FourLinearModels

9/15/2021

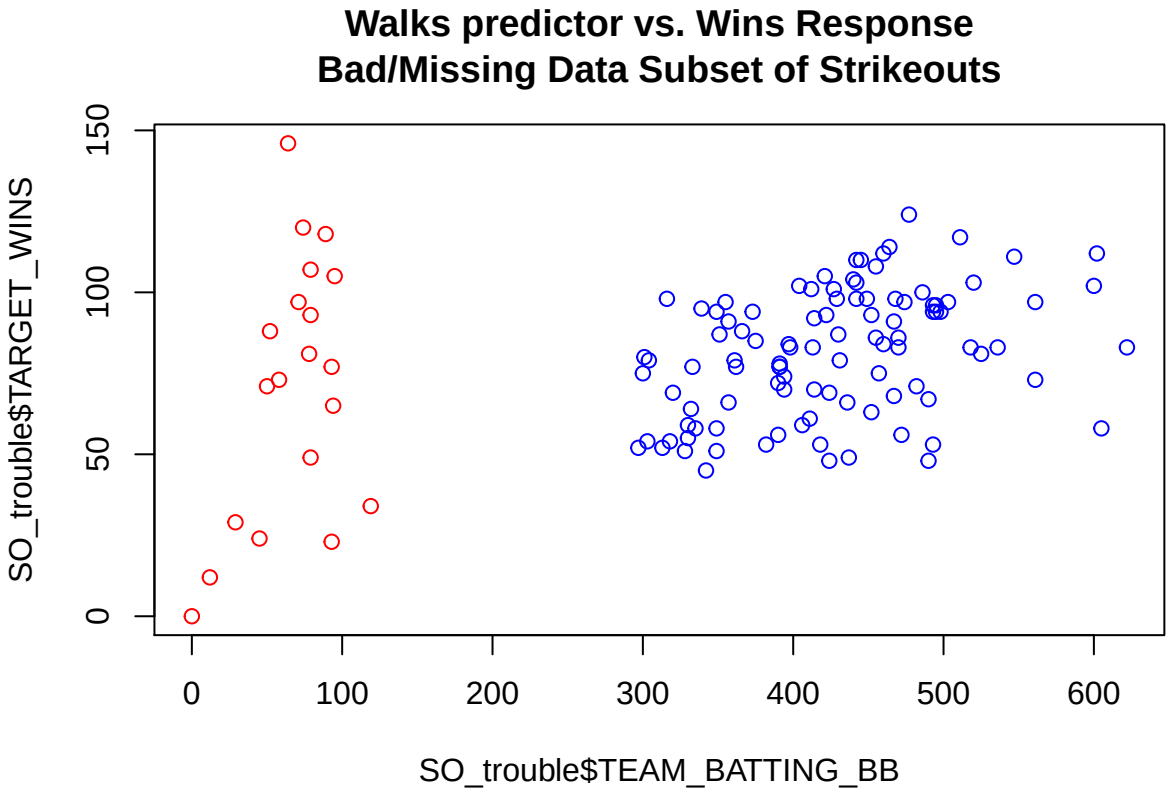
Dealing with garbage subsets of the data by training a separate linear model on each subset

Note that this approach is akin to what we talked about last week – Using clustering techniques to create features that indicate which hypothetical category each row belongs to (for example, maybe data from 1871 to 1901 are inherently different than modern data, in terms of how they correlate to the target outcome). The difference with the subset-models approach shown below is that the subsets are chosen by hand here, but more importantly, they each have their own entire set of model parameters, not just one single categorical parameter which can't make up for the other parameters' bias on its own.

