KC Housing Predictive Analytics

SANATAN

SNEHI

10/1/2023

Pre-processing Data

Load required packages

```
library(lubridate)
library(car)
library(psych)
library(leaps)
library(FNN)
library(MASS)
library(glmnet)
library(broom)
library(ggplot2)
```

Read in data

```
options(scipen=999)
df <- read.csv("KC_House_Data.csv")
```

Remove commas and dollar signs from numeric data

```
df$price <- gsub(",", "", df$price)
df$price <- as.numeric(gsub("$", "", df$price, fixed=TRUE))
df$sqft_living <-as.numeric(gsub(",", "", df$sqft_living))
df$sqft_lot <- as.numeric(gsub(",", "", df$sqft_lot))
df$sqft_above <- as.numeric(gsub(",", "", df$sqft_above))
df$sqft_basement <- as.numeric(gsub(",", "", df$sqft_basement))
```

Fix date column

```
df$date <- gsub("T000000", "", df$date)
df$date <- as.Date(paste0(substr(df$date, 0, 4), "-",
                          substr(df$date, 5, 6), "-",
                          substr(df$date, 7, 9)))
```

```
## Warning in strptime(xx, f <- "%Y-%m-%d", tz = "GMT"): unknown timezone
## "zone/tz/2018c.1.0/zoneinfo/America/New_York"
```

Remove extraneous variables

Only select variables that we can model in a linear regression. Must remove ID and zipcode

```r
mod.data <- df[, !colnames(df) %in% c("id", "zipcode")]
```

Engineer new features

Make new dummy-coded numeric predictor for fiscal quarter

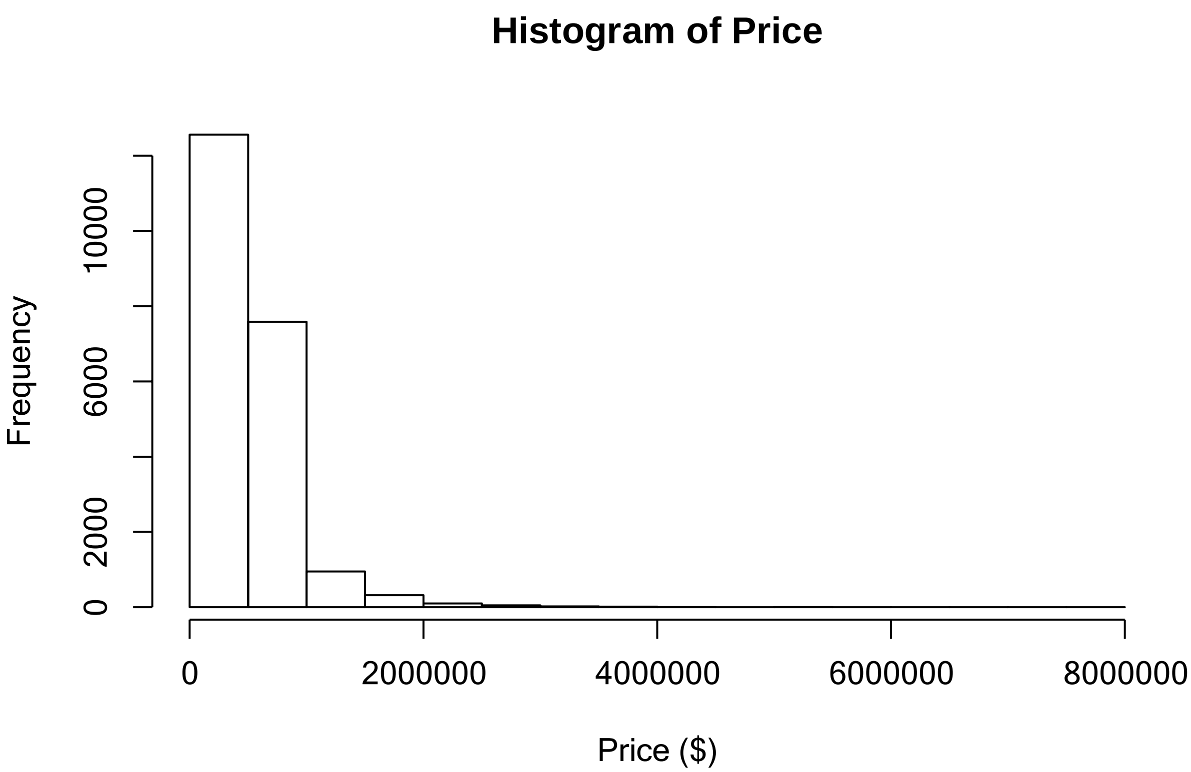
```r
mod.data$fiscal_quarter <- as.factor(quarter(mod.data$date))
fq_dc <- dummy.code(mod.data$fiscal_quarter)
colnames(fq_dc) <- c("fq_1", "fq_2", "fq_3", "fq_4")
mod.data <- cbind(mod.data, fq_dc)
mod.data$fiscal_quarter <- NULL
```

Make new predictor for year of date and remove original date column

```r
mod.data$year <- as.numeric(year(mod.data$date))
mod.data$date <- NULL
```

Explore a few variables

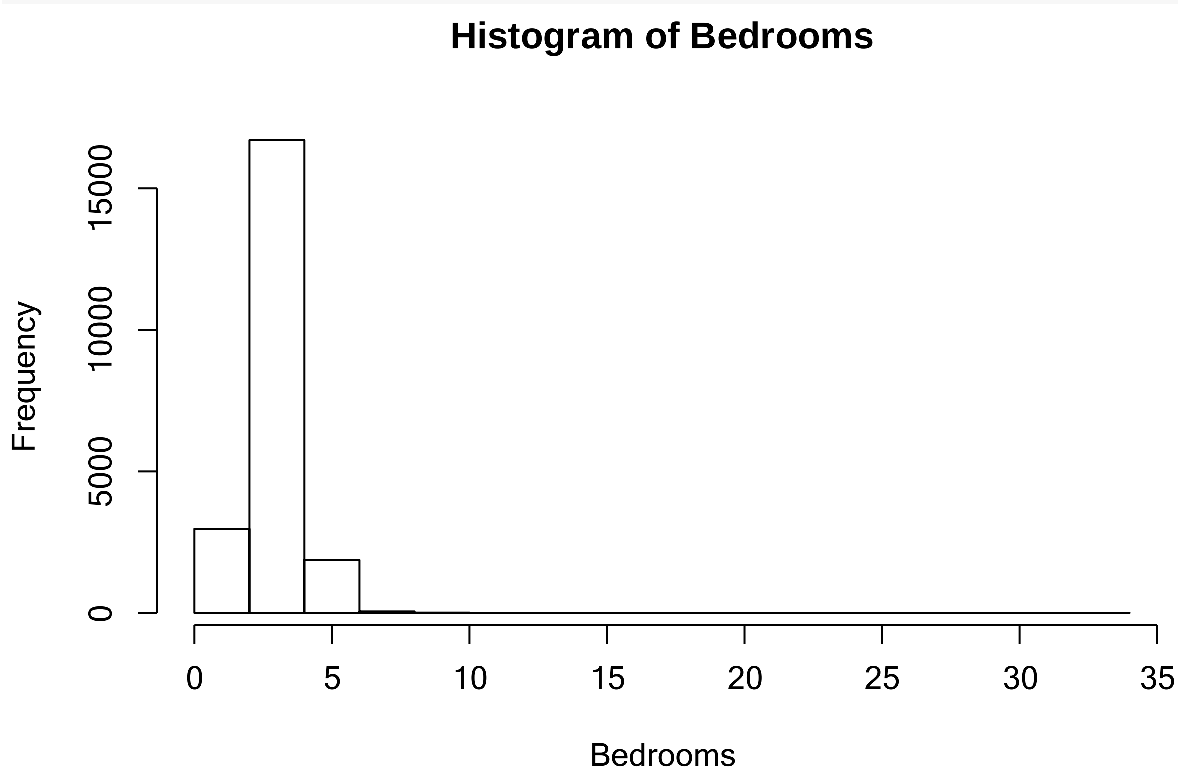
```r
hist(mod.data$price,
     main="Histogram of Price",
     xlab="Price ($)")
```



Looks like price is right skewed. This is good to keep in mind when checking the regression assumptions.

```r
hist(mod.data$sqft_living,
     main="Histogram of Sqft living",
     xlab="Sq. feet")
```


```r
hist(mod.data$bedrooms,
     main="Histogram of Bedrooms",
     xlab="Bedrooms")
```



Check for multicollinearity

```r
cor(mod.data)[which(cor(mod.data) > .8)]
```

```
##  [1] 1.0000000 1.0000000 1.0000000 1.0000000 0.8765966 1.0000000 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 1.0000000 0.8765966 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [22] 1.0000000 1.0000000 1.0000000 1.0000000
```

