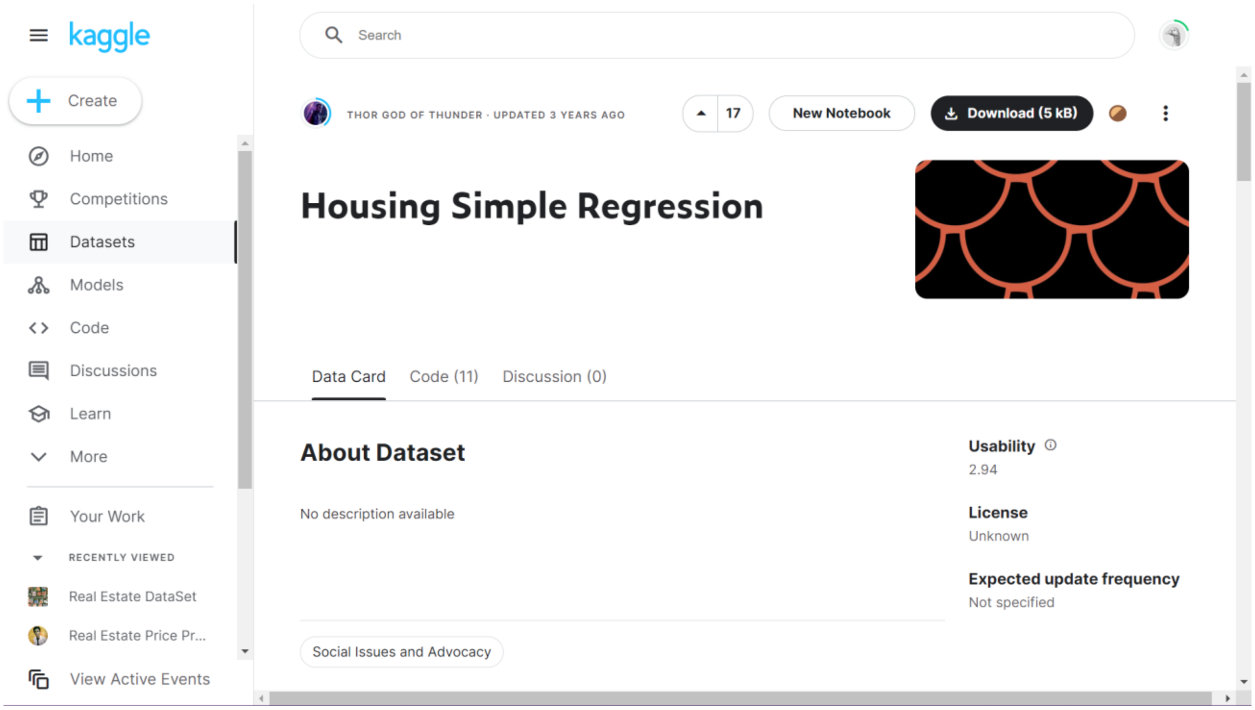
M1 - SA Chapter Project (CART and Decision Tree)

For this chapter project, researchers were tasked to create a CART and Decision Tree model that will process real estate/housing data. To do this, the appropriate data from the internet must first be downloaded. Afterwards, they will identify dependent and independent variables, split the data into training and testing sets to ensure that the model can accurately predict output on new data, and test the data to evaluate the model. To fine-tune the decision tree, hyperparameters related to depth and the number of observations on each leaf will be adjusted until optimal results are achieved. Finally, to understand the results, the data will be represented through a confusion matrix.

