Predicting quantity of sales given a price

Floriano Peixoto

The main objective of our model will be predict the quantity that will be sold, given a determined price to a product.

Frist of all let´s load the data we have.

```
sales <- read.csv2("sales.csv", sep=",", stringsAsFactors = T)
prices <- read.csv2("comp_prices.csv", sep=",", stringsAsFactors = F)
```

As we can see the data we want is divided between the both sets, the quantity is in sales and the prices is in comp_prices, so we need to find a way to merge the two sets.

```
names(sales)
```

```
## [1] "PROD_ID"    "DATE_ORDER" "QTY_ORDER"  "REVENUE"
```

```
names(prices)
```

```
## [1] "PROD_ID"          "DATE_EXTRACTION"  "COMPETITOR"
## [4] "COMPETITOR_PRICE" "PAY_TYPE"
```

Between the two sets we can find PROD_ID and a date field, we can use them to merge it to a unique set, but the problem is that we have a lot of prices listed to the same date, two for each competitor. So we can transform this field into derivated fields, let´s try to use min, max, mean, median and standard deviation. The data field could be transformed as well, so let´s create a field for each attribute.

```
prices_processed <- filter(prices, complete.cases(prices)) %>%
                    mutate(YEAR = lubridate::year(as_datetime(DATE_EXTRACTION)),
                           MONTH = lubridate::month(as_datetime(DATE_EXTRACTION)),
                           DAY = lubridate::day(as_datetime(DATE_EXTRACTION)),
                           COMPETITOR_PRICE = as.numeric(COMPETITOR_PRICE),
                           PROD_ID = as.factor(PROD_ID)) %>%
                    group_by(PROD_ID, YEAR, MONTH, DAY) %>%
                    summarise(MIN_PRICE = min(COMPETITOR_PRICE),
                              MAX_PRICE = max(COMPETITOR_PRICE),
                              MEAN_PRICE = mean(COMPETITOR_PRICE),
                              SD_PRICE = sd(COMPETITOR_PRICE),
                              MEDIAN_PRICE = median(COMPETITOR_PRICE))



prices_processed
```

```
## Source: local data frame [1,929 x 9]
## Groups: PROD_ID, YEAR, MONTH [?]
##
##    PROD_ID  YEAR MONTH   DAY MIN_PRICE MAX_PRICE MEAN_PRICE SD_PRICE
##     <fctr> <dbl> <dbl> <int>     <dbl>     <dbl>      <dbl>    <dbl>
## 1       P1  2015     3    15   1499.00      1499   1499.000  0.00000
## 2       P1  2015     3    16   1362.50      1499   1464.205 58.47609
## 3       P1  2015     3    17   1362.50      1499   1429.793 65.61760
## 4       P1  2015     3    18   1362.50      1499   1441.008 64.24923
## 5       P1  2015     3    19   1304.13      1499   1407.363 80.50857
## 6       P1  2015     3    20   1304.13      1499   1439.727 74.83624
## 7       P1  2015     3    21   1359.00      1499   1439.842 65.80861
```

```
## 8       P1  2015     3    22   1424.05      1499   1488.293 27.21708
## 9       P1  2015     3    23   1403.90      1499   1456.487 46.08149
## 10      P1  2015     3    24   1403.90      1499   1454.808 46.68677
## # ... with 1,919 more rows, and 1 more variables: MEDIAN_PRICE <dbl>
```

Now let´s process the sales data too.

```
sales_processed <- filter(sales, complete.cases(sales)) %>%
                    mutate(YEAR = lubridate::year(as_datetime(DATE_ORDER)),
                           MONTH = lubridate::month(as_datetime(DATE_ORDER)),
                           DAY = lubridate::day(as_datetime(DATE_ORDER)),
                           QTY_ORDER = as.numeric(QTY_ORDER),
                           PROD_ID = as.factor(PROD_ID)) %>%
                    group_by(PROD_ID, YEAR, MONTH, DAY) %>%
                    summarise(QTY_ORDER = sum(QTY_ORDER))

sales_processed
```

```
## Source: local data frame [2,162 x 5]
## Groups: PROD_ID, YEAR, MONTH [?]
##
##    PROD_ID  YEAR MONTH   DAY QTY_ORDER
##     <fctr> <dbl> <dbl> <int>     <dbl>
## 1       P1  2015     2     4        10
## 2       P1  2015     2     5        12
## 3       P1  2015     2     6        21
## 4       P1  2015     2     7         4
## 5       P1  2015     2     8         7
## 6       P1  2015     2     9         5
## 7       P1  2015     2    10        10
## 8       P1  2015     2    11        11
## 9       P1  2015     2    12        16
## 10      P1  2015     2    13         7
## # ... with 2,152 more rows
```

