JULY 17, 2017

KING COUNTY HOUSING ANALYSIS

MACHINE LEARNING SEIS 763

1. INTRODUCTION

This analysis focuses on using machine learning to predict housing prices for King County, USA as well as identify key factors that dictate housing prices. We utilize a variety of machine learning techniques, generate various models and share some of the challenges encountered within this analysis. As housing prices have reached pre-recession ranges and many home-owners are still not sure about moving, the housing environment as a whole is in an interesting state. First-time home buyers are having a harder time getting in the game and the low interest rates are not tempting current home owners to sell. This analysis will provide an in-depth look at various models used and recommendations to better understand the future of housing prices.

2. DATA OVERVIEW

A. DATA SOURCE

The dataset our team will use for our project is *House Sales in King County, USA.* This dataset incorporates the sale prices and attributes of homes in King County, Washington between May 2014 and May 2015. The dataset was released to the public 10 months ago under CC0:Public Domain License and is 777.92 KB in size. King County, Washington is currently categorized as the most populous county in Washington, as it includes Seattle, and the 13th most populous in the United States.

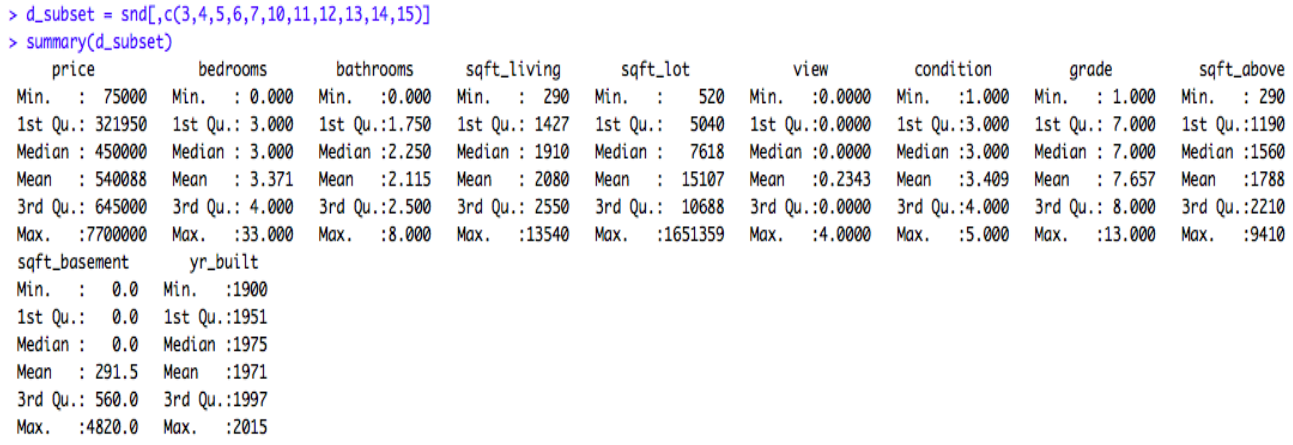
Link to Data: **https://www.kaggle.com/harlfoxem/housesalesprediction**

B. DATA ATTRIBUTES

- ID: Unique identifier for the house that was sold.
- Date: Date the house was sold on.
- Price: The monetary price the home was sold for.
- Bedrooms: Total number of bedrooms within the house.
- Bathrooms: Total number of bathrooms within the house.
  - 1.0 = Full, include bath
  - 0.75 = Partial, includes shower
  - 0.50 = Half, does not include bath or shower
- Sqft_living: Total amount of livable square footage within the house.
- Sqft_lot: Total square footage of the land the house sits upon.
- Floors: Total number of floors/stories the house includes.
- Waterfront: Is this house/land a waterfront property.
- View: Does the house/land have a noteworthy view of the area (water, city, valley, etc.)
- Condition: Scale of 1 to 5 on the overall condition of the house that sold
  - 1 = need a complete remodel
  - 5 = move-in-ready
- Grade: The amount of slope that is near a house for water run-off.
  - 1-13 range
- Sqft_above: Total amount of livable square footage above ground level and the foundation on record

- Sqft_basement: If applicable, the total livable square footage below ground level
- Yr_Built: Year the house was built.
- Yr_Renovated: If applicable, the most recent year the house has been renovated.
- Zipcode: The zip code for the address of the house.
- Lat: Latitude of the geo-location for the house.
- Long: Longitude of the geo-location for the house.
- Sqft_Living15: Actual total livable square footage of house sold in 2015.
- Sqft_Lot15: Actual total square footage of land sold in 2015.

C. GENERAL STATISTICS



```
> d_subset = snd[,c(3,4,5,6,7,10,11,12,13,14,15)]
> summary(d_subset)
     price            bedrooms        bathrooms      sqft_living       sqft_lot           view          condition         grade         sqft_above
 Min.   :  75000   Min.   : 0.000   Min.   :0.000   Min.   :  290   Min.   :    520   Min.   :0.0000   Min.   :1.000   Min.   : 1.000   Min.   : 290
 1st Qu.: 321950   1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1427   1st Qu.:   5040   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.: 7.000   1st Qu.:1190
 Median : 450000   Median : 3.000   Median :2.250   Median : 1910   Median :   7618   Median :0.0000   Median :3.000   Median : 7.000   Median :1560
 Mean   : 540088   Mean   : 3.371   Mean   :2.115   Mean   : 2080   Mean   :  15107   Mean   :0.2343   Mean   :3.409   Mean   : 7.657   Mean   :1788
 3rd Qu.: 645000   3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550   3rd Qu.:  10688   3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.: 8.000   3rd Qu.:2210
 Max.   :7700000   Max.   :33.000   Max.   :8.000   Max.   :13540   Max.   :1651359   Max.   :4.0000   Max.   :5.000   Max.   :13.000   Max.   :9410
 sqft_basement       yr_built
 Min.   :   0.0   Min.   :1900
 1st Qu.:   0.0   1st Qu.:1951
 Median :   0.0   Median :1975
 Mean   : 291.5   Mean   :1971
 3rd Qu.: 560.0   3rd Qu.:1997
 Max.   :4820.0   Max.   :2015
```

D. DATA PREPWORK & BAD DATA

After initial review of the data we determined that the id and date columns would to be removed as they did not have any impact on the pricing analysis. After further review, we also decided to exclude the sqft_living15 and Sqft_Lot15 as our data source did not provide a clear enough definition of the data columns and we wanted to be sure to avoid data leakage within the analysis.

Upon reviewing our general statistics of each column within the raw data file we found out that few records had unrealistic figures, classifying them as bad data, and they were removed from our file. There were 13 total initial bad data rows identified and a few examples are listed below:

I. Row 15872 had number of bedrooms as 33 and sq.ft living 1620.
II. Row 12655 had the following values:

Price	Bedroom	Bathroom	SqFtLiving	SqFtLot	Floors
320000	0	2.5	1490	7111	2

Overall our dataset was completely populated with no missing data, known of, but we suspect there were more bad records within our data file and will be sure to use Cooks Distance and Leverage to identify those records who may have great effect on our model.

3. CONTENT OF PROJECT

A. GOALS OF ANALYSIS

Since the housing bubble in the 2000's many homeowners have been caution on moving as the overall value of their house most likely decreased but in the last 5 years there has been a steady increase in home values and selling price points. In fact there has been such an increase new records are now being set in "hot" areas around the country, including Kings County, WA. As stated in The Seattle Times article *No Escape for Priced-Out Seattleites, June 2017: Home Prices Set Record For an Hour's Drive in Every Direction:*

"Seattle, which has been setting records every month, is on the verge of a once-unthinkable milestone: $1 million for the typical house across the entire area around Capitol Hill and northeast of downtown."

With home prices so unpredictable, the goal of this analysis is to provide individuals looking to purchase a home in Kings County an idea of how much house they could get with their budget and exactly which features within a house may have to be sacrificed to stay within budget.

B. DATA VARIABLES

Ranges/Categories:

Price - used for SVM & NN

• 1: price < 321,950	(Total Records- 5404)
• 2: price > =321,951 & price < 450000	(Total Records- 5460)
• 3: price >= 450001 & price < 645000	(Total Records- 5376)
• 4: price >= 645000	(Total Records – 5373)

Dummy Variables:

- Bedrooms**:** We assigned dummy variables as it was a categorical feature of each house.
- Bathroom: We assigned dummy variables as it was a categorical feature of each house.
- Floors: Floors was assigned dummy variables as it was a categorical feature of each house.
- View: View was on a scale from 1 to 5 so we created dummy variables.
- Condition: Condition was on a scale from 1 to 5 so we created dummy variables.
- Grade: Grade is on a large scale from 1 to 11 so we created dummy variables.
- Yr_built: It is clearly categorical so we created dummy variables for each year.
- Yr_ren: It is clearly categorical so we created dummy variables for each year.
- Zipcodes: We created dummy variables for each individual zip code. We thought about scale-ling up to county or congressional district but the conversion was not 100% accurate due to zip code overlap between counties and congressional districts.
- Waterfront: This is Yes/no so we created binary dummy variables.

Numeric Variables:

- Sqft_living: this was kept a numeric variable throughout the analysis.
- Sqft_basement: this was kept a numeric variable throughout the analysis.
- Lat: this was kept a numeric variable throughout the analysis.
- Long: this was kept a numeric variable throughout the analysis.

C. DESCRIPTION OF PROJECT EXECUTION

We wanted to try multiple things to see what worked best for the problem at hand. We attempted multiple models (not always successful) and reviewed the outcomes with each other.

To ensure we get the lowest error and highest overall accuracy we approached many models individually using a variety of techniques, below we provide in-depth detail of each approach and the results produced:

1. Model 1: Linear Regression Full Data-No Ranges:

Starting with the raw data file we wanted to learn more about what was actually in the data file and work to see if we could find a base file everyone could work with. Meaning a base number of predictors that had a low error rate and higher adjusted R square value as well as any necessary outliers or bad data values removed. We felt that a base file would allow us to be able to compare our classification and regression models more accurately.

