Technical Documentation

Our project is targeted at buyers of HDB resale flats, who can use our website to view predicted HDB resale prices and compare it with the listed price of an existing HDB resale flat. In our technical documentation, we will outline the steps the backend and frontend team took to build the model that churns out the prediction of HDB prices. Our project is deployed at this url.

Navigating the GitHub Repository

The Github Repository for the prediction of HDB resale prices can be found at this url. This repository can be cloned via RStudio or any other IDE.

We split our repository into two folders:

- The backend houses the code for the pre-processing of data, exploratory data analysis and training of the machine learning models that we considered.
- The frontend houses the code for the visualization and the design of our app, separated into the UI, server and global settings files.

In the respective sections below, we will indicate exactly which file we are referring to in our Github Repository and what each R file does. To replicate the steps that we took for the project, you may follow the chronological order within our technical documentation which highlights the files that we used and what steps were taken within each file. You may also refer to the README.Md in our Github Repository.

Backend Technical Documentation

A. Data Sources (backend/raw_data/data_name.csv)

We first gathered the data[1] that we needed for the HDB resale prices. We also gathered names/addresses of various amenities that we think might influence property prices, namely MRT stations, hawker centres, supermarkets, primary schools, and hospitals.

B. Data Preprocessing

Even though there was HDB resale prices data from 1990-2024 available, we decided to use only HDB resale flat transactions from Jan 2017 to Mar 2024 due to our limitation of obtaining accurate geospatial data that covers transactions before 2017. For instance, Marymount MRT station was opened in May 2009, when obtaining nearest distances to MRT stations and counting the number of nearby MRT stations, our dataset would have some erroneous entries for geospatial data before May 2009 due to the inclusion of this MRT station and thus introducing bias into our predictions. To overcome our limitation of not having access to the past geospatial data, we assumed in our case that the location of amenities stayed constant from 2017 - 2024.

Another reason for limiting to 2017 data onwards was because there were sufficient data points (~175k transactions) to build our regression models and train our machine learning models.

Our data preprocessing was split into two parts:

1) Obtaining Geospatial Data of HDB blocks and amenities (backend/geospatial data functions.R)

Geospatial data was lacking for our HDB flats and amenities so we had to gather geo-coordinates and other essential details such as postal code (if not already provided in our dataset) through API calls to OneMap. The “search” API in OneMap allows for the input of either postal codes or street addresses and then outputs the latitude, longitude, x-y coordinates and postal codes. Before we utilised the API, we made sure that our datasets of the HDB resale flats

[1] Our data sources are listed in the appendix.

and various amenities were cleaned such that a valid address/postal code was fed into the API call. For locations with no valid results from the API call, we gave those locations a default geo-coordinate and subsequently filtered them out from each of the dataset. For each amenity, we saved the results of the API call in a Rds file named (type of amenity)_geocode.Rds located in the processed_data folder. For HDB blocks, we filtered out the unique HDB blocks in the HDB resale transactions dataset and then ran the API call to get the geo-coordinates of each HDB block as well as the postal codes.

Given the latitude and longitude of each HDB block and each amenity, we were able to map the nearest amenity, distance to nearest amenity and the number of amenities within that block's 1km radius for each HDB block using the geosphere library. We also mapped the distance from each HDB block to the Central Business District (Downtown Core). We were then able to obtain a dataset that contained the geospatial details of each unique HDB block, which we saved in hdb_block_details_w_schools_mrt_name.Rds.

2) Cleaning of the HDB resale transactions dataset (backend/Data Pre-Processing.R)

For the HDB resale transactions, we made several changes to the variables in the dataset.

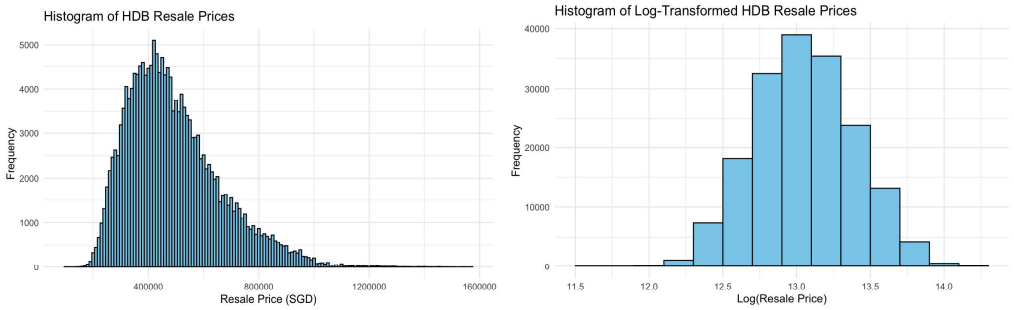
- Converted remaining lease from "X years Y months" to a continuous variable. For example, converting "61 years 06 months" to 631.5 years.
- We separated the date of transaction into month and year variables to create time dummies. This helps to control seasonality and time fixed effects.
- Converted storey_range from categorical to continuous. For example, we will take the average of "01 TO 03" as 2 for the storey range.

We left-joined this dataset with the geospatial dataset above via the HDB block address. For the purposes of our prediction, we extracted only the first two digits of the postal code, which in the Singapore context represents the sector code in Singapore. Along with the town names, the 2-digit postal codes are used to control for spatial heterogeneity for hedonic analysis.

We then performed one hot encoding on the categorical variables to create dummy variables for them. Ultimately, our final dataset consists of **175359 observations** with the **response variable resale prices** and **136 covariates**.

C. <u>Exploratory Data Analysis</u> (backend/Exploratory Data Analysis.Rmd)

To test for multicollinearity, we first filtered out the columns of the categorical variables. We plotted out the correlation matrix of the continuous variables for visualisation. We then calculated the variance inflation factors (VIF) of our continuous regressors. Using VIF > 5 as a benchmark for multi-collinearity, all of the regressors had a VIF < 5 suggesting not a serious problem with multicollinearity.



