

Predicting House Prices in Melbourne, Australia

Final Project Report by Muhammad Ahmad

Background

“Melbourne real estate is BOOMING. Can you find the insight or predict the next big trend to become a real estate mogul... or even harder, to snap up a reasonably priced 2-bedroom unit?” The goal of this project is to get hands-on experience with the entire Data Science process of exploring and then cleaning data and then applying various Machine Learning techniques to make predictions. A dataset obtained from Kaggle on house prices in Melbourne, Australia will be used for the task of predicting housing prices. The dataset was the result of web scraping various sources on the internet and it thus consists of many challenges and will require a thorough cleaning process before any Machine Learning can take place.

DATA EXPLORATION

The dataset has 13580 instances/observations and 21 features/variables. 1 of those features, the Price column is the target feature. In the background section, we had a brief introduction to the features in the dataset but let's now take a deeper look. Let's detail each feature a bit more to truly understand the data we will be working with.

The most important feature of all is the target feature **“Price.”** It is the price point the house was sold at. **“Suburb”** is the name of the suburb where the house is located. **“Address”** is the street address of the house. **“Rooms”** is the number of bedrooms in the house. **“Type”** is the type of the house, for example is it a House or Townhouse or an Apartment type (usually referred to as “Unit” in Australia), etc. **“Method”** is a feature describing the status of the property sale. **“SellerG”** is the Real Estate agent or company that sold the house. **“Date”** is the date the house was sold in, in Day/Month/Year format. **“Distance”** is the distance between the house and its nearest Central Business District (CBD). **“Postcode”** is the Postal Code of the address of the house. **“Bedroom2”** is the same as the Rooms column, but since the data was web scraped from the internet, different sources were used and hence the dataset includes two columns aiming to refer to the same feature value. Unfortunately this may mean there will be data inconsistencies within the dataset. **“Bathroom”** is the number of Bathrooms in the house. Luckily this only came from one source! **“Car”** is the number of spots the house offers for cars (parking spots). **“LandSize”** is the size of the outer land area that comes with the property, such as space for gardening for example. **“BuildingArea”** is the building size, i.e. the enclosed space covered by the property. **“YearBuilt”** is the year the house was originally built. **“CouncilArea”** is the name of the local government area near the house. **“Latitude”** is the Latitude coordinates. **“Longitude”** is the Longitude coordinates. **“Regionname”** is the general region the house is located in. **“Propertycount”** is the number of properties that exist within the suburb that the house is located in.

Now that we have the basic description of each feature, it will be worthwhile to have a preliminary discussion on how important each feature is in relation to our ultimate task of predicting house prices. This will only be to have a general idea of the data and the assumptions here may be wrong or mislead. The suburb may very well be an indication of the price as generally neighborhoods tend to have similar priced houses. The address feature is the most specific indication of the location and may be too specific to give an indication of the price. The number of rooms (given by either the “Rooms” or “Bedroom2” column) will be one of the strongest indications as well as the number of bathrooms. The type of the house will also be a strong indication. It makes sense that an apartment home is worth a lot less than an entire house for example. The Method column will require more probing on the dataset. The date column may hold a lot of important details that can be derived such as the time of the year (season). There

are a few things that can be said about the year the house was sold in and its relation to the general economy. It does not make much sense for the seller to hold any significant impact on the final price of the house. The distance column will also be a strong indicator as it is expected houses closer to the CBD will have higher demand. The postcode seems to be another way of indicating the general address of the house which may be done by the suburb already. It would make sense for the number of car spots to have a similar effect as the information about the rooms. Both LandSize and BuildingArea should be very important in determining the price. For the year the house was built, we'd expect newer built houses to be more expensive. Although this is true, it is important to note that older houses can simply have been renovated a lot thus increasing their prices and the year built may not be as indicative of price. There is no information available about renovation in the dataset. The council area may also be repeating the same information as the suburb. The longitude and latitude should have an effect on price, especially considering they affect weather. They also may just be better than the address column and that column may not be needed. Regionname is the most general address of the House thus may be all we need in terms of location or may not be specific enough to say anything about the price.. The "Propertycount" that is related to the suburb may give another indication of the house price but may be duplicating information available from the suburb column.

According to the Pandas data report, the dataset consists of 8 Categorical and 13 Numerical features/values (including the Price target feature). This is important to take note of, we will need to convert between the two types as appropriate.

The numerical columns include: Rooms, Distance, Postcode, Bedroom2, Bathroom, Car, Landsize, BuildingArea, YearBuilt, Latitude, Longitude, Propertycount.

The categorical columns are: Suburb, Address, Type, Method, SellerG, Date, Council Area, Regionname.

Most of this makes sense, however the Postcode should really not be considered a numeric value, it is actually more of a categorization of the location of the house much like Suburb and Council Area, so should actually be treated as a categorical feature.

Duplication

According to the Pandas Profiling report there are no duplicate rows which is good. If there were, there would be no point in keeping them as they don't contribute in any way only at the expense of more memory and computation. However, from the report we do see that there are specific column values that do repeat. This makes sense for most values such as Number of Bathrooms or Year Built but not so much for the Address value column as each address is unique to each house. It will be worthwhile to investigate further as to why this feature has repetition. Running some code, we see that 202 rows have a duplicated Address. Let's explore exactly what is going on here, by focusing on a few addresses that were repeated. For example, we ran two queries to see all rows where the address "36 Aberfeldie St" and "4 Bell St" were duplicated.

