R Markdown:

````
---
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE)
```

```{r}
library(broom)
library(kableExtra)
library(psych)
library(tidyverse)
library(VIM)
library(GGally)
library(patchwork)
library(corrplot)
library(rsample)
library(recipes)
library(caret)
library(leaps)
library(car)
library(doParallel)

theme_set(theme_bw())
```

```{r}
moneyball <- read.csv("moneyball.csv", header=T)

moneyball$INDEX <- NULL
```

# Bingo Bonus

I did one bingo bonus challenge which is to compare the results of doing OLS regression using glm and lm in R for 20 points. This is done in the section titled "Bingo Section Details" at the end of this document (after the conclusion).

# Introduction

In this project, we analyzed the Moneyball dataset which consists of 2200 observations representing professional baseball teams from the years 1871 to 2006 inclusive. Each observation has the performance metrics of the team for each year and the statistics were adjusted to match the performance of a 162 game season when needed. Ordinary least squares regression was then use to predict the number of wins of a team based on the given statistics. The dataset also included missing values that had to be imputed in order to avoid losing a significant portion of the data if only observations with complete data were used, or losing features if entire features with missing data were not used. During modeling, multiple models were attempted including the three models shown in this document. Repeated 10-fold cross-validation was used in the model evaluation process. The model that provided the best performance was selected as the final model.

\newpage

# Data Exploration
````

Overview

Our moneyball dataset consists of `r nrow(moneyball)` observations and `r ncol(moneyball)` columns after removing the INDEX variable. We have `r ncol(moneyball)-1` features and 1 outcome variable. All the variables are integers.

Variable Distributions

Let's have a look at the variable names and a quick summary of each variable's distribution:

```{r}
moneyball %>% describe(skew=FALSE, quant=c(0.25,0.5,0.75)) %>% select(-8) %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = c("striped","scale_down"))
```

We can see above that while no variables have negative values which is logical, there are several values that have unrealistically high or low values. For example, the minimum value of TARGET_WINS is `r min(moneyball$TARGET_WINS)` while the maximum value is `r max(moneyball$TARGET_WINS)`. Clearly some preprocessing of the data will be required before the modeling stage.

```{r}
set.seed(373904)

my_split <- initial_split(moneyball, strata="TARGET_WINS")

training <- training(my_split)
testing <- testing(my_split)
```

Visualizations

Let's have a closer look at the variable distributions. However, before we do that, let's not look any further at the full dataset and split our data into a training & a testing set. We should use the testing set only once at the end to evaluate the performance of our finally selected model. We shall stratify by the outcome variable to ensure that both the training and testing sets are representative of the distribution of the outcome variable.

Target Wins (Outcome)

```{r, fig.height=4}
g1 <- ggplot(training, aes(TARGET_WINS)) + geom_histogram(aes(fill="#F8766D"), show.legend = FALSE) +
  labs(title="Distribution of TARGET_WINS")

g2 <- ggplot(training, aes("", TARGET_WINS)) + geom_boxplot(aes(fill="#F8766D"), show.legend = FALSE) +
  coord_flip() +
  theme(axis.title.y = element_blank(), axis.text.y = element_blank(), axis.ticks.y = element_blank())

g1/g2
```

The distribution above looks reasonably close to normal but we can clearly see the presence of outliers. Based on theory as well as the distribution above, it would makes sense to perhaps truncate the outlier values here so that the minimum value is 40 and the maximum value is 115.

Features & Outcome

Let's try and visualize our features' distributions and how they relate to the outcome.

```{r}
temp_for_ggpairs <- training
colnames(temp_for_ggpairs) <- str_replace_all(colnames(temp_for_ggpairs), "TEAM_","")
```

```{r}
ggpairs(temp_for_ggpairs[,c(2:6,1)], lower = list(continuous = wrap("smooth_loess", alpha=0.1, color="blue")))
```

```{r}
ggpairs(temp_for_ggpairs[,c(7:11,1)], lower = list(continuous = wrap("smooth_loess", alpha=0.1, color="blue")))
```

```{r}
ggpairs(temp_for_ggpairs[,c(12:16,1)], lower = list(continuous = wrap("smooth_loess", alpha=0.1, color="blue")))
```

Looking at the plots above, we can see that several variables such as BASERUN_SB & BATTING_3B look highly positively skewed. Some have outliers with a vary high magnitude such as PITCHING_BB & PITCHING_SO. We can see from the loess curves above that the variables' outliers seems to be affecting the curves strongly in their regions and are likely to influence our linear model and lead it astray if not handled appropriately. Additionally, some distributions seem to be bimodal such as BATTING_HR & BATTING_SO.

While no variables show a strong magnitude of correlation with the outcome, the ones that do, show low moderate correlations (eg. BATTING_BB & BATTING_SO), while others seem uncorrelated. Note that the Pearson's correlations shown above are highly susceptible to outliers and these values are therefore likely to change once outliers are handled appropriately.

\newpage

Correlations

Let's have a look at the correlations between all the variables.

```{r,fig.height=8, fig.width=8}
cors <- drop_na(temp_for_ggpairs) %>% cor()
corrplot(cors, method="number", order="hclust")
```

We can see that as expected, some of the variables tend to group together with similar correlation values to one another and to the outcome variable such as BATTING_2B, BATTING_H, & PITCHING_H for example. However, the above correlation values are not exactly right since we've had to discard rows with any missing values in order to generate it. The correlation values shown earlier in the Features & Outcome section were more accurate. However, this plot gives us a good idea of how the variables group together in their relationships. Speaking of missing values, let's investigate the missing data in our dataset.

\newpage

Missing Data

Let's have a look at the proportion of missing data in our dataset by variable and variable combination. We'll look at the entire dataset here rather than the training data only since if there are tiny amounts of missing data, they might get assigned to the testing set and we would fail to detect their presence if we only look at the training set.

```{r}
```

```
aggr(moneyball, cex.axis=0.45, numbers=TRUE)
```

We can see above that only about 8% of observations have complete data across all the variables. There are 6 variables with missing data with the variable TEAM_BATTING_HBP having the highest proportion of missing values at over 90%! Next, is the TEAM_BASERUN_CS variable which has over 30% missing. The remaining variables TEAM_FIELDING_DP, TEAM_PITCHING_SO, TEAM_BASERUN_SB, and TEAM_BATTING_SO have smaller proportions of values missing. A notable combination of missing data is the missingness of both TEAM_BATTING_HBP and TEAM_BASERUN_CS together and this is the case with about 15% of the observations.

Let's have a quick look to see if missingness seems to be informative about the outcome. Let's investigate TEAM_BASERUN_CS & TEAM_FIELDING_DP. We'll look in the training set now since we don't want to leak information from the testing set.

```{r}
marginplot(training[, c("TEAM_BASERUN_CS", "TARGET_WINS")])
```

```{r}
marginplot(training[, c("TEAM_FIELDING_DP", "TARGET_WINS")])
```

We can see above that with both of these variables, the missingness of the variable seems to have a wider distribution of TARGET_WINS. And there tends to be some extreme values in both the negative and the positive direction. Both distributions (missing and non-missing) however seem to be centered identically with equal medians of TARGET_WINS. Doing this exercise with the other variables either showed similar results or was unremarkable.

It would be a good idea to impute the missing values of the variables to be able to use all the observations in the linear model and avoid throwing away valuable data.

\newpage

Data Preparation

Fixing Missing Values

We have imputed missing values for the 6 variables which suffer from them using mean imputation in the first model and using bagged tree models for the second and third (final) models. All other variables including other variables with missing predictors are used to predict the missing values of a variable with a bagged tree model.

Mathematical Transformations

The log transformation was used on three predictors used in the second and third (final) models: TEAM_BATTING_3B, TEAM_FIELDING_E, and TEAM_PITCHING_BB. This is because those variables tended to be positively skewed and the log transformation with an exponential base was used to reduce that skewness. As a demonstration, the original and log transformed distributions of the TEAM_BATTING_3B variable are shown below. A constant of 1 was added to all values before the log transformation. This transformation can help bring the values closer together which reduces the effects of outliers and brings the distribution closer to the normal distribution.

