HW1

Business Analytics and Data Mining

Zachary Palmore

2/15/2021

Assignment 1

Purpose

Our purpose is to design a linear regression model that makes the best prediction about how many games a baseball team can win in a season. The best way to do that today is with the training data set provided. In this analysis we are going to explore performance statistics from an MLB team, evaluate their performance, and make predictions about their winnings.

Data Exploration

Background

```r
# Packages
library(tidyverse)
library(ggpubr)
library(kableExtra)
library(reshape2)
library(corrplot)
library(ggstatsplot)
library(MASS)
library(bestNormalize)
library(Metrics)
library(VGAM)
theme_set(theme_minimal())
```

```r
# Our training data
tdata <- read.csv(
"https://raw.githubusercontent.com/palmorezm/msds/main/621/HW1/moneyball-training-data.csv")
# Basic statistics from the set
total.obs <- count(tdata)
avg.wins <- mean(tdata$TARGET_WINS)
max.wins <- max(tdata$TARGET_WINS)
sd.wins <- sd(tdata$TARGET_WINS)
colnames <- colnames(tdata)
missing.values <- (sum(is.na(tdata)))
```

In this data set, there are 2276 observations of a major league baseball team from 1871 to 2006. These statistics have been adjusted to match the typical performance of a 162-game season and no team has won every game in a season. A sample of the data is provided for reference.

```
# create samples of table data
tdata.tbl.1 <- tdata[1:5,c(1:4, 17)]
tdata.tbl.2 <- tdata[1:5,5:8]
tdata.tbl.3 <- tdata[1:5,9:12]
tdata.tbl.4 <- tdata[1:5,13:16]
# display samples of table data
ttbl.1 <- kbl(tdata.tbl.1, booktabs = T, caption = "Training Data Fields 1 - 4 and 17") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
ttbl.2 <- kbl(tdata.tbl.2, booktabs = T, caption = "Training Data Fields 5 - 8") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
ttbl.3 <- kbl(tdata.tbl.3, booktabs = T, caption = "Training Data Fields 9 - 12") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
ttbl.4 <- kbl(tdata.tbl.4, booktabs = T, caption = "Training Data Fields 13 - 16") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
```

```
ttbl.1
```

Table 1: Training Data Fields 1 - 4 and 17

INDEX	TARGET_WINS	TEAM_BATTING_H	TEAM_BATTING_2B	TEAM_FIELDING_DP
1	39	1445	194	NA
2	70	1339	219	155
3	86	1377	232	153
4	70	1387	209	156
5	82	1297	186	168

```
ttbl.2
```

Table 2: Training Data Fields 5 - 8

TEAM_BATTING_3B	TEAM_BATTING_HR	TEAM_BATTING_BB	TEAM_BATTING_SO
39	13	143	842
22	190	685	1075
35	137	602	917
38	96	451	922
27	102	472	920

```
ttbl.3
```

Table 3: Training Data Fields 9 - 12

TEAM_BASERUN_SB	TEAM_BASERUN_CS	TEAM_BATTING_HBP	TEAM_PITCHING_H
NA	NA	NA	9364
37	28	NA	1347
46	27	NA	1377
43	30	NA	1396
49	39	NA	1297

```
ttbl.4
```

Table 4: Training Data Fields 13 - 16

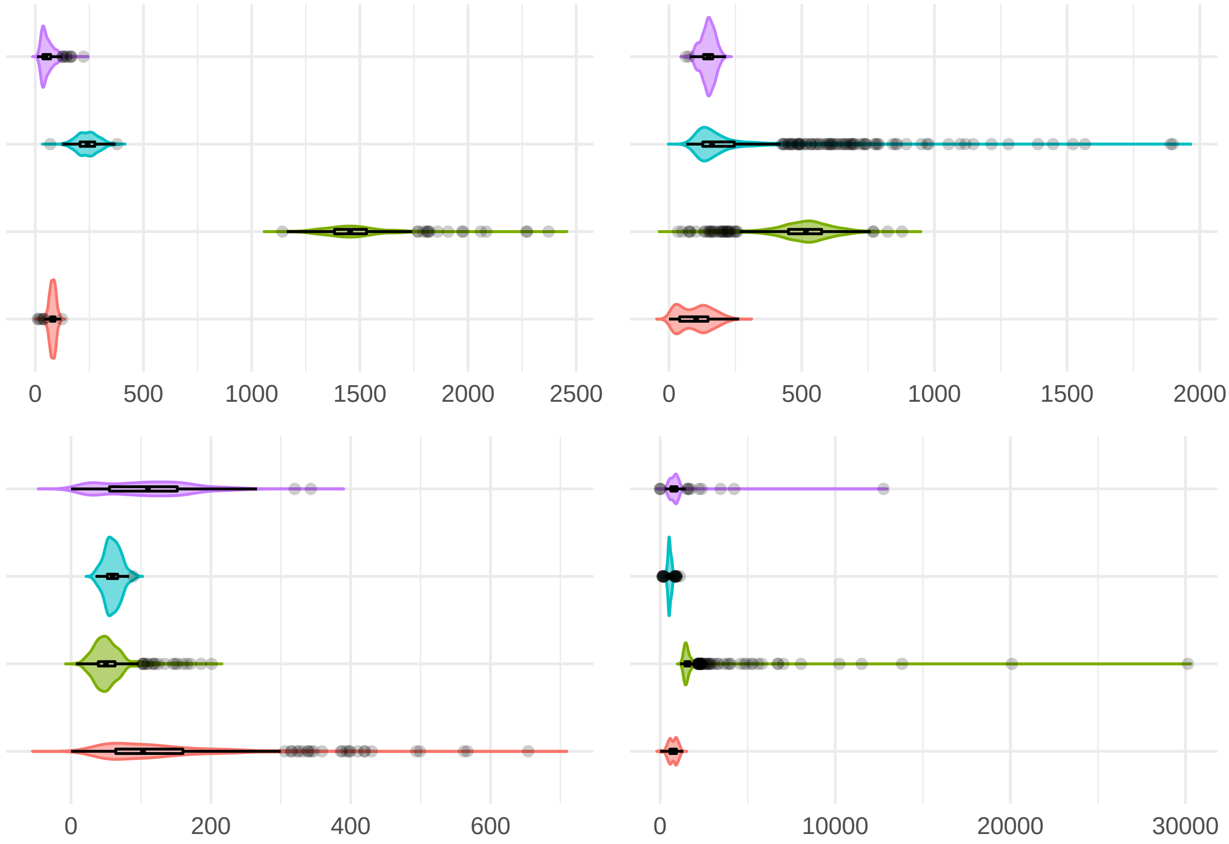
TEAM_PITCHING_HR	TEAM_PITCHING_BB	TEAM_PITCHING_SO	TEAM_FIELDING_E
84	927	5456	1011
191	689	1082	193
137	602	917	175
97	454	928	164
102	472	920	138

There are 17 variables and of course, our target variable is called TARGET_WINS representing the number of wins for this team. We check for outliers and overall patterns in the distributions of each variable. We exclude the index value since it is not a performance statistic, only a placeholder. This will tell us how to proceed in cleaning the data and give us a sense of how normal these data are. At this time, we do not need any labels as we are only looking at overall patterns.

```
# Function to calculate and set pointrange
xysdu <- function(x) {
   m <- mean(x)
   ymin <- m - sd(x)
   ymax <- m + sd(x)
   return(c(y = m, ymin = ymin, ymax = ymax))
}

# Organize data in buckets of similar distribution and scale
df <- tdata %>%
       melt() %>%
       filter(variable != c("INDEX"))
df1 <- tdata %>%
       melt() %>%
       filter(variable == c("TARGET_WINS",
                        "TEAM_BATTING_H",
                        "TEAM_BATTING_2B",
                        "TEAM_BATTING_3B"))
df2 <- tdata %>%
       melt() %>%
       filter(variable == c("TEAM_BATTING_HR",
                        "TEAM_BATTING_BB",
                        "TEAM_FIELDING_E",
                        "TEAM_FIELDING_DP"))
```

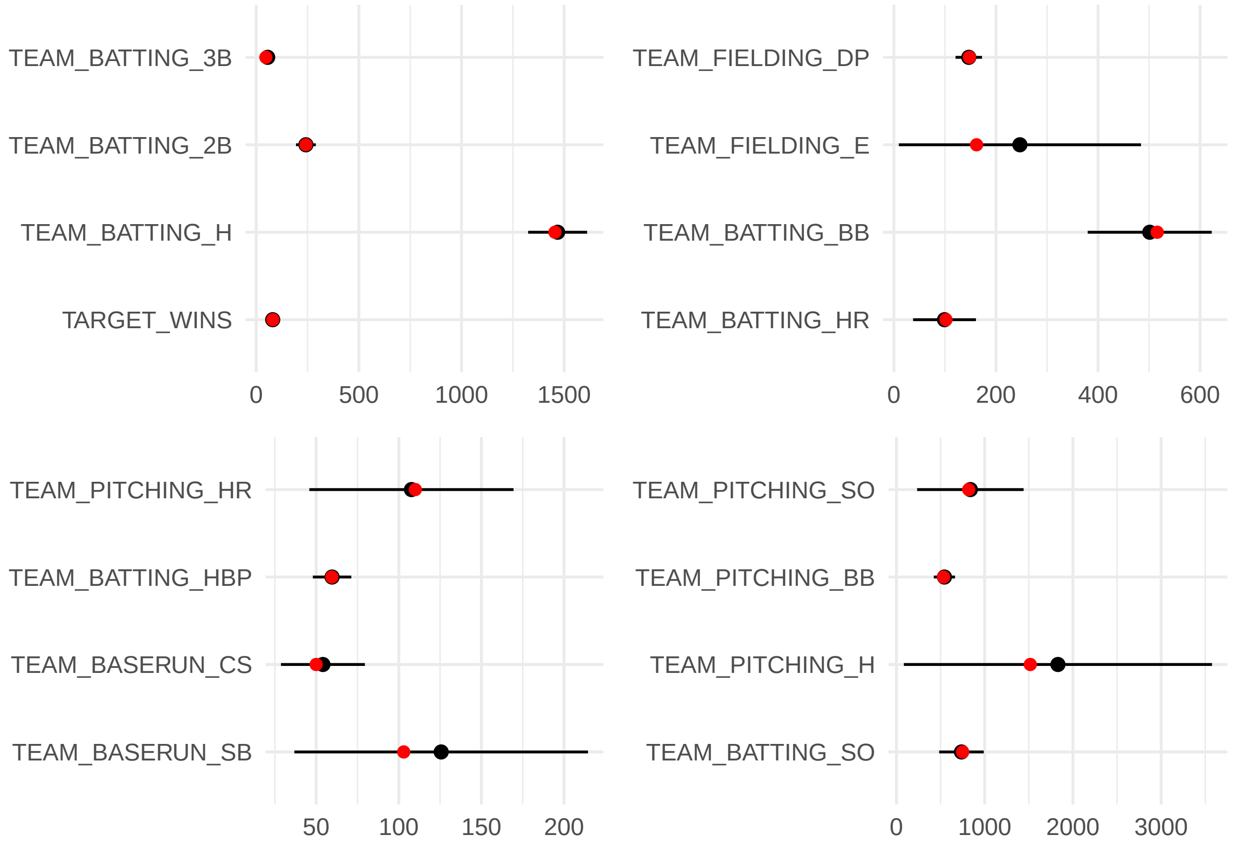
```r
df3 <- tdata %>%
       melt() %>%
       filter(variable == c("TEAM_BASERUN_CS",
                           "TEAM_BATTING_HBP",
                           "TEAM_BASERUN_SB",
                           "TEAM_PITCHING_HR"))
df4 <- tdata %>%
  melt() %>%
  filter(variable == c("TEAM_PITCHING_BB",
                       "TEAM_PITCHING_SO",
                       "TEAM_BATTING_SO",
                       "TEAM_PITCHING_H"))

# Violin plots
vplt.df1 <- ggplot(df1, aes(variable, value)) +
  geom_violin(trim = FALSE, scale="area",
              aes(color=variable, fill=variable, alpha=0.15, )) + coord_flip() +
  geom_boxplot(width=.05, shape=1, alpha=0.20, color="black", size=.5) +
  theme(legend.position = "None",
        axis.title.x = element_blank(),
        axis.title.y = element_blank(), axis.text.y = element_blank())

vplt.df2 <- ggplot(df2, aes(variable, value)) +
  geom_violin(trim = FALSE, scale="area",
              aes(color=variable, fill=variable, alpha=0.15, )) + coord_flip() +
  geom_boxplot(width=.05, shape=1, alpha=0.20, color="black", size=.5) +
  theme(legend.position = "None",
        axis.title.x = element_blank(),
        axis.title.y = element_blank(), axis.text.y = element_blank())

vplt.df3 <- ggplot(df3, aes(variable, value)) +
  geom_violin(trim = FALSE, scale="area",
              aes(color=variable, fill=variable, alpha=0.15, )) + coord_flip() +
  geom_boxplot(width=.05, shape=1, alpha=0.20, color="black", size=.5) +
  theme(legend.position = "None",
        axis.title.x = element_blank(),
        axis.title.y = element_blank(), axis.text.y = element_blank())

vplt.df4 <- ggplot(df4, aes(variable, value)) +
  geom_violin(trim = FALSE, scale="area",
              aes(color=variable, fill=variable, alpha=0.15, )) + coord_flip() +
  geom_boxplot(width=.05, shape=1, alpha=0.20, color="black", size=.5) +
  theme(legend.position = "None",
        axis.title.x = element_blank(),
        axis.title.y = element_blank(), axis.text.y = element_blank())


# Pointrange mean and median distribution
ptrangeplt.df1 <- ggplot(df1, aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color="black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.y = element_blank())
```

```
ptrangeplt.df2 <- ggplot(df2, aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color="black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.y = element_blank())

ptrangeplt.df3 <- ggplot(df3, aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color="black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.y = element_blank())

ptrangeplt.df4 <- ggplot(df4, aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color="black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None",
        axis.title.x = element_blank(), axis.title.y = element_blank())

garnge.vplts.area <- ggarrange(vplt.df1, vplt.df2, vplt.df3, vplt.df4)
garnge.ptrangeplts <- ggarrange(ptrangeplt.df1, ptrangeplt.df2, ptrangeplt.df3, ptrangeplt.df4)
garnge.vplts.area
```



This shows the distribution of each variable by kernel probability density indicating where each variable is most likely to occur. There are also black dots displayed within and along the colored probability density lines to indicate outliers. From this we can gather that there are many outliers and some are much larger and more frequent than others. We can also see that there are a few potential predictors that have little to no outliers and may be normal distributions. This is great but to know if they can help the model, we check the magnitude of outliers by exploiting the difference in robustness between median and mean.

```
garnge.ptrangeplts
```



When evaluating in this way, the variables' medians are plotted as red dots on a black line that is the point range calculated from the standard deviation of each distribution. In the center of that point range is a black dot that represents the mean of each variable. The farther away the red and black dots are from one another, the more influence outliers have on the variable.

