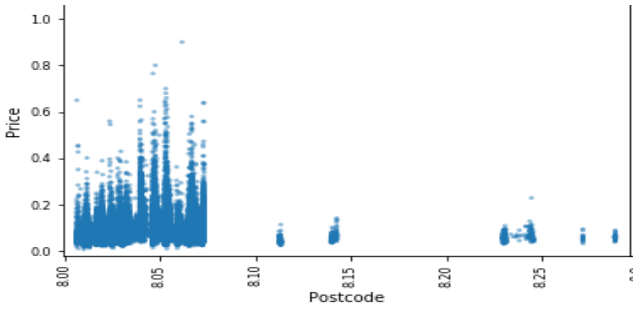
Q1. The entire dataset is comprised of 21 variables with 8 variables as characters or strings. Before moving onto the effect on the target variable (Price) by those 8 variables, the following table gives a summary statistic of some numeric variables. Every row in the dataset denotes information about property (house, townhouse etc) , price it is sold for, seller information and more characteristics such as number of bedrooms, location (latitude, longitude etc)

1. Location based Features –

The entire area is divided into many Regions (Regionname). Each Region in turn is divided into Council areas (CouncilArea) and further Suburbs. Postcode is assigned to every Suburb.

Postcode: I have removed the row containing null value. Postcode gives similar information like other location-based features. To better understand it's results (*Fig 1*), I transformed the price and postcode variable. However, I didn't find any significant relation worth utilizing in my model.



Council Area*: Governing council for the area. There are three null values and as the variable is a string, it cannot be determined and its best if remove the rows. We can see that Council Area has a total of 33 categories which I convert to categorical variables for future modelling. There is large standard deviation in a few categories such as in Manningham City Council, which has portions lying into two Regions. This might be the reason for abrupt standard deviation. Similar trends can be seen in other categoriesd like Nilumbik Shire Council. The Stonnington City Council and Boroondara City Councils seem to be the most expensive one

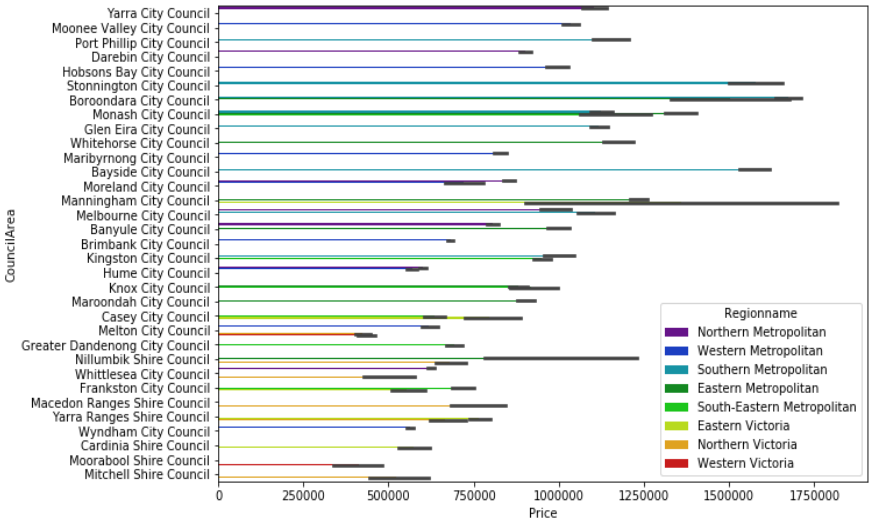


Figure 2

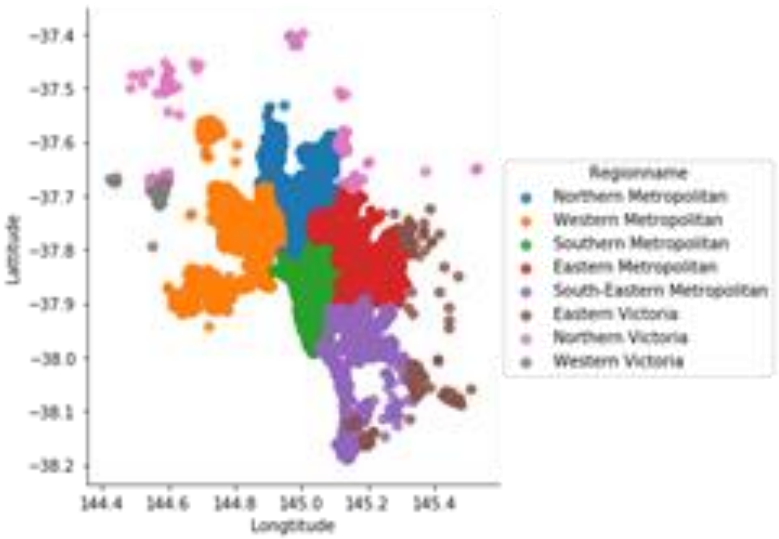


Figure 3

	Regionname	Car	Price	Rooms
0	Eastern Metropolitan	1.837103	1.108723e+06	3.364914
1	Eastern Victoria	2.042169	7.143282e+05	3.518072
2	Northern Metropolitan	1.583800	8.614840e+05	2.804552
3	Northern Victoria	2.108434	6.190512e+05	3.457831
4	South-Eastern Metropolitan	1.930597	8.773078e+05	3.245522
5	Southern Metropolitan	1.633036	1.395928e+06	2.901807
6	Western Metropolitan	1.786758	8.376153e+05	3.076526
7	Western Victoria	2.031250	4.326068e+05	3.312500

Table 1: Regioname and it's attributes

RegionName*: Different regions will have varied level of property prices as evident from the graph. There are three missing values which are strings and best if removed. There are 8 primary regions. I would choose to keep RegionName in case where I need fewer number of Features to capture the data. Convert to one hot vectors for future modelling. Fig 3 shows the regions in the Metropolitan Area. Table 2 shows that the categories NorthernMetro, South-Eastern, Western have similar means i.e they are similarly priced. Seems like Southern Metropolitan is the most expensive Region. Southern Metropolitan Region's high cost is associated with high activity in the region with maximum number of cars and rooms compared to rest. For this reason, maximum sellers are active in this region. The case is exactly opposite for Western Victoria as it is the least expensive region. *Table 2*

Address: Is a unique string for every property and there are no missing values. I haven't utilized this feature as every house would have a unique address, so treating this as a categorical variable seems futile.

