HW15

113078506

2025-05-31

- Gen AI Usage: I use Gen AI to refine my grammar, verify my reasoning and adjust to cleaner code.
- Students who helped: 113078505, 113078502, 113078514, and 113078158 helped with conclusion reasoning.

Question 1) Create some explanatory models to learn more about charges:

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
# Load dataset
insurance <- read.csv("insurance.csv")
insurance <- na.omit(insurance)
```

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

```
# Convert categorical variables to factors
insurance <- insurance |>
  mutate(sex = factor(sex),
         smoker = factor(smoker),
         region = factor(region))
ols_model <- lm(charges ~ age + sex + bmi + children + smoker + region, data = insurance)
summary(ols_model)
```

```
##
## Call:
## lm(formula = charges ~ age + sex + bmi + children + smoker +
##     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 ***
## sexmale           -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 ***
## smokeryes        23848.5      413.1  57.723  < 2e-16 ***
## regionnorthwest   -353.0      476.3  -0.741 0.458769
## regionsoutheast  -1035.0      478.7  -2.162 0.030782 *
## regionsouthwest   -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
```

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

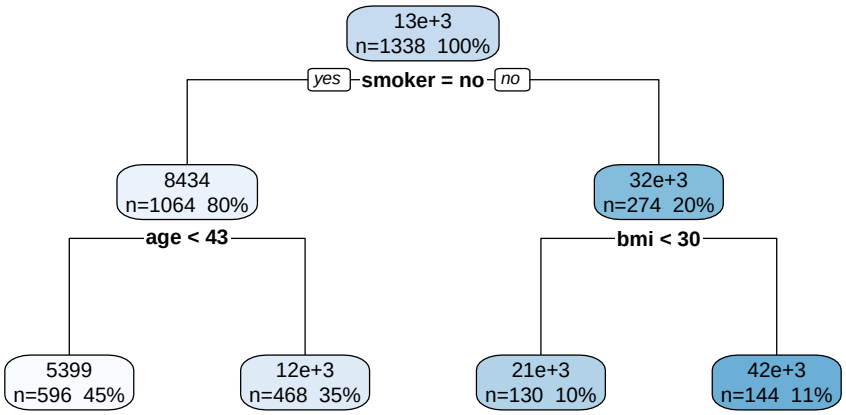
```
library(rpart)
tree_model <- rpart(charges ~ age + sex + bmi + children + smoker + region, data = insurance)
summary(tree_model)
```

```
## Call:
## rpart(formula = charges ~ age + sex + bmi + children + smoker +
##     region, data = insurance)
##   n= 1338
##
##          CP nsplit rel error    xerror       xstd
## 1 0.6197648      0 1.0000000 1.0008488 0.05186122
## 2 0.1439247      1 0.3802352 0.3818555 0.01896370
## 3 0.0636735      2 0.2363104 0.2395251 0.01452848
## 4 0.0100000      3 0.1726369 0.1778606 0.01325479
##
## Variable importance
## smoker    bmi    age region    sex
##     71     17      8      3      1
##
## Node number 1: 1338 observations,    complexity param=0.6197648
##   mean=13270.42, MSE=1.465428e+08
##   left son=2 (1064 obs) right son=3 (274 obs)
##   Primary splits:
##       smoker   splits as  LR,          improve=0.61976480, (0 missing)
##       age      < 42.5    to the left,  improve=0.07793137, (0 missing)
##       bmi      < 30.17   to the left,  improve=0.04212369, (0 missing)
##       children < 1.5     to the left,  improve=0.00831500, (0 missing)
```

