Predicting House Prices in New York City using regression

What is our dataset and problem domain?

Our group initially had agreed on a dataset revolving around reviews given to the best-selling games on the video-game distribution service, Steam (Mahendra, 2019). However, we realised that this dataset was limited in usable features and having a greater number of features would allow us to get more comparable results from our models. We later agreed on a dataset found on Kaggle which detailed every building sold in the New York City property market over a 12-month period (City of New York, 2017). Not only did this dataset many more features to work with, but also presented us with an extremely relevant problem domain: "*Could we use machine learning models to predict housing prices within NYC for those wanting to buy or sell property?***".**

One of New York's greatest problems is the availability of affordable housing, with the average monthly rent for an apartment "increasing by almost 40%" in the last 20 years (NYC Housing, n.d.)). Such a demand for affordability could see machine learning as a solution, by displaying to buyers the most affordable properties against other features such as the year the property was built and its gross square feet. Furthermore, a research paper into real estate opportunities found that machine learning could also be used to help investors by "identifying opportunities in the real estate market in real time such as houses that are listed with a price substantially below the market price." (Alejandro Baldominos, 2018)

Were our model classification or regression, what models did we use?

After investigation the characteristics of our data, we realised we had a regression problem as we were planning to predict continuous data (the house prices of properties across a year-long period) using mostly discrete data. We first performed a feature engineering stage in order to discover what features were most relevant to attain a high predictive performance. We had some features that were empty or just not relevant to our problem domain, so we removed these from our dataset, cutting our total features from 22 to 14.

The feature engineering stage also helped us understand the several machine learning algorithms we wanted to test, which included: **linear regression, regression trees, random forest, and support vector machines,** whilst identifying the advantages and handicaps of each.

Did we have any missing data? How did we cope with it?

Data cleaning had to be done to filter out missing, empty and null values which would have severely affected the accuracy of our model results. For instances of null in our dataset, we replaced these with the value 'NA' instead: "*df = pd.read_csv(filename_read, na_values=['NA', ' - '])*". This allowed us to then replace instances of 'NA' with the median value of said feature. We used this for 'sale_price' instead of removing data as it would have compromised the quality of our models by reducing the sample size – this was done using *"med = dfCon['SALE PRICE'].median()"* and *"dfCon['SALE PRICE'] = dfCon['SALE PRICE'].fillna(med)"* We also removed any existing columns with missing sale prices using the following: *"df = df[df['sale_price'].notnull()]".*

The focus of the task were properties and not every 'building type', so we removed entries where the building class category matched public buildings such as hospitals and warehouses. By encoding the column as numerical, it was easy to use a Boolean expression to filter this data out. *"df = df[df['BUILDING CLASS CATEGORY'].astype('float') <= 18]".*

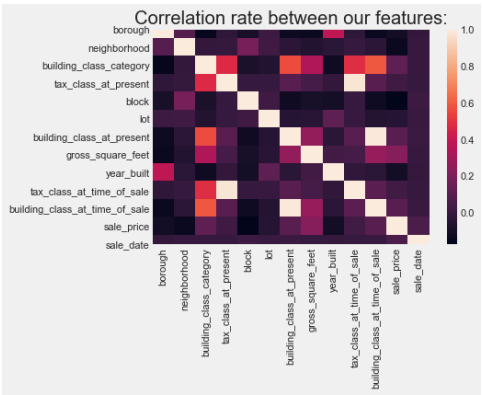
To ensure compatibility with the models working with the dataset, we then converted the data types of both the categorical and numeric columns to their respective types. The categorical data would be used as the features while the sale price numeric one would be the target. *dfEncode['column'] = pd.to_numeric/categorical().*

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download

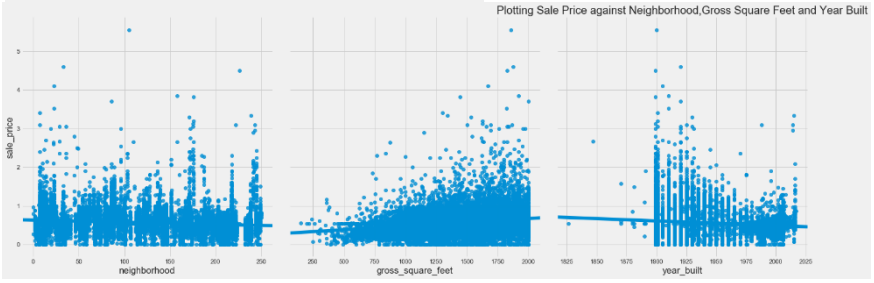
What techniques did we apply to understand our dataset?

