# STAT 6302 - Data Assignment 1

**Heesu Choi, Chengyu Yang & Advika Raj**

**February 2025**

# Contents

# 1 Introduction

The objective of this project is to develop a predictive model for estimating sale prices of houses using the Ames Housing dataset by utilizing a **tidymodels** pipeline in R. The dataset from previous data assignments completed last semester will undergo comprehensive feature engineering for better regression modeling. Following this, we will fit the data to 3 penalized regression models, extract optimal tuning parameter values, evaluate their performance, and select the best-performing model. Finally, as in the previous semester, we will submit our predictions to **Kaggle** for the Kaggle scores, providing a benchmark for our model's accuracy and predictive power.

# 2 Feature Engineering

## 2.1 Creating New Variables

In the first stage of feature engineering, we created three new variables that were not present in the original dataset. We referenced **"House Prices: Lasso, XGBoost, and a Detailed EDA"**[1], authored by **Erik Bruin** on Kaggle for guidance. In his analysis, he proposed five different feature engineering approaches, from which we selected and implemented three in our project.

Firstly, based on his findings, we consolidated the individual bathroom variables into a single predictor. Since half-bathrooms do not have the full functionality of a full bathroom, we counted each half-bathroom as 0.5 in the total calculation.

$$TotBathrooms = FullBath + (HalfBath * 0.5) + BsmtFullBath + (BsmtHalfBath * 0.5)$$

Secondly, we combined the above-ground and below-ground living spaces into a single total living space variable. This adjustment reflects how homebuyers typically consider the total living area as a key factor in property valuation.

$$TotalSqFeet = GrLivArea + TotalBsmtSF$$

Lastly, we consolidated various porch area types (open, enclosed, three-season, and screened porches) into a single predictor while retaining specific porches essential for safety. Although this new variable did not exhibit a strong correlation with SalePrice, as noted in Bruin's analysis, combining these features improves interpretability and provides a more meaningful representation of outdoor living space. After creating the new predictors, we removed the original variables from the dataset to streamline the model and reduce redundancy.

$$TotalPorchSF = OpenPorchSF + EnclosedPorch + X3SsnPorch + ScreenPorch$$

## 2.2 Data Preprocessing

Before identifying near-zero variance variables, we first clean the dataset by combining the training and testing datasets to ensure consistency in columns. During the process of vertically binding the two datasets, we removed column 71 that contains our target variable "SalePrice", ensuring that only predictor features were included in the combined dataset.

To identify and handle near-zero variance variables, we utilized the step functions from the tidymodels framework. In our recipe, we applied six sequential preprocessing steps to ensure data quality and model reliability. First, we filtered out features with more than 20% missing values, as high-missing variables contribute little meaningful information to the model. Next, we imputed missing values in numerical predictors using the median to enhance model performance and prevent data loss. To handle categorical data, we applied step_unknown() to assign missing categorical values to an "unknown" category, ensuring consistency in feature representation. Since we used a linear model, we created dummy variables to properly encode categorical predictors. Additionally, we removed near-zero variance predictors using step_nzv(), eliminating features with minimal variability to reduce complexity. Lastly, we addressed multicollinearity by applying step_corr(), removing highly correlated features to improve model interpretability. This structured preprocessing approach enhances data quality, reduces computational inefficiencies, and optimizes model performance.

| List of near-zero variance variables (NUMERIC ONLY) |
|---|
| Street, LandContour, Utilities, LandSlope, Condition2, RoofMatl, BsmtCond, BsmtFinType2, BsmtFinSF2, Heating, LowQualFinSF, KitchenAbvGr, Functional, GarageQual, GarageCond, PoolArea, MiscVal |

Figure 1: Near-zero variance variables (dummy variables not included)

After that, we split the dataset back into training and testing datasets by ID, then we added "Saleprice" back to the training dataset for the next step.

## 2.3 Natural Cubic Splines

To address potential high variance at the extremes when x takes on very large or very small values, we implemented natural cubic splines, which incorporate boundary constraints to mitigate overfitting. This approach ensures a smooth fit while preventing excessive fluctuations at the edges of the data range. Before applying natural cubic splines, we conducted an exploratory analysis by generating scatterplots to identify non-linear relationships between predictors and the target variable (SalePrice).

[1]Erik Bruin, "House prices: Lasso, XGBoost, and a detailed EDA", Kaggle, Accessed February 16, 2025. https://www.kaggle.com/code/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda/report.