```
##       region   splits as  LLRL,        improve=0.00547327, (0 missing)
##
## Node number 2: 1064 observations,    complexity param=0.0636735
##   mean=8434.268, MSE=3.589166e+07
##   left son=4 (596 obs) right son=5 (468 obs)
##   Primary splits:
##       age      < 42.5    to the left,  improve=0.326922000, (0 missing)
##       children < 1.5     to the left,  improve=0.019154040, (0 missing)
##       bmi      < 20.955  to the left,  improve=0.012695030, (0 missing)
##       region   splits as  RRLL,        improve=0.004790031, (0 missing)
##       sex      splits as  RL,          improve=0.003171961, (0 missing)
##   Surrogate splits:
##       bmi < 35.6325 to the left,  agree=0.58, adj=0.045, (0 split)
##
## Node number 3: 274 observations,    complexity param=0.1439247
##   mean=32050.23, MSE=1.327212e+08
##   left son=6 (130 obs) right son=7 (144 obs)
##   Primary splits:
##       bmi      < 30.01   to the left,  improve=0.776006300, (0 missing)
##       age      < 43.5    to the left,  improve=0.126767300, (0 missing)
##       region   splits as  LLRL,        improve=0.029264480, (0 missing)
##       sex      splits as  LR,          improve=0.010246720, (0 missing)
##       children < 1.5     to the left,  improve=0.003975908, (0 missing)
##   Surrogate splits:
##       region   splits as  LLRR,        agree=0.602, adj=0.162, (0 split)
##       sex      splits as  LR,          agree=0.566, adj=0.085, (0 split)
##       age      < 21.5    to the left,  agree=0.536, adj=0.023, (0 split)
##       children < 2.5     to the right, agree=0.536, adj=0.023, (0 split)
##
## Node number 4: 596 observations
##   mean=5398.85, MSE=2.214521e+07
##
## Node number 5: 468 observations
##   mean=12299.89, MSE=2.672104e+07
##
## Node number 6: 130 observations
##   mean=21369.22, MSE=2.528196e+07
##
## Node number 7: 144 observations
##   mean=41692.81, MSE=3.374313e+07
```

```r
library(rpart.plot)
rpart.plot(tree_model, type = 2, extra = 101, fallen.leaves = TRUE)
```

i. Plot a visual representation of the tree structure



ii. How deep is the tree (see nodes with "decisions" – ignore the leaves at the bottom) The tree starts with a split on the variable smoker, which forms the first level. From there, the left branch splits further on age, and the right branch splits on bmi, forming the second level of decisions. Since no further splits occur beyond these two levels, the tree has a maximum depth of 2. This means the longest path from the root to any leaf contains two decision nodes, making the tree relatively shallow and easy to interpret.

iii. How many leaf groups does it suggest to bin the data into? In the tree_model plot, the leaves are from left to right:

- smoker = yes & age $<$ 43 $\rightarrow$ Leaf: 5399
- smoker = yes & age 43 $\rightarrow$ Leaf: 12,000
- smoker = no & bmi $<$ 30 $\rightarrow$ Leaf: 21,000
- smoker = no & bmi 30 $\rightarrow$ Leaf: 42,000

Each leaf represents a bin or group of observations that share the same path of decision rules in the tree, so this tree_model suggest to bin the data into 4 groups.

iv. What conditions (combination of decisions) describe each leaf group? The conditions of each leaf are from left to right:

- For individuals who smoke and are younger than 43, the predicted insurance charges are approximately \$5,399.
- For individuals who smoke and are 43 or older, the predicted insurance charges are approximately \$12,000.
- For individuals who do not smoke and have a BMI less than 30, the predicted insurance charges are approximately \$21,000.
- For individuals who do not smoke and have a BMI of 30 or higher, the predicted insurance charges are approximately \$42,000.

Question 2) Let's use LOOCV to see how how our models perform predictively overall

```
# fold_i_pe
fold_i_pe <- function(i, k, model ,dataset, outcome) {
  fold_ids <- cut(1:nrow(dataset), breaks = k, labels = FALSE)
  test_indices <- which(fold_ids == 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
}


# k_fold_rmse
k_fold_rmse <- function(model, dataset, outcome, k = nrow(dataset)) {
  shuffled_indices <- sample(1:nrow(dataset))
  dataset <- dataset[shuffled_indices, ]
  fold_pred_errors <- sapply(1:k, function(i) {
    fold_i_pe(i, k, model, dataset, outcome)
  })
  pred_errors <- unlist(fold_pred_errors)
  sqrt(mean(pred_errors^2))
}
```

a. What is the RMSEout for the OLS regression model?

