HarvardX: PH125.9x Final Project

By Philip J Brown, pjbMit@pjb3.com

6/16/2019

HarvardX Data Science Capstone Class PH125.9x (2T2018)

- Student: Philip Brown
- email: Phil@pjb3.com
- github: https://github.com/pjbMit

RealML - *A Real Estate Machine Learning Project*

This is RealML, a real estate machine learning project and report created by Philip J Brown (pjbMit@pjb3.com) as the final capstone project for the Data Science Certificate program offered by HarvardX: PH125.9x from edx.org.

Part 1) EXECUTIVE SUMMARY

The goal of this project is to utilize data analysis and modelling skills to a create machine learning engine and this report as the final exercise in completing the 9 course Data Science Certificate program offered by HarvardX through edx.org.

For my project I chose to use machine learning techniques to build a *RealML,a real estate sales price prediction engine.* More specifically, I wanted to answer this question:

> **Can I reasonably predict the resale price of residential condominum and single family real estate within a five mile radius of *Fairlington Villages (link)*, the condominum development in Arlington Virginia that I call home?**

Through this project, I am able to demonstrate examples of data identificaton and acquisition, data wrangling and cleansing, data analysis, modeling and machine learning techniques, data presentation and data visualization and report generation and presentation. The project was built by acquiring and analyzing more than 20,000 reports of current real estate sales within the stated five mile radius for residential properties that sold for at least $5,000 but less than $1,000,000.

After some research I was able to locate and curate live data for this project, so the basis for this report is real and impactful – at least it is to me as home owner in this area. * :-)

The results of this analysis were very encouraging, and are included in the **results** section and the **conclusion** section, which are the last two sections in this report.

The mission was to locate, curate, wrangle and cleanse real data, and use it build a prediction engine that tries to predict sales prices so as to optimize the model for a low Root Mean Square Error (RMSE), defined as

$$RMSE = \sqrt{\Sigma_{i=1}^{n} \frac{(actual_i - predicted_i)^2}{N}}$$

This project is intended to highlight some of the skills acquired throughout the courses in this program. All programming was done in R Code using RStudio on MacBook Pro. After acquiring and processing the data, the real work began!

In addition to this **executive summary**, this report also includes a **methods and analysis section**, a **results section** and a **conclusion section**.

Key files for this project have been uploaded and stored on my git hub page at github.com/pjbMit/real_estate_project. The three main files for this project are listed below, and can be viewed on git hub – The file names are also links:

- real_ml_script.R *(link)*
- real_ml_report.Rmd *(link)*
- real_ml_report.pdf *(link)*

Additionally, a gzip'd version of the data file is on github at:

- realml_data_file.github.json.gz *(link)*

Part 2) METHODS AND ANALYSIS

The project was created in the RStudio environment using Rstudio Version 1.1.442 on a Macintosh; Intel Mac OS X 10_14_5

R version 3.5.1 (2018-07-02)
nickname Feather Spray

All code was written in R and executed in RStudio.

Here are the methods and techniques used.

Data was downloaded from AttomData.com, a commercial data provider, using their RESTful API and an apikey that is needed in order to get data. Sales data was queried from their API, and results were downloaded 10,000 rows at a time.

Working code to download from Atom Data is included in the .Rmd script, but the function call is commented out.

you may run it by calling **download_and_save_web_data()** which is a function that will connect to the Restful API service, query for the next 10,000 matching rows, and loop five times, downloading up to 50,000 rows of the most recent real estate sales data. Once the data is filtered, you are likely to end up with about 1/2 as many records that in scope for this project.

The data was then filtered to remove property types that aren't residential condos or homes. Data wrangling skills were used to massage and clean up the data. A fairly clean version of the data with Factors is within the factorData object.

Here are some summary results produced from the factorData object:

```
## [1] "Summary of data loaded"
```

```
##   newest_sale oldest_sale
## 1  2019-05-06  2012-01-03
```

```
## [1] "show filtered data cross tab by type and subtype --"
```

```
## [1] "which indicates a little filtering and cleaning is needed."
```

Part 3) RESULTS

After downloaing the raw data, we had sales information covering the following sales date range:

We found additional filter criteria to help us cleanse the data. For example, shown below is the data after filtering to show just the property types and subtypes of interest, which had the effect of removing commercial sales, industrial sales, and other data that is out of scope for this project.

After examining the data, we saw that a little data-cleansing house keeping was in order. We found that about 10% of the data had zero listed for bedrooms, yet those units had about 1400 sqft on average (mean), thus the zero bedrooms was clearly an error. We removed these rows along with two unneeded colums.

```
#Look for zero bedrooms, and determine their mean sqft.
factorData %>% filter(beds==0) %>% summarize(num=n(),mean(sqft))
```

```
##    num mean(sqft)
## 1 3923   1455.371
```

```
#remove zero bedroom errors, and remove unneeded columns.
myData <- factorData %>% filter(beds != 0) %>%
            select(-transtype,-propsubtype)
```

After cleansing the data, here's a grouped summary showing the data by year and property type:

```
##             year_sold 2012 2013 2014 2015 2016 2017 2018 2019
## proptype
## CONDOMINIUM            970 1305 1303 2429 2341 1852 1897  278
## SFR                   1549 1727 1661 3136 2993 2117 2048  419
```

```
## [1] 28025
```

```
## [1] 2452
```

```
## [1] 2452   13
```

That's a decent amount of *current* data. If I can just tun the model appropriately, we should be good to go.

