DATA 621 Homework 1

Fomba Kassoh

2024-14-03

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

Load Required Libraries

```
options(repos = c(CRAN = "https://cloud.r-project.org"))

# Function to install a package if not already installed
install_if_needed <- function(pkg) {
  if (!requireNamespace(pkg, quietly = TRUE)) {
    install.packages(pkg)

  }
}

# List of packages to check and install if necessary
required_packages <- c("dplyr", "ggplot2", "lubridate", "tidyr",
                       "caret", "stats", "tidyverse", "corrplot",
                       "plotly", "DT", "DataExplorer", "mice", "kableExtra",
                       "reshape", "reshape2", "factoextra", "psych", "GGally", "tinytex"
)

# Loop through the list and install packages only if needed
for (pkg in required_packages) {
  install_if_needed(pkg)
}
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
# Function to suppress package startup messages
suppressPackageStartupMessages({
  library(dplyr)
  library(ggplot2)
  library(tidyr)
  library(caret) # for data splitting and pre-processing
  library(stats)
  library(tidyverse)
  library(corrplot)
  library(plotly)
  library(DT)
```

```
  library(DataExplorer)
  library(mice)
  library(kableExtra)
  library(reshape)
  library(reshape2)
  library(factoextra)
  library(psych)
  library(GGally)
  library(tinytex)
})
```

1.Modeling Assumptions

- **Linearity:** The mean values of the outcome variable TARGET_WINS for each increment of the predictor(s) lie on a strainght line. There is a linear relationship between predictors and the target.
- **Linearity:** There exist a relationship between the independent variables and the dependent variable TARGET_WINS.
- **Independence:** The residuals are independent.
- **Homoscedasticity:** At each level of the predictor variables, the variance of the residual terms are constant.
- **Independence:** All the values of the dependent variable are independent. **Normally distributed errors:** The residuals are random, normally distributed with a mean 0.

2. Data Exploration

Import the dataset

```
money_ball_train = read_csv("https://raw.githubusercontent.com/hawa1983/DATA-621/main/moneyball-training
money_ball_evaluate = read_csv("https://raw.githubusercontent.com/hawa1983/DATA-621/main/moneyball-evalu
str(money_ball_train)
```

```
## spc_tbl_ [2,276 x 17] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ INDEX           : num [1:2276] 1 2 3 4 5 6 7 8 11 12 ...
##  $ TARGET_WINS     : num [1:2276] 39 70 86 70 82 75 80 85 86 76 ...
##  $ TEAM_BATTING_H  : num [1:2276] 1445 1339 1377 1387 1297 ...
##  $ TEAM_BATTING_2B : num [1:2276] 194 219 232 209 186 200 179 171 197 213 ...
##  $ TEAM_BATTING_3B : num [1:2276] 39 22 35 38 27 36 54 37 40 18 ...
##  $ TEAM_BATTING_HR : num [1:2276] 13 190 137 96 102 92 122 115 114 96 ...
##  $ TEAM_BATTING_BB : num [1:2276] 143 685 602 451 472 443 525 456 447 441 ...
##  $ TEAM_BATTING_SO : num [1:2276] 842 1075 917 922 920 ...
##  $ TEAM_BASERUN_SB : num [1:2276] NA 37 46 43 49 107 80 40 69 72 ...
##  $ TEAM_BASERUN_CS : num [1:2276] NA 28 27 30 39 59 54 36 27 34 ...
##  $ TEAM_BATTING_HBP: num [1:2276] NA NA NA NA NA NA NA NA NA NA ...
##  $ TEAM_PITCHING_H : num [1:2276] 9364 1347 1377 1396 1297 ...
##  $ TEAM_PITCHING_HR: num [1:2276] 84 191 137 97 102 92 122 116 114 96 ...
##  $ TEAM_PITCHING_BB: num [1:2276] 927 689 602 454 472 443 525 459 447 441 ...
##  $ TEAM_PITCHING_SO: num [1:2276] 5456 1082 917 928 920 ...
##  $ TEAM_FIELDING_E : num [1:2276] 1011 193 175 164 138 ...
##  $ TEAM_FIELDING_DP: num [1:2276] NA 155 153 156 168 149 186 136 169 159 ...
##  - attr(*, "spec")=
##   .. cols(
```

```
##   ..   INDEX = col_double(),
##   ..   TARGET_WINS = col_double(),
##   ..   TEAM_BATTING_H = col_double(),
##   ..   TEAM_BATTING_2B = col_double(),
##   ..   TEAM_BATTING_3B = col_double(),
##   ..   TEAM_BATTING_HR = col_double(),
##   ..   TEAM_BATTING_BB = col_double(),
##   ..   TEAM_BATTING_SO = col_double(),
##   ..   TEAM_BASERUN_SB = col_double(),
##   ..   TEAM_BASERUN_CS = col_double(),
##   ..   TEAM_BATTING_HBP = col_double(),
##   ..   TEAM_PITCHING_H = col_double(),
##   ..   TEAM_PITCHING_HR = col_double(),
##   ..   TEAM_PITCHING_BB = col_double(),
##   ..   TEAM_PITCHING_SO = col_double(),
##   ..   TEAM_FIELDING_E = col_double(),
##   ..   TEAM_FIELDING_DP = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
str(money_ball_evaluate)
```

```
## spc_tbl_ [259 x 16] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ INDEX           : num [1:259] 9 10 14 47 60 63 74 83 98 120 ...
##  $ TEAM_BATTING_H  : num [1:259] 1209 1221 1395 1539 1445 ...
##  $ TEAM_BATTING_2B : num [1:259] 170 151 183 309 203 236 219 158 177 212 ...
##  $ TEAM_BATTING_3B : num [1:259] 33 29 29 29 68 53 55 42 78 42 ...
##  $ TEAM_BATTING_HR : num [1:259] 83 88 93 159 5 10 37 33 23 58 ...
##  $ TEAM_BATTING_BB : num [1:259] 447 516 509 486 95 215 568 356 466 452 ...
##  $ TEAM_BATTING_SO : num [1:259] 1080 929 816 914 416 377 527 609 689 584 ...
##  $ TEAM_BASERUN_SB : num [1:259] 62 54 59 148 NA NA 365 185 150 52 ...
##  $ TEAM_BASERUN_CS : num [1:259] 50 39 47 57 NA NA NA NA NA NA ...
##  $ TEAM_BATTING_HBP: num [1:259] NA NA NA 42 NA NA NA NA NA NA ...
##  $ TEAM_PITCHING_H : num [1:259] 1209 1221 1395 1539 3902 ...
##  $ TEAM_PITCHING_HR: num [1:259] 83 88 93 159 14 20 40 39 25 62 ...
##  $ TEAM_PITCHING_BB: num [1:259] 447 516 509 486 257 420 613 418 497 482 ...
##  $ TEAM_PITCHING_SO: num [1:259] 1080 929 816 914 1123 ...
##  $ TEAM_FIELDING_E : num [1:259] 140 135 156 124 616 572 490 328 226 184 ...
##  $ TEAM_FIELDING_DP: num [1:259] 156 164 153 154 130 105 NA 104 132 145 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   INDEX = col_double(),
##   ..   TEAM_BATTING_H = col_double(),
##   ..   TEAM_BATTING_2B = col_double(),
##   ..   TEAM_BATTING_3B = col_double(),
##   ..   TEAM_BATTING_HR = col_double(),
##   ..   TEAM_BATTING_BB = col_double(),
##   ..   TEAM_BATTING_SO = col_double(),
##   ..   TEAM_BASERUN_SB = col_double(),
##   ..   TEAM_BASERUN_CS = col_double(),
##   ..   TEAM_BATTING_HBP = col_double(),
##   ..   TEAM_PITCHING_H = col_double(),
##   ..   TEAM_PITCHING_HR = col_double(),
##   ..   TEAM_PITCHING_BB = col_double(),
```

```
##   ..   TEAM_PITCHING_SO = col_double(),
##   ..   TEAM_FIELDING_E = col_double(),
##   ..   TEAM_FIELDING_DP = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

i. Summarise the data Get an overview of the data through summary statistics like mean, median, and standard deviation.

```r
# Load necessary libraries
library(psych)

# Assuming your data is stored in money_ball_train

# Summary statistics
summary_stats <- summary(money_ball_train)

# Descriptive statistics
descriptive_stats <- describe(money_ball_train)

descriptive_stats
```

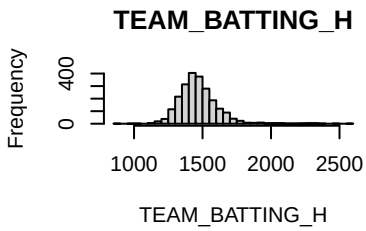
```
##                  vars    n    mean      sd median trimmed    mad  min   max
## INDEX               1 2276 1268.46  736.35 1270.5 1268.57 952.57    1  2535
## TARGET_WINS         2 2276   80.79   15.75   82.0   81.31  14.83    0   146
## TEAM_BATTING_H      3 2276 1469.27  144.59 1454.0 1459.04 114.16  891  2554
## TEAM_BATTING_2B     4 2276  241.25   46.80  238.0  240.40  47.44   69   458
## TEAM_BATTING_3B     5 2276   55.25   27.94   47.0   52.18  23.72    0   223
## TEAM_BATTING_HR     6 2276   99.61   60.55  102.0   97.39  78.58    0   264
## TEAM_BATTING_BB     7 2276  501.56  122.67  512.0  512.18  94.89    0   878
## TEAM_BATTING_SO     8 2174  735.61  248.53  750.0  742.31 284.66    0  1399
## TEAM_BASERUN_SB     9 2145  124.76   87.79  101.0  110.81  60.79    0   697
## TEAM_BASERUN_CS    10 1504   52.80   22.96   49.0   50.36  17.79    0   201
## TEAM_BATTING_HBP   11  191   59.36   12.97   58.0   58.86  11.86   29    95
## TEAM_PITCHING_H    12 2276 1779.21 1406.84 1518.0 1555.90 174.95 1137 30132
## TEAM_PITCHING_HR   13 2276  105.70   61.30  107.0  103.16  74.13    0   343
## TEAM_PITCHING_BB   14 2276  553.01  166.36  536.5  542.62  98.59    0  3645
## TEAM_PITCHING_SO   15 2174  817.73  553.09  813.5  796.93 257.23    0 19278
## TEAM_FIELDING_E    16 2276  246.48  227.77  159.0  193.44  62.27   65  1898
## TEAM_FIELDING_DP   17 1990  146.39   26.23  149.0  147.58  23.72   52   228
##                  range  skew kurtosis    se
## INDEX             2534  0.00    -1.22 15.43
## TARGET_WINS        146 -0.40     1.03  0.33
## TEAM_BATTING_H    1663  1.57     7.28  3.03
## TEAM_BATTING_2B    389  0.22     0.01  0.98
## TEAM_BATTING_3B    223  1.11     1.50  0.59
## TEAM_BATTING_HR    264  0.19    -0.96  1.27
## TEAM_BATTING_BB    878 -1.03     2.18  2.57
## TEAM_BATTING_SO   1399 -0.30    -0.32  5.33
## TEAM_BASERUN_SB    697  1.97     5.49  1.90
## TEAM_BASERUN_CS    201  1.98     7.62  0.59
## TEAM_BATTING_HBP    66  0.32    -0.11  0.94
## TEAM_PITCHING_H  28995 10.33   141.84 29.49
```

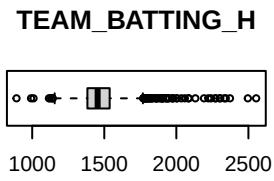
```
## TEAM_PITCHING_HR   343  0.29    -0.60  1.28
## TEAM_PITCHING_BB  3645  6.74    96.97  3.49
## TEAM_PITCHING_SO 19278 22.17   671.19 11.86
## TEAM_FIELDING_E   1833  2.99    10.97  4.77
## TEAM_FIELDING_DP   176 -0.39     0.18  0.59
```

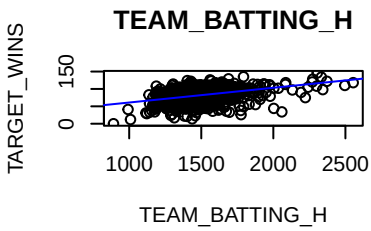
ii. Visualize the data for feature distributions, relationships and outlier Detection

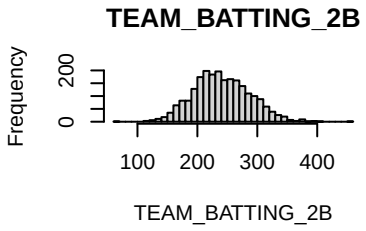
- 1. **Distribution Analysis:** Plot histograms to understand the distribution of each variable.
- 2. **Box Plots:** Create box plots to detect outliers and understand the distribution of each variable.
- 3. **Bivariate Analysis:** Examine relationships between the target variable and independent variables.

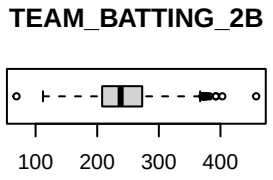
```r
par(mfrow = c(3, 3))  # Set up the plotting area for multiple plots

# List of column names to analyze
columns_to_analyze <- c("TEAM_BATTING_H", "TEAM_BATTING_2B", "TEAM_BATTING_3B",
                        "TEAM_BATTING_HR", "TEAM_BATTING_BB", "TEAM_BATTING_SO",
                        "TEAM_BASERUN_SB", "TEAM_BASERUN_CS", "TEAM_BATTING_HBP",
                        "TEAM_PITCHING_H", "TEAM_PITCHING_HR", "TEAM_PITCHING_BB",
                        "TEAM_PITCHING_SO", "TEAM_FIELDING_E", "TEAM_FIELDING_DP")

# Loop through each specified column name
for (column_name in columns_to_analyze) {
  column_data <- money_ball_train[[column_name]]  # Extract column data

  # Ensure the column is numeric
  if (is.numeric(column_data)) {
    # Remove NAs specific to the current column and TARGET_WINS
    valid_indices <- complete.cases(column_data, money_ball_train$TARGET_WINS)
    clean_data <- money_ball_train[valid_indices, ]

    # Plot histogram
    hist(clean_data[[column_name]], main = column_name,
         xlab = column_name, breaks = 51)

    # Plot boxplot
    boxplot(clean_data[[column_name]], main = column_name,
            horizontal = TRUE)

    # Plot scatterplot and fit a regression line
    plot(clean_data[[column_name]], clean_data$TARGET_WINS,
         main = column_name,
         xlab = column_name,
         ylab = "TARGET_WINS")

