Practical 6

Regression Models

Joan Navarro Bellido

```r
# Load required packages
library(lattice)
library(rpart)
library(nnet)
library(rpart.plot)
library(stargazer)
library(ggplot2)
library(scales)
library(Metrics)
library(caret)
library(randomForest)
library(e1071)
library(dplyr)
```

```
##
## Adjuntando el paquete: 'dplyr'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

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

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

Exercice 1

```r
# Load the dataset
house <- read.csv(file = "./dataset/house.csv", sep = ";")

# Remove street information
house$street <- NULL

# Exploring the dataset
str(house)
```

```
## 'data.frame':    799 obs. of  10 variables:
##  $ zpid   : int  956068 924224 906733 964007 74504350 81982160 7780105 7792424 7833403 56437339 ...
##  $ zipcode: int  35212 35204 35215 35205 99801 85037 85021 85021 85018 2140 ...
```

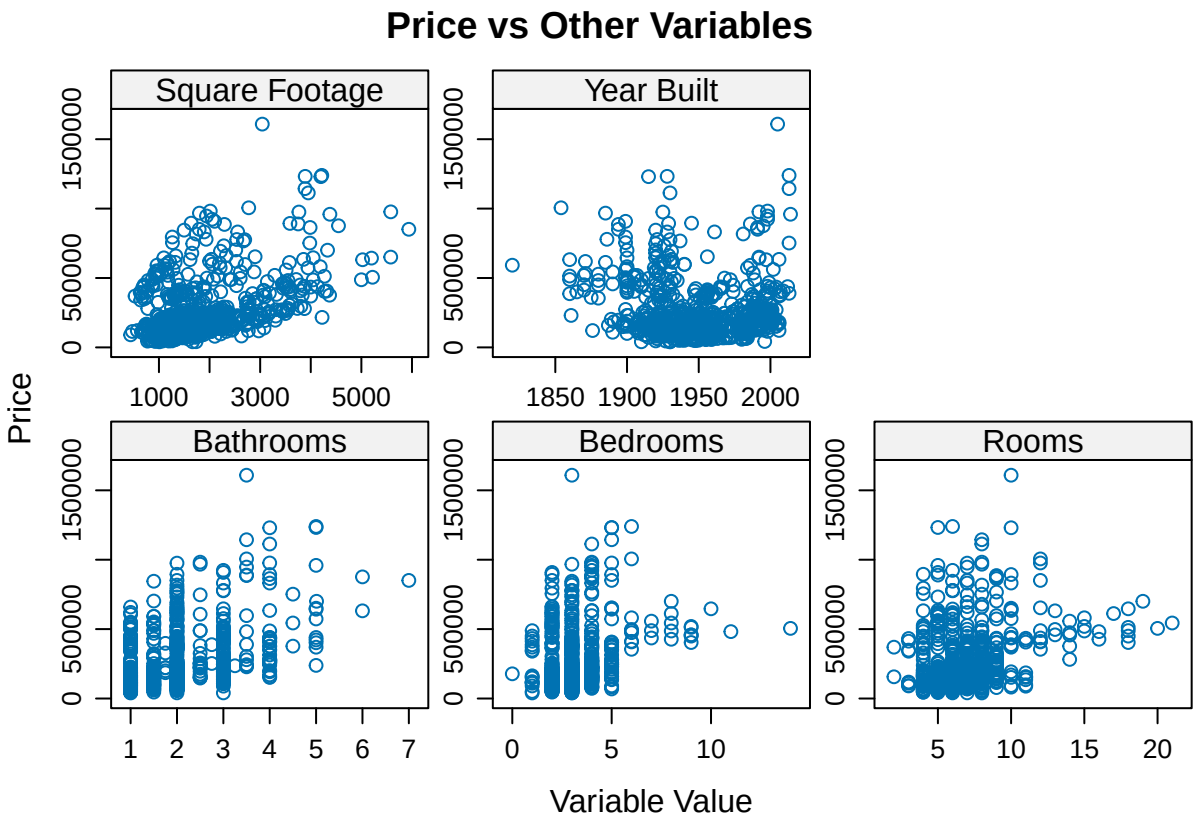
```
##  $ city   : chr  "Birmingham" "Birmingham" "Birmingham" "Birmingham" ...
##  $ state  : chr  "AL" "AL" "AL" "AL" ...
##  $ year   : int  1930 1930 1982 1919 1966 2006 1984 1974 1980 1894 ...
##  $ bath   : num  2 1 2 2.5 1 3 5 2 2 3.5 ...
##  $ bed    : int  3 2 3 4 1 3 5 2 2 4 ...
##  $ rooms  : int  7 6 11 7 3 5 9 4 4 9 ...
##  $ SqFt   : int  1732 1115 1355 2876 476 1652 3945 1625 1794 2294 ...
##  $ price  : int  40745 205906 98672 325474 114726 122241 573258 239086 304824 885883 ...
```

```r
# Summary of the dataset
summary(house)
```

```
##       zpid              zipcode          city              state
##  Min.   :9.060e+05   Min.   : 2121   Length:799         Length:799
##  1st Qu.:1.030e+07   1st Qu.:35215   Class :character   Class :character
##  Median :2.830e+07   Median :53207   Mode  :character   Mode  :character
##  Mean   :3.735e+07   Mean   :49957
##  3rd Qu.:5.910e+07   3rd Qu.:73162
##  Max.   :2.147e+09   Max.   :99801
##       year           bath            bed             rooms
##  Min.   :1820   Min.   :1.000   Min.   : 0.000   Min.   : 2.000
##  1st Qu.:1927   1st Qu.:1.000   1st Qu.: 3.000   1st Qu.: 5.000
##  Median :1951   Median :2.000   Median : 3.000   Median : 6.000
##  Mean   :1951   Mean   :2.039   Mean   : 3.218   Mean   : 6.708
##  3rd Qu.:1980   3rd Qu.:2.000   3rd Qu.: 4.000   3rd Qu.: 8.000
##  Max.   :2014   Max.   :7.000   Max.   :14.000   Max.   :21.000
##       SqFt          price
##  Min.   : 440   Min.   :  40074
##  1st Qu.:1196   1st Qu.: 125058
##  Median :1536   Median : 179886
##  Mean   :1792   Mean   : 252797
##  3rd Qu.:2146   3rd Qu.: 297120
##  Max.   :5938   Max.   :1608791
```

Exercice 2

```r
# Create a data frame for plotting
forplot <- make.groups(
  bath = data.frame(value = house$bath, Variable = "Bathrooms", price = house$price),
  year = data.frame(value = house$year, Variable = "Year Built", price = house$price),
  bed = data.frame(value = house$bed, Variable = "Bedrooms", price = house$price),
  rooms = data.frame(value = house$rooms, Variable = "Rooms", price = house$price),
  SqFt = data.frame(value = house$SqFt, Variable = "Square Footage", price = house$price)
)

# Plot with clear labels and titles
xyplot(price ~ value | Variable, data = forplot, scales = list(relation = "free"),
       main = "Price vs Other Variables",
       xlab = "Variable Value", ylab = "Price")
```



Exercice 3

```r
# It's possible to set the seed for reproducibility
# if needed using set.seed(123)

# Split the data into training (75%) and testing (25%)
sample_index <- sample(seq_len(nrow(house)), size = 0.75 * nrow(house))
train_data <- house[sample_index, ]
test_data <- house[-sample_index, ]

# Convert character columns to factors and align levels in training and test datasets
train_data <- train_data %>% mutate(across(where(is.character), as.factor))
test_data <- test_data %>% mutate(across(where(is.character), as.factor))

# Ensure factor levels are aligned between train and test data for all factor columns
for (col in names(train_data)) {
  if (is.factor(train_data[[col]])) {
    test_data[[col]] <- factor(test_data[[col]], levels = levels(train_data[[col]]))
  }
}