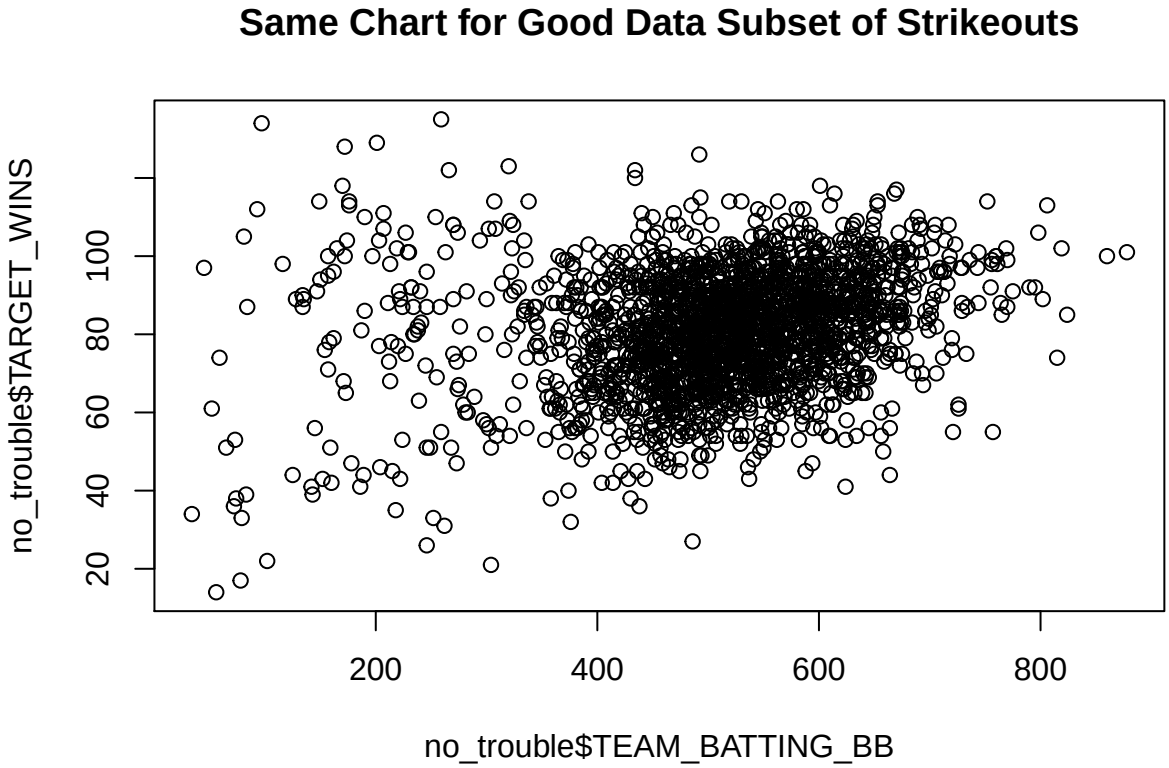
Why might this approach be better than imputing missing values, coaxing outliers back to the IQR, entirely ignoring rows that have undesired garbage data, or other techniques that might allow us to fit one linear model well to the whole dataset?

Instead of sanitizing the garbage as if to correct it, let's inspect whether the inconsistencies in the bad data can give us clues as to what their outcome variable is. It's entirely possible that missing values in a certain column, for example, represent rows from a certain era of data collection (or maybe simply repeated and predictable errors in data gathering). Perhaps in that era, strikeouts rates were more related to the response (target) variable, compared to data from other eras. In that case, we'd want a model that learned a different parameter for the strikeout features, compared to the same parameter in models fit to other eras of data. The alternative is to pretend that those strikeout feature values should be more normal, which may be equivalent to throwing out good information.

For example: What happens if we split the data into 2 subsets: One set has NA's or 0's for its strikeout columns, while the other set has positive values.



A linear model fit to the blue data (NA's for Strikeouts) will find a nice, small, expected positive correlation between more team batting BB's and more team wins. Furthermore the blue cloud looks like real, clean data – The number of wins is realistic, between 50 and 125 – whereas the red data (zeros for Strikeouts, which is already garbage) has garbage for target responses (zero wins? 150 wins??). The number of walks (x-axis) is also much smaller for all red than for all blue. Why? Who knows? Maybe someone accidentally extrapolated an entire 162-game season from one single game's stats (0 TARGET_WINS), in which they didn't keep track of walks and strikeouts. Zeros all around. If that actually happened, we'd like to pick up on that pattern and have our model predict 0 the next time it comes across that combination of values. In other words, it sure would be nice to fit a different set of linear parameters onto the red data.