```{r, fig.height=5.5}
g3 <- ggplot(training, aes(TEAM_BATTING_3B)) + geom_histogram(aes(fill="#00BFC4"), show.legend = FALSE) +
  labs(title="Original Distribution")
```

```
g4 <- ggplot(training, aes(NULL, TEAM_BATTING_3B)) + geom_boxplot(aes(fill="#00BFC4"),
show.legend=FALSE) +
  coord_flip() +
  theme(axis.title.y = element_blank(), axis.text.y = element_blank(), axis.ticks.y = element_blank())

g5 <- ggplot(training,(aes(TEAM_BATTING_3B, TARGET_WINS))) + geom_point(alpha=0.3) +
geom_smooth(method="lm", se=FALSE)

g3/g4/g5
```

```{r, fig.height=5.5}
g6 <- ggplot(training, aes(log((TEAM_BATTING_3B+1)))) + geom_histogram(aes(fill="#00BFC4"), show.legend
= FALSE) +
  labs(title="log Transformed Distribution")

g7 <- ggplot(training, aes(NULL, log((TEAM_BATTING_3B+1)))) + geom_boxplot(aes(fill="#00BFC4"),
show.legend=FALSE) +
  coord_flip() +
  theme(axis.title.y = element_blank(), axis.text.y = element_blank(), axis.ticks.y = element_blank())

g8 <- ggplot(training,(aes(log(TEAM_BATTING_3B+1), TARGET_WINS))) + geom_point(alpha=0.3) +
geom_smooth(method="lm", se=FALSE)

g6/g7/g8
```

Note how the skewness has reduced after the transformation and there are only 3 outliers now (although one is quite extreme).

Truncating Variables

Seven variables including the outcome variable were truncated to reduce the effect of outliers. The outliers can negatively affect the model and truncating those variables proved beneficial to model performance.

Combining Variables

Many interactions and ratios between predictors were attempted and several of them were retained in the model as they had proved useful. A constant of 1 was added to the numerator and the denominator of those ratios to avoid division by 0 even in the testing set. An interaction between two variables such as their product or the ratio of one over the other can be related to the outcome variable which was the case for our second and third (final) models.

\newpage

Build Models

Several different techniques were attempted to build a good model. First, the manual approach was used where variables that showed good correlation levels with the outcome were tried in the model (after being appropriately imputed / transformed if needed). Multiple models with different transformation steps and different combinations of variables were attempted. However, this manual approach was not able to create a model with an acceptable performance level and several model iterations only marginally improved performance.

Next, I looked into building new variables as combinations of predictors and two such variables (both ratios) were identified and they proved useful and ended up being used in the final model. However, that alone wasn't sufficient. I decided to look at a more brute force approach. I created many new predictors as products of two predictors to try in the model. Due to the large number of variables, I used forward stepwise selection from the leaps package in

order to identify an optimal number of variables and an optimal group of variables to use. In order to avoid overfitting and assess the large number of models effectively, repeated 10-fold cross-validation was done on the training set with a repeat value of 10. This means that 100 test folds were used in assessing every model size (number of variables used ranging from 1 to 30), where the variables at each size are selected with the forward stepwise selection method. The model size selected with this approach however was not far superior to some of the slightly smaller models and had some predictors that were non-significant predictors when trained on the full training set. I manually eliminated the most insignificant predictors one by one and evaluated the repeated k-fold cv performance every time. The performance of the model improved as those predictors were removed. The resulting model was retained as the final one and had 19 predictors.

While I have tried many models, I have selected 3 to demonstrate. The first model was one of the ones created manually early on, the second was also created manually and includes a different selection of variables including 2 variables created as ratios of 2 predictors, and the third model is the final model that was created based on the process explained above. For the three models below, truncation of variables was done identically.

\newpage

Model 1

For the first demonstrated model, mean imputation was used, 6 original predictors from the dataset were used. This was a very basic model created early on after eliminating many variables that did not seem to be related to the outcome and including variables that showed strong statistical significance in relation to the outcome while trying to keep the model relatively small and simple. The model fit results are below.

```{r}
training_mod1<- training %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                    TARGET_WINS>115 ~ as.integer(115),
                                    TRUE ~ as.integer(TARGET_WINS)),

                          TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                    TEAM_BATTING_H>2000 ~ as.integer(2000),
                                    TRUE ~ as.integer(TEAM_BATTING_H)),

                          TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                    TEAM_BATTING_BB>750 ~ as.integer(750),
                                    TRUE ~ as.integer(TEAM_BATTING_BB)),

                          TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                    TRUE ~ as.integer(TEAM_BASERUN_SB)),

                          TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                    TEAM_BASERUN_CS>100 ~ as.integer(100),
                                    TRUE ~ as.integer(TEAM_BASERUN_CS)),

                          TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                    TEAM_PITCHING_BB>800 ~ as.integer(800),
                                    TRUE ~ as.integer(TEAM_PITCHING_BB)),

                          TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                    TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                    TRUE ~ as.integer(TEAM_PITCHING_SO))
                          )
```

```{r}
rec1 <- recipe(TARGET_WINS ~ ., data = training_mod1) %>%
```

```
      step_meanimpute(TEAM_BATTING_HBP, TEAM_BASERUN_CS, TEAM_FIELDING_DP,
TEAM_PITCHING_SO, TEAM_BATTING_SO, TEAM_BASERUN_SB) %>%
      step_rm(TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_BASERUN_CS, TEAM_BATTING_2B,
TEAM_PITCHING_SO,
            TEAM_PITCHING_H, TEAM_PITCHING_BB, TEAM_BATTING_SO, TEAM_BATTING_BB)
```

```{r, comment=NA, cache=TRUE}
cl <- makeCluster(4) # We shall use 4 physical cores in parallel
registerDoParallel(cl)

set.seed(364731, kind = "L'Ecuyer-CMRG") # seed that works with parallel processing

train_ctrl <- trainControl(method = "repeatedcv",
                     number = 10,
                     repeats = 10)

lm_fit_1 <- train(rec1, data=training_mod1,
              method = "lm",
              trControl = train_ctrl)

stopCluster(cl)
registerDoSEQ()
```

```{r}
tidy(summary(lm_fit_1)) %>% rename(Term="term", Estimate="estimate", `Std. Error`="std.error", `t
value`="statistic", `Pr(>|t|)`="p.value") %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = "striped")
```

```{r}
glance(summary(lm_fit_1)) %>% select(-(7:10)) %>% mutate(df=df-1, r.squared=round(r.squared,4),
adj.r.squared=round(adj.r.squared,4)) %>%
  rename(`R Squared`="r.squared", `Adj. R Squared`="adj.r.squared", `Residual Standard Error`="sigma",
                                `F-statistic`="statistic", `p-value`="p.value", Df="df") %>%
  select(`R Squared`:`F-statistic`, Df, `p-value`) %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = "striped")
```

Let's look at the resampling results across the 100 test folds (10 x 10-fold cv).

