bacs_hw15

110071010

2024-05-28

Setup

```
insurance <- read.csv("insurance.csv", header=TRUE, na.strings = "?")
names(insurance) <- c("age", "sex", "bmi", "children", "smoker", "regio
n", "charges")
head(insurance)
```

```
##   age    sex    bmi children smoker    region   charges
## 1  19 female 27.900        0    yes southwest 16884.924
## 2  18   male 33.770        1     no southeast  1725.552
## 3  28   male 33.000        3     no southeast  4449.462
## 4  33   male 22.705        0     no northwest 21984.471
## 5  32   male 28.880        0     no northwest  3866.855
## 6  31 female 25.740        0     no southeast  3756.622
```

Question 1

Create some explanatory models to learn more about charges

1a

Instruction

Create an OLS regression model and report which factors are significantly related to charges

```
ols <- lm(charges ~ age + factor(sex) + bmi + children
                    + factor(smoker) + factor(region), data = insurance)

summary(ols)
```

```
##
## Call:
## lm(formula = charges ~ age + factor(sex) + bmi + children + factor(s
moker) +
##     factor(region), data = insurance)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11304.9  -2848.1   -982.1   1393.9  29992.8
##
```

```
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)    
## (Intercept)              -11938.5      987.8 -12.086  < 2e-16 ***
## age                         256.9       11.9  21.587  < 2e-16 ***
## factor(sex)male            -131.3      332.9  -0.394 0.693348    
## bmi                         339.2       28.6  11.860  < 2e-16 ***
## children                    475.5      137.8   3.451 0.000577 ***
## factor(smoker)yes         23848.5      413.1  57.723  < 2e-16 ***
## factor(region)northwest    -353.0      476.3  -0.741 0.458769    
## factor(region)southeast   -1035.0      478.7  -2.162 0.030782 *  
## factor(region)southwest    -960.0      477.9  -2.009 0.044765 *  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 6062 on 1329 degrees of freedom
## Multiple R-squared:  0.7509, Adjusted R-squared:  0.7494 
## F-statistic: 500.8 on 8 and 1329 DF,  p-value: < 2.2e-16
```

Answer

The significant factors related to insurance charges are : **age, bmi, children, factor(smoker)yes, factor(region)southeast, and factor(region)southwest.**

1b

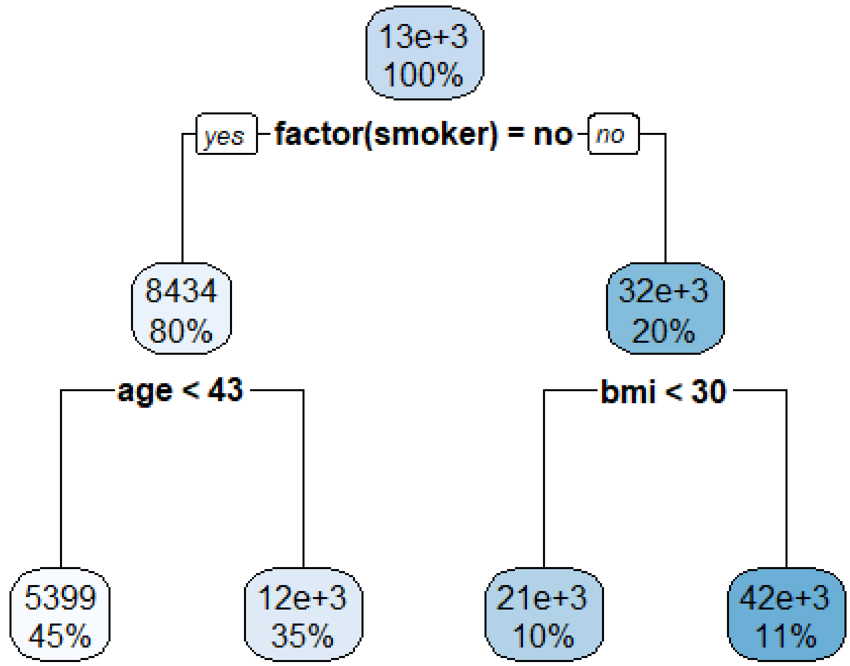
Create a decision tree (specifically, a regression tree) with default parameters to rpart().

```
library(rpart)
library(rpart.plot)
tree <- rpart(charges ~ age + factor(sex) + bmi + children
                    + factor(smoker) + factor(region), data = insurance)
```

i

Plot a visual representation of the tree structure

```
rpart.plot(tree)
```



ii

Question

How deep is the tree ? (see nodes with "decisions" – ignore the leaves at the bottom)

Answer

2

iii

How many leaf groups does it suggest to bin the data into?

```
sum(tree$frame$var == "<leaf>")
## [1] 4
```

iv

What conditions (combination of decisions) describe each leaf group?

