Report

Casey Grasdal

1/19/2019

1. Executive summary

For my own project I've decided to work with publicly available dataset available on Kaggle platform - basic computer stats *Computers.csv*, can be downloaded via url https://www.kaggle.com/kingburrito666/basic-computer-data-set (Please, ensure a proper storing of the dataset in the working folder in order to be able to execute the code as its designed).

The dataset comprises a number of numeric and categorical features describing alternative setups for computers and related price tag. I found this data to be interesting for machine learning challenge such as to build a predictive model for price based on computer features.

The data has 6259 observations and 10 different variables: *price*, *speed*, *hd*, *ram*, *screen*, *ads* and *trend* are numeric variables, *cd*, *multi* and *premium* are categorical variables. The dataset is clean and ready to be utilized for Machine Learning challenge. There was basic preprocessing performed to encode categorical data. The data was divided to train and test dataset.

The modeling stage includes a cross-validation of a number of regression models on Train set: Linear Regression, Support Vector Machine, Generalized Boosted Regression (stochastic gradient boosting) and Random Forest. Although Random Forest demonstrated the longest CPU time performance, it delivers best performing metrics: RMSE and R squared.

I have concluded to proceed with Random Forest. Based on 10-Fold Cross-Validation with tuned hyperparameters on Train set, the best final RF model was chosen. Final model I trained was further used to make price predictions on test set.

Please, note that both the R-code and Rmd execution time may take up to approximately 10-15 minutes depending on one's computer processing power.

2. Methodology

2.1. Data exploration and visualization

The data set was loaded from https://www.kaggle.com/kingburrito666/basic-computer-data-set and saved to the working folder as *Computers.csv* (289.64 KB). The data was well-suited for machine learning challenge with minimal preprocessing requirements. I needed to remove column X as it's a redundant index column and could not be used as a predictor later at work.

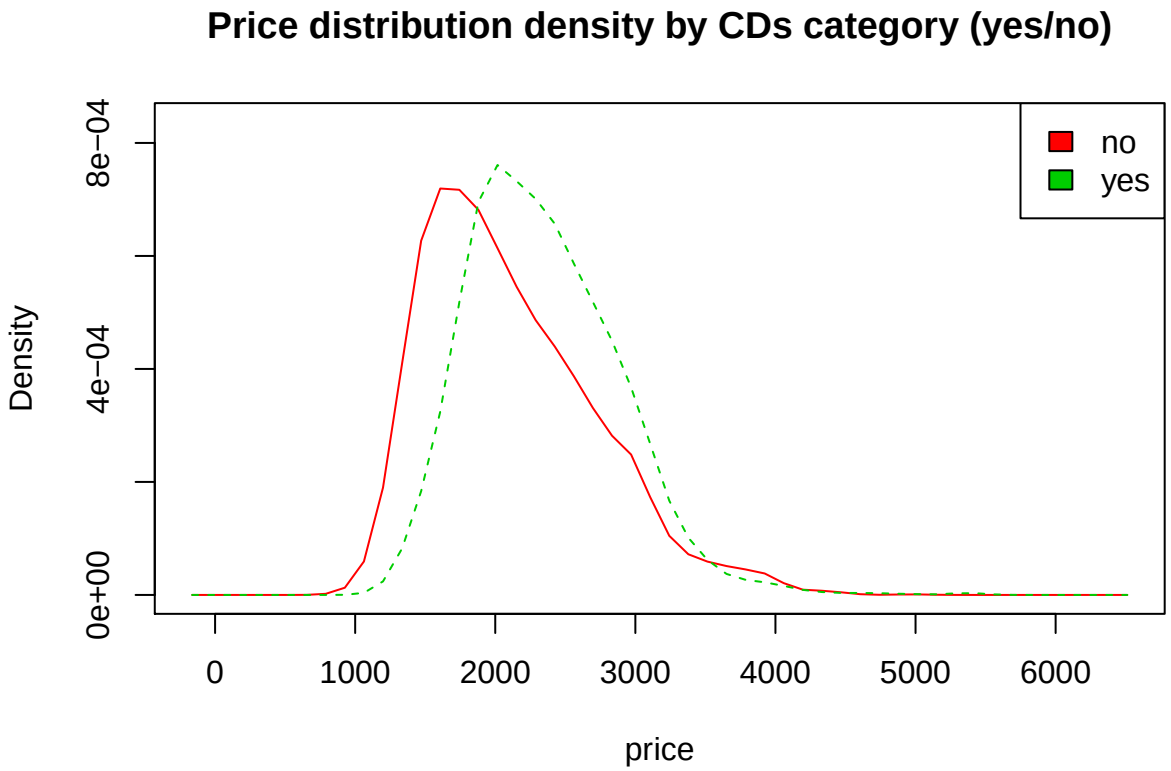
The data has 2 groups of variables:

