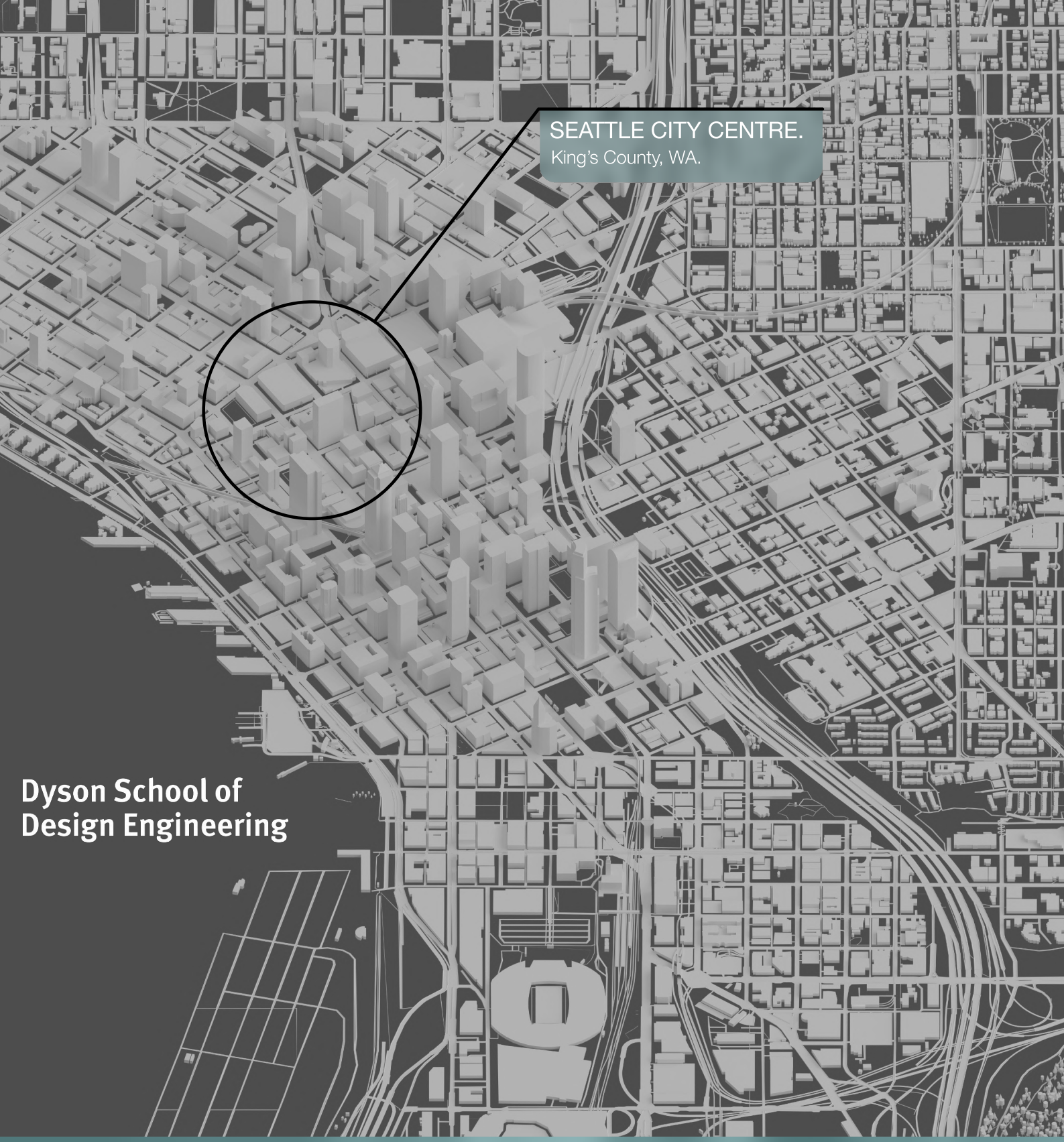


Dyson School of Design Engineering

PREDICTING PRICE BRACKET BASED ON HOUSE FEATURES IN KING COUNTY, WA.

DATA SCIENCE REPORT 2022 - Group 3.

Tanya AHMED Archie BOND Rohil J DAVE Ken MAH Ben LOVELL

PREDICTING PRICE BRACKET BASED ON HOUSE FEATURES IN KING COUNTY, WA

ABSTRACT

This report aimed to explain the variables that affect house prices in King County, USA through applying various machine learning methods, namely that of Random Forest, Linear Regression, Logistic Regression, Decision Trees, and Support Vector Machine. A predictive model was created that aimed to predict the how the price of a property in King County, USA, is affected by features such as location, size, view, and condition of the property. The analysis from the Decision Tree and Random Forest models found that the latitude had the biggest influence over predicting what price bracket a random sample falls into. Whereas the Linear and Logistic Regression models both determined that the square footage of living space for each house had the strongest correlation to the sale price of a house. This is a valid conclusion seeing that square footage of living was the most correlated variable on the Seaborn map set up during the data cleaning process. Support Vector Machine proved to be an optimal method to predict the house values using variables with reduced dimensionality.

1 INTRODUCTION

Many variables play a role in the decision when purchasing a property including location, bedroom or bathroom number, and the square footage of the living space to name a few. This report attempts to expose which machine learning method is the most accurate at predicting how house features influence property price. By doing this, it will expose what attributes can be assumed to drive sale prices of houses in King County. This would allow property investors to tailor their pitches to match the current trends of house prices in particular locations in order to increase business and meet customer demands.

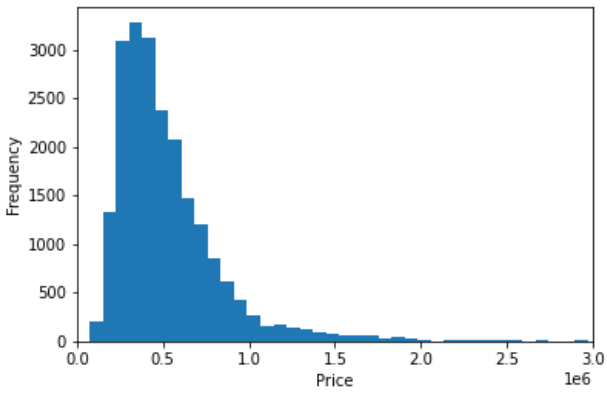


Figure 1 – A Histogram depicting the house prices for every house represented in the dataset, depicting that the data is not normally distributed (histograms for each variable are available in the appendix)

2 RELATED WORK

In a report detailing house price prediction in Iowa, there was high emphasis on discretizing the categorical data into sub-columns to prevent data loss by binarizing the data over ranges. Fuerthermore, tree- based machine learning algorithms were implemented as they do not requrie normal distribution, which for the IOWA dataset, was not present **(1).**

3 METHODOLOGY

3.1 Our Dataset

The original dataset contains the house prices for properties in King County, Washington, from May 2014 to May 2015. It is verified by official public records and sourced from Kaggle, consisting of 21 column variables for 21,612 houses in King County **(2).**

TABLE 3.1.1: ORIGINAL DATASET VARIABLES

Variable	Value Description
id	The unique ID for each home sold
date	Date of the home sale
price	Price of each home sold
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms, where .5 accounts for a room with a toilet but no shower
sqft_living	Square footage of the apartment's interior living space
sqft_lot	Square footage of the land space
floors	Number of floors
waterfront	A dummy variable for whether the apartment was overlooking the waterfront or not
view	An index from 0 to 4 of how good the view of the property was
condition	An index from 1 to 5 on the condition of the apartment
grade	An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design
sqft_above	The square footage of the interior housing space that is above ground level
sqft_basement	The square footage of the interior housing space that is below ground level
yr_built	The year the house was initially built
yr_renovated	The year of the house's last renovation
zipcode	What zip code area the house is in
lat	Latitude
long	Longitude
sqft_living15	The square footage of interior housing living space for the nearest 15 neighbours
sqft_lot15	The square footage of the land lots of the nearest 15 neighbours

Initially an exploratory analysis was conducted consisting of Ordinary Least Squares regression (OLS), each attribute was tested against house prices using the sci-kit toolbox. The variables' significance was determined by their p-values, shown in figure X ($p<0.001$ being significant, otherwise insignificant.

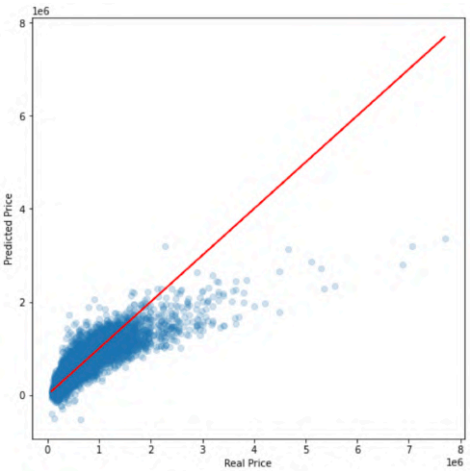


Figure 3.1.1– An OLS regression graph on predicted price against real price

OLS Regression Results

Dep. Variable:	price	R-squared:	0.700
Model:	OLS	Adj. R-squared:	0.700
Method:	Least Squares	F-statistic:	2797.
Date:	Wed, 29 Jun 2022	Prob (F-statistic):	0.00
Time:	00:18:02	Log-Likelihood:	-2.9460e+05
No. Observations:	21613	AIC:	5.892e+05
Df Residuals:	21594	BIC:	5.894e+05
Df Model:	18		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.939e+06	2.93e+06	2.366	0.018	1.19e+06	1.27e+07
bedrooms	-3.576e+04	1891.574	-18.907	0.000	-3.95e+04	-3.21e+04
bathrooms	4.116e+04	3253.218	12.651	0.000	3.48e+04	4.75e+04
sqft_living	110.3987	2.269	48.647	0.000	105.951	114.847
sqft_lot	0.1218	0.048	2.539	0.011	0.028	0.216
floors	6764.7566	3595.459	1.881	0.060	-282.608	1.38e+04
condition	2.625e+04	2351.639	11.164	0.000	2.16e+04	3.09e+04
grade	9.601e+04	2152.926	44.594	0.000	9.18e+04	1e+05
sqft_above	70.6885	2.253	31.368	0.000	66.272	75.106
sqft_basement	39.7051	2.646	15.005	0.000	34.519	44.892
yr_built	-2622.2522	72.653	-36.093	0.000	-2764.657	-2479.847
yr_renovated	19.6458	3.656	5.374	0.000	12.481	26.811
lat	6.024e+05	1.07e+04	56.115	0.000	5.81e+05	6.23e+05
long	-2.128e+05	1.32e+04	-16.179	0.000	-2.39e+05	-1.87e+05
sqft_living15	21.6256	3.447	6.273	0.000	14.869	28.383
sqft_lot15	-0.3964	0.073	-5.398	0.000	-0.540	-0.252
id	-1.289e-06	4.82e-07	-2.671	0.008	-2.23e-06	-3.43e-07
waterfront	5.827e+05	1.74e+04	33.567	0.000	5.49e+05	6.17e+05
view	5.303e+04	2140.594	24.774	0.000	4.88e+04	5.72e+04
zipcode	-582.2481	32.981	-17.654	0.000	-646.894	-517.603

Omnibus:	18406.018	Durbin-Watson:	1.990
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1881616.175
Skew:	3.571	Prob(JB):	0.00
Kurtosis:	48.149	Cond. No.	3.34e+22

Figure 3.1.2– OLS results table containing p values

3.2 Data Pre-processing and Cleaning

Unused variables were removed and the Seaborn Correlation Heatmap helped refine the data further. The following attributes were removed: the *id*, *date*, *sqft_above* and *zipcode* columns as these features were either nonpredictive attributes, displayed minimal correlation to the price or a very high correlation to another variable. Some categorical variables such as view, grade, and condition were binarized. A new binary variable was created from *yr_built* and *yr_renovated* called *is_new_or_renovated* which is true if one occurred after the year 2000. Furthermore, the type of the data and existence of null values within the data frame was inspected and modified accordingly.

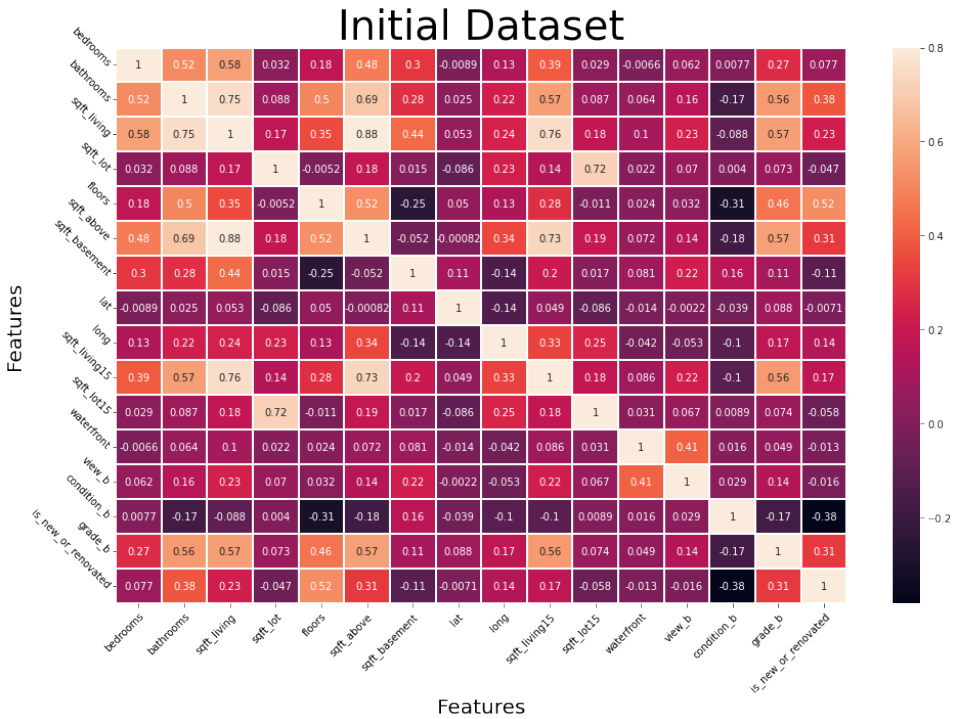


FIGURE 3.2.3: CORRELATION MATRIX INITIAL DATASET

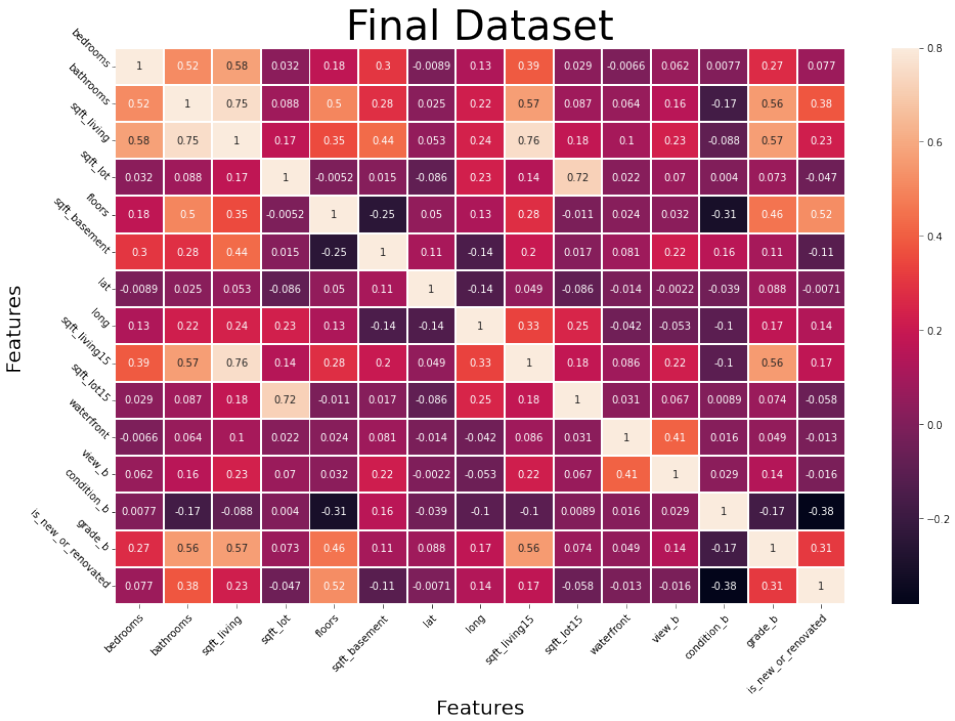


FIGURE 3.2.4: CORRELATION MATRIX FINAL DATASET

Training, validation and test sets were created with 80-10-10 split. Variables were normalised to have a mean=0 and a std=1.

3.3 Balancing the Data

price was converted to a categorical variable *Price Range* with 5 brackets that allowed the data to become well balanced.

