Seminar6

2024-10-23

Content

- Import data, load packages and estimate previous models: LM1, LM2, LM6, BackwardElim, LASSO, RIDGE, EN
- Complete Subset Linear Regression
- Simple decision tree
  - Estimate
  - Visualize
  - Predict
  - Evaluate evaluation
- More variables - shallow tree
- More variables - deep tree
- More variables - deep tree -> pre-pruning
- More variables - deep tree -> penalization
- Bagging
- Random forest
- Boosted tree
- Task

Import data, load packages and estimate models from previous session

First load training and testing datasets into the workspace

```
oct <- read.csv(file='C://Users//ckt//OneDrive - MUNI//Institutions//MU//ML Finance//2024//oct.csv')
ocr <- read.csv(file='C://Users//ckt//OneDrive - MUNI//Institutions//MU//ML Finance//2024//ocr.csv')
ocr$km2   <- ocr$km^2
ocr$km3   <- ocr$km^3
ocr$age2  <- ocr$age^2
ocr$age3  <- ocr$age^3
ocr$kmage <- ocr$km * ocr$age
ocr$kmkw  <- ocr$km * ocr$kw
ocr$kmkwage <- ocr$km * ocr$kw * ocr$age
oct$km2   <- oct$km^2
oct$km3   <- oct$km^3
oct$age2  <- oct$age^2
oct$age3  <- oct$age^3
oct$kmage <- oct$km * oct$age
oct$kmkw  <- oct$km * oct$kw
oct$kmkwage <- oct$km * oct$kw * oct$age
```

Packages/libraries that we will need

```
library(glmnet)  # LASSO, RIDGE, EN
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(rpart)   # Decision trees
library(tree)    # Creating trees
library(rpart.plot) # Having some plots of trees
library(ranger)  # Random forest
library(gbm) # Boosted regression tree
```

```
## Loaded gbm 2.1.8.1
```

```
library(foreach) # Parallel look
library(doParallel) # Parallel computing
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(MCS)     # Model confidence set
```

Before working with backward, forward and bi-directional elimination we estimate some baseline models that we want to out-perform.

```
m1 <- lm(price ~ km      , data=ocr)
m2 <- lm(price ~ age     , data=ocr)
m3 <- lm(price ~ km + age, data=ocr)
p1 <- predict(m1,new=oct)
p2 <- predict(m2,new=oct)
p3 <- predict(m3,new=oct)
```

We are going to import function

```
source(file='C://Users//ckt//OneDrive - MUNI//Institutions//MU//ML Finance//2024//Functions.R')
```

And we can create model 6 and corresponding predictions

```
features <- c('km','age','kw','trans','combi','allw','ambi','styl','rs','dsg','scr','diesel','lpg','hybr
m6 <- bwelim(dt=ocr,pt=0.05,dep='price',features=features)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
p6 <- predict(m6,new=oct)
```

We will work with octavia dataset

```
features <- c('km','age','kw','trans','combi','allw','ambi','styl','rs','dsg','scr','diesel','lpg','hybr
X = as.matrix(ocr[,features])
Y = as.matrix(ocr['price'])
XT = as.matrix(oct[,features])
```

Estimate penalized models

```
cvm = cv.glmnet(x=X,y=Y,type.measure='mse',nfolds=10,alpha=1)
p9 = predict(cvm,newx = XT,s='lambda.min')
```

```
cvm = cv.glmnet(x=X,y=Y,type.measure='mse',nfolds=10,alpha=0)
p11 = predict(cvm,newx = XT,s='lambda.min')
cvm = cv.glmnet(x=X,y=Y,type.measure='mse',nfolds=10,alpha=0.5)
p13 = predict(cvm,newx = XT,s='lambda.min')
```

Complete Subset Linear Regression

We are going to an imported function (it was created by me for the purpose of the course) csr.method(), q - is the complexity parameter, cv - is the size of the cross-validation samples, trim - trimmed average, nc - number of cores to use. Let's try it out. However, if you type 'wrong' number into 'q' the number of models you have to estimates will be too large and we do not have time for this. Let's try q=3 (if you have 4 cores) and q=4 if you have 4+ physical cores.

```
spec <- gen.fm(dep='price',x=features)
A = Sys.time()
p = csr.method(spec,train=ocr,test=oct,q=5,cv=10,trim=0.2,nc=8)
Sys.time()-A
```

```
## Time difference of 2.32052 mins
```

Highly correlated predictions

```
cor(p)
```

```
##              pred.top.1 pred.top.10 pred.top.100
## pred.top.1    1.0000000   0.9987280    0.9974273
## pred.top.10   0.9987280   1.0000000    0.9992452
## pred.top.100  0.9974273   0.9992452    1.0000000
```

```
p14 = p[,1]
p15 = p[,2]
p16 = p[,3]
```

We combine forecasts & apply the sanity filter

```
pred <- cbind(true=oct$price,p1,p2,p3,p6,p9,p11,p13,p14,p15,p16)
for (i in 2:ncol(pred)) pred[pred[,i]<0,i] = min(ocr$price)
mse = pred[,-1]
mae = mse
for (i in 2:ncol(pred)) {
  mse[,i-1] =    (pred[,1] - pred[,i])^2
  mae[,i-1] = abs(pred[,1] - pred[,i])
}
apply(mse,2,mean)
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16
##    4474717    4157797    4248709
```

```
apply(mae,2,mean)
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16
##   1571.303   1495.547   1517.360
```

