

1. Introduction - Motivation & Overview

Understanding the rental scene in Singapore is a tedious process not just for international students, but even for the local residents. Getting an idea of how much a house is worth (monthly rent) is a cumbersome process since multiple factors come into play. With agencies involved, this process may not be as transparent as one would expect. But if a curated dataset such as the one given in this course's project is available, the process would be more streamlined and help even someone without real estate-specific domain knowledge to arrive at a close estimate of how much rent would be reasonable for a given house. As a team, we also wanted to develop our skills in end-to-end data mining which would involve EDA of the dataset, modifying existing columns, creating new columns from other datasets, and also building and validating models. Hence, we took a comprehensive approach to EDA and model building in such a manner that every team member has a part in all aspects of the end-to-end flow of the project. We also wanted to implement the various data mining techniques and algorithms that were covered in the course syllabus. Algorithms such as KNN, Linear Regression, Polynomial Regression, XGboost, Random Forest Regressors, and Decision Tree Regressors were implemented along with a couple of algorithms that are not part of the course syllabus - Extra Trees Regressor and Neural Networks.

Our EDA was two-fold: EDA on the primary dataset and auxiliary datasets. The primary dataset involved a lot of data cleaning while the EDA on the auxiliary dataset was focused more on extracting new meaningful features. We also created a new dataset by ourselves using information from Google Maps (town_centroid_radius.csv). Apart from extracting information from auxiliary data such as distance to the closest MRT, school and mall, using the town_centroid_radius dataset we created two new features in the primary dataset - 'town_importance' and 'town_centrality_page_rank'.

Project GitHub Link: <https://github.com/ParasharaRamesh/Home-sweet-home>

2. EDA

2.1 Primary Dataset

a. Data cleaning - Cleaning of data is the first and most important step in data preparation, for ensuring good data quality, enabling accurate analysis, building effective Machine Learning models, ensuring consistency, and for data reproducibility. The [EDA steps that were carried out](#) are summarized below

- Removing the "elevation" column - We observe that all values under the "elevation" column are zero. This is not adding any useful information to our dataset. So we remove this column
- Removing the "furnished" column - We see that all values under the "furnished" column are "Yes". This is data redundancy. We remove this column to improve our dataset.
- Removing the "planning_area" column - We see that values under "planning_area" are usually the same as the values under the corresponding "town". There are about 2 percent of the records which have a value under "planning_area" that is different from the value under "town". Hence we remove this column.
- Removing the "block" column - The block information is not required as we have lat, long and town, region information which can be used to train the model. Having this granular of information would mean we have to perform one hot encoding which will make our input data size a lot bigger.
- Removing the "street_name" column - Similar to block, there are a lot of street names which cannot be encoded reliably therefore we are choosing to delete this column as we don't require such granular information in the dataset for training any of the models
- Removing the "subzone" column - Similar to block & street name we are choosing to delete subzone information as in most of the cases the subtown exactly matches that of the town name.

b. Converting all string values to lower - This converts all columns with type string into lowercase.

c. Converting the date to Unix Time Step - The rationale behind doing this is to ensure that we get a numerical value for the date. Since the date might also have something to do with the prediction of monthly rent.

d. Feature Extraction of 'Flat_Models' on Primary dataset:

The primary objective of the `get_ordinality_for_flat_model` function is to convert the categorical variable of flat models into a meaningful and interpretable ordinal representation. By assigning ordinal values based on specific criteria, the function transforms the qualitative nature of flat models into a quantitative form that enables more effective analysis and comparison. By creating a unified identifier for each combination and assigning ordinal values based on the average floor area, the function facilitates improved data structuring and analysis, enabling clearer insights about relative importance and significance of different flat models within the dataset.

The function first creates a 'combined_column' by merging values from the 'flat_type' and 'flat_model' columns. By merging the values from the 'flat_type' and 'flat_model' columns into a single 'combined_column', we create a comprehensive representation that combines the distinguishing features of both aspects. It then extracts unique values from the combined column. Iterating through these unique combinations, it calculates the minimum and maximum 'floor_area_sqm' for each combination. Then it computes the average floor area for each combination based upon the min and max calculated above, and sorts the dictionary

based on this average. Next it assigns ordinality to each combination based on their sort. Sorting is necessary to establish a clear and standardized ranking based on the average floor area, enabling a meaningful comparison and differentiation between the different combinations, while ordinality selection further helps in categorizing combinations based on their relative importance. Finally, it removes the 'combined_column' and 'flat_model' columns from the DataFrame and returns the modified DataFrame with the assigned ordinality for each unique combination.

2.2 Auxiliary Datasets

Getting data from auxiliary dataset - The auxiliary datasets provided for economic factors, location-based factors, and town importance scores are essential for enhancing the predictive power and better understanding of a house rental price prediction model. Once we get the values we add them as new columns in the original train dataset. The details of the auxiliary data set processing are added in next section. We approached the auxiliary dataset from two angles - [Economic Factors](#) and [Location-based factors](#).

a. Economic Factors

The datasets - 'sg-stock-prices', 'sg-coe-prices' were considered for the purpose of extracting economic factors.