Now it's time to look at a summary() of the data to get a better feel some the individual data attributes.

```
##       beds            baths            sqft         yearbuilt
##  Min.   :1.000   Min.   :1.000   Min.   : 510   Min.   :   0
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.: 900   1st Qu.:1945
##  Median :2.000   Median :2.000   Median :1150   Median :1961
##  Mean   :2.468   Mean   :2.078   Mean   :1328   Mean   :1964
##  3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:1568   3rd Qu.:1986
##  Max.   :8.000   Max.   :7.000   Max.   :5656   Max.   :2017
##
##       lat             lon             saledate              zip
##  Min.   :38.77   Min.   :-77.19   Min.   :2018-06-02   22204  :318
##  1st Qu.:38.84   1st Qu.:-77.12   1st Qu.:2018-07-30   22201  :263
##  Median :38.86   Median :-77.10   Median :2018-10-05   22206  :192
##  Mean   :38.86   Mean   :-77.10   Mean   :2018-10-17   22203  :159
##  3rd Qu.:38.88   3rd Qu.:-77.08   3rd Qu.:2018-12-21   22207  :127
##  Max.   :38.91   Max.   :-77.01   Max.   :2019-05-06   22310  :127
##                                                        (Other):777
##            city      state            proptype
##  Alexandria  : 394   DC:  70   CONDOMINIUM:1055
##  Annandale   :  31   VA:1893   SFR        : 908
##  Arlington   :1348
##  ARLINGTON   :   0
##  Falls Church: 112
##  Springfield :   8
##  Washington  :  70
##                         addr          price
##  1024 N UTAH ST APT 423  :   2   Min.   :  30000
##  10 HALLEY PL SE APT 303 :   1   1st Qu.: 379000
##  100 N TRENTON ST # 100-1:   1   Median : 525000
```

```
##  1000 20TH ST S          :   1   Mean   : 542073
##  1000 26TH ST S          :   1   3rd Qu.: 715000
##  1000 N KENTUCKY ST      :   1   Max.   :1000000
##  (Other)                 :1956
```

