Intro to data science

isken

August 14, 2017

This is a short R Markdown document based on Chapter 1 of "Practical Data Science with R". This chapter introduces the process of data science (business analytics) projects. It is NOT meant to be an example of an exhaustive analysis. It's just meant to give you a peek at the nature of a predictive modeling project and how R can be used to support such analytical projects. Yes, it's overly simplified and sanitized. We'll get into more messy realities throughout the semester.

Preliminaries

We need to set the working directory to the directory containing this file.

Session | Set Working Directory | To Source File Location

We'll almost always be loading some libraries.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

Stages of a business analytics project

Like most projects, analytical projects are seldom linear. There are numerous feedback loops between the stages and projects tend to be iterative in nature. Just defining the problem can be quite difficult. Nevertheless, it can be helpful to discuss the typical stages. Following the sections of Chapter 1, we'll use an example from an ongoing Kaggle "Getting Started" competition to both illustrate the concepts as well as give you a preview of the wonders of R.

The following diagram is from Chapter 1 of "R for Data Science" by Hadley Wickham. This is a new book that we'll be usign quite a bit. We'll refer to it as r4ds. The online version is free.

Resources on the CRISP-DM framework, a process for data mining projects.

- https://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining
- https://itsalocke.com/crisp-dm/
- https://www.the-modeling-agency.com/crisp-dm.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.198.5133&rep=rep1&type=pdf
- https://www.coursera.org/learn/big-data-machine-learning/lecture/icJH6/crisp-dm

Data Science Workflow

Each data science project is different, but each follows the same general steps. You:

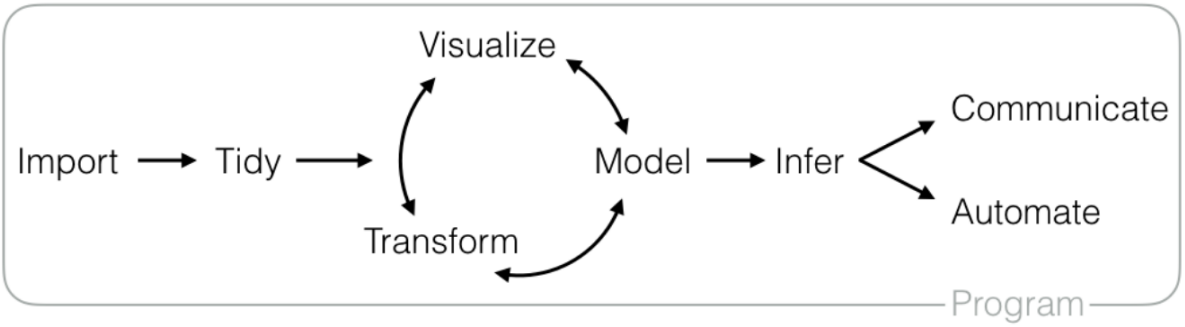


Figure 1: data science workflow

Define the goal

Predicting the selling price of a home based on a number of features of the home makes for a very nice predictive modeling problem. Anyone who has purchased or is thinking of purchasing a home is interested in paying a fair price for the home. In fact, Zillow's home price prediction model has had quite an impact on the real estate industry. Currently, there is an ongoing Kaggle competition to improve upon Zillow's home value model that has a prize of $1,200,000. Yep.

There is also a similar, but simpler, Kaggle competition called House Prices: Advanced Regression Techniques. It's what's known as a Getting Started competition, has no prize money, is worth no ranking points, but is designed for data science beginners to learn to do predictive modeling. Let's go visit the website to get a sense of the problem. We'll start with the Overview section and the Data section. ...

Whoa! That's quite a few fields to deal with. Let's use an even simpler dataset for predicting house prices. The House Sales in King County, USA dataset can be found in the Datsets section of Kaggle. This is one of Kaggle's great features beyond the competitions. People create and share interesting datasets so that others can use them to learn to do data analysis. You can even share your analysis through something called Kernals.

We see that the King County dataset has 19 predictors that we can use to try to predict the selling price. That's our goal - predict selling price. How will we evaluate our predictions? We'll use the same metric as in the similar getting Getting Started competition:

> Submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

Data collection and management

In this case, we are given the dataset with a well defined target column, `price`. Of course, this is totally unrealistic. Instead you'll often deal with spectaculary ill structured text files or massive relational databases with tons of missing values, columns you wish you had and non-existent or poorly written data dictionaries. A relevant target variable may be elusive to define. You'll have to be extremely careful that you aren't accidentally *clairvoyant* or permissive of *future leaking* into your predictors.