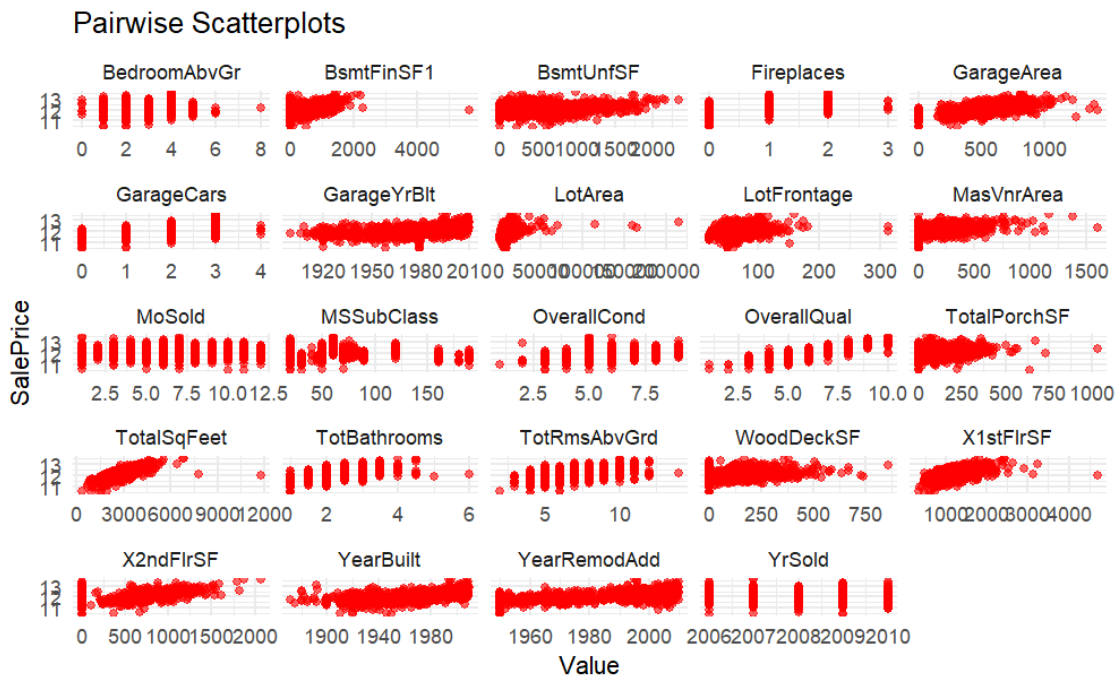


Figure 2: Pairwise Scatterplots

The pairwise scatterplots revealed a non-linear relationship between GarageYrBlt and SalePrice, prompting us to employ a natural cubic spline transformation. To capture this non-linearity effectively, we applied a natural cubic spline with five degrees of freedom within a preprocessing recipe, ensuring that GarageYrBlt contributes meaningfully to the predictive model while maintaining smoothness and interpretability.

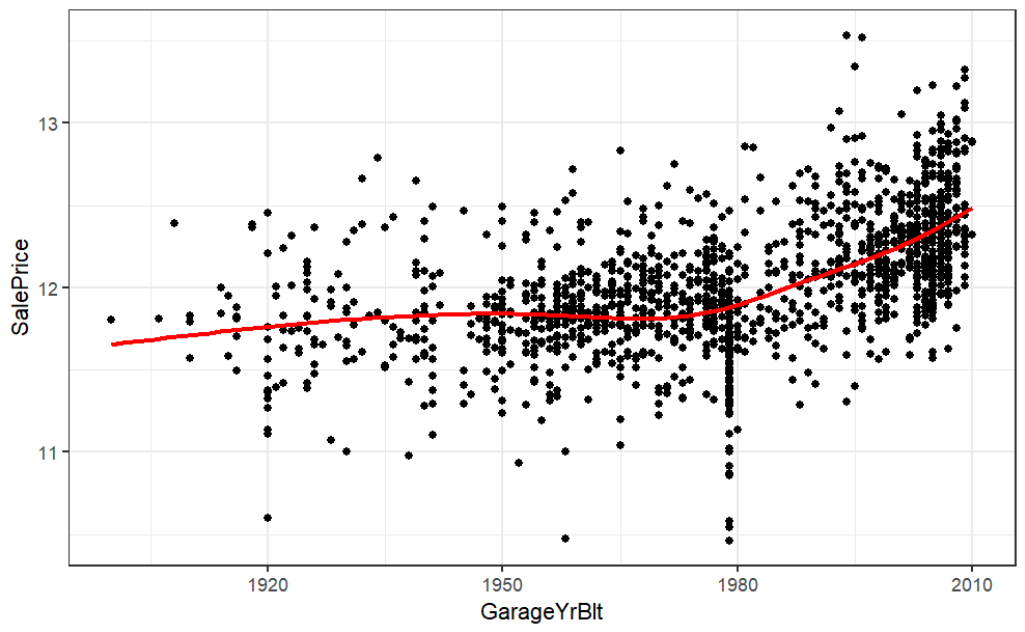


Figure 3: Natural cubic spline with DF = 5

## 3 Ridge/Lasso/Elastic Net Results

We used $ln(Saleprice)$ as our dependent variable. For CV methods, we decided to utilize a 10-fold cross-validation with 5 repeats in all 3 penalized regressions. The 103 variables we started with are shown in the figure below:

| Initial Variables |
| --- |
| MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, YearBuilt, YearRemodAdd, MasVnrArea, BsmtFinSF1, BsmtUnfSF, X1stFlrSF, X2ndFlrSF, BedroomAbvGr, TotRmsAbvGrd, Fireplaces, GarageCars, GarageArea, WoodDeckSF, MoSold, YrSold, TotBathrooms, TotalSqFeet, TotalPorchSF, MSZoning_RL, MSZoning_RM, LotShape_Reg, LandContour_Lvl, LotConfig_CulDSac, LotConfig_Inside, Neighborhood_CollgCr, Neighborhood_Edwards, Neighborhood_Gilbert, Neighborhood_NAmes, Neighborhood_NridgHt, Neighborhood_OldTown, Neighborhood_Sawyer, Neighborhood_Somerst, Condition1_Feedr, Condition1_Norm, BldgType_TwnhsE, HouseStyle_X1Story, HouseStyle_X2Story, RoofStyle_Gable, RoofStyle_Hip, Exterior1st_HdBoard, Exterior1st_MetalSd, Exterior1st_Plywood, Exterior1st_VinylSd, Exterior1st_Wd.Sdng, Exterior2nd_HdBoard, Exterior2nd_MetalSd, Exterior2nd_Plywood, Exterior2nd_VinylSd, Exterior2nd_Wd.Sdng, MasVnrType_BrkFace, MasVnrType_None, MasVnrType_Stone, ExterQual_Gd, ExterQual_TA, ExterCond_Gd, ExterCond_TA, Foundation_CBlock, Foundation_PConc, BsmtQual_Gd, BsmtQual_TA, BsmtCond_TA, BsmtExposure_Gd, BsmtExposure_Mn, BsmtExposure_No, BsmtFinType1_BLQ, BsmtFinType1_GLQ, BsmtFinType1_LwQ, BsmtFinType1_Rec, BsmtFinType1_Unf, BsmtFinType2_Unf, HeatingQC_Gd, HeatingQC_TA, CentralAir_Y, Electrical_SBrkr, KitchenQual_Gd, KitchenQual_TA, Functional_Typ, GarageType_Attchd, GarageType_BuiltIn, GarageType_Detchd, GarageType_unknown, GarageFinish_RFn, GarageFinish_Unf, GarageFinish_unknown, GarageQual_TA, GarageQual_unknown, GarageCond_TA, GarageCond_unknown, PavedDrive_Y, SaleType_New, SaleType_WD, SaleCondition_Normal, SaleCondition_Partial, SalePrice, GarageYrBlt_ns_1, GarageYrBlt_ns_2, GarageYrBlt_ns_3, GarageYrBlt_ns_4, GarageYrBlt_ns_5 |

Figure 4: Initial Variables

## 3.1 Ridge Regression

The results of the ridge regression show that the best-performing model had a penalty value of 1.22783e-10, with an RMSE of 0.140838. The Ridge regression workflow was implemented using the tidymodels framework. The Ridge regression model demonstrated strong predictive performance while keeping all features in the model. This approach is particularly useful when dealing with multicollinearity, as it distributes variance among correlated predictors.