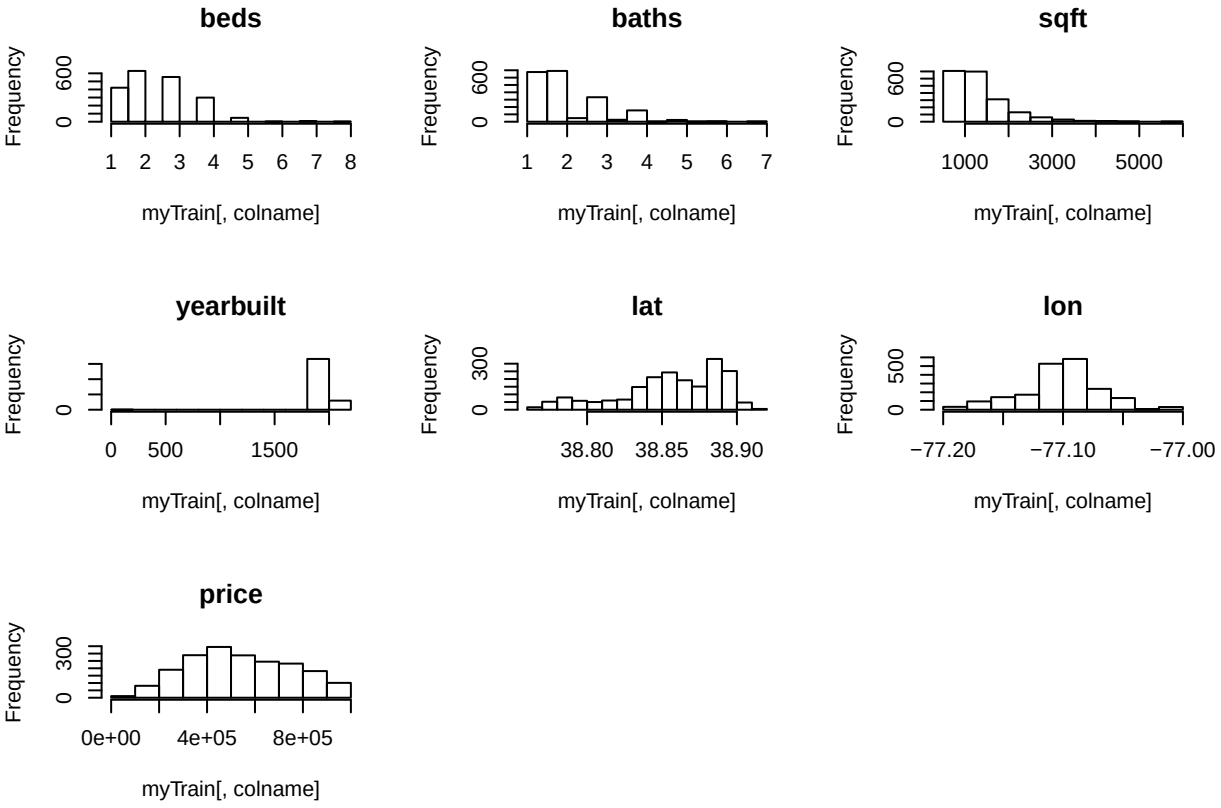
```
##      beds     baths      sqft yearbuilt       lat       lon  saledate
## "integer" "numeric" "integer" "integer" "numeric" "numeric"    "Date"
##       zip      city     state  proptype      addr     price
##  "factor"  "factor"  "factor"  "factor"  "factor" "integer"
```

```
##       beds           baths            sqft        yearbuilt
##  Min.   :1.000   Min.   :1.000   Min.   : 510   Min.   :   0
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.: 900   1st Qu.:1945
##  Median :2.000   Median :2.000   Median :1150   Median :1961
##  Mean   :2.468   Mean   :2.078   Mean   :1328   Mean   :1964
##  3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.:1568   3rd Qu.:1986
##  Max.   :8.000   Max.   :7.000   Max.   :5656   Max.   :2017
##
##       lat             lon            saledate               zip
##  Min.   :38.77   Min.   :-77.19   Min.   :2018-06-02   22204  :318
##  1st Qu.:38.84   1st Qu.:-77.12   1st Qu.:2018-07-30   22201  :263
##  Median :38.86   Median :-77.10   Median :2018-10-05   22206  :192
##  Mean   :38.86   Mean   :-77.10   Mean   :2018-10-17   22203  :159
##  3rd Qu.:38.88   3rd Qu.:-77.08   3rd Qu.:2018-12-21   22207  :127
##  Max.   :38.91   Max.   :-77.01   Max.   :2019-05-06   22310  :127
##                                                        (Other):777
##            city      state            proptype
##  Alexandria  : 394   DC:  70   CONDOMINIUM:1055
##  Annandale   :  31   VA:1893   SFR        : 908
##  Arlington   :1348
##  ARLINGTON   :   0
##  Falls Church: 112
##  Springfield :   8
##  Washington  :  70
##                         addr          price
##  1024 N UTAH ST APT 423  :   2   Min.   :  30000
##  10 HALLEY PL SE APT 303 :   1   1st Qu.: 379000
##  100 N TRENTON ST # 100-1:   1   Median : 525000
##  1000 20TH ST S          :   1   Mean   : 542073
##  1000 26TH ST S          :   1   3rd Qu.: 715000
##  1000 N KENTUCKY ST      :   1   Max.   :1000000
##  (Other)                 :1956
```

The text is useful, but graphs are better...

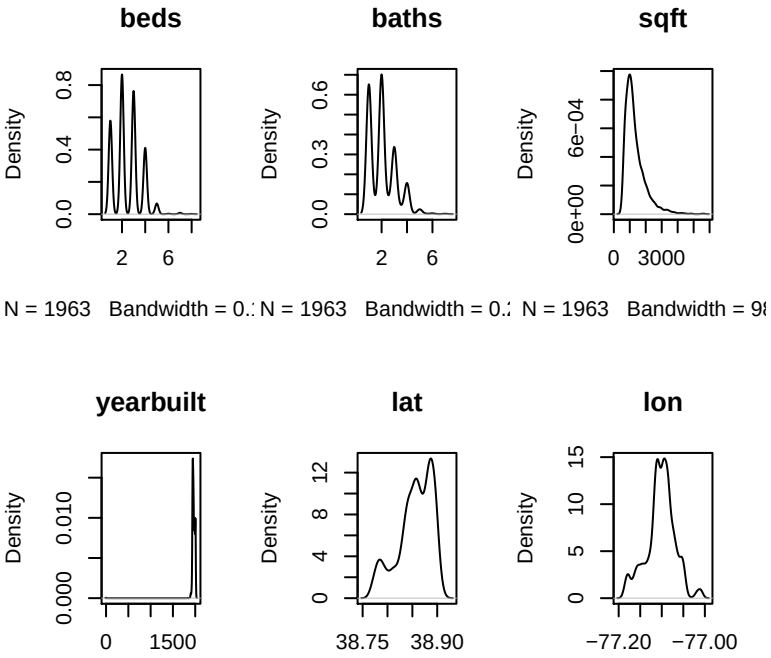
```
##          beds                 baths                sqft
## breaks   Numeric,15           Numeric,13           Integer,12
## counts   Integer,14           Integer,12           Integer,11
## density  Numeric,14           Numeric,12           Numeric,11
## mids     Numeric,14           Numeric,12           Numeric,11
## xname    "myTrain[, colname]" "myTrain[, colname]" "myTrain[, colname]"
## equidist TRUE                 TRUE                 TRUE
##          yearbuilt            lat                  lon
## breaks   Numeric,12           Numeric,17           Numeric,11
## counts   Integer,11           Integer,16           Integer,10
## density  Numeric,11           Numeric,16           Numeric,10
```

```
## mids     Numeric,11           Numeric,16           Numeric,10
## xname    "myTrain[, colname]" "myTrain[, colname]" "myTrain[, colname]"
## equidist TRUE                 TRUE                 TRUE
##          price
## breaks   Numeric,11
## counts   Integer,10
## density  Numeric,10
## mids     Numeric,10
## xname    "myTrain[, colname]"
## equidist TRUE
```



