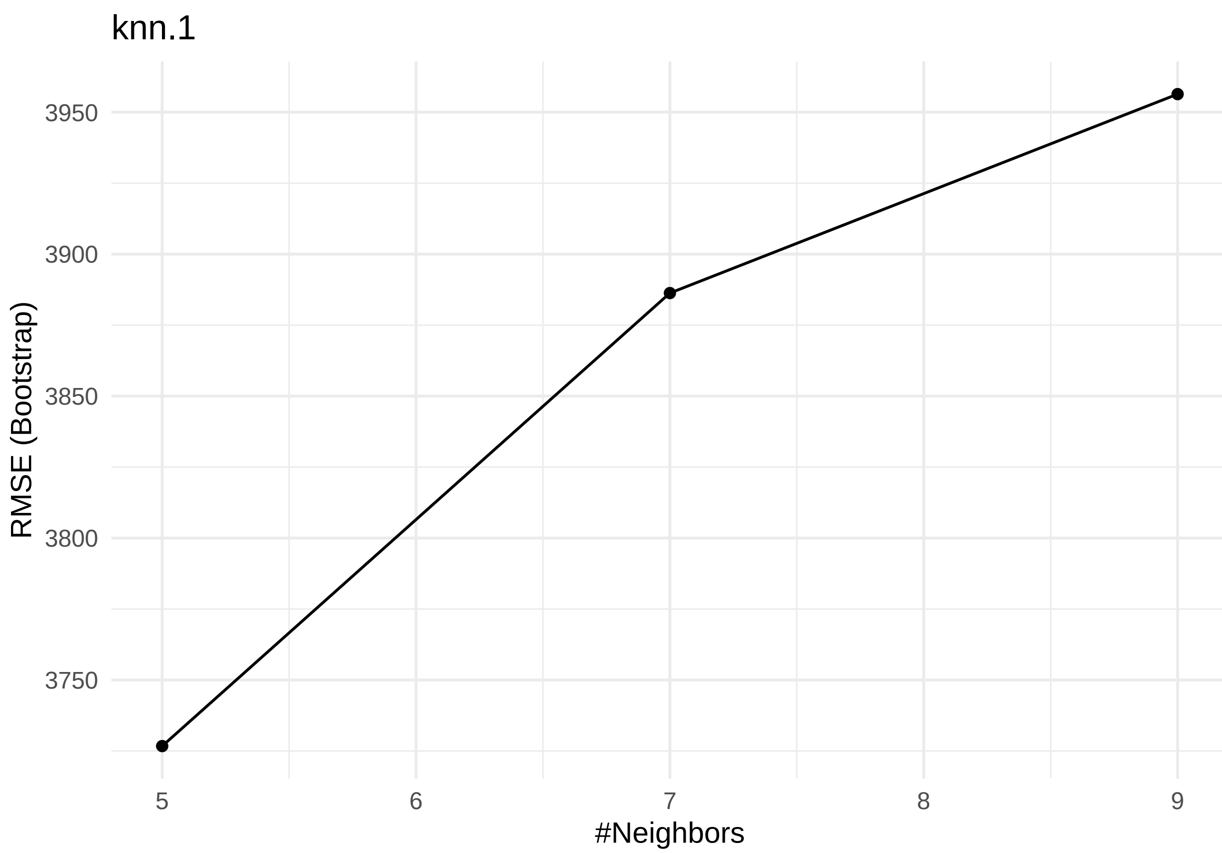
```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# # knn.1 simple ----------------------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  # Method
    method = 'knn.1'

  # Train
    start <- as.numeric(Sys.time())
      set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
      train_knn.1 <-
        train(price ~ ., method = "knn", data = am.pp.dum.train)
    end <- as.numeric(Sys.time())

  # Predict
    pred.knn.1 <- predict(train_knn.1, newdata = X)

  # Measure
    RMSE.knn.1 <- RMSE(Y, pred.knn.1)
    RMSE.knn.1
## [1] 4497.96
    rmse <- RMSE.knn.1
    runtime = ceiling( end - start )

  # Store results

    comment = 'knn.1 no tuning'

    results.details.temp <- data.frame(method = method,
                                       true_price = Y, pred_price = pred.knn.1)
    results.details <- bind_rows(results.details, results.details.temp)

    results.temp <- data.frame(method = method, RMSE = rmse,
                               runtime = runtime, comment = comment)
    results <- bind_rows (results, results.temp)
```

```r
  slctn.results <- c(slctn.results, method)

# Review results
  results.details %>% filter(method %in% all_of(slctn.results)) %>%
    ggplot(aes(true_price, pred_price, col=method, shape=method)) +
    geom_point(size=2, alpha=0.5) +
    geom_abline(slope=1, intercept = 0) +
    ylim(min.price*0.5,max.price*1.1) +
    xlim(min.price*0.5,max.price*1.1) +
    ggtitle(method) + theme_minimal()
```




```r
  train_knn.1
## k-Nearest Neighbors
##
## 161 samples
##  67 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 161, 161, 161, 161, 161, 161, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared   MAE
##   5  3726.722  0.7804540  2324.707
##   7  3886.280  0.7643184  2390.621
```

```
##   9  3956.361  0.7616217  2392.202
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.

    ggplot(train_knn.1) +
      ggtitle(method) + theme_minimal()
```




```
    kable(results)
```

| method | RMSE | runtime | comment |
|---|---|---|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |

---

In the graph we see more outliers in knn. This visual impression is supported by higher RMSE.

As graphical reference, we still keep knn.1 as the best knn result so far.

We see minimal k = 5 and it is bestTune. The plot implies, that left of 5 might be lower RMSE's.

For resampling bootstrap was used.

---

## 11.2 knn with tuneGrid & trainControl (knn.2)

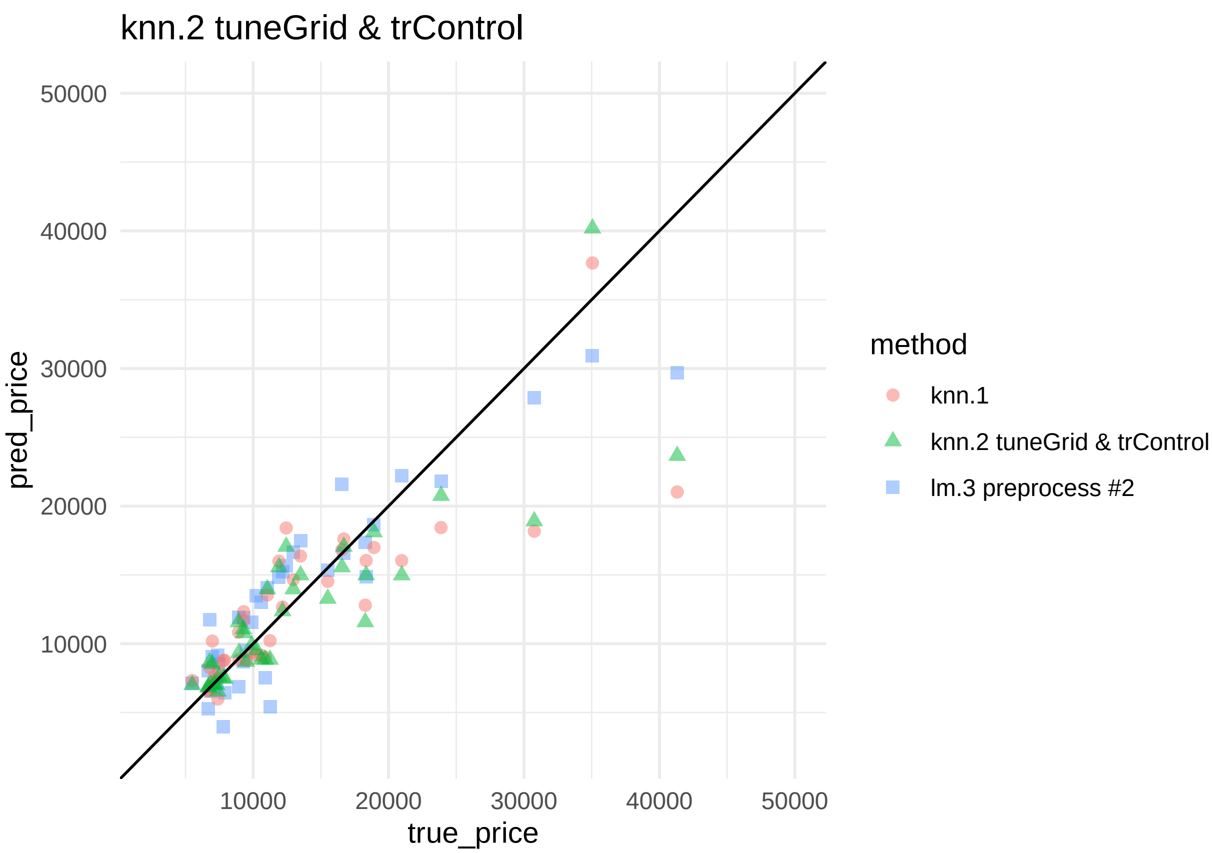
In this model we try to find a k with lower RMSE.