```
ols_model_rmse_out <- lm(charges ~ ., data = insurance)
ols_rmse_out <- k_fold_rmse(ols_model_rmse_out, insurance, "charges")
print(ols_rmse_out)
```

```
## [1] 6087.388
```

b. What is the RMSEout for the decision tree model?

```
tree_model_rmse_out <- rpart(charges ~ ., data = insurance)
tree_rmse_out <- k_fold_rmse(tree_model_rmse_out, insurance, "charges")
print(tree_rmse_out)
```

```
## [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.

```r
set.seed(123)
train_index <- sample(nrow(insurance), 0.8 * nrow(insurance))
train_data <- insurance[train_index, ]
test_data <- insurance[-train_index, ]
```

Question 3) Let's see if bagging helps our models

a. Implement the bagged_learn(...) and bagged_predict(...) functions using the hints in the class notes and help from your classmates on Teams. Feel free to share your code on Teams to get feedback, or ask others for help.

```r
# bagged_learn(…)
bagged_learn <- function(model_func, dataset, formula, b = 100) {
  lapply(1:b, function(i) {
    boot_sample <- dataset[sample(nrow(dataset), replace = TRUE), ]
    model_func(formula, data = boot_sample)
  })
}

# bagged_predict(…)
bagged_predict <- function(bagged_models, new_data) {
  predictions <- lapply(bagged_models, predict, newdata = new_data)
  prediction_matrix <- do.call(cbind, predictions)
  rowMeans(prediction_matrix)
}

#rmse
rmse_oos <- function(actual, preds) {
  mean((actual - preds)^2) |> sqrt()  # RMSE
}
```

b. What is the RMSEout for the bagged OLS regression?

```r
set.seed(123)
bagged_learn(lm, train_data, charges ~ age + sex + bmi + children + smoker + region, b = 100) |>
  bagged_predict(bagged_models = _, new_data = test_data) |>
  rmse_oos(actual = test_data$charges, preds = _)
```

```
## [1] 5769.604
```

c. What is the RMSEout for the bagged decision tree?

```r
set.seed(123)
bagged_learn(rpart, train_data, charges ~ age + sex + bmi + children + smoker + region, b = 100) |>
  bagged_predict(bagged_models = _, new_data = test_data) |>
  rmse_oos(actual = test_data$charges, preds = _)
```

```
## [1] 4902.21
```

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.

a. Write boosted_learn(...) and boosted_predict(...) functions using the hints in the class notes and help from your classmates on Teams. Feel free to share your code generously on Teams to get feedback, or ask others for help.

```
# boosted_learn(…)
boost_learn <- function(model, dataset, outcome, n = 100, rate = 0.1) {
  predictors <- dataset[, setdiff(names(dataset), outcome)]  # remove outcome column
  res <- dataset[[outcome]]  # start with the true y values (residuals = y at first)
  models <- list()

  for (i in 1:n) {
    # Fit model on current residuals
    this_model <- update(model, data = cbind(charges = res, predictors))  # reuse model structure
    pred <- predict(this_model, newdata = predictors)
    res <- res - rate * pred  # update residuals
    models[[i]] <- this_model
  }

  list(models = models, rate = rate)
}

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

  predictions <- lapply(boosted_models, function(m) {
    predict(m, newdata = new_data)
  })

  pred_frame <- as.data.frame(predictions) |> unname()
  apply(pred_frame, 1, function(row) sum(rate * row))
}
```

b. What is the RMSEout for the boosted OLS regression?

```
based_ols_model <- lm(charges ~ ., data = train_data)
boost_learn(based_ols_model, train_data, outcome="charges", n=1000) |>
boost_predict(test_data) |>
rmse_oos(test_data$charges, preds = _)
```

```
## [1] 5763.385
```

c. What is the RMSEout for the boosted decision tree?

```r
based_tree_model <- rpart(charges ~ ., data = train_data)
boost_learn(based_tree_model, train_data, outcome="charges", n=1000) |>
boost_predict(test_data) |>
rmse_oos(test_data$charges, preds = _)
```

```
## [1] 4435.98
```

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.