We then created a linear regression model using the following steps:

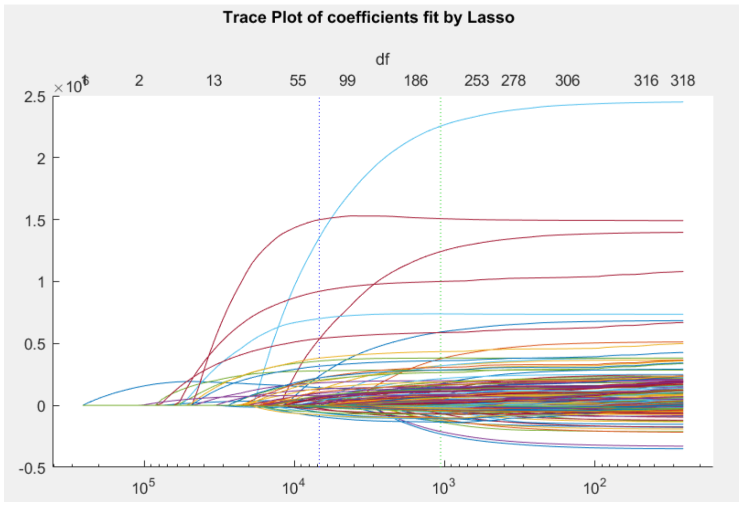
i. *Standardized (zScore) and categorized (dummy variables) to the data set:* Standardization was only applied to the numeric variables while the creation of dummy variables was applied to all non-numerical variables. Detailed list of variables is located in 3B: Data Variables.

ii. Create Linear Model 1 (MD1): Using the new dummy variables and standardized variables we created our first linear model to provide a base for error (MSE), adjusted R squared and take a look at the p-values produced by our predictors. The results were as follows:

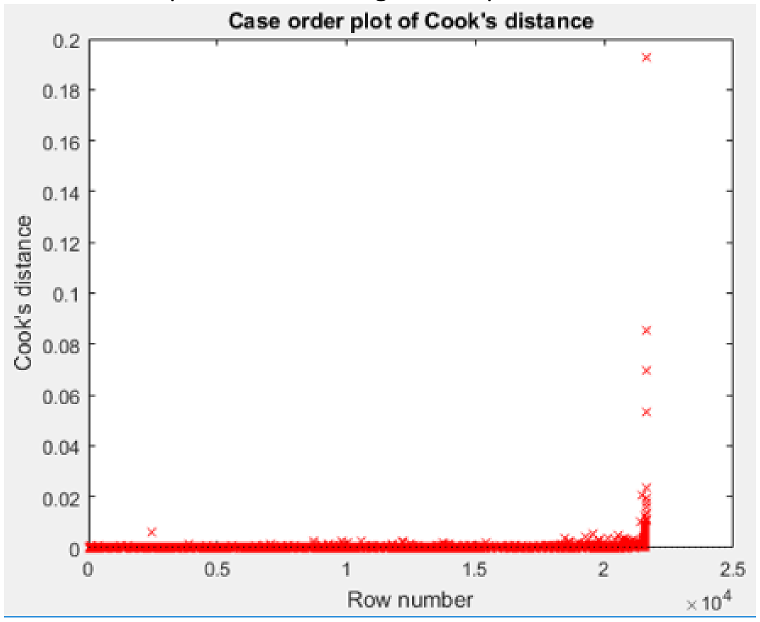
- Total Predictors: 321
- Adjusted R-squared: 0.845
- MSE: 2.09e+10 (exact 20942940057.3651)
- R-squared: 0.847
- Root Mean Squared Error: 1.45e+05

We were surprised by how high our MSE was but knew that we could improve on this base number. We also felt our total number of predictors was higher compared to the overall data set size. The adjusted R-squared was at a positive number but with the MSE so high we knew it was not a great indicator of how great our linear model was.

iii. To lower the total number of predictors we generated a lasso plot using 10-flod cross validation. As this first lasso plot was generated on a personal laptop using matlab and took over 2 hours to identify which lambda value would produce the best MSE and remove unnecessary predictors, therefore we utilize the B table to determine what lambda value to choose and which predictors were categorized as most important.



iv. *Identifying Mistake, take step back to remove outliers*: Once we were reviewing the B-table we noticed that a handful of predictors had 0's in the middle of the table and then went negative before coming back to 0. We determined that there was an issue with our model and we needed to take a close look at outliers to see if we could improve our results. The initial run of cooks distance produced the image below, which had indicated a high range of outliers due to the fact we could barely see the 3-times distance line. Similar results happened when we ran the leverage plot. With much in-depth discussion across the team on whether to remove 3 times the outliers or 5 times, the decision was to be more cautious and only go with 5 times the cooks distance and leverage as our dataset was already on the small side and we wanted to keep as much training data as possible.



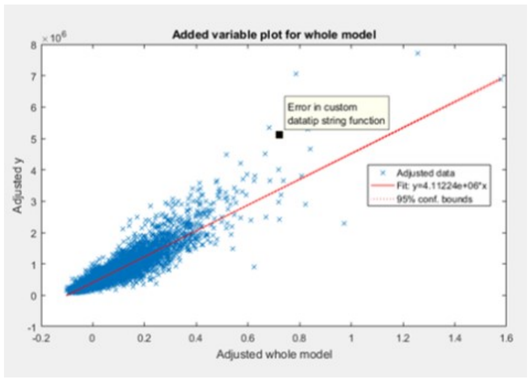
v. *Remove Outliers and Re-create variables/standardize:* We removed the 259 identified outliers using the following code:

```
potential_outlier=find((md1.Diagnostics.CooksDistance+md1.Diagnostics.Lever
age)>5*mean(md1.Diagnostics.CooksDistance+md1.Diagnostics.Leverage));
removerows (houseData, 'ind', potential_outlier(:));
```

Then it was critical to recreate our dummy variables as data rows have been adjusted as well as standardize our data. Without conducting this step our results would not be accurate going forward.

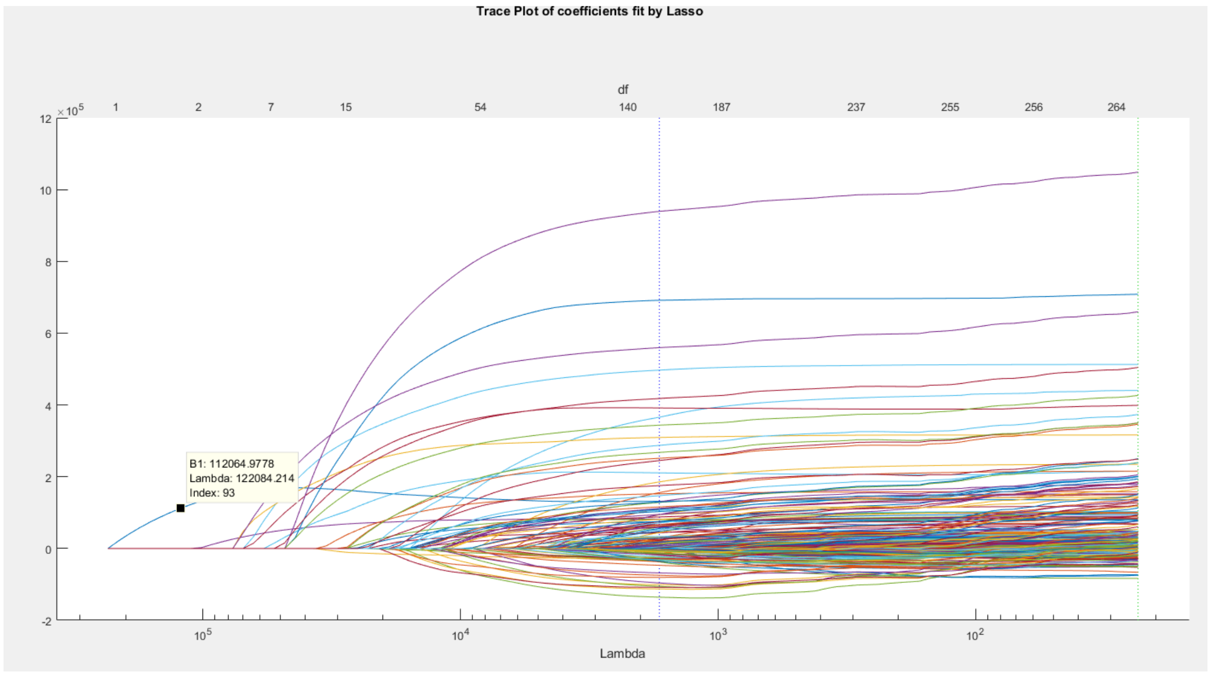
vi. *Created new linear model (lm2):* Satisfied with the new set of variables created and outliers removed from our original dataset we ran a second linear model using the updated information and the model produced the following results:

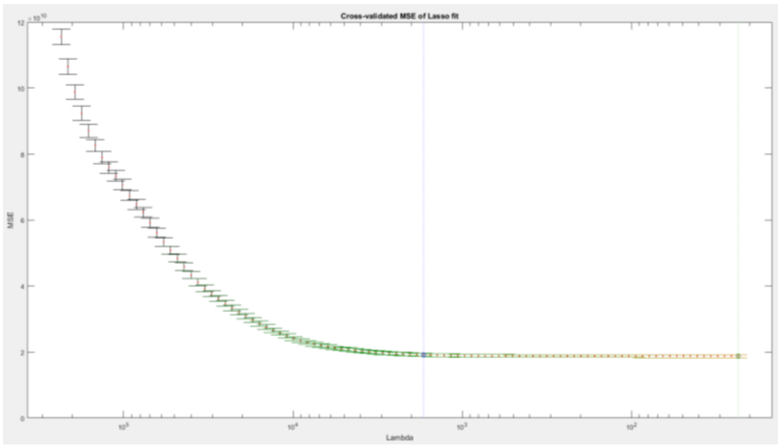
- Total Predictors: 266
  (a decrease of 55 predictors from original 321)
- Adjusted R-squared: 0.843
  (slight decrease from .845)
- MSE: 1.81e+10 (exact 18180765433.3296)
  (decrease from 2.09e10 in lm1)
- R-squared: 0.845
  (a slight decrease from 0.847)



- Root Mean Squared Error: 1.35e+05 (as light decrease from 1.45e+05)

vii. *Generate new Lasso Plot with ml2:* After another iteration of the lasso plot, which this time took only about 2 hours due to the lower number of predictors, produced the following result. There was not a significant change as the first 3 predictors stayed consistent between the two lasso plots produced. What was a change was the range between 0 MSE (green vertical line) and 1SD of MSE (blue vertical line).





viii. *Choose a lambda value and adjust our model accordingly:* At this point we used the following code to conduct a trial and error period to identify the lambda that produced the lowest MSE while removing unnecessary predictors.

```
B8 = b(:,50); %index of the b table
nonzeros = sum(B8 ~= 0)          % 140 no zeros found
predictors8 = find(B8);          % indices of nonzero predictors
md10 = fitlm(XfinalNew,Ynew,'PredictorVars',predictors8)
```

In total we conducted over 12 different trials to identify which lambda would produce the best results and landed at a lambda of 50, which was slightly over the 1SD MSE line but we discussed as a team that the slight sacrifice in MSE was worth losing the high number of predictors at 50.

