HARVARD EXTENSION SCHOOL

EXT CSCI E-106 Model Data Class Group Project Template

Will Greaves Flora Lo Matt Michel Thaylan Toth Kaylee Vo Zhenzhen Yin

02 December 2023

Abstract

This is the location for your abstract. It must consist of two paragraphs.

Contents

House Sales in King County, USA data to be used in the Final Project

Variable	Description
id	**Unique ID for each home sold (it is not a predictor)**
date	*Date of the home sale*
price	*Price of each home sold*
bedrooms	*Number of bedrooms*
bathrooms	*Number of bathrooms, where ".5" accounts for a bathroom with a toilet but no shower*
sqft_living	*Square footage of the apartment interior living space*
sqft_lot	*Square footage of the land space*
floors	*Number of floors*
waterfront	*A dummy variable for whether the apartment was overlooking the waterfront or not*
view	*An index from 0 to 4 of how good the view of the property was*
condition	*An index from 1 to 5 on the condition of the apartment,*
grade	*An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 has a high-quality level of construction and design.*
sqft_above	*The square footage of the interior housing space that is above ground level*
sqft_basement	*The square footage of the interior housing space that is below ground level*
yr_built	*The year the house was initially built*
yr_renovated	*The year of the house's last renovation*
zipcode	*What zipcode area the house is in*
lat	*Latitude*
long	*Longitude*
sqft_living15	*The square footage of interior housing living space for the nearest 15 neighbors*
sqft_lot15	*The square footage of the land lots of the nearest 15 neighbors*

Instructions:

0. Join a team with your fellow students with appropriate size (Four Students total)
1. Load and Review the dataset named "KC_House_Sales'csv
2. Create the train data set which contains 70% of the data and use set.seed (1023). The remaining 30% will be your test data set.
3. Investigate the data and combine the level of categorical variables if needed and drop variables as needed. For example, you can drop id, Latitude, Longitude, etc.
4. Build a regression model to predict price.
5. Create scatter plots and a correlation matrix for the train data set. Interpret the possible relationship between the response.
6. Build the best multiple linear models by using the stepwise selection method. Compare the performance of the best two linear models.
7. Make sure that model assumption(s) are checked for the final model. Apply remedy measures (transformation, etc.) that helps satisfy the assumptions.
8. Investigate unequal variances and multicollinearity. If necessary, apply remedial methods (WLS, Ridge, Elastic Net, Lasso, etc.).
9. Build an alternative model based on one of the following approaches to predict price: regression tree, NN, or SVM. Check the applicable model assumptions. Explore using a logistic regression.
10. Use the test data set to assess the model performances from above.
11. Based on the performances on both train and test data sets, determine your primary (champion) model and the other model which would be your benchmark model.
12. Create a model development document that describes the model following this template, input the name of the authors, Harvard IDs, the name of the Group, all of your code and calculations, etc..:

Due Date: December 18th, 2023 at 11:59 pm EST

Notes No typographical errors, grammar mistakes, or misspelled words, use English language All tables need to be numbered and describe their content in the body of the document All figures/graphs need to be numbered and describe their content All results must be accurate and clearly explained for a casual reviewer to fully understand their purpose and impact Submit both the RMD markdown file and PDF with the sections with appropriate explanations. A more formal document in Word can be used in place of the pdf file but must include all appropriate explanations.

Executive Summary

This section will describe the model usage, your conclusions and any regulatory and internal requirements. In a real world scneario, this section is for senior management who do not need to know the details. They need to know high level (the purpose of the model, limitations of the model and any issues).

I. Introduction (5 points)

This section needs to introduce the reader to the problem to be resolved, the purpose, and the scope of the statistical testing applied. What you are doing with your prediction? What is the purpose of the model? What methods were trained on the data, how large is the test sample, and how did you build the model?

In this project, our goal is to build a statistical model that can predict house sales prices in Kings county, USA based on house sales data collected in that area between May 2014 and May 2015. The dataset contains information on 21613 houses, including the sale price, number of rooms, square footage, year built and renovated, view, and condition of the property. The house sale price was used as the outcome variable and all other variables were considered as independent variables. 70% of the dataset was used as training set and 30% was used as testing set.

Using this framework, we built several different models using linear regression, logistic regression, regression tree, and neural network. When building each model, feature selection methods were used and the appropriate diagnostic tests were applied to verify that model assumptions were met.

Finally, the performance of each models was evaluated by its accuracy in predicting house prices in the test set, specifically by examining the MSE of predicted values. Based on that, we propose that the best predictive model is _____. This model indicated that the most important factors that influence house prices in King County, USA are _____.

II. Description of the data and quality (15 points)

Here you need to review your data, the statistical test applied to understand the predictors and the response and how are they correlated. Extensive graph analysis is recommended. Is the data continuous, or categorical, do any transformation needed? Do you need dummies?

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

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

```
library(ggplot2)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

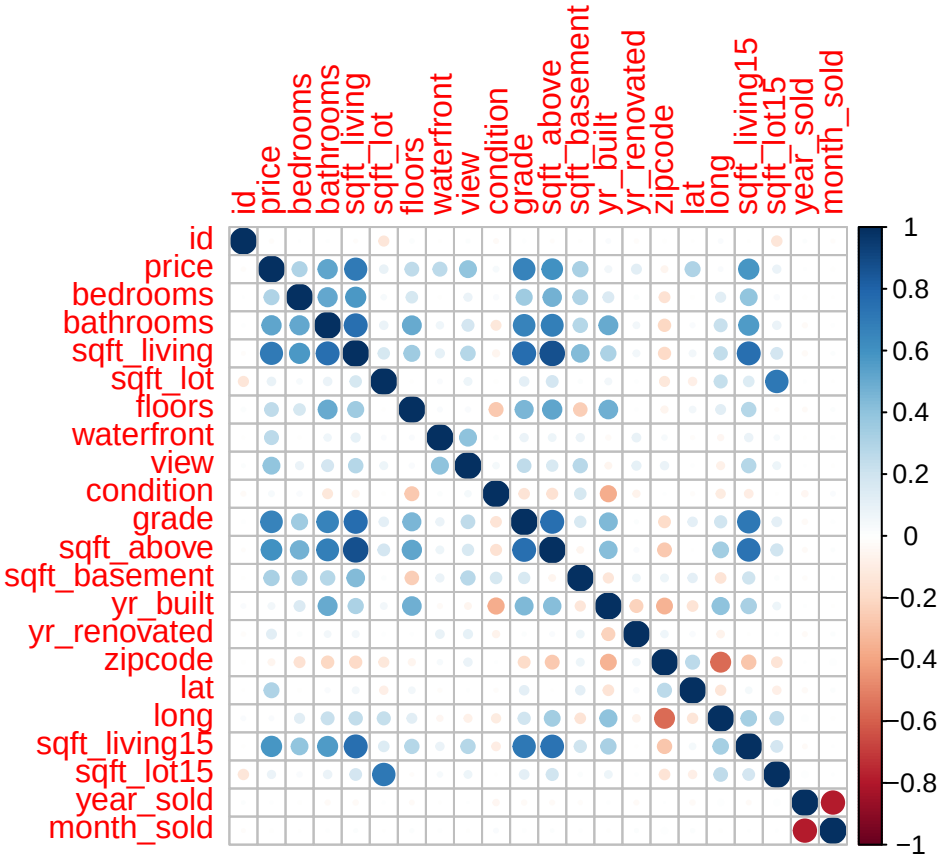
```
# Data cleaning (Separate date into year and month)
HouseSales$year_sold <- as.integer(substring(HouseSales$date, 1, 4))
HouseSales$month_sold <- as.integer(substring(HouseSales$date, 5, 6))
HouseSales$date <-NULL
# Data cleaning (price to numeric)
HouseSales$price<-parse_number(HouseSales$price)
```

