DC Housing Price Prediction with Deep Learning and LIME

Ryan Mannix

2025-06-13

Contents


```
# Calling libraries & initializing h2o
library(h2o)
library(dplyr)
library(ggplot2)
library(stringr)
library(reshape2)
library(lime)

# Initialize H2O
h2o.init()
```

1.Data Preprocessing

Loading and Cleaning the Data

```
# Load the DC housing data
frame <- read.csv("DC_PropertieResidentialunder1mill.csv")
glimpse(frame) # View a summary of the dataset (10,617 rows, 108 columns)

# Filter out rows with NA values in column "X"
frame_filt <- frame %>% filter(!(is.na(frame$X)))
```

Feature Engineering

```
# Convert house condition (CNDTN) from character to numeric (1-6 scale)
frame_filt <- frame_filt %>%
  mutate(
    CONDITION = case_when(
      CNDTN == "Poor" ~ 1,
      CNDTN == "Fair" ~ 2,
      CNDTN == "Average" ~ 3,
      CNDTN == "Good" ~ 4,
      CNDTN == "Very Good" ~ 5,
      CNDTN == "Excellent" ~ 6
  )
  )

# Round Latitude and Longitude to 3 decimal places for easier grouping
frame_filt$LONGITUDE <- round(frame_filt$X,3)
frame_filt$LATITUDE <- round(frame_filt$Y,3)

# Extract 4-digit code from Census Block column
pattern <- " (\\w+)"
frame_filt<- frame_filt %>%
  mutate(
    CENSUS_BLOCK_4Digit = str_extract(CENSUS_BLOCK, pattern)
  )
frame_filt$CENSUS_BLOCK_4Digit <- as.numeric(frame_filt$CENSUS_BLOCK_4Digit)

# Convert SQUARE column to numeric type
frame_filt$SQUARE <- as.numeric(frame_filt$SQUARE)
```

Splitting the Data into Train and Test Sets

```
# Load the pre-sampled 70% training indices
load("ind2.RData") #ind <- sample(1:nrow(frame_filt), (0.7*nrow(frame_filt)))
set.seed(42)
train <- frame_filt[ind,]
test<- frame_filt[-ind,]
```

2. Price by Categorical Columns

(Converting categorical character columns into useful numerical columns)

Average Price by External Walls, Roofing, and Floor Types

```
# Add average price by EXTWALL, ROOF, and INTWALL columns
# These are added in a similar manner, with each categorical variable grouped and summarized based on P
# The tables are then merged with train and test datasets
```

```
EXTWALL_Table <- test %>% group_by(EXTWALL) %>% summarize(Avg_Price_By_EXTWALL = mean(PRICE))
EXTWALL_Table <- EXTWALL_Table %>% arrange(Avg_Price_By_EXTWALL)
EXTWALL_Table$EXTWALL_INDEX <- c(1:nrow(EXTWALL_Table))
train <- train %>% left_join(EXTWALL_Table[,c(1,3)], by="EXTWALL")
test <- test %>% left_join(EXTWALL_Table[,c(1,3)], by="EXTWALL")

ROOF_Table <- train %>% group_by(ROOF) %>% summarize(Avg_Price_By_ROOF = mean(PRICE))
ROOF_Table <- ROOF_Table %>% arrange(Avg_Price_By_ROOF)
ROOF_Table$ROOF_INDEX <- c(1:nrow(ROOF_Table))
train <- train %>% left_join(ROOF_Table[,c(1,3)], by="ROOF")
test <- test %>% left_join(ROOF_Table[,c(1,3)], by="ROOF")

INTWALL_Table <- train %>% group_by(INTWALL) %>% summarize(Avg_Price_By_INTWALL = mean(PRICE))
INTWALL_Table <- INTWALL_Table %>% arrange(Avg_Price_By_INTWALL)
INTWALL_Table$INTWALL_INDEX <- c(1:nrow(INTWALL_Table))
train <- train %>% left_join(INTWALL_Table[,c(1,3)], by="INTWALL")
test <- test %>% left_join(INTWALL_Table[,c(1,3)], by="INTWALL")
```

