NYC Sales Project

Anshuman Dash

1/6/2021

1. Introduction:

Machine Learning has many different applications. One application is in the use of prediction models, specifically predicting property sales. For my project I'll be using the "NYC Property Sales" data set. This data set contains one year of property sales in NYC accross 2016-2017. This data set contains over 84000 sales with 22 variables. These variables help describe and classify the sales. For example we can get the sale price, date of sale, the borough, zipcode, and a lot more. But in the scope of this project we want to try to predict the sale price using only a few of the variables.

For this project we will be using two primary models, the first a simple linear regression model and the second is a decision tree model. The linear regression model's goals is to create an overall prediction model, while using the decision tree to find good predictor variables and see how our data split based on probabilites from the decision trees.

Overall for this project we will download and clean up the data for analysis. Then we'll begin with developing our linear regression. We'll try to identify variables best suited to help predict the sales price, then we'll incorporate them into our model and check the RMSE. Then we'll develop a decision tree using the "ANOVA" method.

2. Methods/Analysis:

Our methodology can be broken down into a few key stages. The first stage is downloading the data and cleaning it up for later use. We then shape our data by filtering out unnecessary data that we don't need. We also drop NA's and create additioinal columns to help analyze some of our categorical data. The next stage is to create our linear regression models using the "lm()" function. We use a correlation matrix to identify variables that will aid in predicting sale prices. We'll slowly test a few models and then use the "train()" function to train a linear model and use cross-validation to see which one was the best.

Our final step is to create a decision tree using the "rpart()" function, and specifically the "ANOVA" method. We then view the tree using the "rpart.plot()" function.

We will begin by downloading and processing the data for analysis later.

2.2 Data Ingestion/Pre-Processing:

We'll begin by downloading any libraries we may need, then reading the csv file containing our data. Then we'll clean it up by reformatting it for easier refernce, readability, and analysis.

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 84548 obs. of  22 variables:
##  $ X1                            : num  4 5 6 7 8 9 10 11 12 13 ...
##  $ BOROUGH                       : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ NEIGHBORHOOD                  : chr  "ALPHABET CITY" "ALPHABET CITY" "ALPHABET CITY" "ALPHABET CI
```

```
##  $ BUILDING CLASS CATEGORY       : chr  "07 RENTALS - WALKUP APARTMENTS" "07 RENTALS - WALKUP APARTM
##  $ TAX CLASS AT PRESENT          : chr  "2A" "2" "2" "2B" ...
##  $ BLOCK                         : num  392 399 399 402 404 405 406 407 379 387 ...
##  $ LOT                           : num  6 26 39 21 55 16 32 18 34 153 ...
##  $ EASE-MENT                     : logi  NA NA NA NA NA NA ...
##  $ BUILDING CLASS AT PRESENT     : chr  "C2" "C7" "C7" "C4" ...
##  $ ADDRESS                       : chr  "153 AVENUE B" "234 EAST 4TH   STREET" "197 EAST 3RD   STREE
##  $ APARTMENT NUMBER              : chr  NA NA NA NA ...
##  $ ZIP CODE                      : num  10009 10009 10009 10009 10009 ...
##  $ RESIDENTIAL UNITS             : num  5 28 16 10 6 20 8 44 15 24 ...
##  $ COMMERCIAL UNITS              : num  0 3 1 0 0 0 0 2 0 0 ...
##  $ TOTAL UNITS                   : num  5 31 17 10 6 20 8 46 15 24 ...
##  $ LAND SQUARE FEET              : chr  "1633" "4616" "2212" "2272" ...
##  $ GROSS SQUARE FEET             : chr  "6440" "18690" "7803" "6794" ...
##  $ YEAR BUILT                    : num  1900 1900 1900 1913 1900 ...
##  $ TAX CLASS AT TIME OF SALE     : num  2 2 2 2 2 2 2 2 2 2 ...
##  $ BUILDING CLASS AT TIME OF SALE: chr  "C2" "C7" "C7" "C4" ...
##  $ SALE PRICE                    : chr  "6625000" "-" "-" "3936272" ...
##  $ SALE DATE                     : POSIXct, format: "2017-07-19" "2016-12-14" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   X1 = col_double(),
##   ..   BOROUGH = col_double(),
##   ..   NEIGHBORHOOD = col_character(),
##   ..   `BUILDING CLASS CATEGORY` = col_character(),
##   ..   `TAX CLASS AT PRESENT` = col_character(),
##   ..   BLOCK = col_double(),
##   ..   LOT = col_double(),
##   ..   `EASE-MENT` = col_logical(),
##   ..   `BUILDING CLASS AT PRESENT` = col_character(),
##   ..   ADDRESS = col_character(),
##   ..   `APARTMENT NUMBER` = col_character(),
##   ..   `ZIP CODE` = col_double(),
##   ..   `RESIDENTIAL UNITS` = col_double(),
##   ..   `COMMERCIAL UNITS` = col_double(),
##   ..   `TOTAL UNITS` = col_double(),
##   ..   `LAND SQUARE FEET` = col_character(),
##   ..   `GROSS SQUARE FEET` = col_character(),
##   ..   `YEAR BUILT` = col_double(),
##   ..   `TAX CLASS AT TIME OF SALE` = col_double(),
##   ..   `BUILDING CLASS AT TIME OF SALE` = col_character(),
##   ..   `SALE PRICE` = col_character(),
##   ..   `SALE DATE` = col_datetime(format = "")
##   .. )
```

```
## # A tibble: 6 x 22
##      X1 BOROUGH NEIGHBORHOOD `BUILDING CLASS~ `TAX CLASS AT P~ BLOCK   LOT
##   <dbl>   <dbl> <chr>        <chr>            <chr>            <dbl> <dbl>
## 1     4       1 ALPHABET CI~ 07 RENTALS - WA~ 2A                 392     6
## 2     5       1 ALPHABET CI~ 07 RENTALS - WA~ 2                  399    26
## 3     6       1 ALPHABET CI~ 07 RENTALS - WA~ 2                  399    39
## 4     7       1 ALPHABET CI~ 07 RENTALS - WA~ 2B                 402    21
## 5     8       1 ALPHABET CI~ 07 RENTALS - WA~ 2A                 404    55
## 6     9       1 ALPHABET CI~ 07 RENTALS - WA~ 2                  405    16
```

```
## # ... with 15 more variables: 'EASE-MENT' <lgl>, 'BUILDING CLASS AT
## #   PRESENT' <chr>, ADDRESS <chr>, 'APARTMENT NUMBER' <chr>, 'ZIP CODE' <dbl>,
## #   'RESIDENTIAL UNITS' <dbl>, 'COMMERCIAL UNITS' <dbl>, 'TOTAL UNITS' <dbl>,
## #   'LAND SQUARE FEET' <chr>, 'GROSS SQUARE FEET' <chr>, 'YEAR BUILT' <dbl>,
## #   'TAX CLASS AT TIME OF SALE' <dbl>, 'BUILDING CLASS AT TIME OF SALE' <chr>,
## #   'SALE PRICE' <chr>, 'SALE DATE' <dttm>
```

I stored the csv data in nyc_data. When we take a look at nyc_data we notice the column names are all uppercase and have spaces in them, this is tough to reference and analyze. So I'll go ahead and make it all lowercase and replace the spaces with underscores.