```
cor_matrix = round(cor(HouseSales), 3)
cor_matrix
```

```
##                    id  price bedrooms bathrooms sqft_living sqft_lot floors
## id              1.000 -0.017    0.001     0.005      -0.012   -0.132  0.019
## price          -0.017  1.000    0.308     0.525       0.702    0.090  0.257
## bedrooms        0.001  0.308    1.000     0.516       0.577    0.032  0.175
## bathrooms       0.005  0.525    0.516     1.000       0.755    0.088  0.501
## sqft_living    -0.012  0.702    0.577     0.755       1.000    0.173  0.354
## sqft_lot       -0.132  0.090    0.032     0.088       0.173    1.000 -0.005
## floors          0.019  0.257    0.175     0.501       0.354   -0.005  1.000
## waterfront     -0.003  0.266   -0.007     0.064       0.104    0.022  0.024
## view            0.012  0.397    0.080     0.188       0.285    0.075  0.029
## condition      -0.024  0.036    0.028    -0.125      -0.059   -0.009 -0.264
## grade           0.008  0.667    0.357     0.665       0.763    0.114  0.458
## sqft_above     -0.011  0.606    0.478     0.685       0.877    0.184  0.524
## sqft_basement  -0.005  0.324    0.303     0.284       0.435    0.015 -0.246
## yr_built        0.021  0.054    0.154     0.506       0.318    0.053  0.489
## yr_renovated   -0.017  0.126    0.019     0.051       0.055    0.008  0.006
## zipcode        -0.008 -0.053   -0.153    -0.204      -0.199   -0.130 -0.059
## lat            -0.002  0.307   -0.009     0.025       0.053   -0.086  0.050
## long            0.021  0.022    0.129     0.223       0.240    0.230  0.125
## sqft_living15  -0.003  0.585    0.392     0.569       0.756    0.145  0.280
## sqft_lot15     -0.139  0.082    0.029     0.087       0.183    0.719 -0.011
## year_sold       0.010  0.004   -0.010    -0.027      -0.029    0.005 -0.022
## month_sold     -0.012 -0.010   -0.002     0.007       0.012   -0.002  0.014
##                waterfront   view condition  grade sqft_above sqft_basement
## id                 -0.003  0.012    -0.024  0.008     -0.011        -0.005
## price               0.266  0.397     0.036  0.667      0.606         0.324
## bedrooms           -0.007  0.080     0.028  0.357      0.478         0.303
## bathrooms           0.064  0.188    -0.125  0.665      0.685         0.284
## sqft_living         0.104  0.285    -0.059  0.763      0.877         0.435
## sqft_lot            0.022  0.075    -0.009  0.114      0.184         0.015
## floors              0.024  0.029    -0.264  0.458      0.524        -0.246
## waterfront          1.000  0.402     0.017  0.083      0.072         0.081
## view                0.402  1.000     0.046  0.251      0.168         0.277
```

```
## condition          0.017  0.046     1.000 -0.145     -0.158         0.174
## grade              0.083  0.251    -0.145  1.000      0.756         0.168
## sqft_above         0.072  0.168    -0.158  0.756      1.000        -0.052
## sqft_basement      0.081  0.277     0.174  0.168     -0.052         1.000
## yr_built          -0.026 -0.053    -0.361  0.447      0.424        -0.133
## yr_renovated       0.093  0.104    -0.061  0.014      0.023         0.071
## zipcode            0.030  0.085     0.003 -0.185     -0.261         0.075
## lat               -0.014  0.006    -0.015  0.114     -0.001         0.111
## long              -0.042 -0.078    -0.107  0.198      0.344        -0.145
## sqft_living15      0.086  0.280    -0.093  0.713      0.732         0.200
## sqft_lot15         0.031  0.073    -0.003  0.119      0.194         0.017
## year_sold         -0.004  0.001    -0.046 -0.030     -0.024        -0.016
## month_sold         0.008 -0.006     0.022  0.008      0.010         0.006
##                yr_built yr_renovated zipcode    lat   long sqft_living15
## id                0.021       -0.017  -0.008 -0.002  0.021        -0.003
## price             0.054        0.126  -0.053  0.307  0.022         0.585
## bedrooms          0.154        0.019  -0.153 -0.009  0.129         0.392
## bathrooms         0.506        0.051  -0.204  0.025  0.223         0.569
## sqft_living       0.318        0.055  -0.199  0.053  0.240         0.756
## sqft_lot          0.053        0.008  -0.130 -0.086  0.230         0.145
## floors            0.489        0.006  -0.059  0.050  0.125         0.280
## waterfront       -0.026        0.093   0.030 -0.014 -0.042         0.086
## view             -0.053        0.104   0.085  0.006 -0.078         0.280
## condition        -0.361       -0.061   0.003 -0.015 -0.107        -0.093
## grade             0.447        0.014  -0.185  0.114  0.198         0.713
## sqft_above        0.424        0.023  -0.261 -0.001  0.344         0.732
## sqft_basement    -0.133        0.071   0.075  0.111 -0.145         0.200
## yr_built          1.000       -0.225  -0.347 -0.148  0.409         0.326
## yr_renovated     -0.225        1.000   0.064  0.029 -0.068        -0.003
## zipcode          -0.347        0.064   1.000  0.267 -0.564        -0.279
## lat              -0.148        0.029   0.267  1.000 -0.136         0.049
## long              0.409       -0.068  -0.564 -0.136  1.000         0.335
## sqft_living15     0.326       -0.003  -0.279  0.049  0.335         1.000
## sqft_lot15        0.071        0.008  -0.147 -0.086  0.254         0.183
## year_sold         0.004       -0.024   0.001 -0.029  0.000        -0.022
## month_sold       -0.006        0.013   0.000  0.015 -0.008         0.002
##                sqft_lot15 year_sold month_sold
## id                -0.139     0.010     -0.012
## price              0.082     0.004     -0.010
## bedrooms           0.029    -0.010     -0.002
## bathrooms          0.087    -0.027      0.007
## sqft_living        0.183    -0.029      0.012
## sqft_lot           0.719     0.005     -0.002
## floors            -0.011    -0.022      0.014
## waterfront         0.031    -0.004      0.008
## view               0.073     0.001     -0.006
## condition         -0.003    -0.046      0.022
## grade              0.119    -0.030      0.008
## sqft_above         0.194    -0.024      0.010
## sqft_basement      0.017    -0.016      0.006
## yr_built           0.071     0.004     -0.006
## yr_renovated       0.008    -0.024      0.013
## zipcode           -0.147     0.001      0.000
## lat               -0.086    -0.029      0.015
## long               0.254     0.000     -0.008
## sqft_living15      0.183    -0.022      0.002
## sqft_lot15         1.000     0.000      0.004
## year_sold          0.000     1.000     -0.782
## month_sold         0.004    -0.782      1.000
```

```
corrplot(cor_matrix, method = "circle")
```



III. Model Development Process (15 points)

Build a regression model to predict price. And of course, create the train data set which contains 70% of the data and use set.seed (1023). The remaining 30% will be your test data set. Investigate the data and combine the level of categorical variables if needed and drop variables. For example, you can drop id, Latitude, Longitude, etc.

