DSA3361 Group Project

Team 17

Contents

Introduction

The primary goal of this project is to make predictions on HDB resale price based on a number of factors. All data used in this project has been extracted from data.gov.sg.

To give a brief summary: Our analysis identified several critical factors influencing the resale prices of HDB flats in Singapore, namely location, floor area, and macroeconomic factors such as interest rate and consumer price index for housing & utilities.

Model Performance and Predictive Accuracy: Through comparative analysis, advanced machine learning models such as Random Forest and XGBoost demonstrated superior predictive performance over linear models with regularization like Lasso, Ridge, and Elastic Net. These models not only achieved higher accuracy but also effectively captured complex, non-linear relationships within the data, providing more reliable price estimates.

This study extends the existing discourse on HDB resale pricing by integrating advanced machine learning techniques alongside conventional linear models. By employing models like XGBoost and MLP, the research challenges the reliance on linear assumptions prevalent in previous studies, offering a more nuanced understanding of the factors affecting property prices. Additionally, we further consider potential large variations in data due to unforeseen events such as COVID-19. Data during and before the COVID-19 period was hence excluded from this study and only data from 2022 onwards was included for analysis and model training.

While the study offers valuable insights, it is subject to certain limitations. The analysis is constrained by the availability and granularity of data from data.gov.sg, potentially omitting influential variables such as buyer demographics and macroeconomic indicators. Furthermore, the models may exhibit biases inherent in the training data, affecting generalizability. Future research could address these limitations by incorporating a broader range of variables, utilizing real-time data streams, and exploring ensemble modeling techniques to enhance prediction robustness. Additionally, qualitative analyses could complement the quantitative findings, offering a more comprehensive understanding of market dynamics.

Data Preparation

Import Data

We explore resale flat prices (based on registration date) from 2022 to 2024 August. We arbitrarily set 2022 as a start date for this project as it is the period after COVID-19, thus allowing us to avoid large potential irregularities in data. We filter the dates for our dataset after joining of relevant data to be explored.

```
resale2017 <-
  read.csv("Resale flat prices based on registration date from Jan-2017 onwards.csv",
           stringsAsFactors = T)
```

Merging and Cleaning Data

```
# We first conduct preliminary data cleaning

resale_clean <- resale2017 %>%
  mutate(month = ym(month)) %>%
  mutate(remaining_lease = as.numeric(substr(remaining_lease, 1, 2)))

cat("Number of missing entries:",
    sum(is.na(resale_clean)), "\n")
```

```
## Number of missing entries: 0
```

```
cat("Number of duplicate entries:",
    sum(duplicated(resale_clean)), "\n")
```

```
## Number of duplicate entries: 359
```

We see that there are a number of duplicate entries in the data. However, these duplicate entries may be purely coincidental, with houses in the same block and storey range being sold for the same price. Ultimately, we decide to keep these entries as we have no grounds to remove them without further information (e.g. postal code and unit number to identify the exact house being sold).

Joining Data

From inspecting the data, we note some potentially important factors such as size, remaining lease years, location etc. in determining resale value. Before analyzing these factors, we pull in additional pieces of information which we have confidence that are important in determining HDB resale value as well, in order to build a better model to predict prices. These factors are CPI, and Bank Interest Rates (specifically, the Singapore Overnight Rate Average).

We first load in the relevant data from data.gov.sg.

```
interest <- read.csv("CurrentBanksInterestRatesEndOfPeriodMonthly.csv",
                     stringsAsFactors = T)
cpi <- read.csv("ConsumerPriceIndexCPI2019AsBaseYearMonthly.csv",
                stringsAsFactors = T)
```

We then cleaned and reformatted the data sets to allow for subsequent joining with HDB resale data.

```
interest_clean <- interest %>%
  filter(DataSeries == "Singapore Overnight Rate Average") %>%
  mutate(across(X2024Sep:X1988Jan, as.character))%>%
  mutate(across(X2024Sep:X1988Jan, as.numeric))%>%
  pivot_longer(X2024Sep:X1988Jan,names_to = "yrmonth", values_to="interest_rate") %>%
  mutate(yrmonth = str_replace(yrmonth, "X", "")) %>%
  mutate(yrmonth = ym(yrmonth))

cpi_clean <- cpi %>%
  filter(DataSeries == "    Housing & Utilities") %>%
  mutate(across(X2024Sep:X1961Jan, as.character)) %>%
  mutate(across(X2024Sep:X1961Jan, as.numeric)) %>%
  pivot_longer(X2024Sep:X1961Jan, names_to = "yrmonth", values_to = "CPI_housing") %>%
  mutate(yrmonth = str_replace(yrmonth, "X", "")) %>%
  mutate(yrmonth = ym(yrmonth))
```

Lastly, we join the data obtained based on interest and CPI with the HDB resale data, and filter for the relevant dates up to August 31.

```
resale_total <- resale_clean %>%
  left_join(interest_clean, by = c("month" = "yrmonth")) %>%
  left_join(cpi_clean, by = c("month" = "yrmonth")) %>%
  select(-DataSeries.x, -DataSeries.y)

resale_aug <- resale_total%>%
  filter(month >= "2022-01-01", month < "2024-09-01")
```

Exploratory Data Analysis

Numerical Variables

Resale price is a numerical response variable, and numerical predictor variables of interest include floor area, remaining lease, interest rate and CPI. We plot histograms of these variables to examine their distribution.