```
### Data Pre-Processing
## Organizing and Formatting Data
# Make data easier to process and reference by making everything lowercase and replacing spaces with "_
colnames(nyc_data) %<>% str_replace_all("\\s", "_") %>% tolower()
```

Now that the data is easier to read, we'll go ahead make the sale_price, land_square_feet, and gross_square_feet columns numerical so we can model them. Then we'll drop any NA's that were made by coercion from as.numeric().

Now it's time to take a look at our interested columns sale_price, gross_square_feet, and land_square_feet. We'll create some frequency tables for each and see if their is anything going on, or if can proceed to modelling.

```
## # A tibble: 5,090 x 2
##    gross_square_feet  Freq
##                <dbl> <int>
##  1                 0 11417
##  2                60     2
##  3                80     1
##  4               100     2
##  5               120     1
##  6               150     2
##  7               200     6
##  8               205     1
##  9               210     1
## 10               240     2
## # ... with 5,080 more rows
```

```
## # A tibble: 5,173 x 2
##    land_square_feet  Freq
##               <dbl> <int>
##  1                0 10326
##  2                2     1
##  3               60     1
##  4               73     1
##  5               75     1
##  6               98     1
##  7              100     1
##  8              120     1
##  9              130     1
## 10              150     1
## # ... with 5,163 more rows
```

```
## # A tibble: 6,231 x 2
##    sale_price  Freq
##         <dbl> <int>
##  1          0 10228
##  2          1   105
##  3          3     2
##  4          5     1
##  5          8     1
##  6         10   651
##  7         19     1
##  8         20     4
##  9        100    82
## 10        200     1
## # ... with 6,221 more rows
```

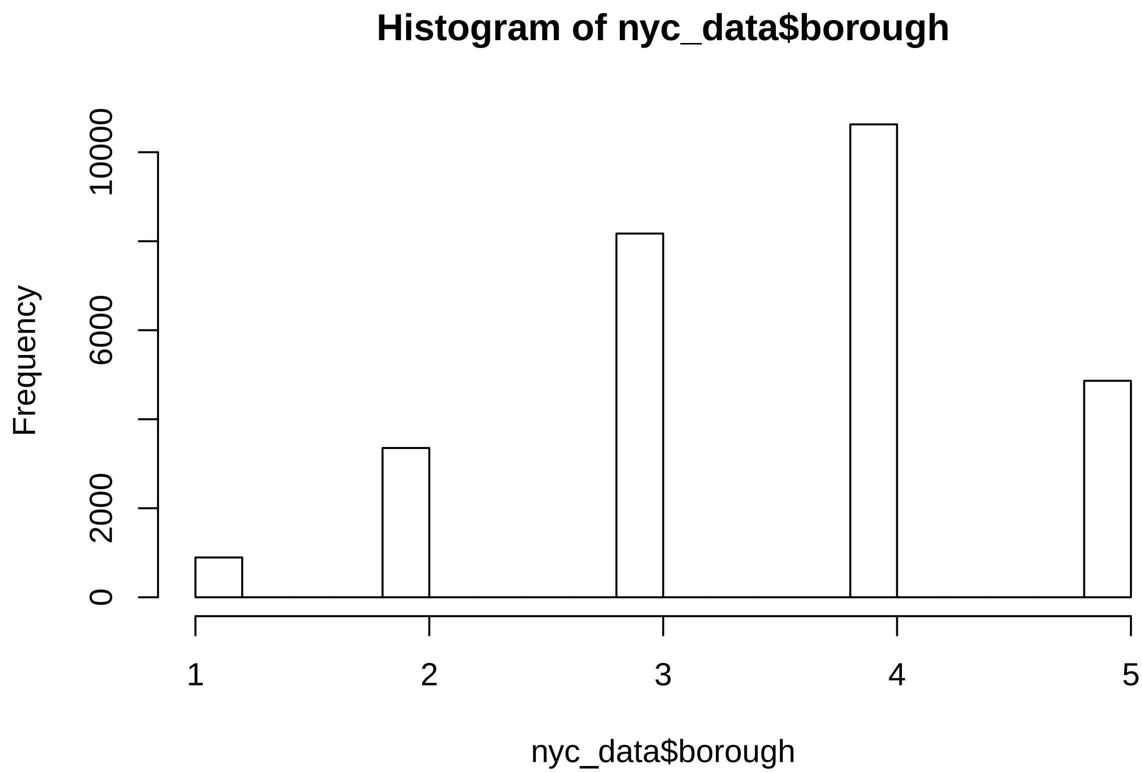
What we notice from our frequency tables is that we have over 10,000 sales of $0 and quite a few sales that are less than $100,000. This is NYC where property prices are expensive, we will use $100,000 as our cutoff. Additionally for both land_square_feet and gross_square_feet we don't want anything less than 100 square feet since we'll deem the smaller spaces as uninhabitable for our study. So we'll go ahead and filter out sales under $100,000, and square feet under 100.

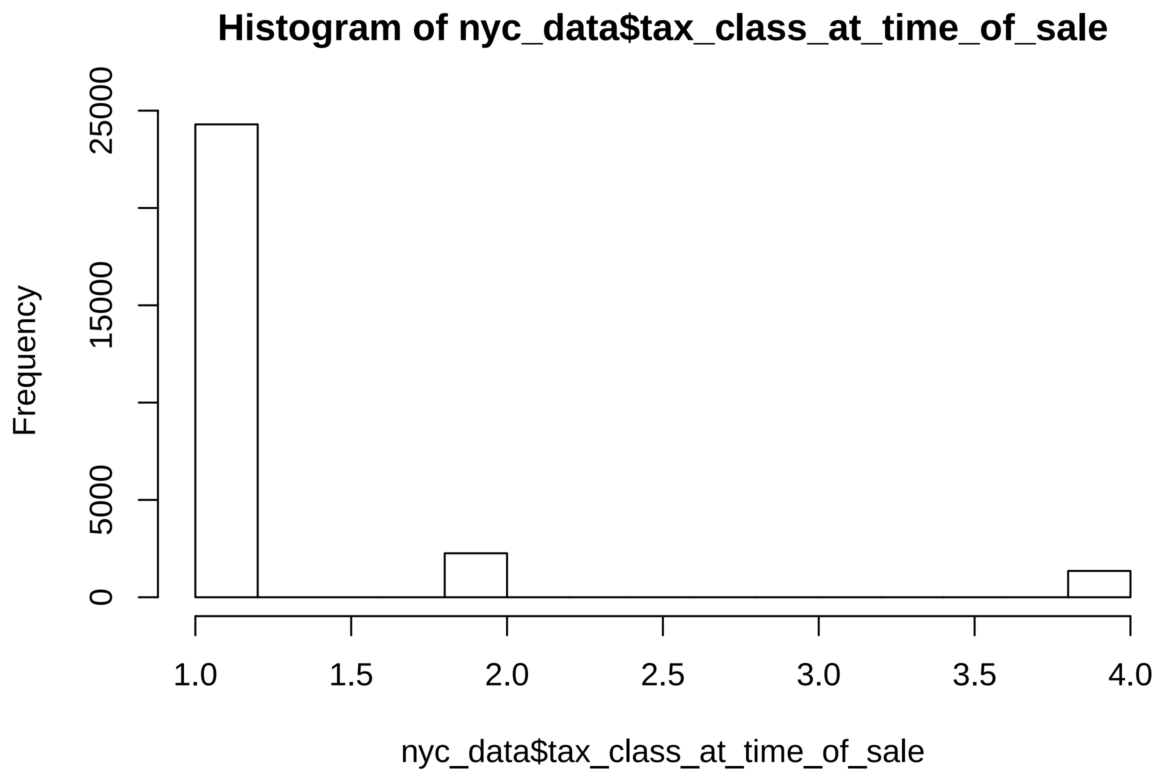
```r
# Filter Unwanted Data