Answer

1) smoker == yes & age < 43
2) smoker == yes & age >= 43
3) smoker == no & bmi < 30

4) smoker == no & bmi >= 30

Question 2

Let's use LOOCV to see how how our models perform predictively overall

```
fold_i_pe <- function(i, k, model, dataset, outcome) {
  folds <- cut(1:nrow(dataset), breaks=k, labels=FALSE)
  test_indices <- which(folds==i)
  test_set <- dataset[test_indices, ]
  train_set <- dataset[-test_indices, ]
  trained_model <- update(model, data = train_set)
  predictions <- predict(trained_model, test_set)
  dataset[test_indices, outcome] - predictions
}

# Run LOOCV
loocv_rmse <- function(model, dataset, outcome, k=nrow(dataset)) {
  shuffled_indices <- sample(1:nrow(dataset))
  dataset <- dataset[shuffled_indices,]
  fold_pred_errors <- sapply(1:k, \(kth) {
    fold_i_pe(kth, k, model, dataset, outcome)
  })
  pred_errors <- unlist(fold_pred_errors)
  rmse(pred_errors)
}

rmse <- function(errors) {
  sqrt(mean(errors^2))
}
```

2a

What is the RMSEout for the OLS regression model?

```
loocv_rmse(model = ols,dataset = insurance,outcome = "charges",k=nrow(i
nsurance))

## [1] 6087.388
```

2b

What is the RMSEout for the decision tree model?

```
loocv_rmse(model = tree,dataset = insurance,outcome = "charges",k=nrow
(insurance))

## [1] 5135.175
```

Moving onto bagging and boosting, we will only use split-sample testing to save time: partition the data to create training and test sets using an 80:20 split. Use the regression model and decision tree you created earlier for bagging and boosting.

Question 3

Let's see if bagging helps our models

3a

Implement the bagged_learn(…) and bagged_predict(…) functions.

```
bagged_learn <- function(model, dataset, b=100) {
  lapply(1:b, \(i) {
  n = nrow(dataset)
  train_set <- dataset[sample(1:n,n,replace = TRUE),]
  update(model,data = train_set)
  })
}

bagged_predict <- function(bagged_models, new_data) {
  predictions <- lapply(bagged_models,\(model){
    predict(model, new_data)
  })# get b predictions of new_data
as.data.frame(predictions) |> apply(X = _,1,mean) # apply a mean over t
he columns of predictions
}

rmse_out <- function(actuals, preds){
  sqrt(mean((actuals - preds)^2))
}
```

3b

What is the RMSEout for the bagged OLS regression?

```
train_indices <- sample(1:nrow(insurance),size = 0.80*nrow(insurance))
train_set <-insurance[train_indices,]
test_set <- insurance[-train_indices,]

bagged_models <- bagged_learn(ols, train_set, b =100)
bagged_predictions_via_ols <- bagged_predict(bagged_models, new_data =
test_set)
rmse_out(actuals= test_set$charges, preds = bagged_predictions_via_ols)

## [1] 6180.099
```

3c

What is the RMSEout for the bagged decision tree?

```
train_indices <- sample(1:nrow(insurance),size = 0.80*nrow(insurance))
train_set <-insurance[train_indices,]
test_set <- insurance[-train_indices,]

bagged_models <- bagged_learn(tree, train_set, b =100)
bagged_predictions_via_tree <- bagged_predict(bagged_models, new_data =
 test_set)
rmse_out(actuals= test_set$charges, preds = bagged_predictions_via_tree)

## [1] 5460.399
```

Question 4

Let's see if boosting helps our models. You can use a learning rate of 0.1 and adjust it if you find a better rate.