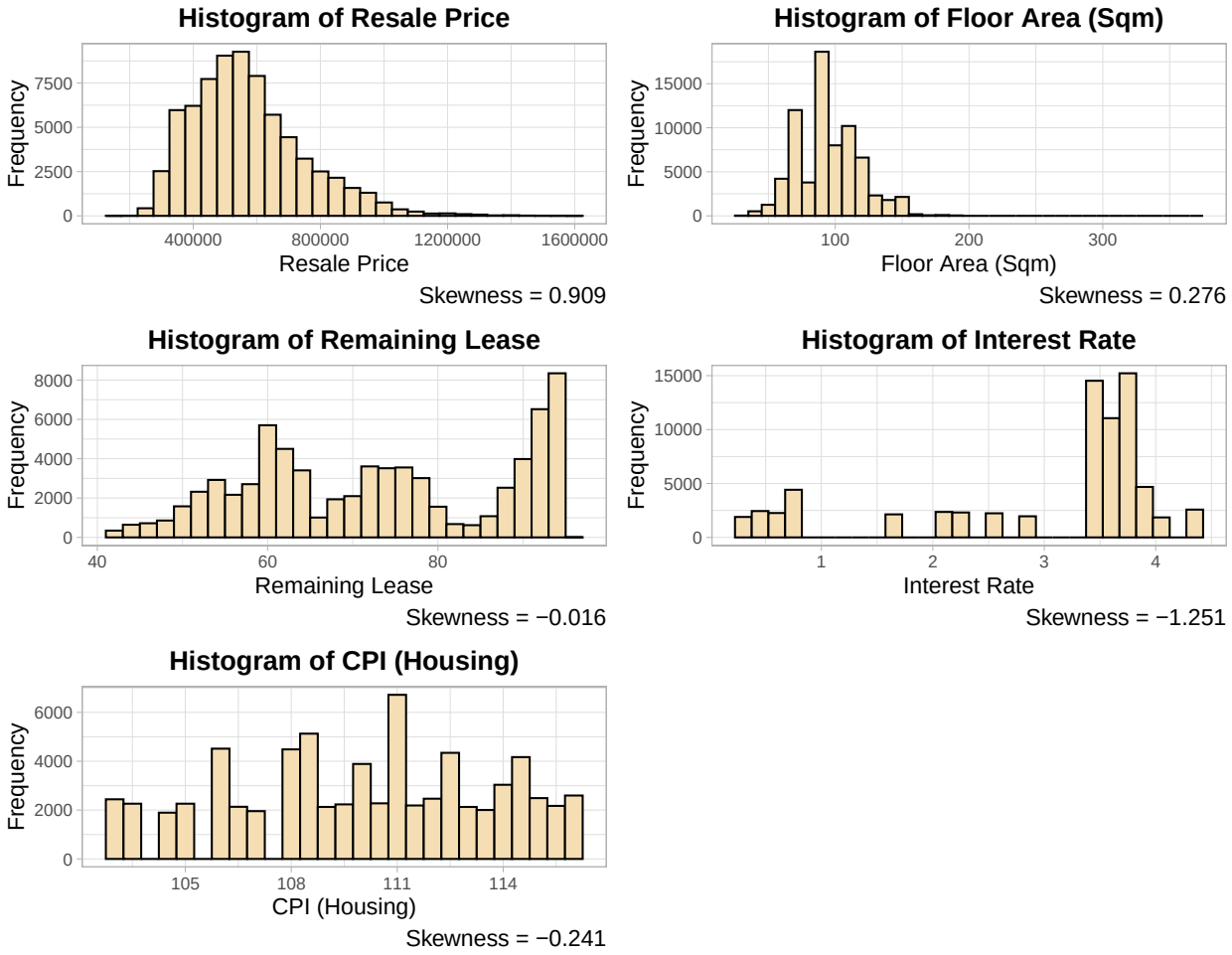
```
numvars <- data.frame(
  variables = c("resale_price", "floor_area_sqm", "remaining_lease",
                "interest_rate", "CPI_housing"),
  labels = c("Resale Price", "Floor Area (Sqm)", "Remaining Lease",
                    "Interest Rate", "CPI (Housing)"),
  binwidths = c(5*10^4, 10, 2, 0.15, 0.5)
)

numvars_plotlist <- list()

for (i in 1:nrow(numvars)) {
  numvars_plotlist[[i]] <-
    ggplot(resale_aug, aes_string(x = numvars[i, 1])) +
    geom_histogram(binwidth = numvars[i,3],
                   fill = "wheat", color = "black") +
    labs(x = numvars[i, 2], y = "Frequency",
```

```
        title = paste0("Histogram of ", numvars[i, 2]),
        caption = paste0("Skewness = ",
                         round(skewness(resale_aug[[numvars[i, 1]]]), 3))) +
    theme_light() +
    theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5),
          plot.caption = element_text(size = 12),
          axis.title = element_text(size = 12))
}

do.call(grid.arrange, c(numvars_plotlist, ncol = 2))
```



Resale price seems to be slightly right-skewed, with floor area following an approximately normal distribution. Remaining lease and CPI seem to have a multimodal distribution with multiple peaks. Interest rate is left-skewed, with the highest frequencies around the 3.5-4 range.

