*This is the main submission document. **Save and rename this document filename with your <u>registered full name</u> as Prefix before submission.***

Class	6
Full Name	Ng Chi Hui
Matriculation Number	U1922243C

** : Delete and replace as appropriate.*

Declaration of Academic Integrity
By submitting this assignment for assessment, I declare that this submission is my own work, unless otherwise quoted, cited, referenced or credited. I have read and understood the Instructions to CBA.PDF provided and the Academic Integrity Policy.
I am aware that failure to act in accordance with the University's Academic Integrity Policy may lead to the imposition of penalties which may include the requirement to revise and resubmit an assignment, receiving a lower grade, or receiving an F grade for the assignment; suspension from the University or termination of my candidature.

I consent to the University copying and distributing any or all of my work in any form and using third parties to verify whether my work contains plagiarised material, and for quality assurance purposes.

Please insert an "X" within the square brackets below to indicate your selection.
[✓] **I have read and accept the above.**

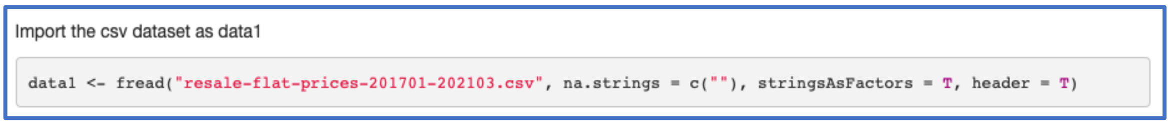
Table of Contents

Answer to Q1: 2
Answer to Q2: 5
Answer to Q3: 11
Answer to Q4: 18
Answer to Q5: 26
Answer to Q6: 28
Answer to Q7: 31

Answer to Q1:

A. Import the csv dataset as **data1** and ensure that all textual data are treated as categories instead of text string characters. Show your code.

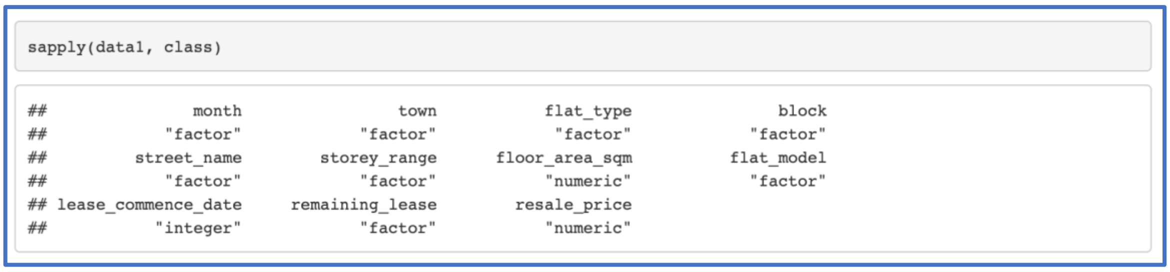
The csv dataset is imported as data using the fread() function found in the data.table library. By importing the data using stringsAsFactors = T, this will treat all textual data as categories/factors instead of text string characters.



Import the csv dataset as data1

```
data1 <- fread("resale-flat-prices-201701-202103.csv", na.strings = c(""), stringsAsFactors = T, header = T)
```

To further verify that all textual data are treated as categories, we can use the sapply() function to check the data types of all the columns.

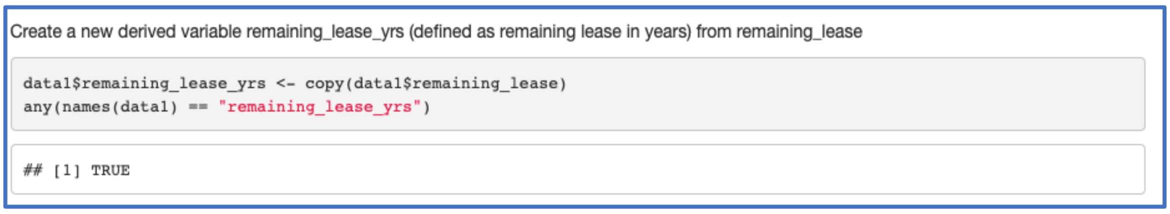


```
sapply(data1, class)
```

```
##                month                town           flat_type               block
##             "factor"            "factor"            "factor"            "factor"
##          street_name        storey_range      floor_area_sqm          flat_model
##             "factor"            "factor"           "numeric"            "factor"
## lease_commence_date     remaining_lease        resale_price
##            "integer"            "factor"           "numeric"
```

This shows that all textual data have the data type of “factor”. Only resale_price, floor_area_sqm have “numeric” data type and lease_commence_data have data type of “integer”.

B. Create a new derived variable **remaining_lease_yrs** (defined as remaining lease in years) from remaining_lease and save as an integer datatype column in data1. Show your code.

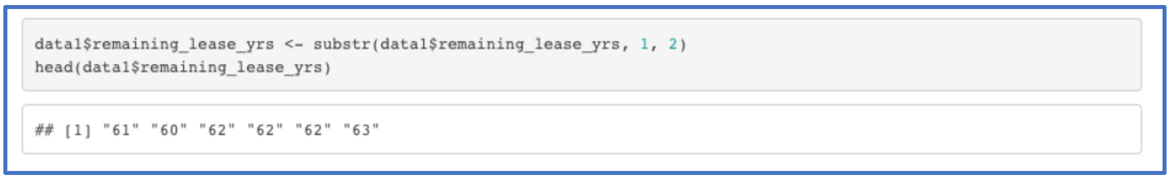


Create a new derived variable remaining_lease_yrs (defined as remaining lease in years) from remaining_lease

```
data1$remaining_lease_yrs <- copy(data1$remaining_lease)
any(names(data1) == "remaining_lease_yrs")
```

```
## [1] TRUE
```

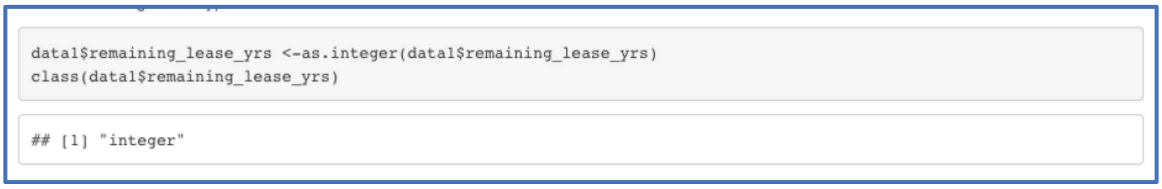
Using the copy() function, we can copy the enter columns of remaining_lease to the new column remaining_lease_yrs. To ensure that the column is created we can check if the data contains the name of the new column and it returns TRUE which shows that the column has been successfully created.



```
data1$remaining_lease_yrs <- substr(data1$remaining_lease_yrs, 1, 2)
head(data1$remaining_lease_yrs)
```

```
## [1] "61" "60" "62" "62" "62" "63"
```

Using the substr() functions helps to extract the first two characters in each of the remaining_lease_yrs row.

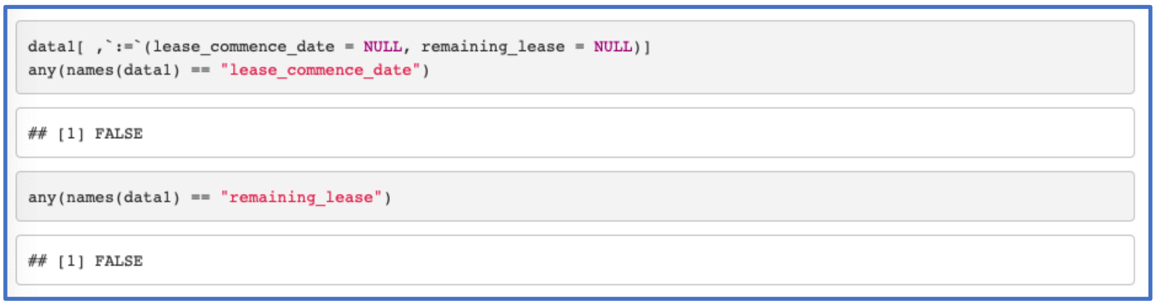


```
data1$remaining_lease_yrs <-as.integer(data1$remaining_lease_yrs)
class(data1$remaining_lease_yrs)
```

```
## [1] "integer"
```

The remaining_lease_yrs column is currently in textual form. To obtained the column is an integer datatype, we can invoke the as.integer() function

C. Remove lease_commence_date and remaining_lease from data1. Show your code.



```
data1[ ,`:=`(lease_commence_date = NULL, remaining_lease = NULL)]
any(names(data1) == "lease_commence_date")
```

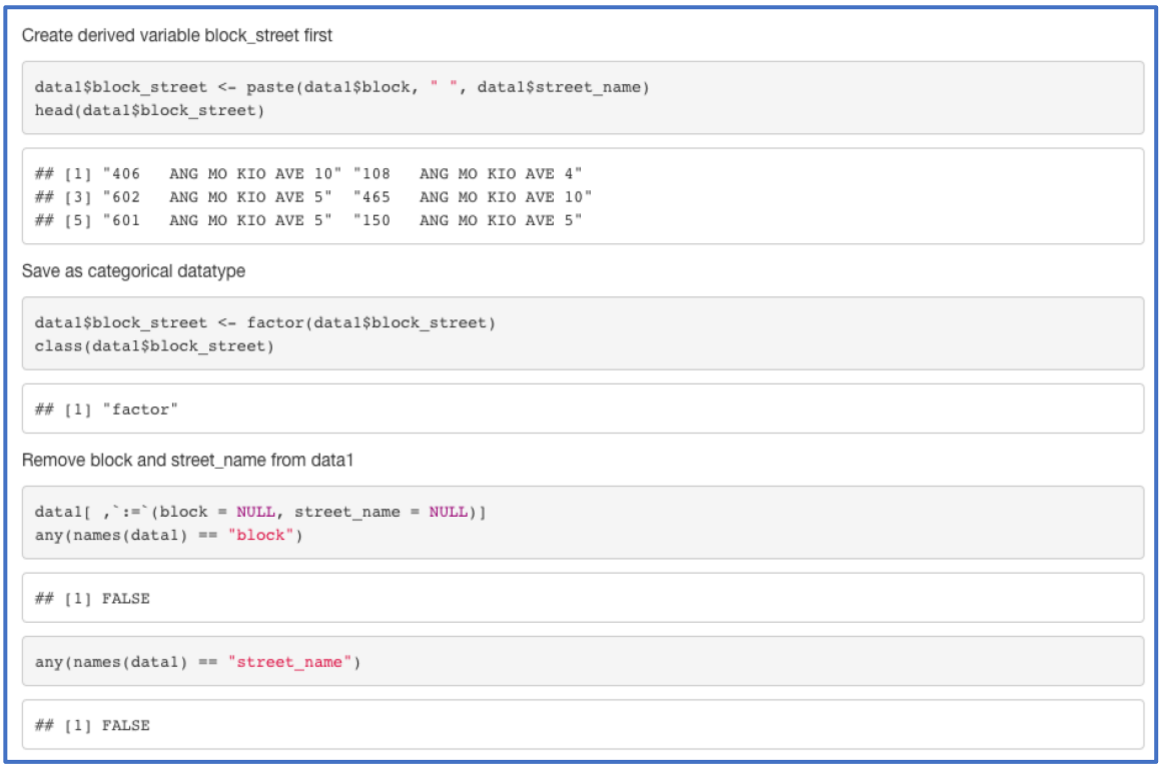
```
## [1] FALSE
```

```
any(names(data1) == "remaining_lease")
```

```
## [1] FALSE
```

Using the functions available in the data.table library, we are able to assign lease_commece date and remaining_lease to null, which deletes these columns. Following which we check if the names of these two columns exists and FALSE is being returned showing that these columns no longer exists.

D. Create a new derived variable **block_street** by combining block and street information (with one white space as separator) and save as a categorical datatype column in data1. Remove block and street_name from data1. Show your code.



Create derived variable block_street first

```
data1$block_street <- paste(data1$block, " ", data1$street_name)
head(data1$block_street)
```

```
## [1] "406   ANG MO KIO AVE 10" "108   ANG MO KIO AVE 4"
## [3] "602   ANG MO KIO AVE 5"  "465   ANG MO KIO AVE 10"
## [5] "601   ANG MO KIO AVE 5"  "150   ANG MO KIO AVE 5"
```

Save as categorical datatype

```
data1$block_street <- factor(data1$block_street)
class(data1$block_street)
```

```
## [1] "factor"
```

Remove block and street_name from data1

```
data1[ ,`:=`(block = NULL, street_name = NULL)]
any(names(data1) == "block")
```

```
## [1] FALSE
```

```
any(names(data1) == "street_name")
```

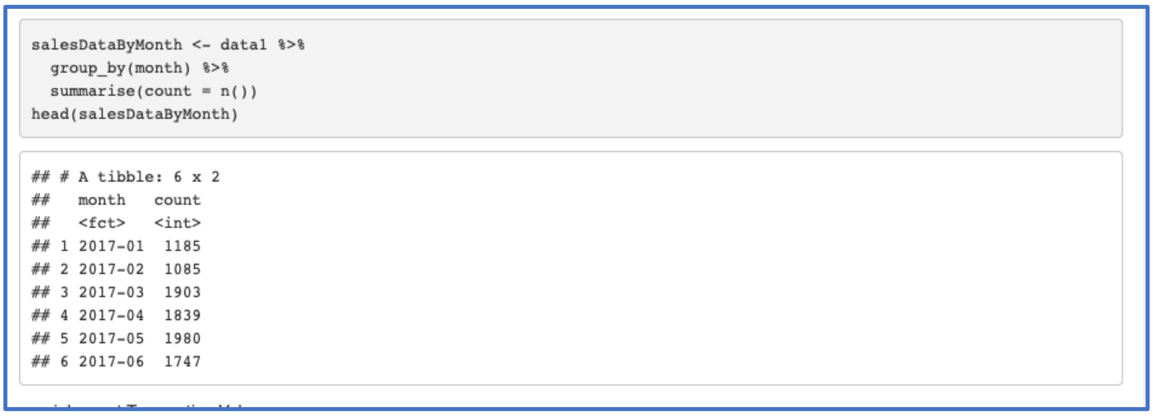
```
## [1] FALSE
```

Using the paste() function we are able to merge the data in the block and street_name column with a white space as a separator. Using the factor() function, we convert the column into a categorical datatype and verify that the datatype is correctly saved as categorical. Lastly, we drop the block and street_name column and check that the column is successfully being dropped.

Answer to Q2:

A. Which month year has the (i) lowest transaction volume, (ii) highest transaction volume, and what are their number of sales?

To answer this question, we first group the data by the month and count the number of transaction in each month year.