```
set.seed(1023)

response <- "price"

#Dropping variables as specified
drop.vars <- c("id", "lat", "long")
kc.house.df <- dplyr::select(HouseSales, -drop.vars)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use `all_of()` or `any_of()` instead.
##   # Was:
##   data %>% select(drop.vars)
##
##   # Now:
##   data %>% select(all_of(drop.vars))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
#Renaming to conform to unified convention
kc.house.df <- kc.house.df %>% rename("year_built" = "yr_built")
```

```
#Feature Ideas
#Quality Adjusted Features
transform_sqft_adj_grade <- function(kc.house.df){

  sqft_adj_grade <- kc.house.df[, "sqft_living"] / kc.house.df[, "grade"]

  return (sqft_adj_grade)
}

transform_sqft_adj_condition <- function(kc.house.df){

  sqft_adj_condition <- kc.house.df[, "sqft_living"] / kc.house.df[, "condition"]

  return (sqft_adj_condition)
}

transform_sqft_adj_waterfront <- function(kc.house.df){

  sqft_adj_waterfront <- kc.house.df[, "sqft_living"] * kc.house.df[, "waterfront"]

  return (sqft_adj_waterfront)
}

transform_poly_sqft_living <- function(kc.house.df){
  #Center variables for polynomial terms
  sqft_living <- (kc.house.df$sqft_living - mean(kc.house.df$sqft_living))
  sqft_living_squared <- sqft_living^2

  return (list(center = sqft_living, squared = sqft_living_squared))
}

transform_poly_floor <- function(kc.house.df){
```

```
  floors <- (kc.house.df$floors - mean(kc.house.df$floors))
  floors_squared <- floors^2

  return(list(center = floors, squared = floors_squared))
}
```

Centering the variables do not change the interpretation of the coefficients. However they do change the interpretation of the intercept. Instead of the intercept being read as the value of price when all variables are zero. It is not read as price when all variables are zero, `floors` is equal to its average and `sqft_living` is equal to its average.

```
#Apply transformations

kc.house.df$sqft_adj_grade <- transform_sqft_adj_grade(kc.house.df)
kc.house.df$sqft_adj_condition <- transform_sqft_adj_condition(kc.house.df)
kc.house.df$sqft_adj_waterfront <- transform_sqft_adj_waterfront(kc.house.df)

res <- transform_poly_sqft_living(kc.house.df)
kc.house.df$sqft_living <- res$center
kc.house.df$sqft_living_squared <- res$squared

res <- transform_poly_floor(kc.house.df)
kc.house.df$floors <- res$center
kc.house.df$floors_squared <- res$squared

#Remove collinear variables
# sqft_basement + sqft_above = sqft_living
kc.house.df$sqft_basement <- NULL
```

```
set.seed(1023)

#use 70% of dataset as training set and 30% as test set
train_prop <- 0.7

index <- sample(1:nrow(HouseSales), size = round(train_prop * nrow(kc.house.df)))
kc.house.train.X <- kc.house.df[index, -which(names(kc.house.df) %in% c(response))] # modified slightly
kc.house.train.y <- kc.house.df [index, response]

kc.house.test.X <- kc.house.df [-index, -which(names(kc.house.df) %in% c(response))]
kc.house.test.y <- kc.house.df [-index, response]
```

```
#baseline model
baseline.model <- lm(price ~ ., data = cbind(price = kc.house.train.y, kc.house.train.X))
summary(baseline.model)
```

```
##
## Call:
## lm(formula = price ~ ., data = cbind(price = kc.house.train.y,
##     kc.house.train.X))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1869442  -101703   -11735    82023  2456321
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -6.521e+07  1.162e+07  -5.614 2.01e-08 ***
## bedrooms            -1.213e+04  2.241e+03  -5.410 6.41e-08 ***
## bathrooms            4.917e+04  3.776e+03  13.021  < 2e-16 ***
## sqft_living          4.596e+02  3.100e+01  14.828  < 2e-16 ***
## sqft_lot            -5.832e-03  5.883e-02  -0.099    0.921
## floors               3.553e+04  4.901e+03   7.251 4.34e-13 ***
```

```
## waterfront          -2.501e+05  4.636e+04  -5.395 6.94e-08 ***
## view                 3.882e+04  2.517e+03  15.422  < 2e-16 ***
## condition           -2.364e+04  5.712e+03  -4.139 3.51e-05 ***
## grade                4.653e+04  7.799e+03   5.966 2.48e-09 ***
## sqft_above          -2.734e+01  5.268e+00  -5.190 2.13e-07 ***
## year_built          -3.200e+03  8.239e+01 -38.845  < 2e-16 ***
## yr_renovated         2.750e+01  4.253e+00   6.465 1.04e-10 ***
## zipcode             -1.211e+01  3.433e+01  -0.353    0.724
## sqft_living15        5.873e+01  4.084e+00  14.380  < 2e-16 ***
## sqft_lot15          -4.122e-01  8.512e-02  -4.843 1.29e-06 ***
## year_sold            3.651e+04  5.507e+03   6.629 3.50e-11 ***
## month_sold           1.260e+03  8.254e+02   1.527    0.127
## sqft_adj_grade      -2.012e+03  2.063e+02  -9.755  < 2e-16 ***
## sqft_adj_condition  -3.473e+02  3.301e+01 -10.523  < 2e-16 ***
## sqft_adj_waterfront  2.441e+02  1.319e+01  18.512  < 2e-16 ***
## sqft_living_squared  2.484e-02  1.900e-03  13.070  < 2e-16 ***
## floors_squared       4.057e+04  5.586e+03   7.263 3.96e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 196700 on 15106 degrees of freedom
## Multiple R-squared:  0.7046, Adjusted R-squared:  0.7041
## F-statistic:  1637 on 22 and 15106 DF,  p-value: < 2.2e-16
```

IV. Model Performance Testing (15 points)

Use the test data set to assess the model performances. Here, build the best multiple linear models by using the stepwise both ways selection method. Compare the performance of the best two linear models. Make sure that model assumption(s) are checked for the final linear model. Apply remedy measures (transformation, etc.) that helps satisfy the assumptions. In particular you must deeply investigate unequal variances and multicollinearity. If necessary, apply remedial methods (WLS, Ridge, Elastic Net, Lasso, etc.).

#TODO 1. Stepwise feature selection 2. Add plots (e.g. scale location, residual vs fitted) 3. Normality plots

TODO: The following code needs to be updated once we have the updated models from stepwise feature selection.