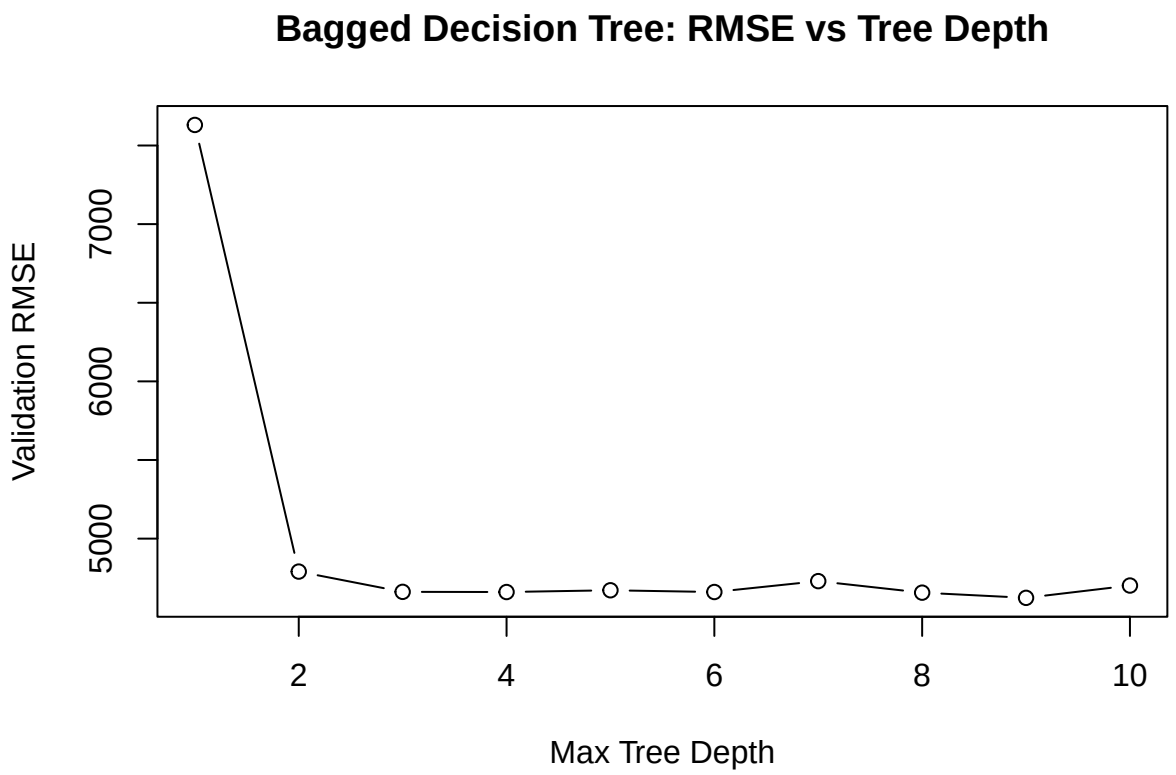
```r
set.seed(123)
split_train_index <- sample(nrow(insurance), 0.7 * nrow(insurance))
split_train_data <- insurance[split_train_index, ]
to_split_data <- insurance[-split_train_index, ]
split_validate_index <- sample(nrow(to_split_data), 0.5 * nrow(to_split_data))
split_validate_data <- to_split_data[split_validate_index, ]
split_test_data <- to_split_data[-split_validate_index, ]
```

a. 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
# create another parameter for bagged_learn "max_depth"
bagged_learn <- function(model_func, dataset, formula, b = 100,  maxdepth = 1) {
  lapply(1:b, function(i) {
    boot_sample <- dataset[sample(nrow(dataset), replace = TRUE), ]
    model_func(formula, data = boot_sample,
               control = rpart.control(maxdepth = maxdepth))
  })
}