<u>Histogram of HDB Resale Prices (L) vs Histogram of log-transformed HDB Prices (R)</u>

We also plotted out the distribution of HDB resale prices and observed a right skewed distribution as seen in the left graph above. To ensure that HDB resale prices follow a more

normal distribution, we performed a log transformation on HDB resale prices and obtained a distribution that is more normally distributed as seen in the right graph.

We then used Cook's Distance to test for influential points in our data set. We did not find any influential points when Cook's Distance > 1 is applied on our linear regression model.

D. Machine Learning (backend/ML.Rmd)

When running the code for the train/test split and for all the machine learning models, we set the seed by invoking set.seed(42). This is to ensure the reproducibility of our code.

D.1. Train/Test split:

Before we split the data into train and test set, we realised that we had an imbalanced dataset, where there were unequal numbers of observations belonging to each of the unique years recorded in the dataset (i.e. 2017 - 2024).

As such, we conducted stratified sampling by year to ensure that our train / test dataset has equal proportions of observations belonging to each year. This minimises the probability of training biassed models and making biassed predictions. We placed 35,072 observations (20%) into our train dataset, and 140,194 observations (80%) into our test dataset. We had a smaller proportion allocated to the training set as bigger train data leads to longer model training time without significant improvements in model predictive ability.

We rescaled our resale_price to log(resale_price) to ensure that the y variable is approximately normally distributed, as resale_price displays a right skewed distribution as mentioned above.

Finally, we labelled training data as train_ml which contained 136 covariates and the response variable and test data as test_ml, containing the 136 covariates only.

D.2. Running the Machine Learning Models and Testing Code

To run each machine learning model and test the code, we need to firstly set up the Environment. This is done by ensuring proper installation of R and RStudio on the system. Install the required libraries by running install.packages(c("tidyverse", "randomForest", "rpart", "VIM", "gbm", "hdm", "xgboost", "recipes")) in RStudio. Next, we load the data by making sure the processed data file (hdb_resale_prices.RDS) is available in the specified directory or adjust the file path accordingly. Next, we run each model. To do this, we open the R Markdown file (ML.Rmd) in RStudio and execute each code chunk sequentially by pressing the "Run" button or using the shortcut Ctrl + Shift + Enter. Each ML model has its own code chunk. Note that for each model, training the model (including hyperparameter tuning) is done by fitting data = train_ml, and predictions made to calculate RMSE are done by fitting data = test_ml. Compare the RMSE with other models to assess relative performance.

D.3. Model Performance Study

Table summary of all ML models considered and their comparative performance, as measured by the Root Mean Squared Error (RMSE) given our log transformed resale prices:

Model	**RMSE**
Benchmark OLS based on domain knowledge	0.1349587838384
Post - LASSO Regression	0.104452066137839
OLS using important features selected from XGBoost	0.128174207150794
Regression Trees	0.185634606041432

Random Forest	0.0631247324693827
Bagging	0.0670001339301782
Traditional Gradient Boosting Machine (GBM)	0.0919899399523445
Extreme Gradient Boosting Machine (XGBoost)	0.057909164860733

D.4.1. Model Discussions

Under this section, we will discuss the specific features chosen to be fitted into each Machine Learning Model, the specific instructions on tuning, and what are the hyperparameters we tune for each model together with some explanations. We will also evaluate the strengths and limitations of each model alongside their performance, measured by RMSE. Lastly, we elaborate on the reasons why we considered each model and our justification on our final chosen model.

A. Regression Models (log-linear specifications):

A.1. Ordinary Least Squares (OLS) Regression:

In the Ordinary Least Squares (OLS) Regression model, we utilised the following features as covariates: floor_area_sqm, dist_cbd, remaining_lease, year, ave_storey, dist_to_nearest_mrt, and dist_to_nearest_primary_schools, chosen based on domain knowledge. The model achieved an RMSE of approximately 0.135, serving as a benchmark for other ML models to surpass for consideration as our final choice. Additionally, the adjusted R squared indicated that this model predicted log_price values with 83.33% accuracy, with all chosen covariates being statistically significant at a 1% significance level. OLS regression possesses strengths in being simple, interpretable, and computationally efficient.

However, it assumes a linear relationship between log_price and covariates, potentially limiting its ability to detect and capture nonlinear interactions. Violations of assumptions such as linearity, normality, and homoscedasticity of residuals can lead to biassed estimates and inaccurate predictions. Moreover, high multicollinearity among predictors can inflate standard errors and affect coefficient interpretation, necessitating remedies like feature selection or regularisation techniques.

A.2. Post-LASSO Regression:

Our Post-LASSO Regression model achieved an RMSE of approximately 0.104, which is lower than that of our benchmark OLS model. For feature selection, we utilized all 136 covariates in our model. Our code automatically conducts variable selection and applies OLS regression to estimate the coefficients of the selected covariates after implementing the LASSO regularization technique. LASSO introduces a penalty term (lambda) to the standard regression equation, shrinking the coefficients of less important features to 0, and selecting features with non-zero coefficients. Detailed results can be accessed by calling plasso$coefficients, where "plasso" denotes our assigned name for the fitted Post-LASSO model. We did not tune the hyperparameter lambda, utilizing the function lasso() from the hdm() package, known for implementing the optimal data-dependent value of lambda. This approach performs on par with, or better than, the glmnet package with hyperparameter tuning via cross-validation. Hence, we opted for the default hyperparameters chosen via rlasso() due to their robustness.

LASSO, known as L1 regularization, controls the level of regularization applied to the penalized residual sum of squares (RSS), with lambda being the variable that governs this regularization.

Addressing multicollinearity issues in the dataset, the Post-LASSO model possesses feature selection capabilities and fits the model based on the most important features. However, it assumes a linear relationship and may not adequately capture complex nonlinear interactions between covariates. Additionally, we opted for Post-LASSO over LASSO regression as LASSO performs variable selection and coefficient estimation by penalizing the absolute values of the coefficients using a specified lambda value. This penalization leads to the shrinking of coefficients towards zero, potentially resulting in biased estimates, especially for variables with small, non-zero effects[2].

