Untitled

Joy Payton

12/14/2016

First Tasks: Variable Selection

Pick one of the quanititative independent variables from the training data set (train.csv), and define that variable as X. Make sure this variable is skewed to the right! Pick the dependent variable and define it as Y.

To do this, let's review the metadata and download the data itself to understand which variables are independent and which dependent, and make a variable selection.

Get the metadata

The Kaggle page itself summarizes the contest like this: "With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home." So we know that our dependent variable, or Y, is final home price. And we have a big possible selection of independent variable!

Kaggle supplies us with a mini data dictionary in that same page, which tells us that the home price variable (our Y) has the column name SalePrice.

We'll need to actually get the data and check out the skewness of some of the independent variables in order to select one. Let's do that! Because you have to be logged in to get the data, I'm pulling it from my local copy.

Obtain and Investigate the Data

```
train <- read.csv("Downloads/train.csv")
```

Let's figure out what the skewness is for the independent variables! First, we'll want ones that have a numeric basis, not a categorical one.

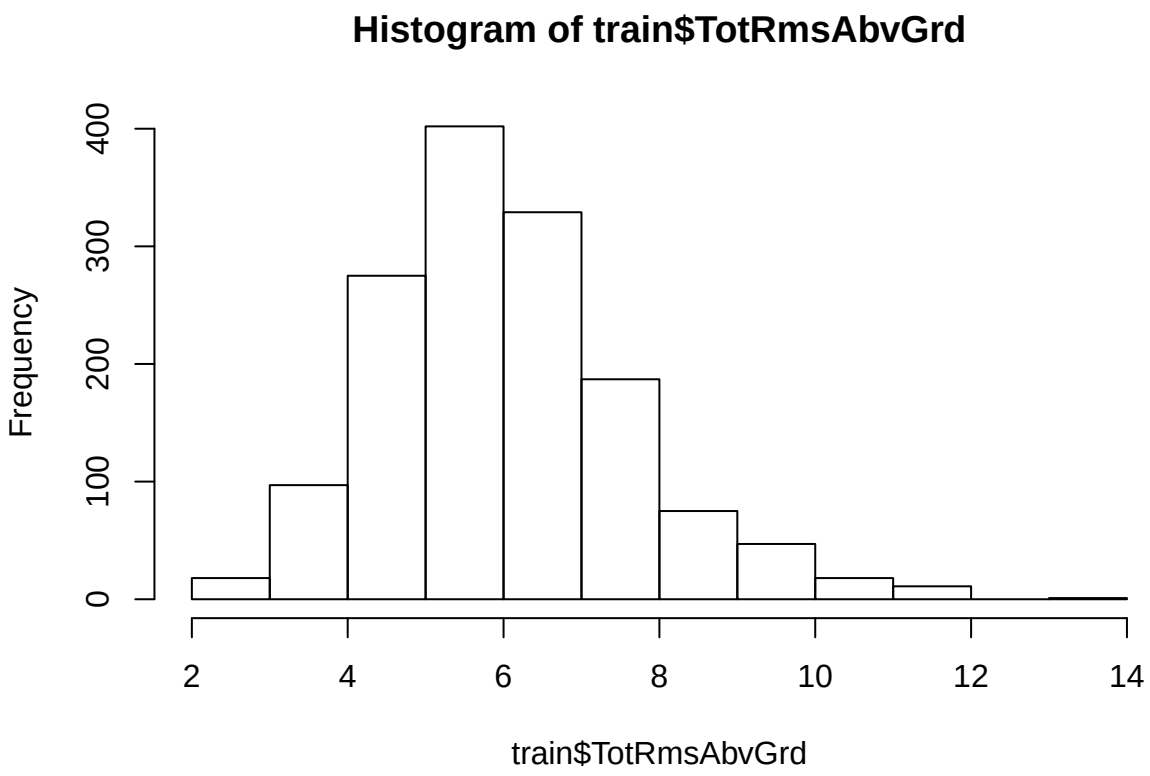
Say, the total number of rooms above ground.

```
library(moments)
skewness(train$TotRmsAbvGrd)
```

```
## [1] 0.6756458
```

The skewness is positive (skewed to the right). Let's look at a histogram!

```
hist(train$TotRmsAbvGrd)
```



Yep, skewed to the right. Let's check out a few more, though!

```
skewness(train$TotalBsmtSF)
```

```
## [1] 1.522688
```

```
skewness(train$FullBath)
```

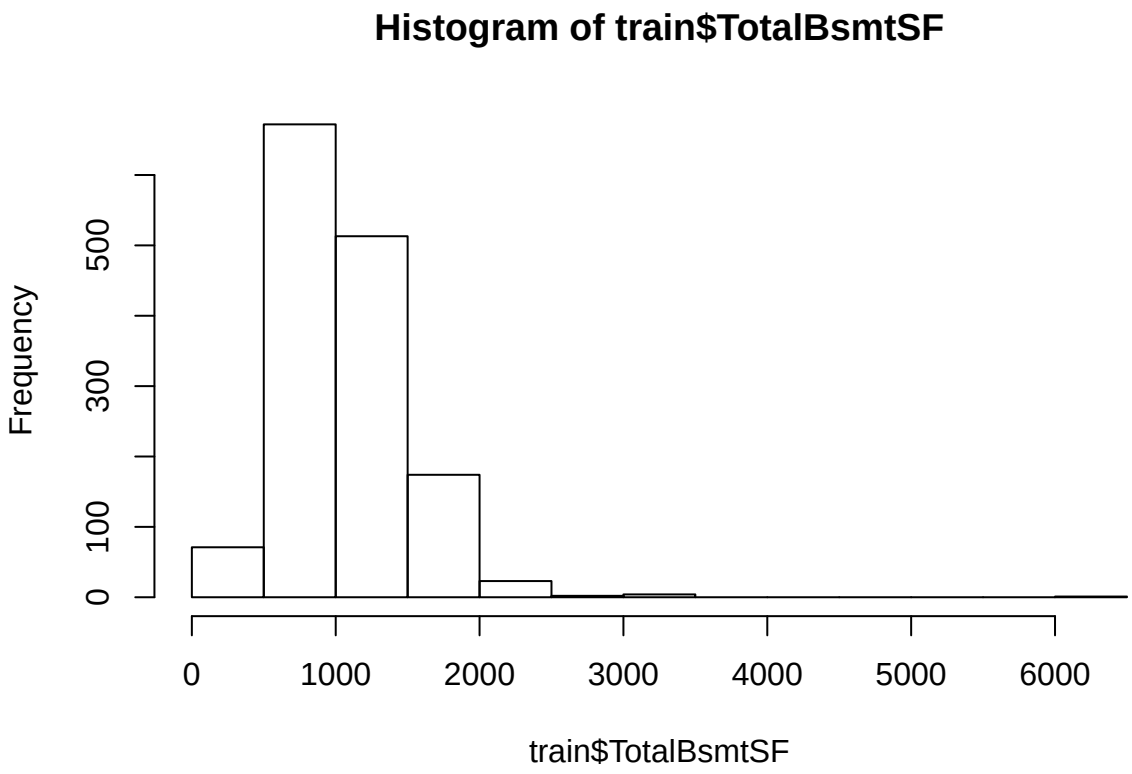
```
## [1] 0.03652398
```

```
skewness(train$GarageCars)
```

```
## [1] -0.3421969
```

Of those three, only the TotalBsmtSF is skewed to the right. It's fairly strong:

```
hist(train$TotalBsmtSF)
```



Still, I kind of like the first one I tried. Total Rooms Above Ground has a fairly consistent tail, while Total Basement Square Footage is skewed because of some outliers. I think I'll get better predictive value out of Total Rooms Above Ground.

Set the variables

```
X <- train$TotRmsAbvGrd
Y <- train$SalePrice
roomsToSalePrice <- data.frame(X=X,Y=Y)
```

Probability Time!

