CSCI 550 - Project 2

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

2024-10-25

1 Executive Summary

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$Description[i], "sold on ")[[1]][2], ", is a
    rooms[i] = str_extract_all(str_split(data$Description[i], "total of ")[[1]][2], "\\
    bedrooms[i] = str_extract_all(str_split(data$Description[i], "total of ")[[1]][2], "
    baths[i] = str_split(str_split(data$Description[i], "bedrooms, and ")[[1]][2], " of
    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.

```r
# 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.

```r
# 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 ($)")
```

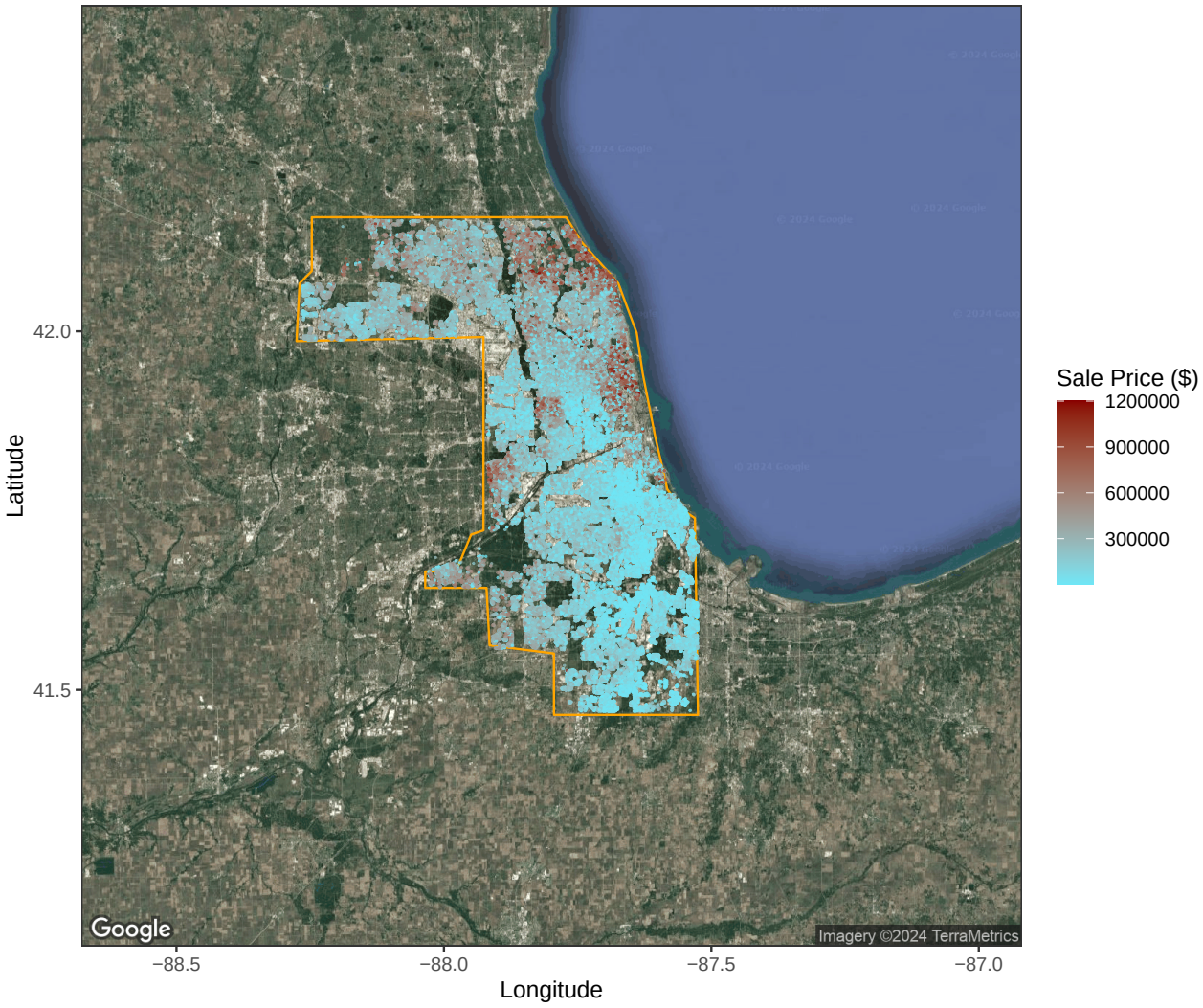

Figure 1: A spatial map of the sale prices over the region.

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 = " ")
```