The 50 lambda value left us with 140 total predictors (141 with Y value), which was a 126 decrease from the ml2 results. The lm10 (lambda 50) model produced the following results:

```
ml10: Linear regression model:    y ~ [Linear formula with 141 terms in 140 predictors]

Estimated Coefficients:
                 Estimate        SE            tStat          pValue
               ___________    _______        _______        ________

 (Intercept)    4.696e+05     4293.4          109.38         0
  x1           1.3142e+05     1978.6          66.42          0
  x2            -14878        1239.3          -12.005        4.264e-33
  x3           78651          2448.3           32.125        3.8639e-221
  x5            10430         1011.9          10.307         7.5002e-25
  x7            11058         2034.2           5.4358        5.514e-08
  x9            -14157        3924.5          -3.6073        0.00031006
  x10           -50840        8812.7          -5.7689        8.0887e-09
  x18           -4564.8       2522.4          -1.8098        0.070348
  x20           15740         5335.4          2.9501         0.0031797
  x21           52335         6116.9          8.5558         1.2491e-17
  x23           84231         11446           7.3591         1.9187e-13
  x24           82118         12357           6.6457         3.0916e-11
  x25           1.6146e+05    16004           10.089         7.019e-24
  x26           85467         14648           5.8346         5.4713e-09
  x27           4.2064e+05    29568           14.226         1.0272e-45
  x28           1.3774e+05    32429           4.2474         2.1717e-05
  x30           -2168.6       3575            -0.6066        0.54412
  x32           58699         11220           5.2315         1.6977e-07
  x33           -60839        6226.8          -9.7706        1.6795e-22
  x34           99119         7772.1          12.753         4.0952e-37
  x35           79376         4730.3          16.78          8.6647e-63
  x36           1.5873e+05    6476.1          24.51          7.5881e-131
  x37           3.1551e+05    10146           31.096         1.2513e-207
  x38           -33859        10751           -3.1495        0.0016378
  x39           -28465        2347.6          -12.125        1.0054e-33
  x41           47690         3835.4          12.434         2.2643e-35
  x42           -50344        9497.3          -5.3008        1.1642e-07
  x43           -45995        4277.5          -10.753        6.7598e-27
                Estimate        SE            tStat          pValue
               ___________    _______        _______        ________

  x44           -32365        2620.8          -12.349        6.4903e-35
  x46           81353         3523.3          23.09          1.5973e-116
  x47           2.0879e+05   5264.6           39.659         0
  x48           3.9253e+05   8570.5           45.8           0
  x49           7.0609e+05   17356            40.683         0
```

x54	50272	16097	3.1232	0.0017915
x56	31617	17522	1.8044	0.071181
x58	35170	14551	2.417	0.015656
x59	20545	12288	1.672	0.094541
x60	40513	16796	2.4121	0.015871
x62	45839	19183	2.3896	0.016875
x65	39978	15646	2.5551	0.010623
x70	34740	15850	2.1918	0.028406
x71	33534	14483	2.3154	0.020603
x72	47890	15347	3.1205	0.0018077
x75	33365	10427	3.1997	0.0013776
x77	40166	12515	3.2095	0.0013316
x82	72173	26320	2.7421	0.0061093
x85	70956	22789	3.1135	0.0018511
x86	45593	17024	2.6783	0.0074064
x87	52769	19699	2.6788	0.0073948
x88	39624	13658	2.9011	0.0037225
x89	27213	11130	2.445	0.014492
x90	45194	11048	4.0908	4.3156e-05
x98	32493	9979	3.2562	0.0011311
x104	-22266	8434.4	-2.6399	0.0082991
x115	-18381	8770.1	-2.0958	0.03611
x127	-10669	7084.9	-1.5058	0.13213
x128	-16562	7478	-2.2148	0.026788
x139	-44029	7817	-5.6325	1.7987e-08
x147	-23617	8979.4	-2.6302	0.0085407
x154	-20252	6629.9	-3.0546	0.0022564
x155	-20926	6648.4	-3.1475	0.001649
x162	47885	9761.9	4.9053	9.397e-07
x163	43220	6082.9	7.1051	1.2405e-12
x169	75387	33070	2.2796	0.022641
x175	1.0106e+05	31363	3.2222	0.001274
x176	1.5517e+05	32187	4.821	1.438e-06
x177	1.0018e+05	34037	2.9432	0.0032515
x181	1.1211e+05	34182	3.2799	0.0010402
x184	2.3454e+05	29718	7.892	3.1182e-15
x185	84774	23049	3.678	0.00023567
x186	1.028e+05	26709	3.8488	0.00011906
x187	84664	23062	3.6711	0.00024207
x188	71247	27827	2.5604	0.010463
x189	83689	23077	3.6266	0.00028788
x190	2.1335e+05	32109	6.6444	3.1194e-11
x191	1.4769e+05	29679	4.9763	6.5331e-07

	Estimate	SE	tStat	pValue
x192	1.4901e+05	33988	4.3841	1.1702e-05
x194	1.0761e+05	22450	4.7933	1.6517e-06
x195	88789	14361	6.1827	6.4148e-10
x199	5.7635e+05	8718.1	66.11	0
x200	1.5371e+05	11194	3.731	1.0123e-42

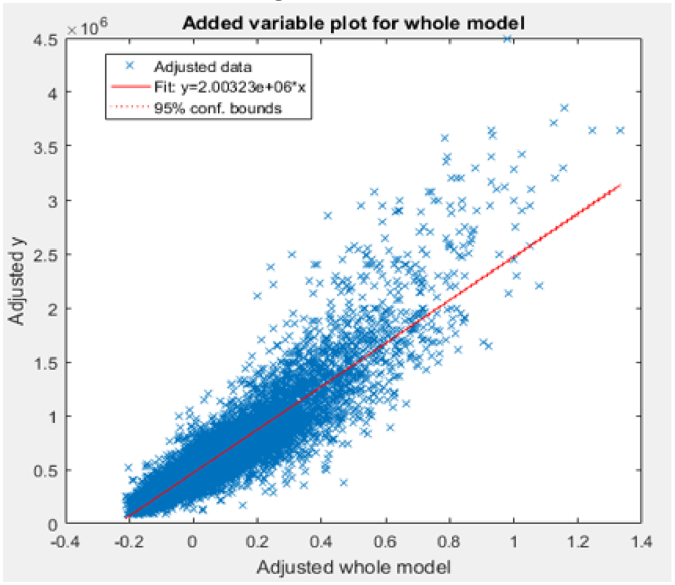
```
x201     1.219e+05     7083.5     17.208     6.4316e-66
x202     89932         12118      7.4215     1.2019e-13
x203     96599         9021       10.708     1.0899e-26
x204     41218         14353      2.8717     0.0040868
x205     -1.0846e+05   11392      -9.5205    1.9021e-21
x206     -1.178e+05    13174      -8.942     4.1289e-19
x207     -1.4788e+05   11361      -13.017    1.3772e-38
x208     37815         10408      3.6332     0.00028063
x209     -24845        7143.1     -3.4782    0.00050588
x211     44469         7308.5     6.0846     1.1877e-09
x212     -1.1769e+05   10018      -11.748    9.1114e-32
x213     74333         8264.9     8.9938     2.5844e-19
x214     -27783        9001       -3.0867    0.0020267
x215     -35812        8665.2     -4.1328    3.5977e-05
x216     -38242        12528      -3.0526    0.0022712
x217     1.4568e+05    8108.3     17.967     1.1986e-71
x218     -31000        7968       -3.8906    0.0001003
x220     9.7059e+05    21453      45.242     0
x221     3.5861e+05    9017.3     39.77      0
x222     -26770        6549.9     -4.0871    4.3833e-05
x224     35064         7360.6     4.7637     1.9133e-06
x226     -42932        8730.5     -4.9175    8.8309e-07
x227     -25536        7355.2     -3.4718    0.00051794
x228     -42565        6895.8     -6.1726    6.8404e-10
x229     -15295        6839.1     -2.2364    0.025333
x230     -25731        8339.7     -3.0854    0.0020356
x231     -72811        13303      -5.4731    4.4714e-08
x232     -82657        10047      -8.2272    2.0261e-16
x234     26427         8057.1     3.28       0.0010397
x235     -1.197e+05    11243      -10.646    2.1126e-26
x236     -25260        8183.9     -3.0866    0.0020273
x237     2.7331e+05    14378      19.009     6.6538e-80
x238     1.2676e+05    7750       16.356     9.0874e-60
x239     2.6734e+05    10371      25.777     2.6438e-144
x240     -14462        8185.7     -1.7668    0.077279
x241     1.3488e+05    9837       13.711     1.3236e-42
x242     -29039        10539      -2.7554    0.0058674
x243     3.1298e+05    13876      22.556     2.4035e-111
x244     4.3319e+05    9551.5     45.353     0
x245     1.1873e+05    7677.7     15.464     1.1892e-53
x246     1.3053e+05    8482.8     15.388     3.8467e-53
x247     97169         7820.1     12.426     2.5185e-35
x248     22387         6979.6     3.2076     0.0013407
           Estimate      SE         tStat       pValue
         ___________   ______     ______     ________

x249     2.8614e+05    11184      25.585     3.2243e-142
x250     1.6397e+05    9056.3     18.106     1.0162e-72
x251     -31768        8751.2     -3.6301    0.00028393
x252     43988         8061.7     5.4564     4.9123e-08
x253     -82726        8479.8     -9.7557    1.9442e-22
```

x254	1.1055e+05	9139.4	12.096	1.4291e-33
x255	1.1168e+05	8338.3	13.394	9.624e-41
x256	-19590	8721.1	-2.2463	0.024698
x258	-1.1085e+05	8842.3	-12.537	6.2997e-36
x260	-57598	8920.9	-6.4565	1.0946e-10
x261	-19033	10410	-1.8284	0.067501
x262	-77477	8955.2	-8.6516	5.4318e-18
x263	-47111	12043	-3.9119	9.184e-05
x264	-46483	8646.1	-5.3762	7.687e-08
x265	1.8912e+05	9012	20.985	8.6904e-97
x266	5.1255e+05	14081	36.4	1.8867e-281