COE Auxiliary dataset: In the context of Singapore cars, "COE" stands for "Certificate of Entitlement." It is a unique and significant component of Singapore's vehicle ownership and registration system. The COE is essentially a license that grants the holder the right to register, own, and use a vehicle in Singapore for a specific period, usually ten years. There is usually a bidding process where a car of a particular category is usually auctioned off.

'year-month' columns gives the year and month when a particular car was auctioned. The 'category' column gives the category of car (e.g. luxury, sport, trucks etc). There is some ordinality here but we are not considering it since we will look at the price instead. 'bidding' column gives the bidding round in which the car was auctioned. 'quota' is the number of bids accepted. 'bids' is the number of bids people raised in the interest of buying a particular car, and 'price' column is the final price the car was sold for.

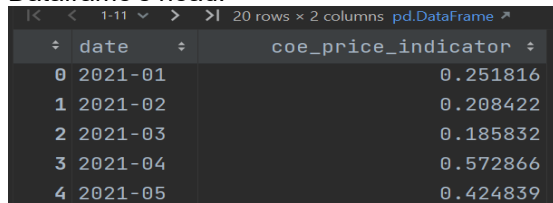
With this dataset, we can indirectly infer the current economic conditions as the more no of people who are ready to buy a car indicates how "well off" people are financially to even consider such an option. Based on the ratio of the number of bids to the existing quota it shows the "interest" of how contested/popular a particular car is.

We came up with a "**coe-indicator**" score which is a number which represents the state of the economy for a particular month. We essentially do **(bids/quota) * price**.

which means out of the existing quota, how many bids came for a particular car category multiplied with the price. This ratio can be both >1 or <1 as that is an indication of how "interested" people are to buy this particular car.

We find this indicator and then do min-max normalization.

Dataframe's head:



	date	coe_price_indicator
0	2021-01	0.251816
1	2021-02	0.208422
2	2021-03	0.185832
3	2021-04	0.572866
4	2021-05	0.424839

And then we are finding **cumulative month-wise** statistics:

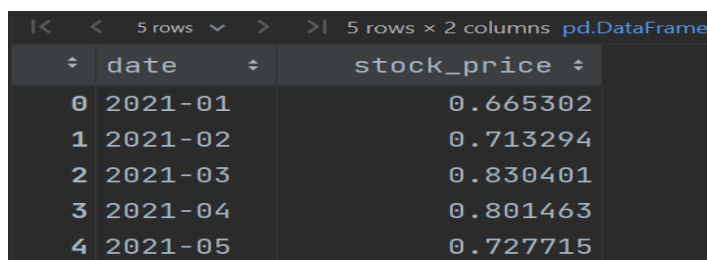
1. We first perform grouping across date & category and take the mean of the coe-indicator.
2. Using this data frame, we find out the total coe_price across all categories.

Eventually, this will give us some number which is indicative of the economic condition for every month across all categories of cars. Since there are a few months which don't have any coe-indicator value, we basically do **imputation by finding the mean for that year** and then assign those values.

Stock Auxiliary dataset: The 'sg-stock-prices' dataset has the trading information for different tickers on different dates along with its opening & closing values. 'name' column is the Company name. 'symbol' is the Company ticket, 'date' is the exact date, 'open' is the opening price, 'high' is the highest price in a day, 'low' is the lowest price in a day, 'close' is the closing price and 'adjusted_close' is a financial metric applied on the closing value.

Since we have daily trading prices for each company, we can first find out the **monthly average trading price for each company**. Using this we can take an average of the trading values across all companies to get one "score" which is a number indicating how well all companies performed in a particular month. **This score is indirectly indicative of how the economy is performing** which will help in house price prediction. Similar to the coe-indicator we will come up with a **stock-indicator** which represents the state of the economy. The final output will also have month, stock_indicator_score value.

Dataframe's head:



	date	stock_price
0	2021-01	0.665302
1	2021-02	0.713294
2	2021-03	0.830401
3	2021-04	0.801463
4	2021-05	0.727715

Since we have daily trading prices for each company, we can first find out the **monthly average trading price for each company**. Using this we can take an average of the trading values across all companies to get one "score" which is a number indicating how well all companies performed in a particular month. **This score is indirectly indicative of how the economy is performing** which will help in house price prediction. Similar to the coe-indicator we will come up with a **stock-indicator** which represents the state of the economy. The final output will also have month, stock_indicator_score value.

Grouping by company name and date is done to find the average monthly closing prices for all companies. Using this we take a mean by grouping on month to find out the "economic" state for a particular month across all companies. Since we have the month and its corresponding indicator values (both coe & stock) we will use these values and add 2 new columns in the original dataset where we use each row's particular month to fill the value indicator values.

b. Location (Distance from entities) Factors:

We have the following information - existing mrts and their latitude & longitudes, planned mrts and their latitude & longitudes, primary schools and their latitude & longitudes, shopping malls and their latitude & longitudes in the datasets. It is fair to assume that the prices also get influenced by the number of such locations in its vicinity. For e.g. a house next to a mall or right next to an MRT station could potentially be pricy for a house.