We can see from the chart that there are a lot of outliers and the scale at which each occurs varies in magnitude from less than 1 to greater than 50 points. These outliers must be dealt with prior to the analysis if we are to use them to make predictions with. Otherwise, the models will be skewed and the results will not be as accurate. For now, we continue exploring the data.

Hypothesis

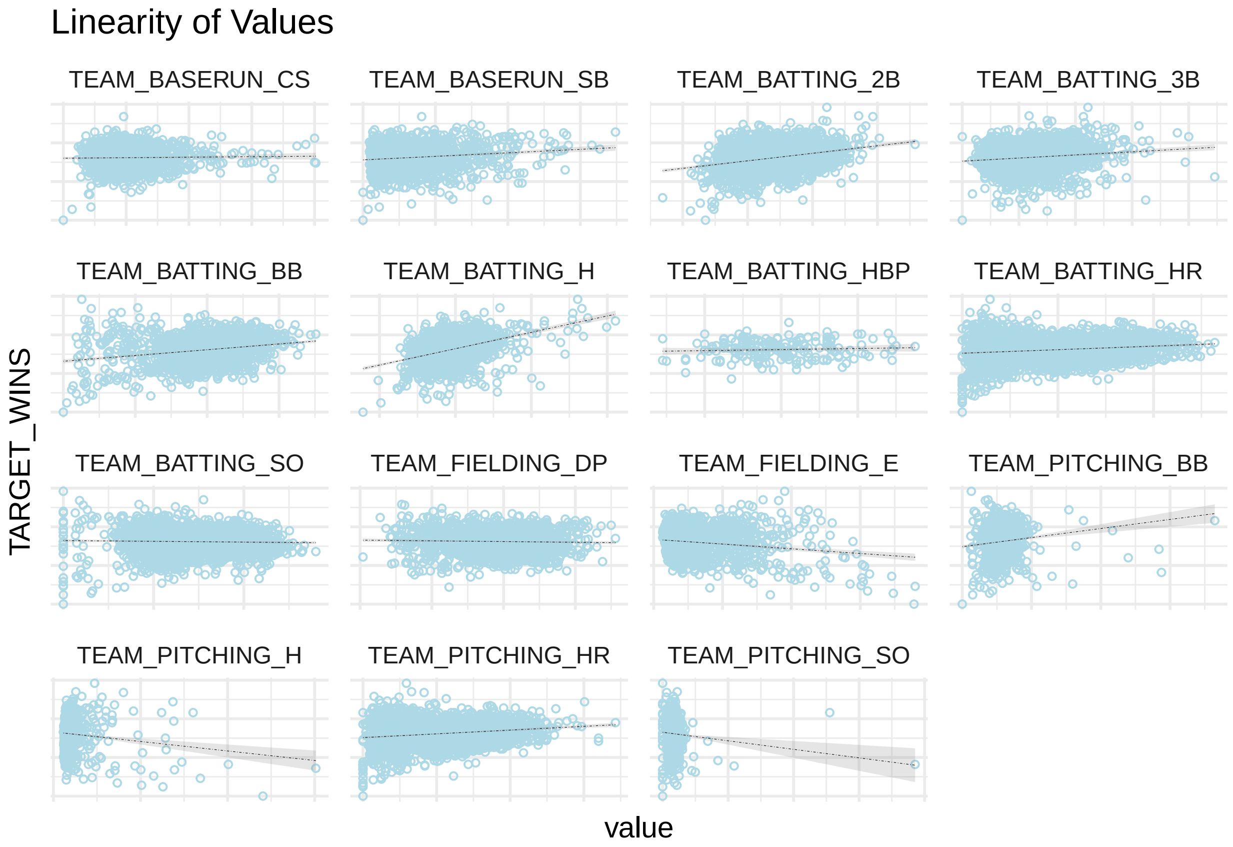
The average number of wins to a team is about 81 while the highest number of wins recorded for a team at 146 in a season. Note that, these calculations do not include any missing values nor outliers which should make them reasonably representative of the real-world. Recall that our main goal is to use this data to predict the number of wins for the team.

Our expectation is that the number of wins will correlate best with hits, home runs, fielding errors, and pitching strikeouts to produce a linear model that is capable of predicting whether a team will win with over 50% accuracy. With that, we are making some assumptions about the model, especially regarding TARGET_WINS which records the number of wins each season. These assumptions include normality in the distribution, linearity, independence among events, and homoscedasticity. With these assumptions satisfied, we can confirm that it is reasonable to make predictions using the number of wins as our target variable through proofs.

For purposes of this analysis, we accept the idea that each event must be independent of one another since a statistic about one player does not imply that we know any statistics about another player, nor the entire team. Rather, we assume that a combination of performance statistics about each player over time, can inform us about how well the team will fare in future seasons. With that acceptance, we have our first proof; independence among events.

The other conditional assumptions of linearity, normality, and homoscedasticity can be proven using visualizations. Starting with linearity, we intend to demonstrate that all of these variables are able to be represented by a linear pattern. For this, an array of scatter plots is shown with a linear trend overlaying the performance statistics.

```r
tdata[2:17]  %>%
  gather(variable, value, -TARGET_WINS) %>%
  ggplot(., aes(value, TARGET_WINS)) +
  ggtitle("Linearity of Values") +
  geom_point(fill = "white",
             size=1,
             shape=1,
             color="light blue") +
  geom_smooth(formula = y~x,
              method = "lm",
              size=.1,
              se = TRUE,
              color = "black",
              linetype = "dotdash",
              alpha=0.25) +
  facet_wrap(~variable,
             scales ="free",
             ncol = 4) +
  theme(axis.text.x = element_blank(), axis.text.y = element_blank())
```

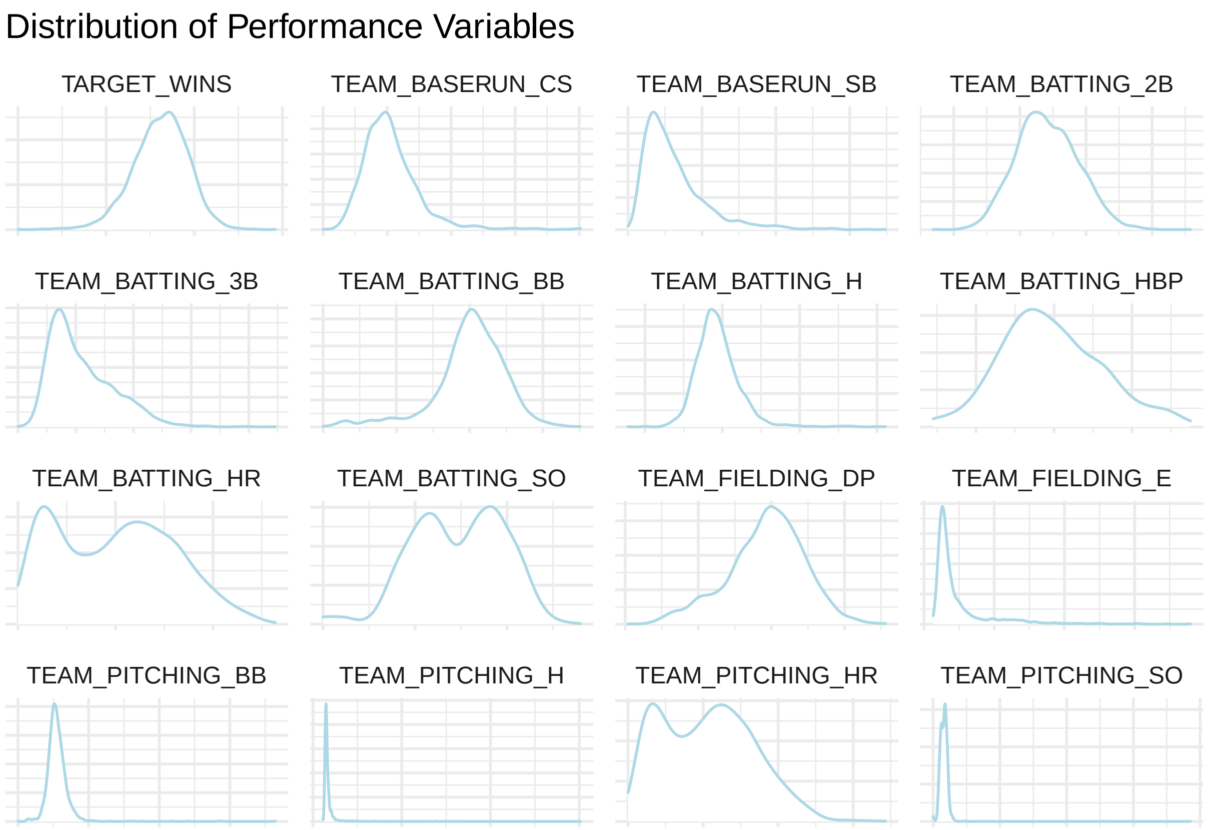


While this includes the outliers mentioned previously, it is reasonable to assume that almost all of these variables could be predicted using linear trends. The only potential non-linear variables are TEAM_FIELDING_E, TEAM_PITCHING_BB, TEAM_PITCHING_H, and TEAM_PITCHING_SO. At first glance, these look like they might fit best with another kind of model. But, with outliers removed and perhaps a normalizing transformation we may alter the fit enough for their prediction potential (and subsequent correlation with wins) to improve model results.

Intentionally, we also do not know which correlations will be useful for the model at this point. To decide on them now would not be helpful as we are unsure of their efficacy in any linear model. Thus, we continue validating these assumptions to build the models properly.

Normality is another assumption that we must ensure is present to make accurate predictions with linear regression. It can be viewed and confirmed in several ways but here we use a density plot to search for smooth bell-curves and limited skewness.

```
tdata %>%
  gather(variable, value, TARGET_WINS:TEAM_FIELDING_DP) %>%
  ggplot(., aes(value))  +
  ggtitle("Distribution of Performance Variables") +
  geom_density(fill = "white", color="lightblue") +
  labs(x = element_blank(), y = element_blank()) +
  facet_wrap(~variable, scales ="free", ncol = 4) +
  theme(axis.text.x = element_blank(), axis.text.y = element_blank())
```

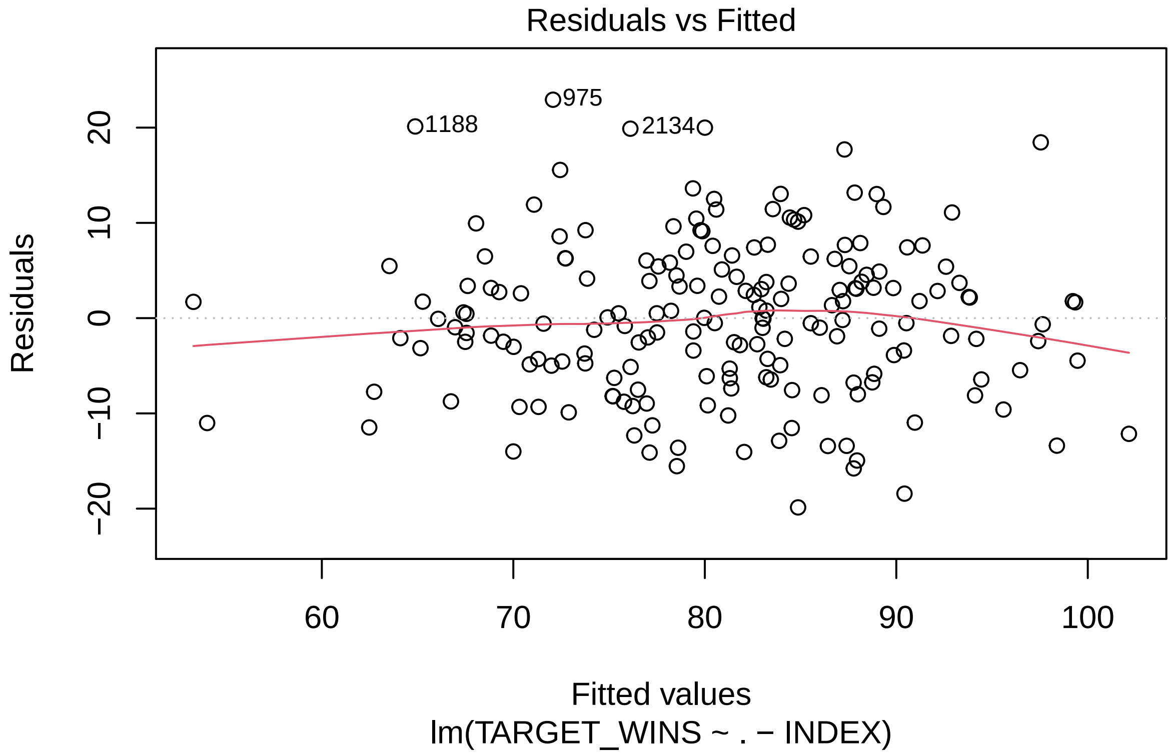


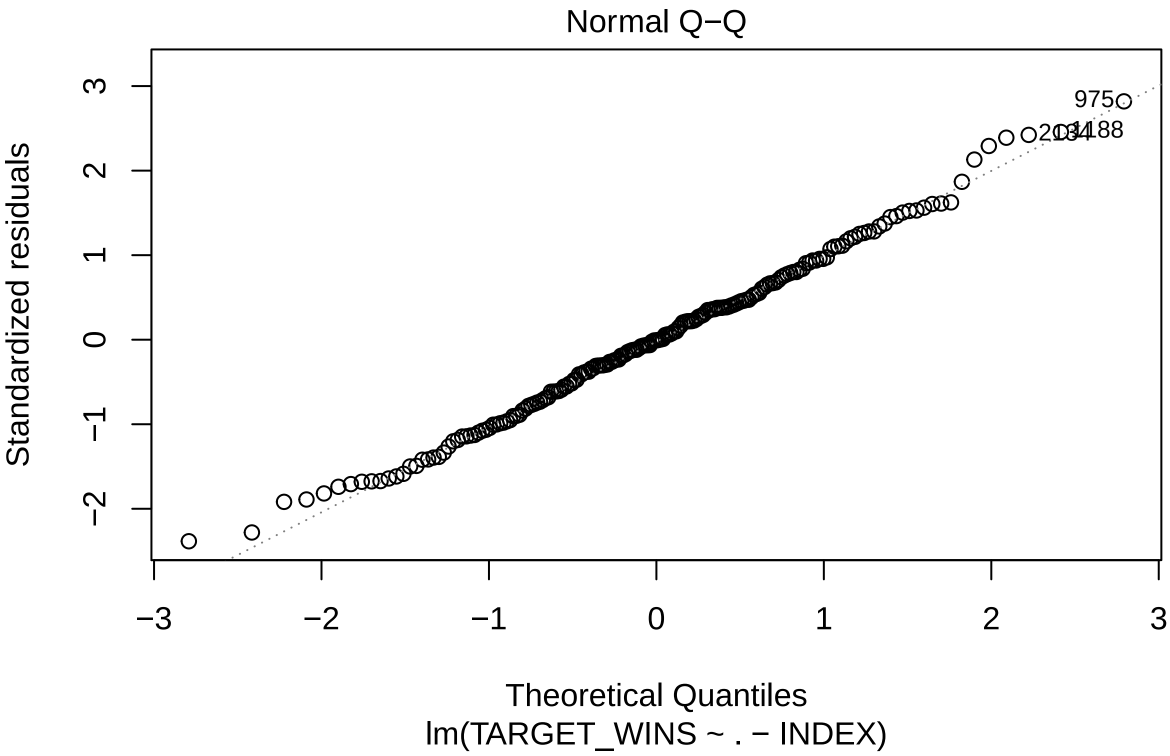
Those same potential predictors that exhibited a nonlinear trend are causing problems. As individual predictors of game wins they do not appear normal. This could be caused entirely by outliers but we will have to take a closer look at all the variables to identify the cause of their variation. For this, we run a simple linear model that contains all variables except the index to see how they interact together. In doing so we also check for homoscedasticity, or consistency in variance among the variables.