We use tuneGrid to specify the range of k=1:10 to run through and via trainControl we define to run 10-fold cross validation with 10 repetitions for resampling.

---

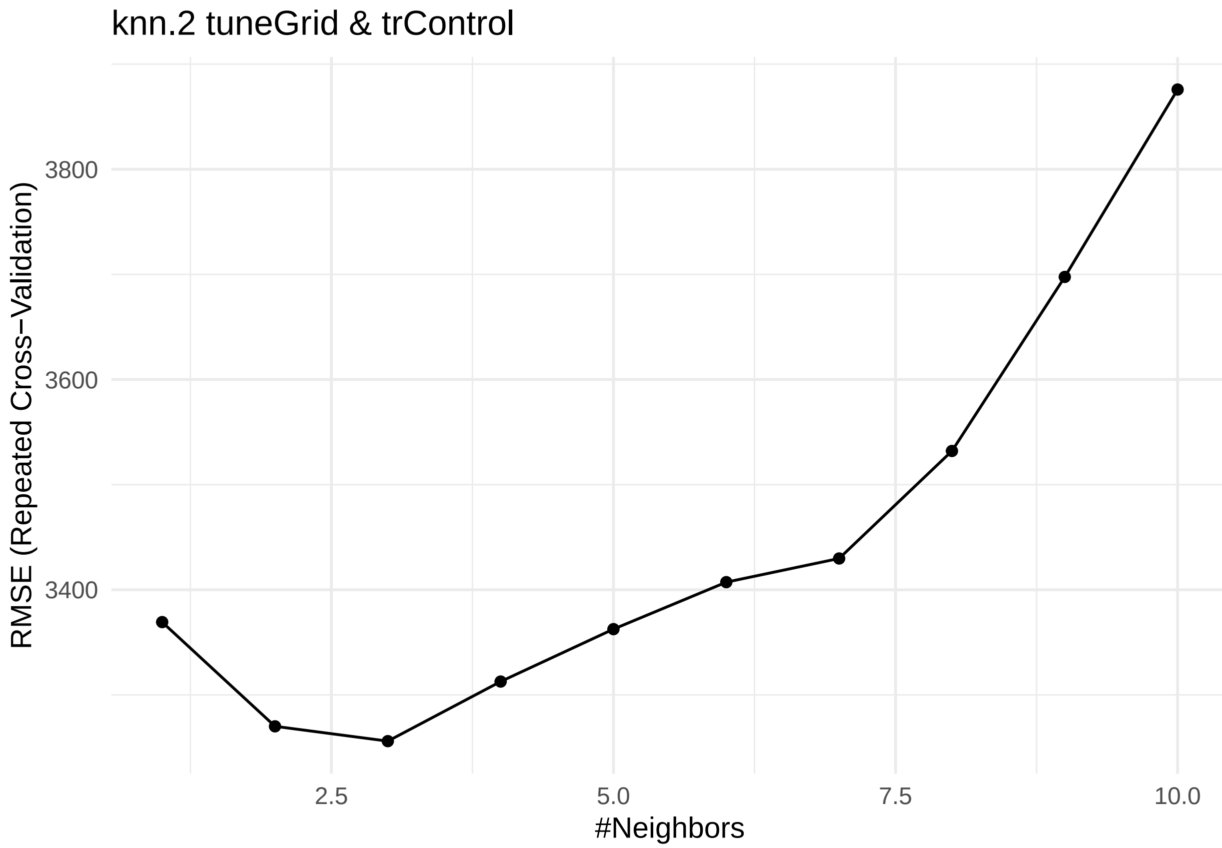
```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# # Train knn with tuneGrid & trainControl (knn.2) -------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


   # Method
    method = 'knn.2 tuneGrid & trControl'

  #Train
    start <- as.numeric(Sys.time())
      set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
        train_knn.2 <- train(price ~ ., method = "knn",
                        data = am.pp.dum.train,
                        trControl=trainControl(method="repeatedcv",
                                               number=10, repeats = 10),
                        tuneGrid = data.frame(k = seq(1:10)))
    end <- as.numeric(Sys.time())

  # Predict
    pred.knn.2 <- predict(train_knn.2, newdata = X)

  # Measure
    RMSE.knn.2 <- RMSE(Y, pred.knn.2)
    RMSE.knn.2
## [1] 4143.973
    rmse <- RMSE.knn.2
    runtime = ceiling( end - start )

  # Store results
    comment = 'knn.1 + tune k=1:10 & 10x repeated 10fold-CV '

    results.details.temp <- data.frame(method = method,
                                       true_price = Y, pred_price = pred.knn.2)
    results.details <- bind_rows(results.details, results.details.temp)

    results.temp <- data.frame(method = method, RMSE = rmse,
                               runtime = runtime, comment = comment)
    results <- bind_rows (results, results.temp)

    slctn.results <- c(slctn.results, method)

  # Review results
    results.details %>% filter(method %in% all_of(slctn.results)) %>%
      ggplot(aes(true_price, pred_price, col=method, shape=method)) +
      geom_point(size=2, alpha=0.5) +
      geom_abline(slope=1, intercept = 0) +
```

```
    ylim(min.price*0.5,max.price*1.1) +
    xlim(min.price*0.5,max.price*1.1) +
    ggtitle(method) + theme_minimal()
```




```
  train_knn.2
## k-Nearest Neighbors
##
## 161 samples
##  67 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 145, 145, 145, 145, 145, 145, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared   MAE
##    1  3369.250  0.8065340  2045.399
##    2  3270.119  0.8299522  2056.697
##    3  3256.014  0.8328236  2106.592
##    4  3312.698  0.8267962  2147.807
##    5  3362.569  0.8243466  2161.558
##    6  3407.248  0.8164449  2177.629
##    7  3429.766  0.8152679  2176.171
##    8  3532.022  0.8053904  2217.301
##    9  3697.571  0.7925091  2286.496
```

```
##   10  3875.797  0.7737250  2365.825
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 3.

    ggplot(train_knn.2) +
      ggtitle(method) + theme_minimal()
```




```
    kable(results)
```

| method | RMSE | runtime | comment |
|---|---:|---:|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |
| knn.2 tuneGrid & trControl | 4143.973 | 2 | knn.1 + tune k=1:10 & 10x repeated 10fold-CV |

---

Good improvement, best knn model so far. k=2 is below 5 as in the model before.

We skip knn.1 in results graph and keep knn.2.

---

```
#Do not keep knn.1 in results graph
  slctn.results <- slctn.results[slctn.results  !='knn.1']
```