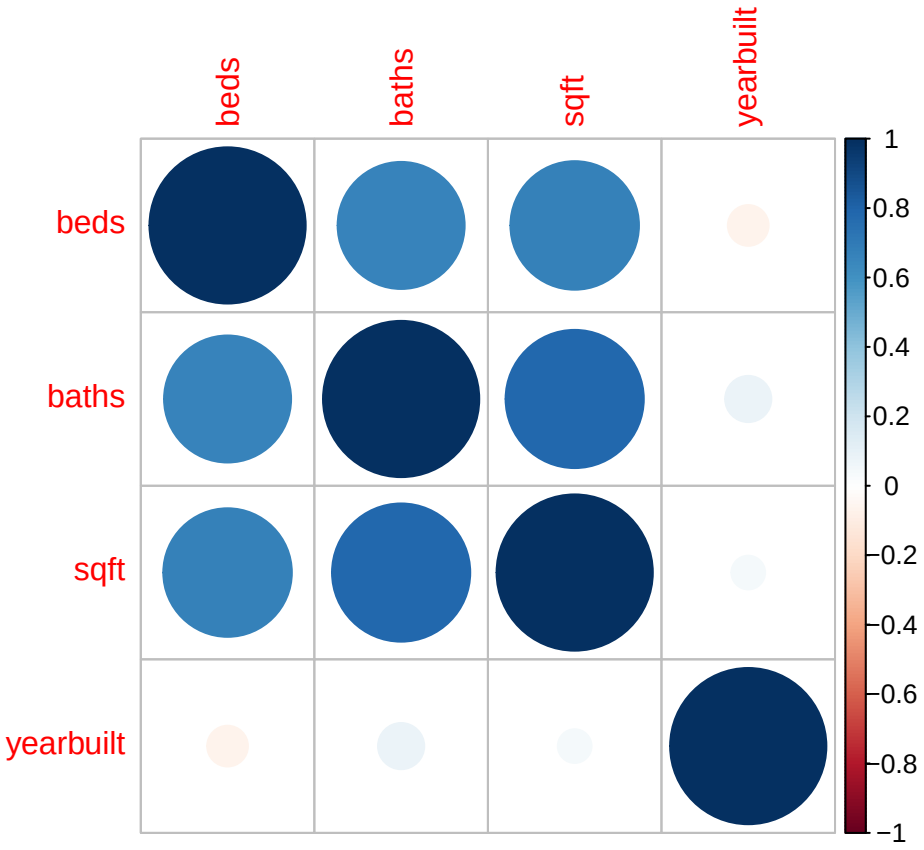
The bar plots are a little jagged. Lets get a different perspective by smoothing out the graphs using a density plot. This helps us better visualize the data to see if we have a binomial distribution, or other anomally.

```
par(mfrow=c(2,3))
for(i in c(1:6)) {
    plot(density(myTrain[,i]), main=names(myTrain)[i])
}
```



The SqFt, in particular has a long tail towards some very large houses. This *could* be a problem for some models, and might be a source of model error. (Update – after looking into the rather large model errors, I believe that the skew towards large million dollar houses, has the effect either through the model or through my sampling methods, of biasing the model to over-estimated. I suspect that other models could compensate, especially one the uses regularization. I ran almost a dozen different models. Quite truthfully, something made my models *exceptionally* slow, which kept me from working more with different models.)

Next, we looked the remaining columns to see how they correlated, to see if we can remove any columns that are highly correlated.



The data is highly correlated. We use the calculation below to determine which attribute, if any, we should consider removing to see if our models improve...

```
set.seed(2020)
cutoff <- 0.70
correlations <- cor(myTrain[,c("beds","baths","sqft","yearbuilt")])
highlyCorrelated <- findCorrelation(correlations, cutoff=cutoff)
for (value in highlyCorrelated) {
    print(names(myTrain)[value])
}
```

```
## [1] "baths"
```

and the answer turns out to be baths. So, as part of the data modelling, we should experiment with models both with and with baths in them, to see if removing baths helps the model perform better.

Modelling

Now it's time to explore our data models. Let's try a mix of non-linear models, and see how we do. Note that we enabled parallel processing in the caret package by loading the **doMC** package and library. (If it gives you trouble, comment it out on lines 43 and 44 in the script. For me, it made some modes wayyyyyy faster!!!)