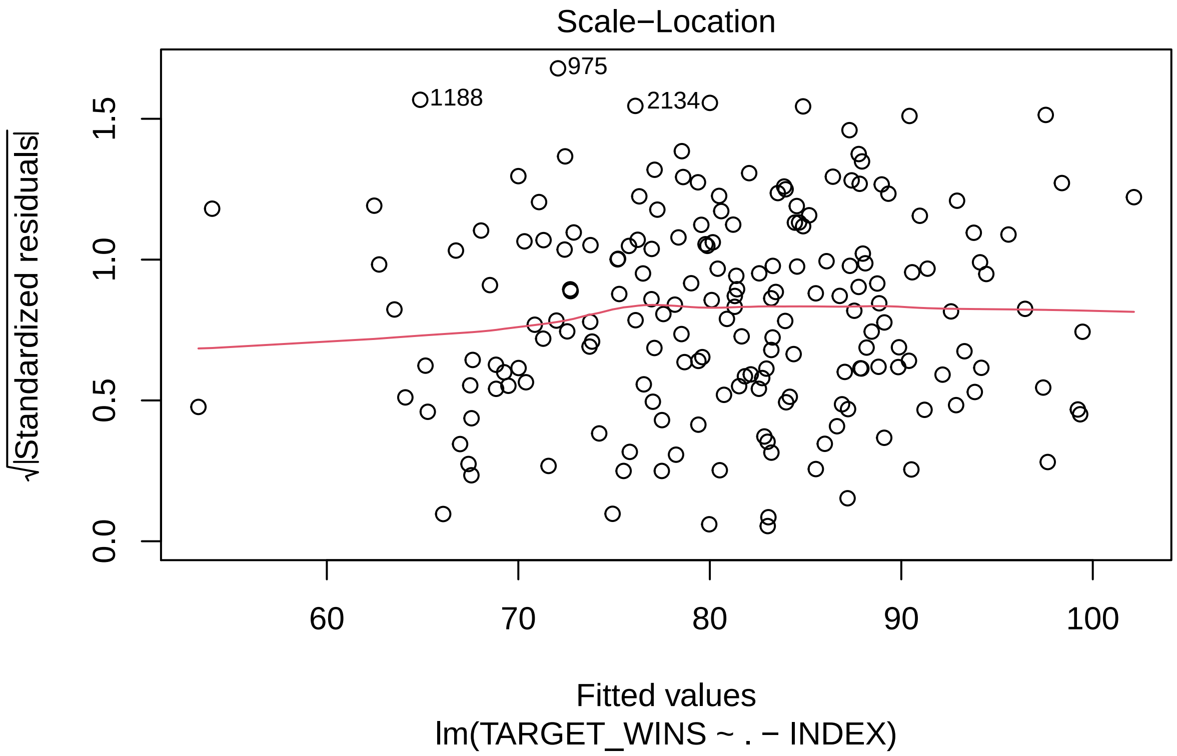
```
ks.test <- lm(TARGET_WINS ~. -INDEX, tdata)
summary(ks.test)
```

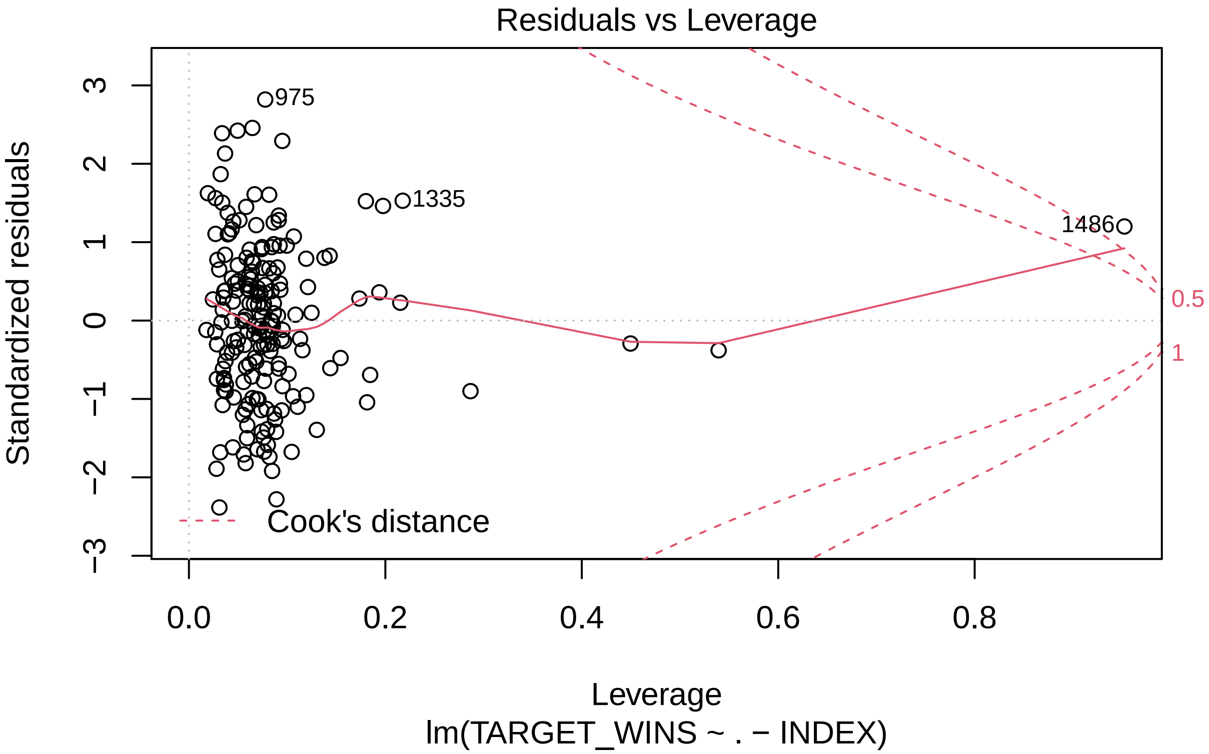
```
##
## Call:
## lm(formula = TARGET_WINS ~ . - INDEX, data = tdata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.8708  -5.6564  -0.0599   5.2545  22.9274
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       60.28826   19.67842   3.064  0.00253 **
## TEAM_BATTING_H     1.91348    2.76139   0.693  0.48927
## TEAM_BATTING_2B    0.02639    0.03029   0.871  0.38484
## TEAM_BATTING_3B   -0.10118    0.07751  -1.305  0.19348
## TEAM_BATTING_HR   -4.84371   10.50851  -0.461  0.64542
## TEAM_BATTING_BB   -4.45969    3.63624  -1.226  0.22167
## TEAM_BATTING_SO    0.34196    2.59876   0.132  0.89546
## TEAM_BASERUN_SB   0.03304    0.02867   1.152  0.25071
## TEAM_BASERUN_CS  -0.01104    0.07143  -0.155  0.87730
## TEAM_BATTING_HBP  0.08247    0.04960   1.663  0.09815 .
## TEAM_PITCHING_H  -1.89096    2.76095  -0.685  0.49432
## TEAM_PITCHING_HR  4.93043   10.50664   0.469  0.63946
## TEAM_PITCHING_BB  4.51089    3.63372   1.241  0.21612
## TEAM_PITCHING_SO -0.37364    2.59705  -0.144  0.88577
## TEAM_FIELDING_E  -0.17204    0.04140  -4.155 5.08e-05 ***
## TEAM_FIELDING_DP -0.10819    0.03654  -2.961  0.00349 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.467 on 175 degrees of freedom
##   (2085 observations deleted due to missingness)
## Multiple R-squared:  0.5501, Adjusted R-squared:  0.5116
## F-statistic: 14.27 on 15 and 175 DF,  p-value: < 2.2e-16
```

```
plot(ks.test)
```

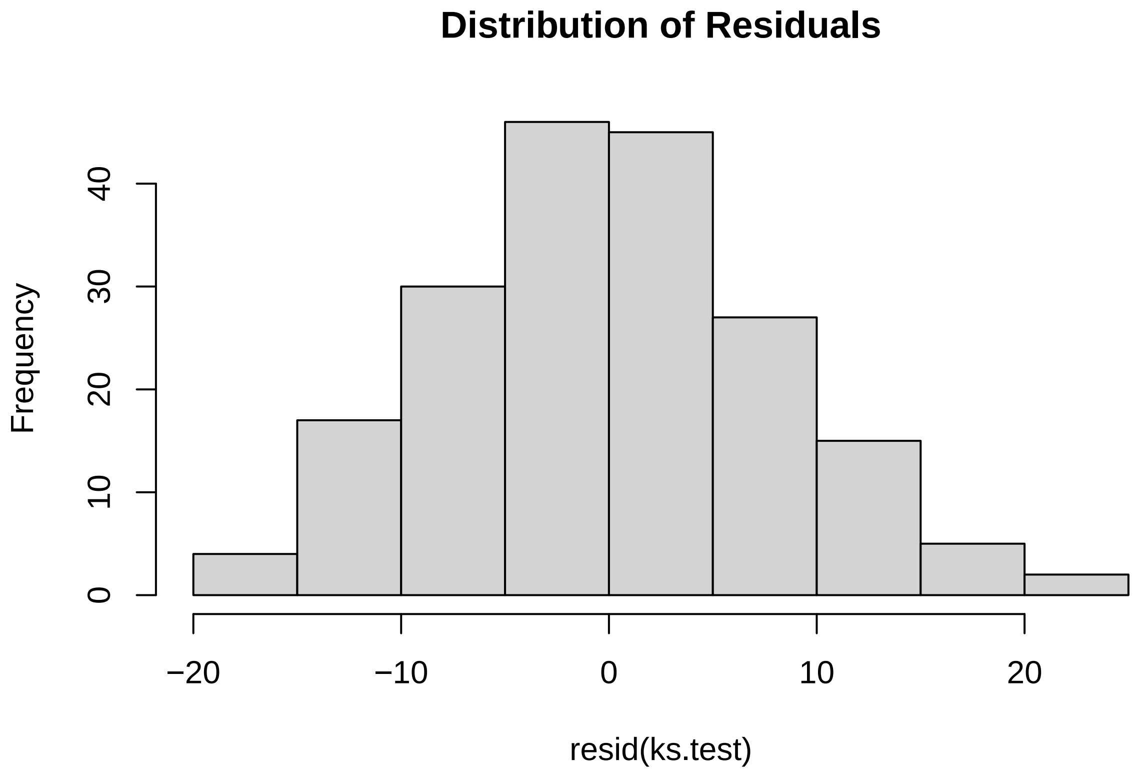



Normal Q−Q
Standardized residuals
3
2
1
0
−1
−2
975
2134
1188
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(TARGET_WINS ~ . − INDEX)


Scale−Location
975
1188
2134
√|Standardized residuals|
0.0
0.5
1.0
1.5
60
70
80
90
100
Fitted values
lm(TARGET_WINS ~ . − INDEX)



```r
hist(resid(ks.test), main="Distribution of Residuals")
```

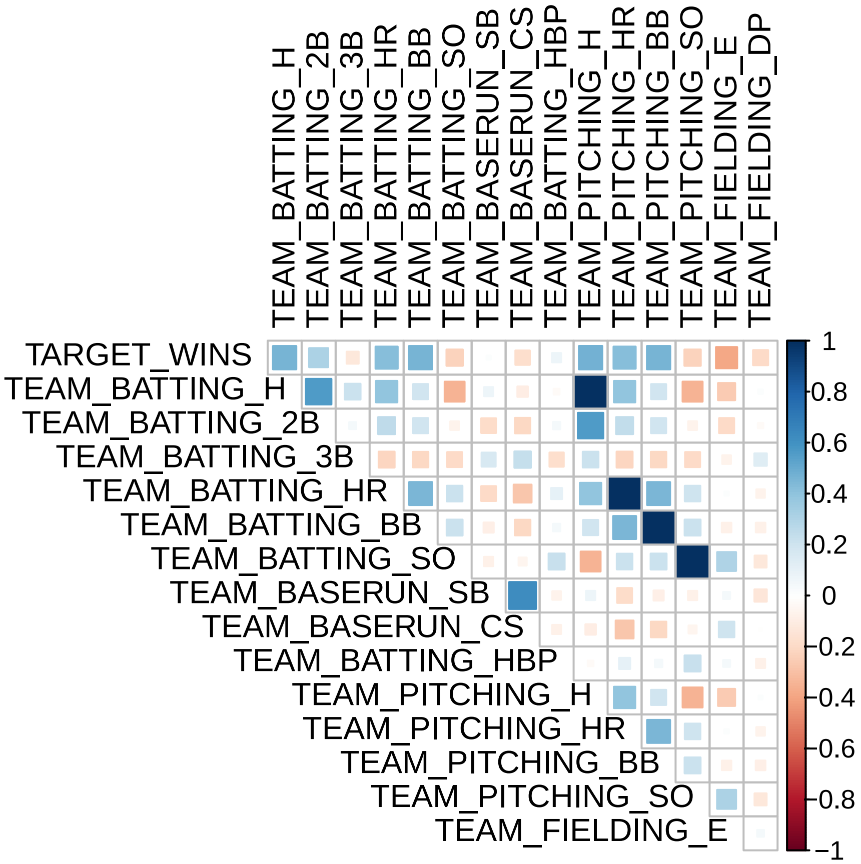


Represented by the standardized residuals the homoscedasticity assumption is confirmed. There is constant variance among the random variables. However, normality seems to have slightly fixed itself when considering all variables. As an added bonus to viewing this kind of model, it reveals patterns of interaction with the variables and their strengths. This model would be good enough to run on its own, but we should continue to consider how it could be improved for data preparation. We should fix missing variables, reduce high leverage points, and make at least one transformation on those problem variables even though our assumptions are closely met.