Suburb*: Are neighborhoods and every region will have several of those. There are no missing values, so it is an import variable in our model. However, there are a total of 344 categories in this variable, converting this to one hot vectors would be computationally challenging to compute for algorithms like K-Means. Some suburbs sharing the same postal code have a higher number of properties than other, denoting more crowded neighborhoods then rest.

Latitude/Longitude: A geographic coordinate system is a coordinate system used in geography that enables every location to be specified by a set of numbers, letters or symbols. There are 7976 missing values which are filled by imputing the suburb average (as it has the most exhaustive number of categories) and assigned the mean values of each suburb to the row. This can be justified as they both are location-based features. From Fig 4 we can see that most of the metropolitan has similar prices besides the central regions which have a higher price. There is a peak in property price in when Latitude is close to -37.4 and longitude is close to 145. Location within the country as well as elevation above

ground plays a factor in determining the property and in turn, the house prices. I chose not to transform this variable. In some models I have normalized as Standard distribution.

Distance: As just one value is missing, we can remove the corresponding row. I chose not to transform it as it is already calculated from a point. As expected, the closer we get to the central business district (distance), the property prices get elevated with the major area in the scatterplot towards the left between Distance and Price. The post codes starting 4 are much further from the central business district which in turn results in decreased property prices. The major population of properties have distance from businesses, averaging at close to 10 units. (Fig 5)

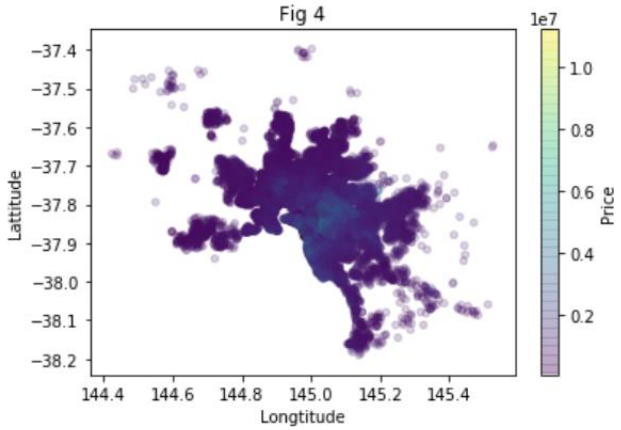


Figure 5

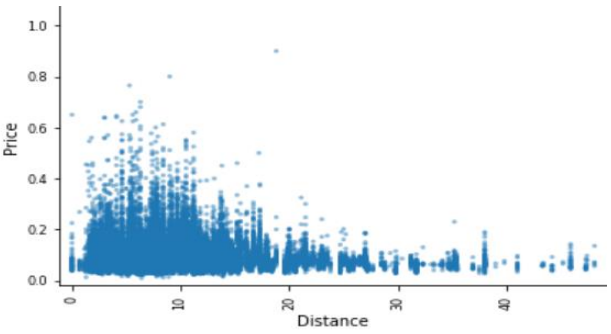
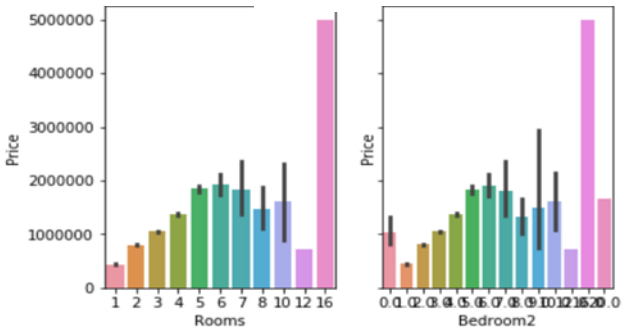


Figure 6

2. House based Attributes –

Rooms/ Bedroom2*:
There are no missing values in rooms. Number of Bedrooms with 8217 missing values. We compare the distribution to check the similarity. We can conclude that the distributions are similar, Hence I removed the bedroom2 feature. (Fig 6)



Type*:
br - bedroom(s); h - house, cottage, villa, semi, terrace; u - unit, duplex; t - townhouse; dev site - development site; o res - other residential. There are no missing values. In the figure we can see, that units have smaller number of rooms and have low price. We can see some cluster forming which can be help in the next question. (Fig 7)

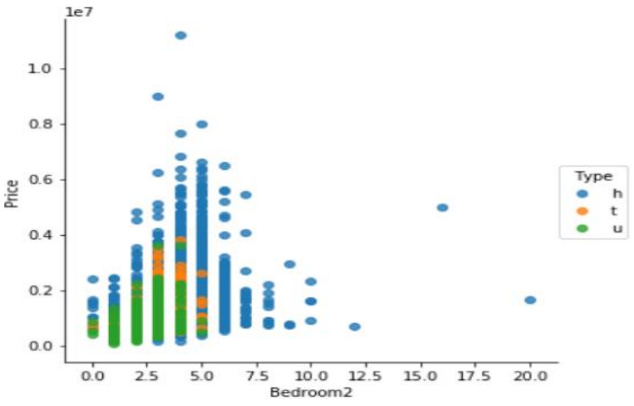


Figure 7

Bathroom: number of Bathrooms with 8226 missing values and can be imputed with mean value. Will use the original scale for analysis. Also, rounding of Imputed values to get whole values

Car*: Number of car spots with 8728 missing values and can be imputed with mean values. Treating as categorical. Also, rounding of

Imputed values to get whole values

Landsize: A geographic coordinate system is a coordinate system used in geography that enables every location to be specified by a set of numbers, letters or symbols. There are 11810 missing values and can be imputed with mean. It has a number of outliers because mean is 593.48893 and standard deviation 3757.26642. I have transformed the variable to log for the analysis. *Distance and log(Landsize)* have high close to the center of city (low distance) as we had observed earlier. (Fig 8)