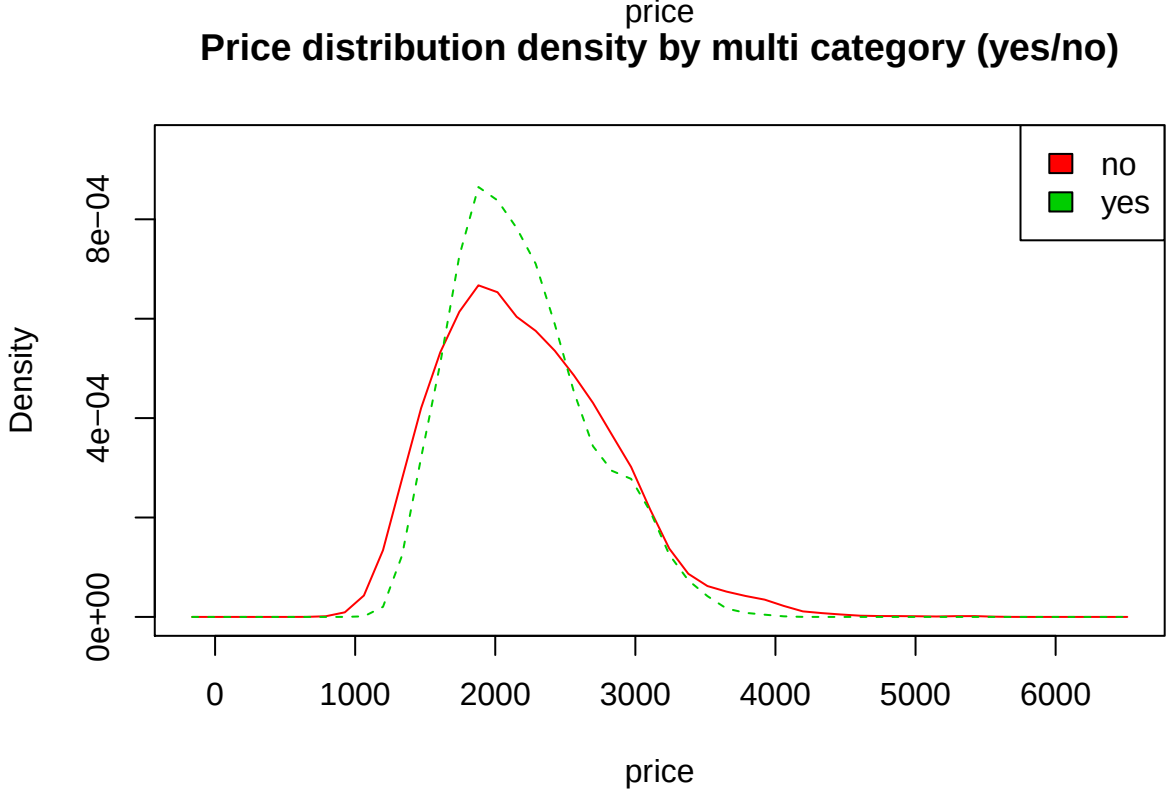
1. *price*, *speed*, *hd*, *ram*, *screen*, *ads* and *trend* are numeric features
2. *cd*, *multi* and *premium* are categorical features ('yes' or 'no')

