Predicting Ventura County Real Estate Prices

Joseph Gonzalez

3/16/2021

Introduction

While home seekers may find it hard to decide on their future home, some may overlook how difficult it is for home sellers to list their real estate for fair prices. In other words, the seller must decide on the right price to attract serious buyers. This can be a challenging task due to the varying home features that sellers must consider. An intuitive approach to this problem is to estimate based on the past sales of local real estate. However, this may not always be available.

In this project, we attempt to build a price prediction model for Ventura County Real Estate using real estate data from Zillow. Zillow is a real estate marketplace that delivers information on homes for sale, for rent, and not on the market. We are interested in homes for sale and Zillow provides both quantitative and qualitative data for these houses. While Zillow provides a wide range of information on these houses, this project considers the following house characteristics: price, number of bathrooms, number of bedrooms, house area, floor cost, heating existence, cooling existence, home type, garage type, total parking spaces, number of stories, house age, lot size, number of appliances included, foundation, fireplace existence, and city population. The goal is to use these variables to form an accurate regression prediction model.

Methodologies

Scraping Zillow

Terms of Service

The Zillow website states that the user must agree to not "conduct automated queries (including screen and database scraping, spiders, robots, crawlers, bypassing "captcha" or similar precautions, or any other automated activity to obtain information from the Services) on the Services." Before proceeding, we assert that this project is only intended for academic purposes and the obtained data will not be used for monetary purposes. We will not use the scraping code to produce automated calls to the Zillow website. This project uses the code for a one-time scrape only.

Scraping Process

To obtain the Zillow data, we recommend interacting with Zillow's API to find the necessary information. However, we used a traditional scraping method to obtain the data(See appendix A for initial scraping code). The initial scrape obtained the house address, price, number of beds, number of baths, house area(most in square feet), and the house page links. The secondary scrape(See appendix B) uses the house links to obtain house-specific data, like the flooring type, heating existence, cooling existence, total parking spaces, number of stories, house age, lot size, number of appliances included, foundation, and fireplace existence. After scraping, We cleaned the data to form a usable data set(Appendix C) and discovered that some variables, like foundation and fireplace existence, had many missing values.

Table 1: Numeric Variables Table

House Feature	Variable Name	Data Type	Included	Units/Levels
Price	Price	Numeric	Yes	USD($)
House Area	House_Area	Numeric	Yes	Square Feet
Parking Spaces	Total_spaces	Numeric	Yes	Parking Spaces
House Age	Age	Numeric	Yes	Years
Lot Area	Lot_Size	Numeric	Yes	Square Feet
Appliances Included	Appliances	Numeric	Yes	Appliances
City Population	population	Numeric	Yes	People

Table 2: Categorical and Logical Variables Table

House Feature	Variable Name	Data Type	Included	Units/Levels
Floor Cost	Flooring	Categorical	Yes	Low, Medium, High
Heater	Heating	Logical	Yes	TRUE, FALSE
A/C	Cooling	Logical	Yes	TRUE, FALSE
Home Type	Home_type	Categorical	Yes	SingleFamily, Condo, Mobile
Garage	Garage_type	Categorical	Yes	Three, Two, One, none
Stories	Levels	Categorical	Yes	1, 2, 3+
Foundation Type	NA	Categorical	No	Various Categories
Fireplace	NA	Logical	No	TRUE, FALSE

Data Location

The real estate data originates from the Ventura County area in Southern California. To get data from all parts of the county, we use the county's cities as search terms. These Southern California cities include Camarillo, Fillmore, Moorpark, Ojai, Oxnard, Ventura, Santa Paula, and Simi Valley.

Description of Variables

The following tables(Table 1 and Table 2) provide preliminary explanations for the variables. The table includes the house feature name, variable name, data type, and whether we included the variable in the final data set. While forming prediction models, we will treat Beds and Baths as both numeric and categorical variables(Table 3). The final data type will depend on the best prediction model.

Explanation For Non-included Variables

In table 2, the variables that are not in the final data set are foundation and fireplace. We do not include foundation and fireplace because, after scraping, there were too many missing values(more than half). In other words, there were not enough observations to form reasonable insights.

Table 3: Room Variables Table

House Feature	Variable Name	Data Type	Included	Units/Levels
Bedrooms	Beds	Numeric/Categorical	Yes	Rooms
Bathrooms	Baths	Numeric/Categorical	Yes	Rooms

Exploratory analysis

First, we will look at descriptive statistics for the variables. This includes the mean, standard deviation, and variance for numeric variables and counts for categorical variables.

Table 4: Data summary

Name	real_estate_data[, -2]
Number of rows	1025
Number of columns	16
Column type frequency:	
factor	5
logical	2
numeric	9
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	n_unique	top_counts
City	0	1.00	9	Sim: 181, Tho: 180, Oxn: 172, Ven: 152
Flooring	0	1.00	4	Uns: 272, Hig: 260, Low: 251, Med: 242
Home_Type	2	1.00	6	Sin: 701, Mob: 117, Con: 111, Tow: 86
Garage_Type	0	1.00	5	Sin: 554, Non: 221, Cov: 114, Two: 102
Levels	181	0.82	3	2: 418, 1: 381, 3+: 45

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
Heating	0	1	0.76	TRU: 780, FAL: 245
Cooling	0	1	0.56	TRU: 573, FAL: 452

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	median	min	max
Price	0	1.00	1010394.01	1303417.24	699998	75000	24500000
Beds	0	1.00	3.42	1.10	3	2	10
Baths	0	1.00	2.91	1.26	3	1	12
House_Area	8	0.99	2232.43	1377.24	1839	600	11406
Total_spaces	0	1.00	2.31	2.87	2	0	84
Age	6	0.99	35.24	22.05	36	0	126
Lot_Size	137	0.87	4628546.13	57234318.69	6614	1	996478560
Appliances	219	0.79	4.75	2.97	4	1	23
population	0	1.00	109851.32	58628.47	125613	7470	208881

The factor variables table(Table 4) shows that most observations are from Simi Valley, Thousand Oaks, Oxnard, and Ventura. We also see that most homes are singleFamily(labeled "Sin" on the table) and have a single-door garage type. It is also interesting that many homes have heating(780) and about half the observations(573) have cooling. Price and Lot_Size have relatively large maximum values, small minimum values, and large standard deviations. This may indicate that outliers and large leverage values are present in our data.

Data Visualizations

In this section, we analyze descriptive visualizations for Ventura County cities. The plots include mean, maximum, and minimum real estate prices for Ventura County cities(Figure 1).

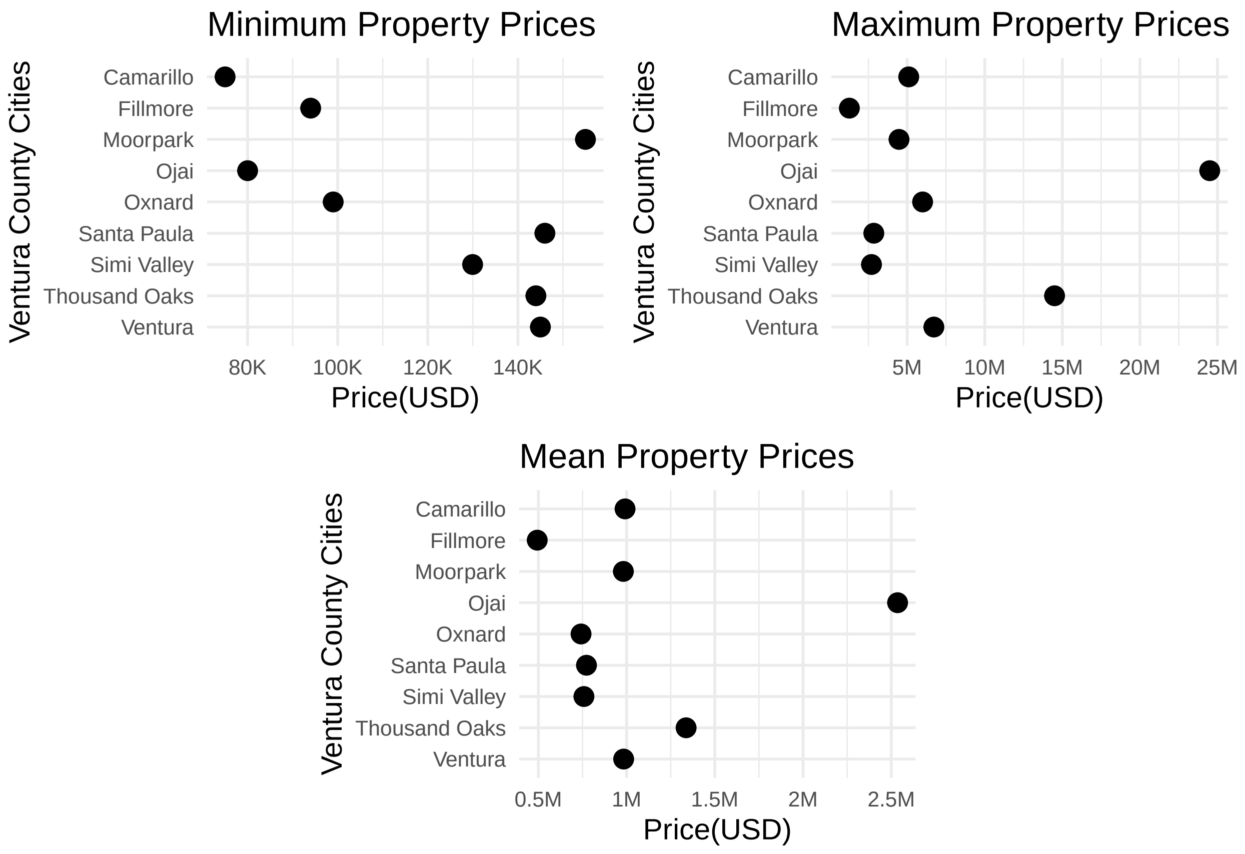


Figure 1: The dot plots in this figure represent the minimum, maximum, and mean prices for Ventura County real estate. The black points indicate the price variable. For the x-axes, the "K" represents USD in thousands of dollars and the "M" represents USD in millions of dollars.

In figure 1, we see that Camarillo has the smallest minimum sale price(around $75 thousand) and Moorpark has the largest minimum sale price(around $155 thousand). Ojai has the largest maximum sale price(almost $25 million) and Fillmore has the smallest maximum sale price(Under 2.5 million). Fillmore also has the smallest mean sale price(around $500 thousand) and Ojai has the largest mean sale price(around 2.5 million).

We also investigated the number of properties for sale, mean house area, mean lost size, and mean property age(figure 2). Simi Valley and Thousand oaks have the most real estate listings(over 175). Fillmore has the least amount of real estate listings(less than 50). Thousand Oaks also has the largest mean home area in square feet(near 2750 square feet) and Oxnard has the smallest average home area in square feet(close to 1750 square feet). Due to the large mean lot size for Santa Paula, we log-transformed the values to get a better interpretation for the mean lot sizes. From the Mean Property Lot Size plot, we can see that Santa Paula has the largest mean lot size and Simi Valley has the smallest mean property lot size. Lastly, Fillmore has the smallest mean real estate age and Ojai has the largest mean property age.

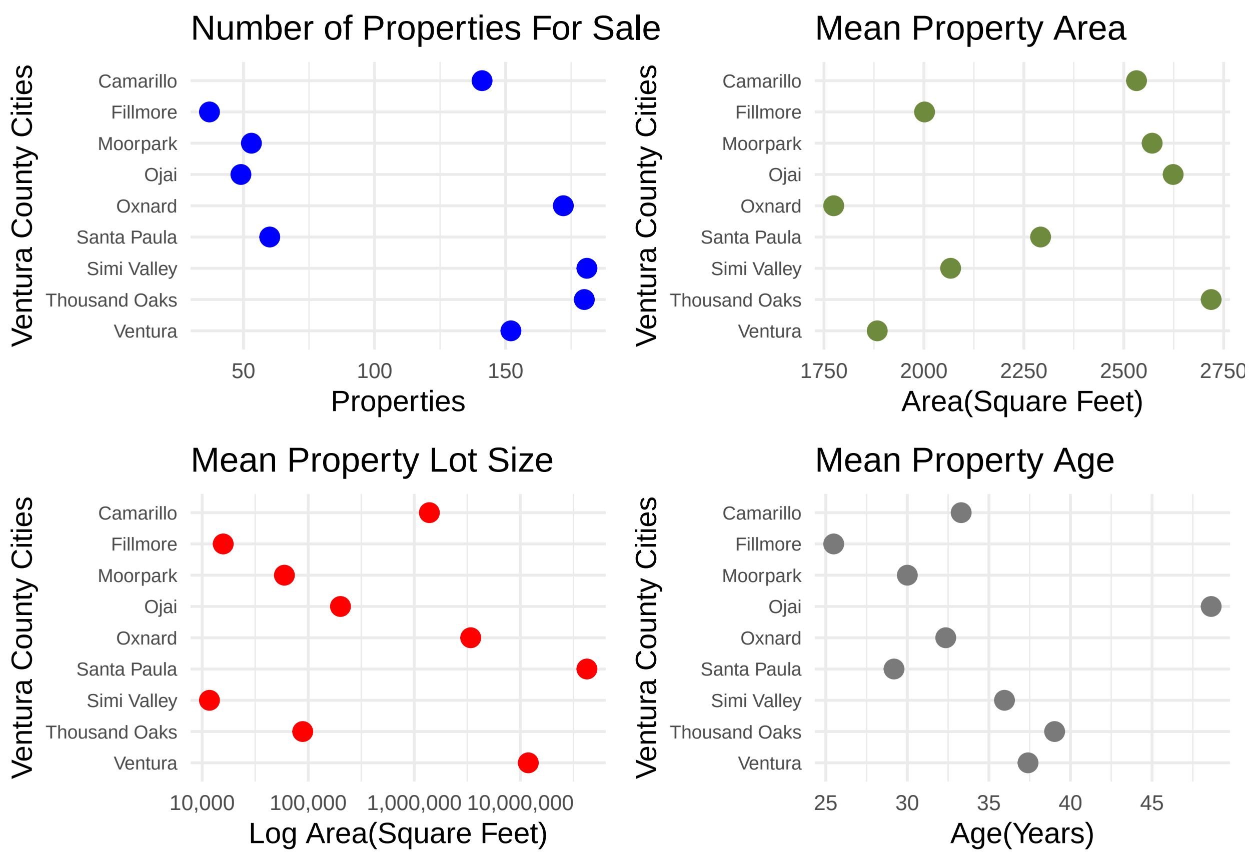


Figure 2: The dot plots in this figure represent the number of properties for sale, mean house area(square feet), mean lot area(square feet), and mean property age for Ventura County cities. The blue points indicate listing counts, dark green points indicate house area, red points indicate lot size, and gray points indicate property age.