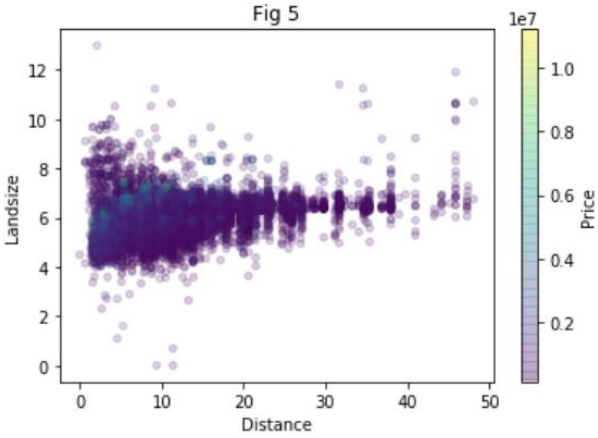
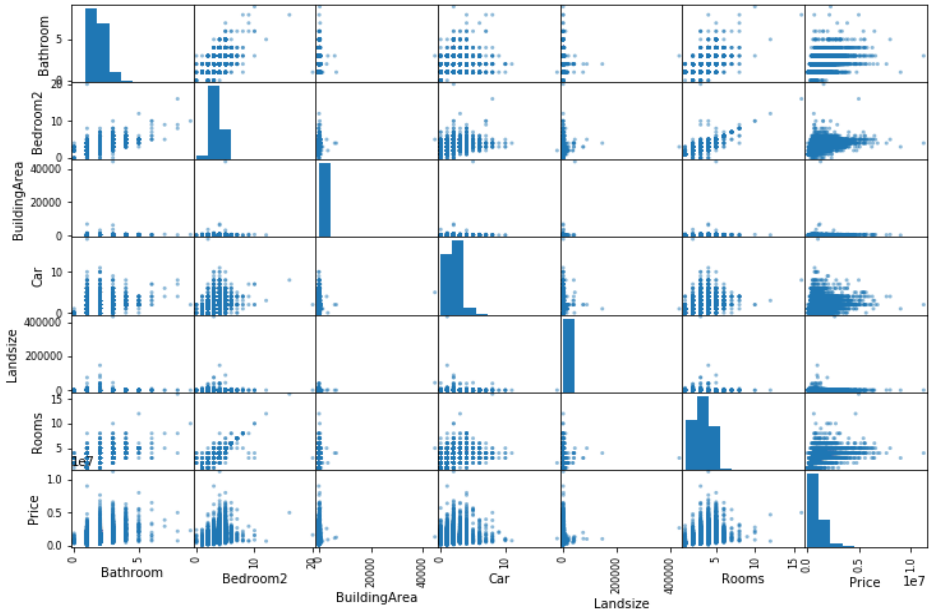


Figure 8

Building Area/ Year Built: BuildingArea and YearBuilt have 21115 and 19306 missing values which are more than 50 percent of the dataset. Imputing so many values would add bias to our dataset. So, we will remove these features.



Additional Observations-

- As expected, houses with higher number of bedrooms will have a higher number of bathrooms and vice versa to make it proportionate. Although, rooms bedroom etc don't have a strong correlation with prices as increase in number does not ensure a high price. Higher number of car spots is not associated with more landsize which means parking is a problem in most places. Although, properties with fewer car spots have a higher property price. It makes sense as houses are more expensive than apartments and have fewer parking spots.

3. Seller based and Historical Attributes –

Method*: S - property sold; SP - property sold prior; PI - property passed in; PN - sold prior not disclosed; SN - sold not disclosed; NB - no bid; VB - vendor bid; W - withdrawn prior to auction; SA - sold after auction; SS - sold after auction price not disclosed. N/A - price or highest bid not available. For analysis we will convert to one hot vectors

It seems that houses trail over townhouses and unit, respectively when talking in terms of properties sold, property sold prior, property passed in etc. Also, apart from unit houses, houses and townhouses tend to have higher bids than what it is actually sold for post auction. For some reason, units don't tend to go for high bids even though the properties sold have a higher price than the bids. There are no missing values. (Fig 8)

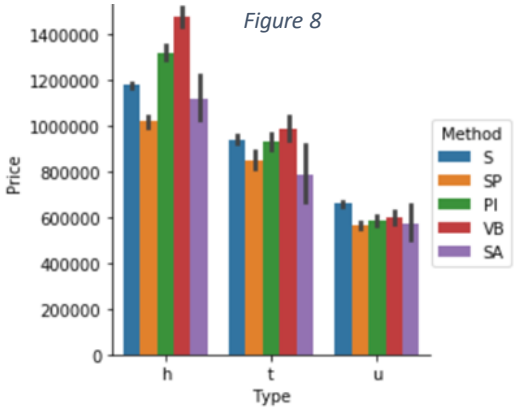


Figure 8

SellerG: This is a factor variable telling us about the seller's identity/name. It has 349 unique values, just like Suburb variable this variable with a lot of categories I do not think will be very helpful in regression so there is a possibility that we might have to remove it before applying regression. There are no missing values.

Property Count:
Number of properties that exist in the suburb.
The distribution of property count is slightly skewed with mean close to 5000 properties in an area. There is not a strong indication whether having large number of properties in a neighborhood will necessarily lead to higher prices as relation is not particularly strong.

(Fig 9)

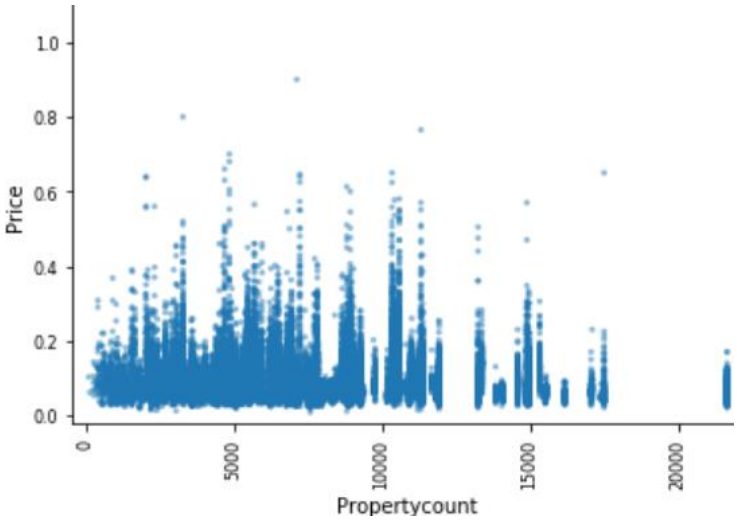


Figure 9

Year Built In: Year the house was built. There are three missing values which are removed along with Council and region name because all three of them have three missing values in the same rows and the council and region cannot be determined so missing of values of property count is eliminated. I haven't used Year Built in regression.