    # Add regression line
    abline(lm(TARGET_WINS ~ get(column_name), data = clean_data), col = 'blue')
  } else {
    message("Skipping non-numeric column: ", column_name)
  }
}
```

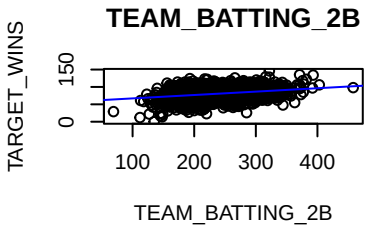

TEAM_BATTING_H
Frequency
0
400
1000
1500
2000
2500
TEAM_BATTING_H

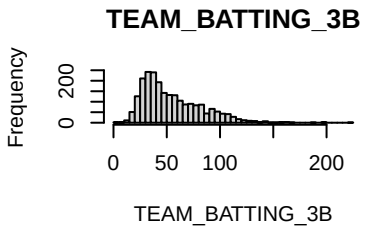

TEAM_BATTING_H
1000
1500
2000
2500

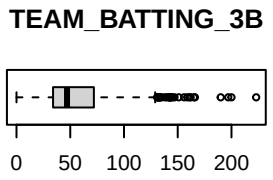

TEAM_BATTING_H
TARGET_WINS
0
150
1000
1500
2000
2500
TEAM_BATTING_H

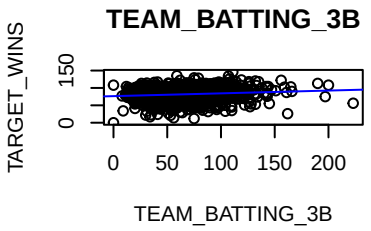

TEAM_BATTING_2B
Frequency
0
200
100
200
300
400
TEAM_BATTING_2B

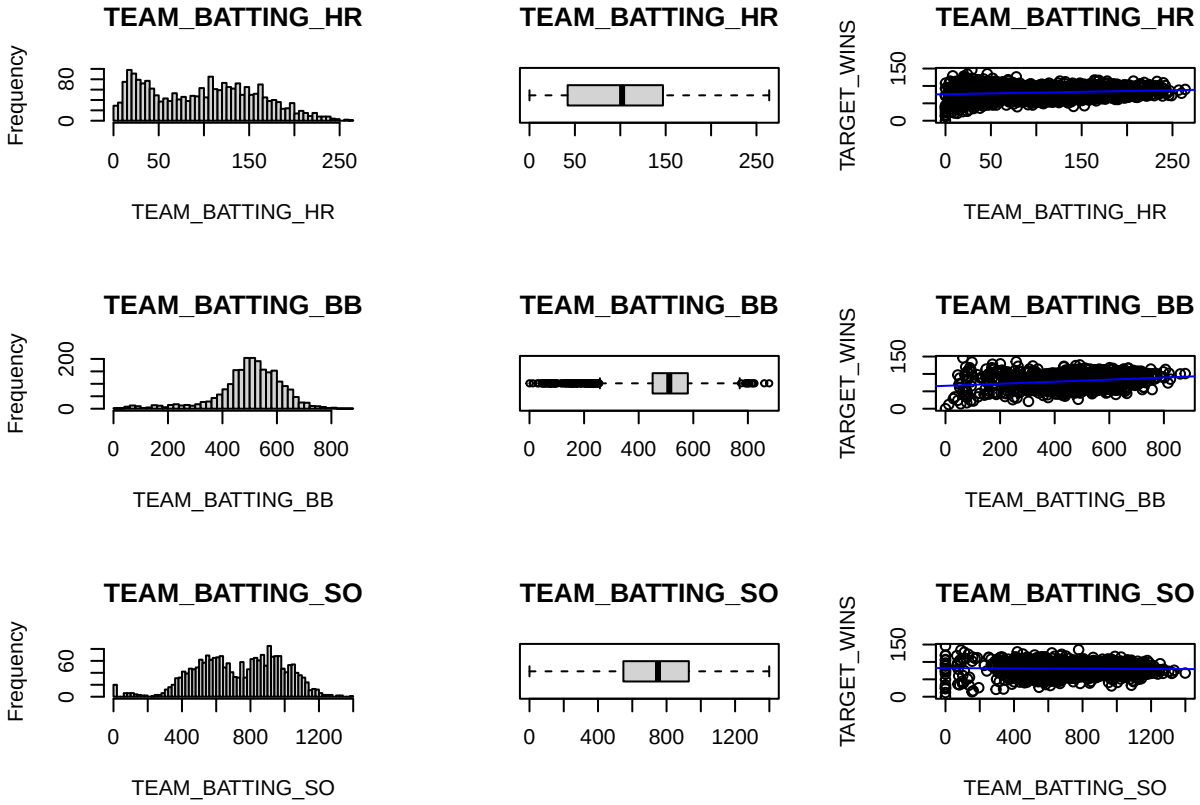

TEAM_BATTING_2B
100
200
300
400

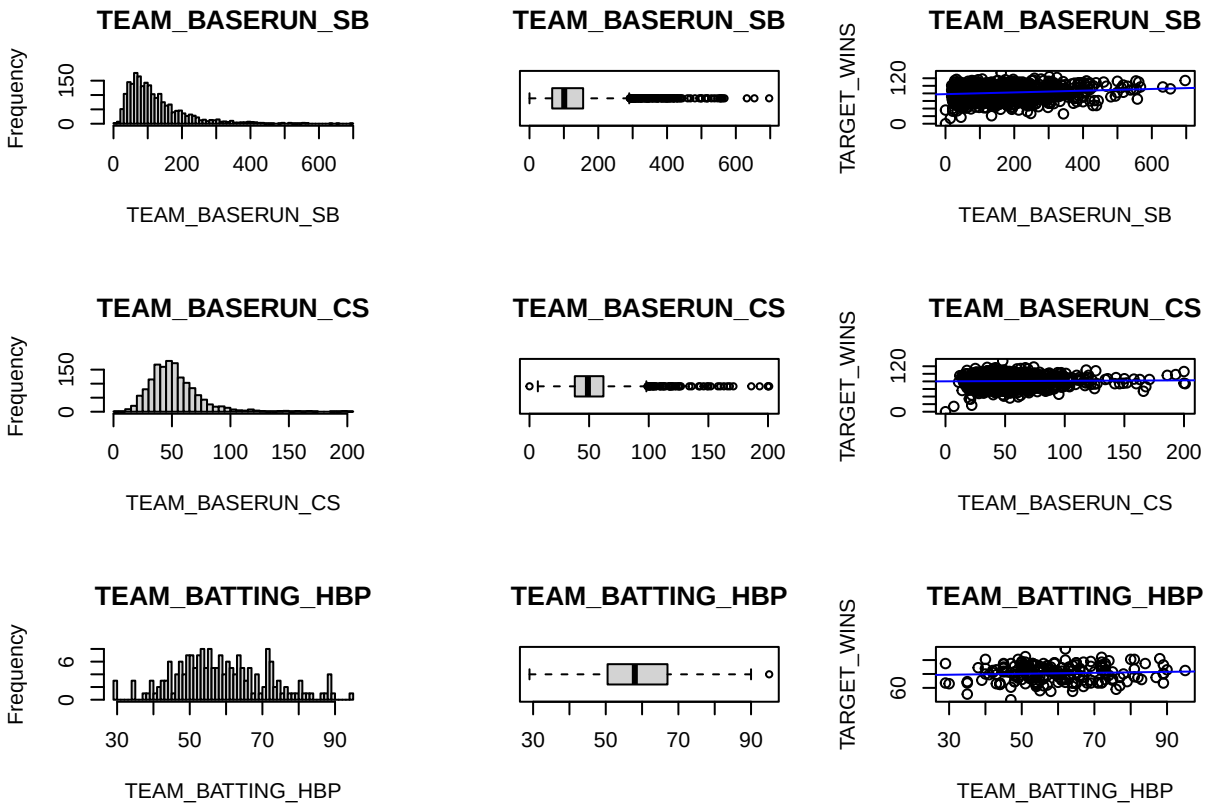

TEAM_BATTING_2B
TARGET_WINS
0
150
100
200
300
400
TEAM_BATTING_2B

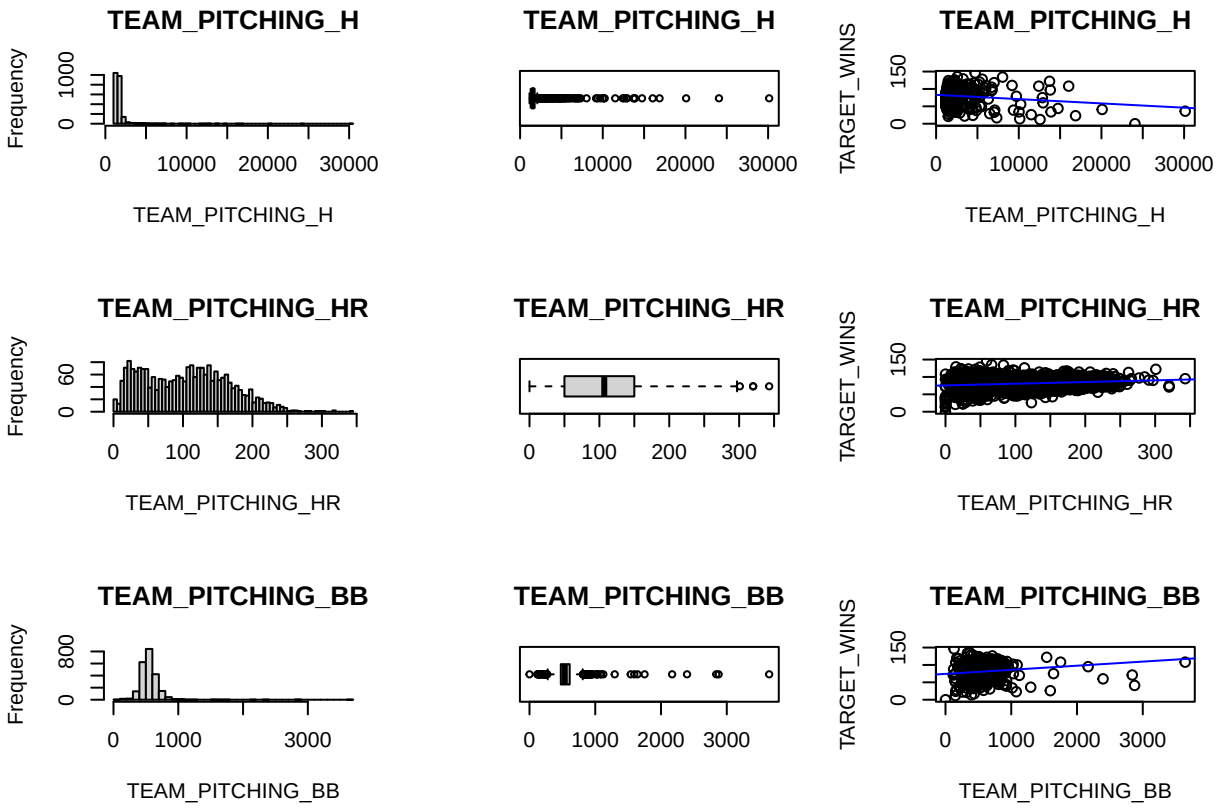

TEAM_BATTING_3B
Frequency
0
200
0
50
100
200
TEAM_BATTING_3B

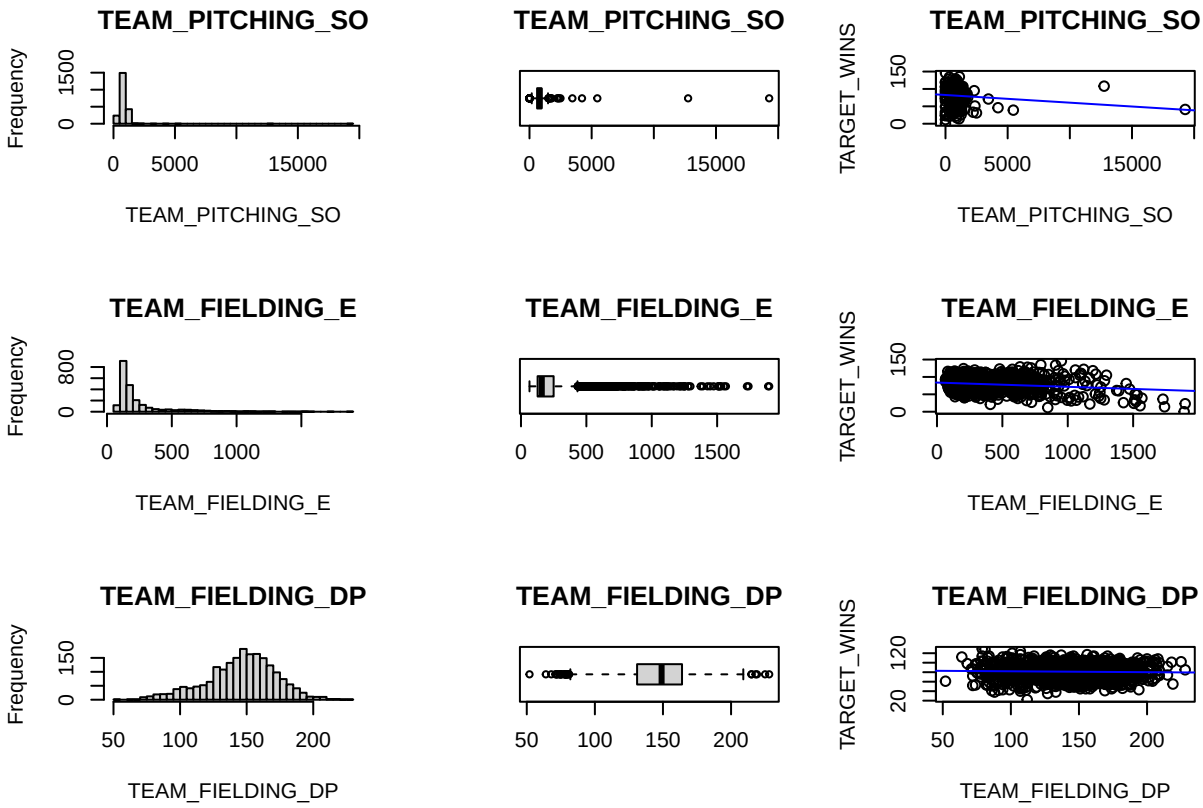

TEAM_BATTING_3B
0
50
100
150
200


TEAM_BATTING_3B
TARGET_WINS
0
150
0
50
100
150
200
TEAM_BATTING_3B


TEAM_BATTING_HR
Frequency
0
80
0
50
150
250
TEAM_BATTING_HR
TEAM_BATTING_HR
0
50
150
250
TEAM_BATTING_HR
TARGET_WINS
0
150
0
50
150
250
TEAM_BATTING_HR
TEAM_BATTING_BB
Frequency
0
200
0
200
400
600
800
TEAM_BATTING_BB
TEAM_BATTING_BB
0
200
400
600
800
TEAM_BATTING_BB
TARGET_WINS
0
150
0
200
400
600
800
TEAM_BATTING_BB
TEAM_BATTING_SO
Frequency
0
60
0
400
800
1200
TEAM_BATTING_SO
TEAM_BATTING_SO
0
400
800
1200
TEAM_BATTING_SO
TARGET_WINS
0
150
0
400
800
1200
TEAM_BATTING_SO


TEAM_BASERUN_SB
Frequency
0
150
0
200
400
600
TEAM_BASERUN_SB
TEAM_BASERUN_SB
0
200
400
600
TEAM_BASERUN_SB
TARGET_WINS
0
120
0
200
400
600
TEAM_BASERUN_SB
TEAM_BASERUN_CS
Frequency
0
150
0
50
100
150
200
TEAM_BASERUN_CS
TEAM_BASERUN_CS
0
50
100
150
200
TEAM_BASERUN_CS
TARGET_WINS
0
120
0
50
100
150
200
TEAM_BASERUN_CS
TEAM_BATTING_HBP
Frequency
0
6
30
50
70
90
TEAM_BATTING_HBP
TEAM_BATTING_HBP
30
50
70
90
TEAM_BATTING_HBP
TARGET_WINS
60
30
50
70
90
TEAM_BATTING_HBP


TEAM_PITCHING_H
Frequency
0
1000
0
10000
20000
30000
TEAM_PITCHING_H
TEAM_PITCHING_H
0
10000
20000
30000
TEAM_PITCHING_H
TARGET_WINS
0
150
0
10000
20000
30000
TEAM_PITCHING_H
TEAM_PITCHING_HR
Frequency
0
60
0
100
200
300
TEAM_PITCHING_HR
TEAM_PITCHING_HR
0
100
200
300
TEAM_PITCHING_HR
TARGET_WINS
0
150
0
100
200
300
TEAM_PITCHING_HR
TEAM_PITCHING_BB
Frequency
0
800
0
1000
3000
TEAM_PITCHING_BB
TEAM_PITCHING_BB
0
1000
2000
3000
TEAM_PITCHING_BB
TARGET_WINS
0
150
0
1000
2000
3000
TEAM_PITCHING_BB



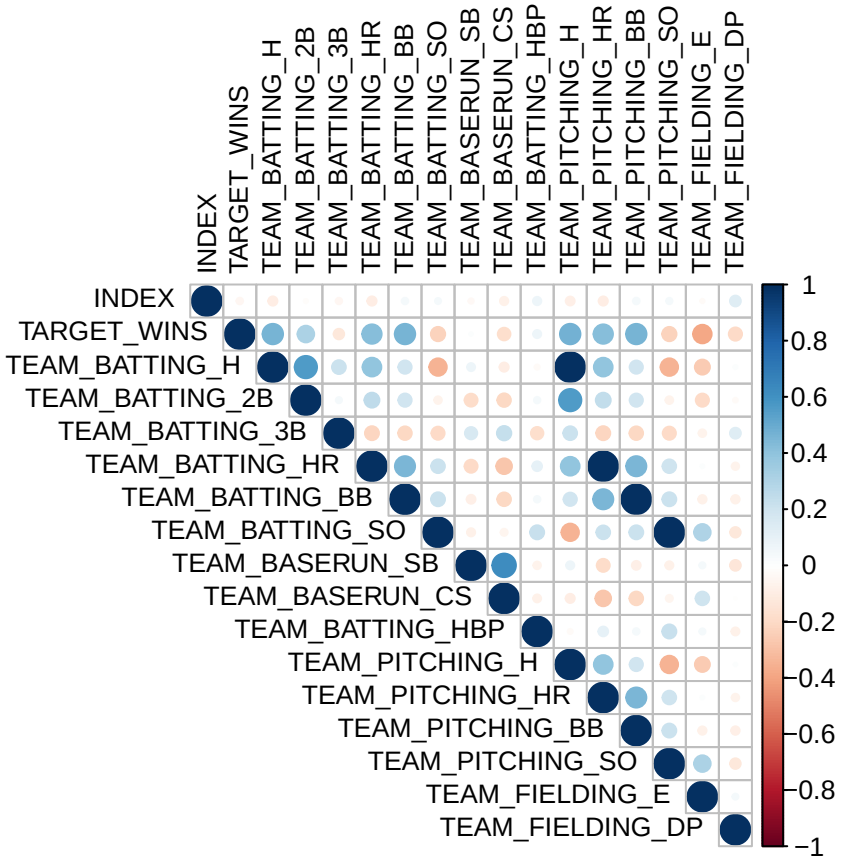
Based on the summary statistics and visualizations:

- **TARGET_WINS:** The mean and median are close, indicating symmetry. The spread of the data (from minimum to maximum) does not show extreme values, suggesting no significant outliers.
- **TEAM_BATTING_H:** The mean is slightly higher than the median, suggesting a slight positive skew. The presence of outliers is likely due to extreme values in the data.
- **TEAM_BATTING_2B:** The mean and median are close, suggesting symmetry. The range of data is within typical bounds, indicating no significant outliers.
- **TEAM_BATTING_3B:** A larger difference between the mean and median indicates positive skewness. The presence of extreme values suggests outliers.
- **TEAM_BATTING_HR:** A slight difference between mean and median indicates slight positive skewness. The data likely contains outliers due to extreme values.
- **TEAM_BATTING_BB:** Mean and median are close, indicating symmetry. The range suggests no significant outliers.
- **TEAM_BATTING_SO:** Slight positive skewness is indicated by a higher mean than median. The data shows the presence of outliers.
- **TEAM_BASERUN_SB and TEAM_BASERUN_CS:** Both show positive skewness with a higher mean than median, and the presence of outliers is suggested by extreme values in the data.
- **TEAM_BATTING_HBP:** Slight positive skewness is indicated by the mean being slightly higher than the median, but no extreme outliers are detected.
- **TEAM_PITCHING_H:** Mean and median are close, suggesting symmetry. The range of data does not suggest outliers.
- **TEAM_PITCHING_HR, TEAM_PITCHING_BB, TEAM_PITCHING_SO:** These show slight positive skewness (mean $>$ median), with varying indications of outliers based on the range of data.
- **TEAM_FIELDING_E and TEAM_FIELDING_DP:** Both show symmetry (mean median)

and no significant outliers.

iii. Correlation Analysis Create a correlation matrix to identify relationships between variables.