Now we can merge the two data sets.

```
merged_data <- merge(sales_processed, prices_processed)

tail(merged_data)
```

```
##      PROD_ID YEAR MONTH DAY QTY_ORDER MIN_PRICE MAX_PRICE MEAN_PRICE
## 1913      P9 2015     9   4        44    429.90       579   454.3022
## 1914      P9 2015     9   5        42    411.48       579   453.7212
## 1915      P9 2015     9   6        39    411.48       579   452.2540
## 1916      P9 2015     9   7       118    411.48       579   452.6156
## 1917      P9 2015     9   8        75    411.48       579   452.3818
## 1918      P9 2015     9   9        55    411.48       579   451.4950
##      SD_PRICE MEDIAN_PRICE
## 1913 45.10212       429.90
## 1914 47.13315       429.90
## 1915 44.00008       431.52
## 1916 46.43189       429.90
## 1917 48.13763       429.90
## 1918 44.58876       433.14
```

Now we´ll divide the data into two sets, the training set and the test set, we use this to ensure a trusty model validated by results we previously know. We´ll use 80% of the data to the training set and the rest to the test set.

```
merged_data.hex <- as.h2o(merged_data,destination_frame = "merged_data.hex")
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
data_to_model <- h2o.splitFrame(data = merged_data.hex ,
                                ratios = 0.80,
                                seed=1234)

train <- data_to_model[[1]]
test <- data_to_model[[2]]
```

Now we´ll try to find a good linear regression model that can predicts our price. As our target, we´ll use the MEDIAN_PRICE as it´s a real value of price present in our data.

For this work we´ll use (definitions from: https://github.com/h2oai/h2o-training-book/blob/master/hands-on_training/regression.md):

- Generalized Linear Models (GLM): Average an ensemble of weakly predicting (small) trees where each tree "adjusts" to the "mistakes" of the preceding trees.
- Gradient (Tree) Boosting Machines (GBM): Average an ensemble of weakly predicting (small) trees where each tree "adjusts" to the "mistakes" of the preceding trees.
- Random Forests: Average an ensemble of weakly predicting (larger) trees where each tree is de-correlated from all other trees.

```
result <- c()

model <- c("GBM", "GLM" , "RF")

fields <- setdiff(setdiff(names(train), "QTY_ORDER"), "MEDIAN_PRICE")


GBM <- h2o.gbm(x = fields, build_tree_one_node = T,
            y = "QTY_ORDER",
            training_frame = train,
            validation_frame = test,
            seed=1234)
```

```
##
  |
  |                                                                 |   0%
  |
  |==================================                               |  52%
  |
  |=================================================================| 100%
```

```
result[[1]] <-  h2o.r2(GBM, valid = TRUE)


GLM <- h2o.glm(x = fields,
              y = "QTY_ORDER",
              training_frame = train,
```

```r
              validation_frame = test,
              family = "poisson",
              seed=1234)
```

```
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   2%
  |
  |=================================================================| 100%
```