TABLE 3.3.1: SAMPLE BREAKDOWN BY PRICE BRACKET

Identifier	Range	Number of Samples
A	<300000 USD	4570
B	300000–400000 USD	4269
C	400000–500000 USD	3721
D	500000–700000 USD	4730
E	>700000 USD	4323

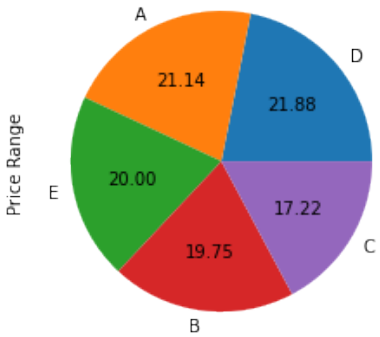


FIGURE 3.3.1: BALANCED DATA PRICE BRACKET COMPARISON

4.1 LINEAR REGRESSION (TANYA AHMED)

Which continuous attribute shows the strongest correlation with the price of a House in King County, WA?

4.1.1 Data Preparation

Linear regression was determined the most suitable method to answer this question as it is used to predict the relationship between *two* variables. A brief analysis was conducted on the correlations between two factors: the dependent variable (the price of a House in King County USA) and the independent variable (an attribute from the *X_train* data frame). As this method centres regression, the data frame must consist of continuous data, therefore attributes that were binarized during the data cleaning process were not considered. Separate data frames containing the raw price data were created (as opposed to the letter categorised data shown in the methodology). Training and testing data frames were used respectively where the training sets were used to determine the parameters of the model and the testing sets demonstrating the accuracy of the model constructed by the training set.

4.1.2 Method

The training set data frame, *X_train* consisted of approximately 17,000 data points. For each comparison, an attribute was extracted from the dataset as an array, and a scatter graph of the array against the price (which was logarithmically scaled) was plotted. Using the inbuilt linear regression 'predict', a graph of the chosen attribute against the predicted value of said attribute was plotted to depict the 'fit' of the data. Finally, the R – squared value, coefficients and intercepts were calculated, as an indication of the correlation strength.

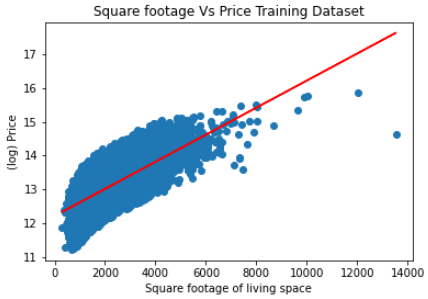


Figure 4.1.1 – Linear regression graph of square footage against log price using training data

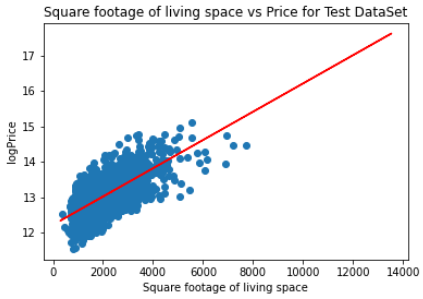


Figure 4.1.2– Linear regression graph of square footage against log price using test data

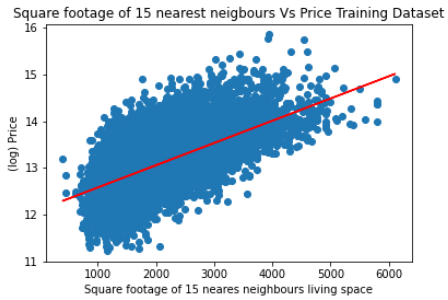


Figure 4.1.3– Linear regression graph of square footage of nearest 15 neighbours against log price using training data

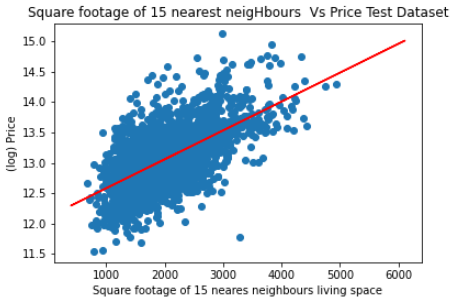


Figure 4.1.4– Linear regression graph of square footage of nearest 15 neighbours against log price using testing data

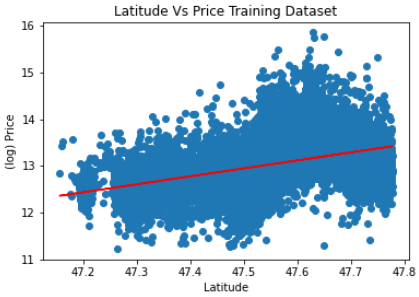


Figure 4.1.5– Linear regression graph of latitude against log price using training data

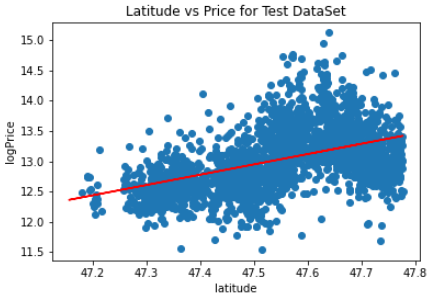


Figure 4.1.6– Linear regression graph of latitude against log price using test data

Linear regression graphs for each continuous variable and their corresponding R^2 values can be found in the appendix

4.1.3 Discussion

The method found that the attribute with the highest correlation was the square footage of living space for each house, with a coefficient of determination (R^2) of 0.4852105. Similarly, the living space square footage of the nearest 15 neighbours demonstrated a strong correlation (R^2 = 0.382683), followed by latitude with a (R^2 = 0.19977603). The correlation between price and square footage is self-explanatory: the larger the living space the more the house will sell for, however the comparison between the effects of longitude and latitude are more insightful. The R^2 value for longitude against price was 0.0022977, significantly lower than that of latitude, applying context to the scenario explains this disparity. As the latitude increases, houses are situated closer to the river, meaning there is now a desirable attribute of a 'waterfront' view – therefore houses are sold for higher prices.

4.2 LOGISTIC REGRESSION (BEN LOVELL)

What non – locational factor of a House in King County, WA is most influential in driving up the price into the critical threshold?

Due to the nature of the question, logistic regression was chosen as a suitable method as the price was defined into pass or fail – in context to being above the threshold or not, thus being considered a binary value. The 4 physical assets that were deciphered between were: *'number of bedrooms', 'number of bathrooms', 'number of floors'* and *'square feet of living space.'* These factors were chosen based on people's general priorities when looking for a house on the market **(3).**

4.2.1 Analysis and Results

To set up the model, there was some further pre-processing that was needed for binarizing the price, previously, the price was categorised into 5 distinct categories; A – E, based around the average price of a house in the King County Area **(4)**. To discretize this, the upper two bounds of price – D and E were deemed to be critical ($700,000 USD +) so all price datapoint values above this were converted to have a value of 1. All other values were converted to 0. However, this caused a large imbalance between the number of critical and non – critical samples as seen in *figure 3.2.1*. Therefore, the non – critical data was under sampled to even out the distribution resulting in a reduced number of total samples – from 21, 613 to 18106. Furthermore, feature scaling was carried out on the 'sqft_living' values as they were all far higher magnitudes than the rest of the features, carrying this out meant that we sped up the gradient descent process.

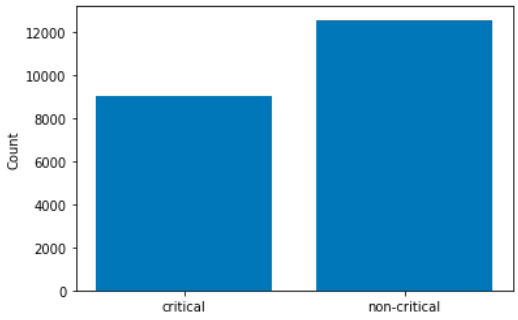


Figure 4.2.1 – unbalanced discritized price range

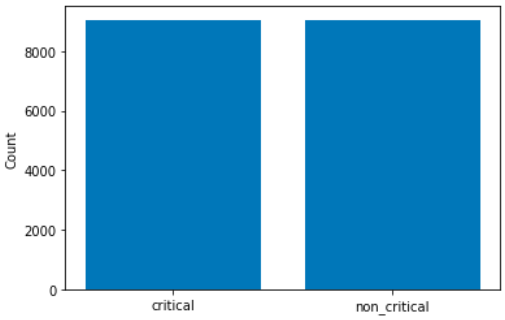


Figure 4.2.2 – even balance after non-critical undersampling.

In order to set up the training, validation and test sets, the data was split into 80%, 10% and 10% respectively. This gives the model a large section of the dataset to learn with, whilst still saving sufficient datapoints for testing. After undersampling the non – critical datapoints and creating the training split, there was 14484 samples for training and 1811 samples each for testing and validation.

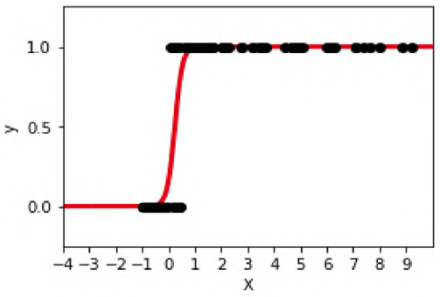


Figure 4.2.3 – training dataset logistic regression plot.

Now that the data had been prepared, the training set was plotted as a sanity check by looking at whether the values seemed to around the right places, such as high densities on integers (due to number of floors) as well as some minus values due to normalizing the data during pre-processing.

The plot confirmed that the regression was as expected, however, when obtaining the accuracy plot of the model, some further methods were carried out to try and improve the model.

An automatic forward selection function which produced a plot of the model's accuracy was carried out on the dataset. However, it was obvious from early on that the model lost accuracy with a low number of variables, after the third variable the validation line began to tend downwards from the training line. To check this, a manual forward selection process was carried out by adding other interior physical factors such as: 'sqft_basement' and 'sqft_lot', however, with both additions the number of false positives started to rise rapidly, hinting that the dataset was experiencing overfitting. The accuracy of the training model also fell from 0.74 to 0.55. Therefore original 3 independent variables were kept.

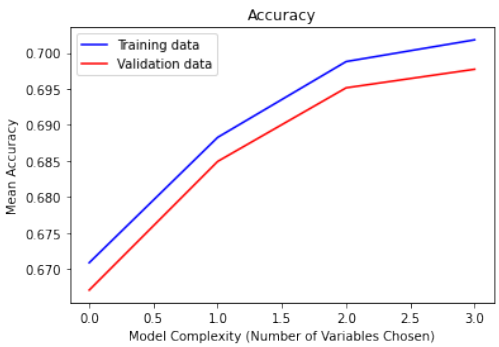
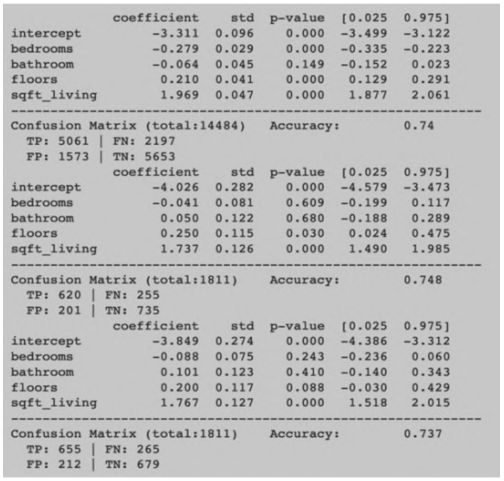


Figure 4.2.4 – automated model accuracy plot.

4.2.2 Discussion



```
             coefficient    std  p-value  [0.025  0.975]
intercept         -3.311  0.096    0.000  -3.499  -3.122
bedrooms          -0.279  0.029    0.000  -0.335  -0.223
bathroom          -0.064  0.045    0.149  -0.152   0.023
floors             0.210  0.041    0.000   0.129   0.291
sqft_living        1.969  0.047    0.000   1.877   2.061
--------------------------------------------------------
Confusion Matrix (total:14484)   Accuracy:         0.74
  TP: 5061 | FN: 2197
  FP: 1573 | TN: 5653
             coefficient    std  p-value  [0.025  0.975]
intercept         -4.026  0.282    0.000  -4.579  -3.473
bedrooms          -0.041  0.081    0.609  -0.199   0.117
bathroom           0.050  0.122    0.680  -0.188   0.289
floors             0.250  0.115    0.030   0.024   0.475
sqft_living        1.737  0.126    0.000   1.490   1.985
--------------------------------------------------------
Confusion Matrix (total:1811)    Accuracy:         0.748
  TP: 620 | FN: 255
  FP: 201 | TN: 735
             coefficient    std  p-value  [0.025  0.975]
intercept         -3.849  0.274    0.000  -4.386  -3.312
bedrooms          -0.088  0.075    0.243  -0.236   0.060
bathroom           0.101  0.123    0.410  -0.140   0.343
floors             0.200  0.117    0.088  -0.030   0.429
sqft_living        1.767  0.127    0.000   1.518   2.015
--------------------------------------------------------
Confusion Matrix (total:1811)    Accuracy:         0.737
  TP: 655 | FN: 265
  FP: 212 | TN: 679
```

Figure 4.2.5 – model summaries.

As shown by the results of the three models in figure 3.2.4, the accuracy of the model fluctuated around 0.741 across the three sets. Furthermore, the number of false negatives and false positives had reduced since past models. Meaning there were less type I errors and type II errors.

From looking at the variable coefficients in the model summary, we can see that the sqft_living has the largest impact on pushing the price into the upper threshold with a coefficient average of 1.78. Whilst the number of bedrooms, interestingly, inversly affected the chance of the price being in the critical threshold. Furthermore, in the training set the P – Values < 0.005 for sqft_living, showing that it was significant.

To further validate the model, I took out the highly influential 'sqft_living' variable from the logistic regression set and printed the model summary again. The accuracy of the model dropped to 0.51, suggesting that this variable does indeed have a strong correlation to driving up the price into the critical threshold. This makes sense as the larger the amount of living space relates to the size of the property in the most scalable value as your are categorically able to say one house is bigger than the other when marketing properties, this means it is far easier to sell houses for more when the living space is larger **(5).** This idea is further supported by the far weaker correlation of floors and bathrooms to driving up the price into the critical threshold, due to them being non – scalable as houses with many floors and bathrooms aren't necessarily larger than others, making them harder to justify putting them at a high price point.

4.3 DECISION TREES (ARCHIE BOND)

4.3.1 Analysis and results

The decision trees machine learning method uses an important measure called Gini impurity which calculates the probability of a certain feature being incorrect when selected randomly. In other words, it shows how well the decision tree was split. It has a maximum and minimum value of 1 and 0 respectively. Ideally, the Gini impurity should be as close to 0 as possible to have a better probability of correctly choosing a sample.

Firstly, the question was defined as "What feature has the most influence over predicting the sale price of a house in King County, USA?". The data was cleaned and balanced as stated in the introduction section, and the decision tree used 15 different attributes. It should be noted that the decision tree stops when a new tree branch would result in an impurity decrease of less than the specified minimum impurity decrease or when the maximum depth is reached. Therefore, these values had to be defined. The maximum depth was plotted against the model accuracy which can be seen in figure 4.3.1 below.

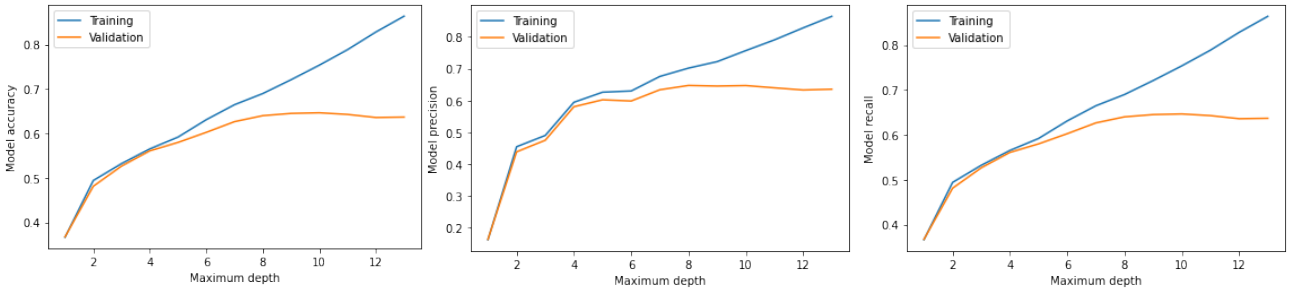


FIGURE 4.3.1: PLOT OF ACCURACY OF TRAINING AND VALIDATION DATA AGAINST THE MAXIMUM DEPTH

From the graphs, it can be seen that the accuracy of the validation and training data begins to split apart by a considerable amount at a maximum depth of around 8. This shows that although the training data set's accuracy continues to increase, the validation starts to plateau which would infer that the data begins overfitting. This is not desirable as the model would not be able to achieve generalisation meaning that it cannot be applied to unseen data. As a result, the maximum depth for the decision tree was chosen to be 8 as this was a good enough balance between accuracy and having reliable data. The same method was used with minimum impurity decrease and the graphs can be seen below. Therefore, the minimum impurity decrease was chosen to be 0.001 as this would give the most accurate model.