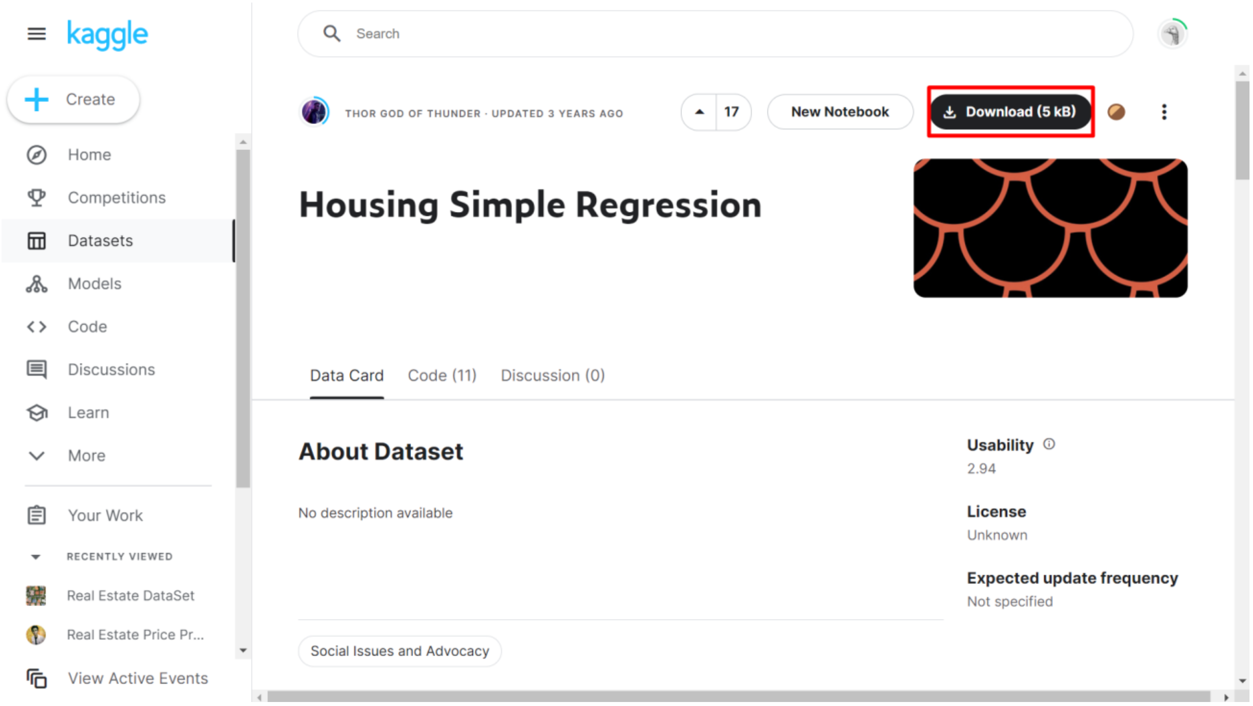
I. Finding the Dataset

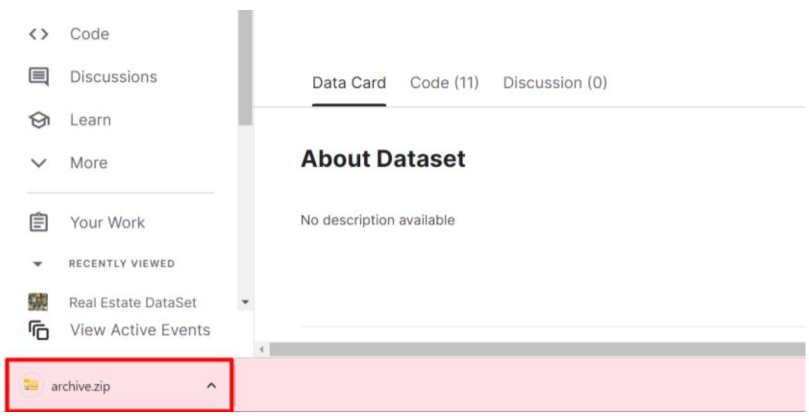
The dataset used in this chapter project was required to fulfill the criterion of being related to real estate or housing. Therefore, the data scientists utilized an open-source platform to find the dataset that will match that requirement. Using Kaggle, we searched for real estate datasets that could predict specified dependent variables and eventually came across this dataset. The author of this dataset also includes a corresponding code to perform Exploratory Data Analysis on the data; therefore, we also used that as a reference to help us understand the dataset further.

The figure below shows the window that will appear after selecting the hyperlink above.

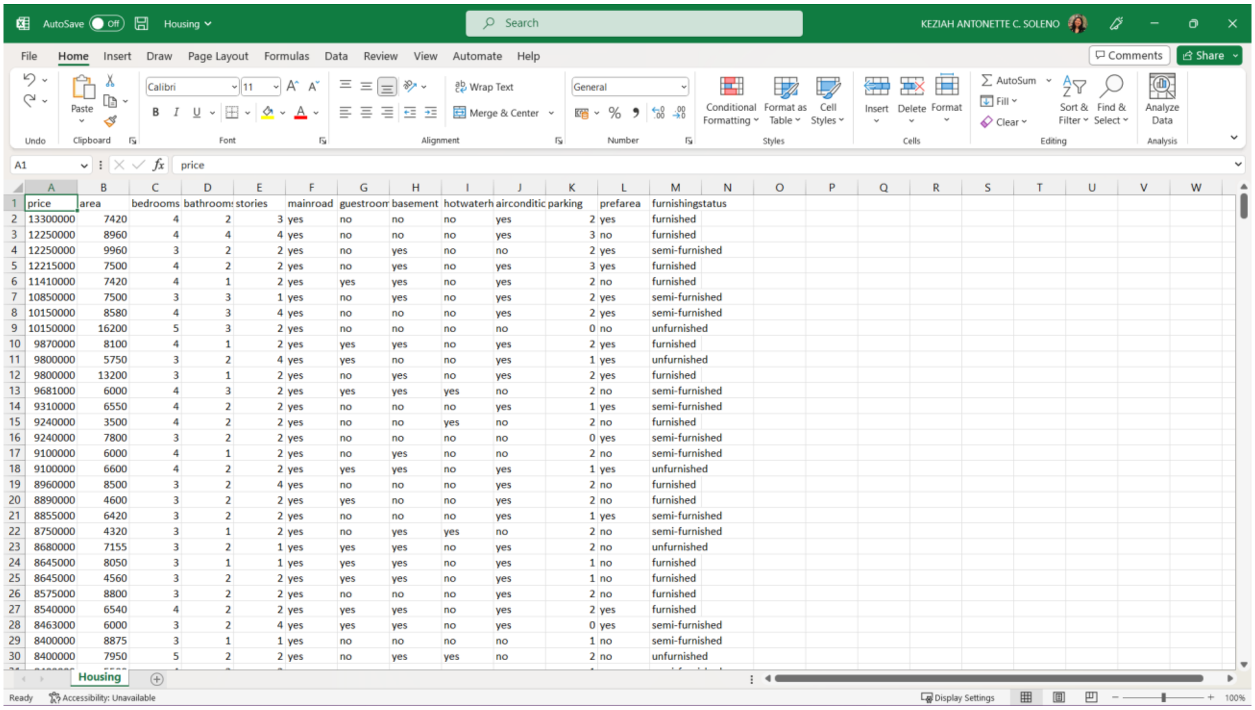


To download the dataset, we selected the highlighted region on the figure below. The .csv file should download shortly after, as indicated in the succeeding screenshot.





Extracting the downloaded file and selecting the *Housing.csv* file should reveal the figure below.