You'll spend huge amounts of time wrangling this data into usable shape and then still find there are many

elements important to the problem at hand that you simply don't have. You will be well served by getting good with things like:

- regular expressions
- a good text hacking language like Python, Perl, R, Sed, Awk
- SQL and database knowledge
- web scraping skills
- business domain common sense
- communication and detective skills
- a lot of persistence and patience

Let's read the data in and check it out.

```
kc_house_data <- read.csv("./data/kc_house_data.csv", stringsAsFactors = FALSE)
str(kc_house_data)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : int  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: int  1340 1690 2720 1360 1800 4760 2238 1650 1780 2390 ...
##  $ sqft_lot15   : int  5650 7639 8062 5000 7503 101930 6819 9711 8113 7570 ...
```

A few things to note:

- the variable we are trying to predict is called `price`.
- the `id` column isn't useful for prediction.
- the date is in a string format that we could parse, but we will just ignore for now.
- `waterfront` and `view` are likely (maybe) binary variables where a 1 means yes and 0 no.
- while some of the variables are pretty self explanatory, others (e.g. `condition, grade, sq_living15`) are not.

A big part of most analytics projects involves a bunch of detective work to figure out the meaning and usefulness of the fields in the dataset. For this example, I went into the Discussion forum associated with the dataset and found a link to a nice data dictionary. See https://rstudio-pubs-static.s3.amazonaws.com/155304_cc51f448116744069664b35e7762999f.html.

Drop unneeded columns

To simplify things, let's just use the following fields to start:

- price
- bedrooms
- bathrooms
- sqft_living
- sqft_lot
- floors
- waterfront
- sqft_above
- sqft_basement
- yr_built

This is called "subsetting the data". Since I can easily reread in the original data, I'm going to use the same variable for the reduced data set.

```
cols_to_use <- c("price", "bedrooms", "bathrooms", "sqft_living",
                 "sqft_lot", "floors", "waterfront",
                 "sqft_above", "sqft_basement", "yr_built")

kc_house_data <- kc_house_data[, cols_to_use]
str(kc_house_data)
```

```
## 'data.frame':    21613 obs. of  10 variables:
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
```

Feature engineering

Another extremely important activity that takes place in this stage is called "feature engineering". This is when we create new variables based on existing variables that we believe will be more useful in our predictive models. Again, you must be hypervigilant against accidental clairvoyance in that you don't allow information into your predictor variables that you wouldn't have at the time you are trying to make predictions. A silly example of this for our problem would be inclusion of the amount of the real estate commision.

In this example, we'll create two such new variables.

- age = 2017 - yr_built
- basement = 1 if sqft_basement > 0 and 0 otherwise.

Then we'll use `age` and `basement` instead of `yr_built` and `sqft_basement`.

In Excel, this would play out as new column formulas. In R, we do something like this:

```
kc_house_data$age <- 2017 - kc_house_data$yr_built
kc_house_data$basement <- ifelse(kc_house_data$sqft_basement > 0, 1, 0)
```

Exploratory data analysis

Another common task at this stage is to do some preliminary exploration of your data.

- missing data?
- erroneous data?
- descriptive statistics
- correlations
- random insights

For now, we'll just do a basic summary of the variables and a few basic plots.