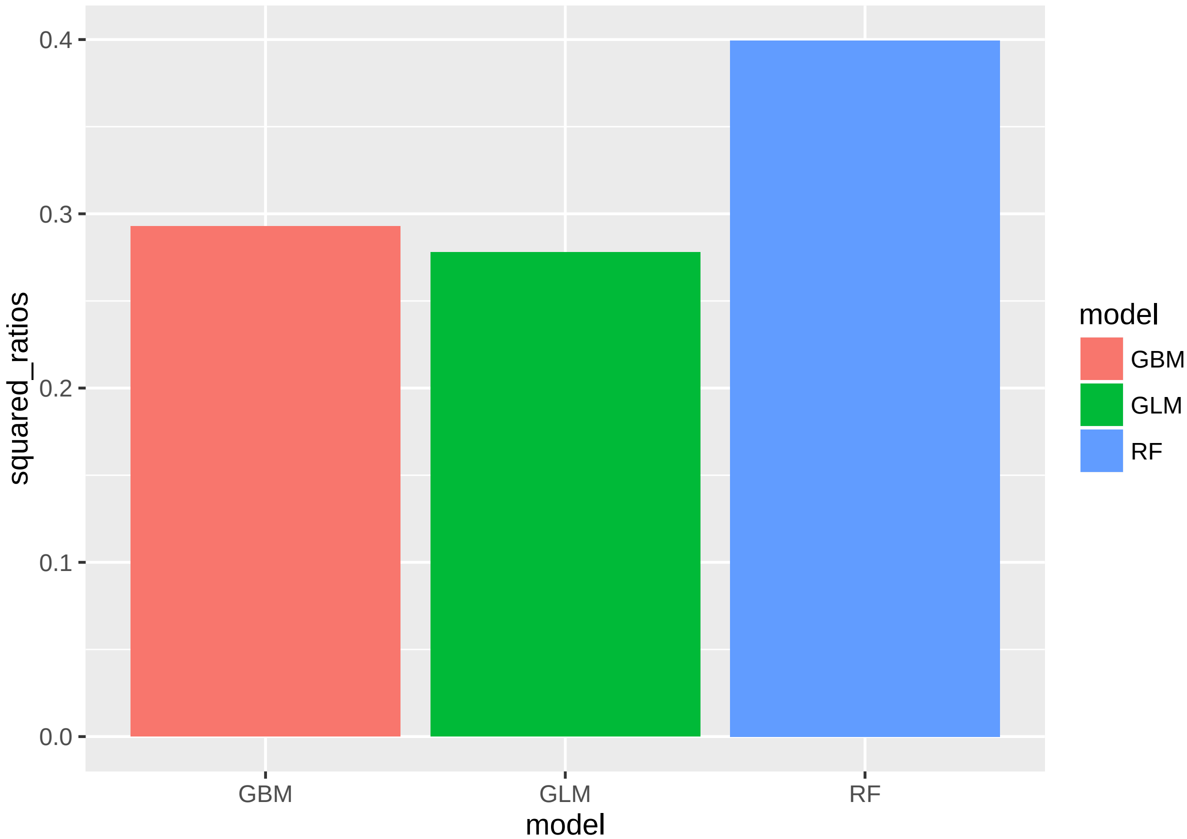
```r
result[[2]] <-  h2o.r2(GLM, valid = TRUE)

RF <- h2o.randomForest(x = fields,
                       y = "QTY_ORDER",
                       training_frame = train,
                       validation_frame = test,
                       seed=1234)
```

```
##
  |
  |                                                                 |   0%
  |
  |=====================================================            |  82%
  |
  |=================================================================| 100%
```

```r
result[[3]] <-  h2o.r2(RF, valid = T)


results <- data.frame(model,result)


names(results) <- c("model", "squared_ratios")

results
```

```
##   model squared_ratios
## 1   GBM      0.2927767
## 2   GLM      0.2779196
## 3    RF      0.3995000
```

```r
ggplot(data=results, aes(x = model, y = squared_ratios, fill = model)) +
 geom_bar(stat="identity", position = "dodge")
```



As we can see the random forest had the best score above all others, although the rate was not too impressive, so let´s use it and see what more this model has to tell us.

