CSCI 550 - Project 2

Eliot Liucci, Eric Folsom, Nick Clausen, Christal O'Connell

2024-11-09

1 Executive Summary

This project aims to determine the most suitable information for buyers and sellers to collect in order to accurately predict real estate listings in the Cook County Chicago area. Using data from 2013 to 2019, we focused on key listing characteristics like location, building materials, the square footage of the land, etc. Our analysis showed that listing size, room count, estimated value, and quality of life features(e.g., garage, fireplaces) are the top physical attributes affecting sale price. To further refine our selection, we tested and compared various virtual structures (models) that grouped these features and adjusted certain criteria according to the patterns and relationships found in our data. One common limitation of all models used was the likelihood of too closely tailoring them to our data. This problem occurred if we included too many attributes. However, including less sometimes led to lower performance. So finding a balance was critical and was a major factor in achieving feasible and comprehensive results. Furthermore, there may be other influential factors not included in our analysis, such as local economic changes, neighborhood amenities such as restaurants and educational facilities, or market fluctuations. Excluding these elements could limit the accuracy of our model since they can also influence sale price. Also, the relationship between certain grouped attributes may not necessarily be captured by a grouped model and may in fact work better together versus separately. And lastly, since the data is focused solely on Cook County residences, applying it to other markets might led to inaccuracies as factors like land value and buyer preferences, which can vary significantly across different locations. To conclude, the overall results of this project are subjective and only focus on select data and model structures. The limitations mentioned highlight potential areas where further testing and refinement may be needed to ensure the model remains reliable and relevant as market conditions continue to shift and evolve. Based on our findings, we formally recommend prioritizing two essential estimates in assessing a property's sale price: land value, and building's value. These estimates are the main factors for determining a listing's market value but are not the only factors that may need to be taken into consideration.

2 Data Preprocessing and Exploration

2.1 Data Cleaning

The data received had a few issues that needed to be dealt with. First of all, the `Description` variable contained pieces of key information that were extracted (number of bathrooms, number of bedrooms, total number of rooms, and sell date). Additionally, Area, Sub-Area, Block, Parcel, and Multicode were parsed from the `PIN` variable.

```
# Extracting important information from descriptions
sell_date = as.numeric(0)
rooms = as.numeric(0)
bedrooms = as.numeric(0)
baths = as.numeric(0)

# Takes a minute to run, not too bad though
for(i in 1:nrow(data)){
    sell_date[i] = str_split(
        str_split(data$Description[i], "sold on ")[[1]][2],
         ", is a")[[1]][1]
    rooms[i] = str_extract_all(
        str_split(data$Description[i], "total of ")[[1]][2],
         "\\d")[[1]][1]
    bedrooms[i] = str_extract_all(
        str_split(data$Description[i], "total of ")[[1]][2],
         "\\d")[[1]][2]
    baths[i] = str_split(
        str_split(data$Description[i], "bedrooms, and ")[[1]][2],
         " of which are bathrooms")[[1]][1]
    Area[i] = substring(data$PIN[i], first = 1, last = 2)
    Sub_Area[i] = substring(data$PIN[i], first = 3, last = 4)
    Block[i] = substring(data$PIN[i], first = 5, last = 6)
    Parcel[i] = substring(data$PIN[i], 7, 8)
    Multicode[i] = substring(data$PIN[i], 9, 12)
}
```

Once these variables were extracted, the `Description` variable was dropped from the data set. We noticed some houses with strange recording like "42 bathrooms and 7 rooms", so we dropped any listings where the number of bedrooms and number of bathrooms was greater than the total recorded number of rooms.

```
# Adding features of descriptions, removing descriptions
data = data %>%
    mutate(
      Sell_Date = mdy(sell_date),
```

```
      Rooms = as.numeric(rooms),
      Bedrooms = as.numeric(bedrooms),
      Baths = as.numeric(baths)
    ) %>%
    select(-Description)

# Removing rows where num bathrooms/bedrooms exceeds number of rooms
data = data %>%
    filter(Bedrooms < Rooms | Baths < Rooms)
```

Next, we dropped any listings with a missing value in at least one of the variables.

```
# Removing rows with at least 1 missing value
data = data %>%
    drop_na()
```

We also wanted to deal with outliers, so a function was written that would identify a listing as an outlier if it was greater than 3 standard errors away from the mean value and used this to remove outliers for variables where the maximum value was significantly higher than the 3rd quartile.

```
# Filtering extreme observations
is_outlier = function(x){
    result = abs(x - mean(x)) > 3*sd(x)
    return(result)
}

# Removing outliers for variables where max() is greater than q3()
data = data %>%
    filter(!is_outlier(`Sale Price`),
           !is_outlier(`Land Square Feet`),
           !is_outlier(Baths),
           !is_outlier(`Lot Size`),
           !is_outlier(`Town and Neighborhood`),
           !is_outlier(`Age Decade`),
           !is_outlier(`Age`),
           !is_outlier(`Estimate (Land)`),
           !is_outlier(`Estimate (Building)`),
           !is_outlier(`Building Square Feet`),
           !is_outlier(`Other Improvements`))
```

Finally, we removed all spaces from variable names and replaced them with underscores.

```
# Removing Spaces
data = data %>%
  rename(Property_Class = `Property Class`,
         Neighborhood_Code = `Neighborhood Code`,
         Land_Square_Feet = `Land Square Feet`,
         ...
         Age_Decade = `Age Decade`,
         Neighborhood_Code_Mapping = `Neigborhood Code (mapping)`,
         Town_and_Neighborhood = `Town and Neighborhood`,
         )
```

This cleaned data set was written as `data_cleaned.csv` so it could easily be reloaded for the remainder of the analysis.

```
# Save cleaned data
write_csv(data, "data_cleaned.csv")
```

2.2 Exploration of Data

Within the data, latitude and longitude coordinates were provided for each listing. A map of the sale price of listings is overlaid on a satellite map of the region (Figure 1). Here, it can be seen that a lot of the higher price listings are on the water, with sale price generally decreasing the more in-land the listing is.

```
# Spatial Map of Sale Price
ggmap(Map, darken = c(0.1, "white")) +
    geom_polygon(data = cook_map, aes(x = long, y = lat),
        fill = NA, color = "orange") +
    geom_point(data = data,
                aes(x = Longitude,
                    y = Latitude,
                    color = Sale_Price),
                size = 0.02,
                alpha = 0.75) +
    scale_color_gradient(low = "#6fe7f7", high = "#890000") +
    labs(x = "Longitude", y = "Latitude", color = "Sale Price ($)")
```