Date*: Date sold. There are no missing values. I have extracted the month of the year 2018 as everyone has almost similar years and transformation won't help. There's a higher change of trend in months due to business bonuses or discounts that we can catch. The month is a categorical variable converted to one hot vector

```
Suburb            0
Address           0
Rooms             0
Type              0
Price          7610
Method            0
SellerG           0
Date              0
Distance          1
Postcode          1
Bedroom2       8217
Bathroom       8226
Car            8728
Landsize      11810
BuildingArea  21115
YearBuilt     19306
CouncilArea       3
Lattitude      7976
Longtitude     7976
Regionname        3
Propertycount     3
dtype: int64
```

Table 1 Number of Null

*Converted to one hot vectors for regression

Q2. The next step would be to treat every row of the data as vectors in pursuit of finding properties which are similar to each other. The algorithm used to achieve this feat is K means Clustering. For each cluster, you will place a point(a centroid) in space and the vectors are grouped based on their proximity to their nearest centroid. The following steps are used in clustering-

Step 1: Cleaning Data

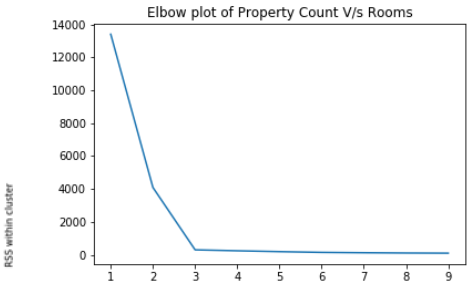
For clustering, your data must be indeed integers. Moreover, since k-means is using Euclidean distance, having categorical column is not a good idea. I normalized my dataset which would have a certain set of variables chosen. All the none values were either imputed or removed in the previous step. The variable 'Date' in its standard format will not be fed well in the model so I made a new column which encodes the month of the date present and dropped the former.

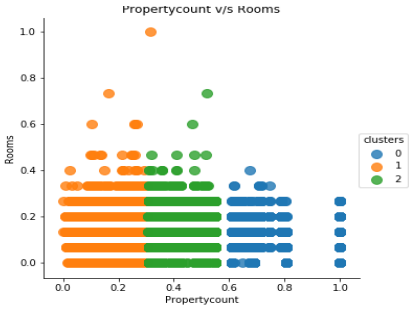
Step 2: Choose Number of Clusters and Standardize Data-

For choosing the number of clusters *k,* I use the Elbow method. Elbow method tries different values of k and plot the average distance of data points from their respective centroid (average within-cluster sum of squares) as a function of k. We look for the "elbow" point where increasing the k value drastically lowers the average distance of each data point to its respective centroid, but as you continue increasing k the improvements begin to decrease asymptotically. The ideal k value is found at the elbow of such a plot. I have also standardized the data to counter the high influence by variables having unusual scales then the rest.

Results –

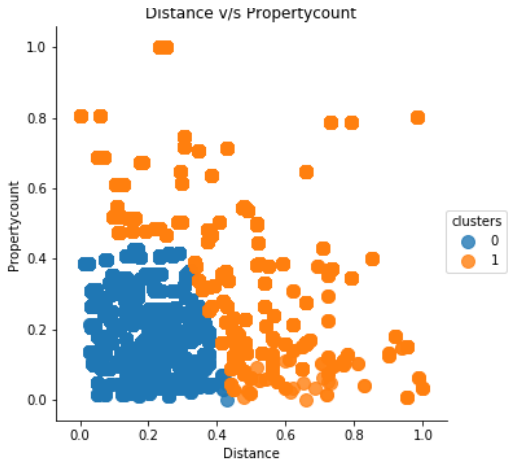
1. Features Observed – Property Count and Rooms





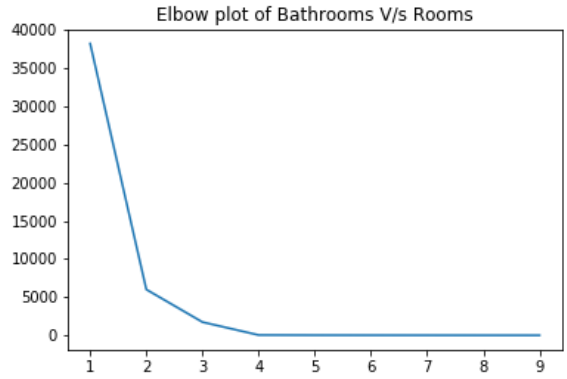
We can see the elbow point lies near k =2. We don't get good clusters visually but there is little overlap as result the RSS within cluster is low. The elbow plot is between *RSS within Cluster* and range of *k*. There is no direct relation between number of rooms and property count.

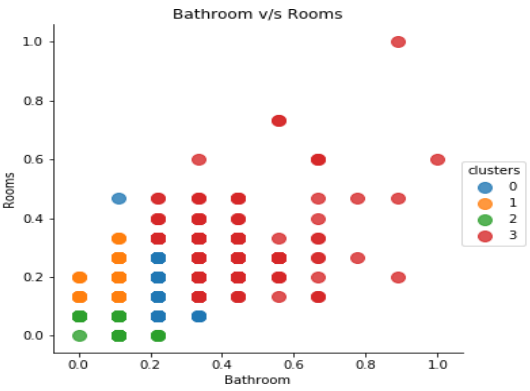
2. Propertycount and Distance-



The elbow point is 2. We see that the model can identify tightly packed cluster and loosely packed points well. Again there is no relation relation between property count and distance, which isn't useful in regression