price	area	bedrooms	bathroom	stories	mainroad	guestroom	basement	hotwaterh	airconditic	parking	prefarea	furnishingstatus
13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
10850000	7500	3	3	1	yes	no	yes	no	yes	2	yes	semi-furnished
10150000	8580	4	3	4	yes	no	no	no	yes	2	yes	semi-furnished
10150000	16200	5	3	2	yes	no	no	no	no	0	no	unfurnished
9870000	8100	4	1	2	yes	yes	yes	no	yes	2	yes	furnished
9800000	5750	3	2	4	yes	yes	no	no	yes	1	yes	unfurnished
9800000	13200	3	1	2	yes	no	yes	no	yes	2	yes	furnished
9681000	6000	4	3	2	yes	yes	yes	yes	no	2	no	semi-furnished
9310000	6550	4	2	2	yes	no	no	no	yes	1	yes	semi-furnished
9240000	3500	4	2	2	yes	no	no	yes	no	2	no	furnished
9240000	7800	3	2	2	yes	no	no	no	no	0	yes	semi-furnished
9100000	6000	4	1	2	yes	no	yes	no	no	2	no	semi-furnished
9100000	6600	4	2	2	yes	yes	yes	no	yes	1	yes	unfurnished
8960000	8500	3	2	4	yes	no	no	no	yes	2	no	furnished
8890000	4600	3	2	2	yes	yes	no	no	yes	2	no	furnished
8855000	6420	3	2	2	yes	no	no	no	yes	1	yes	semi-furnished
8750000	4320	3	1	2	yes	no	yes	yes	no	2	no	semi-furnished
8680000	7155	3	2	1	yes	yes	yes	no	yes	2	no	unfurnished
8645000	8050	3	1	1	yes	yes	yes	no	yes	1	no	furnished
8645000	4560	3	2	2	yes	yes	yes	no	yes	1	no	furnished
8575000	8800	3	2	2	yes	no	no	no	yes	2	no	furnished
8540000	6540	4	2	2	yes	yes	yes	no	yes	2	yes	furnished
8463000	6000	3	2	4	yes	yes	yes	no	yes	0	yes	semi-furnished
8400000	8875	3	1	1	yes	no	no	no	no	1	no	semi-furnished
8400000	7950	5	2	2	yes	no	yes	yes	no	2	no	unfurnished

II. Preparing the RStudio Environment

The following dependencies and libraries were installed and ran on RStudio:

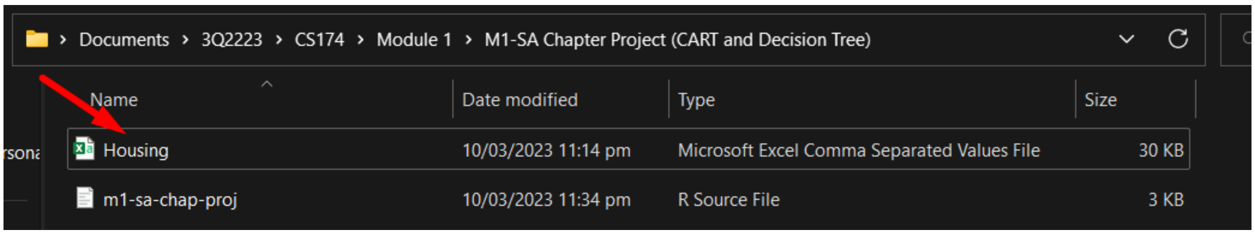
```r
install.packages("dplyr")        library(dplyr)
install.packages("rpart")        library(rpart)
install.packages("rpart.plot")   library(rpart.plot)
install.packages("Metrics")      library(Metrics)
install.packages("mlr")          library(mlr)
install.packages("ggplot2")      library(ggplot2)
install.packages("plotly")       library(plotly)
install.packages("magrittr")     library(magrittr)
install.packages("caTools")      library(caTools)
install.packages("ggcorrplot")   library(ggcorrplot)
install.packages("corrplot")     library(corrplot)
```

- *dplyr* to make data manipulation easier
- *rpart* to provide a framework for growing decision trees without tuning
- *rpart.plot* to help plot our decision tree
- *Metrics* to implement metrics for our model
- *mlr* to adjust the hyperparameters of the model
- *ggplot2* for plotting correlation and accuracy
- *plotly* to make plots in 3-D

After installing the required packages and their dependencies, a working directory was set to store all necessary datasets and files that will be used in the program. The following code snippet was run to do so.

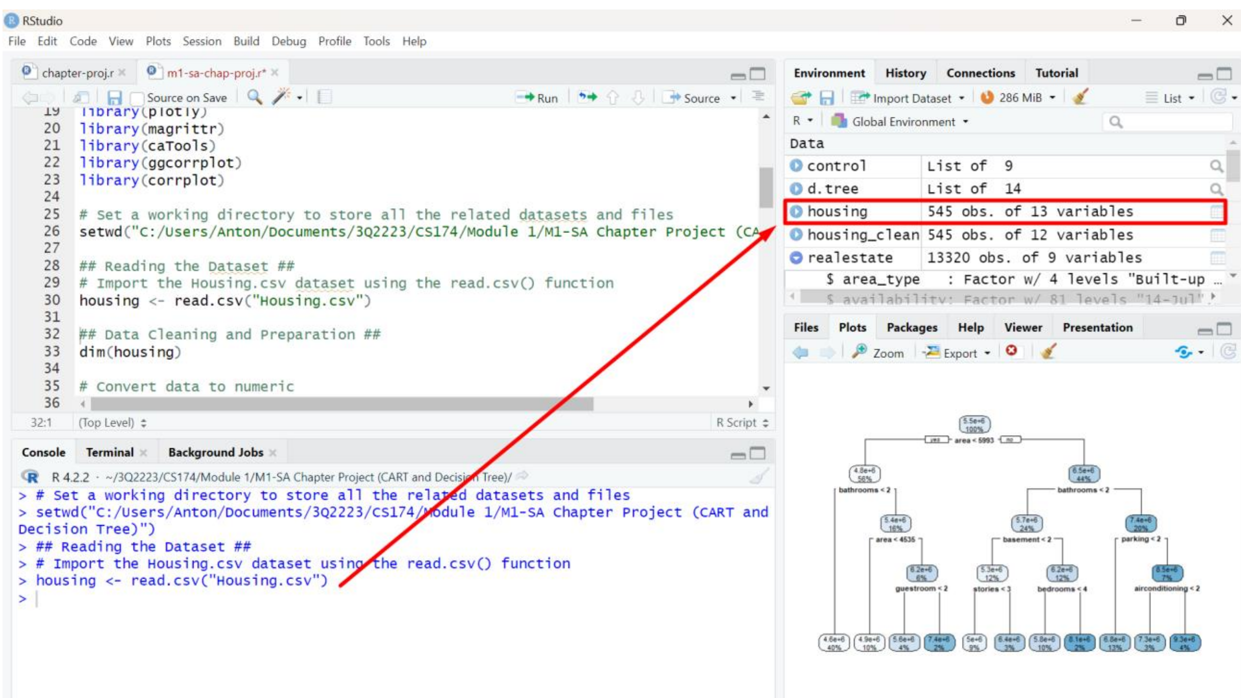
```
> # Set a working directory to store all the related datasets and files
> setwd("C:/Users/Anton/Documents/3Q2223/CS174/Module 1/M1-SA Chapter Project
(CART and Decision Tree)")
```

Before we imported the downloaded dataset, we made sure to store a copy of the dataset in the specified directory, as indicated below.