nyc_data <- nyc_data %>% filter(land_square_feet>100) %>% filter(gross_square_feet>100) %>% filter(sale_
```

If we take a look at the borough column, we notice that they have an integer value in the range of 1-5. These numbers have a real world meaning, 1 = Manhatten, 2 = Bronx, 3 = Brooklyn, 4 = Queens, and 5 = Staten Island. We don't just want borogh identity to be stuck in this column, we want to create individual columns that will state whether the property is in the borough. This will be important later to identify correlation between the boroughs and sale_price. Additionally this will help identify any collinearity problems. We'll go ahead and add those columns now.

```
## # A tibble: 6 x 27
##      x1 borough neighborhood building_class_~ tax_class_at_pr~ block   lot
##   <dbl>   <dbl> <chr>        <chr>            <chr>            <dbl> <dbl>
## 1     4       1 ALPHABET CI~ 07 RENTALS - WA~ 2A                 392     6
## 2     7       1 ALPHABET CI~ 07 RENTALS - WA~ 2B                 402    21
## 3     8       1 ALPHABET CI~ 07 RENTALS - WA~ 2A                 404    55
## 4    10       1 ALPHABET CI~ 07 RENTALS - WA~ 2B                 406    32
## 5    13       1 ALPHABET CI~ 08 RENTALS - EL~ 2                  387   153
## 6    15       1 ALPHABET CI~ 08 RENTALS - EL~ 2B                 400    21
## # ... with 20 more variables: `ease-ment` <lgl>,
## #   building_class_at_present <chr>, address <chr>, apartment_number <chr>,
## #   zip_code <dbl>, residential_units <dbl>, commercial_units <dbl>,
## #   total_units <dbl>, land_square_feet <dbl>, gross_square_feet <dbl>,
## #   year_built <dbl>, tax_class_at_time_of_sale <dbl>,
## #   building_class_at_time_of_sale <chr>, sale_price <dbl>, sale_date <dttm>,
## #   in_manhattan <dbl>, in_bronx <dbl>, in_brooklyn <dbl>, in_queens <dbl>,
## #   in_staten_island <dbl>
```


Histogram of nyc_data$borough
Frequency
0
2000
6000
10000
1
2
3
4
5
nyc_data$borough




```
## # A tibble: 27,910 x 30
##       x1 borough neighborhood building_class_~ tax_class_at_pr~ block   lot
##    <dbl>   <dbl> <chr>        <chr>            <chr>            <dbl> <dbl>
##  1     4       1 ALPHABET CI~ 07 RENTALS - WA~ 2A                 392     6
##  2     7       1 ALPHABET CI~ 07 RENTALS - WA~ 2B                 402    21
##  3     8       1 ALPHABET CI~ 07 RENTALS - WA~ 2A                 404    55
##  4    10       1 ALPHABET CI~ 07 RENTALS - WA~ 2B                 406    32
##  5    13       1 ALPHABET CI~ 08 RENTALS - EL~ 2                  387   153
##  6    15       1 ALPHABET CI~ 08 RENTALS - EL~ 2B                 400    21
##  7    26       1 ALPHABET CI~ 09 COOPS - WALK~ 2                  376    14
##  8   176       1 ALPHABET CI~ 14 RENTALS - 4-~ 2A                 391    19
##  9   177       1 ALPHABET CI~ 14 RENTALS - 4-~ 2A                 393     4
## 10   178       1 ALPHABET CI~ 14 RENTALS - 4-~ 2A                 394     5
## # ... with 27,900 more rows, and 23 more variables: `ease-ment` <lgl>,
## #   building_class_at_present <chr>, address <chr>, apartment_number <chr>,
## #   zip_code <dbl>, residential_units <dbl>, commercial_units <dbl>,
## #   total_units <dbl>, land_square_feet <dbl>, gross_square_feet <dbl>,
## #   year_built <dbl>, tax_class_at_time_of_sale <dbl>,
## #   building_class_at_time_of_sale <chr>, sale_price <dbl>, sale_date <dttm>,
## #   in_manhattan <dbl>, in_bronx <dbl>, in_brooklyn <dbl>, in_queens <dbl>,
## #   in_staten_island <dbl>, tax_class_one <dbl>, tax_class_two <dbl>,
## #   tax_class_four <dbl>
```

```
## # A tibble: 30 x 2
##    building_class_category           freq
```

```
##    <chr>                             <int>
##  1 01 ONE FAMILY DWELLINGS          12443
##  2 02 TWO FAMILY DWELLINGS           9582
##  3 03 THREE FAMILY DWELLINGS         2252
##  4 05 TAX CLASS 1 VACANT LAND          12
##  5 06 TAX CLASS 1 - OTHER               8
##  6 07 RENTALS - WALKUP APARTMENTS    1709
##  7 08 RENTALS - ELEVATOR APARTMENTS   198
##  8 09 COOPS - WALKUP APARTMENTS         9
##  9 10 COOPS - ELEVATOR APARTMENTS      25
## 10 11 SPECIAL CONDO BILLING LOTS        1
## # ... with 20 more rows
```

We now five additional columns with in_boroughname for each borough, which will be very important when looking at the correlation matrix. We also added 3 additioinal columns specifying what tax class each property had at the time of the sale. It's important that we have this info to help us Before we create our correlation matrix, we want to remove variables that we just don't want to focus on right now. For the scope of our project we don't want to include easements, zipcodes, sale dats, apartment numbers, etc. In a future expansion of this project I would like to see if sale_date has an effect on the sale price. But this is something for the future. So we will go ahead and use the subset() function to remove the columns we don't want. We just want to keep

If we take a look at the histogram of how often we see specific boroughs, we notice that most of the sales take part in Brooklyn and Queens. Another unique thing to note is that if we take look at the frequency table of the different type of building classes. We will discuss later on how we can expand our project for the future to involve looking at specific building classes and specific classes in boroughs. Since their are 30 building classes, and to keep the scope of our project mangeable, I have chosen to instead focus on the tax classes, like I mentioned earlier. Looking at the histogram we can see just how prevalent tax class one properties there were.