This subset of data, with positive strikeout values, shows the same general distribution of values as the previous plot, but it wouldn't be so obvious to train separate models on left and right groups, and if you did, you'd have to figure out where to split them. This group also has less extreme response values, so that any benefits coming from predicting outlier values are reduced for a model.

What's a good way to start subsetting the data?

We could first split our data into two subsets: One whose rows have no NA's in ANY of their feature values, and another that has at least one NA in each row.

We modify the features a bit and fit a linear model to the "cleaner" data:

```
##
## Call:
## lm(formula = TARGET_WINS ~ ., data = fullTrains)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -31.1493  -6.7213   0.1099   6.5797  28.3140
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)              2.129e+02  4.862e+02   0.438 0.661592
## TEAM_BATTING_2B          9.273e-02  1.024e-01   0.905 0.365555
## TEAM_BATTING_3B          2.843e-01  1.189e-01   2.391 0.016977 *
## TEAM_BATTING_HR         -3.212e-02  1.217e-01  -0.264 0.791883
## TEAM_BATTING_BB          7.473e-02  6.772e-02   1.104 0.270009
## TEAM_BATTING_SO          1.763e-02  3.202e-02   0.550 0.582150
## TEAM_BASERUN_SB          3.738e-02  2.484e-02   1.505 0.132618
```

```
## TEAM_BASERUN_CS           9.201e-02  5.197e-02   1.770 0.076958 .
## TEAM_BATTING_HBP          2.898e+00  1.195e+00   2.426 0.015429 *
## TEAM_PITCHING_HR          7.003e-02  1.182e-01   0.592 0.553718
## TEAM_PITCHING_BB         -6.927e-03  5.398e-02  -0.128 0.897920
## TEAM_PITCHING_SO         -3.005e-02  3.001e-02  -1.001 0.316973
## TEAM_FIELDING_E          -1.796e-01  5.039e-02  -3.565 0.000379 ***
## TEAM_FIELDING_DP          1.664e-01  1.460e-01   1.140 0.254543
## TEAM_BATTING_1B          -6.498e-02  8.745e-02  -0.743 0.457600
## TEAM_PITCHING_1B2B3B      6.140e-02  3.614e-02   1.699 0.089630 .
## HR2SOfor                  1.908e+04  2.736e+04   0.697 0.485823
## HR2SOvs                  -1.942e+04  2.740e+04  -0.709 0.478762
## log_TEAM_BATTING_2B      -4.130e+01  2.031e+01  -2.033 0.042244 *
## log_TEAM_BATTING_3B      -6.587e+00  3.194e+00  -2.063 0.039373 *
## log_TEAM_BATTING_HR       5.926e+02  3.143e+02   1.885 0.059620 .
## log_TEAM_BATTING_BB      -1.465e+03  6.197e+02  -2.364 0.018226 *
## log_TEAM_BATTING_SO       9.257e+02  6.843e+02   1.353 0.176383
## log_TEAM_BASERUN_SB      -1.215e-01  2.347e+00  -0.052 0.958722
## log_TEAM_BASERUN_CS      -3.598e+00  2.868e+00  -1.255 0.209852
## log_TEAM_PITCHING_HR     -6.064e+02  3.153e+02  -1.923 0.054750 .
## log_TEAM_PITCHING_BB      1.449e+03  6.205e+02   2.336 0.019685 *
## log_TEAM_PITCHING_SO     -9.060e+02  6.848e+02  -1.323 0.186119
## log_TEAM_FIELDING_E       2.174e+00  7.543e+00   0.288 0.773214
## log_TEAM_FIELDING_DP     -4.174e+01  2.236e+01  -1.867 0.062216 .
## log_TEAM_BATTING_1B       5.549e+01  1.389e+02   0.400 0.689547
## log_TEAM_PITCHING_1B2B3B -1.918e+01  1.397e+02  -0.137 0.890810
## log_HR2SOfor             -2.603e+04  3.366e+04  -0.773 0.439469
## log_HR2SOvs               2.661e+04  3.372e+04   0.789 0.430190
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.473 on 1144 degrees of freedom
## Multiple R-squared:  0.4634, Adjusted R-squared:  0.4479
## F-statistic: 29.93 on 33 and 1144 DF,  p-value: < 2.2e-16
```