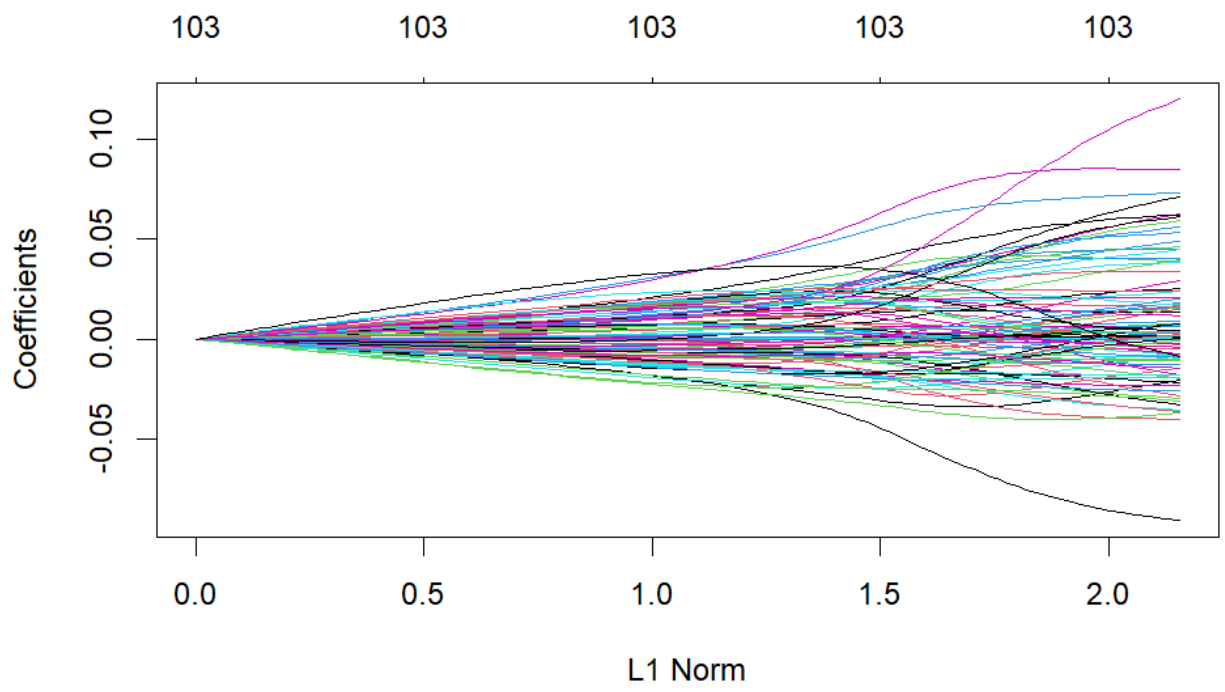


Figure 5: Ridge Solution Path

## 3.2 LASSO

The results of the LASSO show that the best-performing model had a penalty value of 0.002540586, with an RMSE of 0.1410021. Unlike Ridge regression, which retained all predictors while shrinking them, Lasso selected the key variables while the rest of the coefficients were set to zero. This ensures that only the most relevant predictors are included in the final model. In this case, 38 variables—including Mason Veneer Area and Land Contour Level—were removed from the model, while others, such as Overall Quality and Year Built, were kept with non-zero estimates.