```
nyc_data <- subset(nyc_data, select = -c(zip_code, block, lot, `ease-ment`, address, apartment_number, s
```

2.3 Variables of Interest:

Now we will make our correlation matrix, and only take into account columns that have numerical values. Using the correlation matrix we can identify variables of interest to help predict sales prices.

```
## Identify possible correlation
cor_matrix <- nyc_data %>% select_if(is.numeric) %>% cor(use = 'pairwise.complete.obs')
cor_matrix
```

```
##                                    x1      borough residential_units
## x1                       1.0000000000  0.058941504      -0.017643341
## borough                  0.0589415043  1.000000000      -0.103255126
## residential_units       -0.0176433409 -0.103255126       1.000000000
## commercial_units        -0.0008102543 -0.010891922       0.011416455
## total_units             -0.0147294678 -0.089835864       0.815945842
## land_square_feet        -0.0085096579  0.009604747       0.458340756
## gross_square_feet       -0.0188963666 -0.090943898       0.721339867
## year_built              -0.1129516346  0.226804077      -0.002968256
## tax_class_at_time_of_sale -0.0176115200 -0.221398789       0.079213786
## sale_price              -0.0145225575 -0.104598316       0.141021984
## in_manhattan            -0.0575814557 -0.456387174       0.150050650
```

```
## in_bronx                  -0.3241203480 -0.562889003       0.023507282
## in_brooklyn                0.0957618583 -0.345716475      -0.009717975
## in_queens                  0.4320498581  0.351656988      -0.030950365
## in_staten_island          -0.3633453555  0.658918585      -0.038525951
## tax_class_one              0.0310207568  0.297987336      -0.185196210
## tax_class_two             -0.0352249443 -0.273033158       0.242737206
## tax_class_four            -0.0037454606 -0.118971419      -0.018845557
##                           commercial_units  total_units land_square_feet
## x1                           -0.0008102543 -0.014729468     -0.008509658
## borough                      -0.0108919218 -0.089835864      0.009604747
## residential_units             0.0114164555  0.815945842      0.458340756
## commercial_units              1.0000000000  0.587398204      0.054092843
## total_units                   0.5873982039  1.000000000      0.402236802
## land_square_feet              0.0540928429  0.402236802      1.000000000
## gross_square_feet             0.0654223276  0.621645170      0.666460864
## year_built                    0.0008704220 -0.001899371      0.017856585
## tax_class_at_time_of_sale     0.0786796741  0.109438536      0.101156776
## sale_price                    0.0476942296  0.141707255      0.040220973
## in_manhattan                  0.0252950968  0.136014667      0.006290490
## in_bronx                     -0.0034919453  0.017010933      0.001832698
## in_brooklyn                  -0.0064013424 -0.011575882     -0.014910563
## in_queens                     0.0050764661 -0.022092206     -0.004925520
## in_staten_island             -0.0075673831 -0.035552135      0.019699946
## tax_class_one                -0.0554686123 -0.181858470     -0.083388362
## tax_class_two                 0.0039438037  0.198758684      0.027356802
## tax_class_four                0.0816907472  0.031797297      0.095592287
##                           gross_square_feet   year_built
## x1                              -0.01889637 -0.112951635
## borough                         -0.09094390  0.226804077
## residential_units                0.72133987 -0.002968256
## commercial_units                 0.06542233  0.000870422
## total_units                      0.62164517 -0.001899371
## land_square_feet                 0.66646086  0.017856585
## gross_square_feet                1.00000000  0.012980996
## year_built                       0.01298100  1.000000000
## tax_class_at_time_of_sale        0.17880994 -0.034102525
## sale_price                       0.52721856  0.001081664
## in_manhattan                     0.14923472 -0.090286987
## in_bronx                         0.01240597 -0.007033345
## in_brooklyn                     -0.01098607 -0.161945587
## in_queens                       -0.03080792 -0.025200322
## in_staten_island                -0.02729271  0.274457700
## tax_class_one                   -0.18685619  0.077032691
## tax_class_two                    0.12118581 -0.099523814
## tax_class_four                   0.13813916  0.006005798
##                           tax_class_at_time_of_sale   sale_price in_manhattan
## x1                                      -0.01761152 -0.014522558  -0.05758146
## borough                                 -0.22139879 -0.104598316  -0.45638717
## residential_units                        0.07921379  0.141021984   0.15005065
## commercial_units                         0.07867967  0.047694230   0.02529510
## total_units                              0.10943854  0.141707255   0.13601467
## land_square_feet                         0.10115678  0.040220973   0.00629049
## gross_square_feet                        0.17880994  0.527218562   0.14923472
## year_built                              -0.03410253  0.001081664  -0.09028699
```

```
## tax_class_at_time_of_sale              1.00000000  0.160481541   0.26874075
## sale_price                             0.16048154  1.000000000   0.19960571
## in_manhattan                           0.26874075  0.199605708   1.00000000
## in_bronx                               0.02544426 -0.013837176  -0.06722967
## in_brooklyn                            0.06018622 -0.004534614  -0.11704984
## in_queens                             -0.09120770 -0.034717317  -0.14263333
## in_staten_island                      -0.10200392 -0.030913881  -0.08357132
## tax_class_one                         -0.86006892 -0.141899488  -0.35293499
## tax_class_two                          0.33637665  0.061133743   0.31522534
## tax_class_four                         0.91707451  0.144145674   0.15126621
##                               in_bronx  in_brooklyn    in_queens
## x1                        -0.324120348  0.095761858  0.432049858
## borough                   -0.562889003 -0.345716475  0.351656988
## residential_units          0.023507282 -0.009717975 -0.030950365
## commercial_units          -0.003491945 -0.006401342  0.005076466
## total_units                0.017010933 -0.011575882 -0.022092206
## land_square_feet           0.001832698 -0.014910563 -0.004925520
## gross_square_feet          0.012405974 -0.010986066 -0.030807922
## year_built                -0.007033345 -0.161945587 -0.025200322
## tax_class_at_time_of_sale  0.025444263  0.060186224 -0.091207696
## sale_price                -0.013837176 -0.004534614 -0.034717317
## in_manhattan              -0.067229666 -0.117049845 -0.142633327
## in_bronx                   1.000000000 -0.237801902 -0.289778055
## in_brooklyn               -0.237801902  1.000000000 -0.504516511
## in_queens                 -0.289778055 -0.504516511  1.000000000
## in_staten_island          -0.169785939 -0.295604888 -0.360214990
## tax_class_one             -0.033749008 -0.092440449  0.130467821
## tax_class_two              0.030460578  0.095328563 -0.126708243
## tax_class_four             0.014061365  0.023405896 -0.042987134
##                           in_staten_island tax_class_one tax_class_two
## x1                            -0.363345355    0.03102076  -0.035224944
## borough                        0.658918585    0.29798734  -0.273033158
## residential_units             -0.038525951   -0.18519621   0.242737206
## commercial_units              -0.007567383   -0.05546861   0.003943804
## total_units                   -0.035552135   -0.18185847   0.198758684
## land_square_feet               0.019699946   -0.08338836   0.027356802
## gross_square_feet             -0.027292712   -0.18685619   0.121185813
## year_built                     0.274457700    0.07703269  -0.099523814
## tax_class_at_time_of_sale     -0.102003924   -0.86006892   0.336376654
## sale_price                    -0.030913881   -0.14189949   0.061133743
## in_manhattan                  -0.083571316   -0.35293499   0.315225343
## in_bronx                      -0.169785939   -0.03374901   0.030460578
## in_brooklyn                   -0.295604888   -0.09244045   0.095328563
## in_queens                     -0.360214990    0.13046782  -0.126708243
## in_staten_island               1.000000000    0.13664117  -0.124595187
## tax_class_one                  0.136641173    1.00000000  -0.769755642
## tax_class_two                 -0.124595187   -0.76975564   1.000000000
## tax_class_four                -0.055320297   -0.58533129  -0.066999248
##                           tax_class_four
## x1                          -0.003745461
## borough                     -0.118971419
## residential_units           -0.018845557
## commercial_units             0.081690747
## total_units                  0.031797297
```