The CSLR does not seem to work especially well with our data

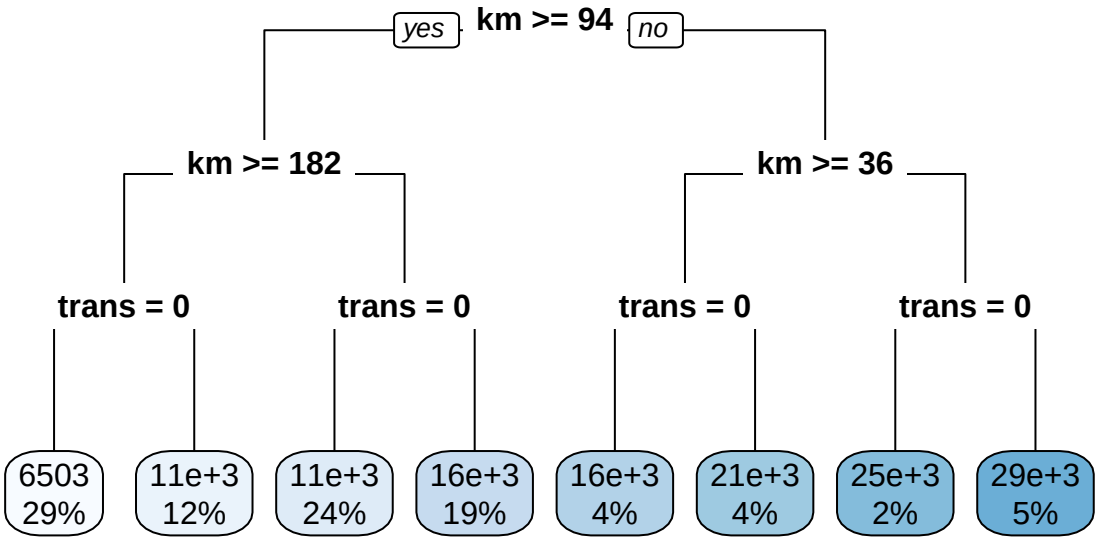
Decision trees

Can we do any better with a decision tree? Let's see. Estimate a simple one

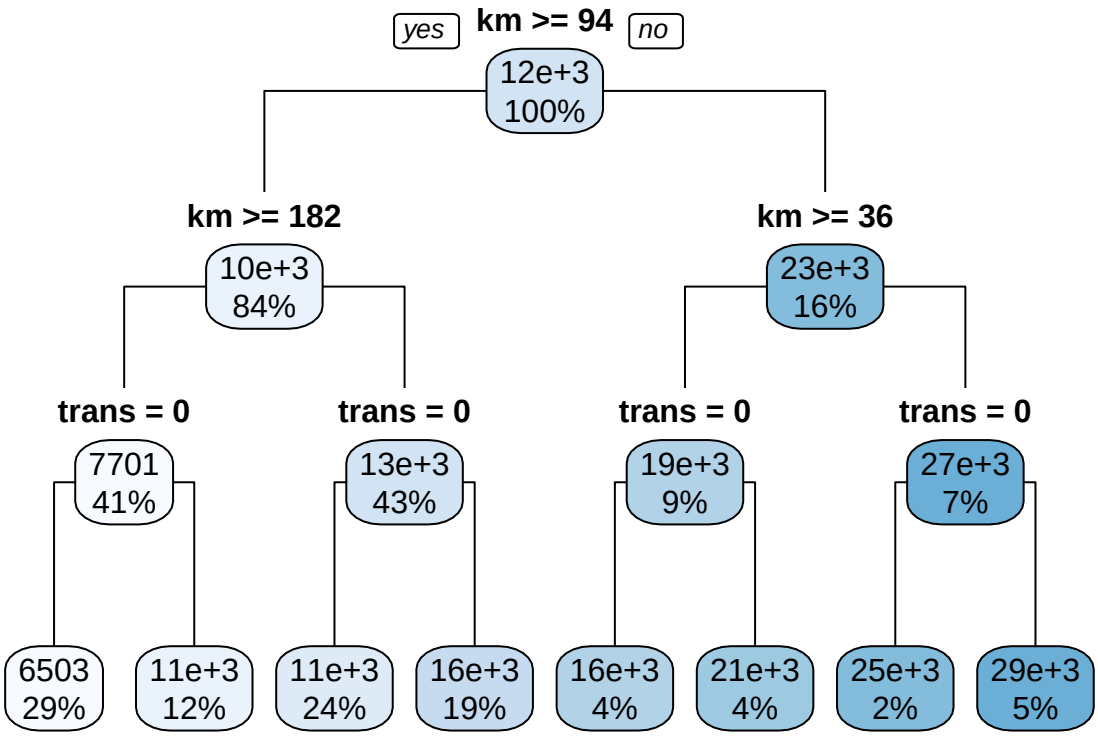
```
t1 = rpart(price~km+trans,data=ocr,method='anova',model=TRUE,
           control=rpart.control(cp=0,maxdepth=3))
```

Visualize & Predict & Evaluate

```
rpart.plot(t1,type=0)
```




```
rpart.plot(t1,type=1)
```




```
p17  = predict(t1,new=oct)
pred = cbind(pred, p17)
apply(   (pred[,-1] - pred[,1])^2,2,mean)
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16        p17
##    4474717    4157797    4248709   14691249
```

```
apply(abs(pred[,-1] - pred[,1])  ,2,mean)
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17
##   1571.303   1495.547   1517.360   3032.057
```

Not very accurate...

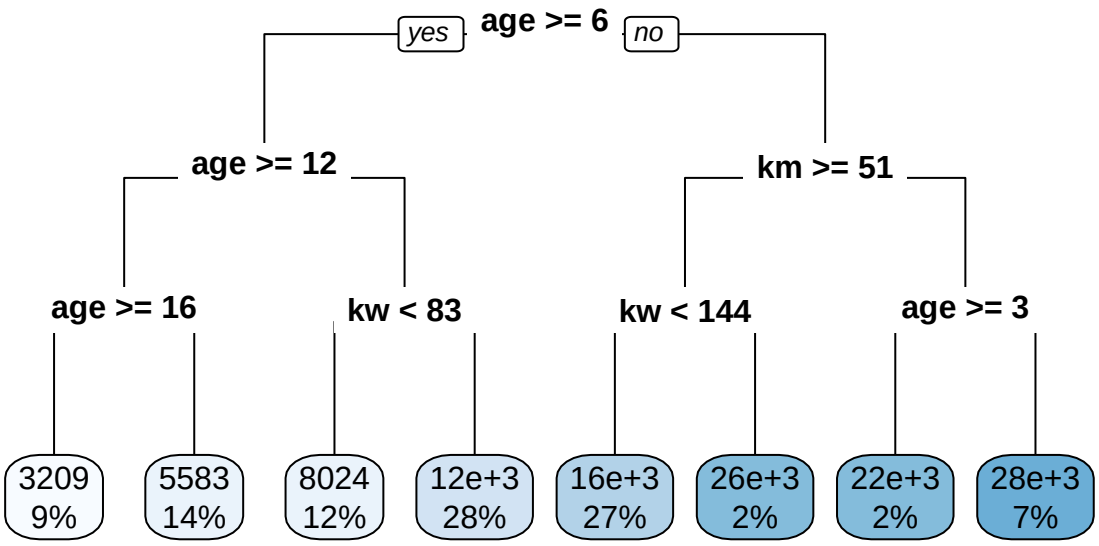
Advanced Task

Test your skills. Create a figure - similar to that on slide 9 of the Lecture 5 (Segmentation of the feature space) (use ChatGPT if you dare)

