DSA3361 Team Project

Contents

Authors: Phua Zhuo Wei, Brandon Owen Sjarif, Han Zimeng, Jonathan Mohnish Umaibalan, Tan Chuan Hee

Date Compiled: 2024-11-15

Introduction

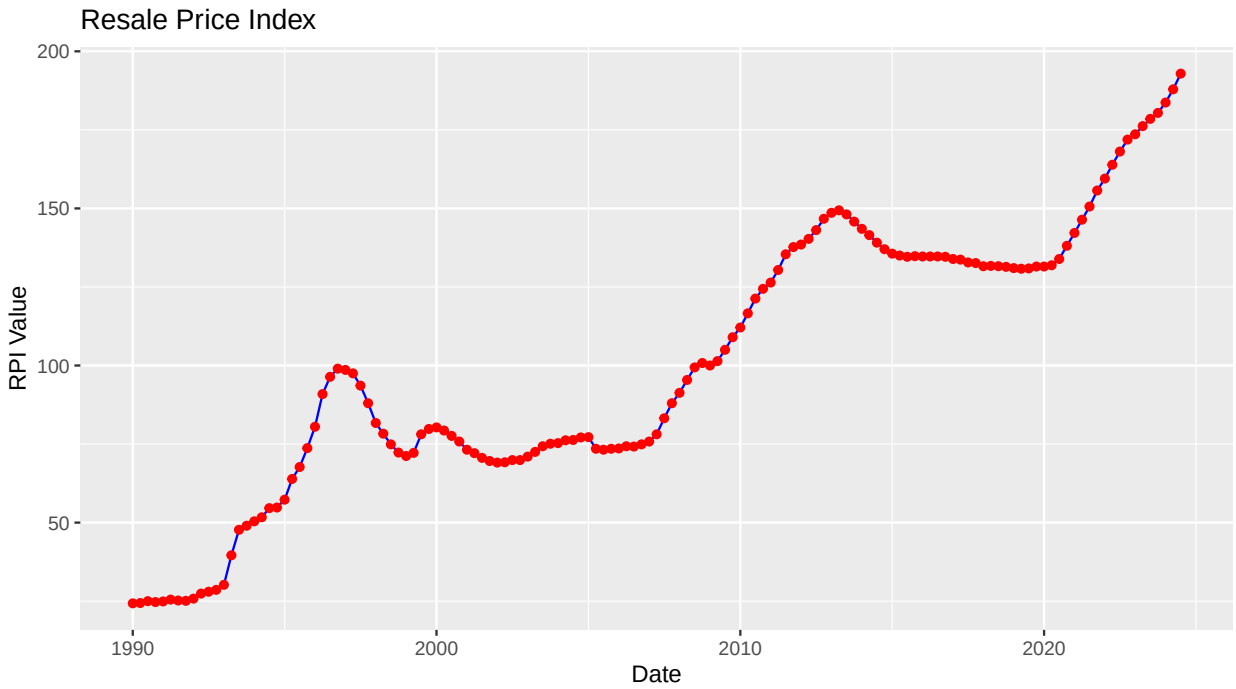
For this project, we analyzed the factors contributing to the resale prices of HDB Flats in Singapore. In October 2024, HDB data concluded that 8035 resale flats have been sold in just the third quarter alone, which is an increase of 9.3% (7352) from the previous quarter. [1] This trend shows no sign of slowing down and hence, emphasizing the importance of identifying key factors that drive sales. We will be using data publicly available from the data.gov website and only considering data before 30th August 2024. We have also included the Resale Price Index into the model, which helps normalise the data. Through identifying these factors, we aim to build a predictive model which predicts resale HDB prices for the future.

What is RPI

To account for supply and demand, inflations, market economy, policy changes throughout the years, we have decided to use an index called the Resale Price Index for resale HDBs in Singapore. This index reflects the resale price overall movement throughout the years and will be used to help normalise the resale prices to reflect relative costs independent of the general markets.

[1] https://www.businesstimes.com.sg/property/hdb-resale-prices-2-7-q3-volumes-also-surpass-expectations

```r
# resale price index (Base period is 2009 1st Quarter)

rpi <- read.csv("resalepriceindex.csv")
rpi <- rpi %>%
  rename(rpi_value = index) %>%
  mutate(date = as.Date(as.yearqtr(quarter, format = "%Y-Q%q",)),
         index = row_number())

rpi_graph <- ggplot(rpi, aes(x = date, y = rpi_value)) +
  geom_line(color = "blue") +
  geom_point(color = 'red') +
  labs(title = "Resale Price Index",
       x = "Date",
       y = "RPI Value")

rpi_graph
```



Based on the RPI graph plotted above, we have decided to omit data before 2021-Q1 as the graph shows that the rpi before that is unstable and possibly due to COVID-19, the resale prices plateaued and slightly decreased.

Importing Data

For this report, we have decided to input additional entries such as the Latitude and longitude. Furthermore, we would also like to include predictors such as nearest MRT and distance to nearest MRT. As we believe that transport and distance would be a crucial factor to predict prices of HDBs.

To do so, we have created a python script to help extract the coordinates based on the block and street name of the data. After obtaining the latitude and longitude, we can now search for the nearest MRT station and the distance to this MRT station. Detailed information about these scripts can be found in the

supplementary section at the end of the report. After obtaining these predictors, we can now import this data to conduct futher analysis.

```
df <- read.csv('HDBresale.csv', stringsAsFactors = T)
sum(is.na(df))
```

```
## [1] 0
```

```
sum(duplicated(df))
```

```
## [1] 0
```

```
nrow(df)
```

```
## [1] 191480
```

```
# Transforming data to be > 2021 and < 2024-08-30
df_2021 <- df %>% filter(year(month) >= 2021)

df_2021 <- df_2021 %>%
  mutate(quarter = paste0(year(month), "-Q", quarter(month)))

df_2021 <- df_2021 %>% filter(as.Date(month) <= as.Date('2024-08-30'))
```

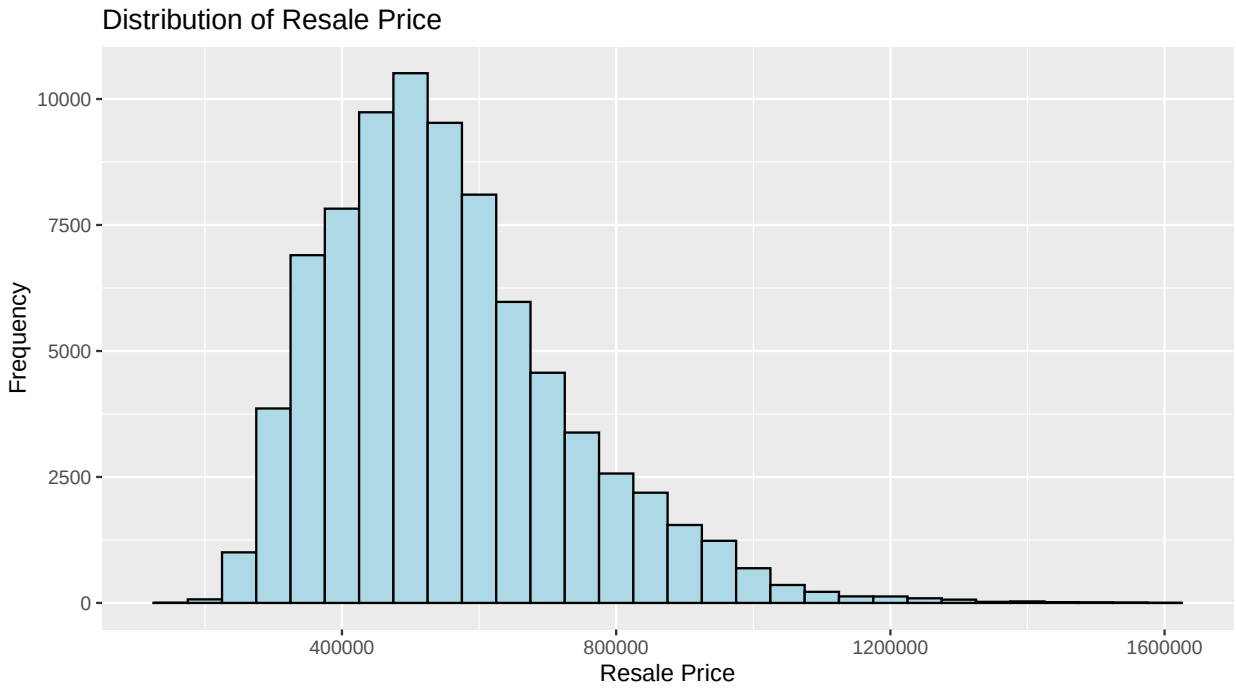
For training the model, we have decided to set a 80/20 training and validation split to evaluate the results on both the training and the validation set.

```
# inserting rpi into base data
set.seed(124)

df_rpi <- df_2021 %>%
  left_join(rpi, by = "quarter") %>%
  dplyr::select(month, town, flat_type, storey_range, floor_area_sqm, flat_model,
         resale_price, nearest_mrt, distance_to_mrt, mrt_type, remaining_lease, rpi_value)

df_rpi <- df_rpi %>% mutate(normalised_price = df_rpi$resale_price/df_rpi$rpi_value)

# Splitting Data into training and testing 80/20
index <- sample(x=nrow(df_rpi), size=nrow(df_rpi)*.8)
index <- sort(index)

train <- df_rpi[index,]
val <- df_rpi[-index,]
```

Exploratory Data Analysis

Resale Price Distribution plots

```r
ggplot(train, aes(x = resale_price)) +
  geom_histogram(binwidth = 50000, fill = "lightblue", color = "black") +
  labs(title = "Distribution of Resale Price", x = "Resale Price", y = "Frequency")
```



Based on the plot of distribution of resale price, we can conclude that resale price is normally distributed and right skewed.