Multiple R-squared: 0.4634

Adjusted R-squared: 0.4479

F-statistic: 29.93 on 33 and 1144 DF, p-value: $<$ 2.2e-16

That's not bad for the non-NA data. But we have to make predictions for data with NA's eventually.

Let's see what we get for NA's

How tough is this task compared to the cleaner model we just fit?

```
## [1] "The variance in response values for the missing data subset is  405"
```

```
## [1] "For complete data the variance was  163"
```

There's a lot more chance of making big errors on the missing values subset, based on that difference in response variance.

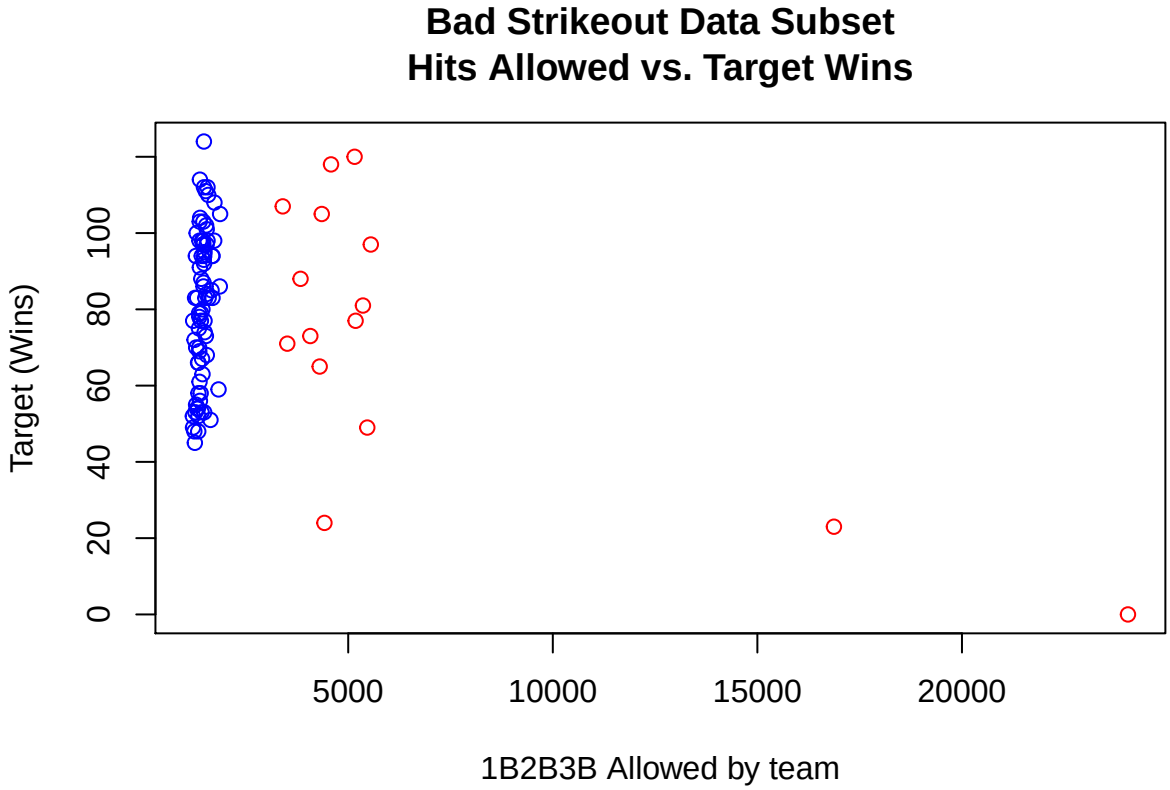
How to deal with each NA in that subset?