Since we also have the latitude and longitude for each house, we can find out the distance to the nearest existing/planned MRT, school and shopping mall and add those distances in meters as new columns to our dataset. Our regression models should then have enough context to potentially learn some unseen rules from these newly added columns as well.

To find distances given a pair of latitudes and longitudes, there were mainly two options which immediately came to our mind- using Google Maps API to find out the walking distance to each point, using an approximation of "**haversine distance**" to compute the distance along the surface area of the Earth. The former was not feasible given the API prices therefore we are going for haversine distance computation.

The haversine distance is a formula used to calculate the distance between two points on the surface of a sphere, such as the Earth. It's commonly used in geospatial applications, particularly for calculating distances between latitude and longitude coordinates. In the real-world there may not be a straight-line path between two points. We would have go through different street routes, bridges and other man-made terrains. Although haversine distance between two points may not be the actual distance in the real-world route, haversine did act as a decent supplement for Google Maps API distance. More about haversine distance can be found in the appendix.

The logic to compute the **distance to the nearest existing MRT, distance to the nearest school and distance to the nearest mall** for each house was similar - computing pair-wise distance between each MRT and the house and find the nearest MRT. Similarly, we found the nearest school and the nearest shopping mall.

For each house, while computing the **distances to the nearest planned MRT**, we took a slightly different approach. Since none of these MRTs exist currently they should not be given the same weightage as existing MRTs.

Hence, we defined a **damping factor called "alpha"** which accounts for the expected future distance to the planned MRT while also considering the years of construction left to inaugurate the MRT.

The further in future a particular planned MRT is to be opened, the distance to that MRT is also increased accordingly by considering the wait time. This way the values here would represent some kind of "future reward" that the model can learn.

Hence four new columns were created in the original dataset - '*distance_to_nearest_existing_mrt*'.

'*distance_to_nearest_planned_mrt*', '*distance_to_nearest_school*', '*distance_to_nearest_mall*'

$$distance_to_planned_mrt = estimated_real_world_distance + 100e^{years_left_for_opening}$$

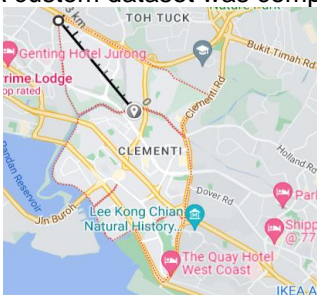
2.3 Determining Importance Score and Centrality for Towns

A key factor that would decide the rent of a home would be the area or town. Hence, we wanted to quantify the "**importance**" of an area. We drew inspiration from this course's **lecture 10: Graph Mining**. The idea of Centrality would suit our goal of assigning "importance score" to towns.

But apart from assigning "importance score" to towns, we also wanted to factor in "how close this town (which might have an importance score that is low or high) is to a town that is of high importance. **The town by itself may not be a hotspot, but if it's just a kilometre away from a very important area, then it should still receive some form of reward.**

Being half a kilometre away from Marina Bay would have more weightage than being half a kilometre away from a relatively not-so-important area.

A custom dataset was compiled for this task - [The town centroid radius.csv dataset](#).



The given original train dataset had 26 unique towns in total.

For each of these 26 towns, the town centroid was found using google maps. The centroid_lat and centroid_long were found along with the approximate town radius and the 'town_centroid_radius.csv' dataset was created.

Images for other towns can be found in

['resources/useful_pics/town_centroid_radius_pictures.pdf'](#) document.

The town_centroid_radius.csv dataset that we compiled:

A	B	C	D
town	centroid_lat	centroid_long	town_radius_km
jurong west	1.341443	103.703521	3.36
tampines	1.345962	103.951249	2.87
sengkang	1.39311	103.883115	2.8
bedok	1.325809	103.933736	3.46
ang mo kio	1.382177	103.841626	2.79
yishun	1.417996	103.840275	2.88
bukit merah	1.282343	103.821103	2.69
woodlands	1.439668	103.788821	2.46
hougang	1.366029	103.890594	2.89
punggol	1.405524	103.909898	2.55
toa payoh	1.337213	103.86311	2.92
clementi	1.323533	103.762224	2.43
bukit batok	1.353846	103.754385	2.44
choa chu kang	1.39109	103.745614	2.07
queenstown	1.295994	103.791323	3.29
kallang/whampoa	1.313183	103.864736	2.28
pasir ris	1.375153	103.947255	3.68
geylang	1.322243	103.890282	2.07
jurong east	1.32525	103.735567	2.19
bukit panjang	1.37097	103.772176	2.65
sembawang	1.454814	103.818557	3.16
bishan	1.35512	103.837545	1.91
serangoon	1.370309	103.867557	2.59
marine parade	1.302132	103.896979	2.7
central	1.295888	103.843593	2.41
bukit timah	1.331353	103.790679	3.45

The newly created dataset was used to find the importance and centrality of towns.