Predictor Correlations

To determine which variables could prove to be the best predictors of game wins, it is important to consider the strength and direction of correlation. For a quick look at how each variable compares with each other, this correlation plot was built using only complete observations.

```
tdata[2:17] %>%
  cor(., use = "complete.obs") %>%
  corrplot(., method = "square", type = "upper", tl.col = "black", diag = FALSE)
```



Predictors such as TEAM_FIELDING_DP and TEAM_FIELDING_E do not correlate well with a correlation value of around zero. Meanwhile, the predictors TEAM_BATTING_HR and TEAM_PITCHING_HR have strong positive correlations of nearly one. Functionally, this makes sense since the quantity of pitches that gave a home run are directly effected by the quantity of batters that hit home runs. For our purposes, we are mainly concerned with the top row of correlations with game wins which are the most important to our model.

If we want to make accurate predictions we need to include the the most feasible predictors with correlations that have the strongest relationships with the winnings. We understand that autocorrelated predictors are impractical and will misrepresent the relationships between predictors. In some other cases, the relationship may be strong enough to use only one or two predictors for a model. For us, this is not the case. Take a look at this table of correlations.

```
# Gather complete observations
cors <- tdata %>%
  cor(., use = "complete.obs")
# Create fillable matrix
cors[lower.tri(cors, diag=TRUE)] <- ""
# Use observations to find correlation with wins
cors <- cors %>%
  as.data.frame() %>%
  rownames_to_column() %>%
  gather(Variable, Correlation, -rowname) %>%
  mutate(Correlation = as.numeric(Correlation)) %>%
# Format table to include Strength of correlations
  rename(‘ Wins‘ = rowname) %>%
  filter(Correlation != "") %>%
  arrange(desc(abs(Correlation))) %>%
  filter(‘ Wins‘ == "TARGET_WINS") %>%
  mutate(Strength = abs(Correlation))
# Display table of correlations
kbl(cors, booktabs = T, caption = "Predictor Correlations with Wins") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
                full_width = F) %>%
  footnote(c("No correlation is higher than 0.48"))
```

Table 5: Predictor Correlations with Wins

Wins	Variable	Correlation	Strength
TARGET_WINS	TEAM_PITCHING_H	0.4712343	0.4712343
TARGET_WINS	TEAM_BATTING_H	0.4699467	0.4699467
TARGET_WINS	TEAM_BATTING_BB	0.4686879	0.4686879
TARGET_WINS	TEAM_PITCHING_BB	0.4683988	0.4683988
TARGET_WINS	TEAM_PITCHING_HR	0.4224668	0.4224668
TARGET_WINS	TEAM_BATTING_HR	0.4224168	0.4224168
TARGET_WINS	TEAM_FIELDING_E	-0.3866880	0.3866880
TARGET_WINS	TEAM_BATTING_2B	0.3129840	0.3129840
TARGET_WINS	TEAM_PITCHING_SO	-0.2293648	0.2293648
TARGET_WINS	TEAM_BATTING_SO	-0.2288927	0.2288927
TARGET_WINS	TEAM_FIELDING_DP	-0.1958660	0.1958660
TARGET_WINS	TEAM_BASERUN_CS	-0.1787560	0.1787560
TARGET_WINS	TEAM_BATTING_3B	-0.1243459	0.1243459
TARGET_WINS	TEAM_BATTING_HBP	0.0735042	0.0735042
TARGET_WINS	TEAM_BASERUN_SB	0.0148364	0.0148364

Note:
No correlation is higher than 0.48

No correlation has a strength greater than 0.48, as shown in the table of predictor correlations with wins. This implies that there are other variables that influence whether a team wins a game. Fortunately, this was expected since baseball, although it has a theoretical advantage with exceptional individuals, remains a team sport.

Based on the strength of the correlation the variables home runs, hits on base, number of walks, and errors when fielding have the greatest probability of being the best predictors in our model. They represent the upper 50% of our correlations with the target variable, games won. By contrast, stolen bases, double plays, and strikeouts from either the pitcher or batter, do not exert as much influence over game outcome.

As it stands, there is still a significant influence from outliers, missing data points, and variables we may not need. These correlations give us a good sense of where to begin in preparing the data for the model. But before we settle on these predictors, there are a few other considerations to make that may sway our model tremendously.

Model Considerations

Before we prepare the data for modeling, we consider the bigger picture and how this model holds under the scope of baseball. There are plenty of factors that might influence the predictive power of the model that are not necessarily present in the charts but certainly present in reality. We start by thinking critically about the timeline of the sport.

Given such a large span of history, there have been many changes from the beginning of this analysis to the present day. These changes may cause a team's success to change, thereby lowering the overall accuracy of these statistics as predictors.

As an example of this, consider the social factors that changed the fabric of baseball and the MLB since the start of this data set in 1871. There was racial segregation which did not end until 1964 and jim crow laws thereafter. There were several major wars including two world wars, the creation of new leagues that divided the players into separate groups and players began to contractually obligate themselves to play for teams as their main profession. Additionally, the rules have changed over time, all continuously shaping a sport that was invented just 32 years prior to the start of this study.

These are not factors that we can control for in the model. Rather, it is important to consider them since they may have great influence over the results and prediction accuracy. To learn from the model we must understand the level and scope of influence outside variables not included in this data set may have on the model its6elf. Otherwise, we cannot hope to improve it.

Data Preparation

Sample Set

Based on this data set we have options to choose from when deciding how to best prepare and test the data. In this training data set, the full set of performance statistics are given from 1871 to 2006. This implies that the data could be used for training a model built solely on this set alone with no other sources of data. For simplicity, that is what we will do. This begins by taking a sample.

```
# Take sample of tdata or
# Use evaluation data to predict wins
set.seed(0621)
edata <- read.csv("https://raw.githubusercontent.com/palmorezm/msds/main/621/HW1/moneyball-evaluation-da
tsample_300 <- sample_n(tdata[2:17], 300,
                        replace = FALSE, weight = NULL) # random
# sample of 300 rows; no replacement
edata <- tsample_300 # will be used to predict win from random sample
```

The sample contains 300 randomly selected records from the training data without replacement. For now, this data is stored as our evaluation data. The first portion of the data set is shown for reference.

```
# only 15 columns - index and games won are not included
edata.tbl.1 <- edata[1:5,c(2:5)]
edata.tbl.2 <- edata[1:5,6:9]
edata.tbl.3 <- edata[1:5,10:13]
edata.tbl.4 <- edata[1:5,14:16]
# display samples of table data
etbl.1 <- kbl(tdata.tbl.1, booktabs = T, caption = "Evaluation Data Fields 1 - 4") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
etbl.2 <- kbl(tdata.tbl.2, booktabs = T, caption = "Evaluation Data Fields 5 - 8") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
etbl.3 <- kbl(tdata.tbl.3, booktabs = T, caption = "Evaluation Data Fields 9 - 12") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
etbl.4 <- kbl(tdata.tbl.4, booktabs = T, caption = "Evaluation Data Fields 13 - 15") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
```

```
etbl.1
```

Table 6: Evaluation Data Fields 1 - 4

INDEX	TARGET_WINS	TEAM_BATTING_H	TEAM_BATTING_2B	TEAM_FIELDING_DP
1	39	1445	194	NA
2	70	1339	219	155
3	86	1377	232	153
4	70	1387	209	156
5	82	1297	186	168

```
etbl.2
```

Table 7: Evaluation Data Fields 5 - 8

TEAM_BATTING_3B	TEAM_BATTING_HR	TEAM_BATTING_BB	TEAM_BATTING_SO
39	13	143	842
22	190	685	1075
35	137	602	917
38	96	451	922
27	102	472	920

```
etbl.3
```

Table 8: Evaluation Data Fields 9 - 12

TEAM_BASERUN_SB	TEAM_BASERUN_CS	TEAM_BATTING_HBP	TEAM_PITCHING_H
NA	NA	NA	9364
37	28	NA	1347
46	27	NA	1377
43	30	NA	1396
49	39	NA	1297

```
etbl.4
```

Later in the analysis, we will use this test evaluation data (or edata), to test the selected model since it is quite similar to our training data (but not exactly the same). This too can be shown. Observe the differences in missing data from the sample and training data sets.

```
# calculate missing values
edata.missing.values <- sum(is.na(edata))
edata.total.obs <- count(edata)
```

Table 9: Evaluation Data Fields 13 - 15

TEAM_PITCHING_HR	TEAM_PITCHING_BB	TEAM_PITCHING_SO	TEAM_FIELDING_E
84	927	5456	1011
191	689	1082	193
137	602	917	175
97	454	928	164
102	472	920	138

```r
# find number of fields
edata.fields <- ncol(edata)
tdata.fields <- ncol(tdata)
# find total entries
edata.entry <- edata.total.obs * ncol(edata)
tdata.entry <- total.obs * ncol(tdata)
# calculate percent missing
edata.percent.missing <- edata.missing.values / edata.entry * 100
percent.missing <- missing.values / tdata.entry * 100

# create matrix of values
compare.data <- matrix(c(edata.missing.values,
                         missing.values,
                         edata.total.obs,
                         total.obs,
                         tdata.fields,
                         edata.fields,
                         edata.entry,
                         tdata.entry,
                         round(edata.percent.missing, digits=2),
                         round(percent.missing, digits=2)), ncol=5, byrow=FALSE)
compare.data
```

```
##      [,1] [,2] [,3] [,4]  [,5]
## [1,] 463  300  17   4800  9.65
## [2,] 3478 2276 16   38692 8.99
```

```r
compare.data <- as.data.frame(compare.data)
header <- c("Missing Values", "Observations", "Fields", "Total Entries", "Percent Missing")
row <- c("eData", "tData")
colnames(compare.data) <- header
row.names(compare.data) <- row
kbl(compare.data)
```

	Missing Values	Observations	Fields	Total Entries	Percent Missing
eData	463	300	17	4800	9.65
tData	3478	2276	16	38692	8.99

The training and sample data have relatively similar missing data points. One might expect that if the data were randomly sampled from the training data set then the relationships between the variables should be largely preserved. We rely on these principles in future testing.

Adjustments

When we explored the data set we found that there were missing values in the data. These should be fixed or removed. We also discovered that some fields were impractical or had little to no value over prediction. These fields should also be removed. Outliers are also a major concern and must be dealt with. Additionally, we fix the remaining missing values by assigning them the median value of the distribution so as to prevent altering the relationship too much. Performing these adjustments should improve the model.

```r
# select all but index, batting_hbp, baserun_cs
# replace missing values with the median value for the distribution
tdata <- tdata %>%
  rename("win" =TARGET_WINS, # For use with edata sampling
        "bhit" =TEAM_BATTING_H,
        "bdouble"  = TEAM_BATTING_2B,
        "btriple"  = TEAM_BATTING_3B,
        "bhomerun"  = TEAM_BATTING_HR,
        "bwalk"  = TEAM_BATTING_BB,
        "bstrikeout"  = TEAM_BATTING_SO,
        "stolenbase"  = TEAM_BASERUN_SB,
        "caught"  = TEAM_BASERUN_CS,
        "bhitbyp"  = TEAM_BATTING_HBP,
        "phit"  = TEAM_PITCHING_H,
        "phomerun"  = TEAM_PITCHING_HR,
        "pwalk"  = TEAM_PITCHING_BB,
        "pstrikeout"  = TEAM_PITCHING_SO,
        "fielderror"  = TEAM_FIELDING_E,
        "fielddp"  = TEAM_FIELDING_DP)
```

```r
# drops insignificant columns
# replaces NA with median (given a removal of missing values in calculation)
df <- tdata[c(2:9,12:17)]          # use if raw tdata
# df <- tdata[c(1:7,10:15)] # use with sampled tdata excluding win variable
# produces 13 columns/fields with sampled tdata excluding win variable
# produces 15 columns/fields with raw tdata excluding win variable
for (i in colnames(df)) {
  df[[i]][is.na(df[[i]])] <- median(df[[i]], na.rm=TRUE)
}
```

Above, we convert the field variable name into something readable and easier to work with. Then, insignificant fields are removed. The selection process is based on those variables that had low correlations near zero, in the exploration section. Missing values are then replaced with the median value for the distribution. Importantly, this calculation was made assuming there were no missing values so as not to further skew or deviate from the original distribution. It is our attempt to find the truth in the noisy mess of figures. The median was chosen due to its status as a robust statistic.

Next, we identify and remove outliers. As shown in the violin plots with box plots overlaid, there are quite a few extreme data points that are skewing many variables. Some of these have high correlation values and may make for good predictors. These are dealt with using the interquartile range (IQR) of each field's distribution.

```r
# remove outliers based on IQR
for (i in colnames(df)) {
  iqr <- IQR(df[[i]])
  q <- quantile(df[[i]], probs = c(0.25, 0.75), na.rm = FALSE)
  qupper <- q[2]+1.5*iqr
  qlower <- q[1]+1.5*iqr
  outlier_free <- subset(df, df[[i]] > (q[1] - 1.5*iqr) & df[[i]] < (q[2]+1.5*iqr) )
}
df <- outlier_free
```

Cycling through each variable in the data, we identify the extreme values beyond the lower and upper quartiles. Where outliers are found the data is subset, effectively removing the data points from each field. Outliers were determined by one and a half standard deviations above the upper and below the lower quartile in each distribution. This gave us a little bit of wiggle room to include rare occurrences but exclude factually extreme values. Given a bit of domain knowledge, some outlier points are possible while others are not which means a small amount of these outliers will remain, as they should, in each field.

