Location, Location, Location?

CS5228 Course Project
Group Name on Kaggle: Group 05

Team 5

Chen Yixun	Liyanarachchi Lekamlage Rashini Kavindya Liyanarachchi	Saisha Ajay Chhabria	Niharika Shrivastava
A0245243L E0894590	A0254386W E0954787	A0244449X E0863008	A0254355A E0954756

Table of Content

1. Introduction

Motivation

Dataset

The starting dataset comprises of a set of approximately 430,000 sale events and details in the training dataset, and 100,000 in the test dataset for prediction. A set of 7 auxiliary datasets were also made available. We first began by pre-processing the training, testing, and auxiliary datasets to ensure they were clean and standardized. We then post processed and merged the auxiliary datasets with the training/testing datasets, while deriving certain features along the way.

Cleaning

Standard cleaning measures were first taken to ensure the data adhered to a standardized format, such as capitalization in street name, inconsistent categorical naming (4 room vs 4-room), standardization of date formats etc. Several categories were also reformatted for easier analysis, such as transforming the categorical storey_range (e.g., 01 to 03) into a numerical storey value (the median of the range). The category eco_category was dropped as all values were labeled as uncategorized, and cursory research did not show anything significant relating to eco categories for HDB flats.

One field that was not straight forward to transform was elevation – the original dataset had all rows filled with a 0.0 value. While elevation is not something that immediately comes to mind when one considers a given HDB apartment, we decided to impute this value in case it turns out to be significant. Given that coordinates for each resale event were already provided, an elevation lookup should not be too complex. One approach is to simply query a third-party mapping API for the elevation data for every row of data – however, since we have over 500,000 rows of data, this approach is unfeasible. **Instead, we chose to source and download a Digital Elevation Model (DEM) height map of Singapore.** The python package elevation makes this a straightforward task. With this local copy of the height map, querying elevation data for a given latitude/longitude coordinate becomes an easy task, and we then imputed all elevation values in the train/test dataset.

Augmentation

The next step in data preparation is to explore the auxiliary data provided and merge them into the train/test datasets, deriving new features, as necessary. The idea behind transforming points of interest (commercial centers, hawker centers, schools, malls) into features per sale event was simple, simply calculating the distance in kilometers to the nearest given point of interest type. However, the actual execution proved slightly challenging. A naïve brute force approach (for a given row, calculate the distances to all points of interest of a given type and take the minimum) proved unfeasible, taking upwards of 15-30 minutes per point type, even utilizing NumPy's vectorization (O(n^2) complexity). This actual calculation can be distilled as a k-Nearest-Neighbours problem, where k = 1, as we are only interested in a single point of interest. Given the nature of distance calculation using latitude/longitude coordinates, we opted to use a BallTree to perform our kNN calculations, as it natively supports Haversine distance for calculating geographical distances. In addition, it has a fast average query speed of O(klogN) for k=NN, or simply O(logN) for 1-NN. This was a vast improvement over the original approach, and calculating the nearest point of interest for each sale event for each data type did not take any significant time.

One point-of-interest augmentation worth mentioning was the nearest MRT/LRT station feature. A simple approach would simply be to run the same aforementioned distance calculation on the provided MRT

dataset. However, this would be an incorrect approach, as when a given sale happened, a given MRT in the vicinity may not have been operational. Thus, we needed to carefully query by date ranges and only calculate the nearest MRT from a subset of valid MRTS for a given sale data.

Before going into the distance by operational date calculation, we noticed that a large number of MRTs had missing dates. By sourcing an additional dataset online from Kaggle and Wikipedia, we were able to impute most of the missing opening dates into the MRT dataset.

For the actual distance calculation, once again, brute forcing (for each row, calculate the subset of valid MRTs, fit the BallTree on this subset, then perform the 1-NN calculation) proved unfeasible. Instead, we query by “date buckets” instead – for all unique opening dates in the MRT dataset, in descending order, query all sales that happened between the start of the current bucket and before the end of the previous bucket, where the set of valid MRTs are all MRTs that were operational before or equal to the start date of the current bucket. This allows us to fit a single BallTree for multiple distance calculations, instead of fitting one BallTree for every calculation. With some clever index masking, this calculation by buckets once again did not take any significant computation time.