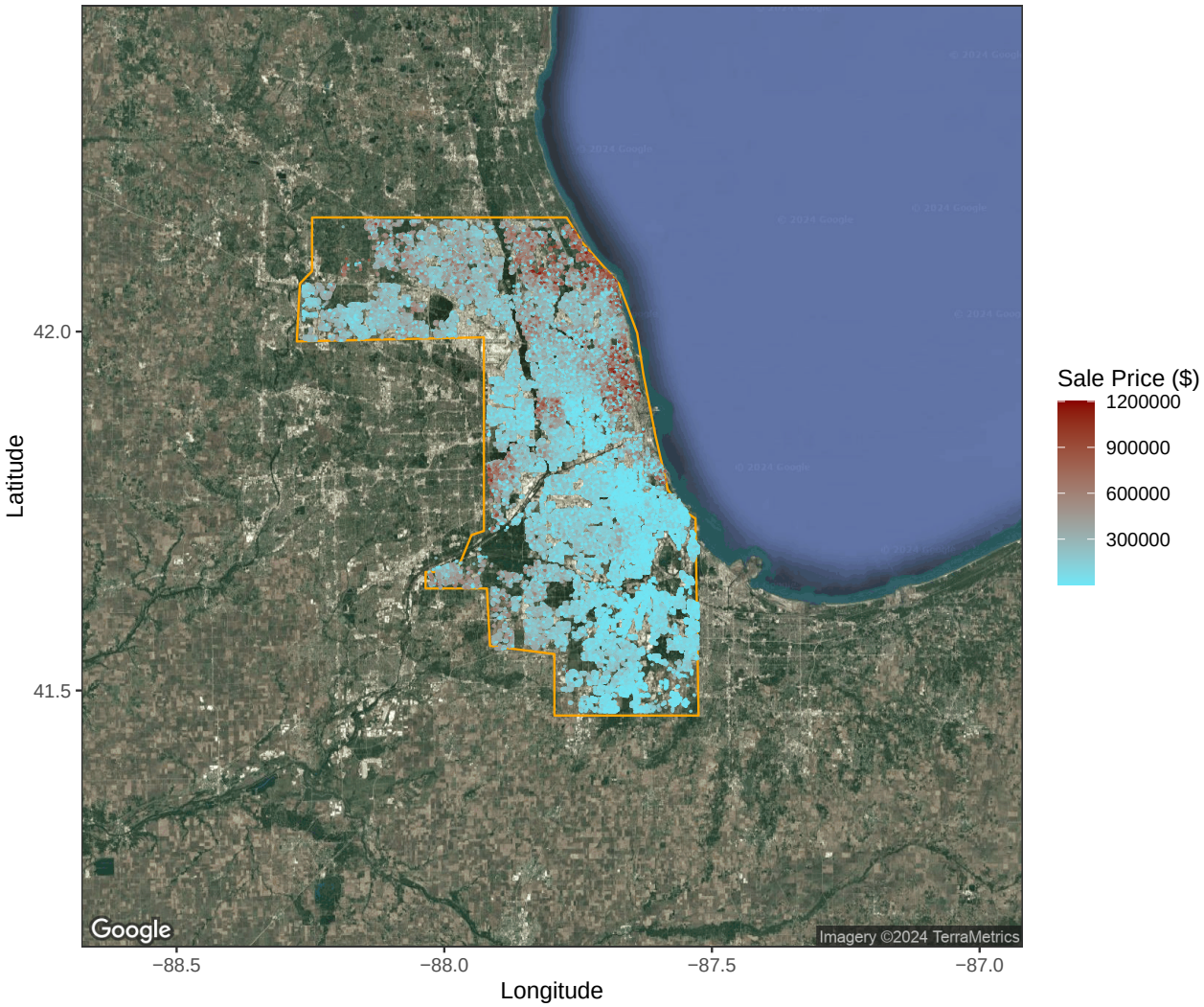

Figure 1: A spatial map of the sale prices over the region.

Multiple boxplots were created for all of the variables pertaining to a garage against the response (Figure 2). As the first garage increases in size, so too does the sale price. Having a garage attached to the building (`Garage_#_Attachement` = 1) is also associated with a higher sale price for both garage 1 and garage 2. Higher quality materials (larger values of `Garage_#_Material`) is associated with higher sale prices too.

```
# Plotting Against Garage Variables
data %>%
    select(Sale_Price,
           Garage_1_Area,
           Garage_1_Size,
           Garage_1_Attachment,
           Garage_1_Material,
           Garage_2_Area,
           Garage_2_Size,
```

```
            Garage_2_Attachment,
            Garage_2_Material) %>%
    pivot_longer(cols = 2:9, names_to = "Variable", values_to = "Value") %>%
    ggplot(aes(x = factor(Value), y = Sale_Price, group = Value)) +
    geom_boxplot() +
    facet_wrap(~Variable, scales = "free_x", nrow = 2) +
    labs(x = " ")
```