One of our immediate problems were overlapping ‘sale_prices’ values on our models as they were six figures long, so we instead divided all instances of ‘sale_price’ by one million – this would be easier to interpret in a graphical format. We also had to sort our data by ‘sale_date’ from earliest to newest by changing the date format from American to English and to of type “datetime”, this made working with our dataset easier and allow us to see if there was a period where more homes were sold.

Before we could start applying machine learning models to our dataset, we first had to understand how our features correlated. We created a heatmap using the seaborn library documentation (Waskom, 2018) to find out how our features correlated.

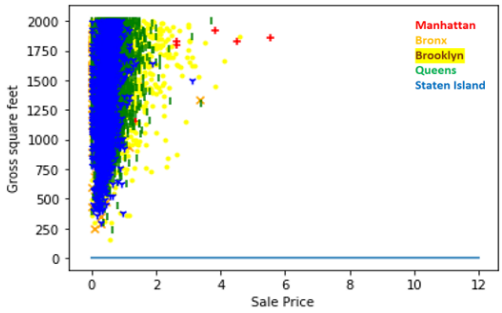


The brighter the colour, the closer the relationship between features (with 1.0 being optimal). As you can see, it’s quite apparent that there aren’t many features with a positive correlation between them. This was particularly worrisome considering ‘sale_price’ also didn’t have many strong correlations amongst other features.



To depict this further, we mapped out our ‘sale_price’ feature against some of our other features, to see whether other correlations could be found. As you can see, there seems to be a strong relationship between ‘gross_square_feet’ and the ‘sale_price’ of a property, however the same can’t be said for ‘neighborhood’ and ‘build_date’. We expected a difficult to review relationship between ‘neighborhood’ and ‘sale_price’ as a properties neighbourhood isn’t a definitive factor when considering sale price, unlike gross square feet.

For ‘year_built’, one could argue that there aren’t as many properties in the year 2000 as there are in the 1900-1950’s, hence why ‘sale_price’ appears higher than in 2000.



Separating the dataset into the boroughs shows that there is a close clustering of properties being within the $2 million mark, but there are a few in Manhattan and Brooklyn that reach a higher price of $4 to $6 million as building size increases. This indicates that these boroughs may have more expensive properties compared to the other boroughs.

How did we encode our input variables?

Many of the features were categorical data with the only real numerical one being the sale price which we wanted to predict. With our remaining columns, we attempted one-hot encoding before settling with LabelEncoder due to the former changing the column headers to numerical as well. With the latter, we could do reverse encoding easily so that if we

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download

plotted data, we could understand what was being shown (E.G – Instead of Neighbourhood 1, we would see Alphabet City).

This also proved to be helpful with filtering our data using a simple Boolean expression on building class category rather than having a whole list of what names we wanted to remove.

What were the criteria we selected for model accuracy evaluation tools?

Research done online found that "RMSE is the most popular evaluation metric used in regressed problems" (Srivastava, 2019). Looking at our models, we established we could use RMSE on them to keep our result accuracy metrics consistent across our models, evaluating the one that worked the best using this. We aimed to keep the RMSE within 25% of our mean of the regressors as opposed to 10% due to the complex nature of our dataset and how interdependent our features were.

What were our model outputs and the evaluation + analysis of our results?

Linear Regression (Multiple Regression): Multiple regression is used to observe the relationship between several independent variables and a dependent variable. For our dataset, we used the rest of our features to predict the sale price of properties, which could be beneficial to real estate agents analysing the value of houses or those looking to purchase a property in New York City.

One of biggest advantages when using linear regression is its ability to identify outliers or anomalies - such as what features have a strong correlation to sale prices and what don't. On the other hand, outliers could also cause negative effects on the results of regression.