Here are the models from our first run.

```
# Run algorithms using 10-fold cross-validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
model_cols <- c("beds","baths","sqft")
trainX <- myTrain[,model_cols]
trainY <- myTrain[,"price"]
```

```r
y_hat <- mean(trainY)
y_hat
```

```
## [1] 542073.2
```

```r
set.seed(2931)


fit1 <- train(trainX, trainY, method="knn",tunelength = 10,
                  trControl=trainControl)



grid <- expand.grid(.cp=c(0, 0.05, 0.1))

set.seed(2931)
fit2 <- train(trainX, trainY, method="rpart",
                  preProcess =c("center","scale"),
                  tuneLength = 9,
                  tuneGrid=grid,
                  trControl=trainControl)

res<-resamples(list(knn=fit1,rpart=fit2))
summary(res)
```

```
##
## Call:
## summary.resamples(object = res)
##
## Models: knn, rpart
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn   102663.40 109902.4 113414.4 114265.2 118284.4 130728.5    0
## rpart  97243.74 107957.2 113773.1 113198.4 119292.5 128387.7    0
##
## RMSE
##           Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn   129145.9 140716.4 146125.9 147453.1 154463.1 169525.6    0
## rpart 125640.6 142110.8 148207.9 148660.1 155899.8 168538.0    0
##
## Rsquared
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn   0.3825521 0.5041984 0.5419801 0.5385767 0.5769404 0.6490939    0
## rpart 0.4129235 0.5018945 0.5383389 0.5354760 0.5823529 0.6790664    0
```

```r
# Stochastic Gradient Boosting
set.seed(2931)
fit3 <- train(trainX, trainY, method="gbm",
                 trControl=trainControl, verbose=FALSE)
```

```r
model_cols2 <- c("beds","baths","sqft","lat","yearbuilt","zip")
trainX2 <- myTrain[,model_cols]
trainY2 <- myTrain[,"price"]

set.seed(2931)
fit4 <- train(trainX2, trainY2, method="knn",tunelength = 10,
              preProc=c("center", "scale"), trControl=trainControl)


set.seed(2931)
fit5 <- train(trainX2, trainY2, method="rpart",
              preProcess =c("center","scale"),
              tuneLength = 9,
              tuneGrid=grid,
              trControl=trainControl)

# Stochastic Gradient Boosting
set.seed(2931)
fit6 <- train(trainX2, trainY2, method="gbm",
              trControl=trainControl, verbose=FALSE)


model_cols3 <- c("beds","sqft","lat","yearbuilt","zip")
trainX3 <- myTrain[,model_cols]
trainY3 <- myTrain[,"price"]

set.seed(2931)
fit7 <- train(trainX3, trainY3, method="knn",tunelength = 10,
              preProc=c("center", "scale"), trControl=trainControl)


set.seed(2931)
fit8 <- train(trainX3, trainY3, method="rpart",
              preProcess =c("center","scale"),
              tuneLength = 9,
              tuneGrid=grid,
              trControl=trainControl)

# Stochastic Gradient Boosting
set.seed(2931)
fit9 <- train(trainX3, trainY3, method="gbm",
              trControl=trainControl, verbose=FALSE)


results <- resamples(list(knn=fit1, rpart=fit2, gbm=fit3, knn_2=fit4, rpart_2=fit5, gbm_2=fit6))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: knn, rpart, gbm, knn_2, rpart_2, gbm_2
## Number of resamples: 30
##
```

```
## MAE
##              Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn     102663.40 109902.4 113414.4 114265.2 118284.4 130728.5    0
## rpart    97243.74 107957.2 113773.1 113198.4 119292.5 128387.7    0
## gbm     100681.44 106503.1 108852.6 110119.5 112562.0 124161.1    0
## knn_2   101947.96 106820.6 112511.8 111346.4 114584.1 121547.5    0
## rpart_2  97243.74 107957.2 113773.1 113198.4 119292.5 128387.7    0
## gbm_2   100681.44 106503.1 108852.6 110119.5 112562.0 124161.1    0
##
## RMSE
##             Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn     129145.9 140716.4 146125.9 147453.1 154463.1 169525.6    0
## rpart   125640.6 142110.8 148207.9 148660.1 155899.8 168538.0    0
## gbm     128222.2 135788.6 140608.0 141647.8 144628.3 164723.1    0
## knn_2   130482.4 139565.3 145296.0 145443.9 149278.5 165034.8    0
## rpart_2 125640.6 142110.8 148207.9 148660.1 155899.8 168538.0    0
## gbm_2   128222.2 135788.6 140608.0 141647.8 144628.3 164723.1    0
##
## Rsquared
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn     0.3825521 0.5041984 0.5419801 0.5385767 0.5769404 0.6490939    0
## rpart   0.4129235 0.5018945 0.5383389 0.5354760 0.5823529 0.6790664    0
## gbm     0.4167143 0.5477817 0.5763397 0.5712591 0.6080254 0.6602776    0
## knn_2   0.4264019 0.5234677 0.5485585 0.5507051 0.5821683 0.6521425    0
## rpart_2 0.4129235 0.5018945 0.5383389 0.5354760 0.5823529 0.6790664    0
## gbm_2   0.4167143 0.5477817 0.5763397 0.5712591 0.6080254 0.6602776    0
```

```
results2<- resamples(list(knn=fit7, rpart=fit8, gbm=fit9))
summary(results2)
```