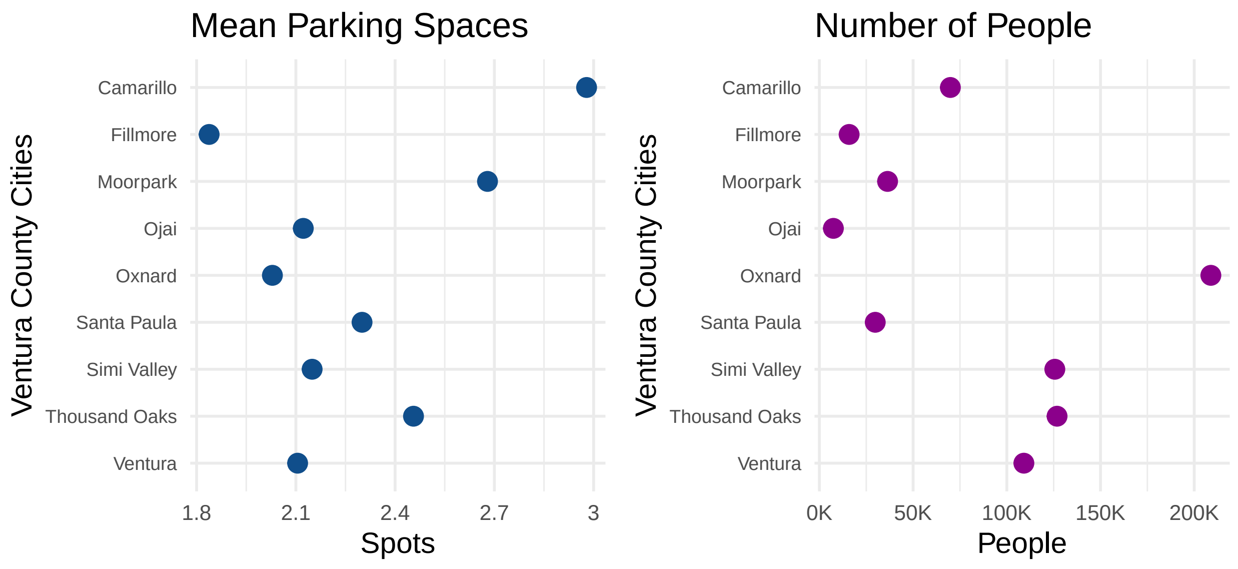


Figure 3: The dot plots in this figure represent the mean number of parking spaces, and city populations. The dark blue points indicate parking spaces and the magenta points indicate population.

In figure 3, Camarillo has the largest mean number of parking spaces and Filmore has the smallest value. In the right plot, Oxnard has the largest population and Ojai has the smallest population.

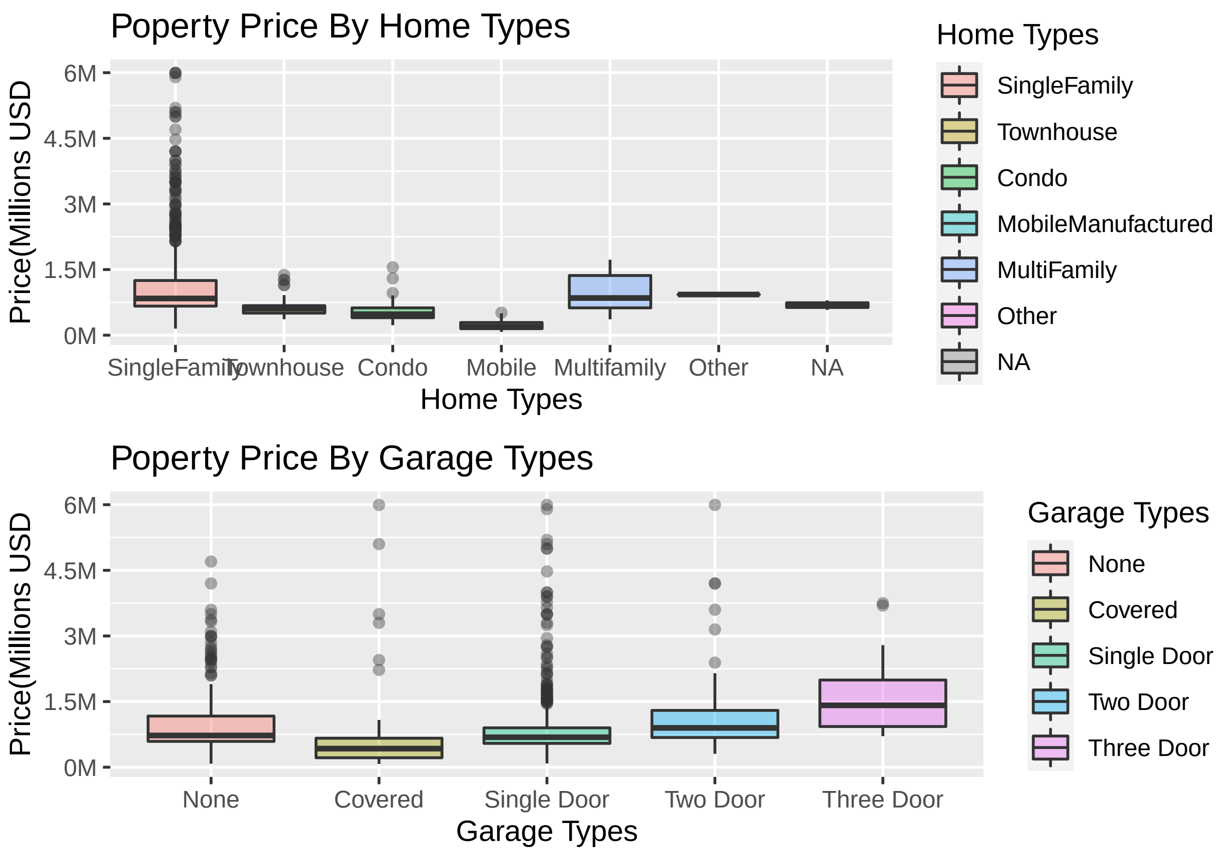


Figure 4: Box plots for real estate prices by home type and garage type. Boxplot colors indicate the category.

Next, we formed side-by-side boxplots for the categorical variables(Flooring, Home_Type, Garage_Type, Levels) with the property price and logical variables(heating, cooling) with the property price. In figure 4(top plot), we can see that SingleFamily homes and Multifamily homes have medians or distributions that are slightly larger than the other types. For the Property Price By Garage Type plot(lower plot), we see that houses with 3 garage doors tend to have a larger price.

In figure 5, we see that there are small increases in real estate prices from low to high floor cost and from one floor homes to 3+ floor homes. Figure 6 shows that there does not appear to be a distinguishable difference between the price distributions for A/C and Heater presence.

The plots in Figure 7, treat the Baths and Beds variables as categorical. When the number of beds and baths increases, we see that the median real estate price values also increase.

Lastly, figure 8 shows the real estate price variable's distribution and we can see that the distribution for price is skewed right and this is a result of the few large real estate prices. Table 4 also shows that the response variable has a very high variance. This may make it difficult to fit a good model and make accurate predictions. To control the variance, we take the log of the real estate price and we can see the resulting histogram in the right plot of figure 8. After the log transformation the distribution looks slightly more normal(slighlty satisfying the normality assumption). We maintain the log transformation on the response variable for the regression analysis.

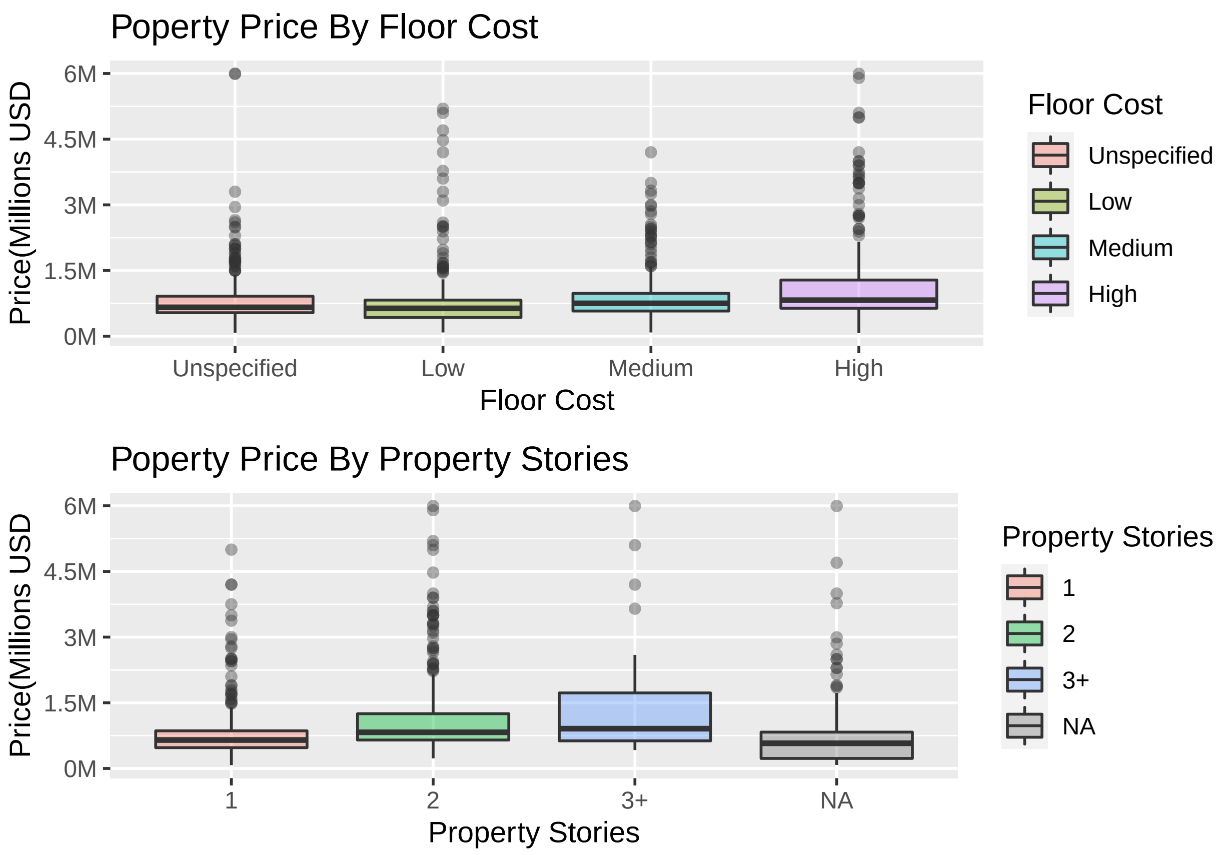


Figure 5: Box plots for real estate prices by floor cost and property levels. Boxplot colors indicate the category.

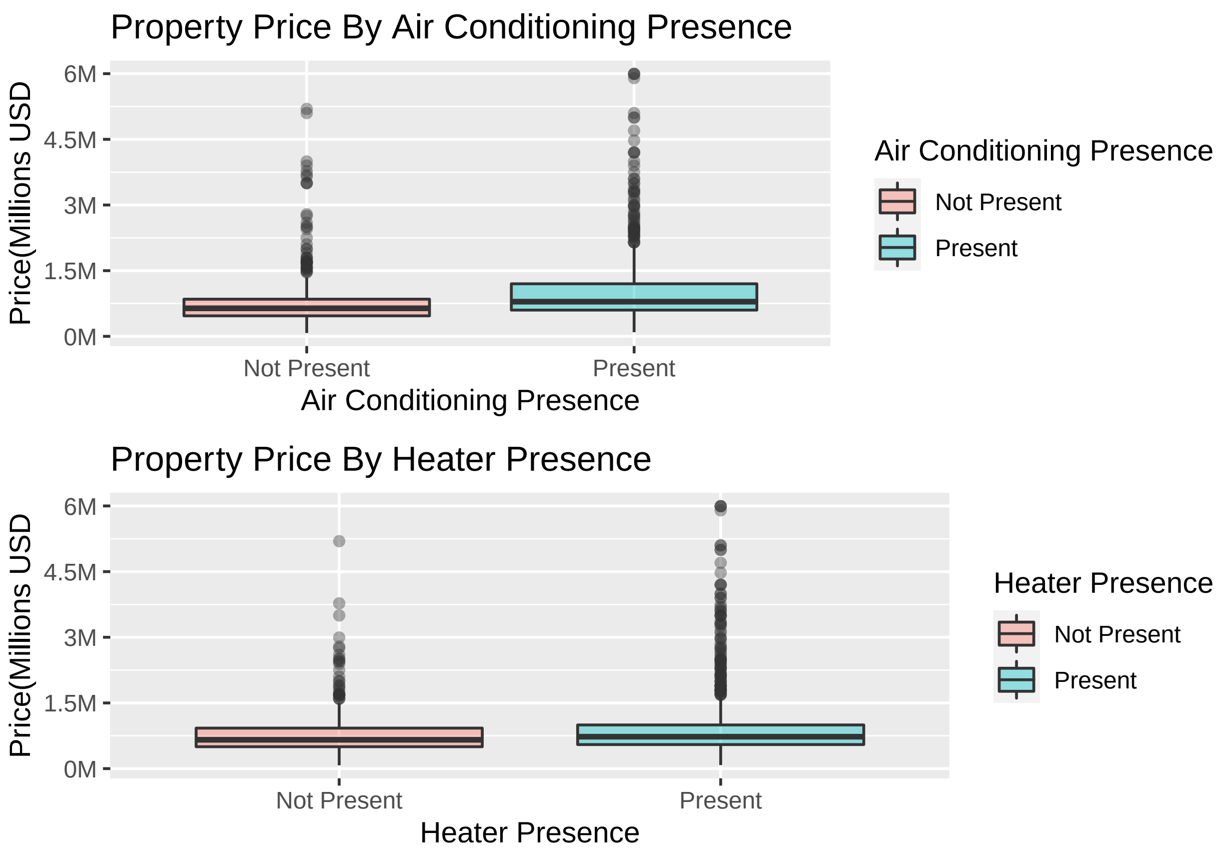
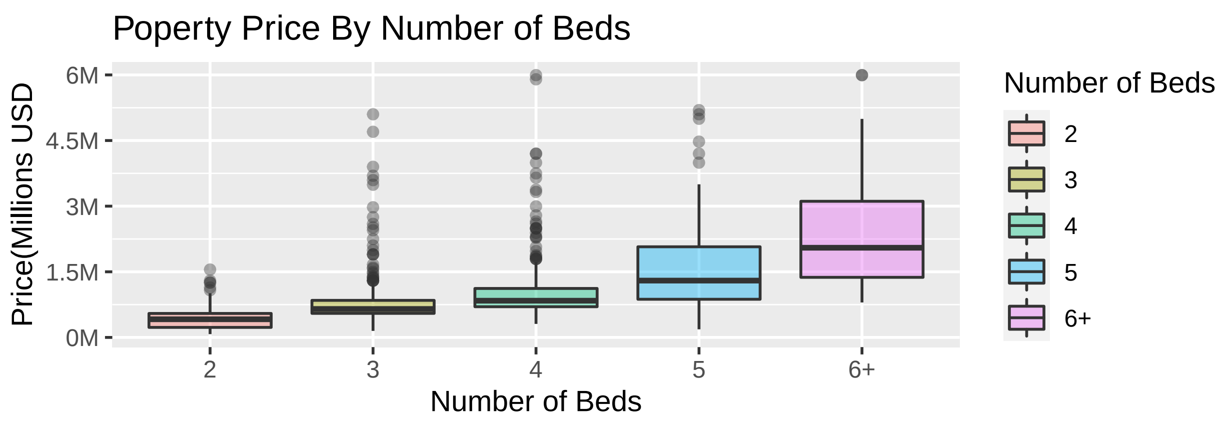


Figure 6: Box plots for real estate prices by A/C and heating existence. Boxplot colors indicate present or not present.