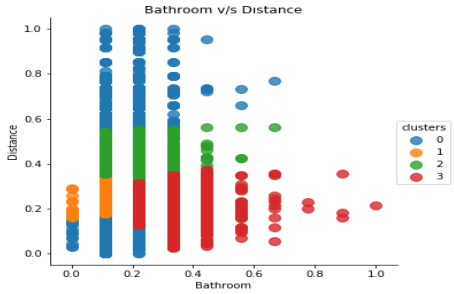
3. Bathrooms and Rooms-

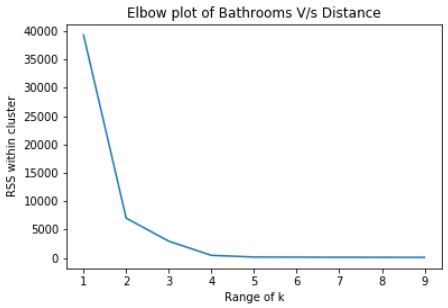




The elbow point is k =2. As we increase the cluster size, we get more number of clusters but it would result in "overfitting". We can observe that as No of rooms increase, the number of bathrooms increase too. This is something we observed in the previous analysis and is an important feature to use in regression.

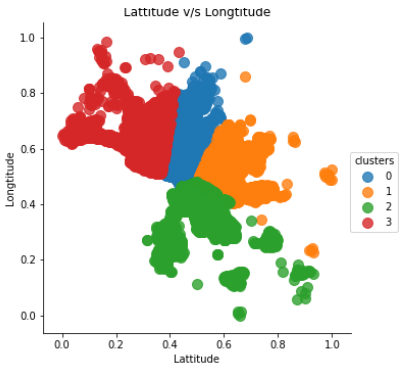
4. <u>Bathrooms and Distance-</u>

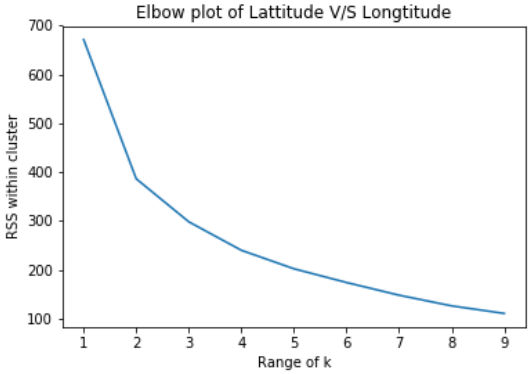




The elbow point is k =3. We can see that as the distance from center increases, we have more number of bathrooms, probably due to bigger size of homes. However this stagnates after a while as captured by Cluster 3

5. <u>Latitude and Longitude-</u>





The elbow point is 3 or 4. It's able to plot the landscape well despite transformation of the features. We can see that K means divides the area into multiple non-overlapping features. This can be important to regression as some distinction can be provide using the latitude and longitude values. However same features can provided by other features such as Regionname , council area etc.

OBSERVATION OF KMeans-

- We have performed Kmeans algorithm on a limited dataset. It was observed that performing K Means on larger dataset not only distorted clusters but also increased overall computation time
- It's important to determine how to choose the cluster size *k*. We have used the elbow method, but it doesn't automatically select the value. We have to select for it which isn't the most efficient way.

Q3. **Penalized Linear Regression model:**

Following pre-processing steps were performed for all the models

- No scaling has been performed on the dataset. The RMSE values reported are of original scale.
- We selected *Room, Price, Distance, Bathroom, Car, Landsize, Lattitude, Longtitude, Propertycount, CouncilArea, Month, Method, Type, RegionName* as features
- Some of the categorical variables like *Month, Method, Type, RegionName* was one hot encoded.
- Train/Test set split is 80%/20%

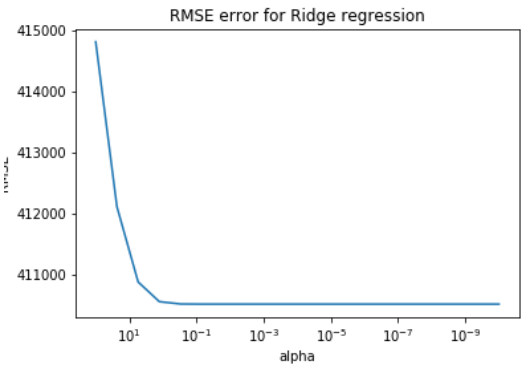
Ridge Regression-

Alpha:
For my model, it represents the regularization strength (a positive float). Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. I performed Parameter Tuning over Alpha variable on the train set using Grid search method for Ridge. The values of *alpha* selected for tuning are-

```
{'ridge__alpha': array([  1.00000000e-10,   4.28133240e-10,   1.83298071e-09,
         7.84759970e-09,   3.35981829e-08,   1.43844989e-07,
         6.15848211e-07,   2.63665090e-06,   1.12883789e-05,
         4.83293024e-05,   2.06913808e-04,   8.85866790e-04,
         3.79269019e-03,   1.62377674e-02,   6.95192796e-02,
         2.97635144e-01,   1.27427499e+00,   5.45559478e+00,
         2.33572147e+01,   1.00000000e+02])}
```

Programming Language	Python
Package	Sklearn.linear_models – Ridge, RidgeCV
Cross Validation	5
Best *alpha*	{'ridge__alpha': 0.2976351441631313}
Scaling	StandardScaler()
Loss Function	Man Squared Error
Parameter Tuning	GridSearch CV
RMSE on Test set (Best Param)	410511.4300866021
Test/Train Split	0.8/0.2
Time to Optimize	3.1784531810532854 s

I further plotted the RMSE scores for a set of *alpha* values mentioned earlier. The plot suggests that the GridSearch selected the **elbow** point in the graph. The tuned alpha was selected using the train set. The figure is created on test set. The *alpha* = 0.297 is the elbow point for the graph

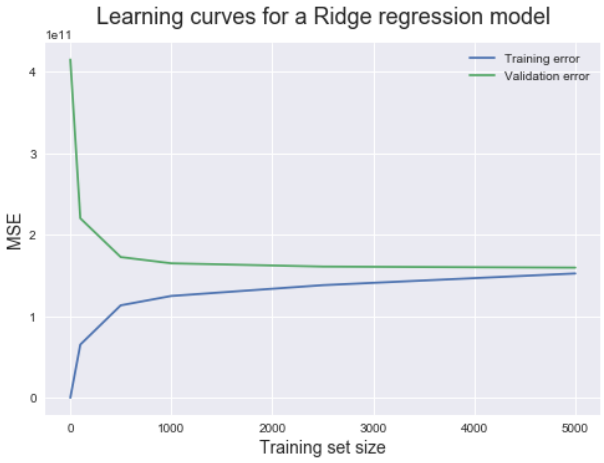


Learning Curves - We'll use the *learning_curve()* function from the scikit-learn library to generate a learning curve for a regression model. There's no need on our part to put aside a validation set because learning_curve() will take care of that. A cross-validation generator splits the whole dataset k times in training and Validation data. Subsets of the training set with varying sizes will be used to train the

estimator and a score for each training subset size and the Validation set will be computed. Afterwards, the scores will be averaged over all k runs for each training subset size.