Upon further investigation we realize there is nothing unusual for addresses to be repeated a few times for our dataset. They are simply different sales at different points in time. This makes sense as we are not simply interested in houses but more specifically the sale of a house. An important insight from this is that the price will not simply be affected by details intrinsic to the house but also details of factors external to the house itself such as the date being sold and other factors that change without the actual house being changed. For example, the season (e.g. Winter) the house was sold in could affect or tell us something about the price.

Running the queries, another interesting fact and seemingly anomaly was discovered. The house at "4 Bell St" appears twice in the dataset but with different suburbs. It is not clear at first if this is a data entry error or something else is going on. At first glance, it makes sense to think that if a house is repeated in the dataset, other features that

are intrinsic to the physical house and not to a particular sale should not change. It will be worthwhile to investigate what exactly is going on with the Address column.

A custom function called “find_discrepancies” was developed and ran to see when Addresses are repeated what discrepancies are found that one would think should not be there. It turned out the code returned many cases where discrepancies occurred. Meaning rows of data that share the same address but differ in aspects such as Location or other features not tied to a sale. On further inspection it was realized that the “Address” feature is not unique to each house. It totally makes sense that two houses can have the same street address but are actually two different houses, an address is not always a unique identifier.

Correlations

One of the easiest and most general indications of a numerical value and its effect on the price is the correlation between the feature and price. As mentioned above, PostCode is not truly a numerical feature, but for the rest of the features, the Pandas Data report shows us the numerical correlation values for each variable. The higher the correlation with price the more important the variable will generally be in affecting the price. The report also produces a heatmap that makes it easy to visualize the correlations without the actual numerical values (Figure 1.) Ranking the features Rooms, Distance, Bedroom2, Bathroom, Car, Landsize, BuildingArea, YearBuilt, Latitude, Longitude, and Propertycount in terms of correlation we see that the most important features with positive correlation are Rooms, Bedroom2 and BuildingArea. Interestingly BuildingArea had the highest correlation of all with Landsize having only half as much! Rooms had higher correlation than Bedroom2 indicating it is the more accurate column for number of bedrooms. Latitude and Longitude although had smaller correlation, had around the same magnitude but opposite signs. This makes sense because lower latitude areas are warmer hence it makes sense that the prices would be higher there.