In contrast, Post-LASSO addresses the biases associated with LASSO. Post-LASSO first performs variable selection using LASSO and then applies ordinary least squares (OLS) regression to estimate the coefficients of the selected covariates. OLS regression provides unbiased coefficient estimates since it does not penalize any coefficient, allowing for more accurate estimation of the true values of the coefficients.

A.3. OLS using variables selected by XGBoost:

We utilised XGBoost's variable selection feature to extract important features and performed regression of log_price against the selected covariates using OLS regression. The chosen features include floor_area_sqm, dist_cbd, remaining_lease, year, ave_storey, dist_to_nearest_mrt, and dist_to_nearest_hawkers. XGBoost selects features based on three crucial metrics: Gain, Weight, and Cover. Gain measures the average improvement in the model's objective function when a feature is used across all stages of boosting, indicating its contribution to predictive performance. Weight represents the frequency of a feature's appearance in the trees across all stages of boosting. Cover reflects the average number of observations affected by a feature across all stages of boosting, indicating its relative importance based on the number of affected observations (Lawrence, 2023).

In terms of performance, this model explains 85.5 percent of the variation in log_price (adjusted R squared), with an RMSE value of 0.128. All variables selected by XGBoost are statistically significant at a 1 percent level.

Evaluation of log-linear models: We considered log-linear OLS regression models due to their simplicity and powerful predictive ability with correctly specified feature engineering. However, one limitation is that it may not be able to capture nonlinear interactions as extensively as other ML models, as non-linear interactions must be explored and specified manually. In addition, as a number of our covariates are correlated with each other, OLS regression may not be the optimal model to use due to its lack of features in handling correlated regressors. As such, we move on to consider regression trees to better capture nonlinear specifications and interactions.

B. Tree-Based Models:

Aim: We considered tree-based methods due to their advantages over linear-based methods (OLS). Firstly, it allows a wider variety of functional forms to best explain our data and secondly, it automatically detects nonlinear relationships and interactions between covariates.

B.1. Regression Trees (General):

The general algorithm begins by growing large trees. It recursively conducts binary splits by selecting a predictor Xi and a cut-off point n. This process divides the predictor space of Xi into two regions: $\{X|Xj < s\}$ and $\{X|Xj \geq s\}$, aiming for the greatest reduction in the Sum of Squared Residuals. The algorithm continues to perform binary splits within each partitioned region until

[2]See equation 1 in Appendix to see the role of the hyperparameter lambda in LASSO regression.

the stopping condition is met, typically when each region contains no more than t observations, where t is usually set to 5. Pruning of large trees is then conducted to obtain a more parsimonious split, often by selecting the subtree with the lowest cross-validated error (Lam, 2021).

Our code implements default hyperparameter tuning, and the performance is measured by obtaining an RMSE of around 0.186. Regression trees have strengths in their ability to capture nonlinear relationships and interactions between variables. However, they are prone to overfitting and lack interpretability compared to linear models. Due to their high variance, they are often used in ensemble schemes aimed at reducing variance (Lam, 2021).

B.2. Regression Trees with Ensemble Methods:

A. Parallel Ensemble

Learners are generated in parallel, where base learners are generated independently to explore the datasets, allowing them to explore different sets of effects and patterns. The overall prediction error is reduced by averaging the results of prediction. Examples of these models include Bootstrap Aggregation (Bagging) and Random Forest (Hachcham, 2023).

1. Bootstrap Aggregation (Bagging):

The general algorithm fits highly biased but low variance trees, and the final prediction is averaged from that of multiple tree models using the bootstrap method. This approach achieved a low RMSE of about 0.067, with the optimal hyperparameter being 1000 n_trees. All features were used in this model. As ensembles of decision trees have demonstrated strong performance in economic forecasting using default settings, no cross-validation is required for tuning. We opted to fit n_tree = 500 and 1000, with n_tree = 1000 proving to be optimal and producing a lower RMSE. Given that bagging utilizes out-of-bag errors to validate the model, a large n_tree of at least 1000 is necessary to stabilize the error. For bagging, mtry = P = number of covariates, as trees are not decorrelated.

The strengths of bagging lie in its ability to reduce the model variance of the base model (regression tree) by averaging predictions from multiple base models trained on different subsets of the training data, leading to more stable and reliable predictions. Additionally, bagging improves generalization by training base learners on different subsets of the data, reducing the risk of overfitting and enhancing performance on unseen data, particularly beneficial for handling complex datasets with noisy or sparse features. Bagging is also inherently robust to outliers in the training data because it combines predictions from multiple models, each trained on a subset of the data, thus mitigating the impact of outliers on overall predictions and resulting in more robust and stable model performance. However, a limitation arises from the potential correlation between trees, which can diminish the benefit of averaging to achieve an accurate final prediction.

2. Random Forest:

The general algorithm for Random Forest involves fitting a subset of predictors in each tree model, aiming to decorrelate the tree models, which ultimately improves the benefit of averaging to generate more accurate predictions. This approach achieved a lower RMSE of about 0.063 compared to bagging, with the optimal hyperparameter being 1000 n_trees. All features were utilized in this model. Tuning involved conducting 10-fold cross-validation to determine the number of bootstrap iterations (n_tree), selecting between 500 and 1000 trees based on their proven performance in economic applications. The mtry was fixed to a default setting of the number of covariates (P) divided by 3 for regression problems. The strengths of Random Forest lie in its ability to reduce correlation between trees by using feature bagging, where each tree is trained independently on a subset of features, leading to less correlated predictions and

exploring a more diverse set of models, particularly suitable for datasets with many correlated predictors. Random Forest was included in our model selection to address limitations from bagging, which it effectively mitigated, as reflected in the RMSE. However, it is less interpretable than individual trees and has a slower training process compared to single decision trees.

