1. Drop Conditions: Only single story family houses with no pools will be included in this. This is similar to the one I previously did, however I've left more variables in this time.

Checking my split and training data, I can see it's successfully split into 30% and 70% sections.

```
> # Verify the split
> cat("Total observations:", nrow(ames_data), "\n")
Total observations: 956
> cat("Training set observations:", nrow(train.df), "\n")
Training set observations: 672
> cat("Test set observations:", nrow(test.df), "\n")
Test set observations: 284
> cat("Sum of train and test observations:", nrow(train.df) + nrow(test.df), "\n")
Sum of train and test observations: 956
```

test.df	284 obs. of 85 variables
train.df	672 obs. of 85 variables

2.
```
 [1] "LotConfig"    "Condition1"   "GarageType"   "GarageCond"   "PavedDrive"   "ExterQual"
 [7] "BsmtQual"     "LotArea"      "OverallQual"  "YearBuilt"    "MasVnrArea"   "BsmtHalfBath"
[13] "FullBath"     "HalfBath"     "TotRmsAbvGrd" "Fireplaces"   "GarageCars"   "MoSold"
[19] "YrSold"       "TotalSqftCalc" "QualityIndex" "SalePrice"
```

I started my process with about 3 times as many variables here, but quickly realized why that was a problem. Having to OneHotEncode that many categorical variables created dataframes in the tens of thousands in columns, and was incredibly unrealistic to do. On top of that, most of the information inside was only slight variations of information that already existed. Square footage of the basement, total square footage, square footage of living space, these all accomplished the same thing and were highly correlated with one another. My decision boiled down to exposure of just one element per category (meaning, only one for square footage, one for quality of any category such as the kitchen, basement, etc.). Also, having progressed further into the assignment, I quickly saw which variables were going to cause issues and removed them since their predictive power was mediocre at best and they were causing issues with multicollinearity.

For the junk model, I simply added all the variables to it without going through a cleaning step so I could get a baseline on how much they have improved.

One of the first problems I ran into cleaning this was that some categorical variables contained NA variables, so when using lm() it would break the code and say there was multi-collinearity as essentially there were columns that were 100% correlated.

I naturally assume this meant there was some variable that was named differently, but contained exactly the same values, however it was the other problem where one Condition based variable had those NA values, so by dropping that categorical variable it fixed my problem.

```
> # Compute VIF for the forward selection model
> vif_forward <- vif(forward.model)
> print(sort(vif_forward, decreasing = TRUE))
 [1] 8.837178 8.245805 8.000000 6.426313 5.112705 5.000000 4.000000 4.000000 3.064072 3.000000
[11] 2.972739 2.810336 2.587489 2.374498 2.261129 2.233215 1.791363 1.750449 1.740858 1.692725
[21] 1.676406 1.608567 1.494395 1.419641 1.394935 1.363516 1.338418 1.319416 1.301755 1.191487
[31] 1.135243 1.090328 1.065478 1.042484 1.033443 1.000000 1.000000 1.000000 1.000000 1.000000
[41] 1.000000 1.000000 1.000000 1.000000 1.000000
>
> # Compute VIF for the backward elimination model
> vif_backward <- vif(backward.model)
> print(sort(vif_backward, decreasing = TRUE))
 [1] 8.837178 8.245805 8.000000 6.426313 5.112705 5.000000 4.000000 4.000000 3.064072 3.000000
[11] 2.972739 2.810336 2.587489 2.374498 2.261129 2.233215 1.791363 1.750449 1.740858 1.692725
[21] 1.676406 1.608567 1.494395 1.419641 1.394935 1.363516 1.338418 1.319416 1.301755 1.191487
[31] 1.135243 1.090328 1.065478 1.042484 1.033443 1.000000 1.000000 1.000000 1.000000 1.000000
[41] 1.000000 1.000000 1.000000 1.000000 1.000000
>
> # Compute VIF for the stepwise selection model
> vif_stepwise <- vif(stepwise.model)
> print(sort(vif_stepwise, decreasing = TRUE))
 [1] 8.837178 8.245805 8.000000 6.426313 5.112705 5.000000 4.000000 4.000000 3.064072 3.000000
[11] 2.972739 2.810336 2.587489 2.374498 2.261129 2.233215 1.791363 1.750449 1.740858 1.692725
[21] 1.676406 1.608567 1.494395 1.419641 1.394935 1.363516 1.338418 1.319416 1.301755 1.191487
[31] 1.135243 1.090328 1.065478 1.042484 1.033443 1.000000 1.000000 1.000000 1.000000 1.000000
[41] 1.000000 1.000000 1.000000 1.000000 1.000000
>
```