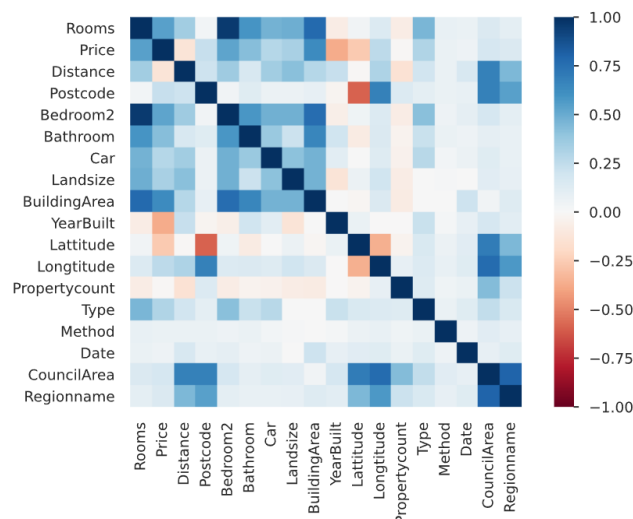


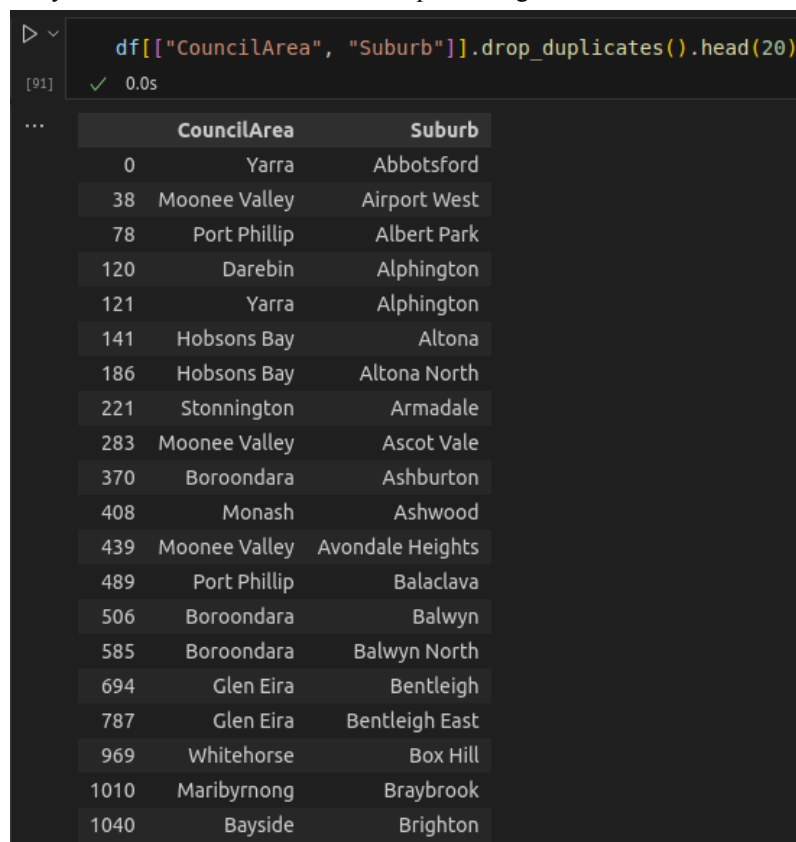
Figure 1: Correlation Heatmap

Categorical Variables relation to Price

To see how categorical variables are related to the Price target we have a few different options. First of all the categorical variables are Suburb, Address, Type, Method, SellerG, Date, Council Area, Regionname and then the PostCode which although is a numeric value is actually a category where the exact numeric value has no relation to the price

From all the categorical variables we expect Address and SellerG to have no bearing on the price. The address is too specific and does not provide any insight in the same way that the latitude/longitude coordinates and the general location attributes such as Suburb and Regionname do. The seller may provide a small amount of indication, for example certain real estate agents work in more expensive areas but this information can be given from the location attributes.

Furthermore, we run into problems with high cardinalities. As we will need to one-hot encode categorical variables, those with extremely high cardinality, meaning number of possible values will result in a huge number of features which will be problematic and lead to the well known issue of “curse of dimensionality.” According to the Pandas Data report some of the categorical values with high cardinality include Suburb. This means it will not be a good idea to use the suburb feature. However, we see the CouncilArea with a much smaller cardinality captures most of that information. For example we can tell how related the two are and find that generally a council area maps to many suburbs as shown in a few examples in Figure 2.



```
df[['CouncilArea', 'Suburb']].drop_duplicates().head(20)
```

	CouncilArea	Suburb
0	Yarra	Abbotsford
38	Moonee Valley	Airport West
78	Port Phillip	Albert Park
120	Darebin	Alphington
121	Yarra	Alphington
141	Hobsons Bay	Altona
186	Hobsons Bay	Altona North
221	Stonnington	Armadale
283	Moonee Valley	Ascot Vale
370	Boroondara	Ashburton
408	Monash	Ashwood
439	Moonee Valley	Avondale Heights
489	Port Phillip	Balaclava
506	Boroondara	Balwyn
585	Boroondara	Balwyn North
694	Glen Eira	Bentleigh
787	Glen Eira	Bentleigh East
969	Whitehorse	Box Hill
1010	Maribyrnong	Braybrook
1040	Bayside	Brighton

Figure 2: Relationship between Council Area and Suburb

Although intuitively it would be better to use Suburb as an indication of house pricing, the Council Area will still capture much of that information without having the trouble of the curse of dimensionality. Council Areas can also affect house prices due to politics and socioeconomics going hand to hand. But again, we will lose some

information, as one council area can encompass many Suburbs and hold houses of varying pricing. Similarly, from the Pandas report we find the Council Area is highly correlated with the PostCode hence can capture both the suburb and categorical postcode. The regionname is the most general of all. It will be good to keep region, council area and distance from CBD as three important location attributes that capture most of what we need.

Running some descriptive statistics such as mean, median, standard deviation, and variance of the price per categorical feature we can get a lot of information about how important each feature will be and can find other insights (see Figure 3 for an example with the type of house). Council Area and Region were both found to be somewhat indicative of pricing. As expected the type of the House is a strong indicator of Price, showing Apartment types (with a “u”) have lower pricing. On the other hand, the method seems to not have much of an influence on the price at all, with only one type of method having drastically higher average price, indicating that the feature will not make the biggest difference in determining price.

The date column will require more extraction from it to see if it will be of value. This can be covered in the cleaning process.

	mean	median	std	count	var
Type					
h	1.242665e+06	1080000.0	668078.742092	9449	4.463292e+11
t	9.337351e+05	846750.0	395038.245773	1114	1.560552e+11
u	6.051275e+05	560000.0	260987.452871	3017	6.811445e+10

Figure 3: Descriptive statistics of price per type of house (h = house, u = unit, t = townhouse)

Missing Values

In the dataset 4 columns are missing values. Some are missing a few values, but unfortunately the BuildingArea feature which was previously found to be one of the strongest indicators of price is missing almost 48 percent of the time. The year built although most likely not as important is also missing about 40 percent.

Outliers

We will need to ensure any extreme outliers are taken care of. Not all outliers will be bad as the housing market is diverse and can accompany a large range of houses but there can be no values that are clearly out of the ordinary. The Pandas data report makes this analysis very easy for us, showing extreme values and their frequencies for each feature. We will only look at features that we considered important from the previous analysis.

The report alerts us that “Landsize” and “BuildingArea” is highly skewed, indicating that they may have outliers (Figure 4.) Upon further inspection these two columns do have extreme outliers that don’t seem accurate. Land Size has many 0 values but that is actually expected as it is not unusual for a house especially units to not have any outdoor area to it. However, around after the top 5 percent of the Land Sizes range have values that are too high compared to the rest of the distribution. BuildingArea for example has one very extreme outlier with a value of 44515 where even the second highest value is only 6791! BuildingArea also includes a few outliers on the lower end with a value of 0. Unlike LandSize it does not make sense for a house to have a BuildingArea of 0. They also have other smaller values ranging from 1 to 11 which also seem too low and inaccurate.