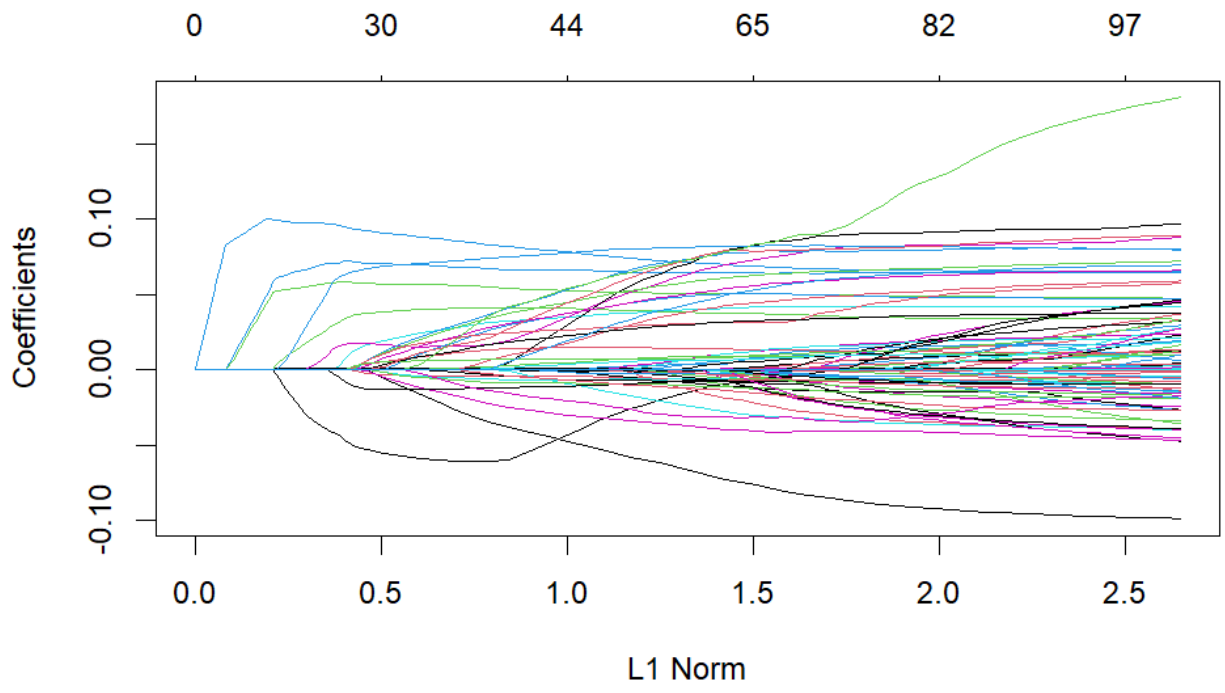


Figure 6: LASSO Solution Path

| Variable Retained |
| --- |
| GarageCars, GarageArea, WoodDeckSF, YrSold, TotBathrooms, TotalSqFeet, TotalPorchSF, MSZoning_RL, MSZoning_RM, LotShape_Reg, LotConfig_CulDSac, LotConfig_Inside, Neighborhood_Edwards, Neighborhood_Gilbert, Neighborhood_NAmes, Neighborhood_NridgHt, Neighborhood_OldTown, Neighborhood_Sawyer, Neighborhood_Somerst, Condition1_Norm, HouseStyle_X2Story, RoofStyle_Gable, Exterior1st_HdBoard, Exterior1st_MetalSd, Exterior1st_Wd.Sdng, MasVnrType_None, ExterQual_Gd, ExterQual_TA, ExterCond_TA, Foundation_PConc, BsmtQual_TA, BsmtCond_TA, BsmtExposure_Gd, BsmtExposure_No, BsmtFinType1_GLQ, BsmtFinType1_Unf, BsmtFinType2_Unf, HeatingQC_Gd, HeatingQC_TA, CentralAir_Y, KitchenQual_Gd, KitchenQual_TA, Functional_Typ, GarageType_Attchd, GarageFinish_RFn, GarageFinish_Unf, GarageCond_TA, PavedDrive_Y, SaleType_New, SaleType_WD, SaleCondition_Normal, GarageYrBlt_ns_1, GarageYrBlt_ns_4 |

Figure 7: Variables Retained by LASSO

### 3.3 Elastic Net

As for the Elastic Net, the selected optimal penalty was 0.001293884, with a mixture value of 0.2666. The corresponding RMSE was 0.1420733, which was slightly higher than the Ridge and Lasso models. Only 8 variables were removed in the Elastic Net's variable selection process. Its performance in terms of RMSE was slightly worse than Ridge regression, suggesting that Ridge's model and variable selection might have led to slightly better predictive power.

