Analyzing Baseball Statistics Across Cultures: A Study of the KBO

Junho Eum

Data collected from https://www.kaggle.com/datasets/mattop/korean-baseball-pitching-data-1982-2021 https://www.kaggle.com/datasets/mattop/baseball-kbo-batting-data-1982-2021 Description of the dataset can be found at the end of the document or on the link.

Analyzing Baseball Statistics Across Cultures: A Study of the KBO

The Korean Baseball League (KBO) did start in 1982 and was founded relatively late compared to other professional baseball leagues such as Nippon Professional Baseball (NPB) in Japan and Major League Baseball (MLB) in the United States, which were established in 1936 and 1876, respectively. As a result, KBO has been influenced by both the Japanese and American baseball cultures and has developed its unique playing style. The tactical and agile approach to running the bases, which is emphasized in the Japanese league, has combined with the power-hitting and home run-focused approach of the American league to create a distinct style of play in the KBO.

I am introducing the statistic used in Major League Baseball, which is now considered a standard matrix to compute the expected win rate. This statistic was made by Bill James and is called Pythagorean Expectation. He stated that the Expected win rate could be computed by dividing the sum of the squared values of runs scored and runs allowed from the squared value of runs scored. The equation looks like this.

Pythagorean Expectation=(RS$^{2)/(RS}$2+RAˆ2)

I wanted to test if this equation also works for teams in the KBO league.

Since there is a considerable difference in play styles in each league, I wanted to find the best model to predict the win_loss_percentage and then compare the accuracy with the Pythagorean Expectation model which will be built using multi polynomial regression.

Two datasets, containing pitching and batting data, were obtained from Kaggle. To model the Pythagorean expectation, the variables "runs.x" and "runs.y" were used to represent runs allowed and runs scored, respectively. The datasets were concatenated based on matching columns. The resulting equation was used to calculate the dependent variable, "win_loss_probability". Different models were tested and compared based on their error rates in predicting this variable.

Data Visualization

First, after importing two datasets, I checked for null values in each column. It shows that some data points from the past are missing because the stat recording system was not implemented yet. Therefore I removed the duplicates and dropped rows with null values.

Next, I generated a correlation heatmap to see which variable affects the win-loss percentage most.