Calculate as a minimum the below probabilities a through c. Assume the small letter "x" is estimated as the 3d quartile of the X variable, and the small letter "y" is estimated as the 2d quartile of the Y variable. Interpret the meaning of all probabilities. In addition, make a table of counts as shown below.

a. P(X>x | Y>y)

b. P(X>x, Y>y)

c. P(Xy)

x/y	<= 2d Quartile	> 2d Quartile	Total
<= 3d Quartile			

x/y	<= 2d Quartile	> 2d Quartile	Total
>3d Quartile			
Total			

Get Quartiles

Let's figure out what x and y are:

```
summary(roomsToSalePrice$X)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   5.000   6.000   6.518   7.000  14.000
```

```
summary(roomsToSalePrice$Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  130000  163000  180900  214000  755000
```

```
x <- summary(roomsToSalePrice$X)[["3rd Qu."]]
y <- summary(roomsToSalePrice$Y)[["Median"]]
```

Figure out Probabilities, Fill in Table

(a) P(X>x | Y>y)

Let's figure out (a) P(X>x | Y>y). We'll want to figure out how many Y > y, and out of those, how many have X > x and how many have X <= x.

```
library(dplyr)
dim(roomsToSalePrice %>% filter(Y>y, X>x))[1]
```

```
## [1] 258
```

```
dim(roomsToSalePrice %>% filter(Y>y, X<=x))[1]
```

```
## [1] 470
```

```
dim(roomsToSalePrice %>% filter(Y>y))[1]
```

```
## [1] 728
```

So, we have the following data for the table:

x/y	<= 2d Quartile	> 2d Quartile	Total
<= 3d Quartile		470	
>3d Quartile		258	
Total		728	

And we can figure out P(X>x | Y>y) (probability that X > 3rd quartile given that Y > 2nd quartile) by taking 258 (X>3rd quartile and Y>2nd quartile) divided by 728 (all Y > 2nd quartile):

```
258/728
```

```
## [1] 0.3543956
```

P(X>x | Y>y) = 0.3544. There is a 35.44% chance that X > 3rd quartile, given that Y > 2nd quartile.

(b) P(X>x, Y>y)

Let's figure out (b) P(X>x, Y>y). We'll want to figure out how many cases we have, and out of those, how many have both X > x and Y > y.

```
dim(roomsToSalePrice)[1]
```

```
## [1] 1460
```

```
dim(roomsToSalePrice %>% filter(Y>y, X>x))[1]
```

```
## [1] 258
```

So, we have the following data for the table:

x/y	<= 2d Quartile	> 2d Quartile	Total
<= 3d Quartile		470	
>3d Quartile		258	
Total		728	1460

And we can figure out P(X>x,Y>y) (probability that X > 3rd quartile and Y > 2nd quartile) by taking 258 (X>3rd quartile and Y>2nd quartile) divided by 1460 (all cases):

```
258/1460
```

```
## [1] 0.1767123
```

P(X>x, Y>y) = 0.1767. There is a 17.67% chance that X > 3rd quartile and Y > 2nd quartile.

(c) P(X<x | Y>y)

Let's figure out (c) P(X<x | Y>y)
We'll want to figure out how many Y > y, and out of those, how many have X < x and how many have X >= x. This will answer (c). However, to fill in the table, we'll want to swap the case where X=x! We've already done this in step (a), so we're good with the table. We'll just figure out the probability.

```
library(dplyr)
dim(roomsToSalePrice %>% filter(Y>y, X<x))[1]
```

```
## [1] 243
```

```
dim(roomsToSalePrice %>% filter(Y>y, X>=x))[1]
```

```
## [1] 485
```

```
dim(roomsToSalePrice %>% filter(Y>y))[1]
```

```
## [1] 728
```

And we can figure out P(X<x | Y>y) (probability that X < 3rd quartile given that Y > 2nd quartile) by taking 243 (X>3rd quartile and Y>2nd quartile) divided by 728 (all Y > 2nd quartile):

```
243/728
```

```
## [1] 0.3337912
```

P(X<x | Y>y) = 0.3338. There is a 33.38% chance that X < 3rd quartile, given that Y > 2nd quartile.

Finish the table

Currently, we have the following data for the table:

x/y	<= 2d Quartile	> 2d Quartile	Total
<= 3d Quartile		470	
>3d Quartile		258	
Total		728	1460

We need to figure out:

- The number where y<= 2d quartile, in all cases of X
- The totals for X <= 3rd quartile and >3rd quartile

```
dim(roomsToSalePrice %>% filter(Y<=y, X>x))[1]
```

```
## [1] 81
```

```
dim(roomsToSalePrice %>% filter(Y<=y, X<=x))[1]
```

```
## [1] 651
```

```
dim(roomsToSalePrice %>% filter(Y<=y))[1]
```

```
## [1] 732
```

```
dim(roomsToSalePrice %>% filter(X<=x))[1]
```

```
## [1] 1121
```

```
dim(roomsToSalePrice %>% filter(X>x))[1]
```

```
## [1] 339
```

This gives us the following table:

x/y	<= 2d Quartile	> 2d Quartile	Total
<= 3d Quartile	651	470	1121
>3d Quartile	81	258	339
Total	732	728	1460

Do all our calculations line up?

```
651+470 == 1121
```

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

```
81+258 == 339
```

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

```
732+728 == 1460
```

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

```
651+81 == 732
```

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

```
470+258 == 728
```

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

```
1121+339 == 1460
```

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

OK, so the math works out. Our marginal counts are the sum of our conditional counts.

Independence

Does splitting the training data in this fashion make them independent? Let A be the new variable counting those observations above the 3d quartile for X, and let B be the new variable counting those observations above the 2d quartile for Y. Does P(A|B)=P(A)P(B)? Check mathematically, and then evaluate by running a Chi Square test for association.

Set variables as functions

```
A <- function(df) {
  return(df %>% filter(X>x))
}

B <- function(df) {
  return(df %>% filter(Y>y))
}
```

Now we can easily count and get probability for P(A):

```
dim(A(roomsToSalePrice))[1]
```

```
## [1] 339
```

```
dim(A(roomsToSalePrice))[1]/dim(roomsToSalePrice)[1]
```

```
## [1] 0.2321918
```

For P(B):

```
dim(B(roomsToSalePrice))[1]
```

```
## [1] 728
```

```
dim(B(roomsToSalePrice))[1]/dim(roomsToSalePrice)[1]
```

```
## [1] 0.4986301
```

For P(A)P(B):

```
dim(A(roomsToSalePrice))[1]/dim(roomsToSalePrice)[1] *
  dim(B(roomsToSalePrice))[1]/dim(roomsToSalePrice)[1]
```

```
## [1] 0.1157778
```

For P(A|B):

```
dim(A(B(roomsToSalePrice)))[1]/dim(roomsToSalePrice)[1]
```

```
## [1] 0.1767123
```

P(A)P(B) is *not* the same as P(A|B). These variables are not independent! Let's confirm via a Chi-Squared test, which will give the probability of independence.

```
chisq.test(roomsToSalePrice$X, roomsToSalePrice$Y)
```

```
##
##  Pearson's Chi-squared test
##
## data:  roomsToSalePrice$X and roomsToSalePrice$Y
## X-squared = 8753, df = 7282, p-value < 2.2e-16
```

Wow, that's a low p-value. We can reject the null hypothesis of independence. These varibles are not independent!

Descriptive and Inferential Statistics.

Provide univariate descriptive statistics and appropriate plots for the training data set. Provide a scatterplot of X and Y. Provide a 95% CI for the difference in the mean of the variables. Derive a correlation matrix for two of the quantitative variables you selected. Test the hypothesis that the correlation between these variables is 0 and provide a 99% confidence interval. Discuss the meaning of your analysis.

Univariate Descriptive Statistics

Let's get some idea of what X and Y look like numerically:

```
summary(roomsToSalePrice$X)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   5.000   6.000   6.518   7.000  14.000
```

```
summary(roomsToSalePrice$Y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  130000  163000  180900  214000  755000
```

```
sd(roomsToSalePrice$X)
```

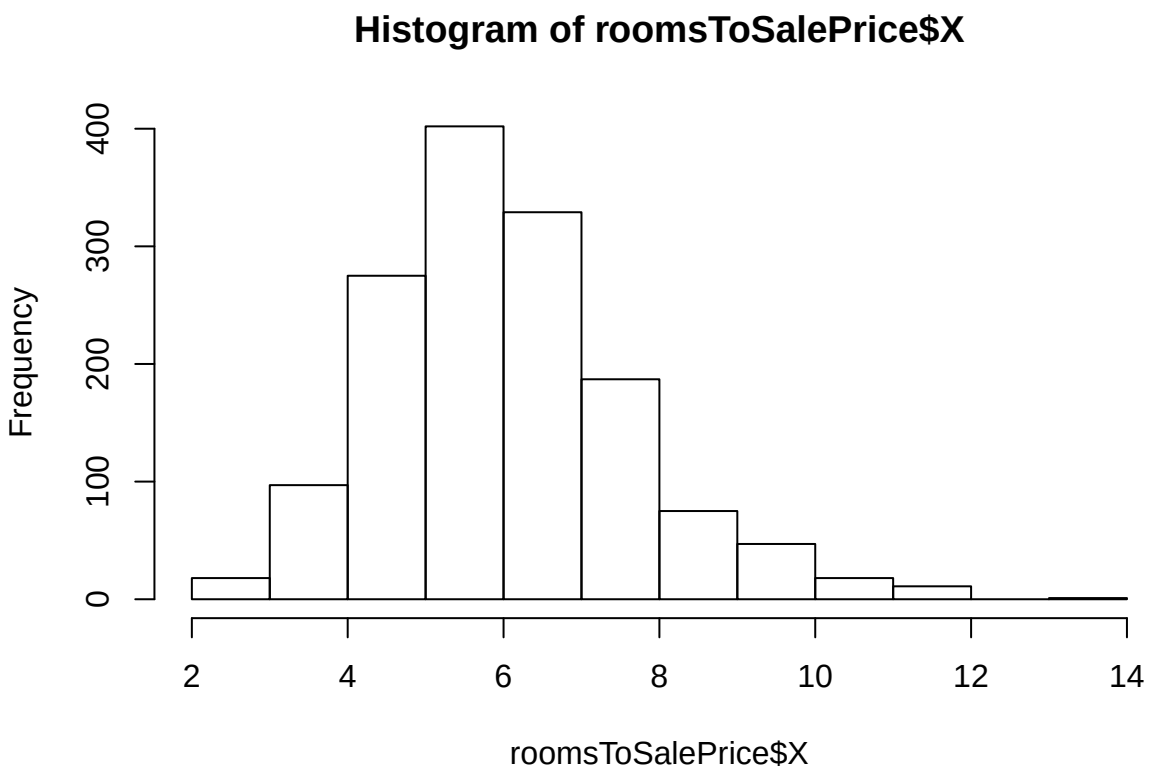
```
## [1] 1.625393
```

```
sd(roomsToSalePrice$Y)
```