```{r}
lm_fit_1_results <- tibble(Metric=c("RMSE","Rsquared","MAE"),
                     Mean=c(lm_fit_1$results$RMSE, lm_fit_1$results$Rsquared, lm_fit_1$results$MAE),
                     `Lower 95% C.I.`=c(lm_fit_1$results$RMSE-qnorm(0.975)*(lm_fit_1$results$RMSESD/10),
                                  lm_fit_1$results$Rsquared-qnorm(0.975)*(lm_fit_1$results$RsquaredSD/10),
                                  lm_fit_1$results$MAE-qnorm(0.975)*(lm_fit_1$results$MAESD/10)),
                     `Upper 95% C.I.`=c(lm_fit_1$results$RMSE+qnorm(0.975)*(lm_fit_1$results$RMSESD/10),
                                  lm_fit_1$results$Rsquared+qnorm(0.975)*(lm_fit_1$results$RsquaredSD/10),
                                  lm_fit_1$results$MAE+qnorm(0.975)*(lm_fit_1$results$MAESD/10)))

lm_fit_1_results %>% kable("latex", booktabs=TRUE) %>% add_header_above(c("Model 1 Resampling
Results"=4), bold=TRUE) %>%
  kable_styling(latex_options = "striped")
```

\newpage

Model 2

For the second demonstrated model, bagged trees imputation was used, 10 predictors in total were used, 2 of which were created as ratios while adding a constant of 1 to the numerator and denominator variables to avoid division by 0 in the denominator. Additionally, 3 predictors were log transformed for the model, one of which had a constant of 1 added to avoid getting the log of 0. The model fit results are below. The included predictors were selected to be statistically significantly related to the outcome and I tried to expand on the number of predictors in the model compared to model 1.

```{r}
training_mod2 <- training %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                        TARGET_WINS>115 ~ as.integer(115),
                                        TRUE ~ as.integer(TARGET_WINS)),

                            TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                          TEAM_BATTING_H>2000 ~ as.integer(2000),
                                          TRUE ~ as.integer(TEAM_BATTING_H)),

                            TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                           TEAM_BATTING_BB>750 ~ as.integer(750),
                                           TRUE ~ as.integer(TEAM_BATTING_BB)),

                            TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                           TRUE ~ as.integer(TEAM_BASERUN_SB)),

                            TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                           TEAM_BASERUN_CS>100 ~ as.integer(100),
                                           TRUE ~ as.integer(TEAM_BASERUN_CS)),

                            TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                            TEAM_PITCHING_BB>800 ~ as.integer(800),
                                            TRUE ~ as.integer(TEAM_PITCHING_BB)),

                            TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                            TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                            TRUE ~ as.integer(TEAM_PITCHING_SO))
                            )

training_mod2 <- mutate(training_mod2,
                  TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                  TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
                  TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                  TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                  TEAM_BATTING_3B = TEAM_BATTING_3B + 1)
```

```{r}
rec2 <- recipe(TARGET_WINS ~ ., data = training_mod2) %>%
      step_bagimpute(TEAM_BATTING_HBP, TEAM_BASERUN_CS, TEAM_FIELDING_DP,
TEAM_PITCHING_SO, TEAM_BATTING_SO, TEAM_BASERUN_SB) %>%
      step_ratio(TEAM_BATTING_BB, denom=denom_vars(TEAM_BATTING_SO)) %>%
      step_ratio(TEAM_PITCHING_SO, denom=denom_vars(TEAM_PITCHING_BB)) %>%
      step_log(TEAM_BATTING_3B, TEAM_FIELDING_E, TEAM_PITCHING_BB) %>%
```

```
      step_rm(TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_PITCHING_H, TEAM_BASERUN_CS,
TEAM_BATTING_2B, TEAM_PITCHING_BB, TEAM_PITCHING_SO)
```

```{r, comment=NA, cache=TRUE}
cl <- makeCluster(4)
registerDoParallel(cl)

train_ctrl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 10)

lm_fit_2 <- train(rec2, data=training_mod2,
                  method = "lm",
                  trControl = train_ctrl)

stopCluster(cl)
registerDoSEQ()
```

```{r}
tidy(summary(lm_fit_2)) %>% rename(Term="term", Estimate="estimate", `Std. Error`="std.error", `t
value`="statistic", `Pr(>|t|)`="p.value") %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = "striped")
```

```{r}
glance(summary(lm_fit_2)) %>% select(-(7:10)) %>% mutate(df=df-1, r.squared=round(r.squared,4),
adj.r.squared=round(adj.r.squared,4)) %>%
  rename(`R Squared`="r.squared", `Adj. R Squared`="adj.r.squared", `Residual Standard Error`="sigma",
                                  `F-statistic`="statistic", `p-value`="p.value", Df="df") %>%
  select(`R Squared`:`F-statistic`, Df, `p-value`) %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = "striped")
```

Let's look at the resampling results across the 100 test folds (10 x 10-fold cv).

```{r}
lm_fit_2_results <- tibble(Metric=c("RMSE","Rsquared","MAE"),
                           Mean=c(lm_fit_2$results$RMSE, lm_fit_2$results$Rsquared, lm_fit_2$results$MAE),
                           `Lower 95% C.I.`=c(lm_fit_2$results$RMSE-qnorm(0.975)*(lm_fit_2$results$RMSESD/10),
                                              lm_fit_2$results$Rsquared-qnorm(0.975)*(lm_fit_2$results$RsquaredSD/10),
                                              lm_fit_2$results$MAE-qnorm(0.975)*(lm_fit_2$results$MAESD/10)),
                           `Upper 95% C.I.`=c(lm_fit_2$results$RMSE+qnorm(0.975)*(lm_fit_2$results$RMSESD/10),
                                              lm_fit_2$results$Rsquared+qnorm(0.975)*(lm_fit_2$results$RsquaredSD/10),
                                              lm_fit_2$results$MAE+qnorm(0.975)*(lm_fit_2$results$MAESD/10)))

lm_fit_2_results %>% kable("latex", booktabs=TRUE) %>% add_header_above(c("Model 2 Resampling
Results"=4), bold=TRUE) %>%
  kable_styling(latex_options = "striped")
```

\newpage

Model 3

For the third demonstrated model, bagged trees imputation was used, 19 predictors in total were used including many predictors that were created as products between other predictors. 3 predictors were log transformed for the model, one of which had a constant of 1 added to avoid getting the log of 0. The model fit results are below. Recall that the predictors had been selected as described at the top of this section (Build Models).

```{r}
training_mod3 <- training %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                        TARGET_WINS>115 ~ as.integer(115),
                                        TRUE ~ as.integer(TARGET_WINS)),

                            TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                          TEAM_BATTING_H>2000 ~ as.integer(2000),
                                          TRUE ~ as.integer(TEAM_BATTING_H)),

                            TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                           TEAM_BATTING_BB>750 ~ as.integer(750),
                                           TRUE ~ as.integer(TEAM_BATTING_BB)),

                            TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                           TRUE ~ as.integer(TEAM_BASERUN_SB)),

                            TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                           TEAM_BASERUN_CS>100 ~ as.integer(100),
                                           TRUE ~ as.integer(TEAM_BASERUN_CS)),

                            TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                           TEAM_PITCHING_BB>800 ~ as.integer(800),
                                           TRUE ~ as.integer(TEAM_PITCHING_BB)),

                            TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                           TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                           TRUE ~ as.integer(TEAM_PITCHING_SO))
                            )

training_mod3 <- mutate(training_mod3,
                TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
                TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                TEAM_BATTING_3B = TEAM_BATTING_3B + 1,
                TEAM_PITCHING_HR = TEAM_PITCHING_HR + 1)
```

```{r}
rec3 <- recipe(TARGET_WINS ~ ., data = training_mod3) %>%
    step_bagimpute(TEAM_FIELDING_DP, TEAM_BATTING_SO, TEAM_BASERUN_SB,
TEAM_PITCHING_SO, TEAM_BATTING_HBP, TEAM_BASERUN_CS) %>%
    step_ratio(TEAM_BATTING_BB, denom=denom_vars(TEAM_BATTING_SO)) %>%
    step_ratio(TEAM_PITCHING_SO, denom=denom_vars(TEAM_PITCHING_BB)) %>%
    step_log(TEAM_BATTING_3B, TEAM_FIELDING_E, TEAM_PITCHING_BB) %>%
    step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_3B) %>%
    step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_BB) %>%
    step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_SO) %>%
    step_interact(terms = ~ TEAM_BATTING_H:TEAM_FIELDING_E) %>%
    step_interact(terms = ~ TEAM_BATTING_3B:TEAM_BATTING_BB) %>%
    step_interact(terms = ~ TEAM_BATTING_3B:TEAM_PITCHING_HR) %>%
```