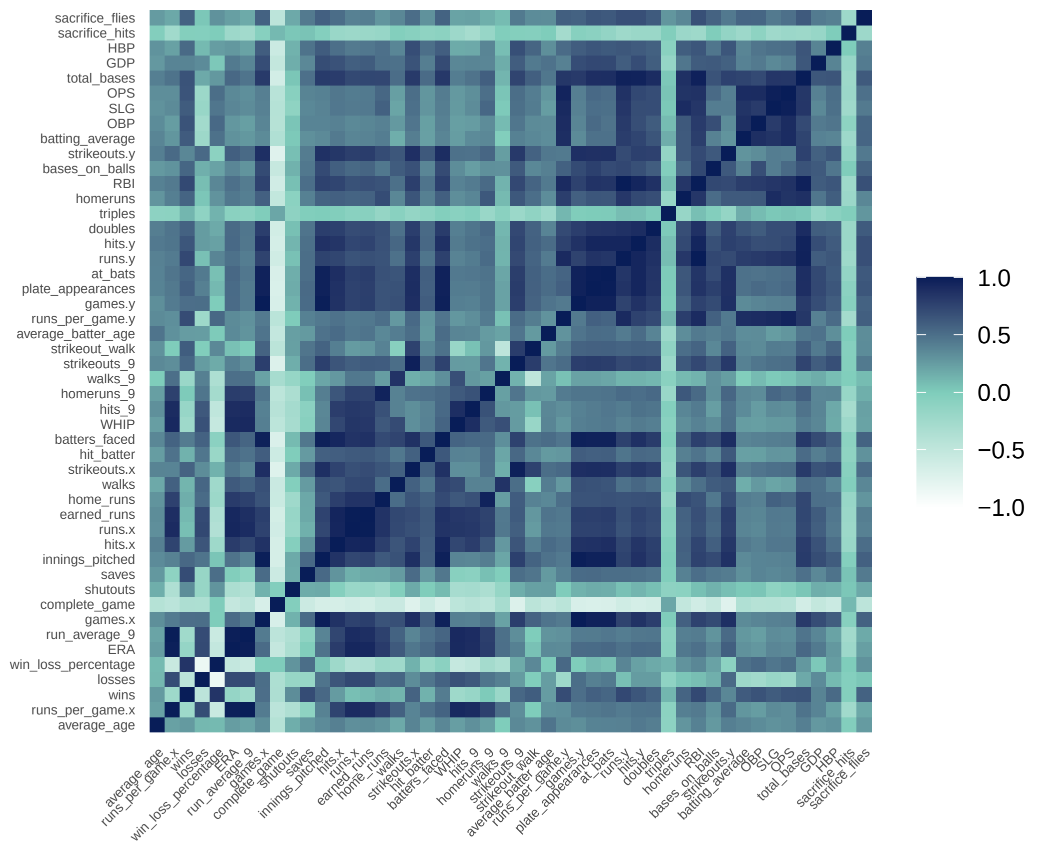
```r
df1=read.csv("kbopitchingdata.csv")
df2 = read.csv("kbobattingdata.csv")
train_df <- data.frame()
train_df = merge(df1, df2, by = c("team","year"))
train_df <- distinct(train_df)
train_df <- train_df[, colSums(is.na(train_df)) == 0]
```

```
library(ggcorrplot)
```

```
## Loading required package: ggplot2
```

```
corr=cor(train_df[,4:ncol(train_df)], use = "pairwise.complete.obs")

ggplot(data = reshape2::melt(corr)) +
  geom_tile(aes(Var1, Var2, fill = value)) +
  scale_fill_gradient2(low = "#ffffff", mid = "#7fcdbb", high = "#081d58", midpoint = 0, space = "Lab",
                       na.value = "grey50", guide = "colourbar", limits = c(-1,1), name = "") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1,size=5), axis.text.y = element_text(angle = 0,
        panel.grid.major = element_blank(), panel.border = element_blank(), panel.background = element_
        axis.line = element_blank(), axis.title = element_blank(), axis.ticks = element_blank()) +
  coord_fixed()
```



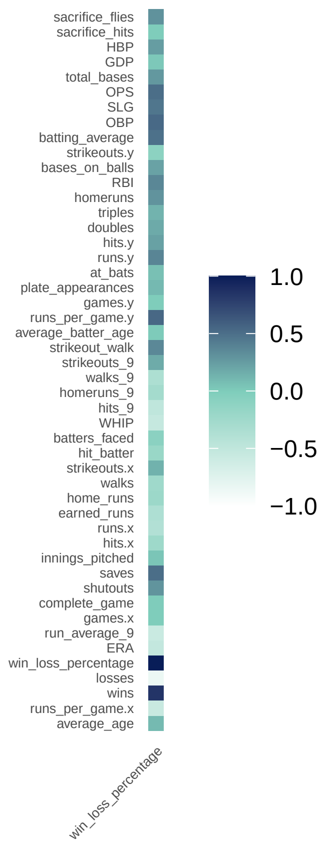
After observing the complete pairwise correlation heatmap, I specified the heatmap to a correlation with win-loss percentage vs. other variables. The plot shows that wins, saves,strike_walk, shutouts, and saves have the highest correlation absolute value with the win_loss_percentage. Pythagorean Expectation uses only runs_scores and runs_allowed parameters to predict the win_loss_percentage. So I will use these variables except 'win' because it has high collinearity with the win_loss percentage for multiple logistic regression to build a model to predict win_loss_percentage to see the difference in accuracy between these two models.

```
corr=cor(train_df["win_loss_percentage"],train_df[,4:ncol(train_df)], use = "pairwise.complete.obs")

top_5 <- sort(abs(corr), decreasing = TRUE)[2:6]
top_5
```

```
## [1] 0.8486702 0.8472265 0.5650472 0.5543284 0.5404793
```

```
ggplot(data = reshape2::melt(corr)) +
  geom_tile(aes(Var1, Var2, fill = value)) +
  scale_fill_gradient2(low = "#ffffff", mid = "#7fcdbb", high = "#081d58", midpoint = 0, space = "Lab",
                       na.value = "grey50", guide = "colourbar", limits = c(-1,1), name = "") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1,size=5), axis.text.y = element_text(angle = 0,
        panel.grid.major = element_blank(), panel.border = element_blank(), panel.background = element_
        axis.line = element_blank(), axis.title = element_blank(), axis.ticks = element_blank()) +
  coord_fixed()
```



```
df1=read.csv("kbopitchingdata.csv")
df2 = read.csv("kbobattingdata.csv")

merged_df = merge(df1, df2, by = c("team","year"))
merged_df <- distinct(merged_df)
merged_df <- merged_df[, colSums(is.na(merged_df)) == 0]
```