B. Sequential Ensemble

Tree models are generated sequentially. The next learner will learn from the errors made by the previous one to generate a more accurate model. The models that fall under this category would be Traditional Gradient Boosting(GBM) and Extreme Gradient Boosting(XGBoost) (Hachcham, 2023).

1. Gradient Boosting Machine (GBM):

This method yielded an RMSE of approximately 0.092, worse than both Random Forest and bagging. All features were employed in this model. Tuning involved conducting 10-fold cross-validation to optimize the hyperparameters (n_trees and interaction_depth). The number of trees was selected between 500 and 1000 based on their performance in economic forecasting. For interaction depth, options included 2 (indicating a close-to-linear relationship in the data) and 5 (indicating significant non-linear relationships with multiple interactions between covariates). The shrinkage value (learning rate) (Boehmke, 2019) was fixed to 0.01 as a common setting practice. Optimal hyperparameters were determined to be n_trees = 1000 and interaction_depth = 5, suggesting the presence of many nonlinear relationships within the data. Strengths of GBM include its ensemble method, which combines multiple weak learners to often provide high predictive accuracy. However, it is sensitive to hyperparameters, requires longer training time, and is prone to overfitting, as indicated by its relatively high RMSE in our dataset application (Mohtadi, 2017).

2. Extreme Gradient Boosting (XGBoost):

XGBoost achieved the lowest RMSE of about 0.058 among the models evaluated. All features were utilized in this model, with XGBoost automatically selecting certain covariates to be fitted into the final data. Tuning for XGBoost can be challenging due to numerous hyperparameters to consider. Initially fitting XGBoost with 500 trees (nrounds) and default values for other hyperparameters yielded the lowest RMSE compared to other ML models. Subsequently, only the nrounds hyperparameter was tuned, with values tested at 500 and 1000. Hyperparameters for XGBoost include nrounds, representing the number of trees to train.

Boosting evaluation[3]: We considered both a traditional gradient boosting machine and extreme gradient boosting machine to improve on the limitations of random forest and bagging. Boosting is another well-known tree-ensemble method for regression. The aim is to fit small tree models with low variance and initially high bias, with added prediction iteratively to reduce the bias. Boosting uses sequential training on weak learners to spot and improve the limitations of previous learning via gradient descent (Brownlee, 2016). Thus, unlike random forest and bagging where they fit a single large tree model, boosting starts off from small tree models and allows them to learn the algorithm step-by-step. This is done to prevent the final model from overfitting. The tuning hyperparameter is usually the learning rate (which measures the contribution of each tree to the final model), but since the default learning rate has already generated well-performing models, in this case we do not change the learning rate further. The use of weights (sequential ensemble) over simple averaging (parallel ensemble) of the predictors from multiple tree models further minimise loss and produces more accurate predictions, leading us to consider boosting over bagging and random forest (Tseng, 2018).

[3] See table 2 in Appendix for a more detailed reference of strengths of XGB compared to traditional GBM

Overall Model Evaluation:

We decided to use Extreme Gradient Boosting as our final solution, as this model has features that tackles the weaknesses of our dataset and performs the best (lowest out-of-sample RMSE) as compared to other models. Our two main concerns with our dataset lie on the fact our dataset is large with a lot of correlated variables / regressors. We did not do any transformations to reduce collinearity as our variable inflation factors are not large enough (well below 5) for collinearity to be a serious problem. However, we suspect that the presence of so many correlated variables may pose a challenge towards churning out accurate predictions, and overfitting. As such, since traditional gradient boosting does not have any features to handle multiple correlated regressors, we considered extreme gradient boosting for the presence of its regularisation techniques to handle multicollinearity. As the RMSE of XGB when predicting using our test dataset (140,000 observations) is the lowest among all the other ML models, this shows that our XGB model is not overfitted to our train dataset (35,000 observations). Thus, due to the advantages of boosting over bootstrap aggregation, we prefer boosting techniques. As XGB tackles the limitations of our data that were not addressed by GBM and other ML models, we chose XGB as our optimal model. Our literature research is supported by our results that shows XGB has the best predictive ability (lowest RMSE).

One future consideration is that we could train more samples, reduce the number of features used in model fitting, and reduce the maximum depth and minimum sample of leaves to completely control overfitting (Hachcham, 2023). In addition, we could have tuned all hyperparameters using 10 fold Cross-Validation to find the absolute best hyperparameters for optimal model performance. Another model we could have considered would be the use of neural networks as they have demonstrated remarkable capabilities in handling complex data patterns and large datasets (K. Haritha et al., 2023).

D.4.2 Model Tracking and Improvement

After deploying the models, it is crucial to monitor their performance and make necessary improvements. This involves regularly tracking model performance metrics such as RMSE using monitoring tools. Additionally, gathering feedback from users and stakeholders helps identify model shortcomings or areas for improvement, establishing a feedback loop for continuous enhancement. Periodically retraining the models using new data allows for the incorporation of recent trends and improvement of performance over time. Continuously optimising hyperparameters based on performance feedback further enhances model accuracy. Exploring ensemble techniques like model stacking or blending helps combine the strengths of multiple models and mitigate individual weaknesses. Maintaining comprehensive documentation detailing model architecture, training procedures, and performance evaluation facilitates understanding and future enhancements. By iteratively refining the models based on feedback and leveraging ensemble techniques, we can enhance their predictive capabilities and ensure their effectiveness in addressing real-world challenges.

Front-end technical documentation

A. Global_settings.R file

This R script is the core of our interactive Shiny application, incorporating essential libraries for data manipulation, visualisation, and UI interaction—'dplyr', 'ggplot2', 'plotly', and 'shiny'. It loads and processes critical datasets, including maps and address listings, to support geospatial visualisations and predictive analytics.

The script organises geographical and predictive data to enhance both the UI and server components of the application, ensuring a smooth user experience. Functions within the script handle data preprocessing, categorical conversion, and one-hot encoding, which are crucial for integrating machine learning models and processing user inputs. This setup not only facilitates accurate HDB price forecasts but also enables dynamic, visually engaging representations of data through interactive maps and charts.