```
## land_square_feet              0.095592287
## gross_square_feet             0.138139160
## year_built                    0.006005798
## tax_class_at_time_of_sale     0.917074513
## sale_price                    0.144145674
## in_manhattan                  0.151266207
## in_bronx                      0.014061365
## in_brooklyn                   0.023405896
## in_queens                    -0.042987134
## in_staten_island             -0.055320297
## tax_class_one                -0.585331286
## tax_class_two                -0.066999248
## tax_class_four                1.000000000
```

We used "pairwise.complete.obs" to deal with any missing observations or issues. This is not the most reliable way, but it's the simplest for me due to my lack of complete knowledge. From the correlation matrix we are able to see that gross_square_feet has the best correlation at around 0.527, we will want to use this as at least one of our predictors. Additoinally we notice that residential_units, total_units, tax_class_one, tax_class_four, and in_manhattan will be of interest to us too. Before we proceed we'll drop any NA's from these variables of interest.

2.4 Model Creation:

It's finally time to create our model. We need to start with the simplest model, and add on predictors and check for improvements. We'll use the lm() function to create our model and then store the RMSE's in a dataframe to compare results easily. Let's start with the predictor with the highest correlation with sale_price; gross_square_feet.

```r
set.seed(1996)

nyc_partition <- createDataPartition(nyc_data$sale_price, p=0.8, list=FALSE)
nyc_train <- nyc_data[nyc_partition, ]
nyc_test <- nyc_data[-nyc_partition, ]

nyc_rmse_results <- data_frame()  # To store our model and RMSE results

model_gsf <- lm(sale_price~gross_square_feet, data = nyc_train)
summary(model_gsf)
```

```
##
## Call:
## lm(formula = sale_price ~ gross_square_feet, data = nyc_train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.040e+09 -5.938e+05 -4.152e+05 -1.563e+05  1.770e+09
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       5.427e+05  1.101e+05   4.929 8.32e-07 ***
## gross_square_feet 2.772e+02  2.985e+00  92.873  < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16330000 on 22327 degrees of freedom
## Multiple R-squared:  0.2787, Adjusted R-squared:  0.2786
## F-statistic:  8625 on 1 and 22327 DF,  p-value: < 2.2e-16
```

```r
#sigma(model_gsf)  #RMSE

nyc_rmse_results <- data_frame(Model = "gsf", RMSE = sigma(model_gsf))
nyc_rmse_results %>% knitr::kable() # To look at our results
```

Model	RMSE
gsf	16332040

We get an RMSE value of 14800912, and we will definetely want to decrease it. Our next step is to add two new predictors; total_units and residential_units. We can now build this model and test it.

```r
set.seed(1996)

## Model Considering gross_square_feet and residential_units
model_2 <- update(model_gsf, . ~ . + residential_units + total_units)
summary(model_2)
```

```
##
## Call:
## lm(formula = sale_price ~ gross_square_feet + residential_units +
##     total_units, data = nyc_train)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -922554828    -841713    -582320    -212956 1437843060
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.109e+06  9.918e+04  11.187   <2e-16 ***
## gross_square_feet  4.860e+02  3.922e+00 123.928   <2e-16 ***
## residential_units -4.824e+05  8.848e+03 -54.517   <2e-16 ***
## total_units       -6.117e+03  6.150e+03  -0.995     0.32
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14670000 on 22325 degrees of freedom
## Multiple R-squared:  0.4184, Adjusted R-squared:  0.4184
## F-statistic:  5354 on 3 and 22325 DF,  p-value: < 2.2e-16
```

```r
#sigma(model_2)

nyc_rmse_results <- bind_rows(nyc_rmse_results, data_frame(Model = "2", RMSE = sigma(model_2)))
nyc_rmse_results %>% knitr::kable() # To look at our results
```

Model	RMSE
gsf	16332040
2	14665244

With these new predictors we were able to reduce the RMSE to 13522502, an improvement. But we are not done here. Now we want to inlude the tax classes, and see how much we can improve our model.

```
set.seed(1996)
## Model considering gross_square_feet, residential_units, total_units, and tax_class_at_time_of_sale
model_3 <- update(model_2, . ~ . + tax_class_one + tax_class_two + tax_class_four)
#sigma(model_3)

nyc_rmse_results <- bind_rows(nyc_rmse_results, data_frame(Model = "3", RMSE = sigma(model_3)))
nyc_rmse_results %>% knitr::kable() # To look at our results
```

Model	RMSE
gsf	16332040
2	14665244
3	14587655

Again another improvement, but this time not so much. How about we include all our variables? Will this be the best improvement we see? My prediction is yes, even though we have identified a few important predictors, we have noticed just how many different variables affect the sale price. Maybe by including all the numerical predictors we can create the best model yet.

```
set.seed(1996)
# Model Inluding all variables

model_all <- lm(sale_price ~ ., data = nyc_train)
nyc_rmse_results <- bind_rows(nyc_rmse_results, data_frame(Model = "all", RMSE = sigma(model_all)))
nyc_rmse_results %>% knitr::kable() # To look at our results
```

Model	RMSE
gsf	16332040
2	14665244
3	14587655
all	11174750

We improved our model by a lot. We started with an RMSE of 14800912 and now we are at 10746102, a huge improvement from when we started. Something to know is that initially I stopped my project much eariler and was only able to improve my model to an RMSE of about 13200000. I initially thought this was the best I could do, but I decided to go back and include the tax classes. I intially wanted to try to do building classes, but that can be an improvement for the future.

But we are not done yet. From an overall standpoint by using the RMSE we perceive that our final model, model_all, is the best one. But we need to verify this using cross-validation.

2.5 Cross Validation:

We are going to use the train() function from the caret package because it has built-in cross-validation options, unlike lm(), which we were using earlier. We will still use the "lm" method, but we will specify a k-fold cross-validation with k=10. Overall this means that we are training our models using 10-fold cross-validation.