Logistic regression with "age" as parameter

In order to investigate the impact of team age on win-loss percentage in baseball, a quartile range was used to categorize team age. Specifically, teams were classified as either age <= 25.7, 25.7 < age <= 26.9, or age > 28, and dummy variables were created accordingly, with the age0 group serving as the reference. A linear model was then constructed with win_loss_percentage as the response variable and age1 and age2 as the predictors.

An ANOVA analysis was performed on the model, revealing that age2 (i.e., age > 28) had a statistically significant effect on win_loss_percentage at an alpha level of 0.01.

These findings suggest that, despite baseball being a sport that places less emphasis on cardiovascular fitness than other sports, team age still plays a significant role in determining win-loss percentage.

```
avg_age = mean(merged_df$average_age)
quantile(merged_df$average_age, probs = c(0.25, 0.5, 0.75))
```

```
##  25%  50%  75%
## 25.7 26.9 28.0
```

```
merged_df$age0 <- ifelse(merged_df$average_age<=25.7, 1, 0)
merged_df$age1 <- ifelse(merged_df$average_age>25.7 && merged_df$average_age<=26.9, 1, 0)
```

```
## Warning in merged_df$average_age > 25.7 && merged_df$average_age <= 26.9:
## 'length(x) = 323 > 1' in coercion to 'logical(1)'

## Warning in merged_df$average_age > 25.7 && merged_df$average_age <= 26.9:
## 'length(x) = 323 > 1' in coercion to 'logical(1)'
```

```
merged_df$age2 <-ifelse(merged_df$average_age>28, 1, 0)

model <- lm(win_loss_percentage ~ age1 + age2, data=merged_df)
summary(model)
```

```
##
## Call:
## lm(formula = win_loss_percentage ~ age1 + age2, data = merged_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.30656 -0.05863  0.00144  0.06544  0.20544
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.494560   0.005561  88.933   <2e-16 ***
## age1              NA         NA      NA       NA
## age2        0.022140   0.011174   1.981   0.0484 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08669 on 321 degrees of freedom
## Multiple R-squared:  0.01208,    Adjusted R-squared:  0.009005
## F-statistic: 3.926 on 1 and 321 DF,  p-value: 0.0484
```

