King County, USA, Housing Prices

Nigel Brown

2020/11/12

Contents

1 Introduction

The goal of this project is to predict house prices from the House Sales in King County, USA dataset downloaded from kaggle.

The project followed the stages of:

1. Data Exploration
2. Splitting the data into training and test set
3. Feature extraction and selection of predictors
4. Modeling: Linear Regression, Random Forest and XGBoost
5. Evaluating the models performance and finalizing the results

2 Initial data exploration

The dataset consists of house prices in King County, Washington, USA from observed sales between May 2014 and May 2015. The data consists of 21613 rows of data, each row observes a single sale. There are 21 features in the dataset. The features are:

Variable	Description
id	Unique ID for each sale
date	Date of the observed sale
price	Price of each house sold in USD
bedrooms	Number of bedrooms
bathrooms	A full bathroom consists of bath, shower, sink, toilet. Each are scored 0.25
sqft living	Square footage of the interior living space of the house
sqft lot	Square footage of the land area the house resides on
floors	Number of floors, 0.5 is an attic or mezzanine level
waterfront	Does the house overlook the waterfront front
view	An index from 0 to 4 of how good the view of the property is
condition	An index from 1 to 5 on the condition of the house when sold
grade	An index from 1 to 13 of the construction and design
sqrt above	Square footage of the interior housing space above ground level
sqrt basement	Square footage of the interior housing space below ground level
yr_built	the year the house was built
yr_renovated	the year of the house's last renovation
zipcode	the area zip code where the house is situated
lat	Latitude
long	Longitude
sqft_living15	The square footage of the interior living space for the closest 15 neighbors
sqft_lot15	The square footage of land for the closest 15 neighbors' houses

2.1 Data status

The next step is to analyze the data for missing values. For this analysis the df_status function from the funModelling package is utilized.

Table 2: Dataset status

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
id	0	0.00	0	0	0	0	character	21436
date	0	0.00	0	0	0	0	POSIXct/POSIXt	372
price	0	0.00	0	0	0	0	numeric	4028
bedrooms	13	0.06	0	0	0	0	numeric	13
bathrooms	10	0.05	0	0	0	0	numeric	30
sqft_living	0	0.00	0	0	0	0	numeric	1038
sqft_lot	0	0.00	0	0	0	0	numeric	9782
floors	0	0.00	0	0	0	0	numeric	6
waterfront	21450	99.25	0	0	0	0	numeric	2
view	19489	90.17	0	0	0	0	numeric	5
condition	0	0.00	0	0	0	0	numeric	5
grade	0	0.00	0	0	0	0	numeric	12
sqft_above	0	0.00	0	0	0	0	numeric	946
sqft_basement	13126	60.73	0	0	0	0	numeric	306
yr_built	0	0.00	0	0	0	0	numeric	116
yr_renovated	20699	95.77	0	0	0	0	numeric	70
zipcode	0	0.00	0	0	0	0	numeric	70
lat	0	0.00	0	0	0	0	numeric	5034
long	0	0.00	0	0	0	0	numeric	752
sqft_living15	0	0.00	0	0	0	0	numeric	777
sqft_lot15	0	0.00	0	0	0	0	numeric	8689

- **q_zeros:** quantity of zeros (p_zeros: in percent)
- **q_inf:** quantity of infinite values (p_inf: in percent)
- **q_na:** quantity of NA (p_na: in percent)
- **type:** the variable type
- **unique:** quantity of unique values

3 Data Preprocessing

As is shown in the table above there are no instances of missing data or values being infinite, however there are a number of variables where the percentage of zeros is greater than 60%, these may not be useful for modeling and they may dramatically bias the model. Therefore for this project the decision is made to remove these variables from the dataset. The features removed are: waterfront, view, sqft_basement, yr_renovated.

Once the predominately zero columns are removed, there are 17 left in the dataset. The date feature is split into year and month features and the original date feature is dropped. It is also decided to convert all variables of type double except bathrooms, lat and long to integer type.

3.1 Data Anomalies

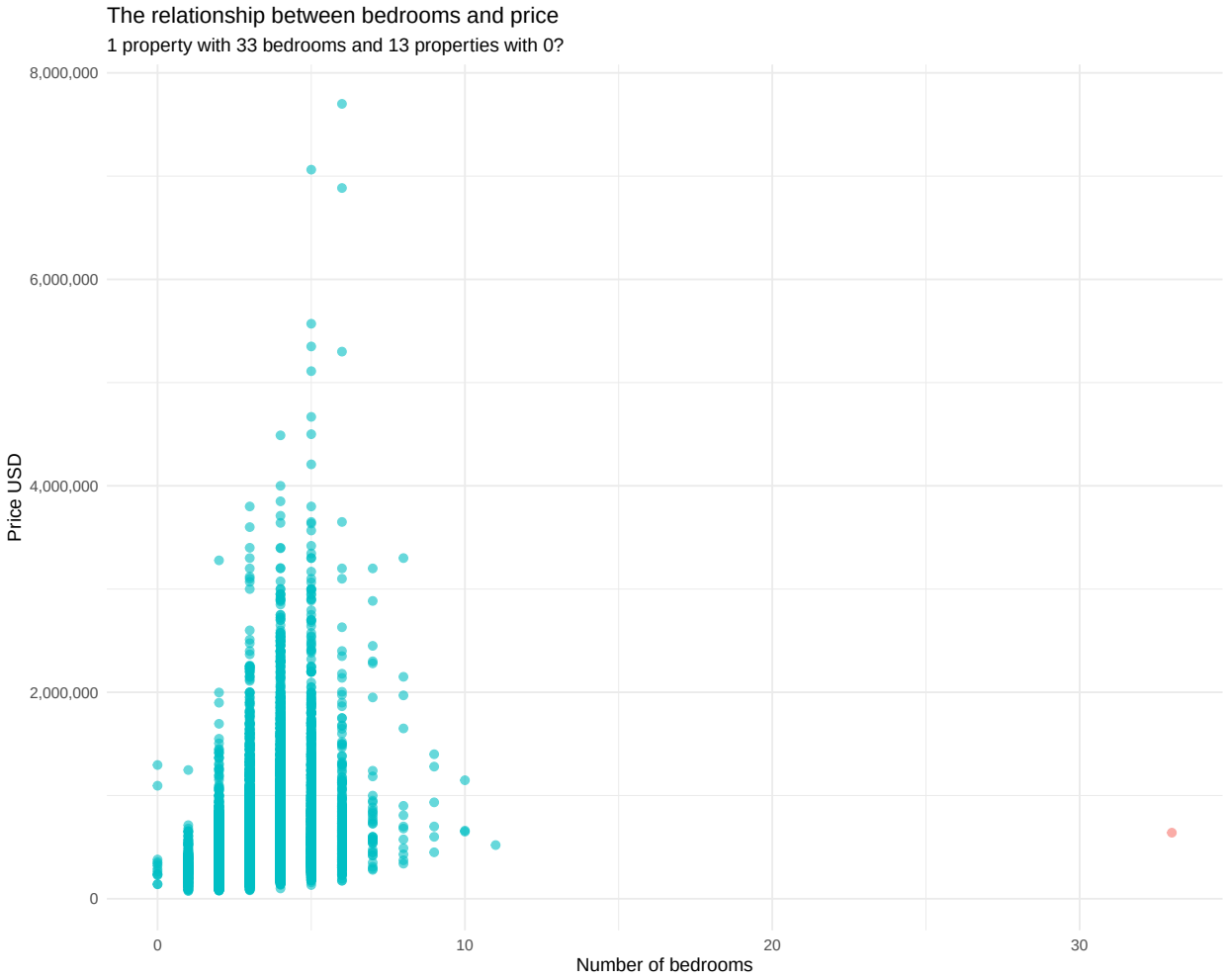


Table 3: Anomalous Data

id	price	bedrooms	bathrooms	floors	sqft_living
6306400140	1095000	0	0.00	3	3064
3918400017	380000	0	0.00	3	1470
1453602309	288000	0	1.50	3	1430
6896300380	228000	0	1.00	1	390
2954400190	1295650	0	0.00	2	4810
2569500210	339950	0	2.50	2	2290
2310060040	240000	0	2.50	2	1810
3374500520	355000	0	0.00	2	2460
7849202190	235000	0	0.00	2	1470
7849202299	320000	0	2.50	2	1490
9543000205	139950	0	0.00	1	844
2402100895	640000	33	1.75	1	1620
1222029077	265000	0	0.75	1	384
3980300371	142000	0	0.00	1	290

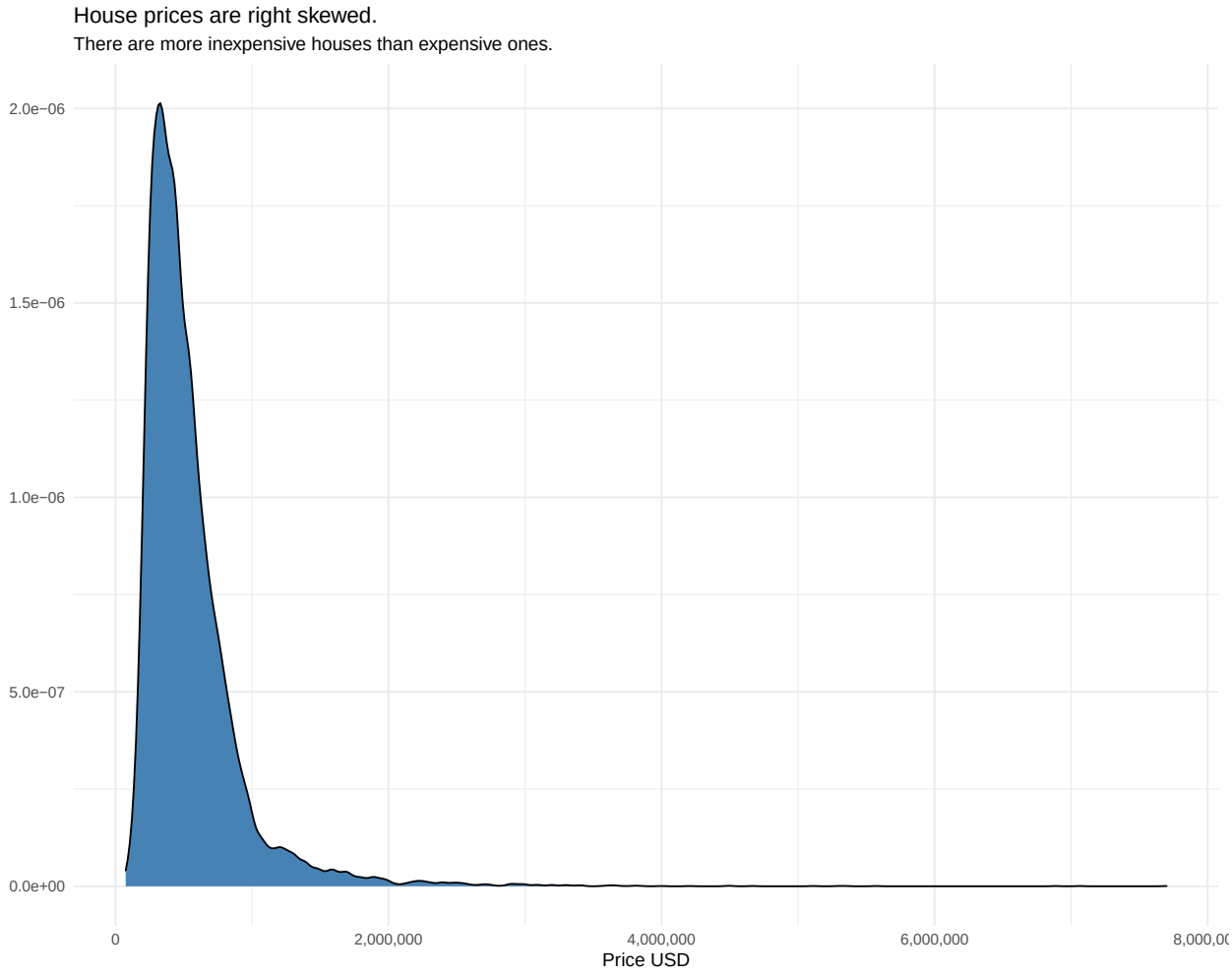
Exploring the data it is found that a single property has been listed with 33 bedrooms. This property is