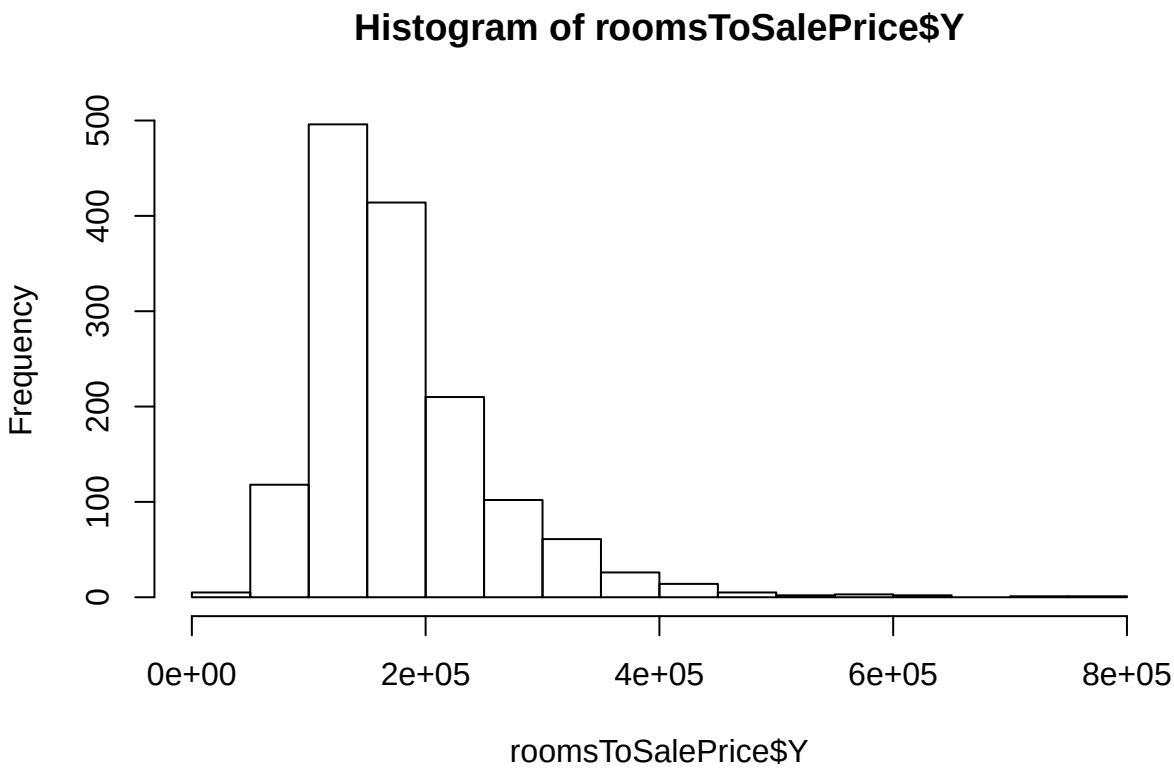
```
## [1] 79442.5
```

And let's see their distributions:

```
hist(roomsToSalePrice$X)
```



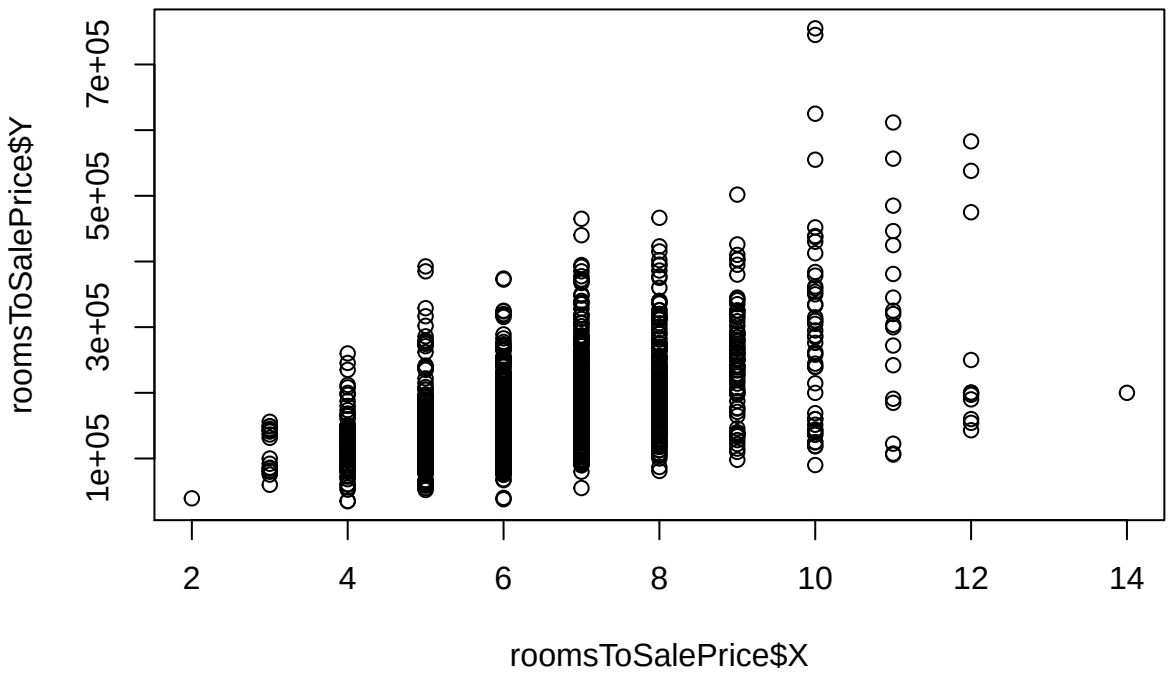

```
hist(roomsToSalePrice$Y)
```



Both are right-skewed. X has a mean of 6.518 and standard deviation of 1.625. Y has a mean of 180900 and a standard deviation of 79442.5.

Let's see their relationship!

```
plot(roomsToSalePrice$X, roomsToSalePrice$Y)
```



Even this ugly exploratory plot seems to show a positive correlation! Let's see if there's significance, if I look at the difference in Y mean between the cases where X > 3d quartile and where X <= 3rd quartile.

```
t.test(roomsToSalePrice$Y ~ as.factor(roomsToSalePrice$X > x))
```

```
##
##  Welch Two Sample t-test
##
## data:  roomsToSalePrice$Y by as.factor(roomsToSalePrice$X > x)
## t = -13.227, df = 404.74, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -90434.09 -67030.40
## sample estimates:
## mean in group FALSE  mean in group TRUE
##            162640.2            241372.5
```

Well, the sample estimates are quite different! Where X>x, the mean of Y is considerably higher. Is it statistically significant? Yes, the p value is very low. And the 95% confidence interval for the difference in mean from X > x and X <= x is (-90434, -67030). That means that when comparing homes where the number of rooms above ground is less than or equal to the 3rd quartile to homes where the number of rooms above ground is higher than the 3rd quartile, I can expect a large difference in home sale price: homes with fewer rooms (<= 3rd quartile) will sell for $67k - $90k less than homes with more rooms (above the 3rd quartile).