Correlation matrix finds only one pairwise correlation above .8 So aside from this, multicollinearity may not be too big of a problem.

```r
cor(mod.data)
```

```
##                     price      bedrooms    bathrooms  sqft_living
## price         1.000000000  0.3083495981  0.525137505  0.702035055
## bedrooms      0.308349598  1.0000000000  0.515883638  0.576670693
## bathrooms     0.525137505  0.5158836376  1.000000000  0.754665279
## sqft_living   0.702035055  0.5766706925  0.754665279  1.000000000
## sqft_lot      0.089660861  0.0317032429  0.087739662  0.172825661
## floors        0.256793888  0.1754289352  0.500653173  0.353949290
## waterfront    0.266369434 -0.0065824787  0.063743629  0.103817818
## view          0.397293488  0.0795318518  0.187737024  0.284611186
## condition     0.036361789  0.0284721044 -0.124981933 -0.058752587
## grade         0.667434256  0.3569667254  0.664982534  0.762704476
## sqft_above    0.605567298  0.4776001614  0.685342476  0.876596599
## sqft_basement 0.323816021  0.3030933753  0.283770034  0.435042974
## yr_built      0.054011531  0.1541780695  0.506019438  0.318048769
## yr_renovated  0.126433793  0.0188408231  0.050738978  0.055362927
## lat           0.307003480 -0.0089310097  0.024572953  0.052529462
## long          0.021626241  0.1294729753  0.223041843  0.240223298
## sqft_living15 0.585378904  0.3916375240  0.568634290  0.756420259
## sqft_lot15    0.082447153  0.0292442236  0.087175361  0.183285551
## fq_1         -0.015074780 -0.0036233555 -0.022572597 -0.024368392
## fq_2          0.030735765  0.0078135837  0.011862476  0.010560451
## fq_3         -0.004490588 -0.0002494376  0.015496533  0.012632080
## fq_4         -0.015375474 -0.0050667751 -0.008625109 -0.002381837
## year          0.003576041 -0.0098384336 -0.026595984 -0.029038341
##                   sqft_lot       floors    waterfront         view
## price         0.0896608606  0.256793888  0.2663694340  0.397293488
## bedrooms      0.0317032429  0.175428935 -0.0065824787  0.079531852
## bathrooms     0.0877396615  0.500653173  0.0637436291  0.187737024
## sqft_living   0.1728256613  0.353949290  0.1038178177  0.284611186
## sqft_lot      1.0000000000 -0.005200991  0.0216036833  0.074710106
## floors       -0.0052009909  1.000000000  0.0236983203  0.029443820
## waterfront    0.0216036833  0.023698320  1.0000000000  0.401857351
## view          0.0747101056  0.029443820  0.4018573507  1.000000000
## condition    -0.0089582495 -0.263767946  0.0166531574  0.045989737
## grade         0.1136211236  0.458182514  0.0827749139  0.251320585
## sqft_above    0.1835122809  0.523884710  0.0720745917  0.167649344
## sqft_basement 0.0152862016 -0.245704542  0.0805879390  0.276946579
## yr_built      0.0530803670  0.489319425 -0.0261610856 -0.053439851
## yr_renovated  0.0076435050  0.006338401  0.0928848367  0.103917288
## lat          -0.0856827882  0.049614131 -0.0142737756  0.006156732
## long          0.2295208588  0.125419028 -0.0419102001 -0.078399712
```

```
## sqft_living15  0.1446081737  0.279885265  0.0864631361  0.280439082
## sqft_lot15     0.7185567524 -0.011269187  0.0307032831  0.072574568
## fq_1           0.0053529387 -0.022974159 -0.0053781383  0.004873296
## fq_2           0.0001700892  0.004302714 -0.0005436224  0.002452316
## fq_3          -0.0080443272  0.012911930 -0.0032189162 -0.003417410
## fq_4           0.0034026204  0.003016589  0.0091643027 -0.003683922
## year           0.0054684312 -0.022314901 -0.0041647548  0.001363816
##                  condition        grade    sqft_above  sqft_basement
## price          0.036361789  0.667434256  0.6055672984  0.3238160207
## bedrooms       0.028472104  0.356966725  0.4776001614  0.3030933753
## bathrooms     -0.124981933  0.664982534  0.6853424759  0.2837700340
## sqft_living   -0.058752587  0.762704476  0.8765965987  0.4350429737
## sqft_lot      -0.008958250  0.113621124  0.1835122809  0.0152862016
## floors        -0.263767946  0.458182514  0.5238847103 -0.2457045423
## waterfront     0.016653157  0.082774914  0.0720745917  0.0805879390
## view           0.045989737  0.251320585  0.1676493441  0.2769465788
## condition      1.000000000 -0.144673671 -0.1582136164  0.1741049139
## grade         -0.144673671  1.000000000  0.7559229376  0.1683918249
## sqft_above    -0.158213616  0.755922938  1.0000000000 -0.0519433068
## sqft_basement  0.174104914  0.168391825 -0.0519433068  1.0000000000
## yr_built      -0.361416562  0.446963205  0.4238983517 -0.1331240989
## yr_renovated  -0.060617787  0.014414281  0.0232846879  0.0713229017
## lat           -0.014941006  0.114084057 -0.0008164986  0.1105379580
## long          -0.106500448  0.198372153  0.3438030175 -0.1447647738
## sqft_living15 -0.092824268  0.713202093  0.7318702924  0.2003549834
## sqft_lot15    -0.003405523  0.119247897  0.1940498619  0.0172761806
## fq_1          -0.030257645 -0.027215698 -0.0201824158 -0.0128069844
## fq_2           0.003769628  0.017600577  0.0038221326  0.0147637719
## fq_3           0.027277156  0.014032788  0.0164638764 -0.0045907892
## fq_4          -0.004955903 -0.009091731 -0.0029097468  0.0005015163
## year          -0.045589391 -0.030386811 -0.0238228160 -0.0156866982
##                   yr_built yr_renovated            lat          long
## price          0.054011531  0.126433793  0.30700348000  0.0216262410
## bedrooms       0.154178069  0.018840823 -0.00893100969  0.1294729753
## bathrooms      0.506019438  0.050738978  0.02457295277  0.2230418429
## sqft_living    0.318048769  0.055362927  0.05252946218  0.2402232975
## sqft_lot       0.053080367  0.007643505 -0.08568278824  0.2295208588
## floors         0.489319425  0.006338401  0.04961413102  0.1254190281
## waterfront    -0.026161086  0.092884837 -0.01427377564 -0.0419102001
## view          -0.053439851  0.103917288  0.00615673208 -0.0783997123
## condition     -0.361416562 -0.060617787 -0.01494100639 -0.1065004479
## grade          0.446963205  0.014414281  0.11408405712  0.1983721531
## sqft_above     0.423898352  0.023284688 -0.00081649857  0.3438030175
## sqft_basement -0.133124099  0.071322902  0.11053795798 -0.1447647738
## yr_built       1.000000000 -0.224873518 -0.14812240214  0.4093562026
## yr_renovated  -0.224873518  1.000000000  0.02939760922 -0.0683723687
## lat           -0.148122402  0.029397609  1.00000000000 -0.1355117836
## long           0.409356203 -0.068372369 -0.13551178361  1.0000000000
## sqft_living15  0.326228900 -0.002672555  0.04885793208  0.3346049838
## sqft_lot15     0.070957926  0.007853765 -0.08641880719  0.2544512877
## fq_1           0.002156640 -0.016179532 -0.03028156895 -0.0007071224
## fq_2          -0.003046733  0.002717915  0.01797090123 -0.0017607349
## fq_3           0.010299121  0.008008316  0.00786030454  0.0192464818
## fq_4          -0.009709802  0.003642511  0.00003852295 -0.0180733511
```

```
## year           0.003507321 -0.023706790 -0.02921244195  0.0002697435
##               sqft_living15    sqft_lot15          fq_1          fq_2
## price            0.585378904  0.08244715252 -0.0150747798  0.0307357653
## bedrooms         0.391637524  0.02924422365 -0.0036233555  0.0078135837
## bathrooms        0.568634290  0.08717536082 -0.0225725971  0.0118624763
## sqft_living      0.756420259  0.18328555134 -0.0243683924  0.0105604514
## sqft_lot         0.144608174  0.71855675243  0.0053529387  0.0001700892
## floors           0.279885265 -0.01126918663 -0.0229741590  0.0043027139
## waterfront       0.086463136  0.03070328314 -0.0053781383 -0.0005436224
## view             0.280439082  0.07257456782  0.0048732957  0.0024523165
## condition       -0.092824268 -0.00340552298 -0.0302576448  0.0037696281
## grade            0.713202093  0.11924789718 -0.0272156976  0.0176005770
## sqft_above       0.731870292  0.19404986189 -0.0201824158  0.0038221326
## sqft_basement    0.200354983  0.01727618056 -0.0128069844  0.0147637719
## yr_built         0.326228900  0.07095792640  0.0021566404 -0.0030467326
## yr_renovated    -0.002672555  0.00785376504 -0.0161795321  0.0027179146
## lat              0.048857932 -0.08641880719 -0.0302815690  0.0179709012
## long             0.334604984  0.25445128774 -0.0007071224 -0.0017607349
## sqft_living15    1.000000000  0.18319174870 -0.0225080997  0.0167719820
## sqft_lot15       0.183191749  1.00000000000 -0.0073487790  0.0077742015
## fq_1            -0.022508100 -0.00734877902  1.0000000000 -0.3288548811
## fq_2             0.016771982  0.00777420155 -0.3288548811  1.0000000000
## fq_3             0.015864823 -0.00602208389 -0.2974870551 -0.4175004942
## fq_4            -0.014589417  0.00471680372 -0.2572601695 -0.3610451147
## year            -0.021734099 -0.00008494424  0.7008848304  0.1432275985
##                        fq_3          fq_4          year
## price           -0.0044905883 -0.01537547369  0.00357604088
## bedrooms        -0.0002494376 -0.00506677512 -0.00983843356
## bathrooms        0.0154965325 -0.00862510944 -0.02659598439
## sqft_living      0.0126320799 -0.00238183693 -0.02903834116
## sqft_lot        -0.0080443272  0.00340262037  0.00546843123
## floors           0.0129119301  0.00301658907 -0.02231490127
## waterfront      -0.0032189162  0.00916430266 -0.00416475482
## view            -0.0034174099 -0.00368392220  0.00136381629
## condition        0.0272771557 -0.00495590295 -0.04558939064
## grade            0.0140327880 -0.00909173136 -0.03038681059
## sqft_above       0.0164638764 -0.00290974678 -0.02382281599
## sqft_basement   -0.0045907892  0.00050151633 -0.01568669819
## yr_built         0.0102991211 -0.00970980203  0.00350732057
## yr_renovated     0.0080083156  0.00364251059 -0.02370678953
## lat              0.0078603045  0.00003852295 -0.02921244195
## long             0.0192464818 -0.01807335105  0.00026974353
## sqft_living15    0.0158648232 -0.01458941662 -0.02173409890
## sqft_lot15      -0.0060220839  0.00471680372 -0.00008494424
## fq_1            -0.2974870551 -0.25726016948  0.70088483039
## fq_2            -0.4175004942 -0.36104511465  0.14322759852
## fq_3             1.0000000000 -0.32660682287 -0.42444499033
## fq_4            -0.3266068229  1.00000000000 -0.36705055998
## year            -0.4244449903 -0.36705055998  1.00000000000
```

Correlation matrix suggests sqft_above and sqft_living are highly correlated. But do the Variance Inflation Factors also suggest this?

Let’s build a model and check the Variance Inflation Factors (VIF)

```r
mod <- lm(price ~., data = mod.data)
summary(mod)
```

```
##
## Call:
## lm(formula = price ~ ., data = mod.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1255857   -99218    -9251    76552  4349902
##
## Coefficients: (2 not defined because of singularities)
##                       Estimate        Std. Error t value
## (Intercept)   -122930858.69569   10128516.63370 -12.137
## bedrooms          -34295.06543       1898.19159 -18.067
## bathrooms          42244.95271       3267.63621  12.928
## sqft_living          146.93260          4.40067  33.389
## sqft_lot               0.12624          0.04814   2.622
## floors              1220.17056       3596.59018   0.339
## waterfront        588083.77822      17435.25654  33.730
## view               49182.23532       2140.94740  22.972
## condition          32340.52300       2351.25773  13.755
## grade              97422.17522       2161.36349  45.074
## sqft_above            32.79708          4.37894   7.490
## sqft_basement               NA               NA      NA
## yr_built           -2453.12751         72.38325 -33.891
## yr_renovated          22.54051          3.67173   6.139
## lat               563146.30161      10522.03594  53.521
## long             -116918.64214      11965.22544  -9.772
## sqft_living15         27.39131          3.44789   7.944
## sqft_lot15            -0.39535          0.07360  -5.372
## fq_1              -21331.98557       6568.04369  -3.248
## fq_2                -349.73849       4356.22781  -0.080
## fq_3                -544.00700       3937.94624  -0.138
## fq_4                        NA               NA      NA
## year               42680.77712       4962.76967   8.600
##                             Pr(>|t|)
## (Intercept)   < 0.0000000000000002 ***
## bedrooms      < 0.0000000000000002 ***
## bathrooms     < 0.0000000000000002 ***
## sqft_living   < 0.0000000000000002 ***
## sqft_lot                   0.00874 **
## floors                     0.73442
## waterfront    < 0.0000000000000002 ***
## view          < 0.0000000000000002 ***
## condition     < 0.0000000000000002 ***
## grade         < 0.0000000000000002 ***
## sqft_above     0.00000000000007166 ***
## sqft_basement                   NA
## yr_built      < 0.0000000000000002 ***
## yr_renovated   0.00000000084521335 ***
## lat           < 0.0000000000000002 ***
## long          < 0.0000000000000002 ***
## sqft_living15  0.00000000000000205 ***
```

```
## sqft_lot15     0.00000007884524058 ***
## fq_1                       0.00116 **
## fq_2                       0.93601
## fq_3                       0.89013
## fq_4                            NA
## year          < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 202100 on 21592 degrees of freedom
## Multiple R-squared:  0.6972, Adjusted R-squared:  0.6969
## F-statistic:  2485 on 20 and 21592 DF,  p-value: < 0.00000000000000022
```

Interesting - lm function could not fit this model due to sqft_basement and fq_4 All information contained in sqft_basement is fully explained by a combination of the other variables. Same with fq_4.

