ADM - assignment -2

Name – Anurodh Singh

KSUID – 811288773

Email – asingh68@kent.edu

QA1. What is the key idea behind bagging? Can bagging deal both with high variance (overfitting) and high bias (underfitting)?

Ans. - An ensemble learning method called bagging, or bootstrap aggregating, is intended to lower the variance of machine learning models, especially those that are prone to overfitting, like decision trees. Bootstrapping is the process of creating numerous subsets of the training data by sampling with replacement. A distinct model is then trained on each of these subsets, and the predictions are subsequently aggregated (either by voting in classification tasks or by average in regression tasks). Through the act of smoothing out variations brought on by noisy or highly particular patterns in the data, the model's output is stabilized.

Bagging **does not** automatically lessen bias, even though it is quite successful at lowering variance and preventing overfitting. Bagging won't increase the prediction power of the underlying model if it is too simplistic to identify patterns in the data (high bias or underfitting). Bagging is therefore very helpful when high variance is the main problem**, but it is not the best way** to cope with excessive bias in models.

QA2. Why bagging models are computationally more efficient when compared to boosting models with the same number of weak learners?

Ans.- Bagging models are computationally more efficient than boosting models with the same number of weak learners because of the way the individual learners are trained.   
Using several bootstrap samples from the training data, bagging trains each weak learner separately and concurrently. This implies that the training procedure can be split up among several threads or processors, greatly cutting down on calculation time. The models are independent of one another; they all learn at the same time and don't rely on one another's output or performance.

On the other hand, weak learners are gradually strengthened by boosting. Because it needs to learn from the mistakes of prior models, each new model must wait for the previous one to finish (i.e., it gives more weight to misclassified cases). This sequential dependency makes boosting less parallelizable than bagging and lengthens computation times, making it more computationally demanding overall.

QA3. James is thinking of creating an ensemble mode to predict whether a given stock will go up or down in the next week. He has trained several decision tree models but each model is not performing any better than a random model. The models are also very similar to each other. Do you think creating an ensemble model by combining these tree models can boost the

performance? Discuss your answer.

Ans. - **No**, performance is unlikely to be greatly improved by integrating multiple decision tree models that perform poorly but are very similar to create an ensemble model.  
The individual models (commonly referred to as weak learners) must ideally have some degree of variety and perform marginally better than random guessing in order for ensemble techniques like bagging or boosting to be successful. Combining decision trees won't offer much value if they are highly correlated (i.e., producing comparable errors) and each one is no better than a coin flip; instead, the ensemble will only make their weaknesses worse.

In James's scenario, the trees might all be learning the same patterns (or failing to learn relevant patterns) if they were all trained on comparable data or with the same parameters and features. In order to perform better, James ought to think about:

* Introducing **diversity** (e.g., different features, data samples, or tree depths),
* Using more informative features or feature engineering,
* Trying different base learners or **ensemble methods like random forests or gradient boosting**, which introduce randomness and learning from errors.

Without such strategies, simply averaging or voting among similar weak models won’t meaningfully improve predictive accuracy.

QA4. Consider the following Table that classifies some objects into two classes of edible (+) and non-edible (-), based on some characteristics such as the object color, size and shape. What would be the Information gain for splitting the dataset based on the “Size” attribute?

Ans. To calculate the **Information Gain (IG)** for splitting the dataset based on the **"Size"** attribute, follow these steps:

**Step 1: Count Total Instances and Classes**

We are classifying between **Edible (+)** and **Non-edible (-)**.

Let’s go through the data and tally:

**Total instances:** 15 Let’s count how many are Edible (+) and Non-edible (-):

* **Edible (+):** 9
* **Non-edible (-):** 6

**Step 2: Calculate Entropy of the Entire Dataset (Parent Entropy)**

We use the entropy formula:

Entropy(S)=−p(+) log2 (p(+) )−p(−) log2 (p(−) )

Where:

* p(+) = 9/15 = 0.6
* p(−) = 6/15 = 0.4

Entropy(S) = −(0.6\*log2(0.6)+0.4log2(0.4)) = −(0.6\*−0.737 + 0.4\*− 1.322) = 0.442 + 0.529 = 0.971 units

**Step 3: Split by “Size” Attribute**

Let’s see how many examples fall into each “Size”:

**Small:**

Instances = 6  
Check how many edible (+) and non-edible (−):

* Yellow Small Round ++
* Yellow Small Round −−
* Green Small Irregular ++
* Yellow Small Round ++
* Yellow Small Round ++
* Green Small Round −−

→ Total = 6  
→ Edible (+): 4  
→ Non-edible (−): 2

Entropy(Small)= −(4/6⋅log2 (4/6)+2/6⋅log2(2/6)) = −(0.667⋅−0.585+0.333⋅−1.585) = 0.390 + 0.528 = 0.918 units.