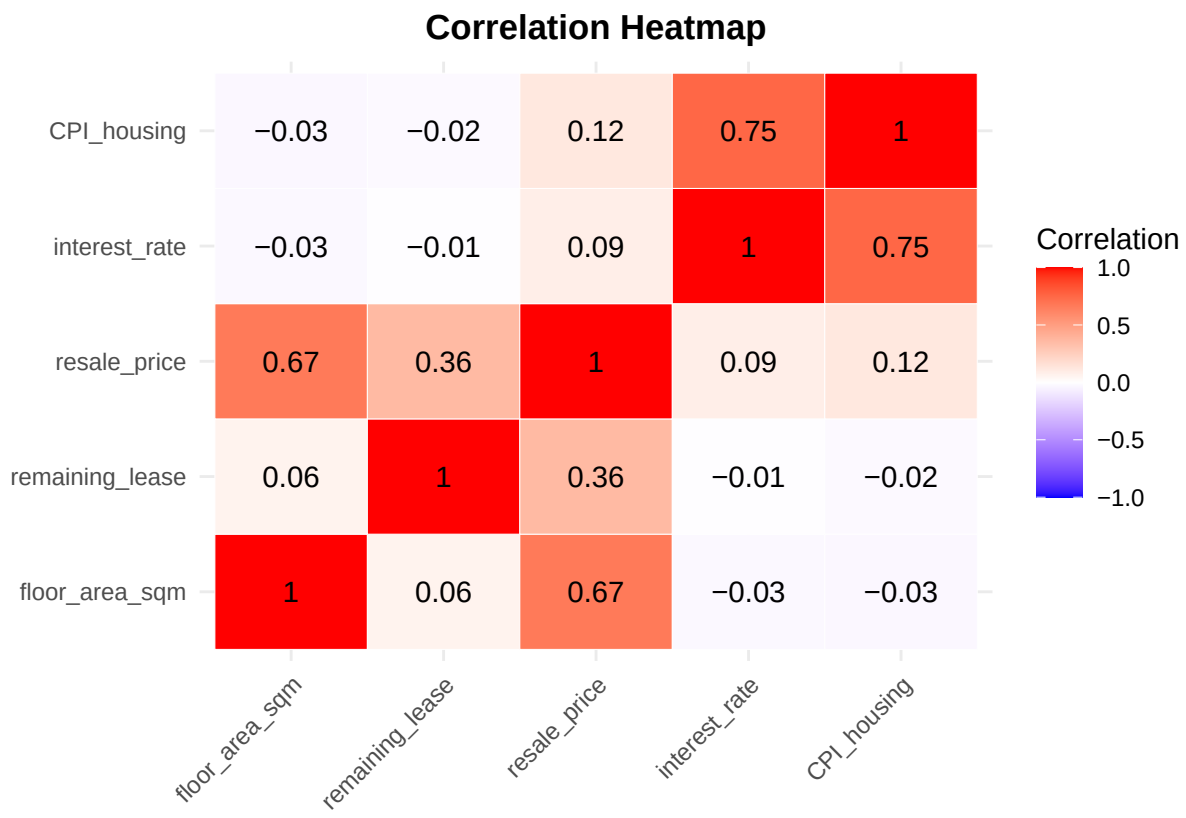
Next, we plotted a heatmap to visualize the correlation values between the numerical variables.

```
corr_matrix <- cor(resale_aug[,c(7,10:13)])

melted_corr <- melt(corr_matrix)

ggplot(data = melted_corr, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile(color = "white") +
```

```
  geom_text(aes(label = round(value, 2)), color = "black", size = 4) +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1, 1),
                       space = "Lab", name = "Correlation") +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold", hjust = 0.5),
        axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
  labs(title = "Correlation Heatmap", x = "", y = "")
```

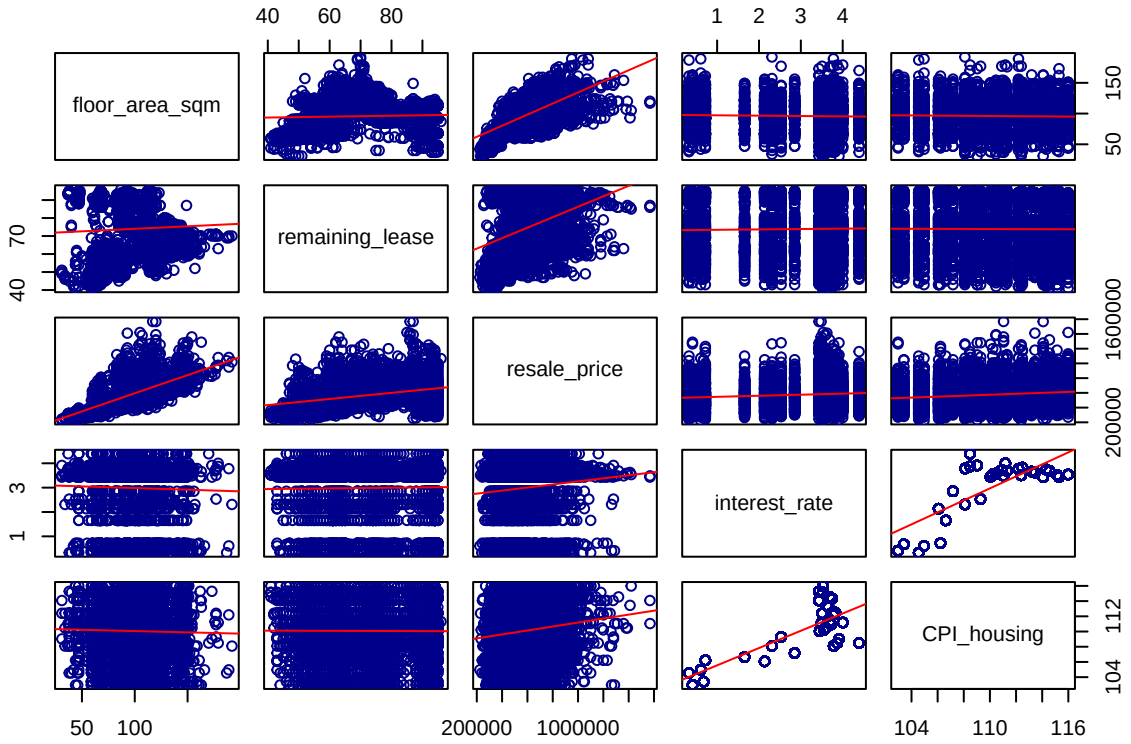


We see that there is a moderate positive correlation between the house size represented by floor_area_sqm and resale price, as well as a weak positive correlation between remaining lease and resale price. Interest rate and CPI also have a strong positive correlation, which could possibly lead to multicollinearity.

The correlations and relationship between the numerical variables can also be observed from the corresponding scatter plots.

```
scatter_with_line <- function(x, y) {
  points(x, y, col = "darkblue")
  abline(lm(y ~ x), col = "red")
}

set.seed(123)
pairs(sample_n(resale_aug[, c(7, 10:13)], 5000),
      panel = scatter_with_line)
```



Categorical Variables

We then explored categorical variables such as town, storey range, flat type and flat model.