## 11.3 knn with knn.2 + preProcess (knn.3)

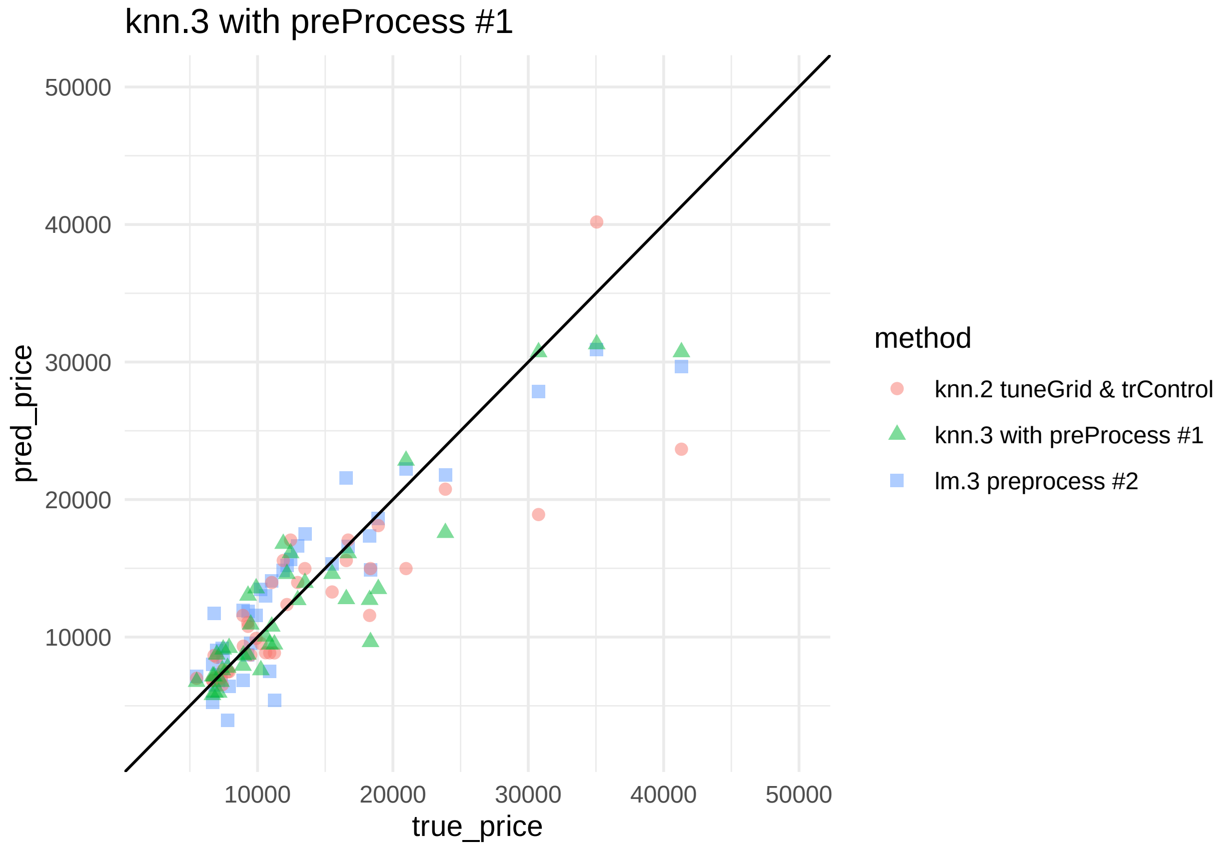
Can we still improve with pre-processing?

We train like knn.2 but preProcess with same preProcess steps as in lm.2.

---

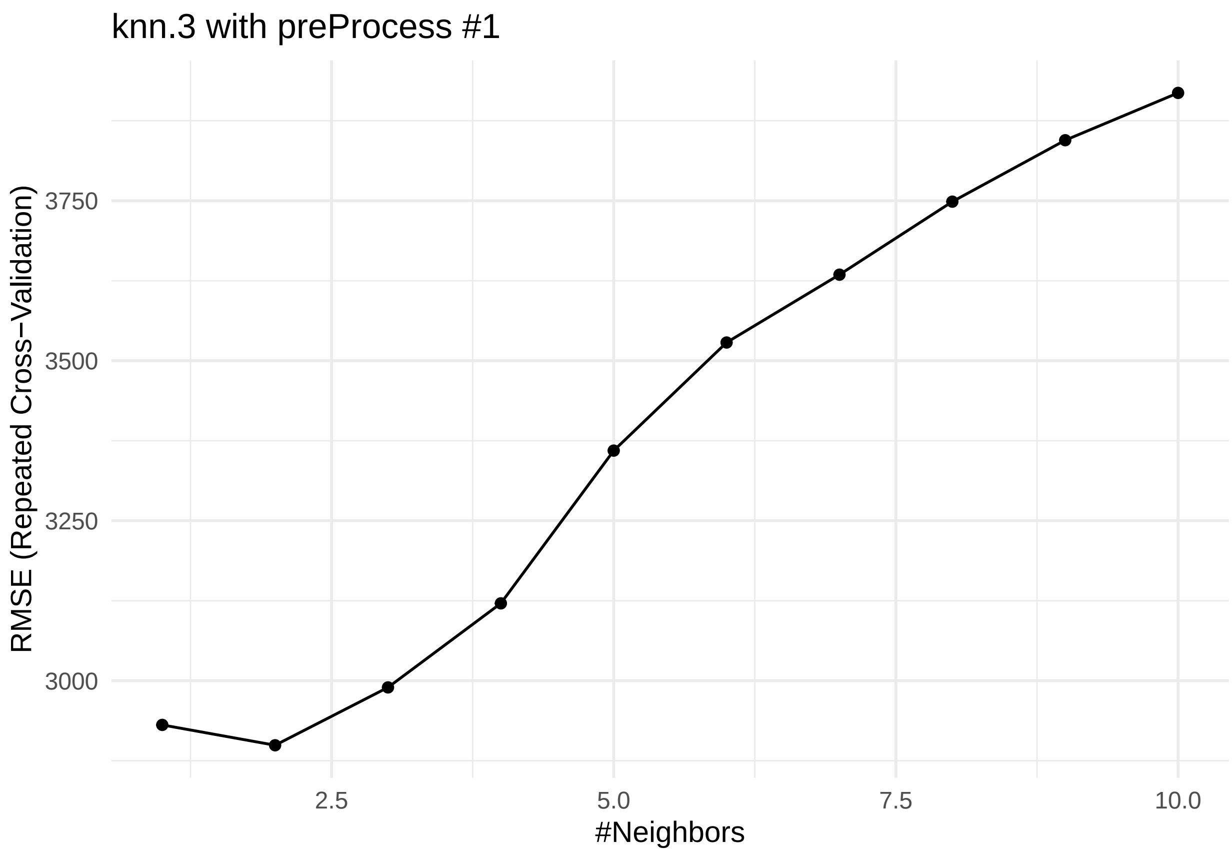
```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# # Train knn with tuneGrid and preprocess (knn.3) ------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  # Method
    method = 'knn.3 with preProcess #1'

    start <- as.numeric(Sys.time())
      set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
      train_knn.3 <- train(price ~ ., method = "knn",
                          data = am.pp.dum.train,
                          trControl=trainControl(method="repeatedcv",
                                                 number=10, repeats = 10),
                          tuneGrid = data.frame(k = seq(1:10)),
                          preProcess = c('zv','nzv','corr','center', 'scale') )
    end <- as.numeric(Sys.time())

  # Predict
    pred.knn.3 <- predict(train_knn.3, newdata = X)

  # Measure
    RMSE.knn.3 <- RMSE(Y, pred.knn.3)
    rmse <- RMSE.knn.3
    runtime = ceiling( end - start )

  # Store results
    comment = 'knn.2 + zv/nzv/corr/center/scale'

    results.details.temp <- data.frame(method = method,
                                       true_price = Y, pred_price = pred.knn.3)
    results.details <- bind_rows(results.details, results.details.temp)

    results.temp <- data.frame(method = method, RMSE = rmse,
                               runtime = runtime, comment = comment)
    results <- bind_rows (results, results.temp)

    slctn.results <- c(slctn.results, method)

  # Review results
    results.details %>% filter(method %in% all_of(slctn.results)) %>%
      ggplot(aes(true_price, pred_price, col=method, shape=method)) +
      geom_point(size=2, alpha=0.5) +
      geom_abline(slope=1, intercept = 0) +
      ylim(min.price*0.5,max.price*1.1) +
      xlim(min.price*0.5,max.price*1.1) +
```

```
    ggtitle(method) + theme_minimal()
```




```
    train_knn.3
## k-Nearest Neighbors
##
## 161 samples
##  67 predictor
##
## Pre-processing: centered (37), scaled (37), remove (30)
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 145, 145, 145, 145, 145, 145, ...
## Resampling results across tuning parameters:
##
##    k   RMSE      Rsquared   MAE
##     1  2930.932  0.8636920  2095.146
##     2  2899.129  0.8609385  2017.488
##     3  2989.533  0.8533701  2112.110
##     4  3120.852  0.8501430  2178.575
##     5  3359.530  0.8336161  2307.833
##     6  3528.442  0.8198942  2413.948
##     7  3634.476  0.8127941  2460.777
##     8  3748.546  0.8046522  2533.633
##     9  3844.534  0.7989287  2564.156
##    10  3918.468  0.7943555  2587.918
##
```

```
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 2.

    ggplot(train_knn.3) +
      ggtitle(method) + theme_minimal()
```




```
    kable(results)
```

| method | RMSE | runtime | comment |
|---|---:|---:|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |
| knn.2 tuneGrid & trControl | 4143.973 | 2 | knn.1 + tune k=1:10 & 10x repeated 10fold-CV |
| knn.3 with preProcess #1 | 3283.149 | 29 | knn.2 + zv/nzv/corr/center/scale |

---

Difficult to see in the graph, but the RMSE for knn.3 is lower as knn.2.