The VIF values look decent with only 8.84 being the highest one and many below 2. The most obvious thing I notice right off the bat is that the forward, backward, and step propagation have all selected the exact same model. On one hand, I think this is a good thing as no matter what method was used to generate these models, they all agreed this was the optimal way.

On the other hand, it also shows that employing different techniques has no real added benefits.

The thing that concerns me is that there are so many values in here that contain 1.0, which in reality, is impossible. However, I'm not sure how to remove these and if I do, how much they will change the underlying model.

Later when viewing the models, I will remove those that are offsetting this prediction.

```
[1] "Forward Selection Errors:"
> print(errors_forward)
$ARS
[1] 0.9108249

$AIC
[1] 15649.85

$BIC
[1] 15812.21

$MSE
[1] 1612241978

$MAE
[1] 18835

> print("Backward Elimination Errors:")
[1] "Backward Elimination Errors:"
> print(errors_backward)
$ARS
```

```
[1] 0.9108249

$AIC
[1] 15649.85

$BIC
[1] 15812.21

$MSE
[1] 1612241978

$MAE
[1] 18835

> print("Stepwise Selection Errors:")
[1] "Stepwise Selection Errors:"
> print(errors_stepwise)
$ARS
[1] 0.9108249

$AIC
[1] 15649.85

$BIC
[1] 15812.21

$MSE
[1] 1612241978

$MAE
[1] 18835

> print("Arbitrary 'Junk' Model Errors:")
[1] "Arbitrary 'Junk' Model Errors:"
> print(errors_junk)
$ARS
[1] 0

$AIC
[1] 17241.12

$BIC
[1] 17250.14

$MSE
[1] 5911015002

$MAE
[1] 61113.66
```

The ARS of the three model at 0.91 indicates a high proportion of variance explained by each model.

AIC of 15649.85 suggests that these models balance model complexity and fit effectively.

BIC of 15812 supports the previously mentioned with a similar balance in terms of model selection.

In short, this decided upon model outperforms the baseline junk model.

I cannot comment on which models rank the best since the method selected the same model, but had it produced differently, we may be able to rank them based off of the ARS, AIC, BIC, MSE, or MAE.

3.

In the previous photo, although I can't comment if the sheer number of MSE and MAE are poor, one thing is for certain. The other three models, which are the same, are considerably better than the "junk" model.

The best model of course based on MSE and MAE was the only model produced.

```
> cat("Forward Selection Model In-Sample R-squared: ", summary(forward.model)$r.squared, "\n")
Forward Selection Model In-Sample R-squared:  0.9153434
> cat("Forward Selection Model Out-of-Sample R-squared: ", R2(forward.test, test.df$SalePrice), "\n")
Forward Selection Model Out-of-Sample R-squared:  0.7873076
>
```

Unsurprisingly, the out of sample test was not nearly as accurate as the in sample. This is why it's so critical to constantly find the best model because applying the model on unseen information is always going to be different, and you have to figure out at what level is it acceptable, or how much more data will you need to see its true accuracy rate.

4.

Business sense is more difficult. MSE or MAE may be sufficient depending on the risk involved. If we have a model that can predict the price of a house and we find that it's well below the listing price, we have a deal. The issue is, there are so many factors one has to take into account before pulling the trigger on the purchase of a house that a drop from 91% to 78% is concerning. Especially when purchases have to be based off of info alone and one can't go and see the property themselves.

```
Forward Selection Test Prediction Grades:
> print(normalize_table(test_prediction_grade_forward))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.6021127             0.1936620             0.1126761            0.0915493
>
> cat("\nBackward Elimination Test Prediction Grades:\n")

Backward Elimination Test Prediction Grades:
> print(normalize_table(test_prediction_grade_backward))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.6021127             0.1936620             0.1126761            0.0915493
>
> cat("\nStepwise Selection Test Prediction Grades:\n")

Stepwise Selection Test Prediction Grades:
> print(normalize_table(test_prediction_grade_stepwise))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.6021127             0.1936620             0.1126761            0.0915493
>
> cat("\n'Junk' Model Test Prediction Grades:\n")

'Junk' Model Test Prediction Grades:
> print(normalize_table(test_prediction_grade_junk))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
           0.13380282            0.06690141            0.17605634           0.62323944
```

Here we have the model that predicts the different grades, and what percentage fall within investment grades.