```
# Correlation matrix
cor_matrix <- cor(money_ball_train, use = "complete.obs")

# Visualize the correlation matrix
library(corrplot)
corrplot(cor_matrix, method = "circle", type = "upper", tl.cex = 0.8, tl.col = "black")
```



Based on the correlation matrix provided in the plot:

1. **Strong Positive Correlations:**
   - **TEAM_BATTING_H** and **TEAM_BATTING_2B**: There is a strong positive correlation between these two variables, indicating that as the number of hits increases, the number of doubles also tends to increase.
   - **TEAM_PITCHING_H** and **TEAM_PITCHING_BB**: These two pitching-related variables are also strongly positively correlated, suggesting that teams that give up more hits also tend to issue more walks.
   - **TEAM_PITCHING_H** and **TEAM_PITCHING_HR**: There's a notable positive correlation between the number of hits and home runs allowed by the team's pitching, indicating a relationship between general hitting success against the team and home run success.
2. **Moderate Positive Correlations:**

- **TEAM_BATTING_HR** and **TEAM_BATTING_2B**: The number of home runs and doubles have a moderate positive correlation, which may indicate that teams with more powerful hitting overall (resulting in more home runs) also hit more doubles.
- **TEAM_BASERUN_SB** and **TEAM_BASERUN_CS**: Stolen bases and caught stealing are moderately correlated, indicating that teams that attempt more steals have both higher successful steals and higher times caught stealing.

3. **Weak or No Correlations:**
   - **TARGET_WINS** with most other variables shows generally weak correlations. This suggests that individual batting, baserunning, or pitching statistics alone might not be strong predictors of team wins, which is often the case because wins are influenced by a combination of factors.
   - **TEAM_FIELDING_E** and other variables: Fielding errors do not show a strong correlation with other variables, indicating that errors may not be directly related to the batting or pitching performances.
4. **Negative Correlations:**
   - There are some weak negative correlations, but none stand out as particularly strong or meaningful in this plot. Negative correlations are generally less pronounced, and none seem to be significantly impactful in the overall dataset.

iv. Check for missing values Some of the variables have missing values.

```
# Total number of rows in the dataset
total_rows <- nrow(money_ball_train)

# Check for missing values and calculate the percentage of missing values
missing_values <- colSums(is.na(money_ball_train))
missing_percent <- (missing_values / total_rows) * 100

# Create a dataframe that includes both the count and percentage of missing values
missing_values_df <- data.frame(
  Column = names(missing_values),
  Missing_Values = missing_values,
  Percent_Missing = missing_percent
)

# Filter only columns with missing values and sort the dataframe by Missing_Values in descending order
missing_values_df <- missing_values_df %>%
  filter(Missing_Values > 0) %>%             # Only include columns with missing values
  arrange(desc(Missing_Values))              # Sort by missing values

# View the sorted table
print(missing_values_df)
```

```
##                            Column Missing_Values Percent_Missing
## TEAM_BATTING_HBP TEAM_BATTING_HBP           2085       91.608084
## TEAM_BASERUN_CS   TEAM_BASERUN_CS            772       33.919156
## TEAM_FIELDING_DP TEAM_FIELDING_DP            286       12.565905
## TEAM_BASERUN_SB   TEAM_BASERUN_SB            131        5.755712
## TEAM_BATTING_SO   TEAM_BATTING_SO            102        4.481547
## TEAM_PITCHING_SO TEAM_PITCHING_SO            102        4.481547
```

3. Data Preparation

i. Impute Missing Values

a. Imputation Methods The following methods will be used for missing values imputation

```
# Calculate the percentage of missing values for each column
missing_values_df <- data.frame(
  Column = names(money_ball_train),
  Percent_Missing = colMeans(is.na(money_ball_train)) * 100
)

# Filter only columns with missing values
missing_values_with_na <- missing_values_df[missing_values_df$Percent_Missing > 0, ]

# Apply recommendations based on missing value percentages
missing_values_with_na$Imputation_Recommendation <- ifelse(
  missing_values_with_na$Percent_Missing <= 5, "Mean/Median Imputation",
  ifelse(
    missing_values_with_na$Percent_Missing > 5 & missing_values_with_na$Percent_Missing <= 20, "KNN or M
    ifelse(
      missing_values_with_na$Percent_Missing > 20 & missing_values_with_na$Percent_Missing <= 50, "Multi
      "Consider Dropping or Advanced Techniques"
    )
  )
)

# Display the recommendations
print(missing_values_with_na)
```

```
##                            Column Percent_Missing
## TEAM_BATTING_SO   TEAM_BATTING_SO        4.481547
## TEAM_BASERUN_SB   TEAM_BASERUN_SB        5.755712
## TEAM_BASERUN_CS   TEAM_BASERUN_CS       33.919156
## TEAM_BATTING_HBP TEAM_BATTING_HBP       91.608084
## TEAM_PITCHING_SO TEAM_PITCHING_SO        4.481547
## TEAM_FIELDING_DP TEAM_FIELDING_DP       12.565905
##                                    Imputation_Recommendation
## TEAM_BATTING_SO                       Mean/Median Imputation
## TEAM_BASERUN_SB                    KNN or Multiple Imputation
## TEAM_BASERUN_CS  Multiple Imputation or Predictive Modeling
## TEAM_BATTING_HBP   Consider Dropping or Advanced Techniques
## TEAM_PITCHING_SO                      Mean/Median Imputation
## TEAM_FIELDING_DP                   KNN or Multiple Imputation
```

```
money_ball_train_imputed <- money_ball_train

# Impute or drop based on the recommendations
for (i in 1:nrow(missing_values_with_na)) {
  column_name <- missing_values_with_na$Column[i]
```

```
  recommendation <- missing_values_with_na$Imputation_Recommendation[i]

  if (recommendation == "Mean/Median Imputation") {
    # Mean/Median Imputation
    money_ball_train_imputed[[column_name]][is.na(money_ball_train_imputed[[column_name]])] <-
      median(money_ball_train_imputed[[column_name]], na.rm = TRUE)  # Or use mean if preferred
  } else if (recommendation == "KNN or Multiple Imputation") {
    # For simplicity, using median imputation here, but you can replace with more advanced methods like
    money_ball_train_imputed[[column_name]][is.na(money_ball_train_imputed[[column_name]])] <-
      median(money_ball_train_imputed[[column_name]], na.rm = TRUE)
  } else if (recommendation == "Multiple Imputation or Predictive Modeling") {
    # For simplicity, using median imputation here, but you can replace with more advanced methods
    money_ball_train_imputed[[column_name]][is.na(money_ball_train_imputed[[column_name]])] <-
      median(money_ball_train_imputed[[column_name]], na.rm = TRUE)
  } else if (recommendation == "Consider Dropping") {
    # Drop the variable if recommended
    money_ball_train_imputed[[column_name]] <- NULL
  }
}

# Display the updated data frame
head(money_ball_train_imputed)
```

b. Perform the imputations for training data

```
## # A tibble: 6 x 17
##   INDEX TARGET_WINS TEAM_BATTING_H TEAM_BATTING_2B TEAM_BATTING_3B
##   <dbl>       <dbl>          <dbl>           <dbl>           <dbl>
## 1     1          39           1445             194              39
## 2     2          70           1339             219              22
## 3     3          86           1377             232              35
## 4     4          70           1387             209              38
## 5     5          82           1297             186              27
## 6     6          75           1279             200              36
## # i 12 more variables: TEAM_BATTING_HR <dbl>, TEAM_BATTING_BB <dbl>,
## #   TEAM_BATTING_SO <dbl>, TEAM_BASERUN_SB <dbl>, TEAM_BASERUN_CS <dbl>,
## #   TEAM_BATTING_HBP <dbl>, TEAM_PITCHING_H <dbl>, TEAM_PITCHING_HR <dbl>,
## #   TEAM_PITCHING_BB <dbl>, TEAM_PITCHING_SO <dbl>, TEAM_FIELDING_E <dbl>,
## #   TEAM_FIELDING_DP <dbl>
```

```
money_ball_evaluate_imputed <- money_ball_evaluate

# Impute or drop based on the recommendations
for (i in 1:nrow(missing_values_with_na)) {
  column_name <- missing_values_with_na$Column[i]
  recommendation <- missing_values_with_na$Imputation_Recommendation[i]

  if (recommendation == "Mean/Median Imputation") {
    # Mean/Median Imputation
    money_ball_evaluate_imputed[[column_name]][is.na(money_ball_evaluate_imputed[[column_name]])] <-
```

```
      median(money_ball_evaluate_imputed[[column_name]], na.rm = TRUE)  # Or use mean if preferred
  } else if (recommendation == "KNN or Multiple Imputation") {
    # For simplicity, using median imputation here, but you can replace with more advanced methods like
    money_ball_evaluate_imputed[[column_name]][is.na(money_ball_evaluate_imputed[[column_name]])] <-
      median(money_ball_evaluate_imputed[[column_name]], na.rm = TRUE)
  } else if (recommendation == "Multiple Imputation or Predictive Modeling") {
    # For simplicity, using median imputation here, but you can replace with more advanced methods
    money_ball_evaluate_imputed[[column_name]][is.na(money_ball_evaluate_imputed[[column_name]])] <-
      median(money_ball_evaluate_imputed[[column_name]], na.rm = TRUE)
  } else if (recommendation == "Consider Dropping") {
    # Drop the variable if recommended
    money_ball_evaluate_imputed[[column_name]] <- NULL
  }
}

# Display the updated data frame
head(money_ball_evaluate_imputed)
```

c. Perform the imputations for evaluation data

```
## # A tibble: 6 x 16
##   INDEX TEAM_BATTING_H TEAM_BATTING_2B TEAM_BATTING_3B TEAM_BATTING_HR
##   <dbl>          <dbl>           <dbl>           <dbl>           <dbl>
## 1     9           1209             170              33              83
## 2    10           1221             151              29              88
## 3    14           1395             183              29              93
## 4    47           1539             309              29             159
## 5    60           1445             203              68               5
## 6    63           1431             236              53              10
## # i 11 more variables: TEAM_BATTING_BB <dbl>, TEAM_BATTING_SO <dbl>,
## #   TEAM_BASERUN_SB <dbl>, TEAM_BASERUN_CS <dbl>, TEAM_BATTING_HBP <dbl>,
## #   TEAM_PITCHING_H <dbl>, TEAM_PITCHING_HR <dbl>, TEAM_PITCHING_BB <dbl>,
## #   TEAM_PITCHING_SO <dbl>, TEAM_FIELDING_E <dbl>, TEAM_FIELDING_DP <dbl>
```

ii. Transform the data based on the summary statistic and visualizations Transform the data to correct skewness, non-linearity, or to stabilize variance (i.e., address heteroscedasticity).

a. Data transformation methods Base on the summary statistics and visualizations, the following transformations will be applied to the data

```
# Create a data frame with variables, transformation methods, and reasons
transformation_recommendations <- data.frame(
  Variable = c("TEAM_BATTING_H", "TEAM_BATTING_2B", "TEAM_BATTING_3B",
               "TEAM_BATTING_HR", "TEAM_BATTING_BB", "TEAM_BATTING_SO",
               "TEAM_BASERUN_SB", "TEAM_BASERUN_CS", "TEAM_BATTING_HBP",
               "TEAM_PITCHING_H", "TEAM_PITCHING_HR", "TEAM_PITCHING_BB",
               "TEAM_PITCHING_SO", "TEAM_FIELDING_E", "TEAM_FIELDING_DP"),
  Transformation_Method = c("Logarithmic", "None", "Square Root",
                            "Logarithmic", "None", "Logarithmic",
                            "Logarithmic", "Square Root", "None",
                            "Logarithmic", "Logarithmic", "Logarithmic",
                            "Logarithmic", "Logarithmic", "None"),
```

```
  Reason = c("The distribution appears right-skewed.",
             "The distribution is fairly normal, and the relationship with TARGET_WINS seems linear.",
             "The distribution is right-skewed with some extreme values.",
             "The distribution is right-skewed.",
             "The distribution and the relationship with TARGET_WINS are acceptable.",
             "The distribution is right-skewed.",
             "The distribution is highly right-skewed.",
             "The distribution is right-skewed.",
             "The distribution is relatively normal.",
             "The distribution is extremely right-skewed.",
             "The distribution is right-skewed.",
             "The distribution is right-skewed with outliers.",
             "The distribution is right-skewed.",
             "The distribution is right-skewed.",
             "The distribution is fairly normal, and the relationship with TARGET_WINS appears linear.")
)

# Display the data frame
print(transformation_recommendations)
```

```
##            Variable Transformation_Method
## 1    TEAM_BATTING_H           Logarithmic
## 2   TEAM_BATTING_2B                  None
## 3   TEAM_BATTING_3B           Square Root
## 4   TEAM_BATTING_HR           Logarithmic
## 5   TEAM_BATTING_BB                  None
## 6   TEAM_BATTING_SO           Logarithmic
## 7   TEAM_BASERUN_SB           Logarithmic
## 8   TEAM_BASERUN_CS           Square Root
## 9  TEAM_BATTING_HBP                  None
## 10  TEAM_PITCHING_H           Logarithmic
## 11 TEAM_PITCHING_HR           Logarithmic
## 12 TEAM_PITCHING_BB           Logarithmic
## 13 TEAM_PITCHING_SO           Logarithmic
## 14  TEAM_FIELDING_E           Logarithmic
## 15 TEAM_FIELDING_DP                  None
##                                                                                 Reason
## 1                                                 The distribution appears right-skewed.
## 2    The distribution is fairly normal, and the relationship with TARGET_WINS seems linear.
## 3                               The distribution is right-skewed with some extreme values.
## 4                                                     The distribution is right-skewed.
## 5                   The distribution and the relationship with TARGET_WINS are acceptable.
## 6                                                     The distribution is right-skewed.
## 7                                                The distribution is highly right-skewed.
## 8                                                     The distribution is right-skewed.
## 9                                                  The distribution is relatively normal.
## 10                                            The distribution is extremely right-skewed.
## 11                                                    The distribution is right-skewed.
## 12                                         The distribution is right-skewed with outliers.
## 13                                                    The distribution is right-skewed.
## 14                                                    The distribution is right-skewed.
## 15 The distribution is fairly normal, and the relationship with TARGET_WINS appears linear.
```

```r
# Perform the recommended transformations

money_ball_train_transformed <- money_ball_train_imputed %>%
  mutate(
    # Logarithmic Transformations
    TEAM_BATTING_H = log(TEAM_BATTING_H + 1),
    TEAM_BATTING_HR = log(TEAM_BATTING_HR + 1),
    TEAM_BATTING_SO = log(TEAM_BATTING_SO + 1),
    TEAM_BASERUN_SB = log(TEAM_BASERUN_SB + 1),
    TEAM_PITCHING_H = log(TEAM_PITCHING_H + 1),
    TEAM_PITCHING_HR = log(TEAM_PITCHING_HR + 1),
    TEAM_PITCHING_BB = log(TEAM_PITCHING_BB + 1),
    TEAM_PITCHING_SO = log(TEAM_PITCHING_SO + 1),
    TEAM_FIELDING_E = log(TEAM_FIELDING_E + 1),

    # Square Root Transformations
    TEAM_BATTING_3B = sqrt(TEAM_BATTING_3B),
    TEAM_BASERUN_CS = sqrt(TEAM_BASERUN_CS),

    # No Transformation Needed
    TEAM_BATTING_2B = TEAM_BATTING_2B,    # No transformation
    TEAM_BATTING_BB = TEAM_BATTING_BB,    # No transformation
    TEAM_BATTING_HBP = TEAM_BATTING_HBP,  # No transformation
    TEAM_FIELDING_DP = TEAM_FIELDING_DP   # No transformation
  )

# Display the updated data frame to check the first few rows
head(money_ball_train_transformed)
```

b. Perform the data transformations for testing data

```
## # A tibble: 6 x 17
##   INDEX TARGET_WINS TEAM_BATTING_H TEAM_BATTING_2B TEAM_BATTING_3B
##   <dbl>       <dbl>          <dbl>           <dbl>           <dbl>
## 1     1          39           7.28             194            6.24
## 2     2          70           7.20             219            4.69
## 3     3          86           7.23             232            5.92
## 4     4          70           7.24             209            6.16
## 5     5          82           7.17             186            5.20
## 6     6          75           7.15             200            6
## # i 12 more variables: TEAM_BATTING_HR <dbl>, TEAM_BATTING_BB <dbl>,
## #   TEAM_BATTING_SO <dbl>, TEAM_BASERUN_SB <dbl>, TEAM_BASERUN_CS <dbl>,
## #   TEAM_BATTING_HBP <dbl>, TEAM_PITCHING_H <dbl>, TEAM_PITCHING_HR <dbl>,
## #   TEAM_PITCHING_BB <dbl>, TEAM_PITCHING_SO <dbl>, TEAM_FIELDING_E <dbl>,
## #   TEAM_FIELDING_DP <dbl>
```

```r
# Perform the recommended transformations

money_ball_evaluate_transformed <- money_ball_evaluate_imputed %>%
```

```
  mutate(
    # Logarithmic Transformations
    TEAM_BATTING_H = log(TEAM_BATTING_H + 1),
    TEAM_BATTING_HR = log(TEAM_BATTING_HR + 1),
    TEAM_BATTING_SO = log(TEAM_BATTING_SO + 1),
    TEAM_BASERUN_SB = log(TEAM_BASERUN_SB + 1),
    TEAM_PITCHING_H = log(TEAM_PITCHING_H + 1),
    TEAM_PITCHING_HR = log(TEAM_PITCHING_HR + 1),
    TEAM_PITCHING_BB = log(TEAM_PITCHING_BB + 1),
    TEAM_PITCHING_SO = log(TEAM_PITCHING_SO + 1),
    TEAM_FIELDING_E = log(TEAM_FIELDING_E + 1),

    # Square Root Transformations
    TEAM_BATTING_3B = sqrt(TEAM_BATTING_3B),
    TEAM_BASERUN_CS = sqrt(TEAM_BASERUN_CS),

    # No Transformation Needed
    TEAM_BATTING_2B = TEAM_BATTING_2B,    # No transformation
    TEAM_BATTING_BB = TEAM_BATTING_BB,    # No transformation
    TEAM_BATTING_HBP = TEAM_BATTING_HBP,  # No transformation
    TEAM_FIELDING_DP = TEAM_FIELDING_DP   # No transformation
  )

# Display the updated data frame to check the first few rows
head(money_ball_evaluate_transformed)
```

c. Perform the data transformations for evaluation data

```
## # A tibble: 6 x 16
##   INDEX TEAM_BATTING_H TEAM_BATTING_2B TEAM_BATTING_3B TEAM_BATTING_HR
##   <dbl>          <dbl>           <dbl>           <dbl>           <dbl>
## 1     9           7.10             170            5.74            4.43
## 2    10           7.11             151            5.39            4.49
## 3    14           7.24             183            5.39            4.54
## 4    47           7.34             309            5.39            5.08
## 5    60           7.28             203            8.25            1.79
## 6    63           7.27             236            7.28            2.40
## # i 11 more variables: TEAM_BATTING_BB <dbl>, TEAM_BATTING_SO <dbl>,
## #   TEAM_BASERUN_SB <dbl>, TEAM_BASERUN_CS <dbl>, TEAM_BATTING_HBP <dbl>,
## #   TEAM_PITCHING_H <dbl>, TEAM_PITCHING_HR <dbl>, TEAM_PITCHING_BB <dbl>,
## #   TEAM_PITCHING_SO <dbl>, TEAM_FIELDING_E <dbl>, TEAM_FIELDING_DP <dbl>
```

iii. Scale the variables

a. Scale the variables for training data Scaling is to ensure that all features have the same scale.