Training Size Selected - [1, 100, 500, 1000, 2500, 5000] ** Same parameter is used to compare other models. All features are used. Cross Validation is 5. To plot the learning curves, we need only a single error score per training set size, not 5. For this reason, we take the mean value of each row.

Observations -

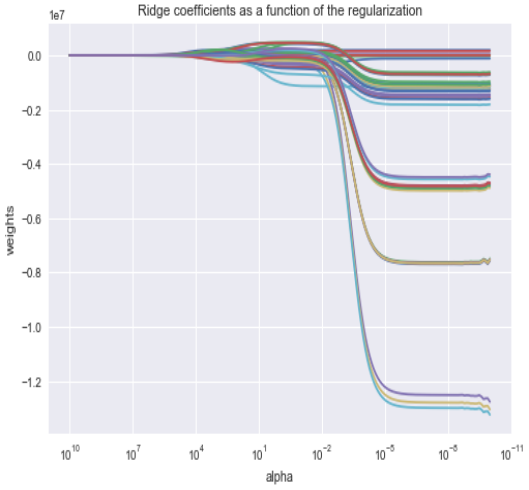


- When the training set size is 1, we can see that the MSE for the training set is 0. This is normal behavior, since the model has no problem fitting perfectly a single data point. So when tested upon the same data point, the prediction is perfect.
- But when tested on the validation set, the MSE rockets up. Such a high value is expected, since it's extremely unlikely that a model trained on a single data point can generalize accurately to new instances it hasn't seen in training. When the training set size increases, the validation MSE stays roughly the same. This tells us something extremely important: adding more training data points won't lead to significantly better models but adding features might

- As the error is high, the model is biased. This model has a high bias problem as the training error is high, it means that the training data is not fitted well enough by the estimated model.

- The variance seems low as the training error - validation error is low as training size increases. The model doesn't fit training data well.

Ridge coefficients as a function of the regularization:



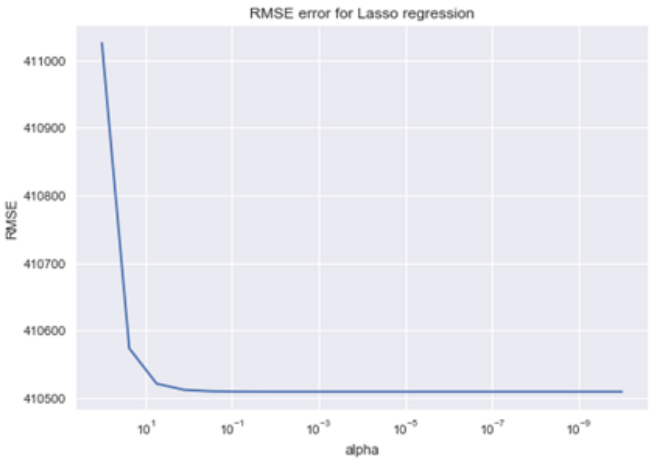
- This plot shows the effect of collinearity in the coefficients of an estimator. Each color represents a different feature of the coefficient vector, and this is displayed as a function of the regularization parameter. Ridge is useful to set a certain regularization (alpha) to reduce this variation (noise).

- When alpha is very large, the regularization effect dominates the squared loss function and the coefficients tend to zero. At the end of the path, as alpha tends toward zero and the solution tends towards the ordinary least squares, coefficients exhibit big oscillations.

Lasso Regression-

Alpha:

For my model, it represents the regularization strength (a positive float). Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. I performed Parameter Tuning over Alpha variable on the train set using Grid search method for Ridge. The values of *alpha* selected is Considered following values of -




```
array([  1.00000000e-10,   4.28133240e-10,   1.83298071e-09,
         7.84759970e-09,   3.35981829e-08,   1.43844989e-07,
         6.15848211e-07,   2.63665090e-06,   1.12883789e-05,
         4.83293024e-05,   2.06913808e-04,   8.85866790e-04,
         3.79269019e-03,   1.62377674e-02,   6.95192796e-02,
         2.97635144e-01,   1.27427499e+00,   5.45559478e+00,
         2.33572147e+01,   1.00000000e+02])
```

Programming Language	Python
Package	Sklearn.linear_models – Lasso, LassoCV
Cross Validation	10
Best *alpha*	23.357214690901213
Loss Function	Mean Squared Error
Parameter Tuning	LassoCV
RMSE on Test set (Best Param)	410507.2777
Test/Train Split	0.8/0.2

I further plotted the RMSE scores for a set of alpha values mentioned earlier. The plot suggests that the LassoCV selected the elbow point in the graph. The tuned alpha was selected using the train set. The figure is created on test set. The alpha = 3.357 is the elbow point for the graph

Learning Curves - Subsets of the training set with varying sizes will be used to train the estimator and a score for each training subset size and the Validation set will be computed. Afterwards, the scores will be averag Training Size Selected - [1, 100, 500, 1000, 2500, 5000]. All features are used. Cross Validation is 5. To plot the learning curves, we need only a single error score per training set size, not 5. For this reason, we take the mean value of each row over all k runs for each training subset size.