No surprise, the top three models are the same, with 60% falling within Grade 1, 19% in grade 2, with 79% being in the top half total. Compared to junk's 13% this is much better.

This approach gives more weight to certain predictions. I feel this is an accurate representation of guesses, however I would not hold them up to the same scrutiny of Fannie Mae and Freddie Mac's models for underwriting. Although I've learned a lot, I would not trust these models for any commercial purposes.

5.

Already, from earlier, I had a huge issue with multi-collinearity. As you suggested, I removed a lot variables that were just repeating already known information. After calculating the overall square footage, most others were distracting and so I removed them. Certain quality variables as well such as multiple exterior conditions were repeating the same information in different ways. I'm not sure if that was generated by different people, but if they did not line up precisely, it could also affect the model.

The first round to see what is truly needed is seeing the final summary and removing any variables that don't hit the p-value requirement. In R's case, anything that isn't starred.

```
Call:
lm(formula = SalePrice ~ ., data = train.clean)

Residuals:
    Min      1Q  Median      3Q     Max
-135198  -13997     228   12824  137705

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       -1.564e+05  1.690e+06  -0.093 0.926312
LotConfigCulDSac   1.294e+04  6.242e+03   2.073 0.038569 *
LotConfigFR2      -1.031e+04  7.272e+03  -1.418 0.156811
LotConfigFR3      -3.761e+04  1.956e+04  -1.923 0.054978 .
LotConfigInside   -8.991e+02  2.765e+03  -0.325 0.745133
Condition1Feedr    2.000e+03  8.914e+03   0.224 0.822590
Condition1Norm     8.092e+03  8.061e+03   1.004 0.315859
Condition1PosA     2.701e+04  1.238e+04   2.181 0.029553 *
Condition1PosN     1.947e+04  1.166e+04   1.670 0.095517 .
Condition1RRAe    -1.207e+04  1.130e+04  -1.068 0.286049
Condition1RRAn    -7.869e+03  1.185e+04  -0.664 0.507008
Condition1RRNe    -4.871e+04  2.903e+04  -1.678 0.093887 .
Condition1RRNn    -2.464e+04  2.123e+04  -1.160 0.246407
GarageTypeAttchd   2.913e+04  1.425e+04   2.044 0.041397 *
GarageTypeBasment  2.396e+04  1.832e+04   1.308 0.191320
GarageTypeBuiltIn  3.207e+04  3.072e+04   1.044 0.296850
GarageTypeCarPort  4.451e+04  2.204e+04   2.019 0.043896 *
GarageTypeDetchd   3.466e+04  1.446e+04   2.397 0.016816 *
GarageCondFa      -2.922e+03  1.764e+04  -0.166 0.868455
GarageCondGd      -1.073e+04  2.512e+04  -0.427 0.669408
GarageCondPo      -2.523e+04  2.655e+04  -0.951 0.342195
GarageCondTA      -1.621e+03  1.585e+04  -0.102 0.918560
PavedDriveP       -1.538e+04  1.004e+04  -1.533 0.125878
PavedDriveY       -1.037e+04  5.939e+03  -1.747 0.081173 .
ExterQualFa       -8.278e+04  1.653e+04  -5.009 7.14e-07 ***
ExterQualGd       -5.036e+04  5.932e+03  -8.490  < 2e-16 ***
ExterQualTA       -6.632e+04  7.275e+03  -9.116  < 2e-16 ***
```

```
BsmtQualFa      -3.531e+04  9.630e+03  -3.667 0.000266 ***
BsmtQualGd      -2.960e+04  4.585e+03  -6.456 2.16e-10 ***
BsmtQualNA      -3.526e+04  8.788e+03  -4.013 6.73e-05 ***
BsmtQualTA      -3.444e+04  6.197e+03  -5.558 4.05e-08 ***
LotArea          1.506e+00  2.649e-01   5.685 2.01e-08 ***
OverallQual      1.093e+04  2.025e+03   5.397 9.62e-08 ***
YearBuilt        3.964e+02  9.342e+01   4.243 2.54e-05 ***
MasVnrArea       3.307e+01  7.897e+00   4.187 3.23e-05 ***
BsmtHalfBath    -4.681e+03  4.747e+03  -0.986 0.324529
FullBath        -2.245e+02  3.931e+03  -0.057 0.954472
HalfBath         5.156e+03  3.274e+03   1.574 0.115880
TotRmsAbvGrd     3.098e+03  1.351e+03   2.292 0.022211 *
Fireplaces       5.322e+03  2.157e+03   2.467 0.013892 *
GarageCars       1.116e+04  2.449e+03   4.557 6.23e-06 ***
MoSold          -1.129e+02  4.061e+02  -0.278 0.781082
YrSold          -2.945e+02  8.346e+02  -0.353 0.724265
TotalSqftCalc    3.217e+01  2.203e+00  14.604  < 2e-16 ***
QualityIndex     6.611e+02  2.175e+02   3.039 0.002470 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 26950 on 627 degrees of freedom
Multiple R-squared:  0.9162,    Adjusted R-squared:  0.9103
F-statistic: 155.7 on 44 and 627 DF,  p-value: < 2.2e-16
```