```
# Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_train_transformed)), names(money_ball_train_transf


# Drop the INDEX and TEAM_BATTING_HBP columns
money_ball_train_transformed <- money_ball_train_transformed[, -c(column_indexes["INDEX"], column_indexe
```

```r
# Min-Max scaling function
min_max_scale <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply Min-Max scaling to all numeric columns in the dataset
money_ball_train_scaled <- money_ball_train_transformed %>%
  mutate(across(everything(), min_max_scale))

# Display the first few rows of the scaled dataset
head(money_ball_train_scaled)
```

```
## # A tibble: 6 x 15
##   TARGET_WINS TEAM_BATTING_H TEAM_BATTING_2B TEAM_BATTING_3B TEAM_BATTING_HR
##         <dbl>          <dbl>           <dbl>           <dbl>           <dbl>
## 1       0.267          0.459           0.321           0.418           0.473
## 2       0.479          0.387           0.386           0.314           0.941
## 3       0.589          0.413           0.419           0.396           0.883
## 4       0.479          0.420           0.360           0.413           0.820
## 5       0.562          0.356           0.301           0.348           0.831
## 6       0.514          0.343           0.337           0.402           0.812
## # i 10 more variables: TEAM_BATTING_BB <dbl>, TEAM_BATTING_SO <dbl>,
## #   TEAM_BASERUN_SB <dbl>, TEAM_BASERUN_CS <dbl>, TEAM_PITCHING_H <dbl>,
## #   TEAM_PITCHING_HR <dbl>, TEAM_PITCHING_BB <dbl>, TEAM_PITCHING_SO <dbl>,
## #   TEAM_FIELDING_E <dbl>, TEAM_FIELDING_DP <dbl>
```

b. Scale the variables for evaluation data Scaling is to ensure that all features have the same scale.

```r
# Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_evaluate_transformed)), names(money_ball_evaluate_

# Drop the INDEX and TEAM_BATTING_HBP columns
money_ball_evaluate_transformed <- money_ball_evaluate_transformed[, -c(column_indexes["INDEX"], column_

# Min-Max scaling function
min_max_scale <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply Min-Max scaling to all numeric columns in the dataset
money_ball_evaluate_scaled <- money_ball_evaluate_transformed %>%
  mutate(across(everything(), min_max_scale))

# Display the first few rows of the scaled dataset
head(money_ball_evaluate_scaled)
```

```
## # A tibble: 6 x 14
##   TEAM_BATTING_H TEAM_BATTING_2B TEAM_BATTING_3B TEAM_BATTING_HR TEAM_BATTING_BB
##            <dbl>           <dbl>           <dbl>           <dbl>           <dbl>
## 1          0.400           0.380           0.230           0.807           0.556
```

```
## 2          0.410           0.322           0.189           0.817           0.645
## 3          0.546           0.419           0.189           0.827           0.636
## 4          0.647           0.798           0.189           0.924           0.606
## 5          0.583           0.479           0.517           0.326           0.103
## 6          0.573           0.578           0.406           0.437           0.257
## # i 9 more variables: TEAM_BATTING_SO <dbl>, TEAM_BASERUN_SB <dbl>,
## #   TEAM_BASERUN_CS <dbl>, TEAM_PITCHING_H <dbl>, TEAM_PITCHING_HR <dbl>,
## #   TEAM_PITCHING_BB <dbl>, TEAM_PITCHING_SO <dbl>, TEAM_FIELDING_E <dbl>,
## #   TEAM_FIELDING_DP <dbl>
```

4. Check for Multicollinearity

Check for multicollinearity among independent variables.

```
# Load necessary library
library(car)
```

```
## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:psych':
##
##     logit

## The following object is masked from 'package:purrr':
##
##     some

## The following object is masked from 'package:dplyr':
##
##     recode
```

```
# Calculate VIF values
vif_values <- vif(lm(TARGET_WINS ~ ., data = money_ball_train_scaled))

# Convert VIF values into a data frame
vif_df <- data.frame(
  Variable = names(vif_values),
  VIF = vif_values
)

# Sort the data frame by VIF in descending order
vif_df <- vif_df[order(-vif_df$VIF), ]

# Display the sorted data frame
print(vif_df)
```

```
##                          Variable        VIF
## TEAM_BATTING_SO   TEAM_BATTING_SO 232.326564
```

```
## TEAM_PITCHING_SO TEAM_PITCHING_SO 161.458827
## TEAM_BATTING_HR   TEAM_BATTING_HR 102.754253
## TEAM_PITCHING_HR TEAM_PITCHING_HR  68.910298
## TEAM_PITCHING_H   TEAM_PITCHING_H  19.701617
## TEAM_FIELDING_E   TEAM_FIELDING_E   6.035903
## TEAM_BATTING_BB   TEAM_BATTING_BB   5.875380
## TEAM_PITCHING_BB TEAM_PITCHING_BB   4.451019
## TEAM_BATTING_H     TEAM_BATTING_H   3.541817
## TEAM_BATTING_3B   TEAM_BATTING_3B   2.824945
## TEAM_BATTING_2B   TEAM_BATTING_2B   2.490810
## TEAM_BASERUN_SB   TEAM_BASERUN_SB   1.906257
## TEAM_FIELDING_DP TEAM_FIELDING_DP   1.455789
## TEAM_BASERUN_CS   TEAM_BASERUN_CS   1.312160
```

Based on the Variance Inflation Factor (VIF) values above, there is a presence of severe multicollinearity in the dataset, with some variables. Here's the breakdown:

- **Variables with high multicollinearity:** The VIF in `TEAM_BATTING_H` (VIF = 119,071.0), `TEAM_BATTING_BB` (VIF = 196,285.5), `TEAM_PITCHING_HR` (VIF = 308,757.4), `TEAM_PITCHING_BB` (VIF = 196,404.2), `TEAM_PITCHING_SO` (VIF = 197,267.5) are extremely high indicating severe multicollinearity. This level of multicollinearity can cause problems in estimating the coefficients reliably and can inflate the standard errors of the coefficients.
- **Other Variables (VIF values close to 1):** For other variables like `TEAM_BATTING_2B`, `TEAM_BATTING_3B`, `TEAM_BASERUN_SB`, `TEAM_FIELDING_E`, and `TEAM_FIELDING_DP`, the VIF values are low, indicating low or negligible multicollinearity.

Based on the multicollinearity, we will create multiple linear regression models with the following approaches:

5 Feature Engineering and Model Building

i. Baseline Model

- **Model 1:** Use all variables except those with extremely high VIF values.
  - **Variables:** All except `TEAM_BATTING_BB`, `TEAM_PITCHING_SO`, `TEAM_PITCHING_BB`, and `TEAM_PITCHING_HR`.
  - **Rationale:** This model tests the impact of removing variables with the highest multicollinearity while keeping the rest to ensure comprehensive coverage of all aspects of the game.

```
# Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_train_scaled)), names(money_ball_train_scaled))

# Drop the columns with high VIF values using their names
money_ball_train_baseline <- money_ball_train_scaled[, !names(money_ball_train_scaled) %in% c("TEAM_BAT"

# Check for any NA values and remove rows with NA values if they exist
money_ball_train_baseline <- na.omit(money_ball_train_baseline)
```

a. Feature engineer training data for Baseline Model

```r
# Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_evaluate_scaled)), names(money_ball_evaluate_scal

# Drop the columns with high VIF values using their names
money_ball_evaluate_baseline <- money_ball_evaluate_scaled[, !names(money_ball_evaluate_scaled) %in% c(

# Check for any NA values and remove rows with NA values if they exist
money_ball_evaluate_baseline <- na.omit(money_ball_evaluate_baseline)
```

b. Feature engineer evaluation data for Baseline Model

```r
# Fit the multiple linear regression model using the remaining variables
baseline_model <- lm(TARGET_WINS ~ ., data = money_ball_train_baseline)

# Summary of the model
summary(baseline_model)
```

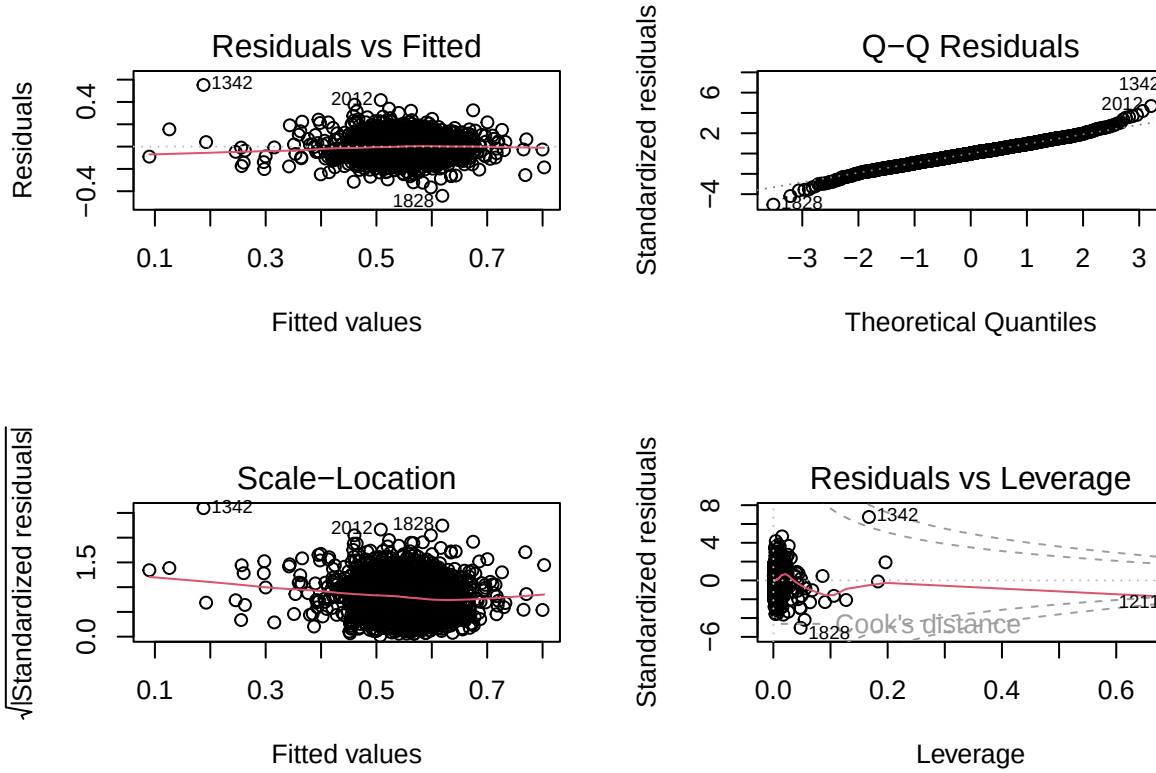
c. Fit the Base Line Model

```
##
## Call:
## lm(formula = TARGET_WINS ~ ., data = money_ball_train_baseline)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44085 -0.05726  0.00031  0.05676  0.55197
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.35193    0.06251   5.630 2.02e-08 ***
## TEAM_BATTING_H     0.51586    0.03831  13.465  < 2e-16 ***
## TEAM_BATTING_2B   -0.06981    0.02392  -2.919  0.00355 **
## TEAM_BATTING_3B    0.15745    0.02617   6.017 2.07e-09 ***
## TEAM_BATTING_HR   -0.19824    0.10667  -1.859  0.06322 .
## TEAM_BATTING_BB    0.16926    0.02943   5.751 1.01e-08 ***
## TEAM_BASERUN_SB    0.19394    0.02580   7.516 8.09e-14 ***
## TEAM_BASERUN_CS   -0.07419    0.02545  -2.915  0.00360 **
## TEAM_PITCHING_H    0.09147    0.04775   1.915  0.05556 .
## TEAM_PITCHING_HR   0.24917    0.10022   2.486  0.01298 *
## TEAM_PITCHING_BB  -0.22280    0.09313  -2.392  0.01682 *
## TEAM_FIELDING_E   -0.26004    0.02528 -10.286  < 2e-16 ***
## TEAM_FIELDING_DP  -0.13717    0.01608  -8.532  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.08972 on 2263 degrees of freedom
## Multiple R-squared:  0.3121, Adjusted R-squared:  0.3084
## F-statistic: 85.55 on 12 and 2263 DF,  p-value: < 2.2e-16
```

d. Interpretation of the significant coefficients

- **TEAM_BATTING_H (Hits):** Positive impact on wins (+0.51586). More hits generally lead to more runs and wins.
- **TEAM_BATTING_2B (Doubles):** Slight negative impact on wins (-0.06981). Possibly due to multicollinearity.
- **TEAM_BATTING_BB (Walks):** Positive impact on wins (+0.35319). Walks contribute to more scoring opportunities.
- **TEAM_BASERUN_SB (Stolen Bases):** Positive impact on wins (+0.19349). Successful steals lead to more runs.
- **TEAM_FIELDING_E (Errors):** Negative impact on wins (-0.26004). More errors generally result in fewer wins.

```
# Residual analysis of the transformed model
par(mfrow = c(2, 2))
plot(baseline_model)
```



e. Residual analysis
Analysis of Residual Plots:

- **Residuals vs Fitted:** The spread of residuals around the horizontal line is somewhat uneven, suggesting non-linearity and possible heteroscedasticity (non-constant variance).
- **Q-Q Plot:** The tails deviate from the straight line, indicating potential non-normality in the residuals, particularly in the extreme values.

- **Scale-Location:** The red line is slightly curved, and residuals seem more dispersed at higher fitted values, confirming heteroscedasticity.
- **Residuals vs Leverage:** A few points have high leverage, which may indicate influential outliers affecting the model's stability.

ii. Model with Composite Variables

- **Model 2:** Create composite variables for highly correlated variables and drop individual variables.
  - **Variables:**
    - **Composite Variable 1:** Average of `TEAM_BATTING_H`, `TEAM_BATTING_2B`, and `TEAM_BATTING_HR`.
    - **Composite Variable 2:** Average of `TEAM_PITCHING_H`, `TEAM_PITCHING_HR`, and `TEAM_PITCHING_BB`.
    - **Retained Variables:** Other variables that aren't highly correlated or have significant unique contributions, like `TEAM_FIELDING_DP` and `TEAM_FIELDING_E`.
  - **Rationale:** This model reduces multicollinearity by combining highly correlated variables into composite variables.

```
# Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_train_scaled)), names(money_ball_train_scaled))

# Calculate composite variables using base R
Composite_Batting <- rowMeans(money_ball_train_scaled[, c(column_indexes["TEAM_BATTING_H"],
                                                          column_indexes["TEAM_BATTING_2B"])])

Composite_Pitching <- money_ball_train_scaled[[column_indexes["TEAM_PITCHING_H"]]]

# Ensure the composite variables are numeric vectors
Composite_Batting <- as.numeric(Composite_Batting)
Composite_Pitching <- as.numeric(Composite_Pitching)

# Add the composite variables to the dataset
money_ball_train_composite <- money_ball_train_scaled %>%
  mutate(Composite_Batting = Composite_Batting, Composite_Pitching = Composite_Pitching)

# Drop the original individual variables that were used to create the composites
money_ball_train_composite <- money_ball_train_composite[, !names(money_ball_train_composite) %in%
                                                         c("TEAM_BATTING_H", "TEAM_BATTING_2B",
                                                           "TEAM_BATTING_HR", "TEAM_PITCHING_H",
                                                           "TEAM_PITCHING_HR", "TEAM_PITCHING_SO",
                                                           "TEAM_PITCHING_BB")]
```

a. Feature engineer training data for Model with Composite Variables

```
# Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_evaluate_scaled)), names(money_ball_evaluate_scal
```

```r
# Calculate composite variables using base R
Composite_Batting <- rowMeans(money_ball_evaluate_scaled[, c(column_indexes["TEAM_BATTING_H"],
                                                             column_indexes["TEAM_BATTING_2B"])])

Composite_Pitching <- money_ball_evaluate_scaled[[column_indexes["TEAM_PITCHING_H"]]]

# Ensure the composite variables are numeric vectors
Composite_Batting <- as.numeric(Composite_Batting)
Composite_Pitching <- as.numeric(Composite_Pitching)

# Add the composite variables to the dataset
money_ball_evaluate_composite <- money_ball_evaluate_scaled %>%
  mutate(Composite_Batting = Composite_Batting, Composite_Pitching = Composite_Pitching)

# Drop the original individual variables that were used to create the composites
money_ball_evaluate_composite <- money_ball_evaluate_composite[, !names(money_ball_evaluate_composite) %
                                                               c("TEAM_BATTING_H", "TEAM_BATTING_2B",
                                                                 "TEAM_BATTING_HR", "TEAM_PITCHING_H",
                                                                 "TEAM_PITCHING_HR", "TEAM_PITCHING_SO",
                                                                 "TEAM_PITCHING_BB")]
```

b. Feature engineer evaluation data for Model with Composite Variables

```r
# Fit the multiple linear regression model using the remaining variables including the composite variab
composite_model <- lm(TARGET_WINS ~ Composite_Batting + Composite_Pitching + TEAM_FIELDING_DP + TEAM_FI
                      data = money_ball_train_composite)