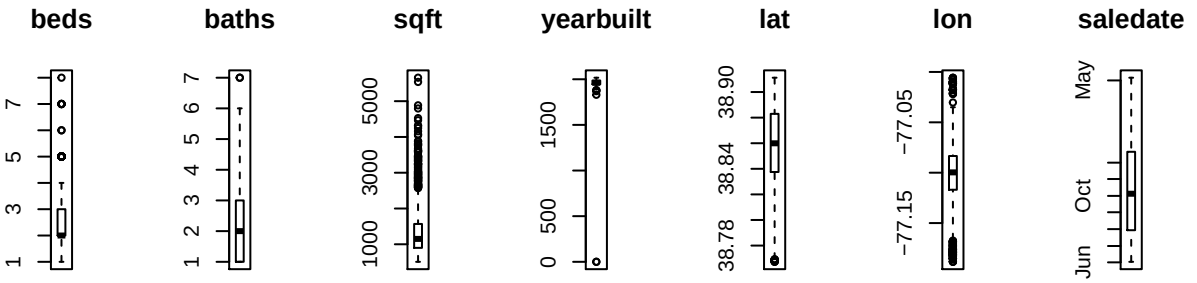
```
##
## Call:
## summary.resamples(object = results2)
##
## Models: knn, rpart, gbm
## Number of resamples: 30
##
## MAE
##             Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn    101947.96 106820.6 112511.8 111346.4 114584.1 121547.5    0
## rpart   97243.74 107957.2 113773.1 113198.4 119292.5 128387.7    0
## gbm    100681.44 106503.1 108852.6 110119.5 112562.0 124161.1    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## knn    130482.4 139565.3 145296.0 145443.9 149278.5 165034.8    0
## rpart  125640.6 142110.8 148207.9 148660.1 155899.8 168538.0    0
## gbm    128222.2 135788.6 140608.0 141647.8 144628.3 164723.1    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn    0.4264019 0.5234677 0.5485585 0.5507051 0.5821683 0.6521425    0
## rpart  0.4129235 0.5018945 0.5383389 0.5354760 0.5823529 0.6790664    0
## gbm    0.4167143 0.5477817 0.5763397 0.5712591 0.6080254 0.6602776    0
```

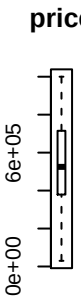
And here are the results of the output.

So, the average home cost is about 540K, and our best RSME is about 140K... that's not very good. We need to look into the data and find out what to change to do better.

Let's add "lat","yearbuilt","zip" to the model, and rerun it both with and without "bath" as a paramter. The results are below...

I'm still not happy with the results. Let's look at a box-and-whisker plot for more insight. The outliers will deserve more attention, and a box and whisker plot can help us look for skews in the data.





Time to drill down into the sources of the errors...

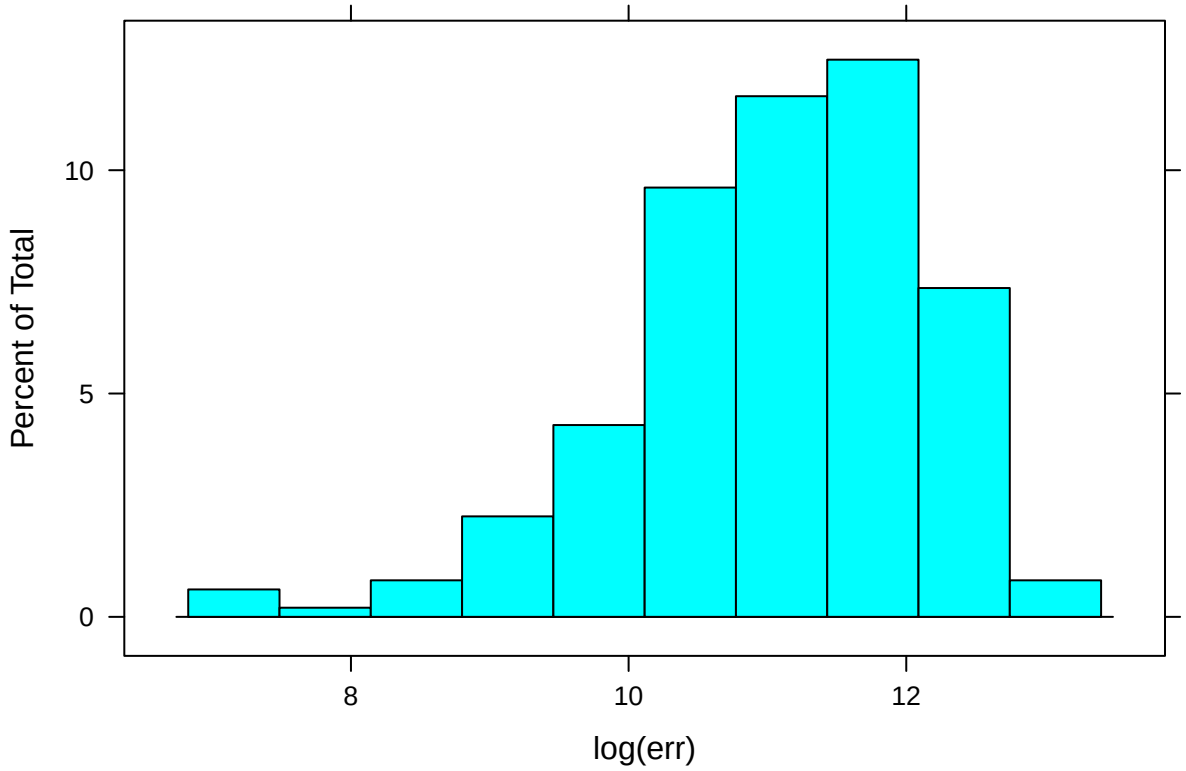
```
##  [1] 340500 385000 495000 880000 590000 920000 719000 695000 800000 450000
## [11] 705000 280000 339000 500000 639000 435000 445000 410575 703205 546500
## [21] 429000 540000 470000 268000 490000 430000 490000 440000 779950 510000
```

```
## Observations: 489
## Variables: 3
## $ y  <int> 340500, 385000, 495000, 880000, 590000, 920000, 719000, 695...
## $ p1 <dbl> 354218.8, 875593.3, 818670.6, 847659.3, 642289.7, 826220.0,...
## $ p2 <dbl> 354636.4, 894277.8, 821387.8, 810543.3, 630777.8, 782277.8,...
```

```
## [1] 148658.7
```

```
## [1] 144065.9
```

```
## Warning in log(err): NaNs produced
```

