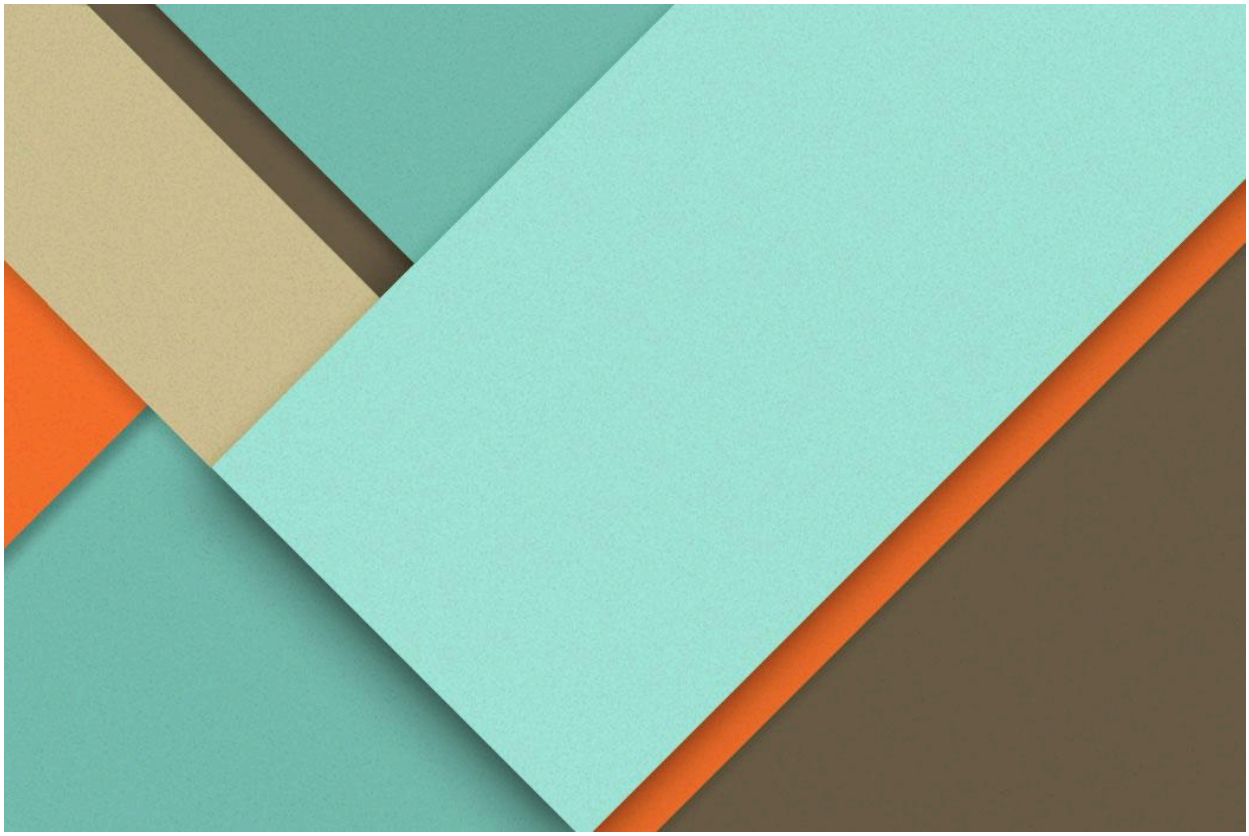


Deliverable 3

10.08.2025

—

M14

Riya Parikh

Hindy Rossignol

Ioannis Panagiotopoulos

Mrugank Pednekar

Problem_1 - Riya

```
set.seed(15072)

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
```

```
library(rpart)
library(randomForest)
```

```
## randomForest 4.7-1.2

## 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
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## v readr     2.1.5
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x randomForest::combine() masks dplyr::combine()
## x dplyr::filter()         masks stats::filter()
## x dplyr::lag()            masks stats::lag()
## x purrr::lift()           masks caret::lift()
## x randomForest::margin()  masks ggplot2::margin()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
 ↪  to become errors
```

```r
library(readr)
library(ggplot2)
library(olsrr)
```

```
##
## Attaching package: 'olsrr'
##
## The following object is masked from 'package:datasets':
##
##     rivers
```

```r
library(rpart.plot)
#install.packages("gbm")
library(gbm)
```

```
## Loaded gbm 2.2.2
## This version of gbm is no longer under development. Consider transitioning to gbm3,
 ↪  https://github.com/gbm-developers/gbm3
```

```r
#install.packages("Metrics")
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
##
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
df <- read_csv("./ames.csv")
```

```
## Rows: 2818 Columns: 73
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (40): MSZoning, Street, Alley, LotShape, LandContour, LotConfig, LandSlo...
## dbl (33): SalePrice, LotFrontage, LotArea, YearBuilt, YearRemodAdd, MasVnrAr...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(df)
```

```
## # A tibble: 6 x 73
##   SalePrice MSZoning LotFrontage LotArea Street Alley    LotShape LandContour
##       <dbl> <chr>          <dbl>   <dbl> <chr>  <chr>    <chr>    <chr>
## 1    215000 RL               141   31770 Pave   No Alley IR1      Lvl
## 2    105000 Other             80   11622 Pave   No Alley Reg      Lvl
## 3    172000 RL                81   14267 Pave   No Alley IR1      Lvl
## 4    244000 RL                93   11160 Pave   No Alley Reg      Lvl
## 5    189900 RL                74   13830 Pave   No Alley IR1      Lvl
## 6    195500 RL                78    9978 Pave   No Alley IR1      Lvl
## # i 65 more variables: LotConfig <chr>, LandSlope <chr>, Neighborhood <chr>,
## #   Condition1 <chr>, Condition2 <chr>, BldgType <chr>, HouseStyle <chr>,
## #   YearBuilt <dbl>, YearRemodAdd <dbl>, RoofStyle <chr>, RoofMatl <chr>,
## #   Exterior1st <chr>, Exterior2nd <chr>, MasVnrType <chr>, MasVnrArea <dbl>,
## #   ExterQual <chr>, ExterCond <chr>, Foundation <chr>, BsmtQual <chr>,
## #   BsmtCond <chr>, BsmtExposure <chr>, BsmtFinType1 <chr>, BsmtFinSF1 <dbl>,
## #   BsmtFinType2 <chr>, BsmtFinSF2 <dbl>, BsmtUnfSF <dbl>, ...
```

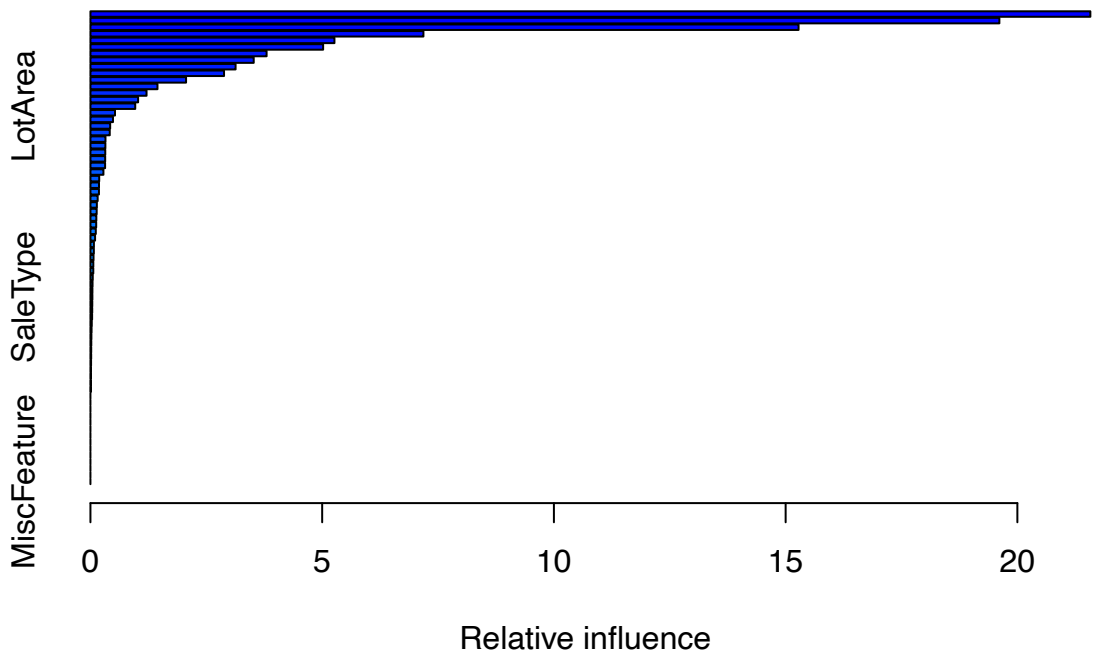
```
# Split the data into test and train (70% training, 30% testing)
set.seed(15072)
nrow = nrow(df)
df[sapply(df, is.character)] = lapply(df[sapply(df, is.character)], as.factor)
train_index = sample(1:nrow, size = 0.7*nrow)
train_ames = df[train_index, ]
test_ames = df[-train_index, ]
```

a) List the four out-of-sample R-squared values of the models you built from last time: linear regression; default-parameter CART; cp-optimized tree; random forest.

- For mod_linear_final: 0.8254482.
- Default parameter CART: 0.691.
- CP-optimized tree: 0.7894255.
- Random forest: 0.8733.