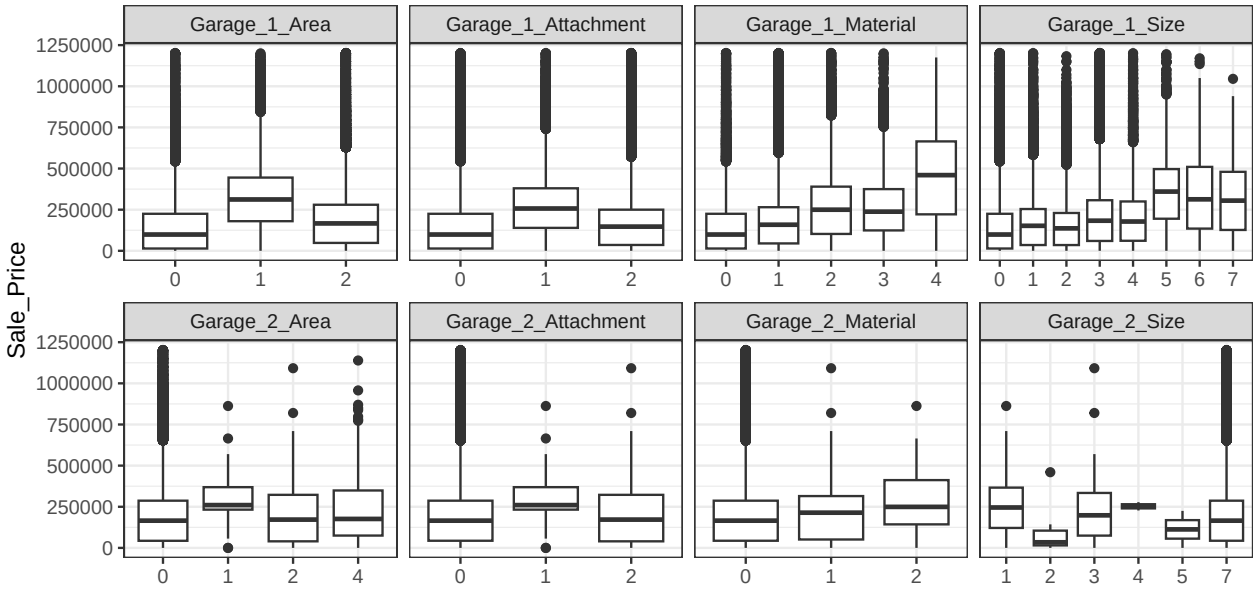

Figure 2: A series of boxplots for all garage variables against sale price.

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 = " ")
```

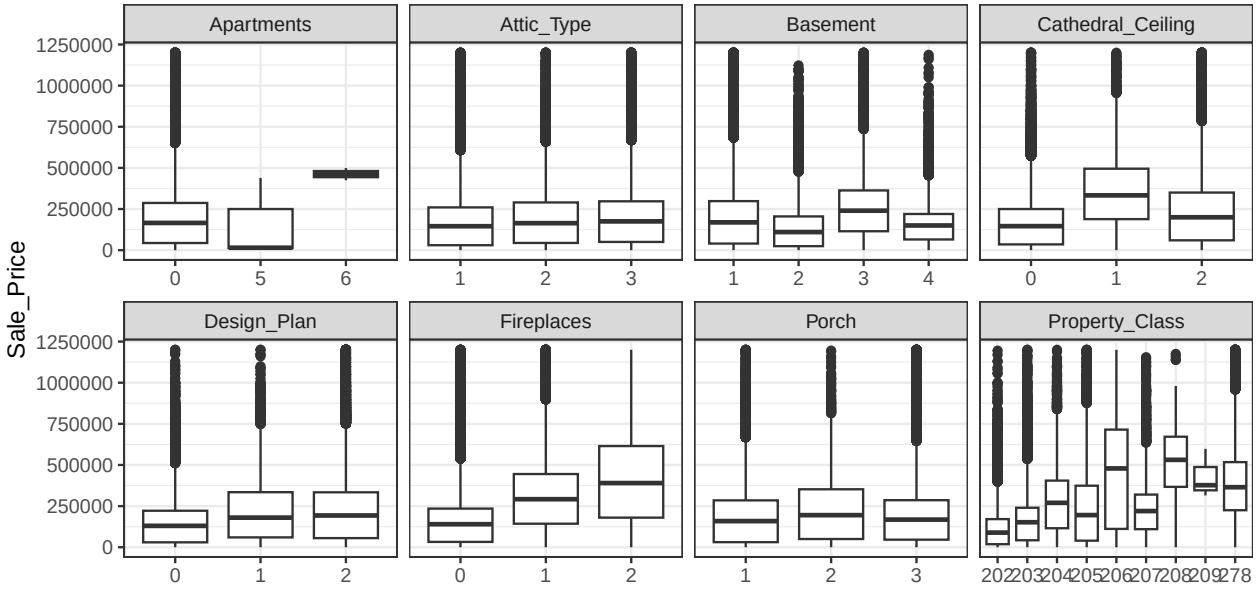

Figure 3: A series of boxplots for variables related to the property itself.

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 = " ")
```