This part is a bit difficult because the encoded variables contain some that are useful, and some that are not, and if I drop any of the categorical ones, I have to drop them all, unless I wish to only remove those with the problematic variables.

I’m not sure, but I think you can get skewed data if you pick and chose what works, so I removed the following.

Unsurprisingly, all of the categorical ones had almost no impact on the model except for those that had detached garages. The condition of the garage also seemed to not matter.

I also removed the amount of baths, month sold, and year sold. To me this was surprising how little they added to the model because anecdotally, the amount of bathrooms was a huge selling point, but even though it may move a house faster, it doesn’t necessarily mean it will sell for more money.

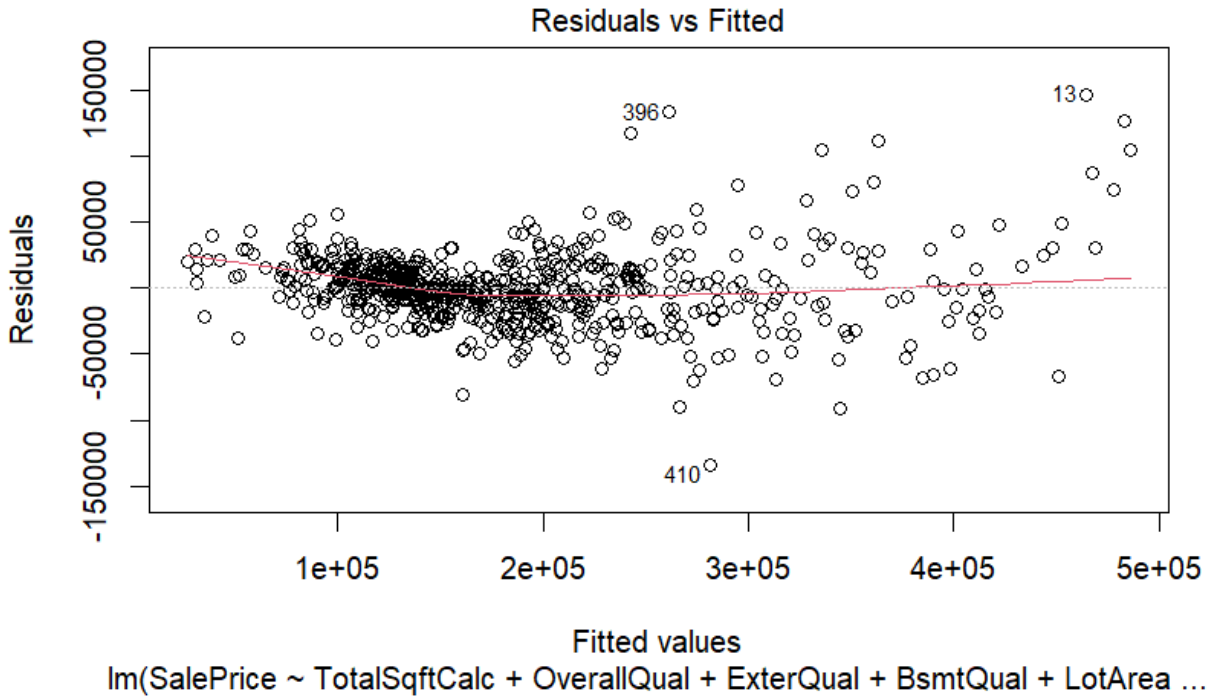
```
Call:
lm(formula = SalePrice ~ TotalSqftCalc + OverallQual + ExterQual +
    BsmtQual + LotArea + GarageCars + MasVnrArea + YearBuilt +
    QualityIndex + Fireplaces, data = train.clean)

Residuals:
    Min      1Q  Median      3Q     Max
-134533  -13425    -526   13717  146705

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   -5.569e+05  1.650e+05  -3.376 0.000780 ***
TotalSqftCalc  3.334e+01  2.099e+00  15.882  < 2e-16 ***
OverallQual    1.099e+04  1.965e+03   5.595 3.25e-08 ***
ExterQualFa   -8.871e+04  1.525e+04  -5.818 9.30e-09 ***
ExterQualGd   -5.478e+04  5.884e+03  -9.310  < 2e-16 ***
ExterQualTA   -7.194e+04  7.151e+03 -10.061  < 2e-16 ***
BsmtQualFa    -3.557e+04  9.514e+03  -3.739 0.000201 ***
BsmtQualGd    -3.089e+04  4.504e+03  -6.860 1.60e-11 ***
BsmtQualNA    -3.413e+04  8.655e+03  -3.943 8.90e-05 ***
BsmtQualTA    -3.738e+04  6.029e+03  -6.200 1.00e-09 ***
LotArea        1.573e+00  2.512e-01   6.262 6.88e-10 ***
GarageCars     1.192e+04  2.301e+03   5.178 2.99e-07 ***
MasVnrArea     3.627e+01  7.835e+00   4.630 4.42e-06 ***