The importance of a town was determined the following manner:

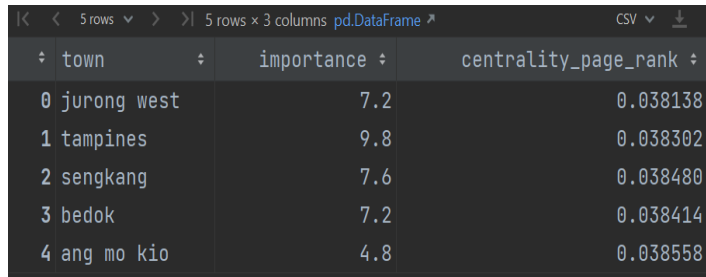
$$\text{importance} = (\text{no. of MRTs existing within radius} * 0.4) + (\text{no. of MRTs planned within radius} * 0.2) + (\text{no. of schools within radius} * 0.2) + (\text{no. of malls within radius} * 0.2)$$

To determine what would be a reasonable radius for all towns, we took the mean of the town radius column which was 2.730 KM. Hence, we considered 2.7km or 2700m as the radius. If an entity such as MRT or a school is within this distance ([Haversine distance](#)) from the town's centre, then we added that entity to the town's count.

We wanted to provide higher weightage to existing MRTs, hence in the formula the weightage for no. of existing MRTs would be 0.4, while it would be 0.2 for planned MRTs, schools and malls.

To find the page rank of towns, a simplified implementation of page rank was used. The importance scores would be the score obtained from the above formula. The distance matrix (instead of adjacency matrix) was the pair-wise haversine distance between towns. These two columns – [importance](#) and [centrality_page_rank](#) were then merged with the original dataset based on the town name. The merging of these two columns into the original dataset was done along other columns that were extracted from auxiliary data such as 'stock_price', 'coe_prices', 'distance_to_nearest_existing_mrt' and so on.

Dataframe's head:



	town	importance	centrality_page_rank
0	jurong west	7.2	0.038138
1	tampines	9.8	0.038302
2	sengkang	7.6	0.038480
3	bedok	7.2	0.038414
4	ang mo kio	4.8	0.038558