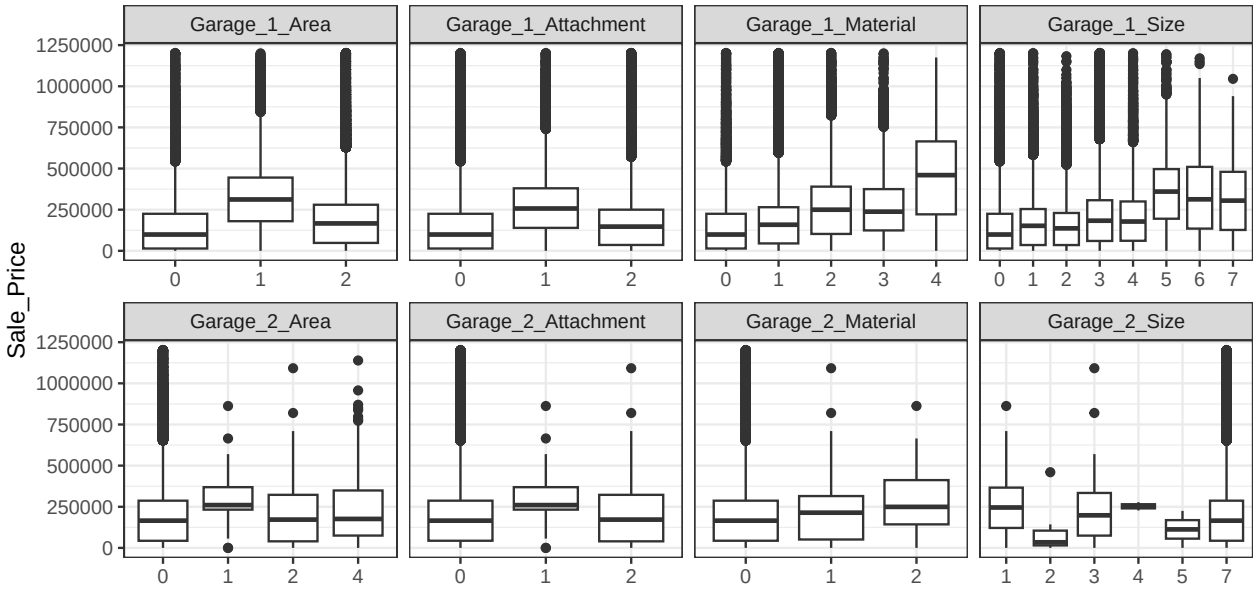

Figure 2: A series of boxplots for all garage variables against sale price.

When comparing sale price to variables related to the physical property, it can be noted that for the `Apartments` variable, there is large varibility in the "0" group, which may be due to non-apartment buildings being more expensive (Figure 3). Sale price is generally the same across attic types, porch groups, and design plans. Sale price appears to increase as the number of fireplaces increases. Additionally, sale price is higher for listings with cathedral ceilings. The `Proprty_Class` variable apperas to have equal variability in sale price for all classes except "209".

```
data %>%
    select(`Sale_Price`,
           `Property_Class`,
           `Apartments`,
           `Basement`,
           `Attic_Type`,
           `Design_Plan`,
           `Cathedral_Ceiling`,
           Fireplaces,
```

```
            Porch) %>%
    pivot_longer(cols = 2:9, names_to = "Variable", values_to = "Value") %>%
    ggplot(aes(x = factor(Value), y = Sale_Price, group = Value)) +
    geom_boxplot() +
    facet_wrap(~Variable, scales = "free_x", nrow = 2) +
    labs(x = " ")
```

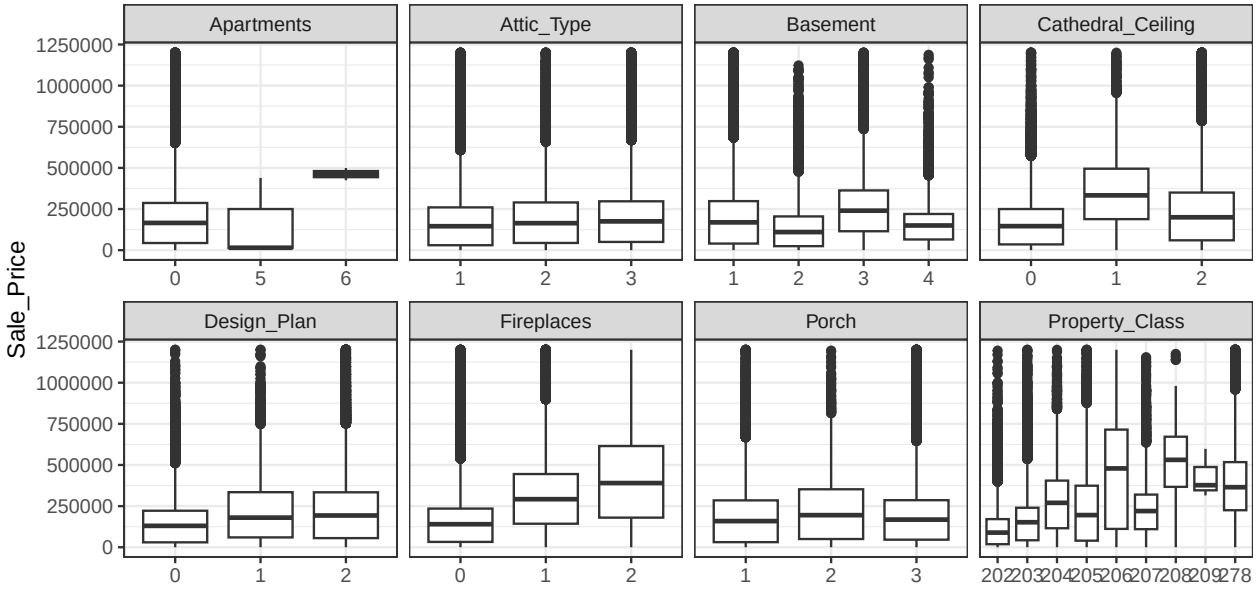

Figure 3: A series of boxplots for variables related to the property itself.

Additional boxplots were created for the number of rooms in each listing (Figure 4). For all room variables, there is generally an increase in sale price as the number of rooms increases. However, the sale price for listings with 1 room is higher, on average, than all other room groups. The same goes for listings with 0 bedrooms.

```
# Plotting Against Room Variables
data %>%
    select(`Sale_Price`,
           `Bedrooms`,
           `Baths`,
           `Rooms`) %>%
    pivot_longer(cols = 2:4, names_to = "Variable", values_to = "Value") %>%
    ggplot(aes(x = factor(Value), y = Sale_Price, group = Value)) +
    geom_boxplot() +
    facet_wrap(~Variable, scales = "free_x") +
    labs(x = " ")
```