Predictors: 140

- Total Predictors: 140 (decreased 126 predictors)
- Adjusted R-squared: 0.841 (slight decrease from lm2 0.843)
- MSE: 1.83e+10 (exact: 18358559276.96520) (slight increase lm2 1.81e+10)
- R-squared: 0.843 (slight decrease from lm2 0.845)
- Root Mean Squared Error: 1.35e+05 (no change from lm2)
- Number of observations: 21348,
- Error degrees of freedom: 2120
- F-statistic vs. constant model: 810, p-value = 0

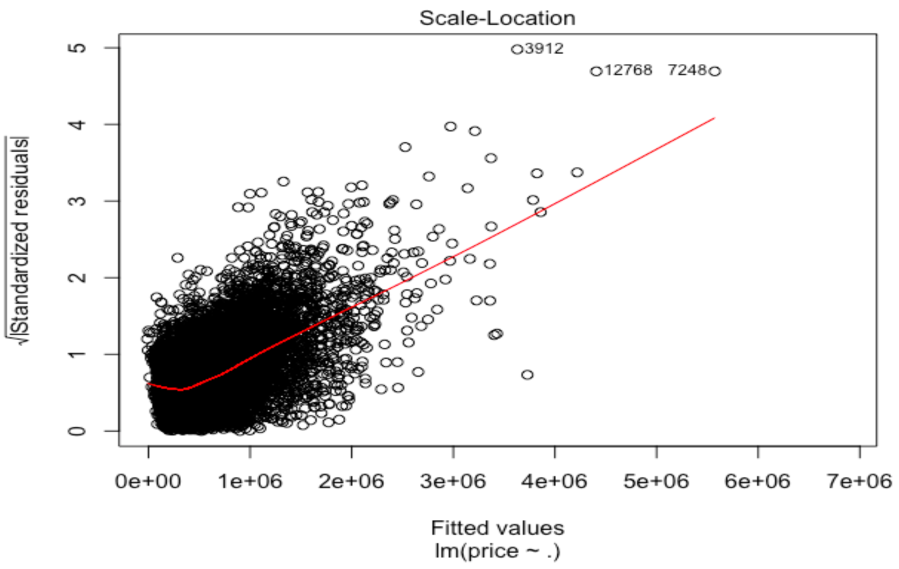
With the removal of those 126 predictors we have now regularized our model to an optimal base of predictors to use moving forward with our SVM model. The MSE is only slightly increased compared to the number of predictors we are now working with. We feel positive this base will produce solid results moving forward.

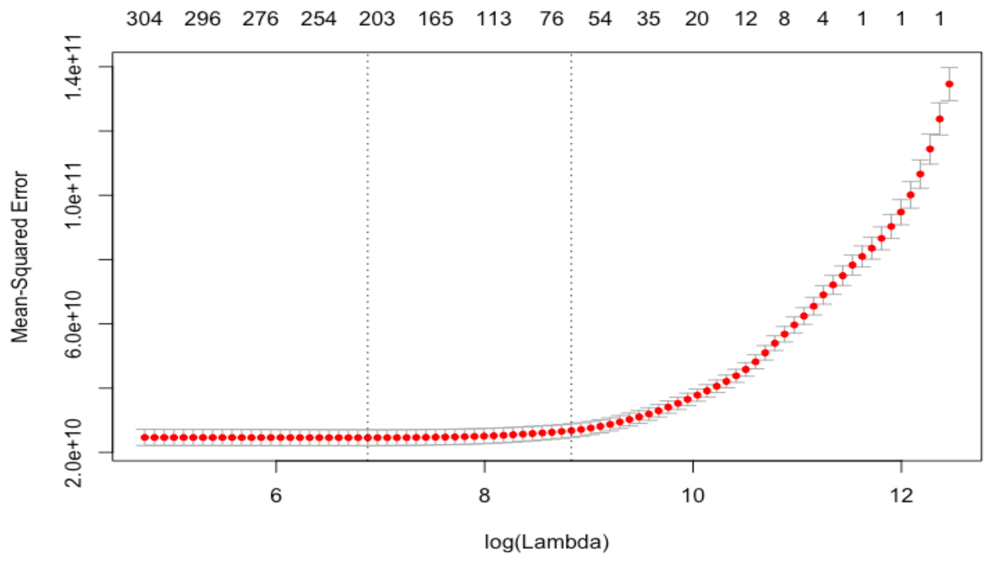


2. <u>Model 2: Liner Regression using R</u>

As a member of our group was more efficient using R we wanted to test out the second program to conclude we produce the best results across multiple platforms. R also proved to be significantly more efficient in building components of the model, such as the lasso plot. Therefore we also used R as a double check tool when time was running out during group meetings. Just as the linear model produced in Matlab did, this R model converted the dataset variables to dummy variables and standardized the rest of numeric variables. We used the following steps to build the model in R:

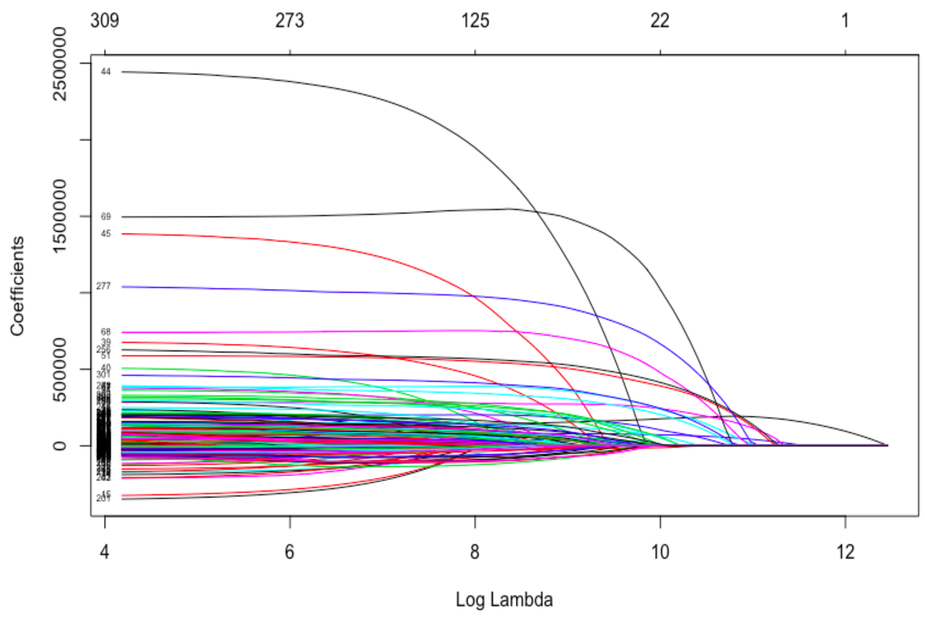
I. Generate a base linear model (ml1): This produced the following results, using price as the Y indicator, with no ranges applied:
   - Number of predictors: 321
   - Multiple R-squared: 0.8471
   - Adjusted R-squared: 0.8448 (Matlab result: 0.845)





We noticed that this model produced less visible outliers versus the Matlab program.

II. *Create new lasso plot with base model:*



The results were similar to the Matlab program but obviously the visuals are flipped since R scales lambda left to right versus right to left in Matlab.