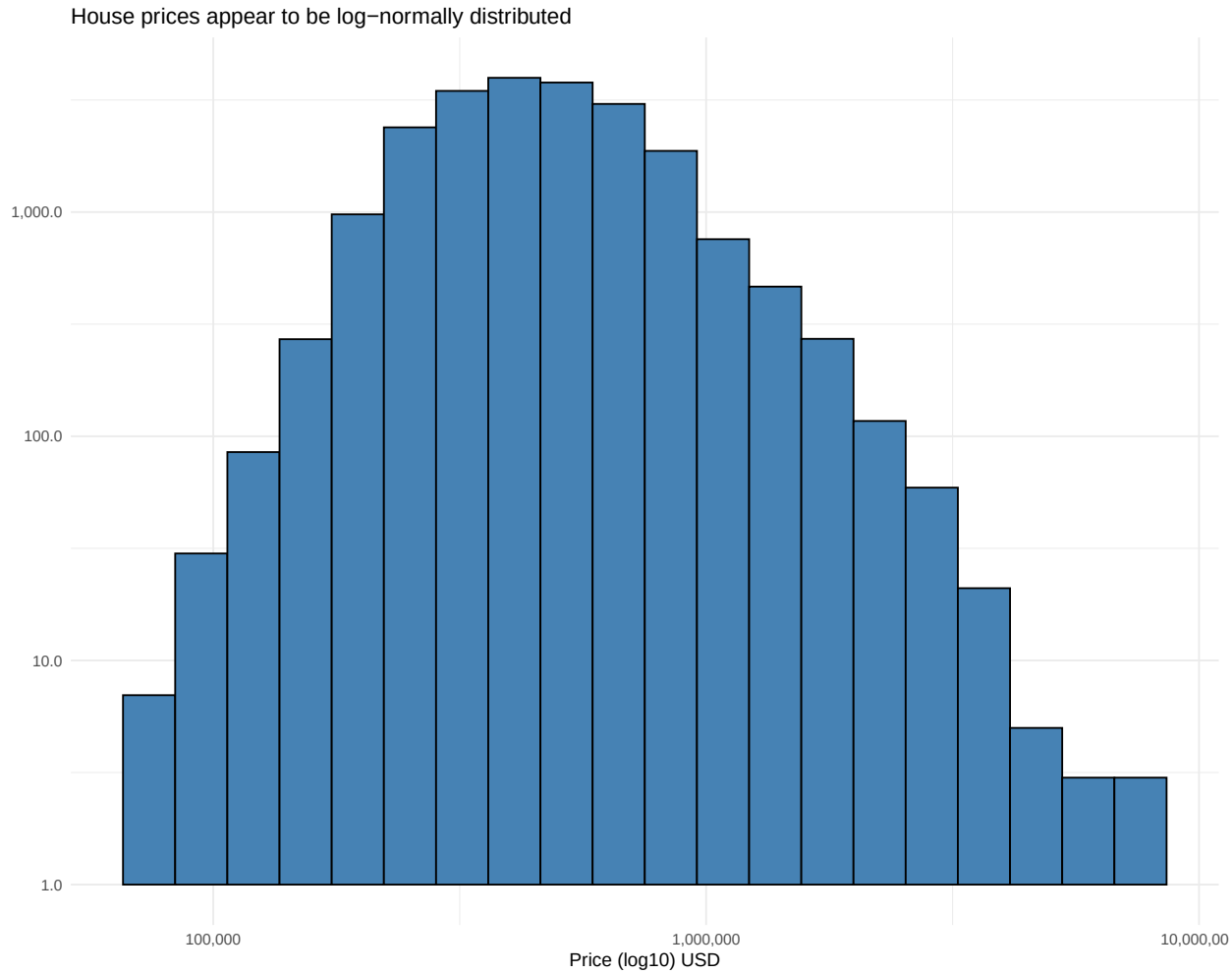
re-entered as a 3 bedroom property. Also 13 properties are listed with zero bedrooms, as these properties range in size and there is no obvious method of reincorporating these properties in the data with a bedroom count that is explainable, these properties are dropped from the data.

4 Exploratory Data Analysis

Now that the data is cleaned it consists of 21600 rows of data and 18 columns. An exploratory data analysis is now performed to visualize relationships and patterns in the data.

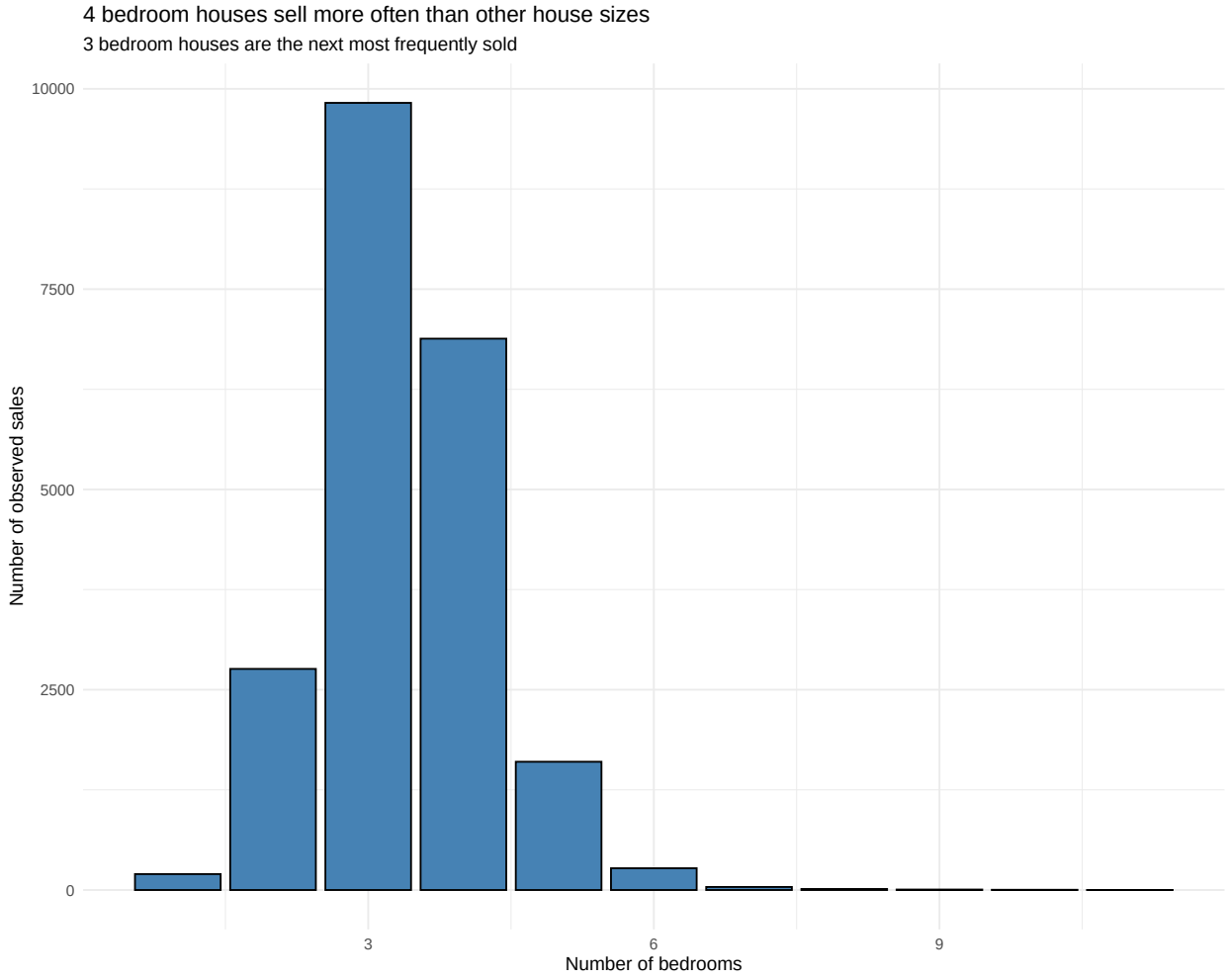
4.1 How are house prices distributed?



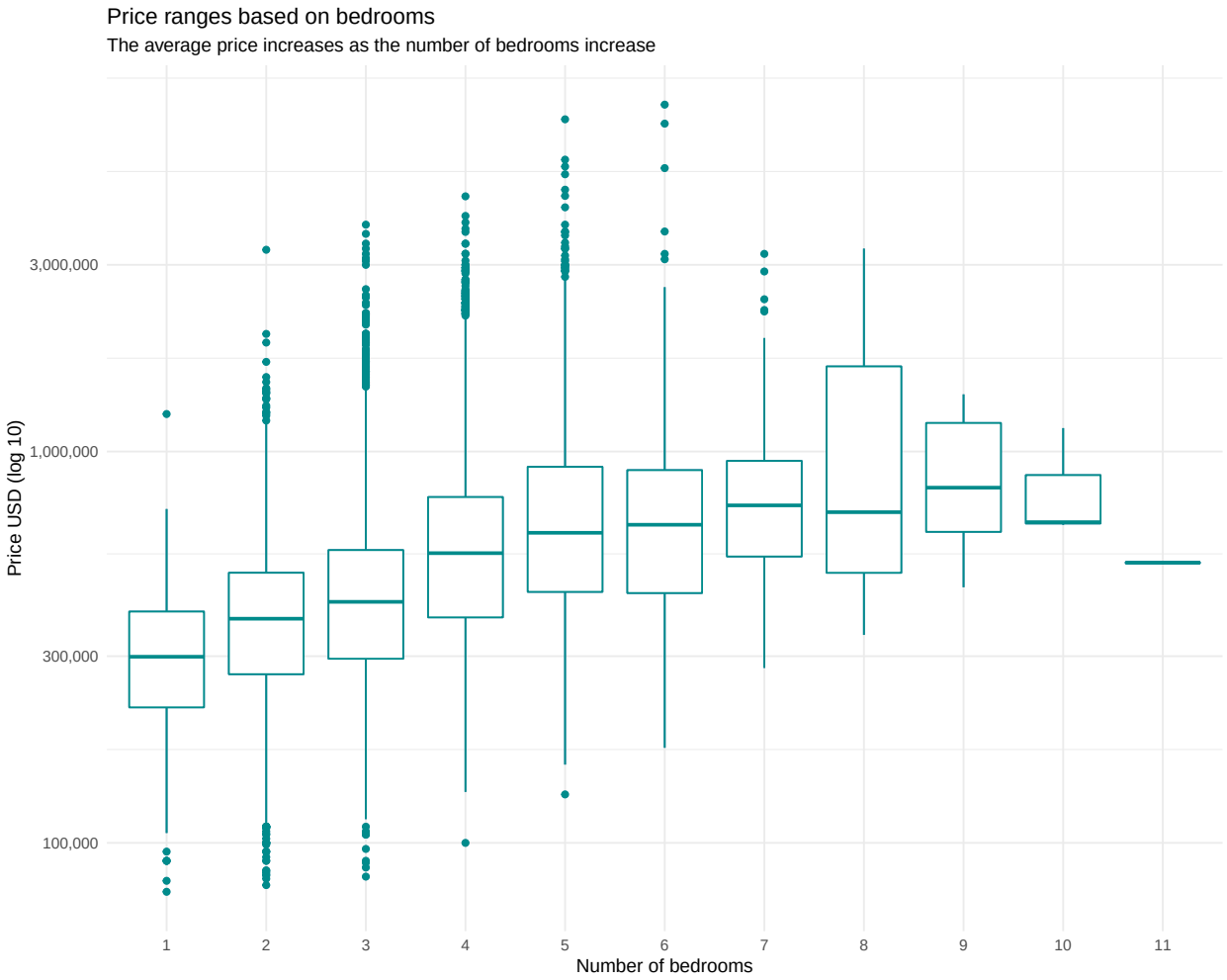


A strong argument can be made that the price should be log-transformed. The advantages of doing this are that no houses would be predicted with negative sale prices and that errors in predicting expensive houses will not have an undue influence on the model. Also, from a statistical perspective, a logarithmic transform may also stabilize the variance in a way that makes inference more legitimate. When the models are built a final pre-processing step of transforming the prices into logs will be performed.