Let's try a tree with more variables - make it a shallow tree. We will not use interactions

```
features <- c('km','age','kw','trans','combi','allw','ambi','styl','rs','dsg','scr','diesel','lpg','hybr
spec <- gen.fm(dep='price',x=features)
t2 = rpart(spec,
```

```
          data=ocr,method='anova',model=TRUE,
          control=rpart.control(cp=0.0001,maxdepth=3))
rpart.plot(t2,type=0)
```




```
p18 = predict(t2,new=oct)
pred = cbind(pred, p18)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16        p17        p18
##    4474717    4157797    4248709   14691249    7903309
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17        p18
##   1571.303   1495.547   1517.360   3032.057   2029.873
```

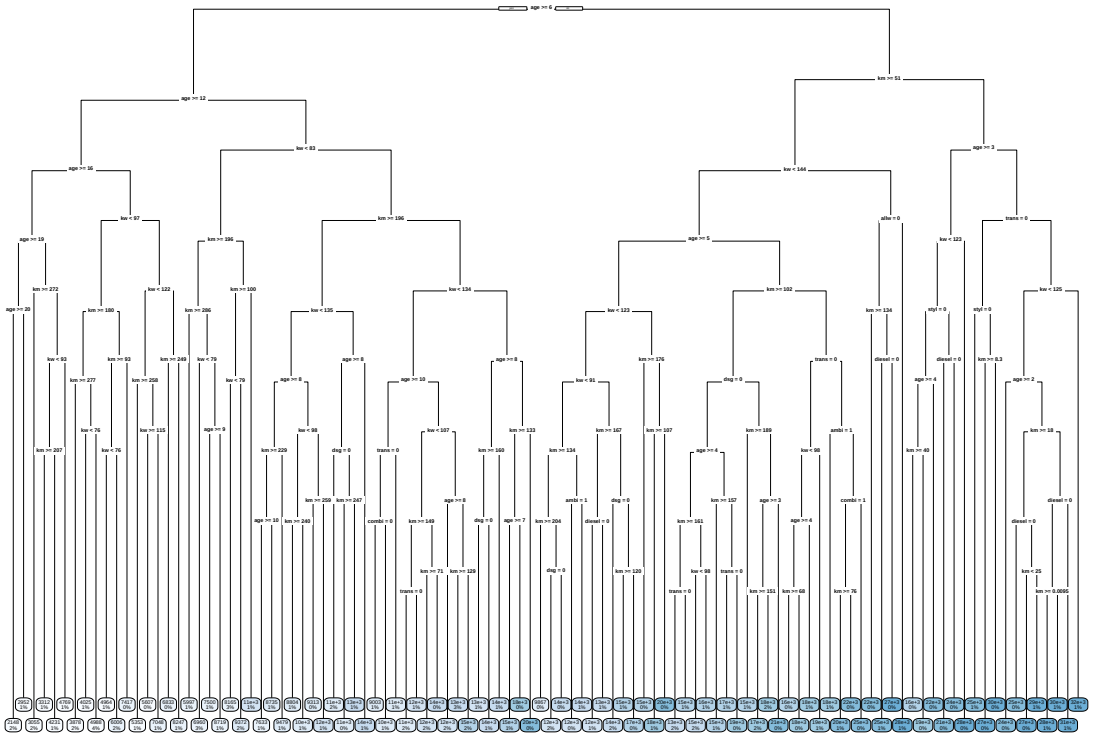
Better but still not great. Let's try a deeper tree

```
t3 = rpart(spec,
          data=ocr,method='anova',model=TRUE,
          control=rpart.control(cp=0.0001,maxdepth=9))
```

Visualization get's a little bit tricky - it's not necessary

```
rpart.plot(t3,type=0)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
p19  = predict(t3,new=oct)
pred = cbind(pred, p19)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16        p17        p18        p19
##    4474717    4157797    4248709   14691249    7903309    3886492
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17        p18        p19
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063
```

This is so much better -> closing on the best

Can we improve via pre-pruning?

```
?rpart.control
```

```
## starting httpd help server ... done
```

```
t4 = rpart(spec,
           data=ocr,method='anova',model=TRUE,
           control=rpart.control(cp=0.0001,minsplit=25,minbucket=12,maxdepth=9))
p20  = predict(t4,new=oct)
pred = cbind(pred, p20)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

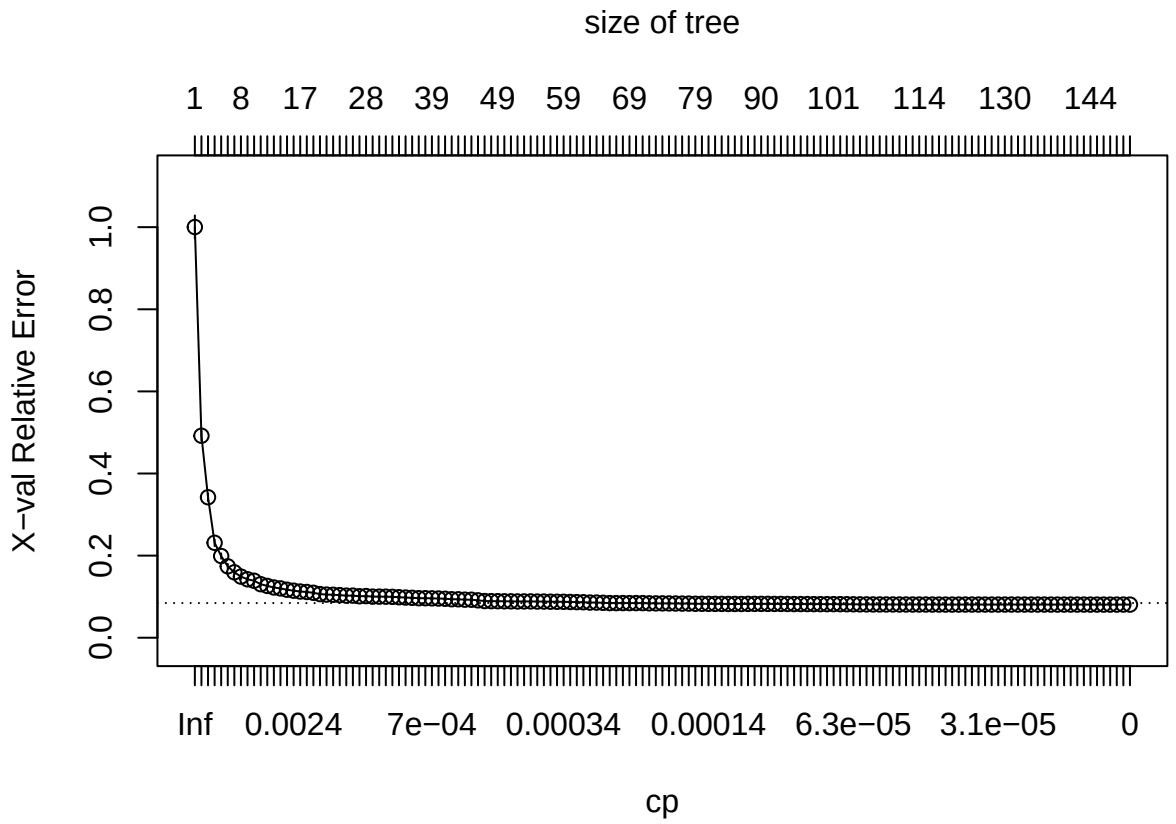
```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##         p14        p15        p16        p17        p18        p19        p20
##    4474717    4157797    4248709   14691249    7903309    3886492    3936528
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##         p14        p15        p16        p17        p18        p19        p20
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063   1389.144
```

Can we improve via penalization?