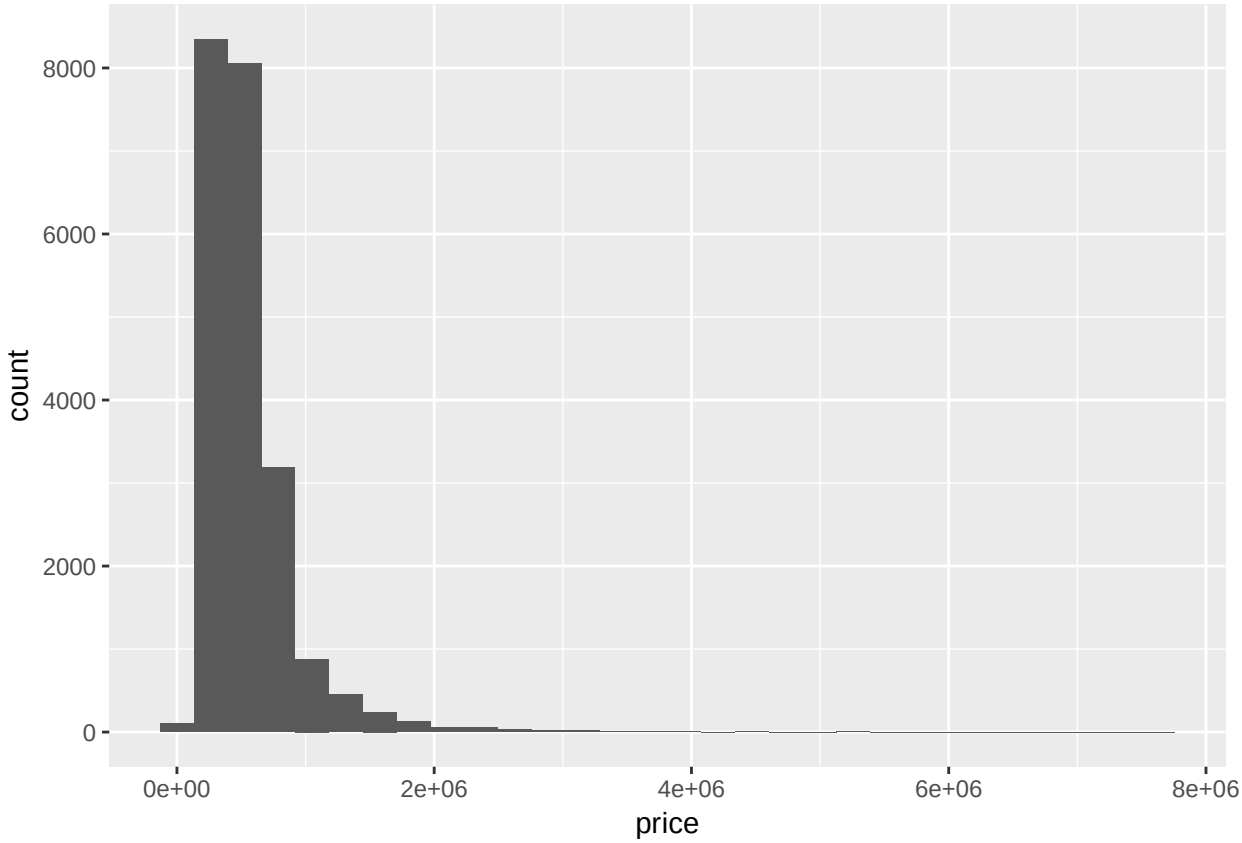
```
summary(kc_house_data)
```

```
##      price            bedrooms        bathrooms      sqft_living
##  Min.   :  75000   Min.   : 0.000   Min.   :0.000   Min.   :  290
##  1st Qu.: 321950   1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1427
##  Median : 450000   Median : 3.000   Median :2.250   Median : 1910
##  Mean   : 540088   Mean   : 3.371   Mean   :2.115   Mean   : 2080
##  3rd Qu.: 645000   3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550
##  Max.   :7700000   Max.   :33.000   Max.   :8.000   Max.   :13540
##     sqft_lot           floors        waterfront         sqft_above
##  Min.   :    520   Min.   :1.000   Min.   :0.000000   Min.   : 290
##  1st Qu.:   5040   1st Qu.:1.000   1st Qu.:0.000000   1st Qu.:1190
##  Median :   7618   Median :1.500   Median :0.000000   Median :1560
##  Mean   :  15107   Mean   :1.494   Mean   :0.007542   Mean   :1788
##  3rd Qu.:  10688   3rd Qu.:2.000   3rd Qu.:0.000000   3rd Qu.:2210
##  Max.   :1651359   Max.   :3.500   Max.   :1.000000   Max.   :9410
##  sqft_basement       yr_built          age            basement
##  Min.   :   0.0   Min.   :1900   Min.   :  2.00   Min.   :0.0000
##  1st Qu.:   0.0   1st Qu.:1951   1st Qu.: 20.00   1st Qu.:0.0000
##  Median :   0.0   Median :1975   Median : 42.00   Median :0.0000
##  Mean   : 291.5   Mean   :1971   Mean   : 45.99   Mean   :0.3927
##  3rd Qu.: 560.0   3rd Qu.:1997   3rd Qu.: 66.00   3rd Qu.:1.0000
##  Max.   :4820.0   Max.   :2015   Max.   :117.00   Max.   :1.0000
```

Here's a histogram of `price`.

```
ggplot(data = kc_house_data) + geom_histogram(aes(x=price))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
library(corrplot)
```