So remove these problematic variables and re-build model

```
mod.data$sqft_basement <- NULL
mod.data$fq_4 <- NULL

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

```
##
## Call:
## lm(formula = price ~ ., data = mod.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1255857   -99218    -9251    76552  4349902
##
## Coefficients:
##                         Estimate       Std. Error t value
## (Intercept)   -122930858.69569   10128516.63370 -12.137
## bedrooms          -34295.06543       1898.19159 -18.067
## bathrooms          42244.95271       3267.63621  12.928
## sqft_living          146.93260          4.40067  33.389
## sqft_lot               0.12624          0.04814   2.622
## floors              1220.17056       3596.59018   0.339
## waterfront        588083.77822      17435.25654  33.730
## view               49182.23532       2140.94740  22.972
## condition          32340.52300       2351.25773  13.755
## grade              97422.17522       2161.36349  45.074
## sqft_above            32.79708          4.37894   7.490
## yr_built           -2453.12751         72.38325 -33.891
## yr_renovated          22.54051          3.67173   6.139
## lat               563146.30161      10522.03594  53.521
## long             -116918.64214      11965.22544  -9.772
## sqft_living15         27.39131          3.44789   7.944
## sqft_lot15            -0.39535          0.07360  -5.372
## fq_1              -21331.98557       6568.04369  -3.248
## fq_2                -349.73849       4356.22781  -0.080
## fq_3                -544.00700       3937.94624  -0.138
## year               42680.77712       4962.76967   8.600
```

```
##                              Pr(>|t|)
## (Intercept)    < 0.0000000000000002 ***
## bedrooms       < 0.0000000000000002 ***
## bathrooms      < 0.0000000000000002 ***
## sqft_living    < 0.0000000000000002 ***
## sqft_lot                    0.00874 **
## floors                      0.73442
## waterfront     < 0.0000000000000002 ***
## view           < 0.0000000000000002 ***
## condition      < 0.0000000000000002 ***
## grade          < 0.0000000000000002 ***
## sqft_above      0.00000000000007166 ***
## yr_built       < 0.0000000000000002 ***
## yr_renovated    0.00000000084521335 ***
## lat            < 0.0000000000000002 ***
## long           < 0.0000000000000002 ***
## sqft_living15   0.00000000000000205 ***
## sqft_lot15      0.00000007884524058 ***
## fq_1                        0.00116 **
## fq_2                        0.93601
## fq_3                        0.89013
## year           < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 202100 on 21592 degrees of freedom
## Multiple R-squared:  0.6972, Adjusted R-squared:  0.6969
## F-statistic:  2485 on 20 and 21592 DF,  p-value: < 0.00000000000000022
```

Check VIF

```
vif(mod)
```

```
##      bedrooms     bathrooms   sqft_living      sqft_lot        floors
##      1.648686      3.350172      8.641191      2.103003      1.995197
##    waterfront          view     condition         grade    sqft_above
##      1.203638      1.423848      1.238380      3.414323      6.955493
##      yr_built  yr_renovated           lat          long sqft_living15
##      2.391218      1.150624      1.124431      1.501952      2.954046
##    sqft_lot15          fq_1          fq_2          fq_3          year
##      2.136259      3.509813      2.168990      1.632374      2.848795
```

Only two possibly problematic variables: sqft_above and sqft_living. However, neither VIFs exceed 10, which means it is probably safe to leave them in.

Check condition index

Condition index is the square root of the ratio of the largest eigenvalue to the corresponding eigenvalue.

```
cor.mat <- cor(mod.data)
eigens <- eigen(cor.mat)

con.ind <- sqrt(max(eigens$values)/eigens$values)
con.ind
```

```
##  [1] 1.000000 1.660717 1.685613 1.757691 1.989242 2.125974 2.237777
##  [8] 2.353701 2.664405 2.851077 2.898956 3.253407 3.420251 3.626746
## [15] 4.230936 4.503124 4.720692 5.178546 5.472258 6.162769 8.831967
```

Condition number is the largest condition index

```r
con.num <- max(con.ind)
con.num
```

```
## [1] 8.831967
```

Condition numbers of 30-100 are considered strong multicollinearity. This data has a low condition number, so multicollinearity is likely not a problem.

Models for Predictive Analytics

First, lets split the data into a 70% training and 30% test set, for model validation.

```r
set.seed(42)
test.i <- sample(1:nrow(mod.data), .3*nrow(mod.data), replace=FALSE)
test.data <- mod.data[test.i,]
train.data <- mod.data[-test.i,]
```

Folk Wisdom Model

First we will build a model that is based on folk wisdom and common sense about real estate. We'll later see how it compares to the other models we built with an algorithm.

```r
folk.mod <- lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors +
                      waterfront + view + condition + sqft_above +
                      year + yr_built + yr_renovated, data = train.data)
summary(folk.mod)
```

```
##
## Call:
## lm(formula = price ~ bedrooms + bathrooms + sqft_living + sqft_lot +
##     floors + waterfront + view + condition + sqft_above + year +
##     yr_built + yr_renovated, data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1388569  -121710   -13276    97446  3904580
##
## Coefficients:
##                     Estimate      Std. Error t value            Pr(>|t|)
## (Intercept)  -34247352.26010   7992291.30934  -4.285     0.00001838298335
## bedrooms        -52764.28858      2511.07846 -21.013 < 0.0000000000000002
## bathrooms        58291.36143      4362.37755  13.362 < 0.0000000000000002
## sqft_living        238.43744         5.53536  43.075 < 0.0000000000000002
## sqft_lot            -0.24353         0.04411  -5.521     0.00000003420766
## floors           52639.20806      4679.07532  11.250 < 0.0000000000000002
## waterfront      535947.42745     23154.15334  23.147 < 0.0000000000000002
## view             61985.22585      2827.80234  21.920 < 0.0000000000000002
## condition        21977.74982      3147.92977   6.982     0.00000000000304
```

```
## sqft_above          37.81664        5.54945   6.814     0.00000000000982
## year             19690.44527     3967.23021   4.963     0.00000070069680
## yr_built         -2818.94952       87.59508 -32.182 < 0.0000000000000002
## yr_renovated        12.41092        4.89131   2.537               0.0112
##
## (Intercept)  ***
## bedrooms     ***
## bathrooms    ***
## sqft_living  ***
## sqft_lot     ***
## floors       ***
## waterfront   ***
## view         ***
## condition    ***
## sqft_above   ***
## year         ***
## yr_built     ***
## yr_renovated *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 227700 on 15117 degrees of freedom
## Multiple R-squared:  0.5934, Adjusted R-squared:  0.5931
## F-statistic:  1839 on 12 and 15117 DF,  p-value: < 0.000000000000000022
```

```r
paste("The R^2 coefficient of determination is", summary(folk.mod)$r.squared)
```