```
set.seed(1996)
# 10-fold Cross Validation for model_gf
# using train() from caret package, specifying lm() method and k =10
(model_gsf_cv <- train(
  form = sale_price ~ gross_square_feet,
  data = nyc_train,
  method = "lm",
  trControl = trainControl(method = "cv", number = 10)
))
```

```
## Linear Regression
##
## 22329 samples
##     1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 20096, 20097, 20097, 20096, 20095, 20096, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   15223708  0.4194236  1416507
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

We notice that we get a cv-RMSE of 13981977. We'll talk more about what this RMSE means later, but first we'll perform cv-models for the rest of our models from earlier.

```
## Linear Regression
##
## 22329 samples
##     3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 20096, 20097, 20097, 20096, 20095, 20096, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   14981932  0.3519446  1724972
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Linear Regression
##
## 22329 samples
```

```
##     6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 20095, 20096, 20095, 20097, 20096, 20097, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   14403823  0.3778356  1562645
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
## Linear Regression
##
## 22329 samples
##    19 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 20096, 20096, 20095, 20096, 20096, 20097, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   12731160  0.5669951  1512990
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
##
## Call:
## summary.resamples(object = resamples(list(modelgsf = model_gsf_cv, model2
##  = model_2_cv, model3 = model_3_cv, modelall = model_all_cv)))
##
## Models: modelgsf, model2, model3, modelall
## Number of resamples: 10
##
## MAE
##             Min. 1st Qu.  Median    Mean 3rd Qu.    Max. NA's
## modelgsf 1098302 1132426 1277757 1416507 1420215 2490812    0
## model2   1345713 1601256 1643881 1724972 1749292 2641175    0
## model3   1239681 1324562 1511470 1562645 1724243 2036177    0
## modelall 1203265 1283163 1436941 1512990 1716583 2011248    0
##
## RMSE
##             Min. 1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## modelgsf 4734663 6358805  8660406 15223708 13462247 44689224    0
## model2   5655997 8130748 10175649 14981932 10659002 41022857    0
## model3   5120613 7447403  9759950 14403823 11695204 37401524    0
## modelall 4996705 6193880  7783978 12731160 18475670 31017554    0
##
## Rsquared
##                 Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## modelgsf 0.010358117 0.1870152 0.3868750 0.4194236 0.5649490 0.8961875    0
## model2   0.007210256 0.0365779 0.1843027 0.3519446 0.7175381 0.9361386    0
## model3   0.032976747 0.1301683 0.2880983 0.3778356 0.6332972 0.8651655    0
```

```
## modelall 0.045632261 0.1984486 0.7314734 0.5669951 0.8558186 0.9341375    0
```

Our model_all continues to be the best model with an average RMSE of 11232332. So turns our using all the numeric predictors is the best option in trying to predict the sales price. This is as far as I am willing to take the linear regression model. There are other methods and options to continue with, but will discuss that later in our conclusion section.

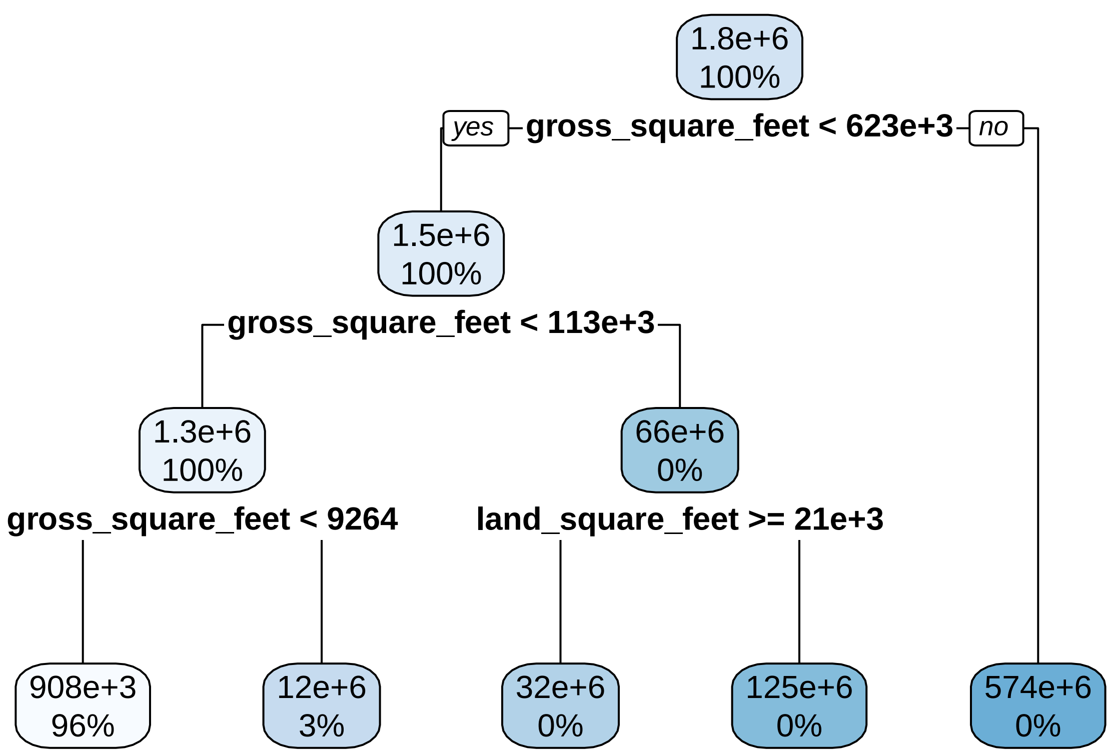
Our next step is to create a decision tree for our data set.