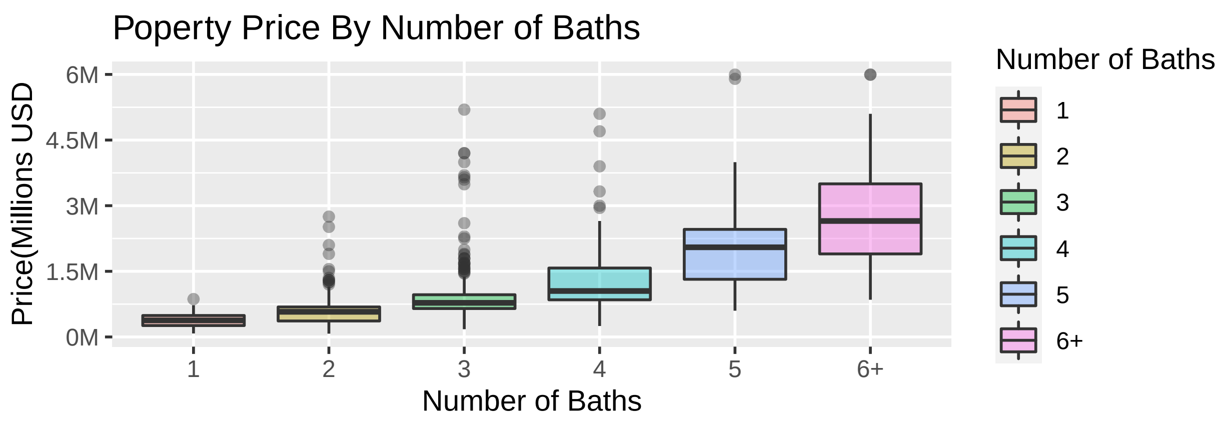


Figure 7: Box plots for real estate prices by beds and baths. Boxplot colors indicate number of rooms.

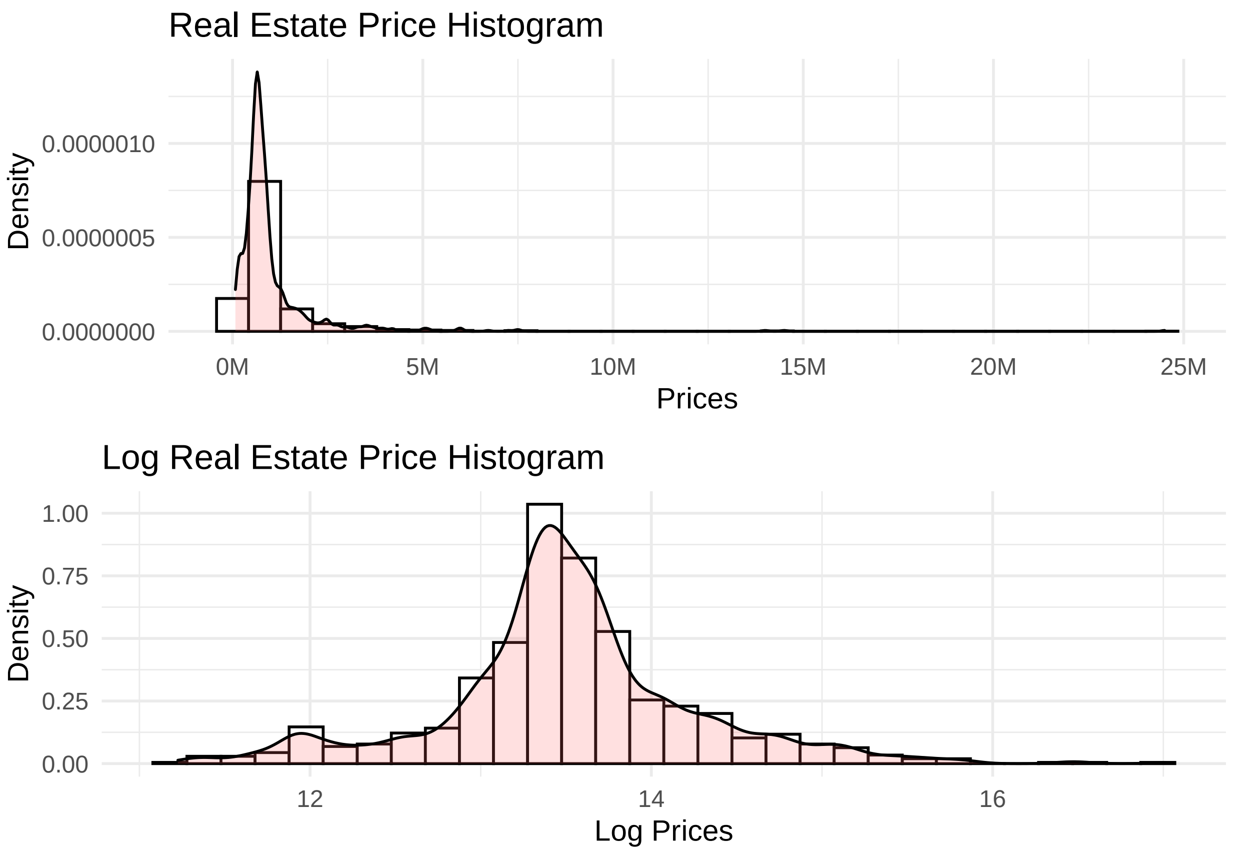


Figure 8: Histogram and density plots for real estate prices.

Building Prediction Model

For this section, we attempt to form a regression model to predict house prices. First, we list the regression assumptions and methods below.

Regression Model Assumptions and Methods

Assumptions

1. **Linearity:** The response, $\mathbf{Y}$, should be linearly associated with the predictors in $\mathbf{X}$.

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon$$

2. **Independence:** The residuals should be independent and identically distributed.
3. **Constant Variance:** The variance among the residuals should be constant.
4. **Normality:** The residuals have a normal distribution.

$$\epsilon \sim N(0, \sigma^2 \mathbf{I}_{n \times n})$$

Model Selection Criteria

We will use the best subsets, stepwise AIC, and cross-validation to select the best predictive model.

- **Best subsets**: The best subset method uses an exhaustive search procedure to find the best model that contains a specified number of predictors. The method identifies the best model as the model with the lowest sum squared error. We will not allow interactions for this method.
- **Stepwise AIC**: We use the AIC criterion for better predictive power. The stepwise AIC method uses both the forward and backward methods to add and remove variables. The method adds or removes variables that will improve/reduce the AIC. When the method cannot improve AIC further, the method stops. The starting point for this method will be the null model(no variables). We will also allow interaction terms for this method.
- **Cross-Validation:** We use cross-validation to estimate the models' mean squared prediction error. Cross-validation tests the model's ability to predict values for new data. We will select the model with the smallest cross-validation error as the best predictive model. For this project, we will utilize the 10-fold cross-validation approach.

Regression Analysis

Missing Values

From the exploratory analysis, we saw that a few variables had more than 100 missing values. The variables and their missing values are shown in table 8. The variables with the most missing values are Levels, Lot_size, and Appliances. The first option is to replace the missing values with averages or the most common occurring category. This may reduce the number of missing values, but this action makes less conservative assumptions about the houses and can increase bias on variable coefficient estimates. The second option is to remove these variables from consideration. Removing these variables will allow us to keep more *correctly entered* observations for the regression analysis(Hopefully increasing quality of fit). After interpreting the variable correlations, we proceed with the second option in this project. For house area and age, we will remove the observations with missing values from the analysis.

Table 8: Missing Values

	Missing Values
Price	0
Beds	0
Baths	0
House_Area	8
Flooring	0
Heating	0
Cooling	0
Home_Type	2
Garage_Type	0
Total_spaces	0
Levels	181
Age	6
Lot_Size	137
Appliances	219
population	0

Table 9: Pairwise Complete Variable Correlations

	Price	Lot_Size	Appliances
Price	1.00000	0.00745	0.25006
Lot_Size	0.00745	1.00000	-0.01671
Appliances	0.25006	-0.01671	1.00000

Correlations

In this section, we investigate the correlations between numeric variables. In table 10, we include bedrooms and bathrooms variables as numeric variables. Before we remove Lot_Size and Appliances, we also consider their pairwise complete correlations with price(Table 9). We use the "pairwise.complete.obs" argument to include as many observations as possible for the correlation outputs in Table 9.

Table 9 and Figure 9 show that Lot_Size has almost no linear correlation with house price and Appliances has a relatively small correlation with house price. In table 10 and figure 10, we can also see that Beds and Baths are moderately correlated with Price and House_Area is strongly correlated with Price. However, the House_Area, Beds, and Baths variables are also moderately correlated with each other. This suggests that multicollinearity may be present and, to avoid this issue, we need to check the variance inflation factors(VIF).

Table 10: Complete Variable Correlations

	Price	Beds	Baths	House_Area	Total_spaces	Age	population
Price	1.00000	0.63914	0.68910	0.75173	0.18410	-0.06119	-0.07271
Beds	0.63914	1.00000	0.67948	0.68738	0.19600	0.01317	-0.03570
Baths	0.68910	0.67948	1.00000	0.85022	0.18946	-0.25597	-0.07021
House_Area	0.75173	0.68738	0.85022	1.00000	0.25313	-0.18656	-0.13598
Total_spaces	0.18410	0.19600	0.18946	0.25313	1.00000	-0.04083	-0.04224
Age	-0.06119	0.01317	-0.25597	-0.18656	-0.04083	1.00000	-0.00925
population	-0.07271	-0.03570	-0.07021	-0.13598	-0.04224	-0.00925	1.00000

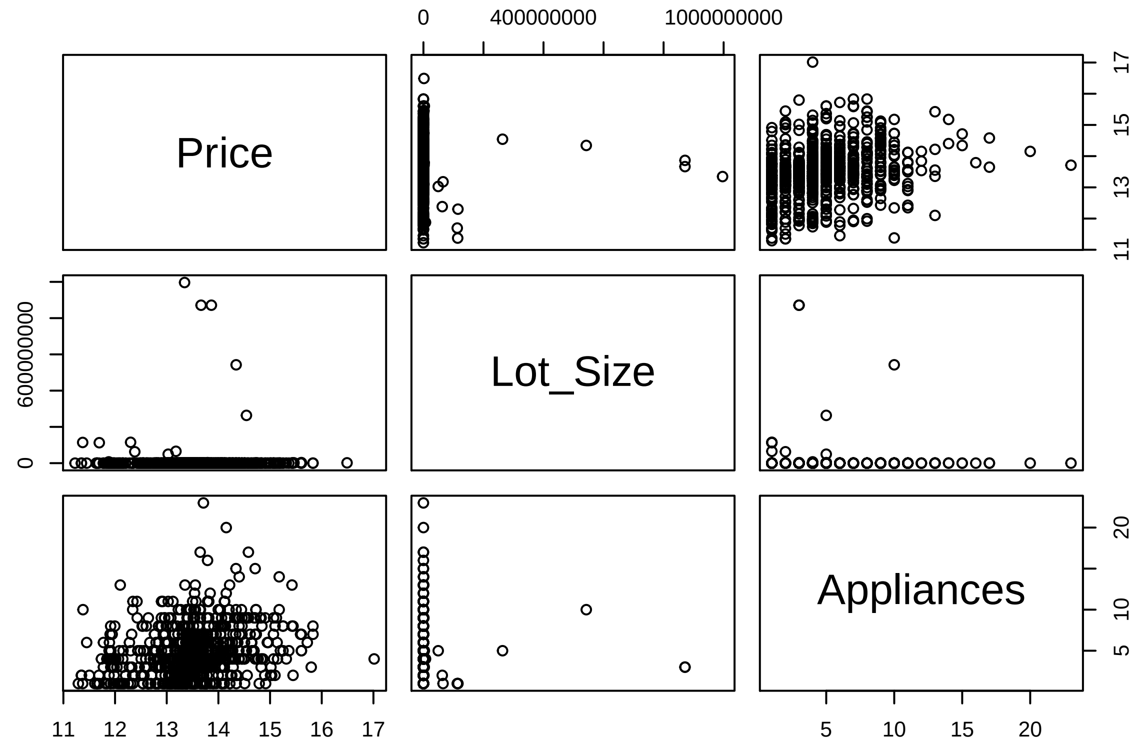


Figure 9: Pairwise correlation plots for price, lot size, and appliances. Lot size does not appear to have a linear correlation with house price and Appliances appears to have a weak positive correlation with house price.

Table 11: Full Models' F-Statistics

	Model 1	Model 2
F-statistic	96.5429881672046	76.9712027868563
P-value	< 2.2e-16	< 2.2e-16

Preliminary Fit

To test for multicollinearity, we can fit the full models on a random data subset and check the variance inflation factors(VIF). We also confirm that not all coefficients are equal to zero(significant F-statistic). In the tables 11 and 12, model 1 is the full model with numeric Baths and Beds, and model 2 is the full model with categorical Baths and Beds.

In table 11, we see that the F-statistics for both models are significant and, therefore, at least one coefficient in the models is different from zero. For Model 1, The VIF values(Table 12) for all variables are less than 5(VIF values larger than 5 are problematic and larger than 10 are serious issue for inference). For Model 2, some VIF values are more than 5 but less than 10. From these results, it does not seem that multicollinearity will cause major issues for the prediction model with no interactions(We can still get a good fit). If we add interactions, multicollinearity may increase and be an issue for inference. To be cautious, we perform the stepwise AIC method starting at the null model and, if necessary, adjust the continuous variables(subtract the mean) to avoid any possible extreme multicollinearity issues(perfect collinearity). Since the goal for this project is prediction, we only add references to VIF in the model outputs, but do not rely on VIF to decide the final model.

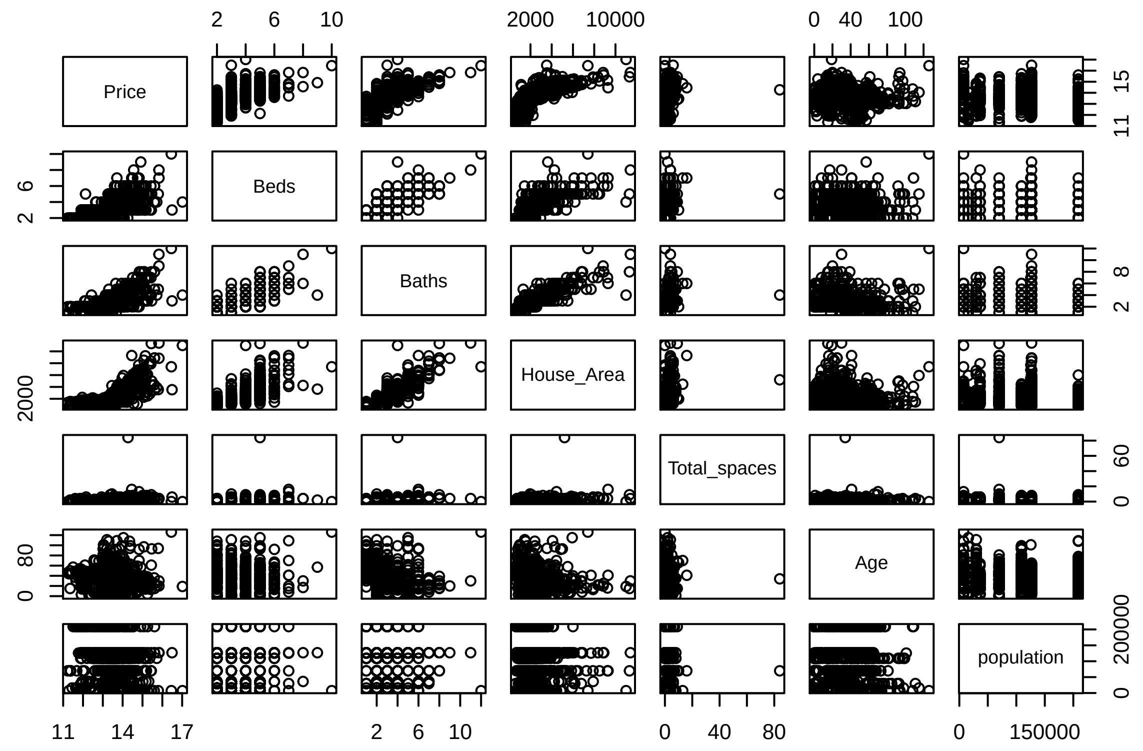


Figure 10: Pairwise correlation plots for all variables except lot size and appliances. Beds, baths, and house area seem to have positive correlations with price. Total spaces has a weak positive correlation with price. Age and population have a weak negative correlation with price.