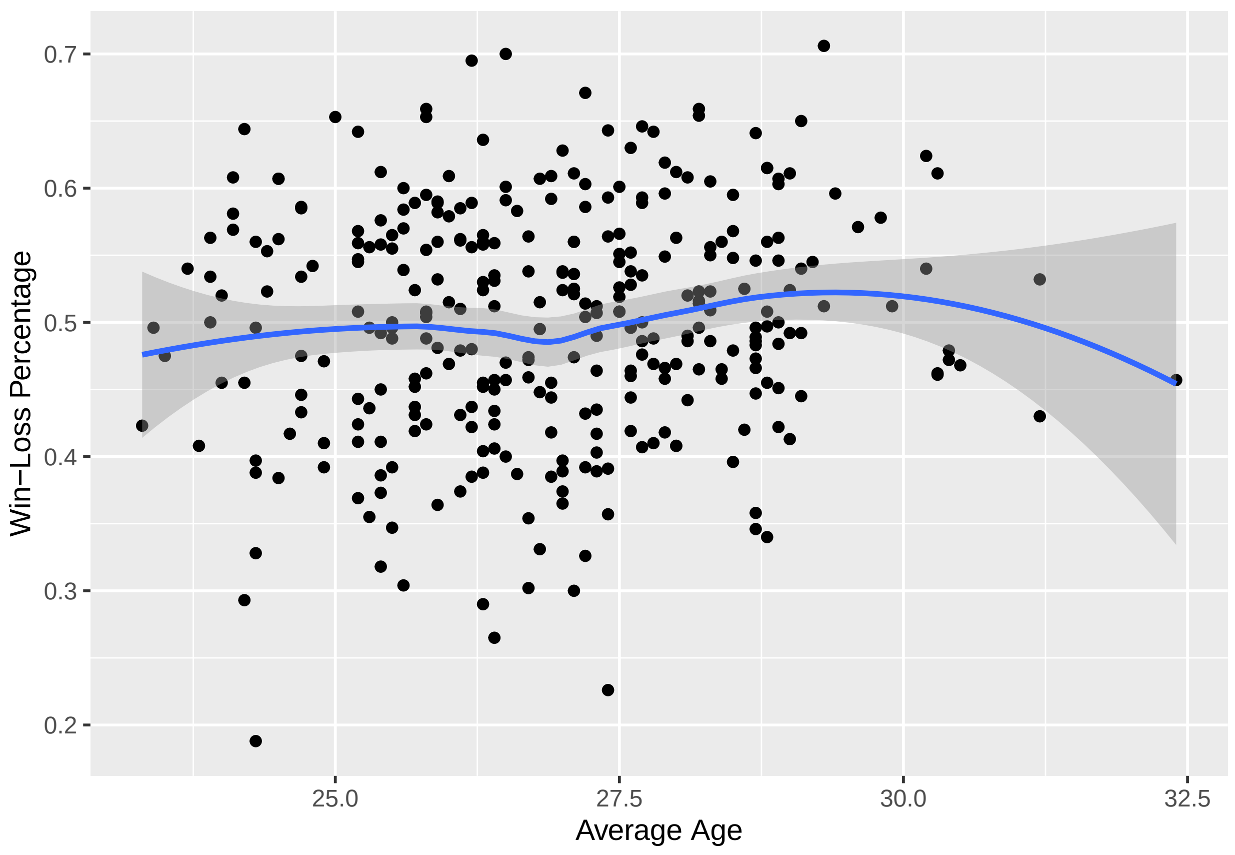
```
anova(model)
```

```
## Analysis of Variance Table
##
## Response: win_loss_percentage
##            Df Sum Sq   Mean Sq F value Pr(>F)
## age2        1 0.0295 0.0295027  3.9259 0.0484 *
## Residuals 321 2.4123 0.0075149
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Plotting the trend between win loss percentage vs average age.

```
ggplot(data = train_df, aes(x = average_age, y = win_loss_percentage)) +
  geom_point() +
  geom_smooth(method = "loess", se = TRUE)+
  labs(x = "Average Age", y = "Win-Loss Percentage")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Multinomial Logistic Regression Model

To build a multinomial logistic regression classification model using saves,strike_walk, and shutouts as parameters, I built multinomial logistic regression model using to compare it with the Pythagorean Expectation

```
df1=read.csv("kbopitchingdata.csv")
df2 = read.csv("kbobattingdata.csv")
# Merging another dataframe containing batter data
merged_df = merge(df1, df2, by = c("team","year"))
merged_df <- distinct(merged_df)
merged_df <- merged_df[, colSums(is.na(merged_df)) == 0]
```