New Estimates

Based on the descriptions of the available fields, there are no singles from batters. While this is theoretically possible, in reality, the probability of there never being a single base hit in baseball for the MLB is near impossible. This is corrected and an additional field, one that we are calling 'gt1base' is computed as a new field estimate by each year.

```
# Create new estimates from stats
df <- df %>%
  # Hits resulting in greater than one base
  mutate(gt1base = bdouble + btriple + bhomerun) %>%
  # Hits resulting in one base
  mutate(bsingle = bhit - gt1base)
```

We created the 'gt1base' from the sum of performance statistics about base hits. As its name implies, it records base hits that are greater than one base. That is, all doubles, triples, and homeruns. This may help prove that big hitters improve the outcome of games. Alternatively, it could also disprove it in favor of consistently hitting singles and getting baseruns.

Singles from batters are also created as a new estimate. Specifically, this records the number of batters that hit the ball and got to first base. To get singles, we found the difference in total hits from those with 'gt1base'. As with 'gt1base', these new estimates are calculated based on the year they were recorded. We utilized the relationship between base hits and those batters who hit and run beyond one base to extract these singles.

Predictor Ratios

When exploring the strongest correlations, or specifically, those that were near perfect with values of 0.99 and others in the ballpark of 0.90-0.99, we discovered the correlation table, had autocorrelations that resulted in strength values near one. Rather than have to drop the strength of these correlations, it might prove beneficial to the model if we can leverage the strength of these correlations as a ratio. Thus, they were computed.

```
# Compute predictor ratios
df <- df %>%
  mutate(pbh = phit/bhit) %>%
  mutate(pbhr = phomerun/bhomerun) %>%
  mutate(pbw = pwalk/bwalk) %>%
  mutate(fdpe = fielddp/fielderror)
```

Hits from batters thrown by pitchers, for example, is not an independent relationship. The pitcher must throw for the batter to hit and unless the pitcher intentionally allows the batter to hit the ball, then they are in fact directly caused by one another. To make use of it, the ratio of times the pitcher threw the ball to the number of times the batter hit the ball, is a valid, independent event (or rather series of events) that may lead to a better outcome and increase the chances of a team to win a game. The remaining ratios follow this same principle.

We calculated how many times a pitcher allowed a homerun with the number of times the batter hit a homerun. The number of times the pitcher gave the batter a walk and the number of times the batter walked. Lastly, the number of times fielders made a double play compared to the number of times fielders made a mistake. Importantly, every game is a unique play stemming from factors in the environment which mean these ratios should be askew from perfect autocorrelations. If this is the case, minute variation in play ratios may change the game's outcome.

Transformations

At this point, some of the data remains abnormal. Those distributions in the exploratory section describe how TEAM_FIELDING_E, TEAM_PITCHING_BB, TEAM_PITCHING_H, and TEAM_PITCHING_SO exhibit non-normal, perhaps nonlinear trends. To fix this we need to determine what the best kind of transformation is to normalize it. We do so using the bestNormalize function within the 'bestNormalize' package. The process is shown.

```
# determine best way to transform and their metrics
bestNorms <- df[1:11,]
for (i in colnames(df)) {
  bestNorms[[i]] <- bestNormalize(df[[i]], allow_orderNorm = FALSE, out_of_sample = FALSE)
}
```

In this function, it performs multiple normalizing transformations including the Lambert W x F, Box-Cox, and Yeo-Johnson transformations along with other methods like the ordered quantile technique. The results are then reviewed by the function to determine which distribution and associated residuals appear the most Gausian. It relies on a few other functions, like those of the MASS package, to estimate the best value of lambda or other appropriate variables necessary for performing the transformation and produces a list of recommendations.

```
# focus on problem areas; those with non-normal stats
bestNorms$fielderror$chosen_transform
```

```
## Standardized Yeo-Johnson Transformation with 2184 nonmissing obs.:
##  Estimated statistics:
##  - lambda = -0.9878936
##  - mean (before standardization) = 1.005778
##  - sd (before standardization) = 0.002801129
```

```
bestNorms$pwalk$chosen_transform
```

```
## Standardized Yeo-Johnson Transformation with 2184 nonmissing obs.:
##  Estimated statistics:
##  - lambda = 0.301843
##  - mean (before standardization) = 18.87912
##  - sd (before standardization) = 1.68402
```

```
bestNorms$phit$chosen_transform
```

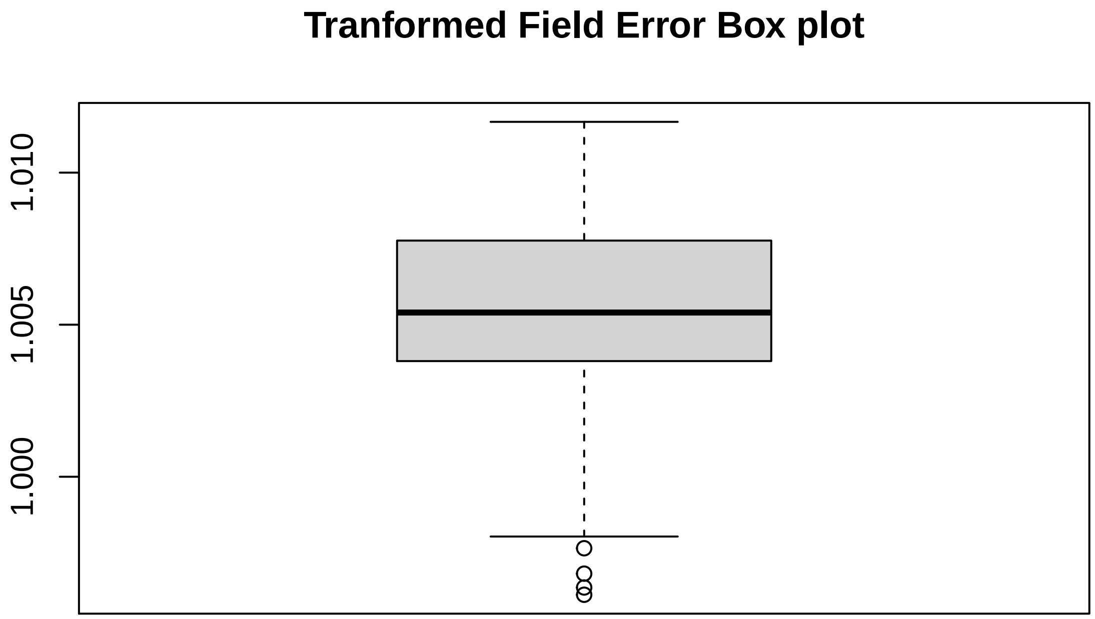
```
## Standardized Yeo-Johnson Transformation with 2184 nonmissing obs.:
##  Estimated statistics:
##  - lambda = -3.125283
##  - mean (before standardization) = 0.319971
##  - sd (before standardization) = 1.435836e-11
```

```
bestNorms$pstrikeout$chosen_transform
```

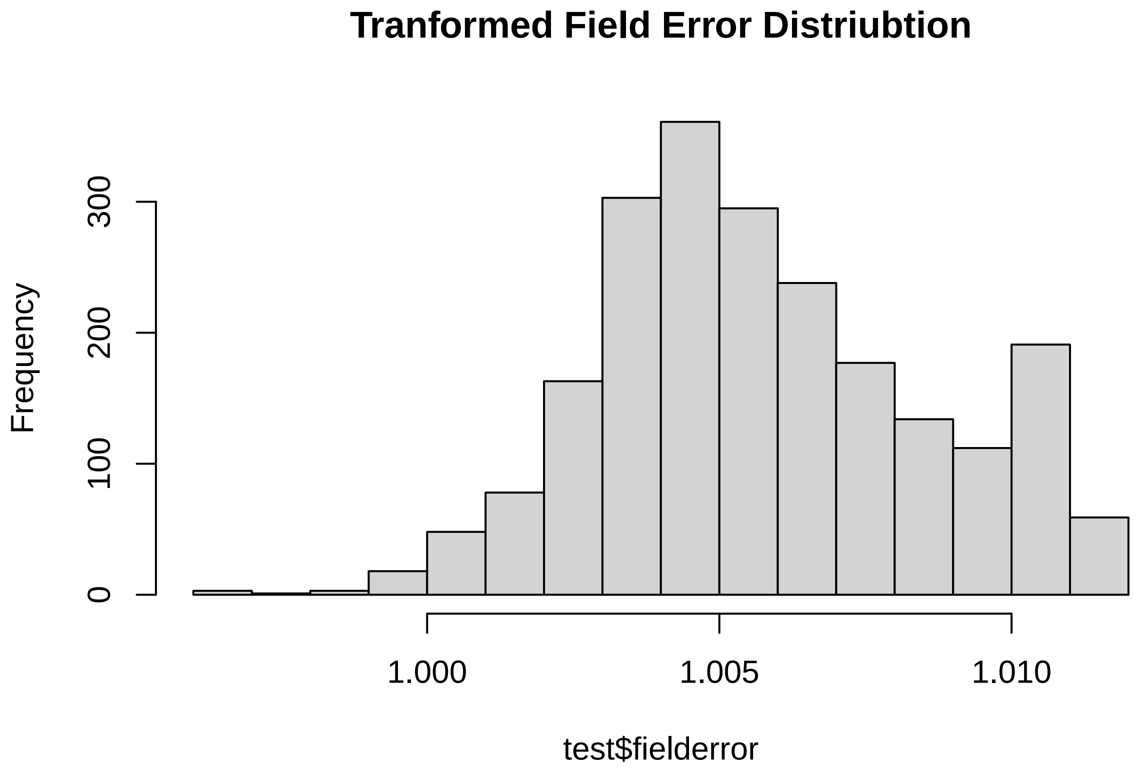
```
## Standardized Yeo-Johnson Transformation with 2184 nonmissing obs.:
##  Estimated statistics:
##  - lambda = 0.4299074
##  - mean (before standardization) = 38.4267
##  - sd (before standardization) = 7.402144
```

The Yeo-Johnson is the preferred choice given the options available in the bestNormalize function. Perhaps, unsurprisingly, since these distributions shared the same levels of skew, nonlinearity, and heteroscedasticity, the same transformation is recommended for each. This is similar to a Box-Cox transformation but can work on a wider array of values. An example Yeo-Johnson is performed below with associate box plots and histograms for reference.

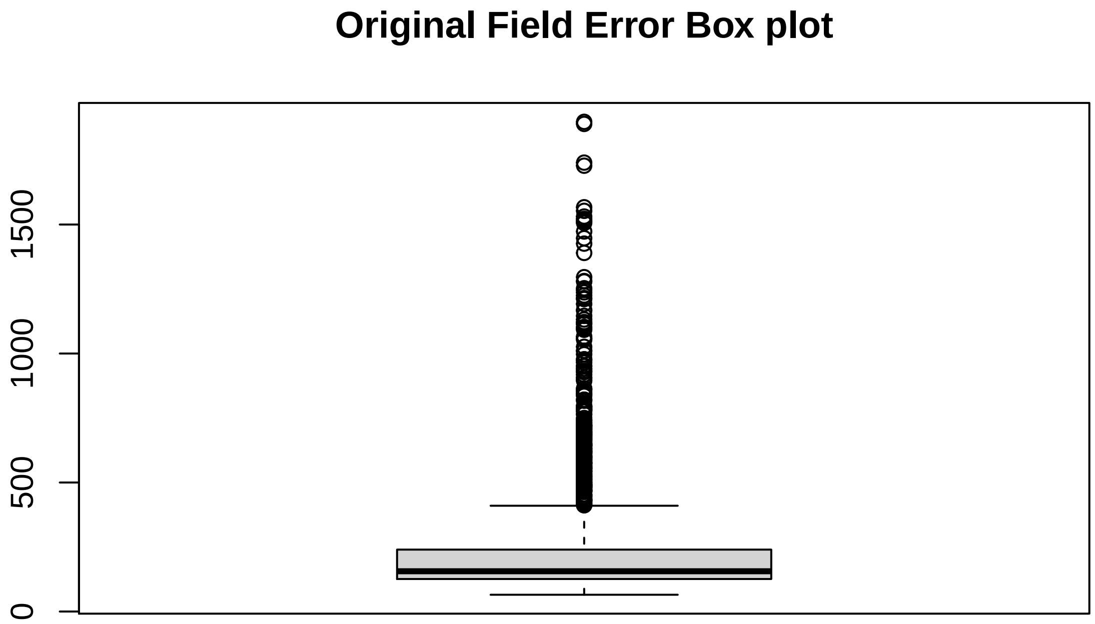
```
# Example Yeo-Johnson transformation
test <- df
test$fielderror <- yeo.johnson(df$fielderror, lambda = -0.9878936, derivative = 0, epsilon = sqrt(.Mach
boxplot(test$fielderror, main= "Tranformed Field Error Box plot")
```



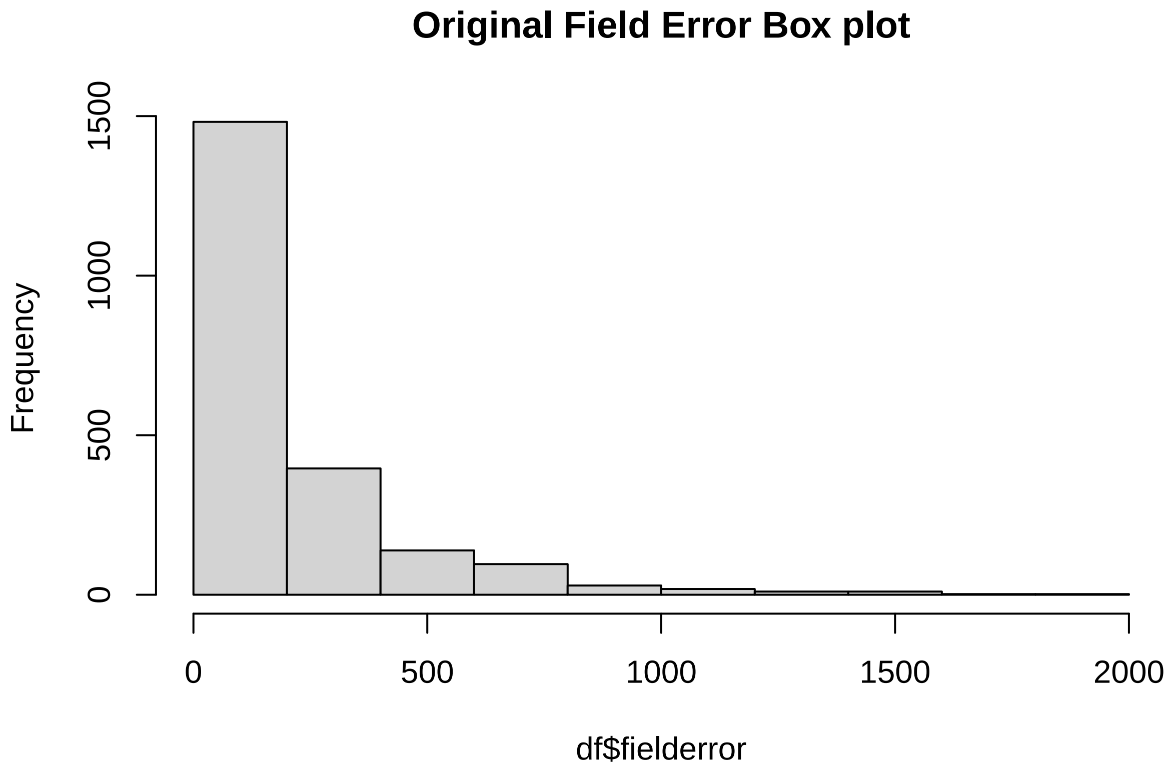
```r
hist(test$fielderror, main= "Tranformed Field Error Distriubtion")
```



```
boxplot(df$fielderror, main= "Original Field Error Box plot")
```



```r
hist(df$fielderror, main= "Original Field Error Box plot")
```



