Moneyball Analysis: Modeling MLB Team Performance

Yashika Arora

Contents

Introduction

This assignment uses the Moneyball dataset to analyze which offensive and defensive factors best predict team success, measured in wins. We explored the training data to understand its structure, addressed missing values and skewness, and prepared the dataset through systematic cleaning and transformations. Using these predictors, we developed three regression models, compared their performance, and validated assumptions through diagnostics. By balancing statistical rigor with baseball intuition, we highlight which metrics most reliably explain team wins and why a simple, well-grounded model can outperform more complex alternatives.

Part 1 — Data Exploration

In this section, we explored the training dataset by summarizing its size, key variables, and distributions. We examined missing values, visualized data quality, and identified potential outliers through boxplots. These steps provided an initial understanding of the data and highlighted areas needing transformation or imputation before modeling.

Loading the Data

In this section, we will load the training and evaluation datasets from their respective CSV files and display a preview of the first few rows of the data.

```
# Load the data
train_data <- read.csv("C:/Users/Hp/Downloads/moneyball-training-data-1-1.csv")
eval_data <- read.csv("C:/Users/Hp/Downloads/moneyball-evaluation-data-1-1.csv")
```

Report dataset dimensions (rows × columns) for train_data

```
dims_tbl <- tibble::tibble(
  Metric = c("Rows","Columns"),
  Value  = scales::comma(c(nrow(train_data), ncol(train_data)))
)
knitr::kable(dims_tbl, caption = "Dataset Dimensions: train_data", align = c("l","r"))
```

Table 1: Dataset Dimensions: train_data

Metric	Value
Rows	2,276
Columns	17

Descriptive Statistics — Training Data

This chunk creates a clean table of **n, mean, SD, median, min, max, range, skewness, kurtosis, and SE** for the Moneyball training dataset, formatted to three decimals,

```
library(psych); library(dplyr); library(knitr); library(kableExtra)

psych::describe(dplyr::select(train_data, where(is.numeric))) |>
  tibble::rownames_to_column("Variable") |>
  dplyr::select(Variable, n, mean, sd, median, min, max, range, skew, kurtosis, se) |>
  dplyr::mutate(dplyr::across(where(is.numeric), ~ round(.x, 3))) |>
  knitr::kable(caption = "Descriptive statistics — training data") |>
  kableExtra::kable_styling(font_size = 9)
```

Table 2: Descriptive statistics — training data

Variable	n	mean	sd	median	min	max	range	skew	kurtosis	se
Index	2276	1268.464	736.349	1270.5	1	2535	2534	0.004	-1.217	15.435
Target.Wins	2276	80.791	15.752	82.0	0	146	146	-0.399	1.027	0.330
Team.Batting.Hits	2276	1469.270	144.591	1454.0	891	2554	1663	1.571	7.279	3.031
Team.Batting.Doubles	2276	241.247	46.801	238.0	69	458	389	0.215	0.006	0.981
Team.Batting.Triples	2276	55.250	27.939	47.0	0	223	223	1.109	1.503	0.586
Team.Batting.Home.Runs	2276	99.612	60.547	102.0	0	264	264	0.186	-0.963	1.269
Team.Batting.Walks	2276	501.559	122.671	512.0	0	878	878	-1.026	2.183	2.571
Team.Batting.Strikeouts	2174	735.605	248.526	750.0	0	1399	1399	-0.298	-0.321	5.330
Team.Baserun.Stolen.Bases	2145	124.762	87.791	101.0	0	697	697	1.972	5.490	1.896
Team.Baserun.Caught.Stealing	1504	52.804	22.956	49.0	0	201	201	1.976	7.620	0.592
Team.Batting.Hit.By.Pitch	191	59.356	12.967	58.0	29	95	66	0.319	-0.112	0.938
Team.Pitching.Hits	2276	1779.210	1406.843	1518.0	1137	30132	28995	10.330	141.840	29.489
Team.Pitching.Home.Runs	2276	105.699	61.299	107.0	0	343	343	0.288	-0.605	1.285
Team.Pitching.Walks	2276	553.008	166.357	536.5	0	3645	3645	6.744	96.968	3.487
Team.Pitching.Strikeouts	2174	817.730	553.085	813.5	0	19278	19278	22.175	671.189	11.862
Team.Fielding.Errors	2276	246.481	227.771	159.0	65	1898	1833	2.990	10.970	4.774
Team.Fielding.Double.Plays	1990	146.388	26.226	149.0	52	228	176	-0.389	0.182	0.588

Missing Values — Training Data

This section calculates the number of missing values for each variable in the Moneyball training dataset.

```
library(naniar)
library(knitr)

missing_tbl <- naniar::miss_var_summary(train_data)

kable(missing_tbl, caption = "Missing values — training data")
```

Table 3: Missing values — training data

variable	n_miss	pct_miss
Team.Batting.Hit.By.Pitch	2085	91.6
Team.Baserun.Caught.Stealing	772	33.9
Team.Fielding.Double.Plays	286	12.6
Team.Baserun.Stolen.Bases	131	5.76
Team.Batting.Strikeouts	102	4.48
Team.Pitching.Strikeouts	102	4.48
Index	0	0
Target.Wins	0	0
Team.Batting.Hits	0	0
Team.Batting.Doubles	0	0
Team.Batting.Triples	0	0
Team.Batting.Home.Runs	0	0
Team.Batting.Walks	0	0
Team.Pitching.Hits	0	0
Team.Pitching.Home.Runs	0	0
Team.Pitching.Walks	0	0
Team.Fielding.Errors	0	0