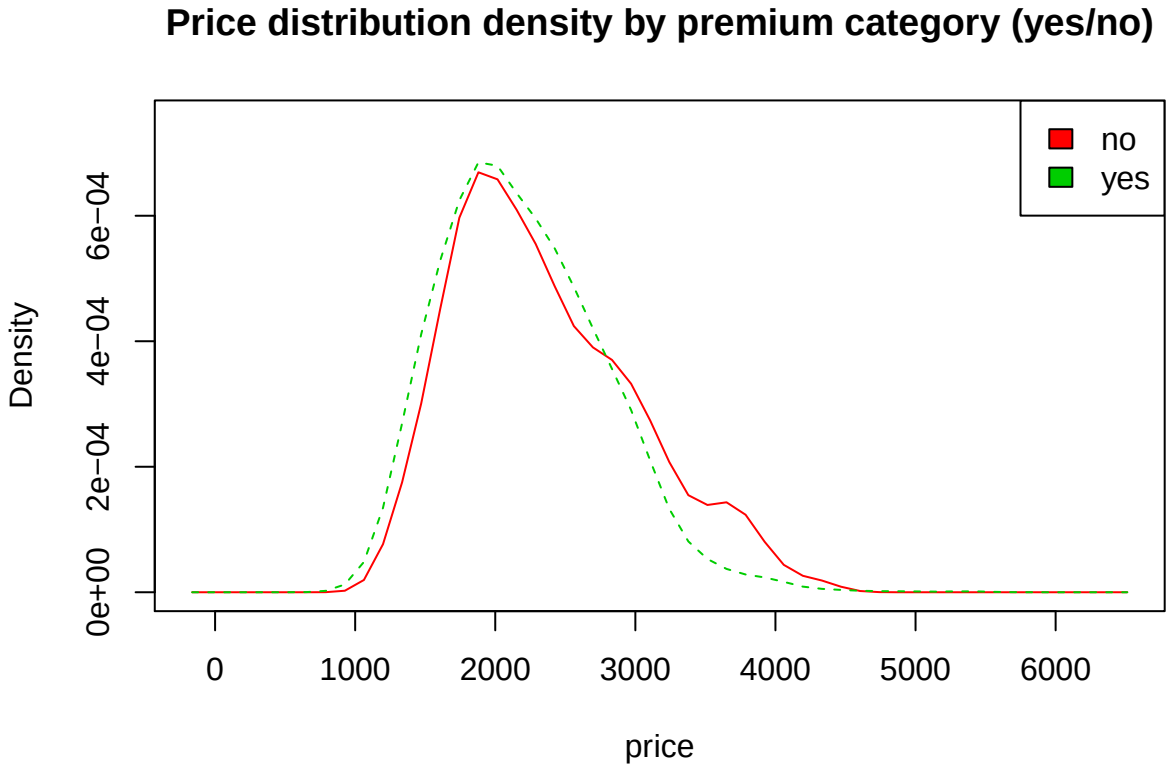
price is dependent variable, while the rest of variables are independent and can be used as predictors for our future model.

First of all, I have decided to look at kernel density plots for categorical variables. Kernel density plots show price distribution based on categorical variables grouping ('yes' or 'no'). *sm.density.compare* allows me to superimpose the kernal density plots of price distribution for two different groups of each catergorical variable. Analysis of kernel density plots shows that in case of grouping variable *cd*, the price distribution shifts to the

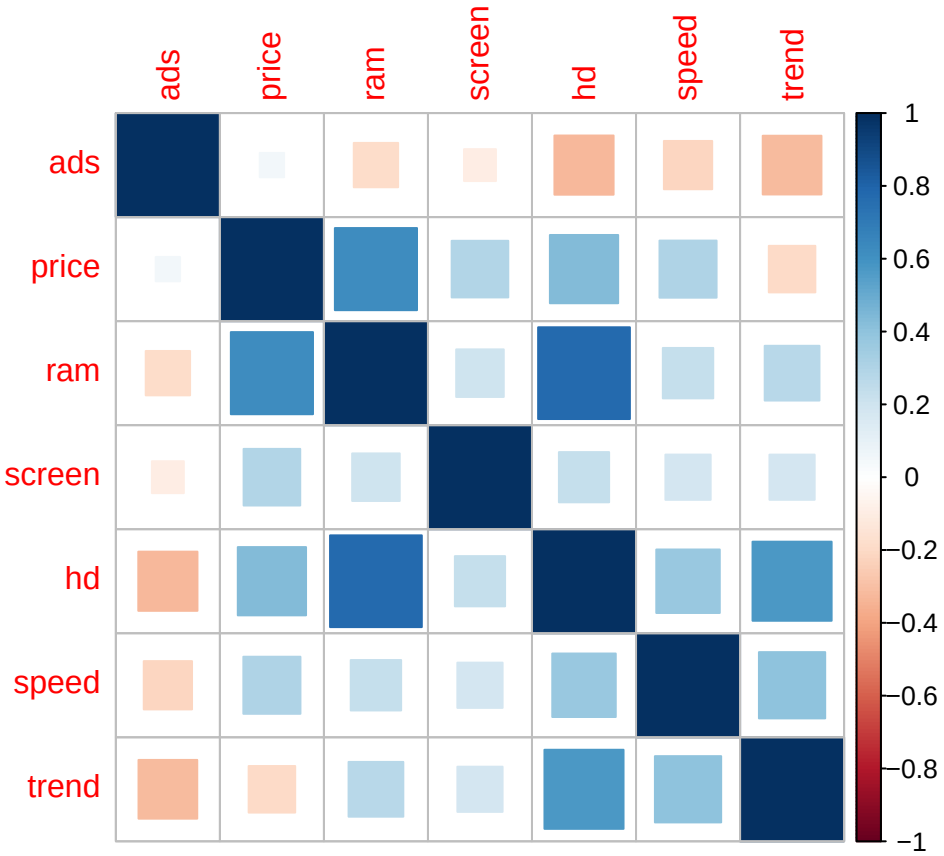
right if computer has a CD ('yes'). This implies that there is a positive correlation between *cd* and *price* - straightforward conclusiong from visualization (plots are below).