4a

Write boosted_learn(…) and boosted_predict(…) functions.

```
boosted_learn <- function(model, dataset, outcome, n=100, rate=0.1) {
  # Extract predictor variables
  predictors <- dataset[, setdiff(names(dataset), outcome)]

  # Initialize residuals with the actual outcome values
  res <- dataset[outcome]
  models <- list()

  # Iteratively train models on the residuals
  for (i in 1:n) {
    this_model <- update(model, data = cbind(residual=res, predictors))
    models[[i]] <- this_model
    predictions <- predict(this_model, newdata = dataset)
    res <- res - rate * predictions
  }
  list(models=models, rate=rate)
}

boosted_predict <- function(boosted_learning, new_data) {
  boosted_models <- boosted_learning$models
  rate <- boosted_learning$rate

  # Get predictions of new_data from each model
  predictions <- lapply(boosted_models, function(model) {
    predict(model, newdata = new_data)
```

```
  })

  # Convert list of predictions to data frame and remove row names
  pred_frame <- as.data.frame(predictions) |> unname()

  # Apply a sum over the columns of predictions, weighted by learning r
ate
  apply(pred_frame, 1, function(row) sum(row) * rate)
}
```

4b

What is the RMSEout for the boosted OLS regression?

```
train_indices <- sample(1:nrow(insurance),size = 0.80*nrow(insurance))
train_set <-insurance[train_indices,]
test_set <- insurance[-train_indices,]

boosted_models <- boosted_learn(ols, train_set, outcome = "charges")
boosted_predictions_via_ols <- boosted_predict(boosted_models, new_data
 = test_set)
rmse_out(actuals= test_set$charges, preds = boosted_predictions_via_ols)

## [1] 6593.229
```

4c

What is the RMSEout for the boosted decision tree?

```
train_indices <- sample(1:nrow(insurance),size = 0.80*nrow(insurance))
train_set <-insurance[train_indices,]
test_set <- insurance[-train_indices,]

boosted_models <- boosted_learn(tree, train_set, outcome = "charges")
boosted_predictions_via_tree <- boosted_predict(boosted_models, new_dat
a = test_set)
rmse_out(actuals= test_set$charges, preds = boosted_predictions_via_tre
e)

## [1] 5071.517
```

Question 5

Let’s engineer the best predictive decision trees. Let’s repeat the bagging and boosting of the decision tree several times to see if we can improve their performance. But this time, split the data 70:15:15 — use 70% as the training set, 15% as the validation set, and use the last 15% as the test set to obtain the final RMSEout.

5a

Repeat the bagging of the decision tree, using a base tree of maximum depth 1, 2, … n, keep training on the 70% training set, while the RMSEout of your 15% validation set keeps dropping; stop when the RMSEout has started increasing again (show prediction error at each depth). When you have identified the best maximum depth from the validation set, report the final RMSEout using the final 15% test set data.

```r
set.seed(111555777)
indices <- sample(1:nrow(insurance))
train_indices <- indices[1:round(0.7 * length(indices))]
validation_indices <- indices[(round(0.7 * length(indices)) + 1):round(0.85 * length(indices))]
test_indices <- indices[(round(0.85 * length(indices)) + 1):length(indices)]

train_set <- insurance[train_indices, ]
validation_set <- insurance[validation_indices, ]
test_set <- insurance[test_indices, ]

# Bagged learning function with depth control
bagged_learn_with_depth <- function(model, dataset, b=100, maxdepth) {
  lapply(1:b, function(i) {
    n <- nrow(dataset)
    train_set <- dataset[sample(1:n, n, replace = TRUE), ]
    rpart(charges ~ ., data = train_set, control = rpart.control(maxdepth = maxdepth))
  })
}

# Bagged predict function
bagged_predict <- function(bagged_models, new_data) {
  predictions <- lapply(bagged_models, function(model) {
    predict(model, new_data)
  })
  pred_frame <- as.data.frame(predictions)
  apply(pred_frame, 1, mean)
}

# RMSE calculation function
rmse_out <- function(actuals, preds) {
  sqrt(mean((actuals - preds)^2))
}

# Evaluate different depths
depths <- 1:10
validation_errors <- sapply(depths, function(depth) {
  bagged_models <- bagged_learn_with_depth(tree, train_set, b=100, maxdepth=depth)
```

```
  predictions <- bagged_predict(bagged_models, validation_set)
  error <- rmse_out(actuals = validation_set$charges, preds = predictio
ns)
  cat("Depth:", depth, "- RMSE:", error, "\n")
  error
})

best_depth <- depths[which.min(validation_errors)]
cat("Best depth:", best_depth, "\n")

# Train final model and evaluate on test set
combined_train_val_set <- rbind(train_set, validation_set)
final_bagged_models <- bagged_learn_with_depth(tree, combined_train_val
_set, b=100, maxdepth=best_depth)
final_predictions <- bagged_predict(final_bagged_models, test_set)
final_rmse <- rmse_out(actuals = test_set$charges, preds = final_predic
tions)
cat("Final RSME:", final_rmse)

## Depth: 1 - RMSE: 7122.474
## Depth: 2 - RMSE: 4615.182
## Depth: 3 - RMSE: 4357.394
## Depth: 4 - RMSE: 4382.662
## Depth: 5 - RMSE: 4375.483
## Depth: 6 - RMSE: 4392.027
## Depth: 7 - RMSE: 4388.18
## Depth: 8 - RMSE: 4419.96
## Depth: 9 - RMSE: 4380.942
## Depth: 10 - RMSE: 4390.918
## Best depth: 3
## Final RSME: 5609.81
```