Price Information by Location

(Neighborhood, Sub-Neighborhood, Census Block, Latitude/Longitude, and Ward)

```
# Add neighborhood-based price information. (Test price information by neighborhood derived from traini
Assess_NBHD_Table <- train %>% group_by(ASSESSMENT_NBHD) %>%
  summarize(
    Avg_Price_By_NBHD = mean(PRICE),
    Sd_Train_Price_By_NBHD = sd(PRICE),
    Count_By_NBHD = n()
  )
train <- train %>% left_join(Assess_NBHD_Table, by="ASSESSMENT_NBHD")
test <- test %>% left_join(Assess_NBHD_Table, by="ASSESSMENT_NBHD")


# Add sub-neighborhood-based average price information. Filled in NAs with neighborhood average prices.
Assess_SUBNBHD_Table <- train %>% group_by(ASSESSMENT_SUBNBHD) %>% summarize(Avg_Price_By_SUBNBHD = mean
train <- train %>% left_join(Assess_SUBNBHD_Table, by="ASSESSMENT_SUBNBHD")
Train_INDEX <- c(1:nrow(train))
SUBNBHD_NAs <- Train_INDEX[train$ASSESSMENT_SUBNBHD == ""]
ASSESSMENT_NBHD <- train$ASSESSMENT_NBHD[SUBNBHD_NAs]
NBHD_forSUBNBHDNAs_MeanPrice <- data.frame(ASSESSMENT_NBHD) %>% left_join(Assess_NBHD_Table, by="ASSESSM
train[SUBNBHD_NAs,"Avg_Price_By_SUBNBHD"] <- NBHD_forSUBNBHDNAs_MeanPrice$Avg_Price_By_NBHD

test <- test %>% left_join(Assess_SUBNBHD_Table, by="ASSESSMENT_SUBNBHD")
Test_INDEX <- c(1:nrow(test))
SUBNBHD_NAs <- c(Test_INDEX[is.na(test$Avg_Price_By_SUBNBHD)], Test_INDEX[test$ASSESSMENT_SUBNBHD == ""]
ASSESSMENT_NBHD <- test$ASSESSMENT_NBHD[SUBNBHD_NAs]
NBHD_forSUBNBHDNAs_MeanPrice <- data.frame(ASSESSMENT_NBHD) %>% left_join(Assess_NBHD_Table, by="ASSESSM
test[SUBNBHD_NAs,"Avg_Price_By_SUBNBHD"] <- NBHD_forSUBNBHDNAs_MeanPrice$Avg_Price_By_NBHD


# Add price by census block, filling missing values with census tract prices.
Census_Block_Table <- train %>% group_by(CENSUS_TRACT,CENSUS_BLOCK_4Digit) %>% summarize(Avg_Train_Price
```

```
train <- train %>% left_join(Census_Block_Table, by=c("CENSUS_TRACT","CENSUS_BLOCK_4Digit"))
test <- test %>% left_join(Census_Block_Table, by=c("CENSUS_TRACT","CENSUS_BLOCK_4Digit"))
Test_INDEX <- c(1:nrow(test))
CB_NAs <- Test_INDEX[is.na(test$Avg_Train_Price_By_CENSUS_BLOCK)]
CENSUS_TRACT <- test$CENSUS_TRACT[CB_NAs]

Census_Tract_Table <- train %>% group_by(CENSUS_TRACT) %>% summarize(Avg_Train_Price_By_CENSUS_TRACT = m
CT_forCBNAs_MeanPrice <- data.frame(CENSUS_TRACT) %>% left_join(Census_Tract_Table, by="CENSUS_TRACT")
test[CB_NAs,"Avg_Train_Price_By_CENSUS_BLOCK"] <- CT_forCBNAs_MeanPrice$Avg_Train_Price_By_CENSUS_TRACT
test[is.na(test$Avg_Train_Price_By_CENSUS_BLOCK),"Avg_Train_Price_By_CENSUS_BLOCK"] <- test[is.na(test$A


# Add price by latitude/longitude. Since test is based on training data some test latitudes/longitudes
LAT_LONG_Table <- train %>% group_by(LONGITUDE,LATITUDE) %>% summarize(Avg_Train_Price_By_LAT_LONG = mea
train <- train %>% left_join(LAT_LONG_Table, by= c("LONGITUDE", "LATITUDE"))
test <- test %>% left_join(LAT_LONG_Table, by= c("LONGITUDE", "LATITUDE"))
test_LAT_LONG_NAs <- test[is.na(test$Avg_Train_Price_By_LAT_LONG), c("LONGITUDE", "LATITUDE", "Avg_Train
test_LAT_LONG_NAs$Sum_Of_Squares <- test_LAT_LONG_NAs$LONGITUDE^2 + test_LAT_LONG_NAs$LATITUDE^2
LAT_LONG_Table$Sum_Of_Squares <- LAT_LONG_Table$LONGITUDE^2 + LAT_LONG_Table$LATITUDE^2
for (i in 1:nrow(test_LAT_LONG_NAs)) {
  Value_If_Above <- test_LAT_LONG_NAs$Sum_Of_Squares[i] + min(abs(LAT_LONG_Table$Sum_Of_Squares - test_L
  Value_If_Below <- test_LAT_LONG_NAs$Sum_Of_Squares[i] - min(abs(LAT_LONG_Table$Sum_Of_Squares - test_L
  test_LAT_LONG_NAs$Avg_Train_Price_By_LAT_LONG[i] <- LAT_LONG_Table$Avg_Train_Price_By_LAT_LONG[LAT_LO
}
test[rownames(test_LAT_LONG_NAs),"Avg_Train_Price_By_LAT_LONG"]<- test_LAT_LONG_NAs$Avg_Train_Price_By_L

#Add ward-based average price, standard deviation, and count information.
Ward_Table <- train %>% group_by(Ward) %>%
  summarize(
    Avg_Train_Price_By_Ward = mean(PRICE),
    Sd_Train_Price_by_Ward = sd(PRICE),
    Count_by_Ward = n()
  )
train <- train %>% left_join(Ward_Table, by="Ward")
test <- test %>% left_join(Ward_Table, by="Ward")
```