```
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BASERUN_SB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_PITCHING_HR:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_FIELDING_E:TEAM_FIELDING_DP) %>%
      step_interact(terms = ~ TEAM_FIELDING_DP:TEAM_PITCHING_SO_o_TEAM_PITCHING_BB) %>%
      step_rm(TEAM_FIELDING_E, TEAM_FIELDING_DP, TEAM_BATTING_3B, TEAM_PITCHING_HR,
TEAM_BATTING_H,
          TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_PITCHING_H, TEAM_BASERUN_CS,
TEAM_BATTING_2B,
          TEAM_PITCHING_BB, TEAM_PITCHING_SO, TEAM_PITCHING_SO_o_TEAM_PITCHING_BB)
```

```{r, comment=NA, cache=TRUE}
cl <- makeCluster(4)
registerDoParallel(cl)

train_ctrl <- trainControl(method = "repeatedcv",
                   number = 10,
                   repeats = 10)

lm_fit_3 <- train(rec3, data=training_mod3,
             method = "lm",
             trControl = train_ctrl)

stopCluster(cl)
registerDoSEQ()
```

```{r}
tidy(summary(lm_fit_3)) %>% rename(Term="term", Estimate="estimate", `Std. Error`="std.error", `t
value`="statistic", `Pr(>|t|)`="p.value") %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = c("striped","scale_down"))
```

```{r}
glance(summary(lm_fit_3)) %>% select(-(7:10)) %>% mutate(df=df-1, r.squared=round(r.squared,4),
adj.r.squared=round(adj.r.squared,4)) %>%
  rename(`R Squared`="r.squared", `Adj. R Squared`="adj.r.squared", `Residual Standard Error`="sigma",
                                `F-statistic`="statistic", `p-value`="p.value", Df="df") %>%
  select(`R Squared`:`F-statistic`, Df, `p-value`) %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = "striped")
```

Let's look at the resampling results across the 100 test folds (10 x 10-fold cv).

```{r}
lm_fit_3_results <- tibble(Metric=c("RMSE","Rsquared","MAE"),
                   Mean=c(lm_fit_3$results$RMSE, lm_fit_3$results$Rsquared, lm_fit_3$results$MAE),
                   `Lower 95% C.I.`=c(lm_fit_3$results$RMSE-qnorm(0.975)*(lm_fit_3$results$RMSESD/10),
                              lm_fit_3$results$Rsquared-qnorm(0.975)*(lm_fit_3$results$RsquaredSD/10),
                              lm_fit_3$results$MAE-qnorm(0.975)*(lm_fit_3$results$MAESD/10)),
                   `Upper 95% C.I.`=c(lm_fit_3$results$RMSE+qnorm(0.975)*(lm_fit_3$results$RMSESD/10),
```

```
                    lm_fit_3$results$Rsquared+qnorm(0.975)*(lm_fit_3$results$RsquaredSD/10),
                    lm_fit_3$results$MAE+qnorm(0.975)*(lm_fit_3$results$MAESD/10)))

lm_fit_3_results %>% kable("latex", booktabs=TRUE) %>% add_header_above(c("Model 3 Resampling Results"=4), bold=TRUE) %>%
  kable_styling(latex_options = "striped")
```

Note that in our final selected model (model 3), the coefficients of the stand-alone predictors had signs appropriate to what is expected from theory. We can see that TEAM_BATTING_BB & TEAM_BASERUN_SB have positive coefficients which is to be expected as both predictors should have a positive impact on TARGET_WINS theoretically while TEAM_BATTING_SO has a negative coefficient which is also to be expected as it has a negative impact on TARGET_WINS theoretically. The remaining predictors are products of other predictors and are therefore difficult to interpret.

\newpage

Select Models

Regarding the criteria to select the best model, since our focus is on predictive power rather than on inference, we will evaluate performance on 100 test folds by using 10 repeats of 10-fold cross-validation. Our main evaluation metric will be the Root Mean Square Error (RMSE) but we will also look at the R-Squared value (computed on the test set as the square of the correlation between the predictions and the observed values) as well as look at the Mean Absolute Error (MAE). If all metrics agree on a clear winner then we would be more confident with our selection, however the RMSE will be the decisive metric in case they don't.

Let's look at the performance distribution below.


```{r}
mod_1_res <- gather(select(lm_fit_1$resample, -Resample)) %>% mutate(model="Model 1")

mod_2_res <- gather(select(lm_fit_2$resample, -Resample)) %>% mutate(model="Model 2")

mod_3_res <- gather(select(lm_fit_3$resample, -Resample)) %>% mutate(model="Model 3")

mod_res <- bind_rows(mod_1_res, mod_2_res, mod_3_res)
```

```{r}
ggplot(mod_res, aes(model,value)) + geom_boxplot() + facet_wrap(~fct_inorder(key), scales = "free_y") +
  labs(title="Performance Distribution Across 100 Test Folds (10 x 10-Fold CV)", y="Value") +
  theme(axis.title.x = element_blank())
```


We can see above that model 3 shows a superior performance compared to the other 2 models across all metrics. We therefore select it as our final model to use.
In general, I would select a model that shows slightly worse performance if it makes more sense because that can be a powerful tool in convincing people to use the model and to be able to accept its predictions. I would also select a slightly worse performing model if it's more parsimonious because the model would be potentially easier to deploy and because it has less predictors, it is able to make predictions in the absence of those additional predictors which is an added advantage if some of those predictors are not always available. This is in addition to the fact that simpler models are less likely to overfit the data in general.

Typically after the model selection we would run the final model one last time on the test set to get a good estimate on the model performance on previously unseen data. We would preprocess the testing data the same way we preprocessed the training data and impute its missing values using the same models we used on the training data.

Let's run our final model on the entire moneyball dataset (after preprocessing and imputing the missing values in the dataset) and see the Adjusted R-Squared value we get as well as the model coefficients.

```{r}
moneyball_mod <- moneyball %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                      TARGET_WINS>115 ~ as.integer(115),
                                      TRUE ~ as.integer(TARGET_WINS)),

                        TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                       TEAM_BATTING_H>2000 ~ as.integer(2000),
                                       TRUE ~ as.integer(TEAM_BATTING_H)),

                        TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                        TEAM_BATTING_BB>750 ~ as.integer(750),
                                        TRUE ~ as.integer(TEAM_BATTING_BB)),

                        TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                        TRUE ~ as.integer(TEAM_BASERUN_SB)),

                        TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                        TEAM_BASERUN_CS>100 ~ as.integer(100),
                                        TRUE ~ as.integer(TEAM_BASERUN_CS)),

                        TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                         TEAM_PITCHING_BB>800 ~ as.integer(800),
                                         TRUE ~ as.integer(TEAM_PITCHING_BB)),

                        TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                         TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                         TRUE ~ as.integer(TEAM_PITCHING_SO))
                        )

moneyball_mod <- mutate(moneyball_mod,
                TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
                TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                TEAM_BATTING_3B = TEAM_BATTING_3B + 1,
                TEAM_PITCHING_HR = TEAM_PITCHING_HR + 1)
```

```{r}
recF <- recipe(TARGET_WINS ~ ., data = moneyball_mod) %>%
      step_bagimpute(TEAM_FIELDING_DP, TEAM_BATTING_SO, TEAM_BASERUN_SB,
TEAM_PITCHING_SO, TEAM_BATTING_HBP, TEAM_BASERUN_CS) %>%
      step_ratio(TEAM_BATTING_BB, denom=denom_vars(TEAM_BATTING_SO)) %>%
      step_ratio(TEAM_PITCHING_SO, denom=denom_vars(TEAM_PITCHING_BB)) %>%
      step_log(TEAM_BATTING_3B, TEAM_FIELDING_E, TEAM_PITCHING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_3B) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_SO) %>%
```