5b

Let's find the best set of max tree depth and learning rate for boosting the decision tree: Use tree stumps of differing maximum depth (e.g., try intervals between 1 – 5) and differing learning rates (e.g., try regular intervals from 0.01 to 0.20). For each combination of maximum depth and learning rate, train on the 70% training set while and use the 15% validation set to compute RMSEout. When you have tried all your combinations, identify the best combination of maximum depth and learning rate from the validation set, but report the final RMSEout using the final 15% test set data.

```
# Split the data into training (70%), validation (15%), and test (15%)
sets
set.seed(111555777)
indices <- sample(1:nrow(insurance))
train_indices <- indices[1:round(0.7 * length(indices))]
validation_indices <- indices[(round(0.7 * length(indices)) + 1):round
(0.85 * length(indices))]
```

```r
test_indices <- indices[(round(0.85 * length(indices)) + 1):length(indi
ces)]

train_set <- insurance[train_indices, ]
validation_set <- insurance[validation_indices, ]
test_set <- insurance[test_indices, ]

# Parameters for depth and learning rates to try
depths <- 1:5
learning_rates <- seq(0.01, 0.20, by=0.01)

# Initialize variables to store the best parameters and the lowest RMSE
best_rmse <- Inf
best_depth <- NULL
best_learning_rate <- NULL

# Evaluate different depths and learning rates without creating a grid
for (depth in depths) {
  for (learning_rate in learning_rates) {
    boosted_models <- boosted_learn(rpart(charges ~ ., data=train_set,
control=rpart.control(maxdepth=depth)), train_set, outcome = "charges",
 n = 100, rate = learning_rate)
    predictions <- boosted_predict(boosted_models, validation_set)
    current_rmse <- rmse_out(actuals = validation_set$charges, preds =
predictions)
    cat("Depth:",depth, "- Learning rate:", learning_rate, " - RMSE on
validation set:", current_rmse, "\n" )

    if (current_rmse < best_rmse) {
      best_rmse <- current_rmse
      best_depth <- depth
      best_learning_rate <- learning_rate
    }
  }
}

# Print the combination of depth and learning rate that generate least
Rmse
cat("The best-case scenario:","\n")
cat("Depth", best_depth, "- Learning rate:", best_learning_rate, " - RM
SE on validation set:", best_rmse, "\n")

# Train final model with the best parameters
combined_train_val_set <- rbind(train_set, validation_set)
final_boosted_models <- boosted_learn(rpart(charges ~ ., data=combined_
train_val_set, control=rpart.control(maxdepth=best_depth)), combined_tr
ain_val_set, outcome = "charges", n = 100, rate = best_learning_rate)
final_predictions <- boosted_predict(final_boosted_models, test_set)
```

```r
# Calculate RMSE on the test set
final_rmse <- rmse_out(actuals = test_set$charges, preds = final_predic
tions)
cat("Final RMSE:", final_rmse)
```

```