Our front-end code can be found in the frontend folder of the repository and is organised into distinct sections corresponding to the various tabs of the application in both the UI (frontend/ui.R) and server (frontend/server.R): the Geospatial Analysis tab, the Predicted Price tab, and the Price Trend tab, each meticulously structured within both the UI and server components. Therefore, our technical documentation has also been separated into these few tabs and the flow of our explanation of our codes in the UI and server tabs, as well as justification of our visualisation, will also be grouped into the different tabs.

The design and layout of the application were predominantly influenced by considering the perspective of potential buyers. We aimed to determine the most effective manner for users to interact with and derive value from the app. Consequently, we have engineered each tab to be self-explanatory, clearly conveying its intended purpose and utility. Our objective is also to ensure the app remains accessible, particularly for first-time users.

Firstly, for the overall design of the webpage, we have incorporated insights from academic lectures and a thorough review of numerous projects from the ShinyTuesday gallery to enhance both the inspiration and effectiveness of our application's presentation. For example, during a guest lecture by Rebecca Pazos, the data visualisation editor from SPH, it was highlighted that most readers engage with only the first 20% of content. This observation has guided our design strategy, focusing on creating an immediately engaging and visually captivating interface to foster trust with users upon their initial interaction with the app. Additionally, to maintain user engagement and encourage exploration of all features, we meticulously designed the app's layout and colour scheme. Given that this project is related to the Housing Development Board (HDB), we opted for the HDB colour palette to convey a sense of officialdom and reliability. The layout is deliberately sleek and concise, facilitating easy navigation through tabs and controls without overwhelming the users. These design choices were implemented through the dashboard configuration in the UI tab, with further details provided below:

For instance, the header in the dashboard layout is styled with a white background and black text, providing a clean look for users of the website. For the dashboard sidebar, it features a dark red background with white text, which ensures high readability and a visually attractive contrast. This colour scheme also highlights active menu items with a white border to enhance user navigation experience. This colour scheme is fitted to the colour scheme of our HDB in Singapore, to invoke a sense of familiarity. We have also come up with a sidebar that shows different tabs and is clear for users to click through. For our dashboard body, we used a CSS styling format. Similar styling rules are reiterated here to possibly correct or reinforce sidebar styles not applied initially due to load order or specificity issues in CSS.

B. Home Tab

Going into the codes of the home tab, the aim of this tab is to allow users to have a better understanding of how to navigate this page, as well have a better understanding of how our app works. We added a motivation for our app, so that users can understand that if they fall into this group of people that our app is targeting, then they are looking at the right place. We also

included a related picture of HDB, so that what our app is related to is clear from the moment users open our page – a HDB-related app.

1. Choice of visualisation & code explanation

The 'Home' tab, as the initial interface users engage with, is crafted with particular attention to aesthetics. We achieved a responsive, center-aligned layout using a fluid row structure and flexbox CSS. This approach, articulated through the code `div(style = "flex: 1; display: flex; justify-content: center; align-items: center;", ...)`, allows the design to accommodate various screen sizes, ensuring a consistent and adaptable user experience.

To enhance the visual impact upon arrival, we integrated a full-sized image that responsively adjusts to the viewport, governed by the style max-width: 100%; height: auto; in the image tag. Complementing the imagery, we introduced an introductory text that outlines the site's intent and advantages for first-time home buyers, rendered seamlessly within the server tab. This design strategy not only aligns with web design best practices but also provides an informative and visually appealing gateway for users.

C. Geospatial Analysis Tab

The aim of creating this tab is to allow users to explore the amenities near their selected housing block. We designed this tab to help users better understand the distances from the amenities to their selected HDB block. For the front-end design, we planned to feature an interactive map, providing users with a clear view of the amenities around the selected block. We also intended to insert a marker for the selected block to make users' choices more evident. Most importantly, we designed the map to be interactive, enabling users to drag it around and explore nearby amenities. Additionally, taking into consideration first-time users of our app, who may be unsure about entering inputs for visualisations, we included a user manual to clarify how to use our app. We segmented the inputs and outputs because this would give a clean look, and this was the insight obtained by looking at how other developers presented their visualization on ShinyTuesday.

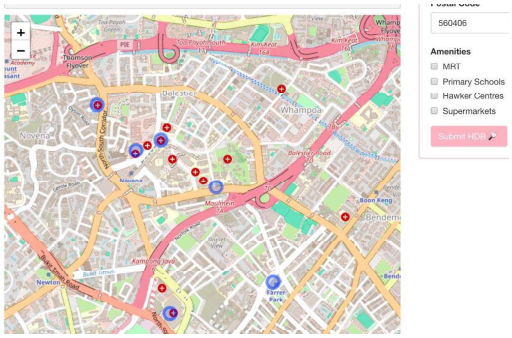
1. Choice of visualisations

Below are some visualisations that we have considered, when presenting our geospatial data so that users can best utilise and understand our app:

a. Having an interactive map with all the markers popping out so that the users can identify the blocks in each area, as well as amenities being highlighted

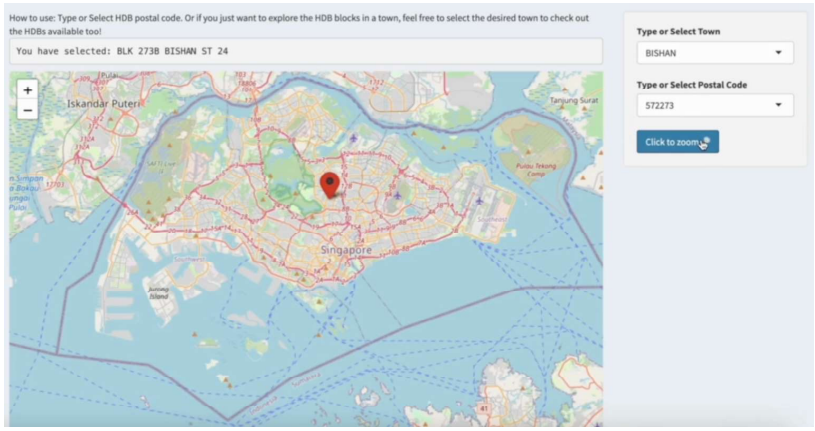
However, we then realised that this method would be way too messy for users who already have a specific block in mind and would simply want to explore the amenities around the house. Therefore, we decided to come up with a marker that would pop out when a specific postal code is inserted.