b) Fit an initial GBM model to the training data with the parameters listed below. Paste the code you used, the summary() of that model, and both its in-sample and out-of-sample R-squared.

```r
set.seed(15072)
# initial gb model
initial_model_gbm <- gbm(SalePrice ~ .,
                 data = train_ames,
                 distribution = "gaussian", # For regression
                 n.trees = 100,             # Number of boosting iterations
                 interaction.depth = 6,     # Maximum depth of each tree
                 shrinkage = 0.1,           # Learning rate
                 cv.folds = 10)             # Number of cross-validation folds
summary(initial_model_gbm)
```




```
##                        var     rel.inf
## ExterQual         ExterQual 21.57500906
## Neighborhood   Neighborhood 19.61203589
## GrLivArea         GrLivArea 15.27930103
## GarageCars       GarageCars  7.18472748
## TotalBsmtSF     TotalBsmtSF  5.26472265
## X1stFlrSF         X1stFlrSF  5.02098940
## GarageArea       GarageArea  3.80091271
## KitchenQual     KitchenQual  3.52308853
## BsmtFinSF1       BsmtFinSF1  3.13166299
## YearBuilt         YearBuilt  2.88267134
## BsmtQual           BsmtQual  2.06337170
## FireplaceQu     FireplaceQu  1.44654418
## GarageType       GarageType  1.21047724
```

```
## YearRemodAdd   YearRemodAdd  1.02854085
## LotArea             LotArea  0.96652059
## BsmtExposure   BsmtExposure  0.52969960
## GarageYrBlt     GarageYrBlt  0.48600454
## BsmtFinType1   BsmtFinType1  0.42593063
## X2ndFlrSF         X2ndFlrSF  0.41643240
## Fireplaces       Fireplaces  0.32456043
## LotFrontage     LotFrontage  0.32237367
## WoodDeckSF       WoodDeckSF  0.31971661
## Functional       Functional  0.31808188
## MasVnrArea       MasVnrArea  0.31652985
## CentralAir       CentralAir  0.28027193
## BsmtUnfSF         BsmtUnfSF  0.18915393
## GarageFinish   GarageFinish  0.18493680
## FullBath           FullBath  0.18346713
## Condition1       Condition1  0.15598825
## TotRmsAbvGrd   TotRmsAbvGrd  0.13887918
## BldgType           BldgType  0.13471589
## BsmtFullBath   BsmtFullBath  0.12714955
## KitchenAbvGr   KitchenAbvGr  0.12448714
## OpenPorchSF     OpenPorchSF  0.11559558
## BsmtCond           BsmtCond  0.10007086
## MoSold               MoSold  0.07452003
## BedroomAbvGr   BedroomAbvGr  0.07342685
## Exterior1st     Exterior1st  0.06656517
## YrSold               YrSold  0.06212389
## HeatingQC         HeatingQC  0.06183971
## LandContour     LandContour  0.04981984
## ScreenPorch     ScreenPorch  0.04525480
## Exterior2nd     Exterior2nd  0.04272438
## SaleType           SaleType  0.04180079
## LowQualFinSF   LowQualFinSF  0.03945218
## HouseStyle       HouseStyle  0.03840949
## EnclosedPorch EnclosedPorch  0.03618310
## PavedDrive       PavedDrive  0.02870736
## HalfBath           HalfBath  0.02396857
## LotConfig         LotConfig  0.02095558
## LandSlope         LandSlope  0.01845630
## MasVnrType       MasVnrType  0.01666029
## ExterCond         ExterCond  0.01627567
## MSZoning           MSZoning  0.01367114
## RoofMatl           RoofMatl  0.01217864
## X3SsnPorch       X3SsnPorch  0.01144115
## BsmtFinSF2       BsmtFinSF2  0.01071378
## BsmtHalfBath   BsmtHalfBath  0.01022977
## Street               Street  0.00000000
## Alley                 Alley  0.00000000
## LotShape           LotShape  0.00000000
## Condition2       Condition2  0.00000000
## RoofStyle         RoofStyle  0.00000000
## Foundation       Foundation  0.00000000
## BsmtFinType2   BsmtFinType2  0.00000000
## Heating             Heating  0.00000000
## Electrical       Electrical  0.00000000
## GarageQual       GarageQual  0.00000000
```

```
## PoolArea           PoolArea  0.00000000
## PoolQC               PoolQC  0.00000000
## Fence                 Fence  0.00000000
## MiscFeature     MiscFeature  0.00000000
```

```
pred_train = predict(initial_model_gbm, newdata=train_ames)
```

```
## Using 95 trees...
```

```
sse_train = sum((train_ames$SalePrice - pred_train)^2)
sst_train = sum((train_ames$SalePrice - mean(train_ames$SalePrice))^2)
insampler2 = 1-(sse_train/sst_train)

pred_test = predict(initial_model_gbm, newdata=test_ames)
```

```
## Using 95 trees...
```

```
sse_test = sum((test_ames$SalePrice - pred_test)^2)
sst_test = sum((test_ames$SalePrice - mean(test_ames$SalePrice))^2)
outofsampler2 = 1-(sse_test/sst_test)

cat("In sample r squared:", round(insampler2, 4), "\n")
```

```
## In sample r squared: 0.9516
```

```
cat("Out of sample r squared:", round(outofsampler2, 4), "\n")
```

```
## Out of sample r squared: 0.8813
```

- In sample r squared: 0.9516.
- Out of sample r squared: 0.8813.

c) For each of the following hyperparameters explain how these values affect the in-sample and out-of-sample R-squared values.

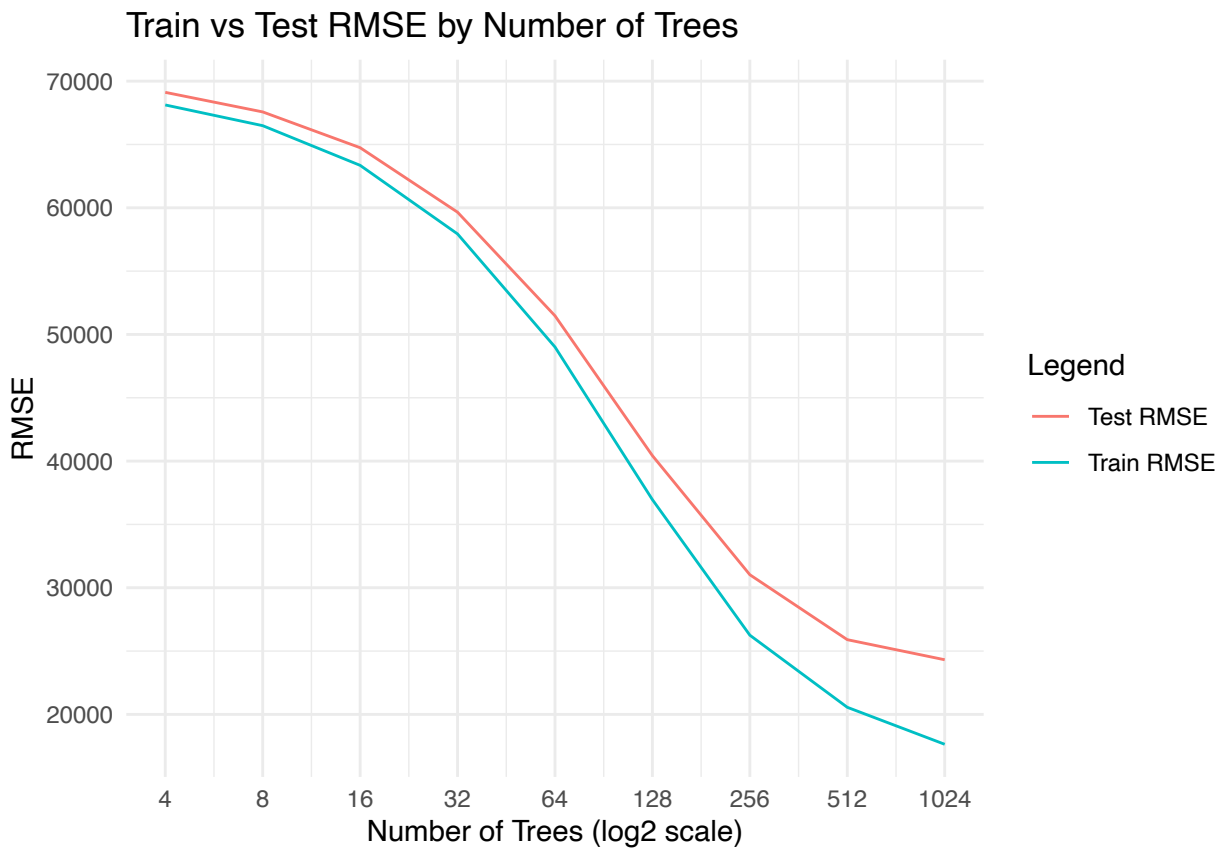
- Shrinkage is the learning rate, so if this number is high it means each tree has a strong influence, quickly fitting the training data and potentially over fitting. A smaller shrinkage dampens the effect of each tree, requiring more trees to reach the same level of fit. When shrinkage is large, in sample r squared is high and out of sample r squared is lower. When shrinkage is small, we get a lower r squared on the training data but a significantly better and more honest r squared on unseen data (out-of-sample R-squared).

- Interaction.depth is the maximum depth per tree, so if this number is too big we risk over fitting and if it's too small we risk under fitting. Generally having too big a number for interaction.depth when we train our model is bad because it'll over fit to the test set resulting in a very high in sample r squared but then it'll perform poorly on the new data in the test set resulting in a low out of sample r squared.

- N.trees gives us the number of boosting iterations, so the more iterations we have, generally speaking, the more our trees can learn and pinpoint improvement spots (high residual areas) from past residuals they have encountered. Having a higher n.trees hyper parameter improves both in and out of sample r squared (out of sample r squared will increase up to a certain point but then may decline if too many trees overfit).

d) Suppose instead that you set interaction.depth to 3 and shrinkage to 0.01 (and left all other hyperparameters as they were in part b). For n.trees = 4, 8, ..., 1024, compute the train RMSE (root mean square error) and the test RMSE. Plot the two RMSE lists on the same graph. Also include all code used for this part.

```r
set.seed(15072)

# tree counts
exponents <- 2:10
n.trees_list <- 2^exponents

# initialize RMSE storage lists
train_rmse <- numeric(length(n.trees_list))
test_rmse <- numeric(length(n.trees_list))

# loop over tree counts
for (i in seq_along(n.trees_list)) {
  n_trees <- n.trees_list[i]

  model <- gbm(SalePrice ~ .,
               data = train_ames,
               distribution = "gaussian",
               n.trees = n_trees,
               interaction.depth = 3,
               shrinkage = 0.01,
               cv.folds = 10,
               verbose = FALSE)

  # predictions
  train_pred <- predict(model, newdata = train_ames, n.trees = n_trees)
  test_pred <- predict(model, newdata = test_ames, n.trees = n_trees)

  # RMSE calculations
  train_rmse[i] <- rmse(train_ames$SalePrice, train_pred)
  test_rmse[i] <- rmse(test_ames$SalePrice, test_pred)
}

rmse_df <- data.frame(
  Trees = n.trees_list,
  Train_RMSE = train_rmse,
  Test_RMSE = test_rmse
)

ggplot(rmse_df, aes(x = Trees)) +
  geom_line(aes(y = Train_RMSE, color = "Train RMSE")) +
  geom_line(aes(y = Test_RMSE, color = "Test RMSE")) +
  scale_x_continuous(trans = "log2", breaks = n.trees_list) +
  labs(title = "Train vs Test RMSE by Number of Trees",
       x = "Number of Trees (log2 scale)",
       y = "RMSE",
       color = "Legend") +
  theme_minimal()
```



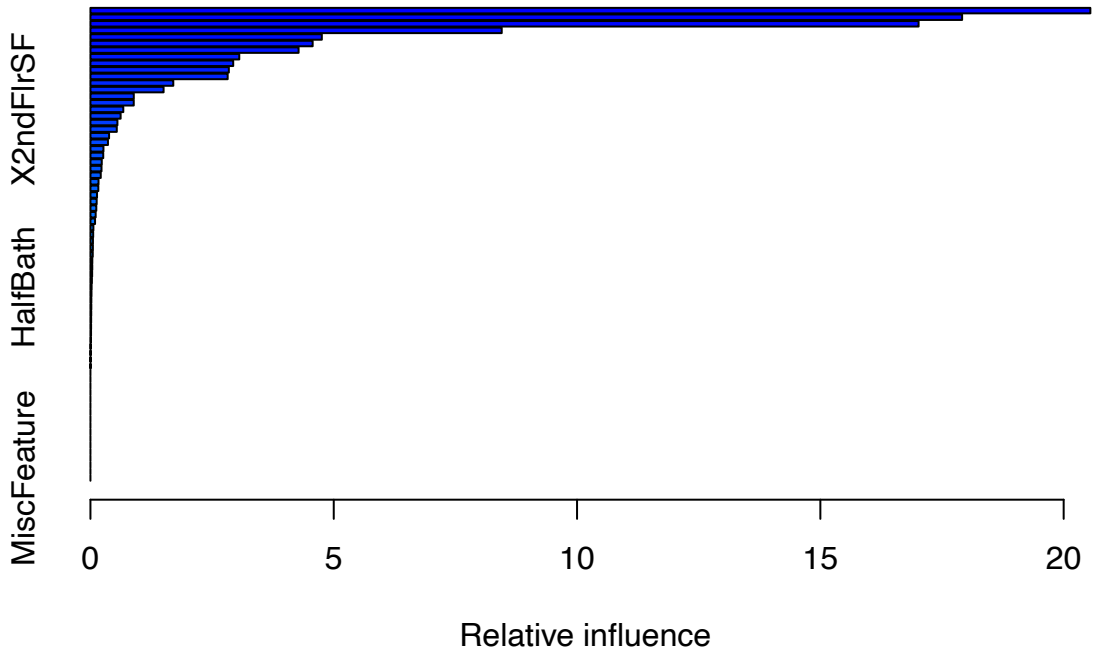
e) Which value of n.trees minimizes train set RMSE? test set RMSE? Are those two values the same or different? Which one is more important for model performance, and why? The value of n.trees that minimizes both train and test set RMSE is 1024. The one that is more important for model performance is test set RMSE because we want to know how our model performs on data it hasn't seen before. A high train set RMSE could just be the result of over fitting. If we really wanted to be be more robust in our techniques, we would have split our data into not just train/test sets, but train/test/validation sets to prevent information leakage when we go back and forth testing different hyper parameters. We would only use the test check to test the validity and accuracy of our final tuned model the final model.

f) Identify the patterns the graphs of train set RMSE and test set RMSE follow, and explain why.