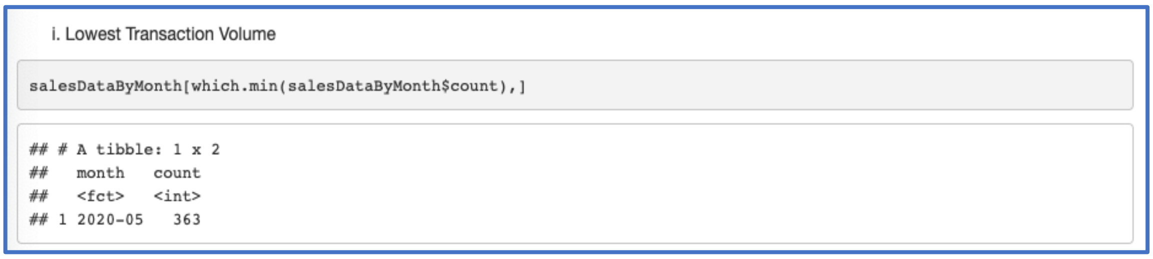


```
salesDataByMonth <- data1 %>%
  group_by(month) %>%
  summarise(count = n())
head(salesDataByMonth)
```

```
## # A tibble: 6 x 2
##   month   count
##   <fct>   <int>
## 1 2017-01  1185
## 2 2017-02  1085
## 3 2017-03  1903
## 4 2017-04  1839
## 5 2017-05  1980
## 6 2017-06  1747
```

(i) Lowest Transaction Volume

To find the month year with the lowest transaction, we select the row from the table which has the least transaction.



i. Lowest Transaction Volume

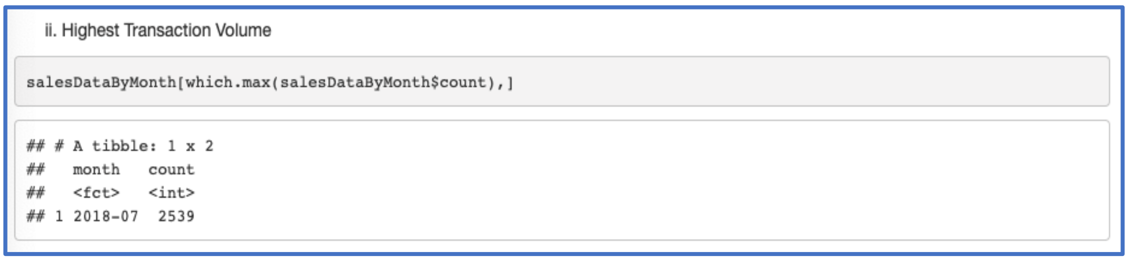
```
salesDataByMonth[which.min(salesDataByMonth$count),]
```

```
## # A tibble: 1 x 2
##   month   count
##   <fct>   <int>
## 1 2020-05   363
```

The lowest transaction value is during in the month of May (05) and year 2020 which occurs 363 times in the dataset.

(ii) Highest Transaction Volume

Similarly, to find the highest transaction volume, we can select the row from the table above which has the highest count.



ii. Highest Transaction Volume

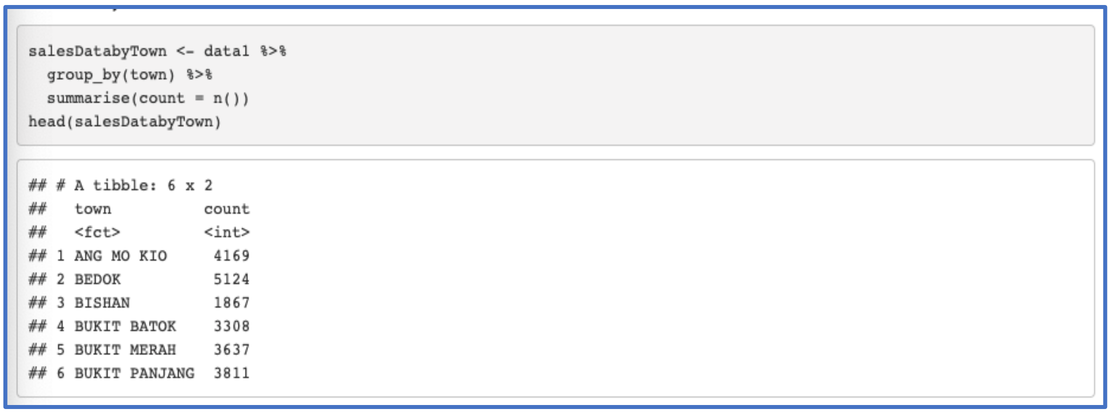
```
salesDataByMonth[which.max(salesDataByMonth$count),]
```

```
## # A tibble: 1 x 2
##   month   count
##   <fct>   <int>
## 1 2018-07  2539
```

Thus, the month year which has the highest transaction volume is 2018-07, where this level occurred 2539 times in the dataset.

B. Which town has the (i) lowest transaction volume, (ii) highest transaction volume, and what are their number of sales?

To find the town which has the lowest transaction volume, we group the data by town and count the number of transactions for each town.

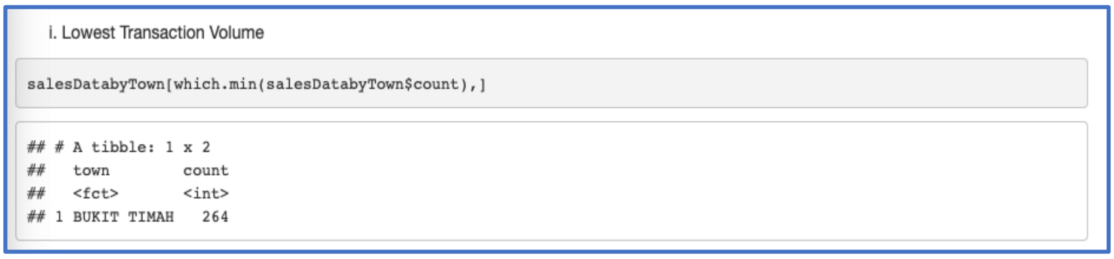


```
salesDatabyTown <- data1 %>%
  group_by(town) %>%
  summarise(count = n())
head(salesDatabyTown)
```

```
## # A tibble: 6 x 2
##   town          count
##   <fct>         <int>
## 1 ANG MO KIO     4169
## 2 BEDOK          5124
## 3 BISHAN         1867
## 4 BUKIT BATOK    3308
## 5 BUKIT MERAH    3637
## 6 BUKIT PANJANG  3811
```

(i) Lowest Transaction Volume

To find the town with the lowest transaction, we can select the row which has the lowest count from the table above.



i. Lowest Transaction Volume

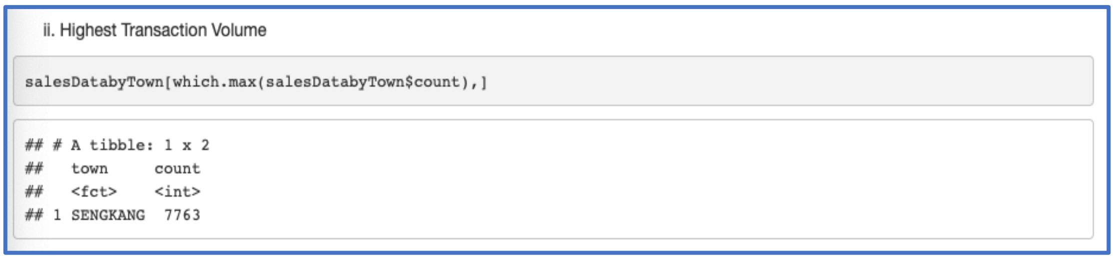
```
salesDatabyTown[which.min(salesDatabyTown$count),]
```

```
## # A tibble: 1 x 2
##   town        count
##   <fct>       <int>
## 1 BUKIT TIMAH   264
```

The town with the lowest transaction is Bukit Timah with 264 transactions.

(ii) Highest Transaction Volume

Similarly to find the town with the highest transaction, we can select the row which has the highest count from the table above.



ii. Highest Transaction Volume

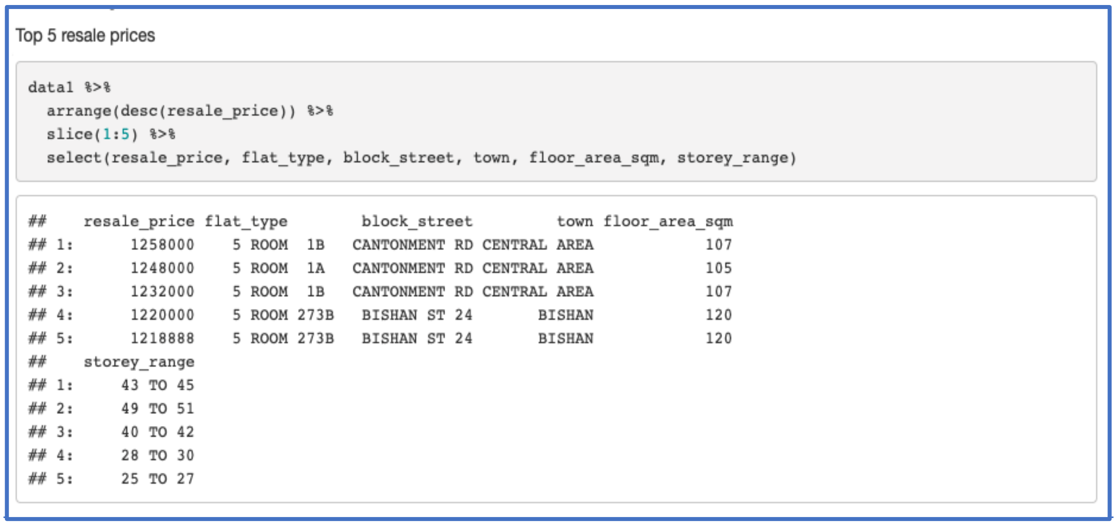
```
salesDatabyTown[which.max(salesDatabyTown$count),]
```

```
## # A tibble: 1 x 2
##   town     count
##   <fct>    <int>
## 1 SENGKANG  7763
```

The town with the highest transaction is Sengkang with 7763 transaction.

C. Generate an output that shows the top 5 resale prices and bottom 5 resale prices in terms of flat_type, block_street, town, floor_area_sqm, storey_range, and resale_price.

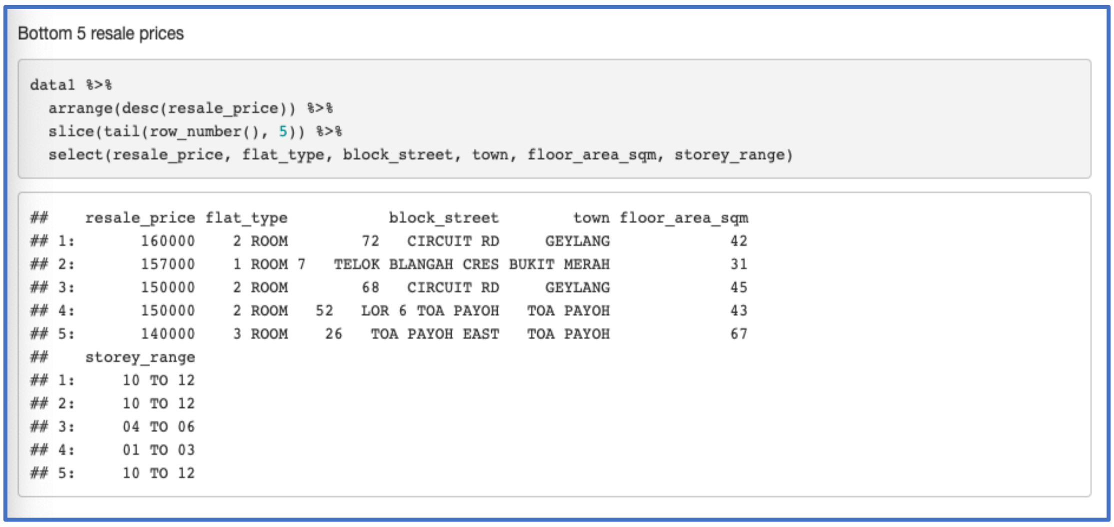
We first arrange the data in descending order of resale_price. To find the top 5 resale prices, we then select rows 1 to 5 and display the required columns in the table.


Top 5 resale prices

```
data1 %>%
  arrange(desc(resale_price)) %>%
  slice(1:5) %>%
  select(resale_price, flat_type, block_street, town, floor_area_sqm, storey_range)
```

```
##    resale_price flat_type       block_street         town floor_area_sqm
## 1:      1258000    5 ROOM   1B   CANTONMENT RD CENTRAL AREA            107
## 2:      1248000    5 ROOM   1A   CANTONMENT RD CENTRAL AREA            105
## 3:      1232000    5 ROOM   1B   CANTONMENT RD CENTRAL AREA            107
## 4:      1220000    5 ROOM 273B    BISHAN ST 24       BISHAN            120
## 5:      1218888    5 ROOM 273B    BISHAN ST 24       BISHAN            120
##    storey_range
## 1:     43 TO 45
## 2:     49 TO 51
## 3:     40 TO 42
## 4:     28 TO 30
## 5:     25 TO 27
```

Using similar approach of arranging the data in descending order of resale_price, to find the bottom 5 resale prices, we select the last 5 row numbers and the required columns we want to display.

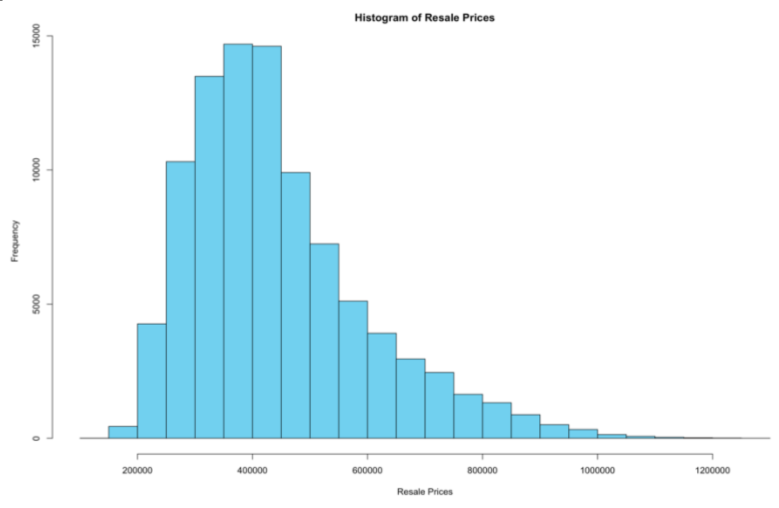
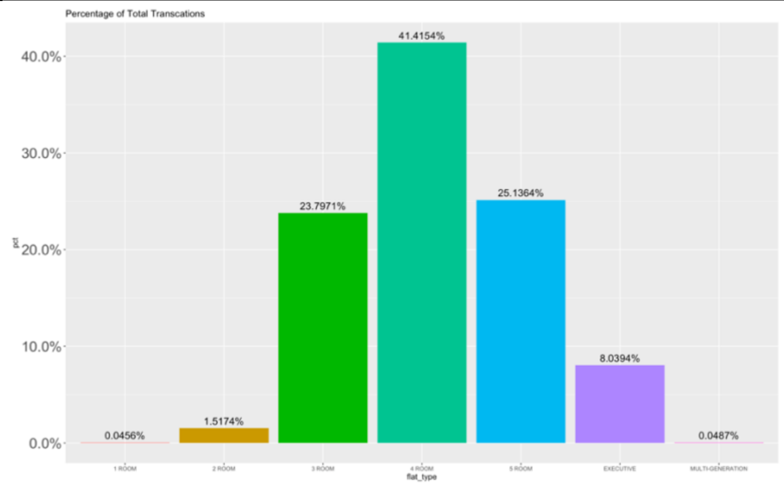
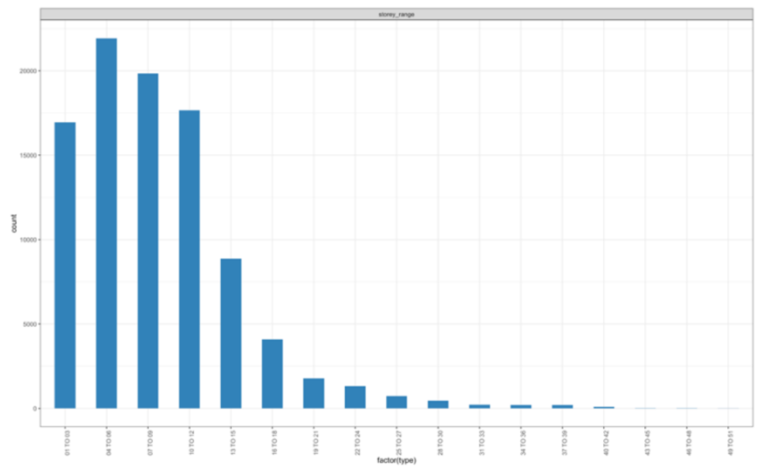