Next, we imported the dataset using the **read.csv() function** of R and stored that into the data frame *realestate*.

```
> housing <- read.csv("Housing.csv")
```



RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

chapter-proj.r | m1-sa-chap-proj.r* | housing

Filter

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
1	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
2	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
3	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
4	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
5	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished
6	10850000	7500	3	3	1	yes	no	yes	no	yes	2	yes	semi-furnished
7	10150000	8580	4	3	4	yes	no	no	no	yes	2	yes	semi-furnished
8	10150000	16200	5	3	2	yes	no	no	no	no	0	no	unfurnished
9	9870000	8100	4	1	2	yes	yes	yes	no	yes	2	yes	furnished
10	9800000	5750	3	2	4	yes	yes	no	no	yes	1	yes	unfurnished
11	9800000	13200	3	1	2	yes	no	yes	no	yes	2	yes	furnished
12	9681000	6000	4	3	2	yes	yes	yes	yes	no	2	no	semi-furnished
13	9310000	6550	4	2	2	yes	no	no	no	yes	1	yes	semi-furnished
14	9240000	3500	4	2	2	yes	no	no	yes	no	2	no	furnished
15	9240000	7800	3	2	2	yes	no	no	no	no	0	yes	semi-furnished
16	9100000	6000	4	1	2	yes	no	yes	no	no	2	no	semi-furnished
17	9100000	6600	4	2	2	yes	yes	yes	no	yes	1	yes	unfurnished
18	8960000	8500	3	2	4	yes	no	no	no	yes	2	no	furnished
19	8890000	4600	3	2	2	yes	yes	no	no	yes	2	no	furnished

Showing 1 to 20 of 545 entries, 13 total columns

Console

III. Data Exploration, Cleaning, and Preparation

Now that the dataset has been imported into the working environment, it must be prepared and cleaned for processing using similar methods from the previous activity.

For data exploration, the **dim() function** was used to print the number of entries and attributes contained within the uncleaned dataset. According to the results, *housing* has a total of **545 entries and thirteen attributes** or variables.

```
> dim(housing)
[1] 545  13
```

To view a summary of each variable from the dataset, the code snippet below was run. The figure below is its corresponding output.

```
> summary(housing)
```

```
> summary(housing)
     price               area          bedrooms       bathrooms        stories     
 Min.   : 1750000   Min.   : 1650   Min.   :1.000   Min.   :1.000   Min.   :1.000  
 1st Qu.: 3430000   1st Qu.: 3600   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000  
 Median : 4340000   Median : 4600   Median :3.000   Median :1.000   Median :2.000  
 Mean   : 4766729   Mean   : 5151   Mean   :2.965   Mean   :1.286   Mean   :1.806  
 3rd Qu.: 5740000   3rd Qu.: 6360   3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:2.000  
 Max.   :13300000   Max.   :16200   Max.   :6.000   Max.   :4.000   Max.   :4.000  
   mainroad          guestroom           basement         hotwaterheating    airconditioning   
 Length:545         Length:545         Length:545         Length:545         Length:545        
 Class :character   Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character  


    parking         prefarea         furnishingstatus  
 Min.   :0.0000   Length:545         Length:545        
 1st Qu.:0.0000   Class :character   Class :character  
 Median :0.0000   Mode  :character   Mode  :character  
 Mean   :0.6936                                        
 3rd Qu.:1.0000                                        
 Max.   :3.0000                                        
```

For data cleaning and preparation, we converted the data into numeric values using the **as.numeric() function** with the **factor() function** as its parameter.

```
> # Convert data to numeric
> housing$mainroad <- as.numeric(factor(housing$mainroad))
> housing$guestroom <- as.numeric(factor(housing$guestroom))
> housing$basement <- as.numeric(factor(housing$basement))
> housing$hotwaterheating <- as.numeric(factor(housing$hotwaterheating))
> housing$airconditioning <- as.numeric(factor(housing$airconditioning))
> housing$furnishingstatus <- as.numeric(factor(housing$furnishingstatus))
```

Then, to view the first few lines of code, we ran the **head() function**. The figure below shows the appropriate output.

```
> head(housing)
```

```
     price area bedrooms bathrooms stories mainroad guestroom basement hotwaterheating
1 13300000 7420        4         2       3        2         1        1               1
2 12250000 8960        4         4       4        2         1        1               1
3 12250000 9960        3         2       2        2         1        2               1
4 12215000 7500        4         2       2        2         1        2               1
5 11410000 7420        4         1       2        2         2        2               1
6 10850000 7500        3         3       1        2         1        2               1
  airconditioning parking prefarea furnishingstatus
1               2       2      yes                1
2               2       3       no                1
3               1       2      yes                2
4               2       3      yes                1
5               2       2       no                1
6               2       2      yes                2
```

Next, to clean any irrelevant variables, the **select() function** was implemented to store its output in the *housing_clean* data frame using the code snippet below.

```
> #clean unnecessary variables
> housing_clean <- housing %>%
+   select(-c(prefarea)) #Removing Unnecessary Variables
```