b. Highlight the amenities around the map using circle markers (where the markers are in blue), so that amenities would be more obvious to the users



However, we found the markers to be redundant after zooming into the map, as the leaflet package already has the different amenities labelled out (as seen in the above screenshot). After researching this topic on the needs of buyers, we found that they were more interested in the specific details and the nearest amenities to their house. Therefore, we decided to zoom in to only the nearest amenities of each house by utilising the OneMap API as designed by the backend team, so that users are able to obtain the information they need and want to know.

c. We then improvised and combined the ideas from the two versions in a and b to formulate this current and final visualisation.



Nearest MRT Station	Distance to nearest MRT (in km)	MRT stations within 1km
BISHAN MRT STATION (NS17)	0.781038596601709	3

Showing 1 to 1 of 1 entries

Nearest Hawker Centre	Distance to nearest Hawker Centre (in km)	Hawker Centres within 1km
BLK 320 SHUNFU ROAD	0.847472389449456	1

Showing 1 to 1 of 1 entries

Nearest Primary School	Distance to nearest Primary School (in km)	Primary Schools within 1km
CATHOLIC HIGH SCHOOL (PRIMARY SECTION)	0.216824354357361	1

Showing 1 to 1 of 1 entries

Nearest Supermarket	Distance to nearest Supermarket (in km)	Supermarkets within 1km
279 BISHAN STREET 24 DBS NTUC BISHAN STREET 24 SINGAPORE 570279	0.139908289820262	3

Showing 1 to 1 of 1 entries

Nearest Hospital	Distance to nearest Hospital (in km)	Hospitals within 1km
820 THOMSON ROAD OCBC MOUNT ALVERNIA HOSPITAL SINGAPORE 574623	1.68609754205403	0

Showing 1 to 1 of 1 entries

We then came up with this map because we wanted users to be able view the amenities and explore any amenities of their choice. Slimming down to only one marker per block would also ensure that the design is cleaner, as what was learnt in Prof Huang's lecture about how the visualization should not interfere with the point we are bringing across. The point we are bringing across in this case, is that we want users to have a better idea of the nearest amenities around their selected ideal home.

2. Code explanation

We have split our codes into two columns, the main column and the side column.

Our tab is named "geospatial" and features a fluid layout that organises the user interface into responsive rows and columns. The components within the UI are designed to interact with the server-side logic of the Shiny application. Users can select specific towns or postal codes from dropdown menus, which then update the tables and maps displayed on the page. An action button is used to trigger a zoom-in function on the map output. For the server side, codes like `observeEvent` and reactive functions are created so that the map and actions change according to the different inputs given.

For the main column, an interactive map that displays the markers as well as data tables that display information of the nearest amenities are shown:

A text output placeholder for the user manual as well as the confirmation of the address of the user's would be required. This is to provide a user manual for users, so that first-time users are able to understand how to use, and what they have to do so that they provide the required inputs for the map. We used the `verbatimTextOutput("geoSelectionOutput")` so that the text output is formatted and can be highlighted to the user when the output is a confirmation. Next, the outputs for the map are created, using the Leaflet package, to render the interactive map for users to see their selected point (indicated by a maker) as well as the amenities nearby. We also set the view initially to the entire Singapore map, so that users can see the entire map when they select a postal code, as indicated by the markers. When hovering over the markers, the markers will also display the detailed information of the selected house. Lastly, data tables are shown based on the filtered postal codes as well as amenities. Using renderDT function, the respective amenities are then shown using an interactive data table, while taking in data that has been filtered using the reactive function. The inputs for the data table were decided from the buyer's perspective. We considered what the buyers would want to know when buying a flat, and we figured that they would want to know the nearest amenities as well as how many amenities are in the 1km radius of the selected HDB block. We then included these factors into the headers of our table. The data table will also be modified such that the row numbering and the search function are removed. This is so that only a single observation will be returned, in this case, the single observation refers to the amenities that are nearest to the selected HDB block.

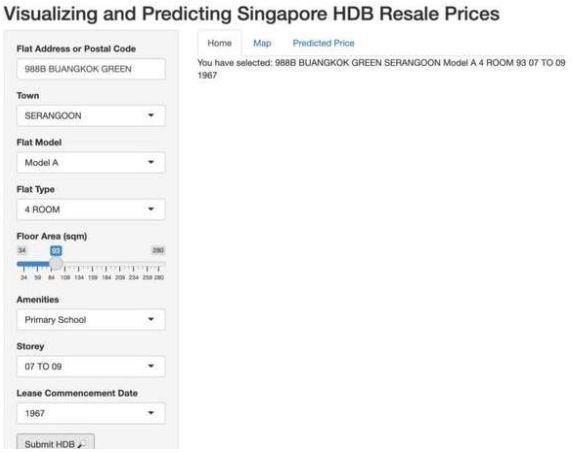
For the side column, the inputs for the towns as well as postal code dropdown boxes are created, so that users can select the inputs for the outputs to be displayed:

A dropdown that allows users to select or type a postal code. The selection will be funnelled into the interactive maps, as well as the data tables. The list is sorted in descending order, so that it is clearer when it comes to selecting a postal code. To filter for the postal codes in a selected town for our input, we used the reactive function, so that the sets of postal codes that are displayed in the dropdown selection box will change according to the towns changed. In other words, only postal codes belonging to the selected town will be displayed. From there, users can choose the postal codes based on the towns selected. In addition, to zoom the map when the action button is clicked, an observeEvent function is used so that the map can be zoomed according to the latitude and longitude of the given input (postal code), so that users can have a closer look on the nearby amenities on the leaflet map.