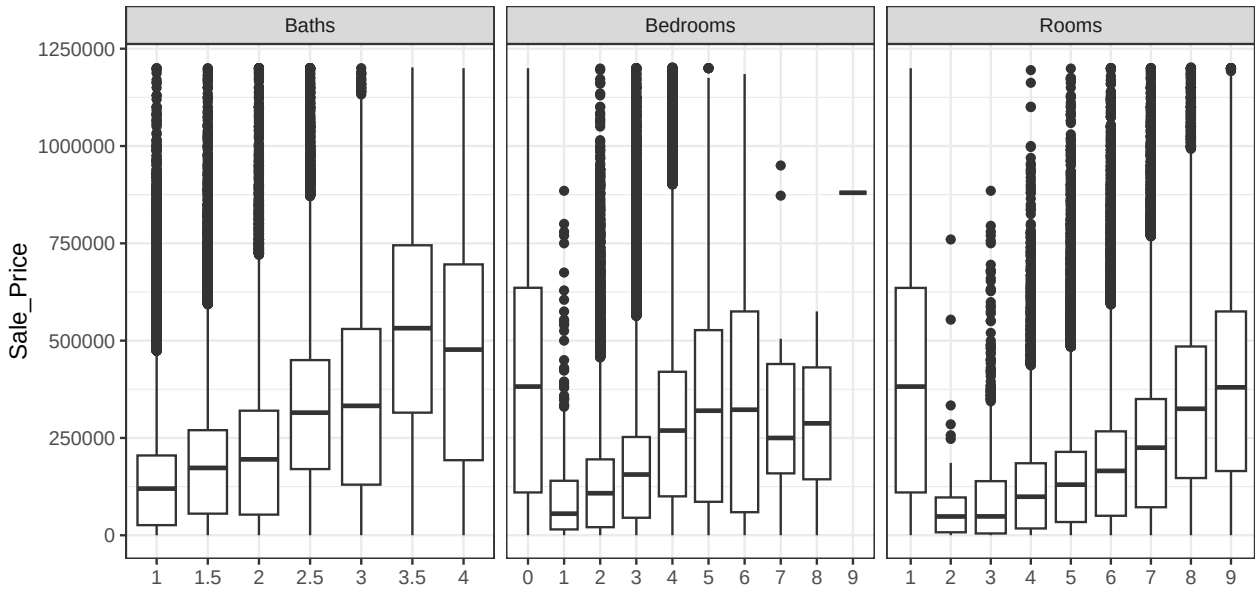

Figure 4: A series of boxplots for the variables relating to number of rooms present.