Then, divide the win-loss percentage into three levels <0.45(bad), <0.55 (standard), >0.63 (good), and set the normal state as a reference variable. Then I factored in the dependent variable and built a multinomial logistic regression model focusing on standard errors from the summary of the model. The standard error of average age is lower than the other variables, so the average age is a relatively accurate model parameter. Moreover, the confusion matrix for the model using predicted values from the model gives only 0.54 accuracy, so this model is not a good fit for the data. Lastly, I conducted a z-test to check the significance of each parameter for this model.

```
library(nnet)
#factor the win_loss_percentage into 3 levels: <0.45:0 (bad), <0.55 (normal), >0.63 (good)
# Creating function to generate the levels
create_level <- function(x) {
  ifelse(x < 0.45, "bad", ifelse(x < 0.55, "normal", "good"))
}

train_df <- train_df %>%
  mutate(response = create_level(win_loss_percentage))
# Create new variable for factors
train_df$responseF=factor(train_df$response)

# Identify a reference level
# normal as a reference level
train_df$out = relevel(train_df$responseF, ref="normal")
# Build multinomial logistic regression model

my_model = multinom(out~shutouts+saves+strikeout_walk,data=train_df)
```

```
## # weights:  15 (8 variable)
## initial  value 354.851769
## iter  10 value 298.120249
## final  value 298.073269
## converged
```

```
summary(my_model)
```

```
## Call:
## multinom(formula = out ~ shutouts + saves + strikeout_walk, data = train_df)
##
## Coefficients:
##      (Intercept)    shutouts       saves strikeout_walk
## bad     3.059325 -0.12910232 -0.06419364     -0.7181508
## good   -4.049828  0.06999978  0.08319118      0.3710075
##
## Std. Errors:
##      (Intercept)   shutouts      saves strikeout_walk
## bad    0.7326335 0.05014728 0.01879015      0.4488187
## good   0.8410989 0.04076095 0.01979807      0.3972874
##
## Residual Deviance: 596.1465
## AIC: 612.1465
```

```
#Predicting classes
head(predict(my_model,train_df))
```

```
## [1] bad    bad    normal normal normal normal
## Levels: normal bad good
```

```
#Predicting probabilities
head(predict(my_model,train_df,type='prob'))
```

```
##      normal       bad       good
## 1 0.3023282 0.6775501 0.02012168
## 2 0.2824187 0.6958618 0.02171949
## 3 0.5003434 0.2438177 0.25583894
## 4 0.4705695 0.3574923 0.17193813
## 5 0.4156154 0.2297743 0.35461034
## 6 0.4773236 0.3165975 0.20607899
```

```
# Misclassification error
cm = table(predict(my_model,train_df),train_df$out)
cm
```

```
##
##          normal bad good
##   normal     82  46   46
##   bad        25  38    4
##   good       29   1   52
```

```
#Calculating misclassifcation percentage
1-sum(diag(cm))/sum(cm)
```

```
## [1] 0.4674923
```

```
#two-tailed z test
z=summary(my_model)$coefficients/summary(my_model)$standard.errors
p=(1-pnorm(abs(z),0,1))*2
1-p
```

```
##      (Intercept)  shutouts     saves strikeout_walk
## bad    0.9999703 0.9899604 0.9993653      0.8904216
## good   0.9999985 0.9140801 0.9999735      0.6496196
```

Classification model did not perform well. I will try regression modeling 2. Split the dataset into 0.3/0.7

```
set.seed(123)
train_indices <- sample(1:nrow(merged_df), round(0.7 * nrow(merged_df)), replace = FALSE)
train_data <- merged_df[train_indices, ]
test_data <- merged_df[-train_indices, ]
```