Condition-based Price Percentage Relative to Average Price

```
# Condition-based price percentage relative to average price
CNDTN_Table <- train %>% group_by(CNDTN) %>%
  summarize(
    count_by_CNDTN = n(),
    Avg_Train_Price_By_CNDTN = mean(PRICE)
  )
CNDTN_Table$DC_Train_Mean <- mean(train$PRICE)
CNDTN_Table$CNDTN_Prcnt_Avg <- CNDTN_Table$Avg_Train_Price_By_CNDTN / CNDTN_Table$DC_Train_Mean
train <- train %>% left_join(CNDTN_Table[,c(1,5)], by="CNDTN")
test <- test %>% left_join(CNDTN_Table[,c(1,5)], by="CNDTN")

# Add CNDTN_Prcnt_Avg_By_QUADRANT- House "Condition" expressed as a percentage of average price by quad
QUADRANT_CNDTN_Table <- train %>% group_by(QUADRANT,CNDTN) %>%
  summarize(
```

```
    count_by_QUADRANT_CNDTN = n(),
    Avg_Train_Price_By_QUADRANT_CNDTN = mean(PRICE)
  )

QUADRANT_Table <- train %>% group_by(QUADRANT) %>%
  summarize(
    Avg_Train_Price_By_QUADRANT = mean(PRICE),
  )
QUADRANT_CNDTN_Table <- QUADRANT_CNDTN_Table %>% left_join(QUADRANT_Table, by="QUADRANT")
QUADRANT_CNDTN_Table$CNDTN_Prcnt_Avg_By_QUADRANT <- round(QUADRANT_CNDTN_Table$Avg_Train_Price_By_QUADRA
train <- train %>% left_join(QUADRANT_CNDTN_Table[,c(1,2,6)], by=c("QUADRANT", "CNDTN"))
test <- test %>% left_join(QUADRANT_CNDTN_Table[,c(1,2,6)], by=c("QUADRANT","CNDTN"))
```

3. Selecting & Preparing Data for Deep Learning Model

```
# Remove unnecessary columns and prepare the features for modeling
train_NN <- train %>% select(-c("LONGITUDE":"CENSUS_BLOCK_4Digit", "ASSESSMENT_NBHD":"CENSUS_BLOCK", "QU
test_NN <- test %>% select(-c("LONGITUDE":"CENSUS_BLOCK_4Digit", "ASSESSMENT_NBHD":"CENSUS_BLOCK", "QUA

train_NN_h2o <- as.h2o(train_NN)
test_NN_h2o<- as.h2o(test_NN)

# List of dependent variables for model to use.
X_list <- setdiff(colnames(train_NN),"PRICE")
```

4. Model Training with H2O Deep Learning

Grid Search for Optimal Model

```
hyper_params <- list(
  activation= c("Rectifier", "Tanh"),
  hidden = list(c(9,6,4), c(36,18,9)),
  epochs = c(30, 500),
  l1= c(0,.01),
  l2= c(0,.01)
)

#Run grid search
dl_grid_12 <- h2o.grid(algorithm = "deeplearning",
                       x=X_list,
                       y="PRICE",
                       training_frame = train_NN_h2o,
                       validation_frame = test_NN_h2o,
                       nfolds=5,
                       adaptive_rate= TRUE,
                       #epochs= 30,
                       standardize=TRUE,
                       hyper_params = hyper_params,
```