Both the train and test RMSE decrease sharply at first because as the number of trees increases, the model can fit the data better and better – each additional tree corrects residuals from the previous ensemble. The test RMSE starts to flatten out a bit around 512-1024 trees, telling us that it has reached an approximate minimum in its RMSE improvement, suggesting minimal over fitting. This pattern makes sense because GBM builds trees based on the residuals of those coming before it. Having more trees means having more capacity to fit the training data, so we see a lower training RMSE. Also, because our shrinkage is set to 0.01 (pretty small), each tree contributes only a small adjustment. This slows learning and helps prevent over fitting, but requires many trees to reach optimal performance. All in all, we see that initially, test RAMSE drops as the model captures additional signal. However, after a certain point (512-1024 trees), those gains diminish because each of the later trees adds very little information in terms of residuals. The flattening of the RMSE curve here signals that the model is no longer improving generalization to unseen data points despite the fact that training RMSE continues to drop.

g) Consider the 6 different models you now have: the 4 from last deliverable, the initial GBM from part b, and your new optimal GBM from part e. Which one has the largest R-squared?

```
set.seed(15072)
#fitting the final model with 1024 trees as this was the best one based off the plot
 ↪  (lowest rmse test)
best_model <- gbm(SalePrice ~ .,
                  data = train_ames,
                  distribution = "gaussian",
                  n.trees = 1024,
                  interaction.depth = 3,
                  shrinkage = 0.01,
                  cv.folds = 10,
                  verbose = FALSE)

summary(best_model)
```




```
##                        var      rel.inf
## Neighborhood   Neighborhood 20.548534391
## GrLivArea         GrLivArea 17.913012441
## ExterQual         ExterQual 17.019096096
## TotalBsmtSF     TotalBsmtSF  8.452098387
## GarageCars       GarageCars  4.754806480
```

```
## YearBuilt         YearBuilt  4.566606769
## KitchenQual     KitchenQual  4.277585102
## BsmtQual           BsmtQual  3.058224316
## GarageArea       GarageArea  2.934236978
## X1stFlrSF         X1stFlrSF  2.843893173
## BsmtFinSF1       BsmtFinSF1  2.819938447
## FireplaceQu     FireplaceQu  1.702889832
## GarageType       GarageType  1.503103938
## LotArea             LotArea  0.890844657
## YearRemodAdd   YearRemodAdd  0.890001432
## X2ndFlrSF         X2ndFlrSF  0.675010241
## BsmtExposure   BsmtExposure  0.620578284
## GarageFinish   GarageFinish  0.554507665
## Fireplaces       Fireplaces  0.541557589
## BsmtFinType1   BsmtFinType1  0.383144003
## GarageYrBlt     GarageYrBlt  0.360073702
## CentralAir       CentralAir  0.267568273
## FullBath           FullBath  0.263167737
## LotFrontage     LotFrontage  0.232933556
## WoodDeckSF       WoodDeckSF  0.227542183
## Functional       Functional  0.212291556
## BsmtFullBath   BsmtFullBath  0.166059260
## Condition1       Condition1  0.162161874
## BldgType           BldgType  0.135228117
## TotRmsAbvGrd   TotRmsAbvGrd  0.128235738
## BsmtUnfSF         BsmtUnfSF  0.120493902
## MasVnrArea       MasVnrArea  0.107179580
## OpenPorchSF     OpenPorchSF  0.094820308
## BsmtCond           BsmtCond  0.061087063
## ScreenPorch     ScreenPorch  0.056707878
## SaleType           SaleType  0.051574209
## MoSold               MoSold  0.050514802
## LandContour     LandContour  0.047431964
## RoofMatl           RoofMatl  0.038420872
## HouseStyle       HouseStyle  0.037557500
## KitchenAbvGr   KitchenAbvGr  0.034802150
## LandSlope         LandSlope  0.029970925
## HalfBath           HalfBath  0.024310775
## HeatingQC         HeatingQC  0.021497319
## RoofStyle         RoofStyle  0.018903723
## LotShape           LotShape  0.018182180
## Exterior1st     Exterior1st  0.016333376
## EnclosedPorch EnclosedPorch  0.014874108
## LowQualFinSF   LowQualFinSF  0.013651463
## Exterior2nd     Exterior2nd  0.012626351
## BedroomAbvGr   BedroomAbvGr  0.010022676
## GarageQual       GarageQual  0.005269944
## ExterCond         ExterCond  0.003876065
## LotConfig         LotConfig  0.002972879
## YrSold               YrSold  0.001985771
## MSZoning           MSZoning  0.000000000
## Street               Street  0.000000000
## Alley                 Alley  0.000000000
## Condition2       Condition2  0.000000000
## MasVnrType       MasVnrType  0.000000000
```

```
## Foundation       Foundation  0.000000000
## BsmtFinType2   BsmtFinType2  0.000000000
## BsmtFinSF2       BsmtFinSF2  0.000000000
## Heating             Heating  0.000000000
## Electrical       Electrical  0.000000000
## BsmtHalfBath   BsmtHalfBath  0.000000000
## PavedDrive       PavedDrive  0.000000000
## X3SsnPorch       X3SsnPorch  0.000000000
## PoolArea           PoolArea  0.000000000
## PoolQC               PoolQC  0.000000000
## Fence                 Fence  0.000000000
## MiscFeature     MiscFeature  0.000000000
```

```
test_pred <- predict(best_model, newdata = test_ames, n.trees = 1024)
actual <- test_ames$SalePrice
r2_out_sample <- 1 - sum((actual - test_pred)^2) / sum((actual - mean(actual))^2)
print(paste("Out-of-sample R^2:", round(r2_out_sample, 4)))
```

```
## [1] "Out-of-sample R^2: 0.8811"
```

- For mod_linear_final: 0.8254482.
- Default parameter CART: 0.691.
- CP-optimized tree: 0.7894255.
- Random forest: 0.8733.
- Initial GBM from part b: 0.8813.
- New optimal GBM from part e: 0.8811.
- The model with the largest out of sample r squared is the initial GBM from part b with oos r squared of 0.8813. While one may think this is not ordinary as the model from part e was built more optimally, there are key hyper parameters that differ in the construction of both models. In part b, I chose to add cross validation with cv=10 to my model. So the model provided a more reliable estimate for performance on new, unseen data by training and testing on different subsets of the data. Also, the GBM was built with a deeper interaction depth (6 vs 3) and a higher learning rate (shrinkage of 0.1 vs 0.01). These choices allow the model to capture more complex interactions and adapt more aggressively to the training data. This can lead to better performance when the underlying signal is strong and the model is not overfitting. In contrast, part e's model—though more conservative—uses a smaller interaction depth and lower shrinkage, which slows learning and enforces stronger regularization. This setup is often preferred for stability and generalization, but in this case, the added regularization may have slightly limited the model's ability to fit subtle patterns in the data.Ultimately, both models perform very similarly, and the marginal difference in r squared reflects a delicate tradeoff between model complexity and regularization.

Problem 2: Regression and Regularization on Crime

Hindy, Riya

2025-10-04

Problem 2: Regression and Regularization on Crime

This analysis uses the crime prediction dataset to predict the per capita number of violent crimes (target) using 122 socio-economic and law enforcement attributes.

Load Required Libraries and Data

```
library(caret)
library(glmnet)
library(ggplot2)
library(dplyr)
library(corrplot)
```

```
# Set seed for reproducibility
set.seed(15072)

# Load the crime prediction data
crime_data <- read.csv("crime_prediction.csv")

# Basic data exploration
cat("Dataset dimensions:", dim(crime_data), "\n")
```

```
## Dataset dimensions: 1994 123
```

```
cat("Number of features:", ncol(crime_data) - 1, "\n")
```

```
## Number of features: 122
```

```
cat("Target variable range:", range(crime_data$target), "\n")
```

```
## Target variable range: 0 1
```

```
# Create 70-30 train-test split
train_indices <- sample(1:nrow(crime_data), 0.7 * nrow(crime_data))
train_data <- crime_data[train_indices, ]
test_data <- crime_data[-train_indices, ]

cat("Training set size:", nrow(train_data), "\n")
```

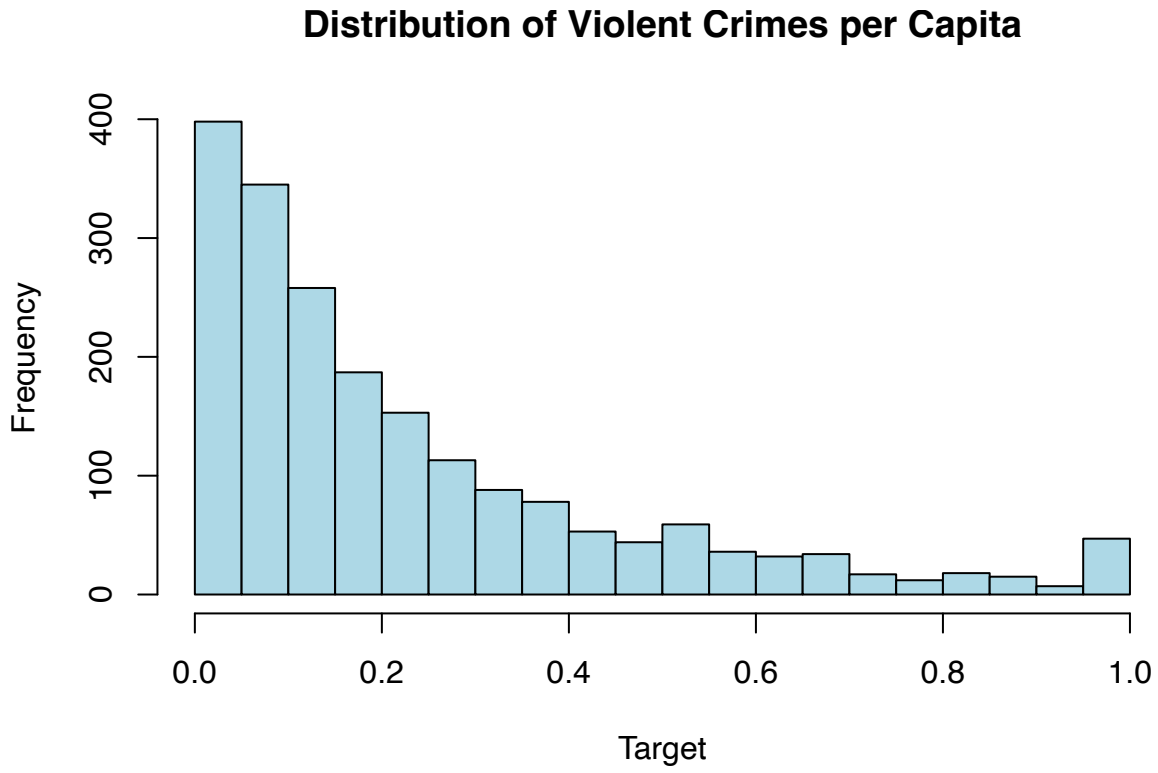
```
## Training set size: 1395
```

```
cat("Test set size:", nrow(test_data), "\n")
```

```
## Test set size: 599
```