Another problem regarding regression is that it assumes all data is independent, suggesting one feature has nothing do with others. This isn't always sensible and had effects on our results, such as where the results of plotting 'sale price' against 'neighborhood' led to scattered values (as shown above), in comparison to 'gross_square_feet' which is not affected in the same way, as gross square feet is extremely definitive of price.

Our hypothesis was that linear regression would most likely produce the worst results as our dataset was clearly not very linear. In our first attempt at linear regression, our results were extremely inaccurate with an RMSE value of 9.2 against a mean of 1. We realised that such high results were being produced due to how skewed our data was, with values that were clearly inaccurate.

This included properties that had a sale price and gross square feet radius of 0, and several hundred properties that had a sale price of $10 which were removed as they were outliers to our data. We also filtered out properties above the range of $10 million and gross square feet above 2000, as these often weren't residential properties and instead whole blocks of housing and skyscrapers.

```
df = df[df['sale_price'] !=0]
df = df[df['gross_square_feet'] !=0]
df = df[df['year_built'] !=0]
df = df[df.sale_price != 0.00001]
df = df[(df['gross_square_feet'] <= 2000)]
df = df[(df['sale_price'] <= 10)]
```

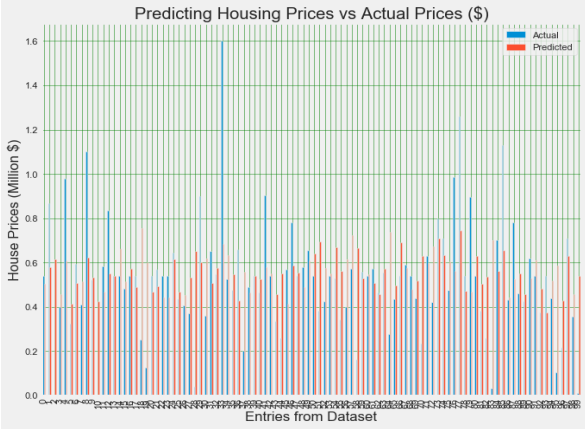
Using these filters considerably lowered our RMSE to 0.7 and our mean to 0.6, and although this also reduced the size of our dataset, we felt that we still had enough data to work with. As mentioned in our section about data cleaning, we realised we wanted to focus on residential properties instead of all properties, so filtering building class category properties above 18 (which was public buildings such as hospitals and office blocks) also had a drastic effect on our RMSE result, lowering our RMSE to ultimately lay at around 50% of our mean.

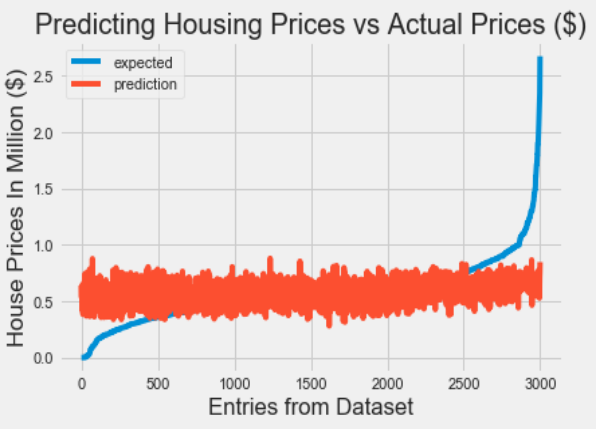
```
Our Mean value is: 0.5574388313872948
Our RMSE using Linear Regression is:
0.2713014271375793
```

As you can see from the graphs below, the expected vs predicted results from our bar chart initially does not seem too inaccurate, albeit a few anomalies. Our second chart tests this out further but on 3000 entries from our dataset, and it's clear to see that our predicted results do not match our expected results closely.

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download

Below show the results of linear regression with a training and testing split of 75% and 25%





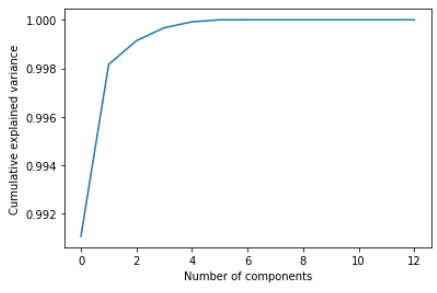
Overall, our hypothesis that linear regression would produce the least accurate results turned out to be true. After much tinkering with features used, filtering properties out based on their size and price, we managed to reduce our RMSE to a value much lower than what we started with, but still not to a point that could be considered good.