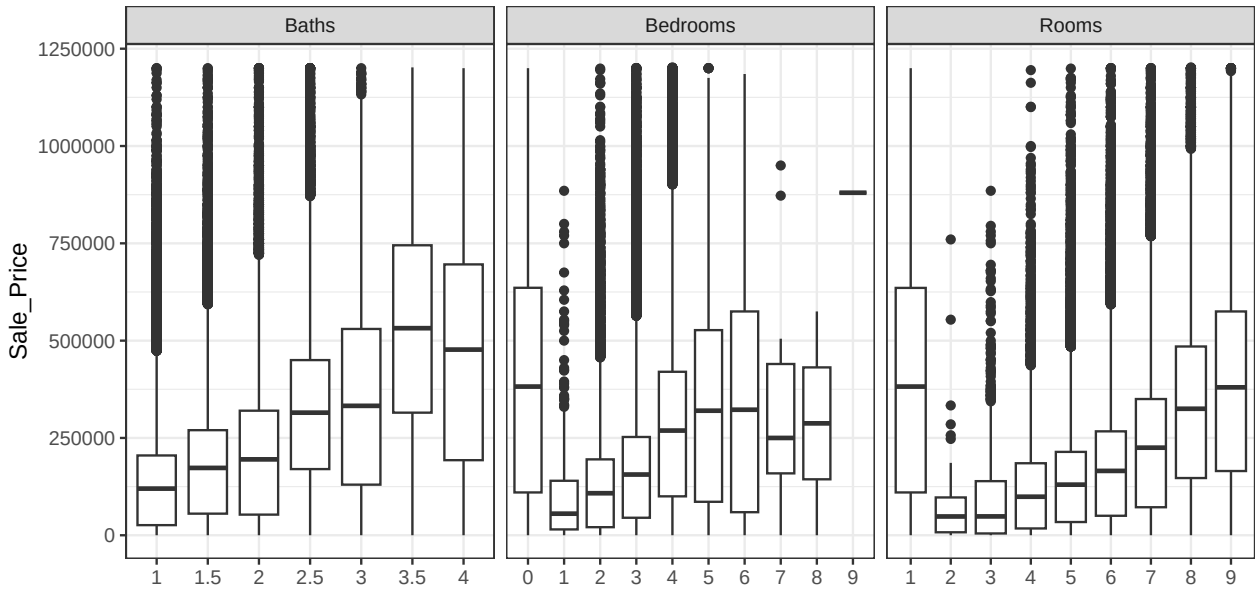

Figure 4: A series of boxplots for the variables relating to number of rooms present.

The final visualization of interest is a scatterplot of sale price against both building square footage and land square footage (Figure 5). Here, it can be seen that for listings with high building square footage, we see higher sale price. The relationship is the same for land square footage, although the highest sale prices occur at high building square footage and average land square footage.

```
data %>%
    ggplot(aes(x = `Building_Square_Feet`,
               y = `Land_Square_Feet`,
               color = `Sale_Price`)) +
    geom_point(size = 2) +
    labs(x = "Building Square Footage",
        y = "Land Square Footage",
        color = "Sale Price ($)")
```

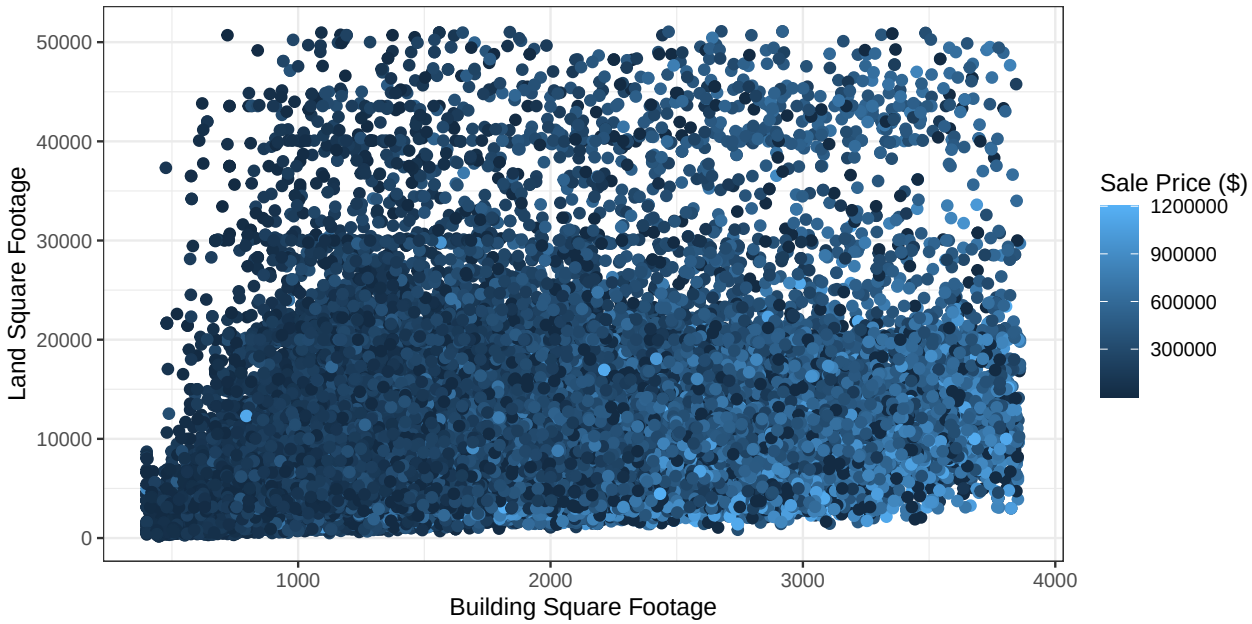

Figure 5: A scatterplot of the relationship between sale price and building/land square footage.

2.3 Hypothesis Development

From the exploration performed above, we hypothesize that building square footage, land square footage, the number of rooms, and location are likely going to have the highest impact on sale price.