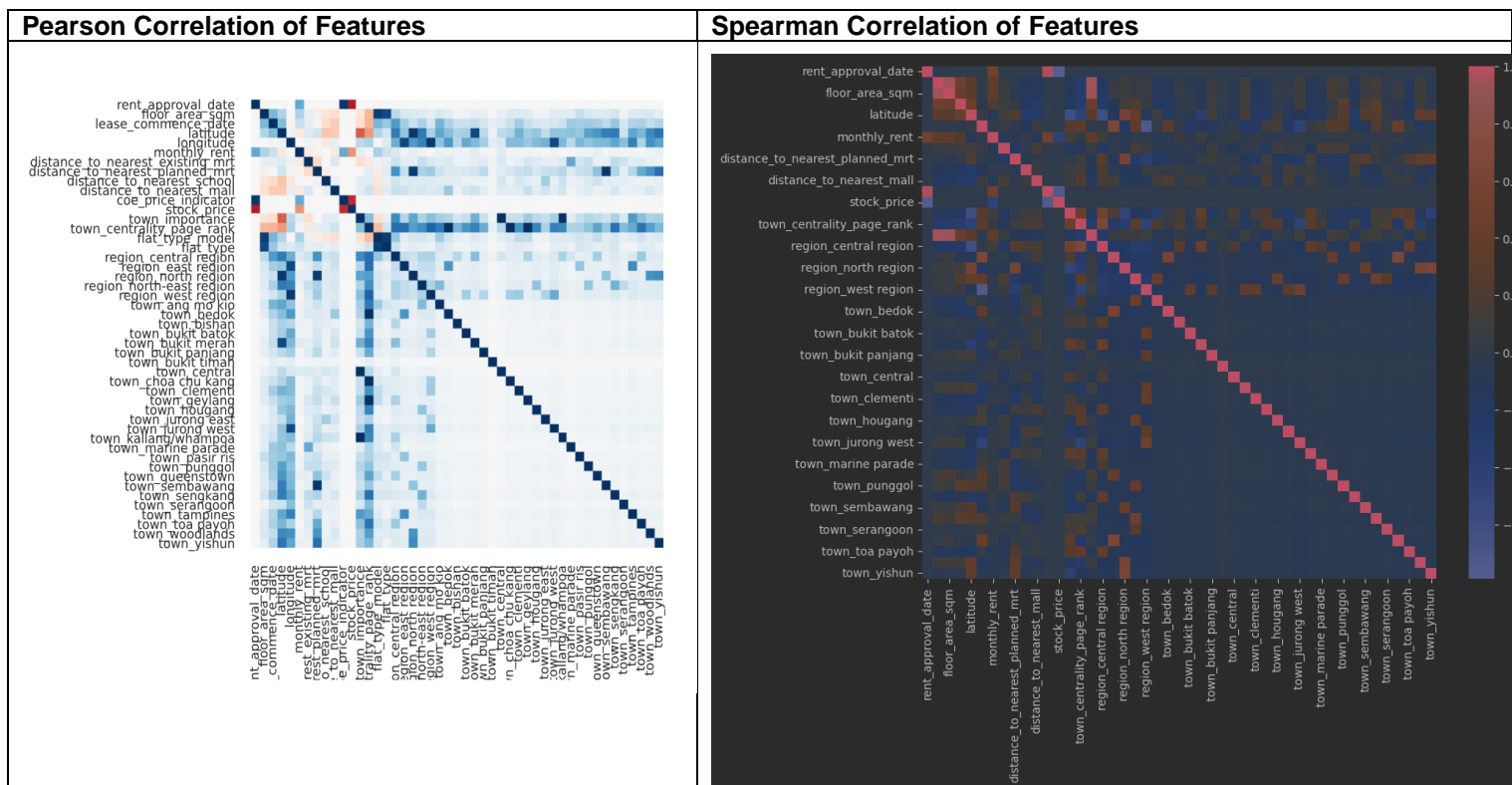
2.4) Removing exact duplicates

We observe that there are 523 exact duplicates in the dataset at this stage. Hence we remove the exact duplicates, which results in 59477 rows. We carry on our further processing with these rows.

2.5) Normalize columns

Normalization is done on the following columns: 'rent_approval_date' - as it is now a unix timestamp. 'lease_commense_date' - as it is just an year value. 'floor_area_sqm', 'coe_price_indicator' (from auxiliary dataset), 'stock_price' (from auxiliary dataset) and all the distance values computed to mrt, school and malls.

The pair-wise correlation coefficients between features in the dataset after all EDA step indicate a good degree of correlation.

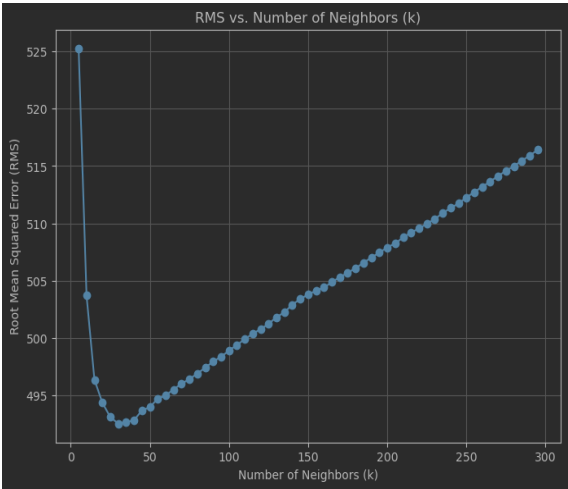


3. Models and Results

For all models, GridSearchCV was used to find the best hyperparameters (except for Neural Networks, where a trial-and-error experimentation approach was taken). Detailed implementation has been documented in [the respective notebooks](#). The RMSE values mentioned for each model is obtained from validation set (train-val: 80-20 split)

3.1) KNN

We selected a simple baseline model and considered integrating real-world distances using latitude and longitude features. Initially, we experimented with Vanilla KNN, identifying the optimal K value of 30 (using grid search) with the Manhattan distance metric, resulting in an RMSE of approximately 491.73.



To leverage latitude and longitude data, we implemented a variation of KNN which includes haversine distance for these coordinates and Euclidean distance for other features. Surprisingly, this approach produced a higher RMSE of 507.45 compared to Vanilla KNN.

Subsequently, we combined haversine distance, Euclidean distance, and Manhattan distance to enhance generalization by taking different distances across different latent spaces as a combined distance, achieving an RMSE of 496.34, which was a minor improvement over the previous approach but still below Vanilla KNN's performance.

Due to the uncertainty regarding the normalizing factor for haversine distances, we decided to delay further pursuit of this approach, despite its promising potential.

3.2) Regression

a. Linear Regression

Linear Regression is a fundamental supervised machine learning algorithm that computes the linear relationship between the dependent variable(in our case house rental price) and one or more independent features.The objective of this analysis is to find the best hyperparameters for a Linear Regression model and evaluate its performance on validation data. This is achieved through a grid search and the calculation of the Root Mean Square Error. Grid search CV is created with the following parameters :

```
{'fit_intercept': [True, False], //Determines whether to calculate the bias term in the model
'positive': [True, False], //Constraint on the coefficients of the model required them to be positive
'copy_X': [True], //Controls whether copy of input data X is made before fitting the model
'n_jobs': [None, -1]} //Specifies the number of CPU cores to use while fitting
```

The grid search identified the best hyper parameters as {'copy_X': True, 'fit_intercept': False, 'n_jobs': None, 'positive': False}