```
## [1] "The R^2 coefficient of determination is 0.593443046567577"
```

```r
paste("The adjusted R^2 is", summary(folk.mod)$adj.r.squared)
```

```
## [1] "The adjusted R^2 is 0.593120318285431"
```

Check VIFs

```r
vif(folk.mod)
```

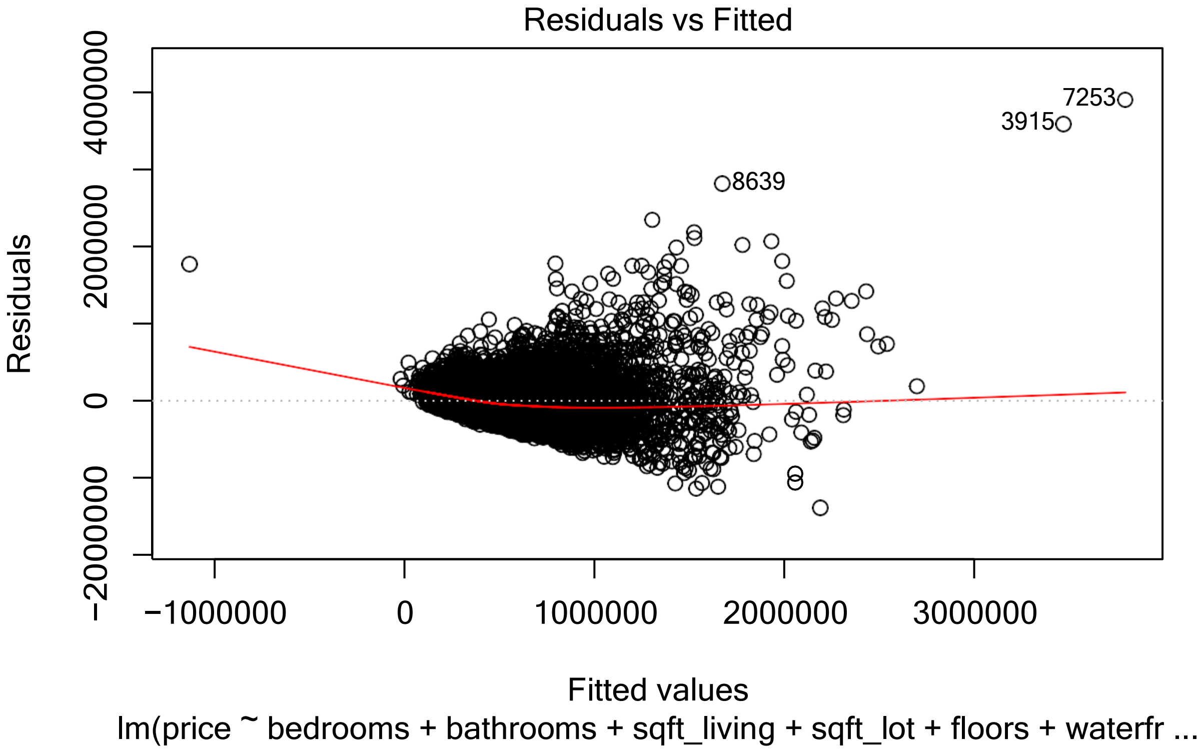
```
##     bedrooms    bathrooms  sqft_living     sqft_lot       floors
##     1.609802     3.261407     7.362610     1.051109     1.863783
##   waterfront         view    condition   sqft_above         year
##     1.210291     1.358390     1.221462     6.014934     1.005230
##     yr_built yr_renovated
##     1.932195     1.149663
```

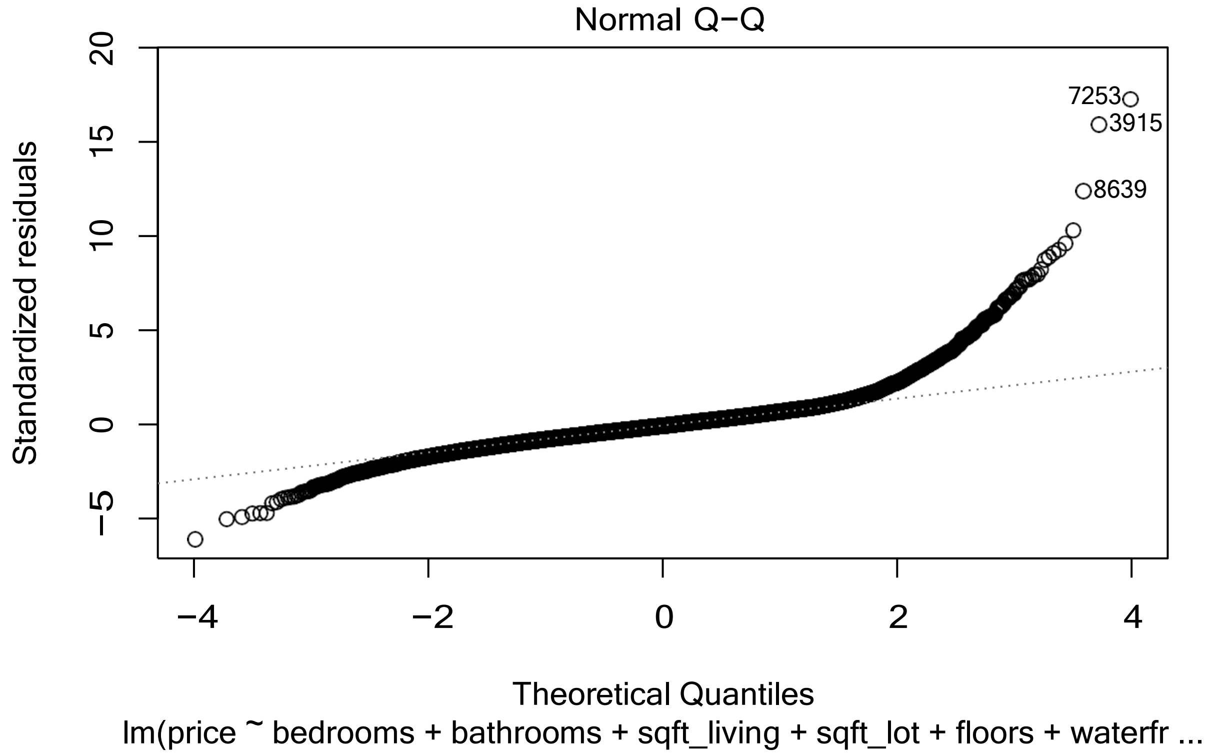
Vifs appear to be pretty solid. None over 10. This implies there is no multicollinearity.

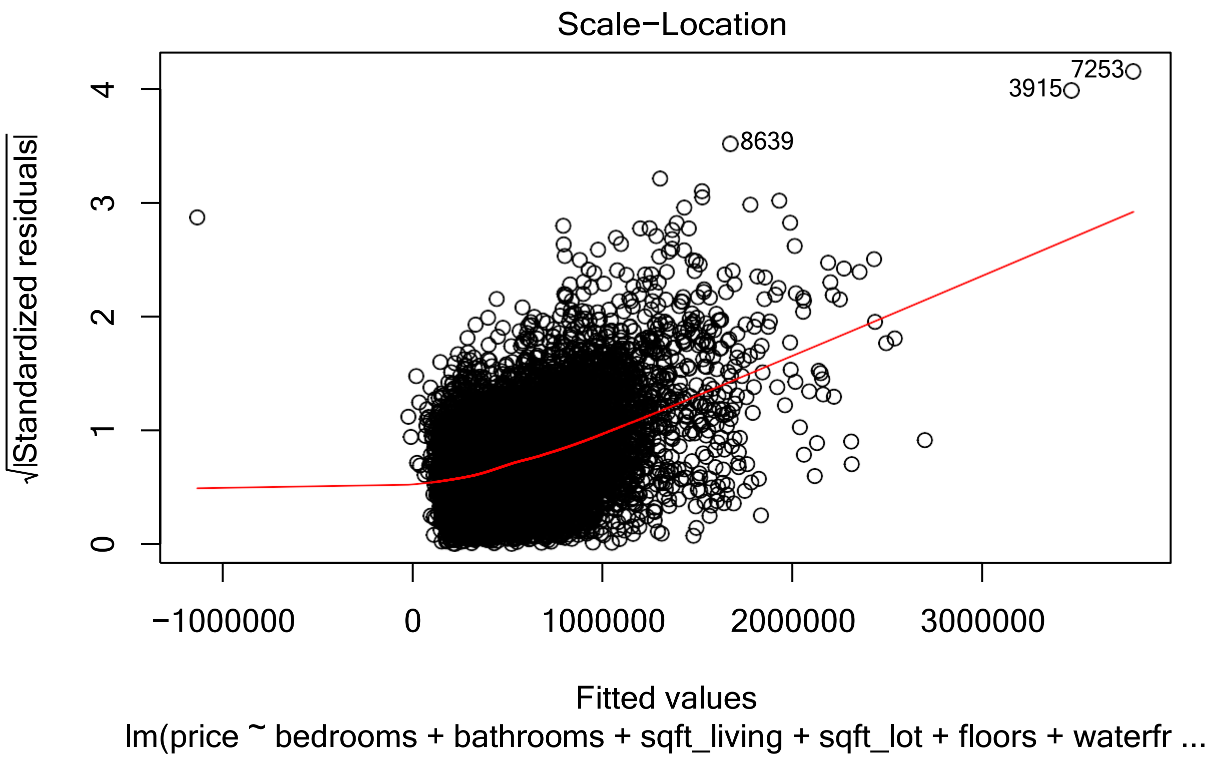
Checking Regression Assumptions

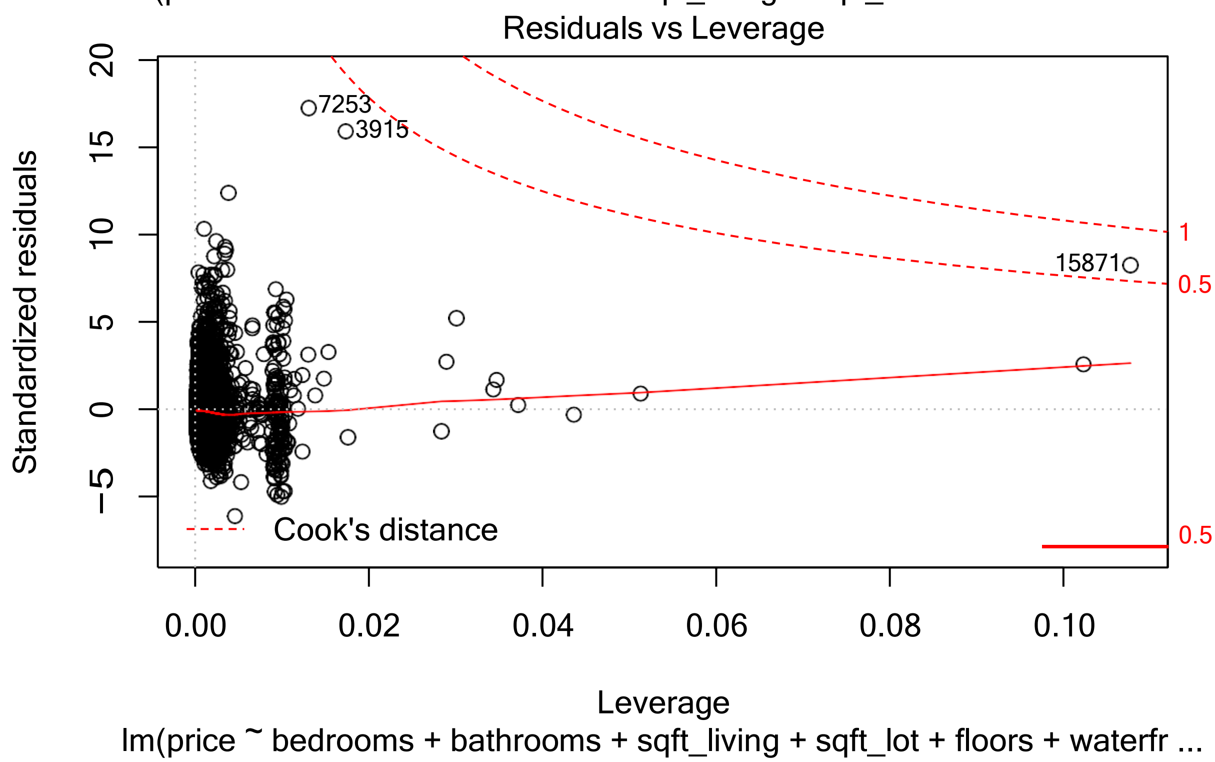
Now let's assess the regression assumptions of this model