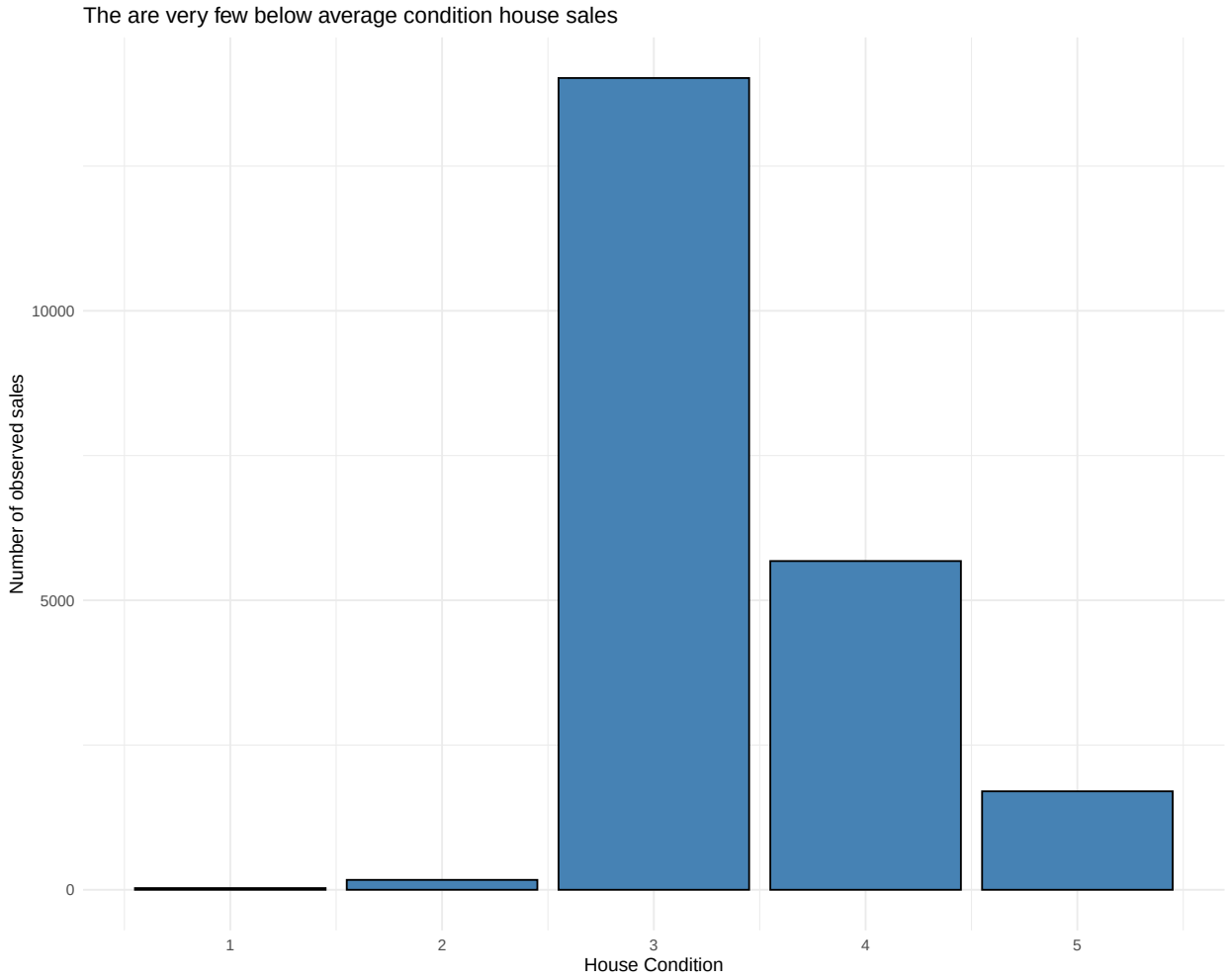
4.2 How many houses of each bedroom count were sold?



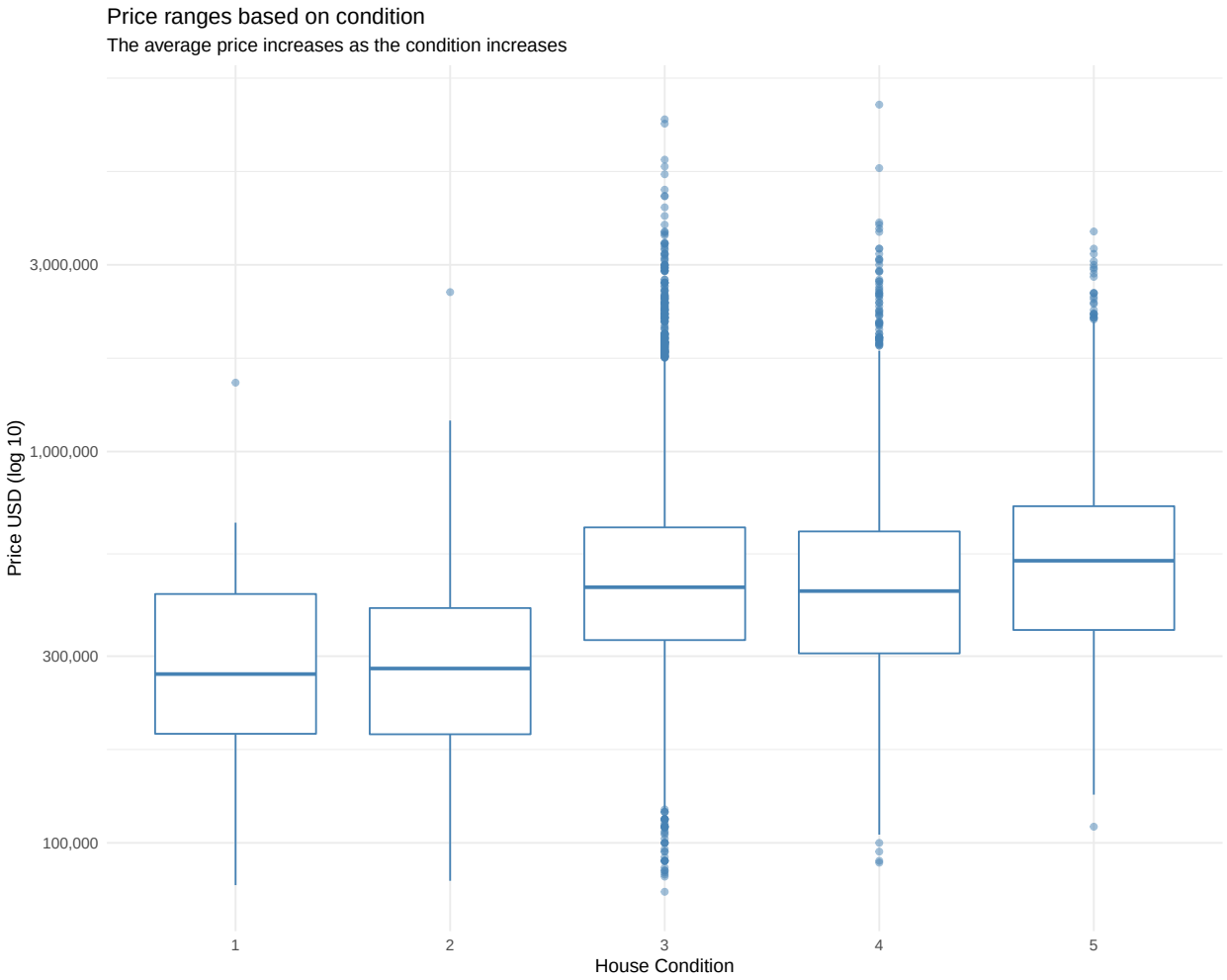
4.2.1 Price to number of bedroom relationship



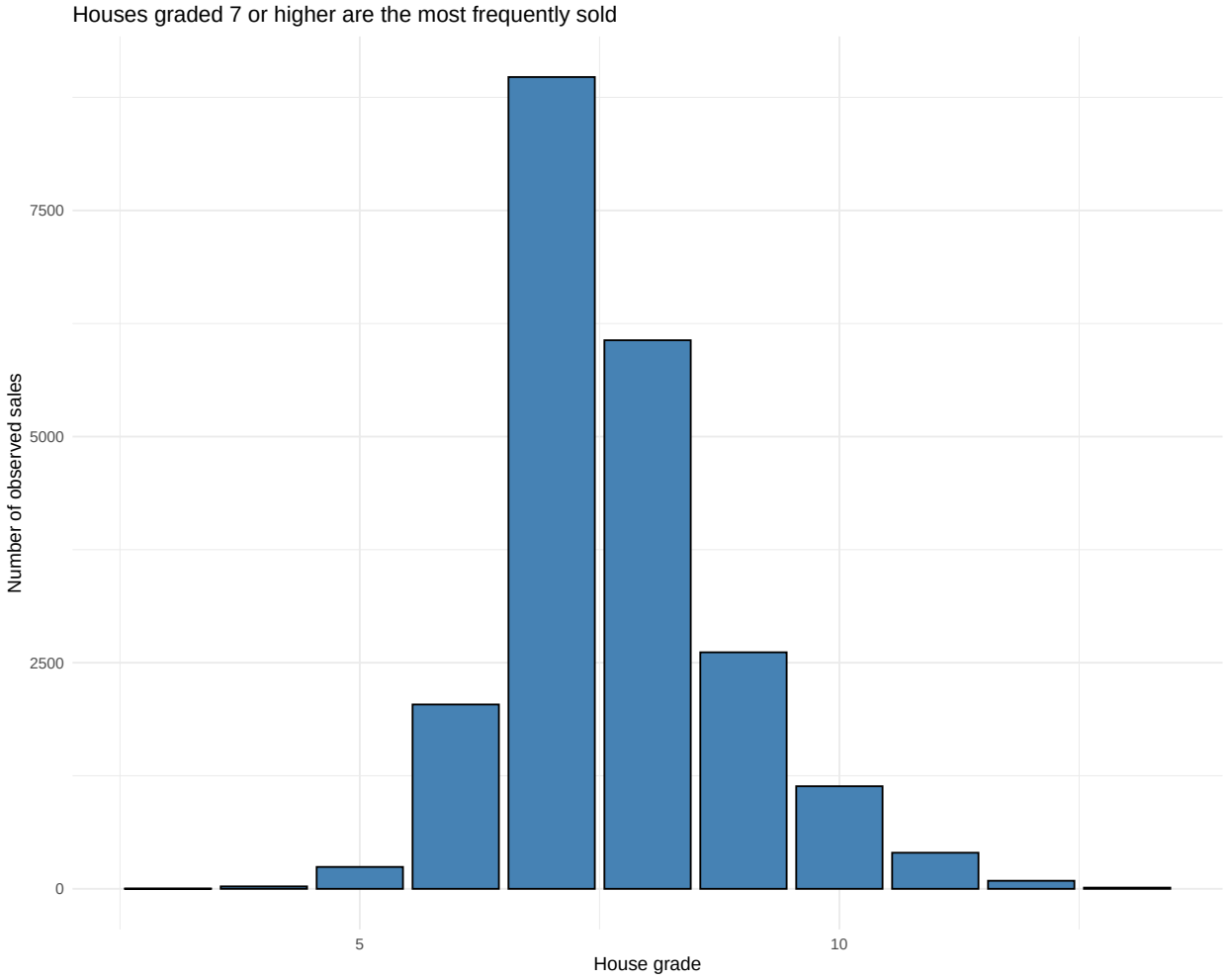
4.3 What condition were the houses in when sold?