Let's see a correlation matrix:

```
cor(roomsToSalePrice)
```

```
##           X         Y
## X 1.0000000 0.5337232
## Y 0.5337232 1.0000000
```

Obviously our diagonal has 1.0 as the value, as each variable is perfectly correlated with itself. What interests us is the 0.5337 value, which shows a relatively strong correlation between X and Y. Is that value statistically significant, however? Let's do a correlation test to get the p value:

```
cor.test(roomsToSalePrice$X, roomsToSalePrice$Y, conf.level = 0.99)
```

```
##
##  Pearson's product-moment correlation
##
## data:  roomsToSalePrice$X and roomsToSalePrice$Y
## t = 24.099, df = 1458, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 99 percent confidence interval:
##  0.4837398 0.5802363
## sample estimates:
##       cor
## 0.5337232
```

That's a tiny p-value, as low as R can go on my computer. So we can be confident that there really is a positive correlation. My 99% confidence interval is (0.4837, 0.5802).

Overall, this analysis of the relationship between my X and Y, graphically, in terms of correlation and correlation significance, and in terms of mean difference in Y given different X categories, gives me confidence that there is a relationship between these variables. Changes in X result in changes in Y. Specifically, as X increases, so does Y, and these differences are extremely unlikely to be related to random chance (extremely low p-values).

But a correlation of just two variables doesn't tell us much! Let's grab some additional numerical variables and throw them into the mix!

```
multiToSalePrice <- train %>% select(TotRmsAbvGrd, LotArea, OverallQual, WoodDeckSF,SalePrice)
cor(multiToSalePrice)
```

```
##              TotRmsAbvGrd   LotArea OverallQual WoodDeckSF SalePrice
## TotRmsAbvGrd    1.0000000 0.1900148   0.4274523  0.1659839 0.5337232
## LotArea         0.1900148 1.0000000   0.1058057  0.1716977 0.2638434
## OverallQual     0.4274523 0.1058057   1.0000000  0.2389234 0.7909816
## WoodDeckSF      0.1659839 0.1716977   0.2389234  1.0000000 0.3244134
## SalePrice       0.5337232 0.2638434   0.7909816  0.3244134 1.0000000
```

Let's consider something with low correlation: Lot Area and Overall Quality. Its correlation is 0.1058. Is it significant?

```
cor.test(multiToSalePrice$LotArea, multiToSalePrice$OverallQual)
```

```
##
##  Pearson's product-moment correlation
##
## data:  multiToSalePrice$LotArea and multiToSalePrice$OverallQual
## t = 4.0629, df = 1458, p-value = 5.106e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.0548009 0.1562599
```

```
## sample estimates:
##       cor
## 0.1058057
```

It is significant, but the 95% confidence interval takes us close to 0. There is a real, but slight, correlation.

What about a stronger correlation, that, say, of Total Rooms Above Ground and Overall Quality?

```
cor.test(multiToSalePrice$TotRmsAbvGrd, multiToSalePrice$OverallQual)
```

```
##
##  Pearson's product-moment correlation
##
## data:  multiToSalePrice$TotRmsAbvGrd and multiToSalePrice$OverallQual
## t = 18.054, df = 1458, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3845837 0.4684812
## sample estimates:
##       cor
## 0.4274523
```

Also significant, but with a 95% confidence interval much further from 0. This is a moderate to strong correlation: the more rooms above ground you have, the more likely you are to have a higher overall quality home.

Linear Algebra and Correlation

Invert your correlation matrix. (This is known as the precision matrix and contains variance inflation factors on the diagonal.) Multiply the correlation matrix by the precision matrix, and then multiply the precision matrix by the correlation matrix. Conduct principle components analysis (research this!) and interpret. Discuss.

```
corMatrix <- cor(multiToSalePrice)
precMatrix <- solve(corMatrix)
# we round to correct for precision errors
round(corMatrix %*% precMatrix,2)
```

```
##              TotRmsAbvGrd LotArea OverallQual WoodDeckSF SalePrice
## TotRmsAbvGrd            1       0           0          0         0
## LotArea                 0       1           0          0         0
## OverallQual             0       0           1          0         0
## WoodDeckSF              0       0           0          1         0
## SalePrice               0       0           0          0         1
```

```
round(precMatrix %*% corMatrix,2)
```

```
##              TotRmsAbvGrd LotArea OverallQual WoodDeckSF SalePrice
## TotRmsAbvGrd            1       0           0          0         0
## LotArea                 0       1           0          0         0
## OverallQual             0       0           1          0         0
## WoodDeckSF              0       0           0          1         0
## SalePrice               0       0           0          0         1
```

Multiplication in either direction gives the identity matrix (makes sense, because it's the inverse!)

PCA attempts to reduce the dimensionality of complex data by discarding data with low variance, and combining highly correlated features into a single "principal component". The cluster of correlated features with maximum variance is the first principal component and accounts for the most variation in the outcome, the second principal component accounts for the second-highest variation explanation, etc.

Since the scale of our variables is wide (some values are single digit, others are in units of hundreds or thousands), we should scale (and center) our variables.

Let's get all of the training set's numerical data and center and scale those numbers by getting their Z statistic!

```
numeric <-  sapply(train, is.numeric)
numericalTrain <- train[,numeric]
```

Let's remove ID, shall we? As well as the outcome variable!

```
numericalTrain <- numericalTrain %>% select (-c(Id, SalePrice))
```

Let's check for missing data:

```
summary(numericalTrain)
```

```
##    MSSubClass     LotFrontage        LotArea        OverallQual
##  Min.   : 20.0   Min.   : 21.00   Min.   :  1300   Min.   : 1.000
##  1st Qu.: 20.0   1st Qu.: 59.00   1st Qu.:  7554   1st Qu.: 5.000
##  Median : 50.0   Median : 69.00   Median :  9478   Median : 6.000
##  Mean   : 56.9   Mean   : 70.05   Mean   : 10517   Mean   : 6.099
##  3rd Qu.: 70.0   3rd Qu.: 80.00   3rd Qu.: 11602   3rd Qu.: 7.000
##  Max.   :190.0   Max.   :313.00   Max.   :215245   Max.   :10.000
##                  NA's   :259
##   OverallCond      YearBuilt     YearRemodAdd    MasVnrArea
##  Min.   :1.000   Min.   :1872   Min.   :1950   Min.   :   0.0
##  1st Qu.:5.000   1st Qu.:1954   1st Qu.:1967   1st Qu.:   0.0
##  Median :5.000   Median :1973   Median :1994   Median :   0.0
##  Mean   :5.575   Mean   :1971   Mean   :1985   Mean   : 103.7
##  3rd Qu.:6.000   3rd Qu.:2000   3rd Qu.:2004   3rd Qu.: 166.0
##  Max.   :9.000   Max.   :2010   Max.   :2010   Max.   :1600.0
##                                                NA's   :8
##    BsmtFinSF1       BsmtFinSF2        BsmtUnfSF       TotalBsmtSF
##  Min.   :   0.0   Min.   :   0.00   Min.   :   0.0   Min.   :   0.0
##  1st Qu.:   0.0   1st Qu.:   0.00   1st Qu.: 223.0   1st Qu.: 795.8
##  Median : 383.5   Median :   0.00   Median : 477.5   Median : 991.5
##  Mean   : 443.6   Mean   :  46.55   Mean   : 567.2   Mean   :1057.4
##  3rd Qu.: 712.2   3rd Qu.:   0.00   3rd Qu.: 808.0   3rd Qu.:1298.2
##  Max.   :5644.0   Max.   :1474.00   Max.   :2336.0   Max.   :6110.0
##
##    X1stFlrSF      X2ndFlrSF     LowQualFinSF       GrLivArea
##  Min.   : 334   Min.   :   0   Min.   :  0.000   Min.   : 334
##  1st Qu.: 882   1st Qu.:   0   1st Qu.:  0.000   1st Qu.:1130
##  Median :1087   Median :   0   Median :  0.000   Median :1464
##  Mean   :1163   Mean   : 347   Mean   :  5.845   Mean   :1515
##  3rd Qu.:1391   3rd Qu.: 728   3rd Qu.:  0.000   3rd Qu.:1777
##  Max.   :4692   Max.   :2065   Max.   :572.000   Max.   :5642
##
##   BsmtFullBath     BsmtHalfBath        FullBath        HalfBath
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:1.000   1st Qu.:0.0000
```

```
##  Median :0.0000   Median :0.00000   Median :2.000   Median :0.0000
##  Mean   :0.4253   Mean   :0.05753   Mean   :1.565   Mean   :0.3829
##  3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:2.000   3rd Qu.:1.0000
##  Max.   :3.0000   Max.   :2.00000   Max.   :3.000   Max.   :2.0000
##
##   BedroomAbvGr    KitchenAbvGr    TotRmsAbvGrd      Fireplaces
##  Min.   :0.000   Min.   :0.000   Min.   : 2.000   Min.   :0.000
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.: 5.000   1st Qu.:0.000
##  Median :3.000   Median :1.000   Median : 6.000   Median :1.000
##  Mean   :2.866   Mean   :1.047   Mean   : 6.518   Mean   :0.613
##  3rd Qu.:3.000   3rd Qu.:1.000   3rd Qu.: 7.000   3rd Qu.:1.000
##  Max.   :8.000   Max.   :3.000   Max.   :14.000   Max.   :3.000
##
##   GarageYrBlt     GarageCars      GarageArea       WoodDeckSF
##  Min.   :1900   Min.   :0.000   Min.   :   0.0   Min.   :  0.00
##  1st Qu.:1961   1st Qu.:1.000   1st Qu.: 334.5   1st Qu.:  0.00
##  Median :1980   Median :2.000   Median : 480.0   Median :  0.00
##  Mean   :1979   Mean   :1.767   Mean   : 473.0   Mean   : 94.24
##  3rd Qu.:2002   3rd Qu.:2.000   3rd Qu.: 576.0   3rd Qu.:168.00
##  Max.   :2010   Max.   :4.000   Max.   :1418.0   Max.   :857.00
##  NA's   :81
##   OpenPorchSF     EnclosedPorch      X3SsnPorch      ScreenPorch
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00
##  Median : 25.00   Median :  0.00   Median :  0.00   Median :  0.00
##  Mean   : 46.66   Mean   : 21.95   Mean   :  3.41   Mean   : 15.06
##  3rd Qu.: 68.00   3rd Qu.:  0.00   3rd Qu.:  0.00   3rd Qu.:  0.00
##  Max.   :547.00   Max.   :552.00   Max.   :508.00   Max.   :480.00
##
##     PoolArea          MiscVal             MoSold           YrSold
##  Min.   :  0.000   Min.   :    0.00   Min.   : 1.000   Min.   :2006
##  1st Qu.:  0.000   1st Qu.:    0.00   1st Qu.: 5.000   1st Qu.:2007
##  Median :  0.000   Median :    0.00   Median : 6.000   Median :2008
##  Mean   :  2.759   Mean   :   43.49   Mean   : 6.322   Mean   :2008
##  3rd Qu.:  0.000   3rd Qu.:    0.00   3rd Qu.: 8.000   3rd Qu.:2009
##  Max.   :738.000   Max.   :15500.00   Max.   :12.000   Max.   :2010
##
```