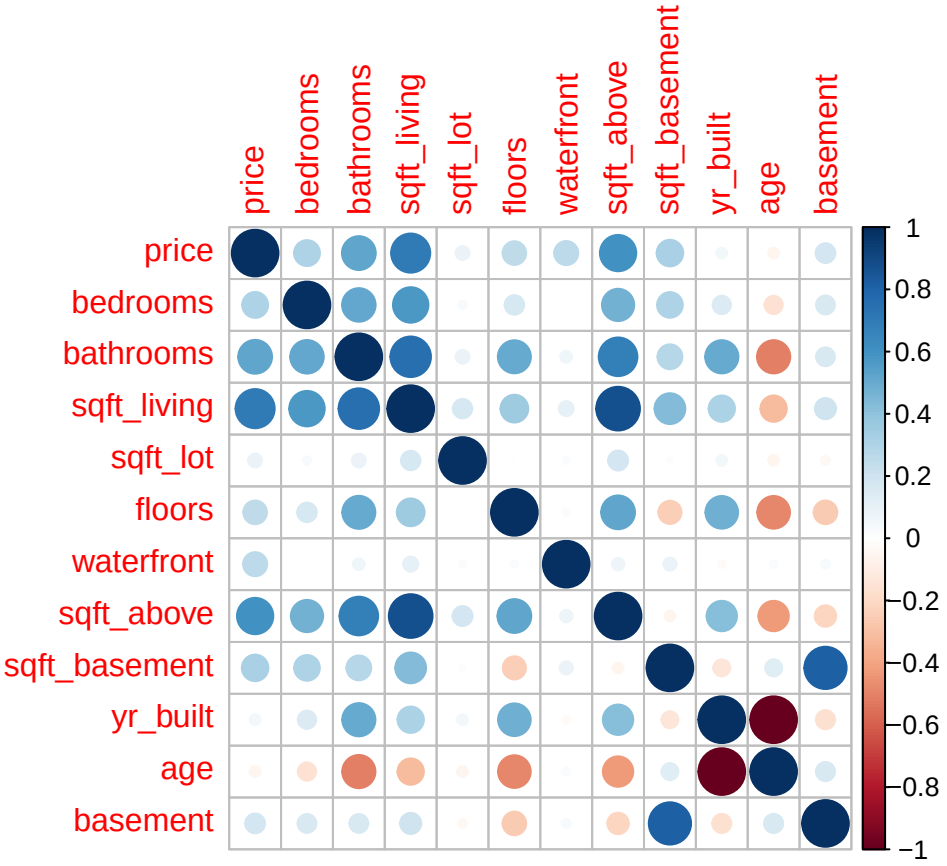
```
## corrplot 0.84 loaded
```

```
cor(kc_house_data)
```

```
##                    price     bedrooms   bathrooms sqft_living
## price          1.00000000  0.308349598  0.52513751   0.7020351
## bedrooms       0.30834960  1.000000000  0.51588364   0.5766707
## bathrooms      0.52513751  0.515883638  1.00000000   0.7546653
## sqft_living    0.70203505  0.576670693  0.75466528   1.0000000
## sqft_lot       0.08966086  0.031703243  0.08773966   0.1728257
## floors         0.25679389  0.175428935  0.50065317   0.3539493
## waterfront     0.26636943 -0.006582479  0.06374363   0.1038178
## sqft_above     0.60556730  0.477600161  0.68534248   0.8765966
## sqft_basement  0.32381602  0.303093375  0.28377003   0.4350430
## yr_built       0.05401153  0.154178069  0.50601944   0.3180488
## age           -0.05401153 -0.154178069 -0.50601944  -0.3180488
## basement       0.18023009  0.163470686  0.16353417   0.2044950
##                   sqft_lot       floors   waterfront  sqft_above
## price          0.089660861  0.256793888  0.266369434  0.60556730
## bedrooms       0.031703243  0.175428935 -0.006582479  0.47760016
## bathrooms      0.087739662  0.500653173  0.063743629  0.68534248
## sqft_living    0.172825661  0.353949290  0.103817818  0.87659660
## sqft_lot       1.000000000 -0.005200991  0.021603683  0.18351228
## floors        -0.005200991  1.000000000  0.023698320  0.52388471
## waterfront     0.021603683  0.023698320  1.000000000  0.07207459
```

```
## sqft_above     0.183512281  0.523884710  0.072074592  1.00000000
## sqft_basement  0.015286202 -0.245704542  0.080587939 -0.05194331
## yr_built       0.053080367  0.489319425 -0.026161086  0.42389835
## age           -0.053080367 -0.489319425  0.026161086 -0.42389835
## basement      -0.035345763 -0.256559908  0.037226752 -0.21099066
##               sqft_basement    yr_built         age    basement
## price            0.32381602  0.05401153 -0.05401153  0.18023009
## bedrooms         0.30309338  0.15417807 -0.15417807  0.16347069
## bathrooms        0.28377003  0.50601944 -0.50601944  0.16353417
## sqft_living      0.43504297  0.31804877 -0.31804877  0.20449501
## sqft_lot         0.01528620  0.05308037 -0.05308037 -0.03534576
## floors          -0.24570454  0.48931942 -0.48931942 -0.25655991
## waterfront       0.08058794 -0.02616109  0.02616109  0.03722675
## sqft_above      -0.05194331  0.42389835 -0.42389835 -0.21099066
## sqft_basement    1.00000000 -0.13312410  0.13312410  0.81915156
## yr_built        -0.13312410  1.00000000 -1.00000000 -0.16790154
## age              0.13312410 -1.00000000  1.00000000  0.16790154
## basement         0.81915156 -0.16790154  0.16790154  1.00000000
```

```
corrplot(corr = cor(kc_house_data), method="circle")
```



Not surprisingly, price is positively correlated with the number of bathrooms and a few variables related to square footage of living space. It's interesting to see that the number of bathrooms is strongly negatively correlated with the age of the house - modern living means more bathrooms. You must be careful to not put too much stock into things like simple pairwise correlations. Two variables can be highly related and yet have a simple correlation coefficient of zero.