# Ensure columns are in the same order in both datasets
test_data <- test_data[, names(train_data)]
```

Exercice 4

```r
# Fit a Linear Model
lm_model <- lm(price ~ ., data = train_data)

# Fit a Regression Tree (CART)
cart_model <- rpart(price ~ ., data = train_data, method = "anova")

# Fit a Neural Network
nn_model <- nnet(price ~ ., data = train_data, size = 12, linout = TRUE, skip = TRUE, maxit = 500)
```

```
## # weights:  896
## initial  value 2596248088205684.500000
## iter  10 value 110111904548632.953125
## iter  20 value 36214130229734.437500
## iter  30 value 14158378764797.273438
## iter  40 value 5389716322368.878906
## iter  50 value 4840412723153.148438
## iter  60 value 4833524580523.650391
## iter  70 value 4833006116931.970703
## final  value 4828232756281.985352
## converged
```

```r
# Summarize the neural network model training
nn_final_result <- summary(nn_model)
print(paste("Convergence Status:", ifelse(nn_final_result$convergence == 0, "Converged", "Not Converged
```

```
## [1] "Convergence Status: Converged"
```

```r
print(paste("Final Error Value:", round(nn_final_result$value, 2)))
```

```
## [1] "Final Error Value: 4828232756281.98"
```

```r
# Optional: Display the hyperparameters and summary for confirmation
cat("Hyperparameters used:\n")
```

```
## Hyperparameters used:
```

```r
print(paste("Size:", nn_model$n[1]))
```

```
## [1] "Size: 67"
```

```r
print(paste("Max iterations:", nn_model$maxit))
```

```
## [1] "Max iterations: "
```

```r
print(paste("Weight decay:", nn_model$decay))
```

```
## [1] "Weight decay: 0"
```

Exercice 5

```r
# Linear Model Summary
stargazer(lm_model, type = "text", title = "Linear Model Summary",
          single.row = TRUE, digits = 2, align = TRUE)
```

```
##
## Linear Model Summary
```

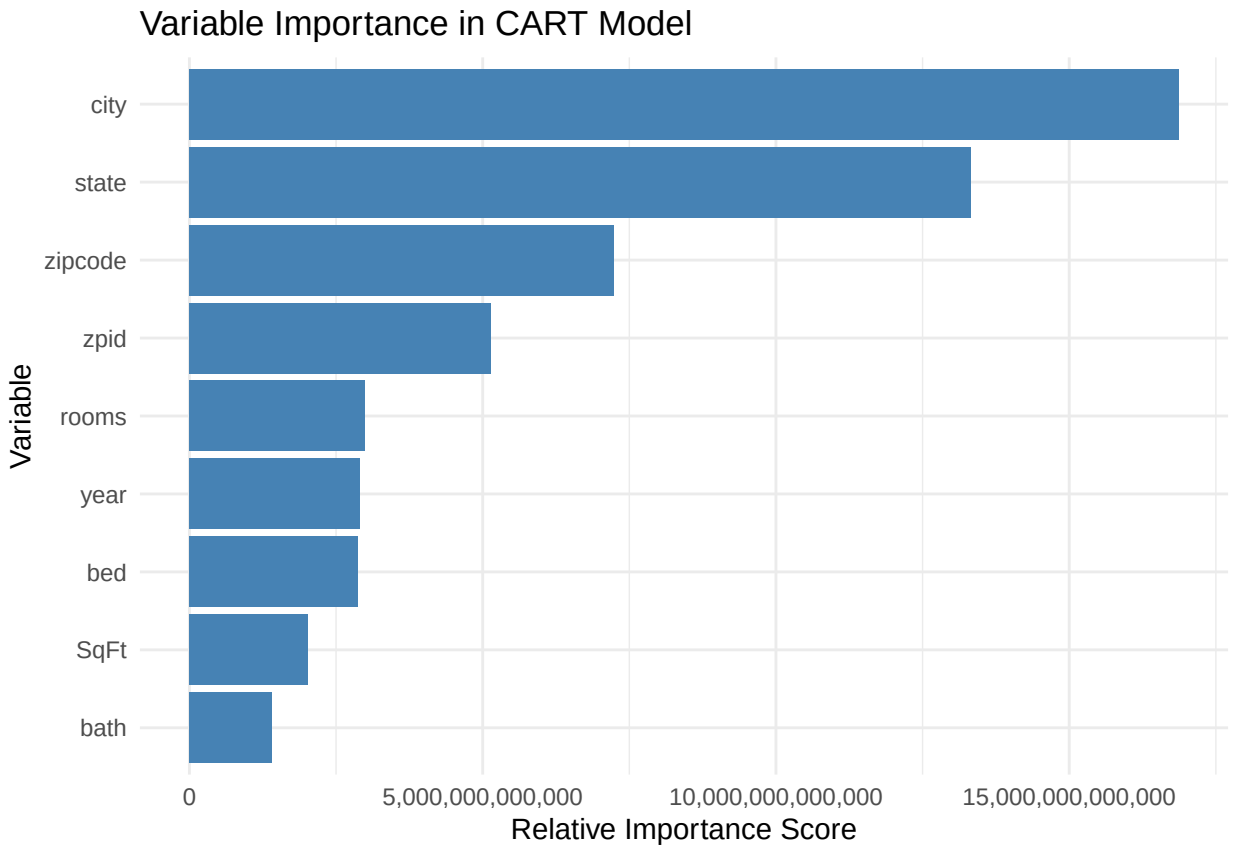
```
## ================================================
##                          Dependent variable:
##                      ---------------------------
##                                  price
## ------------------------------------------------
## zpid                          0.0000 (0.0000)
## zipcode                       331.36 (426.17)
## cityATLANTA                35,374.99 (96,522.78)
## cityBel Aire           -12,431,632.00 (15,726,464.00)
## cityBirmingham          -1,658,687.00 (2,085,630.00)
## cityBoston              9,798,542.00 (12,011,971.00)
## cityBOSTON              9,783,260.00 (12,010,897.00)
## cityCambridge           9,860,645.00 (12,007,927.00)
## cityCAMBRIDGE           9,850,795.00 (12,008,067.00)
## cityCarmel              -5,206,183.00 (6,694,355.00)
## cityCHARLESTOWN         9,760,561.00 (12,010,887.00)
## cityCharlotte             604,562.70 (891,992.50)
## cityChicago            -9,978,231.00 (12,928,514.00)
## cityCleveland           -4,605,927.00 (5,881,436.00)
## cityConcord            -15,613,209.00 (20,147,490.00)
## cityDenver             -16,288,232.00 (21,261,603.00)
## cityDorchester          9,345,058.00 (12,014,561.00)
## cityEncino             -19,699,653.00 (26,045,586.00)
## cityFalmouth            8,942,587.00 (11,172,328.00)
## cityGermantown          -2,626,370.00 (3,329,900.00)
## cityHouston            -15,646,211.00 (19,912,490.00)
## cityIndianapolis        -5,317,120.00 (6,797,011.00)
## cityJackson             -3,051,597.00 (3,788,094.00)
## cityJuneau             -22,899,067.00 (29,610,706.00)
## cityLouisville          -3,329,100.00 (4,211,896.00)
## cityMemphis             -2,677,128.00 (3,321,978.00)
## cityMiami                -918,903.00 (1,203,980.00)
## cityMilton              9,503,852.00 (11,987,903.00)
## cityMilwaukee           -7,570,089.00 (9,756,024.00)
## cityMinneapolis        -8,189,103.00 (10,694,209.00)
## cityOklahoma City      -14,224,904.00 (18,245,407.00)
## cityPasadena           -15,726,898.00 (20,107,807.00)
## cityPhiladelphia        3,782,540.00 (4,761,030.00)
## cityPhoenix            -18,125,828.00 (23,311,924.00)
## cityPortland            8,762,656.00 (11,172,464.00)
## cityRichfield          -8,250,306.00 (10,698,568.00)
## citySacramento         -21,650,707.00 (27,912,279.00)
## citySylmar             -19,986,404.00 (26,005,719.00)
## cityWebster            -15,724,608.00 (20,147,279.00)
## cityWichita            -12,380,358.00 (15,725,417.00)
## stateAL
## stateAZ
## stateCA
## stateCO
## stateFL
## stateGA
## stateIL
## stateIN
## stateKS
```