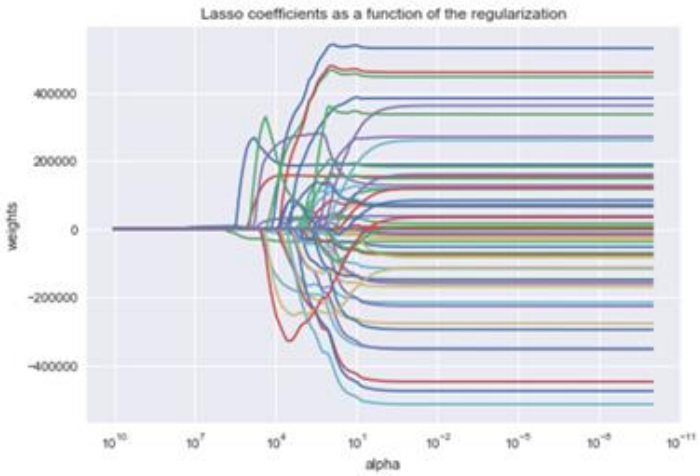
Observations –

- The trend is like that of Ridge Regression as it initially tends to behave similarly. As the training size is increased, we can see that Lasso zeroes some of the features leading to sharper fall
- As the error is high, the model is biased. This model has a high bias problem as the training error is high, it means that the training data is not fitted well enough by the estimated model.

- The variance seems low as the training error - validation error is low as training size increases. The model doesn't fit training data well.

Lasso coefficients as a function of the regularization:

- When alpha is very large, the regularization effect dominates the squared loss function and the coefficients tend to zero.
- At the end of the path, as alpha tends toward zero and the solution tends towards the ordinary least squares, coefficients exhibit big oscillations. In Lasso's case it performs feature selection



Feature Selection In Lasso: The following features have zero coefficients in the optimized model - *Type_t, Method_SA, Method_SP, CouncilArea_Manningham City Council, Regionname_South-Eastern Metropolitan, Regionname_Southern Metropolitan, Regionname_Western Metropolitan*

Non - parametric models.

Random Forest Regressor-

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. Some of the constant paramters throught are 0

'bootstrap'	True
Loss	MSE
'max_depth	None
'warm_start'	False
'min_samples_leaf	1
Package	RandomForestRegressor SkLearn

Parameters Tuned –

a.) 'max_depth': The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. The range of this is [10,132,255,377,500].

b.) 'n_estimators': The number of trees in the forest. The range is [10, 53, 96, 140, 183, 226, 270, 313, 356, 400]

We plot a graph of *MSE V/S Parameters*. This is helpful in comparing the trend of parameters over error and compare it with the optimal results found by grid search

RandomizedSearchCV:

I used RandomizedSearchCV to perform parameter tuning with cv = 3. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings.

Using the above I optimized my model to get the best parameters, I got the following results.

MSE Error	320600.882013
max_depth	None
N_estimator	10

Before Tuning

MSE Error	304694.798413
max_depth	500
n_estimators	205

After Tuning

Learning Curves -

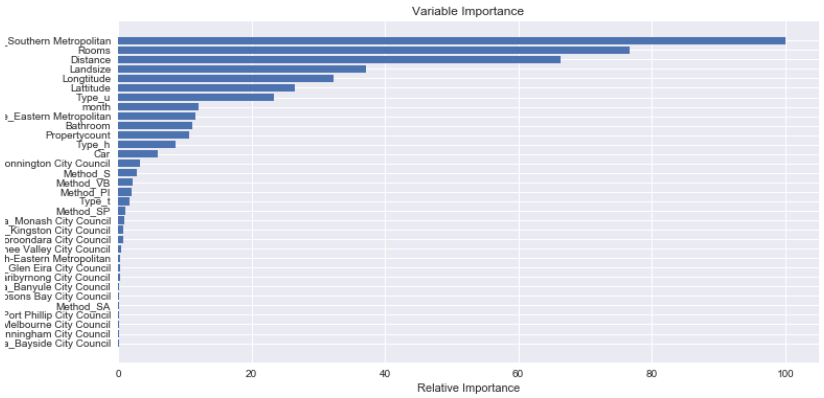
Observations-

1. The validation curve could converge toward the training curve if more training instances were added
2. This suggests that adding more training instances would help
3. As the training error upon converging is low, the model has a low bias / high variance problem.
4. As the gap between training error and validation error is large, it suggests high variance. Also, due to low training error, we can conclude that the model overfits the training data.

Feature Selection:

Observations-

1. The following are the most import variables are the most important features.
2. These will be useful in comparing models in Linear Models at the end.



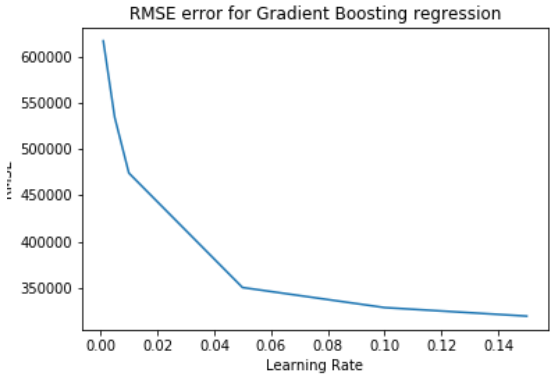
Gradient Boosting Regressor-

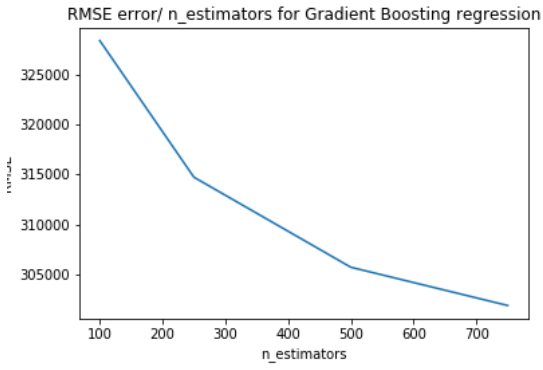
GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function. Parameters which are constant throughout the model.

Loss	MSE
'max_depth	None
'warm_start'	False
'min_samples_leaf	1
Package	GradientBoostingRegressor SkLearn

Parameter Tuning –

1. Learning Rate- learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators. The range tested was [0.15,0.1,0.05,0.01,0.005,0.001]
2. No. of estimators- The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting, so a large number usually results in better performance.