The YearBuilt has an outlier year of 1196 which is clearly an inaccuracy. The Bedroom2 column holds an outlier of 20 bedrooms which is significantly greater than the other values. This is another reason why this column seems to be

less accurate compared to the Rooms column. Bedroom2 also lists a few houses with 0 bedrooms where the Rooms column doesn't. The Bathrooms column also has outlier values of 0 and some higher values such as 5 and higher.

The Price column has outliers on the outer edge that are most likely data entry errors. A simple google search on the highest Price in the dataset showed that there was an extra 0 added by accident. This is a clear indication that the outliers should be treated. However the 2nd highest priced house which may also seem like an outlier is most likely correct and has a more reasonable sounding BuildingArea and Landsize to go with it. This means when we deal with outliers we may have to deal with them a bit more carefully.

Landsize is highly skewed ($y1 = 95.23740045$)

BuildingArea is highly skewed ($y1 = 77.69154092$)

Figure 4: Highly skewed features

Data Cleaning

The data cleaning step will now take the observations in the exploration phase and simply apply cleaning where necessary.

Removing Features

It was found that some features are not very useful, so we can get rid of the features: 'Suburb', 'Address', 'Method', 'SellerG', and 'Postcode'. 'Propertycount' was found to have a low correlation with the target so was also removed. 'Bedroom2' was previously found to not be as accurate as the 'Rooms' column so was also removed. It would not make sense to keep two features that are trying to say the same thing but are often in conflict with each other.

Missing Values

Next missing values were dealt with. We used a combination of KNN Imputation and Median Imputation. Ideally KNN Imputation is better because it has the chance of the most accurate prediction for a missing value. Median Imputation could possibly distort the dataset and reality as the actual value that for various reasons is missing in the dataset could be completely different from the median value. Median Imputation was used for categorical features or for features that aren't highly correlated with any features. KNN Imputation really depends on the assumption that there are similar data points related to that feature. If a feature does not have high correlations with other features it is likely KNN Imputation will not be as effective.

For 'CouncilArea' we ideally wanted to do KNN Imputation but since it was a categorical value we simply used the "Missing" String to replace rows where it was missing. The decision to not replace it with the most common value was simply for the benefit of avoiding distorting reality. Adding additional information to the dataset compared to the possibility of distorting should be preferred.

'BuildingArea' was filled in using KNN imputation and used the 'Rooms', and 'Bathroom' columns to see its nearest neighbors. This makes sense because these 3 variables were highly correlated with each other and it makes sense that generally properties with higher living areas are bigger because of more rooms. Similarly the 'Car' feature also used KNN Imputation with the 'Rooms', and 'Landsize' columns because they were the most closely related.

The 'YearBuilt' Column was not found to be significantly correlated with any features and this makes sense because most features in the dataset do not say much about the age of a house. For this we imputed it with the Median value of 1970 that was in the Pandas Profiling report.

Outliers

Next we dealt with any outliers in the dataset. The Pandas Profiling report was used extensively for this to probe the data. The general philosophy that was undertaken in this stage in general was to use Domain Knowledge about housing in Melbourne, Australia and also use the fact that this data was web scraped off of the internet. Some google searching was used to help guide decisions, specifically in seeing when values truly are outliers and when some outliers are accurate. A combination of completely removing values/rows and replacing was used. Generally, we do not want to remove rows because we would like to feed Machine Learning Models as much information as possible. However, changing values can also be detrimental as it can distort reality and create more noise in the dataset than necessary. This is where some of the domain knowledge and internet searching was used: to guide us to the best decision as much as possible.

There was a clear data entry error in the 'YearBuilt' column where the entry was 1196 and the entire row was simply removed. The reason for this is that there was no other way to accurately replace the value, and since it was only 1 row, it would not cause too much loss in the dataset. The 'Bathroom' column had values of 0 on the lower end and values above 5 on the higher end. Looking at the frequency of these values and general trend, it was clear these values are out of the ordinary and simply replaced with the closest, more realistic value.

Outliers in the 'Price' column had to be dealt with very carefully. It did not seem like a good idea to change the prices because as the target feature, price is very important and can not be distorted. It was decided best to remove these values but in a sensible way. We want to remove outlier values but too much to the point where we are out of data. Prices of 235000 on the lower and 5000000 upper end were found to be good cut-offs in terms of number of rows that can be removed and values that were clearly abnormal. A similar thought process as for 'Price' was carried out with the 'BuildingArea' feature removing rows where the value was 10 or less or more than 6091. For the 'LotSize' column it was realized that there is nothing unusual for sizes of 0 but the upper end outliers were problematic. A LotSize of 4000 or more was found to be a clear anomaly.

After the cleaning process 13359 out of 13580 data instances remained. Meaning around 1.6% of the data was removed. Overall, this seems like a good choice as it gets rid of noise in the dataset without clipping away too much data that we need to make solid predictions.

Feature Conversion

Next is the step of converting between numerical and categorical variables. The 'YearSold' column was transformed to extract the season of the sale (Spring, Summer, etc) and the year of the sale. The season was one-hot encoded. The season of the sale can very well affect the pricing of houses as can the year. The 'YearBuilt' column was removed and instead we extracted the age at time of sale of the house. This should be all we need from the 'YearBuilt' column as all we are interested in is the age of a house and generally expect newer houses to sell more at any point in time. Figure 5 is a visual of these changes.