```
## stateKY
## stateMA
## stateME
## stateMN
## stateMS
## stateNC
## stateNH             24,595,588.00 (31,661,746.00)
## stateOH
## stateOK
## statePA
## stateTN
## stateTX
## stateWI
## year                      61.56 (188.55)
## bath                  43,502.49*** (7,523.76)
## bed                   -12,832.79** (6,034.06)
## rooms                   4,987.15 (3,198.19)
## SqFt                     137.77*** (10.34)
## Constant            -10,291,305.00 (12,831,931.00)
## -------------------------------------------------
## Observations                  599
## R2                            0.80
## Adjusted R2                   0.79
## Residual Std. Error     93,524.32 (df = 552)
## F Statistic           49.30*** (df = 46; 552)
## =================================================
## Note:                 *p<0.1; **p<0.05; ***p<0.01
```

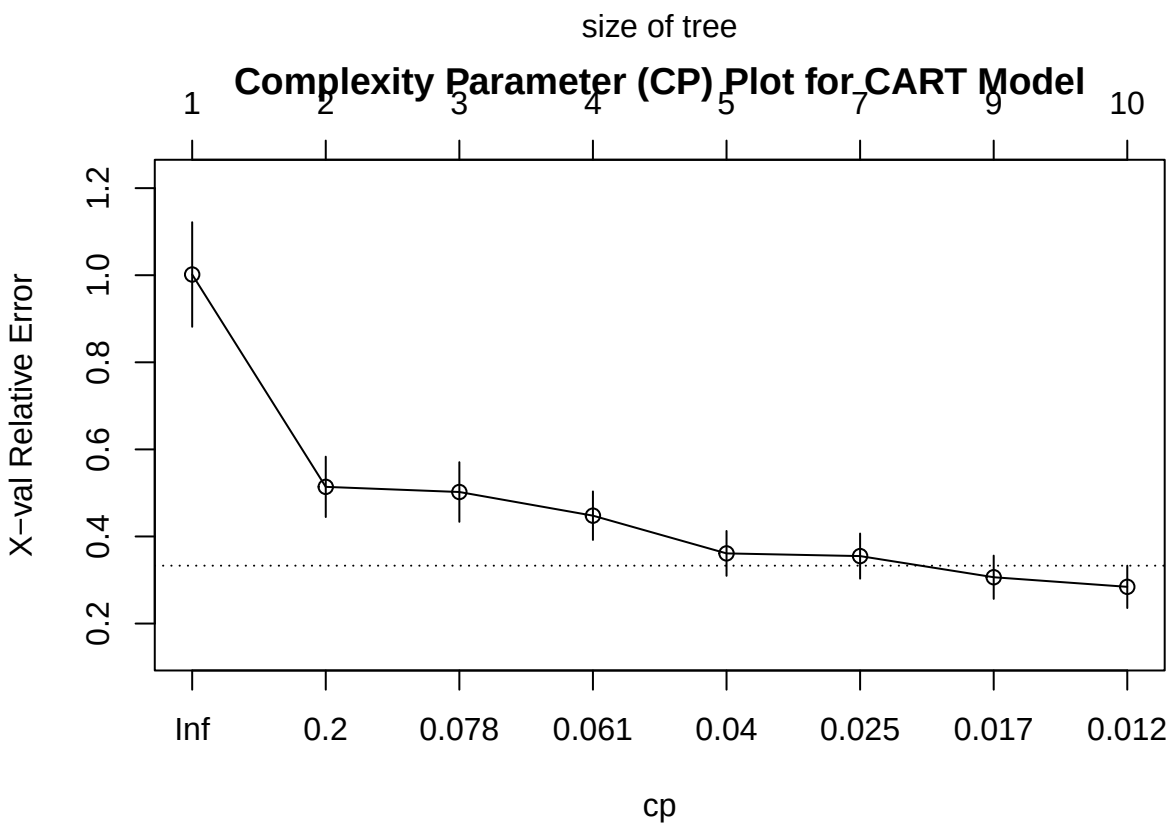
```r
# Summary of Variable Importance
importance <- cart_model$variable.importance
print(importance)
```

```
##         city        state      zipcode         zpid        rooms         year
## 1.686447e+13 1.332089e+13 7.232675e+12 5.138862e+12 2.999969e+12 2.911027e+12
##          bed         SqFt         bath
## 2.878574e+12 2.012325e+12 1.407382e+12
```

```r
# Plot Variable Importance
importance_df <- data.frame(Variable = names(importance), Importance = importance)

# Plot Variable Importance with improved labels
ggplot(importance_df, aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  scale_y_continuous(labels = comma) +  # Format y-axis with commas
  labs(title = "Variable Importance in CART Model", x = "Variable", y = "Relative Importance Score") +
  theme_minimal()
```



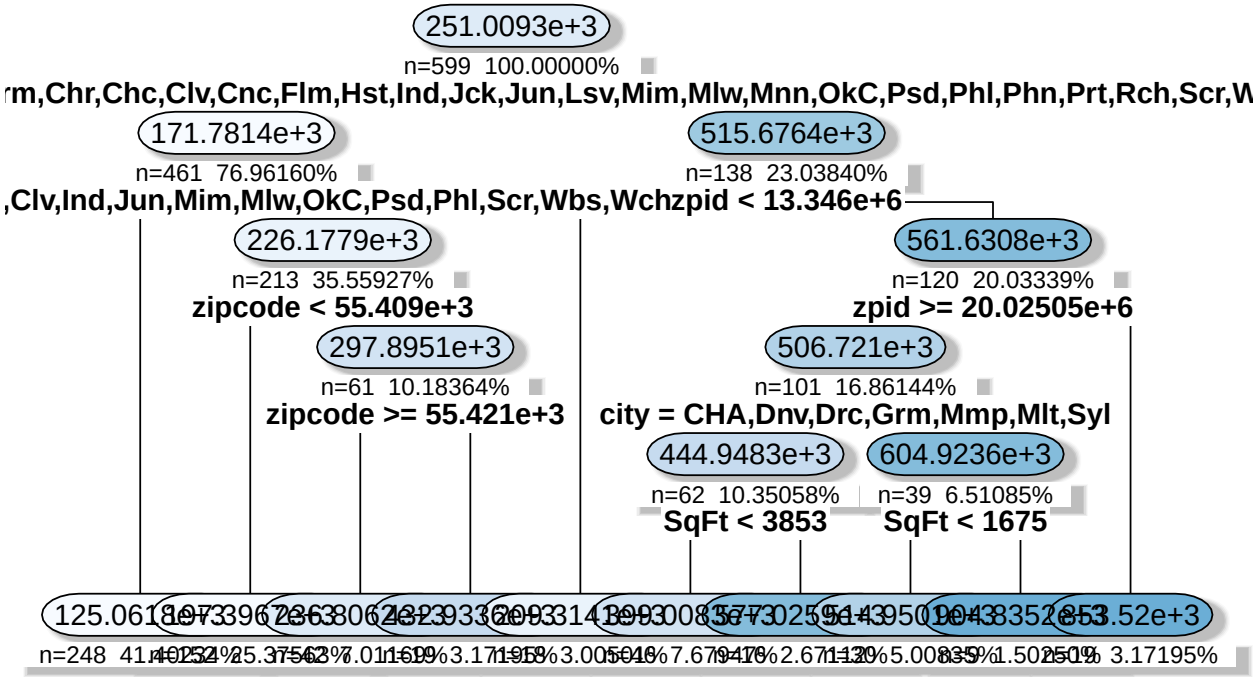