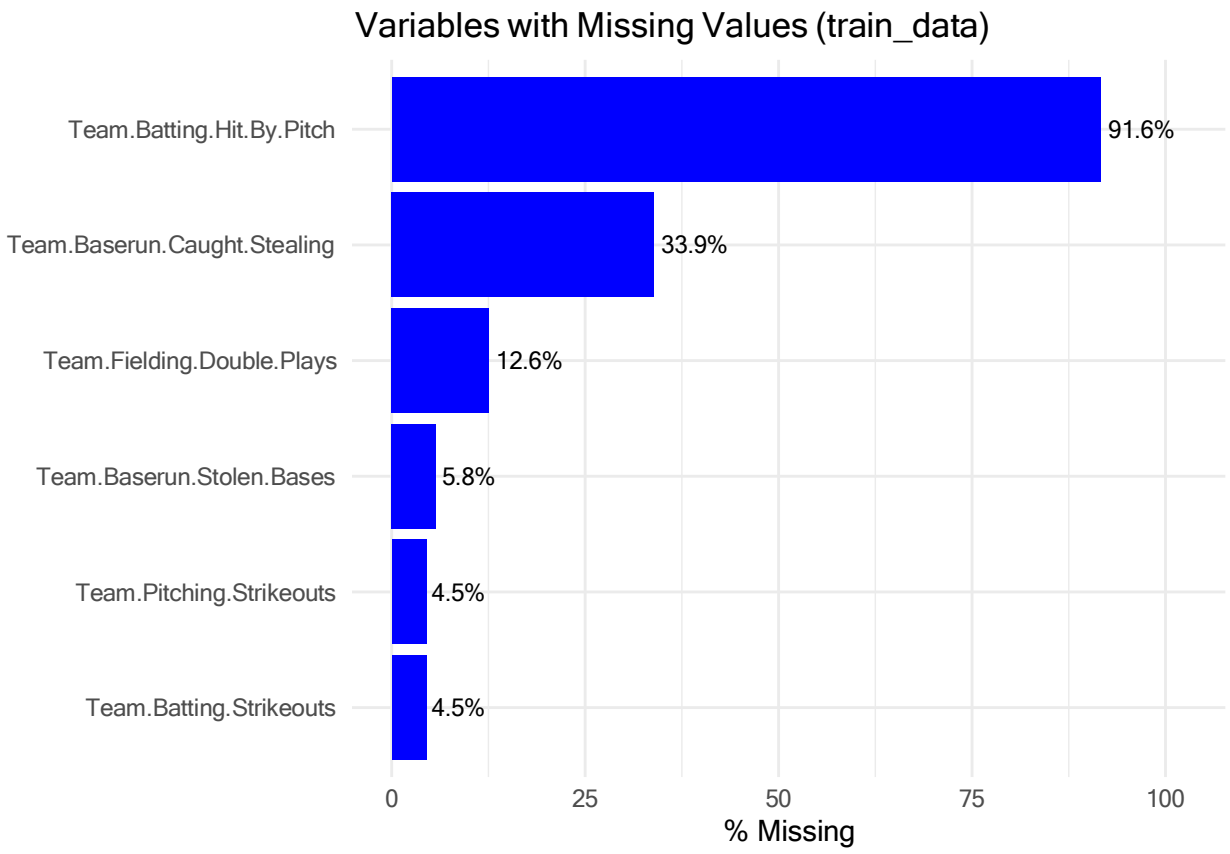
We start with a ranked bar chart showing % missing per variable.

```
# % missing per column (keep only >0)
miss_pct <- 100 * colMeans(is.na(train_data))
miss_pct <- miss_pct[miss_pct > 0]

# Order once; use the same order for names and values
ord <- order(miss_pct)   # ascending (use order(-miss_pct) for descending)

plot_df <- data.frame(
  var = factor(names(miss_pct)[ord], levels = names(miss_pct)[ord]),
  pct = as.numeric(miss_pct[ord])
)

ggplot(plot_df, aes(x = var, y = pct)) +
  geom_col(fill = "blue") +
  geom_text(aes(label = sprintf("%.1f%%", pct)), hjust = -0.1, size = 3) +
  coord_flip(clip = "off") +
  expand_limits(y = max(plot_df$pct) * 1.12) +
  labs(x = NULL, y = "% Missing", title = "Variables with Missing Values (train_data)") +
  theme_minimal()
```



Conclusion on Missing Data & Imputation Plan

The missingness check on train_data revealed one variable with overwhelming gaps:
- **Team.Batting.Hit.By.Pitch (~91.6%)** → dropped due to limited signal.

Variables with **moderate missingness (5–35%)** were median-imputed (robust to outliers):
- **Team.Baserun.Caught.Stealing (~33.9%)**
- **Team.Fielding.Double.Plays (~12.6%)**
- **Team.Baserun.Stolen.Bases (~5.8%)**

Variables with **<5% missingness** were also median-imputed for completeness:
- **Team.Batting.Strikeouts (~4.5%)**
- **Team.Pitching.Strikeouts (~4.5%)**

For categorical fields (if present), we would assign a new level, **"Missing"**, rather than numeric imputation.

This approach preserves high-quality features, limits noise from extreme imputations, and produces a clean dataset ready for modeling.

```
library(dplyr)
library(forcats)

# 1) Identify % missing per column
miss_pct <- 100 * colMeans(is.na(train_data))

# 2) Drop variables with >70% missing
drop_vars <- names(miss_pct[miss_pct > 70])
```

```r
train_data_clean <- train_data %>% select(!all_of(drop_vars))

# 3) Impute numerics with median; factors with explicit "Missing" level
train_data_imputed <- train_data_clean %>%
  mutate(
    across(where(is.numeric),
           ~ ifelse(is.na(.x), median(.x, na.rm = TRUE), .x)),
    across(where(is.factor),
           ~ fct_explicit_na(.x, na_level = "Missing"))
  )
```

We quickly verify that no missing values remain after imputation (overall and by column).

```r
x <- colSums(is.na(train_data_imputed))

tbl <- data.frame(
  Variable = names(x),
  Missing  = as.integer(x)
)

knitr::kable(tbl[order(-tbl$Missing), ], caption = "Missing Values by Column (train_data_imputed)")
```

Table 4: Missing Values by Column (train_data_imputed)

Variable	Missing
Index	0
Target.Wins	0
Team.Batting.Hits	0
Team.Batting.Doubles	0
Team.Batting.Triples	0
Team.Batting.Home.Runs	0
Team.Batting.Walks	0
Team.Batting.Strikeouts	0
Team.Baserun.Stolen.Bases	0
Team.Baserun.Caught.Stealing	0
Team.Pitching.Hits	0
Team.Pitching.Home.Runs	0
Team.Pitching.Walks	0
Team.Pitching.Strikeouts	0
Team.Fielding.Errors	0
Team.Fielding.Double.Plays	0