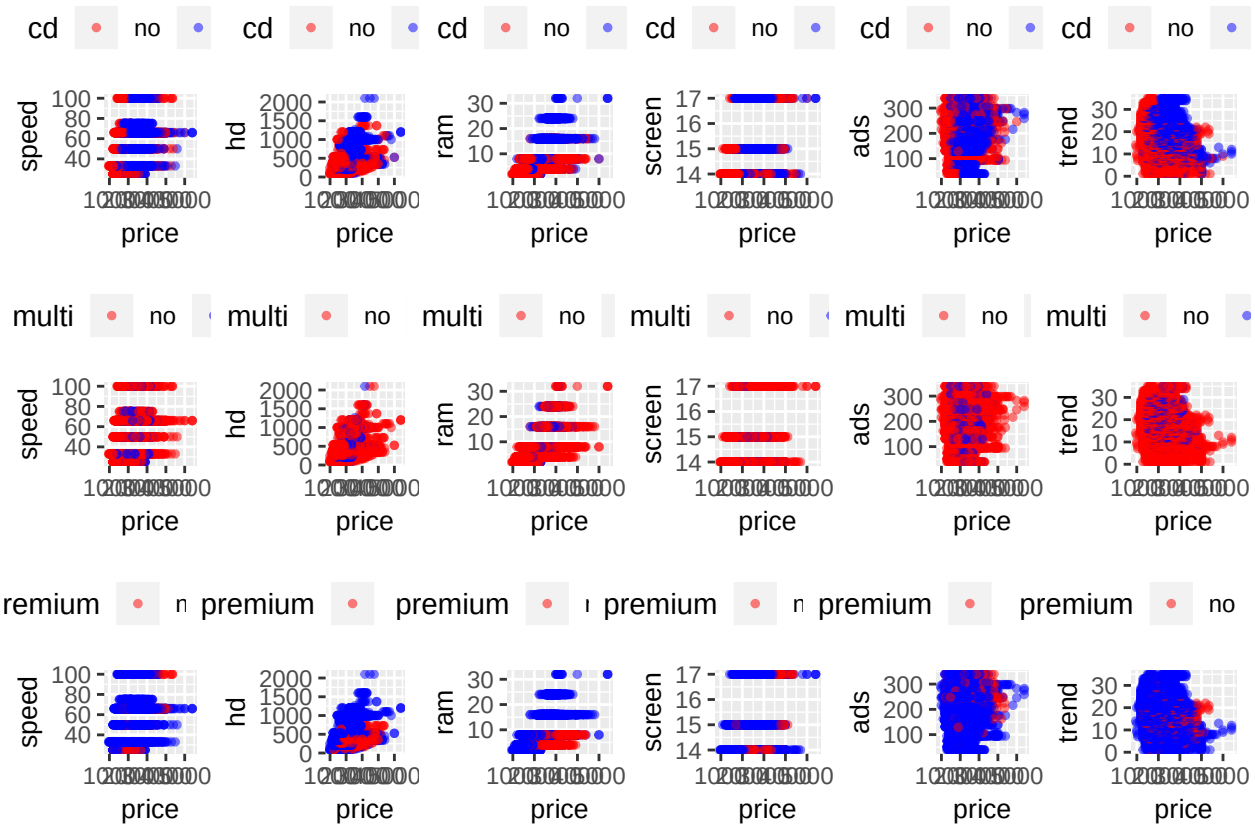
There is a set of numeric features that interesting to get insight into. For the next step, I need *corrplot* and *corrgram* packages to generate a correlation matrix that shows correlation coefficients between numeric variables. Correlation matrix shows that *price* is positively correlated with *ram*, *hd*, *screen* and *speed* and negatively with *trend*:



So far, I tried to understand how variables are related to dependent variable *price* by separating them into 2 groups: numeric and categorical. By appropriating some insights on relations, I've decided to look at more complex visualization, which comprises all variables.

The following picture contains 18 subplots. We can see that this visualization supports insights from previous drawings. For example, almost all numeric variables positively correlated with price except *trend*, and that *cd* variable is more often takes 'yes' option with increasing *price* variable.