Nevertheless linear regression is still performing best.

The computation time severly increased.

Note that many variables were removed but the RMSE is improved.

We skip knn.2 in results graph and keep knn.3.

```
# Do not keep method in results graph
  slctn.results <- slctn.results[slctn.results  !='knn.2 tuneGrid & trControl']
```

## 12 Random Forest

From our HarvardX professor Rafael Irizari we learned (*Introduction to Data Science, 31.11*):

'Random forests are a very popular machine learning approach that addresses the shortcomings of decision trees using a clever idea. The goal is to improve prediction performance and reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness).'

For this project we use the package *Ranger*: 'A fast implementation of Random Forests, particularly suited for high dimensional data.'

*Tuning parameters in caret are:*

- mtry (mtry Number of variables to possibly split at in each node)
- splitrule (Splitting Rule)
- min.node.size (Minimal Node Size, default 5 for regression)

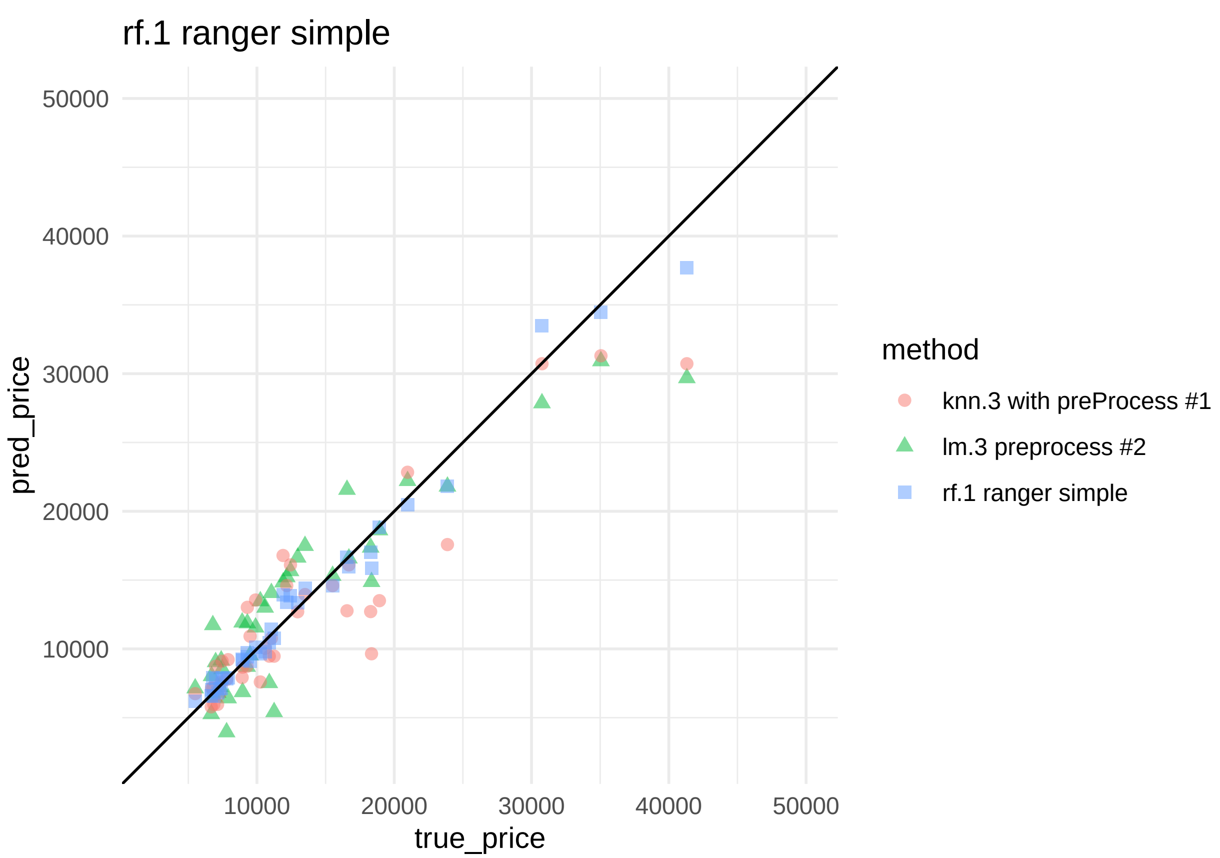
### 12.1 rf.1 simple random forest

Just ranger out-of-the-box:

---

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# # ranger simple (rf.1) -----------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  # Method
     method = 'rf.1 ranger simple'

  # Train
     start <- as.numeric(Sys.time())
     set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
     train_ranger.rf.1 <- train(price ~ .,
                                  method = "ranger",
                                  data = am.pp.dum,
                                  verbose=T)
     end <- as.numeric(Sys.time())

  # Predict
     pred.rf.1 <- predict(train_ranger.rf.1, newdata = X)

  # Measure
     RMSE.rf.1 <- RMSE(Y, pred.rf.1)
     rmse <- RMSE.rf.1
     runtime = ceiling( end - start )

  # Store results
     comment = 'no parameters'

     results.details.temp <- data.frame(method = method,
                                         true_price = Y, pred_price = pred.rf.1)
     results.details <- bind_rows(results.details, results.details.temp)
```

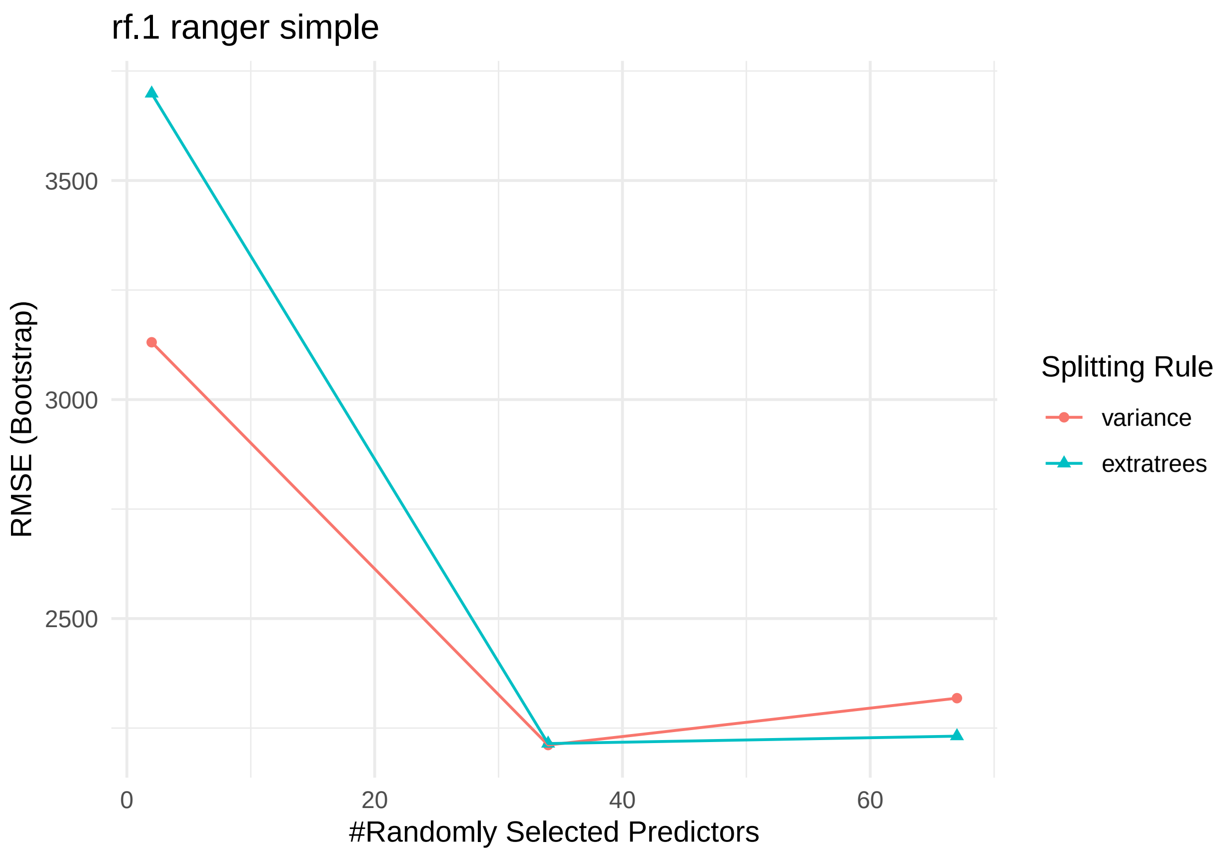
```r
    results.temp <- data.frame(method = method, RMSE = rmse,
                               runtime = runtime, comment = comment)
    results <- bind_rows (results, results.temp)

    slctn.results <- c(slctn.results, method)