Although our RMSE could have been reduced further by filtering out even more properties by a lower max sale price and gross square feet, this would end up reducing the size of our dataset to a point which did not seem feasible and was instead just hiding our data for the sake of a lower RMSE. Such a huge depreciation in the size of our data set needed to be justified, and we felt this wouldn't have been reasonable.

Support Vector Machine (Regression): Support vector machines can be used for both classification and regression. They use a hyperplane to separate different classes of data and have different kernels to adapt itself to the data depending on which kernel is selected. Based on the results of linear regression, our hypothesis was that we would receive yield better results using the RBF kernel and would likely produce the best results of our models.

Using all features yielded bad accuracy. The linear kernel had an RMSE of 30 compared to a mean of 0.56 and was predicting negative values. The RBF kernel proved even worse with it predicting the same result across the dataset. To resolve the RBF models, we tried amending the parameters for the regressor. By changing the epsilon value from 1.0 to 0.1, it was able to produce different predictions, but the accuracy was still low. So, we attempted using PCA to reduce the dimensionality of the features. This resulted in a similar outcome.

A post about improving accuracy suggested using standard scaler on the input data. (Arora, 2016). Using it on both the features targets caused some inconsistencies with the actual values on all the models. But when it was applied only to the features, the results were drastically improved. The expected results remained unchanged and the linear kernels were no longer producing negative predictions. By applying standard scaler, it also resulted in quicker execution times which meant we were able to use the whole dataset on the models and get results quicker. This allowed us to get results that were further improved upon.



It was apparent that while we expected PCA to improve the results, they ended up producing the worst results when setting n_components = 2. Plotting a graph of components to explained variance using function *pcaPlot()* showed that I would need to set n_components to at least 4 to get high variance. When reapplied, the results were improved for these models with an accuracy score increase of around 0.1, but they still weren't as good the models without PCA.

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download

Attempts	Results
svr2RBF (All features)	Accuracy Score: 0.3919518540171755 Mean: 0.5638754162436547 Root Mean Squared Error: 0.23879302520694196
svr2Lin (All features)	Accuracy Score: 0.1170912940658503 Mean: 0.5638754162436547 Root Mean Squared Error: 0.2877468332992549
svr3RBF (PCA)	Accuracy Score: 0.10564962440574444 Mean: 0.5638754162436547 Root Mean Squared Error: 0.28960529638474153
svr3Lin (PCA)	Accuracy Score: 0.07309288101455824 Mean: 0.5638754162436547 Root Mean Squared Error: 0.29482938199494135
svrRBF (1 feature)	Accuracy Score: 0.04767716137912482 Mean: 0.5638754162436548 Root Mean Squared Error: 0.2988441461074952
svrLin (1 feature)	Accuracy Score: 0.047887435110971266 Mean: 0.5638754162436548 Root Mean Squared Error: 0.2988111517591462

Looking back at the dataset itself, there were some entries where the gross square feet was 0 which isn't accurate. I also felt the id column wasn't relevant for predicting. Removing these values did have a bit of an increase in accuracy since there were only a handful of these entries.

The table has the results of each attempt from best to worst results. This good thing about using SVR is that it yielded better results than linear regression. However, it couldn't get a high prediction accuracy despite the fine tuning. One reason could be due to SVM not working well on noisy data requiring a lot of filtering as was the case with our dataset. As mentioned with linear regression, further filtering wouldn't seem feasible making our dataset smaller than it should be.

Overall with SVR, it further emphasises the fact that our dataset is not linear with the regressors using linear kernel producing lower results than its RBF counterpart. It produced better results than linear regression and proved that support vector machines work better when provided more training points as evident by applying the whole dataset compared to only 100 entries.