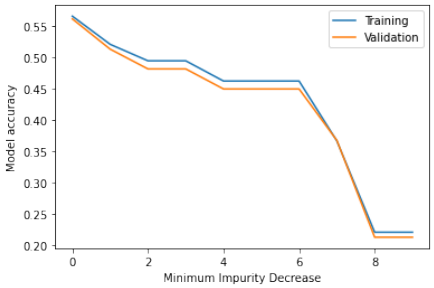


FIGURE 4.3.2: PLOT OF TRAINING AND VALIDATION ACCURACY AGAINST MINIMUM IMPURITY DECREASE

Using the above parameters, the decision tree was plotted which can be seen in Figure 4.4.3 (sections in appendix).

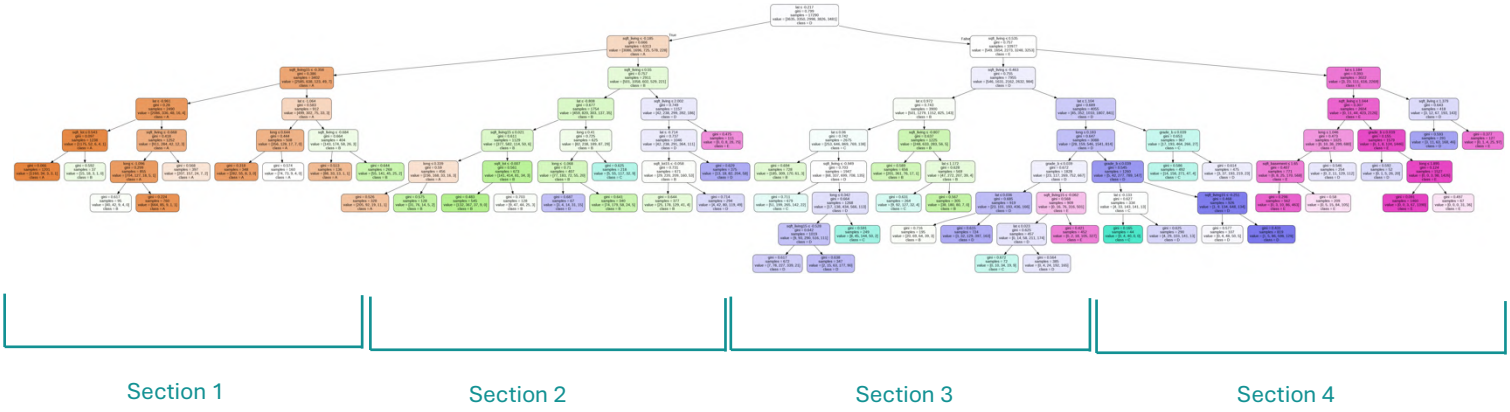


FIGURE 4.3.3: DECISION TREE WITH A MAX DEPTH OF 8 AND MIN IMPURITY DECREASE OF 0.001

The sale price for each house was categorized into 5 different bins A-E as mentioned before, shown by the class in each node. At the root node, the randomly selected variable is predicted to be in bin D and has a Gini of 0.799, showing that this has a 79.9% chance of being incorrect. The tree then splits based

on the lat variable showing that it is the most correlated. If lat ≤ -0.217 then the data is split by sqft_living ≤-0.185 but if it is bigger than -0.217 it is split by sqft_living ≤0.535. This continues until the Gini becomes 0 or the maximum depth/minimum impurity decrease has been met. In section 1 of the tree for example, the far most left orange box has a Gini of 0.066 with the number of samples meeting the above requirements being 1201. This means that there is around a 99% chance that the selected sample is correctly predicted to be in the A class (1160 samples). Though some nodes showed a small Gini impurity, others did not which is reflected in the accuracy, precision and recall values shown below in the table.

Data Set	Accuracy	Precision	Recall
Validation	61.9%	62.4%	61.9%
Test	62.2%	63.2%	62.2%

TABLE 4.3.1: SHOWING THE ACCURACY, PRECISION AND RECALL VALUES FROM THE DECISION TREE

In relation to the question, some more code was carried out to determine which feature influences the sale price of a house the most. This was done by eliminating one feature and then restating the accuracy of the data without this feature. The process was repeated with each feature by first removing it, computing the accuracy on the test data, adding it back each time and then repeating. It was concluded from this analysis that when the lat variable was removed, the accuracy decreases by the maximum of 26.3%.

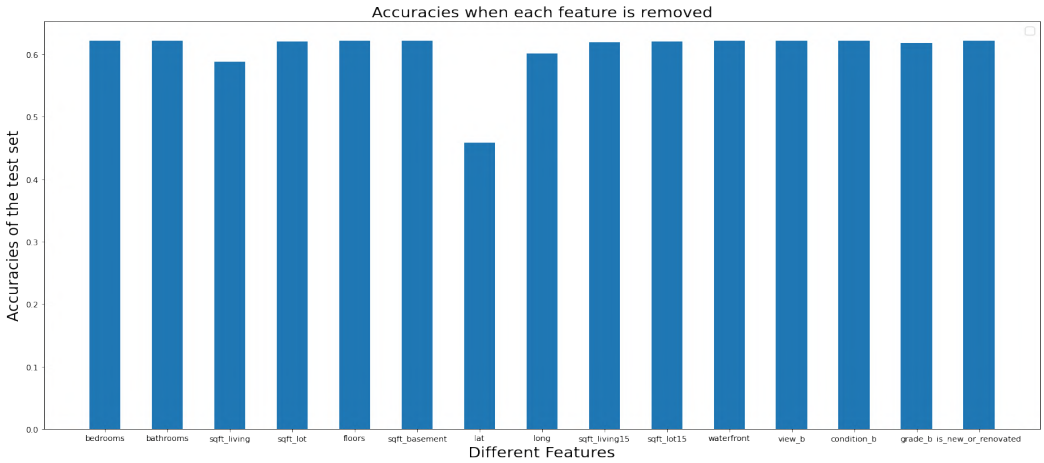


FIGURE 4.3.4: THE ACCURACY OF THE TEST SET WHEN EACH FEATURE IS REMOVED ONE BY ONE

4.3.2 Discussion

Firstly, the limitations of the model should be considered. To answer the question, a decision tree is appropriate as it allows you to evaluate which option has the best outcome and makes them easily comparable. They can deal with missing data automatically and analyse continuous/ categorical variables. However, it should also be noted that it cannot predict continuous variables (hence why the different price categories were created). Furthermore, decision trees become complex when there is a considerable amount of data used. If this particular data set was any bigger, the tree would grow a lot of nodes leading to longer computation times, complexity, and overfitting.

Although the results in table 4.3.1 are not as close to 100% as desired, the values are reasonable enough to consider the method reliable. The results for accuracy, precision and recall are all similar to one

another within their respective data set, with none varying by more than 1%. It should also be noted that the accuracy and recall values are the same suggesting a viable and balanced model.

As figure 4.3.4 shows, the lat feature clearly has the most influence on predicting the price category that a random sample falls into, followed by sqft_living and long. Without the lat feature, the accuracy of the model substantially decreases to 45.8%. This suggests that there is a strong correlation between price and latitude, which when looking at a map of King County Washington adds validity to the prediction; the west coast of King County is placed along the shore meaning that house prices are likely to increase due to coastal views, weather etc. Additionally, this area is known for stormy weather which would naturally increase house price due to flood insurance etc **(6).**

4.4 RANDOM FOREST (ROHIL J DAVE)

A Random Forest is an ensemble ML method that creates a predictive tree-based model that outputs a single result from a combination of multiple decision trees. Each tree is created by randomly selecting samples and features. First, rows are randomly selected (with replacement) and then columns (without replacement) which select the best split. This is repeated until the desired depth of the tree is reached or stopping criteria is met. The process continues until several trees form a forest which then compete against each to yield the best model.

Random Forests have several advantages over a single decision tree. The risk of overfitting is reduced since prediction error and variance are lowered when uncorrelated trees are averaged. Additionally, Random Forests are excellent at predicting the importance, or contribution, of a specific feature by measuring the decrease in model accuracy when that variable is removed from the data **(7)**. Hence, a Random Forest is apt for answering the question: **Which house feature is the most important (contributes the most) to predicting the price bracket a house that was sold in King County falls in?**

4.4.1 Analysis and Results

The problem is a Random Forest classification that will classify the price bracket the house sold for based on its features. This will inherently provide more accuracy than a stand-alone decision tree as a majority vote will be the basis of the prediction and not all decision making will be relied on one tree.

Two core hyperparameters, max_depth and min_sample_split, were selected based on reasonable performance metric scores that strike a balance between the training and validation datasets. These parameters both determine when a tree stops splitting. max_depth determines how many nodes will split until the end of the tree is reached. min_sample_split determines if a node of a tree will continue to split based on the number of samples left in the node.

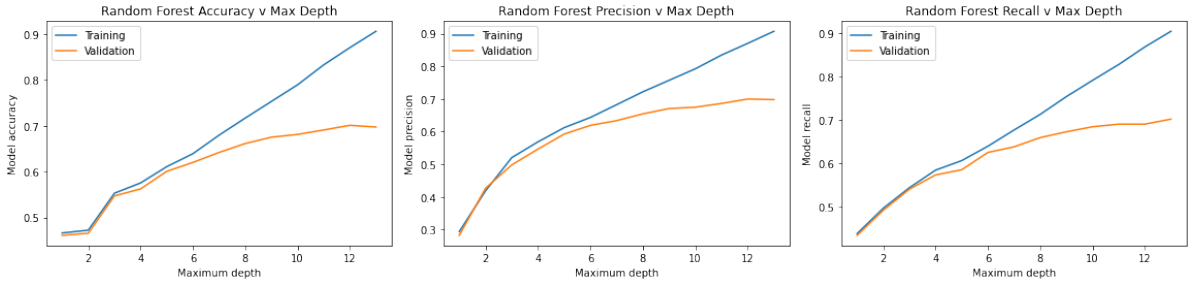


FIGURE 4.4.1: PLOTS OF PERFORMANCE METRICS VS MAX DEPTH FOR TRAINING AND VALIDATION SETS

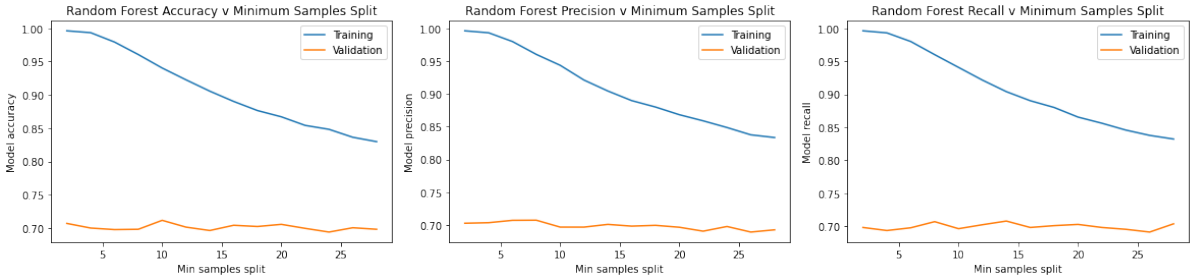


FIGURE 4.4.2: PLOTS OF PERFORMANCE METRICS VS MIN SAMPLES SPLIT FOR TRAINING AND VALIDATION SETS

Figure 4.4.1 shows that the training and validation scores begin the separate largely around a max_depth of 6. The validation set very closely follows the training set in accuracy, precision, and recall up until that point. However, to ensure a good enough training accuracy and good enough validation accuracy a slightly larger max_depth was chosen at 8. Since the accuracy starts to plateau soon after, this was justified. Figure 4.4.2 shows that training set performance metrics decrease as the minimum number of samples increases while the validation set stays around 0.70 all the way to 30 minimum samples. Since the question values accuracy most, a min_samples_split of 10 was chosen as it yields the highest validation accuracy score while maintaining high training accuracy.

With the parameters set, the full Random Forest was constructed. Detailed sections of Figure 3.3.3 are in the Appendix.

FIGURE 4.4.3: FINAL EXAMPLE RANDOM FOREST TREE (MAX_DEPTH = 8, MIN_SAMPLES_SPLIT = 10)

For all three datasets, the accuracy score and recall score are the same and the precision score is only slightly larger. This indicates the model's ability to predict positives is nearly the same to its ability to predict negatives, being balanced in its predictive nature. As seen in Table 4.4.1, the scores are closest in the test set which has an accuracy of 66.3%. It is noted that the test set scores are higher than that of the validation set albeit by less than 0.01 difference.

TABLE 4.4.1: FINAL MODEL PERFORMANCE METRICS FOR TRAINING, VALIDATION AND TEST SETS

Dataset	Accuracy	Precision	Recall
Training	0.706	0.711	0.706
Validation	0.656	0.662	0.656
Test	0.663	0.665	0.663

To identify the most contributing feature to predicting the sale price bracket of the house, a function was created that iterates through the X variables one by one to evaluate the model's accuracy of the test set when each respective feature is excluded. Each iteration assembles the confusion matrix of the specific Random Forest model and assesses the accuracy score based on the predicted values from the test set. The model that results with lowest accuracy score reveals which feature is the most important. Table 4.4.2 details the model accuracy according to each feature removal.

TABLE 4.4.2: MODEL ACCURACY RELATED TO SPECIFIC FEATURE REMOVAL

Feature Excluded	Accuracy
bedrooms	0.665
bathrooms	0.671
sqft_living	0.642
sqft_lot	0.666
floor	0.665
lat	0.514
long	0.647
sqft_living15	0.655
sqft_lot15	0.673
waterfront	0.670
view_b	0.667
condition_b	0.667
grade_b	0.654
is_new_or_renovated	0.670

The model accuracy is the lowest at 51.4% when the variable lat is excluded from the data. This shows that the latitude location of the house is the most important feature in predicting the price bracket the house sold for.

4.4.2 Discussion

Table 4.4.2 shows that lat is the only variable which, when removed, causes the model accuracy to decrease by about 15%. All other features have a model accuracy within ±2% of the final model's test set accuracy, showing the effect of lat is much stronger than the other variables. The map of King County, WA makes the effect of latitude clear to understand. West King County includes houses located closer to the water, on the island, and near port of Seattle and Tacoma. As these locations are more desirable, the sale price would be higher. The model suggests that properties more west in the county tend to have higher prices that they were sold at. A variety of other features such as view and waterfront may drive up the individual price a house sold for, but the latitude is the key identifier in predicting a price bracket from the established price ranges A through E.

As with all models, the Random Forest analysis comes with some limitations. There may be other features contributing strongly to the price a house sold for such as popularity of certain locations in King County. This could be influenced by community infrastructures, school districts, and insurance measures of safety **(6)**. Furthermore, a more detailed price bracket breakdown with more even differences between ranges could be created and then sampled for a more balanced dataset. This will help identify and differentiate clearer the most important feature for houses that sold for over 1000000 USD.

It is noted that the Random Forest model improved accuracy over a single Decision Tree by roughly 4%. The model is correcting some of the overfitting created by the Decision Tree model and yields better performance metrics. Although, the Decision Tree shows the effect of latitude to a similar degree.

4.5 SUPPORT VECTOR MACHINE (KENNARD S MAH)

4.5.1 Data Preparation

SVM is a linear model used for classification problems. It is utilised to find the best separation between the classes and fit the error instances within a threshold while limiting margin violations. Therefore, the SVM method is suited to answer the question: **how can we predict the classification of the property value based on its space, view, coast side, and condition?** The data set was organised separately to meet the criteria to solve this question. After basic descriptive analysis -- resulting in a mean property value of approximately 540,088 -- a new column was added to represent whether the property was within the 50% percentile. Prices above the mean value were labelled as 'Upper-cost' with a Boolean value of 1; prices were otherwise categorised as 'Lower-cost' with a Boolean value of 0. Select principal components with low error and variables with reduced dimensional features were used as inputs to run the model. Based on the selected question, the following independent variables were used in the analysis: *sqft_living, sqft_lot, waterfront, view_b, condition_*b and *is_new_or_renovated*. Values that were not utilised were dropped from the temporary data set.

4.5.2 Analysis

The function of the SVM is to input the training data and parameters (C-value, Kernel, Gamma) to generate a hyperplane that can efficiently sort the property values.

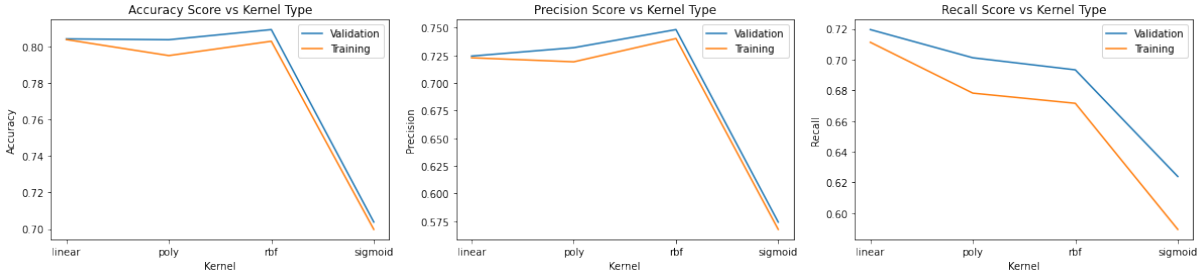


Figure 4.5.1 : Plots of varying types of kernel against Accuracy, Precision, and Recall for Validation and Training Sets

The kernel determines the shape of the hyperplane that divides the data. To select the type of kernel that gives the highest accuracy, SVM was ran with the values of c = 2.75 and default gamma = 'scale'. With these values, it appears that the 'linear' kernel yields the best overall results; in the following simulations, therefore, we carried out SVM with the 'linear' kernel type.