```
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_BATTING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BASERUN_SB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_PITCHING_HR:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_FIELDING_E:TEAM_FIELDING_DP) %>%
      step_interact(terms = ~ TEAM_FIELDING_DP:TEAM_PITCHING_SO_o_TEAM_PITCHING_BB) %>%
      step_rm(TEAM_FIELDING_E, TEAM_FIELDING_DP, TEAM_BATTING_3B, TEAM_PITCHING_HR,
TEAM_BATTING_H,
            TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_PITCHING_H, TEAM_BASERUN_CS,
TEAM_BATTING_2B,
            TEAM_PITCHING_BB, TEAM_PITCHING_SO, TEAM_PITCHING_SO_o_TEAM_PITCHING_BB)
```

```{r, comment=NA}
train_ctrl <- trainControl(method = "none")

lm_fit_F <- train(recF, data=moneyball_mod,
              method = "lm",
              trControl = train_ctrl)
```

```{r}
tidy(summary(lm_fit_F)) %>% rename(Term="term", Estimate="estimate", `Std. Error`="std.error", `t
value`="statistic", `Pr(>|t|)`="p.value") %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = c("striped","scale_down"))
```

```{r}
glance(summary(lm_fit_F)) %>% select(-(7:10)) %>% mutate(df=df-1, r.squared=round(r.squared,4),
adj.r.squared=round(adj.r.squared,4)) %>%
  rename(`R Squared`="r.squared", `Adj. R Squared`="adj.r.squared", `Residual Standard Error`="sigma",
                                  `F-statistic`="statistic", `p-value`="p.value", Df="df") %>%
  select(`R Squared`:`F-statistic`, Df, `p-value`) %>%
  kable("latex", booktabs=TRUE) %>% kable_styling(latex_options = "striped")
```

Next, we would do the preprocessing steps and use the imputation bagged tree models on the official (assignment) test set and then use the coefficient values above to make predictions.

Conclusion

In this project we explored the moneyball dataset and saw the value of exploration in determining appropriate transformations to variables that helped with modeling. We also imputed missing values in the dataset and saw how using more advanced imputation techniques such as bagged tree models can help in delivering superior model performance compared to simpler techniques such as mean imputation. We then tried different models and selected as our final model, the model with the lowest average Root Mean Square Error (RMSE) on 10 repeats of 10-fold cross-validation. That model included multiple predictors that were created as ratios or products of other predictors and this was shown to improve performance to levels that using only the original predictors could not reach. As part of the project, we also created a data step that takes new data, does the necessary preprocessing to it, imputes missing values, predicts the number of wins of teams and writes those predictions to file.

\newpage

Bingo Section Details (Using lm vs. glm in R)

glm

Let's train a glm model in R similar to our final model and look at the fit summary.

```{r, comment=NA}
# Comparing the results of lm and glm in R

train_ctrl <- trainControl(method = "none") # model will be fit once on the dataset given

lm_fit_glm <- train(recF, data=moneyball_mod,
                    method = "glm", # using glm this time
                    trControl = train_ctrl)

# Looking at the models summaries including the coefficients
summary(lm_fit_glm) # glm model
```

lm

Let's review our lm final model fit summary.

```{r, comment=NA}
summary(lm_fit_F) # lm model
```

Comparison

We can see above that the coefficient estimates, standard errors, t values, and p-values are identical between the glm and lm fits. However, there are differences in the metrics shown for each type. The glm fit summary shows metrics such as null deviance, residual deviance, and AIC, while lm shows metrics such as the the residual standard error, R-Squared, Adjusted R-Squared, and the F-statistic.

Code:

```
## ---------------------------------------------------------------------------
## Setup
## ---------------------------------------------------------------------------

# Please set the appropriate working environment for the script to work
# setwd("")

# Loading libraries & setting ggplot theme
suppressPackageStartupMessages(library(kableExtra))
suppressPackageStartupMessages(library(psych))
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(VIM))
suppressPackageStartupMessages(library(GGally))
suppressPackageStartupMessages(library(patchwork))
suppressPackageStartupMessages(library(corrplot))
suppressPackageStartupMessages(library(rsample))
suppressPackageStartupMessages(library(recipes))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(leaps))
suppressPackageStartupMessages(library(car))
suppressPackageStartupMessages(library(doParallel))

theme_set(theme_bw())

# Reading moneyball dataset
moneyball <- read.csv("moneyball.csv", header=T)

moneyball$INDEX <- NULL # Removing index column since we don't intend to use it

## ---------------------------------------------------------------------------
## Data Exploration
## ---------------------------------------------------------------------------

# Looking at a numeric summary of the dataset
moneyball %>% describe(skew=FALSE, quant=c(0.25,0.5,0.75)) %>% select(-8)

# Splitting data into training and testing sets
set.seed(373904)

my_split <- initial_split(moneyball, strata="TARGET_WINS")

training <- training(my_split)
testing <- testing(my_split)

# Close Exploration of TARGET_WINS
g1 <- ggplot(training, aes(TARGET_WINS)) + geom_histogram(aes(fill="#F8766D"), show.legend = FALSE) +
  labs(title="Distribution of TARGET_WINS")

g2 <- ggplot(training, aes("", TARGET_WINS)) + geom_boxplot(aes(fill="#F8766D"), show.legend = FALSE) +
  coord_flip() +
  theme(axis.title.y = element_blank(), axis.text.y = element_blank(), axis.ticks.y = element_blank())

g1/g2
```

```
# Scatterplot matrices of variables vs. outcome and each other (partially)
# Also includes histograms of the variables and Pearson correlation values
temp_for_ggpairs <- training
colnames(temp_for_ggpairs) <- str_replace_all(colnames(temp_for_ggpairs), "TEAM_","")


ggpairs(temp_for_ggpairs[,c(2:6,1)], lower = list(continuous = wrap("smooth_loess", alpha=0.1, color="blue")))


ggpairs(temp_for_ggpairs[,c(7:11,1)], lower = list(continuous = wrap("smooth_loess", alpha=0.1, color="blue")))


ggpairs(temp_for_ggpairs[,c(12:16,1)], lower = list(continuous = wrap("smooth_loess", alpha=0.1, color="blue")))

# Exploring correlations where variables are ordered with hierarchical clustering
cors <- drop_na(temp_for_ggpairs) %>% cor()
corrplot(cors, method="number", order="hclust")

# Exploring Missing Data
aggr(moneyball, cex.axis=0.45, numbers=TRUE)

marginplot(training[, c("TEAM_BASERUN_CS", "TARGET_WINS")])

marginplot(training[, c("TEAM_FIELDING_DP", "TARGET_WINS")])

# Exploring the TEAM_BATTING_3B variable before and after the log transformation
# Before log transformation:
g3 <- ggplot(training, aes(TEAM_BATTING_3B)) + geom_histogram(aes(fill="#00BFC4"), show.legend =
FALSE) +
  labs(title="Original Distribution")

g4 <- ggplot(training, aes(NULL, TEAM_BATTING_3B)) + geom_boxplot(aes(fill="#00BFC4"),
show.legend=FALSE) +
  coord_flip() +
  theme(axis.title.y = element_blank(), axis.text.y = element_blank(), axis.ticks.y = element_blank())

g5 <- ggplot(training,(aes(TEAM_BATTING_3B, TARGET_WINS))) + geom_point(alpha=0.3) +
geom_smooth(method="lm", se=FALSE)

g3/g4/g5

# After log transformation:
g6 <- ggplot(training, aes(log((TEAM_BATTING_3B+1)))) + geom_histogram(aes(fill="#00BFC4"), show.legend
= FALSE) +
  labs(title="log Transformed Distribution")

g7 <- ggplot(training, aes(NULL, log((TEAM_BATTING_3B+1)))) + geom_boxplot(aes(fill="#00BFC4"),
show.legend=FALSE) +
  coord_flip() +
  theme(axis.title.y = element_blank(), axis.text.y = element_blank(), axis.ticks.y = element_blank())

g8 <- ggplot(training,(aes(log(TEAM_BATTING_3B+1), TARGET_WINS))) + geom_point(alpha=0.3) +
geom_smooth(method="lm", se=FALSE)

g6/g7/g8
```

```
## ---------------------------------------------------------------------
## Model 1
## ---------------------------------------------------------------------