Heat Mapping

```r
singapore_map <- st_read("HDBExistingBuilding.geojson")
```

```
## Reading layer 'HDBExistingBuilding' from data source
##   '/Users/zhuowei/Desktop/University_Studies/Y4S1/DSA3361/HDBExistingBuilding.geojson'
##   using driver 'GeoJSON'
## Simple feature collection with 12847 features and 2 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY, XYZ
## Bounding box:  xmin: 103.6848 ymin: 1.270005 xmax: 103.989 ymax: 1.457245
## z_range:       zmin: 0 zmax: 0
## Geodetic CRS:  WGS 84
```

```r
mrt_coords <- read.csv("mrt_locations.csv")

lrt_coords <- mrt_coords %>% filter(type == "LRT")
mrt_coords <- mrt_coords %>% filter(type == "MRT")

# Please view in HTML for full interactive experience
df$resale_price_scaled <- rescale(df$resale_price, to = c(0, 1))
```

```r
map <- leaflet(df) %>%
  addTiles() %>%
  addProviderTiles(providers$CartoDB.DarkMatter) %>%
  addHeatmap(lng = ~longitude, lat = ~latitude, intensity = ~resale_price, blur = 20, radius = 15) %>%
  addCircleMarkers(
    data = mrt_coords, lng = ~longitude, lat = ~latitude, radius = 5, color = "blue", fill = TRUE, fill(
  addCircleMarkers(
    data = lrt_coords, lng = ~longitude, lat = ~latitude, radius = 5, color = "green", fill = TRUE, fil
  setView(lng = 103.8198, lat = 1.3521, zoom = 12)

map # interactive map to be viewed in HTML only
```

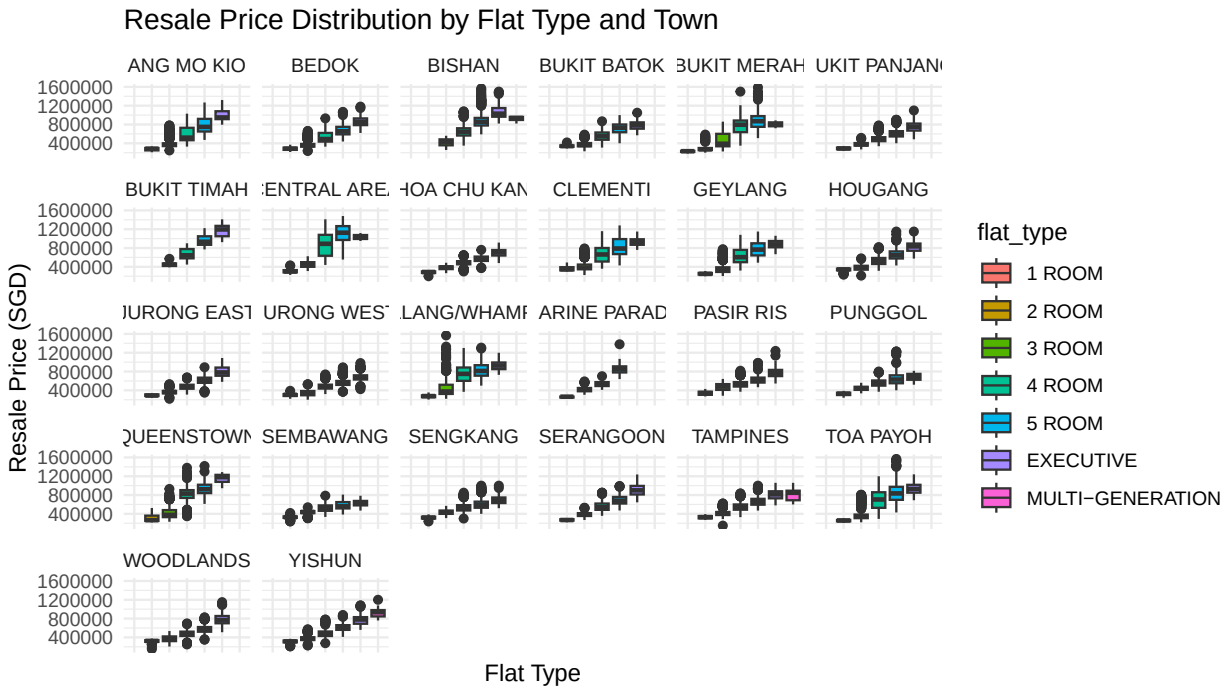

```
#
# # Screenshot of the map for PDF
```

```
# saveWidget(map, "sg_map.html", selfcontained = TRUE)
# webshot("sg_map.html", file = "map.png")
```

Blue circles denotes MRT, green denotes LRTs. From this heat map, we can observe that the places which are densely populated with MRTs, such as the central and south of Singaporem, usually have higher resale prices than that of other places.

Resale Price distribution by flat type and town

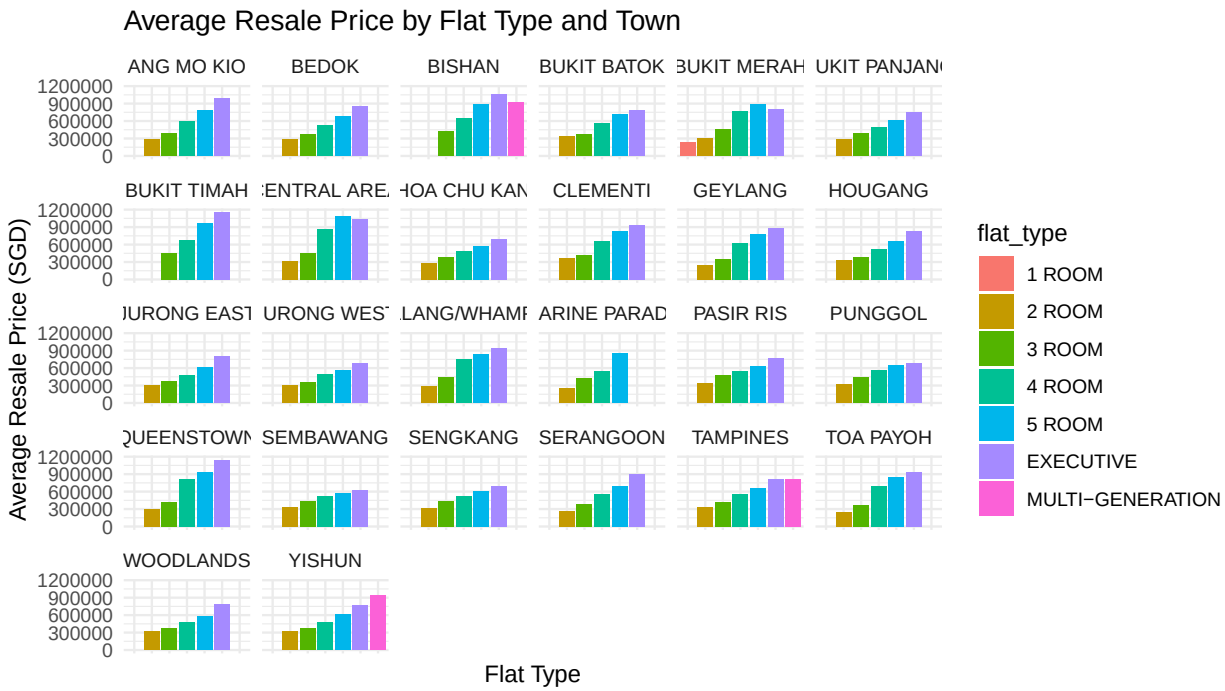
```
ggplot(train, aes(x = flat_type, y = resale_price, fill = flat_type)) +
  geom_boxplot() +
  facet_wrap(~town) +
  labs(title = "Resale Price Distribution by Flat Type and Town",
       x = "Flat Type",
       y = "Resale Price (SGD)") +
  theme_minimal() +
  theme(
    axis.text.x = element_blank()
  )
```



From these plots, we can see that most towns actually have similar price ranges for the different types of flats. However, it is notable that the price fluctuates more in Bishan, Bukit Merah, Central Area, Kallang/Whampoa , Queenstown and Toa Payoh. All of which are close to City Center/Central Business District (CBD), mature estate with limited land supply and have greater accessibility to major transport modes.