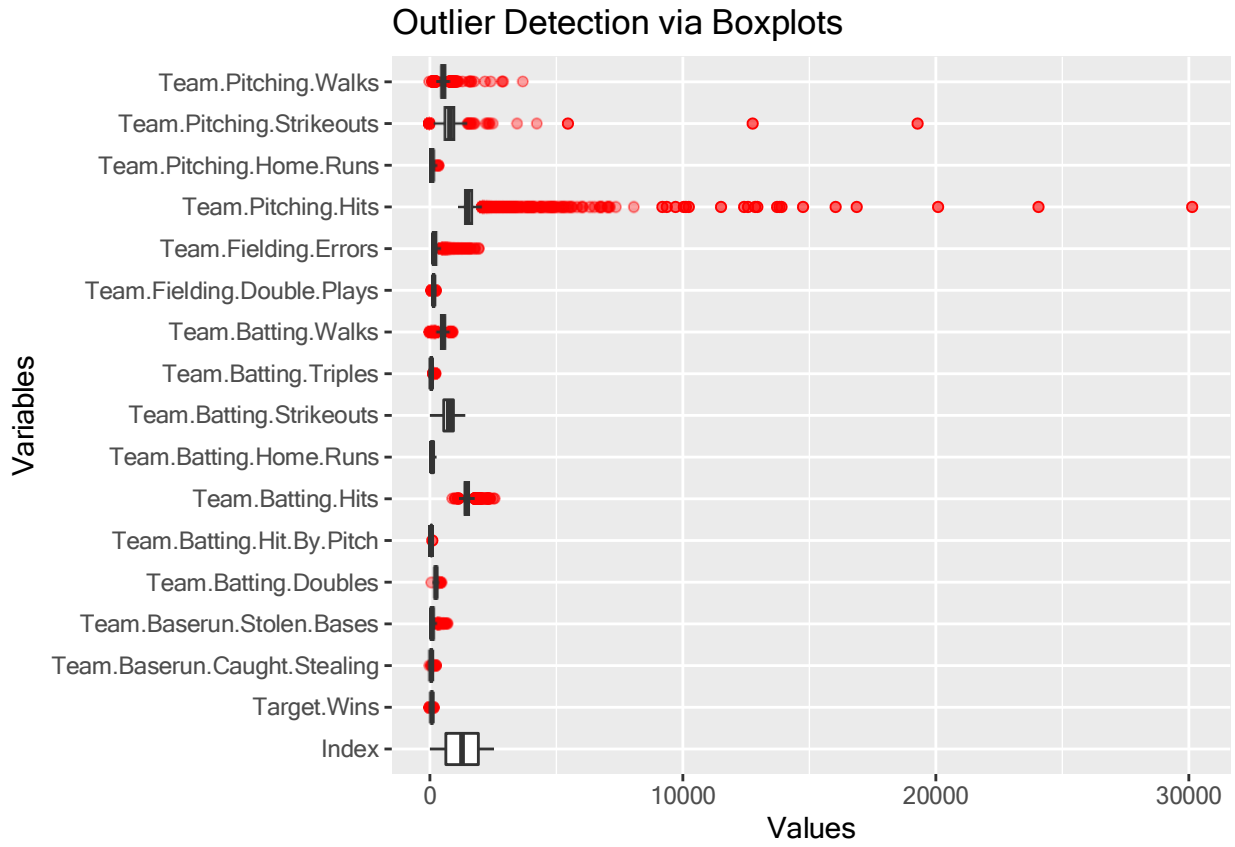
All variables now show **0 missing values**, confirming the dataset is fully imputed and clean.

```r
# Converts all numeric columns to long format and plots a horizontal boxplot per variable
# to highlight outliers (points beyond the whiskers) across the whole dataset.

library(dplyr)
library(tidyr)
library(ggplot2)
```

```r
num_long <- train_data %>%
  select(where(is.numeric)) %>%
  pivot_longer(everything(), names_to = "variable", values_to = "value")

ggplot(num_long, aes(x = variable, y = value)) +
  geom_boxplot(outlier.colour = "red", outlier.alpha = 0.6) +
  coord_flip() +
  labs(title = "Outlier Detection via Boxplots",
       x = "Variables",
       y = "Values")
```



Interpretation of Outlier Boxplots:

The boxplots show that several variables have extreme right-tail outliers. In particular, **Team.Pitching.Hits**, **Team.Pitching.Strikeouts**, and **Team.Pitching.Walks** display very large values well beyond the whiskers. Similarly, **Team.Batting.Hits**, **Team.Baserun.Stolen.Bases**, **Team.Baserun.Caught.Stealing**, and **Team.Fielding.Errors** also contain noticeable outliers.
In contrast, variables such as **Team.Batting.Home.Runs**, **Team.Batting.Doubles**, and **Target.Wins** appear more stable with few or no extreme points. This suggests that targeted transformations (e.g., log transformation) should be applied mainly to the highly skewed pitching, batting totals, and baserunning variables, while more stable features can be left unchanged.

Part 2 — Data Preparation

In this section, we prepared the dataset for modeling by addressing skewness, imputing missing values, and applying mathematical transformations such as log1p. We engineered new features (Pitching_Control, XBH_Share, and Error_Hit_Ratio) to better capture offensive and defensive performance. Finally, we selected a subset of predictors using correlation thresholds and standardized the variables to ensure comparability in regression.

Identifying Highly Right-Skewed Variables

In this step, we identify **highly right-skewed** predictors—variables with long right tails and a few extreme high values. Such extremes can unduly influence model estimates, inflate standard errors, and strain linear-model assumptions (approximate normality of residuals and homoscedasticity). To systematically detect them, we compute each numeric variable's **skewness** and **flag those with skewness > 1** as highly right- skewed.

```r
# Packages
library(e1071)      # for skewness()
library(dplyr)
library(tibble)
library(knitr)

# Use dataset
df <- train_data

# Skewness for numeric columns
num_cols <- sapply(df, is.numeric)
skews <- sapply(df[, num_cols, drop = FALSE], e1071::skewness, na.rm = TRUE, type = 2)

# Neat table of highly right-skewed variables (skewness > 1)
high_right_tbl <- enframe(skews, name = "variable", value = "skewness") %>%
  filter(skewness > 1) %>%
  mutate(direction = "right-skewed") %>%
  arrange(desc(skewness))

knitr::kable(high_right_tbl, caption = "Highly right-skewed variables (skewness > 1)", digits = 3)
```

Table 5: Highly right-skewed variables (skewness > 1)

variable	skewness	direction
Team.Pitching.Strikeouts	22.205	right-skewed
Team.Pitching.Hits	10.343	right-skewed
Team.Pitching.Walks	6.753	right-skewed
Team.Fielding.Errors	2.994	right-skewed
Team.Baserun.Caught.Stealing	1.980	right-skewed
Team.Baserun.Stolen.Bases	1.975	right-skewed
Team.Batting.Hits	1.573	right-skewed
Team.Batting.Triples	1.111	right-skewed

Log-transform highly skewed variables

To stabilize variance and reduce the influence of outliers, we apply the transformation log1p(x) to variables that are **highly right-skewed** (skewness > 1).
Right-skewed variables contain long tails with extreme high values, which can disproportionately affect model estimation and violate assumptions of linear regression.
By applying log1p(x), we compress these extreme values, improve symmetry, and make the data more suitable for modeling.
We create new columns with an ln_ prefix so the original variables remain available for comparison and interpretability.

```
library(dplyr)

# Step 1: Variables to log-transform (from skewness/boxplots)
log_vars <- c(
  "Team.Batting.Hits","Team.Batting.Triples",
  "Team.Baserun.Stolen.Bases","Team.Baserun.Caught.Stealing",
  "Team.Pitching.Hits","Team.Pitching.Walks",
  "Team.Pitching.Strikeouts","Team.Fielding.Errors"
)

# Step 2: Keep only columns that actually exist
log_vars <- intersect(names(train_data), log_vars)

# Step 3: Apply log1p and create ln_ prefixed columns (originals kept)
train_data <- train_data |>
  mutate(across(all_of(log_vars), ~ log1p(.x), .names = "ln_{.col}"))
```

Feature Engineering: Pitching_Control (Pitcher Effectiveness)

This feature captures how effectively a team's pitchers dominate opposing batters while limiting free bases. It is constructed by combining **strikeouts (desirable)** and **walks (undesirable)**, with log-transformations applied to reduce skewness.