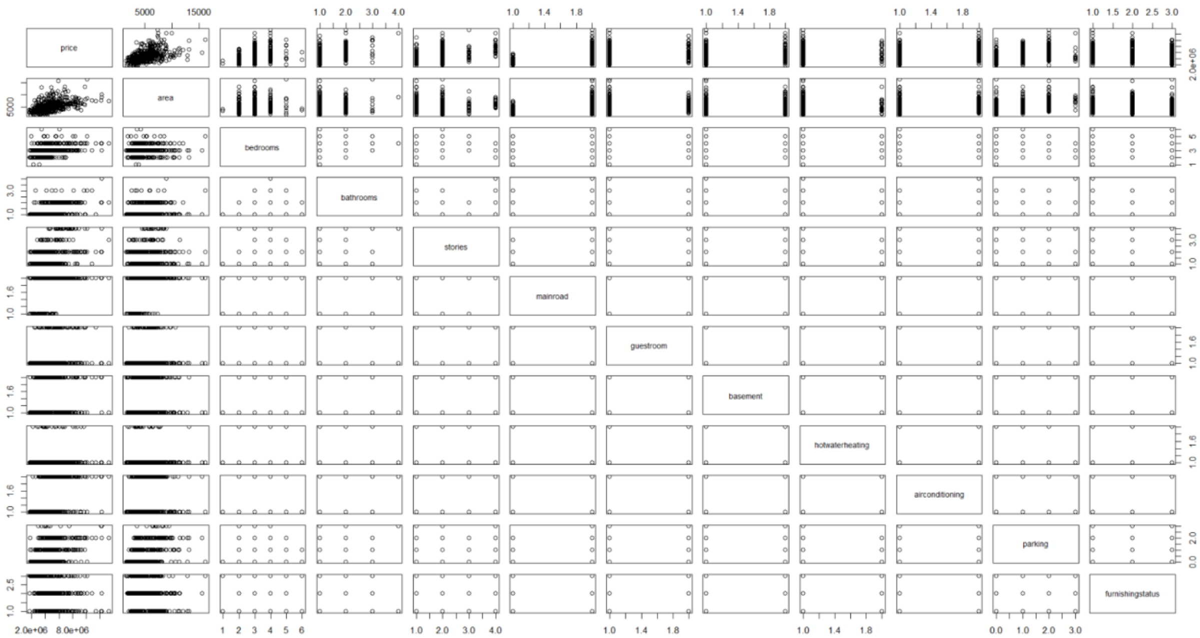
After the data has been cleaned and prepared, its variables were further explored to determine which variables could serve as the dependent variable and independent variables. This will be further discussed in the following section.

IV. Identifying the Dependent and Independent Variables

Before any variables were identified as dependent or independent, the correlation within the variables from the cleaned dataset were first deciphered using the **pairs() function**.

```
> pairs(housing_clean)
```

The figure below is a clearer view of the correlations within the cleaned dataset. Since the attribute *price* has the most positive correlation among all the other variables, we selected such as the dependent variable. On the other hand, all the other attributes were considered as independent variables to observe if there will be other relevant correlations with *price*.



Correlations within cleaning *Housing* dataset

V. Splitting the Data into Training and Testing Sets

After identifying the dependent and independent variables, the next step is to split the dataset into 70% for training and 30% for testing. Looking at the snippet below, the **training dataset consists of 382 instances** while the **testing dataset has 163 instances**.

```
> ## Splitting data into Train and Test into 70/30 ##
> housing_clean['row_id'] = rownames(housing_clean)
> set.seed(123)
> train_data <- housing_clean %>%
+   sample_frac(0.7)
> test_data <- housing_clean %>%
+   anti_join(train_data, by='row_id')
> # Drop row_id from both dataframes
> train_data[,'row_id'] <- NULL
> test_data[,'row_id'] <- NULL
> dim(train_data)
[1] 382  12
> dim(test_data)
[1] 163  12
```

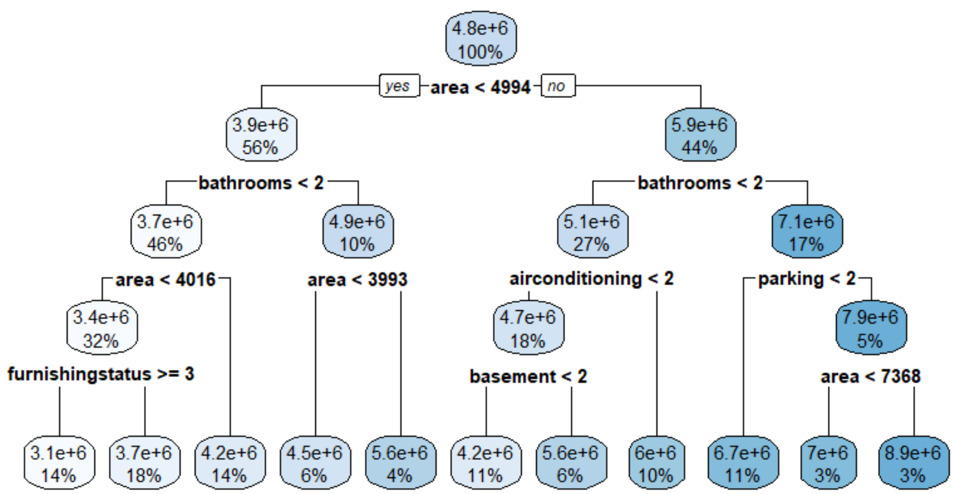
VI. Making a Decision Tree with Default Parameters

Now that the dataset has been split into training and testing sets, the decision tree can now be created. Using the *rpart()* function, a decision tree of the default parameters can be created. The *rpart*'s default values consist of the following:

- maxdepth = 30
- minsplit = 20
- minbucket = 7
- cp = 0.01

Using the code below, a decision tree was created using the training dataset as its data and the price as the dependent variable.

```
# Decision Tree with default parameters
d.tree = rpart(train_data$price~., data=train_data)
```