III. *Simplify the model by removing some predictors:* The below data shows each lambda value with the number of predicates as well as the R square:

```
        N    R       Lambda
[1,]   0 0.00000 257900.00
[2,]   1 0.08365 235000.00
[3,]   1 0.15310 214100.00
[4,]   1 0.21080 195100.00
[5,]   1 0.25860 177700.00
[6,]   1 0.29840 161900.00
[7,]   1 0.33140 147600.00
[8,]   1 0.35880 134400.00
[9,]   1 0.38150 122500.00
[10,]   1 0.40040 111600.00
[11,]   2 0.42020 101700.00
[12,]   2 0.44490  92670.00
[13,]   2 0.46540  84440.00
[14,]   4 0.48960  76940.00
[15,]   5 0.51630  70100.00
[16,]   5 0.53880  63870.00
```

```
         N      R        Lambda
[17,]   7 0.55970  58200.00
[18,]   8 0.58160  53030.00
[19,]  10 0.60200  48320.00
[20,]  12 0.62570  44030.00
[21,]  12 0.64670  40110.00
[22,]  12 0.66410  36550.00
[23,]  13 0.67860  33300.00
[24,]  14 0.69190  30350.00
[25,]  14 0.70310  27650.00
[26,]  17 0.71430  25190.00
[27,]  20 0.72440  22960.00
[28,]  22 0.73490  20920.00
[29,]  25 0.74490  19060.00
[30,]  30 0.75410  17360.00
[31,]  33 0.76320  15820.00
[32,]  35 0.77140  14420.00
[33,]  38 0.77900  13140.00
[34,]  40 0.78550  11970.00
[35,]  50 0.79220  10910.00
[36,]  51 0.79840   9937.00
[37,]  54 0.80360   9054.00
[38,]  59 0.80820   8250.00
[39,]  61 0.81230   7517.00
[40,]  63 0.81570   6849.00
[41,]  71 0.81880   6241.00
[42,]  76 0.82170   5686.00
[43,]  81 0.82420   5181.00
[44,]  89 0.82650   4721.00
[45,]  98 0.82870   4301.00
[46,] 104 0.83090   3919.00
[47,] 112 0.83270   3571.00
[48,] 113 0.83440   3254.00
[49,] 125 0.83590   2965.00
[50,] 133 0.83720   2701.00
```

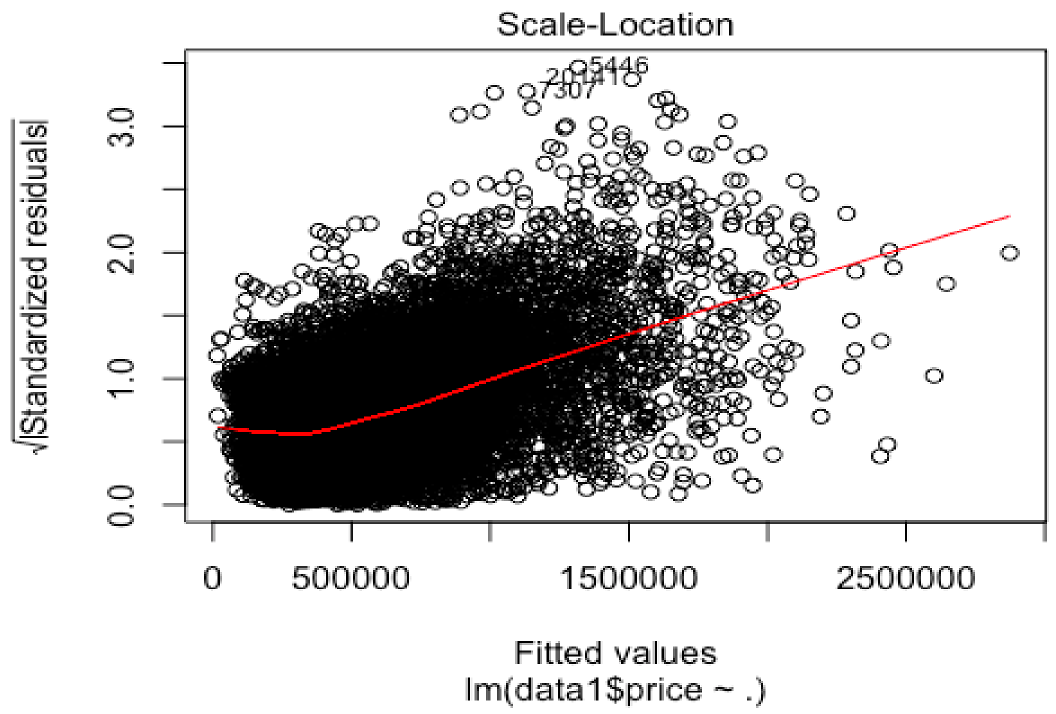
```
           N      R        Lambda
 [51,] 143 0.83830   2461.00
[52,] 154 0.83940   2243.00
[53,] 159 0.84030   2044.00
[54,] 165 0.84110   1862.00
[55,] 174 0.84180   1697.00
[56,] 186 0.84250   1546.00
[57,] 194 0.84310   1409.00
[58,] 195 0.84360   1283.00
[59,] 199 0.84400   1169.00
[60,] 203 0.84440   1065.00
[61,] 213 0.84470    970.80
[62,] 219 0.84490    884.60
[63,] 227 0.84520    806.00
[64,] 237 0.84540    734.40
[65,] 247 0.84560    669.20
[66,] 254 0.84570    609.70
[67,] 261 0.84590    555.50
[68,] 263 0.84600    506.20
[69,] 266 0.84620    461.20
[70,] 270 0.84630    420.20
[71,] 273 0.84630    382.90
[72,] 276 0.84640    348.90
[73,] 284 0.84650    317.90
[74,] 287 0.84650    289.70
[75,] 287 0.84660    263.90
[76,] 289 0.84660    240.50
[77,] 294 0.84660    219.10
[78,] 296 0.84670    199.70
[79,] 300 0.84670    181.90
[80,] 302 0.84670    165.80
[81,] 302 0.84680    151.00
[82,] 303 0.84680    137.60
[83,] 304 0.84680    125.40
[84,] 304 0.84680    114.20
```

```
          N     R        Lambda
[85,] 307 0.84680    104.10
[86,] 307 0.84680     94.85
[87,] 306 0.84690     86.42
[88,] 307 0.84690     78.75
[89,] 308  0.84690     71.75
[90,] 309 0.84690     65.38
```

We used a different approach in R and choose lambda value = 4721. Even though the R square has slightly decrees from original model, we cut off the about 232 attributes.

After locating the outlier and removing them as well as re-standardized the attributes and then re-generated the model (lm2):

- Number of predictors: 89 (decrease 232 predictors from orig 321)
- Multiple R-squared: 0.8243 (slight decrease from lm1 0.8471)
- Adjusted R-squared: 0.8236 (slight decrease from lm1 0.8448)
- MSE =1.37e+5 (exact 137246.35)



3. <u>Model 3 SVM Model:</u>

Using the Linear Regression model that identified 140 predictors at the best MSE for the price prediction at lambda value of 2234.8803and index 50 we moved to building an SVM model. The model was created using the following steps:

i. *Generated price ranges*: We created the labels for our dependent variable Price by dividing the data in 4 categories as below:

Price	**Class**
price < 321,950	**0**
Price>=321950 & Price<450000	**1**
Price>=450000 & Price<645000	**2**
Price>=645000	**3**

Note: We divided the categories in a balanced manner using the three quartiles and median and thus chose accuracy to be the indicator of a good mode

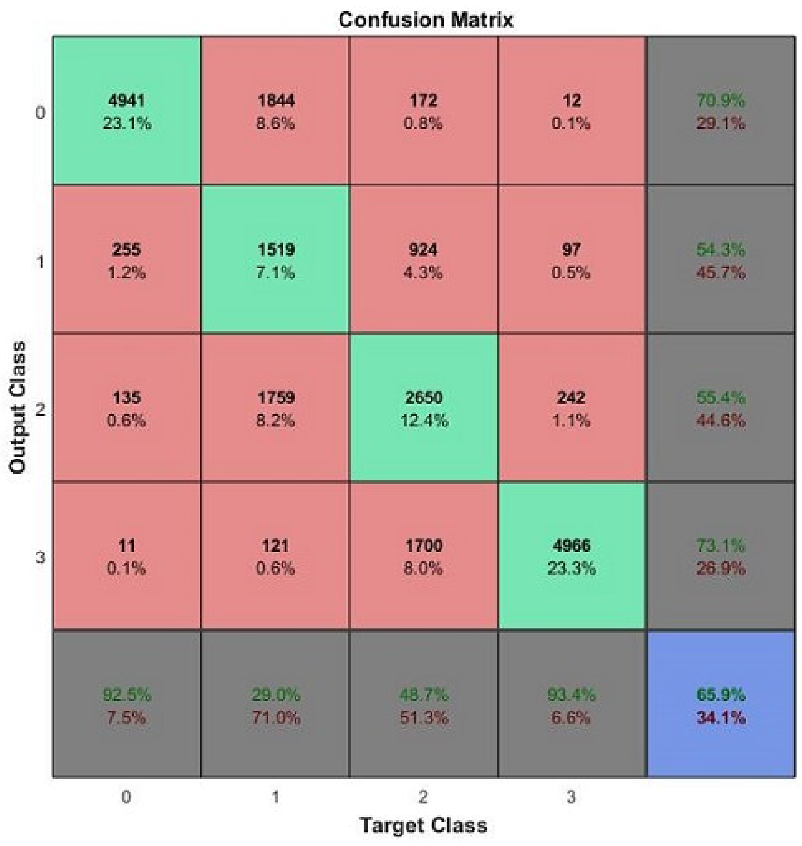
ii. *Utilized 140 predictor linear model and build different SVM models below*:
- We started with a multi-class SVM model with 10-fold Cross Validation.
- We built "One Vs ALL" model for multiclass as we had divided the price into 4 labels.
- We then ran the kfoldLoss for each of the 10 models generated and checked for the accuracy of the one of 10, which had the least loss.
- used different values of box constraint to check the impact on accuracy
- Calculated the accuracy by plotting a Confusion Matrix for Target Class vs Output Class
- We also used different combinations to build our models to see what impact they make on the accuracy of our model which were as follows:

Kernel Function	**Box Constraint, C**	**Accuracy**
None	None	65.9%
Gaussian	None	83.1%
Gaussian	0.000001	58%
Gaussian	1	83.1%
Gaussian	2.5	86%
Gaussian	3	86.7%
Gaussian	3.5	87.4%
Gaussian	4	87.4%
Gaussian	4.5	87.4%
Gaussian	5	88%
Gaussian	10	89.2%
Gaussian	**25**	**90.8%**
None	25	66.5%