```
# Plot the Complexity Parameter Table for pruning insight
plotcp(cart_model, main = "Complexity Parameter (CP) Plot for CART Model")
```




```r
# Plot of the CART Model
rpart.plot(cart_model,
           type = 2,                # Display split labels only on branches
           extra = 101,             # Display node percentage and response values
           under = TRUE,            # Show the response value under each node
           faclen = 3,              # Shorten factor level names if they are too long
           cex = 0.75,              # Adjust text size for better readability
           box.palette = "Blues",   # Use a color palette for easier differentiation of nodes
           shadow.col = "gray",     # Add shadow for a 3D effect
           tweak = 1.2,             # Adjust spacing in the plot for a clearer view
           digits = 0,              # Show rounded values
           main = "Enhanced CART Model Visualization")
```

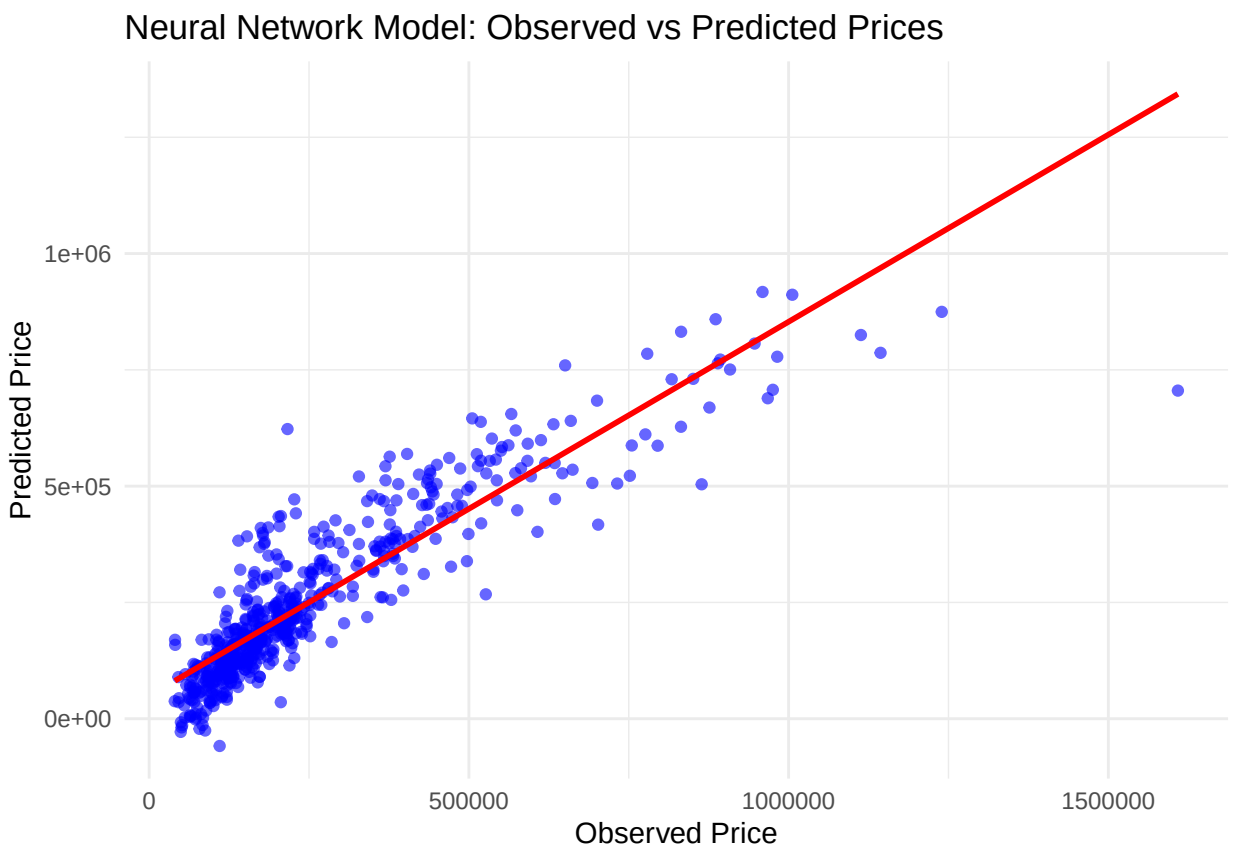
```
## Warning: cex and tweak both specified, applying both
```

Enhanced CART Model Visualization



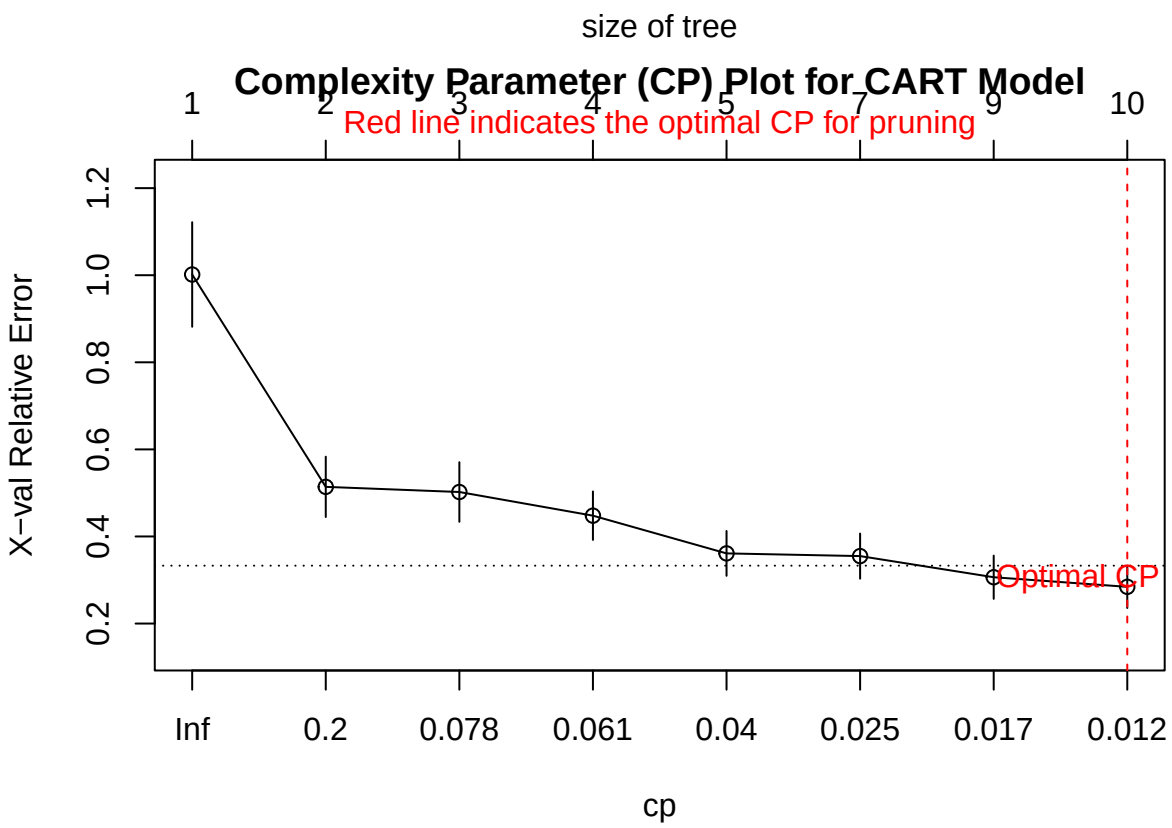