However, we achieved RMSE of: 502.1382774955841

Here we observe that though linear regression can be useful for predicting house rental values, its success depends upon the linearity of the relationship between the features and rental prices.

b. Polynomial Regression

The objective of this analysis is to employ Polynomial Regression to model and analyze data. Polynomial Regression is an extension of linear regression that can capture nonlinear relationships between features and the target variables. Here we have selected a polynomial with degree 2, 'PolynomialFeatures' class is used to expand the feature set. By creating polynomial combinations of the original features. A Linear Regression model is selected for the analysis. The usefulness of polynomial Regression lies in the fact that it allows for modeling of complex data patterns that may not be adequately represented by a simple linear model. The results obtained are as below :

RMSE train = 484.9438816351702

RMSE val = 484.0383976317791

Here we observe that the Polynomial Regression model gives lower loss values among all other experiments conducted till now.

3.3) SVM

SVM is a type of supervised learning algorithm which can be used for both regression and classification. Here we make use of SVR(Support Vector Regressor) for the regression analysis, which finds a function that approximates the relationship between the input and target variable by minimizing the error. There are many factors that affect the house rental prediction. The motivation for using an SVR is that it chooses different combinations as parameters. Grid Search CV was created with the hyperparameters

```
param_grid = {
    'model__C': [0.1, 1, 10, 100], # Regularization parameter
    'model__kernel': ['rbf'], # Kernel type
    'model__gamma': ['scale', 0.1, 1] # Kernel coefficient for 'rbf' kernel
}
```

The grid search identified the best hyper parameters as :

```
{'model__C': 100, 'model__gamma': 'scale', 'model__kernel': 'rbf'}
```

Using these we achieved RMSE of : 490.4634532324051

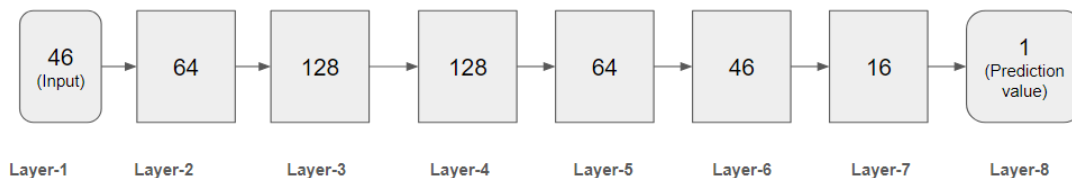
Though SVM mainly performs well on classification tasks, its decent performance here can be attributed to the fact that SVM works well with the complex datasets when the right choice of kernel is made.

3.4) Neural Networks

We have tried 5 different architectures-hyperparameter combinations in total. We train on the 'train_clean.csv', which the train dataset obtained after processing all the EDA steps mentioned in the 'EDA.ipynb'.

Since neural networks were not our primary focus, in our report we have provided details only about the architecture which gave us the best results. Architecture details about the other 4 models can be found in the Neural Network ipynb notebook.

Best Model – Architecture Model-5:



Batch Size = 32, Optimizer = Adam,
Learning Rate = 0.001, Epochs = 200

All Models and RMSE on val set:

Model-1: 495.3260

Model-2: 492.2489

Model-3: 493.9776

Model-4: 491.8362

```
model = Sequential()
model.add(layers.Input(shape=(46,)))
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(46, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dense(1))
```

The volume of data available is not enough for the network to learn effectively and hence the performance is not as good as traditional regressors and tree-based regressors.

3.5) Tree-based Regressors

a) Decision Trees

The Decision Tree algorithm is a fundamental tool for predictive modeling in machine learning. It partitions the data into subsets based on the values of individual features and makes predictions accordingly. The simplicity and interpretability of Decision Trees make them particularly valuable for understanding the underlying patterns and relationships within the data.

This analysis aims to identify the optimal hyperparameters for an Decision trees model and evaluate its performance on the validation dataset. The code employs GridSearchCV to explore a range of parameters.

```
param_grid_dt = {
    'max_depth': list(range(0, 20, 1)),
    'criterion': ['mse', 'friedman_mse', 'mae'],
    'splitter': ['best', 'random'],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

The param_grid_dt variable contains a set of hyperparameters such as max_depth, criterion, splitter, min_samples_split, and min_samples_leaf.

The max_depth parameter determines the maximum depth of the tree, while criterion defines the function to measure the quality of a split.

Additionally, splitter indicates the strategy for choosing the split at each node, and min_samples_split and min_samples_leaf specify the minimum number of samples required to split an internal node and to be at a leaf node, respectively.

After conducting the Grid Search, we evaluate the performance of the Decision Tree model to draw meaningful conclusions. RMSE of best decision tree model: 500.4649

b) Random Forest Regressor

Random Forest is a robust and versatile machine learning algorithm widely used for regression and classification tasks. It's an ensemble learning method that constructs a multitude of decision trees during training and outputs the mean prediction of the individual trees.

The goal of this analysis is to determine the optimal hyperparameters for a Random Forest Regressor model and assess its performance on the validation dataset. By employing GridSearchCV, the code explores the following parameters:

```
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': list(range(7, 12, 1)),
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

The 'n_estimators' parameter represents the number of trees in the forest, while 'max_depth' defines the maximum depth of the tree. Furthermore, 'min_samples_split' specifies the minimum number of samples required to split an internal node, and 'min_samples_leaf' sets the minimum number of samples required to be at a leaf node.