3 Model Development and Performance Evaluation

Before starting the modelling process, variables were factored (if categorical) or scaled to have mean of 0 and variance 1 (if quantitative). This ensured that models had the best chance of converging. Additionally, some variables were removed due to having only 1 value or being of no importance.

```
# Scaling numeric variable
Model_Data = data %>%
    select(-c(Modeling_Group,
              Age,
              Use,
              Sale_Half_of_Year,
              `...1`,
              PIN,
              Census_Tract,
              Deed_No,
              Town_and_Neighborhood,
```

```
            Neighborhood_Code_Mapping)) %>%
    mutate(Land_Square_Feet = scale(Land_Square_Feet),
            Building_Square_Feet = scale(Building_Square_Feet),
            Estimate_Land = scale(Estimate_Land),
            Estimate_Building = scale(Estimate_Building),
            Lot_Size = scale(Lot_Size)) %>%
    filter(Sale_Price > 499)
```

At this point, the data were split into training and testing sets by randomly sampling 80% of the rows for the training set and leaving the remaining rows for the testing set.

```
# Sampling rows at random
ids_train = sample(1:nrow(Model_Data),
                   size = round(0.8*nrow(Model_Data)),
                   replace = FALSE)

# Splitting data
train = Model_Data[ids_train,]
test = Model_Data[-ids_train,]
```

To find the best Simple Linear Regression model, all single predictor models were searched and compared on AIC. The top-performing model involved the predictor `Estimate_Building` to predict sale price. A summary of the model fit using K-fold cross-validation is shown below. Note that the R^2 indicates fairly poor model fit, but this is to be expected as it is only a single predictor.

```
# Define cross-validation method with 5 folds
train_control <- trainControl(method = "cv",
                              number = 5)


# Build model using linear model
slr <- train(Sale_Price~Estimate_Building, data = train,
                trControl = train_control,
                method = "lm")

# Summary of model
print(slr)
```

```
## Linear Regression
##
## 126664 samples
##      1 predictor
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101331, 101331, 101332, 101331, 101331
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   111210.9  0.6496349  71987.55
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
summary(slr$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -878792  -60524  -11826   40506 1185718
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       242180.6      312.5   774.9   <2e-16 ***
## Estimate_Building 150668.7      310.9   484.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 111200 on 126662 degrees of freedom
## Multiple R-squared:  0.6496, Adjusted R-squared:  0.6496
## F-statistic: 2.348e+05 on 1 and 126662 DF,  p-value: < 2.2e-16
```

```
slr_preds = predict(slr, test)
slr_mse = Metrics::mse(slr_preds, test$Sale_Price)
print(paste("MSE from Test Set: ", slr_mse))
```

```
## [1] "MSE from Test Set:  11706639710.3668"
```

To find the best Multiple Linear Regression model, we used all available predictors with no interactions. A summary of the cross-validation process is provided. Then, we predicted to the test data and calculated the mean squared error.

```
# Train mlr model using k-fold cross validation
mlr <- train(Sale_Price~., data = train,
               trControl = train_control,
```

```
               method = "lm")

# Print model specifications
print(mlr)
```

```
## Linear Regression
##
## 126664 samples
##     60 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101332, 101331, 101331, 101330, 101332
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   85815.31  0.7913858  58971.62
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# Predict to test data
mlr_preds = predict(mlr, test)
mlr_mse = Metrics::mse(mlr_preds, test$Sale_Price)
print(paste("MSE from Test Set: ", mlr_mse))
```

```
## [1] "MSE from Test Set:  7260118401.05626"
```

Starting with the MLR model, subset selection was performed via backwards selection. Each iteration, the term that had the lowest impact on AIC was removed until no other terms needed to be removed. Once the ideal model formula was determined, this model was re-fit using 5-fold cross-validation. The best subset selection process resulted in a model using `Lot_Size`, `Building_Square_Feet`, `Beds`, and `Baths` to predict `Sale_Price`.

```
# Refit MLR model
mlr = lm(Sale_Price ~ ., data = train)

# Determine best model
best_subset <- MASS::stepAIC(mlr, direction = "backward", scope = ~ 1)
```

```
# Refit best model
best_subset <- train(Sale_Price ~ Lot_Size + Building_Square_Feet + Bedrooms + Baths,
                     data = train,
                     trControl = train_control,
                     method = 'lm')
```

```r
# Print model specifications
print(best_subset)
```

```
## Linear Regression
##
## 126664 samples
##      4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101330, 101332, 101332, 101332, 101330
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   149066.9  0.3705092  109297.1
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
summary(best_subset$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -649135  -96554  -14594   70644 1049546
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          199179.6     2238.1   88.99   <2e-16 ***
## Lot_Size             -11636.8      446.0  -26.09   <2e-16 ***
## Building_Square_Feet  95485.4      681.8  140.05   <2e-16 ***
## Bedrooms             -10529.6      594.8  -17.70   <2e-16 ***
## Baths                 47950.8      973.3   49.27   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 149100 on 126659 degrees of freedom
## Multiple R-squared:  0.3705, Adjusted R-squared:  0.3705
## F-statistic: 1.864e+04 on 4 and 126659 DF,  p-value: < 2.2e-16
```

```r
# Predict to test data
bs_preds = predict(best_subset, test)
bs_mse = Metrics::mse(bs_preds, test$Sale_Price)
print(paste("MSE from Test Set: ", bs_mse))
```

```
## [1] "MSE from Test Set:  21953998248.3359"
```