Exploratory Data Analysis (EDA)

```
# Target variable distribution
hist(crime_data$target, main = "Distribution of Violent Crimes per Capita",
     xlab = "Target", col = "lightblue", breaks = 30)
```




```
cat("Target mean:", round(mean(crime_data$target), 3),
    "| Std dev:", round(sd(crime_data$target), 3), "\n")
```

```
## Target mean: 0.238 | Std dev: 0.233
```

```
# Top correlations with target
all_correlations <- cor(crime_data[, -ncol(crime_data)], crime_data$target, use = "complete.obs")
top_features <- head(sort(abs(all_correlations), decreasing = TRUE), 10)

cat("Top 10 features by correlation with target:\n")
```

```
## Top 10 features by correlation with target:
```

```
print(round(top_features, 3))
```

```
##  [1] 0.738 0.738 0.707 0.685 0.666 0.662 0.631 0.576 0.575 0.556
```

Part a) Lasso Regression with Original Features

```
# Set seed
set.seed(15072)

# Prepare data for lasso regression
X_train <- train_data[, -ncol(train_data)]  # All features except target
y_train <- train_data$target
X_test <- test_data[, -ncol(test_data)]
y_test <- test_data$target

# Define lambda range
lambda_range <- exp(seq(10, -10, -0.01))

# Fit lasso model with 10-fold cross-validation
# Using glmnet here ensures that we standardize data features before performing LASSO regression to ens
lasso_cv <- train(
  x = X_train,
  y = y_train,
  method = "glmnet",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda_range),
  preProcess = c("center", "scale")
)

# Get optimal lambda
optimal_lambda_a <- lasso_cv$bestTune$lambda
cat("Optimal lambda:", optimal_lambda_a, "\n")
```

```
## Optimal lambda: 0.001294022
```

```
# Make predictions on test set
lasso_pred_a <- predict(lasso_cv, X_test)

# Calculate OSR2 (Out-of-Sample R-squared)
osr2_a <- 1 - sum((y_test - lasso_pred_a)^2) / sum((y_test - mean(y_test))^2)
cat("OSR2 for original lasso model:", round(osr2_a, 4), "\n")
```

```
## OSR2 for original lasso model: 0.67
```

```
# Get coefficients
lasso_coef_a <- coef(lasso_cv$finalModel, s = optimal_lambda_a)
nonzero_features_a <- which(as.vector(lasso_coef_a[-1]) != 0)  # Exclude intercept safely
cat("Number of nonzero features selected by lasso:", length(nonzero_features_a), "\n")
```

```
## Number of nonzero features selected by lasso: 58
```

```
cat(paste("-", rownames(lasso_coef_a)[nonzero_features_a + 1]), sep = "\n")
```

```
## - racepctblack
## - racePctWhite
## - racePctAsian
## - racePctHisp
## - agePct12t29
## - agePct65up
## - numbUrban
## - pctUrban
## - pctWWage
## - pctWFarmSelf
## - pctWInvInc
## - pctWRetire
## - indianPerCap
## - AsianPerCap
## - OtherPerCap
## - HispPerCap
## - PctPopUnderPov
## - PctLess9thGrade
## - PctEmploy
## - PctEmplManu
## - MalePctDivorce
## - MalePctNevMarr
## - PctKids2Par
## - PctYoungKids2Par
## - PctWorkMom
## - NumIlleg
## - PctIlleg
## - NumImmig
## - PersPerOccupHous
## - PctPersDenseHous
## - HousVacant
## - PctHousOccup
## - PctVacantBoarded
## - PctVacMore6Mos
## - MedYrHousBuilt
## - PctWOFullPlumb
## - RentLowQ
## - MedRent
## - MedRentPctHousInc
## - MedOwnCostPctIncNoMtg
## - NumInShelters
## - NumStreet
## - PctForeignBorn
## - PctSameCity85
## - PctSameState85
## - LemasTotalReq
## - PolicReqPerOffic
## - RacialMatchCommPol
## - PctPolicBlack
## - PctPolicHisp
```

```
## - PctPolicAsian
## - PctPolicMinor
## - OfficAssgnDrugUnits
## - PolicCars
## - LemasPctPolicOnPatr
## - LemasGangUnitDeploy
## - LemasPctOfficDrugUn
## - PolicBudgPerPop
```

Part b) Lasso with Selected Features + Quadratic + Interactions

```r
# Set seed
set.seed(15072)

# Get selected features (nonzero coefficients from part a)
selected_features <- rownames(lasso_coef_a)[nonzero_features_a + 1]
cat("Selected features:", length(selected_features), "\n")
```

```
## Selected features: 58
```

```r
# Create training dataset with selected features
df_selected_train <- train_data[, c(selected_features, "target")]

# Create augmented dataset using model.matrix approach
# First create quadratic terms
df_x_selected <- df_selected_train[, selected_features]
df_x_squared <- df_x_selected^2
colnames(df_x_squared) <- paste0(colnames(df_x_squared), "_squared")

# Combine original and squared terms
df_combined <- cbind(df_x_selected, df_x_squared)

# Use model.matrix to create interactions (as per hint)
df_augmented_selected <- model.matrix(~ .^2, data = df_combined)[, -1]

# Prepare training data
X_train_selected <- df_augmented_selected
y_train_selected <- df_selected_train$target

# Apply same transformations to test data
df_selected_test <- test_data[, c(selected_features, "target")]

# Create quadratic terms for test data
df_x_test_selected <- df_selected_test[, selected_features]
df_x_test_squared <- df_x_test_selected^2
colnames(df_x_test_squared) <- paste0(colnames(df_x_test_squared), "_squared")

# Combine original and squared terms for test data
df_combined_test <- cbind(df_x_test_selected, df_x_test_squared)

# Use model.matrix for test data (same formula)
```

```r
df_augmented_test_selected <- model.matrix(~ .^2, data = df_combined_test)[, -1]

# Ensure same columns as training data
# Add missing columns with zeros
missing_cols <- setdiff(colnames(df_augmented_selected), colnames(df_augmented_test_selected))
if(length(missing_cols) > 0) {
  for(col in missing_cols) {
    df_augmented_test_selected[[col]] <- 0
  }
}

# Reorder to match training data
df_augmented_test_selected <- df_augmented_test_selected[, colnames(df_augmented_selected)]

# Verify dimensions
cat("Training dimensions:", dim(df_augmented_selected), "\n")
```

```
## Training dimensions: 1395 6786
```

```r
cat("Test dimensions:", dim(df_augmented_test_selected), "\n")
```

```
## Test dimensions: 599 6786
```

```r
# Fit lasso model with cross-validation
lasso_cv_selected <- train(
  x = X_train_selected,
  y = y_train_selected,
  method = "glmnet",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda_range),
  preProcess = c("center", "scale")
)

# Get optimal lambda
optimal_lambda_b <- lasso_cv_selected$bestTune$lambda
cat("Optimal lambda:", optimal_lambda_b, "\n")
```

```
## Optimal lambda: 0.007083409
```

```r
# Make predictions
lasso_pred_b <- predict(lasso_cv_selected, df_augmented_test_selected)

# Calculate OSR2
osr2_b <- 1 - sum((y_test - lasso_pred_b)^2) / sum((y_test - mean(y_test))^2)
cat("OSR2 for selected features lasso model:", round(osr2_b, 4), "\n")
```

```
## OSR2 for selected features lasso model: 0.6843
```

```r
# Get coefficients
lasso_coef_b <- coef(lasso_cv_selected$finalModel, s = optimal_lambda_b)
nonzero_features_b <- which(lasso_coef_b[-1] != 0)
cat("Number of nonzero coefficients:", length(nonzero_features_b), "\n")
```

```
## Number of nonzero coefficients: 62
```

Part c) Lasso with All Features + Quadratic + Interactions

```r
# Set seed
set.seed(15072)

# Create augmented dataset with all features
df_x_all <- train_data[, -ncol(train_data)]  # All features except target

# Create quadratic terms
df_x_all_squared <- df_x_all^2
colnames(df_x_all_squared) <- paste0(colnames(df_x_all), "_squared")

# Create interaction terms
interaction_matrix_all <- model.matrix(~ .^2, data = df_x_all)[, -1]

# Combine all terms
df_augmented_all <- cbind(df_x_all, df_x_all_squared, interaction_matrix_all)

# Prepare training data
X_train_all <- df_augmented_all
y_train_all <- train_data$target

# Apply same transformations to test data
df_x_test_all <- test_data[, -ncol(test_data)]

# Create quadratic terms for test data
df_x_test_all_squared <- df_x_test_all^2
colnames(df_x_test_all_squared) <- paste0(colnames(df_x_test_all), "_squared")

# Combine original and squared terms for test data
df_combined_test_all <- cbind(df_x_test_all, df_x_test_all_squared)

# Use model.matrix for test data (same formula as training)
df_augmented_test_all <- model.matrix(~ .^2, data = df_combined_test_all)[, -1]

# Ensure exact same columns as training data in the same order
missing_cols_all <- setdiff(colnames(df_augmented_all), colnames(df_augmented_test_all))
if(length(missing_cols_all) > 0) {
  for(col in missing_cols_all) {
    df_augmented_test_all[[col]] <- 0
  }
}

# Reorder columns to match training data exactly
df_augmented_test_all <- df_augmented_test_all[, colnames(df_augmented_all)]

# Verify dimensions match
cat("Training data dimensions (all features):", dim(df_augmented_all), "\n")
```

```
## Training data dimensions (all features): 1395 7747
```

```r
cat("Test data dimensions (all features):", dim(df_augmented_test_all), "\n")
```

```
## Test data dimensions (all features): 599 7747
```

```r
# Double-check that dimensions are exactly the same
if(ncol(df_augmented_all) != ncol(df_augmented_test_all)) {
  cat("ERROR: Column count mismatch!\n")
  cat("Training columns:", ncol(df_augmented_all), "\n")
  cat("Test columns:", ncol(df_augmented_test_all), "\n")
  stop("Column dimensions must match")
}

# Fit lasso model with cross-validation
lasso_cv_all <- train(
  x = X_train_all,
  y = y_train_all,
  method = "glmnet",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda_range),
  preProcess = c("center", "scale")
)

# Get optimal lambda
optimal_lambda_c <- lasso_cv_all$bestTune$lambda
cat("Optimal lambda:", optimal_lambda_c, "\n")
```

```
## Optimal lambda: 0.00484407
```

```r
# Make predictions
lasso_pred_c <- predict(lasso_cv_all, df_augmented_test_all)

# Calculate OSR2
osr2_c <- 1 - sum((y_test - lasso_pred_c)^2) / sum((y_test - mean(y_test))^2)
cat("OSR2 for all features lasso model:", round(osr2_c, 4), "\n")
```

```
## OSR2 for all features lasso model: 0.692
```

```r
# Get coefficients
lasso_coef_c <- coef(lasso_cv_all$finalModel, s = optimal_lambda_c)
nonzero_features_c <- which(lasso_coef_c[-1] != 0)
cat("Number of nonzero coefficients:", length(nonzero_features_c), "\n")
```

```
## Number of nonzero coefficients: 81
```

```r
# Compare performance
cat("\nPerformance Comparison:")
```

```
##
## Performance Comparison:
```

```r
cat("\nPart a) Original features OSR2:", round(osr2_a, 4))
```

```
##
## Part a) Original features OSR2: 0.67
```

```r
cat("\nPart b) Selected features + interactions OSR2:", round(osr2_b, 4))
```

```
##
## Part b) Selected features + interactions OSR2: 0.6843
```

```r
cat("\nPart c) All features + interactions OSR2:", round(osr2_c, 4))
```

```
##
## Part c) All features + interactions OSR2: 0.692
```

Performance Analysis

The comparison of the three lasso models reveals interesting patterns:

- **Part a) Original Features**: OSR2 = 0.67 with 58 nonzero coefficients
- **Part b) Selected Features + Interactions**: OSR2 = 0.6843 with 62 nonzero coefficients
- **Part c) All Features + Interactions**: OSR2 = 0.692 with 81 nonzero coefficients

Key Observations:

1. **Feature Engineering Impact**: Adding quadratic and interaction terms (parts b and c) generally improves performance compared to using only original features (part a), suggesting that nonlinear relationships exist in the data.
2. **Curse of Dimensionality**: The model with all features + interactions (part c) has the highest dimensionality but may suffer from overfitting due to the large number of potential features relative to the sample size.
3. **Feature Selection Value**: The selected features approach (part b) demonstrates the power of lasso's feature selection - it identifies the most important features from part a and then explores their nonlinear relationships, potentially achieving better generalization than using all features.