Question: Why this is true? If you don't know, it's time to brush up on basic statistics.

Dataset partitioning

Our goal is to use the data we have to build a predictive model that will perform well on **new data that wasn't used to build the model**. I'll often say, and we'll certainly see, that it's pretty easy to build models that **fit** well. It's much harder to build models that **predict** well.

Question: If you do a scatter plot with 10 data points, how could you create a model curve that fits these points perfectly?

One simple way of trying to do this is to partition our data into a *training* set and a *test* set. Use the training set for model building and then compare competing models on their performance on the test data. You'll notice that this is what is done in Kaggle competitions.

Create a simple 80/20 split for fit and test data. We'll learn better ways to do data partitioning later in the class.

```
set.seed(1592) # Make our example reproducible by controlling random number generator
trainrecs <- sample(nrow(kc_house_data),0.8 * nrow(kc_house_data))
kc_house_data_train <- kc_house_data[trainrecs,]
kc_house_data_test <- kc_house_data[-trainrecs,]  # Negative in front of vector means "not in"
```

Modeling

This is where we finally get to build some predictive models. There are a number of different modeling techniques we'll learn about in this class. For now we'll just try two different techniques and just use a few of the predictors.

- multiple linear regression
- regression tree

Linear regression

We'll build one regression model using all of our predictors. In real projects, the issue of *variable selection* can be a tricky one as we may have many potential predictor variables and we have to beware of *overfitting*. This is something we'll discuss quite a bit.

```
lm1 <- lm(price ~ ., data = kc_house_data_train)
summary(lm1)
```

```
##
## Call:
## lm(formula = price ~ ., data = kc_house_data_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1540854  -126080   -14157    99038  3888432
##
## Coefficients: (2 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.214e+06  1.487e+05  41.796  < 2e-16 ***
## bedrooms      -6.350e+04  2.510e+03 -25.303  < 2e-16 ***
## bathrooms      6.578e+04  4.220e+03  15.588  < 2e-16 ***
## sqft_living    2.245e+02  7.877e+00  28.504  < 2e-16 ***
```

```
## sqft_lot      -3.127e-01  4.494e-02  -6.959 3.54e-12 ***
## floors         4.376e+04  4.467e+03   9.795  < 2e-16 ***
## waterfront     7.455e+05  2.096e+04  35.571  < 2e-16 ***
## sqft_above     7.803e+01  8.614e+00   9.058  < 2e-16 ***
## sqft_basement         NA         NA      NA       NA
## yr_built      -3.192e+03  7.700e+01 -41.459  < 2e-16 ***
## age                   NA         NA      NA       NA
## basement       5.240e+04  6.798e+03   7.708 1.35e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 233600 on 17280 degrees of freedom
## Multiple R-squared:  0.5798, Adjusted R-squared:  0.5796
## F-statistic:  2649 on 9 and 17280 DF,  p-value: < 2.2e-16
```

We'll be reviewing what all of that output means, but I'm sure you remember that R^2 (*R-squared*) is a measure of the proportion of variation in the data explained by the model. It ranges from 0 to 1 and is a measure of *fit* of the model to the training data.