Bottom 5 resale prices

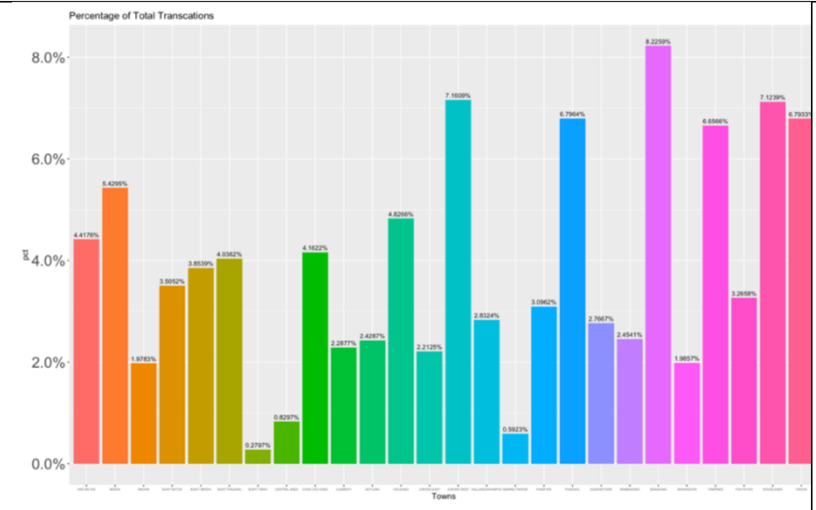
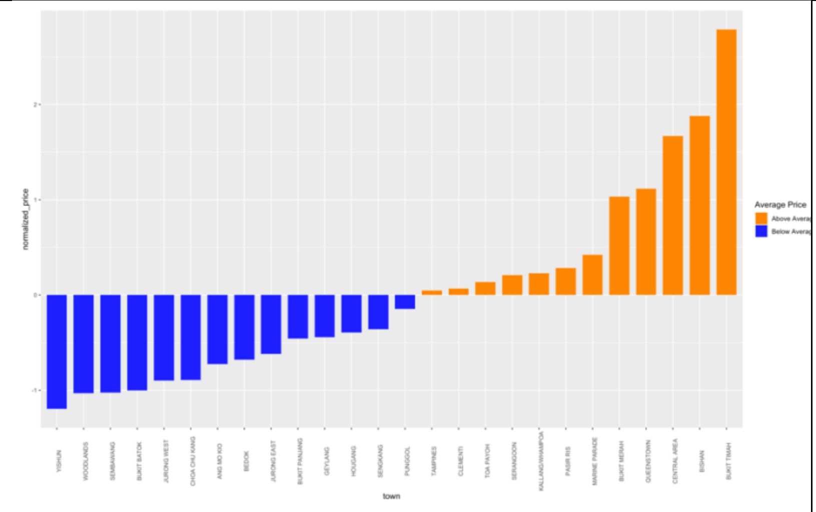
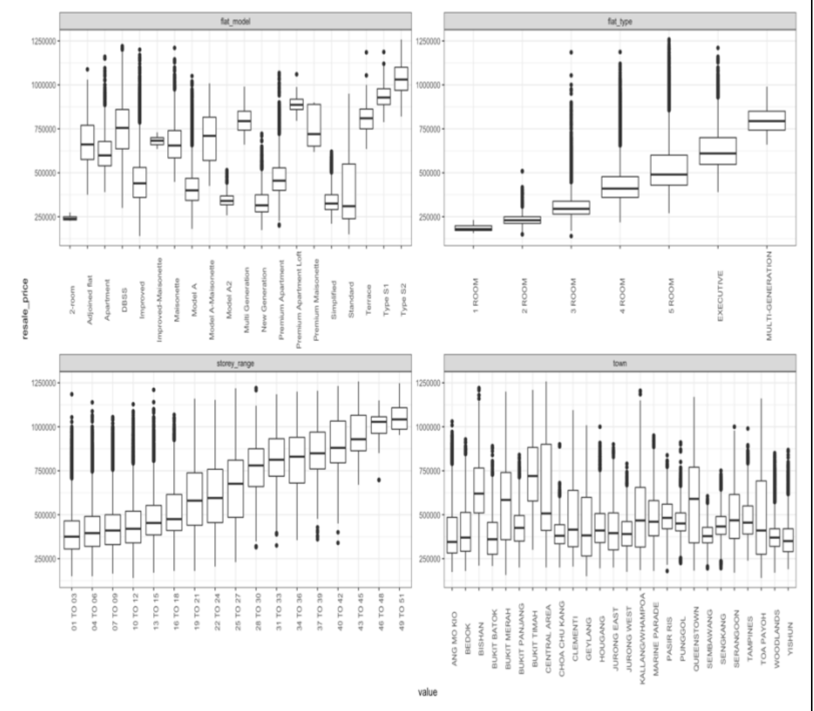
```
data1 %>%
  arrange(desc(resale_price)) %>%
  slice(tail(row_number(), 5)) %>%
  select(resale_price, flat_type, block_street, town, floor_area_sqm, storey_range)
```

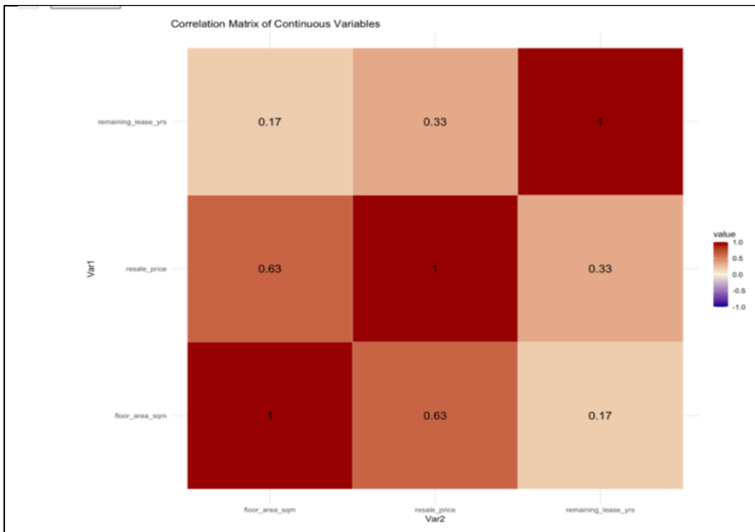
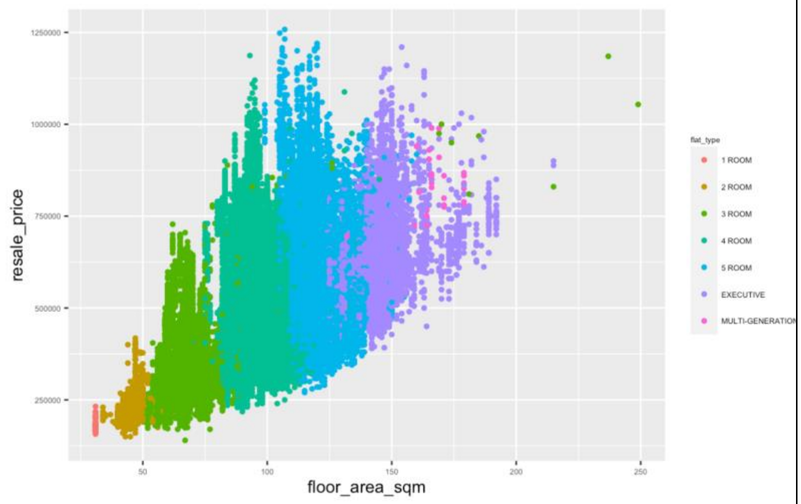
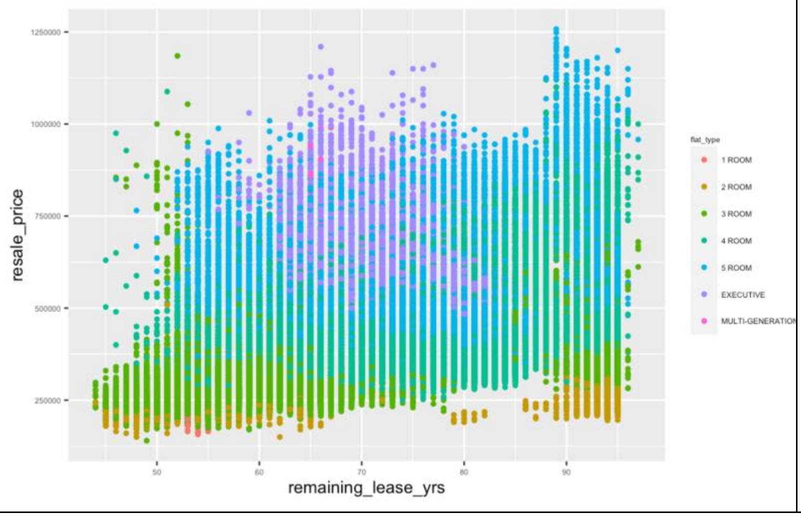
```
##    resale_price flat_type          block_street        town floor_area_sqm
## 1:       160000    2 ROOM       72    CIRCUIT RD     GEYLANG             42
## 2:       157000    1 ROOM 7   TELOK BLANGAH CRES BUKIT MERAH             31
## 3:       150000    2 ROOM       68    CIRCUIT RD     GEYLANG             45
## 4:       150000    2 ROOM    52   LOR 6 TOA PAYOH   TOA PAYOH             43
## 5:       140000    3 ROOM    26    TOA PAYOH EAST   TOA PAYOH             67
##    storey_range
## 1:     10 TO 12
## 2:     10 TO 12
## 3:     04 TO 06
## 4:     01 TO 03
## 5:     10 TO 12
```

From this 2 tables, we are able to see that the most expensive and cheapest flats tends to be around the same area and even the same few blocks. Flats near town are more expensive compared to flats that are very far away from town area. On the same note, lower priced flats are generally on the lower floors, with small floor areas vice versa for higher priced flats

D. Conduct additional data exploration. Show (with screenshots of software outputs) and explain the interesting findings discovered.

Visualisation	Explanation
	From the histogram of resale prices, we can observe that flat prices are concentrated in the range of $300,000 and $500,000. There is a right skew in resale prices as observed by the longer right tail relative to the left tail. High level of skewness can generate misleading results from statistical test later on and data transformation tools may be employed to make the skewed data close to normal distribution for model building.
	From the histogram of flat_type, we can observe that we have the highest frequencies of 4 room flats in the dataset, followed by 3 rooms and 5 room flats. There is a disproportionately low count of 1 room and multigenerational flats which should be removed later on to before building the models to prevent and imbalance dataset that can generate noise.
	The graph on the left reflects histogram of the storey range column in the dataset. We could observe that categories in the range of 40 - 51 have disproportionately low proportion compared to other categories.

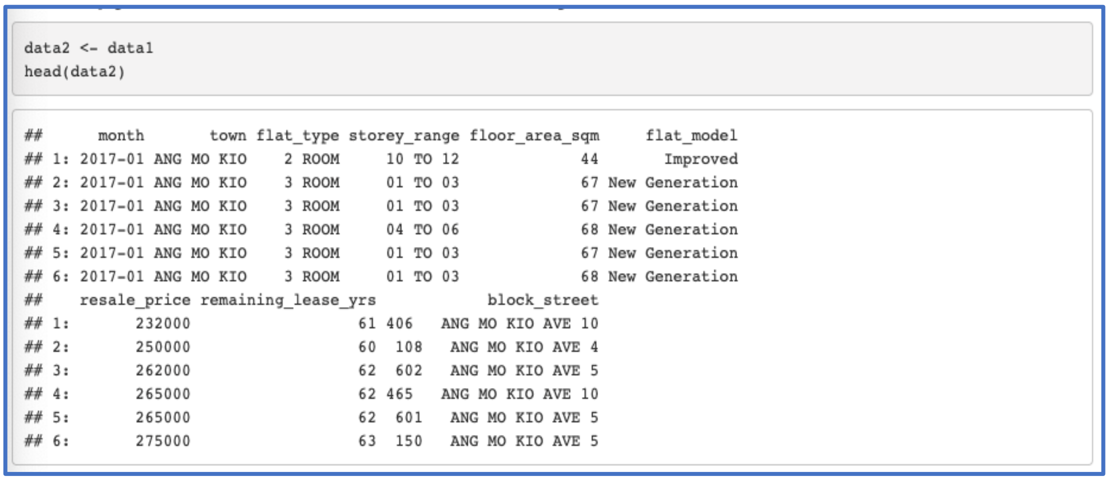
	The town column appears to have values that are quite spread about and is unlikely to have categories that cause noise during model building.
	The deviation plot on the left shows the relationship between town and resale price. The bars shows the deviation/difference from the mean of resale_price. The distance from town might be affect the deviation from the mean. Resale prices might be significantly affected by the town a flat is sitting in.
	The plots on the left show the resale price against 4 different variables namely flat_model, flat_type, storey_range and town. From the graph on the top right of bottom left, of flat_type and storey_range respectively there are clear trends. Whereas, for flat_model and town from the box plots a clear trend cannot be derived. For the flat_type, generally, the more number of rooms, the higher the resale_price though they are a number of outliers for each categories. Similarly, for storey_range, the higher the storey the higher the resale price

	The graph on the left shows a correlation matrix of continuous variables. Correlation measures “how much” two random variables change jointly. The correlation coefficient takes value between +1 and -1. The higher the magnitude of correlation coefficient is to 1 the more pronounce the linear relationship is. Negative values indicates inverse relationship The diagram depicts a strong and positive linear relationship between resale_price and floor_area_sqm. To further observe this trend, we can plot a scatter plot of resale_price against floor_area_sqm, as shown below.
	From this plot we can observe that larger flats tend to have higher resale value than smaller flatstele
	The plot on the left shows a scatterplot of Resale price against remaining_lease_yrs. There seems to be little visible relationship between remaining lease and price, no significant insights that can be derived. This is as expected as there’s positive and low correlation between resale_price and remaining_lease_years as observed in the correlation matrix above.