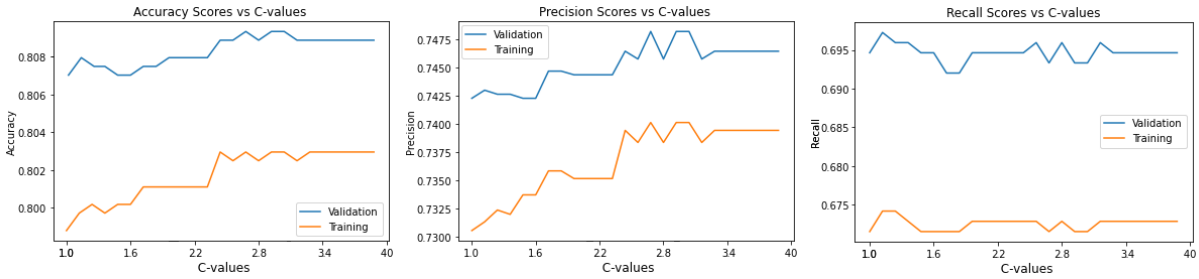


Figure 4.5.2: Plots of C-values against Accuracy, Precision, and Recall for Validation and Training Sets

The C-value is the degree of optimisation that the SVM must meet. The data frame was tested in the range of 1 to 4 and increments of 0.025. c = 2.70 produced the highest scores apart from the recall

score. However, the recall score at the given C-value is a local maximum as displayed in the figure above. Hence, c = 2.70 was selected moving forward.

The gamma parameter defines the level of influence a single training example reaches. Low gamma can cause the model to be too constrained to capture the complexity of the data (under classification); whereas high gamma can cause the radius of the area of influence to be the support vector itself and make it difficult to prevent overfitting by adjusting the C-value (over classification.) The ideal setting for gamma in this scenario is 'scale.' 'scale' has an inverse correlation to the number of samples and variance and yielded the best results when testing with various gamma values.

4.5.3 Discussion

Dataset	Accuracy	Precision	Recall
Training	0.7870	0.8042	0.8039
Validation	0.7225	0.7243	0.7227
Test	0.6896	0.7195	0.7113

Figure 4.5.3: Metrics of Performance for SVM Classifier

The resulting metrics of performance for the SVM classifier with the hyperparameters **(C-value = 2.70, kernel = 'linear', gamma = 'scale')** is as shown. The SVM classifier is the most suited machine learning model to predict house values given its higher accuracy and minimal joint lowest difference in accuracy between the test and validations set at 0.003. Given the computation complexity of SVM and 17290 data values, the time required to test and train the model is greater than other models. However, given that the time to train the model was not a concern during the analysis, it is safe to assume that the SVM model can be utilised for determining prices in other regions as well as King County, Washington. However, when broadening the scope and dealing with larger datasets, other ML models should be investigated.

5 CONCLUSION

Linear regression demonstrated that the attribute with the strongest correlation to price was the square footage of living space available, alongside latitude which showed a significant relationship.

Logistic regression also revealed square foot of living space to be the most influential non-locational factor in driving the price point into the critical threshold, supported by the linear regression model.

Decision Tree and Random Forest both reveal that latitude is the most important feature in predicting the price bracket a house sold in King County falls in with Random Forest having a slightly better accuracy score.

Support Vector Machine has shown to be the optimal method to predict the property values given its ability to achieve high accuracy with limited sample learning and nonlinear regression problems. When analysing property values in other areas, it is crucial to determine the parameters for learning and generalisation. Furthermore, there are limitations with analysing larger datasets.

Addressing the limitations discussed in each model will help to increase accuracy and viability of the predictions.

REFERENCES:

(1) Ozancan Ozdemir – House Price Prediction Using Machine Learning: A Case in Iowa. Feb 2022. https://www.researchgate.net/publication/358573294_House_Price_Prediction_Using_Machine_Learning_A_Case_in_Iowa

(2) HARLFOXEM – House Sales in King County, USA. 2016 https://www.kaggle.com/datasets/harlfoxem/housesalesprediction?datasetId=128&searchQuery=logistic

(3) Gloria RUSSELL – 10 key features to consider when buying a house. May 2021 https://homeia.com/10-important-features-to-consider-when-buying-a-house/

(4) Seattle Met Staff – The most Eye Popping King's County Real Estate numbers. Jan 2022. https://www.seattlemet.com/home-and-real-estate/2022/01/how-expensive-is-a-house-in-seattle-bellevue-redmond-washington

(5) JD Esajian, What are the biggest factors in determining property value? 2021. https://www.fortunebuilders.com/what-are-the-biggest-factors-in-determining-property-value/

(6) Redfin Agents – King County Housing Market. Jan 2020. https://www.redfin.com/county/118/WA/King-County/housing-market#agent-insights

(7) IBM Cloud Education – Random Forest. Dec 2020. https://www.ibm.com/cloud/learn/random-forest

APPENDIX

A – Data Pre-processing, Cleaning, and Balancing

TABLE A.1: FINAL DATASET X VARIABLES (STANDARDISED IN PREPROCESSING)

Variable	Value Description
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms, where .5 accounts for a room with a toilet but no shower
sqft_living	Square footage of the apartment's interior living space
sqft_lot	Square footage of the land space
floors	Number of floors
waterfront	A dummy variable for whether the apartment was overlooking the waterfront or not
view_b	*view* made binary; 1 if rating greater than 2, 0 if less than or equal to 2
condition_b	*condition* made binary; 1 if greater than 3, 0 if less than or equal to 3
grade_b	*grade* made binary; 1 if greater than 7, 0 if less than or equal to 7
sqft_basement	The square footage of the interior housing space that is below ground level
lat	Latitude
long	Longitude
sqft_living15	The square footage of interior housing living space for the nearest 15 neighbours
sqft_lot15	The square footage of the land lots of the nearest 15 neighbours
is_new_or_renovated	Combined *yr_built* and *yr_renovated* and made binary; 1 if built OR renovated after 2000, o if neither are tru

```
# import all libraries used
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
import sklearn.tree as tree
import graphviz
import pylab
```

```
# read original dataset and create dataframe object
hdf = pd.read_csv('kc_house_data.csv')
hdf = pd.DataFrame(hdf)

hdf.info()
```

```
# remove variables we do not plan to use
hdf = hdf.drop(columns=['id','date','zipcode'])

# separate continous and categorical variables
continuous_hdf = hdf[['price','bedrooms','bathrooms','sqft_living','sqft_lot','floors','sqft_above','sqft_basement','lat','long','sqft_living15','sqft_lot
categorical_hdf = hdf.drop(columns=['price','bedrooms','bathrooms','sqft_living','sqft_lot','floors','sqft_above','sqft_basement','lat','long','sqft_livin

# restitch dataset but now with all continuous variables together
hdf = pd.concat([continuous_hdf, categorical_hdf], axis=1)

# hdf.to_csv('kc_house_reorganised.csv')
```

```
# make variables view, grade, condition binarised
hdf['view_b'] = hdf['view'] > 2
hdf['view_b'] = hdf['view_b'].astype(int)
hdf['condition_b'] = hdf['condition'] > 3
hdf['condition_b'] = hdf['condition_b'].astype(int)
hdf['grade_b'] = hdf['grade'] > 7
hdf['grade_b'] = hdf['grade_b'].astype(int)

# function to combine year built and year renovated into one binarised variable to see if house has been updated or created after 2000
def label_new(row):
    if row['yr_renovated'] > 2000 :
        return '1'
    elif row['yr_built'] > 2000 :
        return '1'
    else:
        return '0'

hdf['is_new_or_renovated'] = hdf.apply(lambda row: label_new(row), axis=1)
hdf['is_new_or_renovated'] = hdf['is_new_or_renovated'].astype(int)

# print(hdf['is_new_or_renovated'])

# remove original unbinarised data columns
hdf = hdf.drop(columns=['view','grade','condition','yr_built','yr_renovated'])

# create a copy of the dataframe with price column removed
price_pd = pd.DataFrame(hdf.copy().pop('price'))

# convert price into a categorical variable with 10 appropriate ranges based on typical housing prices USD
price_lab = ['A', 'B', 'C', 'D', 'E']
price_pd['price_range'] = pd.cut(price_pd.price, [0, 300000, 400000, 500000, 700000, 8000000], labels=price_lab)
print(price_pd)
```

```
# dataframe copy for correlation matrix
hdf_corr = hdf.copy()

# create attributes axis for correlation matrix
columns = list(hdf_corr.columns)
y_col = columns.pop(0)
y = hdf_corr[y_col].to_numpy()
X = hdf_corr[columns].to_numpy()

scaler = MinMaxScaler()
hdf_corr[columns] = scaler.fit_transform(hdf_corr[columns])

# remove price to see correlations between other attributes
hdf_corr = hdf_corr.drop(['price'], axis=1)

# defines and creates initial correlation matrix
corrmat = hdf_corr.corr()
fig, ax = plt.subplots(figsize=[16,10])
sns.heatmap(corrmat,vmax=0.8, annot=True, linewidths=1, linecolor='white')
ax.set_xlabel('Features', fontsize = 20)
ax.set_ylabel('Features', fontsize =20)
plt.title("Initial Dataset", fontsize = 40)
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
plt.setp(ax.get_yticklabels(), rotation=-45, ha="right", rotation_mode="anchor")
plt.show()
```

```
# There is a high correlation between sqft_living and sqft_above (0.88) so it was removed to reduce skew
hdf_corr = hdf_corr.drop(['sqft_above'], axis=1)
# defines and creates final correlation matrix, where correlations of magnitude greate than 0.8 were removed
corrmat = hdf_corr.corr()
fig, ax = plt.subplots(figsize=[16,10])
sns.heatmap(corrmat,vmax=0.8, annot=True, linewidths=1, linecolor='white')
ax.set_xlabel('Features', fontsize = 20)
ax.set_ylabel('Features', fontsize =20)
plt.title("Final Dataset", fontsize = 40)
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
plt.setp(ax.get_yticklabels(), rotation=-45, ha="right", rotation_mode="anchor")
plt.show()

corrT = hdf_corr.corr(method = 'pearson').round(4)
```

```
# dataframe copy for preprocessing
hdf_prepro = hdf.copy()
# add categorical price variable
hdf_prepro['Price Range'] = price_pd['price_range']
# remove columns no longer needed
hdf_prepro = hdf_prepro.drop(['sqft_above', 'sqft_basement', 'price'], axis=1)
```

```
# see count in each price bracket
hdf_prepro['Price Range'].value_counts()
```

```
D    4730
A    4570
E    4323
B    4269
C    3721
Name: Price Range, dtype: int64
```

```
# visualise data split of price brackets to verify balance
hdf_prepro['Price Range'].value_counts().plot.pie(autopct = '%.2f')
```

B – HISTOGRAMS

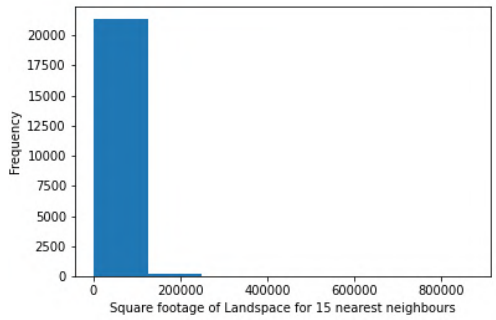


FIGURE B.1: SQFT_LOT15 HISTOGRAM

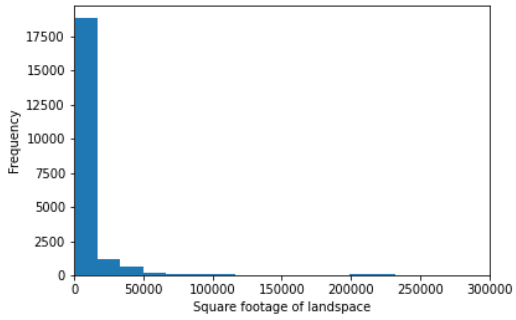


FIGURE B.2: SQFT_LOT HISTOGRAM

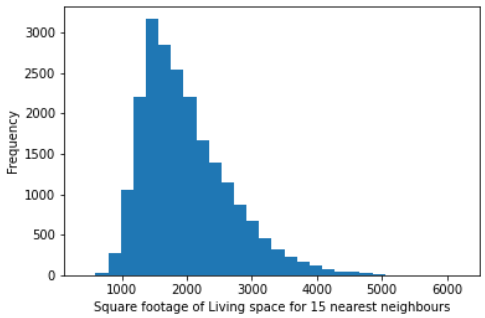


FIGURE B.3: SQFT_LIVING15 HISTOGRAM

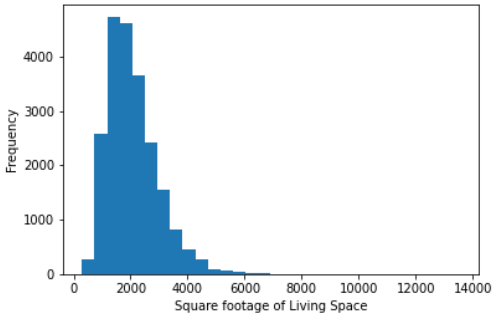


FIGURE B.4: SQFT_LIVING HISTOGRAM

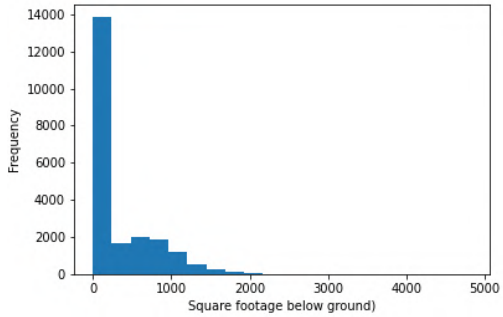


FIGURE B.5: SQFT_BASEMENT HISTOGRAM

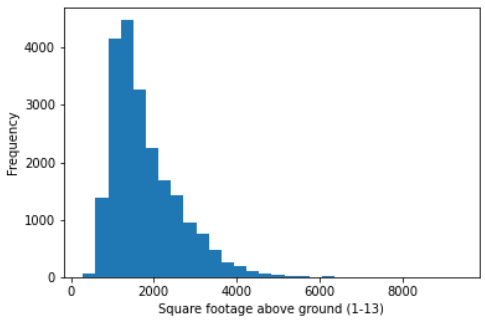


FIGURE B.6: SQFT_ABOVE HISTOGRAM

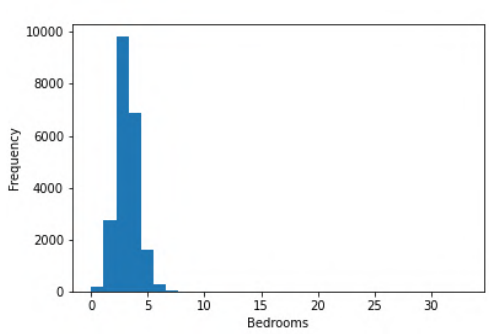


FIGURE B.7: BEDROOMS HISTOGRAM

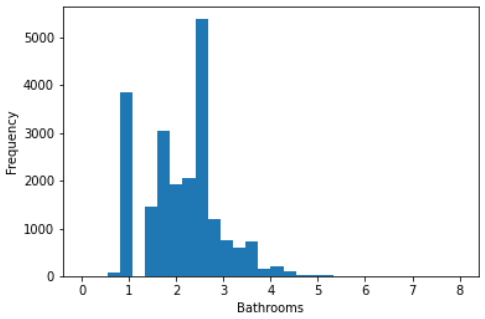


FIGURE B.8: BATHROOMS HISTOGRAM

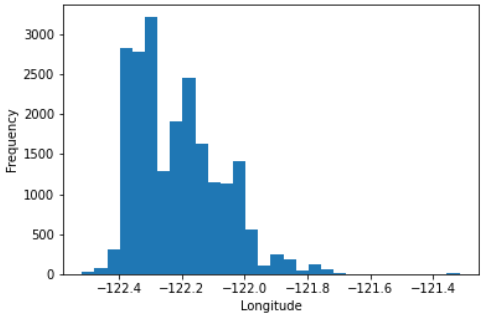


FIGURE B.9: LONGITUDE HISTOGRAM

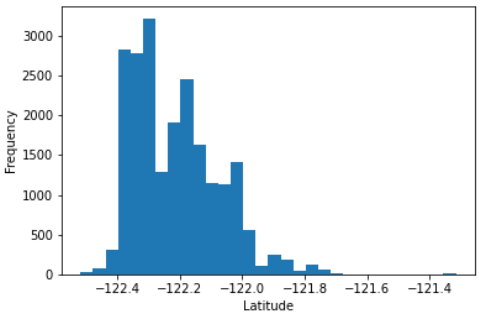
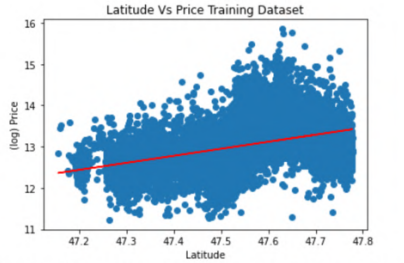