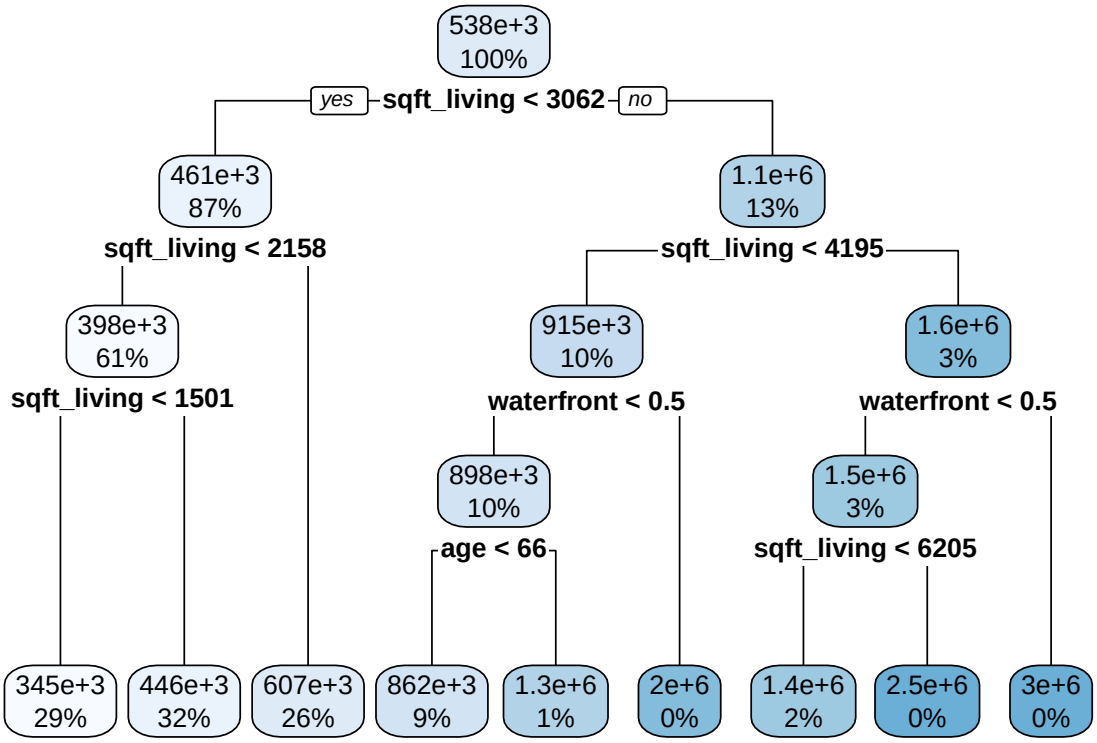
Question: If you add an additional variable to an existing linear regression model, can the R^2 value ever decrease? What about the adjusted R^2 value?

Regression (decision) trees

There are numerous predictive modeling techniques based on the notion of *trees*. Nodes of the trees are the predictor variables and the branches represent splits of the variable based on its value. The picture below will give you a general idea. We'll explore several popular tree based techniques.

```
library(rpart)
library(rpart.plot)
library(RColorBrewer)
```

```
tree1 <- rpart(price ~ ., data = kc_house_data_train, method="anova")
rpart.plot(tree1)
```




```
tree1
```

```
## n= 17290
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 17290 2.243669e+15  538429.8
##    2) sqft_living< 3062 15060 7.029953e+14  460843.3
##      4) sqft_living< 2157.5 10520 2.797209e+14  397758.7
##        8) sqft_living< 1501 5027 9.239571e+13  345473.6 *
##        9) sqft_living>=1501 5493 1.610061e+14  445608.2 *
##      5) sqft_living>=2157.5 4540 2.843969e+14  607021.8 *
##    3) sqft_living>=3062 2230 8.377852e+14 1062399.0
##      6) sqft_living< 4195 1764 3.043327e+14  915349.5
##       12) waterfront< 0.5 1735 2.601527e+14  897512.2
##         24) age< 65.5 1583 1.932294e+14  861976.6 *
##         25) age>=65.5 152 4.410596e+13 1267597.0 *
##       13) waterfront>=0.5 29 1.060165e+13 1982514.0 *
##      7) sqft_living>=4195 466 3.509179e+14 1619042.0
##       14) waterfront< 0.5 439 2.593294e+14 1533415.0
##         28) sqft_living< 6205 404 1.650975e+14 1448051.0 *
##         29) sqft_living>=6205 35 5.730615e+13 2518761.0 *
##       15) waterfront>=0.5 27 3.603539e+13 3011274.0 *
```

Make predictions for the test data

Notice how the R `predict` command is used here for two totally different model types.

Question: Why is that kind of nice?

```
lm1_predict <- predict(lm1, newdata = kc_house_data_test)
```

```
## Warning in predict.lm(lm1, newdata = kc_house_data_test): prediction from a
## rank-deficient fit may be misleading
```

```
head(lm1_predict) # Output of predict is a vector of predictions
```