Notice how the distribution normalizes from a large right skew to a nearly pristine normal distribution. With same relationship between the values, this transformation merely secures our option to model them as predictors. The box plot distribution also shows that there are far fewer outliers since our attempt to limit them. Our transformation appears to have also assisted with the reduction of those high leverage values, as well as creating homogeneous standard errors. The Yeo-Johnson transformation is applied to the other variables in the same way with their appropriate values of lambda.

```
# Applying Yeo-Johnson transformation
df$fielderror <- yeo.johnson(df$fielderror, lambda = -0.9878936, derivative = 0, epsilon = sqrt(.Machin
df$pwalk <- yeo.johnson(df$pwalk, lambda = 0.301843, derivative = 0, epsilon = sqrt(.Machine$double.eps
df$phit <- yeo.johnson(df$phit, lambda = -3.125283, derivative = 0, epsilon = sqrt(.Machine$double.eps)
df$pstrikeout <- yeo.johnson(df$pstrikeout, lambda = 0.4299074, derivative = 0, epsilon = sqrt(.Machine
```

We re-evaluate for missing or non-applicable data points to ensure none remain. We should not have any for our original variables but the new estimates we created need readjusting. The same process for fixing missing values is applied. A new median metric is found for each estimate and stored in place of the missing value; thereby limiting the change to the distribution or relationship between variables.

```
# Reassign median values to new ratios where NA value is present
for (i in colnames(df)) {
  df[[i]][is.na(df[[i]])] <- median(df[[i]], na.rm=TRUE)
}
# sum(is.na(df)) # ensure no missing values remain
```

Review

To bring it all together, the final prepared data frame contains several new variables. This includes the predictor ratios, the estimate of batters hitting singles, and the estimate of times that batters hit and ran more than one base. These are shown below separate from the main data frame.

```r
# New variables
df.tbl.2 <- df[1:5,15:20] # raw tdata only
# head(df[14:19]) # sample edata only
kbl(df.tbl.2, booktabs = T, caption = "Clean Training Data Fields 15 - 20") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
```

Table 10: Clean Training Data Fields 15 - 20

gt1base	bsingle	pbh	pbhr	pbw	fdpe
246	1199	6.480277	6.461538	6.482518	0.1473788
431	908	1.005975	1.005263	1.005839	0.8031088
404	973	1.000000	1.000000	1.000000	0.8742857
343	1044	1.006489	1.010417	1.006652	0.9512195
315	982	1.000000	1.000000	1.000000	1.2173913

We also made changes to the original variables. Altering all fields including our ratio by fixing missing values, adjusting the data distribution, changing the their names, and removing outliers. These actions have reduced the size of the data. The first few rows of the original training data after preparation are provided for reference and labeled as cleaned via their table's title.

```r
df.tbl.1 <- df[1:5, 1:4] # raw tdata only
# head(df[1:13]) # sample edata only
dtbl.1 <- kbl(df.tbl.1, booktabs = T, caption = "Clean Training Data Fields 1 - 4") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
df.tbl.3 <- df[1:5, 5:8] # raw tdata only
# head(df[1:13]) # sample edata only
dtbl.3 <- kbl(df.tbl.3, booktabs = T, caption = "Clean Training Data Fields 5 - 8") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
df.tbl.4 <- df[1:5, 9:12] # raw tdata only
# head(df[1:13]) # sample edata only
dtbl.4 <- kbl(df.tbl.4, booktabs = T, caption = "Clean Training Data Fields 9 - 12") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
df.tbl.5 <- df[1:5, 13:14] # raw tdata only
# head(df[1:13]) # sample edata only
dtbl.5 <- kbl(df.tbl.5, booktabs = T, caption = "Clean Training Data Fields 13 - 14") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
```

Model Building

Types

There are many kinds of models that could be used to predict this data. However, the best model is not clear. To determine which models are best we will assess the training data through multiple models and compare. Those models include a kitchen sink approach, high correlations model, ratio model, and a polynomial model with a twist. We refrain from ensemble modeling to avoid overcomplication or accidentally introducing errors in the prediction.

Reasoning

Our first choice of model is the kitchen sink. This was chosen because it can act as a sort of preliminary test to see what patterns arise when we pass all the variables through it. It is the same model run in the exploration exercises yet, this time pulls from cleaned training data. As the saying goes, we are throwing everything but the kitchen sink at it to see what works. This should help inform our choices for the remaining models. Our second and third models are the high correlation and ratio models. High correlation model is just that, a model with only the top 50% of variables that correlated better than the other variables with the target variable wins. The ratio model follows the same principle by taking the computed ratios and assessing how well they predict the target. We do not expect either of these models to produce better results than the kitchen sink model due to the nature of the predictors; most notably that correlation does not equate to causation. However, they will help inform us of the outcome if one judged the likelihood of games by their most phenomenal characteristics.

Last is the polynomial model, but with a twist. We are going to raise the order of the polynomial sequentially from the first to the third degree then instead of running the model altogether, tell it to pick the variables that result in the highest coefficients of determination. The twist here is in the selection process for the polynomial. Since it is run forwards and backwards to determine the best fit, this should be a good choice for prediction if there is indeed a linear relationship present.

Creation

The first model is created following the kitchen sink method. Subsequent aforementioned models are run thereafter. A summary of the results is displayed below each model containing the errors for each predictor, residual summaries (min, median, max), respective coefficients of determination, p-values, f-statistics, standard errors and the like. An analysis of these is to follow.

```r
# Kitchen sink
model.ks <- lm(win~., df)
summary(model.ks)
```

```
##
## Call:
## lm(formula = win ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.966  -8.055  -0.016   8.039  66.645
##
## Coefficients: (3 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 116.628153 857.859972   0.136 0.891872
## bhit          0.044995   0.003850  11.688  < 2e-16 ***
## bdouble      -0.026595   0.009422  -2.823 0.004807 **
## btriple       0.096432   0.017548   5.495 4.36e-08 ***
## bhomerun      0.012773   0.032299   0.395 0.692547
## bwalk         0.030271   0.008345   3.628 0.000293 ***
## bstrikeout   -0.021044   0.003637  -5.787 8.23e-09 ***
## stolenbase    0.027267   0.004195   6.499 9.99e-11 ***
## phit                NA         NA      NA       NA
## phomerun      0.037582   0.028790   1.305 0.191900
## pwalk        -1.172939   0.566361  -2.071 0.038476 *
## pstrikeout    0.480884   0.093710   5.132 3.13e-07 ***
## fielderror  -82.889511 852.282744  -0.097 0.922532
## fielddp      -0.208871   0.037336  -5.594 2.49e-08 ***
## gt1base             NA         NA      NA       NA
## bsingle             NA         NA      NA       NA
## pbh          -1.377182   0.552550  -2.492 0.012762 *
## pbhr         -1.958194   1.048096  -1.868 0.061850 .
## pbw           0.015477   0.895341   0.017 0.986210
## fdpe         13.726684   5.805135   2.365 0.018138 *
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 12.8 on 2167 degrees of freedom
## Multiple R-squared:   0.33,  Adjusted R-squared:  0.325
## F-statistic:  66.7 on 16 and 2167 DF,  p-value: < 2.2e-16
```

As a general rule, we rank these models with their residuals and standard errors since our main goal is prediction of wins. The first results from this kitchen sink model are promising. A residual standard error of 12.8 is not bad given the amount of variation in the data. The median residual value is very close to zero too, hovering around -0.016. However, our coefficients of determination are not that great resting at about 0.33 and 0.325 adjusted. This is not a good sign for any models relying on strength of correlation to explain their relationships. Results from the high correlation model are shown below.

```
# High correlations model
model.hc <- lm(win~ phit + bwalk + phomerun + fielderror, df)
summary(model.hc)
```

```
##
## Call:
## lm(formula = win ~ phit + bwalk + phomerun + fielderror, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -64.858  -9.749   0.597   9.705  77.333
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.513e+02  1.757e+02  -1.430   0.1528
## phit                NA         NA      NA       NA
## bwalk        2.677e-02  3.164e-03   8.460  < 2e-16 ***
## phomerun     3.854e-02  7.772e-03   4.959 7.64e-07 ***
## fielderror   3.125e+02  1.737e+02   1.799   0.0722 .
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 15.04 on 2180 degrees of freedom
## Multiple R-squared:  0.07035,    Adjusted R-squared:  0.06907 
## F-statistic: 54.99 on 3 and 2180 DF,  p-value: < 2.2e-16
```

As expected, the results are worse-off at describing the patterns of wins. Consider the range of erorrs possible in the residuals. On this model they are frequently somewhere between -64 and 77 with a median of about 0.597. Compared to the kitchen sink run, this model is telling us it knows less than typical random variation and based on the p-value of each predictor, it is quite confident these results are significant. Fortunately, this did show us that occurrences of events where the pitcher throws a ball to intentionally have the batter hit it, is irrelevant for this model due to singularities. Ideally, we want some ratio to solve this singularity issue.

```
# Ratio model
model.rm <- lm(win~ pbh + pbw + pbhr +  fdpe + gt1base, df)
summary(model.rm)
```

```
## 
## Call:
## lm(formula = win ~ pbh + pbw + pbhr + fdpe + gt1base, data = df)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -55.589  -9.263   0.449   9.488  62.688 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 55.847999   1.843514  30.294  < 2e-16 ***
## pbh         -2.478428   0.552902  -4.483 7.76e-06 ***
## pbw         -0.267745   0.684900  -0.391    0.696    
## pbhr         1.215926   0.813274   1.495    0.135    
## fdpe        -4.581644   0.894496  -5.122 3.29e-07 ***
## gt1base      0.077559   0.004615  16.804  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 14.31 on 2178 degrees of freedom
## Multiple R-squared:  0.1592, Adjusted R-squared:  0.1573 
## F-statistic: 82.51 on 5 and 2178 DF,  p-value: < 2.2e-16
```

From the ratio model we see a standard error in the residuals of 14.31 with a median residual of 0.449 and range of about 119 (-55.6 from 62.7). This produces results closer to our true values of wins than the high correlation model but more importantly, it solves our singularity issues. Knowing this means we can test the polynomial model for patterns and expect a truthful resolve. By small margins, the ratio model also appears to have better coefficients of determination and indicates that the ratio of these events could predict better than strength of correlation alone. This is what we expected, but we believe there may be a chance to improve.

```
# Polynomial model
model.pm <- lm(win~
               + bhit
               + bdouble
               + btriple
               + bhomerun
               + bwalk
               + bstrikeout
               + stolenbase
               + phit
               + phomerun
               + pwalk
               + pstrikeout
               + fielderror
               + fielddp
               + gt1base
               + bsingle
               + pbh
               + pbhr
               + pbw
               + fdpe
               # 2nd degree
               + I(bhit^2)
               + I(bdouble^2)
               + I(btriple^2)
               + I(bhomerun^2)
               + I(bwalk^2)
               + I(bstrikeout^2)
               + I(stolenbase^2)
               + I(phit^2)
               + I(phomerun^2)
               + I(pwalk^2)
               + I(pstrikeout^2)
               + I(fielderror^2)
               + I(fielddp^2)
               + I(gt1base^2)
               + I(bsingle^2)
               + I(pbh^2)
               + I(pbhr^2)
               + I(pbw^2)
               + I(fdpe^2)
               # 3rd degree
               + I(bhit^3)
               + I(bdouble^3)
               + I(btriple^3)
               + I(bhomerun^3)
               + I(bwalk^3)
               + I(bstrikeout^3)
               + I(stolenbase^3)
               + I(phit^3)
               + I(phomerun^3)
               + I(pwalk^3)
               + I(pstrikeout^3)
               + I(fielderror^3)
               + I(fielddp^3)
               + I(gt1base^3)
               + I(bsingle^3)
               + I(pbh^3)
               + I(pbhr^3)
               + I(pbw^3)
               + I(fdpe^3)
      ,df)
pm <- stepAIC(model.pm, trace = F, direction = "both")
p <- summary(pm)$call
pm <- lm(p[2], df)
summary(pm)
```

```
##
## Call:
## lm(formula = p[2], data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -39.923  -7.329   0.084   7.336  61.810
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.815e+07  1.923e+07   3.024 0.002525 **
## bdouble          7.817e-01  2.065e-01   3.785 0.000158 ***
## btriple          4.128e-01  5.361e-02   7.700 2.06e-14 ***
## bhomerun        -6.788e-01  2.449e-01  -2.772 0.005613 **
## bwalk           -8.496e-01  9.908e-02  -8.575  < 2e-16 ***
## bstrikeout       7.807e-02  1.296e-02   6.022 2.02e-09 ***
## stolenbase       4.806e-02  4.666e-03  10.301  < 2e-16 ***
## phomerun         7.352e-01  2.045e-01   3.596 0.000331 ***
## pwalk            4.576e+01  9.516e+00   4.809 1.62e-06 ***
## pstrikeout      -8.262e-01  2.017e-01  -4.095 4.37e-05 ***
## fielderror      -1.745e+08  5.751e+07  -3.035 0.002433 **
## fielddp          2.115e-01  9.510e-02   2.224 0.026279 *
## pbhr            -9.858e+00  3.018e+00  -3.266 0.001108 **
## pbw             -4.198e+01  6.898e+00  -6.087 1.36e-09 ***
## fdpe            -1.740e+02  4.586e+01  -3.794 0.000152 ***
## I(bdouble^2)    -2.684e-03  8.516e-04  -3.152 0.001646 **
## I(bhomerun^2)    4.840e-03  1.292e-03   3.746 0.000184 ***
## I(bwalk^2)       1.200e-03  1.546e-04   7.765 1.26e-14 ***
## I(bstrikeout^2) -4.797e-05  6.420e-06  -7.472 1.14e-13 ***
## I(phomerun^2)   -3.125e-03  1.017e-03  -3.073 0.002148 **
## I(pwalk^2)      -1.720e+00  4.324e-01  -3.977 7.21e-05 ***
## I(pstrikeout^2)  3.668e-03  1.420e-03   2.583 0.009858 **
## I(fielderror^2)  1.746e+08  5.733e+07   3.046 0.002344 **
## I(gt1base^2)    -2.386e-04  6.392e-05  -3.733 0.000194 ***
## I(bsingle^2)     1.946e-05  1.527e-06  12.742  < 2e-16 ***
## I(pbh^2)        -1.714e+02  4.884e+01  -3.511 0.000456 ***
## I(pbw^2)         1.744e+02  4.882e+01   3.572 0.000362 ***
## I(fdpe^2)        8.793e+01  2.428e+01   3.621 0.000300 ***
## I(bdouble^3)     3.991e-06  1.046e-06   3.816 0.000139 ***
## I(btriple^3)    -2.845e-06  9.257e-07  -3.074 0.002139 **
## I(bhomerun^3)   -7.944e-06  2.580e-06  -3.080 0.002099 **
## I(bwalk^3)      -5.925e-07  9.377e-08  -6.319 3.20e-10 ***
## I(phomerun^3)    4.537e-06  1.792e-06   2.532 0.011401 *
## I(pwalk^3)       2.595e-02  6.500e-03   3.992 6.76e-05 ***
## I(fielderror^3) -5.825e+07  1.905e+07  -3.058 0.002258 **
## I(pbh^3)         6.352e+00  1.807e+00   3.516 0.000447 ***
## I(pbhr^3)        6.876e-02  3.869e-02   1.777 0.075648 .
## I(pbw^3)        -6.426e+00  1.804e+00  -3.562 0.000376 ***
## I(fdpe^3)       -1.745e+01  5.451e+00  -3.202 0.001387 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.78 on 2145 degrees of freedom
## Multiple R-squared:  0.4381, Adjusted R-squared:  0.4282
## F-statistic: 44.01 on 38 and 2145 DF,  p-value: < 2.2e-16
```