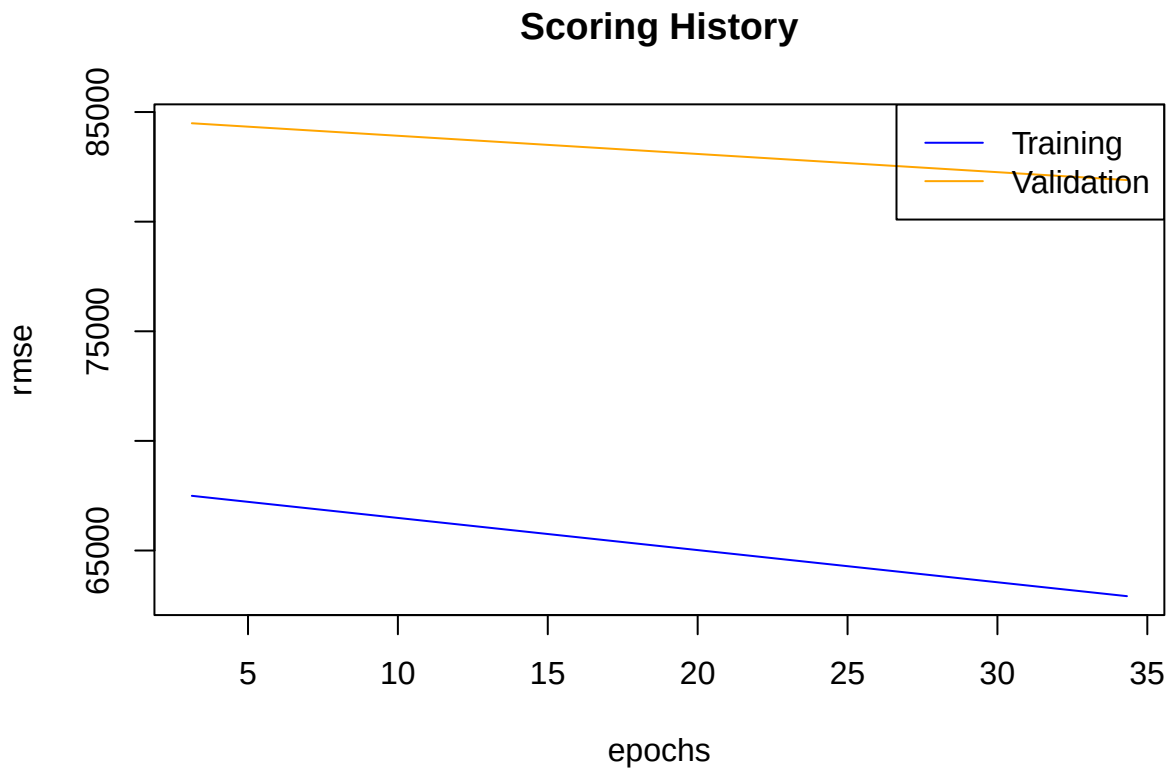
```
                    seed = 33
                    )
```

Retrieve & Review Best Model

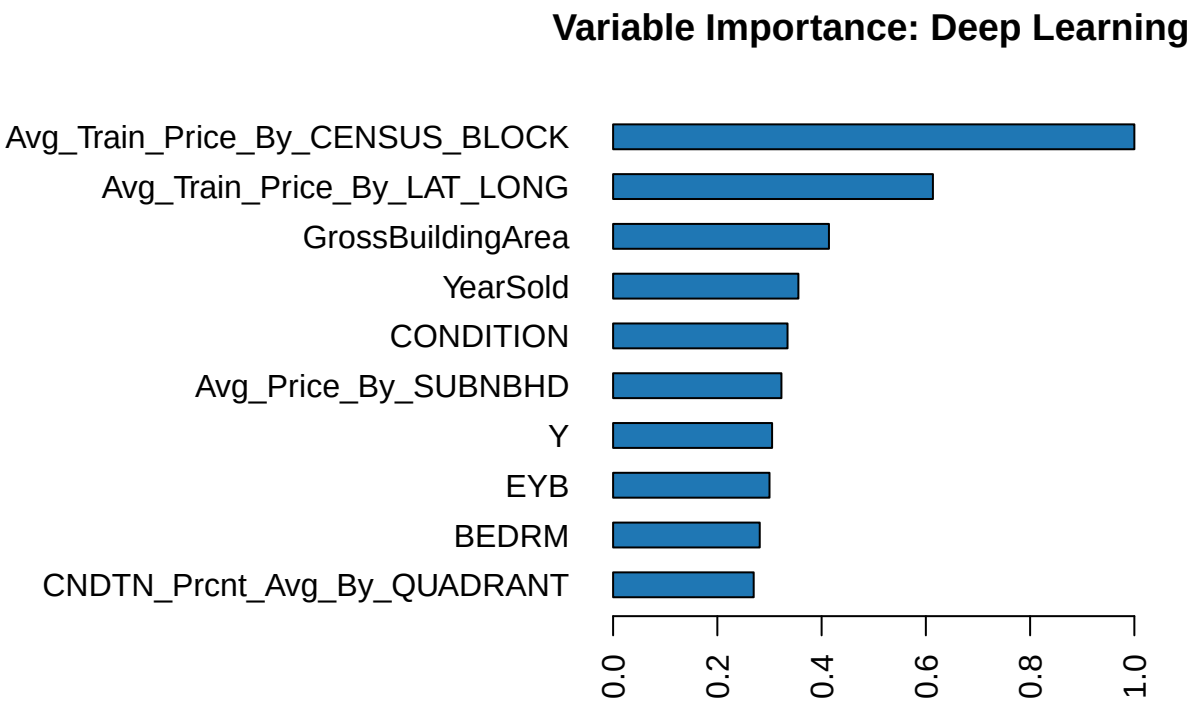
```
# Retrieve the best model from grid search
grid <- h2o.getGrid(grid_id = dl_grid_12@grid_id, sort_by = "mae")
best_model_id <- grid@model_ids[[1]]
best_model <- h2o.getModel(best_model_id)

# Review model
print(best_model)
summary(best_model)
```