```
# Neural Network Predictions Visualization
nn_train_pred <- predict(nn_model, train_data, type = "raw")
nn_comparison <- data.frame(Observed = train_data$price, Predicted = nn_train_pred)

ggplot(nn_comparison, aes(x = Observed, y = Predicted)) +
  geom_point(color = "blue", alpha = 0.6) +
  geom_smooth(method = "lm", color = "red", se = FALSE) +
  labs(title = "Neural Network Model: Observed vs Predicted Prices",
       x = "Observed Price", y = "Predicted Price") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Exercice 6

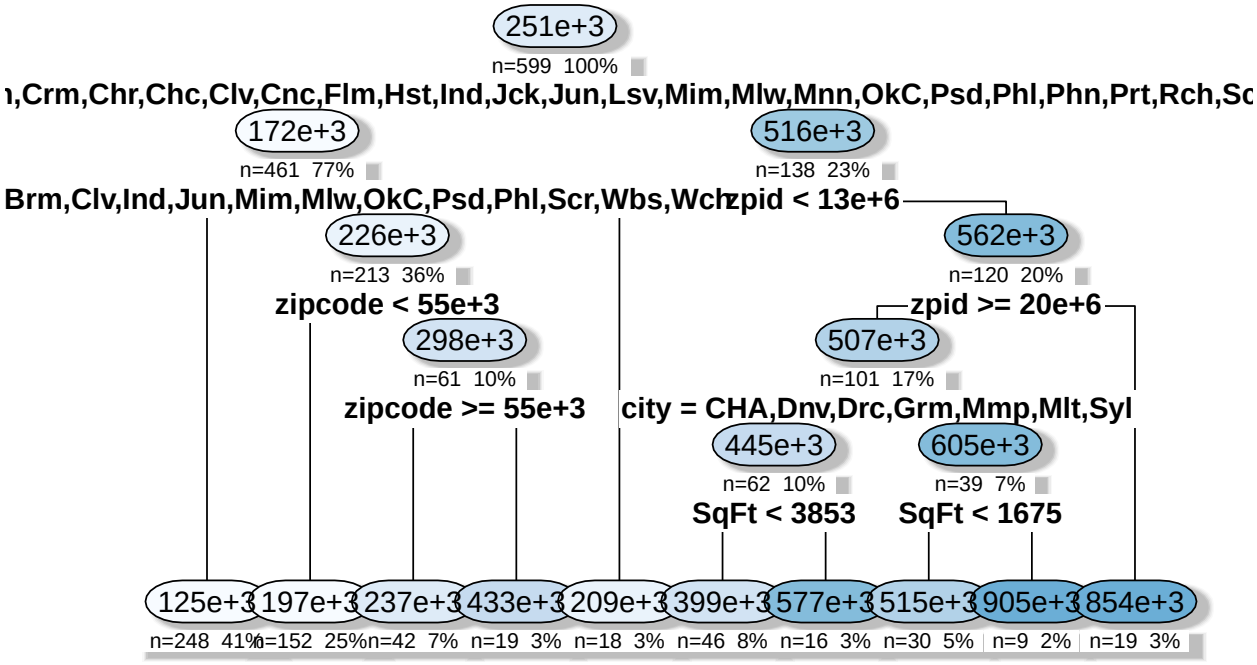
```r
# Visualize the complexity parameter plot with better formatting
plotcp(cart_model, main = "Complexity Parameter (CP) Plot for CART Model")
abline(v = which.min(cart_model$cptable[, "xerror"]), col = "red", lty = 2)
text(which.min(cart_model$cptable[, "xerror"]), min(cart_model$cptable[, "xerror"]) + 0.02,
     labels = paste("Optimal CP =", round(cart_model$cptable[which.min(cart_model$cptable[, "xerror"]),
     col = "red")
mtext("Red line indicates the optimal CP for pruning", side = 3, line = 0.5, col = "red")
```




```
# Prune the tree at an optimal CP value (e.g., 0.01 or based on the CP plot)
pruned_cart_model <- prune(cart_model, cp = 0.01)  # Adjust the cp based on plotcp output

# Visualize the pruned CART model with improved aesthetics
rpart.plot(pruned_cart_model,
           type = 2,                # Display split labels only on branches
           extra = 101,             # Show percentage of observations and response values
           under = TRUE,            # Display response values below each node
           faclen = 3,              # Shorten factor level names for brevity
           cex = 0.8,               # Adjust text size for better readability
           box.palette = "Blues",   # Use a color palette for nodes
           shadow.col = "gray",     # Add a shadow for a 3D effect
           tweak = 1.1,             # Adjust spacing in the plot
           main = "Pruned CART Model Visualization with Optimal Complexity")
```

```
## Warning: cex and tweak both specified, applying both
```

Pruned CART Model Visualization with Optimal Complexity



Exercice 7

```
evaluate_model <- function(model, train_data, test_data, target_variable) {
  # Align levels in test data
  test_data$city <- factor(test_data$city, levels = levels(train_data$city))
  test_data$state <- factor(test_data$state, levels = levels(train_data$state))

  # Predict on training and test data
  train_predictions <- predict(model, train_data)
  test_predictions <- predict(model, test_data)

  # Filter NA values for consistent metric calculation
  actual_train <- na.omit(train_data[[target_variable]])
  pred_train <- train_predictions[!is.na(train_predictions)]

  actual_test <- na.omit(test_data[[target_variable]])
  pred_test <- test_predictions[!is.na(test_predictions)]

  # Calculate RMSE and MAE
  train_rmse <- rmse(actual_train, pred_train)
  train_mae <- mae(actual_train, pred_train)

  test_rmse <- rmse(actual_test, pred_test)
  test_mae <- mae(actual_test, pred_test)

  return(list(train_rmse = train_rmse, train_mae = train_mae,
              test_rmse = test_rmse, test_mae = test_mae))
}
```

```r
# Assuming the target variable is "price" and the models are already trained
lm_results <- evaluate_model(lm_model, train_data, test_data, "price")
```

```
## Warning in actual - predicted: longitud de objeto mayor no es múltiplo de la
## longitud de uno menor
## Warning in actual - predicted: longitud de objeto mayor no es múltiplo de la
## longitud de uno menor
```

```r
cart_results <- evaluate_model(cart_model, train_data, test_data, "price")
nn_results <- evaluate_model(nn_model, train_data, test_data, "price")
```

```
## Warning in actual - predicted: longitud de objeto mayor no es múltiplo de la
## longitud de uno menor
## Warning in actual - predicted: longitud de objeto mayor no es múltiplo de la
## longitud de uno menor
```

```r
# Print results
print("Linear Model Performance:")
```

```
## [1] "Linear Model Performance:"
```

```r
print(lm_results)
```

```
## $train_rmse
## [1] 89780.22
##
## $train_mae
## [1] 56248.25
##
## $test_rmse
## [1] 198449
##
## $test_mae
## [1] 127398.4
```

```r
print("CART Model Performance:")
```

```
## [1] "CART Model Performance:"
```

```r
print(cart_results)
```

```
## $train_rmse
## [1] 82543.79
##
## $train_mae
## [1] 52948.14
##
## $test_rmse
## [1] 100828.4
##
## $test_mae
## [1] 64653.4
```

```r
print("Neural Network Model Performance:")
```

```
## [1] "Neural Network Model Performance:"
```

```r
print(nn_results)
```

```
## $train_rmse
## [1] 89780.22
##
## $train_mae
## [1] 56248.28
##
## $test_rmse
## [1] 198449
##
## $test_mae
## [1] 127398.4
```