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 ($)")
```

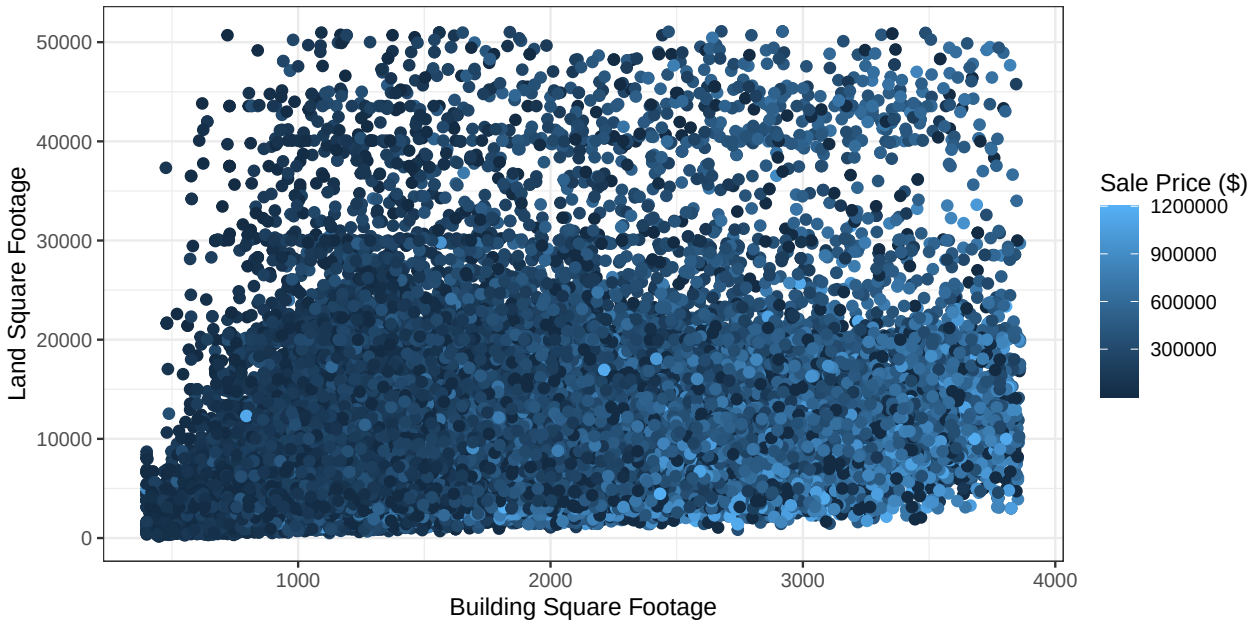

Figure 5: A scatterplot of the relationship between sale price and building/land square footage.

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))
```

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
##
## 153672 samples
##      1 predictor
##
## No pre-processing
```

```
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 122937, 122937, 122939, 122938, 122937
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   145174.7  0.4401627  97681.94
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

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

```
## [1] "MSE from Test Set:  21567219262.6313"
```

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 mly model using k-fold cross validation
mlr <- train(Sale_Price~., data = train,
               trControl = train_control,
               method = "lm")

# Print model specifications
print(mlr)
```

```
## Linear Regression
##
## 153672 samples
##     60 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 122938, 122936, 122937, 122939, 122938
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   101423.7  0.7267419  70937.59
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

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

```
## [1] "MSE from Test Set:  10429798940.6481"
```

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 principle component is 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] 7.574198e-01 9.818933e-02 7.629277e-02 3.227175e-02 1.537802e-02
##  [6] 1.047620e-02 3.995394e-03 2.156503e-03 1.789165e-03 1.285839e-03
## [11] 3.983943e-04 2.629972e-04 4.550630e-05 3.830653e-05 1.325818e-29
```

Then, a data frame is created using the first principle component's 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.

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

# 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
##
## 153672 samples
##      1 predictor
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 122937, 122937, 122938, 122937, 122939
## Resampling results:
##
##   RMSE      Rsquared     MAE
##   193620.7  0.004086993  146493.1
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

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

```
## [1] "MSE from Test Set:  37437607836.8467"
```

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

```
##     Land_Square_Feet Building_Square_Feet        Estimate_Land
##         0.0004719429         0.0014993774         0.0077748162
##    Estimate_Building            Longitude             Latitude
##         0.0019213318         0.0003683583        -0.0005235448
##            Sale_Year         Sale_Quarter       Sale_Half_Year
##         0.2158928520         0.8730746372         0.4357375635
##   Sale_Month_of_Year           Age_Decade             Lot_Size
##         0.0280297168         0.0201215289         0.0004719429
##                Rooms             Bedrooms                Baths
##         0.0018168899         0.0013253760         0.0009427495
```

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
##
## 153672 samples
##      4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 122939, 122937, 122938, 122937, 122937
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   143941.7  0.4495825  98931.89
##
## 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')
print(paste("MSE from Test Set: ", Metrics::mse(poly_slr_preds, poly_test$Sale_Price)))
```

```
## [1] "MSE from Test Set:  21249303121.8831"
```