At the moment, this model is poised to predict our target better than any other. In this case, since the relationships among the predictors were highly varied and often weakly correlated, adding a route for predictors to have more than one pattern for each variable seemed to improve our accuracy. We had the smallest range of potential error between 61.8 and -39.9, lowest f-statistic overall at about 44, the lowest error in our standard errors. The only condition where the kitchen sink model has the statistical edge is with its coefficients of determination. However, these may be unconnected to making accurate predictions.

Model Selection

Domain

There are some interesting conundrums that occur when considering this model from a baseball perspective. People often equate things that could cause a team to lose runs or miss opportunities to get someone out with an increased probability of loss in the game. However, not all aspects of this model prove that to be true. In fact, there are multiple places where this idea fails.

For example, allowing hits by a pitcher is correlated well with increased numbers of game wins. This is contrary to preconceived notions of baseball because allowing the batter to hit the ball is generally thought to be a bad thing. Ideally, the winning team would pitch a no-hitter, or game when the batter never hits the ball. Our model suggests otherwise. That you are more likely to win, if the pitcher allows the batter to hit the ball more frequently.

Another interesting concept is the notion that strikeouts, double plays, and steals could increase the chances of victory over another team. This is often false. Strikeouts especially, by batter or pitcher do not seem to improve your chances at all and double plays and steals also decrease your chances of winning. Meanwhile, walks that are allowed by the pitcher improve the chance of winning.

These peculiarities suggest that there is a lot of interaction between variables, which may lead to at least a partially subdued level of independence. Being one of our main assumptions to perform the analysis, this could present a major flaw in prediction capabilities.

Consider the following situation between two teams on a field. Sometimes, those teams whose batters get more than one person on base creates a stressful, high pressure environment for the opposing team to make a play. While under this circumstantial pressure, there are increased odds of getting at least one person out. In practice, this may increase the chances of loss for the batting team because the stakes are higher.

Constrastingly, if no one were on base, there is a higher chance of losing a point to the other team from a single, double, triple, or homerun. The stakes are not set in the game until the bases are at least partially loaded and thus, no 'winning' can occur without high stakes because one team cannot lose (or at least refuse to gain traction in scoring). However, most of the games are low-risk interactions between the batter and pitcher which would resemble an eb-and-flow pattern rather than straight predictions.

Such an atmosphere suggests that, while much of the game depends on the relationship between the pitchers and batters, the interacting variables, such as how many people are on base, how many outs there are, and plenty of other factors, may significantly alter the course of the game and thus change the outcome without changing the variables given here.

There is a certain level of complexity that we are not considering in this model. We are also under the assumption that there are good teams and bad teams, which given the normality of the distribution of games won, we seem to be correct. However, this phenomena in baseball of being partially good teams and bad teams and partially circumstantial probability applies elsewhere. Each individual game has its own set of factors that would have to be considered in detail to predict the outcome because there is too great of an opportunity for deviation from normal.

We leave the model prediction at this thought broken down from a single piece of the sport. What if the game relied solely on fielders to ensure outs from each hit by a batter and they were successful 99% of the time. We believe no progress could be made by either team and thereby no winners so the random variation in the distribution of these predictors is quite literally the game itself.

Best Models

For evaluation purposes, we determined how many errors in the predicted values there were from the models and how far they were from the actual value. We reviewed the residuals, standard errors, root mean squared errors (RMSE), coefficients of determination, and other figures. We pulled the predicted values from each model, created their own value set in the data frame, then ran equations to calculate the errors, their magnitude, and find the RMSE.

```r
# Calculate prediction values
ks.predicted <- predict(model.ks, df)
hc.predicted <- predict(model.hc, df)
rm.predicted <- predict(model.rm, df)
pm.predicted <- predict(model.pm, df)

# Add them to the data frame
df$ks.p <- data.frame(ks.predicted)
df$hc.p <- data.frame(hc.predicted)
df$rm.p <- data.frame(rm.predicted)
df$pm.p <- data.frame(pm.predicted)


# Calculate errors, magnitude, rmse
df <- df %>%
  mutate(ks.error = win - ks.predicted) %>%
  mutate(hc.error = win - hc.predicted) %>%
  mutate(rm.error = win - rm.predicted) %>%
  mutate(pm.error = win - pm.predicted) %>%
  mutate(ks.mag = ks.error^2) %>%
  mutate(hc.mag = hc.error^2) %>%
  mutate(rm.mag = rm.error^2) %>%
  mutate(pm.mag = pm.error^2) %>%
  mutate(ks.eravg = mean(ks.mag)) %>%
  mutate(hc.eravg = mean(hc.mag)) %>%
  mutate(rm.eravg = mean(rm.mag)) %>%
  mutate(pm.eravg = mean(pm.mag)) %>%
  mutate(ks.rmse = sqrt(ks.eravg)) %>%
  mutate(hc.rmse = sqrt(hc.eravg)) %>%
  mutate(rm.rmse = sqrt(rm.eravg)) %>%
  mutate(pm.rmse = sqrt(pm.eravg))
```

Rather than dig through the lists of information in the data frame, we organized it into this table. Each model was given a shortened moniker, ‘KS’ for kitchen sink, ‘HC’, for high correlation, ‘RM’ for ratio model and 'PM' for polynomial model respectively. These metric s were discussed previously but we will revist here. The statistical significance of each model in its p-value, for all intensive purposes, is the same at roughly zero, or below an alpha level of 0.0001.

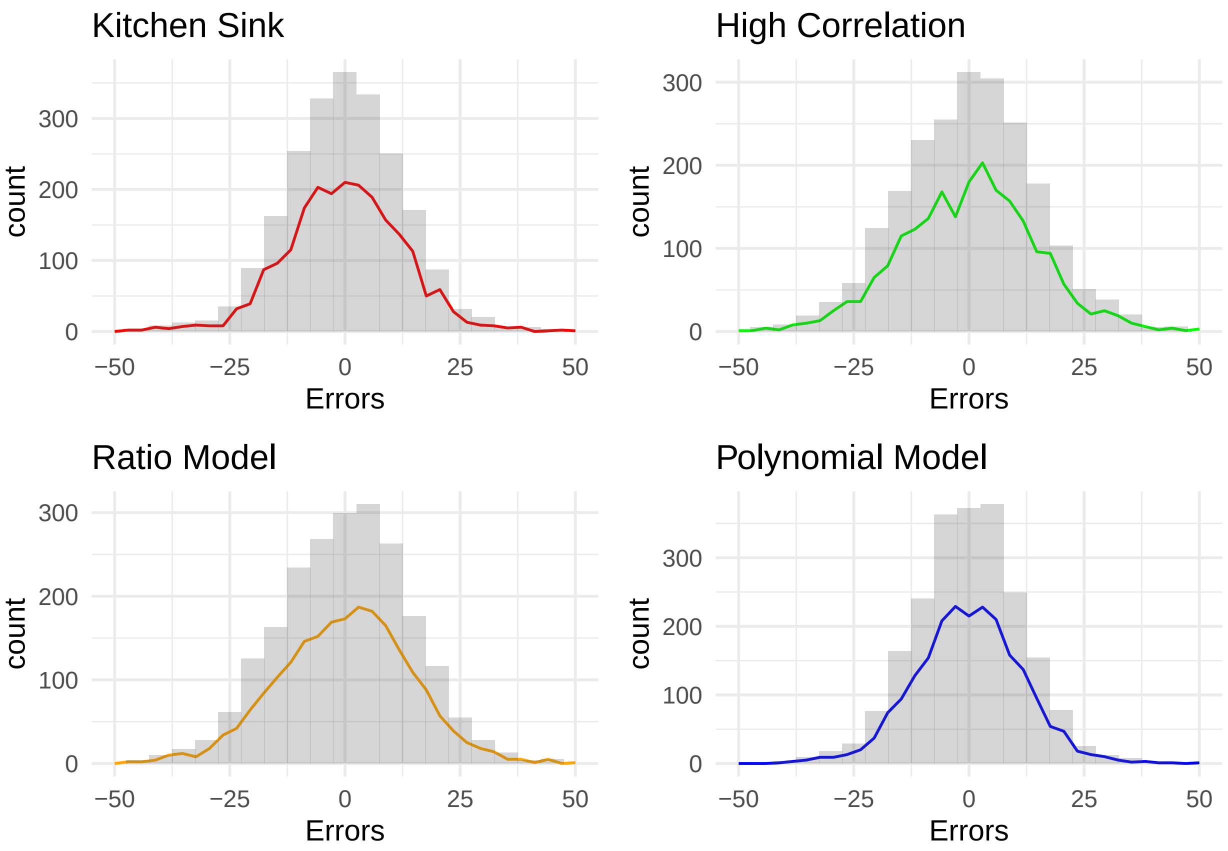
```r
header <- c("mod", "res.min", "res.med", "res.max", "r", "adj.r", "res.se","f")
matrix <- matrix(c(
  "KS", -51.966, -0.016, 66.645, 0.330, 0.325, 12.8, 66.7,
  "HC", -64.858, 0.597, 77.333, 0.07035, 0.06907, 15.04, 54.99,
  "RM", -55.589, 0.449, 62.688, 0.1590, 0.16, 14.31, 82.51,
  "PM", -39.923, 0.084, 61.810, 0.44, 0.43, 11.78, 44.01), ncol = 8,  byrow = TRUE)
models.performance <- as.data.frame(matrix)
colnames(models.performance) <- header
kbl(models.performance, booktabs = T, caption = "Model Performance") %>%
  kable_styling(latex_options = c("striped", "hold_position"),
  full_width = F)
```

Table 11: Model Performance

mod	res.min	res.med	res.max	r	adj.r	res.se	f
KS	-51.966	-0.016	66.645	0.33	0.325	12.8	66.7
HC	-64.858	0.597	77.333	0.07035	0.06907	15.04	54.99
RM	-55.589	0.449	62.688	0.159	0.16	14.31	82.51
PM	-39.923	0.084	61.81	0.44	0.43	11.78	44.01

Our criteria for the best model is the one that produced the lowest errors and predicted the closest average value while maintaining residual values closest to zero. In this case, the polynomial model with its features that prioritize improving the coefficients of determination, is the one that satisfies this criteria. This is reaffirmed in the distribution of errors for each model type below.

```r
kser.hist <- ggplot(df, aes(ks.error)) +
  geom_freqpoly(bins=35, color="red") +
  geom_histogram(alpha=.25, binwidth = 5)  +
  xlim(-50, 50) +
  ggtitle("Kitchen Sink") +
  xlab("Errors")

hcer.hist <- ggplot(df, aes(hc.error)) +
  geom_freqpoly(bins=35, color="green") +
  geom_histogram(alpha=.25, binwidth = 5) +
  xlim(-50, 50) +
  ggtitle("High Correlation") +
  xlab("Errors")

rmer.hist <- ggplot(df, aes(rm.error)) +
  geom_freqpoly(bins=35, color="orange") +
  geom_histogram(alpha=.25, binwidth = 5) +
  xlim(-50, 50) +
  ggtitle("Ratio Model") +
  xlab("Errors")

pmer.hist <- ggplot(df, aes(pm.error)) +
  geom_freqpoly(bins=35, color="blue") +
  geom_histogram(alpha=.25, binwidth = 5) +
  xlim(-50, 50) +
  ggtitle("Polynomial Model") +
  xlab("Errors")

ggarrange(kser.hist, hcer.hist, rmer.hist, pmer.hist)
```