```
t5 = rpart(spec,
           data=ocr,method='anova',model=TRUE,
           control=rpart.control(cp=0,xval=10,minsplit=c(25),minbucket=12,maxdepth=9))
plotcp(t5)
```



Which is the optimal shrinkage parameter

```
cv.tbl = printcp(t5)
```

```
##
## Regression tree:
## rpart(formula = spec, data = ocr, method = "anova", model = TRUE,
##     control = rpart.control(cp = 0, xval = 10, minsplit = c(25),
##         minbucket = 12, maxdepth = 9))
##
## Variables actually used in tree construction:
##  [1] age    allw   ambi   combi  diesel dsg    km     kw     scr    styl
## [11] trans
##
## Root node error: 1.4895e+11/3024 = 49257135
##
## n= 3024
##
##             CP nsplit rel error   xerror      xstd
## 1   5.0849e-01      0  1.000000 1.000287 0.0284629
## 2   1.4983e-01      1  0.491514 0.491982 0.0131199
## 3   1.1092e-01      2  0.341681 0.342153 0.0093284
## 4   3.2540e-02      3  0.230761 0.231255 0.0082044
## 5   2.5058e-02      4  0.198221 0.198979 0.0065480
## 6   1.4588e-02      5  0.173163 0.173870 0.0061907
## 7   1.2357e-02      6  0.158575 0.159360 0.0055515
## 8   8.0535e-03      7  0.146218 0.148860 0.0053073
## 9   6.7339e-03      8  0.138164 0.142288 0.0051499
## 10  6.3434e-03      9  0.131430 0.138770 0.0050120
## 11  5.0120e-03     10  0.125087 0.130356 0.0048576
## 12  3.9368e-03     11  0.120075 0.126213 0.0045952
## 13  3.3318e-03     12  0.116138 0.122398 0.0044409
## 14  2.8001e-03     13  0.112806 0.120114 0.0043712
## 15  2.5920e-03     14  0.110006 0.116825 0.0042838
## 16  2.2836e-03     15  0.107414 0.114607 0.0042714
## 17  2.2740e-03     16  0.105131 0.112502 0.0042387
## 18  2.1365e-03     17  0.102857 0.111425 0.0042307
## 19  2.0208e-03     18  0.100720 0.109463 0.0042165
## 20  1.6233e-03     19  0.098699 0.106271 0.0042042
## 21  1.5793e-03     20  0.097076 0.105116 0.0041666
## 22  1.5446e-03     21  0.095497 0.104555 0.0041636
## 23  1.2989e-03     22  0.093952 0.103742 0.0041234
## 24  1.2639e-03     23  0.092653 0.102487 0.0041146
## 25  1.2173e-03     25  0.090125 0.102540 0.0041153
## 26  1.1151e-03     26  0.088908 0.101028 0.0040981
## 27  1.0650e-03     27  0.087793 0.101465 0.0041573
## 28  1.0649e-03     28  0.086728 0.100100 0.0041034
## 29  1.0641e-03     29  0.085663 0.100100 0.0041034
## 30  9.5436e-04     30  0.084599 0.099959 0.0041118
## 31  8.8912e-04     31  0.083645 0.099576 0.0041802
## 32  8.1429e-04     32  0.082756 0.098695 0.0041605
## 33  8.1278e-04     34  0.081127 0.097812 0.0041227
## 34  7.6183e-04     35  0.080314 0.097185 0.0041203
## 35  7.4718e-04     36  0.079552 0.096271 0.0040979
## 36  7.0551e-04     37  0.078805 0.096253 0.0041760
```