Answer to Q3:

A. Copy data1 and save as data2. Show your code.

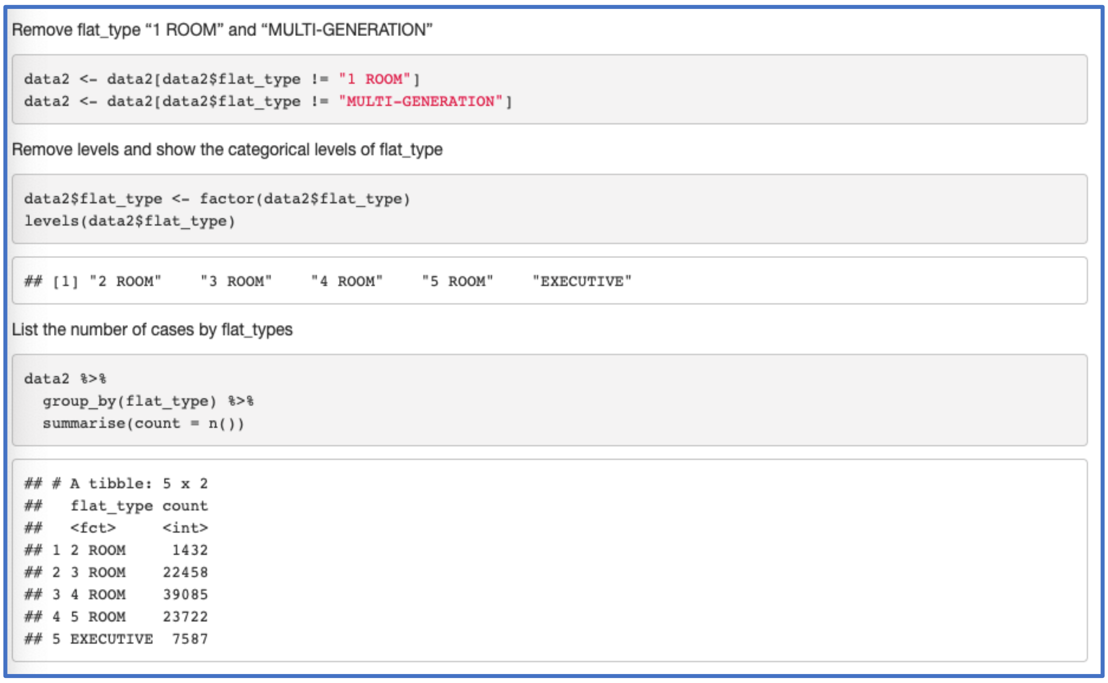
To copy data1 as save as data2, we can just use an assignment to assign data1 to data2.



```
data2 <- data1
head(data2)
```

```
##      month      town flat_type storey_range floor_area_sqm     flat_model
## 1: 2017-01 ANG MO KIO    2 ROOM     10 TO 12             44       Improved
## 2: 2017-01 ANG MO KIO    3 ROOM     01 TO 03             67 New Generation
## 3: 2017-01 ANG MO KIO    3 ROOM     01 TO 03             67 New Generation
## 4: 2017-01 ANG MO KIO    3 ROOM     04 TO 06             68 New Generation
## 5: 2017-01 ANG MO KIO    3 ROOM     01 TO 03             67 New Generation
## 6: 2017-01 ANG MO KIO    3 ROOM     01 TO 03             68 New Generation
##    resale_price remaining_lease_yrs            block_street
## 1:       232000                  61 406   ANG MO KIO AVE 10
## 2:       250000                  60  108   ANG MO KIO AVE 4
## 3:       262000                  62  602   ANG MO KIO AVE 5
## 4:       265000                  62 465   ANG MO KIO AVE 10
## 5:       265000                  62  601   ANG MO KIO AVE 5
## 6:       275000                  63  150   ANG MO KIO AVE 5
```

B. Remove flat_type "1 ROOM" and "MULTI-GENERATION" cases from data2, and ensure these levels are also removed from the categorical level definition. Show the categorical levels of flat_type and list the number of cases by flat_type.



Remove flat_type “1 ROOM” and “MULTI-GENERATION”

```
data2 <- data2[data2$flat_type != "1 ROOM"]
data2 <- data2[data2$flat_type != "MULTI-GENERATION"]
```

Remove levels and show the categorical levels of flat_type

```
data2$flat_type <- factor(data2$flat_type)
levels(data2$flat_type)
```

```
## [1] "2 ROOM"    "3 ROOM"    "4 ROOM"    "5 ROOM"    "EXECUTIVE"
```

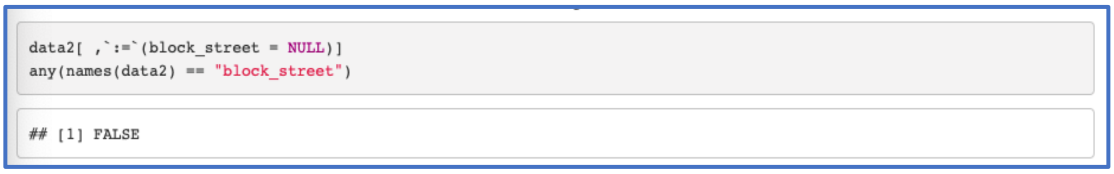
List the number of cases by flat_types

```
data2 %>%
  group_by(flat_type) %>%
  summarise(count = n())
```

```
## # A tibble: 5 x 2
##   flat_type count
##   <fct>     <int>
## 1 2 ROOM     1432
## 2 3 ROOM    22458
## 3 4 ROOM    39085
## 4 5 ROOM    23722
## 5 EXECUTIVE  7587
```

We first filter the column of flat_type to exclude rows containing “1 ROOM” or “MULTI-GENERATIONAL”. We then refactor the column again using factor() which will remove levels that have 0 rows in them as these levels were still present in the dataset. To verify that the levels have been correctly removed, we call the levels() function again. Lastly, we can view the count by each flat_type.

C. Remove block_street from data2. Show your code.

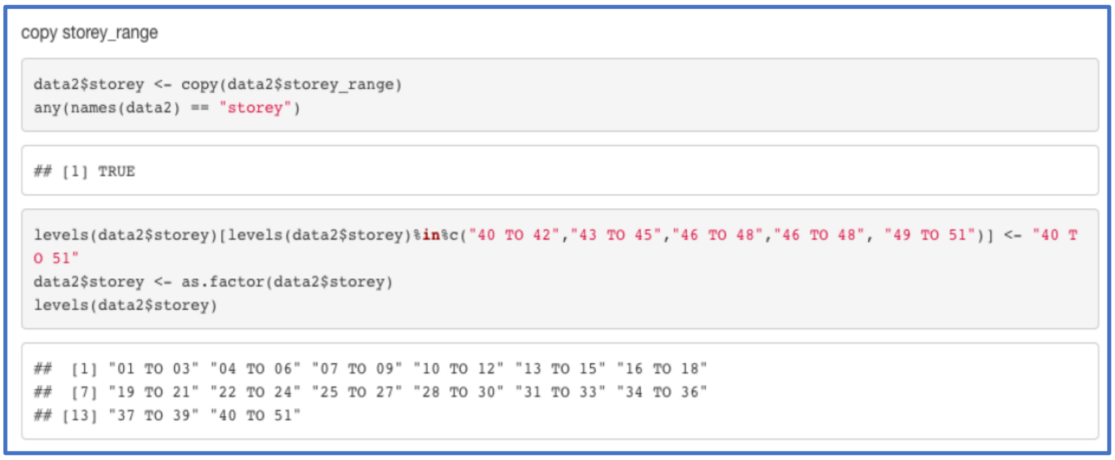


```
data2[ ,`:=`(block_street = NULL)]
any(names(data2) == "block_street")
```

```
## [1] FALSE
```

To do this, we just have to set the “block_street” column to NULL, which will help drop this column from the dataset. We can verify this by if “block_street” exists in the names of the all the columns in the dataset, for which FALSE is being returned.

D. In data2, create a new variable **storey** by copying storey_range, and then create and use the categorical level “40 to 51” to combine all the relevant storey levels into this bigger category. Show and verify that the categorical levels in storey are created correctly to hold the right cases.



copy storey_range

```
data2$storey <- copy(data2$storey_range)
any(names(data2) == "storey")
```

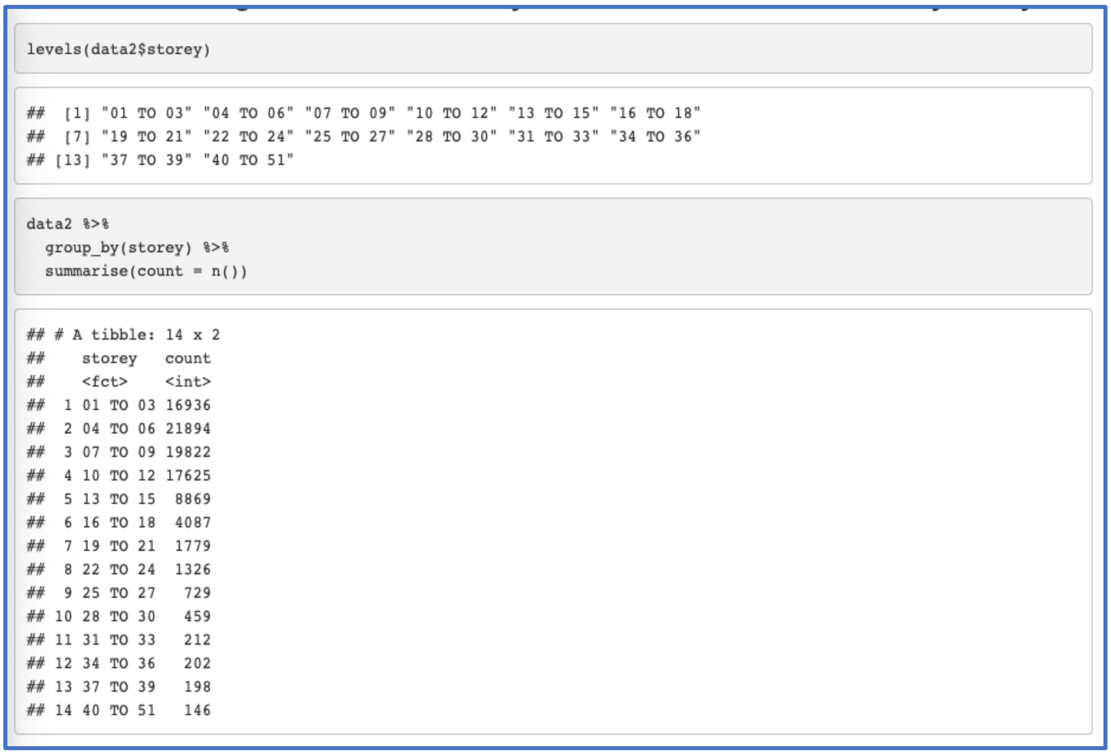
```
## [1] TRUE
```

```
levels(data2$storey)[levels(data2$storey)%in%c("40 TO 42","43 TO 45","46 TO 48","46 TO 48", "49 TO 51")] <- "40 T
O 51"
data2$storey <- as.factor(data2$storey)
levels(data2$storey)
```

```
##  [1] "01 TO 03" "04 TO 06" "07 TO 09" "10 TO 12" "13 TO 15" "16 TO 18"
##  [7] "19 TO 21" "22 TO 24" "25 TO 27" "28 TO 30" "31 TO 33" "34 TO 36"
## [13] "37 TO 39" "40 TO 51"
```

Using the copy() function we can duplicate storey_range column into storey and check the new storey column exists in the dataset which returns us TRUE. We will the combine the small levels into one big new category of “40 TO 51” and refactor the storey column with the factor() function. Lastly, call levels to check that the levels is what we want.

E. Show the categorical levels in storey and list the number of cases by storey.



```
levels(data2$storey)
```

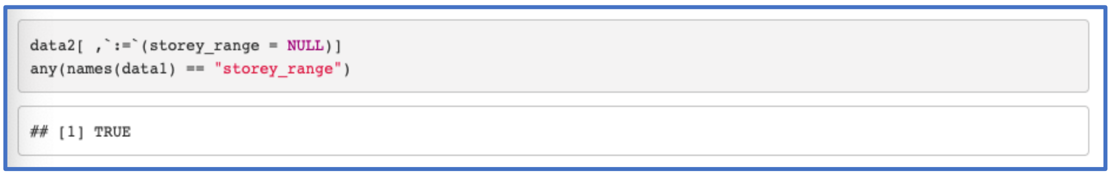
```
##  [1] "01 TO 03" "04 TO 06" "07 TO 09" "10 TO 12" "13 TO 15" "16 TO 18"
##  [7] "19 TO 21" "22 TO 24" "25 TO 27" "28 TO 30" "31 TO 33" "34 TO 36"
## [13] "37 TO 39" "40 TO 51"
```

```
data2 %>%
  group_by(storey) %>%
  summarise(count = n())
```

```
## # A tibble: 14 x 2
##    storey   count
##    <fct>    <int>
##  1 01 TO 03 16936
##  2 04 TO 06 21894
##  3 07 TO 09 19822
##  4 10 TO 12 17625
##  5 13 TO 15  8869
##  6 16 TO 18  4087
##  7 19 TO 21  1779
##  8 22 TO 24  1326
##  9 25 TO 27   729
## 10 28 TO 30   459
## 11 31 TO 33   212
## 12 34 TO 36   202
## 13 37 TO 39   198
## 14 40 TO 51   146
```

Using the levels() function, we are able to view all the levels under the storey column. Using the pipe operator(%>%) we can group the data by storey and count the number of rows of cases for each storey category.

F. Remove storey_range from data2. Show your code.

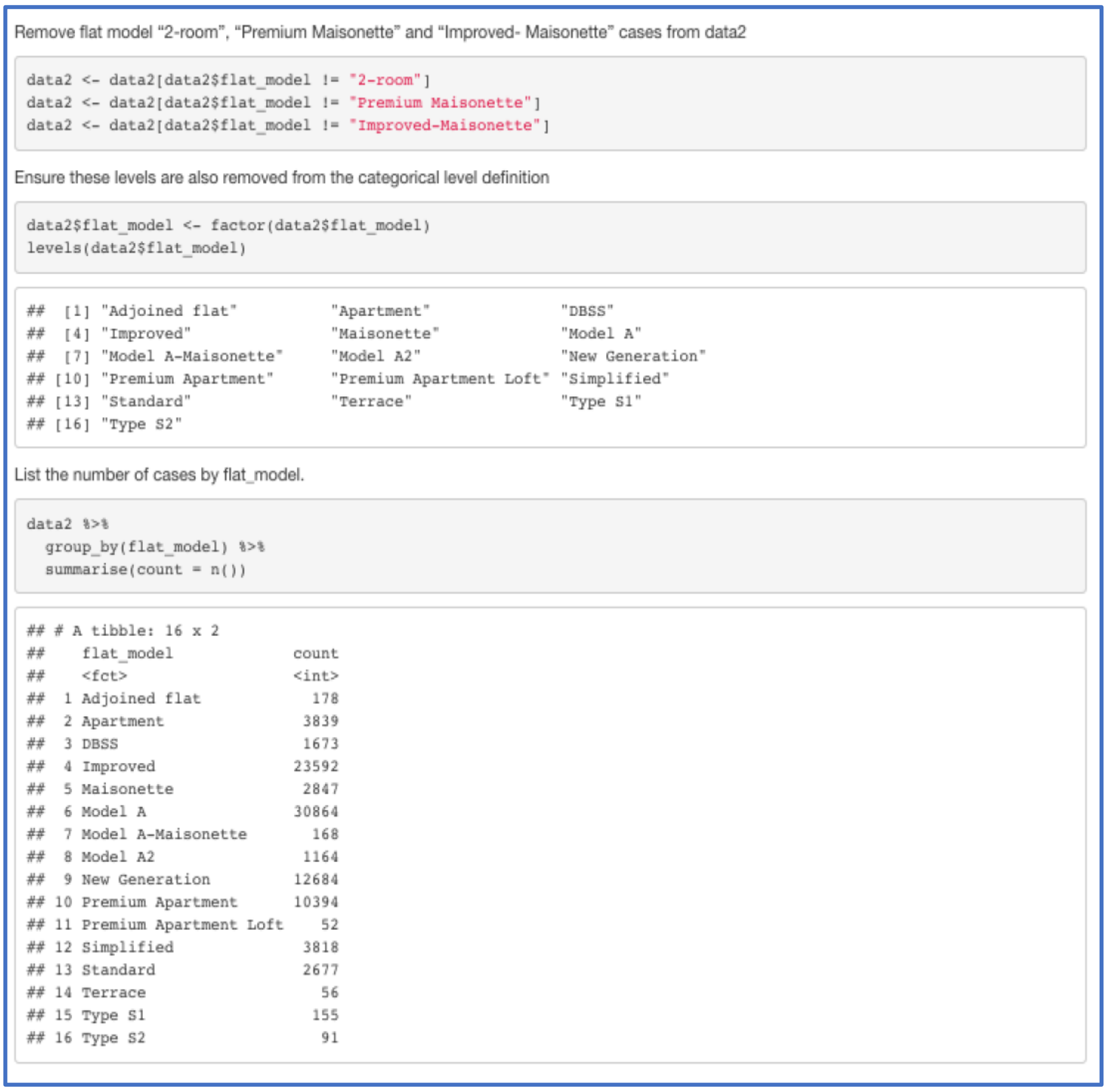


```
data2[ ,`:=`(storey_range = NULL)]
any(names(data1) == "storey_range")
```

```
## [1] TRUE
```

Set the storey_range column to NULL and verify that the column no longer exists.