We fit Ridge and Lasso regression models using am mixutre of the `caret` and `glmnet` packages. `glmnet` was used to find the sequence of λ values to choose from, while `caret` was used to ensure consistency with how we fit our other models. Ridge and Lasso regression can be easily fit using the `glmnet` package alone.

```r
library(glmnet)
data.train.mat <- model.matrix(Sale_Price ~. , data = train)[,-1]
data.test.mat <- model.matrix(Sale_Price ~., data = test)[,-1]


# Fitting ridge regression model
tune.grid.ridge = expand.grid(alpha = 0,
                              lambda = glmnet(data.train.mat,
                                              train$Sale_Price,
                                              alpha = 0)$lambda)
fit.ridge <- train(Sale_Price ~ ., data = train,
  method = 'glmnet',
  trControl = train_control,
  tuneGrid = tune.grid.ridge)

# Tuning Parameters
print(fit.ridge)
```

```
## glmnet
##
## 126664 samples
##     60 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101331, 101332, 101332, 101331, 101330
## Resampling results across tuning parameters:
##
##   lambda        RMSE       Rsquared   MAE
##       15142.56   86434.65  0.7887067   59256.50
##       16618.94   86496.57  0.7884718   59288.58
##       18239.26   86565.95  0.7882125   59324.89
```

```
##       20017.57   86643.26  0.7879283   59365.45
##       21969.26   86729.42  0.7876167   59411.01
##       24111.23   86825.45  0.7872752   59462.54
##       26462.05   86932.48  0.7869009   59519.96
##       29042.06   87051.77  0.7864908   59584.40
##       31873.63   87184.67  0.7860417   59656.99
##       34981.27   87332.70  0.7855503   59738.42
##       38391.90   87497.48  0.7850128   59829.47
##       42135.06   87680.80  0.7844256   59931.36
##       46243.18   87884.57  0.7837846   60045.69
##       50751.83   88110.87  0.7830857   60173.91
##       55700.07   88361.93  0.7823248   60317.76
##       61130.76   88640.16  0.7814975   60478.92
##       67090.93   88948.09  0.7805993   60659.65
##       73632.22   89288.45  0.7796259   60862.00
##       80811.27   89664.11  0.7785727   61088.25
##       88690.26   90078.02  0.7774355   61341.77
##       97337.46   90533.66  0.7762093   61625.33
##      106827.74   91033.99  0.7748916   61941.99
##      117243.31   91576.29  0.7734861   62288.71
##      128674.39   92176.10  0.7719715   62676.60
##      141219.98   92831.46  0.7703524   63105.09
##      154988.76   93546.03  0.7686253   63577.54
##      170099.97   94323.53  0.7667872   64098.24
##      186684.50   95167.71  0.7648353   64670.59
##      204886.01   96082.25  0.7627670   65298.00
##      224862.15   97070.71  0.7605803   65983.41
##      246785.93   98136.44  0.7582728   66730.76
##      270847.25   99282.51  0.7558428   67543.89
##      297254.52  100511.57  0.7532883   68423.35
##      326236.47  101825.80  0.7506076   69373.35
##      358044.13  103226.84  0.7477996   70394.89
##      392952.99  104715.73  0.7448637   71488.56
##      431265.42  106292.91  0.7418000   72653.29
##      473313.27  107958.10  0.7386100   73888.78
##      519460.72  109710.23  0.7352961   75196.54
##      570107.50  111547.36  0.7318624   76575.77
##      625692.28  113471.16  0.7283457   78028.99
##      686696.51  115469.82  0.7246897   79548.64
##      753648.57  117541.15  0.7209299   81133.62
##      827128.37  119679.42  0.7170756   82780.39
##      907772.37  121877.98  0.7131376   84481.79
##      996279.05  124129.38  0.7091281   86233.06
##     1093415.03  126425.46  0.7050608   88029.70
##     1200021.65  128757.44  0.7009503   89864.20
```

```
##     1317022.28  131116.07  0.6968124   91728.69
##     1445430.32  133491.78  0.6926635   93617.86
##     1586357.99  135874.78  0.6885204   95522.56
##     1741025.94  138255.30  0.6844002   97433.27
##     1910773.82  140623.67  0.6803196   99342.93
##     2097071.91  142970.48  0.6762951  101243.79
##     2301533.83  145286.72  0.6723426  103128.98
##     2525930.54  147563.88  0.6684767  104990.25
##     2772205.65  149794.08  0.6647109  106819.92
##     3042492.28  151970.10  0.6610573  108610.22
##     3339131.52  154085.50  0.6575264  110355.25
##     3664692.72  156134.64  0.6541268  112049.82
##     4021995.73  158113.05  0.6508665  113689.85
##     4414135.35  160015.91  0.6477476  115271.05
##     4844508.10  161840.67  0.6447766  116790.57
##     5316841.66  163585.02  0.6419554  118246.00
##     5835227.16  165247.47  0.6392843  119636.10
##     6404154.60  166827.25  0.6367626  120959.41
##     7028551.77  168324.29  0.6343882  122215.97
##     7713826.89  169739.15  0.6321581  123406.67
##     8465915.49  171072.98  0.6300685  124531.41
##     9291331.80  172327.40  0.6281147  125590.59
##    10197225.18  173504.46  0.6262917  126585.77
##    11191442.04  174606.56  0.6245939  127518.76
##    12282593.83  175636.41  0.6230156  128391.54
##    13480131.56  176596.91  0.6215507  129206.22
##    14794427.75  177491.15  0.6201931  129965.30
##    16236866.19  178322.31  0.6189367  130671.46
##    17819940.60  179093.64  0.6177756  131327.23
##    19557362.81  179808.41  0.6167037  131935.30
##    21464181.54  180469.89  0.6157153  132498.33
##    23556912.74  181081.28  0.6148048  133019.15
##    25853682.65  181645.72  0.6139668  133500.38
##    28374384.80  182166.25  0.6131963  133944.35
##    31140852.29  182645.82  0.6124883  134353.49
##    34177046.94  183087.24  0.6118383  134730.23
##    37509266.82  183493.20  0.6112419  135076.72
##    41166374.03  183866.33  0.6106995  135395.36
##    45180044.68  184208.90  0.6101975  135688.08
##    49585043.26  184523.28  0.6097378  135956.87
##    54419523.77  184811.61  0.6093168  136203.47
##    59725360.16  185075.91  0.6089316  136429.52
##    65548509.04  185318.05  0.6085792  136636.66
##    71939407.75  185539.78  0.6082570  136826.33
##    78953411.20  185742.74  0.6079624  136999.94
```

```
##    86651271.32  185928.44  0.6076932  137158.77
##    95099663.30  186098.29  0.6074472  137304.03
##   104371763.07  186253.59  0.6072225  137436.82
##   114547881.12  186395.55  0.6070173  137558.19
##   125716158.12  186525.27  0.6068300  137669.10
##   137973328.34  186651.40  0.6066595  137776.93
##   151425557.53  187867.69  0.6076299  138817.77
##
## Tuning parameter 'alpha' was held constant at a value of 0
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0 and lambda = 15142.56.