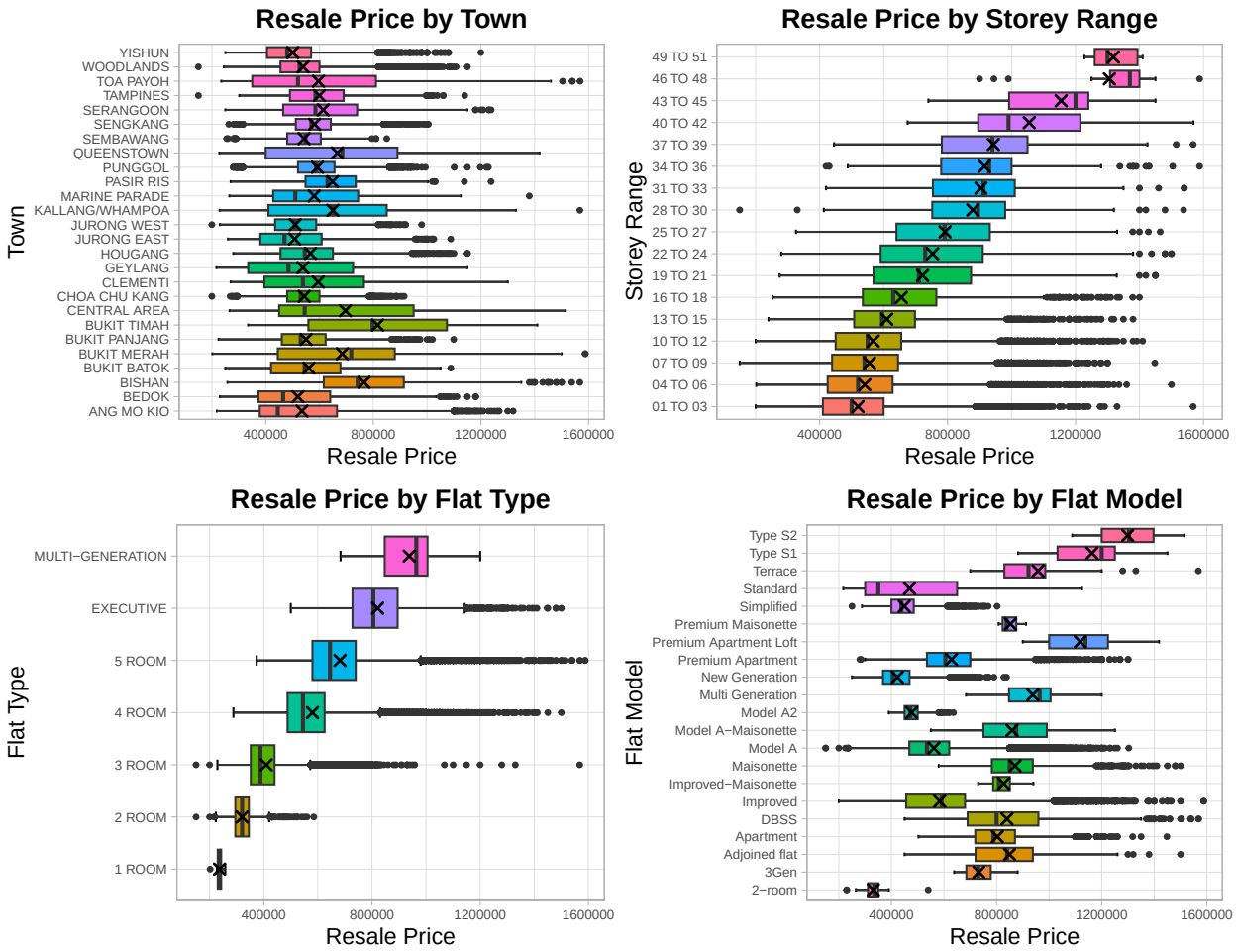
We plot boxplots to visualize how the distribution of resale price changes across the variables.

```
catvars <- c("town", "storey_range", "flat_type", "flat_model")
catvars_labs <- c("Town", "Storey Range", "Flat Type", "Flat Model")
catvars_plotlist <- list()

for (i in 1:length(catvars)) {
  catvars_plotlist[[i]] <-
    ggplot(resale_aug, aes_string(x = "resale_price",
                                  y = catvars[i])) +
    stat_boxplot(geom = 'errorbar', width = 0.2) +
    geom_boxplot(aes_string(fill = catvars[i]),
                 outlier.size = 1) +
    stat_summary(fun = "mean", shape = 4) +
    labs(x = "Resale Price", y = catvars_labs[i],
         title = paste0("Resale Price by ", catvars_labs[i])) +
    theme_light() +
    theme(plot.title = element_text(size = 14, face = "bold", hjust = 0.5),
          axis.title = element_text(size = 12),
          axis.text = element_text(size = 7),
          legend.position = "none")
```

```
}

do.call(grid.arrange, c(catvars_plotlist, ncol = 2))
```



We see that the distribution of resale price increases with a higher storey range, and a larger number of rooms under flat type. Its distribution also varies according to the town and flat model of the house. We should consider these 4 categorical variables to investigate if they improve the predictive power of our model. However, upon further inspection, we also see that flat_model and flat_type both contain the same category "Multi-Generation" with similar distributions, which we must address subsequently to prevent perfect multicollinearity.