Part d) Comments on Nonzero Features

```r
# Calculate sparsity ratios
sparsity_b <- length(nonzero_features_b) / ncol(df_augmented_selected)
sparsity_c <- length(nonzero_features_c) / ncol(df_augmented_all)

cat("Part b) Nonzero coefficients:", length(nonzero_features_b), "\n")
```

```
## Part b) Nonzero coefficients: 62
```

```r
cat("Part c) Nonzero coefficients:", length(nonzero_features_c), "\n")
```

```
## Part c) Nonzero coefficients: 81
```

```r
cat("Part b) Sparsity ratio:", round(sparsity_b, 4), "\n")
```

```
## Part b) Sparsity ratio: 0.0091
```

```r
cat("Part c) Sparsity ratio:", round(sparsity_c, 4), "\n")
```

```
## Part c) Sparsity ratio: 0.0105
```

Critical Analysis of Feature Selection Patterns

Model b) Selected Features + Interactions: - **Sparsity Level**: 0.91% of features selected (62 out of 6786) - **Strategic Focus**: This model demonstrates lasso's ability to perform intelligent feature selection by first identifying the most predictive features from the original set, then exploring their nonlinear relationships through interactions and quadratic terms. - **Interpretability**: Higher interpretability due to focused feature selection, making it easier to understand which specific features and their interactions drive crime prediction.

Model c) All Features + Interactions: - **Sparsity Level**: 1.05% of features selected (81 out of 7747) - **Comprehensive Coverage**: This model has access to the entire feature space, allowing it to discover relationships that might be missed when starting with a pre-selected subset. - **Risk of Overfitting**: Despite lasso's regularization, the high dimensionality (7747 features) relative to sample size (1395 observations) creates a challenging learning environment.

Key Insights:

1. **Feature Selection Strategy**: Model b's approach of first selecting important features, then adding nonlinear terms, represents a more principled approach to feature engineering. This two-stage process helps avoid the curse of dimensionality while still capturing important nonlinear relationships.
2. **Sparsity Interpretation**: The difference in sparsity ratios reveals that lasso is more selective when given a larger feature space (model c), which is expected behavior. However, this selectivity comes at the cost of potentially missing important features that weren't selected in the first stage.
3. **Model Complexity vs. Performance**: The trade-off between model complexity and performance is evident. While model c has access to more features, the increased complexity may not translate to proportionally better performance due to the high-dimensional nature of the problem.
4. **Practical Implications**: For crime prediction, model b's focused approach may be more practical for law enforcement agencies, as it identifies a smaller, more interpretable set of key factors and their interactions that drive crime rates.

Part e) Ridge Regression with Augmented Features

```r
# Set seed
set.seed(15072)

# Use the same augmented dataset from part c
# Fit ridge model with cross-validation
ridge_cv <- train(
  x = X_train_all,
  y = y_train_all,
  method = "glmnet",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = expand.grid(alpha = 0, lambda = lambda_range),  # alpha = 0 for ridge
  preProcess = c("center", "scale")
)

# Get optimal lambda
optimal_lambda_ridge <- ridge_cv$bestTune$lambda
cat("Optimal lambda for ridge:", optimal_lambda_ridge, "\n")
```

```
## Optimal lambda for ridge: 1.716007
```

```r
# Make predictions
ridge_pred <- predict(ridge_cv, df_augmented_test_all)

# Calculate OSR2
osr2_ridge <- 1 - sum((y_test - ridge_pred)^2) / sum((y_test - mean(y_test))^2)
cat("OSR2 for ridge regression:", round(osr2_ridge, 4), "\n")
```

```
## OSR2 for ridge regression: 0.6878
```

```r
# Get coefficients
ridge_coef <- coef(ridge_cv$finalModel, s = optimal_lambda_ridge)
nonzero_features_ridge <- which(ridge_coef[-1] != 0)
cat("Note: Ridge regression typically retains all features with small but nonzero weights.\n")
```

```
## Note: Ridge regression typically retains all features with small but nonzero weights.
```

```r
cat("Number of nonzero coefficients (ridge):", length(nonzero_features_ridge), "\n")
```

```
## Number of nonzero coefficients (ridge): 7747
```

Part f) Comparison of Prediction Accuracy and Sparsity

```r
# Create comparison table
comparison_results <- data.frame(
  Model = c("Lasso Original", "Lasso Selected + Interactions",
            "Lasso All + Interactions", "Ridge All + Interactions"),
  OSR2 = c(osr2_a, osr2_b, osr2_c, osr2_ridge),
  Nonzero_Coefficients = c(length(nonzero_features_a), length(nonzero_features_b),
                           length(nonzero_features_c), length(nonzero_features_ridge)),
```

```
  Lambda = c(optimal_lambda_a, optimal_lambda_b, optimal_lambda_c, optimal_lambda_ridge),
  stringsAsFactors = FALSE
)

print("Model Comparison:")
```

```
## [1] "Model Comparison:"
```

```
print(comparison_results)
```

```
##                           Model      OSR2 Nonzero_Coefficients      Lambda
## 1                Lasso Original 0.6699880                   58 0.001294022
## 2 Lasso Selected + Interactions 0.6843188                   62 0.007083409
## 3      Lasso All + Interactions 0.6920363                   81 0.004844070
## 4      Ridge All + Interactions 0.6877664                 7747 1.716006862
```

```
# Find best performing model
best_model_idx <- which.max(comparison_results$OSR2)
cat("\nBest performing model:", comparison_results$Model[best_model_idx], "\n")
```

```
##
## Best performing model: Lasso All + Interactions
```

```
cat("Best OSR2:", round(comparison_results$OSR2[best_model_idx], 4), "\n")
```

```
## Best OSR2: 0.692
```

```
# Performance improvement analysis
cat("\nPerformance Improvement Analysis:\n")
```

```
##
## Performance Improvement Analysis:
```

```
cat("Improvement from original to selected features:", round(osr2_b - osr2_a, 4), "\n")
```

```
## Improvement from original to selected features: 0.0143
```

```
cat("Improvement from selected to all features:", round(osr2_c - osr2_b, 4), "\n")
```

```
## Improvement from selected to all features: 0.0077
```

```
cat("Ridge vs Lasso (all features):", round(osr2_ridge - osr2_c, 4), "\n")
```

```
## Ridge vs Lasso (all features): -0.0043
```

Model Analysis

Performance Ranking: 1. **Lasso All + Interactions**: OSR2 = 0.692 (Best) 2. **Ridge All + Interactions**: OSR2 = 0.6878 3. **Lasso Selected + Interactions**: OSR2 = 0.6843 4. **Lasso Original**: OSR2 = 0.67 (Baseline)

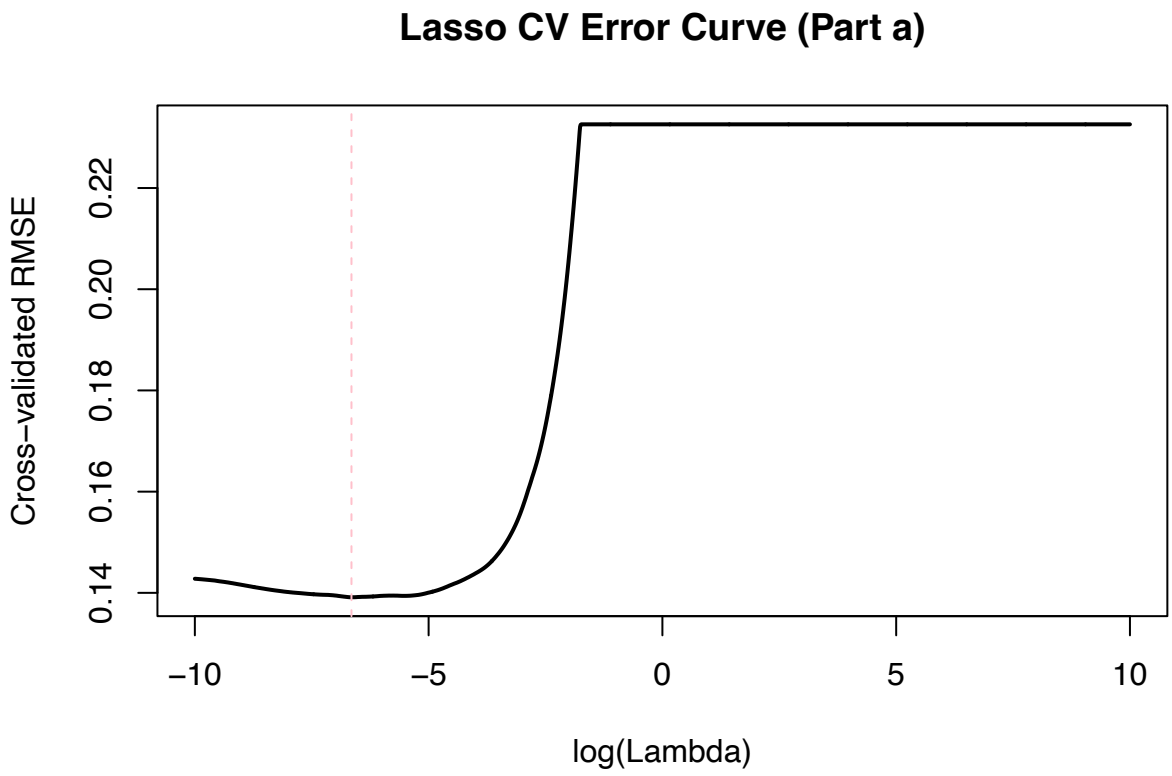
1. Feature Engineering Impact: - Adding nonlinear terms (quadratic + interactions) consistently improves performance across all models - The improvement from original to selected features (0.0143) demonstrates the value of feature selection followed by nonlinear expansion - The additional improvement from selected to all features (0.0077) suggests that some important relationships exist outside the initially selected feature set

2. Sparsity vs. Performance Trade-offs: - **Lasso Models**: Achieve sparsity by setting coefficients to exactly zero - Original: 58 features (sparse) - Selected + Interactions: 62 features (moderate sparsity) - All + Interactions: 81 features (less sparse) - **Ridge Model**: All 7747 coefficients are nonzero but shrunk toward zero