2.6 Decision Tree:

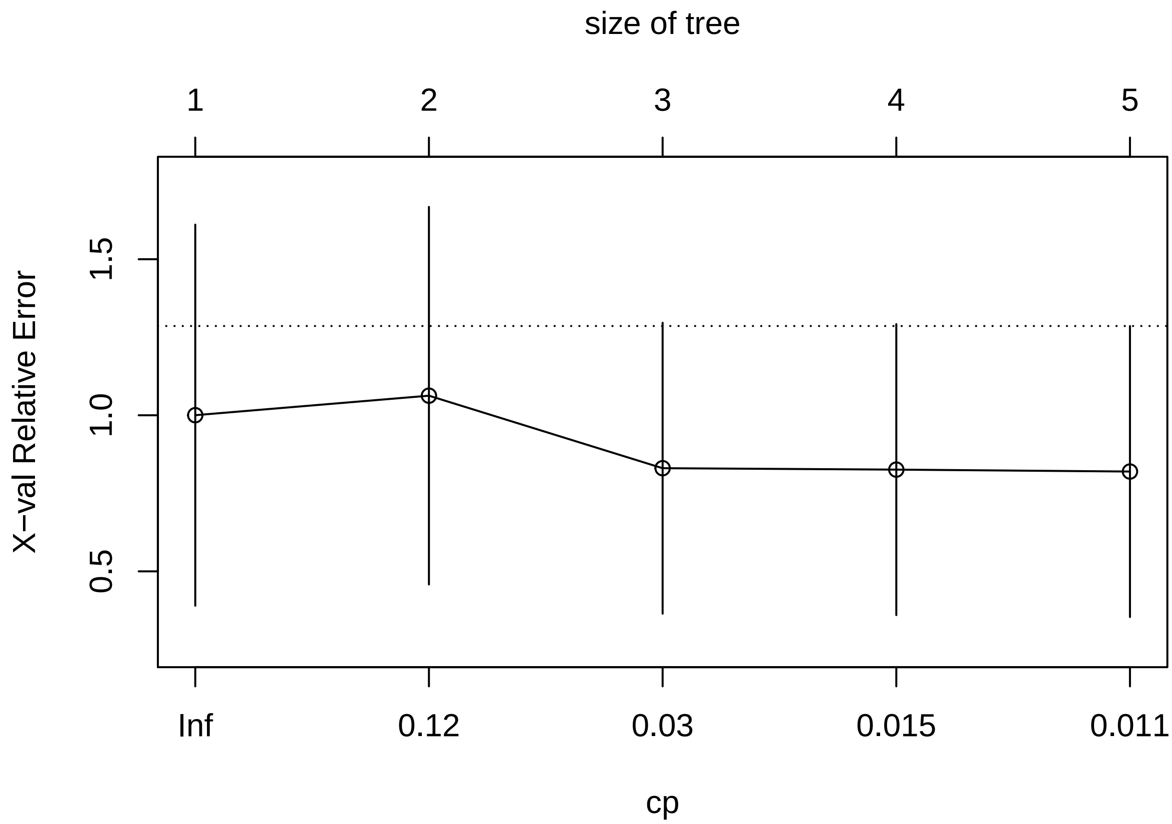
For the decision tree we will be using the rpart and rpart.plot libraries. These are essential in creating our decision tree. We'll build the tree around sale_price and include all predictors.

```
set.seed(1996)
nyc_dec_tree1 <- rpart(
  formula = sale_price ~ .,
  data = nyc_train,
  method = "anova"
)

rpart.plot(nyc_dec_tree1)
```

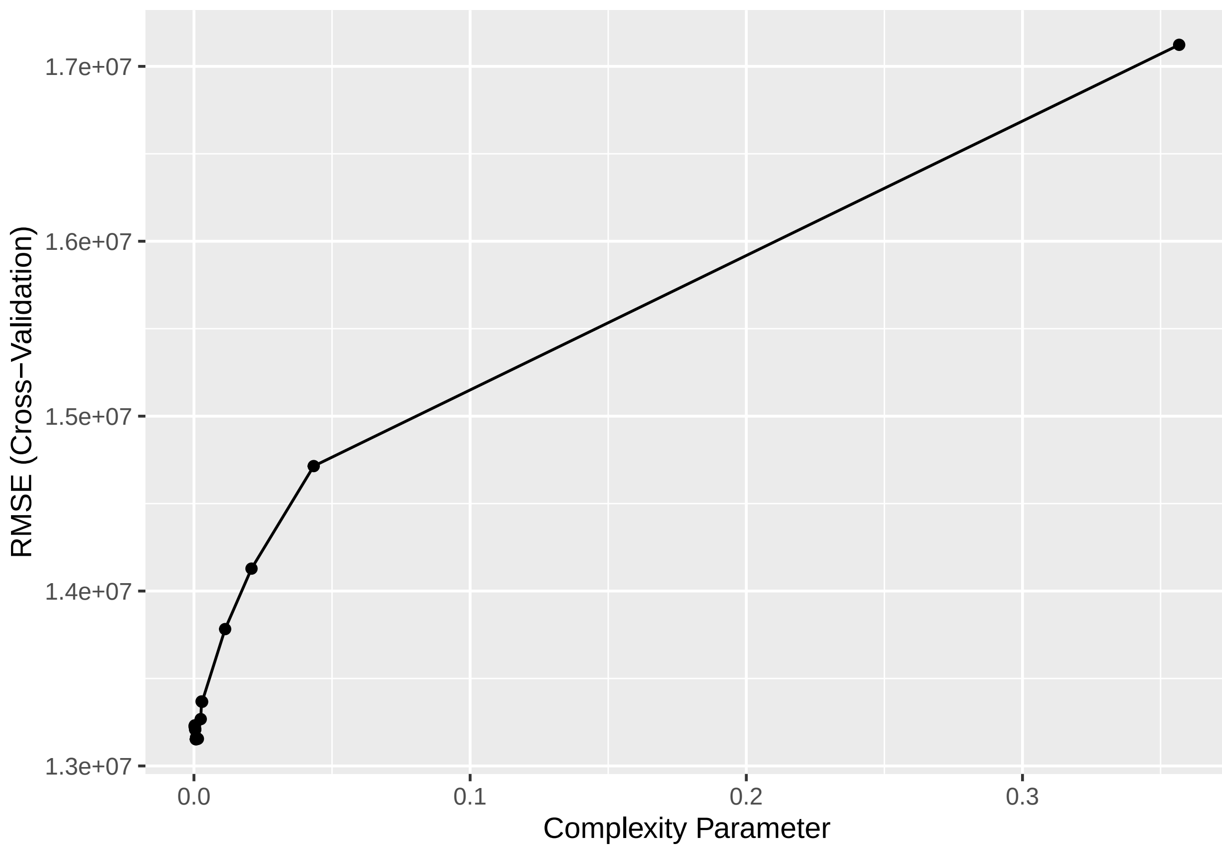


The firs thing that we notice is that rpart recognizes how important gross_square_feet is. But we see on the main problems we have been facing, in that because the ranges are so big in the gross_square_feet we really don't see the best spread in the trees. Even rpart actually self-prunes the tree and places importance on certain features. What we can do is try to see complexity parameter (cp) plot just to illustrate the relative cross-validation error.