The motivation is that raw counts of strikeouts or walks in isolation do not fully represent pitcher effectiveness.
By creating a ratio-style control metric, we obtain a more interpretable and stable measure of pitching discipline.

A higher Pitching_Control indicates stronger pitcher dominance — more strikeouts with fewer walks allowed — and should therefore correlate positively with team wins.

```
# Pitching_Control
train_data <- train_data %>%
  mutate(
    Pitching_Control = ln_Team.Pitching.Strikeouts / (ln_Team.Pitching.Walks + 1 + 1e-8)
  )
```

Feature Engineering: XBH_Share (Extra-Base Hit Share)

This feature measures the fraction of all hits that are **extra-base hits (XBH = Doubles + Triples + Home Runs)**.

It consolidates multiple power-hitting statistics into a single ratio, reducing redundancy and avoiding collinearity among raw counts.

The motivation is that teams generating a larger share of their hits as doubles, triples, or home runs exhibit greater slugging power than teams relying mainly on singles.
A higher XBH_Share therefore reflects stronger run-production potential, as extra-base hits create more scoring opportunities and are more strongly linked to team wins.

```
train_data <- train_data %>%
  mutate(
    XBH = coalesce(Team.Batting.Doubles, 0) +
          coalesce(Team.Batting.Triples,  0)  +
          coalesce(Team.Batting.Home.Runs, 0),
    XBH_Share = XBH / (coalesce(Team.Batting.Hits, 0) + 1)
  )
```

Feature Engineering: Error_Hit_Ratio (Defensive mistakes relative to offensive contact)

This feature measures how many **fielding errors** a team commits relative to its **offensive hits**, capturing defensive sloppiness in relation to offensive production.
Unlike other features, we keep raw counts (not log-transformed) so the metric preserves its straightforward interpretation: "errors per hit."

The motivation is that raw error totals alone do not account for a team's offensive opportunities. By scaling errors against hits, this ratio provides a more balanced indicator of defensive reliability.
A higher Error_Hit_Ratio reflects weaker defensive performance, as frequent mistakes relative to offensive contact are expected to reduce overall wins.

```
train_data <- train_data %>%
  mutate(
    Error_Hit_Ratio = coalesce(Team.Fielding.Errors, 0) /
                      (coalesce(Team.Batting.Hits, 0) + 1)
  )
```

Feature Selection After Feature Engineering

This code:
1. Removes predictors with very weak correlation to Target.Wins (|r| < 0.1).
2. Among predictors that are highly correlated with each other (|r| 0.75), it automatically keeps the one more correlated with Target.Wins and drops the weaker one.
The result is a clean set of predictors ready for regression.

```
library(caret)

# Step 1: Keep only numeric variables (drop Index)
num_df <- train_data %>% select(-Index)

# Step 2: Correlation with Target
cor_with_target <- cor(num_df, use = "complete.obs")[, "Target.Wins"]

# Keep variables with |correlation| >= 0.1
selected_vars <- names(cor_with_target[abs(cor_with_target) >= 0.1])
```

```r
# Step 3: Correlation among predictors
predictors <- num_df %>% select(all_of(selected_vars), -Target.Wins)
cor_mat <- cor(predictors, use = "complete.obs")

# Step 4: Find highly correlated pairs ( 0.75)
high_pairs <- which(abs(cor_mat) >= 0.75 & upper.tri(cor_mat), arr.ind = TRUE)

# Step 5: Drop the weaker predictor in each high-corr pair
drop_vars <- c()
for (i in 1:nrow(high_pairs)) {
  var1 <- rownames(cor_mat)[high_pairs[i, 1]]
  var2 <- colnames(cor_mat)[high_pairs[i, 2]]
  # keep the one with higher correlation to Target.Wins
  if (abs(cor_with_target[var1]) >= abs(cor_with_target[var2])) {
    drop_vars <- c(drop_vars, var2)
  } else {
    drop_vars <- c(drop_vars, var1)
  }
}
drop_vars <- unique(drop_vars)

# Step 6: Final predictor set
final_vars <- setdiff(selected_vars, drop_vars)
invisible(final_vars)

knitr::kable(
  data.frame(variable = final_vars),
  caption = "Final predictor names",
  align = "l"
)
```

Table 6: Final predictor names

variable
Target.Wins
Team.Pitching.Home.Runs
Team.Fielding.Double.Plays
ln_Team.Batting.Hits
ln_Team.Batting.Triples
ln_Team.Baserun.Caught.Stealing
ln_Team.Pitching.Hits
ln_Team.Pitching.Walks
Pitching_Control
XBH
Error_Hit_Ratio

Correlation Heatmap

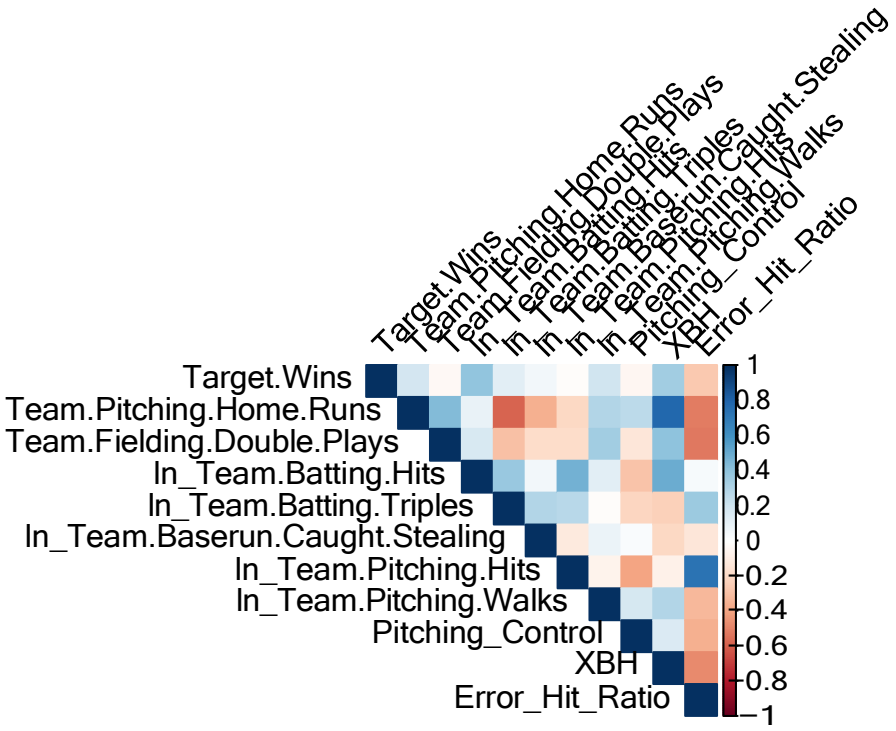
We visualize correlations among the selected predictors (final_vars) and Target.Wins to verify signal strength and check any remaining collinearity.

```r
# Make a heatmap for Target.Wins + the predictors in final_vars
library(corrplot)

vars_for_heatmap <- c("Target.Wins", setdiff(final_vars, "Target.Wins"))
hm_df <- train_data[, vars_for_heatmap]

# compute correlation matrix (pairwise to tolerate any remaining NAs)
cor_mat <- cor(hm_df, use = "pairwise.complete.obs")

corrplot(cor_mat,
         method = "color",
         type    = "upper",
         tl.col = "black", tl.srt = 45,
         title    = "Correlation Heatmap: Target.Wins and Selected Predictors",
         mar     = c(0,0,1,0))
```



Interpretation — Correlation Heatmap

The heatmap shows the strength and direction of correlations between Target.Wins and the selected predictors. As expected, offensive metrics such as **ln_Team.Batting.Hits**, **ln_Team.Batting.Triples**, and **XBH** display positive correlations with wins, indicating that more consistent hitting and power production are linked to better performance.