```
## 37  6.8840e-04     38  0.078100 0.095848 0.0041572
## 38  6.4620e-04     39  0.077411 0.095466 0.0041515
## 39  6.3144e-04     40  0.076765 0.094896 0.0041363
## 40  6.2352e-04     41  0.076134 0.093745 0.0040866
## 41  5.9955e-04     42  0.075510 0.093591 0.0040869
## 42  5.8956e-04     43  0.074911 0.092664 0.0040637
## 43  5.8890e-04     44  0.074321 0.092555 0.0040637
## 44  4.8092e-04     45  0.073732 0.091039 0.0040003
## 45  4.6164e-04     46  0.073251 0.089301 0.0039559
## 46  4.3442e-04     47  0.072790 0.089103 0.0039512
## 47  4.1881e-04     48  0.072355 0.088990 0.0039478
## 48  4.1231e-04     49  0.071936 0.088733 0.0039435
## 49  4.0384e-04     50  0.071524 0.088566 0.0039393
## 50  3.9911e-04     51  0.071120 0.088365 0.0039352
## 51  3.7370e-04     52  0.070721 0.088416 0.0039335
## 52  3.6413e-04     53  0.070347 0.088409 0.0039184
## 53  3.6411e-04     54  0.069983 0.088084 0.0038915
## 54  3.6115e-04     55  0.069619 0.088084 0.0038915
## 55  3.4844e-04     56  0.069258 0.087693 0.0038905
## 56  3.4532e-04     57  0.068910 0.087570 0.0038893
## 57  3.4384e-04     58  0.068564 0.087521 0.0038882
## 58  3.1362e-04     59  0.068220 0.087041 0.0039108
## 59  3.0589e-04     60  0.067907 0.086738 0.0039117
## 60  2.8529e-04     61  0.067601 0.086436 0.0038939
## 61  2.7449e-04     62  0.067316 0.085702 0.0038780
## 62  2.7063e-04     63  0.067041 0.085631 0.0038775
## 63  2.4685e-04     64  0.066770 0.085032 0.0038604
## 64  2.4177e-04     65  0.066524 0.084347 0.0038010
## 65  2.3638e-04     66  0.066282 0.084327 0.0038028
## 66  2.2817e-04     67  0.066045 0.084387 0.0038136
## 67  2.1651e-04     68  0.065817 0.084287 0.0038133
## 68  2.1471e-04     69  0.065601 0.084335 0.0038209
## 69  2.1028e-04     70  0.065386 0.084308 0.0038207
## 70  1.9694e-04     71  0.065176 0.083782 0.0038043
## 71  1.9654e-04     72  0.064979 0.083601 0.0037963
## 72  1.8674e-04     73  0.064782 0.083658 0.0038231
## 73  1.7411e-04     74  0.064596 0.083547 0.0038215
## 74  1.6794e-04     75  0.064421 0.083323 0.0038121
## 75  1.6610e-04     76  0.064253 0.083140 0.0038110
## 76  1.4873e-04     77  0.064087 0.083117 0.0038133
## 77  1.4509e-04     78  0.063939 0.082687 0.0037929
## 78  1.4415e-04     79  0.063794 0.082611 0.0037932
## 79  1.3862e-04     80  0.063649 0.082521 0.0037787
## 80  1.3469e-04     81  0.063511 0.082408 0.0037773
## 81  1.2429e-04     82  0.063376 0.082355 0.0037770
## 82  1.2314e-04     83  0.063252 0.082232 0.0037757
## 83  1.1896e-04     84  0.063129 0.082208 0.0037758
## 84  1.1231e-04     86  0.062891 0.082160 0.0037764
## 85  1.1138e-04     87  0.062778 0.082413 0.0037866
## 86  1.0982e-04     88  0.062667 0.082414 0.0037873
## 87  1.0959e-04     89  0.062557 0.082407 0.0037873
## 88  9.4377e-05     90  0.062448 0.082395 0.0037957
## 89  9.4286e-05     91  0.062353 0.082295 0.0037972
## 90  9.2321e-05     92  0.062259 0.082271 0.0037972
```

```
## 91  9.1997e-05     93  0.062167 0.082134 0.0037929
## 92  8.7576e-05     94  0.062075 0.081948 0.0037928
## 93  8.5671e-05     95  0.061987 0.081994 0.0037955
## 94  8.4646e-05     96  0.061901 0.081943 0.0037957
## 95  8.3563e-05     97  0.061817 0.081830 0.0037894
## 96  8.2598e-05     98  0.061733 0.081806 0.0037891
## 97  8.2066e-05     99  0.061651 0.081749 0.0037891
## 98  8.1969e-05    100  0.061569 0.081749 0.0037891
## 99  8.1292e-05    101  0.061487 0.081706 0.0037856
## 100 6.7098e-05    102  0.061405 0.081585 0.0037830
## 101 5.8314e-05    103  0.061338 0.081482 0.0037768
## 102 5.4532e-05    104  0.061280 0.081230 0.0037705
## 103 5.3884e-05    105  0.061225 0.081194 0.0037711
## 104 5.0563e-05    106  0.061171 0.081064 0.0037688
## 105 4.9379e-05    107  0.061121 0.081057 0.0037686
## 106 4.8199e-05    108  0.061072 0.080989 0.0037686
## 107 4.5713e-05    109  0.061023 0.080984 0.0037689
## 108 4.4998e-05    110  0.060978 0.080925 0.0037685
## 109 4.4907e-05    111  0.060933 0.080920 0.0037685
## 110 4.3673e-05    112  0.060888 0.080879 0.0037687
## 111 4.1518e-05    113  0.060844 0.080875 0.0037732
## 112 4.1424e-05    114  0.060802 0.080842 0.0037733
## 113 3.9942e-05    115  0.060761 0.080804 0.0037732
## 114 3.9191e-05    117  0.060681 0.080790 0.0037734
## 115 3.8147e-05    118  0.060642 0.080769 0.0037734
## 116 3.7939e-05    119  0.060604 0.080763 0.0037734
## 117 3.7877e-05    120  0.060566 0.080763 0.0037734
## 118 3.6732e-05    121  0.060528 0.080800 0.0037733
## 119 3.5101e-05    122  0.060491 0.080771 0.0037733
## 120 3.4752e-05    123  0.060456 0.080809 0.0037731
## 121 3.3277e-05    124  0.060421 0.080791 0.0037734
## 122 3.2023e-05    126  0.060355 0.080752 0.0037726
## 123 3.0774e-05    128  0.060291 0.080794 0.0037758
## 124 2.9529e-05    129  0.060260 0.080821 0.0037752
## 125 2.6406e-05    131  0.060201 0.080821 0.0037748
## 126 2.4364e-05    132  0.060175 0.080823 0.0037767
## 127 2.3353e-05    133  0.060150 0.080821 0.0037770
## 128 1.7396e-05    134  0.060127 0.080803 0.0037761
## 129 1.7385e-05    135  0.060110 0.080819 0.0037759
## 130 1.6911e-05    136  0.060092 0.080819 0.0037759
## 131 1.6856e-05    137  0.060075 0.080755 0.0037748
## 132 1.6175e-05    138  0.060058 0.080757 0.0037749
## 133 1.5092e-05    139  0.060042 0.080732 0.0037750
## 134 1.5080e-05    140  0.060027 0.080736 0.0037750
## 135 1.3670e-05    141  0.060012 0.080741 0.0037749
## 136 1.2791e-05    142  0.059998 0.080735 0.0037745
## 137 8.6119e-06    143  0.059986 0.080688 0.0037745
## 138 7.7197e-06    144  0.059977 0.080730 0.0037785
## 139 7.1078e-06    145  0.059969 0.080680 0.0037767
## 140 6.1418e-06    146  0.059962 0.080673 0.0037767
## 141 5.2843e-06    147  0.059956 0.080691 0.0037766
## 142 4.0469e-06    148  0.059951 0.080710 0.0037776
## 143 0.0000e+00    149  0.059947 0.080690 0.0037776
```