Exercice 8

1. Linear Model (lm)

```r
# Interaction Terms: Add interaction terms to capture relationships between features.
# This can be helpful if certain variables influence each other.
# Feature Transformation: Apply transformations like log or polynomial terms to features
# that might have non-linear relationships with the target variable.

lm_model_improved <- lm(price ~ . + I(SqFt^2) + bed:bath + log(SqFt), data = train_data)
summary(lm_model_improved)
```

```
##
## Call:
## lm(formula = price ~ . + I(SqFt^2) + bed:bath + log(SqFt), data = train_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -403666  -37515     387   32010  902786
##
## Coefficients: (21 not defined because of singularities)
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -9.783e+06  1.295e+07  -0.756   0.4502
## zpid               3.343e-05  4.621e-05   0.723   0.4698
## zipcode            3.094e+02  4.296e+02   0.720   0.4716
## cityATLANTA        3.456e+04  9.678e+04   0.357   0.7212
## cityBel Aire      -1.162e+07  1.585e+07  -0.733   0.4638
## cityBirmingham    -1.550e+06  2.102e+06  -0.737   0.4613
## cityBoston         9.184e+06  1.211e+07   0.759   0.4484
## cityBOSTON         9.168e+06  1.211e+07   0.757   0.4492
## cityCambridge      9.246e+06  1.210e+07   0.764   0.4452
## cityCAMBRIDGE      9.236e+06  1.210e+07   0.763   0.4457
## cityCarmel        -4.862e+06  6.748e+06  -0.721   0.4715
## cityCHARLESTOWN    9.148e+06  1.211e+07   0.756   0.4502
## cityCharlotte      5.596e+05  8.992e+05   0.622   0.5340
## cityChicago       -9.312e+06  1.303e+07  -0.715   0.4752
## cityCleveland     -4.302e+06  5.928e+06  -0.726   0.4683
## cityConcord       -1.458e+07  2.031e+07  -0.718   0.4732
## cityDenver        -1.519e+07  2.143e+07  -0.709   0.4787
```

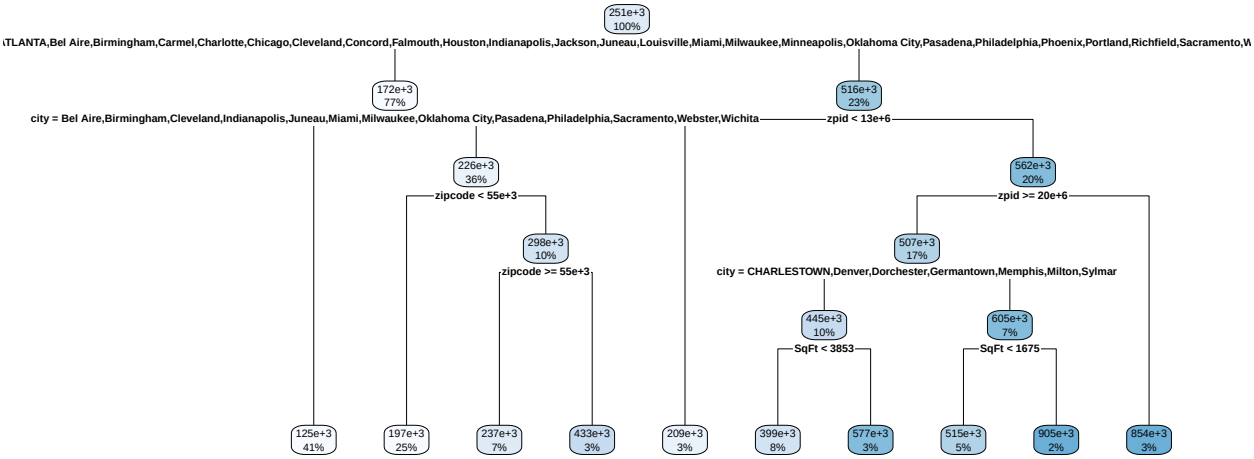
```
## cityDorchester     8.737e+06  1.211e+07   0.722   0.4709
## cityEncino        -1.836e+07  2.625e+07  -0.699   0.4846
## cityFalmouth       8.374e+06  1.126e+07   0.744   0.4574
## cityGermantown    -2.452e+06  3.357e+06  -0.730   0.4655
## cityHouston       -1.462e+07  2.007e+07  -0.728   0.4667
## cityIndianapolis  -4.967e+06  6.851e+06  -0.725   0.4688
## cityJackson       -2.856e+06  3.818e+06  -0.748   0.4548
## cityJuneau        -2.136e+07  2.985e+07  -0.716   0.4745
## cityLouisville    -3.113e+06  4.245e+06  -0.733   0.4637
## cityMemphis       -2.501e+06  3.349e+06  -0.747   0.4556
## cityMiami         -8.551e+05  1.214e+06  -0.704   0.4814
## cityMilton         8.888e+06  1.208e+07   0.736   0.4623
## cityMilwaukee     -7.067e+06  9.834e+06  -0.719   0.4726
## cityMinneapolis   -7.637e+06  1.078e+07  -0.708   0.4790
## cityOklahoma City -1.328e+07  1.839e+07  -0.722   0.4704
## cityPasadena      -1.469e+07  2.027e+07  -0.725   0.4688
## cityPhiladelphia   3.540e+06  4.799e+06   0.738   0.4610
## cityPhoenix       -1.692e+07  2.350e+07  -0.720   0.4717
## cityPortland       8.189e+06  1.126e+07   0.727   0.4674
## cityRichfield     -7.698e+06  1.078e+07  -0.714   0.4756
## citySacramento    -2.021e+07  2.813e+07  -0.718   0.4728
## citySylmar        -1.865e+07  2.621e+07  -0.711   0.4771
## cityWebster       -1.469e+07  2.031e+07  -0.723   0.4698
## cityWichita       -1.157e+07  1.585e+07  -0.730   0.4657
## stateAL                   NA         NA      NA       NA
## stateAZ                   NA         NA      NA       NA
## stateCA                   NA         NA      NA       NA
## stateCO                   NA         NA      NA       NA
## stateFL                   NA         NA      NA       NA
## stateGA                   NA         NA      NA       NA
## stateIL                   NA         NA      NA       NA
## stateIN                   NA         NA      NA       NA
## stateKS                   NA         NA      NA       NA
## stateKY                   NA         NA      NA       NA
## stateMA                   NA         NA      NA       NA
## stateME                   NA         NA      NA       NA
## stateMN                   NA         NA      NA       NA
## stateMS                   NA         NA      NA       NA
## stateNC                   NA         NA      NA       NA
## stateNH            2.297e+07  3.191e+07   0.720   0.4720
## stateOH                   NA         NA      NA       NA
## stateOK                   NA         NA      NA       NA
## statePA                   NA         NA      NA       NA
## stateTN                   NA         NA      NA       NA
## stateTX                   NA         NA      NA       NA
## stateWI                   NA         NA      NA       NA
## year               6.591e+01  1.895e+02   0.348   0.7281
## bath               4.675e+04  1.748e+04   2.675   0.0077 **
## bed               -1.099e+04  1.149e+04  -0.956   0.3395
## rooms              4.813e+03  3.297e+03   1.460   0.1450
## SqFt               1.227e+02  1.058e+02   1.160   0.2464
## I(SqFt^2)          9.155e-04  1.200e-02   0.076   0.9392
## log(SqFt)          2.232e+04  1.047e+05   0.213   0.8313
## bath:bed          -9.515e+02  4.488e+03  -0.212   0.8322
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 93750 on 549 degrees of freedom
## Multiple R-squared:  0.8043, Adjusted R-squared:  0.7869
## F-statistic: 46.06 on 49 and 549 DF,  p-value: < 2.2e-16
```

2. CART Model (rpart)

```
# Parameter Tuning: Use a grid search to try different values for parameters like cp (complexity parame
# This can help reduce overfitting or underfitting.
# Pruning: Automatically prune the tree at the optimal cp level.

cart_model_improved <- rpart(price ~ ., data = train_data, method = "anova",
                             control = rpart.control(cp = 0.01, maxdepth = 10))
# Prune the tree
optimal_cp <- cart_model_improved$cptable[which.min(cart_model_improved$cptable[,"xerror"]), "CP"]
pruned_cart_model <- prune(cart_model_improved, cp = optimal_cp)
rpart.plot(pruned_cart_model, main = "Optimally Pruned CART Model")
```