These variables have NA's:

- LotFrontage (259)
- MasVnrArea (8)
- GarageYrBlt (81)

It's only three variables out of 36. Let's just remove them – it's easier!

```
numericalTrain <- numericalTrain %>% select(-c(LotFrontage,
                                                MasVnrArea,
                                                GarageYrBlt))
```

And let's apply PCA with centering and scaling.

```
principalComponents <- prcomp(numericalTrain,
                              center = TRUE,
                              scale = TRUE)
```

We have a much larger number of principal components than we'll need. Let's plot their relative influence and

see how many we want to stick with for our model. We'll calculate the variance for each principal component, and then figure out how much of the total variance explained each principal component supplies.

```
pcaVariance <- principalComponents$sdev^2
head(pcaVariance)
```

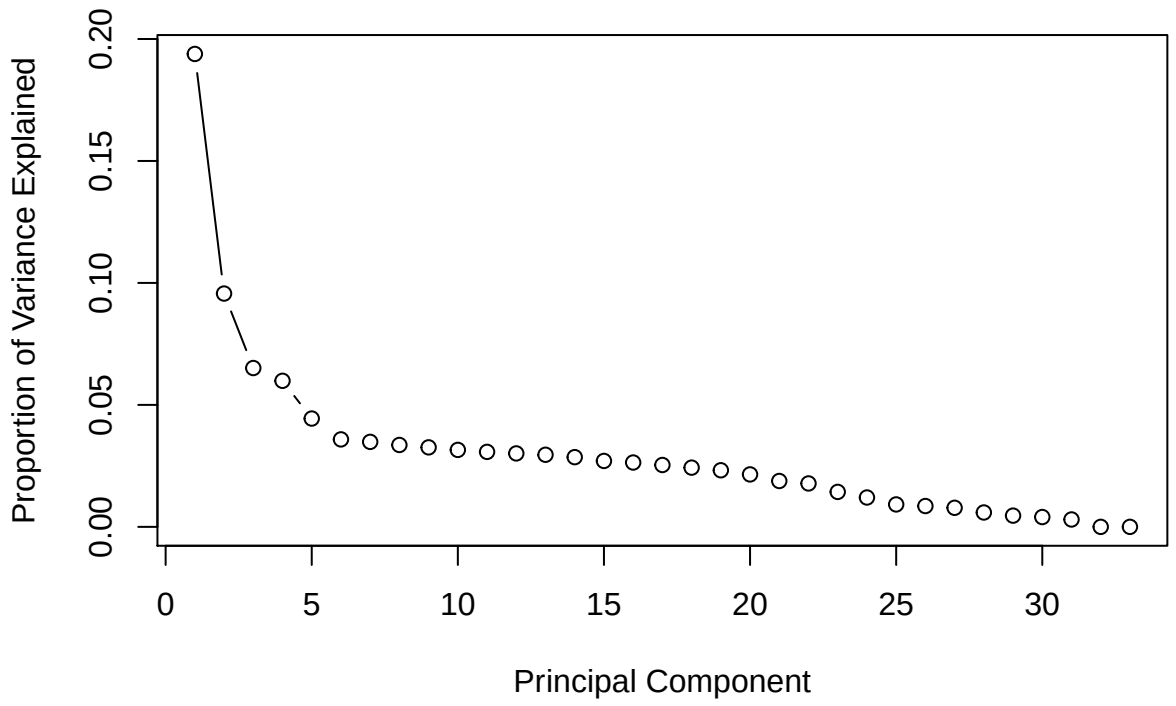
```
## [1] 6.397591 3.156201 2.148440 1.975762 1.465261 1.183036
```

```
propVariance <- pcaVariance / sum(pcaVariance)
head(propVariance)
```

```
## [1] 0.19386639 0.09564245 0.06510423 0.05987158 0.04440183 0.03584957
```

So our first principal component accounts for 19.4% of variance, the second accounts for 9.6%, the third 6.5%, etc. Do we experience a telling dropoff of when additional principal components don't add much?

```
plot(propVariance, xlab = "Principal Component",
             ylab = "Proportion of Variance Explained",
             type = "b")
```



Looks like we can get a lot of variance explained by just the first 5 principal components. After that, we have diminishing returns – the improvement for each additional principal component isn't that great. So let's look at our first five principal components:

```
head(principalComponents$x[,1:5])
```

```
##             PC1        PC2         PC3        PC4         PC5
## [1,]  1.2235030  0.5904077 -0.68634187  2.3416164 -0.71344628
## [2,]  0.0214895 -1.2072514  0.97801777 -0.2740593  1.87060682
## [3,]  1.4673109  0.4103098 -0.79683808  1.7862875 -0.03947134
```

```
## [4,] -0.3130099  1.0122778  1.40095222 -0.3792061 -0.30678515
## [5,]  4.1058373  1.0503830 -0.07746432  1.5323302 -0.03271351
## [6,] -0.9656718 -1.6828408 -1.03722699  2.0850835  0.55207936
```

These five "variables", which consist of weighted combinations of all 33 variables, allow us to do prediction on a much smaller data set. Instead of accounting for 33 variables, we now only have to work with five!

Calculus-Based Probability & Statistics

Many times, it makes sense to fit a closed form distribution to data. For your variable that is skewed to the right, shift it so that the minimum value is above zero. Then load the MASS package and run fitdistr to fit an exponential probability density function. (See https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/fitdistr.html). Find the optimal value of lambda for this distribution, and then take 1000 samples from this exponential distribution using this value (e.g., rexp(1000, lambda)). Plot a histogram and compare it with a histogram of your original variable. Using the exponential pdf, find the 5th and 95th percentiles using the cumulative distribution function (CDF). Also generate a 95% confidence interval from the empirical data, assuming normality. Finally, provide the empirical 5th percentile and 95th percentile of the data. Discuss.

So, that skewed-right variable should only be positive. Let's check!

```
summary(train$TotRmsAbvGrd)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   5.000   6.000   6.518   7.000  14.000
```

OK, let's continue and create our exponential distribution.