Please, note, PDF won't be generated with proper display of 18 subplots thus I strongly recommend to run R-code file to be able to zoom in and inspect the visualization:



2.2. Data preprocessing.

The dataset doesn't have any missing values. The only preprocessing to perform is to prepare categorical variables for further calculations. One way to take care of categorical variables is to introduce dummy variables. First, I usea *dummy.code()* from 'psych'-package. But later I've decided to give up on this approach as to avoid loading additional packages to the memory. Instead, to keep it simple by encoding the categorical variables with familiar to us factor()-approach:

```
df$cd = factor(df$cd,
                  levels = c('yes', 'no'),
                  labels = c(1, 0))

df$multi = factor(df$multi,
                  levels = c('yes', 'no'),
                  labels = c(1, 0))

df$premium = factor(df$premium,
                  levels = c('yes', 'no'),
                  labels = c(1, 0))

head(df)
```

```
##   price speed  hd ram screen cd multi premium ads trend
## 1  1499    25  80   4     14  0     0       1  94     1
## 2  1795    33  85   2     14  0     0       1  94     1
## 3  1595    25 170   4     15  0     0       1  94     1
## 4  1849    25 170   8     14  0     0       0  94     1
```

```
## 5  3295     33 340  16      14  0      0        1  94      1
## 6  3695     66 340  16      14  0      0        1  94      1
```

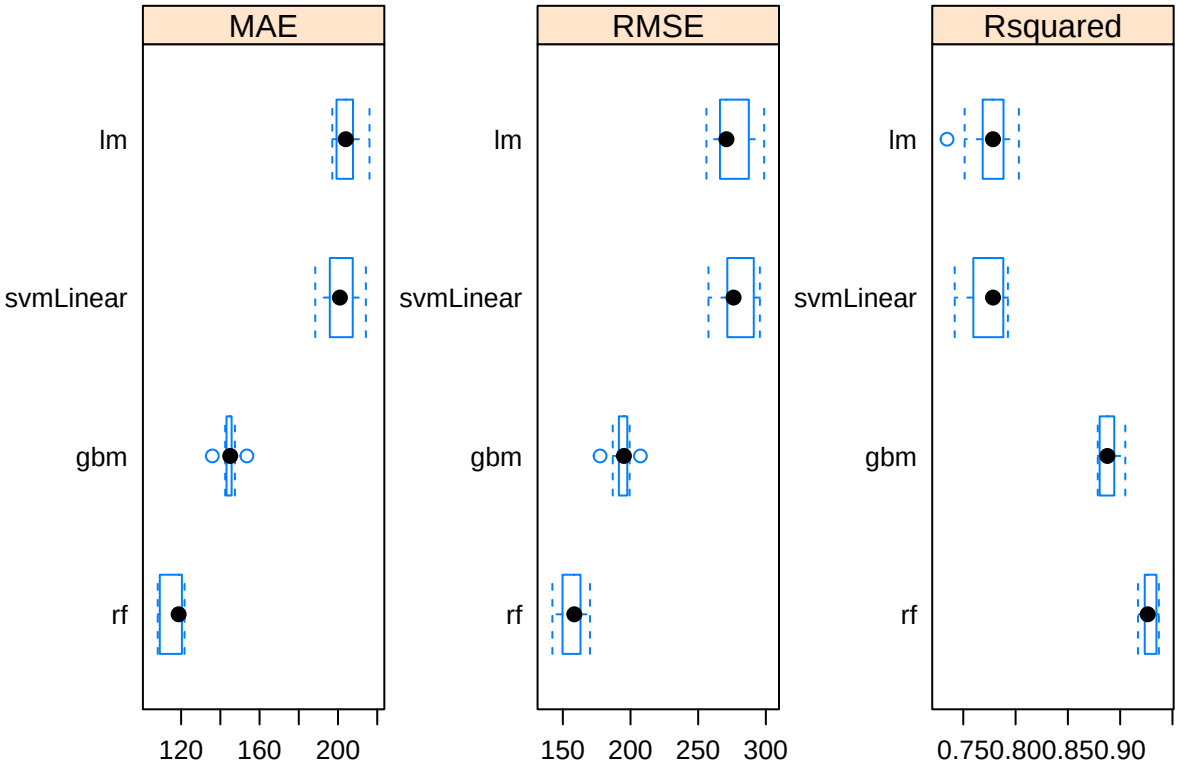
2.3. Modelling

At this stage I had to consider following facts:

- The dependent variable (the one I want to predict) *price* is continuous, which makes it a regression problem with mix of continuous and categorical predictors (the latter were encoded).
- Cross-validation should be performed to assess well-known algorithms which are well-suited for regression problems and choose one based on time and performance metrics. For regression class of problems the most appropriate metric is Root Mean Squared Error (RMSE). R squared is another statistical measure demonstrating how close data to regression line, ranging from 0 to 1 (0 - 100%), where 1 stands for the best fit and means that the model explains all the variability of the response data around its mean. Time is another factor I want to look at in order to understand how much CPU time is required for one or another model and how critical it can be.