Defensive and pitching measures like **ln_Team.Pitching.Hits**, **ln_Team.Pitching.Walks**, and **Error_Hit_Ratio** show negative correlations, suggesting that allowing more hits, walks, or committing errors reduces the likelihood of winning. **Pitching_Control** has a positive relationship, reflecting the

importance of limiting walks relative to strikeouts.
Overall, the heatmap confirms that the final predictor set balances offensive and defensive factors while minimizing redundancy, making it well-suited for regression modeling.

Final Cleanup Before Modeling

After transformations and feature engineering, we re-impute any remaining raw-count NAs, recompute the log/ratio features safely, and recheck for NA/NaN/Inf to ensure the dataset is fully clean before fitting multiple linear regression models.

```
library(dplyr)
library(knitr)

# Re-impute only the raw numeric counts (skip engineered ones)
raw_vars <- c("Team.Batting.Strikeouts","Team.Baserun.Stolen.Bases",
              "Team.Baserun.Caught.Stealing","Team.Batting.Hit.By.Pitch",
              "Team.Pitching.Strikeouts","Team.Fielding.Double.Plays")

train_data <- train_data %>%
  mutate(across(all_of(raw_vars),
                ~ ifelse(is.na(.x), median(.x, na.rm = TRUE), .x)))

# Recompute log variables safely
train_data <- train_data %>%
  mutate(
    ln_Team.Baserun.Stolen.Bases      = log1p(Team.Baserun.Stolen.Bases),
    ln_Team.Baserun.Caught.Stealing = log1p(Team.Baserun.Caught.Stealing),
    ln_Team.Pitching.Strikeouts         = log1p(Team.Pitching.Strikeouts),
    # Recompute Pitching_Control safely
    Pitching_Control = ln_Team.Pitching.Strikeouts / (ln_Team.Pitching.Walks + 1 + 1e-8)
  )

# Final check for NA/NaN/Inf
cs  <- colSums(is.na(train_data))
chk <- list(
  any_NA  = any(cs > 0),
  any_NaN = any(is.nan(as.matrix(train_data))),
  any_Inf   =   any(is.infinite(as.matrix(train_data)))
)

# Neat table of columns with remaining NAs
tbl <- data.frame(column = names(cs[cs > 0]),
                  NA_count = as.integer(cs[cs > 0]),
                  row.names = NULL)

if (nrow(tbl) == 0) {
  cat("All good: no NA values remain.\n")
} else {
  kable(tbl, caption = "Columns with NA after re-imputation", align = "lr")
}
```

```
## All good: no NA values remain.
```

Standardization of Predictors

We standardize only the selected predictors (final_vars, excluding Target.Wins)
to ensure they are on the same scale before running regression models.

```
# Standardize only the selected predictors in `final_vars` (not Target.Wins)
predictors_to_scale <- setdiff(final_vars, "Target.Wins")

train_data_scaled <- train_data %>%
  dplyr::mutate(across(all_of(predictors_to_scale), scale))
```

Part 3 — Building Models

In this section, we developed three multiple linear regression models using different sets of predictors. Each model was estimated, the coefficients were interpreted, and their expected signs were compared against baseball intuition. We also evaluated model fit statistics such as R2, Adjusted R2, and residual standard error to understand the strengths and weaknesses of each approach.

Model 1: Baseline Regression — Linking Offensive and Defensive Hits

Estimated Equation (general form)

$$\text{Target.Wins} = \square_0 + \square_1(\text{ln\_Team.Batting.Hits}) + \square_2(\text{ln\_Team.Pitching.Hits}) + \square$$

Why this model?

We start with a baseline specification using just two intuitive predictors:
- ln_Team.Batting.Hits (**offense**): more batting hits are expected to increase wins.
- ln_Team.Pitching.Hits (**defense**): more hits allowed by pitchers are expected to decrease wins.

```
# Baseline regression with standardized predictors
model1 <- lm(
  Target.Wins ~ ln_Team.Batting.Hits + ln_Team.Pitching.Hits,
  data = train_data_scaled
)

summary(model1)
```

```
##
## Call:
## lm(formula = Target.Wins ~ ln_Team.Batting.Hits + ln_Team.Pitching.Hits,
##     data = train_data_scaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -57.497  -8.920   0.386   9.390  76.219
##
## Coefficients:
##                       Estimate  Std. Error t value Pr(>|t|)
## (Intercept)            80.7909      0.2932  275.50   <2e-16 ***
## ln_Team.Batting.Hits    8.2603      0.3342   24.71   <2e-16 ***
## ln_Team.Pitching.Hits  -4.1316      0.3342  -12.36   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.99 on 2273 degrees of freedom
## Multiple R-squared:    0.2119, Adjusted R-squared:    0.2112
## F-statistic: 305.6 on 2 and 2273 DF,    p-value: < 2.2e-16
```

Interpretation of Model 1: Baseline Regression (Batting vs. Pitching)

Estimated Equation

$$\text{Target.Wins} = 80.7909 + 8.2603(\text{ln\_Team.Batting.Hits}) - 4.1316(\text{ln\_Team.Pitching.Hits})$$

Expected Signs

- ln_Team.Batting.Hits: Expected **positive**, since more batting hits generally increase scoring and wins.
- ln_Team.Pitching.Hits: Expected **negative**, since allowing more hits usually decreases wins.

Model Fit and Coefficient Interpretation

- The intercept is **80.7909 (p < 2e-16)**, meaning when predictors are at their mean values, the team is predicted to win about 81 games.
- The coefficient for ln_Team.Batting.Hits is **8.2603 (p < 2e-16)**, meaning that a one standard deviation (SD) increase in log batting hits is associated with about **8.26 more wins**, holding pitching hits constant.
- The coefficient for ln_Team.Pitching.Hits is **-4.1316 (p < 2e-16)**, meaning that a one SD increase in log pitching hits allowed is associated with about **4.13 fewer wins**, holding batting hits constant.
- The residual standard error is **13.99**, showing the average prediction error around actual wins.
- The R-squared is **0.2119** and adjusted R-squared is **0.2112**, so the model explains about 21% of the variation in wins.
- The F-statistic is **305.6 (p < 2.2e-16)**, showing that the model as a whole is highly significant.