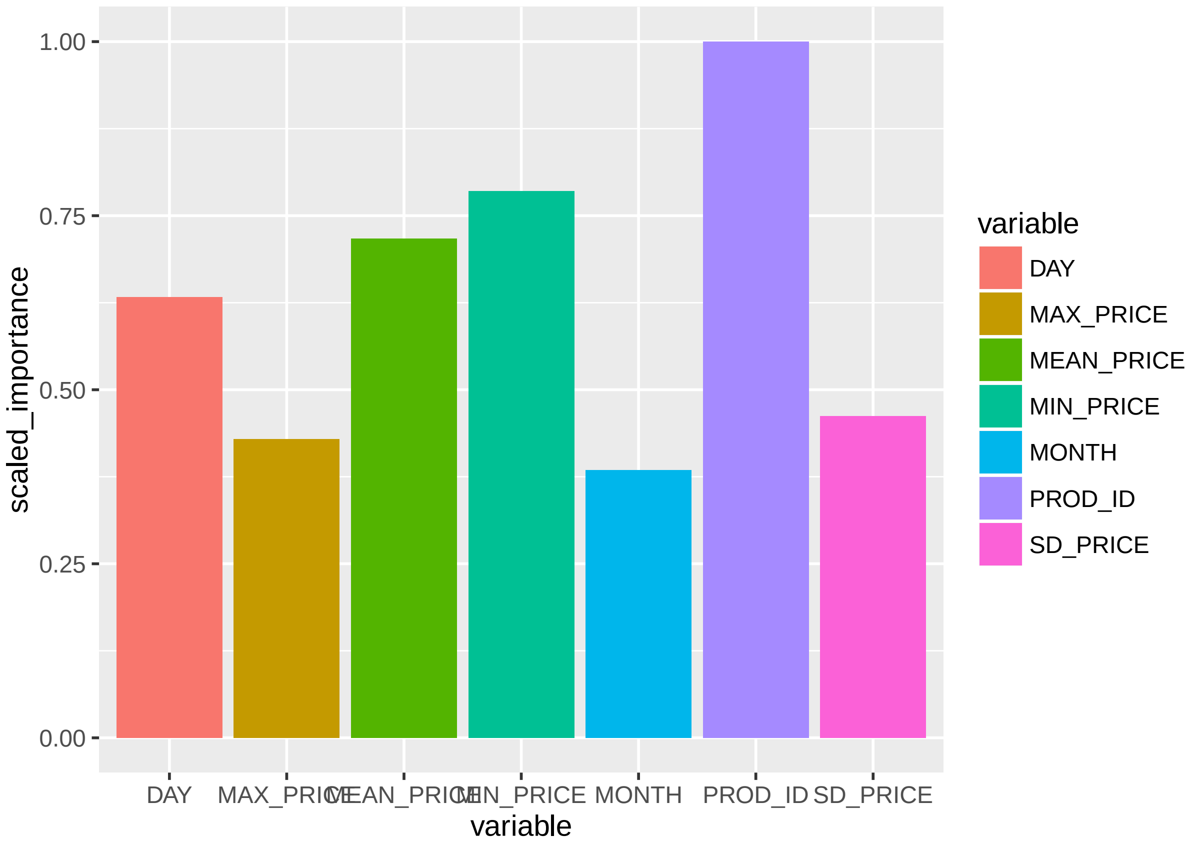
```
RF_summary <- summary(RF)
```

```
## Model Details:
## ==============
##
## H2ORegressionModel: drf
## Model Key:  DRF_model_R_1477301123940_3
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1              50                       50              431188        19
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        20   19.94000        294        849   681.30000
##
## H2ORegressionMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## MSE:  339102.6
## RMSE:  582.3252
## MAE:  226.5663
## RMSLE:  1.166771
## Mean Residual Deviance :  339102.6
##
##
```

```
## H2ORegressionMetrics: drf
## ** Reported on validation data. **
##
## MSE:  228748
## RMSE:  478.2761
## MAE:  195.3498
## RMSLE:  1.130537
## Mean Residual Deviance :  228748
##
##
##
##
## Scoring History:
##             timestamp   duration number_of_trees training_rmse
## 1 2016-10-24 07:25:50  0.033 sec               0
## 2 2016-10-24 07:25:50  0.154 sec               1     804.38679
## 3 2016-10-24 07:25:50  0.190 sec               2     751.94285
## 4 2016-10-24 07:25:50  0.220 sec               3     678.41282
## 5 2016-10-24 07:25:50  0.241 sec               4     662.39115
##   training_mae training_deviance validation_rmse validation_mae
## 1
## 2    291.80519      647038.10250       624.14024      223.87314
## 3    273.17998      565418.05247       510.94162      186.17343
## 4    251.93005      460243.94775       493.22184      185.16384
## 5    245.43263      438762.03954       486.20123      183.31704
##   validation_deviance
## 1
## 2        389551.03933
## 3        261061.34282
## 4        243267.78820
## 5        236391.63733
##
## ---
##              timestamp   duration number_of_trees training_rmse
## 46 2016-10-24 07:25:51  1.289 sec              45     581.01261
## 47 2016-10-24 07:25:51  1.318 sec              46     581.08096
## 48 2016-10-24 07:25:51  1.353 sec              47     583.00305
## 49 2016-10-24 07:25:51  1.395 sec              48     583.14751
## 50 2016-10-24 07:25:52  1.439 sec              49     584.46314
## 51 2016-10-24 07:25:52  1.470 sec              50     582.32520
##    training_mae training_deviance validation_rmse validation_mae
## 46    227.48140      337575.65440       479.94194      195.79442
## 47    227.30703      337655.07915       479.19784      195.26834
## 48    227.26138      339892.56157       478.46123      195.47823
## 49    226.88415      340061.01890       479.27009      195.00290
## 50    227.37359      341597.16733       477.78017      194.88105
## 51    226.56632      339102.64393       478.27606      195.34978
##    validation_deviance
## 46        230344.26112
## 47        229630.56557
## 48        228925.14389
## 49        229699.81772
## 50        228273.89458
## 51        228747.98951
```

```
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##     variable relative_importance scaled_importance percentage
## 1    PROD_ID    5758912512.000000          1.000000   0.226696
## 2  MIN_PRICE    4524772352.000000          0.785699   0.178115
## 3 MEAN_PRICE    4128257792.000000          0.716847   0.162506
## 4        DAY    3647966464.000000          0.633447   0.143600
## 5   SD_PRICE    2659613696.000000          0.461826   0.104694
## 6  MAX_PRICE    2469554176.000000          0.428823   0.097212
## 7      MONTH    2214638592.000000          0.384558   0.087178
```

```r
g <- ggplot(data = RF_summary, aes(x = variable, y = scaled_importance, fill = variable)) +
geom_bar(stat = "identity", position = "dodge")
g
```