## Depth: 1 - Learning rate: 0.01  - RMSE on validation set: 9916.116
## Depth: 1 - Learning rate: 0.02  - RMSE on validation set: 7449.005
## Depth: 1 - Learning rate: 0.03  - RMSE on validation set: 6547.977
## Depth: 1 - Learning rate: 0.04  - RMSE on validation set: 6113.695
## Depth: 1 - Learning rate: 0.05  - RMSE on validation set: 5867.319
## Depth: 1 - Learning rate: 0.06  - RMSE on validation set: 5720.481
## Depth: 1 - Learning rate: 0.07  - RMSE on validation set: 5633.878
## Depth: 1 - Learning rate: 0.08  - RMSE on validation set: 5619.856
## Depth: 1 - Learning rate: 0.09  - RMSE on validation set: 5618.571
## Depth: 1 - Learning rate: 0.1  - RMSE on validation set: 5616.362
## Depth: 1 - Learning rate: 0.11  - RMSE on validation set: 5621.957
## Depth: 1 - Learning rate: 0.12  - RMSE on validation set: 5606.98
## Depth: 1 - Learning rate: 0.13  - RMSE on validation set: 5627.464
## Depth: 1 - Learning rate: 0.14  - RMSE on validation set: 5596.331
## Depth: 1 - Learning rate: 0.15  - RMSE on validation set: 5620.682
## Depth: 1 - Learning rate: 0.16  - RMSE on validation set: 5624.346
## Depth: 1 - Learning rate: 0.17  - RMSE on validation set: 5624.511
## Depth: 1 - Learning rate: 0.18  - RMSE on validation set: 5627.572
## Depth: 1 - Learning rate: 0.19  - RMSE on validation set: 5586.895
## Depth: 1 - Learning rate: 0.2  - RMSE on validation set: 5614.141
## Depth: 2 - Learning rate: 0.01  - RMSE on validation set: 8041.857
## Depth: 2 - Learning rate: 0.02  - RMSE on validation set: 4986.729
## Depth: 2 - Learning rate: 0.03  - RMSE on validation set: 4292.526
## Depth: 2 - Learning rate: 0.04  - RMSE on validation set: 4117.771
## Depth: 2 - Learning rate: 0.05  - RMSE on validation set: 4096.442
## Depth: 2 - Learning rate: 0.06  - RMSE on validation set: 4109.131
## Depth: 2 - Learning rate: 0.07  - RMSE on validation set: 4103.851
## Depth: 2 - Learning rate: 0.08  - RMSE on validation set: 4091.626
## Depth: 2 - Learning rate: 0.09  - RMSE on validation set: 4113.629
## Depth: 2 - Learning rate: 0.1  - RMSE on validation set: 4069.701
## Depth: 2 - Learning rate: 0.11  - RMSE on validation set: 4092.275
## Depth: 2 - Learning rate: 0.12  - RMSE on validation set: 4081.29
## Depth: 2 - Learning rate: 0.13  - RMSE on validation set: 4087.231
## Depth: 2 - Learning rate: 0.14  - RMSE on validation set: 4078.528
## Depth: 2 - Learning rate: 0.15  - RMSE on validation set: 4078.839
## Depth: 2 - Learning rate: 0.16  - RMSE on validation set: 4061.914
## Depth: 2 - Learning rate: 0.17  - RMSE on validation set: 4078.763
## Depth: 2 - Learning rate: 0.18  - RMSE on validation set: 4053.241
## Depth: 2 - Learning rate: 0.19  - RMSE on validation set: 4056.336
## Depth: 2 - Learning rate: 0.2  - RMSE on validation set: 4032.879
## Depth: 3 - Learning rate: 0.01  - RMSE on validation set: 7770.781
## Depth: 3 - Learning rate: 0.02  - RMSE on validation set: 4721.327
## Depth: 3 - Learning rate: 0.03  - RMSE on validation set: 4135.136
## Depth: 3 - Learning rate: 0.04  - RMSE on validation set: 4059.653
## Depth: 3 - Learning rate: 0.05  - RMSE on validation set: 4050.223
```

```
## Depth: 3 - Learning rate: 0.06  - RMSE on validation set: 4045.674
## Depth: 3 - Learning rate: 0.07  - RMSE on validation set: 4045.319
## Depth: 3 - Learning rate: 0.08  - RMSE on validation set: 4044.905
## Depth: 3 - Learning rate: 0.09  - RMSE on validation set: 4051.289
## Depth: 3 - Learning rate: 0.1  - RMSE on validation set: 4057.054
## Depth: 3 - Learning rate: 0.11  - RMSE on validation set: 4044.805
## Depth: 3 - Learning rate: 0.12  - RMSE on validation set: 4061.455
## Depth: 3 - Learning rate: 0.13  - RMSE on validation set: 4052.327
## Depth: 3 - Learning rate: 0.14  - RMSE on validation set: 4058.323
## Depth: 3 - Learning