Some variables, like HBP, make it easy by all being missing, so we can remove them.

We saw with Strikeouts how merely splitting on one feature's missingness separated the remaining data into different groups. But it would take awhile to run the same procedure for all columns with missing values. Maybe we can go halfway by simply turning each such column into a binary variable indicating whether the value provided is NA or numeric. This may or may not be a standard method, but it does allow a linear model one more dimension to stretch into, should the missingness have some correlation with the target, in combination with other features.

Zero strikeouts and NA strikeouts

Within nested subsets now, we can engineer separate features for separate models, when we discover strange plots that weren't apparent before subsetting:



Again we see the NA data, in blue, looking separate, all crunched up on the left side with reasonable x and y values.
The red (zero SO) data has garbage predictor values and at least one garbage response (lower right corner).

So let's make a feature for the red model: An inverse of the x-value, to capture some of the red plot shape.

Train 3 models on the 3 NA subsets:

```
##
## Call:
## lm(formula = TARGET_WINS ~ ., data = naTrains)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -42.328 -10.797   1.091  10.539  43.721
##
```

```
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)            2.307e+01  1.385e+01   1.666 0.096364 .
## TEAM_BATTING_2B        5.917e-02  1.766e-02   3.351 0.000864 ***
## TEAM_BATTING_3B        1.838e-01  2.981e-02   6.166 1.42e-09 ***
## TEAM_BATTING_HR        1.077e-02  6.794e-02   0.158 0.874130
## TEAM_BATTING_BB        9.294e-03  1.276e-02   0.729 0.466609
## TEAM_BATTING_SO        9.534e-03  6.997e-03   1.363 0.173630
## TEAM_BASERUN_SB        2.792e+01  3.285e+00   8.499  < 2e-16 ***
## TEAM_BASERUN_CS       -2.860e+01  6.935e+00  -4.123 4.35e-05 ***
## TEAM_PITCHING_HR      -2.313e-02  5.532e-02  -0.418 0.676091
## TEAM_PITCHING_BB       1.660e-02  8.028e-03   2.068 0.039128 *
## TEAM_PITCHING_SO      -1.612e-03  1.463e-03  -1.102 0.270951
## TEAM_FIELDING_E       -4.984e-02  5.908e-03  -8.437 3.35e-16 ***
## TEAM_FIELDING_DP       1.563e+00  2.107e+00   0.742 0.458420
## TEAM_BATTING_1B        5.065e-02  7.695e-03   6.582 1.15e-10 ***
## TEAM_PITCHING_1B2B3B -8.993e-04  7.455e-04  -1.206 0.228239
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.35 on 513 degrees of freedom
## Multiple R-squared:  0.4019, Adjusted R-squared:  0.3856
## F-statistic: 24.62 on 14 and 513 DF,  p-value: < 2.2e-16
```

The bigger subset, which had strikeout data but was missing something else, had Multiple R-squared: 0.4019 Adjusted R-squared: 0.3856

Not bad, although not as good as the previous model, which had .45-.46 R-squared.