```r
plot(folk.mod)
```


Residuals vs Fitted
Residuals
−2000000
0
2000000
4000000
−1000000
0
1000000
2000000
3000000
Fitted values
lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + waterfr ...
7253
3915
8639


Normal Q−Q
Standardized residuals
−5
0
5
10
15
20
−4
−2
0
2
4
Theoretical Quantiles
lm(price ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors + waterfr ...
7253
3915
8639





To me, it looks like the residual variance is not constant. As fitted values increase, the residual variance also increases.

The qqplot reveals that the residuals are not normally distributed.

Thus, while this linear model is significant, there is probably a better non-linear fit to the data.

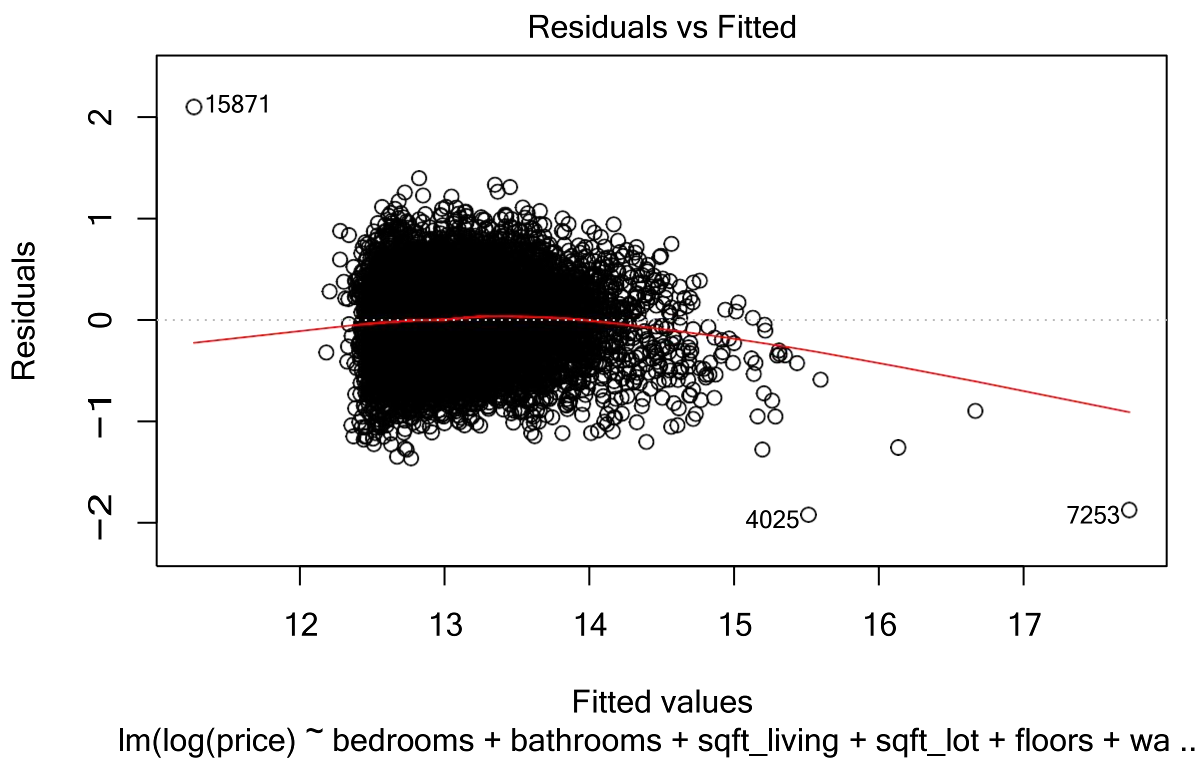
Let's try log-transforming price to see if that improves our fit.

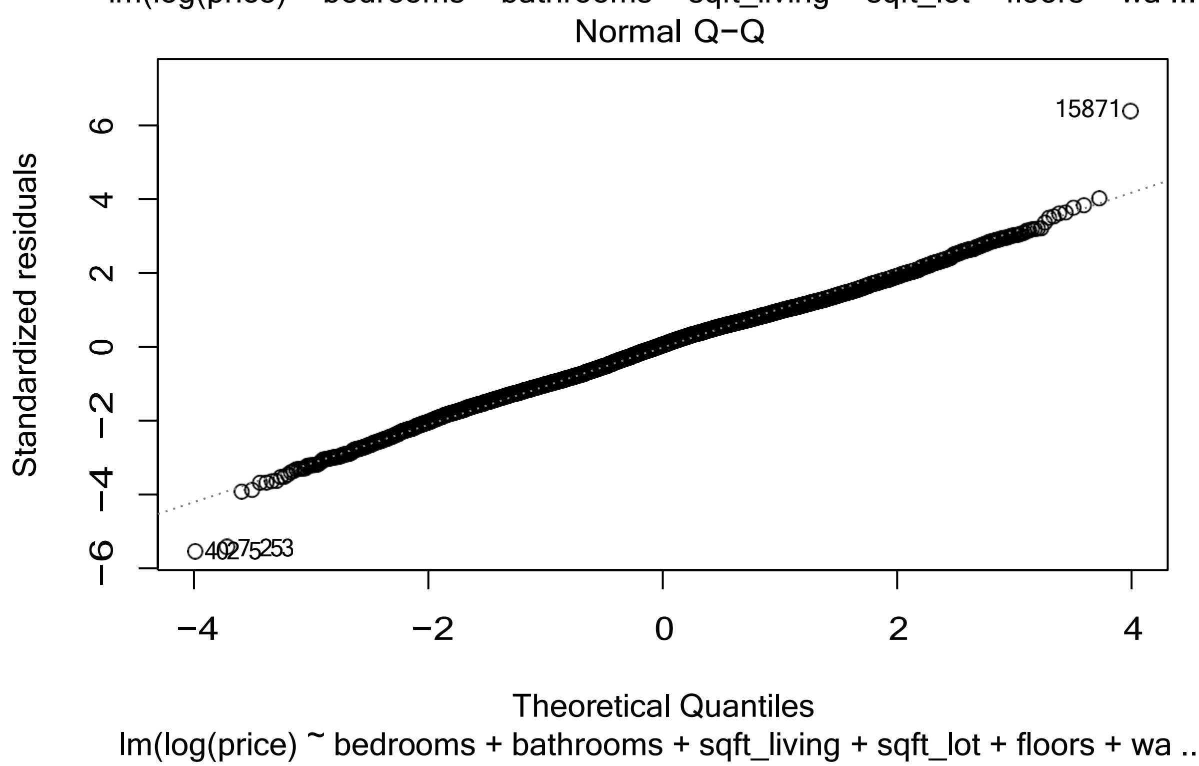
```r
folk.mod.log <- lm(log(price) ~ bedrooms + bathrooms + sqft_living + sqft_lot + floors +
                      waterfront + view + condition + sqft_above +
                      year + yr_built + yr_renovated, data = train.data)
summary(folk.mod.log)
```

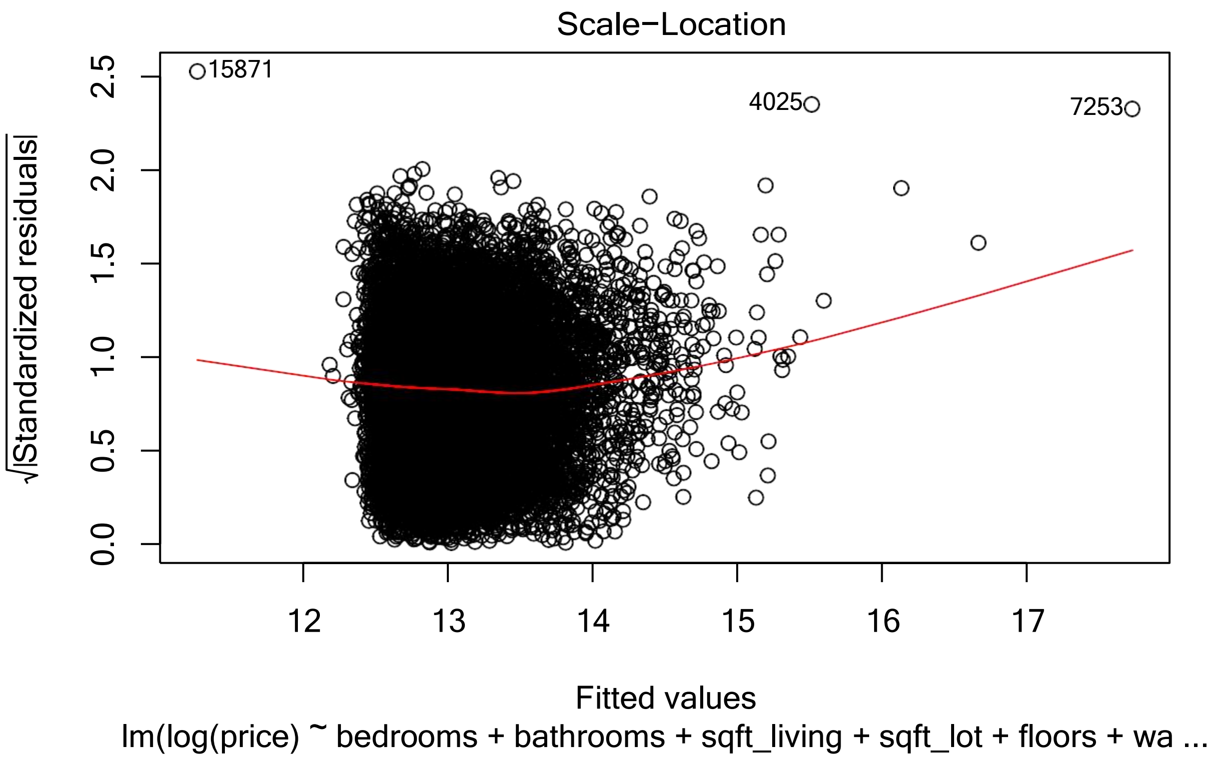
```
##
## Call:
## lm(formula = log(price) ~ bedrooms + bathrooms + sqft_living +
##     sqft_lot + floors + waterfront + view + condition + sqft_above +
##     year + yr_built + yr_renovated, data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.92114 -0.24988  0.01848  0.24158  2.09989
##
## Coefficients:
##                    Estimate      Std. Error t value             Pr(>|t|)
## (Intercept)  -43.5087383243  12.2128483678  -3.563             0.000368
## bedrooms      -0.0551158021   0.0038371250 -14.364 < 0.0000000000000002
## bathrooms      0.1084423847   0.0066660553  16.268 < 0.0000000000000002
## sqft_living    0.0003473065   0.0000084585  41.060 < 0.0000000000000002
## sqft_lot      -0.0000001047   0.0000000674  -1.554             0.120151
## floors         0.1457002632   0.0071499943  20.378 < 0.0000000000000002
## waterfront     0.2883792688   0.0353813635   8.151  0.00000000000000039
## view           0.0813910599   0.0043211039  18.836 < 0.0000000000000002
## condition      0.0447609165   0.0048102837   9.305 < 0.0000000000000002
## sqft_above     0.0000105691   0.0000084800   1.246             0.212655
## year           0.0314375446   0.0060622391   5.186  0.00000021786806888
## yr_built      -0.0040321517   0.0001338522 -30.124 < 0.0000000000000002
## yr_renovated   0.0000138226   0.0000074743   1.849             0.064427
##
## (Intercept)  ***
## bedrooms     ***
## bathrooms    ***
## sqft_living  ***
## sqft_lot
## floors       ***
## waterfront   ***
## view         ***
## condition    ***
## sqft_above
## year         ***
## yr_built     ***
## yr_renovated .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.348 on 15117 degrees of freedom
## Multiple R-squared:  0.5601, Adjusted R-squared:  0.5597
## F-statistic:  1604 on 12 and 15117 DF,  p-value: < 0.00000000000000022
```

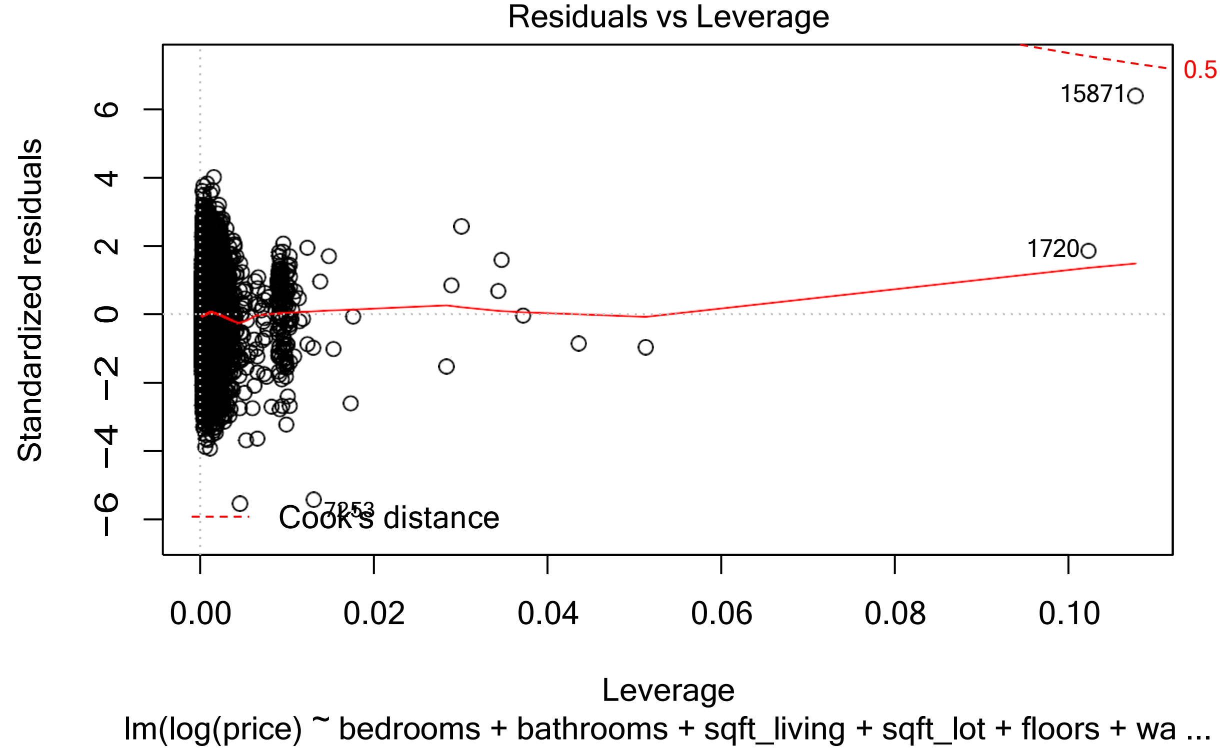
Notice that R^2 went down slightly. However, the sums of squares between this and the previous model are not comparable because we log-transformed the DV. Of more importance, let's check the residuals plot to see if our transformed model better meets the regression assumptions.