D. Predicted Price Tab

The aim of creating this tab is to provide users with a comprehensive tool to accurately gauge the resale price of any HDB unit they have in mind. This is crucial for helping users plan their budgets more effectively. The "Predicted Price" tab in the Shiny application is designed to collect user inputs necessary for predicting the resale price of an HDB unit. These inputs are linked to a machine learning model, which then displays the predicted price of the housing unit based on factors such as the housing model, housing type, and floor level of the unit. Additionally, information about the remaining lease is provided to help users understand how long their chosen unit will be their home.

1. Choice of visualisations
    a. Our first round of design

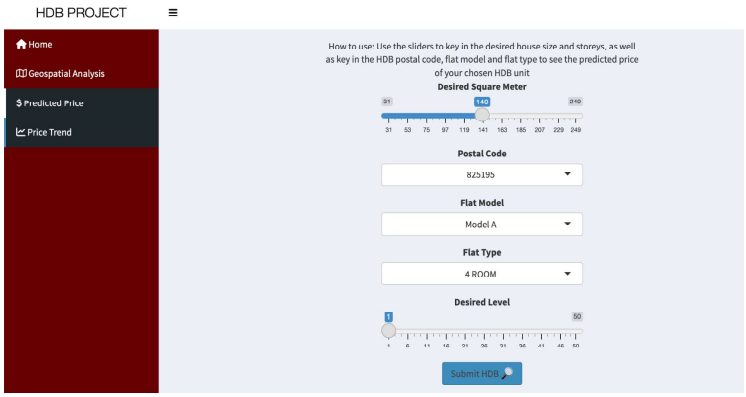


When we were determining the layout for our website's input fields and tabs, our initial design included a sidebar on the left that would be consistent across all tabs, but it ended up resulting in a cluttered appearance. It became clear that the sidebar was unnecessary for the home tab, as the input fields were only relevant to the 'Predicted Price' and 'Price Analysis' tabs. This realisation led us to adjust our design approach to better suit the specific needs of each tab, ensuring a cleaner and more user-friendly interface.

    b. Our second design idea

In our second iteration, we refined the design by confining input fields to their respective tabs, which improved the layout significantly. This change ensured that the home tab remained uncluttered, while input fields pertinent to price prediction were isolated to the 'Predicted Price' tab. Although this approach marked a clear improvement, providing ample space and a more organised structure, it also introduced a new challenge. The full-width input bars dominated the screen, creating an imbalance in the visual real estate and imparting a somewhat unfinished, unprofessional appearance, inducing the potential for further enhancements

Our final design



In the final design, we've refined the UI to present a clear and professional aesthetic. The relocated sidebar to the left tidily segments navigation, offering users an uncluttered space in the central 'Predicted Price' tab for inputting details. The design smartly integrates a modal dialog for detailed predictions post-submission and includes a user manual for ease of use. This efficient layout ensures intuitive navigation and a visually pleasing experience, free from distractions.

2. Code explanation

User Input and Interface Configuration:

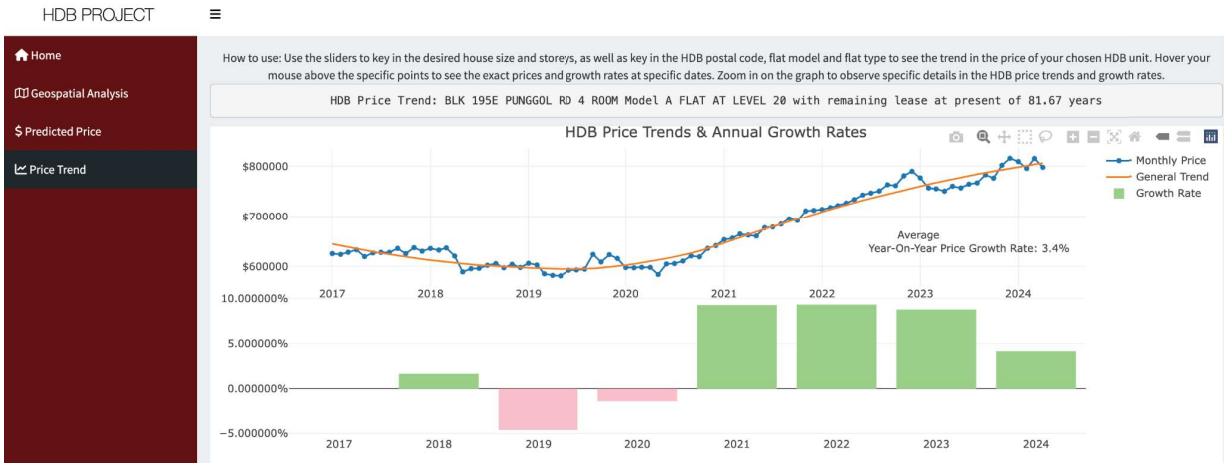
In the UI, a `div` element is styled to take up 50% of the width, centering content within the tab, and user instructions are dynamically generated by the server through the `output$intro` render function. The input components are designed to collect necessary data for the backend's gradient boosting model; numerical inputs like 'Square Meter' and 'Level' use slider inputs, while categorical data use select inputs. Correspondingly, on the server side, the `renderText` function is employed to display textual outputs such as user manual details and the address of the selected postal code, ensuring the user interface and server outputs are closely linked and responsive.

Data Processing and Prediction Execution:

Upon user submission, the UI's submit button triggers the price prediction process, updating outputs to display the predicted price in a modal or direct output area. Simultaneously, the server employs a `fittedPredict` function to retrieve the relevant dataset row from the 'laty' dataset based on the user-provided postal code. An `observeEvent` function is set up to process the inputs when the user submits details like postal code, flat model, and type. The data is pre-processed and fed into our machine learning model to predict the price using the expression `exp(predict(model, newdata))`. Despite the UI collecting only 6 inputs, the model requires 136, necessitating our lines of code used to work with the ML, linking these inputs effectively. The final predicted price and details such as address and remaining lease are then displayed in a pop-up modal.