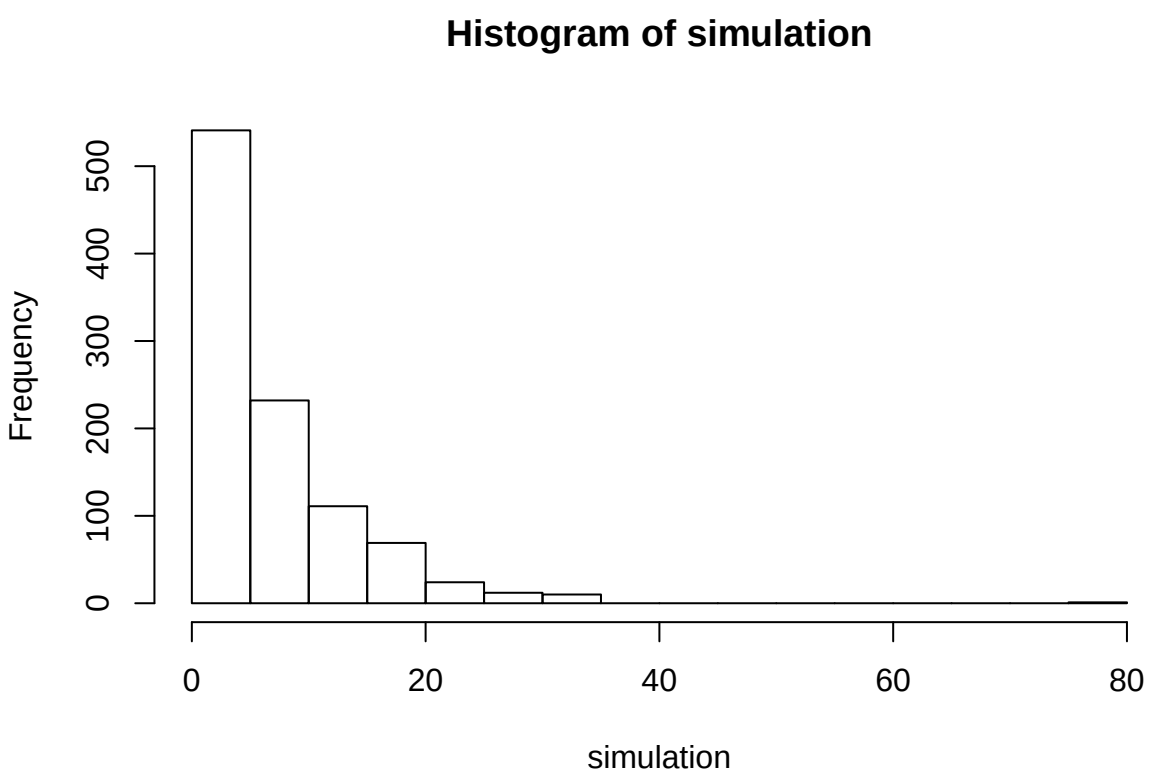
```
library(MASS)
dist <- fitdistr(train$TotRmsAbvGrd, "exponential")
dist$estimate
```

```
##      rate
## 0.1534258
```

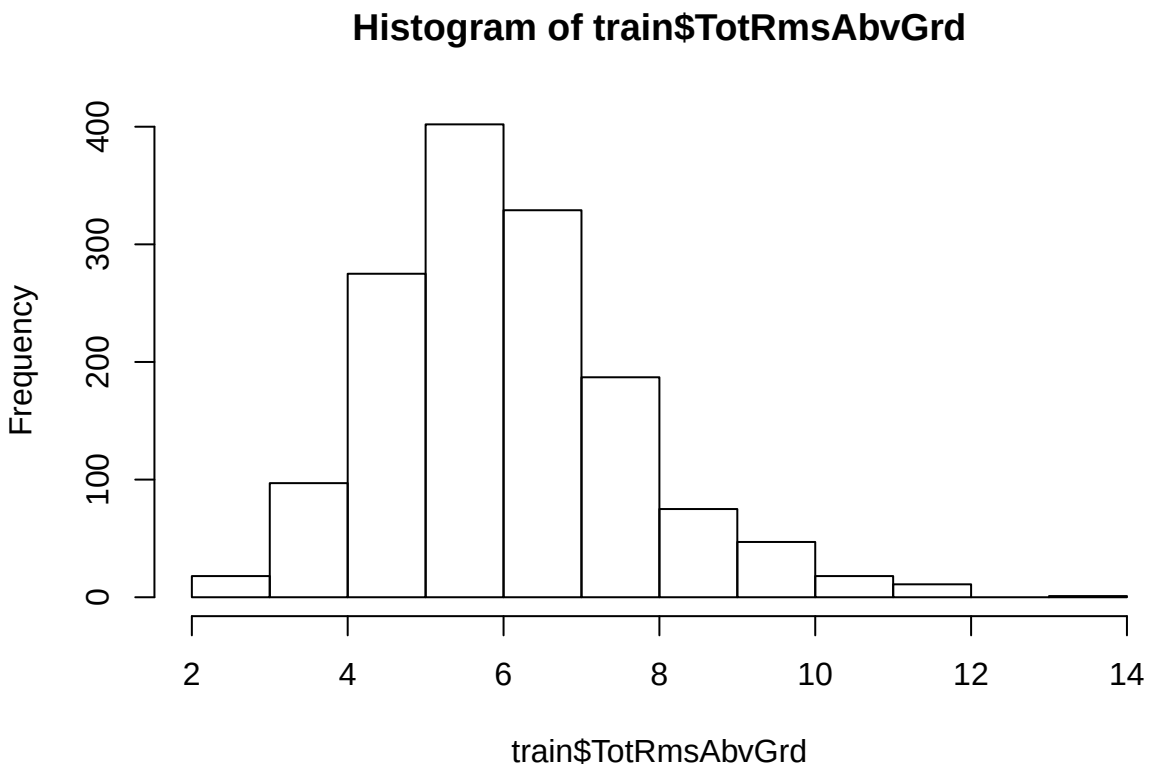
```
simulation <- rexp(1000, rate=0.1534258)
```

Let's compare histograms!

```
hist(simulation)
```




```
hist(train$TotRmsAbvGrd)
```



Hmmm. These are pretty different – the actual data is more gaussian, and the simulation misses this!

Let's get cumulative percentiles for my simulation data:

```
quantile(ecdf(simulation),probs=c(0.05,0.95))
```

```
##         5%        95%
##  0.3853296 19.7244984
```

What's the 95% confidence interval for the **mean** of my variable? I know that the standard error will be the sample standard deviation divided by the square root of the sample size:

```
se <- sd(train$TotRmsAbvGrd)/sqrt(length(train$TotRmsAbvGrd))
```

A 95% confidence interval is within 2 standard errors, so:

```
myConfInt <- mean(train$TotRmsAbvGrd) + c(-2*se, 2*se)
myConfInt
```

```
## [1] 6.432731 6.602885
```

A 95% confidence interval for the **mean** is (6.4327, 6.6029).

But do we want instead the range that encompasses 95% of the data? In other words, 2 standard deviations out from the sample mean? In that case, we would calculate it like this:

```
ninetyFive <- mean(train$TotRmsAbvGrd) +
  c(-2*sd(train$TotRmsAbvGrd), 2*sd(train$TotRmsAbvGrd))
ninetyFive
```

```
## [1] 3.267022 9.768595
```

Finally, what's my empirical 5th and 95th percentile on the actual data? This would encompass 90% of my data.

```
quantile(train$TotRmsAbvGrd,probs=c(0.05,0.95))
```

```
##  5% 95%
##   4  10
```

The investigations here show that an exponential model does not adequately capture the actual behavior of my data. It's much more normal / gaussian than it is exponential!

Modeling

Build some type of regression model and submit your model to the competition board. Provide your complete model summary and results with analysis. Report your Kaggle.com user name and score.

Oh, this is the fun part!!! I like machine learning ensembling. I'll find a few regression models that work reasonably well and then ensemble them (using a Tree or Random Forest algorithm). Let's first use the PCA stuff and see how that works in a generalized linear model!

First off, let's load some libraries we'll want.

```
library(lattice)
library(ggplot2)
library(caret)
```

We want to put the five PCA's we know are the most helpful next to the all of of our training data – that way we have a data frame that has all of our variables, both given and computed. We will, however, remove the "Id" field.

```
pcaDataFrame <- cbind.data.frame(principalComponents$x[,1:5],
                                 SalePrice=train$SalePrice)
```