Decision Tree Regression: One of the main advantages of decision trees is that they are easy interpret, not requiring much preparation during the pre-processing of data, as decision trees are not affected by missing values. Yet slight changes in the data being used can cause a decision tree to become unstable. Furthermore, decision trees are also expensive when it comes to the time and space taken to run and are not quite suited for applying regression with predicted continuous values. (K, 2019)

Our hypothesis was that decision trees would ultimately yield a lower RMSE value than linear regression as our dataset isn't very linear and would suit a decision tree model instead. Linear regression is also more at risk from outliers in comparison to decision trees.

Splitting the training and testing data by 75% and 25% respectively, the outputs below were given when the training values were fitted with the DecisionTreeRegressor and using these values to predict the sale price using all the features in our dataset. The below results show our actual vs predicted sale prices based on using the following hyperparemeters: 'criterion=mse', 'max_depth=15' and 'min_samples_split=8'. These hyperparameters set our criterion to MSE, set the longest path to the next leaf node (15) and set a minimum of 8 samples in order to split the code. This gives an RMSE value of roughly 0.29 where the mean is 0.57.

```
#DecisionTreeRegressor and fit regressor tree
regressor = DecisionTreeRegressor(criterion='mse',
                max_depth=15, min_samples_split=8)
regressor.fit(X_train, y_train)
#Predict new value
y_pred = regressor.predict(X_test)
#dataframe displaying actual price compared to predicted
prices = pd.DataFrame({'Actual':y_test,
                       'Predicted':y_pred})
```

	Actual	Predicted
0	0.445	0.454335
1	0.649	0.529812
2	0.490	0.390000
3	0.900	0.731947
4	0.650	0.546667
...	...	...
4995	0.445	0.360112
4996	0.250	0.558000
4997	0.180	0.414099
4998	0.410	0.383076
4999	0.400	0.518704

As this is not quite accurate, we did some fine tuning by adjusting the hyperparameters. We changed our max_depth to 9 (previously 15) and halved the minimum number of samples given before splitting a node from 8 to 4.

```
regressor = DecisionTreeRegressor(criterion='mse',
                max_depth=9, min_samples_split=4)
```


Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download

From this tuning, we get some improved results for our root mean squared value:

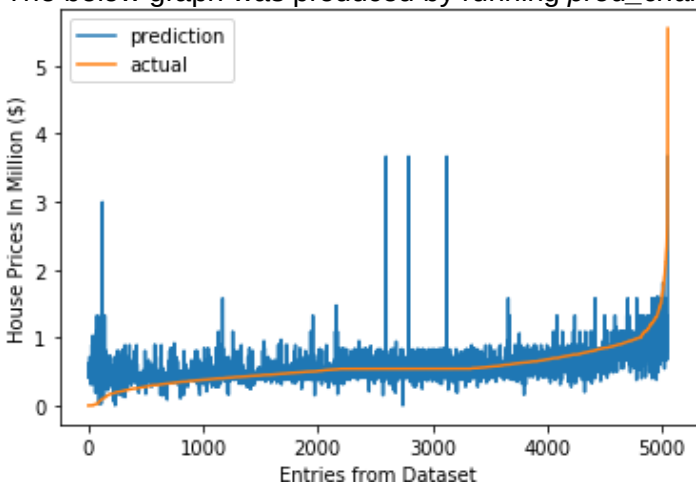
	Actual	Predicted
0	0.445	0.460432
1	0.649	0.559093
2	0.490	0.480278
3	0.900	0.732565
4	0.650	0.495468
...	...	...
4995	0.445	0.354732
4996	0.250	0.407335
4997	0.180	0.407335
4998	0.410	0.391855
4999	0.400	0.735450

```
Mean: 0.5574388313872948
Mean Absolute Error: 0.1501667405192572
Mean Squared Error: 0.05708234313545217
Root Mean Squared Error: 0.23891911421117434
```

The predicted values shown were then plotted against our actual values for 'sale_price' to generate the graph shown below.

The graphs actual results incline slowly with a large spike at near the end, in contrast to our predicted values which are slightly more erratic in comparison. We noticed several spikes above $2 million but most of our predictions remained around the $1 million territory.