  # Review results
    results.details %>% filter(method %in% all_of(slctn.results)) %>%
      ggplot(aes(true_price, pred_price, col=method, shape=method)) +
      geom_point(size=2, alpha=0.5) +
      geom_abline(slope=1, intercept = 0) +
      ylim(min.price*0.5,max.price*1.1) +
      xlim(min.price*0.5,max.price*1.1) +
      ggtitle(method) + theme_minimal()
```




```r
    train_ranger.rf.1
## Random Forest
##
## 200 samples
##  67 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
```

```
##
##   mtry  splitrule   RMSE      Rsquared   MAE
##    2    variance    3130.729  0.9130891  2105.777
##    2    extratrees  3698.689  0.8858498  2477.667
##   34    variance    2211.196  0.9285766  1496.799
##   34    extratrees  2214.662  0.9306199  1477.054
##   67    variance    2318.257  0.9205618  1575.639
##   67    extratrees  2231.522  0.9287549  1487.560
##
## Tuning parameter 'min.node.size' was held constant at a value of 5
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 34, splitrule = variance
##  and min.node.size = 5.
```

```
ggplot(train_ranger.rf.1) +
  ggtitle(method) + theme_minimal()
```




```
kable(results)
```

| method | RMSE | runtime | comment |
|---|---|---|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |
| knn.2 tuneGrid & trControl | 4143.973 | 2 | knn.1 + tune k=1:10 & 10x repeated 10fold-CV |

| method | RMSE | runtime | comment |
|---|---|---|---|
| knn.3 with preProcess #1 | 3283.149 | 29 | knn.2 + zv/nzv/corr/center/scale |
| rf.1 ranger simple | 1112.354 | 19 | no parameters |

---

Random Forest outperforms lm and knn by far and is faster as knn.

### 12.2 rf.2 with tuneGrid

The default setting took mtry 2,34 & 67 (67 is the number of our variables). We do not know about the values in between.

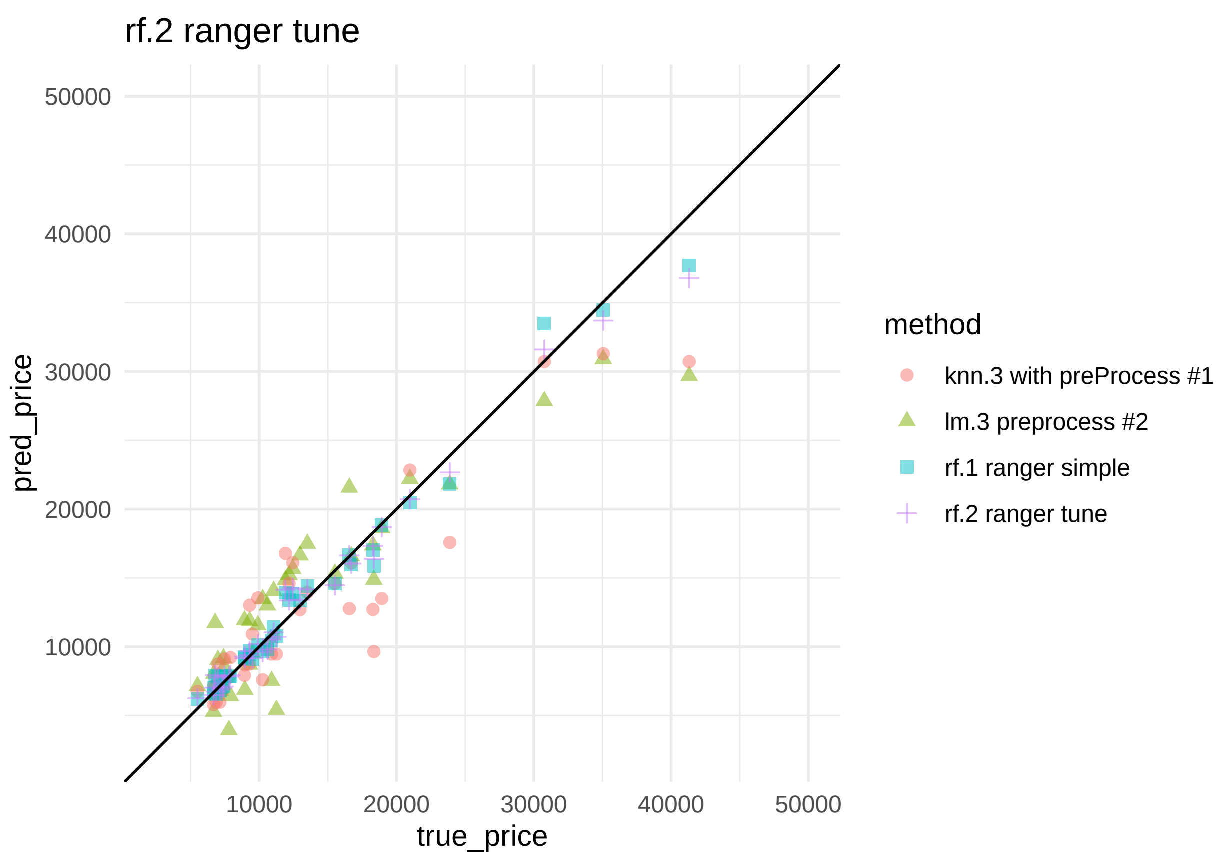
min.node.size was fixed at 5.

With tuneGrid we zoom into mtry (in steps of 11) and extend the range of min.node.size. We proceed only with splitrule variance.

---

```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ranger with tunegrid (rf.2) ---------------------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  # Method
    method = 'rf.2 ranger tune'

    tune.ranger <- data.frame(expand.grid(mtry = seq(1,67,11) ,
                                          min.node.size = seq(1,7,1),
                                          splitrule = 'variance'))

  # Train
    start <- as.numeric(Sys.time())
    set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
    train_ranger.rf.2 <- train(price ~ .,
                               method = "ranger",
                               data = am.pp.dum,

                               tuneGrid = tune.ranger )
    end <- as.numeric(Sys.time())