Model RMSE vs. Epochs, Variable Importance Plots

```
plot(best_model)
```




```
h2o.varimp_plot(best_model, 10)
```



5. Model Predictions & Evaluation

Train Set Mean Error / Prediction Price vs. Actual Price

```
#Training Set Predictions / Evaluation
train_pred_df <- as.data.frame(h2o.predict(best_model,train_NN_h2o))
```

```
##   |                                                                      |
```

```
actual_train <- as.vector(as.data.frame(train_NN_h2o)$PRICE)

# MAE for Test Set
mean_error_train <- mean(abs(train_pred_df$predict - actual_train))

# RMSE for Test Set
rmse_train <- sqrt(mean((train_pred_df$predict - actual_train)^2))

# R² for Test Set
sst_train <- sum((actual_train - mean(actual_train))^2)
sse_train <- sum((train_pred_df$predict - actual_train)^2)
r2_train <- 1 - sse_train/sst_train
```

```r
#Print train results clearly
cat("Train MAE:", round(mean_error_train,2), "\n")
```

```
## Train MAE: 45636.21
```

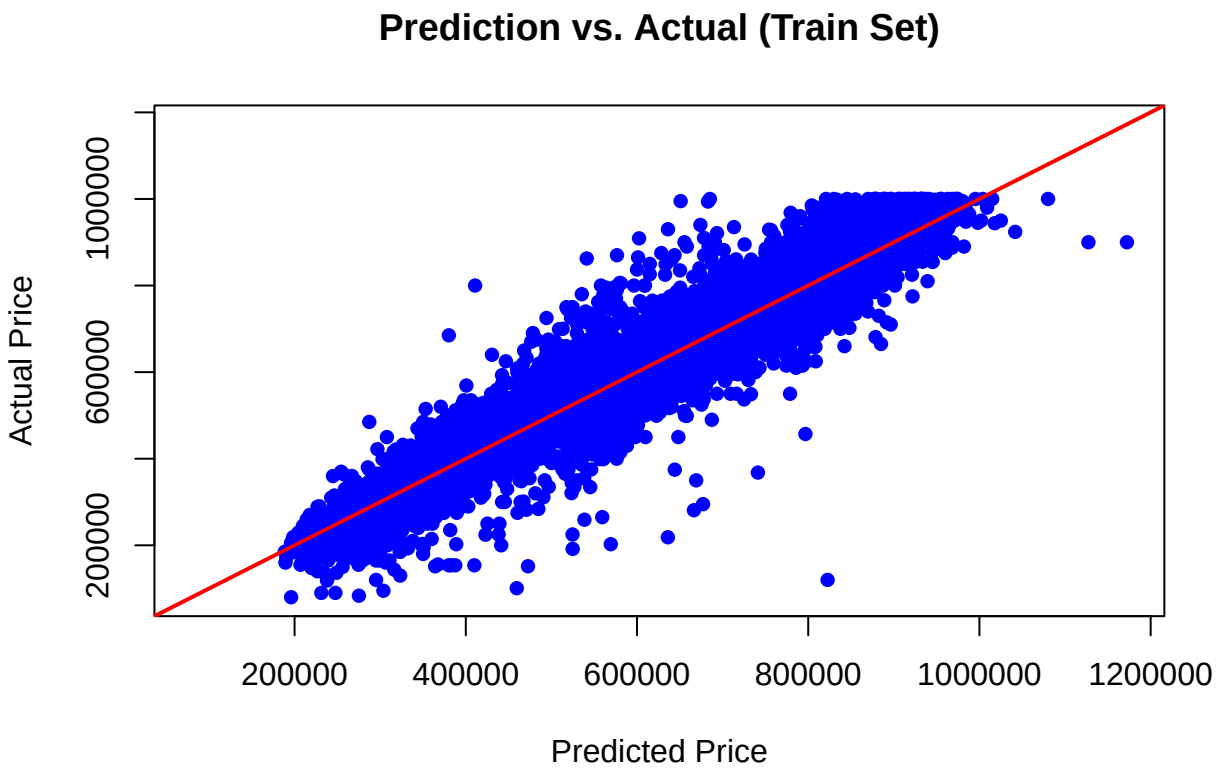
```r
cat("Train RMSE:", round(rmse_train,2), "\n")
```

```
## Train RMSE: 62918.04
```

```r
cat("Train R²:", round(r2_train,2), "\n")
```

```
## Train R²: 0.92
```

```r
# Combine actual and predicted values to set axis limits appropriately
all_vals_train <- c(actual_train, train_pred_df$predict)

# Plot predicted vs. actual prices for the test set
plot(train_pred_df$predict, actual_train,
     main = "Prediction vs. Actual (Train Set)",
     xlab = "Predicted Price",
     ylab = "Actual Price",
     pch=16, col="blue",
     xlim= range(all_vals_train),
     ylim= range(all_vals_train))

# Add the reference 1:1 line
abline(0,1,lwd=2,col='red')
```