E. Price Analysis Trend

1. Choice of visualisation



For the 'Price Analysis Trend' tab, we retained the format consistent with the 'Predicted Price' tab, while the visualisation takes precedence. In selecting the design elements, our aim was to mirror the input inspiration from the 'Predicted Price' tab, ensuring coherence across the application. The decision to place the trend graph prominently above the input fields was intentional. Users can also see the year-on-year price growth with the charts at the bottom. We prioritised the graphical display of HDB price trends to immediately catch the user's attention and anchor the analytical narrative. This strategic placement allows users to draw insights at a glance before proceeding to tailor the analysis using the input fields provided below.

2. Code explanation

In the 'Predicted Price' tab of our application, we start with an introductory text which provides basic instructions on how to utilise the forecasting features. Our user manual is displayed using `textOutput` in the UI. User inputs are then processed in the server tab and the resulting forecasted price information is displayed in a textual format via `verbatimTextOutput` after the user's submission. On the server side, the output generation begins with a `renderText` function that acts as a manual for first-time users, guiding them on how to navigate and use the forecasting features effectively. Additionally, our tab features a dynamic `Plotly` chart that visualises the price trends based on the user's inputs and the data from our predictive model, making the information both interactive and easily digestible.

On the server side, forecast event handling is managed by an observeEvent function that uses our fittedForecast function to retrieve datasets based on user-entered postal codes. This data, representing future months, is then processed by our sophisticated machine learning model. The model performs necessary transformations and predictions, seamlessly linking dynamic user inputs to predictive outputs, ensuring accurate and customised forecasts.

The dynamic forecast and price trend visualisation are handled by `renderPlotly`, which creates an interactive chart that updates in real-time, reflecting the predicted price trends based on user inputs. The `selectionMessage` function outputs a customised message, displaying the inputs selected by the users along with a crafted message to enhance user interaction. Additionally, `priceOutputF` displays a custom message summarising the details of the user-inputted HDB unit and its forecasted price trend. Finally, the `forecasted_prices` chart ensures that the data is correctly ordered by date before plotting, offering users a chronological view of the price trends. Through this structured integration of UI and server functionalities, our application ensures that users not only receive accurate and timely data-driven forecasts but also experience an intuitive and user-friendly interface that maximises engagement and utility.

References

Boehmke, B. (2019, October 19). *Chapter 12 Gradient Boosting | Hands-On Machine Learning with R*. Github.io. https://bradleyboehmke.github.io/HOML/gbm.html

Brownlee, J. (2016, September 11). *How to Configure the Gradient Boosting Algorithm*. Machine Learning Mastery. https://machinelearningmastery.com/configure-gradient-boosting-algorithm/

Hachcham, A. (2023). *XGBoost: Everything You Need to Know*. Neptune.ai. https://neptune.ai/blog/xgboost-everything-you-need-to-know

K. Haritha, S. Shailesh, Judy, M. V., Ravichandran, K. S., Raghunathan Krishankumar, & Gandomi, A. H. (2023). A novel neural network model with distributed evolutionary approach for big data classification. *Scientific Reports*, *13*(1). https://doi.org/10.1038/s41598-023-37540-z

LAM , N. Z. D. (2021). *UNDERSTANDING HDB RESALE PRICES IN SINGAPORE: A MACHINE LEARNING & ECONOMETRICS APPROACH*. https://scholarbank.nus.edu.sg/handle/10635/228209

Lawrence, R. (2023, September 1). *How to use Feature Importance with XGBoost*. EvolvingDev. https://www.evolvingdev.com/post/xgboost-model-feature-importance#Best-Practices-for-Feature-Selection-in-XGBoost

Mohtadi Ben Fraj. (2017, December 24). *In Depth: Parameter tuning for Gradient Boosting*. Medium; All things AI. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-gradient-boosting-3363992e9bae

Tseng, G. (2018, November 29). *Gradient Boosting and XGBoost*. Medium. https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5

Appendix:

Table 1: Source of datasets

Data	URL link
HDB Resale Flats Transactions (from 2017 onwards)	https://beta.data.gov.sg/collections/189/datasets/d_8b84c4ee58e3cfc0ece0d773c8ca6abc/view
MRT Stations	https://datamall.lta.gov.sg/content/datamall/en/static-data.html
Government Market Hawker Centers	https://beta.data.gov.sg/datasets/d_68a42f09f350881996d83f9cd73ab02f/view
Supermarkets	https://beta.data.gov.sg/datasets/d_11edd0117280c5776651d7891114c88c/view
Primary Schools (Web-scraped)	https://cloverhome.sg/primary-school-list/
Hospitals (Web-scraped)	https://en.wikipedia.org/wiki/List_of_hospitals_in_Singapore

Table 2: The strengths and limitations of XGBoost as compared to traditional GBM:

Aspect	**XGBoost**	**Traditional GBM**
Regularization	Incorporates L1 and L2 regularization to handle dataset with a large number of correlated variables, which is highly apt to our dataset.	May lack regularization techniques and therefore risk of inaccurate predictions due to the multicollinearity problem.
Tree Pruning	Offers advanced tree pruning techniques	Tree pruning may be less sophisticated
Feature Selection	Utilizes feature bagging for variable selection	May not have built-in feature selection methods
Complexity	More complex to understand and tune due to the larger number of hyperparameters, therefore risking inaccurate model predictions due to non-optimal choice of hyperparameters.	Simpler and more straightforward to use.

Sensitivity to Hyperparameters	Performance may be sensitive to hyperparameters	Hyperparameters may be easier to tune.
Overfitting with Large Datasets	May be prone to overfitting with large datasets. The challenge lies in trying to balance model complexity and regularization. As such, tuning is extremely important for XGBoost.	More robust to overfitting with large datasets.

Equation 1: the equation for LASSO regression

$$\min_{\beta} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 + \lambda \sum_{p=1}^{P} |\beta_p|$$