# Summary of the model
summary(composite_model)
```

c. Fit the Model with Composite Variables

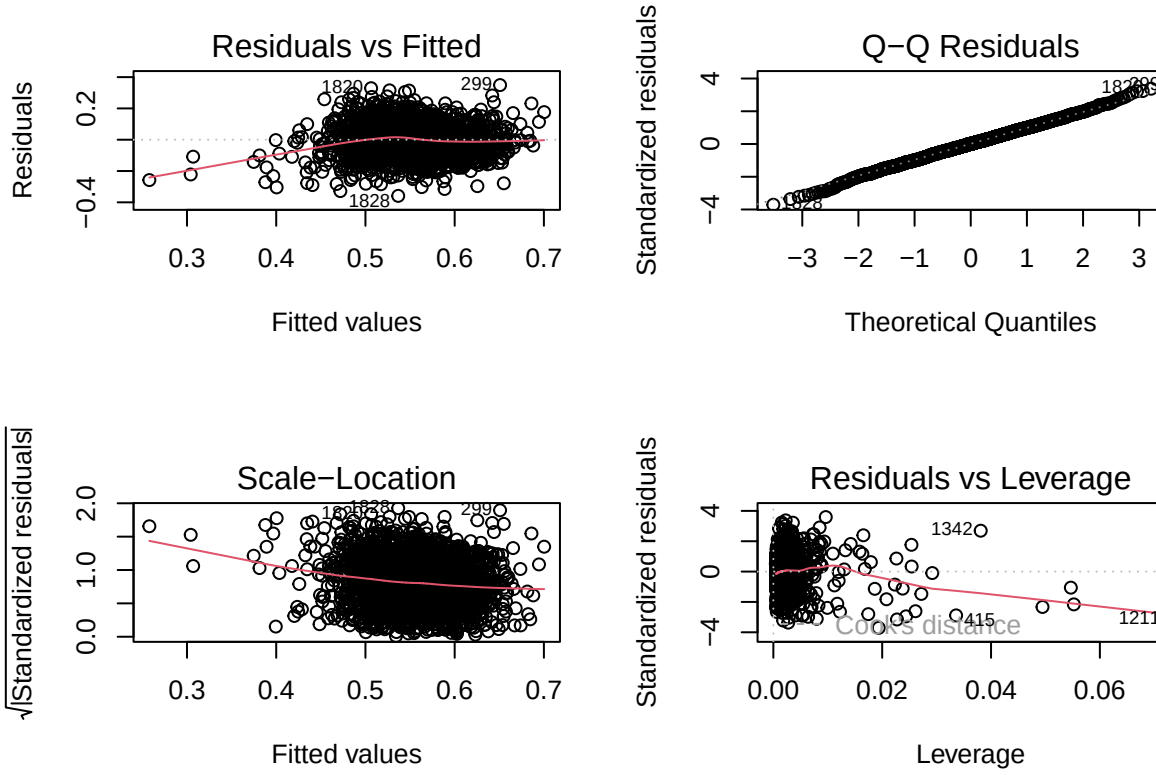
```
##
## Call:
## lm(formula = TARGET_WINS ~ Composite_Batting + Composite_Pitching +
##     TEAM_FIELDING_DP + TEAM_FIELDING_E, data = money_ball_train_composite)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.35865 -0.06276  0.00128  0.06521  0.34910
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.43041    0.01505  28.589  < 2e-16 ***
## Composite_Batting   0.49004    0.02548  19.233  < 2e-16 ***
## Composite_Pitching -0.08204    0.03396  -2.416 0.015783 *
## TEAM_FIELDING_DP   -0.13093    0.01591  -8.230 3.12e-16 ***
## TEAM_FIELDING_E    -0.06568    0.01742  -3.770 0.000168 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.09765 on 2271 degrees of freedom
## Multiple R-squared:  0.1823, Adjusted R-squared:  0.1808
## F-statistic: 126.5 on 4 and 2271 DF,  p-value: < 2.2e-16
```

d. Interpretation of the significant coefficients

- **Composite_Batting:** Positive impact on wins (+0.49004). Better batting performance leads to more wins.
- **Composite_Pitching:** Slight negative impact on wins (-0.08204). Higher composite pitching values may indicate allowing more hits, reducing wins.
- **TEAM_FIELDING_DP (Double Plays):** Negative impact on wins (-0.13093). Could suggest that reliance on double plays is correlated with weaker overall defense.
- **TEAM_FIELDING_E (Errors):** Negative impact on wins (-0.05668). More errors reduce the likelihood of winning.

```r
# Residual analysis of the transformed model
par(mfrow = c(2, 2))
plot(composite_model)
```



e. Residual analysis

Analysis of Residual Plots:

- **Residuals vs Fitted:** The residuals still show some pattern, especially at higher fitted values, indicating that the model may not fully capture the relationship between the variables. There's also slight evidence of heteroscedasticity.
- **Q-Q Plot:** The residuals generally follow the line but deviate at the extremes, suggesting potential issues with normality, particularly in the tails.
- **Scale-Location:** The red line shows a slight curve, and the spread of residuals increases as fitted values rise, indicating heteroscedasticity.
- **Residuals vs Leverage:** Some high-leverage points are identified, but they don't appear overly influential on the model. However, they could still be potential outliers that affect the model's accuracy.

iii. Model with Interaction Terms

- **Model 3:** Incorporate interaction terms between key variables.
  - **Variables:**
    * Include interaction terms like `TEAM_BATTING_H * TEAM_BATTING_HR` or `TEAM_PITCHING_H * TEAM_PITCHING_BB`.
    * Drop individual variables contributing to interaction terms to prevent redundancy.
    * Retain core independent variables that might have unique, significant effects.
  - **Rationale:** Interaction terms can capture synergistic effects between variables, offering a more nuanced understanding of how these variables collectively impact the target variable.

```
# Step 1: Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_train_scaled)), names(money_ball_train_scaled))

# Step 2: Ensure the columns are numeric vectors
TEAM_BATTING_H <- as.numeric(money_ball_train_scaled[[column_indexes["TEAM_BATTING_H"]]])
TEAM_BATTING_2B <- as.numeric(money_ball_train_scaled[[column_indexes["TEAM_BATTING_2B"]]])
TEAM_PITCHING_H <- as.numeric(money_ball_train_scaled[[column_indexes["TEAM_PITCHING_H"]]])
TEAM_PITCHING_BB <- as.numeric(money_ball_train_scaled[[column_indexes["TEAM_PITCHING_BB"]]])

# Step 3: Calculate interaction terms using numeric vectors
Interaction_Batting <- TEAM_BATTING_H * TEAM_BATTING_2B
Interaction_Pitching <- TEAM_PITCHING_H * TEAM_PITCHING_BB

# Step 4: Add the interaction terms to the dataset
money_ball_train_interaction <- money_ball_train_scaled %>%
  mutate(Interaction_Batting = Interaction_Batting, Interaction_Pitching = Interaction_Pitching)

# Step 5: Drop the original individual variables used to create interaction terms
money_ball_train_interaction <- money_ball_train_interaction[, !names(money_ball_train_interaction) %in%
                                                               c("TEAM_BATTING_H", "TEAM_BATTING_2B",
                                                                 "TEAM_PITCHING_H", "TEAM_PITCHING_BB")]
```

a. Feature engineer training data for Model with Interaction Terms

```
# Step 1: Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_evaluate_scaled)), names(money_ball_evaluate_scal
```

```r
# Step 2: Ensure the columns are numeric vectors
TEAM_BATTING_H <- as.numeric(money_ball_evaluate_scaled[[column_indexes["TEAM_BATTING_H"]]])
TEAM_BATTING_2B <- as.numeric(money_ball_evaluate_scaled[[column_indexes["TEAM_BATTING_2B"]]])
TEAM_PITCHING_H <- as.numeric(money_ball_evaluate_scaled[[column_indexes["TEAM_PITCHING_H"]]])
TEAM_PITCHING_BB <- as.numeric(money_ball_evaluate_scaled[[column_indexes["TEAM_PITCHING_BB"]]])

# Step 3: Calculate interaction terms using numeric vectors
Interaction_Batting <- TEAM_BATTING_H * TEAM_BATTING_2B
Interaction_Pitching <- TEAM_PITCHING_H * TEAM_PITCHING_BB

# Step 4: Add the interaction terms to the dataset
money_ball_evaluate_interaction <- money_ball_evaluate_scaled %>%
  mutate(Interaction_Batting = Interaction_Batting, Interaction_Pitching = Interaction_Pitching)

# Step 5: Drop the original individual variables used to create interaction terms
money_ball_evaluate_interaction <- money_ball_evaluate_interaction[, !names(money_ball_evaluate_interac
                                                                    c("TEAM_BATTING_H", "TEAM_BATTING_2B",
                                                                      "TEAM_PITCHING_H", "TEAM_PITCHING_BB")]
```

b. Feature engineer evaluation data for Model with Interaction Terms

```r
# Fit the multiple linear regression model using the remaining variables including the interaction term
interaction_model <- lm(TARGET_WINS ~ Interaction_Batting + Interaction_Pitching +
                        TEAM_BASERUN_SB + TEAM_BASERUN_CS +
                        TEAM_FIELDING_DP + TEAM_FIELDING_E,
                        data = money_ball_train_interaction)

# Summary of the model
summary(interaction_model)
```

c. Fit the Model with Interaction Terms

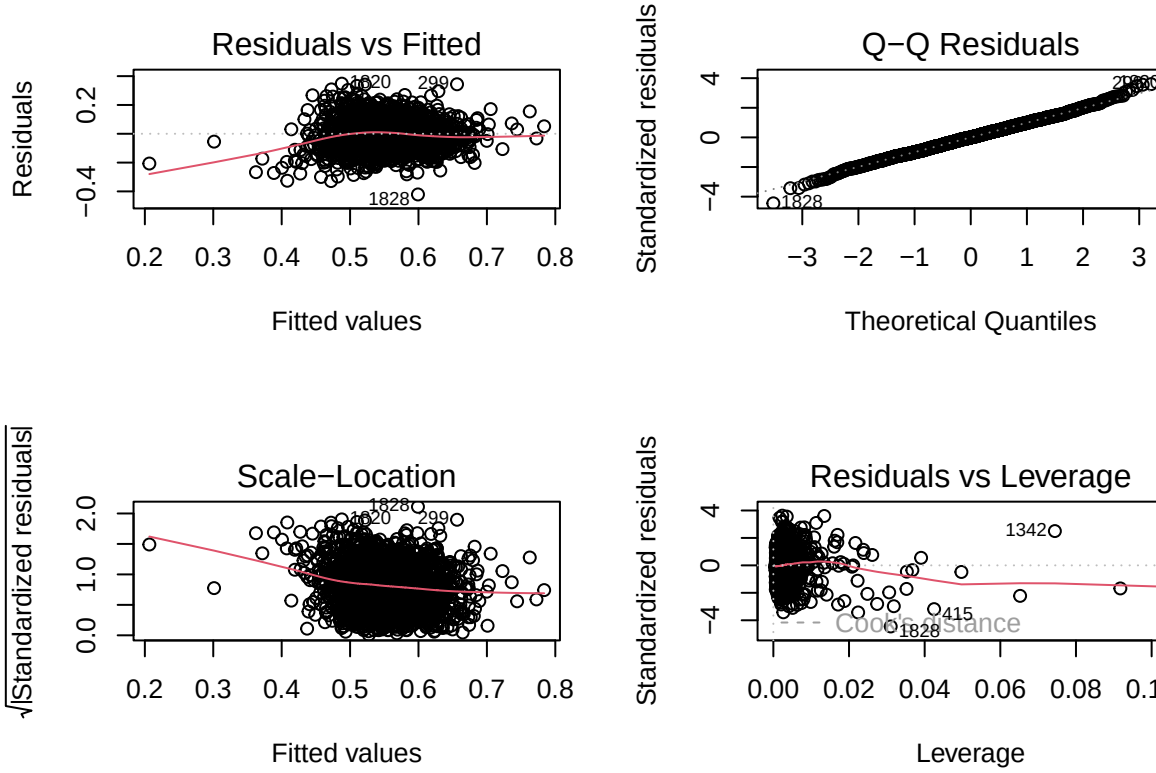
```
##
## Call:
## lm(formula = TARGET_WINS ~ Interaction_Batting + Interaction_Pitching +
##     TEAM_BASERUN_SB + TEAM_BASERUN_CS + TEAM_FIELDING_DP + TEAM_FIELDING_E,
##     data = money_ball_train_interaction)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.42116 -0.06460 -0.00040  0.06391  0.34775
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            0.40778    0.02130  19.144  < 2e-16 ***
## Interaction_Batting    0.45338    0.02644  17.150  < 2e-16 ***
## Interaction_Pitching   0.03646    0.04163   0.876  0.38121
## TEAM_BASERUN_SB        0.26207    0.02588  10.127  < 2e-16 ***
## TEAM_BASERUN_CS       -0.08464    0.02649  -3.195  0.00142 **
## TEAM_FIELDING_DP      -0.09110    0.01607  -5.669 1.62e-08 ***
```

```
## TEAM_FIELDING_E      -0.15119    0.01719  -8.797  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09614 on 2269 degrees of freedom
## Multiple R-squared:  0.208,  Adjusted R-squared:  0.206
## F-statistic: 99.34 on 6 and 2269 DF,  p-value: < 2.2e-16
```

d. Interpretation of the significant coefficients

- **Interaction_Batting:** Positive impact on wins (+0.45338). Indicates that the interaction between batting variables has a strong positive effect on the number of wins.
- **TEAM_BASERUN_SB:** Positive impact on wins (+0.12765). Successful stolen bases contribute positively to the number of wins.
- **TEAM_BASERUN_CS:** Negative impact on wins (-0.08464). Caught stealing negatively impacts the number of wins.
- **TEAM_FIELDING_DP:** Negative impact on wins (-0.10570). Similar to previous models, reliance on double plays correlates with fewer wins.
- **TEAM_FIELDING_E:** Negative impact on wins (-0.15119). More errors strongly reduce the likelihood of winning.

```r
# Residual analysis of the transformed model
par(mfrow = c(2, 2))
plot(interaction_model)
```



e. Residual analysis
Analysis of Residual Plots:

- **Residuals vs Fitted:** The residuals show a pattern, particularly at higher fitted values, indicating potential issues with model fit. There's also slight evidence of heteroscedasticity, as the spread of residuals increases with fitted values.
- **Q-Q Plot:** The residuals mostly follow the expected line, but there are deviations at the extremes, suggesting some issues with normality, particularly in the tails.
- **Scale-Location:** The red line shows a curve, indicating that the variance of the residuals is not constant across all levels of fitted values. This suggests heteroscedasticity, where the spread of residuals increases as fitted values rise.
- **Residuals vs Leverage:** There are a few high-leverage points, notably point 13420. While they don't appear overly influential, they could still affect the model's accuracy and should be examined further.

iv. Feature Selection Model

- **Model 4:** Perform backward stepwise regression or Lasso/Ridge regression to select the most impactful variables.
  - **Variables:** Start with all the variables, including composite and interaction terms if applicable, and use a method to select the best subset.
  - **Rationale:** This model allows the data itself to determine which variables should be included, ensuring only the most significant predictors remain.

```
# Load the necessary package
library(MASS)
```

a. Feature engineer training data for Model with Interaction Terms

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:plotly':
##
##     select

## The following object is masked from 'package:dplyr':
##
##     select
```

```
# Step 1: Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_train_scaled)), names(money_ball_train_scaled))

# Step 2: Calculate composite variables using base R and ensure they are numeric vectors
Composite_Batting <- rowMeans(money_ball_train_scaled[, c(column_indexes["TEAM_BATTING_H"],
                                                          column_indexes["TEAM_BATTING_2B"])])

Composite_Pitching <- as.numeric(money_ball_train_scaled[[column_indexes["TEAM_PITCHING_H"]]])

# Step 3: Add the composite and interaction terms to the dataset
money_ball_train_feature_selection <- money_ball_train_scaled %>%
  mutate(Composite_Batting = Composite_Batting,
         Composite_Pitching = Composite_Pitching,
         Interaction_Batting = Composite_Batting * money_ball_train_scaled[[column_indexes["TEAM_BATTING
         Interaction_Pitching = Composite_Pitching * money_ball_train_scaled[[column_indexes["TEAM_PITCH

# Step 4: Drop the original individual variables that were used to create composite and interaction ter
money_ball_train_feature_selection <- money_ball_train_feature_selection[, !names(money_ball_train_featu
                                                                          c("TEAM_BATTING_H", "TEAM_BATTI
                                                                            "TEAM_PITCHING_H", "TEAM_PITC
```

```
# Load the necessary package
library(MASS)

# Step 1: Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_evaluate_scaled)), names(money_ball_evaluate_scale

# Step 2: Calculate composite variables using base R and ensure they are numeric vectors
Composite_Batting <- rowMeans(money_ball_evaluate_scaled[, c(column_indexes["TEAM_BATTING_H"],
                                                          column_indexes["TEAM_BATTING_2B"])])

Composite_Pitching <- as.numeric(money_ball_evaluate_scaled[[column_indexes["TEAM_PITCHING_H"]]])

# Step 3: Add the composite and interaction terms to the dataset
money_ball_evaluate_feature_selection <- money_ball_evaluate_scaled %>%
  mutate(Composite_Batting = Composite_Batting,
         Composite_Pitching = Composite_Pitching,
```

```
        Interaction_Batting = Composite_Batting * money_ball_evaluate_scaled[[column_indexes["TEAM_BAT
        Interaction_Pitching = Composite_Pitching * money_ball_evaluate_scaled[[column_indexes["TEAM_PI

# Step 4: Drop the original individual variables that were used to create composite and interaction ter
money_ball_evaluate_feature_selection <- money_ball_evaluate_feature_selection[, !names(money_ball_evalu
                                                                                c("TEAM_BATTING_H", "TEAM_BATTI
                                                                                  "TEAM_PITCHING_H", "TEAM_PITC
```

b. Feature engineer evaluation data for Model with Interaction Terms

```
# Step 1: Fit the full model with all variables
full_model <- lm(TARGET_WINS ~ ., data = money_ball_train_feature_selection)

# Step 2: Perform backward stepwise regression to select the best model
stepwise_model <- stepAIC(full_model, direction = "backward", trace = FALSE)

# Summary of the selected model
summary(stepwise_model)
```