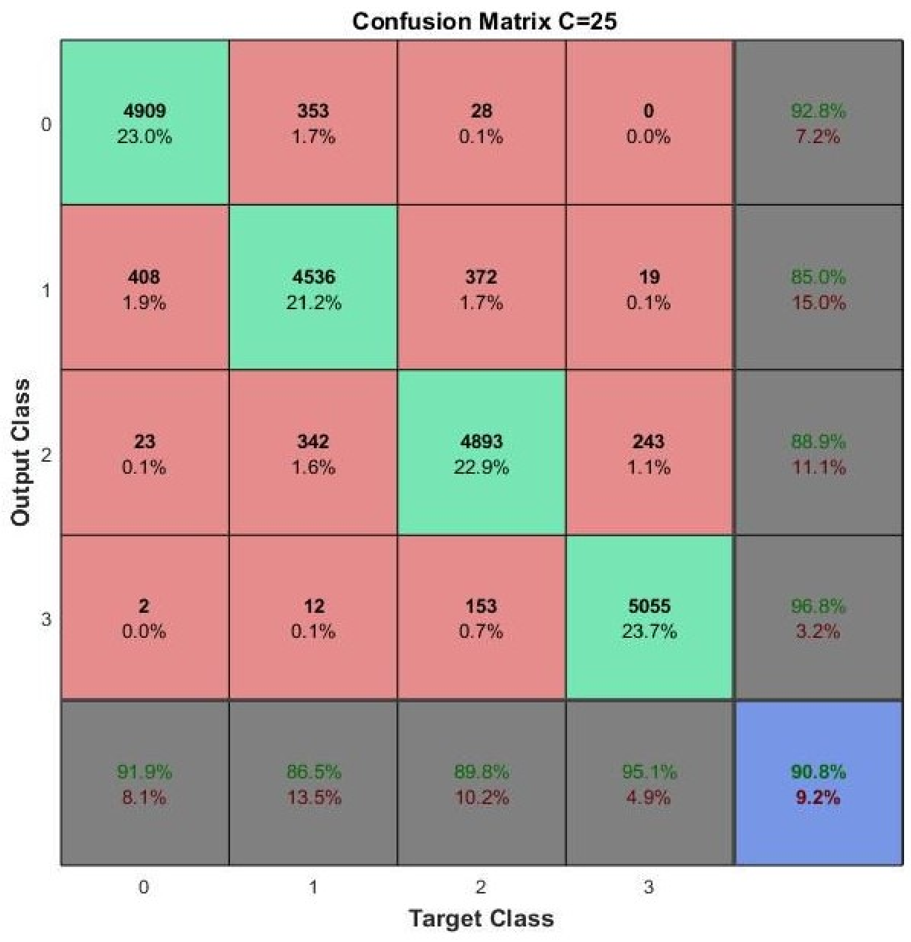
iii. *Outcome of all trial SVM models above*: After running all these models above we came to the conclusion that Kernel function definitely improved the accuracy of our model from 65.9% (without kernel) to 83.1%, as kernel tries to find out the boundaries in the infinite dimension space and thus is able to improve the classification of labels. Moreover, when we further added another parameter, box constraint C and gradually improved its value from C = 1 to C = 25, accuracy improved quite impressively from 83.1% to 90.8%. Which was phenomenal! Thus we chose the final model to be the one with Kernel function and box constraint C 25.

The plots for initial model for SVM without Kernel function and the final one with kernel and box constraint 25 are below:

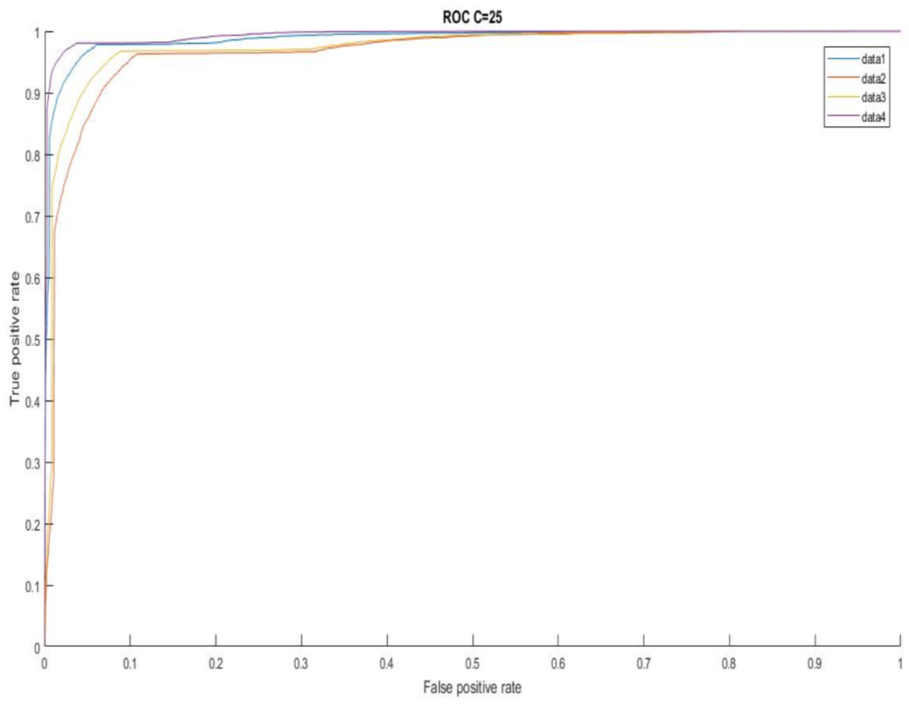
I. SVM model without Kernel function and no box constraint:



II. SVM model with Kernel function and box constraint 25:


Confusion Matrix C=25

Output Class \ Target Class	0	1	2	3	
0	4909 23.0%	353 1.7%	28 0.1%	0 0.0%	92.8% 7.2%
1	408 1.9%	4536 21.2%	372 1.7%	19 0.1%	85.0% 15.0%
2	23 0.1%	342 1.6%	4893 22.9%	243 1.1%	88.9% 11.1%
3	2 0.0%	12 0.1%	153 0.7%	5055 23.7%	96.8% 3.2%
	91.9% 8.1%	86.5% 13.5%	89.8% 10.2%	95.1% 4.9%	90.8% 9.2%



4. Model 4 Neural Network – Patternnet with Ranges Applied

For the Patternnet Neural Network model we utilized the base list established after taking out the necessary outliers but prior to narrowing the list down to 140 predictors as we wanted to test if all of the predictors included would produce better results than the model with only 140 predictors. Plus since we didn't know exactly which combination of features the Neuro Network would find most useful we did not want to eliminate any on our own and create a sparser model right away.

i. *Generated price ranges*: This model also used the same Price categories as the SVM model to compare accuracy:

Price	**Class**
price < 321,950	**1**
Price>=321950 & Price<450000	**2**
Price>=450000 & Price<645000	**3**
Price>=645000	**4**

Note: We divided the categories in a balanced manner and thus chose accuracy to be the indicator of a good model.

ii. *Patternnet Model/Network variations:*
- All trial models utilized a network ReLu transfer function
- All trial models utilized a network Scaled Conjugate Gradient Descent
- Initially we were utilizing the network default min gradient (1e-5) but after a few trials we found the learning would stop rather quickly due to the gradient and the accuracy rate was at best around 55%. Therefore after the first few trials we adjusted the min gradient to 1e-10.
- All trials had Epochs set to 2000 and our trials never reached that many, the average was around 700 depending on the make-up of the trial.
- As with the min gradient parameter we initially started out with the goal default (0) but found we would only get to a couple hundred Epochs before the model stopped learning and the accuracy rate was at best around 50%, therefore we adjusted the performance goal to 1e-20 and this effected our model positively.
- All other parameters were kept as default.

iii. Patternnet Model Neuron/Layers variations:
- As we have a larger data set we thought the larger number of neurons per layer would be more beneficial but we quickly found that the accuracy was lower with anything over 10 neurons long.
- We also thought that going up to 4 layers would produce a better accuracy with the feature combination of feature combinations but found that 2 layers was producing the highest accuracy for this Patternnet model.
- We also tested out variations to the number of price ranges we had, from 4 to 5, then to 6 and even 10 ranges but we never saw the accuracy never increase over 74.4%. We found it difficult to create proportionate data throughout the various ranges as we had such a high number of home prices near our median, therefore some of the ranges would have 100K price difference while others would have only 1,000. This caused the data to become more unbalanced and our accuracy more unreliable. We knew that less ranges would not achieve our goal strategically, even though the data may be more balanced and the accuracy higher due to less categories, it will not help

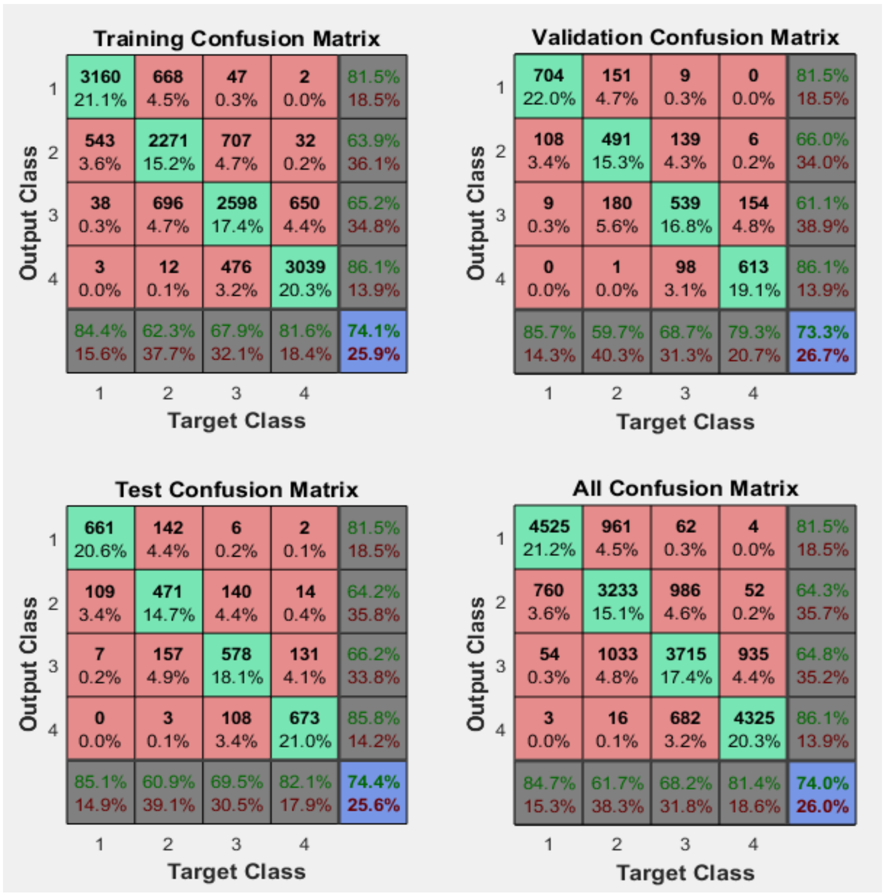
anyone understand a house price if the range is somewhere between 0 and 450,000. Therefore we stuck with 4 categories as it provided the best accuracy results and worked well to compare with other models in our overall analysis.