# Preprocessing training dataset for model 1
training_mod1 <- training %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                        TARGET_WINS>115 ~ as.integer(115),
                                        TRUE ~ as.integer(TARGET_WINS)),

                            TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                          TEAM_BATTING_H>2000 ~ as.integer(2000),
                                          TRUE ~ as.integer(TEAM_BATTING_H)),

                            TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                           TEAM_BATTING_BB>750 ~ as.integer(750),
                                           TRUE ~ as.integer(TEAM_BATTING_BB)),

                            TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                           TRUE ~ as.integer(TEAM_BASERUN_SB)),

                            TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                           TEAM_BASERUN_CS>100 ~ as.integer(100),
                                           TRUE ~ as.integer(TEAM_BASERUN_CS)),

                            TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                           TEAM_PITCHING_BB>800 ~ as.integer(800),
                                           TRUE ~ as.integer(TEAM_PITCHING_BB)),

                            TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                           TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                           TRUE ~ as.integer(TEAM_PITCHING_SO))
                            )

# Creating recipe for model 1
rec1 <- recipe(TARGET_WINS ~ ., data = training_mod1) %>%
      step_meanimpute(TEAM_BATTING_HBP, TEAM_BASERUN_CS, TEAM_FIELDING_DP, # mean
imputation of variables
                      TEAM_PITCHING_SO, TEAM_BATTING_SO, TEAM_BASERUN_SB) %>%
      step_rm(TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_BASERUN_CS, TEAM_BATTING_2B,
# removing predictors that wont be used
            TEAM_PITCHING_SO, TEAM_PITCHING_H, TEAM_PITCHING_BB, TEAM_BATTING_SO,
TEAM_BATTING_BB)

# Training Model 1 with Repeated Cross Validation (The summary is for the model trained on the full training data
however)
cl <- makeCluster(4) # We shall use 4 cores in parallel
registerDoParallel(cl)

set.seed(364731, kind = "L'Ecuyer-CMRG") # seed that works with parallel processing

train_ctrl <- trainControl(method = "repeatedcv", # repeated cross-validation
                     number = 10, # 10-folds
                     repeats = 10) # 10 repeats

lm_fit_1 <- train(rec1, data=training_mod1,
               method = "lm", # a linear regression model type
```

```
            trControl = train_ctrl)

stopCluster(cl) # stopping the parallel processing to continue sequentially
registerDoSEQ()

# Looking at the model summary including the coefficients
summary(lm_fit_1)

# Viewing model 1 resampling results on test folds
lm_fit_1_results <- tibble(Metric=c("RMSE","Rsquared","MAE"),
                    Mean=c(lm_fit_1$results$RMSE, lm_fit_1$results$Rsquared, lm_fit_1$results$MAE),
                    `Lower 95% C.I.`=c(lm_fit_1$results$RMSE-qnorm(0.975)*(lm_fit_1$results$RMSESD/10),
                                lm_fit_1$results$Rsquared-qnorm(0.975)*(lm_fit_1$results$RsquaredSD/10),
                                lm_fit_1$results$MAE-qnorm(0.975)*(lm_fit_1$results$MAESD/10)),
                    `Upper 95% C.I.`=c(lm_fit_1$results$RMSE+qnorm(0.975)*(lm_fit_1$results$RMSESD/10),
                                lm_fit_1$results$Rsquared+qnorm(0.975)*(lm_fit_1$results$RsquaredSD/10),
                                lm_fit_1$results$MAE+qnorm(0.975)*(lm_fit_1$results$MAESD/10)))

lm_fit_1_results

## ------------------------------------------------------------------------
## Model 2
## ------------------------------------------------------------------------

# Preprocessing training dataset for model 2
training_mod2 <- training %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                        TARGET_WINS>115 ~ as.integer(115),
                                        TRUE ~ as.integer(TARGET_WINS)),

                            TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                                TEAM_BATTING_H>2000 ~ as.integer(2000),
                                                TRUE ~ as.integer(TEAM_BATTING_H)),

                            TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                                TEAM_BATTING_BB>750 ~ as.integer(750),
                                                TRUE ~ as.integer(TEAM_BATTING_BB)),

                            TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                                TRUE ~ as.integer(TEAM_BASERUN_SB)),

                            TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                                TEAM_BASERUN_CS>100 ~ as.integer(100),
                                                TRUE ~ as.integer(TEAM_BASERUN_CS)),

                            TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                                TEAM_PITCHING_BB>800 ~ as.integer(800),
                                                TRUE ~ as.integer(TEAM_PITCHING_BB)),

                            TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                                TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                                TRUE ~ as.integer(TEAM_PITCHING_SO))
                            )

training_mod2 <- mutate(training_mod2,
                    TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                    TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
```

```
                TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                TEAM_BATTING_3B = TEAM_BATTING_3B + 1)

# Creating recipe for model 2
rec2 <- recipe(TARGET_WINS ~ ., data = training_mod2) %>%
    step_bagimpute(TEAM_BATTING_HBP, TEAM_BASERUN_CS, TEAM_FIELDING_DP, # imputing
variables with bagged tree models
                   TEAM_PITCHING_SO, TEAM_BATTING_SO, TEAM_BASERUN_SB) %>%
    step_ratio(TEAM_BATTING_BB, denom=denom_vars(TEAM_BATTING_SO)) %>% # creating predictor as
ratio between two predictors
    step_ratio(TEAM_PITCHING_SO, denom=denom_vars(TEAM_PITCHING_BB)) %>%
    step_log(TEAM_BATTING_3B, TEAM_FIELDING_E, TEAM_PITCHING_BB) %>% # log transforming 3
variables
    step_rm(TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_PITCHING_H, TEAM_BASERUN_CS, #
removing predictors that wont be used
          TEAM_BATTING_2B, TEAM_PITCHING_BB, TEAM_PITCHING_SO)

# Training Model 2 with Repeated Cross Validation (The summary is for the model trained on the full training data
however)
cl <- makeCluster(4)
registerDoParallel(cl)

train_ctrl <- trainControl(method = "repeatedcv",
                   number = 10,
                   repeats = 10)

lm_fit_2 <- train(rec2, data=training_mod2,
            method = "lm",
            trControl = train_ctrl)

stopCluster(cl)
registerDoSEQ()

# Looking at the model summary including the coefficients
summary(lm_fit_2)

# Viewing model 2 resampling results on test folds
lm_fit_2_results <- tibble(Metric=c("RMSE","Rsquared","MAE"),
                   Mean=c(lm_fit_2$results$RMSE, lm_fit_2$results$Rsquared, lm_fit_2$results$MAE),
                   `Lower 95% C.I.`=c(lm_fit_2$results$RMSE-qnorm(0.975)*(lm_fit_2$results$RMSESD/10),
                              lm_fit_2$results$Rsquared-qnorm(0.975)*(lm_fit_2$results$RsquaredSD/10),
                              lm_fit_2$results$MAE-qnorm(0.975)*(lm_fit_2$results$MAESD/10)),
                   `Upper 95% C.I.`=c(lm_fit_2$results$RMSE+qnorm(0.975)*(lm_fit_2$results$RMSESD/10),
                              lm_fit_2$results$Rsquared+qnorm(0.975)*(lm_fit_2$results$RsquaredSD/10),
                              lm_fit_2$results$MAE+qnorm(0.975)*(lm_fit_2$results$MAESD/10)))

lm_fit_2_results

## ------------------------------------------------------------------------
## Model 3
## ------------------------------------------------------------------------