The data is split to train and test dataset. I also created *test_final* set by excluding a *price* column to be able to create a full set of predictions after analysis. *test* set keeps original price values and will be used to compare predictions with original price.

The models I consider are Linear Regression (lm), Support Vector Machine (svmLinear), Generalized Boosted Regression (gbm) and Random Forest (rf). I further display box-and-whisker plot for cross-validation results based on RMSE and R squared:




```
##
## Call:
## summary.resamples(object = results)
##
## Models: lm, svmLinear, gbm, rf
```

```
## Number of resamples: 10
##
## MAE
##                Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## lm        197.0834 199.6385 203.9303 204.5520 207.5051 216.0589    0
## svmLinear 188.3963 195.9789 200.9483 201.5528 207.4702 214.2014    0
## gbm       135.9144 143.5066 145.0368 144.8401 145.6831 153.5143    0
## rf        108.0602 111.3608 118.7372 116.4971 120.2519 121.7380    0
##
## RMSE
##                Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## lm        256.0907 266.0989 270.9125 275.3783 287.3740 298.7666    0
## svmLinear 257.6001 271.9859 276.1457 278.4662 290.6613 295.5956    0
## gbm       177.4908 192.1213 195.1349 194.0454 197.2045 207.2849    0
## rf        142.2640 151.5605 158.4623 157.0580 162.5224 170.0282    0
##
## Rsquared
##                 Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lm        0.7344831 0.7693366 0.7784195 0.7752649 0.7869713 0.8031792    0
## svmLinear 0.7418033 0.7601388 0.7784672 0.7729110 0.7882348 0.7926621    0
## gbm       0.8785753 0.8820520 0.8877804 0.8899010 0.8942427 0.9049006    0
## rf        0.9170517 0.9237515 0.9263049 0.9277883 0.9337173 0.9369708    0
```

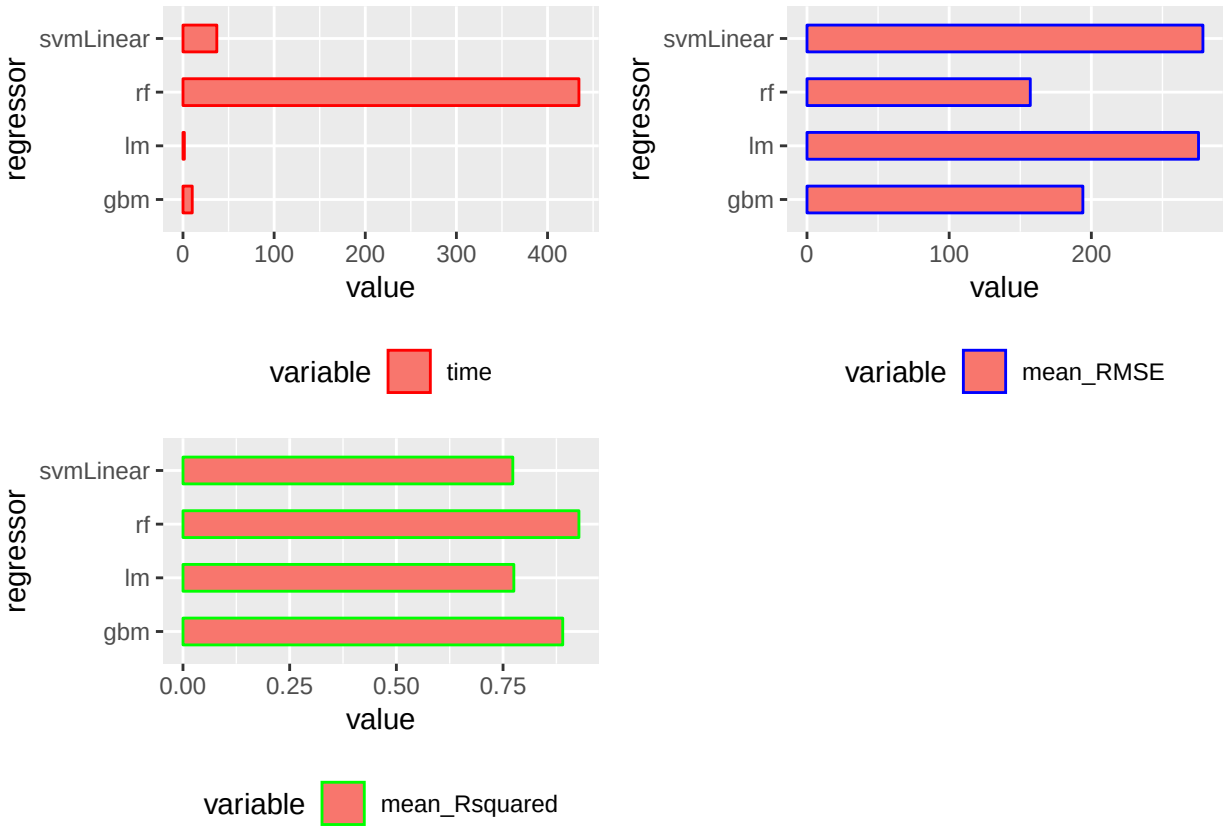
I also visualize model key performing metrics incorporating all 3 key metrics I've described above: time, RMSE and R squared. We can see that Random Forest requires significantly longer time to perform with mean RMSE comparably smaller and mean R squared being proportionally higher. Generalized Boosted Regression also demonstrates good performance, with significantly shorter time, small mean RMSE and high mean R squared. Both models, Generalized Boosted Regression and Random Forest, worth to be considered.

```
rmse_time
```

```
##   regressor       time mean_RMSE mean_Rsquared
## 1        lm   1.504998  275.3783     0.7752649
## 2 svmLinear  37.118185  278.4662     0.7729110
## 3       gbm  10.151868  194.0454     0.8899010
## 4        rf 434.425775  157.0580     0.9277883
```