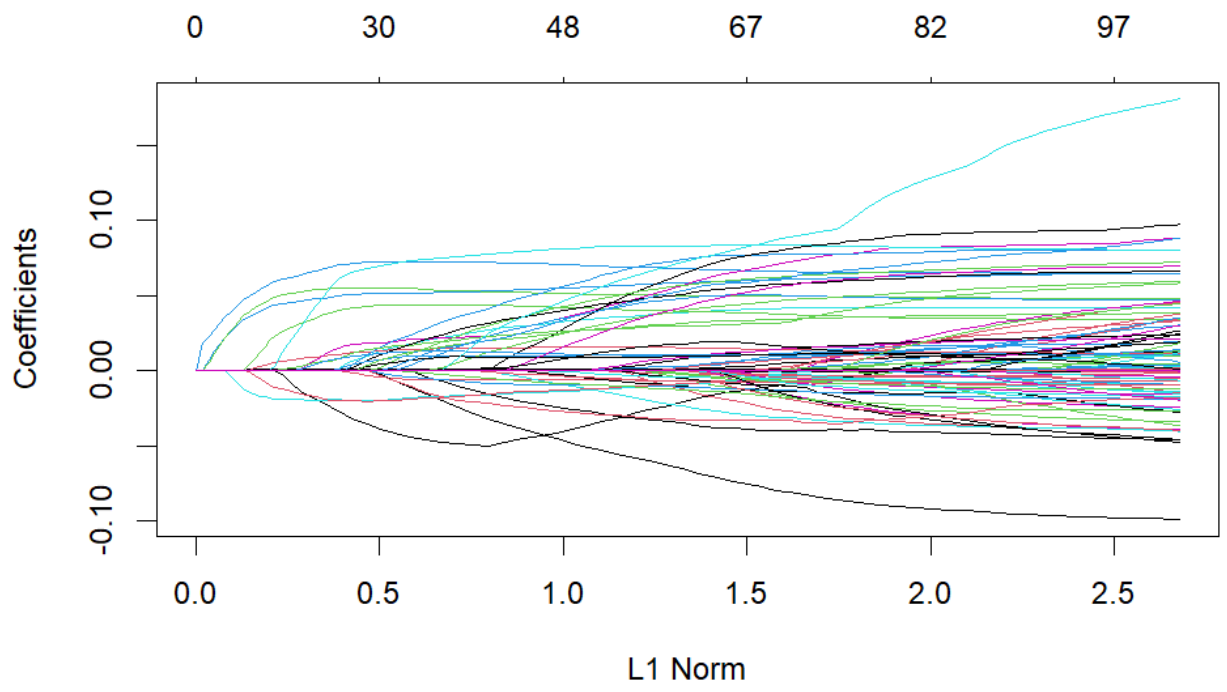


Figure 8: EN Solution Path

| Variables Retained |
| --- |
| MSSubClass, LotFrontage, LotArea, OverallQual, OverallCond, YearBuilt, YearRemodAdd, MasVnrArea, BsmtFinSF1, BsmtUnfSF, X1stFlrSF, X2ndFlrSF, BedroomAbvGr, TotRmsAbvGrd, Fireplaces, GarageCars, GarageArea, WoodDeckSF, YrSold, TotBathrooms, TotalSqFeet, TotalPorchSF, MSZoning_RL, LotShape_Reg, LandContour_Lvl, LotConfig_CulDSac, LotConfig_Inside, Neighborhood_CollgCr, Neighborhood_Edwards, Neighborhood_Gilbert, Neighborhood_NAmes, Neighborhood_NridgHt, Neighborhood_OldTown, Neighborhood_Sawyer, Neighborhood_Somerst, Condition1_Feedr, Condition1_Norm, BldgType_TwnhsE, HouseStyle_X1Story, HouseStyle_X2Story, RoofStyle_Gable, RoofStyle_Hip, Exterior1st_HdBoard, Exterior1st_MetalSd, Exterior1st_Plywood, Exterior1st_Wd.Sdng, Exterior2nd_HdBoard, Exterior2nd_MetalSd, Exterior2nd_Plywood, Exterior2nd_VinylSd, Exterior2nd_Wd.Sdng, MasVnrType_BrkFace, MasVnrType_None, MasVnrType_Stone, ExterQual_Gd, ExterCond_TA, Foundation_CBlock, Foundation_PConc, BsmtQual_Gd, BsmtQual_TA, BsmtCond_TA, BsmtExposure_Gd, BsmtExposure_No, BsmtFinType1_BLQ, BsmtFinType1_GLQ, BsmtFinType1_LwQ, BsmtFinType1_Rec, BsmtFinType1_Unf, BsmtFinType2_Unf, HeatingQC_Gd, HeatingQC_TA, CentralAir_Y, Electrical_SBrkr, KitchenQual_Gd, KitchenQual_TA, Functional_Typ, GarageType_Attchd, GarageType_Detchd, GarageType_unknown, GarageFinish_RFn, GarageFinish_Unf, GarageFinish_unknown, GarageQual_TA, GarageCond_TA, GarageCond_unknown, PavedDrive_Y, SaleType_New, SaleType_WD, SaleCondition_Normal, SaleCondition_Partial, GarageYrBlt_ns_1, GarageYrBlt_ns_2, GarageYrBlt_ns_3, GarageYrBlt_ns_4, GarageYrBlt_ns_5 |

Figure 9: Variables Retained by EN

## 4 Improvements on Modeling

This analysis enhances the predictive modeling process by utilizing tidymodels, which helped with the cleaning process and overall workflow for the model. Furthermore, using tidymodels, dealing with missing values, imputation, and dummy encoding were all able to be performed in the recipe rather than in multiple different cleaning procedures. In addition, another way this analysis improved the

project was the use of cross-validation to optimize model selection. By tuning parameters through 10-fold cross-validation, this analysis is able to identify the best-performing model. Lastly, after running the models and submitting the Kaggle scores, this analysis proved to have improvement as this metric performance was even better than the original.

| Predictive Model | Kaggle Score |
|---|---|
| Ridge | 0.15206 |
| LASSO | 0.14827 |
| Elastric Net | 0.14890 |

Figure 10: Kaggle Scores: Last Project

| Predictive Model | Kaggle Score |
|---|---|
| Ridge | 0.14061 |
| LASSO | 0.13972 |
| Elastric Net | 0.14012 |

Figure 11: Kaggle Scores: Data Assignment 1

# 5 Conclusion

This analysis highlights the use of tidymodels' workflow feature with penalized regression models in predicting house prices. The Ridge, Lasso, and Elastic Net regression allowed for a detailed analysis of different regularization effects, ultimately leading to tuned and the best performing combinations to each model, all of which strengthen predictive analysis.

The tidymodels package streamlines the "data cleaning-modeling" process, significantly improving efficiency. While it may not provide the same level of customization as manual data cleaning (extremely time-consuming), it strikes an effective balance between efficiency and predictive power, enabling users to focus on building robust models without sacrificing too much flexibility.

# 6 Bibliography

1. Erik Bruin, "House prices: Lasso, XGBoost, and a detailed EDA", Kaggle, Accessed February 16, 2025.

https://www.kaggle.com/code/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda/report.