Rpart is doing a lot of tuning actually, we can tell because we only see 4 splits even though we have a lot more variables included. We can actually take a look at the rpart cross validation results. We'll go ahead and use caret again to perform cross validation and show the results too.

```
##           CP nsplit rel error    xerror      xstd
## 1 0.35672879      0 1.0000000 1.0002197 0.6105331
## 2 0.04336195      1 0.6432712 1.0627580 0.6048473
## 3 0.02084179      2 0.5999093 0.8304472 0.4660979
## 4 0.01128448      3 0.5790675 0.8258631 0.4661759
## 5 0.01000000      4 0.5677830 0.8197133 0.4661711
```



This graph shows the cross-validated accurace rate for our different alpha parameters. We notice that by keeping the alpha value low, resulting in deeper trees, it would help reduce errors. Overall the decision tree really didn't help because we already knew of the importance of the gross_square_feet. But it does point out a major problem overally in our problem, which is the scope of our data is too large. We'll discuss this more in the results and conclusion sections.

3. Results:

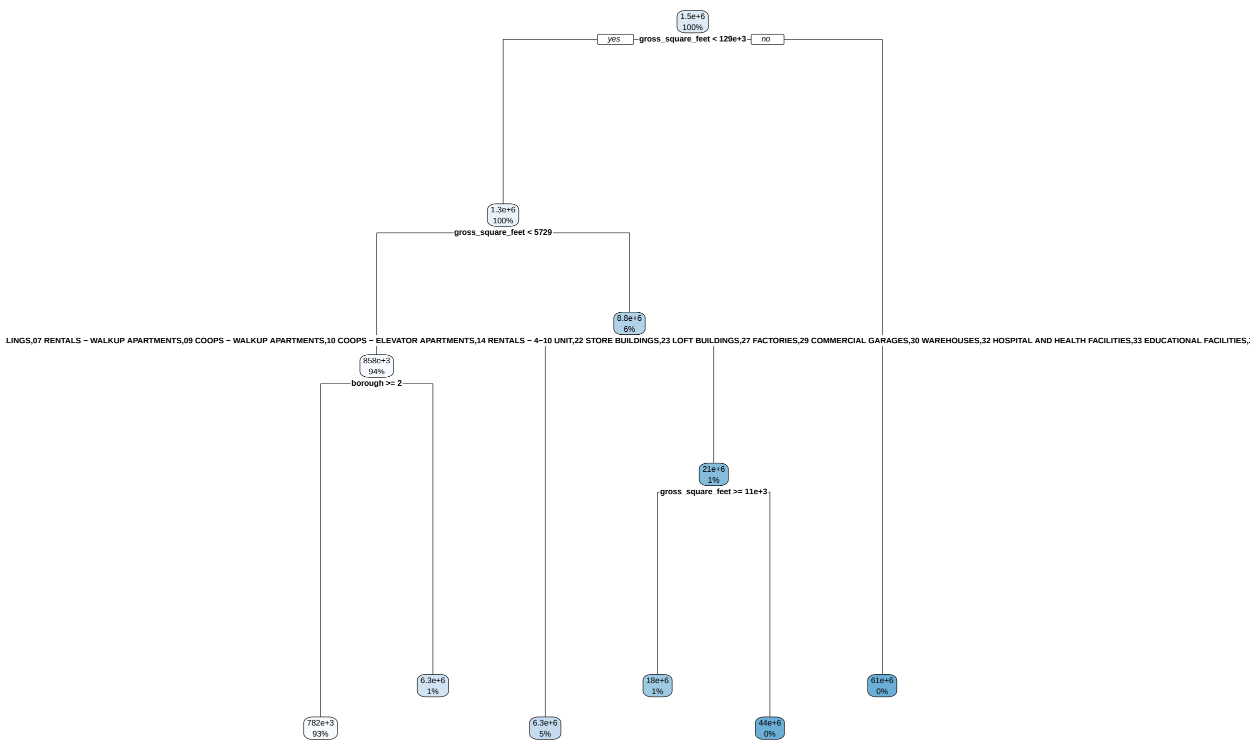
We can see the final model_all that has been cross-validated still performs the best. We get an average RMSE of 12731160, which is the lowest among all the cv models. We can test to see how this model does with the test set.

```
## Linear Regression
##
## 5581 samples
##   19 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5022, 5022, 5024, 5021, 5024, 5023, ...
## Resampling results:
##
##   RMSE     Rsquared   MAE
##   5533742  0.3389189  982608.6
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

We see that we get an RMSE value of 12117002, which is lower than the RMSE we got on our train set. This is good, it means our model is working similarily, if not better. The downside is that we notice our Rsquared term is lower than what we got on the training set, but that's fine.

Overall what do we understand about this model's performance? My overall interpertation is very poor. With an RMSE value of 1211700 on our final validation set, we are saying that our model's prediction is $1,211,700 off the actual sale price, and this is very poor. It is correct that we have reduced our RMSE from where we started, but from an overall performance stand-point this is still poor. We'll discuss some possible reasonings and areas of improvement in the conclusion.

We'll now evaluate the decision tree, and see if we see any significant changes with our test data set.



No we don't see a difference in the probability distribution. Which is good, but we still see a major flaw from a problem statement standpoint, our scope was too big. We immedietly see how the decision tree has huge ranges for the gross_square_feet and places the probabilities in such huge end nodes. We'll discuss how these decision trees can help in a future project.

Overall while our performance improved at each step in our linear regression model, our decision tree really didn't help us identifying other important parameters, or identifying interesting points. Additionally our linear regression model is still way off in being able to come close to predicting sales prices in NYC property sales.

4. Conclusion:

Our objective on this project was to create models to predict property sales in NYC. We first downloaded, then processed the data into a form we could easily handle, while removing outliers. We then set out to identify key parameters and built our linear models using lm(). We then used 10-fold cross validation to check our model performance. We then created a decision tree and indentified the poor results from both.

Sadly we did not create a ground-breaking way to predict NYC property sales. Instead we learned a valuable lesson in how their are a lot of variables that can affect sales prices, and we need to understanding how to incorporate such variables. For example, how can different months affect prices, do specific zipcodes in different boroughs affect price, etc? Their are a lot of different ways that we can try to improve our overall model.

But the most important thing we could do is reduce the scope of the project. Instead of trying to predict property sales in all of NYC, we could focus on specific boroughs, or specific building classes, like single family dwellings. This approach of starting small could help improve our initial project goal, of trying to better predict NYC property sales. By understanding what is happening on a more micro model, we can better understand how different predictors can vary as you "zoom-out."

A great example of this is in our decision trees, we notice immedietly how rpart() emphasizes gross_square_feet, which makes sense. The more area of your property, the higher sales price, generally speaking of course. But what did not help was just how big the probability distribution was in our terminatino nodes. Our tree essentially predicted that 96% of our sales are most likely going to be less than 9543 square feet. It would be very interesting to see how these decision trees would look if created multiple trees for data sets focusing on each building classification.

However we were limited by the fact that a lof the classifications need to be encoded, by encoding we can better detect patterns, and possible make for better tree models.

Even though I am dissapointed in the results, I am happy that I learned a valuable lesson in looking to start small to better understand your data. This was one of my first times working on creating statistical models, and I feel I have learned a lot of valuable skills that I can apply to real-world problems. Overall it maybe better to have small goals than try to aim huge and not get results you were hoping for.