We found out that the field YEAR was irrelavant for the set, so we can remove it, but for meanings of futher inspection we´ll let it be. We could detect the principals fields within the summary of the model:

- PROD_ID
- DAY
- MEAN_PRICE
- MIN_PRICE

It appears that some produtcs sell more than others and some days are more important as well, problably it has something to do with the day of week. The mean and min could work together as lower prices would decrease the mean and make the product more affordable.

Let´s see if we can make it even more precise.

```
RF2 <- h2o.randomForest(
    x = fields,
    y = "QTY_ORDER",
    training_frame = train,
    validation_frame = test,
    ntrees = 75,
    max_depth = 35,
    seed = 1234
  )
```

```
## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Dropping constant columns: [YEAR].
```

```
##
  |
  |                                                                      |   0%
  |
  |=                                                                     |   1%
  |
  |=================================                                     |  47%
  |
  |============================================================          |  85%
  |
  |======================================================================| 100%
```

```
  h2o.r2(RF2, valid = T)
```

```
## [1] 0.4140149
```

It represents a small increase in precision but it´s not a good one, let´s try to create a more accurate model. We could use the DAY variable, let´s find the median prices for each day and what day of week it is. The week must have something to do with the sales as well, so we´ll calculate the median

```
merged_data <-
  mutate(merged_data,
  WD = wday(ymd(paste(
    YEAR, sprintf("%02d", MONTH), sprintf("%02d", DAY), sep = ""))),
    WEEK = lubridate::week(ymd(paste(
    YEAR, sprintf("%02d", MONTH), sprintf("%02d", DAY), sep = ""))))

  mean_prices_wd <- group_by(merged_data, WD) %>%
  summarise(MEAN_WD = median(MIN_PRICE))


  mean_prices_week <- group_by(merged_data, WEEK) %>%
  summarise(MEAN_WEEK = median(MIN_PRICE))


  merged_data <- merge(merged_data, mean_prices_wd)

  merged_data <- mutate(merged_data,
              DIF_WD = MEDIAN_PRICE / MEAN_WD,
              DIF_WEEK = MEDIAN_PRICE / WEEK)
```

```r
tail(merged_data)
```

```
##      WD PROD_ID YEAR MONTH DAY QTY_ORDER MIN_PRICE MAX_PRICE MEAN_PRICE
## 1913  7      P4 2015    10   3       108    492.15       497   493.7667
## 1914  7      P7 2015     1   3       915    703.12       799   771.8340
## 1915  7      P3 2015     9  12         1   1002.32      1499  1188.8814
## 1916  7      P2 2015     4   4       131    695.13       879   790.7672
## 1917  7      P5 2015     7   4        23    797.05       799   798.1750
## 1918  7      P3 2015     8   8         3   1099.00      1499  1213.2857
##         SD_PRICE MEDIAN_PRICE WEEK MEAN_WD    DIF_WD  DIF_WEEK
## 1913   2.3879749      492.150   40  738.63 0.6663011  12.30375
## 1914  38.6591560      799.000    1  738.63 1.0817324 799.00000
## 1915 124.3456555     1185.575   37  738.63 1.6051000  32.04257
## 1916  59.9173717      760.710   14  738.63 1.0298932  54.33643
## 1917   0.7030776      798.000   27  738.63 1.0803785  29.55556
## 1918 184.0174825     1099.000   32  738.63 1.4878897  34.34375
```