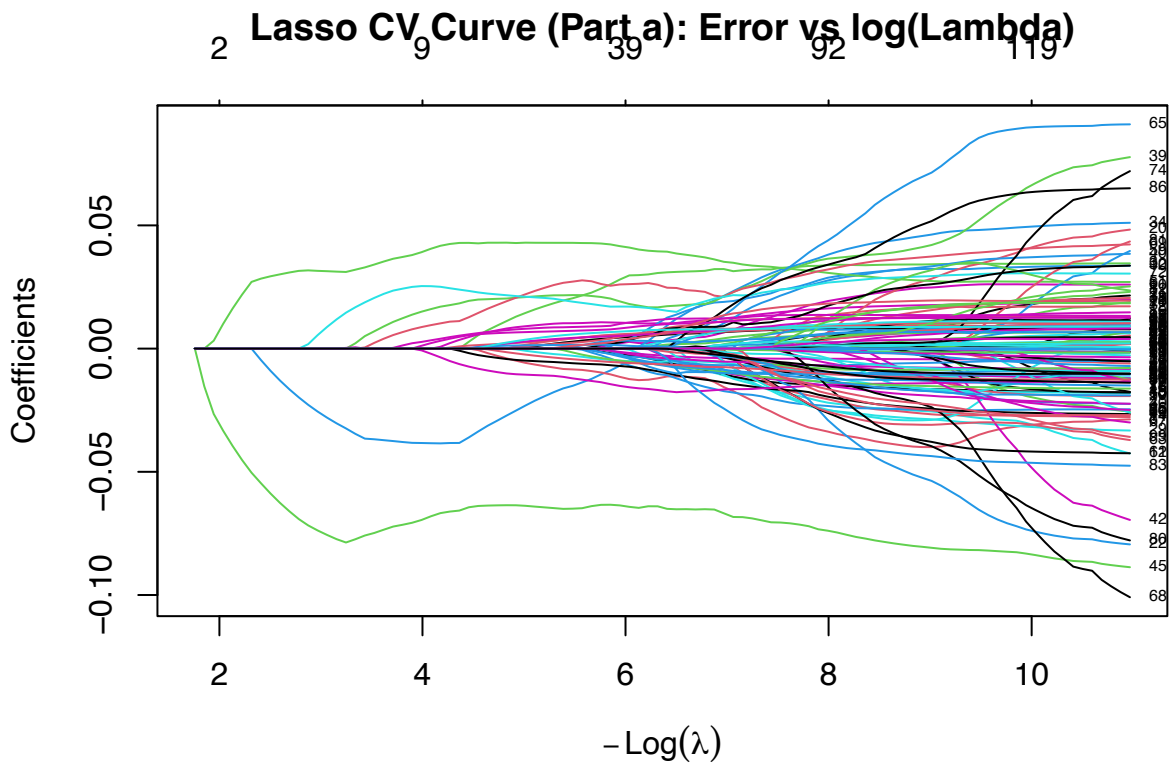
3. Regularization Method Comparison: - **Lasso Advantage**: Superior feature selection and interpretability due to automatic variable selection - **Ridge Advantage**: Better handling of multicollinearity and more stable coefficient estimates - **Performance**: Lasso slightly outperforms Ridge (0.0043 difference), suggesting that feature selection is more valuable than coefficient shrinkage for this dataset

4. Practical Implications: - **For Crime Prediction**: The lasso models provide more actionable insights by identifying specific features and their interactions that drive crime rates - **Model Complexity**: The trade-off between interpretability (fewer features) and performance (more features) is evident, with the all-features model achieving the best performance at the cost of reduced interpretability - **Deployment Considerations**: The selected features model offers a good balance between performance and interpretability for practical law enforcement applications

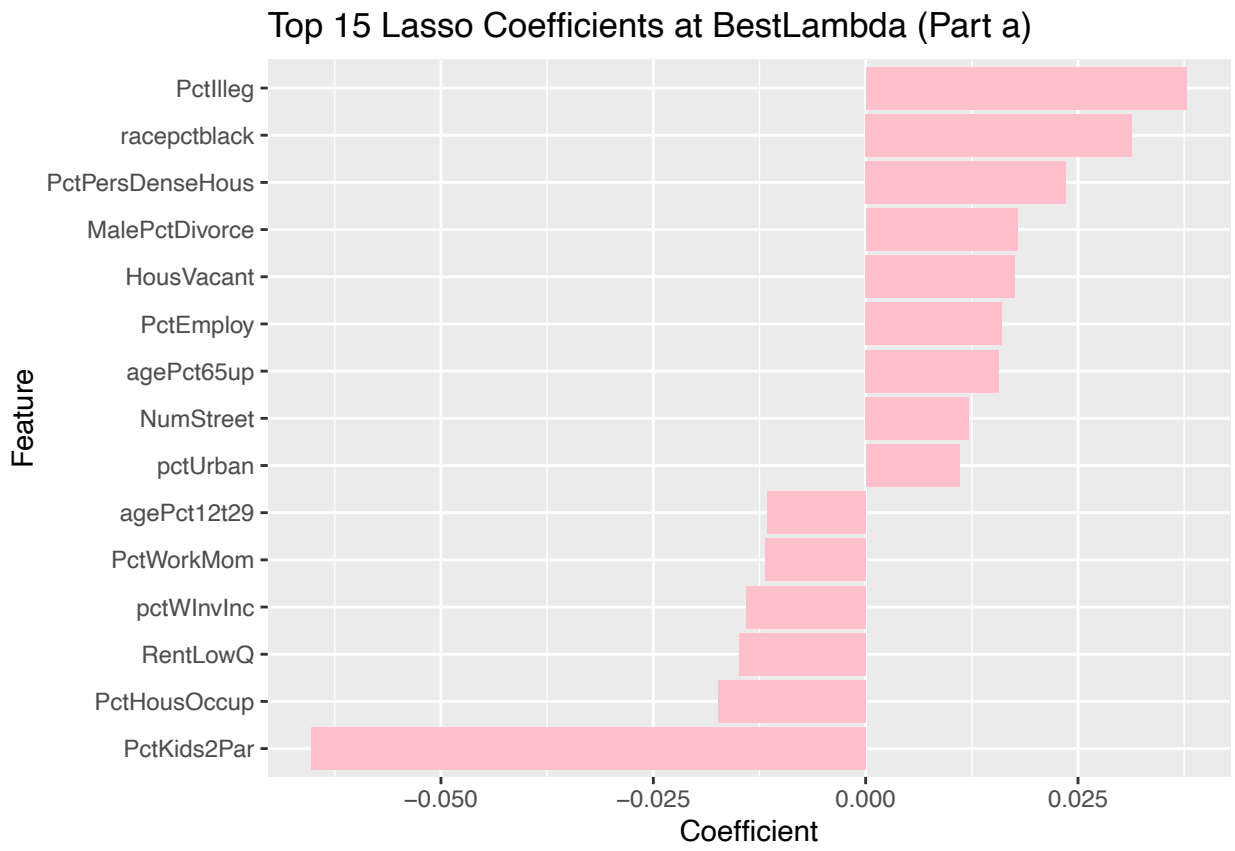
```r
# Lasso CV Error Curve (Part a): RMSE vs log(lambda)
cv_results <- lasso_cv$results
plot(log(cv_results$lambda), cv_results$RMSE, type = "l", lwd = 2,
     xlab = "log(Lambda)", ylab = "Cross-validated RMSE",
     main = "Lasso CV Error Curve (Part a)")
abline(v = log(optimal_lambda_a), col = "pink", lty = 2)
```



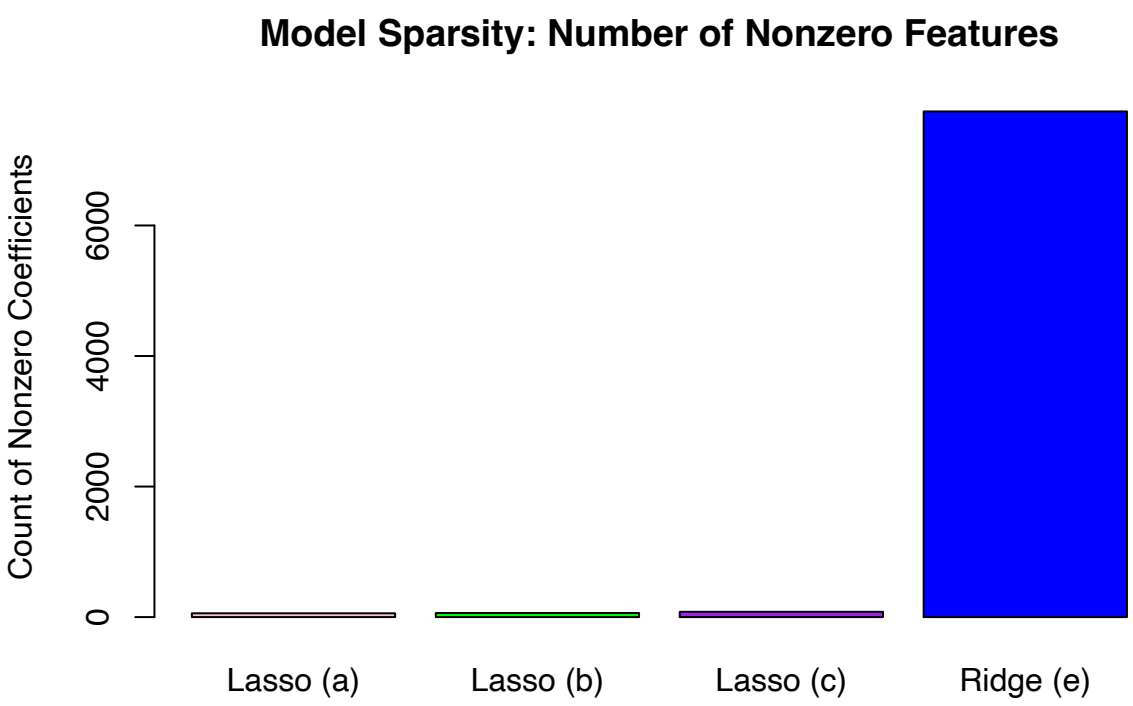

```
# Lasso CV Curve (Part a): Error vs log(lambda)
plot(lasso_cv$finalModel, xvar = "lambda", label = TRUE)
abline(v = log(optimal_lambda_a), col = "pink", lty = 2)
title("Lasso CV Curve (Part a): Error vs log(Lambda)")
```