YearBuilt      3.221e+02  8.432e+01   3.820 0.000146 ***
QualityIndex   6.938e+02  2.137e+02   3.247 0.001226 **
Fireplaces     4.624e+03  2.113e+03   2.188 0.028989 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 27440 on 656 degrees of freedom
Multiple R-squared:  0.909,     Adjusted R-squared:  0.9069
F-statistic: 436.9 on 15 and 656 DF,  p-value: < 2.2e-16
```

Removing those affected has increased my F-statistic from 155 to 436, indicating a much more accurate model.

```
> cat("Forward Selection Test Prediction Grades:\n")
Forward Selection Test Prediction Grades:
> print(normalize_table(test_prediction_grade_forward))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.61267606            0.16549296            0.14788732            0.07394366
>
> cat("\nBackward Elimination Test Prediction Grades:\n")

Backward Elimination Test Prediction Grades:
> print(normalize_table(test_prediction_grade_backward))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.61267606            0.16549296            0.14788732            0.07394366
>
> cat("\nStepwise Selection Test Prediction Grades:\n")

Stepwise Selection Test Prediction Grades:
> print(normalize_table(test_prediction_grade_stepwise))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.61267606            0.16549296            0.14788732            0.07394366
>
> cat("\n'Junk' Model Test Prediction Grades:\n")

'Junk' Model Test Prediction Grades:
> print(normalize_table(test_prediction_grade_junk))
grades
   Grade 1: [0, 0.10] Grade 2: (0.10, 0.15] Grade 3: (0.15, 0.25]      Grade 4: (0.25+]
            0.13380282            0.06690141            0.17605634            0.62323944
>
```

Unfortunately, the model has only improved slightly for selection criteria for the Freddie Mac model, meaning that it may not matter as much or the amount the other variables were detracting from didn't mean as much, so although it was much more accurate, it didn't improve in the areas we needed to.