YearSold	Season_Fall	Season_Spring	Season_Summer	Season_Winter	AgeAtSale
2016	0	0	0	1	46.0
2016	0	0	0	1	116.0
2017	0	1	0	0	117.0

Figure 5: Data after transforming 'Date' and 'YearBuilt'

The 'Type' of the house was also one-hot encoded. In the Exploration phase both CouncilArea and general Region were also considered to be important, one as a general indication of location and one more specific, so both were kept and one-hot encoded. The CouncilArea did lead to a huge increase in the number of features but that should not be a huge problem as the total number of features (60) is generally not considered to lead to the curse of dimensionality.

Future Transformation

At this stage, we decide if we want to scale our features or not. Generally, it is a good idea to scale features. This is especially important for machine learning models that are sensitive to different features having different scales. Also some models are not negatively or positively affected by feature scaling so it does not hurt us in any case.

For this data set, we decided to go with normalization over standardization. The choice between the two is often tricky with no seemingly clear answer. In this case, we decided to go with normalization because it does not assume

any underlying distribution on the data unlike standardization. A usual benefit of standardization over normalization is that it is not sensitive to outliers but this is not a huge problem for us as we already covered dealing with outliers beforehand.

Model Development

Once we have gone through the dataset and cleaned it, it is time to start with actual Machine Learning to start making predictions. In this case, we decided to go with 4 different machine learning models: Linear Regression, Decision Tree Regressor, KNN Regression, and the XGBoost Ensemble model.

The choices made were first and foremost a consideration of diversity. We want to experiment with a variety of types of Machine Learning models, in this case an Error based model (linear regression), a similarity-based model (KNN), a tree-based model and an Ensemble model (XGBoost) .

Each type of model usually has multiple different models to choose from. This can be a tricky choice to make between models but the general philosophy was to pick models that are generally known to be well performing. For example, for the choice of which Ensemble model to pick, we decided to go with boosting over bagging. The reason for this choice was that overall boosting has a better reputation for being more accurate.

Initial Development

To get the basic idea of what to expect we ran the 4 models on a basic 80/20 Train/Test split. Meaning for each model, we first trained the model on the same data that was 80 percent the size of the original dataset, and reserved 20 percent to examine how well each model performs on unseen data. Overall, XGBoost performed the best and Linear Regression performed the worst. The exact results are given in table 1.

Model Name	R2	RMSE	MAE
Linear Regression	0.650	371,744.62	251,502.86
Decision Tree	0.655	369,241.24	225,592.91
K Nearest Neighbor	0.672	359,920.50	228,576.794
XGBoost	0.836	254,325.5	160,174.89

Table 1: Accuracy Metrics for Initial ML Models

Learning Curves

After the initial modeling, it is going to be useful to see if a model is overfitting or underfitting. This will give us an idea of what is going well and not so well for a model and also helps push us towards the direction of improvement for each model. Figures 6-9 display learning curves for each model. These learning curves essentially plot an R2 score as an indicator of performance on the y-axis and the data the model is using on the x-axis. The end result is for us to see how the model is performing as the number of data points increases for both the training data and testing data. Ideally we expect the performance on both training and testing data to increase the size of data increases. However, it is possible we don't see such results and instead find a model that is either overfitting or underfitting.