```
#evaluate on test data

CalcTestMetrics <- function(pred, act, n, p) {
  SST <- var(act)*(length(act)-1)
  SSE <- sum((act-pred)^2)
  SSR <- sum(pred - mean(act))^2
  rsquared <- 1- SSE/SST

  adj.rsquared <- 1 - (((1 - rsquared)*(n-1)) / (n-p-1))
  mse <- sum((act - pred)^2) / (n-p)
  mae <- (sum(abs(act-pred))) / n

  return (
    list(
      adj.rsquared = adj.rsquared,
      rsquared = rsquared,
      mse = mse,
      mae = mae
    )
  )
}

pred <- predict(baseline.model, kc.house.test.X)
act <- kc.house.test.y
n <- dim(model.matrix(baseline.model)[, -1])[1]
p <- dim(model.matrix(baseline.model)[, -1])[2]

cbind(CalcTestMetrics(pred, act, n, p))
```

```
##              [,1]
## adj.rsquared 0.699768
## rsquared     0.7002046
## mse          18533366140
## mae          56665.42
```

V. Challenger Models (15 points)

Build an alternative model based on one of the following approaches to predict price: regression tree, NN, or SVM. Explore using a logistic regression. Check the applicable model assumptions. Apply in-sample and out-of-sample testing, backtesting and review the comparative goodness of fit of the candidate models. Describe step by step your procedure to get to the best model and why you believe it is fit for purpose.

```
# Load libraries
library(caret)
library(rpart)
library(nnet)
library(knitr)
library(dplyr)
library(stringr)
library(readr)

HouseSales <- read.csv('KC_House_Sales.csv')

# Step a: Data Splitting
set.seed(123)  #
split_ratio <- 0.7  # 70% training, 30% testing
index <- sample(1:nrow(HouseSales), size = round(split_ratio * nrow(HouseSales)))
train_data <- HouseSales[index, ]
test_data <- HouseSales[-index, ]

# Step b: Model Building

# Model 1: Logistic Regression

# Create a binary variable 'high_price' based on the chosen percentile
logisticdf <- HouseSales
logisticdf$price <- parse_number(logisticdf$price)
logisticdf$high_price <- ifelse(logisticdf$price >= 1000000, 1, 0)

# Create a new dataframe with 'high_price' column
logisticdf <- logisticdf %>%
  select(-price)

# Split data for logistic regrssion model
set.seed(123)  #
split_ratio <- 0.7  # 70% training, 30% testing
index <- sample(1:nrow(logisticdf), size = round(split_ratio * nrow(logisticdf)))
train_data_logistic <- logisticdf[index, ]
test_data_logistic <- logisticdf[-index, ]

#Build basic logistic regrssion model
logistic_model <- glm(high_price ~ ., data = train_data_logistic, family = binomial)
summary(logistic_model)

# Model 2: Regression Tree
tree_model <- rpart(price ~ ., data = train_data)
printcp(tree_model)  # Display complexity parameter plot
plot(tree_model)  # Visualize the decision tree

# Model 3: Neural Network
nn_model <- neuralnet(price ~ ., data = train_data, hidden = c(5, 2), linear.output = TRUE)

# Tune neural network hyperparameters - placehlder

# Step c: Model Evaluation
```

```r
# Model 1: Logistic Regression
logistic_predictions <- predict(logistic_model, newdata = test_data, type = 'response')
logistic_accuracy <- sum((logistic_predictions > 0.5) == test_data$high_price) / length(test_data$high_price)
cat("Logistic Regression Accuracy:", logistic_accuracy, "\n")

# Model 2: Regression Tree
tree_predictions <- predict(tree_model, newdata = test_data)
tree_rmse <- sqrt(mean((tree_predictions - test_data$price)^2))
cat("Regression Tree RMSE:", tree_rmse, "\n")

# Model 3: Neural Network
nn_predictions <- predict(nn_model, newdata = test_data)
# Calculate relevant performance metrics for the neural network model

# Step d: Model Comparison (Choose the best model based on evaluation metrics)

# Calculate evaluation metrics for all models
logistic_predictions <- predict(logistic_model, newdata = test_data, type = 'response')
logistic_accuracy <- sum((logistic_predictions > 0.5) == test_data$high_price) / length(test_data$high_price)
cat("Logistic Regression Accuracy:", logistic_accuracy, "\n")

tree_predictions <- predict(tree_model, newdata = test_data)
tree_rmse <- sqrt(mean((tree_predictions - test_data$price)^2))
cat("Regression Tree RMSE:", tree_rmse, "\n")

nn_predictions <- predict(nn_model, newdata = test_data)
cat("NN RMSE:", tree_rmse, "\n")

# Step e: Backtesting - placeholder

# Function to compare model performance
compare_model_performance <- function(models, model_names, data_train, data_test) {
  rmse_values <- c()
  rsquared_values <- c()
  mape_values <- c()

  for (i in 1:length(models)) {
    model <- models[[i]]
    predictions <- predict(model, newdata = data_test)
    rmse <- sqrt(mean((data_test$price - predictions)^2))
    rsquared <- cor(data_test$price, predictions)^2
    mape <- mean(abs((data_test$price - predictions) / data_test$price)) * 100

    rmse_values <- c(rmse_values, rmse)
    rsquared_values <- c(rsquared_values, rsquared)
    mape_values <- c(mape_values, mape)
  }

  # Create a data frame to hold the results
  model_results <- data.frame(Model = model_names, RMSE = rmse_values, R_squared = rsquared_values, MAPE = map

  # Create a table to compare model results
  kable(model_results, format = "markdown")
}
```

VI. Model Limitation and Assumptions (15 points)

Based on the performances on both train and test data sets, determine your primary (champion) model and the other model which would be your benchmark model. Validate your models using the test sample. Do the residuals look normal? Does it matter given your technique? How is the prediction performance using Pseudo R^2, SSE, RMSE? Benchmark the model against alternatives. How good is the relative fit? Are there any serious violations of the model assumptions? Has the model had issues or limitations that the user must know? (Which assumptions are needed to support the Champion model?)

VII. Ongoing Model Monitoring Plan (5 points)

How would you picture the model needing to be monitored, which quantitative thresholds and triggers would you set to decide when the model needs to be replaced? What are the assumptions that the model must comply with for its continuous use?

To Be Ripped Out

As a team we need to decide how to create a train/test split because their might be data leakage. Cutting the threshold by year, 2014 gets put into training and 2015 into test gets the sample sizes pretty close.

```
# Find Cutoff point
yyyymm <- kc.house.df$year_sold*100 + kc.house.df$month_sold

sum(yyyymm < 201501) / nrow(kc.house.df)
```

```
## [1] 0.6770462
```

```
sum(yyyymm < 201502) / nrow(kc.house.df)
```

```
## [1] 0.7222968
```

```
# It's a toss up which cutoff since none are exactly 70%
# We'll use 201501 as the cutoff since it's the easier
concat_date <- kc.house.df$year_sold*100 + kc.house.df$month_sold
kc.house.df$yyyymm <- concat_date
```

Overview

For model monitoring there are four areas to keep track of model stability, prediction performance, incremental data quality checks and pipeline fail safes.

1. Model Stability

This section concerns itself with stability of predictions. Price predictions often are used by loan originators or for some consumer product. If customers get wild loan offerings or sale price recommendations month to month, it'll be a bad user experience, creating distrust between the customer and the firm.

To check for model stability the approach is to store last month's prediction values in a database table and the model itself in an S3 bucket as an RDS file. We are also assuming that we get data in monthly batches, as that is the frequency of our data set.