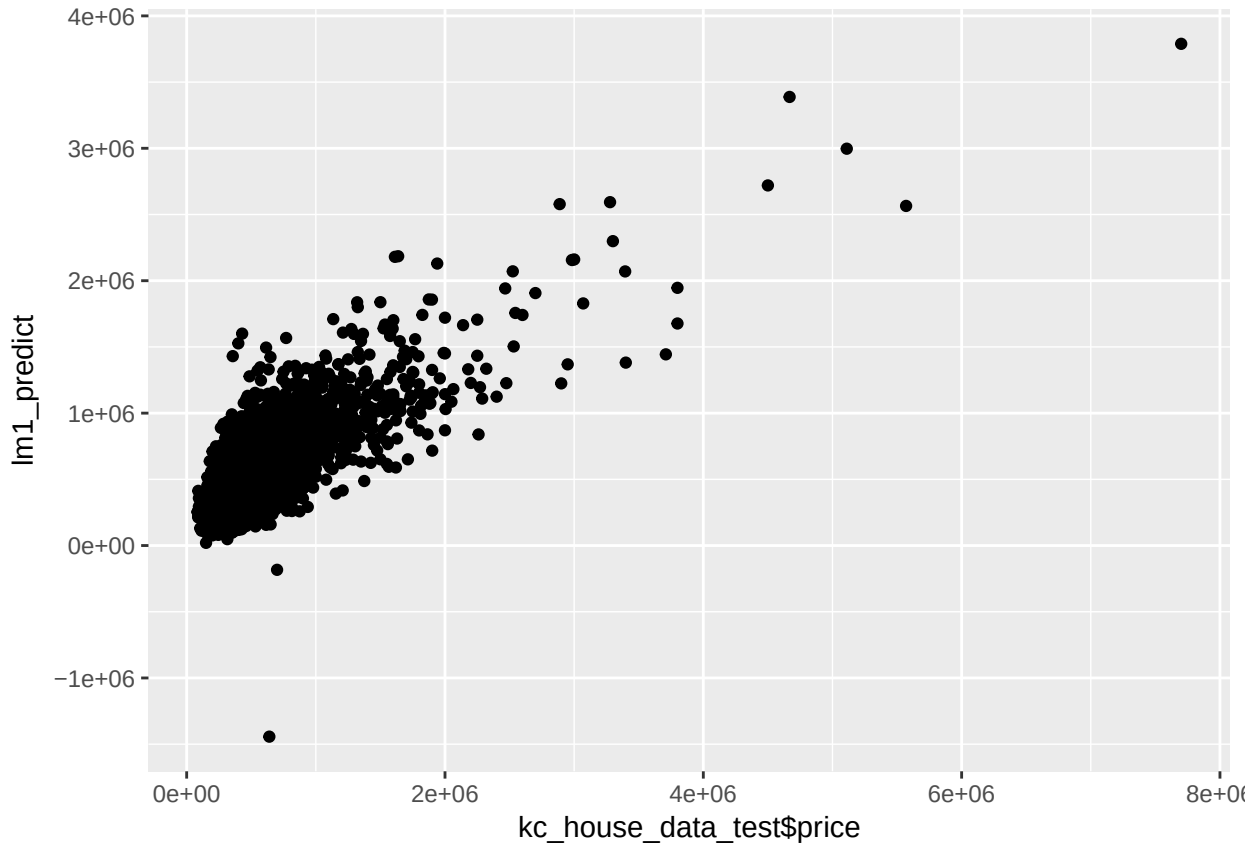
```
##        2        7       19       22       34       51
## 826922.8 406717.8 423715.5 870275.2 325411.7 248846.0
```

```
tree1_predict <- predict(tree1, newdata = kc_house_data_test)
head(tree1_predict) # Output of predict is a vector of predictions
```

```
##        2        7       19       22       34       51
## 607021.8 445608.2 345473.6 607021.8 345473.6 345473.6
```

Scatter plots of actual sales price vs predicted sales price can give a quick visual indication of how well the models fit.

```
ggplot() + geom_point(aes(x=kc_house_data_test$price, y=lm1_predict))
```



Notice that there are two negative predicted values. That can be a problem with simple regression models.