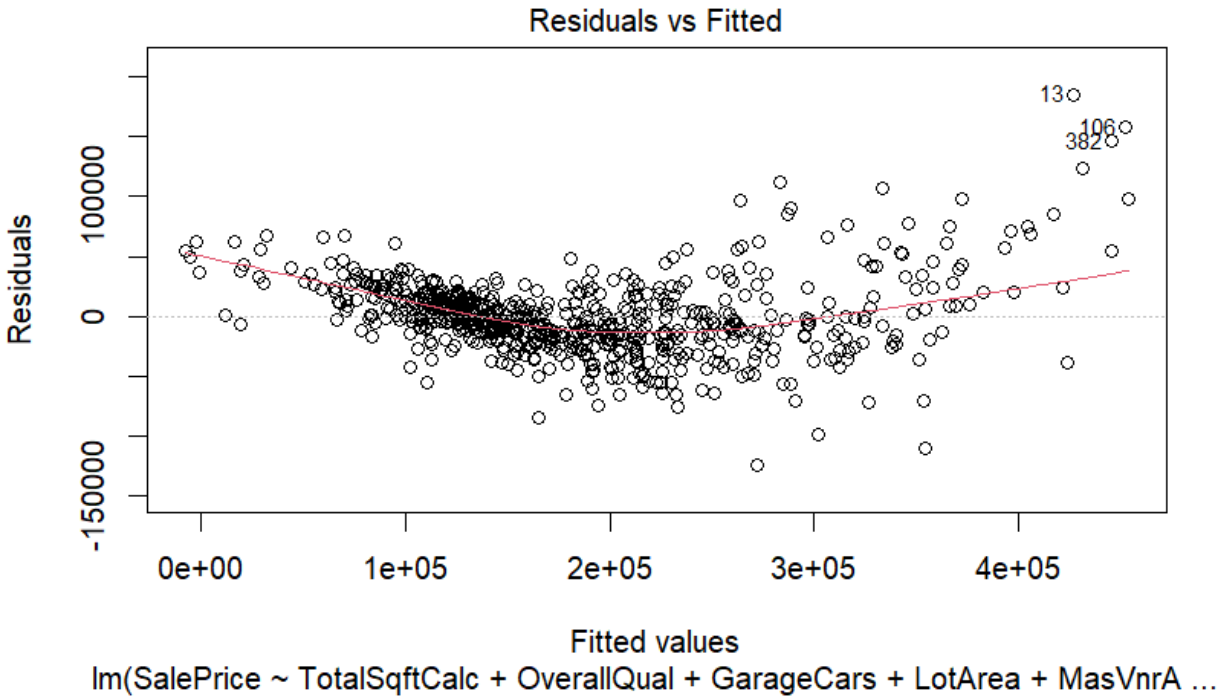
Since there are no more categorical variables, we do not have an ANCOVA model. Instead, it is just simply linear regression.



Plotting the residuals vs fitted is problematic at first glance.

The residuals are not evenly distributed across the range of fitted values. There is a large presence of outliers, which is likely affecting the spread.

Attempting to apply log to numeric values helps a bit.



The scatter is more evenly distributed, which is a step in the right direction, but there is still a long way to go.

6.

The largest challenges by far are.

a. Is the data from which you're working accurate enough to make predictions?
b. Is the data in a format that is beneficial, or does it need to be transformed to be of any use?
c. Is the problem you're trying to solve related to the data being presented? It doesn't do you any good to have data that seems like it's related to houses, but in reality it has no prediction power. Sometimes just a large number means more expensive. For example, if someone had 100k baseball cards that owned the house, there's a good chance that person owns a more expensive house, therefore there's correlation. That does not predict for other matters. That's an extreme example, but I could see lots of data sets that seem to have related items, but have none whatsoever.

If I were to start a new data analysis, I would not move onto the next step until every form of transformation had occurred, and then make sure that each step is fully covered. This way, you don't have to get to the end of the process and go back to make corrections. I'm sure there is a more standardized approach that eliminates heteroskedasticity, which I will find for future assignments.

I agree that parsimony is the way to go. Already, just by doing this assignment, I started with the notion that adding more data must make a better model, but by the end I had dropped 80% of what I started with. I think it's natural to make simple models to start, and then specialize from there.

I vote for a simpler but more interpretable model.