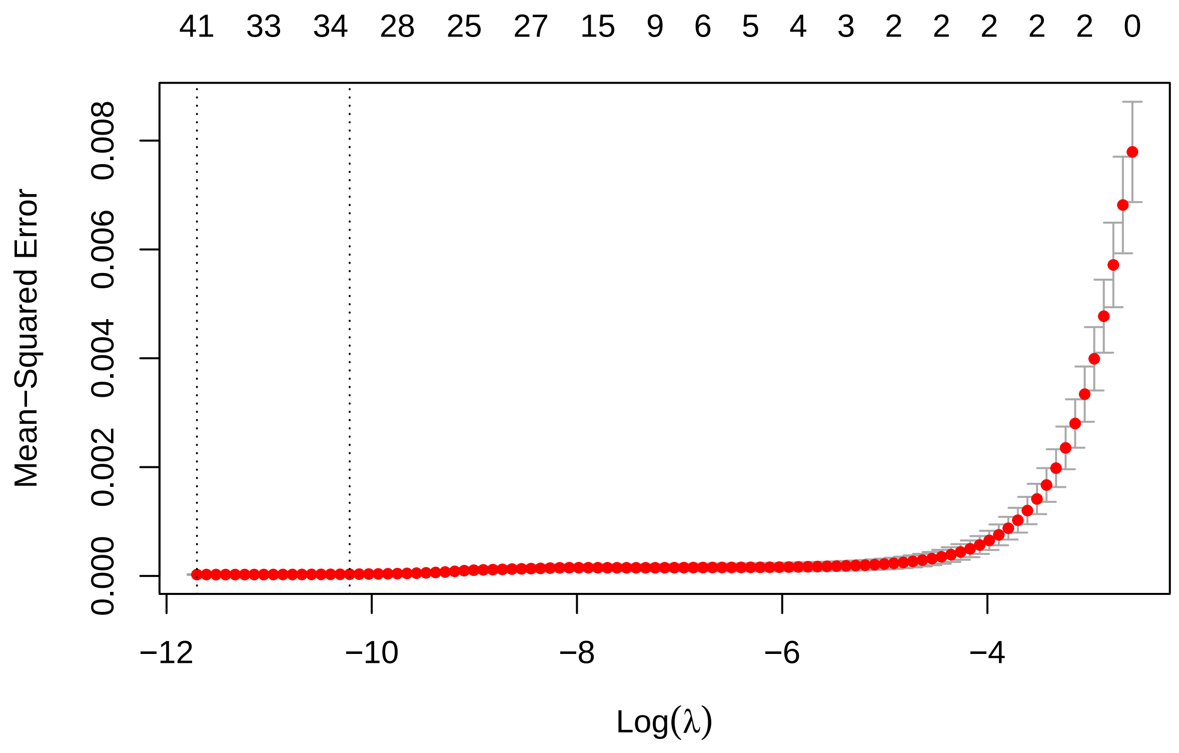
Lasso Regression for variable selection

In Lasso regression, the coefficients of the variables are shrunk toward zero. The larger the variable's coefficient, the more influential the variable is for predicting the outcome variable. I picked the top4 variables that have the highest coefficients using lasso regression.

After selecting all numerical variables, Regression was performed with 10-fold cross-validation.

Then, I plotted the regression model, which shows the mean cross-validated error as a function of the log of lambda. I adjusted the lambda value to the minimum lambda value that the model gives. Then performed Lasso regression again with the optimal lambda value.

Variables with the highest coefficient were batting_average WHIP OBP runs_per_game.x.



```
## The optimal lambda is 8.261899e-06
```

```
## The selected variables are batting_average WHIP OBP runs_per_game.x
```

Ridge Regression Modeling

Now I will use the ridge regression to shrink the coefficients of the top 4 variables selected from lasso regression.

For the model parameter, I added runs_scored and runs_allowed to compare two models later.

The ridge regression model resulted in mse=0.0024, r_squared =0.79.

The mean squared error (MSE) value of 0.0024 means that, on average, the predicted win_loss_percentage of the ridge regression model differs from the actual win_loss_percentage by 0.0024 units.

The R-squared value of 0.79 indicates that the ridge regression model explains about 79% of the variation in the outcome variable.

The results suggest that the ridge regression model is a good predictor for the KBO dataset.

```r
# Ridge regression
library(glmnet)
# Prepare the data for modeling
x <- as.matrix(train_data[,c("batting_average", "WHIP", "OBP", "runs_per_game.x")])
y <- train_data$win_loss_percentage

# Fit the ridge regression model using glmnet
ridge_model <- glmnet(x, y, alpha = 0, lambda = 0.1)

# Predict using the ridge regression model on the test data
x_test <- as.matrix(test_data[,c("batting_average", "WHIP", "OBP", "runs_per_game.x")])
y_pred <- predict(ridge_model, newx = x_test)

# Evaluate the model's performance on the test data
mse <- mean((y_pred - test_data$win_loss_percentage)^2)
rsq <- cor(y_pred, test_data$win_loss_percentage)^2

mse
```