G. Remove flat model "2-room", "Premium Maisonette" and "Improved-Maisonette" cases from data2, and ensure these levels are also removed from the categorical level definition. Show the categorical levels of flat_model and list the number of cases by flat_model.



Remove flat model “2-room”, “Premium Maisonette” and “Improved- Maisonette” cases from data2

```
data2 <- data2[data2$flat_model != "2-room"]
data2 <- data2[data2$flat_model != "Premium Maisonette"]
data2 <- data2[data2$flat_model != "Improved-Maisonette"]
```

Ensure these levels are also removed from the categorical level definition

```
data2$flat_model <- factor(data2$flat_model)
levels(data2$flat_model)
```

```
##  [1] "Adjoined flat"          "Apartment"              "DBSS"
##  [4] "Improved"               "Maisonette"             "Model A"
##  [7] "Model A-Maisonette"     "Model A2"               "New Generation"
## [10] "Premium Apartment"      "Premium Apartment Loft" "Simplified"
## [13] "Standard"               "Terrace"                "Type S1"
## [16] "Type S2"
```

List the number of cases by flat_model.

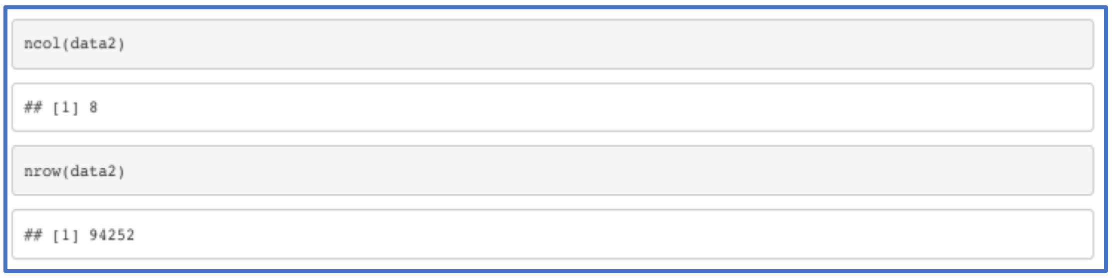
```
data2 %>%
  group_by(flat_model) %>%
  summarise(count = n())
```

```
## # A tibble: 16 x 2
##    flat_model             count
##    <fct>                  <int>
##  1 Adjoined flat            178
##  2 Apartment               3839
##  3 DBSS                    1673
##  4 Improved               23592
##  5 Maisonette              2847
##  6 Model A                30864
##  7 Model A-Maisonette       168
##  8 Model A2                1164
##  9 New Generation         12684
## 10 Premium Apartment      10394
## 11 Premium Apartment Loft    52
## 12 Simplified              3818
## 13 Standard                2677
## 14 Terrace                   56
## 15 Type S1                  155
## 16 Type S2                   91
```

First, the data2 dataset was filtered to remove the columns of “2-room”, “Premium Maisonette” and “Improved-Maisionette”, using the same method as before we refactor the columns using factor() and ensure that the levels that have 0 rows are removed.

Also using the pipe operator, we can group the flat_model and count the number of cases from each level

H. How many cases and columns are in data2 after completing all the data prep steps above?



```
ncol(data2)
```

```
## [1] 8
```

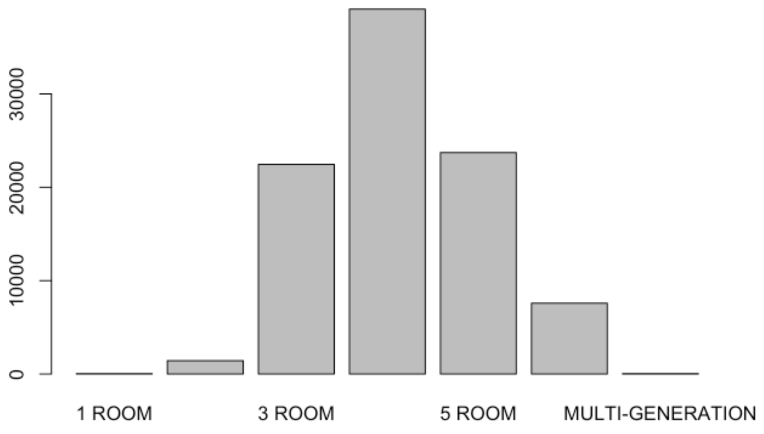
```
nrow(data2)
```

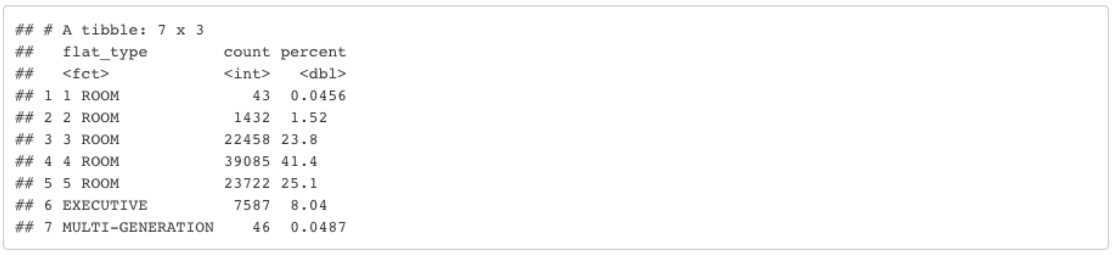
```
## [1] 94252
```

Using the ncol() function, we get 8 columns. The nrow() function returns use 94252 rows in the dataset after completing the data prep step

I. Suggest a reason for executing such data preparation steps listed above.

Step A: We duplicate the data1 to data2 as we do not want to make changes directly in data1 to overwrite in data1. Overwriting of data might not be preferred as changes to data might not be correctly executed by R. Extra effort has to be taken to reimport the data set if data is overwritten and if not careful writing new changes to the raw data will be disastrous.

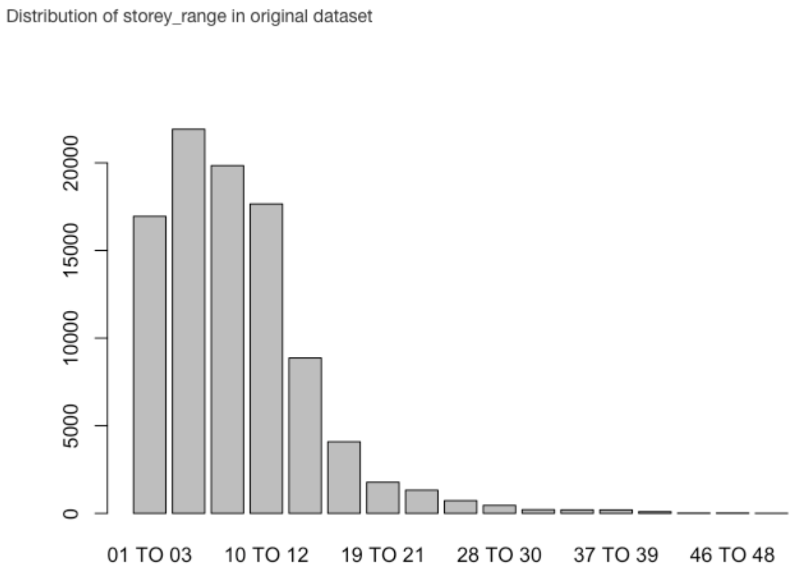


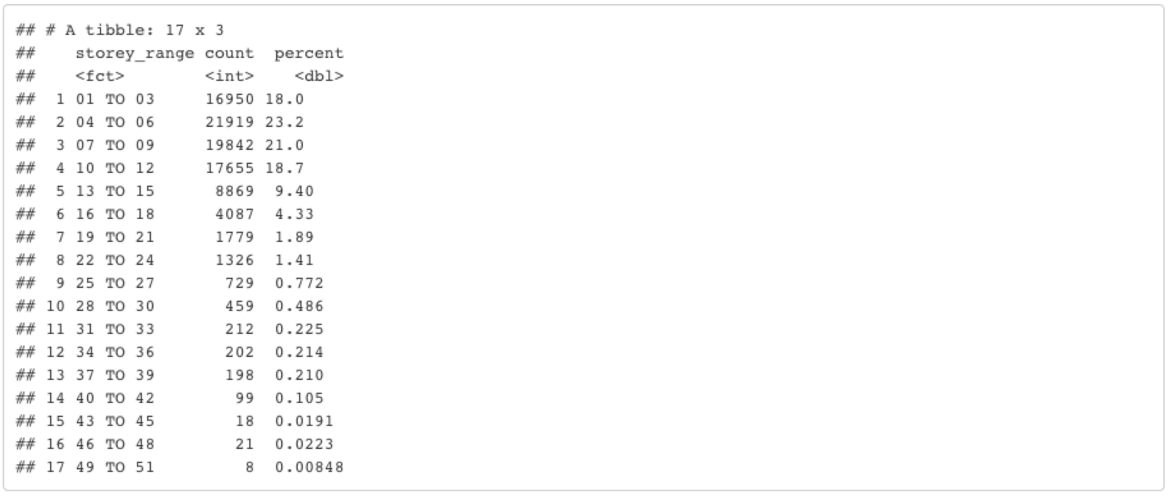


```
## # A tibble: 7 x 3
##   flat_type        count percent
##   <fct>            <int>   <dbl>
## 1 1 ROOM              43  0.0456
## 2 2 ROOM            1432  1.52
## 3 3 ROOM           22458 23.8
## 4 4 ROOM           39085 41.4
## 5 5 ROOM           23722 25.1
## 6 EXECUTIVE         7587  8.04
## 7 MULTI-GENERATION    46  0.0487
```

Step B: “1 room” and “Multi-Generation” flat type consists of 0.0456% and 0.0487% of the flat_type column. These are outliers which will present noise later when training the model. When building the models later on, keeping these categories means that the models will be trained on very few cases corresponding to these levels making predictions inaccurate. This might also introduce biasness in the models, cause errors and unexpected results especially when fitting into linear models that assumes "sensible" probability distribution (Brownlee, 2020).

Step C: Block_street was removed as this data was meaningless and is being reflected in other parts of the dataset already. This data consists of combined data and unique values. Having columns with repeated or similar information might result cause high multicollinearity later in our regression analysis. Even if the column was to be included, predictions can only be made for flats with address already in the dataset which might not be meaningful.

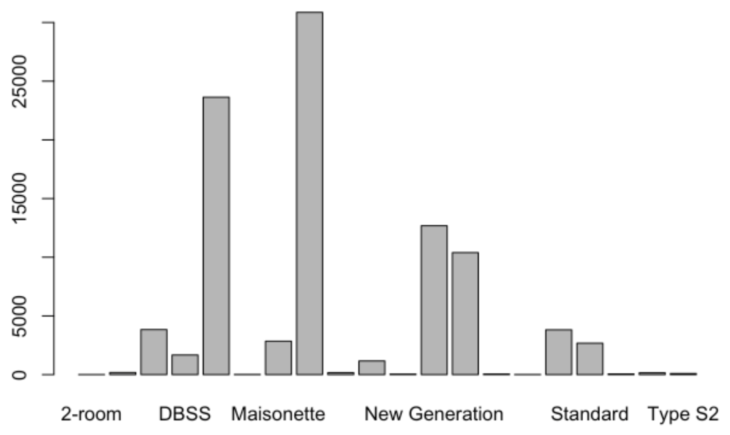


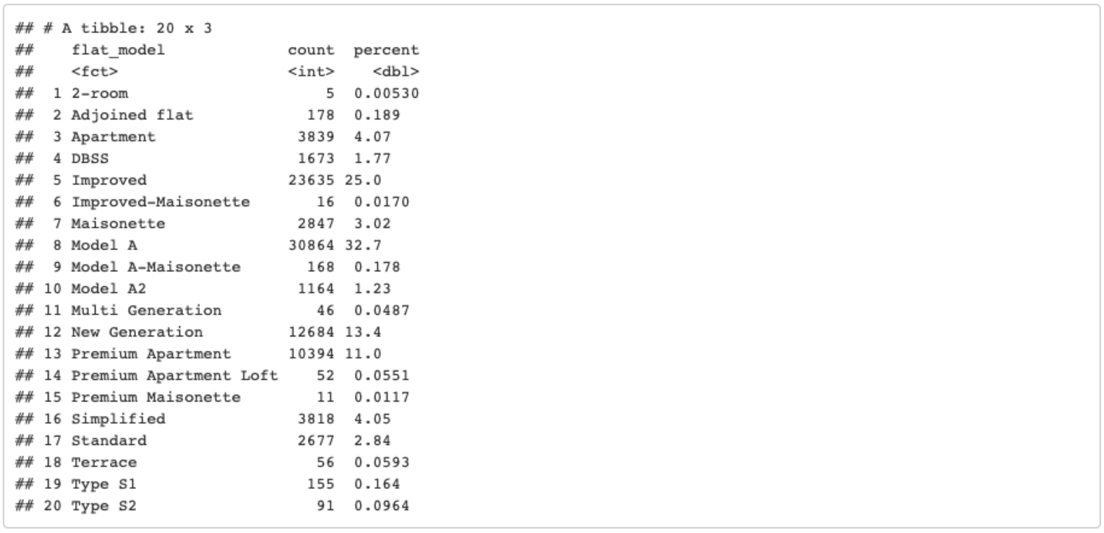


```
## # A tibble: 17 x 3
##    storey_range count  percent
##    <fct>        <int>    <dbl>
##  1 01 TO 03     16950 18.0
##  2 04 TO 06     21919 23.2
##  3 07 TO 09     19842 21.0
##  4 10 TO 12     17655 18.7
##  5 13 TO 15      8869  9.40
##  6 16 TO 18      4087  4.33
##  7 19 TO 21      1779  1.89
##  8 22 TO 24      1326  1.41
##  9 25 TO 27       729  0.772
## 10 28 TO 30       459  0.486
## 11 31 TO 33       212  0.225
## 12 34 TO 36       202  0.214
## 13 37 TO 39       198  0.210
## 14 40 TO 42        99  0.105
## 15 43 TO 45        18  0.0191
## 16 46 TO 48        21  0.0223
## 17 49 TO 51         8  0.00848
```

Step D: The relevant storey are being combined into one large category as the proportion of each individual storey are in small proportions relative to entire column most are < 0.1% This is done for the same reasons as step C.