**Large:**

Instances = 9  
→ Check edible/non-edible:

* Green Large Irregular −−
* Yellow Large Round ++
* Yellow Large Round −−
* Yellow Large Round ++
* Yellow Large Round −−
* Yellow Large Round −−
* Yellow Large Round −−
* Yellow Large Irregular ++
* Yellow Large Irregular ++

→ Total = 9  
→ Edible (+): 5  
→ Non-edible (−): 4

Entropy(Large) = −(5/9⋅log2 (5/9) + 4/9⋅log2 (4/9)) = −(0.556⋅−0.848 + 0.444⋅−1.17) = 0.471 + 0.520 = 0.991 bits

**Step 4: Weighted Entropy After Splitting on “Size”:**

Entropy(Size) = (6/15) \* 0.918 + (9/15) \* 0.991 = 0.367 + 0.595 = 0.962 units.

**Step 5: Compute Information Gain:**

IG(Size) = Entropy(S) – Entropy(Size) = 0.971 − 0.962 = 0.009 units.

**Final Answer:**

The **Information Gain** for the **“Size”** attribute is **approximately 0.009 bits**.

QA5. Why is it important that the m parameter (number of attributes available at each split) to be optimally set in random forest models? Discuss the implications of setting this parameter too small or too large.

Ans. It is important that the m parameter (number of attributes available at each split) to be optimally set in random forest models because it directly affects the bias-variance trade-off and the variety among individual trees in the ensemble. Only a small number of attributes are considered at each split if m is set too small, increasing variety and randomness among trees and lowering variance and overfitting. But this can also result in weaker trees that might overlook significant patterns, which can lead to underfitting and increased bias. However, if m is set too big, each tree might employ more features, which can reduce bias but also diminish diversity, increasing the likelihood of overfitting and making the trees more identical.

Furthermore, the computational complexity rises with a bigger m. In order to guarantee that the Random Forest strikes a balance between model correctness, generalization ability, and computational efficiency, it is crucial to choose the ideal value for m.

QB1. Build a decision tree regression model to predict Sales based on all other attributes  
("Price", "Advertising", "Population", "Age", "Income" and "Education"). Which attribute is used  
at the top of the tree (the root node) for splitting? Hint: you can either plot () and text()  
functions or use the summary() function to see the decision tree rules.

QB2. Consider the following input:  
• Sales=9  
• Price=6.54  
• Population=124  
• Advertising=0  
• Age=76  
• Income= 110  
• Education=10  
What will be the estimated Sales for this record using the decision tree model?

QB3. Use the caret function to train a random forest (method=’rf’) for the same dataset. Use the  
caret default settings. By default, caret will examine the “mtry” values of 2,4, and 6. Recall that  
mtry is the number of attributes available for splitting at each splitting node. Which mtry value gives the best performance?  
(Make sure to set the random number generator seed to 123)

QB4. Customize the search grid by checking the model’s performance for mtry values of 2, 3 and 5 using 3 repeats of 5-fold cross validation.

2025-04-19

# Installing required packages   
#install.packages("ISLR2", repos = "http://cran.us.r-project.org") # for Carseats dataset  
#install.packages("rpart")  
#install.packages("rpart.plot")  
#install.packages("dplyr")  
#install.packages("caret")  
  
  
# Loading libraries  
library(ISLR2)  
library(rpart)  
library(rpart.plot)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

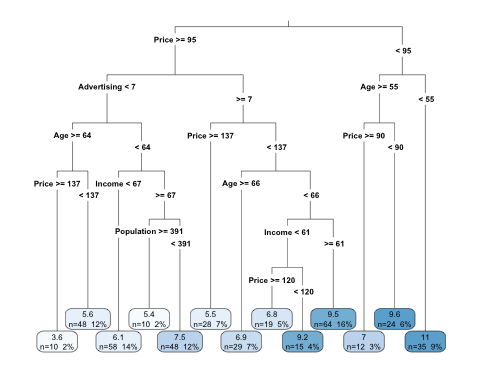
## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