  # Predict
  pred.rf.2 <- predict(train_ranger.rf.2, newdata = X)

  # Measure
  RMSE.rf.2 <- RMSE(Y, pred.rf.2)
  rmse <- RMSE.rf.2
  runtime = ceiling( end - start )

  # Store results
  comment = 'tuned: min.node.size, mtry'

  results.details.temp <- data.frame(method = method, true_price = Y,
```

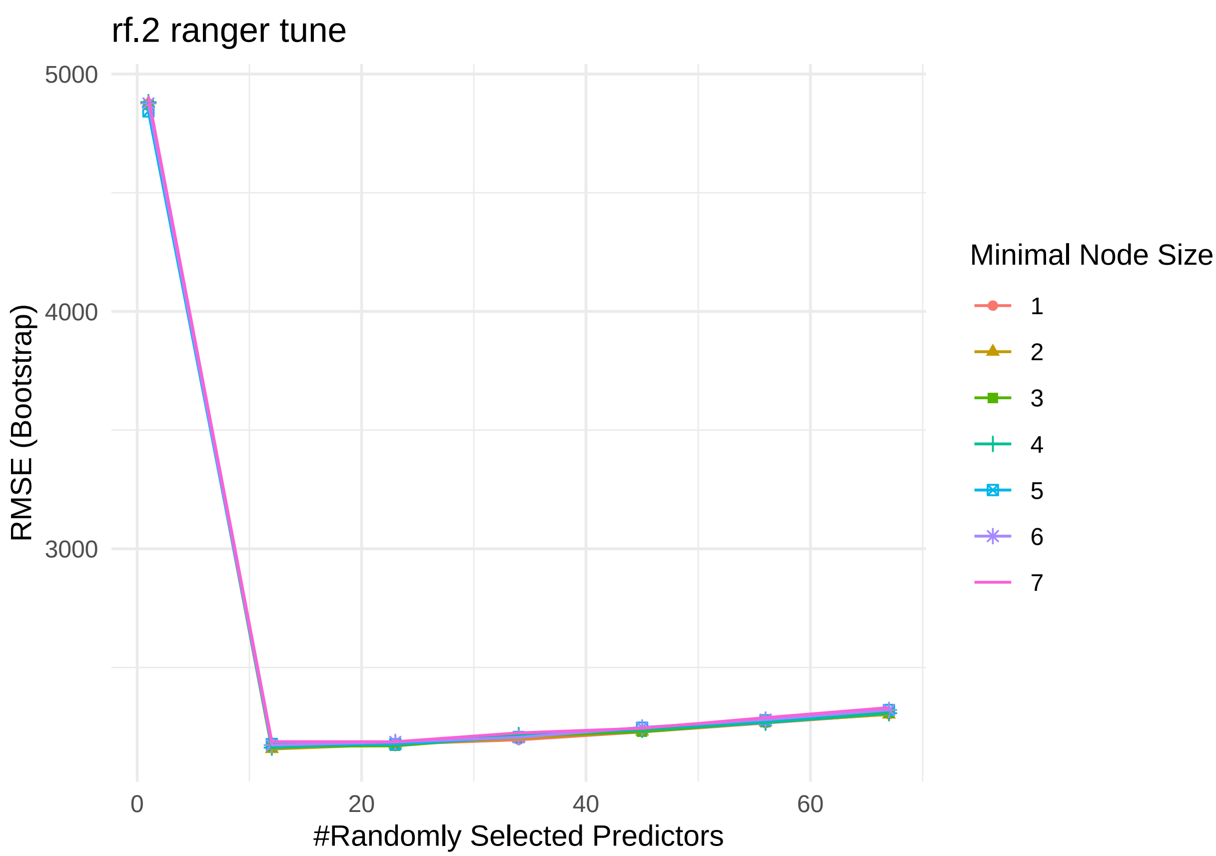
```
                                        pred_price = pred.rf.2)

results.details <- bind_rows(results.details, results.details.temp)

results.temp <- data.frame(method = method, RMSE = rmse,
                           runtime = runtime, comment = comment)
results <- bind_rows (results, results.temp)

slctn.results <- c(slctn.results, method)

# Review results
results.details %>% filter(method %in% all_of(slctn.results)) %>%
  ggplot(aes(true_price, pred_price, col=method, shape=method)) +
  geom_point(size=2, alpha=0.5) +
  geom_abline(slope=1, intercept = 0) +
  ylim(min.price*0.5,max.price*1.1) +
  xlim(min.price*0.5,max.price*1.1) +
  ggtitle(method) + theme_minimal()
```




```
train_ranger.rf.2
## Random Forest
##
## 200 samples
##  67 predictor
##
```

```
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  RMSE      Rsquared   MAE
##    1    1              4869.919  0.8921066  3452.246
##    1    2              4877.114  0.8946393  3459.415
##    1    3              4869.481  0.8919029  3458.042
##    1    4              4881.747  0.8921711  3464.261
##    1    5              4842.637  0.8925348  3437.417
##    1    6              4876.422  0.8919388  3458.820
##    1    7              4904.652  0.8920981  3477.043
##   12    1              2169.349  0.9336457  1453.734
##   12    2              2156.510  0.9347032  1447.130
##   12    3              2168.125  0.9336087  1454.212
##   12    4              2162.387  0.9342576  1450.469
##   12    5              2177.587  0.9333145  1459.154
##   12    6              2173.105  0.9333165  1461.181
##   12    7              2187.800  0.9323638  1471.198
##   23    1              2177.267  0.9318300  1464.060
##   23    2              2176.436  0.9314768  1465.161
##   23    3              2169.794  0.9317875  1465.755
##   23    4              2181.513  0.9310796  1468.699
##   23    5              2176.767  0.9313785  1465.214
##   23    6              2186.576  0.9309091  1475.022
##   23    7              2187.233  0.9306851  1472.782
##   34    1              2195.365  0.9296405  1483.279
##   34    2              2201.469  0.9291848  1485.751
##   34    3              2211.047  0.9289686  1492.283
##   34    4              2214.916  0.9285024  1499.253
##   34    5              2207.156  0.9287424  1491.332
##   34    6              2205.584  0.9288010  1494.262
##   34    7              2224.683  0.9276996  1505.071
##   45    1              2228.650  0.9271482  1508.559
##   45    2              2230.469  0.9267759  1507.470
##   45    3              2229.093  0.9272650  1509.556
##   45    4              2237.616  0.9264346  1513.966
##   45    5              2246.274  0.9261160  1522.766
##   45    6              2247.354  0.9256475  1523.323
##   45    7              2243.859  0.9259318  1522.573
##   56    1              2265.958  0.9243425  1539.153
##   56    2              2269.559  0.9241756  1541.339
##   56    3              2270.521  0.9240446  1542.341
##   56    4              2267.713  0.9242212  1538.959
##   56    5              2277.522  0.9235921  1548.006
##   56    6              2280.613  0.9234021  1549.450
##   56    7              2288.702  0.9226672  1556.435
##   67    1              2303.671  0.9215835  1562.514
##   67    2              2302.070  0.9218251  1564.090
##   67    3              2309.848  0.9210706  1567.739
##   67    4              2307.414  0.9212411  1568.756
##   67    5              2320.843  0.9205554  1575.473
```

```
##   67    6             2321.119  0.9204584  1580.600
##   67    7             2331.108  0.9195356  1587.868
##
## Tuning parameter 'splitrule' was held constant at a value of variance
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 12, splitrule = variance
##  and min.node.size = 2.

  ggplot(train_ranger.rf.2) +
    ggtitle(method) + theme_minimal()
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
## Warning: Removed 7 rows containing missing values (geom_point).
```




```
  kable(results)
```

| method | RMSE | runtime | comment |
|---|---|---|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |
| knn.2 tuneGrid & trControl | 4143.973 | 2 | knn.1 + tune k=1:10 & 10x repeated 10fold-CV |
| knn.3 with preProcess #1 | 3283.149 | 29 | knn.2 + zv/nzv/corr/center/scale |
| rf.1 ranger simple | 1112.354 | 19 | no parameters |

| method | RMSE | runtime | comment |
|---|---|---|---|
| rf.2 ranger tune | 1089.639 | 123 | tuned: min.node.size, mtry |

---

We could improve the RMSE further but nearly tripled the runtime.

We skip rf.1 in results graph and keep rf.2.

---