```
head(cv.tbl)
```

```
##           CP nsplit rel error    xerror        xstd
## 1 0.50848560      0 1.0000000 1.0002870 0.028462924
## 2 0.14983351      1 0.4915144 0.4919815 0.013119890
## 3 0.11091999      2 0.3416809 0.3421528 0.009328403
## 4 0.03254003      3 0.2307609 0.2312549 0.008204436
## 5 0.02505781      4 0.1982209 0.1989790 0.006547976
## 6 0.01458827      5 0.1731631 0.1738696 0.006190652
```

```
opt.cp = cv.tbl[which(cv.tbl[,4] == min(cv.tbl[,4])),1]
```

You can force him to estimate the model with optimal shrinkage

```
t5 = rpart(spec,
           data=ocr,method='anova',model=TRUE,
           control=rpart.control(cp=opt.cp,minsplit=c(25),minbucket=12,maxdepth=9))
p21  = predict(t5,new=oct)
pred = cbind(pred, p21)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16        p17        p18        p19        p20
##    4474717    4157797    4248709   14691249    7903309    3886492    3936528
##        p21
##    3891180
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17        p18        p19        p20
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063   1389.144
##        p21
##   1364.625
```

Bagging

Recall - it involves bootstrapping! Number of bootstrap samples

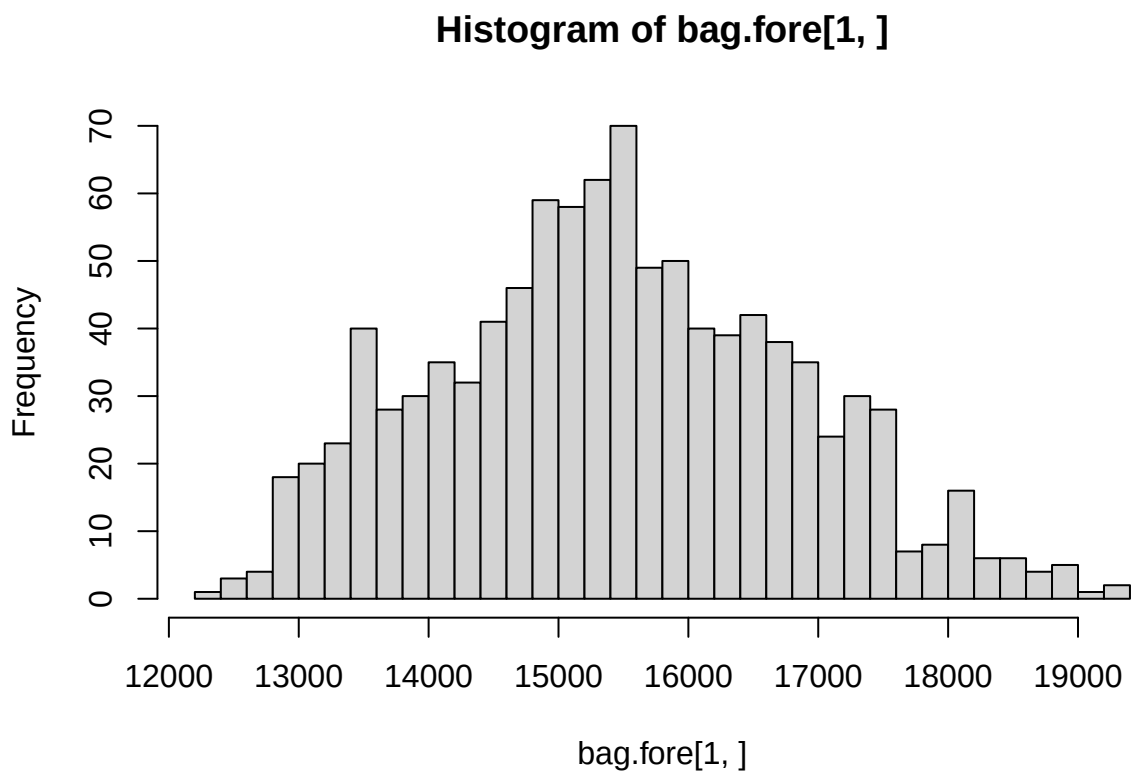
```
B = 1000
NT = dim(ocr)[1]
bag.fore = matrix(NA,nrow=dim(oct)[1],ncol=B)
rownames(bag.fore) = rownames(oct)
for (b in 1:B) {
  if (b %in% seq(0,B,100)) print(b)
  # Randomly select (with replacement) some row numbers
  idx = sample(1:NT,NT,replace=T)
  # Create the bootstrap sample
  auta.train.b = ocr[idx ,]
  # Estimate the model
  bt = rpart(spec,
             data=auta.train.b,method='anova',model=TRUE,
             control=rpart.control(cp=0,xval=10,minsplit=c(25),minbucket=12,maxdepth=9))
  # Prediction on a testing sample
```

```
  bag.fore[,b] = predict(bt,new=oct)
}
```

```
## [1] 100
## [1] 200
## [1] 300
## [1] 400
## [1] 500
## [1] 600
## [1] 700
## [1] 800
## [1] 900
## [1] 1000
```

For every single observations in the testing dataset you have B predictions

```
hist(bag.fore[1,],breaks=50) # nice
```



This way you could estimate 'confidence in your prediction'. For example, for the first car, the 95% confidence would be

```
quantile(bag.fore[1,],p=c(0.025,0.975))
```

```
##     2.5%    97.5%
## 12976.91 18195.91
```

Now evaluate

```
p22  = apply(bag.fore,1,mean,na.rm=T)
```

```
pred = cbind(pred, p22)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16        p17        p18        p19        p20
##    4474717    4157797    4248709   14691249    7903309    3886492    3936528
##        p21        p22
##    3891180    3091080
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17        p18        p19        p20
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063   1389.144
##        p21        p22
##   1364.625   1217.055
```

https://tenor.com/bnHxa.gif

Random forest - can it get any better?

Let's try to decorrelate trees. Number of trees