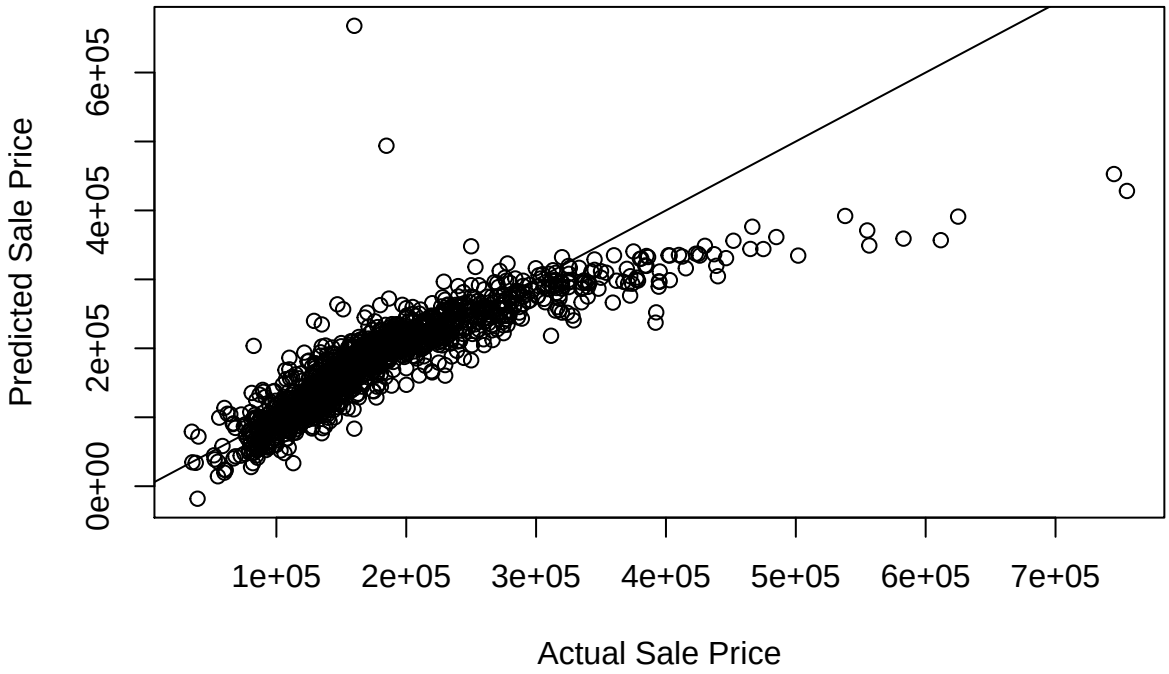
Now let's set up a model.

```
set.seed(42)
model_1<-train(SalePrice ~ .,
                      data=pcaDataFrame, method="glm",
                      trControl=trainControl())
model_1
```

```
## Generalized Linear Model
##
## 1460 samples
##    5 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1460, 1460, 1460, 1460, 1460, 1460, ...
## Resampling results:
##
##   RMSE      Rsquared
##   38767.37  0.765214
##
##
```

The R^2 is a nice high 0.77, and the root mean squared error is not too shabby at around $39k. Let's plot both the actual and predicted values and see if they form a nice line. We will plot a line with slope 1 which would represent 100% accuracy.

```
plot(pcaDataFrame$SalePrice,
     predict(model_1, pcaDataFrame),
     xlab = "Actual Sale Price",
              ylab = "Predicted Sale Price")
abline(a=0, b=1)
```



Not terrible! There are a couple of major outliers, and high value homes don't follow our predictions that well, but overall we did ok.

What if, instead, we did a linear model on 15 principal components?

```
pcaDataFrame2 <- cbind.data.frame(principalComponents$x[,1:15],
                                  SalePrice=train$SalePrice)
```

Now let's set up a model.

```
set.seed(42)
model_2<-train(SalePrice ~ .,
                      data=pcaDataFrame2, method="glm",
                      trControl=trainControl())
model_2
```

```
## Generalized Linear Model
##
## 1460 samples
```

```
##   15 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1460, 1460, 1460, 1460, 1460, 1460, ...
## Resampling results:
##
##   RMSE      Rsquared
##   39656.09  0.7548709
##
##
```

We didn't have any improvement. We'll stick with the first one. But let's do some machine learning tweaks and add some repeated crossvalidation:

```
set.seed(42)
model_3<-train(SalePrice ~ .,
                      data=pcaDataFrame, method="glm",
                     trControl = trainControl(method="repeatedcv",
                                              number=5, repeats = 25))
model_3
```

```
## Generalized Linear Model
##
## 1460 samples
##    5 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 25 times)
## Summary of sample sizes: 1169, 1167, 1168, 1168, 1168, 1168, ...
## Resampling results:
##
##   RMSE      Rsquared
##   38734.87  0.7683633
##
##
```

OK, we squeaked out another percentage point of R^2. Not that impressive, but an improvement nonetheless. We'll keep that repeated cross-validation in place for some of the other models, also. Let's keep in mind, however, that this only accounts for our numeric variables, not our factor variables (we kept categorical and even a few numerical variables out of our PCA). Could that be why our model fails?

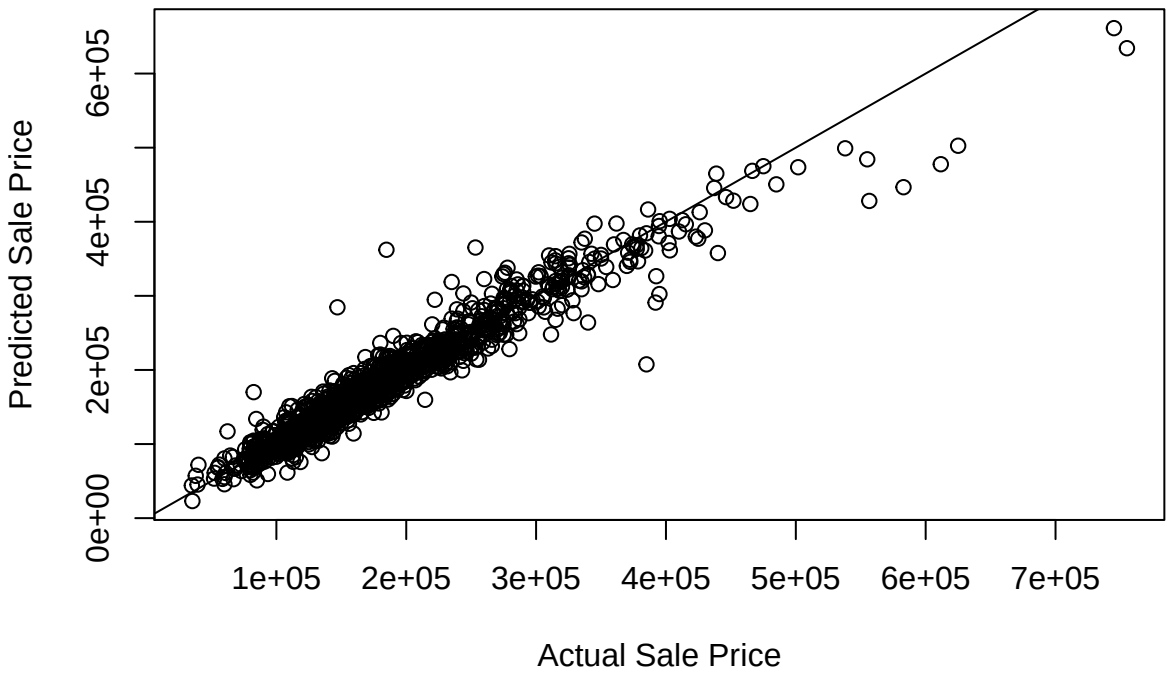
Let's make dummy variables out of our factors so that our categories become numeric, and let's also impute missing values, so that the linear model will work. Note that now that we're working with the original data, we go back to the "train" so that we don't double-dip and overuse some variables (by modeling both on the original variable and on its PCA distillation.)

```
dummy <- dummyVars("~.", data=train)
factorsDummied <- data.frame(predict(dummy, newdata = train))
imputation <- preProcess(factorsDummied, method=("medianImpute"))
imputedFactorsDummied <- predict(imputation,factorsDummied)
set.seed(42)
model_4<-train(SalePrice ~ .,
                      data=imputedFactorsDummied, method="glm",
                      trControl = trainControl(method="repeatedcv",
                                              number=5, repeats = 25))
model_4
```

```
## Generalized Linear Model
##
## 1460 samples
##  289 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 25 times)
## Summary of sample sizes: 1169, 1167, 1168, 1168, 1168, 1168, ...
## Resampling results:
##
##   RMSE      Rsquared
##   66028.25  0.7214582
##
##
```

Jeez, after all that work it's got a higher error and worse R^2! Still, let's look at the plot, out of curiosity.

```
plot(imputedFactorsDummied$SalePrice,
     predict(model_4, imputedFactorsDummied),
     xlab = "Actual Sale Price",
              ylab = "Predicted Sale Price")
abline(a=0, b=1)
```



Huh. It actually looks better than our first one for a lot of the points, especially for the high-end homes. Let's hang on to it as a possible contributor to our ensemble.

In the meantime, what if we did a different algorithm, something like a stepwise linear regression? Let's try it!