The following plot is the results of cross-validation for our chosen regressors:

```
## Using regressor as id variables
```



Because GBM model showed promissing results performing as the second best algorithm for the problem, I have decided to go ahead and try to tune it to see if metrics can improve and exceed those demonstrated by Random Forest. However, time for model tuning went above 23 minutes and I've made a decision not to pursue with this path. The chunk of code for GBM tuning is excluded from my submitted R-code, but I decided to include it into my current report so my peers can take a look in case of interest:

```
control <- trainControl(method="cv", number=10, repeats=3)

tunegrid <- expand.grid(n.trees = seq(500, 1000, 100),      # step = 100
                        shrinkage = seq(.1, .5, .1),        # step = 0.1
                        interaction.depth = seq(1, 9, 1),   # step = 1
                        n.minobsinnode=seq(1, 5, 1))        # step = 1

gbm_tune<-train(price ~., data = train,
                method='gbm', tuneGrid = tunegrid,
                trControl = control, preProcess=c('center', 'scale'))

print(gbm_tune)
plot(gbm_tune)
```

Thus, my choice is a Random Forest model that delivered best performance metrics while CPU execution time was not critical (it took 6 minutes on my machine, I do expect the time should be in the range (5:10) minutes for most of my peer reviewers based on their computational power).

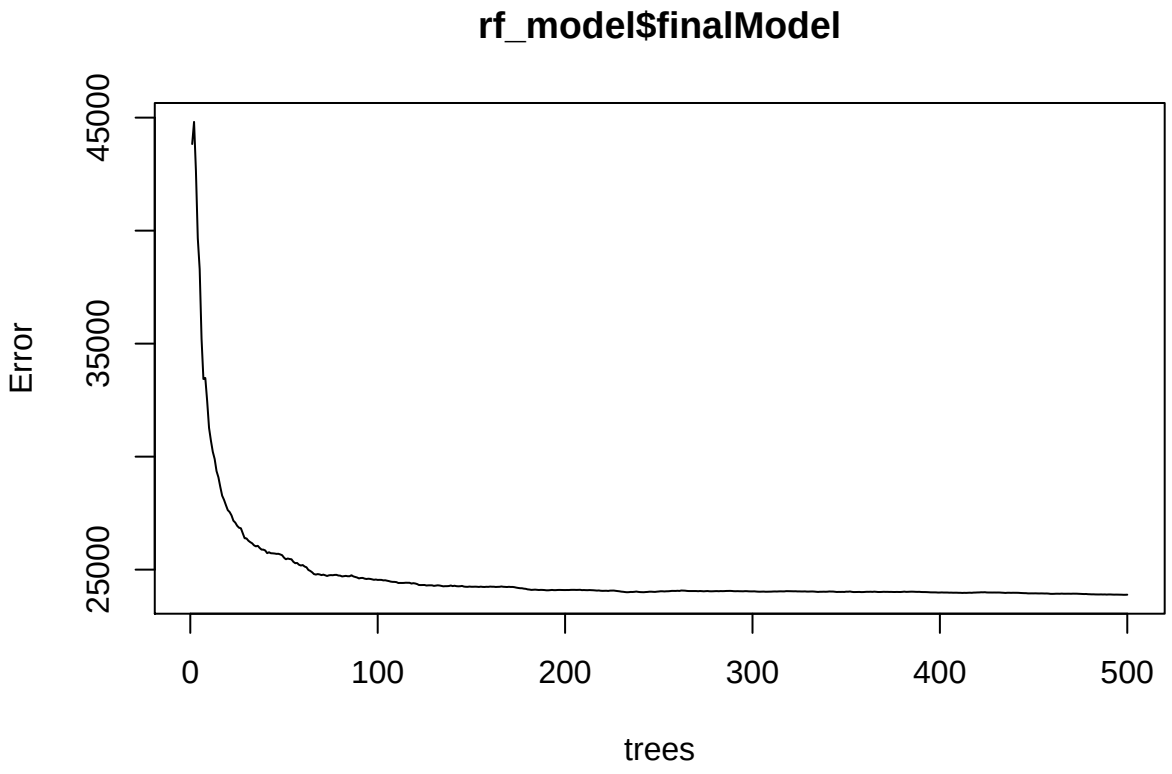
2.4. Random Forest - Final model

To train a Random Forest model I use *train()* function. By default, the train function without any arguments re-runs the model over 25 bootstrap samples and across 3 options of the tuning parameter. *trainControl*

function allows to specify a number of parameters (including sampling parameters) in my model. There are 2 tuning parameters for RF model, *mtry* - the number of randomly selected predictors at each cut in the tree and *ntree* - the number of trees. By running the following code, I obtained the best RF model which minimizes RMSE, and can check its optimal parameters: mtry is equaled to 5, and number of trees is 500.

```
regressor <- 'rf'
control <- trainControl(method="cv", number=10)
set.seed(7)
rf_model <- train(price ~ . , data = train, method = regressor, trControl = control, preProcess=c('cente
rf_model
```

```
## Random Forest
##
## 5631 samples
##    9 predictor
##
## Pre-processing: centered (9), scaled (9)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5067, 5068, 5068, 5068, 5068, 5068, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     195.9324  0.9036757  146.4181
##   5     157.8930  0.9268294  116.0544
##   9     159.0006  0.9252814  116.0065
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 5.
```



3. Results

I follow with producing predicted values of price for the test set (to be precise, *test_final* with removed price tags)

```
rf_results <- predict(rf_model, test_final)