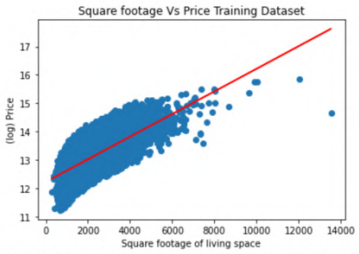
FIGURE B.10: LATITUDE HISTOGRAM

C – Linear Regression (Tanya Ahmed)

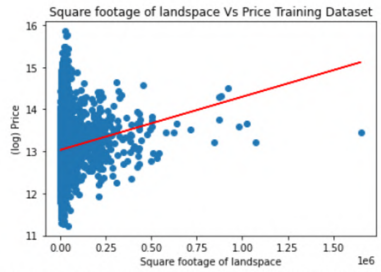
Linear regression graphs for continuous variables



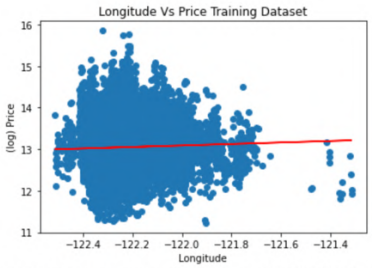
Coefficient of Determination 0.19977603521621623



Coefficient of Determination 0.4852104645433002



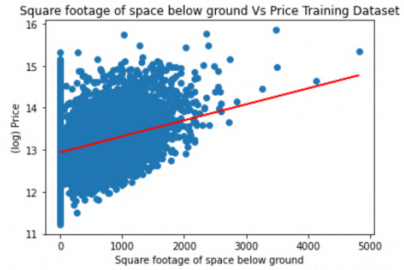
Coefficient of Determination 0.010214239816436765



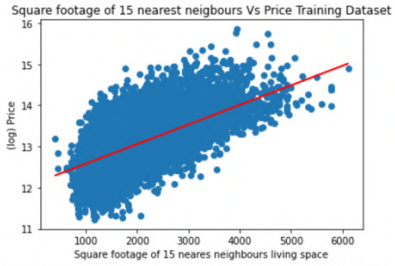
Coefficient of Determination 0.0022977963979891047


```
Coefficient of Determination 0.12321638985290417
```


```
Coefficient of Determination 0.30641341728848137
```




```
Coefficient of Determination 0.10215967225656286
```




```
Coefficient of Determination 0.3826837073305318
```

Code

```
[ ] # linear regression code
    # creates training, validation, and test sets with 80/10/10
    train, other = train_test_split(pricedf, test_size=0.2, random_state=0)
    validation, test = train_test_split(other, test_size=0.5, random_state=0)

    # z sets include just the price
    z_train = train['price']
    z_val = validation['price']
    z_test = test['price']

    # logarithmically scaling the price for the training sets and test sets
    z_log = np.log1p(z_train)
    z_testlog = np.log1p(z_test)
```

```
[ ] from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()



    #Extracting arrays and reshaping the values from the data frames
    lat_train = X_train['lat'].values.reshape(-1, 1)
    lat_test = X_test['lat'].values.reshape(-1,1)
    regressor = LinearRegression()
    regressor.fit(lat_train, z_log)

    #Visualizing the latitude training Test Results
    plt.scatter(lat_train, z_log)
    plt.plot(lat_train, regressor.predict(lat_train), color = 'red')
    plt.title ("Latitude Vs Price Training Dataset")
    plt.xlabel("Latitude")
    plt.ylabel("(log) Price")
    plt.savefig('lat.png')
    plt.show()

    #Calculating R_square, coefficient and intercept
    R_square = regressor.score(lat_train, z_log)
    print('Coefficient of Determination', R_square)
    coefficients = (regressor.fit(lat_train, z_log)).coef_
    print('coefficients', coefficients)
    intercept = (regressor.fit(lat_train, z_log)).intercept_
    print('intercept', intercept)


    #Visualizing the latitude Test Results
    plt.scatter(lat_test, z_testlog)
    plt.plot(lat_train, regressor.predict(lat_train), color = 'red')
    plt.title("Latitude vs Price for Test DataSet")
    plt.xlabel("latitude")
    plt.ylabel("(log) Price")
    plt.savefig('lattest.png')
    plt.show()

    R_square2 = regressor.score(lat_test, z_testlog)
    print('Coefficient of Determination', R_square2)
```

```
[ ]
#Extracting arrays and reshaping the values from the data frames
sqftliving_train = X_train['sqft_living'].values.reshape(-1, 1)
sqfttest = X_test['sqft_living'].values.reshape(-1,1)
regressor = LinearRegression()
regressor.fit(sqftliving_train, z_log)

#Visualizing the square footage living space training Test Results
plt.scatter(sqftliving_train, z_log)
plt.plot(sqftliving_train, regressor.predict(sqftliving_train), color = 'red')
plt.title ("Square footage Vs Price Training Dataset")
plt.xlabel("Square footage of living space")
plt.ylabel("(log) Price")
plt.savefig('sqftsct.png')
plt.show()

#Calculating R_square, coefficient and intercept
R_square = regressor.score(sqftliving_train, z_log)
print('Coefficient of Determination', R_square)
print(regressor.coef_)
print(regressor.intercept_)


#Visualizing the square footage living space  Test Results
plt.scatter(sqfttest, z_testlog)
plt.plot(sqftliving_train, regressor.predict(sqftliving_train), color = 'red')
plt.title("Square footage of living space vs Price for Test DataSet")
plt.xlabel("Square footage of living space")
plt.ylabel("(log)Price")
plt.savefig('sqfttest.png')
plt.show()

R_squaretest = regressor.score(sqfttest, z_testlog)
print('Coefficient of Determination', R_squaretest)
```

```
[ ]
#Extracting arrays and reshaping the values from the data frames
sqftlot_train = X_train['sqft_lot'].values.reshape(-1, 1)
regressor = LinearRegression()
regressor.fit(sqftlot_train, z_log)

#Visualizing the square footage of landscape training Test Results
plt.scatter(sqftlot_train, z_log)
plt.plot(sqftlot_train, regressor.predict(sqftlot_train), color = 'red')
plt.title ("Square footage of landspace Vs Price Training Dataset")
plt.xlabel("Square footage of landspace")
plt.ylabel("(log) Price")
plt.show()

#Calculating R_square
R_square = regressor.score(sqftlot_train, z_log)
print('Coefficient of Determination', R_square)
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
long_train = X_train['long'].values.reshape(-1, 1)
regressor.fit(long_train, z_log)


#Visualizing the Longitude training Test Results
plt.scatter(long_train, z_log)
plt.plot(long_train, regressor.predict(long_train), color = 'red')
plt.title ("Longitude Vs Price Training Dataset")
plt.xlabel("Longitude")
plt.ylabel("(log) Price")
plt.show()

#Calculating R_square
R_square = regressor.score(long_train, z_log)
print('Coefficient of Determination', R_square)
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
bedrooms_train = X_train['bedrooms'].values.reshape(-1, 1)
regressor.fit(bedrooms_train, z_log)


#Visualizing the bedrooms vs price training Results
plt.scatter(bedrooms_train, z_log)
plt.plot(bedrooms_train, regressor.predict(bedrooms_train), color = 'red')
plt.title ("Bedrooms Vs Price Training Dataset")
plt.xlabel("Bedrooms")
plt.ylabel("(log) Price")
plt.show()


#Calculating R_square
R_square = regressor.score(bedrooms_train, z_log)
print('Coefficient of Determination', R_square)
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
bathrooms_train = X_train['bathrooms'].values.reshape(-1, 1)
regressor.fit(bathrooms_train, z_log)


#Visualizing the bathrooms vs price training Results
plt.scatter(bathrooms_train, z_log)
plt.plot(bathrooms_train, regressor.predict(bathrooms_train), color = 'red')
plt.title ("Bathrooms Vs Price Training Dataset")
plt.xlabel("Bathrooms")
plt.ylabel("(log) Price")
plt.show()

#Calculating R_square
R_square = regressor.score(bathrooms_train, z_log)
print('Coefficient of Determination', R_square)
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
    floors_train = X_train['floors'].values.reshape(-1, 1)
    regressor.fit(floors_train, z_log)


    #Visualizing the Number of floors training Training Results
    plt.scatter(floors_train, z_log)
    plt.plot(floors_train, regressor.predict(floors_train), color = 'red')
    plt.title ("Number of floors Vs Price Training Dataset")
    plt.xlabel("Floors")
    plt.ylabel("(log) Price")
    plt.show()

    #Calculating R_square
    R_square = regressor.score(floors_train, z_log)
    print('Coefficient of Determination', R_square)
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
    sqftbase_train = X_train['sqft_basement'].values.reshape(-1, 1)
    regressor.fit(sqftbase_train, z_log)


    #Visualizing the square footage of space below ground training Results
    plt.scatter(sqftbase_train, z_log)
    plt.plot(sqftbase_train, regressor.predict(sqftbase_train), color = 'red')
    plt.title ("Square footage of space below ground Vs Price Training Dataset")
    plt.xlabel("Square footage of space below ground")
    plt.ylabel("(log) Price")
    plt.show()

    #Calculating R_square
    R_square = regressor.score(sqftbase_train, z_log)
    print('Coefficient of Determination', R_square)
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
    sqftliving15_train = X_train['sqft_living15'].values.reshape(-1, 1)
    sqftliving15_test = X_test['sqft_living15'].values.reshape(-1, 1)

    regressor = LinearRegression()
    regressor.fit(sqftliving15_train, z_log)


    #Visualizing the square footage living space of 15 nearest neighbours training Results
    plt.scatter(sqftliving15_train, z_log)
    plt.plot(sqftliving15_train, regressor.predict(sqftliving15_train), color = 'red')
    plt.title ("Square footage of 15 nearest neigbours Vs Price Training Dataset")
    plt.xlabel("Square footage of 15 neares neighbours living space")
    plt.ylabel("(log) Price")
    plt.savefig('sqftliv15.png')
    plt.show()

    #Calculating R_square
    R_square = regressor.score(sqftliving15_train, z_log)
    print('Coefficient of Determination', R_square)

    #Visualizing the square footage living space of 15 nearest neighbours Test Results
    plt.scatter(sqftliving15_test, z_testlog)
    plt.plot(sqftliving15_train, regressor.predict(sqftliving15_train), color = 'red')
    plt.title ("Square footage of 15 nearest neigHbours  Vs Price Test Dataset")
    plt.xlabel("Square footage of 15 neares neighbours living space")
    plt.ylabel("(log) Price")
    plt.savefig('sqftliv15test.png')
    plt.show()
```

```
[ ] #Extracting arrays and reshaping the values from the data frames
    sqftlot15_train = X_train['sqft_lot15'].values.reshape(-1, 1)
    regressor = LinearRegression()
    regressor.fit(sqftlot15_train, z_log)


    #Visualizing the square footage of landspace of nearest 15 neighbours training  Results
    plt.scatter(sqftlot15_train, z_log)
    plt.plot(sqftlot15_train, regressor.predict(sqftlot15_train), color = 'red')
    plt.title ("Square footage of 15 neighbours landspace  Vs Price Training Dataset")
    plt.xlabel("Square footage of 15 neighbours landspace")
    plt.ylabel("(log) Price")
    plt.show()

    #Calculating R_square
    R_square = regressor.score(sqftlot15_train, z_log)
    print('Coefficient of Determination', R_square)
```

C – Logistic Regression (Ben Lovell)

Further Data Pre-Processing:

```
price_lab = ['A', 'B', 'C', 'D', 'E']
price_pd['price_range_bin'] = pd.cut(price_pd.price, [0, 300000, 400000, 500000, 700000, 8000000], labels=price_lab)
print(price_pd)

price_pd_bin = price_pd.replace(['A', 'B', 'C'], 0, regex = True)
price_pd_bin = price_pd_bin.replace(['D', 'E'], 1, regex = True)

print(price_pd_bin)

price_pd_bin.to_csv('priceRangeBinary')

hdf_prepro['PriceBinary'] = price_pd_bin['price_range_bin']
print(hdf_prepro)

hdf_prepro.to_csv("KC_House_Processed_PriceBinary.csv")
```

```
critical = (price_pd_bin.price_range_bin == 1).sum()
non_critical = (price_pd_bin.price_range_bin == 0).sum()
graph = ['critical', 'non-critical']
plt.bar(graph, [critical, non_critical])
plt.ylabel('Count')
plt.show()

price_pd_bin.sample(5)
```

```
total = len(price_pd_bin)
#nb_crit = price_pd_bin['price_range'].sum()
crit = (price_pd_bin.price_range_bin == 1).sum()
non_crit = total - crit
print(non_crit)
hf_non = price_pd_bin.loc[price_pd_bin['price_range_bin'] == 0].sample(crit)
hf_crit = price_pd_bin.loc[price_pd_bin['price_range_bin'] == 1]
resampled_df = pd.concat((hf_crit, hf_non))
resampled_df
```

```
critical = (resampled_df.price_range_bin == 1).sum()
non_critical = (resampled_df.price_range_bin == 0).sum()
graphs = ['critical', 'non_critical']
plt.bar(graphs, [critical, non_critical])
plt.ylabel('Count')
plt.show()
print((resampled_df.price_range_bin == 1).sum())
print((resampled_df.price_range_bin == 0).sum())
```

Method:

```
resampled_df['bedrooms'] = hdf_prepro['bedrooms']
resampled_df['bathroom'] = hdf_prepro['bathrooms']
resampled_df['floors'] = hdf_prepro['floors']

resampled_df['sqft_living'] = hdf_prepro['sqft_living']
resampled_df['sqft_living'] = resampled_df['sqft_living'].div(1000)
```

```
train, other = train_test_split(resampled_df, test_size = 0.2, random_state = 0)
validation, test = train_test_split(other, test_size = 0.5, random_state = 0)
```

```
X_train = train.drop(columns=['price_range_bin'])
y_train = train['price_range_bin']

X_val = validation.drop(columns = ['price_range_bin'])
y_val = validation['price_range_bin']

X_test = test.drop(columns = ['price_range_bin'])
y_test = test['price_range_bin'].values.reshape(-1,1)
```

```
total_train = len(train)
total_val = len(validation)
total_test = len(test)
```

```
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

mylr = LogisticRegression()
mylr.fit(X_train, y_train)

mylt = LogisticRegression()
mylt.fit(X_val, y_val)

mylv = LogisticRegression()
mylv.fit(X_test, y_test)
```

```python
from scipy import stats
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score

class ModelSummary:
    """ This class extracts a summary of the model

    Methods
    -------
    get_se()
        computes standard error
    get_ci(SE_est)
        computes confidence intervals
    get_pvals()
        computes p-values
    get_summary(name=None)
        prints the summary of the model
    """

    def __init__(self, clf, X, y):
        """
        Parameters
        ----------
        clf: class
            the classifier object model
        X: pandas Dataframe
            matrix of predictors
        y: numpy array
            matrix of variable
        """
        self.clf = clf
        self.X = X
        self.y = y
        pass

    def get_se(self):
        # from here https://stats.stackexchange.com/questions/89484/how-to-compute-the-standard-errors-of-a-logistic-regressions-coefficients
        predProbs = self.clf.predict_proba(self.X)
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))

    def get_ci(self, SE_est):
        """
        Parameters
        ----------
        SE_est: numpy array
            matrix of standard error estimations
        """
        p = 0.975
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:,0] = lower
        cis[:,1] = upper
        return cis
```

```python
    def get_pvals(self):
        # from here https://stackoverflow.com/questions/25122999/scikit-learn-how-to-check-coefficients-significance
        p = self.clf.predict_proba(self.X)
        n = len(p)
        m = len(self.clf.coef_[0]) + 1
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        se = self.get_se()
        t =  coefs/se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
        pvals = self.get_pvals()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        data = []
        for i in range(len(coefs)):
            currlist = []
            currlist.append(np.round(coefs[i], 3))
            currlist.append(np.round(ses[i], 3))
            currlist.append(np.round(pvals[i], 3))
            currlist.append(np.round(lower[i], 3))
            currlist.append(np.round(upper[i], 3))
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
        sumdf = pd.DataFrame(columns=cols, data=data)
        if names is not None:
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        else:
            try:
                names = list(self.X.columns)
                new_names = ['intercept']*(len(names) + 1)
                new_names[1:] = [i for i in names]
                sumdf.index = new_names
            except:
                pass
        print(sumdf)
        acc = accuracy_score(self.y, self.clf.predict(self.X))
        confmat = confusion_matrix(self.y, self.clf.predict(self.X))
        print('-'*60)
        print('Confusion Matrix (total:{}) \t Accuracy: \t  {}'.format(len(self.X),np.round(acc, 3)))
        print('  TP: {} | FN: {}'.format(confmat[1][1],confmat[1][0]))
        print('  FP: {} | TN: {}'.format(confmat[0][1],confmat[0][0]))
```

```
modsummary = ModelSummary(mylr, X_train, y_train)
modsummary.get_summary()

modsummary = ModelSummary(mylt, X_val, y_val)
modsummary.get_summary()

# modsummary = ModelSummary(mylv, X_test, y_test)
# modsummary.get_summary()
```