Average resale price by flat type and town

```
ggplot(train, aes(x = flat_type, y = resale_price, fill = flat_type)) +
  stat_summary(fun = "mean", geom = "bar", position = "dodge") +
  facet_wrap(~town) +
  labs(title = "Average Resale Price by Flat Type and Town",
       x = "Flat Type",
       y = "Average Resale Price (SGD)") +
  theme_minimal() +
  theme(
    axis.text.x = element_blank()
  )
```

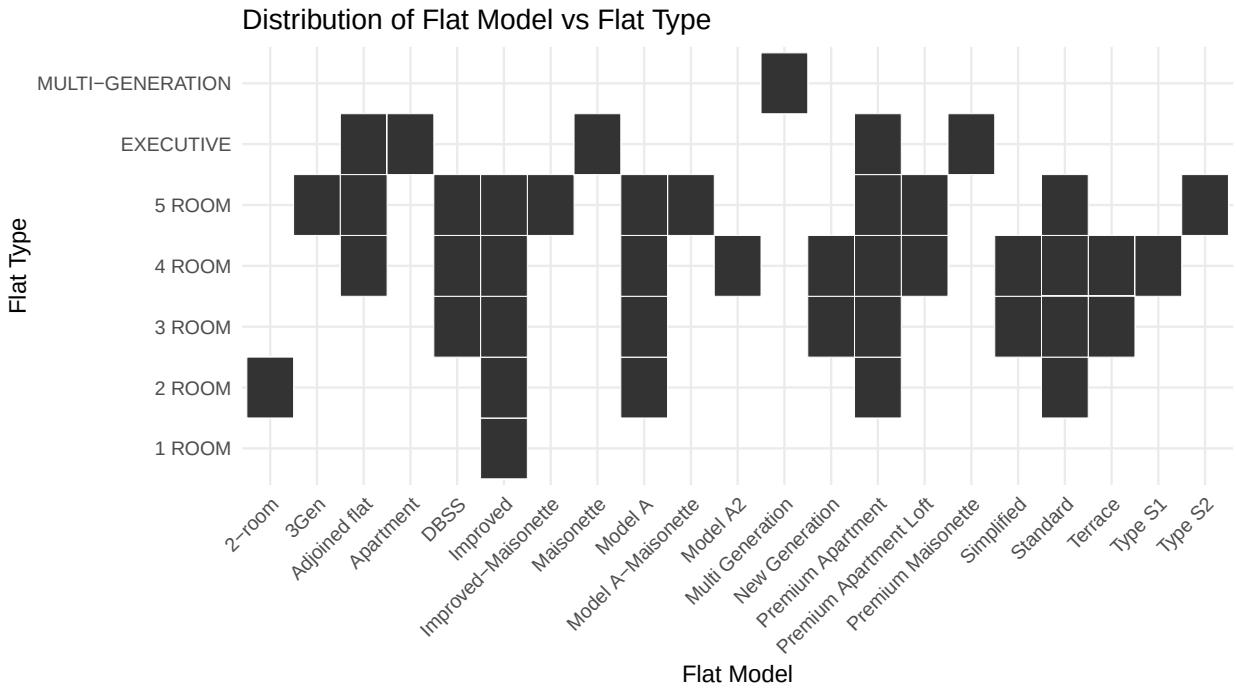


To futher break it down, by plotting average resale price of each flat type and town, we can conclude that prices for each upgraded version of flat type becomes more expensive than their previous ones. The only exception to this is in Bishan, where the price of a multi generation flat is lower than an executive one. In Bukit Merah, where the price of executive is lower than the 5 room flat. Central Area, where prices of executives are cheaper than 5 room ones. This trend can be explained due to the old estates in Bishan and Bukit Merah. Due to how old most multi-generation and executive flats there, the newer executives/5 room flats could potentially be more expensive than the remaining multi-generation/executive ones. For the central area, executive flats could be newer than the 5 room ones resulting in a higher price for the executive flats.

Distribution of flat model vs flat type

```
ggplot(train, aes(x = flat_model, y = flat_type)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "lightblue", high = "darkblue") +
  labs(title = "Distribution of Flat Model vs Flat Type", x = "Flat Model", y = "Flat Type") +
```

```
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

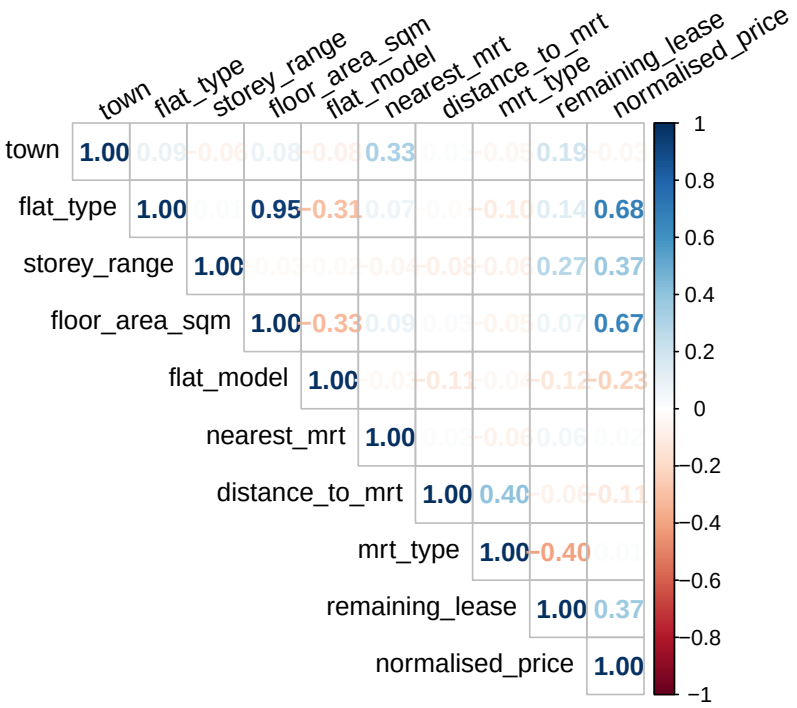


From the flat type vs flat model plot, we can conclude that most flat models have 3, 4 and 5 room flats with the exception of special ones such as Apartment, Maisonette, Model A2, Multi-Generation, Premium Maisonetter, Type S1 and Type S2.

Correlation Plot

To further understand how each predictor may affect other predictors, we decided to plot a correlation plot.

```
df_cor <- train %>%
  mutate_if(is.factor, as.numeric) %>%
  dplyr::select(-rpi_value, -resale_price, -month)

cor_stuff <- cor(df_cor)
corrplot(cor_stuff,
         method = "number", type = "upper",
         tl.col = "black", tl.srt = 30)
```



From this, we observe that the predictors flat_type and floor_area_sqm are highly positively correlated.

Floor_area_sqm, flat_type, remaining_lease and storey_range are also moderately positively correlated to the normalised price. We have decided to combine both flat_type and flat_model as we believe that these 2 factors can be categorized as one and to remove the correlation between flat_type and floor_area_sqm

```
train <- train %>% mutate(flat_combined = paste((train$flat_model), (train$flat_type), sep = ' '))
train$flat_combined <- as.factor(train$flat_combined)
```

RPI prediction model

We first created a prediction model for RPI. As observed in the RPI distribution plot above, the RPI value from 2021 was linear thus, we decided to use a simple linear regression model to predict the upcoming RPI values in the following financial quarters.

```
# Data only filtered from 2021 as trend before that seems to be affected from COVID-19 which presented
rpi_filtered <- rpi %>% filter(date >= as.Date(as.yearqtr("2021-Q1", format = "%Y-Q%q")) &
                      date < as.Date(as.yearqtr("2024-Q3", format = "%Y-Q%q")))

lm_rpi <- lm(rpi_value ~ index, data=rpi_filtered)
summary(lm_rpi)
```

```
##
## Call:
## lm(formula = rpi_value ~ index, data = rpi_filtered)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.72000 -1.75907  0.05714  1.33038  3.15538
##
```

```
## Coefficients:
##              Estimate Std. Error t value         Pr(>|t|)
## (Intercept) -280.5196    17.3486  -16.17 0.00000000164216 ***
## index          3.4035     0.1319   25.81 0.00000000000698 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.989 on 12 degrees of freedom
## Multiple R-squared:  0.9823, Adjusted R-squared:  0.9808
## F-statistic: 666.2 on 1 and 12 DF,  p-value: 0.000000000006977
```

The Adjusted R-squared values is 0.9808, which suggests that this model is capable of explaining 98.08% of the variance in the response variable. Hence, we will use this rpi simple linear regression model as part of the data preparation, to predict the rpi value in a certain financial quarter and also normalise the resale prices.

Multi-Linear Regression Model

Next, we decided to create a baseline multi-linear regression model with log(normalised_price) as the response variable to better represent the distribution of price.

```
names(train)
```

```
##  [1] "month"            "town"             "flat_type"        "storey_range"
##  [5] "floor_area_sqm"   "flat_model"       "resale_price"     "nearest_mrt"
##  [9] "distance_to_mrt"  "mrt_type"         "remaining_lease"  "rpi_value"
## [13] "normalised_price" "flat_combined"
```

```
train_mlr <- train %>% select(-month, -resale_price, -rpi_value, -flat_type, -flat_model)
lm_df <- lm(log(normalised_price) ~ . , data = train_mlr)
options(max.print = 10000)
lm_summary <- summary(lm_df)
# lm_summary #Commented out as it taking up too many pages

lm_coef <- as.data.frame(lm_summary$coefficients)
high_pvalue_predictors <- lm_coef[lm_coef$`Pr(>|t|)` > 0.05, ]
nrow(high_pvalue_predictors)
```

```
## [1] 32
```

```
#high_pvalue_predictors #uncomment to see which predictors are not included
```

When observing the summary of the model, most of the predictors have a p value of < 0.05, which suggests that it is significant to the model and helps the model with its predicting power. However, we also realised that there were 32 predictors which are not significant. Despite this, we still decided to keep these predictors as we wanted to give a more accurate representation of all the towns and nearest_mrt. If we were to remove these predictors, we would not be able to predict data where these factors are present. Thus, the model was kept the same.