```
B = 5000
```

Depth of the trees

```
depth = c(3,6,9,12)
```

Number of depth parameters

```
ND = length(depth)
```

Number of random 'picks' of features to consider in each split

```
mtry = c(3,6,9)
```

Number of mtry parameters

```
NR = length(mtry)
```

Number of cross-validations

```
cv = 10
```

For each cross-validation sample I need to store predictions. MSE for CV

```
rf.cv = array(NA,dim=c(NR,ND,cv))
dimnames(rf.cv)[[1]] = paste('Try',mtry,'features')
dimnames(rf.cv)[[2]] = paste('Depth',depth)
dimnames(rf.cv)[[3]] = paste('CV sample',1:cv)
```

I need to find the average values for each cross-validation

```
rf.cv.ave = array(NA,dim=c(NR,ND))
dimnames(rf.cv.ave)[[1]] = paste('Try',mtry,'features')
dimnames(rf.cv.ave)[[2]] = paste('Depth',depth)
```

Now we loop over different parameters & cross-validation samples.

```
# Number of mtry
for (m in 1:NR) {
  num.try = mtry[m]
  # Depth
  for (d in 1:ND) {
    num.depth = depth[d]
    # Now cross-validation
    for (r in 1:cv) {
      # Select data
      idx = c(((r-1)*(NT/10)+1):(r*(NT/10)))
      auta.train.cvin = ocr[-idx,]
      auta.train.cout = ocr[+idx,]

      # Estimate the model
      rf.tree = ranger(spec,
                       data=auta.train.cvin,
                       num.trees=B,mtry=num.try,min.node.size=5,max.depth=num.depth)
      pred.rf.cv = predict(rf.tree,data=auta.train.cout)$predictions
      rf.cv[m,d,r] = mean((pred.rf.cv - auta.train.cout$price)^2)
    }
    # Average
    rf.cv.ave[m,d] = mean(rf.cv[m,d,])
  }
}
```

We estimated random forest(S) under different hyper-parameters. Now which rando forest has the lowest forecast error via cross-validation?

```
which(rf.cv.ave == min(rf.cv.ave), arr.ind=TRUE)
```

```
##                row col
## Try 6 features   2   4
```

```
mtry.opt  = mtry[3]
depth.opt = depth[4]
rf = ranger(spec,data=ocr,num.trees=B,mtry=mtry.opt,min.node.size=5,max.depth=depth.opt)
```

Now evaluate

```
p23  = predict(rf,data=oct)$predictions
pred = cbind(pred, p23)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##        p14        p15        p16        p17        p18        p19        p20
##    4474717    4157797    4248709   14691249    7903309    3886492    3936528
##        p21        p22        p23
##    3891180    3091080    2700650
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17        p18        p19        p20
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063   1389.144
##        p21        p22        p23
```

```
##   1364.625   1217.055   1164.026
```

https://tenor.com/bSFw3.gif

Boosting

We will use a package gbm()

```
mod.gbm = gbm(spec,data=ocr,distribution='gaussian',n.trees=10000,
              interaction.depth=depth.opt,shrinkage=0.001,bag.fraction=1)
```

Now evaluate

```
p24  = predict(mod.gbm,new=oct)
```

```
## Using 10000 trees...
```

```
pred = cbind(pred,p24)
apply(pred[,-1],2,function(x,y) mean((x-y)^2), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##         p14        p15        p16        p17        p18        p19        p20
##    4474717    4157797    4248709   14691249    7903309    3886492    3936528
##         p21        p22        p23        p24
##    3891180    3091080    2700650    2764740
```

```
apply(pred[,-1],2,function(x,y) mean(abs(x-y)), y=pred[,1])
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##         p14        p15        p16        p17        p18        p19        p20
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063   1389.144
##         p21        p22        p23        p24
##   1364.625   1217.055   1164.026   1149.121
```

Final evaluation with winsorization!

```
for (i in 2:ncol(pred)) pred[pred[,i]<0,i] = min(ocr$price)
```

Now we find the Mean Square Error and Mean Absolute Error:

```
mse = pred[,-1]
mae = mse
for (i in 2:ncol(pred)) {
  mse[,i-1] =    (pred[,1] - pred[,i])^2
  mae[,i-1] = abs(pred[,1] - pred[,i])
}
apply(mse,2,mean)
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   20753493   13590599    9458852    3267236    3382070    4781821    3319137
##         p14        p15        p16        p17        p18        p19        p20
##    4474717    4157797    4248709   14691249    7903309    3886492    3936528
##         p21        p22        p23        p24
##    3891180    3091080    2700650    2764740
```

```
apply(mae,2,mean)
```

```
##         p1         p2         p3         p6 lambda.min lambda.min lambda.min
##   3529.504   2770.189   2352.664   1313.841   1340.571   1595.134   1325.330
##        p14        p15        p16        p17        p18        p19        p20
##   1571.303   1495.547   1517.360   3032.057   2029.873   1389.063   1389.144
##        p21        p22        p23        p24
##   1364.625   1217.055   1164.026   1149.121
```

Which model is the best one?

```
MCSprocedure(mse,B=1000)
```

```
## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p17 eliminated 2024-10-23 13:56:01.578839

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p1 eliminated 2024-10-23 13:56:06.54383

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p3 eliminated 2024-10-23 13:56:11.311871
```