D – Decision Tree (Archie Bond)

```
##Decision Tree Method###

#import modules
from sklearn.tree import DecisionTreeClassifier
#build decision tree
dtree = DecisionTreeClassifier(max_depth = 8, min_impurity_decrease = 0.001)
dtree.fit(X_train_standardised, y_train)
```

```
DecisionTreeClassifier(max_depth=8, min_impurity_decrease=0.001)
```

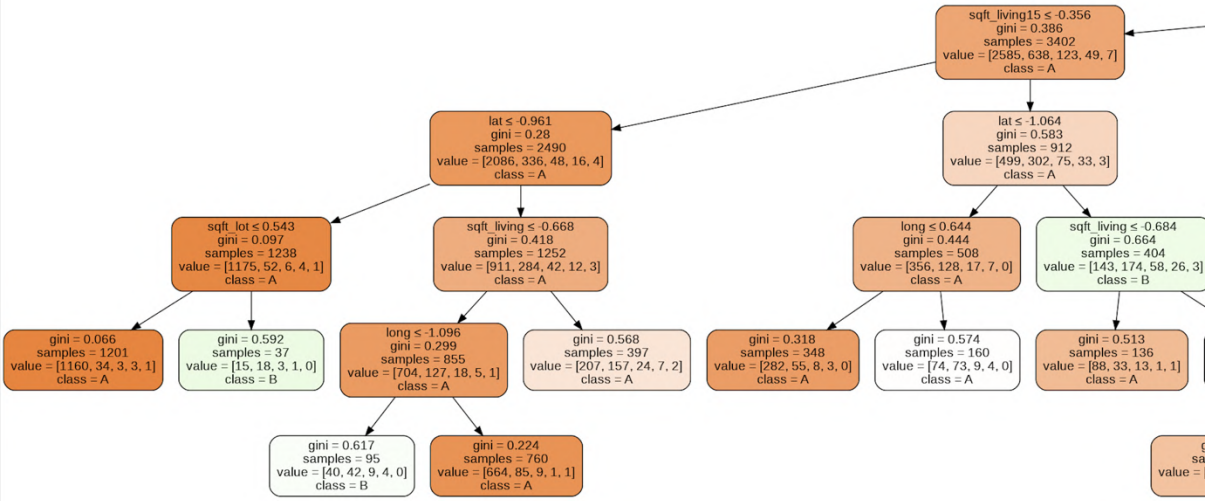
```
print('Accuracy on the val set: {}'.format(acc_val))
print('Precision on the val set: {}'.format(prec_val))
print('Recall on the val set: {}'.format(rec_val))
print('Accuracy on the test set: {}'.format(acc_test))
print('Precision on the test set: {}'.format(prec_test))
print('Recall on the test set: {}'.format(rec_test))
print('Accuracy on the train set: {}'.format(acc_train))
print('Precision on the train set: {}'.format(prec_train))
print('Recall on the train set: {}'.format(rec_train))
```

```
Accuracy on the val set: 0.6186950485886163
Precision on the val set: 0.6239913089118172
Recall on the val set: 0.6186950485886163
Accuracy on the test set: 0.6216466234967623
Precision on the test set: 0.6317815497736411
Recall on the test set: 0.6216466234967623
Accuracy on the train set: 0.6320416425679584
Precision on the train set: 0.6430032992041828
Recall on the train set: 0.6320416425679584
```

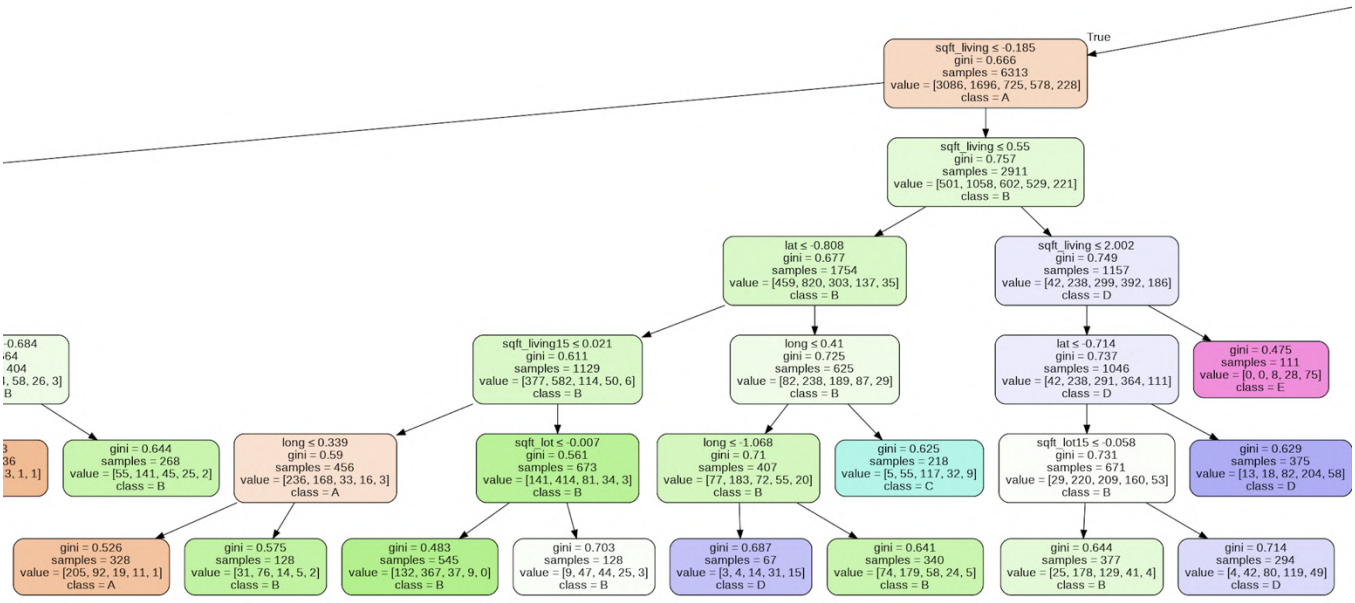
```
#checking performance metrics for decision tree models
from sklearn.metrics import accuracy_score, precision_score, recall_score

# uses model to predict price range category
ypred_val = dtree.predict(X_val_standardised)
ypred_test = dtree.predict(X_test_standardised)
ypred_train = dtree.predict(X_train_standardised)

# calculates performance metrics for validation set
acc_val = accuracy_score(y_val, ypred_val)
prec_val = precision_score(y_val, ypred_val, average='weighted', zero_division=0)
rec_val = recall_score(y_val, ypred_val, average='weighted')
# calculates performance metrics for test set
acc_test = accuracy_score(y_test, ypred_test)
prec_test = precision_score(y_test, ypred_test, average='weighted', zero_division=0)
rec_test = recall_score(y_test, ypred_test, average='weighted')
# calculates performance metrics for train set
acc_train = accuracy_score(y_train, ypred_train)
prec_train = precision_score(y_train, ypred_train, average='weighted', zero_division=0)
rec_train = recall_score(y_train, ypred_train, average='weighted')
```

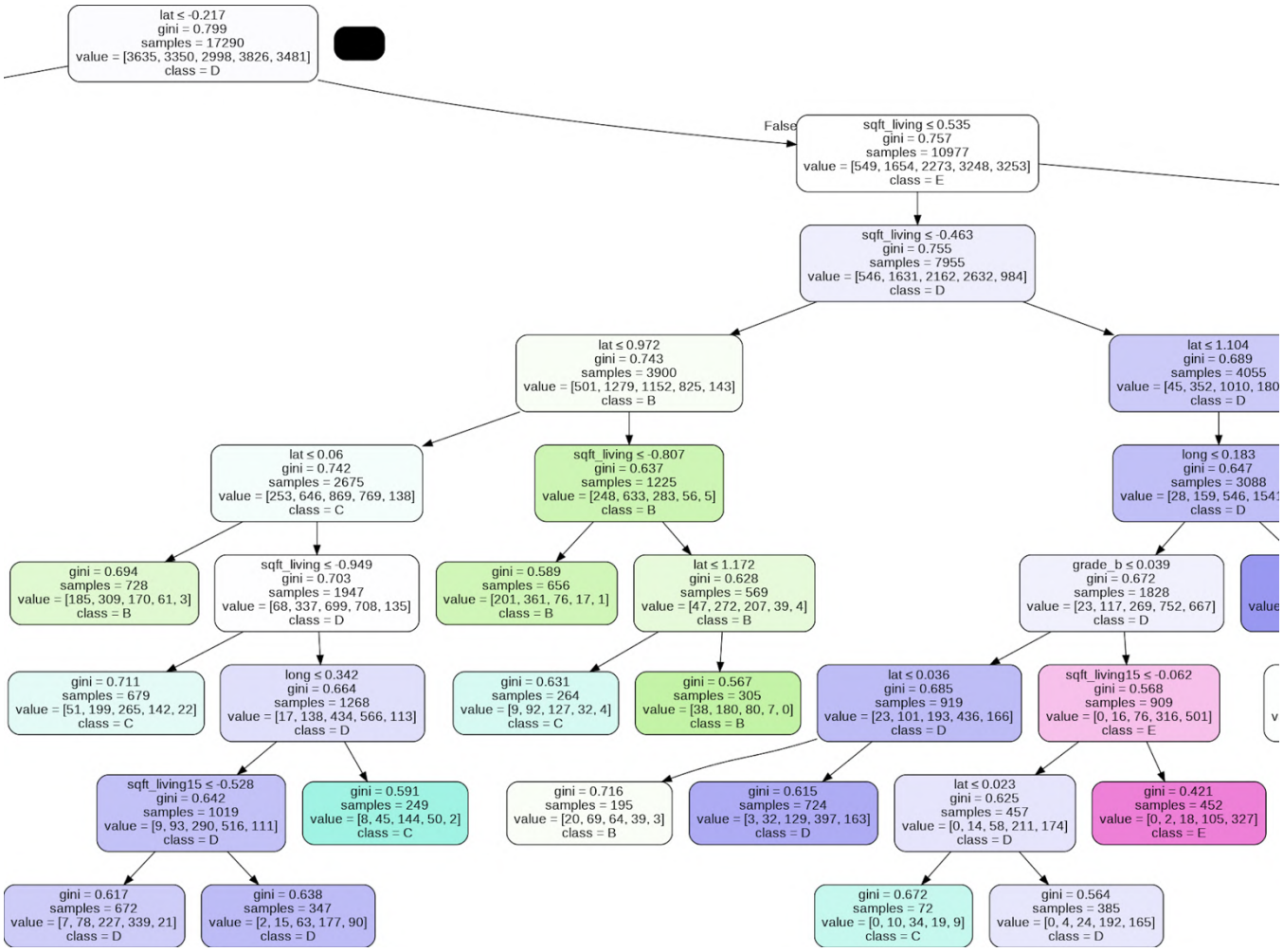
```
import sklearn.tree as tree
import graphviz
dot_data = tree.export_graphviz(dtree, out_file=None)
graph = graphviz.Source(dot_data)

predictors = X_val_standardised.columns
dot_data = tree.export_graphviz(dtree, out_file=None,
                                feature_names=predictors,
                                class_names=('A', 'B', 'C', 'D', 'E'),
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Section 1

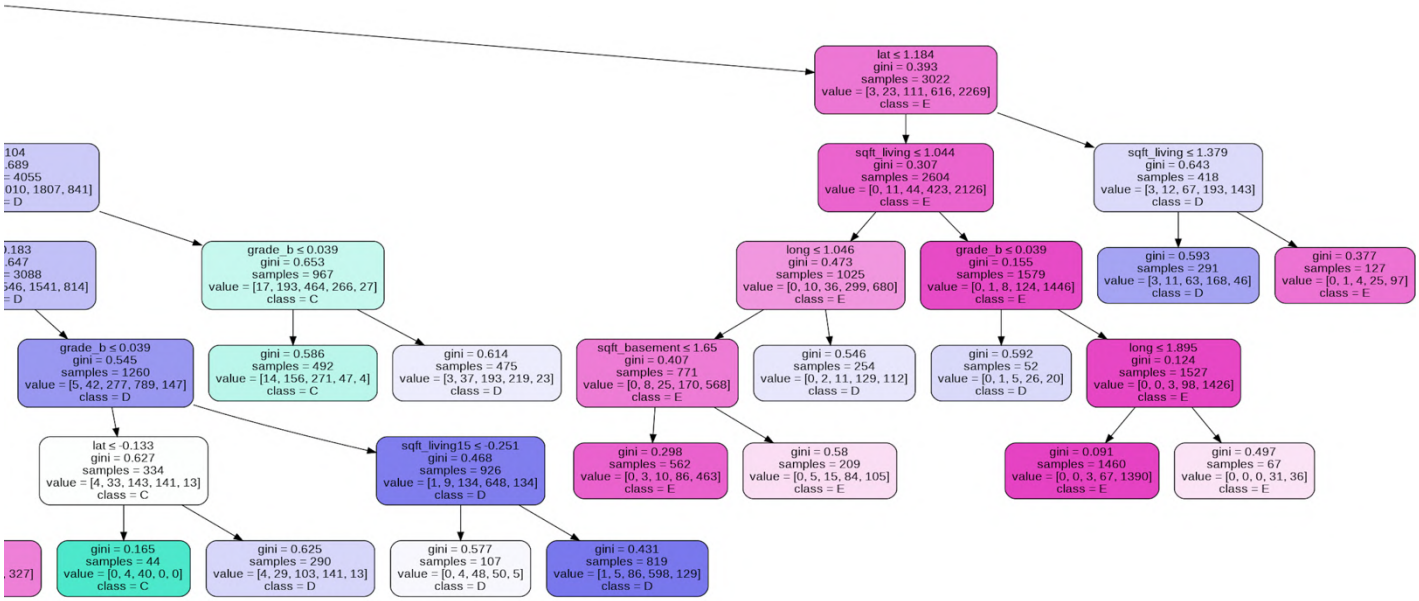


Section 2




lat ≤ -0.217
gini = 0.799
samples = 17290
value = [3635, 3350, 2998, 3826, 3481]
class = D
False
sqft_living ≤ 0.535
gini = 0.757
samples = 10977
value = [549, 1654, 2273, 3248, 3253]
class = E
sqft_living ≤ -0.463
gini = 0.755
samples = 7955
value = [546, 1631, 2162, 2632, 984]
class = D
lat ≤ 0.972
gini = 0.743
samples = 3900
value = [501, 1279, 1152, 825, 143]
class = B
lat ≤ 1.104
gini = 0.689
samples = 4055
value = [45, 352, 1010, 180
class = D
lat ≤ 0.06
gini = 0.742
samples = 2675
value = [253, 646, 869, 769, 138]
class = C
sqft_living ≤ -0.807
gini = 0.637
samples = 1225
value = [248, 633, 283, 56, 5]
class = B
long ≤ 0.183
gini = 0.647
samples = 3088
value = [28, 159, 546, 154
class = D
gini = 0.694
samples = 728
value = [185, 309, 170, 61, 3]
class = B
sqft_living ≤ -0.949
gini = 0.703
samples = 1947
value = [68, 337, 699, 708, 135]
class = D
gini = 0.589
samples = 656
value = [201, 361, 76, 17, 1]
class = B
lat ≤ 1.172
gini = 0.628
samples = 569
value = [47, 272, 207, 39, 4]
class = B
grade_b ≤ 0.039
gini = 0.672
samples = 1828
value = [23, 117, 269, 752, 667]
class = D
gini = 0.711
samples = 679
value = [51, 199, 265, 142, 22]
class = C
long ≤ 0.342
gini = 0.664
samples = 1268
value = [17, 138, 434, 566, 113]
class = D
gini = 0.631
samples = 264
value = [9, 92, 127, 32, 4]
class = C
gini = 0.567
samples = 305
value = [38, 180, 80, 7, 0]
class = B
lat ≤ 0.036
gini = 0.685
samples = 919
value = [23, 101, 193, 436, 166]
class = D
sqft_living15 ≤ -0.062
gini = 0.568
samples = 909
value = [0, 16, 76, 316, 501]
class = E
sqft_living15 ≤ -0.528
gini = 0.642
samples = 1019
value = [9, 93, 290, 516, 111]
class = D
gini = 0.591
samples = 249
value = [8, 45, 144, 50, 2]
class = C
gini = 0.716
samples = 195
value = [20, 69, 64, 39, 3]
class = B
gini = 0.615
samples = 724
value = [3, 32, 129, 397, 163]
class = D
lat ≤ 0.023
gini = 0.625
samples = 457
value = [0, 14, 58, 211, 174]
class = D
gini = 0.421
samples = 452
value = [0, 2, 18, 105, 327]
class = E
gini = 0.617
samples = 672
value = [7, 78, 227, 339, 21]
class = D
gini = 0.638
samples = 347
value = [2, 15, 63, 177, 90]
class = D
gini = 0.672
samples = 72
value = [0, 10, 34, 19, 9]
class = C
gini = 0.564
samples = 385
value = [0, 4, 24, 192, 165]
class = D

Section 4


lat ≤ 1.184
gini = 0.393
samples = 3022
value = [3, 23, 111, 616, 2269]
class = E
sqft_living ≤ 1.044
gini = 0.307
samples = 2604
value = [0, 11, 44, 423, 2126]
class = E
sqft_living ≤ 1.379
gini = 0.643
samples = 418
value = [3, 12, 67, 193, 143]
class = D
grade_b ≤ 0.039
gini = 0.653
samples = 967
value = [17, 193, 464, 266, 27]
class = C
long ≤ 1.046
gini = 0.473
samples = 1025
value = [0, 10, 36, 299, 680]
class = E
grade_b ≤ 0.039
gini = 0.155
samples = 1579
value = [0, 1, 8, 124, 1446]
class = E
gini = 0.593
samples = 291
value = [3, 11, 63, 168, 46]
class = D
gini = 0.377
samples = 127
value = [0, 1, 4, 25, 97]
class = E
grade_b ≤ 0.039
gini = 0.545
samples = 1260
value = [5, 42, 277, 789, 147]
class = D
gini = 0.586
samples = 492
value = [14, 156, 271, 47, 4]
class = C
gini = 0.614
samples = 475
value = [3, 37, 193, 219, 23]
class = D
sqft_basement ≤ 1.65
gini = 0.407
samples = 771
value = [0, 8, 25, 170, 568]
class = E
gini = 0.546
samples = 254
value = [0, 2, 11, 129, 112]
class = D
gini = 0.592
samples = 52
value = [0, 1, 5, 26, 20]
class = D
long ≤ 1.895
gini = 0.124
samples = 1527
value = [0, 0, 3, 98, 1426]
class = E
lat ≤ -0.133
gini = 0.627
samples = 334
value = [4, 33, 143, 141, 13]
class = C
sqft_living15 ≤ -0.251
gini = 0.468
samples = 926
value = [1, 9, 134, 648, 134]
class = D
gini = 0.298
samples = 562
value = [0, 3, 10, 86, 463]
class = E
gini = 0.58
samples = 209
value = [0, 5, 15, 84, 105]
class = E
gini = 0.091
samples = 1460
value = [0, 0, 3, 67, 1390]
class = E
gini = 0.497
samples = 67
value = [0, 0, 0, 31, 36]
class = E
gini = 0.165
samples = 44
value = [0, 4, 40, 0, 0]
class = C
gini = 0.625
samples = 290
value = [4, 29, 103, 141, 13]
class = D
gini = 0.577
samples = 107
value = [0, 4, 48, 50, 5]
class = D
gini = 0.431
samples = 819
value = [1, 5, 86, 598, 129]
class = D