# run from 1-10 depth, observe the rmse of validation set
rmse_by_depth <- sapply(1:10, function(d) {
  bagged_models <- bagged_learn(rpart, split_train_data,
                                formula = charges ~ age + sex + bmi + children + smoker + region,
                                b = 25,
                                maxdepth = d)
  preds <- bagged_predict(bagged_models, split_validate_data)
  rmse <- rmse_oos(split_validate_data$charges, preds)
  cat("Training bagged trees with depth =", d, "RMSE:",rmse,"\n")
  return(rmse)
})
```

```
## Training bagged trees with depth = 1 RMSE: 7630.702
## Training bagged trees with depth = 2 RMSE: 4790.365
## Training bagged trees with depth = 3 RMSE: 4661.447
## Training bagged trees with depth = 4 RMSE: 4660.353
## Training bagged trees with depth = 5 RMSE: 4671.545
## Training bagged trees with depth = 6 RMSE: 4660.532
## Training bagged trees with depth = 7 RMSE: 4729.584
## Training bagged trees with depth = 8 RMSE: 4656.467
## Training bagged trees with depth = 9 RMSE: 4623.371
## Training bagged trees with depth = 10 RMSE: 4701.589
```

```r
plot(1:10, rmse_by_depth, type = "b",
     xlab = "Max Tree Depth",
     ylab = "Validation RMSE",
     main = "Bagged Decision Tree: RMSE vs Tree Depth")
```




```r
set.seed(123)
best_depth <- which.min(rmse_by_depth)

final_models <- bagged_learn(rpart, split_train_data,
                             formula = charges ~ age + sex + bmi + children + smoker + region,
                             b = 25,
                             maxdepth = best_depth)

final_preds <- bagged_predict(final_models, split_test_data)
```

```r
final_rmse <- rmse_oos(split_test_data$charges, final_preds)

cat("best depth =", best_depth, "RMSE:",rmse_by_depth[best_depth],"\n")
```

```
## best depth = 9 RMSE: 4623.371
```

```r
cat("Final test RMSE =", round(final_rmse, 2), "\n")
```

```
## Final test RMSE = 4972.59
```

We used 15% of the data as a validation set to select the best max tree depth via RMSE. The final model was then evaluated on a held-out test set. As expected, the test RMSE is slightly higher than the validation RMSE due to no tuning being performed on the test set. This confirms that the model generalizes reasonably well without overfitting to the validation data.

b. 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.

```r
# create another parameter for boost_learn "max_depth"
boost_learn <- function(model, dataset, outcome, n = 100, rate = 0.1, maxdepth = 1) {
  predictors <- dataset[, setdiff(names(dataset), outcome)]  # remove outcome column
  res <- dataset[[outcome]]  # start with the true y values (residuals = y at first)
  models <- list()

  for (i in 1:n) {
    # Fit model on current residuals
    this_model <- rpart(residual ~ ., data = cbind(residual = res, predictors),
                        control = rpart.control(maxdepth = maxdepth))  # reuse model structure
    pred <- predict(this_model, newdata = predictors)
    res <- res - rate * pred  # update residuals
    models[[i]] <- this_model
  }

  list(models = models, rate = rate)
}

# Grid Search
depth_range <- 1:5
rate_range <- seq(0.01, 0.2, by = 0.03)

results <- expand.grid(depth = depth_range, rate = rate_range)
results$val_rmse <- apply(results, 1, function(row) {
  d <- as.numeric(row["depth"])
  r <- as.numeric(row["rate"])
```

```
  boosted <- boost_learn(charges ~ ., dataset = split_train_data,
                         outcome = "charges", n = 100, rate = r, maxdepth = d)

  preds <- boost_predict(boosted, split_validate_data)
  rmse <- rmse_oos(split_validate_data$charges, preds)
  cat("Training boosted trees → depth:", d, "rate:", r,"RMSE:",rmse, "\n")
  return(rmse)
})
```