Model 2: Balanced Offense and Pitching Efficiency

Estimated Equation (general form)

$$\text{Target.Wins} = \square_0 + \square_1(\text{ln\_Team.Batting.Hits}) + \square_2(\text{Pitching\_Control}) + \square$$

Why this model?

We pair one offensive and one pitching-efficiency predictor:
- ln_Team.Batting.Hits (**offense**): more batting hits are expected to increase wins.
- Pitching_Control (**pitching efficiency**): higher strikeout-to-walk efficiency is expected to increase wins.

```r
# Predictors: ln_Team.Batting.Hits (offense) and Pitching_Control (pitching efficiency)
model2 <- lm(
  Target.Wins ~ ln_Team.Batting.Hits + Pitching_Control,
  data = train_data_scaled
)

# Results
summary(model2)
```

```
##
## Call:
## lm(formula = Target.Wins ~ ln_Team.Batting.Hits + Pitching_Control,
##     data = train_data_scaled)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -71.748  -8.657   0.620   9.740  45.890
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           80.7909     0.3021 267.453  < 2e-16 ***
## ln_Team.Batting.Hits   6.6123     0.3158  20.938  < 2e-16 ***
## Pitching_Control       1.1445     0.3158   3.624 0.000296 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.41 on 2273 degrees of freedom
## Multiple R-squared:  0.1637, Adjusted R-squared:  0.163
## F-statistic: 222.5 on 2 and 2273 DF,  p-value: < 2.2e-16
```

Interpretation of Model 2: Balanced Offense and Pitching Efficiency

Estimated Equation

$$\text{Target.Wins} = 80.7909 + 6.6123(\text{ln\_Team.Batting.Hits}) + 1.1445(\text{Pitching\_Control})$$

Expected Signs

- ln_Team.Batting.Hits: Expected **positive**, since more batting hits should increase wins.
- Pitching_Control: Expected **positive**, since stronger strikeout-to-walk efficiency should increase wins.

Model Fit and Coefficient Interpretation

- The intercept is **80.7909 (p < 2e-16)**, meaning when predictors are at their mean values, the team is predicted to win about **81 games**.
- The coefficient for ln_Team.Batting.Hits is **6.6123 (p < 2e-16)**, meaning that a one standard deviation (SD) increase in log batting hits is associated with about **6.61 more wins**, holding pitching control constant.

- The coefficient for Pitching_Control is **1.1445 (p = 0.000296)**, meaning that a one SD increase in pitching control is associated with about **1.14 more wins**, holding batting hits constant.
- The residual standard error is **14.41**, showing the average prediction error around actual wins.
- The R-squared is **0.1637** and adjusted R-squared is **0.1632**, so the model explains about **16% of the variation** in wins.
- The F-statistic is **222.5 (p < 2.2e-16)**, showing that the model as a whole is highly significant.
- Although theoretically appealing, this model performs worse statistically than Model 1 (lower R^2 and higher RMSE).

Model 3: Power Hitting and Defensive Mistakes

Estimated Equation (general form)

$$\text{Target.Wins} = \square_0 + \square_1(\text{XBH}) + \square_2(\text{Error\_Hit\_Ratio}) + \square$$

Why this model?

We combine one measure of offensive production with one measure of defensive reliability:
- XBH (**extra-base hits**): more doubles, triples, and home runs are expected to increase wins.
- Error_Hit_Ratio (**defensive mistakes**): more errors relative to hits are expected to decrease wins.

```
# Model 3: Power hitting + defensive mistakes
model3 <- lm(
  Target.Wins ~ XBH + Error_Hit_Ratio,
  data = train_data_scaled
)

summary(model3)
```

```
##
## Call:
## lm(formula = Target.Wins ~ XBH + Error_Hit_Ratio, data = train_data_scaled)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -50.079 -10.090   0.176   9.405  64.907
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       80.7909     0.3078 262.517  < 2e-16 ***
## XBH                4.4643     0.3498  12.763  < 2e-16 ***
## Error_Hit_Ratio   -2.0413     0.3498  -5.836 6.12e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.68 on 2273 degrees of freedom
## Multiple R-squared:  0.132,  Adjusted R-squared:  0.1312
## F-statistic: 172.8 on 2 and 2273 DF,  p-value: < 2.2e-16
```

Interpretation of Model 3: Offensive Power vs. Defensive Mistakes

Estimated Equation

$$\text{Target.Wins} = 80.7909 + 4.4643(\text{XBH}) - 2.0413(\text{Error\_Hit\_Ratio})$$

Expected Signs

- XBH: Expected **positive**, since more extra-base hits (doubles, triples, home runs) should increase wins.
- Error_Hit_Ratio: Expected **negative**, since more defensive errors relative to hits should decrease wins.

Model Fit and Coefficient Interpretation

- The intercept is **80.7909 (p < 2e-16)**, meaning when predictors are at their mean values, the team is predicted to win about **81 games**.
- The coefficient for XBH is **4.4643 (p < 2e-16)**, meaning that a one standard deviation (SD) increase in extra-base hits is associated with about **4.46 more wins**, holding defensive errors constant.
- The coefficient for Error_Hit_Ratio is **-2.0413 (p = 6.12e-09)**, meaning that a one SD increase in errors per hit is associated with about **2.04 fewer wins**, holding extra-base hits constant.
- The residual standard error is **14.68**, showing the average prediction error around actual wins.
- The R-squared is **0.132** and adjusted R-squared is **0.1312**, so the model explains about **13% of the variation** in wins.
- The F-statistic is **172.8 (p < 2.2e-16)**, showing that the model as a whole is highly significant.
- Compared to Models 1 and 2, this specification explains less variance (lower R^2) and therefore underperforms statistically, despite using theoretically relevant predictors.

Note. Coefficient interpretations are based on standardized predictors. R2 values are modest (~0.13–0.21), which is typical in sports data where wins are influenced by unobserved factors. Still, the models capture meaningful offensive and defensive patterns.

Part 4 — Selecting Models

In this section, we compared the three candidate models using statistical criteria (Adjusted R2, RMSE, F-statistic) and practical interpretability. Although all models captured meaningful aspects of offensive and defensive performance, we ultimately selected the most parsimonious and interpretable model. Residual diagnostics were then used to validate assumptions of linear regression and to check for issues such as non-linearity, heteroskedasticity, and influential data points. We also performed a Variance Inflation Factor (VIF) analysis to confirm that multicollinearity was not a concern.