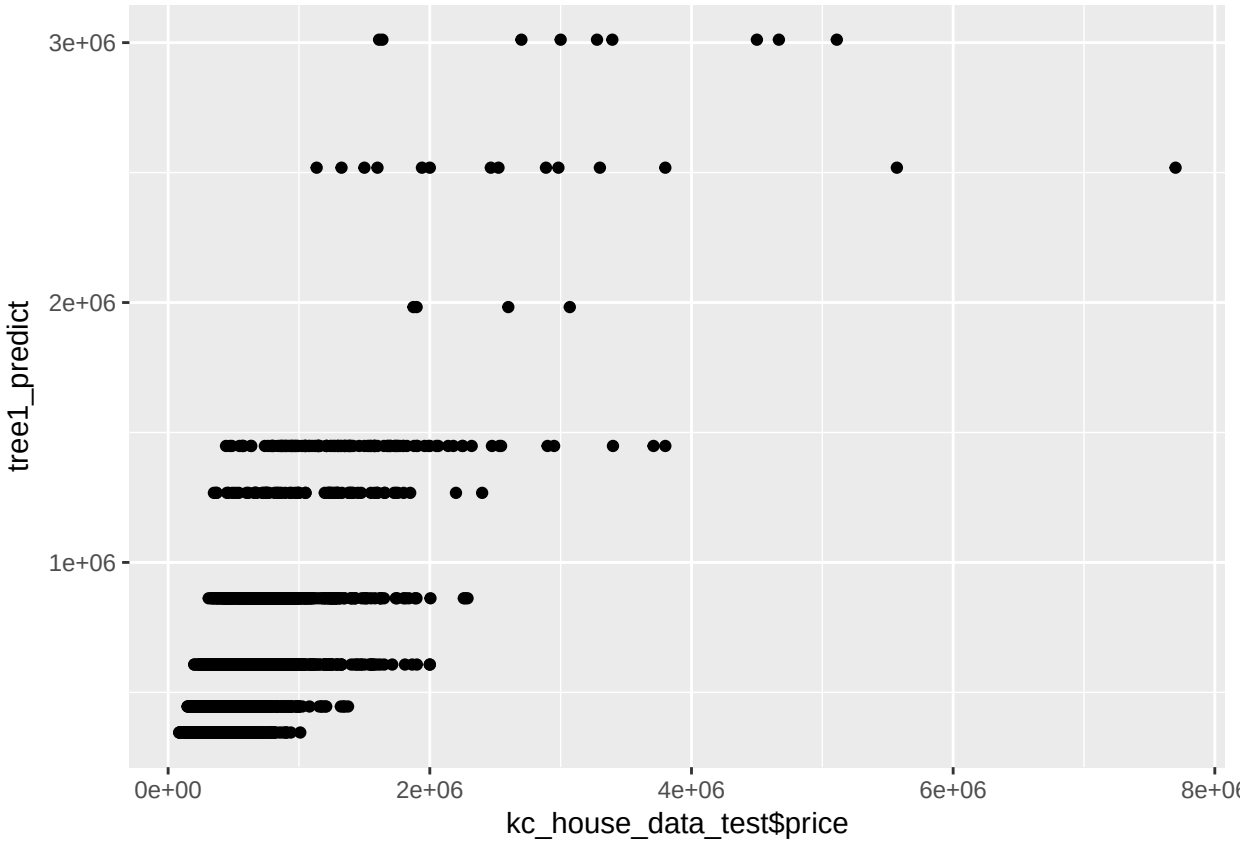
Question: Why would a linear regression model ever return a negative prediction if all of the X and y variables are positive?

We'll just set those predictions to (near) 0.

```
lm1_predict <- ifelse(lm1_predict <= 0, 0.1, lm1_predict)
```

Here's the same scatter plot for the tree model.

```
ggplot() + geom_point(aes(x=kc_house_data_test$price, y=tree1_predict))
```



Question: Can you figure out why the two plots look so different?

Model evaluation

Now let's compute the error metric for the two models. One of the great strengths of R is that we can write our own functions.

```
rms_log_err <- function(actual, predicted){

  logerrsqrd <- (log10(actual) - log10(predicted)) ** 2
  sqrt(mean(logerrsqrd))
}
```

```
err_lm1 <- rms_log_err(kc_house_data_test$price, lm1_predict)
err_tree1 <- rms_log_err(kc_house_data_test$price, tree1_predict)

sprintf("lm1 error: %.3f   tree1 error: %.3f", err_lm1, err_tree1)
```

```
## [1] "lm1 error: 0.223   tree1 error: 0.172"
```

In this case, the tree outperforms the linear regression model on our chosen error metric. The scatter plot for the tree model reveals some perhaps undesirable "banding" of the predictions. Obviously, there is much more we could do in terms of modeling and model evaluation but this small example gives you a sense of the process.

Question: What are some other reasonable error metrics we could use for this problem?

Question: What are some potential issues with the random 80/20 split we made of thie data into training and test sets?

Presentation and documentation

Communicating the results of technical analyses can be a challenge and very audience dependent. The level of technical detail needed by an executive decision maker is likely much different than needed by an IT group charged with deploying such a model in an operational system. No matter who the audience, it's important that results be presented clearly, important assumptions and caveats articulated, and the analysis linked to business objectives.

Don't *oversell* your models! They are just models and are unlikely to capture all the complexities of real problems. They can play an important role in complex decisions but must be complemented with business judgement, common sense, and a good bit of skepticism.

The scatter plots showing actual vs predicted values for the two models are ideal candidates for inclusion in a technical presentation. The correlation matrix presented earlier is not likely a good candidate for a final presentation to decision makers.

Deployment

Moving from offline analysis to a deployable model can be a huge challenge.

- Does model need to be implemented in another language?
- How will model get "fed" its input values?
- How will end users interact with the model?
- How often do model parameters need to be updated?
- Who is responsible for model and data maintenance?

Can you envision various deployment scenarios for the two simple models we fit here?

Next steps

We'll spend the rest of the semester diving into these and related topics along the whole data science workflow pipeline. In addition to using R, we'll also learn the other widely used language for data science - Python. Both of these tools are major players in the data science and business analytics worlds. Learning how to use them will set you apart from the huge number of analysts who can only do Excel and maybe some SQL. And yes, both Excel and SQL are also must have skills for any business analytics person.