```r
plot(folk.mod.log)
```









Indeed, we can see the issue with non-constant variance is much improved.

In addition, the qqplot shows the residuals are now much more normally distributed.

Predictive Performance

Let's now predict with it

```
preds <- predict(folk.mod.log, test.data)
mse <- mean((preds-test.data$price)^2)
paste("The MSE prediction error is", mse)
```

```
## [1] "The MSE prediction error is 453345163468.168"
```

Multiple Linear Regression via Hand-Coded Forward Selection

The following code we found online and adapted it for our problem. It performs stepwise linear regression.

On the first iteration, it finds the single best predictor of price in terms of R^2. On the second iteration, it includes that predictor in the model and then finds the next best predictor of price in terms of overall R^2 of the model. It repeats this process until there are no more predictors remaining.

The result will be a full model in order of

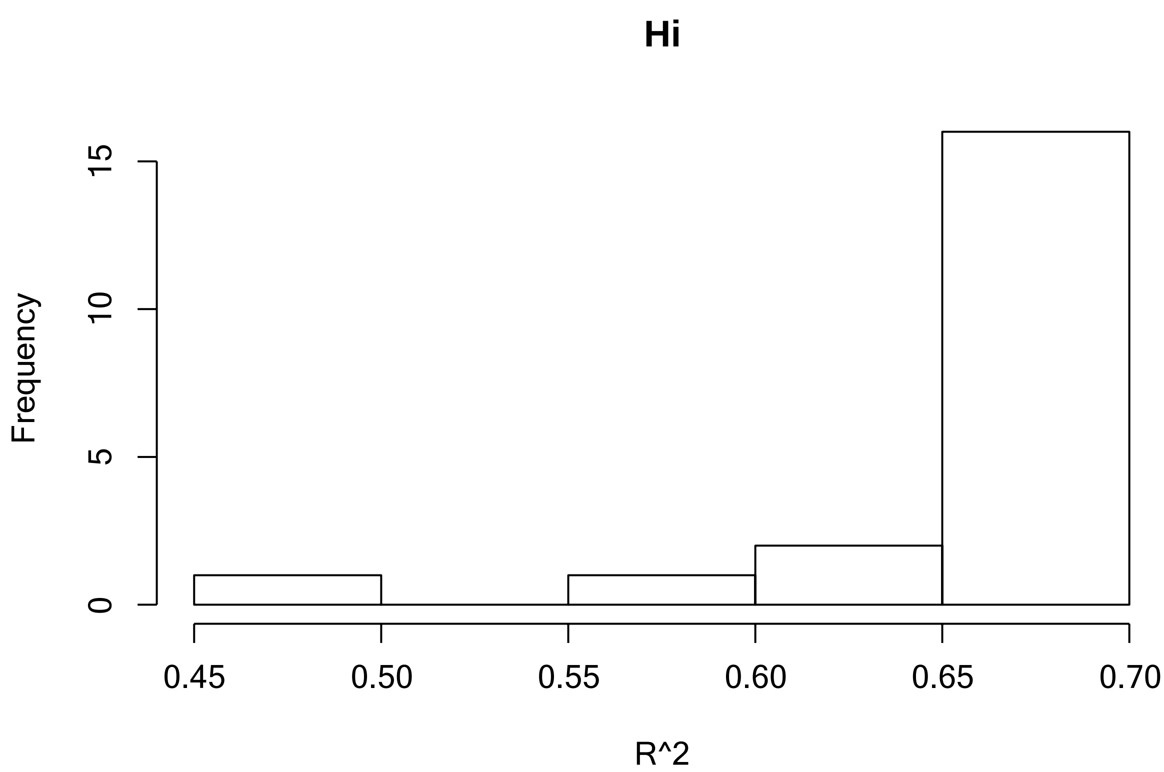
```
# Stepwise linear regression
list.of.used.predictors = list()

mod.predictors <- colnames(mod.data)[!colnames(mod.data) %in% "price"]
r2.bin <- data.frame(Formula=character(length(mod.predictors)),
                     R2 = numeric(length(mod.predictors)),
                     AdjR2 = numeric(length(mod.predictors)),
                     stringsAsFactors = FALSE)

for(j in 1:length(mod.predictors)){
        mod.predictors <- mod.predictors[!(mod.predictors %in% list.of.used.predictors)]

        r.squared.bin <- data.frame(Var = mod.predictors,
                                    R2 = numeric(length(mod.predictors)),
                                    AdjR2 = numeric(length(mod.predictors)))

        for(i in 1:length(mod.predictors)){
                predictor_vars_thusfar = paste(unlist(list.of.used.predictors), collapse="+")
                formula <- paste("price ~ ", predictor_vars_thusfar, " + ", mod.predictors[i], sep = ""
                #print(formula)
                mod <- lm(formula, data = train.data)
                r.squared.bin$Var[i] <- mod.predictors[i]
                r.squared.bin$R2[i] <- summary(mod)$r.squared
                r.squared.bin$AdjR2[i]  <-  summary(mod)$adj.r.squared
        }

        best.var <- r.squared.bin$Var[which.max(r.squared.bin$R2)]
        list.of.used.predictors[[j]] <- as.character(best.var)
        if (j == 1){
                best.formula <- paste("price ~ ", best.var, sep = "")
        } else {
                best.formula <- paste(best.formula, "+", best.var, sep="")}
        print(paste("Best formula on iteration", j, "based on R^2 is: ", best.formula))

        best.current.mod <- lm(best.formula, data = mod.data)
```

```
        r2.bin$Formula[j] <-  as.character(best.formula)
        r2.bin$R2[j] <- summary(best.current.mod)$r.squared
        r2.bin$AdjR2[j]  <-  summary(best.current.mod)$adj.r.squared
}
```

```
## [1] "Best formula on iteration 1 based on R^2 is:  price ~ sqft_living"
## [1] "Best formula on iteration 2 based on R^2 is:  price ~ sqft_living+lat"
## [1] "Best formula on iteration 3 based on R^2 is:  price ~ sqft_living+lat+view"
## [1] "Best formula on iteration 4 based on R^2 is:  price ~ sqft_living+lat+view+grade"
## [1] "Best formula on iteration 5 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built"
## [1] "Best formula on iteration 6 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+water
## [1] "Best formula on iteration 7 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+water
## [1] "Best formula on iteration 8 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+water
## [1] "Best formula on iteration 9 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+water
## [1] "Best formula on iteration 10 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 11 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 12 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 13 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 14 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 15 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 16 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 17 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 18 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 19 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
## [1] "Best formula on iteration 20 based on R^2 is:  price ~ sqft_living+lat+view+grade+yr_built+wate
```

Plot a histogram of the R^2 values

```
hist(r2.bin$R2, main="Hi", xlab="R^2")
```



Select best model based on adjusted R^2

```
paste("The best model overall in terms of adjusted R^2 is:", r2.bin[which.max(r2.bin$AdjR2),"Formula"])
```

```
## [1] "The best model overall in terms of adjusted R^2 is: price ~ sqft_living+lat+view+grade+yr_built
```

Predictive Performance

Let's now predict with it

```
best.hand.fwd.selection.mod <- lm(r2.bin[which.max(r2.bin$AdjR2),"Formula"],
                                  data = train.data)
preds <- predict(best.hand.fwd.selection.mod, test.data)
mse <- mean((preds-test.data$price)^2)
paste("The MSE prediction error is", mse)
```