```

```
# fitting the model on the test data
ridge_pred <- predict(fit.ridge, test)
ridge_mse = Metrics::mse(ridge_pred, test$Sale_Price)
print(paste("MSE from Test Set: ", ridge_mse))
```

```
## [1] "MSE from Test Set:  7345535132.13478"
```

```
# model coefficients
# coef(fit.ridge$finalModel, s = fit.ridge$bestTune$lambda)


# Fitting Lasso regression model
tune.grid.lasso = expand.grid(alpha = 1,
                              lambda = glmnet(data.train.mat,
                                              train$Sale_Price,
                                              alpha = 1)$lambda)
fit.lasso <- train(Sale_Price ~., data = train,
                   method = 'glmnet',
                   trControl = train_control,
                   tuneGrid = tune.grid.lasso)

# Tuning Parameters
print(fit.lasso)
```

```
## glmnet
##
## 126664 samples
##     60 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101332, 101332, 101331, 101330, 101331
```

```
## Resampling results across tuning parameters:
##
##   lambda        RMSE       Rsquared   MAE
##       26.46205   85801.28  0.7914340   58896.84
##       29.04206   85798.20  0.7914486   58889.81
##       31.87363   85795.08  0.7914633   58881.49
##       34.98127   85791.39  0.7914808   58872.74
##       38.39190   85787.73  0.7914982   58862.90
##       42.13506   85783.41  0.7915188   58852.90
##       46.24318   85778.87  0.7915405   58842.49
##       50.75183   85777.26  0.7915478   58829.60
##       55.70007   85772.40  0.7915711   58817.29
##       61.13076   85767.87  0.7915928   58803.95
##       67.09093   85764.53  0.7916089   58789.68
##       73.63222   85761.25  0.7916247   58774.42
##       80.81127   85757.28  0.7916440   58759.06
##       88.69026   85752.49  0.7916675   58743.75
##       97.33746   85751.83  0.7916709   58725.31
##      106.82774   85748.50  0.7916877   58707.56
##      117.24331   85746.78  0.7916970   58688.86
##      128.67439   85749.26  0.7916859   58669.27
##      141.21998   85749.88  0.7916845   58649.17
##      154.98876   85753.84  0.7916672   58629.67
##      170.09997   85758.93  0.7916451   58609.98
##      186.68450   85769.23  0.7915980   58591.77
##      204.88601   85782.89  0.7915352   58574.63
##      224.86215   85800.94  0.7914518   58558.84
##      246.78593   85822.94  0.7913501   58544.15
##      270.84725   85850.16  0.7912238   58531.01
##      297.25452   85883.35  0.7910696   58521.23
##      326.23647   85924.26  0.7908788   58514.36
##      358.04413   85971.17  0.7906601   58510.02
##      392.95299   86026.12  0.7904039   58511.24
##      431.26542   86092.09  0.7900949   58518.33
##      473.31327   86171.71  0.7897208   58533.20
##      519.46072   86264.09  0.7892855   58554.90
##      570.10750   86372.64  0.7887724   58585.36
##      625.69228   86492.34  0.7882065   58619.94
##      686.69651   86628.86  0.7875596   58661.30
##      753.64857   86782.88  0.7868278   58709.99
##      827.12837   86957.25  0.7859969   58769.22
##      907.77237   87158.03  0.7850363   58842.86
##      996.27905   87379.42  0.7839730   58927.51
##     1093.41503   87587.02  0.7829813   58995.73
##     1200.02165   87809.41  0.7819198   59071.85
```

```
##     1317.02228   88054.85  0.7807462   59161.04
##     1445.43032   88330.81  0.7794184   59266.60
##     1586.35799   88639.23  0.7779291   59387.85
##     1741.02594   88982.96  0.7762597   59525.61
##     1910.77382   89362.81  0.7744049   59687.77
##     2097.07191   89782.83  0.7723409   59872.90
##     2301.53383   90212.09  0.7702270   60062.92
##     2525.93054   90647.86  0.7680810   60253.18
##     2772.20565   91074.10  0.7659884   60443.54
##     3042.49228   91462.05  0.7641070   60610.57
##     3339.13152   91871.91  0.7621231   60791.62
##     3664.69272   92324.30  0.7599168   61000.70
##     4021.99573   92831.25  0.7574284   61242.79
##     4414.13535   93360.94  0.7548235   61503.51
##     4844.50810   93914.52  0.7521005   61794.89
##     5316.84166   94515.50  0.7491260   62132.24
##     5835.22716   95189.48  0.7457482   62533.97
##     6404.15460   95891.57  0.7422215   62961.76
##     7028.55177   96481.90  0.7393733   63349.67
##     7713.82689   97109.49  0.7363359   63767.76
##     8465.91549   97782.21  0.7330947   64221.58
##     9291.33180   98423.39  0.7300884   64681.11
##    10197.22518   99037.15  0.7273358   65131.96
##    11191.44204   99658.64  0.7246481   65623.55
##    12282.59383  100394.43  0.7213752   66209.54
##    13480.13156  101161.22  0.7180409   66792.48
##    14794.42775  101915.00  0.7149500   67383.65
##    16236.86619  102728.85  0.7116966   68036.30
##    17819.94060  103700.70  0.7076579   68813.01
##    19557.36281  104755.72  0.7032689   69677.65
##    21464.18154  105780.76  0.6994367   70606.34
##    23556.91274  106673.12  0.6969915   71327.51
##    25853.68265  107703.58  0.6941556   72167.79
##    28374.38480  108840.43  0.6912491   73108.56
##    31140.85229  109959.88  0.6893845   74019.48
##    34177.04694  111234.76  0.6874492   75084.24
##    37509.26682  112634.42  0.6859212   76311.38
##    41166.37403  114190.06  0.6848885   77722.95
##    45180.04468  116034.82  0.6835127   79379.08
##    49585.04326  118219.94  0.6816282   81317.22
##    54419.52377  120799.45  0.6790136   83585.62
##    59725.36016  123835.14  0.6753109   86228.53
##    65548.50904  127395.64  0.6699365   89296.29
##    71939.40775  131556.19  0.6619040   92835.49
##    78953.41120  136375.46  0.6498428   96891.59
```

```
##    86651.27132  140987.04  0.6495900  100843.07
##    95099.66330  146331.34  0.6495891  105357.17
##   104371.76307  152520.08  0.6495891  110501.31
##   114547.88112  159656.27  0.6495891  116343.19
##   125716.15812  167849.50  0.6495891  122959.71
##   137973.32834  177216.07  0.6495891  130426.57
##   151425.55753  187788.58  0.6482713  138755.82
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 117.2433.
```

```
# Fitting the model on our test data
lasso_pred <- predict(fit.lasso, test)
lasso_mse = Metrics::mse(lasso_pred, test$Sale_Price)
print(paste("MSE from Test Set: ", lasso_mse))
```

```
## [1] "MSE from Test Set:  7247272334.68586"
```

```
# model coefficients
# coef(fit.lasso$finalModel, s = fit.lasso$bestTune$lambda)
```