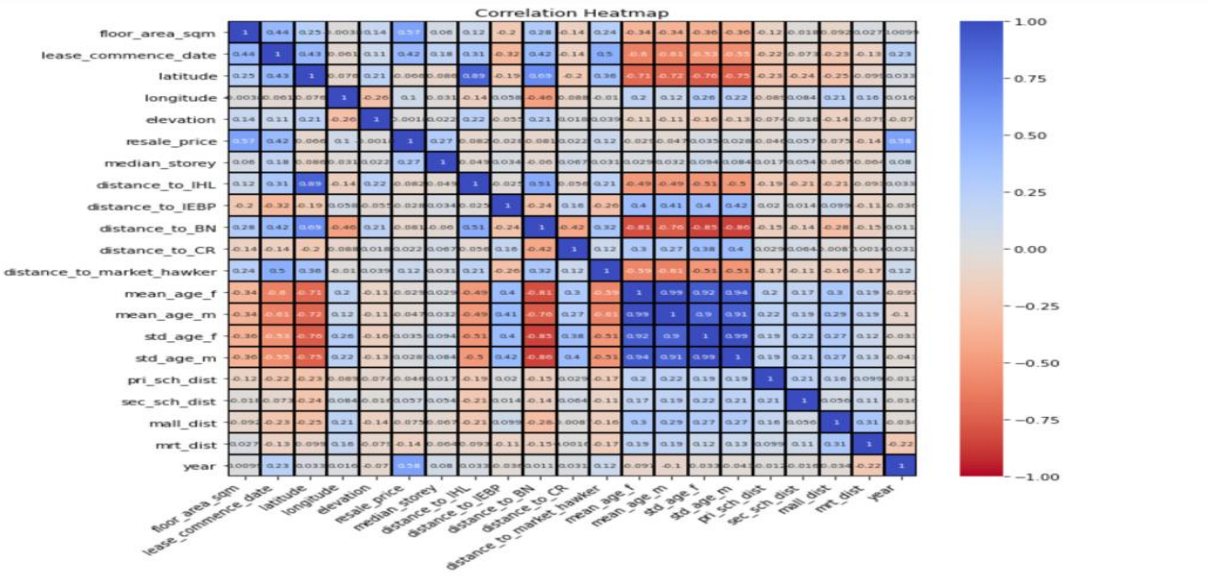
Lastly, the populations demographic auxiliary dataset also required some careful engineering. This dataset was laid out by the count of age groups per sex, per subzone, per planning area. In order to better summarize this dataset, we opted to instead take the mean plus standard deviation per sex per planning area, generating four new features (mean_age_m, mean_age_f, std_age_m, std_age_f) and then joining these into the original dataset on planning_area. We opted to use mean instead of median as the demographic dataset was not very granular. We were unable to use the more granular subzone as there was missing population data for the city hall subzone.

2. Exploratory Data Analysis and Preprocessing

Data Correlation Analysis

As described in the previous section, the original dataset was first cleaned and then was extended by adding auxiliary data provided. A simple correlation analysis was done on these two datasets; cleaned and augmented.

Figure 01 shows the correlation heatmap for the augmented dataset. We can see that distance to an MRT, distance to a mall, and distance to commercial centers have a weak link with resale price by plotting the linear correlations and calculating the correlation coefficients between "resale price" and other continuous input variables. As a result, we can disregard such inputs when training our models.

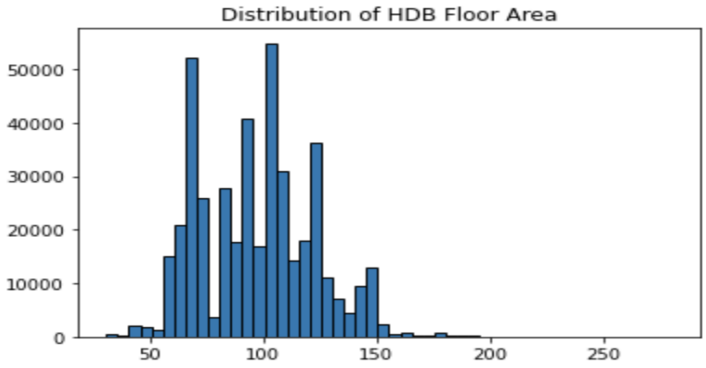


As seen from the correlation heatmap, there are some inputs that have quite a strong correlation with the resale price. Hence, we thought that it would be better to get an understanding of this relationship in a more detailed manner.

Floor Area

Distribution of Flats by Floor area

One of the inputs that have a strong correlation with the resale price is the floor area. We see most flats are within the floor area between 50 – 150 Square meters.