```r
# Helper functions
# Eval Data function
eval_results <- function(fit, true) {
  actual <- data.matrix(true)
  SSE <- sum((actual - fit)^2)
  SST <- sum((actual - mean(actual))^2)
  R_square <- 1 - SSE / SST
  data.frame(
    MSE = MSE(fit, true),
    MAE = MAE(fit, true),
    RMSE = RMSE(fit, true),
    MAPE = MAPE(fit, true),
    R2 = R_square
  )
}

# RPI index value function to run in model
rpi_index <- function(data, month_col) {
  data %>%
    mutate(
      year = year(!!sym(month_col)),
      month = as.Date(!!sym(month_col)),
      quarter = quarter(month),
      index = 1 + (4 * (year - 1990)) + (quarter - 1)
    ) %>%
    select(index)
}
```

Evaluating MLR model

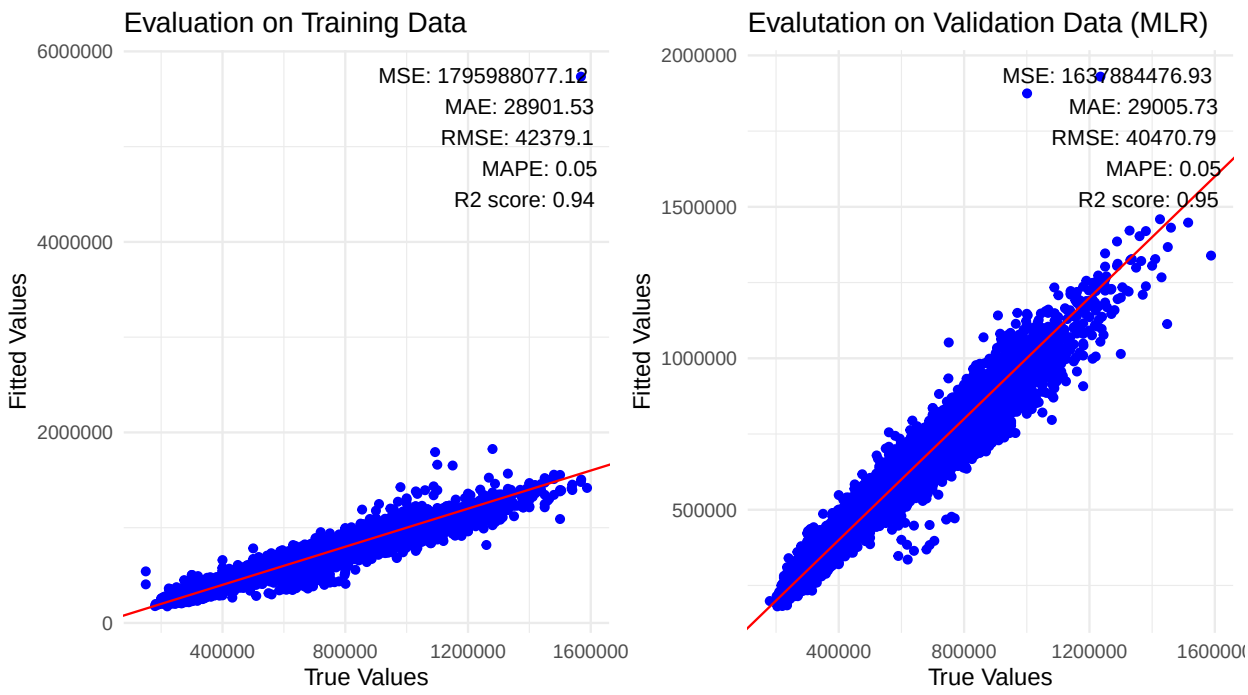
We then evaluated the model on the training and validation data.

```r
# Evaluate model on training data
train2 <- train %>% select(-rpi_value, -normalised_price)
train2_index <- rpi_index(train2, "month")
train2_rpi <- predict(lm_rpi, train2_index)
train2$normalised_price <- train2$resale_price/train2_rpi

train2_mlr <- train2 %>%
  dplyr::select(town, storey_range, floor_area_sqm, nearest_mrt, distance_to_mrt, mrt_type, remaining_l

train2_fit_mlr <- exp(predict(lm_df, train2_mlr)) * train2_rpi
train2_eval <- eval_results(train2_fit_mlr, train$resale_price)

eval_text <- function(eval_data) {
  paste0(
    "MSE: ", round(eval_data$MSE, 2), "\n",
    "MAE: ", round(eval_data$MAE, 2), "\n",
    "RMSE: ", round(eval_data$RMSE, 2), "\n",
    "MAPE: ", round(eval_data$MAPE, 2), "\n",
    "R2 score: ", round(eval_data$R2, 2), "\n"
  )
}
```

```r
train2_eval_text <- eval_text(train2_eval)

train2_eval_data <- data.frame(fit = train2_fit_mlr,
                               true = train$resale_price)

plot_evaluation <- function(data, x_var, y_var, text_label, plot_title) {
  ggplot(data, aes(x = {{ x_var }}, y = {{ y_var }})) +
    geom_point(color = 'blue') +
    geom_abline(slope = 1, intercept = 0,
                linetype = "solid", color = "red") +
    labs(
      title = plot_title,
      x = "True Values",
      y = "Fitted Values"
    ) +
    annotate("text", x = Inf, y = Inf, label = text_label,
             hjust = 1.1, vjust = 1.1, size = 3.5, color = "black",
             parse = FALSE) +
    theme_minimal()
}
train2_eval_plot <- plot_evaluation(train2_eval_data, true, fit, train2_eval_text, 'Evaluation on Train
```

```r
# Evaluate model on validation data
val_index <- rpi_index(val, "month")

val <- val %>% mutate(flat_combined = paste(as.character(val$flat_model), as.character(val$flat_type), s
val$flat_combined <- as.factor(val$flat_combined)

# Predicting rpi value
val_rpi <- predict(lm_rpi, val_index)
true <- val$resale_price

val_mlr <- val %>%
  select(town, storey_range, floor_area_sqm, nearest_mrt, distance_to_mrt, mrt_type, remaining_lease, n

val_fit_mlr <- exp(predict(lm_df, val_mlr)) * val_rpi
val_eval <- eval_results(val_fit_mlr, true)

val_eval_text <- eval_text(val_eval)
val_eval_data <- data.frame(fit = val_fit_mlr,
                            true = true)
val_eval_plot <- plot_evaluation(val_eval_data, true, fit, val_eval_text, 'Evalutation on Validation Da
```