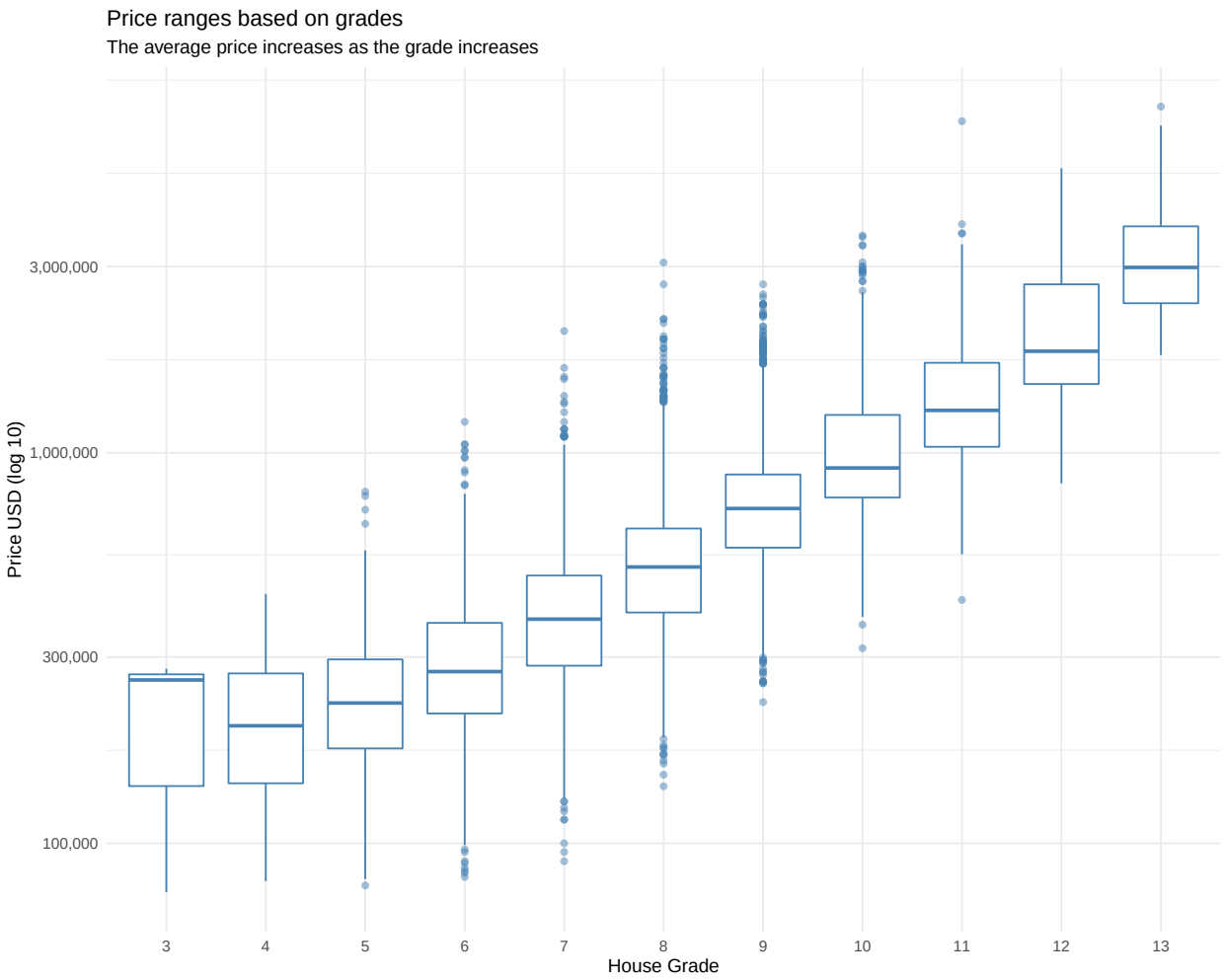
4.3.1 Price to house condition relationship



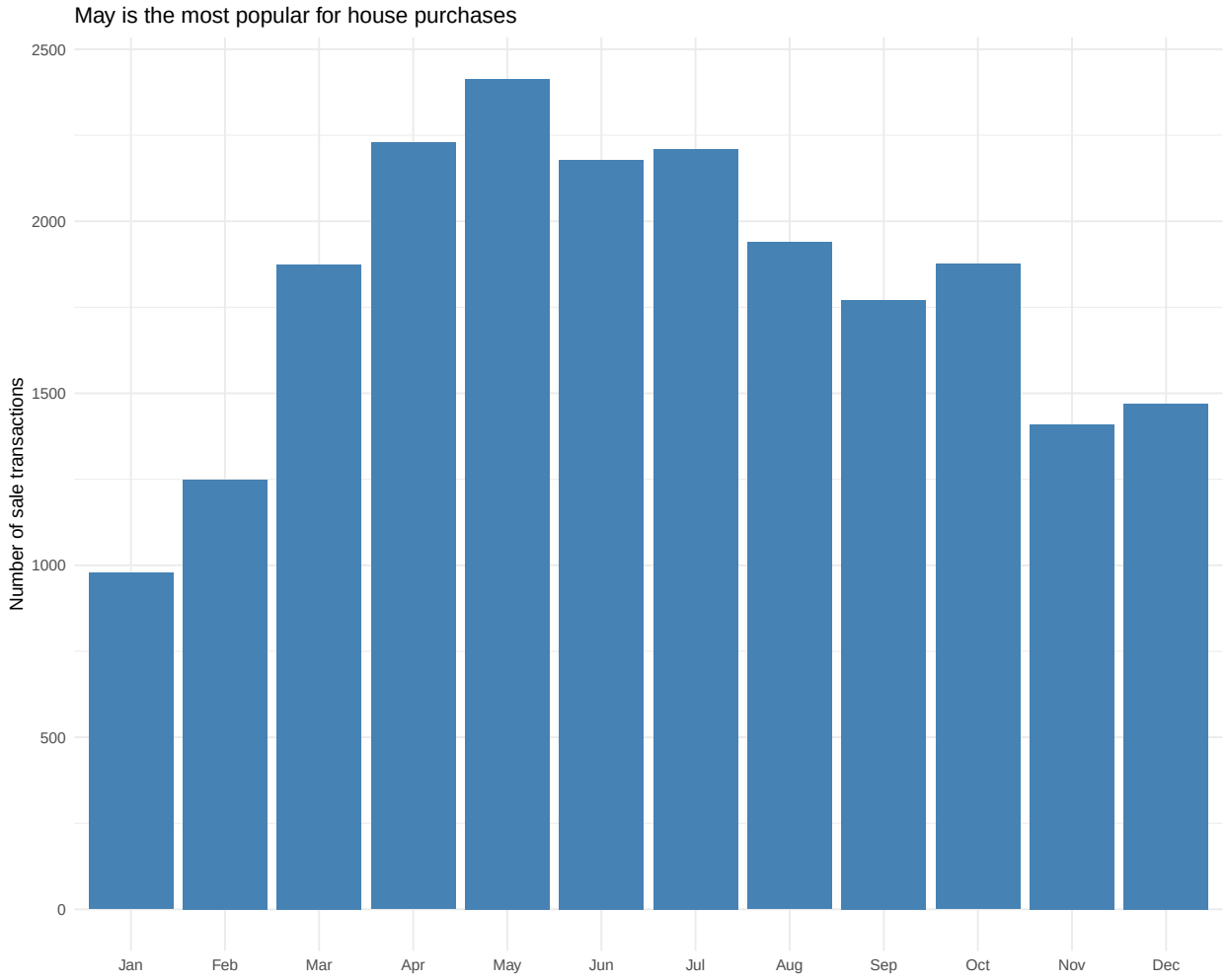
4.4 Which grade of houses sold the most?