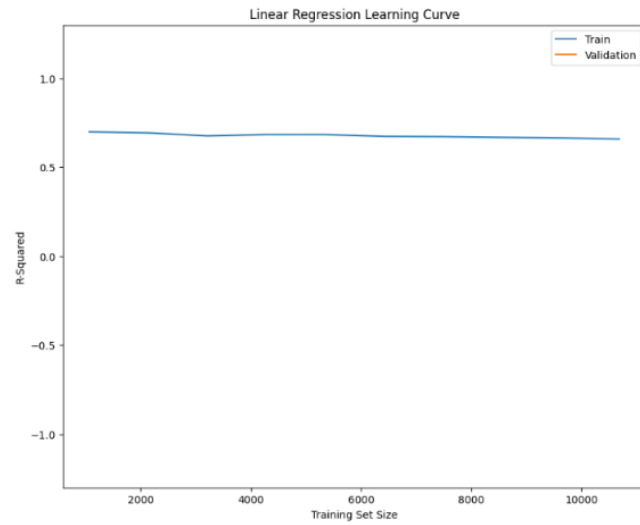


Figure 6: Learning curve for Initial Linear Regression model

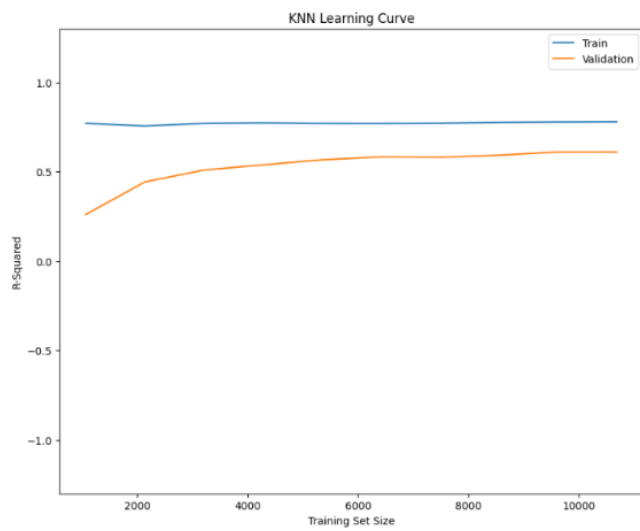


Figure 7: Learning curve for Initial KNN model

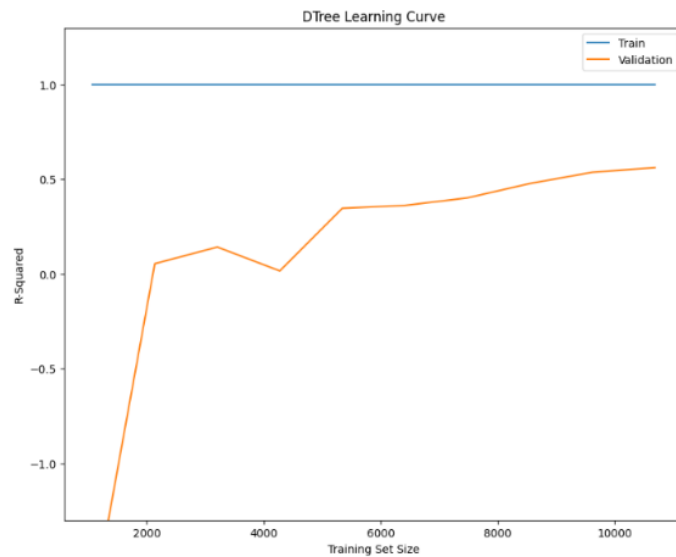


Figure 8: Learning curve for Initial Decision Tree Regression model

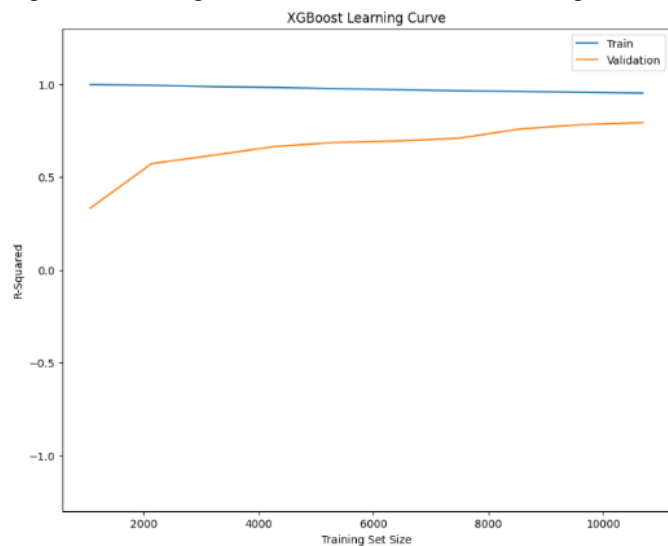


Figure 9: Learning curve for Initial XGBoost model

What we find is that the Linear regression model was underfitting with mediocre performance on both the training and test set. Actually the score was so similar to the point that the learning curve seems to have only one line but there's actually two. On the exact opposite end, we see that the decision tree regressor was overfitting with near perfect performance on the training set, and bad performance on the testing set. Knn seemed to not either have a huge overfitting or underfitting problem but it is more likely a case of underfitting as even the performance on the training set was not that good. XGBoost as the best model had a solid learning curve. With really good performance on the training set and also good performance on the test set.

Improving Models

All hope is not lost and we can still improve models through various techniques. How exactly to improve will be guided by the insights given from the learning curves. The general strategy was hyperparameter tuning through grid search with a cross validation of 5 folds. Essentially how this would work is a "grid" is made that enumerates different possible combinations of the hyperparameters of the model. Each variation is then taken through the process of cross validation where the training set is divided into 5 parts (5 folds.) At each iteration of the 5 total iterations one fold is treated as the test set and the remaining are treated as the training set. The model is evaluated on the test set. At the end, the hyperparameter combinations that gave the best average result on the test sets in the validation process is declared the best with that grid.

From Linear Regression (LR) we see that since the model was underfitting and LR is one of the simpler models with not many hyperparameters so not much can be done here. Usually if L.R is showing overfitting, different regularization techniques are taken. However there would be no point in doing that here. However, to test out the theory we still did grid search including multiple types of LR with some hyperparameters. Table 2 below shows the results. Clearly LR without any regularization actually slightly performed better than with regularization which makes sense given the initial learning curve.

Model Type	R2 Score
Linear Regression w/o Regularization	0.650107
Ridge Regression	0.649815
Lasso Regression	0.649933
Elastic Net Regression	0.649933

Table 2: Comparisons between various Linear Regression techniques

It is interesting to see how LR was not effective for a regression problem. We can analyze this a bit further. Most likely, the assumptions of Linear Regression were not met. LR can be a strong model but it makes certain assumptions about your data set. One of those is linearity: it assumes a linear relationship between the target variable and other features. Another assumption is homoscedasticity. We can actually find if this assumption was met or not by plotting a residuals vs predictions graph, which is what the model predicted for each data point on the x axis and the difference between actual price and predicted price on y axis. From Figure 10 we can see that this assumption was not met. The graph indicated Heteroscedasticity and not Homoscedasticity. Essentially the graph shows a generally increasing pattern as prediction increases whereas with homoscedasticity we usually see a graph with random scattering of points with constant variance. Therefore we can conclude that this dataset was simply not a good match with Linear Regression.