```
## [1] "The MSE prediction error is 45752558828.1456"
```

Comparison to folk model

Note, the MSE is slightly higher here than the folk wisdom model.

Multiple Linear Regression via Leaps Package Forward Selection

Preprocessing

Variable selection

Perform a forward selection algorithm to find best models. The following code will find the best single predictor model, the best 2-predictor model, the best 3-predictor model, . . . , the best 8-predictor model.

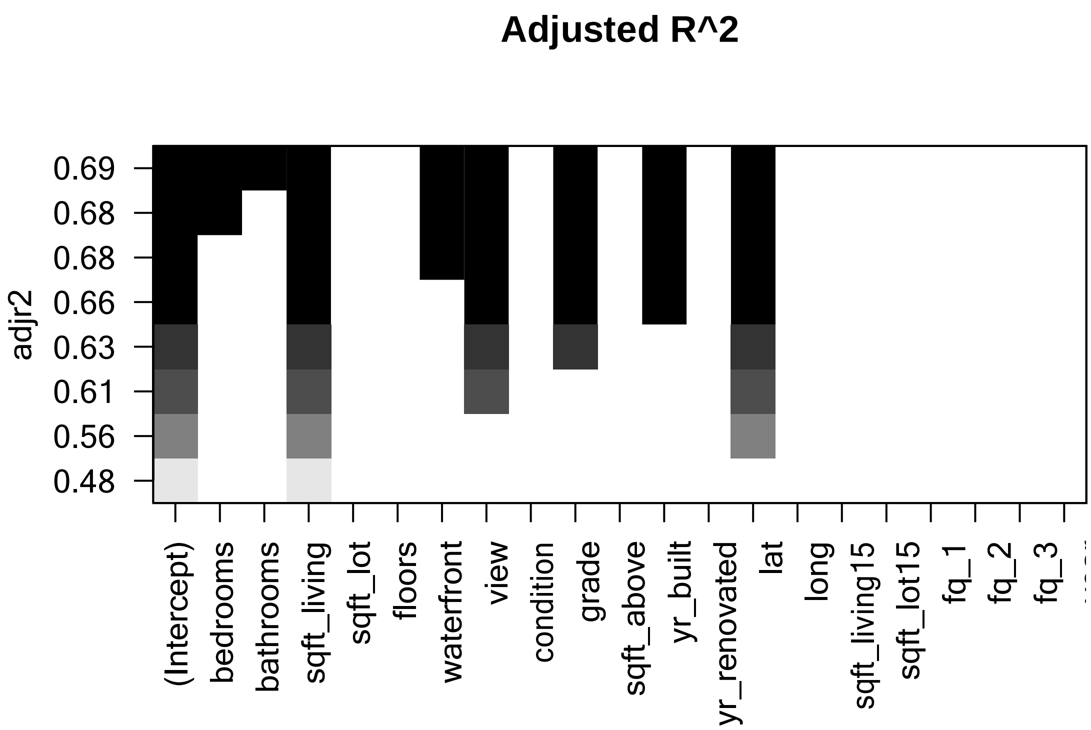
```
regsubsets.out <-
        regsubsets(price ~ .,
                   data = train.data,
                   nbest = 1,        # 1 best models for each number of predictors
                   method = "forward")


summary.out <- summary(regsubsets.out)
summary.out
```

```
## Subset selection object
## Call: regsubsets.formula(price ~ ., data = train.data, nbest = 1, method = "forward")
## 20 Variables  (and intercept)
##               Forced in Forced out
## bedrooms          FALSE      FALSE
## bathrooms         FALSE      FALSE
## sqft_living       FALSE      FALSE
## sqft_lot          FALSE      FALSE
## floors            FALSE      FALSE
## waterfront        FALSE      FALSE
## view              FALSE      FALSE
## condition         FALSE      FALSE
## grade             FALSE      FALSE
## sqft_above        FALSE      FALSE
## yr_built          FALSE      FALSE
## yr_renovated      FALSE      FALSE
## lat               FALSE      FALSE
## long              FALSE      FALSE
## sqft_living15     FALSE      FALSE
## sqft_lot15        FALSE      FALSE
## fq_1              FALSE      FALSE
## fq_2              FALSE      FALSE
## fq_3              FALSE      FALSE
## year              FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          bedrooms bathrooms sqft_living sqft_lot floors waterfront view
## 1  ( 1 ) " "      " "       "*"         " "      " "    " "        " " 
## 2  ( 1 ) " "      " "       "*"         " "      " "    " "        " " 
## 3  ( 1 ) " "      " "       "*"         " "      " "    " "        "*" 
## 4  ( 1 ) " "      " "       "*"         " "      " "    " "        "*" 
## 5  ( 1 ) " "      " "       "*"         " "      " "    " "        "*" 
## 6  ( 1 ) " "      " "       "*"         " "      " "    "*"        "*" 
## 7  ( 1 ) "*"      " "       "*"         " "      " "    "*"        "*" 
## 8  ( 1 ) "*"      "*"       "*"         " "      " "    "*"        "*" 
##          condition grade sqft_above yr_built yr_renovated lat long
## 1  ( 1 ) " "       " "   " "        " "      " "          " " " " 
## 2  ( 1 ) " "       " "   " "        " "      " "          "*" " " 
## 3  ( 1 ) " "       " "   " "        " "      " "          "*" " " 
## 4  ( 1 ) " "       "*"   " "        " "      " "          "*" " " 
## 5  ( 1 ) " "       "*"   " "        "*"      " "          "*" " " 
## 6  ( 1 ) " "       "*"   " "        "*"      " "          "*" " " 
```

```
## 7  ( 1 ) " "        "*"   " "         "*"      " "           "*" " "
## 8  ( 1 ) " "        "*"   " "         "*"      " "           "*" " "
##          sqft_living15 sqft_lot15 fq_1 fq_2 fq_3 year
## 1  ( 1 ) " "           " "        " "  " "  " "  " "
## 2  ( 1 ) " "           " "        " "  " "  " "  " "
## 3  ( 1 ) " "           " "        " "  " "  " "  " "
## 4  ( 1 ) " "           " "        " "  " "  " "  " "
## 5  ( 1 ) " "           " "        " "  " "  " "  " "
## 6  ( 1 ) " "           " "        " "  " "  " "  " "
## 7  ( 1 ) " "           " "        " "  " "  " "  " "
## 8  ( 1 ) " "           " "        " "  " "  " "  " "
```

The output above shows the selected variables for each model (with asterisks).

```r
plot(regsubsets.out, scale = "adjr2", main = "Adjusted R^2")
```



This plot shows the features associated with the best models in terms of Adjusted R^2. Adjusted R^2 is on the y-axis and the included features for each model are colored in.

Building out best multiple linear regression model

Based on Adjusted R^2, let's build our 8 best multiple linear regression models to use for prediction.

```r
mod1 <- lm(price ~ bedrooms + bathrooms + sqft_living + waterfront +
                   view + grade + yr_built + lat,
           data = train.data)

mod2 <- lm(price ~ bedrooms + sqft_living + waterfront + view +
                   grade + yr_built + lat,
           data = train.data)

mod3 <- lm(price ~ bedrooms + sqft_living + waterfront +
                   view + grade + yr_built + lat,
```

```
            data = train.data)

mod4 <- lm(price ~ sqft_living + view + grade +
                   yr_built + lat,
            data = train.data)

mod5 <- lm(price ~ sqft_living + view + grade + lat,
            data = train.data)

mod6 <- lm(price ~ sqft_living + view + lat,
            data = train.data)

mod7 <- lm(price ~ sqft_living + lat,
            data = train.data)

mod8 <- lm(price ~ sqft_living,
            data = train.data)
```

Predictive Performance

Build a list of models to iterate over and make predictions.

```
mlr.mod.list <- list(mod1, mod2, mod3, mod4, mod5, mod6, mod7, mod8)
```

Also set up an empty storage bin for keeping track of MSE (prediction error).

```
storage.bin <- data.frame(Mod = c("mod1", "mod2", "mod3", "mod4", "mod5", "mod6",
                                   "mod7", "mod8"),
                          MSE = numeric(length(mlr.mod.list)))
storage.bin
```

```
##    Mod MSE
## 1 mod1   0
## 2 mod2   0
## 3 mod3   0
## 4 mod4   0
## 5 mod5   0
## 6 mod6   0
## 7 mod7   0
## 8 mod8   0
```

Next, iterate over the model list. On each iteration: 1) Predict prices on all the rows in the test data
2) Calculate MSE (mean of all the squared differences between actual and predicted prices)
3) Store this MSE in storage bin

```
for(i in 1:nrow(storage.bin)){
        preds <- predict(mlr.mod.list[[i]], test.data)
        mse <- mean((preds-test.data$price)^2)
        storage.bin$MSE[i] <- mse
}
```

Find best model in terms of minimum prediction error

```
which.min(storage.bin$MSE)
```

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

```r
best.pred.mod <- which.min(storage.bin$MSE)
storage.bin[best.pred.mod,]
```

```
##    Mod          MSE
## 1 mod1 46677637166
```

This shows that mod1 performed the best in terms of prediction:

price ~ bedrooms + bathrooms + sqft_living + waterfront + view + grade + yr_built + lat

```r
summary(mod1)
```

```
##
## Call:
## lm(formula = price ~ bedrooms + bathrooms + sqft_living + waterfront +
##     view + grade + yr_built + lat, data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1169040   -98514   -11311    74053  4414195
##
## Coefficients:
##                    Estimate    Std. Error t value            Pr(>|t|)
## (Intercept)  -21296209.060    621360.126  -34.27 <0.0000000000000002 ***
## bedrooms        -29315.962      2214.083  -13.24 <0.0000000000000002 ***
## bathrooms        44120.724      3626.076   12.17 <0.0000000000000002 ***
## sqft_living        163.823         3.587   45.66 <0.0000000000000002 ***
## waterfront      591574.707     20245.826   29.22 <0.0000000000000002 ***
## view             51386.340      2461.426   20.88 <0.0000000000000002 ***
## grade           108561.174      2372.532   45.76 <0.0000000000000002 ***
## yr_built         -2873.173        70.211  -40.92 <0.0000000000000002 ***
## lat             553291.806     12147.144   45.55 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 199400 on 15121 degrees of freedom
## Multiple R-squared:  0.6882, Adjusted R-squared:  0.688
## F-statistic:  4171 on 8 and 15121 DF,  p-value: < 0.00000000000000022
```

Conclusion

This suggests that based on multiple linear regression, out of these 8 models the following is the best equation to predict housing prices:

```r
cf = as.character(coefficients(mod1))
cf.names = names(coefficients(mod1))
best.line.str <- paste("The best multiple regression line is described by: Price =", cf[1], "+ ", cf.na
      cf[4], "+",cf.names[5], "*", cf[5], "+",cf.names[6], "*", cf[6], "+",
      cf.names[7], "*", cf[7], "+",cf.names[8], "*", cf[8], "+",cf.names[9], "*", cf[9])
str_break(best.line.str)
```

```
## [1] "The best multiple regression line is described by: Price = -21296209.0604265 +  "
## [2] "bedrooms * -29315.9616520279 + bathrooms * 44120.7241753154 + sqft_living * 163."
## [3] "823459566203 + waterfront * 591574.70727402 + view * 51386.3395778446 + grade * "
## [4] "108561.173765076 + yr_built * -2873.17310384018 + lat * 553291.806436363"
```

Comparison to the other models

So far of the 3 regression models we've build, our folk wisdom model is the best at predicting.

K-Nearest Neighbors Regression

Now let's try a different version of regression to see if it improves our predictive performance.

Preprocessing

First standardize variables for better neighborhood calculations

```
train.data <- data.frame(apply(train.data, 2, scale))
test.data <- data.frame(apply(test.data, 2, scale))
```

Initial model comparison to multiple linear regression

To start, let's fit a KNN with K=5 then check test performance using training set. For this let's use the same predictors from our best multiple linear regression model to compare.