Let´s see how our model goes now.

```r
fields <- setdiff(setdiff(names(train), "QTY_ORDER"), "MEDIAN_PRICE")

merged_data.hex <-
  as.h2o(merged_data, destination_frame = "merged_data2.hex")
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
data_to_model <-
  h2o.splitFrame(data = merged_data.hex ,
                 ratios = 0.80,
                 seed = 1234)

train <- data_to_model[[1]]
test <- data_to_model[[2]]


RF3 <- h2o.randomForest(
  x = fields,
  y = "QTY_ORDER",
  training_frame = train,
  validation_frame = test,
  ntrees = 150,
  max_depth = 45,
  seed = 1234
)
```

```
## Warning in .h2o.startModelJob(algo, params, h2oRestApiVersion): Dropping constant columns: [YEAR].
```

```
##
  |
  |                                                                 |   0%
  |
  |====================                                             |  31%
```

```
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%
```

```r
h2o.r2(RF3, valid = T)
```

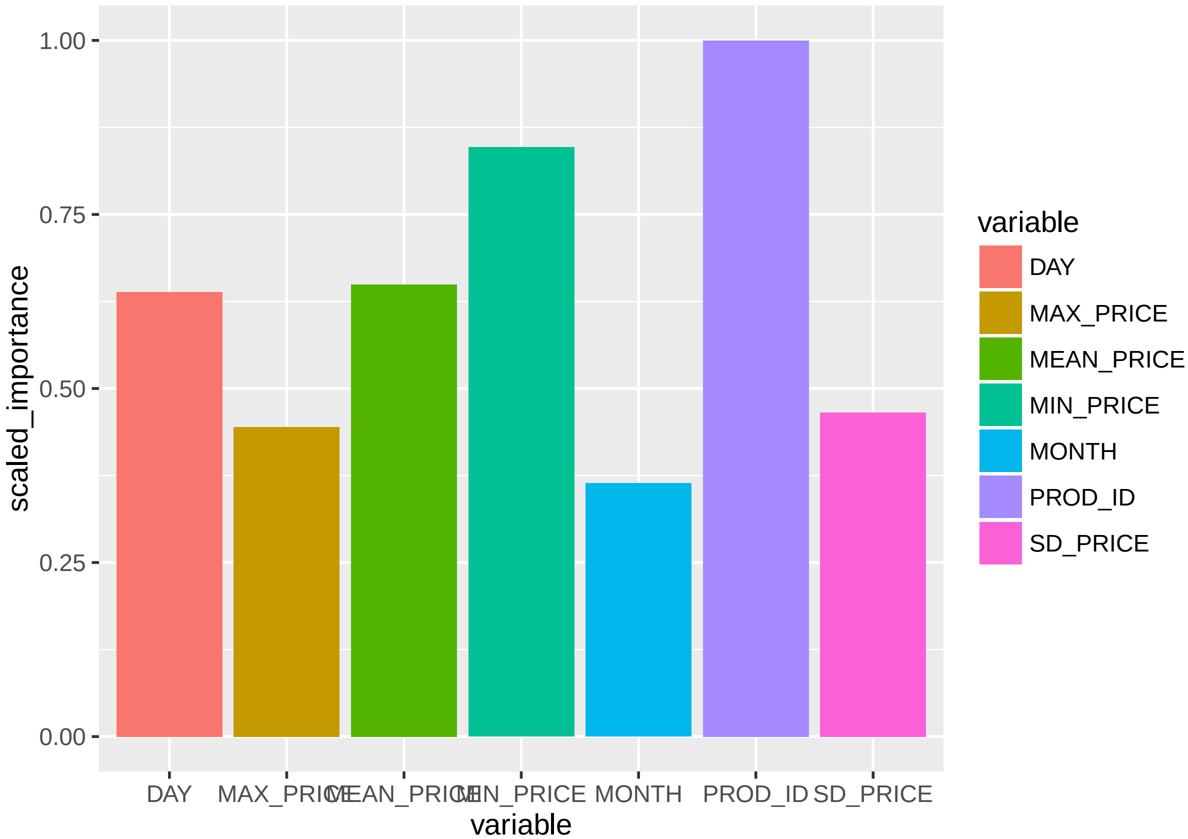
```
## [1] 0.4717004
```

We could get a precision of 47% this time. Let´s see how this new tunning changed the model.

```
## Model Details:
## ==============
##
## H2ORegressionModel: drf
## Model Key:  DRF_model_R_1477301123940_5
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             150                      150             1381127        19
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        33   23.55333        273        870   727.82000
##
## H2ORegressionMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## MSE:  313319.3
## RMSE:  559.7493
## MAE:  213.1122
## RMSLE:  1.162761
## Mean Residual Deviance :  313319.3
##
##
## H2ORegressionMetrics: drf
## ** Reported on validation data. **
##
## MSE:  295811.9
## RMSE:  543.8859
## MAE:  228.3817
## RMSLE:  1.12914
## Mean Residual Deviance :  295811.9
##
##
##
##
## Scoring History:
##             timestamp   duration number_of_trees training_rmse
## 1 2016-10-24 07:25:58  0.044 sec               0
## 2 2016-10-24 07:25:58  0.213 sec               1     890.97959
## 3 2016-10-24 07:25:58  0.234 sec               2     788.72355
## 4 2016-10-24 07:25:58  0.252 sec               3     728.94894
## 5 2016-10-24 07:25:58  0.268 sec               4     679.64253
##   training_mae training_deviance validation_rmse validation_mae
## 1
```

```