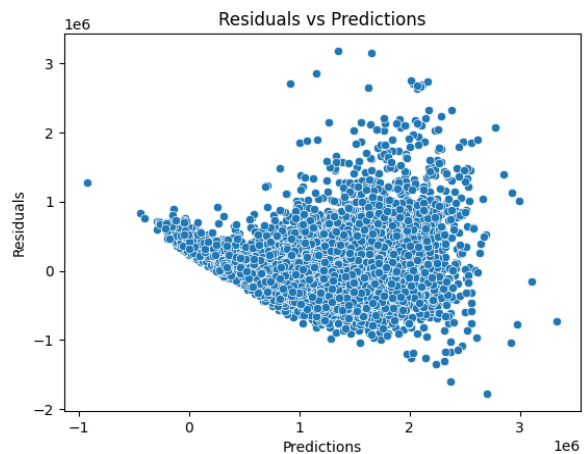


Figure 10: Plot of Residuals against Predictions

For the remaining models, the previously mentioned strategy of hyperparameter tuning was undertaken. There is not much more to be said so we can say that KNN, Decision Tree regression and XGBoost hyperparameter tuning was all done in a similar way. Essentially we build a grid, do 5 fold cross validation and then get the best possible hyperparameters. Since this can be a long process as the computer will run models over and over again, some techniques were taken to enhance efficiency and reduce computation. Instead of running a huge grid with all possible values for each hyperparameter, Initially, a broad range of hyperparameter values were tried first using smaller, more manageable grids. This allowed for a preliminary understanding of which ranges held the most promise. And then we narrowed down the grid searches with more specific values until we found the best possible hyperparameters. Another note on this: there was an additional step for Decision Tree regression. Before running the grid searches we took an iterative approach to first find the best `ccp_alpha` value which determines how much pruning is done. Remember our initial Decision Tree regression was overfitting therefore it was important to add pruning. All models were improved and this XGBoost comes out as the best model overall.

Figures 11-13 show the initial learning curves (which were also shown before) on the left compared with the learning curves of the final/best models that were found through hyperparameter tuning on the right. This gives us a good idea of what problems we were able to fix and not. KNN performance became almost perfect on the training set and slightly improved on the testing set, essentially having the best KNN model have even more overfitting than before. For DTree we were able to solve the overfitting issue and improve performance on the test set. XGBoost improved even more and should be the model we chose.