# Filtering the dataset  
Carseats\_Filtered <- Carseats %>%  
 select("Sales", "Price", "Advertising", "Population", "Age", "Income", "Education")  
  
# Building the decision tree regression model  
set.seed(123)  
tree\_model <- rpart(Sales ~ ., data = Carseats\_Filtered, method = "anova")  
  
# Plotting the tree  
rpart.plot(tree\_model, type = 3, extra = 101, fallen.leaves = TRUE)



# Viewing summary of tree rules  
summary(tree\_model)

## Call:  
## rpart(formula = Sales ~ ., data = Carseats\_Filtered, method = "anova")  
## n= 400   
##   
## CP nsplit rel error xerror xstd  
## 1 0.14251535 0 1.0000000 1.0063165 0.06963484  
## 2 0.08034146 1 0.8574847 0.9159590 0.06531010  
## 3 0.06251702 2 0.7771432 0.8840071 0.06507510  
## 4 0.02925241 3 0.7146262 0.8264811 0.05974859  
## 5 0.02537341 4 0.6853738 0.8251007 0.05677601  
## 6 0.02127094 5 0.6600003 0.8329759 0.05626529  
## 7 0.02059174 6 0.6387294 0.8443029 0.05786670  
## 8 0.01632010 7 0.6181377 0.8420832 0.05750353  
## 9 0.01521801 8 0.6018176 0.8264536 0.05528371  
## 10 0.01042023 9 0.5865996 0.8551362 0.05717815  
## 11 0.01000559 10 0.5761793 0.8736311 0.05778555  
## 12 0.01000000 12 0.5561681 0.8858429 0.05833926  
##   
## Variable importance  
## Price Advertising Age Income Population Education   
## 49 18 16 8 6 3   
##   
## Node number 1: 400 observations, complexity param=0.1425153  
## mean=7.496325, MSE=7.955687   
## left son=2 (329 obs) right son=3 (71 obs)  
## Primary splits:  
## Price < 94.5 to the right, improve=0.14251530, (0 missing)  
## Advertising < 7.5 to the left, improve=0.07303226, (0 missing)  
## Age < 61.5 to the right, improve=0.07120203, (0 missing)  
## Income < 61.5 to the left, improve=0.02840494, (0 missing)  
## Population < 174.5 to the left, improve=0.01077467, (0 missing)  
##   
## Node number 2: 329 observations, complexity param=0.08034146  
## mean=7.001672, MSE=6.815199   
## left son=4 (174 obs) right son=5 (155 obs)  
## Primary splits:  
## Advertising < 6.5 to the left, improve=0.11402580, (0 missing)  
## Price < 136.5 to the right, improve=0.08411056, (0 missing)  
## Age < 63.5 to the right, improve=0.08091745, (0 missing)  
## Income < 60.5 to the left, improve=0.03394126, (0 missing)  
## Population < 23 to the left, improve=0.01831455, (0 missing)  
## Surrogate splits:  
## Population < 223 to the left, agree=0.599, adj=0.148, (0 split)  
## Education < 10.5 to the right, agree=0.565, adj=0.077, (0 split)  
## Age < 53.5 to the right, agree=0.547, adj=0.039, (0 split)  
## Income < 114.5 to the left, agree=0.547, adj=0.039, (0 split)  
## Price < 106.5 to the right, agree=0.544, adj=0.032, (0 split)  
##   
## Node number 3: 71 observations, complexity param=0.02537341  
## mean=9.788451, MSE=6.852836   
## left son=6 (36 obs) right son=7 (35 obs)  
## Primary splits:  
## Age < 54.5 to the right, improve=0.16595410, (0 missing)  
## Price < 75.5 to the right, improve=0.08365773, (0 missing)  
## Income < 30.5 to the left, improve=0.03322169, (0 missing)  
## Education < 10.5 to the right, improve=0.03019634, (0 missing)  
## Population < 268.5 to the left, improve=0.02383306, (0 missing)  
## Surrogate splits:  
## Advertising < 4.5 to the right, agree=0.606, adj=0.200, (0 split)  
## Price < 73 to the right, agree=0.592, adj=0.171, (0 split)  
## Population < 272.5 to the left, agree=0.592, adj=0.171, (0 split)  
## Income < 79.5 to the right, agree=0.592, adj=0.171, (0 split)  