4.4.1 Price to house grade relationship



4.5 Is there a month when most sales occur?



4.6 Spatial analysis plots

4.6.1 Where are the sales located within King County?

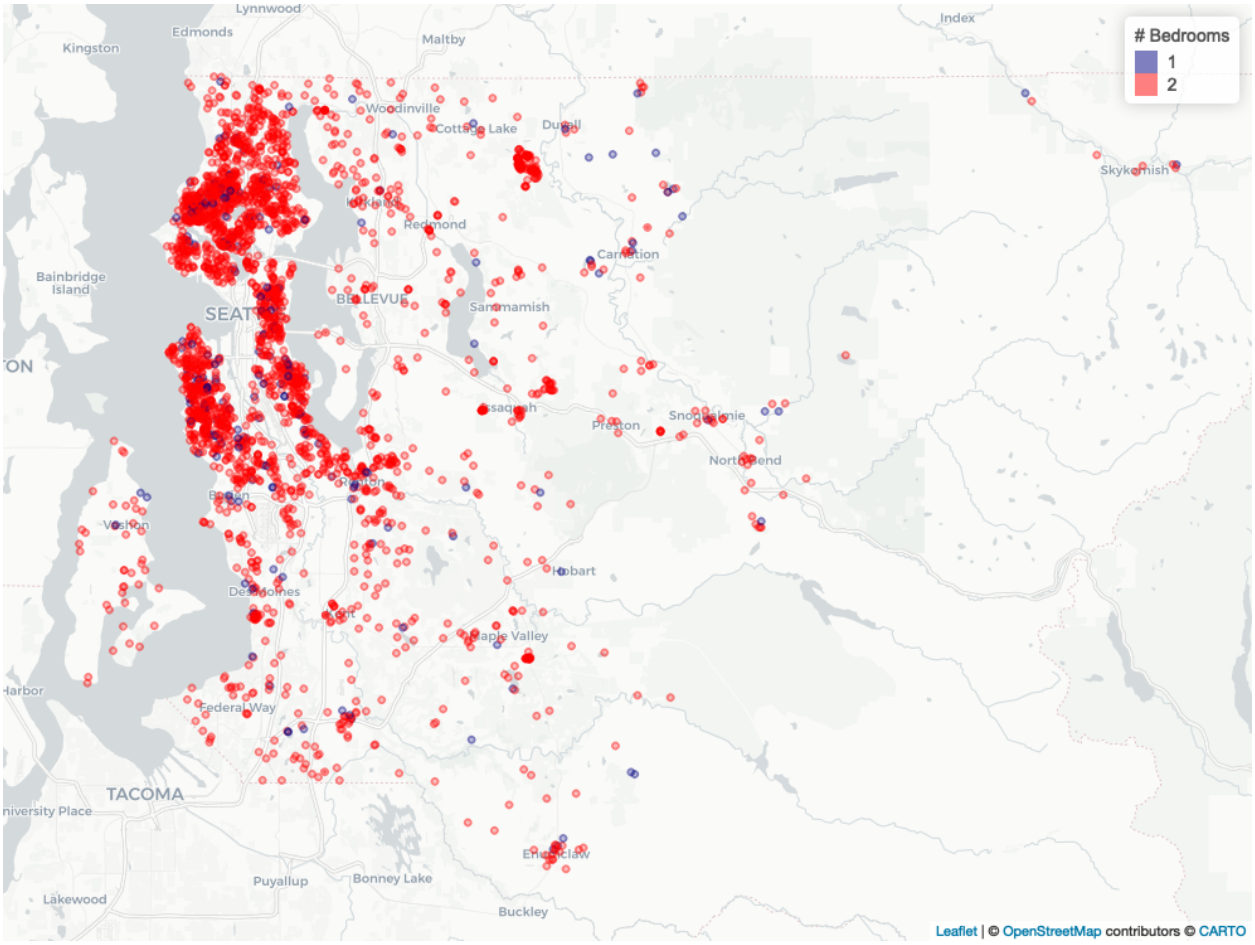


Figure 1: Location of 1 and 2 bedroom houses sold

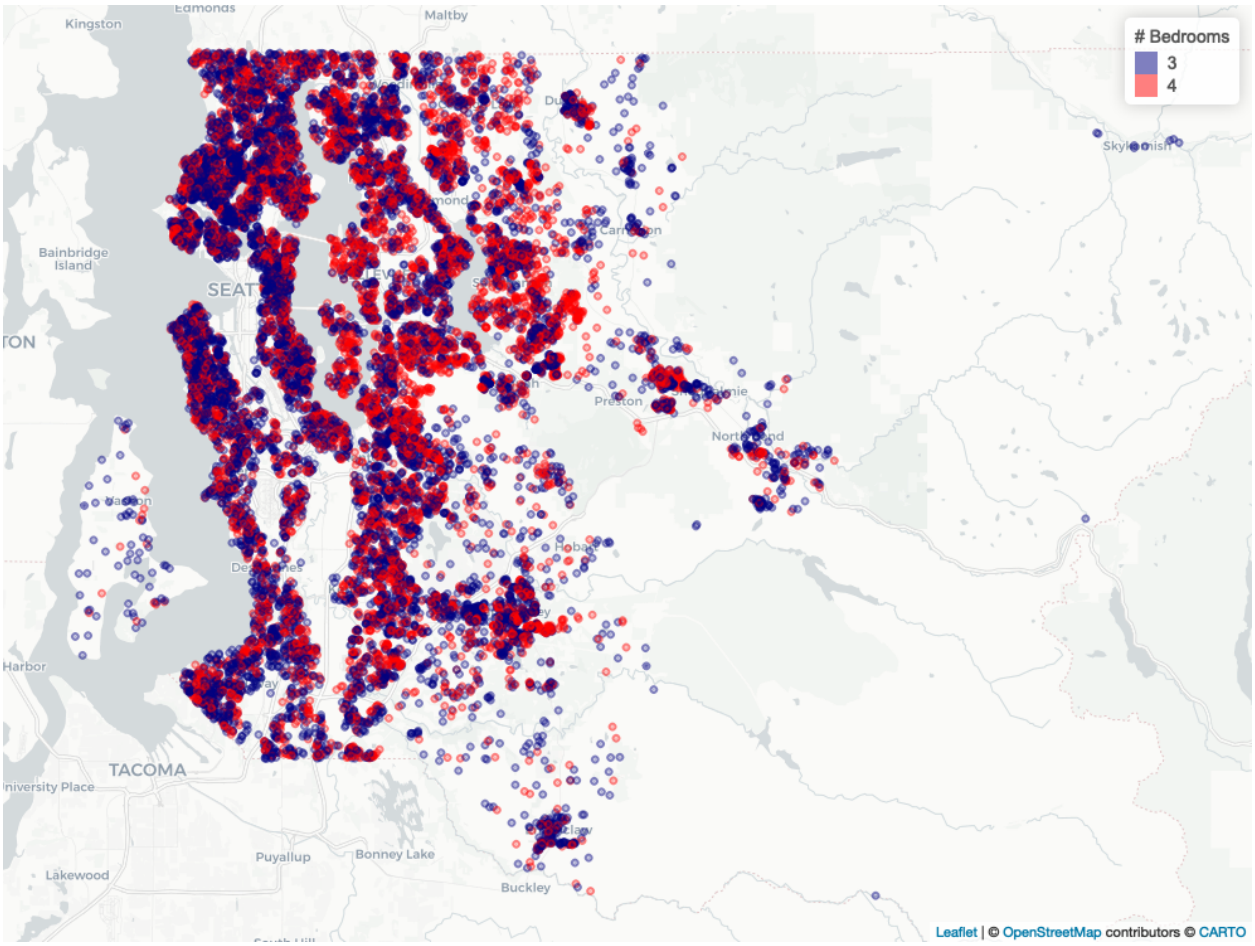


Figure 2: Location of 3 and 4 bedroom houses sold

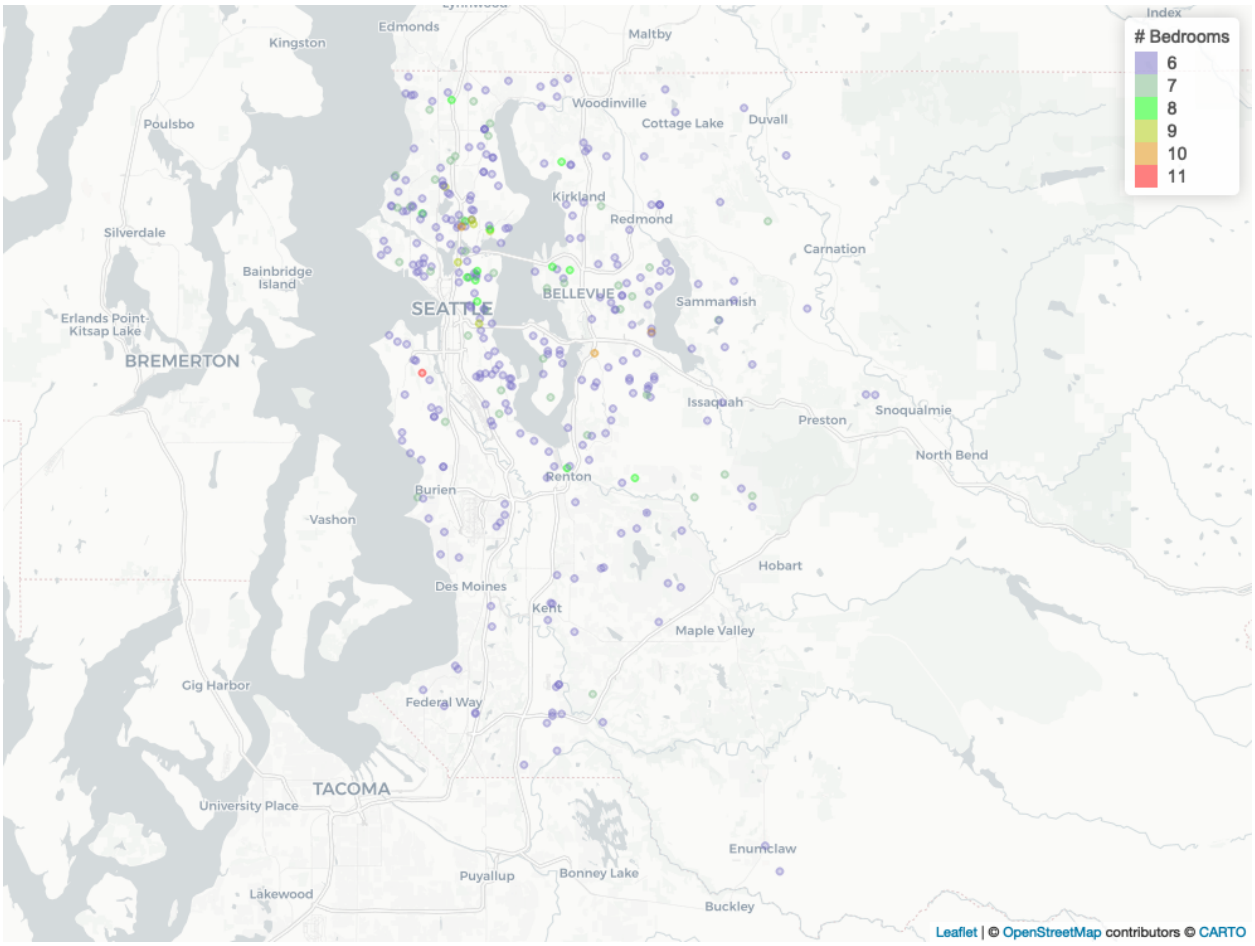
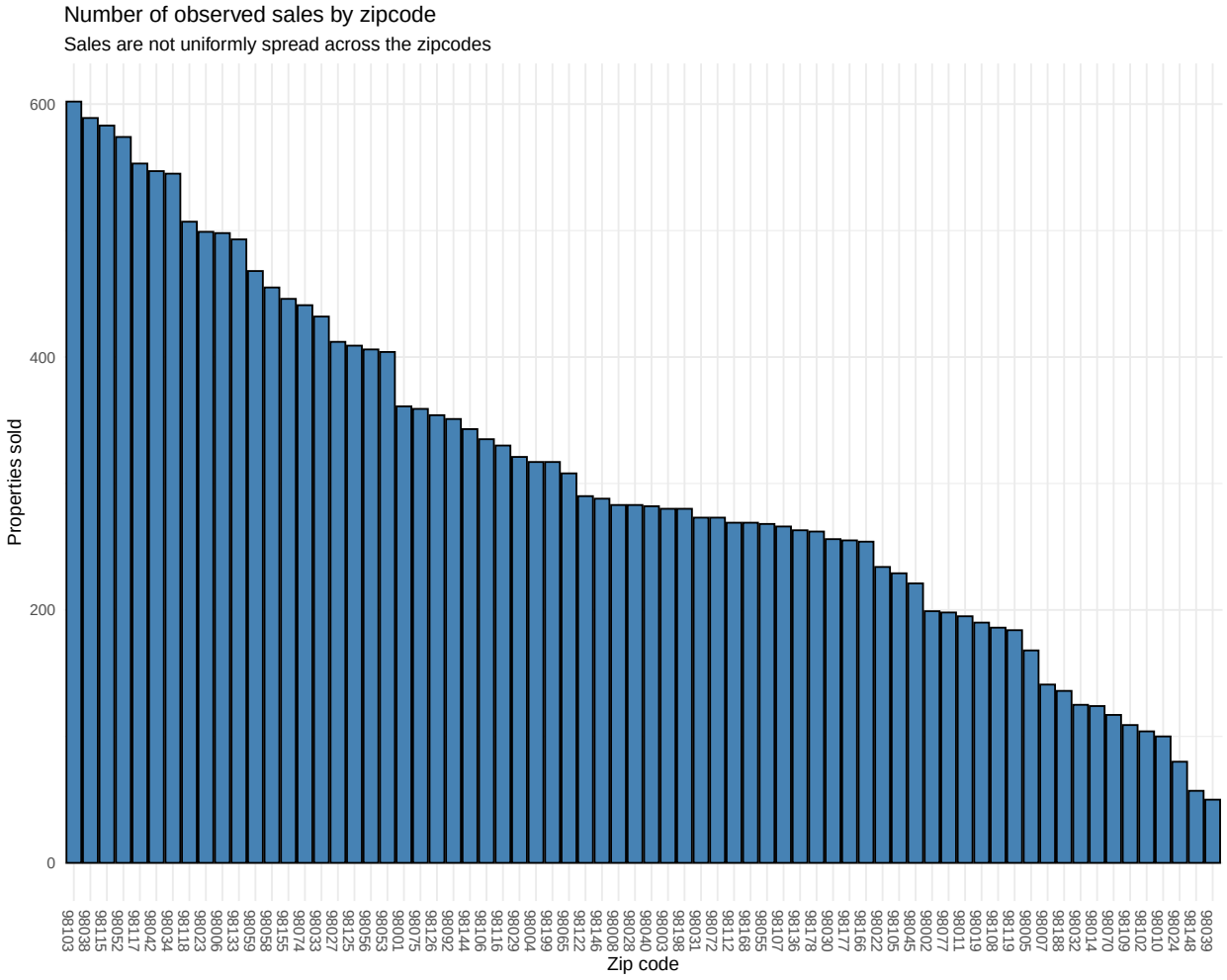
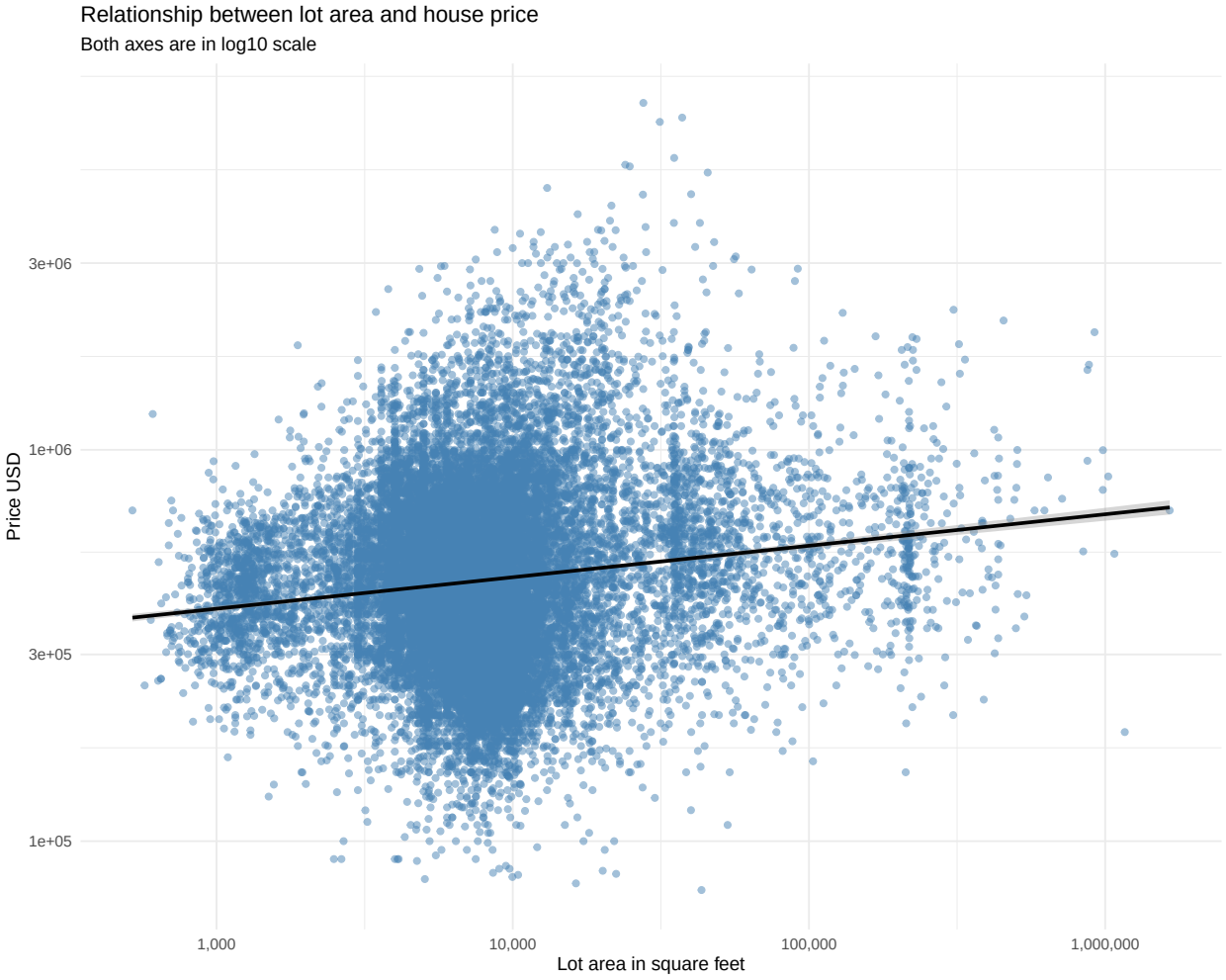