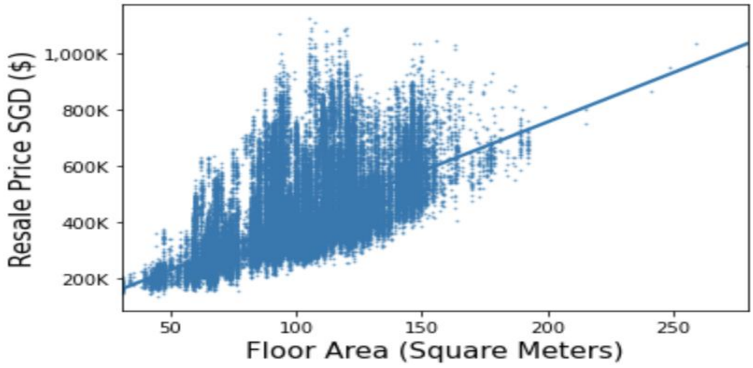
However, there are several outliers as below. The following are the number of flats with the floor area greater than 200 square meters.

terrace	28
premium maisonette	6
maisonette	5
apartment	3
adjoined flat	1

However, these outliers in the floor area are special HDBs (Housing & Development Board) that are larger than ordinary ones. As a result, from a multivariate standpoint, they may not be outliers.

Resale price vs Floor Area Distribution.

When comparing the resale price with the floor area of the flats, we can simply see the correlation between them from the plot below. Although there are several outliers, when comparing the means, it is clear that the resale price increases with the floor area.



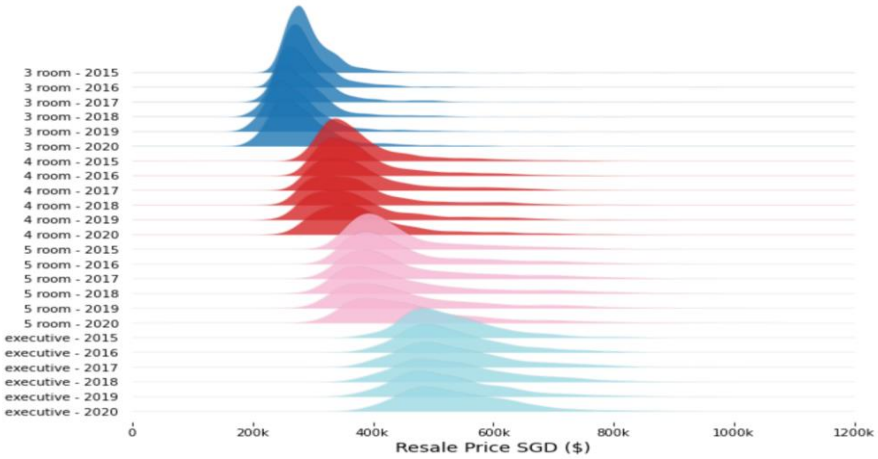
Flat Type

Proportion of Flat Types in the dataset

It is obvious that the type of flat correlates with the resale price. In this plot, we tried to understand the distribution of the dataset with respect to the type of flat. It can be seen that the most common types of flats are 3-room, 4-room, and 5 room apartments.

Resale price distribution throughout the year for different types of flats.

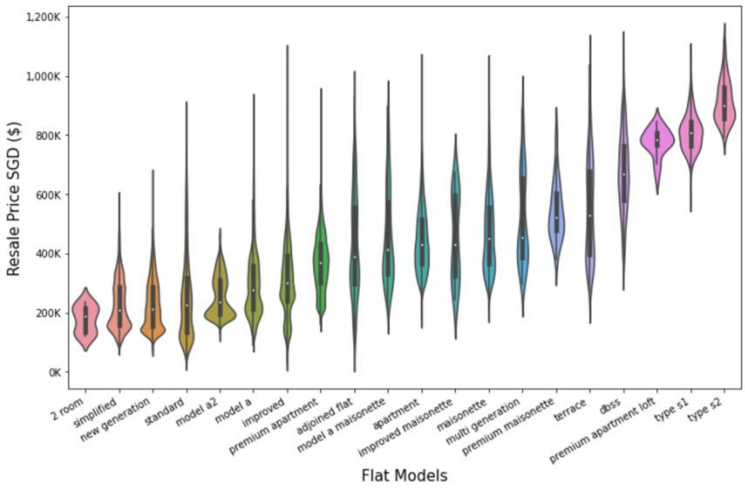
If we only consider the most common types of flats, it seems like the resale prices have not changed much for each flat type throughout the years, but the only difference is that the price increases with the number of rooms.



Flat Model

Resale price vs Flat Model Distribution.

In order to get a better idea on whether the model of the flat also has a correlation with the resale price, we came up with the following violin chart.