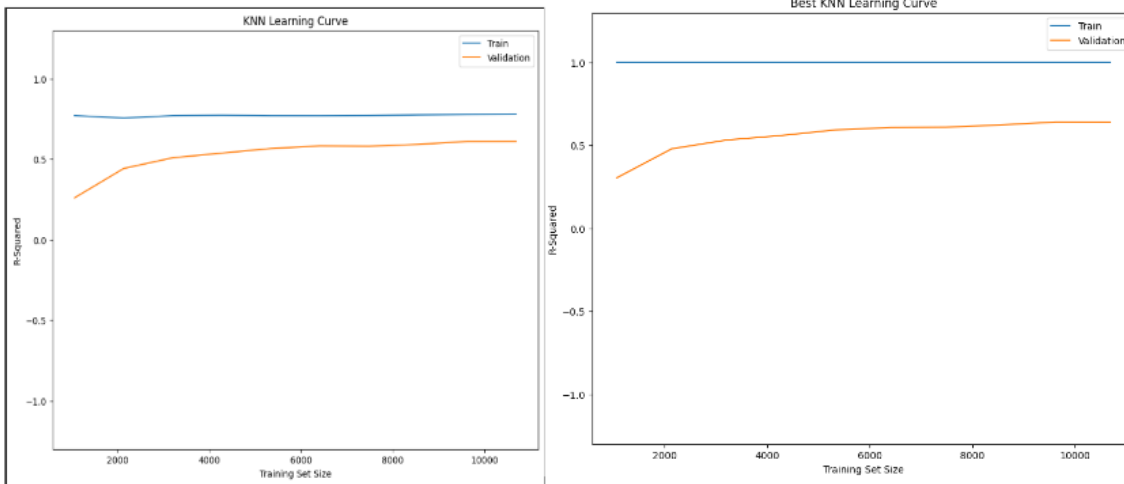


Figure 11: Before and after performance for KNN

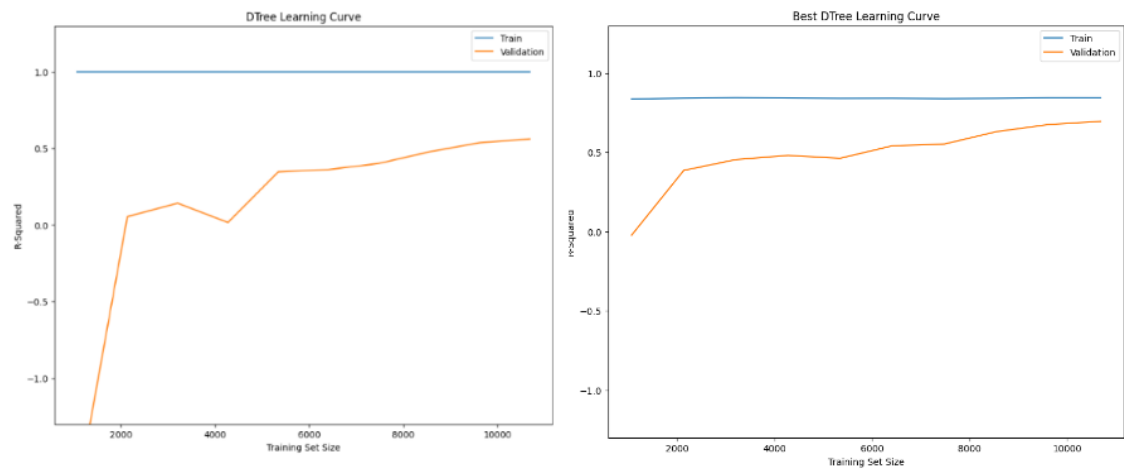


Figure 12: Before and after performance for DTree

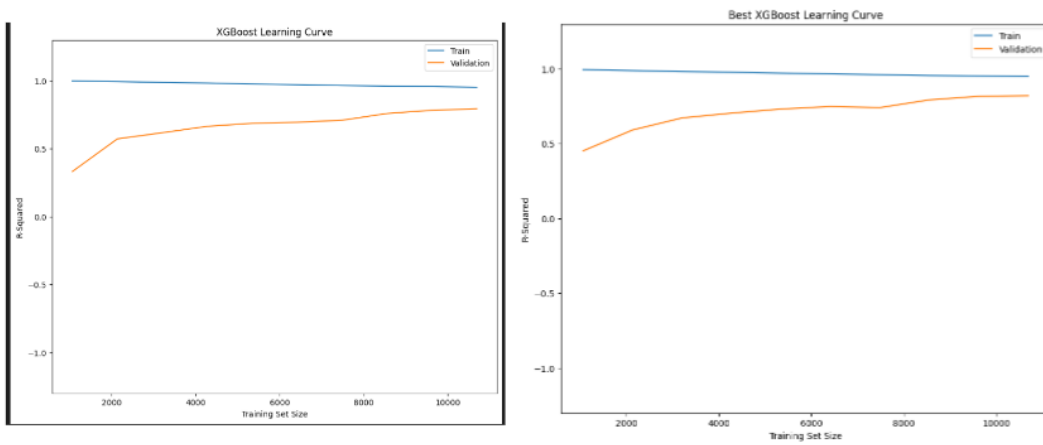


Figure 13: Before and after performance for XGBoost

Model Evaluation

As seen above, the best model turned out to be XGBoost. As it had the best overall performance on the test set and did not have any underfitting or overfitting issues. It had the best scores for each metric that we used: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Square Error (RMSE), and R-Squared (R2). However, all of these error metrics are different and much can be said about which should be used in general to compare models. RMSE should be preferred over MSE as they are essentially the same score that say the same thing but on different scales. RMSE is just easier to evaluate with it being on a smaller scale as it is the square root of MSE. Both MAE and RMSE are valuable scores but RMSE punishes predictions residuals that were a lot higher, thus might be preferable overall as we want to ensure to use the model that minimizes the chances of predicting with significant errors on any single data point. R2 is a general good score that is very easy and straightforward to interpret and assess the general accuracy of a model but may be discouraged to be used to compare models with different numbers of predictors or features as R2 is said to increase as more predictors are added to a model. Nonetheless, it can be used as a quick way to assess performances but best used along with MAE and RMSE.

In conclusion, XGBoost was the best model and should be the one deployed in a real world application. The other models were simply not appropriate for this data set. Linear regression should especially be avoided for this dataset as the data clearly did not meet the assumptions of the model.

Conclusion

Overall, this project was a great introduction to the overall process of real world machine learning from data cleaning to modeling to make predictions. Many challenges were encountered. First and foremost, data cleaning turned out to be especially challenging. This is because the original data was gathered through various sources from the internet and web scraped. Therefore it had a lot of noise and seemingly messy data points where it was not clear when there were data entry errors and not. Even that aside, data cleaning is tricky because there are so many options and techniques to choose from and it is not always straightforward to pick just one.

Model development was not as challenging, however it was tricky to see why certain models were not performing as well as others. Even now there is no way to give a surefire answer as to why a certain model could not perform as well as another. However, good news is we were able to find a model (XGBoost) that performed decently well on the dataset with a final R2 score of 0.849.

The next step would be to deploy the XGBoost model in a real world application so it can be used by the various groups mentioned early on as a source of future house price predictions in Melbourne, Australia. Before deploying the model it would be wise to first test it out on some recent unseen data points. Meaning, it would make sense to see if the model does a good job of predicting prices of sales around the date that the model will be deployed. This would make sure that the deployed application will continue to work well.

Additional improvements could include the need for cleaner and more data to the dataset. We believe that while the project was overall a success, it was seen in the data cleaning phase that the data was not particularly clean. More and cleaner data could certainly help in finding an even better model or improving the current model. Nonetheless, the project overall was a success.