Optimally Pruned CART Model



3. Neural Network (nnet)

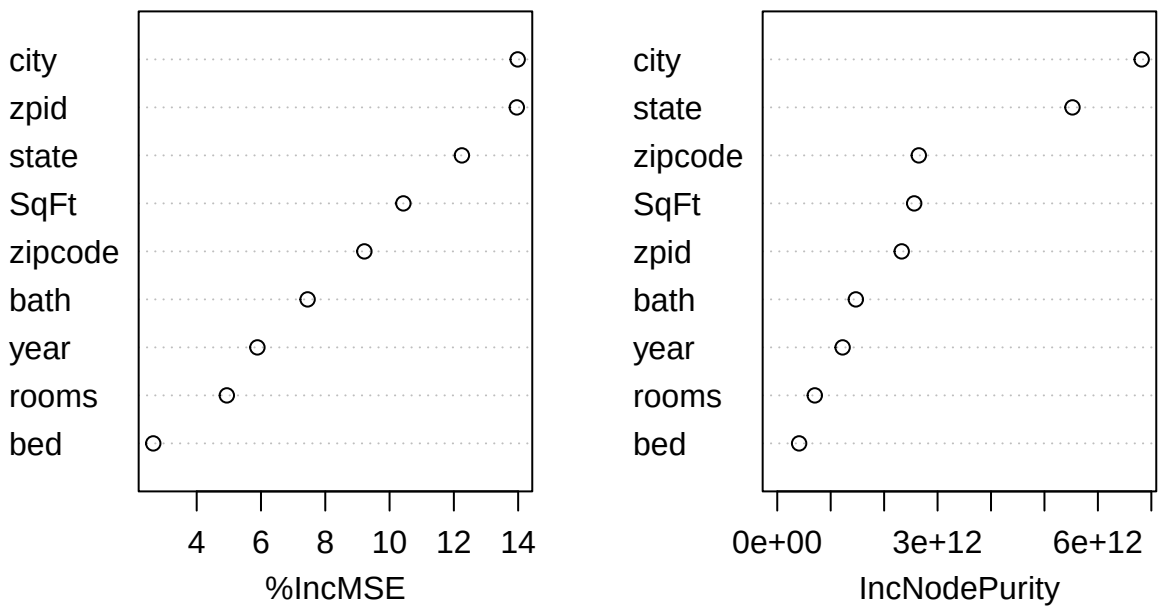
```
# Additional Hidden Layers: Adding layers and nodes can improve
# the model's capacity to capture complex relationships.
# Parameter Tuning: Adjust parameters like size (number of neurons),
# decay (weight decay for regularization), and maxit (max iterations).
# Standardize Features: Neural networks benefit from standardized data,
# so scaling features before training may help.

# Standardize features
preProcess <- preProcess(train_data, method = c("center", "scale"))
train_data_scaled <- predict(preProcess, train_data)
```

```r
test_data_scaled <- predict(preProcess, test_data)

# Train a neural network with modified parameters
nn_model_improved <- nnet(price ~ ., data = train_data_scaled, size = 5, decay = 0.1, linout = TRUE, ma
```

```
## # weights:  346
## initial  value 744.449165
## iter  10 value 208.125910
## iter  20 value 133.086883
## iter  30 value 104.121291
## iter  40 value 83.232779
## iter  50 value 73.163249
## iter  60 value 65.935558
## iter  70 value 61.277463
## iter  80 value 58.803711
## iter  90 value 56.873795
## iter 100 value 55.803968
## iter 110 value 55.002513
## iter 120 value 54.770205
## iter 130 value 54.664755
## iter 140 value 54.560944
## iter 150 value 54.430944
## iter 160 value 54.294034
## iter 170 value 54.204364
## iter 180 value 54.135168
## iter 190 value 54.065778
## iter 200 value 53.996695
## iter 210 value 53.967613
## iter 220 value 53.916960
## iter 230 value 53.857852
## iter 240 value 53.830456
## iter 250 value 53.819369
## iter 260 value 53.817280
## iter 270 value 53.815392
## iter 280 value 53.814912
## iter 290 value 53.814818
## iter 300 value 53.814717
## iter 310 value 53.814660
## final  value 53.814658
## converged
```

4. Random Forest as an Alternative to CART

```r
# Random Forest is a tree-based method that uses an ensemble
# of decision trees to improve model accuracy and robustness.
# It is typically less prone to overfitting than a single CART model.

# Train a random forest model
rf_model <- randomForest(price ~ ., data = train_data, ntree = 100, mtry = 3, importance = TRUE)
# Evaluate feature importance
varImpPlot(rf_model, main = "Random Forest Variable Importance")
```



5. Support Vector Regression (SVR)

```
# SVR can be effective for regression, especially with a mix
# of continuous and categorical features.
# It can capture non-linear relationships through kernel functions.

# Train a Support Vector Regression model
svr_model <- svm(price ~ ., data = train_data, kernel = "radial", cost = 1, gamma = 0.1)
```

Evaluation function

```
evaluate_model <- function(model, train_data, test_data, target_variable) {
  # Ensure consistent factor levels between train and test data
  test_data$city <- factor(test_data$city, levels = levels(train_data$city))
  test_data$state <- factor(test_data$state, levels = levels(train_data$state))

  # Predict on training data
  train_predictions <- predict(model, train_data)

  # Predict on test data
  test_predictions <- predict(model, test_data)

  # Calculate RMSE and MAE for training data
  train_rmse <- rmse(train_data[[target_variable]], train_predictions)
  train_mae <- mae(train_data[[target_variable]], train_predictions)

  # Calculate RMSE and MAE for test data
```

```
  test_rmse <- rmse(test_data[[target_variable]], test_predictions)
  test_mae <- mae(test_data[[target_variable]], test_predictions)

  # Return a list of metrics
  return(list(train_rmse = train_rmse, train_mae = train_mae,
              test_rmse = test_rmse, test_mae = test_mae))
}
```