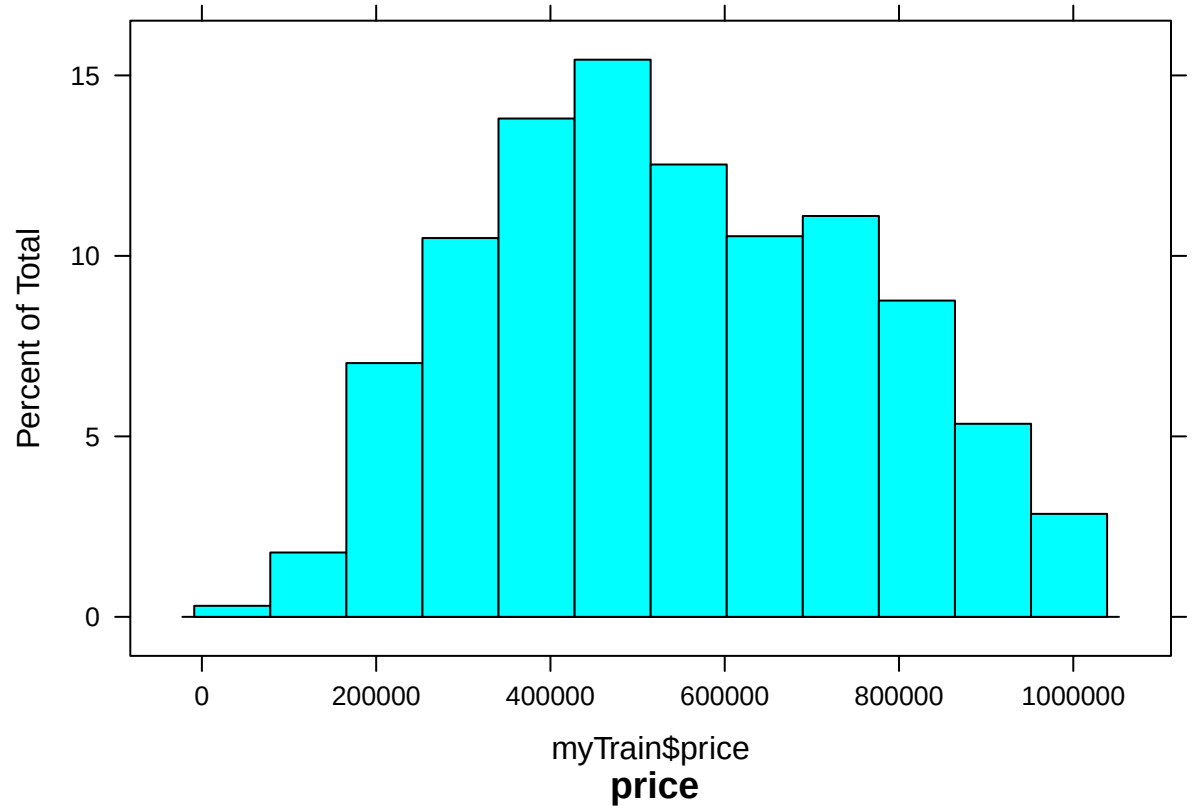


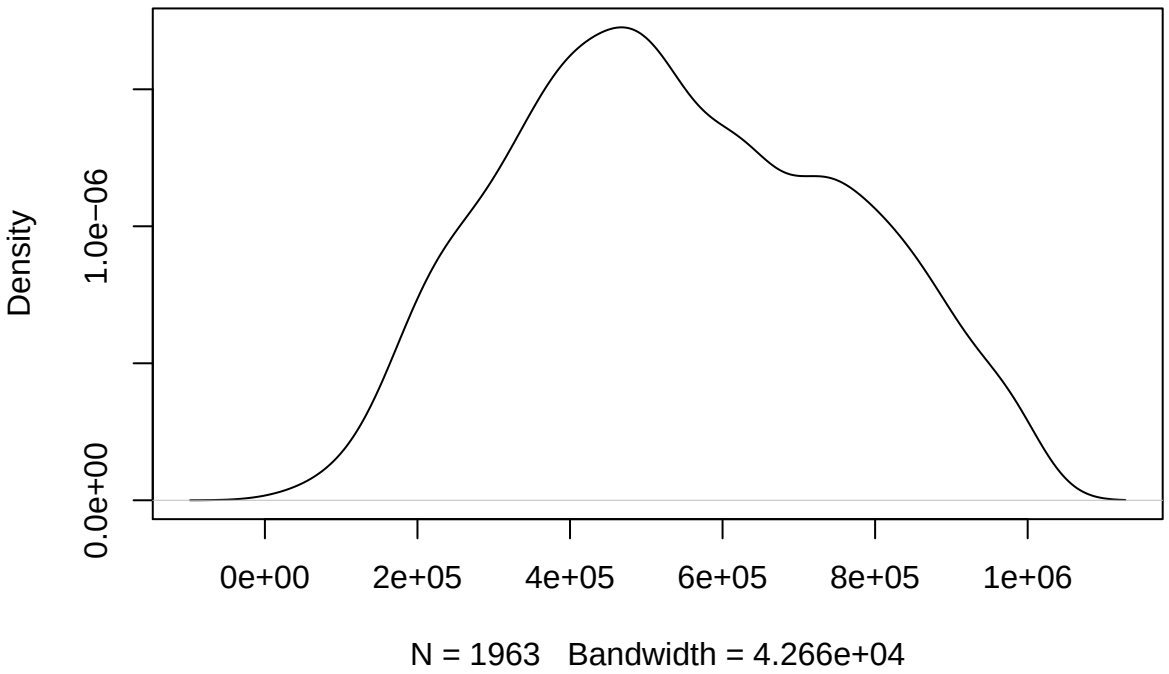
Putting the errors on a logarithmic scale, helps clarify the skew. Our error distribution is skewed, and we have a small number of very large over-estimates. For example, the data above shows that one of the y's is $385K, but two of the models predicted $875K and $894k!.

Part 4) CONCLUSION

Massive over-predictions are skewing the model. It will take more investigation and research to build a better model. I had initially used the log function with the createDataPartition argument to impement binning so that we would get better represeentation from the whole range of values. I thought that this might have skewed the model, but when I re-ran it without binning, the results weren't much better.

Next, I'd like to try some of the models the implement regularization, since I suspect that a small number of very expensive homes are skewing the model.





As the two price plots above show, it seems like the prices themselves have an underlying bi-nomial distribution. There are the main group of houses centered around $500K, plus there appears to be another group of homes centered around $850K. Armed with this insight, the search will contiue! I suspect that some of the machine learning algorithms may handle this situation with ease. Certainly more experimentation and research is called for. Plus, for models that have tunining paramters, some manual tuning may be required. Note that I used a grid for some of the models, and took advantage of the grid tuning inherint with specifying the model

and a grid.

I have demonstrated a desire to pick a challenging topic – namely, getting real, live data for *current* home sales near my home. The search, learnign the api, getting access to the data, and coding to pull the data are all part of a good data science exercise. It would have been easier to have