Step F: Remove storey_range because the relevant results have already been reflected in storey. This is to prevent high correlation and multicollinearity between the variables later when training the models.





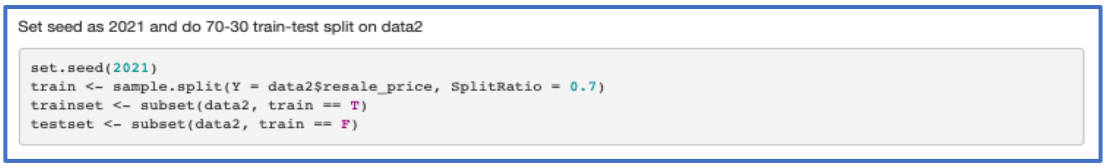
```
## # A tibble: 20 x 3
##    flat_model             count  percent
##    <fct>                  <int>    <dbl>
##  1 2-room                     5  0.00530
##  2 Adjoined flat            178  0.189
##  3 Apartment               3839  4.07
##  4 DBSS                    1673  1.77
##  5 Improved               23635 25.0
##  6 Improved-Maisonette       16  0.0170
##  7 Maisonette              2847  3.02
##  8 Model A                30864 32.7
##  9 Model A-Maisonette       168  0.178
## 10 Model A2                1164  1.23
## 11 Multi Generation          46  0.0487
## 12 New Generation         12684 13.4
## 13 Premium Apartment      10394 11.0
## 14 Premium Apartment Loft    52  0.0551
## 15 Premium Maisonette        11  0.0117
## 16 Simplified              3818  4.05
## 17 Standard                2677  2.84
## 18 Terrace                   56  0.0593
## 19 Type S1                  155  0.164
## 20 Type S2                   91  0.0964
```

Step G: Similarly, the categories of 2-room", "Premium Maisonette" and "Improved-Maisonette" has a very small percentage relative to the other categories in the dataset. By removing categories with a very small percentage to the dataset helps to make the dataset more balance and this reduces the noise during the training phase which can help improve the accuracy of the models.

Answer to Q4:

4. Set seed as 2021 and do 70-30 train-test split on data2. Execute (i) Linear Regression (all predictor variables), (ii) MARS degree 2, and (iii) Random Forest to predict the target variable. Create and show a summary table that lists the trainset RMSE and testset RMSE for the 3 models (to nearest dollar). Which model performed the best?

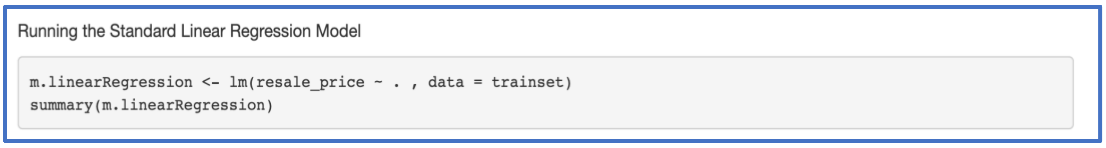
Set seed as 2021 to ensure reproducibility. Perform a 70-30 train-test split on data2, where the models will be trained on 70% of the data and evaluated on 30% of the remaining unseen data.


Set seed as 2021 and do 70-30 train-test split on data2

```r
set.seed(2021)
train <- sample.split(Y = data2$resale_price, SplitRatio = 0.7)
trainset <- subset(data2, train == T)
testset <- subset(data2, train == F)
```

Construction of Models

(i) Linear Regression

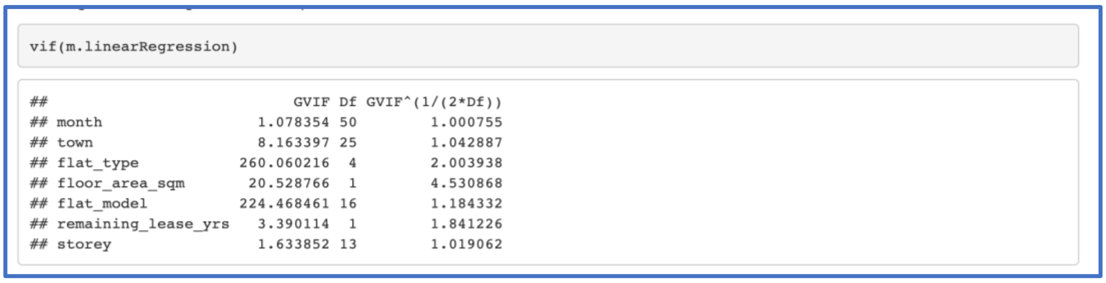

Running the Standard Linear Regression Model

```r
m.linearRegression <- lm(resale_price ~ . , data = trainset)
summary(m.linearRegression)
```

We first build the standard linear model using the function lm() with resale_price as the output variable and the remaining variables as the predictor variables as stated in the question.

To ensure the model is suitable and appropriate, diagnostic checks should always be carried out as there will always other possible complications such as influential outliers and multicollinearity.

Next we run the vif() which gives us the Variance Inflation Factor allowing us to check for multicollinear variables used in our formula. Assuming that multicollinearity exists if VIF > 5, which is the general agreed among statisticians. We will be able to conclude that multicollinearity does not exists in our model. This is important as multicollinearity can weaken the statistical power of our model by wrongly classifying a statistically significant variable as not significant impacting inference.



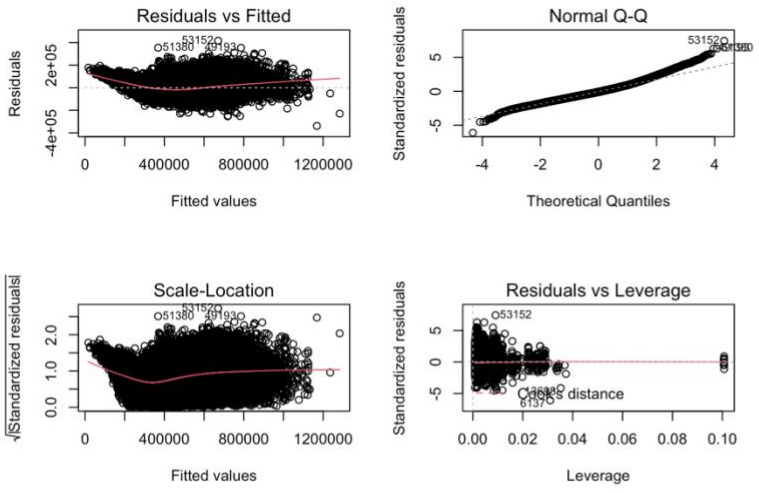
```r
vif(m.linearRegression)
```

```
##                           GVIF Df GVIF^(1/(2*Df))
## month                 1.078354 50        1.000755
## town                  8.163397 25        1.042887
## flat_type           260.060216  4        2.003938
## floor_area_sqm       20.528766  1        4.530868
## flat_model          224.468461 16        1.184332
## remaining_lease_yrs   3.390114  1        1.841226
## storey                1.633852 13        1.019062
```

To further check that linear model is appropriate, we can run diagnostic checks that determines if influential outliers exists using the plot() function. This gives us the diagnostic graphs as seen below.

The Linear Regression model has several assumptions:

(1) Linear Relationship between Xs and Y
(2) Errors have normal distribution with mean 0
(3) Errors are independent of X and has constant standard deviation



The top left chart of Residuals against Fitted values and the bottom left chart of Scale Location can help us analyse the fit of our model. If our assumptions are correct, the points should be spread evenly above and below the dotted line in the top left graph. However, the residuals vs fitted plot might indicate that our model is poorly fitted.

(1) Linearity Violation: Red Line does not appear to be straight appears more like 2nd order polynomial
(2) Residuals are not spread out evenly on the line. This might suggest that the variances of the error terms are not equal.

Looking at the normal quantile-quantile plot on the top right, if the model data do in fact have residuals that follow a normal distribution then all the datapoints in the plot should fall along the straight line. The Q-Q plot helps us test our assumptions that errors should have a normal distribution with mean 0. We can observed in the Q-Q plot that there is some deviations in the tails. The upper end of the Q-Q plot deviates more from the straight line that the lower end, this suggests that the curve has a longer tail to its right and is right skewed (Varshney, 2020).

The last plot of residuals vs leverage tells us that there's no influential outliers.

When these assumptions are violated, this might result in a poor MSE performance, poor accuracy and inference. Given that our linear regression model is poorly fitted and violated the assumptions (Alabanza, 2020).This will result in a high RSME value.

```
library(e1071)
skewness(trainset$resale_price)
```

```
## [1] 1.092454
```

Also from the data exploration part, we got an indication that the distribution of resale_price is highly right skewed. To further verify this, we can use the skewness function in the e1071 package, we are able to find out the skewness off the variable, resale_price. We can use >+1, <-1 which is considered highly skewed as a rule of thumb. We discover that the skewness of the resale_price variable was 1.0924 which is consider highly right skewed.

```
m.log.linearRegression <- lm(log(resale_price) ~ ., data = trainset)
```

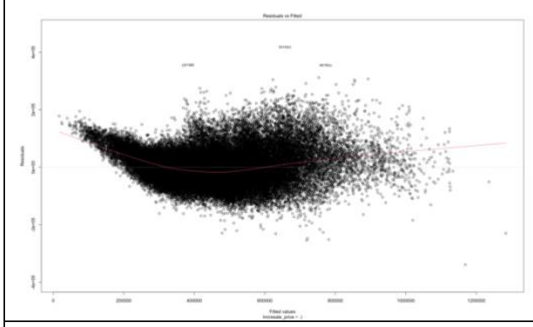
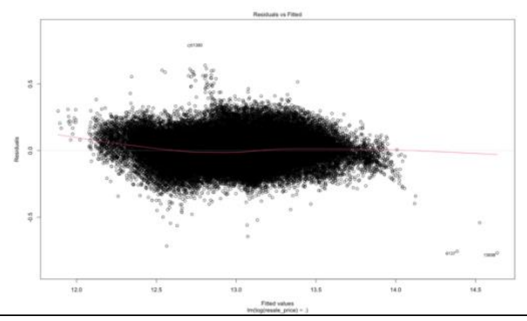
Finding the RSME

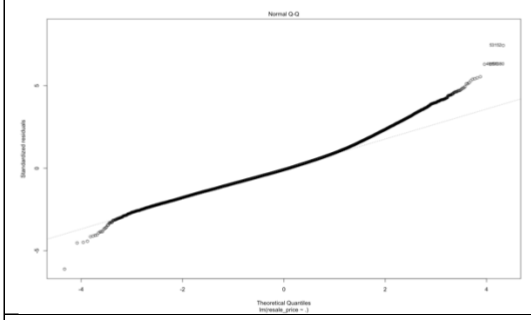
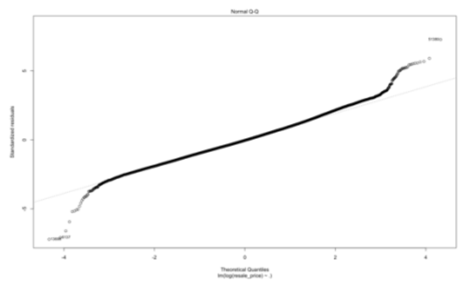
```
# Standard Linear Regression Trainset RSME
m.log.linearRegression.predicted <- exp(predict(m.log.linearRegression, newdata = trainset))
trainset.error <- trainset$resale_price - m.log.linearRegression.predicted
m.log.linearRegression.train.rmse <- round(sqrt(mean(trainset.error^2)))

# Standard Linear Regression Testset RSME
m.log.linearRegression.predicted <- exp(predict(m.log.linearRegression, newdata = testset))
testset.error <- testset$resale_price - m.log.linearRegression.predicted
m.log.linearRegression.test.rmse <- round(sqrt(mean(testset.error^2)))
```

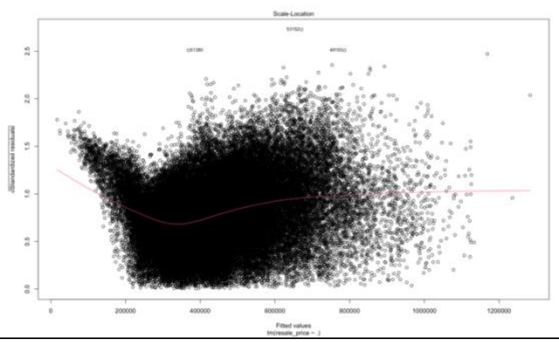
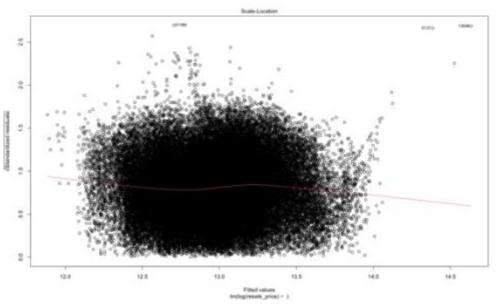
Writing Log Linear Regression RSME results into results table

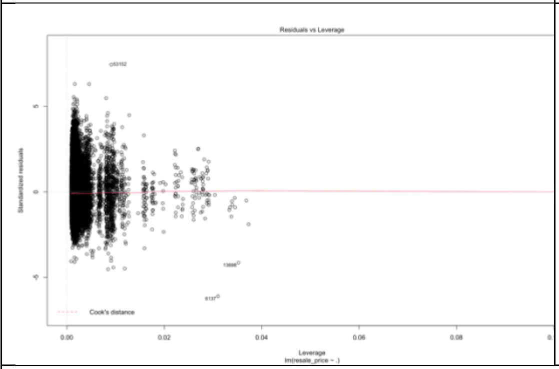
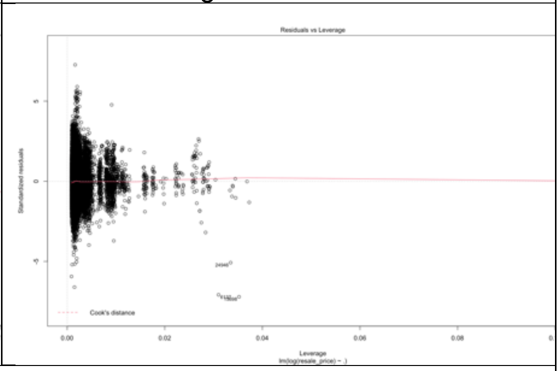
```
results[nrow(results) + 1, ] <- c("Log Linear Regression", m.log.linearRegression.train.rmse, m.log.linearRegress
ion.test.rmse)
```

Thus, to improve on the linear regression, log transformed was applied to the dependent variable (i.e, log(resale_price)) when running the linear regression which can help to normalise the data and reduce skewness. The log linear regression resulted in much better RMSE performance as compared to the standard linear regression. Let’s compare the differences in in the log transformed and standard linear regression.