Table 12: VIFs for full models

	Model 1	Model 2
Beds	2.786928	5.008759
Baths	4.842789	7.449946
House_Area	4.921577	4.056989
Flooring	1.380464	1.421545
Heating	1.342862	1.411542
Cooling	1.354059	1.383262
Home_Type	2.323141	2.902566
Garage_Type	1.688673	1.675145
Total_spaces	1.293345	1.098868
Age	1.286173	1.599825
population	1.123952	1.140147

Identifyig Possible Curviliear Relationships

To identify the curvilinear relationships, we plot the residuals for the full model against the predictor variables. In figure 11, there does not appear to be a curvilinear relationship between the response variable and the predictors(Same with model 2). However, there does appear to be a slight curvilinear pattern for house area in figure 10. Therefore, this may be worth investigating and, in this project, we add quadratic terms to the data set for the stepwise AIC method.

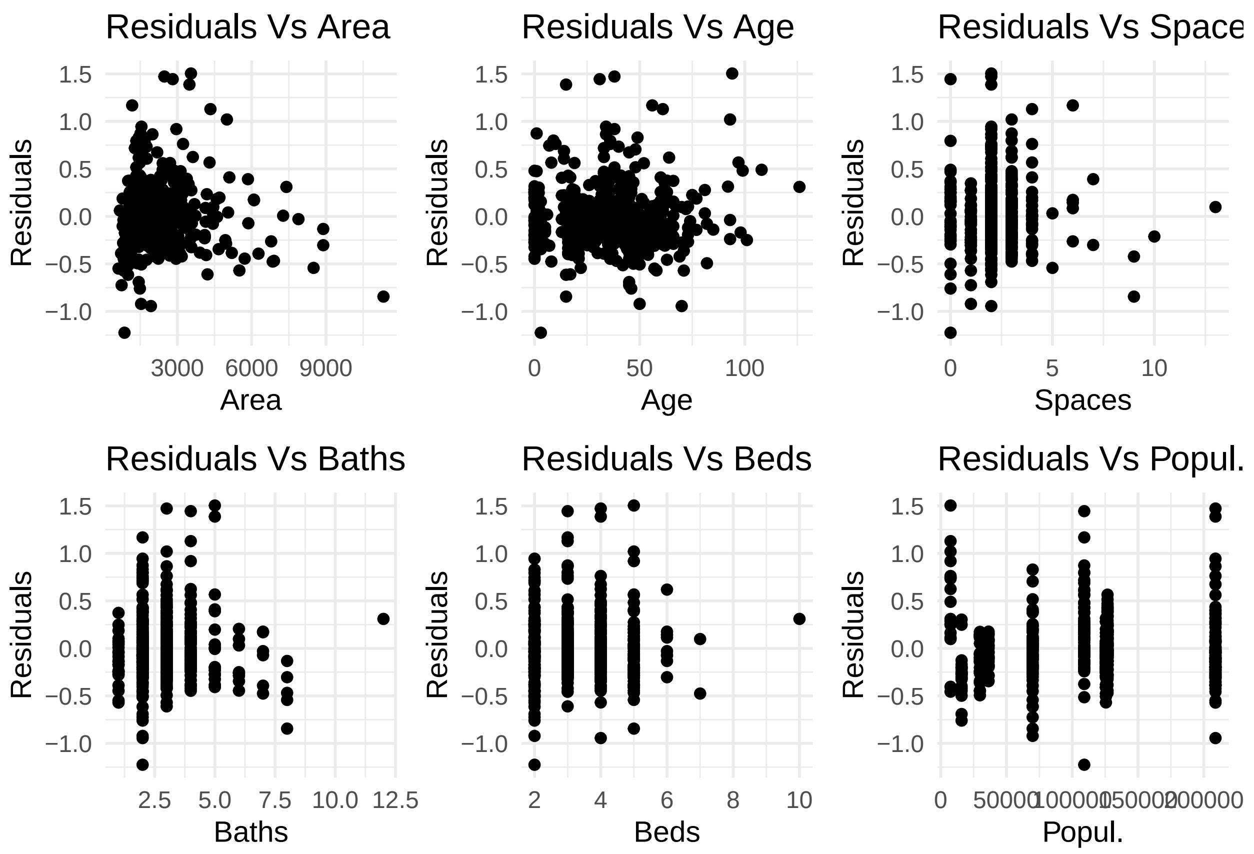


Figure 11: Residual Vs Predictor plots. Price does not seem to have a curvilinear relationship with any predictor.

Model Selection

Full Models

First, we determined the cross-validation errors for model 1 and model 2 from the *preliminary fit* section. For these full models(all variables and no interaction terms), we calculated the 10-fold cross-validation error and provided the results in table 13.

In table 13, it appears that model 2 performed better than model 1 with new data(0.1138 cross-validation error). However, Model 2 has variables with VIFs larger than 5. Model 1 has less parameters(20) and a 0.1178 CV error.

Table 13: Full Models 1 and 2

	Model 1	Model 2
Number of Parameters	20	27
CV Error	0.1178	0.1138
Largest VIF	Less than 5	More than 5

Table 14: Best Subsets Model 3 and 4

	Model 3	Model 4
Number of Variables	10	19
CV error	0.1135	0.1101
Largest VIF	Less than 5	More than 5

Best Subsets Model And Stepwise AIC Models

Using the best subsets method, we test each full model's best fitting parameter subsets. Table 14 shows the results for the best subsets method and 10-fold cross-validation. Model 3 is the best subsets method result performed on the data set with numeric beds and baths, and model 4 is the best subsets method result performed on the data set with categorical beds and baths. For this method, model 4 performed better than model 3 with new data(smaller CV error 0.1101). Model 4 also has more parameters(19) and a predictor with a VIF larger than 5.

Next, we cross-validated each model formed using the stepwise AIC method. Table 15 shows the results for the stepwise AIC method and 10-fold cross-validation. Models 5 and 6 allowed interaction terms, models 7 and 8 did not allow interaction terms, and models 9 and 10 allowed quadratic terms. The stepwise AIC method produced the odd-numbered models on the data set with numeric baths and beds, and the even-numbered models on the data set with categorical baths and beds. In table 15, we can see that model 5 performed the best with new data(CV error 0.1006). Model 5 also has a smaller cross-validation error compared to the full models and the best subsets models. Therefore, we select model 5 as the final predictive model.

Comment About Results

From the full models and selection method results(tables 13-15), we can see that most models that treated the number of beds and baths as categorical variables produced lower cross-validation errors. In this situation, treating the beds and baths as categorical variables may be better for prediction. However, it is also important to note that treating the variables as categorical produced VIFs much larger than 5. This means that these models' coefficients are not reliable and should not be used for inference. We prefer models that treat beds and baths as numeric data types, like models 3, 7, and 9, for inference(Model 9 highest VIF only slightly higher than 5).

Table 15: Stepwise AIC Models 5-10

	Model 5	Model 6	Model 7	Model 8	Model 9	Model 10
Number of Parameters	18	38	12	21	10	19
CV error	0.1006	0.1007	0.1132	0.1089	0.1084	0.1058
Interaction	Yes	Yes	No	No	No	No
Largest VIF	More than 10	More than 10	Less than 5	More than 5	More than 5	More than 5

Table 16: Selected Predictive Model Statistics

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC
0.823	0.82	0.313	271.311	0	17	-249.773	537.545

Final Model

The final predictive model treats beds and baths as numeric and includes interaction terms. We centered the numeric variables to avoid extreme multicollinearity issues. We provide the equation for the prediction model below:

Predictor Adjustments

$$\tilde{x}_{\text{house area}} = x_{\text{house area}} - \bar{x}_{\text{house area}} \qquad \tilde{x}_{\text{baths}} = x_{\text{baths}} - \bar{x}_{\text{baths}}$$

$$\tilde{x}_{\text{age}} = x_{\text{age}} - \bar{x}_{\text{age}} \qquad \tilde{x}_{\text{beds}} = x_{\text{beds}} - \bar{x}_{\text{beds}}$$

Model

$$\hat{y} = 13.69 + (4.07 \times 10^{-4})\tilde{x}_{\text{house area}} - 0.17x_{\text{townhouse}} - 0.25x_{\text{condo}} - 1.12x_{\text{mobile}} - 0.13x_{\text{multifamily}} + 0.05\tilde{x}_{\text{baths}}$$

$$-0.06\tilde{x}_{\text{beds}} + (3.31 \times 10^{-3})\tilde{x}_{\text{age}} + 0.06x_{\text{cooling}} - (1.97 \times 10^{-5})\tilde{x}_{\text{house area}}\tilde{x}_{\text{baths}} - (3.87 \times 10^{-3})x_{\text{townhouse}}\tilde{x}_{\text{age}}$$

$$-(5.99 \times 10^{-3})x_{\text{condo}}\tilde{x}_{\text{age}} - 0.01x_{\text{mobile}}\tilde{x}_{\text{age}} + (3.30 \times 10^{-3})x_{\text{multifamily}}\tilde{x}_{\text{age}}$$

$$+(6.24 \times 10^{-4})\tilde{x}_{\text{baths}}\tilde{x}_{\text{age}} - (6.28 \times 10^{-5})\tilde{x}_{\text{house area}}\tilde{x}_{\text{beds}} + 0.05\tilde{x}_{\text{beds}}\tilde{x}_{\text{baths}}$$

Actual Property Price Estimate

Property Price Estimate(USD) $= e^{\hat{y}}$

Table 16 shows a summary of the regression model and the $R^2_{adj}(0.82)$ value shows that the model has a good fit on the entire data set.

Addressing Assumptions

1. Linearity: From the preliminary fit, we can see in the pairwise scatter plots(Figure 10) and the correlation matrix(Table 10) that there are linear associations between the predictors and the response.

2. Independence: To our knowledge, a property's price does not affect a different property's price and sellers list the estates at random times. This suggests that there is no autocorrelation between the residuals(residuals do not grow over time).

3. Constant Variance: In figure 12, we can see the residuals plotted against the fitted values. There does not appear to be a severe departure from constant variance.

4. Normality: Figure 13 shows the Q-Q plot for the residuals. The plot suggests that residual distribution has more probability in the tails compared to the theoretical normal distribution. While the residuals are not exactly normal, this should not affect the estimated real estate prices. However, this phenomenon does affect confidence intervals and suggests that large outliers may be present in the data.

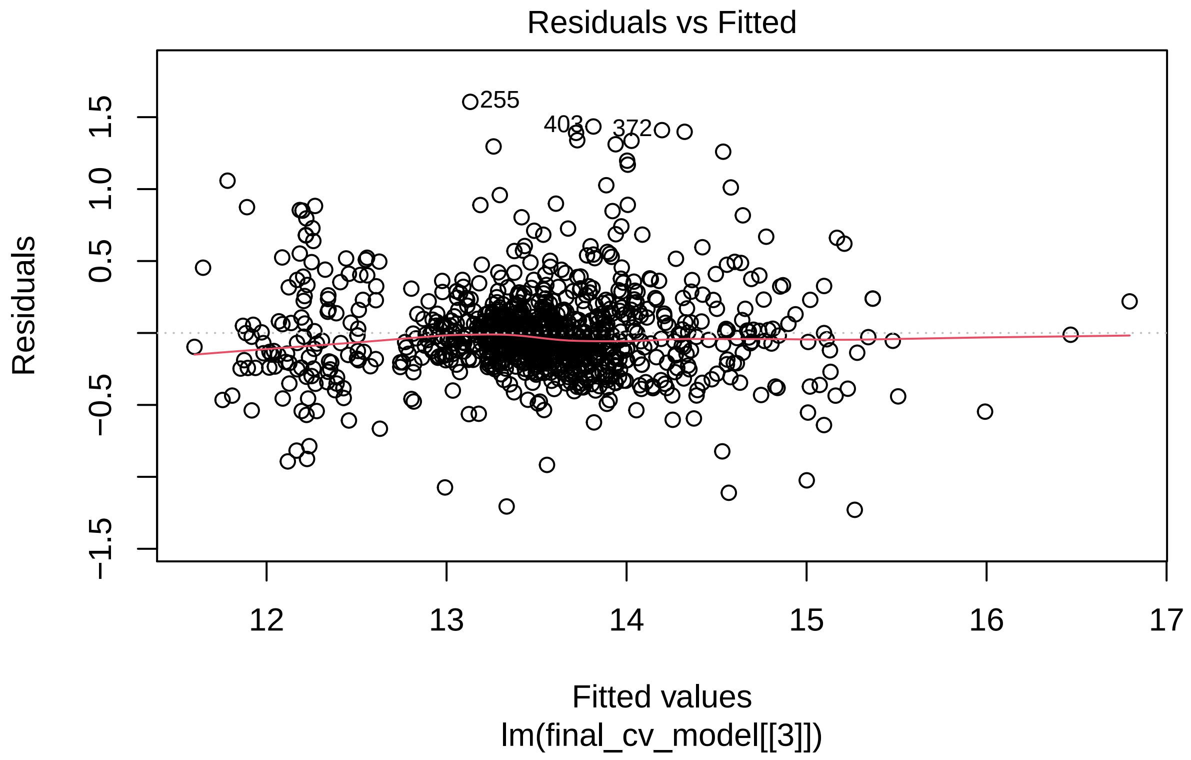


Figure 12: Residuals Vs Fitted Values plot. There does not appear to be any departure from constant variance. We can also see that the residuals fluctuate around zero.

Conclusion

In this project, we built a price predictive model using Ventura County real estate data(Zillow). The data contains around 1,000 listed homes for sale with the fields based on several house features, like beds, baths, and house area. Using visualizations, we saw how the real estate price behavior differed in Ventura County Cities. Then, we used the stepwise AIC method to produce the best performing predictive model on new data. This model consists of 18 parameters(including interactions) and generated the smallest cross-validation error compared to nine other models. The small CV error and the relatively small standard error(0.313) suggest that is model is more accurate than the other considered models. On the full data set, we also found that this model has a good fit and explained the variance in Y fairly well.

For a further investigation, it may be interesting to see if outliers are affecting this model. We could remove the outliers and re-run the model selection process to see if the method chooses a different model. It would be interesting to see how much the fit and cross-validation error improves. We may also want to investigate different model types, like regression splines and tree-based models.