Test Set Mean Error / Prediction Price vs. Actual Price

```
#Test Set Predictions / Evaluation
test_pred_df <- as.data.frame(h2o.predict(best_model,test_NN_h2o))
```

```
##   |                                                                      |
```

```
actual_test <- as.vector(as.data.frame(test_NN_h2o)$PRICE)

# MAE for Test Set
mean_error_test <- mean(abs(test_pred_df$predict - actual_test))

# RMSE for Test Set
rmse_test <- sqrt(mean((test_pred_df$predict - actual_test)^2))

# R² for Test Set
sst_test <- sum((actual_test - mean(actual_test))^2)
sse_test <- sum((test_pred_df$predict - actual_test)^2)
r2_test <- 1 - sse_test / sst_test

# Print test results clearly
cat("Test MAE:", round(mean_error_test,2), "\n")
```

```
## Test MAE: 59089.03
```

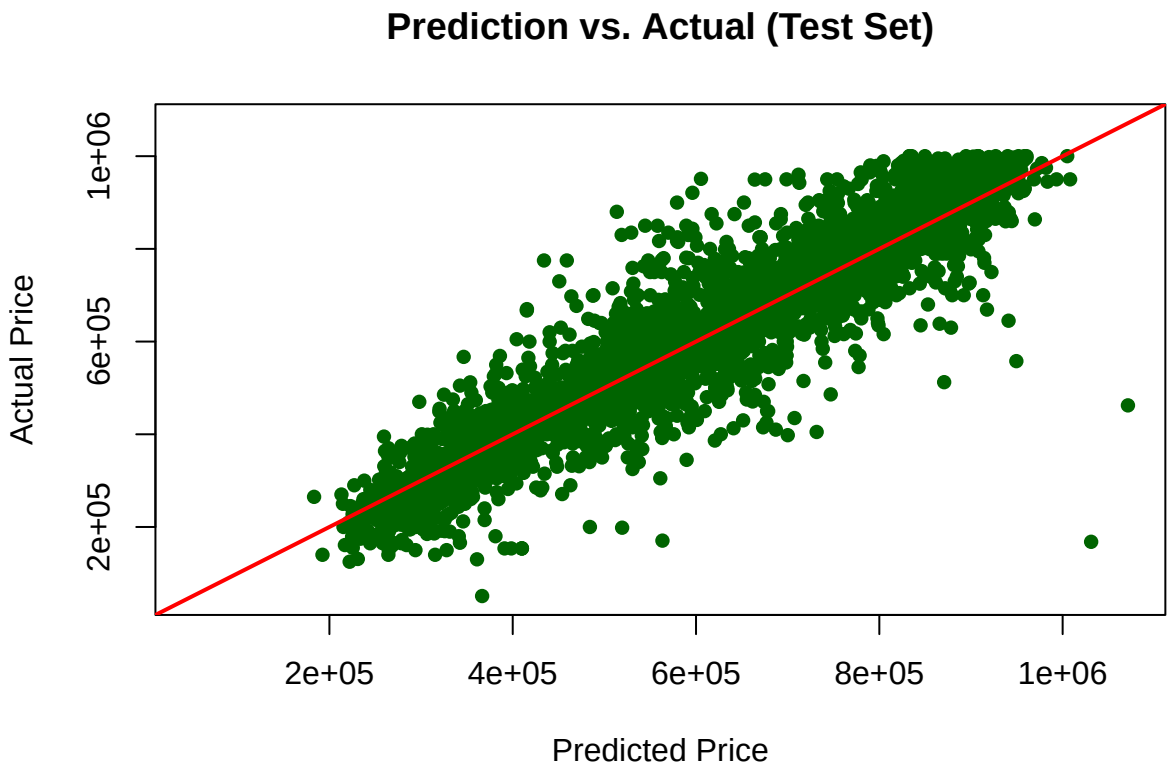
```r
cat("Test RMSE:", round(rmse_test,2), "\n")
```

```
## Test RMSE: 81898.83
```

```r
cat("Test R²:", round(r2_test,2), "\n")
```

```
## Test R²: 0.86
```

```r
# Combine actual and predicted values to set axis limits appropriately
all_vals_test <- c(actual_test, test_pred_df$predict)


# Plot predicted vs. actual prices for the test set
plot(test_pred_df$predict, actual_test,
     main = "Prediction vs. Actual (Test Set)",
     xlab = "Predicted Price",
     ylab = "Actual Price",
     pch=16, col="darkgreen",
     xlim= range(all_vals_test),
     ylim= range(all_vals_test))

# Add the reference 1:1 line
abline(0,1,lwd=2,col='red')
```