# 7 Appendix

```
library(caret)
library(tidymodels)
library(tidyverse)
library(tidyr)
library(glmnet)
```

```
saleprice <- read.csv('C:/SASDATA/train.csv')
sp_test <- read.csv('C:/SASDATA/test.csv')
```

## 7.1 Q1+Q2 Feature Engineering & Data Cleaning

```
## feature engineering
## Combine all bathrooms
saleprice$TotBathrooms <- saleprice$FullBath + (saleprice$HalfBath*0.5) +
                          saleprice$BsmtFullBath + (saleprice$BsmtHalfBath*0.5)

sp_test$TotBathrooms <- sp_test$FullBath + (sp_test$HalfBath*0.5) +
                        sp_test$BsmtFullBath + (sp_test$BsmtHalfBath*0.5)

## Combine living space areas
saleprice$TotalSqFeet <- saleprice$GrLivArea + saleprice$TotalBsmtSF

sp_test$TotalSqFeet <- sp_test$GrLivArea + sp_test$TotalBsmtSF

## Consolidating Porch variables
saleprice$TotalPorchSF <- saleprice$OpenPorchSF + saleprice$EnclosedPorch +
                          saleprice$X3SsnPorch + saleprice$ScreenPorch

sp_test$TotalPorchSF <- sp_test$OpenPorchSF + sp_test$EnclosedPorch +
                        sp_test$X3SsnPorch + sp_test$ScreenPorch
```

```
## Deleting old columns
saleprice$FullBath <- NULL
saleprice$HalfBath <- NULL
saleprice$BsmtFullBath <- NULL
saleprice$BsmtHalfBath <- NULL
saleprice$GrLivArea <- NULL
saleprice$TotalBsmtSF <- NULL
saleprice$OpenPorchSF <- NULL
saleprice$EnclosedPorch <- NULL
saleprice$X3SsnPorch <- NULL
saleprice$ScreenPorch <- NULL

sp_test$FullBath <- NULL
sp_test$HalfBath <- NULL
sp_test$BsmtFullBath <- NULL
sp_test$BsmtHalfBath <- NULL
sp_test$GrLivArea <- NULL
sp_test$TotalBsmtSF <- NULL
sp_test$OpenPorchSF <- NULL
sp_test$EnclosedPorch <- NULL
sp_test$X3SsnPorch <- NULL
sp_test$ScreenPorch <- NULL

## Log-transformation
saleprice$SalePrice <- log(saleprice$SalePrice)
```

```
## I wanted to cleaning the dataset first (avoiding warnings in the regression part), so I created a recipe
## Combining training & testing datasets to get consistent data cleaning results
combined <- rbind(saleprice[,-71],sp_test)

## Identify nzv
nearZeroVar(combined,names=T)

combined_rec <- recipe(~ ., data = combined) %>%
                step_filter_missing(all_predictors(),threshold = 0.2) %>%
                step_impute_median(all_numeric_predictors()) %>%
                step_unknown(all_nominal_predictors()) %>%
                step_dummy(all_nominal_predictors()) %>%
                step_nzv(all_predictors()) %>%
```

```
                step_corr(threshold = 0.8)

## Applying the pre-processing steps to the full dataset
combined2 <- juice(prep(combined_rec))

## Retrieve training & testing datasets & remove ids
train <- combined2[1:1460,][,-1]
train$SalePrice <- saleprice$SalePrice
test <- combined2[1461:2919,][,-1]
```

## 7.2 Q3 Natural Cubic Splines

```
# Checking variables
head(train)
```

```
# Remove dummy variables
train_tmp <- train[-c(25:99)]

# Reshape the data to long format
train_tmp2 <- pivot_longer(train_tmp, cols = -SalePrice, names_to = "variable", values_to = "value")

# Create faceted scatterplots to find non-linear relationships
ggplot(train_tmp2, aes(x = value, y = SalePrice)) +
  geom_point(color = "red", alpha = 0.6) +
  facet_wrap(~ variable, scales = "free_x") +
  labs(title = "Pairwise Scatterplots",
       x = "Value", y = "SalePrice") +
  theme_minimal()
```

```
## checking non-linear relationship
plot(train$GarageYrBlt,train$SalePrice)
```

```
## Natural Cubic Splines
ggplot(train, aes(GarageYrBlt, SalePrice)) +
  geom_point() +
  geom_smooth(
    method = lm, se = FALSE, color = "red",
    formula = y ~ ns(x, df = 5)) +
  theme_bw()
```

```
## Setting up the recipe for regression
train_rec <- recipe(SalePrice ~ ., data = train) %>%
                step_ns(GarageYrBlt, deg_free = 5)

## Finalizing training dataset
train2 <- juice(prep(train_rec))
```