```
adjust_months <- function(date, num_months) {
  # Convert to character for easier manipulation
  date_str <- as.character(date)

  # Extract year and month components
  year <- substr(date_str, 1, 4)
  month <- substr(date_str, 5, 6)

  # Convert to numeric and adjust months
  result_month <- as.numeric(month) + num_months
  result_year <- as.numeric(year) + floor((result_month - 1) / 12)
  result_month <- (result_month - 1) %% 12 + 1  # Ensure the month is within 1 to 12

  # Create the result as "yyyymm"
  result <- paste0(result_year, sprintf("%02d", result_month))

  return(result)
}
```

Since we do not have infrastructure stood up we use this code block to simulate pulling in previous month's model and model predictions.

```
set.seed(1023)
```

```
lookback.window <- 12 #one year lookback
cutoff_date <- 201501
train_start <- adjust_months(cutoff_date, -lookback.window)

prev_train_data <- kc.house.df[
  which(
    kc.house.df$yyyymm < cutoff_date &
    kc.house.df$yyyymm >= train_start
  ),
]

horizon <- 1
test_start <- cutoff_date
test_end <- adjust_months(test_start, horizon)

prev_test_data <- kc.house.df[
  which(
    (kc.house.df$yyyymm < test_end) &
    (kc.house.df$yyyymm >= test_start)

  ),
]
previous.model <- lm(price ~ . -year_sold -month_sold, data = prev_train_data)
previous.pred <- predict(previous.model, prev_test_data)
```

Now we simulate getting a new batch of data and fitting the model on new data. We compare the new model predictions with the old model using

1. T-test on the means of the predictions
2. F-test for the variance of the predictions
3. T-test on the squared error.

The last test is a Anderson-Darling test to check normality of predictions. Shapiro-Wilk test will not work for these large sample

In practice, model stability tests tend to be overly sensitive so we will trigger Slack warnings when the p-values of the test are less than 0.01.

```
library(nortest)

#Current Model
cutoff_date <- adjust_months(cutoff_date, 1)
train_start <- adjust_months(cutoff_date, -lookback.window)

curr_train_data <- kc.house.df[
  which(
    kc.house.df$yyyymm < cutoff_date &
    kc.house.df$yyyymm >= train_start
  ),
]

horizon <- 1
test_start <- cutoff_date
test_end <- adjust_months(test_start, horizon)

curr_test_data <- kc.house.df[
  which(
    (kc.house.df$yyyymm < test_end) &
    (kc.house.df$yyyymm >= test_start)

  ),
```

```
]
current.model <- lm(price ~ . -year_sold -month_sold, data = curr_train_data)
curr.pred <- predict(current.model, curr_test_data)

# H0: same means
# H1: means are not the same
t.test(previous.pred, curr.pred)
```

```
##
##  Welch Two Sample t-test
##
## data:  previous.pred and curr.pred
## t = -0.77098, df = 1986.6, p-value = 0.4408
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -34327.39  14953.67
## sample estimates:
## mean of x mean of y
##  492171.2  501858.1
```

```
# H0: same variance
# H1: variance are not the same
var.test(previous.pred, curr.pred)
```

```
##
##  F test to compare two variances
##
## data:  previous.pred and curr.pred
## F = 1.2298, num df = 977, denom df = 1249, p-value = 0.0005855
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  1.092940 1.385091
## sample estimates:
## ratio of variances
##            1.22984
```

```
#Check if mean squared error is equivalent
prev_sq_error <- (prev_test_data[, "price"] - previous.pred)^2
curr_sq_error <- (curr_test_data[, "price"] - curr.pred)^2
t.test(prev_sq_error, curr_sq_error)
```

```
##
##  Welch Two Sample t-test
##
## data:  prev_sq_error and curr_sq_error
## t = -0.47251, df = 2198.2, p-value = 0.6366
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -15414156014   9428345163
## sample estimates:
##   mean of x   mean of y
## 39809793540 42802698965
```