Residuals vs Fitted	
Non-Transformed	Log-Transformed
	
Explanation	
As explained, in the earlier parts, non-transformed linear regression do not meet the assumptions of the linear regression model, violating the linearity assumption and errors having a constant standard deviation. As observed, after log transforming the linear regression, there was a straighter red line with the data points being more equally spread out, meaning the linearity assumptions are more likely to be met after log transforming the model.	

Normal Q-Q	
Non-Transformed	Log-Transformed
	
Explanation	
Similarly, in the earlier part, it has been proven in the standard linear regression before log transforming the assumptions of errors have normal distribution with mean 0 is being violated. Upon log transforming, the assumption of normally distributed residuals holds more strongly as the residuals deviates less and falls more closely on the straight line upon log transformations.	

Scale Location	
Non-Transformed	Log-Transformed
	
Explanation	
In the non-transformed model, the residuals do not have constant standard deviation and are definitely not independent of the predictors. Upon log transformation, the residuals are much more evenly spread out in the log transformed model and deviates less from a straight line. Thus the linear regression assumptions hold more strongly upon log transformation.	

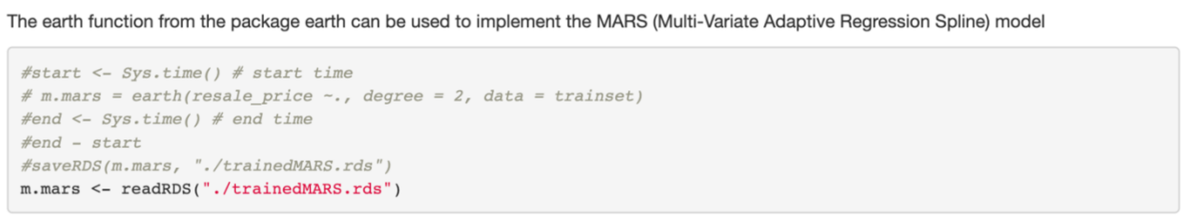
Residuals vs Leverage	
Non-Transformed	Log-Transformed
	
Explanation	
No influential outliers in either models	

The log regression holds the assumptions of a linear regression model much better and this should be use in place of the standard linear regression model. Actual sales price can be obtained using $e^{\log(resale\_price)}$ as the model will predict log(resale_price).

(ii) MARS degree 2

In part(i), we showed that the standard linear regression model might not be a good fit for the current dataset as there were violations to the given assumptions. Though the log transformed linear regression performed better, there might be a possibility that non-linear patterns can occur in the data and MARS can be used. Thus, a MARS model with degree 2 is being carried out to find out if there lies non-linear patterns in our data where MARS might be a better fit that the linear regression model.

The MARS model is a weighted combination of the hinge functions, it is likely to perform better than the standard linear regression. MARS fits local hinge functions adaptively i.e., not global trends.


The earth function from the package earth can be used to implement the MARS (Multi-Variate Adaptive Regression Spline) model

```r
#start <- Sys.time() # start time
# m.mars = earth(resale_price ~., degree = 2, data = trainset)
#end <- Sys.time() # end time
#end - start
#saveRDS(m.mars, "./trainedMARS.rds")
m.mars <- readRDS("./trainedMARS.rds")
```

We can save the model into an R object file so that we do not need to run the model again.

(iii) Random Forest

The Random Forest is a supervised learning algorithm that builds multiple decision trees that are trained with a bagging method (Bootstrap Aggregation).

Begin by running the standard Random Forest using the default settings

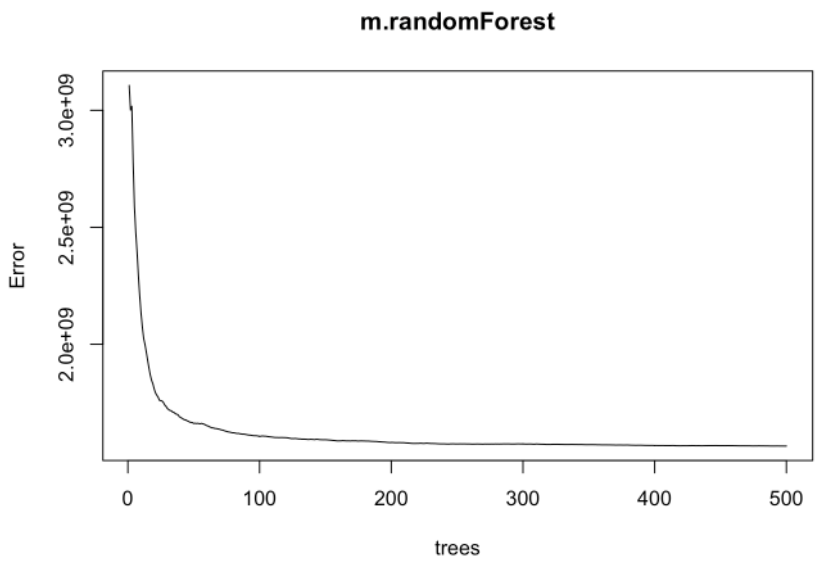
```
#start <- Sys.time() # start time
#m.randomForest <- randomForest(resale_price ~ ., data = trainset, importance = T)
#Saving the trained model to disk
#saveRDS(m.randomForest, "./trainedRF.rds")
#end <- Sys.time() # end time
#end-start
#save.image()
```

Load the trained model for use

```
m.randomForest <- readRDS("./trainedRF.rds")
```

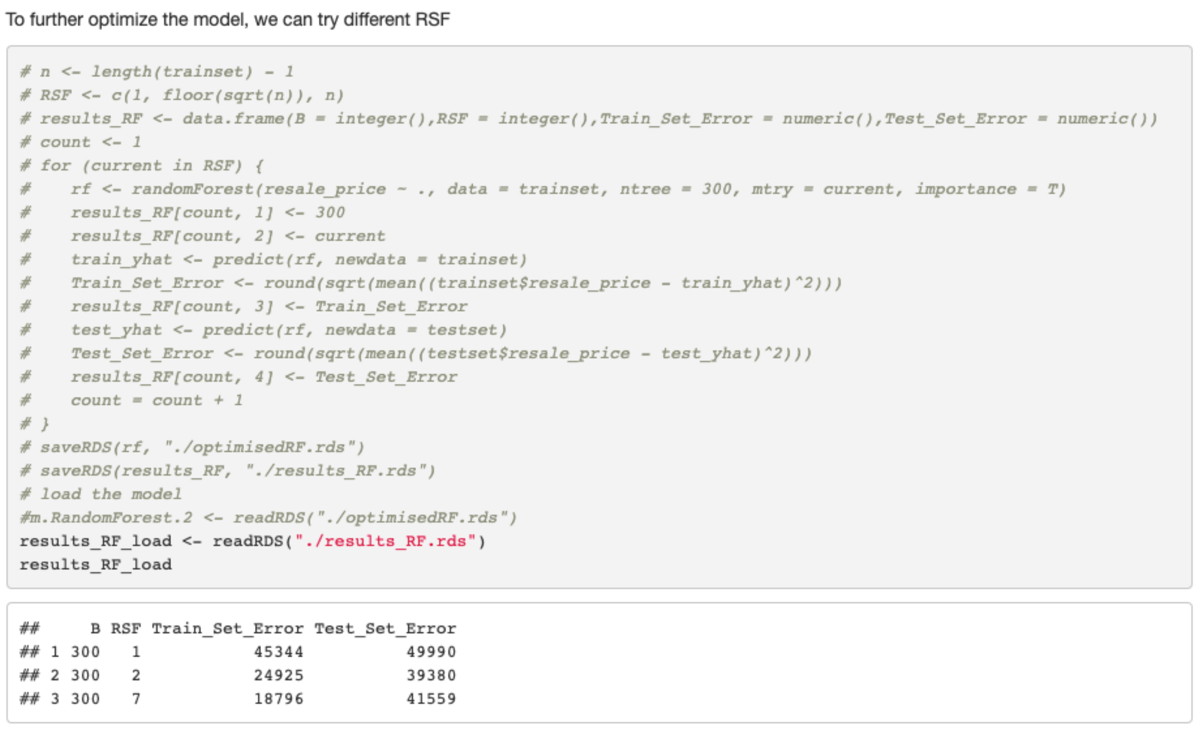
We first built the random forest based of the default parameters in R.

```
plot(m.randomForest)
```



The error rate stabilized after 300 trees, so we can set the number of trees to be used in the model as 300.

Plotting the model obtained, we observe that the error rate stabilised after 300 trees, so I set the number of tree to be used in the model tuning as 300 as a larger number of trees will also cause a longer run time. We can try to evaluate the best number of variables in each split mtry since the default value of mtry is the floor(#variables/3)


To further optimize the model, we can try different RSF

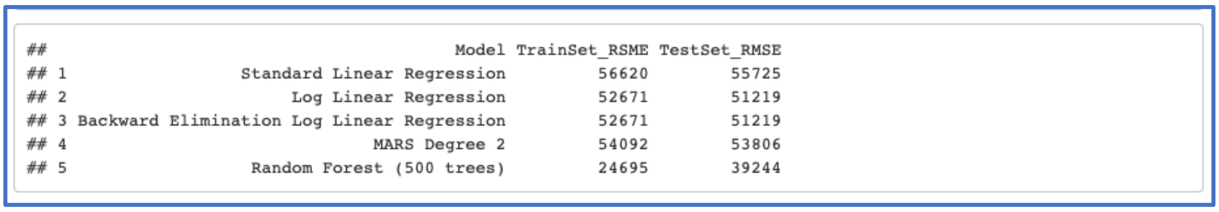
```
# n <- length(trainset) - 1
# RSF <- c(1, floor(sqrt(n)), n)
# results_RF <- data.frame(B = integer(),RSF = integer(),Train_Set_Error = numeric(),Test_Set_Error = numeric())
# count <- 1
# for (current in RSF) {
#    rf <- randomForest(resale_price ~ ., data = trainset, ntree = 300, mtry = current, importance = T)
#    results_RF[count, 1] <- 300
#    results_RF[count, 2] <- current
#    train_yhat <- predict(rf, newdata = trainset)
#    Train_Set_Error <- round(sqrt(mean((trainset$resale_price - train_yhat)^2)))
#    results_RF[count, 3] <- Train_Set_Error
#    test_yhat <- predict(rf, newdata = testset)
#    Test_Set_Error <- round(sqrt(mean((testset$resale_price - test_yhat)^2)))
#    results_RF[count, 4] <- Test_Set_Error
#    count = count + 1
# }
# saveRDS(rf, "./optimisedRF.rds")
# saveRDS(results_RF, "./results_RF.rds")
# load the model
#m.RandomForest.2 <- readRDS("./optimisedRF.rds")
results_RF_load <- readRDS("./results_RF.rds")
results_RF_load
```

```
##     B RSF Train_Set_Error Test_Set_Error
## 1 300   1           45344          49990
## 2 300   2           24925          39380
## 3 300   7           18796          41559
```

By recording the train and test set error on each different mtry value, the table above was obtained. The initial value of mtry = 2 was still the most appropriate model as it returns the lowest test set error. Thus the final model was built with mtry = 2 and ntrees = 500 instead of 300 as in general the more trees used the higher accuracy of the model though with diminishing returns.

To better tune the model in the future, more advance tools like grid search can be used.

Evaluation on Model's Performance



```
##                                     Model TrainSet_RSME TestSet_RMSE
## 1              Standard Linear Regression         56620        55725
## 2                   Log Linear Regression         52671        51219
## 3 Backward Elimination Log Linear Regression      52671        51219
## 4                           MARS Degree 2         54092        53806
## 5                Random Forest (500 trees)         24695        39244
```

Overall, the rankings base of test set accuracy are as follows

<u>Best Testset RMSE</u>

Random Forest
Log Linear Regression
MARS
Standard Linear Regression

<u>Worst Testset RMSE</u>

The random forest appears to be the best model out of all the models that have been tested and employed as it returns the lowest RMSE error.

However, it can also be observed the testset RMSE is significantly higher than the trainset RMSE and this might be an indicator that overfitting might have occurred in our random forest model. Random forest has been observed to overfit for dataset perhaps due to noisy dataset or even random fluctuations. To avoid overfitting, hyperparameters of the model could be tuned, have more predictor variables or even a larger dataset to achieve better prediction accuracy.

There is also a severe limitation to the random forest model for regression problems. Random Forest models unable to extrapolate beyond the data (Mwiti, 2021). For example, the range of the values of the resale_price in this dataset is 140000 – 1258000. The highest value the model can predict is 1258000, it cannot extrapolate the trends and accurately predict new values that have never been seen in the training data as compared to another model like linear regression. This will be a problem if we want to use the model to predict resale_price. Currently, the limitation is not a huge issue in this question as the goal of the question assumed is to be to compare the predictive performance across the models.

Log Linear regression performed marginally better than MARS. This might be due to statistical fluctuations and can be improved upon by using a bigger dataset with more variables. Thus, the difference is not that significant.

Evaluation on Question

There could be two plausible interpretation of “predictive performance” to this question/paper stated in the intro.

(1) The problem defined in the question could be the predictive performance of the models for future use. This means that the predictive models adopted will be used to predict future HDB resale prices. If the problem is being defined as such, “month” will not be a suitable predictor variable and the column can be dropped. This is because, for example, if we want to predict the price of the flat in May 2022 using the linear regression model (for which the question told us to use all predictor variable), it will not be possible as there will not be level “2022-05” in the dataset for the current model to be used as a predictor. Thus, it’s worth noting that currently all models built have limited practical use.

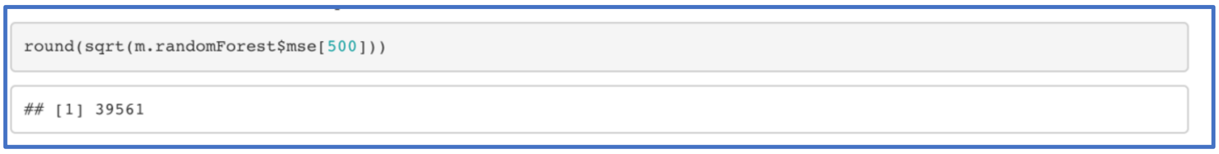
(2) The goal of this question could also be to find out the comparism of predictive performance of the 3 models. If this interpretation is adopted, “month” can then be kept as a predictor variable to ensure the same variables are used in the comparisms across the models adopted.

Since the question told us to use all variables as predictors in the linear regression model, it is more likely that we are finding out the relative performance across the models instead of choosing a model to predict the resale_price. Thus, in my approach above, “month” was not dropped from the models due to the assumption that the problem is defined as the one in (2).

Answer to Q5:

What is the OOB RMSE of the Random Forest? Can this be used as the estimate of Random Forest performance instead of testset RMSE? Explain.

The OOB RMSE can be obtained from random forest model and taking the square root of Mean squared residuals



```
round(sqrt(m.randomForest$mse[500]))
```

```
## [1] 39561
```

The OOB RMSE of the final random forest model is 39561 which is very close to the testset RMSE of 39244.

Before we evaluate which is a better choice of evaluation metrics, OOB RMSE or testset RMSE, let’s first understand what is the OOB RMSE of the Random Forest.

What is OOB RMSE of Random Forest?