How about the NA Strikeout rows model?

```
##
## Call:
## lm(formula = TARGET_WINS ~ ., data = naSO)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.8125  -8.3190   0.9375   6.9986  19.7960
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -3.233e+03  3.133e+03  -1.032    0.306
## TEAM_BATTING_2B          -5.557e-01  6.747e-01  -0.824    0.413
## TEAM_BATTING_3B          -1.114e-01  8.457e-01  -0.132    0.896
## TEAM_BATTING_HR          -4.019e+00  5.388e+00  -0.746    0.459
## TEAM_BATTING_BB           1.498e-01  6.830e-01   0.219    0.827
## TEAM_PITCHING_HR          4.617e+00  4.856e+00   0.951    0.345
## TEAM_PITCHING_BB         -3.102e-02  5.308e-01  -0.058    0.954
## TEAM_FIELDING_E          -1.121e-01  3.325e-01  -0.337    0.737
## TEAM_BATTING_1B          -3.468e-01  7.552e-01  -0.459    0.648
## TEAM_PITCHING_1B2B3B     -2.733e-02  3.681e-01  -0.074    0.941
## log_TEAM_BATTING_2B       4.241e+01  7.537e+01   0.563    0.576
## log_TEAM_BATTING_3B      -1.844e+01  4.970e+01  -0.371    0.712
## log_TEAM_BATTING_HR       3.873e+01  1.179e+02   0.329    0.744
## log_TEAM_BATTING_BB       6.524e+02  1.180e+03   0.553    0.582
```

```
## log_TEAM_PITCHING_HR     -4.902e+01  1.125e+02  -0.436    0.665
## log_TEAM_PITCHING_BB     -6.931e+02  1.178e+03  -0.588    0.558
## log_TEAM_FIELDING_E      -3.840e+01  9.884e+01  -0.389    0.699
## log_TEAM_BATTING_1B      -1.257e+02  1.196e+03  -0.105    0.917
## log_TEAM_PITCHING_1B2B3B  7.001e+02  9.652e+02   0.725    0.471
##
## Residual standard error: 11.65 on 60 degrees of freedom
## Multiple R-squared:  0.732,  Adjusted R-squared:  0.6516
## F-statistic: 9.103 on 18 and 60 DF,  p-value: 3.116e-11
```

Higher R-squared with relatively many variables to fit relatively few datapoints. Lower F-stat.

And that questionable zero-SO subset model, fit to the bizarre, red points?

```
##
## Call:
## lm(formula = TARGET_WINS ~ ., data = noSO)
##
## Residuals:
## ALL 15 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (6 not defined because of singularities)
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               4.381e+03         NA      NA       NA
## TEAM_BATTING_2B           2.108e+00         NA      NA       NA
## TEAM_BATTING_3B           5.461e-01         NA      NA       NA
## TEAM_BATTING_HR          -3.567e+01         NA      NA       NA
## TEAM_BATTING_BB           1.085e+01         NA      NA       NA
## TEAM_PITCHING_HR          1.491e+01         NA      NA       NA
## TEAM_PITCHING_BB          2.880e+00         NA      NA       NA
## TEAM_FIELDING_E           7.915e-02         NA      NA       NA
## TEAM_BATTING_1B           7.875e-01         NA      NA       NA
## TEAM_PITCHING_1B2B3B     -1.715e-01         NA      NA       NA
## invHitsAllowed            1.239e+06         NA      NA       NA
## log_TEAM_BATTING_2B      -2.720e+02         NA      NA       NA
## log_TEAM_BATTING_3B       2.946e+01         NA      NA       NA
## log_TEAM_BATTING_HR      -2.205e+01         NA      NA       NA
## log_TEAM_BATTING_BB      -1.290e+03         NA      NA       NA
## log_TEAM_PITCHING_HR             NA         NA      NA       NA
## log_TEAM_PITCHING_BB             NA         NA      NA       NA
## log_TEAM_FIELDING_E              NA         NA      NA       NA
## log_TEAM_BATTING_1B              NA         NA      NA       NA
## log_TEAM_PITCHING_1B2B3B         NA         NA      NA       NA
## log_invHitsAllowed               NA         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 14 and 0 DF,  p-value: NA
```

Uh-oh... **There are more variables than rows (n > m) so the matrix is not singular**
If we get rid of the log features, it should learn something useful