The identified best hyperparameters are used to make predictions, and the Root Mean Square Error (RMSE) is computed as a measure of the model's accuracy. RMSE of best decision tree model: 486.9367

c) XG Boost Regressor

This Algorithm gave the best results on the dataset for us. Here, trees are built in parallel, instead of sequentially like gradient boosted trees. It follows a level-wise strategy, scanning across gradient values by using these partial sums to evaluate the quality of splits at every possible split in the training set. Therefore, it is most preferred for regression and classification tasks, especially in large-scale data applications. The code employs GridSearchCV to explore a range of parameters, including 'max_depth', 'learning_rate', 'gamma', 'subsample', 'random_state', and 'colsample_bytree'.

```
param_grid_xgb1 = {
    'max_depth': list(range(0, 12, 1)),
    'learning_rate': [0.1, 0.01, 0.001],
    'gamma': [0, 0.1, 0.3],
    'subsample': [0.8, 1.0],
    'random_state': [42],
    'colsample_bytree': [0.3]
}
```

The 'max_depth' parameter controls the maximum depth of a tree, while the 'learning_rate' parameter determines the step size at each iteration during the optimization process. 'Gamma' represents the minimum loss reduction required to make a further partition on a leaf node, and 'subsample' defines the fraction of samples used for training trees.

Additionally, 'random_state' ensures reproducibility, and 'colsample_bytree' sets the fraction of features to be randomly sampled for each tree. XGBoost outperforms other algorithms due to its implementation of gradient boosting, which uses multiple weak learners to make strong predictions. RMSE of best decision tree model: 482.9581

d) Extra Tree Regressor

Extra Trees(Extremely Randomized Trees) is an ensemble Machine Learning algorithm that combines the predictions of multiple decision trees. Feature importance being a valuable aspect of Extra Tree Regressor is the primary motivation for using this model. Gini index is used as a metric for calculating the feature importance. For the multiple decision trees that are used, Extra Tree algorithm aggregates the Gini importance across all the trees in the ensemble. It adds up the importance score for each feature to determine its overall importance. Grid Search CV was created with the hyperparameters:

```
'n_estimators': [100, 200], //The number of decision trees in the ensemble
'max_depth': [10, 15, 20], //The maximum depth of each decision tree in the ensemble
'min_samples_split': [2, 5, 10], //The minimum number of samples required to split an internal node
during tree construction
'min_samples_leaf': [1, 2, 4], //The minimum number of samples required to be in a leaf node
'max_features': ['sqrt'], //The number of features to consider when looking for the best split
```

The grid search identified the best hyperparameters as :

```
{'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200}
```

The performance of this model is not as good as Polynomial Regression or other ensemble methods like AdaBoost, which can be attributed to the fact that Extra trees Regressor essentially requires more data to effectively generalize complex relationships. RMSE value : 486.9619342942999.

3.6) Model Performance Comparison

	Algorithm	Best Hyperparameter	Validation & Kaggle Test Score (RMSE)
1	KNN Vanilla	'algorithm': 'ball_tree', 'leaf_size': 40, 'n_neighbors': 30, 'p': 1, 'weights': 'uniform'	Validation Data : 491.73 Kaggle Test Data : 491.92
2	KNN Custom Haversine	Same hyperparameters as vanilla KNN	Validation Data : 496.34 Kaggle Test Data : 496.70
3	Linear Regression	'copy_X': True, 'fit_intercept': False, 'n_jobs': None, 'positive': False	Validation Data : 502.14 Kaggle Test Data : 501.98
4	Polynomial Regression	-	Validation Data : 484.04 Kaggle Test Data : 484.97
5	SVM	'model_C': 100, 'modelgamma': 'scale', 'model_kernel': 'rbf'	Validation Data : 490.46 Kaggle Test Data : 489.25
6	Neural Networks	Batch Size = 32, Optimizer = Adam, Learning Rate = 0.001, Epochs = 200 Architecture : 46->64->128->128->64->46->16->1	Validation Data : 483.61 Kaggle Test Data : 484.17
7	Decision Tree Regressor	'criterion': 'friedman_mse', 'max_depth': 8, 'min_samples_leaf': 4, 'min_samples_split': 10, 'splitter': 'best'	Validation Data : 503.30 Kaggle Test Data : 500.40
8	Random Forest Regressor	'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 200	Validation Data : 486.32 Kaggle Test Data : 486.93
9	XGBoost	'colsample_bytree': 0.3, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 6, 'random_state': 42, 'subsample': 1.0	Validation Data : 480.65 Kaggle Test Data : 482.95
10	Extra Trees Regressor	'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200	Validation Data : 486.96 Kaggle Test Data : 486.37

4. Conclusion

The best performance was observed in the XGboost model, closely followed by Polynomial Regression. The EDA performed on the primary dataset and the features extracted from auxiliary dataset proved to be helpful. During EDA, the primary focus was to arrive at a fair trade-off between retaining a column, hence increasing the feature set size and dropping a column and reducing the feature set complexity. We had to look at the data from different levels of granularity - region-level, town-level, street-level and take a call on retaining/dropping the feature.