## Education < 11.5 to the left, agree=0.577, adj=0.143, (0 split)  
##   
## Node number 4: 174 observations, complexity param=0.02127094  
## mean=6.169655, MSE=4.942347   
## left son=8 (58 obs) right son=9 (116 obs)  
## Primary splits:  
## Age < 63.5 to the right, improve=0.078712160, (0 missing)  
## Price < 130.5 to the right, improve=0.048919280, (0 missing)  
## Population < 26.5 to the left, improve=0.030421540, (0 missing)  
## Income < 67.5 to the left, improve=0.027749670, (0 missing)  
## Advertising < 0.5 to the left, improve=0.006795377, (0 missing)  
## Surrogate splits:  
## Income < 22.5 to the left, agree=0.678, adj=0.034, (0 split)  
## Price < 96.5 to the left, agree=0.672, adj=0.017, (0 split)  
## Population < 26.5 to the left, agree=0.672, adj=0.017, (0 split)  
##   
## Node number 5: 155 observations, complexity param=0.06251702  
## mean=7.935677, MSE=7.268151   
## left son=10 (28 obs) right son=11 (127 obs)  
## Primary splits:  
## Price < 136.5 to the right, improve=0.17659580, (0 missing)  
## Age < 73.5 to the right, improve=0.08000201, (0 missing)  
## Income < 60.5 to the left, improve=0.05360755, (0 missing)  
## Advertising < 13.5 to the left, improve=0.03920507, (0 missing)  
## Population < 399 to the left, improve=0.01037956, (0 missing)  
## Surrogate splits:  
## Advertising < 24.5 to the right, agree=0.826, adj=0.036, (0 split)  
##   
## Node number 6: 36 observations, complexity param=0.0163201  
## mean=8.736944, MSE=4.961043   
## left son=12 (12 obs) right son=13 (24 obs)  
## Primary splits:  
## Price < 89.5 to the right, improve=0.29079360, (0 missing)  
## Income < 39.5 to the left, improve=0.19043350, (0 missing)  
## Advertising < 11.5 to the left, improve=0.17891930, (0 missing)  
## Age < 75.5 to the right, improve=0.04316067, (0 missing)  
## Education < 14.5 to the left, improve=0.03411396, (0 missing)  
## Surrogate splits:  
## Advertising < 16.5 to the right, agree=0.722, adj=0.167, (0 split)  
## Income < 37.5 to the left, agree=0.722, adj=0.167, (0 split)  
## Age < 56.5 to the left, agree=0.694, adj=0.083, (0 split)  
##   
## Node number 7: 35 observations  
## mean=10.87, MSE=6.491674   
##   
## Node number 8: 58 observations, complexity param=0.01042023  
## mean=5.287586, MSE=3.93708   
## left son=16 (10 obs) right son=17 (48 obs)  
## Primary splits:  
## Price < 137 to the right, improve=0.14521540, (0 missing)  
## Education < 15.5 to the right, improve=0.07995394, (0 missing)  
## Income < 35.5 to the left, improve=0.04206708, (0 missing)  
## Age < 79.5 to the left, improve=0.02799057, (0 missing)  
## Population < 52.5 to the left, improve=0.01914342, (0 missing)  
##   
## Node number 9: 116 observations, complexity param=0.01000559  
## mean=6.61069, MSE=4.861446   
## left son=18 (58 obs) right son=19 (58 obs)  
## Primary splits:  
## Income < 67 to the left, improve=0.05085914, (0 missing)  
## Population < 392 to the right, improve=0.04476721, (0 missing)  
## Price < 127 to the right, improve=0.04210762, (0 missing)  
## Age < 37.5 to the right, improve=0.02858424, (0 missing)  
## Education < 14.5 to the left, improve=0.01187387, (0 missing)  
## Surrogate splits:  
## Education < 12.5 to the right, agree=0.586, adj=0.172, (0 split)  
## Age < 58.5 to the left, agree=0.578, adj=0.155, (0 split)  
## Price < 144.5 to the left, agree=0.569, adj=0.138, (0 split)  
## Population < 479 to the right, agree=0.560, adj=0.121, (0 