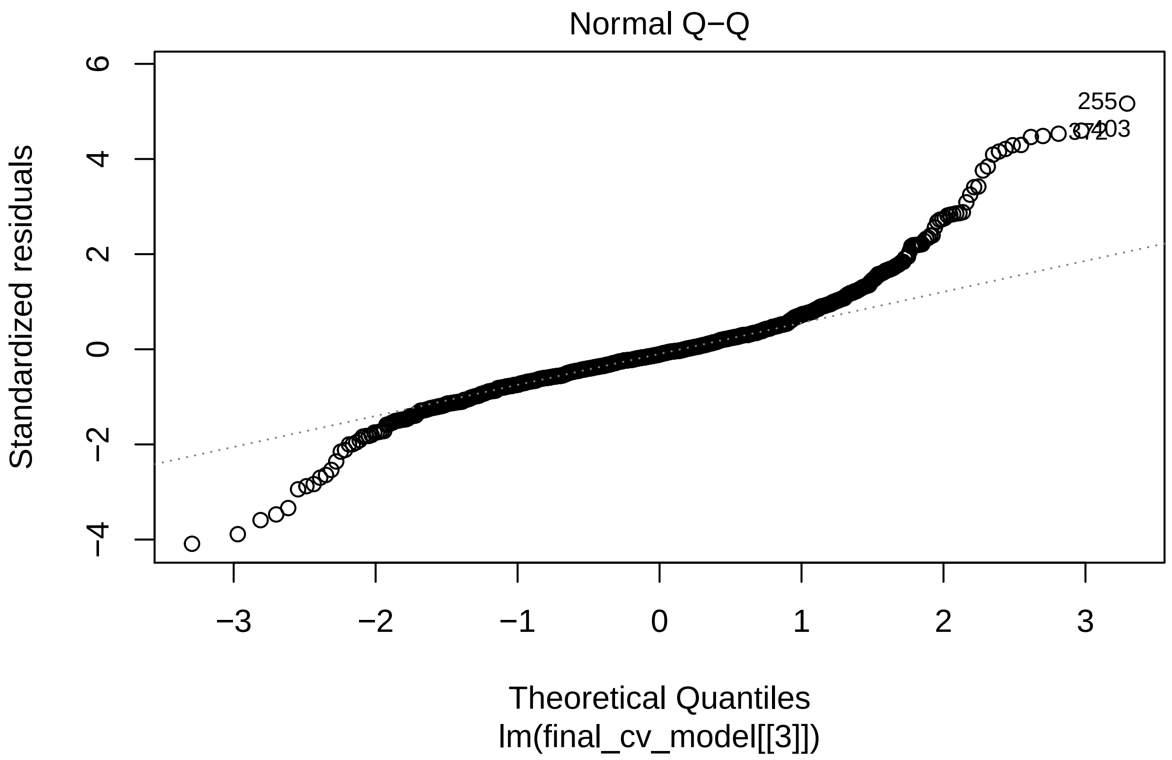


Figure 13: Normal Q-Q plot. There appears to be more probability in the error distribution's tails compared to the theoretical normal distribution. Since the linearity and constant variance assumptions are satisfied, this departure from normality may suggest large outliers.

Rmarkdown Code Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
# RMD File Libraries:
library(tidyverse)
library(pander)
library(knitr)
library(moderndive)
library(broom)
library(skimr)
library(ggplot2)
library(ggpubr)
library(gridExtra)
library(boot)
# Table 1 Numeric Variables:---------------------------------------------------
char_names = c( "Price", "House Area","Parking Spaces",
           "House Age", "Lot Area","Appliances Included", "City Population" )

variable_names = c("Price", "House_Area", "Total_spaces", "Age",
                   "Lot_Size", "Appliances", "population")

data_type = rep("Numeric", 7)
included = rep("Yes", 7)

units_levels = c( "USD($)", "Square Feet",  "Parking Spaces",  "Years",
                  "Square Feet", "Appliances", "People")
# Build the table:
table_1 = data.frame(char_names, variable_names,
                     data_type, included, units_levels)

colnames(table_1) = c("House Feature", "Variable Name", "Data Type",
                       "Included", "Units/Levels" )
kable(table_1, format = "latex",
      caption = "Numeric Variables Table")
# Table 2 Categorical and Logical Variables: -----------------------------------
char_names = c("Floor Cost", "Heater", "A/C",
          "Home Type", "Garage", "Stories",
          "Foundation Type", "Fireplace")

variable_names = c("Flooring", "Heating", "Cooling",
                   "Home_type", "Garage_type", "Levels",
                   "NA", "NA")

data_type = c("Categorical", rep("Logical",2),
              rep("Categorical",4), "Logical")

included = c(rep("Yes", 6), "No", "No")

units_levels = c( "Low, Medium, High", "TRUE, FALSE",
                  "TRUE, FALSE", "SingleFamily, Condo, Mobile",
                  "Three, Two, One, none",  "1, 2, 3+",
                  "Various Categories", "TRUE, FALSE")
# Build the table:
```

```r
table_2 = data.frame(char_names, variable_names,
                     data_type, included, units_levels)

colnames(table_2) = c( "House Feature", "Variable Name", "Data Type",
                       "Included", "Units/Levels" )

kable(table_2, format = "latex",
      caption = "Categorical and Logical Variables Table")


# Table for Beds and Baths: --------------------------------------------------
char_names = c("Bedrooms", "Bathrooms")
variable_names = c("Beds", "Baths")
data_type = rep("Numeric/Categorical", 2)
included = rep("Yes", 2)
units_levels = c("Rooms", "Rooms")

# Build the table:
table_1 = data.frame(char_names, variable_names,
                     data_type, included, units_levels)
colnames(table_1) = c( "House Feature", "Variable Name", "Data Type",
                       "Included", "Units/Levels" )
kable(table_1, format = "latex", caption = "Room Variables Table")

# Import the data set:--------------------------------------------------------
real_estate_data = read_csv2("final_zillow_datset.csv",
  col_types = list(col_factor(), col_character(), col_double(),col_double(),
                   col_double(),col_double(), col_factor(), col_logical(),
                   col_logical(), col_factor(), col_factor(), col_double(),
                   col_factor(), col_double(), col_double(), col_double(),
                   col_double()), col_names = TRUE)
table(real_estate_data$Price)
# Fix values and variables:
# Change MobileManufactured, Double and MobileManufactured, Triple to just
# MobileManufactured.
real_estate_data = real_estate_data %>%
          mutate(Home_Type = replace(Home_Type, Home_Type %in%
                c("MobileManufactured, Double", "MobileManufactured, Triple"),
                "MobileManufactured"))
real_estate_data$Home_Type = droplevels(real_estate_data$Home_Type)

# A few observations are missing trailing zeros
real_estate_data = real_estate_data %>%
    mutate(Price = replace(Price, Price < 20000, 1000*Price[Price < 20000]),
          Price = replace(Price, Price == 55900, 377595))

# Two oxnard houses have wrong square footage:
real_estate_data = real_estate_data %>%
    mutate(House_Area = replace(House_Area,
                           House_Area==1, mean(House_Area[City == "Oxnard"])))


# Produce a table for descriptive stats: ---------------------------------------
```

```r
skim_opt = skim_with(
  numeric = sfl(median = ~median(., na.rm = TRUE), min = ~min(., na.rm = TRUE),
                max = ~max(., na.rm = TRUE),
                complete_rate = NULL, p0 = NULL, p25 = NULL, p50 = NULL,
                p75 = NULL, p100 = NULL, hist = NULL),
  factor = sfl(complete_rate = NULL, ordered = NULL))
skim_opt(real_estate_data[,-2])
# Code to create dot plots for Ventura County Cities: -------------------------
# First, organize the data to get each city's mean, min, and
# max for certain variables.
city_data =
  real_estate_data %>%
    group_by(City) %>%
    summarise(house_counts = n(),
            mean_price = mean(Price, na.rm = TRUE),
            max_price = max(Price, na.rm = TRUE),
            min_price = min(Price, na.rm = TRUE),
            mean_house_area = mean(House_Area, na.rm = TRUE),
            mean_lot_size = mean(Lot_Size, na.rm = TRUE),
            mean_age = mean(Age, na.rm = TRUE),
            mean_parking = mean(Total_spaces, na.rm = TRUE),
            city_pop = unique(population))
house_stat_names = names(city_data)[c(3, 4, 5, 2, 6, 7, 8, 9, 10)]

# Colors for plots:
plot_colors = c("black", "blue", "darkolivegreen4",
                "red", "gray48", "dodgerblue4", "darkmagenta")
names(plot_colors) = c("Price", "of Properties For Sale",
                       "Property Area", "Property Lot Size",
                       "Property Age", "Parking Spaces", "of People")

# Function for x axis labels:
y_labels = function(axis_values){
  if(all(axis_values < 10000, na.rm = TRUE)){
    return(axis_values)
  }else if(all(axis_values > 2e+05, na.rm = TRUE) |
           max(axis_values, na.rm = TRUE) > 3.5e+07){
    nums = axis_values/1e+06
    num_labs = paste0(nums,"M")
    return(num_labs)
  }
  nums = axis_values/1000
  num_labs = paste0(nums,"K")
  return(num_labs)
}

# Function to build dot plots:
build_plot = function(house_feat, feat_name, stat_lab, y_lab = NULL){
  # We need two options in order to manipulate the axes if needed
  dot_color = plot_colors[house_feat]
if(house_feat == "Price"){
  data_plot = city_data %>%
    mutate(City = factor(City,
              levels = sort(City, decreasing = TRUE))) %>%
              select(City, plot_values = feat_name)
        ggplot(data_plot, mapping = aes(City, plot_values)) +
            scale_y_continuous(labels = y_labels) +
            geom_point(colour = dot_color, size = 3) +
            coord_flip() +
            xlab("Ventura County Cities") +
            ylab("Price(USD)") +
            theme_minimal()+
            theme(axis.text.x= element_text(size=8),
              axis.text.y=element_text(size=8))+
              labs(title = paste(stat_lab,
                                   "Property Prices"))
    }else if(house_feat == "Property Lot Size"){
      data_plot = city_data %>%
       mutate(City = factor(City,
              levels = sort(City, decreasing = TRUE))) %>%
              select(City, plot_values = feat_name)
        ggplot(data_plot, mapping = aes(City, plot_values)) +
            geom_point(colour = dot_color, size = 3) +
            scale_y_log10(labels = scales::comma) +
            coord_flip() +
            xlab("Ventura County Cities") +
            ylab("Log Area(Square Feet)") +
            theme_minimal()+
            theme(axis.text.x= element_text(size=8),
            axis.text.y=element_text(size=7))+
            labs(title = paste(stat_lab, house_feat))
    }else{
     data_plot = city_data %>%
       mutate(City = factor(City,
              levels = sort(City, decreasing = TRUE))) %>%
              select(City, plot_values = feat_name)
        ggplot(data_plot, mapping = aes(City, plot_values)) +
            geom_point(colour = dot_color, size = 3) +
            scale_y_continuous(labels = y_labels) +
            coord_flip() +
            xlab("Ventura County Cities") +
            ylab(y_lab) +
            theme_minimal()+
            theme(axis.text.x= element_text(size=8),
              axis.text.y=element_text(size=7))+
              labs(title = paste(stat_lab, house_feat))
  }
}

# Call function and create plots
options(scipen = 5) # Expands scientific notation

mean_price_p = build_plot("Price", house_stat_names[1], "Mean") # Good
max_price_p = build_plot("Price", house_stat_names[2], "Maximum") # Good
min_price_p = build_plot("Price", house_stat_names[3], "Minimum") # Good
num_prop_p = build_plot("of Properties For Sale",
                         house_stat_names[4], "Number", "Properties")
mean_area_p = build_plot("Property Area",
                         house_stat_names[5], "Mean", "Area(Square Feet)")
mean_lotsize_p = build_plot("Property Lot Size",
                         house_stat_names[6], "Mean")
mean_age_p = build_plot("Property Age",
                         house_stat_names[7], "Mean", "Age(Years)")
mean_parking_p = build_plot("Parking Spaces",
                         house_stat_names[8], "Mean", "Spots")
population_p = build_plot("of People",
                         house_stat_names[9], "Number", "People")
# Align plots:
layout_matrix <- matrix(c(1, 1, 2, 2, 4, 3, 3, 4), nrow = 2, byrow = TRUE)
# Plots for min, max, and mean price:------------------------------------------
grid.arrange(min_price_p, max_price_p,
             mean_price_p, layout_matrix = layout_matrix)
# Plots for number of properties, mean property area, lot size, and Age:-------
grid.arrange(num_prop_p, mean_area_p,
             mean_lotsize_p, mean_age_p)
# Plots for mean parking spaces and population:--------------------------------
grid.arrange(mean_parking_p, population_p,
             heights=c(2,1), ncol = 2)
# Code to create Box plots for Categorical Variables:---------------------------
#First, we mutate bathrooms and bedrooms variables to be factors
box_plot_data = real_estate_data %>%
                mutate(Beds = as.character(Beds),
                          Baths = as.character(Baths)) %>%
                mutate(Beds = replace(Beds,
                                    Beds %in% as.character(6:10), "6+"),
                       Baths = replace(Baths,
                                    Baths %in%  as.character(6:12), "6+")) %>%
                mutate(Beds = as.factor(Beds),
                          Baths = as.factor(Baths))

# Revel floor cost, garage type
box_plot_data$Flooring = factor(box_plot_data$Flooring,
                         levels = levels(box_plot_data$Flooring)[c(2,4,3,1)])

box_plot_data$Garage_Type = factor(box_plot_data$Garage_Type,
                  levels = levels(box_plot_data$Garage_Type)[c(4, 5, 2, 1, 3)])


# Function for x axis labels:
y_labels = function(axis_values){
    nums = axis_values/1e+06
    num_labs = paste0(nums,"M")
    return(num_labs)
}

# Function to build the boxplots:
y_lim = 6e+06
get_box_plot = function(house_feat_cats, x_lab){
  if(x_lab == "Air Conditioning Presence" | x_lab == "Heater Presence"){
    box_plot_data %>%
      ggplot(aes(x = house_feat_cats, y=Price, fill=house_feat_cats)) +
               scale_y_continuous(labels = y_labels,
                                   breaks = round(seq(0,
                                                      y_lim, by = 1.5e+06),1),
                                  limits = c(NA, y_lim + 500))+
                scale_x_discrete(labels = c("Not Present", "Present")) +
                scale_fill_discrete(name = x_lab,
                                    labels = c("Not Present", "Present")) +
                geom_boxplot(alpha=0.4) +
                xlab(x_lab) +
                ylab("Price(Millions USD")+
                labs(title = paste("Property Price By", x_lab))

  }else{
  box_plot_data %>%
  ggplot(aes(x = house_feat_cats, y=Price, fill=house_feat_cats)) +
               scale_y_continuous(labels = y_labels,
                                   breaks = round(seq(0,
                                                      y_lim, by = 1.5e+06),1),
                                  limits = c(NA, y_lim + 500))+
                geom_boxplot(alpha=0.4) +
                xlab(x_lab) +
                ylab("Price(Millions USD")+
                labs(title = paste("Poperty Price By", x_lab), fill = x_lab)
  }
}

# Plot objects:

floor_plot = get_box_plot(box_plot_data$Flooring,
             "Floor Cost")


home_type_plot = get_box_plot(box_plot_data$Home_Type,
             "Home Types")

garage_type_plot = get_box_plot(box_plot_data$Garage_Type,
             "Garage Types")

levels_plot =get_box_plot(box_plot_data$Levels,
             "Property Stories")