The random forest models builds a number/forest of decision trees on bootstrapped training samples which is obtained sampling the training set with replacement. Each of the bootstrapped samples are used to train an individual tree. A large number of trees are built by training on the bootstrap samples. Data points that were not used to build the tree(s) in the ensemble are known as the out of bag samples.

The OOB error can be measured by obtaining of the prediction error on the training set by only including in the calculation of a row’s error trees where that row was not included in training. In a regression problem like ours, we obtain a single prediction for the *i*th observation by averaging the predicted response from the different trees, giving us a single OOB prediction for the *i*th observation. Following which, we can do this for *n* observations we have in the OOB sample set, from which the overall OOB MSE can be calculated. RMSE of the OOB simply refers to the square root of the OOB MSE (Hastie et al., 2009, pp. 1–3). The OOB MSE is calculated by subtracting the average of all predicted outputs from trees and actual data.

Since the response for each observation is predicted on the tree that were not trained on that observation (unseen data), OOB error will this be a valid estimate of the test error for the bagged model.

<u>When should OOB RMSE be used?</u>

1. Efficient, Natural test set and Computationally Advantageous

The OOB prediction provides a native and convenient estimate of prediction error without having the need to use n-fold cross validation error or train-test split. The OOB RMSE has a computational advantage as we are able to fit and test the random forest at the same time when training the model. The value and be computed automatically by the randomForest() function in the R program. This also helps reduce the additional time needed to test on an separate independent test set.

2. Good for small datasets

Using the OOB RMSE is also useful when the dataset is small. When the dataset is small further splitting the data will result in the random forest having less training data which might inevitably affect the predictive power of the random forest model. Thus the OOB RMSE will definitely be helpful in generating useful statistic, when we will only have a small amount of training data. It allows us to see how the model generalise without have to separate the already small dataset to create a separate validation set, thus model can still be fairly evaluated.

<u>When shouldn't OOB RMSE be used?</u>

1. **Cannot compare Random Forest Model performance to other types of Model**

The benchmark for Random Forest model is the use of the OOB error instead of the usual test set error. Thus, in the context of only evaluating one model - the Random Forest model only, the OOB RMSE if the Random Forest remains as a suitable metric. However, as the OOB estimates can only be found in Random Forest it's not a suitable metrics used for comparison of models. To compare models, we will need to ensure that the same metrics is being used to ensure fairness. Due to the implementation of the model itself, this metric is not available on other models such as linear regression or MARS.

Thus, performing train test split to obtain the testset RMSE will be useful and necessary if we would like to perform comparisms with the other models.

<u>Evaluation: Can OOB RMSE be used in place of testset RMSE?</u>

In conclusion, yes, OOB RSME can be used as an estimate of the Random Forest performance instead of testset RMSE depending on the context of the problem defined.

However, as set out earlier in the evaluation section of question 4 the approach taken in this paper is that we would like to do a comparisms of the predictive accuracy across the models on HDB resale price. To do this, we will need to ensure that the same evaluation metric is being used across the models to ensure fairness and standardize the process. Thus in this case, testset RSME will be a better choice as an evaluation metrics compared to the OOB RMSE. In fact, OOB RMSE cannot be used at all in our case despite it being advantageous as it does not help us in answering the questions we like to answer. Also, given that our dataset contains over 90,000 rows, the issue of small dataset doesn't really exists and thus OOB RMSE is not significantly advantageous either.

However, if the problem is being defined as, using one model, Random Forest Model for predicting the future resale prices of HDBs, OOB RSME estimates can be a suitable metrics that we can use.

Answer to Q6:

In Soifua (2018) report , Gini-based variable importance was used instead of Accuracy-based variable importance via permutation. What are the Pros and Cons of using Gini-based variable importance? Is this better than using Accuracy-based variable importance?

To make a fair comparisms of whether Gini-based variable importance should be used or Accuracy-based variable importance (Permutation-based variable importance). We will need to understand the pros and cons of both approaches.

What is Permutation-based Variable Importance?

Permutation Importance is calculated by the decrease in the model score when a single feature value is randomly shuffled. This is done upon the random shuffling of a single feature and checking the increase in error due to the permutation. Permutation breaks the relationship between the feature and target thus when there is a sharp fall in the model score that means that feature is important in the model prediction (Permutation Feature Importance — Scikit-Learn 0.24.1 Documentation, n.d.).

Pros of Permutation-based Variable Importance

1. Model Agnostic

Permutation Importance algorithm is model agnostic, this means that the model can be used for any machine learning pipeline (Wilcox, 2020). This is a model inspection technique that can be used for any fitted, non-linear or opaque estimator (Permutation Feature Importance — Scikit-Learn 0.24.1 Documentation, n.d.).

Cons of Permutation-based Variable Importance

1. Extremely Computationally Intensive

When multiple iterations of permutation importance are ran especially on high dimensional data such as in Biostatistics, genomics problem, this will cause an increase in the run time. This is because there will be a large number of combinations to permutate and test (Płoński, 2020). The increase in computational overhead will be significant and is an important factor to be taken into consideration. Compared to the Gini importance measure, permutation importance is much more computationally expensive.

2. Misleading results when Predictor Variables (Features) are Significantly Correlated

There have been several studies that states that when the features in the training set are significantly correlated and statistically dependent on each other, the use of permutation methods can lead to misleading results (Hooker, G., & Mentch, L, 2019).

For example, given feature A and feature B that’s both significantly correlated. Permutating feature A will cause only a small decrease in the performance as there are similar information in feature B and model still have access to the correlated feature.

Thus, permutation importance severely discounts and punishes correlated features by assigning low importance values reliable (Strobl, C., Boulesteix, AL., Kneib, T. et al, 2008) though the features might actually be important. To ensure that permutation importance is used correctly, data pre-processing might be needed handle the multi-collinearity before running the permutation importance.

What is Gini-Based Variable Importance?
Gini-based variable importance is defined as the total decrease in node impurity averaged over all trees in the ensemble. This means that a split with a huge decrease in impurity will be considered important so is consequent variables used for splitting at important splits and vice versa. The impurity importance will then be computed by the sum of all impurity decrease measurement of all nodes in the forest which a split has been conducted (Nembrini et al., 2018, p. 3713)

<u>Pros of Gini-based variable importance</u>

1. Gini based variable importance has been shown to be "sensitive to predictors with different scales of measurement (e.g., binary versus continuous)" (Nicodemus K. K, 2011)

2. **Less Computationally Intensive**

Relative to the permutation importance, Gini variable importance is less computationally intensive (Nembrini et al., 2018, p. 3713).

<u>Cons of Gini-based Variable Importance</u>

1. **Biased Variable Selection towards High Cardinality Features**

Gini-based variable importance has showed artificial inflation for predictors with larger numbers of categories. In many research papers, Gini importance have been found to be biased estimator when the data set contains predictor variables with many categories (variables with high category frequencies) or continuous variables are being used (Strobl et al, 2007 and Strobl et al, 2008). These high frequencies categorical variables are often found genomic and bioinformatics problems for example single-nucleotide polymorphisms (SNPs).

Gini Variable Importance is calculated based on the variables that are selected for splitting, the variables selected for the next split is the one the will produce the highest criterion value overall i.e best cut-off. Thus, variable with many categories, with more potential cut-offs has a higher probability and chance to produce a split with high reduction in impurity than a variable with far less split points even when both variables are not associated with the outcome. (Nembrini et al., 2018, p. 3713).

For example, if we compare variable with 2 category which provides only one split vs a variable that has four categories that provide 7 splits, the latter with give a better Gini impurity value. As the number of split points increases exponentially with the number of categories of unordered categorical predictors, there will always be a preference for variables with more categories (Nembrini et al., 2018, p. 3713). Thus, variables that offer many cut points systematically obtain higher Gini importance scores (Strobl et al. 2007).

This biasness is not relevant to permutation-based variable importance.

2. **Inability to Generalise to TestSet**

   According to Scikit-Learn (Scikit-Learn 0.24.1 Documentation, n.d.), with impurity importance being computed on training set statistics, it is thus unable to reflect how useful the feature is in making predictions that generalise to the test set.

Conclusion
(Is Gini-based importance better than using Accuracy-based variable importance?)

Whether Gini-based importance should be used or accuracy-based should depends entirely from the dataset. From the evaluation on the pros and cons of both Gini and Accuracy based importance, 3 key findings have been derived (Nicodemus K. K. , 2011).

(1) When the predictor variable in the dataset contains high frequencies of categories/high cardinality features, the accuracy-based method may be preferred.
(2) When the dataset shows significant within-predictors correlation, Gini importance might be a more appropriate measurement technique as the method is more sensitive to within-predictor correlation.
(3) Depending on the application, if a fast response if needed Gini importance method would be preferred over the Accuracy based importance.

Thus, perhaps, a conclusion on which method is more suitable can only be made when data-specific characteristics are understood.

However, in most situation Accuracy-based/Permutation variable importance is preferred over variable importance. It will be easier to resolve collinearity problems in the dataset/model such as using improved models like the cforest() function that works based on conditional permutation importance (Arbor, 2018). These functions are already available and implemented in programming software like R. There is also the added advantage of it being applicable to any model class not just tree-based models.

Whereas the problems with Gini variable importance is innate to the inner workings of the model and might be harder to be resolved. In recent years, there have been new methods to resolve this issue such as the use of corrected impurity importance measure (AIR) (Hooker, G., & Mentch, L, 2019). However, this does not seem be available in implementation software yet. It will also be easier to use permutation than trying to find out if the dataset conforms to what is being needed to use Gini variable importance and to interpret the internal model parameters (Parr et al., 2018).

Regardless, any interpretation of random forest variable importance can only happen when the number of trees chosen is sufficiently large so that the results that is yield remains similar. Only then, we can be assured that of the differences in results is due to the method and not random variations.

Answer to Q7:

Soifua (2018) report Appendix B mentioned the idea of weighting each tree in the random forest by their respective OOB error to improve model performance. However, it was "concluded that the improvements were too modest to be worthwhile..." Suggest a reason based on your understanding of random forest

In random forest, individual trees are trained on the in bag samples and predicted using the out of bag samples. In the conventional Random Forest, all trees are weighted equally. In the weighted random forest trees are penalised according to their OOB error rates. When Soifua (2018) mentioned that the weighing scheme was unsuccessful and "concluded that the improvements were too modest to be worthwhile..." this suggest that the weighted random forest did not result in significant changes in predictive power and thus, there is little advantage given the added complexity of introducing performance based weights. The reason for this will be explained below.

Weighted Random Forest being an Unfair Algorithm

An unfair algorithm is one where decisions are skewed to one particular group. The weighted random forest algorithm is an unfair algorithm in itself. By using weights derived from OOB errors and then using it to penalise the trees, is unfair as the OOB error only considers how well the weights fits into that particular tree.

In Random forest, some trees might have better predictive performance on the OOB samples because coincidentally they might be trained on similar observations (referring to the in bag samples) they are being trained on. Thus, it will appear the tree is accurate and given a high weight. Similarly, other OOB samples might be hard to predict correctly by the tree because they did not have similar observations that were used in the in-bag samples. This might not mean then the tree does not fit the data well either.

This will unnecessarily penalises tree that perform well for specific variable types/data. (Gajowniczek, K., Grzegorczyk, I., Ząbkowski, T., & Bajaj, C. , 2020). On the same vein, each tree will have thus different ability in predicting each OOB sample. There is no common test set for weighing the tree.

In reality, unseen data will fit very differently from the data the tree was being trained on. Although OOB errors might make it appear that a tree is making accurate predictions but it might not actually be the case. Thus, using weighted Random Forest will incorrectly give this tree more weight. When unseen data is given to the random forest. The prediction will not perform well with unseen data though it performs well with seen/training data.

By weighing the tree using OOB error, lower weight will be given to trees who has high OOB error though in actual fact is that the tree might be a good and accurate predictor of unfamiliar data and poor predictor of OOB data. More weightage will be given to tree which has low OOB error likely good predictor of OOB data and familiar data. Thus during aggregation, the forest of trees will be good at predicting OOB data that it has seen before and not so much of unfamiliar data resulting in little improvements in the model's predictive power

Bibliography

Alabanza, H. (2020, September 8). *The Consequences of Violating Linear Regression Assumptions*. Medium. https://towardsdatascience.com/the-consequences-of-violating-linear-regression-assumptions-4f0513dd3160

Arbor, M. L. C. O. A. (2018, June 20). *Be Aware of Bias in RF Variable Importance Metrics*. Pi: Predict/Infer. https://blog.methodsconsultants.com/posts/be-aware-of-bias-in-rf-variable-importance-metrics/

Brownlee, J. (2020, June 30). *How to Perform Data Cleaning for Machine Learning with Python*. Machine Learning Mastery. https://machinelearningmastery.com/basic-data-cleaning-for-machine-learning/

Brownlee, J. (2019, October 25). *Gentle Introduction to the Bias-Variance Trade-Off in Machine Learning*. Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction-to-the-bias-variance-trade-off-in-machine-learning/

Gajowniczek, K., Grzegorczyk, I., Ząbkowski, T., & Bajaj, C. (2020). Weighted Random Forests to Improve Arrhythmia Classification. Electronics, 9(1), 10.3390/electronics9010099. https://doi.org/10.3390/electronics9010099

Hooker, G., & Mentch, L. (2019). Please Stop Permuting Features: An Explanation and Alternatives. ArXiv, abs/1905.03151.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)* (2nd ed.). Springer. https://link.springer.com.remotexs.ntu.edu.sg/content/pdf/10.1007/978-0-387-84858-7.pdf

Mwiti, D. (2021, April 9). *Random Forest Regression: When Does It Fail and Why?* Neptune.Ai. https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why

Nicodemus K. K. (2011). Letter to the editor: on the stability and ranking of predictors from random forest variable importance measures. Briefings in bioinformatics, 12(4), 369–373. https://doi.org/10.1093/bib/bbr016

Nembrini, S., König, I. R., & Wright, M. N. (2018). The revival of the Gini importance? *Bioinformatics*, *34*(21), 3711–3718. https://doi.org/10.1093/bioinformatics/bty373

Parr, T., Turgutlu, K., Csiszar, C., & Howard, J. (2018, March 26). *Beware Default Random Forest Importances*. Explained.Ai. https://explained.ai/rf-importance/

Permutation feature importance — scikit-learn 0.24.1 documentation. (n.d.). Scikit-Learn.Org. https://scikit-learn.org/stable/modules/permutation_importance.html#:%7E:text=The%20permutation%20feature%20importance%20is,model%20depends%20on%20the%20feature.

Płoński, P. (2020, June 29). Random Forest Feature Importance Computed in 3 Ways with Python. MLJAR Automated Machine Learning. https://mljar.com/blog/feature-importance-in-random-forest/

Strobl, C., Boulesteix, AL., Zeileis, A. et al. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. BMC Bioinformatics 8, 25 (2007). https://doi.org/10.1186/1471-2105-8-25

Strobl, C., Boulesteix, AL., Kneib, T. et al. (2008) Conditional variable importance for random forests. BMC Bioinformatics 9, 307. https://doi.org/10.1186/1471-2105-9-307

Varshney, P. (2020, October 18). Q-Q Plots Explained - Towards Data Science. Medium. https://towardsdatascience.com/q-q-plots-explained-5aa8495426c0

Wilcox, P. (2020, March 2). Paul Wilcox. Lucena Research. https://lucenaresearch.com/2020/03/02/selecting-features-with-permutation-importance-2/