The below graph was produced by running *pred_chart(y_test, y_pred.flatten(), sort = True)*:



Overall, the RMSE of this model was refined through lots of fine tuning and produced a final RMSE of 0.23 against a mean of 0.55 (roughly). Whilst this was in-line with our hypothesis, we would have liked to have seen a bigger difference in RMSE rates between our linear regression model and decision trees. Whilst it did produce results close to our SVR model, we felt as though this could have been improved further using random forest regression.

Random Forest Regression: We decided to use random forest as a model when we weren't satisfied with the results given from our decision trees model. Random forest is a supervised learning algorithm which can be used for both classification and regression. It works by using multiple decision trees and getting the most popular response out of all of them to provide an answer. (Chakure, 2019)

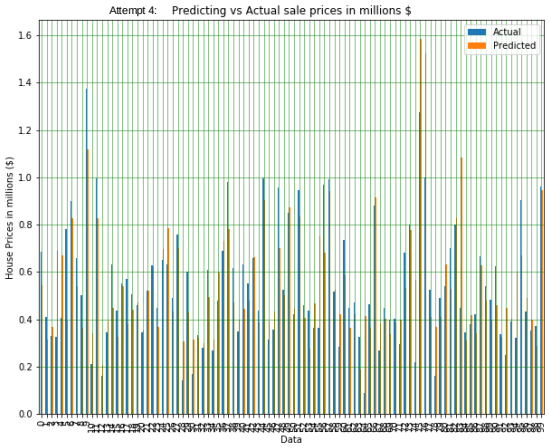
The advantage of this model is to get a better response since multiple decision trees are aggregated providing a wide range of data to learn from, since each tree is trained using a random section of the training data preventing overfitting. As such, we hypothesised that it would therefore yield better results than our decision tree model and ultimately the rest of our models too.

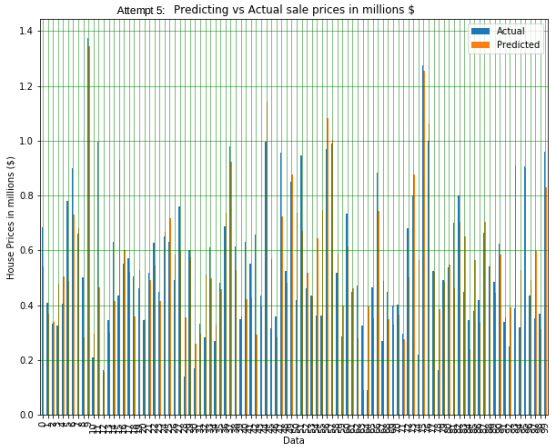
As with the other models, the training and testing split was done as 75% and 25% respectively. Different approaches were made to this regressor. Initially using 1 feature yielded a high RMSE value of 0.3 and low accuracy score resulting in poor predictions.

Attempt 4: rfr4 (All features)	Accuracy Score: 0.530065230459779 Mean: 0.5638754162436547 RMSE: 0.20992851934333695
Attempt 5: rfr5 (All features + PCA)	Accuracy Score: 0.24989870279261026 Mean: 0.5638754162436547 RMSE: 0.26522387229109046
Attempt 3: rfr3 (1 feature)	Accuracy Score: 0.0017996941915966456 Mean: 0.5638754162436547 RMSE: 0.3059577810357122
Attempt 2: rfr2 (1 feature)	Accuracy Score: 0.0017996941915966456 Mean: 0.5638754162436547 RMSE: 0.3059577810357122

Then we attempted using all the features to improve accuracy. The results were much improved with the best approach producing an RMSE of 0.21 and the highest score out of the other approaches. Removing an additional feature 'id' proved useful in increasing accuracy. (Koehrsen, 2018) PCA was attempted to get better results, but like with SVR, got worse results.

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download





As you can see, attempt 4 is the best result that could be obtained with a high accuracy score resulting in better predictions. The above charts are plotting the first 100 results of the actual result and what was predicted. For some results, a lower value is predicted more than a higher value being predicted.

The results would significantly improve when using a high tree count as we initially started with only 10 trees and moving up to 100. Going above 100 only improves the results marginally and increases the time to execute the model.

All models RMSE with a mean of 0.56:

- Random Forest: 0.2099
- Support Vector Machines: 0.2388
- Decision Trees: 0.2389
- Linear regression: 0.2713

Did we have any problems or difficulties working with our dataset?