```
max_depth = range(1,14)
train_score = []
val_score = []
d_out = 0

for d in max_depth:
  dtree = tree.DecisionTreeClassifier(random_state=0, max_depth=d)
  dtree.fit(np.asarray(X_train_standardised), y_train)
  val_score.append(accuracy_score(y_val, dtree.predict(np.asarray(X_val_standardised))))
  train_score.append(accuracy_score(y_train, dtree.predict(np.asarray(X_train_standardised))))
  if (d_out == 0) & (d>2):
    if (train_score[-1]/train_score[-2] < 1.001) | (np.mean(val_score[-3:]) < np.mean(val_score[-4:-1])):
      d_out = d-1


plt.plot(max_depth, train_score, label='Training')
plt.plot(max_depth, val_score, label='Validation')
plt.plot(d_out, train_score[d_out-1],'or')
plt.legend()
plt.xlabel('Maximum depth')
plt.ylabel('Model accuracy')
plt.show()
print(d_out)
```

```
max_depth = range(1,14)
train_score = []
val_score = []
d_out = 0

for d in max_depth:
  dtree = tree.DecisionTreeClassifier(random_state=0, max_depth=d)
  dtree.fit(np.asarray(X_train_standardised), y_train)
  val_score.append(precision_score(y_val, dtree.predict(np.asarray(X_val_standardised)), average='weighted'))
  train_score.append(precision_score(y_train, dtree.predict(np.asarray(X_train_standardised)), average='weighted'))
  if (d_out == 0) & (d>2):
    if (train_score[-1]/train_score[-2] < 1.001) | (np.mean(val_score[-3:]) < np.mean(val_score[-4:-1])):
      d_out = d-1


plt.plot(max_depth, train_score, label='Training')
plt.plot(max_depth, val_score, label='Validation')
plt.legend()
plt.xlabel('Maximum depth')
plt.ylabel('Model precision')
plt.show()
print(d_out)
```

```
max_depth = range(1,14)
train_score = []
val_score = []
d_out = 0

for d in max_depth:
  dtree = tree.DecisionTreeClassifier(random_state=0, max_depth=d)
  dtree.fit(np.asarray(X_train_standardised), y_train)
  val_score.append(recall_score(y_val, dtree.predict(np.asarray(X_val_standardised)), average='weighted'))
  train_score.append(recall_score(y_train, dtree.predict(np.asarray(X_train_standardised)), average='weighted'))
  if (d_out == 0) & (d>2):
    if (train_score[-1]/train_score[-2] < 1.001) | (np.mean(val_score[-3:]) < np.mean(val_score[-4:-1])):
      d_out = d-1


plt.plot(max_depth, train_score, label='Training')
plt.plot(max_depth, val_score, label='Validation')
plt.legend()
plt.xlabel('Maximum depth')
plt.ylabel('Model recall')
plt.show()
print(d_out)
```

```
import pydotplus

pydot_graph = pydotplus.graph_from_dot_data(dot_data)
pydot_graph.write_png('original_tree.png')
pydot_graph.set_size('"50,50!"')
pydot_graph.write_png('resized_tree.png')
```

```
def DecTree(hdf_prepro: pd.DataFrame) -> pd.DataFrame:
  ''' Takes a dataset as a dataframe and trains a Decision Tree Classifier, returns predicted values and test values'''
  dtree = tree.DecisionTreeClassifier(max_depth=8, min_impurity_decrease=0.001)
  columns=list(hdf_prepro.columns)
  index=columns.index("Price Range")
  y_col = columns.pop(index)
  y = hdf_prepro[y_col].to_numpy()
  X = hdf_prepro[columns].to_numpy()
  train, other = train_test_split(hdf_prepro, test_size=0.2, random_state=0)
  validation, test = train_test_split(other, test_size=0.5, random_state=0)

  X_train = train.drop(['Price Range'], axis = 1)
  y_train = train['Price Range']
  X_val = validation.drop(['Price Range'], axis = 1)
  y_val = validation['Price Range']
  X_test = test.drop(['Price Range'], axis = 1)
  y_test = test['Price Range']
  means = X_train.mean(axis=0)
  stds = X_train.std(axis=0)
  X_train_standardised = (X_train - means) / stds
  X_test_standardised = (X_test - means) / stds
  dtree = dtree.fit(X_train_standardised, y_train)
  pred  = dtree.predict(X_test_standardised)
  return pred, y_test
```

```
def conf_matrix(ytest: np.ndarray, predict: np.ndarray) -> pd.DataFrame:
    '''Creates a confusion matrix and normalises with the sum of the matrix and returns the normalized matrix'''

    price_brack = ['A', 'B', 'C', 'D', 'E']
    conf_matrix= confusion_matrix(ytest, predict, labels=price_brack)
    conf_matrixper = conf_matrix/np.sum(conf_matrix)
    conf_matrixper = pd.DataFrame(conf_matrixper, index=price_brack, columns=price_brack)

    return conf_matrixper

col = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_basement', 'lat', 'long',
       'sqft_living15', 'sqft_lot15', 'waterfront', 'view_b', 'condition_b', 'grade_b', 'is_new_or_renovated']
accuracies = []

for i in (col):
    hdf_conf = hdf_prepro.copy().drop(i, axis = 1)
    pred, y_test = DecTree(hdf_conf)
    conf = conf_matrix(y_test, pred)
    conf = pd.DataFrame(conf)
    accuracy = accuracy_score(y_test, pred)
    accuracies.append(accuracy)
    # prints model accuracy for each instance of a variable removed
    print(f'{accuracy} is the accuracy when {i} is not considered')
accuracies = np.array(accuracies)
lowAcc = accuracies.argmin()
# prints which variable influences the predictability of the price the most (most influential variable for predicting price)
print(f'{col[lowAcc]} is the most influential feature when predicting the price range')
```

```
0.6216466234967623 is the accuracy when bedrooms is not considered
0.6216466234967623 is the accuracy when bathrooms is not considered
0.5888066604995374 is the accuracy when sqft_living is not considered
0.6207215541165587 is the accuracy when sqft_lot is not considered
0.6216466234967623 is the accuracy when floors is not considered
0.6216466234967623 is the accuracy when sqft_basement is not considered
0.45837187789084183 is the accuracy when lat is not considered
0.6017576318223867 is the accuracy when long is not considered
0.6197964847363552 is the accuracy when sqft_living15 is not considered
0.6211840888066605 is the accuracy when sqft_lot15 is not considered
0.6216466234967623 is the accuracy when waterfront is not considered
0.6216466234967623 is the accuracy when view_b is not considered
0.6216466234967623 is the accuracy when condition_b is not considered
0.6184088806660499 is the accuracy when grade_b is not considered
0.6216466234967623 is the accuracy when is_new_or_renovated is not considered
lat is the most influential feature when predicting the price range
```

```
import matplotlib.pyplot as plt
accuracies = np.array(accuracies)
print(accuracies)

f, ax = plt.subplots(figsize=(22,10)) # set the size that you'd like (width, height)
plt.bar(col,accuracies, width=0.5) #creates barchart of accuracies of each feature
plt.title('Accuracies when each feature is removed', fontsize=20)
plt.xlabel('Different Features', fontsize=20)
plt.ylabel('Accuracies of the test set', fontsize=20)
ax.legend(fontsize = 14)
```

E – Random Forest (Rohil J Dave)

Code

```
# import all libraries used
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
import sklearn.tree as tree
import graphviz
import pylab
```

```
# create csv of processed data
hdf_prepro.to_csv("KC_House_Processed_Data.csv")
```

```
# creates training, validation, and test sets with 80/10/10
train, other = train_test_split(hdf_prepro, test_size=0.2, random_state=0)
validation, test = train_test_split(other, test_size=0.5, random_state=0)

# X sets include all attributes except price range, y sets are just price range
X_train = train.drop(['Price Range'], axis = 1)
y_train = train['Price Range']
X_val = validation.drop(['Price Range'], axis = 1)
y_val = validation['Price Range']
X_test = test.drop(['Price Range'], axis = 1)
y_test = test['Price Range']
X_train.head()
```

```
# standardising (normalising) the data so that the mean=0 and std=1, helps to balance data along with the previous binarisation of some features
means = X_train.mean(axis=0)
stds = X_train.std(axis=0)

X_train_standardised = (X_train - means) / stds
X_val_standardised = (X_val - means) / stds
X_test_standardised = (X_test - means) / stds

X_train_standardised.head()
# print(X_train_standardised['sqft_living'].std())
# print(X_test_standardised['sqft_living'].mean())
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	lat	long	sqft_living15	sqft_lot15	waterfront	view_b	condition_b	grade_b	is_n
5268	-0.390023	-1.448095	-0.553934	-0.230230	-0.917914	0.964830	-0.744673	-0.316901	-0.232451	-0.090673	-0.200307	1.395306	-0.961610	
16909	-0.390023	0.494282	-0.326475	-0.100351	-0.917914	-0.022009	0.463181	0.469978	-0.112885	-0.090673	-0.200307	1.395306	1.039863	
16123	-0.390023	-0.800636	-1.073840	-0.127273	-0.917914	0.469968	1.266049	-0.404332	-0.107050	-0.090673	-0.200307	-0.716647	1.039863	
12181	0.670275	0.494282	0.139273	-0.193112	0.925056	-1.008849	0.221611	-0.025464	-0.210123	-0.090673	-0.200307	-0.716647	1.039863	
12617	-0.390023	0.494282	-0.304813	-0.247902	0.925056	0.617128	-1.050189	-0.287757	-0.290686	-0.090673	-0.200307	-0.716647	-0.961610	

```
# method to calculate max_depth of random forest and create plots against performance metrics
max_depth = range(1,14)
t_score = []
v_score = []
d_out = 0

for d in max_depth:
  rfc = RandomForestClassifier(max_depth=d)
  rfc.fit(np.asarray(X_train_standardised), y_train)
  v_score.append(precision_score(y_val, rfc.predict(np.asarray(X_val_standardised)), average='weighted'))
  t_score.append(precision_score(y_train, rfc.predict(np.asarray(X_train_standardised)), average='weighted'))
  # if (d_out == 0) & (d>2):
  #   if (t_score[-1]/t_score[-2] < 1.001) | (np.mean(v_score[-3:]) < np.mean(v_score[-4:-1])):
  #     d_out = d-1


plt.plot(max_depth, t_score, label='Training')
plt.plot(max_depth, v_score, label='Validation')
# plt.plot(d_out, t_score[d_out-1],'or')
plt.legend()
plt.title('Random Forest Precision v Max Depth')
plt.xlabel('Maximum depth')
plt.ylabel('Model precision')
plt.show()
```

```
# method to calculate min_samples_split of random forest and create plots against performance metrics
min_samples_split = range(2,30,2)
t_score = []
v_score = []
d_out = 0

for s in min_samples_split:
  rfc = RandomForestClassifier(min_samples_split=s)
  rfc.fit(np.asarray(X_train_standardised), y_train)
  v_score.append(accuracy_score(y_val, rfc.predict(np.asarray(X_val_standardised))))
  t_score.append(accuracy_score(y_train, rfc.predict(np.asarray(X_train_standardised))))
  # if (d_out == 0) & (d>2):
  #   if (t_score[-1]/t_score[-2] < 1.001) | (np.mean(v_score[-3:]) < np.mean(v_score[-4:-1])):
  #     d_out = d-1


plt.plot(min_samples_split, t_score, label='Training')
plt.plot(min_samples_split, v_score, label='Validation')
# plt.plot(d_out, t_score[d_out-1],'or')
plt.legend()
plt.title('Random Forest Accuracy v Minimum Samples Split')
plt.xlabel('Min samples split')
plt.ylabel('Model accuracy')
plt.show()
```

```
# initialises and trains random forest using standardised training set
rfc  = RandomForestClassifier(max_depth=8, min_samples_split=10)
rfc = rfc.fit(X_train_standardised, y_train)

# uses model to predict price range category
ypred_train = rfc.predict(X_train_standardised)
ypred_val = rfc.predict(X_val_standardised)
ypred_test = rfc.predict(X_test_standardised)

acc_train = accuracy_score(y_train, ypred_train)
prec_train = precision_score(y_train, ypred_train, average='weighted', zero_division=0)
rec_train = recall_score(y_train, ypred_train, average='weighted')

# calculates performance metrics for validation set
acc_val = accuracy_score(y_val, ypred_val)
prec_val = precision_score(y_val, ypred_val, average='weighted', zero_division=0)
rec_val = recall_score(y_val, ypred_val, average='weighted')

# calculates performance metrics for test set
acc_test = accuracy_score(y_test, ypred_test)
prec_test = precision_score(y_test, ypred_test, average='weighted', zero_division=0)
rec_test = recall_score(y_test, ypred_test, average='weighted')

# prints performance metrics
print('Training Precision:{}'.format(prec_train))
print('Training Accuracy:{}'.format(acc_train))
print('Training Recall:{}'.format(rec_train))
print('Validation Precision:{}'.format(prec_val))
print('Validation Accuracy:{}'.format(acc_val))
print('Validation Recall:{}'.format(rec_val))
print('Test Precision:{}'.format(prec_test))
print('Test Accuracy:{}'.format(acc_test))
print('Test Recall:{}'.format(rec_test))
```

```
Training Precision:0.7108625178922031
Training Accuracy:0.7056101792943898
Training Recall:0.7056101792943898
Validation Precision:0.652851165397151
Validation Accuracy:0.6557149467838963
Validation Recall:0.6557149467838963
Test Precision:0.6650339438960163
Test Accuracy:0.6632747456059205
Test Recall:0.6632747456059205
```

```
# creates visualization of random forest tree
estimator = rfc.estimators_[5]
dot_data = tree.export_graphviz(estimator, out_file=None)
graph = graphviz.Source(dot_data)

predictors = X_train_standardised.columns
dot_data = tree.export_graphviz(estimator, out_file=None,
                                feature_names = predictors,
                                class_names = ('A', 'B', 'C', 'D', 'E'),
                                filled = True, rounded = True,
                                special_characters = True)
graph = graphviz.Source(dot_data)
graph

import pydotplus

pydot_graph = pydotplus.graph_from_dot_data(dot_data)
pydot_graph.write_png('original_tree.png')
pydot_graph.set_size('"150,150!"')
pydot_graph.write_png('resized_tree.png')
```

```
def RandForest(hdf_prepro: pd.DataFrame) -> pd.DataFrame:
  ''' Takes a dataset as a dataframe and trains a Random Forest Classifier, returns predicted values and test values'''
  rfc = RandomForestClassifier(max_depth = 8, min_samples_split = 10)
  columns=list(hdf_prepro.columns)
  index=columns.index("Price Range")
  y_col = columns.pop(index)
  y = hdf_prepro[y_col].to_numpy()
  X = hdf_prepro[columns].to_numpy()
  train, other = train_test_split(hdf_prepro, test_size=0.2, random_state=0)
  validation, test = train_test_split(other, test_size=0.5, random_state=0)

  X_train = train.drop(['Price Range'], axis = 1)
  y_train = train['Price Range']
  X_val = validation.drop(['Price Range'], axis = 1)
  y_val = validation['Price Range']
  X_test = test.drop(['Price Range'], axis = 1)
  y_test = test['Price Range']
  means = X_train.mean(axis=0)
  stds = X_train.std(axis=0)
  X_train_standardised = (X_train - means) / stds
  X_test_standardised = (X_test - means) / stds
  rfc = rfc.fit(X_train_standardised, y_train)
  pred  = rfc.predict(X_test_standardised)
  return pred, y_test
```

```
def conf_matrix(ytest: np.ndarray, predict: np.ndarray) -> pd.DataFrame:
    '''Creates a confusion matrix and normalises with the sum of the matrix and returns the normalized matrix'''

    price_brack = ['A', 'B', 'C', 'D', 'E']
    conf_matrix= confusion_matrix(ytest, predict, labels=price_brack)
    conf_matrixper = conf_matrix/np.sum(conf_matrix)
    conf_matrixper = pd.DataFrame(conf_matrixper, index=price_brack, columns=price_brack)

    return conf_matrixper

col = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'lat', 'long', 'sqft_living15', 'sqft_lot15', 'waterfront', 'view_b', 'condition_b',
accuracies = []

for i in (col):
    hdf_conf = hdf_prepro.copy().drop(i, axis = 1)
    pred, y_test = RandForest(hdf_conf)
    conf = conf_matrix(y_test, pred)
    conf = pd.DataFrame(conf)
    accuracy = accuracy_score(y_test, pred)
    accuracies.append(accuracy)
    # prints model accuracy for each instance of a variable removed
    print(f'{accuracy} is the accuracy when {i} is not considered')
accuracies = np.array(accuracies)
lowAcc = accuracies.argmin()
# prints which variable influences the predictability of the price the most (most influential variable for predicting price)
print(f'{col[lowAcc]} is the most influential feature when predicting the price range')
```

```
0.6646623496762257 is the accuracy when bedrooms is not considered
0.6711378353376504 is the accuracy when bathrooms is not considered
0.6415356151711379 is the accuracy when sqft_living is not considered
0.666049953746531 is the accuracy when sqft_lot is not considered
0.6646623496762257 is the accuracy when floors is not considered
0.5138760407030527 is the accuracy when lat is not considered
0.6466234967622572 is the accuracy when long is not considered
0.6554116558741906 is the accuracy when sqft_living15 is not considered
0.6734505087881592 is the accuracy when sqft_lot15 is not considered
0.6702127659574468 is the accuracy when waterfront is not considered
0.6665124884366328 is the accuracy when view_b is not considered
0.6665124884366328 is the accuracy when condition_b is not considered
0.6535615171137835 is the accuracy when grade_b is not considered
0.669750231267345 is the accuracy when is_new_or_renovated is not considered
lat is the most influential feature when predicting the price range
```