```
## Training boosted trees → depth: 1 rate: 0.01 RMSE: 9905.88
## Training boosted trees → depth: 2 rate: 0.01 RMSE: 8035.822
## Training boosted trees → depth: 3 rate: 0.01 RMSE: 7792.266
## Training boosted trees → depth: 4 rate: 0.01 RMSE: 7791.301
## Training boosted trees → depth: 5 rate: 0.01 RMSE: 7791.301
## Training boosted trees → depth: 1 rate: 0.04 RMSE: 6492.201
## Training boosted trees → depth: 2 rate: 0.04 RMSE: 4413.155
## Training boosted trees → depth: 3 rate: 0.04 RMSE: 4369.059
## Training boosted trees → depth: 4 rate: 0.04 RMSE: 4367.17
## Training boosted trees → depth: 5 rate: 0.04 RMSE: 4369.162
## Training boosted trees → depth: 1 rate: 0.07 RMSE: 6041.43
## Training boosted trees → depth: 2 rate: 0.07 RMSE: 4418.449
## Training boosted trees → depth: 3 rate: 0.07 RMSE: 4364.009
## Training boosted trees → depth: 4 rate: 0.07 RMSE: 4347.665
## Training boosted trees → depth: 5 rate: 0.07 RMSE: 4350.652
## Training boosted trees → depth: 1 rate: 0.1 RMSE: 6018.305
## Training boosted trees → depth: 2 rate: 0.1 RMSE: 4402.351
## Training boosted trees → depth: 3 rate: 0.1 RMSE: 4353.089
## Training boosted trees → depth: 4 rate: 0.1 RMSE: 4348.716
## Training boosted trees → depth: 5 rate: 0.1 RMSE: 4356.345
## Training boosted trees → depth: 1 rate: 0.13 RMSE: 6023.405
## Training boosted trees → depth: 2 rate: 0.13 RMSE: 4441.178
## Training boosted trees → depth: 3 rate: 0.13 RMSE: 4350.95
## Training boosted trees → depth: 4 rate: 0.13 RMSE: 4340.928
## Training boosted trees → depth: 5 rate: 0.13 RMSE: 4337.903
## Training boosted trees → depth: 1 rate: 0.16 RMSE: 6018.161
## Training boosted trees → depth: 2 rate: 0.16 RMSE: 4382.211
## Training boosted trees → depth: 3 rate: 0.16 RMSE: 4330.081
## Training boosted trees → depth: 4 rate: 0.16 RMSE: 4334.567
## Training boosted trees → depth: 5 rate: 0.16 RMSE: 4334.567
## Training boosted trees → depth: 1 rate: 0.19 RMSE: 6017.451
## Training boosted trees → depth: 2 rate: 0.19 RMSE: 4375.731
## Training boosted trees → depth: 3 rate: 0.19 RMSE: 4336.548
## Training boosted trees → depth: 4 rate: 0.19 RMSE: 4347.998
## Training boosted trees → depth: 5 rate: 0.19 RMSE: 4342.822
```