```
knnTest <- knn.reg(train = train.data[,c("bedrooms", "bathrooms", "sqft_living",
                                          "waterfront", "view", "grade", "yr_built", "lat")],
                   test = test.data[,c("bedrooms", "bathrooms", "sqft_living",
                                        "waterfront", "view", "grade", "yr_built", "lat")],
                   y = train.data$price, k = 5, algorithm = "brute")

knnTestMSE <- mean((test.data$price-knnTest$pred)^2)
paste("The KNN MSE is ", knnTestMSE)
```

```
## [1] "The KNN MSE is  0.223482273422157"
```

This MSE is very different from our MSE in the multiple linear regression models. That is because with KNN we scaled the data. Let's now re-train our best multiple linear regression model on the scaled data and compare the results to the KNN.

```
mod1b <- lm(price ~ bedrooms + bathrooms + sqft_living + waterfront +
              view + grade + yr_built + lat, data = train.data)
preds <- predict(mod1b, test.data)
mlr.mse <- mean((test.data$price-preds)^2)
paste("The best multiple linear regression MSE is", mlr.mse)
```

```
## [1] "The best multiple linear regression MSE is 0.305775771540126"
```

These results suggests that our KNN model obtains better predictive performance than our best multiple linear regression.

Model parameter tuning

Now let's find the best K parameter. K could be 2, 3, 4, 5, 6, 7, 8, etc.

So, let's first set up an empty storage bin to iterate over.

```
k.bin <- data.frame(K_val = 2:10, MSE = numeric(length(2:10)))
k.bin
```

```
##   K_val MSE
## 1     2   0
## 2     3   0
## 3     4   0
## 4     5   0
## 5     6   0
## 6     7   0
## 7     8   0
## 8     9   0
## 9    10   0
```

We'll iterate over each row in this bin, building a model with the respective K parameter, and then making predictions on the test data. We'll store the prediction error (MSE) in the appropriate column.

```r
for (i in 1:nrow(k.bin)){
        knnTest <- knn.reg(train = train.data[,c("bedrooms", "bathrooms", "sqft_living",
                                                  "waterfront", "view", "grade", "yr_built", "lat")],
                           test = test.data[,c("bedrooms", "bathrooms", "sqft_living",
                                                "waterfront", "view", "grade", "yr_built", "lat")],
                           y = train.data$price, k = k.bin$K_val[i], algorithm = "brute")

        testMSE <- mean((test.data$price-knnTest$pred)^2)
        k.bin$MSE[i] <- testMSE
}
```

Predictive Performance

```r
best.k.row <- which.min(k.bin$MSE)
k.bin[best.k.row,]
```

```
##   K_val       MSE
## 7     8 0.2148577
```

This suggests a K-value of 8 is the best parameter to use and that this model is giving us marked improvement over our multiple linear regression model.

Ridge Regression

Now let's try ridge regression to see if we can improve performance any more.

Preprocessing

Note, for glmnet, the cross validation does K-Fold CV.

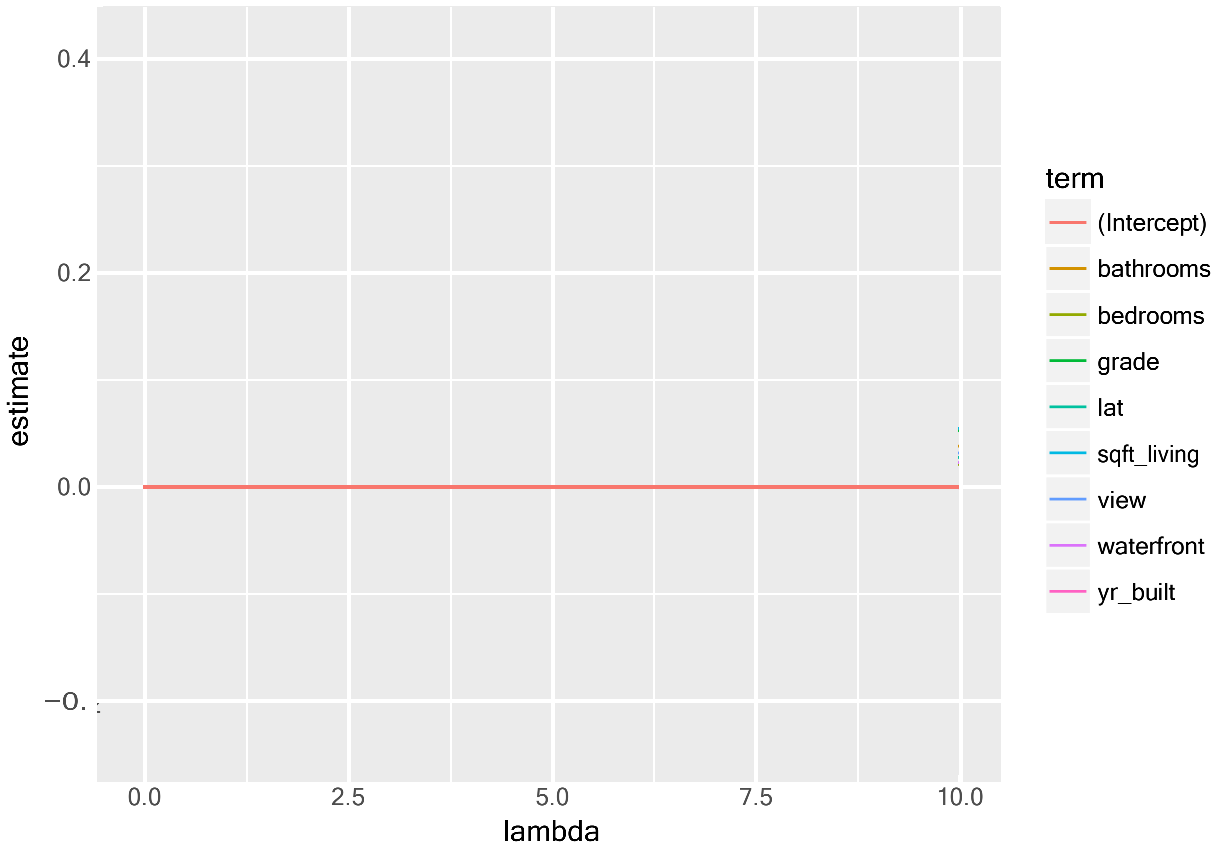
Model parameter validation

First build 4 models, each with a different lambda value. Lambda ranges from .001 to 10 to give good coverage of possible values.

```r
my_ridge_mods <- glmnet(x = as.matrix(train.data[,c("bedrooms", "bathrooms", "sqft_living",
                                                     "waterfront", "view", "grade", "yr_built", "lat")]), y
                        alpha = 0, lambda = c(.001, .1, 1, 10))
```

The following plot shows how the regression coefficients get penalized across values of lambda. You can see that as lambda increases, the penalty on each predictor increases, and the respective values of each coefficient approach zero.

```
tidy_ridge_mods <- tidy(my_ridge_mods)
ggplot(tidy_ridge_mods, aes(lambda, estimate, color = term)) + geom_line()
```



Next let's do 5-fold cross validation on the models. We'll do an order of magnitude approach so our lambda values cover a wide range of possibilities. Once we determine which lambda values are best in terms of predictive performance, we can then fine-tune within that range.

```
my_ridge_mods.cv <- cv.glmnet(x = as.matrix(train.data[,c("bedrooms", "bathrooms", "sqft_living","water
                              y = train.data$price,
                              alpha = 0,
                              lambda = c(.001, .1, 1, 10))
my_ridge_mods.cv$lambda
```

```
## [1] 10.000  1.000  0.100  0.001
```

```
my_ridge_mods.cv$cvm
```

```
## [1] 0.7751733 0.4097190 0.3178120 0.3141330
```

So .001 to .1 is the best in terms of prediction error.

Now we'll train many models within lambda range of .001 to .1 and see which one predicts the best.

```
my_best_ridge_mods <- cv.glmnet(x = as.matrix(train.data[,c("bedrooms", "bathrooms", "sqft_living","wat
                                alpha = 0, lambda = seq(0.001, .1, by = .01))
```

Find best lambda values

```r
which.min(my_best_ridge_mods$cvm)
```

```
## [1] 10
```

```r
my_best_lambda <- my_best_ridge_mods$lambda[which.min(my_best_ridge_mods$cvm)]
paste("My best lambda is", my_best_lambda)
```

```
## [1] "My best lambda is 0.001"
```

Build model with best equation

```r
newmyridge <- lm.ridge(price ~ bedrooms + bathrooms + sqft_living + waterfront + view + grade + yr_buil

cf = as.character(coefficients(newmyridge))
cf.names = names(coefficients(newmyridge))
best.ridge.str <- paste("The best ridge regression line is described by:", "Price =", cf[1], "+",
      cf.names[2], "*", cf[2], "+", cf.names[3], "*", cf[3], "+",cf.names[4], "*",
      cf[4], "+",cf.names[5], "*", cf[5], "+",cf.names[6], "*", cf[6], "+",
      cf.names[7], "*", cf[7], "+",cf.names[8], "*", cf[8], "+",cf.names[9], "*",
      cf[9])
str_break(best.ridge.str)
```

```
## [1] "The best ridge regression line is described by: Price = -0.0000000000000003199885"
## [2] "94399051 + bedrooms * -0.0768170756216568 + bathrooms * 0.0947217753623692 + sqf"
## [3] "t_living * 0.416460649360917 + waterfront * 0.14576506795167 + view * 0.10983449"
## [4] "913158 + grade * 0.353970773819112 + yr_built * -0.236447868464709 + lat * 0.215"
## [5] "006242795985"
```

Predictive Performance

Now that we have the best lambda, plug in to our model and predict with it.

Note, there is no predict function for lm.ridge, so we must make predictions manually based on coefficient values.

```r
pred.ridge <- coef(newmyridge)[1] +
        coef(newmyridge)[2]*test.data[,"bedrooms"] + coef(newmyridge)[3]*test.data[,"bathrooms"] +
        coef(newmyridge)[4]*test.data[,"sqft_living"] + coef(newmyridge)[5]*test.data[,"waterfront"] +
        coef(newmyridge)[6]*test.data[,"view"] + coef(newmyridge)[7]*test.data[,"grade"] +
        coef(newmyridge)[8]*test.data[,"yr_built"] + coef(newmyridge)[9]*test.data[,"lat"]
```

Find the prediction error for ridge.

```r
final.ridge.mse <- mean((pred.ridge - test.data$price)^2)
final.ridge.mse
```

```
## [1] 0.3057758
```

Final comparison of all models

```r
paste("The multiple linear regression MSE is ", mlr.mse)
```

```
## [1] "The multiple linear regression MSE is  0.305775771540126"
```

```r
paste("The KNN MSE is ", knnTestMSE)
```

```
## [1] "The KNN MSE is  0.223482273422157"
```

```r
paste("The Ridge Regression MSE is ", final.ridge.mse)
```

```
## [1] "The Ridge Regression MSE is  0.305775772074856"
```

Overall, ridge and multiple linear regression perform at about the same level. Of noticable improvement was the K-Nearest Neighbors model, which had the lowest prediction error.