split)  
## Advertising < 2.5 to the right, agree=0.543, adj=0.086, (0 split)  
##   
## Node number 10: 28 observations  
## mean=5.522857, MSE=5.084213   
##   
## Node number 11: 127 observations, complexity param=0.02925241  
## mean=8.467638, MSE=6.183142   
## left son=22 (29 obs) right son=23 (98 obs)  
## Primary splits:  
## Age < 65.5 to the right, improve=0.11854590, (0 missing)  
## Income < 51.5 to the left, improve=0.08076060, (0 missing)  
## Advertising < 13.5 to the left, improve=0.04801701, (0 missing)  
## Education < 11.5 to the right, improve=0.02471512, (0 missing)  
## Population < 479 to the left, improve=0.01908657, (0 missing)  
##   
## Node number 12: 12 observations  
## mean=7.038333, MSE=2.886964   
##   
## Node number 13: 24 observations  
## mean=9.58625, MSE=3.834123   
##   
## Node number 16: 10 observations  
## mean=3.631, MSE=5.690169   
##   
## Node number 17: 48 observations  
## mean=5.632708, MSE=2.88102   
##   
## Node number 18: 58 observations  
## mean=6.113448, MSE=3.739109   
##   
## Node number 19: 58 observations, complexity param=0.01000559  
## mean=7.107931, MSE=5.489285   
## left son=38 (10 obs) right son=39 (48 obs)  
## Primary splits:  
## Population < 390.5 to the right, improve=0.10993270, (0 missing)  
## Price < 124.5 to the right, improve=0.07534567, (0 missing)  
## Advertising < 0.5 to the left, improve=0.07060488, (0 missing)  
## Age < 45.5 to the right, improve=0.04611510, (0 missing)  
## Education < 11.5 to the right, improve=0.03722944, (0 missing)  
##   
## Node number 22: 29 observations  
## mean=6.893793, MSE=6.08343   
##   
## Node number 23: 98 observations, complexity param=0.02059174  
## mean=8.933367, MSE=5.262759   
## left son=46 (34 obs) right son=47 (64 obs)  
## Primary splits:  
## Income < 60.5 to the left, improve=0.12705480, (0 missing)  
## Advertising < 13.5 to the left, improve=0.07114001, (0 missing)  
## Price < 118.5 to the right, improve=0.06932216, (0 missing)  
## Education < 11.5 to the right, improve=0.03377416, (0 missing)  
## Age < 49.5 to the right, improve=0.02289004, (0 missing)  
## Surrogate splits:  
## Education < 17.5 to the right, agree=0.663, adj=0.029, (0 split)  
##   
## Node number 38: 10 observations  
## mean=5.406, MSE=2.508524   
##   
## Node number 39: 48 observations  
## mean=7.4625, MSE=5.381106   
##   
## Node number 46: 34 observations, complexity param=0.01521801  
## mean=7.811471, MSE=4.756548   
## left son=92 (19 obs) right son=93 (15 obs)  
## Primary splits:  
## Price < 119.5 to the right, improve=0.29945020, (0 missing)  
## Advertising < 11.5 to the left, improve=0.14268440, (0 missing)  
## Income < 40.5 to the right, improve=0.12781140, (0 missing)  
## Population < 152 to the left, improve=0.03601768, (0 missing)  
## Age < 49.5 to the right, improve=0.02748814, (0 missing)  
## Surrogate splits:  
## Education < 12.5 to the right, agree=0.676, adj=0.267, (0 split)  
## Advertising < 7.5 to the right, agree=0.647, adj=0.200, (0 split)  
## Age < 53.5 to the left, agree=0.647, adj=0.200, (0 split)  
## Population < 240 to the right, agree=0.618, adj=0.133, (0 split)  
## Income < 41.5 to the right, agree=0.618, adj=0.133, (0 split)  
##   
## Node number 47: 64 observations  
## mean=9.529375, MSE=4.5078   
##   
## Node number 92: 19 observations  
## mean=6.751053, MSE=3.378915   
##   
## Node number 93: 15 observations  
## mean=9.154667, MSE=3.273025