# Preprocessing training dataset for model 3
training_mod3 <- training %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                     TARGET_WINS>115 ~ as.integer(115),
```

```
                                TRUE ~ as.integer(TARGET_WINS)),

                        TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                                   TEAM_BATTING_H>2000 ~ as.integer(2000),
                                                   TRUE ~ as.integer(TEAM_BATTING_H)),

                        TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                                    TEAM_BATTING_BB>750 ~ as.integer(750),
                                                    TRUE ~ as.integer(TEAM_BATTING_BB)),

                        TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                                    TRUE ~ as.integer(TEAM_BASERUN_SB)),

                        TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                                    TEAM_BASERUN_CS>100 ~ as.integer(100),
                                                    TRUE ~ as.integer(TEAM_BASERUN_CS)),

                        TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                                     TEAM_PITCHING_BB>800 ~ as.integer(800),
                                                     TRUE ~ as.integer(TEAM_PITCHING_BB)),

                        TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                                     TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                                     TRUE ~ as.integer(TEAM_PITCHING_SO))
                        )

training_mod3 <- mutate(training_mod3,
                        TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                        TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
                        TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                        TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                        TEAM_BATTING_3B = TEAM_BATTING_3B + 1,
                        TEAM_PITCHING_HR = TEAM_PITCHING_HR + 1)

# Creating recipe for model 3
rec3 <- recipe(TARGET_WINS ~ ., data = training_mod3) %>%
      step_bagimpute(TEAM_FIELDING_DP, TEAM_BATTING_SO, TEAM_BASERUN_SB, # imputing
variables with bagged tree models
                     TEAM_PITCHING_SO, TEAM_BATTING_HBP, TEAM_BASERUN_CS) %>%
      step_ratio(TEAM_BATTING_BB, denom=denom_vars(TEAM_BATTING_SO)) %>% # creating predictor as
ratio between two predictors
      step_ratio(TEAM_PITCHING_SO, denom=denom_vars(TEAM_PITCHING_BB)) %>%
      step_log(TEAM_BATTING_3B, TEAM_FIELDING_E, TEAM_PITCHING_BB) %>% # log transforming 3
variables
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_3B) %>% # creating predictor as a product
between two predictors
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_SO) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_BATTING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_FIELDING_E) %>%
```

```
      step_interact(terms = ~ TEAM_BASERUN_SB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_PITCHING_HR:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_FIELDING_E:TEAM_FIELDING_DP) %>%
      step_interact(terms = ~ TEAM_FIELDING_DP:TEAM_PITCHING_SO_o_TEAM_PITCHING_BB) %>%
      step_rm(TEAM_FIELDING_E, TEAM_FIELDING_DP, TEAM_BATTING_3B, TEAM_PITCHING_HR,
TEAM_BATTING_H, # removing predictors that wont be used
          TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_PITCHING_H, TEAM_BASERUN_CS,
TEAM_BATTING_2B,
          TEAM_PITCHING_BB, TEAM_PITCHING_SO, TEAM_PITCHING_SO_o_TEAM_PITCHING_BB)

# Training Model 3 with Repeated Cross Validation (The summary is for the model trained on the full training data
however)
cl <- makeCluster(4)
registerDoParallel(cl)

train_ctrl <- trainControl(method = "repeatedcv",
                   number = 10,
                   repeats = 10)

lm_fit_3 <- train(rec3, data=training_mod3,
             method = "lm",
             trControl = train_ctrl)

stopCluster(cl)
registerDoSEQ()

# Looking at the model summary including the coefficients
summary(lm_fit_3)

# Viewing model 3 resampling results on test folds
lm_fit_3_results <- tibble(Metric=c("RMSE","Rsquared","MAE"),
                   Mean=c(lm_fit_3$results$RMSE, lm_fit_3$results$Rsquared, lm_fit_3$results$MAE),
                   `Lower 95% C.I.`=c(lm_fit_3$results$RMSE-qnorm(0.975)*(lm_fit_3$results$RMSESD/10),
                              lm_fit_3$results$Rsquared-qnorm(0.975)*(lm_fit_3$results$RsquaredSD/10),
                              lm_fit_3$results$MAE-qnorm(0.975)*(lm_fit_3$results$MAESD/10)),
                   `Upper 95% C.I.`=c(lm_fit_3$results$RMSE+qnorm(0.975)*(lm_fit_3$results$RMSESD/10),
                              lm_fit_3$results$Rsquared+qnorm(0.975)*(lm_fit_3$results$RsquaredSD/10),
                              lm_fit_3$results$MAE+qnorm(0.975)*(lm_fit_3$results$MAESD/10)))

lm_fit_3_results

## ---------------------------------------------------------------------
## Evaluating performance of our 3 models
## ---------------------------------------------------------------------

mod_1_res <- gather(select(lm_fit_1$resample, -Resample)) %>% mutate(model="Model 1")

mod_2_res <- gather(select(lm_fit_2$resample, -Resample)) %>% mutate(model="Model 2")

mod_3_res <- gather(select(lm_fit_3$resample, -Resample)) %>% mutate(model="Model 3")

mod_res <- bind_rows(mod_1_res, mod_2_res, mod_3_res)

ggplot(mod_res, aes(model,value)) + geom_boxplot() + facet_wrap(~fct_inorder(key), scales = "free_y") +
  labs(title="Performance Distribution Across 100 Test Folds (10 x 10-Fold CV)", x="Model", y="Value")
```

```
## ---------------------------------------------------------------------
## Using our final recipe and model on the entire Moneyball dataset
## ---------------------------------------------------------------------

# Preprocessing moneyball dataset for final model (model 3 was selected)
moneyball_mod <- moneyball %>% mutate(TARGET_WINS = case_when(TARGET_WINS<40 ~ as.integer(40),
                                      TARGET_WINS>115 ~ as.integer(115),
                                      TRUE ~ as.integer(TARGET_WINS)),

                      TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                       TEAM_BATTING_H>2000 ~ as.integer(2000),
                                       TRUE ~ as.integer(TEAM_BATTING_H)),

                      TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                        TEAM_BATTING_BB>750 ~ as.integer(750),
                                        TRUE ~ as.integer(TEAM_BATTING_BB)),

                      TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                        TRUE ~ as.integer(TEAM_BASERUN_SB)),

                      TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                        TEAM_BASERUN_CS>100 ~ as.integer(100),
                                        TRUE ~ as.integer(TEAM_BASERUN_CS)),

                      TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                         TEAM_PITCHING_BB>800 ~ as.integer(800),
                                         TRUE ~ as.integer(TEAM_PITCHING_BB)),

                      TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                                         TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                                         TRUE ~ as.integer(TEAM_PITCHING_SO))
                      )

moneyball_mod <- mutate(moneyball_mod,
                  TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                  TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
                  TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                  TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                  TEAM_BATTING_3B = TEAM_BATTING_3B + 1,
                  TEAM_PITCHING_HR = TEAM_PITCHING_HR + 1)

# Creating the final model recipe
recF <- recipe(TARGET_WINS ~ ., data = moneyball_mod) %>%
      step_bagimpute(TEAM_FIELDING_DP, TEAM_BATTING_SO, TEAM_BASERUN_SB, # imputing
variables with bagged tree models
                     TEAM_PITCHING_SO, TEAM_BATTING_HBP, TEAM_BASERUN_CS) %>%
      step_ratio(TEAM_BATTING_BB, denom=denom_vars(TEAM_BATTING_SO)) %>% # creating predictor as
ratio between two predictors
      step_ratio(TEAM_PITCHING_SO, denom=denom_vars(TEAM_PITCHING_BB)) %>%
      step_log(TEAM_BATTING_3B, TEAM_FIELDING_E, TEAM_PITCHING_BB) %>% # log transforming 3
variables
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_3B) %>% # creating predictor as a product
between two predictors
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_BATTING_SO) %>%
      step_interact(terms = ~ TEAM_BATTING_H:TEAM_FIELDING_E) %>%
```