Figure 3: Location of 5 or more bedroom houses sold

4.7 Are the sales evenly spread across the county?

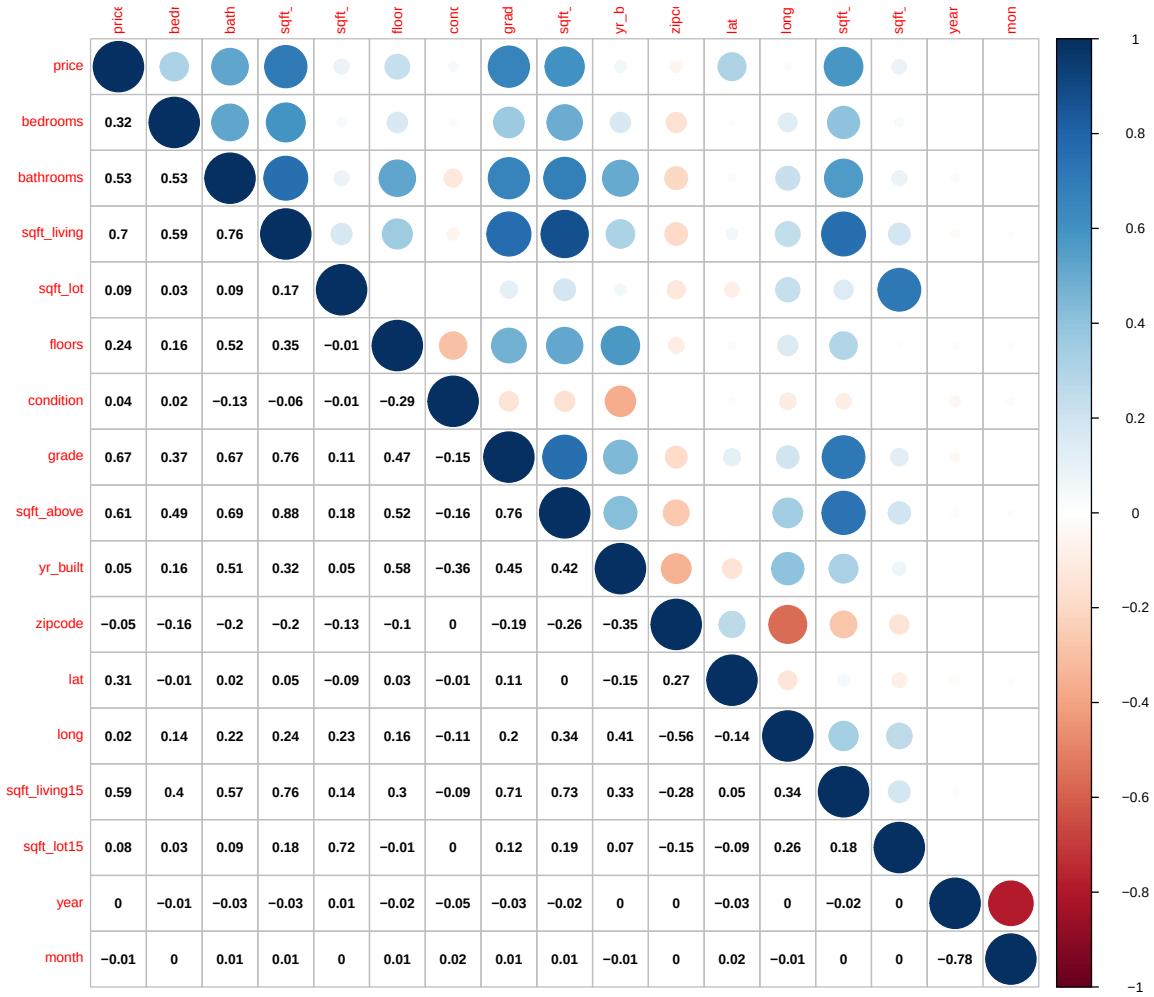


4.8 Does price increase as the lot area gets larger?



4.9 Does price increase as the interior area gets larger?

4.10 How are the features correlated ?



The matrix indicates some correlation between price and:

- **sqft_ features**, logically this makes sense as the more space a property has normally equates to a higher price
- **grade**, again this makes sense as the better condition a property is in, demands a higher price.
- **bathrooms**, the number of bathrooms correlates with the price, this is probably due to the space needed to have multiple bathrooms in a property.

5 Methods/Analysis

5.1 Data split

The first step is to split the data into training and testing datasets. The split will be an 80/20 split with 80% of the data in the training set and the remaining 20% in the test set. The split of the data is stratified by district, to ensure that the number of data points in the training data is equivalent to the proportions in the original data set.

Parameter tuning will be performed, when modeling and to do that cross-validation is used. A cross-validated version of the training set is created in preparation for the parameter tuning. As the training and test set

are stored in two separate dataframes this ensures that information leakages do not accidentally occur.

5.2 Linear Regression model

A model is built using price as the outcome, the id column is unused and the remaining features are predictors. A recipe is created using the tidymodels package. The predictors that start with **sqft_** are transformed into log10 values to coincide with the price log transformation. The data is grouped for **bedrooms, bathrooms and zipcodes** where the value in the data is less than 1% of the data set into a variable value 'other'.