Baseline Linear Model and Multicollinearity

We first build a baseline multiple linear regression model. However, to address the "Multi-Generation" category represented in both the flat_type and flat_model, we decide to drop one of the variables. We hence build a linear model with each of these variables to determine which variable should be dropped. Additionally, we remove additional factors that we deem insignificant or are not in use for our modelling, such as street name, block number, date, and lease commence date (which is already represented by remaining lease).

```
options(scipen = 999)
```

```
lm_flatmodel <- lm(resale_price ~. -month -block -street_name
                   -lease_commence_date -flat_type, resale_aug)
summary_flatmodel <- summary(lm_flatmodel)
adj_r_squared_flatmodel <- summary_flatmodel$adj.r.squared
cat("Adjusted R-squared for lm_flatmodel:",
    adj_r_squared_flatmodel, "\n")
```

```
## Adjusted R-squared for lm_flatmodel: 0.8856285
```

```
lm_flattype<- lm(resale_price ~. -month -block -street_name
                 -lease_commence_date -flat_model, resale_aug)
summary_flattype <- summary(lm_flattype)
adj_r_squared_flattype <- summary_flattype$adj.r.squared
cat("Adjusted R-squared for lm_flattype:",
    adj_r_squared_flattype, "\n")
```

```
## Adjusted R-squared for lm_flattype: 0.8688217
```

As the adjusted R-squared value for the model including flat_model is higher, we will choose lm_flatmodel as our linear model and drop the variable flat_type in our dataset instead.

```
resale_full<- resale_aug %>%
  select(-flat_type)
```

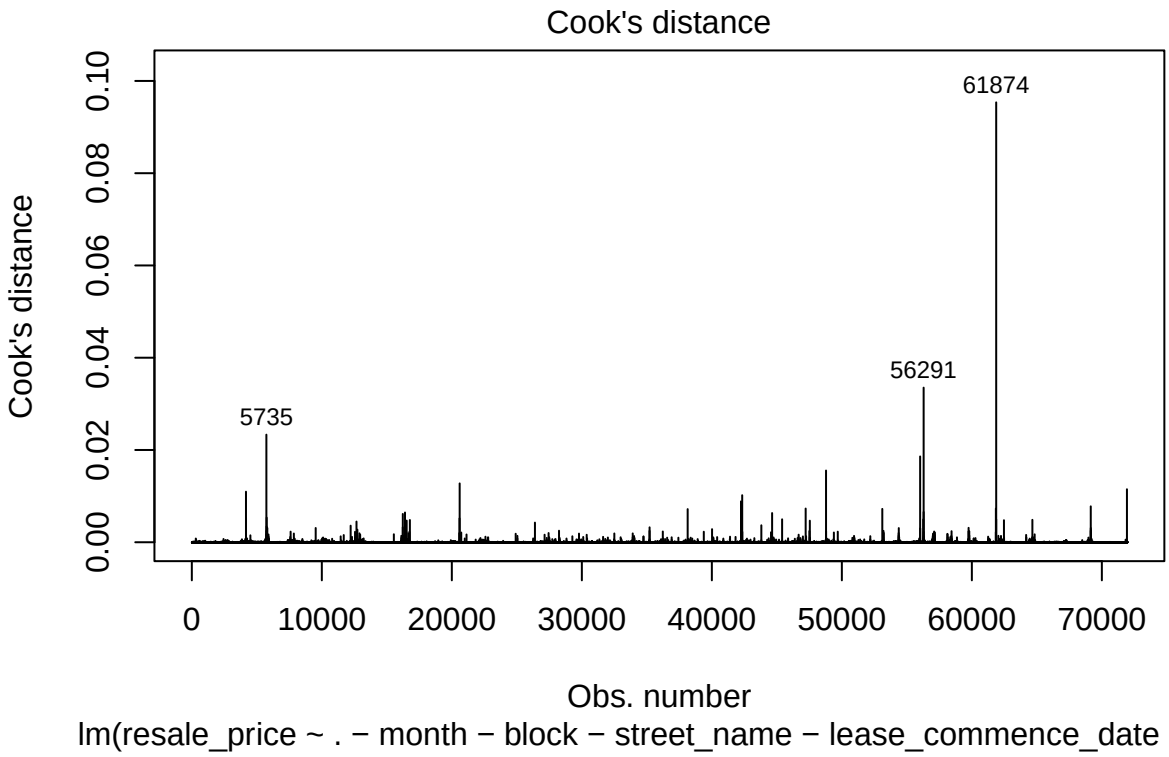
We move on to investigate if multicollinearity and influential points exist for the model of choice.

```
vif(lm_flatmodel)
```

```
##                      GVIF Df GVIF^(1/(2*Df))
## town             5.644847 25        1.035221
## storey_range     1.747614 16        1.017598
## floor_area_sqm   1.953489  1        1.397673
## flat_model      10.583933 20        1.060758
## remaining_lease  2.382367  1        1.543492
## interest_rate    2.305314  1        1.518326
## CPI_housing      2.310913  1        1.520169
```

We see that there is high multicollinearity for town and flat_model, hence suggesting that further manipulation must be done to optimize this model, which will be done through subsequent regularization.

```
plot(lm_flatmodel, which = 4)
```




```
cooksD <- cooks.distance(lm_flatmodel)
outlier <- cooksD > 4 / (length(cooksD)-(7+1))
table(outlier)
```

```
## outlier
## FALSE  TRUE
## 68023  3908
```

```
resale_noout <- resale_full %>%
  mutate(CooksD = cooksD, Outlier = outlier) %>%
  filter(Outlier == F)

filtered_noout <- resale_noout %>% select(-month,-block,-street_name
                  ,-lease_commence_date,-CooksD,-Outlier)

lm_no_outliers <- lm(resale_price ~., filtered_noout)
summary_noout <- summary(lm_no_outliers)
adj_r_squared_noout <- summary_noout$adj.r.squared

cat("Adjusted R-squared for lm_no_outliers:",
    adj_r_squared_noout, "\n")
```

```
## Adjusted R-squared for lm_no_outliers: 0.9021004
```

After calculating the Cook's distance, while there were no points that were above a distance of 0.5, there were 3908 observations with Cook's distance greater than 4/n-(p+1), suggesting that these points may potentially

be outliers. Nonetheless, the Cook's distance values observed were generally low, all under 0.1. Removal of these points allowed for a marginal improvement of the adjusted R-squared value (from 0.886 to 0.902). Since Cook's distances are generally low and removing influential points did not lead to a largely significant increase in model performance, we decided to retain them in the subsequent analysis.

Models

In this section, we build a Multiple Linear Regression model, and incorporate regularization techniques to reduce overfitting, and address multicollinearity. The performance of these models will subsequently be evaluated.