This one has lots of annoying output. Sorry!

```
set.seed(42)
model_5<-train(SalePrice ~ .,
                      data=imputedFactorsDummied, method="leapSeq",
                      trControl = trainControl(method="repeatedcv",
                                               number=5, repeats = 25))
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```
model_5
```

```
## Linear Regression with Stepwise Selection
##
## 1460 samples
##  289 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 25 times)
## Summary of sample sizes: 1169, 1167, 1168, 1168, 1168, 1168, ...
## Resampling results across tuning parameters:
##
##   nvmax  RMSE      Rsquared
##   2      77585.11  0.04231492
##   3      75933.11  0.07991196
##   4      75791.29  0.08380667
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was nvmax = 4.
```

Ugh. Not great. What if we tried, say, something that isn't exclusively linear? Like the Generalized Additive Model with LOESS?

```
set.seed(42)
model_6<-train(SalePrice ~ .,
                      data=imputedFactorsDummied, method="gamLoess",
                      trControl = trainControl(method="repeatedcv",
                                               number=5, repeats = 25))
```

```
model_6
```

```
## Generalized Additive Model using LOESS
##
## 1460 samples
##  289 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 25 times)
## Summary of sample sizes: 1169, 1167, 1168, 1168, 1168, 1168, ...
## Resampling results:
##
##   RMSE      Rsquared
##   61051.99  0.7235426
##
## Tuning parameter 'span' was held constant at a value of 0.5
##
## Tuning parameter 'degree' was held constant at a value of 1
##
```

Meh. Not great! But let's throw a k nearest neighbors in there for fun to see what happens!

```
set.seed(42)
model_7<-train(SalePrice ~ .,
                      data=imputedFactorsDummied, method="knn",
                      trControl=trainControl())
model_7
```

```
## k-Nearest Neighbors
##
## 1460 samples
##  289 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1460, 1460, 1460, 1460, 1460, 1460, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared
##   5  51359.93  0.5805240
##   7  50124.23  0.6009557
##   9  49514.34  0.6133504
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was k = 9.
```

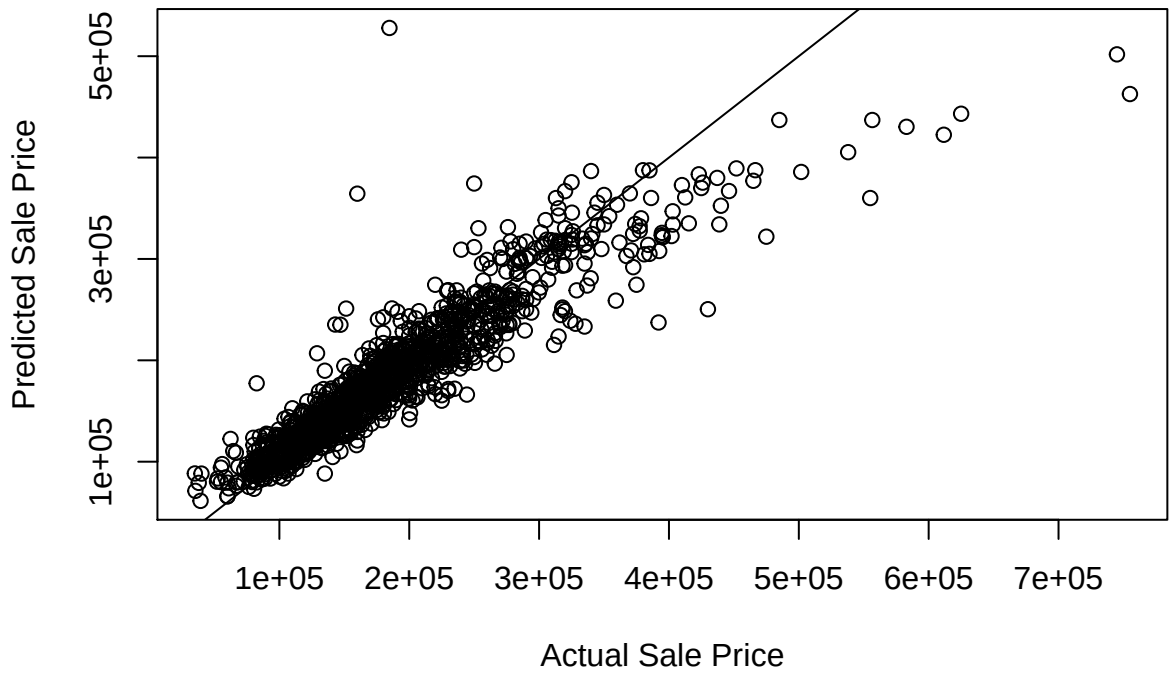
But what if we do a knn on the 5-PCA data frame, instead of the full data?

```
set.seed(42)
model_8<-train(SalePrice ~ .,
                      data=pcaDataFrame, method="knn",
                     trControl = trainControl(method="repeatedcv",
                                              number=5, repeats = 25),
               preProcess = c("center","scale","medianImpute"))
model_8
```

```
## k-Nearest Neighbors
##
## 1460 samples
##    5 predictor
##
## Pre-processing: centered (5), scaled (5), median imputation (5)
## Resampling: Cross-Validated (5 fold, repeated 25 times)
## Summary of sample sizes: 1169, 1167, 1168, 1168, 1168, 1168, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared
##   5  36109.61  0.8004192
##   7  35840.50  0.8069985
##   9  35954.96  0.8087859
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was k = 7.
```

WOW! It really looks like doing models on the PCA is where it's at! This is our best by far so far. Let's look!

```
plot(pcaDataFrame$SalePrice,
     predict(model_8, pcaDataFrame),
     xlab = "Actual Sale Price",
             ylab = "Predicted Sale Price")
abline(a=0, b=1)
```



It's a bit scattered. But it really accounts for the data well. It definitely belongs in our ensemble.

Let's ensemble using k nearest neighbors:

```
predictions <- data_frame(model_3 = predict(model_3, pcaDataFrame,
                                            na.action=na.pass),
                          model_4 = predict(model_4, imputedFactorsDummied,
                                            na.action=na.pass),
                          model_8 = predict(model_8, pcaDataFrame,
                                            na.action=na.pass),
                          SalePrice = train$SalePrice)
set.seed(42)
ensemble<-train(SalePrice ~ .,
                       data=predictions, method="knn",
                      trControl = trainControl(),
                  preProcess = c("center","scale","medianImpute"))
ensemble
```

```
## k-Nearest Neighbors
##
## 1460 samples
##    3 predictor
##
## Pre-processing: centered (3), scaled (3), median imputation (3)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1460, 1460, 1460, 1460, 1460, 1460, ...
## Resampling results across tuning parameters:
##
##   k  RMSE      Rsquared
##   5  23687.90  0.9098027
##   7  22939.57  0.9151857
##   9  22566.40  0.9182086
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was k = 9.
```

That's pretty amazing: 92% of variance is explained! I think we have a winner!

Let's sum up what we've done to the training set so we can do the same thing to the test set.

- We applied PCA to the set, and used it to get a set of PC variables. In our training data we just used the $x part, but for the test data we'll need to use a predict setup.
- We used dummying to handle data in the full model, and imputed empty data using medians.
- Two models used the PCA data: model 8 and model 3.
- One model used the entire data set: model 4.
- We ensembled them together using knn.

Let's do that same thing to the test set:

```
test <- read.csv("Downloads/test.csv")
pcaTest<-predict(principalComponents,test)[,1:5]

# Here we'll get into trouble if we don't add levels to the test set
# that existed in the training set.  They'll be empty, but that's okay.
levels(test$Utilities)<- c(levels(test$Utilities), "NoSeWa")
levels(test$Condition2)<-c(levels(test$Condition2),   "RRAe",   "RRAn", "RRNn")
levels(test$HouseStyle)<-c(levels(test$HouseStyle), "2.5Fin")
levels(test$RoofMatl) <- c(levels(test$RoofMatl), "ClyTile", "Membran", "Metal","Roll")
levels(test$Exterior1st) <- c(levels(test$Exterior1st), "ImStucc", "Stone")
```

```
levels(test$Exterior2nd) <- c(levels(test$Exterior2nd), "Other")
levels(test$Heating) <- c(levels(test$Heating), "Floor", "OthW")
levels(test$Electrical) <- c(levels(test$Electrical), "Mix")
levels(test$GarageQual) <- c(levels(test$GarageQual), "Ex")
levels(test$PoolQC) <- c(levels(test$PoolQC), "Fa")
levels(test$MiscFeature) <- c(levels(test$MiscFeature), "TenC")

dummyTest <- dummyVars("~.", data=test, drop2nd = TRUE)
factorsDummiedTest <- data.frame(predict(dummyTest, newdata = test))
imputationTest <- preProcess(factorsDummiedTest, method=("medianImpute"))
imputedFactorsDummiedTest <- predict(imputation,factorsDummiedTest)

predictionsTest <- data_frame(model_3 = predict(model_3, pcaTest,
                                        na.action=na.pass),
                          model_4 = predict(model_4, imputedFactorsDummiedTest,
                                        na.action=na.pass),
                          model_8 = predict(model_8, pcaTest,
                                        na.action=na.pass))
myPreds <- data.frame(Id = test$Id, SalePrice = predict(ensemble,predictionsTest, na.action=na.pass))
write.csv(myPreds, "myPreds.csv", row.names = FALSE)
```

When I submitted my predictions to Kaggle (as JoyPayton), my score was 0.1474.