```r
# Both plots together
grid.arrange(train2_eval_plot, val_eval_plot, nrow = 1)
```

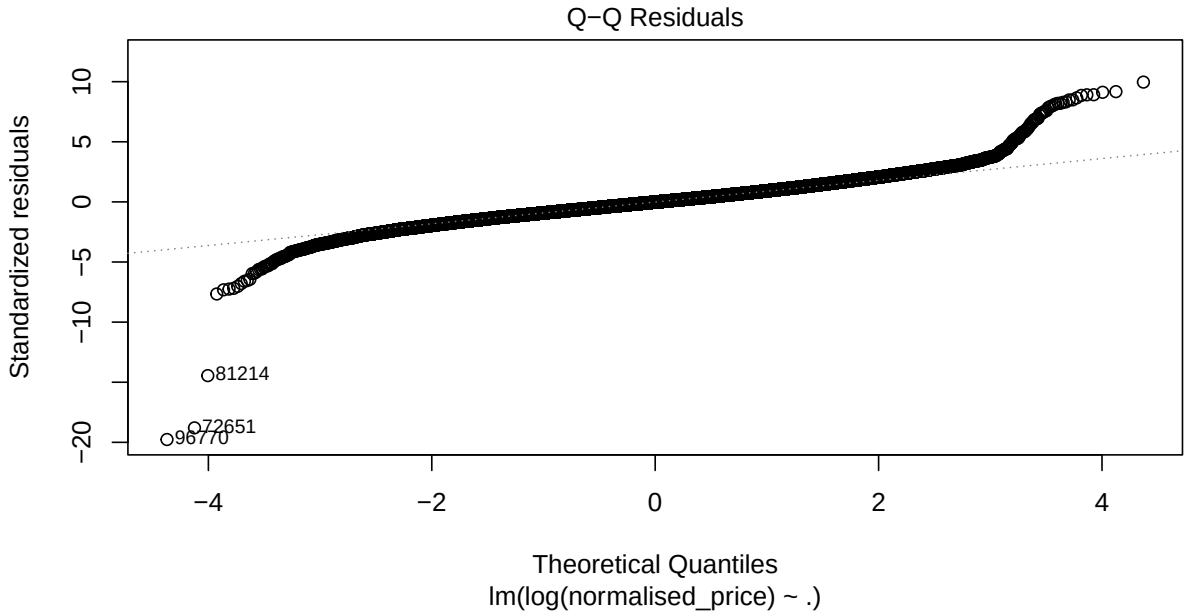


Based on these plots, we observe that the MSE for training is 1.80e^9 while the MSE for the validation set is 1.64e^9. This was unexpected as models tend to perform better in training compared to validation. However, when the curves of fitted vs true price were plotted, this could be explained due to the large anomaly in the training set, which caused the MSE value in the training set to be higher than that of the validation set. This is further supported as we compare the MAE for both, where the MAE for training is lower than that of the validation set, suggesting that the model's average prediction error is smaller for most training samples, but the training set's higher MSE is disproportionately affected by the extreme outlier.

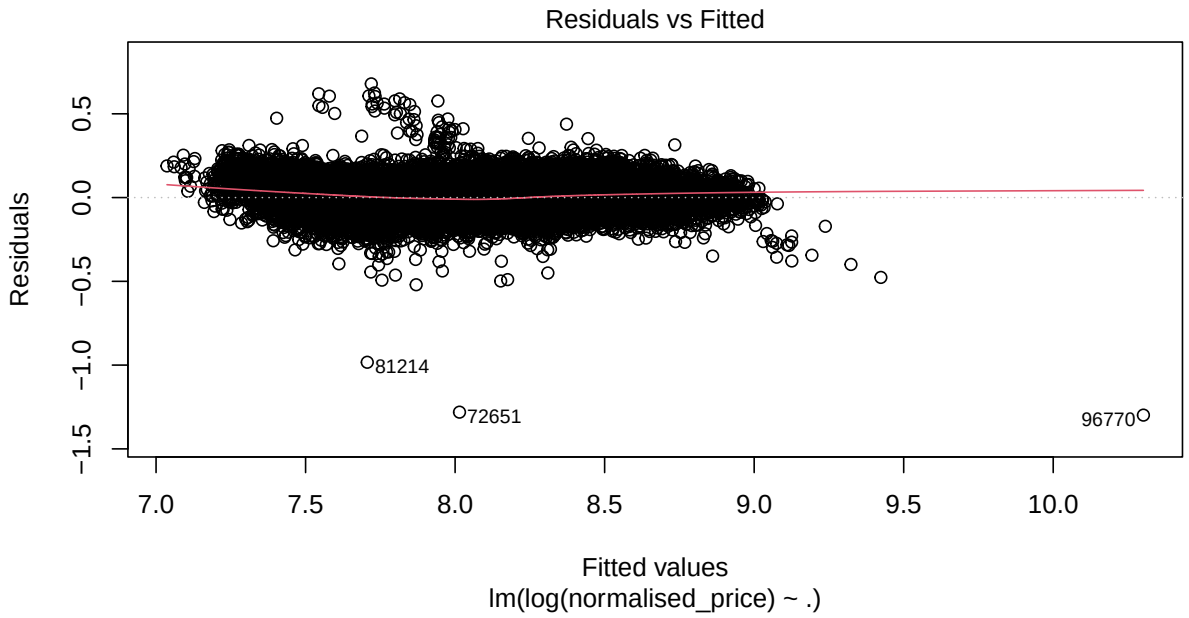
Analysing model

Normality and Homoscedasticity plots were plotted to analyse the residuals in the model

```
# Normality
plot(lm_df, which=2)
```




```
# Homoscedasticity
plot(lm_df, which=1)
```




```
# vif(lm_df) # errors encountered
# alias(lm_df) # uncomment to see alias predictors.
```

Based on the normality plot, we observe that there are some points far away from the line and for homoscedascity, there were also some residuals which had significant error distance. Thus, to further improve

this model, we will be looking at the cook's distance to determine if there are any outliers and possibly remove these from the model.

VIF function was also used to attempt to analyze any multicollinear predictors. However, when we ran VIF we experienced aliasing problems and by through further investigations by running alias(lm_df), we found out that most of these alias predictors were betwen nearest_mrt and flat_combined. Hence, we concluded that we will not be removing these as we would prefer a more generalised model with minimal decrease in accuracy compared to a model that is accurate but unable to be used due to limited categories.

Cooks Distance

We also applied a cook's distance threshold of 4/(n-(p+1)) to extract any outliers in this model.

```
cd <- cooks.distance(lm_df)
outlier <- cd > 4 / (length(cd) - (2+1))
table(outlier)
```

```
## outlier
## FALSE  TRUE
## 76812  3945
```

```
outlierdf <- train %>% mutate(CooksD = cd, Outlier = outlier)
train_new <- outlierdf[outlierdf$Outlier == FALSE, ] %>% select(-CooksD, -Outlier)
```

Based on a cook's distance threshold is set at 4/(n-(p+1)), we observe that there were 3945 outliers. However, upon closer inspection of these points, we realised that many of them were 'special'. Such as special room types, or located in central areas such as esplanade MRT. Hence, we have decided not to remove these points to further preserve the accuracy if these locations were to be present in the testing dataset. Thus, we have decided to stick with the baseline multi-linear regression model with log(normalised_price) as the response variable.

Regularisation Models

Next, we decided to use regularisation techniques such as lasso, ridge and elastic net, to build models and compare the results of these models with that of the muli-linear regression model.

Data preparation

Before building the models, we had to prepare the data in matrices

```
train_true <- train$resale_price
val_true <- val$resale_price

traindf <- train %>% dplyr::select(-flat_type, -flat_model, -resale_price, -rpi_value, -month)
valdf <- val %>% dplyr::select(-resale_price, -rpi_value, -flat_model, -flat_type, -month)

# Dummy encoding predictors to ensure all categorical variables are present in validation set
dummy <- dummyVars(" ~ .", data = traindf %>% select(-normalised_price))

# Apply to train and test data to ensure consistent columns
```

```
train.x <- predict(dummy, newdata = traindf %>% select(-normalised_price))
val.x <- predict(dummy, newdata = valdf %>% select(-normalised_price))
train.y <- log(traindf$normalised_price)
val.y <- log(valdf$normalised_price)
```

Lasso

We first built the lasso regression model, attempting to reduce the number of predictor variables which do not contribute to the predicting power of the model.

```
cv_lasso <- cv.glmnet(train.x, train.y, alpha = 1, nfolds=10)
lasso <- glmnet(train.x, train.y, alpha = 1, lambda = cv_lasso$lambda.min)

train_lasso <- exp(predict(lasso, train.x)) * train$rpi_value
val_lasso <- exp(predict(lasso, val.x)) * val$rpi_value


lasso_coefficients <- coef(lasso)

# Convert the coefficients to a matrix and filter for non-zero values
non_zero_indices <- which(lasso_coefficients != 0)
non_zero_coefficients <- lasso_coefficients[non_zero_indices]
non_zero_variables <- rownames(lasso_coefficients)[non_zero_indices]

sum(lasso$beta != 0)
```

```
## [1] 228
```

```
total_variables <- length(coef(lm_df))-1
total_variables
```

```
## [1] 236
```

```
cv_lasso$lambda.min
```

```
## [1] 0.00006079153
```

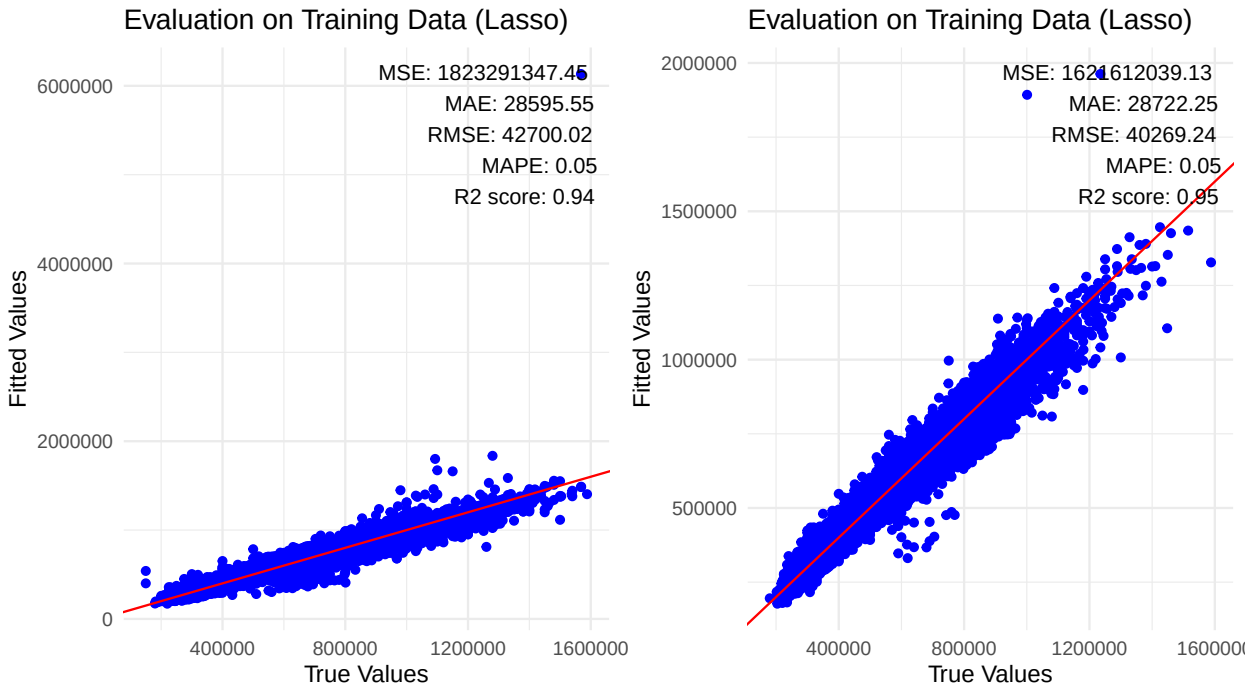
```
#non_zero_variables # uncomment to see non-zero variables
```

After running the lasso regularisation, there is a total of 228 variables left, from the base of 236. The optimal parameter lambda was found to be 0.0000608 and this was used to build the lasso regression model. Prediction on the training and validation sets were then conducted.

Evaluating Lasso Model

We then evaluated the lasso regression model on the training and the validation data.