```
#Check predictions are normal
ad.test(current.model$residuals)
```

```
##
##  Anderson-Darling normality test
##
## data:  current.model$residuals
## A = 338.39, p-value < 2.2e-16
```

Looks like in our case, the tests will trigger and warn us that the mean predictions have deviated, variances have not, MSE

has not and the residuals are triggered as not normal. Means of predictions deviating doesn't warrant much of a concern but still worth notifying because that way consumers and downstream stakeholders can be informed about the current month's prediction.

The most important of these is MSE as we do not want our model to fluctuate in terms of error. Normality is important in the world of inference, especially if model coefficients become part of a product. However right now we are only concerned with predictive power.

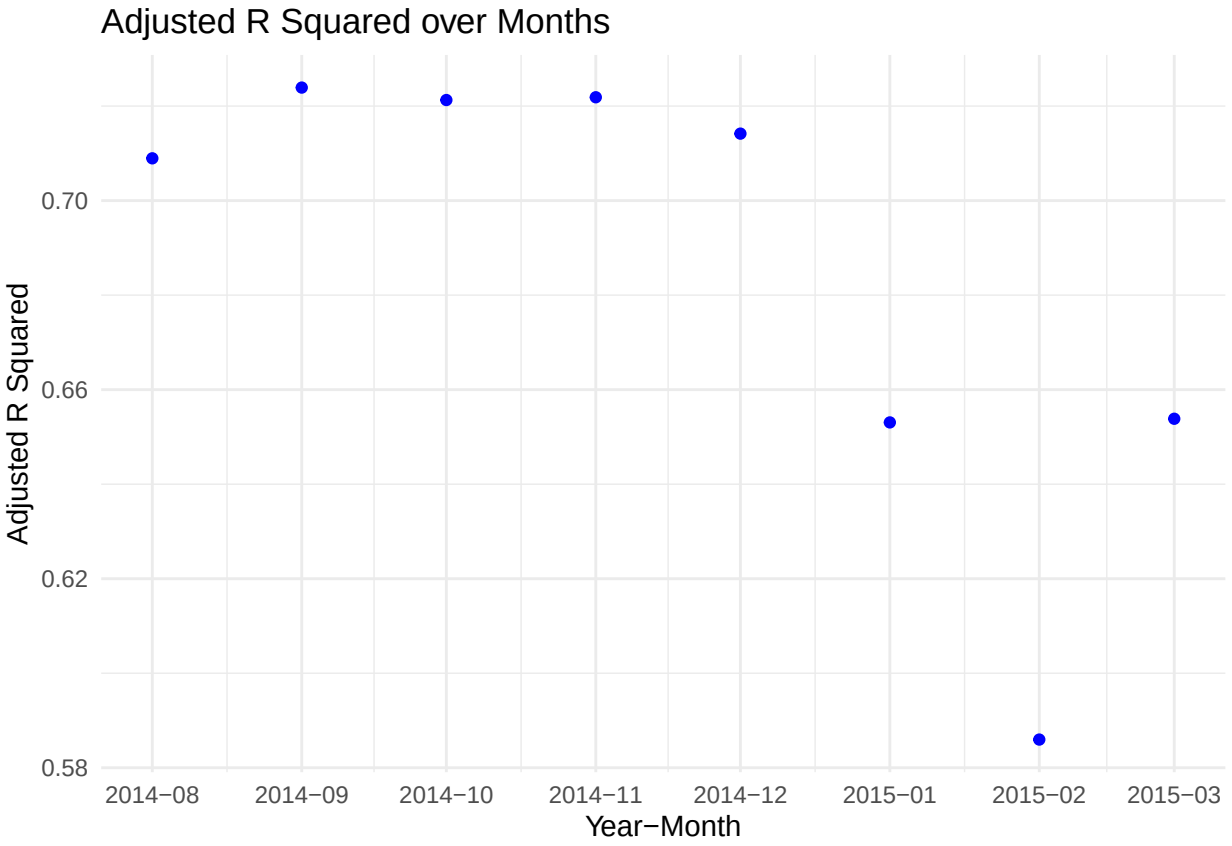
2. Prediction Performance over Time

For this section we track key model metrics over time to ensure there aren't any anomalies that occur with the model. Metrics we choose are adjusted R^2 and RMSE. We apply a function to create a rolling window for the model to train on and then test with data outside the window. We collect the metrics and plot them over time.

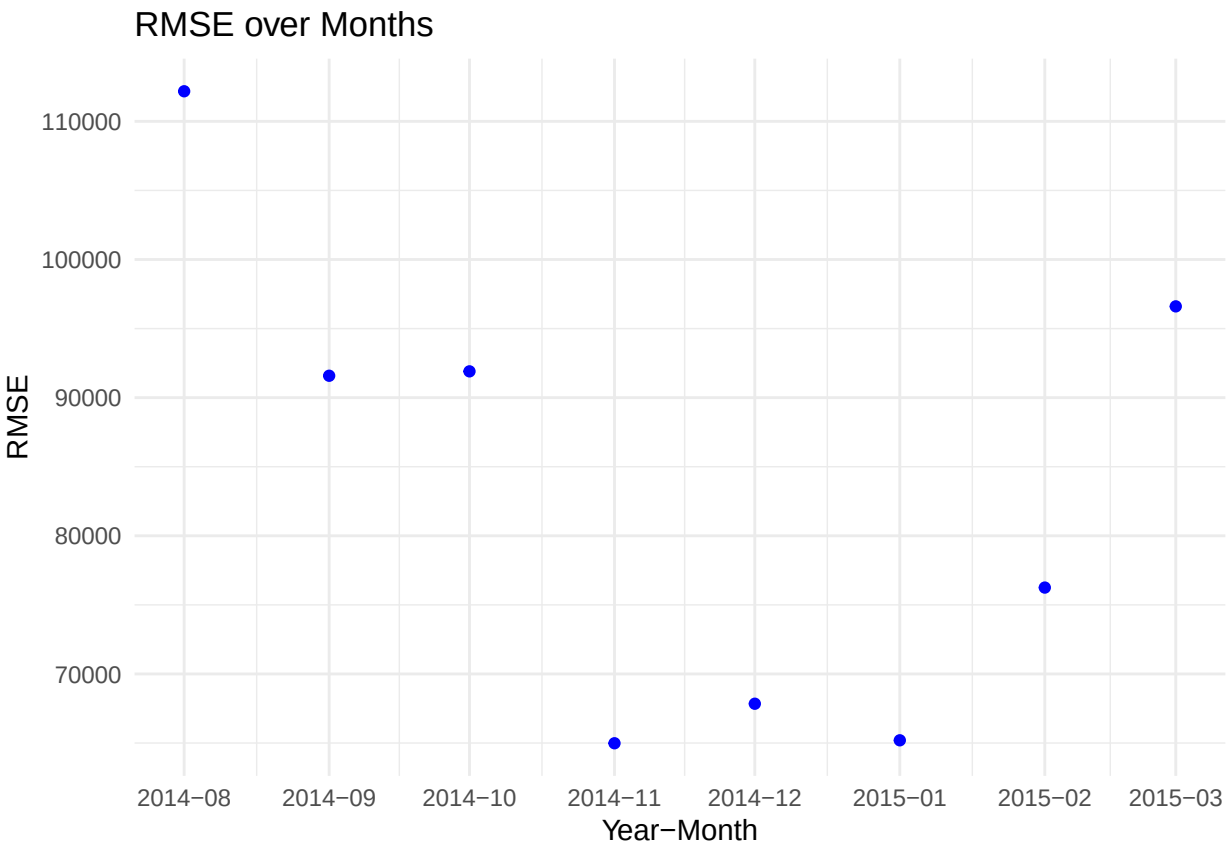
If any metrics fall below the baseline production model, we will trigger a slack message.

```
#Iterate over folds
roll_model <- function(cutoff_dates){

  metrics <- list()

  for (cutoff_date in cutoff_dates){
    lookback.window <- 6 #one year lookback
    train_start <- adjust_months(cutoff_date, -lookback.window)

    train_data <- kc.house.df[
      which(
        kc.house.df$yyyymm < cutoff_date &
        kc.house.df$yyyymm >= train_start
      ),
    ]

    horizon <- 1
    test_start <- cutoff_date
    test_end <- adjust_months(test_start, horizon)

    test_data <- kc.house.df[
      which(
        (kc.house.df$yyyymm < test_end) &
        (kc.house.df$yyyymm >= test_start)

      ),
    ]
    model <- lm(price ~ . -year_sold -month_sold, data = train_data)
    pred <- predict(model, test_data)

    act <- test_data[, "price"]
    n <- dim(model.matrix(model))[1]
    p <- dim(model.matrix(model))[2]

    metric <- CalcTestMetrics(pred, act, n, p)
    metrics <- c(metrics, metric)
  }

  return (metrics)
}

years <- c(201408, 201409, 201410, 201411, 201412, 201501, 201502, 201503)
metrics_table <- do.call(rbind, lapply(years, roll_model))
metrics_table <- as.data.frame(metrics_table)
metrics_table$years <- years
metrics_table$rmse <- sqrt(as.numeric(metrics_table$mse))
```

```r
# Convert date to Date class
metrics_table$date <- as.Date(paste0(metrics_table$years, "01"), format = "%Y%m%d")

# Create the ggplot
ggplot(metrics_table, aes(x = date, y = as.numeric(adj.rsquared))) +
  geom_point(color = 'blue') +
  labs(x = "Year-Month", y = "Adjusted R Squared", title = "Adjusted R Squared over Months") +
  scale_x_date(date_labels = "%Y-%m", date_breaks = "1 month") +
  theme_minimal()
```