6. LIME for Model Explanation

Local Interpretable Model-agnostic Explanations (LIME) is a technique that approximates any black box machine learning model with a local, interpretable model to explain each individual prediction.

Custom functions for LIME compatibility with H2O regression model

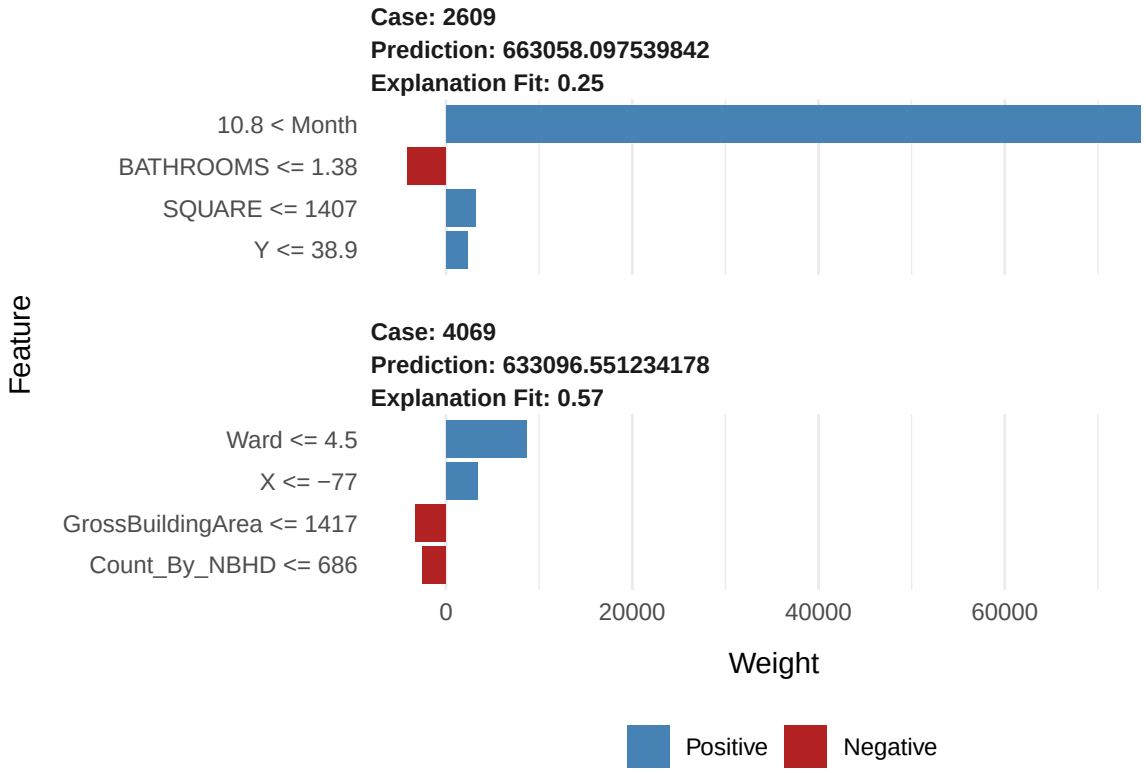
```
#Ensure LIME recognizes the H2O model as a regression model
model_type.H2ORegressionModel <- function(x, ...) "regression"

#Define how LIME should get predictions from the H2O model
predict_model.H2ORegressionModel <- function(x, newdata, ...){
  as.data.frame(h2o.predict(x, as.h2o(newdata)))
}
```