rate: 0.15  - RMSE on validation set: 4039.375
## Depth: 3 - Learning rate: 0.16  - RMSE on validation set: 4034.273
## Depth: 3 - Learning rate: 0.17  - RMSE on validation set: 4029.215
## Depth: 3 - Learning rate: 0.18  - RMSE on validation set: 4062.935
## Depth: 3 - Learning rate: 0.19  - RMSE on validation set: 4058.149
## Depth: 3 - Learning rate: 0.2  - RMSE on validation set: 4054.489
## Depth: 4 - Learning rate: 0.01  - RMSE on validation set: 7770.344
## Depth: 4 - Learning rate: 0.02  - RMSE on validation set: 4704.584
## Depth: 4 - Learning rate: 0.03  - RMSE on validation set: 4128.354
## Depth: 4 - Learning rate: 0.04  - RMSE on validation set: 4055.154
## Depth: 4 - Learning rate: 0.05  - RMSE on validation set: 4037.592
## Depth: 4 - Learning rate: 0.06  - RMSE on validation set: 4044.377
## Depth: 4 - Learning rate: 0.07  - RMSE on validation set: 4037.081
## Depth: 4 - Learning rate: 0.08  - RMSE on validation set: 4038.843
## Depth: 4 - Learning rate: 0.09  - RMSE on validation set: 4049.321
## Depth: 4 - Learning rate: 0.1  - RMSE on validation set: 4040.719
## Depth: 4 - Learning rate: 0.11  - RMSE on validation set: 4029.917
## Depth: 4 - Learning rate: 0.12  - RMSE on validation set: 4055.023
## Depth: 4 - Learning rate: 0.13  - RMSE on validation set: 4047.856
## Depth: 4 - Learning rate: 0.14  - RMSE on validation set: 4054.146
## Depth: 4 - Learning rate: 0.15  - RMSE on validation set: 4023.236
## Depth: 4 - Learning rate: 0.16  - RMSE on validation set: 4030.982
## Depth: 4 - Learning rate: 0.17  - RMSE on validation set: 4018.415
## Depth: 4 - Learning rate: 0.18  - RMSE on validation set: 4046.296
## Depth: 4 - Learning rate: 0.19  - RMSE on validation set: 4018.843
## Depth: 4 - Learning rate: 0.2  - RMSE on validation set: 4021.532
## Depth: 5 - Learning rate: 0.01  - RMSE on validation set: 7770.344
## Depth: 5 - Learning rate: 0.02  - RMSE on validation set: 4704.53
## Depth: 5 - Learning rate: 0.03  - RMSE on validation set: 4121.607
## Depth: 5 - Learning rate: 0.04  - RMSE on validation set: 4051.572
## Depth: 5 - Learning rate: 0.05  - RMSE on validation set: 4047.995
## Depth: 5 - Learning rate: 0.06  - RMSE on validation set: 4037.193
## Depth: 5 - Learning rate: 0.07  - RMSE on validation set: 4035.667
## Depth: 5 - Learning rate: 0.08  - RMSE on validation set: 4038.859
## Depth: 5 - Learning rate: 0.09  - RMSE on validation set: 4042.981
## Depth: 5 - Learning rate: 0.1  - RMSE on validation set: 4040.401
## Depth: 5 - Learning rate: 0.11  - RMSE on validation set: 4040.952
## Depth: 5 - Learning rate: 0.12  - RMSE on validation set: 4032.155
## Depth: 5 - Learning rate: 0.13  - RMSE on validation set: 4055.909
## Depth: 5 - Learning rate: 0.14  - RMSE on validation set: 4049.084
## Depth: 5 - Learning rate: 0.15  - RMSE on validation set: 4050.107
```

```
## Depth: 5 - Learning rate: 0.16  - RMSE on validation set: 4044.888
## Depth: 5 - Learning rate: 0.17  - RMSE on validation set: 4017.53
## Depth: 5 - Learning rate: 0.18  - RMSE on validation set: 4035.443
## Depth: 5 - Learning rate: 0.19  - RMSE on validation set: 4022.801
## Depth: 5 - Learning rate: 0.2  - RMSE on validation set: 4024.684
## The best-case scenario:
## Depth 5 - Learning rate: 0.17  - RMSE on validation set: 4017.53
## Final RMSE: 5353.02
```