```
results$val_rmse
```

```
##  [1] 9905.880 8035.822 7792.266 7791.301 7791.301 6492.201 4413.155 4369.059
##  [9] 4367.170 4369.162 6041.430 4418.449 4364.009 4347.665 4350.652 6018.305
## [17] 4402.351 4353.089 4348.716 4356.345 6023.405 4441.178 4350.950 4340.928
## [25] 4337.903 6018.161 4382.211 4330.081 4334.567 4334.567 6017.451 4375.731
## [33] 4336.548 4347.998 4342.822
```

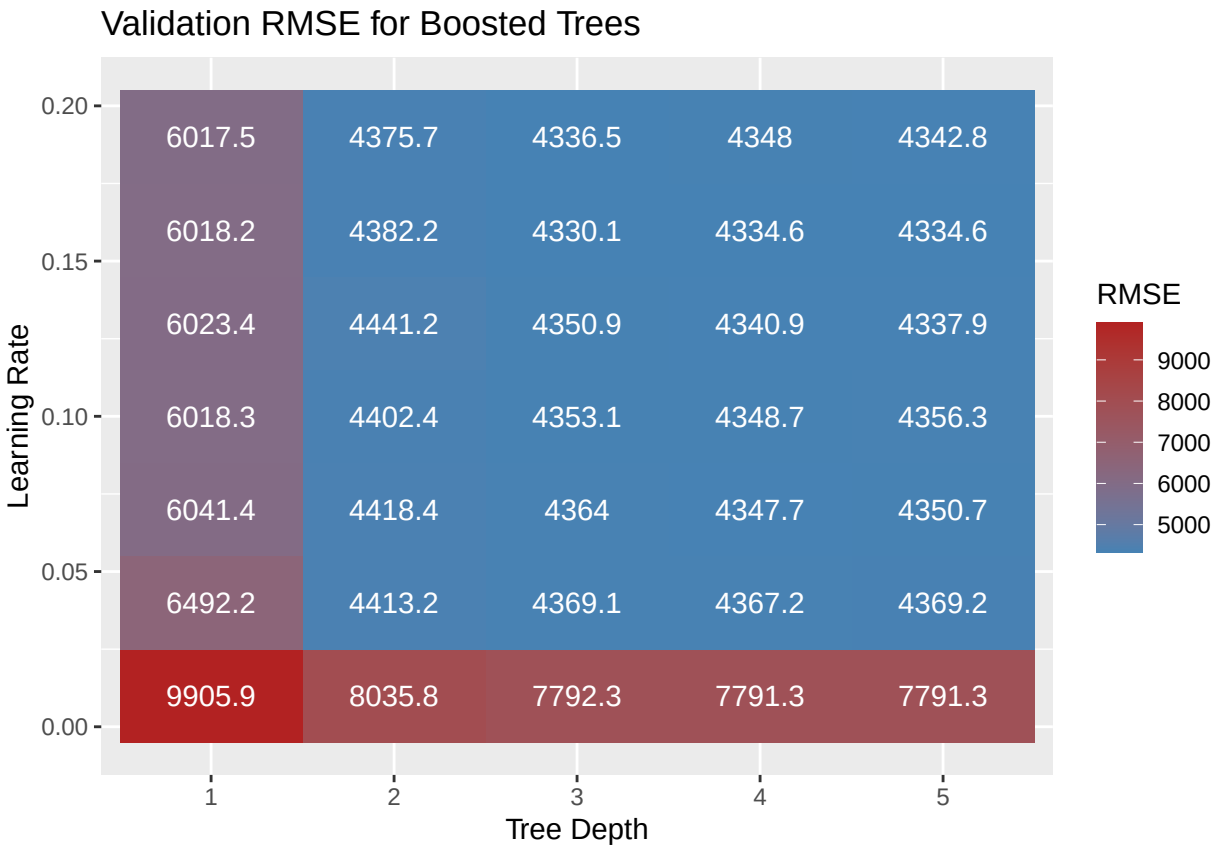
```r
set.seed(123)
best_idx <- which.min(results$val_rmse)
best_depth <- results$depth[best_idx]
best_rate <- results$rate[best_idx]

cat("Best combo → depth:", best_depth, "rate:", best_rate,"RMSE:",results$val_rmse[best_idx], "\n")
```

```
## Best combo → depth: 3 rate: 0.16 RMSE: 4330.081
```

```r
final_boosted <- boost_learn(charges ~ ., dataset = split_train_data,
                             outcome = "charges", n = 100,
                             rate = best_rate, maxdepth = best_depth)

final_preds <- boost_predict(final_boosted, split_test_data)
final_rmse <- rmse_oos(split_test_data$charges, final_preds)

cat("Final Test RMSE =", round(final_rmse, 2), "\n")
```

```
## Final Test RMSE = 4611.92
```

```r
library(ggplot2)

ggplot(results, aes(x = factor(depth), y = rate, fill = val_rmse)) +
  geom_tile() +
  geom_text(aes(label = round(val_rmse, 1)), color = "white") +
  scale_fill_gradient(low = "steelblue", high = "firebrick") +
  labs(title = "Validation RMSE for Boosted Trees",
       x = "Tree Depth", y = "Learning Rate", fill = "RMSE")
```



We used 15% of the data as a validation set to select the best combo of max tree depth and learning rate via RMSE. The final model was then evaluated on a held-out test set. As expected, the test RMSE is slightly higher than the validation RMSE due to no tuning being performed on the test set. This confirms that the model generalizes reasonably well without overfitting to the validation data.