```
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_BATTING_BB) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_3B:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_BB:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_BATTING_SO:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_BASERUN_SB:TEAM_PITCHING_HR) %>%
      step_interact(terms = ~ TEAM_PITCHING_HR:TEAM_FIELDING_E) %>%
      step_interact(terms = ~ TEAM_FIELDING_E:TEAM_FIELDING_DP) %>%
      step_interact(terms = ~ TEAM_FIELDING_DP:TEAM_PITCHING_SO_o_TEAM_PITCHING_BB) %>%
      step_rm(TEAM_FIELDING_E, TEAM_FIELDING_DP, TEAM_BATTING_3B, TEAM_PITCHING_HR,
TEAM_BATTING_H, # removing predictors that wont be used
            TEAM_BATTING_HR, TEAM_BATTING_HBP, TEAM_PITCHING_H, TEAM_BASERUN_CS,
TEAM_BATTING_2B,
            TEAM_PITCHING_BB, TEAM_PITCHING_SO, TEAM_PITCHING_SO_o_TEAM_PITCHING_BB)

# Creating the model
train_ctrl <- trainControl(method = "none")

lm_fit_F <- train(recF, data=moneyball_mod,
              method = "lm",
              trControl = train_ctrl)

# Looking at the model summary including the coefficients
summary(lm_fit_F)

## ----------------------------------------------------------------------
## Official Test Data Step
## ----------------------------------------------------------------------

# Set appropriate working environment if it's different from the environment used above
# setwd("")

# Reading testing data
actual_testing_set <- read.csv("moneyball_test.csv",header=T)

# Truncating and modifying testing data (pre-processing)
actual_testing_set_trunc <- actual_testing_set %>%
  mutate(TEAM_BATTING_H = case_when(TEAM_BATTING_H<1100 ~ as.integer(1100),
                                    TEAM_BATTING_H>2000 ~ as.integer(2000),
                                    TRUE ~ as.integer(TEAM_BATTING_H)),

         TEAM_BATTING_BB = case_when(TEAM_BATTING_BB<250 ~ as.integer(250),
                                     TEAM_BATTING_BB>750 ~ as.integer(750),
                                     TRUE ~ as.integer(TEAM_BATTING_BB)),

         TEAM_BASERUN_SB = case_when(TEAM_BASERUN_SB<18 ~ as.integer(18),
                                     TRUE ~ as.integer(TEAM_BASERUN_SB)),

         TEAM_BASERUN_CS = case_when(TEAM_BASERUN_CS<11 ~ as.integer(11),
                                     TEAM_BASERUN_CS>100 ~ as.integer(100),
                                     TRUE ~ as.integer(TEAM_BASERUN_CS)),

         TEAM_PITCHING_BB = case_when(TEAM_PITCHING_BB<300 ~ as.integer(300),
                                      TEAM_PITCHING_BB>800 ~ as.integer(800),
```

```
                        TRUE ~ as.integer(TEAM_PITCHING_BB)),

      TEAM_PITCHING_SO = case_when(TEAM_PITCHING_SO<100 ~ as.integer(100),
                        TEAM_PITCHING_SO>1750 ~ as.integer(1750),
                        TRUE ~ as.integer(TEAM_PITCHING_SO)))

actual_testing_set_trunc <- mutate(actual_testing_set_trunc,
                        TEAM_BATTING_BB = TEAM_BATTING_BB + 1,
                        TEAM_BATTING_SO = TEAM_BATTING_SO + 1,
                        TEAM_PITCHING_SO = TEAM_PITCHING_SO + 1,
                        TEAM_PITCHING_BB = TEAM_PITCHING_BB + 1,
                        TEAM_BATTING_3B = TEAM_BATTING_3B + 1,
                        TEAM_PITCHING_HR = TEAM_PITCHING_HR + 1)

# Preparing testing data set by imputing missing values, log transforming some variables
# and creating new variables in the same way we did to our moneyball data.
# To do that we prepare (prep) the recipe on the moneyball data and then bake the recipe on
# the testing data.
prepared_recipe <- prep(recF, training = moneyball_mod, retain = FALSE, verbose = FALSE)
actual_testing_set_trunc_baked <- bake(prepared_recipe, newdata = actual_testing_set_trunc)

# Stand Alone Scoring
actual_testing_set_trunc_baked <- mutate(actual_testing_set_trunc_baked,
                         INDEX = actual_testing_set$INDEX,
                         P_TARGET_WINS = 4.376e+01 +
                         3.569e-01 * TEAM_BATTING_BB +
                        -1.149e-01 * TEAM_BATTING_SO +
                         1.030e-01 * TEAM_BASERUN_SB +
                         9.645e-02 * TEAM_BATTING_BB_o_TEAM_BATTING_SO +
                         1.824e-02 * TEAM_BATTING_H_x_TEAM_BATTING_3B +
                        -1.513e-04 * TEAM_BATTING_H_x_TEAM_BATTING_BB +
                         1.941e-05 * TEAM_BATTING_H_x_TEAM_BATTING_SO +
                         4.267e-03 * TEAM_BATTING_H_x_TEAM_FIELDING_E +
                         1.929e-02 * TEAM_BATTING_3B_x_TEAM_BATTING_BB +
                        -6.152e-02 * TEAM_BATTING_3B_x_TEAM_PITCHING_HR +
                        -4.534e+00 * TEAM_BATTING_3B_x_TEAM_FIELDING_E +
                         2.702e-04 * TEAM_BATTING_BB_x_TEAM_PITCHING_HR +
                        -3.865e-02 * TEAM_BATTING_BB_x_TEAM_FIELDING_E +
                        -9.880e-05 * TEAM_BATTING_SO_x_TEAM_PITCHING_HR +
                         1.384e-02 * TEAM_BATTING_SO_x_TEAM_FIELDING_E +
                        -4.025e-04 * TEAM_BASERUN_SB_x_TEAM_PITCHING_HR +
                         5.334e-02 * TEAM_PITCHING_HR_x_TEAM_FIELDING_E +
                        -3.609e-02 * TEAM_FIELDING_E_x_TEAM_FIELDING_DP +
                         3.164e-02 *
TEAM_FIELDING_DP_x_TEAM_PITCHING_SO_o_TEAM_PITCHING_BB)


# Post-processing of prediction
actual_testing_set_trunc_baked <- mutate(actual_testing_set_trunc_baked,
                         P_TARGET_WINS = case_when(P_TARGET_WINS < 40 ~ 40,
                                                  P_TARGET_WINS > 115 ~ 115,
                                                  TRUE ~ P_TARGET_WINS))

# Writing index and predictions to a file
final_to_write <- select(actual_testing_set_trunc_baked, INDEX, P_TARGET_WINS)
write.csv(final_to_write, "Just_Created_TEST.csv", row.names = FALSE)
```

```
## ------------------------------------------------------------------------

# Checking mean and median of post-processed predictions
mean(actual_testing_set_trunc_baked$P_TARGET_WINS) # mean = 80.38182
median(actual_testing_set_trunc_baked$P_TARGET_WINS) # median = 80.30146

## ------------------------------------------------------------------------
## Bingo Section
## ------------------------------------------------------------------------

# Comparing the results of lm and glm in R

train_ctrl <- trainControl(method = "none") # model will be fit once on the dataset given

lm_fit_glm <- train(recF, data=moneyball_mod,
                    method = "glm", # using glm this time
                    trControl = train_ctrl)

# Looking at the models summaries including the coefficients
summary(lm_fit_glm) # glm model

summary(lm_fit_F) # lm model
```