# printing full tree structure  
print(tree\_model)

## n= 400   
##   
## node), split, n, deviance, yval  
## \* denotes terminal node  
##   
## 1) root 400 3182.27500 7.496325   
## 2) Price>=94.5 329 2242.20000 7.001672   
## 4) Advertising< 6.5 174 859.96840 6.169655   
## 8) Age>=63.5 58 228.35070 5.287586   
## 16) Price>=137 10 56.90169 3.631000 \*  
## 17) Price< 137 48 138.28890 5.632708 \*  
## 9) Age< 63.5 116 563.92770 6.610690   
## 18) Income< 67 58 216.86830 6.113448 \*  
## 19) Income>=67 58 318.37860 7.107931   
## 38) Population>=390.5 10 25.08524 5.406000 \*  
## 39) Population< 390.5 48 258.29310 7.462500 \*  
## 5) Advertising>=6.5 155 1126.56300 7.935677   
## 10) Price>=136.5 28 142.35800 5.522857 \*  
## 11) Price< 136.5 127 785.25910 8.467638   
## 22) Age>=65.5 29 176.41950 6.893793 \*  
## 23) Age< 65.5 98 515.75040 8.933367   
## 46) Income< 60.5 34 161.72260 7.811471   
## 92) Price>=119.5 19 64.19938 6.751053 \*  
## 93) Price< 119.5 15 49.09537 9.154667 \*  
## 47) Income>=60.5 64 288.49920 9.529375 \*  
## 3) Price< 94.5 71 486.55130 9.788451   
## 6) Age>=54.5 36 178.59760 8.736944   
## 12) Price>=89.5 12 34.64357 7.038333 \*  
## 13) Price< 89.5 24 92.01896 9.586250 \*  
## 7) Age< 54.5 35 227.20860 10.870000 \*

new\_data <- data.frame(  
 Price = 6.54,  
 Advertising = 0,  
 Population = 124,  
 Age = 76,  
 Income = 110,  
 Education = 10  
)  
  
# Predicting Sales  
predicted\_sales <- predict(tree\_model, new\_data)  
  
print(paste("The estimated Sales for this record using the decision tree model would be:", predicted\_sales))

## [1] "The estimated Sales for this record using the decision tree model would be: 9.58625"

# Loading extra required libraries  
#install.packages("randomForest")  
library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

## The following object is masked from 'package:dplyr':  
##   
## combine

# Preparing the dataset (if not already)  
Carseats\_Filtered <- Carseats %>% select(Sales, Price, Advertising, Population, Age, Income, Education)  
  
# Setting the random seed for reproducibility as mentioned in the question.  
set.seed(123)  
  
# Training a Random Forest model using caret  
rf\_model <- train(  
 Sales ~ ., # Predicting Sales using all other variables  
 data = Carseats\_Filtered,  
 method = "rf", # Random forest method  
 trControl = trainControl(method = "cv", number = 5) # 5-fold cross-validation  
)  
  
# Viewing the model results  
print(rf\_model)

## Random Forest   
##   
## 400 samples  
## 6 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 320, 321, 319, 320, 320   
## Resampling results across tuning parameters:  
##   
## mtry RMSE Rsquared MAE   
## 2 2.406539 0.2838955 1.926998  
## 4 2.405609 0.2874877 1.916925  
## 6 2.415585 0.2834264 1.924429  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was mtry = 4.

print("Hence mtry = 4 gives the best performance, because it has the lowest RMSE (2.405609)")

## [1] "Hence mtry = 4 gives the best performance, because it has the lowest RMSE (2.405609)"

# Defining trainControl with 3 repeats of 5-fold cross-validation  
ctrl <- trainControl(method = "repeatedcv",  
 number = 5,  
 repeats = 3,  
 verboseIter = TRUE)  
  
# Creating custom tuning grid for mtry  
grid <- expand.grid(mtry = c(2, 3, 5))  
  
# Training the Random Forest model  
set.seed(123) # for reproducibility  
rf\_model <- train(Sales ~ .,  
 data = Carseats\_Filtered,  
 method = "rf",  
 metric = "RMSE",  
 trControl = ctrl,  
 tuneGrid = grid)