```r
ggplot(metrics_table, aes(x = date, y = as.numeric(rmse))) +
  geom_point(color = 'blue') +
  labs(x = "Year-Month", y = "RMSE", title = "RMSE over Months") +
  scale_x_date(date_labels = "%Y-%m", date_breaks = "1 month") +
  theme_minimal()
```



3. Incremental Data Quality Check

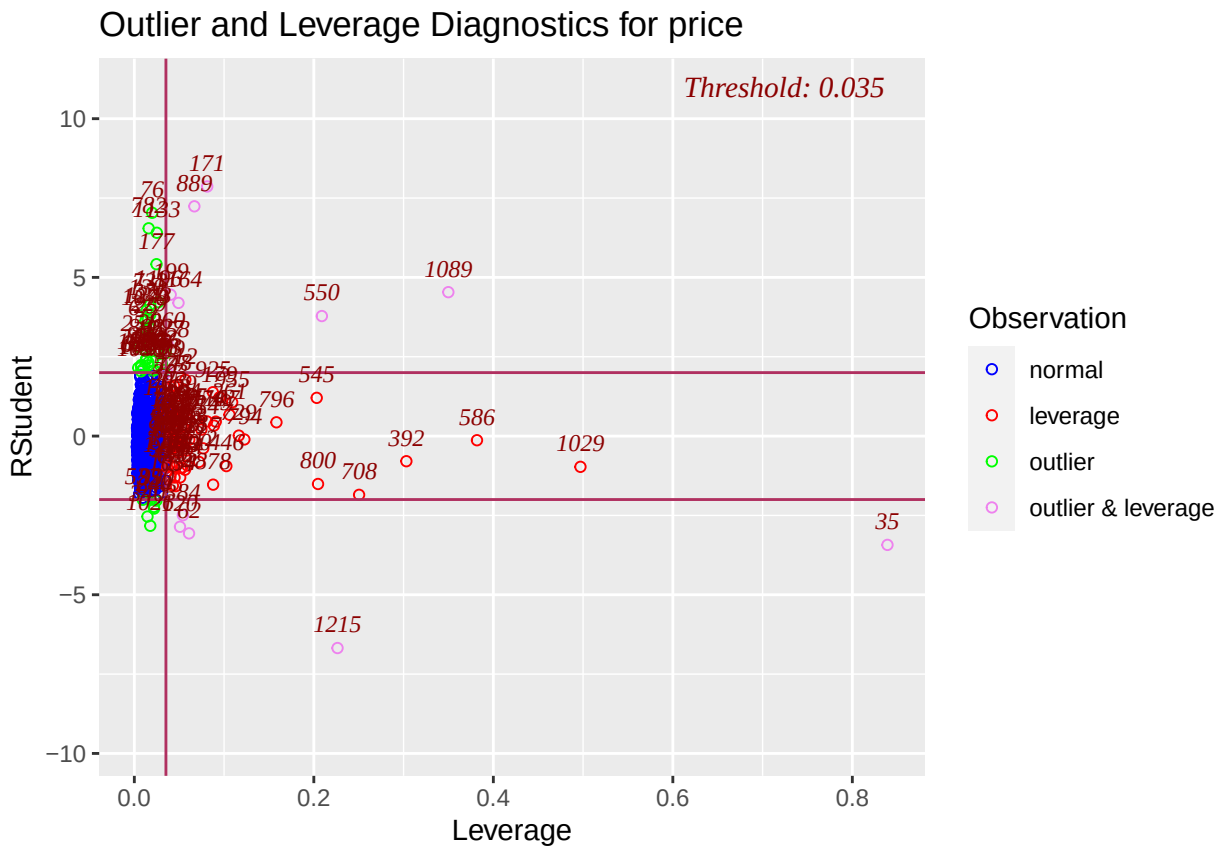
When new data comes in at monthly interval we will perform checks on it to ensure that it will not cause the model problems. The first we do is create graphs to add to the dashboard for visual inspection. These are the Residuals vs Leverage plot to detect outliers, leverage points or both and the Cook's Distance chart to detect influential points. The hope is to catch problematic points here before they reach the model.

```
library(olsrr)
```

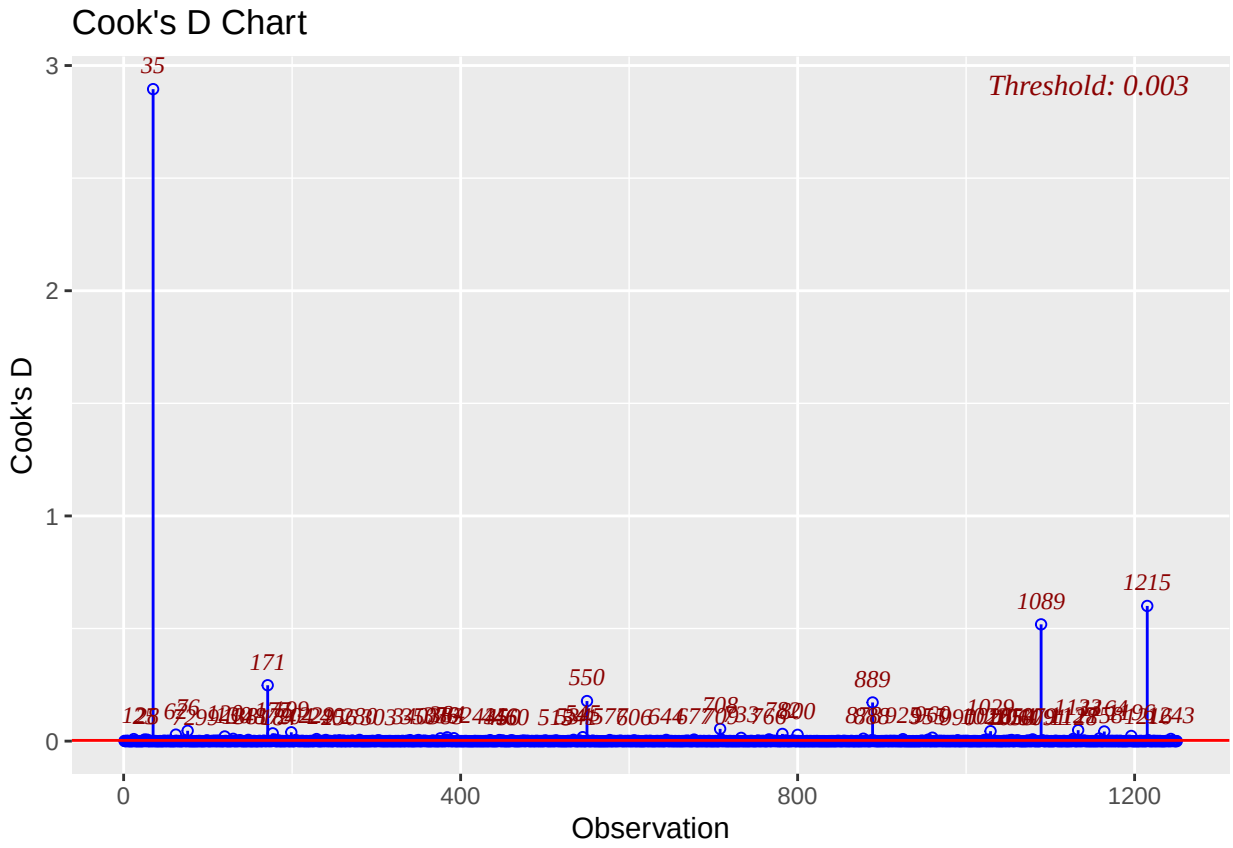
```
##
## Attaching package: 'olsrr'
```

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

```
incremental_data <- curr_test_data
incremental.model <- lm(price ~ . -year_sold -month_sold, data = incremental_data)

#Residual vs Leverage
ols_plot_resid_lev(incremental.model)
```




```
#Cook's Distance Visualized
ols_plot_cooksd_chart(incremental.model)
```



To compare the old data with the new incremental data batch we perform a Kolmogorov–Smirnov test for continuous features and Chi-Square Goodness of Fit test for categorical data. The threshold we set will be a conservative 0.01 as these tests tend to be overly sensitive in practice because real data is extremely variable.

Lastly, a performance optimization that we can do is to pre-compute the values for the old data because we do not want to pull in all historical data and run calculations over and over again.