One of the main issues was the inconsistent data being 0 values for sale price, year built and gross square feet. While $0 properties did represent that ownership of the building was transferred and wasn't sold, it wasn't relevant to our findings and skewed our data. There was quite a lot of these entries resulting in having a significant reduction to the dataset from 80000 to 60000 values approximately. Further filtering had to be done by only retaining data where sale price and gross square feet were under $10 million and 2000ft respectively to keep the dataset relevant to our goal. Land square feet was also removed as it was included in gross square feet.

The non-linear nature of our dataset was discovered upon attempting to use linear regression on it. It was assumed to be a good model to start off with but ended up being the worst one out of the 4. Most of the time was spent in data cleaning prior to beginning the models. Even when working on the models, we sometimes did further data cleaning which would have to be applied across all existing created models to retain consistency.

Conclusion: Lessons Learnt and Future Work

In conclusion, despite the complex nature of the dataset, we managed to get some good results from our models after lots of tuning and in-depth data cleaning methods.

Whilst our team initially believed linear regression would provide quite good results, we later realised that linear regression would inevitably be the least accurate of our models due to the nonlinearity of our dataset - which was discovered during when we began implementing our models.

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download

Our hypothesis that support vector machines using regression would yield the best results of our models was initially true. However, on discovery of our dataset being rather non-linear, we ended up also applying random forest regression which turned out to be the best model out of the four because of this. Our support vector regression only proved this further by having higher accuracy using the RBF kernel compared to the linear kernel.

Looking back, one of the lessons learnt would have been to fully complete our data cleaning before implementing our models, in comparison to incrementally cleaning data as soon as we faced problems during development. Not only is this good practice and a sensible choice, but would have saved us a lot of time spent debugging errors.

Throughout the course of this project, we feel we though we have learnt a lot about applying AI models to a very complex dataset and fine tuning it to yield good prediction results. In the future, the consideration of implementing a neural network would be useful as most regression models do not perfectly fit the data at hand. Applying a more complex model (such as a neural network) could provide much more predictive power compared to traditional regression and may have yielded the lowest RMSE of them all.

Overall, we feel we have gained a better understanding of our dataset, how to fine tune it to make it work better on our models, and how to apply different types of models and evaluate them based on their results.

References

Alejandro Baldominos, I. B. (2018, November 21). *Identifying Real estate opportunities using machine learning.* Retrieved from MDPI: https://arxiv.org/pdf/1809.04933.pdf

Arora, A. (2016, June 28). *Machine learning - How to increase the model accuracy of logistic regression in SciKit Python*. Retrieved from StackOverflow: https://stackoverflow.com/questions/38077190/how-to-increase-the-model-accuracy-of-logistic-regression-in-scikit-python

Chakure, A. (2019, June 29). *Random Forest Regression - Towards Data Science*. Retrieved from Towards data science: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f

City of New York. (2017). *NYC Property Sales | Kaggle*. Retrieved from Kaggle: https://www.kaggle.com/new-york-city/nyc-property-sales/data#

K, D. (2019, May 26). *Top 5 advantages and disadvantages of Decision Tree algorithm*. Retrieved from Medium: https://medium.com/@dhiraj8899/top-5-advantages-and-disadvantages-of-decision-tree-algorithm-428ebd199d9a

Koehrsen, W. (2018, Jan 7). *Improving the Random Forest in Python Part 1*. Retrieved from Towards data science: https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd

Mahendra, L. (2019, March). *Steam Reviews Dataset | Kaggle*. Retrieved from Kaggle: https://www.kaggle.com/luthfim/steam-reviews-dataset

NYC Housing. (n.d.). *Problem - NYC Housing Plan*. Retrieved from NYC Housing: https://www1.nyc.gov/site/housing/problem/problem.page

Srivastava, T. (2019, August 6). *Model Evaluation Metrics*. Retrieved from analyticsvidhya: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

Waskom, M. (2018). *seaborn.heatmap - seaborn 0.9.0 documentation*. Retrieved from seaborn: https://seaborn.pydata.org/generated/seaborn.heatmap.html

Our dataset can be found at: https://www.kaggle.com/new-york-city/nyc-property-sales/download