## + Fold1.Rep1: mtry=2   
## - Fold1.Rep1: mtry=2   
## + Fold1.Rep1: mtry=3   
## - Fold1.Rep1: mtry=3   
## + Fold1.Rep1: mtry=5   
## - Fold1.Rep1: mtry=5   
## + Fold2.Rep1: mtry=2   
## - Fold2.Rep1: mtry=2   
## + Fold2.Rep1: mtry=3   
## - Fold2.Rep1: mtry=3   
## + Fold2.Rep1: mtry=5   
## - Fold2.Rep1: mtry=5   
## + Fold3.Rep1: mtry=2   
## - Fold3.Rep1: mtry=2   
## + Fold3.Rep1: mtry=3   
## - Fold3.Rep1: mtry=3   
## + Fold3.Rep1: mtry=5   
## - Fold3.Rep1: mtry=5   
## + Fold4.Rep1: mtry=2   
## - Fold4.Rep1: mtry=2   
## + Fold4.Rep1: mtry=3   
## - Fold4.Rep1: mtry=3   
## + Fold4.Rep1: mtry=5   
## - Fold4.Rep1: mtry=5   
## + Fold5.Rep1: mtry=2   
## - Fold5.Rep1: mtry=2   
## + Fold5.Rep1: mtry=3   
## - Fold5.Rep1: mtry=3   
## + Fold5.Rep1: mtry=5   
## - Fold5.Rep1: mtry=5   
## + Fold1.Rep2: mtry=2   
## - Fold1.Rep2: mtry=2   
## + Fold1.Rep2: mtry=3   
## - Fold1.Rep2: mtry=3   
## + Fold1.Rep2: mtry=5   
## - Fold1.Rep2: mtry=5   
## + Fold2.Rep2: mtry=2   
## - Fold2.Rep2: mtry=2   
## + Fold2.Rep2: mtry=3   
## - Fold2.Rep2: mtry=3   
## + Fold2.Rep2: mtry=5   
## - Fold2.Rep2: mtry=5   
## + Fold3.Rep2: mtry=2   
## - Fold3.Rep2: mtry=2   
## + Fold3.Rep2: mtry=3   
## - Fold3.Rep2: mtry=3   
## + Fold3.Rep2: mtry=5   
## - Fold3.Rep2: mtry=5   
## + Fold4.Rep2: mtry=2   
## - Fold4.Rep2: mtry=2   
## + Fold4.Rep2: mtry=3   
## - Fold4.Rep2: mtry=3   
## + Fold4.Rep2: mtry=5   
## - Fold4.Rep2: mtry=5   
## + Fold5.Rep2: mtry=2   
## - Fold5.Rep2: mtry=2   
## + Fold5.Rep2: mtry=3   
## - Fold5.Rep2: mtry=3   
## + Fold5.Rep2: mtry=5   
## - Fold5.Rep2: mtry=5   
## + Fold1.Rep3: mtry=2   
## - Fold1.Rep3: mtry=2   
## + Fold1.Rep3: mtry=3   
## - Fold1.Rep3: mtry=3   
## + Fold1.Rep3: mtry=5   
## - Fold1.Rep3: mtry=5   
## + Fold2.Rep3: mtry=2   
## - Fold2.Rep3: mtry=2   
## + Fold2.Rep3: mtry=3   
## - Fold2.Rep3: mtry=3   
## + Fold2.Rep3: mtry=5   
## - Fold2.Rep3: mtry=5   
## + Fold3.Rep3: mtry=2   
## - Fold3.Rep3: mtry=2   
## + Fold3.Rep3: mtry=3   
## - Fold3.Rep3: mtry=3   
## + Fold3.Rep3: mtry=5   
## - Fold3.Rep3: mtry=5   
## + Fold4.Rep3: mtry=2   
## - Fold4.Rep3: mtry=2   
## + Fold4.Rep3: mtry=3   
## - Fold4.Rep3: mtry=3   
## + Fold4.Rep3: mtry=5   
## - Fold4.Rep3: mtry=5   
## + Fold5.Rep3: mtry=2   
## - Fold5.Rep3: mtry=2   
## + Fold5.Rep3: mtry=3   
## - Fold5.Rep3: mtry=3   
## + Fold5.Rep3: mtry=5   
## - Fold5.Rep3: mtry=5   
## Aggregating results  
## Selecting tuning parameters  
## Fitting mtry = 3 on full training set

# Viewing the results  
print(rf\_model)

## Random Forest   
##   
## 400 samples  
## 6 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold, repeated 3 times)   
## Summary of sample sizes: 320, 321, 319, 320, 320, 319, ...   
## Resampling results across tuning parameters:  
##   
## mtry RMSE Rsquared MAE   
## 2 2.405235 0.2813795 1.930855  
## 3 2.401365 0.2858295 1.920612  
## 5 2.417771 0.2821938 1.934886  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was mtry = 3.

plot(rf\_model)