While columns such as 'street_name' were dropped, the columns 'town', 'region' were encoded. The models could only be as good as the data, hence a considerable portion of this project focused on proper EDA and feature engineering. Since we wanted to try different flavors of data mining techniques and models, a wide range of algorithms were selected. Out of the Tree-based models such as Random Forest Regressor, XGBoost, Decision Tree Regressor and Extra Trees Regressor, only XGboost provided good results. Given the nature of data, Polynomial Regression and SVM performed better than Linear Regression. The dataset seemed too complex for KNN (vanilla and custom-implementation), as the model struggled to map the distances between features across high dimensions. The size of the training data proved not to be enough for a Neural Network to train and extract patterns efficiently. A key take away was that complex models are not a guarantee for better results.

There is also room for improvement in multiple avenues of the project. One would be using Google Maps API distance instead of Haversine distance. Secondly, while our models did provide good results, the potential of the dataset could be exploited more by combining it with more additional data such as nearby hospitals, attraction spots. Thirdly, at the end of the day, the importance of a feature of a house would be subjective to the customer or the individual searching for a home. Having a primary school right next door might be of immense importance to a family with a four-year old. But this may not be important for a bachelor looking for a home. Similarly, NUS students would be ready to pay extra for accommodations that are 10 minutes from campus. But proximity to NUS would not be of any importance for a working professional. The models could be improved even more by adding another layer of pipeline in the end-to-end process - user's preference and profile. A dynamic weight distribution model that could assign importance to features based on the user's preference would provide more accurate results from the user's perspective.

5. Appendix

Team Member Contributions

Team Member	Work
Adithya Ragothaman E1124526	EDA of primary 'train' dataset Feature extraction of ordinal 'flat_model' Implementation of Decision Tree Regressor Implementation of Random Forest Regressor Implementation of XGboost
Parashara Ramesh E1216292	EDA of ' <i>sg-stock-prices</i> ' to extract new feature 'stock_price' EDA of ' <i>sg-coe-prices</i> ' to extract new feature 'coe_price' Consolidated the EDA of primary and auxiliary dataset Implementation of Vanilla KNN Implementation of Custom Haversine KNN
Poornima Sridhara E1124722	EDA of primary 'train' dataset Implementation of SVM Implementation of Linear Regression Implementation of Polynomial Linear Regression Implementation of Extra Trees Regressor
Sriram Srikanth E1132261	EDA of ' <i>sg-mrt-existing-stations</i> ' to extract new feature ' <i>distance_to_nearest_existing_mrt</i> '. EDA of ' <i>sg-mrt-planned-stations</i> ', to extract new ' <i>distance_to_nearest_planned_mrt</i> '. EDA of ' <i>sg-primary-schools</i> ' to extract new ' <i>distance_to_nearest_school</i> '. EDA of ' <i>sg-shopping-malls</i> ' to extract new feature ' <i>distance_to_nearest_mall</i> ' Creation of new dataset ' <i>town_centroid_radius</i> ' to extract new columns ' <i>town_importance</i> ', ' <i>town_centrality_page_rank</i> '. Implementation of model-1, model-2, model-3, model-4, model-5 in neural networks

Haversine Distance

```
def haversine_distance(lat1, lon1, lat2, lon2):  
    # Radius of the Earth in kilometers  
    earth_radius_km = 6371.0  
  
    # Convert latitude and longitude from degrees to radians  
    lat1_rad = math.radians(lat1)  
    lon1_rad = math.radians(lon1)  
    lat2_rad = math.radians(lat2)  
    lon2_rad = math.radians(lon2)  
  
    # Haversine formula  
    dlon = lon2_rad - lon1_rad  
    dlat = lat2_rad - lat1_rad  
    a = math.sin(dlat / 2) ** 2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2) ** 2  
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))  
  
    # Calculate the distance  
    distance = earth_radius_km * c  
  
    #returns distance in meters  
    return distance * 1000
```

This is how the haversine distance was computed.

```
lat1,lon1 = 1.333084, 103.750254  
lat2,lon2 = 1.3307673380970195, 103.76720560712226  
  
#1.9km  
print(haversine_distance(lat1, lon1, lat2, lon2))  
  
Executed at 2023.11.04 19:55:19 in 18ms  
  
1901.9489863160638
```

As we can see from google maps, the distance provided by the haversine formula matches with google maps.