selected a pre-cleansed data set of off kaggle, and followed a scripted approach to building a model, but I decided to be more adventurous. I know that with time, experimentation and maybe even collaboration with others in the field, that I will be able to improve the model results substantially. Ensemble models, and ones with feature identification could all help. Nonetheless, I consider this assignment a success, in that I pulled together many of the skills and lessons tauht in this program to produce running code, data analysis, graphics and some insight. Yes, I realize that the machine learning algorithm needs more work, but I *did* implement several, and worked through them. The project was very useful. It helped clarify many of the subtlties that you can only learn by *doing*.

One last note. For quite some time, my machine ran exceptionally slow on models. I eventually re-wrote my model code, and changed to using the x,y format instead of the formula format. There must have been some error in my previous code, because suddenly, these models ran orders of magnitude faster. This isn't a bad thing – it's all part of the journey, and the reason why I enrolled in the program – namely, to learn by doing. With those barriers behind me, I look forward to my next challenge.

I also wanted to highlight that I installed the *doMC* package, and configured it to use 8 parallel threads for some the carret packages... what a differnce multi-core computing makes!

I believe that this assignment sucessfully fulfilled the requirements. To answer my initial question, right now I would have to say "No!" – I have not yet built a machine learning model that is accurate enough to be impressive, but, my road has just begun, and I *did* build a machine learning model. It will be an itereative process, and a fun one. Cheers!

```
## [1] "Thanks for checking this out!  pjbMit@pjb3.com  :-)"
```