AC_plot = get_box_plot(box_plot_data$Cooling,
             "Air Conditioning Presence")

heating_plot = get_box_plot(box_plot_data$Heating,
             "Heater Presence")

beds_plot = get_box_plot(box_plot_data$Beds,
             "Number of Beds")

baths_plot = get_box_plot(box_plot_data$Baths,
             "Number of Baths")
# Home and garage type plots:----------------------------------------------------
home_type_plot =
  home_type_plot + scale_x_discrete(
    labels = c("SingleFamily", "Townhouse", "Condo",
               "Mobile", "Multifamily", "Other",  "NA"))
suppressWarnings(grid.arrange(home_type_plot, garage_type_plot))
# Floor and levels plots:--------------------------------------------------------
suppressWarnings(grid.arrange(floor_plot, levels_plot))
# AC and heating plots:----------------------------------------------------------
suppressWarnings(grid.arrange(AC_plot, heating_plot))
# Beds and Baths plots:----------------------------------------------------------
suppressWarnings(grid.arrange(beds_plot, baths_plot))
# Before proceeding, we look at the Price behavior:----------------------------
y_plot_1 = real_estate_data %>%
        ggplot(mapping = aes(x = Price)) +
        geom_histogram(aes(y=..density..),
                       colour="black", fill="white") +
        geom_density(alpha=.2, fill="#FF6666") +
        scale_x_continuous(labels = y_labels) +
        ylab("Density") +
        xlab("Prices") +
        theme_minimal()+
        theme(axis.text.x= element_text(size=9),
              axis.text.y=element_text(size=9)) +
        labs(title = "Real Estate Price Histogram")

y_plot_2 = real_estate_data %>%
           ggplot(mapping = aes(x = log(Price))) +
           geom_histogram(aes(y=..density..),
                          colour="black", fill="white") +
           geom_density(alpha=.2, fill="#FF6666") +
           ylab("Density") +
           xlab("Log Prices") +
           theme_minimal() +
           theme(axis.text.x= element_text(size=9),
           axis.text.y=element_text(size=9)) +
           labs(title  = "Log Real Estate Price Histogram")
# Price distribution looks better with log transformation

# Price plots:-------------------------------------------------------------------
suppressMessages(grid.arrange(y_plot_1, y_plot_2, nrow = 2))
# Create the sets that allow for the beds and baths interpretations:-----------
# Remove the cities variable:
vc_data_num = real_estate_data[, -c(1,2)] #Remove city and address
vc_data_num = vc_data_num %>%
                mutate(Price = log(Price))
vc_data_cat = box_plot_data[,-c(1,2)]
vc_data_cat = vc_data_cat %>%
                mutate(Price = log(Price))
# Look at NA numbers:
na_check = sapply(vc_data_num, is.na)
num_nas = apply(na_check, 2, sum)
```

```r
num_nas = as.matrix(num_nas)
colnames(num_nas) = "Missing Values"
kable(num_nas, format = "latex",
      caption = "Missing Values")
# Missing Values: Levels - 181, Appliances - 221, Lot_Size - 140
# May need to remove them

#Investigate Correlations:-----------------------------------------------------
cor_table_1 = round(cor(vc_data_num[, -c(2:4, 5:11, 12, 15)],
                        use = "pairwise.complete.obs"),5)
kable(cor_table_1, format = "latex",
      caption = "Pairwise Complete Variable Correlations")
pairs(vc_data_num[, -c(2:4, 5:11, 12, 15)])
cor_table_2 = round(cor(vc_data_num[, -c(5:9, 11, 13:14)],
                        use = "complete.obs"), 5)
kable(cor_table_2, format = "latex",
      caption = "Complete Variable Correlations")
pairs(vc_data_num[, -c(5:9, 11, 13:14)])
# Remove the variables with too many NAs:---------------------------------------
vc_data_num = vc_data_num[,-c(11, 13:14)] %>%
              filter(!is.na(House_Area) & !is.na(Age))

vc_data_cat = vc_data_cat[, -c(11, 13:14)] %>%
              filter(!is.na(House_Area) & !is.na(Age))

# Get random subsets:----------------------------------------------------------
set.seed(2)

test_subset_num = vc_data_num[sample(c(TRUE,FALSE),
                                     dim(vc_data_num)[1], replace = TRUE),]

test_subset_cat = vc_data_cat[sample(c(TRUE,FALSE),
                                     dim(vc_data_num)[1], replace = TRUE),]

# Get fits: --------------------------------------------------------------------
pre_fit_num = lm(Price ~ ., data = test_subset_num)
pre_fit_cat = lm(Price ~ ., data = test_subset_cat)
sum_var_num = summary(pre_fit_num)
sum_var_cat = summary(pre_fit_cat)
model_1_f = c(sum_var_num$fstatistic[1], "< 2.2e-16")
model_2_f = c(sum_var_cat$fstatistic[1], "< 2.2e-16")
models_table = cbind(model_1_f, model_2_f)
colnames(models_table) = c("Model 1", "Model 2")
rownames(models_table) = c("F-statistic", "P-value")
kable(models_table, format = "latex", caption = "Full Models' F-Statistics")

# Check the vif: ---------------------------------------------------------------
vif_table_1 = car::vif(pre_fit_num)
vif_table_2  = car::vif(pre_fit_cat)
vif_table_full = cbind(vif_table_1[, 1], vif_table_2[, 1])
colnames(vif_table_full) = c("Model 1", "Model 2")
kable(vif_table_full, format = "latex",
      caption = "VIFs for full models")
```

```r
#Function to get residual plots : ----------------------------------------------
get_residual_plot = function(house_feat_name, var_name){
   test_subset_num  %>%
    select(x_values = var_name) %>%
              ggplot(mapping =
                       aes(x = x_values, y = pre_fit_num$residuals)) +
            geom_point() +
            xlab(house_feat_name) +
            ylab("Residuals") +
            theme_minimal()+
            theme(axis.text.x= element_text(size=9),
            axis.text.y=element_text(size=9))+
            labs(title = paste0("Residuals Vs ", house_feat_name))
}
# residual plats for numeric variables
res_plot_1 = get_residual_plot("Area", "House_Area")
res_plot_2 = get_residual_plot("Age", "Age")
res_plot_3 = get_residual_plot("Spaces", "Total_spaces")
res_plot_4 = get_residual_plot("Baths", "Baths")
res_plot_5 = get_residual_plot("Beds", "Beds")
res_plot_6 = get_residual_plot("Popul.", "population")

grid.arrange(res_plot_1, res_plot_2, res_plot_3,
             res_plot_4, res_plot_5, res_plot_6, nrow = 2, ncol = 3)

# Functions for selection and cross-validation:---------------------------------
# Best subsets method
# alllow for forward and backward
best_subsets = function(h_data, k_folds, p_vars,
                method_name, interaction_allowed){
  set.seed(1)
  folds = sample(1:k_folds, nrow(h_data), replace = TRUE)
  cv_errors = matrix(NA, k_folds, p_vars,
                     dimnames = list(NULL, paste(1:p_vars)))
  for(j in 1:k_folds){ # regsubsets allows for interaction
    if(interaction_allowed == "yes"){
    best_fit =
      invisible(
      suppressWarnings(
      leaps::regsubsets(Price ~ .^2, data = h_data[folds != j,],
                        nvmax = p_vars, method = method_name,
                        really.big=TRUE)))
    }else{
       best_fit = suppressPackageStartupMessages(
         invisible(
           suppressWarnings(
         leaps::regsubsets(Price ~ ., data = h_data[folds != j,],
                                 nvmax = p_vars, method = method_name))))
    }
    for(i in 1:p_vars){
      pred_value = predict_regfit(best_fit, h_data[folds == j,], id = i)
      cv_errors[j,i] = mean((h_data$Price[folds ==j] -  pred_value)^2)
    }
  }
  return(cv_errors)
}

# To get the prediction:
predict_regfit = function(reg_object, new_data, id){
  obj_form = as.formula(Price ~ .)
  mat_vals = model.matrix(obj_form, new_data)
  coefi = coef(reg_object, id = id)
  x_vars = names(coefi)
  return(mat_vals[,x_vars]%*%coefi)
}

#AIC method for prediction: ---------------------------------------------------
best_aic_model = function(h_data, k_folds, starting_point,
              interaction_allowed, poly_present = NULL, include_3_way = NULL){
  #Forward and Backward Methods
  set.seed(1)
  folds = sample(1:k_folds, nrow(h_data), replace = TRUE)
  models_errors = list()
  if(starting_point == "forward"){ # Check if we start with no variables
  for(j in 1:k_folds){
    print(j)
      if(interaction_allowed == "yes" & poly_present == "yes"){
      best_fit = step(glm(Price ~ 1, data = h_data[folds != j,]),
                     scope =~Beds*Baths*House_Area*
                     House_Area_2*Flooring*Heating*Cooling*
                     Home_Type*Garage_Type*Total_spaces*
                     Age*population,
                     direction = "both")
      }else if(interaction_allowed == "no" & poly_present == "yes"){
       best_fit = suppressWarnings(
                     step(glm(Price ~ 1, data = h_data[folds != j,]),
                     scope =~Beds+Baths+House_Area+
                     Flooring+Heating+Cooling+
                     Home_Type+Garage_Type+Total_spaces+
                     Age+population+House_Area_2+
                     Total_spaces_2+ Age_2 + population_2,
                     direction = "both"))
      }else if(interaction_allowed == "yes" & include_3_way == "yes"){
        best_fit = step(glm(Price ~ 1, data = h_data[folds != j,]),
                     scope =~Beds*Baths*
                    House_Area*
                     Flooring*Heating*Cooling*
                     Home_Type*Garage_Type*Total_spaces*
                     Age*population,
                     direction = "both")
      }else if(interaction_allowed == "yes"){
        best_fit = suppressWarnings(
                     step(glm(Price ~ 1, data = h_data[folds != j,]),
                     scope =~(Beds+Baths+
                     House_Area+
                     Flooring+Heating+Cooling+
                     Home_Type+Garage_Type+Total_spaces+
                     Age+population)^2,
                     direction = "both"))
      }else{
          best_fit = suppressWarnings(
                    step(glm(Price ~ 1, data = h_data[folds != j,]),
                    scope = ~Beds+Baths+House_Area+
                    Flooring+Heating+Cooling+
                    Home_Type+Garage_Type+Total_spaces+
                    Age+population,
                    direction = "both"))
      }
      pred_value = predict(best_fit, h_data[folds == j,], id = i)
      cv_error = mean((h_data$Price[folds ==j] -  pred_value)^2)
      models_errors[[j]] = list(cv_error, best_fit$rank,
                                         best_fit$formula,
                                         best_fit$coefficients,
                                         best_fit$converged)
                }
        }
  return(models_errors)
}

# Numerical beds and baths:----------------------------------------------------
# MODEL 1********
full_model_num = glm(Price ~ ., data = vc_data_num)
cv_error_num = cv.glm(vc_data_num, full_model_num, K = 10)
cv_error_1 = cv_error_num$delta[1]
num_parameters_1 = length(full_model_num$coefficients)

# Categorical beds and baths:----------------------------------------------------
# MODEL 2*******
full_model_cat = glm(Price ~ ., data = vc_data_cat)
cv_error_cat = cv.glm(vc_data_cat, full_model_cat, K = 10)
cv_error_2 = cv_error_cat$delta[1]
num_parameters_2 = length(full_model_cat$coefficients)

# Build table:
models_errors = matrix(c(num_parameters_1, round(cv_error_1, 4),
                        "Less than 5", num_parameters_2, round(cv_error_2, 4),
                        "More than 5"), nrow = 3, ncol = 2)
colnames(models_errors) = c("Model 1", "Model 2")
rownames(models_errors) = c("Number of Parameters", "CV Error", "Largest VIF")
kable(models_errors, format = "latex",
      caption = "Full Models 1 and 2")

#Best subsets results:---------------------------------------------------------
# Numeric beds and baths:
# MODEL 3***************
best_cv_results1 = best_subsets(vc_data_num, 10, 19,
                "exhaustive", "no")
mean_cv_errors = apply(best_cv_results1, 2, mean)
smallest_index = which.min(mean_cv_errors)
```

```
#The model with 10 variables has the smallest CV
best_table_1 = c(smallest_index,
                 round(mean_cv_errors[smallest_index], 4), "Less than 5")

# Categorical beds and baths:---------------------------------------------------
# MODEL 4***************
best_cv_results2 = best_subsets(vc_data_cat, 10, 26,
                "exhaustive", "no")
mean_cv_errors2 = apply(best_cv_results2, 2, mean)
smallest_index = which.min(mean_cv_errors2)
best_table_2 = c(smallest_index,
                 round(mean_cv_errors2[smallest_index], 4), "More than 5")
best_full_table = cbind(best_table_1, best_table_2)
rownames(best_full_table) = c("Number of Variables", "CV error", "Largest VIF")
colnames(best_full_table) = c("Model 3",
                              "Model 4")

kable(best_full_table, format = "latex",
      caption = "Best Subsets Model 3 and 4")
# functions for AIC:
# Funtion for necessary info in list
get_cv_rank_coef = function(model_list){
  aic_cv_errors = lapply(model_list, function(x) x[[1]]) %>%
                          unlist()
  num_coefs = lapply(model_list, function(x) length(x[[4]])) %>%
                          unlist()
  ranks = lapply(model_list, function(x) x[[2]]) %>%
                          unlist()
  return(list(aic_cv_errors,  num_coefs, ranks ))
}

# function to identify similar models:
find_similar = function(model_list, best_model){#returns true and false
  same_model =  lapply(model_list,
                 function(x) all(names(x[[4]]) %in% names(best_model))) %>%
                          unlist()
  return(same_model)
}

#function for cross-validation:
get_cv = function(reg_model, cv_data){
  lm_model = glm(reg_model, data = cv_data)
  cv_error_sum = cv.glm(cv_data,  lm_model, K = 10)
  cv_error_final = cv_error_sum$delta[1]
  return(cv_error_final)
}


# Corrected data sets to lower multicollinearity: -----------------------------
data_num_corrected = vc_data_num %>%
                mutate(Beds = Beds - mean(Beds),
                        Baths = Baths - mean(Baths),
                        House_Area = House_Area - mean(House_Area),
```

```r
                        Total_spaces = Total_spaces - mean(Total_spaces),
                        Age = Age - mean(Age),
                        population = population - mean(population))

data_cat_corrected = vc_data_cat %>%
                mutate(House_Area = House_Area - mean(House_Area),
                        Total_spaces = Total_spaces - mean(Total_spaces),
                        Age = Age - mean(Age),
                        population = population - mean(population))

#vc_data_num = vc_data_num %>%
          # mutate(Beds = (Beds - mean(Beds)) / sd(Beds),
          #  Baths = (Baths - mean(Baths))/ sd(Baths),
          # House_Area = (House_Area - mean(House_Area))/sd(House_Area),
          # Total_spaces = (Total_spaces - mean(Total_spaces))/sd(Total_spaces),
          # Age = (Age - mean(Age))/sd(Age),
          # population = (population - mean(population))/sd(population))

# Stepwise AIC:---------------------------------------------------------------
# Two-way interactions:
# Numeric beds and baths:
# MODEL 5*******
best_aic_models = best_aic_model(data_num_corrected, 10, "forward",
                                 "yes", "no", "no")
# Relevant values:
values_1 = get_cv_rank_coef(best_aic_models)
aic_cv_errors = values_1[[1]]; num_coefs =  values_1[[2]]; ranks = values_1[[3]]

# Cross-validate the acquired models and
# find which one produces smallest cv error:
valid_models = ranks==num_coefs
cross_validat_vals =  lapply(best_aic_models,
                         function(x) get_cv(x[[3]], data_num_corrected)) %>%
                        unlist()
min_valid_cv = min(cross_validat_vals[valid_models])
index = which(cross_validat_vals == min_valid_cv)[1]
final_cv_model = best_aic_models[[index]]

# Test VIF
test_model1= lm(final_cv_model[[3]], data = data_num_corrected )
car::vif(test_model1) #Multicollinearity present

# Construct past of table:
aic_table_num = c(num_coefs[index],
                  round(min_valid_cv, 4), "Yes", "More than 10")