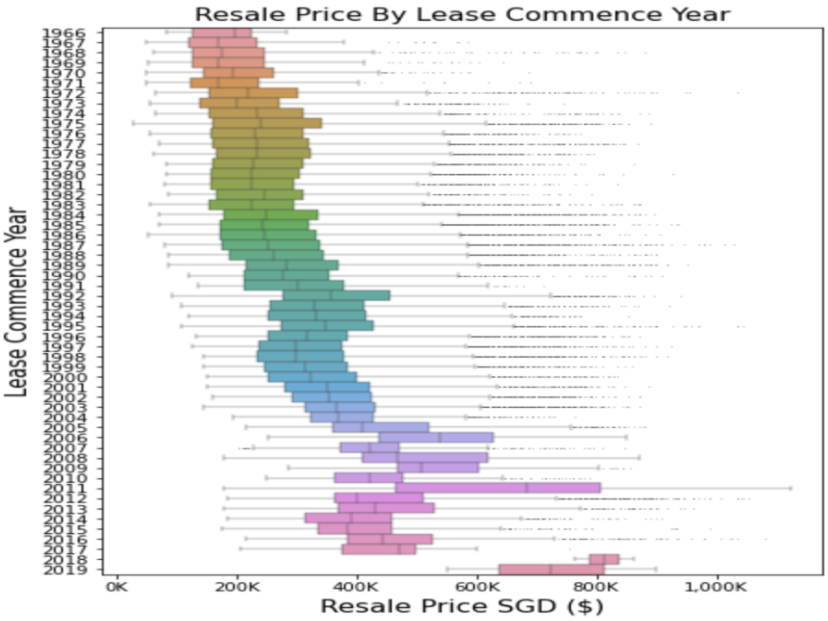


The dataset contains several models when to comes to flats in Singapore. We can see that there is indeed a correlation with the resale price. Hence, this input will be considered for the model training.

Commence Year

Resale price By Lease Commence Year

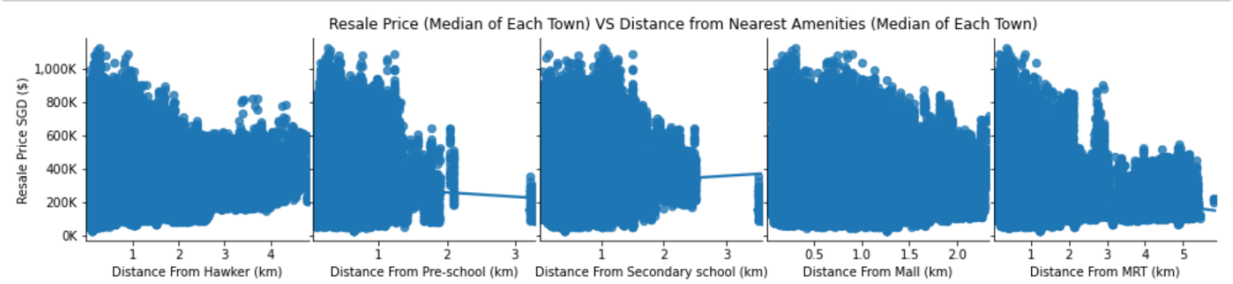
According to the correlation heatmap, the lease commence year also has a strong correlation with the resale price. It is depicted in the plot below too. Although there are several outliers, the median price clearly has a correlation with the commencement year.



Distance to Amenities

Resale price (Median of Each Town) VS Distance from Nearest Amenities

Several amenities were considered in the dataset. The main ones were the nearest hawker center, distance to pre-school, the distance to secondary school, distance to a mall, and distance to the MRT station. The correlation between these with the resale price is quite weak as seen by the plots below. *Hence, these will not be considered for the model training.*



3. Data Mining

Model Selection

For model selection, we tried several different approaches. Starting with a simple linear model, we implemented decision trees, random forest, bagging regression, gradient boosting, and artificial neural networks (ANNs).

Before being fed into the model, the data is cleaned and augmented. Some of the columns are dropped including block, town, street name, eco category, latitude, longitude, subzone, planning area, age variables, schools, and the MRT name. The categorical columns are converted into numerical; the date time columns are modified to be encoded as a single integer, and the values are scaled.