GridSearch for parameters-

I used the following parameters for Gradient Boosting changing only learning parameters and No. of estimators. *{max_depth=4, min_samples_split=2, min_samples_leaf=1, subsample=1, max_features='sqrt', random_state=10), param_grid = p_test3, scoring='explained_variance', n_jobs=4, iid=False, cv=5}*

I used RandomizedSearchCV to perform parameter tuning with cv = 3. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings.

RMSE	328474.20397842064
learning_rate	0.1
No Estimators	100

Before Tuning

RMSE	291257.119966
learning_rate	750
No Estimators	0.1

After Tuning

Learning Curves -

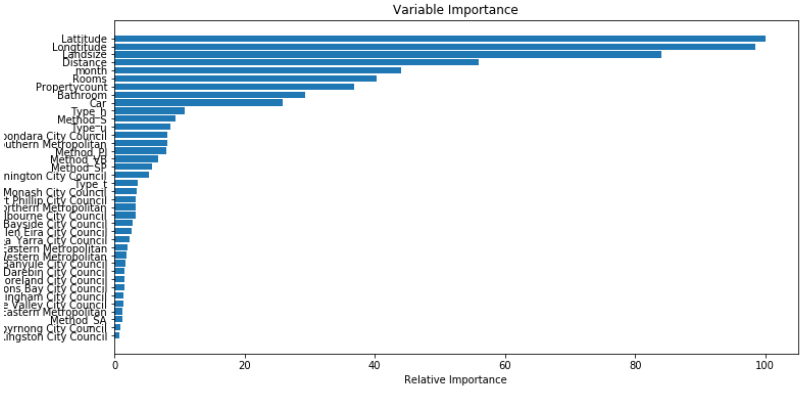
Observations-

1. The validation curve could converge toward the training curve if more training instances were added
2. As the training error upon converging is low, the model has a low bias / high variance problem.
3. As the gap between training error and validation error is large, it suggests high variance. Also, due to near zero training error, we can conclude that the model overfits the training data

Feature Selection:

Observations-

1. The following are the most import variables are the most important features.
2. These will be useful in comparing models in building modified model at the end.



COMPARASION OF MODELS-

Model	MSE (Tuned / Best Model)	Time to Optimize (s)
Ridge	410511.4300866021	3.1784531810532854
Lasso	410507.2777	98.35102263379402
Random Forest Regressor	304694.798413	
Gradient Boosting Machine	291257.119966	

Advantages/Disadvantages of Models –

1. Ridge V/s Lasso- Lasso doesn't significantly reduce the RMSE error significantly but Lasso is useful in feature selection which is used in the future models saving computation. Ridge doesn't do feature selection. Ridge is computationally less expensive than Lasso.
2. Random Forest V/s Gradient Boosting Machine – GBM is the best performing model after parameter tuning. There is not a huge effect in the accuracy. However Random Forest Regressor is more computationally effect.
3. Ridge/Lasso V/s Linear Regression V/s Subset Selection – I chose Ridge and Lasso regression over other Linear regression methods as they have a regularization effect which worked well with the high dimensional dataset I had. Linear model was too simplified to correct model this data and subset selection was computationally expensive due to the large number of features.
4. Random Forest/ Gradient Boosting Machine V/s KNN – KNN would have been computationally expensive and I was not convinced with my cluster results in the previous results. Due to poor clusters, I anticipated KNN to perform poorly. On the other hand, Boosting and Random Forest base on Decision Trees which can learn complex models.

Q4. Non-parametric models like Random Forest Regressor upon training generate the importance of features used in training. Another method could be ensembling techniques of different types of regression models.

Intuition –

Feature Selection - Random Forest is a tree based technique. A single random forest will unlike Penalized regression with lasso regularization completely ignore features. Decision tree splits by features are chosen by local criteria in any of the thousands or millions of nodes and cannot later be undone. As Random Forest does a more exhaustive decision making, it's best to input the entire dataset (without any prior feature selection) and henceforth upon using Random Foreset achieve substantial increase in prediction performance (estimated by a repeated outer cross-validation) using its variable selection.

Reducing Error –

Ensemble methods are used to use predictions of several weak estimators built to improve generalization/ robustness over a single estimator. These are of two types-

- Averaging methods - The combined estimator is usually better than any of the single base estimator reducing overall model variance. Examples: Bagging methods
- Boosting methods - base estimators are built sequentially and one tries to reduce the bias of the combined estimator. It aims in improving a accuracy of week model.

In my analysis, I have performed Gradient Boosting Machine previously, and I think improving this model can improve accuracy. It already has the best accuracy by a good margin as compared to the Penalized regression models

Computational Challenges: In my approach, there weren't computational issues except during tuning of parameters for each model. As the features get added or dataset increases, this tuning would take longer. This is the reason I chose not to do stacking, where we give weightage to every model trained and do a weighted average of all the predictions. Training all the 4 models to get a single model is more computationally a disadvantage over the minor improvement of RMSE .

Methodology-

For feature selection, Random Forest provides: mean decrease impurity and mean decrease accuracy. Random forest consists of several decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. The measure based on which the (node) optimal condition is chosen is called impurity (variance). Using the training set, it computes how much each feature decreases the weighted impurity in a tree. Combining these decision trees (Forest) the impurity decreases from each feature which is averaged and ranked to give the most important features. As a result, I use top *n* features from Random Forest and input those in to the boosting model. The *n* feature becomes a hyperparamter and can be tuned. We haven't tuned it in this section, but this can be done to improve future results.

Results of Improved Model –

I use the clusters created earlier and assign them as feature. As they are categorical variable, they get converted to One hot vectors. Rest of the features remain the same.

Model	MSE
GBM (pre-feature selection, tuned)	291257.119966
GBM (feature selection, not tuned	334113.86904414993
GBM (feature selection, tuned)	292033.3933783364

I tune the new GBM on the same set of parameters range as earlier (n_estimators, learning rate) and max depth of a single tree which is tuned over the range *{'max_depth':[2,3,4,5,6,7] }*.

OBSERVATIONS –

- We observe that with the selected feature selection method doesn't work well. Perhaps, tuning the number of feature to include from Random Forest would improve the feature.
- Also, we haven't scaled any features, doing that in the future can improve accuracy
- The overall computation time of the modified model has decreased with considerable improvement in error.