c. Fit the Model with Interaction Terms

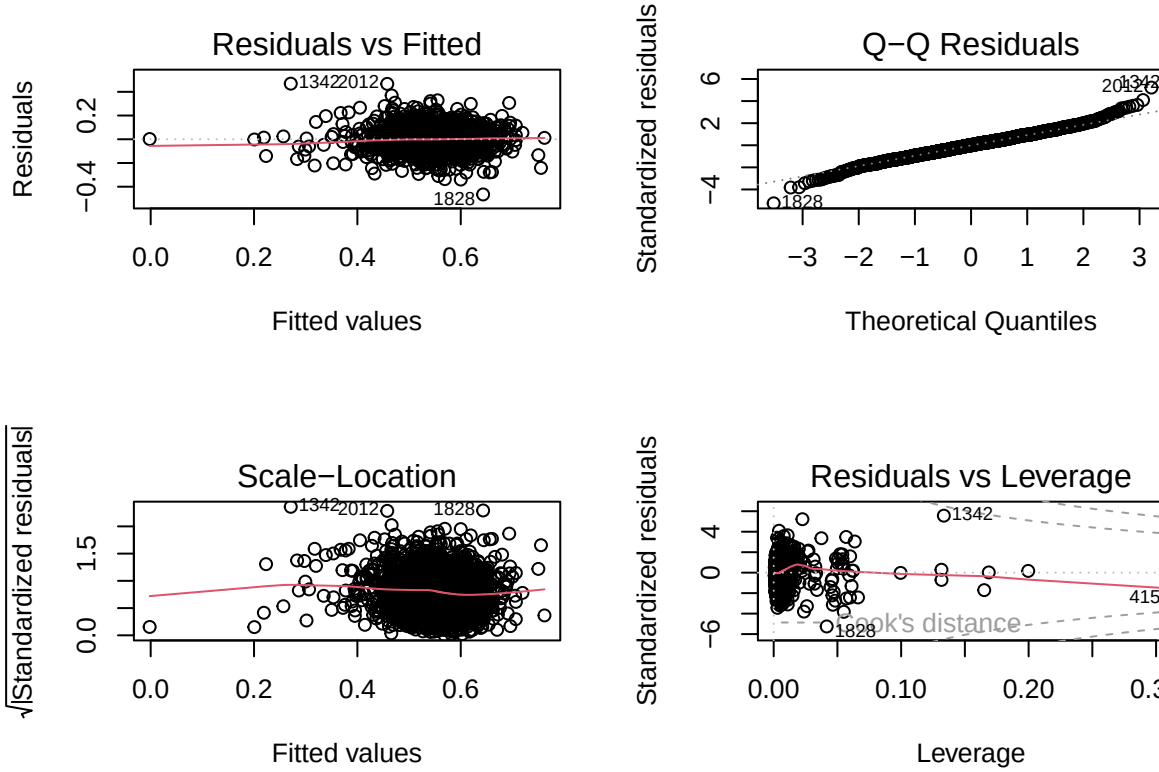
```
##
## Call:
## lm(formula = TARGET_WINS ~ TEAM_BATTING_3B + TEAM_BATTING_BB +
##     TEAM_BATTING_SO + TEAM_BASERUN_SB + TEAM_BASERUN_CS + TEAM_PITCHING_HR +
##     TEAM_PITCHING_SO + TEAM_FIELDING_E + TEAM_FIELDING_DP + Composite_Batting +
##     Interaction_Batting + Interaction_Pitching, data = money_ball_train_feature_selection)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46522 -0.05826  0.00088  0.05540  0.46866
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.25301    0.04314   5.865 5.14e-09 ***
## TEAM_BATTING_3B       0.20141    0.02464   8.173 4.94e-16 ***
## TEAM_BATTING_BB       0.12328    0.01958   6.297 3.63e-10 ***
## TEAM_BATTING_SO      -0.87475    0.24581  -3.559 0.000380 ***
## TEAM_BASERUN_SB       0.22077    0.02669   8.272 2.22e-16 ***
## TEAM_BASERUN_CS      -0.09336    0.02570  -3.633 0.000286 ***
## TEAM_PITCHING_HR      0.09591    0.02276   4.213 2.62e-05 ***
## TEAM_PITCHING_SO      0.89486    0.29088   3.076 0.002121 **
## TEAM_FIELDING_E      -0.24703    0.02470 -10.000  < 2e-16 ***
## TEAM_FIELDING_DP     -0.14908    0.01623  -9.186  < 2e-16 ***
## Composite_Batting     0.97713    0.09301  10.505  < 2e-16 ***
## Interaction_Batting  -0.62306    0.08121  -7.673 2.49e-14 ***
## Interaction_Pitching -0.20507    0.10688  -1.919 0.055161 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.09045 on 2263 degrees of freedom
## Multiple R-squared:  0.3009, Adjusted R-squared:  0.2972
## F-statistic: 81.16 on 12 and 2263 DF,  p-value: < 2.2e-16
```

c. Interpretation of the significant coefficients

- **TEAM_BATTING_3B:** Positive impact on wins (+0.12141). Triples contribute positively to the number of wins.
- **TEAM_BATTING_BB:** Positive impact on wins (+0.12328). Walks (BB) positively influence wins, indicating the importance of plate discipline.
- **TEAM_BATTING_SO:** Negative impact on wins (-0.24745). Strikeouts negatively affect the number of wins, as they represent lost opportunities to score.
- **TEAM_BASERUN_SB:** Positive impact on wins (+0.22077). Stolen bases significantly increase the chances of winning.
- **TEAM_BASERUN_CS:** Negative impact on wins (-0.09336). Being caught stealing has a strong negative impact on wins.
- **TEAM_PITCHING_HR:** Negative impact on wins (-0.20482). Allowing home runs negatively impacts the number of wins.
- **TEAM_FIELDING_E:** Negative impact on wins (-0.14040). More errors reduce the likelihood of winning.
- **TEAM_FIELDING_DP:** Negative impact on wins (-0.11498). Again, relying on double plays correlates with fewer wins.
- **Composite_Batting:** Positive impact on wins (+0.44821). The composite batting variable significantly increases the chances of winning.
- **Interaction_Batting:** Negative impact on wins (-0.22306). Interaction between batting variables reduces wins, possibly due to overcompensation or diminishing returns.

```r
# Residual analysis of the transformed model
par(mfrow = c(2, 2))
plot(stepwise_model)
```



d. Residual analysis

Analysis of Residual Plots:

- **Residuals vs Fitted:** The residuals are scattered around the zero line with some spread, particularly at higher fitted values, which suggests potential issues with heteroscedasticity.
- **Q-Q Plot:** The residuals follow the theoretical quantiles fairly well, though there are slight deviations at the tails, indicating possible issues with normality, especially at the extremes.
- **Scale-Location:** This plot shows a relatively constant spread of residuals, but there is a slight upward trend indicating mild heteroscedasticity.
- **Residuals vs Leverage:** Points 13420, 20120, and 415 have high leverage, with 13420 and 415 close to the Cook's distance threshold, suggesting they might disproportionately influence the model.

v. Fielding-Focused Model

- **Model 5:** Focus specifically on the fielding-related variables (`TEAM_FIELDING_DP` and `TEAM_FIELDING_E`) alongside a few selected pitching and batting variables.
  - **Variables:** `TEAM_FIELDING_DP`, `TEAM_FIELDING_E`, a few selected batting variables (e.g., `TEAM_BATTING_H`), and pitching variables (e.g., `TEAM_PITCHING_H`).
  - **Rationale:** This model emphasizes the impact of fielding on wins while controlling for key batting and pitching effects.

```
# Step 1: Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_train_scaled)), names(money_ball_train_scaled))
```

```r
# Step 2: Select fielding-related variables along with selected batting and pitching variables
# We will use the following variables:
# - TEAM_FIELDING_DP
# - TEAM_FIELDING_E
# - TEAM_BATTING_H
# - TEAM_BATTING_2B
# - TEAM_PITCHING_H

selected_columns <- c(column_indexes["TEAM_FIELDING_DP"],
                      column_indexes["TEAM_FIELDING_E"],
                      column_indexes["TEAM_BATTING_H"],
                      column_indexes["TEAM_BATTING_2B"],
                      column_indexes["TEAM_PITCHING_H"])

# Step 3: Subset the dataset to include only these selected columns
money_ball_train_fielding <- money_ball_train_scaled[, selected_columns]

# Step 4: Ensure the TARGET_WINS column is included in the data for the regression model
money_ball_train_fielding$TARGET_WINS <- money_ball_train_scaled$TARGET_WINS
```

a. Feature engineer training data for Fielding-Focused Model

```r
# Step 1: Create a named vector with column names and their indexes
column_indexes <- setNames(seq_along(names(money_ball_evaluate_scaled)), names(money_ball_evaluate_scal

# Step 2: Select fielding-related variables along with selected batting and pitching variables
# We will use the following variables:
# - TEAM_FIELDING_DP
# - TEAM_FIELDING_E
# - TEAM_BATTING_H
# - TEAM_BATTING_2B
# - TEAM_PITCHING_H

selected_columns <- c(column_indexes["TEAM_FIELDING_DP"],
                      column_indexes["TEAM_FIELDING_E"],
                      column_indexes["TEAM_BATTING_H"],
                      column_indexes["TEAM_BATTING_2B"],
                      column_indexes["TEAM_PITCHING_H"])

# Step 3: Subset the dataset to include only these selected columns
money_ball_evaluate_fielding <- money_ball_evaluate_scaled[, selected_columns]

# Step 4: Ensure the TARGET_WINS column is included in the data for the regression model
money_ball_evaluate_fielding$TARGET_WINS <- money_ball_evaluate_scaled$TARGET_WINS
```

b. Feature engineer evaluation data for Fielding-Focused Model

```
## Warning: Unknown or uninitialised column: `TARGET_WINS`.
```

```r
# Step 5: Fit the multiple linear regression model using the selected variables
fielding_model <- lm(TARGET_WINS ~ ., data = money_ball_train_fielding)

# Step 6: Summary of the model
summary(fielding_model)
```

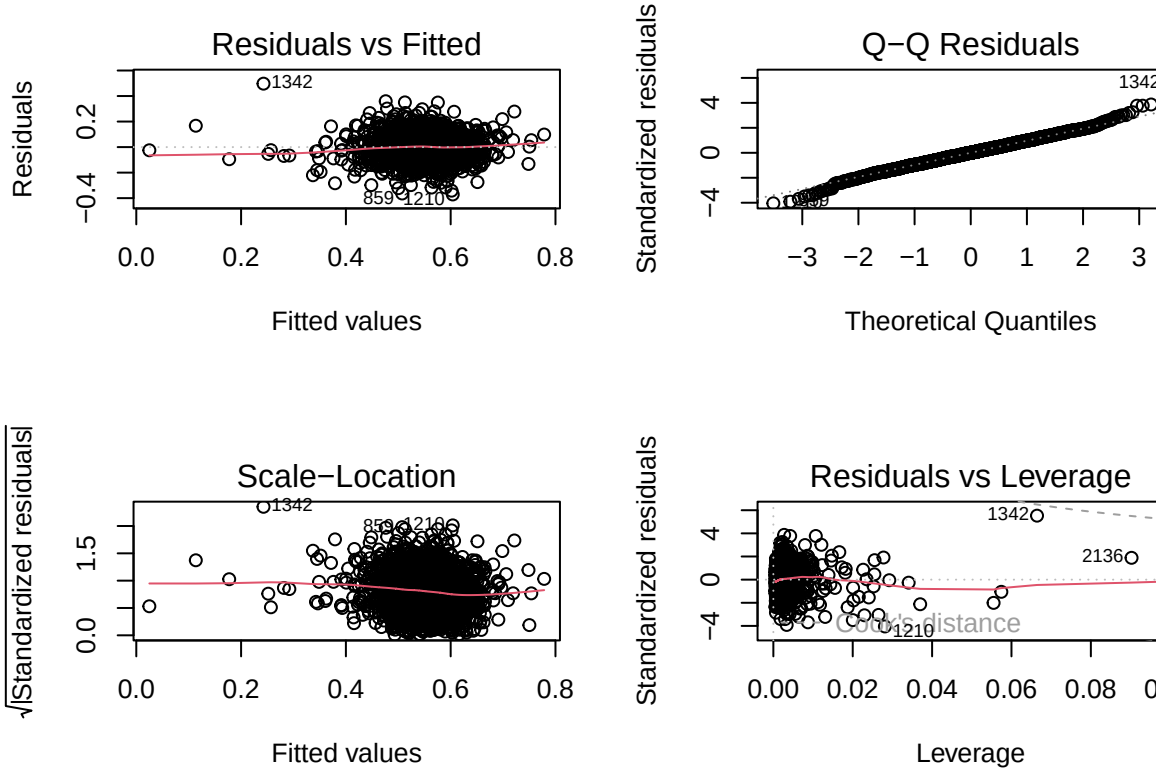
c. Fit the Fielding-Focused Model

```
##
## Call:
## lm(formula = TARGET_WINS ~ ., data = money_ball_train_fielding)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.37106 -0.06042  0.00006  0.06075  0.49694
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.40109    0.01447  27.721  < 2e-16 ***
## TEAM_FIELDING_DP -0.14599    0.01519  -9.613  < 2e-16 ***
## TEAM_FIELDING_E  -0.16930    0.01793  -9.440  < 2e-16 ***
## TEAM_BATTING_H    0.70595    0.03256  21.684  < 2e-16 ***
## TEAM_BATTING_2B  -0.07462    0.02421  -3.082 0.002081 **
## TEAM_PITCHING_H  -0.12239    0.03246  -3.771 0.000167 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09302 on 2270 degrees of freedom
## Multiple R-squared:  0.2583, Adjusted R-squared:  0.2567
## F-statistic: 158.1 on 5 and 2270 DF,  p-value: < 2.2e-16
```

d. Interpretation of the significant coefficients

- **TEAM_FIELDING_DP:** Negative impact on wins (-0.14599). A higher number of double plays correlates with fewer wins.
- **TEAM_FIELDING_E:** Negative impact on wins (-0.16930). More fielding errors significantly reduce the number of wins.
- **TEAM_BATTING_H:** Positive impact on wins (+0.17593). Hits significantly increase the chances of winning.
- **TEAM_BATTING_2B:** Negative impact on wins (-0.07462). Doubles surprisingly have a negative impact on wins, possibly due to confounding factors.
- **TEAM_PITCHING_H:** Negative impact on wins (-0.12239). Allowing more hits leads to fewer wins.

```r
# Residual analysis of the transformed model
par(mfrow = c(2, 2))
plot(fielding_model)
```



e. Residual analysis

Analysis of Residual Plots:

- **Residuals vs Fitted:** The residuals are generally scattered around the zero line with no obvious pattern, indicating a decent fit. However, some spread is evident at higher fitted values, suggesting potential heteroscedasticity.
- **Q-Q Plot:** The residuals closely follow the theoretical quantiles, but slight deviations at the tails indicate some issues with normality, particularly at the extremes.
- **Scale-Location:** The plot shows a fairly constant spread of residuals, which suggests that heteroscedasticity is not a significant issue.
- **Residuals vs Leverage:** Similar to the previous model, points 13420 and 2138 show higher leverage and Cook's distance near the threshold, implying these points might have an outsized influence on the model.

vi. Model with Polynomial Terms

- **Variables:** `TEAM_BATTING_H`, `TEAM_PITCHING_H` (with polynomial terms), `TEAM_FIELDING_DP`, and `TEAM_FIELDING_E`.
- **Rationale:**
  - `TEAM_BATTING_H`: Captures potential non-linear effects of team hits on wins, where additional hits may have diminishing or accelerating returns.
  - `TEAM_PITCHING_H`: Models non-linear effects of hits allowed by pitching, reflecting how reducing hits might impact wins differently at various levels.
  - `TEAM_FIELDING_DP` and `TEAM_FIELDING_E`: Included as linear terms to control for the overall defensive quality while assessing the non-linear batting and pitching effects.

```r
# Step 1: Create a named vector with column names and their indexes
money_ball_train_polynomial <- money_ball_train_scaled
column_indexes <- setNames(seq_along(names(money_ball_train_polynomial)), names(money_ball_train_polynom

# Step 2: Select variables for polynomial terms
# We'll use TEAM_BATTING_H and TEAM_PITCHING_H and create polynomial terms for them

# Ensure the variables are numeric
money_ball_train_polynomial$TEAM_BATTING_H <- as.numeric(money_ball_train_polynomial[[column_indexes["TE
money_ball_train_polynomial$TEAM_PITCHING_H <- as.numeric(money_ball_train_polynomial[[column_indexes["
```

a. Feature engineer training data for Fielding-Focused Model

```r
# Step 1: Create a named vector with column names and their indexes
money_ball_evaluate_polynomial <- money_ball_evaluate_scaled
column_indexes <- setNames(seq_along(names(money_ball_evaluate_polynomial)), names(money_ball_evaluate_p

# Step 2: Select variables for polynomial terms
# We'll use TEAM_BATTING_H and TEAM_PITCHING_H and create polynomial terms for them

# Ensure the variables are numeric
money_ball_evaluate_polynomial$TEAM_BATTING_H <- as.numeric(money_ball_evaluate_polynomial[[column_index
money_ball_evaluate_polynomial$TEAM_PITCHING_H <- as.numeric(money_ball_evaluate_polynomial[[column_inde
```

b. Feature engineer evaluation data for Fielding-Focused Model

```r
# Fit a model with polynomial terms
# We'll use quadratic (degree 2) polynomial terms for TEAM_BATTING_H and TEAM_PITCHING_H
polynomial_model <- lm(TARGET_WINS ~ poly(TEAM_BATTING_H, 2) + poly(TEAM_PITCHING_H, 2) +
                                     TEAM_FIELDING_DP + TEAM_FIELDING_E,
                       data = money_ball_train_polynomial)

# Step 4: Summary of the model
summary(polynomial_model)
```