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##         ID          1
##    outcome          1
##  predictor         16
##
## Training data contained 16201 data points and no missing data.
##
## Operations:
##
## Collapsing factor levels for bedrooms, bathrooms, zipcode [trained]
## Log transformation on sqft_living, sqft_lot, ... [trained]
```

Once the recipe is created, a specification is defined using **lm** as the engine for the linear regression model. Finally both the model and recipe are added to a workflow.

This workflow is then used to fit the model using the training dataset. The result of the data fit is shown below:

term	estimate	std.error	statistic	p.value
(Intercept)	-125.6405419	5.3045225	-23.6855518	0.0000000
bedrooms3	-0.0180269	0.0025152	-7.1672673	0.0000000
bedrooms4	-0.0248653	0.0029711	-8.3691684	0.0000000
bedrooms5	-0.0358558	0.0039389	-9.1028967	0.0000000
bedrooms6	-0.0531001	0.0070345	-7.5484884	0.0000000
bedroomsother	-0.0066372	0.0068822	-0.9644106	0.3348546
bathrooms1.5	-0.0016322	0.0033680	-0.4846180	0.6279539
bathrooms1.75	0.0068248	0.0029197	2.3375201	0.0194243
bathrooms2	0.0092273	0.0032725	2.8196287	0.0048138
bathrooms2.25	0.0118030	0.0035732	3.3031531	0.0009581
bathrooms2.5	0.0190563	0.0034312	5.5537494	0.0000000
bathrooms2.75	0.0261841	0.0043141	6.0694680	0.0000000
bathrooms3	0.0339860	0.0049012	6.9341924	0.0000000
bathrooms3.25	0.0555243	0.0056234	9.8737545	0.0000000
bathrooms3.5	0.0486587	0.0054119	8.9910802	0.0000000
bathroomsother	0.0740676	0.0056074	13.2089551	0.0000000
sqft_living	0.3330435	0.0108440	30.7123101	0.0000000
sqft_lot	0.0530047	0.0047622	11.1303954	0.0000000
floors	0.0026340	0.0021607	1.2190418	0.2228462
condition	0.0236490	0.0012256	19.2959346	0.0000000
grade	0.0530720	0.0011201	47.3822044	0.0000000

term	estimate	std.error	statistic	p.value
sqft_above	0.0902251	0.0099686	9.0509532	0.0000000
yr_built	-0.0006550	0.0000408	-16.0492710	0.0000000
zipcode98003	-0.0210894	0.0082358	-2.5607002	0.0104552
zipcode98004	0.3180431	0.0089489	35.5400169	0.0000000
zipcode98006	0.1687646	0.0079958	21.1065308	0.0000000
zipcode98008	0.1697763	0.0092268	18.4002784	0.0000000
zipcode98022	0.1797979	0.0095752	18.7775424	0.0000000
zipcode98023	-0.0487092	0.0072400	-6.7278244	0.0000000
zipcode98027	0.1698134	0.0082990	20.4618868	0.0000000
zipcode98028	-0.0644712	0.0102714	-6.2767891	0.0000000
zipcode98029	0.2012915	0.0089778	22.4211235	0.0000000
zipcode98030	0.0116561	0.0084249	1.3835264	0.1665227
zipcode98031	-0.0011347	0.0084568	-0.1341732	0.8932673
zipcode98033	0.1533608	0.0089693	17.0984965	0.0000000
zipcode98034	0.0299585	0.0091059	3.2899929	0.0010040
zipcode98038	0.1198723	0.0075149	15.9512978	0.0000000
zipcode98040	0.2504830	0.0090860	27.5680809	0.0000000
zipcode98042	0.0463659	0.0073046	6.3475129	0.0000000
zipcode98045	0.2348816	0.0107394	21.8709266	0.0000000
zipcode98052	0.1073616	0.0085829	12.5088001	0.0000000
zipcode98053	0.1241582	0.0092263	13.4569783	0.0000000
zipcode98055	-0.0016462	0.0086507	-0.1902938	0.8490813
zipcode98056	0.0564451	0.0079701	7.0821483	0.0000000
zipcode98058	0.0269561	0.0075845	3.5541114	0.0003803
zipcode98059	0.0807970	0.0077462	10.4304942	0.0000000
zipcode98065	0.1941671	0.0096722	20.0747901	0.0000000
zipcode98072	-0.0065520	0.0103157	-0.6351455	0.5253425
zipcode98074	0.1317231	0.0085923	15.3303851	0.0000000
zipcode98075	0.1561429	0.0089388	17.4679527	0.0000000
zipcode98092	0.0385466	0.0079578	4.8439033	0.0000013
zipcode98103	0.1194636	0.0089643	13.3265617	0.0000000
zipcode98105	0.1855722	0.0103167	17.9875452	0.0000000
zipcode98106	0.0034881	0.0087705	0.3977128	0.6908471
zipcode98107	0.1227086	0.0101592	12.0785486	0.0000000
zipcode98112	0.2382149	0.0098169	24.2658311	0.0000000
zipcode98115	0.1277207	0.0088908	14.3655431	0.0000000
zipcode98116	0.1433272	0.0090373	15.8595091	0.0000000
zipcode98117	0.1004173	0.0092125	10.9000859	0.0000000
zipcode98118	0.0687505	0.0079450	8.6533234	0.0000000
zipcode98122	0.1585531	0.0094011	16.8653182	0.0000000
zipcode98125	0.0104350	0.0096136	1.0854420	0.2777422
zipcode98126	0.0803498	0.0086502	9.2887711	0.0000000
zipcode98133	-0.0666529	0.0096440	-6.9113552	0.0000000
zipcode98136	0.1351963	0.0093071	14.5261588	0.0000000
zipcode98144	0.1289776	0.0088775	14.5286678	0.0000000
zipcode98146	-0.0069980	0.0088337	-0.7921968	0.4282576
zipcode98155	-0.0727969	0.0098777	-7.3698531	0.0000000
zipcode98166	0.0405326	0.0088387	4.5858264	0.0000046
zipcode98168	-0.0806746	0.0087787	-9.1897729	0.0000000
zipcode98177	-0.0217280	0.0108042	-2.0110661	0.0443351
zipcode98178	-0.0251016	0.0089596	-2.8016508	0.0050902
zipcode98198	-0.0180141	0.0084775	-2.1249278	0.0336078

term	estimate	std.error	statistic	p.value
zipcode98199	0.1302631	0.0096231	13.5365018	0.0000000
zipcodeother	0.0714375	0.0068707	10.3973411	0.0000000
lat	0.5496462	0.0144420	38.0587363	0.0000000
long	-0.3545959	0.0131752	-26.9139006	0.0000000
sqft_living15	0.1959208	0.0087638	22.3556023	0.0000000
sqft_lot15	-0.0104592	0.0052201	-2.0036522	0.0451240
year	0.0299632	0.0024365	12.2977520	0.0000000
month	0.0011008	0.0003652	3.0145469	0.0025776

The model fit is next used to predict house prices and plot, the accuracy of the predictions is shown via the metrics rmse (Root mean square error, the standard deviation of the prediction errors), rsq (R^2 a measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model) and mae (mean absolute error, the average of all absolute errors).

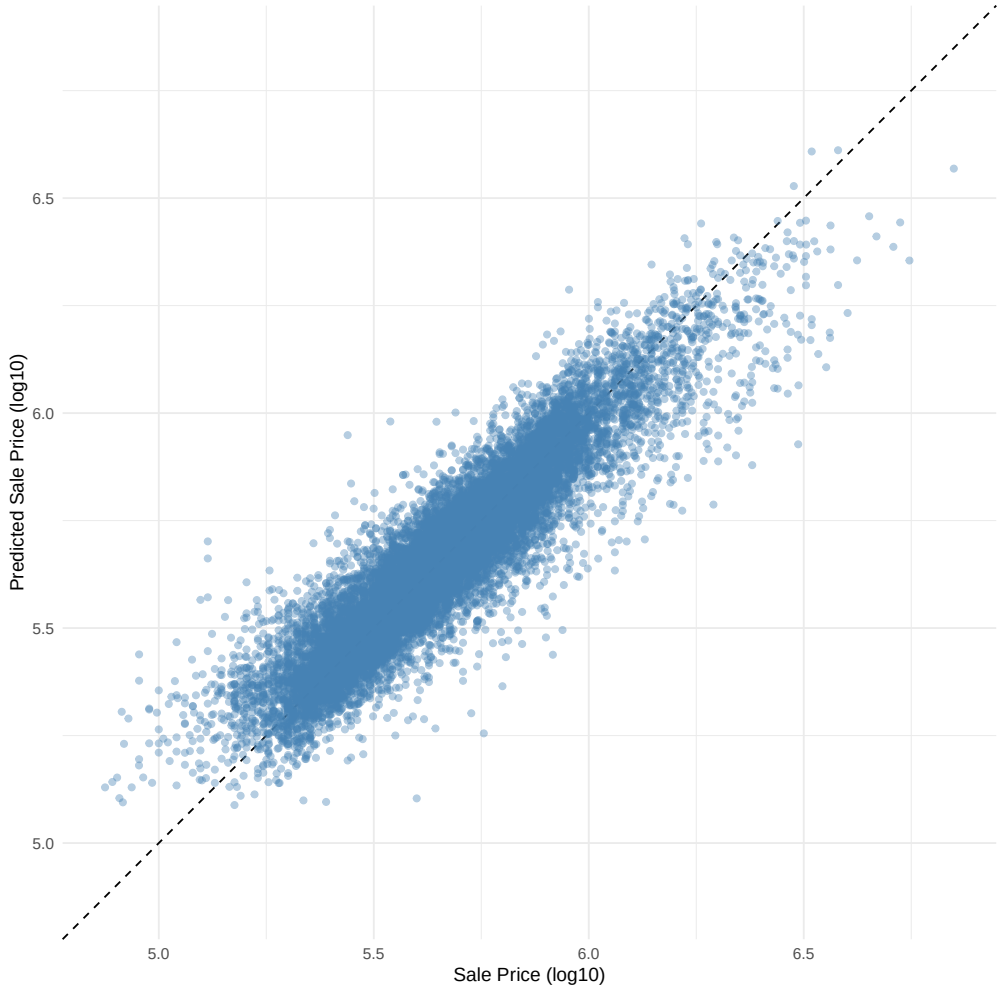


Table 5: Linear Regression Model Result Metrics

.metric	.estimator	.estimate
rmse	standard	0.0895170
rsq	standard	0.8471636

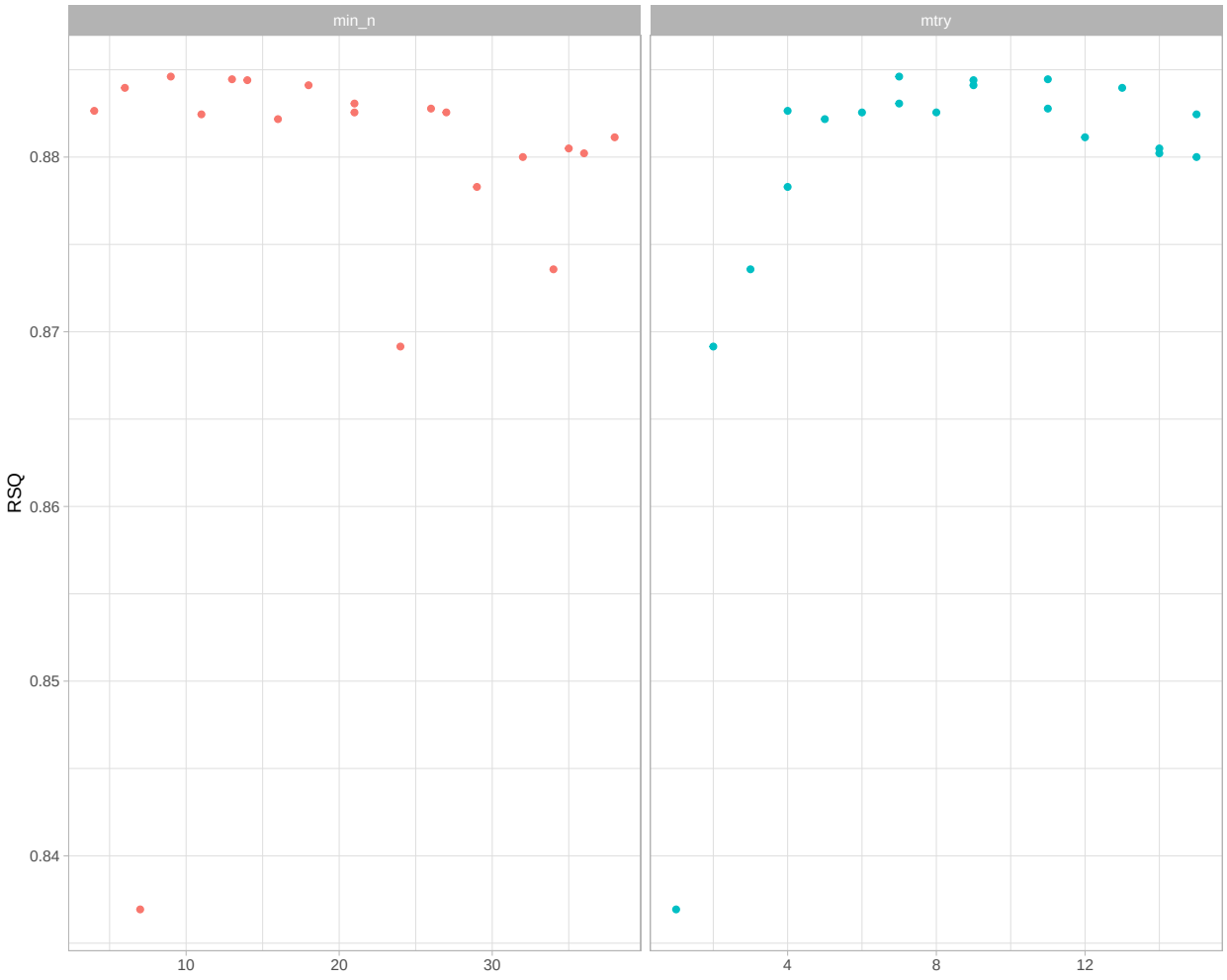
.metric	.estimator	.estimate
mae	standard	0.0655405

5.3 Random Forest model

Next a random forest specification is created, the mtry (the number of predictors to sample at each split) and min_n (the number of observations needed to keep splitting nodes) hyperparameters will be tuned. The engine used is **Ranger** and the mode is set as regression. The specification and recipe are then added to a random forest workflow.

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_other()
## * step_log()
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 1000
##   min_n = tune()
##
## Computational engine: ranger
```

To tune the hyperparameters the cross validation folds created earlier will be utilized and a tuning grid of 20 candidate parameter sets to be created automatically. The result of the first tuning can be plotted and shows that the min_n seems best at values under 10 and the mtry is best at values over 4.