```
old_data <- curr_train_data

continuous_features <- c("price", "sqft_living", "sqft_lot", "sqft_above", "year_built", "yr_renovated", "sqft
discrete_features <- c("bedrooms", "bathrooms", "floors", "waterfront", "view", "condition", "grade", "zipcode

ks_pvalues <- c()
for (feature in continuous_features){
  pval <- ks.test(old_data[, feature], incremental_data[, feature], simulate.p.value = TRUE)$p.value
  ks_pvalues <- c(ks_pvalues, pval)
}

chisq_pvalues <- c()
for (feature in discrete_features){
  # Extract the column data
  old_feature_data <- old_data[[feature]]
  expected_probabilities <- table(old_feature_data) / length(old_feature_data)

  incremental_feature_data <- incremental_data[[feature]]
  observed_probabilities <- table(old_feature_data) / length(incremental_feature_data)

  #want to compare new to old
  pval <- chisq.test(observed_probabilities, expected_probabilities, simulate.p.value = TRUE)$p.value
  chisq_pvalues <- c(chisq_pvalues, pval)
}

cbind(continuous_features, ks_pvalues)
```

```
##      continuous_features ks_pvalues
## [1,] "price"             "0.00499750124937526"
## [2,] "sqft_living"       "0.00599700149925032"
## [3,] "sqft_lot"          "0.19640179910045"
## [4,] "sqft_above"        "0.00299850074962513"
## [5,] "year_built"        "0.529735132433783"
## [6,] "yr_renovated"      "0.000999500249875007"
## [7,] "sqft_living15"     "0.217891054472764"
## [8,] "sqft_lot15"        "0.432783608195902"
```

```
cbind(discrete_features, chisq_pvalues)
```

```
##      discrete_features chisq_pvalues
## [1,] "bedrooms"        "0.0104947526236882"
## [2,] "bathrooms"       "0.000499750124937531"
## [3,] "floors"          "1"
## [4,] "waterfront"      "1"
## [5,] "view"            "1"
## [6,] "condition"       "1"
## [7,] "grade"           "0.0154922538730635"
## [8,] "zipcode"         "0.000499750124937531"
```

Looks like `price`, `bedrooms`, `bathrooms` and `zipcode` deviated in our incremental data batch.

4. Pipeline Fail Safes

Note: No examples are shown here because we do not have data infrastructure

The plan to handle pipeline failures is to persist every production model and every test and training set of data. Models will be stored in an S3 bucket. The data sets will be stored as compressed parquet files because database tables usually only contain the most updated version (upserted records) and in this case we want to use old data. When the pipeline fails, we can use old data and the old model to continue to generate predictions. This ensures downstream stakeholders are free from breaking changes.

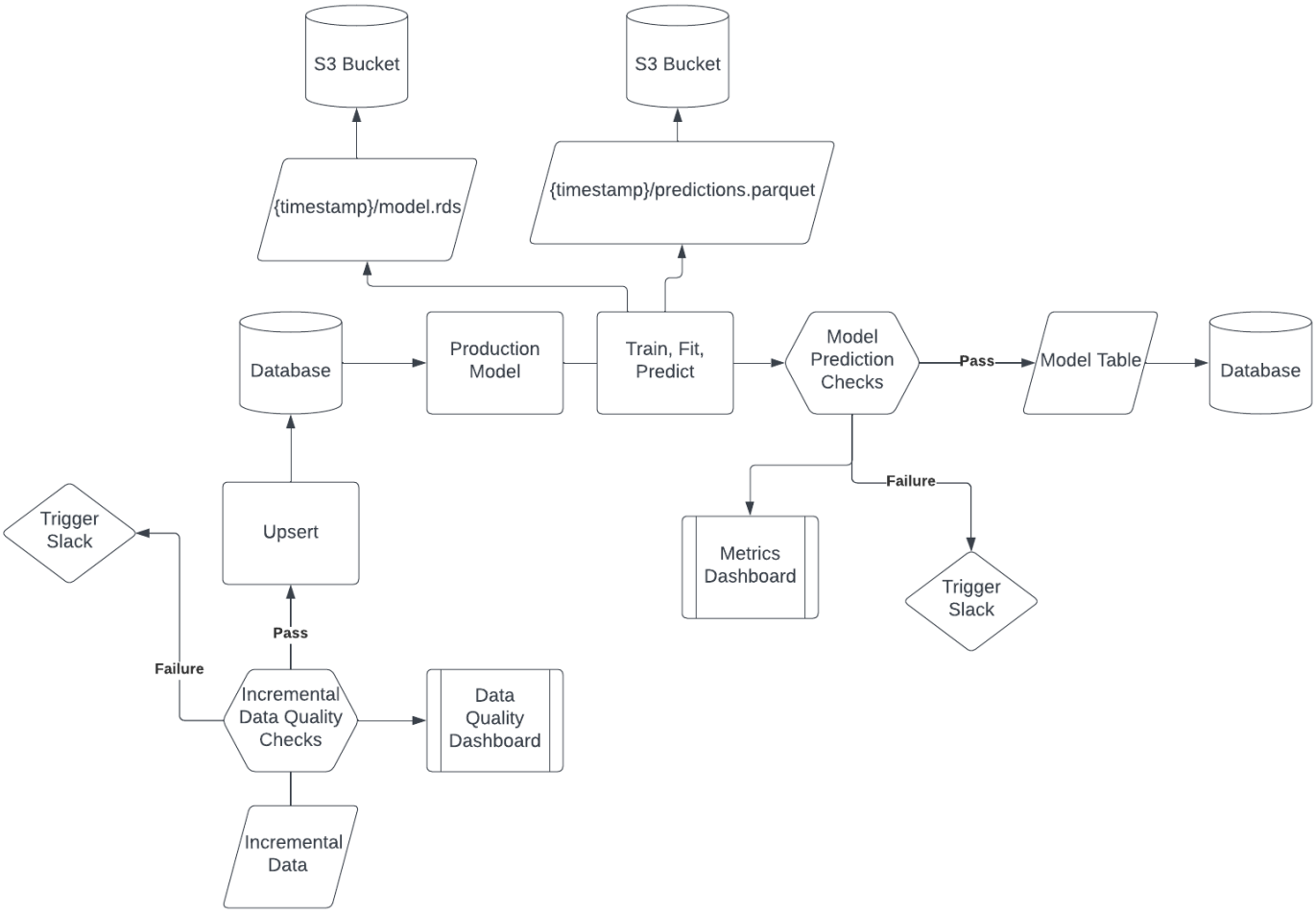


Figure 1: Model Monitoring Architecture

VIII. Conclusion (5 points)

Summarize your results here. What is the best model for the data and why?

Bibliography (7 points)

Please include all references, articles and papers in this section.

1. McGill University

- http://www.med.mcgill.ca/epidemiology/joseph/courses/EPIB-621/centered_var.pdf

Appendix (3 points)

Please add any additional supporting graphs, plots and data analysis.