Evaluating each model

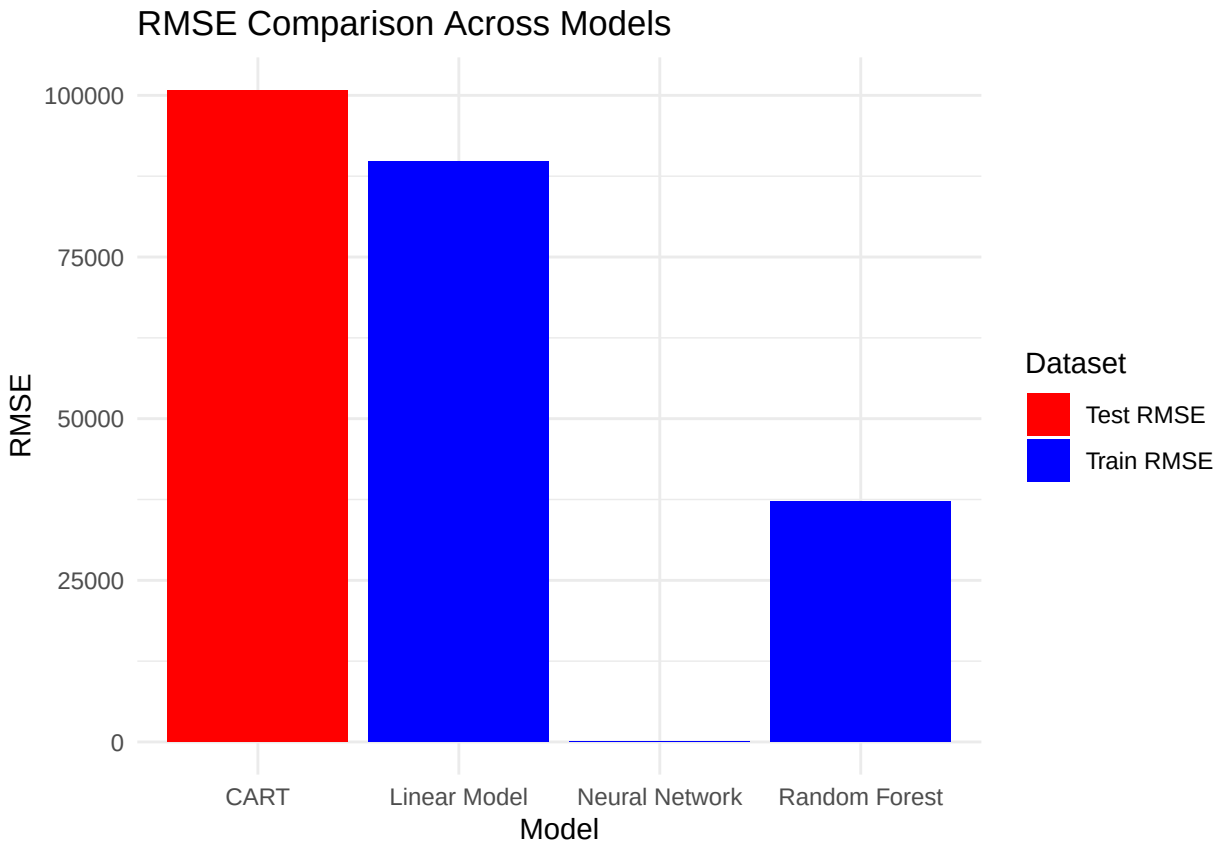
```
# Assuming models are already trained
lm_results <- evaluate_model(lm_model_improved, train_data, test_data, "price")
cart_results <- evaluate_model(pruned_cart_model, train_data, test_data, "price")
nn_results <- evaluate_model(nn_model_improved, train_data_scaled, test_data_scaled, "price")
rf_results <- evaluate_model(rf_model, train_data, test_data, "price")

# Organize results into a data frame for easier comparison
results_df <- data.frame(
  Model = c("Linear Model", "CART", "Neural Network", "Random Forest"),
  Train_RMSE = c(lm_results$train_rmse, cart_results$train_rmse, nn_results$train_rmse, rf_results$trai
  Test_RMSE = c(lm_results$test_rmse, cart_results$test_rmse, nn_results$test_rmse, rf_results$test_rms
  Train_MAE = c(lm_results$train_mae, cart_results$train_mae, nn_results$train_mae, rf_results$train_ma
  Test_MAE = c(lm_results$test_mae, cart_results$test_mae, nn_results$test_mae, rf_results$test_mae)
)
print(results_df)
```

```
##            Model   Train_RMSE Test_RMSE    Train_MAE Test_MAE
## 1   Linear Model 8.975672e+04        NA 5.629530e+04       NA
## 2           CART 8.254379e+04  100828.4 5.294814e+04  64653.4
## 3 Neural Network 2.549733e-01        NA 1.547593e-01       NA
## 4  Random Forest 3.726755e+04        NA 1.930358e+04       NA
```

Visualizing the results

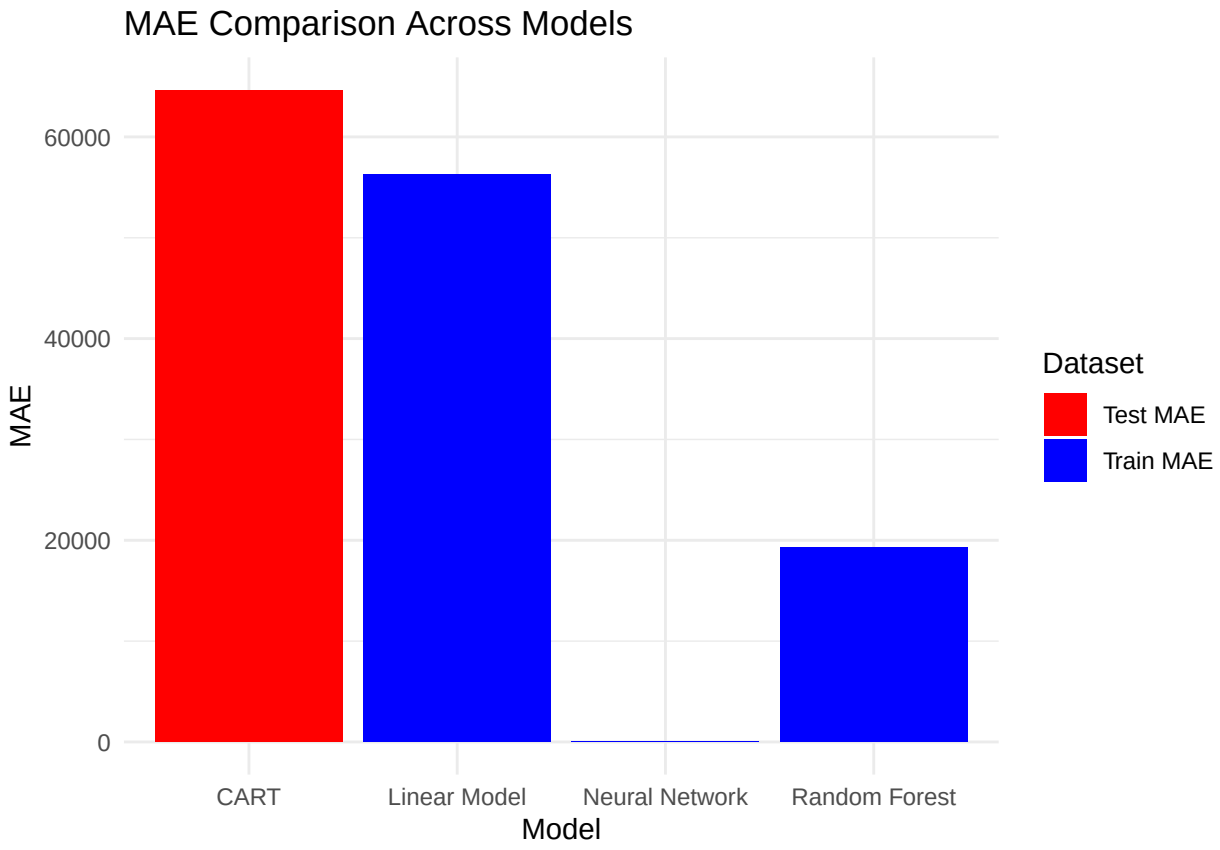
```
library(ggplot2)

# RMSE Comparison
ggplot(results_df, aes(x = Model)) +
  geom_bar(aes(y = Train_RMSE, fill = "Train RMSE"), stat = "identity", position = position_dodge()) +
  geom_bar(aes(y = Test_RMSE, fill = "Test RMSE"), stat = "identity", position = position_dodge()) +
  labs(title = "RMSE Comparison Across Models", y = "RMSE") +
  scale_fill_manual(name = "Dataset", values = c("Train RMSE" = "blue", "Test RMSE" = "red")) +
  theme_minimal()
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale
## range (`geom_bar()`).
```




```
# MAE Comparison
ggplot(results_df, aes(x = Model)) +
  geom_bar(aes(y = Train_MAE, fill = "Train MAE"), stat = "identity", position = position_dodge()) +
  geom_bar(aes(y = Test_MAE, fill = "Test MAE"), stat = "identity", position = position_dodge()) +
  labs(title = "MAE Comparison Across Models", y = "MAE") +
  scale_fill_manual(name = "Dataset", values = c("Train MAE" = "blue", "Test MAE" = "red")) +
  theme_minimal()
```

```
## Warning: Removed 3 rows containing missing values or values outside the scale
## range (`geom_bar()`).
```


MAE Comparison Across Models
60000
40000
20000
0
MAE
CART
Linear Model
Neural Network
Random Forest
Model
Dataset
Test MAE
Train MAE