```
lasso_train_eval <- eval_results(train_lasso, train_true)
lasso_val_eval <- eval_results(val_lasso, val_true)

lasso_train_eval_data <- data.frame(fit = as.vector(train_lasso),
                                    true = train_true)

lasso_val_eval_data <- data.frame(fit = as.vector(val_lasso),
                                  true = val_true)

lasso_train_eval_text <- eval_text(lasso_train_eval)
lasso_val_eval_text <- eval_text(lasso_val_eval)

lasso_train_eval_plot <- plot_evaluation(lasso_train_eval_data, true, fit, lasso_train_eval_text, 'Evalu
lasso_val_eval_plot <- plot_evaluation(lasso_val_eval_data, true, fit, lasso_val_eval_text, 'Evaluation
grid.arrange(lasso_train_eval_plot, lasso_val_eval_plot, nrow = 1)
```



When comparing the MSE on both the training and validation set, it is observed that the MSE for the validation (1.62e^9) is lower than that of the training (1.82e^9). This was again not expected as models usually perform slightly better on the training set. This could again be explained due to the extreme outlier in the training set, which causes the MSE to be greatly affected but by observing the MAE, the training set performed better.

Ridge

Next, we built the ridge regression model.

```
cv_ridge <- cv.glmnet(train.x, train.y, alpha = 0, nfolds=10)
ridge <- glmnet(train.x, train.y, alpha = 0, lambda = cv_ridge$lambda.min)

train_ridge <- exp(predict(ridge, newx = train.x)) * train$rpi_value
```

```
val_ridge <- exp(predict(ridge, newx = val.x)) * val$rpi_value

ridge_train_eval <- eval_results(train_ridge, train_true)
ridge_val_eval <- eval_results(val_ridge, val_true)
cv_ridge$lambda.min
```

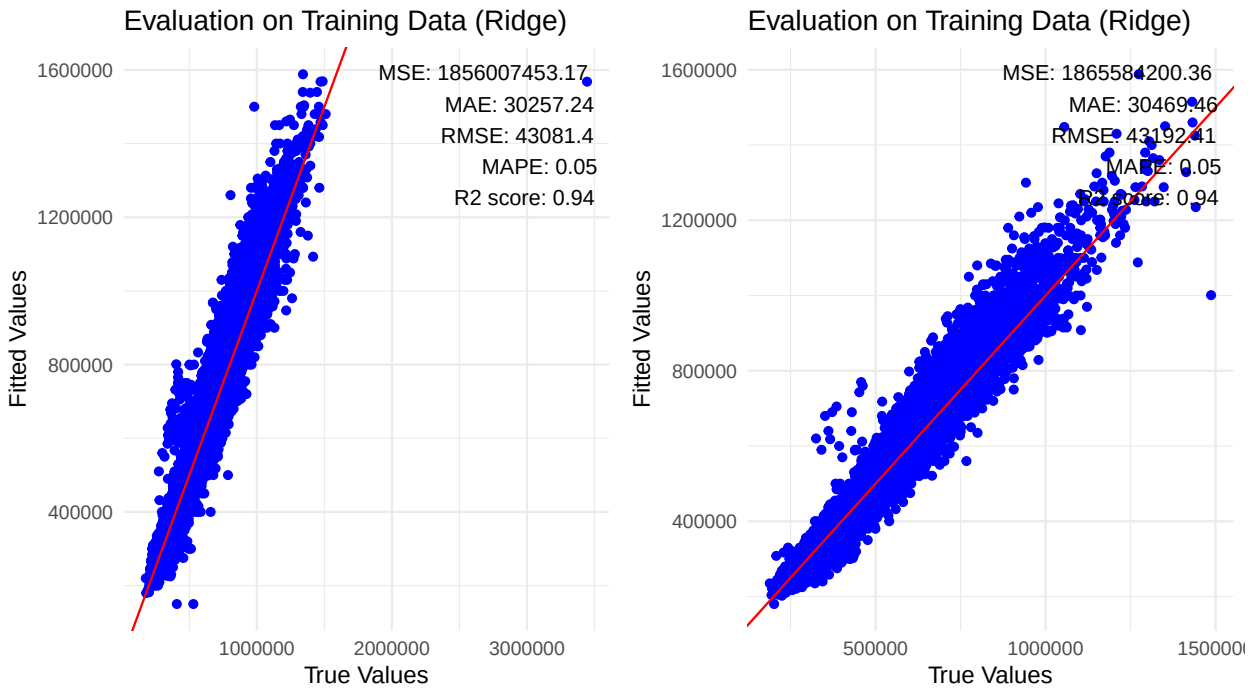
```
## [1] 0.02184734
```

The optimal lambda was found to be 0.0218 and prediction was done on the training and validation sets.

Evaluating Ridge Model

We then evaluated the ridge regression model on the training and the validation data.

```
ridge_train_eval_data <- data.frame(fit = as.vector(train_ridge),
                          true = train_true)

ridge_val_eval_data <- data.frame(fit = as.vector(val_ridge),
                                  true = val_true)

ridge_train_eval_text <- eval_text(ridge_train_eval)
ridge_val_eval_text <- eval_text(ridge_val_eval)

ridge_train_eval_plot <- plot_evaluation(ridge_train_eval_data, train_ridge, train_true, ridge_train_eva
ridge_val_eval_plot <- plot_evaluation(ridge_val_eval_data, val_ridge, val_true, ridge_val_eval_text, '
grid.arrange(ridge_train_eval_plot, ridge_val_eval_plot, nrow = 1)
```



When comparing the MSE on both the training and validation set in ridge regression, it is observed that the MSE for the validation (1.86e^9) is higher than that of the training (1.87e^9). This is to be expected as models usually perform better in their training sets.

Elastic-Net

Lastly, we built an elastic-net regression model to hopefully get the best results from both ridge and lasso.

```
set.seed(124)
generate_cvmodels <- function(x){
  return(cv.glmnet(train.x,train.y,
         type.measure = "mse", alpha = x/10))
}
cv_models <- lapply(0:10, generate_cvmodels)

cv_error <- unlist(lapply(cv_models, function(x) x$cvm[x$lambda == x$lambda.min]))
best_alpha <- (which(cv_error == min(cv_error))-1)/10
best_alpha
```

```
## [1] 0.8
```

```
cv_elastic <- cv.glmnet(train.x, train.y, alpha = best_alpha, type.measure = "mse")
glm_elastic <- glmnet(train.x, train.y, alpha = best_alpha, lambda = cv_elastic$lambda.min)

train_elastic <- exp(predict(glm_elastic, train.x)) * train$rpi_value
val_elastic <- exp(predict(glm_elastic, val.x)) * val$rpi_value
cv_elastic$lambda.min
```

```
## [1] 0.00007598941
```

```
elastic_coeff <- coef(cv_elastic, s = "lambda.min")

# Count the number of non-zero coefficients (excluding the intercept)
elastic_non_zero <- sum(elastic_coeff != 0) - 1  # Subtract 1 to exclude the intercept
elastic_non_zero
```

```
## [1] 225
```

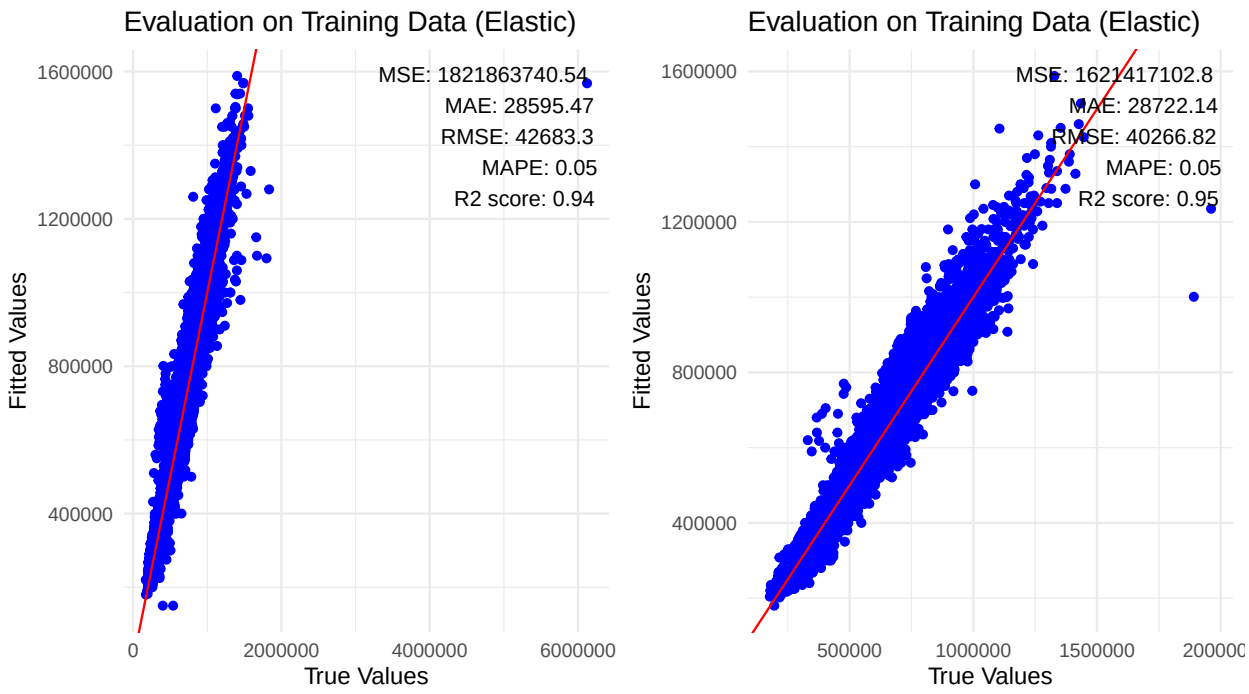
We observed that the elastic-net regression kept 225 predictor variables, the best alpha value is 0.8 and the optimal lambda is 0.0000760. Prediction was then done to the training and validation sets.

Evaluating Elastic Net model

We then evaluated the model on the training and validation data.