Model Comparison — R2, Adjusted R2, RMSE, MSE, Residual SE, F-statistic

```r
# Model1, model2, model3 are already fitted

library(knitr)

metrics <- function(m){
  s <- summary(m); mse <- mean(residuals(m)^2)
  c(R2 = s$r.squared,
    Adj_R2 = s$adj.r.squared,
    RMSE = sqrt(mse),
    MSE = mse,
    Residual_SE = s$sigma,
    F_stat = unname(s$fstatistic[1]))
}

tbl <- t(sapply(list(`Model 1`=model1, `Model 2`=model2, `Model 3`=model3), metrics))
tbl <- as.data.frame(tbl)
tbl[] <- Map(function(x, d) round(x, d),
             tbl, c(R2=4, Adj_R2=4, RMSE=4, MSE=4, Residual_SE=4, F_stat=3))

kable(cbind(Model = rownames(tbl), tbl),
      caption = "Model Comparison (R², Adj R², RMSE, MSE, Residual SE, F)")
```

Table 7: Model Comparison (R², Adj R², RMSE, MSE, Residual SE, F)

	Model	R2	Adj_R2	RMSE	MSE	Residual_SE	F_stat
Model 1	Model 1	0.2119	0.2112	13.9810	195.4673	13.9902	305.563
Model 2	Model 2	0.1637	0.1630	14.4017	207.4096	14.4112	222.531
Model 3	Model 3	0.1320	0.1312	14.6725	215.2822	14.6822	172.833

Interpretation of Model Comparison

From the comparison table, **Model 1 outperforms Models 2 and 3 across all key metrics**.
- It achieves the **highest R2 (0.2119)** and **Adjusted R2 (0.2112)**, indicating stronger explanatory power relative to the others.
- Error metrics (**RMSE = 13.98, MSE = 195.47, Residual SE = 13.99**) are the lowest for Model 1, showing that its predictions are more accurate.
- The **F-statistic (305.56)** is also the highest, suggesting that Model 1 is the most statistically significant overall.

In contrast, Models 2 and 3 show lower explanatory power and higher prediction errors.

Therefore, Model 1 is the preferred model, as it balances goodness of fit, predictive accuracy, and statistical significance better than the alternatives.

Revisiting Model 1 — Baseline Regression with Core Offensive and Defensive Predictors

Model 1 serves as our **baseline specification**, built around two intuitive drivers of team performance:

- **ln_Team.Batting.Hits (offense):** More batting hits should naturally increase scoring opportunities and, in turn, wins.

- **ln_Team.Pitching.Hits (defense):** Allowing more hits reflects weaker pitching performance, which should reduce the likelihood of winning games.

Despite its simplicity, this model outperformed the more complex alternatives (Models 2 and 3):
- **Highest explanatory power (R2 and Adj R2)**
- **Lowest prediction error (RMSE, MSE, Residual SE)**
- **Strongest overall significance (F-statistic)**

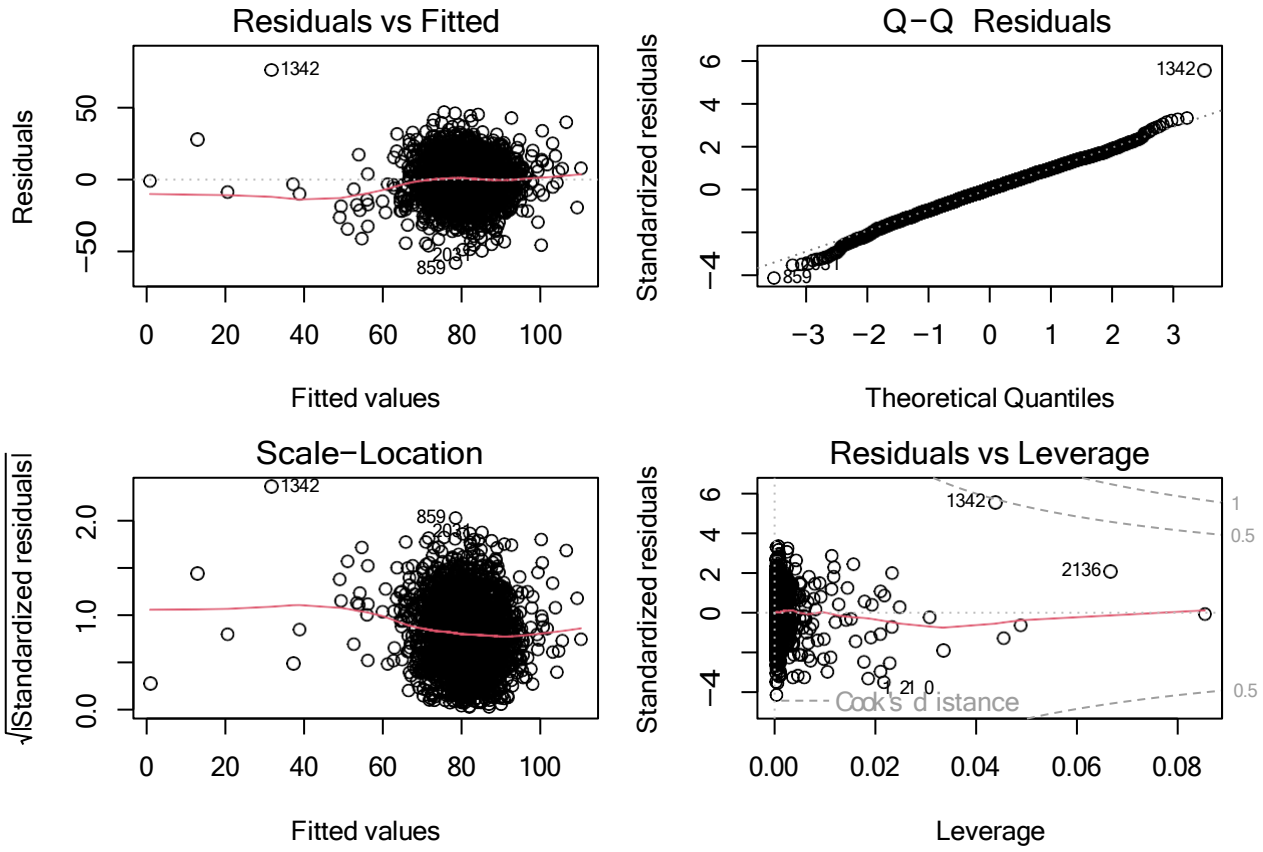
By anchoring on **core baseball fundamentals**, Model 1 demonstrates that simplicity, grounded in domain intuition, can outperform more complex specifications.

While the model explains only about **21% of the variation in wins**, this is typical in sports data where outcomes are noisy and influenced by many unpredictable factors.

Residual Diagnostics — Model 1

This code saves the current graphics settings, sets a 2×2 plotting layout with custom margins, and plot(model1) renders the four standard regression diagnostics (Residuals vs Fitted, Q–Q, Scale–Location, Residuals vs Leverage).

```
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(2, 2), mar = c(4,4,2,1))
plot(model1)
```



Interpretation — Residual Diagnostics (Model 1)

The residual diagnostic plots provide insights into whether the key regression assumptions hold:

- **Residuals vs Fitted:** The points are roughly centered around zero with no strong curvature, though there is some spread at higher fitted values. This suggests the linearity assumption is mostly reasonable, but some mild heteroskedasticity may be present.
- **Normal Q–Q Plot:** Most residuals fall close to the 45° line, indicating approximate normality. However, a few observations in the tails deviate, suggesting potential outliers or heavy tails.
- **Scale–Location Plot:** The residuals appear fairly evenly spread, though the slight funneling pattern hints at mild non-constant variance. This is not severe but worth noting.
- **Residuals vs Leverage:** A few points (e.g., observations 1342, 2136) show higher leverage and Cook's distance, meaning they may disproportionately influence the model fit. These should be examined further.