Applying LIME

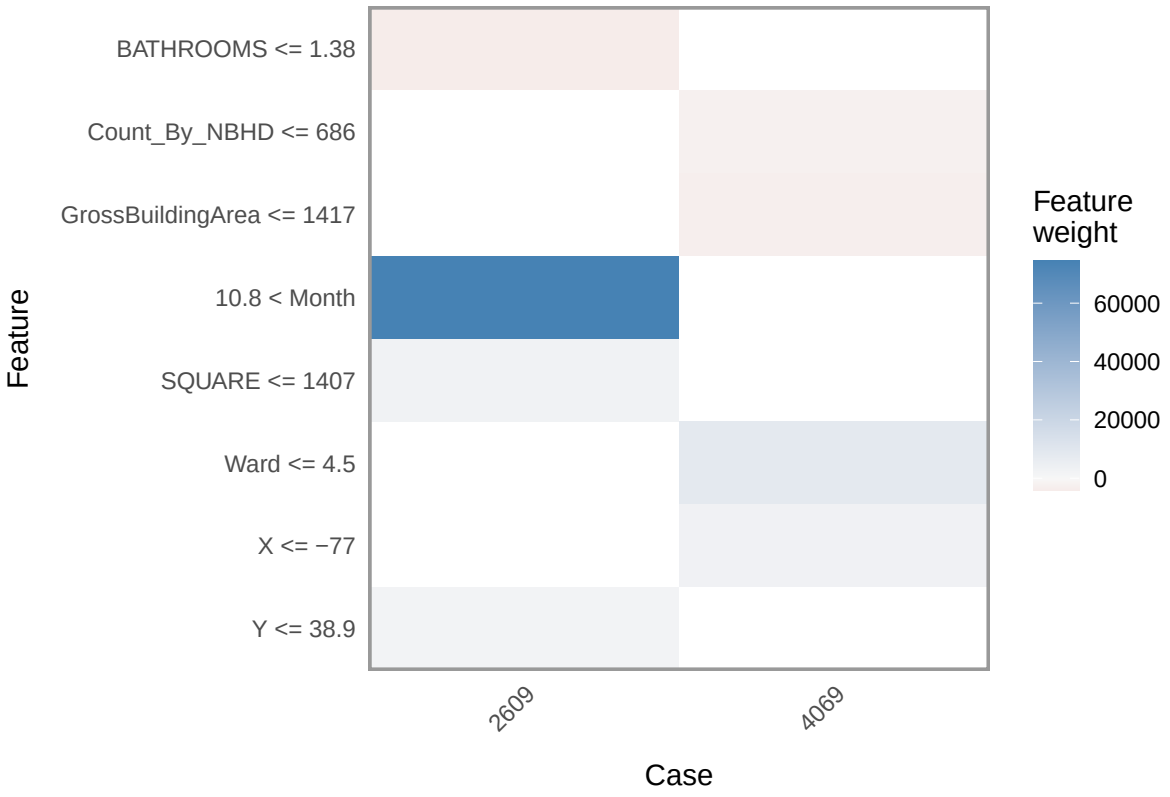
```
# Selecting 2 sample rows
for_lime <- sample(1:nrow(train_NN), 2)
train_NN_df <- as.data.frame(train_NN)
data_for_lime <- train_NN_df[for_lime,]
print(data_for_lime)

# Reviewing predictions
predict_data_for_lime <- as.h2o(data_for_lime)
predictions_for_lime <- h2o.predict(best_model,predict_data_for_lime)
print(predictions_for_lime)

# Running LIME. Looking at top 5 features.
explainer_price <- lime(data_for_lime, best_model)
explanation <- explain(data_for_lime, explainer_price, n_labels = 2, n_features = 4)
```

Plotting LIME Explanations

```
plot_features(explanation, ncol=1)
```




```
plot_explanations(explanation)
```



7. Conclusion & Insights

Model Performance Summary

The H2O deep learning model was trained on a comprehensive, engineered dataset of DC housing records. It achieved the following performance metrics:

- **Train Set Mean Absolute Error (MAE):** 4.563621×10^4
- **Train Set Root Mean Squared Error (RMSE):** 6.291804×10^4
- **Train Set R^2:** 0.918
- **Test Set Mean Absolute Error (MAE):** 5.908903×10^4
- **Test Set Root Mean Squared Error (RMSE):** 8.189883×10^4
- **Test Set R^2:** 0.859

These results indicate that the model performs reasonably well on unseen data, suggesting good generalization with some variance likely due to location- or condition-based outliers.

LIME Model Interpretability

Using **LIME**, we examined how the model made individual predictions. Key insights:

- The most influential features were often **location-based** (e.g., neighborhood price averages, census block), as well as **house condition** and **structure characteristics**.
- LIME helped validate that the model aligns with housing domain intuition—e.g., houses in neighborhoods with higher average prices predict higher prices, all else equal.

Final Thoughts

This pipeline demonstrates the value of combining deep learning with explainability. By engineering domain-specific features and using tools like LIME, we can not only build accurate predictive models, but also interpret their reasoning—crucial for applications like real estate appraisal, policy evaluation, or property investment.