```
elastic_train_eval <- eval_results(train_elastic, train_true)
elastic_val_eval <- eval_results(val_elastic, val_true)

elastic_train_eval_data <- data.frame(fit = as.vector(train_elastic),
                          true = train_true)

elastic_val_eval_data <- data.frame(fit = as.vector(val_elastic),
                                    true = val_true)

elastic_train_eval_text <- eval_text(elastic_train_eval)
```

```
elastic_val_eval_text <- eval_text(elastic_val_eval)

elastic_train_eval_plot <- plot_evaluation(elastic_train_eval_data, train_elastic, train_true, elastic_
elastic_val_eval_plot <- plot_evaluation(elastic_val_eval_data, val_elastic, val_true, elastic_val_eval
grid.arrange(elastic_train_eval_plot, elastic_val_eval_plot, nrow=1)
```



By observing the 2 plots on training and validation, the MSE for the training set is 1.82e^9 while the validation was 1.62e^9. This was again unexpected due to models usually performing better in the training dataset. However, once again, due to the extreme anomaly in the training dataset, it caused the MSE to increase significantly. This can be further proven by the MAE, where the MAE of the training is lower than that of the validation.

Summary

To summarize, we will look at the MSE and R2 score for all the models used

```
summary <- data.frame(
  Model = c('MLR', 'Ridge', 'Lasso', 'Elastic'),
  Train_MSE = c(train2_eval$MSE, ridge_train_eval$MSE, lasso_train_eval$MSE, elastic_train_eval$MSE),
  Validation_MSE = c(val_eval$MSE, ridge_val_eval$MSE, lasso_val_eval$MSE,elastic_val_eval$MSE),
  Train_R2 = c(train2_eval$R2, ridge_train_eval$R2, lasso_train_eval$R2, elastic_train_eval$R2),
  Validation_R2 = c(val_eval$R2, ridge_val_eval$R2, lasso_val_eval$R2, elastic_val_eval$R2))

summary <- summary %>% mutate('MSE Difference (%)' = (((Validation_MSE-Train_MSE)/Validation_MSE)*100))

knitr::kable(summary, digits = 10)
```

Model	Train_MSE	Validation_MSE	Train_R2	Validation_R2	MSE Difference (%)
MLR	1795988077	1637884477	0.9416969	0.9464833	-9.6529152
Ridge	1856007453	1865584200	0.9397485	0.9390434	0.5133377
Lasso	1823291347	1621612039	0.9408106	0.9470150	-12.4369642
Elastic	1821863741	1621417103	0.9408569	0.9470214	-12.3624351

```
coefficients <- coef(lm_df)
coefficients <- coefficients[-1]
top10 <- sort(abs(coefficients), decreasing = TRUE)[1:10]
predictors <- coefficients[names(top10)]
predictors
```

```
##                     flat_combinedTerrace 3 ROOM
##                                       0.7985202
##                     flat_combinedTerrace 4 ROOM
##                                       0.7713433
##                     flat_combinedType S2 5 ROOM
##                                       0.6050207
##               nearest_mrtBRAS BASAH MRT STATION
##                                       0.5870444
##                     flat_combinedType S1 4 ROOM
##                                       0.5783309
##        flat_combinedImproved-Maisonette 5 ROOM
##                                       0.5222013
##     flat_combinedPremium Apartment Loft 5 ROOM
##                                       0.4868045
##           nearest_mrtCHANGI AIRPORT MRT STATION
##                                       0.4711999
##            nearest_mrtTANJONG PAGAR MRT STATION
##                                       0.4707452
## flat_combinedMulti Generation MULTI-GENERATION
##                                       0.4589691
```

To summarize, the MSE and R2 score are provided above. Elastic net regression observed the lowest validation MSE, followed by lasso, MLR and lastly ridge The R2 score showed the same trends as that of the MSE. These results are to be surprising as ridge theoretically gives the most accurate results. However, this was not the case. This could be due to ridge regression over penalizing large coefficients which improve the predicting power of the model. In doing so, it results in less accurate results. Both Lasso and Elastic performed better than the basic MLR as these models remove predictors to only consider those that are significant and improve the predicting power of the model. Based on the MSE difference (%), Ridge showed the lowest percentage change, which in theory is good as it suggests that the model is able to generalize well and have less overfitting. However, when looking at the MSE, this small difference in MSE suggests that the model is able to generalize the outliers/noise well, resulting in less accurate results.

One major flaw in lasso and elastic net regression is that since it reduces the number of predictors, it is more likely to experience errors when trying to predict for an entry that is not present in the training dataset. Hence, although lasso and elastic net regression showed better performance (MSE and R2 score wise), we still decided to go with the basic MLR. By keeping MLR, we will have greater coverage across categories and can enhance generalizability with minimal sacrifice to accuracy.

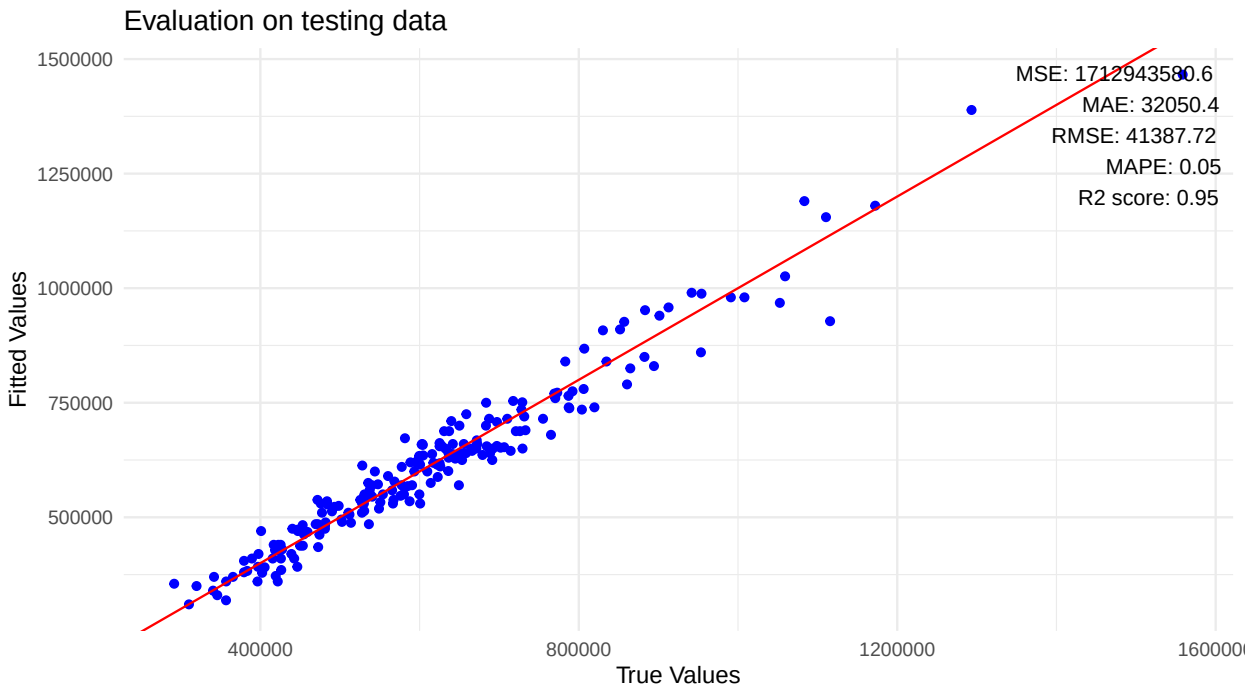
From this study, by observing the largest 10 coefficients of the MLR model and assuming relative equal significance of each predictor to their predicting power of the model, we can conclude that the 3 most important factors that determine the resale price of HDB flats are the **flat type**, **flat model** and the **nearest MRT**.

Testing

Finally, we will test our MLR model on a testing dataset. The steps to test the model are as follows:

1. Run the dataset in the Generating Coords Script
2. Run the following dataset into the Generating MRT Script
3. Predict the resale prices of HDBs with the linear regression model.