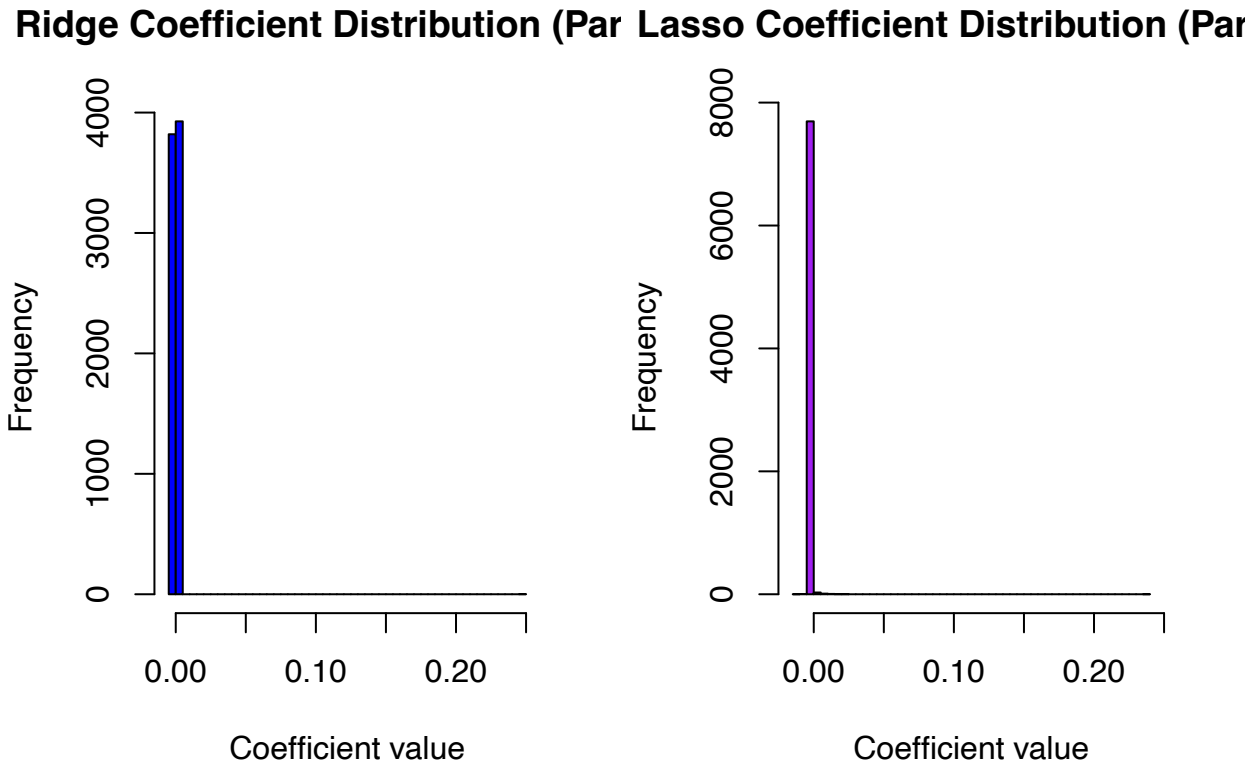

```r
# Top 15 Lasso Coefficients at Best Lambda (Part a)
coef_a <- coef(lasso_cv$finalModel, s = optimal_lambda_a)
coef_df <- data.frame(
  feature = rownames(coef_a),
  coef = as.numeric(coef_a)
)
coef_df <- subset(coef_df, coef != 0 & feature != "(Intercept)")
library(ggplot2)
library(dplyr)
coef_df %>%
  top_n(15, wt = abs(coef)) %>%
  ggplot(aes(x = reorder(feature, coef), y = coef)) +
  geom_bar(stat = "identity", fill = "pink") +
  coord_flip() +
  labs(title = "Top 15 Lasso Coefficients at BestLambda (Part a)",
       x = "Feature", y = "Coefficient")
```




```r
# Model Sparsity: Number of Nonzero Features
nonzero_count_a <- length(nonzero_features_a)
nonzero_count_b <- length(nonzero_features_b)
nonzero_count_c <- length(nonzero_features_c)
nonzero_count_ridge <- length(nonzero_features_ridge)
barplot(
  c(nonzero_count_a, nonzero_count_b, nonzero_count_c, nonzero_count_ridge),
  names.arg = c("Lasso (a)", "Lasso (b)", "Lasso (c)", "Ridge (e)"),
  col = c("pink", "green", "purple", "blue"),
  main = "Model Sparsity: Number of Nonzero Features",
  ylab = "Count of Nonzero Coefficients"
)
```




```r
# Coefficient Distribution: Ridge vs Lasso
ridge_coefs <- as.vector(coef(ridge_cv$finalModel, s = optimal_lambda_ridge))
lasso_coefs <- as.vector(coef(lasso_cv_all$finalModel, s = optimal_lambda_c))
par(mfrow = c(1, 2))
hist(ridge_coefs, breaks = 50,
     main = "Ridge Coefficient Distribution (Part e)",
     xlab = "Coefficient value", col = "blue")
hist(lasso_coefs, breaks = 50,
     main = "Lasso Coefficient Distribution (Part c)",
     xlab = "Coefficient value", col = "purple")
```




```
par(mfrow = c(1, 1))
```

Problem_3 - Riya

```
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
```

```
library(rpart)
library(randomForest)
```

```
## randomForest 4.7-1.2

## 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
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## v readr     2.1.5

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x randomForest::combine() masks dplyr::combine()
## x dplyr::filter()         masks stats::filter()
## x dplyr::lag()            masks stats::lag()
## x purrr::lift()           masks caret::lift()
## x randomForest::margin()  masks ggplot2::margin()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
 ↪  to become errors
```

```
library(readr)
library(ggplot2)
library(olsrr)
```

```
##
## Attaching package: 'olsrr'
##
## The following object is masked from 'package:datasets':
##
##     rivers
```

```
library(rpart.plot)
library(gbm)
```

```
## Loaded gbm 2.2.2
## This version of gbm is no longer under development. Consider transitioning to gbm3,
 ↪  https://github.com/gbm-developers/gbm3
```

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
##
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(dendextend)
```

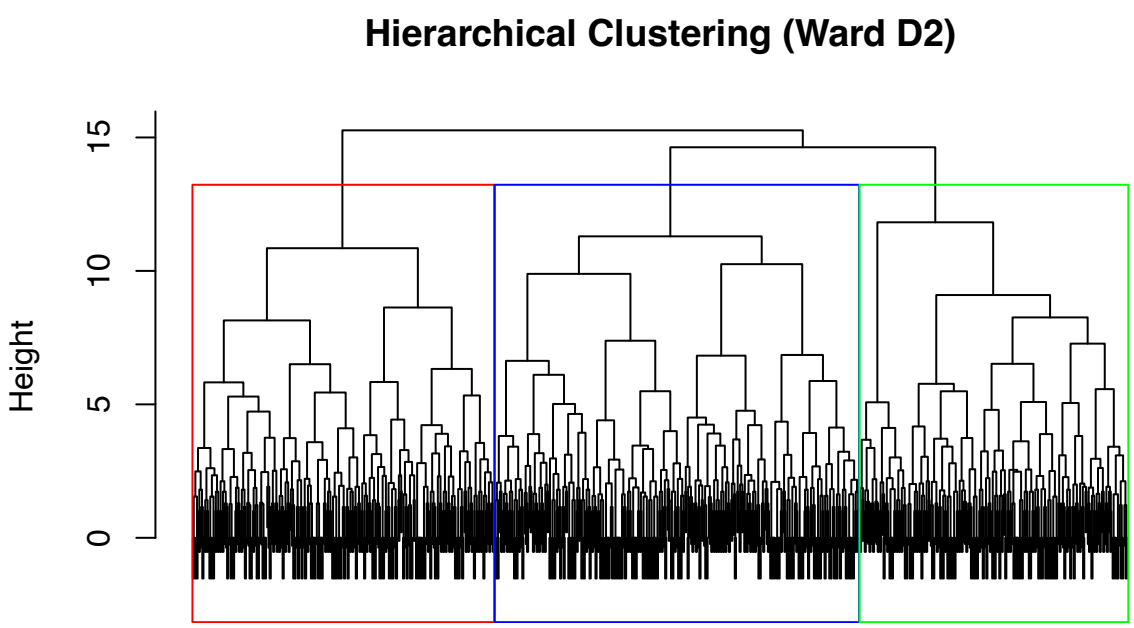
```
##
## ---------------------
## Welcome to dendextend version 1.19.1
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
```

```
##
## Suggestions and bug-reports can be submitted at:
 ↪  https://github.com/talgalili/dendextend/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ----------------------
##
##
## Attaching package: 'dendextend'
##
## The following object is masked from 'package:rpart':
##
##     prune
##
## The following object is masked from 'package:stats':
##
##     cutree
```

```
library(ggplot2)
library(ggfortify)
```

a) Cluster the data using hierarchical clustering, using the "Ward D2" measure of cluster dissimilarity. Then, use the cutree function to cut the dendrogram tree to 3 clusters and extract cluster assignments. Report the cluster sizes and comment on the characteristics of each cluster.

```
df <- read.csv("autoSurvey2024.csv")

# euclidean distance matrix
distance_matrix <- dist(df, method = "euclidean")

# hierarchical clustering
hc_result <- hclust(distance_matrix, method = "ward.D2")

# cut dendrogram to form 3 clusters
cluster_assignments <- cutree(hc_result, k = 3)

plot(hc_result, labels = FALSE, main = "Hierarchical Clustering (Ward D2)", xlab = "",
 ↪  sub = "")
cluster_colors <- c("red", "blue", "green")
rect.hclust(hc_result, k = 3, border = cluster_colors)  # only border is allowed
```




```
# add cluster labels to the original data
df_clustered <- df
df_clustered$cluster <- cluster_assignments

print(table(df_clustered$cluster))
```

```
##
##   1   2   3
## 256 228 309
```

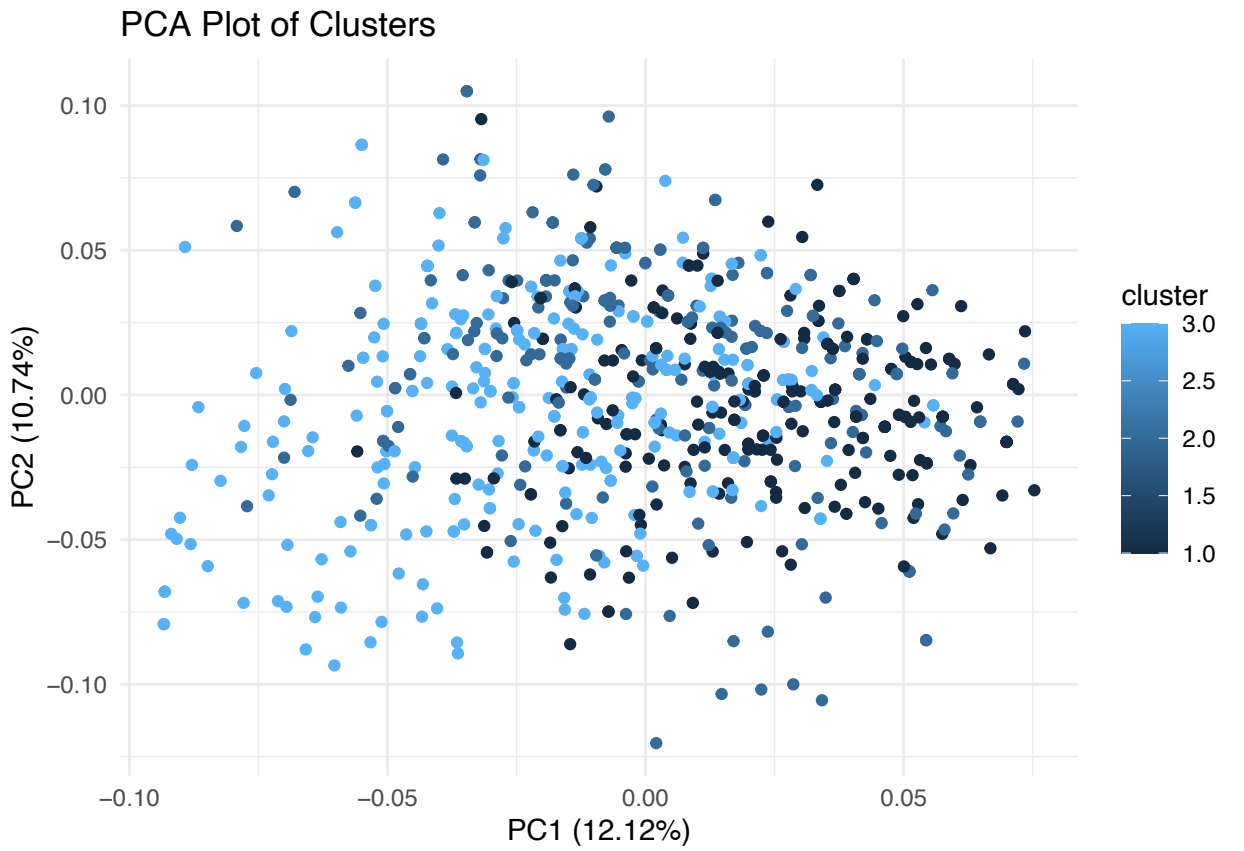
```
print(round(aggregate(. ~ cluster, data = df_clustered, FUN = mean), 2))
```