#Remove variables to be cautious
rm(best_aic_models, cross_validat_vals,
   index, ranks, num_coefs, aic_cv_errors, valid_models)

# Categorical beds and baths:-------------------------------------------------
# Model 6******
best_aic_models2 = best_aic_model(data_cat_corrected, 10, "forward",
                                 "yes", "no", "no")
# Relevant Values:
values_2 = get_cv_rank_coef(best_aic_models2)
aic_cv_errors2 = values_2[[1]]; num_coefs2 = values_2[[2]]
ranks2 = values_2[[3]]

# Cross-validate the acquired models and
# find which one produces smallest cv error:
valid_models = ranks2==num_coefs2
cross_validat_vals2 =  lapply(best_aic_models2,
                        function(x) get_cv(x[[3]], data_cat_corrected)) %>%
                          unlist()
min_valid_cv2 = min(cross_validat_vals2[valid_models])
index2 = which(cross_validat_vals2 == min_valid_cv2)[1]
correct_cv_model = best_aic_models2[[index2]]

# Test VIF
test_model2 = lm(correct_cv_model[[3]], data = data_cat_corrected)
car::vif(test_model2) # Too high, Multicollinearity

aic_table_cat = c(num_coefs2[index2],
                  round(min_valid_cv2, 4), "Yes", "More than 10")

# Remove variables to be cautious:
rm(best_aic_models2, cross_validat_vals2,
   index2, valid_models, correct_cv_model, min_valid_cv2)

#No interaction: ----------------------------------------------------------------
#Numeric:
# MODEL 7******
best_aic_models3 = best_aic_model(vc_data_num, 10, "forward",
                                 "no", "no", "no")

values_3 = get_cv_rank_coef(best_aic_models3)
aic_cv_errors = values_3[[1]]; num_coefs = values_3[[2]];
ranks = values_3[[3]]

# Cross-validate the acquired models and
# find which one produces smallest cv error:
valid_models = ranks==num_coefs
cross_validat_vals =  lapply(best_aic_models3,
                      function(x) get_cv(x[[3]], vc_data_num)) %>%
                                  unlist()
min_valid_cv = min(cross_validat_vals[valid_models])
index = which(cross_validat_vals == min_valid_cv)[1]
correct_cv_model = best_aic_models3[[index]]

# Test VIF
test_model3 = lm(correct_cv_model[[3]], data = vc_data_num)
car::vif(test_model3) # No mulitcollinearity present

aic_noint_num = c(num_coefs[index],
                  round(min_valid_cv,4), "No", "Less than 5")
```

```r
# Remove variables to be cautious:
rm(best_aic_models3, cross_validat_vals,
   index, ranks, num_coefs, aic_cv_errors, valid_models)

#No interaction: ---------------------------------------------------------------
# Categorical:
# MODEL 8******
best_aic_models4 = best_aic_model(vc_data_cat, 10, "forward",
                                  "no", "no", "no")
values_4 = get_cv_rank_coef(best_aic_models4)
aic_cv_errors = values_4[[1]]; num_coefs = values_4[[2]];
ranks = values_4[[3]]

# Cross-validate the acquired models and
# find which one produces smallest cv error:
valid_models = ranks==num_coefs
cross_validat_vals =  lapply(best_aic_models4,
                      function(x) get_cv(x[[3]], vc_data_cat)) %>%
                                  unlist()
min_valid_cv = min(cross_validat_vals[valid_models])
index = which(cross_validat_vals == min_valid_cv)[1]
correct_cv_model = best_aic_models4[[index]]

# Test VIF
test_model4 = lm(correct_cv_model[[3]], data = vc_data_cat)
car::vif(test_model4) # No mulitcollinearity present

aic_noint_cat = c(num_coefs[index],
                  round(min_valid_cv, 4), "No", "More than 5")

# Remove variables to be cautious:
rm(best_aic_models4, cross_validat_vals,
   index, ranks, num_coefs, aic_cv_errors, valid_models)

#TEST QUADRATIC:----------------------------------------------------------------
data_num_corrected = data_num_corrected %>%
   mutate(Beds_2 = Beds^2,
                         Baths_2 = Baths^2,
                         House_Area_2 = House_Area^2,
                         Total_spaces_2 = Total_spaces^2,
                         Age_2 = Age^2,
                         population_2 = population^2)
# MODEL 9******
best_aic_models5 = best_aic_model(data_num_corrected, 10, "forward",
                                  "no", "yes", "no")

values_5 = get_cv_rank_coef(best_aic_models5)
aic_cv_errors = values_5[[1]]; num_coefs = values_5[[2]];
ranks = values_5[[3]]

# Cross-validate the acquired models and
# find which one produces smallest cv error:
valid_models = ranks==num_coefs
```

```
cross_validat_vals =  lapply(best_aic_models5,
                      function(x) get_cv(x[[3]], data_num_corrected)) %>%
                            unlist()
min_valid_cv = min(cross_validat_vals[valid_models])
index = which(cross_validat_vals == min_valid_cv)[1]
correct_cv_model = best_aic_models5[[index]]

# Check the VIF:
test_model5 = lm(correct_cv_model[[3]], data =  data_num_corrected)
car::vif(test_model5) #Multicollinearity present

aic_quad_num = c(num_coefs[index],
                  round(min_valid_cv, 4), "No", "More than 5")

# Remove variables to be cautious:
rm(best_aic_models5, cross_validat_vals,
   index, ranks, num_coefs, aic_cv_errors, valid_models)

#TEST QUADRATIC:----------------------------------------------------------------
data_cat_corrected = data_cat_corrected %>%
   mutate(House_Area_2 = House_Area^2,
          Total_spaces_2 = Total_spaces^2,
          Age_2 = Age^2,
          population_2 = population^2)

# MODEL 10******
best_aic_models6 = best_aic_model(data_cat_corrected, 10, "forward",
                                 "no", "yes", "no")

values_6 = get_cv_rank_coef(best_aic_models6)
aic_cv_errors = values_6[[1]]; num_coefs = values_6[[2]];
ranks = values_6[[3]]

# Want to cross validate the acquired models and
# find which one produces smallest cv error:
valid_models = ranks==num_coefs
cross_validat_vals =  lapply(best_aic_models6,
                      function(x) get_cv(x[[3]], data_cat_corrected)) %>%
                            unlist()
min_valid_cv = min(cross_validat_vals[valid_models])
index = which(cross_validat_vals == min_valid_cv)[1]
correct_cv_model = best_aic_models6[[index]]

# Check the VIF:
test_model6 = lm(correct_cv_model[[3]], data = data_cat_corrected)
car::vif(test_model6) #No indication of multicollinearity

aic_quad_cat = c(num_coefs[index],
                  round(min_valid_cv, 4), "No", "More than 5")
# build the table
final_table_data = cbind(aic_table_num, aic_table_cat, aic_noint_num,
                         aic_noint_cat, aic_quad_num, aic_quad_cat)
colnames(final_table_data) = c("Model 5", "Model 6", "Model 7",
                               "Model 8", "Model 9", "Model 10")
rownames(final_table_data) = c("Number of Parameters", "CV error",
                             "Interaction", "Largest VIF")
kable(final_table_data, format = "latex",
      caption = "Stepwise AIC Models 5-10")
# Fit the final
final_model =  lm(final_cv_model[[3]], data = data_num_corrected)
reg_results = round(glance(final_model)[c(1:6,7:8)],3)
kable(reg_results, format = "latex",
      caption = "Selected Predictive Model Statistics")
plot(final_model, which = 1)
plot(final_model, which = 2)
```

Appendix A: Scraping Code 1

```
#---------------------------------------------------------------
#-------------SCRAPING FUNCTIONS FOR ZILLOW
#---------------------------------------------------------------
# Libraries:
library(tidyverse)
library(rvest)
library(stringr)
#Here we scrape some important feature and the links to the houses
#We write to a csv file to save the information.
#The links will be used in the data cleaning file
# Function to get the total number of posts on the current page:


get_post_total = function(page_html){
  str_amount = page_html %>%
    html_node(".total-text") %>%
    html_text()
  results_amount = as.integer(gsub("[, ]+","", str_amount))
  return(results_amount)
}

# Function to extract home specific info:
# Add site cookie***
get_home_info = function(num_pages, search_url, area_name){


  links = sprintf(paste0(search_url,"%d_p"), 1:num_pages)

  results = map(links, ~ {
    Sys.sleep(30)
    # select houses
    houses = read_html(.x, cookie = ".") %>%
      #Nodes for all listings, photo-cards refers to section attribute,
      #li is the list node, article refers to each listing
      html_nodes(".photo-cards li article")
    z_id = houses %>%
      html_attr("id")

    #Links for houses
    house_links = houses %>%
      html_node(".list-card-info a") %>%
      html_attr("href")

    # address
    address = houses %>%
      html_node(".list-card-addr") %>%
      html_text()

    # price
    price = houses %>%
      html_node(".list-card-price") %>%
      html_text() %>%
      readr::parse_number()

    # info
    params = houses %>%
      html_node(".list-card-info") %>%
      html_text2()

    # number of bedrooms
    beds = params %>%
      str_extract("\\d+(?=\\s*bds)") %>%
      as.numeric()

    # number of bathrooms
    baths = params %>%
      str_extract("\\d+(?=\\s*ba)") %>%
      as.numeric()

    # total square footage
    house_a = params %>%
      str_extract("[0-9,]+(?=\\s*sqft)") %>%
      str_replace(",", "") %>%
      as.numeric()

    tibble(location_name = area_name, address = address,
           price = price,  beds= beds, baths=baths,
           house_area = house_a, house_links = house_links)

  }
  ) %>%
    bind_rows(.id = 'page_no')
  return(results)
}

write_to_csv = function(home_data, area_name){
  write_csv2(home_data,
             paste0("/Users/joseph_gonzalez/Desktop/STA 220/",
                    "FINAL Project/Zillow datasets ventura county/",
```

```r
            area_name, "-home_data.csv"), col_names = TRUE)
}

# Function to get listings:
# Add site cookie***
get_listings = function(area_name, area_url){
  Sys.sleep(120) #Change this for faster scrape
  first_page_URL = paste0(zillow_url, area_url)
  first_page_html = read_html(first_page_URL, cookie = ".")
  post_total = get_post_total(first_page_html)
  if(post_total>=240){
    num_pages = 6
  }else if(post_total > 0 & post_total < 240){
    num_pages = ceiling(post_total/40)
  }else{
    results = tibble(location_name = area_name, price = NA,
                     beds = NA, baths= NA, house_area = NA,
                     house_links = NA)
    write_to_csv(results, area_name)
    return(results)
  }
  home_data = get_home_info(num_pages, first_page_URL, area_name)
  write_to_csv(home_data, area_name)
  return(home_data)
}

#----------------------------------------------------------------------------------------------------
#-------------END OF MAIN FUNCTIONS
#----------------------------------------------------------------------------------------------------
#Zillow URL
zillow_url = "https://www.zillow.com/"

### EXPLANATION FOR FOLLOWING CODE TO SCRAPE:
# At first, the full code intended to scrape for real estate in all California
# counties. Due to limitations with zillow, I decided to only scrape for real
# estate data in cities with Ventura County(My current county). The code can be
#altered to do all california counties.

# County Names
#county_data = read.csv("California_Counties.csv", header = TRUE)
#county_half_url = gsub(" ", "-", county_data[,"Name"])
#county_half_url = paste0(county_half_url,"-county-CA/")

#----------------------------------------------------------------------------------------------------
# Import the city data:
#City Names:
ventura_county_data = read.csv("Ventura_Cities_Data.csv",
                               header = TRUE, sep= ";")
area_names = ventura_county_data$city_names
areas_half_url =ventura_county_data$zillow_part_urls
house_data = mapply(get_listings, area_names, areas_half_url)
```

Appendix B: Scraping Code 2

```r
#-------------------------------------------------------------------------------
### Scraping & Cleaning Zillow data:
#-------------------------------------------------------------------------------
# This files scraps individual home data and cleans some features
library(tidyverse)
library(rvest)
library(stringr)
library(lubridate)

# ****Working Directory should be the file with all data files*****
data_files = list.files(getwd()) #File with the zilow datasets only
real_estate_data = lapply(data_files,
                          function(x) read.csv(x, header = TRUE,sep = ";"))
real_estate_data = bind_rows(real_estate_data)
real_estate_data = real_estate_data[which(
  !is.na(real_estate_data$beds) & !is.na(real_estate_data$baths)),]
html_list = list()


##-------------------------------------------------------------------------
## FUNCTIONS TO OBTAIN HOUSE SPECIFIC DATA
##-------------------------------------------------------------------------
# This function grabs the values we want to form variables
# These variables include:
          ## Family type, year built, lot sqft, parking, heating, cooling
          # Number of appliances included
          # flooring, new construction, levels, parking spaces etc
# Set house keys to keys that we want variables for:
house_keys = c("flooring", "heating", "cooling",
               "appliances", "home type", "parking features",
               "total spaces", "(levels)|(stories)",
               #year built", "Foundation", "Fire Place", "lot size"
               )


get_num_appliances = function(house_details){
  appliance_string = house_details[grepl("appliances?",
                                        house_details, ignore.case = TRUE)]
  # On Zillow, the appliances seem to be separated by either a comma
  # or a space. Therefore, we can split on both and get max.
  appliances = str_extract(appliance_string, "[^:]+$") %>%
               trimws("both")
  appliances =  appliances[which.max(nchar(appliances))]
  if(nchar(appliances) == 1 |
     nchar(appliances)==0 |
     is.na(appliances)){
    return(NA)
  }
  appliances_1 = str_split(appliances, ",")
  return(length(appliances_1[[1]]))
}
```

```r
get_years_old = function(house_details){
  year_built = house_details[grepl("year[ ]?built",
                                   house_details, ignore.case = TRUE)]
  house_year = str_extract(year_built, "[^:]+$")
  house_year =  house_year[1]
  if(length(house_year) == 0 |
     is.na(house_year) |
     nchar(house_year) != 4){
    return(NA)
  }
  house_age = year(Sys.time())-as.integer(house_year)
  return(house_age)
}

## The Function below finds the key in the house details and
## extracts the information in a convenient form. For the appliances data,
## I aim to get the number of appliances that come with the house.
## For year built, I aim to convert this value to house age.
check_house_keys = function(house_key, house_details){
  if(house_key == "appliances" &
     any(grepl("appliances?", house_details, ignore.case = TRUE))){
      return(get_num_appliances(house_details))
  }else if(house_key == "year built" &
           any(grepl("year[ ]?built", house_key, ignore.case = TRUE))){
      return(get_years_old(house_details))
  }else{
      index = grepl(house_key, house_details,
                    ignore.case = TRUE)
      if(all(index == FALSE)){
        return(NA)
      }
      house_detail = house_details[index]
      house_feature = str_extract(house_detail, "[^:]+$") %>%
                      trimws("both")
      house_feature = paste(house_feature, collapse =", ")
      return(house_feature)
    }
}

#This function reads the html file and obtains the data unorganized
#ENTER WEBSITE COOKIE BELOW!!! ***********************************************
web_cookie_new = paste0("")
get_features = function(house_link){
  Sys.sleep(6)
  home_html = read_html(house_link,
                        cookie = web_cookie_new)
  html_list <<- c(html_list, as.character(home_html))
  various_feat_html = home_html %>%
    html_nodes('.bjTesh') #This does change(CSS one above property details)****
  #Below gives full list of house characteristics
  #Only takes CSS values
  house_details = various_feat_html %>%
    html_nodes(css_values) %>%
    html_text()
  house_characteristics = sapply(house_keys, check_house_keys, house_details)
  return(house_characteristics)
}