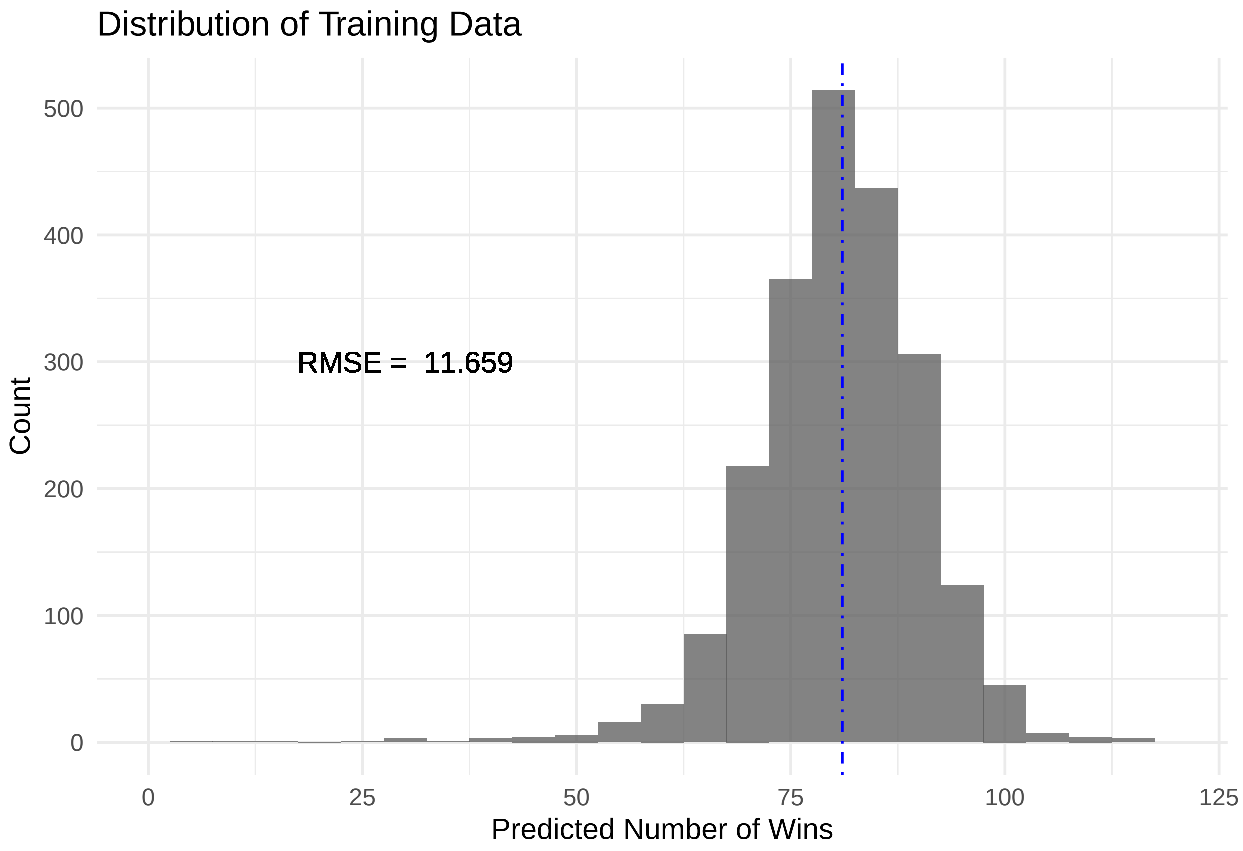


The best distribution is the polynomial model with a twist. It is most concentrated around the true median, has the lowest amount and magnitude of errors with residuals close to zero. It also has the second largest coefficients of determination, the best f-statistic, and maintains a linear line of best fit closest to our actual values.

Prediction

Our prediction depends on the data set supplied. Given that we sampled the original evaluation data to create an evaluation data set 'edata' we can supply an estimated win in a few steps. With the decision made on the best model, we need only reapply the process above on this new data. As a reminder, the original data set produced a distribution of scenarios where the average number of games won is about 81 per season.

```
pm.values.t <- df$pm.p$pm.predicted
median.pm.values <- median(pm.values.t)
t.hist <- ggplot(df$pm.p, aes(pm.predicted)) +
  geom_histogram(alpha=0.75, binwidth = 5) +
  xlim(0, 120) +
  geom_vline(xintercept=median.pm.values, color="blue", linetype="dotdash") +
  annotate("text",x=30,y=300,
    label = paste("RMSE = ", round(df$pm.rmse, digits=3))) +
  ggtitle("Distribution of Training Data") +
  xlab("Predicted Number of Wins") +
  ylab("Count")
t.hist
```



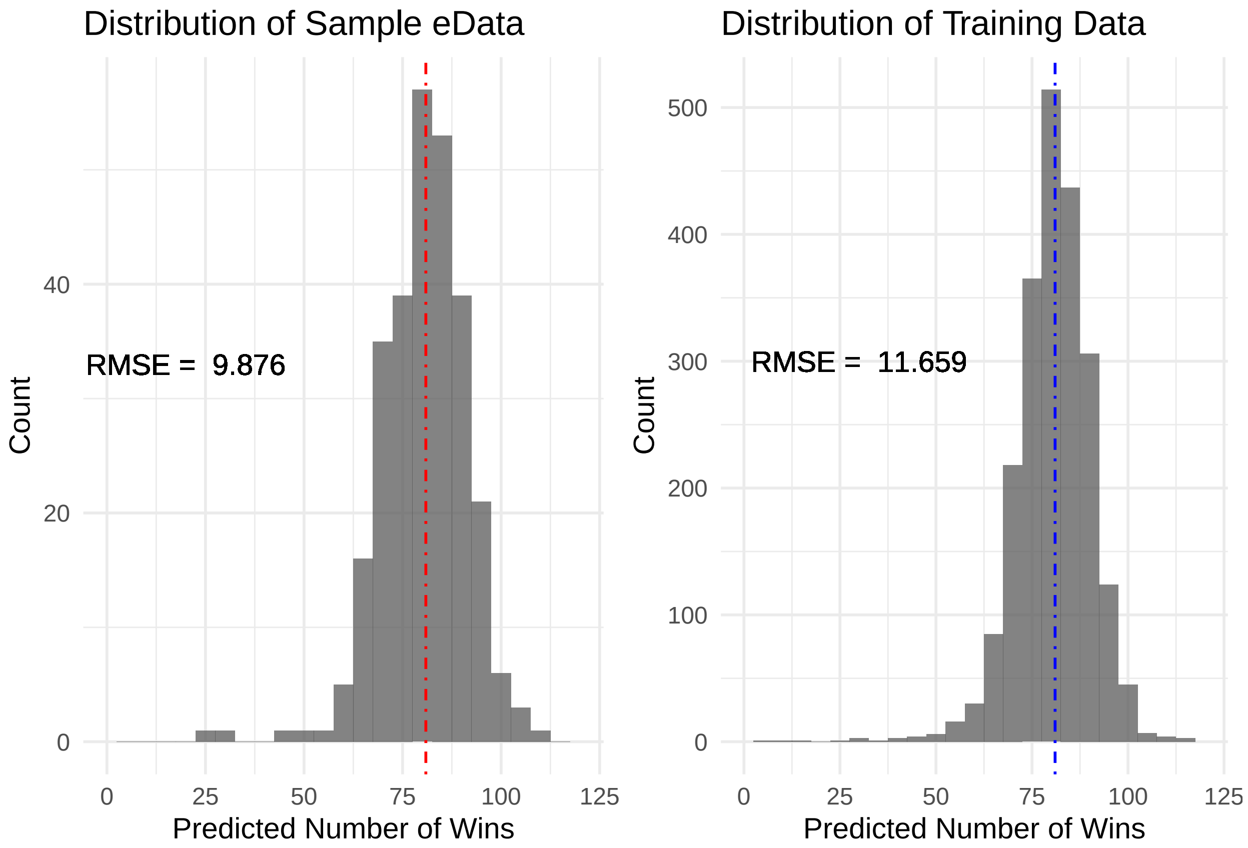
From the data preparation section we sampled the training data for testing and stored the values in an object called 'edata'. Now, we are at the point of testing and will recall this data set. We repeat the process with this new evaluation data set.

The average value of the true training data was about 81 wins per season. In our new predicted data with the evaluation data set, the average comes out to about 80 games won in a season. The root mean errors squared for the evaluation data is about 9.876 and for the training data 11.659. If we had to estimate the number of games that a team will win in each season, it would follow the same distribution and likely result in about the same average. Additionally, we can be 95% confident that the average number of games one for the teams is between 79 and 82 games in a 162 game season.

```
# repeat process with sample/edata

# select all but index, batting_hbp, baserun_cs
# replace missing values with the median value for the distribution
edata <- edata %>%
  rename("win" =TARGET_WINS, # For use with edata sampling
         "bhit" =TEAM_BATTING_H,
         "bdouble"  = TEAM_BATTING_2B,
         "btriple"  = TEAM_BATTING_3B,
         "bhomerun"  = TEAM_BATTING_HR,
         "bwalk"  = TEAM_BATTING_BB,
         "bstrikeout"  = TEAM_BATTING_SO,
         "stolenbase"  = TEAM_BASERUN_SB,
         "caught"  = TEAM_BASERUN_CS,
         "bhitbyp"  = TEAM_BATTING_HBP,
         "phit"  = TEAM_PITCHING_H,
         "phomerun"  = TEAM_PITCHING_HR,
         "pwalk"  = TEAM_PITCHING_BB,
         "pstrikeout"  = TEAM_PITCHING_SO,
         "fielderror"  = TEAM_FIELDING_E,
         "fielddp"  = TEAM_FIELDING_DP)


# drops insignificant columns
# replaces NA with median (given a removal of missing values in calculation)
# df <- tdata[c(2:9,12:17)]          # use if raw tdata
# df <- edata[c(1:7,10:15)] # use with sampled tdata excluding win variable
df <- edata[c(1:8,11:16)] # use with edata
# produces 13 columns/fields with sampled tdata excluding win variable
# produces 15 columns/fields with raw tdata excluding win variable
for (i in colnames(df)) {
  df[[i]][is.na(df[[i]])] <- median(df[[i]], na.rm=TRUE)
}



# remove outliers based on IQR
for (i in colnames(df)) {
  iqr <- IQR(df[[i]])
  q <- quantile(df[[i]], probs = c(0.25, 0.75), na.rm = FALSE)
  qupper <- q[2]+1.5*iqr
  qlower <- q[1]+1.5*iqr
  outlier_free <- subset(df, df[[i]] > (q[1] - 1.5*iqr) & df[[i]] < (q[2]+1.5*iqr) )
}
df <- outlier_free


# Create new estimates from stats
df <- df %>%
  # Hits resulting in greater than one base
  mutate(gt1base = bdouble + btriple + bhomerun) %>%
  # Hits resulting in one base
  mutate(bsingle = bhit - gt1base)



# Compute predictor ratios
df <- df %>%
  mutate(pbh = phit/bhit) %>%
  mutate(pbhr = phomerun/bhomerun) %>%
  mutate(pbw = pwalk/bwalk) %>%
  mutate(fdpe = fielddp/fielderror)


# Applying Yeo-Johnson transformation
df$fielderror <- yeo.johnson(df$fielderror, lambda = -0.9878936, derivative = 0, epsilon = sqrt(.Machin
df$pwalk <- yeo.johnson(df$pwalk, lambda = 0.301843, derivative = 0, epsilon = sqrt(.Machine$double.eps
df$phit <- yeo.johnson(df$phit, lambda = -3.125283, derivative = 0, epsilon = sqrt(.Machine$double.eps)
df$pstrikeout <- yeo.johnson(df$pstrikeout, lambda = 0.4299074, derivative = 0, epsilon = sqrt(.Machine


# Reassign median values to new ratios where NA value is present
for (i in colnames(df)) {
  df[[i]][is.na(df[[i]])] <- median(df[[i]], na.rm=TRUE)
}


# Polynomial model
model.pm <- lm(win~
               + bhit
               + bdouble
               + btriple
               + bhomerun
               + bwalk
               + bstrikeout
               + stolenbase
               + phit
               + phomerun
               + pwalk
               + pstrikeout
               + fielderror
               + fielddp
               + gt1base
               + bsingle
               + pbh
               + pbhr
               + pbw
               + fdpe
               # 2nd degree
               + I(bhit^2)
               + I(bdouble^2)
               + I(btriple^2)
               + I(bhomerun^2)
               + I(bwalk^2)
               + I(bstrikeout^2)
               + I(stolenbase^2)
               + I(phit^2)
               + I(phomerun^2)
               + I(pwalk^2)
               + I(pstrikeout^2)
               + I(fielderror^2)
               + I(fielddp^2)
               + I(gt1base^2)
               + I(bsingle^2)
               + I(pbh^2)
               + I(pbhr^2)
               + I(pbw^2)
               + I(fdpe^2)
               # 3rd degree
               + I(bhit^3)
               + I(bdouble^3)
               + I(btriple^3)
               + I(bhomerun^3)
               + I(bwalk^3)
               + I(bstrikeout^3)
               + I(stolenbase^3)
               + I(phit^3)
               + I(phomerun^3)
               + I(pwalk^3)
               + I(pstrikeout^3)
               + I(fielderror^3)
               + I(fielddp^3)
               + I(gt1base^3)
               + I(bsingle^3)
               + I(pbh^3)
               + I(pbhr^3)
               + I(pbw^3)
               + I(fdpe^3)
      ,df)
pm <- stepAIC(model.pm, trace = F, direction = "both")
p <- summary(pm)$call
pm <- lm(p[2], df)


# Calculate prediction values
pm.predicted <- predict(model.pm, df)

# Add them to the data frame
df$pm.p <- data.frame(pm.predicted)


# Calculate errors, magnitude, rmse
df <- df %>%
  mutate(pm.error = win - pm.predicted) %>%
  mutate(pm.mag = pm.error^2) %>%
  mutate(pm.eravg = mean(pm.mag)) %>%
  mutate(pm.rmse = sqrt(pm.eravg))
```

```
pmvals <- as.numeric(df$pm.p$pm.predicted)
x <- mean(pmvals)
s <- sd(pmvals)
z <- 1.96 # 95% confidence
```

```
pm.values.p <- df$pm.p$pm.predicted
median.pm.values <- median(pm.values.p)
p.hist <- ggplot(df$pm.p, aes(pm.predicted)) +
  geom_histogram(alpha=0.75, binwidth = 5) +
  xlim(0, 120) +
  geom_vline(xintercept=median.pm.values, color="red", linetype="dotdash") +
  annotate("text",x=20,y=33,
    label = paste("RMSE = ", round(df$pm.rmse, digits=3))) +
  ggtitle("Distribution of Sample eData") +
  xlab("Predicted Number of Wins") +
  ylab("Count")
ggarrange(p.hist, t.hist)
```


Distribution of Sample eData
Count
RMSE = 9.876
Predicted Number of Wins
Distribution of Training Data
Count
RMSE = 11.659
Predicted Number of Wins