MLR & Regularization Methods

We split the data for our subsequent models into train test sets. For our models, we only remove the non-critical values such as month, block, street name, and lease date, as done previously.

```
set.seed(123)
rate <- 0.8
train.size <- round(nrow(resale_full) * rate)
sample <- sample(nrow(resale_full), train.size)

training_full <- resale_full[sample, ]
training <- training_full %>%
  select(-month, -block, -street_name, -lease_commence_date)

test_full <- resale_full[-sample, ]
test <- test_full %>%
  select(-month, -block, -street_name, -lease_commence_date)
```

MLR

```
lm_model <- lm(resale_price~., data=training)
```

Regularization Methods

Converting models into train and test matrices, and splitting to predictors and target values

```
train.x <- model.matrix(resale_price ~ ., data = training)[, -1]
train.y <- training$resale_price
test.x  <- model.matrix(resale_price ~ ., data = test)[, -1]
test.y  <- test$resale_price
```

Ridge Regression

Using cross validation to get lambda min and training the ridge regression model

```
cv_ridge <- cv.glmnet(train.x, train.y, alpha = 0, type.measure = "mse")
glm_Ridge <- glmnet(train.x, train.y, alpha = 0, lambda = cv_ridge$lambda.min)
```

LASSO Regression

Using cross validation to get lambda min and training the lasso regression model

```
cv_LASSO <- cv.glmnet(train.x, train.y, alpha = 1, type.measure = "mse")
glm_LASSO <- glmnet(train.x, train.y, alpha = 1, lambda = cv_LASSO$lambda.min)
```

Elastic Net Regression

Define a function that runs cross validation on a specified alpha.

```
generate_cvmodels <- function (x) {
  set.seed(123)
  return(cv.glmnet(train.x, train.y,
                   type.measure = "mse", alpha = x/10))
}
```

Running the above function 11 times from alpha = 0 to 1 in intervals of 0.1.

```
cv_models <- lapply(0:10, generate_cvmodels)
# Generate cross-validation mses for all 11 models
(cv_error <- unlist(lapply(cv_models,
                           function(x) x$cvm[x$lambda == x$lambda.min] )))
```

```
##  [1] 3944967327 3625672892 3626595698 3626687586 3627490972 3627218802
##  [7] 3627057100 3626889460 3626856246 3626830830 3626937957
```

```
# Return smallest alpha
(which(cv_error == min(cv_error))-1)/10
```

```
## [1] 0.1
```

```
get_best_model <- function (models, errors) {
  best_n <- which(errors == min(errors))
  return(
    data.frame(
      alpha = (best_n - 1)/10,
      lambda = models[[best_n]]$lambda.min,
      CV_error = errors[best_n]
    )
  )
}

(best_parameter <- get_best_model(cv_models, cv_error))
```

```
##   alpha   lambda   CV_error
## 1   0.1 249.9513 3625672892
```

With these parameters, we build our Elastic Net Regression model

```
glm_ElaNet <- glmnet(train.x,
                     train.y,
                     alpha = best_parameter$alpha,
                     lambda = best_parameter$lambda)
```

Model Evaluation

Defining the helper function for evaluating models based on various categories

```
eval_results <- function(fit, true) {
  actual <- data.matrix(true)
  RSS <- sum((actual - fit)^2)
  TSS <- sum((actual - mean(actual))^2)
  R_square <- 1 - RSS / TSS
  n<-length(fit)
  p<-7
  Adj_Rsquare<-1 - ((1 - R_square)*(n - 1)/(n - p - 1))
  data.frame(
    MSE = MSE(fit, true),
    MAE = MAE(fit, true),
    RMSE = RMSE(fit, true),
    MAPE = MAPE(fit, true),
    R2 = R_square,
    Adj_R2 = Adj_Rsquare
  )
}
```

We evaluate the performance of our models on both train and test data.

```
# Base MLR
fit_lin_train <- predict(lm_model, newdata = training)
true_lin_train <- training$resale_price
summary_linear_train <- eval_results(fit_lin_train, true_lin_train)

fit_lin_test <- predict(lm_model,newdata = test)
true_lin_test <- test$resale_price
summary_linear_test <- eval_results(fit_lin_test,true_lin_test)

# Regularization Models
true_train <- train.y
true_test <- test.y

# Ridge
fit_Ridge_train <- predict(glm_Ridge, train.x)
summary_Ridge_train <- eval_results(fit_Ridge_train, true_train)
fit_Ridge_test <- predict(glm_Ridge, test.x)
summary_Ridge_test <- eval_results(fit_Ridge_test, true_test)

# LASSO
fit_LASSO_train <- predict(glm_LASSO, train.x)
```

```
summary_LASSO_train <- eval_results(fit_LASSO_train, true_train)
fit_LASSO_test <- predict(glm_LASSO, test.x)
summary_LASSO_test <- eval_results(fit_LASSO_test, true_test)

# ElaNet
fit_ElaNet_train <- predict(glm_ElaNet, train.x)
summary_ElaNet_train <- eval_results(fit_ElaNet_train, true_train)
fit_ElaNet_test <- predict(glm_ElaNet, test.x)
summary_ElaNet_test <- eval_results(fit_ElaNet_test, true_test)
```

We create a summary table to evaluate how our 4 models perform across the various error metrics, for both the train and test data sets.

```
summary_trainperf <- rbind(summary_linear_train, summary_Ridge_train,
                           summary_LASSO_train, summary_ElaNet_train)

rownames(summary_trainperf) <- c("Linear_train", "Ridge_train",
                                 "LASSO_train", "ElaNet_train")

knitr::kable(summary_trainperf, digits = 3)
```

	MSE	MAE	RMSE	MAPE	R2	Adj_R2
Linear_train	3606603803	45123.95	60055.01	0.083	0.886	0.886
Ridge_train	3928932836	45631.12	62681.20	0.081	0.876	0.876
LASSO_train	3609604220	45099.03	60079.98	0.083	0.886	0.886
ElaNet_train	3610276105	45057.85	60085.57	0.083	0.886	0.886

```
summary_testperf <- rbind(summary_linear_test, summary_Ridge_test,
                          summary_LASSO_test, summary_ElaNet_test)

rownames(summary_testperf) <- c("Linear_test", "Ridge_test",
                                "LASSO_test", "ElaNet_test")

knitr::kable(summary_testperf, digits = 3)
```