#These css values seem to change daily:
## **May need to update everytime we run or use regex
css_values = ".foiYRz"

first_half_links = real_estate_data$house_links #retrunto 1:500
home_features_data_1 = sapply(first_half_links, get_features)

#bFix the data set to be in a form where we will combine it with the other data:
colnames(home_features_data_1) = NULL
home_features_transposed = t(home_features_data_1)
home_features_transposed = as.data.frame(home_features_transposed)

# I would suggest saving the html list.
# If more data is needed, It can be referred to easily.

# We stopped half way, now we finish the rest:
second_half_links = real_estate_data$house_links[501:length(
                                               real_estate_data$house_links)]
second_half_data = sapply(second_half_links, get_features)


# Fix the data set to be in a form where we will combine it with the other data:
colnames(second_half_data ) = NULL
second_half_data_transposed = t(second_half_data)
second_half_data_transposed = as.data.frame(second_half_data_transposed)

write_csv(second_half_data_transposed,
          paste0("/Users/joseph_gonzalez/Desktop/STA 220/FINAL Project/",
                 "Zillow datasets ventura county/",
                 "vc_home_dataset_part2_organized.csv"),
          col_names = TRUE)

#I would suggest saving the html list.
# If more data is needed, It can be referred to easily:
#html_files = unlist(html_list)
#write_csv(as.data.frame(html_files),
# "/Users/joseph_gonzalez/Desktop/STA 220/FINAL Project/
# Zillow datasets ventura county/house_html_files_2.csv")
#write.table(as.data.frame(html_files),
            #"/Users/joseph_gonzalez/Desktop/STA 220/FINAL Project/
            #Zillow datasets ventura county/house_html_files_2.text",
            #sep ="--xx--")
```

Appendix C: Data Cleaning Code

```r
# ------------------------------------------------------------------------------
# CLEANING THE DATA:
#-------------------------------------------------------------------------------
# Libraries:
library(tidyverse)
library(rvest)
library(stringr)

# In this file, I cleaned the data and form the final dataset:
data_folder = paste0("/Users/joseph_gonzalez/Desktop/",
                  "STA 220/FINAL Project/Zillow datasets ventura county")
house_data_files = list.files(data_folder) #File with the zillow datasets only
real_estate_data = lapply(house_data_files, function(x)
                      read.csv(paste0(data_folder,"/", x),
                               header = TRUE,sep = ";"))
real_estate_data = bind_rows(real_estate_data)
real_estate_data = real_estate_data %>%
                 filter(!is.na(real_estate_data$beds) &
                        !is.na(real_estate_data$baths))

# Import the scraped data and combine the data frames:
house_data_1 = read.csv("vc_home_dataset_part1_organized.csv", header = TRUE)
house_data_2 = read.csv("vc_home_dataset_part2_organized.csv", header = TRUE)
ind_home_data = rbind(house_data_1, house_data_2)

# read in appliances data:
# Scraping file fixed this code can be deleted afte writing final file:
appliances_data = read.csv("home_appliances.csv", header = TRUE)
all_homes_data = data.frame(real_estate_data, ind_home_data, appliances_data)
#Save this as full dataset

#-------------------------------------------------------------------------------
### FIX VARIABLES
#-------------------------------------------------------------------------------
## First, we can decide which variables can be removed:
colnames(all_homes_data)

# We can remove address, links, and page number.
full_homes_data = all_homes_data[, -c(1, 8, 12)] #Remove 12 later
colnames(full_homes_data) = c("City", "Address", "Price", "Beds",
                              "Baths", "House_Area", "Flooring",
                              "Heating", "Cooling", "Home_Type",
                              "Garage_Type", "Total_spaces", "Levels",
                              "Age", "Foundation", "Fire_Place",
                              "Lot_Size", "Appliances")
sapply(full_homes_data, class)
address_tables = table(full_homes_data$Address)
remove_index = names(address_tables[which(address_tables > 1)])
for (i in 1:length(remove_index)){
  index = which(full_homes_data$Address == remove_index[i])
  full_homes_data = full_homes_data[-index[1],]
}

# City, Price, Beds, Baths, House_Area, age, total spaces, and
# appliances are in the correct format

# Format Flooring variable:--------------------------------------------------------
table(full_homes_data$Flooring)

# Categorical variable, we can make cost levels for this variable
# High: house has costly materials - Stone, Tile, Hardwood
# Medium: house has moderately costly materials - wood, bamboo
# Low: house has low cost materials - vinyl, carpet, laminate, concrete
#Start with high cost materials first:

full_homes_data = full_homes_data %>%
      mutate(Flooring = replace(Flooring,
                        grepl("(Stone)|(Hardwood)|(Tile)", Flooring), "High"))

full_homes_data = full_homes_data %>%
  mutate(Flooring = replace(Flooring,
                            grepl("(Wood)|(Bamboo)", Flooring), "Medium"))

#We can assume that if flooring material isn't provided,
#it is most likely low cost
full_homes_data = full_homes_data %>%
  mutate(Flooring = replace(Flooring,
                          grepl("(Carpet)|(Vinyl)|(Laminate)", Flooring), "Low"),
         Flooring = replace(Flooring,
                          grepl("(Concrete)|(Linoleum)|(Other)|(See Remarks)",
                                                        Flooring), "Low"),
         Flooring = replace(Flooring, is.na(Flooring), "Unspecified"))

table(full_homes_data$Flooring)

# Format Heating variable:---------------------------------------------------------
# We want to capture that the house has a fireplace -- Removed

#full_homes_data = full_homes_data %>%
   #mutate(Fire_Place = replace(Fire_Place, grepl("Fireplace", Heating), TRUE))

#Now, we investigate the options:
table(full_homes_data$Heating)
any(grepl("Space", full_homes_data$Heating))
which(grepl("Pump", full_homes_data$Heating))

# There appears to be many options,
# One: We can try to organize by most expensive to least expensive
# Heat Pump, Gas, Electric, Natural Gas, Zoned etc.
# Two: We can indicate whether the estate has heating
# For this project, I decided to go with option two

#Option 1 code:
#full_homes_data = full_homes_data %>%
  #mutate(Heating = replace(Heating, grepl("Heat Pump",Heating), "Heat Pump"),
  #Heating = replace(Heating, grepl("(^Gas)|([^A-z]+ Gas)|(Propane)",
                                                          #Heating), "Gas"),
  #Heating = replace(Heating, grepl("Electric",Heating), "Electric"),
  #Heating = replace(Heating, grepl("Natural Gas",Heating), "Natural Gas"),
  #Heating = replace(Heating, grepl("Forced Air",Heating), "Forced Air"),
  #Heating = replace(Heating, grepl("(^Furnace)|([^A-z]+ Furnace)",
                                                      # Heating), "Furnace"),
  #Heating = replace(Heating, grepl("Zoned",Heating), "Zoned"),
  #Heating = replace(Heating, grepl("(Central)|(Other)|(Yes)", Heating),
                                                    #"Unspecified Central"),
  #Heating = replace(Heating, grepl("See Remarks",
                                  #Heating) | is.na(Heating), "None"))

# option two
full_homes_data = full_homes_data %>%
  mutate(Heating = replace(Heating, is.na(Heating) |
                    grepl("(^None$)|(^See Remarks$)|(^Fire Place\\(?s?\\)?$)",
                          Heating), FALSE),
         Heating = replace(Heating, !(Heating == FALSE),TRUE)
         )
full_homes_data$Heating = as.logical(full_homes_data$Heating)
table(full_homes_data$Heating)

# Format Cooling variable:-----------------------------------------------------
# Similar to heating, we have two options
# For this project, I decided option two:
table(full_homes_data$Cooling)

# The "15" and "c" options are cooling systems, scraping code adjusted
# "None, C" is also a cooling system.
# I don't count ceiling fans as a cooling system

full_homes_data = full_homes_data %>%
  mutate(Cooling = replace(Cooling, is.na(Cooling) |
                  grepl("(^See Remarks$)|(^None$)|(^Ceiling Fan\\(?s?\\)?$)",
                                                  Cooling), FALSE),
         Cooling = replace(Cooling,  !(Cooling == FALSE), TRUE))

full_homes_data$Cooling = as.logical(full_homes_data$Cooling)
table(full_homes_data$Cooling)

# Check the home types:--------------------------------------------------------
table(full_homes_data$Home_Type)


# We will check other, MobileManufactured triple, unknown,
# and MobileManufactured Double
# If there is info I can add, I will adjust the data entry
# other:
full_homes_data %>%
  filter(Home_Type == "Other")
```

```r
# Not much info on this house, I suggest not including in the
# modeling or possibly removing these observations.
# Check Na home types:
all_homes_data %>%
  filter(is.na(home.type))
# There are only 2 NA home types. Most variables are missing and,
# therefore, these should not be considered in the modeling part.

# MobileManufactured, triple:
full_homes_data %>%
  filter(Home_Type == "MobileManufactured, Triple")

# The estate is single level and does have a fire place

full_homes_data = full_homes_data %>%
  mutate(Levels = replace(Levels,
                          Home_Type == "MobileManufactured, Triple", "One"))

# MobileManufactured Double:
full_homes_data %>%
  filter(Home_Type == "MobileManufactured, Double")

#Single Level and Raised
full_homes_data = full_homes_data %>%
  mutate(Levels = replace(Levels,
                          Home_Type == "MobileManufactured, Double", "One"),
         Foundation = replace(Foundation,
                          Home_Type == "MobileManufactured, Double", "Raised"))

#Unknown:
full_homes_data %>%
  filter(Home_Type == "Unknown")

all_homes_data %>%
  filter(home.type == "Unknown")
# all unknowns apear to be single level, except last one
# most seem to be a singlefamily homes
#Unknowns seem to be singlefamily
full_homes_data = full_homes_data %>%
  mutate(Levels = replace(Levels,
                          is.na(Levels) & Home_Type =="Unknown", "One"),
         Foundation = replace(Foundation,
                        (is.na(Foundation) | grepl("See Remarks", Foundation))
                                         & Home_Type == "Unknown" , "Raised"),
          Home_Type = replace(Home_Type,
                              Home_Type == "Unknown" , "SingleFamily"))


# Edit Parking features:------------------------------------------------------
#Most concerned with garage types
table(full_homes_data$Garage_Type)
# Three Door, Two Door garage, Single door|garage|attached,
# Covered|carport, Assigned, Unspecified
# Few issues with scraping. This will be adjusted later
# Notes:
# "1" is covered
# "2" garage, "3" garage
# "6", "2, Inside Entrance",  two door
# "3, Inside Entrance" three door
# blanks will be entered manually

all_homes_data %>%
  filter(parking.features == "")

full_homes_data = full_homes_data %>%
  mutate(Garage_Type = replace(Garage_Type,
                          grepl("(Three Door)|(3, Inside Entrance)",
                                Garage_Type), "Three Door"),
         Garage_Type = replace(Garage_Type,
                          grepl("(Two Door)|(6)|(2, Inside Entrance)",
                                 Garage_Type), "Two Door"),
         Garage_Type = replace(Garage_Type,
                            grepl("(Single Door)|(Garage)|(Attached)|(2)|(3)",
                                Garage_Type), "Single Door"),
         Garage_Type = replace(Garage_Type,
                             grepl("(Covered)|(Carport)|(Cochere)|(1)",
                                 Garage_Type), "Covered")
         )
# Blanks:
blanks_corrected = rep("Covered", 27)
blanks_corrected[c(1,9,20,21,25:27)] = "None"
blanks_corrected[c(18,22,23)] = "Single Door"

full_homes_data$Garage_Type[
  which(full_homes_data$Garage_Type == "")] = blanks_corrected

#Check NAs:
all_homes_data %>%
  filter(is.na(parking.features))

# Fix the rest and NAs
#Most NAs do not have garages

full_homes_data = full_homes_data %>%
  mutate(Garage_Type = replace(Garage_Type, is.na(Garage_Type) |
         !(grepl("(Three Door)|(Two Door)|(Single Door)|(Covered)",
                 Garage_Type)), "None"))
table(full_homes_data$Garage_Type)

# Edit Total Spaces:------------------------------------------------------------
class(full_homes_data$Total_spaces) #Appears to be in correct form
all_homes_data %>%
  filter(is.na(total.spaces))
# all seem to have at least two spaces
full_homes_data = full_homes_data %>%
          mutate(Total_spaces = replace(Total_spaces, is.na(Total_spaces), 2))
```

```
# Edit Levels-------------------------------------------------------------------
table(full_homes_data$Levels)

all_homes_data %>%
  filter(levels == "Split")

# "Level" has 3 stories
# "Split" is for multi/split
# For the Missing values, we can assume that they are one story.
full_homes_data = full_homes_data %>%
  mutate(Levels = replace(Levels, grepl("Three|(Levels?)", Levels), "3+"),
         Levels = replace(Levels, grepl("Two|(Split)", Levels), "2"),
         Levels = replace(Levels, grepl("One", Levels), "1"),
         )

# After further consideration, the missing values vary too much on level.
# Therefore, we may need to remove level from the regression analysis

# Age:--------------------------------------------------------------------------
class(full_homes_data$Age)

all_homes_data %>%
  filter(is.na(year.built))
#No futher details on age

# Edit Foundation---------------------------------------------------------------
table(full_homes_data$Foundation)
#Too many missing values, removing foundation from analysis
colnames(full_homes_data)
full_homes_data = full_homes_data[, -15]

#Edit Fireplace:----------------------------------------------------------------
sum(is.na(full_homes_data$Fire_Place))
# Too m any missing values from fire place
# Remove from analysis
full_homes_data = full_homes_data[, -15]


# Next, we can turn lot size into an integer: ----------------------------------
#Make sure all are in sqft first:
full_homes_data %>%
  filter(!grepl("sqft",Lot_Size))
# It appears some entries are in acres
# When I convert to integers, I will keep in mind the positions for acre values
acre_index = which(grepl("Acres", full_homes_data$Lot_Size))

convert_to_integers = function(size_string){
  num_string = str_extract(size_string, "[\\d,]+")
  comma_removed_values = gsub(",", "", num_string)
  return(as.integer(comma_removed_values))
}

sq_feet_values = sapply(full_homes_data$Lot_Size, convert_to_integers)
```

```r
#Convert the acres to square feet:
sq_feet_values[acre_index] = sq_feet_values[acre_index] * 43560
names(sq_feet_values) = NULL
full_homes_data$Lot_Size = sq_feet_values

# Check Appliances:---------------------------------------------
sum(is.na(full_homes_data$Appliances)) #Too many missing to check
class(full_homes_data$Appliances)

# Add population: ------------------------------
population_data = read.csv("Ventura_Cities_Data.csv",
                           header = TRUE, sep = ";")
population_data$populations = gsub(",", "", population_data$populations) %>%
                                      as.integer()

full_homes_data = full_homes_data %>%
                    mutate(population =  NA)
# Function to make a population variable.
# City populations:

get_population = function(city_name){
  full_homes_data <<- full_homes_data %>%
                      mutate(population = replace(population, City == city_name,
                      population_data[population_data$city_names == city_name,
                                      "populations"]))
}

sapply(population_data$city_names, get_population)

# check the datatypes one more time:
sapply(full_homes_data, class)

#Write final dataset to a CSV
write_csv2(full_homes_data, paste0("/Users/joseph_gonzalez/Desktop/STA 220/",
                    "FINAL Project/final_zillow_datset.csv"), col_names = TRUE)
```