## 2    291.50841      793844.62158       712.97027      258.89478
## 3    274.73954      622084.83473       581.06056      221.52541
## 4    254.60491      531366.55673       572.85010      221.30372
## 5    248.85136      461913.97013       577.91312      222.56125
##   validation_deviance
## 1
## 2        508326.60115
## 3        337631.37608
## 4        328157.23671
## 5        333983.57166
##
## ---
##               timestamp   duration number_of_trees training_rmse
## 141 2016-10-24 07:26:02  3.937 sec             140     560.77210
## 142 2016-10-24 07:26:02  3.968 sec             141     560.30275
## 143 2016-10-24 07:26:02  4.017 sec             142     560.15930
## 144 2016-10-24 07:26:02  4.053 sec             143     560.62519
## 145 2016-10-24 07:26:02  4.084 sec             144     560.76013
## 146 2016-10-24 07:26:02  4.174 sec             150     559.74930
##     training_mae training_deviance validation_rmse validation_mae
## 141    213.79224      314465.34436       542.33695      227.49078
## 142    213.67385      313939.16811       543.05268      227.79596
## 143    213.59217      313778.44258       543.37974      227.79245
## 144    213.58200      314300.60058       543.53076      228.04200
## 145    213.62094      314451.92561       543.38057      228.12174
## 146    213.11223      313319.28132       543.88589      228.38168
##     validation_deviance
## 141        294129.36906
## 142        294906.20831
## 143        295261.54631
## 144        295425.68686
## 145        295262.44072
## 146        295811.86385
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##     variable relative_importance scaled_importance percentage
## 1    PROD_ID  16743229440.000000          1.000000   0.226808
## 2  MIN_PRICE  14175352832.000000          0.846632   0.192023
## 3 MEAN_PRICE  10871889920.000000          0.649331   0.147273
## 4        DAY  10690825216.000000          0.638516   0.144820
## 5   SD_PRICE   7796886016.000000          0.465674   0.105618
## 6  MAX_PRICE   7446653952.000000          0.444756   0.100874
## 7      MONTH   6096412160.000000          0.364112   0.082583
```

```
ggplot(data=F3_summary, aes(x = variable, y = scaled_importance, fill = variable)) +
 geom_bar(stat="identity", position = "dodge")
```



There were some increasing in importance for the fields, specially the MIN_PRICE. Now our model had a good precison gain.

Conclusions

Our model could predict the prices with a 47% score. To futher improvements we could extract more relationships with the time period and price practiced, as it appears to have significent relationship.

If we had more data, including more years this model could show better results.