c. Fit the Fielding-Focused Model

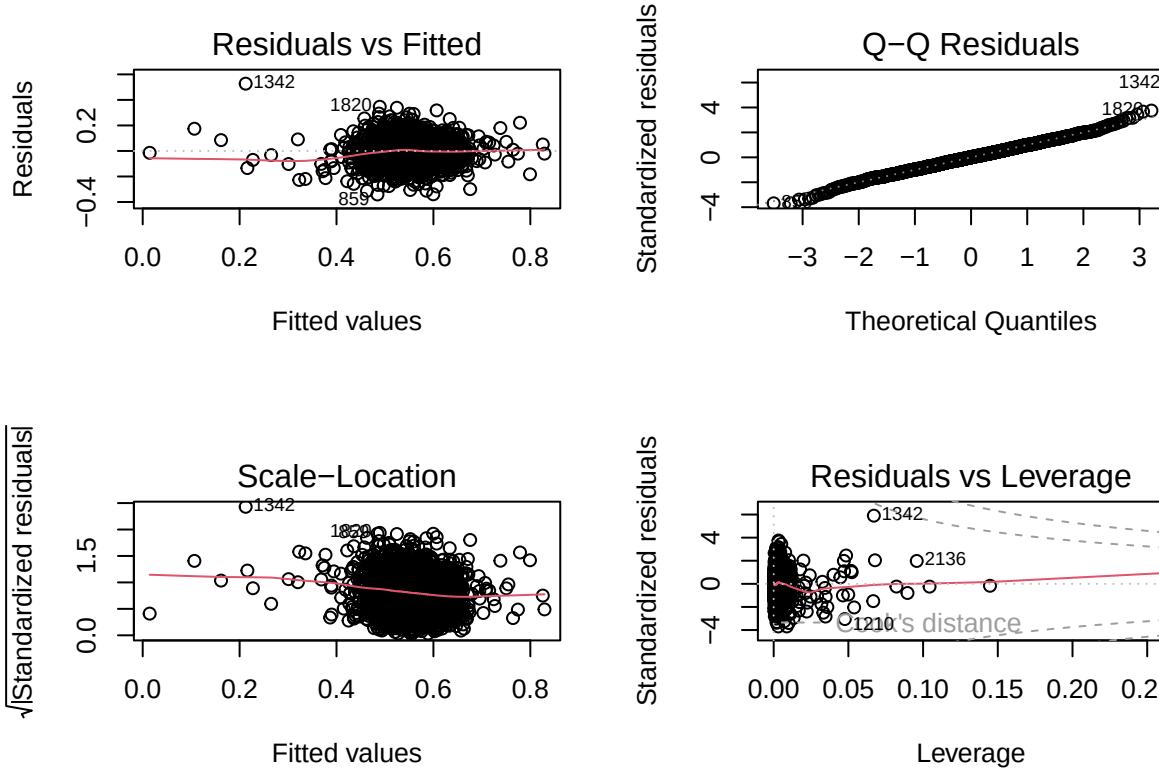
```
##
## Call:
## lm(formula = TARGET_WINS ~ poly(TEAM_BATTING_H, 2) + poly(TEAM_PITCHING_H,
##     2) + TEAM_FIELDING_DP + TEAM_FIELDING_E, data = money_ball_train_polynomial)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.34066 -0.06262  0.00083  0.06195  0.52719
```

```
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 0.69178    0.01129  61.255  < 2e-16 ***
## poly(TEAM_BATTING_H, 2)1    2.37904    0.13169  18.065  < 2e-16 ***
## poly(TEAM_BATTING_H, 2)2    0.44967    0.11395   3.946 8.18e-05 ***
## poly(TEAM_PITCHING_H, 2)1 -0.35234    0.16446  -2.142   0.0323 *
## poly(TEAM_PITCHING_H, 2)2 -0.74150    0.12806  -5.790 8.00e-09 ***
## TEAM_FIELDING_DP          -0.14335    0.01511  -9.484  < 2e-16 ***
## TEAM_FIELDING_E           -0.18983    0.01750 -10.850  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.09248 on 2269 degrees of freedom
## Multiple R-squared:  0.2672, Adjusted R-squared:  0.2652
## F-statistic: 137.9 on 6 and 2269 DF,  p-value: < 2.2e-16
```

c. Interpretation of the significant coefficients

- **poly(TEAM_BATTING_H, 2)_1:** Positive quadratic effect on wins (+2.37904). The first-order term indicates that the relationship between hits and wins is not strictly linear, with increasing hits leading to more wins, but at a varying rate.
- **poly(TEAM_BATTING_H, 2)_2:** Positive effect (+0.44967), reinforcing the non-linear relationship.
- **poly(TEAM_PITCHING_H, 2)_1:** Negative quadratic effect on wins (-0.53243). As the number of hits allowed increases, the negative impact on wins becomes more pronounced.
- **poly(TEAM_PITCHING_H, 2)_2:** Negative effect (-0.35234), further suggesting a non-linear, detrimental effect of pitching hits on wins.
- **TEAM_FIELDING_DP:** Negative impact on wins (-0.14335). More double plays are correlated with fewer wins.
- **TEAM_FIELDING_E:** Negative impact on wins (-0.18983). Fielding errors significantly reduce the number of wins.

```r
# Residual analysis of the transformed model
par(mfrow = c(2, 2))
plot(polynomial_model)
```



d. Residual analysis

Analysis of Residual Plots:

- **Residuals vs Fitted:** The residuals do not appear to follow a clear pattern, indicating a reasonable model fit. However, there is some spread at higher fitted values, which may suggest slight heteroscedasticity.
- **Q-Q Plot:** The residuals mostly follow the theoretical quantiles, but there are deviations at the tails, suggesting some issues with the normality of residuals, particularly at the extremes.
- **Scale-Location:** The red line is mostly flat, indicating that the variance of residuals is relatively constant across fitted values. This suggests that heteroscedasticity is not a major issue in this model.
- **Residuals vs Leverage:** There are a few points with higher leverage, notably point 13420, and one point (2138) with a Cook's distance near the threshold, indicating it could be influential. These points should be further investigated to ensure they are not unduly affecting the model.

6. Model Selection

```
# Calculate the R-squared, Adjusted R-squared, AIC, BIC, and Residual Standard Error (RSE) for each mod

model_list <- list(baseline_model, composite_model, interaction_model, stepwise_model, fielding_model, 

# Create a data frame to store the comparison metrics
model_comparison <- data.frame(
  Model = c("Baseline Model", "Composite Variables Model", "Interaction Terms Model", "Feature Selectio
  R_squared = sapply(model_list, function(model) summary(model)$r.squared),
```

```
  Adjusted_R_squared = sapply(model_list, function(model) summary(model)$adj.r.squared),
  AIC = sapply(model_list, AIC),
  BIC = sapply(model_list, BIC),
  RSE = sapply(model_list, function(model) sqrt(sum(residuals(model)^2) / model$df.residual))
)

# Print the comparison data frame
print(model_comparison)
```

a. Model Selection based on matrix

```
##                       Model R_squared Adjusted_R_squared       AIC       BIC
## 1            Baseline Model 0.3120711          0.3084232 -4500.993 -4420.770
## 2 Composite Variables Model 0.1822507          0.1808104 -4123.540 -4089.159
## 3   Interaction Terms Model 0.2080464          0.2059522 -4192.493 -4146.651
## 4   Feature Selection Model 0.3008771          0.2971698 -4464.256 -4384.033
## 5    Fielding-Focused Model 0.2583120          0.2566783 -4343.739 -4303.628
## 6          Polynomial Model 0.2671662          0.2652284 -4369.074 -4323.232
##          RSE
## 1 0.08972372
## 2 0.09765158
## 3 0.09614139
## 4 0.09045077
## 5 0.09301981
## 6 0.09248328
```

```
# Perform ANOVA to compare models and store the result
anova_results <- anova(baseline_model, composite_model, interaction_model, stepwise_model, fielding_mod

# Extract the ANOVA results into a data frame
anova_df <- data.frame(
  Model = c("Baseline Model", "Composite Variables Model", "Interaction Terms Model", "Feature Selectio
  Res_Df = anova_results$Res.Df,
  RSS = anova_results$RSS,
  Df = c(NA, diff(anova_results$Res.Df)),  # Compute the difference in Res.Df for Df column
  Sum_of_Sq = c(NA, anova_results$`Sum of Sq`[-1]),  # Sum of Sq starting from second model
  F_value = c(NA, anova_results$F[-1]),  # F-values starting from second model
  Pr_F = c(NA, anova_results$`Pr(>F)`[-1])  # p-values starting from second model
)

# Display the ANOVA results data frame
print(anova_df)
```

```
##                       Model Res_Df      RSS Df  Sum_of_Sq  F_value         Pr_F
## 1            Baseline Model   2263 18.21793 NA         NA       NA           NA
## 2 Composite Variables Model   2271 21.65587  8 -3.4379395 53.38186 1.107219e-79
## 3   Interaction Terms Model   2269 20.97274 -2  0.6831282 42.42850 8.139747e-19
## 4   Feature Selection Model   2263 18.51438 -6  2.4583685 50.89571 5.763087e-59
## 5    Fielding-Focused Model   2270 19.64160  7 -1.1272205 20.00305 3.480850e-26
## 6          Polynomial Model   2269 19.40711 -1  0.2344812 29.12684 7.486338e-08
```

Based on the above comparison of the models.

- **Best Model:**
  - The **Baseline Model** performs well, with the highest R-squared and Adjusted R-squared values (0.312 and 0.308 respectively). It also has the lowest AIC and BIC values, indicating that it provides a good balance between model complexity and fit. This model explains the most variance.
- **Second Best:**
  - The **Feature Selection Model** is very close to the Baseline Model in terms of R-squared and Adjusted R-squared values, with a slightly lower complexity as shown by its similar AIC and BIC. This suggests that the Feature Selection Model may be a suitable alternative if model simplicity is a priority.
- **Other Models:**
  - The **Polynomial Model** offers some improvement over the **Composite Variables Model** and the **Fielding-Focused Model** in terms of R-squared and Adjusted R-squared, but it is still not as effective as the Baseline or Feature Selection models. It shows that introducing polynomial terms adds some value, but not as much as expected.
  - The **Interaction Terms Model** performs worse than the Baseline and Feature Selection models but slightly better than the Composite Variables Model. This indicates that while interaction terms are important, they may not be as impactful as other model adjustments like feature selection.
  - The **Composite Variables Model** and **Fielding-Focused Model** are less effective overall, with lower R-squared and Adjusted R-squared values. This was somewhat expected given their design to focus on specific aspects of the game rather than capturing the overall complexity.