```
# Do not keep method in results graph
  slctn.results <- slctn.results[slctn.results  !='rf.1 ranger simple']
```

---

## 12.3 rf.3 with tuneGrid & preProcess

The graph of rf.2 has a turning point at 12 and then seems too increases continuously.

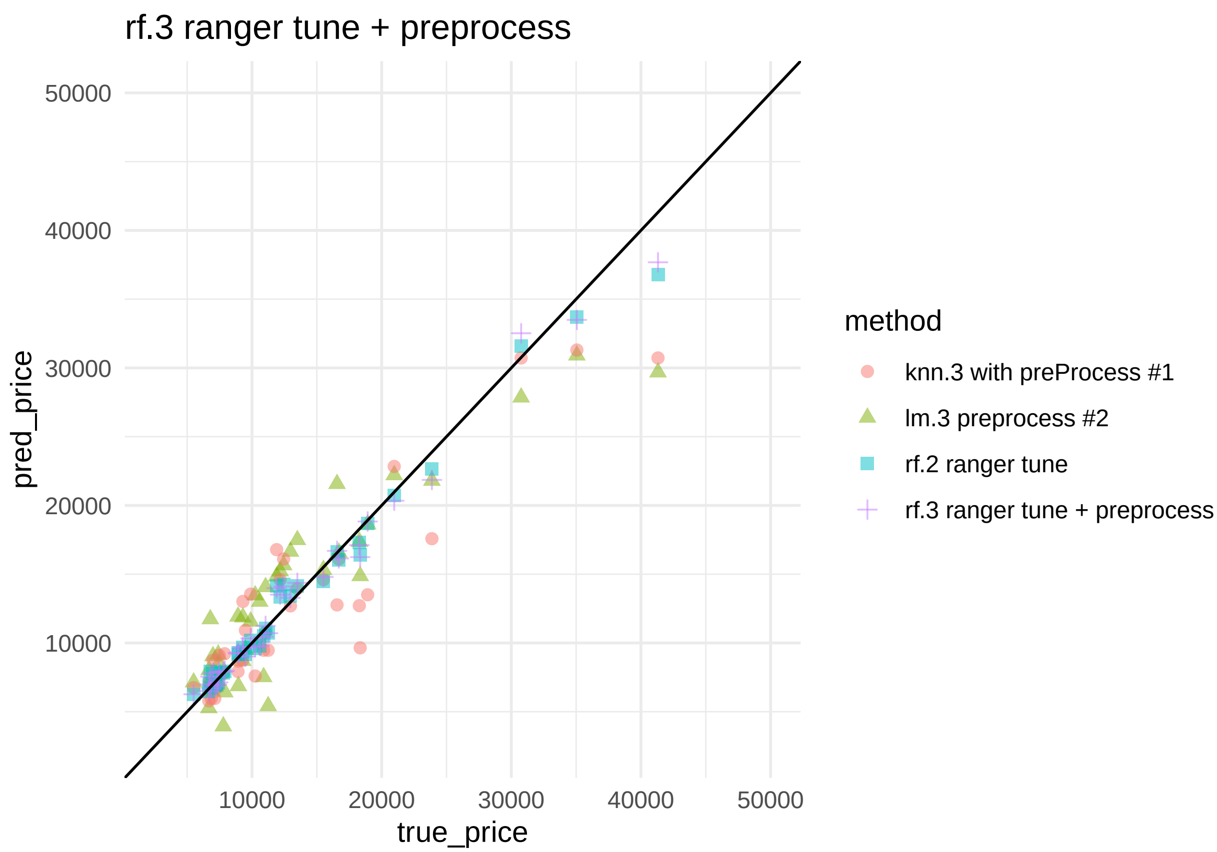
We try to zoom in between 6 and 24.

The RMSE difference in the min.node.size's is very low, thus we set min.node.size=3 to avoid overfitting.

Additionally we add the same pre-proccess steps as with lm.2 / knn.3 in order to improve even more.

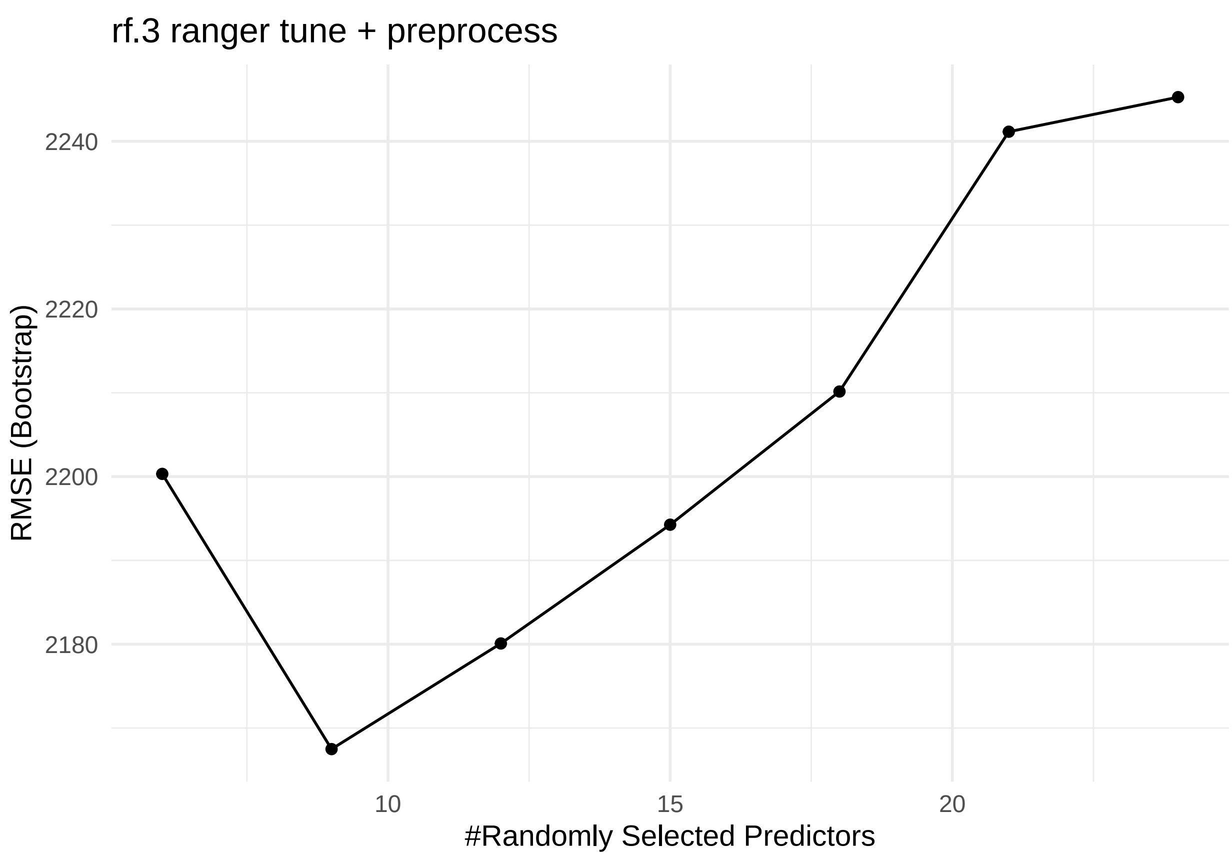
```
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# ranger with tunegrid & preprocess (rf.3)-------------------------------
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# Method
    method = 'rf.3 ranger tune + preprocess'

    tune.ranger <- data.frame(expand.grid(mtry = seq(6,24,3) ,
                                        min.node.size = 3,
                                        splitrule = 'variance' ))
  # Train
    start <- as.numeric(Sys.time())
    set.seed(803, sample.kind = 'Rounding')
## Warning in set.seed(803, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
    train_ranger.rf.3 <- train(price ~ .,
                          method = "ranger",
                          data = am.pp.dum,
                          tuneGrid = tune.ranger,
                          preProcess = c('zv','nzv','corr'
                                        ,'center', 'scale'))
    end <- as.numeric(Sys.time())

  # Predict
    pred.rf.3 <- predict(train_ranger.rf.3, newdata = X)

  # Measure
    RMSE.rf.3 <- RMSE(Y, pred.rf.3)
    rmse <- RMSE.rf.3
    runtime = ceiling( end - start )

  # Store results
    comment = 'tuned as rf.2 + zv/nzv/corr/center/scale'

  results.details.temp <- data.frame(method = method,
                                     true_price = Y, pred_price = pred.rf.3)
  results.details <- bind_rows(results.details, results.details.temp)

  results.temp <- data.frame(method = method, RMSE = rmse,
                             runtime = runtime, comment = comment)
  results <- bind_rows (results, results.temp)

  slctn.results <- c(slctn.results, method)

  # Review results
  results.details %>% filter(method %in% all_of(slctn.results)) %>%
```

```
    ggplot(aes(true_price, pred_price, col=method, shape=method)) +
    geom_point(size=2, alpha=0.5) +
    geom_abline(slope=1, intercept = 0) +
    ylim(min.price*0.5,max.price*1.1) +
    xlim(min.price*0.5,max.price*1.1) +
    ggtitle(method) + theme_minimal()
```




```
  train_ranger.rf.3
## Random Forest
##
## 200 samples
##  67 predictor
##
## Pre-processing: centered (36), scaled (36), remove (31)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    6    2200.323  0.9323347  1480.586
##    9    2167.488  0.9327388  1465.878
##   12    2180.091  0.9314460  1474.174
##   15    2194.260  0.9297609  1487.236
##   18    2210.150  0.9283591  1495.735
##   21    2241.147  0.9261189  1516.260
```

```
##   24    2245.273  0.9258592  1523.131
##
## Tuning parameter 'splitrule' was held constant at a value of variance
##
## Tuning parameter 'min.node.size' was held constant at a value of 3
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mtry = 9, splitrule = variance
##  and min.node.size = 3.