Forest Visualization by section from the left side

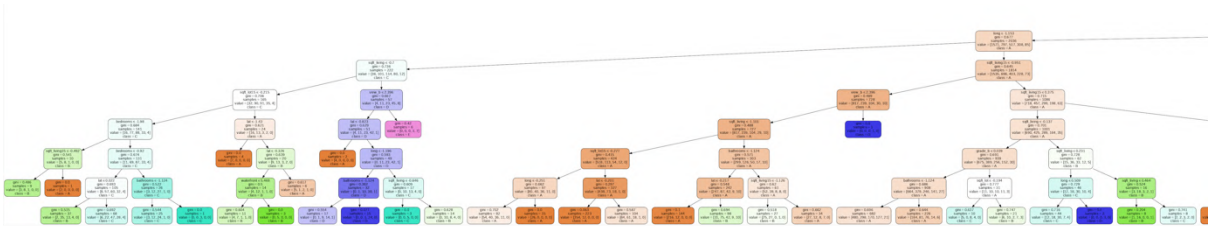


FIGURE E.1: FOREST SECTION 1

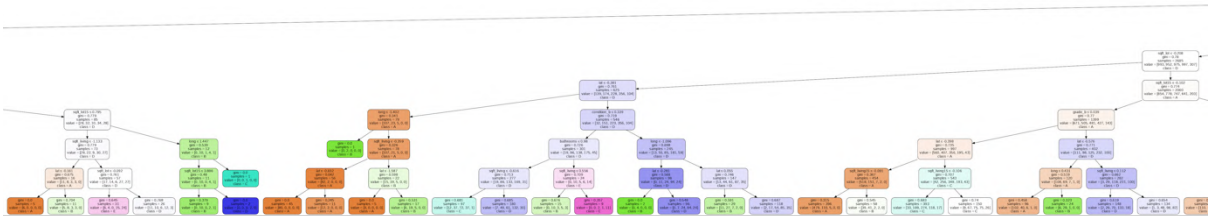


FIGURE E.2: FOREST SECTION 2

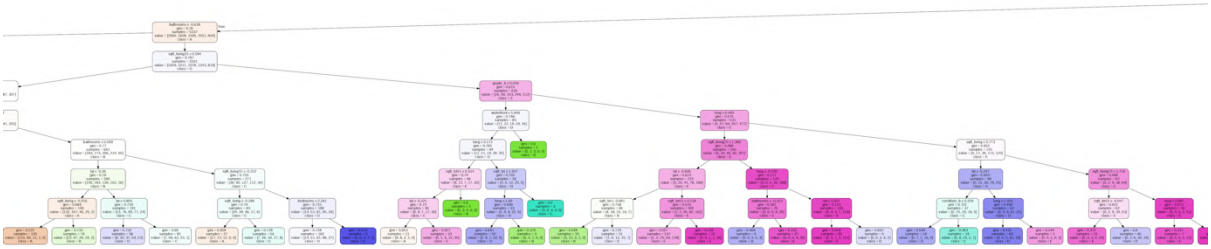


FIGURE E.3: FOREST SECTION 3

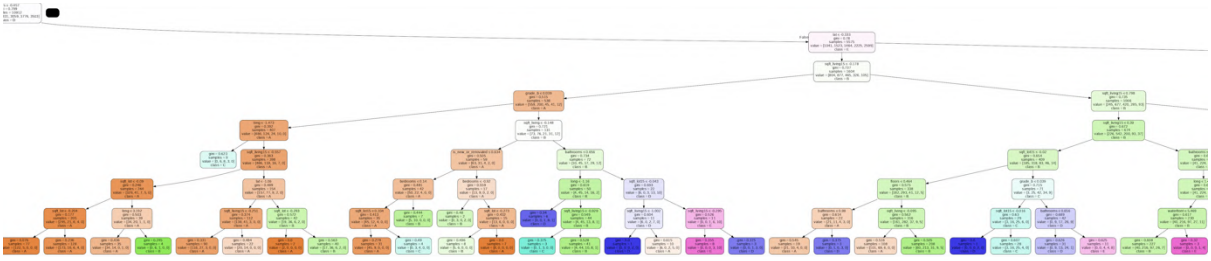


FIGURE E.4: FOREST SECTION 4

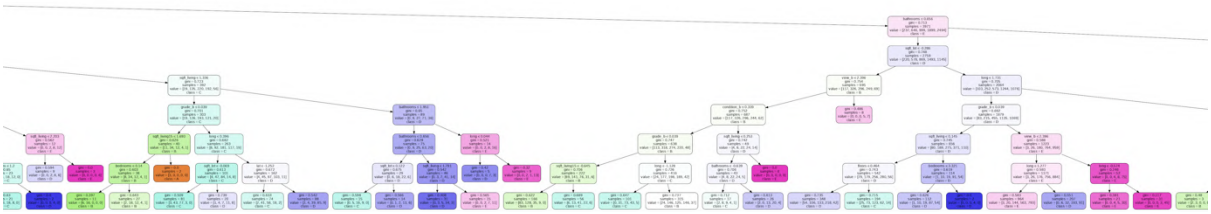


FIGURE E.5: FOREST SECTION 5

FIGURE E.6: FOREST SECTION 6

F – SVM (Kennard S Mah)

```
[31] from sklearn.svm import SVC

     # BINARISE PRICE TO DETERMINE WHETHER PROPERTY IS UPPER VALUE OR LOWER VALUE

     price_pd['price_range'] = hdf['price'] > 540088.1418

     # DROPPING COLUMNS THAT ARE NOT BEING USED

     X_train_standardised = X_train_standardised.drop(['bedrooms', 'bathrooms', 'sqft_lot', 'floors', 'sqft_basement','lat','long','sqft_living15', 'sq
     X_val_standardised = X_val_standardised.drop(['bedrooms', 'bathrooms', 'sqft_lot', 'floors', 'sqft_basement','lat','long','sqft_living15', 'sqft_
     X_test_standardised = X_test_standardised.drop(['bedrooms', 'bathrooms', 'sqft_lot', 'floors', 'sqft_basement','lat','long','sqft_living15', 'sqft
```

```
[15] # DEFINING VALIDATION, TRAINING, AND TESTING FUNCTIONS

     def acc_val(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return accuracy_score(y_val, model.predict(X_val_standardised))
     def prec_val(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return precision_score(y_val, model.predict(X_val_standardised))
     def rec_val(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return recall_score(y_val, model.predict(X_val_standardised))
     def acc_train(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return accuracy_score(y_train, model.predict(X_train_standardised))
     def prec_train(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return precision_score(y_train, model.predict(X_train_standardised))
     def rec_train(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return recall_score(y_train, model.predict(X_train_standardised))
     def acc_test(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return accuracy_score(y_test, model.predict(X_test_standardised))
     def prec_test(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return precision_score(y_test, model.predict(X_test_standardised))
     def rec_test(c,k,g):
       model = SVC(C=c, kernel=k, gamma=g)
       model = model.fit(X_train_standardised, y_train)
       return recall_score(y_test, model.predict(X_test_standardised))
```

```
[16] # SET VARIABLES FOR TEST

     kernel_type = ['linear', 'poly', 'rbf', 'sigmoid']
     acc_value = []
     pre_value = []
     rec_value = []
     acc_testing = []
     pre_testing = []
     rec_testing = []

     # RUN TEST WITH VARYING KERNEL TYPES

     for i in range(len(kernel_type)):
       K = kernel_type[i]
       print('testing with kernel as {}'.format(K))
       SVM3 = SVC(C = 2.5, kernel = K, gamma = 'scale')
       SVM3 = SVM3.fit(X_train_standardised, y_train)
       ypred_val = SVM3.predict(X_val_standardised)
       ypred_test = SVM3.predict(X_test_standardised)
       acc_value.append(accuracy_score(y_val, ypred_val))
       pre_value.append(precision_score(y_val, ypred_val))
       rec_value.append(recall_score(y_val, ypred_val))
       acc_testing.append(accuracy_score(y_test, ypred_test))
       pre_testing.append(precision_score(y_test, ypred_test))
       rec_testing.append(recall_score(y_test, ypred_test))
```

```
     # DISPLAY ACCURACY PLOT

     plt.plot(kernel_type, acc_value, label='Validation')
     plt.plot(kernel_type, acc_testing, label='Training')
     plt.legend()
     plt.title('Accuracy Score vs Kernel Type')
     plt.xlabel('Kernel')
     plt.ylabel('Accuracy')
     plt.savefig('Accuracy Score vs Kernel Type')
     plt.show()

     # DISPLAY PRECISION PLOT

     plt.plot(kernel_type, pre_value, label='Validation')
     plt.plot(kernel_type, pre_testing, label='Training')
     plt.legend()
     plt.title('Precision Score vs Kernel Type')
     plt.xlabel('Kernel')
     plt.ylabel('Precision')
     plt.savefig('Precision Score vs Kernel Type')
     plt.show()

     # DISPLAY RECALL PLOT

     plt.plot(kernel_type, rec_value, label='Validation')
     plt.plot(kernel_type, rec_testing, label='Training')
     plt.legend()
     plt.title('Recall Score vs Kernel Type')
     plt.xlabel('Kernel')
     plt.ylabel('Recall')
     plt.savefig('Recall Score vs Kernel Type')
     plt.show()
```

```
[17] dfacc = pd.DataFrame()
     for n in np.linspace(250, 3500, 25):
       dfacc["{}".format(n/1000)] = [acc_val(n/1000,'linear','scale'), acc_test(n/1000,'linear','scale')]
     dfacc = dfacc.T   # tranpose
     dfacc.plot()      # plot graph
     plt.title('Accuracy Scores vs C-Values')  # plot graph
     plt.xlabel('C-Values')
     plt.ylabel('Accuracy')
     plt.show()

     dfprec = pd.DataFrame()
     for n in np.linspace(250, 3500, 25):
       dfprec["{}".format(n/1000)] = [prec_val(n/1000,'linear','scale'), prec_test(n/1000,'linear','scale')]
     dfprec = dfprec.T
     dfprec.plot()
     plt.title('Precision Scores vs C-Values')
     plt.xlabel('C-Values')
     plt.ylabel('Precision')
     plt.show()

     dfrec = pd.DataFrame()
     for n in np.linspace(250, 3000, 25):
       dfrec["{}".format(n/1000)] = [rec_val(n/1000,'linear','scale'), rec_test(n/1000,'linear','scale')]
     dfrec = dfrec.T
     dfrec.plot()
     plt.title('Recall Scores vs C-Values')
     plt.xlabel('C-Values')
     plt.ylabel('Recall')
     plt.show()
```

```
[18] # RUN TEST WITH VARYING C-VALUES

     acc1 = pd.DataFrame()
     for n in np.linspace(1000, 3500, 25):
       acc1["{}".format(n/1000)] = [acc_val(n/1000,'rbf','scale'), acc_test(n/1000,'rbf','scale')]
     acc1 = acc1.T
     acc1.plot()
     plt.title('Accuracy Scores vs C-values')
     plt.show()

     prec1 = pd.DataFrame()
     for n in np.linspace(1000, 3500, 25):
       prec1["{}".format(n/1000)] = [prec_val(n/1000,'rbf','scale'), prec_test(n/1000,'rbf','scale')]
     prec1 = prec1.T
     # prec1.plot(label = ['Validation', 'Training'])
     prec1.plot()
     plt.title('Precision Scores vs C-values')
     plt.show()

     rec1 = pd.DataFrame()
     for n in np.linspace(1000, 3500, 25):
       rec1["{}".format(n/1000)] = [rec_val(n/1000,'rbf','scale'), rec_test(n/1000,'rbf','scale')]
     rec1 = rec1.T
     rec1.plot()
     plt.title('Recall Scores vs C-values')
     plt.show()
```

```
[19] model = SVC(C=2.7, kernel='linear', gamma='scale')
     model = model.fit(X_train_standardised, y_train)
     print('Accuracy_Val: {}'.format(accuracy_score(y_val, model.predict(X_val_standardised))))
     print('Precision Val: {}'.format(precision_score(y_val, model.predict(X_val_standardised))))
     print('Recall Val: {}'.format(recall_score(y_val, model.predict(X_val_standardised))))
     print('Accuracy Train: {}'.format(accuracy_score(y_train, model.predict(X_train_standardised))))
     print('Precision Train: {}'.format(precision_score(y_train, model.predict(X_train_standardised))))
     print('Recall Train: {}'.format(recall_score(y_train, model.predict(X_train_standardised))))
     print('Accuracy Test: {}'.format(accuracy_score(y_test, model.predict(X_test_standardised))))
     print('Precision Test: {}'.format(precision_score(y_test, model.predict(X_test_standardised))))
     print('Recall Test: {}'.format(recall_score(y_test, model.predict(X_test_standardised))))
```