7. Prediction using the models

```
# Assuming the evaluation datasets are already transformed and scaled:
# money_ball_evaluate_baseline
# money_ball_evaluate_composite
# money_ball_evaluate_interaction
# money_ball_evaluate_feature_selection
# money_ball_evaluate_fielding
# money_ball_evaluate_polynomial

# Also assuming the models are:
# baseline_model, composite_model, interaction_model, stepwise_model, fielding_model, polynomial_model

# Step 1: Get the min and max values of TARGET_WINS from the original (unscaled) training data
min_target <- min(money_ball_train$TARGET_WINS)
max_target <- max(money_ball_train$TARGET_WINS)

# Step 2: Reverse the Min-Max scaling function
reverse_min_max_scale <- function(pred, min_val, max_val) {
  return(pred * (max_val - min_val) + min_val)
}

# Step 3: Make predictions on the evaluation datasets
predictions_combined <- data.frame(
  Baseline_Predictions = predict(baseline_model, newdata = money_ball_evaluate_baseline),
  Composite_Predictions = predict(composite_model, newdata = money_ball_evaluate_composite),
  Interaction_Predictions = predict(interaction_model, newdata = money_ball_evaluate_interaction),
```

```r
  Feature_Selection_Predictions = predict(stepwise_model, newdata = money_ball_evaluate_feature_selecti
  Fielding_Predictions = predict(fielding_model, newdata = money_ball_evaluate_fielding),
  Polynomial_Predictions = predict(polynomial_model, newdata = money_ball_evaluate_polynomial)
)

# Step 4: Reverse the scaling of the predictions to bring them back to their original scale
predictions_combined_original_scale <- data.frame(
  Baseline_Predictions = reverse_min_max_scale(predictions_combined$Baseline_Predictions, min_target, ma
  Composite_Predictions = reverse_min_max_scale(predictions_combined$Composite_Predictions, min_target,
  Interaction_Predictions = reverse_min_max_scale(predictions_combined$Interaction_Predictions, min_targ
  Feature_Selection_Predictions = reverse_min_max_scale(predictions_combined$Feature_Selection_Predicti
  Fielding_Predictions = reverse_min_max_scale(predictions_combined$Fielding_Predictions, min_target, ma
  Polynomial_Predictions = reverse_min_max_scale(predictions_combined$Polynomial_Predictions, min_targe
)

# Step 5: Display the reversed predictions
print(predictions_combined_original_scale)
```

```
##     Baseline_Predictions Composite_Predictions Interaction_Predictions
## 1               78.17705              76.18385                74.24624
## 2               78.51413              73.44010                72.44069
## 3               88.12298              82.35093                78.92087
## 4               98.76405              99.71393               104.37911
## 5               88.69616              80.62631                77.42297
## 6               84.30510              88.94025                84.01810
## 7               88.41887              83.85999                86.79870
## 8               92.39348              83.39301                82.22149
## 9               89.86488              79.90986                80.21275
## 10              87.76419              85.88388                80.85467
## 11              82.54949              84.33077                82.01044
## 12              94.89749              87.04892                86.06607
## 13              93.39764              85.46037                82.03634
## 14              90.83109              88.61556                86.43144
## 15              96.56320              90.48951                86.54169
## 16              87.09654              80.60579                78.42253
## 17              85.10411              86.04610                83.05763
## 18              90.64991              92.61806                94.24988
## 19              89.82556              84.80113                81.46832
## 20             101.08645              99.63857               101.40889
## 21              99.29625              96.06732                96.21803
## 22              95.95819              95.20084                95.51877
## 23              95.20387              90.83665                85.97590
## 24              77.63303              83.21838                74.77962
## 25              96.56614              94.87525                98.21592
## 26             100.72033              98.13508               104.10688
## 27              72.93475              73.98791                69.63759
## 28              89.45513              92.09789                86.80482
## 29             100.29234             102.51898               100.77321
## 30              89.16934              99.76861                95.13044
## 31             103.29758             101.89016               101.97287
## 32              98.61237              91.69847                90.27121
## 33              96.89310              92.76724                91.71510
## 34              95.19904              91.69451                89.06064
```

```
## 35    95.83279   92.22164   88.85851
## 36    92.99360   94.16530   89.25981
## 37    90.14829   88.33958   87.43335
## 38   102.45857   93.38000   93.21653
## 39    95.03866   91.31846   91.22002
## 40   105.45706   98.79395   95.63925
## 41    91.27145  101.02975   98.95233
## 42   109.56606  105.73752  107.38785
## 43    58.89194   56.38334   54.64717
## 44   113.55826   84.94436   90.81760
## 45   100.48467   81.12756   85.17998
## 46   107.56729   89.66017   93.71117
## 47   111.74202   89.10293   92.52956
## 48    90.30035   90.33871   84.15297
## 49    87.07779   83.77065   81.71446
## 50    93.48384   91.98038   85.25010
## 51    90.90507   90.75607   88.30687
## 52    98.48108   97.04324   98.32476
## 53    92.02002   89.02285   86.09103
## 54    89.37692   84.08440   80.77517
## 55    88.47775   90.27641   87.98151
## 56    97.10792   89.56134   88.31903
## 57   109.01530   89.39839   91.83270
## 58    94.16911   87.52230   87.92501
## 59    77.82097   72.66348   74.72945
## 60    99.89554   86.45910   86.12826
## 61   107.33419   94.64413   97.63116
## 62    93.18991   81.46490   80.18280
## 63   100.92898   97.00399   97.84974
## 64    92.44078   94.59260   96.09028
## 65    94.73330   90.89451   94.24462
## 66   108.87416   93.35399  102.00651
## 67    95.24478   86.49169   84.12904
## 68   102.29157   89.56052   88.53768
## 69    94.68427   80.75344   83.11651
## 70   108.92928   87.80819   89.74354
## 71   107.02291   92.02756   89.71150
## 72    86.11216   82.83831   82.00997
## 73    88.12520   81.49305   81.23513
## 74    98.30429   87.63679   86.54724
## 75    97.94957   98.36148   96.37022
## 76    98.20945   89.44309   87.62042
## 77    98.56919   87.78581   89.89645
## 78    96.07118   93.93380   94.49354
## 79    91.08924   86.84248   85.82119
## 80    96.87154   87.30901   87.72116
## 81    95.93966  102.85298  105.89415
## 82   104.69396  104.22670  106.00452
## 83   110.33516  109.44438  115.68388
## 84    88.69665   88.87434   88.41528
## 85    97.91140   88.67890   84.68835
## 86    90.45449   86.76111   87.51732
## 87    94.85271   87.96325   88.42625
## 88    98.78717   90.10911   87.31547
```

```
## 89          102.27317          92.46009          93.48286
## 90          101.75629         103.72164         108.54706
## 91           91.93779          77.11361          79.34120
## 92          103.01974          91.74177         107.64509
## 93           89.55447          80.13047          78.48615
## 94          108.41620          93.10567          91.81506
## 95          102.79201          94.24958          95.76311
## 96          103.39308          91.65413          91.57515
## 97          103.57581          90.96327          87.22066
## 98          120.83876         106.04523         104.54819
## 99          105.06330          97.62010          97.63466
## 100         104.07997          94.23601          97.84688
## 101          97.48281          94.28743          94.46305
## 102          84.25547          86.40986          80.22546
## 103          97.26661          95.12448          91.53720
## 104          95.49506          87.80564          84.50085
## 105          88.58209          90.62811          90.29441
## 106          86.73152          85.09769          82.45571
## 107          65.14375          69.63642          64.93304
## 108          92.34623          97.48263         100.95230
## 109         102.31137          99.51690         101.93765
## 110          72.08048          81.01120          77.19468
## 111          97.72403          92.10733          93.01296
## 112          98.81818          97.30893         100.72568
## 113         111.70348          97.18547         103.37735
## 114         109.83301         101.44575         105.86788
## 115          98.52207          93.73198          94.55480
## 116          95.17655          98.14052         100.16362
## 117         102.19718          94.28895         101.34176
## 118          94.57018          94.37537          95.99595
## 119          90.47786          94.19861          93.31651
## 120          86.00232          79.29996          82.04607
## 121         107.33159          85.42234          88.39190
## 122          84.37666          81.82360          81.08287
## 123          88.29264          80.34874          78.33083
## 124          85.08595          77.47224          76.68738
## 125          87.45108          78.99502          79.32675
## 126         105.64739          94.48449          92.19980
## 127         108.93739          98.68859          95.06365
## 128          96.46367          87.27903          84.76176
## 129         108.71795         103.46045         103.07789
## 130         105.77894         100.36055         102.72447
## 131         100.62730          99.81222          98.98442
## 132          93.28199          93.53184          93.84177
## 133          87.66268          91.63756          90.50732
## 134         100.24419          89.70523          91.55420
## 135         104.53858          91.74324          95.01120
## 136          87.98890          91.63499          84.52474
## 137          88.89822          84.27938          83.80725
## 138          91.10158          88.92932          85.11696
## 139         102.40840          96.54681         100.93806
## 140          92.06019          90.46396          90.06462
## 141          85.56891          83.15165          80.93536
## 142          93.47927          80.57785          79.98433
```

```
## 143  108.79245  98.67833  104.07108
## 144   88.41700  89.93250   85.92641
## 145   88.21201  86.28367   83.40037
## 146   80.54888  83.93286   73.98848
## 147   90.64526  89.33513   89.57293
## 148   94.76796  89.95467   90.78778
## 149   93.27245  86.92926   88.34727
## 150   97.58230  97.15041   96.04798
## 151   93.49682  90.24472   90.02986
## 152   99.02187  97.99320  100.69664
## 153   63.42127  53.60365   31.08794
## 154   84.49508  82.76482   77.48973
## 155   94.94241  87.37643   87.26619
## 156   83.10965  76.76255   78.59016
## 157  104.89398 103.50440  108.57936
## 158   79.77823  76.73243   71.54244
## 159  115.98167  95.58947   95.05196
## 160   92.45301  81.94832   80.87735
## 161  119.23709 104.86370  107.30157
## 162  121.03266 103.25494  107.54499
## 163  110.92798 100.37165   99.30395
## 164  117.08621 105.38130  111.25689
## 165  111.10936 100.90948  101.16870
## 166  104.00918  92.86793   89.67950
## 167   95.65582  82.03015   81.33140
## 168   92.27878  84.95438   84.89566
## 169   84.42858  82.39544   84.54098
## 170   90.92121  93.77780   95.54035
## 171  111.68652  91.34888   93.95922
## 172  108.99257  94.48826   96.82933
## 173   96.66547  95.29596   91.86693
## 174  110.38353 102.99567  103.73976
## 175   97.10043  98.60469   96.91617
## 176   89.13761  84.58259   80.77510
## 177   89.98276  85.69921   82.64846
## 178   81.91901  78.98417   76.46273
## 179   86.39962  85.79834   79.52468
## 180   92.01782  84.50780   85.79619
## 181   93.44381  86.83747   88.97276
## 182   98.93896  96.50975   96.64355
## 183   98.07821  92.99579   95.92571
## 184   97.44144  96.55731   96.13919
## 185  111.11939  93.90085  107.61383
## 186  103.61745  94.25837  102.36101
## 187   99.37364  94.98195   99.92262
## 188   70.77130  88.13272   81.97551
## 189   69.09341  80.05886   74.54998
## 190  121.20009 108.05291  122.58096
## 191   89.65111  85.32858   81.96058
## 192  103.31011  89.81268   88.94707
## 193   91.64633  90.09568   86.03500
## 194   91.07632  90.52914   88.03304
## 195   92.82522  99.62923   98.28430
## 196   76.61401  81.38434   76.15151
```

```
## 197  90.59324  90.91765  86.76310
## 198 100.47571  91.20546  87.48440
## 199  94.38483  86.53946  83.87912
## 200 102.77546  93.53959  91.75804
## 201  92.05851  84.48094  80.81918
## 202  99.37530  89.84253  87.76346
## 203  94.13245  88.27500  85.16062
## 204 102.72031  93.62103  97.68719
## 205  93.77991  92.45904  92.38067
## 206  97.92234  89.30723  87.89251
## 207  96.69000  96.21958  99.47798
## 208  95.34705  97.68652  97.11785
## 209  92.84318  88.57758  91.43110
## 210  87.03879  77.33190  77.72205
## 211 120.22309  97.69676 105.20913
## 212 112.90955  89.75202  90.26458
## 213 102.93591  87.01468  87.54581
## 214  73.69973  75.77800  71.69358
## 215  81.21571  79.88430  73.94542
## 216 100.74120  90.93782  89.33035
## 217  98.19047  85.26099  86.52119
## 218 105.87253  98.52615 101.25293
## 219  93.33503  90.73987  89.65048
## 220  95.85995  91.47023  90.99658
## 221  91.05244  93.10447  93.35235
## 222  90.65922  91.03762  94.07691
## 223  93.55006  97.87159 100.64178
## 224  90.33101  88.58631  89.83867
## 225 107.88805  92.68147  90.48246
## 226  91.44302  81.83602  81.44303
## 227  93.65891  92.46812  90.68035
## 228  93.59290  93.14625  94.87655
## 229  99.61657  95.63026  96.55251
## 230  87.18515  83.11752  77.71921
## 231  96.91146  87.23674  87.12285
## 232 108.02665  92.99137  92.53835
## 233  89.75444  81.23254  78.21316
## 234  97.76426  91.27418  89.75353
## 235  92.21308  89.01693  86.05493
## 236  91.85923  86.24661  84.06315
## 237  95.71778  92.09926  90.67448
## 238  94.68096  86.03436  86.05902
## 239 107.75020  86.25135  87.13246
## 240  90.19941  79.51920  80.04038
## 241 106.57410  96.45743  96.35673
## 242 101.48066  96.10058  94.64443
## 243  99.10892  94.96642  93.93778
## 244  99.12934  96.00532 100.21682
## 245  69.33191  74.29938  74.62336
## 246  97.68638  93.92954  96.10741
## 247  95.02981  91.02884  89.52541
## 248  97.27347  96.69182  97.23354
## 249  84.26666  84.39881  77.40492
## 250  97.38026  92.77951  99.11458
```

```
## 251              97.63192              97.24347              100.38817
## 252              85.80779              87.01311               79.39076
## 253             106.55936              88.03743               91.56509
## 254              33.15306              39.25295               56.81013
## 255              80.16673              77.72555               75.10942
## 256              84.77039              81.34989               78.17446
## 257              99.56753              97.02074               98.82328
## 258              95.01419              93.28984               97.08887
## 259              89.94374             100.77913              100.15117
##     Feature_Selection_Predictions Fielding_Predictions Polynomial_Predictions
## 1                         72.15273             76.38488               73.48109
## 2                         71.87236             77.02300               73.38248
## 3                         80.49442             89.84173               86.11691
## 4                         86.92991             97.19790               98.73778
## 5                         82.44192             79.30133               81.51843
## 6                         81.84048             83.73845               87.23966
## 7                         86.59711             82.23539               79.91001
## 8                         84.83630             90.73168               87.89559
## 9                         87.51721             79.75160               77.14638
## 10                        82.74758             88.58588               86.71343
## 11                        76.74111             83.09745               82.11334
## 12                        89.09590             91.04376               90.56467
## 13                        90.04664             88.83104               87.49823
## 14                        84.92933             93.14973               92.33800
## 15                        89.93631             99.19281               96.08222
## 16                        78.54530             88.56793               85.39546
## 17                        80.58782             86.17695               83.52416
## 18                        83.11239             89.48646               89.80609
## 19                        83.84839             87.04820               83.23763
## 20                        88.21163             99.03772              103.76238
## 21                        94.00780             93.05982               94.64591
## 22                        86.75936             96.14845               98.53488
## 23                        93.03208             89.98327               89.85957
## 24                        73.07962             83.41576               81.97252
## 25                        85.56155             96.50074               96.95362
## 26                        88.98518            100.23429              101.46700
## 27                        69.04438             73.76334               72.57377
## 28                        82.90187             89.64137               89.22274
## 29                        92.06685             97.07428               99.25078
## 30                        78.53803             93.61554               95.24153
## 31                        94.44365             98.51972              101.45303
## 32                        93.50982             94.58853               95.28785
## 33                        91.42505             91.80710               92.91331
## 34                        89.21939             92.18186               93.19826
## 35                        90.39712             92.20068               91.56684
## 36                        87.64222             94.90994               93.57873
## 37                        87.77861             86.29477               85.32660
## 38                        94.27314             98.20042               98.43099
## 39                        88.91977             97.07234              101.52067
## 40                        95.86660            105.55278              105.94756
## 41                        81.35924             94.69973               96.30375
## 42                        99.41534            106.29278              108.39539
## 43                        51.79958             44.32493               50.44626
## 44                       112.40502             94.46321               94.16662
```

```
## 45  97.67076  90.16896  88.34127
## 46 106.23105  94.60883  96.72640
## 47 106.93937 101.33498 103.86746
## 48  88.52424  86.10400  84.13236
## 49  84.71489  80.63803  78.13778
## 50  89.84064  91.76251  90.14863
## 51  83.72639  89.83317  90.15702
## 52  87.72043  99.65914 102.15200
## 53  87.35554  88.36138  86.41435
## 54  84.30104  86.40171  83.68332
## 55  82.02697  88.50158  86.92475
## 56  89.82060  94.62359  92.48187
## 57 108.23926  96.24099  95.65390
## 58  91.71831  84.86920  82.47314
## 59  71.20000  70.92626  68.49547
## 60  95.39773  91.06894  87.42243
## 61 102.98886  98.47582  99.81039
## 62  90.89553  84.53038  83.03349
## 63  93.30810  99.95238  99.65843
## 64  82.68549  93.85741  94.30693
## 65  85.84239  93.41706  92.71168
## 66 105.40016  96.79783  99.92027
## 67  91.14757  88.69130  87.03213
## 68  93.74446  97.00442  97.10452
## 69  91.15696  84.71473  80.71921
## 70 106.69018  94.30300  91.21628
## 71 100.95424  98.08020  99.19429
## 72  80.36912  84.07766  83.54437
## 73  84.45932  83.68962  82.77419
## 74  92.75980  96.31337  94.95687
## 75  84.18590 100.67510 101.50915
## 76  88.20196  98.13929  96.81446
## 77  95.89244  90.29561  89.09200
## 78  89.66986  94.59908  93.28457
## 79  87.92941  84.72062  82.06231
## 80  90.05042  90.66664  87.98265
## 81  81.04743  95.12933  99.85568
## 82  89.86632 100.16472 104.93277
## 83  88.71552 105.81486 114.11219
## 84  80.41181  87.13388  88.38365
## 85  94.19668  94.05850  92.68665
## 86  83.04618  92.50443  91.74642
## 87  89.13128  93.95055  90.93332
## 88  92.08116  97.27391  95.49414
## 89  97.90801  98.57044  96.62380
## 90  89.85141  98.01660 100.37821
## 91  88.75938  83.59645  81.14474
## 92 102.72445  88.56250  81.55585
## 93  86.81735  85.17477  82.88502
## 94  99.47717 101.77495  99.27296
## 95  97.51316  96.52957  94.56948
## 96 101.13414  94.15468  91.02509
## 97 102.15779  93.63327  91.39177
## 98 109.72336 115.68215 121.09048
```

```
## 99          96.91458   97.95816  100.41294
## 100         96.79654   93.26203   95.84332
## 101         89.33895   91.57474   93.35542
## 102         81.96320   85.56621   83.91499
## 103         89.17895   97.87148   99.04124
## 104         91.74118   94.10635   92.17650
## 105         81.69915   89.22355   88.11951
## 106         85.55365   79.42284   83.23989
## 107         63.15883   60.80869   62.63969
## 108         80.82157   89.81313   91.46665
## 109         95.82837   96.78182   97.31100
## 110         71.81845   68.70329   70.34761
## 111         94.66265   90.12679   89.13045
## 112         92.01527   91.31200   91.93339
## 113        102.07257  104.12524  105.32548
## 114         98.49148  103.74196  105.53576
## 115         89.55688   94.28908   94.01458
## 116         85.27965   91.36147   92.76187
## 117         91.04725   95.27922   97.22747
## 118         88.96372   89.38060   89.22656
## 119         81.03644   90.07916   90.24347
## 120         84.92968   76.23888   74.15258
## 121        104.70162   95.24423   93.69232
## 122         81.94394   78.53748   75.59765
## 123         82.79117   81.61236   78.65102
## 124         77.82082   77.18507   75.19597
## 125         85.32967   77.45101   74.73043
## 126         96.49748  102.62685  103.73170
## 127         99.43993  106.80784  109.55518
## 128         92.81311   89.60115   87.10604
## 129        100.65214  104.01266  106.53187
## 130         91.12815  102.43929  107.70300
## 131         90.30248   99.61705  102.50510
## 132         88.22759   91.54785   92.11796
## 133         81.43405   91.06693   88.69839
## 134         94.26134   96.41551   95.84996
## 135         97.56078   99.64027   98.23425
## 136         87.27899   89.12857   91.71722
## 137         85.33620   83.26253   81.29669
## 138         83.33015   92.77904   91.15562
## 139         94.06932   98.22998   98.31055
## 140         85.44922   92.35784   90.74263
## 141         78.64247   80.42307   77.70477
## 142         86.20488   85.05081   80.84403
## 143         98.61415   99.90052  104.06344
## 144         83.38636   85.57247   85.60701
## 145         86.55758   84.35942   83.45963
## 146         77.55691   83.06338   81.86987
## 147         86.81047   85.91025   84.53424
## 148         89.20996   90.40045   89.60993
## 149         84.60791   90.90172   90.81994
## 150         91.10612   94.97666   94.92460
## 151         86.71871   90.87142   90.06111
## 152         89.88080   96.63724   97.37305
```

```
## 153  59.40415  71.20215  28.02007
## 154  75.02409  87.13132  83.72504
## 155  89.37275  92.51933  89.70731
## 156  74.74361  81.65344  77.77128
## 157  96.39615  97.13513  98.64104
## 158  77.63683  75.32646  75.72177
## 159 112.50693 104.05868 104.85131
## 160  86.41545  86.84650  82.53119
## 161 108.07138 108.51330 113.85188
## 162 111.11790 110.56227 115.43074
## 163 106.18242 103.48886 105.40087
## 164 103.17095 107.12479 113.60542
## 165 104.07935 102.47339 105.16831
## 166  99.92961  97.03444  98.07485
## 167  90.51757  90.10417  88.78115
## 168  86.84149  89.66465  88.22169
## 169  77.80953  83.63730  81.05749
## 170  82.52777  92.07442  92.18304
## 171 107.14433  98.04412  96.61330
## 172 105.95374  97.91392  96.85098
## 173  93.61945  91.72578  92.48174
## 174 101.22982 106.38763 109.73160
## 175  90.52772  94.14155  95.69832
## 176  87.59646  84.66548  83.40119
## 177  88.87285  86.64493  85.16170
## 178  77.26141  80.54714  78.74734
## 179  82.84014  87.08521  85.69803
## 180  87.00205  88.93584  86.25817
## 181  90.83740  88.02689  85.34147
## 182  94.00236  95.47923  94.48967
## 183  90.74799  96.60247  95.82689
## 184  91.24952  95.38293  95.38856
## 185 108.53307  97.86502  93.45793
## 186  96.04409  92.70711  95.81969
## 187  93.59683  91.11009  94.17214
## 188  65.14683  70.15325  74.46463
## 189  69.01579  69.03933  71.32997
## 190 101.82218 114.49909 127.88569
## 191  84.58669  86.73597  82.93454
## 192  97.84957  95.27291  92.56010
## 193  83.25139  90.42977  89.36643
## 194  82.49588  92.69532  92.99390
## 195  79.64364  96.99816 100.07728
## 196  72.59365  80.57662  78.13273
## 197  85.83005  91.08309  89.76998
## 198 100.47244  93.32640  92.05330
## 199  93.08106  88.59143  86.89521
## 200  97.59923  97.12980  97.02125
## 201  86.60677  88.35786  85.48401
## 202  92.72838  95.75507  93.85441
## 203  91.57104  87.35670  85.54438
## 204 100.18402  97.67338 101.76800
## 205  87.33982  93.77832  92.34497
## 206  95.70905  91.75671  88.80741
```

```
## 207  89.98982  92.31661  94.72722
## 208  89.25900  92.15115  92.26166
## 209  91.84161  83.63314  84.52521
## 210  85.95931  78.02837  75.76286
## 211 115.11658 104.22853 110.27073
## 212 105.71819 102.09789 102.90162
## 213 102.45944  88.64099  86.45980
## 214  70.74308  74.16965  72.00439
## 215  79.58487  77.56077  76.04045
## 216  92.67075  97.58195  98.19389
## 217  90.00076  91.24373  90.20444
## 218  96.44173 100.35305 100.79351
## 219  86.25680  92.60503  90.85098
## 220  93.53688  89.42844  88.06028
## 221  84.52230  90.10952  91.57649
## 222  82.81499  86.56102  85.85210
## 223  82.65054  91.80347  93.36908
## 224  81.45042  87.35677  87.18884
## 225  94.89549 109.31769 116.15448
## 226  88.29015  84.72886  81.65888
## 227  86.11233  95.31497  93.66905
## 228  88.24554  88.46802  88.03779
## 229  94.95976  94.07409  93.52108
## 230  83.12563  83.50593  84.48666
## 231  94.40099  90.54093  87.00451
## 232 100.38154 101.45025 103.18142
## 233  86.96677  85.20273  83.62867
## 234  89.50834  98.60530  97.01481
## 235  88.59416  89.80573  87.34298
## 236  88.30962  87.95089  84.91346
## 237  86.90777  97.86388  96.04321
## 238  90.23441  88.41820  86.10906
## 239 107.34924  93.68731  94.11004
## 240  88.73048  80.83120  80.30063
## 241  99.13935 100.07723 101.61897
## 242  93.54107  98.39605 100.60819
## 243  90.85781  98.80079 100.06441
## 244  87.99482  94.59576  95.93433
## 245  66.32968  69.20037  68.42053
## 246  86.21524 100.14554 101.30677
## 247  86.66418  96.07934  94.26038
## 248  89.16134  94.61762  94.68638
## 249  78.86195  86.22890  83.94393
## 250  87.64569  91.85666  92.54253
## 251  89.39584  92.25856  93.15241
## 252  78.24607  82.35898  84.83688
## 253 104.25803  95.11872  96.31235
## 254  35.23215  13.51457  36.16923
## 255  73.87920  81.46579  78.47868
## 256  80.73676  85.14293  81.62669
## 257  94.38506  94.37000  93.36460
## 258  90.52887  89.32203  88.43965
## 259  78.75336  88.58339  90.21146
```

```r
# Step 6: Optionally, view the first few rows of the reversed predictions
head(predictions_combined_original_scale)
```

```
##   Baseline_Predictions Composite_Predictions Interaction_Predictions
## 1             78.17705              76.18385                74.24624
## 2             78.51413              73.44010                72.44069
## 3             88.12298              82.35093                78.92087
## 4             98.76405              99.71393               104.37911
## 5             88.69616              80.62631                77.42297
## 6             84.30510              88.94025                84.01810
##   Feature_Selection_Predictions Fielding_Predictions Polynomial_Predictions
## 1                      72.15273             76.38488                73.48109
## 2                      71.87236             77.02300                73.38248
## 3                      80.49442             89.84173                86.11691
## 4                      86.92991             97.19790                98.73778
## 5                      82.44192             79.30133                81.51843
## 6                      81.84048             83.73845                87.23966
```