For the model using principle components, we first selected all numeric variables except for the response and calculated the principle components. The cumulative variance is shown below, indicating that only the first two principle components are needed.

```
# Calculating principle components using numeric data
numeric_data = Model_Data %>% select_if(is.numeric) %>% select(-Sale_Price)
PCs = prcomp(numeric_data)

# Determining how many PCs to use
PCs$sdev^2/sum(PCs$sdev^2)
```

```
##  [1] 7.599511e-01 9.662336e-02 7.520483e-02 3.207665e-02 1.571743e-02
##  [6] 1.045565e-02 3.960374e-03 2.199857e-03 1.770254e-03 1.281368e-03
## [11] 3.996414e-04 2.743077e-04 4.706330e-05 3.811454e-05 1.099480e-28
```

Then, a data frame is created using the first two principle components' variable values. The observed response values were then binded to this data frame and the train-test splits were made again using the original training IDs. The model was trained on the training data using the 5-fold cross-validation and predictions were made on the test set.

```r
# Obtain first principle component as data
PC_data = data.frame(PCs$x[,1:2])

# Convert into data frame, adding response
PC_data = bind_cols(PC_data, data.frame(Sale_Price = Model_Data$Sale_Price))

# Split into test and train
PC_train = PC_data[ids_train,]
PC_test = PC_data[-ids_train,]

# train model with k-fold cv
pc_slr <- train(Sale_Price~., data = PC_train,
                trControl = train_control,
                method = "lm")

# display model
print(pc_slr)
```

```
## Linear Regression
##
## 126664 samples
##      2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101331, 101332, 101331, 101331, 101331
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   186970.1  0.009652362  137585.4
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
summary(pc_slr$finalModel)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -280284 -127403  -41191   73678  996501
##
## Coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 244287.11     525.35  465.00   <2e-16 ***
## PC1           1861.00      57.41   32.42   <2e-16 ***
## PC2           2192.22     161.07   13.61   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 187000 on 126661 degrees of freedom
## Multiple R-squared:  0.009655,   Adjusted R-squared:  0.009639
## F-statistic: 617.4 on 2 and 126661 DF,  p-value: < 2.2e-16
```

```
# predict to test data
pc_slr_preds = predict(pc_slr, PC_test)
pc_mse = Metrics::mse(pc_slr_preds, PC_test$Sale_Price)
print(paste("MSE from Test Set: ", pc_mse))
```

```
## [1] "MSE from Test Set:  34419311090.9157"
```

```
# Loadings from first PC
PCs$rotation[,1]
```

```
##     Land_Square_Feet Building_Square_Feet        Estimate_Land
##         0.0008882040         0.0010883960         0.0066974675
##    Estimate_Building            Longitude             Latitude
##         0.0010046124         0.0002311904        -0.0005799016
##            Sale_Year         Sale_Quarter       Sale_Half_Year
##         0.2162570564         0.8730671078         0.4360019681
##   Sale_Month_of_Year           Age_Decade             Lot_Size
##         0.0232759222         0.0171276107         0.0008882040
##                Rooms              Bedrooms                Baths
##         0.0016421643         0.0012608425         0.0007531130
```

A similar process was performed for the non-linear model as for the principle component model. We start by calculating the orthogonal 4^{th} degree polynomial variables from the `Estimate_Building` variable, renaming each term as `Poly_#`. This data set was binded with the original data as to have all polynomial variables and the response in a single data frame. Then, train and test splits were made. The model was trained on the training set and predicted to the testing set, resulting in the MSE provided.

```
# Adding polynomial terms for later use
poly_predictors = data.frame(poly(Model_Data$Estimate_Building, 4))

Polynomial_Data = poly_predictors %>%
    rename(Poly_1 = X1,
```

```
            Poly_2 = X2,
            Poly_3 = X3,
            Poly_4 = X4) %>%
    bind_cols(Model_Data)

# Split polynomial data
poly_train = Polynomial_Data[ids_train,]
poly_test = Polynomial_Data[-ids_train,]

# Training 5th degree polynomial
poly_slr = train(Sale_Price ~ Poly_1 + Poly_2 + Poly_3 + Poly_4,
                 data = poly_train,
                 trControl = train_control,
                 method = 'lm')


# Summary of model
print(poly_slr)
```

```
## Linear Regression
##
## 126664 samples
##      4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 101331, 101332, 101330, 101331, 101332
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   108801.8  0.6646246  73464.14
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# Predicting and Calculating MSE
poly_slr_preds = predict(poly_slr, poly_test, type = 'raw')
poly_mse = Metrics::mse(poly_slr_preds, poly_test$Sale_Price)
print(paste("MSE from Test Set: ", poly_mse))
```

```
## [1] "MSE from Test Set:  11269749295.8029"
```

Comparing the models on mean squared error when predicting to the test set, we can see which model performed the best when predicting to the test data (Table 1). From this table, the top performing model on test data was the Lasso regularized model, followed closely by

Table 1: A table of MSE for the models compared when predicting to the test data.

Model	MSE
Lasso	7247272335
MLR	7260118401
Ridge	7345535132
Polynomial	11269749296
SLR	11706639710
Backward Selection	21953998248
PCA	34419311091

the MLR model. This is likely due to overfitting from the MLR model and effective variable selection for the Lasso model.

```
tibble(Model = c("SLR", "MLR", "Backward Selection",
                 "Ridge", "Lasso", "PCA", "Polynomial"),
    MSE = c(slr_mse, mlr_mse, bs_mse, ridge_mse, lasso_mse, pc_mse, poly_mse)) %>%
    arrange(MSE) %>%
    knitr::kable(caption = "A table of MSE for the models compared when predicting to th
```