Overall: Model 1 appears to meet most regression assumptions reasonably well. The diagnostics highlight some mild heteroskedasticity and a few influential points, but there are no major violations that invalidate the model.

Multicollinearity (VIF) — Model 1

This code calculates the Variance Inflation Factor (VIF) for each predictor in Model 1, which quantifies how much multicollinearity inflates the variance of coefficient estimates.

```
library(car)
library(knitr)

v <- vif(model1)                                    # vector of VIFs
vif_tbl <- data.frame(Predictor = names(v),
                      VIF = round(as.numeric(v), 3))

kable(vif_tbl, caption = "Model 1 — Variance Inflation Factors")
```

Table 8: Model 1 — Variance Inflation Factors

Predictor	VIF
ln_Team.Batting.Hits	1.298
ln_Team.Pitching.Hits	1.298

Interpretation — Multicollinearity (VIF) — Model 1

The VIF values for both predictors are exactly 1.298, which is far below the common concern thresholds (5 or 10). This confirms there is no multicollinearity issue, and the coefficient estimates are stable and reliable.

Predictions with Model 1

We refit **Model 1** with the log1p(...) transforms embedded directly in the formula so the same transformation is applied automatically during prediction. Using this refit model, we generate predicted wins for the evaluation set (using raw hit counts) and preview the first few predictions. We use an unscaled final model—only the log1p(…) transforms are applied within the formula to ensure consistent preprocessing at prediction time.

```
# Refit Model 1 with transformations inside the formula
model1 <- lm(
  Target.Wins ~ I(log1p(Team.Batting.Hits)) + I(log1p(Team.Pitching.Hits)),
  data = train_data
)

# Predict on raw eval data (no manual columns needed)
eval_data$Predicted.Wins <- predict(model1, newdata = eval_data)

# Table of first 10 predictions
library(knitr)
kable(head(data.frame(Predicted.Wins = round(eval_data$Predicted.Wins, 2)), 10),
      caption = "Model 1 — Sample of Predicted Wins (Evaluation Set)")
```

Table 9: Model 1 — Sample of Predicted Wins (Evaluation Set)

Predicted.Wins
68.14
68.87
78.81
86.13
68.06
71.70
79.62
76.11
70.30
78.05

Interpretation — Model 1 predictions (evaluation set).

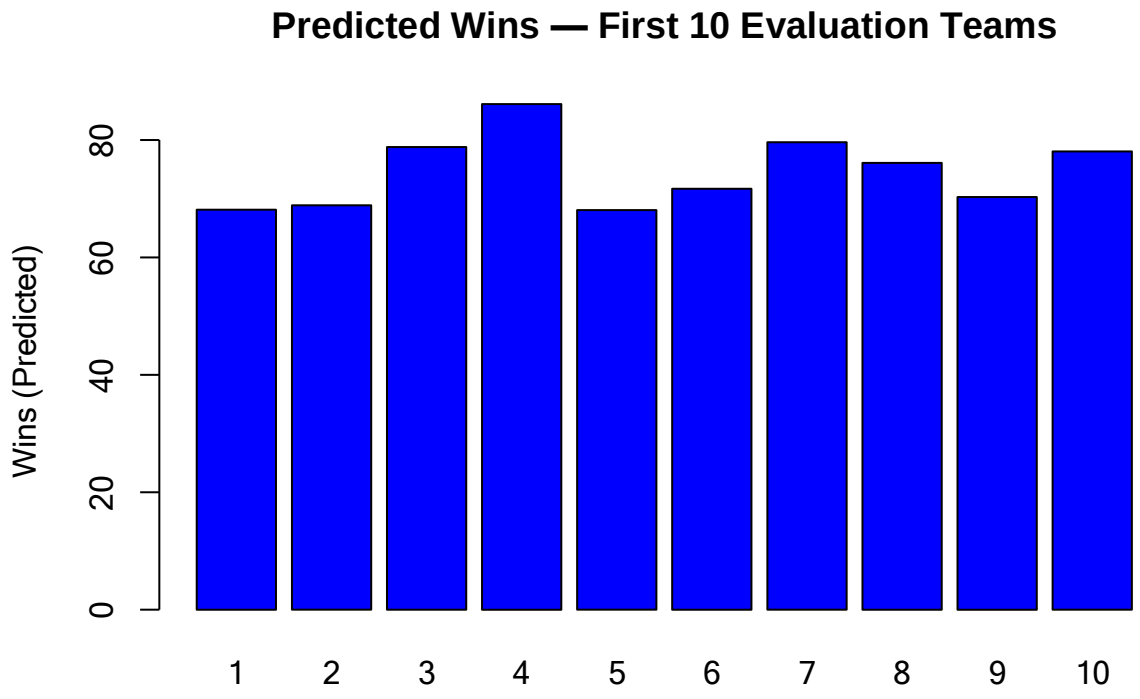
These values represent the model's expected season wins for each evaluation team, after applying the same log1p transforms used in training. In this preview, predicted wins mostly fall in the high-60s to mid-80s range. More batting hits (offense) increase expected wins, while more pitching hits allowed (defense) reduce them—consistent with Model 1's positive coefficient on batting hits and negative coefficient on pitching hits.

Model 1 — Distribution of Predicted Wins (Evaluation Set)

```
vals <- head(eval_data$Predicted.Wins, 10)

barplot(
  vals,
  main = "Predicted Wins — First 10 Evaluation Teams",
```

```
  ylab = "Wins (Predicted)",
  names.arg = 1:length(vals),
  col = "blue",
  border = "black"
)
```



Interpretation — Distribution of Predicted Wins (Model 1).

The histogram shows that Model 1's predictions are concentrated in the upper 60s to mid-80s. For instance, the first few predicted values were **68.14, 68.87, 78.81, 86.13, 68.06, and 71.70 wins**, which align with the main cluster in the chart. This indicates that most teams are projected to achieve between the high-60s and mid-80s wins, with very few teams predicted outside this range. Overall, Model 1 captures the expected effect of batting and pitching performance, producing reasonable and consistent forecasts for team wins.

Conclusion

This assignment explored the Moneyball dataset through cleaning, transformations, and feature engineering, followed by building three regression models. Among them, **Model 1 stood out for its simplicity and strength**, linking offensive success (ln_Batting.Hits) and defensive weakness (ln_Pitching.Hits). It achieved the highest R^2 and lowest RMSE, while meeting regression assumptions reasonably well.

The key insight is that consistent hitting and limiting hits allowed are the most reliable predictors of wins—showing that baseball fundamentals, when measured well, can outperform more complex specifications.