- Throughout all of the trials a consistent trend we noticed was out of the four categories the 1st and 4th always had the highest Precision and Recall (at least 70% or higher) but the 3rd and 4th categories had ranges in the 45-60%. This does make sense as the 1st and 4th categories have more extreme ranges and would be less likely to have observations near the support vector line while 2nd and 3rd are closer in range and would have more complexity and more support vectors in infinite dimensional space.

All conducted trials are below with the total number of layers and neurons per layer.

Total Layers	**Total Neurons (each layer)**	**Accuracy**
1	100	56.3%
1	50	60.1%
1	10	60.9%
1	5	67.2%
2	100 50	59.9%
2	10 5	69.3%
2	**5 3**	**74.4%**
2	6 3	70.1%
3	100 50 25	58.2%
3	50 25 10	63.5%
3	10 5 2	68.4%
3	5 3 2	70.3%
4	100 50 75 25	62.2%
4	10 5 3 2	69.8%

Patternnet Confusion Matrix of Model with 2 layers and [5   3] neurons and highest producing accuracy:



Training Confusion Matrix

Output Class / Target Class

	1	2	3	4	
1	3160 21.1%	668 4.5%	47 0.3%	2 0.0%	81.5% 18.5%
2	543 3.6%	2271 15.2%	707 4.7%	32 0.2%	63.9% 36.1%
3	38 0.3%	696 4.7%	2598 17.4%	650 4.4%	65.2% 34.8%
4	3 0.0%	12 0.1%	476 3.2%	3039 20.3%	86.1% 13.9%
	84.4% 15.6%	62.3% 37.7%	67.9% 32.1%	81.6% 18.4%	74.1% 25.9%

Validation Confusion Matrix

Output Class / Target Class

	1	2	3	4	
1	704 22.0%	151 4.7%	9 0.3%	0 0.0%	81.5% 18.5%
2	108 3.4%	491 15.3%	139 4.3%	6 0.2%	66.0% 34.0%
3	9 0.3%	180 5.6%	539 16.8%	154 4.8%	61.1% 38.9%
4	0 0.0%	1 0.0%	98 3.1%	613 19.1%	86.1% 13.9%
	85.7% 14.3%	59.7% 40.3%	68.7% 31.3%	79.3% 20.7%	73.3% 26.7%

Test Confusion Matrix

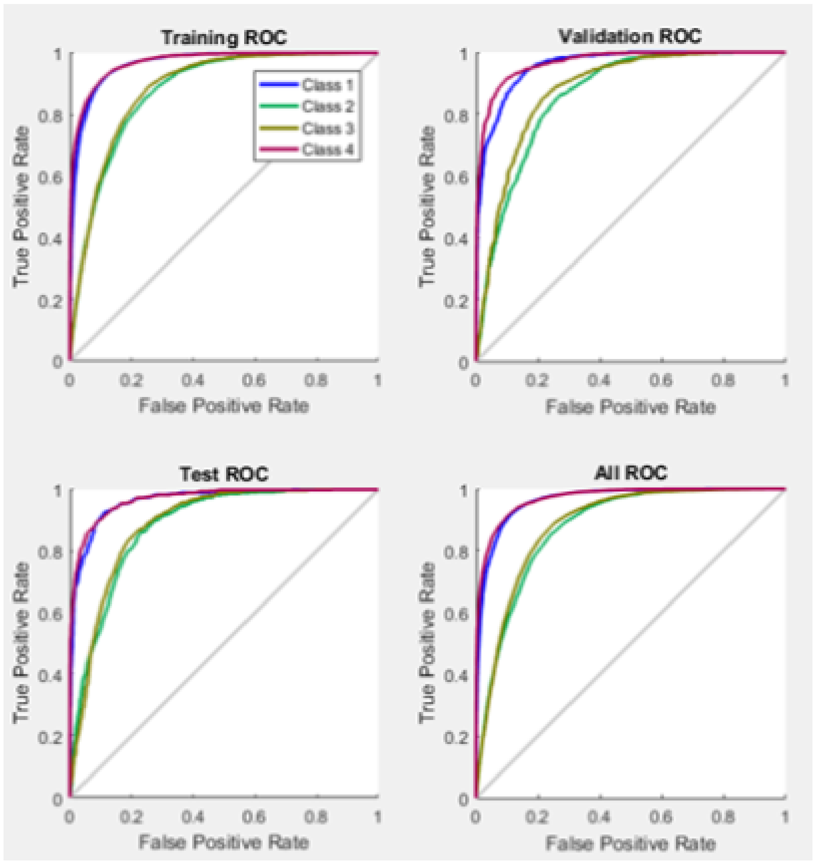
Output Class / Target Class

	1	2	3	4	
1	661 20.6%	142 4.4%	6 0.2%	2 0.1%	81.5% 18.5%
2	109 3.4%	471 14.7%	140 4.4%	14 0.4%	64.2% 35.8%
3	7 0.2%	157 4.9%	578 18.1%	131 4.1%	66.2% 33.8%
4	0 0.0%	3 0.1%	108 3.4%	673 21.0%	85.8% 14.2%
	85.1% 14.9%	60.9% 39.1%	69.5% 30.5%	82.1% 17.9%	74.4% 25.6%

All Confusion Matrix

Output Class / Target Class

	1	2	3	4	
1	4525 21.2%	961 4.5%	62 0.3%	4 0.0%	81.5% 18.5%
2	760 3.6%	3233 15.1%	986 4.6%	52 0.2%	64.3% 35.7%
3	54 0.3%	1033 4.8%	3715 17.4%	935 4.4%	64.8% 35.2%
4	3 0.0%	16 0.1%	682 3.2%	4325 20.3%	86.1% 13.9%
	84.7% 15.3%	61.7% 38.3%	68.2% 31.8%	81.4% 18.6%	74.0% 26.0%



5. Model 5 Neural Network – AutoEncoder with Ranges Applied

In building the AutoEncoder Neural Network we used the same dataset as the Patternnet as well as the same price ranges as the SVM and Patternnet models to constantly compare models for this dataset. The main difference between the AutoEncoder and Patternnet data set-up include the training and testing data. For the Patternnet we used the default 85% training and 15% testing data, while the autoencoder used 90% training and 10% testing. The testing was chosen randomly and we wanted to provide more training data to the AutoEncoder to use for learning the weights versus testing.

i. *Generated price ranges*: This model also used the same Price categories as the SVM model to compare accuracy:

Price	**Class**
price < 321,950	**1**
Price>=321950 & Price<450000	**2**
Price>=450000 & Price<645000	**3**
Price>=645000	**4**

Note: We divided the categories in a balanced manner and thus chose accuracy to be the indicator of a good model.

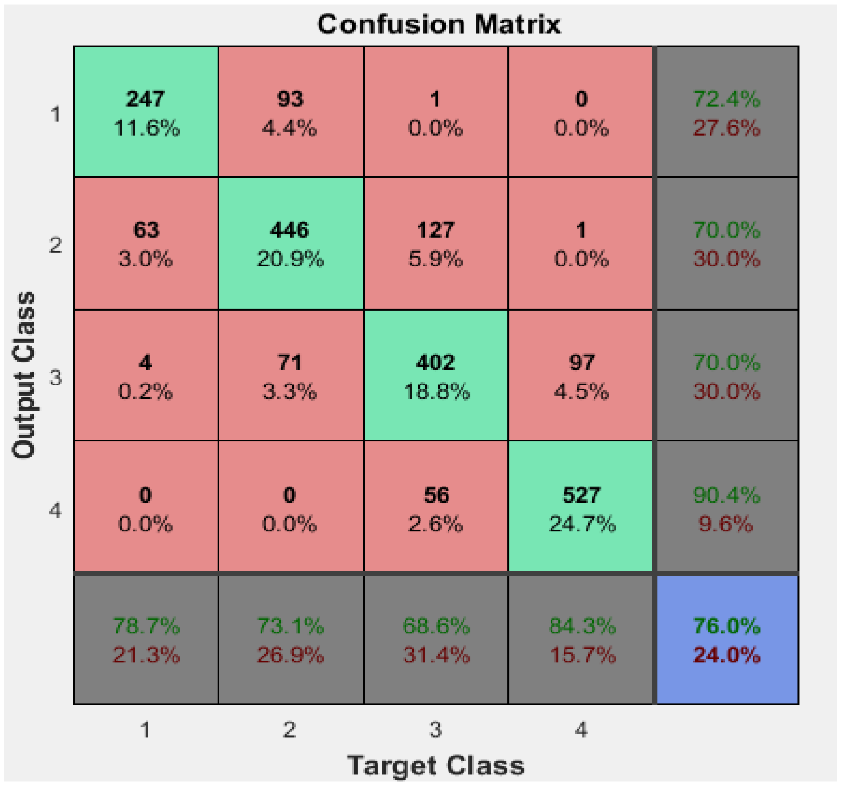
ii. *Autoencoder Model/Network variations:*
- All trial models had the ScaleData parameter set to false so the data would not be scaled while training the autoencoder.
- As we wanted to test the effects of sparsity within our models we conducted various trials using Sparsity Proportion of 0.05 (default), 0.10, and 0.15. As a lower sparsity proportion encourages a higher degree of sparsity we thought that we may gain better performance and accuracy with a lower degree of sparsity. We found an increase in accuracy using the 0.15 sparsity proportion but the best accuracy was produced using a value of 0.10. Each value was used for a third of the trials.
- The number of Epochs used throughout the trials proved to have a big impact on our accuracy as well. Many of our initial trials would stop learning due to the fact it reached the max epochs limit. We structured our epochs differently as we added more layers, lowering the max epochs every time we added another layer. What we found most interesting was that when we increased the number of epochs for the softnet layer we would get better results. Originally we did not anticipate the softnet layer needed as many epochs (around 200) but our final model actually had 700 epochs within this layer, which then allowed us to have more iterations during our fine tuning model. This was extremely beneficial.
- All other parameters were kept as default.

iii. *Neurons/Layers variations:*
- Unlike the Patternnet Neural Network we had more success utilizing deeper layers and more neurons within each layers. Along with testing the various epochs and sparsity proportion mentioned above we tried out a variety of variations with layers and neurons. We were surprised to see that the learning rates were not higher but if we had more training and testing data we may be able to increase the accuracy.
- Just as we noted in the Patternnet section, categories 1 and 4 had the highest Precision and Recall rates while categories 2 and 3 had the lowest. Category 4 in particular consistently had the highest rates which could equate to the extreme high prices, which made them easier to categorize.