Notice that the tree is relatively shallow with five levels yet with many leaf nodes at the bottom. Next, a confusion matrix was created to evaluate the model. Using the *table()* function, a confusion matrix was created using the predict_price as the column and price as the row.

```
## Predicting with test set ##
predict_price <- predict(d.tree, test_data)
table_price <- table(test_data$price, predict_price)
#Printing confusion matrix
print(table_price)
```

predict_price

	3053884.61538462	3730150	4170534.88372093	4175896.22641509	4452000	5551875
1855000	1	0	0	0	0	0
1890000	2	0	0	0	0	0
1960000	1	0	0	0	0	0
2100000	2	0	0	0	0	0
2233000	0	0	0	0	0	0
2380000	1	0	0	0	1	0
2408000	1	0	0	0	0	0
2450000	1	0	1	1	0	0
2485000	0	1	0	0	0	0
2590000	1	0	0	1	0	0
2604000	1	0	0	0	0	0
2653000	1	0	0	0	0	0
2660000	1	1	0	0	1	0
2835000	0	1	0	0	0	0
2852500	0	0	1	0	0	0
2870000	1	0	0	0	0	0
2940000	1	0	0	1	0	1
2975000	0	0	0	1	0	0
3010000	1	2	0	0	0	0
3115000	1	0	0	0	0	0
3143000	0	0	0	1	0	0
3150000	2	0	2	1	0	0
3220000	0	0	0	1	0	0
3234000	1	0	0	0	0	0
3290000	0	1	0	0	0	1
3360000	0	0	0	2	0	0
3465000	1	0	0	0	0	0
3500000	2	1	0	0	0	0
3535000	1	0	0	0	0	0
3570000	0	0	0	1	0	0
3605000	0	0	0	1	0	0
3640000	0	1	0	0	0	0
3675000	0	1	0	0	0	0
3710000	0	0	0	2	0	0
3773000	0	0	1	0	0	0
3780000	0	1	0	0	0	0
3850000	0	1	0	0	0	0
3920000	0	2	0	0	0	1
3990000	0	0	0	1	0	0

4025000	0	0	1	0	2	0
4060000	0	1	0	0	0	0
4098500	0	1	0	0	0	0
4130000	0	0	0	0	0	1
4200000	1	0	0	1	1	0
4235000	0	1	0	0	0	0
4270000	1	0	0	1	0	0
4340000	0	1	1	0	0	0
4403000	0	1	0	1	0	0
4410000	0	1	0	0	0	0
4473000	0	1	0	0	0	0
4480000	0	0	0	0	0	0
4550000	0	2	1	0	0	0
4613000	0	0	0	0	0	0
4655000	0	1	0	0	0	0
4690000	0	0	1	0	0	0
4795000	0	0	0	0	0	1
4830000	0	0	0	2	0	0
4893000	0	0	0	0	1	0
4900000	0	0	0	2	0	0
5040000	0	1	0	0	0	1
5110000	0	0	1	0	0	0
5145000	0	1	0	0	0	0
5215000	0	0	0	0	1	1
5250000	0	0	0	0	0	0
5460000	0	0	0	0	0	0
5523000	0	0	0	0	0	1
5600000	0	0	1	0	0	0
5652500	0	0	0	0	0	0
5740000	0	0	0	1	0	0
5810000	0	0	1	0	0	0
5866000	0	0	0	1	0	0
5943000	0	0	0	0	0	0
5950000	0	0	0	0	0	0
6090000	0	0	0	0	0	0
6107500	0	1	0	0	0	0
6195000	0	0	0	0	0	0
6230000	0	0	0	0	0	0
6293000	0	0	0	0	0	0
6300000	0	0	0	0	0	0
6510000	0	0	0	0	0	1
6650000	0	0	0	0	0	0
6790000	0	0	0	0	0	0
6895000	0	0	0	0	0	0
7000000	0	0	0	0	0	0
7070000	0	0	0	0	0	0
7210000	0	0	0	0	0	0
7245000	0	0	0	0	0	0
7350000	0	0	0	0	0	0
7455000	0	0	0	0	0	0
7525000	0	0	0	0	0	0

	predict_price				
	5624937.5	5968824.32432432	6700790.69767442	7004721.81818182	8932000
1855000	0	0	0	0	0
1890000	0	0	0	0	0
1960000	0	0	0	0	0
2100000	0	0	0	0	0
2233000	0	1	0	0	0
2380000	0	0	0	0	0
2408000	0	0	0	0	0
2450000	0	0	0	0	0
2485000	0	0	0	0	0
2590000	0	0	0	0	0
2604000	0	0	0	0	0
2653000	0	0	0	0	0
2660000	0	0	0	0	0
2835000	0	0	0	0	0
2852500	0	0	0	0	0
2870000	0	0	0	0	0
2940000	0	0	0	0	0
2975000	0	0	0	0	0
3010000	0	0	0	0	0
3115000	0	0	0	0	0
3143000	0	0	0	0	0
3150000	0	0	0	0	0
3220000	0	0	0	0	0
3234000	0	0	0	0	0
3290000	0	0	0	0	0
3360000	0	0	0	0	0
3465000	0	0	0	0	0
3500000	0	0	0	0	0
3535000	0	0	0	0	0
3570000	1	0	0	0	0
3605000	0	0	0	0	0
3640000	1	0	0	0	0
3675000	0	0	0	0	0
3710000	0	0	0	0	0
3773000	0	0	0	0	0
3780000	0	0	0	0	0
3850000	0	1	1	0	0
3920000	0	1	0	0	0
3990000	0	1	0	0	0