Re-train model

```
##
```

```
## Call:
## lm(formula = TARGET_WINS ~ ., data = noSO)
##
## Residuals:
##      393     1350     1211      415      999     1813     1812      298
##  3.53920  1.80159  0.09939 -0.12887 -6.11503  0.52381  3.57633 -2.95802
##     1345     1824     1823      861      998     1349      860
## -8.24421  6.74429 -0.44100  2.24492  2.04495  1.75748 -4.44484
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -1.703e+02  1.236e+02  -1.378   0.2403
## TEAM_BATTING_2B      -9.322e-02  7.878e-02  -1.183   0.3022
## TEAM_BATTING_3B       1.620e-01  7.291e-02   2.222   0.0904 .
## TEAM_BATTING_HR      -4.242e+00  2.377e+00  -1.785   0.1489
## TEAM_BATTING_BB       6.175e-01  1.800e-01   3.430   0.0265 *
## TEAM_PITCHING_HR      2.148e+00  9.928e-01   2.163   0.0965 .
## TEAM_PITCHING_BB     -1.792e-02  1.506e-02  -1.190   0.2998
## TEAM_FIELDING_E      -4.400e-02  2.039e-02  -2.158   0.0971 .
## TEAM_BATTING_1B       7.825e-02  3.176e-02   2.464   0.0694 .
## TEAM_PITCHING_1B2B3B  7.764e-03  2.701e-03   2.875   0.0453 *
## invHitsAllowed        4.805e+05  2.145e+05   2.240   0.0887 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.428 on 4 degrees of freedom
## Multiple R-squared:  0.9879, Adjusted R-squared:  0.9575
## F-statistic: 32.53 on 10 and 4 DF,  p-value: 0.002143
```

Those are very high R-sq without superlow p-vals, so we probably gave the model too many features (10) to fit to 15 rows.
But there are very few validation examples either, so any somewhat close prediction should suffice for these few rows. And remember that this is the garbage data, so if the model picks up on patterns, that's more important than the coefficients making sense (they don't!).

About that Validation Data

With all these high R-squared on NA data, there may be concern that the validation data will produce some poor numbers.

See how much variation the trained model takes care of for the full (no NA's) validation subset:

```
## [1] "Proportion of response variance fitted by model (Approximate R-sq)"
```

```
## [1] "for predictions on the validation set with no NA's:  0.443731022738413"
```

Zero strikeouts and NA strikeouts

See how much the trained models take care of on these rough data rows.

Models reminder:

naMod1 = lm(TARGET_WINS ~ ., naTrains)

naMod2 = lm(TARGET_WINS ~ ., naSO)

naMod3 = lm(TARGET_WINS ~ ., noSO)

```
## [1] "Approximate R-sq for the validation set with NA's but with SO's:  0.418691813714385"
```

```
## [1] "Approximate R-sq for the validation set with NA's for SO's:  0.596750805481747"
```

```
## [1] "Approximate R-sq for the validation set with 0's for SO's:  0.693888115799142"
```

That's surprisingly good, considering how few data there were to train in that last batch, so the model picked up on something important.

```
## [1] "True target values first, then predicted ones."
```

```
## [1]  12  29  34 146  93
```

```
##       2233       2239       2015        299       2016
##  -0.951009 -16.605991  18.069781 100.092531  81.564058
```

Since predictions are below zero: We can get even better results (slightly) by clipping the minimum win predictions to be zero.
For example:

```
## [1] "Approximate R-sq for the (clipped prediction) validation set with 0's for SO's:  0.775435247928
```

(We can't clip anything useful in the other NA SO group,
but should remember to clip the evaluation predictions later.)

Let's see what the overall validation set MSE or R-squared is, since this should be the best indicator of how our final evaluation predictions will be for the HW.

```
## [1] "Approximate R-squared for the validation set:  0.475201138515239"
```

47.5% of the variance in the validation set was accounted for/fit by the 4-model approach.
That's very comparable to, maybe better than, the Training models,
and a good indicator of how these four models will end up doing on evaluation set predictions.

– We can mix the validation data back into the training data if we want, and refit 4 models to it all, before making evaluation predictions.

– We can also try pruning unimportant variables from each of the four models.

– We could look for other features that split usefully into NA, 0, and positive non-NA subsets, the way SO did.

– We could statistically compare the distribution of Training values to Evaluation values, to see if they even appear to come from the same distribution. After all, no point in fitting models that won't apply to the eval set. (At first glance, I thought they looked similar)