All conducted trials are below with the total number of layers and neurons per layer:

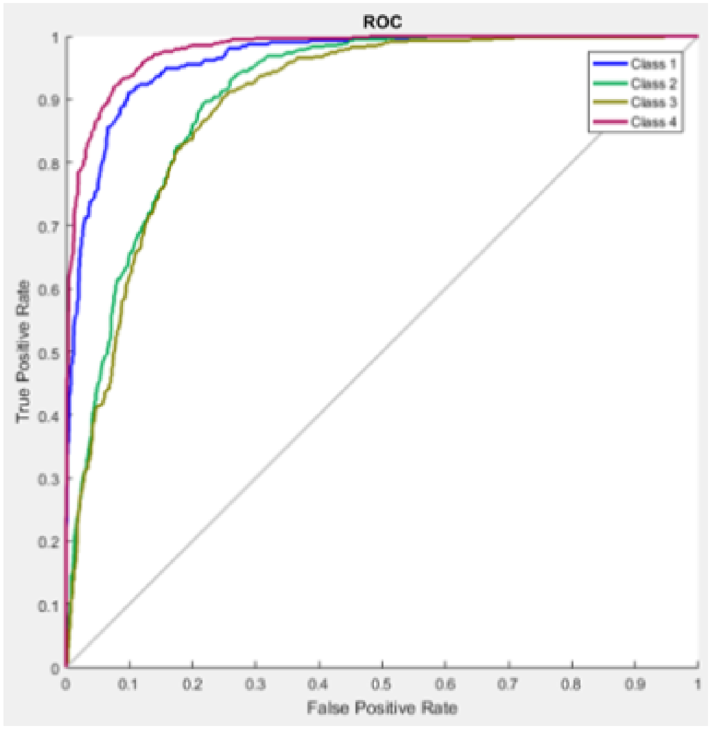
Total Layers	Total Neurons (each layer)	Deepnet Accuracy	Fine Tuning Accuracy
1	100	43.2%	56.8%
1	50	48.6%	59.2%
1	10	55.4%	64.4%
1	5	56.6%	65.7%
2	200 100	55.5%	67.4%
2	100 50	68.3%	70.0%
2	25 10	65.8%	69.8%
2	5 2	64.2%	67.7%
3	200 100 50	44.5%	70.2%
3	**100 50 25**	**68.6%**	**76.0%**
3	50 25 10	61.7%	69.8%
3	10 5 2	60.0%	66.3%
4	200 200 100 50	36.2%	56.8%
4	10 5 3 2	41.1%	54.3%

Autoencoder Confusion Matrix of Model with 3 layers and [100 50 25] neurons:



Output Class \ Target Class	1	2	3	4	
1	247 11.6%	93 4.4%	1 0.0%	0 0.0%	72.4% 27.6%
2	63 3.0%	446 20.9%	127 5.9%	1 0.0%	70.0% 30.0%
3	4 0.2%	71 3.3%	402 18.8%	97 4.5%	70.0% 30.0%
4	0 0.0%	0 0.0%	56 2.6%	527 24.7%	90.4% 9.6%
	78.7% 21.3%	73.1% 26.9%	68.6% 31.4%	84.3% 15.7%	76.0% 24.0%

We were very unsure why we could not get results higher than this 76.00% and did try numerous variations of layers and neuron levels. This included increasing the level of layers but we were consistently seeing results in the 60-70 range. With more time we would have liked to place our Autoencoder into our SVM model and see if we could increase our overall accuracy. We also could have played around with the total amount of training versus testing data we had.



4. OUTCOME

[discuss the outcome and what we have learned. How do we know we are right? Give examples of how our technique works]

Looking over the models created throughout our analysis we concluded at this time our SVM model produced the best prediction results for our 4 price ranges. At the 90.8% accuracy rate, we understand this is a good result but moving forward we would like to continue to find ways to improve this over potentially breaking down more price ranges using the SVM (especially within the mean range of the data, since that is where our results have the most error). We would also like to improve our Autoencoder model to see if we can get that into the 90% range by potentially adding more training data, such as data from 2016. Or by adding in more predictors to take into account such as school district rating, crime data, income values or taxes since they could either have large impact on the selling price of a house or be a key selling factor we are missing from this dataset. With more time we would also like to incorporate a geographic aspect to this analysis that would allow a home buyer to enter their wish list and credentials then populate a map of areas they could consider. Thinking back to the reason we started this analysis, there is endless possibility in the housing market and this is just the start.

5. CHALLENGES ENCOUNTERED

Our group encountered varying challenges that acted as great learning experiences for future projects:

1. <u>During our first model approaches:</u>
   a. Ranges - As we had many categorical variables to incorporate we initially thought the best approach would be to create ranges for each variable, we did this by simply taking the total and dividing it by four. Once we started running a few models we realized that the ranges were not evenly distributed causing a high amount of error. Therefore we re-evaluated our approach and conducting ranges based on the data's four quartiles. This provided a more evenly distributed dataset but did set us back on generating our models.
   b. Geo Variables - When starting the project we suspected that the outcome would rely heavily on the location predictor. The geographical information we had available included the zip code, latitude and longitude, but to use categorical variables for every zip code would have had over 150 dummy variables. Therefore we looked for new ways to decrease the granularity but as we were working with data from only one County we had limited hierarchy steps to choose from. We originally decided to do Congressional districts by looking up each zip code and choosing which district they fell in, but we realized the Zip Codes would overlap with the Congressional District and without the exact address we had no way of knowing which District they actually fell in. We did not want to conduct any guessing so we looked for batch uploading processes that would allow us to apply City to our records from the longitude and latitude. https://www.doogal.co.uk/BatchReverseGeocoding.php We ran our code though this Doogal batch service but found it would take days to run and get us each address. We abandoned this idea and looked for new ways to incorporate zip code, latitude and longitude.
   c. Speed – as we have so many categorical variables, which resulted in over 300 dummy variables, we found that, on average, our models would take at least 10 minutes to run in Matlab but if we ran our code in R we found much faster turnaround. We ended up using a combination of Matlab and R to test our various models to see who would produce the best results.
2. <u>Identifying Base Dataset to use:</u> Our initial approach was to have each member of our group get comfortable with the data and generate a few models on our own. Then we would come together and discuss. What we came to find after our first meeting was no two team members were getting the same results, they were all different. Although we all wanted to create our own models to compare and contrast we realized we needed to work together to identify one base dataset to use to create any future models.
3. <u>Data Leakage:</u> As a team we reviewed the original dataset and removed bad data (Bedrooms 0, Price 0$) discovered early on. What any of us failed to recognize was the fact that two of our predictors were causing data leakage (sqft_above and sqft_basement together calculate sqft_Living). This by far was our biggest issue as we did not discover the data leakage until late in our process after we had created many models. The effect of this discovery improved our outcomes but caused us again to rework our base dataset and all models.
4. <u>Timing constraints:</u> Majority of our models required regularization but when running regularization functions in MATLAB our processing time would typically take anywhere from 2 to 3 hours to complete. As it worked best to meet in the evenings our models would not event complete running until anywhere between 8 or 9pm. In order to save time we generated several models after lasso regression with modified lambdas and screen printed

the information to compare and contrast with our team members. We also uploaded the final model to One drive but ran into the tech issues with saving our results. This resulted in us not capturing all of the predictors and values we needed, such as the lambda value. This lead us to re-running our lass model again, which resulted in another 2.5 hour wait time.

5. Technology Issues**:** Members of the team worked in either R, Matlab 2017 or Matlab 2016 and after a few weeks it was clear that code that worked within the 2016 version did not run on the 2016 version. This caused setbacks in code sharing and again caused different results within our models.
6. Team Meeting Coordination**:** we all have very different schedules as well as commute to the school and back. So, getting everyone to meet at a particular date and time was difficult.

6. CONCLUDING REMARKS

Overall this analysis produced a SVM model with a 90.8% accuracy rate for predicting home prices and identified that the total square feet within a home, the latitude location (R had zip code, but both were in location category) and view rating predictors had the largest impact on the price of a home within Kings County, WA. We speculated that View may be more personalized to Kings County as they have the mountain and ocean as possible views, where somewhere like Chicago may not consider view as important as walk score. Taking this further more emphasis and time should be spent on producing a high-accuracy deep neural network but at least we are confident that the SVM model is a great first step.

7. REFERENCES

- CC0: Public Domain License, September 2016, House Sales in King County, USA, Retrieved from: https://www.kaggle.com/harlfoxem/housesalesprediction
- Rosenberg, Mike, June 6, 2017, Seattle Times: *No Escape for Priced-Out Seattleites, June 2017: Home Prices Set Record For an Hour's Drive in Every Direction*, Retrieved from: http://www.seattletimes.com/business/real-estate/no-escape-for-priced-out-seattleites-home-prices-set-record-for-hour-drive-in-every-direction/