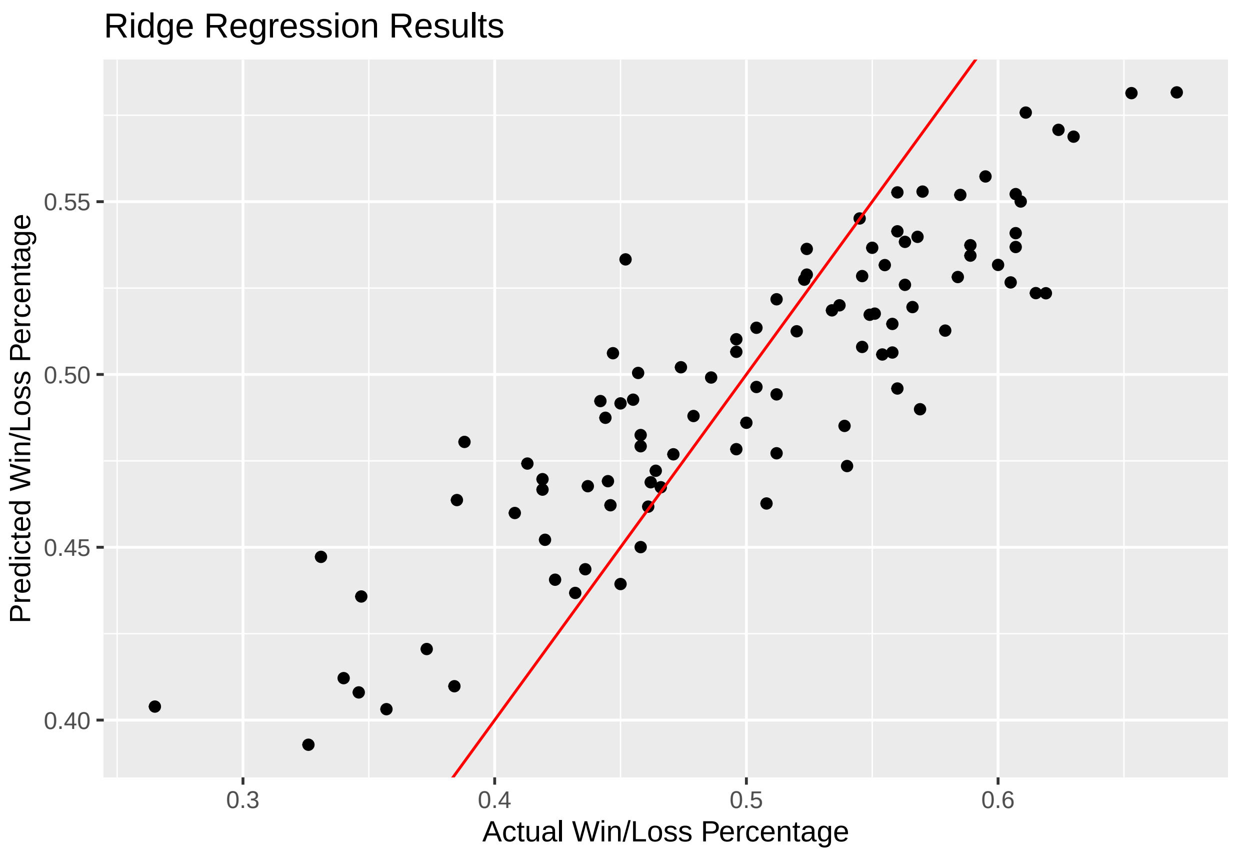
```
## [1] 0.002384898
```

```r
rsq
```

```
##         [,1]
## s0 0.7939391
```

```r
results <- data.frame(actual = test_data$win_loss_percentage, predicted = y_pred)

# Create a scatterplot of the results using ggplot2
ggplot(data = results, aes(x = actual, y = s0)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  xlab("Actual Win/Loss Percentage") +
  ylab("Predicted Win/Loss Percentage") +
  ggtitle("Ridge Regression Results")
```