4025000	0	0	0	0	0
4060000	0	0	0	0	0
4098500	0	0	0	0	0
4130000	0	0	0	0	0
4200000	0	0	0	0	0
4235000	0	0	0	0	0
4270000	0	0	0	0	0
4340000	1	0	0	0	0
4403000	0	0	0	0	0
4410000	0	0	0	0	0
4473000	0	0	0	0	0
4480000	0	1	0	0	0
4550000	0	0	0	0	0
4613000	1	0	0	0	0
4655000	0	0	0	0	0
4690000	0	2	0	0	0
4795000	0	0	0	0	0
4830000	0	0	0	0	0
4893000	0	0	0	0	0
4900000	0	2	0	0	0
5040000	0	0	0	0	0
5110000	0	1	0	0	0
5145000	0	0	0	0	0
5215000	0	0	0	0	0
5250000	0	1	1	0	0
5460000	0	1	0	0	0
5523000	0	0	0	0	0
5600000	1	0	1	0	0
5652500	0	0	1	0	0
5740000	0	1	0	0	0
5810000	0	1	0	0	0
5866000	0	0	0	0	0
5943000	0	1	0	0	0
5950000	0	1	2	0	0
6090000	0	1	0	0	0
6107500	0	0	0	0	0
6195000	0	0	1	0	0
6230000	0	0	1	0	0
6293000	0	0	1	0	0
6300000	0	0	1	0	0
6510000	0	0	0	0	0
6510000	0	0	0	0	0
6650000	0	1	1	0	0
6790000	1	0	0	0	0
6895000	0	0	0	0	1
7000000	0	1	0	0	0
7070000	0	1	1	0	0
7210000	0	0	1	0	0
7245000	0	0	1	0	0
7350000	0	1	0	0	0
7455000	1	0	0	0	0
7525000	0	0	1	0	0

Unfortunately, the confusion matrix is too long to be able to decipher and understand it. However, an accuracy test was made to reveal that the decision tree model has a ~0.6135% accuracy. The decision tree is very inaccurate, but this can still be improved by adjusting the hyperparameters.

```
> accuracy_Test <- sum(diag(table_price)) / sum(table_price)
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 0.00613496932515337"
```

VII. Adjusting the Hyperparameters

To adjust the hyperparameters of the decision tree, the *mlr* library is a great artificial intelligence package for R that allows data scientists to tune and train various models. One of its benefits includes its ability to enable the user to see how each hyperparameter affects the model's performance. Before using *mlr*, one must define a classification task with the training dataset and target, which in this case is the *price*.

```
> # hyperparameter tune
> d.tree.params <- makeClassifTask(
+   data=train_data,
+   target="price"
+ )
```

Then, a grid of parameters must be created to iterate on. We need the *makeParamSet()* function and use a *makeDiscreteParam()* and *makeNumericParam()* functions for the parameters. While *makeDiscreteParam()* provides discrete parameters like integers (such as maxdepth or minsplit), *makeNumericParam()* creates numerical parameters (such as cp with decimal places). The hyperparameters being tweaked in the code snippet below are *maxdepth*, *cp*, *minsplit*, *minbucket*, and *xval*.

```
> param_grid_multi <- makeParamSet(
+   makeDiscreteParam("maxdepth", values=4:5),
+   makeNumericParam("cp", lower = 0.009, upper = 0.01),
+   makeDiscreteParam("minsplit", values=5:10),
+   makeDiscreteParam("minbucket", values=0:5),
+   makeDiscreteParam("xval", values=0:5)
+ )
```

Next the control grid experiment should be initialized, and the cross-validation method and measure should be chosen to evaluate the results. Here, the three-fold cross validation was used to improve the decision tree results.

```
> # Define Grid
> control_grid = makeTuneControlGrid()
>
> # Define Cross Validation
> resample = makeResampleDesc("CV", iters = 3L)
>
> # Define Measure
> measure = acc
```

After everything is set, the *tuneParams()* function will be used to start the hyperparameter tuning. As the code below was run, the hyperparameter search will begin executing and outputting the execution's feedback.

```
dt_tuneparam_multi <- tuneParams(learner='classif.rpart',
                                 task=d.tree.params,
                                 resampling = resample,
                                 measures = measure,
                                 par.set=param_grid_multi,
                                 control=control_grid,
                                 show.info = TRUE)
```

After 4,320 test trees were fitted, the results reveal that the best parameters for the decision tree should be:

- maxdepth of 4,
- cp of 0.00933,
- minsplit of 10,
- minbucket of 3,
- and an xval of 0

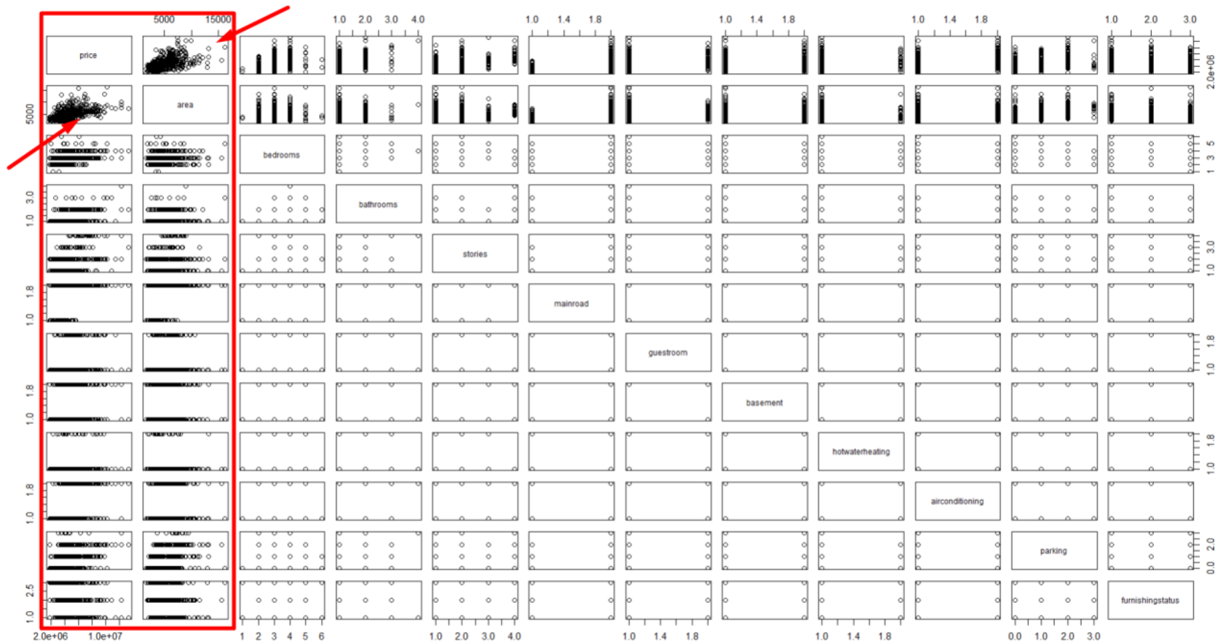
```
[Tune-y] 4318: acc.test.mean=0.0261852; time: 0.0 min
[Tune-x] 4319: maxdepth=4; cp=0.01; minsplit=10; minbucket=5; xval=5
[Tune-y] 4319: acc.test.mean=0.0261852; time: 0.0 min
[Tune-x] 4320: maxdepth=5; cp=0.01; minsplit=10; minbucket=5; xval=5
[Tune-y] 4320: acc.test.mean=0.0261852; time: 0.0 min
[Tune] Result: maxdepth=4; cp=0.00933; minsplit=10; minbucket=3; xval=0 : acc.test.mean=0.028
8099
```