```
## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p2 eliminated 2024-10-23 13:56:15.361759
## Model p18 eliminated 2024-10-23 13:56:18.934327
## Model p14 eliminated 2024-10-23 13:56:22.124757
## Model p15 eliminated 2024-10-23 13:56:24.563432
## Model p19 eliminated 2024-10-23 13:56:26.65316
## Model p16 eliminated 2024-10-23 13:56:28.461275
## Model p20 eliminated 2024-10-23 13:56:30.040678
## Model p21 eliminated 2024-10-23 13:56:31.426917
## Model p6 eliminated 2024-10-23 13:56:32.596552
## ##############################################################################################################
## Superior Set Model created   :
##            Rank_M       v_M MCS_M Rank_R        v_R MCS_R    Loss
## lambda_min      5 -1.444257 1.000      6  3.5317261 0.011 3382070
## lambda_min     NA -1.444257    NA     NA  3.5317261    NA 4781821
## lambda_min     NA -1.444257    NA     NA  3.5317261    NA 3319137
## p22             6 -0.203317 0.971      3  3.1049738 0.011 3091080
## p23             1 -3.791215 1.000      1 -0.7490124 1.000 2700650
## p24             2 -3.691419 1.000      2  0.7490124 0.957 2764740
## p-value  :
## [1] 0.971
##
## ##############################################################################################################
##
## ----------------------------------------
## -          Superior Set of Models        -
## ----------------------------------------
##            Rank_M       v_M MCS_M Rank_R        v_R MCS_R    Loss
## lambda_min      5 -1.444257 1.000      6  3.5317261 0.011 3382070
## lambda_min     NA -1.444257    NA     NA  3.5317261    NA 4781821
## lambda_min     NA -1.444257    NA     NA  3.5317261    NA 3319137
## p22             6 -0.203317 0.971      3  3.1049738 0.011 3091080
## p23             1 -3.791215 1.000      1 -0.7490124 1.000 2700650
## p24             2 -3.691419 1.000      2  0.7490124 0.957 2764740
##
## Details
## ----------------------------------------
##
## Number of eliminated models  :   12
## Statistic    :   Tmax
## Elapsed Time :   Time difference of 37.68626 secs
```

```
MCSprocedure(mae,B=1000)
```

```
## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p17 eliminated 2024-10-23 13:56:39.236984

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p1 eliminated 2024-10-23 13:56:44.204261

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

##
## Model p3 eliminated 2024-10-23 13:56:48.808968

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
## non-missing arguments to max; returning -Inf

## Warning in max((abs(d_ij_mean[model_temp]))/(d_var_ij[1, model_temp]^0.5)): no
```

```
## non-missing arguments to max; returning -Inf
##
## Model p2 eliminated 2024-10-23 13:56:52.9596
## Model p14 eliminated 2024-10-23 13:56:56.355984
## Model p18 eliminated 2024-10-23 13:56:59.507008
## Model p15 eliminated 2024-10-23 13:57:02.265178
## Model p16 eliminated 2024-10-23 13:57:04.742087
## Model p19 eliminated 2024-10-23 13:57:06.744694
## Model p20 eliminated 2024-10-23 13:57:08.427741
## Model p21 eliminated 2024-10-23 13:57:09.900822
## Model p6 eliminated 2024-10-23 13:57:11.023123
## ##############################################################################################################
## Superior Set Model created   :
##            Rank_M        v_M MCS_M Rank_R        v_R MCS_R     Loss
## lambda_min      6 -0.1270448 0.986      6  4.6027304 0.003 1340.571
## lambda_min     NA -0.1270448    NA     NA  4.6027304    NA 1595.134
## lambda_min     NA -0.1270448    NA     NA  4.6027304    NA 1325.330
## p22             3 -1.6894839 1.000      3  3.4035661 0.003 1217.055
## p23             2 -3.9371679 1.000      2  0.7668813 0.947 1164.026
## p24             1 -4.9262356 1.000      1 -0.7668813 1.000 1149.121
## p-value  :
## [1] 0.986
##
## ##############################################################################################################
##
## ------------------------------------------
## -          Superior Set of Models        -
## ------------------------------------------
##            Rank_M        v_M MCS_M Rank_R        v_R MCS_R     Loss
## lambda_min      6 -0.1270448 0.986      6  4.6027304 0.003 1340.571
## lambda_min     NA -0.1270448    NA     NA  4.6027304    NA 1595.134
## lambda_min     NA -0.1270448    NA     NA  4.6027304    NA 1325.330
## p22             3 -1.6894839 1.000      3  3.4035661 0.003 1217.055
## p23             2 -3.9371679 1.000      2  0.7668813 0.947 1164.026
## p24             1 -4.9262356 1.000      1 -0.7668813 1.000 1149.121
##
## Details
## ------------------------------------------
##
## Number of eliminated models  :   12
## Statistic    :   Tmax
## Elapsed Time :   Time difference of 38.27958 secs
```

Competition - Task

Can you improve upon the apartments dataset's predictions?

```
apartments <- read.csv(file='C:\\Users\\ckt\\OneDrive - MUNI\\Institutions\\MU\\ML Finance\\2024\\datas
set.seed(99)
N     <- dim(apartments)[1]
idx   <- sample(1:N,size=floor(0.7*N),replace=FALSE)
```

```
train <- apartments[idx ,]
test  <- apartments[-idx,]
head(train)
```

```
##    rent city sub1 sub2 sub3 sub4 sub5 sub6 sub7 sub8 new recon part_recon area
## 48  800    1    1    0    0    0    0    0    0    0   0     1          0   70
## 33  800    1    0    0    0    0    0    1    0    0   1     0          0   57
## 44  800    1    0    0    0    1    0    0    0    0   1     0          0   56
## 22  900    1    1    0    0    0    0    0    0    0   0     1          0  105
## 62  550    0    0    0    0    0    0    0    0    0   0     1          0   54
## 32  800    1    1    0    0    0    0    0    0    0   0     1          0   70
##    balcony rooms floor cellar lift shops_km shops_min_car shops_min_walk
## 48       1     3     7      1    1      2.7             5             30
## 33       1     2     3      1    1      5.0             7             53
## 44       1     2     5      0    0      4.0             8             46
## 22       1     3     2      0    0      0.8             3              4
## 62       1     2     2      0    0      1.6             4             16
## 32       1     3     4      1    1      2.9             8             23
##    hospital_km hospital_min_car hospital_min_walk school_km school_min_car
## 48        3.00                5                33      2.30              4
## 33        7.10               10                81      0.29              2
## 44        5.30               11                60      3.40              8
## 22        3.10                8                21      2.10              4
## 62        0.65                2                 7      0.80              2
## 32        2.30                6                26      2.70              8
##    school_min_walk station_km station_min_car station_min_walk center_km
## 48              26       2.60               5                26       1.8
## 33               4       5.60               8                59       6.9
## 44              33       3.90               8                39       4.1
## 22              23       0.75               2                 5       1.9
## 62               7       1.90               5                20       1.9
## 32              24       2.90               7                21       1.3
##    center_min_car center_min_walk
## 48              5              19
## 33             15              64
## 44             10              37
## 22              5               9
## 62              5              15
## 32              6              12
```