```

I, then, add predicted values to the dataframe and rearrange columns to its original order following with re-factoring categorical variables to its original format.

So, at this final stage of my modelling exercise, I obtained 2 test data sets: one with original price tags and another one is with predicted. I am curious to see how well my predictions were made on the test set and if the error is approximately the same as for the train set. And I further see that RMSE on the test set for RF model is quite close to RMSE on the train set (153 vs 157). My model performed on the test set as expected.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
set.seed(7)
rf_rmse <- RMSE(test$price, test_final$price)
rf_rmse
```

```
## [1] 153.6056
```

4. Concluding Remarks

For the second project of the Capstone course I've chosen the publicly available dataset from Kaggle platform describing basic computer statistics and related price (access via https://www.kaggle.com/kingburrito666/basic-computer-data-set). It's a well-suited dataset ready to be used for ML challenge, for example, for such supervised problem as predicting the price for subset of computer stat records. It's a regression problem with continuous dependent variable. I utilized cross-validation to identify best performing algorithm among 4 alternatives - Linear Regression, Support Vector Machine, Generalized Boosted Regression and Random Forest, by judging based on set of metrics such as time, RMSE and R squared. By comparing different algorithms, I've chosen Random Forest and used caret package for resampling and tuning in order to obtain best final RF model. I proceeded with predictions made on the test set and compared my results to original price tags.

The final RF model had performed well demonstrating a minimal RMSE among alternative algorithms (Linear Regression, SVM and GBM), such as 157 on the train set (note, the range of *price* variable is (949:5399)). RMSE is one of the most common metrics used to measure accuracy for continuous variables. Moreover, RF showed the best R squared measure (0.92), which tells us about goodness of fit of a regression model. Although RF takes comparatively longer CPU performance time, it's not critical (about 6 minuts on 2.6 GHz Intel Core i5 and 8 Gb RAM).

As a result, I obtained the final dataset with predicted price for each combination of computer stats from the test set. RF model demonstrated a good performance for the assigned regression problem with multiple predictors.

4.1. Possible Future work

As a possible extention to this work the list of alternative algorithms may be extended by including less popular regressors. Alternatively, it is possible to take more time or utilize more powerful machine to run tuning algorithm for Generalized Boosted Regression Model (GBM) in order to understand if GBM may outperform RF. For interested parties, the challenge can be to replicate the code in Python with use of scikit-learn library.