	MSE	MAE	RMSE	MAPE	R2	Adj_R2
Linear_test	3544977446	44787.44	59539.71	0.083	0.885	0.885
Ridge_test	3852325824	45284.08	62067.11	0.081	0.875	0.875
LASSO_test	3547308121	44774.86	59559.28	0.083	0.885	0.885
ElaNet_test	3548074737	44738.87	59565.72	0.083	0.885	0.885

Surprisingly, all the models perform better on the unseen test data, with lower MSE, MAE and RSME, and similar MAPE and R2 values. The base linear model also seems to be comparable to the regularized linear models, albeit with slightly higher error values. We further analyze our coefficients for the Elastic Net Regression model.

```
coef(glm_ElaNet)
```

```
## 66 x 1 sparse Matrix of class "dgCMatrix"
##                                          s0
## (Intercept)                     -1225974.057
## townBEDOK                         -20210.104
## townBISHAN                        105008.644
## townBUKIT BATOK                   -97737.900
## townBUKIT MERAH                    99853.201
## townBUKIT PANJANG                -133819.758
## townBUKIT TIMAH                   202717.473
## townCENTRAL AREA                  102867.024
## townCHOA CHU KANG                -170248.855
## townCLEMENTI                       20697.654
## townGEYLANG                        44414.235
## townHOUGANG                       -78541.412
## townJURONG EAST                   -93286.822
## townJURONG WEST                  -143365.702
## townKALLANG/WHAMPOA                65044.770
## townMARINE PARADE                 121913.017
## townPASIR RIS                     -99015.556
## townPUNGGOL                      -123405.826
## townQUEENSTOWN                     97301.822
## townSEMBAWANG                    -156009.211
## townSENGKANG                     -133421.585
## townSERANGOON                      8410.529
## townTAMPINES                      -47296.183
## townTOA PAYOH                      60565.309
## townWOODLANDS                    -151309.177
## townYISHUN                       -113501.854
## storey_range04 TO 06               17790.274
## storey_range07 TO 09               32584.136
## storey_range10 TO 12               40180.997
## storey_range13 TO 15               51923.094
## storey_range16 TO 18               71773.644
## storey_range19 TO 21              103865.618
## storey_range22 TO 24              119042.423
## storey_range25 TO 27              134311.221
## storey_range28 TO 30              167608.378
## storey_range31 TO 33              184673.380
## storey_range34 TO 36              194887.603
## storey_range37 TO 39              206742.146
## storey_range40 TO 42              237781.503
## storey_range43 TO 45              260109.752
## storey_range46 TO 48              314733.957
## storey_range49 TO 51              265074.697
## floor_area_sqm                      5364.222
## flat_model3Gen                    -54162.689
## flat_modelAdjoined flat            55495.983
## flat_modelApartment                29206.332
## flat_modelDBSS                    130892.537
## flat_modelImproved                -11851.325
## flat_modelImproved-Maisonette     143379.688
## flat_modelMaisonette               74539.348
## flat_modelModel A                 -13270.935
## flat_modelModel A-Maisonette       99205.989
```

```
## flat_modelModel A2                   2206.539
## flat_modelMulti Generation          86098.581
## flat_modelNew Generation            12310.337
## flat_modelPremium Apartment          7197.400
## flat_modelPremium Apartment Loft   180939.742
## flat_modelPremium Maisonette        25144.440
## flat_modelSimplified                31535.395
## flat_modelStandard                  -5751.150
## flat_modelTerrace                  294142.053
## flat_modelType S1                  246987.314
## flat_modelType S2                  315845.186
## remaining_lease                      5529.500
## interest_rate                         736.754
## CPI_housing                          8250.095
```

We see that all of the coefficients are non-zero and Elastic Net regression does not perform any feature selection, indicating that all of the coefficients contribute to the predictive power of the model.

Thus far, we have explored various models, using regularization techniques to optimize the linear regression model. We have found that both Lasso and Elastic Net regression models performed best in their predictive power, with the lowest MSE, MAE and RMSE as well as high R2 values. In the following sections, we employed machine learning models to further increase the accuracy of our predictions.

Machine Learning Models

We now explore various machine learning models, including XGBoost and Random Forest models to further explore better models for prediction of hdb resale values. For this section, all code will be in python to allow for easier modelling with python machine learning packages.

Importing modules

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import *
import xgboost as xgb
```

Importing Original dataset for training

```
df = pd.read_csv('cleandata.csv')
```

Encoding Categorical data

```
storey_range_order = ['01 TO 03', '04 TO 06', '07 TO 09', '10 TO 12',
                      '13 TO 15', '16 TO 18', '19 TO 21', '22 TO 24',
                     '25 TO 27', '28 TO 30', '31 TO 33', '34 TO 36',
                      '37 TO 39', '40 TO 42', '43 TO 45', '46 TO 48',
                      '49 TO 51']

ordinal_encoder = OrdinalEncoder(categories=[storey_range_order])
df["storey_range"] = ordinal_encoder.fit_transform(df[["storey_range"]])
df_encoded = pd.get_dummies(df, columns=['town', 'flat_model'])
```

Train-test split