VIII. Prediction Results and Analysis

According to the test of the *tuneParams()* function, the estimated accuracy would be 2.88%. Therefore, these optimized parameters could yield a better decision tree than the default one. To test that theory, the best parameters must be extracted from the multi search.

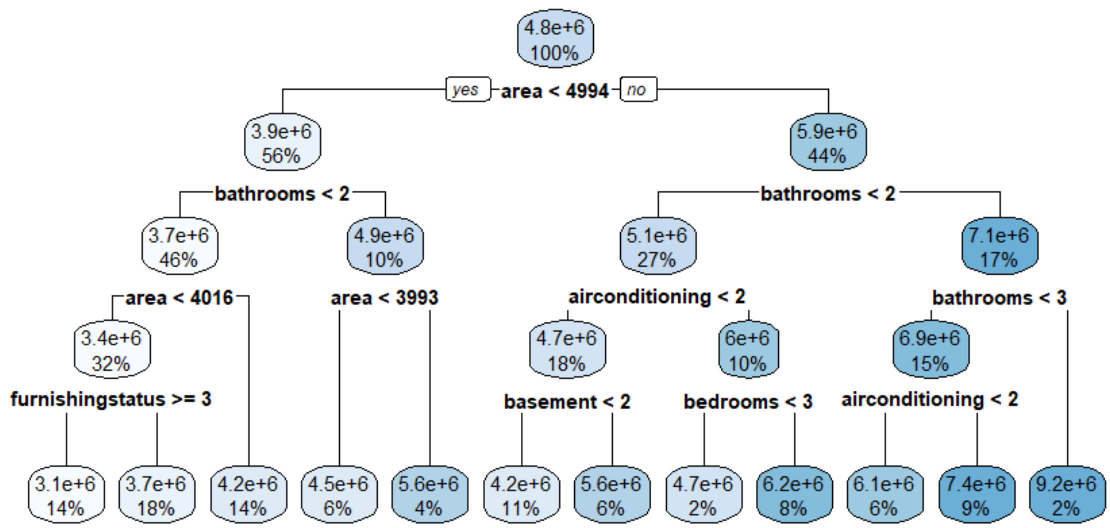
```
> ## Extracting best Parameters from Multi Search ##
> best_parameters_multi = setHyperPars(
+   makeLearner("classif.rpart", predict.type = "prob"),
+   par.vals = dt_tuneparam_multi$x
+ )

> best_model_multi = train(best_parameters_multi, d.tree.params)
>
> ## Predicting the best Model ##
> results <- predict(best_model_multi, task = d.tree.params)$data
> accuracy(results$truth, results$response)
[1] 0.03403141
```

The new decision tree model's accuracy has reached **3.40%**. By tweaking these parameters, the decision tree improved by three percentage points of the baseline accuracy. However, this does not mean the model is still accurate. The new confusion matrix was just as incomprehensible as the [previous one](). Nonetheless, this proves that tweaking a model's hyperparameters will always yield more outstanding results. Furthermore, the model's accuracy also portrays the correlation within the dataset variables clearly as only the variables price and area have a positive correlation, as evidenced in the figure below.



Finally, the decision tree will be recreated using the best hyperparameters as stated in the code below.

```
> # decision tree with best hyperparameters
> best.d.tree = rpart(train_data$price~.,
+                     data=train_data,
+                     control = c(maxdepth = 4, cp=0.00933, minsplit = 10, minbucket = 3, x
val = 0))
>
> rpart.plot(best.d.tree)
```


4.8e+6
100%
yes
area < 4994
no
3.9e+6
56%
5.9e+6
44%
bathrooms < 2
bathrooms < 2
3.7e+6
46%
4.9e+6
10%
5.1e+6
27%
7.1e+6
17%
area < 4016
area < 3993
airconditioning < 2
bathrooms < 3
3.4e+6
32%
4.7e+6
18%
6e+6
10%
6.9e+6
15%
furnishingstatus >= 3
basement < 2
bedrooms < 3
airconditioning < 2
3.1e+6
14%
3.7e+6
18%
4.2e+6
14%
4.5e+6
6%
5.6e+6
4%
4.2e+6
11%
5.6e+6
6%
4.7e+6
2%
6.2e+6
8%
6.1e+6
6%
7.4e+6
9%
9.2e+6
2%

References

Bernardo, Ivo. “Decision Tree Hyperparameter Tuning in R Using Mlr - Towards Data Science.” *Medium*, 9 June 2022, towardsdatascience.com/decision-tree-hyperparameter-tuning-in-r-using-mlr-3248bfd2d88c.

Housing Simple Regression. (2020, February 27). Kaggle. https://www.kaggle.com/datasets/thorgodofthunder/housing-simple-regression

Soetewey, Antoine. “Correlation Coefficient and Correlation Test in R.” *Stats and R,* 28 May 2020, statsandr.com/blog/correlation-coefficient-and-correlation-test-in-r.