## 7.3 Q4 Regression & Prediction

### 7.3.1 Ridge

```
set.seed(123)

## We are using a 10-fold CV with 5 repeats
sp_kfolds <- vfold_cv(train, v = 10, repeats = 5, strata = "SalePrice")

#Specify model (penalty=lambda, mixture=alpha)
ridge_glmnet <- linear_reg(
                  penalty = tune(),
                  mixture = 0) %>%
                  set_mode("regression") %>%
                  set_engine("glmnet")

## Specify modeling procedure
ridge_wf <- workflow() %>%
            add_recipe(train_rec) %>%
            add_model(ridge_glmnet)

## Tuning the model
ridge_tune <- tune_grid(
              ridge_wf,
              resamples = sp_kfolds,
```

```
              grid = 10,
              metrics = metric_set(rmse))

#Display best performing combinations
ridge_tune %>% show_best(metric = "rmse")

##Pull/save the optimal tuning parameter combination
ridge_best_tune <- ridge_tune %>% select_best(metric = "rmse")
ridge_best_tune

#Now update our workflow
ridge_final_workflow <- ridge_wf %>%
                         finalize_workflow(ridge_best_tune)
ridge_final_workflow

##Extract coefficients
ridge_coef <- ridge_final_workflow %>%
  fit(train) %>%
  extract_fit_parsnip() %>%
  tidy()

ridge_coef

## Plotting the regularization path of the fitted model
ridge_glmnet <- ridge_final_workflow %>%
  fit(train) %>%
  extract_fit_parsnip()

plot(ridge_glmnet$fit)
```

### 7.3.2 Ridge Predictions

```
# Fit the model using training data
ridge_fit <- fit(ridge_final_workflow, data = train)

# Predict on the test dataset
ridge_predictions <- predict(ridge_fit, new_data = test)

## Extracting predictions
## write.csv(exp(ridge_predictions$.pred),'C:/SASDATA/123.CSV')
```

### 7.3.3 LASSO

```
#Specify model (penalty=lambda, mixture=alpha)
LASSO_glmnet <- linear_reg(
                  penalty = tune(),
                  mixture = 1) %>%
                  set_mode("regression") %>%
                  set_engine("glmnet")

## Specify modeling procedure
LASSO_wf <- workflow() %>%
            add_recipe(train_rec) %>%
            add_model(LASSO_glmnet)

## Tuning the model
LASSO_tune <- tune_grid(
              LASSO_wf,
              resamples = sp_kfolds,
              grid = 10,
              metrics = metric_set(rmse))

#Display best performing combinations
LASSO_tune %>% show_best(metric = "rmse")

##Pull/save the optimal tuning parameter combination
LASSO_best_tune <- LASSO_tune %>% select_best(metric = "rmse")
LASSO_best_tune

#Now update our workflow
LASSO_final_workflow <- LASSO_wf %>%
```

```
                        finalize_workflow(LASSO_best_tune)
LASSO_final_workflow

##Extract coefficients
LASSO_coef <- LASSO_final_workflow %>%
  fit(train) %>%
  extract_fit_parsnip() %>%
  tidy()

LASSO_coef

## Plotting the regularization path of the fitted model
LASSO_glmnet <- LASSO_final_workflow %>%
  fit(train) %>%
  extract_fit_parsnip()

plot(LASSO_glmnet$fit)
```

### 7.3.4 LASSO Prediction

```
# Fit the model using training data
LASSO_fit <- fit(LASSO_final_workflow, data = train)

# Predict on the test dataset
LASSO_predictions <- predict(LASSO_fit, new_data = test)

## Extracting predictions
## write.csv(exp(LASSO_predictions$.pred),'C:/SASDATA/123.CSV')
```

### 7.3.5 Elastic Net

```
#Specify model (penalty=lambda, mixture=alpha)
EN_glmnet <- linear_reg(
                  penalty = tune(),
                  mixture = tune()) %>%
                  set_mode("regression") %>%
                  set_engine("glmnet")

## Specify modeling procedure
EN_wf <- workflow() %>%
            add_recipe(train_rec) %>%
            add_model(EN_glmnet)

## Tuning the model
EN_tune <- tune_grid(
              EN_wf,
              resamples = sp_kfolds,
              grid = 10,
              metrics = metric_set(rmse))

#Display best performing combinations
EN_tune %>% show_best(metric = "rmse")

##Pull/save the optimal tuning parameter combination
EN_best_tune <- EN_tune %>% select_best(metric = "rmse")
EN_best_tune

#Now update our workflow
EN_final_workflow <- EN_wf %>%
                      finalize_workflow(EN_best_tune)
EN_final_workflow

##Extract coefficients
EN_coef <- EN_final_workflow %>%
  fit(train) %>%
  extract_fit_parsnip() %>%
  tidy()

EN_coef

## Plotting the regularization path of the fitted model
```

```r
EN_glmnet <- EN_final_workflow %>%
  fit(train) %>%
  extract_fit_parsnip()

plot(EN_glmnet$fit)
```

### 7.3.6 EN Prediction

```r
# Fit the model using training data
EN_fit <- fit(EN_final_workflow, data = train)

# Predict on the test dataset
EN_predictions <- predict(EN_fit, new_data = test)

## Extracting predictions
## write.csv(exp(EN_predictions$.pred),'C:/SASDATA/123.CSV')
```