```
test <- read.csv('Proj_test.csv')
test <- test %>% mutate(flat_combined = paste((test$flat_model), (test$flat_type), sep = ' '))
test$flat_combined <- as.factor(test$flat_combined)
test_index <- rpi_index(test, "month")

# Predicting rpi value
test_rpi <- predict(lm_rpi, test_index)
test_true <- test$resale_price

test_mlr <- test %>%
  mutate(normalised_price = resale_price/test_rpi) %>%
  dplyr::select(town, storey_range, floor_area_sqm, nearest_mrt, distance_to_mrt, mrt_type, remaining_l

test_fit_mlr <- exp(predict(lm_df, test_mlr)) * test_rpi
test_eval <- eval_results(test_fit_mlr, test_true)

test_eval_data <- data.frame(fit = test_fit_mlr,
                                  true = test_true)

test_eval_text <- eval_text(test_eval)

test_eval_plot <- plot_evaluation(test_eval_data, fit, true, test_eval_text, 'Evaluation on testing data
test_eval_plot
```



From this test, we obtained a MSE of 1.71e^9, MAE of 32100, RMSE of 41400, MAPE of 0.01 and R2 Score of 0.95.

Limitations

Limitation 1: RPI

The RPI prediction may not always be accurate. Based on the graph plotted earlier of RPI, plateau and decrease will happen in times of crisis/due to external government factors/policies [2]. It is important to note that the RPI is a 'mirror' of the financial economy of Singapore and should not be taken as the driving factor for demand.

Some examples which has historically affected the RPI are as follows:

1997: Asian financial crisis

2013: Singapore's cooling measures

Limitation 2: Short-term Model

This model can only predict categories which are currently present in the training dataset. If new categories such as towns and MRTs are developed in the future, this model would not be able to predict of resale price with these factors and would need to be re-trained. Hence, this model should only be used in the short term from the date of publishing.

Limitation 3: Nearest MRT

This model takes the center point between all train stations and marks that as the MRT station. However, this may not always be true as certain MRT exits could be further than others. Furthermore, some locations may be close to an MRT exit, but actually far away from the MRT Station itself. This model does not consider the MRT exits which could lead to inaccurate results.

[2] https://www.channelnewsasia.com/singapore/property-cooling-measures-hdb-resale-prices-2013-2018-each-singapore-town-2385831

Supplementary Scripts

Generate Coordinates

This script takes the block number and street name from the original dataset and extracts the latitude and longitude from an api. [3]

Generating Coordinates Script

In [ ]:

```
import pandas as pd
import requests
import subprocess
import json
import csv

bearer_token = "YOUR_BEARER_TOKEN"
headers = {
    "Authorization": f"Bearer {'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZTNkMjhhNjFmMDQzNDg2Y2VkMDVkYTZjN2M
    "Content-Type": "application/json"
}

class Address:
    def __init__(self, block, street):
        self.block = block
        self.street = street
        self.coordinates = None
        self.nearest_mrt = None
        self.distance_to_mrt = None

    def __str__(self):
        return f"{self.block} {self.street}"
```

```
    def __eq__(self, other):
        if isinstance(other, Address):
            return self.block == other.block and self.street == other.street
        return False

    def __hash__(self):
        return hash((self.block, self.street))

def getCoordinates(address):
    print('Getting coordinates for : ' + address)
    url = "https://www.onemap.gov.sg/api/common/elastic/search?searchVal=" + address + "&returnGeom=Y&getAddrDetails
    response = requests.request("GET", url, headers=headers)
    return response.json()['results'][0]['LONGITUDE'], response.json()['results'][0]['LATITUDE']

def findNearestMRT(coordinates):
    try:
        coordinates_str = f"{coordinates[0]},{coordinates[1]}"
        result = subprocess.run(['node', 'findNearestMRT.js', coordinates_str], capture_output=True, text=True, chec
        nearestMRT = json.loads(result.stdout)
        mrt_name, mrt_distance = nearestMRT['result'][0]['station']['name'], nearestMRT['result'][0]['distance']
        print(f'Nearest MRT: {mrt_name}, Distance: {mrt_distance}')

        return mrt_name, mrt_distance

    except IndexError:
        print("Error finding nearest MRT")
        return None, None
```

[3] https://www.onemap.gov.sg/apidocs/

```python
def write_to_csv(address):
    with open('temp_output.csv', mode='a', newline='', encoding='utf-8') as file:
        writer = csv.DictWriter(file, fieldnames=["block", "street", "longitude", "latitude"])
        # writer = csv.DictWriter(file, fieldnames=["block", "street", "longitude", "latitude", "nearest_mrt", "dist

        # Check if file is empty to write the header
        if file.tell() == 0:
            writer.writeheader()

        writer.writerow({
            "block": address.block,
            "street": address.street,
            "longitude": address.coordinates[0],
            "latitude": address.coordinates[1]
        })

def main():
    file_path = 'test.csv'

    df = pd.read_csv(file_path)

    addresses = []

    for block, street in df[['block', 'street_name']].values:
        address = Address(block, street)
        if address not in addresses:
            addresses.append(address)
        else:
            print(f'Address {address} not appended')
```

```python
    for address in addresses:
        address.coordinates = getCoordinates(address.block + ' ' + address.street)
        # address.nearest_mrt, address.distance_to_mrt = findNearestMRT(address.coordinates)
        print(f'Progress: {addresses.index(address) + 1}/{len(addresses)}')

        write_to_csv(address)

    print("Addresses:")
    print(addresses)


if __name__ == "__main__":
    main()
```

Generate nearest MRT and distance

This script extracts the MRT coordinates from LTA's open data [4] and compares the latitude and longitude of each location in the dataset and compares it with the MRT coordinates to find the nearest MRT.

```
This Script uses:
1. Data with longitude and latitude
2. Shapefile of all mrt stations in Singapore (from data.gov)

# Importing Coordinates
```{r}
df <- read.csv('HDB_resale.csv', stringsAsFactors = T)

# convert month column to date type
df$month <- as.character(df$month)

# Asumming all entries are done on the 1st of each month
df$month <- paste0(df$month, "-01")
# Quantifying remaining lease as numerical terms
df$remaining_lease <- as.character(df$remaining_lease)
df <- df %>%
  mutate(
    years = as.numeric(str_extract(remaining_lease, "\\d+(?= years)")),  # Extract the years
    months = as.numeric(str_extract(remaining_lease, "(?<=years )\\d+")),  # Extract the months
    months = ifelse(is.na(months), 0, months),
    remaining_lease = years + (months / 12)  # Convert to decimal
  ) %>%
  select(-years, -months)

sum(is.na(df))
sum(duplicated(df))
nrow(df)
df <- unique(df)
```
```

[4] https://datamall.lta.gov.sg/content/datamall/en/static-data.html

```
[1] 0
[1] 0
[1] 200
```

MRT Coordinates

```{r}
shapefile_path <- "RapidTransitSystemStation.shp"

mrt_stations <- st_read(shapefile_path)
mrt_stations <- mrt_stations[-167, ] # not a MRT Station

# print(st_crs(mrt_stations))
# SG format uses SVY21 Coordinate reference system

mrt_df <- data.frame(
  station = character(nrow(mrt_stations)),
  longitude = numeric(nrow(mrt_stations)),
  latitude = numeric(nrow(mrt_stations)))

#Fixing the Centroid stuff:
# Filter out invalid geometries
mrt_stations <- mrt_stations[st_is_valid(mrt_stations$geometry), ]
# Calculate centroids after filtering
centroids <- st_centroid(mrt_stations$geometry)
centroids_wgs84 <- st_transform(centroids, crs = 4326)
coordinates <- st_coordinates(centroids_wgs84)
```

```{r}
# Store into mrt_df
mrt_df <- data.frame(
  station = mrt_stations$STN_NAM_DE,
  type = mrt_stations$TYP_CD_DES,
  longitude = coordinates[, 1],
  latitude = coordinates[, 2],
  stringsAsFactors = TRUE
)
mrt_df <- na.omit(mrt_df)
head(mrt_df)
```

Description: df [6 × 4]

	station <fctr>	type <fctr>	longitude <dbl>	latitude <dbl>
1	GALI BATU DEPOT	MRT	103.7561	1.397585
2	BEAUTY WORLD MRT STATION	MRT	103.7758	1.341204
3	DHOBY GHAUT MRT STATION	MRT	103.8458	1.299044
4	LAVENDER MRT STATION	MRT	103.8628	1.307372
5	RENJONG LRT STATION	LRT	103.8904	1.386748
6	ALJUNIED MRT STATION	MRT	103.8829	1.316432

6 rows

Calculating Nearest MRT and distance from MRT

```{r}
df$nearest_mrt <- NA
df$distance_to_mrt <- NA

dist_matrix <- distm(df[, c("longitude", "latitude")], mrt_df[, c("longitude", "latitude")], fun = distHaversine)

# Finding the nearest MRT station and its distance for each location
nearest_mrt_indices <- apply(dist_matrix, 1, which.min)
nearest_mrt_distances <- apply(dist_matrix, 1, min)

df$nearest_mrt <- mrt_df$station[nearest_mrt_indices]
df$distance_to_mrt <- nearest_mrt_distances
df$mrt_type <- mrt_df$type[nearest_mrt_indices]
```

Saving Data

```{r}
write.csv(df, "test_mrt.csv", row.names = FALSE)
```