  ggplot(train_ranger.rf.3) +
    ggtitle(method) + theme_minimal()
```




```
  kable(results)
```

| method | RMSE | runtime | comment |
|---|---:|---:|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |
| knn.2 tuneGrid & trControl | 4143.973 | 2 | knn.1 + tune k=1:10 & 10x repeated 10fold-CV |
| knn.3 with preProcess #1 | 3283.149 | 29 | knn.2 + zv/nzv/corr/center/scale |
| rf.1 ranger simple | 1112.354 | 19 | no parameters |
| rf.2 ranger tune | 1089.639 | 123 | tuned: min.node.size, mtry |
| rf.3 ranger tune + preprocess | 1066.999 | 19 | tuned as rf.2 + zv/nzv/corr/center/scale |

Further improvement of the RMSE makes rf.3 the best method in this project.

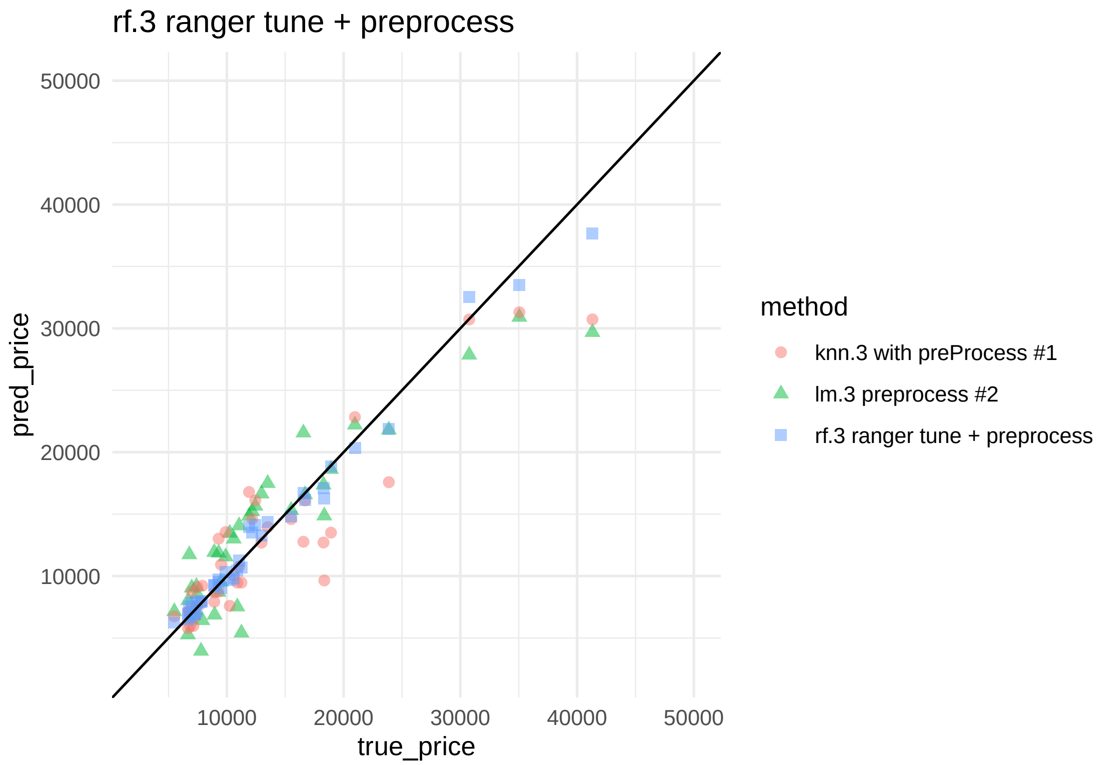
We skip rf.1 in results graph and keep rf.2.

---

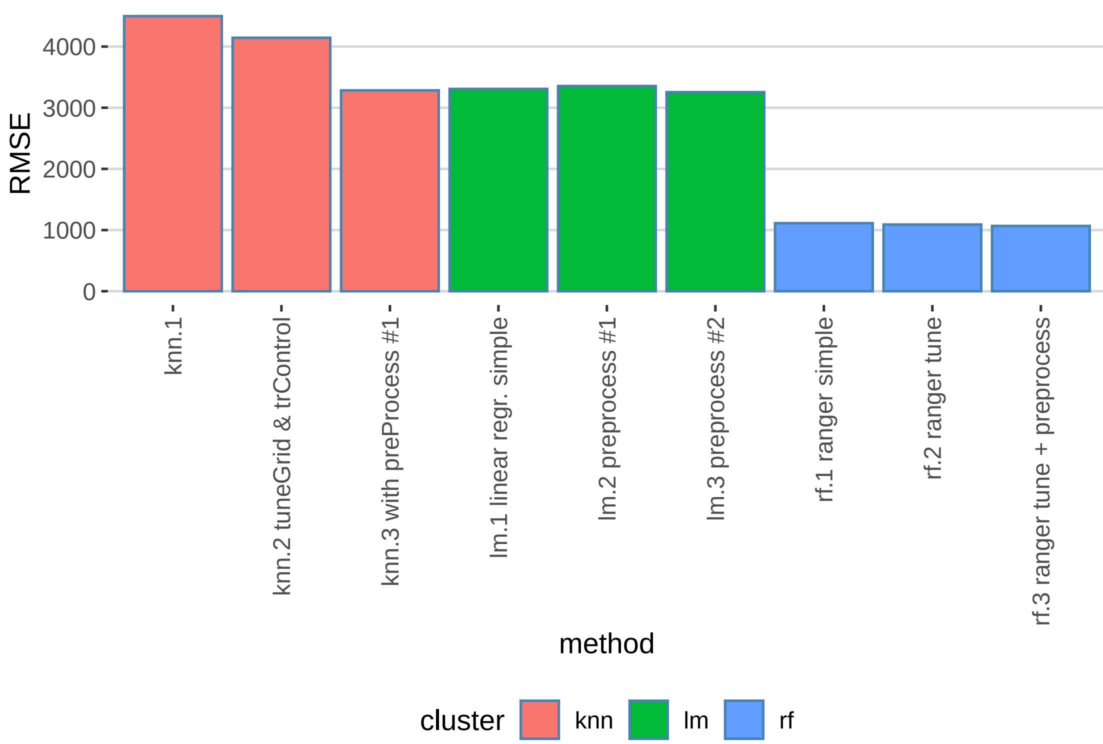
```
# Do not keep method in results graph
  slctn.results <- slctn.results[slctn.results  !='rf.2 ranger tune']
```

---

# 13 Results



rf.3 ranger tune + preprocess
50000
40000
30000
20000
10000
pred_price
10000
20000
30000
40000
50000
true_price
method
knn.3 with preProcess #1
lm.3 preprocess #2
rf.3 ranger tune + preprocess

4000
3000
2000
1000
0
RMSE
knn.1
knn.2 tuneGrid & trControl
knn.3 with preProcess #1
lm.1 linear regr. simple
lm.2 preprocess #1
lm.3 preprocess #2
rf.1 ranger simple
rf.2 ranger tune
rf.3 ranger tune + preprocess
method
cluster
knn
lm
rf

| method | RMSE | runtime | comment |
|---|---:|---:|---|
| lm.1 linear regr. simple | 3307.258 | 1 | no train parameters ; warnings |
| lm.2 preprocess #1 | 3354.609 | 2 | preProcess zv/nzv/corr/center/scale; warnings |
| lm.3 preprocess #2 | 3254.774 | 2 | all of lm.2 + pca ; no warnings |
| knn.1 | 4497.960 | 1 | knn.1 no tuning |
| knn.2 tuneGrid & trControl | 4143.973 | 2 | knn.1 + tune k=1:10 & 10x repeated 10fold-CV |
| knn.3 with preProcess #1 | 3283.149 | 29 | knn.2 + zv/nzv/corr/center/scale |
| rf.1 ranger simple | 1112.354 | 19 | no parameters |
| rf.2 ranger tune | 1089.639 | 123 | tuned: min.node.size, mtry |
| rf.3 ranger tune + preprocess | 1066.999 | 19 | tuned as rf.2 + zv/nzv/corr/center/scale |

---

From comparing the methods, rf is the clear winner with knn.3 as the lowest RMSE. knn and lm play in another league.

In the comparison scatterplot contains only small outliers for rf.3.

The effects of the pre-processing and tuning depend on the method. The most effect of pre-prossesing and tuning is with knn (at least in the project).

Impressing is the out of the box performance of random forest.

The winner in runtime is linear regression.

# 14 Summary

After we familiarized with the data and did first pre-processing, we could get a good insight via the data exploration and visualization.

We introduced and used two ways of pre-processing in caret:

- directly to the data with the preProcess function
- within the train as preProcess parameter

There would me much more option for pre-processing, thus we got here only an appetizer.

The same with the tuning parameters. The project used only those, tuneable with caret.

By applying pre-processing and tuning step-wise to the methods, we got an impression of the impact. One learning take-away is, to consider the functions applied in the context with the method (especially with linear regression).

In terms of method, this project of course covered only three out of many. Even more options exist with packages, as methods can be applied with several packages.

Thus in overall the project could only scratch on the surface of machine learning methods but could not deep dive into a method or package neither cover as wide area.

For me as machine learning newbie, this was a great learning experience and an eye-opener of the variety of the R opportunities.