We used GridSearchCV from sklearn to do hyperparameter tuning of Tree-based models. The following parameters were tweaked.

```
parameters = {
    'n_estimators':[100, 150, 200, 250],
    'max_features': ['sqrt', 'log2', 'auto'],
    'min_samples_split': [2, 3, 5],
    'learning_rate': [0.01, 0.05, 0.1, 0.5]
}
```

Model	**Hyperparameters**	**Result**
Linear Regression	Default parameters	Mean squared error: 3387072961.49 Mean absolute error: 45055.39 R-squared: 0.80
Bagging	BaggingRegressor(n_estimators=40, random_state=0)	Mean squared error: 316661937.48 Mean absolute error: 12399.11 R2 score: 0.9810954341139839
Decision Trees	DecisionTreeRegressor(random_state = 0, max_depth = 50, min_samples_leaf = 100)	Mean squared error: 786995768.73 Mean absolute error: 19008.06669 R2 score: 0.9530167298281123
Random Forest	RandomForestRegressor(n_estimators=100, random_state=0)	Mean squared error: 326323831.339 Mean absolute error: 12750.7568 R2 score: 0.9805186236819932
Gradient Boosting	GradientBoostingRegressor(n_estimators= 300, learning_rate=0.2, max_depth=10)	Mean squared error: 286303191.42 Mean absolute error: 11913.619 R2 score: 0.9829078367030853
XGBoost	XGBRegressor('objective': 'reg:squarederror', 'learning_rate': 0.1, 'max_depth': 19, 'n_estimators': 100)	Mean squared error: 289524608.0 Mean absolute error: 11881.869 R2 score: 0.982715521381119
ANN – 1	Linear(input_size, 256), ReLU(), Linear(256, 128), ReLU(), Linear(128, 64), ReLU(), Linear(64, 32), ReLU(), Linear(32, 1) Learning rate: 0.01 Batch size: 512	Mean squared error: 385760816.229 Mean absolute error: 14140.2 R2 score: 0.9769702642754826
ANN – 2	Linear(input_size, 256), LeakyReLU(), Linear(256, 256), LeakyReLU(), Linear(256, 128), LeakyReLU(), Linear(128, 64), LeakyReLU(), Linear(64, 64), LeakyReLU(), Linear(64, 1) Learning rate: 0.02 Batch size: 512	Mean squared error: 349381079.369 Mean absolute error: 13575.90387 R2 score: 0.9791421171189456
ANN – 3	Linear(input_size, 512), ReLU(), Linear(512, 256), ReLU(), Linear(256, 128), ReLU(), Linear(128, 128), ReLU(), Linear(128, 64), ReLU(), Linear(64, 64), ReLU(), Linear(64, 32), ReLU(), Linear(32, 1) Learning rate: 0.05 (Divides by 1.5 every 10 epochs) Batch size: 256	Mean squared error: 329983107.599 Mean absolute error: 13131.349 R2 score: 0.980300166730642

XGBoost had the best results.

4. Evaluation and Interpretation

For the evaluation of different models on predicting the result, we use **hyperparameter-tuning and 5-fold Cross-Validation.** The dataset is split based upon the predefined random seed for reproducibility. We train the model using the cleaned and preprocessed training dataset and compare the performance of different models based on the:

- Root Mean Error score
- Root Mean Absolute score
- R2 score

of the predicted results on the cross-validation dataset.

After intensive tuning for different methods, we find that the XGBoost method has the best prediction result. This is because XGBoost is an ensemble of decision trees that uses the Gradient Boost framework, which minimizes errors when constructing multiple sequential trees. It has consistently shown good results on our data for various choices of hyperparameters. In addition, XGBoost's regularization technique further reduces the overfitting and generalization errors, and compared to other scikit-learn families of algorithms, XGBoost has a good built-in missing data handling feature.

XGBoost	RandomForest	ANN	Logistic Regression
Uses sparse matrices with sparsity aware algorithms. Uses data structures for better processor cache utilization, which makes it faster. Supports multicore processing better, which reduces overall training time. Supports DART, the dropout regularization for regression trees. **RMAE: 11801**	Training time was 24x XGBoost. Uses an average ensemble of randomly bootstrapped decision trees instead of correcting residual values. Was quickly diverging by increasing no. Of estimators. (even though learning rate was decreased appropriately). **RMAE: 12750.75**	Showed similar results to XGBoost. Requires an intrinsically designed neural network architecture based on intuition and hyper-parameter tuning. However, the number of trainable parameters increase with increase in network size. **RMAE: 13131**	We treated this as a baseline. The relationships in the data were too complex to be modelled as a simple linear regression model. Performed the worst. **RMAE: 45055.39**

Overall, we see that XGBoost performed the best among all the models with a testing accuracy of 98.2%. Our Kaggle rank was 19 during the submission and it is important to note that all submissions above us are extremely close in accuracy. This indicates that our implementation reached the ballpark of near optimal very closely.