Using this information a new regular grid is created, with the mtry and min_n being tuned again using a pre-defined range of values.

Table 6: Regular grid for tuning hyperparameters

mtry	min_n
4	2
8	2
12	2
16	2
20	2
4	4
8	4
12	4
16	4
20	4
4	6
8	6
12	6
16	6
20	6
4	8

mtry	min_n
8	8
12	8
16	8
20	8
4	10
8	10
12	10
16	10
20	10

The result of the re-tuning of the hyperparameters is shown below:

mtry	min_n	.config
8	4	Model07

Using these values the model is finalized, by updating the original specification created before tuning.

```
## Random Forest Model Specification (regression)
##
## Main Arguments:
##   mtry = 8
##   trees = 1000
##   min_n = 4
##
## Computational engine: ranger
```

Now that the specification is finalized the **vip** package is used to visualize the variable importance within the model. Variable importance shows the latitude and longitude i.e. the location of a property has very high importance, as do the grade, year a property was built and the square footage of the living area.

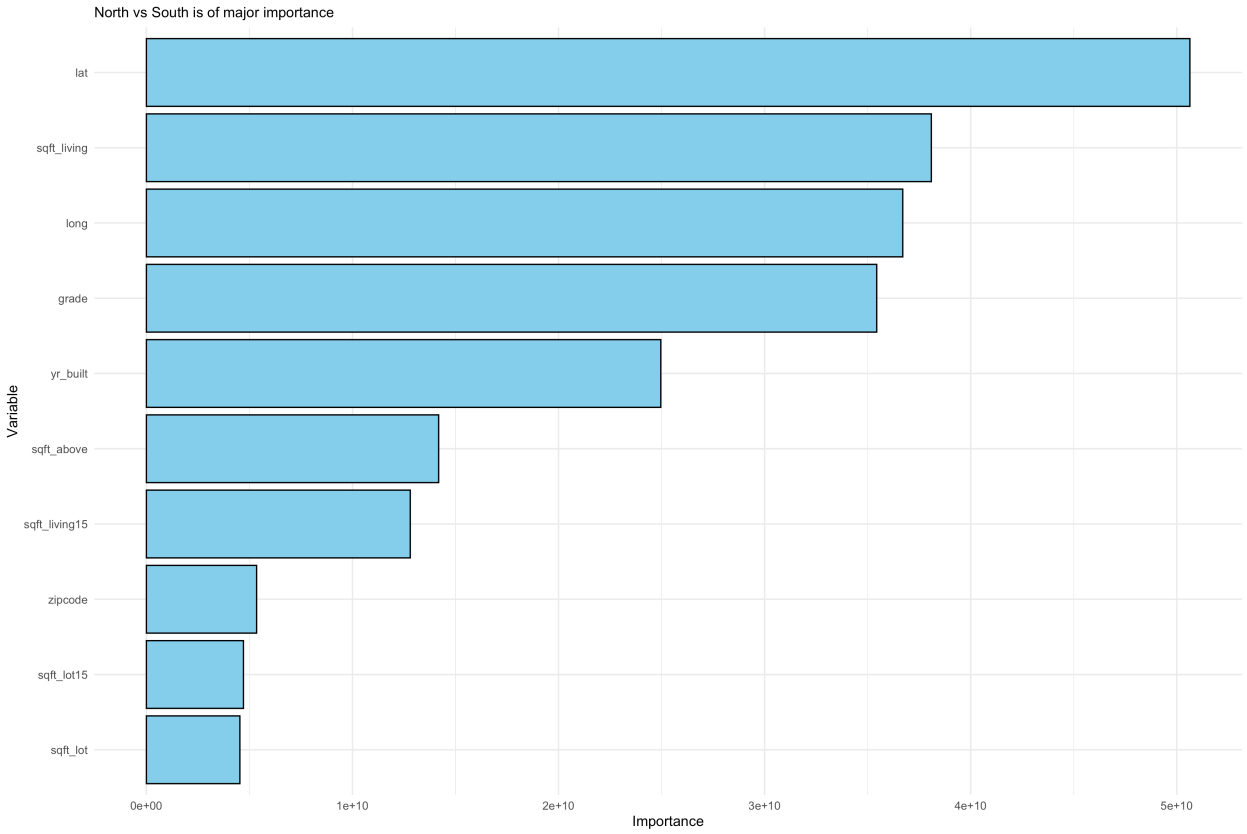


Figure 4: Random forest variable importance

A final workflow is created and then fitted one last time. The fit of the final model is on the entire training set and evaluated against the testing set, for this the original split is used.

Table 8: Random Forest Model Result Metrics

.metric	.estimator	.estimate
rmse	standard	0.0778687
rsq	standard	0.8848294

5.4 XGBoost model

As a final model to predict the housing prices, an XGBoost model will be created. This model is based on trees and allows for multiple hyperparameters to be tuned. The initial step is to create a specification.

```
## Boosted Tree Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 1000
##   min_n = tune()
##   tree_depth = tune()
##   learn_rate = tune()
##   loss_reduction = tune()
##   sample_size = tune()
##
## Computational engine: xgboost
```

Once the specification is created, a space-filling parameter grid is defined using latin hypercube sampling. In the workflow, a formula can be used due to the lack of pre-processing of data.

```
## == Workflow ====================================================================
## Preprocessor: Formula
## Model: boost_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors +
##     condition + grade + sqft_above + yr_built + zipcode + lat +
##     long + sqft_living15 + sqft_lot15 + year + month
##
## -- Model -----------------------------------------------------------------------
## Boosted Tree Model Specification (regression)
##
## Main Arguments:
##   mtry = tune()
##   trees = 1000
##   min_n = tune()
##   tree_depth = tune()
##   learn_rate = tune()
##   loss_reduction = tune()
##   sample_size = tune()
##
## Computational engine: xgboost
```

The hyperparameters are tuned using the workflow, specification and the cross validation folds that were created previously. The metrics from the tuning results are displayed below

There appear to be a number of parameter combinations that perform well, they are displayed in table below.

Table 9: The best combinations of hyperparameters for the XGBoost model

mtry	min_n	tree_depth	learn_rate	loss_reduction	sample_size	.config
15	39	12	0.0427096	0.0000000	0.3121462	Model13
10	24	11	0.0065669	0.0000038	0.1309916	Model29
2	7	2	0.0718031	0.0000071	0.9606421	Model20
4	31	13	0.0050549	0.0051816	0.3423769	Model17
12	26	4	0.0011427	0.0000198	0.3704763	Model15

The XGBoost workflow is finalized using the best parameter set.

```
## == Workflow ====================================================================
## Preprocessor: Formula
## Model: boost_tree()
##
```

```
## -- Preprocessor ----------------------------------------------------------
## price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors +
##     condition + grade + sqft_above + yr_built + zipcode + lat +
##     long + sqft_living15 + sqft_lot15 + year + month
##
## -- Model -----------------------------------------------------------------
## Boosted Tree Model Specification (regression)
##
## Main Arguments:
##   mtry = 15
##   trees = 1000
##   min_n = 39
##   tree_depth = 12
##   learn_rate = 0.0427096066700779
##   loss_reduction = 1.16809448232603e-08
##   sample_size = 0.312146165666636
##
## Computational engine: xgboost
```

Before running the finalized XGBoost model, an analysis of the variable importance is performed, the results of which are shown below. In the XGBoost model the longitude of a property is of less importance than in the random forest model.

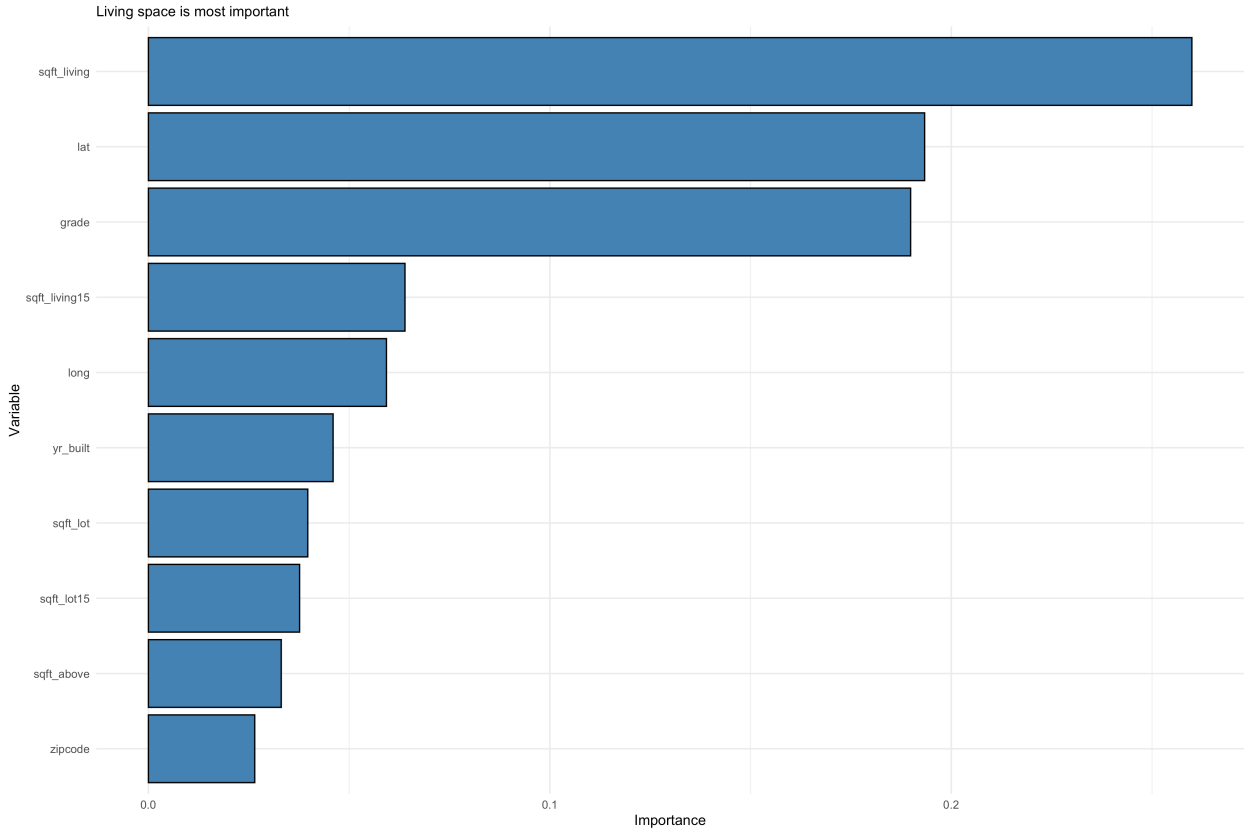


Figure 5: XGBoost variable importance

The finalized model is then run using the intital split of data with the evalutaion of the results performed against the test set.

Table 10: XGBoost Model Result Metrics

.metric	.estimator	.estimate
rmse	standard	0.0746896
rsq	standard	0.8925374

6 Results

Collating the results from the three different models into a single table, we see that the XGBoost model performs marginally better than the random forest model, with the linear regression model producing the worst results.

Table 11: Model Comparison Metrics

Method	RMSE	RSQ
Linear Regression Model	0.0905196	0.8421240
Random Forest Model	0.0786766	0.8810547
XGBoost Model	0.0746896	0.8925374

7 Conclusion

In conclusion the dataset enabled models to be trained that performed with a fairly high degree of accuracy. The linear regression model although it produces the forecasts with the highest margin of error, still enables very quick predictions to be made with very little set up involved. The random forest model was the most involved process to produce forecasts due to the tuning and re-tuning of the hyperparameters. The extra effort involved did have a payoff as the results improved, however the third method XGBoost had the best results with minimal effort required to build and tune the model and would be the recommended model methodology.