```
X = df_encoded.drop(['resale_price'], axis=1)
y = df_encoded['resale_price']
y = y.values.reshape(-1, 1)  # Reshape y to be of shape (num_samples, 1)


features = X.columns
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

XGBoost (Training the Model)

```
xgb_reg = xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=100000,
    learning_rate=0.002,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.65,
    random_state=3361,
    early_stopping_rounds=100,
    booster='gbtree',
    reg_lambda=0.5,
    reg_alpha=0.2,
    tree_method = "approx"
)
```

```
xgb_reg.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    verbose=10000
)
```

```
[0] validation_0-rmse:598584.81978
[10000] validation_0-rmse:42017.33425
[20000] validation_0-rmse:38490.26364
[30000] validation_0-rmse:37238.83077
[40000] validation_0-rmse:36579.25237
[50000] validation_0-rmse:36165.22004
[60000] validation_0-rmse:35888.24127
[70000] validation_0-rmse:35684.65580
[80000] validation_0-rmse:35541.05812
[90000] validation_0-rmse:35431.45099
[97139] validation_0-rmse:35372.25405
```

```
booster = xgb_reg.get_booster()

importance_dict = booster.get_score(importance_type='gain')

importance_df = pd.DataFrame({
    'Feature': list(importance_dict.keys()),
    'Importance': list(importance_dict.values())
})

importance_df = importance_df.sort_values(by='Importance', ascending=False).reset_index(drop=True)

print(importance_df.head(6))
```

```
                          Feature    Importance
0                town_CENTRAL AREA  3.619646e+11
1                  flat_model_DBSS  3.451659e+11
2             flat_model_Maisonette  3.308009e+11
```

```
3                    town_BUKIT MERAH  2.937989e+11
4                     town_QUEENSTOWN  2.818452e+11
5                      floor_area_sqm  2.806709e+11
```

Looking at the top 6 most important features as identified by XGBoost, we see that several factors such as location represented by town, flat model, and floor area are all highly important in determining resale value

Testing the Model (XGBoost)

```
y_pred_boost = xgb_reg.predict(X_test)

n, p = len(y_test), X_test.shape[1]

mse_boost = mean_squared_error(y_test, y_pred_boost)
mae_boost = mean_absolute_error(y_test, y_pred_boost)
r2_boost = r2_score(y_test, y_pred_boost)
adjusted_r2_boost = 1 - (1 - r2_boost) * ((n - 1) / (n - p - 1))

print(f'Mean Squared Error: {mse_boost}')
print(f'Mean Absolute Error: {mae_boost}')
print(f'R^2 Score: {r2_boost}')
print(f'Adjusted R^2 Score: {adjusted_r2_boost}')
```

```
Mean Squared Error: 1251188643.2887862
Mean Absolute Error: 24727.601734508582
R^2 Score: 0.960118062166001
Adjusted R^2 Score: 0.9599733809348466
```

Random Forest (Training the Model)

```
rf = RandomForestRegressor(n_estimators=800, random_state=42)
```

```
rf.fit(X_train, y_train)
```

```
importances = rf.feature_importances_

feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
})

feature_importance_df = feature_importance_df.sort_values(by='Importance',
                                                          ascending=False).reset_index(drop=True)
print(feature_importance_df.head(6))
```

```
                        Feature  Importance
0                  floor_area_sqm    0.535589
1                    storey_range    0.114303
```

```
2                    remaining_lease    0.084310
3                        CPI_housing    0.030980
4                   town_BUKIT MERAH    0.025407
5                town_KALLANG/WHAMPOA    0.016594
```

Again looking at the top 6 most important features as identified by our random forest model, we see that similar factors are important, including floor area, storey range, remaining lease, and CPI

Testing the Model (Random Forest)

```
y_pred_rf = rf.predict(X_test)

mse_rf = mean_squared_error(y_test, y_pred_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
adjusted_r2_rf = 1 - (1 - r2_rf) * ((n - 1) / (n - p - 1))

print(f'Mean Squared Error: {mse_rf}')
print(f'Mean Absolute Error: {mae_rf}')
print(f'R^2 Score: {r2_rf}')
print(f'Adjusted R^2 Score: {adjusted_r2_rf}')
```

```
Mean Squared Error: 1627641989.1742074
Mean Absolute Error: 27551.06237509467
R^2 Score: 0.9481185215543316
Adjusted R^2 Score: 0.947930309130781
```

Comparing the evaluation metrics between XGBoost and Random Forest, we see that XGBoost has the highest Adj R2 value and lowest error metrics across all of the models explored so far, with an Adj R2 value of 0.960. We hence use XGBoost as our model of choice for testing on unseen data from September onwards

Predicting with project_testdata

Project Test Data loading and preprocessing

```
df2 = pd.read_csv('test_clean_resale.csv')
#where test_clean_resale contains data from September onwards with n=200
```

```
df2["storey_range"] = ordinal_encoder.fit_transform(df2[["storey_range"]])

df2_encoded = pd.get_dummies(df2, columns=['town', 'flat_model'])

X2 = df2_encoded.drop(
    ['month', 'block', 'street_name', 'lease_commence_date', 'resale_price'],
    axis=1)
```

```
missing_cols = set(features) - set(X2.columns)

for col in missing_cols:
    X2[col] = 0
X2_aligned = X2[features]
```

```
y2 = df2_encoded['resale_price']
y2 = y2.values.reshape(-1, 1)
```

Testing on XGBoost

```
xgb_test_pred = xgb_reg.predict(X2_aligned)

n, p = len(y2), X2_aligned.shape[1]

mse_xgb_test = mean_squared_error(y2, xgb_test_pred)
mae_xgb_test = mean_absolute_error(y2, xgb_test_pred)
r2_xgb_test = r2_score(y2, xgb_test_pred)
adjusted_r2_xgb_test = 1 - (1 - r2_xgb_test) * ((n - 1) / (n - p - 1))

print(f'Mean Squared Error: {mse_xgb_test}')
print(f'Mean Absolute Error: {mae_xgb_test}')
print(f'R^2 Score: {r2_xgb_test}')
print(f'Adjusted R^2 Score: {adjusted_r2_xgb_test}')
```

```
Mean Squared Error: 1796043504.134453
Mean Absolute Error: 32722.4746875
R^2 Score: 0.9486681192425697
Adjusted R^2 Score: 0.9305099029202133
```

We see that XGBoost maintains a considerably high Adj R2 value and low error metrics on our project test dataset, achieving high accuracy and demonstrating high predictive power as compared to the other models thus far.