```
##   cluster driving_properties interior technology comfort reliability handling
## 1       1               0.80     0.17       0.33    0.25        0.26     0.47
## 2       2               0.72     0.25       0.59    0.62        0.76     0.32
## 3       3               0.57     0.31       0.42    0.33        0.28     0.12
##   power consumption sporty safety gender household space
## 1  0.62        0.22   0.80   0.24   0.05      0.68  0.08
## 2  0.55        0.17   0.39   0.71   0.08      0.42  0.08
## 3  0.34        0.35   0.10   0.30   0.06      0.65  0.23
```

```
# PCA plot for visualization
df_numeric <- df[sapply(df, is.numeric)]
df_scaled <- scale(df_numeric)
```

```r
autoplot(prcomp(df_scaled), data = df_clustered, colour = 'cluster') +
  ggtitle("PCA Plot of Clusters") +
  theme_minimal()
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## i The deprecated feature was likely used in the ggfortify package.
##   Please report the issue at <https://github.com/sinhrks/ggfortify/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Through Ward D2 clustering, we found 3 distinct customer segments based on preferences for vehicle features. Cluster1 with 256 customers demonstrates a strong affinity for driving performance (driving_properties = 0.80, power = 0.62) and sportiness (0.80), with moderate interest in handling (0.47) and larger household sizes (0.68). This group likely represents performance-oriented buyers who prioritize dynamic driving and style. Cluster 2 with 228 customers places the highest emphasis on safety (0.71) and reliability (0.76), alongside elevated scores for comfort (0.62) and technology (0.59). These customers appear to favor well-rounded, dependable vehicles, suggesting a more cautious or family-focused profile. Cluster 3 - the largest segment with 309 customers - stands out for its preference for space (0.23), consumption awareness (0.35), and interior features (0.31), with moderate comfort and household size scores. This group likely reflects practical buyers who value spaciousness and efficiency over performance or advanced features. Compared to earlier models, this segmentation offers clearer differentiation between performance-driven, safety-conscious, and space-seeking customer profiles.

b) Increase the number of clusters to 4 and analyze them. In what ways does the 4-cluster model differ from the 3-cluster model?

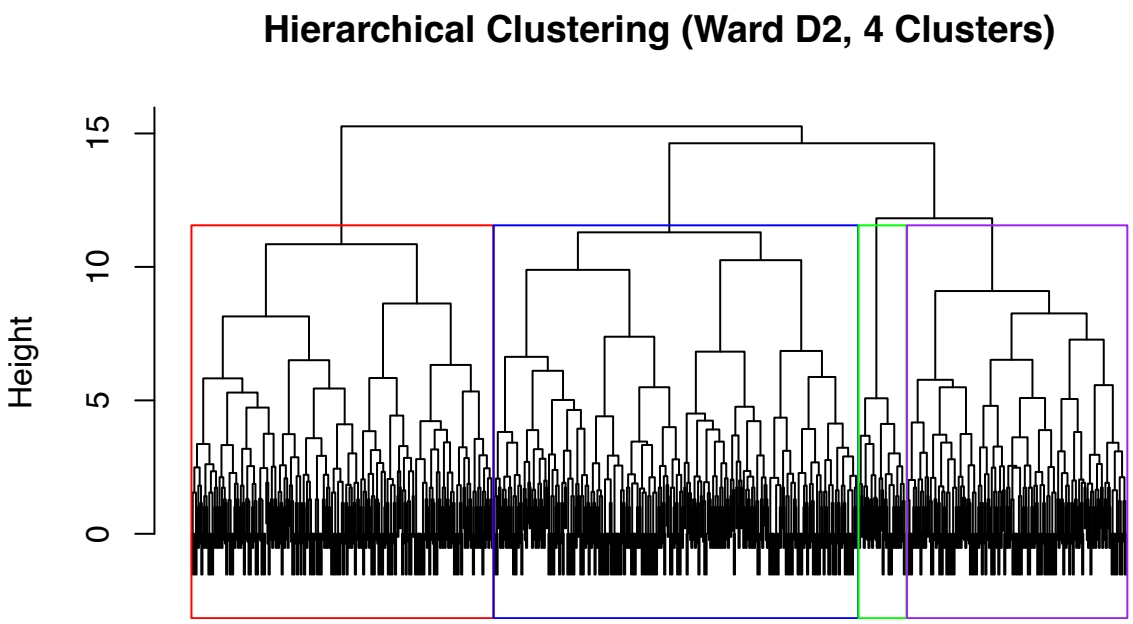
```
# cut dendrogram to form 4 clusters
cluster_assignments_4 <- cutree(hc_result, k = 4)

# add cluster labels to the original data
df_clustered_4 <- df
df_clustered_4$cluster_4 <- cluster_assignments_4

print(table(df_clustered_4$cluster_4))
```

```
##
##   1   2   3   4
## 256 187 309  41
```

```
print(round(aggregate(. ~ cluster_4, data = df_clustered_4, FUN = mean), 2))
```

```
##   cluster_4 driving_properties interior technology comfort reliability handling
## 1         1               0.80     0.17       0.33    0.25        0.26     0.47
## 2         2               0.66     0.16       0.53    0.57        0.76     0.18
## 3         3               0.57     0.31       0.42    0.33        0.28     0.12
## 4         4               1.00     0.66       0.83    0.85        0.76     0.98
##   power consumption sporty safety gender household space
## 1  0.62        0.22   0.80   0.24   0.05      0.68  0.08
## 2  0.48        0.07   0.29   0.67   0.10      0.37  0.04
## 3  0.34        0.35   0.10   0.30   0.06      0.65  0.23
## 4  0.88        0.61   0.83   0.88   0.00      0.63  0.24
```

```
# plot dendrogram for 4 clusters
plot(hc_result, labels = FALSE, main = "Hierarchical Clustering (Ward D2, 4 Clusters)",
 ↪  xlab = "", sub = "")
cluster_colors_4 <- c("red", "blue", "green", "purple")
rect.hclust(hc_result, k = 4, border = cluster_colors_4)
```



Expanding to a 4 cluster model revealed a more nuanced segmentation of customer preferences. Cluster 1 (256 customers) remains performance-driven, with high scores in driving properties (0.80), power (0.62), and sportiness (0.80), suggesting a style-focused group with larger households. Cluster 2 (187 customers) emphasizes safety (0.67), reliability (0.76), and comfort (0.57), reflecting practical buyers who value dependability and well-rounded features. Cluster 3 (309 customers) leans toward space (0.23), consumption awareness (0.35), and interior features (0.31), indicating a segment focused on efficiency and spaciousness. Cluster 4, though small (41 customers), isolates a premium, tech-savvy group with the highest scores in driving properties (1.00), handling (0.98), power (0.88), and comfort (0.85), along with strong interest in safety (0.88) and technology (0.83). Compared to the three-cluster model, this solution offers clearer separation, especially by splitting the original performance-oriented group into distinct mainstream and premium segments (cluster 2 seems to split into clusters 2 and 4 in this 4-cluster model).

c) Now cluster the data using the k-means algorithm, using 3 and 4 clusters. Set the seed to 15072 and max.iter to 100, and extract cluster assignments and compare them to those obtained with hierarchical clustering.

```r
# K-means clustering with 3 clusters
set.seed(15072)
kmeans_3 <- kmeans(df, centers = 3, iter.max = 100)

# K-means clustering with 4 clusters
set.seed(15072)
kmeans_4 <- kmeans(df, centers = 4, iter.max = 100)
```

```r
# Attach cluster labels to df
df$kmeans3 <- factor(kmeans_3$cluster)
df$kmeans4 <- factor(kmeans_4$cluster)

# Cluster sizes
cat("K-means (3 clusters):\n")
```

```
## K-means (3 clusters):
```

```r
print(table(df$kmeans3))
```

```
##
##   1   2   3
## 303 246 244
```

```r
cat("\nK-means (4 clusters):\n")
```

```
##
## K-means (4 clusters):
```

```r
print(table(df$kmeans4))
```

```
##
##   1   2   3   4
## 215 218 198 162
```

```r
cat("\nHierarchical clustering (3 clusters):\n")
```

```
##
## Hierarchical clustering (3 clusters):
```

```r
print(table(df_clustered$cluster))  # assuming df$cluster exists from earlier
```

```
##
##   1   2   3
## 256 228 309
```

```r
cat("\nHierarchical clustering (4 clusters):\n")
```

```
##
## Hierarchical clustering (4 clusters):
```

```r
print(table(df_clustered_4$cluster_4))  # assuming df$cluster_4 exists from earlier
```

```
##
##   1   2   3   4
## 256 187 309  41
```

```r
# Cross-tab comparisons
cat("\nCross-tab: Hierarchical vs K-means (3 clusters):\n")
```

```
##
## Cross-tab: Hierarchical vs K-means (3 clusters):
```

```r
print(table(Hierarchical = df_clustered$cluster, KMeans3 = df$kmeans3))
```

```
##             KMeans3
## Hierarchical   1   2   3
##            1 133  74  49
##            2  99  66  63
##            3  71 106 132
```

```r
cat("\nCross-tab: Hierarchical vs K-means (4 clusters):\n")
```

```
##
## Cross-tab: Hierarchical vs K-means (4 clusters):
```

```r
print(table(Hierarchical = df_clustered_4$cluster_4, KMeans4 = df$kmeans4))
```

```
##             KMeans4
## Hierarchical   1   2   3   4
##            1  81  51  25  99
##            2  62  61  52  12
##            3  71 104 121  13
##            4   1   2   0  38
```

To compare clustering methods, we used both hierarchical and k-means clustering on the binary data. The cross-tab results show that hierarchical cluster 1 (256 customers) is spread across all k-means groups in the 3-cluster model, landing in k-means clusters 1 (81), 2 (51), and 3 (124). This means k-means breaks up the largest hierarchical group into smaller subgroups. Hierarchical cluster 2 (228 customers) also splits fairly evenly across k-means clusters, while cluster 3 (309 customers) mostly maps to k-means cluster 3 (134) but still overlaps with the others. In the 4-cluster comparison, hierarchical cluster 1 again spreads across all k-means groups, especially cluster 4 (94), showing that k-means gives more detailed segmentation. The small hierarchical cluster 4 (41 customers) mostly matches k-means cluster 4 (38), which suggests k-means can pick out more specific customer types. Overall, k-means helps broke down our broad groups into finer segments, which would be useful for identifying customer sub types and improving customer targeting.

d) Based on your responses in parts a, b, and c, in what ways are the clusters similar and different across the two clustering approaches (hierarchical and k-means)?

Across hierarchical clustering and k-means, several consistent patterns emerge, but the methods also reveal distinct segmentation strategies. Hierarchical clustering groups similar things together one by one after treating them each as individual entities, while k-means tries to find the points most similar to a nearest central point. Both approaches identify a cluster of customers who prioritize comfort, reliability, and safety, as well as a group focused on interior features and space, suggesting agreement on core consumer archetypes.

However, hierarchical clustering tends to preserve broader groupings—especially in the 3-cluster model—while k-means more aggressively partitions large clusters into finer subgroups. For example, hierarchical cluster 1, which contains the majority of customers, is split across multiple k-means clusters, indicating that k-means detects latent variation within this dominant segment. In the 4-cluster comparison, k-means isolates a high-performance, tech-savvy group with elevated scores in driving properties, handling, and power—similar to the premium cluster identified in hierarchical clustering. Overall, while both methods capture similar thematic clusters, k-means offers sharper boundaries and more granular segmentation, especially within large, heterogeneous groups.