Pythagorean Expectation Modeling

Next, I build a multi-polynomial regression model using the Pythagorean Expectation. The summary of the model gives an adjusted R squared value of 0.68, which is lower than the R squared value from the ridge regression, and it means that the ridge regression model fits the data better. Moreover, the model gave a mean squared error of 0.0023, which is almost the same as the mse from the ridge regression model. Two models gave similar accuracy, but since ridge regression fits the data better, I can conclude that my model built from lasso and ridge regression is better than the quadratic model built from the Pythagorean Expectation.

```
Pyth_exp = win_loss_percentage~ poly(runs.y,2)/(poly(runs.x,2)+poly(runs.y,2))
pyth_model = lm(Pyth_exp,data=train_data)

predicted_y <- predict(pyth_model, newdata = test_data)
# Calculate the MSE between predicted and actual response values
mse <- mean((test_data$win_loss_percentage - predicted_y)^2)
mse
```

```
## [1] 0.002281025
```

```
summary(pyth_model)
```

```
##
```

```
## Call:
## lm(formula = Pyth_exp, data = train_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.275770 -0.030335 -0.004779  0.026571  0.204946
##
## Coefficients:
##                                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        0.475880   0.004908  96.953  < 2e-16 ***
## poly(runs.y, 2)1                   0.230374   0.066229   3.478 0.000608 ***
## poly(runs.y, 2)2                  -0.590767   0.077939  -7.580 9.68e-13 ***
## poly(runs.y, 2)1:poly(runs.x, 2)1   8.023558   1.388335   5.779 2.56e-08 ***
## poly(runs.y, 2)2:poly(runs.x, 2)1  12.861809   1.030906  12.476  < 2e-16 ***
## poly(runs.y, 2)1:poly(runs.x, 2)2 -13.980628   0.782867 -17.858  < 2e-16 ***
## poly(runs.y, 2)2:poly(runs.x, 2)2  -1.367661   1.006405  -1.359 0.175558
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05004 on 219 degrees of freedom
## Multiple R-squared:  0.6912, Adjusted R-squared:  0.6828
## F-statistic: 81.72 on 6 and 219 DF,  p-value: < 2.2e-16
```

This is a table of R_squared and MSE for both models

```
models = c("Pythagorean_Expectation","Ridge_Regression")
R_squared=c(0.69,0.79)
MSE = c(0.0025,0.0024)

table = data.frame(models,R_squared,MSE)
table
```

```
##                    models R_squared    MSE
## 1 Pythagorean_Expectation      0.69 0.0025
## 2        Ridge_Regression      0.79 0.0024
```

The dataset is consisted of 51 columns and the following is the description of each column

team year id average_age: Average pitcher age runs_per_game.x: Runs scored per game wins losses win_loss_percentage: win/loss ERA: Pitching ERA run_average_9: runs allowed per 9 innings games.x: games played with the pitcher complete_game shutouts: No runs allowed and complete games saves: saves made by the pitcher innings_pitched hits.x:hits allowed runs.x:runs allowed earned_runs:earned runs allowed home_runs walks:walks allowed strikeouts.xpitcher strikeouts hit_batter batters_faced WHIP: (Walks + Hits) / Total Innings Pitched hits_9: Hits per 9 innings homeruns_9: Homeruns per 9 innings walks_9: walks per 9 innings strikeouts_9: strikeouts per 9 innings strikeout_walk: strikeout/walk ratio average_batter_age runs_per_game.y: runs scored per game games.y: games played with the batter plate_appearances: Times batter appeared on plate at_bats runs.y: runs scored by batter hits.y: hits made by batter doubles triples homeruns RBI:Runs batted in bases_on_balls bases_on_balls: Walks strikeouts: batter strikeouts batting_average: Batting average OBP: On base percentage SLG: Slugging percentage OPS: On base + slugging percentage total_bases: Total bases GDP: Double plays grounded into HBP: Times hit by pitch sacrifice_hits: Sacrifice hits sacrifice_flies: Sacrifice flies IBB: Intentional bases on balls